CBS COPENHAGEN BUSINESS SCHOOL
HANDELSHØJSKOLEN

Final Report for the Course Business Data Processing & Business Intelligence

Maximizing the potential of Boliga.dk through data visualization and comparative analysis

Author: Konrad Schulte (149872)

Examiner: Abid Hussain

Copenhagen, 19.12.2022

Pages: 10

Characters including Spaces: 21,760

Table of Contents

1. Introduction 1

2. Theoretical Background 2

2.1. API 2

2.2. Relational Databases 2

2.3. Dashboards 2

3. Methodology 3

3.1. Data Analysis Process 3

3.2. Data Retrieval 3

3.3. Data Description 4

3.4. Data Preprocessing 4

3.5. RDB Implementation and Normalization 5

3.6. Data Visualization 6

4. Results 8

4.1. Scenario One 8

4.2. Scenario Two 9

5. Limitations 10

6. Conclusion 10

References 11

Appendix 12

1. Introduction

The acquisition of the first own apartment or the first own house is a special event in the life of every adult. And because it is a very long-term investment, the decision to buy a property should be made carefully. However, those who move to a new country, or a new area usually do not know much about it, there is a certain information gap for national but also local immigrants. To close this information gap, one must do some research. In Denmark, you can do this mainly with the help of a website called Boliga.dk. Boliga.dk belongs to the media company Boliga Gruppen A/S and went live for the first time in January 2007. Its mission is to make the Danish housing market more transparent. (Boliga Gruppen a/S, 2022) Therefore Boliga.dk lists nearly all apartment and/or house sales in Denmark within the last 30 years and gives information about the year of construction, the price, the size, and many other characteristics of the object.

In addition to its list display, Boliga.dk also has a map function. However, both functions do not allow for a visual, in-depth analysis of residential areas. But the attractiveness of an individual property is always dependent on its surroundings, and the information about such an object only unfolds its true potential when it is seen in comparison and certain trends are revealed. In this paper, the existing sales information from Boliga.dk is used to create an interactive dashboard that enables a comparative analysis of properties and residential areas across Denmark. Hence, the research question to be answered is as follows:

How can Boliga.dk leverage various business intelligence tools to improve its customer experience?

2. Theoretical Background

2.1 API

Application Programming Interfaces (API) allow two different software components to communicate with each other. The software that sends a request in this context is called a client and the application that responds is called a server. The most widely used APIs today are the REST APIs. They were first described by Fielding in his paper in 2000 and are known for their flexibility as well as statelessness. (Fielding, 2000) Stateless means that every HTTP request takes place in absolute isolation. According to Fielding statelessness "induces the properties of visibility, reliability, and scalability." (Fielding, 2000, p.79)

While APIs are usually used to enable this communication between two programs, they can also be used or requested to collect certain data. In this paper, all sales data from Boliga.dk was collected via its (hidden) API (more about this in chapter 3.2).

2.2 Relational Databases

Data can be stored in different types of databases: in-memory databases, columnar databases, streaming databases and many more. However, the most frequently used database is the relational database. The idea of the relational database was first published in 1970 by E.F. Codd. (Codd, 1970) According to this concept, a relational database consists of many different (relational) tables, which store data in a structured way and according to certain rules. For example, each relation must have a primary key, i.e. a column (or a combination of columns) whose value is unique for each row. (Jukic et al., 2014, p.65) Relational databases are created, updated and managed by relational database management systems (RDBMS). To communicate with an RDBMS, the Structured Query Language (SQL) is used. SQL statements can be divided into two categories: Data Definition Language (DDL) is used to create databases or set authorizations. Data Manipulation Language (DML) is used to query or manipulate data.

In order to remove redundancy in a relational database and to simplify information retrieval, normalization is performed. This technique decomposes all relations until they are in “good form" (without redundancy and stored in the most efficient and organized way), while maintaining dependencies and avoiding information loss. (Silberschatz, Korth, & Sudarshan, 2002)

In this paper, the data was stored in a relational database using SQL with pgAdmin, normalized and retrieved from there (see chapter 3.5).

2.3 Dashboards

A dashboard is an interactive user interface that visualizes relevant information in an intuitive, engaging way. Dashboards are designed for a specific audience, who can use filters and other features to quickly, easily, and comprehensibly access key insights. There is some research on design principles for dashboards. Such theoretical design principles include all visual elements from positioning and colors (Cleveland & McGill, 1984) to hierarchies, contrasts and symmetries (Ware, 2019).

In this paper, using sales data from Boliga.dk and Microsoft's PowerBI software, a dashboard was created that allows Boliga.dk's customers to analyze the attractiveness of a residential area.

3. Methodology

3.1 Data Analysis Process

First, the data was requested from a hidden API of Boliga.dk using the Python programming language.

Then the created dataset was cleaned in Python: It was filtered for relevant columns, null values and various outliers were removed. The dataset was then kept as a comma-separated values file.

Using SQL via pgAdmin, a relational database and its tables were created. The tables were filled with the filtered dataset and a normalization was performed.

In the last step a live connection to the database was established and the interactive dashboard was created with PowerBI.

The entire data analysis process of this paper is illustrated in figure 1 and described in more detail in the following chapters.

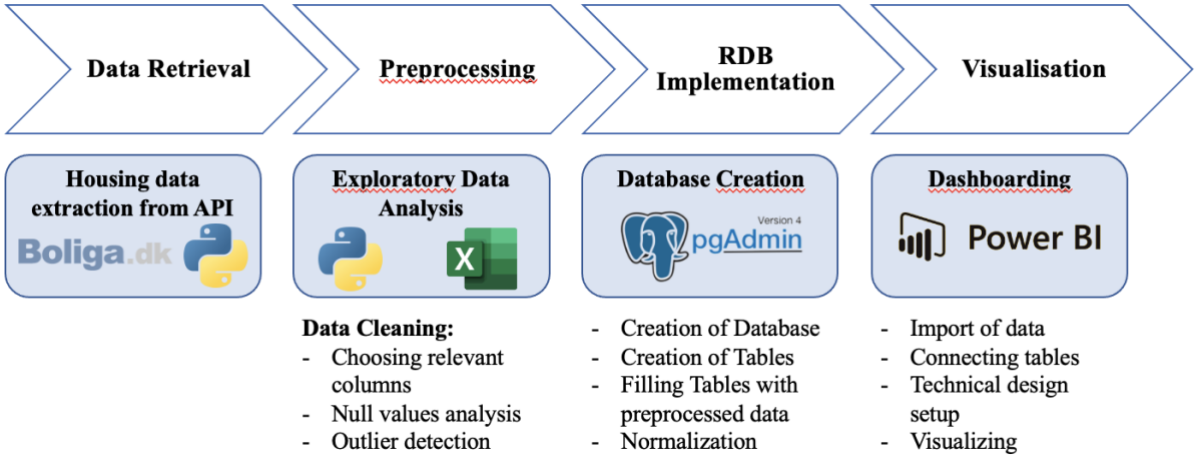

Figure 1: Data Analysis Process Diagram

3.2 Data Retrieval

To collect the information about the Danish housing market, it was originally planned to use the API from Boliga.dk (https://api.boliga.dk/). However, when it turned out that the documentation of this API is insufficient, a new way had to be found. Webscraping (i.e., data extraction directly from websites) seemed to be quite complex in this context, because the HTML code was dynamic. Through deeper research a hidden API from Boliga.dk was discovered. With this API not only all sales objects could be queried, but they were delivered directly in JSON format, which facilitated the later processing of the data.

An example link of this hidden API is as follows:

https://api.boliga.dk/api/v2/sold/search/results?page=1&sort=date-a&pagesize=1000

From this URL it can be seen that the (or this) first part of the database ("page=1") is 1000 entries long ("pagesize=1000") and sorted in ascending chronological order ("sort=date-a"). This structure was used to program the necessary Python code. For this the libraries "urllib.request", "pandas", and "json" were used.

Since each request to the hidden API also contained metadata about the database of sales objects, it could be determined in a first step that the total number of entries was about 1.75 million. In order not to send too large requests to the server, a loop was created with 175 iterations for each 10,000 entries. After receiving each request, the data was transformed from JSON format to a pandas dataframe. This way all 1.75M entries could be collected and finally exported as the first raw dataset in csv format.

3.3 Data Description

The raw dataset included 1,746,613 rows and 22 columns of sales data ranging from 1974 to 2022. A brief overview of this dataset can be seen in appendix 1.

From the overview, it can be seen that the raw dataset had a few inconsistencies. For example, the lowest price was only 2DKK, while the highest was over 240 million. Also, a minimum square meter price of 0.01DKK or maximum square meter price of more than 56 million DKK was hard to believe. Similar anomalies were found in the number of rooms, property size and year of construction. All these findings led to the need for further (pre-)processing of the dataset.

3.4 Data Preprocessing

The preprocessing of this dataset was performed using the "pandas" library in Python. Before dealing with the outliers described above, it was first necessary to determine which of the columns were actually needed for the final visualization. The following 8 columns were found to be redundant or unnecessary and therefore removed: "Unnamed:0", "Estate ID", "Change", "GU ID", "Estate Code", "Group Key", "Can get VR" and "Bf Enr". After removing these columns some column data types had to be changed, because Python had difficulties to identify them correctly. The "Sold Date" column was shortened to the "YYYY-MM-DD" format to facilitate later processing. A unique identifier was also created and added to the dataset. The original dataset had an ("Unnamed") index column. However, this had to be removed in the first step due to inconsistency. Similarly, the "GU ID", which was initially assumed to be a unique identifier, had fewer unique values than the length of the entire dataset.

In total, two of the remaining columns had zero values, namely "Sqm price" and "City". In the case of null values, there are two ways to clean up the dataset: either remove the rows containing null values or impute values. In this paper, an imputation of missing values was performed both times. In the case of "Sqm price", missing values were determined by dividing the "Size" and "Price" columns. An examination of the rows with missing "City" values revealed that they all shared the same postal code: 8799. An online check then revealed that this was the city of Tunø and the missing values were filled in accordingly.

The raw dataset contained only numbers of 1, 2, 3, 4 and 6 as values for the “Property Type” column. Based on research, these numbers could be assigned to the respective property type "Villa", "Terraced House", "Condominium", "Holiday Home" and "Country Property" respectively and the "Property Type" column was decoded accordingly.

After these steps, the dataset has now been cleaned of outliers: All entries of objects smaller than 20 sqm or larger than 2,000 sqm were removed. Also, properties with less than 1 or more than 12 rooms were filtered out. Finally, all sales properties with a sqm price below 5,000DKK

or above 100,000DKK were filtered out and construction years below 0 or above 2022 were removed. Figure 2 shows a short overview of all the different preprocessing steps.

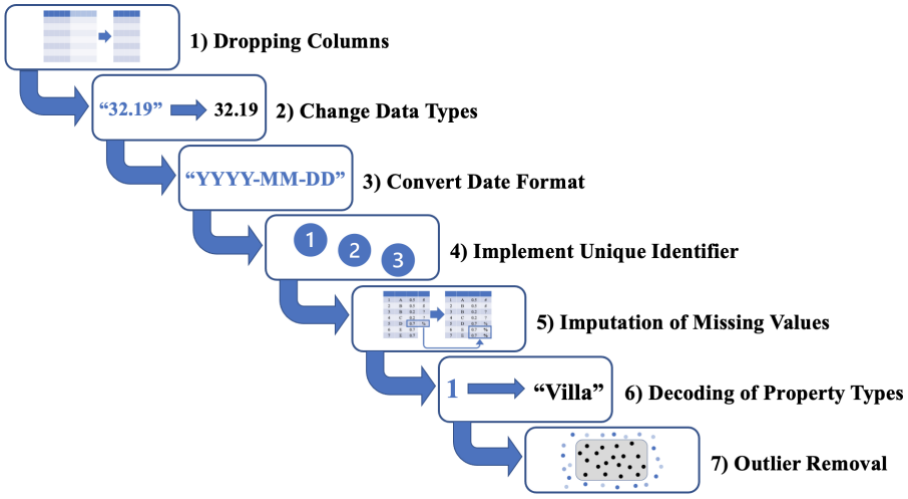

Figure 2: Data Preprocessing Steps

After all these preprocessing steps, the resulting dataset had 1,305,802 rows and 14 columns.

3.5 RDB Implementation and Normalization

In reality, one hardly ever works with a single dataset; normally, the data for a project must be compiled or queried from various sources. To make the project more realistic, a SQL database was simulated. To create the relational database the widely used administration and development platform for PostgreSQL called pgAdmin was used. Using the Data Definition Language, the resulting table from the preprocessing was created and then populated with the appropriate data.

However, this table contained a redundancy. While the "SellID" acts as a unique identifier, the following columns can be derived from the "Address" column: "ZIP Code", "City", "Latitude", "Longitude" and "Build Year". Whereas other values such as the price of an object or its size (due to annexes) may vary over time, these properties remain constant. To eliminate this redundancy, the table was decomposed into two relational tables: R1, consisting of "SellID", "Address", "Price", "PPSM", "Sold Date", "Property Type", "Rooms", and "Size", and R2, consisting of "Address", "ZIP Code", "City", "Latitude", "Longitude", and "Build Year". However, the relation R2 still contained a functional dependency, namely the derivation of the column "City" from the column "ZIP Code". To resolve this redundancy, the R2 relation was then further decomposed into the following two relations: R2, consisting of "Address", "ZIP Code", "City", "Latitude", "Longitude" and "Build Year" and R3, consisting of "ZIP Code" and "City". After this last decomposition all tables were now in BCNF, so the normalization was completed.

The code of the Data Definition Language for the steps described in this chapter can be found in appendix 2. The relational schema of the final relational database is shown in figure 3.

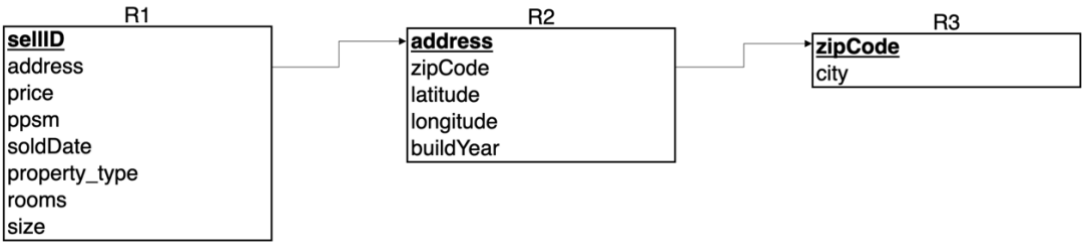

Figure 3: Relational Schema of Relational Database

3.6 Data Visualization

The dashboard shown in Figure 4 was created using Microsoft's PowerBI software with a live connection to the relational database.

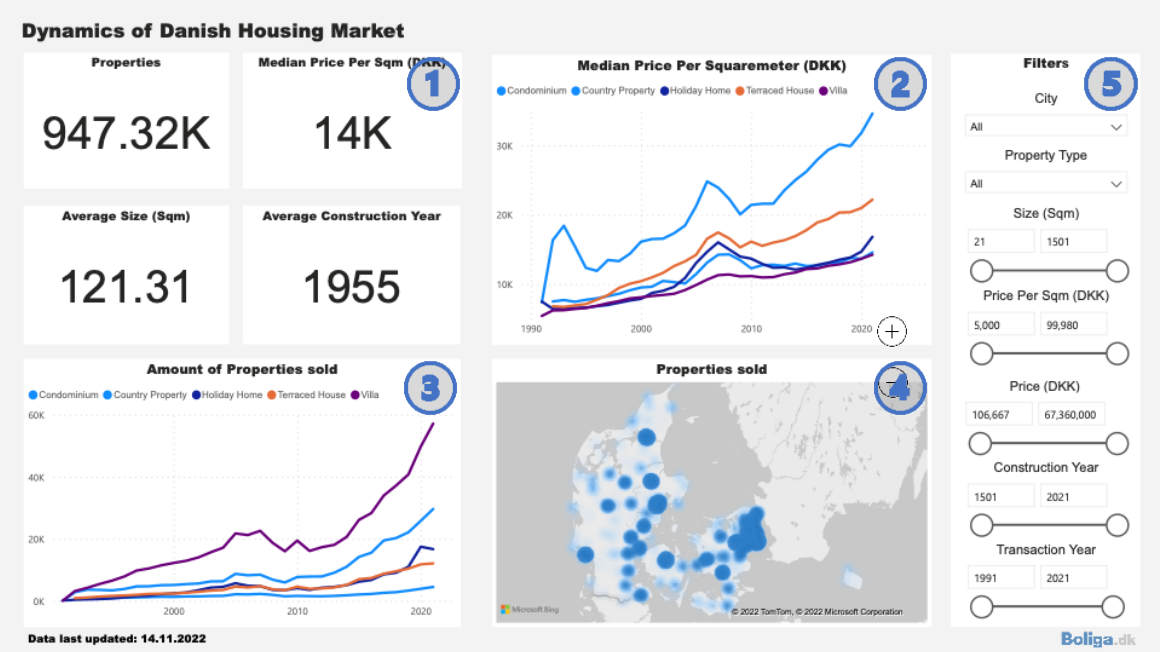

Figure 4: Dashboard

The target audience of this dashboard is any user of Boliga.dk (mainly private individuals) who is thinking about buying a residential property in the (near) future and is conducting research with a computer. The goal of this quantitative dashboard is to help the target audience in their location search by illustrating how different residential areas have developed over time, where historically many sales have taken place and what the development of prices is. Since covering all these goals means having to visualize some data in just one view, it was decided to keep the design as simple and intuitive as possible to avoid overwhelming the user. By deliberately avoiding green and red colors, accessibility for people with the most widespread color blindness called deuteranopia has also been ensured (Domingues, 2022).

A dashboard should be readable like a book, starting at the top left and ending at the bottom right. This premise was also adhered to when creating this dashboard. In addition, the individual visualizations were displayed in boxes to preserve a clean structure. In total, the dashboard comprises four areas: (1) KPIs, (2) price per square meter (over time), (3) amount of properties sold (over time) and (4) properties sold.

(1) The KPI section comprises four key figures: properties, median price per square meter, average size, and average built year. These KPIs have been deliberately displayed in large format to give the user a quick overview of the data shown (and possibly filtered). Here, the "Properties" indicate how many individual objects are used or mapped in the view. "Median price per sqm" is an indicator of the price level that prevails in an area. For this KPI, it was deliberately decided not to aggregate using the average, as the median is less sensitive to outliers. "Average size (Sqm)" indicates the average size of residential properties and "average construction year" their average age, respectively.
(2) In addition to these KPIs, a line plot was placed to show the development of square meter prices in the selected area over time. Again, the median was chosen as it is more resilient to outliers.
(3) Another line chart was placed in the lower left corner of the dashboard which shows the development of (count of) property sales over time. For both representations of developments over time line charts were used, since these can represent trends particularly well. Both line charts are also divided and colored by property type to show differences between them.
(4) The last visualization in the lower right corner of the dashboard is a density map. It shows how often objects were sold in a certain area and includes two buttons to zoom in and out.
(5) To provide a comprehensive customization of the dashboard, a column with filters has been added on the right side. This filter bar includes a total of seven filters: two filters with discrete values (City and Property Type), where either one value or different combinations of values can be selected, as well as five filters with continuous values (Size, Price per sqm, Price, Construction Year, and Transaction Year) where a minimum and a maximum value can be set. The filters are arranged from top to bottom in a hierarchical logic: The first two filters allow for a rough geographic delimitation as well as a specification of the object type. The properties themselves can then be further filtered according to size, price (total price as well as price per square meter) and construction and sales year.
In addition to the filters visible to and usable by users, two permanent filters have also been applied to the dashboard. Since the year 2022 was not yet completed at the time of data retrieval, the values of this year led to visual distortions. To counter this, all sales days after 12/31/2021 were filtered out. In addition, to filter out sales of properties not yet completed, the years of construction were limited to a time range of 1500 to 2021. The lower limit of 1500 was set to make the "Construction Year" filter more sensitive. Previously, the values ranged to year 0, but included only 333 additional sales, so almost three-quarters of the filter range affected only 0.03% of the total data. Together, the two permanent filters removed 6.1% (or 80,114 rows) of the dataset, reducing it to 1,225,688 remaining rows.

4. Results

To better illustrate the benefits of the finished dashboard, two scenarios are described below in which users make use of this dashboard to facilitate their location search.

4.1 Scenario One

In the first scenario, a retired couple wants to move to a new home in Amager (Copenhagen). The home does not have to be very large, but it is very important to them to move to a sustainable neighborhood, i.e. an area where neighbors do not change frequently. Using the dashboard they created, they can (1) first search for the right location (in this case "København S") and also (2) filter for their desired size. To find out how much fluctuation has taken place in the real estate market in this area, (3) the transaction years are also limited to the last five years. After applying all these filters, the density map unfolds its true potential, as figure 5 shows.

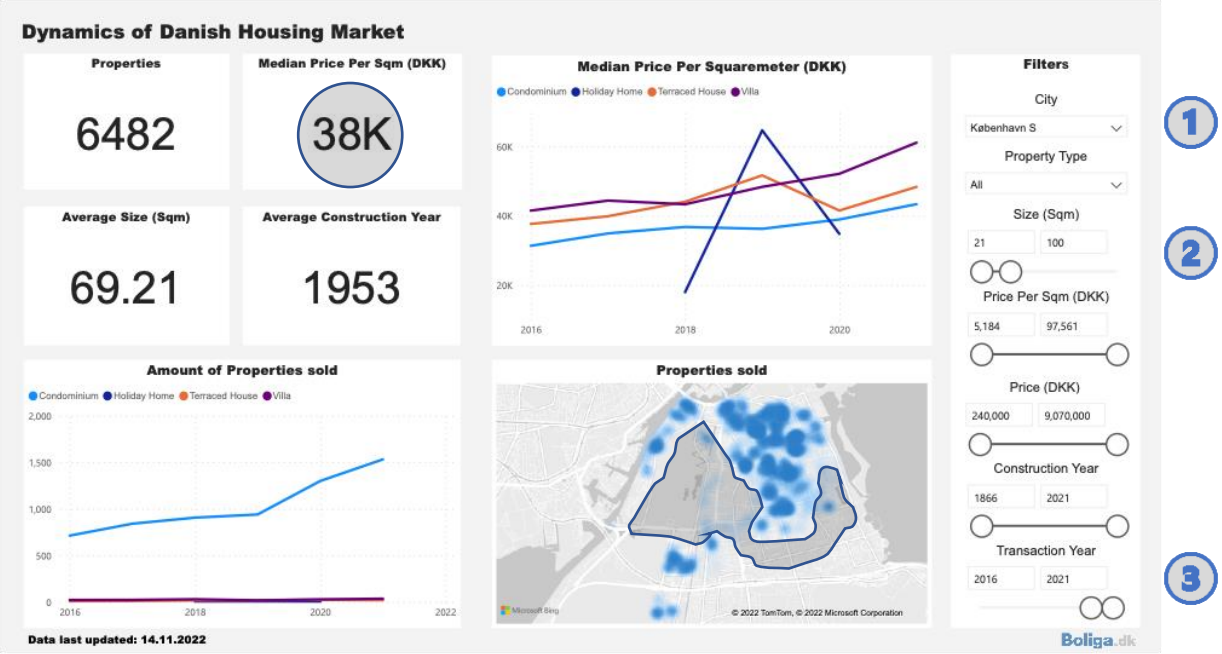

Figure 5: Scenario 1 – Pensioners looking for "sustainable" residential area in Amager

For the pensioners, the dashboard visualizes very vividly which residential area meets their requirements; in figure 5, this area is circled and shaded gray. In addition, the dashboard shows that this relatively old neighborhood is almost exclusively made up of apartments. The fact that the median square meter price is around 38,000DKK (also circled and shaded grey in figure 5), which is 4,0000DKK lower than the price of whole Copenhagen, further delights the pensioners and encourages them in their choice of neighborhood.

4.2 Scenario Two

In a second scenario, a young family moves from Copenhagen to Odense for a new job. On the one hand, it is important to the parents that they move into their own house with enough space for their children. In addition, they would like to move to a modern new development because many young families will probably live there and modern style with existing technology is important to them.

Also in this case it is possible to (1) filter for the city of Odense first. Since the family does not want to move into an apartment but into a house, (2) only "Terraces House" and "Villa" are selected with the Property Type filter. In order to meet their (3) size expectations and (4) demands regarding the novelty of the residential area, the dashboard is further filtered accordingly and the result looks like this:

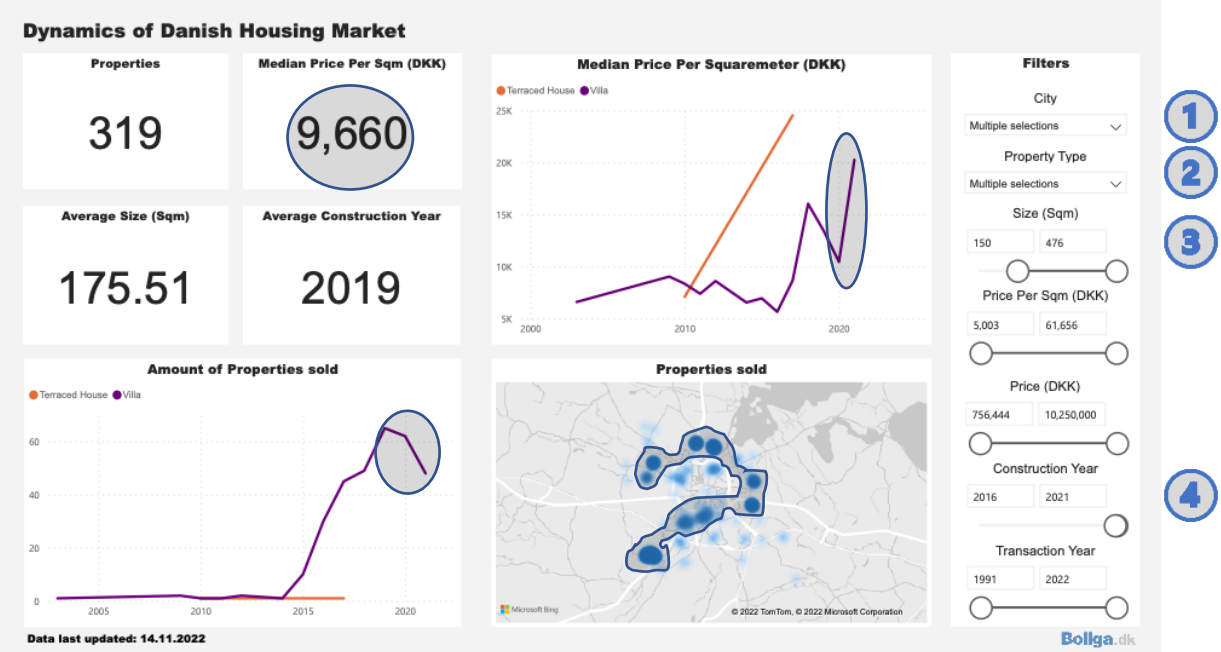

Figure 6: Scenario 2 – Young family looking for a modern housing area in Odense

First, the couple is happy that the prices per square meter are much lower than in Copenhagen. The fact that the value development of the villas suggests a positive trend, while the total number of villas sold is currently decreasing, also shows them that the timing for buying a house in Odense might be favorable. The density map further indicates the areas in which the couple could locate a residential area that meets their requirements. All these insights are circled and shaded grey in figure 6.

5. Limitations

This project involves several assumptions and limitations which are described in the following. The first limitation is related to the data not being adjusted for inflation. This could lead to price developments being distorted and price increases being over-interpreted. Furthermore, the dashboard only uses sales data, i.e. data from the past, and by filtering out the values from 2022, this effect was even amplified. Historical developments can have an influence on the present and future, but this is not necessarily the case. In addition, the data are data from the change of ownership. An assumption, as in Scenario 1, that a neighborhood is more sustainable simply because residential properties are sold less frequently may be misleading, as this data does not provide information about tenant turnover (a property may have the same owner for decades while tenants change annually). The data from Boliga.dk is of course not complete, which can also lead to visual distortions. Due to the lack of space in the dashboard, a filter for rooms has been omitted. But this can be important for some users.
One last limitation is that this project did not describe how exactly the dashboard would be published. This would be done using the "Publish" service of Power BI, however, this is in the hands of Boliga.dk.

6. Conclusion

This paper has shown how Boliga.dk can use different business intelligence tools to improve its customer experience. A first step was to collect all sales data from Boliga.dk using Python and a hidden API. Then a relational database was simulated with pgAdmin and finally an interactive dashboard for the users of Boliga.dk was created with Power BI. Two examples were used to illustrate how this dashboard can be used for different purposes and how it can be useful. Even though the dashboard is based on some assumptions and has its limitations, it could be shown that Boliga.dk has possibilities to improve the customer experience in a sustainable way and ultimately enable its users to analyze not only residential properties but also residential areas from home.

References

Boliga Gruppen A/S. (2022). Retrieved December 19, 2022, from https://boligagruppen.dk/

Cleveland, W. S., & McGill, R. (1984). Graphical perception: Theory, experimentation, and application to the development of graphical methods. *Journal of the American statistical association*, *79*(387), 531-554.

Codd, E. F. (1970). A relational model of data for large shared data banks. *Communications of the ACM*, *13*(6), 377-387.

Domingues, M. (2022, November 14). *Dashboard Design: How to Color Your Data Wisely*. Retrieved December 19, 2022, from https://www.biztory.com/blog/how-to-color-your-data-wisely

Fielding, R. T. (2000). *Architectural styles and the design of network-based software architectures*. University of California, Irvine.

Jukic, N., Vrbsky, S., Nestorov, S., & Sharma, A. (2014). *Database systems: Introduction to databases and data warehouses* (p. 400). Pearson.

Silberschatz, A., Korth, H. F., & Sudarshan, S. (2002). *Database system concepts* (Vol. 5). New York: McGraw-Hill.

Ware, C. (2019). *Information visualization: perception for design*. Morgan Kaufmann.

Appendix

Appendix 1: Raw Dataset Overview

Column Name	Data Type	Unique Values	NA Values	Minimum	Maximum
Unnamed:0	int64	1,746,613	0	0	1,746,612
Estate ID	int64	635,739	0	0	1,941,006
Address	string	1,288,718	0	-	-
ZIP Code	int64	961	0	1050	9990
Price	int64	159,351	0	2	243,472,000
Sold Date	string	11,270	0	-	-
Property Type	int64	5	0	1	6
Sale Type	string	5	0	-	-
Sqm Price	float64	399,344	334	0.013245	56,250,000
Rooms	float64	90	0	-8	465
Size	int64	943	0	-360	3635
Build Year	int64	477	0	0	2023
Change	float64	54,630	0	-	-
GU ID	string	1,746,575	0	-	-
Latitude	float64	550,173	0	5	57.747334
Longitude	float64	1,181,276	0	8.086473	15.1538
Municipality Code	int64	98	0	101	860
Estate Code	int64	410,458	0	1	999980
City	string	604	159	-	-
Group Key	float64	560	1,478,219	1	560
Can get VR	bool	2	0	-	-
Bf Enr	int64	1508565	0	0	100,531,668

Appendix 2: DDL Code for RDB Implementation and Normalization

A) Create the table for the data

```
CREATE TABLE R1(
        SellId INT PRIMARY KEY,
        address VARCHAR NOT NULL,
        zipCode INTEGER NOT NULL,
        city VARCHAR NOT NULL,
        latitude DECIMAL NOT NULL,
        longitude DECIMAL NOT NULL,
        price REAL NOT NULL,
        ppsm INTEGER NOT NULL,
        soldDate DATE NOT NULL,
        property type VARCHAR NOT NULL,
        rooms INTEGER NOT NULL,
        size INTEGER NOT NULL,
        buildYear INTEGER NOT NULL);
```

B) Inserting data into the table

```
COPY R1(SellId, address, zipCode, city, latitude, longitude, price, ppsm, soldDate,
propertype, rooms, size, buildYear)
        FROM 'REALLY final dataset.csv'
        DELIMITER ','
        CSV HEADER;
```

C) Normalization:

```
CREATE TABLE R2(
        Address VARCHAR PRIMARY KEY,
        zipCode INTEGER NOT NULL,
        latitude DECIMAL NOT NULL,
        longitude DECIMAL NOT NULL,
        buildYear INTEGER NOT NULL );

CREATE TABLE R3 (
        zipCode INT PRIMARY KEY,
        city VARCHAR NOT NULL);
```

D) Add the data in the new relations:

```
INSERT INTO R2
        SELECT (DISTINCT Address), zipCode, latitude, longitude, buildYear
        FROM R1;

INSERT INTO R3
        SELECT (DISTINCT zipCode), city
```

```
    FROM R1;
```

E) Alter the old relation by removing the decomposed columns:

```
ALTER TABLE R1
    DROP COLUMN zipCode, city, latitude, longitude, buildYear;
```