Apple Stock Price Data Warehouse

-By Anushil Timsina

Objective

- Create a data warehouse to store data and complete the ETL pipeline.

  The project is to collect data on the stock of Apple company which has the symbol “APPL” and is traded on the S&P 500 market. I collect data from 2013 till the present time and will also include automation to include data from the future. All of the collected data comes through an ETL pipeline and is stored in a Data Warehouse. The data will be useful to analyze the performance of the stock of Apple company.

Requirements

- Python Programming Language
- Kaggle
- Data Scraping
- yfinance API
- Microsoft SQL Server Management Studio
- Talend Open Studio
- Different AWS Cloud Services

Theory

ETL Pipeline

An ETL (Extract, Transform, Load) pipeline is a data integration process that involves extracting data from various sources, transforming it into a format that is suitable for analysis, and loading it into a target data store.
Here's an overview of the ETL pipeline components and their functions:

1. **Data Sources**: The data sources are the systems or applications from which data is extracted. Data can be extracted from a variety of sources, such as databases, flat files, APIs, or web scraping. The data sources can be on-premises or in the cloud.
2. **Extractors**: The extractors are the components that extract data from the data sources. The extractors can be built in-house or can be third-party tools that are specialized for extracting data from specific data sources. Examples of extractors include Talend's "tFileInputDelimited" component for reading CSV files and AWS Glue for extracting data from databases.

3. **Data Storage**: The data storage is the system or application where the extracted data is temporarily stored before it is transformed. The data storage can be a file system, a database, or a distributed storage system like Hadoop or AWS S3.
4. **Transformers**: The transformers are the components that transform the data into a format that is suitable for analysis. The transformers can perform a variety of operations on the data, such as cleaning, filtering, aggregating, or joining. Examples of transformers include Talend's "tMap" component for mapping and transforming data, and Apache Spark for performing complex data transformations.
5. **Data Quality Checkers**: The data quality checkers are the components that check the quality of the transformed data to ensure that it is accurate and complete. The data quality checkers can perform various checks, such as checking for missing values, duplicates, or inconsistencies. Examples of data quality checkers include Talend's "tAssert" component and AWS Glue's "DynamicFrameValidator" class.
6. **Loaders**: The loaders are the components that load the transformed data into the target data store. The target data store can be a database, a data warehouse, or a data lake. The loaders can perform a variety of operations on the data, such as inserting, updating, or deleting records. Examples of loaders include Talend's "tMysqlOutput" component for loading data into MySQL databases and AWS Glue's "DynamicFrameWriter" class for writing data to S3.
7. **Orchestration**: The orchestration is the process of scheduling and coordinating the ETL pipeline components to ensure that the pipeline runs smoothly and efficiently. The orchestration can be performed by a variety of tools, such as Apache Airflow, AWS Step Functions, or Kubernetes.

By using these ETL pipeline components, we can extract data from various sources, transform it into a format that is suitable for analysis, and load it into a target data store. This allows us to perform analytics and gain insights from our data, which can help us make informed business decisions.

Data Warehouse

A data warehouse is a large, centralized repository of data that is specifically designed for analytical processing and reporting. A data warehouse is typically used to store historical data from various sources, such as databases, flat files, or external sources, and to consolidate and integrate the data to provide a single, unified view of the data.

Here are some of the key characteristics and components of a data warehouse:

1. **Data Integration**: A data warehouse integrates data from multiple sources, such as transactional databases, operational systems, and external data sources. The data is transformed and cleansed to ensure consistency and accuracy across different data sources.
2. **Data Modeling**: A data warehouse uses a specific type of data modeling called dimensional modeling, which organizes data into a series of tables that represent facts

(e.g. sales) and dimensions (e.g. time, product, location). This allows for easy querying and analysis of data.
3. **Data Storage**: A data warehouse stores data in a structured, optimized format that is designed for analytical processing. The data is stored in a way that enables fast query performance and efficient storage utilization.
4. **Query and Analysis**: A data warehouse enables users to query and analyze data using a variety of tools and techniques, such as SQL, OLAP (Online Analytical Processing), and data mining. Users can perform complex queries and analyses to gain insights into their data and make informed business decisions.
5. **Reporting and Visualization**: A data warehouse enables users to generate reports and visualizations of their data using tools such as dashboards, charts, and graphs. These reports and visualizations provide a way to communicate insights and findings to stakeholders.
6. **Metadata Management**: A data warehouse includes metadata management, which is the process of managing the information about the data in the warehouse. This includes information such as data definitions, data lineage, and data quality metrics.
7. **Security and Access Control**: A data warehouse includes security and access control mechanisms to ensure that only authorized users can access the data. This includes measures such as authentication, authorization, and encryption.

By using a data warehouse, organizations can gain a better understanding of their data and make informed decisions based on that data. The data warehouse provides a central repository of data optimized for analytical processing, enabling users to query and analyze large volumes of data quickly and efficiently.

Initial Steps

1) Data Source Definition

As a data warehouse contains diverse sources of data, the project has 3 distinct data sources.

- The first one is a CSV file taken from Kaggle. It contains data from the date 2013-02-08 till 2018-02-7.

- The second one is data scraped from the Yahoo finance website. It contains data from date 2018-01-01 to 2023-05-12.

- The third source is yfinance API which will be taken every month and updated to the total dataset.

2) Data Extraction

- The CSV from Kaggle is downloaded.

AAP_kaggle.csv

```
date,open,high,low,close,volume,Name
2013-02-08,78.34,79.72,78.01,78.9,1298137,AAP
2013-02-11,78.65,78.91,77.23,78.39,758016,AAP
2013-02-12,78.39,78.63,77.5132,78.6,876859,AAP
2013-02-13,78.9,79.13,77.85,78.97,1038574,AAP
2013-02-14,78.66,79.72,78.585,78.84,1005376,AAP
2013-02-15,78.83,79.18,77.93,79.0,1247063,AAP
2013-02-19,79.12,81.44,78.57,80.72,1730690,AAP
2013-02-20,80.42,80.94,79.39,79.5,648693,AAP
2013-02-21,79.26,80.05,78.56,79.06,915160,AAP
2013-02-22,79.2,79.46,78.63,79.21,538188,AAP
```

- Data is scraped and saved as CSV from the Yahoo finance website.

AAPL_scraped.csv

```
Date,Open,High,Low,Close*,Adj. close**,Volume
"Jan 05, 2018",43.36,43.84,43.26,43.75,41.97,"94,640,000"
"Jan 04, 2018",43.13,43.37,43.02,43.26,41.49,"89,738,400"
"Jan 03, 2018",43.13,43.64,42.99,43.06,41.30,"118,071,600"
"Jan 02, 2018",42.54,43.08,42.31,43.06,41.31,"102,223,600"
"Jan 12, 2018",44.04,44.34,43.91,44.27,42.47,"101,672,400"
"Jan 11, 2018",43.65,43.87,43.62,43.82,42.03,"74,670,800"
"Jan 10, 2018",43.29,43.58,43.25,43.57,41.80,"95,839,600"
"Jan 09, 2018",43.64,43.76,43.35,43.58,41.81,"86,336,000"
"Jan 08, 2018",43.59,43.90,43.48,43.59,41.81,"82,271,200"
```

- Data taken from the API is also saved in a CSV format.

latest.csv

```
date,open,high,low,close,volume
2023-04-13,161.40674980193992,165.57098806709064,161.19703316457137,165.33131408691406,68445600
2023-04-14,164.36265753254696,166.0902789530472,163.5937320609274,164.9818115234375,49386500
2023-04-17,164.86196400554493,165.16155267486212,163.80343057997516,165.00177001953125,41516200
2023-04-18,165.87058246245033,167.17877060262967,165.42119182938217,166.2400665283203,49923000
2023-04-19,165.5709938538562,167.92773474263433,165.31134322349524,167.39846801757812,47720200
```

Steps

Without Using AWS

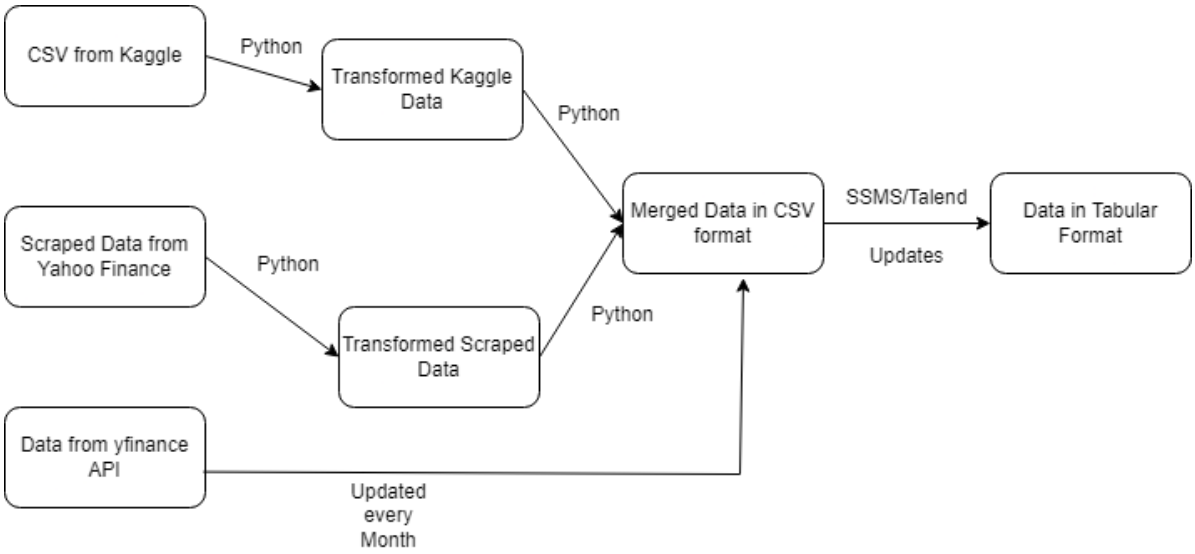


Fig: Architecture of the project without using AWS

1) Data Transformation

As seen in the above snippets, the data collected are not in the required form.
For the data taken from Kaggle, here is the info.

```
kaggle.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1259 entries, 0 to 1258
Data columns (total 7 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   date    1259 non-null   object
 1   open    1259 non-null   float64
 2   high    1259 non-null   float64
 3   low     1259 non-null   float64
 4   close   1259 non-null   float64
 5   volume  1259 non-null   int64
 6   Name    1259 non-null   object
dtypes: float64(4), int64(1), object(2)
memory usage: 69.0+ KB
```

The Name column is not required as all the data is of Apple stock. So, the Name column is removed.
The date column is of data type object. Better, convert it to datetime format.
As there are no null values, no need to impute data or remove a row.

For the scraped data, here is the info.

```
scraped.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1350 entries, 0 to 1349
Data columns (total 7 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   Date          1350 non-null   object
 1   Open          1350 non-null   float64
 2   High          1350 non-null   float64
 3   Low           1350 non-null   float64
 4   Close*        1350 non-null   float64
 5   Adj. close**  1350 non-null   float64
 6   Volume        1350 non-null   object
dtypes: float64(5), object(2)
memory usage: 74.0+ KB
```

There are 2 columns for close. One is Close and another is adjusted close. As adjusted close is considered more accurate, the Close column is dropped.
The date column is also converted to datetime data type.
The column names are also changed so that it matches the column name of the Kaggle dataset.
Looking at a few of the rows of the scraped data,

```
scraped.head()
```

	date	open	high	low	close	volume
0	2018-01-05	43.36	43.84	43.26	41.97	94,640,000
1	2018-01-04	43.13	43.37	43.02	41.49	89,738,400
2	2018-01-03	43.13	43.64	42.99	41.30	118,071,600
3	2018-01-02	42.54	43.08	42.31	41.31	102,223,600
4	2018-01-12	44.04	44.34	43.91	42.47	101,672,400

The volume has ‘,’ so is in the object datatype. Replace ‘,’ by ‘’ and convert the volume column to int64 data type.
So, the final info of both datasets is as follows.

```
kaggle.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1259 entries, 0 to 1258
Data columns (total 6 columns):
 #   Column  Non-Null Count  Dtype         
---  ------  --------------  -----         
 0   date    1259 non-null   datetime64[ns]
 1   open    1259 non-null   float64       
 2   high    1259 non-null   float64       
 3   low     1259 non-null   float64       
 4   close   1259 non-null   float64       
 5   volume  1259 non-null   int64         
dtypes: datetime64[ns](1), float64(4), int64(1)
memory usage: 59.1 KB
```

```
scraped.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1350 entries, 0 to 1349
Data columns (total 6 columns):
 #   Column  Non-Null Count  Dtype         
---  ------  --------------  -----         
 0   date    1350 non-null   datetime64[ns]
 1   open    1350 non-null   float64       
 2   high    1350 non-null   float64       
 3   low     1350 non-null   float64       
 4   close   1350 non-null   float64       
 5   volume  1350 non-null   int64         
dtypes: datetime64[ns](1), float64(4), int64(1)
memory usage: 63.4 KB
```

The transformed data is saved in CSV format for further use.

2) Combine the Transformed Data

The datasets are identical. So, combining them will not be a problem. Info of the combined dataset.

```
merged.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2609 entries, 0 to 2608
Data columns (total 6 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   date    2609 non-null   object 
 1   open    2609 non-null   float64
 2   high    2609 non-null   float64
 3   low     2609 non-null   float64
 4   close   2609 non-null   float64
 5   volume  2609 non-null   int64  
dtypes: float64(4), int64(1), object(1)
memory usage: 122.4+ KB
```

But, there are some duplicates as we have data whole of January and the first week of February on both datasets.

```
# check for duplicates
merged.date.duplicated().sum()
```

```
26
```

There are 26 duplicated values. Remove them.
Then finally download the merged dataset as a CSV.

3) Use yfinance API to get the latest data

As a Data Warehouse is mostly stable and the data is not added in real-time, in the project, the data is added every month. To do this, the yfinance API is used.

```
stock = yf.Ticker("AAPL")
stock.info
```

```
{'address1': 'One Apple Park Way',
 'city': 'Cupertino',
 'state': 'CA',
 'zip': '95014',
 'country': 'United States',
 'phone': '408 996 1010',
 'website': 'https://www.apple.com',
 'industry': 'Consumer Electronics',
 'industryDisp': 'Consumer Electronics',
 'sector': 'Technology',
 'longBusinessSummary': 'Apple Inc. designs, manufactures,
 'fullTimeEmployees': 164000,
 'companyOfficers': [{'maxAge': 1,
   'name': 'Mr. Timothy D. Cook',
   'age': 61,
   'title': 'CEO & Director',
   'yearBorn': 1961,
   'fiscalYear': 2022,
   'totalPay': 16425933,
   'exercisedValue': 0,
   'unexercisedValue': 0},
  {'maxAge': 1,
   'name': 'Mr. Luca  Maestri',
   'age': 59,
   'title': 'CFO & Sr. VP',
...
 'grossMargins': 0.43181,
 'ebitdaMargins': 0.32145,
 'operatingMargins': 0.29163,
 'financialCurrency': 'USD',
 'trailingPegRatio': 2.8478}
```

It has so much data. But, the required columns are only date, open, high, low, close, and volume.

```
hist = stock.history( period='1mo')
data = hist[['Open', 'High', 'Low', 'Close', 'Volume']]
```

```
data
```

Date	Open	High	Low	Close	Volume
2023-04-13 00:00:00-04:00	161.406750	165.570988	161.197033	165.331314	68445600
2023-04-14 00:00:00-04:00	164.362658	166.090279	163.593732	164.981812	49386500
2023-04-17 00:00:00-04:00	164.861964	165.161553	163.803431	165.001770	41516200
2023-04-18 00:00:00-04:00	165.870582	167.178771	165.421192	166.240067	49923000
2023-04-19 00:00:00-04:00	165.570994	167.927735	165.311343	167.398468	47720200
2023-04-20 00:00:00-04:00	165.860576	167.638116	165.331309	166.419800	52456400
2023-04-21 00:00:00-04:00	164.822026	166.220086	164.262802	164.792068	58337300
2023-04-24 00:00:00-04:00	164.772093	165.371270	163.663625	165.101639	41949600
2023-04-25 00:00:00-04:00	164.961829	166.080278	163.503839	163.543793	48714100

The date has a timestamp as well. As it is not required, it is removed. After transformation, the data's info looks like

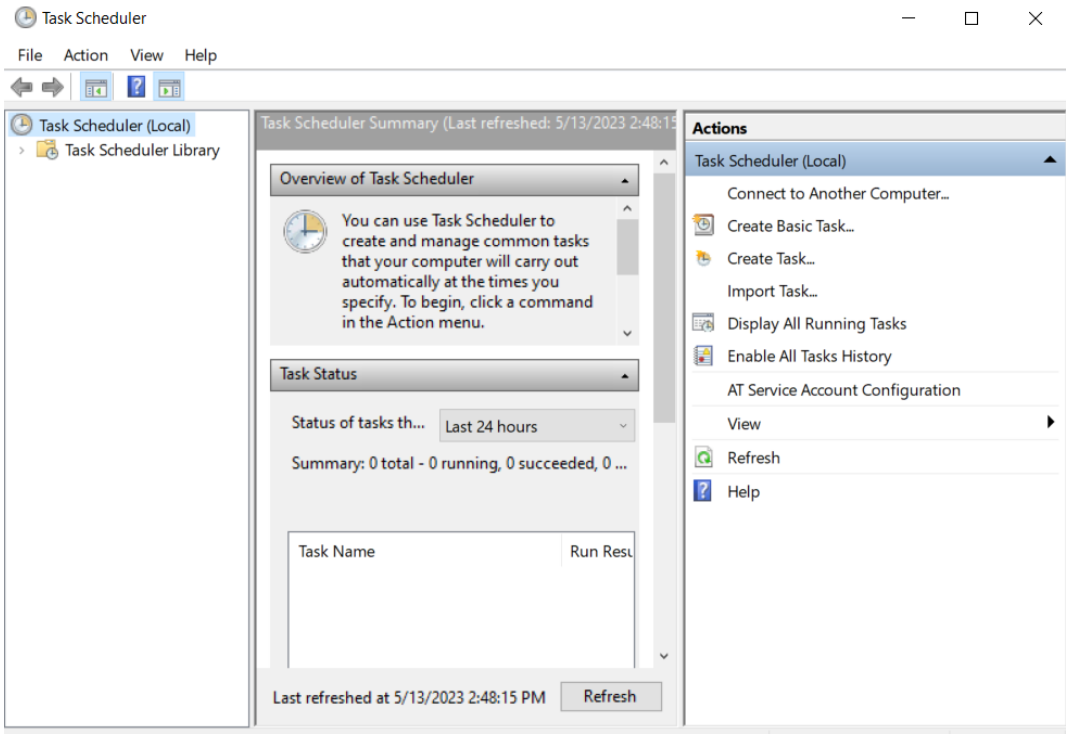
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 22 entries, 0 to 21
Data columns (total 6 columns):
 #   Column  Non-Null Count  Dtype         
---  ------  --------------  -----         
 0   date    22 non-null     datetime64[ns]
 1   open    22 non-null     float64       
 2   high    22 non-null     float64       
 3   low     22 non-null     float64       
 4   close   22 non-null     float64       
 5   volume  22 non-null     int64         
dtypes: datetime64[ns](1), float64(4), int64(1)
memory usage: 1.2 KB
```

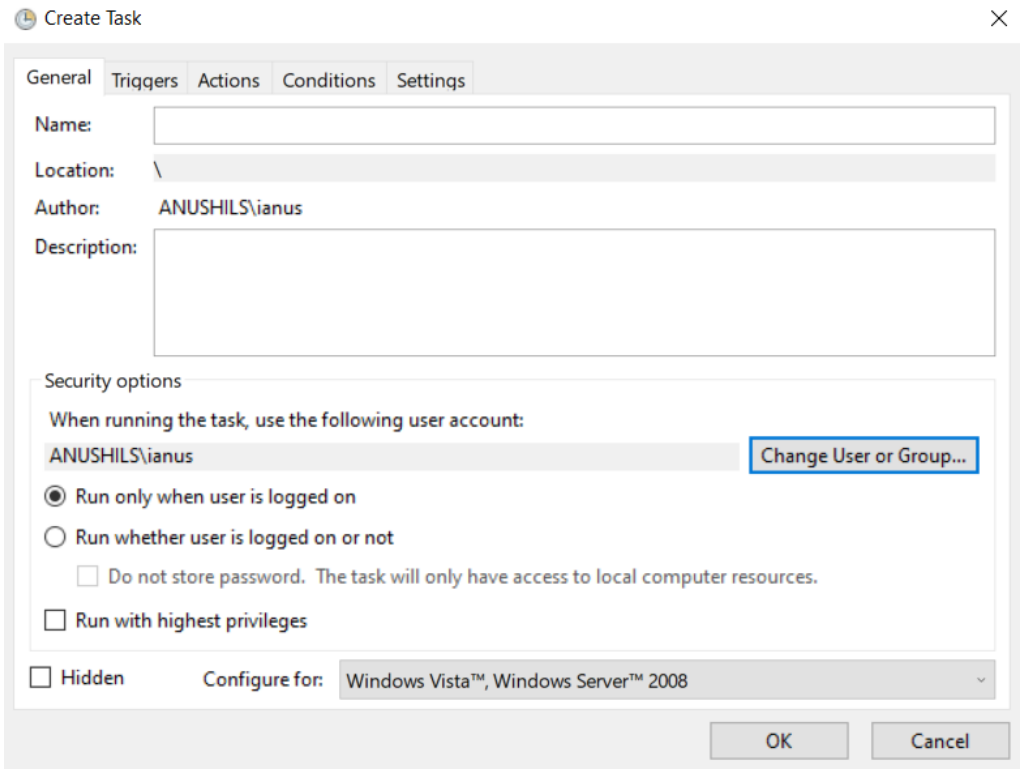
The dataset is merged with the dataset we got by merging above. Duplicates are removed as above.

4) Schedule the latest.ipynb dataset to run every month

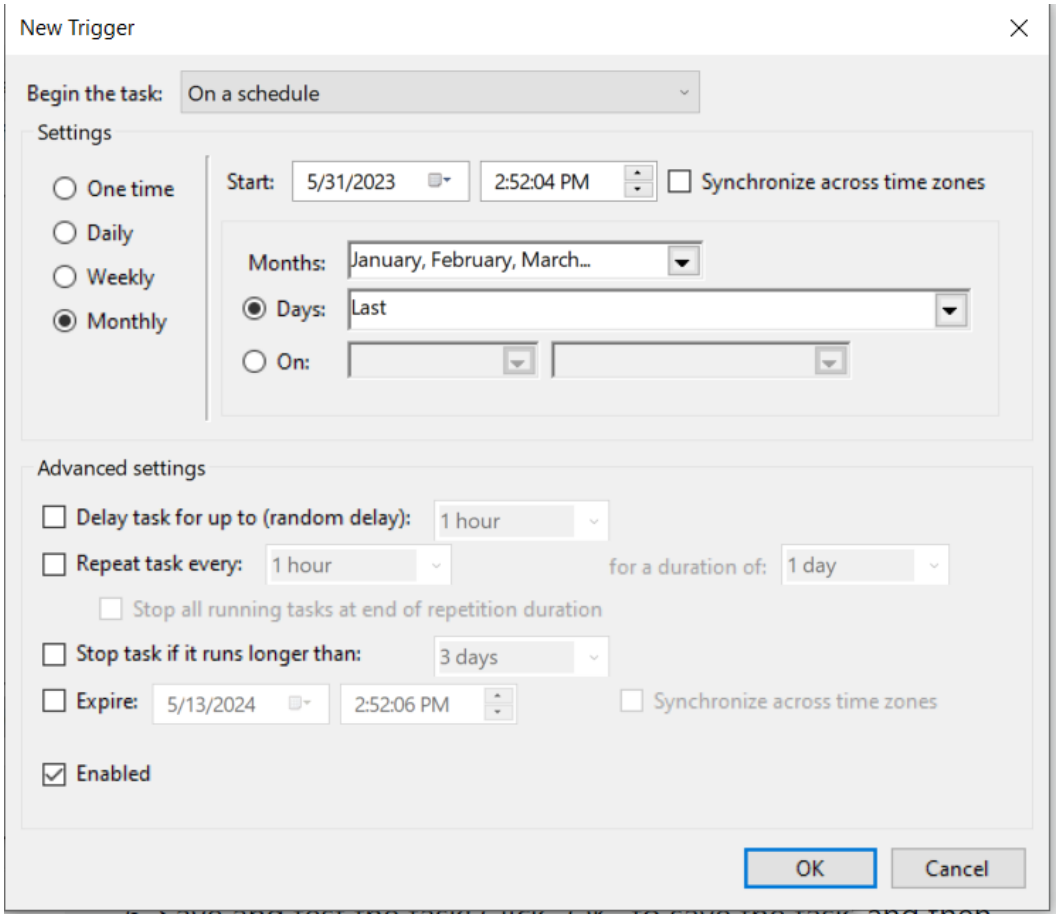
As the data is to be integrated every month, it is done in the following ways:



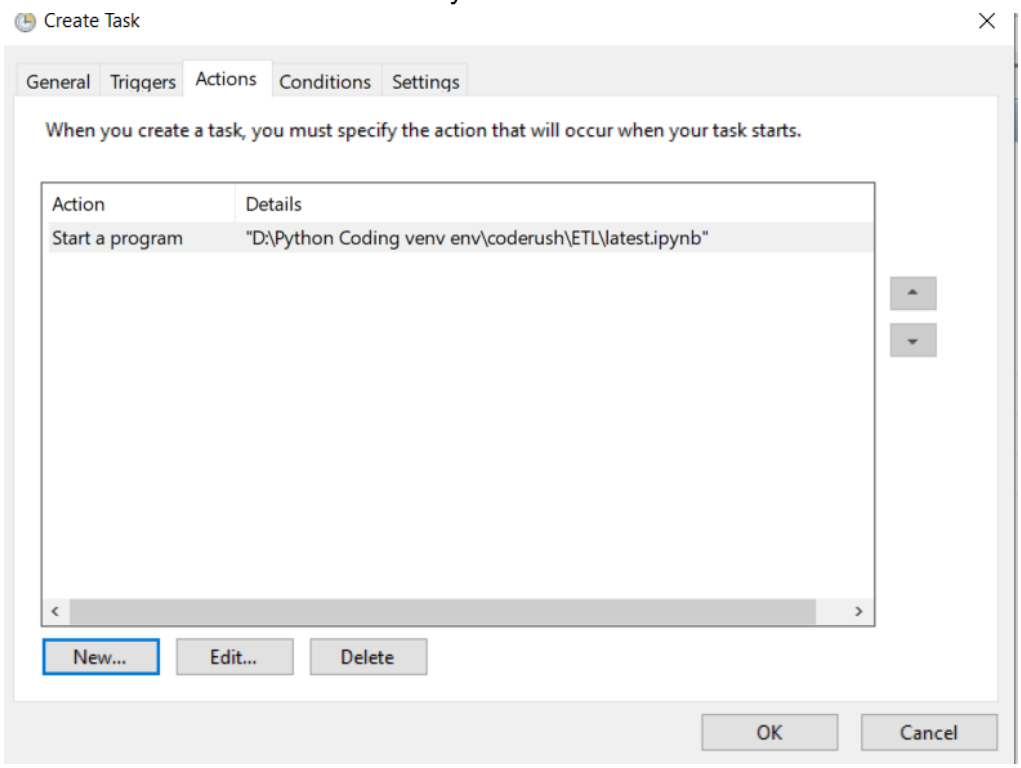
Task Scheduler is available in Windows.
A task is created.



A trigger is set.



The task is saved and will run every month end.



5) Use Microsoft SQL Server to Change CSV file to a tabular format

As a data warehouse has a fixed schema, it is easier to use a SQL Database to store and query the data.
The snippet of the database table is shown below.

Results | Messages

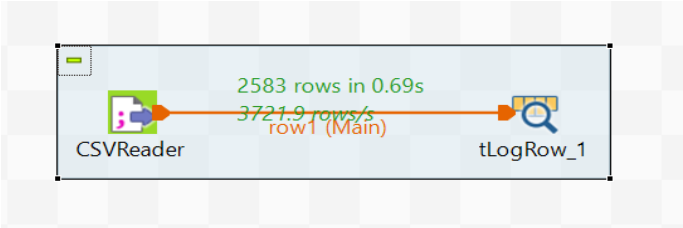
	date	open	high	low	close	volume
1	2013-02-08	78.34	79.72	78.01	78.9	1298137
2	2013-02-11	78.65	78.91	77.23	78.39	758016
3	2013-02-12	78.39	78.63	77.5132	78.6	876859
4	2013-02-13	78.9	79.13	77.85	78.97	1038574
5	2013-02-14	78.66	79.72	78.585	78.84	1005376
6	2013-02-15	78.83	79.18	77.93	79	1247063
7	2013-02-19	79.12	81.44	78.57	80.72	1730690
8	2013-02-20	80.42	80.94	79.39	79.5	648693
9	2013-02-21	79.26	80.05	78.56	79.06	915160
10	2013-02-22	79.2	79.46	78.63	79.21	538188
11	2013-02-25	79.46	79.74	78.36	78.36	579811
12	2013-02-26	78.42	78.925	76.42	77.15	990145
13	2013-02-27	77.29	77.58	76.22	77.3	720813
14	2013-02-28	77.35	77.59	76.34	76.34	1180713
15	2013-03-01	76.37	76.66	75.623	76.37	1272624
16	2013-03-04	76.24	77.25	76.1	77.02	1098428
17	2013-03-05	77.07	77.49	76.72	76.95	1288012

But, the updated dataset in the future cannot be integrated into the table in the database using SSMS as it requires SSIS which is a paid service.

6) Using Talend to automate the change in CSV file to the Database Table

Talend is a popular data integration platform that allows users to connect, transform and share data across various systems and applications. It is an open-source tool that provides a wide range of connectors and components to perform data integration tasks.
Steps to import CSV data in Talend Open Studio:

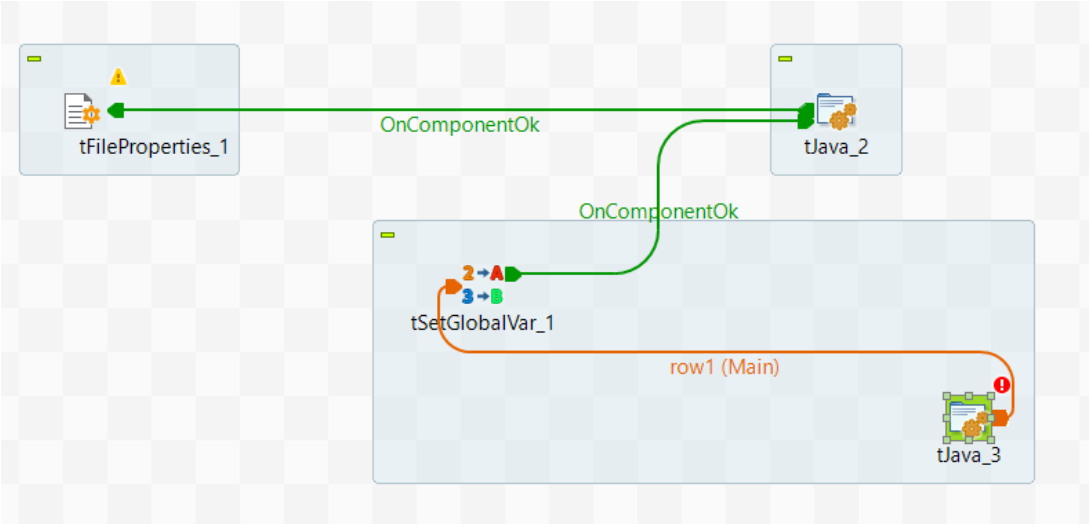
1. Create a new Talend job: Open Talend Open Studio and create a new Talend job.
2. Add a tFileInputDelimited component: Drag a tFileInputDelimited component from the Palette onto the job canvas.
3. Configure the CSV file: Configure the tFileInputDelimited component to read the CSV file. You will need to provide the file name and location, as well as specify the field delimiter (e.g., comma, semicolon, etc.) and text qualifier (if applicable).
4. Define the schema: Define the schema for the CSV file. You can either manually define the schema or use the "Guess schema" feature to automatically create the schema based on the first few rows of the CSV file.
5. Map the columns: Map the columns from the tFileInputDelimited component to the destination component (e.g., tMap, tAggregateRow, tDBOutput, etc.) using the drag-and-drop interface.
6. Run the job: Run the job to import the CSV data.



Logical Mapping.

```
.---------+-------+--------+--------+------+---------.
|                      tLogRow_1                       |
|=---------+-------+--------+--------+------+--------=|
|date      |open   |high    |low     |close |volume   |
|=---------+-------+--------+--------+------+--------=|
|05-08-0013|78.34  |79.72   |78.01   |78.9  |1298137  |
|05-08-0016|78.65  |78.91   |77.23   |78.39 |758016   |
|05-08-0017|78.39  |78.63   |77.5132 |78.6  |876859   |
|06-08-0018|78.9   |79.13   |77.85   |78.97 |1038574  |
|06-08-0019|78.66  |79.72   |78.585  |78.84 |1005376  |
|05-08-0020|78.83  |79.18   |77.93   |79.0  |1247063  |
|05-08-0024|79.12  |81.44   |78.57   |80.72 |1730690  |
|05-08-0025|80.42  |80.94   |79.39   |79.5  |648693   |
|06-08-0026|79.26  |80.05   |78.56   |79.06 |915160   |
|06-08-0027|79.2   |79.46   |78.63   |79.21 |538188   |
|06-08-0030|79.46  |79.74   |78.36   |78.36 |579811   |
|06-08-0031|78.42  |78.925  |76.42   |77.15 |990145   |
|05-08-0032|77.29  |77.58   |76.22   |77.3  |720813   |
|05-08-0033|77.35  |77.59   |76.34   |76.34 |1180713  |
|03-09-0006|76.37  |76.66   |75.623  |76.37 |1272624  |
|03-09-0009|76.24  |77.25   |76.1    |77.02 |1098428  |
|03-09-0010|77.07  |77.49   |76.72   |76.95 |1288012  |
|03-09-0011|77.1   |77.24   |75.97   |76.18 |1255676  |
|02-09-0012|76.04  |76.75   |75.7    |76.22 |965808   |
|03-09-0013|76.44  |77.16   |75.6302 |76.84 |736386   |
|02-09-0016|76.88  |78.06   |76.84   |77.53 |689362   |
|03-09-0017|77.26  |77.66   |76.89   |77.36 |660535   |
|03-09-0018|77.54  |78.85   |77.32   |78.77 |644976   |
|03-09-0019|78.9   |79.12   |77.36   |77.67 |763234   |
|02-09-0020|77.9   |79.59   |77.41   |79.57 |1237958  |
|03-09-0023|79.09  |80.76   |79.0101 |80.7  |1219201  |
|02-09-0024|80.64  |80.9399 |79.77   |80.07 |1370222  |
```

The table has the following structure.



To automate the update of the data in the database table, the above structure was created. But, it was full of errors, and could not understand the Java code. So, the last step was left without solving the error.

Using AWS Cloud Service

To solve the problem of continuous monitoring and integration, AWS is a great tool.

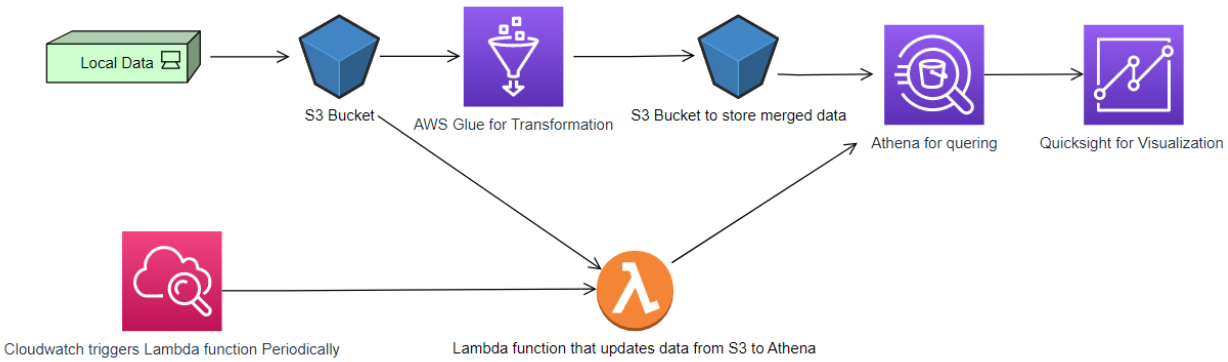
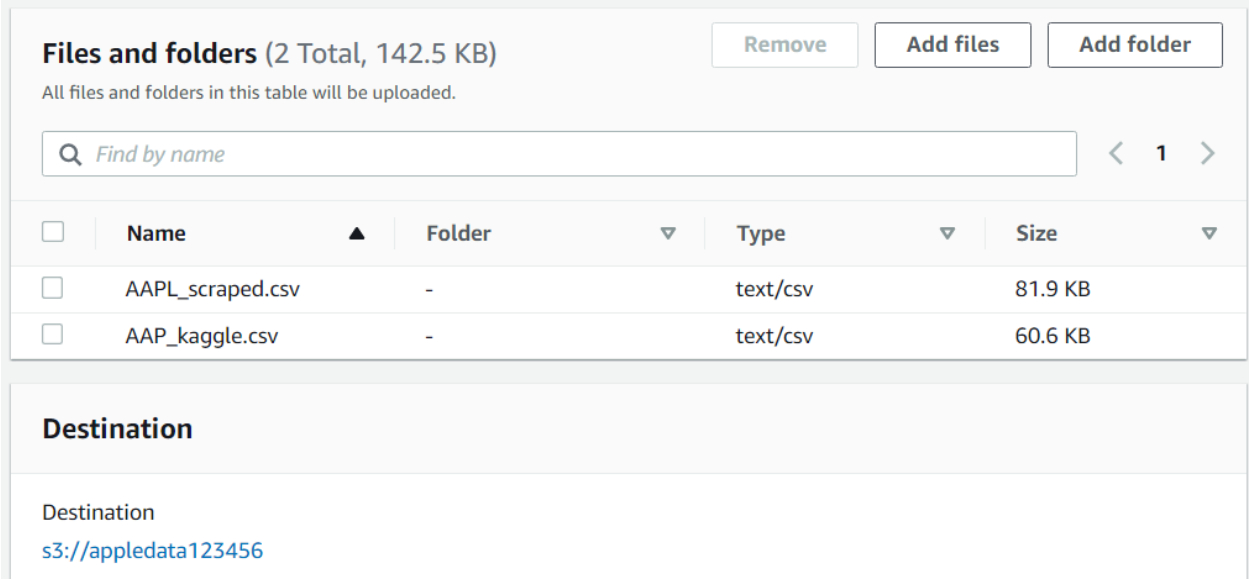


Fig: Architecture of the project using AWS

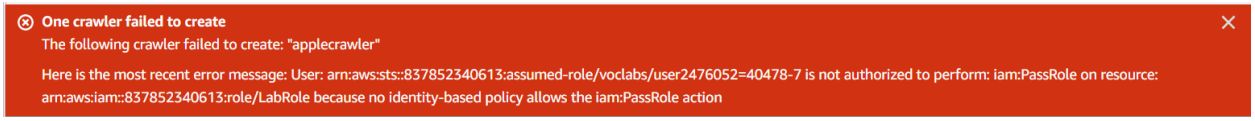
Loading the Data into S3

The scraped data as well as the data collected were loaded in a S3 bucket.



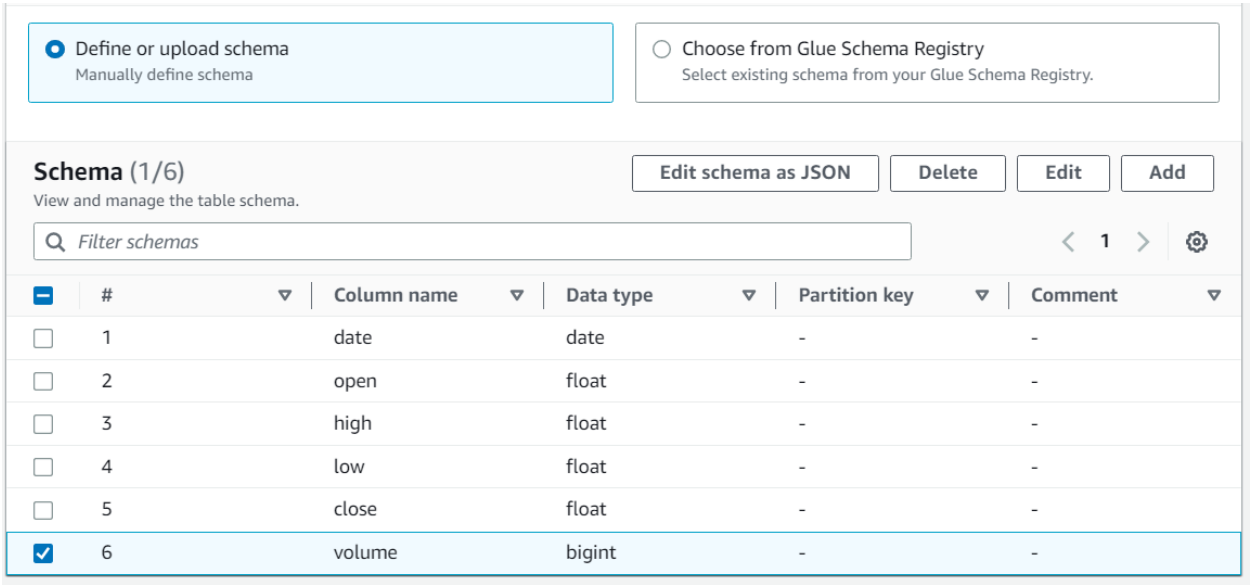
Using AWS Glue for ETL Operations

Glue was to be used for 2 things: Crawlers and Glue Jobs.
Crawlers can automatically find the schema of the given dataset and continuously take the updated data from the S3 bucket. A crawler was setup but gave an error:



The permission to create a Crawler was not given to the role LabRole present in the sandbox account. Also, defining any policies or roles for a user was not possible as Admin access was not present.

So, a manual schema was defined as in Fig below.



ETL Glue job was set up. The input data from a S3 bucket was transformed into another form and saved back to the S3 bucket which was created to hold results. All the schemas were defined as required. But, the same problem occurred as the LabRole had no permission to run a Glue Job.

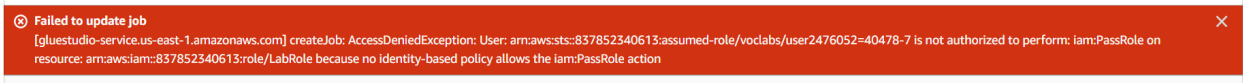


Fig: Glue Job Error

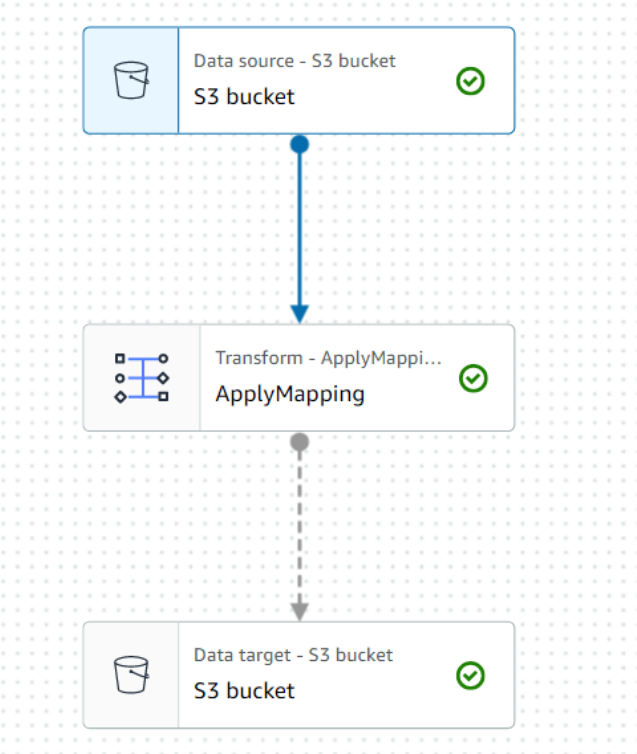


Fig: Glue Job Steps

Schema Info

Key	Data type
date	date
open	float
high	float
low	float
close	float
volume	float

Fig: Output schema of transformation

Use Athena to Query the Data

Redshift would be the ideal data warehouse. But, the permission for Redshift was not present and it could not be used. So, Athena was used as an alternative. It gave the option of using SQL to query the database.

```
1  select *
2  from appletable;
```

Fig: SQL query in Athena to select all the data from the created table

#	date	open	high	low	close	volume
2	2013-02-08	78.34	79.72	78.01	78.9	1298137
3	2013-02-11	78.65	78.91	77.23	78.39	758016
4	2013-02-12	78.39	78.63	77.5132	78.6	876859
5	2013-02-13	78.9	79.13	77.85	78.97	1038574
6	2013-02-14	78.66	79.72	78.585	78.84	1005376
7	2013-02-15	78.83	79.18	77.93	79.0	1247063
8	2013-02-19	79.12	81.44	78.57	80.72	1730690
9	2013-02-20	80.42	80.94	79.39	79.5	648693
10	2013-02-21	79.26	80.05	78.56	79.06	915160
11	2013-02-22	79.2	79.46	78.63	79.21	538188
12	2013-02-25	79.46	79.74	78.36	78.36	579811
13	2013-02-26	78.42	78.925	76.42	77.15	990145
14	2013-02-27	77.29	77.58	76.22	77.3	720813

Fig: Snippet of the output data

```
CREATE EXTERNAL TABLE IF NOT EXISTS `applemerged`.`appletable` (
  `date` date,
  `open` float,
  `high` float,
  `low` float,
  `close` float,
  `volume` bigint
)
ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe'
WITH SERDEPROPERTIES ('field.delim' = ',')
STORED AS INPUTFORMAT 'org.apache.hadoop.mapred.TextInputFormat' OUTPUTFORMAT 'org.apache
    .hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat'
LOCATION 's3://applemerged12345/'
TBLPROPERTIES ('classification' = 'csv');
```

Fig: Code to create a table apple table in Athena

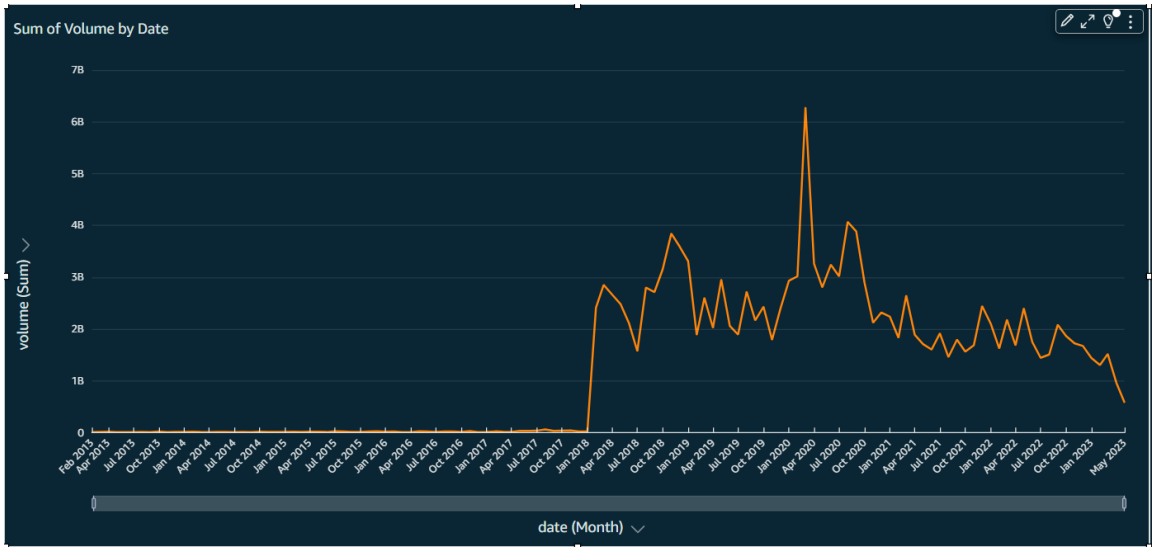
This code created a table called apple table which contained all the required columns and their respective data types. It was selected in the console and the Athena generated the code according to the selection in the console.

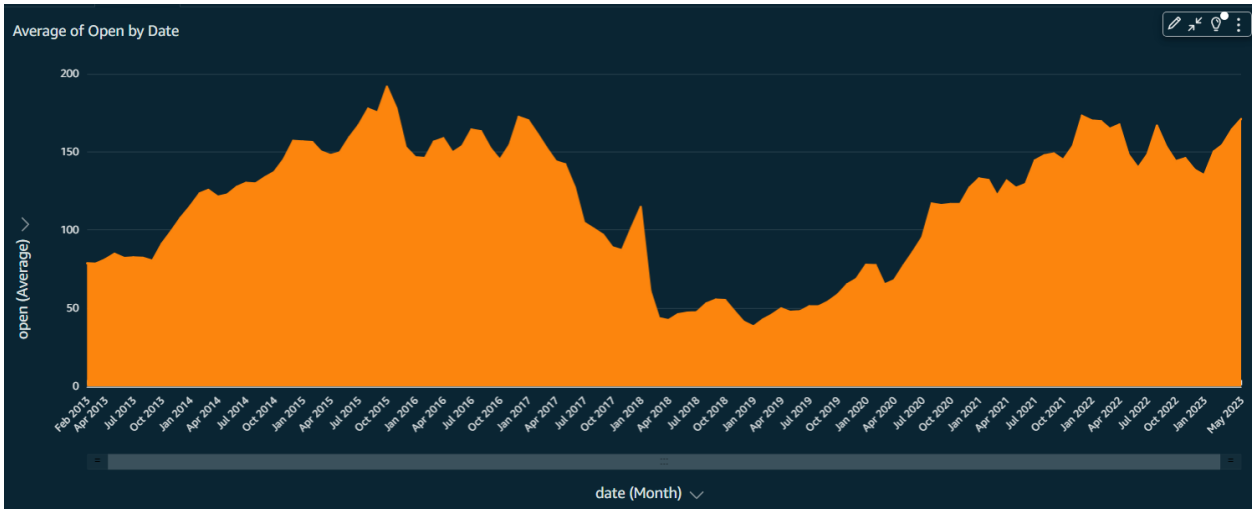
A view was created which included all the data for visualization.

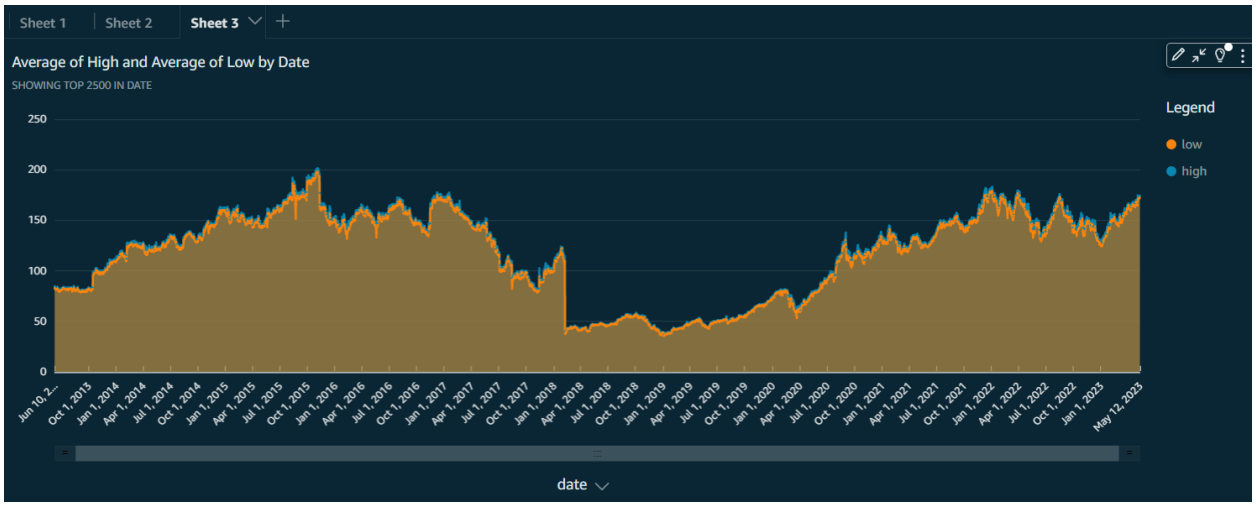
```
create view appleview as
select *
from appletable
```

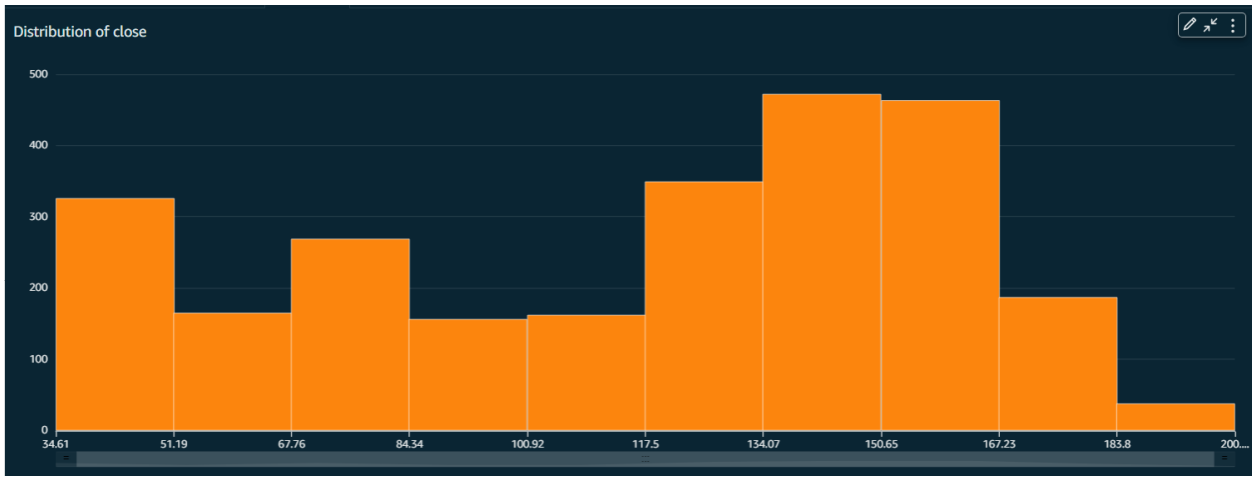
Using Quicksight to Visualize the Data

To show the use case of the data warehouse, some visualizations were created using the view created above in Quicksight.



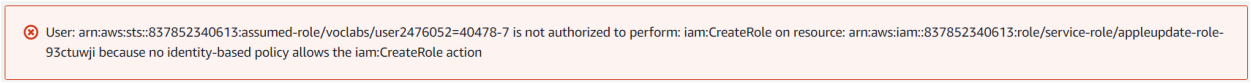

Average of Open by Date
200
150
100
50
0
open (Average)
Feb 2013
Apr 2013
Jul 2013
Oct 2013
Jan 2014
Apr 2014
Jul 2014
Oct 2014
Jan 2015
Apr 2015
Jul 2015
Oct 2015
Jan 2016
Apr 2016
Jul 2016
Oct 2016
Jan 2017
Apr 2017
Jul 2017
Oct 2017
Jan 2018
Apr 2018
Jul 2018
Oct 2018
Jan 2019
Apr 2019
Jul 2019
Oct 2019
Jan 2020
Apr 2020
Jul 2020
Oct 2020
Jan 2021
Apr 2021
Jul 2021
Oct 2021
Jan 2022
Apr 2022
Jul 2022
Oct 2022
Jan 2023
May 2023
date (Month)


Sheet 1
Sheet 2
Sheet 3
Average of High and Average of Low by Date
SHOWING TOP 2500 IN DATE
Legend
low
high
250
200
150
100
50
0
Jun 10, 2...
Oct 1, 2013
Jan 1, 2014
Apr 1, 2014
Jul 1, 2014
Oct 1, 2014
Jan 1, 2015
Apr 1, 2015
Jul 1, 2015
Oct 1, 2015
Jan 1, 2016
Apr 1, 2016
Jul 1, 2016
Oct 1, 2016
Jan 1, 2017
Apr 1, 2017
Jul 1, 2017
Oct 1, 2017
Jan 1, 2018
Apr 1, 2018
Jul 1, 2018
Oct 1, 2018
Jan 1, 2019
Apr 1, 2019
Jul 1, 2019
Oct 1, 2019
Jan 1, 2020
Apr 1, 2020
Jul 1, 2020
Oct 1, 2020
Jan 1, 2021
Apr 1, 2021
Jul 1, 2021
Oct 1, 2021
Jan 1, 2022
Apr 1, 2022
Jul 1, 2022
Oct 1, 2022
Jan 1, 2023
May 12, 2023
date


Distribution of close
500
400
300
200
100
0
34.61
51.19
67.76
84.34
100.92
117.5
134.07
150.65
167.23
183.8
200....

Create a Lambda Function to find changes in the CSV file

It was planned to use a Lambda function along with Cloudwatch to update the Athena data with new data. Cloudwatch can be used to periodically trigger the Lambda function. Then the lambda function will update the data from S3 to Athena. But, the permission to create the Lambda function was not there.



The best case scenario would be to update the CSV data in the S3 bucket using AWS Cloud9 to generate the data using Python programming language and using AWS Glue Crawler, the data could be automatically updated. But, the lack of proper permissions in the given role did not provide this easy option.

Result and Conclusion

A data warehouse using ETL was created both with and without the use of AWS. The data warehouse can be used in various use cases like to create intelligence reports, find out underlying patterns and rules, and so on.

The project could be completed with and without the use of AWS. But, the process was much easier when AWS was used. An AWS account with admin privilege or at least all the required permission given will completely automate the ETL process to create the Data Warehouse.