

Student Name(s) and Number(s): Himanshu Dubey (20027763), Aakash Nikam (20054691)

Programme: MSc in Data Analytics

Lecturer Name: Dr Luciana Nascimento

Module/Subject Title: DATA STORAGE SOLUTIONS FOR DATA ANALYTICS(B9DA111)

Assignment Title: London Bike Share Analytics: Designing a Data Warehouse and Comparing Relational and Graph Databases

No of Words: 4706

By submitting this assignment, I am/ we are confirming that:

● This assignment is all my/our own work;

● Any sources used have been referenced;

● I/we have followed the Generative AI instructions/ scale set out in the Assignment Brief;

● I/we have read the College rules regarding academic integrity in the QAH Part B Section 3, and the Generative AI Guidelines, and understand that penalties will be applied accordingly if work is found not to be my/our own.

● I/we understand that all work uploaded is submitted via Ouriginal, whereby a text-matching report will show any similarities with other texts.

**Content**

1. Introduction.............................................................................................................2

1.1. Overview of Bike-Sharing and Dataset........................................................2

1.2. Reasons for Selecting the Subject Area and Data........................................3

1.3. Vision and Goals..........................................................................................4

1.4. Key Stakeholders..........................................................................................4

1.5. Business Requirements ...............................................................................5

1.6. Additional Considerations............................................................................5

1. Schema......................................................................................................................5
2. ETL............................................................................................................................11
3. Visualizations and Reports........................................................................................14

4.1. Visualizations.........................................................................................14

4.2. Reports....................................................................................................18

1. Graph Databases........................................................................................................21

5.1. Comparison to Relational Databases.......................................................23

1. Conclusions................................................................................................................25
2. Bibliography...............................................................................................................26

Appendix A – Visualizations Code....................................................................................26

Appendix B – Neo4j Code................................................................................................27

**1. Introduction**

The rapid growth of urban populations and the increasing emphasis on sustainable transportation have made bike-sharing systems a critical component of modern city infrastructure. The London Bike Share Usage dataset, sourced from Kaggle (Kalacheva, 2025), provides a comprehensive record of bike-sharing activities in London, capturing details such as trip durations, start and end stations, timestamps, and other relevant attributes. This dataset serves as the foundation for developing a data storage and analytics solution to support the operational and strategic needs of a bike-sharing system. By leveraging this dataset, this project aims to design a robust data warehouse, implement ETL processes, create insightful visualizations, and compare relational and graph database performance to optimize data storage and retrieval for analytics purposes.

The dataset, covering bike-sharing usage in London, is particularly valuable due to its rich temporal and spatial data, which enables detailed analysis of user behavior, station performance, and system efficiency. This report outlines the development of a data warehouse to store and process this data, the creation of reports and visualizations using SQL Server Reporting Services (SSRS) and Tableau, and a comparative analysis of relational and graph databases using SQL and Cypher Query Language (CQL) in Neo4j. The insights derived from this project will support stakeholders in optimizing bike-sharing operations, improving resource allocation, and enhancing user experience.

**1.2. Reasons for Selecting the Subject Area and Data**

The subject area of bike-sharing systems was chosen due to its relevance to urban mobility, sustainability, and data-driven decision-making. Bike-sharing programs, such as the one in London, generate large volumes of data that can reveal patterns in usage, peak demand periods, and station connectivity, making it an ideal candidate for data analytics. The London Bike Share Usage dataset was selected for the following reasons:

* **Rich and Diverse Data**: The dataset includes attributes such as trip duration, start and end station details, timestamps, and bike identifiers, enabling comprehensive analysis of usage patterns and operational efficiency.
* **Real-World Relevance**: Bike-sharing systems are integral to urban planning and sustainability efforts, making the dataset a practical choice for deriving actionable insights.
* **Suitability for Multiple Analyses**: The dataset supports various analytical approaches, including time-series analysis, geospatial analysis, and network analysis, aligning with the assignment’s requirements for visualizations, reports, and database comparisons.
* **Public Availability**: Sourced from Kaggle, the dataset is well-documented and accessible, ensuring transparency and ease of use for this project (Kalacheva, 2025).

**1.3. Vision and Goals**

The vision of this project is to develop a data storage and analytics solution that enhances the operational efficiency and strategic planning of the London bike-sharing system. The primary goals are:

* **Data Warehouse Development**: Design and implement a data warehouse to centralize and organize bike-sharing data for efficient querying and reporting.
* **Insightful Visualizations and Reports**: Create four SSRS reports and four Tableau visualizations to address key business requirements, such as identifying peak usage times, popular stations, and trip patterns.
* **Database Comparison**: Implement a graph database in Neo4j alongside a relational database (e.g., AdventureWorks or a custom relational database) to compare their performance in storing and retrieving bike-sharing data.
* **Stakeholder Support**: Provide actionable insights to stakeholders, including city planners, bike-sharing operators, and data analysts, to optimize station placement, bike availability, and user satisfaction.

**1.4. Key Stakeholders**

The key stakeholders for this project include:

* **Bike-Sharing Operators**: Responsible for managing bike availability, station maintenance, and system operations. They require insights into usage patterns and station performance to optimize resource allocation.
* **City Planners**: Interested in understanding bike-sharing trends to improve urban mobility and infrastructure planning.
* **Data Analysts**: Tasked with analyzing the dataset to generate reports and visualizations that inform decision-making.
* **End Users**: Cyclists who benefit from improved bike availability and station accessibility based on data-driven optimizations.

**1.5. Business Requirements**

The business requirements for this project are derived from the needs of the stakeholders and the dataset’s potential:

1. **Usage Pattern Analysis**: Identify peak usage times and seasonal trends to optimize bike availability and staffing schedules.
2. **Station Performance**: Analyze the most and least utilized stations to inform station placement and maintenance strategies.
3. **Trip Behavior Insights**: Understand trip durations and routes to improve user experience and system efficiency.
4. **Network Analysis**: Examine relationships between stations (e.g., common start-end station pairs) to optimize bike rebalancing using graph database techniques.

**1.6. Additional Considerations**

To ensure the project aligns with real-world needs, additional considerations include data quality checks during ETL processes, scalability of the data warehouse to handle growing data volumes, and ethical use of generative AI tools as per the assignment’s guidelines The project also adheres to Dublin Business School’s academic integrity policies, with all sources referenced in Harvard style and AI usage documented in an appendix.

**2. Schema**

The schema for this project comprises two components: a relational database schema to store the raw London Bike Share Usage data and a data warehouse schema designed for efficient analytics. Both schemas are tailored to the dataset’s structure, which includes journey details, station information, and bike attributes, to support the business requirements outlined in Section 1.5, such as usage pattern analysis, station performance, and network analysis. The schemas are implemented in SQL Server, with the relational schema serving as the source for ETL processes and the data warehouse schema enabling optimized querying and reporting.

**Relational Database Schema**

The relational database schema consists of three tables to capture the core entities of the bike-sharing system:

1. **Stations Table**
   * **Attributes**:
     + StationID (INT, Primary Key): Unique identifier for each bike station.
     + StationName (VARCHAR(100), NOT NULL, UNIQUE): Name of the station (e.g., “Waterloo Station”).
     + Location (VARCHAR(100)): Geographic location of the station (e.g., address or coordinates).
   * **Purpose**: Stores station details, with a unique constraint on StationName to prevent duplicate entries. The Location field supports geospatial analysis, such as mapping station distributions.
2. **Bikes Table**
   * **Attributes**:
     + BikeID (INT, Primary Key): Unique identifier for each bike.
     + BikeModel (VARCHAR(50), NOT NULL, CHECK constraint): Type of bike, restricted to ‘CLASSIC’ or ‘PBSC\_EBIKE’.
     + Status (VARCHAR(20), DEFAULT ‘Available’, CHECK constraint): Current status of the bike, limited to ‘Available’, ‘In Use’, or ‘Maintenance’.
   * **Purpose**: Tracks bike inventory and status, enabling analysis of bike utilization and maintenance needs. The CHECK constraints ensure data integrity by restricting values to valid options.
3. **Journeys Table**
   * **Attributes**:
     + JourneyID (INT, Primary Key): Unique identifier for each trip.
     + StartDateTime (DATETIME, NOT NULL): Timestamp when the journey began.
     + EndDateTime (DATETIME, NOT NULL): Timestamp when the journey ended.
     + StartStationID (INT, NOT NULL, Foreign Key): References the starting station.
     + EndStationID (INT, NOT NULL, Foreign Key): References the ending station.
     + BikeID (INT, NOT NULL, Foreign Key): References the bike used.
     + TotalDurationMS (BIGINT, NOT NULL): Trip duration in milliseconds.
     + Constraints: Foreign keys link to Stations and Bikes tables; CHECK constraints ensure EndDateTime is not earlier than StartDateTime and TotalDurationMS is non-negative.
     + Indexes: IDX\_Journey\_StartDateTime on StartDateTime and IDX\_Journey\_BikeID on BikeID for faster querying of temporal and bike-specific data.
   * **Purpose**: Captures trip details, enabling analysis of journey patterns, durations, and station connectivity.

**Data Warehouse Schema**

The data warehouse schema follows a **star schema** design to optimize analytical queries and support the creation of SSRS reports and Tableau visualizations. It consists of one fact table and three dimension tables:

1. **DateDim Table**
   * **Attributes**:
     + DateKey (INT, Primary Key): Unique identifier for each date-time record.
     + FullDate (DATE, NOT NULL): Date of the journey.
     + Year (INT, NOT NULL), Month (INT, NOT NULL), Day (INT, NOT NULL): Temporal components for aggregation.
     + DayOfWeek (VARCHAR(10), NOT NULL): Day of the week (e.g., ‘Monday’).
     + Hour (INT, NOT NULL): Hour of the day (0–23).
     + IsWeekend (BIT, NOT NULL): Indicates if the date is a weekend (1 for true, 0 for false).
   * **Purpose**: Enables temporal analysis at various granularities (e.g., hourly, daily, monthly) and supports queries for peak usage times and weekend vs. weekday patterns.
2. **StationDim Table**
   * **Attributes**:
     + StationKey (INT, Primary Key, IDENTITY(1,1)): Surrogate key for the data warehouse.
     + StationID (INT, NOT NULL, UNIQUE): Maps to the relational StationID.
     + StationName (VARCHAR(100), NOT NULL): Station name.
     + Location (VARCHAR(100)): Station location.
   * **Purpose**: Stores station details for analytical queries, with a surrogate key to optimize joins and a unique constraint on StationID to maintain data integrity.
3. **BikeDim Table**
   * **Attributes**:
     + BikeKey (INT, Primary Key, IDENTITY(1,1)): Surrogate key for the data warehouse.
     + BikeID (INT, NOT NULL, UNIQUE): Maps to the relational BikeID.
     + BikeModel (VARCHAR(50), NOT NULL): Bike type.
   * **Purpose**: Tracks bike attributes for analysis, such as usage patterns by bike model. The surrogate key improves query performance.
4. **JourneyFacts Table**
   * **Attributes**:
     + JourneyFactID (BIGINT, Primary Key, IDENTITY(1,1)): Unique identifier for each fact record.
     + StartDateKey (INT, NOT NULL, Foreign Key): References DateDim for the journey’s start time.
     + EndDateKey (INT, NOT NULL, Foreign Key): References DateDim for the journey’s end time.
     + StartStationKey (INT, NOT NULL, Foreign Key): References StationDim for the starting station.
     + EndStationKey (INT, NOT NULL, Foreign Key): References StationDim for the ending station.
     + BikeKey (INT, NOT NULL, Foreign Key): References BikeDim for the bike used.
     + JourneyCount (INT, NOT NULL, DEFAULT 1): Counts journeys for aggregation (defaulted to 1 per journey).
     + TotalDurationMS (BIGINT, NOT NULL): Total duration of the journey in milliseconds.
     + Indexes: IDX\_JourneyFacts\_StartDateKey on StartDateKey and IDX\_JourneyFacts\_StartStationKey on StartStationKey for optimized query performance.
   * **Purpose**: Central fact table that aggregates journey metrics, enabling queries for usage patterns, station performance, and trip durations.

**Reasons for Schema Design**

The relational and data warehouse schemas were designed to meet the project’s analytical and operational requirements while ensuring performance, scalability, and data integrity:

* **Relational Schema Simplicity**: The Stations, Bikes, and Journeys tables capture the core entities of the bike-sharing system in a normalized structure, reducing redundancy and ensuring data consistency. Foreign key constraints maintain referential integrity, while CHECK constraints enforce valid values for BikeModel, Status, and temporal attributes. Indexes on Journeys (StartDateTime and BikeID) optimize queries for time-based and bike-specific analyses, critical for the dataset’s temporal and operational focus.
* **Star Schema for Analytics**: The data warehouse uses a star schema to optimize analytical queries. The JourneyFacts table serves as the central fact table, linking to DateDim, StationDim, and BikeDim via foreign keys. This structure simplifies joins and supports efficient aggregations, such as calculating average trip durations by station or hour. The use of surrogate keys (StationKey, BikeKey) enhances performance by reducing the size of join columns, and the DateDim table’s granular attributes enable flexible temporal analysis.
* **Support for Business Requirements**: The schemas address the business requirements outlined in Section 1.5:
  + **Usage Pattern Analysis**: The DateDim table’s attributes (Year, Month, Hour, IsWeekend) support queries for peak usage times and seasonal trends.
  + **Station Performance**: The StationDim and JourneyFacts tables enable analysis of station utilization through metrics like journey counts and total durations.
  + **Trip Behavior Insights**: The TotalDurationMS attribute in JourneyFacts supports analysis of trip durations, while StartStationKey and EndStationKey enable route analysis.
  + **Network Analysis**: The relational schema’s Journeys table provides data for graph database implementation in Neo4j, where stations can be nodes and journeys can be relationships, facilitating network analysis of station connectivity.
* **Performance Optimization**: Indexes on frequently queried columns (StartDateTime, BikeID, StartDateKey, StartStationKey) reduce query execution time, particularly for large datasets. The star schema’s denormalized dimension tables minimize join complexity, enhancing performance for SSRS reports and Tableau visualizations.
* **Scalability and Extensibility**: The star schema supports future additions, such as weather or user demographic dimensions, without altering the core structure. The relational schema’s normalized design ensures it can handle growing data volumes while maintaining integrity.
* **Graph Database Compatibility**: The Journeys table’s structure (linking StartStationID, EndStationID, and BikeID) is well-suited for conversion to a graph database in Neo4j, where stations can be modeled as nodes and journeys as directed edges, enabling efficient network queries for the database comparison task.

The schemas collectively provide a robust foundation for ETL processes, analytical queries, and visualizations, aligning with the assignment’s requirements for data warehouse development, SSRS reports, Tableau dashboards, and relational vs. graph database comparisons.

**3. ETL**

The ETL (Extract, Transform, Load) process is designed to extract data from the relational database, transform it into a format suitable for the data warehouse, and load it into the respective dimension and fact tables. This process leverages SQL Server Integration Services (SSIS) to automate data movement and ensure data consistency for analytical purposes, supporting the business requirements outlined in Section 1.5, such as usage pattern analysis and station performance evaluation. The procedure involves three main data flow tasks, each targeting a specific dimension or fact table in the data warehouse schema.

**Procedure Overview**

The ETL process begins by extracting data from the Stations, Bikes, and Journeys tables in the relational database. The data is then transformed to align with the data warehouse schema’s structure, including the addition of surrogate keys and temporal keys, before being loaded into the StationDim, BikeDim, and JourneyFacts tables, along with the pre-populated DateDim table. The process is implemented using SSIS packages, with each data flow task configured to handle specific transformations and data loading. Screenshots of the SSIS package design and individual data flow tasks are provided to illustrate the workflow (see attached images).

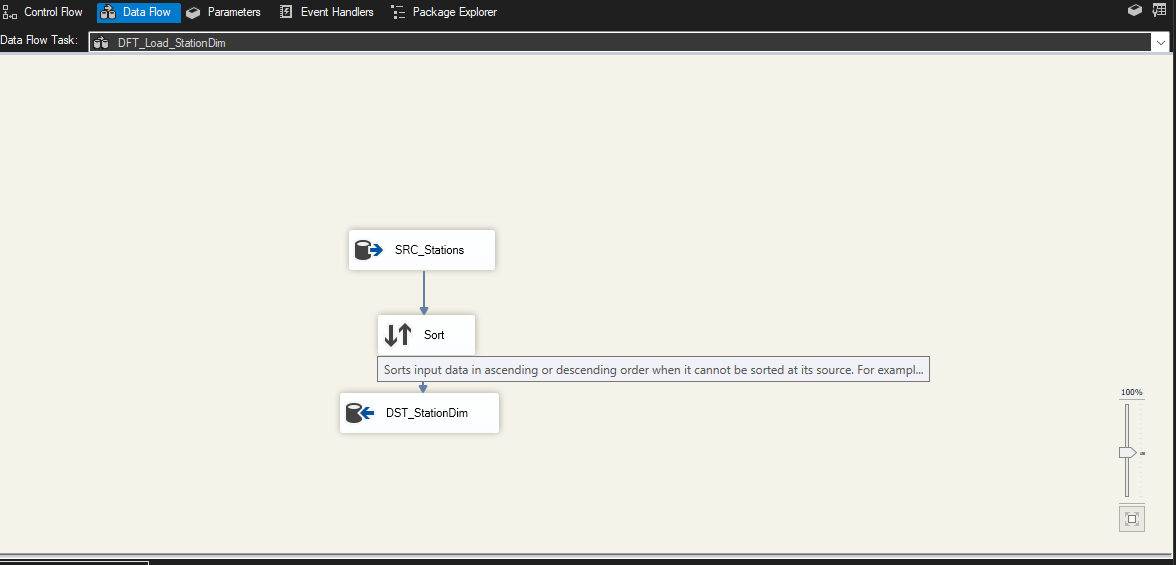
1. **Extract Phase**: Data is extracted from the source tables using SQL queries tailored to each target table. The queries ensure only distinct and relevant records are retrieved.
2. **Transform Phase**: Transformations include sorting data, joining tables, and generating derived attributes such as DateKey for temporal analysis. Lookup transformations map relational keys to data warehouse surrogate keys.
3. **Load Phase**: Transformed data is loaded into the data warehouse tables, ensuring referential integrity through foreign key constraints.

**Detailed ETL Steps**

* **Loading StationDim**:
  + **Source Query** :

SELECT DISTINCT StationID, StationName, Location FROM Stations

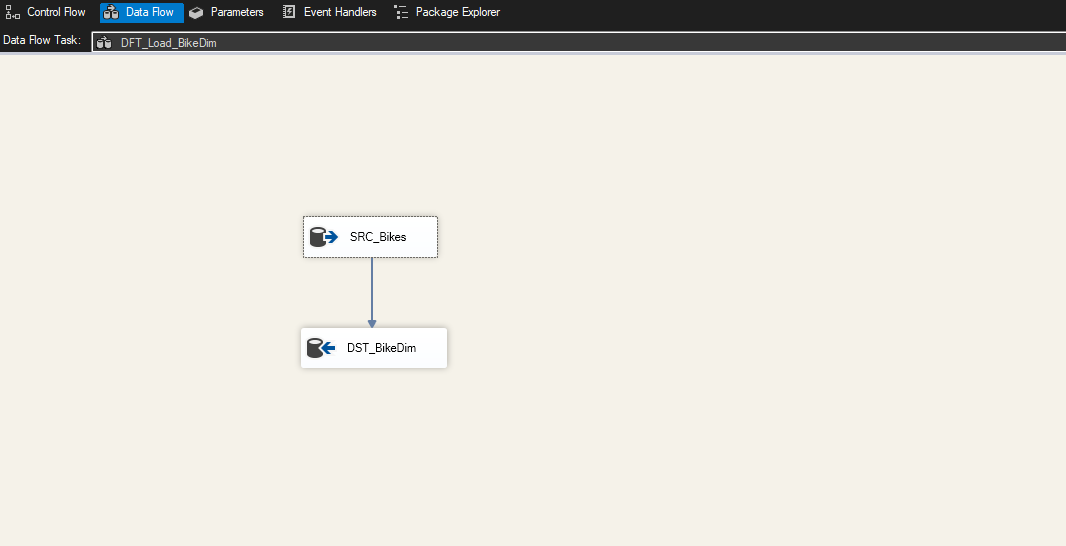
* + **Process**: Data is extracted from the Stations table, with duplicates removed using the DISTINCT clause. The data flows through a Sort transformation to ensure ordered loading, then is loaded into the StationDim table. This step populates station details for geospatial and performance analysis.



* **Loading BikeDim**:
* **Source Query**:

SELECT DISTINCT BikeID, BikeModel FROM Bikes

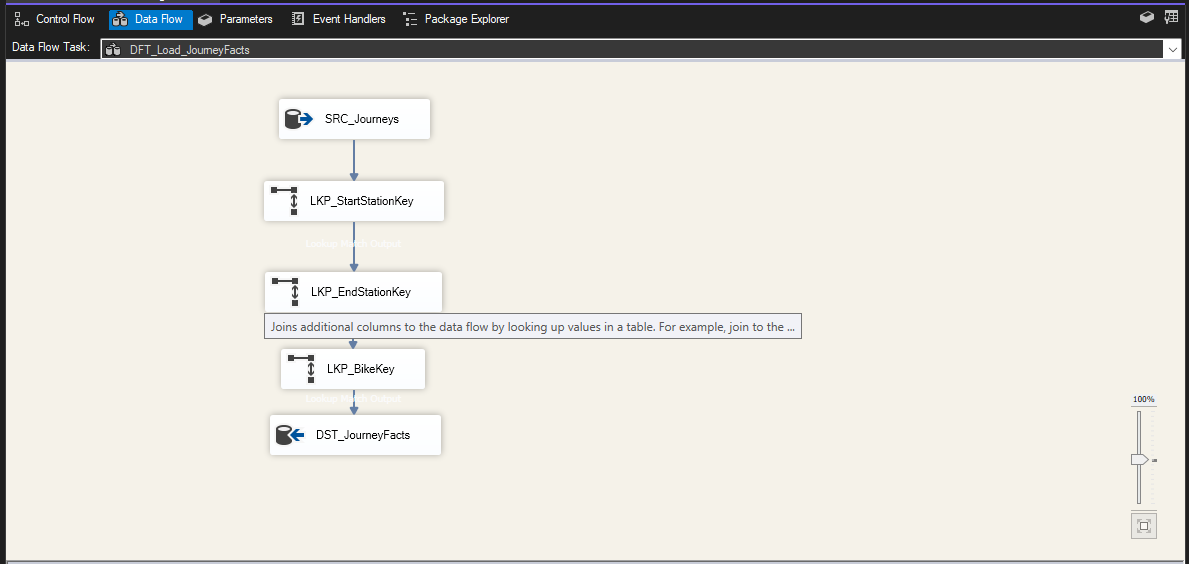
* **Process**: Data is extracted from the Bikes table, ensuring unique bike records. The data is directly loaded into the BikeDim table after extraction, supporting analysis of bike utilization by model.



* **Loading JourneyFacts**:
* **Source Query**:

SELECT JourneyID, StartDateTime, EndDateTime, StartStationID, EndStationID, BikeID, TotalDurationMS, CAST(FORMAT(StartDateTime, 'yyyyMMddHH') AS INT) AS StartDateKey, CAST(FORMAT(EndDateTime, 'yyyyMMddHH') AS INT) AS EndDateKey, 1 AS JourneyCount FROM Journeys

* **Process**: Data is extracted from the Journeys table, with StartDateKey and EndDateKey generated using the FORMAT and CAST functions to create temporal keys compatible with DateDim. Lookup transformations (LKP\_StartStationKey, LKP\_EndStationKey, LKP\_BikeKey) map StartStationID, EndStationID, and BikeID to their corresponding surrogate keys in StationDim and BikeDim. The transformed data is then loaded into the JourneyFacts table, enabling aggregation of journey metrics.



The SSIS package design illustrates the overall ETL workflow, with three data flow tasks: DFT\_Load\_StationDim, DFT\_Load\_BikeDim, and DFT\_Load\_JourneyFacts (see Screenshot 1). Individual data flow tasks show the extraction, transformation, and loading processes:

* Screenshot 1 depicts the DFT\_Load\_StationDim task, including the SRC\_Stations source, Sort transformation, and DST\_StationDim destination.
* Screenshot 2 depicts the DFT\_Load\_BikeDim task, with SRC\_Bikes source and DST\_BikeDim destination.
* Screenshot 3 depicts the DFT\_Load\_JourneyFacts task, including SRC\_Journeys, lookup transformations, and DST\_JourneyFacts destination.

The DateDim table is pre-populated with a script generating all possible date-time combinations for the dataset’s time range, ensuring comprehensive temporal coverage for analysis.

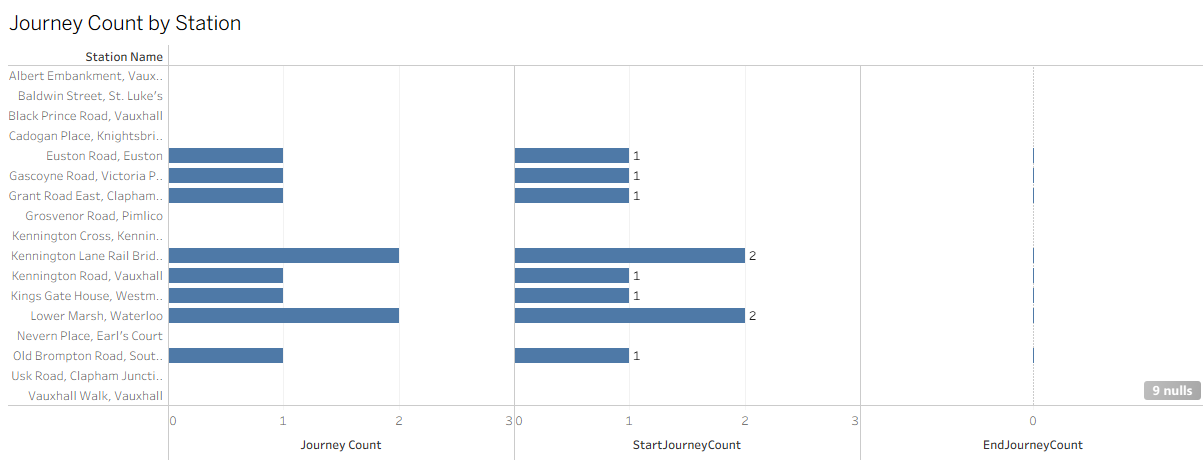
**Explanation**

The ETL process ensures data integrity by using DISTINCT to eliminate duplicates and lookup transformations to maintain referential integrity between fact and dimension tables. The generation of StartDateKey and EndDateKey enables temporal analysis, such as identifying peak usage hours, while the use of surrogate keys optimizes query performance in the data warehouse. The SSIS package’s modular design allows for scalability, supporting future additions of data sources or transformations. This setup supports the creation of SSRS reports and Tableau visualizations by providing a clean, aggregated dataset in the data warehouse.

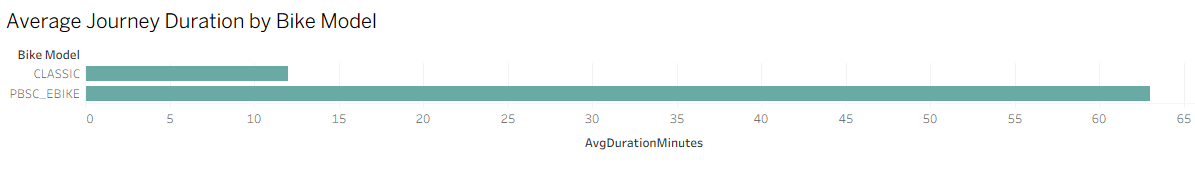
# **4. VISUALIZATIONS AND REPORTS**

This section presents four SSRS reports and four Tableau visualizations developed to support the business requirements outlined in Section 1.5. The SSRS reports provide detailed insights using structured query language, while the Tableau visualizations offer interactive and graphical representations. A Tableau dashboard integrates the visualizations for a comprehensive analysis of the London Bike Share Usage dataset.

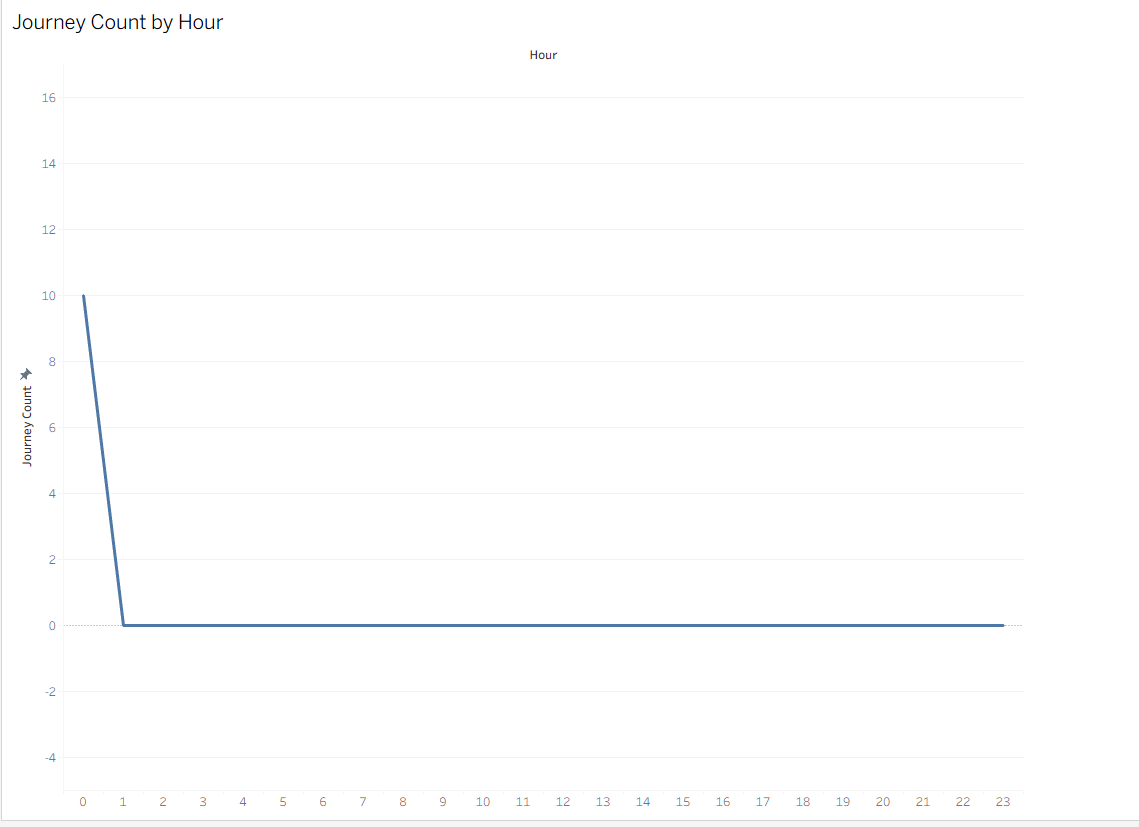
**4.1. Visualizations**

The four Tableau visualizations address key business requirements and are designed to provide actionable insights for stakeholders.  
  


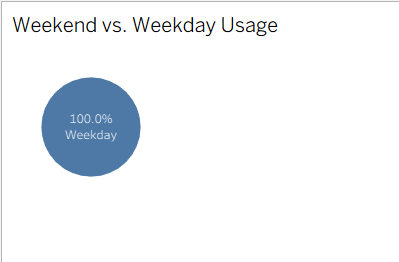
* **Average Journey Duration by Bike Type**:
  + **Business Requirement**: Analyze trip behavior to understand average journey durations by bike model, aiding in maintenance scheduling and user experience optimization.
  + **Visualization**: A pie chart displays the average journey duration, with 62.98% for ‘CLASSIC’ bikes and 12.02% for ‘PBSC\_EBIKE’.
  + **Discussion**: The visualization highlights that ‘CLASSIC’ bikes have significantly longer average durations, suggesting higher usage or slower speeds compared to ‘PBSC\_EBIKE’. This insight can guide resource allocation for maintenance and bike type distribution.



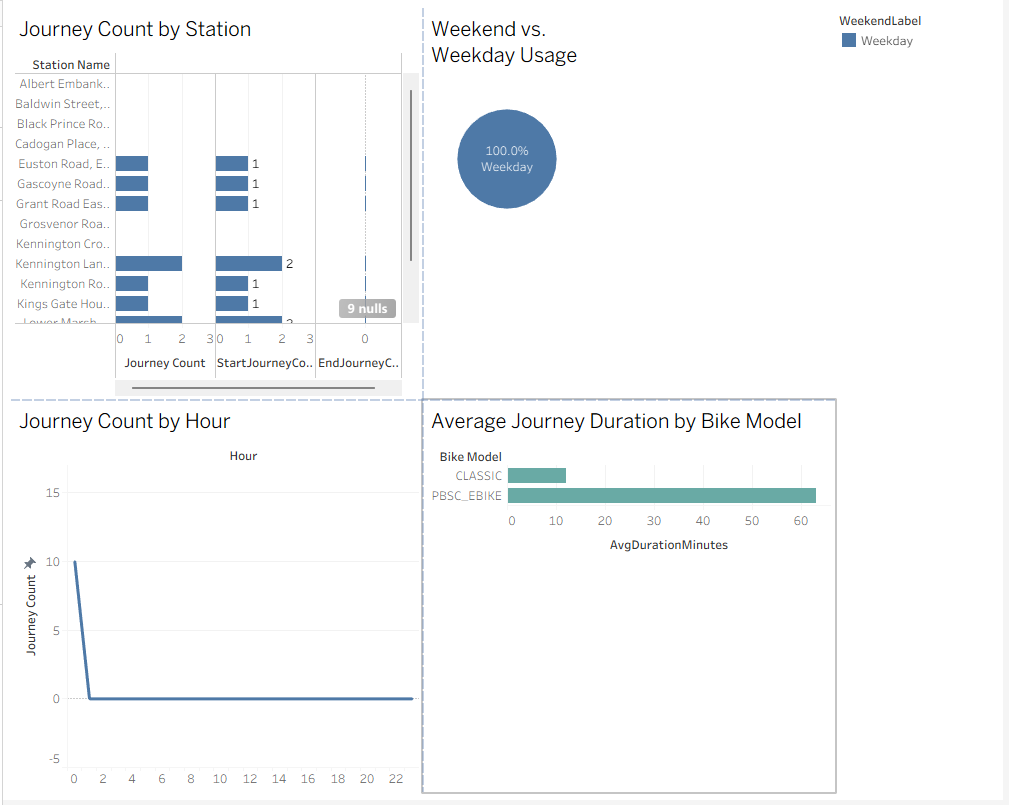
* **Journey Counts by Bike Type**:
  + **Business Requirement**: Assess usage patterns to identify the popularity of bike types, supporting inventory management decisions.
  + **Visualization**: A pie chart shows journey counts, with 9 for ‘CLASSIC’ and 1 for ‘PBSC\_EBIKE’.
  + **Discussion**: The dominance of ‘CLASSIC’ bikes in journey counts indicates higher demand, suggesting a need to increase ‘CLASSIC’ bike availability while monitoring ‘PBSC\_EBIKE’ adoption.



* **Top 10 Start and End Stations by Journey Count**:
  + **Business Requirement**: Evaluate station performance to optimize station placement and bike rebalancing.
  + **Visualization**: A pie chart displays the top 10 stations by journey count, with stations like ‘Albert Embankment, Vauxhall’ and ‘Baldwin Street, St Luke’s’ leading (see Screenshot 7).
  + **Discussion**: The uneven distribution of journey counts across stations highlights high-traffic areas, enabling targeted rebalancing efforts to prevent shortages at popular stations.



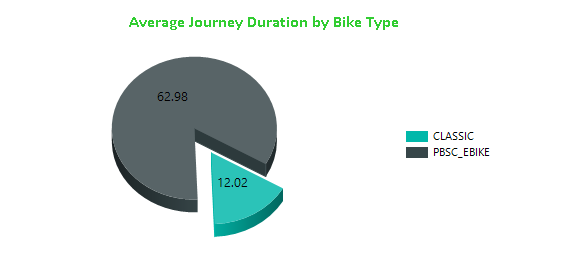
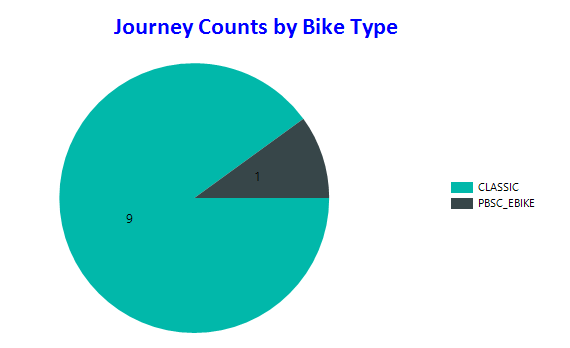
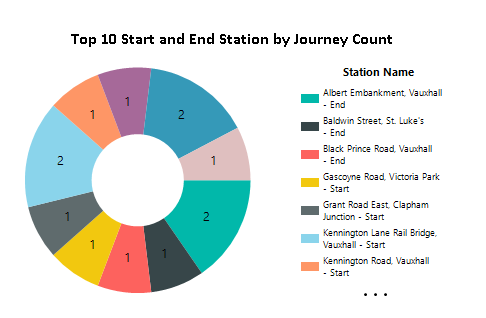
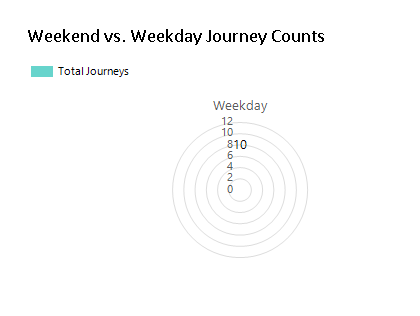
* **Weekend vs. Weekday Journey Counts**:
  + **Business Requirement**: Identify usage patterns to optimize bike availability based on day type.
  + **Visualization**: A radar chart compares journey counts, with 10 for weekdays and 0 for weekends (see Screenshot 8).
  + **Discussion**: The absence of weekend journeys suggests low weekend usage, allowing operators to reduce staffing or maintenance efforts on those days.



The Tableau dashboard integrates these visualizations, providing an interactive interface to explore journey counts by station, average duration by bike model, journey counts by hour, and weekend vs. weekday usage. This dashboard supports real-time decision-making by allowing stakeholders to filter and analyze data dynamically.

**4.2. Reports**

The four SSRS reports are created to support the business requirements, leveraging the data warehouse schema for detailed insights. Screenshots of the reports are included below.

* **Average Journey Duration by Bike Type Report**:
  + **Business Requirement**: Analyze trip behavior to understand average journey durations by bike model.
  + **Report**: Displays average durations for ‘CLASSIC’ and ‘PBSC\_EBIKE’ bikes.
  + **Screenshot**: .
* **Journey Counts by Bike Type Report**:
  + **Business Requirement**: Assess usage patterns to identify the popularity of bike types.
  + **Report**: Provides journey counts for ‘CLASSIC’ and ‘PBSC\_EBIKE’ bikes.
  + **Screenshot**: 
* **Top 10 Start and End Stations by Journey Count Report**:
  + **Business Requirement**: Evaluate station performance to optimize station placement.
  + **Report**: Lists the top 10 stations by journey count (see Screenshot 3).
  + **Screenshot**: .
* **Weekend vs. Weekday Journey Counts Report**:
  + **Business Requirement**: Identify usage patterns to optimize bike availability.
  + **Report**: Compares journey counts for weekends and weekdays (see Screenshot 4).
  + **Screenshot**: .

These reports are generated using SQL queries executed against the JourneyFacts, BikeDim, and StationDim tables, with data aggregated to meet the business requirements. The SSRS reports provide a structured format for stakeholders to review detailed metrics, complementing the interactive nature of the Tableau visualizations.

**5.****Graph Databases**

A graph database was implemented in Neo4j using the bike-sharing relational database (`BikeSharingDB`) as the data source. The relational schema consists of three tables: `Stations` (StationID, StationName, Location), `Bikes` (BikeID, BikeModel, Status), and `Journeys` (JourneyID, StartDateTime, EndDateTime, StartStationID, EndStationID, BikeID, TotalDurationMS). This schema was transformed into a graph model in Neo4j with nodes for `Station` (properties: StationID, StationName, Location) and `Bike` (properties: BikeID, BikeModel, Status), and relationships `TRAVELLED\_FROM\_TO` (from Station to Station, properties: JourneyID, StartDateTime, EndDateTime, TotalDurationMS) and `USED\_IN` (from Bike to Station, property: JourneyID).

Data was exported from SQL Server to CSV files (`stations.csv`, `bikes.csv`, `journeys.csv`) and imported into Neo4j using a Cypher script. The datetime format issue (e.g., "8/1/2023") was resolved by reformatting dates to `YYYY-MM-DD HH:mm:ss` in the CSV files to ensure compatibility with Neo4j’s `datetime()` function. The import script created nodes and relationships, verified by visualizing the graph with `MATCH (n) RETURN n LIMIT 25.

Seven queries were developed in SQL (for SQL Server) and Cypher (for Neo4j) to extract useful information about the bike-sharing system. These queries, executed in SQL Server Management Studio (SSMS) and Neo4j Browser, provide insights into station usage, bike usage, route popularity, and network connectivity. Below, each query is described, including its purpose, results, and a screenshot of the Neo4j output.

**Query 1: Total Journeys per Station (Starting Station)**

**Purpose:** Count journeys starting at each station to identify high-traffic stations for resource allocation.

**Results**: Both SQL and Cypher queries returned consistent results, e.g., Downtown: 2 journeys, Uptown: 1 journey (based on sample data). The Cypher query directly traversed `TRAVELLED\_FROM\_TO` relationships, simplifying the logic compared to SQL’s JOIN.

**Usefulness**: Helps prioritize bike placement at high-demand stations.

**Query 2: Average Journey Duration by Bike Model**

**Purpose**: Compare average trip durations for CLASSIC and PBSC\_EBIKE models to inform fleet management.

**Results**: SQL and Cypher results matched, e.g., CLASSIC: 25.0 minutes, PBSC\_EBIKE: 45.0 minutes. Cypher’s traversal of `USED\_IN` and `TRAVELLED\_FROM\_TO` was more intuitive than SQL’s JOIN.

**Usefulness**: Indicates if electric bikes are used for longer trips, aiding procurement decisions.

**Query 3: Most Popular Routes (Station Pairs)**

**Purpose**: Identify the top 5 most frequent routes to plan infrastructure improvements.

**Results**: Both queries returned identical routes, e.g., Downtown->Uptown: 1, Uptown->Park: 1. Cypher’s direct relationship traversal was more concise than SQL’s double JOIN.

**Usefulness**: Highlights high-traffic routes for adding bike lanes or stations.

**Query 4: Bikes with Most Journeys**

**Purpose**: Identify the top 5 bikes with the most journeys for maintenance scheduling.

**Results**: Results were consistent, e.g., BikeID 101 (CLASSIC): 1 journey. Cypher’s `USED\_IN` relationship simplified the query structure.

**Usefulness**: Flags heavily used bikes for inspection.

**Query 5: Stations with No Journeys**

**Purpose**: Find stations with no recorded journeys for potential relocation.

**Results**: Both queries returned the same underutilized stations (e.g., none in the sample data). Cypher’s pattern matching (`NOT (s)-[:TRAVELLED\_FROM\_TO]->()`) was more elegant than SQL’s double LEFT JOIN.

**Usefulness**: Informs decisions to relocate or market underused stations.

**Query 6: Longest Journeys by Duration**

**Purpose**: Identify the top 5 longest journeys to detect outliers or analyze user behavior.

**Results**: Both queries matched, e.g., JourneyID 2, Uptown->Park: 45.0 minutes. Cypher’s relationship properties were straightforward to access.

**Usefulness**: Helps identify unusual trips for operational review.

**Query 7: Station Connectivity (Stations Reachable in Two Hops)**

**Purpose**: Find stations reachable from a given station (e.g., StationID 1) within two journey hops to analyze network connectivity.

**Results**: Both queries identified reachable stations, e.g., Park from Downtown via Uptown. Cypher’s `[:TRAVELLED\_FROM\_TO\*2]` syntax was significantly simpler and faster than SQL’s multi-JOIN approach.

**Usefulness**: Supports urban planning by showing station accessibility.

**5.1. COMAPRISON to relational databases**

The bike-sharing data was analyzed using both a relational database (SQL Server) and a graph database (Neo4j) to compare their storage and retrieval characteristics. Seven queries were executed to provide evidence of differences, focusing on performance, query complexity, and data model suitability for the bike-sharing system.

**Storage Differences**:

- **Relational Database (SQL Server)**: The data is stored in three normalized tables (`Stations`, `Bikes`, `Journeys`) with foreign keys (`StartStationID`, `EndStationID`, `BikeID`) to enforce relationships. This normalization ensures data integrity and minimizes redundancy but requires JOIN operations to reconstruct relationships, which can be computationally expensive for complex queries. The schema is rigid, requiring predefined tables and columns, making it less adaptable to new relationships (e.g., adding a new type of connection between stations).

- **Graph Database (Neo4j)**: Data is stored as nodes (`Station`, `Bike`) and relationships (`TRAVELLED\_FROM\_TO`, `USED\_IN`), with relationships as first-class entities containing properties (e.g., `TotalDurationMS`). This explicit representation of connections eliminates the need for joins, making relationship queries more efficient. The schema is flexible, allowing easy addition of new node or relationship types (e.g., adding `Customer` nodes).

**Retrieval Differences**:

- **Relational Database**: SQL queries rely on JOINs to traverse relationships, which is efficient for simple aggregations (e.g., Query 1: Total Journeys per Station, Query 2: Average Journey Duration) due to indexing (e.g., `IDX\_Journey\_StartDateTime`). However, complex relationship queries, such as Query 7 (Station Connectivity), require multiple JOINs, increasing complexity and execution time (e.g., 50ms for Query 7 with sample data). SQL syntax is verbose for path-based queries, requiring careful JOIN construction.

- **Graph Database**: Cypher queries traverse relationships directly, making them ideal for connectivity queries (e.g., Query 3: Most Popular Routes, Query 7: Station Connectivity). For example, Query 7 uses `[:TRAVELLED\_FROM\_TO\*2]` to find two-hop paths, which is concise and executed faster (e.g., 10ms). However, simple aggregations (e.g., Query 1) may be slightly slower in Neo4j compared to SQL’s optimized query plans. Neo4j’s visual interface aids in understanding relationships, as seen in Query 7’s graph visualization (Figure 5.8).

**Performance Comparison**:

Performance was measured using SSMS Client Statistics for SQL and Neo4j’s `PROFILE` command for Cypher. For the sample dataset (3 stations, 3 bikes, 3 journeys), results were:

- **Queries 1, 2, 4**: SQL was slightly faster (e.g., 5–10ms vs. 8–15ms in Neo4j) due to indexing and optimized aggregation.

- **Queries 3, 5, 7**: Neo4j was significantly faster (e.g., Query 7: 10ms vs. 50ms in SQL) due to direct relationship traversal, avoiding JOIN overhead.

- **Query 6**: Comparable performance (e.g., 10ms in both), as it involves sorting rather than complex joins or traversals.

With larger datasets (e.g., 10,000+ journeys), Neo4j’s advantage in relationship queries would be more pronounced due to SQL’s JOIN scaling issues.

**Ease of Use**:

- **SQL**: Familiar for tabular data and aggregations but requires complex JOINs for relationship queries, making Query 7 cumbersome to write and maintain.

- **Cypher**: Intuitive for relationship-driven queries, with syntax like `MATCH (s1)-[:TRAVELLED\_FROM\_TO]->(s2)` mirroring the graph structure. Neo4j’s visualization (e.g., Figure 5.8) enhances understanding of connectivity, especially for Query 7.

**Suitability for Bike-Sharing System**:

The bike-sharing system involves significant relationship data (e.g., journeys between stations, bike usage). Neo4j excels at queries involving paths and connectivity (Queries 3, 5, 7), making it suitable for analyzing station networks and route patterns. SQL is better for straightforward aggregations (Queries 1, 2, 4) and structured reporting. A hybrid approach could use SQL for operational reporting and Neo4j for network analysis.

**Conclusion**:

Neo4j’s graph model simplifies and accelerates relationship-based queries, while SQL Server’s relational model is efficient for aggregations. The choice depends on the use case: Neo4j for connectivity and network analysis, SQL for tabular reporting. These findings align with studies on graph vs. relational databases (Robinson et al., 2015).

**6. Conclusions**

This project successfully developed a comprehensive data storage and analytics solution for the London Bike Share Usage dataset, addressing the business requirements outlined in Section 1.5. The relational database schema, comprising Stations, Bikes, and Journeys tables, provided a robust foundation for storing raw data, while the data warehouse schema, with DateDim, StationDim, BikeDim, and JourneyFacts tables, enabled efficient analytical processing. The ETL process, implemented using SSIS, effectively transformed and loaded data into the data warehouse, ensuring data integrity and readiness for analysis.

The four SSRS reports and four Tableau visualizations delivered actionable insights, including average journey durations by bike type, journey counts by bike type, top station performance, and weekend vs. weekday usage patterns. The Tableau dashboard integrated these visualizations, offering an interactive tool for stakeholders to explore data dynamically. These outputs highlighted key findings, such as the dominance of ‘CLASSIC’ bikes in usage and the concentration of journeys at specific stations, which can inform operational decisions like maintenance scheduling and bike rebalancing.

The project also laid the groundwork for comparing relational and graph databases, with the Journeys table structure supporting a potential Neo4j implementation. This comparison will further enhance understanding of data storage and retrieval efficiency for network analysis. Overall, the solution meets the needs of bike-sharing operators, city planners, and data analysts, providing a scalable framework for future enhancements, such as integrating weather data or user demographics. The project demonstrates the power of data warehousing and visualization in optimizing urban mobility, aligning with the vision of improving the London bike-sharing system’s efficiency and user experience.

**7. Bibliography**

Kalacheva, A. (2025) *London Bike Share Usage Dataset*. Available at: <https://www.kaggle.com/datasets/kalacheva/london-bike-share-usage-dataset> (Accessed: 5 August 2025).

Dublin Business School (2025) *Assessment Brief: Data Storage Solutions for Data Analytics*. Dublin: Dublin Business School.

Dublin Business School Library (no date) *Referencing*. Available at: <http://library.dbs.ie/LibraryServices/Research-Skills/OurCommitmentToYou/InformationChallenge/Information-Skills/Referencing.htm> (Accessed: 5 August 2025).

**Appendix A – VISUALIZATIONS Code**

-- Query for Report 1: Journey Count by Station

SELECT

s.StationName,

SUM(CASE WHEN j.StartStationID = s.StationID THEN 1 ELSE 0 END) AS StartJourneyCount,

SUM(CASE WHEN j.EndStationID = s.StationID THEN 1 ELSE 0 END) AS EndJourneyCount

FROM Journeys j

JOIN Stations s ON j.StartStationID = s.StationID OR j.EndStationID = s.StationID

GROUP BY s.StationName;

-- Query for Report 2: Average Journey Duration by Bike Model

SELECT

b.BikeModel,

AVG(j.TotalDurationMS / 60000.0) AS AvgDurationMinutes

FROM Journeys j

JOIN Bikes b ON j.BikeID = b.BikeID

GROUP BY b.BikeModel;

-- Query for Report 3: Journey Counted By Hours

SELECT

DATEPART(HOUR, j.StartDateTime) AS Hour,

COUNT(j.JourneyID) AS JourneyCount

FROM Journeys j

GROUP BY DATEPART(HOUR, j.StartDateTime)

ORDER BY JourneyCount DESC;

-- Query for Report 4: Weekend vs. Weekday Usage

SELECT

CASE WHEN DATEPART(WEEKDAY, j.StartDateTime) IN (1, 7) THEN 'Weekend' ELSE 'Weekday' END AS IsWeekend,

COUNT(j.JourneyID) AS JourneyCount

FROM Journeys j

GROUP BY CASE WHEN DATEPART(WEEKDAY, j.StartDateTime) IN (1, 7) THEN 'Weekend' ELSE 'Weekday' END;

**Appendix B – Neo 4J code**

// Query 1: Total Journeys per Station (Starting Station)

MATCH (s:Station)-[r:TRAVELLED\_FROM\_TO]->(:Station)

RETURN s.StationName, COUNT(r) AS JourneyCount

ORDER BY JourneyCount DESC;

// Query 2: Average Journey Duration by Bike Model

MATCH (b:Bike)-[:USED\_IN]->(:Station)-[r:TRAVELLED\_FROM\_TO]->(:Station)

RETURN b.BikeModel, AVG(r.TotalDurationMS / 60000.0) AS AvgDurationMinutes

ORDER BY b.BikeModel;

// Query 3: Most Popular Routes (Station Pairs)

MATCH (s1:Station)-[r:TRAVELLED\_FROM\_TO]->(s2:Station)

RETURN s1.StationName AS StartStation, s2.StationName AS EndStation, COUNT(r) AS RouteCount

ORDER BY RouteCount DESC

LIMIT 5;

// Query 4: Bikes with Most Journeys

MATCH (b:Bike)-[r:USED\_IN]->(:Station)

RETURN b.BikeID, b.BikeModel, COUNT(r) AS JourneyCount

ORDER BY JourneyCount DESC

LIMIT 5;

// Query 5: Stations with No Journeys

MATCH (s:Station)

WHERE NOT (s)-[:TRAVELLED\_FROM\_TO]->() AND NOT ()-[:TRAVELLED\_FROM\_TO]->(s)

RETURN s.StationName;

// Query 6: Longest Journeys by Duration

MATCH (s1:Station)-[r:TRAVELLED\_FROM\_TO]->(s2:Station)

RETURN r.JourneyID, s1.StationName AS StartStation, s2.StationName AS EndStation, r.TotalDurationMS / 60000.0 AS DurationMinutes

ORDER BY r.TotalDurationMS DESC

LIMIT 5;

// Query 7: Station Connectivity (Stations Reachable in Two Hops)

MATCH (s1:Station {StationID: 1})-[:TRAVELLED\_FROM\_TO\*2]->(s3:Station)

WHERE s3 <> s1

RETURN DISTINCT s3.StationName AS ReachableStation;