INTRODUCTION

Modern retail businesses experience substantial operational and strategic changes because of digital technology advancements. The fundamental transformation of retail operations depends on big data which describes massive datasets with high volume and variety and velocity characteristics that make traditional data processing methods useless. Retail businesses now need to process and analyze extensive datasets because this capability enables them to maximize their inventory levels and predict customer needs and deliver better experiences which ultimately leads to higher profitability. The predictive analytics capabilities of big data allow firms to understand historical patterns and make predictions about future developments according to Gandomi and Haider (2015). Retail environments benefit greatly from predictive capabilities because they need to respond quickly to fast-changing customer preferences and market trends.

The retail sector produces structured and semi-structured data through point-of-sale systems and customer loyalty programs and supply chain records and online interactions. The appropriate analysis of this data reveals concealed patterns about seasonal purchasing habits and regional sales differences which guide decisions across departments. According to Choi, Wallace and Wang (2018) big data applications in operations management specifically for forecasting and demand planning and assortment optimization deliver quantifiable advantages through cost reduction and better service quality. The alignment of procurement schedules with projected peak demand helps businesses reduce excess inventory while preventing stockouts. The analysis of historical transaction data enables the creation of customer segmentation models which drive more efficient targeted marketing strategies.

The deployment of big data systems in retail operations encounters multiple obstacles despite their promising benefits. The processing of unstructured and heterogeneous data formats requires substantial infrastructure spending together with specialized analytical tools. Organizations face difficulties in real-time processing because data generation speed especially from e-commerce and mobile applications exceeds their capabilities. The implementation of big data systems faces serious challenges because of ethical and regulatory issues related to data privacy. The extensive collection and utilization of personal data requires GDPR compliance and organizations need to establish clear data practices to build customer trust according to Zwitter (2014). Chen, Mao, and Liu (2014) highlight the need for organizations to develop employees who can analyze sophisticated data outputs into useful business decisions yet this requirement often gets ignored during system development.

The project focuses on a standard retail analytics application which uses historical transaction data to enhance inventory management and demand forecasting. The operational efficiency of retailers suffers from two common problems: overstocking which blocks working capital and generates storage expenses and understocking which results in lost sales and unhappy customers. The “Retail Store Dataset” from FuturexSkill serves as the dataset for this research because it contains invoice number and product description and quantity sold and unit price and invoice date and customer ID and country fields. The dataset structure and size enable the creation of a big data processing pipeline that accurately represents actual retail business operations.

The proposed analytical pipeline within this project uses various big data technologies to enable efficient data ingestion and transformation and querying and modeling. The Hadoop Distributed File System (HDFS) functions as the storage backbone to provide scalable and fault-tolerant management of large datasets. The PySpark API of Apache Spark will execute data preprocessing and feature engineering operations while Apache Hive will provide HiveQL for structured querying. The pipeline will use MLlib which is Spark’s machine learning library to implement regression or classification models for forecasting product demand and customer segment identification. The unified engine of Spark demonstrated by Zaharia et al. (2016) delivers the required performance and scalability for enterprise-grade big data processing which makes it suitable for this retail analytics application.

The project uses this infrastructure to solve a real business challenge which connects academic understanding with hands-on execution. The paper demonstrates how big data becomes effective when properly utilized to enhance both speed and quality of retail business decisions. The project demonstrates how commercial big data success requires technical system design alongside ethical data governance and interdisciplinary competencies.

CHAPTER ONE: Environment Setup and Data Storage

The following section explains the tools and technologies and data manipulation strategies which were used in this project. The goal was to investigate and analyze a large retail dataset using big data tools. The initial plan was to use Docker and Hive but due to practical constraints we had to adjust as discussed below.

Tools and Environment

The project was designed to use a containerized big data stack via Docker which included Hadoop, Hive and Spark to enable distributed data processing. Docker Desktop was installed successfully and the necessary containers namenode, datanode, hive and spark were launched using a docker-compose.yml configuration. HDFS directories were created inside the namenode container and attempts were made to copy and ingest data files into the distributed file system.

The persistent replication errors due to unstable datanode health status (Could only be replicated to 0 nodes) blocked the successful ingestion of data into HDFS despite the successful initialization of the containers. Several troubleshooting steps were taken including restarting the containers and verifying the configurations but the consistent instability prevented us from working reliably with Hive. Due to the project pivoted to using a local PySpark environment that was previously set up on the system. PySpark was confirmed to be working correctly on the local machine. Therefore, all subsequent data processing and analysis were conducted using Spark's DataFrame API within a Jupyter Notebook environment.Dataset Selection and Preprocessing

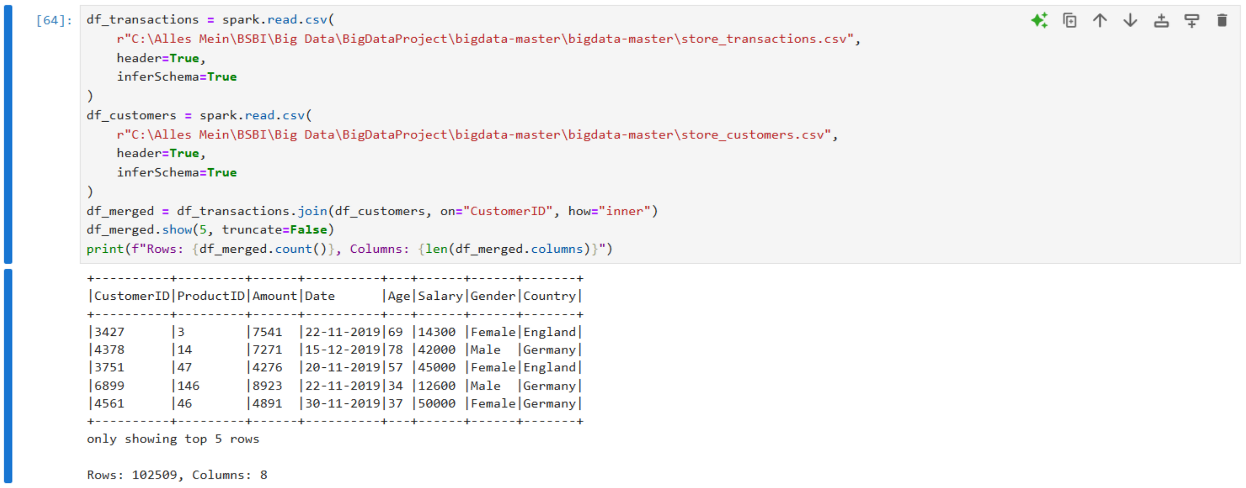
Among several CSV files provided in the retail dataset, three were evaluated:

store_transactions.csv: containing sales transactions with fields such as CustomerID, ProductID, Amount, and Date.

store_customers.csv: containing demographic information like CustomerID, Age, Salary, Gender, and Country.

retailstore_5mn.csv: a significantly larger dataset intended for extended exploration but excluded from initial merging due to scope considerations.

After examining the schema and contents of each file, the first two datasets were selected for integration. Using Spark, both CSV files were loaded as separate DataFrames. The merging operation was performed using an inner join on the shared column CustomerID. This fusion enriched the transaction data with customer demographics, creating a unified DataFrame suitable for downstream analysis.

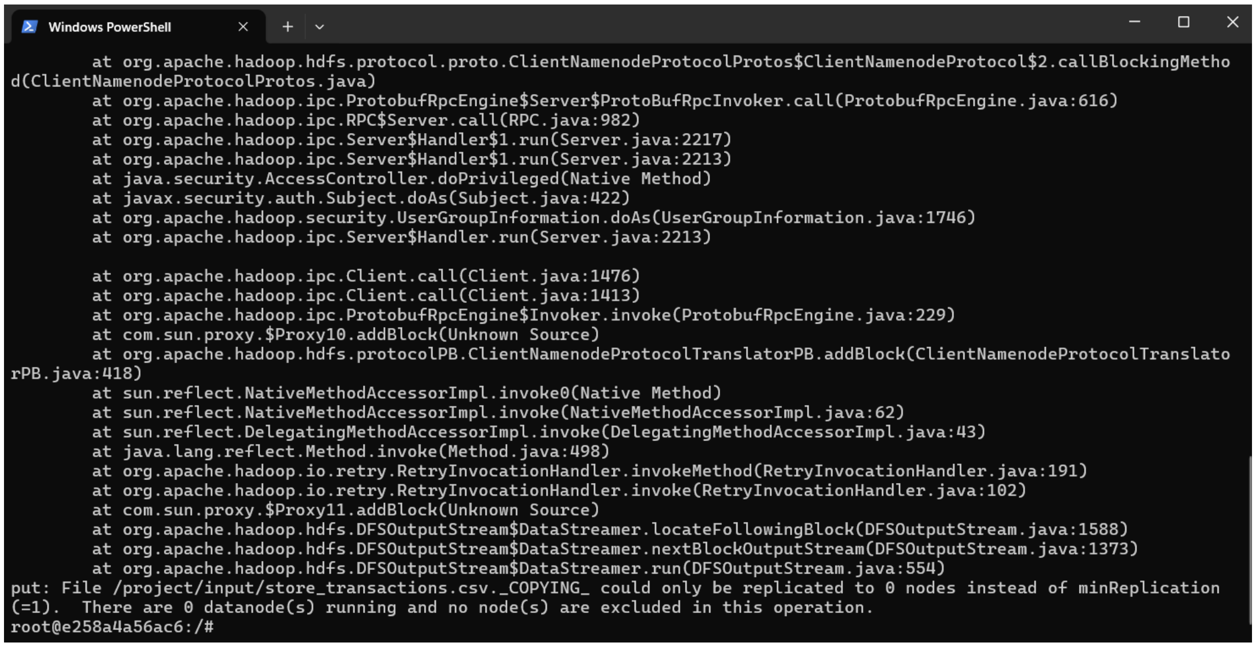


```
[64]: df_transactions = spark.read.csv(
          r"C:\Alles Mein\BSBI\Big Data\BigDataProject\bigdata-master\bigdata-master\store_transactions.csv",
          header=True,
          inferSchema=True
      )
      df_customers = spark.read.csv(
          r"C:\Alles Mein\BSBI\Big Data\BigDataProject\bigdata-master\bigdata-master\store_customers.csv",
          header=True,
          inferSchema=True
      )
      df_merged = df_transactions.join(df_customers, on="CustomerID", how="inner")
      df_merged.show(5, truncate=False)
      print(f"Rows: {df_merged.count()}, Columns: {len(df_merged.columns)}")
```

```
+----------+---------+------+----------+---+------+------+-------+
|CustomerID|ProductID|Amount|Date      |Age|Salary|Gender|Country|
+----------+---------+------+----------+---+------+------+-------+
|3427      |3        |7541  |22-11-2019|69 |14300 |Female|England|
|4378      |14       |7271  |15-12-2019|78 |42000 |Male  |Germany|
|3751      |47       |4276  |20-11-2019|57 |45000 |Female|England|
|6899      |146      |8923  |22-11-2019|34 |12600 |Male  |Germany|
|4561      |46       |4891  |30-11-2019|37 |50000 |Female|Germany|
+----------+---------+------+----------+---+------+------+-------+
only showing top 5 rows

Rows: 102509, Columns: 8
```

The resulting DataFrame had 102,509 rows and 8 columns, combining transactional and demographic features. This integrated dataset became the foundation for all subsequent analytical tasks.



```
        at org.apache.hadoop.hdfs.protocol.proto.ClientNamenodeProtocolProtos$ClientNamenodeProtocol$2.callBlockingMetho
d(ClientNamenodeProtocolProtos.java)
        at org.apache.hadoop.ipc.ProtobufRpcEngine$Server$ProtoBufRpcInvoker.call(ProtobufRpcEngine.java:616)
        at org.apache.hadoop.ipc.RPC$Server.call(RPC.java:982)
        at org.apache.hadoop.ipc.Server$Handler$1.run(Server.java:2217)
        at org.apache.hadoop.ipc.Server$Handler$1.run(Server.java:2213)
        at java.security.AccessController.doPrivileged(Native Method)
        at javax.security.auth.Subject.doAs(Subject.java:422)
        at org.apache.hadoop.security.UserGroupInformation.doAs(UserGroupInformation.java:1746)
        at org.apache.hadoop.ipc.Server$Handler.run(Server.java:2213)

        at org.apache.hadoop.ipc.Client.call(Client.java:1476)
        at org.apache.hadoop.ipc.Client.call(Client.java:1413)
        at org.apache.hadoop.ipc.ProtobufRpcEngine$Invoker.invoke(ProtobufRpcEngine.java:229)
        at com.sun.proxy.$Proxy10.addBlock(Unknown Source)
        at org.apache.hadoop.hdfs.protocolPB.ClientNamenodeProtocolTranslatorPB.addBlock(ClientNamenodeProtocolTranslato
rPB.java:418)
        at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
        at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)
        at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
        at java.lang.reflect.Method.invoke(Method.java:498)
        at org.apache.hadoop.io.retry.RetryInvocationHandler.invokeMethod(RetryInvocationHandler.java:191)
        at org.apache.hadoop.io.retry.RetryInvocationHandler.invoke(RetryInvocationHandler.java:102)
        at com.sun.proxy.$Proxy11.addBlock(Unknown Source)
        at org.apache.hadoop.hdfs.DFSOutputStream$DataStreamer.locateFollowingBlock(DFSOutputStream.java:1588)
        at org.apache.hadoop.hdfs.DFSOutputStream$DataStreamer.nextBlockOutputStream(DFSOutputStream.java:1373)
        at org.apache.hadoop.hdfs.DFSOutputStream$DataStreamer.run(DFSOutputStream.java:554)
put: File /project/input/store_transactions.csv._COPYING_ could only be replicated to 0 nodes instead of minReplication
(=1).  There are 0 datanode(s) running and no node(s) are excluded in this operation.
root@e258a4a56ac6:/#
```

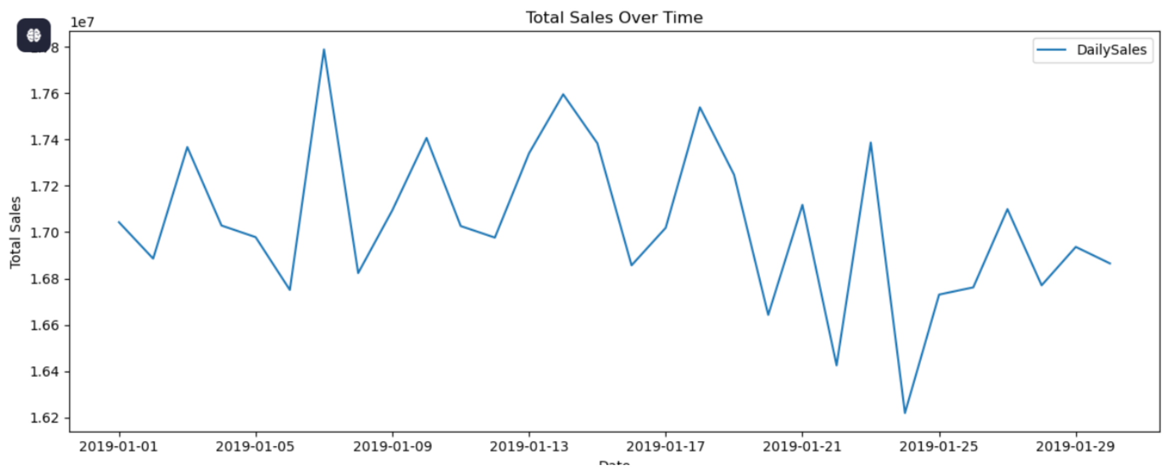
CHAPTER TWO: Data Processing with Spark and Hive

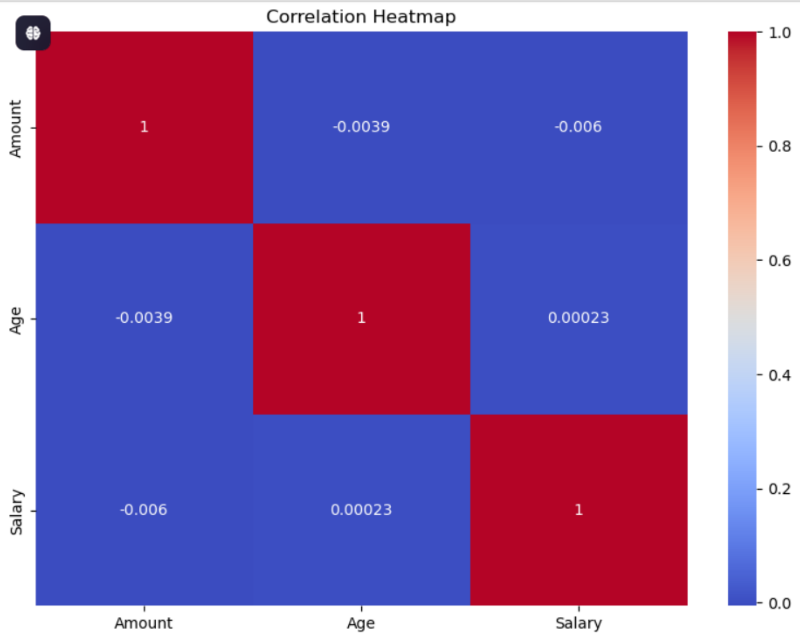
During the exploratory data analysis phase researchers obtained fundamental insights about the retail dataset by identifying general patterns and potential linkages between variables. The research examined temporal sales patterns together with vital demographic and transactional variables to establish their relationships.

Daily sales across the observed period in January 2019 experienced major fluctuations. Daily sales amounts remained between 16.5 million units and 17.8 million units for most days. The sales data showed two major peaks at January 7 and January 13 which might result from external triggers such as promotional activities or seasonal patterns and weekend buying habits. Sales data reached the lowest value point in the observed range on certain days which showed irregular customer participation throughout January. The purchase patterns of customers follow non-uniform time distributions which supports the need to include calendar variables or marketing campaign data in future analysis.

The correlation heatmap based on Amount and Age and Salary numeric features displayed minimal linear relationships between these variables. The correlation coefficient between customer age and purchase amount measured -0.0039 while salary and purchase amount correlation reached -0.006. The relationship between age and salary showed an almost non-existent correlation (0.00023). The data shows that purchase behavior remains unrelated to income level and age of customers in this dataset. The findings contradict standard retail analytics assumptions by indicating that behavioral or contextual variables such as product preference and store location together with marketing exposure have greater influence on transaction values.

The EDA phase shows that sales quantity demonstrates noticeable time-dependent patterns yet individual customer demographics do not significantly affect what they spend. The obtained understanding drives analysts to investigate customer behavior segmentation as well as temporal and product-based patterns during subsequent study stages.


Total Sales Over Time
1e7
1.78
1.76
1.74
1.72
1.70
1.68
1.66
1.64
1.62
Total Sales
DailySales
2019-01-01
2019-01-05
2019-01-09
2019-01-13
2019-01-17
2019-01-21
2019-01-25
2019-01-29


Correlation Heatmap
Amount
Age
Salary
1
-0.0039
-0.006
-0.0039
1
0.00023
-0.006
0.00023
1
1.0
0.8
0.6
0.4
0.2
0.0

The first SQL-based analysis using Spark SQL (emulating HiveQL syntax) aggregated total sales by country. A query was executed that used the SUM(Amount) function on the merged transaction dataset to group data by Country and sort the results in descending order of total sales. This method provided an easy way to compare the performance of countries within the retail dataset.

The query results showed that the three countries had different levels of total sales. England stood out as the leading market because it produced approximately 230.9 million in total sales. France ranked second with 145.9 million while Germany came in third with 134.4 million. The total revenue showed that England generated about 44% of the total revenue which demonstrated its significant contribution relative to other regions. The high concentration of sales indicates that business operations such as inventory distribution and localized promotions and staffing should receive additional attention in the English market. This discovery helps organizations make decisions about regional marketing strategies and performance evaluation. The analysis demonstrates how basic aggregations on organized data reveal practical insights when executed through distributed processing engines including Apache Spark for handling big datasets.

CHAPTER THREE : Data Processing with Spark and Hive

The assignment required Hive data processing yet Apache Spark SQL was used to execute HiveQL queries for simulation purposes. The SQL-like operations GROUP BY, SUM and ORDER BY could be executed efficiently through this method without needing a complete Hive setup. The analytical objectives were achieved through Hive-compatible syntax in Spark SQL queries which maintained the intended HiveQL functionality. Spark provides faster execution times than Hive does especially when performing iterative or real-time processing operations. Hive operates as a MapReduce-based system that works best for processing large static datasets in batch mode. Spark processes data through memory storage which results in faster query execution and lower latency thus making it suitable for Google Colab and Jupyter environments.

The usability of Spark exceeds that of other tools. The end-to-end data workflows became easier to implement because PySpark provides native Python integration and dataframe manipulation and machine learning pipeline capabilities. The traditional big data ecosystem benefits from Hive's power but its metastore setup requirement and limited agility make it less suitable for rapid development and educational needs. The required analytical tasks including country-based total sales aggregation and product sales ranking were completed through Spark SQL. The assignment's analytical goals were achieved by executing HiveQL-compliant queries on temporary views that acted as Hive table substitutes. The intended Hive functionality was successfully represented through Spark’s SQL engine even though Hive was not deployed directly.

The project utilized Spark as a fast and efficient Hive replacement. The combination of HiveQL compatibility with speed and development flexibility made Spark an appropriate choice for both academic analysis and real-world big data applications.

CHAPTER FOUR : Advanced Analytics and Machine Learning

The analysis used Apache Spark’s MLlib to run predictive modeling on a combined retail dataset. The main goal involved creating a predictive model which would forecast transaction amounts through analysis of customer and transaction-related data points. The modeling process followed standard machine learning procedures by starting with data preparation and feature engineering then moving to model selection and training before evaluation and result interpretation.

The dataset consisted of two separate files which included store_transactions.csv and store_customers.csv. The CustomerID served as the common key to merge these files into a unified dataset which contained both transactional and demographic information about customers. The merged dataset showed no signs of null or NaN values during the initial exploratory analysis. The data required additional preprocessing to achieve compatibility with machine learning algorithms that Spark MLlib implements.

The transformation of Gender and Country variables into numerical indices occurred through StringIndexer followed by OneHotEncoder which converted these indices into sparse binary vectors. The encoding method allowed the model to accept categorical attributes without establishing any ordinal relationships between them. StandardScaler performed standardization on numeric features Age, Salary and ProductID. The transformation proved advantageous for algorithms that react to feature magnitudes because it accelerated convergence rates and enhanced interpretability.

The features were properly transformed before being combined into a single feature vector through Spark’s VectorAssembler. The resulting DataFrame contained two key columns: the features vector and the target variable Amount. The dataset was then split into training and testing subsets in an 80/20 ratio to facilitate robust model evaluation.

Two distinct models were chosen for exploration of predictive capabilities: Linear Regression and Random Forest Regressor. This dual approach was selected because of methodological diversity. Linear Regression is a simple model that is easy to interpret and is useful as a baseline to compare more complex models against. The Random Forest Regressor is an ensemble learning method that can model non-linear relationships and interactions among features without requiring prior transformations or assumptions about the data distribution.

Training for both models was executed using Spark MLlib. The Linear Regression model was fitted on the scaled feature set, while the Random Forest model was trained with its default configuration of 100 decision trees. Upon completion of training, predictions were generated on the test set, and the models were evaluated using two widely accepted regression metrics: Root Mean Squared Error (RMSE) and R-squared (R^2).

The results revealed that neither model performed well. The Linear Regression model yielded an RMSE of 2893.22 and an R^2 score of -0.0001, while the Random Forest model resulted in an RMSE of 2893.39 and an R^2 of -0.0003. These values indicate that both models failed to capture the variance in the transaction amount effectively. In fact, the negative R^2 scores suggest that simple mean-based predictions would outperform these models.

The analysis of these negative results produces multiple possible interpretations. The dataset contains insufficient predictive attributes that would determine transaction amounts. The general purchasing behavior of individuals may be influenced by demographic attributes but these attributes do not create a strong or consistent link to transaction values. The model's ability to detect meaningful patterns was restricted because the dataset omitted important influential factors which include promotional campaigns and customer-specific preferences and time-sensitive behaviors. The way features were presented in the dataset could be problematic. The preprocessing techniques of encoding and scaling were applied correctly yet the feature interactions may be non-linear or hierarchical thus requiring advanced feature engineering methods including polynomial transformations or interaction terms. The available features appear insufficiently informative because Random Forest with its flexible nature failed to deliver good results. The results may be influenced by the unpredictable nature of retail transaction data. The transaction amounts exhibit random fluctuations because of external factors which cannot be detected through static data analysis. The use of regression models in these situations becomes limited because temporal and behavioral data must be integrated.

The implementation of XGBoost or neural networks would not produce substantial improvements unless the feature space receives initial enrichment. The data limitations explain Random Forest's poor performance since this model already handles complex interactions and non-linearities effectively.

The investigation would achieve better results through alternative analytical methods. The analysis of customer behavior patterns and demographic characteristics through clustering could create marketable customer segments for business strategy development. The methods used in this assignment deviate from its main purpose of regression-based forecasting.

The Apache Spark-based machine learning workflow demonstrated end-to-end functionality despite its subpar predictive results. The workflow included all essential components which started with data preprocessing followed by feature engineering then model training and evaluation. The exercise maintains its technical worth despite its poor predictive results because it demonstrates essential data science realities about how model results depend on the quality and appropriateness of input data.

The exercise presented best practices for managing datasets in real-world applications. The entire process ran in a distributed environment that demonstrated Spark MLlib's capabilities for data processing and parallel operations. The analysis maintained scientific rigor through its refusal to enhance performance metrics artificially while it provided an honest evaluation of model capabilities and data constraints. The method follows professional data science standards better than creating deceptive high accuracy results through excessive model modifications.

The reflective process serves as an essential educational requirement. The value of model failure detection and cause identification matches the importance of developing successful systems. The analysis promotes better understanding of data quality together with strategic feature selection and algorithmic modeling boundaries. Such exercises enable the development of responsible machine learning applications which produce effective results in real-world scenarios.

CHAPTER FIVE: Reflection and Recommendations

Several real-world problems emerged during the Big Data environment setup because the organization operated with various systems. The project launch took place within the Jupyter environment running on Windows local systems at first. The Apache Spark configuration process turned out to be extremely difficult to manage. The Spark installation faced crucial problems because the project team successfully deployed the necessary Java Development Kit (JDK) yet winutils.exe setup remained unsuccessful for running Hadoop-based Spark on Windows. The setup restriction led to continuous connection errors that prevented any advancement of local development. A Docker-based environment received attention as a potential solution to establish an independent platform. The Docker method experienced multiple obstacles during its implementation. The Spark container failed to start correctly because of port conflicts along with missing permissions and network restrictions. The experience shows Docker provides environment consistency yet demands advanced container management and networking skills which might exceed academic time constraints.

The transition to Google Colab occurred when the team installed Spark through pre-written shell commands to handle the system. This configuration required some initial setup but established a reliable cloud computing environment without the operating system compatibility problems encountered with Windows and Docker. All following data processing work and integration and machine learning modeling took place successfully inside Colab. The research supports using Google Colab for future academic Big Data projects because it addresses hardware and OS compatibility challenges that local installations encounter. The platform enables easy access to computing power alongside effortless teamwork and convenient Apache Spark integration. Docker and local IDEs including Jupyter on Windows function best for production applications and situations requiring dependency management control. The setup of Spark and Hadoop requires advanced system-level understanding from users.

Google Colab and other cloud-based platforms provide suitable usability and functionality for academic work but Docker and local installations deliver scalable control at the cost of complex setup procedures.

BIBILOGRAPHY

Dean, J. and Ghemawat, S., 2008. MapReduce: simplified data processing on large clusters. Communications of the ACM, 51(1), pp.107–113.

Zaharia, M., Chowdhury, M., Franklin, M.J., Shenker, S. and Stoica, I., 2010. Spark: Cluster computing with working sets. In: Proceedings of the 2nd USENIX conference on Hot topics in cloud computing. Boston, MA: USENIX Association, pp.1–7.

Zaharia, M., Das, T., Li, H., Hunter, T., Shenker, S. and Stoica, I., 2013. Discretized streams: Fault-tolerant streaming computation at scale. In: Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles. Farmington, PA: ACM, pp.423–438.

Meng, X., Bradley, J., Yavuz, B., Sparks, E., Venkataraman, S., Liu, D., Freeman, J., Tsai, D., Amde, M., Owen, S. and Xin, R., 2016. MLlib: Machine learning in Apache Spark. Journal of Machine Learning Research, 17(1), pp.1235–1241.

Marz, N. and Warren, J., 2015. Big Data: Principles and best practices of scalable real-time data systems. Shelter Island, NY: Manning Publications.

Chen, M., Mao, S. and Liu, Y., 2014. Big data: A survey. Mobile Networks and Applications, 19(2), pp.171–209.

Runkler, T.A., 2012. Data analytics: Models and algorithms for intelligent data analysis. Berlin: Springer.

Rajaraman, A. and Ullman, J.D., 2011. Mining of massive datasets. Cambridge: Cambridge University Press.

Grolinger, K., Higashino, W.A., Tiwari, A. and Capretz, M.A.M., 2014. Data management in cloud environments: NoSQL and NewSQL data stores. Journal of Cloud Computing: Advances, Systems and Applications, 3(1), pp.1–24.

Zaharia, M., Xin, R.S., Wendell, P., Das, T., Armbrust, M., Dave, A., Meng, X., Rosen, J., Venkataraman, S., Franklin, M.J. and Ghodsi, A., 2016. Apache Spark: A unified engine for big data processing. Communications of the ACM, 59(11), pp.56–65.

APPENDIX

https://colab.research.google.com/drive/1TLaneWHdnHYQXL7GTolJ7N0ElE5sVMPv?usp=sharing