Project Report:

Big Data Processing and Analysis Framework for Stock Price Prediction

Big Data Course

M2-DAS

Group: Han Chandeth, Li Nita and Nuon Roatny

1. Introduction

This project aims to create a specialized Big Data Processing and Analysis Framework tailored for Stock Price Prediction. By leveraging specific tools and technologies, the framework will encompass data ingestion, storage, and advanced analytics, ultimately predicting stock prices for the next day.

Objectives:

- Establish a streamlined pipeline for ingesting, processing, and storing stock market data.
- Ensure data quality, consistency, and readiness for predictive analysis.

2. Methodology

2.1 Data Collection

We developed Python scripts for data scraping to collect historical stock data from the CSX website from 01/01/2019-10/11/2023.

2.2 Data Storage

Hadoop's HDFS was utilized for efficient storage of historical stock data.

CSV files containing historical stock data were saved in HDFS.

2.3 Data Processing and Analytics

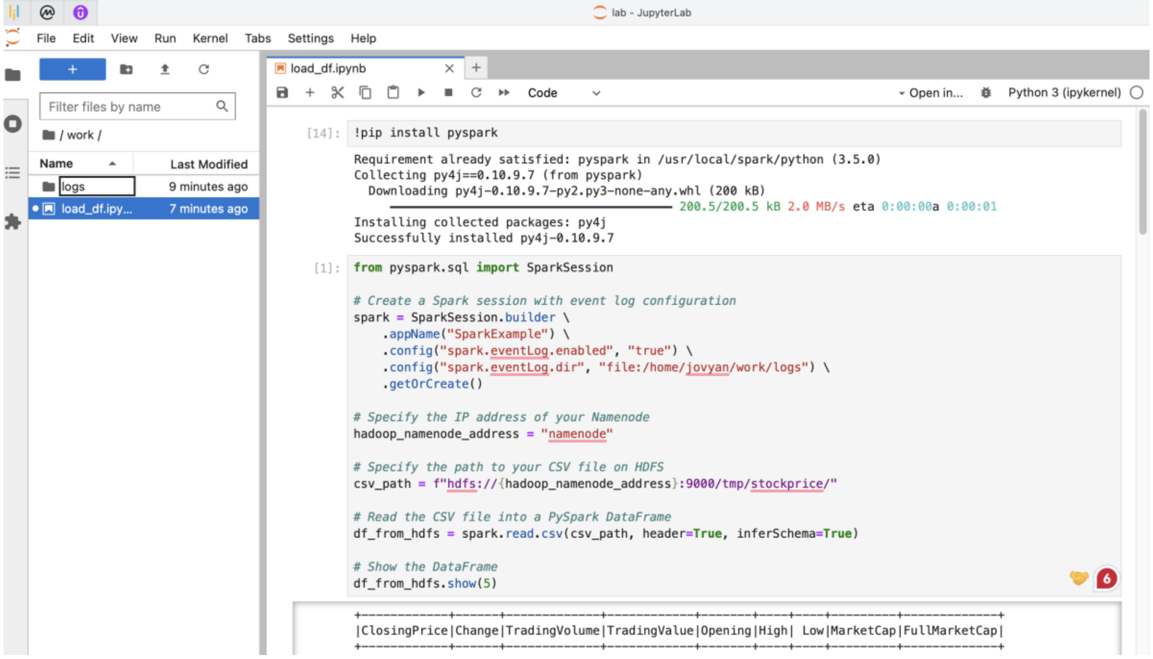
Apache Spark was leveraged for distributed in-memory data processing.

PySpark was used to create pipelines for feature extraction, exploration, and model training.

2.4 Machine Learning

Support Vector Machine (SVM) algorithm from Spark's MLlib was applied for stock price prediction.

The SVM model was trained using historical stock data from HDFS



```
[14]: !pip install pyspark

Requirement already satisfied: pyspark in /usr/local/spark/python (3.5.0)
Collecting py4j==0.10.9.7 (from pyspark)
  Downloading py4j-0.10.9.7-py2.py3-none-any.whl (200 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 200.5/200.5 kB 2.0 MB/s eta 0:00:00a 0:00:01
Installing collected packages: py4j
Successfully installed py4j-0.10.9.7
```

```
[1]: from pyspark.sql import SparkSession

# Create a Spark session with event log configuration
spark = SparkSession.builder \
    .appName("SparkExample") \
    .config("spark.eventLog.enabled", "true") \
    .config("spark.eventLog.dir", "file:/home/jovyan/work/logs") \
    .getOrCreate()

# Specify the IP address of your Namenode
hadoop_namenode_address = "namenode"

# Specify the path to your CSV file on HDFS
csv_path = f"hdfs://{hadoop_namenode_address}:9000/tmp/stockprice/"

# Read the CSV file into a PySpark DataFrame
df_from_hdfs = spark.read.csv(csv_path, header=True, inferSchema=True)

# Show the DataFrame
df_from_hdfs.show(5)
```

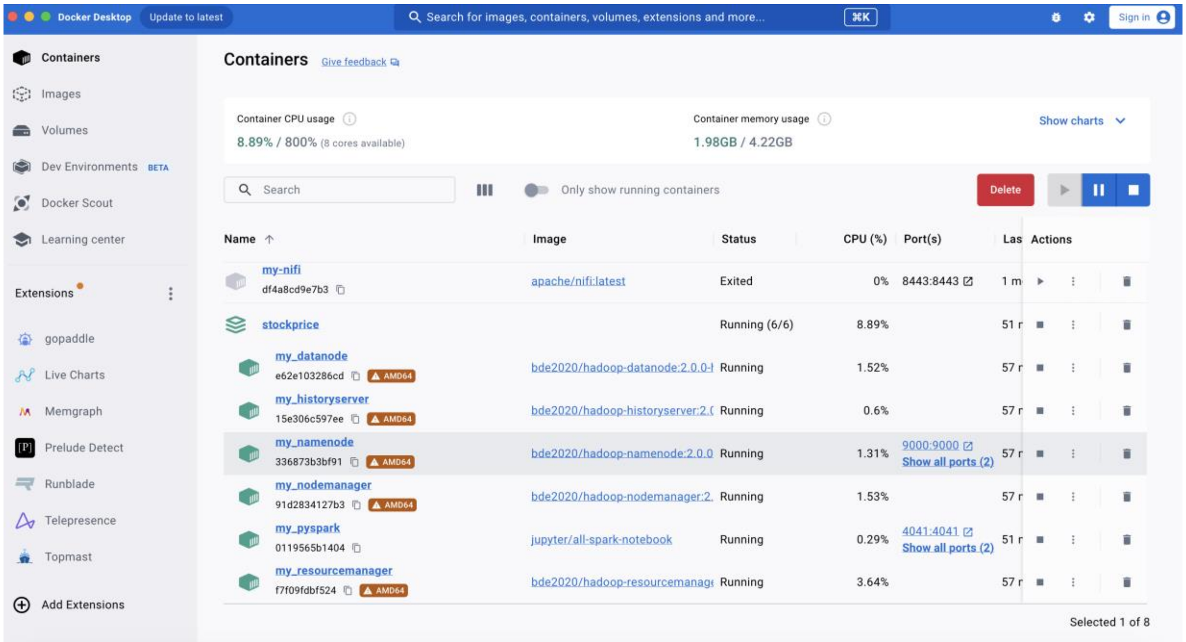
```
+-----------+------+-------------+------------+-------+----+----+---------+-------------+
|ClosingPrice|Change|TradingVolume|TradingValue|Opening|High| Low|MarketCap|FullMarketCap|
+-----------+------+-------------+------------+-------+----+----+---------+-------------+
```

3. Implementation

Docker-compose was used to set up a combined environment of Hadoop and Spark.

Data collection scripts, and Spark ML pipelines were implemented in Python.

Configuration files for Hadoop and Spark were customized to integrate with the Docker-compose environment.



Containers

Container CPU usage
8.89% / 800% (8 cores available)

Container memory usage
1.98GB / 4.22GB

Name	Image	Status	CPU (%)	Port(s)	Las	Actions
my-nifi df4a8cd9e7b3	apache/nifi:latest	Exited	0%	8443:8443	1 m	
stockprice		Running (6/6)	8.89%		51 r	
my_datanode e62e103286cd AMD64	bde2020/hadoop-datanode:2.0.0-l	Running	1.52%		57 r	
my_historyserver 15e306c597ee AMD64	bde2020/hadoop-historyserver:2.(	Running	0.6%		57 r	
my_namenode 336873b3bf91 AMD64	bde2020/hadoop-namenode:2.0.0	Running	1.31%	9000:9000 Show all ports (2)	57 r	
my_nodemanager 91d2834127b3 AMD64	bde2020/hadoop-nodemanager:2.	Running	1.53%		57 r	
my_pyspark 0119565b1404	jupyter/all-spark-notebook	Running	0.29%	4041:4041 Show all ports (2)	51 r	
my_resourcemanager f7f09fdbf524 AMD64	bde2020/hadoop-resourcemanage	Running	3.64%		57 r	

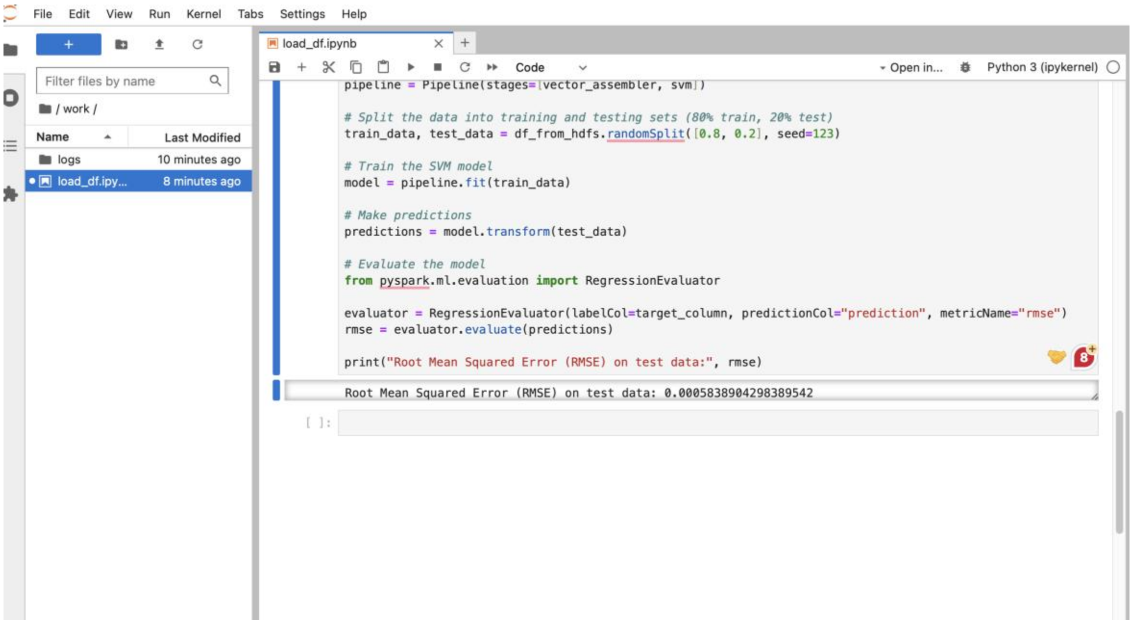
Selected 1 of 8

4. Result

Historical stock data was successfully collected and stored in HDFS.

Spark ML pipelines were developed for feature extraction and model training.

The SVM model was trained and evaluated, achieving a Root Mean Squared Error (RMSE) of 0.0005838904298389542 on test data.



Name	Last Modified
logs	10 minutes ago
load_df.ipy...	8 minutes ago

```
pipeline = Pipeline(stages=[vector_assembler, svm])

# Split the data into training and testing sets (80% train, 20% test)
train_data, test_data = df_from_hdfs.randomSplit([0.8, 0.2], seed=123)

# Train the SVM model
model = pipeline.fit(train_data)

# Make predictions
predictions = model.transform(test_data)

# Evaluate the model
from pyspark.ml.evaluation import RegressionEvaluator

evaluator = RegressionEvaluator(labelCol=target_column, predictionCol="prediction", metricName="rmse")
rmse = evaluator.evaluate(predictions)

print("Root Mean Squared Error (RMSE) on test data:", rmse)
```

```
Root Mean Squared Error (RMSE) on test data: 0.0005838904298389542
```

[ ]:

5. Challenges and Future implementation

Setting up Kafka for real-time data streaming posed challenges due to configuration issues and port conflicts. Integrating Docker-compose environment with Kafka and resolving port conflicts required careful troubleshooting. This challenge will be our future task after completing this project too.

5. Conclusion

We successfully implemented data collection, storage, processing, and model training stages using Big Data technologies. And for real-time implementation will be the next task. The project lays a strong foundation for future enhancements and real-time prediction implementation. By combining specific tools and technologies, the framework aims to provide accurate insights for investors and financial analysts, enhancing decision-making processes in the stock market.