Navio: a visualization widget for summarizing, exploring and navigating large multivariate datasets

John A Guerra-Gomez*
Universidad de los Andes
UC Berkeley

Juan C. Ortiz-Roman†
Universidad de los Andes

Juan G. Murillo-Castillo‡
Universidad de los Andes

Tiberio Hernandez§
Universidad de los Andes

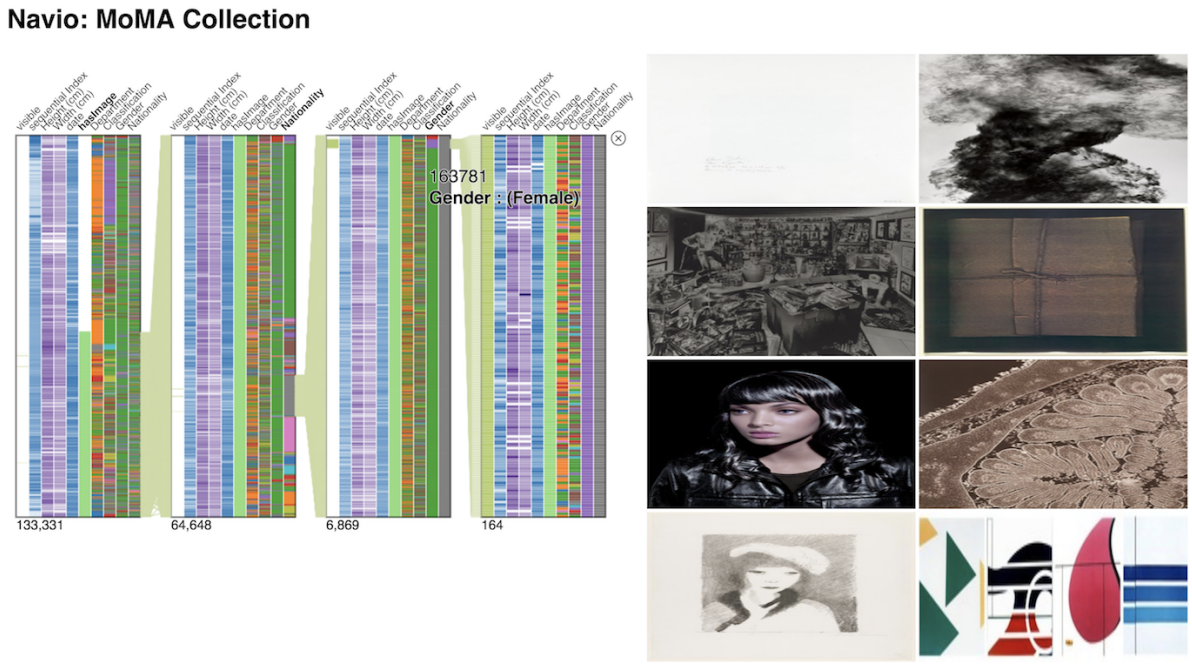

Figure 1: MoMA collection explored with an application supporting Navio, which allow users to summarize, navigate and explore the data. From the original 133,331 dataset, the user has used the widget to filter for only the pieces including images (64,648 pieces), then sorted by Nationality and selected France (6,869 pieces) and finally sorted by Gender, identified the underrepresentation of French female artists in the dataset and selected them (only 164 pieces). Available at `https://john-guerra.github.io/momaExplorer/`

Abstract

Where do you start when analyzing a dataset with 1M+ entries and 20+ attributes? One could describe it with statistics, load it Tableau or Voyager, however each of these approaches will show you facets of the data instead of summarizing it as a whole. Moreover, what if you want your users to be able to navigate it? You could use faceted search, but users wont' be able to preserve the context of their explorations. To address this we present NavIO, an open source, web based and reusable interactive visualization widget for summarizing, filtering and navigating large datasets. Initially presented as a static widget for summarizing networks, we greatly improved navio by scaling it up to large datasets, adding direct manipulation filtering, and adapting it also to multivariate datasets. We released Navio as an open source widget that can be added to your visualization with less than 10 lines of code, and even provide an on-line application for loading your data and generating skeletons for using on your applications. Navio has been evaluated with an usability study with 8 participants, an experiment where a data scientist used it to explore their own complex data, and by a domain expert on political data. Our results show that even though Navio requires training and getting used to, it is an useful and powerful tool for summarizing and navigating large datasets.

Index Terms: Human-centered computing—Visualization—Visualization systems and tools—; Human-centered computing—Visualization—Visualization application domains—Visual analytics

1 Introduction

When facing large datasets, the visualization analysts usually follows one of two approaches: they aggregate the data and display a **summary** of it, or they display **subsets** of the data providing a filtering mechanisms such as faceted search. Either of these two approaches have their own advantages and disadvantages, but maybe the biggest drawback is the lack of context. This happens because users can only see small samples of the data at a time, and then apply filters on different dimensions that they cannot keep track of. Let's illustrate this by means of an example. In recent years, the Museum of Modem Art of New York released a dataset containing more than 130 thousand pieces of their collection [13]. With

*e-mail: ja.guerrag@uniandes.edu.co
†e-mail: jc.ortiz939@uniandes.edu.co
‡e-mail: jg.murillo10@uniandes.edu.co
§e-mail:jhernand@uniandes.edu.co

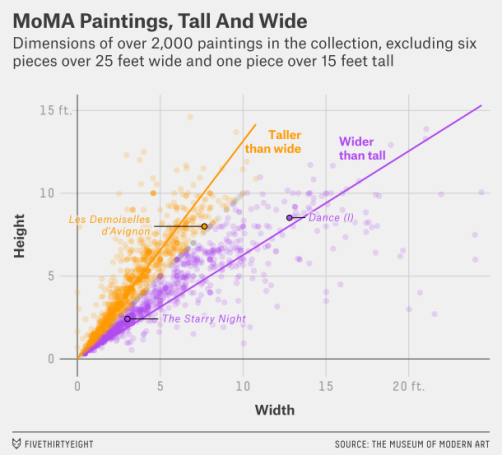

Figure 2: FiveThirtyEight summary of the MoMA dataset comparing 2,229 Paintings width and heights.

such rich dataset, the visualization and other communities faced the challenge of exploring it and creating many interactive pieces. The MoMa itself created a compendium of the projects and people that analyzed the data or created an application to explore it [14]. Let's focus on two of these: FiveThirtyEight's data nerd guide for the MoMA Collection [14] on Figure 2, and MoMA's own searching interface [12] on Figure 3'.

FiveThirtyEight, with their experienced data analysts created beautiful and meaningful **summaries** that aggregated the data by some of their dimensions and presented interesting results. Figure 2 shown one of the analysis they conducted, when focusing on only 2,229 Paintings (out of the more than 130 thousand total pieces) and comparing only their width vs height. They presented an interesting result, however it is one that focuses only in two of the attributes of the dataset and on a subset of the data. This is an excellent example on how summaries lose context, as the user can't get an idea of the whole dataset.

As a second example, the MoMA itself created an application to help explore the dataset using a search interface, pagination and some faceted search [12]. The whole interface is very beautiful and intuitive, as it shows the standard UI for presenting search results of multimedia objects. However, it only shows about 30 elements at a time, which hides the remaining 129 thousand images. Moreover, users have to know what query to search for to explore the collection. The faceted search provides a way of navigating on a more structural way, but probably to try to keep the interface simple, it only allows to search by category or date period, which ignores the many more attributes of the dataset (e.g. dimensions of the piece, author, country, date acquired, etc). This type of faceted search is very easy to understand and useful when one knows what to look for, but it doesn't allow for serendipitous exploration. Moreover, despite users being able to filter images by different criteria very easily, they have no way of keeping track of the context of their queries, neither can they perform nested queries.

On this paper we present Navio, a visualization widget for helping users summarize large and complex datasets, explore them and perform advanced nested queries while keeping the context. Navio was conceived initially as an static summarization visualization widget to support a network exploration tool [5]. Given the potential identified on our initial explorations of it as a summarization widget, we decided to expanded in three ways:

- Scalability, Navio was completely redesigned to support larger datasets, by selecting the most important nodes to draw, and optimizing memory allocation

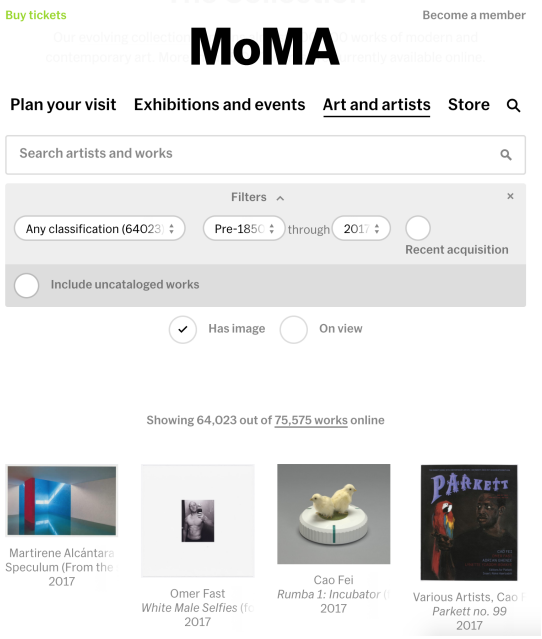

Figure 3: MoMa search interface using faceted search. It's a friendly and intuitive way of searching on the MoMA collection if you know what to look for. However, for serendipitous exploration or just getting a sense of the size of the data is not so helpful.

- Generalization, the original widget was designed to visualize only networks, Navio was extended to support large multivariate data with less than 100 columns and of up to 400MB (more of less the restriction of current browsers)

- Filtering, we added a direct manipulation filtering that allow users to query the data on specific attributes on an intuitive way. Users can query either by selecting a range of attributes (dragging) or by clicking on a specific value they want to select

- Navigation, where the original conception users could filter the network and jump into subsets of the data and see them reflected on the widget, on Navio users can navigate directly on the widget exploring subsets with a clear context of their trail

On this paper we contribute:

- Navio the visualization widget for summarizing, navigating and exploring complex large datasets. The widget was released as an open source library that can be easily added to your application. Available at `https://github.com/john-guerra/Navio`

- Navio Shipyard, a ready to use application where users can upload their own data, summarize, navigate and explore it using Navio, and generate skeleton code for it, following the tasks definitions by Munzner [11]. Available at `https://john-guerra.github.io/Navio/shipyard/`

- The results of our experimentation with the widget that reveal that even though it requires training, it can be a powerful tool for exploring large datasets, understanding your own data or as side navigation widget for your app.

The rest of this paper is organized as follows: first we describe the related work that inspired our work. Then we describe Navio, its

algorithms and features. We present then three case studies where apply it, to describe later the three experiments we use to evaluate it, and finally present some thoughts on applicability, limitations and our conclusions.

2 Related Work

There are many tasks an user can perform for analysing and visualizing multivariate datasets. On her Visualization Design and Analysis book, Tamara Munzner [11] presents an useful framework, that among other defines a taxonomy of tasks that a user can perform on a visualization. *Summarization*, *Navigation* and *Exploration* are the three that Navio addresses. Summarization refers to getting a general idea of the dataset as a whole, while Navigation and Exploration involve being able to run specific queries. The main difference is that with Explorations users don't know before hand what they are looking for. Despite being such common tasks, currently there is still space for improvement on user available tools for performing them on a new dataset. On this section we describe some of the tools and research projects that try to address these three tasks.

Tableau is one of the most commonly used tools for first exploring a dataset for analysts. Originally published as the Polaris project [21], its Vizql language and easy to use interface represents and appealing option for exploration. However, despite being a commercial and powerful tool, when an user first opens a dataset on tableau it is faced with a blank canvas, where she has to take action and decide what dimensions or values to visualize. After this, the analysts can create many different types of visualizations and dashboards and test very quickly many different combinations. This is great for the *navigation* (i.e. when the user knows what to look for on the first place), but it doesn't lend easily to *summarization* or *exploration*. As a contrast, an application using the Navio will immediately display a summary of the full dataset, displaying missing values, patterns and distributions. Moreover, the user is able to perform complex queries, all while keeping context. However, according to our experiments results, Navio is significantly more difficult to understand than the traditional visualizations that Tableau uses.

A different approach to Tableau, would be using a tool like Voyager [22]. As shown on Figure 5, after loading a dataset Voyager would use its Compass engine to generate many summary visualizations for each one of the data attributes, and from there the user can start exploring the data by fixing certain attributes and exploring other combinations. This is great for exploration, as the common visualizations used by Voyager are easy to understand, but comes at a cost. First, loading a dataset like the MoMa Collection used on this paper, takes more than two minutes on Voyager and will show many empty charts after loading(Figure 5). Second, the independent summaries will hide any patterns on the same items and across attributes as described later on this paper. Compared to Voyager, Navio presents a summary of all the information on the dataset. Loads many times faster (which is understandable given that Voyager is trying to recommend visualizations). It is also less sensitive to outliers, missing values or poorly formated data.

A simpler approach for visualizing multivariate data would be to use techniques such as a simple search interface or a faceted search interface [18]. For its dataset [13], the MoMa itself created a simple application [12] that allows users to *navigate* it very easily if they know what to look for. It even includes some optional filters that help users getting an idea of how the data is distributed by categories, and performed more elaborated filters. Compared to an application using Navio (e.g. `https://john-guerra.github.io/momaExplorer/`), the MoMa's search interface is significantly easier to understand and to use at a first glance. However, it doesn't provide a good summary of the whole dataset, doesn't include many of the attributes in the data, doesn't a allow for complex nested queries and neither helps the user understand their query history, therefore it fails on more advanced summarization and exploration.

There are other specialized tools in the market, that allow for summarization and complex queries similar to the ones that Navio provides. InfoZoom [6], initially published as FOCUS [20] is a powerful Business Intelligence software that displays an useful summary of a multivariate dataset, using meaningful organization and useful labels. Moreover it allows for exploration and navigation using rich interactions. Navio, shares and builds upon their practices using a similar interface, but adding a history of the queries performed among other things. Moreover, Navio has been designed as a widget that can be included in other applications and that uses modern web technologies, where InfoZoom is a desktop application. This is both a benefit and a disadvantage as InfoZoom probably can scale up better using all the computer resources without the security caps established on browsers.

Apart from these, we provide a comparison of other research projects that addressed in some ways the tasks handled by Navio. Examples of these include Pixel Bar Chart [9], Value Bars [2], Parallels Coordinates [7] and Generalized Spirals [8]. Table 1 presents a comparison of the different features of each one of these projects compared to Navio.

We developed Navio using D3.js [1], one of the most widely used libraries for web visualization, which offers a good starting for building network visualizations. Navio also uses concepts from Shneiderman's Direct manipulation [15–17] and Dyanmic queries [18]. Moreover, Navio uses some of the concepts of Dunne's Graph Trail [4], by allowing users to track the trail that they have followed through the data. However, the Navio is a significantly different widget; it is specifically designed to help users navigate networks, and provides information about the attributes of multiple nodes in a single place.

Shneiderman's visualization mantra of "Overview first, zoom and filter, then details-on-demand" [19] dictates that a visualization systems should provide a good overview of the data, support zoom and filtering options, and offer details when the user requests them. Following the infoviz mantra, Navio features an overview mode that allows users to obtain an idea of the dataset as a whole. It also provides controls to filter on demand, and offers detailed information at users' request.

3 Navio

The Navio is a side visualization widget that helps users summarize, explore and navigate large datasets. Users can summarize the dataset by means of the main visualization that represents each data row as a line on a vertical bar. They can also change the sorting order of the data to identify distributions by each attribute. For navigation, users can sort then filter by ranges or specific attribute values. Finally, they can explore better datasets by being able to summarize and filter at the same time on all the attributes, identifying interesting patterns they want to explore further. On this section we present first the main functionalities of Navio, then we explain its new algorithm for supporting larger datasets and describe how it can be used.

3.1 Navio Widget Functionalities

Navio uses three main ways of interactions. First users can hover over the widget to obtain detailed information of the item represented on the corresponding line. Second users can sort items by their attributes by clicking on the corresponding column title. This proved to be useful to understand attributes distributions and to identify missing values. Finally, users can click and drag anywhere on the bar to filter the elements by an specific value or a range. Using these simple interactions Navio supports three main types of analysis tasks:

3.1.1 Summarize and sorting

When a user first opens an application that uses the Navio widget, she can see a summary of the whole dataset and all its attributes

Characteristic \ Visualization	Pixel Bar Chart	Value Bars	Par. Coords	Gen. Spirals	Navio
Provides a useful overview of the whole dataset	X	X		X	X
Many attributes' distribution overviews		X	X		X
Ability to sort to identify attributes of interest	X		X		X
Handle Multidimensional data			X	X	X
Avoids overlaps	X	X		X	X
Enables exploration		X	X		X
Maintains context of filtered items			X		X
Nested filtering			X		X
Limit of dimensions		X			

Table 1: State of the art comparison

visualized as a vertical bar with columns for each attribute and distributing the vertical space between rows. She can use this summary to get an initial idea of how the data looks like in its natural order. One thing users commonly do when first examining this view is to identify missing values as they get represented with empty space. Figure 4 (a) shows an example of this with the MoMa dataset, where the analyst discovered immediately that there are many registers with a Height of 0.

To better understand the distribution of values for an specific attribute, users can also click on the header of the column representing the dimension to sort the data by this measure. Shown on Figure 4 this proved to be useful for understanding distributions, identifying missing values and easing selections.

Users can also hover the mouse over the items to get details on demand with information about the current item, its id and it's value for the currently selected attribute.

From our explorations, users had a harder time determining distributions on Navio compared to presenting results with faceted bar charts such as Voyager does [22]. However, that approach uses more space, and doesn't keep the item context. For example, with Navio users can easily see that art pieces with a Height of 0 commonly also have a Width of 0 on the MoMa example of Figure 4, this wouldn't have been possible to identify on Voyager, unless the user performed a query first or knew to look for it on the first place. This is evident on Figure 5 that shows the main summary Voyager displays when first loading the MoMa dataset, which took close to two minutes on a MacBook Pro 2017 with 16GB of RAM.

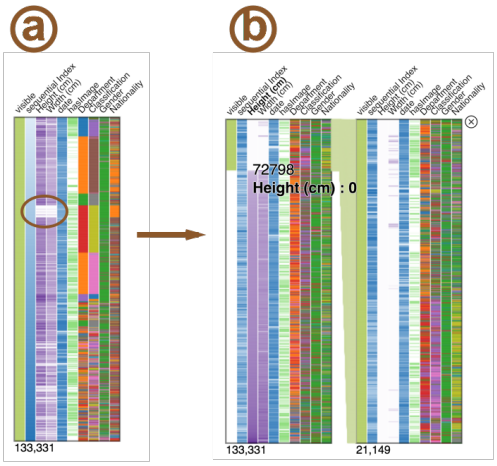

Figure 4: Identifying and counting missing values on the MoMa collection dataset. The image on the left shows a whole on the **Height** column which represents many art pieces without a proper height value. The analysts then decides to select them, by first sorting by sort and then selecting all the 21,149 pieces without a value. Available at `https://john-guerra.github.io/momaExplorer/`

3.1.2 Navigation

The second main analysis task that a user can perform on an application that uses the Navio widget is to navigate the dataset by means of advanced nested queries. For this they start from the original summary and then decide to either to re-sort the data by a different attribute or just to perform an specific query. Afterwards they will obtain a subset of the data represented with a new Navio bar where they can repeat the process recursively. As an example Figure 1 shows how the user decided to query for art pieces by French women that included images. For this she first sorted and filtered by pieces with images, getting 64,648 results. From there the analyst sorted the subset by Nationality (which shows by hovering that there is a significant number of French pieces), and selected France, which resulted on a third level with 6,869 pieces. Finally the analyst decides to re-sort again by Gender, finding that French women are highly underrepresented, and decides to filter to their pieces finding only 164 in the whole dataset. Navio also shows other genders that have weird values in the dataset, that can be easily found by hovering. As shown by the Figure, Navio helps users keep track of their query trail and by means of level connections (on green) users can easily understand what part of the data they are looking at. Moreover, the first attribute to the left, also on green highlights on each level of the query how many items have been selected. All levels remain interactive, and users can come back at any time to each level and re-sort or perform new queries, every time the user selects a new subset, the Navio widget triggers a customizable callback that for the MoMa application displays the first 50 images on the subset.

From our experiments, compared with other tools an application that uses Navio provides better support and understanding for better nested queries, while keeping the user informed of the context of what they are exploring at all times. On the flip side, Navio requires more training and is not as intuitive to use as other tools that use faceted search [18] for instance.

3.1.3 Exploration

The final analysis task for Navio is exploration. This is similar task to navigation with the main difference of the user not knowing before hand what they are looking for, or having more open ended questions. Applications using Navio support exploration by allowing the user to first get fast summaries of the whole dataset or subsets of it, and second supporting fast and complex queries. With Navio users can identify outliers, tendencies, distributions, missing values from the summary visualization and then filter the data and continue the exploration on a subset of it. The Colombian Senate dataset is a good example of exploration, as with the collaboration of the domain experts of Congreso Visible, we analyzed all the voting patterns of Colombian senators since 2006. Figure 6 shows a network visualization application that uses Navio for exploring the dataset. The application allows analysts to explore the whole history of votes, selecting different periods of time, parties, types of votes, congress chamber, and topics among others. Once the user selects

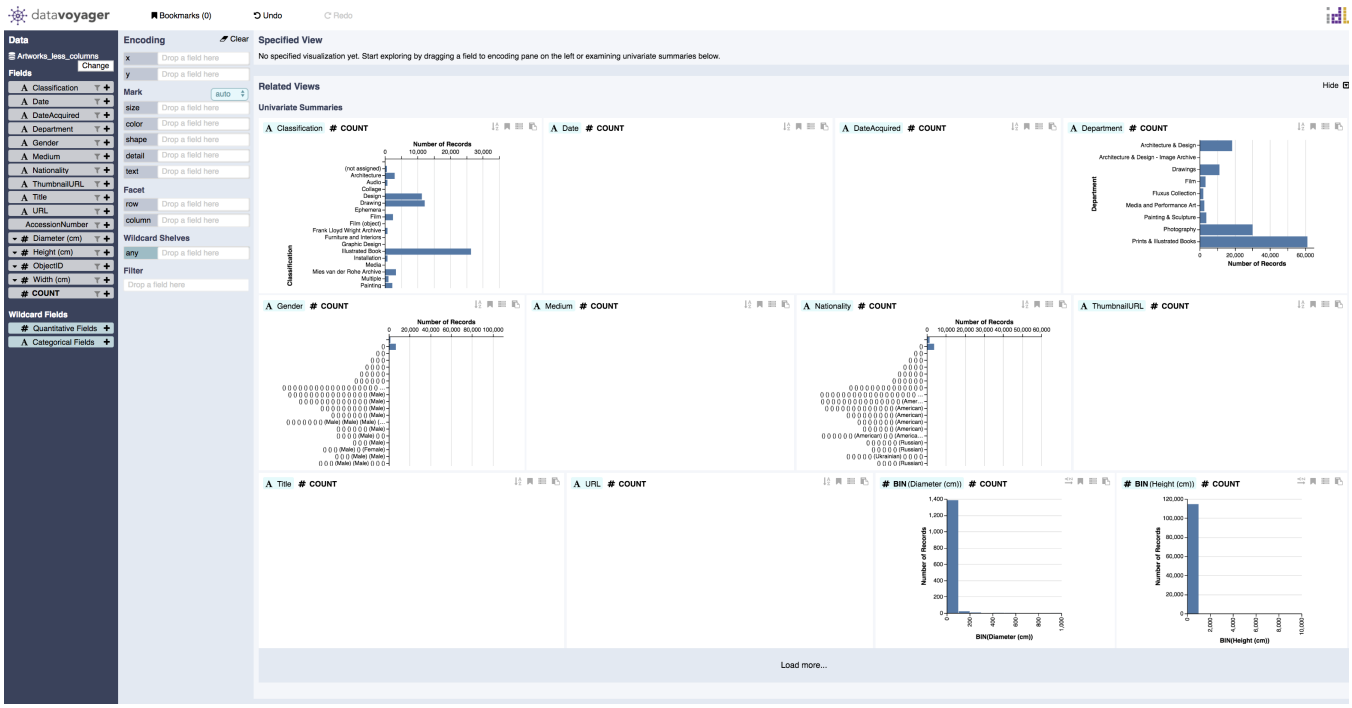

Figure 5: MoMA collection initial summary after loading it with Voyager. Since the tool performs much more on the background on try to identify the best views to show users and attributes it takes significantly more time to load. It also fails to display charts for attributes with weird formating. Available at `https://john-guerra.github.io/momaExplorer/`

an specific subset, the application displays co-voting network, where each senator (represented by a dot, colored by its party) is connected to another one if they have voted on the same way for more than a certain number of occasions. With Navio the application allows the exploration of interesting subnetworks for specific patterns that illustrate for instance how the Colombian Senate splits on the topic of Economy.

3.2 Navio Shipyard

Navio Shipyard is a web application built with ReactJS that allows the user to upload, preprocess, explore, summarize and visualize a dataset using Navio without writing a single line of code. In order to use the application, the user must upload a dataset; the dataset file has to be semicolon-separated values, comma-separated values or tab-separated values. After that, the Navio widget, its configuration panel and an useful reactive table with a sample of the data will be displayed as shown on figure 7. In the configuration panel, the user is able to hide or show attributes and change their type (categorical or ordinal). The Navio Shipyard is not just an configurable interface for the Navio because the user can export the filtered data into a comma-separated values file and the embedded visualization. Due to the architecture of the Navio Shipyard, that is just front-end and is handling all the data in the client side, this application has data size limitations and in short term we expect to develop a scalable back-end that implements elastic search to overcome this limitation. Navio Shipyard is also an open source project and is available at `https://john-guerra.github.io/Navio/shipyard/`.

4 Implementation Details

Navio has been implemented to scale up to large datasets directly on the browser. As of today, it works well with datasets to the size of what you can load on a modern browser, being this its main restriction (i.e. about 400MB depending of the computer and browser). This presents a great improvement over the original implementation created for the Network Explorer [5], that didn't support filtering and will start being to slow with datasets on the hundreds of thousands of records. Navio's current implementation has been tested with millions of records with little impact on performance. However, to address even bigger datasets we are working on future improvements that will leverage a fast and scalable backend to handle some of the processing while keeping the same architecture on the front end. On this section we describe the main implementation details of Navio.

For enabling Navio to handle millions of records with dozens of attributes, it uses the D3 v4 [1] library combining SVG and Canvas rendering for different components. A single Navio level bar consists on a series of horizontally stacked bars that displays the data attributes as columns, while showing one line per each data item vertically inside each attribute bar. Since any small dataset could have thousands of items, drawing each line using SVG wouldn't be efficient (as it will flood the DOM with an excessive number of tags). Because of this, Navio uses canvas for drawing each line, while still utilizing SVG for creating transparent attribute placeholders on top that allow for supporting its interactions: (i) sorting attributes by clicking on its titles, (ii) selecting an attribute value by clicking on it, and (iii) filtering a range by using a brush. Every time the user performs a filter a new Navio level bar is created to represent the new selected subset of the data. Connecting lines are drawn to do the matching between items on the parent and children level bars, and since each bar can have different sorting criteria, each line needs to drawn individually. To achieve this Navio uses again canvas for drawing the lines and the indexing and sampling techniques that are described next in this section. The remaining of the section describes the types of attributes supported and the technique used for reporting filtering to the main application

4.1 Indexing and Sorting

When a Navio widget is first displayed, it shows the data in it's natural order (i.e. the sequential order in which it comes). However,

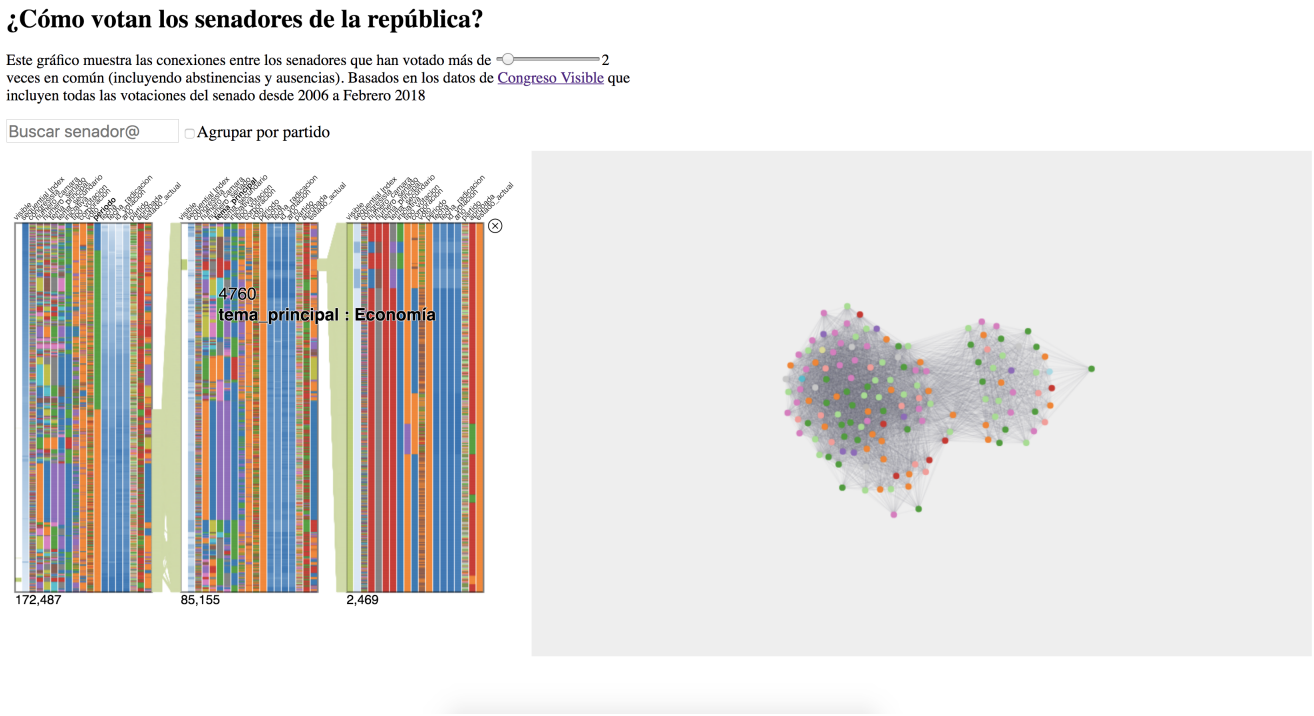

Figure 6: Exploratory network visualization application using Navio for helping users explore 172k votes collected from the Colombian Senate since 2006. With this application users can explore and navigate the voting patterns, selecting very specific subsets such as from the votings of the current period 2010-2014 (85,155 in total), only the ones about Economy (2,499). Once the user selects a subset, a network on co-voting appears on the right. Each dot represent a senator colored by party, and they are connected if they have more than 2 votes in common. Available at `http://johnguerra.co/viz/senadoColombia`

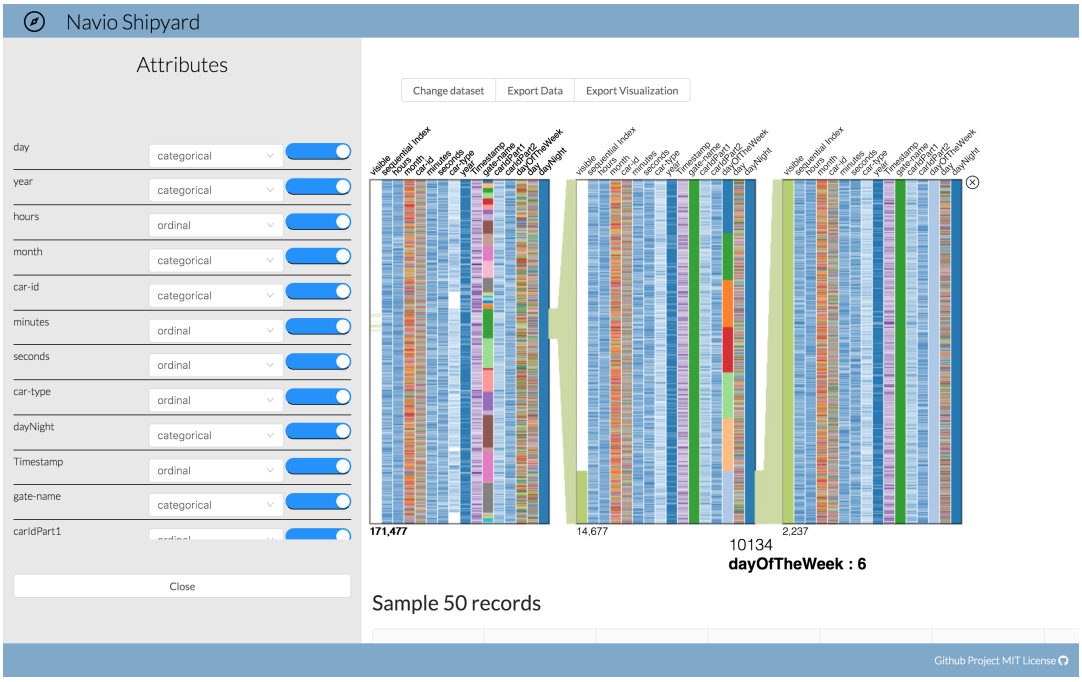

Figure 7: Navio Shipyard, an stand alone application for summarizing, exploring and navigating complex multivariate data, and a generator of applications using Navio. On the figure, shipyard has been loaded with the 2017 VAST Mini Challenge 1 featuring “the movement of traffic through the Boonsong Lekagul Nature Preserve” `http://vacommunity.org/VAST+Challenge+2017+MC1`. On the Figure the user has selected all the traffic through the general-gate-1 and then selected to see only the traffic during Sundays

users can re-sort the data by different attributes by clicking on a different attribute title. To allow for going back to the original ordering, Navio appends a new seqId attribute to the data and displays it as a new attribute that the user can click on to re-sort. To achieve this sorting, Navio internally keeps an index of the current sorting with the position of each item, for each one of the level bars. Every time a level bar is sorted the index is updated to reflect its new value. This index is crucial for supporting the filtering interactions. Each time the user clicks or drags on the items Navio first detects the attribute where the user clicked using the *x* position and the SVG placeholders described above. Then it uses the *y* position to find the matching item using the index and a reverse D3 scale that goes from the domain of the pixels to the range of the index. The implementation of this reverse scale isn't trivial and requires two components: First the original scale used to distribute the attributes across the *y* axis is a d3.scaleBand() (because depending on the size of the data each item can occupy one or more pixels). Second, given that d3.scaleBand() doesn't have a reverse functionality, Navio implements it by creating a d3.scaleQuantize() that goes from pixels to the index. This way Navio can match a range of pixels to the same item if need be. These scales are kept for each one of the level bars and are updated every time the user re sorts.

4.2 Sampling

The original implementation of Navio for the Network Explorer [5] used Canvas for drawing each item on the dataset, but, it will try to draw all of the items, despite each one not getting even a full pixel and therefore resulting on item overlapping. From the user perspective this was fine, as it they will still see a sample of the items that will allow them to identify distributions and grasp a summary of the data. Our current implementation is much more efficient. For this, Navio computes a sample of the data to guarantee that only one item is drawn per pixel, which greatly improved the overall widget performance and allowed it to scale to the millions of items. This sample is computed by calculating a *step* that computes how many items per pixel would have been drawn depending on the dataset size and the parameters of height of the widget

$$step = max(floor(level_barnumber_of_items/(height*2)),1) \quad (1)$$

Then the dataset is sampled every *step* to create an array of *representative* elements which are finally the ones being drawn. This architecture can be extended to support even greater datasets by leveraging an scalable and efficient back-end architecture that handles the sampling for us, which is one of our current areas of work on the project.

The samples also have to be taken into account when drawing the connection bars between levels, as one item can be a *representative* on one level but not on the previous one. To handle this, we draw the connecting lines from either the item *y* position itself (i.e. if it is a *representative*) or from the closest *representative* that we can find using the following formula.

$$\begin{aligned} indexNextRepr = floor(indexForItem- \\ indexForItem\%itemsPerPixelForLevel) \end{aligned} \quad (2)$$

Even though with this sampling architecture the user doesn't get to see all the items at once, we believe this is fine as the user still gets a representative sample of the data. Having said that, we recognize that this is a limitation of Navio, but it is still an improvement over binning for histograms or only showing one attribute at a time, which are the available options from the state of the art.

4.3 Types of attributes supported

For using Navio, the developer only needs to perform four simple steps on its code: include the library, instantiate the widget, define the attributes to use with their types, and pass the data. Navio supports three types of attributes, sequential, categorical or custom made. For each attribute a different color scale is used. White is usually reserved for empty values, and for sequential values Navio uses a linear d3.sequentialScale on a range of blues from d3-scale-chromatic. For the case of categorical values, Navio defaults to a d3.scaleOdinal with an d3.schemeCategory10. This represents one of the main limitations of Navio, as many times datasets contain more than 10 different values, but from our experiments we have found that this is a bearable problem. Custom attributes are the third option reserved for developers that want to use a predefined color scale of their choice. This is extremely useful when the developer is already using a color scale to represent an attribute of the data on a different visualization. Figure 6 shows an example of this, where color has been used for representing the Senator' party on the network visualization, and therefore Navio was configured to use the same color schema for the corresponding party attribute. Other example of usage of custom attributes are when the developer wants to use a different non linear scale for an attribute, or when they want to differentiate the type of attribute, e.g. between dates and numerical attributes. This last approach was the one used for the VAST 2017 mini challenge 1 example displayed on Figure 7, where dates are displayed with shades of purple.

4.4 Callback

The final implementation characteristic of Navio is the support for filtering callbacks. An optional step when using Navio in your code is to define a callback function that is going to be triggered every time the user performs a filtering action on the widget. This callback is passed to the onUpdate method of a Navio instance, and should be a function that receives as a parameter an array with the items available on the last level bar, which corresponds to the currently selected items on the widget. Apart from this callback, the developer can also call a method getVisible which will also return the list of currently selected items. This method is especially useful when building dashboards that also filter the data on other ways different to the ones provided by Navio. Another use case for this method is when the application performs an operation on the resulting items of the filtering, e.g. the Colombian Senate of Figure 6 that generates a co-voting network out of the filtered votes.

5 Evaluation

Navio has been extensively tested by means of case studies and experimentation. On this section we present three of the activities we conducted to get feedback on the widget, to assess its usability and potential utility. First we conducted an small study with the MoMa Collection dataset to assess the widget usability as well as contrasting it with traditional interfaces for the general user. Second, and to address also a more technical user base, we ran a short exploratory experiment where we asked a data scientist to use Navio Shipyard to analyze his own complex data. Finally, we gathered a domain expert feedback, by means of a network exploration application, that help him understand his own data about the Colombian Senate voting patterns using Navio. On this section we present the results of the experiments and what we learned from them.

5.1 Usability study: MoMa collection

An usability study was conducted to evaluate Navio using the Museum of Modern Art of New York dataset. The dataset is composed by 133,331 works of art along with the medium they were made from, the year it was made, the height and width of the work, the gender and nationality of the artist, the category they belong to according to the museums classification, the department at the museum

where they are found and whether or not the work had an image associated. 8 participants were asked three initial warming questions to familiarize them with the widget, then they were explained the interactions they could make with it, and finally a series of three questions were made for them to answer. Time was taken as soon as the question was given to the participants, when they believed they had found the answer they told the answer out loud. If it was correct time was stopped, if not it continued. Participants did not have limit of errors, they could retry as many times as they want or they could reject the question, although just one of the participants opted to do it in one question. For the final three questions time was taken and the number of mistaken answers was also registered.

The questions were structured to navigate the dataset and develop exploration of the data attributes. These three questions were asked to each participant: 1. How many oil on canvas artworks were made by women? 2. In which medium were classified the majority of artworks from the previous century? 3. How many photographs made by female artists in 1980 have an image associated in the dataset?

The same questions were also asked to be solved using MoMAs online search interface. Even though participants took 27% more time to gave an answer, they made 60% less mistakes when using Navio in comparison with the use of MoMAs search interface. A likeart scale was used to measure the easiness, usefulness and overall satisfaction of using Navio to solve each tasks, according to the participants. For the first task 3 of the 8 participants agreed that the widget was Extremely easy to use, only one said it was Extremely hard to solve question 3 with Navio. Usefulness was assigned a majority of participants giving an score of 6 for all three questions, with 7 being Extremely useful.

5.2 Exploratory study

In the exploration study, a data scientist used Navio Shipyard to explore its own complex data, a dataset with 500,000 registers and 99 attributes of taxes payment by economic sector. Although the data scientist did know about the context of the problem he had not explored the dataset. The study allowed the data scientist to retrieve two important insights in a 25 minute lapse, gaining understanding of the structure of the dataset and identifying two potential groups of interest for his study along with its origin from the data. These groups were found by applying a series of chained filters of the attributes the scientist considered relevant (economic sector, document type, number of taxpayers and amount of tax payment).

In comparison with the tools he used daily, Navio allowed the user to compare categories in an easier way and apply filters directly which help him find groups of interest and data particularities.

5.3 Domain expert validation

Finally to test Navio with non technical domain experts, we developed a collaboration with the Colombian research group CongresoVisible [3]. Congreso Visible has been collecting data about the Colombian Congress since 2006. They have a rich dataset of each one of the votes casted by the Colombian Senators with rich attributes including the Senator, their party, the date, details about the vote, the actual vote and much more. For them we built a simple network exploration application (available at `http://johnguerra.co/viz/senadoColombia/`). The application shows the full dataset of 177k+ votes, and using Navio allows the user to select subsets. Each time a subset is selected the application recomputes the co-voting patterns of the senate and displays them as a force directed network visualization. We presented this visualization to one domain expert from Congreso Visible and ask him about his impressions. The expert expressed that this tool allowed him to explore the data in ways that wasn't possible for him before, despite him been knowledgeable about Network Analysis and simple data analysis tools too. On the flip side he expressed similar concerns expressed by other domains experts we showed Navio too, saying that the tool could be overwhelming at a first glance and that will require some previous training. Having said that, he expressed that the flexibility it allowed justified the extra complexity and need for training.

6 Limitations and Future work

As shown, Navio is a useful widget that can be added to an application to help users summarize, navigate and explore large datasets. However, from our experimentations with it we have been able to identify some of its limitations. The main one might be that is not as intuitive as other commonly known interfaces (e.g. search box, histograms or faceted search) to the untrained user. As explained on the user evaluation section, many users expressed that their initial reaction towards the widget was that of an overwhelming experience. However, many of them also complemented their comments by saying that once they learned how to use it, they felt it was a powerful tool for advanced querying and summarizing.

Another main limitation of Navio is the memory restrictions of modern browsers (around 400MB). This can be somehow restrictive, especially for datasets with several dozens of attributes. Moreover, when deploying an application using Navio to the web, users will experience loading times getting the data to their browsers. This is a restriction that most if not all frontend based data processing applications have, but we are working on new implementations that leverage Navio's sampling architecture to make use of an efficient back-end to lower the data transfer burden and avoid the browser limitations. On the same page, Navio scales better on the number of items than on the number of attributes, as with many of them, the widget starts using too much space for showing nested queries.

Other feedback that we have obtained on the widget, is that when setup for use smaller attribute columns, the widget can be difficult to read. Similarly, the ten colors palette used as a defaults gets repetitive too soon. For this developers can define their own palettes very easily, but perceptual wise it shouldn't help much.

Finally there are many points of improvement that we have collected, as providing better feedback when processing or during ranged filtering, as well as other topics such as progressive loading, multiple level sorting, or disjunct filtering. All of them are on our roadmap for future versions of the widget.

7 Conclusion

We presented the Navio an open source visualization widget designed to help users perform summarization, navigation and exploration tasks (as defined by Munzner [11]). Navio is a significant improvement over the original static widget presented on the original Network Explorer [5, 10] paper, it has been redesign to support significantly larger datasets, as well as to enable users to easily perform complex nested queries using direct manipulation. Moreover, we presented an stand alone, open source, and freely available application called Navio Shipyard that can be used by any user to summarize, navigate and explore their own data.

We have evaluated Navio by means of an small usability study, an exploratory experiment where a data scientist used it to explore his own data and with domain experts exploring the Colombian Senate. Moreover, we have applied it to several other datasets that we don't presented here because of space or confidentially restrictions. With all of these, we believe that Navio is a useful widget for the community, which can be easily added to existing applications or that can be used as an stand alone application with Navio Shipyard for exploring complex multivariate datasets. Navio is available for download and use at `https://github.com/john-guerra/Navio`.

ACKNOWLEDGMENTS

The authors wish to thank Jean Daniel Fekete for his thoughtful feedback on the paper

REFERENCES

[1] M. Bostock, V. Ogievetsky, and J. Heer. D3 Data-Driven Documents. *IEEE Transactions on Visualization and Computer Graphics*, 17(12):2301–2309, dec 2011. doi: 10.1109/TVCG.2011.185

[2] R. Chimera. Value Bars: an information visualization and navigation tool for multi-attribute listings and tables. 000:0–1, 1998.

[3] Congreso Visible. Congreso Visible Guatemala contra la corrupción.

[4] C. Dunne, N. Henry Riche, B. Lee, R. Metoyer, and G. Robertson. GraphTrail: analyzing large multivariate, heterogeneous networks while supporting exploration history. In *Proceedings of the 2012 ACM annual conference on Human Factors in Computing Systems - CHI '12*, p. 1663, 2012. doi: 10.1145/2207676.2208293

[5] J. A. Guerra-Gomez, A. Wilson, J. Liu, D. Davies, P. Jarvis, and E. Bier. Network Explorer: Design, Implementation, and Real World Deployment of a Large Network Visualization Tool. In *Proceedings of the International Working Conference on Advanced Visual Interfaces - AVI '16*, vol. 07-10-June, pp. 108–111. ACM, ACM Press, New York, New York, USA, 2016. doi: 10.1145/2909132.2909281

[6] HumanIT. Versatile Data Analysis: More Than Just Common BI Software — Infozoom.

[7] A. Inselberg and B. Dimsdale. Parallel Coordinates: A Tool for Visualizing Multi-dimensional Geometry. In *Proceedings of the 1st Conference on Visualization '90*, VIS '90, pp. 361–378. IEEE Computer Society Press, Los Alamitos, CA, USA, 1990.

[8] D. A. Keim. Designing pixel-oriented visualization techniques: theory and applications. *IEEE Transactions on Visualization and Computer Graphics*, 6(1):59–78, 2000. doi: 10.1109/2945.841121

[9] D. A. Keim, M. C. Hao, U. Dayal, and M. Hsu. Pixel bar charts: a visualization technique for very large multi-attribute data sets. *Information Visualization*, 1(1):20–34, mar 2002. doi: 10.1057/palgrave/ivs/9500003

[10] J. Liu, E. Bier, A. Wilson, J. A. Guerra-Gomez, T. Honda, K. Sricharan, L. Gilpin, and D. Davies. Graph Analysis for Detecting Fraud, Waste, and Abuse in Healthcare Data. *AI Magazine*, 37(2):33, jul 2016. doi: 10.1609/aimag.v37i2.2630

[11] T. Munzner and E. G. artist) Maguire. *Visualization analysis & design*. 2014.

[12] Museum of Modern Art. The MoMA Collection, 2015.

[13] O. D. Robot. MoMA Collection - Automatic Monthly Update [Data set]. mar 2018. doi: 10.5281/zenodo.1186463

[14] F. Romeo. Here's a roundup of how people have used MoMA's data so far. Medium, 2015.

[15] B. Shneiderman. Direct manipulation: A step beyond programming languages (abstract only). *ACM SIGSOC Bulletin*, 13(2-3):143, 1981. doi: 10.1145/1015579.810991

[16] B. Shneiderman. The future of interactive systems and the emergence of direct manipulation. *Behaviour and Information Technology*, 1(3):237–256, 1982. doi: 10.1080/01449298208914450

[17] B. Shneiderman. Direct Manipulation: A Step Beyond Programming Languages. *Computer*, 16(8):57–69, 1983. doi: 10.1109/MC.1983.1654471

[18] B. Shneiderman. Dynamic queries for visual information seeking. *IEEE Software*, 11(6):70–77, nov 1994. doi: 10.1109/52.329404

[19] B. Shneiderman. The Eyes Have It: A Task by Data Type Taxonomy for Information Visualizations. vol. 0, p. 336. IEEE Computer Society, Los Alamitos, CA, USA, 1996. doi: 10.1109/VL.1996.545307

[20] M. Spenke, C. Beilken, and T. Berlage. FOCUS. *Proceedings of the 9th annual ACM symposium on User interface software and technology - UIST '96*, pp. 41–50, 1996. doi: 10.1145/237091.237097

[21] C. Stolte, D. Tang, and P. Hanrahan. Polaris: a system for query, analysis, and visualization of multidimensional relational databases. *IEEE Transactions on Visualization and Computer Graphics*, 8(1):52–65, 2002. doi: 10.1109/2945.981851

[22] K. Wongsuphasawat, D. Moritz, A. Anand, J. Mackinlay, B. Howe, and J. Heer. Voyager: Exploratory Analysis via Faceted Browsing of Visualization Recommendations. *IEEE Transactions on Visualization and Computer Graphics*, 22(1):649–658, 2016. doi: 10.1109/TVCG.2015.2467191