Università
della
Svizzera
italiana

**Software
Institute**

Jeeves

An interaction–centric approach to data visualization using voice control

Andrea Vicari

Supervised by
Prof. Michele Lanza & Dr. Marco D'Ambros

SOFTWARE & DATA ENGINEERING MASTER THESIS

Abstract

In a world that generates more than 2'500 petabytes of data every day, data analysis and visualization is becoming essential in every type of organization. Data analysts and programmers are the main figures who, with the support of tools, can access to this large amount of information.

A *de facto standard* for data scientists is to use notebook–based tools (like Jupyter or Apache Spark): these tools provide the possibility to interact with cells in which they can perform analysis and create visualizations by writing code. Non–technical users however cannot benefit from these solutions because they require a technical background and a certain level of programming skills.

In this thesis we present our solution to this problem: "*Jeeves*". What we propose with our work is a tool that makes the process of data analysis and visualization available also to people that do not posses programming skills but are subject matter experts in different domains that require data analysis.

Jeeves provides an interaction–centric approach to data visualization using vocal queries. By studying the state–of–the–art tools and from the experience gained with a similar experiment we ran called BoardBile, we created a web application following three requirements that we extracted: *accessibility*, *simplicity* and *ownership of the data*.

The paradigm we propose is to simplify the interaction with data visualization. To this aim, we provide users with the possibility to use natural language speech to perform queries in a seamless and familiar way.

The setup for *Jeeves* consist in deploying the web application on a big screen, next to a vocal assistant (in our work the Amazon Alexa device). The vocal assistant records and sends queries performed by the user to *Jeeves* which is responsible for processing them and suggesting optimal visualizations based on the shape of the data.

To provide a meaningful visual recommendation, we encoded the body of knowledge about the best principles in data visualization as provided by the work of Tufte and Bertin.

To validate our work we ran some experiments by allowing users to try *Jeeves*. From these sessions we gathered some anecdotal evidence and received feedback and suggestions that helped us to shape and to tune the features provided by our tool.

This first experiment demonstrated how vocal interaction can be a promising approach to bring users closer to the world of data analysis and visualization by simplifying the interaction with it.

Dedicated to my beloved

Acknowledgements

First of all I would like to thank my advisor and mentor Prof. Dr. Michele Lanza for his supervision on this work, for being the professor that in the past 5 years has always inspired me to push towards my limits and for having convinced me to enroll in the master in Software & Data Engineering, choice that I would never regret.

I want to thank the Code Lounge Team and, in particular, my co–advisor Dr. Marco D'Ambros who gave me the opportunity to work on such an interesting topic and for transmitting me the passion for data analysis and visualization. For helping me in shaping my ideas, for providing me with the tools to work on this thesis (the Amazon Echo Device), for supporting me in organizational issues and reviewing the drafts of this document even during holidays.

A particular acknowledgement to the Faculty of Informatics of USI and to the professors at the Software Institute for having given me the possibility to study the subject I love the most in the best way I could have ever imagined.

I would never thank my family enough: my mother, my father and my brother Paolo for believing in me, for both the emotional and economical support they always gave me during this last 5 years that I was so far from my hometown.

Thanks to my friends and all the people who stayed close to me during this experience. To the "Fundrisi" company from Enna for being my second family and to the "Mascalzini" fellows for being my best mates during these amazing 5 years of university.

To Redona, for her great love and support; for this last year together and for our projects about the future.

Contents

List of Figures

List of Tables

Listings

Chapter 1

Introduction

"What is good visualization? It is a representation of data that helps you see what you otherwise would have been blind to if you looked only at the naked source. It enables you to see trends, patterns and outliers that tell you about yourself and what surrounds you"[36] this is the definition of good visualization that Nathan Yau provides in his book "Data Points: Visualization That Means Something".

Data visualization has proved to be an effective approach for summarizing information, breaking down complexity, understanding trends and spotting patterns and outliers. The usage of graphics for presentations or study of data has received a lot of attention in the past 50 years especially thanks to the work of people like Bertin[30] or Tufte[32]. In their works they did not provide formal theories[14] but they formulated principles that we followed in our work.

Nowadays, with the huge availability of data in each kind of field, a lot of decisions that used to be made relying on intuitions and experiences, can now be based on real data and tools that, together with the previous knowledge in this field, helps to go towards a data driven decision–making. Companies invest significant resources in the data science field to support the decision–making process based on data. Data scientists benefit from the use of tools to perform operations over data and, at the same time, let them visualize in various ways the information contained in it. Most of those tools require expertise in both programming and data analysis so not all users can benefit from the results generated by those programs.

The state–of–the–art tools can perform data analysis and visualization, but they may require knowledge of programming language or some training before being used. Moreover, there are some limitations in the consumption of data: for example, the growing privacy–concerned problems in cases in which it is encouraged to share data between websites or to put data in cloud services. We developed a tool that lowers the barriers of data fruition by providing a flawless experience that implements different interaction means with the data (using vocal commands) and that suggests to the user the optimal solution based on the shape of data. Moreover, we propose a deployment on premise on the user's machine such that they can maintain the ownership of data.

Before demonstrating the approach we built for this master thesis work, it is important to stress and demonstrate why data visualization is useful.

1.1 The importance of data visualization

According to Stasko et al. software visualization is *"the use of the crafts o typography, graphic design, animation, and cinematography with modern human-computer interaction and computer graphics technology to facilitate both the human understanding and effective use of computer software"*[31].

For Kirk, Data Visualization has a simpler definition: "*The Representation and presentation of data to facilitate understanding*" [21].

The pictorial or graphical format was used as a mechanism for communication since before the formalization of written language. Human brain can process an image much more quickly than a page of words because the interpretation of images is performed in parallel within the human perceptual system while the speed of text analysis is limited by the sequential process of reading [25]. Moreover, pictures can be language-independent, as an image, a graph or a chart may be understood by a group of people with no common language. According to Ward, Grinstein and Keim [25], visualization of data helps to:

1. "Better discover and identify areas that need attention or some kind of improvement
2. Clarify which factors can influence a change in the behavior of the data
3. Understand and/or predict trends in the data"[25].

The purpose of data visualization is to show the information in a good display and aid readers or viewers in seeing the structure in the data [14]. Also, it facilitates the process of understanding as, when consuming a visualization, the viewer will go through the three stages process of *perceiving*, *interpreting* and *understanding* [21]. A good data visualization allows the user to break down the complexity, understand trends, spot patterns and outliers, summarize vast amount of data in a single view and contribute to take better decisions. A visualization can be *interactive* meaning that one can take this concept to go a step further by using technologies that allow to go more in depth with the visualization of graphs and charts to refine the level of detail by interactively changing the data to be shown and its processing.

In many applications, raw data has limited value in itself: extracting the information contained in it [19] and showing them as a data visualization can allow one to get knowledge and understand better trends and pattern that would remain hidden otherwise. Back in 1994 [29], this topic was already considered to be an important and accepted discipline, on the other hand, resources at disposal of scientists were limited. With the progress and evolution of computational power in computer machines, it has been possible to create visualization tools that greatly helped to reduce the effort [29]. Nowadays, the very same problem has moved from experts to non–experts: our goal with this thesis is to bring data closer to the users providing them with various and intuitive ways of interacting with these informations.

As data spreads among many business areas, it is important to have the ability of organizing it in a meaningful manner such that it can support quick understanding. There exist many solutions and tools that empowers specialists to visualize data in order to understand it better.

We now present thew approach that we want to propose with this master thesis work.

1.2 Our approach

In this work, we created a tool called *Jeeves* in which we propose an approach to simplify the interaction with data. We want to allow any person to create data visualizations and interaction without the need of programming skills and also to move the paradigm of having data and visualization on the same desktop device. We use a voice assistant and let people perform queries on data by using vocal commands. The queries are analyzed and system chooses the best way to present a result by inferring the optimal type of visualization. The results of the queries are visible on a separate bigger screen.

With this approach, people can visualize and interact with data, answer to their own questions or easily present insights to others. Data resides in our system only and it is the user who decide which aspect of it to focus on.

We fulfill three main requirements that we extracted by studying the state–of–the–art solutions. Our web application provides:

- *Accessibility*: access to the data will always be available since we provide a web application that can be queried using vocal commands.
- *Simplicity*: users are provided with an interactive dashboard that can be personalized depending on the type of visualization in mind. They ingests data in the system from different sources and, with the use of a vocal assistant, they can perform commands to create a new view, rearrange the dashboard or change a filter types. All of that, using the most natural tool they have at their disposal: voice. Moreover, *Jeeves* provides an automatic inference based on the shape of data: when the user asks to create a visualization, our tool shows the optimal type of chart visualization depending on the type of columns in the dataset.
- *Maintained Ownership of the data*: the information provided by our approach is stored in a centralized server and users can choose whether to share or not the datasets. For example, a company could choose to let the user interact with its data that resides in its data centers rather than give those informations to third-party data visualization companies. Users of the company, will just use their voice to navigate through the available data and understand it in a more exhaustive way.

With this work we do not only provide a web application but also a vocal experience through the use of a voice user interface (VUI). For this reason, we decided to follow some others key concepts as suggested by the Amazon Alexa Documentation. Our vocal application then provides:

- *Adaptability* to users speak naturally and still be understood by the vocal assistant
- *Personability* by always considering the context in which the application is used
- *Availability* by providing users with menus to interact with the application
- *Relatability* to have a natural conversation–like feeling between the user and the vocal assistant

To validate our work, we gathered some anecdotal evidence by deploying the application and running some experiments among users; we received some important feedback on the usability of our tool and suggestions on how to have a better user experience.

1.3 Structure of this document

- **Chapter 2** presents the state of the art. It introduces existing tools which allow data visualization in a virtual environment. In particular, it focuses on tools that use the same idea of making data visualization interactive and accessible to user (even the ones without training) by comparing and contrasting the existing features with the ones proposed in this work.

- **Chapter 3** explains which features *Jeeves* provides and which ones are the requirements we fulfill in the application. We present how *Jeeves* works and how it can answer to questions about a dataset only by using the vocal commands feature. We continue by describing the challenges we encountered during the experiments we runned and we conclude by explain how our tool overcomes them.
- **Chapter 4** focuses on the implementation of *Jeeves*. We discuss the architecture that we chose, the technologies that we adopted and explain the key components of our tool. We then present technical information about how we model the data in our system and how its components interact with each other.
- **Chapter 5** presents the conclusions of this master thesis work: we summarize the process we followed to develop *Jeeves*, how we validate our tool and what are its contributions. We conclude this work by presenting a list of nice–to–have features to further improve *Jeeves* in the future.

Chapter 2

Related Work

In this chapter we present some tools for data visualization, analysis and interaction. We divide them into two categories: *Data Analytics Tools* and *Data Visualization Tools*: the former deals with data at a deep level while the latter makes explicit the trends and patterns inherent in the data (usually generated by a data analysis tool). We compare and contrast the characteristics of those tools with respect to the functionalities we provide in our application, and we present some state–of–the–art libraries used by developers for information visualization and data interaction.

Later in this chapter, we present the tool called *BoardBile*: it is our first experiment towards an interactive approach to data visualization. Starting from the feedback we acquired during the public demonstration of this tool, we decided to continue in this direction by providing different means of interactions with the data.

Before building our tool, we analyzed the current state–of–the–art to understand the most common approaches and technologies used in the field of data analysis and visualization, and we used this study for the development of *Jeeves*.

2.1 Data Analytics

Data Analytics tools can help bring data to life: they allow users to extract and transform data into a usable format by providing tools to clean and filter the data. They also provide users with insights into their datasets. The following sections present two examples of data analytics tools: Jupyter Notebook and Apache Zeppelin.

2.1.1 Jupyter Notebook

Jupyter Notebook[1] is an open source, client–server application that allows the creation of the so called notebook documents (as shown in Figure 2.1) which are then ran through the web browser. Notebook documents are both human-readable documents containing the analysis description and the results (figures, tables, etc..) as well as executable documents which can be run to perform data analysis. This application can be both used offline or in the cloud and it is mainly used for applying data cleaning and data transformation, numerical simulation, statistical modeling, data visualization and machine learning processes.

[1] Jupyter Notebook Official Website: `http://jupyter.org/`

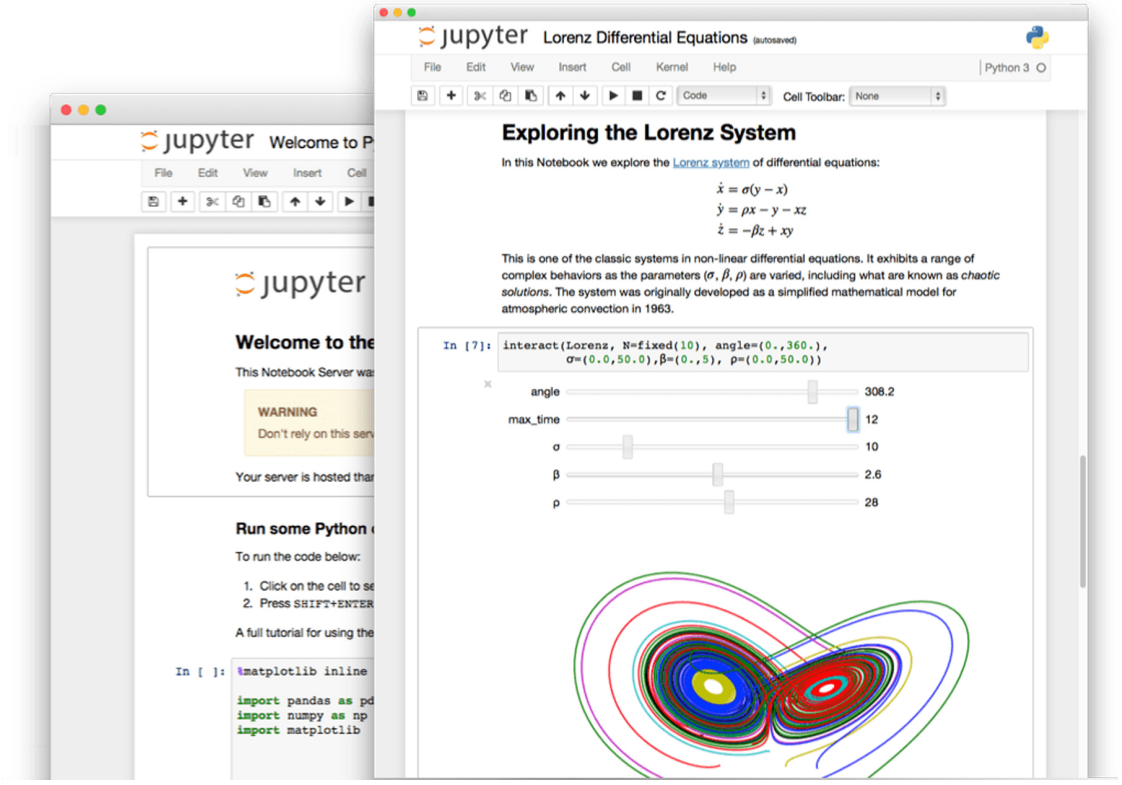


FIGURE 2.1: Examples of notebooks and data visualization in Jupyter

Jupyter is easy to be installed, it can be deployed on a server and used through web browser. Notebooks can become a presentation since Jupyter Notebooks allows adding paragraphs showing both computer code (e.g. Python) and rich text elements (paragraph, equations, figures, links, etc...).

2.1.2 Apache Zeppelin

Zeppelin[2] is an open–source project that enables interactive data analytics. It is web–based and it was mainly born to bring, analysis and visualization on a larger scale of data using Apache Spark[3].

[2] Apache Zeppelin Official Website: https://zeppelin.apache.org/
[3] Apache Spark Website: https://spark.apache.org/

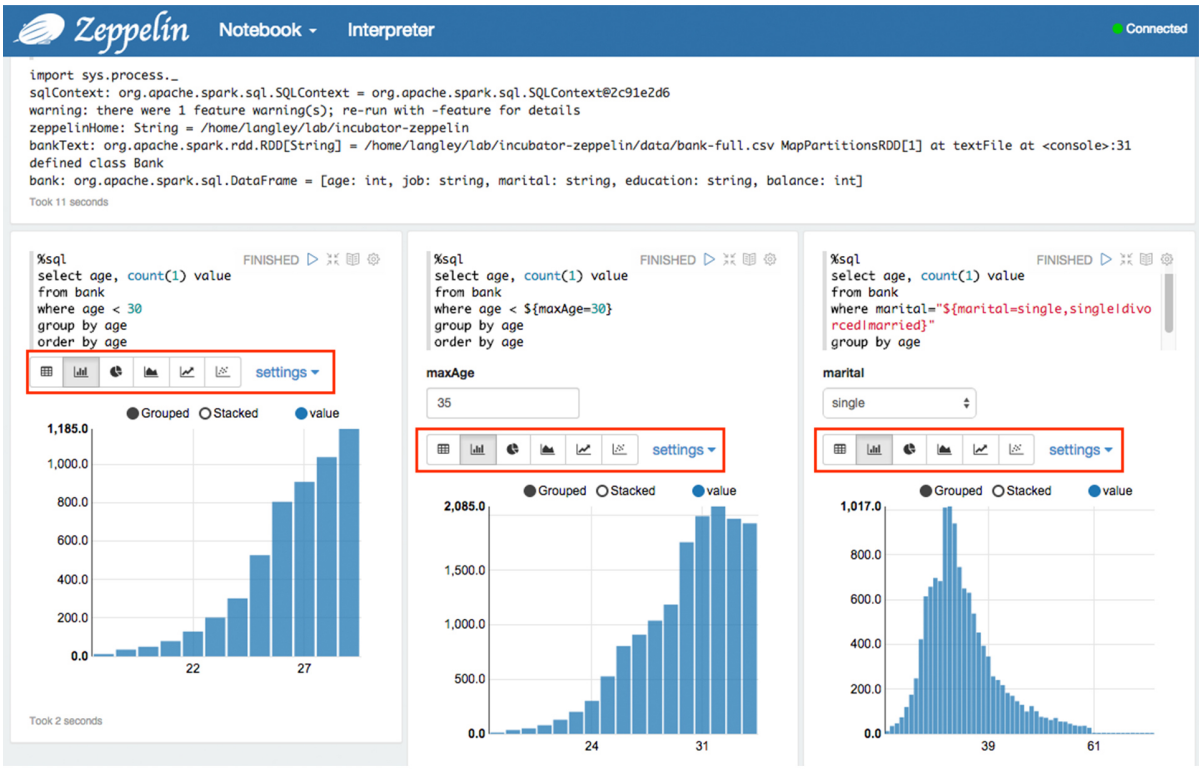


FIGURE 2.2: Apache Zeppelin provides the user with built–in simple data visualization tool

In addition, Zeppelin provides a built–in simple data visualization tool (as shown in the red boxes in Figure 2.2): it is possible, for example, to visualize the same data in various ways (as a bar chart, scatter plot, line chart or tabular) by clicking on different icons. Zeppelin functionalities can be extended by installing plugins.

2.1.3 Conclusion on Data Analytics Tools

Compared with *Jeeves*: both Jupyter and Zeppelin are notebook-based, it is complex to create and analyze data, as a matter of fact one needs programming knowledge. Also, interaction with the data is very limited and users need to perform them using a programming language.

2.2 Data Visualization

Data Visualization tools let the user visualize and interact with data. Unlike the data analytic tools, most of these tools do not require to write source code to create meaningful visualizations. We now describe and compare two data visualization tools: Tableau and Apache Superset.

2.2.1 Tableau

Tableau[4] is an analytics platform for data: it allows people to visualize and interact with the data even without a technical background in programming or data science. This tool is known mainly due to its simplicity of use and ability to produce interactive visualizations. Differently from the tools presented so fare, Tableau is not web based: it is a desktop application, therefore it is necessary to go through an installation process before using it.

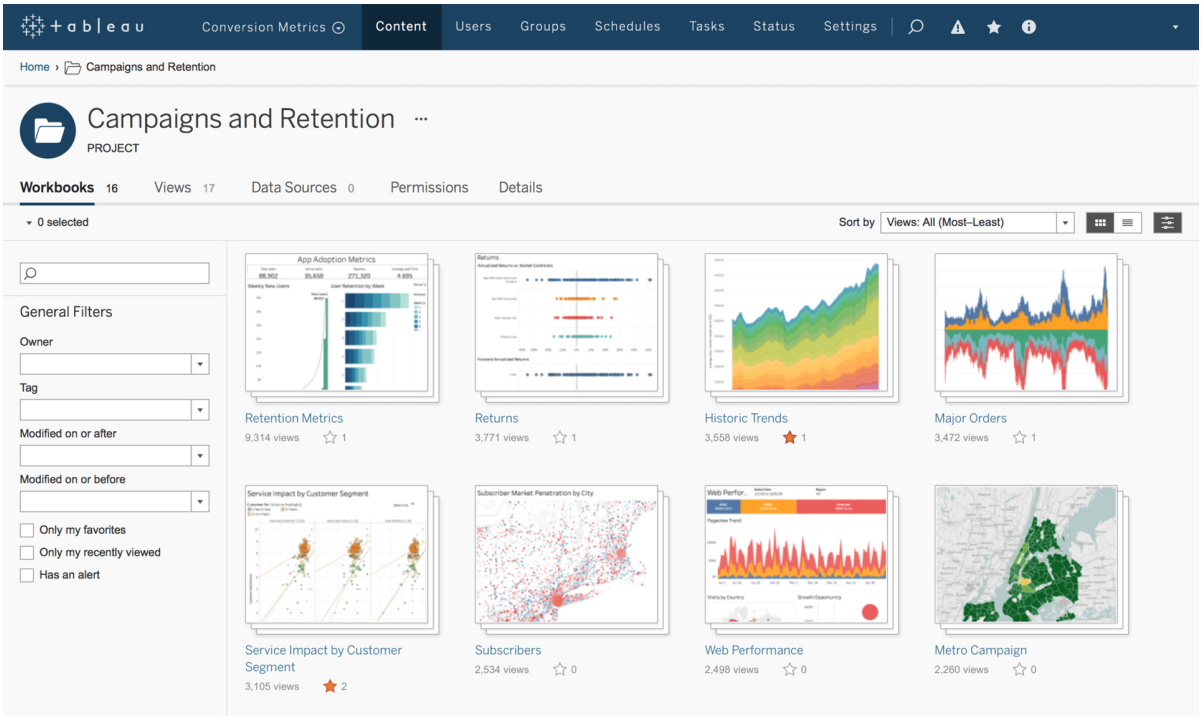


FIGURE 2.3: Tableau User Interface

As shown in Figure 2.3, Tableau provides workbooks that can be saved and accessed at any time. Workbooks are linked to datasets and, in those, one can perform queries and visualizations through an intuitive and easy to use user interface with a system of drag and drop that let users put data into the system and watch it updates in real-time.

2.2.2 Apache Superset

Superset[5] provides data exploration and visualization through the use of an intuitive interface to explore and visualize datasets, and create interactive dashboards. Also, it comes with out–of–the–box visualizations to show the data selected by the user.

[4]Tableau Official Website: https://www.tableau.com/
[5]Tableau Official Website: https://superset.incubator.apache.org/

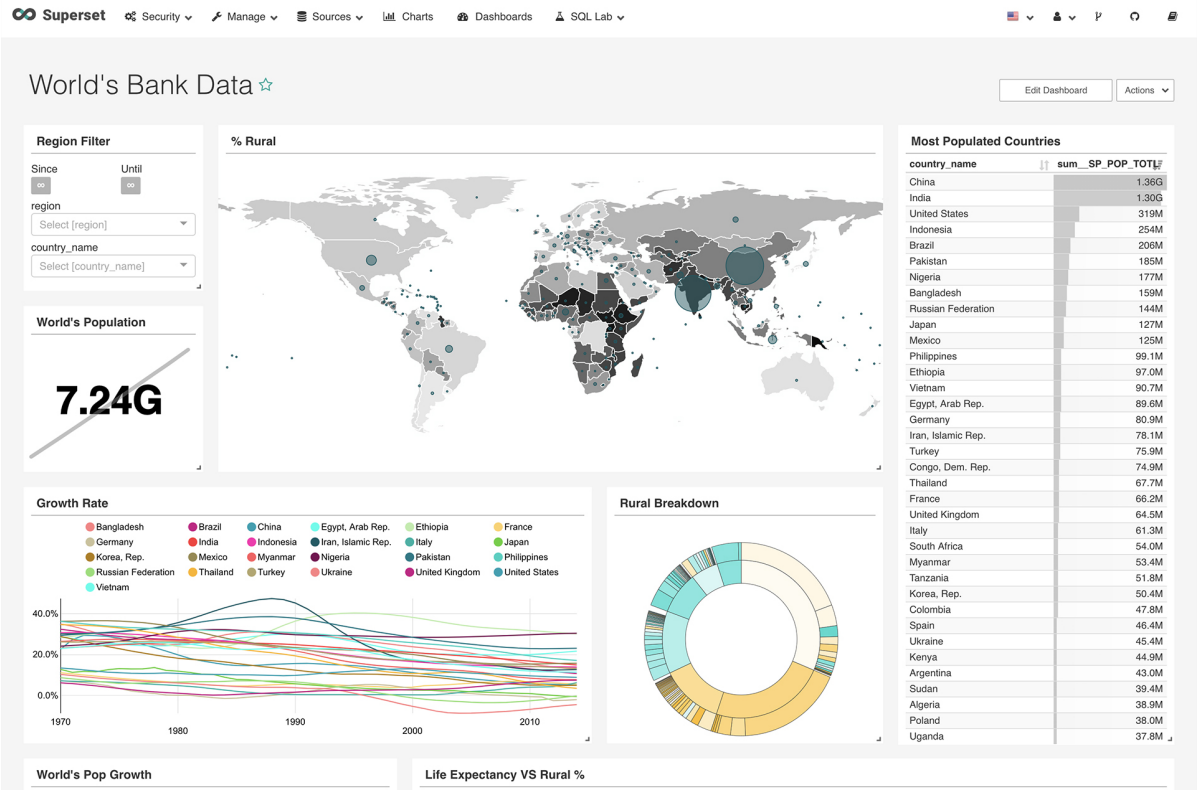


FIGURE 2.4: An example of Dashboard in Apache Superset

Similarly to Tableau, also *Superset* provides various visualizations and the knowledge in programming or SQL is not required but it is harder to deploy since it requires to setup and configure the back–end.

2.2.3 Conclusion on Data Visualization Tools

For the development of our tool, we started by analyzing and trying these dashboarding tools. We took into account the most interesting ideas and we suggested an even more high–level approach in which it is possible to create visualizations through vocal commands.

Tableau offers some of the ideas behind *Jeeves* like the fact that it does not require the user to write code and that it suggest the optimal visualization based on the type of data to visualize. While Tableau pushes to interact with the data by using a computer, we aim to provide the user with an even more novel approach to the visualization of data: adding the possibility to interact directly from vocal commands. Furthermore, we want to push towards the possibility to create highly customizable visualizations by the user.

In *Superset* can be difficult sometimes to apply customized visualization without diving into the source code. Our goal with *Jeeves* is exactly the one to provide the user with a comprehensive and still easy–to–use system for building custom visualization of the data.

2.3 Pure Visualization Chart Generator

In this section, we present tools that are targeted to software developers and data engineers. The ones we present are mostly written in JavaScript and are often used in designing front-end

web pages. For the development of *Jeeves* we used *ECharts*[6]: an open–source data visualization library which provides many types of charts and dynamic visualization effects. An alternative to ECharts is *Highcharts*[7]: a visualization library offered as a paid service that comes with more means of support.

Another chart generator tool is *Plotly.js*[8]: a library built on top of d3.js[9] that provides an high-level set of APIs to build data visualization on a web platform; among the features provided by this library there are 20 different chart types, 3D charts, statistical graphs, and SVG maps.

2.4 Summary

We chose a set of characteristics that we consider to be meaningful for data analysis and visualization tools. We show if each tool:

- Has Data Analysis capabilities: it is important to pre-process data before showing it;
- Provides Data Visualization tools: as explained in Chapter 1, it is essential to understand trends and patterns in data;
- Is Open–source: it is important since the community can contribute to the development of the tool;
- Requires coding skills: important to understand if the tool is accessible by non–expert users;
- Is Easy to setup: if using the tool does not require an installation or, if present, is easy to perform. Useful to lower the barrier of accessibility;
- Provides basic charts (i.e., bar charts, histograms, box plots, scatter plots, etc...): a standard data visualization tool must provide the key components for representation of data;
- Provides Map Charts: it is an interesting type of visualization if the dataset contains coordinates;
- Provides simple interactions like drag & drop to make data visualization process easier.

In table 2.1 we summarize the main features of all the tools we analyzed.

[6]ECharts Official Website: `https://ecomfe.github.io/echarts-doc/public/en/index.html`
[7]Highcharts Official Website: `https://www.highcharts.com/`
[8]Plotly.js Official Website: `https://plot.ly/javascript/`
[9]D3.js Official Website: `https://d3js.org/`

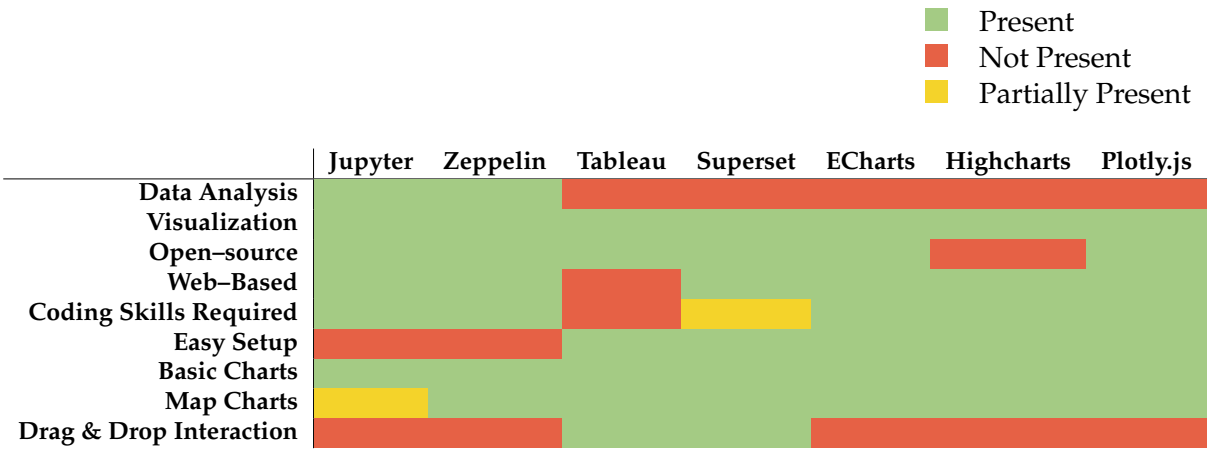


TABLE 2.1: Comparison of data visualization and analysis tools

2.5 Working Experiment: *BoardBile*

During the summer of 2018, we developed a tool called *BoardBile* together with my colleague Leonardo Iandiorio and in collaboration with CodeLounge (a center for software research & development). The aim of this platform was to present data about food for a special event occurred in Lugano called "Lugano Cittá del Gusto" [10]. We created a tool divided into two main components: a big screen that was showing a dashboard containing data visualizations and a smartphone that was acting as a remote control.

The main problem we had to face for this event was how to present to people of different backgrounds technical data and make them easily understand without any previous training. In order to make this process intuitive and seamless, we came up with the idea of letting users query the data by using a tool that everyone would be familiar with: their own smartphone.

Users would connect their personal smartphone to our system and they were able to navigate and get insight about the data we decided to present. To this aim, the user were performing actions like moving the device left and right to change visualization or tapping on the screen to change the settings of the chart and visualizing the results of their query on a bigger screen All of this in real time.

[10] Lugano Cittá del Gusto Webpage: http://www.luganocittadelgusto.ch/

FIGURE 2.5: *BoardBile* presented to the public during the event "Lugano Cittá del Gusto 2018"

In the context of the project called "Show Me the (food) data"[11], BoardBile helped the visitors of the event to receive some useful information about the consumption and the composition of food all over the world.

Part of the inspiration for *Jeeves* comes from the feedback and ideas we gathered during this experience. This event showed us that this kind of interaction with data may work, so we decided to go a step further by experimenting another approach to data visualization: we transformed the paradigm of interaction between screen and smartphone into an interaction between screen and vocal assistant and, instead of providing some datasets of our choice, we let the user ingest any kind of dataset in our system.

2.6 The interaction–based visualization of *Jeeves*

Data analysis and visualization has a very active community and it offers many tools. Our aim in the development of *Jeeves* is to build a tool that not only provides an intuitive visualization but, above all, gives users the possibility to interact with the data itself through a system of familiar queries like vocal commands.

In this work, we take in consideration some components provided by the state–of–the–art tools (like Tableau and Apache Superset): we work towards the direction of lowering the access barriers to data visualization, and we propose an intuitive and seamless approach to it.

[11] Find more at: http://www.luganocittadelgusto.ch/it/event/101/show-me-the-food-data/

On top of that, we added the feature of inferring the optimal visualization based on the shape of data. This way, the user is not required to know concepts of data visualization as *Jeeves* suggests the most effective data representation depending on the type of data that needs to be visualized.

Chapter 3

Jeeves

This section presents our approach: *Jeeves*. We present the features of our tool and also discuss requirements we decided to follow for the implementation of the tool. The first part discusses the results we obtained using some examples datasets. We present the user interface that we built and we explain how to use it and how we allow the user to perform queries on the data using natural language recognized by a vocal assistant.

In the next section we present a discussion about how the experiments we ran were useful to identify some problems in our approach and we explain how we overcome these challenges.

3.1 Our Approach to Voice User Interface

Jeeves provides users with the possibility to visualize and analyze datasets using vocal queries. Our idea is to provide both a traditional User Interface (or *UI*) and to experiment a Voice User Interface (or *VUI*). This latter represents the innovative aspect that we propose in this master thesis work.

Our goal is to lower the barrier to data visualization by providing the possibility to query data simply by using vocal commands. Also, our tool answers to the queries by providing the best way to visualize different types of data (e.g., categorical or numerical). To this aim *Jeeves* analyzes the dataset and, based on the shape of data, suggests the optimal visualization.

We decided to implement the guidelines on the subject of data visualization that has been introduced by Edward R. Tufte in his seminar work called "Envisioning Information"[32] and by Stephen Few in his book "Show Me the Numbers: Designing Tables and Graphs to Enlighten"[17].

Therefore, the tool is able to: 1) answer to vocal queries, 2) analyze their columns and, using guidelines taken from the literature, 3) choose the best visualizations based on the type of columns that the user chooses to visualize.

3.2 Requirements

We extracted some requirements from our previous experiment with the tool *BoardBile* (see Section 2.5) and decided to respect these guidelines for the implementation of *Jeeves*.

As already mentioned in Section 1, our tool must be easily accessible by every type of user, the UI must be simple to use and data must maintain its ownership. Our tool does not only provide a standard User Interface but also a Voice User Interface, for this reason we decided to set some requirements for the correct implementation of this part. We decided to follow the guidelines provided in the documentation of the Amazon Alexa Developer Console that we consider being one of the most exhaustive and detailed about the topic of VUI.

This sections explains the importance of these requirements and how *Jeeves* respects them.

3.2.1 Accessibility

We implemented *Jeeves* as a web application to which one can interact through the use of voice thus we respect the requirement of accessibility. In particular:

- By creating an application accessible through the web browser we ensure easy interaction with the tool: users do not need to install services to access the application;
- By allowing the user to perform queries using their voice: we allow them to use one of the tools they are more familiar with.

The ideal scenario for *Jeeves* consist in having the web application deployed on a screen next to a vocal assistant device. The user needs to get closer to this setup (i.e., in the range of the microphone of the vocal assistant) and start performing vocal queries to both interact with the UI and to create visual representations of the dataset directly on the provided dashboard.

3.2.2 Simplicity

Simple sites remove all unnecessary elements and unnecessary complexity from their design [27]. This helps users to easily use the platform without getting lost into useless or too complex designs.

To allow users of all backgrounds (i.e., experts and not–experts) to use *Jeeves*, we kept the user interface to the essential elements. We provide the essential views and menus that allow the user to:

- Add their dataset into the tool;
- Create sessions in which they can generate and personalize their dashboard;
- Link the datasets to a room;
- Create different data visualizations in each graph of the dashboard.

In Section 3.4, we show views of the application and we explain how to interact with the tool to create visualizations.

3.2.3 Maintained Ownership of Data

Data ownership is a crucial topic since it describes who owns the data or who might be a custodian of the data [18].

One big question with cloud implementations is, who owns the data? In *Jeeves* data resides on the same platform where the server is running. Therefore, the idea that we propose with this tool, sees each user running a version of the server on their personal machine and not on a centralized cloud service. This way, even if datasets loaded into Jeeves may contain sensitive data, the ownership is not shifted to a cloud server but remains unvaried.

Moreover, it is important to clarify that not even the vocal assistant service has access to the data. One may think that the voice service we use needs to have knowledge about the data to query, instead the only information that are received from the Alexa Amazon Cloud are the commands to query the data, not the content of the data itself.

3.2.4 Voice User Interfaces (VUIs) Requirements

As previously introduced in Chapter 2, one of our main aims was to experiment in this new field of interaction: the user can visualize data and interact with the web application through the use of vocal queries. To achieve this, we implemented a voice user interface (VUI). The rise of voice UIs denotes a new trend in the interaction between a human and a computing machine.

As opposed to interactions that relies on the standard use of mouse, keyboard or touch controls, VUIs change the entire design approach shifting into a voice-first type of interaction. To achieve a meaningful and seamless experience, one should plan VUIs to adjust to the numerous ways users may express intents through natural language.

To assure the creation of a meaningful VUI, one should follow some design concepts that allows the creation of a more natural and user–centric dialog. These concepts are mainly four:

- **Be adaptable**[1]: let users speak in their own words. An adaptable–designed VUI understands and process a customer's request appropriately, in whatever situation your interface has outlined;
- **Be personal**[3]: individualize the entire interaction. Collect information about the context in which the VUI is operating and create a personalized experience depending on the user;
- **Be available**[2]: collapse the menus; make all options top-level. Build a horizontal VUI with a voice-first design that keeps all options open for users;
- **Be relatable**[4]: Talk with the user, not at the user. The vocal assistant needs to speak concisely to help the user understand what information your VUI needs and to feel confident about what is happening.

These four requirements are essential to design a VUI that provides an engaging interaction between a human and a computer[1].

The topic of VUI is becoming more and more relevant as even big and international companies like Apple, recently decided to invested on this field and decided to provide their own vocal–based system called "Voice Control"[2]. This new feature, that Apple provides on both MACOS and iOS, has been presented during their annual World Wide Developer Conference[3] (WWDC) in June 2019 in California.

3.3 Use Cases

We validated the tool by allowing people to use it. To explain how *Jeeves* works, we think that the best way is to go through a number of questions and show how our application can help us answer to them.

To this aim, we use a dataset that we show and explain more in detail in Appendix A. Before showing the capabilities of our tool, we formulate some questions about the datasets and we show how we are able to answer them only by using vocal commands in *Jeeves*.

[1]New Alexa Design Guide: https://developer.amazon.com/it/docs/alexa-design/get-started.html
[2]Voice Control by Apple: https://apple.co/2ZazIOF
[3]WWDC: https://developer.apple.com/wwdc19/

3.3.1 Questions about the "Diamonds" Datasets

We use the "Diamonds" dataset (find more in Appendix A.1) to test the capabilities of *Jeeves*. We are able to ingest it in the system and create visual representations of the data contained in it.

To analyze the data, the user needs to setup the environment by adding the dataset to the platform, creating a session (or room) and linking the dataset to the session (we show these steps in Section 3.4). Furthermore, it is possible to personalize the dashboard by choosing the size of the grid, for example by saying "Create a 3 by 4 grid" *Jeeves* creates a dashboard containing a grid with 12 cells in a $3x4$ disposition. Now we are ready to answer the following questions:

1. **Once uploaded in the system, can I create an overview of the dataset?**
   In *Jeeves* it is possible to create a summarization of the dataset by using the command "summarize" (or "summarize the dataset"). This vocal command allows the user to visualize an overview of the dataset, inspecting, column by column, the distribution and the most frequent value.

   Jeeves goes through every column and choses between two different types of visualization: if the column contains numbers (or numerical values) it chooses to use histograms, if the column contains strings (or categorical values) it creates a bar chart. Figure 3.1 shows the result of the summarization command.

FIGURE 3.1: Result of the Summarization Command

From the result, we can observe that the dataset is composed of 10 columns (from cell 0 to cell 9) and we are left with two free cells that can be filled at any time. Each cell shows as a title the name of the column that is being visualized and the distribution of the elements in each column in the shape of bar charts (in blue) and histograms (in red).

2. **How many different color can a diamond have in our dataset?**
   To answer this question, we need to focus on the correct cell. We can do it by inspecting the title of the cell and by selecting the one we are interested in by using the command "open cell 2". The zoomed cell view shows more information (i.e., labels, axes and description), as shown in Figure 3.2.

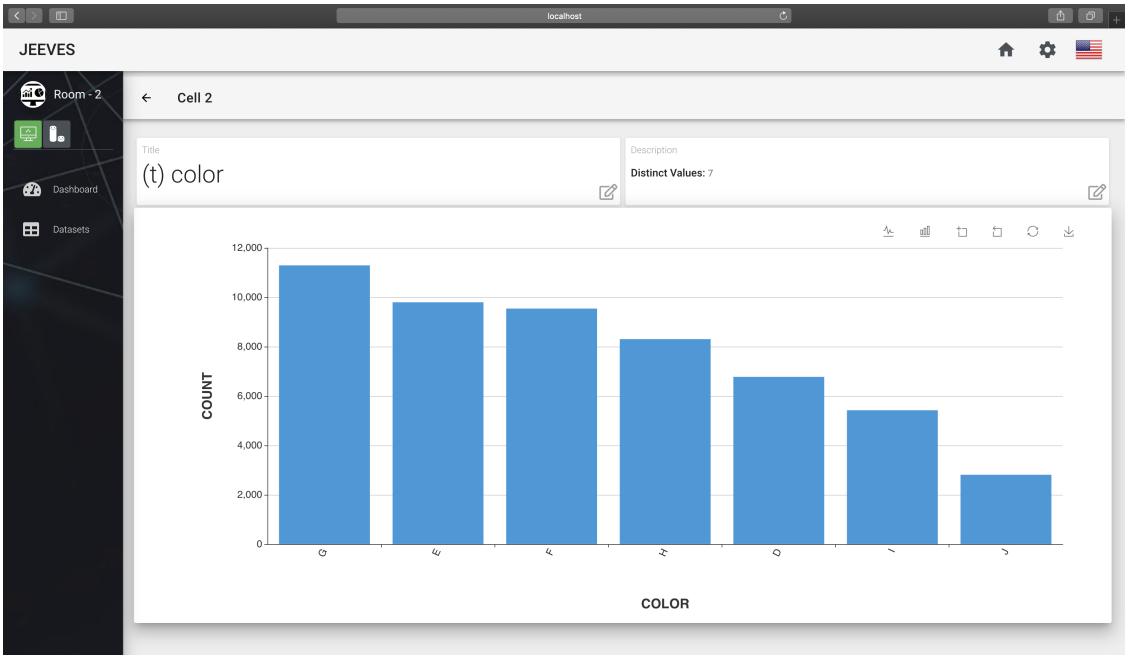
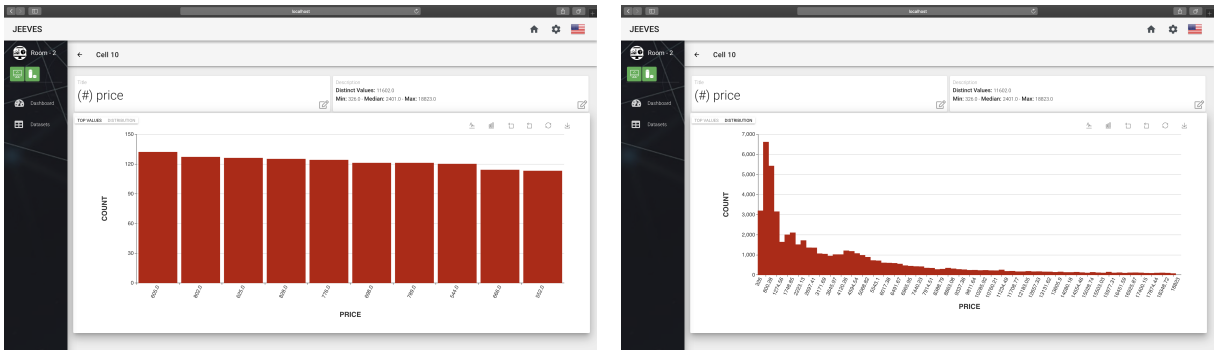


FIGURE 3.2: Cell showing the values of "color" in the dataset

This cell provides both a visual representation of the occurrences in the "color" column (in the form of barchart) and the number of distinct values in the description field. Therefore, to answer the question, a diamond can have **7** different possible colors. We can see that the distribution is smooth: columns go from "J" (worst) to "D" (best) where the most frequent is "G" with more than 10′000 occurrences, "E" and "F" have almost the same number of occurrences each and the least frequent is "J" with around 3′000 occurrences.

3. **How does the distribution of prices in this datasets look like?**
   Since we want to answer to a different question, we need a new cell to interact with. To this aim, we perform two commands: 1)"show grid" (i.e., go back to the grid view) and 2)"open cell 10". Once we are in the cell view, *Jeeves* provides information about the first five rows of the dataset (as in Appendix A.1.3). Since we are interested in the "price" column, we ask to the vocal assistant to "show price" (or "plot price"). By default, *Jeeves* provides a histogram showing the top 10 occurrences in the column as shown in Figure 3.3a. The tool analyzes the data in the column, counts the occurrences and sort them in descending order: we can see that more than 120 diamonds in the dataset cost 605 dollars and, in general, in the top 10 visualization the price varies from around 800 dollars to 552 dollars.



(A) Top 10 values sorted by frequency

(B) Distribution of values

FIGURE 3.3: Cell showing the "price" column in the dataset

Since we are interested in the distribution of values, we can ask to "show distribution". This way, the visualization is updated in real time (see Figure 3.3b) and can finally see how the distribution of price varies in our dataset. As seen in the top 10 visualization the distribution is skewed towards left and most of the diamonds in the datasets ranges in a price between 326 dollars and 1274 dollars. The more the price goes up, the fewer are the diamonds with that price.

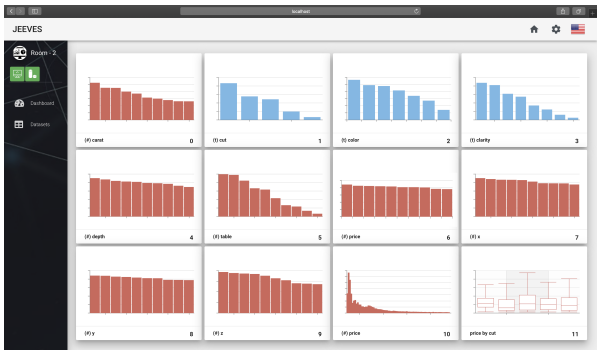
4. **How does the cut of a diamond influence its the price?**
   We now need to focus on the first available cell. Therefore, we perform the two queries "show grid" and "open cell 11".

   To answer the given question we need to ask to our vocal assistant to "show price by cut". As shown in Figure 3.4, the visualization that *Jeeves* suggests us is a box plot: this way we can inspect the distribution of the price of diamonds grouped by the different types of cut.

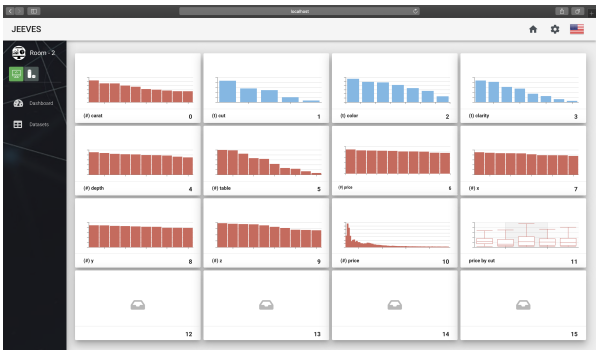
FIGURE 3.4: Cell showing the distribution of price by cut

This result shows us that the diamonds with a "Premium" cut span a bigger range of prices, followed by the ones with a "Very Good" cut. Diamonds with a cut "Fair", "Ideal" and "Good" span , more or less, the same range of prices with significant differences only on the position of the median.

5. **What is the correlation between the carat of a diamond and its price?**
To answer this question, we need a new free cell. We can perform the command "zoom out" to show the dashboard but, in its actual status, it does not seem to provide free cells. We can solve this by asking the vocal assistant to rearrange the dashboard: by saying "create a grid 4x4" we have now 4 new cells ready to be focused (or zoomed) and filled with new visualizations. As visible in Figure 3.5a and Figure 3.5b, the dashboard maintains the status of the previously added cells and fits other 4 cells in the same amount of space.



(A) Dashboard with a grid of size 4x3



(B) Dashboard with a grid of size 4x4

FIGURE 3.5: Cell showing the "price" column in the dataset

Now, we can focus on the first free cell by saying "open cell 13" and create the visualization we need by saying "show carat vs price". The visualization that gets created is visible in Figure 3.6.

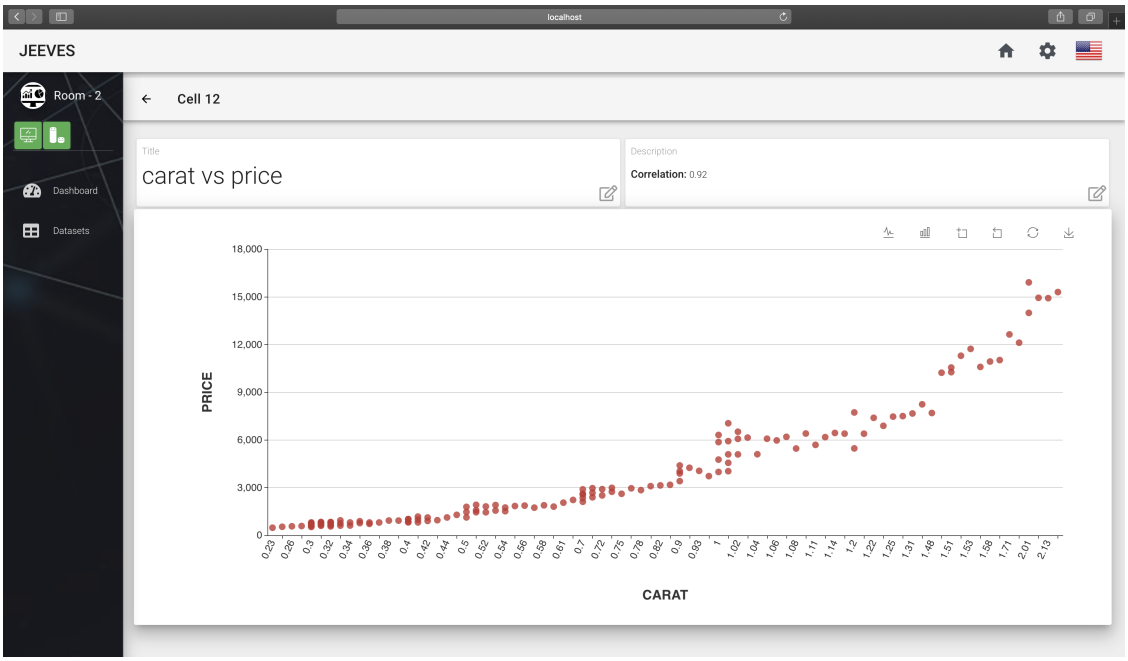


FIGURE 3.6: Cell showing the distribution of price by cut

It is clear from the result that these two columns are highly positively correlated: this means that (as one may expect), the highest value of carats the diamond posses, the highest will be its price. To support this interpretation, *Jeeves* also computes a numerical value called Pearson Correlation Coefficient [13] and puts it in the description of the chart. We can then conclude that the correlation between the the carat of a diamond and its price is equal to "0.92".

In this section we showed how users are able to answer to question about a dataset by using *Jeeves* and its vocal queries. In the next section we describe more in detail how we build each view provided on the web application.

3.4 Usage of the Tool

Being a visual analysis and dashboarding tool, the frontend represents the entry point of our entire work. The most important part of *Jeeves* is, in fact, the visualization of data: the client side needs to be deployed on a big screen (since the visualization on a smartphone screen would not be efficient, we left it outside of the scope of this work) and it represents the "canvas" in which is possible to create data visualization through the use of few clicks and vocal commands.

We created a hybrid approach to data visualization in which it is possible to use both traditional means like keyboard and mouse for complex operations like the uploading of files and innovative ones like the voice for performing queries on the dataset.

3.4.1 Dashboard and Cell View

The core section of the entire graphical user interface is the room page. The user can access it by clicking on the card of one room listed in the Home page shown in Section 3.4.2.

Once the user enters the room page for the first time, it is requested to link the room to one of the available datasets as shown in Section 3.4.3.

A new room that has been linked to a dataset looks like the one in Figure 3.7. To create the dashboard, the user defines vocally the dimension of the grid (up to a maximum of 4 by 4 to ensure a big enough dimension of the cells).

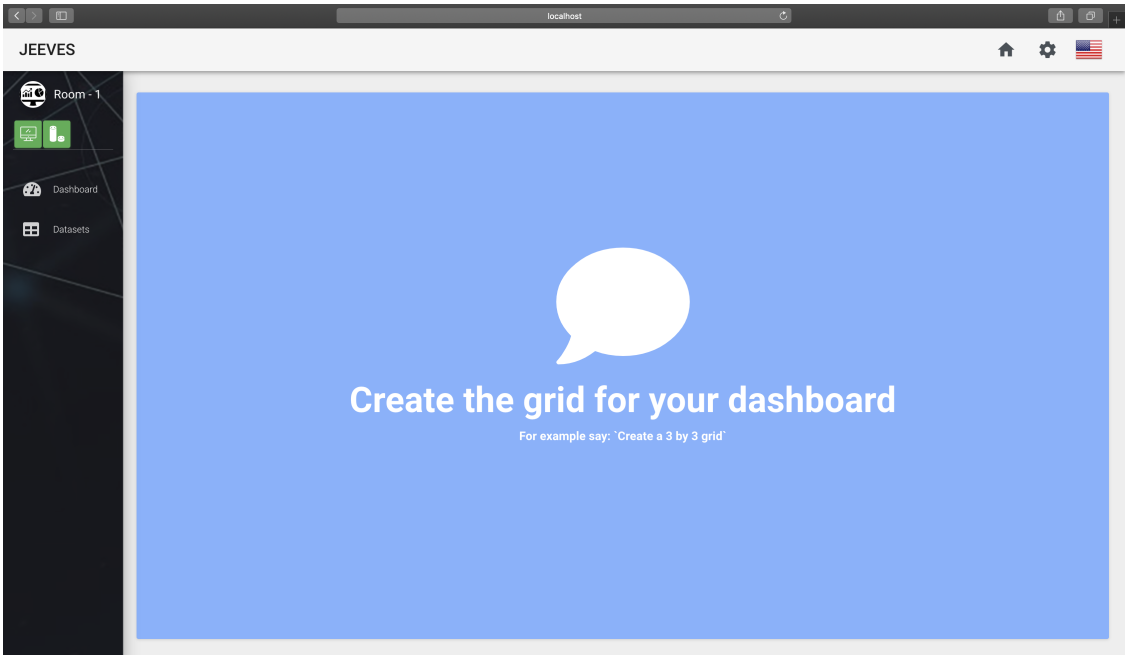


FIGURE 3.7: The dashboard in a room before the grid is initialized

Once the grid dimension has been set, the dashboard will show a list of empty cells that can be filled with charts (as shown in Figure 3.8) using vocal queries.

FIGURE 3.8: The dashboard in a room after the grid of dimensions 3*x*3 is initialized

If, instead, a room that already contains charts is opened, the grid will look like the one in Figure 3.9.

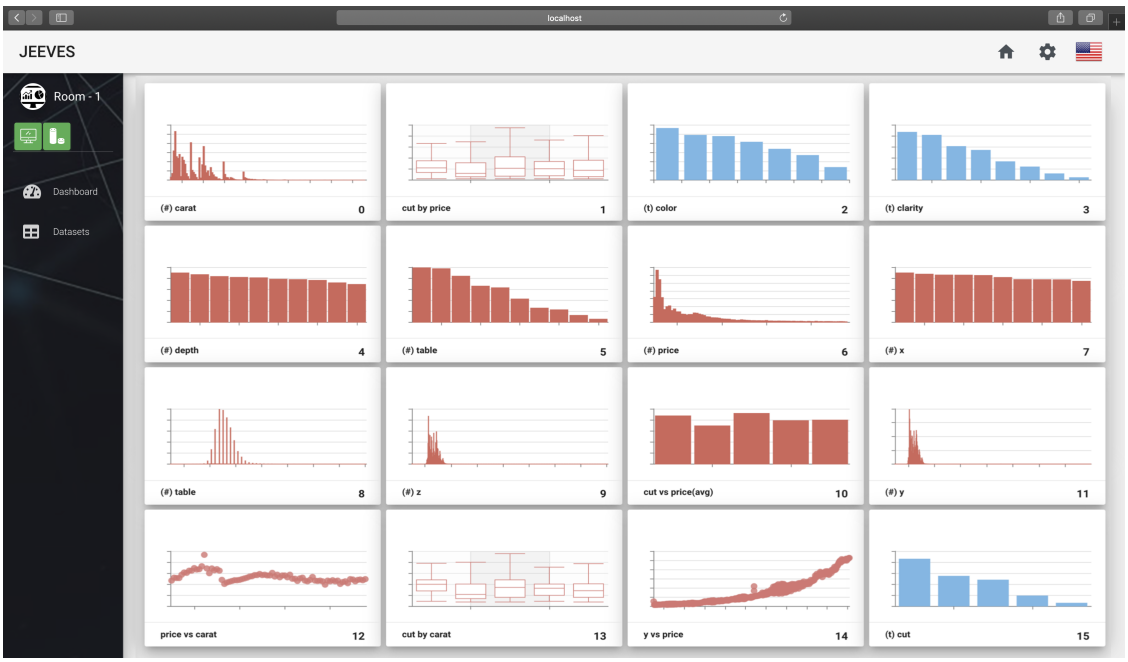


FIGURE 3.9: The dashboard in a room containing charts

Whenever a cell is focused (either using a click of the mouse or using vocal commands), *Jeeves* will show two different views: if the cell has not been initialized, it shows the header of the currently active dataset in the room (see Figure 3.10).

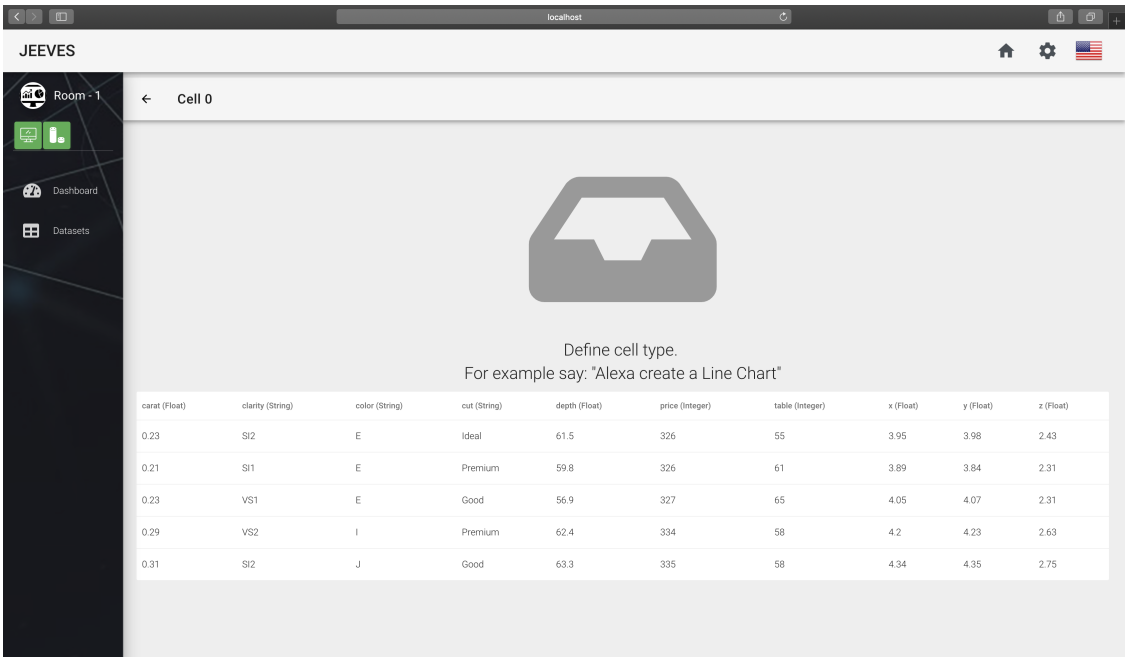


FIGURE 3.10: Visualization of a focused cell not initialized

If the cell has been initialized, *Jeeves* shows a more detailed version of the cell in the dashboard (as shown in Figure 3.11): title and description are shown together with a chart that shows labels on the axes and possibility to interact with the chart using mouse controls.

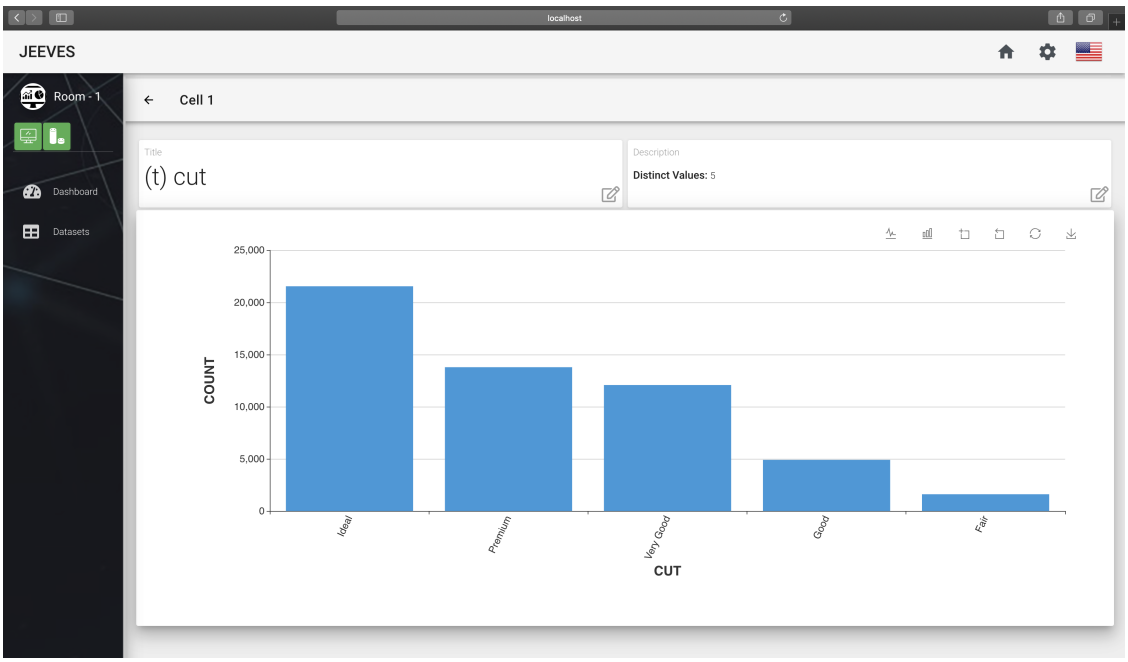


FIGURE 3.11: Visualization of a chart in a cell

3.4.2 Home Page

The landing page of *Jeeves* consist in a view called Home. As shown in Figure 3.12 we provide:

- The list of available rooms with name and connection statuses to understand which devices are connected to which room. These statuses are important since we allow one user only and one vocal assistant to be connected to one room at the same time.
- A "+" button to add a new room when clicked
- A settings button that allows the user to go in the panel to add new datasets in the system

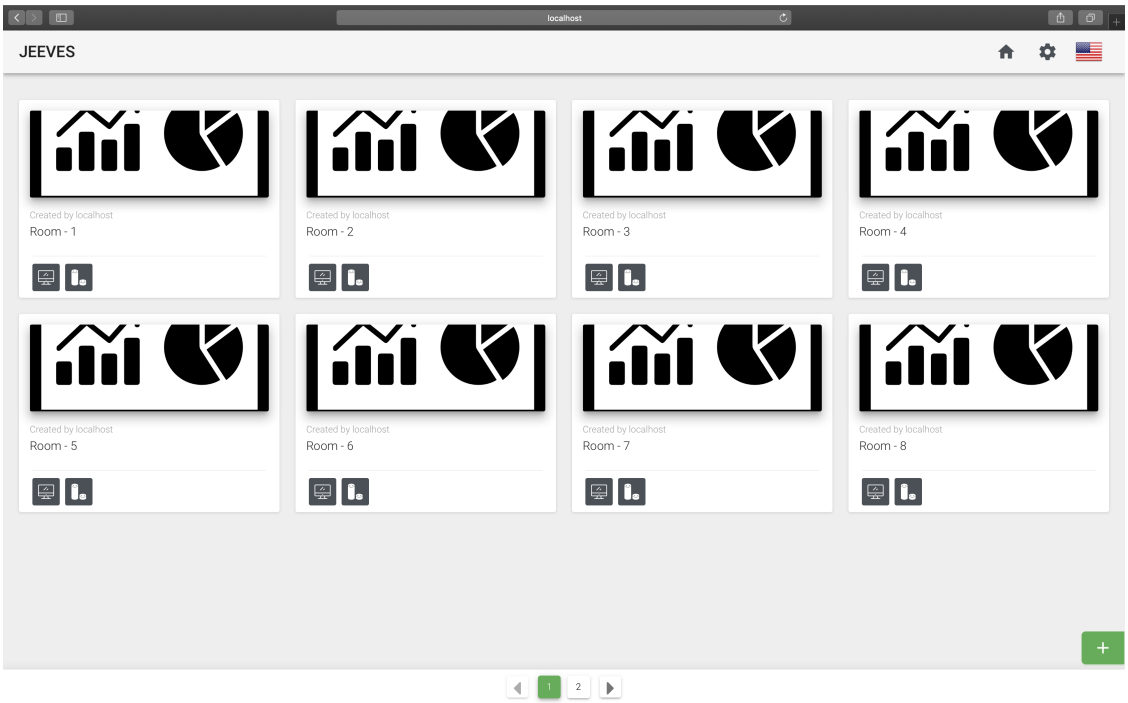


FIGURE 3.12: Home page of *Jeeves*

3.4.3 Dataset Upload and Selection

Figure 3.13 shows the "Manage Datasets" page. Here we show the list of loaded datasets with the possibility to:

- Visualize the information about the dataset by clicking on the "eye" icon
- Delete the dataset by clicking on the "x" icon
- Upload a dataset from the local machine of the user by clicking on the "+" button

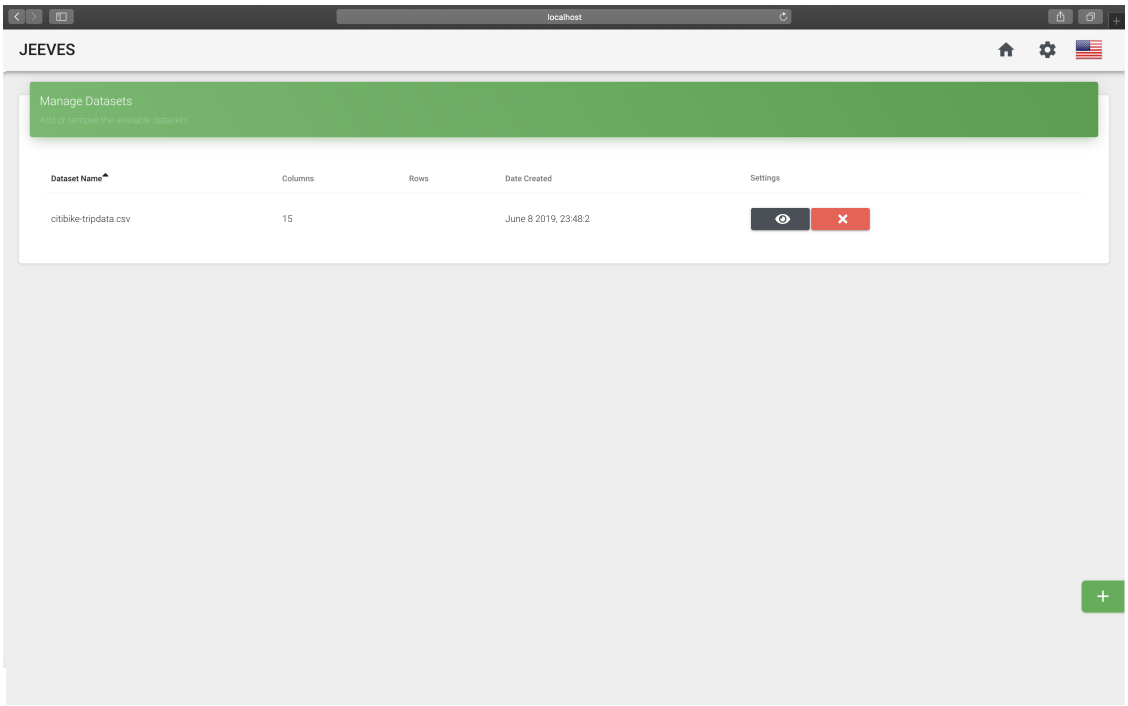


FIGURE 3.13: Manage Datasets page of *Jeeves*

To upload the file (in .csv format) the user clicks on the add button and selects the file that will be sent and analyzed by the *Jeeves* server as shown in Figure 3.14.

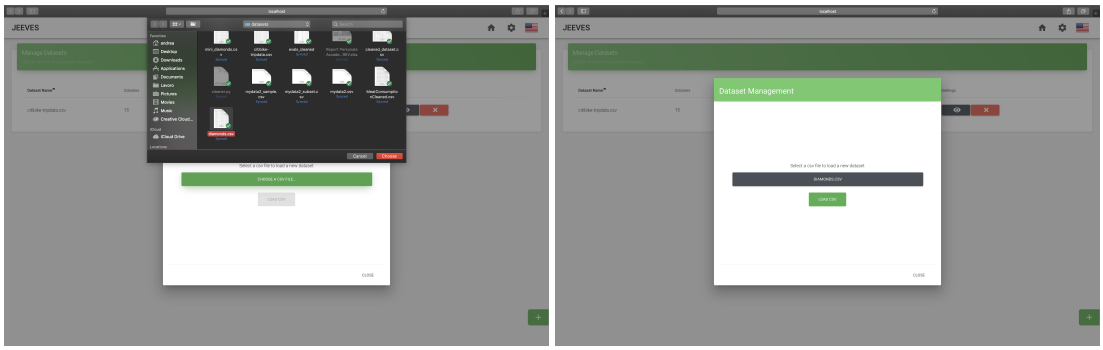


FIGURE 3.14: Add datasets in the modal of *Jeeves*

Once the file is analyzed by the server, the loading page in the interface is substituted with a window showing a summary of the dataset. In this interface (shown in Figure 3.15) it is possible to confirm or modify the name and the types of each column as inferred by the *Jeeves* server.

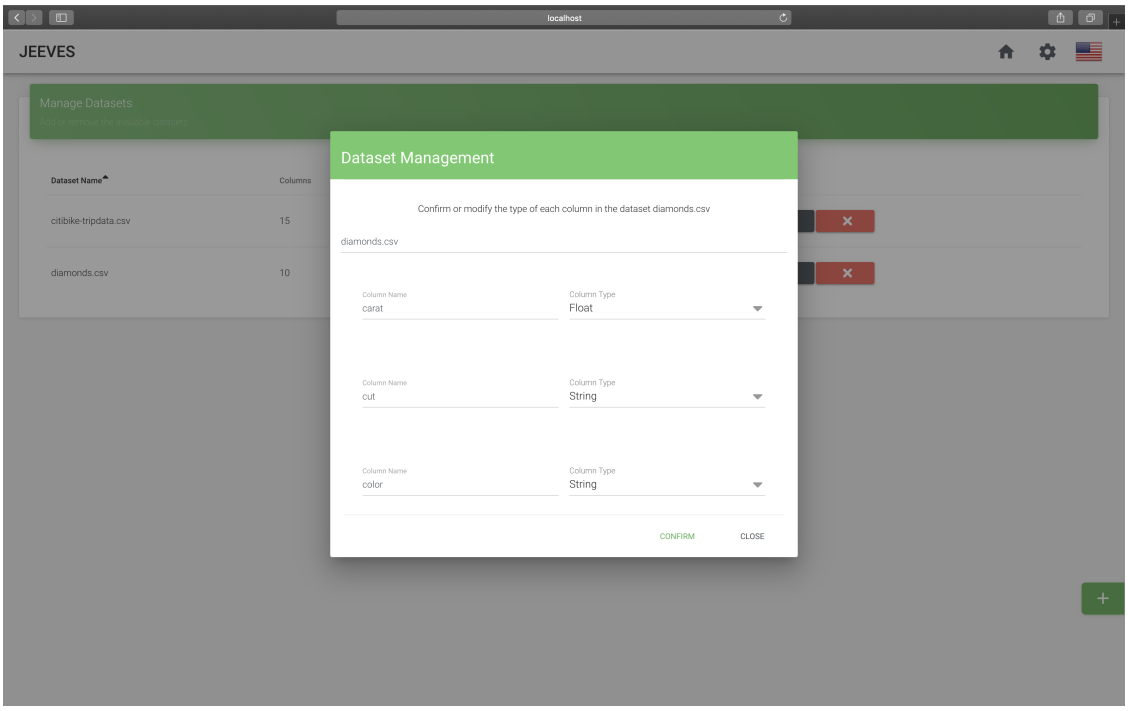


FIGURE 3.15: Modal that lets the user confirm or modify name and type of the columns of the uploaded dataset

Similarly, inside the room section, it is possible to select one of the available datasets to link them to the current room (as shown in Figure 3.16). This means that, from that moment on, all the queries will refer to the selected dataset.

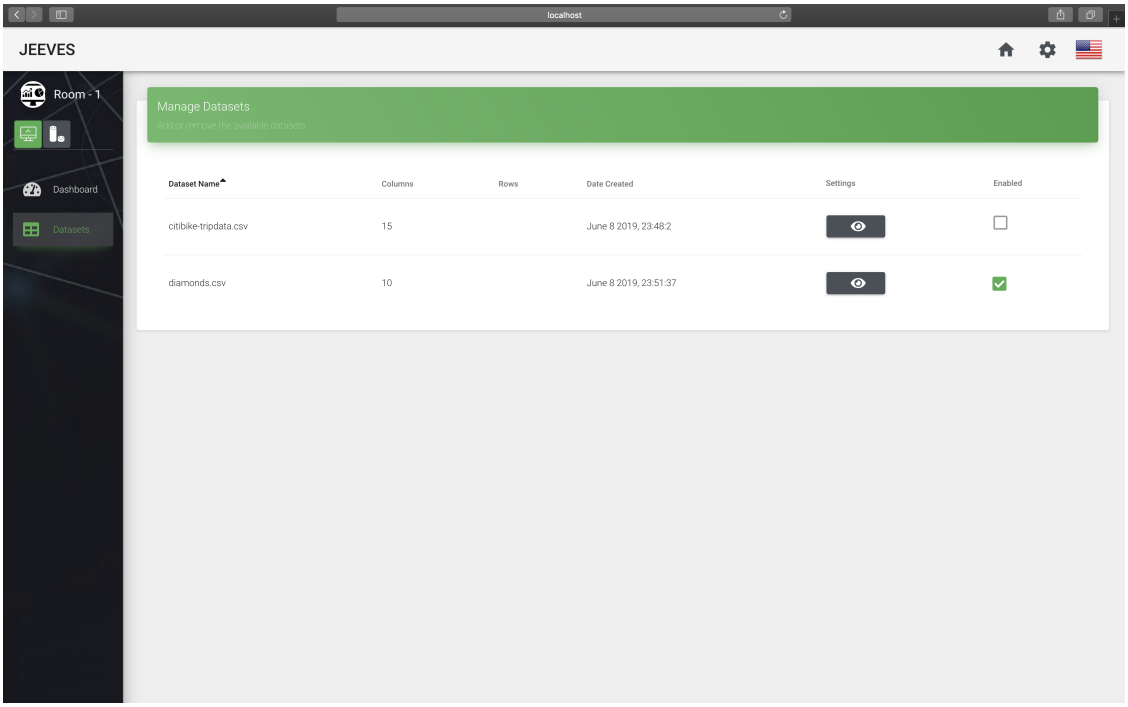


FIGURE 3.16: Linking datasets to rooms in *Jeeves*

3.5 Development Challenges

During the development of *Jeeves* we performed some demos with users in order to perform some anecdotal evidence. During these demos, we encountered unexpected behaviors in the conversation between user and vocal assistant and in the recognition of some column names. Therefore, we implemented some fixes to respect the criteria described in Section 3.2.4 about building a meaningful and working voice user interface. We wanted to:

- Ensure *relatability* by giving the user the feeling of having a seamless conversation with the vocal assistant
- Ensure *adaptability* by making sure that the Alexa skill is able to respond to a variety of user requests trying to overcome possible errors in the pronunciation or in the ambiguity of the language

3.5.1 Relatability

We needed to handle *asynchronous queries* in a Visual User Interface approach. One of the requirements that a good vocal user interface has to provide is a constant feedback: we did not want user to perform vocal queries and wait for seconds without any kind of answer from the vocal assistant.

When performing queries over big datasets, our server needs some seconds before providing a response. At first, the user was waiting without receiving any feedback until the response was ready.

To solve this issue, we used a functionality provided by the Alexa Skills Kit SDK for Java called "Progressive Response". This feature allows us to provide a constant feedback to the user while the server is working (e.g., the vocal assistant tells the user that the server is working for creating the chart) before showing the final result of the query. We present more details about the implementation of this solution in Section 3.6.1.

3.5.2 Adaptability

During the development of *Jeeves*, we faced some difficulties during the process of natural language recognition. It is, in fact, a known problem that not always the vocal assistant understands exactly the words that the user says [7]. This happens because spoken language can be ambiguous, spoken voice inputs are made more complex by accents, background noise, talking at the same time [7].

In Section 3.6.2 we explain how we overcame this issue introducing in our system an approach based on the Levenshtein distance that is used for measuring the difference between two sequences of words.

3.6 Solution to the challenges

In Section 3.5 we introduced the challenges we faced during the development of *Jeeves*. We explain how we decided to handle these challenges by explaining the solutions we implement on the server–side.

3.6.1 Asynchronous Queries

Every Alexa skill needs to receive a response from the server in no more than five seconds otherwise it would communicate that an error has occurred. In our application, when a user was vocally analyzing or querying big datasets, the response time was taking more than five seconds and, therefore, we were providing a bad user experience in which the vocal assistant was communicating that an error had occurred but the result of the query was still being visualized in the dashboard after a couple of seconds.

In order to solve this problem, we used a feature provided by the Amazon Alexa APIs called "Progressive Response"[4]. This type of response allows to send an intermediate vocal feedback to the user, while our server is still creating the visualization.

With this implementation we provide a more *relatable* approach (see concepts in Section 3.2.4). In Figure 3.1, we present a transcript example of a conversation between the user and the vocal assistant.

```
User: "Alexa, summarize the dataset"
// 1 second of delay
Alexa/Jeeves: "Processing summarization, please wait..."
// some seconds of delays
Alexa/Jeeves: "Showing summary of the dataset"
* Dashboard shows the correct visualization *
```

LISTING 3.1: Conversation between user and Vocal Assistant using progressive responses

On the server side we implemented an asynchronous answer that gets executed if the final answer of the skill is not returned under 1 second.

In the code shown in Listing 3.2, we show how we use the feature of progressive response:

```
1  public SpeechletResponse handleIntent(...) {
2   // Some code for initialization
3       ...
4       Runnable task = () -> AlexaUtils.dispatchProgressiveResponse(request.
            getRequestId(), AlexaUtils.addVoiceSSML("Processing Summarization, please
            wait"), AlexaUtils.getSystemState(context), directiveService);
5       ScheduledExecutorService executor Executors.newSingleThreadScheduledExecutor();
6       executor.schedule(task, 1, TimeUnit.SECONDS);
7       Jeeves.appear().findSessionById(roomId).ifPresent(session1 -> {
8        // Some Code for creation of summarization
9            ...
10           roomRepository.save(room.get());
11           executor.shutdown();
12           ...
13       });
14       // Some code
15  }
```

LISTING 3.2: Code that uses Amazon Alexa's progressive responses for asynchronously creating the summary of the dataset

As visible from the above code, in lines 4 to 6, we start an asynchronous task with a timeout of 1 second that, if expired, will execute the progressive response. If, otherwise, the final answer of the skill gets computed before this period of time, in line 11 we shutdown the task and return the actual response.

[4]Progressive Response Documentation: https://developer.amazon.com/it/docs/custom-skills/send-the-user-a-progressive-response.html

3.6.2 Natural Language Ambiguities

We faced this problem when, during the development of *Jeeves*, we were testing our platform using the dataset about diamonds (find more in A.1). We wanted to show the distribution of "cut". The vocal assistant kept understanding "cat" and therefore no visualization was created. For this reason we decided to introduce a factor of tolerance that would allow to show the desired results even if the Vocal Assistant was not 100% accurate in recognizing the sentence said by the user.

In order to introduce this tolerance, we needed a way to compute a score of similarity between the available columns names and the sentence recognized by the vocal assistant. We chose to compute the "Levenshtein distance": a string metric for measuring the difference between two sequences of characters. The Levenshtein distance between two words is the minimum number of single-character edits (insertions, deletions or substitutions) required to change one word into the other.

To choose the optimal value to accept when comparing two words using the Levenshtein distance, we did some experiments based on all the columns of the dataset and, we notice that the value 2 was not giving us any false positive result. Therefore, we decided that a Levenshtein distance up to 2 was to be taken as a good value.

Therefore, after having introduced this computation, the workflow for creating a visualization was working as follows:

1. User says "Plot cut";
2. Vocal assistant understands "Plot cat";
3. The server computes the Levenshtein distance between the understood word (i.e., "cat") and the names of the columns in the dataset;
4. Among all the columns we just take into account only the ones with a value ≤ 2 and, among them, we pick the column with the name that has the smaller Levenshtein distance with respect to the word understood by the vocal assistant.

In this case we were able to tolerate inaccuracies in the pronunciation of the user or ambiguities of the languages.

Selected
Candidates

Column Name	L.D. w.r.t. "cat"
carat	2
clarity	4
color	4
cut	1
price	5
depth	4
table	4
x	3
y	3
z	3

TABLE 3.1: Levenshtein distance computed between the word "cat" and the names of the columns of the reference dataset

We also implemented the recognition of columns with name composed by multiple words. To this aim, we needed to compute the Levenshtein Distance of every combination of contiguous words. Table 3.2 shows how we managed to select the correct columns when the user says "Plot start station name" in the Dataset "New York Citi Bike Trip"A.2:

Selected

Column Name	start	start station	start station name	station	station name	name
bikeid	6	12	17	6	11	5
birth year	8	10	15	9	10	9
end station id	11	8	9	7	8	12
end station latitude	16	14	11	13	10	17
end station longitude	17	15	12	14	11	18
end station name	13	10	5	9	4	12
gender	6	13	17	7	11	5
start station id	11	3	14	9	10	15
start station latitude	17	9	6	15	12	19
start station longitude	18	10	7	16	13	21
start station name	13	5	0	11	6	14
starttime	4	6	9	4	7	6
stoptime	5	9	12	4	7	6
tripduration	10	7	12	7	11	11
usertype	6	11	15	6	10	7

TABLE 3.2: Levenshtein distance computed between the combinations composed by the sentence "start station name" and the names of the columns of the reference dataset

3.7 Summary

This chapter describes *Jeeves*: our web application to visualize and query data through the use of vocal queries. We first illustrated the requirements we extracted and how we applied them in our tool.

Then we presented some use cases in which we show how our tool allows answering to questions regarding a dataset and how *Jeeves* suggests the optimal visualization based on the shape of data to visualize.

We further describe the main challenges that we faced during the development and the demonstration of the tool and explain how we overcame them with efficient solutions.

In the next chapter, we focus on some technical details about the implementation of *Jeeves*: we present the architecture, the technologies used and the way each component in our tool interact with each other.

Chapter 4

Implementation Details

4.1 Architecture

Jeeves is a web-based application. Figure 4.1 shows the diagram of the architecture of our tool.

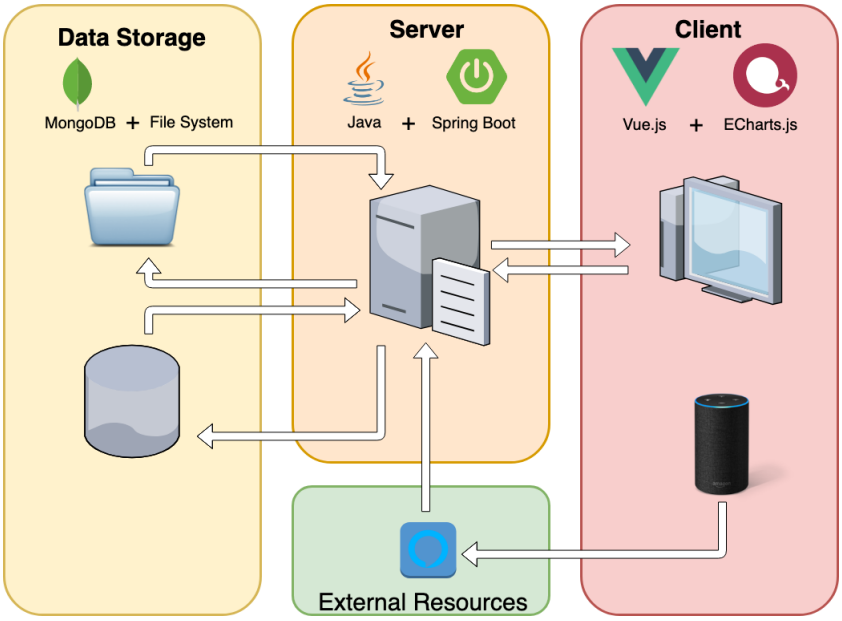


FIGURE 4.1: Architecture of *Jeeves*and interaction of its components

We have a client component that communicates with our server through a set of RESTful APIs. The web application shows the menus and renders the charts while the vocal assistant listens to voice commands and sends them to the external resources.

In the server we perform 1) actions based on the commands sent by the vocal assistant, 2) choose the optimal visualization based on the shape of data and 3) compute the values of the charts so that it can be visualized in the client side.

Finally, the data storage component that stores the information about the datasets and the data visualizations created by the user.

4.1.1 Client

Our client–side component represents the entry point of the *Jeeves* application. It is composed by two devices: one screen running our front–end application and a vocal assistant device (in our case Amazon Alexa).

The application provides the possibility to add datasets through the use of simple menus. Furthermore, the user can create rooms that are environments in which the user can create a personalized dashboard composed of cells containing different type of charts.

Through the use of a vocal assistant, the user is able to perform vocal queries and see the result of the vocal actions directly on the user interface in real time.

4.1.2 External Resources

The server component is connected to the Alexa Developer Console. Whenever the user performs a vocal command using an Alexa device, this assistant sends the request to the Alexa Developer Console that is responsible for generating a response called "intent" that is forwarded to our server. Once the *Jeeves* system receives the intent it will be transformed into an action or visualization that is immediately shown on the user interface.

4.1.3 Server

Our server provides various capabilities. It ingests dataset in csv format, analyzes them understanding the schema and infers the type of each column.

Our server is connected to a vocal assistant service (i.e., Amazon Alexa) and perform actions based on the vocal commands received by the vocal assistant. They can consist in commands to create data visualizations or commands to navigate the web application.

When the user asks to create a visualization, the server analyzes the columns of the dataset and infers the best visualization to show based on the shape of data. Once the type of visualization has been set, the server performs computations and models the data (as explained in Section 4.5.2) such that it can be sent and visualized by the client side.

4.1.4 Data Storage

The data storage component is divided into two main parts: the file system and the database.

The file system is used to store the datasets uploaded by users. This allows *Jeeves* to accept and parse every kind of dataset, no matter how complex the schema can be and we can store the raw data exactly as we receive it from the uploaded file.

The database is used to keep track of the schema and the paths of the datasets stored in the file system (i.e., how to parse them during the analysis) and to store all the visualizations created by the user.

4.2 Technologies

Figure 4.1 shows the technologies that we used for the implementation of our tool. In *Jeeves*:

- The data storage component is build using MongoDB for storing information about active sessions and datasets and using the file system for storing the actual content of the datasets

- The server component is built using the Java[1] programming language alongside with the Spring[2] framework, in particular using Spring Boot[3] that helped to reduce Spring configuration related complexity [10];
- The client is built using the Vue.js[4] framework (together with HTML, CSS and JavaScript) together with the library Echarts.js[5] for the data visualization;
- The Amazon Alexa Developer Platform[6] was used as external resource for recognizing the vocal commands. It sends commands called "intents" that are processed by our server.

4.3 Basic Concepts

It is important to clarify the basic concepts and key components behind *Jeeves* that we mention in this chapter.

In particular, this section describes the purposes of cells, rooms, sessions and datasets inside our tool: which information they contain and how they interact with each other.

4.3.1 Cell

A *cell* is the template of each visualization in *Jeeves*. It contains the following information:

- Title
- Description
- A field "request" containing the information about the request that generated the current cell (in case recomputing the cell is necessary)
- A field "response" containing all the information about the response generated based on the request (it contains the actual data that will be shown in the form of chart)

A cell can be of different types (as shown in Section 4.5.2) each one having different logic and different responses based on the type of visualization we want to show on the dashboard.

4.3.2 Room

A *room* contains information regarding the visualization that are stored in the database. A room is composed by a grid of cells that forms a dashboard-like visualization. The information stored in a room are:

- Name of the room
- Date of creation
- Dimension of the grid (i.e., columns x rows)
- List of cells

[1] Java: https://www.java.com/it/download/
[2] Spring: https://spring.io
[3] Spring Boot: https://spring.io/projects/spring-boot
[4] Vue.js: https://vuejs.org/
[5] Echarts.js: https://echarts.baidu.com/
[6] Amazon Alexa Developer Platform: https://developer.amazon.com/it/alexa

4.3.3 Session

The *session* model contains all the information regarding the current status of a room, they are:

- Name of the reference room
- Status of connection with the screen (i.e., connected or disconnected)
- ID of the connected screen
- Status of connection with the vocal assistant (i.e., connected or disconnected)
- ID of the connected assistant (if connection variable is set as true)
- A status value that declares if the user is currently viewing the zoomed version of a cell with relative ID

4.3.4 Dataset

A *dataset* contains useful information about the structure of the file uploaded by the user. It contains:

- Name of the file
- Reference of the ingested file stored in the File System
- Date of creation
- Counter of rows in the dataset
- Schema of the dataset (each column is composed by a name and a type)

We decided not to store the dataset in the database to have more flexibility in the modeling process: since we want to ingest every kind of schema, we decided to save the actual data into the file system and to store in the database:

1. The path of the file
2. The schema that makes the system understand how to read each column of the file stored.

4.4 Vocal Assistant: Amazon Alexa

Among the numerous vocal assistants in the market, we decided to use the one provided by Amazon, called Alexa. We chose this assistant among the many others because the features provided by this service are easier to integrate with the tools we use. In particular, the Alexa service provides programmatic access to the Alexa features for Java called "Alexa Skills Kit SDK v2 for Java".

Moreover, Amazon provides a platform called "Alexa Developer Console", in which it is possible to create skills that can be directly added to every product that supports the Alexa vocal assistant.

In this section we discuss how we used the Alexa Developer Console for the recognition of vocal queries performed by users and how we introduced the Alexa Skill Kit SDK in our Java server.

4.4.1 Alexa Developer Console

A Skill is a plugin that one user can add to the vocal assistant using the skill store provided by Amazon. Each skill requires:

- An *invocation*: it is the vocal command that the user has to say in order to start the skill

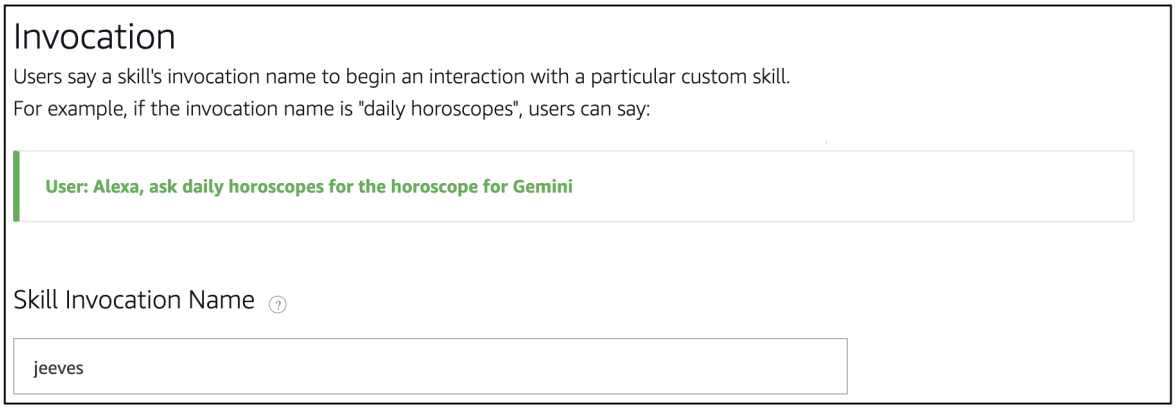


FIGURE 4.2: Invocation commands that starts the *Jeeves*'s skill.

In order to start the *Jeeves* skill, one needs to say "Alexa, open *Jeeves*".

- A set of *slots*: are typed variables that are inserted in the sentences said by the user, recognized by the vocal assistant and sent to our server as parameters of the request.

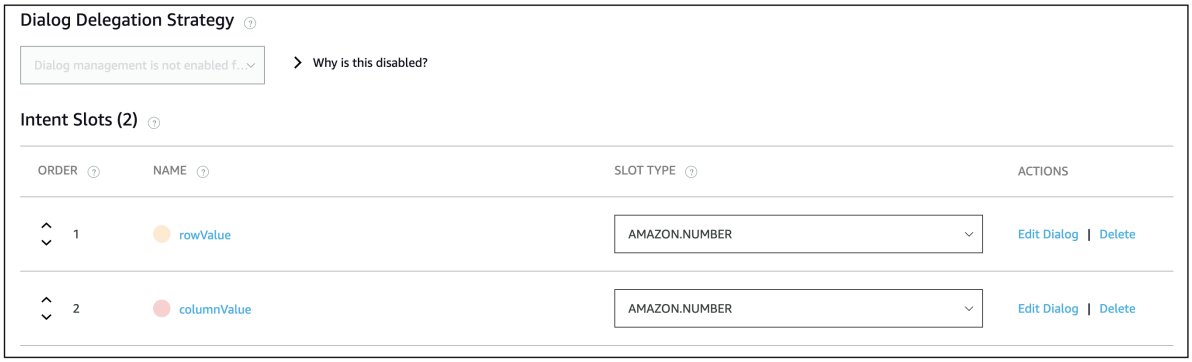


FIGURE 4.3: The two slots used in Figure 4.5 have specified type of AMAZON.Number. It means that, when recognized by the vocal assistant, they will be cast as number values

- Possibility to create *Slot types*: the standard types provided in the Developer Console may limit the capabilities of speech recognition of the vocal assistant. That is why, it is also possible to create custom types with the possibility to link every type with a set of synonyms (adaptability).

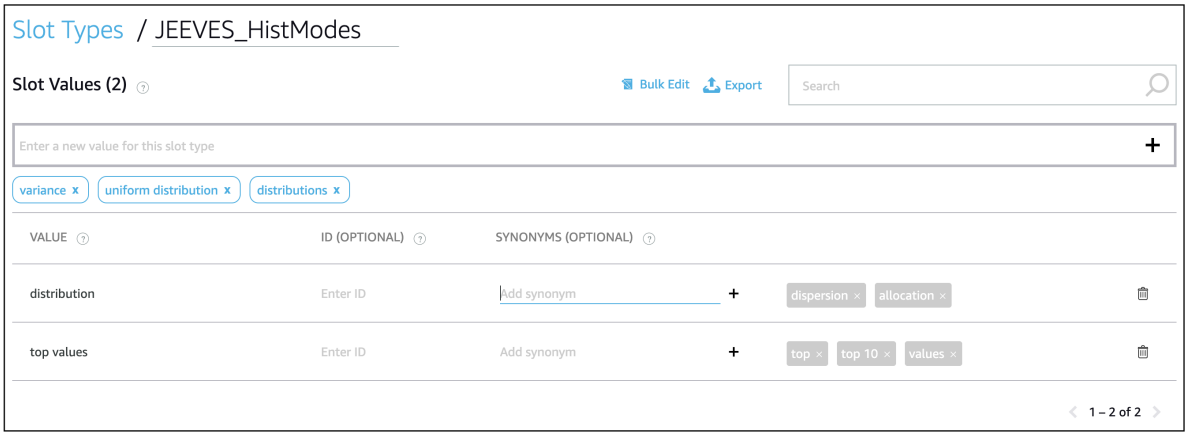


FIGURE 4.4: Creation of custom slot types. Here we defined the type of visualization for the histograms: either distribution or top values (with relative synonyms shown in the grey boxes)

- A set of *intents*: those are the actual command that are supported when the skill is executing. These commands, or sentences, are called *utterances*.

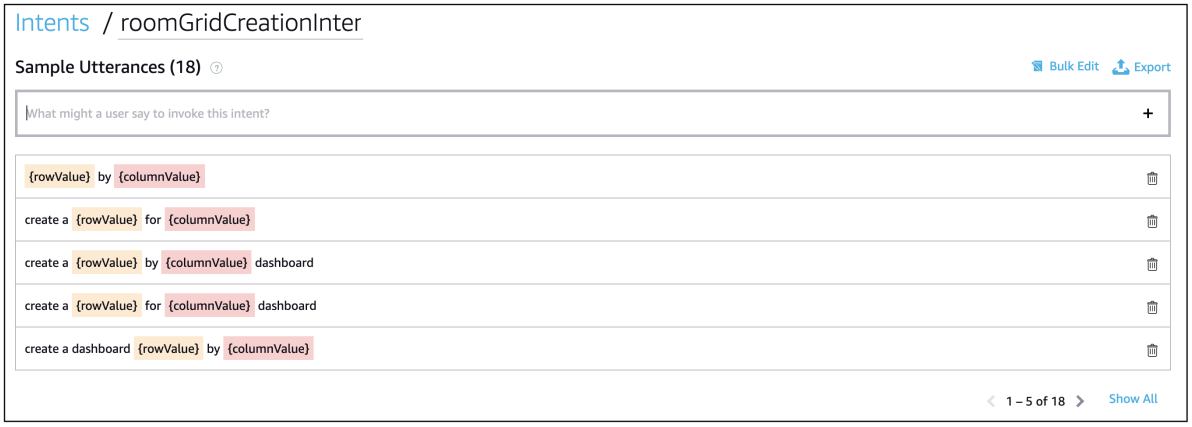


FIGURE 4.5: The creation of utterances allows to match sentences to the creation of specific intents. In the example above it is also visible the use of slots. The sentence will be matched and the slots will be passed as parameters of the intent

If, for example, the user says "Create a grid four by two", then Figure 4.5 shows that a *roomGridCreationIntent* is created in which "four" is interpreted as the slot *rowValue* and "two" as the slot *columnValue* both of type AMAZON.Number (as shown in Figure 4.3).

4.4.2 Alexa Skill Kit: APIs for Java

Amazon Alexa Services allow developers to implement their own skill either as a serverless function called AWS Lambda [7] from Amazon Web Services[8] or using a custom external service that runs over HTTPS.

[7] AWS Lambda: `https://aws.amazon.com/lambda/?nc1=h_ls`
[8] Amazon Web Services: `https://aws.amazon.com/`

In *Jeeves* we do not use AWS Lambda functions since we required a more complex structure for our server. Therefore, we built a custom skill for Alexa by implementing a web service that accepts requests from and sends responses to the Alexa service in the cloud [6]. Quoting the Alexa Developer documentation:

"You can build your web service using any programming language, as long as the service meets the following requirements:

- The service must be accessible over the internet;
- The service must accept HTTP requests on port 443;
- The service must support HTTP over SSL/TLS, using an Amazon-trusted certificate. Your web service's domain name must be in the Subject Alternative Names (SANs) section of the certificate. For testing, you can provide a self-signed certificate;
- The service must verify that incoming requests come from Alexa;
- The service must adhere to the Alexa Skills Kit interface"[6].

We met all these requirements by importing in our Spring Boot server the official Alexa Skills Kit SDK for Java provided by Amazon Alexa as a GitHub Repository[9] and by running a local server that runs over HTTPS. To do so, we used the tool ngrok[10]: a reverse proxy that creates a secure tunnel from a public endpoint to a locally running web service.

To support incoming Alexa Speechlet in Spring Boot using the Alexa Skills Kit SDK for Java, one needs to implement the "SpeechletV2" class and implement the four methods that corresponds to the key element of the lifecycle of any Alexa skill:

1. **onSessionStarted**: This is invoked when a new Alexa session is started. Any initialization logic would go here. In *Jeeves* we use this method to discover and save the current session ID.
2. **onLaunch**: This is called when the skill is started. When this happens, Alexa provides a welcome message and prompt the user to ask a question. In *Jeeves* we use this method to gather the available rooms (the ones not yet connected to a vocal assistant device) and prompt the user to choose which room to connect or to resume the previous session (if exists).
3. **onIntent**: This is invoked whenever an intent is created for the application. Here, the difficulty resides in figuring out which intent is being called, then respond. In *Jeeves* we use this method to dispatch the intent to the correct handler finding in the system the class with the same name of the received intent.
4. **onSessionEnded**: This is invoked whenever the current session ends and it is used to clean up if needed. In *Jeeves* we use this method to disconnect the voice assistant device from the room it is currently connected to.

In Figure 4.6 we present a process view model showing the lifecycle of an Amazon Alexa's skill that interacts with our Spring Boot Server.

[9] Alexa Skills Kit SDK for Java : https://github.com/alexa/alexa-skills-kit-sdk-for-java
[10] ngrok: https://ngrok.com/

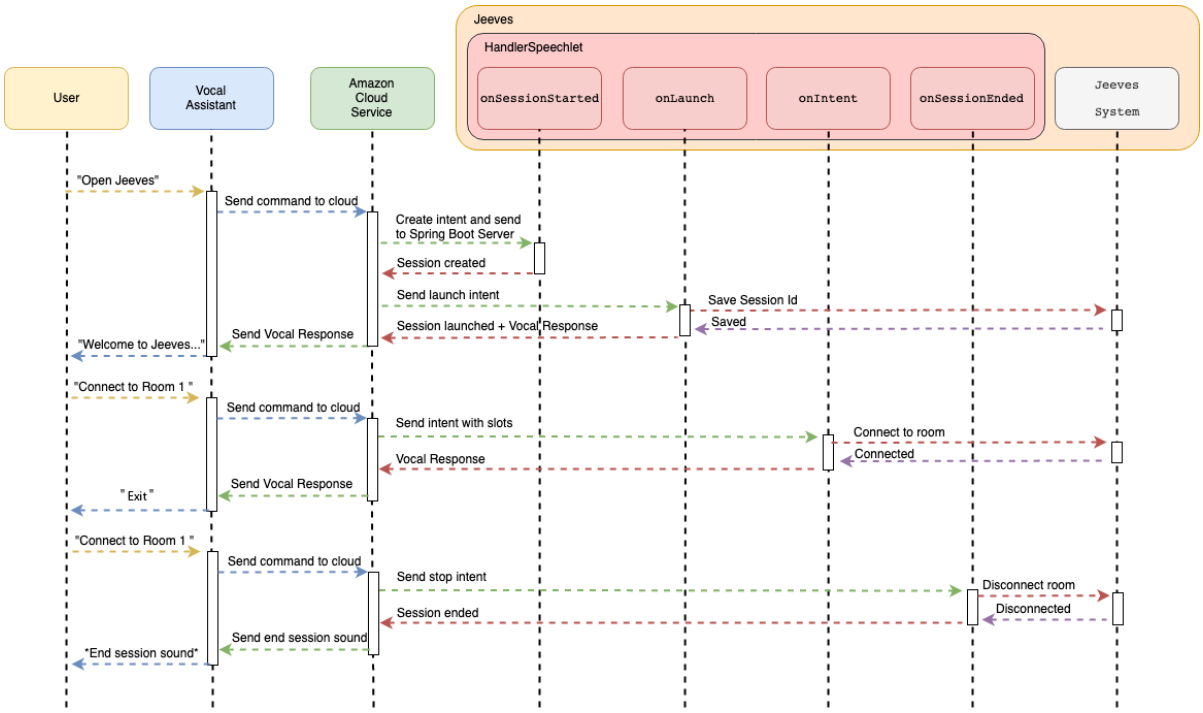


FIGURE 4.6: Process View of the interaction between Amazon Alexa Cloud Service and *Jeeves*'s Spring Boot Server

For example, the first iteration shown in the model starts with a vocal command said by the user containing the wake word for our skill that is "Open Jeeves". This command is sent from the vocal assistant to the Amazon Cloud Service that creates a response called intent and sends it to our server. Once received the intent, *Jeeves* executes the *onSessionStarted* method and saves the ID of the current session and then executes the *onLaunch* to create a welcome message and prompt the user to connect to a room. Similarly, the other two iterations in 4.6 show how the user can trigger an intent to connect to a room and how the current session can be ended.

4.5 Server

In this section we discuss the general structure of the server, how we modeled the data, how we managed the sessions and how we used the Alexa Skill Kit: APIs for Java for making possible the interaction between the Amazon cloud and our server.

In Figure 4.7 we show the logical view of the server in which we highlight the main components and how they interact with each other

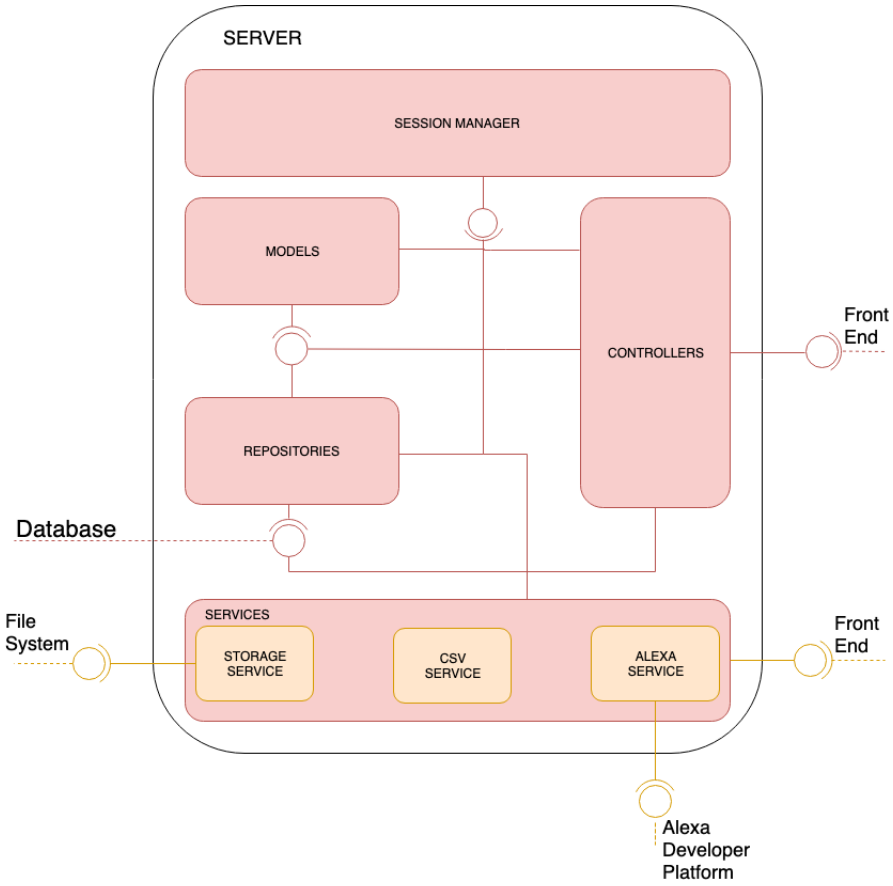


FIGURE 4.7: Logical View[22] of the *Jeeves*'s Server

4.5.1 Session Manager (*Jeeves*)

To maintain an easy and quickly–accessible state inside our application, we create a singleton class called "Jeeves" that, being always available in the entire system, allows to have always at our disposal, information about the current status of the system. Therefore, this singleton acts like a cache and saves us the time of accessing information from the disk as these informations are kept in memory.

For example, whenever the Alexa vocal assistant connects to the server, it needs to know which are the rooms available for connection. In such a case, we just need to write the line of code shown in Listing 4.1.

```
1 List<String> availableConnections = Jeeves.appear().getAvailableRoomsForAlexa();
```

LISTING 4.1: Example of usage of the *Jeeves*'s Session Manager to get all the available rooms

Another case in which we use *Jeeves*'s Session Manager occurs whenever a room opens a Socket connection with the server: we do it because on the client side we want an up-to-date status of the connection of each room. In order to open the socket and save the ID of the connection in the Session object described in section 4.3, we do the following.

```
1 Jeeves.appear().addWebsocketMapping(this.getSessionId(), recipient, sessionID);
```

LISTING 4.2: Example of usage of the *Jeeves*'s Session Manager to add a Websocket connection

Similarly to the use of Web Socket, we also use Server Sent Events (SSE), that can be interpreted as mono–directional Web Socket in which the conversation occurs only from the server to the client. In this case, since the client never sends information to the server, it only instantiates a SSE communication and shows the status of the connection of each room in real time.

4.5.2 Models

In this section, we describe the types of cells that we created and how *Jeeves* chooses the optimal visualization based on the shape of data.

To choose the best type of chart, we perform some analysis of the type of data that we are considering. Quantitative information always consists of two types of data: **quantitative values** (a.k.a. measures) and **categorical labels** (a.k.a. dimensions)][17]:

- *Categorical* variables take on values that are names or labels[11].
- *Quantitative* variables are numerical. They represent a measurable quantity[11].

With this distinction in mind, we now present the different types of data visualization that *Jeeves* offers.

Bar Chart

A bar chart is a chart that represents categorical data using rectangular bars. In a bar chart the height of the bars represents the frequency of the observations in each category [16]. If the barchart is shown vertically, the horizontal (x) axis represents the categories; The vertical (y) axis represents a value for those categories [5].

A bar chart type gets created by *Jeeves* whenever the user asks to visualize columns containing categorical values: we collect all the entries in the selected column, count the occurrences of each entry, sort by number of occurrences in descending order. After this process, we create a title and we insert the number of distinct values in the column as a description of the cell, finally we return the object to the client. The bar chart object, has the structure shown in Appendix B.1.

Histogram

A histogram is a representation of the distribution of numerical data. It is an estimate of the probability distribution of a continuous variable [28]. It consists of contiguous bars that can be represented as both horizontal axis and vertical axis.

The histogram can help to show the shape of the distribution in the data, the center, and the spread of the data [15]. Histograms are used to show the distribution of data.

We used the histogram for visualizing numerical columns. In particular, in this case *Jeeves* offers two different types of visualization:

1. **Top Values**: if the goal is to show the first 10 values sorted by occurrences

2. **Distribution**: if we want to show an overall distribution of the data present in that column.

To limit the number data points to show at once and provide an optimal visualization for our front-end dashboard, we used a standard approach called binning (or bucketing) that helped us reduce the magnitude of data. In order to find the optimal number of bins, we applied the following data-based histogram bin width determination rule, which is known as the *Freedman–Diaconis* rule[33]:

$$h = 2\frac{\text{IQR}(x)}{n^{\frac{1}{3}}}$$

where h is the number of optimal bins, n is the number of observations in the sample x and IQR is the interquartile range which is the difference between the upper and lower quartiles [33].

After this process and after having computed statistics about distinct values, min, median and max in the distribution, the response from the server for the creation of an histogram looks like the one shown in Appendix B.2.

Box Plot

Box plots are a simple but powerful graphing tool that can be used in place of histograms [23]. As shown in Figure 4.8, box plots characterize a sample using the 25th, 50th and 75th percentiles also known as the lower quartile ($Q1$), median (m or $Q2$) and upper quartile ($Q3$) and the interquartile range ($IQR = Q3 - Q1$), which covers the central 50% of the data. Quartiles are insensitive to outliers and preserve information about the center and spread [23].

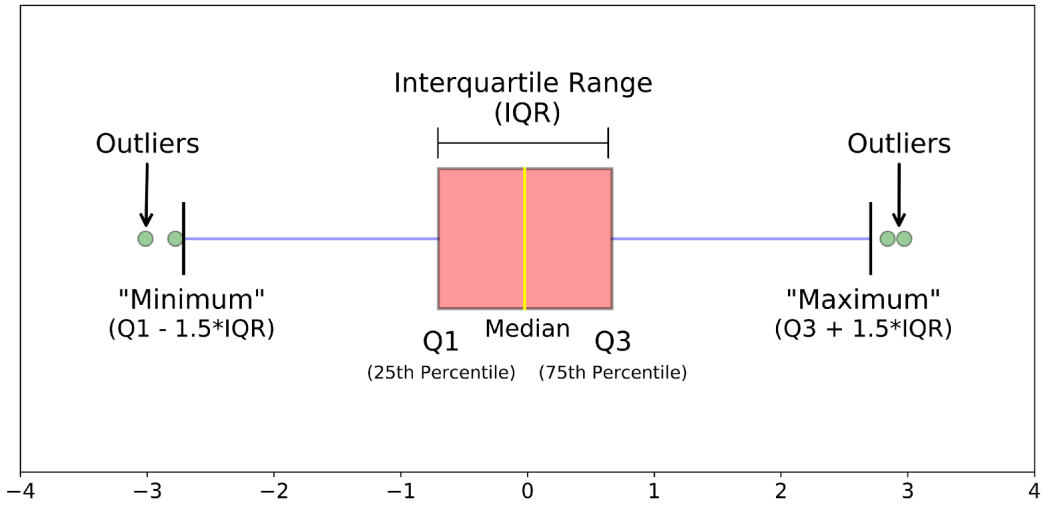


FIGURE 4.8: Example of a box plot

The box plot in *Jeeves* gets created when the user asks to show a numerical variable "by" a categorical one. In such a case, we put each element of the categorical column in the x axis and a collection of collections containing the numerical columns grouped by the category. We set the title of the cell, and finally we return an object that looks like the one shown in Appendix B.3.

Scatter Plot

A scatter plot is used to display the relationship between two quantitative variables. A scatter plot consists of an X axis (the horizontal axis), a Y axis (the vertical axis), and a series of dots. Each dot on the scatter plot represents one observation from a data set. The position of the dot on the scatter plot represents its X and Y values [12].

A scatter plot type gets created by *Jeeves* whenever the user asks to visualize two columns containing numerical values. A well-known problem is that scatter plots can have a high degree of overlap, which may occlude a significant portion of the data values shown[20]. In order to solve this issue we implemented a system to minimize the amount of data transferred through the wire. For the summarization, we again create bins using the *Freedman-Diaconis* rule as explained in Section 4.5.2 in the creation of the Histogram. The scatter plot object, has the structure shown in Listing B.4. Moreover, we compute the Pearson Correlation Coefficient between the two columns; this measures strength and direction of the linear relationship between two variables[26]. We report this value in the description of the cell

4.5.3 Controllers

Controllers are responsible for processing user requests and expose endpoints to the client to provide services[9]. In particular, the main responsibilities of a controller are:

- Intercept incoming requests;
- Convert the payload of the request to the internal structure of the data;
- Send the data to the model for further processing;
- Get processed data from the model and send that data to the View for rendering[8].

In *Jeeves* we created three main controllers: they contain the endpoints that our server exposes to the web application for the visualization of the dashboard and the graph (as also report in Appendix C). They are:

- **CSVController**: that handles the upload and the deletion of datasets in *Jeeves*.
- **DatasetController**: that manages the existing datasets and how to link and unlink them to the rooms
- **RoomController**: this controller is responsible for managing the connections and disconnection to the rooms from both the dashboard and the vocal assistant

4.6 Wrapping Up

In this section we discussed the technical implementation of *Jeeves*: which are the key components, how they are implemented and how they interact with each other.

We presented the Alexa Developer Console and explained how we integrated it in our Spring Boot server by benefitting of the "Alexa Skill Kit SKD for Java".

In conclusion, we discussed the different types of charts provided by our tool and how *Jeeves* infers the best one to choose after the user performs a query.

In the next chapter we wrap up this master thesis work by summarizing the key concepts provided by *Jeeves*, how we validated them and we present the future work we have in mind for a future improvement of this tool.

Chapter 5

Conclusions

5.1 Summary

We start this document by presenting the importance of data analysis and visualization, in particular, how it helps users have insight about data.

We analyze the state–of–the–art tools to compare and focus on the ideas that are present in *Jeeves* and we discuss about BoardBile: the experiment we performed before the implementation of our tool that helped us to gather feedback about the best way to propose data analysis and visualization to people from different backgrounds.

Then, we present the idea that we propose with *Jeeves*: we provide users with a dashboard that can be personalized with data visualizations and built through the usage of vocal commands. In our tool, it is possible to interact and customize charts in the dashboard by talking to a vocal assistant service: in our case we choose Amazon Alexa. Dashboards in *Jeeves* can be used to both visualize data in general by showing a grid view and to focus on a specific detail of a dataset by focusing on a specific cell. With our tool we provide a simple approach to the interaction with data: once the user performs a vocal query to the vocal assistant, *Jeeves* answers by automatically providing an optimal visualization based on the shape of data.

We extracted some important requirements from the literature and from our previous experience with BoardBile: for the user interface part we decided to provide 1) accessibility, 2) simplicity and 3) maintained ownership of the data. For the part of Visual User Interface, we followed the guidelines given by the Amazon Alexa service: 1) adaptability, 2) personability, 3) availability and 4) relatability.

We built a web–based application easily accessible that allows users to create visualizations and insights about the data by using vocal commands. Having such a customizable environment enables people to understand better the data, ask more questions and quickly find answers in the data itself.

People who use *Jeeves* can share their dashboards with others during presentations, lessons or demonstrations with the goal of empowering collaboration and decision–making.

In the end we discuss the challenges we faced during the implementation, how we solved them and we present the technical details of *Jeeves* like: interaction between our system and the Alexa Developer Console, the models we created, the inferrer provided to choose the best type of visualization and the interaction between the server side and our web application.

5.2 Validation

Due to time constraints, performing a controlled experiment was beyond the scope of this work. Still, we validated our application by anecdotal evidence deploying *Jeeves* on a big

screen, next to an Alexa Device. During these occasions, we performed some demo with users and received useful feedback about possible functionalities to provide to *Jeeves* and fixes to apply like the implementation of progressive responses and the introduction of the Levenstein Dinstance.

5.3 Contribution

With *Jeeves* we provide an interaction–centric approach to data visualization using voice control. We deploy our web application on a big screen, next to an Amazon Alexa device. Users can upload dataset to the application and create rooms for interacting with their data. By talking to the vocal assistant, they can create a customized dashboard by selecting the dimension of the grid.

Moreover, they can request to visualize an overview of the dataset or open a single cell in the dashboard and create custom visualization only by using vocal commands. *Jeeves*'s system, contributes by analyzing the dataset and providing answers to the vocal queries by providing the optimal visualization depending on the shape of data.

For example, if the user asks to visualize two numerical values in one chart, then our system will understand that the best visualization to show such information is a scatter plot and will create a visualization that is immediately shown in the web browser.

5.4 Future Work

We conclude this document discussing the possible future improvement for *Jeeves*. The component–based structure of this tool makes it prone to extensibility and introduction of new features. As possible future improvements, we want to investigate the following directions:

- Add some standard approaches to filter the data so that would be possible to insight about the data in a finer lever of detail;
- Join datasets and perform queries on them in the context of the same room;
- Customize graphs, axes and labels using vocal commands;
- Deploy all the components of *Jeeves* in a unique computer (e.g., a Raspberry Pi) so to make the usability of our application even more accessible and easy to use;
- Perform an extensive validation during a controlled experiment [35] (like the one proposed by Richard Wettel et al. in the paper "Software systems as cities: a controlled experiment" [34]) so that we can formulate a more formal set of feedbacks and more ideas about how efficient our approach is and how to extend it;
- Add different types of cell not only related to data visualization. Like Jupyter or Zeppelin, we plan to introduce cells used for creating descriptions and comments about the dataset analysis.

We believe that the concepts shown above may contribute to a big improvement in the usability and potentiality of *Jeeves*.

Appendix A

The Datasets

To validate our approach we used some dataset to test:

- The ingestion of the csv in our system with the subsequent type inferrer system
- The visualization of distribution of each column un the dataset
- The analysis and filtering–applying to multiple columns

The datasets we used are called “Diamonds” and “New York Citi Bike Trip Histories”

A.1 Diamonds

A.1.1 Dataset Description

The dataset “Diamonds” gives information about almost 54,000 diamonds by their cut, color, clarity, price, and other attributes. It is a widely used dataset and according to it is considered a great dataset for beginners learning to work with data analysis and visualization[1].

A.1.2 Dataset Structure

The dataset is composed by 10 columns, they are:

1. **carat** (numerical): weight of the diamond (0.2–5.01)
2. **clarity** (categorical): a measurement of how clear the diamond is (I1 (worst), SI2, SI1, VS2, VS1, VVS2, VVS1, IF (best))
3. **color** (categorical): diamond colour, from J (worst) to D (best)
4. **cut** (categorical): quality of the cut (Fair, Good, Very Good, Premium, Ideal)
5. **price** (numerical): price in US dollars ($326–$18,823)
6. **depth** (numerical): total depth percentage = z / mean(x, y) = 2 * z / (x + y) (43–79)
7. **table** (numerical): width of top of diamond relative to widest point (43–95)
8. **x** (numerical): length in mm (0–10.74)
9. **y** (numerical): width in mm (0–58.9)
10. **z** (numerical): depth in mm (0–31.8)

[1] Diamonds Dataset: https://www.kaggle.com/shivam2503/diamonds

A.1.3 Dataset Head

Here we provide the first 5 rows of the Diamonds dataset following the structure showed in Section A.1.2:

carat	clarity	color	cut	price	depth	table	x	y	z
0.23	SI2	E	Ideal	326	61.5	55	3.95	3.98	2.43
0.21	SI1	E	Premium	326	59.8	61	3.89	3.84	2.31
0.23	VS1	E	Good	327	56.9	65	4.05	4.07	2.31
0.29	VS2	I	Premium	334	62.4	58	4.2	4.23	2.63
0.31	SI2	J	Good	335	63.3	58	4.34	4.35	2.75
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

TABLE A.1: Head of the "Diamonds" dataset

A.2 New York Citi Bike Trip Histories

A.2.1 Dataset Description

This dataset is provided by a company called citibike[2]. The city of New York, participating to an Open Data initiative since 2013, has decided to provide for free information regarding where do customers of the Citi Bike service ride, when they ride, how far do they go, which stations are most popular, what days of the week are most rides taken on, etc...

The dataset is composed by almost 33,000 trips and they span a period from April 2019 to May 2019.

A.2.2 Dataset Structure

The dataset is composed by 15 columns, they are:

1. **Trip Duration**: in seconds
2. **Start Time and Date**: in the format yyyy-MM-dd HH:mm:ss.SSSS
3. **Stop Time and Date**: in the format yyyy-MM-dd HH:mm:ss.SSSS
4. **Start Station Name**
5. **End Station Name**
6. **Station ID**
7. **Start Station Latitude**
8. **Start Station Longitude**
9. **End Station Latitude**
10. **End Station Longitude**
11. **Bike ID**

[2]CitiBike Dataset: https://www.citibikenyc.com/system-data

12. **User Type**: divided into two types
    - *Customer*: who has a 24-hour pass or a 3-day pass.
    - *Subscriber*: who is an annual member
13. **Gender**: divided into three types:
    - *Zero* means unknown gender
    - *1* means male
    - *2* means female
14. **Year of Birth**

A.2.3 Dataset Head

Here we provide the first 5 rows of the Diamonds dataset following the structure showed in Section A.2.2

bikeid	birth year	end station id	end station latitude	end station longitude	end station name	gender
29536	1966	3184	40.7141454	-74.0335519	Paulus Hook	1
26191	1990	3187	40.7211236	-74.03805095	Warren St	1
29302	1987	3202	40.7272235	-74.0337589	Newport PATH	2
26220	1989	3214	40.7127742	-74.0364857	Essex Light Rail	1
26228	1995	3273	40.721650724879986	-74.04288411140442	Manila & 1st	1
⋮	⋮	⋮	⋮	⋮	⋮	⋮

start station id	start station latitude	start station longitude	start station name	starttime	stoptime	tripduration	usertype
3183	40.7162469	-74.0334588	Exchange Place	2019-04-01 07:48:04.7540	2019-04-01 07:49:52.3590	107	Subscriber
3183	40.7162469	-74.0334588	Exchange Place	2019-04-01 12:41:09.6540	2019-04-01 12:49:23.0290	493	Customer
3183	40.7162469	-74.0334588	Exchange Place	2019-04-01 14:27:07.9400	2019-04-01 14:35:40.9370	512	Subscriber
3183	40.7162469	-74.0334588	Exchange Place	2019-04-01 15:07:38.1510	2019-04-01 15:10:54.6260	196	Subscriber
3183	40.7162469	-74.0334588	Exchange Place	2019-04-01 16:19:24.6940	2019-04-01 16:27:08.6450	463	Subscriber
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

TABLE A.2: Head of the "New York Citi Bike Trip Histories" dataset

Appendix B

Chart Models Structure

In this Appendix, we report the structure of the chart models as they are computed by our server and returned to the graphical user interface of *Jeeves*.

B.1 Barchart

This object contains information about the x and y axes of a bar chart. It also provides a title and a description that get automatically generate by *Jeeves*.

```
{
   "title":"(t) cut",
   "description":"<b>Distinct Values: </b>5",
   "response":{
      "axes":{
         "x":[
            "Ideal",
            "Premium",
            "Very Good",
            "Good",
            "Fair"
         ],
         "y":[
            NumberLong(21551),
            NumberLong(13791),
            NumberLong(12082),
            NumberLong(4906),
            NumberLong(1610)
         ]
      },
   },
   "request":{
      "x": "columnID"
   }
}
```

LISTING B.1: Structure of the Barchart object

B.2 Histogram

Very similar to the structure of the Barchart, the Histogram provides also information about the distribution of the column and the top values sorted by occurrency in descending order.

```
{
```

```
2     "title":"(#) carat",
3     "description":"<b>Distinct Values: </b>273.0<br/><b>Min: </b>0.2 - <b>Median: </b
          >0.7 - <b>Max: </b>5.01",
4     "response":{
5        "axes":{
6           "x":[
7              "0.3",
8              "0.31",
9              "1.01",
10             ...
11          ],
12          "y":{
13             "top values":[
14                NumberLong(2604),
15                NumberLong(2249),
16                NumberLong(2242),
17                ...
18             ],
19             "active":0,
20             "distribution":[
21                {
22                   "occurrences":319.0,
23                   "min":0.2,
24                   "max":0.24
25                },
26                {
27                   "occurrences":719.0,
28                   "min":0.24,
29                   "max":0.27
30                },
31                ...
32             ]
33          }
34       },
35    },
36    "request":{
37       "x":"columnID",
38    },
39 }
```

LISTING B.2: Structure of the Histogram object

B.3 Boxplot

The boxplot object contains information about the grouping categories (in the x axis) and the distribution of each category (in the y axes).

```
1  {
2     "title":"cut by price",
3     "response":{
4        "axes":{
5           "x":[
6              "Fair",
7              "Ideal",
8              "Premium",
9              "Good",
10             "Very Good"
```

```
            ],
            "y":[
               [ 2757.0, 2759.0, 2753.0 ],
               [ 2757.0, 2759.0, 2753.0 ],
               [ 2757.0, 2759.0, 2753.0 ],
               [ 2757.0, 2759.0, 2753.0 ],
               [ 2757.0, 2759.0, 2753.0 ]
            ]
         },
      },
      "request":{
         "x":"columnID",
         "y":["columnID"],
      }
}
```

LISTING B.3: Structure of the Boxplot object

B.4 Scatterplot

The scattrplot object contains information about the two numerical distribution that the user want to visualize. They are respectively reported inside the x and the y axes.

```
{
   "title":"carat vs price",
   "description":"<b>Correlation: </b> 0.92"
   "response":{
      "axes":{
         "x":[
            "0.23",
            "0.25",
            "0.26",
            ...
         ],
         "y":[
            [
               475.94,
               535.89,
               565.32,
               ...
            ]
         ]
      },
   },
   "request":{
      "x":"columnID",
      "y":[
         "columnID"
      ],
   },
}
```

LISTING B.4: Structure of the Scatterplot object

Appendix C

APIs

In this Appendix, we report the endpoints that *Jeeves* exposes to the front end.

C.1 Endpoints with Swagger

We decided to introduce the Swagger UI service to provide a navigable view of the endpoints exposed by *Jeeves*.

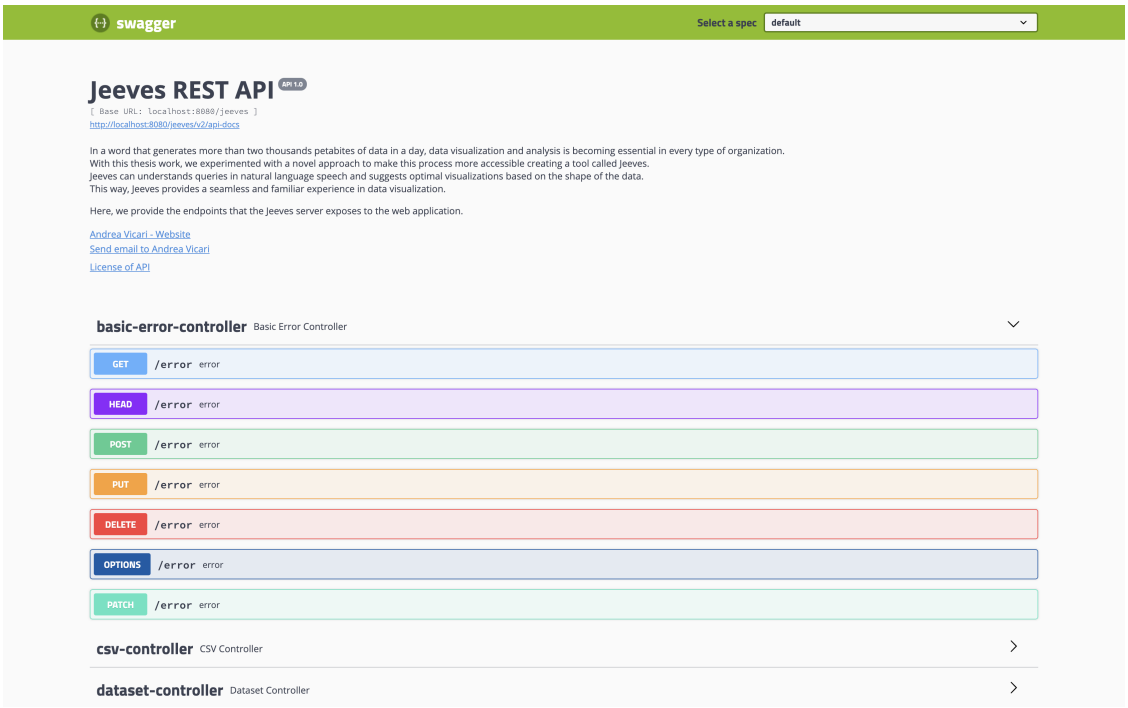


FIGURE C.1: Swagger UI showing the endpoints exposed by *Jeeves*

They are easily visible by accessing the route `http://localhost:8080/jeeves/swagger-ui.html#/` (if *Jeeves* runs on localhost:8080) as shown in Figure C.1.

C.2 The Controllers

Here we list and explain the purpose and how we used the various endpoints that we created.

C.2.1 CSVController

This controller exposes three endpoints:

	Method	Endpoint	Purpose
1	POST	*/csv/*	Upload the dataset in the server
2	GET	*/csv/summarize/{roomId}*	Returns the first 5 rows of the dataset linked to the room with the given roomId
3	DELETE	*/csv/{csvName}*	Deletes the dataset with the given csvname

TABLE C.1: Endpoints provided by the CSV Controller

C.2.2 DatasetController

This controller exposes five endpoints:

	Method	Endpoint	Purpose
1	GET	*/dataset/*	List the available datasets
2	GET	*/dataset/room/{roomId}*	List the available datasets in the given room
3	PUT	*/dataset/room/{roomId}/{fileName}*	Add the given dataset to the given room
4	PATCH	*/dataset/room/{roomId}/{fileName}*	Remove the given dataset to the given room
5	PATCH	*/dataset/{id}*	Modify information regarding the header of the given dataset

TABLE C.2: Endpoints provided by the Dataset Controller

C.2.3 RoomController

This controller exposes thirteen endpoints:

	Method	Endpoint	Purpose
1	GET	*/rooms/*	List the available rooms
2	GET	*/rooms/{roomId}*	Get information about the given room
3	GET	*/rooms/{roomId}/cells/{cellNumber}/renderChart*	Get the chart data in the given room and cell number in base64
4	GET	*/rooms/create*	Create a new room
5	GET	*/rooms/screenConnect/{roomId}*	Connect to a room
6	GET	*/rooms/sse-subscribe*	Subscribe to the Server Sent Event that sends updates about the connected devices to the rooms
7	GET	*/rooms/sse-subscribe/{roomId}*	Subscribe to the Server Sent Event that sends updates about the connected devices to the given room
8	GET	*/rooms/{roomId}/zoomInCell/{cellNumber}*	Select the cell that has been zoomed in
9	GET	*/rooms/{roomId}/zoomOutCell/{cellNumber}*	Select the cell,that has been zoomed out
10	POST	*/rooms/{roomId}/cell/{cellNumber}*	Setup a cell externally
11	POST	*/rooms/{roomId}/cell/{cellId}/render*	Make a request so that a chart response can be returned and rendered in the dashboard at the given room and cell number
12	GET	*/rooms/{roomId}/cell/{cellId}/histCellmode/{mode}*	Used in the histogram cells to switch from "Top Values" mode to "Distribution" mode and vice versa
13	GET	*/rooms/sessionExchange/{roomId}*	Used to establish a web socket connection between the server and the client. Once the client is connected and subscribed to the topic of one room, it will automatically receives updated whenever they come.

TABLE C.3: Endpoints provided by the Room Controller

Appendix D

Code Maintenance

In this Appendix, we report information and metrics about the implementation of tests, coverage and bug fixes.

To keep track of these information, we use SonarQube. This tool provides the capability to show the code quality of our application and highlight introduced issues. It helped us to fix the possible vulnerabilities in the code and improve our code quality.

D.1 Tests

In *Jeeves*'s code base, we provided test cases, we use them to be sure that the systems works as expected also after the introduction of new features.

In the current of version of our system, we provide 105 unit tests, implemented using Junit (Figure D.1).

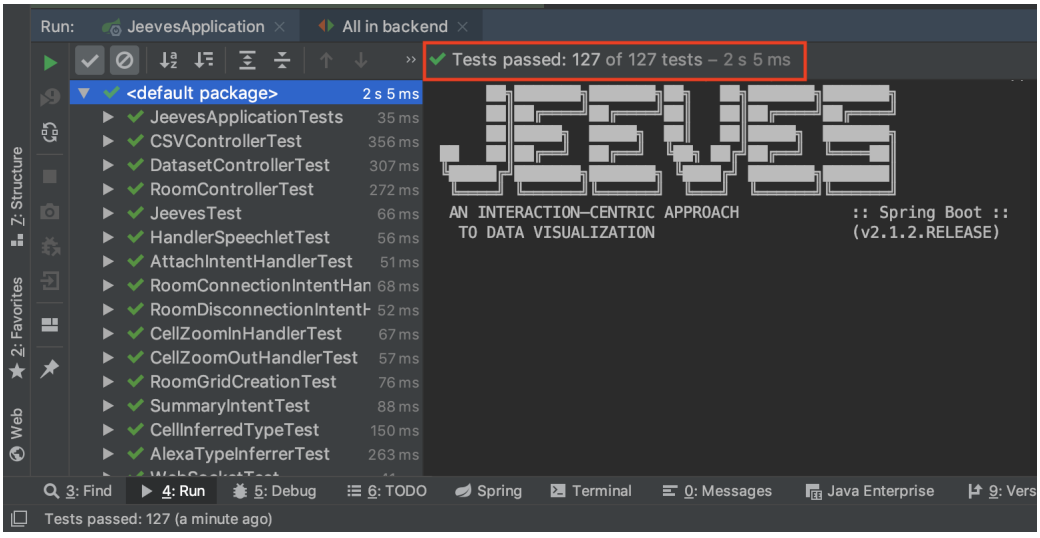


FIGURE D.1: Unit tests implemented in *Jeeves*

D.2 SonarQube

D.2.1 Bugs

SonarQube helped us during the development of *Jeeves* by detecting wrong code, or code that was not having the intended behavior.

At the moment we started using SonarQube we had 33 bugs reported. In the current of version of our system, SonarQube finds 0 bugs (Figure D.2).

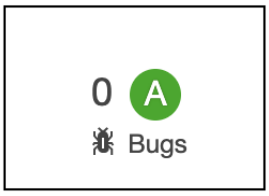


FIGURE D.2: Bugs found by SonarQube in *Jeeves*

D.2.2 Vulnerabilities

By tracking vulnerabilities, SonarQube helped us find and track the insecurities in our code.

At the moment we started using SonarQube we had 6 vulnerabilities reported. In the current of version of our system, SonarQube finds 0 vulnerabilities (Figure D.3).

FIGURE D.3: Vulnerabilities found by SonarQube in *Jeeves*

D.2.3 Code Smells

Code can be defined as "smelly" if it does what it should, but it will be difficult to maintain. Code smells represent factors that indicate bad practices or indicate that the design is starting to get rusty [24].

At the moment we started using SonarQube we had 248 code smells reported. In the current of version of our system, SonarQube finds 9 code smells (Figure D.4).

FIGURE D.4: Code Smells found by SonarQube in *Jeeves*

The only code smells still present in our project are some *TODO* notes and some false positives. As visible in Figure D.5, they are labeled as to the Minor and Info Severity.

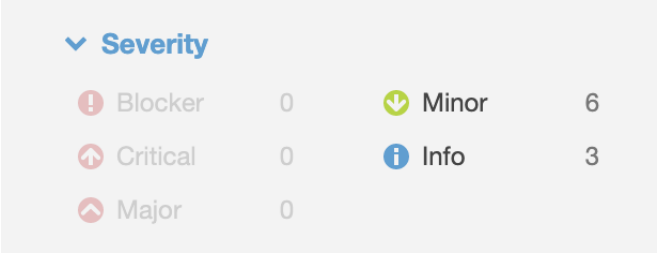


FIGURE D.5: Code Smells found by SonarQube in *Jeeves*are either Info or Minor

D.2.4 Coverage

SonarQube's code analyzers explore all possible execution paths to spot possible bugs. The coverage helped us understand if we were covering all the paths of our tool during the execution of test cases.

In the current of version of our system, SonarQube detects a code coverage of 66.4% (Figure D.6).

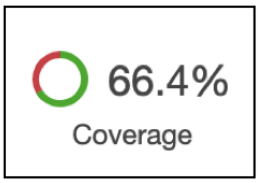


FIGURE D.6: Coverage found by SonarQube in *Jeeves*

Some parts of the tool are uncovered because we found some difficulties in mocking the conversation between our server and the Amazon Service: we are using a testing suite provided with the Amazon Alexa SKD for Java and we aim to bring the coverage at least at the level of 80%.

D.3 Duplications

SonarQube also check the code for duplications of code. Code duplication can, in fact, lead to inconsistency in the code and difficulties in the maintenance of the tool.

At the moment we started using SonarQube we had 0.4% of code duplication reported. In the current of version of our system, SonarQube detects a percentage of duplication equal to the 0.0% (Figure D.7).

FIGURE D.7: Duplications found by SonarQube in *Jeeves*

Bibliography

[1] Amazon alexa - be adaptable. https://developer.amazon.com/it/docs/alexa-design/adaptable.html. Visited: 2019-06-08.

[2] Amazon alexa - be available. https://developer.amazon.com/it/docs/alexa-design/available.html. Visited: 2019-06-08.

[3] Amazon alexa - be personal. https://developer.amazon.com/it/docs/alexa-design/personal.html. Visited: 2019-06-08.

[4] Amazon alexa - be relatable. https://developer.amazon.com/it/docs/alexa-design/relatable.html. Visited: 2019-06-08.

[5] Bar chart/bar graph: Examples, excel steps & stacked graphs. https://www.statisticshowto.datasciencecentral.com/probability-and-statistics/descriptive-statistics/bar-chart-bar-graph-examples/. Visited: 2019-05-31.

[6] Host a custom skill as a web service. https://developer.amazon.com/it/docs/custom-skills/host-a-custom-skill-as-a-web-service.html. Visited: 2019-06-01.

[7] Overcoming challenges in voice-based natural language processing (nlp) in business - challenges in voice (as opposed to text)-based nlp. https://emerj.com/ai-podcast-interviews/spoken-voice-based-nlp-for-business/. Visited: 2019-06-08.

[8] Quick guide to spring controllers. https://www.baeldung.com/spring-controllers. Visited: 2019-06-01.

[9] Spring - mvc framework. https://www.tutorialspoint.com/spring/spring_web_mvc_framework.htm. Visited: 2019-06-01.

[10] Spring vs spring boot. https://www.educba.com/spring-vs-spring-boot/. Visited: 2019-05-29.

[11] Statistics dictionary - categorical and quantitative variables. https://stattrek.com/statistics/dictionary.aspx?definition=categorical%20variable. Visited: 2019-05-31.

[12] Statistics dictionary: Scatterplot. https://stattrek.com/statistics/dictionary.aspx?definition=scatterplot. Visited: 2019-05-31.

[13] J. Benesty, J. Chen, Y. Huang, and I. Cohen. *Pearson Correlation Coefficient*, pages 1–4. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.

[14] Chen, Chun-houh, Hrdle, Wolfgang, Unwin, Antony, Chen, Chun-houh, Hrdle, Wolfgang, Unwin, and Antony. *Handbook of Data Visualization (Springer Handbooks of Computational Statistics)*. Springer-Verlag TELOS, Santa Clara, CA, USA, 1 edition, 2008.

[15] S. Dean and P. Barbara Illowsky. Descriptive statistics: Histogram.

[16] G. Der and B. Everitt. *A Handbook of Statistical Graphics Using SAS ODS*. A Chapman & Hall Book. Taylor & Francis, 2014.

[17] S. Few. *Show Me the Numbers: Designing Tables and Graphs to Enlighten*. Analytics Press, USA, 2nd edition, 2012.

[18] M. Gregg. *CISSP Practice Questions Exam Cram (4th Edition)*. Pearson IT Certification, USA, 4th edition, 2016.

[19] Keim, D. Andrienko, G. Fekete, J.-D. G
"org, C. Kohlhammer, J. Melançon, and Guy. *Visual Analytics: Definition, Process, and Challenges*, chapter 1, pages 154–175. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[20] D. A. Keim, M. C. Hao, U. Dayal, H. Janetzko, and P. Bak. Generalized scatter plots. *Information Visualization*, 9(4):301–311, 2010.

[21] Kirk and Andy. *Data Visualisation: A Handbook for Data Driven Design*. Sage Publications Ltd., 2016.

[22] P. Kruchten. Architecture blueprints - the "4+1" view model of software architecture. In *TRI-Ada Tutorials*, pages 540–555, 1995.

[23] M. Krzywinski and N. Altman. Visualizing samples with box plots. *Nature Methods*, 11:119 EP –, 01 2014.

[24] M. Lanza and R. Marinescu. *Object-Oriented Metrics in Practice: Using Software Metrics to Characterize, Evaluate, and Improve the Design of Object-Oriented Systems*. Springer Publishing Company, Incorporated, 1st edition, 2010.

[25] W. M., G. G., and K. D. *Interactive Data Visualization*. A K Peters/CRC Press, New York, 2015.

[26] D. Moore. *The Basic Practice of Statistics*. Freeman, 2010.

[27] J. Nielsen. *Designing Web Usability: The Practice of Simplicity*. New Riders Publishing, Thousand Oaks, CA, USA, 1999.

[28] K. Pearson. Contributions to the Mathematical Theory of Evolution. II. Skew Variation in Homogeneous Material. *Philosophical Transactions of the Royal Society of London Series A*, 186:343–414, 1895.

[29] Ribarsky, William, Foley, and James. Next-generation data visualization tools. *Croatian Journal of Fisheries*, 1994.

[30] S. Rimbert. Jacques bertin, sémiologie graphique : les diagrammes, les réseaux, les cartes. *Annales de géographie*, 84(462):241–242, 1975.

[31] J. Stasko, J. Domingue, M. Brown, and B. P. (Eds.). *Software Visualization - Programming as a Multimedia Experience*. The MIT Press, 1998.

[32] E. Tufte. *Envisioning Information*. Graphics Press, Cheshire, CT, USA, 1990.

[33] G. Upton and I. Cook. *Understanding Statistics*. OUP Oxford, 1997.

[34] R. Wettel, M. Lanza, and R. Robbes. Software systems as cities: a controlled experiment. In *2011 33rd International Conference on Software Engineering (ICSE)*, pages 551–560, May 2011.

[35] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén. *Experimentation in Software Engineering: An Introduction*. Kluwer Academic Publishers, Norwell, MA, USA, 2000.

[36] N. Yau. *Data Points: Visualization That Means Something*. Wiley Publishing, 1st edition, 2013.