Imperial College London

MEng Individual Project

Imperial College London

Department of Computing

A Data Visualisation Tool Based On Conceptual Modelling

Author:
Yeye Chen

Supervisor:
Dr. Peter McBrien

Second Marker:
Dr. Thomas Heinis

June 18, 2023

Abstract

Modern data visualisation tools are highly flexible, giving users complete freedom to explore their data. However, an excessive degree of freedom can cause confusion and result in an inappropriate visual representation of the data. We deliver a data visualisation tool that utilises the full knowledge of the conceptual schema of the data source, by mapping it to a set of visualisation schema patterns, in order to simplify the process of visualisation generation. This tool has demonstrated its effectiveness in various visualisation tasks. In addition, we have explored expanding additional visualisations for specific patterns.

Acknowledgements

I would like to thank my supervisor Dr. Peter McBrien for the opportunity to work on this project and for the invaluable guidance and support throughout its duration. Dr. McBrien's expertise and insightful advice have been crucial in shaping the success of this project.

I would also like to thank my friends for their constant support and motivation over the past four years, and my family for their belief in my abilities and continuous encouragement to pursue my goals.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Data visualisation plays an important role in data analysis. Visual representations of complex data sets such as graphs, charts and maps, aid individuals to comprehend insights into a data set. Data visualisation also helps in decision-making, by revealing patterns and trends of the raw data. Colours and shapes in visualisations are effective and direct language that can be used for communication.

Modern data visualisation tools such as Tableau [3] and Google Chart [4] are highly capable of fulfilling the needs of most users. They are very mature in terms of user-friendly interfaces and diverse visualisation options. They are also highly flexible, giving users more options to explore their data. However, an excessive degree of freedom can sometimes result in inappropriate visualisations. The underlying reason is that current data visualisation tools are not able to fully make use of the schema information of the data, resulting in a lack of higher-level abstraction in the process of data selection and visualisation generation.

1.2 Example

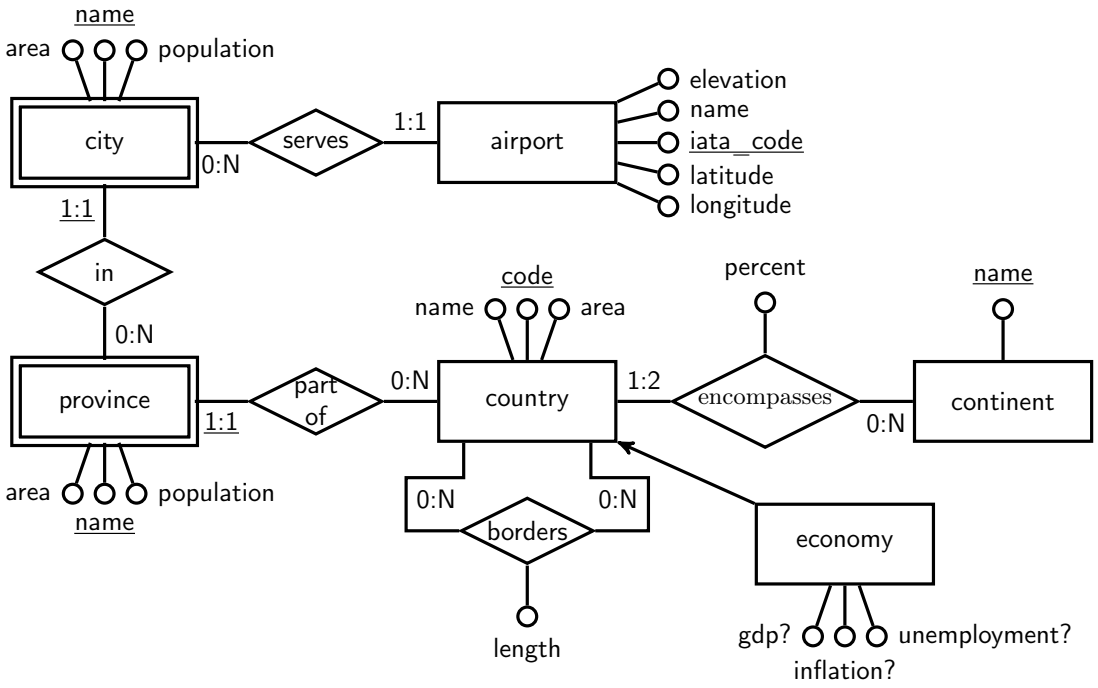


Figure 1.1: ER schema of a fragment of the Mondial database [1]

Figure 1.1 is the ER schema of a fragment of the Mondial database [1], focusing on country-related information.

The relationship **borders** stores information about the borders between adjacent **countries**. The weak entity **province** captures data related to provinces within a country and the weak entity **city** stores information regarding cities within a province. The **airports** entity provides details about airports located in cities. In certain cases, countries may have additional data concerning their economic situation, which is stored in the subset entity **economy**. The optional attributes within **economy** allow for the recording of various economic indicators. Lastly, the **encompasses** relationship establishes the connection between **countries** and **continents**, with the **percent** attribute denoting the proportion of a country's land area that belongs to each respective continent.

Let's consider the relationship between countries and the continents, specifically focusing on the cardinality constraint of **1:2**. This constraint indicates that certain countries, such as Russia and Turkey, span across two continents because of their geographical positions. In Tableau, users are given the option to visualise this relationship using a tree map based on countries' areas. Each country is represented by a rectangle, and the size of each rectangle corresponds to the country's area, as shown in figure 1.2. However, it can be observed that Russia appeared twice in both Asia and Europe, which results in a significant increase in the total area of Europe. This tree map misleads users that the area of Europe is comparable to that of Africa and significantly larger than Australia. This particular visualisation is inaccurate and falls short of meeting user expectations.

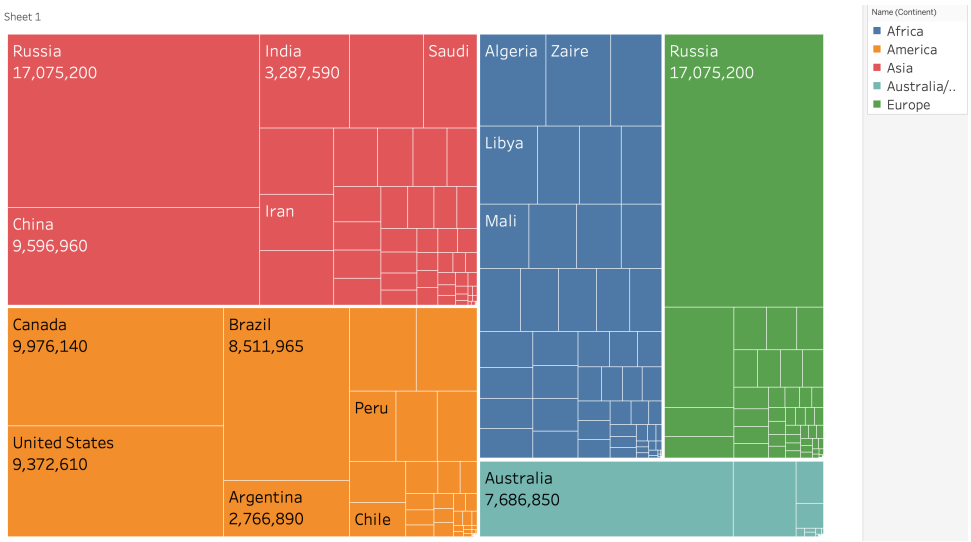


Figure 1.2: Inappropriate tree map by Tableau

To address this issue, a user may need to apply filters to the dataset to make the visualisation better aligned with their requirements. For example, filtering the data by limiting countries where more than 50% of a country's area lies within a specific continent. By doing so, each country is guaranteed to be related to one continent (i.e. one-many relationship). Alternatively, a user may explore other visualisations that offer a more suitable representation of the current relationship between countries and continents (i.e. many-many relationship), such as a sankey diagram.

The reason why inappropriate visualisations are often generated by Tableau and other visualisation tools is that they fail to capture the full schema knowledge, requiring users to manually select data and choose visual encodings from a wide range of options. As a result, users may struggle to comprehend the underlying meaning of the data and the options of visualisations available. This situation can often lead to undesired visualisation results.

In this project, we aim to introduce a data visualisation solution that utilises the schema information of the databases. We will achieve this by using a set of **visualisation schema patterns** which categorise commonly-used data visualisations into distinct groups. By mapping the data schema to these patterns, we can guide users in selecting the most suitable visualisations for their data.

1.3 Contributions

The main goal of this project is to simplify the process of selecting a suitable visualisation by providing users with a more focused set of options. By utilising the full knowledge of the conceptual schema of the data source, this application guides users in choosing visualisations that are appropriate for their data. This is done by employing pattern matching to align data schema with visualisation schema patterns.

In addition, our objective is to introduce a data visualisation tool that enables users to gain a deeper understanding of their data and facilitate exploration. This application offers a wide range of visualisation possibilities and enables users to interactively explore and fulfil their information-seeking needs through filtering.

Furthermore, we have explored expanding and implementing the additional visualisations for specific schema patterns, providing users with a broader selection of suitable visualisations for their data.

Chapter 2

Background

2.1 Grammar of Graphics

The Grammar of Graphics [5] is a conceptual framework for describing and creating data visualisations. It provides a systematic approach to understanding and constructing visualisations by breaking them down into seven fundamental components.

Figure 2.1 shows a data flow diagram consisting of seven classes. These classes represent the sequence of mappings required to generate a statistical graphic from a dataset. The data flow architecture implies that these subtasks must be executed in the specified order to ensure the production of meaningful graphics. Any deviation from this prescribed order can lead to the creation of meaningless graphics.

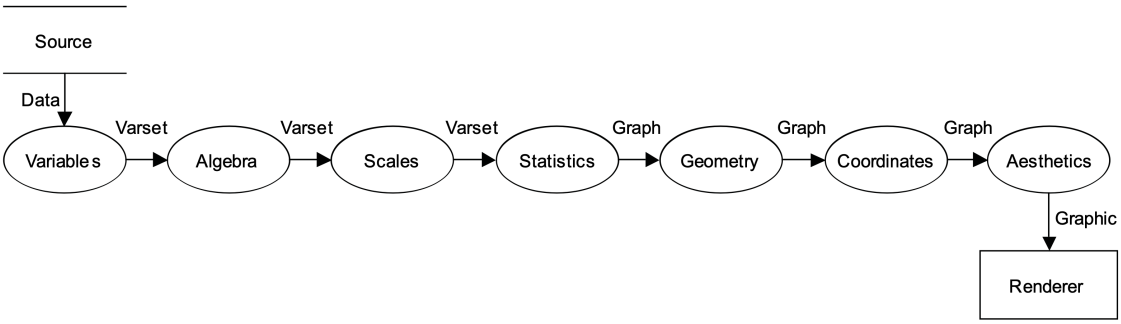


Figure 2.1: The data flow diagram for producing meaningful graphics

2.2 Visualisation Schema Patterns

ER (Entity-Relationship) schemas are typically employed as conceptual models before implementing actual database systems. The ER schema not only captures the structure of data but also incorporates details regarding the relationships between entities and mapping cardinality between tables.

There are five ER schema patterns in recent research [6, 7] by P. McBrien and A. Poulovassilis, accompanied by their corresponding visualisations. Figure 2.2 and table 2.1 illustrate those patterns and visualisations of which the level of complexity increases successively.

Note that the Reflexive Relationship pattern can be regarded as a special case of the Many-Many Relationship pattern. Consequently, the visualisations corresponding to the Reflexive Relationship can be considered a subset of those associated with the Many-Many Relationship. Additionally, it is worth noting that reflexive relationships can exist in both one-many and many-many configurations. However, the latest work has exclusively focused on the many-many case.

Figure 2.3 shows an example that can be mapped to the One-Many Relationship pattern. One-many relationships are particularly well-suited for visualisations that exhibit a hierarchical structure. For instance, we can create a **hierarchy tree**, where nodes represent child-entity instances (**airport**) connected by lines to parent-entity instances (**city**).

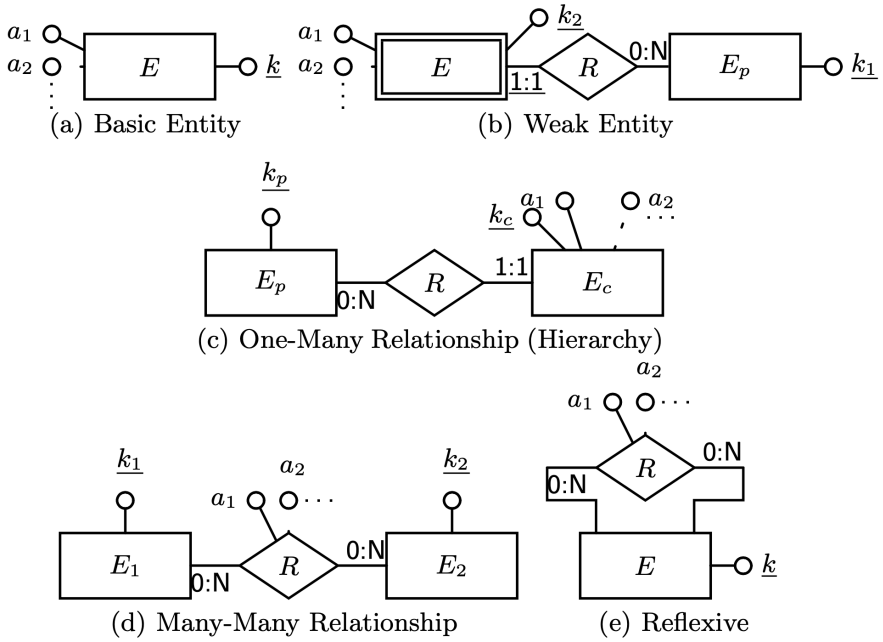


Figure 2.2: Visualisation Schema Patterns for Data Visualisation

Visualisation Schema Patterns	Corresponding Visualisastions	
Basic Entity	Bar Chart Calendar Chart Scatter Diagram	Bubble Chart Choropleth Map Word Cloud
Weak Entity	Line Chart Spider Chart	Stacked Bar Chart Grouped Bar Chart
One-Many Relationship	Hierarchy Tree Circle Packing	Tree Map
Many-Many Relationship	Sankey Diagram Network Chart	Chord Diagram Heatmap
Reflexive Relationship	Network Chart	Chord Diagram Heatmap

Table 2.1: Patterns and corresponding visualisations

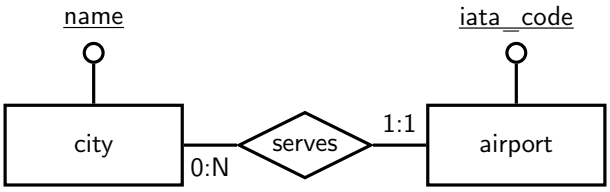


Figure 2.3: ER schema of the one-many relationship between **airport** and **city**

The reason why the example from Section 1.2 fails to meet user expectations is that the relationship between **country** and **continent** is many-many, which is incompatible with tree maps. However, by applying filters to the data, such as selecting countries and continents where more than 50% of a country's area lies within a specific continent, we can transform the relationship between **country** and **continent** from many-many to one-many. This is illustrated in figure 2.4. In this case, the dataset is suitable for those visualisations under the One-Many Relationship pattern.

An alternative approach to visualise data with many-many relationships using one-many visualizations is through a process known as **denormalization**. Denormalization involves creating a new weak entity called **encompasses_denormalised**, which includes the **percent** attribute. This transformation retains all the information present in the original schema while enabling the

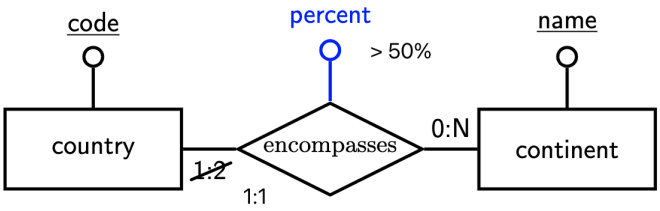


Figure 2.4: transforming a many-many relationship to a one-many relationship by filtering

generation of meaningful one-many visualizations. The denormalisation transformation is illustrated in figure 2.5.

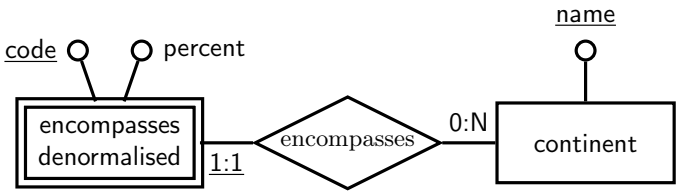


Figure 2.5: denormalisation transformation

Chapter 3

Related Work

3.1 Reverse Engineering

Databases most commonly use the relational database to store data. This project is delivered at the ER level instead of the relational level, hence reverse engineering from relational schemas to ER schemas is the initial step to enable the utilisation of the set of visualisation schema patterns.

One of the primary objectives of the database reverse engineering process is to enhance the understanding of data semantics. This process becomes particularly crucial for legacy databases where knowledge of data semantics has been lost over time. By undertaking reverse engineering, valuable insights can be gained, enabling a deeper comprehension of the underlying data and its meaning.

The forward engineering process from an ER schema to a relational model is relatively straightforward. Database designers can systematically convert the ER schema into a set of relational tables, ensuring data integrity and preserving the relationships between entities. However, reversing this process is no trivial task due to the substantial loss of information that occurs during the transformation from ER schemas to relational models.

3.1.1 Research areas

An approach proposed by Premerlani and Blaha [8] is a seven-stage process, employing an object-oriented model and adopting the Object Modeling Technique notation. The process places particular emphasis on analysing candidate keys rather than solely focusing on primary keys. The algorithm successfully identifies entities with relationships and their cardinalities.

Another approach by Andersson [9] uses his **node rules**, **link rules** and **refinement rules** to extract an extended ER model called the ERC+ model. This enhanced model incorporates multivalued and complex objects, and multi-instantiation. Node rules identify entities while link rules identify relationships. To achieve the final translation into the ERC+ model, refinement rules are applied to both nodes and links.
However, both methods do not effectively capture the necessary information to distinguish between strong and weak entities.

The approach proposed by Petit et al [2] builds upon previous methods, and effectively identifies weak entities and is-a links. The process is divided into two key steps:

1. Eliciting the data semantics from the existing system.
2. Expressing the extracted semantics with a high level data model.

The first step is crucial as it involves extracting information on functional dependencies that influence the way data could be structured. Then that information is used in the restruct algorithm, where a 1NF relational schema is restructed to a 3NF relational schema, that can be represented by an EER schema.

An example demonstrated in the paper is shown as figure 3.1, where (a) represents the original relational schema, while (b) represents the schema after applying the restruct algorithm. In the schema representations, relation names begin with an upper-case letter, underlined attributes

denote keys, and emphasized attributes indicate not-null constraints. The resulting EER schema is shown in figure 3.2, where weak entities are denoted by double boxes, and is-a links indicated by arrows with two pointers at their head.

Person(id, name, street, number, zip–code, state)
HEmployee(no, data, salary)
Department(dep, emp, skill, *location*, proj)
Assignment(emp, dep, proj, date, project–name)

(a) Original relational schema [2]: p.221

Person(id, name, street, number, zip–code, city)
HEmployee(no, date, salary)
Department(dep, emp, *location*)
Assignment(emp, dep, proj, date)
Employee(no)
Ass–Dept(dep)
Other–Dept(dep)
Manager(emp, skill, proj)
Project(proj, project–name)

(b) Relational schema after restruction [2]:p.225

Figure 3.1: Illustration of the restruct algorithm

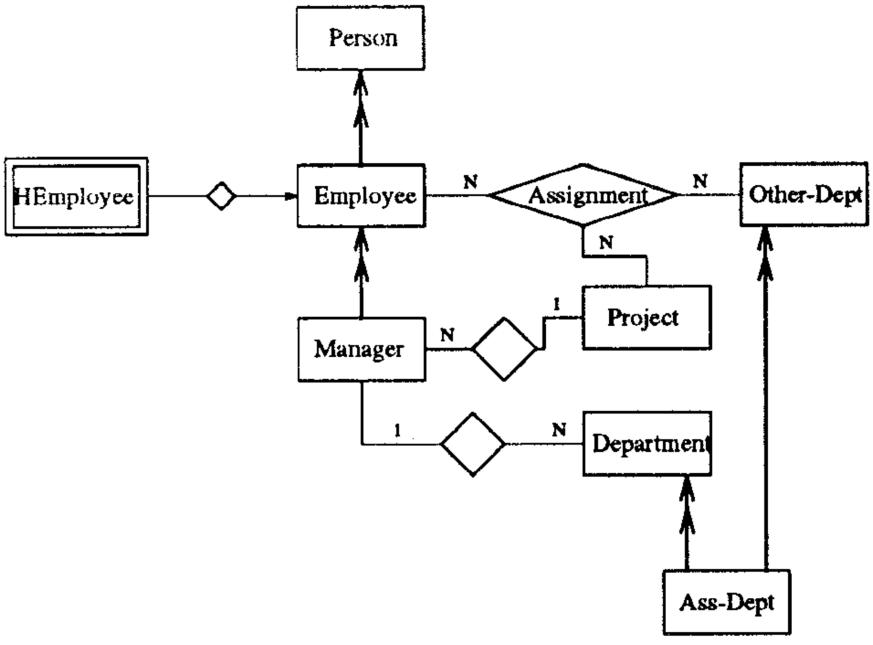


Figure 3.2: The resulting EER schema [2]:p.226

3.1.2 Existing solution

The **AmazingER** [10] is a library, recently developed by a group of students from Imperial College, that offers a comprehensive set of features for ER modelling, including the ability to reverse engineer relational databases into ER schemas. By using the database metadata, it generates Java-based ER schema objects. It identifies entities and relationships, and the associated cardinalities. Moreover, it successfully identifies weak-entity relationships and subset entities.

However, the limitation arises when dealing with reflexive relationships. The exceptions defined in the code indicate that there cannot exist more than one relationship edge between the same relationship and entity objects. As a result, reflexive relationships cannot be identified by the current implementation of the library.

Additionally, it fails to handle the weak entity **city** in figure 1.1. The **city** is a weak entity with respect to **province**, and **province** is a weak entity with respect to **country**. The reason AmazingER fails to parse **city** is that every entity is categorised into an absolute entity type, either strong or weak. Hence **city** as the child entity is not allowed to have a weak-entity relationship with another weak entity, specifically the **province** entity.

Figure 3.3 is the result of the reverse engineering of the relationship between **country** and **continent** in figure 1.1 in JSON format. Notice the cardinality between **country** and **encompasses** is inaccurately translated to "ZeroToMany" instead of "OneToMany" because of the loss of information in the forward engineering process. However, this difference does not affect the overall schema to be a many-many relationship.


```
{"schema":{
  "id" : 1,
  "name" : "reverseEng",
  "entityList" : [ {
    "id" : 2,
    "name" : "country",
    "entityType" : "STRONG",
    "attributeList" : [...]
  }, {
    "id" : 3,
    "name" : "continent",
    "entityType" : "STRONG",
    "attributeList" : [...]
  }],
  "relationshipList" : [{
    "id" : 4,
    "name" : "encompasses",
    "attributeList" : [ {
      "id" : 5,
      "name" : "percentage",
      "belongObjID" : 4,
      "belongObjType" : "RELATIONSHIP",
      "isPrimary" : false,
      "attributeType" : 1,
    } ],
    "edgeList" : [ {
      "id" : 6,
      "relationshipID" : 4,
      "belongObjID" : 2,
      "belongObjType" : "ENTITY",
      "belongObjName" : "country",
      "cardinality" : "ZeroToMany"
    }, {
      "id" : 7,
      "relationshipID" : 4,
      "belongObjID" : 3,
      "belongObjType" : "ENTITY",
      "belongObjName" : "continent",
      "cardinality" : "ZeroToMany"
    }]
  }]
}}
```

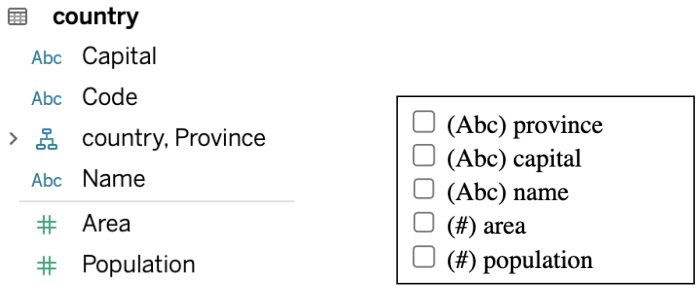

Figure 3.3: ER schema in JSON format generate by AmazingER

3.2 Visualisation Solutions

3.2.1 Tableau

Tableau[3] is a powerful and widely used visualisation tool, with a user-friendly interface and a wide range of visualisation options. It provides various ways that allow users to connect to their data sources, including file format, SQL databases and cloud data. Additionally, Tableau presents the capability to blend and join data from multiple sources, allowing domain experts to delve deeply into their data, conducting a thorough analysis. Furthermore, Tableau facilitates collaboration and sharing functionalities, making it easier for business-oriented users to effectively communicate insights. It has recently integrated artificial intelligence within the software, enabling users to use natural language to generate desired visualisations.

Tableau categorizes data types into **dimensions** and **measures** and these classifications are denoted by specific icons next to the corresponding attributes. This approach helps users to better understand their data and facilitates the selection of appropriate visualisations. Similar concepts have been incorporated into this project as well, as shown in figure 3.4.



(a) visual aid from Tableau (b) visual aid in this project

Figure 3.4: Datatype visual aid comparison

However, mastering Tableau's more advanced features and functionalities can require a learning curve. It takes time to become proficient in this tool to its full potential. Particularly for users with limited data visualisation experience, there is a possibility of creating inappropriate visualisations, as shown in example 1.2.

3.2.2 AutoViz

AutoViz[11] is an automatic visualisation tool based on machine learning. By simply providing a file in CSV, txt or JSON format, AutoViz produces a set of visualisations determined to be relevant by its algorithm. AutoViz is a Python package, and the visualisation stage is achievable through a single line of code. Its characteristics make it well-suited for users who are not familiar with their data and just want a quick exploration by plugging in their data.

AutoViz's primary drawback lies in its limited flexibility. It lacks a dedicated user interface, limiting the interactivity for data exploration beyond the use of Python commands. It does not guide users in choosing appropriate visualisations for their data. Additionally, it solely accepts file inputs and is not capable of connecting to databases.

Overall, AutoViz is not suitable for domain experts who are familiar with their data and seek to conduct extensive exploratory analysis.

3.2.3 SPSS Statistics

SPSS Statistics is a software package developed by IBM that provides advanced statistical analysis and data management capabilities. Under the hood, graphs are created by the GPL, which is a language based on The Grammar of Graphics [5]. SPSS Statistics is widely used by researchers for hypothesis testing and predictive analytics. Similar to Tableau, mastering the advanced features of SPSS Statistics may require some time and effort to overcome the learning curve.

To summarise, existing visualisation solutions, regardless of the graph grammar they follow, generate proper graphs, but they are not necessarily meaningful. By introducing visualisation schema

patterns to the application, more information can be utilised to create relevant and meaningful visualisations that align with the user's data.

3.3 Visualisation Libraries

3.3.1 ggplot2

ggplot2 [12] is a popular data visualisation package in the R programming language. It is built upon the principle of a layered grammar of graphics[13], which is an extension of the traditional grammar of graphics. This package enables users to build complex and layered visualisations by combining components such as data, aesthetics, geometric objects, scales and facets. In ggplot2, visualisations are created by adding layers, where each layer represents different components, which makes it easy to modify and customise visualisations.

However, it is important to note that ggplot2 is primarily designed for static visualisations and does not offer interactivity features. Additionally, compatibility can be an issue. While ggplot2 is widely used in the R ecosystem, it may not always work well with other programming languages or tools.

3.3.2 D3

D3 (Data-Driven Documents) [14] is a JavaScript library for creating interactive data visualisation using web standards. With D3, users have complete control over the visual appearance and behaviour of their visualizations, because it uses the DOM (Document Object Model) to manipulate and transform elements on a web page. In addition, D3 provides a large set of tools for data transformation and manipulation. Operations such as filtering and sorting can be performed to prepare for the visualisation stage. D3 also provides animation and interactive behaviour to the visualisation, enabling users to better explore and interact with the data.

On the other hand, the trade-off for the flexibility and advanced feature that D3 provides is the steep learning curve, particularly for users who are new to JavaScript and web development.

3.3.3 Vega

Vega [15] is a visualisation grammar and open-source framework for creating expressive data visualisations. It uses D3 under the hood for various tasks such as handling data binding and rendering SVG graphics. While D3 primarily focuses on low-level data manipulation and DOM manipulation, Vega provides a higher-level abstractions for compositions of visual elements.

Similarly, **Vega-Lite**[16] is a high-level grammar of interactive graphics that combines the principles of the traditional grammar of graphics with interactive features. Vega and Vega-Lite are often used together with D3 to enhance the efficiency of visualisation development.

Chapter 4

Design

4.1 Overview

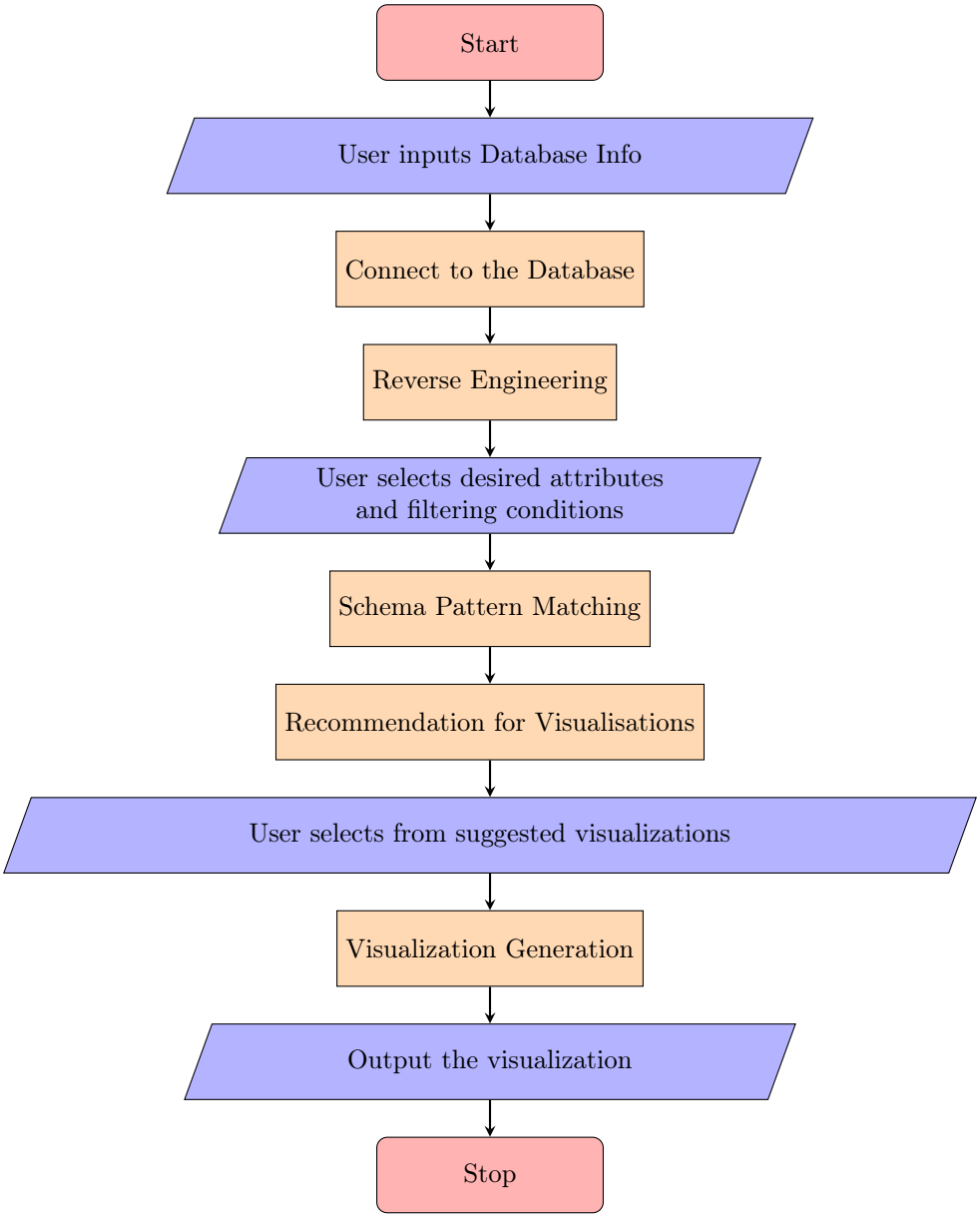


Figure 4.1: Flowchart of the process from user inputs to a visualisation

In figure 4.1, the flowchart illustrates the step-by-step process of generating a visualisation from the beginning, and how a user interacts with the application. In this chapter, our focus will primarily be on the processes under the hood, represented by the yellow rectangles in the figure. The objective is to guide users through the whole visualisation process, with the help of schema information of the data.

4.2 Connect to the Database

This project primarily focuses on visualisation on data from relational databases, because the relational schema provides valuable information which is processed and utilised in later stages of the project. On the other hand, data in file formats offer limited insights into the underlying schema structure.

To ensure wide applicability and usability, it is important to support popular relational database management systems such as MySQL, PostgreSQL, Oracle, and others. This enables users to seamlessly connect and visualise data from various databases commonly used in different domains.

4.2.1 Ethical issues

Ethical issues are important in any project that involves database connections. This project focuses on visualising data from relational databases. It naturally entails the collection of certain personal data, such as account information related to the target database. However, it's important to emphasise that although our application has access to the data, all operations in the database are in a read-only mode.

To maintain data integrity and privacy, any potential data refactoring is performed locally by creating a separate copy, thereby guaranteeing the preservation of the original dataset. The collected personal data is solely used for user authentication purposes, ensuring secure access to the database. We emphasise that no personal information is stored in any format within this project, and all collected data is deleted immediately when no longer required.

Our commitment is to provide users with a trusted and responsible data visualisation experience, upholding both legal and ethical standards throughout the project. We prioritise the protection of user privacy and adhere to applicable data protection regulations, ensuring that personal data is handled properly.

4.3 Reverse Engineering

This project is delivered at the ER level instead of the relational level, hence we need this step to perform the transformation from a relational schema to an ER schema. The resulting ER schema information is then stored for further analysis. To ensure accurate pattern matching with the visualisation schema patterns in figure 2.2, a comprehensive reverse engineering algorithm is required. This algorithm should be able to identify entities with relationships and their cardinalities. It should distinguish between strong and weak entities. Additionally, this algorithm should be able to identify reflexive relationships. To showcase the level of correctness required from the reverse engineering algorithm, we use the Mondial database [1] as an example.

Figure 4.2 displays the relational database schema in PostgreSQL for a fragment of the Mondial database. Based on the metadata this schema provides, it is expected that the reverse engineering algorithm successfully identifies the **province** table as a weak entity with respect to **country** and the **city** table as a weak entity with respect to **province**. It should also recognise the one-many relationship between **airport** and **city**. In addition, the **encompasses** table should be recognised as a many-many relationship between **country** and **continent**, and **borders** table recognised as a reflexive relationship. Finally, **economy** should be recognised as a subset of **country**. These expectations are reflected as an ER schema illustrated in figure 1.1.

```
CREATE TABLE country (
  name VARCHAR(32) NOT NULL CONSTRAINT country_name_ck UNIQUE,
  code VARCHAR(4) CONSTRAINT country_pk PRIMARY KEY,
  area NUMERIC(10,2) NOT NULL CONSTRAINT country_area_range CHECK (area >=
      0));
```

(a) relational schema of the `country` table

```
CREATE TABLE province (
 name VARCHAR(48) CONSTRAINT province_name NOT NULL,
 country  VARCHAR(4) CONSTRAINT province_country NOT NULL,
 population INTEGER CONSTRAINT province_population_range
            CHECK (population >= 0),
 area INTEGER CONSTRAINT province_area_range CHECK (area >= 0),
 CONSTRAINT province_pk PRIMARY KEY (name, country),
 CONSTRAINT province_country_we FOREIGN KEY (country) REFERENCES country);
```

(b) relational schema of the `province` table

```
CREATE TABLE city (
 name VARCHAR(48) NOT NULL,
 country VARCHAR(4) NOT NULL,
 province VARCHAR(48) NOT NULL,
 population DECIMAL NULL,
 CONSTRAINT city_population_range CHECK (population>=0),
 elevation INTEGER,
 CONSTRAINT city_pk PRIMARY KEY (name, province, country),
 CONSTRAINT city_province_we FOREIGN KEY (province, country) REFERENCES
     province);
```

(c) relational schema of the **city** table

```
CREATE TABLE airport (
 iata_code VARCHAR(3) PRIMARY KEY,
 name VARCHAR(100),
 country VARCHAR(4),
 city VARCHAR(48),
 province VARCHAR(48),
 latitude NUMERIC(5,2) NOT NULL
 CONSTRAINT airport_latitude_range CHECK (latitude BETWEEN -90 AND 90),
 longitude NUMERIC(5,2) NOT NULL
 CONSTRAINT airport_longtitude_range CHECK (longitude BETWEEN -180 AND 180),
 elevation INTEGER,
 gmt_offset INTEGER,
 CONSTRAINT airport_city_fk
 FOREIGN KEY (city, province, country) REFERENCES city);
```

(d) relational schema of the **airport** table

```
CREATE TABLE encompasses (
  country VARCHAR(4) NOT NULL
  CONSTRAINT encompasses_country_fk REFERENCES country,
  continent VARCHAR(20) NOT NULL
  CONSTRAINT encompasses_continent_fk REFERENCES continent,
  percentage NUMERIC(10,2) NOT NULL
  CONSTRAINT encompasses_percentage_range CHECK (percentage BETWEEN 0 AND
      100),
  CONSTRAINT encompasses_pk PRIMARY KEY (country,continent));
```

(e) relational schema of the `encompasses` table

Figure 4.2: Relational database schema of a fragment of the Mondial database [1]

```
CREATE TABLE continent (
 name VARCHAR(20) CONSTRAINT continent_pk PRIMARY KEY, area NUMERIC
     (10,2) NOT NULL);
```

(f) relational schema of the `continent` table

```
CREATE TABLE borders (
   country1 VARCHAR(4) CONSTRAINT border_country_a_fk REFERENCES country
       ,
   country2 VARCHAR(4) CONSTRAINT border_country_b_fk REFERENCES country
       ,
   length NUMERIC(10,2) NOT NULL CONSTRAINT border_length CHECK (length
       > 0),
  CONSTRAINT border_pk PRIMARY KEY (country1,country2));
```

(g) relational schema of the `borders` table

```
CREATE TABLE economy (
 country VARCHAR(4) CONSTRAINT economy_pk PRIMARY KEY,
 gdp INT CONSTRAINT economy_gdp CHECK (gdp>=0),
 inflation NUMERIC(5,2),
 unemployment NUMERIC(5,2),
CONSTRAINT economy_isa FOREIGN KEY (country) REFERENCES country);
```

(h) relational schema of the `economy` table

Figure 4.2: Relational database schema of a fragment of the Mondial database [1] (cont.)

4.4 Schema Pattern Matching

4.4.1 Pattern Characteristics

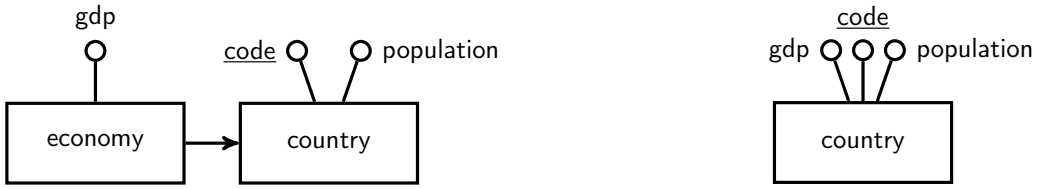
Once the user has selected the attributes to be visualised, the next step is to map the selection to one of the five visualisation schema patterns shown in figure 2.2. Utilising the ER schema acquired from the previous step, the pattern-matching algorithm examines the characteristics of each pattern to distinguish between them.

- The Basic Entity: basic entities are fundamental building blocks of ER models with their own attributes. They do not depend on any other entities for their existence.
- The Weak Entity: a weak entity refers to an entity that cannot be uniquely identified by its own attributes alone. Instead, it relies on a related strong entity to establish its identity. In a weak-entity relationship, the weak entity is considered the dependent or the child entity, and the strong entity is the parent entity.
- The One-Many Relationship: one-many relationships describe a relationship between two entities where one entity can have multiple instances associated with it, while the other entity has only one instance. Note that weak entities also have one-many relationships with their strong entities. However, we do not map weak entities to the One-Many Relationship pattern.
- The Many-Many Relationship: many-many relationships describe a relationship between two entities where each entity can have multiple instances associated with multiple instances of the other entity.
- The Reflexive Relationship: a reflexive relationship refers to a relationship that exists between instances of the same entity. In other words, it represents a connection or association within a single entity. Note that in the patterns, the Reflexive Relationship is many-many.

It is important to note that in the Many-Many Relationship and Reflexive Relationship models, the user-selected attributes are from the table representing the relationship itself rather than from any individual entity as shown in figure 2.2.

4.4.2 Handling Subset Entities

Subset entities are derived or specialised from another entity in the ER model. They represent a subset of the attributes and relationships of the parent entity. When visualising subset entities, they can be treated as basic entities, but they should inherit the key attribute from the parent entity. An example is shown in figure 4.3.



(a) example of a subset

(b) mapped to a basic entity

Figure 4.3: Transforming a subset

4.5 Recommendation for Visualisations

There are various visualisations suitable for each model, as shown in table 2.1. Once we have successfully matched the user selection to one of the visualisation schema patterns, we proceed with a detailed analysis that takes into account the number of attributes and their types.

In the context of visualisation, graphical elements are classified as **marks** (such as points and lines) or **channels** (including colour, shape, texture and more). Each instance of an entity in the database can be represented by one or more graphical elements. The attribute value of an entity or a relationship is associated with a dimension of visualisation. By taking a similar definition in [6], attribute values can be classified into two major types of dimensions.

- **Discrete**: discrete dimensions consist of a relatively small number of distinct values, which may or may not have a natural ordering. These dimensions are utilised to choose a mark or to vary a channel of a mark in a visualisation. For instance, the **code** attribute of the **country** entity is an example of discrete dimensions without natural ordering.
- **Scalar**: scalar dimensions have a significant number of distinct values with a natural numeric ordering, such as integers, floats, or dates. These dimensions are represented by a channel associated with a mark in a visualisation. An example would be the **area** attribute of the **country** entity.

Visualisations suitable for each schema pattern are explained in detail below. Key cardinalities are used to indicate the lower and upper bounds of entries within an entity, ensuring optimal readability of the visualisation.

4.5.1 Basic Entity Visualisations

- **Bar Chart**: each bar in a bar chart represents an individual instance of the entity, and the length of each bar is determined by a scalar attribute.
- **Pie Chart**: each section of a pie chart represents an individual instance of the entity, and the size of each section is determined by a scalar attribute, proportionate to the total value.
- **Calendar Chart**: a calendar chart represents instances of the entity based on a temporal scalar attribute. The intensity of colour in the chart can represent an additional scalar attribute and a colour legend should be provided alongside the calendar chart for clear interpretation.
- **Scatter Diagram**: in a scatter diagram, each point represents an individual instance of the entity. The horizontal and vertical axes represent two scalar attributes. Optionally, the colour of each point can represent an additional discrete attribute.

- **Bubble Chart**: similar to a scatter diagram, each bubble in a bubble chart represents an individual instance of the entity. The horizontal and vertical axes represent two scalar attributes. Additionally, a third dimension represents the size of the bubble. The colour of each bubble can represent an optional discrete attribute.
- **Choropleth Map**: in a choropleth map, each region represents an individual instance of the entity. The key attribute of the entity should be interpretable as a geographical region, which should be discrete (e.g. country names/codes). The intensity of colour in each region represents a scalar attribute associated with the entity.
- **Word Cloud**: in word clouds, the key attribute of the entity should be interpretable as a discrete lexical domain. The size of each word represents a scalar attribute associated with the entity.

Visualistaion	**Entity key cardinality**	**Mandotory scalar attributes**	**Optional discrete attributes**	**Special requirement**
Bar Chart	1..100	1	-	-
Pie Chart	1..20	1	-	-
Calendar Chart	1..*	1	-	scalar attribute should be temporal
Scatter Diagram	1..*	2	1	-
Bubble Chart	1..*	3	1	-
Choropleth Map	1..*	1	-	key attribute should be geographical
Word Cloud	1..*	1	1	key attribute should be lexical

Table 4.1: Basic Entity Visualisations

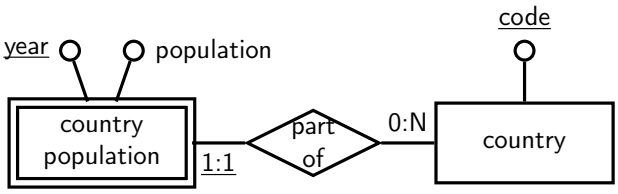
For example, the **airport** entity in figure 1.1 can be matched to a Basic Entity, with the key attribute **iata_code**. A scatter diagram can be produced with **longitude** against **latitude**, shown in figure A.4. More examples of Basic Entity visualisations can be found in the Appendix section:

Visualisation	**Example**
Bar Chart	A.1
Pie Chart	A.2
Calendar Chart	A.3
Scatter Diagram	A.4 and A.5
Bubble Chart	A.6 and A.7
Choropleth Map	A.8
Word Cloud	A.9

4.5.2 Weak Entity Visualisations

For certain visualisations under the Weak Entity pattern, we require data to be **complete**. This means that the set of values that appears for the key attribute of the child entity must be consistent across all values of the key attribute of the parent entity. For example, consider the weak entity **country_population** which records the history of populations for **country**. The data in **country_population** is complete if the years of the population recorded for each country remain consistent across the entire dataset. This example is illustrated in figure 4.4.

- **Line Chart**: in a line chart each individual line represents an individual instance of the parent entity. The horizontal axis represents the key attribute of the child entity, which should be scalar. The vertical axis represents a scalar attribute associated with the child entity.



(a) ER schema of **country_population** and **country** in Mondial database

country	year	population
J	2000	125714674
J	2005	127767994
J	2010	128057352
MEX	2000	97483412
MEX	2005	103263388
MEX	2010	112336538
KIR	2000	84494
KIR	2005	92533
KIR	2010	103058

(b) values of **year** are complete with respect to **country_population**

Figure 4.4: An example of complete dataset

- **Stacked Bar Chart**: in a stacked bar chart, the key attribute of the parent entity is represented by an individual bar, and within each bar, individual sections represent the key attribute of the child entity. The length of each section is determined by a scalar attribute associated with the child entity. In stacked bar charts, dataset completeness is required.
- **Grouped Bar Chart**: similar to stacked bar charts, except that the key attribute of the parent entity is represented by a grouped bar chart consisting of multiple bars instead of a single bar with multiple sections.
- **Spider Chart**: in a spider chart, the key attribute of the parent entity is represented by individual rings, while each spoke represents the key attribute of the child entity. The length of individual spokes reflects a scalar attribute associated with the weak entity. In spider charts, dataset completeness is required.

Visualistaion	Parent key cardinality	Child key cardinality	Mandotory scalar attributes	Optional discrete attributes	Dataset completeness
Line Chart	1..20	1..*	1	-	no
Stacked Bar Chart	1..20	1..20	1	-	yes
Grouped Bar Chart	1..20	1..20	1	-	no
Spider Chart	3..20	1..20	1	-	yes

Table 4.2: Weak Entity Visualisations

An example of a matched Weak Entity in figure 1.1 is **province**. A grouped bar chart can be produced to compare the **populations** of provinces between countries. Figure A.12 illustrates a grouped bar chart comparing the populations of provinces in France and Spain. More examples of Weak Entity visualisations can be found in the Appendix section:

Visualisation	Example
Line Chat	A.10
Stacked Bar Chart	A.11
Grouped Bar Chart	A.12
Spider Chart	A.13

4.5.3 One-Many Relationship Visualisations

- **Tree Map**: in a tree map, each instance of the parent entity is represented by a large rectangle that is divided into smaller rectangles to represent instances of the child entity. The size of smaller rectangles is proportional to the value of a scalar attribute associated with the child entity.

- **Hierarchy Tree**: instances of the parent entity are visualised as nodes with connected lines representing the hierarchical relationship to instances of the child entity. Attributes are not mandatory for this visualisation as its primary focus is to convey hierarchical information. An optional discrete attribute can be used to colour nodes.
- **Circle Packing**: similar to tree maps, except that in circle packing, circles are used instead of rectangles. An optional discrete attribute can be used to colour circles.
- **Sunburst**: in a sunburst graph, each instance of the parent entity is represented by a sector of the inner circle, and instances of the child entity are represented by sectors of the outer circle. The size of each sector represents the proportion of the corresponding scalar attribute.

Visualistaion	**Parent key cardinality**	**Child key cardinality (per parent key)**	**Mandotory scalar attributes**	**Optional discrete attributes**
Tree Map	1..100	1..100	1	-
Hierarchy Tree	1..100	1..100	-	1
Circle Packing	1..100	1..100	1	1
Sunburst	1..100	1..100	1	-

Table 4.3: One-Many Relationship Visualisations

In figure 1.1 the **airport** is an example of having a one-many relationship with **city**. To visually represent this hierarchical relationship, a tree map can be generated, showcasing the distribution of airports across different cities, with the **elevaltion** attribute. This is shown in figure A.14 for the cities of New York, Moskva and London. More examples of One-Many Relationship visualisations can be found in the Appendix section:

Visualisation	**Example**
Tree Map	A.14
Hierarchy Tree	A.16 and A.17
Circle Packing	A.18 and A.19
Sunburst	A.15

4.5.4 Many-Many Relationship Visualisations

- **Sankey Diagram**: in a sankey diagram, elements on the left-hand side represent instances of one entity, while elements on the right-hand side represent instances of the other entity. The width of the flow connecting these elements on different sides represents a scalar dimension associated with the relationship.
- **Network Chart**: in a network chart, instances of entities are represented by nodes, and the connected edges depict the network of relationships between the entities. Similar to hierarchy trees, network charts primarily focus on showcasing relationships between entities, and therefore attributes are not mandatory for this visualisation.
- **Chord Diagram**: in a chord diagram, instances of entities are represented by arcs or segments arranged around a central circle. The width of chords that link pairs of arcs represents a scalar dimension associated with the relationship.
- **Heatmap**: in a heatmap, instances of entities are represented by cells of a matrix. The intensity of the colours of the cells represents a scalar dimension associated with the relationship, and a colour legend should be provided alongside the heatmap for clear interpretation.

In figure 1.1, the relationship between **country** and **continent** is mapped to the Many-Many Relationship pattern. Based on the attribute **percent** from the relationship **encompasses**, we can effectively illustrate this relationship using a sankey diagram. Figure A.20 shows a sankey of all countries whose areas span across more than one continent.

4.5.5 Reflexive Relationship Visualisations

As mentioned previously, visualisations that correspond to the Reflexive Relationship case are a subset of those suitable for the Many-Many Relationship.

Sankey diagrams are not appropriate for visualising reflexive relationship data because they are primarily designed to represent directional information between different entities. Sankey diagrams are commonly used to illustrate the flow from a resource to a target. In contrast, reflexive relationships involve connections between instances of the same entity. Therefore, it is not recommended to use sankey diagrams for visualising reflexive relationships.

Chord diagrams, on the other hand, are particularly well-suited for visualising reflexive relationship data. In chord diagrams, the ribbons connecting the arcs on the circle represent the relationships between instances of the same entity, highlighting the scalar attribute associated with the relationship and provides a clear representation of the connections within the entity.

Heatmaps and network charts are generally suitable for visualising many-many relationships, whether reflexive or not. They both provide informative representations of the connections between entities.

Visualistaion	**Entities' key cardinality**	**Mandotory scalar attributes**	**Optional discrete attributes**	**Reflexive relationship compatible**
Sankey Diagram	1..20	1	-	no
Network Chart	1..1000	-	1	yes
Chord Diagram	1..100	1	-	yes
Heatmap	1..100	1	-	yes

Table 4.4: Many-Many Relationship and Reflexive Relationship Visualisations

In figure 1.1, **country** has a reflexive relationship with itself, through **borders**. A chord diagram can be produced to visualise this relationship, with the attribute **length**. Figure A.22 illustrates the **borders** relationship using a chord diagram, where the border length is greater than 2500 kilometres.

More examples of Many-Many Relationship and Reflexive Relationship visualisations can be found in the Appendix section:

Visualisation	**Example**
Sankey Diagram	A.20
Network Chart	A.21
Chord Diagram	A.22
Heatmap	A.23

4.6 Visualisation Generation

Once the user has selected a visualisation from the suggested list, the next step is to generate the corresponding visualisation. There are many packages available that allow us to generate visualisations directly. As developers, it is important to choose a package that offers sufficient flexibility to accommodate optional attributes in the visualisation.

Chapter 5

Technical Implementation

In this section, we are going to talk about the detailed implementation of this project. The primary focus of this project is to guide users in selecting the appropriate visualisations for their data. Hence we are going to keep the user interface as clean and intuitive as possible, giving users visual aid in using this application.

This project is designed as a web application, offering several advantages. From the user's standpoint, web applications provide easy access without the need for installations. They are platform-independent, allowing users to access them from any device with a browser and an internet connection. From the developer's point of view, web applications typically offer quicker development cycles, benefiting from the personal developer experience gained through previous projects that focused on web development. On the other hand, desktop applications typically require more platform-specific knowledge and are restricted to particular operating systems.

5.1 Framework choices

There are several popular web application development frameworks to choose from, and for this project, we have selected the Java-based Spring Boot framework. Spring Boot offers advantages for web application development, including simplified configuration and a range of developer tools, enabling developers to focus more on internal logic rather than infrastructure. Additionally, it is well-suited for building microservices architecture. Standard web technologies like HTML and JavaScript can be used for front-end development with Spring Boot, including popular JavaScript packages.

5.2 Connect to the Database

To start the visualisation process, the user is required to provide details of the database they wish to visualise. These details include the essential information such as the **database type**, **host address**, **port number**, **database name**, **username** and **password**. Database types supported in this project include PostgreSQL, MySQL, Oracle, SQL Server, DB2 and H2. The provided information will be collected in JSON format, and transmitted to the back-end for further processing. An example of the required database information is illustrated in figure 5.1.

```
{
    "dbType": "postgresql",
    "host": "localhost",
    "port": "5432",
    "dbName": "mondial",
    "username": "yc1319",
    "password": ""
}
```

Figure 5.1: Required database information in JSON format

The database connection and interaction are handled by JDBC [17], which is a widely used Java API for database manipulation. It provides an effective way to interact with databases and retrieve database metadata easily.

5.3 Reverse Engineering

There are several approaches to accomplish the reverse engineering process. One option is to implement algorithms described in research papers such as [8, 9, 2]. Alternatively, existing tools can be utilised to reduce development time. The AmazingER [10] is employed in this project to generate ER schemas from the relational databases. The source code of AmazingER has been made available to me with permission. This provides great flexibility to modify the code and import the customised version as a library into this project. This allows for the incorporation of specific features tailored to the project's requirements.

For instance, the original version of AmazingER did not support multiple relationship edges between the same relationship and entity objects, which is essential for parsing reflexive relationships. To address this limitation, necessary modifications were made, enabling proper parsing of reflexive relationships. The weak entity issue mentioned earlier was tackled by a colleague, who afterwards shared his solution with me. The modified version can now successfully parse all tables in the Mondial database shown in 4.2.

5.3.1 Handling User Selections

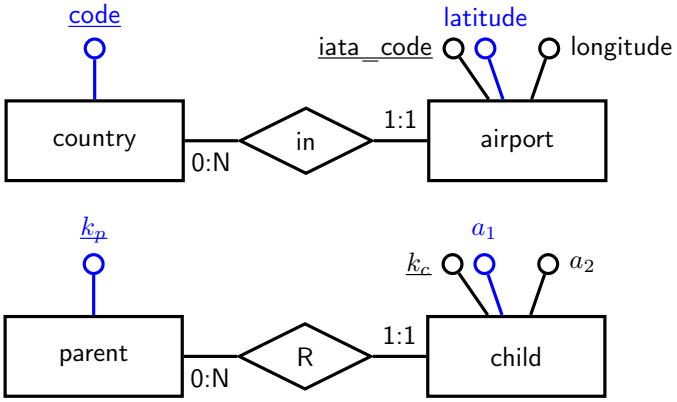


Figure 5.2: one-many example of valid user selection

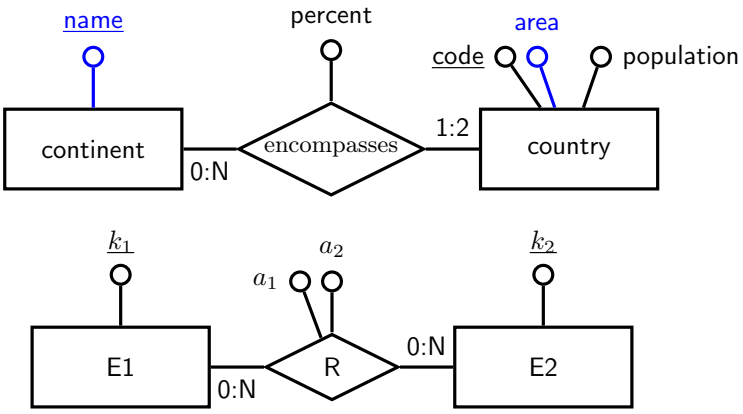


Figure 5.3: example of invalid user selection

Once the relational database has been parsed into an ER schema, entities and relationships from the resulting ER schema are displayed on a page, with their attributes listed as options. Users have

the freedom to choose attributes from at most one entity or relationship. However, it's important to note that the key attribute, if present, cannot be selected unless it belongs to the entity that has a relationship to the entity from which attributes are being selected. This constraint is naturally implied by visualisation schema patterns. An example is shown in figure 5.2, where a user has selected the attribute **latitude** from the child entity **airport** in a one-many relationship. The user has the option to select the key attribute **code** from the parent entity **country** or disregard it. If it's disregarded, this key attribute is processed and utilized automatically in the visualisation process, minimising the need for manual selection and enhancing the efficiency of the visualisation. Conversely, an example of an invalid selection is presented in figure 5.3. In the context of a many-many relationship, the attributes are from the relationship itself, rather than from any individual entity, as dictated by the model.

5.4 Pattern Matching to models

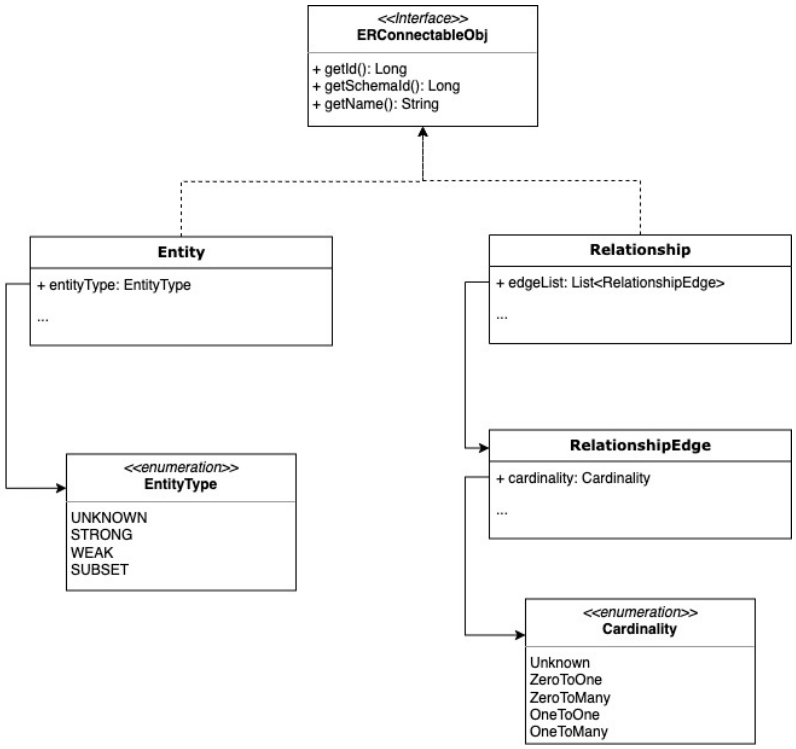


Figure 5.4: Functionalities of AmazingER used in pattern matching phase

The utilization of specific functionalities offered by the AmazingER is shown in figure 5.4, which facilitates the implementation of pattern matching. In cases where the user makes an invalid selection such as the example in figure 5.3, or encounters an ER model that is not one of the visualisation schema patterns, we handle such scenarios by classifying them as the **unknown** model type.

- **Basic Entity**: basic entities are the fundamental building blocks of ER models, if the user selected attributes from a table which is an entity, and this entity is not a weak entity, we classify it to the Basic Entity.

- **Weak Entity**: A weak entity is an entity that does not have a unique identifier on its own, but it's dependent on another entity. We identify it by checking if the EntityType is WEAK.

- **One-Many-Relationship**: A relationship that an entity on one side is related to multiple entities on the other side. We identify it by checking whether there exists two RelationshipEdge, one with cardinality OneToMany (1:N) or ZeroToMany (0:N), the other with cardinality OneToOne (1:1) or ZeroToOne (0:1) and connects the chosen entity table.

- **Many-Many-Relationship**: A relationship that multiple instances of an entity on one side are related to multiple instances of an entity on the other side. We identify it by checking whether there exists two RelationshipEdge, both with cardinality OneToMany (1:N) or ZeroToMany (0:N). They need to connect different entities.
- **Reflexive Relationship**: A reflexive relationship occurs when an entity is related to itself. This is a special case of the Many-Many Relationship. We identify it by checking whether there exists two different RelationshipEdge both with cardinality OneToMany (1:N) or ZeroToMany (0:N), connecting the same entity.

It is important to mention that if the user selection is pattern matched to the One-Many relationship model, it also matches the Basic Entity model and we should provide relevant visualisations under the Basic Entity pattern as options. This is because basic entities naturally exist within one-many relationships. However, for the Many-Many Relationship and Reflexive Relationship cases, we do not provide Basic Entity visualizations as options. In these scenarios, the selected attributes should belong to the relationship table rather than the entity tables.

5.5 Retrieving Dataset from the Database

The dataset we need to retrieve is based on the user selection and JDBC is used to retrieve target datasets from the user's database using SQL queries. When no filters are applied, the SQL queries are relatively simple, involving only the selection of attributes from a specific table. Table joins are unnecessary since all required data of the attributes can be obtained from a single table. In cases where the data of a key attribute from another table is needed, we can use the foreign key information.

5.5.1 Filtering

Users can optionally apply filters to their data. Filtering options are from tables that have a foreign key relation selected tables. These filtering options encompass attributes from entities and relationships, which are also classified into discrete and scalar types. When dealing with discrete attributes as filtering options, concrete data is presented as checkboxes for selection. Scalar attributes are facilitated by a range slider, allowing the user to define the desired minimum and maximum values for the attribute. There is no limit to how many filter conditions are applied.

When filters are applied, we perform INNER JOIN operations with those related tables, and the filtering conditions are specified in the WHERE clause of the SQL queries. AND is used to connect each filter condition. In a discrete filter condition, each distinct discrete value is connected with an OR, and in a scalar filter condition, values are bounded by >= and <= signs.

An example SQL query of filtering countries in Europe and Asia, with a population between 20 million and 50 million is:
SELECT country.code, country.population FROM country INNER JOIN encompasses ON country.code=encompasses.country INNER JOIN continent ON encompasses.continent=continent.name WHERE (continent.name='Europe' OR continent.name='Asia') AND (country.population>= 20000000 AND country.population<=50000000)

5.6 Data Type Classfication

In order for the dataset to fit visualisations, we need to classify SQL data types into data types to meet the requirements. In this project, we have classified data types into three main categories: Numerical, Temporal, and Lexical.

- **Numerical**: data values that have a relatively large and continuous range, and have a natural numeric ordering. Examples of SQL data types that fall into this category are INTEGER, DOUBLE, NUMERIC etc.
- **Temporal**: data values that also have a natural numeric ordering, but are measured in time. SQL data types such as DATE and TIMESTAMP are example of Temporal data types.

- **Lexical**: data values that represent discrete string values. Examples of SQL data types are CHAR, VARCHAR, LONGVARCHAR etc.

It is important to note that the scalar dimension is maintained as Numerical ∪ Temporal data types. However, we are not able to completely capture the definition of the discrete dimension by SQL data types. While attribute values related to strings are commonly associated with the discrete dimension, there are cases where attributes such as **gmt_offset** in the **airport** entity have a numerical SQL data type but still belong to the discrete dimension. In this project, we treat Discrete ≈ Lexical, and we recognise that our definition for discrete attributes is not perfect.

For choropleth maps, we don't need an additional geographical data type. Instead, the discrete data that matches a map of geographical names (from Natural Earth [18]) or codes will be presented in the choropleth maps. However, if the data does not align with a geographical map, it is not recommended to use this specific visualisation.

5.7 Visualisations

Despite the sharp learning curve, D3 is a great library for creating visualisations, and it is used in this project. It gives large room to adjust each visualisation for optional attributes.

To better aid users to explore data in certain visualisations, it is necessary to offer a certain level of customisation. For instance, when multiple scalar attributes are chosen, to create a scatter diagram or a bubble chart, users should be free to change the association between attributes and dimensions. For example, a bubble chart is generated to visualise the **economic** relationship of countries between **agriculture**, **service** and **industry**, users should be able to adjust the representation of the a-axis, y-axis and the size of the bubble to suit their needs.

It is important to note that in order to generate an accurate chord diagram, which is under the Reflexive Relationship pattern, the dataset provided has to be reflexive as well. An example dataset for a chord diagram is about the proportions of survey respondents who currently own a phone from a specific brand, while previously owning a phone from a different brand. This example is illustrated in figure 5.5 below. An inappropriate example dataset would be the **borders** table from the Mondial database. It's our responsibility, to convert the non-reflexive dataset to a reflexive one, before outputting a chord diagram. This is shown in figure 5.6.

previous	current	proportion
Apple	Samsung	1.2%
Samsung	Apple	2.3%
Apple	Huawei	0.5%
Huawei	Apple	1.5%

Figure 5.5: reflexive dataset example for a chord diagram

country1	country2	length
CDN	USA	8893.00
RA	RCH	5150.00

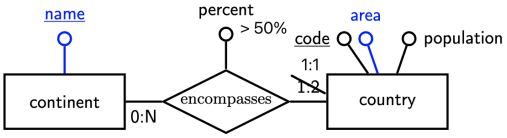
country1	country2	length
CDN	USA	8893.00
USA	CND	8893.00
RA	RCH	5150.00
RCH	RA	5150.00

Figure 5.6: converting an inappropriate dataset to output a chord diagram

When considering key cardinality constraints, let's take basic bar charts as an example. In a bar chart, it is essential to limit the number of entities to avoid a cluttered appearance. To address this, we have implemented a zoomable functionality for the bar chart. If the number of entities exceeds the upper cardinality limit of 100, the chart automatically zooms to a scale that displays the first 100 entities in alphabetical order (options of ascending and descending order are also provided). However, if the user desires to explore more entities, they can easily zoom out to view a broader range of data. This approach ensures that the bar chart remains manageable and allows users to delve deeper into the data when necessary.

5.8 Handling Unknown case

The example illustrated in figure 5.3 is categorised as an unknown pattern, indicating that it does not align with any predefined visualisations options based on available patterns. However, as illustrated in figure 2.4, certain filter conditions can be applied to the relationship between **country** and **continent** from many-many to one-many. In this case, we need to check if the retrieved dataset from is one-many instead of checking the actual schema because filtering does not change the schema structure. If the dataset satisfies the condition of a one-many relationship, we suggest those visualisations under the One-Many Relationship pattern. Figure 5.7 illustrates this by using a tree map, and we can see that Russia only appeared once, and this is a perfect tree map.



Visualistaion	Parent key cardinality	Child key cardinality (per parent key)	Mandotory scalar attributes	Optional discrete attributes
Tree Map	1..100	1..100	1	1
Hierarchy Tree	1..100	1..100	-	1
Circle Packing	1..100	1..100	1	1

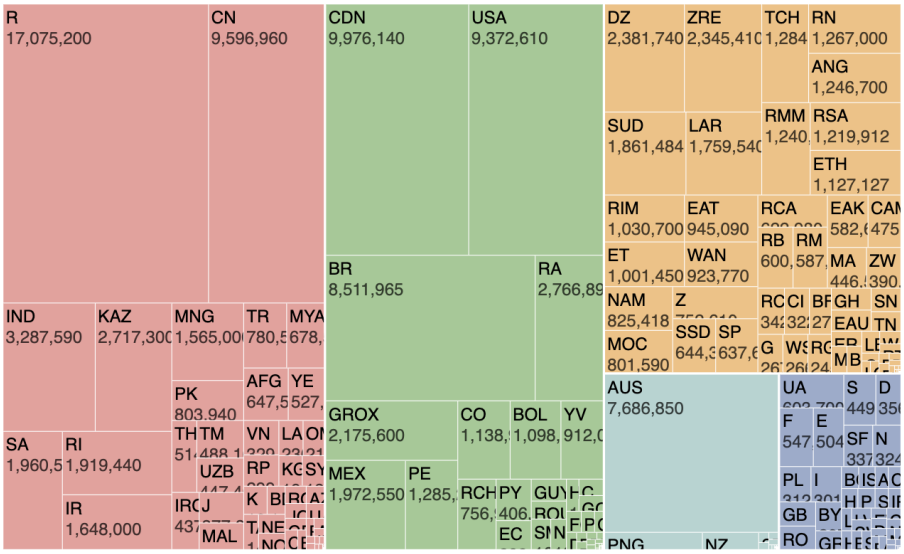


Figure 5.7

Chapter 6

Evaluation

Throughout this project, our main focus has been the development of a data visualisation tool that guides users to find the appropriate visual encodings for their data. Evaluating the success of an application-based project can be challenging, as it is difficult to define success in absolute terms. However, to assess the effectiveness of our tool, we have conducted a series of evaluation that considers various criteria.

In this chapter, we present the results of our evaluation, including aspects such as functionality, usability, and the ability to support data exploration. Additionally, we compare our tool with different visualisation solutions to better understand its unique strengths and areas for improvement.

6.1 Functionality

Task	Capability
Create a bar chart showing top 3 airports in MEX which has the highest elevation	Yes
Create a relevant visualisation showing all countries which have a gmt_offset of 2	Yes
Create a sankey diagram showing all countries which have less than 100 encompass value in Europe	Yes
Create a relevant visualisation of all city's which have a population of greater than 10,000,000 over any time frame	Yes
Create a relevant visualisation that shows the countries which are connected with border lengths of >3000	Yes
Create a scatter diagram and a bubble chart that shows Economy data with lowest gdp with unemployment and industry	No
Create a line chart and a grouped bar chart that shows weak-entity relationship of province_population	Yes
Create a hierarchy tree and a tree map that shows Ethnic_group to country (ordered by key 1)	No
Create a sankey diagram showing all of the 5 countries whose area crosses between continents	Yes
Create a scatter diagram illustrating the relation between industry and service in the economies of different countries	Yes
Create a grouped bar chart showing the population of France, Spain, and Italy for the years 1960 to 2015	Yes
Create a tree map visualizing the elevation of airports, with a minimum elevation of 1998	Yes
Create a sankey diagram illustrating the relation between countries and continents, in Europe and Asia	Yes
Create a chord diagram about borders between countries where the border length is greater than 5000	Yes

Together with two colleagues, we came up with a list of tasks based on the Mondial database to assess the functionality and capabilities of our application. Our application offers users a wide range of visualisations to accommodate various scenarios. We have implemented robust filtering functionality that handles both discrete and scalar attributes, enabling users to refine their data based on specific needs.

However, because of the challenge we encountered related to the classification of attributes in our application, the **gmt_offset** is mistakenly categorised as a scalar attribute instead of a discrete attribute. This problem arises in some particular visualisation tasks, such as the second task where the filtering condition is **gmt_offset=2**. Although users can still apply this filter by setting conditions **gmt_offset>=2** and **gmt_offset<=2**, users might find it counter-intuitive, and this approach is clearly suboptimal. We acknowledge that the flaw in the definition of attribute types may have affected the usability and accuracy of the tool in particular scenarios and we recognise the need for a more refined approach to attribute type classification.

One limitation of our application is the absence of aggregation functionality. As a result, the sixth task cannot be performed within our application because we lack the capability to sort data and rank it based on the **lowest gdp** criterion. However, the first task, which involves determining the **top 3 highest elevations** can still be accomplished because of the unique characteristics of the bar chart we offer. We recognise that incorporating additional aggregation functionalities would enhance user interaction with the data and visualisation, facilitating better data exploration. Aggregation features would also cater to the specific needs of users, allowing for more comprehensive analysis and insights.

The eighth task is about the connection between weak-entity relationships and one-many relationships. In Mondial database, **ethnic_group** is a weak entity connected to the **country** entity, as shown in figure 6.1. While weak entities are typically not ideally suited for hierarchical graph visualisations, they can still be considered as one-many relationships with their parent entities. This characteristic allows for the application of One-Many Relationship visualisations to represent these relationships. Visualisations such as tree maps and circle packing offer a broader range of possibilities to showcase the one-many nature underlying weak entities.

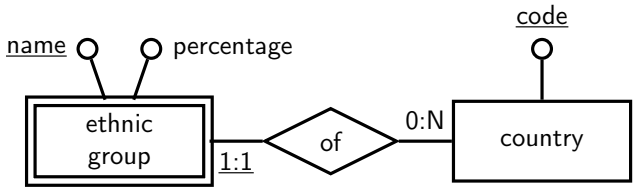


Figure 6.1

6.2 Usability

User feedback is a crucial component in assessing the overall performance of a software application. In our evaluation process, we provided users with five specific tasks to complete using our data visualisation tool as well as the widely-used Tableau.
Overall, users found the visualization process in our tool to be more intuitive compared to Tableau. Using Tableau, users fail to complete the fifth task of creating a sankey diagram in a reasonable time without help from the internet. In our application, especially for tasks related to the basic entity and weak entity visualizations (the first three tasks in the table) users spent significantly less time generating the desired visualisations. This highlights the efficiency of our tool in these scenarios.

However, feedback indicated that there were some challenges in tasks involving relationships (the fourth and fifth tasks). Partly the reason is that users are not familiar with the Mondial database, but the main reason is that indications are not sufficient to communicate that there was no need to manually select key attributes from other tables. In our application, foreign key attributes are automatically collected to better aid users in choosing attributes, but more indications

Task
Create a pie chart illustrating the distribution of continent areas.
Create a bubble chart of airports showcasing the relation between longitude latitude, and elevation
Create a grouped bar chart showing the population of France, Spain, and Italy for the years 1960 to 2015
Create a tree map illustrating the distribution of airports across different cities highlighting the variation in elevation
Create a sankey diagram showing all of the 5 countries whose area crosses between continents.

and context-specific instructions should be added.

One valuable feedback we received is about the limited selection of attributes available to users in our application. The design of attribute selection is guided by visualisation schema patterns, which restricts users to selecting attributes from a single table. While this limitation reduces the level of freedom users have in attribute selection, it is intended to ensure that users can create appropriate visualisations for their data. In contrast, Tableau offers a high degree of freedom in how users interact with the software and provides a broader range of possibilities in visualisation. However, this increased flexibility often leads to confusion among users regarding which options to choose in order to achieve their desired visualisation.

When considering the graph quality for supporting data exploration, users expressed that the visualisations in Tableau were more interactive. In our application, we utilise D3.js visualisation library, which offers powerful functionalities. However, given the time constraints of the project, it was challenging to refine every visualisation to the same extent as in Tableau. We acknowledge that further refinement and enhancement of the visualisations could have been beneficial.

Chapter 7

Conclusion

In this project, we have delivered a data visualisation tool that utilises the full knowledge of the conceptual schema of the data source to simplify the process of selecting suitable visualisations. By applying reverse engineering to relational databases and mapping them to the set of visualisation schema patterns, we have introduced a visual solution that has demonstrated its effectiveness in various visualisation tasks. In addition, we have explored expanding and implementing additional visualisations for specific patterns. For instance, we have employed pie charts for visualising basic entities and sunburst graphs for representing one-many relationships. Despite these achievements, we recognise that there is still room for improvement in our data visualisation tool. For example, enhancing the logic behind user attribute selection for visualisation, and further refinement of each graph in terms of interactivity and customisation.

7.1 Future Work

Future works include schema transformation to create visualisations. As mentioned in [7], schema transformations provide additional mappings between the transformed database schema and the set of visualisation schema patterns, in order to provide more options for visualisation. A few well-known schema transformations are: **pivot**, **denormalisation** and **mandatory attribute specialisation**.

Pivoting transforms a weak entity into a subset of the same strong entity, with the key attribute merged with other attributes. This enables weak entities to be represented by basic entity visualisations. For example, in figure 4.4 (a), we can pivot the **country_population** to a subset, having attribute **population_1950**, **population_1951**, and so on.

Denormalisation enables one-many visualisations to represent many-many relationship data, by changing one of the entities into a weak entity, and the attributes of the relationship are moved into the new weak entity. One example is illustrated in figure 2.5 previously.

Mandatory attribute specialisation transforms optional attributes to mandatory, by creating a new subset entity with mandatory attributes. This transformation is needed for visualisations that accept mandatory attributes only.

Another aspect of future work is to investigate further on the relations between the set of visualisation schema patterns. For example, in the evaluation section, we have discussed the possibility of weak entities being represented by one-many visualisations. Also, investigation can be conducted to expand the set of visualisation schema patterns, such as reflexive relationship but is one-many, and the corresponding group of visualisations.

Appendix A

Appendix

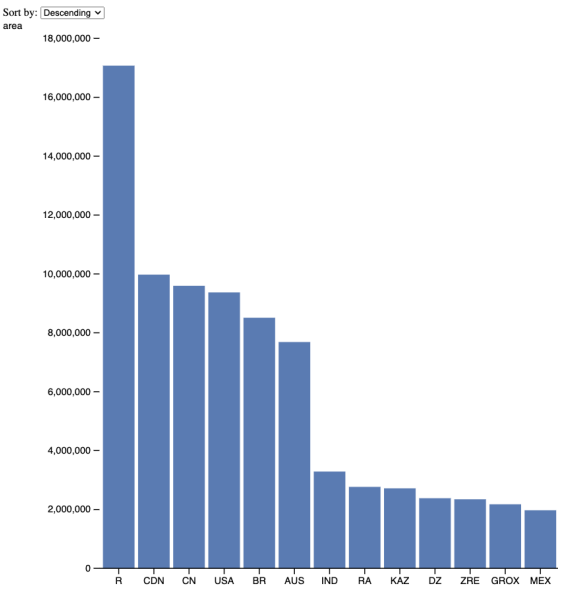


Figure A.1: A descending bar chart showcasing country areas.

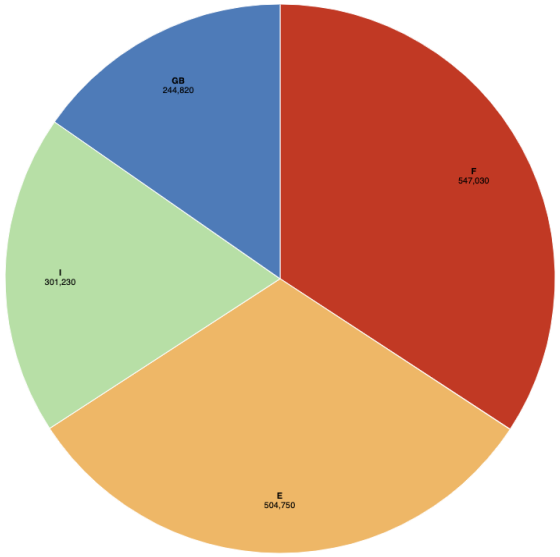


Figure A.2: A pie chart representing the country area for a selected subset of countries.

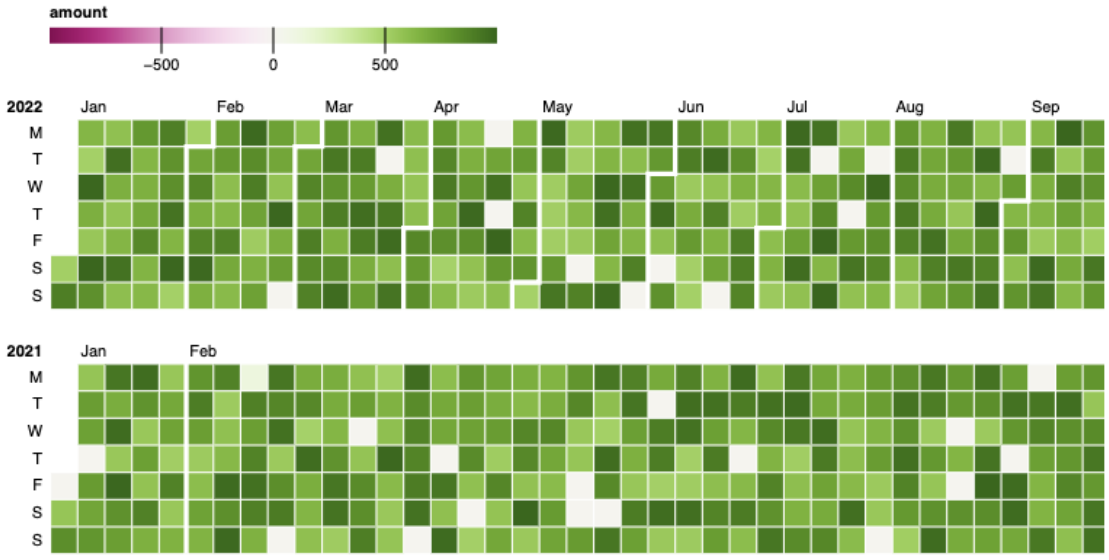


Figure A.3: A calendar chart representing the sales amount of cell phones spanning the years from 2021 to 2022.

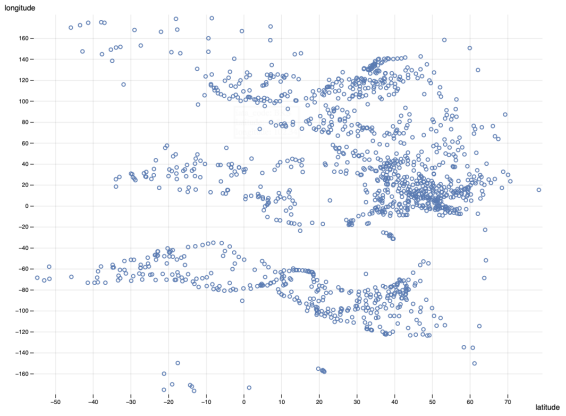


Figure A.4: A scatter plot representing the geographical locations of airports, with latitude plotted against longitude.

Figure A.5: A scatter plot representing airports, with latitude plotted against longitude, and the colour of each data point corresponds to the associated province.

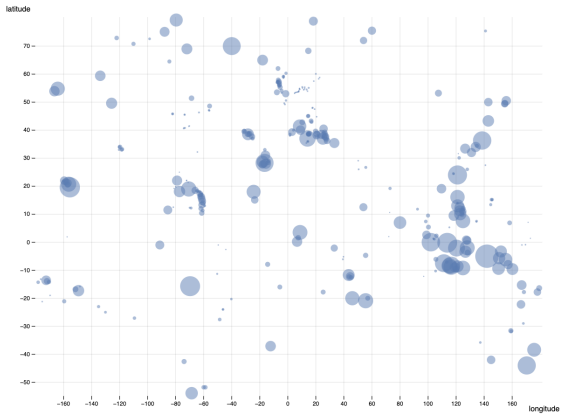


Figure A.6: A bubble chart visualizing the geographical locations of islands, showcasing latitude plotted against longitude. The size of the circles corresponds to the elevation of each island.

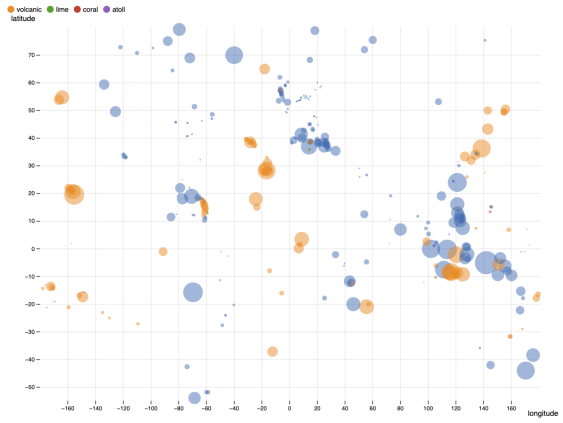


Figure A.7: A bubble chart visualizing the geographical locations of islands, showcasing latitude plotted against longitude with the size of the circles corresponding to the elevation of each island. Each bubble is categorised by the island type in colour.

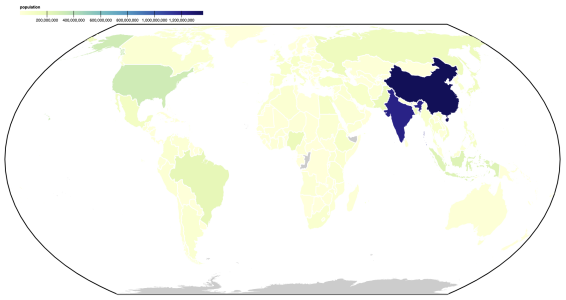


Figure A.8

Figure A.9

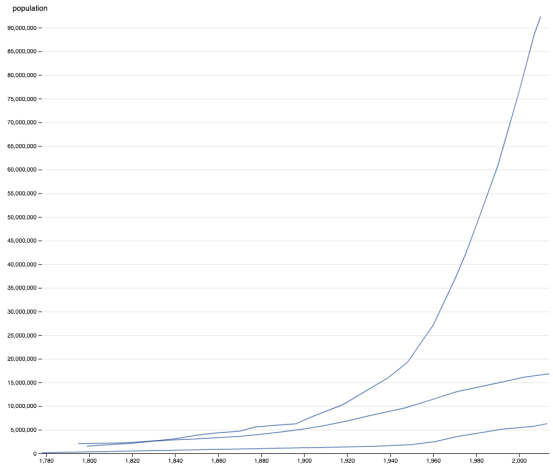


Figure A.10: A line chart depicting population trends over the years for countries of El Salvador, Netherlands and Philippines.

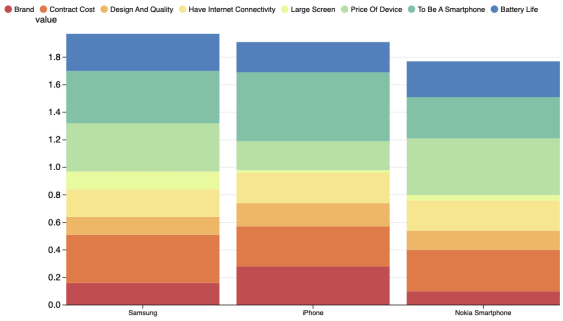


Figure A.11: A stacked bar chart showing user ratings for different phone brands.

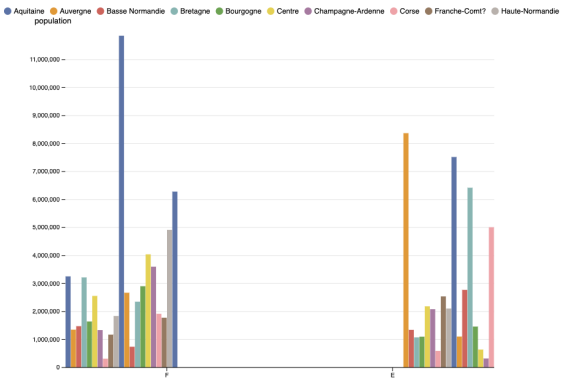


Figure A.12: A grouped bar chart comparing populations of provinces in France and Spain.

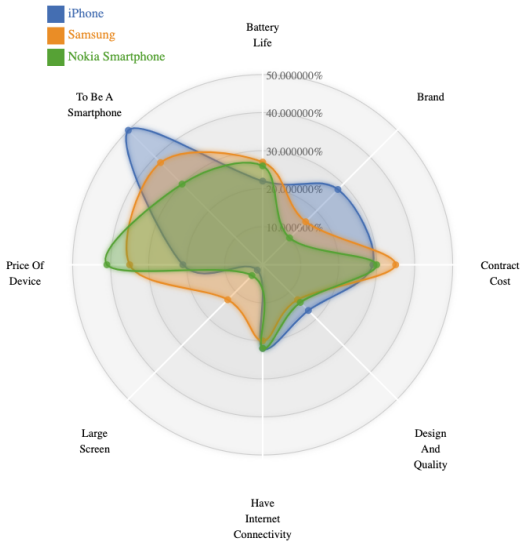


Figure A.13: A spider chart showing user ratings for different phone brands

Figure A.14: A tree map showcasing the distribution of airports across different cities highlighting the variation in elevation.

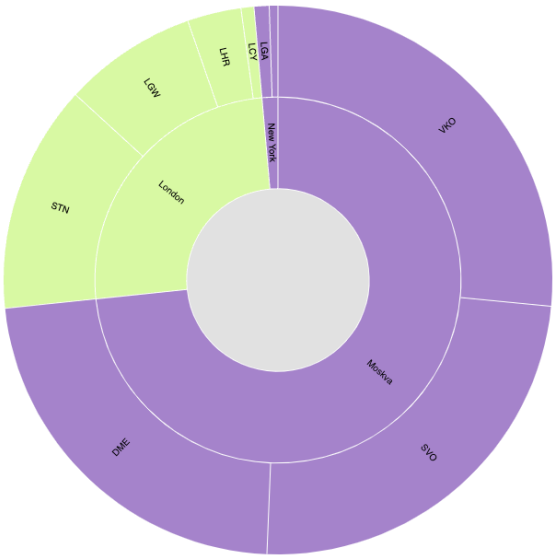


Figure A.15: A sunburst graph showcasing the distribution of airports across different cities highlighting the variation in elevation.

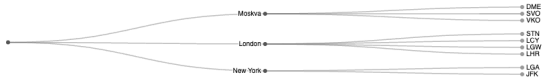


Figure A.16: A hierarchy tree showcasing the distribution of airports across different cities.

Figure A.17: A coloured hierarchy tree showcasing the distribution of airports across different cities.

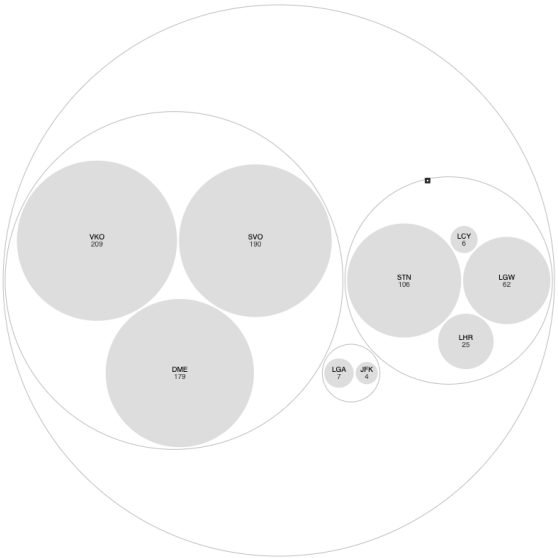


Figure A.18: A circle packing graph showcasing the distribution of airports across different cities.

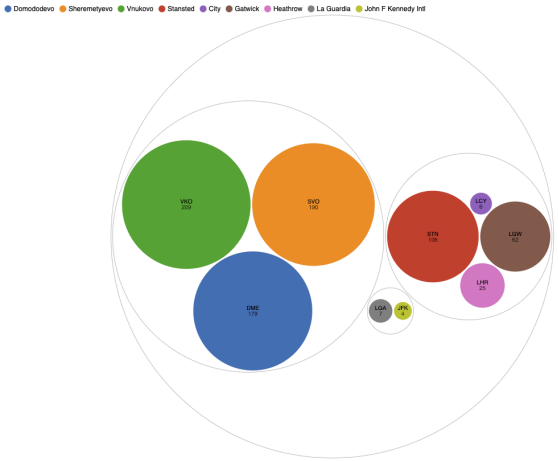


Figure A.19: A coloured circle packing graph showcasing the distribution of airports across different cities.

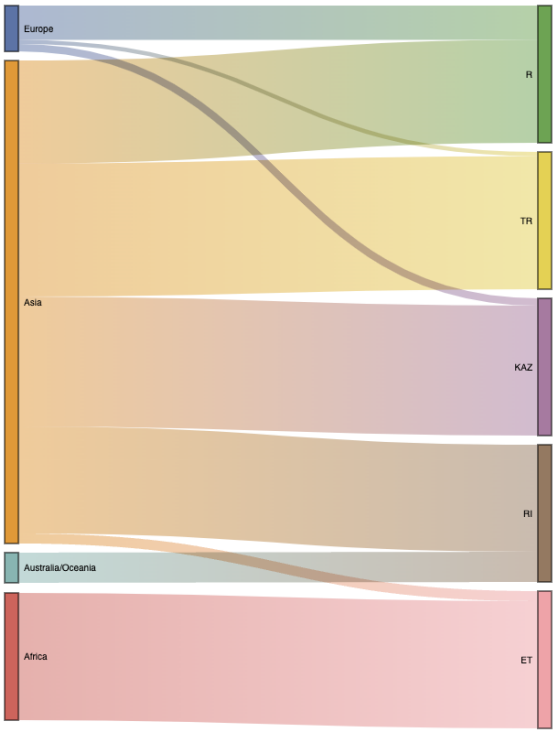


Figure A.20: A sankey diagram of all countries whose areas span across more than one continent.

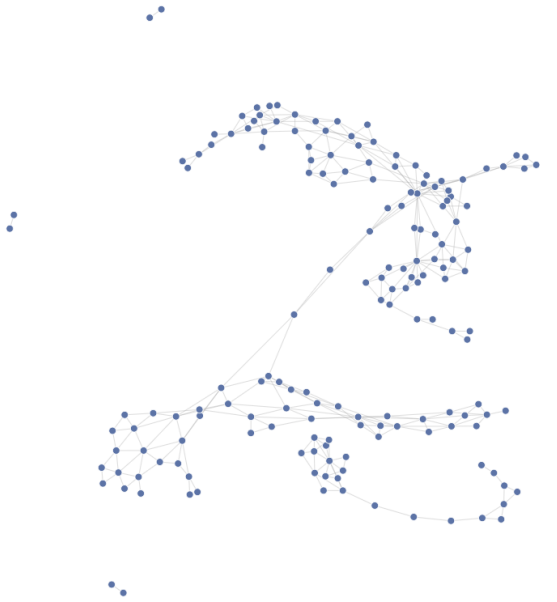


Figure A.21: A network chart depicting the connections between neighbouring countries that share borders.

Figure A.22: A chord diagram showcasing the connections between neighbouring countries whose border length is greater than 2500 kilometers.

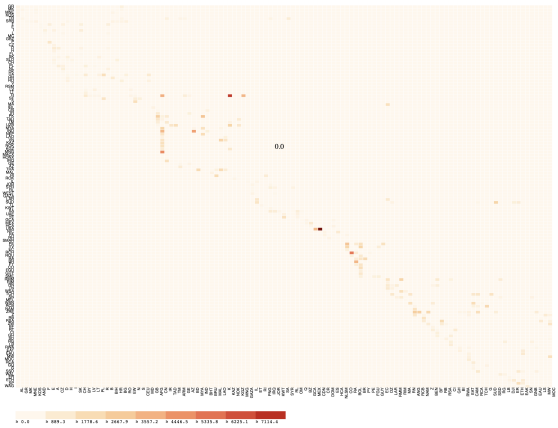


Figure A.23: A heatmap visualising the length of borders between neighbouring countries.

Bibliography

[1] Wolfgang May. *Information Extraction and Integration with* FLORID*: The* MONDIAL *Case Study*. Technical Report 131, Universität Freiburg, Institut für Informatik, 1999.

[2] J.-M. Petit, F. Toumani, J.-F. Boulicaut, and J. Kouloumdjian. Towards the reverse engineering of renormalized relational databases. In *Proceedings of the Twelfth International Conference on Data Engineering*, pages 218–227, 1996. doi: 10.1109/ICDE.1996.492110.

[3] *Tableau*. https://www.tableau.com/ [Accessed: 2023-01-24].

[4] *Google Charts*. https://developers.google.com/chart [Accessed: 2023-01-24].

[5] Leland Wilkinson. *The Grammar of Graphics*. Springer New York, NY, 2nd edition, 2005. URL https://doi.org/10.1007/0-387-28695-0.

[6] P McBrien and A Poulovassilis. Towards data visualisation based on conceptual modelling. pages 91–99. Springer, 2018. doi: 10.1007/978-3-030-00847-5_8. URL http://dx.doi.org/10.1007/978-3-030-00847-5_8.

[7] P McBrien and A Poulovassilis. Conceptual modelling approach to visualising linked data. pages 227–245. Elsevier, 2019. doi: 10.1007/978-3-030-33246-4_15. URL http://dx.doi.org/10.1007/978-3-030-33246-4_15.

[8] W.J. Premerlani and M.R. Blaha. An approach for reverse engineering of relational databases. In *[1993] Proceedings Working Conference on Reverse Engineering*, pages 151–160, 1993. doi: 10.1109/WCRE.1993.287769.

[9] Martin Andersson. Extracting an entity relationship schema from a relational database through reverse engineering. In *Entity-Relationship Approach — ER '94 Business Modelling and Re-Engineering*, pages 403–419, Berlin, Heidelberg, 1994. Springer Berlin Heidelberg. URL https://doi.org/10.1007/3-540-58786-1_93.

[10] *AmazingER*. https://www.javadoc.io/doc/io.github.MigadaTang/amazing-er/latest/index.html[Accessed: 2023-01-23].

[11] *AutoViz*. https://github.com/AutoViML/AutoViz[Accessed: 2023-01-24].

[12] *ggplot2*. https://ggplot2.tidyverse.org/[Accessed: 2023-01-25].

[13] Hadley Wickham. A layered grammar of graphics. *Journal of Computational and Graphical Statistics*, 19(1):3–28, 2010. doi: 10.1198/jcgs.2009.07098.

[14] *D3*. https://d3js.org/[Accessed: 2023-01-25].

[15] *Vega*. https://vega.github.io/vega/[Accessed: 2023-01-25].

[16] Arvind Satyanarayan, Dominik Moritz, Kanit Wongsuphasawat, and Jeffrey Heer. Vega-lite: A grammar of interactive graphics. *IEEE Transactions on Visualization and Computer Graphics*, 23(1):341–350, 2017. doi: 10.1109/TVCG.2016.2599030.

[17] *JDBC*. https://www.doc.ic.ac.uk/~pjm/databases/jdbc.html[Accessed: 2023-05-03].

[18] *Natural Earth*. https://www.naturalearthdata.com/[Accessed: 2023-06-07].