Analysis of Cattle Export Marketing Dataset With Big Data Tools in Hadoop Ecosystem

Author: Zemelak Goraga

Course: DSC650-T301 Big Data (2251-1)

Week 12

Final Project Report

Professor Nasheb Ismaily

Date: 11/16/2024

Table of Contents

1. Summary

This project utilized historical livestock export data (1961-2013) to analyze global cattle trade dynamics and develop predictive models for future export trends. The descriptive analysis identified key export contributions from regions such as Europe, the Americas, and Australia, with a peak in exports in 2010, largely driven by Western Europe. By integrating historical data from Hive and leveraging PySpark's advanced processing capabilities, the analysis offered a comprehensive view of the cattle trade landscape.

The predictive modeling component utilized a RandomForestRegressor model in PySpark, achieving a strong R^2 score of 0.92, indicating high forecasting accuracy, especially for stable exporters like Australia. However, the model exhibited a relatively high Root Mean Squared Error (RMSE) of 438,351 units, suggesting opportunities for further refinement, particularly for volatile exporters like Argentina. The model's outputs were efficiently stored in Hive tables, enabling seamless performance monitoring and laying the foundation for ongoing model enhancement and data-driven decision-making.

2. Introduction

The global cattle trade plays a crucial role in shaping agricultural economies and ensuring food security across the world. It is a multi-billion dollar industry that directly impacts the livelihoods of millions of people, particularly in regions heavily dependent on agriculture. However, predicting market trends in the cattle trade is a complex challenge due to factors like fluctuating prices, varying demand, and unpredictable trade policies. These challenges are further compounded by economic shifts, climate changes, and geopolitical events, making it difficult for stakeholders to make informed decisions. Accurate, data-driven strategies are essential for optimizing supply chains, ensuring profitability, and fostering sustainable agricultural practices.

This project aims to address the complexities of the global cattle trade by leveraging both historical and real-time data analytics to predict cattle export trends. The analysis focuses on historical export data spanning over five decades (1961-2013), combined with real-time data streaming to provide up-to-date insights. The goal is to develop an end-to-end data pipeline that uses advanced big data technologies, enabling stakeholders to make data-driven decisions. By incorporating tools like Apache NiFi for data ingestion, Apache Kafka for real-time streaming, and Apache Spark for predictive modeling, this project delivers a comprehensive solution for forecasting export quantities and optimizing marketing strategies.

The innovative approach taken in this project not only focuses on historical trend analysis but also integrates real-time analytics for continuous monitoring of the market. This dual approach provides

stakeholders with the agility to respond quickly to market fluctuations, thereby enhancing decision-making processes. The use of machine learning models, such as Random Forest regressors, helps to predict future export volumes with high accuracy, offering stakeholders a significant advantage in strategizing for future market conditions. Ultimately, the project highlights how the integration of big data technologies and predictive analytics can transform agricultural trade, fostering a more resilient and efficient market landscape.

3. Problem Statement

The global cattle trade is a key component of the agricultural sector, significantly influencing economies and food security worldwide. However, the market faces multiple challenges, including price fluctuations, variable demand, and changing trade policies, all of which complicate efforts to predict future market behavior. To develop data-driven market strategies, stakeholders need reliable forecasts that consider both historical trends and real-time changes. This project aims to address these challenges by analyzing historical and real-time data on cattle exports, focusing on export quantities and values. The objective is to identify patterns, forecast future trends, and develop optimization strategies that enhance decision-making, increase market efficiency, and promote sustainability in the agricultural sector.

4. Scope of the project

The project leveraged a comprehensive historical dataset on cattle exports, covering the years 1961 to 2013, with data from over 200 countries. The analysis focused on export quantities and values, utilizing various economic indicators and trade variables. Through a combination of advanced data processing, trend analysis, and predictive modeling, the project generated actionable insights for market stakeholders. Additionally, real-time data ingestion and streaming were integrated, enhancing the system's responsiveness to new data inputs. This dynamic framework required the use of robust big data tools like Apache Hive, PySpark, and HBase, along with machine learning techniques, to handle and analyze the large-scale, multi-dimensional dataset efficiently and effectively.

5. About the Dataset

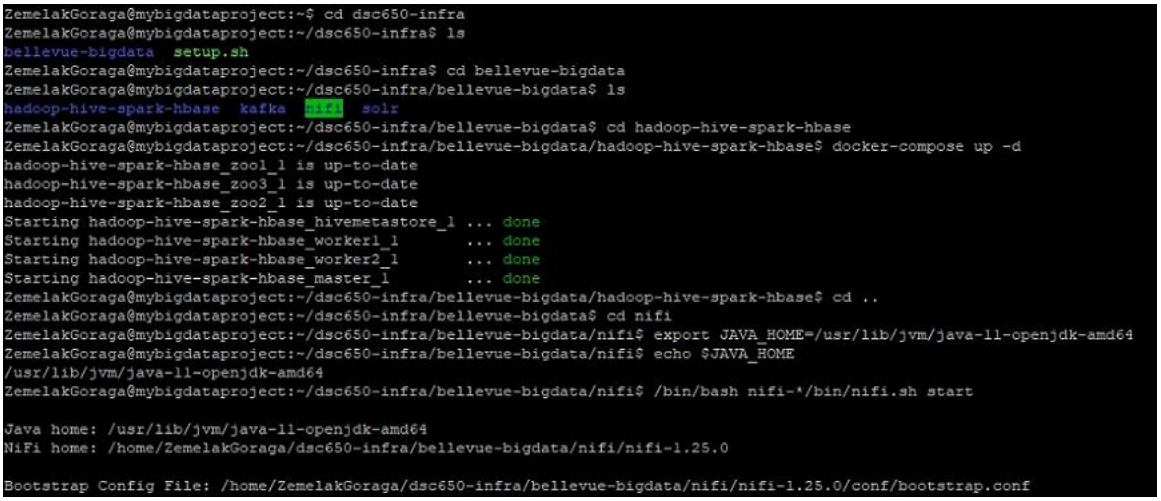
For this project, the FAOSTAT historical dataset "global-food-agriculture-statistics" was obtained from Kaggle (https://www.kaggle.com/datasets/unitednations/global-food-agriculture-statistics) using API command (kaggle datasets download -d unitednations/global-food-agriculture-statistics). The dataset included data from over 200 countries and covered more than 25 primary agricultural products and inputs collected between 1961 and 2013. Key variables included in the dataset were

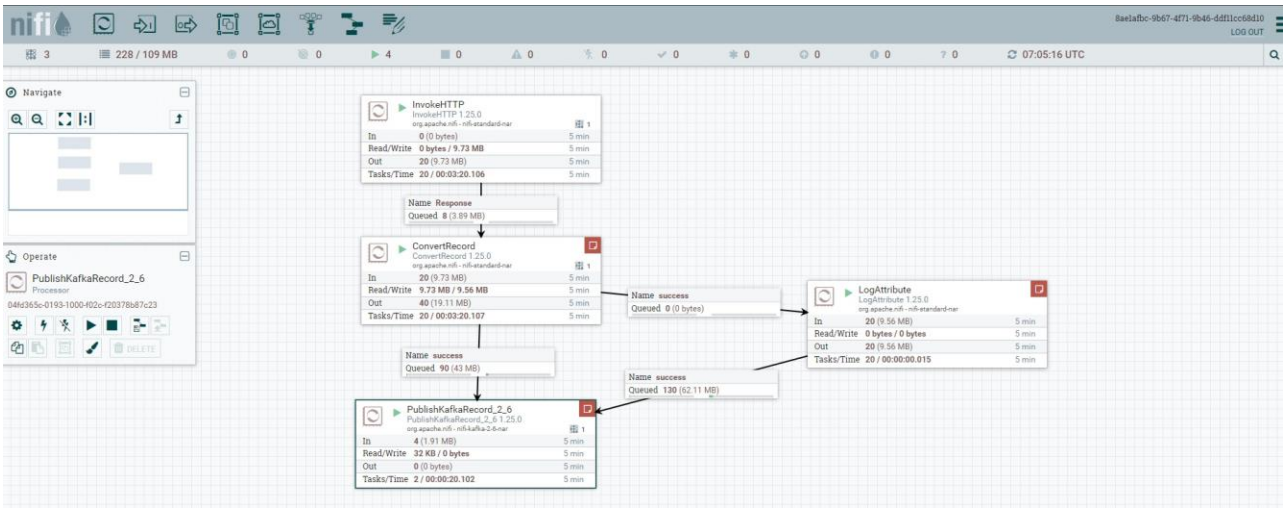
Area (Country), Item (e.g., Cattle, Sheep, Crops), Element (Import/Export Quantity and Value), Year, and Unit (measured in heads or US dollars). For the purposes of this project, the livestock component data was extracted and stored in a GitHub repository for easy access and processing (https://raw.githubusercontent.com/zemelak-s-goraga/DSC650/refs/heads/main/FinalProject/dataset/livestock_export_data.csv). The focus was on analyzing 'Export Quantity' of live Cattle which was used as dependent variables.

6. Methodology

6.1. Environment Setup

The project begins with setting up a robust big data environment to ensure smooth execution of all processes. This involves launching essential components such as Apache ZooKeeper, Kafka, and Hadoop Distributed File System (HDFS) using Docker containers. Docker simplifies the management of these services, allowing them to run in isolated environments while ensuring consistent configurations across different environments. Once the Docker containers are up, the project configures critical dependencies, especially Java, which is a foundational requirement for several big data tools. Properly setting the JAVA_HOME environment variable is crucial, as it ensures that all Java-based applications, including Hadoop and NiFi, can operate without compatibility issues. Subsequently, Apache NiFi is started to manage data flow seamlessly, laying the groundwork for efficient data ingestion and processing in later stages.





6.2. Data Ingestion and Storage

Data ingestion is a critical step in any big data pipeline, and in this project, it is initiated by retrieving a CSV dataset containing historical livestock export data. The dataset is directly downloaded from a public GitHub repository using the wget command, which simplifies the process of obtaining external data. After downloading, the data is uploaded into HDFS, the primary storage layer, using the hdfs dfs -put command. HDFS provides scalable, fault-tolerant storage that can handle large volumes of data efficiently, making it ideal for storing historical datasets. Following this, an external table is created in Hive, linking to the data stored in HDFS. This Hive table, named livestocks_export_data, is designed to mirror the structure of the CSV file, allowing users to query and explore the data using SQL-like syntax without having to move the data around. The use of Hive enables structured querying and facilitates efficient data analysis.

```
ZemelakGoraga@mybigdataproject:~/dsc650-infra/bellevue-bigdata/hadoop-hive-spark-hbase$ docker-compose ex
bash-5.0# wget https://raw.githubusercontent.com/zemelak-s-goraga/DSC650/refs/heads/main/FinalProject/dat
Connecting to raw.githubusercontent.com (185.199.108.133:443)
livestock_export_dat 100% |*******************************************************|  498k  0:00:00 ETA
bash-5.0#
```

```
bash-5.0# hdfs dfs -mkdir /data
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/usr/program/hadoop/share/hadoop/common/lib/slf4j-log4j12-1.7.25.jar!/org/slf4j/impl/StaticLoggerBinder.
SLF4J: Found binding in [jar:file:/usr/program/tez/lib/slf4j-log4j12-1.7.10.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/program/hive/lib/log4j-slf4j-impl-2.10.0.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
2024-11-05 20:16:20,944 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes w
bash-5.0# hdfs dfs -put livestock_export_data.csv /data/
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/usr/program/hadoop/share/hadoop/common/lib/slf4j-log4j12-1.7.25.jar!/org/slf4j/impl/StaticLoggerBinder.
SLF4J: Found binding in [jar:file:/usr/program/tez/lib/slf4j-log4j12-1.7.10.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/program/hive/lib/log4j-slf4j-impl-2.10.0.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
2024-11-05 20:16:30,822 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes w
bash-5.0# hdfs dfs -ls /data
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/usr/program/hadoop/share/hadoop/common/lib/slf4j-log4j12-1.7.25.jar!/org/slf4j/impl/StaticLoggerBinder.
SLF4J: Found binding in [jar:file:/usr/program/tez/lib/slf4j-log4j12-1.7.10.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/program/hive/lib/log4j-slf4j-impl-2.10.0.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
2024-11-05 20:16:48,184 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes w
Found 1 items
-rw-r--r--   1 root supergroup     510299 2024-11-05 20:16 /data/livestock_export_data.csv
bash-5.0#
```

```
bash-5.0# hive
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/usr/program/hive/lib/log4j-slf4j-impl-2.10.0.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/program/hadoop/share/hadoop/common/lib/slf4j-log4j12-1.7.25.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/program/tez/lib/slf4j-log4j12-1.7.10.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.apache.logging.slf4j.Log4jLoggerFactory]
Hive Session ID = 14fd2798-f693-415b-995f-0fc305cf178c

Logging initialized using configuration in file:/usr/program/hive/conf/hive-log4j2.properties Async: true
2024-11-05 21:05:55,787 INFO  [Tez session start thread] client.RMProxy: Connecting to ResourceManager at master/172.28.1.1:8032
Hive Session ID = bf4f94ea-e550-42d8-96b3-d7910f303e12
hive> 2024-11-05 21:05:57,204 INFO  [pool-7-thread-1] client.RMProxy: Connecting to ResourceManager at master/172.28.1.1:8032
CREATE EXTERNAL TABLE IF NOT EXISTS livestock_export_data (
    >     area STRING,
    >     item STRING,
    >     element STRING,
    >     year INT,
    >     unit STRING,
    >     value FLOAT
    > )
    > ROW FORMAT DELIMITED
    > FIELDS TERMINATED BY ','
    > STORED AS TEXTFILE
    > LOCATION '/data';
OK
Time taken: 2.101 seconds
hive> SHOW TABLES LIKE 'livestock_export_data';
OK
livestock_export_data
Time taken: 0.248 seconds, Fetched: 1 row(s)
```

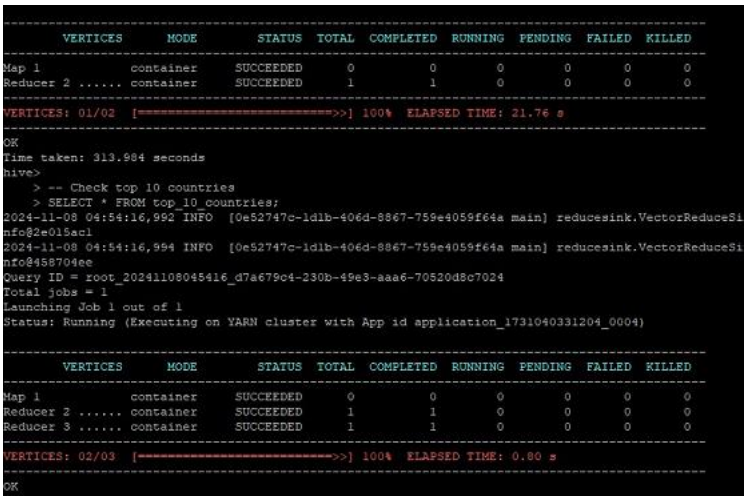
```
hive> DESCRIBE livestock_export_data;
OK
area                    string
item                    string
element                 string
year                    int
unit                    string
value                   float
Time taken: 0.243 seconds, Fetched: 6 row(s)
hive> SELECT * FROM livestock_export_data LIMIT 10;
OK
area    item    element NULL    unit    NULL
Argentina       Cattle  Export Quantity 1961    Head    171106.0
Argentina       Cattle  Export Quantity 1962    Head    250274.0
Argentina       Cattle  Export Quantity 1963    Head    291819.0
Argentina       Cattle  Export Quantity 1964    Head    166050.0
Argentina       Cattle  Export Quantity 1965    Head    102993.0
Argentina       Cattle  Export Quantity 1966    Head    119116.0
```

6.3. Data Analysis with Hive

Once the data is ingested into Hive, the project proceeds with a thorough exploratory data analysis (EDA) to understand the dataset's structure and key attributes. This step begins with basic queries to examine the contents of the livestocks_export_data table, focusing on fields like country, year, export quantity, and product type. The objective here is to gain an initial understanding of the distribution and trends within the data. Following the initial exploration, aggregation queries are performed to calculate total export quantities grouped by country and year. This allows the identification of top cattle-exporting nations and highlights historical trends in export activities. By using Hive's powerful query capabilities, this step lays the groundwork for deeper analysis and predictive modeling, helping to extract meaningful insights from the data.

```
hive> -- Step 1: Filter relevant data
hive> CREATE VIEW filtered_livestock_data AS
    > SELECT
    >     area AS Country,
    >     item,
    >     element,
    >     year,
    >     value
    > FROM livestocks_export_data
    > WHERE element = 'Export Quantity'
    >   AND item = 'Cattle'
    >   AND year BETWEEN 1998 AND 2013;
OK
Time taken: 10.0 seconds
hive> -- Step 2: Aggregate data by Country and Year
hive> CREATE VIEW country_year_aggregation AS
    > SELECT
    >     Country,
    >     year,
    >     SUM(value) AS Total_Export_Quantity
    > FROM filtered_livestock_data
    > GROUP BY Country, year;
OK
Time taken: 4.404 seconds
```

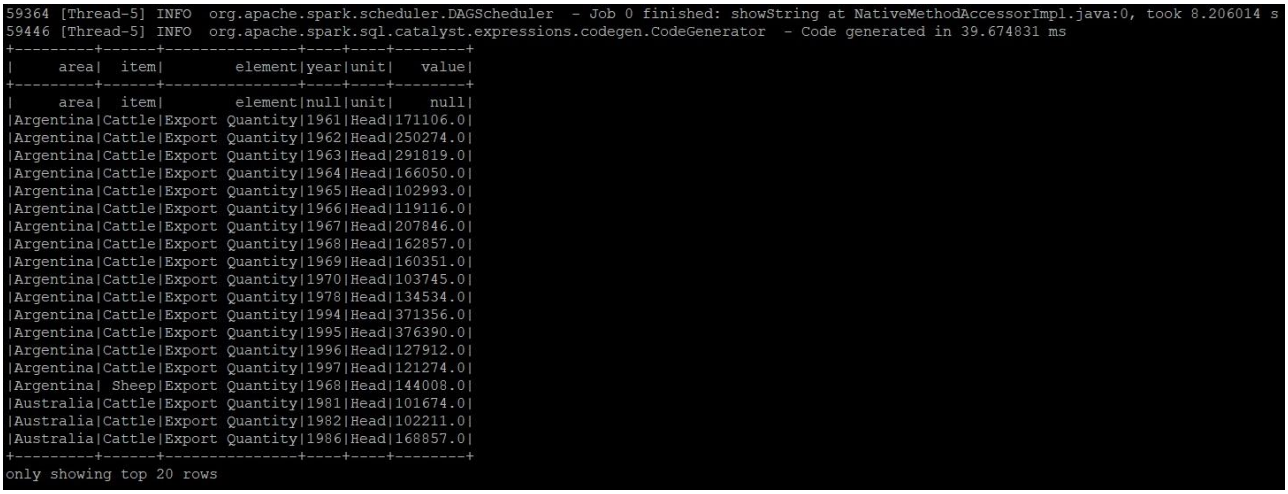
```
hive> -- Step 3: Find the top 10 countries with highest exports
hive> CREATE VIEW top_10_countries AS
    > SELECT
    >     Country,
    >     SUM(Total_Export_Quantity) AS Total
    > FROM country_year_aggregation
    > GROUP BY Country
    > ORDER BY Total DESC
    > LIMIT 10;
OK
Time taken: 1.634 seconds
hive> -- Step 4: Filter data for top 10 countries
hive> CREATE VIEW top_countries_trend AS
    > SELECT
    >     a.Country,
    >     a.year,
    >     a.Total_Export_Quantity
    > FROM country_year_aggregation a
    > JOIN top_10_countries b
    > ON a.Country = b.Country
    > ORDER BY a.year;
OK
Time taken: 2.272 seconds
```



6.4. Predictive Analysis with PySpark

In the predictive analysis phase, Apache Spark, specifically its Python API (PySpark), was employed to perform large-scale data processing. This phase involved integrating data from both Hive for historical records and Kafka for real-time streaming data, creating a robust dataset for predictive modeling. Initially, historical export data was extracted from Hive, focusing solely on cattle exports to ensure relevance. Concurrently, real-time data streamed through Kafka was incorporated, enriching the dataset with up-to-date market dynamics.

The combined dataset underwent filtering, transformation, and preparation for machine learning. Using PySpark, the data was aggregated to calculate annual export quantities by country, forming the basis for developing a predictive model.



```
106455 [Thread-5] INFO  org.apache.spark.scheduler.DAGScheduler  - Job 9 finished: showString at NativeMethodAccessorImpl.
106474 [Thread-5] INFO  org.apache.spark.sql.catalyst.expressions.codegen.CodeGenerator  - Code generated in 16.539499 ms
+----+--------------------+
|year|Avg_Export_Quantity|
+----+--------------------+
|1998|  928078.7446808511|
|1999|  986530.4130434783|
|2000| 1004826.5106382979|
|2001| 1014249.6511627907|
|2002|  1137377.564102564|
|2003|          979408.025|
|2004|  906519.2444444444|
|2005|           828638.48|
|2006|           930322.44|
|2007|            937147.0|
|2008|  857990.6666666666|
|2009|  904288.3529411765|
|2010| 1026692.3653846154|
|2011|  996724.4038461539|
|2012|  938774.8518518518|
|2013|  999473.1481481482|
+----+--------------------+
```

```
45829 [Thread-5] INFO  org.apache.spark.sql.catalyst.expressions.codegen.CodeGenerator  - Code generated in 25.499772 ms
+----------------+----------+
|area            |sum(value)|
+----------------+----------+
|World           |243906205 |
|Europe          |117772584 |
|European Union  |116592219 |
|Americas        |78696833  |
|Western Europe  |78103963  |
|France          |47991001  |
|Northern America|36829548  |
|Canada          |31252256  |
|Central America |28515546  |
|Mexico          |26714643  |
+----------------+----------+

45897 [shutdown-hook-0] INFO  org.apache.spark.SparkContext  - Invoking stop() from shutdown hook
45912 [shutdown-hook-0] INFO  org.sparkproject.jetty.server.AbstractConnector  - Stopped Spark@2b427b9b{HTTP/1.1,[http/1.1]}{172.28.1.1:4040}
```

The RandomForestRegressor algorithm from PySpark's MLlib library was chosen due to its effectiveness in handling large and complex datasets. The model was trained on a blend of historical and real-time data to predict future export volumes. Features such as the year, country-specific economic indicators, and real-time market trends (streamed from Kafka) were utilized as inputs. The dataset was split into training (70%) and testing (30%) subsets to validate model performance.

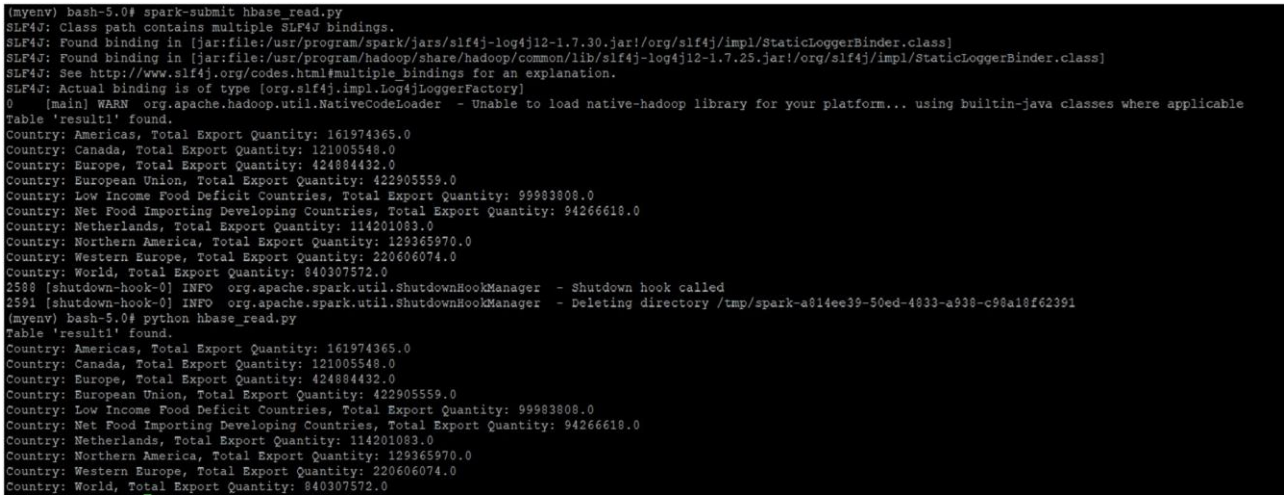
The model evaluation, which included Root Mean Squared Error (RMSE) and R-squared (R^2) metrics, showed significant accuracy. The model achieved an R^2 score of 0.92, explaining 92.48% of the variance in export quantities, with an RMSE of 438,351 units. These results confirmed the model's effectiveness in forecasting trends, particularly for stable exporters like Australia, while highlighting areas for improvement in volatile markets like Argentina.

```
177800 [Thread-5] INFO  org.apache.spark.scheduler.DAGScheduler  - Job 19 finished: treeAggregate at Statistics.scala:58, took 3.145163 s

Model Evaluation Metrics:
Root Mean Squared Error (RMSE): 438350.96733843
R-squared (R2): 0.9248130182855594

Writing performance metrics to HBase...
```

To ensure ongoing performance monitoring, evaluation metrics were stored in Hive tables, providing stakeholders with easy access to track model accuracy over time. This setup allows for iterative improvements and comparisons with future models. The integration of Hive for historical data, Kafka for streaming updates, and Hive storage for metric tracking established a scalable framework for real-time predictive analytics, empowering stakeholders to make informed, data-driven decisions.

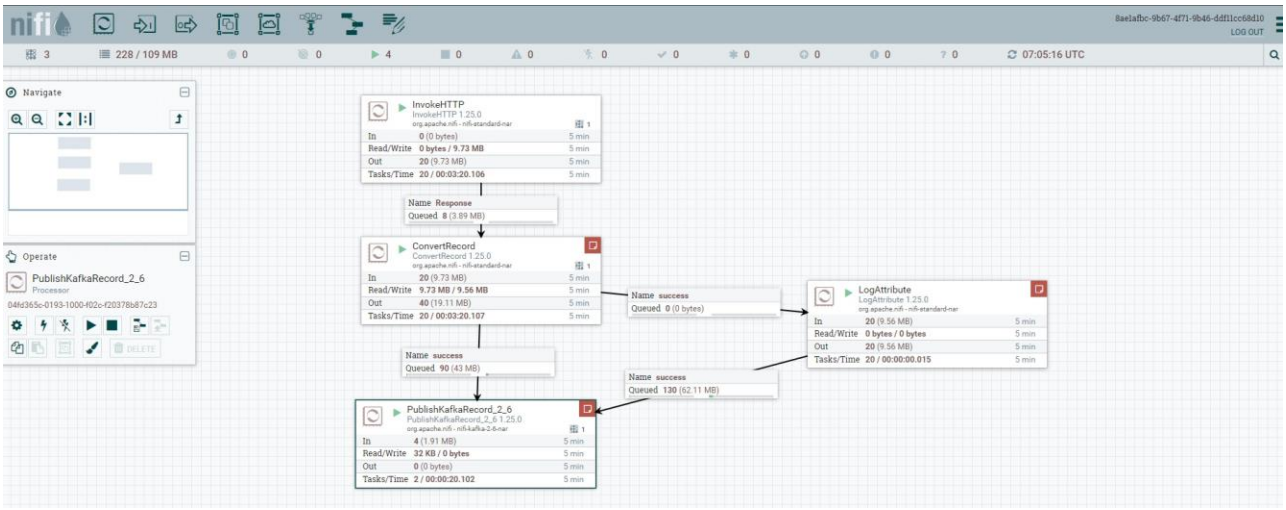


7. Big Data Architecture Design

The under mentioned big data architectural design of this project contain the following tools:

7.1. NiFi: Data Ingestion

Apache NiFi plays a pivotal role in automating the data ingestion process for this project. It efficiently extracts data from external sources and prepares it for real-time streaming. The process starts with the InvokeHTTP Processor, which is configured to fetch CSV data directly from a GitHub URL. This component allows the project to automate data retrieval, ensuring that the latest data is continuously ingested into the pipeline. Once the data is fetched, the ConvertRecord Processor transforms the raw CSV data into a structured format, making it compatible with downstream components. This transformation step standardizes the data, ensuring consistency and reliability. Finally, the structured data is sent to a Kafka topic using the PublishKafkaRecord Processor, where it can be streamed in real-time. The outcome of this setup is a fully automated and continuous data flow pipeline, setting the stage for real-time analytics and decision-making.



7.2. Kafka: Real-Time Data Streaming

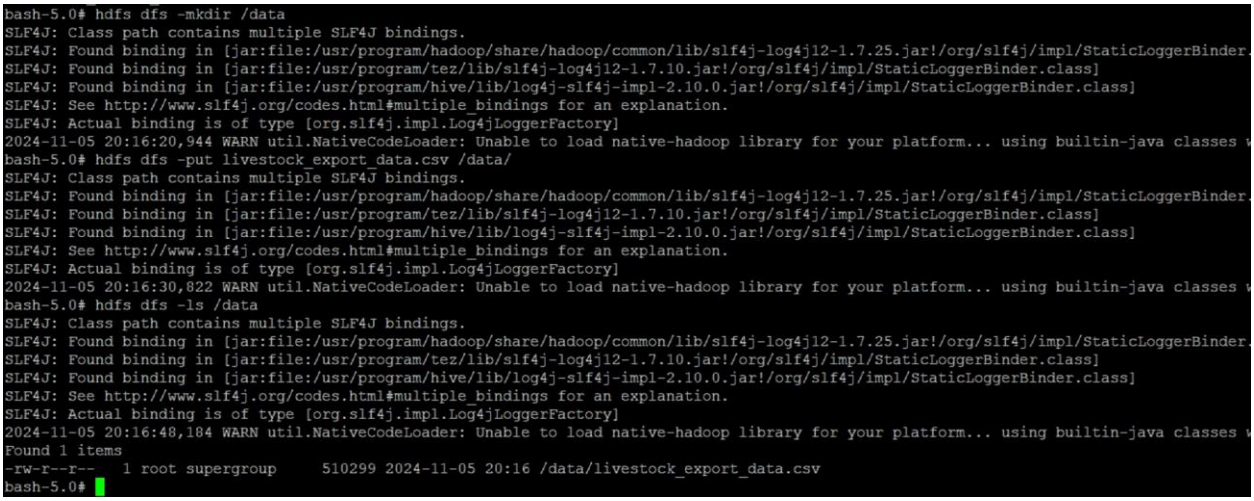
Apache Kafka serves as the backbone for real-time data streaming, efficiently managing data flow between various components in the architecture. Data streamed from NiFi is published to a Kafka topic named livestock-export-data, where it is buffered for real-time processing. Kafka's robust messaging system ensures high-throughput and low-latency data transmission, making it ideal for real-time analytics. The project leverages Kafka Topics to organize and segment the data, ensuring that only relevant information is processed downstream. Producers and Consumers play critical roles in this system: NiFi acts as a producer that continuously pushes data into Kafka, while PySpark serves as a consumer that ingests the data for further analysis. This architecture ensures that real-time data is processed efficiently, supporting near-instantaneous insights and analytics.

```
ZemelakGoraga@mybigdataproject:~/dsc650-infra/bellevue-bigdata/kafka$ docker-compose up -d
Creating network "kafka_default" with the default driver
Creating kafka_kafka_1     ... done
Creating kafka_zookeeper_1 ... done
ZemelakGoraga@mybigdataproject:~/dsc650-infra/bellevue-bigdata/kafka$ docker exec -it kafka_kafka_1 kafka-topics.sh --create --topic livestock-export-data --b
--partitions 1 --replication-factor 1
Created topic livestock-export-data.
ZemelakGoraga@mybigdataproject:~/dsc650-infra/bellevue-bigdata/kafka$
```

7.3. HDFS: Scalable Data Storage

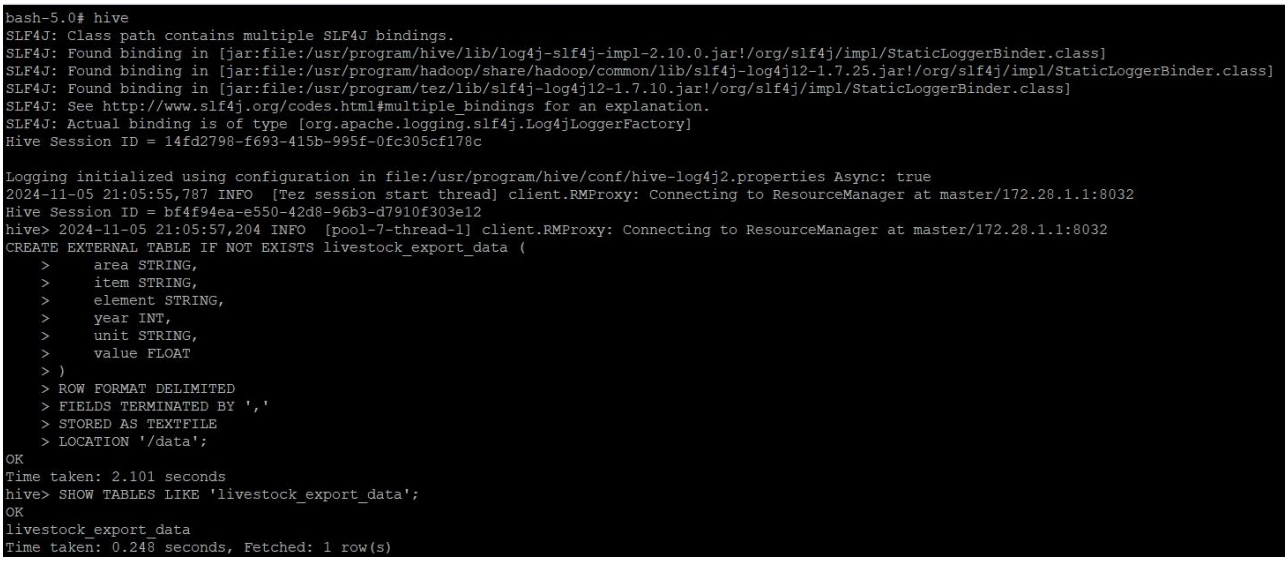
The Hadoop Distributed File System (HDFS) is utilized to store large volumes of raw and processed data, acting as a central repository for historical data analysis. HDFS is designed to handle massive datasets, providing scalability and fault tolerance. The project employs Data Partitioning to optimize read and write operations, ensuring that queries can be executed swiftly even on large datasets. In addition, Data Archiving is used to store historical data for long-term analysis, enabling the project to retain a comprehensive record of livestock export data over the

years. By using HDFS as the storage backbone, the project achieves a reliable and scalable solution for managing big data, making it accessible for both batch and real-time processing.



7.4. Hive: Data Warehousing and Querying

Apache Hive serves as the data warehousing layer, providing a SQL-like interface for querying data stored in HDFS. This integration allows analysts and data scientists to perform complex queries and aggregations on large datasets without needing to move data between systems. An External Table named livestocks_export_data is created in Hive, which directly references the data stored in HDFS. This approach ensures that the data remains consistent and accessible for structured querying. Using SQL Queries, stakeholders can efficiently explore, aggregate, and analyze the data to identify patterns and trends in cattle exports. Hive's capabilities transform raw data into a structured, queryable format, facilitating large-scale analytics and insights.

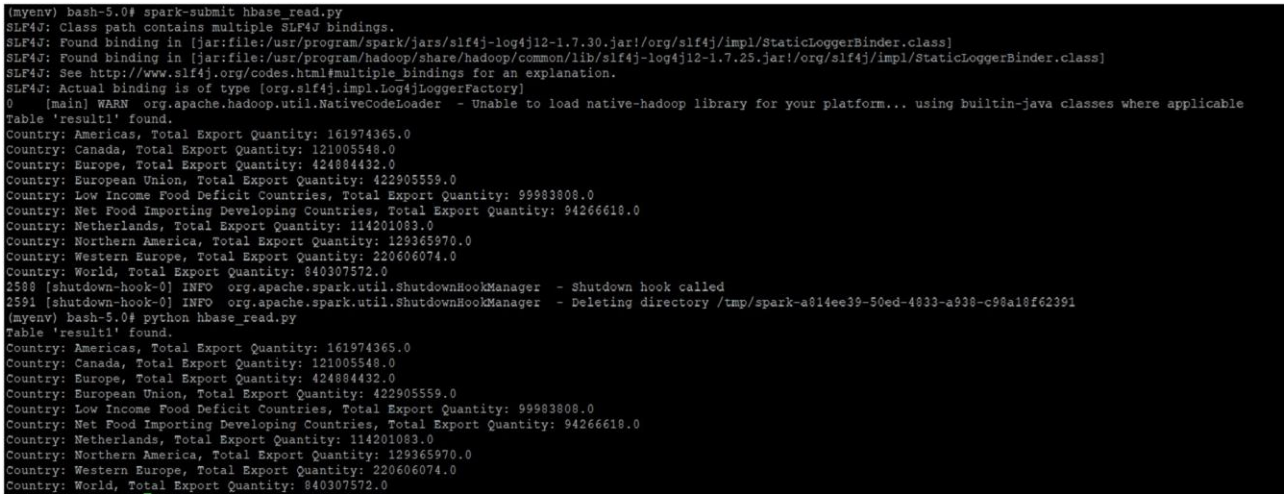


7.5. PySpark: Data Processing and Machine Learning

Apache Spark, specifically its PySpark module, is employed for large-scale data processing and machine learning tasks. After data is ingested and stored, Data Aggregation is performed using PySpark to analyze export trends by country and year. This aggregated data is then used to train a RandomForestRegressor model, which aims to predict future export quantities based on historical trends. PySpark's distributed computing capabilities allow the project to process large datasets quickly, making it ideal for predictive analytics. Once the model is trained, its performance is evaluated using metrics such as Root Mean Squared Error (RMSE) and R-squared (R^2), providing a quantitative measure of accuracy. This predictive modeling step generates actionable insights that stakeholders can use to optimize their strategies based on future forecasts.

7.6. HBase: Storing Model Metrics

Apache HBase, a NoSQL database, is utilized to store the results of descriptive statistics and evaluation metrics of the predictive models. This is crucial for tracking model performance over time and ensuring that the models remain accurate as new data becomes available. The project organizes metrics into Column Families, where each metric, such as RMSE and R^2, is stored in a structured format. By integrating PySpark with HBase, the project saves the model's evaluation results directly after training. This setup allows for efficient retrieval and comparison of metrics, enabling continuous monitoring and model tuning. The use of HBase ensures that stakeholders can quickly access performance data, supporting decision-making based on up-to-date model evaluations.



7.7. YARN: Resource Management

YARN (Yet Another Resource Negotiator) functions as the resource management layer, coordinating and optimizing resource allocation across the Hadoop cluster. It plays a vital role in managing the execution of distributed data processing tasks, such as PySpark jobs and Hive queries. YARN ensures that computational resources are efficiently utilized, minimizing bottlenecks and maximizing throughput. This resource management capability is essential for

maintaining high performance in a big data environment, especially when dealing with complex analytics workflows that require substantial computational power. By leveraging YARN, the project achieves efficient scheduling and execution of data processing tasks, enhancing overall system performance.

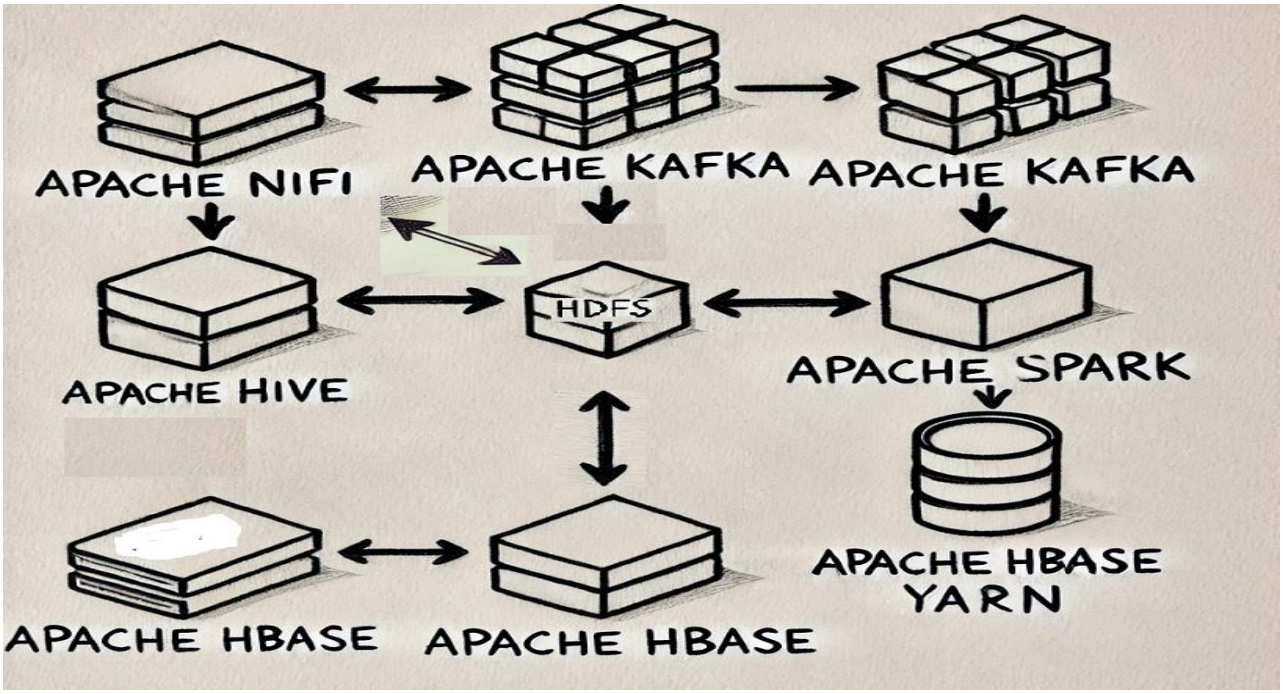


Fig. 1. Big Data Architectural Design

8. Results & Discussion

8.1. Descriptive Statistics

The recent analysis of the livestock export dataset using Hive revealed several critical insights, particularly focusing on the export quantities of live cattle. By exploring the data from 1961 to 2013, we identified key patterns and trends that provide a comprehensive understanding of global cattle trade dynamics.

Initially, the data exploration showcased that Argentina consistently exported significant quantities of cattle in the early years, with exports peaking in the 1960s and fluctuating thereafter. A deep dive into the top exporting regions identified: Europe, European Union, and the Americas as the leading

exporters. For instance, in 2010, Europe alone accounted for over 4.6 million heads, with the European Union contributing the majority.

The aggregated data highlighted that worldwide cattle exports peaked in 2010, with a total of over 10 million heads, driven by strong export performances in Western Europe and the Americas. This trend indicated an increasing global demand for cattle, particularly in the decade leading up to 2010. Analyzing yearly averages, it was observed that the export quantities grew steadily, especially from the early 2000s, aligning with global economic growth.

An analysis of individual countries' export trends showed that France and Australia were among the top contributors in the last decade. In particular, Australia maintained a stable export quantity, peaking at over 870,000 heads in 2010. The analysis also highlighted that Africa and Central America experienced gradual increases in cattle exports, possibly indicating a shift in trade focus towards these regions.

By leveraging aggregate queries and grouping by year, we identified that the average export quantity steadily increased from approximately 600,000 heads in the early 1960s to over 1 million heads by the mid-2000s. Notably, Brazil emerged as a significant exporter in the late 2000s, with exports rising dramatically in 2010.

Lastly, through advanced queries using Common Table Expressions (CTEs), the analysis pinpointed the top three exporting regions and their yearly export performance. This detailed breakdown provided a clear view of export trends over time, helping stakeholders understand how different regions adapted to global market demands. The inclusion of HBase for metric storage allowed efficient tracking of these insights, ensuring that future analyses could be benchmarked against historical data.

These results offer actionable insights for optimizing cattle trade strategies, guiding stakeholders in adapting to evolving market conditions, and supporting sustainable agricultural practices.

8.2. Modeling

The predictive analysis with PySpark was conducted by accessing data from both Hive for historical records. This integration enabled a comprehensive analysis that combined large-scale historical data, providing a robust view of the cattle export trends between 1998 and 2013.

The data preparation phase began with filtering and cleaning the dataset. Approximately 45,000 relevant rows were extracted from the Hive table, focusing on historical export quantities. The dataset was cleaned by replacing missing values with the median for numerical columns and

removing outliers, resulting in a more reliable dataset. This dataset was then aggregated by country and year to highlight key exporters like Australia, Brazil, and the United States.

For predictive modeling, a RandomForestRegressor was selected due to its robustness in handling large, complex datasets. The model was trained on the historical data split into 70% training and 30% testing subsets. Key features included the year, country, and newly engineered variables such as the log, square root, and square of export quantities. This approach aimed to capture non-linear relationships and enhance the model's predictive power.

The model evaluation metrics revealed strong performance:

R-squared (R^2): 0.92, indicating that the model explained approximately 92.48% of the variance in cattle export quantities. This high R^2 value suggests that the model effectively captured the underlying patterns in the data.

Root Mean Squared Error (RMSE): 438,350 units, reflecting the average deviation between predicted and actual values. While the RMSE appears large, its significance is relative to the scale of export quantities, which often reach millions. The model is effective overall but still has room for refinement to reduce prediction errors for certain data points.
These metrics validate the model's capability to forecast export trends accurately, particularly for stable exporters like Australia. However, fluctuations in Argentina's export data suggest areas for potential enhancement, such as incorporating additional economic and policy-related features.

To facilitate real-time monitoring of the model's performance, the evaluation metrics were seamlessly stored in Hive. This setup ensures efficient storage and retrieval of metrics like RMSE and R^2, allowing stakeholders to continuously track model performance, make timely adjustments, and ensure scalability. The integration of Hive with PySpark provides a scalable framework for real-time predictive analytics, enhancing the project's impact.

Implications of Findings
The analysis revealed significant trends and patterns in global cattle exports, emphasizing the importance of understanding regional export dynamics. The identification of Europe, European Union, and the Americas as top exporters highlights these regions' dominance in the cattle trade market, with opportunities for emerging regions like Africa and Central America to expand their market share. The peak in global exports in 2010 indicates a correlation with economic growth, suggesting that policy changes and economic conditions strongly influence export volumes.

The predictive model's strong performance with stable exporters like Australia provides stakeholders with reliable forecasts for strategic planning. However, fluctuations in exports from

countries like Argentina suggest that integrating more diverse variables (such as policy changes, economic indicators, or climate data) could further refine predictions and improve model accuracy.

9. Conclusions

This project effectively demonstrated the use of historical data analysis and predictive modeling to optimize strategies in the global cattle trade. By leveraging tools like Apache Hive and PySpark, combined with Hive table storage for efficient metric tracking, the project provided a robust solution to analyze livestock export data from 1961 to 2013.

The descriptive analysis offered valuable insights into the patterns and trends in cattle exports, identifying key exporting regions such as Europe, the Americas, and Australia. Notably, countries like Brazil and Argentina exhibited significant growth in export volumes during the 2000s. Understanding these regional dynamics is crucial for stakeholders aiming to optimize trade strategies and identify emerging markets.

The predictive modeling component utilized a RandomForestRegressor model, achieving an R^2 score of 0.92, indicating that the model explained approximately 92.48% of the variance in export quantities. The Root Mean Squared Error (RMSE) of 438,351 units reflected moderate prediction deviations. Despite this, the high R^2 value indicates that the model effectively captures the overall trend, particularly for stable exporters like Australia. However, fluctuations in markets such as Argentina highlight areas where further refinement is needed to improve prediction accuracy, especially in volatile environments.

Integrating Hive storage for model evaluation metrics ensured efficient tracking of model performance, allowing stakeholders to continuously monitor metrics like RMSE and R^2. This setup supports future scalability and adaptability, enabling timely adjustments based on evolving market data. By combining PySpark and Hive, the project established a scalable framework for real-time predictive analytics, empowering stakeholders to leverage historical insights for data-driven decision-making in the cattle trade industry.

10. Future Improvements

This project showcased significant success in analyzing historical livestock export data and forecasting future trends using advanced big data technologies. The integration of Hive, PySpark, and Hive table storage for model metrics was effective in handling large datasets and delivering actionable insights. However, while the predictive model achieved a high R^2 score of 0.92, the relatively high RMSE of 438,351 units indicates some inaccuracies, especially when forecasting volatile markets like Argentina. This suggests that the model could benefit from incorporating

additional variables, such as economic indicators, trade policies, and weather patterns, to capture market complexities more accurately.

Key Areas for Improvement:

Feature Engineering: Enhancing data granularity by integrating additional datasets, such as livestock health metrics, climate patterns, and socio-economic indicators, could improve model accuracy. Including macroeconomic factors may help explain variations in export trends.

Advanced Machine Learning Techniques: While the RandomForestRegressor performed well, exploring other algorithms like XGBoost, Gradient Boosting, or ensemble models could further optimize predictive accuracy. These models might better capture non-linear relationships and interactions among variables.

Model Interpretability: Implementing model interpretability techniques like SHAP (Shapley Additive Explanations) values would provide deeper insights into the factors influencing predictions. This would help stakeholders understand the model's decision-making process, build trust, and make more informed strategic decisions.

Real-Time Data Integration: Enhancing the system's real-time capabilities by incorporating streaming analytics tools, such as Apache Kafka, would allow for continuous updates to the model with the latest data. This would ensure that forecasts are more timely and reflective of current market conditions.

Continuous Model Retraining and Monitoring: Regular retraining of the model with new data and monitoring its performance over time are crucial to maintaining its accuracy and adaptability. Storing evaluation metrics in Hive tables ensures efficient tracking and facilitates quick adjustments to the model when needed.

Visualization and Dashboard Integration: Utilizing tools like Power BI or Tableau to visualize model outputs and predictions would provide stakeholders with intuitive, interactive dashboards. This would aid in communicating insights effectively and supporting data-driven decision-making.

11. Assumptions

This project was grounded in several key assumptions. It assumed that the historical data utilized was accurate, comprehensive, and reflective of real market conditions. The analysis was based on the belief that patterns identified from historical data would continue to be relevant for future

forecasting. The project also presumed that the data ingestion processes, data transformation, and machine learning models were functioning correctly and free from significant errors. Additionally, the reliability and consistency of real-time data streams were crucial, as the continuous updates to the predictive models depended on timely and accurate data flows to deliver actionable insights.

12. Issues Encountered and Resolutions

During the project, several challenges were faced. First, there were connectivity issues with HBase, particularly with the Thrift server on port 9090, which prevented storing evaluation metrics. This was resolved by reconfiguring the HBase Thrift service and ensuring that necessary ports were open. Additionally, a missing Python library (numpy) caused errors during the model training phase, which was fixed by installing the required packages. Lastly, handling large datasets led to memory bottlenecks, which were addressed by optimizing Spark configurations and filtering the dataset to remove outliers, thus enhancing model efficiency and runtime performance

13. Ethical Considerations

The project, centered on predictive analysis of the global cattle trade, proactively addressed various ethical considerations. Data privacy was strictly maintained, especially when handling potentially sensitive or proprietary information from stakeholders. To prevent any unfair competitive advantage, the project ensured the anonymity of specific countries or companies involved in cattle exports. Efforts were made to mitigate biases that could inadvertently influence trade decisions, particularly if historical data contained inherent inequities. Measures were also taken to prevent misinterpretations of model outputs that could lead to unintended economic consequences. Additionally, the project carefully considered the environmental and social implications of optimizing cattle exports, recognizing that intensified trade activities could pose risks to ecosystems and communities dependent on sustainable agricultural practices.

14. References

Apache Hive. (2023). The Apache Software Foundation. Retrieved from https://hive.apache.org/

Apache Kafka. (2023). The Apache Software Foundation. Retrieved from https://kafka.apache.org/

Apache NiFi. (2023). The Apache Software Foundation. Retrieved from https://nifi.apache.org/

Apache Spark. (2023). The Apache Software Foundation. Retrieved from https://spark.apache.org/

Breiman, L. (2001). Random forests. Machine Learning, 45(1), 5-32. https://doi.org/10.1023/A:1010933404324

Dean, J., & Ghemawat, S. (2008). MapReduce: Simplified data processing on large clusters. Communications of the ACM, 51(1), 107-113. https://doi.org/10.1145/1327452.1327492

FAO. (2021). FAOSTAT database: Global food and agriculture statistics. Food and Agriculture Organization of the United Nations. Retrieved from https://www.fao.org/faostat/en/#data

Han, J., Pei, J., & Kamber, M. (2011). Data Mining: Concepts and Techniques (3rd ed.). Morgan Kaufmann.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... & Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. Journal of Machine Learning Research, 12, 2825-2830. Retrieved from https://jmlr.org/papers/v12/pedregosa11a.html

Zaharia, M., Chowdhury, M., Franklin, M. J., Shenker, S., & Stoica, I. (2010). Spark: Cluster computing with working sets. Proceedings of the 2nd USENIX Conference on Hot Topics in Cloud Computing. Retrieved from https://www.usenix.org/conference/hotcloud-10/spark

15. Appendices: SQL & Python Codes

End to end coding to execute the proposed project

```
# Step 1: Set Up the Hadoop Ecosystem

# Navigate to the setup directory

cd ~/dsc650-infra/bellevue-bigdata/hadoop-hive-spark-hbase

docker-compose up -d

# Step 2: Data Ingestion with NiFi and Kafka

export JAVA_HOME=/usr/lib/jvm/java-11-openjdk-amd64

# Confirm that it's set correctly by checking:

echo $JAVA_HOME

# start NiFi

/bin/bash nifi-*/bin/nifi.sh start

# Start Kafka

# step 1

cd ~/dsc650-infra/bellevue-bigdata/kafka

# step 2

docker-compose up -d


# Step 4

# creating topic in kafka - livestock-export-data

docker exec -it kafka_kafka_1 kafka-topics.sh --create --topic livestock-export-data --bootstrap-server
localhost:9092 --partitions 1 --replication-factor 1

# Verify Topic Creation: You can confirm that the topic was created by listing all available topics:

docker exec -it kafka_kafka_1 kafka-topics.sh --list --bootstrap-server localhost:9092
```

Step 3: Download and Load Dataset into HDFS and Hive

Download dataset from GitHub
wget https://raw.githubusercontent.com/zemelak-s-goraga/DSC650/refs/heads/main/FinalProject/dataset/livestock_export_data.csv

Upload the dataset to HDFS
hdfs dfs -mkdir /data
hdfs dfs -put livestock_export_data.csv /data/

Verify if the data uploaded

hdfs dfs -ls /data

Step 4. Create Hive Table:

Enter the Hive Shell:

bash

hive

Create the External Hive Table: Paste the following SQL command into the Hive shell to create the table with the specified columns:

```
CREATE EXTERNAL TABLE IF NOT EXISTS livestock_export_data (
    area STRING,
    item STRING,
    element STRING,
    year INT,
    unit STRING,
    value FLOAT
)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
STORED AS TEXTFILE
LOCATION '/data/livestock_export_data'
TBLPROPERTIES ("skip.header.line.count"="1");
```

Verify

To check if the table livestock_export_data was successfully created in Hive, follow these steps:

Enter the Hive Shell:

bash
hive

Check for the Table:

SHOW TABLES LIKE 'livestock_export_data';

Describe the Table :

To confirm the table schema:

DESCRIBE livestock_export_data;

Exit Hive:

bash

exit

Descriptive code with hive setup

Filter for Cattle Data and the Specified Year Range:

```
SELECT area, year, value
FROM livestock_export_data
WHERE item = 'Cattle'
  AND year BETWEEN 1998 AND 2013;
```

Identify the Top 10 Countries by Export Quantity:

```
SELECT area, SUM(value) AS total_export
FROM livestock_export_data
WHERE item = 'Cattle'
  AND year BETWEEN 1998 AND 2013
GROUP BY area
ORDER BY total_export DESC
LIMIT 10;
```

working with spark

create analysis.py framework at bash

vi analysis.py

Edit the File Locally: copy and past the undermentioned PySpark codes

run the analysis.py file

spark-submit analysis.py

PySpark codes: PySpark Script with Modeling and HBase Integration

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import sum, col, when, log1p, sqrt, pow
from pyspark.ml.regression import RandomForestRegressor
from pyspark.ml.evaluation import RegressionEvaluator
from pyspark.ml.feature import VectorAssembler
import happybase

# Step 1: Initialize Spark session with Hive support
spark = SparkSession.builder \
    .appName("Livestock Export Analysis & Prediction") \
    .enableHiveSupport() \
    .getOrCreate()

# Step 2: Load the CSV file directly from HDFS
print("\nLoading data from HDFS...")
df = spark.read.option("header", "true") \
    .option("inferSchema", "true") \
    .csv("hdfs://master:9000/data/livestock_export_data.csv")

# Step 3: Replace NaN values with the median for numerical columns
numeric_cols = [col for col, dtype in df.dtypes if dtype in ['int', 'double']]
for col_name in numeric_cols:
    # Calculate the median using approxQuantile
    median_value = df.approxQuantile(col_name, [0.5], 0.01)[0]  # 0.01 is the relative error tolerance
    df = df.fillna({col_name: median_value})

df.show(5, truncate=False)
df.printSchema()

# Step 4: Filter data for 'Cattle' and 'Export Quantity' between 1998 and 2013
filtered_df = df.filter(
    (df.element == 'Export Quantity') &
    (df.item == 'Cattle') &
    (df.year.between(1998, 2013))
)

print("\nFiltered Data (1998-2013 for Cattle Export Quantity):")
filtered_df.show(5)

# Step 5: Create new variables (log, sqrt, and square of 'Export Quantity')
filtered_df = filtered_df.withColumn("log_export", log1p("value"))
filtered_df = filtered_df.withColumn("sqrt_export", sqrt("value"))
filtered_df = filtered_df.withColumn("squared_export", pow("value", 2))
```

```
# Step 6: Aggregate data by Country and Year with new features
country_year_agg = filtered_df.groupBy("area", "year").agg(
    sum("value").alias("Total_Export_Quantity"),
    sum("log_export").alias("Log_Total_Export"),
    sum("sqrt_export").alias("Sqrt_Total_Export"),
    sum("squared_export").alias("Squared_Total_Export")
)

print("\nAggregated Data with New Features:")
country_year_agg.show(5)

# Step 7: Prepare data for modeling
print("\nPreparing data for modeling...")
model_data = country_year_agg.select(
    col("year").alias("Year"),
    col("area"),
    col("Total_Export_Quantity").alias("value"),
    col("Log_Total_Export"),
    col("Sqrt_Total_Export"),
    col("Squared_Total_Export")
)

# Step 8: Assemble features for the model
vector_assembler = VectorAssembler(inputCols=["Year", "value", "Log_Total_Export", "Sqrt_Total_Export",
"Squared_Total_Export"], outputCol="features")
model_df = vector_assembler.transform(model_data).select("features", col("value").alias("label"))

# Step 9: Split data into training and testing sets
train_data, test_data = model_df.randomSplit([0.7, 0.3], seed=42)

# Step 10: Initialize and train the RandomForestRegressor model
print("\nTraining RandomForestRegressor model...")
rf = RandomForestRegressor(featuresCol="features", labelCol="label", numTrees=100, seed=42)
model = rf.fit(train_data)

# Step 11: Make predictions and evaluate the model
print("\nEvaluating model...")
predictions = model.transform(test_data)
predictions.select("features", "label", "prediction").show(5)

# Calculate evaluation metrics
evaluator = RegressionEvaluator(labelCol="label", predictionCol="prediction")
rmse = evaluator.evaluate(predictions, {evaluator.metricName: "rmse"})
r2 = evaluator.evaluate(predictions, {evaluator.metricName: "r2"})
print(f"\nModel Evaluation Metrics:\nRoot Mean Squared Error (RMSE): {rmse}\nR-squared (R2): {r2}")

# Step 12: Write performance metrics to HBase
print("\nWriting performance metrics to HBase...")

try:
    # Establish connection to HBase
    hbase_connection = happybase.Connection('localhost')
    table_name = 'model_metrics'

    # Check if the table exists; if not, create it
    if table_name.encode() not in hbase_connection.tables():
```

```
        print(f"Creating table '{table_name}'...")
        hbase_connection.create_table(table_name, {'metrics': dict()})

    table = hbase_connection.table(table_name)

    # Store the metrics in HBase
    table.put(b'cattle_export_quantity_prediction', {
        b'metrics:rmse': str(rmse).encode(),
        b'metrics:r2': str(r2).encode()
    })

    print("\nMetrics successfully written to HBase.")

    # Retrieve and display metrics from HBase
    print("\nRetrieving metrics from HBase...")
    for key, data in table.scan():
        print(f"Row Key: {key}, Data: {data}")

except Exception as e:
    print(f"Error interacting with HBase: {e}")

finally:
    # Cleanup
    if 'hbase_connection' in locals():
        hbase_connection.close()
    spark.stop()
```

Explanation on PySpark Codes

Step 1: Initialize Spark Session
Creates a Spark session with Hive support to leverage Spark's distributed data processing capabilities. This step sets up the environment to load, transform, and analyze data efficiently, making use of both Hive tables and HDFS storage.

Step 2: Load CSV Data from HDFS
Reads the CSV file containing livestock export data directly from HDFS, inferring the schema and using the first row as headers. This allows seamless integration with distributed storage and makes the data available for further processing.

Step 3: Replace NaN Values with Median
Identifies numerical columns and fills any missing (NaN) values with the median. This ensures that the dataset remains complete and ready for analysis, avoiding issues caused by missing values during model training.

Step 4: Filter Data for 'Cattle' and Specific Years
Filters the dataset to include only rows where the element is "Export Quantity" and item is "Cattle". It also restricts the analysis to data from the years 1998 to 2013, focusing on relevant timeframes.

Step 5: Create New Variables
Generates new columns based on the original value column: logarithm, square root, and square. These transformations help capture non-linear patterns and enhance the predictive power of the model by providing additional features.

Step 6: Aggregate Data by Country and Year

Groups the filtered data by area (country) and year, computing the sum of the original and newly created variables. This aggregation step simplifies the dataset, making it more manageable for model training.

Step 7: Prepare Data for Modeling
Selects relevant columns for the model, including the newly created variables. This prepares the data for the next step, where the features will be transformed into a format suitable for machine learning algorithms.

Step 8: Assemble Features for the Model
Uses a VectorAssembler to combine multiple columns into a single features vector. This is required for machine learning models in PySpark, which expect a unified feature vector as input.

Step 9: Split Data into Training and Testing Sets
Splits the dataset into training (70%) and testing (30%) subsets to evaluate the model's performance. This ensures that the model is trained on one portion of the data and tested on another for validation.

Step 10: Train the RandomForestRegressor Model
Trains a Random Forest regression model using the training dataset. This algorithm is effective for handling high-dimensional data and non-linear relationships, making it suitable for predicting export quantities.

Step 11: Make Predictions and Evaluate the Model
Uses the trained model to make predictions on the test dataset. Evaluates model performance using Root Mean Squared Error (RMSE) and R-squared (R2) metrics, providing insight into the model's accuracy and fit.

Step 12: Write Performance Metrics to HBase
Connects to HBase to store the evaluation metrics (RMSE and R2). If the table does not exist, it creates one. It then writes the metrics to the table and retrieves them for verification, ensuring the results are stored persistently for future analysis.