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:	4,555

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

Table of Contents

Introduction

The rapid growth of information in the modern retailing environments has transformed the manner in which organizations come up with operational and strategic decisions. The retail businesses generate a huge volume of external, operational, and transactional data on a daily basis. Efficient handling and analysis of huge data requires the distributed processing frameworks and scalable big data technologies. Big data retail datasets involving volumes and variety are often beyond the capacity to be processed using conventional data processing systems especially when analytical tasks such as trend prediction and forecasting are required.

Hadoop and Apache Spark technology are the focus of the present study, which is based on the RetailChain scenario where there was a huge retail sale data that is analyzed. The paper demonstrates that big data infrastructure could help in supporting the machine learning based forecasting, exploratory analytics and data integration. In the real world solution, Apache Spark is used in the distributed data processing and predictive modeling, and Hadoop Distributed File System (HDFS) is used to store the information.

1.1 Background of RetailChain Scenario

RetailChain is a retail enterprise that has numerous stores, which work in different sections and locations. The types of data that the organization produces includes weekly sales transactions, data sales store level, the promotional impact, holidays, and even external economic factors such as the consumer price index (CPI), and gasoline prices and the unemployment rate to mention a few.

The issue of RetailChain is with scaling and integration since the information is provided by several sources. In the case of large scale analytics, the local processing becomes ineffective because of the relatively big and constantly growing datasets.

To resolve this problem a distributed big data environment was developed incorporating:

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

1.2 Business Problem and Decision-Making Challenges

RetailChain would like to rely on data-driven insights to improve the abilities of its decision-making. The primary business challenges are to understand the impact of numerous factors on the performance of shops and make forecasts about sales every week successfully.
The problems that confront the organization include decision making issues:

i. **Complexity of Sales Forecasting**
   Sales performance is influenced by the stores, departments, seasons and external economic conditions. The identification of the key factors affecting sales requires advanced procedures of analysis.
ii. **Volume and Integration of Data**
   The complexity of data increases when the external indications are added to the transactional data. It manages this scaling of integration with distributed processing.
iii. **Holiday and Seasonal Impacts**
   Retail sales are highly influenced by holidays and other events during the seasons. These moving trends may not be well represented using traditional methods of analysis.
iv. **Limitations on Scalability**
   There is a performance problem regarding processing rates as performance grows in size. The scale of a scalable infrastructure is required because of consistent analytical performance.

1.3 Objectives of the Report

The key objective of the present study is to design, implement, and evaluate a big data analytics solution to the case of the RetailChain scenario.
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 addresses the topic of big data strategy and technology analysis. It examines the big data characteristics, processing, models of cloud computing, governance concerns, and the RetailChain data ecosystem. This section shows theoretical background in selection of appropriate technologies.

Part II will contain the applied approach and analytical resolution. It also provides instructions on how to create a Hadoop cluster, put information into HDFS, process the information with Spark, conduct an exploratory analysis, create machine learning models and evaluate them. The section is rounded off with such issues as scalability, tactics to apply the deployment, and critical analysis of the implemented solution.

Major conclusions are also summarized in the conclusion of the report, and they are followed by references and appendices containing the entire source code and instructions on how to execute it.

PART I—RetailChain Data Strategy and Technology Evaluation

2.1 RetailChain Data Environment and Business Context

Transactional and contextual data management is absolutely vital to both efficiency and strategic decision-making in the highly data-intensive retail sector where RetailChain does business. The core datasets that are used in this analysis are simplified models of a larger company data ecosystem by including sales, store information and external attributes. In the real world of retail operations **point-of-sales (POS)** systems, supply chain systems, inventory management systems, marketing campaigns, customer loyalty programs, and external economic indications are all generating and generating data on a continuous basis.

The sales dataset records weekly stores and department sales and reflects the tendencies of consumer demand. The number of features which includes, the temperature, fuel prices, the **consumer price index (CPI),** the unemployment rate, and holiday indicators are all present in the features dataset. The shops data contains store identification and classification-related structural variables. Coupled with external macroeconomic and environmental data, these datasets exhibit the integration of external and internal transactional data.

There are, nevertheless, numerous prospects of data-driven decision-making in the event of the integration of various datasets despite these challenges. By combining the operational and contextual data, RetailChain is able to estimate future tendencies of revenues, assess how the financial aspect of business and the, so-called, holidays impact and influence sales, as well as notice the swings in demand by the season. This forms the basis of strategic planning, stores performance optimization and predictive analytics.

The data environment of RetailChain that requires scalable processing technology and structured governance structures, thus, requires the reflection of strategic opportunities and intricacy characteristic of modern retail ecosystems.

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

The Big Data characteristics of the data environment of RetailChain show that the characteristics are significant and are commonly described by 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** It is referred to as the strategic insights gained through data analytics. The final objective is operational intelligence, one that can be used to predict demand, optimize costs, and achieve higher profits, not gathering data.

These characteristics favor the application of parallel processing models (Spark) and distributed storage (HDFS) at RetailChain which ensures scalability and analytical performance to complex data as the volume increases.

2.3 Big Data Processing Architectures

The two key architectural paradigms that govern the processing of the huge amounts of data are real-time streaming and batch processing.

Data collected is through a batch processing architecture at a predetermined time. It is a very good use in large scale aggregations, forecasting models and historical trend analysis. Distributed batch analytics can be applied to large data sets stored in HDFS, including Hadoop MapReduce technology, or Apache Spark technology. Weekly sales forecasting, which is applicable in this project, is well suited to batch systems since batch system gives preference to throughput over latency.

On the other hand, real-time streaming architecture is a continuous processing mechanism of data as they are generated. Frameworks such as Spark streaming and Apache reef make near-instant data input and analysis possible. Dynamic pricing, fraud detection, and real-time inventory update can be very helpful with the real-time systems.

A comparative analysis reveals that, though streaming architectures provide real-time responsiveness, they require more complex infrastructure and operational management whereas the cost of batch processing is simple, stable and economical to periodic analytical processes.

Taking into account that RetailChain aims at predicting the trends in sales on weekly basis and not make instant decisions at the level of transactions, a batch-oriented distributed architecture is more reasonable. The choice of Apache Spark to do parallel processing and **Hadoop Distributed File System (HDFS)** to store the information provides this need. This architecture provides scalability and the complexity of the system is managed.

2.4 Cloud Computing and Distributed Platform Evaluation

Cloud computing has transformed the way big data processing can be done since it is a scalable infrastructure that does not involve investments in on-premises hardware. The shift of the traditional data center to a distributed system based on the cloud offers the possibility to institute the pay-as-you-go model and the elastic resource allocation to use.

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.

Integrated big data ecosystem products available in the offerings of larger cloud providers, such as **AWS, Microsoft Azure, and Google Cloud Platform**, include managed Hadoop clusters and Spark services. These media enhance scale and reliability as well as reducing cost of operation.

Consequently, cloud computing is better in terms of scalability and reliability, yet it should be evaluated with caution in the context of compliance and organizational control.

2.5 Governance, Security, and Ethical Considerations

The proper data governance will make sure that the analytical resources of RetailChain are functioning within safe ethical limits. Retail data may contain interesting transactional data and, at a larger scale, may have recognizable customer data. It is, therefore, necessary to comply with the data protection regulations.

Anonymization of data, role based access control and encryption protocols should be part of the data privacy mechanism. HDFS allows permission-based access control and containerization also offers another rate of isolation of the environment.

Security is not limited to technical aspects anymore, but it also incorporates network configurations, authentication processes, and controlled access to the analytical environments. Minimizing the risk of unauthorized manipulation of data is ascertained by making sure that administrative privileges are limited.

In predictive analytics, ethical considerations are also of great importance. The sales forecasting models should not induce biased assumptions that may suggest discriminatory pricing or discriminative decision making which will impact the operations unfairly. The accountability is improved through transparency in model interpretation and analysis of feature importance.

Lastly, strategic decision-making is reliable because the high quality of data is upheld with governance structures. The auditability and reproducibility is supported through clear documentation of data sources, preprocessing procedures, and analytical procedures.

Together, governance, security, and ethical management are the pillars of responsible implementation of big data at 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 RetailChain objective is to make the sales projections error-free in a number of units and branches. The main purpose of the company is to predict the weekly sales based on the previous records of transactions and on external factors of the company in terms of their eagerness to go to the stores like holidays, economic indicators, and the store attributes. Accurate forecasting is what makes it possible to plan strategic promotional planning, enhance staffing, and manage inventory.

The objective variable in the analyzed regression problem that will be used to formulate the analysis issue is **Weekly_Sales**. Examples of independent ones are Store, Department, Holiday Indicator, CPI, unemployment rate and fuel price amongst other elements constructed in time. The technique employs distributed processing to achieve the goal of processing enormous transactional data without compromising the levels of analytical scalability.

The forecasting model is useful in the decision-making process of operations by identifying the primary causal factors affecting sales and quantifying their impact on performance in terms of revenues.

3.1.2 System Architecture and Technology Justification

The solution architecture integrates the Apache Spark of distributed data processing and analysis with the **Hadoop Distributed File System (HDFS)** of storage. The Hadoop cluster was deployed using Docker containers along with **NameNode**, and DataNode set up. This was facilitated by uploading the transactional data to HDFS as a systematic directory **(/retailchain/data).**

Apache Spark was picked due to its in-memory distributed processing feature which has been shown to highly improve the performance of tasks requiring machine learning and iterative based analytics. To facilitate data exploration, transformation and model creation, PySpark was used within a Jupyter Notebook.

This design is consistent with enterprise-scale analytics architectures and is an updated big data ecosystem comprising of scalable processing (Spark) and distributed storage (HDFS).

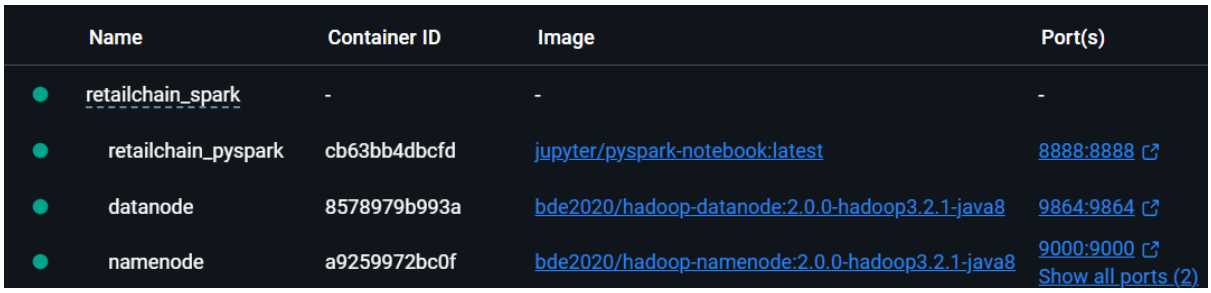

Figure 3.1.2.1: Docker-based Hadoop and Spark Cluster Deployment

Figure 3.1.2.1 illustrates that the Hadoop cluster was installed in containers using Docker in the form of NameNode, DataNode, and PySpark notebook services.

Figure 3.1.2.2.2 demonstrates how datasets were successfully ingested into HDFS with replication factor of three and it guarantees fault-tolerance and distributed storage reliability.

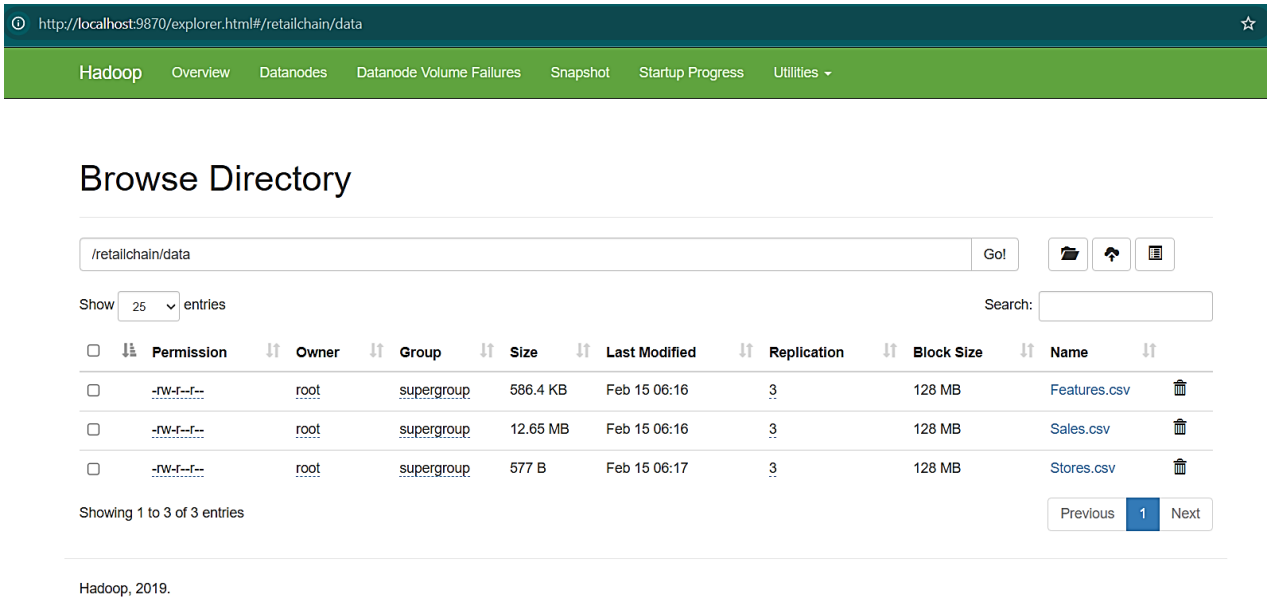

Figure 3.1.2.2: HDFS Directory Structure and Data Replication

3.2 Practical Implementation Using Hadoop and Apache Spark

3.2.1 Hadoop Cluster Setup and HDFS Data Loading

A Hadoop cluster was configured with numerous containers, provided by Docker Compose. 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.

3.2.3 Feature Engineering and Dataset Construction

The feature engineering was intended to help improve prediction. 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

The feature engineering enhanced prediction accuracy because it stored the categorical storage properties and therefore, retrieved the temporal elements of date data. To generate one feature-vector as required by **Spark ML** techniques, multiple predictors were joined together with the help of VectorAssembler. Such an approach ensures compatibility with decentralized training, and normalization of model input.

3.3 Exploratory Data Analysis and Business Insights

3.3.1 Exploratory Findings

Significant differences in weekly sales were observed in the stores and departments as explored in the analysis. By the store level analysis, some types of stores always perform better than others. We experienced seasonal trends particularly during the holidays.

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

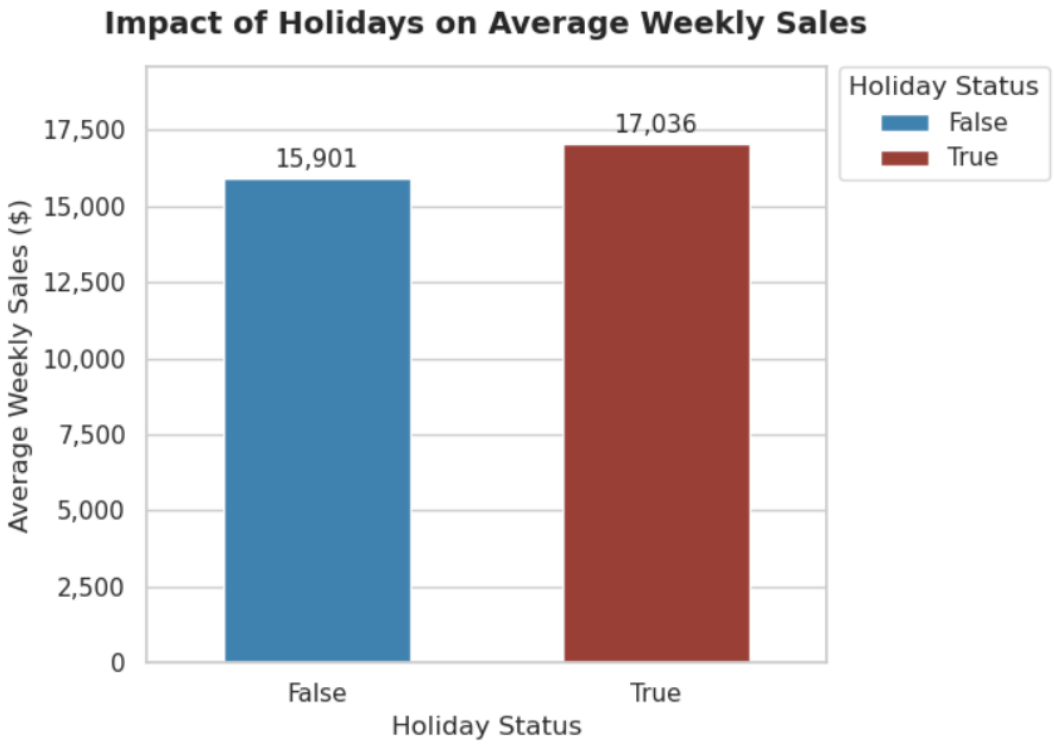

Figure 3.3.1.2: Impact of Holidays on Avg Weekly Sales ($)

The bar chart is used to compare the average weekly sales made in the non-holiday and the holiday periods. As the findings suggest, average sales per week during the period of the holiday weeks are higher **(average =17,036 USD)** than those in the non-holiday weeks **(average =15,901 USD).** This represents an approximately **7 percent** increase in sales over the season of the holidays. The identified gap means that holidays positively affect consumer spending behavior, which is likely explained by the increased activity of the promotional, seasonal, and foot traffic. To maximize revenue performance in the season of holidays, such findings make it possible to underline the importance of the close inventory planning and targeted marketing actions.

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

Cyclical sales patterns were observed in the monthly total with apparent inclinations in marketing seasons. Average sales during the year were statistically high during the holiday weeks as compared to non-holiday periods.

3.3.2 Business Interpretation

The data indicates that the revenue is strongly dependent on the store segmentation and the timing of the promotion. Large stores have a higher growth potential but more variation in sales. Moderate correlations existed between the performance in sales and the economic levels such as the CPI and unemployment.

These insights provide management with the necessary knowledge in the planning of the operations and marketing of their operations.

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

The issue was modelled using a regression task which predicted **Weekly_Sales**. Test and training data sets were separated. The following components were used to create a Spark ML pipeline:

- Feature Vector Assembler
- Feature Scaling
- Random Forest Regressor

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

The selection of the Random Forest regression algorithm was due to its ability to capture complex and non-linear relationships among retail data. Random Forest constructs a combination of decision trees that jointly model interaction, nonlinear effects instead of the traditional linear regression which asserts that there is a linear relationship between the predictors and the dependent variable. The retail sales are influenced by numerous factors which are interrelated such as the style of shop, the type of department, days off during holidays and economic indicators. There is a low possibility that the correlations are necessarily linear. The ensemble principle of learning reduces the variance and enhances stability and robustness by averaging predictions of multiple trees.

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)**

The features chosen represent key determinants of the weekly sales variance, as the features adequately predict the sales variance, as shown by the satisfactory predictive performance of the Random Forest model.

Metric	*Value*
RMSE	18152.67711334845
R^2	0.3614279829565067

Table 3.4.2.1: Model Performance Metrics

By deploying the ML Pipeline of Spark, it is scalable and reproducible. The method maintains uniformity across the scattered nodes by fusing model training and feature transformation (VectorAssembler) in an organized channel. This architecture exhibits the best practice in machine learning engineering at the enterprise level, deploying and re-training are facilitated through automated 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.

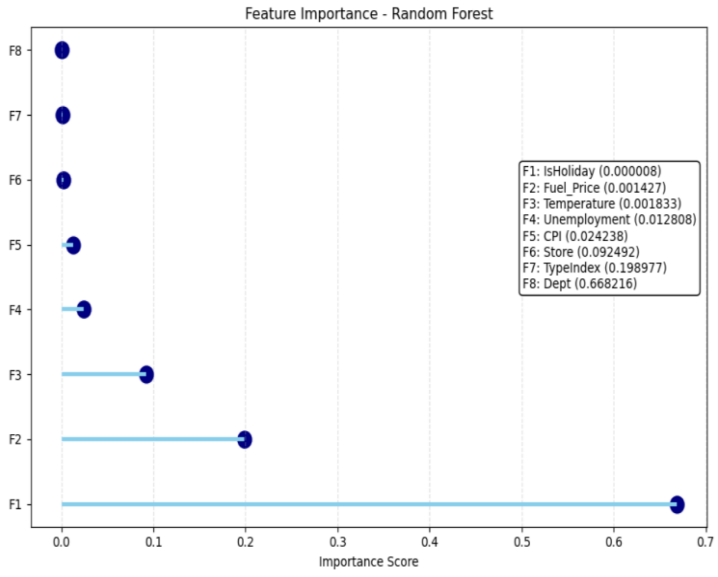

Figure 3.4.3.1: Random Forest Feature Importance Plot

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

The existing design has a scalable nature through the use of parallel processing **(Spark)** and distributed storage **(HDFS).** To scale to the production level, the system may be scaled into cloud-based systems such as **Azure HDInsight** or **AWS EMR**.

Enhancements that may be done in future are:

- 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

The implementation based on **Hadoop and Spark** is an effective demonstration of distributed data processing and predictive analytics. The limitations are however:

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

Some of the technical challenges encountered were troubleshooting **Spark-Hadoop** integration, **Docker configuration, and HDFS connectivity**. These issues were solved to enhance the practical knowledge of the distributed systems.

Altogether, the project can be taken as an example of the ways big data theory can be implemented to enterprise analytics.

3.8 Limitations of the Study

Several limitations of this work will be mentioned, despite the lack of exposed limitations in the distribution of data processing and predictive modeling. To begin with, it does not contain real-time streaming information as inputs, but it is a reflection of historic batch information. Real-time analytics is increasingly being demanded and required in the modern retail contexts to improve inventory optimization, detect fraud, and dynamic pricing. The absence of the streaming architecture limits the possibility to respond promptly to the market alterations.

Second, the size of the dataset is moderate by the standards of enterprise-level retail data environments, where it can include terabytes of information on transactions, although it suffices to display distributed processing. Consequently, it is possible that the performance gains of parallel computing and distributed storage are not completely represented at this simulation.

Third, it was limited by the hyperparameter optimization of the Random Forest model. Systematic adjustment with grid search or randomized search can be used to improve predictive performance. Moreover, the cross-validation methods have not been extensively applied and this may affect the reliability of generalization.

Fourth, behavioral elements at customer-level, such as basket analysis, online clickstream data and loyalty programs interaction are not part of the analytical cycle. These factors can dramatically enhance the levels of forecasting and personalization.

Finally, the deployment was on a local cluster based on Docker, rather than cloud architecture of production grade. Whereas this has been successful in demonstrating distributed principles, there would be the need to have better security setups, orchestration, and monitoring to implement this in the enterprise.

Being aware of such inadequacies forms openings and preconditions future analytical progress.

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.

The application of this structural choice was validated through practical practice. Docker has been used to deploy a containerized Hadoop cluster consisting of **NameNode** service and a **DataNode** service. Fault tolerance is guaranteed by distributed storage and replication with the successful upload of datasets to HDFS. The information was then loaded directly out of **HDFS** using **Apache Spark** demonstrating how the layers of processing and storage were combined. Schema validation, data cleaning, feature engineering, and dataset integration were performed in order to generate a single analytical dataset.

Through the **exploratory data analysis (EDA)**, important business insights were uncovered. The revenue distribution at the store level was uneven as one group of stores generated more than the overall total sales. Groupings within the context of holidays were used to highlight measurable variations in sales performance of the sales performance during the promotional periods. Monthly trend analysis indicated seasonal fluctuations and cyclical demand. These findings suggest that a significant influence on the performance of the retail sales is exerted by the temporal and structural **(department, shop)** factors.

A random forest regression model was developed to determine the sales that would occur in the week. In the machine learning system at Spark, distributed training and feature assembling were the components of the model pipeline. The selected characteristics were effective in capturing sales variance patterns as observed in the performance measures such as the **RMSE** and the **R2** which indicated acceptable levels of predictive performance. The feature significance analysis enhanced interpretability by identifying meaningful predictors of sales performance including information on the above criterion department segmentation and store-level variable. This promotes open and understandable decision making.

The analysis highlights several key implications to RetailChain. To start with, the distributed processing and storage save scalable analytics; these are computing capabilities that are not based on a single machine. Second, internal transactional data, in conjunction with external economic indicators, enhances the forecasting of robustness. Third, the feature importance analysis provides useful data that can be used in strategic marketing campaigns, inventory planning, and planning operations.

Although these are the advantages, it still has its share of disadvantages. The test did not take into consideration real-time streaming capability and was performed on previous batch data. Moreover, the hyperparameter optimization in the Random Forest model was not as effective, and its implementation could be improved by having further tuning. The dataset also lacked richer customer level behavioral information that could enhance predictive depth during a manufacturing environment.

The development of the system ahead should focus on extending the system to cloud-based implementation by utilizing the services of managed services such as **Azure HDInsight** and **AWS EMR**. Real-time monitoring of sales with real-time streaming systems, such as Kafka or Spark streaming, would be a possibility. Learning sophisticated model optimization techniques such as ensemble experimentation and cross-validation can be used to improve the forecasting accuracy. Also, it would be necessary to build interactive dashboards to executive stakeholders to transform analytical findings into convenient corporate intelligence instruments.

In conclusion, The paper provides a predictable and interpretable model to predict retail sales by combining machine learning modeling, Spark-based processing, use of Hadoop storage system, and use of exploratory analysis. The research methodology developed in this study offers firm foundation in implementation of the research at the enterprise level and subsequent analysis expansion.

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

```
import pyspark.sql. SparkSession.
import pyspark.sql.functions as col, to_date, when.
import pyspark.sql.types.DoubleType
|human|>import pyspark.sql.types.DoubleType
imported pyspark.sql.functions sum as Fsum, col
import matplotlib.pyplot as plot.
importing pyspark.SQL.functions) avg (Favg) and count (Fcount).
import seaborn as sns
take matplotlib.ticker as mtick.
imported from pyspark.sql.functions year, month.
import numpy as np
import matplotlib.collections.LineCollection
import mcolors matplotlib.colors
import pyspark.ml.feature. VectorAssembler, StringIndexer
import pyspark.ml.regression.RandomForestRegressor
import Pipeline, pyspark.ml.
import pyspark.ml.evaluation 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

This project source code, configuration files and Instructions of the RetailChain Big Data Analytics project are found at:

Repository Link: [RetailChain Big Data Analytics and Decision]

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.