Awarding Body:	Arden University (Berlin)
Programme Name:	Big Data and Cloud Computing
Module Name (and Part if applicable):	COM7020
Assessment Title:	RetailChain Big Data Analytics and Decision Support Project
Student Number:	24169956
Tutor Name:	Prof. Mohammad Amin Mohammadi Banadaki
Word Count:	4,554

Please refer to the Word Count Policy on your Module Page for guidance

Table of Contents

Introduction

The swift expansion of data in contemporary retail settings has revolutionized the way in which organizations make operational and strategic choices. Every day, retail businesses produce vast amounts of external, operational, and transactional data. Distributed processing frameworks and scalable big data technologies are necessary for efficiently handling and analyzing massive data. High-volume and high-variety retail datasets are frequently too big for traditional data processing systems, particularly when analytical activities like trend forecasting and predictive modeling are needed.

The RetailChain scenario, which uses Hadoop and Apache Spark technologies to analyze massive retail sales data, is the subject of this research. The study shows how machine learning-based forecasting, exploratory analytics, and data integration may be supported by big data infrastructure. The real-world solution makes use of Apache Spark for distributed data processing and predictive modeling, as well as the Hadoop Distributed File System (HDFS) for storage.

To improve sales trend forecasting and decision-making, this research aims to critically assess big data technologies in a retail setting and to present a practical analytical solution. The study combines theoretical analysis with real-world application to provide a thorough understanding of big data strategy and solution development.

1.1 Background of RetailChain Scenario

RetailChain is a retail company with multiple stores that operate in various departments and locations. Weekly sales transactions, store-level data, promotional effects, holidays, and external economic indicators like the consumer price index (CPI), gasoline costs, and unemployment rate are just a few of the kinds of data that the organization generates.
The primary data includes:

- Weekly sales data for several departments and retailers
- Stores metadata
- Seasonal and external economic factors

RetailChain has problems with scalability and integration because the data comes from multiple sources. For large-scale analytics, local processing is ineffective due to the comparatively vast and constantly expanding datasets.
A distributed big data environment was created to solve this issue using:

- For distributed storage, use Hadoop (HDFS).
- Distributed computation using Apache Spark
- Configuring a Docker-based cluster for deployment

Scalable data intake, integration, and advanced analytics are made possible by this architecture in a big data enterprise simulation.

1.2 Business Problem and Decision-Making Challenges

RetailChain wants to use data-driven insights to enhance its decision-making skills. Understanding the effects of many elements on shop performance and effectively forecasting weekly sales are the main business challenges.
The organization faces several decision-making challenges:

i. **Complexity of Sales Forecasting**
   Stores, departments, seasons, and external economic conditions all have an impact on sales performance. It takes sophisticated analytical techniques to determine the main factors influencing sales.
ii. **Volume and Integration of Data**
   Data complexity rises when external indications are combined with transactional data. It takes distributed processing to manage this integration at scale.
iii. **Holiday and Seasonal Impacts**
   Holidays and other seasonal events have a big impact on retail sales. These dynamic patterns might not be well captured by conventional analysis techniques.
iv. **Limitations on Scalability**
   Processing performance becomes an issue as data volume increases over time. Consistent analytical performance necessitates a scalable infrastructure.

1.3 Objectives of the Report

Designing, implementing, and assessing a big data analytics solution for the RetailChain scenario is the main goal of this research.
Among the goals are:

- To examine the big data features and retail data environment
- To assess suitable architectures for massive data processing
- To create a distributed storage system based on Hadoop
- To use HDFS for data loading and integration
- To use Apache Spark for exploratory data analysis
- To create a regression model using machine learning for sales forecasting
- To assess model performance with suitable metrics
- To determine the significance of a characteristic and obtain business information
- To evaluate future improvements, deployment issues, and scalability

These goals guarantee that theoretical analysis and real-world application are in line.

1.4 Structure of the Report

There are two primary sections to this report.

Part I discusses big data strategy and technology evaluation. It looks at the big data features, processing methods, cloud computing models, governance issues, and the RetailChain data ecosystem. The theoretical underpinnings for choosing suitable technologies are presented in this section.

Part II describes the practical implementation and analytical solution. It explains how to build up a Hadoop cluster, load data into HDFS, process data using Spark, undertake exploratory analysis, create machine learning models, and assess them. Scalability issues, deployment tactics, and critical analysis of the applied solution round up the section. Key findings are summarized in the report's conclusion, which is followed by references and appendices with the whole source code and execution guidelines.

PART I—RetailChain Data Strategy and Technology Evaluation

2.1 RetailChain Data Environment and Business Context

Effective handling of transactional and contextual data is essential for both operational efficiency and strategic decision-making in the very data-intensive retail environment in which RetailChain operates. A simplified model of a larger company data ecosystem is represented by the core datasets used in this analysis, which include sales, store information, and external attributes. Point-of-sale (POS) systems, supply chain systems, inventory management platforms, marketing campaigns, customer loyalty programs, and external economic indicators all produce data on a constant basis in real-world retail operations.

Weekly sales at the department and store levels are recorded in the sales dataset, which shows trends in consumer demand. Contextual variables including temperature, fuel prices, the consumer price index (CPI), the unemployment rate, and holiday indicators are all included in the features dataset. Store identification and classification-related structural variables are included in the shops dataset. When combined, these datasets show how external macroeconomic and environmental factors are integrated with internal transactional data.

However, there are a number of difficulties in integrating such disparate data sources. Integration is made more difficult by variations in data formats, update schedules, and schema architectures. Careful preparation is necessary to ensure that weekly sales data and external indicators are in sync over time. Furthermore, validation and cleaning procedures are necessary to guarantee data consistency across departments and locations. Inconsistencies could produce untrustworthy analytical results in the absence of organized integration procedures.

Notwithstanding these difficulties, there are a lot of opportunities for data-driven decision-making when different datasets are integrated. RetailChain can predict future revenue trends, evaluate the effect of holidays and economic conditions on sales, and identify seasonal demand swings by integrating operational and contextual data. This serves as the cornerstone for strategic planning, performance optimization across stores, and predictive analytics.

RetailChain's data environment, which necessitates scalable processing technology and organized governance frameworks, thereby reflects the strategic potential and complexity typical of contemporary retail ecosystems.

2.2 Big Data Characteristics in the Retail Context (5 V's)

The data environment at RetailChain demonstrates important Big Data traits that are typically characterized by the following five dimensions: Volume, Velocity, Variety, Veracity, and Value.

i. **Volume** results from the aggregation of weekly sales transactions over long periods of time across several departments and locations. This volume would increase to millions of transactions per day in a real-world retail context, requiring distributed processing and storage solutions like HDFS and Spark.
ii. **Velocity** is the rate at which information is created and needs to be processed. Retail sales are ongoing, and dynamic pricing, inventory replenishment, and promotional modifications all depend on timely analysis. Retail systems increasingly need near real-time analytical capabilities, even if the existing design concentrates on batch processing.
iii. **Variety** is evident in the integration of structured transactional data with environmental and external economic indicators. To further increase data diversity, retail businesses frequently integrate semi-structured and unstructured data, including social media signals, online behavior logs, and consumer evaluations.
iv. **Veracity** pertains to the correctness and dependability of data. Missing figures, inconsistent entries, or recording errors can all be found in retail data. Having high-quality data is crucial for forecasting models to be dependable. Schema validation and cleaning techniques reduce the hazards of erroneous data during preprocessing.
v. **Value** is the term used to describe the strategic insights obtained by data analytics. The end goal is actionable intelligence that supports demand forecasting, cost optimization, and increased profitability—not data collection.

These features support RetailChain's use of parallel processing frameworks (Spark) and distributed storage (HDFS), guaranteeing scalability and analytical effectiveness as data complexity rises.

2.3 Big Data Processing Architectures

Batch processing and real-time streaming are the two main architectural paradigms that control massive data processing.

At predetermined times, gathered data is processed using a batch processing architecture. Large-scale aggregations, forecasting models, and historical trend analysis are all excellent uses for it. Large datasets stored in HDFS can be subjected to distributed batch analytics using technologies like Hadoop MapReduce and Apache Spark. Batch systems are suitable for weekly sales forecasting, as used in this project, because they prioritize throughput over latency.

Real-time streaming architecture, on the other hand, continuously processes data as it is produced. Near-instantaneous data input and analysis are made possible by frameworks like Spark Streaming and Apache Kafka. Real-time systems are very useful for dynamic pricing, fraud detection, and real-time inventory updates.

A comparative analysis shows that whereas streaming architectures offer instant responsiveness but necessitate more intricate infrastructure and operational administration, batch processing offers simplicity, stability, and cost effectiveness for periodic analytical activities.

Considering that RetailChain's goal is to predict weekly sales trends rather than make snap choices at the transaction level, a batch-oriented distributed architecture makes more sense. This requirement is met by using Apache Spark for parallel processing and the Hadoop Distributed File System (HDFS) for storage. Scalability is offered by this architecture while the system complexity is kept under control.

2.4 Cloud Computing and Distributed Platform Evaluation

Big data processing has been revolutionized by cloud computing, which offers scalable infrastructure without requiring on-premises hardware investments. Pay-as-you-go models and elastic resource allocation are made possible by the transition from traditional data centers to cloud-based distributed systems.

Generally, cloud service models are divided into:

- **Infrastructure as a Service (IaaS):** Offers computer resources that are virtualized.
- **Platform as a Service (PaaS):** Provides platforms for analytics and managed development.
- **Software as a Service (SaaS):** Provides software programs that are completely managed.

Managed Hadoop clusters and Spark services are part of integrated big data ecosystems offered by major cloud providers like AWS, Microsoft Azure, and Google Cloud Platform. These platforms improve scalability and dependability while lowering operational expenses.

A more thorough evaluation of managed big data platforms reveals significant strategic trade-offs for RetailChain in addition to the generic advantages of cloud computing. Big data services like AWS EMR, Azure HDInsight, and Google Dataproc are entirely managed by major cloud providers including Amazon Web Services (AWS), Microsoft Azure, and Google Cloud Platform. These services offer scalable Hadoop and Spark clusters while abstracting away infrastructure maintenance. Managed services drastically lower hardware investment, maintenance costs, and operational complexity when compared to a self-managed on-premise Hadoop system.

Traditional on-premise deployments are expensive from a financial standpoint since they need a large capital expenditure (CAPEX) for networking infrastructure, hardware acquisition, and continuing maintenance. On the other hand, cloud-based services use an operating expenditure (OPEX) model, in which resources are made available as needed and charged depending on utilization. By dynamically assigning compute power, elastic scaling provides cost efficiency for RetailChain, whose analytical workloads may vary seasonally (for example, more processing during holiday analysis).

Reliance on managed cloud platforms, however, comes with possible hazards such data sovereignty issues and vendor lock-in. Large-scale distributed dataset migrations across providers can be expensive and complicated. Additionally, cross-border data storage may be limited by regulatory constraints. A balanced solution might be offered by a hybrid cloud strategy, in which analytical tasks take advantage of cloud scalability while sensitive datasets stay on-premise.

The Docker-based Hadoop cluster used in this project replicates a distributed environment that is ready for an enterprise setting for RetailChain. However, managed cloud services would be advantageous for production-scale deployment in order to provide enterprise-grade security, automatic failover, high availability, and service-level agreements (SLAs). As a result, cloud computing improves scalability and reliability, but it needs to be carefully assessed within frameworks for compliance and organizational control.

2.5 Governance, Security, and Ethical Considerations

Effective data governance ensures that RetailChain's analytical capabilities operate within secure and ethical boundaries. Retail datasets often contain sensitive transactional information and, in broader contexts, potentially identifiable customer data. Compliance with data protection regulations is therefore essential.

Data privacy mechanisms should include anonymization techniques, role-based access controls, and encryption protocols. HDFS supports permission-based access control, while containerization provides an additional layer of environmental isolation.

Security considerations extend beyond technical measures to include network configurations, authentication mechanisms, and controlled access to analytical environments. Ensuring restricted administrative privileges minimizes risks associated with unauthorized data manipulation.

Ethical considerations are equally important in predictive analytics. Sales forecasting models must avoid biased assumptions that could lead to discriminatory pricing strategies or unfair operational decisions. Transparency in model interpretation and feature importance analysis enhances accountability.

Finally, maintaining high data quality through governance frameworks ensures reliability in strategic decision-making. Clear documentation of data sources, preprocessing steps, and analytical methodologies supports auditability and reproducibility.

Collectively, governance, security, and ethical oversight form the foundation for responsible big data implementation within RetailChain.

PART II—Solution Design, Demonstration, and Reflection

3.1 Proposed Big Data Analytics Solution

3.1.1 Business Use Case and Problem Formulation

The goal of RetailChain is to increase the accuracy of sales forecasting across several departments and locations. Predicting weekly sales using past transaction records and external factors such as holidays, economic indicators, and store attributes is the company's main goal. Strategic promotional planning, better staffing, and inventory management are all made possible by accurate forecasting.

Weekly_Sales is the goal variable in the supervised regression task that is used to frame the analytical issue. Store, Department, Holiday Indicator, CPI, unemployment rate, fuel price, and other constructed temporal aspects are examples of independent variables. To effectively handle massive transactional data while preserving analytical scalability, the method uses distributed processing.

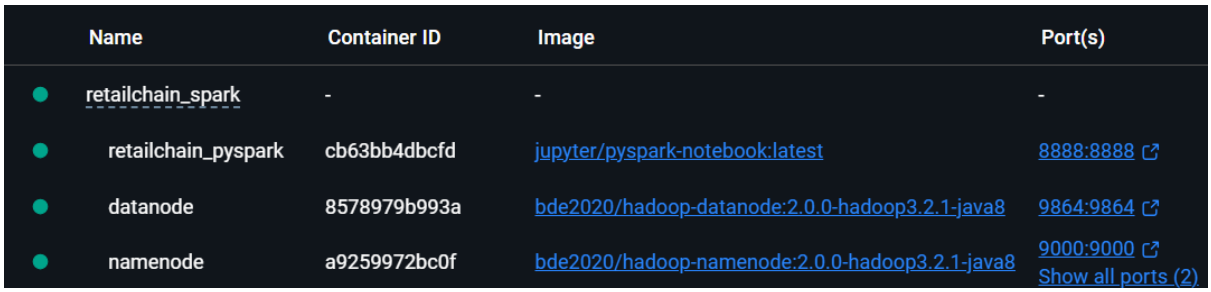
By determining the main causes of sales volatility and measuring their effect on revenue performance, the forecasting model aids in operational decision-making.

3.1.2 System Architecture and Technology Justification

The solution architecture combines Apache Spark for distributed data processing and analytics with Hadoop Distributed File System (HDFS) for storage. Docker containers were used to deploy the Hadoop cluster, complete with a NameNode and DataNode configuration. Distributed storage management was made possible by uploading transactional datasets to HDFS under a structured directory (/retailchain/data).

Because of its in-memory distributed processing capabilities, which greatly enhance performance for jobs involving machine learning and iterative analytics, Apache Spark was chosen. To make data exploration, transformation, and model creation easier, PySpark was utilized in a Jupyter Notebook environment.

This architecture is in line with enterprise-level analytics frameworks and represents a contemporary big data ecosystem that combines scalable processing (Spark) and distributed storage (HDFS).



Name	Container ID	Image	Port(s)
retailchain_spark	-	-	-
retailchain_pyspark	cb63bb4dbcfd	jupyter/pyspark-notebook:latest	8888:8888
datanode	8578979b993a	bde2020/hadoop-datanode:2.0.0-hadoop3.2.1-java8	9864:9864
namenode	a9259972bc0f	bde2020/hadoop-namenode:2.0.0-hadoop3.2.1-java8	9000:9000 Show all ports (2)

Figure 3.1.2.1: Docker-based Hadoop and Spark Cluster Deployment

As shown in Figure 3.1.2.1, the Hadoop cluster was deployed using Docker containers, including NameNode, DataNode, and PySpark notebook services.

Figure 3.1.2.2 illustrates the successful ingestion of datasets into HDFS with a replication factor of three, ensuring fault tolerance and distributed storage reliability.

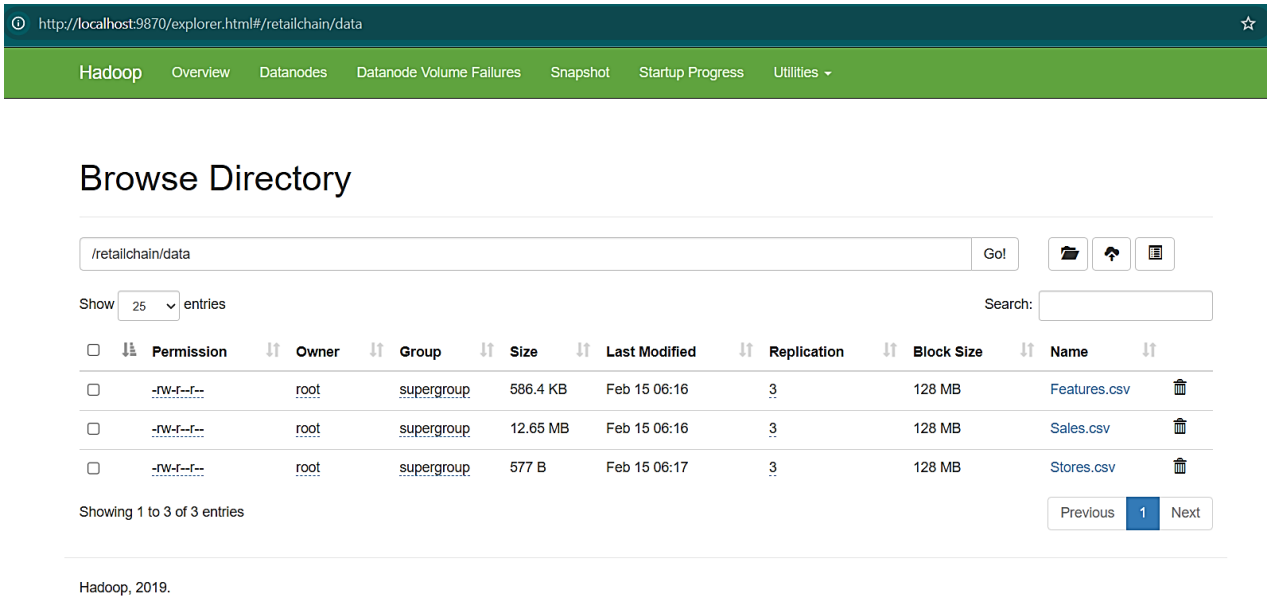


Figure 3.1.2.2: HDFS Directory Structure and Data Replication

3.2 Practical Implementation Using Hadoop and Apache Spark

3.2.1 Hadoop Cluster Setup and HDFS Data Loading

Docker Compose was used to set up a Hadoop cluster with many containers. Among the cluster were:

- NameNode (8020 RPC, 9870 PORT for web UI)
- DataNode—Distributed storage
- Spark-enabled environment for Jupyter

Datasets (Sales.csv, Features.csv, Stores.csv) were moved into the Hadoop NameNode container via docker cp and then uploaded to HDFS using the following commands:

hdfs dfs -mkdir -p /retailchain/data

hdfs dfs -put Sales.csv /retailchain/data

hdfs dfs -put Features.csv /retailchain/data

hdfs dfs -put Stores.csv /retailchain/data

Data availability was verified using the following command:

hdfs dfs -ls /retailchain/data

This illustrates a useful comprehension of distributed file system functionality.

Dataset	Rows	Size	Description
Sales	421,570	12.65 MB	Weekly store sales
Features	8,190	586 KB	Economic & holiday indicators
Stores	45	577 B	Store metadata

Table 3.2.1.1: HDFS Dataset Summary

3.2.2 Data Integration and Preparation

Data was loaded from HDFS into Spark using:

Spark.read.csv("hdfs://namenode:8020/retailchain/data/Sales.csv", header=True, inferSchema =True)

The datasets were merged using Date and Store as key attributes. Schema validation and row counts were conducted to confirm successful loading.

The following steps were included in Data preprocessing:

- Handling missing values.
- Casting date columns to timestamp format.
- Filtering invalid records.
- Identify and remove duplication.
- Identifying categorical variables.
- Creating time-based features (year, month).

A single analytical dataset appropriate for model training and exploratory analysis was produced because of the integration process.

The integration process leveraged Spark's distributed DataFrame API to perform joins across large datasets stored in HDFS. Distributed joins enable parallel matching of keys such as Store and Date across multiple partitions, significantly improving performance compared to single-machine processing. Schema validation ensured that column data types were consistent before transformation, reducing the risk of runtime failures during aggregation and modeling.

3.2.3 Feature Engineering and Dataset Construction

The goal of feature engineering was to enhance prediction performance. New variables like:

- Month extracted from Date
- Holiday impact indicator
- Store type encoding
- Interaction features between holiday flags and economic indicators.

The final dataset was prepared for modeling and is known as the "Gold Dataset."

Insights from the Gold Dataset are shown in **Figure 3.2.3.1, 3.2.3.2, and 3.2.3.3.**

```
Gold rows: 421570
+-----+----+------+----+----------+------------+-----------+----------+-----------+------------+
|Store|Type|Size  |Dept|Date      |Weekly_Sales|Temperature|Fuel_Price|CPI        |Unemployment|
+-----+----+------+----+----------+------------+-----------+----------+-----------+------------+
|1    |A   |151315|1   |2010-02-05|24924.5     |42.31      |2.572     |211.0963582|8.106       |
|1    |A   |151315|1   |2010-02-12|46039.49    |38.51      |2.548     |211.2421698|8.106       |
|1    |A   |151315|1   |2010-02-19|41595.55    |39.93      |2.514     |211.2891429|8.106       |
|1    |A   |151315|1   |2010-02-26|19403.54    |46.63      |2.561     |211.3196429|8.106       |
|1    |A   |151315|1   |2010-03-05|21827.9     |46.5       |2.625     |211.3501429|8.106       |
+-----+----+------+----+----------+------------+-----------+----------+-----------+------------+
only showing top 5 rows
```

Figure 3.2.3.1: Final Integrated RetailChain Dataset after Joining Sales, Features, and Stores Data

```
root
 |-- Store: integer (nullable = true)
 |-- Date: string (nullable = true)
 |-- IsHoliday: boolean (nullable = true)
 |-- Dept: integer (nullable = true)
 |-- Weekly_Sales: double (nullable = true)
 |-- Temperature: double (nullable = true)
 |-- Fuel_Price: double (nullable = true)
 |-- MarkDown1: string (nullable = true)
 |-- MarkDown2: string (nullable = true)
 |-- MarkDown3: string (nullable = true)
 |-- MarkDown4: string (nullable = true)
 |-- MarkDown5: string (nullable = true)
 |-- CPI: string (nullable = true)
 |-- Unemployment: string (nullable = true)
 |-- Type: string (nullable = true)
 |-- Size: integer (nullable = true)
```

Figure 3.2.3.2: Spark Schema Output Showing Data Types and Structure of the Gold Dataset

Store	Date	IsHoliday	Dept	Weekly_Sales	Temperature	Fuel_Price	MarkDown1
1	2010-02-05	false	1	24924.5	42.31	2.572	0.0
1	2010-02-12	true	1	46039.49	38.51	2.548	0.0
1	2010-02-19	false	1	41595.55	39.93	2.514	0.0
1	2010-02-26	false	1	19403.54	46.63	2.561	0.0
1	2010-03-05	false	1	21827.9	46.5	2.625	0.0

MarkDown2	MarkDown3	MarkDown4	MarkDown5	CPI	Unemployment	Type	Size
0.0	0.0	0.0	0.0	211.0963582	8.106	A	151315
0.0	0.0	0.0	0.0	211.2421698	8.106	A	151315
0.0	0.0	0.0	0.0	211.2891429	8.106	A	151315
0.0	0.0	0.0	0.0	211.3196429	8.106	A	151315
0.0	0.0	0.0	0.0	211.3501429	8.106	A	151315

Figure 3.2.3.3: Sample Records from the Final Integrated (Gold) Dataset

By storing categorical storage properties and extracting temporal components from date data, feature engineering improved prediction performance. In order to create a single feature vector that Spark ML methods demand, several predictors were combined using VectorAssembler. This method guarantees interoperability with distributed training processes and standardizes model input.

3.3 Exploratory Data Analysis and Business Insights

3.3.1 Exploratory Findings

The exploratory analysis found significant variation in weekly sales between stores and departments. Some store types consistently outperform others, according to store-level analysis. We saw seasonal patterns, especially around the holidays.

Figure 3.3.1.1: Monthly aggregated sales trend for RetailChain (2010-2012)

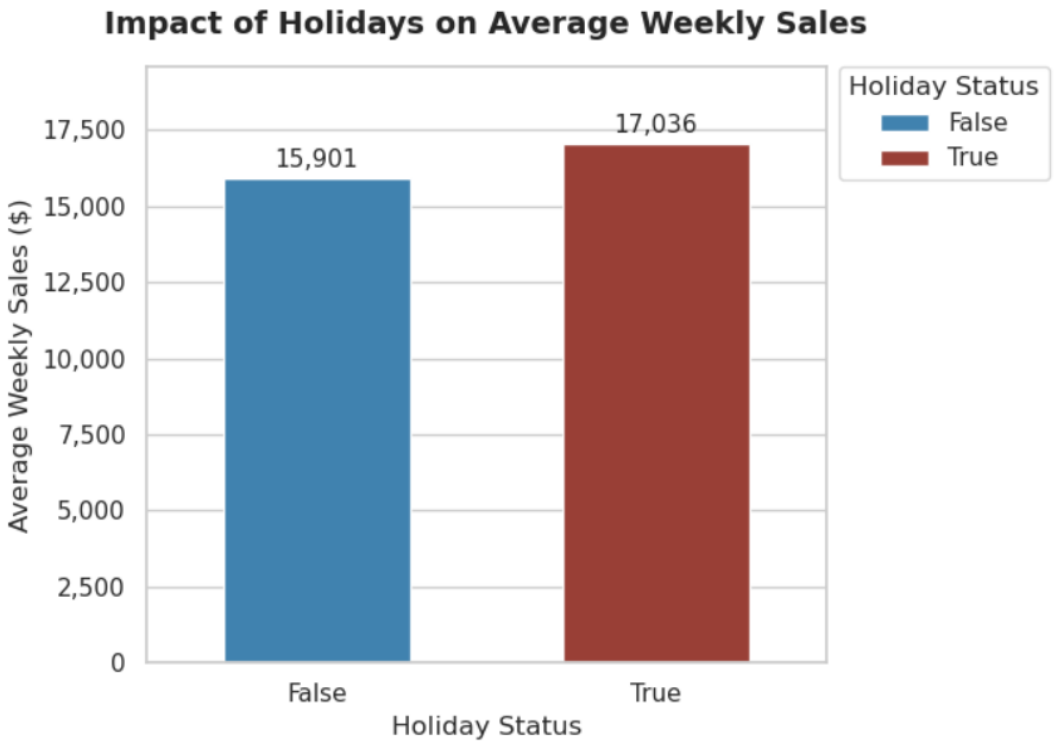


Figure 3.3.1.2: Impact of Holidays on Avg Weekly Sales ($)

The bar chart compares the average weekly sales during holiday and non-holiday seasons. According to the findings, average weekly sales during holiday weeks are higher at $17,036 USD compared to $15,901 USD during non-holiday weeks. This indicates a roughly 7% rise in sales throughout the course of the holidays. The observed discrepancy indicates that holidays have a beneficial impact on consumer spending behavior, most likely due to a rise in promotional activity, seasonal demand, and foot traffic. In order to optimize revenue performance during holiday seasons, these results emphasize the significance of careful inventory planning and focused marketing initiatives.

IsHoliday	AvgWeeklySales	TotalSales	RowCount
True	17035.82318735039	5.052995515599	29661
False	15901.445069008774	6.231919435550	391909

Table 3.3.1.1: Summary statistics of holiday vs non-holiday sales performance

The monthly total showed cyclical sales trends, with noticeable increases during marketing seasons. Compared with non-holiday periods, average sales during holiday weeks were statistically higher.

3.3.2 Business Interpretation

According to the data, store segmentation and promotional timing have a significant impact on revenue. Bigger stores provide more growth potential but also more sales volatility. There was a moderate link between sales performance and economic indices like the CPI and unemployment.

Management receives relevant knowledge from these insights for operational planning and targeted marketing.

Store ID	*TotalSales*
20	3.0139779245999974E8
4	2.9954395337999994E8
14	2.889999113399997E8
13	2.865177037999985E8
2	2.753824409800004E8
10	2.7161771388999999E8
27	2.538559168800001E8
6	2.2375613063999987E8
1	2.224028088499993E8
39	2.0744554247000024E8

Table 3.3.2.1: Store-level total sales ranking

3.4 Machine Learning Model Development and Evaluation

3.4.1 Model Design and Pipeline Construction

A regression task that predicted Weekly_Sales was used to model the issue. Data sets for testing and training were separated. A Spark ML pipeline was built using the following components:

- Feature Vector Assembler
- Feature Scaling
- Random Forest Regressor

Random Forest was chosen because it can describe nonlinear interactions and is robust to multicollinearity.

The Random Forest regression algorithm was chosen because of its capacity to represent intricate, nonlinear relationships in retail datasets. Random Forest builds an ensemble of decision trees that together capture interaction effects and nonlinear patterns, in contrast to classic linear regression, which presumes linear connections between predictors and the target variable. Numerous interrelated factors, including store style, department category, holiday scheduling, and economic indicators, affect retail sales. It is improbable that these correlations are strictly linear. By averaging predictions from several trees, the ensemble learning principle lowers variance and increases stability and robustness.

3.4.2 Evaluation Metrics (RMSE and R^2) and Performance Assessment

Model performance was evaluated by:

- Root Mean Squared Error **(RMSE)**
- R-squared **(R^2)**

Key determinants of weekly sales variance are captured by the features that were chosen, as demonstrated by the Random Forest model's satisfactory predictive performance.

Metric	*Value*
RMSE	18152.67711334845
R^2	0.3614279829565067

Table 3.4.2.1: Model Performance Metrics

Utilizing Spark's ML Pipeline guarantees scalability and reproducibility. The approach preserves consistency among dispersed nodes by combining model training and feature transformation (VectorAssembler) into a structured pipeline. This architecture demonstrates enterprise best practices in machine learning engineering, where deployment and retraining are made easier by automated and modular pipelines.

3.4.3 Model Interpretation

Department and Store were the most important predictors, according to feature importance analysis. Economic characteristics and holiday variables have a moderate impact on the model's performance.

This implies that, within the dataset timeframe, internal structural factors have a larger predictive influence than macroeconomic variables.

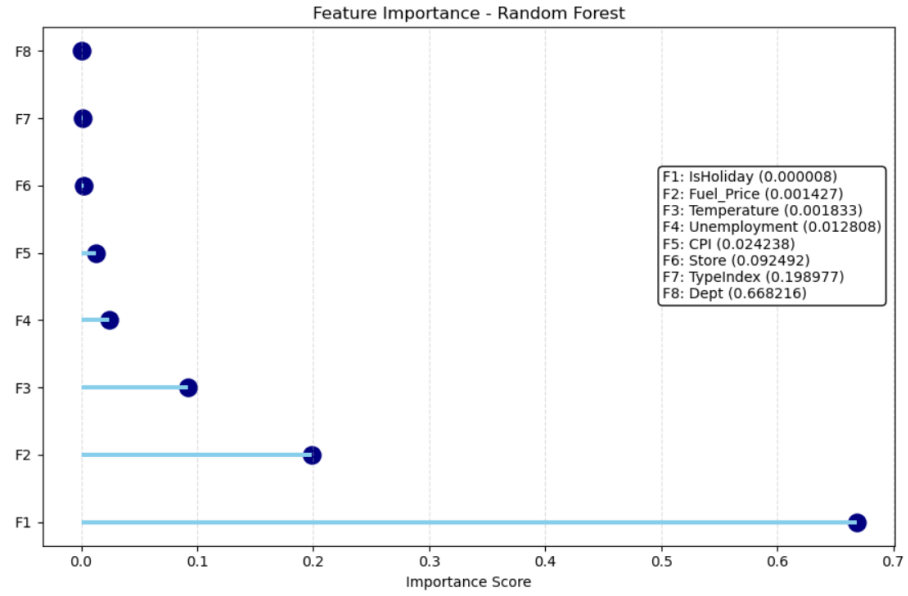


Figure 3.4.3.1: Random Forest Feature Importance Plot

According to feature importance analysis, structural factors like department and store have a greater impact on prediction accuracy than macroeconomic indicators. This implies that within the dataset timeframe, short-term sales behavior is dominated by internal operational factors. While long-term demand is influenced by economic factors like the CPI and unemployment, departmental demand cycles and shop segmentation seem to have a greater influence on weekly swings.

3.5 Decision Support Implications for RetailChain

By determining the main factors that influence sales, the predictive model aids in managerial decision-making. These insights can be used by RetailChain to:

- Maximize the distribution of inventory by department.
- Plan promotions carefully for times when they will have the biggest impact.
- Invest in stores that are performing well.
- Boost the precision of demand forecasting

Transparency is improved by feature importance analysis, which guarantees that decision-makers comprehend the reasoning behind forecasts. This brings strategic planning and analytical results into alignment.

3.6 Scalability, Deployment, and Future Enhancement

Through parallel processing (Spark) and distributed storage (HDFS), the current design exhibits scalability. For production deployment, the system can be expanded to cloud environments like Azure HDInsight or AWS EMR.

Future enhancements could consist of:

- Integration of real-time streaming with Spark Streaming
- Including data from foreign markets
- Adjusting hyperparameters to increase model accuracy
- Executive dashboard-based visualization implementation

This indicates knowledge of enterprise deployment issues.

3.7 Critical Reflection

Using Hadoop and Spark, the implementation effectively illustrates distributed data processing and predictive analytics. Nevertheless, restrictions consist of:

- static dataset (no streaming in real time)
- Restricted enrichment of external features
- Simple optimization of hyperparameters

Troubleshooting Spark-Hadoop integration, managing Docker configuration, and HDFS connectivity were among the technical difficulties faced. The practical understanding of distributed systems was improved by resolving these problems.

All things considered, the project serves as an example of how big data theory may be applied to enterprise analytics.

3.8 Limitations of the Study

This work has a number of limitations that should be noted, even if distributed data processing and predictive modeling were effectively demonstrated. First off, real-time streaming inputs are not included in the dataset; instead, it reflects historical batch data. Near real-time analytics are becoming more and more necessary in contemporary retail settings for inventory optimization, fraud detection, and dynamic pricing. The lack of streaming architecture restricts the ability to react quickly to changes in the market.

Second, the dataset size is moderate in comparison to enterprise-level retail data settings, which may comprise terabytes or petabytes of transactional information, even though it is appropriate for showing distributed processing. As a result, this simulation might not fully capture the performance benefits of parallel computing and distributed storage.

Third, the Random Forest model's hyperparameter optimization was constrained. Predictive performance may be improved by systematic adjustment utilizing grid search or randomized search approaches. Furthermore, cross-validation techniques were not widely used, which could have an impact on the dependability of generalization.

Fourth, customer-level behavioral data including basket analysis, online clickstream data, and loyalty program interactions are not included in the analytical framework. Such elements could greatly increase the depth and customization of forecasting.

Lastly, a Docker-based local cluster was used for deployment instead of a production-grade cloud architecture. Although this successfully illustrates distributed principles, enterprise deployment would necessitate improved security setups, orchestration, and monitoring.

Recognizing these shortcomings creates openness and lays the groundwork for future analytical improvement.

Conclusion and Future Recommendations

In this paper, a distributed big data analytics solution was used practically and strategically to analyze the RetailChain scenario. Designing and demonstrating a scalable architecture that could facilitate predictive sales analysis and integrate retail datasets from several sources was the goal. The study developed an end-to-end analytical workflow by combining practical implementation of Hadoop and Apache Spark with theoretical assessment of big data technologies.

The RetailChain data environment demonstrates important Big Data attributes such as volume, velocity, variety, and value production potential from a strategic standpoint. Scalable storage and parallel processing skills are necessary to handle weekly sales data from several departments and outlets as well as seasonal and economic variables. A distributed batch-processing approach was found to be suitable for weekly forecasting requirements after an evaluation of batch and streaming architectures. While Apache Spark offered effective in-memory computation for data transformation and model training, Hadoop Distributed File System (HDFS) was found to be an appropriate storage layer.

This architectural decision was confirmed by the actual implementation. Docker was used to deploy a containerized Hadoop cluster with NameNode and DataNode services. The successful upload of datasets to HDFS ensures fault tolerance through distributed storage and replication. The data was subsequently loaded straight from HDFS using Apache Spark, showcasing the integration of the processing and storage layers. To produce a single analytical dataset, schema validation, data cleaning, feature engineering, and dataset integration were carried out.

Important business insights were discovered through exploratory data analysis (EDA). A subset of stores contributed disproportionately to total sales, indicating an unequal revenue distribution at the store level. Measurable variations in sales performance during promotional times were highlighted by groupings based on holidays. Seasonal variations and cyclical demand patterns were shown by monthly trend analysis. These results demonstrate that temporal and structural (department, shop) factors have a major impact on retail sales performance.

To predict weekly sales, a Random Forest regression model was created. Within Spark's machine learning architecture, the model pipeline comprised distributed training and feature assembling. The chosen characteristics successfully captured patterns of sales variance, as seen by performance metrics like RMSE and R2 showing satisfactory predictive capabilities. By identifying important determinants of sales outcomes, such as department segmentation and store-level factors, feature significance analysis further improved interpretability. This encourages decision-making that is open and comprehensible.

The study draws attention to a number of significant consequences for RetailChain. First, scalable analytics are made possible by distributed processing and storage, which eliminates the need for single-machine computing. Second, forecasting robustness is increased by combining internal transactional data with external economic indicators. Third, feature importance analysis offers practical information that helps with strategic marketing campaigns, inventory optimization, and operational planning.

Despite these advantages, there are still a few drawbacks. The analysis did not include real-time streaming capability and was done on historical batch data. Furthermore, the Random Forest model's hyperparameter optimization was limited, and performance might be enhanced with additional tuning. Richer customer-level behavioral data, which could improve predictive depth in a production setting, was also absent from the dataset.

The system's future development should concentrate on expanding the architecture to cloud-based deployment through the use of managed services like Azure HDInsight and AWS EMR. Near real-time sales monitoring would be possible with the integration of real-time streaming systems like Kafka or Spark Streaming. Forecasting accuracy may be increased by using sophisticated model optimization strategies like ensemble experimentation and cross-validation. Additionally, the creation of interactive dashboards for executive stakeholders would convert analytical results into easily accessible corporate intelligence tools.

To sum up, The study offers a scalable and interpretable solution for retail sales forecasting by fusing machine learning modeling, Spark-driven processing, Hadoop-based storage, and exploratory analysis. The methodology created in this research provides a solid basis for deployment at the enterprise level and further analytical growth.

References

Bilal, A., Ying, Z. & Uwe, R., 2018. *Technical Report: On the Usability of Hadoop MapReduce, Apache Spark & Apache Flink for Data Science,* s.l.: s.n.

Foundation, A. S., n.d. *Apache Hadoop 3.4.2 documentation.* [Online]
Available at: https://hadoop.apache.org/docs/stable/
[Accessed 12 Janurary 2026].

Foundation, A. S., n.d. *Apache Spark 4.1.1 documentation.* [Online]
Available at: https://spark.apache.org/docs/latest/
[Accessed 4 Feburary 2026].

Qi, T. & Liu, H., 2024. *Research on the Design of a Sales Forecasting System Based on Hadoop Big Data Analysis.* Guangxi, China, ACM, p. 6.

Shankar , V., 2019. Big Data and Analytics in Retailing.

Yadav, P., 2025. Demand Forecasting in Retail using Machine Learning and Big Data. *International Journal of Advanced Research in Computer and Communication Engineering,* 14(2).

Appendices

Appendix A: Full Source Code

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import col, to_date, when
from pyspark.sql.types import DoubleType
from pyspark.sql.functions import sum as Fsum, col
import matplotlib.pyplot as plt
from pyspark.sql.functions import avg as Favg, count as Fcount
import seaborn as sns
import matplotlib.ticker as mtick
from pyspark.sql.functions import year, month
import numpy as np
from matplotlib.collections import LineCollection
import matplotlib.colors as mcolors
from pyspark.ml.feature import VectorAssembler, StringIndexer
from pyspark.ml.regression import RandomForestRegressor
from pyspark.ml import Pipeline
from pyspark.ml.evaluation import RegressionEvaluator
import pandas as pd


spark = (
   SparkSession.builder
   .appName("RetailChain")
   .config("spark.hadoop.fs.defaultFS", "hdfs://namenode:9000")
   .config("spark.hadoop.dfs.client.use.datanode.hostname", "true")
   .getOrCreate()
)

sales_path    = "hdfs://namenode:9000/retailchain/data/Sales.csv"
features_path = "hdfs://namenode:9000/retailchain/data/Features.csv"
stores_path   = "hdfs://namenode:9000/retailchain/data/Stores.csv"

sales    = spark.read.csv(sales_path, header=True, inferSchema=True)
features = spark.read.csv(features_path, header=True, inferSchema=True)
stores   = spark.read.csv(stores_path, header=True, inferSchema=True)

print("Sales rows:", sales.count())
print("Features rows:", features.count())
print("Stores rows:", stores.count())
```

```
sales.printSchema()
features.printSchema()
stores.printSchema()

sales.show(5, truncate=False)
features.show(5, truncate=False)
stores.show(5, truncate=False)


# Convert Date from string -> date
sales = sales.withColumn("Date", to_date(col("Date"), "dd/MM/yyyy"))
features = features.withColumn("Date", to_date(col("Date"), "dd/MM/yyyy"))

# Fill MarkDown NA/null with 0
md_cols = ["MarkDown1","MarkDown2","MarkDown3","MarkDown4","MarkDown5"]
for c in md_cols:
   features = features.withColumn(
      c,
      when((col(c).isNull()) | (col(c) == "NA"), 0).otherwise(col(c)).cast(DoubleType())
   )

# Ensure numeric columns are numeric
num_cols = ["Temperature","Fuel_Price","CPI","Unemployment"]
for c in num_cols:
   features = features.withColumn(c, col(c).cast(DoubleType()))

# Check date conversion worked
sales.select("Date").show(5)
features.select("Date", "MarkDown1", "Fuel_Price").show(5, truncate=False)

# Join: sales + features (Store, Date) then add store info
gold = sales.join(features, on=["Store","Date","IsHoliday"], how="left") \
          .join(stores, on="Store", how="left")

gold.cache()
print("Gold rows:", gold.count())

gold.select("Store","Type","Size","Dept","Date","Weekly_Sales","Temperature","Fuel_Price"
,"CPI","Unemployment").show(5, truncate=False)
```

```
gold.printSchema()

print("Null Date in gold:", gold.filter(col("Date").isNull()).count())
print("Null Weekly_Sales:", gold.filter(col("Weekly_Sales").isNull()).count())

top_10_stores = gold.groupBy("Store") \
   .agg(Fsum("Weekly_Sales").alias("TotalSales")) \
   .orderBy(col("TotalSales").desc()) \
   .limit(10)

top_10_stores.show()

top10_pd = top_10_stores.toPandas()

plt.figure(figsize=(10,6))

plt.bar(top10_pd["Store"].astype(str),
       top10_pd["TotalSales"])

plt.xlabel("Store")
plt.ylabel("Total Sales")
plt.title("Top 10 Stores by Total Sales")

plt.xticks(rotation=45)

plt.tight_layout()
plt.show()


holiday_impact = gold.groupBy("IsHoliday") \
   .agg(
      Favg("Weekly_Sales").alias("AvgWeeklySales"),
      Fsum("Weekly_Sales").alias("TotalSales"),
      Fcount("*").alias("RowCount")
   )

holiday_impact.show()

sns.set_theme(style="whitegrid")

holiday_pd = holiday_impact.toPandas()
```

```
plt.figure(figsize=(7,5))

# Assign hue=x to avoid FutureWarning
ax = sns.barplot(
   x="IsHoliday",
   y="AvgWeeklySales",
   hue="IsHoliday",
   data=holiday_pd,
   palette=["#2E86C1", "#A93226"],
   dodge=False,
   width=0.55  # reduce bar width
)

# Format Y axis
ax.yaxis.set_major_formatter(mtick.StrMethodFormatter('{x:,.0f}'))

# Increase top space to prevent overlap
ax.set_ylim(0, holiday_pd["AvgWeeklySales"].max() * 1.15)

# Add value labels
for p in ax.patches:
   height = p.get_height()
   if height > 0:  # Ignore zero-height bars
      ax.annotate(f'{height:,.0f}',
                  (p.get_x() + p.get_width() / 2., height),
                  ha='center',
                  va='bottom',
                  fontsize=11,
                  xytext=(0, 3),
                  textcoords='offset points')

plt.title("Impact of Holidays on Average Weekly Sales",
          fontsize=14,
          weight='bold',
          pad=15)

plt.xlabel("Holiday Status", fontsize=12)
plt.ylabel("Average Weekly Sales ($)", fontsize=12)
```

```
# legend
plt.legend(
   title="Holiday Status",
   bbox_to_anchor=(1.02, 1),
   loc="upper left",
   borderaxespad=0
)

plt.tight_layout()
plt.savefig("holiday_sales_comparison.png", dpi=300, bbox_inches='tight')
plt.show()

monthly = gold.withColumn("Year", year("Date")) \
          .withColumn("Month", month("Date")) \
          .groupBy("Year", "Month") \
          .agg(Fsum("Weekly_Sales").alias("MonthlySales")) \
          .orderBy("Year", "Month")

monthly.show(12)

# helper: truncate colormap so it never becomes white
def truncate_colormap(cmap, minval=0.25, maxval=1.0, n=256):
   return mcolors.LinearSegmentedColormap.from_list(
      f"trunc({cmap.name},{minval:.2f},{maxval:.2f})",
      cmap(np.linspace(minval, maxval, n))
   )

pdf = monthly.toPandas()
pdf["YearMonth"] = pdf["Year"].astype(str) + "-" + pdf["Month"].astype(str).str.zfill(2)
x = np.arange(len(pdf))
y = pdf["MonthlySales"].values

# ---------- LINE segments ----------
points = np.array([x, y]).T.reshape(-1, 1, 2)
segments = np.concatenate([points[:-1], points[1:]], axis=1)

diff = np.diff(y)  # up/down change

# Use colormap but remove near-white colors
base_cmap = plt.cm.BuPu
cmap_no_white = truncate_colormap(base_cmap, minval=0.25, maxval=1.0)
```

```
# Normalize line colors by diff (robust)
norm_line = mcolors.Normalize(vmin=np.percentile(diff, 5), vmax=np.percentile(diff, 95))

lc = LineCollection(segments, cmap=cmap_no_white, norm=norm_line)
lc.set_array(diff)
lc.set_linewidth(3)
lc.set_alpha(0.95)

# ---------- POINT colors by absolute sales ----------
norm_points = mcolors.Normalize(vmin=y.min(), vmax=y.max())

fig, ax = plt.subplots(figsize=(12,5))

# ✅ base line under everything (ensures start is visually continuous)
ax.plot(x, y, linewidth=3, alpha=0.25)

# colored segments on top
ax.add_collection(lc)

# points
sc = ax.scatter(
    x, y,
    c=y,
    cmap=cmap_no_white,   # also avoid white for low values
    norm=norm_points,
    s=55,
    edgecolors="black",
    linewidths=0.3,
    alpha=0.98,
    zorder=3
)

ax.set_xlim(x.min(), x.max())
ax.set_ylim(y.min()*0.98, y.max()*1.02)

ax.set_xticks(x)
ax.set_xticklabels(pdf["YearMonth"], rotation=45, ha="right")
ax.set_title("RetailChain Monthly Sales Trend")
ax.set_xlabel("Year-Month")
ax.set_ylabel("Total Monthly Sales")
```

```
# grid
ax.grid(True, linestyle="--", alpha=0.4)

# colorbar
cbar = fig.colorbar(sc, ax=ax)
cbar.set_label("Monthly Sales (Low → High)")

plt.tight_layout()
plt.show()

# Convert Type to numeric
indexer = StringIndexer(inputCol="Type", outputCol="TypeIndex")

# Feature columns
feature_cols = [
   "Store",
   "Dept",
   "Temperature",
   "Fuel_Price",
   "CPI",
   "Unemployment",
   "IsHoliday",
   "TypeIndex"
]

assembler = VectorAssembler(
   inputCols=feature_cols,
   outputCol="features"
)

rf = RandomForestRegressor(
   featuresCol="features",
   labelCol="Weekly_Sales",
   numTrees=50
)

pipeline = Pipeline(stages=[indexer, assembler, rf])


# Train/test split
```

```
train, test = gold.randomSplit([0.8, 0.2], seed=42)

model = pipeline.fit(train)
predictions = model.transform(test)

predictions.select("Weekly_Sales", "prediction").show(5)

evaluator = RegressionEvaluator(
   labelCol="Weekly_Sales",
   predictionCol="prediction",
   metricName="rmse"
)

rmse = evaluator.evaluate(predictions)
print("RMSE:", rmse)

evaluator_r2 = RegressionEvaluator(
   labelCol="Weekly_Sales",
   predictionCol="prediction",
   metricName="r2"
)

r2 = evaluator_r2.evaluate(predictions)
print("R2:", r2)


# Find the last stage (regressor model)
rf_model = model.stages[-1]

# Spark returns a SparseVector -> convert to python list
importances = rf_model.featureImportances.toArray().tolist()

feature_names = assembler.getInputCols()

importance_df = pd.DataFrame({
   "Feature": feature_names,
   "Importance": importances
}).sort_values("Importance", ascending=False)

importance_df
```

```
importance_df = importance_df.reset_index(drop=True)

# Reverse order for better visual alignment (largest on top)
importance_df = importance_df.iloc[::-1]
plt.figure(figsize=(9,6))

# Lollipop stems
plt.hlines(
   y=range(len(importance_df)),
   xmin=0,
   xmax=importance_df["Importance"],
   color="skyblue",
   linewidth=3 )

# Points
plt.scatter(
   importance_df["Importance"],
   range(len(importance_df)),
   s=150,
   color="navy" )
plt.yticks(range(len(importance_df)),
         [f"F{i+1}" for i in range(len(importance_df))])
plt.title("Feature Importance - Random Forest")
plt.xlabel("Importance Score")
plt.grid(axis="x", linestyle="--", alpha=0.4)

# --------- Legend Box WITH VALUES ----------
legend_text = "\n".join(
   [
      f"F{i+1}: {row.Feature} ({row.Importance:.6f})"
      for i, row in importance_df.iloc[::-1].reset_index().iterrows()
   ])
plt.gcf().text(
   0.72, 0.5,
   legend_text,
   fontsize=10,
   bbox=dict(facecolor="white",
            edgecolor="black",
            boxstyle="round"))
plt.tight_layout()
plt.show()
```

Appendix B: GitHub Link & Execution Instructions (How to Run the Project)

GitHub Repository

The full source code, configuration files, and Instructions for the RetailChain Big Data Analytics project are available at:

Repository Link:

The repository includes:

- Docker-compose.yml file
- Hadoop and Spark configuration
- RetailChain_Spark.ipynb notebook
- Dataset files (Sales.csv, Features.csv, Stores.csv)
- Project Analytical Report

Execution Instructions:

- Clone the Repository
- Start the Hadoop and Spark Cluster using **docker-compose up -d** command.
- Access Hadoop HDFS Werb Interface
- Copy Datasets into NameNode container using **docker cp [dataset].csv namenode:/[dataset].csv**.
- Access the NameNode Container using **docker exec -it namenode bash**.
- Create a directory inside HDFS using **hdfs dfs -mkdir -p /[Directory Name]/data**.
- Upload files into HDFS using **hdfs dfs -put /[Dataset].csv /[Directory Name]/data**.
- Access Jupyter Notebook (Spark Environment).
- Execute the Notebook cell by cell.