DataScope: Interactive Visual Exploratory Dashboards for Large Multidimensional Data

Ganesh Iyer, Sapoonjyoti DuttaDuwarah, Ashish Sharma

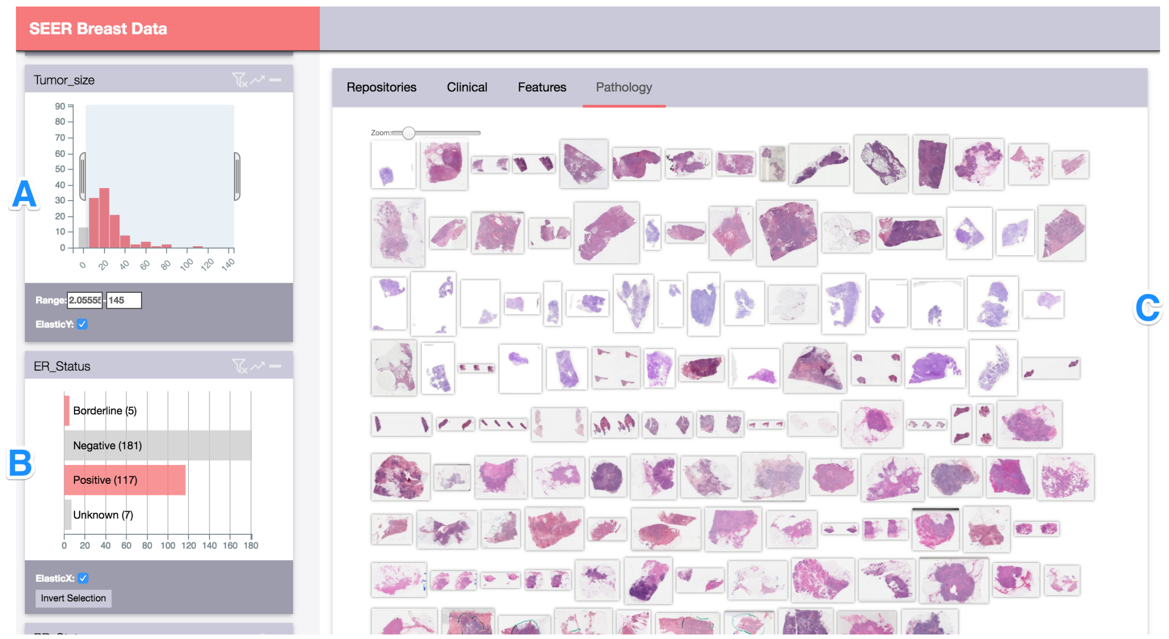


Fig. 1. User interacting with a DataScope dashboard generated using breast cancer registry data from Surveillance, Epidemiology, and End Results Program (SEER). In this screenshot, the user is interested in looking at pathology images of patients with tumor sizes larger than 20 mm in diameter and with estrogen-receptor(ER-Status) positive or borderline. The user sets the corresponding filters in A and B. The image grid C shows the pathology images for this filtered cohort.

Abstract—DataScope is a web-based tool for generating interactive visual dashboards on large scale multidimensional datasets. Users can use these dashboards to explore data and create cohorts for downstream analysis. We describe DataScope's architecture, design considerations and provide an overview of its system design. We highlight some of DataScope's features that were useful in the case studies using datasets from cancer registries and co-clinical trials. In benchmarks DataScope is able to perform sub-second queries on data sizes ranging from thousand to million records.

Index Terms—Information visualization, Data exploration, Interactive data exploration

1 Introduction

DataScope is a web-based tool for visually querying and exploring large datasets. It generates dashboards that allow users to slice-dice and drill down into their data. Users can visually query and filter their datasets to create subsets or cohorts that can be saved or shared for further downstream analysis. In this paper we describe the use-cases for DataScope and provide an overview of the design and architecture.

Data curators and data providers are looking for ways to make their data more accessible to their end users and analysts. DataScope provides them with a quick way of setting up an analytics interface that enables data discovery. For example data providers of cancer repositories need their end users to be able to perform queries like: *Show pathology images of patients with tumor sizes larger than 20 mm in diameter and with estrogen-receptor(ER-Status) positive*. They would also want to see how different attributes correlate as they start performing some exploratory analysis using the dashboards.

Integrative data analysis is widely used in Biomedical sciences. Multiple data sources are pooled into one to perform these analysis [8]. DataScope can take data from a variety of heterogeneous sources and can render visual dashboards to help users understand the data better. With the advent of open data there is scope for exploratory analysis on mashups of datasets across domains [16].

DataScope provides a high-level grammar for authoring interactive dashboards on large multidimensional datasets. DataScope dashboards can be authored in a declarative manner by authoring four configuration files that describe the data sources, the attributes that are in the data, information about filtering that can be performed on the data and the appropriate visualization to render.

In this paper, we refer to the act of generating the dashboards as *authoring* , and the people responsible as *dashboard authors* . The users interacting with these authored dashboards as *end-users*.

• *Ganesh Iyer, Sapoonjyoti DuttaDuwarah and Ashish Sharma are with the Dept. of Biomedical Informatics, Emory University. Email: {ganesh.iyer, sapoonjyoti.duttaduwarah, ashish.sharma}@emory.edu*

1.1 Challenges

Creating interactive visual dashboards can be an involved process requiring a lot of engineering resources. Often a visualization designer, front-end engineer and a back-end engineer are required. Work is required to fetch and integrate datasets of different formats from different sources. A back-end must be developed to create APIs to power the dashboards. Front end engineers are required to set up the user interface and client-side code. Visualization designers are required to write custom charts and visualizations using libraries like D3 [7].

Data providers often have data coming from various heterogeneous sources namely REST APIs, Databases, flat files etc. The data can be in a variety of different formats like JSON, CSV, XML etc. Specialized libraries are required to extract and parse the data. DataScope is able to ingest data from these variety of sources and provides an interface to query and visualize the different data modalities. With DataScope data providers can specify *where* to find the data instead of *how*.

Another challenge was the scale of the data. Web based charting libraries like D3 etc. work well when by all the data is loaded in the browser. Modern web browsers can load data only upto a certain size, based on the RAM of the client and browser configurations. For larger datasets users can very easily exceed these memory limits of the browser. Preforming filtering on large datasets is computationally expensive. Also some data providers might have restrictions for sharing all of their data. Thus loading and filtering the data at the client side is not an option. We need to develop a scalable server based querying system that is able to handle massive amounts of data. We also need specialized data structures to perform sub-second querying over millions of data points.

Data providers have their own customized needs and the system must be flexible enough to allow them to make these changes swiftly. There might be custom data formats that require specialized libraries to parse them. We needed to make the system extensible enough to handle new visualization modes, filtering modes and data formats.

2 Related Work

Kienle et al [11] identified seven quality attributes for visualization tools: a) rendering scalabilty b) information scalability c) interoperability d) customizability e) interactivity f) usability g) adaptability. We base our design decisions based on these attributes. In this section we highlight some other tools that are used to generate exploratory dashboards.

There are several graphing libraries that are built using Javascript. D3 [7] is widely used as a building block for many graphing libraries like DC.js [3]. Google Charts is widely used for quickly setting up a visualization [4]. These libraries can be used to develop custom dashboards for different data providers. This requires reproducing a lot of the effort like defining interactions, linking various charts, ingesting data from different servers and so on. Thus these libraries are low-level tools that can be used as building blocks for developing dashboards.

Keshif [18] is an open source web-based environment for creating interactive visual dashboards. It has an API that can be used to configure new dashboards. Due to its client-side data loading, there is a practical limit on the data volume that can be loaded into the browsers memory. Keshif can support datasets upto 220,000 records. The size and scale of biomedical datasets normally reach the order of a million tuples with a large number of attributes. This limits the use of such client-side rendering, and therefore led us to develop a system that relies on the server-side to load the data.

Voyager [15] is a great tool for visual analysis inspired by exploratory search which is designed using AngularJS framework with HTML's flex display. The tool seems to have an edge because of the importance it gives to data variation instead of design variation. It is great at processing wide range of queries. One of the biggest advantages this tool has is the recommendation system which seems to recommend data variables and transformations which the user might find helpful. Voyager's novelty is in providing the end users recommendations while exploring datasets. However it doesn't address the issues faced by data curators for providing an environment to explore their data. It currently doesn't support a schema to define a dashboard.

Polaris [17] has a very different way to visualize data as compared to the approach that was taken by Voyager. The primary concentration of this system is on large multidimensional databases and how to effectively visualize those datasets. It challenges the way the multidimensional databases are (were) treated using the n-dimensional datacubes. The system interprets the state of the interface as a visual specification of the analysis task and automatically complies it into data and graphical transformations. The system however, lays out the requirements a visualization tool that must be matched in order for it to process large multidimensional databases which are data dense displays, multiple display types and exploratory interface. The system uses tables in order to display these high dimensional datasets. The system tries to compare it's queries against the SQL queries to see which ones give you better results. However, unlike DataScope which provides an interface to be used online with any dataset, this tool is limited to using only multi-dimensional relational databases in order to help explore the relations in the dataset.

3 Technical Overview

In this section we provide a technical overview of DataScope. We describe the purpose and the specification of DataScope's configuration files. We will also describe the architecture of DataScope.

3.1 Specification

DataScope dashboards can be declared using four configuration files, namely: a) Data source b) Data description c) Interactive filters d) Visualization. These files allow authors to specify the dashboard in a declarative manner. We use a high level grammar for generating dashboards which lets users specify *what* instead *how*.

3.1.1 Data source

The data source definition provides information about the different data sources that are pooled into one to perform integrative data exploration. This information is specified in a JSON file called `dataSource.json`. The data source definition is consumed by the application server and is used by it to load the data. It needs to provide information about the source of data, and the information required to obtain the data from the source. Multiple datasources can be provided, DataScope performs an inner-join of the different datasources. Currently DataScope supports: *Flat files*, *REST APIs* and Databases as data sources. The data must be either `JSON` or `CSV` in format. DataScope is extensible enough to add further types of data sources like XML etc. The following JSON document is an example of `dataSource.json` that is used to fetch data from a remote REST API:

```
{
  "dataSourceAlias": "breast_cancer_data",
  "dataSources": [
    {
      "sourceName": "Breast_Case",
      "sourceType": "jsonREST",
      "options": {
        "host": "http://dataprovider.com",
        "path": "/v3/api/data",
        "api_key": "testapikey"
      }
    }
  ]
}
```

3.1.2 Data description

Data description is used by the server to obtain information about each attribute in the pooled data. Each attribute must have information about the data provider, the data type of the attribute, and whether the attribute is a filtering attribute or a visualization attribute or both. This information is specified in a JSON file called `dataDescription.json` Following JSON document is an example of `dataDescription.json`.

In this example, we restrict ourselves to 3 attributes for brevity. It describes the attributes present in *Breast_Case* data provider: *Tumor_size*, *Image* and *ER_Status*.

```
[
  {
    "attributeName": "Tumor_size",
    "attributeType": [
      "filtering",
      "visual"
    ],
    "dataProvider": "Breast_Case",
    "datatype": "integer"
  },
  {
    "attributeName": "ER_Status",
    "attributeType": [
      "filtering",
      "visual"
    ],
    "dataProvider": "Breast_Case",
    "datatype": "enum"
  },
  {
    "attributeName": "Pathology_Image",
    "attributeType": [
      "visual"
    ],
    "dataProvider": "Breast_Case",
    "datatype": "string"
  }
]
```

3.1.3 Interactive Filters

A set of attributes in the pooled dataset can be used as interactive filters. These interactive filters show up on the left hand side of the dashboard and can be used to filter datasets. Each interactive filter can have its own visualization, using which, the user interacting with the system can filter the data set. Currently DataScope support four visualizations for interactively filters. Currently we support:

1. `barChart`: DataScope barCharts plot the histogram of a continuous attribute. Each bar in a histogram represents the tabulated frequency at each bin.
2. `pieChart` : PieCharts are helpful in showing proportions and percentages between categories. They are useful when the number of categories are few.
3. `rowChart` : RowCharts is normally used to show discrete, numerical comparisons across different categories. The Y-axis shows the different categories from the dataset and the X-axis shows the values.
4. `scatterPlot` : ScatterPlots use a collection of points placed on a 2D Cartesian plane to display values from two different variables. By displaying a variable in each axis, you can analyze the relationship between the two variables.

Summary statistics can also be associated with each attribute. These statistcs are shown on flipping-the-card on each interactive filter. By `defualt` DataScope keeps track of: `count, distinct, min, max, mean, median, stdev`. We use datalib library to compute the statistics [2]. In the following example we create interactive filters to display attribute *Tumor_Size* as a `barChart` and *ER_Status* as a `rowChart`. Figure 2 shows the corresponding interactive filters.

```
[
 {
    "attributeName": "Tumor_size",
    "visualization": {
      "visType": "barChart",
      "binFactor": 10,
      "domain": [0,130]
    },
    "statistics": "default"
  },
```

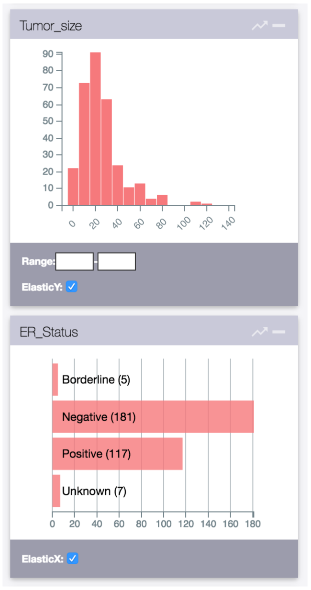


Fig. 2. Example of interactive filters generated from the `interactiveFilters.json` file provided in section 3.1.3

```
  {
    "attributeName": "ER_Status",
    "visualization": {
      "visType": "rowChart"
    }
  }
]
```

3.1.4 Visualization

The main visualization panel which shows up in the right hand side of the dashboards is defined here. The JSON file is called `visualization.json`. DataScope currently supports 10 visualizations, these include:

1. `dataTable`: Provides a tabular view of the data, can be configured to link out to external applications.
2. `imageGrid`: Displays a grid of images and can be configured to link out to external applications and display additional information about the image as a tooltip.
3. `SPLOM`: Scatter plot of matrices are useful for visualizing correlations amongst pairs of variables.
4. `geoChoroplethMap`: Choropleth display geographical areas shaded in relation to the density. Datascope geoChoropleth maps can be configured to work with any geographical area provided that the corresponding GeoJSON files are provided.
5. `markerMap`: Datascope markerMap maps can be used to render shapes over a geographical area. The size of the shape is proportional to the value of the attribute specified.

In this example `visualization.json` document we demonstrate creating an imageGrid using the attribute *Pathology_Image* from the dataset which has the corresponding link of the image. Figure 3 shows the corresponding image grid.

```
[
{
       "visualizationType": "imageGrid",
        "attributes": [
            {
```

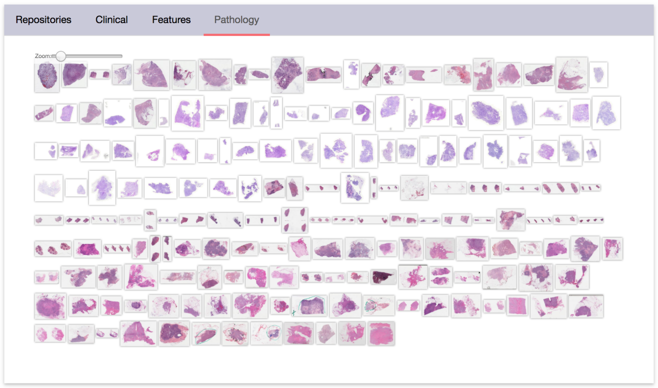


Fig. 3. Image grid generated from the `visualization.json` provided in Section 3.1.4

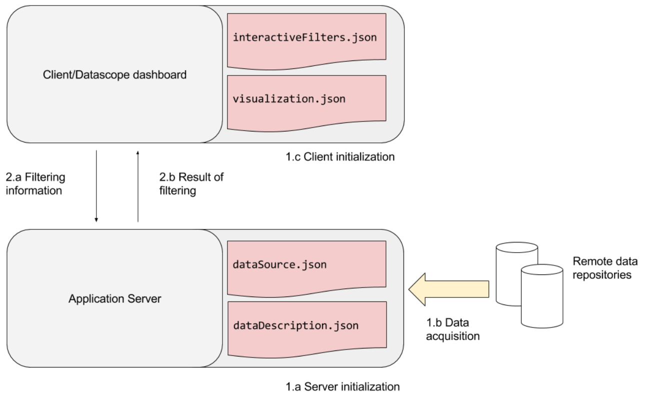


Fig. 4. Overview of DataScope's architecture.

```
                "attributeName": "Pathology_Image",
                "type": "image"
            }
        ],
        "tabTitle": "Pathology"
    }
]
```

3.2 Architecture

We used a client-server architecture to distribute the load on the system. With a fat client, applications can quickly run into browser limits, and can only work with very small datasets. To deal with larger datasets we load the data on the server and used AJAX calls between the client and the server to communicate essential information that was required by the client to render visualizations. Figure 4 provides a high level overview of the architecture.

3.2.1 Initialization

DataScope uses configuration files to initialize the server and client and to load the data into the server. As shown in Figure 4 the `dataSource.json` and `dataDescription.json` files are consumed by the server, whereas `interactiveFilters.json` and visualization.json are consumed by the client side. The configuration files are used as follows:

1. The application server identifies the various data sources. It can ingest data from REST APIs and flat files. Using `dataSource.json` the application server loads data from each data source in memory. It performs SQL-like JOIN and UNION operations based on the key attribute specified.
2. The application server creates binned aggregated representations of the data to perform filtering quickly using a javascript library called: crossfilter [1] for all the filtering attributes in the data description.
3. On the client side, `interactiveFilters.json` file is used to render the interactive filters. The client side renders the visualizations based on the visType provided in the interactive filter.
4. The client side also uses `visualization.json` to render visualizations based on the visualization type specified.

3.2.2 Filtering

Once the dashboards are initialized the users can interact with the filters and slice and dice their data. Following is how the interaction takes place:

1. *Filtering information*: The user interacts with the system and performs filtering, the filter information is sent to the server. The browser makes a GET request to /filter. The filtering information is represented in the form of a JSON object. Some examples are following:
   `{}` = No active filters
   `{age: [20,40]}` = Attribute age from range 20 to 40
   `{age: [20,40], groups: [a, c]}` = Attributes that have age from range 20 to 40 and have value for attribute groups a or c

   This filtering information can be stored to load the dashboards from previous states or to share a filtered cohort.
2. *Result of filtering*: The application server performs filtering, and sends the current state information of the attributes to the view. The results are transferred as a response to AJAX call in JSON format. The result is binned and aggregated to make the payload lightweight.

3.3 Design considerations

In this section we describe some of the design considerations that guided the development of DataScope.

3.3.1 Declarative, reusable visualization components

DataScope visualization components are designed in a generic way that makes it possible for these visualizations to handle various datatypes. Queries on these visualizations can be on of: *Ranged, Categorical or 2-D Ranged*. The goal of DataScope is to be able to specify the type of visualization that users would want to see without having to write any code for it. We use the JSON specifications provided in 3.1.3 and 3.1.4. These JSON documents can be used to specify how each attribute should be visualized without having to write any code.

3.3.2 Scalable visualizations

All DataScope visualizations must be scalable to handle large datasets. To handle larger datasets without overwhelming the user and the filtering engine we only plot summaries of the data. We use binned-aggregation to plot values for attributes [12] [13]. Using this we first bin and aggregate the date and then visualize the densities of each bin. For numeric values one can define the bin factor for binning these values. For enumerated values each value is considered its own bin. This approach can be scaled to handle large datasets without overcrowding the display.

3.3.3 Filtering

One of the major goals of DataScope is enabling the creation of cohorts. Users can filter their data using any of the interactive panels. Filtering will reduce the size of the dataset and will generate a new cohort or a subset of the original dataset. Users can then view the different visualizations to examine trends in these cohorts.

3.3.4 Coordinated interactive visualizations

In DataScope, we use the brushing and linking pattern of interaction. Making selections in one view updates data in all other views. All the other views are rendered again based on the current state of filters. Both *interactive filters* and *visualizations* are linked and coordinated. Users can make selection using *visualizations*(if the visualization supports selections) and the *interactive filters* get updated and vice-versa.

3.3.5 Saving and launching state

Each interaction in the dashboard creates a new state. The state is a JSON object which stores information about the current active filters. Users can drill down and create cohorts using the filters. They can get a sharable link for the dashboard. Any user that uses the link will launch DataScope with the given filters set. External applications can also be used to generate states and launch DataScope from pre-defined states.

3.3.6 RESTful interface

DataScope exposes a RESTful API that can be used to POST new datasets or visualizations at runtime. To add a new visualization to an existing instance of DataScope, the dashboard-authors can POST a new `visualization.json` with the new visualizations. Having a REST API also makes it easier to integrate DataScope with other external analytics applications.

3.4 Compressing data using data dictionaries

The datasets we encountered are often redundant having the same values repeated throughout the data. For example, having categorical attributes with large strings as values increases the size of the data. We can easily encode these large strings into a small, compressed coded value. During our work with SEER data we observed that there were data dictionaries available for the datasets[1]. We observed that using data dictionaries can be used to compress the data by a significant amount. Datascope stores data in a compressed form on the back-end and using a data-dictionary on the front-end to decode the data from the coded state while rendering visualizations.

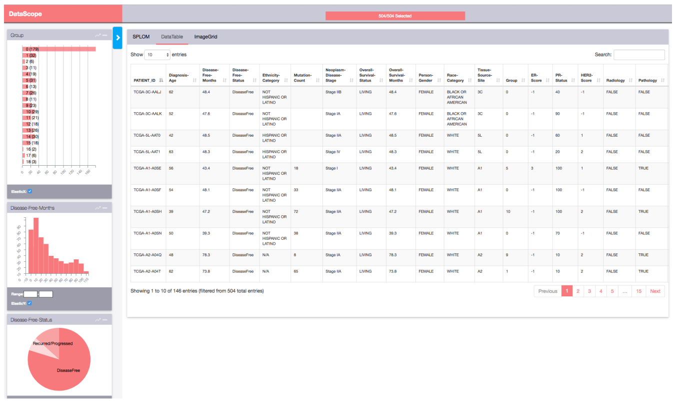
4 CASE STUDIES

In this section we describe some of the case studies where we've used DataScope. We also highlight the features of DataScope that were most useful for that particular use-case.

4.1 Scientific Mashup to Explore Data from Co-clinical trials

This project was one of the initial drivers of DataScope. The overarching objective was to explore data, that is of data typically acquired during a Co-Clinical trial [14], and support an exploration of the integrated human+mouse data. In a Co-Clinical trial mouse models that replicate mutations seen in patients, are constructed, and used in preclinical trials that are run, in parallel with ongoing human trials. From a data integration and exploration perspective, one needs to create a data mashup and give users the ability to examine clinical data, genomic data, and support the exploration of radiology and pathology images. For this project we utilized data provided by Robert Cardiff (UC Davis) and Eran R. Andrechek (Dept. of Physiology, Michigan State University), and described here [10]. The objective was to explore a fusion of mouse model data and data from the Cancer Genome Atlas, both in breast cancer. The radiology images were stored in The Cancer Imaging Archive, while the pathology images were stored at Emory. DataScope allowed Dr. Cardiff and his group to take an integrated view of the data, and hypothesize the human correlates to pathway clusters that were seen in mouse models. The primary visualization was a SPLOM (Scatterplot Matrix), that was constructed on the Age and the PR, Her2 and ER mutation statuses. Users could create custom cohorts using a combination of these, then refine the cohort using clinical attributes, and finally visualize the digitized pathology images as well as radiology images. We highlight the most useful features in the following subsections.

[1]https://seer.cancer.gov/data/documentation.html



504/504 Selected

SPLOM DataTable ImageGrid

Show 10 entries Search:

PATIENT_ID	Diagnosis-Age	Disease-Free-Months	Disease-Free-Status	Ethnicity-Category	Mutation-Count	Neoplasm-Disease-Stage	Overall-Survival-Status	Overall-Survival-Months	Person-Gender	Race-Category	Tissue-Source-Site	Group	ER-Score	PR-Status	HER2-Score	Radiology	Pathology
TCGA-3C-AALJ	62	48.4	DiseaseFree	NOT HISPANIC OR LATINO		Stage IIB	LIVING	48.4	FEMALE	BLACK OR AFRICAN AMERICAN	3C	0	-1	40	-1	FALSE	FALSE
TCGA-3C-AALK	52	47.6	DiseaseFree	NOT HISPANIC OR LATINO		Stage IA	LIVING	47.6	FEMALE	BLACK OR AFRICAN AMERICAN	3C	0	-1	90	-1	FALSE	FALSE
TCGA-5L-AAT0	42	48.5	DiseaseFree	HISPANIC OR LATINO		Stage IIA	LIVING	48.5	FEMALE	WHITE	5L	0	-1	60	1	FALSE	FALSE
TCGA-5L-AAT1	63	48.3	DiseaseFree	HISPANIC OR LATINO		Stage IV	LIVING	48.3	FEMALE	WHITE	5L	0	-1	20	2	FALSE	FALSE
TCGA-A1-A0SE	56	43.4	DiseaseFree	NOT HISPANIC OR LATINO	18	Stage I	LIVING	43.4	FEMALE	WHITE	A1	5	3	100	1	FALSE	TRUE
TCGA-A1-A0SF	54	48.1	DiseaseFree	NOT HISPANIC OR LATINO	33	Stage IIA	LIVING	48.1	FEMALE	WHITE	A1	0	-1	100	-1	FALSE	FALSE
TCGA-A1-A0SH	39	47.2	DiseaseFree	NOT HISPANIC OR LATINO	72	Stage IIA	LIVING	47.2	FEMALE	WHITE	A1	10	-1	100	2	FALSE	TRUE
TCGA-A1-A0SN	50	39.3	DiseaseFree	NOT HISPANIC OR LATINO	38	Stage IIA	LIVING	39.3	FEMALE	WHITE	A1	0	-1	70	-1	FALSE	FALSE
TCGA-A2-A04Q	48	78.3	DiseaseFree	N/A	8	Stage IA	LIVING	78.3	FEMALE	WHITE	A2	9	-1	10	2	FALSE	TRUE
TCGA-A2-A04T	62	73.8	DiseaseFree	N/A	65	Stage IIA	LIVING	73.8	FEMALE	WHITE	A2	1	-1	10	2	FALSE	TRUE

Showing 1 to 10 of 146 entries (filtered from 504 total entries)

Previous 1 2 3 4 5 … 15 Next

Fig. 5. DataScope instance using co-clinical data from UC Davis

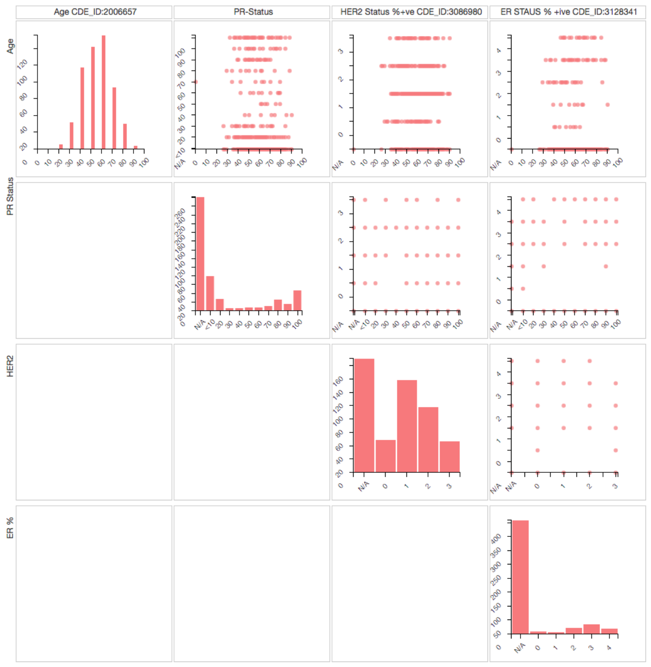


Fig. 6. Scatter plot of matrices to visualize and inspect correlations between *Age, PR-Status, Her2-Status and ER-Status* in the scientific mashup to explore co-clinical trial data

4.1.1 SPLOM((Scatter Plot Matrix)

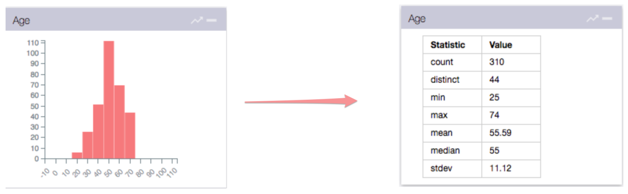
SPLOMs(*Scatter plot matrix*) are very useful for performing visual inspection of correlation amongst pair of variables [9]. Based on our discussions with the pathologists, we identified *Age, PR-Status, Her2-Status and ER-Status* as the attributes to be visualized on the SPLOM, as shown in 5. The goal was to be able to create cohorts of patients using the attributes from interactive filters and examine the correlations using the SPLOM.

Users can interact with the SPLOM by adjusting the adjust the filter in a histogram, or by defining a selection area in the scatter plots.

4.1.2 Linking to external applications

Pathologists were also interested in exploring radiology and pathology data associated with the cohorts. For these use-cases we were able to link a radiology explorer that fetches data using the The Cancer Imaging Archive (TCIA) API[2], also we were able to link it to an instance of

[2]http://www.cancerimagingarchive.net/



Statistic	Value
count	310
distinct	44
min	25
max	74
mean	55.59
median	55
stdev	11.12

Fig. 7. Users can flip the card using the statistics icon to view statistics about the attribute.

caMicroscope[3], which is a whole slide image viewer. The *DataTable* and *ImageGrid* visualizations can be configured to link out to such external applications.

4.2 Cancer registries

The NCI SEER (Surveillance, Epidemiology, and End Results) program collects epidemiological data on the incidence and survival of cancer in the United States [5]. This is a large dataset with diverse attributes. Our objective was to prototype an interactive exploration of this data with the potential to link to related datasets. To develop this we utilized the registry abstracts for anonymized breast cancer patients. A novel extension that was made, for this project, was the inclusion of summary statistics on each of the interactive filters. Users can flip the card on each interactive filter and be presented with pre-defined summaries such as mean and standard deviation. In future work we will add the ability to examine statistics such as age adjusted incidence rates. This use-case also demonstrated the need for geospatial filtering, such as the need to select a geographic subset on a map, refine the resulting cohort on demographic and clinical attributes, and then calculate incidence rates and survival rates.

In the subsections we highlight some of DataScope's features that were most useful in this use-case:

4.2.1 Visualizing geographical data

We obtained cancer registries from California, Hawaii, Utah, Iowa, Louisiana, Kentucky and Connecticut. We chose to visualize it as choropleth map to provide an easy way to visualize how the number of cases vary across different states.

4.2.2 Computing statistics for attributes

In the DataScope for SEER we also incorporated real-time statistics for attributes. Users can flip the card to view statistics on these attributes. Currently we compute *count, distinct, min, max, mean, median and standard deviation* for attributes with a numerical datatype. We also added support for custom statistics which can be used to plug in a formula to compute say age adjusted incidence rates etc. These statistics are also updated as additional filters are set.

5 Performance Evaluation

Using the client-server architecture was motivated by the need to handle large scale data. We designed experiments to run with different sizes of datasets. The datasets were randomly generated, and were composed of a combination different datatypes namely: enumerated, integers, floats etc. Half the attributes were filtering attributes the rest half were visualization attributes. The benchmark tests were performed using test data on a virtual machine running Linux Mint 14, having 8 GB ram, and 4x Intel(R) Core(TM) i7-2630QM @ 2.00 GHz processor.

From the results we observed that the filtering on the server and the AJAX response was the main bottleneck. The effect of SVG rendering performance was negligible compared to filtering and network latency. In Figure 8 we have plotted the results of only the total time for AJAX responses which includes the time for filtering and network latency. These times were averaged over 100 requests. We observe that the

[3]http://camicroscope.org

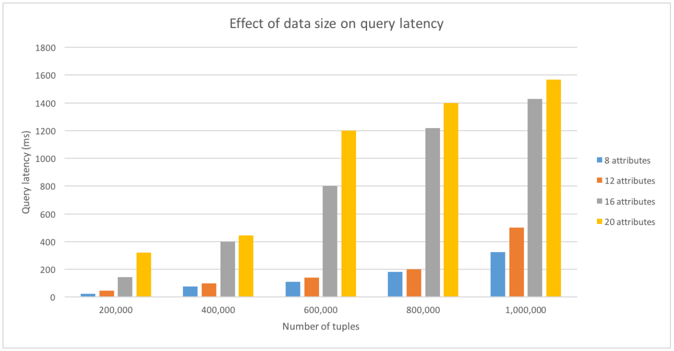


Fig. 8. Effect of size of data set on the filtering performance

system performance gradually decreases as we increase the size of the dataset. We also observed that the number of filtering attributes were the major performance bottleneck. DataScope is able to perform sub second queries on million records when the number of attributes are less than 12. Using a distributed columnar database can lead to better performance results across larger datasets.

6 Future Work

In this section we describe our plans and future work for DataScope. To address the follwing future work and other research questions, DataScope is freely available as an open-source software at *https://github.com/sharmalab/Datascope*.

6.1 Support for newer interactive filters and visualizations

Currently DataScope supports four interactive filters and ten visualizations. Our plan is to add support for newer visualizations like: *Box plot* and *Parallel coordinates*. Parallel coordinates have proven useful in spotting clusters in large multidimensional noisy datasets [6]. We also plan on adding Kaplan-Meier curves to study survival statistics.

6.2 Support for Vega

Vega is a visualization grammar for creating interactive visualizations [15]. In future releases we plan to add support for Vega in DataScope. Users would then be able to specify their visualization or interactive filter in terms of a declarative grammar instead of relying on the templated visualizations that we currently support.

6.3 Using a distributed OLAP backend.

The backend for datascope can be replaced by a distributed datastore. Druid is a distributed, columnar data store designed for exploratory analytics on large datasets [19]. In benchmark tests, Druid has show sub-second querying time for OLAP queries on 1 GB of data. The work would involve wrapping Druid's REST API to generate data structures that DataScope can consume. This would be a more scalable approach for handling larger datasets.

Acknowledgments

We acknowledge the generous funding that made this work possible. In particular, we would like to acknowledge the support of the NCI QIN grant (1U01CA187013-01, *Resources for development and validation of Radiomic Analyses & Adaptive Therapy*), NCIP/Leidos CTIIP Contract (*caMicroscope A Digital Pathology Integrative Query System*), and the Google Summer of Code (2014, 2015, 2016). We would also like to acknowledge CTIIP team (Ulrike Wagner and Ed. Helton) and Dr. Robert Cardiff and his team for the data and guidance provided during the course of the co-clinical data project.

References

[1] Crossfilter: http://square.github.io/crossfilter/.
[2] Datalib: http://vega.github.io/datalib/.
[3] Dc.js dimensional charting: https://dc-js.github.io/dc.js/.
[4] Google charts: https://developers.google.com/chart/.

[5] Surveillance, epidemiology, and end results (seer) program (www.seer.cancer.gov) research data (1973-2014), national cancer institute, dccps, surveillance research program, surveillance systems branch, released april 2017, based on the november 2016 submission.

[6] A. O. Artero, M. C. F. de Oliveira, and H. Levkowitz. Uncovering clusters in crowded parallel coordinates visualizations. In *Information Visualization, 2004. INFOVIS 2004. IEEE Symposium On*, pp. 81–88. IEEE, 2004.

[7] M. Bostock, V. Ogievetsky, and J. Heer. D^3 data-driven documents. *IEEE transactions on visualization and computer graphics*, 17(12):2301–2309, 2011.

[8] P. J. Curran and A. M. Hussong. Integrative data analysis: the simultaneous analysis of multiple data sets. *Psychological methods*, 14(2):81, 2009.

[9] J. Heer, M. Bostock, and V. Ogievetsky. A tour through the visualization zoo. *Commun. Acm*, 53(6):59–67, 2010.

[10] D. P. Hollern and E. R. Andrechek. A genomic analysis of mouse models of breast cancer reveals molecular features ofmouse models and relationships to human breast cancer. *Breast Cancer Research*, 16(3):R59, 2014.

[11] H. M. Kienle and H. A. Muller. Requirements of software visualization tools: A literature survey. In *Visualizing Software for Understanding and Analysis, 2007. VISSOFT 2007. 4th IEEE International Workshop on*, pp. 2–9. IEEE, 2007.

[12] L. Lins, J. T. Klosowski, and C. Scheidegger. Nanocubes for real-time exploration of spatiotemporal datasets. *IEEE Transactions on Visualization and Computer Graphics*, 19(12):2456–2465, 2013.

[13] Z. Liu, B. Jiang, and J. Heer. immens: Real-time visual querying of big data. In *Computer Graphics Forum*, vol. 32, pp. 421–430. Wiley Online Library, 2013.

[14] C. Nardella, A. Lunardi, A. Patnaik, L. C. Cantley, and P. P. Pandolfi. The APL paradigm and the co-clinical trial project, 2011.

[15] A. Satyanarayan, R. Russell, J. Hoffswell, and J. Heer. Reactive vega: A streaming dataflow architecture for declarative interactive visualization. *IEEE transactions on visualization and computer graphics*, 22(1):659–668, 2016.

[16] N. Shadbolt, K. O'Hara, T. Berners-Lee, N. Gibbins, H. Glaser, W. Hall, et al. Linked open government data: Lessons from data. gov. uk. *IEEE Intelligent Systems*, 27(3):16–24, 2012.

[17] 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.

[18] M. A. Yalçın, N. Elmqvist, and B. B. Bederson. Keshif: Out-of-the-box visual and interactive data exploration environment.

[19] F. Yang, E. Tschetter, X. Léauté, N. Ray, G. Merlino, and D. Ganguli. Druid: A real-time analytical data store. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*, pp. 157–168. ACM, 2014.