**----------------------------------------------------------------------------------------**

**----------------------------------------------------------------------------------------**

Comments from professor

Just a couple of points:

You might check out AWS Keyspaces as it is compatible with Cassandra.

You don’t need to use very large volumes of data, just enough to demonstrate your pipeline. So, hundreds or thousands of records, not millions, unless you want to do that.

For your next review of the literature, I would want you to focus on wide column databases in more detail. For example, you could discuss Cassandra, HBase and Google BigTable. In this case what I would want to see is an organized comparison among those databases:

* + Describe each database briefly at a high level
  + Then compare major capabilities among each database. For example, database X can do A and B, but database Y and do A, only some of B, but it can also do C.
  + Then discuss the benefits and drawbacks of the databases. Does one trade performance for more reliability or one has fewer features but each feature offers more tuning options. What about scalability, performance, security.
  + Note: here are some BigTable references:
    - [https://static.googleusercontent.com/media/research.google.com/en//archive/bigtable-osdi06.pdf](https://static.googleusercontent.com/media/research.google.com/en/archive/bigtable-osdi06.pdf)
    - https://cloud.google.com/bigtable/docs/overview
* Remember to profile your data

**----------------------------------------------------------------------------------------**

**---------------------------------------------------------------------------------------------**

**Financial Stock Data Pipeline**

Ajay Mohan (amohan6@hawk.iit.edu)

Anjali Veer (aveer@hawk.iit.edu)

Sachin Janwalkar (sjanwalkar@hawk.iit.edu)

**Abstract.** With fluctuating stock values, it's becoming more difficult to manually monitor current and past prices on a regular basis and predict future behavior. Our application will assist end users in comparing average stock prices from previous years to the present, allowing them to analyze how a stock performed in the past, forecast how a stock will perform in the near future, and see how prices have changed over time. The goal of this project is to create a real-time stock data pipeline. The end unit of this pipeline will be visualization of average stock prices and real time stock charts. Flow of the project is pulling stock information using APIs and processing this information using a messaging queue. As we are aiming to process real-time stock data, we will be using streaming. This streamed data will be used to calculate the average prices. These calculated average prices will be stored in a data lake and then further will be accessed by a visualization tool.

**Keywords:** big data, big data tools, stocks, data visualization, analytics, decision making

**1 Introduction**

We are living in a world where a huge amount of stock information is created, gathered and processed on a daily basis. So why is such a huge amount of data generated? It is due to the ever changing stock prices. The frequent fluctuations in prices creates this data.

This data if analyzed carefully can give users a pattern of information which can help them make decisions of buying or selling a stock. The size, variety and rapid changes of such data created a need for big data analytics as well as different storage and analysis methods. Such sheer amounts of big data need to be properly analyzed, and pertaining information should be extracted.

The aim of this project is to average stock prices over a period of time which clearly shows the price fluctuations and could predict the pattern of price changes. Accordingly, some of the various big data tools, methods, and technologies which can be applied are discussed and they are applied.

**2 Stock Data Analysis**

2.1 Input to the system:

We pull the yahoo finance dataset using the script. yfinance library offers a reliable, threaded, and Pythonic way to download historical market data from Yahoo Finance.

As input for our system, we will use the yahoo finance API. This API will provide daily stock information. The fields that will be used from the information will be as below:

URL:

List of fields:

| Column Name | Data Type | Description |
| --- | --- | --- |
| Date | date | Date of the stock trading |
| Open | float64 | the price at which a stock started trading |
| High | float64 | Highest price at which a stock traded during a period |
| Low | float64 | lowest price of the period |
| Close | float64 | Stock value of the day closing |
| Adj Close | float64 | more accurate measure of stocks' value of that closing of the period |
| Volume | int64 | Total volume that is traded during period |

2.2 Messaging Queue:

As for data transportation, we will be using Kafka

Kafka streaming allows us to perform functional aggregations and mutations. Kafka is more flexible than Kinesis but in kafka we need to manage the cluster ourselves, and it also requires some dedicated DevOps resources to keep it going.

2.3 Raw data storage:

The purpose of storing data in a database at this step is for backup of all raw data coming into the system. We will use a NoSQL database for backing up the records coming from Kafka as they have flexible data models, can scale horizontally and have fast queries. Most of the operations are only going to write data to the DB, so this will be a Write heavy task. There will be very few instances where we would need to read from the DB. We would also need this database to be readily available because in case there is a situation where we have some results and we might need to check original records contributing to the result rather than processed ones, we should be able to get them from this database.

For our project, we compared 4 different NoSQL databases to choose from: MongoDB, Cassandra, ElasticSearch and HBase. On comparing all these databases and our requirements (mentioned above), Cassandra seems a better choice. It is an open source, distributed database system. It can handle petabytes of information and thousands of concurrent requests per second. Our use case involves many write operations as compared to read operations and hence Cassandra is preferred. Also, Cassandra provides CQL which is easier to use as we had a background in using SQL.

And Since we will be creating a real time chart of the Financial stocks, it's pertinent that we have a DB that has a rapid response time and Cassandra is best suited for that.Scalability is also another factor why we chose Cassandra.

MongoDB would have been a good choice if we would have required lots of read/write operations. HBase is also a good option, but it is more suitable where the volume is huge. For our application, we would need a system supporting all amounts of data, from small to medium to huge. ElasticSearch provides very fast search and could have been used if we had to store logs of the system, wherein we would have searched faster for analyzing them. Based on these few pointers we will be proceeding with Cassandra in our project.

Looking further into column databases, we explored and compared Cassandra, HBase and Google BigTable and below are some comparisons based on few factors:

**Features:**

Cassandra and HBase are both open source, which is part of the reason they place #1 and #2 in the Popularity ranking for wide-column stores, compared to Google’s commercial cloud-based Bigtable, which places at #7. All three databases support both Unix and Windows Operating Systems and are schema-free, but only Cassandra offers data typing.

None of the databases has native XML support and only Cassandra allows for restricted secondary indexes, whereas Bigtable and HBase do not at all. Cassandra is also the only database in this comparison which enables SQL-like DML and DDL statements to be queried using their own proprietary language called CQL (Cassandra Query Language). Whilst Bigtable and HBase both support Java APIs, HBase also supports a Cassandra proprietary protocol known as Thrift. Furthermore, HBase and Cassandra support a whole list of different programming languages while Bigtable restricts code to Go and Java. This makes HBase and Cassandra the more versatile and easy to access databases compared to Google’s Bigtable.

All three databases offer sharding as a data partitioning method, and can operate with immediate consistency, but Cassandra is the only database offering an optional configuration for eventual consistency as well. This means that Cassandra can offer low latency responses to read requests for highly time-sensitive applications, although at the risk of returning stale data because there is only eventual consistency between the nodes. While all three databases have concurrency and durability, only Bigtable offers atomic single-row operations as a transaction concept.

**Performance:**

Cassandra maps partition keys onto a token ring using consistent hashing to determine where to store the data. By hashing the partition key every node is able to know the range it belongs to and from there the node in charge of this range. Availability and replication strategy depend on the implementation of the database but it means that Cassandra is a true peer-to-peer system, with no master nodes (and no single point-of-failures). It also means that you can send your queries to any node in the cluster (or even better have your driver sent the request to the most appropriate node). This makes Cassandra extremely quick at returning complicated queries.

This reflects what we have seen in practice, that HBase is slower than Cassandra, which is also supported by most of the benchmarks out there. Cassandra architecture is based on DynamoDB (AWS) and Bigtable design. It’s very fast specifically in workloads which it was designed for (there are many benchmarks for 1 million writes a second). However, Bigtable can handle pretty much everything you throw at it, with some benchmarks showing up to 2 million records/second write, although this comes at a price.

Unlike other NoSQL databases, HBase operations run in real-time on its database rather than MapReduce jobs. HBase is partitioned to tables, and tables are further split into column families. Versioning is available so that previous values of the data can be fetched (the history can be deleted every now and then to clear space via HBase compactions). This makes HBase perfect for real-time querying of Big Data. For example, Facebook uses it for messaging and real-time analytics.

HBase is optimized for reads, supported by single-write master, and results in a strict consistency model, as well as use of Ordered Partitioning which supports row-scans. HBase is well suited for doing Range based scans. However, HBase isn’t fully ACID compliant, although it does support certain properties. Last but not least – in order to run HBase, ZooKeeper is required – a server for distributed coordination such as configuration, maintenance, and naming.

Google Cloud Bigtable is accessible via the HBase API. The performance of the database is comparable but somewhat faster than operating HBase on an off-the shelf server. Because Cloud Bigtable is accessed through the HBase API, it is natively integrated with much of the existing big data and Hadoop ecosystem and supports Google’s big data products. Additionally, data can be imported from or exported to existing HBase clusters through simple bulk ingestion tools using industry-standard formats. As such, Cloud Bigtable excels at large ingestion, analytics, and data-heavy serving workloads. It’s ideal for enterprises and data-driven organizations that need to handle huge volumes of data. Conclusively, while all database systems excel at handling large and complex data-heavy workloads, Cassandra has demonstrated the best performance for view loads.

2.4 Data processing:

As for data processing we will be using Spark, which  **runs 100 times faster in-memory**, and 10 times faster on disk. It's also been used to sort 100 TB of data 3 times faster than Hadoop MapReduce on one-tenth of the machines.

Spark, on the other hand when compared with Hive, is the **best option for running big data analytics**. It provides a faster, more modern alternative to MapReduce.

2.5 Processed data storage:

All Redis data resides in memory, which enables low latency and high throughput data access. Unlike traditional databases, In-memory data stores don’t require a trip to disk, reducing engine latency to microseconds. Because of this, in-memory data stores can support an order of magnitude more operations and faster response times. The result is blazing-fast performance with average read and write operations taking less than a millisecond and support for millions of operations per second.

Redis also supports a vast variety of data structures to meet your application needs and enables us to write traditionally complex code with fewer, simpler lines.

2.6 Data visualization and decision making:

Main aim of data visualization is to present data in such a way that end users can make sense of it and the purpose of data analysis works out well. We have multiple tools now-a-days which can be used for showing data to customers. This ranges from normal HTML pages to a small web page to data visualization tools like Tableau and Power BI.

Both tools Tableau and Power BI are great in their own way. Tableau platform is known for its data visualization functionality whereas Power BI offers a number of data points for data visualization. Major things is where **Tableau** can handle **huge volumes of data** with better performance, Power BI can handle only a limited amount of data. Also, as we could create “no-code” data queries using Tableau it requires **less cost of training**. It is very fast and easy to create visualizations in Tableau. So, we will be proceeding with using Tableau for our project.

2.7 Deployment:

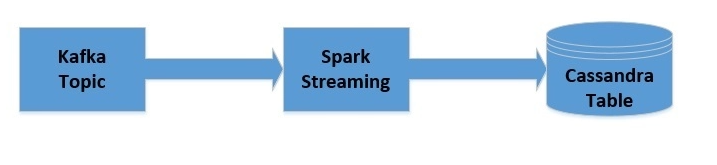
Once the pipeline is created, the project will be deployed in a docker container as it allows rapid deployment and it is also highly versatile as the application can be run from any system or any cloud platform.

**3 Project Results**

This project builds a data pipeline for stock data using Apache Kafka, Spark Stream and Cassandra .The idea behind using this pipeline is to have a highly scalable and fault tolerant data pipeline for a real-time data stream. The stock data for different public healthcare companies such as Anthem, UnitedHealth Group, Humana,Medtronic along with the best performing stocks like Apple and Tesla are fetched from yahoo stocks using a python library and stored in JSON format. These JSON files will be integrated with the kafka topic and the message will be passed on to the Cassandra Table. The Spark stream leverages a checkpoint to retain data that is being passed. The data stored in Cassandra will be used to perform data analysis and visualization using Tableau.

The overview of the pipeline is shown in the below figure.

**

**

**

**

\*add code snippets- kafka, spark and cassandra\*

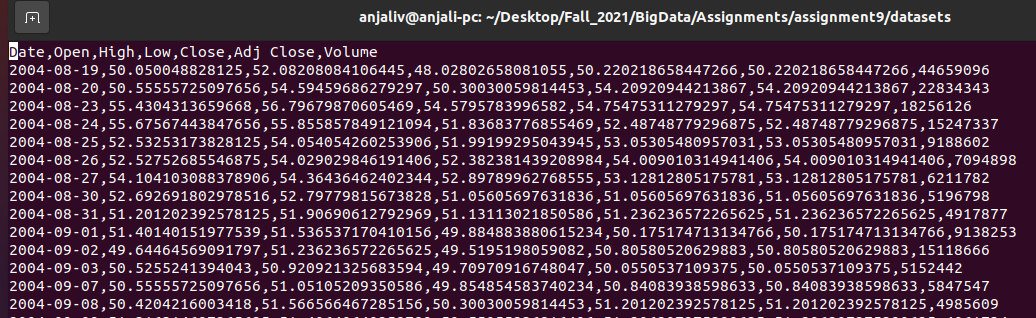
**4 Conclusion**

As part of this project, we have examined and analyzed various NoSQL data storage and processing technology options and their advantages and disadvantages for our use case. As part of this report, we discussed the reasons why the mentioned technologies were used. We compared various NoSQL databases. Conclusively, we found out that each database has its own strengths and weaknesses. After all the comparison, we can say that for our problem statement, Cassandra is the only one enabling SQL like queries and is the fastest database in terms of write speed and hence we used it for storing raw data.

We believe that this stock data analysis is of great significance in this era of data overflow, and can provide unforeseen insights and benefits to decision makers in various areas. If properly exploited and applied, this stock data analysis has the potential to provide a basis for advancements on investment fronts.

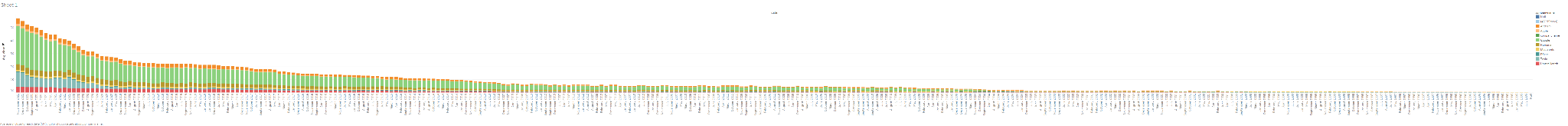
**5 Appendix**

Below are some samples of input data that we used:

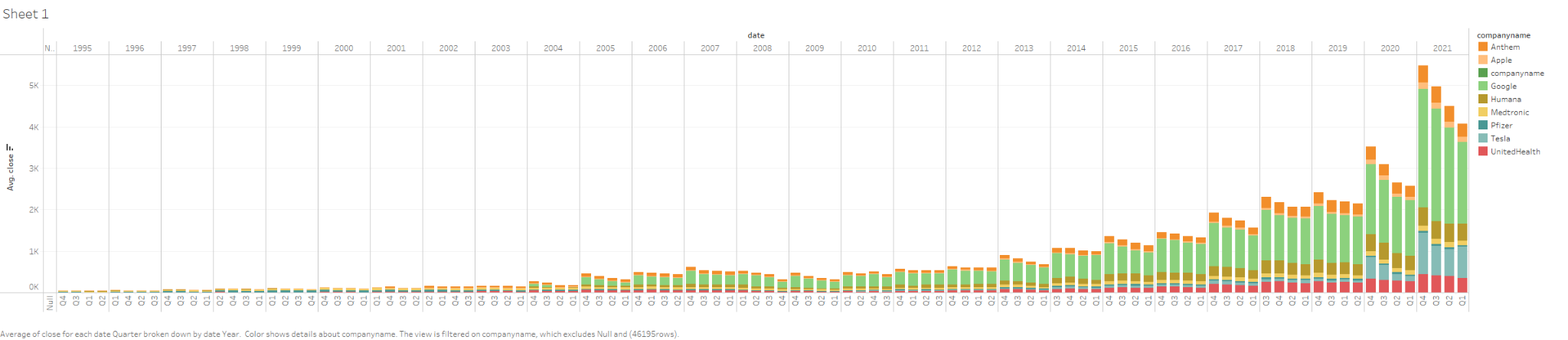


We extracted this data using the yahoo finance library for multiple companies.

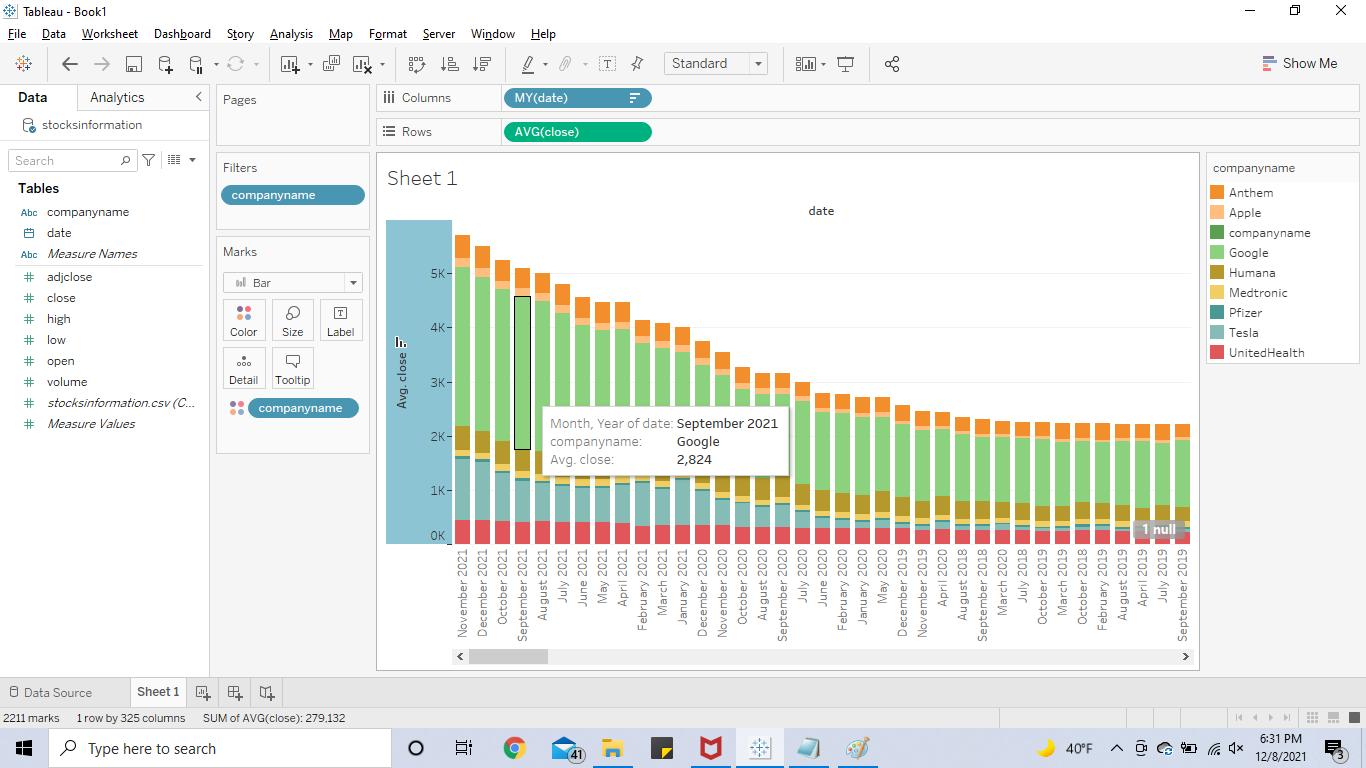
The visualization results for average stock price per month are as below. We have tried to use multiple forms of visualizations. It’s not very clear in this as it’s over 15+ years data there for 7-8 companies. But this below chart shows data per year, per month for the company



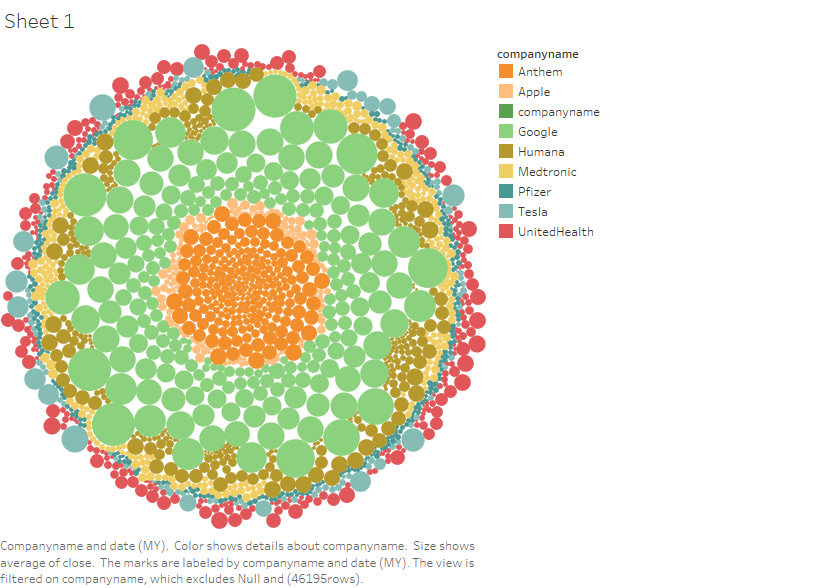
This below image is per year per quarter.



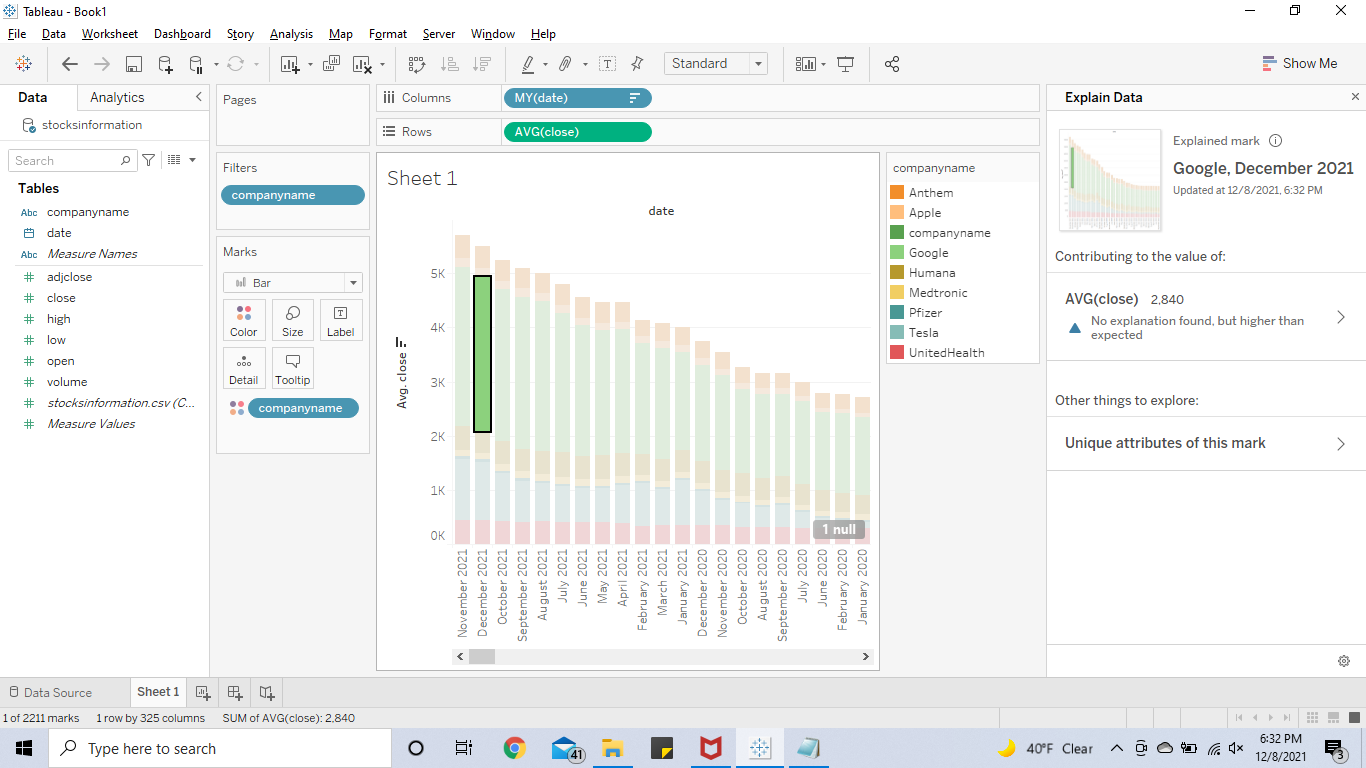
A screenshot for some data is as below. This shows just partial result as full screenshot is huge.



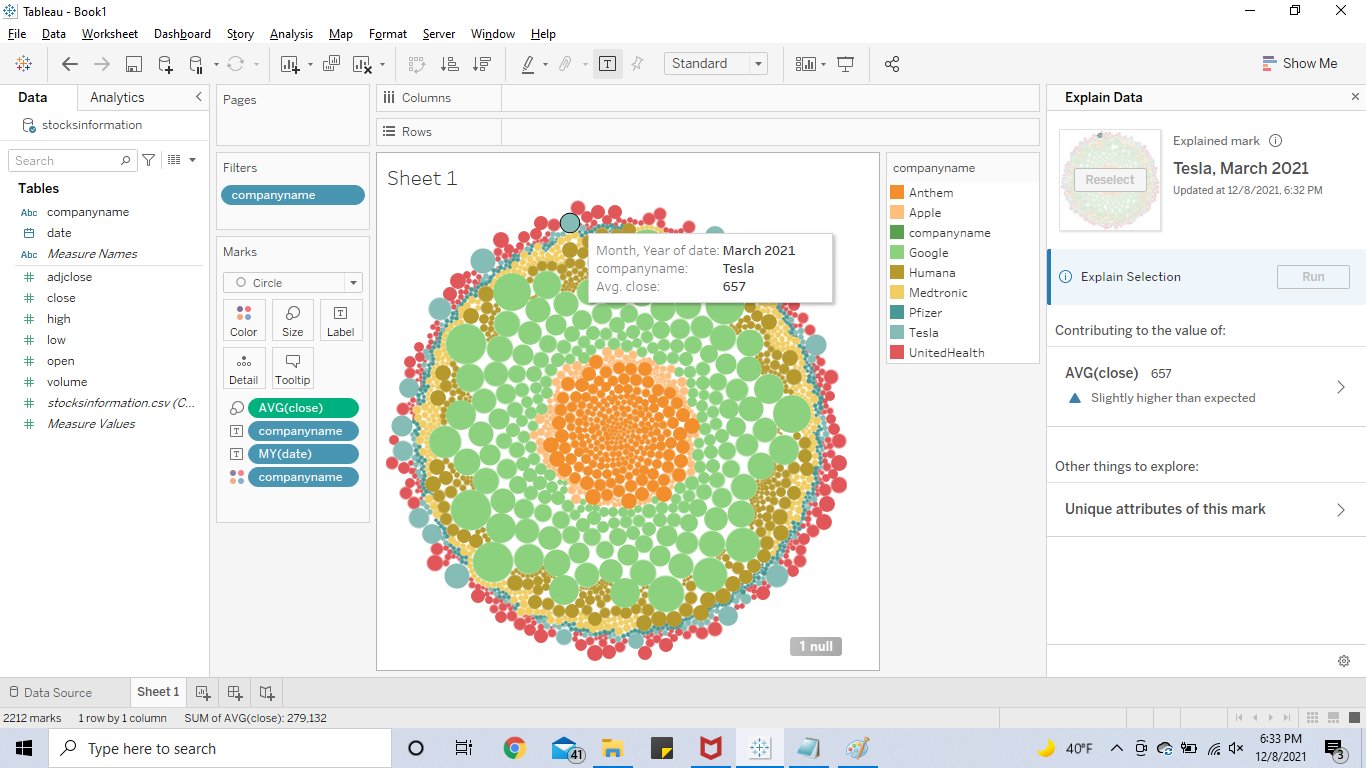
Now, we tried another form of chart which shows a bit more concise result as below:



Below is the explanation of few details checked in the charts:



Similar explanation of data can be viewed for another chart as below:



This explanation of data can help us know the high-level stock price as compared to its earlier price. That’s why we can see descriptions like ‘slightly higher than expected’ or ‘higher than expected’.

This data can be later used for estimating future value of stock.