CMP7248 — Big Data Management and Analytics

Modern Big Data Processing and Analytics

Coursework Assignment 1.2

Alexander Samuel Roberts

M.Sc. Big Data Analytics

Submitted: May 2025

Word count: 5,485

School of Computing and Digital Technology

Faculty of Computing, Engineering and the Built Environment

Birmingham City University

Contents

List of Tables

List of Figures

List of Listings

1 Introduction

The rapid rise and development of various technologies over the recent decades have greatly changed societies across the globe. One area in particular, the field of data analytics, has become a corner piece to many modern organisations due to the insights that can be obtained from such data. A subcategory of this is Big Data. “Big Data” is a term that can have several meanings, but its core meaning is data that can take several forms, is large in volume, is swiftly generated, and is in also form of encoded information (Mills, 2019).

1.1 Project Aim and Objectives

This report aims to contribute to providing an overview of how Big Data is being used in modern day systems and practices.

The project objectives are as follows:

1. Produce a review of the literature to ascertain and evaluate popular and useful Big Data processing paradigms.
   (a) This should include real-world use cases.
   (b) This should also incorporate a review of ethics in Big Data.
2. Investigate trends and patterns within the ”Catch the Pink Flamingo” game data-set using Python and Apache Spark.
   (a) A Big Data environment, such as that proposed by Cloudera Quickstart (Cloudera, 2016), should be set up to help facilitate this process.
3. Use subsequent findings to construct three Machine Learning models to provide further predictive analytical insights into such data. These results should then be evaluated and visualised using appropriate evaluation metrics (e.g., F1-score, precision, for classification).
4. Explore and visualise relations within graphs derived from user chat data of the ”Flamingo Game”. This can be achieved using the Neo4j NoSQL graph database.

5. Summarise the report findings, and the significance Big Data plays, and will continue to play, in society for years to come. Moreover, future work should be discussed.

2 Big Data Processing Paradigms

Big data processing paradigms are specific methods or frameworks used to process a large amount of data, usually either continuously or periodically. There are several types of these methods, but some of the traditional and most recognised ones include stream processing, batch processing, and hybrid processing (Al-Barznji, 2022, pp. 3-6).

These processing paradigms can be applied to structured, semi-structured, and unstructured data.

2.1 Research Methodology

2.1.1 Research Scope

The scope of this research consists of the research material publication date and the publication venue. For this review, publications published between January 2020 and January 2025 have been collected. This was conducted in February and March 2025. The publication venues included were:

- the Institute of Electrical and Electronic Engineers (IEEE) Xplorer,
- the Association for Computing Machinery (ACM) Digital Library,
- Springer Nature Link,
- and Science Direct.

2.1.2 Search Terms

A search string and Boolean operators were used to automatically search for results across publication venues, which included some synonyms of the same keyword to account for semantic discrepancies: (”stream processing” OR ”real-time processing” OR ”streaming analytics” OR ”batch processing” OR ”non-real time processing” OR ”periodic processing” OR ”batch analytics” OR ”hybrid processing” OR ”composite data processing” OR ”big data ethics” OR ”data ethics”).

2.1.3 Selection Criteria

Once the collection of sources were identified, a selection criteria was used to help filter results. The title and abstract of each publication were focused on and filtered using the exclusion criteria described in Table 2.1.

Table 2.1: Study exclusion criteria.

No.	**Criterion**
EC1	Papers without full text available.
EC2	Duplicate research.
EC3	Publications not written in English.
EC4	Papers that do not relate to data processing in the context of Big Data and/or data processing.
EC5	Does not include an abstract or introduction.
EC6	Is unfairly critical or has ambiguous reasoning.

2.1.4 Search Results

Figure 2.1 illustrates the process of filtering, screening and deduplifying sources collected using a PRISMA diagram.

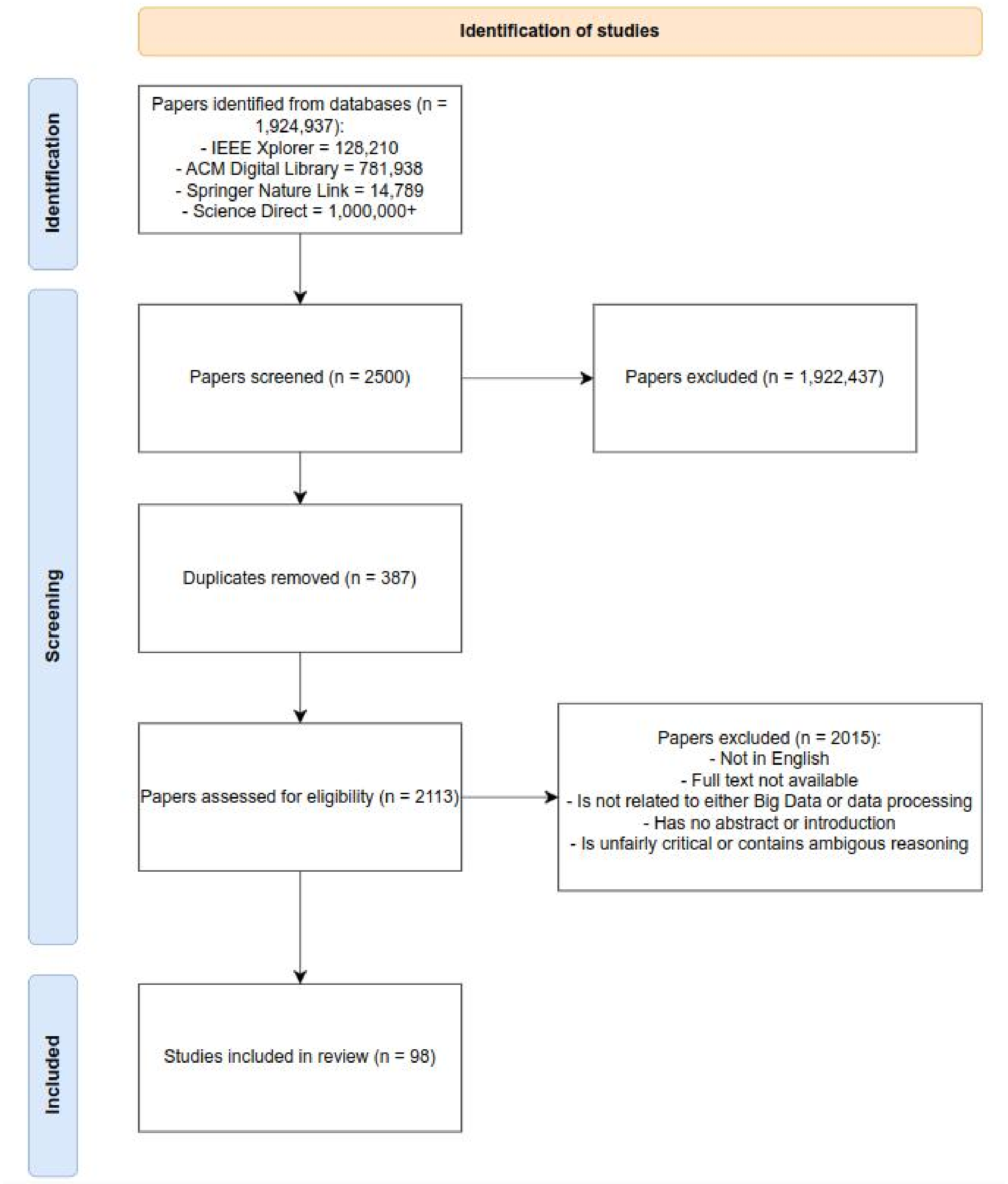

Identification of studies
Identification
Screening
Included
Papers identified from databases (n = 1,924,937):
- IEEE Xplorer = 128,210
- ACM Digital Library = 781,938
- Springer Nature Link = 14,789
- Science Direct = 1,000,000+
Papers screened (n = 2500)
Papers excluded (n = 1,922,437)
Duplicates removed (n = 387)
Papers assessed for eligibility (n = 2113)
Papers excluded (n = 2015):
- Not in English
- Full text not available
- Is not related to either Big Data or data processing
- Has no abstract or introduction
- Is unfairly critical or contains ambigous reasoning
Studies included in review (n = 98)

Figure 2.1: Literature search results PRISMA diagram.

2.2 Stream Processing

Stream processing involves processing continuous volumes of data, often in near real (Jabeen, 2020, p. 49). Moreover, it is used when near real-time, readily available data insights are necessary, especially if the value of data degrades with time (Liu and Buyya, 2020, p. 2). Examples include fraud detection, surveillance monitoring, and traffic monitoring. Live data dashboards are one example of readily presenting data insights gained from stream processing.

Traditionally, stream processing was not particularly popular for Big Data due to a huge drawback: where a single machine can only process so much data at a time. In recent years however, stream processing has been combined with distributed computing to form Distributed Stream Processing Systems (DSPSs) (Liu and Buyya, 2020, pp. 2-3). This provides key advantages, such as fault-tolerance, horizontal scaling, parallel processing, and unified stream management. Horizontal scaling in particular allows an organisation to expand their streaming capabilities more cheaply compared to upgrading each machine itself (known as vertical scaling) (Al-Aamri et al., 2023, pp. 10-11). Examples of some open-source DPDSs include Apache Spark Structured Streaming, Apache Flink, and Apache Storm (Ecker et al., 2025, p. 2).

Figure 2.2 illustrates an example workflow of stream processing.

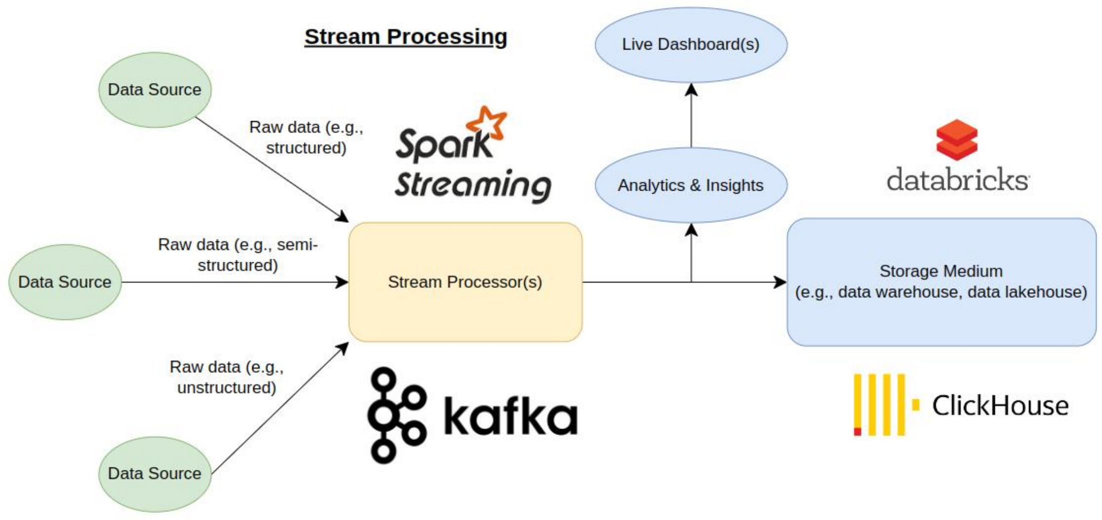


Figure 2.2: Example workflow of stream processing.

2.3 Batch Processing

Batch processing involves collecting and processing data in "chunks". These chunks are often accumulations of data gathered over time, which are then processed at specific time periods, or during set procedures, not continuously (Jabeen, 2020, p. 49). These chunks could be stored in mediums such as data lakes and/or data lakehouses before they are processed. It is also useful for processing historical data as well. This can be done through the use of pipelines, where essentially a set of inter-linked procedures work together to perform a specific action (such as processing data at a set time), as shown by Banerjee (2022, pp. 119-126). Some example technologies that leverage batch processing are Apache Hadoop and Apache Spark, which often are used in conjunction with each other (Chunduri and Cherukuri, 2021, pp. 83-88).

The data storage can be divided into a warm-path and a cold-path. The warm path offers immediate data access for insights and analytics, while the cold-path holds data for longer periods, making it useful for batch processing (Sheikh, Kumar and Ambhaikar, 2021, p. 129). This enables better resource management for organisations, especially if they pay for cloud services to manage their data. Cloud services usually charge more for warm-path usage (as these offer readily available resources) compared to cold-path usage.

Figure 2.3 illustrates an example workflow of batch processing.

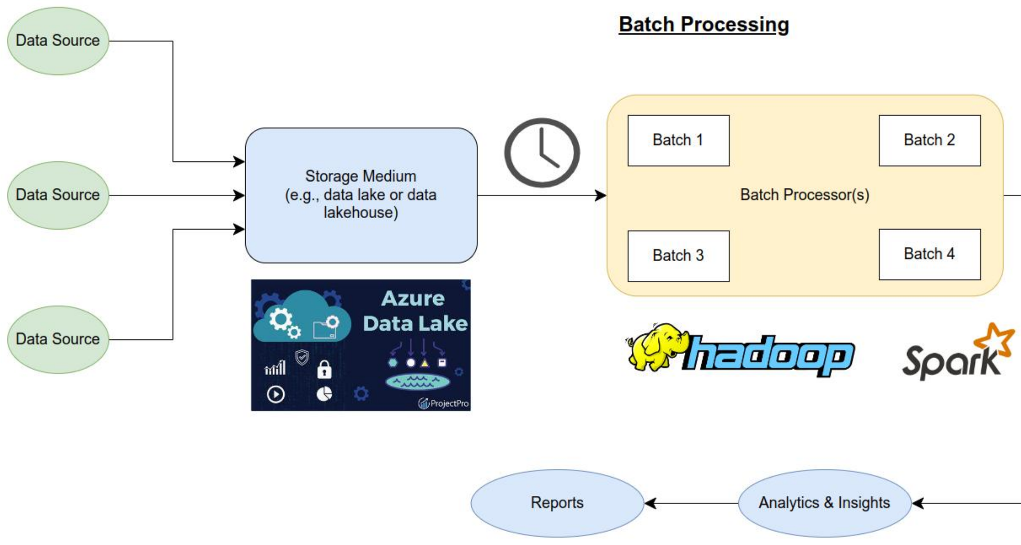


Figure 2.3: Example workflow of batch processing.

2.4 Hybrid Processing

Hybrid processing can be viewed as a combination of stream and batch processing, when a service may need to analyse both real-time data and historical data. For instance, Suthakar et al. (2021) discusses the Lambda Architecture, an architecture designed to handle any requirements for a fault-tolerant and robust system. This architecture has a large degree of flexibility, namely due to the numerous ways stream and batch processing pipelines can be created. One example includes real-time stock market forecasting, where predictive models are leveraged to process both historical data and real-time data, then draw insights based on the results of both (including their significance apart and/or together). Figure 2.4 illustrates an example workflow. But they are not without drawbacks, as highlighted by Song et al. (2023, p. 1488), some key drawbacks include addressing differing file formats (e.g., stream processing may use JavaScript Object Notation while batch processing uses comma-separated values format), decreased performance for multi-version concurrency control (MVCC) systems (which help maintain consistency of data by making multiple copes); which becomes degraded as more unused versions accumulate, increased complexity for data access and synchronisation, and reduced performance isolation.

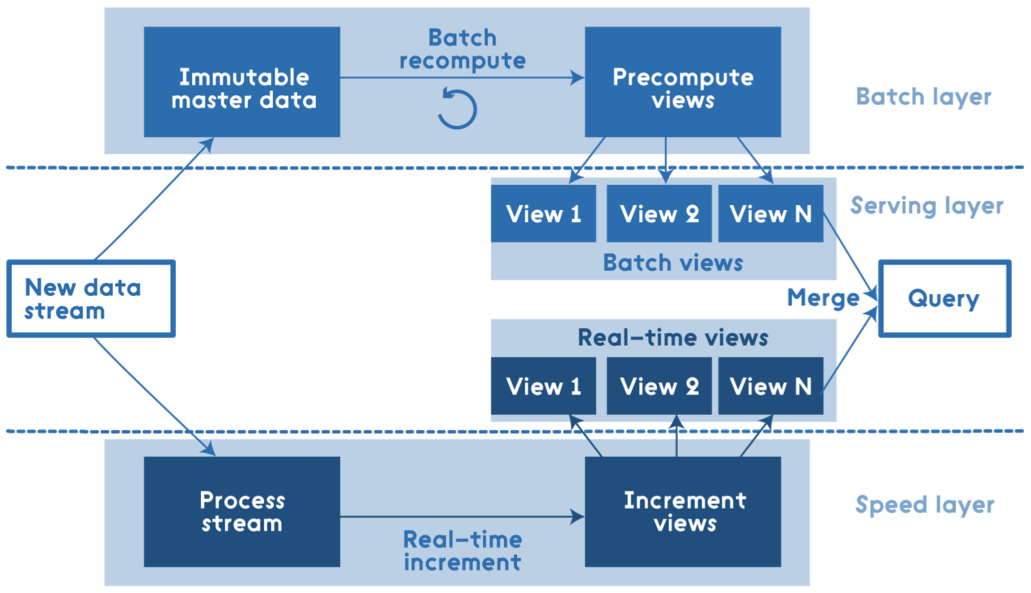


Figure 2.4: The Lambda Architecture

Source: Matyashovskyy (2016).

2.5 Other Processing Paradigms

2.5.1 Hybrid Transactional and Analytical Processing

Another type of hybrid processing is described by Song et al. (2023) and Kim and Moon (2023) as hybrid transactional and analytical processing (HTAP). Online transactional processing (OLTP) systems are commonly used by organisations who prioritise performing data transactions (e.g., purchases from an e-commerce site). In contrast, online analytical processing (OLAP) systems are used to perform large-scale data analysis. OLAP systems often interact with OLTP systems for retrieving the latter's stored information. But this process can often take minutes to hours to complete, namely due to the time it takes to perform the extract-transform-load (ETL) procedure — as it needs to send numerous query requests to the OLTP database Song et al. (2023, p. 1488). For some organisations, data freshness is essential for their business needs and operations. Hence, HTAP was developed to remove the ETL process by combining both OLTP and OLAP systems together, removing the need to transfer the data between the two of them (Lin, 2024, pp. 1-4). Some example platforms that leverage HTAP are SingleStore

(Ravita, 2024) and Google Bigtable (Jacobson, 2024).

2.5.2 The Kappa Architecture

Lipp (2023, p. 12) describes the Kappa Architecture as an alternative to the Lambda Architecture. In the Kappa Architecture, all inputs are treated as stream inputs, including historical data. This makes the structural design of Kappa more straightforward than Lambda, however, the inner complexity becomes more complicated, mainly due to having to convert chunks of data (e.g., batch-based data) into a streaming format. Some technologies that are used to construct Kappa Architectures are Apache Kafka and Apache Storm Penka, Mahmoudi and Debauche (2022, pp. 2-3).

Results by Penka, Mahmoudi and Debauche (2021, pp. 18-19) indicate the Lambda Architecture outperforms the Kappa Architecture in terms of accuracy, but it uses more central processing unit (CPU) and in-memory usage compared to Kappa. It is therefore recommended that Lambda is used for tasks involving accuracy, while Kappa is more suited for tasks which are not accuracy-dependent and require faster processing.

2.5.3 Graph Processing

Graph processing can dynamically be used for stream and batch processing purposes. This is because it can handle dynamic inputs in the forms of graphs (which consist of nodes and edges). Graph databases are often used when relationships between data points are more significant collectively rather than individually. Results by Kotiranta, Junkkari and Nummenmaa (2022, pp. 10-13) indicate graph processing outperforms stream and batch querying for relational databases by about three times, and this only increases the more data relations there are to process. Although, these results are not compared to NoSQL databases. It is important to note, as discussed by Lu et al. (2021, pp. 1-3), the larger and more complex graph there is, the longer it will take to process for certain operations (e.g., calculating centrality measurements). But they have mentioned using graphic processing units (GPUs) can largely mitigate this challenge via parallel processing. Some examples of technologies that use graph processing are Neo4j (2025) and Gephi (2025).

2.6 Industrial Use Cases

As described by Hu et al. (2022, pp. 1-3), traffic management and optimisation are a core piece of Smart Cities. For example, traffic monitoring systems are expected to provide real-time traffic data, which can then be used to update road users to better improve decision-making and optimisation. In particular, emergency services would better reach their destination if they knew certain road conditions (e.g., construction works, congestion, etc.) beforehand, and could therefore take a different approach to reach their destination. Real-time stream processing in this case is essential, which helps update web pyramids, which can handle large amounts of spatial data. Hu et al. (2022, p. 1) notes that GPUs are becoming more widely used in these systems to better reduce the data latency and improve scalability.

As presented by Yuan et al. (2024, pp. 1-2), processing large-scale satellite remote sensor data presents a huge challenge. To tackle this, their use of a one-click batch processing (Snowy Dove) system provides a straightforward but effective method of processing this data regularly, even with many readings at a particular time. Their results are promising, but the transferability of this method could be severely limited to certain proprietary satellite technologies, rather than being generally applicable.

Ristov et al. (2023) leverages the Lambda Architecture to perform stream processing of active electrocardiogram (ECG) monitoring and batch processing for occasional ECG readings every 12 hours, then proceeding to combine the results of each into a single source. This approach helps reduce system maintenance and resource usage (which is particularly important for paid cloud services). Their use of real-world benchmarking data and addressing the limitations of their system makes their research both more reliable and more applicable.

2.7 Ethical Considerations of Big Data

While Big Data can be used to garner new insights, these insights could be invalid due to the nature of the data. Reed-Berendt, Dove and Pareek (2021, pp 1-2) for example, explains that while Covid-19 patient data can be used to analyse epidemiology patterns, these patterns can become bias against disproportionately impacted minority groups (who made up a large portion of Covid-19 fatalities by racial percentage). Furthermore, Nasseef (2020, pp. 93-94) discusses more wide-spread impacts that insights from Big Data can have on society, such as how they can impact automated decision-making by Artificial Intelligence. This can have direct effects such as presenting inaccurate predictions, or indirect effects by making jobs done by humans redundant — which can lead to lay-offs.

Given that Big Data can contain sensitive information regarding its population, this can lead to many legal and ethical challenges for its use and regulation. Bosman, Oladepo and Ngambeki (2024, pp. 69-70) highlights factors such as information rights, accountability (for who manages the data) and control (for who can have access to it and when), and property rights as some of the key considerations regarding data security and standards. Novak and Pavlicek (2021, p. 8) goes on to outline that governments and regulatory bodies attempt to impose legal frameworks and regulation for Big Data, such as the General Data Protection Regulation (GDPR), which is used by most European nations. This helps both guide organisations on how to process data in a safe and ethical manner, while also giving assurances and rights to the subjects of that data. A core aspect of data protection, presented by Chen and Quan-Haase (2020, pp. 4-6), is personally identifiable information (PII). This is a critical aspect of a person's privacy and security, as this information can be used to identify them individually. Therefore, to help resolve this, techniques such as anonymisation and risk assessments are done to help make the data no longer PII. But Rupp and Grafenstein (2024, p. 18) emphasises that these methods are not foolproof. For example, depending on the anonymisation technique used on some PII, with advances in technologies and services, a malicious actor could still use that information to target an individual, therefore making it PII despite being anonymised.

To summarise, the use of Big Data brings about endless opportunities and developments, but it should be treated with caution when using insights for decision-making, and be subject to regulation and standards.

3 Big Data Case Study: Catch the Pink Flamingo

The Flamingo Game data is split into two groups of comma separated value (CSV) files. One group represents in-game data such as event tracking, user strategy, and team assignments, etc. The other group represents chat information within the teams. The former group will be the focus of this section, while the latter will be analysed in Section 4.

3.1 Environment Setup and Data Ingestion

A big data processing environment should be utilised to better simulate real-world industrial settings. To achieve this, a Docker (Docker Inc., 2025) container will be used to host an all-in-one processing platform called Cloudera Quickstart (Cloudera, 2016). This platform contains a variety of connected services commonly used in big data processing, such as Apache Hadoop, Apache Spark, Apache Hive, HBase, and more. Appendix C.1 contains the relevant YAML code to properly create the container using Docker Compose. It may be necessary to enable older container support on newer Docker installations. Moreover, it may be necessary to restart some services within the container, as shown in Appendix A. Additionally, other services which utilise the same ports may need to be suspended while the container is active. Finally, it is necessary to update recognised hosts on the host computer, as shown in Appendix A.1.

There are eight CSV files that concern the in-game data (as shown in Figure B.1), each named as 'ad-clicks', 'buy-clicks', 'game-clicks', 'level-events', 'team', 'team-assignments', 'users', 'user-session'. All columns within all of the files are described in Table B.1. Additionally, Table B.2 describes an overview of each file's purpose. Figure 3.1 illustrates the relationships between these files using a conceptual entity-relationship diagram. These files should be uploaded to the Hadoop Distributed File System (HDFS), and also, have their contents copied to the NoSQL database HBase. This is so the original files are safety stored on HDFS, while the actual contents are readily available for ad-hoc querying in HBase. Both tasks can be completed using the Python code in Appendix C.2.

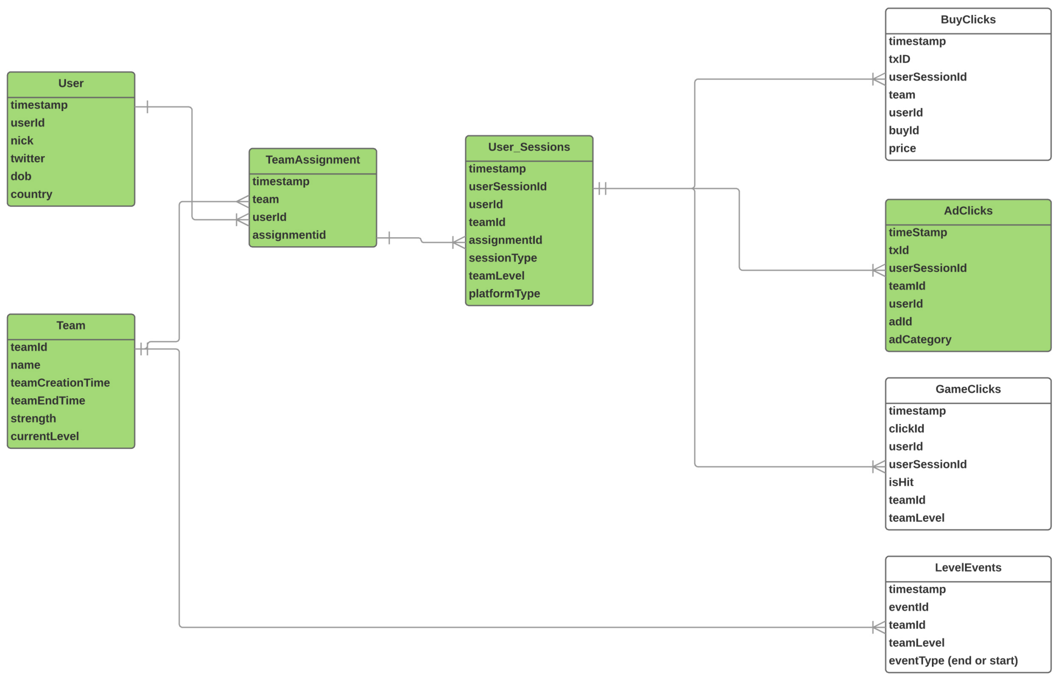


Figure 3.1: Conceptual entity-relationship diagram of the Flamingo Game data.

Source: Rokobo (2024).

3.2 Data Preprocessing for Analysis

With the data now readily available in HBase, it can be extracted to in-memory storage, ready for analysis and manipulation. This can be done using Apache Spark, a distributed data processing framework, which is often used for big data processing due to its versatile and efficient nature (Nudurupati, 2021, p. 29). Due to its distributed design, it can scale horizontally with demand while still maintaining useful data manipulation capabilities. To help maintain consistency, the data relating to the appropriate file (as stored in HBase), should remain separate and not joined into a One Big Table format. This is because not all the tables share a common foreign key (as shown in Figure 3.1), making it difficult to associate them all together at once. A Python dictionary can be used to keep the data separate, where the “key” refers to the original file name, and the “value” corresponds to the Apache Spark dataframe, which will contain the data retrieved from the respective column family in the HBase database.

Once the data is ingested into memory, other preprocessing steps (for preliminary analysis) include:

- handling missing values through complete case analysis,
- removing duplicates,
- and fixing semantic schema differences (e.g., discrepancies in column names).

3.3 Exploratory Data Analysis

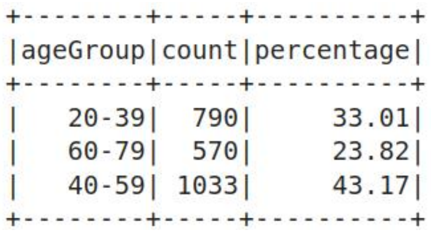
The following visualisations will be available in a Python Jupyter Notebook that will accompany this report.

3.3.1 User analysis

Analysing the available data into the users of the Flamingo Game can provide insights into what categories of people have been attracted to and continue to use the game. Figure 3.2 shows the number of unique users on the platform, while Figure 3.3 illustrates the distribution of age groups who use the game. The results from the former illustrate the enormity of data that can be collected from only a few thousand users. Moreover, the visualisation from the latter clearly shows the main age groups for this game, in order of largest to smallest, are 40-59 year olds, 20-39 year olds, and 60-79 year olds. There are no users below 20 years old or above 80 years old. This suggests future improvements for the game should be aimed at target age ranges for newly-fledged adults and middle-aged persons, if they wish to monopolise on their current target audience.

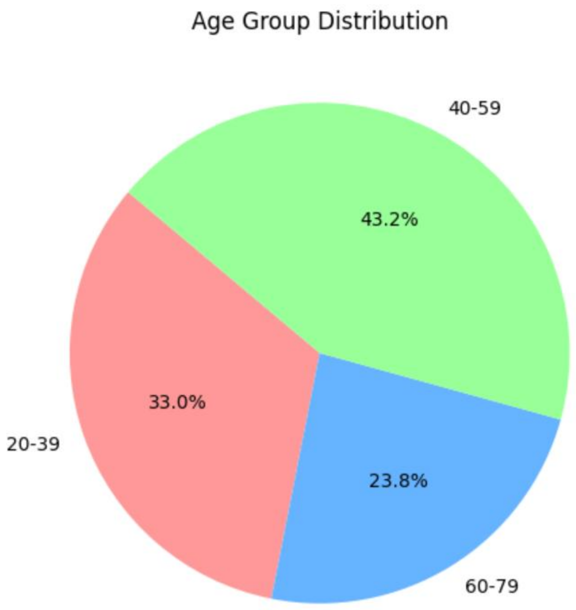
```
2393
```

Figure 3.2: The number of users on the Flamingo Game platform.



```
+--------+-----+----------+
|ageGroup|count|percentage|
+--------+-----+----------+
|   20-39|  790|     33.01|
|   60-79|  570|     23.82|
|   40-59| 1033|     43.17|
+--------+-----+----------+
```

(a) Statistics table of age groups.



(b) Pie chart of age groups.

Figure 3.3: Age group distribution.

Figure 3.4 shows a real-world heatmap of the different concentrations of users across the globe. The visualisation in this figure indicates most of the users are stationed in Europe, Africa, and Mexico. These insights could be useful for the developers if they wanted to implement any special events which resonate with their user-base. For instance, they could make a special event where the challenges are replicas of some European games.

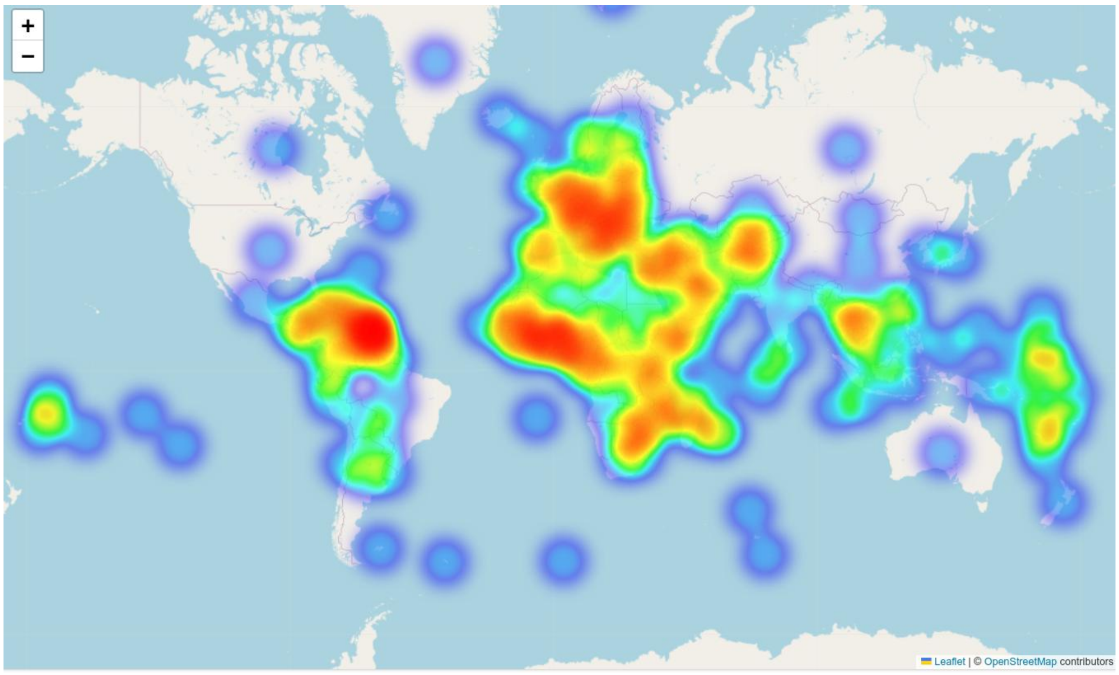


Figure 3.4: Real-world heatmap of user concentrations.

3.3.2 Platform usage

Figure 3.5 visualises what platform mediums are most frequently used for playing the Flamingo Game. As indicated by the figure, the game is mainly played on mobiles like iPhone and Android, then Operating Systems such as Windows, Linux, and Mac. This information could be used to help decide whether to optimise the game for mobile use or for other platforms.

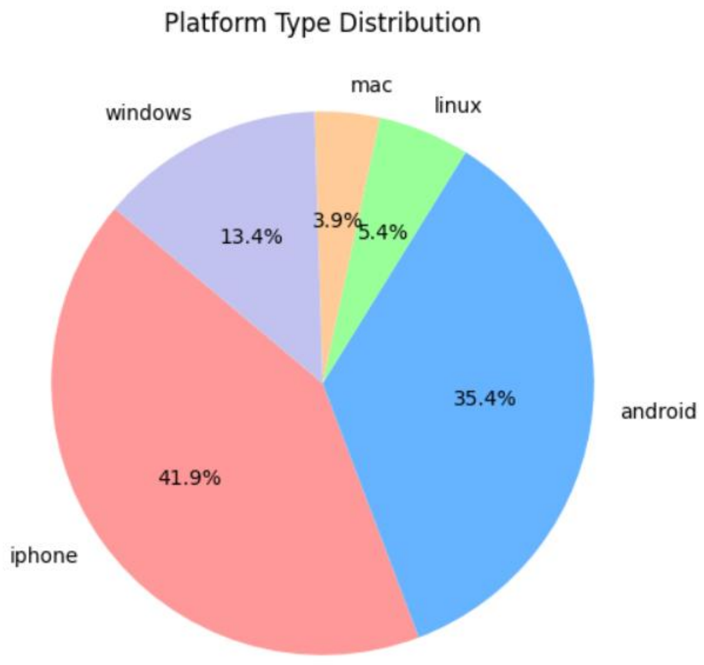


Figure 3.5: Distribution of game usage by platform type.

3.3.3 Team analysis

Figure 3.6 displays the top 10 teams by strength, with team 'vBm7Odv' having the highest strength. This is useful for indicating what team statistics should be examined to better measure a team's performance and potential. Figure 3.7 shows the distribution of strength values across the teams; separated into "bins" for easier visualisation. The results indicate the values do not conform to certain parametric distributions such as the Normal Distribution. This implies mean averages of "strength" would not be as representative nor insightful for investigating general trends into a team's capabilities and progress.

Figure 3.8 shows the oldest registered teams. Interestingly, the third oldest team (with a team ID of 161) also has the highest strength (as shown in Figure 3.6). This hints at the fact that teams that have played the game for longer will likely have higher strength.

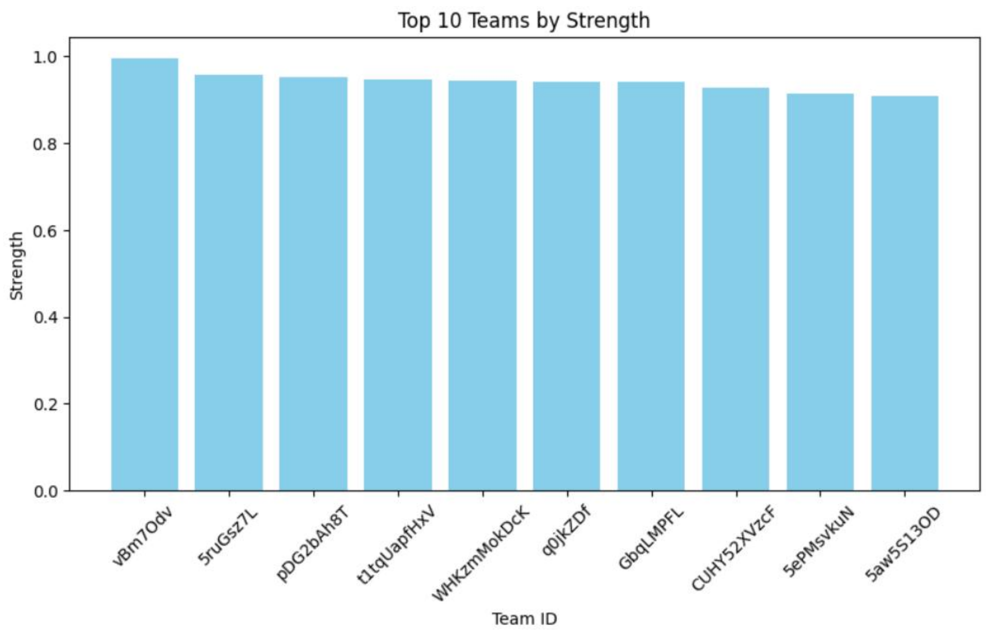


Figure 3.6: Bar chart of the top 10 teams with the highest 'strength'.

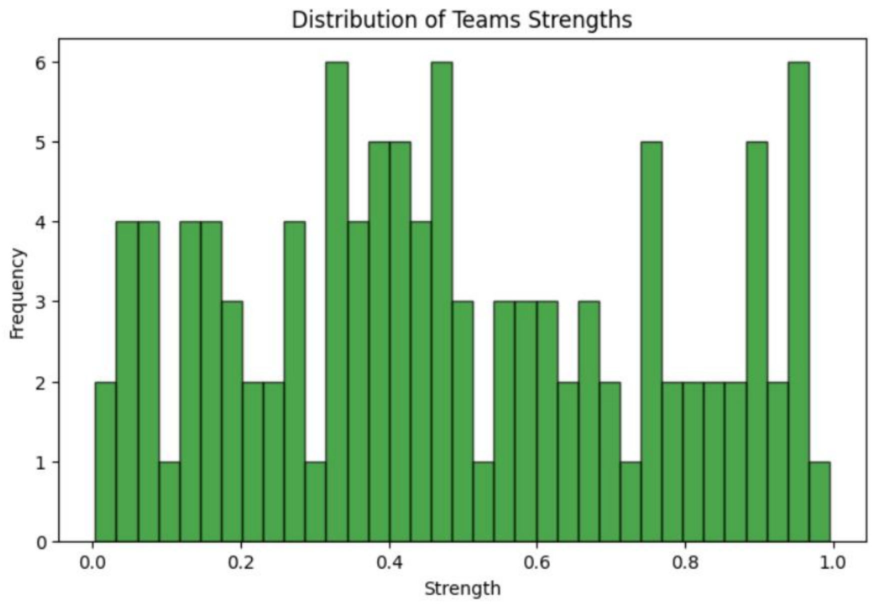
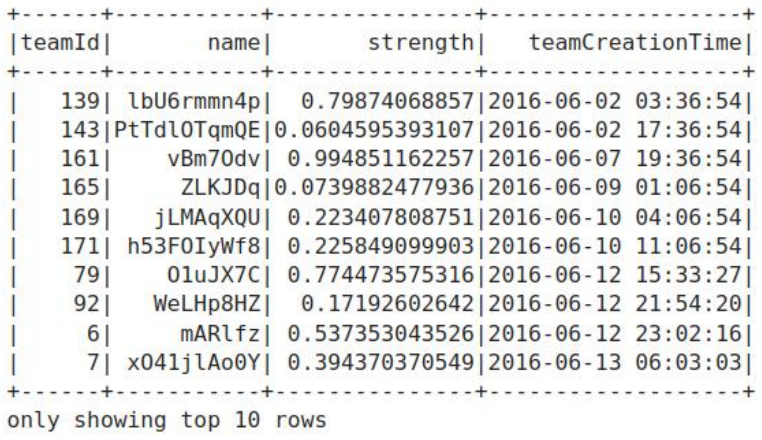


Figure 3.7: Distribution of team strengths in the form of a Histogram.



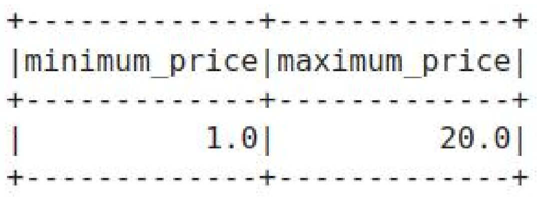
```
+------+-----------+--------------+-------------------+
|teamId|       name|      strength|   teamCreationTime|
+------+-----------+--------------+-------------------+
|   139| lbU6rmmn4p|  0.79874068857|2016-06-02 03:36:54|
|   143|PtTdlOTqmQE|0.0604595393107|2016-06-02 17:36:54|
|   161|    vBm7Odv| 0.994851162257|2016-06-07 19:36:54|
|   165|     ZLKJDq|0.0739882477936|2016-06-09 01:06:54|
|   169|   jLMAqXQU| 0.223407808751|2016-06-10 04:06:54|
|   171| h53FOIyWf8| 0.225849099903|2016-06-10 11:06:54|
|    79|    O1uJX7C| 0.774473575316|2016-06-12 15:33:27|
|    92|   WeLHp8HZ|  0.17192602642|2016-06-12 21:54:20|
|     6|     mARlfz| 0.537353043526|2016-06-12 23:02:16|
|     7| xO41jlAo0Y| 0.394370370549|2016-06-13 06:03:03|
+------+-----------+--------------+-------------------+
only showing top 10 rows
```

Figure 3.8: The oldest registered teams.

3.3.4 Adverts and purchases

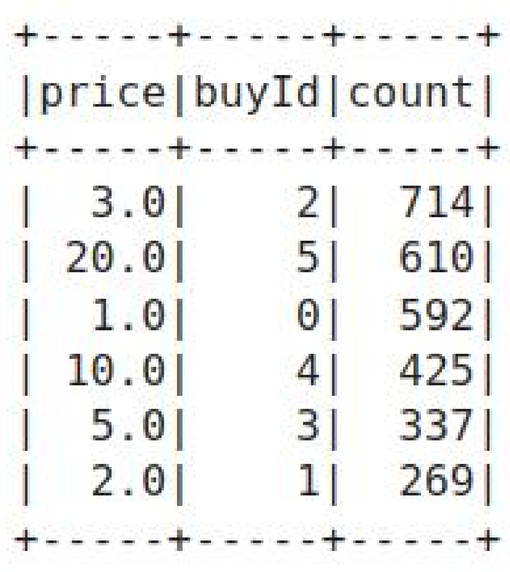
Figure 3.9 shows the minimum and maximum prices for all available products, which stands at 1 unit and 20 units respectively. Figure 3.10 presents all the available products along with their price and the item purchase frequency. It is clear from this that item 6 ('buyId 5') is the most expensive item, while item one ('buyId 0') is the least expensive. Although, item 3 ('buyId 2') appears to be the most frequently bought item (but not by a ginormous amount), indicating its popularity or usefulness to the players.

Figure 3.11 presents the users with the most expenditure for in-game purchases. It has also been colourised to show the age groups of these individuals. The chart implies 40-59 year olds spend the most on in-game purchases for this specific game, followed by 20-39 year olds and 60-79 year olds respectively. This can be helpful for future decision-making on how purchasable items could be improved, or what kind of new products could be added that attract those user demographics.



```
+-------------+-------------+
|minimum_price|maximum_price|
+-------------+-------------+
|          1.0|         20.0|
+-------------+-------------+
```

Figure 3.9: The minimum and maximum prices of all purchasable products.



```
+-----+-----+-----+
|price|buyId|count|
+-----+-----+-----+
|  3.0|    2|  714|
| 20.0|    5|  610|
|  1.0|    0|  592|
| 10.0|    4|  425|
|  5.0|    3|  337|
|  2.0|    1|  269|
+-----+-----+-----+
```

Figure 3.10: Most frequently bought items, their respective prices, and purchase frequency.

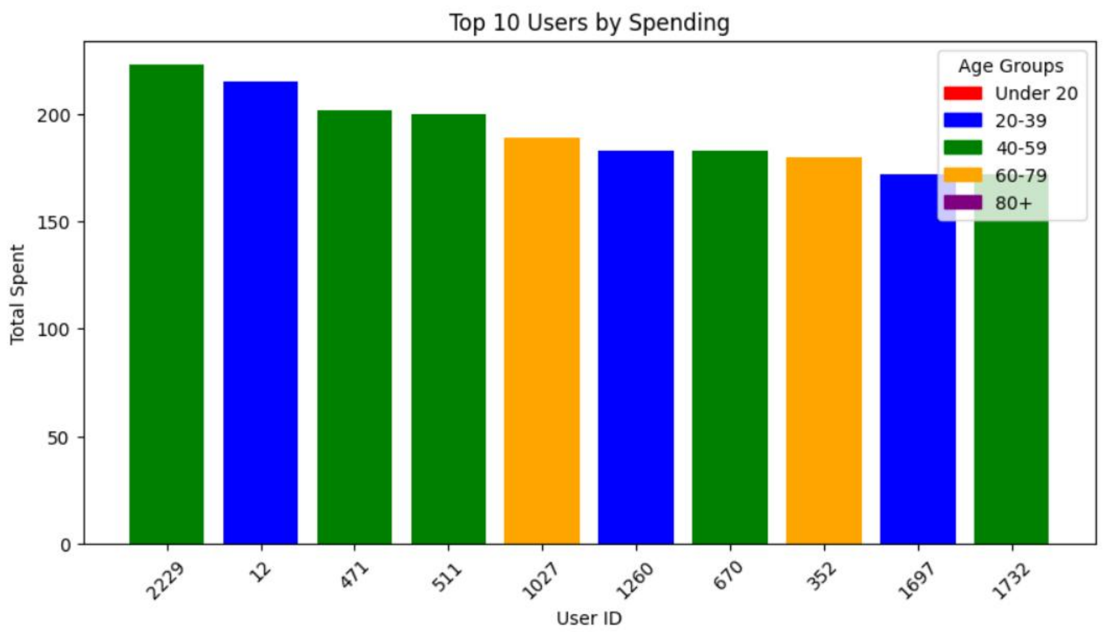


Figure 3.11: Top 10 in-game spenders, with age group colourisation.

3.4 Predictive Analytics (Machine Learning)

The code for the following sections will be available as part of a Jupyter Notebook that will be submitted alongside this report.

3.4.1 Clustering: estimating the average ad clicks against revenue

As indicated by Figure 3.10, revenue is generated by users clicking on posted adverts in-game, then possibly purchasing the item(s). Clustering can be applied in this case to help determine how many ad clicks precede a purchase. This can be useful for identifying categories of user groups (e.g., "quick buyers", "progressive buyers", etc.) and optimising ad strategies. Based on a comprehensive survey by Ezugwu et al. (2022, pp. 4-12), there are two main categories of clustering algorithms: hierarchical clustering and partition clustering. Hierarchical clustering involves partitioning data objects into hierarchical levels, which then help highlight relationships at different levels of granularity. Partition clustering on the other hand operates by assigning each data point to a group based on its characteristics. Unfortunately, hierarchical clustering becomes impractical for larger datasets due to its $O(n^2)$ time complexity nature, whereas, K-Means clustering (which is a type of partition clustering) scales well with larger datasets due to its $O(n)$ time complexity. K-Means also works well with purely numerical data. Therefore, K-Means will be used for the clustering application.

To achieve this, feature selection of two numerical categories representing the total number of clicks per user and the amount of revenue generated per user, must be leveraged. To complete this, other feature engineering techniques such as aggregation and summarisation can be applied. Once created, these columns can be combined to form the final features that will be inputted to the K-Means algorithm. There is a slight issue with K-Means however, the number of clusters needs to be set manually. This can be problematic mainly due to the parameter bias introduced into the process. A common method to resolve this is the Elbow Method, where essentially the optimal number of clusters is brute-forced by sequentially evaluating each amount of set clusters, then plotting the results on a chart. The solution of applying this optimisation step is shown in Figure 3.12.

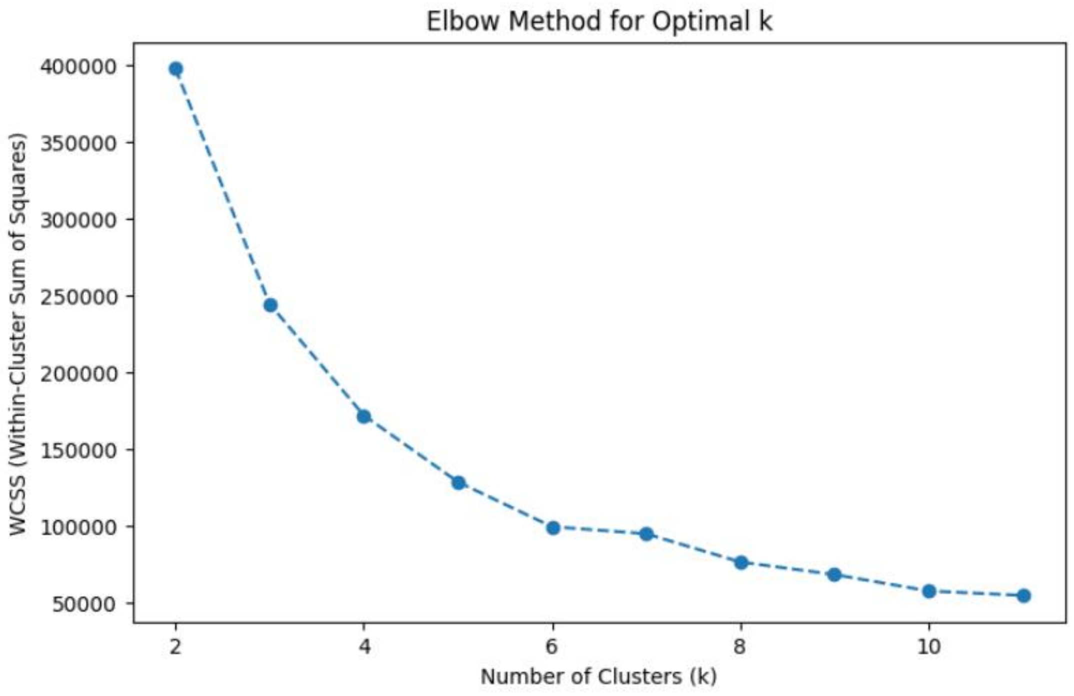
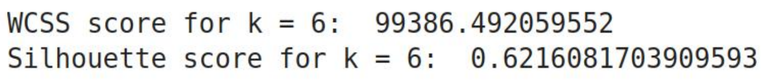


Figure 3.12: Applying the Elbow Method to determine the optimal number of clusters for K-Means.

As shown in Figure 3.12, the lines form a sort of ”elbow”, hence the method name. When using this method, a metric often used to evaluate how tight the clusters are is Within-Cluster Sum of Squares (WCSS) (Khattabi et al., 2024, p. 10). The lower the WCSS, the more likely the clusters are (on average) well-defined and have less overlap. Six clusters appears to be an optimal number of clusters. Therefore, the model with six clusters should be used to construct the final model. There are three evaluation metrics for this model: WCSS, Silhouette Score, and the cluster size distribution. Silhouette Score helps determine how well were the data objects categorised into their respective clusters. The Silhouette Score and WCSS results are shown in Figure 3.13. In addition, Figure 3.14 shows the final model cluster size distribution.



```
WCSS score for k = 6:  99386.492059552
Silhouette score for k = 6:  0.6216081703909593
```

Figure 3.13: WCSS and Silhouette Score of the final clustering model.

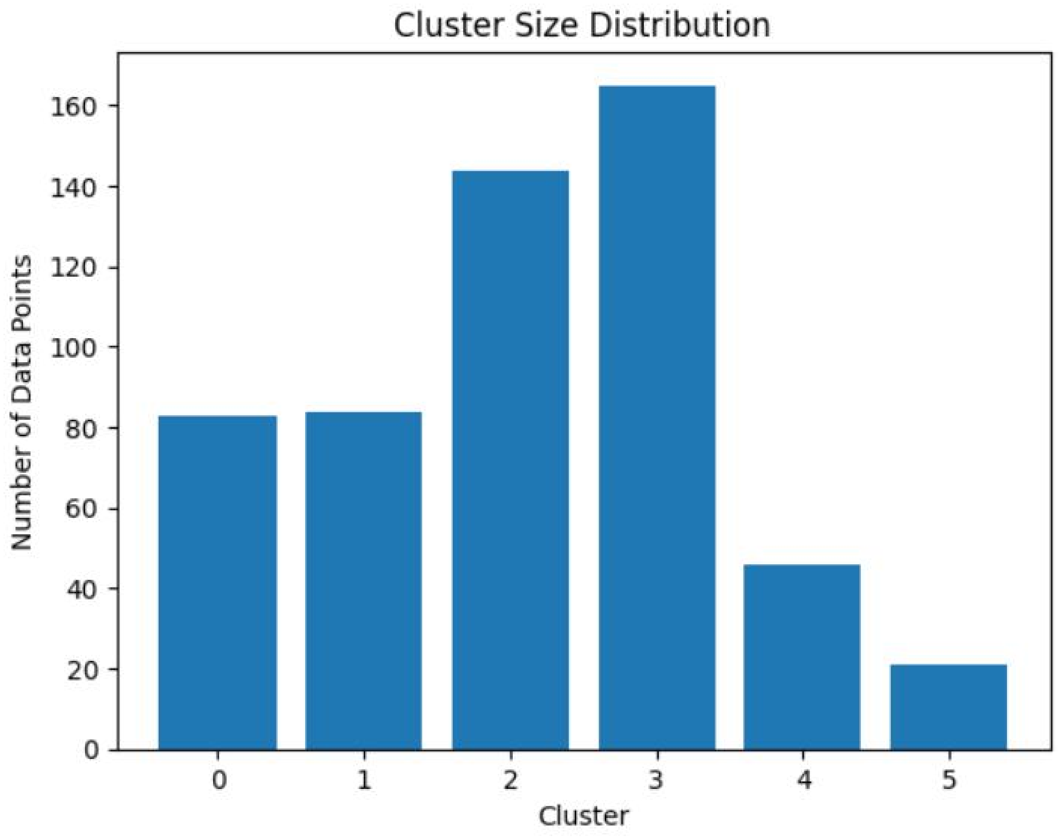


Figure 3.14: Final clustering model cluster size distribution.

The high WCSS score in Figure 3.13 is due to its nature of being the sum of squared values. The Silhouette Score being above 0.6 indicates a fair number of data objects have been assigned correctly to their respective cluster, but there may be a lot of overlap between them. This can be investigated further by actually visualising the clusters, as shown in Figure 3.15. Furthermore, based on Figure 3.14, it is clear more data objects have been placed into clusters two and three, while fewer data objects have been placed in clusters four and five. This suggests the clusters are imbalanced, indicating that some clusters (such as two and three) might be more insightful compared to others when examining patterns and trends.

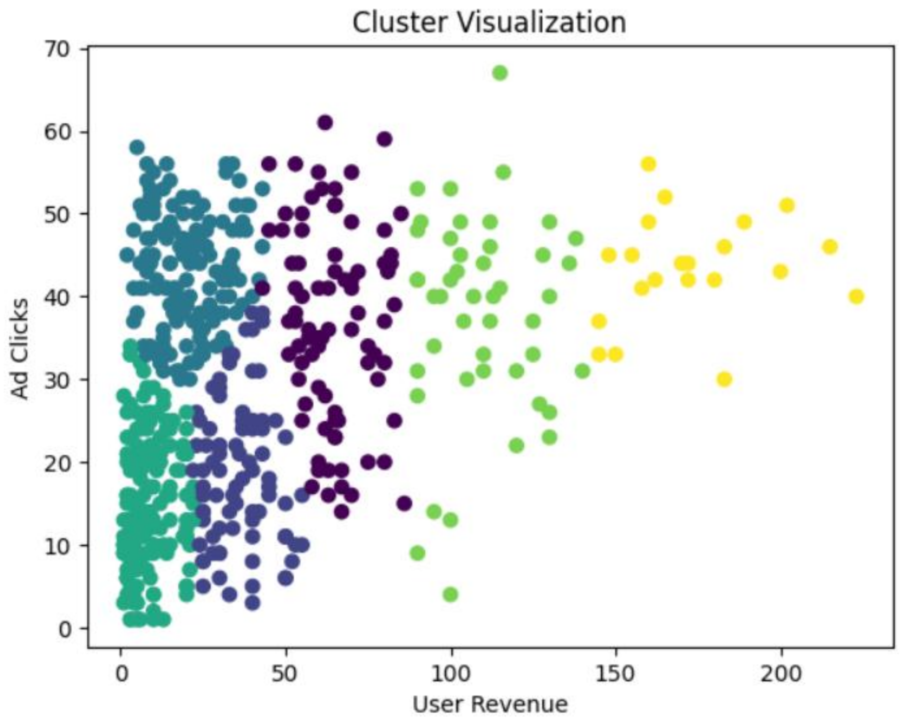


Figure 3.15: Visualising the clusters.

3.4.2 Binary classification of user retention

When a user ends a game session, it would be useful to estimate how likely they are to play another session within a given time period. This can reveal insights into engagement patterns, which can indicate the continuous popularity of the game and its features. User retention in this case can become a binary classification task, as the user can either return within 24 hours from their last game session to play another session, or they will not. Some properties that should be investigated to help construct the input features for the classification task include:

- each session duration,
- the length of time since the user's last session,
- how many sessions the user has had in total,
- the average time between the user's sessions,

- and the time of day it is.

The reasoning for the mainly time-related attributes mentioned is due to how the time played or not played on a game can greatly affect the likelihood that individual will play the game again. The target variable in this case will be a binary label (e.g., 1 or 0) indicating whether the user will play another gaming session within 24 hours. Before deciding on what appropriate classification algorithms should be used, it is important to determine whether the data is balanced. This can also impact what evaluation metrics to use as well. Figure 3.16 shows the distribution for the retention rate of users who play again, and those who do not, within 24 hours. As shown, the class labels are imbalanced, which can undermine models and results. Random undersampling will be used to balance the class labels, as it inputs less algorithmic bias into the process. Figure 3.17 displays the outcome of the undersampling method.

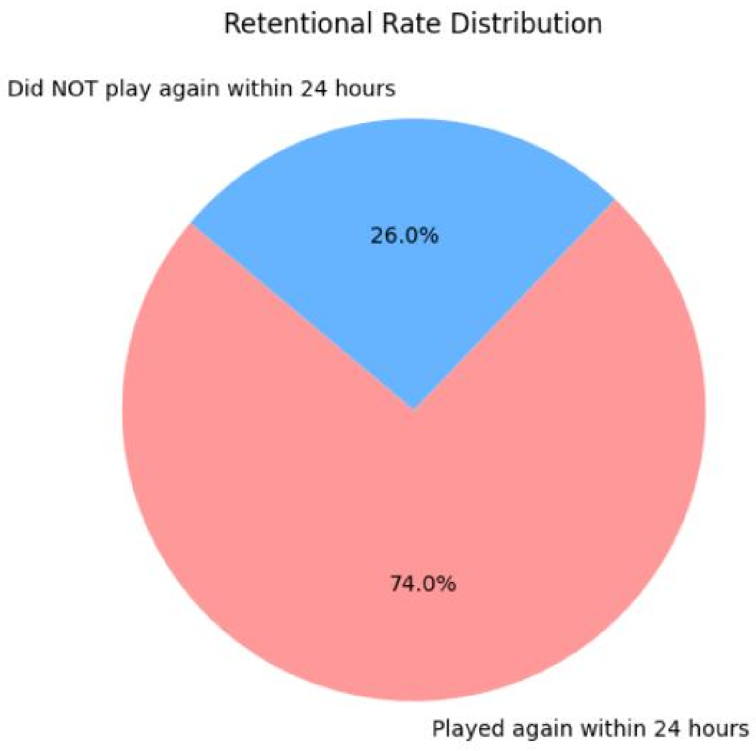


Figure 3.16: Retention rate for players who did and did not play again within 24 hours.

Random Forest and Logistic Regression will be used for the classification algorithms.

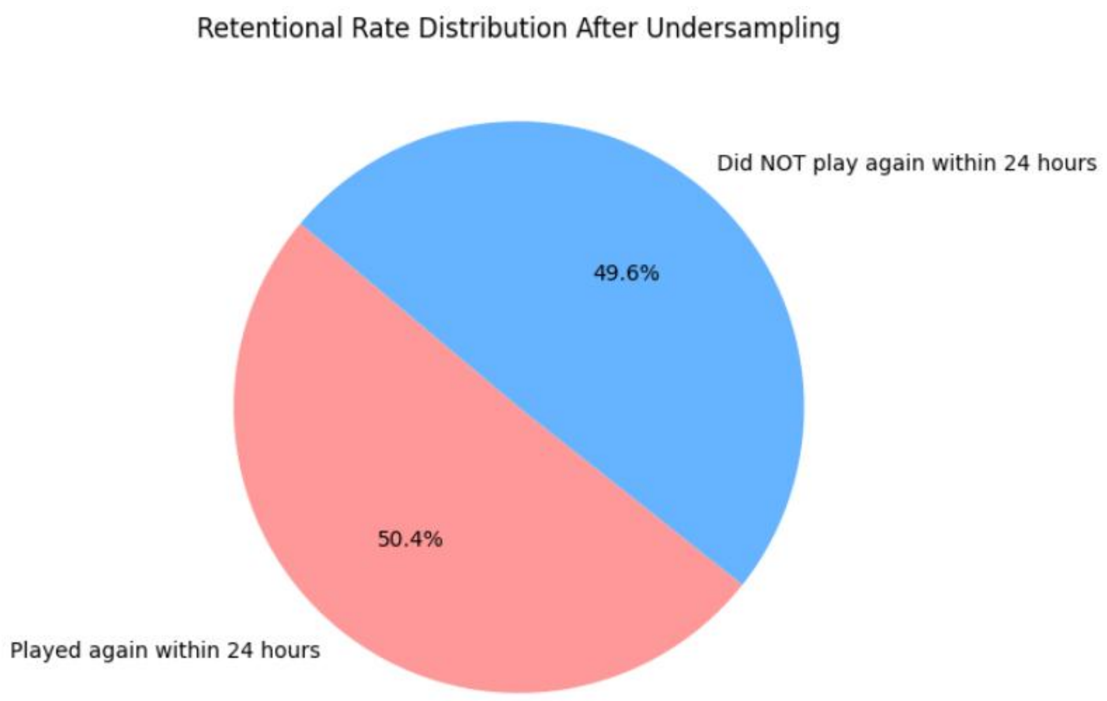


Figure 3.17: Results of undersampling the majority class for the retention rate.

Random Forest is an ensemble method which uses bootstrap aggregation of decision trees to create a strong classifier, which is less prone to individual bias from each tree. Additionally, it can also evaluate feature importance during the process through the use of Entropy Loss and Gini Impurity (Graf, Zeldovich and Friedrich, 2022, p. 6). Logistic regression works by evaluating the probabilities of a certain data object belonging to a target category, namely using the Sigmoid curve (as indicated in Figure 3.18) (Ali et al., 2021, p. 4). It also makes results more explainable as well (as it can be visualised on more simple charts). Both methods can handle numerical and categorical input features. Some common but insightful classification evaluation metrics include Area Under the Curve (AUC), F1-score, precision, recall, and Matthew Correlation Coefficient (MCC) (Szabo et al., 2024, pp. 1-5). AUC is particularly useful for binary classification tasks as it can clearly indicate how well a model performs across multiple probability thresholds. Moreover, MCC is arguably better for evaluating a model's overall performance as it takes into account all four confusion matrix terms, whereas, F1-score only considers the positive terms; it ignores false negatives.

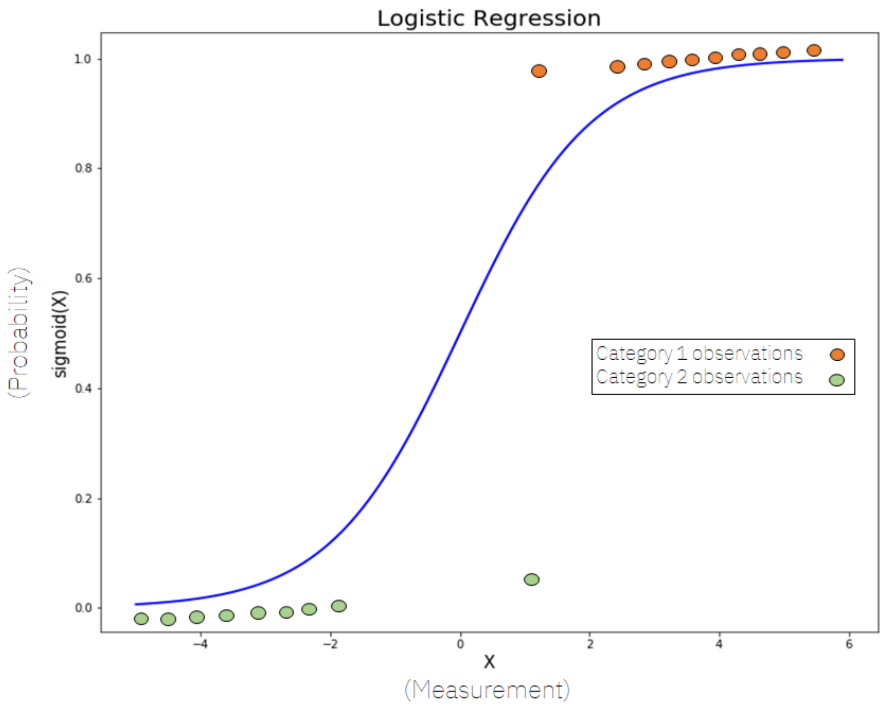
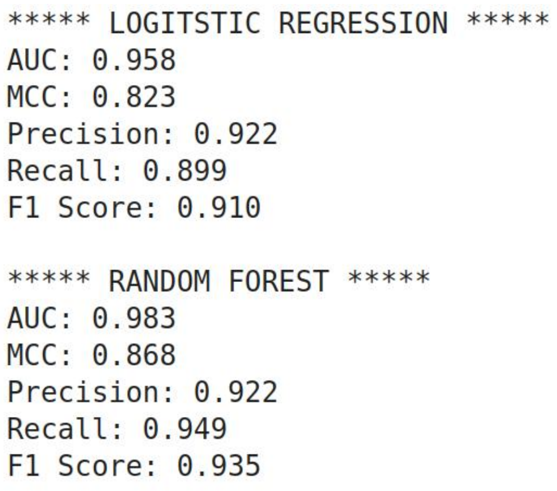


Figure 3.18: Illustration of a Logistic Regression line.

Source: Thorn (2020).

Figure 3.19 illustrates the results of the final models. Overall, both models generalised well, with the Random Forest model slightly outperforming the Logistic Regression model. In almost every metric, Random Forest outperformed Logistic Regression, achieving an F1-score of 0.935 versus 0.910 respectively. Furthermore, the Random Forest model had a higher MCC score (0.868) compared to the Logistic Regression model (0.823), indicating the former is more well-versed overall. Besides the models themselves, the good results could also be explained due to the synthetic nature of the data, which makes it more likely the patterns will be less erratic. In addition, as it is a binary classification task, even if the model "guessed", it still has a "50-50" chance of getting the right answer, compared to a 25% chance in multi-classification tasks for instance.



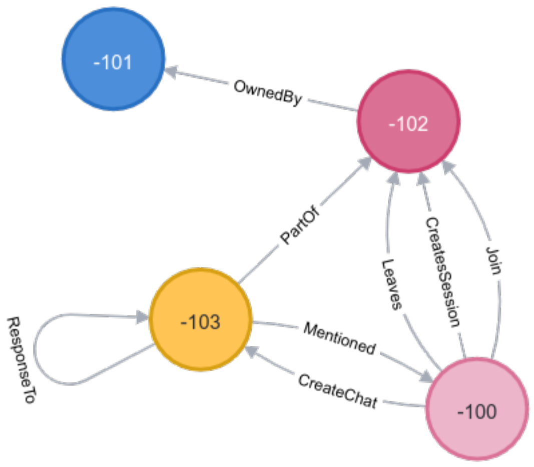
```
***** LOGITSTIC REGRESSION *****
AUC: 0.958
MCC: 0.823
Precision: 0.922
Recall: 0.899
F1 Score: 0.910

***** RANDOM FOREST *****
AUC: 0.983
MCC: 0.868
Precision: 0.922
Recall: 0.949
F1 Score: 0.935
```

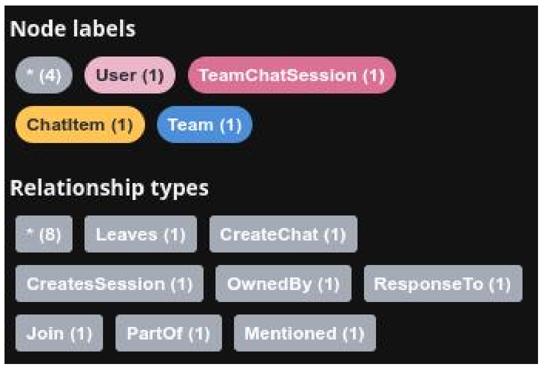
Figure 3.19: Final classification results of both models.

4 Graph Analysis

As mentioned in Section 3, the second group of Flamingo data files includes chat data regarding conversations within the teams. In total, there are six CSV files: 'create_team_chat.csv', 'item_team_chat.csv', 'join_team_chat.csv', 'leave_team_chat.csv', 'mention_team_chat.csv', 'respond_team_chat.csv'. There are four attributes which correspond to all files: the user ID, the team ID, the chat session ID, and the chat item timestamp (which is in the form of a Unix timestamp). Similar to the process described in Section 3.1, the files should be uploaded to the HDFS storage, then be extracted when needed. They will not be uploaded to the HBase storage, as instead, a NoSQL graph database called Neo4j will be used to visualise, analyse, and make the data readily available. A graph database should be used when relationships between data objects should be prioritised; as graph databases are optimised to handle large numbers of both simple and complex relationships (through the use of nodes and edges in Graph Theory). The files will be uploaded to HDFS by re-using the code described in Appendix C.2. The Cypher code (the official query language for Neo4j) for downloading these files into Neo4j, and any subsequent Cypher code segments, can be found in Appendix C.3. Figure 4.1 shows the schema for the chat system database. The colours of the nodes (representing their labels) are consistent across all subsequent visualisations.



(a) Condensed schema.



(b) Schema metadata.

Figure 4.1: Graph database schema.

Figure 4.2 shows two example chat sessions (6777 and 6778) being joined by several users: one by team 188, and the other by team 177. The flurry of lines represents the frequency of users joining and leaving the chat session. This highlights the amount of potential cooperation and communication within the teams, as the more relationships there are, the more likely the individual is proactively trying to play a part in the team. As indicated in Table 4.1, the average degree of activity for a user (joining, leaving, and creating sessions) is around 9.29 (3 s.f.), which implies most users are fairly active with their team communication. The 'TeamChatSession' average degree centrality of 'TeamChatSession' is 0 as it has no out-direction towards 'ChatItem' — the degree centrality is also using direction.

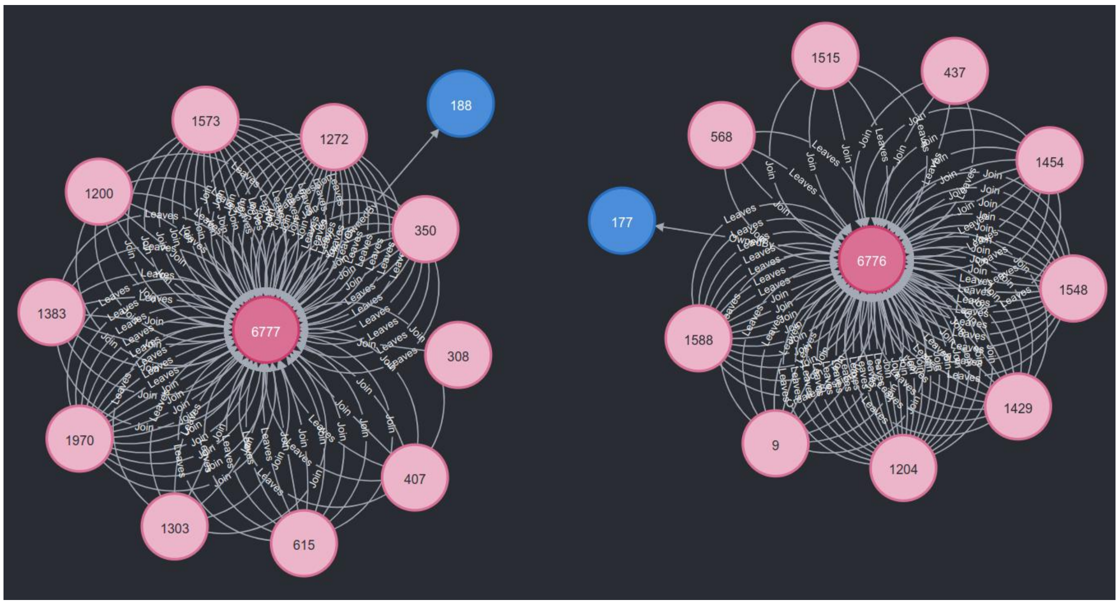


Figure 4.2: Example chat sessions being joined and left by several users.

Node type	Average degree centrality (3 s.f.)	Node count
User	9.29	796
TeamChatSession	0	127

Table 4.1: Average degree centrality of users joining and leaving chat sessions.

Figure 4.3 presents the average number of chat items per team chat session. This shows a single team chat session can have numerous subjects and conversations. Table 4.2 solidifies this point, as there are over 44413 'ChatItem' nodes. The average degree centrality of 'ChatItem' is 1.00 as all of the 'ChatItem' nodes only direct to only one 'TeamChatSession'.

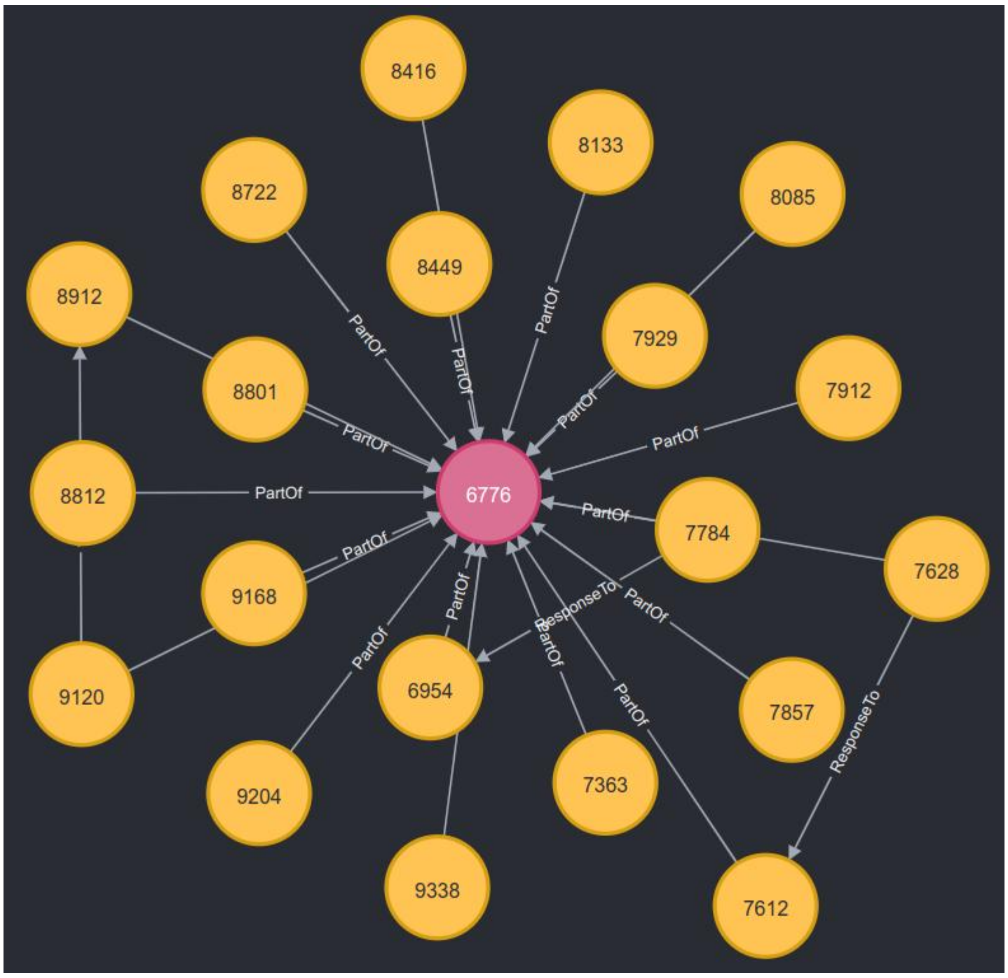


Figure 4.3: Average number of chat items per team chat session.

Node type	Average degree centrality (3 s.f.)	Node count
ChatItem	1.00	44413
TeamChatSession	0	127

Table 4.2: Average degree centrality of the average number of at item timestamps and a chat session.

Figure 4.4 illustrates the relationship between teams and their respective team chat sessions. It is clear that most teams appear to only have one team session chat (which is confirmed by the equal counts in Figure 4.5). This could be due to the teams only having a few team members. For larger teams, they could have multiple team chat sessions.

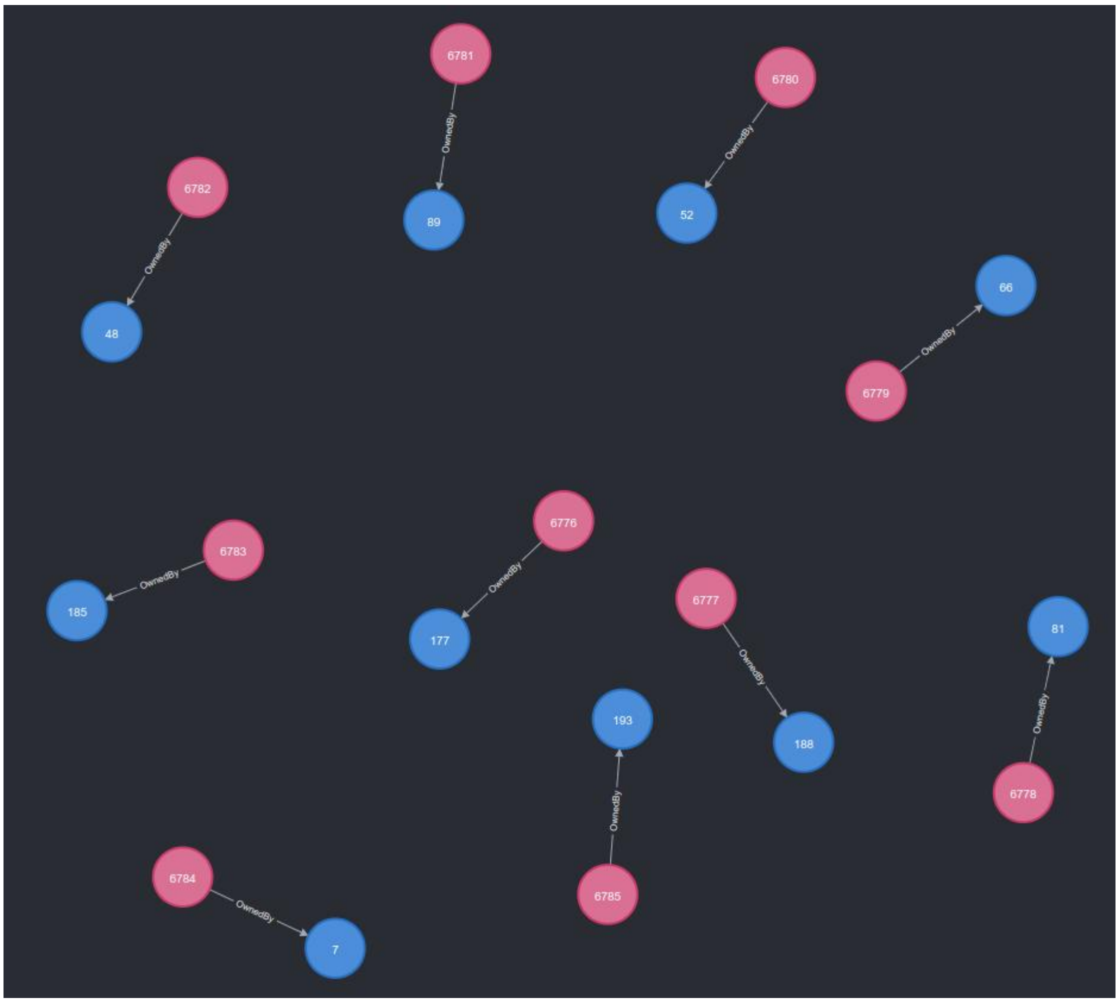
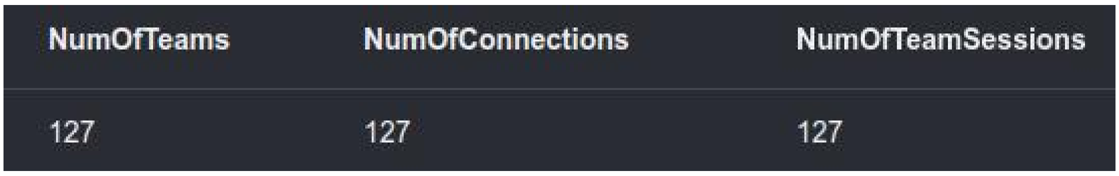


Figure 4.4: The relationship between 'Team' and 'TeamChatSession'.



NumOfTeams	NumOfConnections	NumOfTeamSessions
127	127	127

Figure 4.5: Census of the number of teams, team chat sessions, and their relation.

Figure 4.6 shows the number of chats a user can create. It indicates a number of users actively create conversations and responses to questions. Table 4.3 shows the average degree centrality amongst users and their created chats. This table shows a typical user may generate around 56 conversations and replies during their activity journey. This will likely be higher for users who have played for longer or play more frequently.

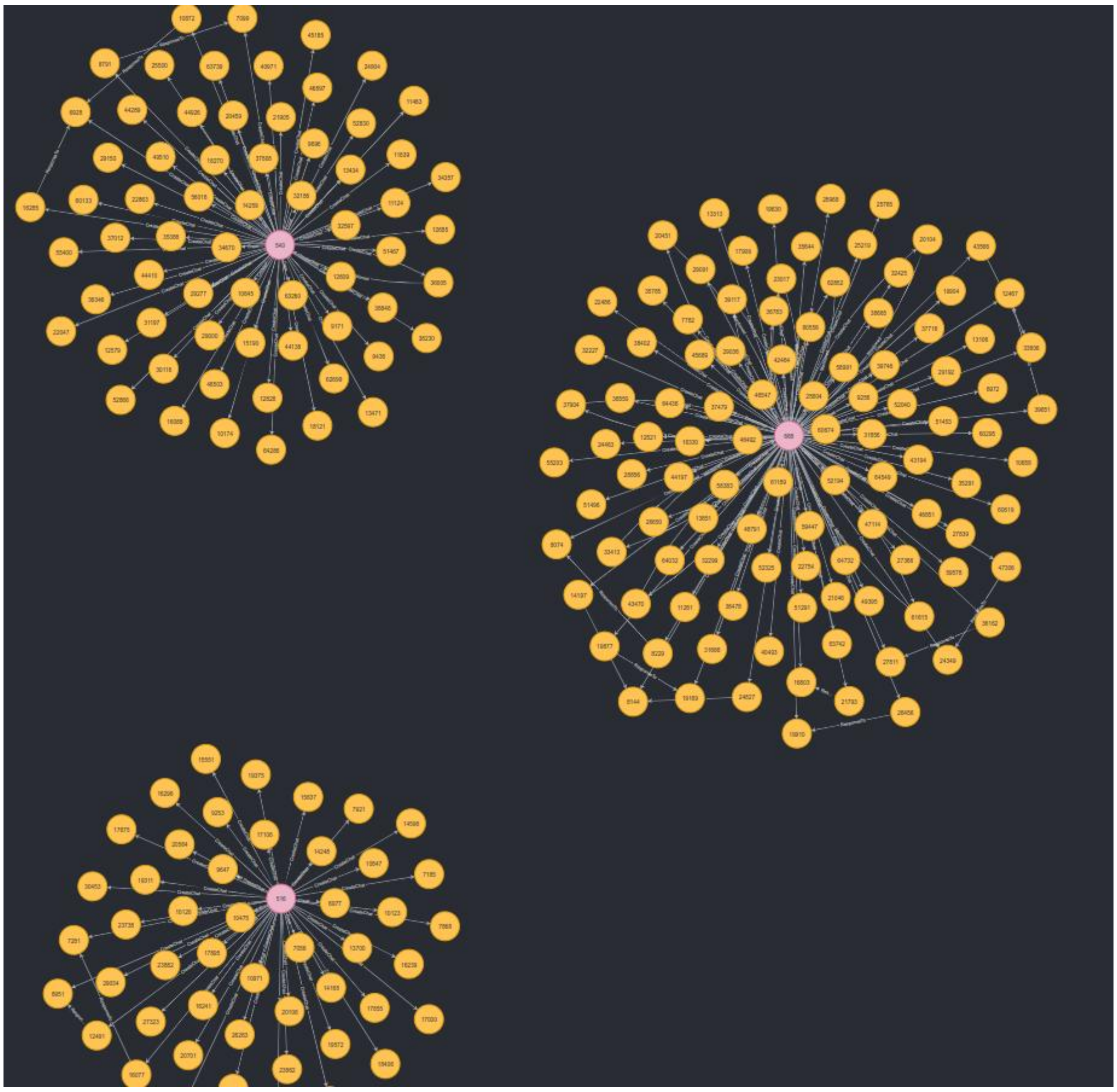


Figure 4.6: The number of 'ChatItem' a user has.

Node type	**Average degree centrality (3 s.f.)**	**Node count**
User	55.80	796
ChatItem	0.0	44413

Table 4.3: Average degree centrality of users creating chats.

5 Conclusions and Future Work

In summary, this report researched, investigated and demonstrated the facilitation and capabilities of Big Data. The potential insights and use cases that can be achieved using Big Data continue to grow, and are being used more and more globally as more data, and more types of data, becomes available. But it is important to always consider the ethical implications of using such data, and the impact it can have on society. Ultimately, the use of Big Data will only continue to increase in the future, and depending on how it is used, will go on to further influence society.

Future work includes investigating scalability strategies for graph-based databases, so that they can dynamically expand with more and more data processing. Additionally, Machine Learning pipelines could be studied to determine better methods to integrate and apply Machine Learning on larger scales, and on more vivid varieties, of data.

Reference List

Al-Aamri, A., Azman, S. K., Elbait, G. D. et al. (2023). Critical assessment of on-premise approaches to scalable genome analysis. In: *BMC Bioinformatics*, 24, pp. 1–27. Available at: 10.1186/s12859-023-05470-2.

Ali, M. M., Paul, B. K., Ahmed, K. et al. (2021). Heart disease prediction using supervised machine learning algorithms: Performance analysis and comparison. In: *Computers in Biology and Medicine*, 136 (1), p. 104672. Available at: 10.1016/j.compbiomed.2021.104672.

Banerjee, S. (2022). *Scalable Data Architecture with Java*. Birmingham, England: Packt Publishing.

Al-Barznji, K. (2022). Big Data Processing Frameworks for Handling Huge Data Efficiencies and Challenges: A Survey. In: *International Journal of Data Science and Big Data Analytics*, 2 (1), pp. 1–9. Available at: 10.51483/IJDSBDA.2.1.2022.1-9.

Bosman, L., Oladepo, T. and Ngambeki, I. (2024). Big data ethics and its role in the innovation and technology adoption process. In: *Journal of Research in Innovative Teaching & Learning*, 17 (1), pp. 66–82. Available at: 10.1108/JRIT-12-2022-0088.

Chen, W. and Quan-Haase, A. (2020). Big Data Ethics and Politics: Toward New Understandings. In: *Social Science Computer Review*, 38 (1), pp. 3–9. Available at: 10.1177/0894439318810734.

Chunduri, R. K. and Cherukuri, A. K. (2021). Big data processing frameworks and architectures: a survey. *Handbook of Big Data Analytics. Volume 1: Methdologies*. Ed. by V. Ravi and A. K. Cherukuri. London: The Institution of Engineering and Technology, pp. 37–104.

Cloudera (2016). *Cloudera Quickstart*. Available at: https://hub.docker.com/r/cloudera/quickstart [Accessed on 20th Apr. 2025].

Docker Inc. (2025). *Docker: Accelerated Container Application Development*. Available at: https://www.docker.com/ [Accessed on 20th Apr. 2025].

Ecker, R., Karagiannis, V., Sober, M. et al. (2025). Latency-aware placement of stream processing operators in modern-day stream processing frameworks. In: *Journal of Parallel and Distributed Computing*, 199, p. 105041. Available at: 10.1016/j.jpdc.2025.105041.

Ezugwu, A. E., Ikotun, A. M., Oyelade, O. O. et al. (2022). A comprehensive survey of clustering algorithms: State-of-the-art machine learning applications, taxonomy, challenges, and future research prospects. In: *Engineering Applications of Artificial Intelligence*, 110. Available at: 10.1016/j.engappai.2022.104743.

Gephi (2025). *The Open Graph Viz Platform*. Available at: https://gephi.org/ [Accessed on 5th Apr. 2025].

Graf, R., Zeldovich, M. and Friedrich, S. (2022). Comparing linear discriminant analysis and supervised learning algorithms for binary classification — A method comparison study. In: *Biometrical Journal*, 66 (1), p. 2200098. Available at: 10.1002/bimj.202200098.

Hu, L., Zhang, F., Qin, M. et al. (2022). A Dynamic Pyramid Tilling Method for Traffic Data Stream Based on Flink. In: *IEEE Transactions on Intelligent Transportation Systems*, 23 (7), pp. 6679–6688. Available at: 10.1109/TITS.2021.3060576.

Jabeen, H. (2020). Big Data Outlook, Tools, and Architectures. *Knowledge Graphs and Big Data Processing*. Ed. by V. Janev, D. Graux, H. Jabeen et al. Cham: Springer, pp. 35–58.

Jacobson, B. (2024). One database to rule them all with Bigtable hybrid transactional and analytical processing (HTAP). In: *Databases*. [blog] 19 July. Available at: https://cloud.google.com/blog/products/databases/hybrid-transactional-and-analytical-processing-on-bigtable/ [Accessed on 5th Apr. 2025].

Khattabi, M. Z. E., Jai, M. E., Lahmadi, Y. et al. (2024). Geometry-Inference Based Clustering Heuristic: New k-means Metric for Gaussian Data and Experimental Proof of Concept. In: *Operations Research Forum*, 5 (13), pp. 1–26. Available at: 10.1007/s43069-024-00291-2.

Kim, J. and Moon, C. (2023). The Distributed HTAP Architecture for Real-Time Analysis and Updating of Point Cloud Data. In: *Electronics*, 12, p. 3959. Available at: 10.3390/electronics12183959.

Kotiranta, P., Junkkari, M. and Nummenmaa, J. (2022). Performance of Graph and Relational Databases in Complex Queries. In: *Applied Sciences*, 12, p. 6490. Available at: 10.3390/app12136490.

Lin, W. (2024). A Design of Hybrid Transactional and Analytical Processing Database for Energy Efficient Big Data Queries. In: *Green, Pervasive, and Cloud Computing*. 22-24 September 2024. Harbin, China: Springer, pp. 128–138. Available at: 10.1007/978-981-99-9893-7_10.

Lipp, B. (2023). *Modern Data Architectures with Python*. Birmingham, England: Packt Publishing.

Liu, X. and Buyya, R. (2020). Resource Management and Scheduling in Distributed Stream Processing Systems: A Taxonomy, Review, and Future Directions. In: *ACM Computing Surveys*, 53, pp. 1–41. Available at: 10.1145/3355399.

Lu, Y., Guo, H., Huang, L. et al. (2021). GraphPEG: Accelerating Graph Processing on GPUs. In: *ACM Transactions on Architecture and Code Optimisation*, 18 (3), pp. 1–24. Available at: 10.1145/3450440.

Matyashovskyy, T. (2016). Lambda Architecture with Apache Spark? In: *Data Engineering*. [blog] 3 November. Available at: https://dzone.com/articles/lambda-architecture-with-apache-spark [Accessed on 4th Apr. 2025].

Mills, K. A. (2019). *Big Data for Qualitative Research.* London, England: Routledge Focus.

Nasseef, M. A. O. (2020). Ethics of Big Data: A Socio-Economic Perspective. In: *Journal of King Abdulaziz University: Islamic Economies*, 33 (1), pp. 91–98.

Neo4j (2025). *Graph Database & Analytics.* Available at: https://neo4j.com/ [Accessed on 5th Apr. 2025].

Novak, R. and Pavlicek, A. (2021). Data Experts as the Balancing Power of Big Data Ethics. In: *Information*, 12 (3), p. 97. Available at: https://doi.org/10.3390/info12030097.

Nudurupati, S. (2021). *Essential PySpark for Scalable Data Analytics.* Birmingham, England: Packt Publishing.

Penka, J. B. N., Mahmoudi, S. and Debauche, O. (2021). A new Kappa Architecture for IoT Data Management in Smart Farming. In: *Procedia Computer Science*, 191, pp. 17–24. Available at: 10.1016/j.procs.2021.07.006.

Penka, J. B. N., Mahmoudi, S. and Debauche, O. (2022). An Optimized Kappa Architecture for IoT Data Management in Smart Farming. In: *Journal of Ubiquitous Systems & Pervasive Networks*, 17 (2), pp. 59–65. Available at: 10.5383/JUSPN.17.02.002.

Ravita, D. (2024). Beginner's Guide to HTAP Database. In: *Data Intensity.* [blog] 15 November. Available at: https://www.singlestore.com/blog/what-is-htap/ [Accessed on 5th Apr. 2025].

Reed-Berendt, R., Dove, E. S. and Pareek, M. (2021). The Ethical Implications of Big Data Research in Public Health: "Big Data Ethics by Design" in the UK-REACH Study. In: *Ethics and Human Research*, 44 (1), pp. 2–17. Available at: 10.1002/eahr.500111.

Ristov, S., Gusev, M., Hohenegger, A. et al. (2023). Severless Electrocardiogram Stream Processing in Federated Clouds With Lambda Architecture. In: *Computer*, 56 (9), pp. 18–27. Available at: 10.1109/MC.2023.3281873.

Rokobo (2024). *Big Data Specialisation.* Available at: https://github.com/rokobo/Big-Data-Specialization [Accessed on 20th Apr. 2025].

Rupp, V. and Grafenstein, M. von (2024). Clarifying "personal data" and the role of anonymisation in data protection law: Including and excluding data from the scope of the GDPR (more clearly) through refining the concept of data protection. In: *Computer Law & Security Review: The International Journal of Technology Law and Practice*, 52 (1), p. 105932. Available at: 10.1016/j.clsr.2023.105932.

Sheikh, A., Kumar, S. and Ambhaikar, A. (2021). IoT Data Analytics Using Cloud Computing. *Big Data Analytics for Internet of Things.* Ed. by T. J. Saleem and M. A. Chishti. Hoboken, New Jersey: Wiley, pp. 115–139.

Song, H., Zhou, W., Cui, H. et al. (2023). A survey on hybrid transactional and analytical processing. In: *The VLDB journal*, 33, pp. 1485–1515. Available at: 10.1007/s00778-024-00858-9.

Suthakar, U., Magnoni, L., Smith, D. R. et al. (2021). Optimised Lambda Architecture for Monitoring Scientific Infrastructure. In: *IEEE Transactions on Parallel and Distributed Systems*, 32 (6), pp. 1395–1408.

Szabo, S., Holb, I. J., Abriha-Molnar, V. E. et al. (2024). Classification Assessment Tool: A program to measure the uncertainty ofn classification models in terms of class-level metrics. In: *Applied Soft Computing*, 155 (1), p. 111468. Available at: 10.1016/j.asoc.2024.111468.

Thorn, J. (2020). Logistic Regression Explained. In: *Artificial Intelligence*. [blog] 8 February. Available at: https://towardsdatascience.com/logistic-regression-explained-9ee73cede081/ [Accessed on 10th May 2025].

Yuan, D., Zhang, L., Dong, J. et al. (2024). Snowy Dove: An open-source toolkit for preprocessing of Chinese Gaofen series data. In: *PLoS One*, 19 (11), pp. 1–15. Available at: https://doi.org/10.1371/journal.pone.0313584.

Appendix A Fixing Cloudera Services

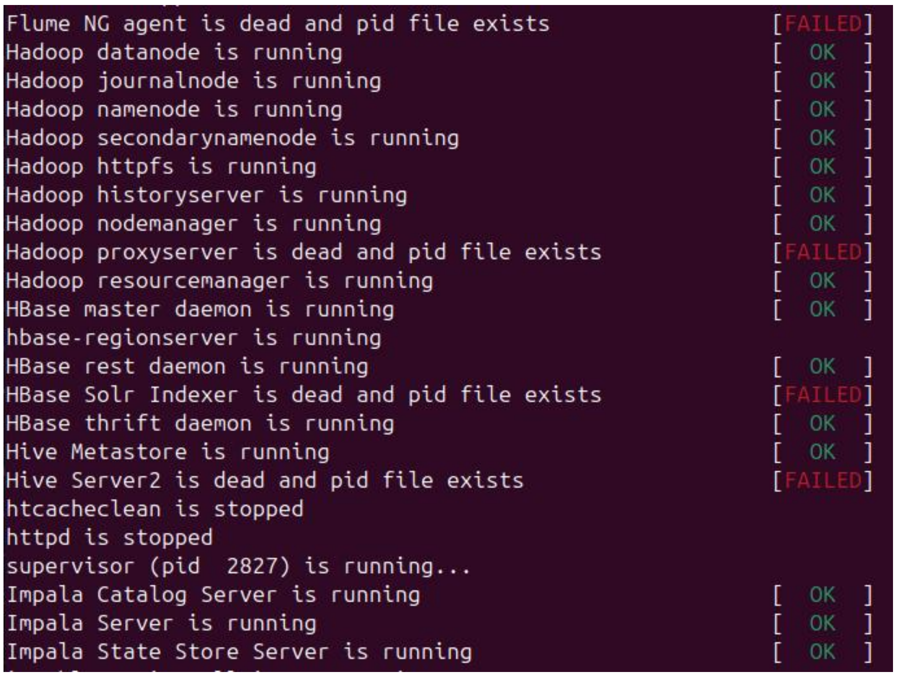
Upon starting the Cloudera Quickstart, not all services may be operating correctly. Some services can be fixed by restarting them, but others unfortunately require adjustments outside the scope of this report. This is shown in Figures A.1, A.2, and A.3. Listing A.1 shows the necessary shell command to start the Cloudera Quickstart container, while Listing A.2 shows the shell command to access the container.

```
docker compose --file cloudera-compose.yaml up
```

Listing A.1: Starting the Cloudera Quickstart container.

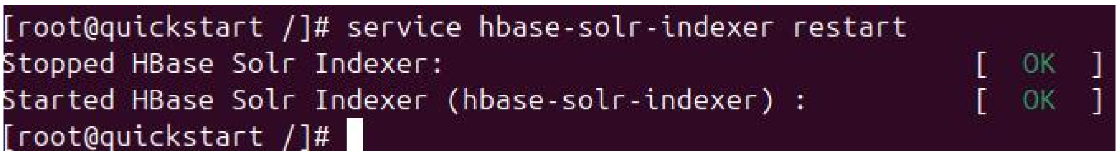
```
docker exec -it cloudera-platform bash
```

Listing A.2: Accessing the Cloudera Quickstart container through the terminal.



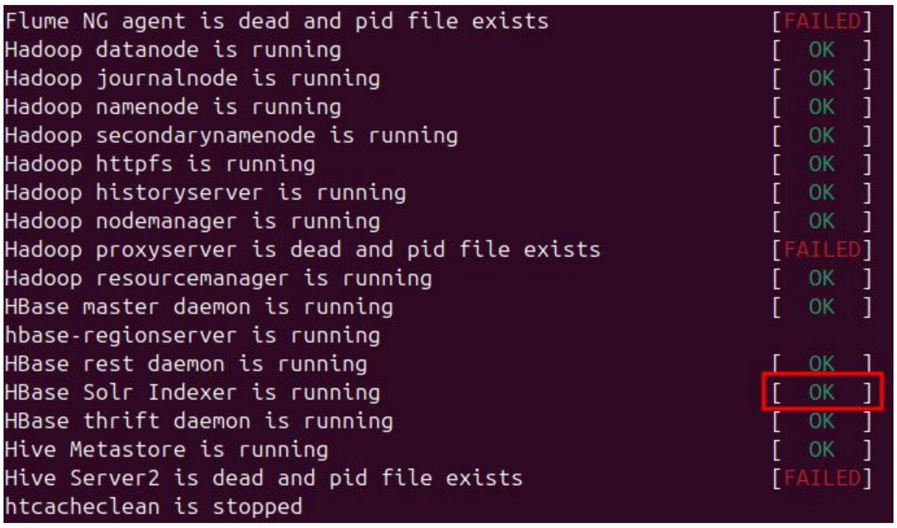
```
Flume NG agent is dead and pid file exists                  [FAILED]
Hadoop datanode is running                                  [  OK  ]
Hadoop journalnode is running                               [  OK  ]
Hadoop namenode is running                                  [  OK  ]
Hadoop secondarynamenode is running                         [  OK  ]
Hadoop httpfs is running                                    [  OK  ]
Hadoop historyserver is running                             [  OK  ]
Hadoop nodemanager is running                               [  OK  ]
Hadoop proxyserver is dead and pid file exists              [FAILED]
Hadoop resourcemanager is running                           [  OK  ]
HBase master daemon is running                              [  OK  ]
hbase-regionserver is running
HBase rest daemon is running                                [  OK  ]
HBase Solr Indexer is dead and pid file exists              [FAILED]
HBase thrift daemon is running                              [  OK  ]
Hive Metastore is running                                   [  OK  ]
Hive Server2 is dead and pid file exists                    [FAILED]
htcacheclean is stopped
httpd is stopped
supervisor (pid  2827) is running...
Impala Catalog Server is running                            [  OK  ]
Impala Server is running                                    [  OK  ]
Impala State Store Server is running                        [  OK  ]
```

Figure A.1: Some Cloudera services that failed upon start-up.



```
[root@quickstart /]# service hbase-solr-indexer restart
Stopped HBase Solr Indexer:                                [  OK  ]
Started HBase Solr Indexer (hbase-solr-indexer) :          [  OK  ]
[root@quickstart /]#
```

Figure A.2: A shell command to restart one of the failed services.

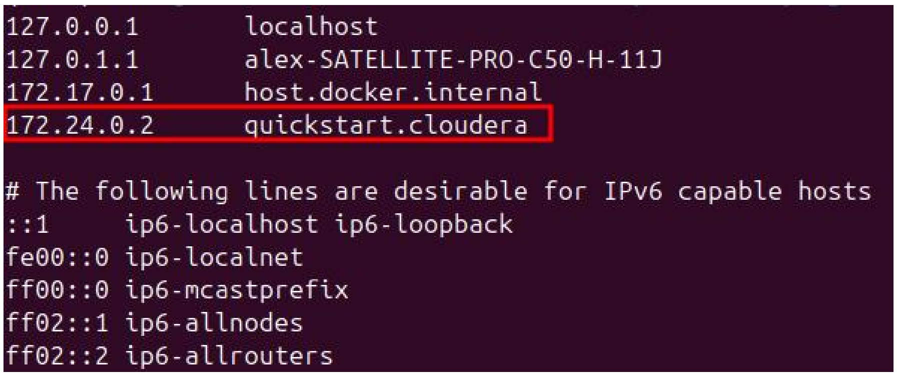


```
Flume NG agent is dead and pid file exists                 [FAILED]
Hadoop datanode is running                                 [  OK  ]
Hadoop journalnode is running                              [  OK  ]
Hadoop namenode is running                                 [  OK  ]
Hadoop secondarynamenode is running                        [  OK  ]
Hadoop httpfs is running                                   [  OK  ]
Hadoop historyserver is running                            [  OK  ]
Hadoop nodemanager is running                              [  OK  ]
Hadoop proxyserver is dead and pid file exists             [FAILED]
Hadoop resourcemanager is running                          [  OK  ]
HBase master daemon is running                             [  OK  ]
hbase-regionserver is running
HBase rest daemon is running                               [  OK  ]
HBase Solr Indexer is running                              [  OK  ]
HBase thrift daemon is running                             [  OK  ]
Hive Metastore is running                                  [  OK  ]
Hive Server2 is dead and pid file exists                   [FAILED]
htcacheclean is stopped
```

Figure A.3: HBase Solr Indexer successfully restarted and working.

A.1 Updating Recognised Hosts

For the Cloudera Quickstart container to operate properly, it may be necessary to update the host computer's associated Internet Protocol version 4 (IPv4). The red box in Figure A.4 highlights the possible service that may need to be added.



```
127.0.0.1       localhost
127.0.1.1       alex-SATELLITE-PRO-C50-H-11J
172.17.0.1      host.docker.internal
172.24.0.2      quickstart.cloudera

# The following lines are desirable for IPv6 capable hosts
::1     ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
```

Figure A.4: Updating recognised name services to include Cloudera Quickstart.

Appendix B Dataset

B.1 File Structure

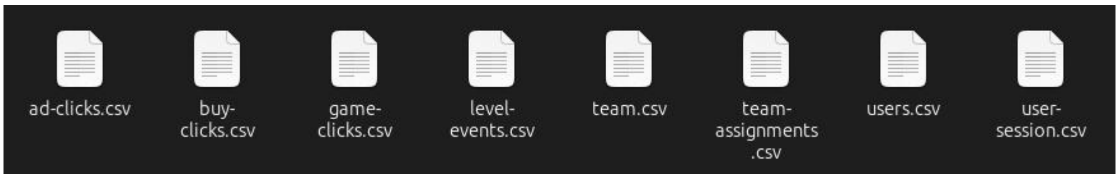


Figure B.1: In-game file data.

B.2 Statistical Types

Table B.1: Group 1 data files statistical types.

Column name	**Description**	**Statistical Type**
teamCreationTime	The time and date the team was created.	Interval
teamId	A unique ID for each team.	Nominal
isHit	Denotes if the click was on a flamingo (1) or not (0).	Nominal
assignmentId	A unique ID for the current assignment for the team.	Nominal
dob	The date of birth of the user.	Interval
userSessionId	A unique ID for each user session.	Nominal
team	The current team ID of the user who made the purchase.	Nominal
nick	The nickname decided by the user.	Nominal
buyId	The unique ID of the item purchased.	Nominal
teamLevel	The level started or completed.	Ordinal
adCategory	The category type of the selected ad.	Nominal
price	The price of the item purchased.	Ratio
teamEndTime	The timestamp when the last member of the team left.	Interval

Continuation of Table B.1		
currentLevel	The current level of the team.	Ordinal
adId	A unique ID for each ad.	Nominal
timestamp (for users)	When the user first played the game.	Interval
timestamp (for purchases)	When the purchase was made.	Interval
timestamp (for teams)	When the user joined the team.	Interval
timestamp (for user session)	When the event occured.	Interval
twitter	The Twitter handle of the user.	Nominal
clickId	A unique ID for the game click.	Nominal
strength	A measure of the team strength.	Ratio
platformType	The type of platform the user used during the session.	Nominal
txId	A unique ID for the in-app purchase.	Nominal
country	The two-letter country code for where the user lives.	Nominal
name	The name of the team.	Nominal
eventId	A unique ID for each event.	Nominal
eventType	The type of event, either start or end.	Nominal

Continuation of Table B.1

sessionType	Whether the event is the start or end of a session.	Nominal
userId	A unique ID for each user.	Nominal

End of Table

Table B.2: Group 1 Flamingo Game file overview.

File name	**Description**
game-clicks.csv	Details some information when a user clicks in the game.
users.csv	Contains information about the registered users.
team.csv	Contains information about registered teams.
team-assignment.csv	Records when a user joins a team.
ad-clicks.csv	Details recorded when a player clicks an ad.
buy-clicks.csv	Records when a player makes an in-game purchase.
level-events.csv	Records details when a team starts or finishes a team.
user-session.csv	Details when a user starts and stops in a game. When a team goes to the next level, all the team user sessions end and new ones start.

Appendix C Code

C.1 Docker Compose: Cloudera Quickstart

```
services:
  cloudera-platform:
    image: cloudera/quickstart:latest
    container_name: cloudera-platform
    ports:
      - 80:80
      - 7180:7180
      - 8888:8888
      - 8020:8020
      - 8032:8032
      - 16000:16000
      - 60000:60000
      - 60010:60010
      - 8090:8090
      - 9090:9090
      - 50070:50070
      - 50010:50010
    hostname: quickstart.cloudera
    privileged: true
    tty: true
    stdin_open: true
    command: /usr/bin/docker-quickstart

# HDFS namenode is port 8020
# YARN resource manager is 8032, and 8042 for node manager
# hbase is 16000
# Hue is 8888
# use ls /etc/init.d to view runnable services
# service hbase-solr-indexer restart
# service flume-ng-agent restart
# service hadoop-mapreduce-historyserver restart
# service hbase-master restart
# service hbase-thrift restart
```

Listing C.1: Docker Compose startup code for Cloudera Quickstart.

C.2 Environment Setup

C.2.1 Uploading the files to HDFS

```
from hdfs import InsecureClient
import os
```

```python
client = InsecureClient('http://localhost:50070', user='hadoop')

client.upload('/user/clouser', '/home/alex/Documents/BigDataAnalytics/flamingo-data', overwrite = True)
```

Listing C.2: Python code to upload files to HDFS.

C.2.2 Copying file contents to HBase

```python
import happybase
import pandas as pd

def upload_to_hbase(path_to_files:str) -> None:
    """
    Upload files of the Flamingo Game data to HBase.

    Parameters:
        - path_to_files (string): The absolute path to the files stored on the current d

    Example:
    >>> upload_to_hbase('/home/alex/Documents/BigDataAnalytics/flamingo-data')
    """
    file_names = os.listdir(path_to_files)

    # port 9090 is the default port for HBase Thrift server
    conn = happybase.Connection('localhost', port = 9090, timeout = 100000)

    for file in file_names:
        path = path_to_files + '/' + file
        file_name = file.replace('.csv', '')
        df = pd.read_csv(path)

        # check if the column family already exists
        if file_name.encode() in conn.tables():
            continue

        # record how many columns there are
        col_count = len(df.columns)

        # create a new table in HBase with the name corresponding to file_name, and an
        # equal number of column families to the number of columns of the dataframe
        conn.create_table(
            file_name,
            {f'cf{i+1}':dict() for i in range(col_count)}
        )

        table = conn.table(file_name)

        # upload the data, line-by-line
        for index, row in df.iterrows():
            data = {}
            i = 1
            for col in df.columns:
                key = 'cf' + str(i) + ':' + col
                value = str(row[col])
                data[key] = value
                i+=1
            row_key = str(index)
            table.put(row_key, data)

            print("Table {} completed, along with successful data upload.".format(file_n

    conn.close()

upload_to_hbase('/home/alex/Documents/BigDataAnalytics/flamingo-data')
```

Listing C.3: Python code to upload files to HBase.

C.2.3 Downloading the data from HBase

```python
from pyspark.sql import SparkSession
import happybase
import pandas as pd

# create a Spark Session
spark = SparkSession.builder \
    .appName("BigDataAnalysis") \
    .getOrCreate()

def retrieve_hbase_data(spark: SparkSession) -> Dict[str, DataFrame]:
    """
    Retrieve the stored flamingo data from the HBase storage.

    Parameter:
        - spark (SparkSession): The current SparkSession.

    Returns:
        - Dict[str, DataFrame]: a dictionary with key values corresponding to the origin
                                and the respective values being Spark DataFrames.
    """
    conn = happybase.Connection('localhost', port = 9090, timeout = 100000)

    # a dictionary to store all of the files in separate dataframes
    # key : file name, value : dataframe of the file data
    df_dict = {}

    for table_name in conn.tables():
        table = conn.table(table_name.decode('utf-8'))

        # list temporarily holds the table data
        data_list = []
        for key, data in table.scan():
            row_dict = {'row_key': key.decode('utf-8')}
            for column, value in data.items():
                row_dict[column.decode('utf-8')] = value.decode('utf-8')
            data_list.append(row_dict)

        # convert the data into a Pandas dataframe temporarily
        df = pd.DataFrame(data_list)
        df = df.drop(columns=['row_key'])

        print(f"Data successfully downloaded for {table_name.decode('utf-8')}.")

        # attempt to convert columns from string (which is the default) into a numbers type or datetime
        # if both attempts fail, it will be ignored (as it should probably be a string)
        for col in df.columns:
            try:
                df[col] = pd.to_numeric(df[col])
            except ValueError:
                try:
                    df[col] = pd.to_datetime(df[col])
                except ValueError:
                    pass

        # remove some semantic wording in the column names that was necessary for HBase
        # e.g., 'cf1:timestamp' becomes 'timestamp'
        df.columns = df.columns.str.replace(r'cf\d+:', '', regex=True)

        # add the dataframe to the dictionary, and convert it into a
        ↪  Spark dataframe
        df_dict[table_name.decode('utf-8')] = spark.createDataFrame(df)

    return df_dict

df_dict = retrieve_hbase_data(spark)
```

Listing C.4: Python code to download data from HBase.

C.3 Neo4j

The following code segments should be inputted into the Neo4j terminal separately.

C.3.1 Downloading the data into Neo4j

```
// create constraints
CREATE CONSTRAINT user_id_unique IF NOT EXISTS FOR (u:User) REQUIRE
↪  u.id IS UNIQUE;
CREATE CONSTRAINT team_id_unique IF NOT EXISTS FOR (t:Team) REQUIRE
↪  t.id IS UNIQUE;
CREATE CONSTRAINT team_session_id_unique IF NOT EXISTS FOR
↪  (ts:TeamChatSession) REQUIRE ts.id IS UNIQUE;
CREATE CONSTRAINT chat_item_id_unique IF NOT EXISTS FOR (ct:ChatItem)
↪  REQUIRE ct.id IS UNIQUE;

// create team chat
LOAD CSV FROM "http://localhost:50070/webhdfs/v1/user
/cloudera/chat-data/chat_create_team_chat.csv?op=OPEN" AS row
MERGE (u:User {id: toInteger(row[0])})
MERGE (t:Team {id: toInteger(row[1])})
MERGE (c:TeamChatSession {id: toInteger(row[2])})
MERGE (u)-[:CreatesSession{timeStamp: row[3]}]->(c)
MERGE (c)-[:OwnedBy{timeStamp: row[3]}]->(t)


// join chat dataset
LOAD CSV FROM "http://localhost:50070/webhdfs/v1/user
/cloudera/chat-data/chat_join_team_chat.csv?op=OPEN" AS row
MERGE (u:User {id: toInteger(row[0])})
MERGE (c:TeamChatSession {id: toInteger(row[1])})
MERGE (u)-[:Join{timeStamp: row[2]}]->(c)
```

```

// leave chat dataset
LOAD CSV FROM "http://localhost:50070/webhdfs/v1/user
/cloudera/chat-data/chat_leave_team_chat.csv?op=OPEN" AS row
MERGE (u:User {id: toInteger(row[0])})
MERGE (c:TeamChatSession {id: toInteger(row[1])})
MERGE (u)-[:Leaves{timeStamp: row[2]}]->(c)


// chat item dataset
LOAD CSV FROM "http://localhost:50070/webhdfs/v1/user
/cloudera/chat-data/chat_item_team_chat.csv?op=OPEN" AS row
MERGE (u:User {id: toInteger(row[0])})
MERGE (c:TeamChatSession {id: toInteger(row[1])})
MERGE (i:ChatItem {id: toInteger(row[2])})
MERGE (u)-[:CreateChat{timeStamp: row[3]}]->(i)
MERGE (i)-[:PartOf{timeStamp: row[3]}]->(c)

// mentions dataset
LOAD CSV FROM "http://localhost:50070/webhdfs/v1/user
/cloudera/chat-data/chat_mention_team_chat.csv?op=OPEN" AS row
MERGE (i:ChatItem {id: toInteger(row[0])})
MERGE (u:User {id: toInteger(row[1])})
MERGE (i)-[:Mentioned{timeStamp: row[2]}]->(u)

// response dataset
LOAD CSV FROM "http://localhost:50070/webhdfs/v1/user
/cloudera/chat-data/chat_respond_team_chat.csv?op=OPEN" AS row
MERGE (i:ChatItem {id: toInteger(row[0])})
MERGE (j:ChatItem {id: toInteger(row[1])})
MERGE (i)-[:ResponseTo{timeStamp: row[2]}]->(j)
```

Listing C.5: Cypher code to download the data from HDFS, then upload the data to Neo4j.

C.3.2 Visualising the database schema.

```
CALL db.schema.visualization()
```

Listing C.6: Cypher code to visualise the database schema.

C.3.3 Exploratory Data Analysis

```
MATCH (u:User) - [:Join] - (tc:TeamChatSession) - [:OwnedBy] - (t:Team)
↪  RETURN tc, collect(u), t LIMIT 2

// create a graph projection (using Neo4j Graph Data Science Library)
CALL gds.graph.project(
  'userTeamChatSession',
  ['User', 'TeamChatSession'],
  ['Join', 'Leaves', 'CreatesSession']
);

// Calculate degree centrality
CALL gds.degree.stream('userTeamChatSession')
YIELD nodeId, score
WITH gds.util.asNode(nodeId) AS node, score
RETURN labels(node) AS nodeType,
       avg(score) AS avgDegreeCentrality,
       count(node) AS nodeCount
ORDER BY avgDegreeCentrality DESC;
```

Listing C.7: Cypher code to illustrate the user-team chat session relation.

```
MATCH (ct:ChatItem) - [:PartOf] -> (tcs:TeamChatSession) RETURN ct, tcs
↪  LIMIT 20

CALL gds.graph.project(
  'itemSessionDegree',
  ['ChatItem', 'TeamChatSession'],
  'PartOf'
);

CALL gds.degree.stream('itemSessionDegree')
YIELD nodeId, score
WITH gds.util.asNode(nodeId) AS node, score
RETURN labels(node) AS nodeType,
       avg(score) AS avgDegreeCentrality,
       count(node) AS nodeCount
ORDER BY avgDegreeCentrality DESC;
```

Listing C.8: Cypher code to show the chat item and team chat session relation.

```
MATCH path = (:TeamChatSession) - [:OwnedBy] -> (:Team) RETURN path
↪  LIMIT 10

```

```
MATCH (tcs:TeamChatSession) - [ob:OwnedBy] -> (t:Team) RETURN count(t)
↪  AS NumOfTeams, count(ob) AS NumOfConnections, count(tcs) AS
↪  NumOfTeamSessions
```

Listing C.9: Cypher code to show the relation between the team chat sessions and teams.

```
MATCH (u:User) - [:CreateChat] -> (ci:ChatItem) RETURN u, ci LIMIT 200

CALL gds.graph.project(
  'userChatItem',
  ['User', 'ChatItem'],
  'CreateChat'
);

CALL gds.degree.stream('userChatItem')
YIELD nodeId, score
WITH gds.util.asNode(nodeId) AS node, score
RETURN labels(node) AS nodeType,
       avg(score) AS avgDegreeCentrality,
       count(node) AS nodeCount
ORDER BY avgDegreeCentrality DESC;
```

Listing C.10: Cypher code to show the relation between the users and chat items.