**CCT College Dublin**

**Assessment Cover Page**

|  |  |
| --- | --- |
| **Module Title:** | Storage Solutions for Big Data (10 ECTS) |
| **Assessment Title:** | CA1 |
| **Lecturer Name:** | Dr. Muhammad Iqbal |
| **Student Full Name:** | Tomás Ruiz Penin |
| **Student Number:** | sbs23085 |
| **Assessment Due Date:** | 5th November 2023 |
| **Date of Submission:** | 5th November 2023 |

Tomás Ruiz Penin

**Declaration**

By submitting this assessment, I confirm that I have read the CCT policy on Academic Misconduct and understand the implications of submitting work that is not my own or does not appropriately reference material taken from a third party or other source. I declare it to be my own work and that all material from third parties has been appropriately referenced. I further confirm that this work has not previously been submitted for assessment by myself or someone else in CCT College Dublin or any other higher education institution.

[Number of words used in the report excluding cover page, titles, references, diagrams and code: 2084 1](#_Toc2097868845)

[1. Dataset Description 2](#_Toc75931639)

[1.1 Website Access 2](#_Toc808066991)

[1.2 Dataset Overview 2](#_Toc393383646)

[1.3 Language distribution 2](#_Toc993878623)

[1.4 Data Attributes 2](#_Toc564102647)

[1.5 Data Structure Analysis 3](#_Toc1994858981)

[1.6 Data Quality Considerations 3](#_Toc551623385)

[1.7 Licensing, Usage, and Ethical Considerations 3](#_Toc739949044)

[1.8 Accessibility and Maintenance 3](#_Toc1219946120)

[2. Data preprocessing 3](#_Toc554967413)

[2.1 Initial Setup and Loading Data 4](#_Toc472747240)

[2.2 Hadoop Configuration and Data Preparation 4](#_Toc750336685)

[2.3 Data Preprocessing with PySpark 5](#_Toc260086025)

[2.4 Integration with Apache HBase 8](#_Toc1212759413)

[2.5 Data Access and Visualization with HappyBase and Matplotlib 10](#_Toc682981510)

[2.6 Data processing conclusions 14](#_Toc1774310347)

[3. Motivation, objectives and handling of data complexity 14](#_Toc84658251)

[3.1 Motivation 15](#_Toc1916602894)

[3.2 Objectives 15](#_Toc562427808)

[3.3 Handling Data Complexity 15](#_Toc1549334779)

[3.4 Data Processing Choices 16](#_Toc199946618)

[3.5 Programming Language Choice 16](#_Toc959738119)

[3.6 Design Patterns Implemented 17](#_Toc700367084)

[3.7 Conclusion 17](#_Toc1360235271)

[4. References 17](#_Toc1308864026)

Number of words used in the report excluding cover page, titles, references, diagrams and code: 2084

Github link: <https://github.com/nemonio/Storage_Solutions_for_Big_Data_CA1>

1. Dataset Description

The lyrics-data.csv file from the Kaggle dataset "Song lyrics from 79 musical genres" is a comprehensive collection of song lyrics data scraped from the Brazilian website Vagalume.com.br.

1.1 Website Access

**URL (Kaggle Dataset Link)**: <https://www.kaggle.com/datasets/neisse/scrapped-lyrics-from-6-genres?select=lyrics-data.csv>

1.2 Dataset Overview

* **Name:** lyrics-data.csv
* **Source:** Vagalume.com.br
* **Scraped by:** Anderson Neisse
* **Last Updated:** 2 years ago
* **Size:** 434.3 MB
* **Number of Songs:** 379,893
* **Number of Artists:** 4,239
* **Languages:** Primarily English (50%) and Portuguese (41%), among others.

1.3 Language distribution

Here's a summary of the dataset's language distribution:

* **English (en):** 191,814 songs
* **Portuguese (pt):** 157,393 songs
* **Spanish (es):** 9,917 songs
* **Italian (it):** 1,432 songs
* **….**

This distribution indicates a strong presence of English and Portuguese songs, which is reflective of the source website's primary audience.

1.4 Data Attributes

* **ALink**: The URL to the artist's profile on Vagalume.com.
* **SName**: The name of the song.
* **SLink**: The URL to the song's lyrics on Vagalume.com.
* **Lyric**: The full text of the song lyrics.
* **Language**: The language code for the lyrics.

1.5 Data Structure Analysis

**Format**: CSV (Comma-Separated Values)

**Nature of Data**: Semi-structured. While the CSV format is a structured format, the lyrical content within the 'Lyric' field is inherently unstructured text data.

**Complexity Level**: Given the size of the dataset and the unstructured nature of text data, the complexity level is high. The dataset requires advanced text processing and natural language processing techniques to structure and analyze the lyrical content.

1.6 Data Quality Considerations

**Consistency**: Potential inconsistencies due to the nature of web scraping.

**Completeness**: The dataset may have missing information due to the limitations of the scraping process.

**Accuracy**: The accuracy of the lyrics and associated metadata may vary and should be verified for critical applications.

1.7 Licensing, Usage, and Ethical Considerations

**License**: Open Database License for the database and Database Contents License for the contents.

**Usage Rights**: Users of the dataset provided by Anderson Neisse / Kaggle are free to share, modify, and use the dataset for various purposes, adhering to the terms of the Open Database License.

**Ethical Considerations**: The use of the dataset should consider the copyright of the lyrical content as well as the privacy of the artists.

1.8 Accessibility and Maintenance

**Download Requirements**: Requires a Kaggle account for access.

**Update Frequency**: No regular updates, last updated 2 years ago.

**Support and Maintenance**: Limited, as it is a scraped dataset with no active maintenance.

2. Data preprocessing

This report provides a detailed account of the data preprocessing steps taken using Apache Spark on a dataset stored within the Hadoop File System (HDFS). The dataset, consisting of song lyrics and additional metadata, underwent a series of transformations to ensure cleanliness, proper formatting, and readiness for further analysis and storage in HBase. Through the application of PySpark, the data was effectively loaded, cleaned, deduplicated, and eventually stored in a new CSV file format suitable for import into HBase.

2.1 Initial Setup and Loading Data

Before uploading the dataset to hadoop, I converted the file to have tabs as separator instead of commas, due to the fact that PySpark didn´t seem to process the columns correctly with the original. So “lyrics-data.csv” was converted to “**lyrics-dataTAB.csv**” with the same size and data.

2.2 Hadoop Configuration and Data Preparation

The notebook then transitions to configuring Hadoop and preparing data:

|  |
| --- |
| x |
| *Caption 01. Running Hadoop and uploading data* |

**A. Starting Hadoop**: A script named ./hdfs\_extended (similar to running start-dfs.sh and start-yarn.sh) is executed to start the Hadoop Distributed File System (HDFS) and Yet Another Resource Negotiator (YARN).

**B. Directory Creation**: The command hadoop fs -mkdir /CA1 is used to create a new directory in the Hadoop filesystem for project CA1.

**C. Data Transfer**: The dataset with TAB separators is transferred to the newly created directory using hadoop fs -put.

**D. Verification**: The existence of the dataset is confirmed with hadoop fs -ls /CA1.

**E. Data Cleaning Script**: A bash script is executed to replace all double-quote characters in the dataset with single quotes to avoid parsing issues in CSV files.

|  |
| --- |
|  |
| *Caption 02. Bash script to replace all double-quote characters in the dataset with single quotes* |

2.3 Data Preprocessing with PySpark

Following data preparation, then I proceeded with preprocessing the data in the Hadoop filesystem using Apache Spark:

**01. Data Loading**: The fixed dataset is loaded into a DataFrame with appropriate CSV settings such as TAB as separator and multiline support.

|  |
| --- |
|  |
| *Caption 03. Loading the dataset from hdfs to PySpark* |

**02. Null Value Removal**: Rows containing null values are dropped to ensure data quality.

|  |
| --- |
|  |
| *Caption 04. Rows containing null values are dropped* |

**03. Duplicate Removal**: Duplicate rows are removed to maintain data integrity.

|  |
| --- |
|  |
| *Caption 05. Duplicate Removal* |

**04. Newline Character Handling**: Newline characters within the 'lyric' column are replaced with spaces to clean up the data.

|  |
| --- |
|  |
| *Caption 06. Newline Character Handling* |

**05. Language Code Filter**: Rows with language codes exceeding two characters are filtered out for consistency.

|  |
| --- |
|  |
| *Caption 07. Dropping rows with more than 2 chars in Language Column* |

**06. Unique Identifiers**: Unique row identifiers are added to the DataFrame for better data management. Hbase would not handle the dataset correctly without this step.

|  |
| --- |
|  |
| *Caption 08. Adding a ID column* |

**07. Data Export for HBase**: The processed data is saved back to Hadoop's filesystem in preparation for import into HBase.

|  |
| --- |
|  |
| *Caption 09. Dataset Export for HBase* |

2.4 Integration with Apache HBase

The preprocessed data is then moved to HBase, a NoSQL database for handling big data:

**8.1 to 8.5 in the Jupyter notebook**: The steps involve verifying the presence of the dataset in HDFS, starting HBase, accessing the HBase shell, creating a table, and importing the dataset into HBase.

|  |
| --- |
|  |
| *Caption 10. Creating the table in Hbase* |

|  |
| --- |
|  |
| *Caption 11. Importing the dataset into Hbase* |

|  |
| --- |
|  |
| *Caption 12. Results of the importing into Hbase* |

**8.6 Thrift Server**: A Thrift server is started to allow connectivity between HBase and other applications, such as HappyBase.

|  |
| --- |
|  |
| *Caption 13. Thrift Server* |

2.5 Data Access and Visualization with HappyBase and Matplotlib

**HappyBase**, a Python library, was utilized to interact with the HBase data:

Data is extracted from HBase and converted into a pandas DataFrame.

|  |
| --- |
|  |
| *Caption 14. Creating the table in Hbase* |

The data is further processed in Python to format artist links, map language codes to names, and perform data visualization.

|  |
| --- |
|  |
| *Caption 15. Creating the table in Hbase* |

**Matplotlib** is employed to generate 2 plots showing Top 10 Languages with Most Songs and Top 10 Artists with Most Songs.

|  |
| --- |
|  |
| *Caption 16. Visualizations code* |

|  |
| --- |
|  |
| *Caption 17. Top 10 Languages with Most Songs* |

|  |
| --- |
|  |
| *Caption 18*  *. Top 10 Artists with Most Songs* |

* **Additional Analysis and Notebook Appendix**

In the appendix of the Notebook, further analysis steps are described:

**Appendix 01**: The dataset is reduced to the first 20 rows for a smaller, more manageable subset, during all the trials before the final ingestion into Hbase.

**Appendix 02**: Unique language codes are mapped to their full names, and the number of songs per language is determined.

2.6 Data processing conclusions

Throughout the notebook, each step incrementally builds upon the previous ones, progressing from initial data storage with Hadoop, cleaning and preprocessing with Spark, to analysis and visualization using HBase, HappyBase, and Matplotlib. This workflow is indicative of a robust big data pipeline capable of handling vast datasets with complex processing requirements. The notebook serves as a comprehensive example of leveraging a combination of big data technologies to store, process, and analyze data at scale.

3. Motivation, objectives and handling of data complexity

3.1 Motivation

Apache Spark, known for its speed and powerful analytics capabilities, combined with Hadoop's reliable storage, seemed like a great team to confront these challenges. Using Apache Spark and Hadoop for preprocessing was motivated by the need to manage and analyze a complex and large dataset of song lyrics effectively. Due to the big quantity of data, I found it essential to have a robust, scalable, and efficient system for data handling and analysis.

3.2 Objectives

**Data Cleaning**: To remove inconsistencies, duplicates, and null values that could skew analysis results.

**Data Transformation**: To convert the raw data into a structured format suitable for further analysis and machine learning tasks.

**Data Reduction**: To filter out irrelevant information, such as songs with language codes longer than two characters, thereby focusing on significant data.

**Efficiency**: To perform the aforementioned tasks as efficiently as possible to save on computational resources and time, which is vital when working with big data.

**Integration**: To ensure the processed data integrates seamlessly with other tools like HBase for storage and analysis, thereby maintaining a cohesive data pipeline.

3.3 Handling Data Complexity

The complexity of the data was handled through a series of strategic preprocessing steps, each designed to address specific challenges:

**Ingesting multi-line CSVs**: Given the multi-line nature of lyrics, the data was ingested with special handling for escape characters to ensure that each song's lyrics were read as a single record.

**Null Values**: Rows with any null values were dropped to maintain data integrity, as they could represent missing or incomplete information that would undermine the analysis.

**Duplicate Data**: I used Spark's distinct() function to ensure that identical rows were removed, avoiding redundancy in the dataset.

**Text Cleaning**: Newline characters within the lyrics were replaced with spaces to standardize the format, making text analytics more straightforward.

**Language Constraints**: I applied filters to only include rows with language codes of two characters or less to maintain consistency and relevance to the study's scope.

**Scalable Storage**: By leveraging Hadoop's HDFS, I ensured that the system could store and manage the data regardless of its size, while also providing the foundation for distributed processing.

**Data Exporting**: Right after ppst-processing, the data was saved back to HDFS in a clean, structured format, ready for import into HBase.

**Integration with HBase**: Using HBase provided a means to store large amounts of sparse data that could be easily retrieved and updated, making it suitable for handling the variability and size of the dataset.

In conclusion, the complexity of the lyrical data was addressed through careful planning and execution of data preprocessing steps using Spark's distributed data processing capabilities and Hadoop's storage system. This approach laid the groundwork for any subsequent data analysis, ensuring that the data was of high quality and in a format conducive to the intended tasks.

3.4 Data Processing Choices

**Apache Spark over Hadoop**: The choice of using Apache Spark on top of a Hadoop Distributed File System (HDFS), was due to several factors:

**Performance**: Spark offers superior processing speed due to its in-memory computation, which is crucial for iterative algorithms in data processing and machine learning tasks.

**Scalability**: As data grows, Spark's distributed nature allows for horizontal scaling, meaning more nodes can be added to the cluster to handle increased load.

**Fault Tolerance**: Spark's resilient distributed datasets (RDDs) provide a reliable way to process big data by automatically handling failures and data replication.

3.5 Programming Language Choice

I justify the decision to use Python as the programming language, and PySpark as the interface to Spark by:

**Ease of Use**: Python's syntax is clear and concise, making it more accessible.

**Community and Library Ecosystem**: Python has a vast ecosystem of libraries.

**PySpark Integration**: PySpark offers Python API for Spark, which provides a way to leverage Spark's scalability with Python's user-friendly interface.

**Interoperability**: Python scripts can easily be integrated with Hadoop components like HDFS and HBase, as well as with data serialization/deserialization formats like CSV.

3.6 Design Patterns Implemented

**Dataframe API Usage**: The Dataframe API was utilized for its optimization and simplicity in expressing data transformations.

**Chain of Transformations**: A chain of transformations pattern was used, where the output DataFrame of one operation becomes the input for the next, culminating in a series of steps that efficiently convert raw data into a refined form.

**Modular Code**: Code was written in a modular fashion, encapsulating each preprocessing step in a separate segment, thus enhancing readability and maintainability.

**Ecosystem Integration**: The choice to integrate closely with Hadoop’s ecosystem components, such as HDFS for storage and HBase for NoSQL capabilities, demonstrates a design pattern focused on leveraging existing, proven infrastructure for reliable big data management.

In summary, the design patterns and choices made reflect a preference for a scalable, robust, and maintainable system. The combination of Apache Spark for processing, Python for programming, and careful design patterns in code structure and data management, align with best practices for big data processing tasks, providing a solid foundation for any data-driven application or analysis.

3.7 Conclusion

The combined deployment of Apache Spark and Hadoop was justified based on the needs for high-speed processing, fault tolerance, scalability, and compatibility with diverse data processing tasks. My coding approach in PySpark took advantage of the framework's user-friendly APIs to ensure that the data was thoroughly preprocessed and ready for downstream tasks like machine learning, querying, and storage in HBase.

4. References

Apache Software Foundation (2019). *Apache Hadoop*. [online] Apache.org. Available at: <https://hadoop.apache.org/>.

hbase.apache.org. (n.d.). *Apache HBase TM Reference Guide*. [online] Available at: <https://hbase.apache.org/book.html>.

Matplotlib (n.d.). *Matplotlib: Python plotting — Matplotlib 3.3.4 documentation*. [online] matplotlib.org. Available at: <https://matplotlib.org/stable/index.html>.

Pandas (n.d.). *pandas documentation — pandas 1.0.1 documentation*. [online] pandas.pydata.org. Available at: <https://pandas.pydata.org/docs/>.

spark.apache.org. (n.d.). *Welcome to Spark Python API Docs! — PySpark 2.4.5 documentation*. [online] Available at: <https://spark.apache.org/docs/latest/api/python/index.html>.