D3-Aggregate

Steven Raden
Informatics
stevengraden@gmail.com

Kevin Yang
Informatics
Kevinjyang126@gmail.com

Howard Lin
Informatics
hhowardlin@msn.com

INTRODUCTION

With the widespread popularity of the D3 library, users are increasingly able to generate powerful, customized, interactive visualizations of any given dataset. Data in the form of a CSV can be loaded through the D3 library for visualization. Unfortunately, D3 does not support a lot of functionality for manipulating and cleaning data. To begin manipulating a dataset, users need to write their own functions or use an external tool as there are currently not many front-end tools for this task.

Once a user has a properly formatted and cleaned dataset chances are they will need to manipulate their original dataset to display different kinds of information and to tell the story that is needed. This kind of manipulation often comes in the form of aggregate functions such as averaging, summing, counting, and finding the max and the min. Each of these functions can have a variety of extra parameters which will help to narrow or limit the amount of data being returned.

To help accomplish important tasks like aggregate functions and cleaning datasets, we created a tool called D3-Aggregate. D3-Aggregate is a front-end, javascript and jQuery based tool to help aggregate the data stored in arrays of javascript objects. The original data format that this tool is built to process is based on the information returned from the javascript library D3, a library specifically designed to work with datasets and visually display them with javascript. With D3, a user can take in a .CSV file and compile it into an array of objects. A user can then pass these arrays into D3-Aggregate and quickly get back data in the same format as before, but combined and aggregated the way the user requested.

RELATED WORKS

Aggregation of data through averaging, summing, etc. is not a new topic and there are many ways to accomplish such a task. With D3-Aggregate we reviewed some of the popular tools, what was good, and what was bad about each. We intend to analyze the good elements and combine them into a new and more powerful tool.

Structured Query Languages (SQL)

SQL is a very common language that is used throughout some of the largest servers. Using easy to use commands a user can query a SQL database and request for very specific or very broad amounts of data across multiple tables. In more recent years, functionality has been added to SQL to perform aggregate functions on the stored data. There are many ways to customize the aggregated data through the use of grouping and the WHERE and HAVING commands which help to filter data.

SQL Databases are hosted on a server. Unless the data is already on a SQL database, a user will initially have to upload their entire dataset which can be difficult and time consuming. Once the data is on the database, the user will have to query the database each time they need different data and they will have to wait for data to be returned. Depending on the server hardware, the amount of data being requested, and the number of concurrent users, the wait time can be inconsistent and range from very quick responses to extremely long. Using SQL also requires a user to learn an entirely different language in order to get the information needed.

The best part about SQL is that it has a very robust set of commands and a user can customize their requests to retrieve anything available on the database. This could be anywhere between a single value to a set of a million rows. SQL also provides a multitude of tools such as aggregate functions to modify the data before it is returned. We hope to be able to bring similar functions to D3-Aggregate. Particularly we want to be able to provide functions like average, min, max, count, range, sort, and have some form of filtering. Optimally we would have similar language formats as SQL to be able to accomplish the same complex tasks that SQL does.

Excel (Pivot Tables)

Excel is suitable for small data sets, originally designed as a spreadsheet, the program from Microsoft now has many analytical and statistical functions that make it similar to a simplified database. Aggregate functions can be completed through Excel functions or Pivot Tables. Pivot Tables are used to perform cross tabulations on columns of data that are linked to the Pivot Table. The Pivot Table can then perform aggregate functions or sorting on the dataset provided.

While Excel can be found on most computers, it does require practice and knowledge about Excel in order to be used properly. Excel is a very hardware intensive tool and quickly begins to slow down when working with larger datasets. There is also the issue that Excel is intended to be used by only a single user at a time and takes complex interactions in order to change the data. In no way is it able to be used on a web page with customized user interactions.

Excel makes combining, manipulating, and transforming large quantities of data very simple for the average user. We intend to emulate the ease of use and complex operations for D3-Aggregate. This will allow users to run Excel like tasks on web browsers in order to generate their visualizations. We also intend to provide cross-tabulations where we would compare and subdivide data with multiple columns.

Datalib

Datalib is a javascript library developed by the UW Interactive Data Lab and released mid-2015. The purpose of this library is to provide functionality for loading various data types, including CSV, type inference of data that is passed in, sum functions, group-by aggregate queries as well as string templates. The Datalib features many useful methods that provide SQL like queries.

The Datalib library requires a small amount of initial technical setup using both node.js and gulp to build the library prior to use. After the library is built, the user is able to begin using the functions provided by Datalib. To non-technical people it definitely requires more work than loading D3 or other javascript libraries like jQuery.

Datalib is a very complex and powerful Javascript library that provides many features and functions to manipulate datasets. We intend to provide a very similar set of commands and functions, but want our tool to be much simpler to initialize and begin working with. We also hope to reduce the necessity of other libraries like D3. Our tool is currently based around the format of D3 datasets, but it will work with any dataset that has a similar format to D3.

METHODS

To make our tool as simple as possible to use we developed it as a library. All the user needs to do is include our Javascript file "aggregate.js" in the header of their HTML page. The user will be able to access functions within the Agg object such as our average function which can be called using "Agg.avg(dataArray, column, GroupOfColumns);." The users will have the ability to aggregate by sum, count, average, max, and min. Users will also have the ability to find the range, sort, filter, and take a stated number of rows. These functions can further help the user to find the subset of data that they wish to visualize.

General

In general for the aggregation functions starts by accepting the dataset, the column to aggregate upon, and the group of columns to group by. We then traverse through the dataset and add the appropriate column values into a map. For the map, we define the key as the values for the columns provided in the "group" parameter with the value being the aggregated value. For example, if the group parameter was passed ["WinnerName", "Gender"] then the key for the map would be "Roger F._Male". This creates a key that is unique to the requested groups and ensures the proper data is being aggregated. While traversing the dataset, if the same grouping exists in a row then we aggregate that new row's value for that column and simply add it to the map with the same key. Finally, we convert this map to an array of objects with the fields being each grouped column and the newly formed aggregate column.

Average

The Average function will follow the general description and is used to find the average value for a specified column. However, there is an additional step for average to function properly. This function calls upon the sum and count functions in order to have the information necessary to create an average. The average function receives a map from both the sum and count functions. Both maps are traversed to grab the values of count and sum for the specified group of columns. This allows us to create an average for that combination and we enter that as a value in a new map. The new map will be converted to the array of objects that works with D3.

Sum

The Sum function follows the general description and is used to find the sum for a specified column. Summing is achieved by traversing the array of objects provided and adding up the values in the specified column. Grouping of columns is allowed and the summed rows are placed as the value inside a map. This is then converted to an array of objects that works with D3. If called internally by another function, Sum returns the map that it generated.

Count

The Count performs differently from the other aggregate functions. Count simply counts the number of occurrences for the specified group of columns. This generates a map which is then converted to an array of objects that works with D3. If called internally by another function, Count returns the map that it generated.

Max & Min

The Max and Min functions operate similarly to the others, but focus on finding the max or min value for the specified group of columns. This function will not return just a single max or min value for the entire dataset. Like the other functions it will return an array of objects that works with D3. Each object will contain the information for the grouped columns, along with the minimum or maximum value found for that group.

Range

The Range function is what a user might typically expect when thinking about minimum or maximum. The user may wish to get the minimum and maximum for a given column in the dataset in order to establish the range or scale of their visualization. This range is typically used to create the axis or the slider range for the visualization. This accepts the dataset and the column to look at for the min and max. The user is provided with a size 2 array where array at 0 is the min and array at 1 is the max.

Sort

The Sort function allows the user to sort the dataset in an ascending or descending order based on a particular column. To use the sort function, a user simply provides the dataset, a column to check, and either "ASC" or "DESC" to specify ordering ascending or descending. This is very similar syntax to the SQL's ORDER BY command. To begin sorting, we copy the values of the dataset passed in order to avoid manipulating the original dataset. Then the traversal of the data begins and each row is arranged based on the provided column and the desired ordering. The data is able to be sorted by either string, number, or any object that contains a comparison method. We make sure to remove upper and lower case during string comparison as this could interfere with the sorting accuracy as upper and lowercase letters are evaluated differently. The user is returned an array of objects that works with D3.

Your references should be published materials accessible to the public. Internal technical reports may be cited only if they are easily accessible (i.e., you provide the address for obtaining the report within your citation) and may be obtained by any reader for a nominal fee. Proprietary information may not be cited. Private communications should be acknowledged in the main text, not referenced (e.g., "[Robertson, personal communication]").

Take

The Take function allows the user to specify the amount of rows they want to use for their visualization. This accepts data and a limit as parameters. Take simply traverses the provided dataset for "limit" amount of times and adds each element to a new array to be returned. This is still an array of objects that works with D3.

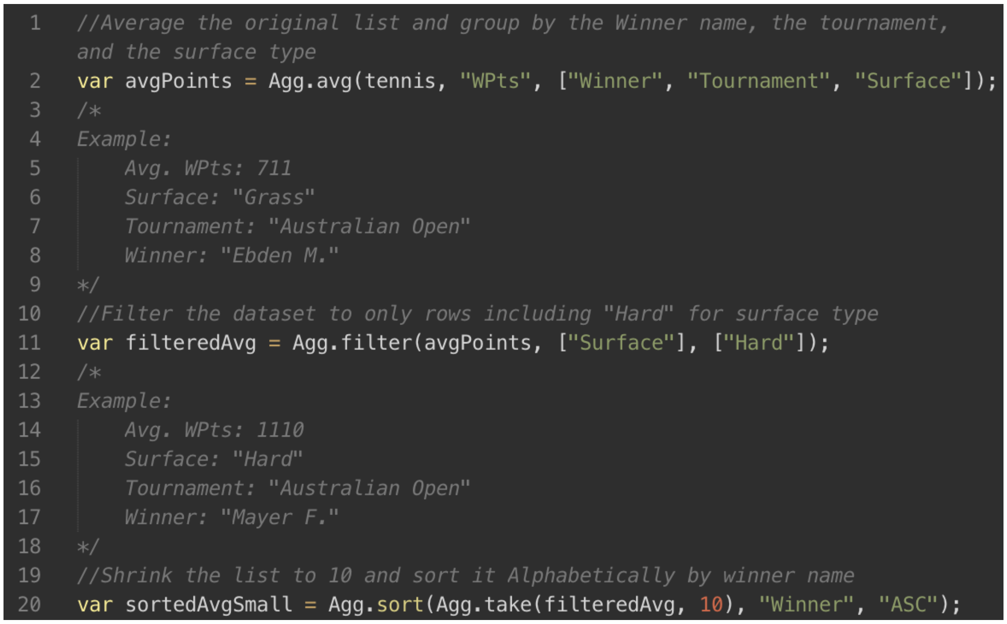
Filter

The Filter function accepts a dataset, an array of columns to check, and an array of the corresponding values to check for. What the function does is search through the dataset, checking the values of each row for the values provided by the user. If all the values match perfectly then that row is put in a new array and the algorithm moves onto the next row. Once all the rows have been processed, the function will return the filtered array of objects back to the user. This is an extremely useful and quick function when applying filters during a visualization. With a single function call, the user is able to reduce their dataset down to exactly what they need and it can be immediately used with the other D3 functions to display the updated visualization.

RESULTS

Example 1

Figure 1. shown below is one example use case for our tool. The user begins with a large dataset called "tennis" which is the array of objects returned by the D3.csv function. In this case the user wants to get an average for the amount of points each winner had. Due to how aggregation works, the user had to cut some information and decided to group the data by the Winner, the Tournament played, and the Surface played on. The user then decided that they only cared about "Hard" surface types and used the filter function to remove all rows not including "Hard" as the surface. Finally, the user only wanted the top 10 rows from the filtered list and afterwards they sorted the list alphabetically by the winner's name. From here, the user can run any D3 function, or any functions which use similarly formatted data, to produce the visualizations that they need.



```
//Average the original list and group by the Winner name, the tournament, and the surface type
var avgPoints = Agg.avg(tennis, "WPts", ["Winner", "Tournament", "Surface"]);
/*
Example:
    Avg. WPts: 711
    Surface: "Grass"
    Tournament: "Australian Open"
    Winner: "Ebden M."
*/
//Filter the dataset to only rows including "Hard" for surface type
var filteredAvg = Agg.filter(avgPoints, ["Surface"], ["Hard"]);
/*
Example:
    Avg. WPts: 1110
    Surface: "Hard"
    Tournament: "Australian Open"
    Winner: "Mayer F."
*/
//Shrink the list to 10 and sort it Alphabetically by winner name
var sortedAvgSmall = Agg.sort(Agg.take(filteredAvg, 10), "Winner", "ASC");
```

Figure 1. Example 1 Sample Query With Caption Below

Example 2

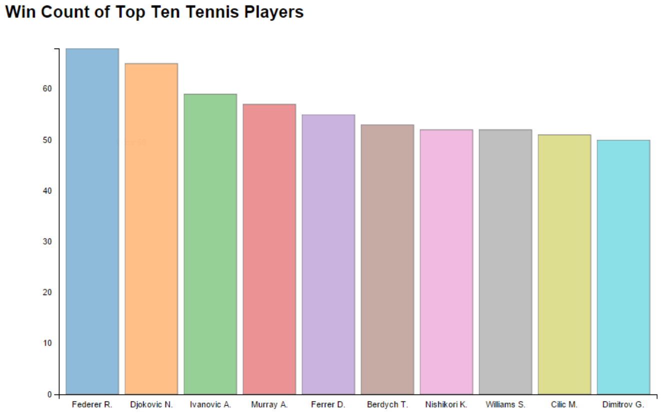
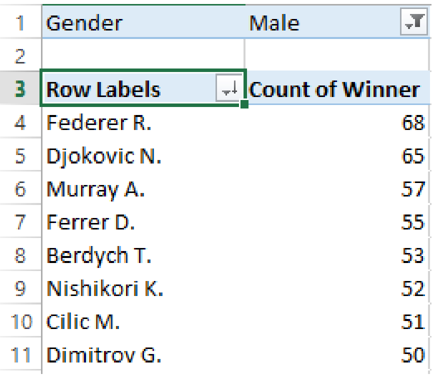


Figure 2. Top Ten Players by Wins Visualization

Through the use of d3-aggregate, users will be able to quickly and easily load datasets and return summary statistics on groups of columns. With the flexibility of the library, as well as existing functionality provided by D3, users will easily narrow and modify data sets in order to tell the story and provide the data they need. For example, in order to make the visualization in Figure 1, a dataset containing tennis match data was used. The dataset has 5076 rows and contained 42 columns. The data was primarily organized by matches played, with important data such as the tournament where the match occurred, the round the match was played, the winner and loser of the match and their rankings at that time, etc. Because players were associated with matches rather than the other way around, on first glance it is very hard to eyeball the player statistics within the data set.

For our example, we wanted to see how many wins a player had and then find the top 10 players with the most wins. We used the "count" function to find out how many wins a player had. Next we called the "sort" function to sort players from most wins to those with the least. Then we used our "take" function to limit our data set to the top ten players. We utilized three methods to modify our data set and created a simple visualization. The beauty of our library is that users can perform queries on the fly because our functions return the data in a consistent format that can be used with tools like D3.



Gender	Male
Row Labels	Count of Winner
Federer R.	68
Djokovic N.	65
Murray A.	57
Ferrer D.	55
Berdych T.	53
Nishikori K.	52
Cilic M.	51
Dimitrov G.	50

Figure 3. Pivot Table of Top Ten Players by Wins

If we were to perform this query in Excel, we would first select the data and build a pivot table. While building the table we would look at "Winners" as our rows and a count of the "Winners" as our values. We would then sort by the count and perhaps filter on gender. These results are seen in

Figure 3. Compared to our method, Excel seems just as simple and quite fast, but it does not maintain the intermediate steps of the dataset when a pivot table is being built. With our library, a new dataset is returned every time a function is called, so we can easily perform queries on the fly where new lines of inquiry are pursued based off analysis of the data returned. There are also very few ways to customize interactions with Excel and to put the results into a visualization tool.

If we were to use SQL to manipulate our data it would look like the code fragment in Figure 3. We would use the COUNT, GROUP BY and ORDER BY SQL functions. The SQL Query is shown below as Figure 4. The problem with utilizing SQL is that it requires understanding of another programming language, requires a database management system to be running on a server, and, to properly format everything, it would require using multiple tables and inserting all the information into the DBMS. Having the data stored in a CSV, while difficult to update, is easily readable in many forms and easily shared through email, USB, online or other formats. With our lightweight library, D3, and javascript, which comes available in every browser, anyone can query and modify datasets, while also building custom visualizations.

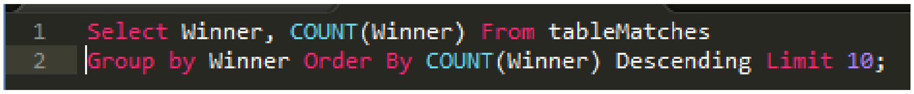
Run Times:

Most of the functions within this library are O(mn) as they have to loop through every row in the dataset (n) and loop through each column for the row (m). This is due to the fact that we need to compare each row with each of the specified columns. However there are some functions like take, sort, and range which run in O(n) time. In these functions we don't need to check every single row for a variety of columns. We just need to check a specific column's value or have a limit passed in by the user.

DISCUSSION

The audience should learn that setting up the data to use for the visualization is very important and time consuming. Whenever one wants to create a visualization, they have to make sure that they have the data that represents the story they are trying to convey. At times the user may wish to have aggregated columns of data for a column in their dataset in order to visualize it. For example, the user may want to count the number of wins for a tennis player on a particular surface. This data isn't present in the current dataset, and a user would have to search through the dataset and calculate this information. The user can utilize Excel's Pivot Tables to manipulate the dataset. This would first require the user to understand how Pivot Tables work and then have to save in another format to actually use it elsewhere. The user can also utilize SQL to aggregate upon the dataset. This would require the user to understand SQL and they would have to create a database and then upload the data into a table. To get the data, the user would have to write a query to aggregate the columns. There is also the new data library, Datalib, that is being worked on and was recently released, but it requires a few Javascript libraries to function and requires tools like Node.js in order to initially set it up.

As a result of the difficulties associated with data manipulation, the audience should know when seeing the various functions inside our library that it isn't an easy process. They should learn that manipulation of data is an important step in the process of creating an interactive visualization. In particular, interactive visualizations will need frequent updating every time a user changes the values on the visualization. If the updates take a long time then users will be more likely to leave. Users of D3-Aggregate will also begin to realize the vast amount of information that exists within a single dataset, even if it isn't explicitly stated.



```
Select Winner, COUNT(Winner) From tableMatches
Group by Winner Order By COUNT(Winner) Descending Limit 10;
```

Figure 4. SQL Query For Visualization

FUTURE WORK

This initial iteration of D3-Aggregate provided some basic, core functionality which can immediately be used by anyone. In the future we would like to expand our library to include additional functionality such as sorting for multiple columns, increasing the abilities of the filter function, and being able to clean datasets. While we originally sought to simply provide aggregate functionality for data returned by D3, we began to realize that there are an enormous amount of functions which would make data manipulation much simpler and there are currently very few, simple to use, solutions.

What D3-Aggregate is still lacking is robustness and breadth. There are many more functions and tools we would like to include to make the user's life simpler. We intend to include functionality to have more advanced sorting which includes tie breaking with other specified columns. There will also be advanced filtering that allows the user to provide an intuitive and easy to use set of commands which will return very specific data. The more advanced filtering will expand on our current model by allowing "less than" and "greater than" commands when working with numbers and they will be able to use wildcards for string comparison, much like SQL. Functions for data formatting and cleaning are also in our future goals because many datasets found online contain inconsistent, poorly formatted, and undefined values which break or disrupt the creation of a visualization.

As D3-Aggregate is early on in development, there was less focus on efficiency and robustness and more on getting functions to work. We currently have an unwanted level of redundant code which needs to be separated into repeatable functions while still retaining the unique elements that exist in each function. There is also very limited error checking and, while we have been testing on our dataset, we are unsure about situations where errors may occur with other datasets. We usually check for undefined values and trust the user will provide accurately typed input, however many other errors could occur that we have not tested for. Adding functionality to ignore casing is simple to implement and will frequently help with user input. We are also looking into improving efficiency on our functions to ensure the fastest possible runtimes. A poorly rated runtime will be an enormous deterrent for users with large datasets because functions with poor efficiency will get exponentially slower as the dataset grows.

DOCUMENTATION

Average

O(nm)

Loop through a dataset (Array of Objects with columns as fields) to calculate the average value for a column. Provide the other columns to group by. Will return an Array of objects in the same layout as given.

param {Array[object]}	data	Array of objects. Fields in objects are the columns from dataset
param {String}	avgCol	The column that will be averaged
param {Array[String]}	group	An array of column names
return {Array[object]}		An array of objects with the grouped and average column as fields.

Sum O(nm)

Loop through a dataset to calculate the sum for a given column. Provide an array of column names to group the data by.

param {Array[object]}	data	Array of objects. Fields in objects are the columns from dataset
param {String}	sumCol	Column to be summed
param {Array[String]}	group	An array of column names in the format of a
param {Boolean}	internal	Whether the function is being called internally or not. Changes what will be returned.
return {Array[object] || map<String> = Sum}		Depending on if it is internal call or not, this function will return an Array of objects or a map

Count O(nm)

Count the number of occurrences of a set of values. Can be grouped in order to specify the values being searched for.

param {Array[object]}	data	Array of objects. Fields in objects are the columns from dataset
param {Array[String]}	Group	An array of column names in the format of a
param {Boolean}	internal	Whether the function is being called internally or not. Changes what will be returned.
return {Array[object]}		An array of objects with the grouped columns and count as fields

Max/Min O(nm)

Find the maximum or minimum set of values based on a specific column. Allows the grouping of elements if desired.

param {Array[object]}	data	Array of objects. Fields in objects are the columns from dataset
param {String}	maxCol/minCol	Column to find the max or min in
param {Array[String]}	group	An array of column names in the format of a
return {Array[object]}		An array of objects containing the Maximum or minimum set of values for the grouped data

Range O(n)

Find the minimum and the maximum of a dataset based on the column to search through

param {Array[object]}	data	Array of objects. Fields in objects are the columns from dataset
param {String}	col	String of column to find the min and max of
return {Array[min, max]}		An array of numbers. 0=Min 1=Max

Sort O(n)

Sort the rows in the dataset based on a column and the direction you would like to sort by. Works on number and strings.

param {Array[object]}	data	Array of objects. Fields in objects are the columns from dataset
param {String}	sortCol	String of column name to sort by
param {String}	direction	Accepts "asc" to sort in an Ascending style or "desc" to sort Descending.
return {Array[object]}		An array of sorted objects.

Take O(n)

Reduce the amount of rows in your dataset. Simply grabs the top rows of the dataset and returns a new array of them. If limit is greater than the length of data then will return original data.

param {Array[object]}	data	Array of objects. Fields in objects are the columns from dataset
param {Number}	limit	The number of rows to keep
return {Array[object]}		An array of objects. Fields in objects are the columns from dataset. Number of objects based on "limit".

Filter O(nm)

Loops through the dataset and searches for exact matching data between the filterCol and filerValue parameters. Only keeps rows which meet all the filter values exactly. filterCol and filterValue need to match column to value positioning within the arrays.

param {Array[object]}	data	Array of objects.
param {Array[String]}	filterCol	Array of column names to match with the values
param {Array[values]}	filterValue	Array of values to check for in each provided column
return {Array[object]}		Array of objects. Fields in objects are the columns from dataset

REFERENCES

1. D3:Data-Driven Documents Michael Bostock, Vadim Ogievetsky, Jeffrey Heer IEEE Trans. Visualization & Comp. Graphics(Proc. InfoVis), 2011
2. Heer, J. Satyanarayan, A., Moritz, D., Prud'hommeaux, M., Wongsuphasawat, K., & Ros, I.(n.d). UWdata/datalib. Retrieved June 12, 2015, from https://github.com/uwdata/datalib
3. Burgess, A. (2012, October 4). Build Your First JavaScript Library - Tuts Code Tutorial. Retrieved June 12, 2015, from http://code.tutsplus.com/tutorials/build-your-first-javascript-library--net-26796