Big Data Project Report – Ajinkya Pawale

Introduction:

In this project I will be analyzing the stock data. There is a lot of data that is generated on a daily basis in the stock market. Stock market data may be fascinating to study, and excellent prediction models can result in significant financial gains. The amount of financial data available on the internet appears to be limitless. It might be difficult to find a substantial, well-structured dataset on a wide range of organizations. This dataset contains historical stock prices (for the previous five years) for all firms currently listed on the S&P 500 index. S&P 500 is an acronym for Standard and Poor's 500, a stock market index that measures 500 publicly listed domestic firms in the United States. Many investors believe it is the most accurate overall indicator of the success of the American stock market.

The data consists of the following fields:

Date - in format: yy-mm-dd

Open - price of the stock at market open (this is NYSE data so all in USD)

High - Highest price reached in the day

Low - Lowest price reached in the day

Close - price of the stock at market close

Volume - Number of shares traded

Name - the stock's ticker name

The steps included in project are: 1) Get the stock data from Kaggle datasets. Store the data in Mongo DB in the form of collections using python connector pymongo. 2) Make changes to the schema, insert/update/delete records using pymongo. 3) Create a data lake and run queries on it using pymongo. 4) Create visualizations on the data. 5) Use PySpark to extract the data from Mongo DB. 6) Perform preprocessing and data cleaning. 7) Use PySpark to create a ML model to predict the stock prices.

Background:

This dataset lends itself to a variety of visually stimulating displays. Simple tasks like price changes over time, graphing and comparing numerous stocks at once, or generating and graphing new metrics from the data supplied are all possible. Stock statistics such as volatility and moving averages may be easily determined using this information.

The main issue is if you can create a model that can outperform the market and enable you to make statistically sound bets. This can be done with the help of various machine learning models that are available in PySpark module. The reason why we are using PySpark is that it is fast (up to 100x quicker than typical Hadoop MapReduce thanks to in-memory operations), provides robust, distributed, fault-tolerant data objects (called RDD), and combines seamlessly with the realm of machine learning and graph analytics via supplemental packages like Mlib and GraphX.

Apart from that, the reason for choosing Mongo DB as the storage system is because it is schema less where each record is stored in the form of documents containing key value pairs. No complex joins required, easy to scale out, easy to tune and it supports dynamic queries. Internal memory is used to store the (windowed) working set, allowing for quicker data access. It also provides replication, high availability, auto sharding, indexing and fast in place updates. Also, with the help of python connector we can make changes to it using the pymongo library.

Methodology:

Importing the dataset and checking its descriptive statistics:

The dataset is imported as a data frame from a csv file. The head and tail commands are performed to check the contents of initial and final rows in the dataset. The dataset consists of almost 600 K rows. Later, the datatype of each column is checked along with count of null values if any in each column. Also value count is done to check the stock ticker counts present in the data.

```
In [3]: data = pd.read_csv('all_stocks_5yr.csv')

In [4]: data.head()
```

Out[4]:

	date	open	high	low	close	volume	Name
0	2013-02-08	15.07	15.12	14.63	14.75	8407500	AAL
1	2013-02-11	14.89	15.01	14.26	14.46	8882000	AAL
2	2013-02-12	14.45	14.51	14.10	14.27	8126000	AAL
3	2013-02-13	14.30	14.94	14.25	14.66	10259500	AAL
4	2013-02-14	14.94	14.96	13.16	13.99	31879900	AAL

```
In [130]: data.tail()
```

Out[130]:

	date	open	high	low	close	volume	Name	diff
619035	2018-02-01	76.84	78.27	76.69	77.82	2982259	ZTS	1.58
619036	2018-02-02	77.53	78.12	76.73	76.78	2595187	ZTS	1.39
619037	2018-02-05	76.64	76.92	73.18	73.83	2962031	ZTS	3.74
619038	2018-02-06	72.74	74.56	72.13	73.27	4924323	ZTS	2.43
619039	2018-02-07	72.70	75.00	72.69	73.86	4534912	ZTS	2.31

```
In [5]: data.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 619040 entries, 0 to 619039
Data columns (total 7 columns):
 #   Column  Non-Null Count   Dtype
---  ------  --------------   -----
 0   date    619040 non-null  object
 1   open    619029 non-null  float64
 2   high    619032 non-null  float64
 3   low     619032 non-null  float64
 4   close   619040 non-null  float64
 5   volume  619040 non-null  int64
 6   Name    619040 non-null  object
dtypes: float64(4), int64(1), object(2)
memory usage: 33.1+ MB
```

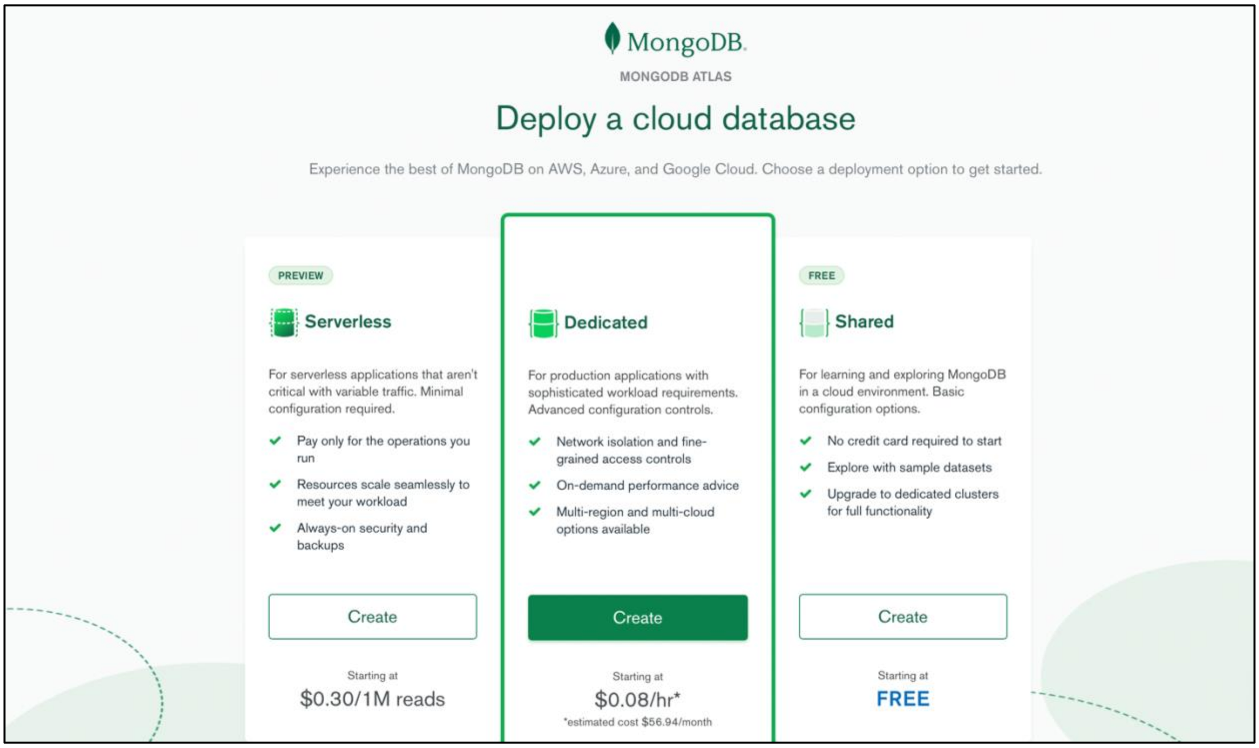
```
In [7]: data['Name'].value_counts()

Out[7]: C       1259
        TAP     1259
        PSX     1259
        FISV    1259
        EQR     1259
                ...
        DXC      215
        BHGE     152
        BHF      143
        DWDP     109
        APTV      44
        Name: Name, Length: 505, dtype: int64
```

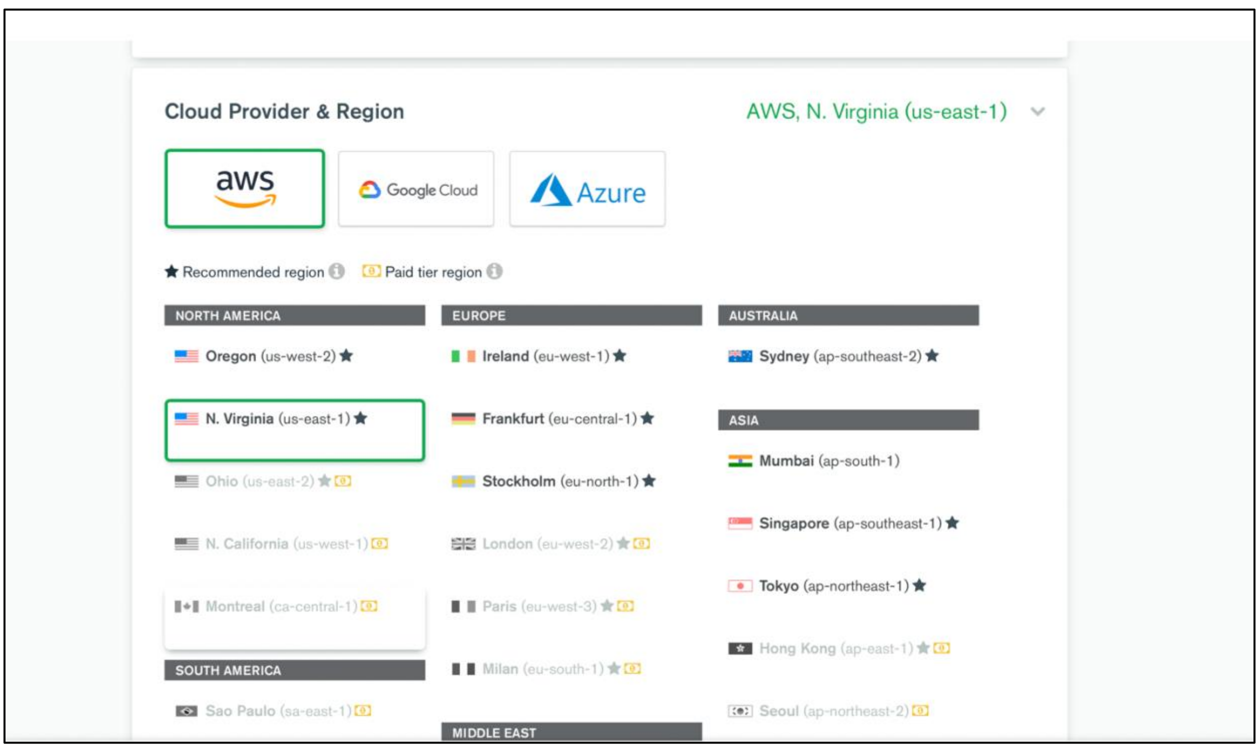
Creating Mongo DB cluster, database, and collection:

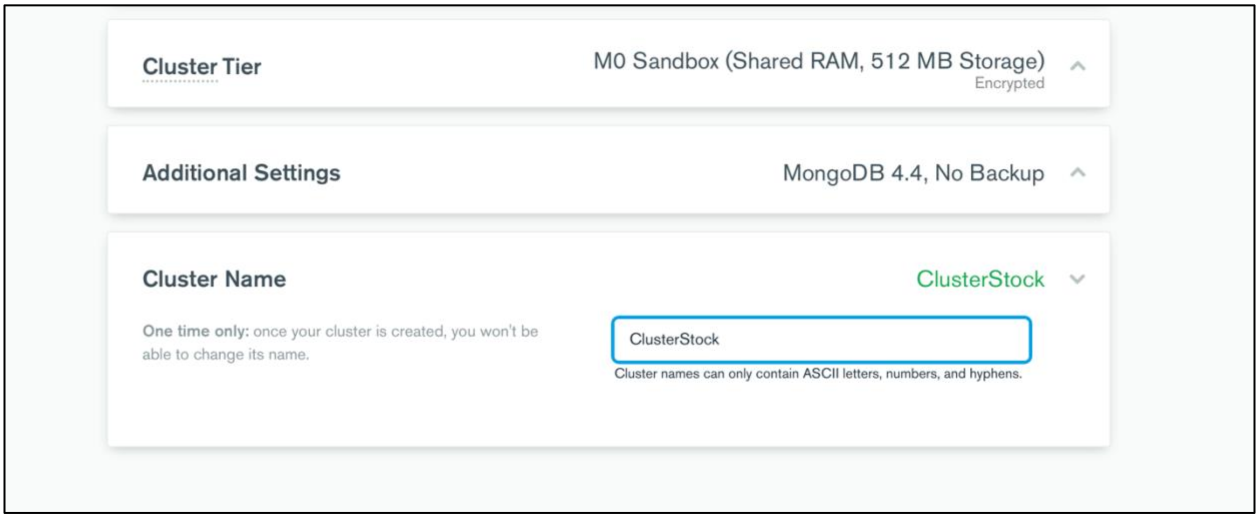
Here, we first go to the MongoDB Atlas page, in that select the shared option for the cloud database option which Is free. Select the cloud provider, here I have chosen AWS. Later, select the region, which is closest to our location, I have taken N. Virginia. Under the cluster options select the cluster tier as M0 sandbox with shared RAM and 512 MB storage. The version of Mongo DB selected is 4.4. Finally, give the cluster name and get the cluster instance started. It will take some time to deploy the cluster.

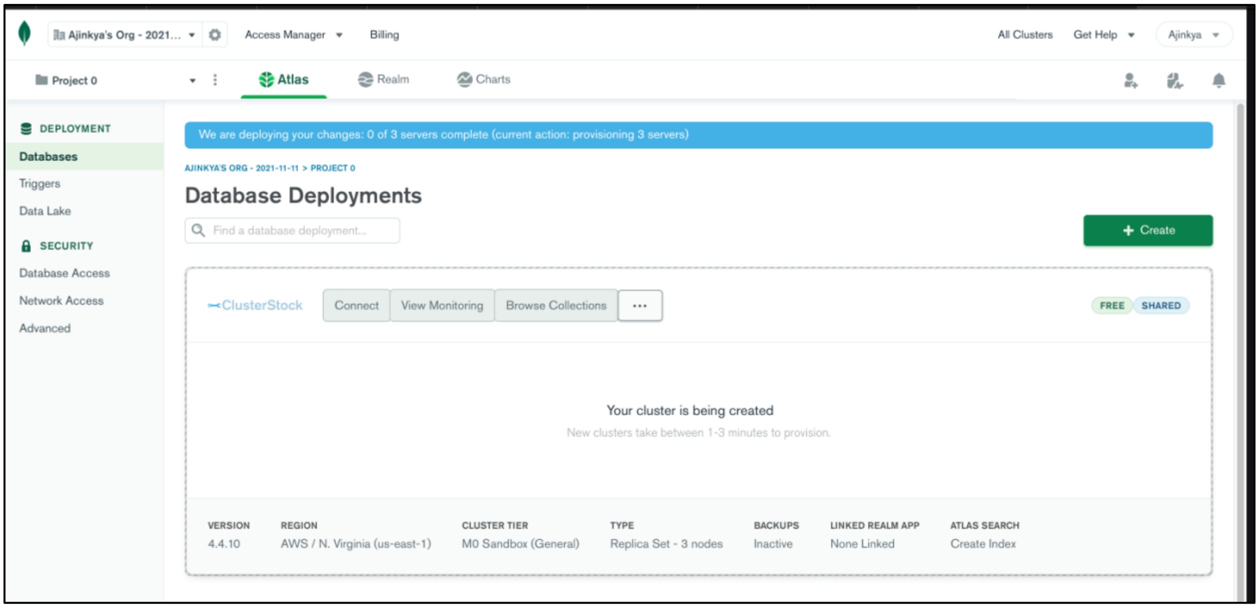
After the cluster is created, set the IP address as your local IP or set it to universal so that it can be accessed from anywhere. Give the username and password so that the database user can be created. Create the database by giving the database name and collection name.

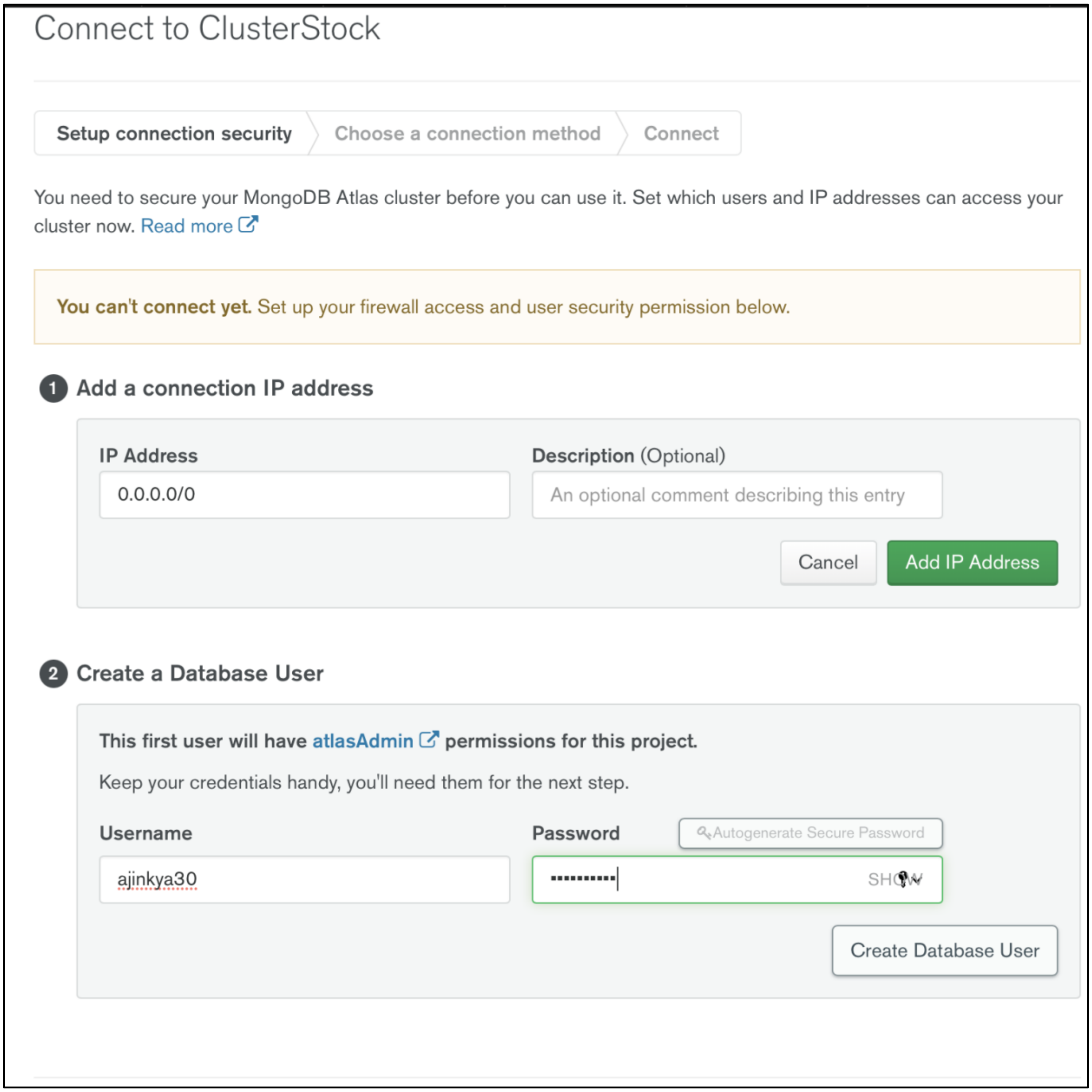
Big Data Project Report – Ajinkya Pawale

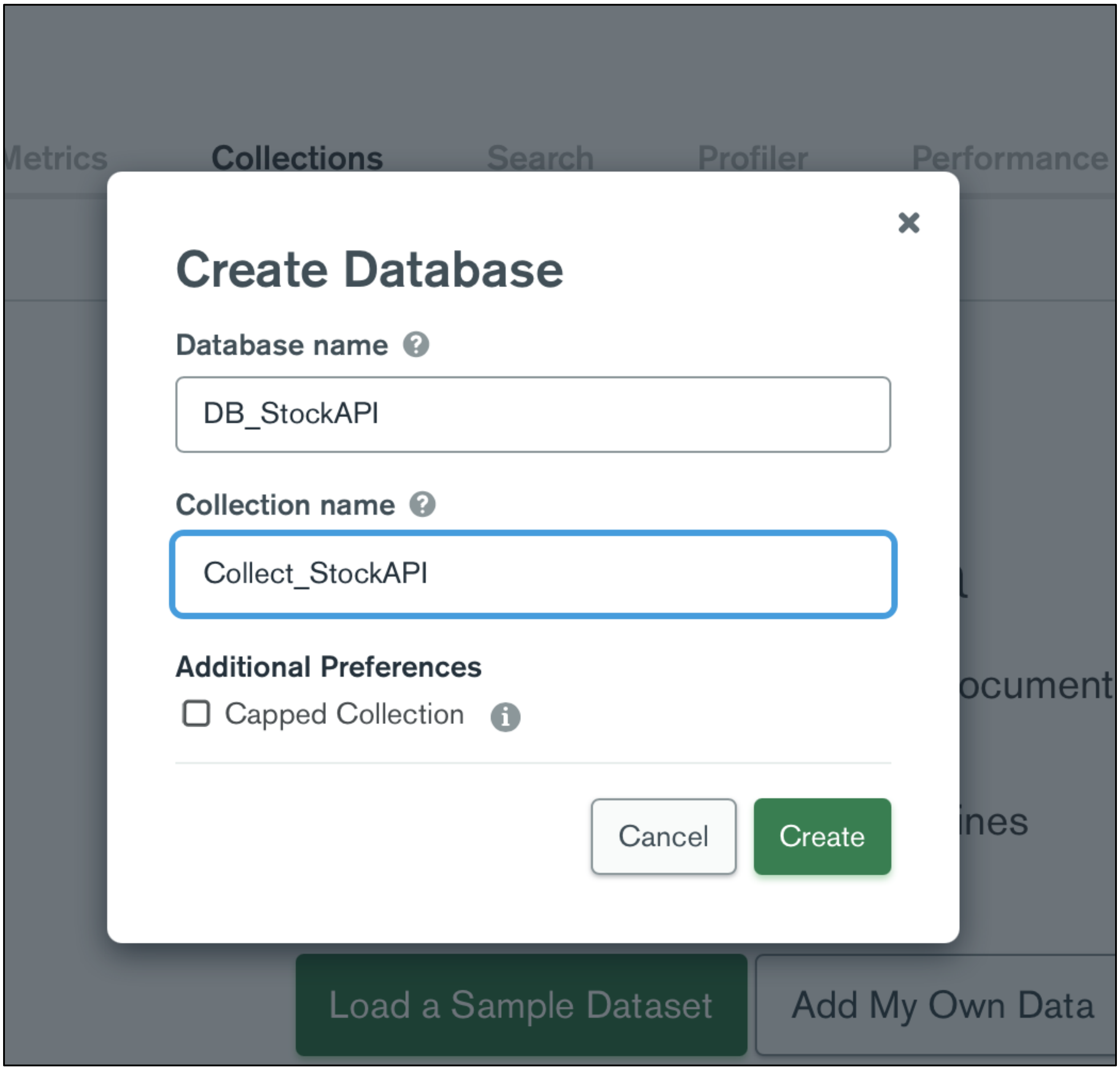





Cluster Tier
M0 Sandbox (Shared RAM, 512 MB Storage)
Encrypted
Additional Settings
MongoDB 4.4, No Backup
Cluster Name
ClusterStock
One time only: once your cluster is created, you won't be able to change its name.
ClusterStock
Cluster names can only contain ASCII letters, numbers, and hyphens.


Ajinkya's Org - 2021...
Access Manager
Billing
All Clusters
Get Help
Ajinkya
Project 0
Atlas
Realm
Charts
DEPLOYMENT
Databases
Triggers
Data Lake
SECURITY
Database Access
Network Access
Advanced
We are deploying your changes: 0 of 3 servers complete (current action: provisioning 3 servers)
AJINKYA'S ORG - 2021-11-11 > PROJECT 0
Database Deployments
Find a database deployment...
+ Create
ClusterStock
Connect
View Monitoring
Browse Collections
FREE
SHARED
Your cluster is being created
New clusters take between 1-3 minutes to provision.
VERSION
4.4.10
REGION
AWS / N. Virginia (us-east-1)
CLUSTER TIER
M0 Sandbox (General)
TYPE
Replica Set - 3 nodes
BACKUPS
Inactive
LINKED REALM APP
None Linked
ATLAS SEARCH
Create Index


Connect to ClusterStock
Setup connection security
Choose a connection method
Connect
You need to secure your MongoDB Atlas cluster before you can use it. Set which users and IP addresses can access your cluster now. Read more
You can't connect yet. Set up your firewall access and user security permission below.
1 Add a connection IP address
IP Address
0.0.0.0/0
Description (Optional)
An optional comment describing this entry
Cancel
Add IP Address
2 Create a Database User
This first user will have atlasAdmin permissions for this project.
Keep your credentials handy, you'll need them for the next step.
Username
ajinkya30
Password
Autogenerate Secure Password
Create Database User

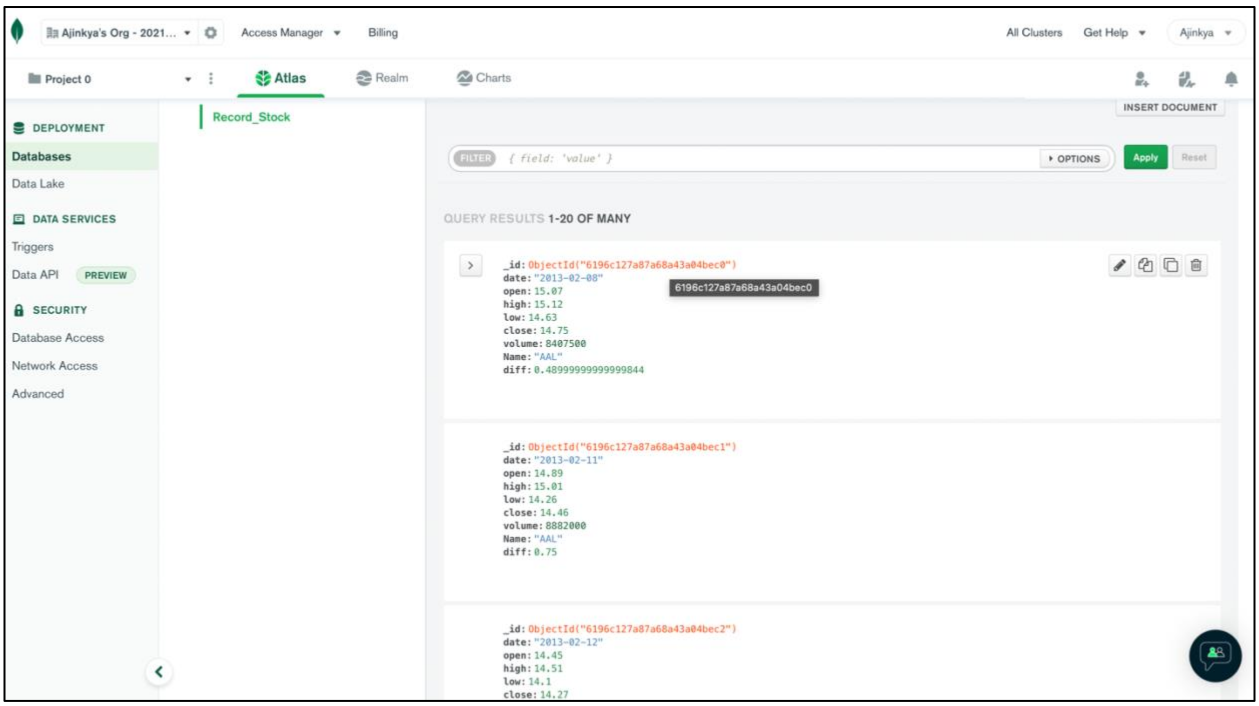

Metrics
Collections
Search
Profiler
Performance
Create Database
Database name
DB_StockAPI
Collection name
Collect_StockAPI
Additional Preferences
Capped Collection
Cancel
Create
Load a Sample Dataset
Add My Own Data

Big Data Project Report – Ajinkya Pawale

Adding the data to MongoDB using pymongo connector:

In this, using pymongo, a connection is established with the Mongo DB collection created in the above steps. After that the records are inserted in the form of dictionary from the data frame that has the data imported from the csv file. The records are inserted in the collections field of the Mongo DB database.

```
In [129]: client =  MongoClient("mongodb+srv://ajinkya30:"+quote('Ajinkya@30')+"@clusterstock.jw7kb.mongodb.net/DB_Stock?retry
          db = client['DB_Stock']
          col = db['Record_Stock']
          #data.reset_index(inplace=True)
          data_dict = data.to_dict("records")
          # Insert collection
          col.insert_many(data_dict)

Out[129]: <pymongo.results.InsertManyResult at 0x7f8c94c0c5c0>

In [70]: list(col.find())[:10]

Out[70]: [{'_id': ObjectId('6196af97a87a68a43adef63b'),
           'date': '2013-02-08',
           'open': 15.07,
           'high': 15.12,
           'low': 14.63,
           'close': 14.75,
           'volume': 8407500,
           'Name': 'AAL'},
          {'_id': ObjectId('6196af97a87a68a43adef63c'),
           'date': '2013-02-11',
           'open': 14.89,
           'high': 15.01,
           'low': 14.26,
           'close': 14.46,
           'volume': 8882000,
           'Name': 'AAL'},
          {'_id': ObjectId('6196af97a87a68a43adef63d'),
           'date': '2013-02-12',
           'open': 14.45,
```



Big Data Project Report – Ajinkya Pawale

Creating data stores for FANG stocks and adding them to a Data Lake within Mongo DB:

In this section, I create 5 different data stores for FANG stock tickers i.e., Facebook, Apple, Netflix, and Google. They are exported to Mongo DB collections with the help of pymongo. Later, all the 5 data stores are combined to form a Data Lake which consists of data from all the FANG stocks. A data lake is a collection of data from several sources. It allows you to accept and store any data to evaluate or respond to it later; data is gathered in real-time from many sources and fed into the data lake in its original format; raw data is stored using low-cost storage choices; data may be updated in real-time or in batches. It provides unlimited scalability, flexibility, integration with ML and also supports advance algorithms.

```
 'updatedExisting': True}
```

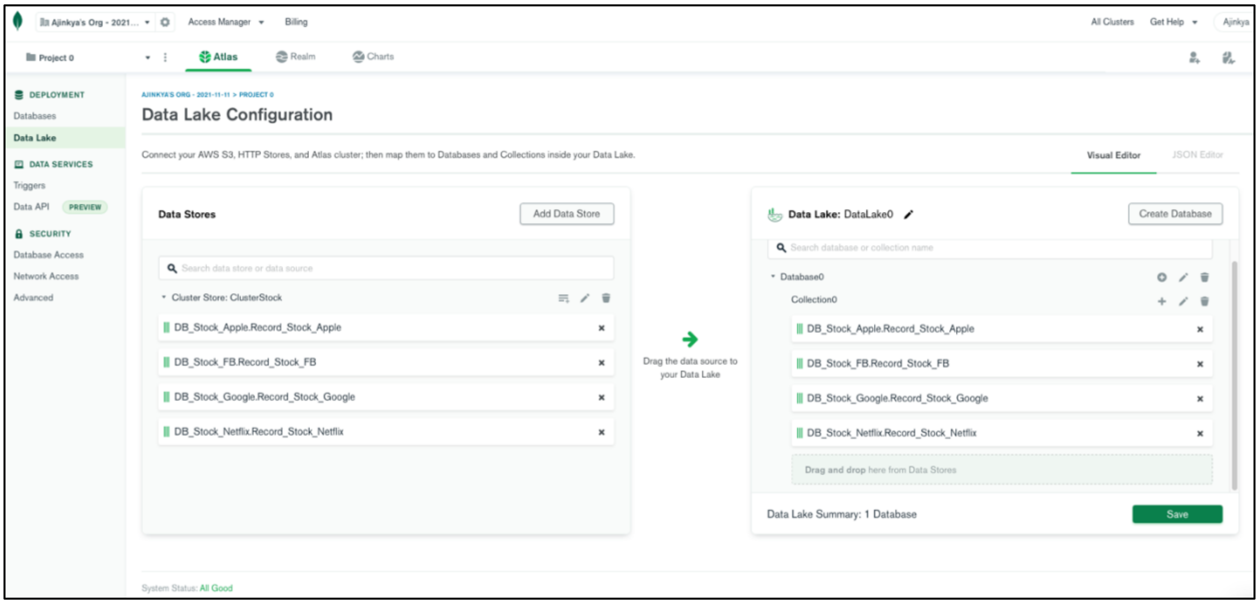
Creating 5 Dbs for FANG stocks

```
In [165]: data_fb = data[data['Name']=='FB']

In [166]: data_fb = data_fb.reset_index(drop=True)

In [167]: data_fb
```

0	2013-02-08	28.89	29.1700	28.51	28.5450	37662614	FB	0.6600
1	2013-02-11	28.61	28.6800	28.04	28.2600	36979533	FB	0.6400
2	2013-02-12	27.67	28.1600	27.10	27.3700	93417215	FB	1.0600
3	2013-02-13	27.36	28.3200	27.31	27.9075	50100805	FB	1.0100
4	2013-02-14	28.02	28.6300	28.01	28.5000	35581045	FB	0.6200
...	...	...	...	...	...	...	...	...
1254	2018-02-01	188.22	195.3200	187.89	193.0900	54211293	FB	7.4300
1255	2018-02-02	192.04	194.2100	189.98	190.2800	26677484	FB	4.2300
1256	2018-02-05	186.93	190.6100	180.61	181.2600	33128206	FB	10.0000
1257	2018-02-06	178.57	185.7700	177.74	185.3100	37758505	FB	8.0300
1258	2018-02-07	184.15	185.0817	179.95	180.1800	27601886	FB	5.1317

```
In [168]: client =  MongoClient("mongodb+srv://ajinkya30:"+quote('Ajinkya@30')+"@clusterstock.jw7kb.mongodb.net/DB_Stock?retry
          db_fb = client['DB_Stock_FB']
          col_fb = db_fb['Record_Stock_FB']

          data_dict = data_fb.to_dict("records")
          col_fb.insert_many(data_dict)

Out[168]: <pymongo.results.InsertManyResult at 0x7f8c4610ce40>
```

Big Data Project Report – Ajinkya Pawale



Big Data Project Report – Ajinkya Pawale

Building a Spark session and importing the data from Mongo DB using Spark Connector and doing analysis on the data:

Here the data is imported from Mongo DB using the spark connector. The descriptive statistics of the data is checked in order to preprocess the data. Null values and NAN values are checked in order to remove them from the data. Since the name column is in the form of string it can't be passed to the model without encoding, so the Name column is encoded using a unique encoder which converts the column into numeric values. Similar encoding is done for the date column. NA and NAN values are dropped from the dataset. Finally, we convert all the variables except the target variable into a single vector as PySpark takes the input as a vector of samples. Also, min max scaler is applied to convert the variables to the same scale.

```
In [3]: spark = SparkSession\
            .builder\
            .master('local')\
            .config('spark.mongodb.input.uri', 'mongodb+srv://ajinkya30:'+quote('Ajinkya@30')+'@clusterstock.jw7kb.mongodb.n
            .config('spark.mongodb.output.uri', 'mongodb+srv://ajinkya30:'+quote('Ajinkya@30')+'@clusterstock.jw7kb.mongodb.
            .config('spark.jars.packages', 'org.mongodb.spark:mongo-spark-connector_2.12:3.0.1')\
            .getOrCreate()

In [4]: spark

Out[4]: SparkSession - in-memory
        SparkContext

        Spark UI
        Version
        v3.2.0
        Master
        local
        AppName
        pyspark-shell

In [103]: df = spark.read\
              .format("com.mongodb.spark.sql.DefaultSource")\
              .option("database","DB_Stock")\
              .option("collection", "Record_Stock")\
              .load()
```

```
In [63]: df.show(5)

+----+--------------------+-----+----------+-------------------+-----+-----+-----+--------+
|Name|                 _id|close|      date|               diff| high|  low| open|  volume|
+----+--------------------+-----+----------+-------------------+-----+-----+-----+--------+
| AAL|{6196c127a87a68a4...|14.75|2013-02-08|0.48999999999999844|15.12|14.63|15.07| 8407500|
| AAL|{6196c127a87a68a4...|14.46|2013-02-11|               0.75|15.01|14.26|14.89| 8882000|
| AAL|{6196c127a87a68a4...|14.27|2013-02-12|0.41000000000000014|14.51| 14.1|14.45| 8126000|
| AAL|{6196c127a87a68a4...|14.66|2013-02-13| 0.6899999999999995|14.94|14.25| 14.3|10259500|
| AAL|{6196c127a87a68a4...|13.99|2013-02-14| 1.8000000000000007|14.96|13.16|14.94|31879900|
+----+--------------------+-----+----------+-------------------+-----+-----+-----+--------+
only showing top 5 rows
```

In [15]:
```
df.dtypes
```

Out[15]:
```
[('Name', 'string'),
 ('_id', 'struct<oid:string>'),
 ('close', 'double'),
 ('date', 'string'),
 ('diff', 'double'),
 ('high', 'double'),
 ('low', 'double'),
 ('open', 'double'),
 ('volume', 'int')]
```

See the decsriptive statistics

In [67]:
```
df.describe().toPandas()
```

Out[67]:

	summary	Name	close	date	diff	high	low	open	volume
0	count	619040	619040	619040	619040	619040	619040	619040	619040
1	mean	None	83.04376276476455	None	NaN	NaN	NaN	NaN	4321823.395568945
2	stddev	None	97.38974800165754	None	NaN	NaN	NaN	NaN	8693609.511967564
3	min	A	1.59	2013-02-08	-0.25500000000000256	1.69	1.5	1.62	0
4	max	ZTS	2049.0	2018-02-07	NaN	NaN	NaN	NaN	618237630

To check null values

In [73]:
```
dataset.select([count(when(isnull(c), c)).alias(c) for c in dataset]).show()
```

```
+----+----+----+-----+----+---+------+----+
|date|Name|open|close|high|low|volume|diff|
+----+----+----+-----+----+---+------+----+
|   0|   0|   0|    0|   0|  0|     0|   0|
+----+----+----+-----+----+---+------+----+
```

In [92]:
```
dataset_1 = dataset.select(col('open'),col('close'),col('high'),col('low'),col('volume'),col('diff'))
```

To check the NAN values

In [81]:
```
dataset.select([count(when(isnan(c), c)).alias(c) for c in dataset_1.columns]).show()
```

```
+----+-----+----+---+------+----+
|open|close|high|low|volume|diff|
+----+-----+----+---+------+----+
|  11|    0|   8|  8|     0|   8|
+----+-----+----+---+------+----+
```

Encoding the name column

In [106]:
```
dataset = StringIndexer(
    inputCol='Name',
    outputCol='Stock_Abbrv',
    handleInvalid='keep').fit(dataset).transform(dataset)
```

In [107]:
```
dataset.show()
```

```
+----------+----+-----+-----+-----+-----+--------+-------------------+-----------+
|      date|Name| open|close| high|  low|  volume|               diff|Stock_Abbrv|
+----------+----+-----+-----+-----+-----+--------+-------------------+-----------+
|2013-02-08| AAL|15.07|14.75|15.12|14.63| 8407500|0.48999999999999844|        1.0|
|2013-02-11| AAL|14.89|14.46|15.01|14.26| 8882000|               0.75|        1.0|
|2013-02-12| AAL|14.45|14.27|14.51| 14.1| 8126000|0.41000000000000014|        1.0|
|2013-02-13| AAL| 14.3|14.66|14.94|14.25|10259500| 0.6899999999999995|        1.0|
|2013-02-14| AAL|14.94|13.99|14.96|13.16|31879900| 1.8000000000000007|        1.0|
|2013-02-15| AAL|13.93| 14.5|14.61|13.93|15628000| 0.6799999999999997|        1.0|
|2013-02-19| AAL|14.33|14.26|14.56|14.08|11354400| 0.4800000000000004|        1.0|
|2013-02-20| AAL|14.17|13.33|14.26|13.15|14725200| 1.1099999999999994|        1.0|
|2013-02-21| AAL|13.62|13.37|13.95| 12.9|11922100|  1.049999999999999|        1.0|
|2013-02-22| AAL|13.57|13.57| 13.6|13.21| 6071400| 0.3899999999999988|        1.0|
|2013-02-25| AAL| 13.6|13.02|13.76| 13.0| 7186400| 0.7599999999999998|        1.0|
|2013-02-26| AAL|13.14|13.26|13.42| 12.7| 9419000| 0.7200000000000006|        1.0|
|2013-02-27| AAL|13.28|13.41|13.62|13.18| 7390500| 0.4399999999999995|        1.0|
|2013-02-28| AAL|13.49|13.43|13.63|13.39| 6143600| 0.2400000000000002|        1.0|
|2013-03-01| AAL|13.37|13.61|13.95|13.32| 7376800|  0.629999999999999|        1.0|
|2013-03-04| AAL| 13.5| 13.9|14.07|13.47| 8174800| 0.5999999999999996|        1.0|
|2013-03-05| AAL|14.01|14.05|14.05|13.71| 7676100|0.33999999999999986|        1.0|
|2013-03-06| AAL|14.52|14.57|14.68|14.25|13243200| 0.4299999999999997|        1.0|
|2013-03-07| AAL| 14.7|14.82|14.93| 14.5| 9125300| 0.4299999999999997|        1.0|
|2013-03-08| AAL|14.99|14.92| 15.2|14.84|10593700|0.35999999999999943|        1.0|
+----------+----+-----+-----+-----+-----+--------+-------------------+-----------+
only showing top 20 rows
```

Converting the date column in to a numeric encoded entry

In [109]:
```
dataset=dataset.withColumn("dateAsInteger", F.unix_timestamp(dataset['date']))
```

In [110]:
```
dataset = dataset.drop('date')
dataset.show()
```

```
+-----+-----+-----+-----+--------+-------------------+-----------+-------------+
| open|close| high|  low|  volume|               diff|Stock_Abbrv|dateAsInteger|
+-----+-----+-----+-----+--------+-------------------+-----------+-------------+
|15.07|14.75|15.12|14.63| 8407500|0.48999999999999844|        1.0|   1360299600|
|14.89|14.46|15.01|14.26| 8882000|               0.75|        1.0|   1360558800|
|14.45|14.27|14.51| 14.1| 8126000|0.41000000000000014|        1.0|   1360645200|
| 14.3|14.66|14.94|14.25|10259500| 0.6899999999999995|        1.0|   1360731600|
|14.94|13.99|14.96|13.16|31879900| 1.8000000000000007|        1.0|   1360818000|
|13.93| 14.5|14.61|13.93|15628000| 0.6799999999999997|        1.0|   1360904400|
|14.33|14.26|14.56|14.08|11354400| 0.4800000000000004|        1.0|   1361250000|
|14.17|13.33|14.26|13.15|14725200| 1.1099999999999994|        1.0|   1361336400|
|13.62|13.37|13.95| 12.9|11922100|  1.049999999999999|        1.0|   1361422800|
|13.57|13.57| 13.6|13.21| 6071400| 0.3899999999999988|        1.0|   1361509200|
| 13.6|13.02|13.76| 13.0| 7186400| 0.7599999999999998|        1.0|   1361768400|
|13.14|13.26|13.42| 12.7| 9419000| 0.7200000000000006|        1.0|   1361854800|
|13.28|13.41|13.62|13.18| 7390500| 0.4399999999999995|        1.0|   1361941200|
|13.49|13.43|13.63|13.39| 6143600| 0.2400000000000002|        1.0|   1362027600|
|13.37|13.61|13.95|13.32| 7376800|  0.629999999999999|        1.0|   1362114000|
| 13.5| 13.9|14.07|13.47| 8174800| 0.5999999999999996|        1.0|   1362373200|
|14.01|14.05|14.05|13.71| 7676100|0.33999999999999986|        1.0|   1362459600|
|14.52|14.57|14.68|14.25|13243200| 0.4299999999999997|        1.0|   1362546000|
| 14.7|14.82|14.93| 14.5| 9125300| 0.4299999999999997|        1.0|   1362632400|
|14.99|14.92| 15.2|14.84|10593700|0.35999999999999943|        1.0|   1362718800|
+-----+-----+-----+-----+--------+-------------------+-----------+-------------+
only showing top 20 rows
```

```
In [115]: dataset = dataset.na.drop()
          dataset.show()

+-----+-----+-----+-----+--------+------------------+-----------+-------------+
| open|close| high|  low|  volume|              diff|Stock_Abbrv|dateAsInteger|
+-----+-----+-----+-----+--------+------------------+-----------+-------------+
|15.07|14.75|15.12|14.63| 8407500|0.4899999999999844|        1.0|   1360299600|
|14.89|14.46|15.01|14.26| 8882000|              0.75|        1.0|   1360558800|
|14.45|14.27|14.51| 14.1| 8126000|0.4100000000000014|        1.0|   1360645200|
| 14.3|14.66|14.94|14.25|10259500| 0.6899999999999995|        1.0|   1360731600|
|14.94|13.99|14.96|13.16|31879900| 1.8000000000000007|        1.0|   1360818000|
|13.93| 14.5|14.61|13.93|15628000| 0.6799999999999997|        1.0|   1360904400|
|14.33|14.26|14.56|14.08|11354400| 0.4800000000000004|        1.0|   1361250000|
|14.17|13.33|14.26|13.15|14725200| 1.1099999999999994|        1.0|   1361336400|
|13.62|13.37|13.95| 12.9|11922100|  1.049999999999999|        1.0|   1361422800|
|13.57|13.57| 13.6|13.21| 6071400| 0.3899999999999988|        1.0|   1361509200|
| 13.6|13.02|13.76| 13.0| 7186400| 0.7599999999999998|        1.0|   1361768400|
|13.14|13.26|13.42| 12.7| 9419000| 0.7200000000000006|        1.0|   1361854800|
|13.28|13.41|13.62|13.18| 7390500| 0.4399999999999995|        1.0|   1361941200|
|13.49|13.43|13.63|13.39| 6143600| 0.2400000000000002|        1.0|   1362027600|
|13.37|13.61|13.95|13.32| 7376800|  0.629999999999999|        1.0|   1362114000|
| 13.5| 13.9|14.07|13.47| 8174800| 0.5999999999999996|        1.0|   1362373200|

In [116]: dataset.select([count(when(isnan(c), c)).alias(c) for c in dataset.columns]).show()

+----+-----+----+---+------+----+-----------+-------------+
|open|close|high|low|volume|diff|Stock_Abbrv|dateAsInteger|
+----+-----+----+---+------+----+-----------+-------------+
|   0|    0|   0|  0|     0|   0|          0|            0|
+----+-----+----+---+------+----+-----------+-------------+

In [146]: dataset = dataset.drop('Stock_Abbrv')
```

Converting the variables into a single target variable and scaling them using Min Max Scaler

```
In [198]: vecAssembler = VectorAssembler(inputCols=dataset.columns, outputCol="features",handleInvalid='skip')
          normalizer = MinMaxScaler(inputCol="features", outputCol="scaledFeatures",min=0,max=1)
          pipeline_normalize = Pipeline(stages=[vecAssembler,normalizer])
          df_transf = pipeline_normalize.fit(dataset).transform(dataset)
          #%%
          to_array = F.udf(lambda x: (x.toArray().tolist()), ArrayType(DoubleType())) # IF spark 3.0.0 then a api - vectortoar
          df_array = df_transf.withColumn("scaledFeatures",to_array("scaledFeatures"))
          #%%\
          df_final= df_array.select([F.col('scaledFeatures')[i].alias(dataset.columns[i]) for i in range(len(dataset.columns))

In [199]: df_final.show()

+--------------------+--------------------+--------------------+--------------------+--------------------+---------
-----------+--------------------+
|                open|               close|                high|                 low|              volume|
diff|       dateAsInteger|
+--------------------+--------------------+--------------------+--------------------+--------------------+---------
-----------+--------------------+
|0.006585454224972825|0.006427632960667379|0.006499540241010503|0.006456498542001663|  0.0135989787187442| 0.005378
47886510487|                 0.0|
|0.006497321752073561|0.006285990592993098|0.006446304989594929|0.006274556084991715|0.014366483080324295|0.0072555
31891852887|0.001643835616438356|
|0.006281886818319802|0.006193190421068569|0.006204326574069594| 0.00619587826574417|0.013143652105920604|0.0048009
24087643957|0.002191780821917808|
|0.006208443090903749|0.006383674984492603|0.006412428011421...|0.006269638721288743|0.016594591105775464| 0.006822
36580875719|0.002739726027397...|
|0.006521802994545579|0.006056432272969263|0.006422107148042396|0.005733646077664843| 0.05156561597217435|0.0148359
38346027531|0.003287671232876712|
|0.006027281896610816|0.006305527471292...|0.006252722257174...|0.006112283082793653| 0.02527814677520167|0.0067501
71461574...|0.003835616438356...|
| 0.00622313183638696|0.006188306201493594|0.006228524415622128|0.006186043538338227|0.018365593266985253|0.0053062
84517922269|0.006027397260273973|
```

Big Data Project Report – Ajinkya Pawale

Results:

Adding new documents, updating, and deleting them from the collection of Mongo DB connection:

In this data is added, updated, and deleted from the collection of Mongo DB with the help of the pymongo connector. A new field is also added to the entire document set, which is the difference column i.e., the high – low price of the stock. This column is added to all the document just one line of code, which establishes the flexibility of Mongo DB.

Add new documents, update and delete them

```
In [111]: client =  MongoClient("mongodb+srv://ajinkya30:"+quote('Ajinkya@30')+"@clusterstock.jw7kb.mongodb.net/DB_Stock?retry
          db = client['DB_Stock']
          col = db['Record_Stock']

In [39]: col

Out[39]: Collection(Database(MongoClient(host=['clusterstock-shard-00-02.jw7kb.mongodb.net:27017', 'clusterstock-shard-00-01
         .jw7kb.mongodb.net:27017', 'clusterstock-shard-00-00.jw7kb.mongodb.net:27017'], document_class=dict, tz_aware=False
         , connect=True, retrywrites=True, w='majority', authsource='admin', replicaset='atlas-13yk0z-shard-0', ssl=True), '
         DB_STOCKAPI'), 'DB_STOCKAPI')

In [40]: client.list_database_names()

Out[40]: ['DB_STOCKAPI', 'DB_Stock', 'admin', 'local']

In [44]: lis = [{'date':"2013-02-08",'open':15.07,'high':15.12,'low':14.63,'close':14.75,'volume':8407500,'Name':"AJN"},
               {'date':"2013-02-09",'open':16.07,'high':18.52,'low':12.83,'close':14.21,'volume':4404800,'Name':"AJN"}]

In [41]: x = col.insert_many(lis)
         print(x)

In [47]: dic = {'date':"2013-02-09",'open':16.07,'high':18.52,'low':12.83,'close':14.21,'volume':4404800,'Name':"AJN"}

In [48]: y = col.delete_one(dic)
         print(y)

         <pymongo.results.DeleteResult object at 0x7f8d30456880>
```

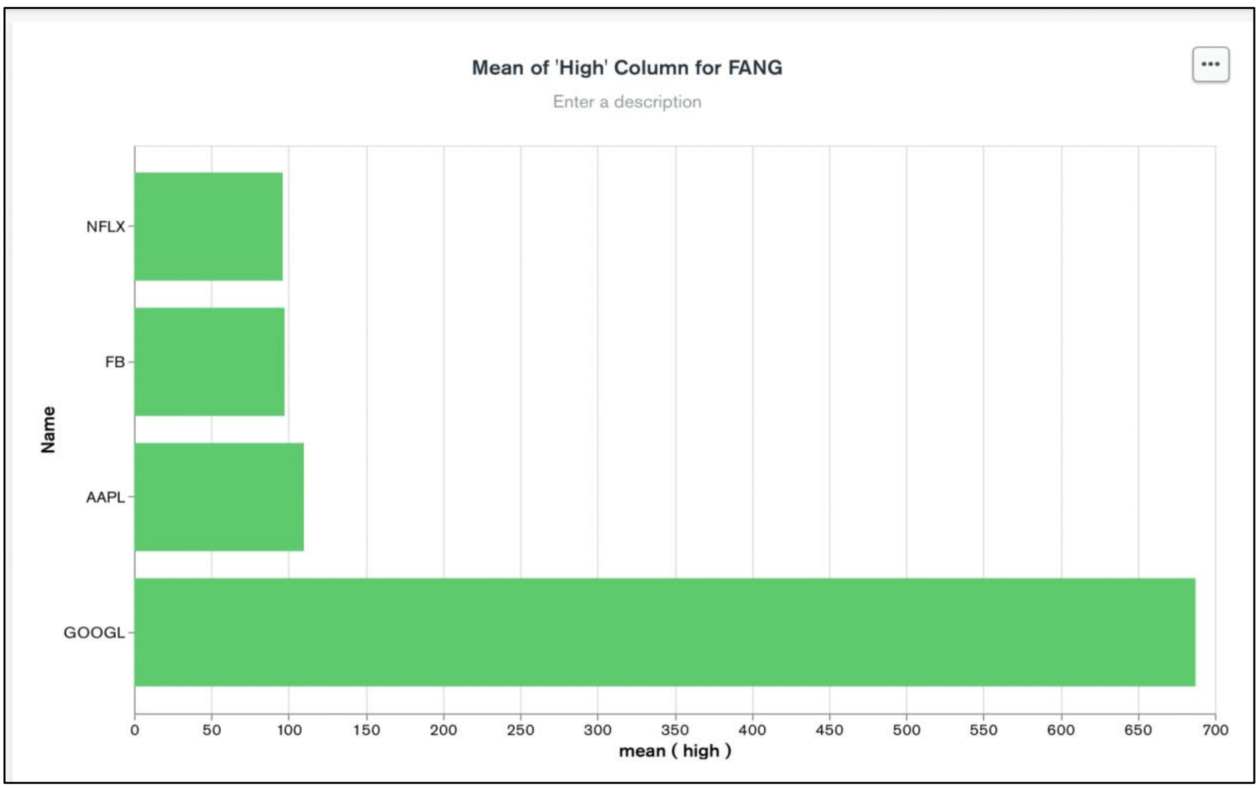
```
In [95]: record_update = {
             'date': '2013-02-08',
              'Name' : 'AAL'
         }

In [98]: col.update_one({'open':15.09},{'$set':record_update})

Out[98]: <pymongo.results.UpdateResult at 0x7f8c62bb6bc0>

In [99]: col.update_one({'close':14.46},{'$set':record_update})

Out[99]: <pymongo.results.UpdateResult at 0x7f8c636143c0>
```

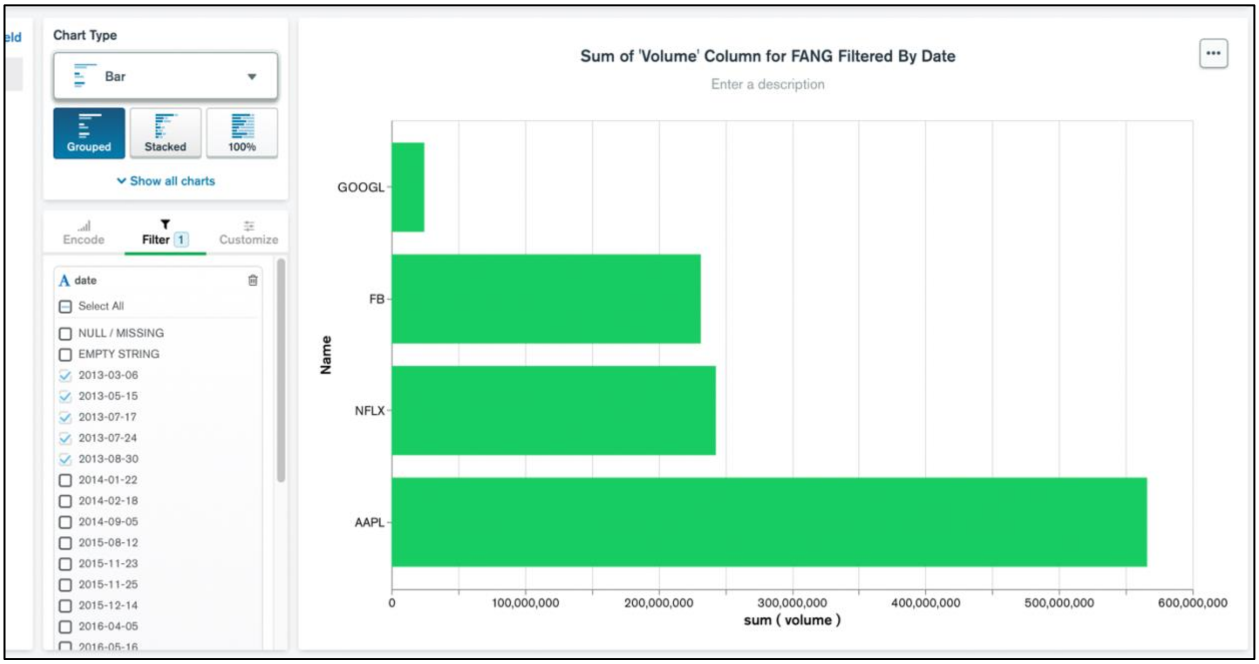
Add a new field to all the documents

```
In [124]: data['diff'] = data['high'] - data['low']

In [125]: diff = data['diff'].tolist()

In [128]: col.update({},{'$set' : {"high-low": diff}},False,True)

<ipython-input-128-80ec125eeb53>:1: DeprecationWarning: update is deprecated. Use replace_one, update_one or update_many instead.
  col.update({},{'$set' : {"high-low": diff}})

Out[128]: {'n': 1,
 'nModified': 1,
 'opTime': {'ts': Timestamp(1637269630, 8), 't': 217},
 'electionId': ObjectId('7fffffff00000000000000d9'),
 'ok': 1.0,
 '$clusterTime': {'clusterTime': Timestamp(1637269630, 10),
  'signature': {'hash': b'\xb5f\t\xfe\xd3\xd0\xd5n\xd1\xd9\xa8\x15\xef<\xb9+\xc1\xe3}\xa2',
   'keyId': 6993535964366766082}},
 'operationTime': Timestamp(1637269630, 8),
 'updatedExisting': True}
```

Creating visualizations under the Mongo DB portal in Atlas to analyze:

1) Mean of High stock value for FANG tickers
2) Sum of Volume for FANG tickers





Connecting to Data Lake that we created above and querying various aggregation and filtered results from it:

Here the Data Lake is connected using pymongo and it is used to query different use cases such as filtering specific stock tickers, applying conditional filters with limit and also using aggregate function to sum up certain fields in order to get the totals across it.

Connection to Data Lake and Quering results from Data Lake

```
In [213]: client =  MongoClient('mongodb://ajinkya30:'+quote('Ajinkya@30')+'@datalake0-jw7kb.a.query.mongodb.net/Database0?ssl
          db_lake = client['Database0']
          col_lake = db_lake['Collection0']

In [214]: col_lake

Out[214]: Collection(Database(MongoClient(host=['datalake0-jw7kb.a.query.mongodb.net:27017'], document_class=dict, tz_aware=F
          alse, connect=True, ssl=True, authsource='admin'), 'Database0'), 'Collection0')

In [221]: cur = list(col_lake.find({"Name": "FB"}).limit(5))

In [222]: cur

Out[222]: [{'_id': ObjectId('6197d041a87a68a43a0e44b1'),
            'date': '2013-03-26',
            'open': 25.08,
            'high': 25.48,
            'low': 25.03,
            'close': 25.205,
            'volume': 26939599,
            'Name': 'FB',
            'diff': 0.4499999999999993},
```

```
In [239]: cur = list(col_lake.find({"Name" : 'FB','diff':{'$gt':8}}).limit(5))

In [240]: cur
Out[240]: [{'_id': ObjectId('6197d041a87a68a43a0e4845'),
            'date': '2016-11-10',
            'open': 123.93,
            'high': 124.18,
            'low': 115.27,
            'close': 120.8,
            'volume': 67846704,
            'Name': 'FB',
            'diff': 8.91000000000001},
```

```
          'diff': 10.0}]

In [248]: cur = list(col_lake.aggregate([{'$match': {"diff" :{'$gt': 5}, "Name": 'AAPL'} },{'$count': "Num"}]))

In [249]: cur
Out[249]: [{'Num': 19}]
```

Modeling the data using Machine Learning model Random Forest Regressor to predict stock prices using PySpark:

The examples below load a MongoDB-formatted and preprocessed dataset, partition it into training and test sets, train on the first dataset, and then evaluate on the held-out test set. To index categorical features, I employ a feature transformer, which adds information to the Data Frame that tree-based algorithms can understand. The target label here is the High value of the stock. Thus, Random Forest Regressor is used as it falls under the regression category. The evaluation matrix used is RMSE and accuracy, for our case the RMSE comes to 0.05 and accuracy is 85 % which is pretty good for the random forest model with baseline parameters.

Modeling

```
In [217]: featureIndexer =\
              VectorIndexer(inputCol="features", outputCol="indexedFeatures", maxCategories=4).fit(transformed_data)

In [218]: (training_data, test_data) = transformed_data.randomSplit([0.8,0.2])

In [219]: training_data.show()

+--------------------+--------------------+--------------------+--------------------+--------------------+---------
-----------+-------------------+--------------------+
|                open|               close|                high|                 low|              volume|
diff|       dateAsInteger|            features|
+--------------------+--------------------+--------------------+--------------------+--------------------+---------
-----------+-------------------+--------------------+
|2.448124247201785E-5|1.465265872492565E-5|4.839568310506708E-6|5.409100073268724E-5|0.019274738981431196|0.0024907
04977800256| 0.4925799086757991|[0.49257990867579...|
|2.448124247201785E-5|3.907375659980162...|                 0.0| 7.37604555445734E-5|0.011330955613986999|0.0021297
33241887179|  0.527648401826484|[0.52764840182648...|
|2.937749096642144...|6.349485447467771E-5|1.935827324202683E-5|7.867781924754497E-5|0.018193007496961577|0.0023463
16283435026| 0.5249086757990867|[0.52490867579908...|
|3.427373946082504E-5|3.418953702482641E-5|4.839568310506708E-6|7.867781924754497E-5| 0.009208373372623260|0.0021297
33241887179| 0.5271004566210046|[0.52710045662100...|
```

```
In [220]: rf = RandomForestRegressor(featuresCol="indexedFeatures",labelCol='high')
          pipeline = Pipeline(stages=[featureIndexer, rf])

In [221]: model = pipeline.fit(training_data)

In [222]: predictions = model.transform(test_data)

In [224]: evaluator = RegressionEvaluator(
              labelCol="volume", predictionCol="prediction", metricName="rmse")
          rmse = evaluator.evaluate(predictions)
          print("Root Mean Squared Error (RMSE) on test data = ", rmse)
          accuracy = evaluator.evaluate(predictions)
          print("Accuracy on test data = s", accuracy)

          Root Mean Squared Error (RMSE) on test data =  0.05810955446086183
          Accuracy on test data =  0.8456334466473277
```

Discussion:

The reason for taking the stock data was because of its high input volume which becomes a challenging task to manage it. With the help of Mongo DB which is a No SQL data store, it becomes easy to handle the data and make changes to the schema. Its advantages include Instead of monolithic architecture, it uses efficient, scale-out architecture. The capacity to work with large amounts of organized, semi-structured, and unstructured data Having a deeper understanding of object-oriented programming. Working well with current software development approaches, such as agile sprints and frequent code pushes. It also provides easy integration with various programming languages such as Python for easy integration and manipulation of data with the help of a connection (pymongo in our case).

Also, the reason for choosing PySpark as the analysis tool was because of its fast processing of data. It becomes easy to extract data from Mongo DB and perform Machine Learning on it with the help of RDDs which are fault tolerant data objects and 100x quicker than typical Hadoop MapReduce thanks to in-memory operations.

In this project I have used concepts from modules within the coursework such as Lifecycles and Pipelines, Ingest and Storage, Modeling and Process and Analytics. Lifecycles and Pipeline includes the entire staging of data from preprocessing, collection, storage, updates, querying, analysis, modeling, and visualization. Ingest and Storage includes the Mongo DB data store used to store and update/change the large amounts of data. Modeling includes the PySpark analysis of the stock value using the Random Forest Regressor model. Finally, Process and Analytics refers to the querying done on the data and the analysis of various averages across the fields.

I faced two major difficulties while working with the project. One was adding a new field to the Mongo DB collection set using pymongo. I tried a lot of update functionalities mentioned but it was not adding the changes to the entire document. Finally, after a lot of trial and errors I was able to get the right code which update the entire document set and added the new field to the collection.

```
In [128]: col.update({},{'$set' : {"high-low": diff}},False,True)

<ipython-input-128-80ec125eeb53>:1: DeprecationWarning: update is depre
cated. Use replace_one, update_one or update_many instead.
  col.update({},{'$set' : {"high-low": diff}})

Out[128]: {'n': 1,
 'nModified': 1,
 'opTime': {'ts': Timestamp(1637269630, 8), 't': 217},
 'electionId': ObjectId('7fffffff0000000000000d9'),
 'ok': 1.0,
 '$clusterTime': {'clusterTime': Timestamp(1637269630, 10),
  'signature': {'hash': b'\xb5f\t\xfe\xd3\xd0\xd5n\xd1\xd9\xa8\x15\xef<
\xb9+\xc1\xe3}\xa2',
   'keyId': 6993535964366766082}},
 'operationTime': Timestamp(1637269630, 8),
 'updatedExisting': True}
```

The other problem that I faced was to build a spark session and establish the connection with the Mongo DB database. There were version issues between the Mongo DB connector and PySpark. Later, it needed Java 8 to work so I had to downgrade my Java version. After both the changes I was able to connect the Mongo DB using the PySpark connector.

Conclusion:

Financial organizations are constantly on the hunt for new chances to take advantage of the big data technologies before other companies in the industry. The major goal is to push the limits by looking at non-traditional data sources and use diverse types of data to obtain a competitive advantage.

Artificial intelligence and bots are commonly used in computerized trading. You'll be able to reduce the social-emotional aspect while trading with machine learning. New traders may now employ a variety of tactics to make transactions that are free of prejudice and illogical fluctuations.

The project aims to cover the following things in short, get stock data from Kaggle datasets. Using the python connector pymongo, save the data in Mongo DB as collections. Using pymongo, make modifications to the schema and insert/update/delete records. Create a data lake and use pymongo to perform queries on it. Use the data to create visuals. Extract data from Mongo DB with PySpark. Perform data cleaning and preparation. Create a machine learning model to forecast stock prices using PySpark.

Big Data Project Report – Ajinkya Pawale

References:

1) https://www.kaggle.com/camnugent/sandp500
2) https://www.kdnuggets.com/2020/04/benefits-apache-spark-pyspark.html
3) https://www.tutorialspoint.com/mongodb/mongodb_advantages.htm
4) https://www.n-ix.com/data-lake-vs-data-warehouse/
5) https://spark.apache.org/docs/latest/ml-classification-regression.html#linear-regression
6) https://towardsdatascience.com/your-first-apache-spark-ml-model-d2bb82b599dd
7) https://docs.mongodb.com/spark-connector/current/python/read-from-mongodb/
8) https://docs.mongodb.com/datalake/tutorial/run-queries/