SCALA PROJECT

Hedge Fund Application:
Real Time Risk Analysis

Team 11:
Amit Pingale
Mayank Gangrade

Goal

Building reactive application for portfolio management and risk analysis

Leveraging:

- Kafka + Spark streaming
- Spark Analysis engine
- MongoDB for maintaining historic records + batch processing

Why Real-time Big Data Pipeline Important

It is estimated that by 2020 approximately 1.7 megabytes of data will be created every second. This results in an increasing demand for real-time and streaming data analysis. For historical data analysis descriptive, prescriptive, and predictive analysis techniques are used. On the other hand, for real-time data analysis, streaming data analysis is the choice. The main benefit of real-time analysis is one can analyze and visualize the report on a real-time basis.

Technology and Tools:

1. Alpha Vantage API for Real Time Data
2. Kafka for capturing Real Time Data
3. Spark Streaming For Consuming Data
4. Spark + Scala for performing Analysis on the data
5. Spark Machine Learning Library
6. MongoDb*
7. Tableau
8. Jupyter Notebook - Scala kernel

* Dumping historic and predicted data on NoSQL database like MongoDB

Risk Analysis Process

1. Fetch real-time stock data from Alpha Vantage API
2. Perform cointegration test for pairs trading strategy
3. Develop ensemble of machine learning models to predict the momentum of the asset
4. Analyse stock and predicted value to make a decision
5. Calibrate portfolio to minimise risk

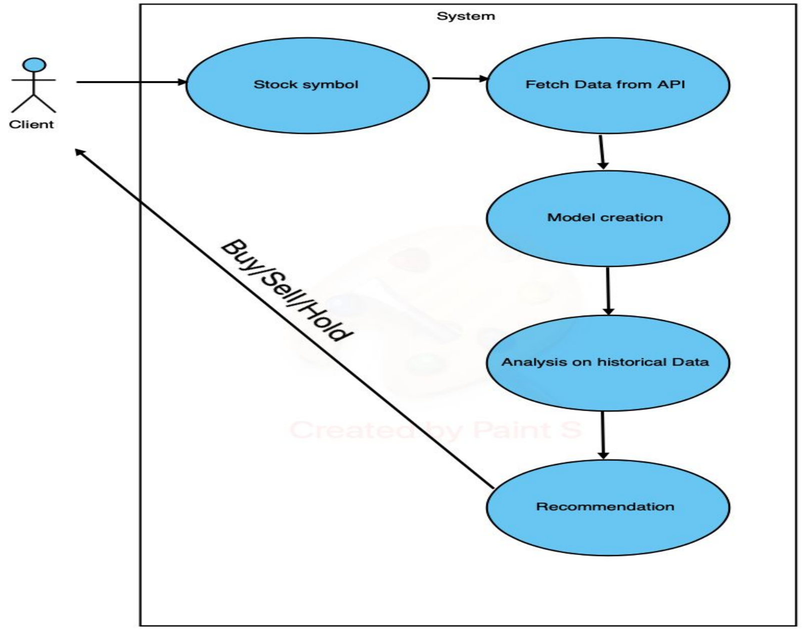
Ensemble Models

1. Linear Regression
2. Decision Tree Regression
3. Random forest Regression
4. Gradient Boosting Regression

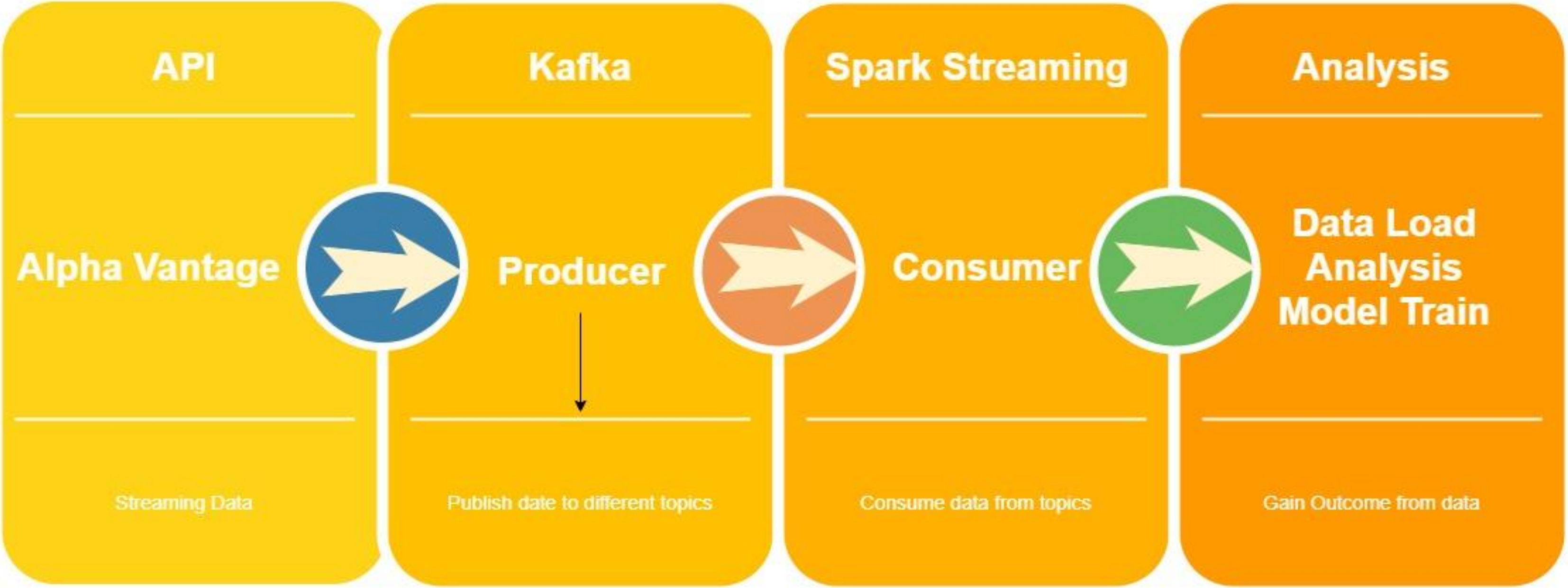
Data features: Stocks

1. Open (Independent feature)
2. High
3. Close
4. Adj Open
5. Volume
6. Low

Use Case Diagram



High Level Architecture



API

Alpha Vantage

Streaming Data

Kafka

Producer

Publish date to different topics

Spark Streaming

Consumer

Consume data from topics

Analysis

Data Load
Analysis
Model Train

Gain Outcome from data

Project Plan

Week 1: (23rd March - 29th March)

- Setting up new git repo
- Setting up whole ecosystem (Spark + Scala + Kafka + API Keys)

Week 2: (30th March - 5th April)

- Improve the code base
- Handle Exceptions and Errors

Project Plan

Week 3: (6th April - 12th April)

- Create test cases and verify all the functions
- Make the code more modular and remove repetition

Week 4: (13 April - 15th April)

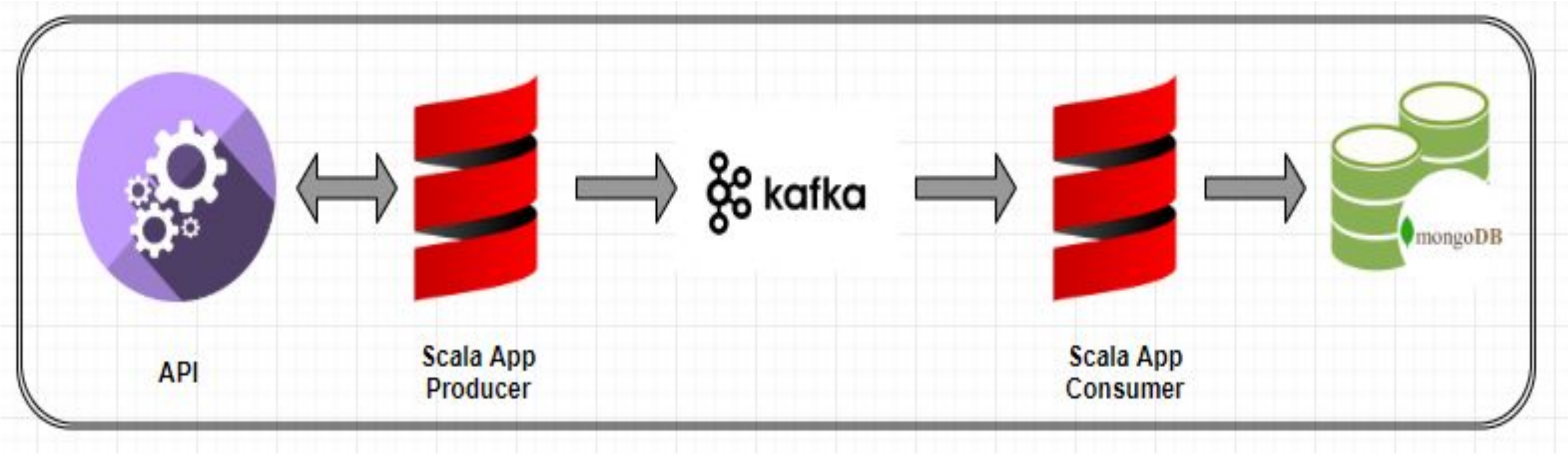
- Check the robustness the code by running on longer time
- Visualize the data using a visualization tool

Project subdivision:

- Data Engineering
- Machine Learning
- Data Analytics

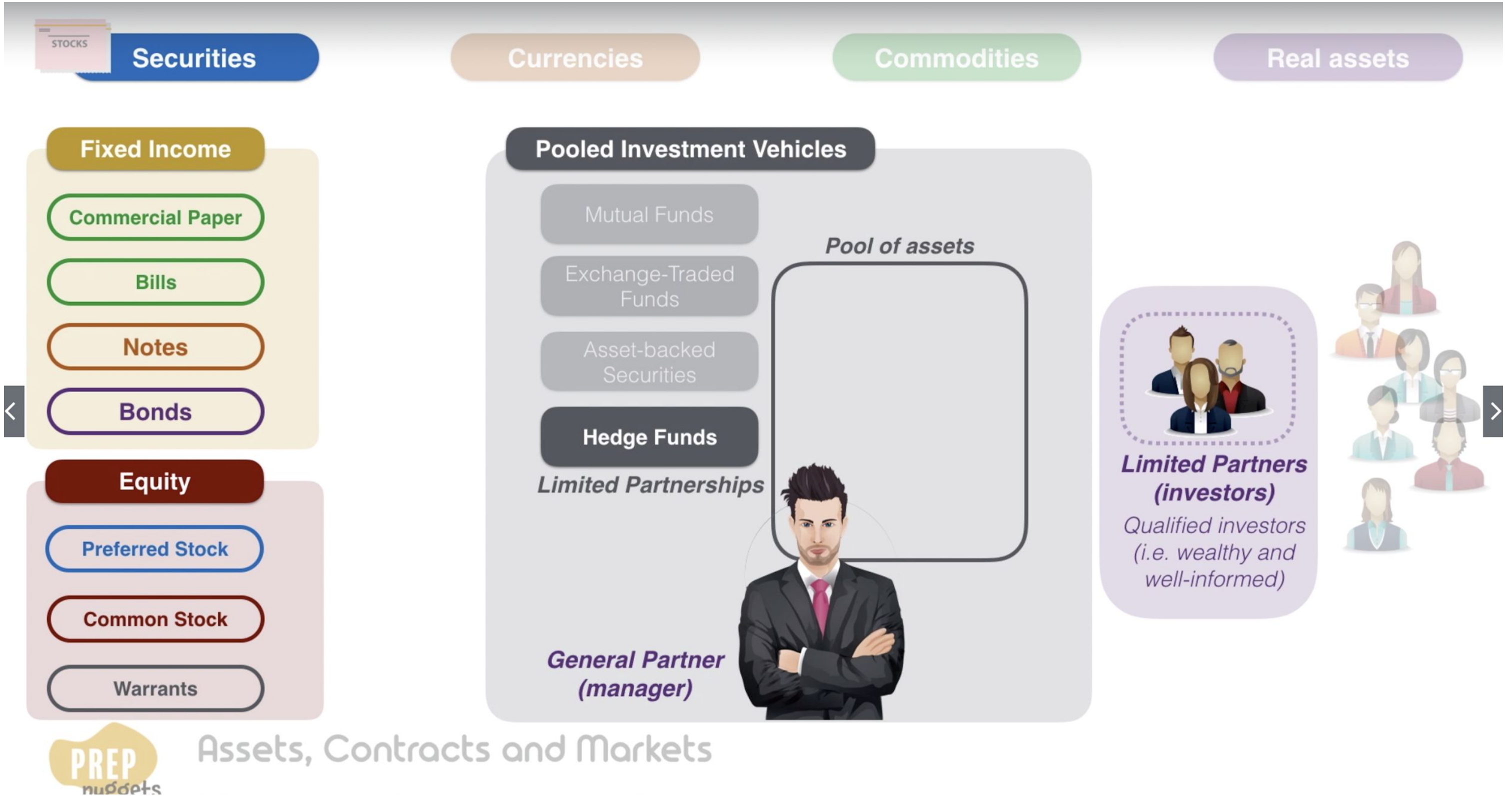
Data Engineering

Data Engineering:



Machine Learning

What is Hedge Fund?

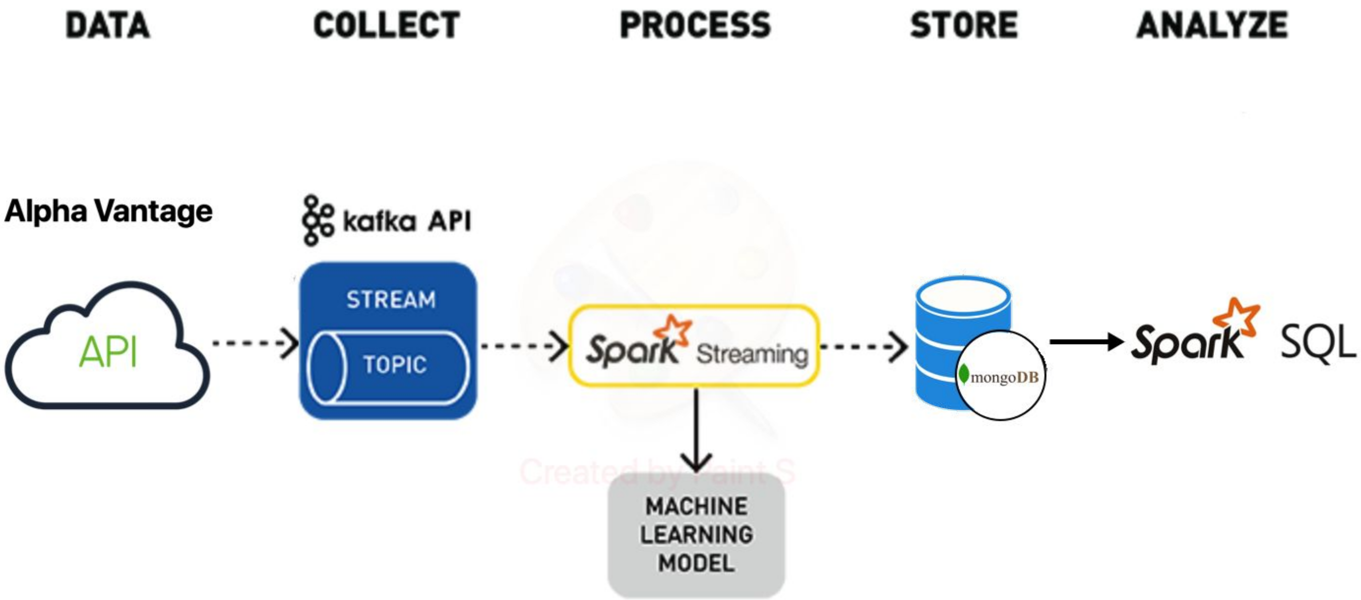


Risk Management in Portfolio Construction

Methods Used in Hedge Fund Company:

- Inverse correlation of the securities
- Protected Put (Long Position on a Asset + Long Put Position on the same Option Market Asset)
- Fiduciary Call (Long Call on the Options Market + Risk Free Bond)

Architecture



DATA

COLLECT

PROCESS

STORE

ANALYZE

Alpha Vantage

API

kafka API

STREAM

TOPIC

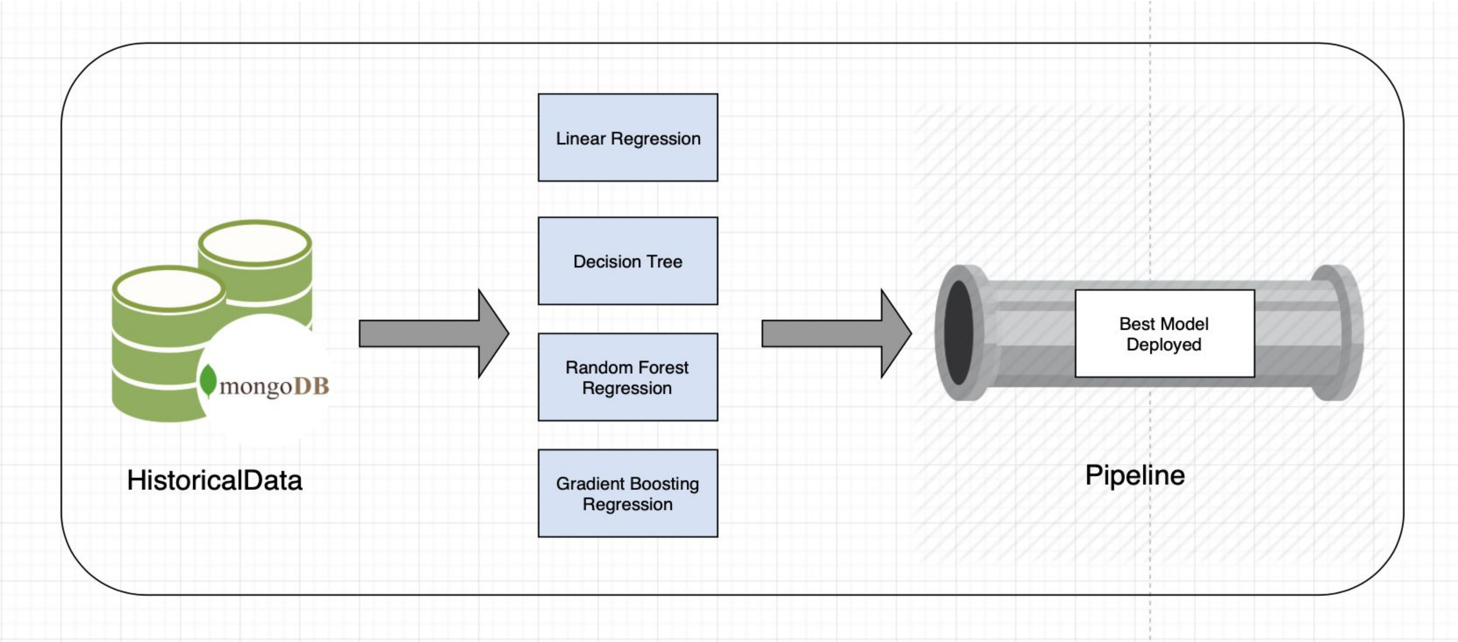
Spark Streaming

MACHINE LEARNING MODEL

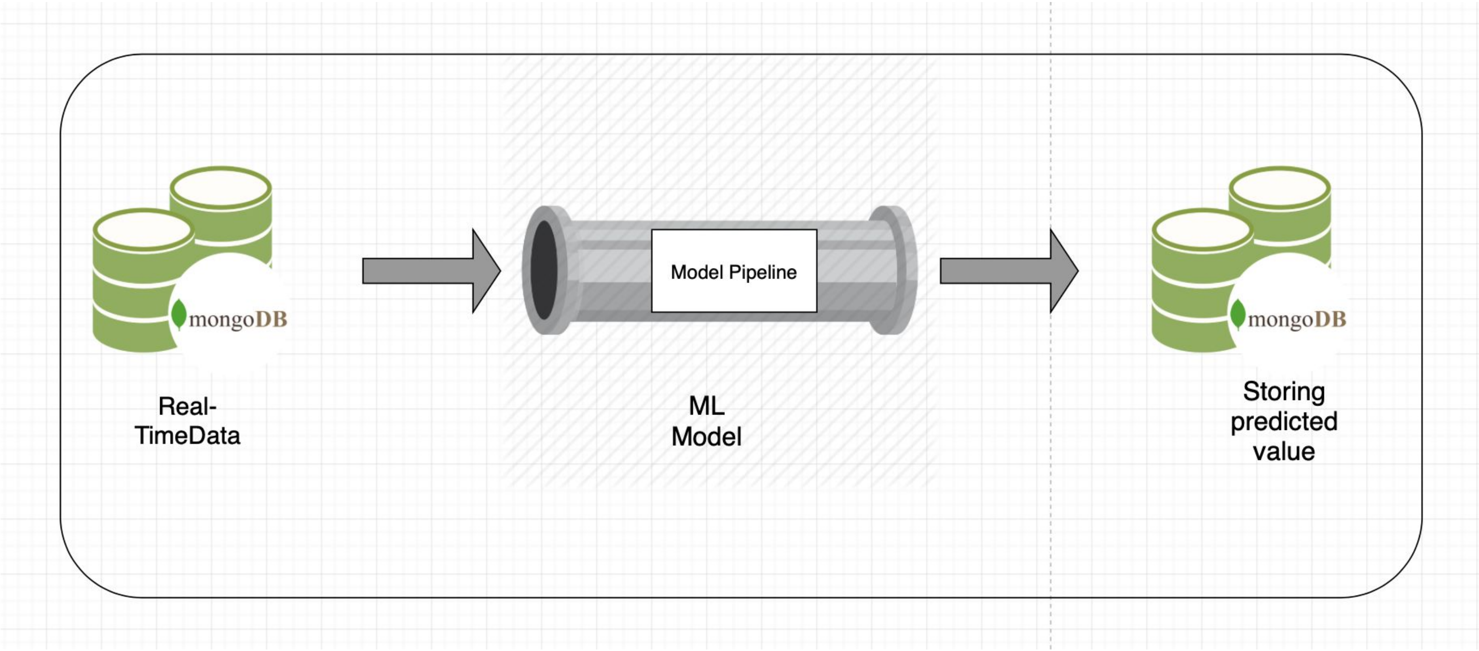
mongoDB

Spark SQL

ML Pipeline Construction



Real-Time Prediction



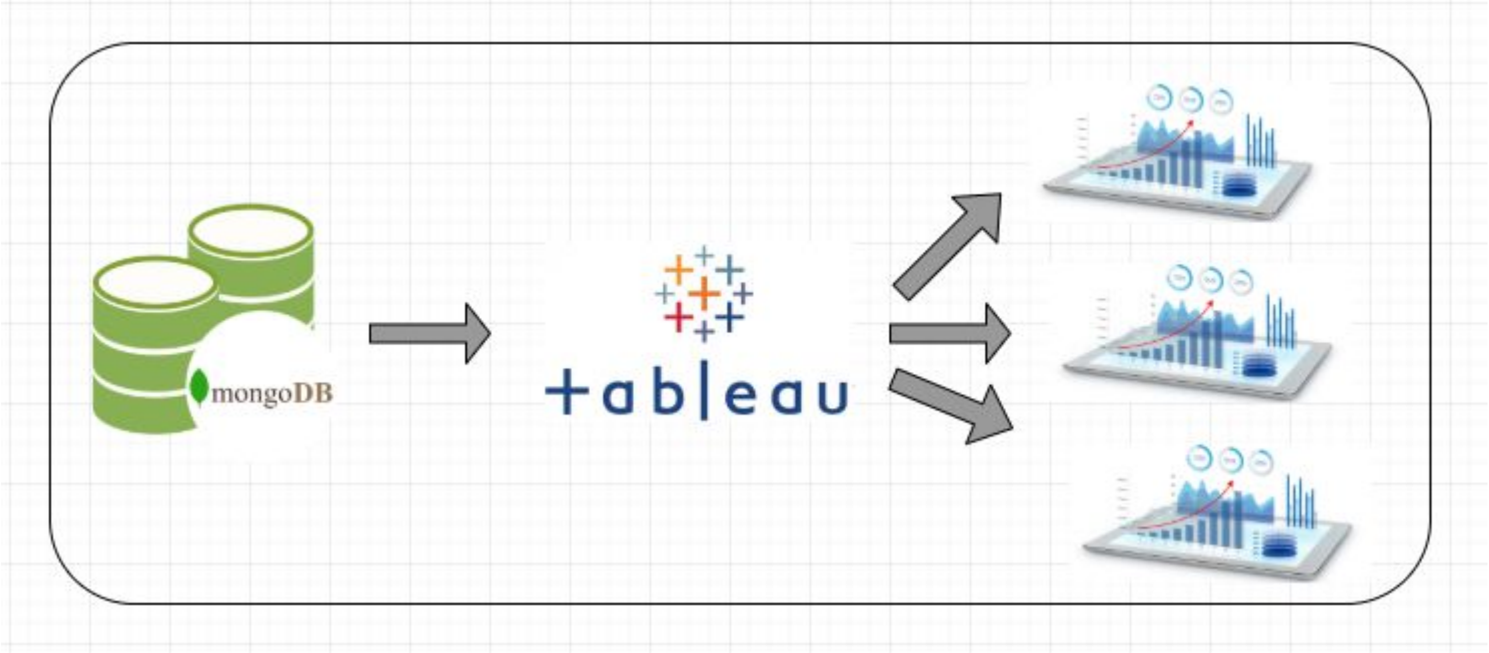
Trading Strategy

Assumption:

- The market value of a security is corrupted by noise
  - High Frequency Trading
  - Market Sentiment
  - False Information
- The model constructed is immune to these noises
- Conclusion:
  - Real Value > Predicted Value ---> Security is overvalued ---> Sell/Short Position
  - Real Value < Predicted Value ---> Security is undervalued ---> Buy/Long Position
  - Real Value == Predicted Value ---> Security is properly valued ---> Hold Position

Data Analysis

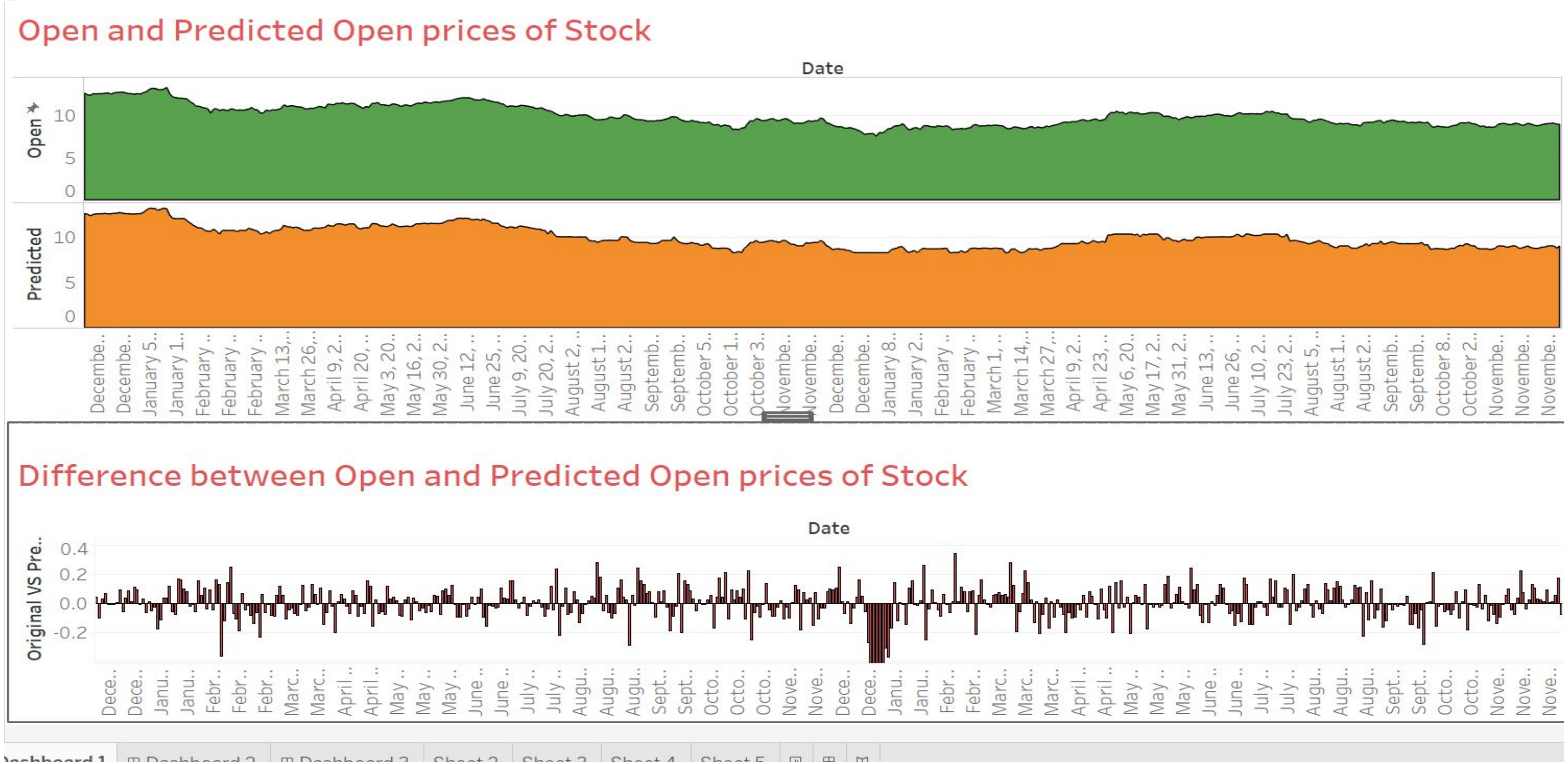
Data Analytics:



Data Analytics

1. Live data Streaming: Fetching Open stock price and Predicted stock price getting fed in MongoDB on real time basis
2. Connecting MongoDB with Tableau using BI Connector - Simba for Real Time data visualisation
3. In Tableau, presenting dynamic charts (changing with respect to Date-time) as per the data input
4. Showing Error (Original - Predicated stock price) , to guide user, if to sell or purchase the stock
5. Showing how change in Open and Predicted Open stock affects the Volume of the stock

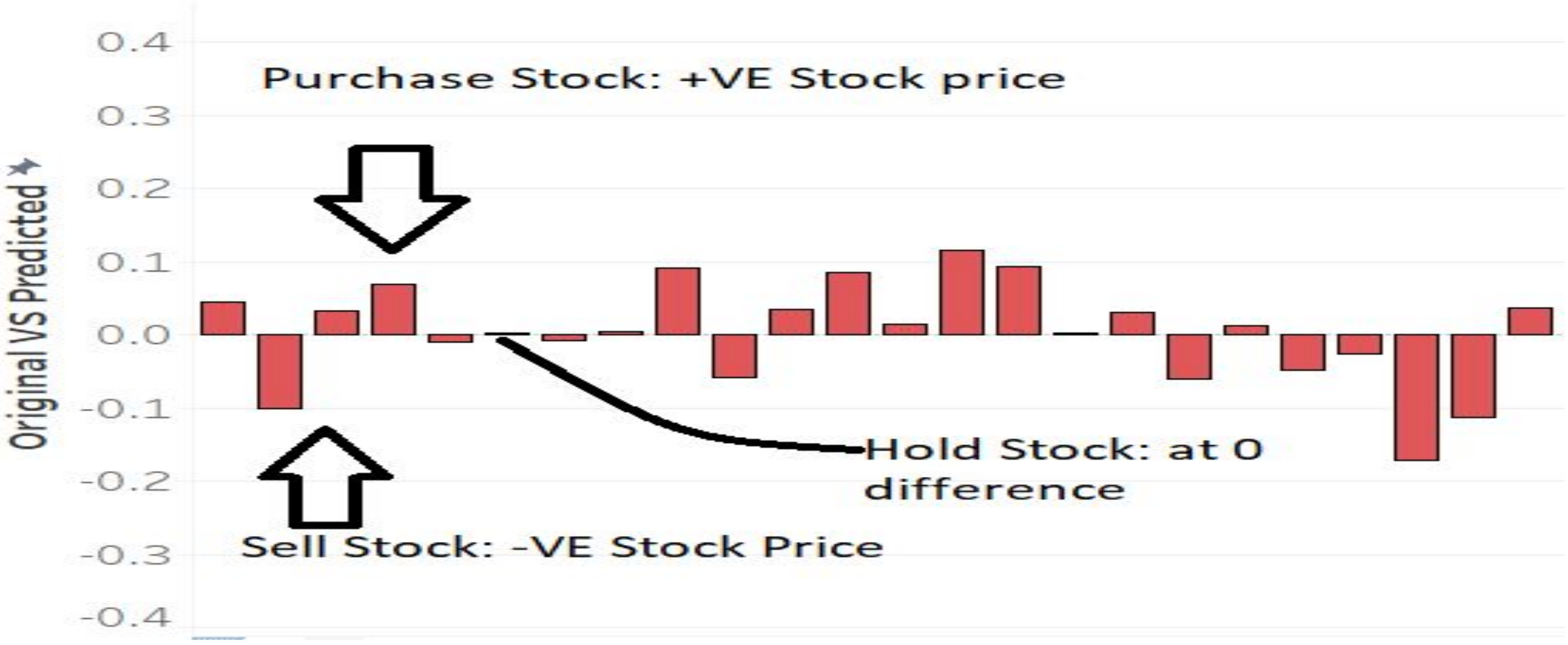
Data Analytics (a)



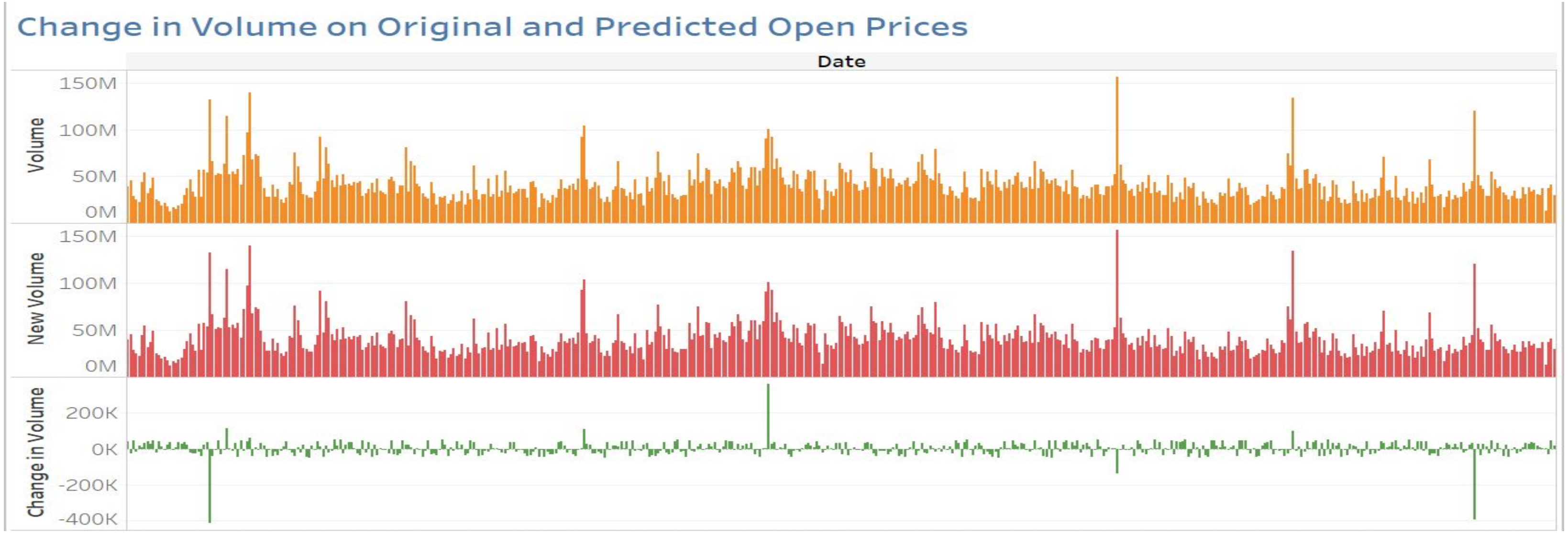
Open and Predicted Open prices of Stock

Difference between Open and Predicted Open prices of Stock

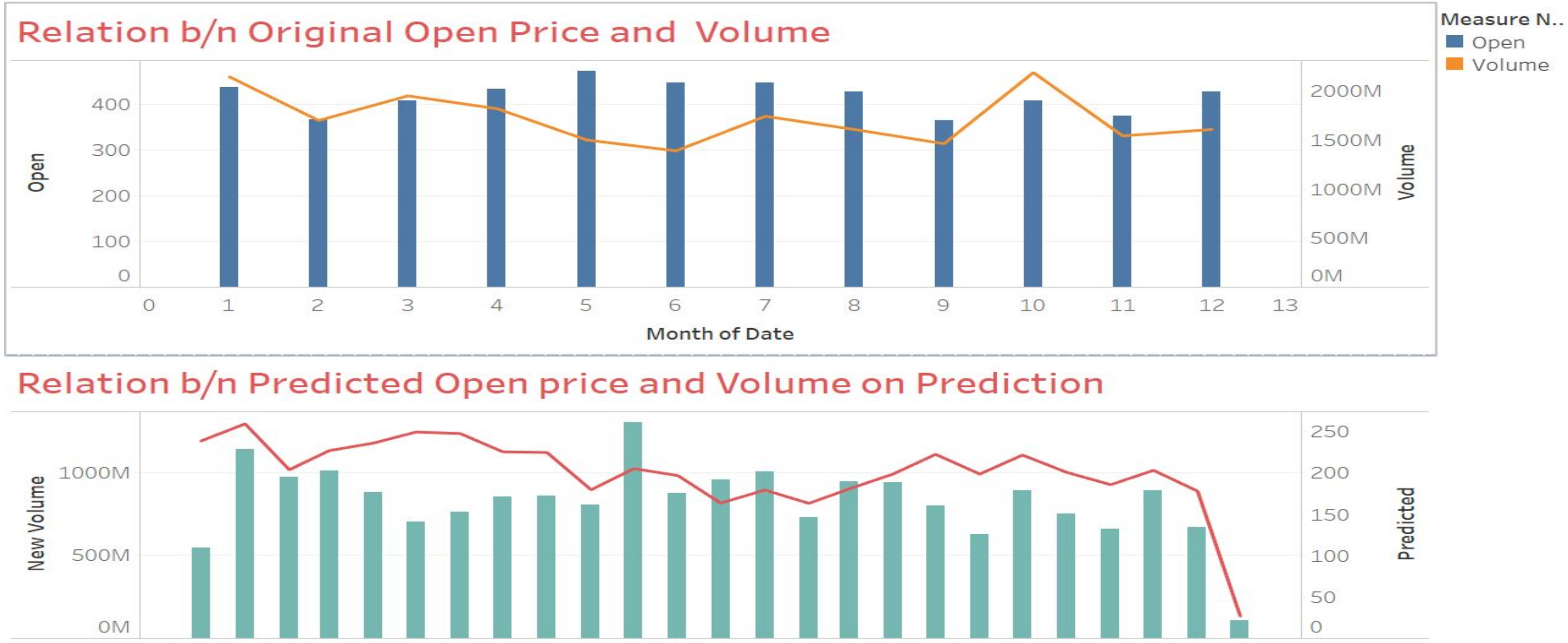
Data Analytics: When to sell, Purchase or Hold a stock



Data Analytics (b)



Data Analytics (c)

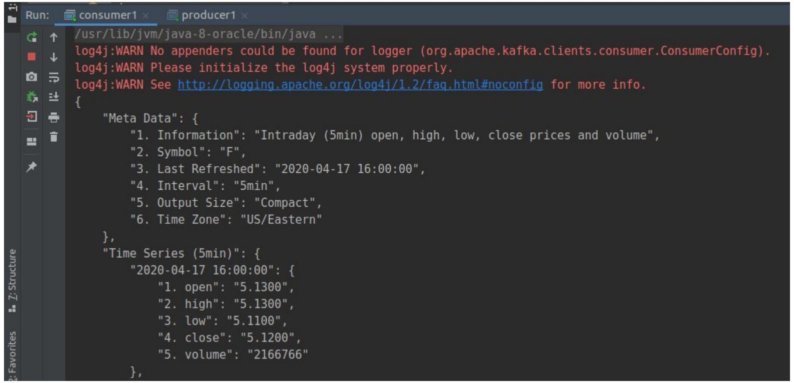


Relation b/n Original Open Price and Volume

Relation b/n Predicted Open price and Volume on Prediction

Acceptance criteria: 1 ✓

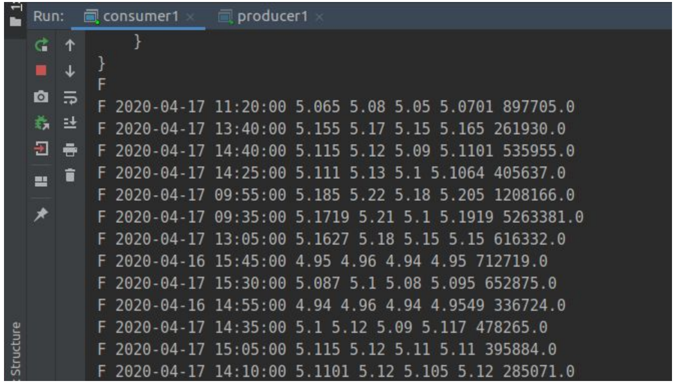
- Data from API should be fetched in every **5 min** and published with in **5 Seconds** on Kafka Topics
  - Achieved, Time to fetch = 5 min
  - Published on kafka topic = ~3.74 Seconds



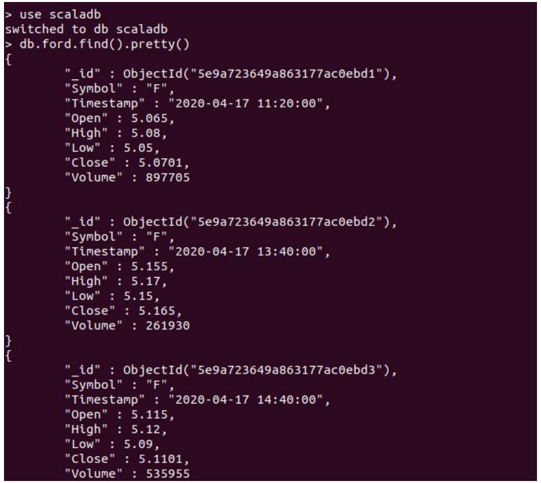
```
/usr/lib/jvm/java-8-oracle/bin/java ...
log4j:WARN No appenders could be found for logger (org.apache.kafka.clients.consumer.ConsumerConfig).
log4j:WARN Please initialize the log4j system properly.
log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more info.
{
    "Meta Data": {
        "1. Information": "Intraday (5min) open, high, low, close prices and volume",
        "2. Symbol": "F",
        "3. Last Refreshed": "2020-04-17 16:00:00",
        "4. Interval": "5min",
        "5. Output Size": "Compact",
        "6. Time Zone": "US/Eastern"
    },
    "Time Series (5min)": {
        "2020-04-17 16:00:00": {
            "1. open": "5.1300",
            "2. high": "5.1300",
            "3. low": "5.1100",
            "4. close": "5.1200",
            "5. volume": "2166766"
        },
```

Acceptance criteria: 2 ✓

- Consume the data from Kafka topics with in **5 seconds** as it arrived, validate each data and load correct data into database
    - ○ Consume from kafka topic = ~2.67 Seconds



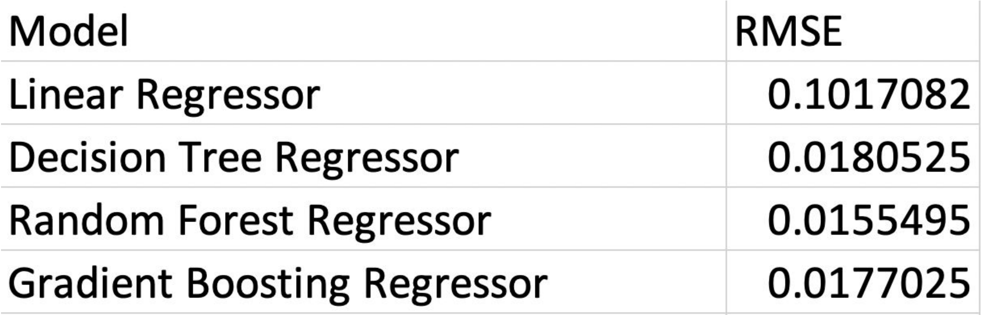
```
        }
}
F
F 2020-04-17 11:20:00 5.065 5.08 5.05 5.0701 897705.0
F 2020-04-17 13:40:00 5.155 5.17 5.15 5.165 261930.0
F 2020-04-17 14:40:00 5.115 5.12 5.09 5.1101 535955.0
F 2020-04-17 14:25:00 5.111 5.13 5.1 5.1064 405637.0
F 2020-04-17 09:55:00 5.185 5.22 5.18 5.205 1208166.0
F 2020-04-17 09:35:00 5.1719 5.21 5.1 5.1919 5263381.0
F 2020-04-17 13:05:00 5.1627 5.18 5.15 5.15 616332.0
F 2020-04-16 15:45:00 4.95 4.96 4.94 4.95 712719.0
F 2020-04-17 15:30:00 5.087 5.1 5.08 5.095 652875.0
F 2020-04-16 14:55:00 4.94 4.96 4.94 4.9549 336724.0
F 2020-04-17 14:35:00 5.1 5.12 5.09 5.117 478265.0
F 2020-04-17 15:05:00 5.115 5.12 5.11 5.11 395884.0
F 2020-04-17 14:10:00 5.1101 5.12 5.105 5.12 285071.0
```



```
> use scaladb
switched to db scaladb
> db.ford.find().pretty()
{
        "_id" : ObjectId("5e9a723649a863177ac0ebd1"),
        "Symbol" : "F",
        "Timestamp" : "2020-04-17 11:20:00",
        "Open" : 5.065,
        "High" : 5.08,
        "Low" : 5.05,
        "Close" : 5.0701,
        "Volume" : 897705
}
{
        "_id" : ObjectId("5e9a723649a863177ac0ebd2"),
        "Symbol" : "F",
        "Timestamp" : "2020-04-17 13:40:00",
        "Open" : 5.155,
        "High" : 5.17,
        "Low" : 5.15,
        "Close" : 5.165,
        "Volume" : 261930
}
{
        "_id" : ObjectId("5e9a723649a863177ac0ebd3"),
        "Symbol" : "F",
        "Timestamp" : "2020-04-17 14:40:00",
        "Open" : 5.115,
        "High" : 5.12,
        "Low" : 5.09,
        "Close" : 5.1101,
        "Volume" : 535955
```

Acceptance criteria 3 ✓

- Selecting best model depending upon RMSE value (**RMSE < 0.7**)



Model	RMSE
Linear Regressor	0.1017082
Decision Tree Regressor	0.0180525
Random Forest Regressor	0.0155495
Gradient Boosting Regressor	0.0177025

Acceptance criteria: 4

- Update data in real time interactive dashboards with maximum lag of **2 mins**
  - Premium subscription of BI tool is required for Real Time Interactive Dashboards
  - Using manual refresh Dashboards getting updated with latency of ~5 mins

Thank You