Team Members:
Christopher Copper
Philip Eittreim
Jeremiah Jekich
Andrew Reisdorph

Client:
Brian Krzys

June 17, 2014

Introduction

Newmont Mining is a resource extraction company with a research and development office located in Englewood, Colorado. Founded in 1921, Newmont has acquired a large quantity of data over the years and has an interest in analyzing such data to inform business decisions. An increasingly effective means of enabling this analysis is data visualization. The main goal of this summer field session project is to develop an instant visualization tool that gathers and analyzes user-identified data to provide the user with a best-guess visualization of that data that is both informative and interactive. Once provided with the initial visualization, the user should then be able to select alternative visualizations to explore various methods of displaying the data.

Requirements

Listed below are the functional and non-functional requirements of the Instant Visualization tool.

Functional

- Create a website with a textbox requesting a URL.
- Scrape the URL provided for tabular data.
- For each set of tabular data found analyze and make a best-guess for the best way to display this data.
- Produce an interactive graph based on the best-guess of data provided.
- Display additional information when the user mouses over a section of the graph, such as the data associated with the selected data point.
- Allow the user to select alternate visualizations if the best-guess display is not optimal.
- Redisplay the data scraped alongside the visualization.
- Allow the user to customize aesthetic properties of the visualization.
- Provide a means for the user to download a static version of the visualization they create.

Non-Functional

- Graphs are produced using a client-side library.
- GitHub is used for source control.

- Documentation of the software is done primarily with the use of JSDoc.
- The system consists of four separate modules which perform the following:
    - Parsing and scraping the data.
    - Formatting the data and making graphing decisions.
    - Displaying the graphs.
    - Facilitating of communication between the other modules.
- Tabular data is parsed into JSON for further processing (visualization production).
- Testing and deployment is performed on an Amazon EC2 server, provided by Newmont.
- Selecting alternate visualizations is accomplished client-side in real time.
- Comprehensive integration and unit testing has been performed.

System Architecture

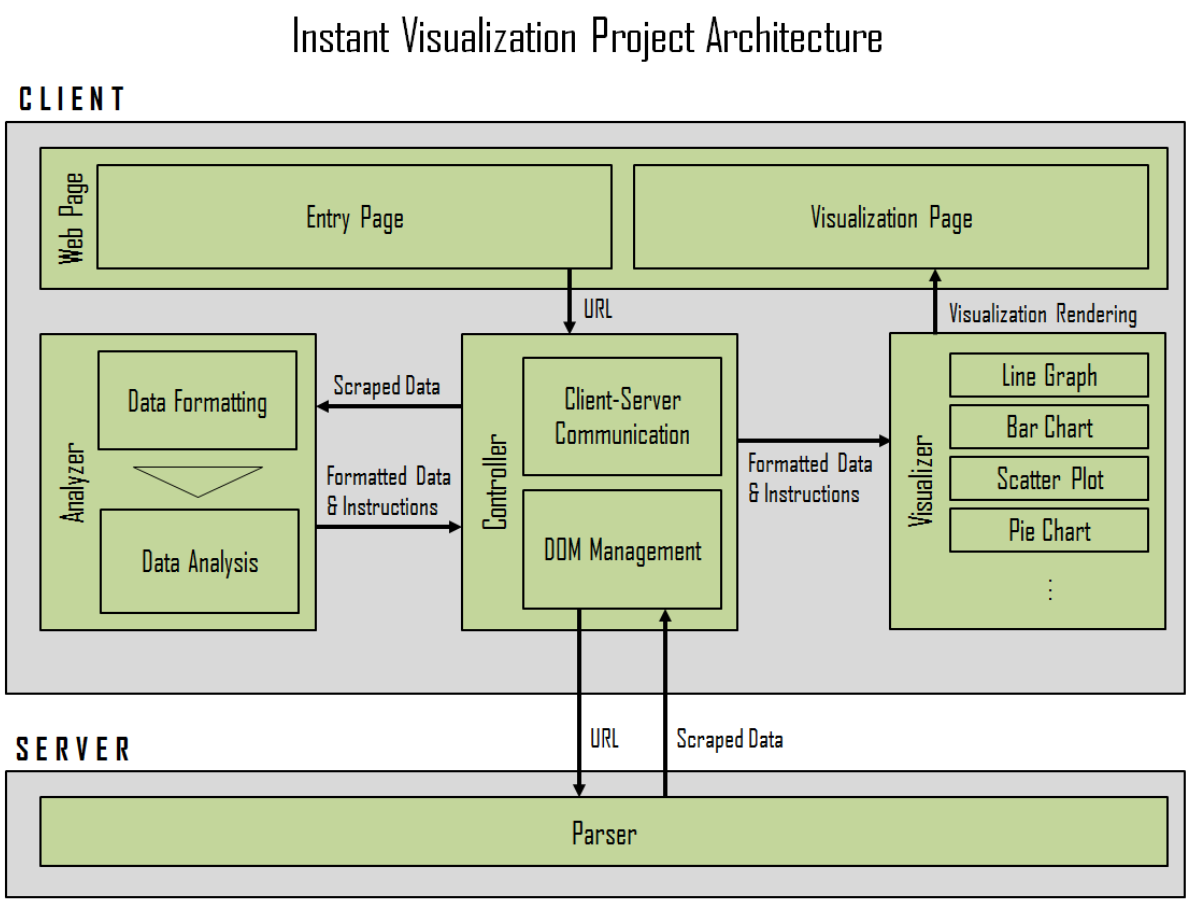

Figure 1: Instant Visualization Project Architecture

Figure 1 shows the overall system architecture of the Instant Visualization tool. The system consists of four primary modules: Controller, Parser, Analyzer, and Visualizer. Additionally, the web page component exists in two different states: Entry and Visualization.

Entry Page

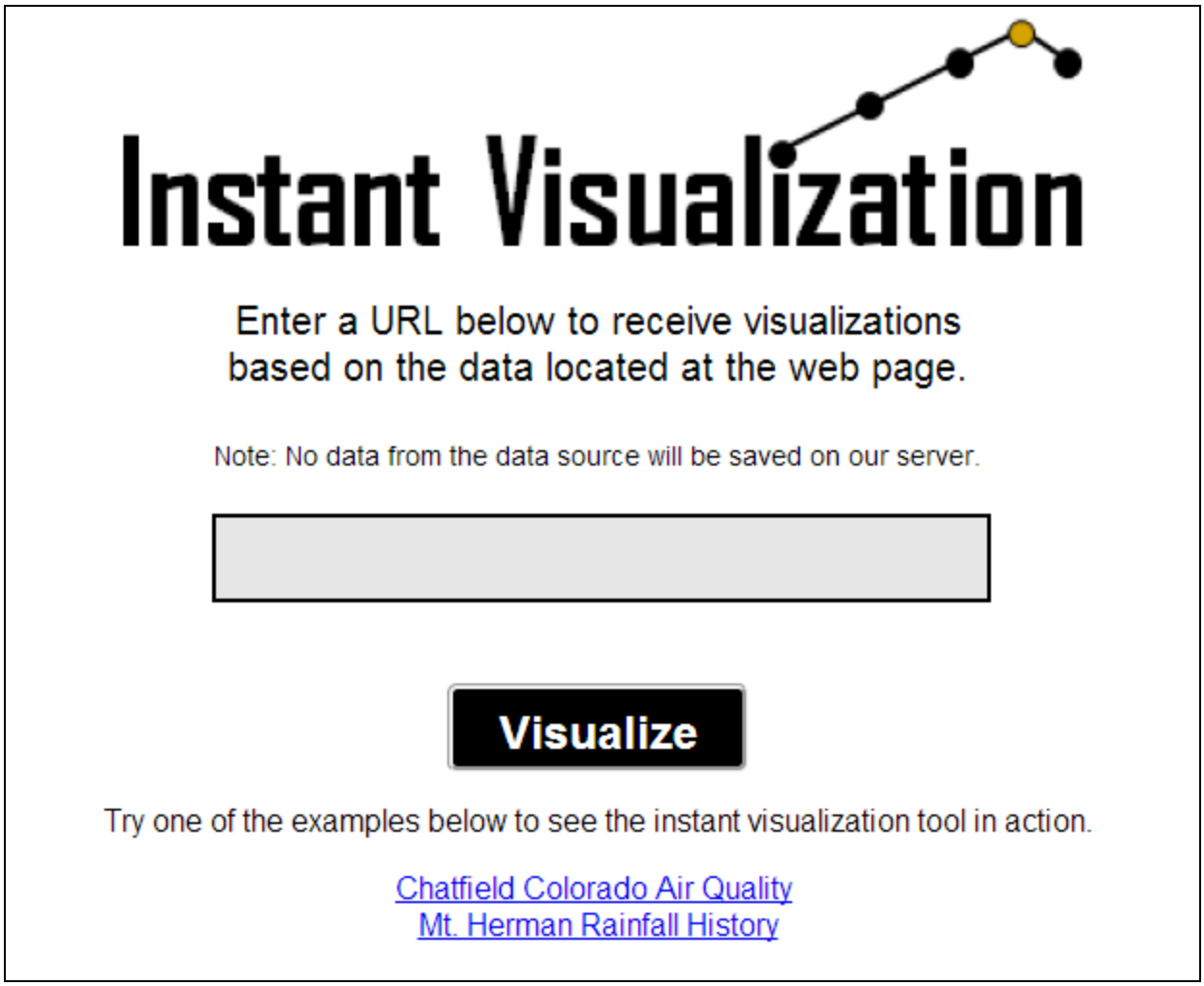

Figure 2: Entry Web Page

The Entry web page for the Instant Visualization tool, which can be viewed in Figure 2, is the user's first view of the tool. It consists of the following:

- The title/logo of the tool.
- Instructions on how to submit data to the tool.
- A textbox into which the user can enter a URL.

- A submit button the user can use to submit the provided information and receive visualizations.
- A disclaimer communicating to the user that no user-provided data is saved on the server.
- A set of links that automatically run the tool with example data to allow the user to see how the tool works..

Controller

The Controller module's primary responsibilities include facilitating communication between all subsequent modules and managing the web page structure.

When the user submits a URL, the Controller sends it to the Parser. After parsing, the data is sent back to the Controller and passed on to the Analyzer. Likewise, the Controller provides all other modules with their respective inputs, and receives the corresponding outputs in return. In doing so, the Controller is able to manage the entire tool's pipeline.

The Controller also handles the structure of the HTML page, including the switch between the Entry page and the Visualization page. It manages the placement and design of all components in the user interface, and is responsible for handling all user interaction.

Parser

The Parser module is responsible for obtaining the data located at the URL provided by the user. It accomplishes this using a server-side JavaScript library and PhantomJS, a headless WebKit browser with a Javascript API. The Parser arranges this data into a form it can then send back to the Analyzer module client-side.

The Parser is designed to be modular so that the data source and the necessary parsing specific to that source can be changed. For example, to allow for CSV files as input, a new parser that extracts data from CSV files can be used instead of, or alongside, the web page parser.

Regardless of the data source, however, the Parser extracts, formats, and transmits the data to the Analyzer according to a predefined API. The specifics for this API exist in the source code and JSDoc.

Analyzer

The Analyzer module is responsible for taking data from the Parser and generating instructions for the Visualizer. These instructions specify what columns to use for each applicable visualization type for each dataset.

To make decisions, the Analyzer relies heavily on the analysis of column data types. To do this the analyzer uses a sophisticated type handling system. The type handler is responsible for tasks like correctly typing all of the data. It can handle a variety of types, such as integers, floating point values, and strings. In addition to checking for types, cells with no data are given default values and completely empty rows are removed from the dataset.

Once a table has been fully typed, it is checked for valid dimensions and values. Tables that only contain a single row or column are discarded. Similarly, tables that only contain string data are also discarded because string data cannot be used as a dependent variable.

The other primary responsibility of the Analyzer is data analysis. The Analyzer looks at the newly formatted data and attempts to determine which subsets of the data are useful and capable of being visualized. Many web pages contain more than one set of data. It is the Analyzer‘s responsibility to identify these sets and make judgements about which ones can be adequately visualized. The uniqueness of each column is taken into account when making these decisions. Uniqueness is determined by counting unique entries in the given column and calculating them as percent of the total entries. Some visualization types work best with the independent variable as the least unique variable, such as the bar chart. Other visualization types make use of the most unique variable as the dependent variable, such as the line graph.

Once the Analyzer has identified relevant data sets, it determines the various ways the data can be visualized. Of the possibilities generated for each dataset, the Analyzer determines which option is most likely to be considered by humans to be the best visualization method. This is called the best-guess visualization.

The Analyzer packages all relevant table and visualization data and sends it to the Visualizer module via the Controller. The data is passed to the Visualizer according to the API in Figures 5 and 6.

Visualizer

The Visualizer module is responsible for generating and displaying the visualizations based on the data and instructions provided by the Analyzer module. It knows how to create visualizations for a variety of visualization types, including line graphs, bar charts, pie charts, scatter plots, and more.

The Visualizer module is itself modular in nature, as new visualization types can be added or removed to adjust its capabilities to the requirements of the tool. Additionally, this means that different visualization generation techniques can be used for different visualization types. For example, the initial implementation of the Instant Visualization tool utilizes D3.js, a JavaScript visualization library, to create it's charts and graphs. However, if D3.js proves insufficient for a specific visualization type, the Visualizer's definition of how to generate that specific visualization type could utilize some other external library, while other visualization types still used D3.js. This means the Visualizer can adapt to a wide variety of applications.

The visualization implementations for the Instant Visualization project produce visually appealing charts and graphs that have some element of interactivity that allows the user to dive deeper into the data and to focus on the areas of the visualization that interest them the most.

In this web-based implementation of the Instant Visualization tool, the Visualizer follows the instructions provided by the Analyzer to visualize the given data, presenting the results in the user's browser.

Visualization Page

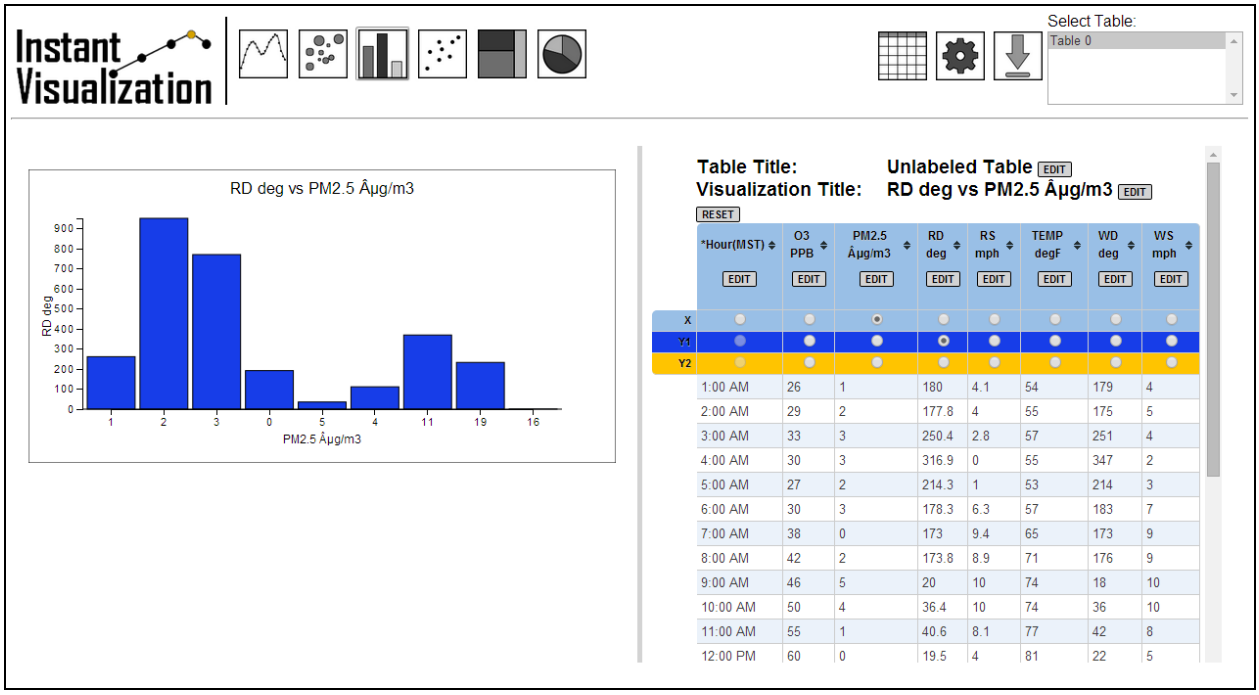

Figure 3: Visualization Web Page

The Visualization web page, which can be viewed in Figure 3, displays to the user the visualizations generated by the Visualizer. It consists of the following:

- The title/logo of the tool.
- A selection box that allows the user to select which table of data from the source URL to visualize.
- A data table that redisplays the data collected from the selected table.
- An icon that allows the user to toggle the data table on and off.
- A set of icons that allow the user to switch between different visualization types.
- A pane containing options to customize aesthetic properties of the visualization, including colors, size, and margin dimensions.
- An icon that allows the user to toggle the customization pane on and off.
- The best-guess visualization for the currently selected table.
- An icon that, when selected, will download an image of the current visualization.

The Visualization web page is the final state of the tool. Here, the user can interact with the visualization and the data table, and the system pipeline has reached its end.

Technical Design

Two of the Instant Visualization tool's more unique components are the server-side web page parsing and the visualization rendering.

Web Page Parsing

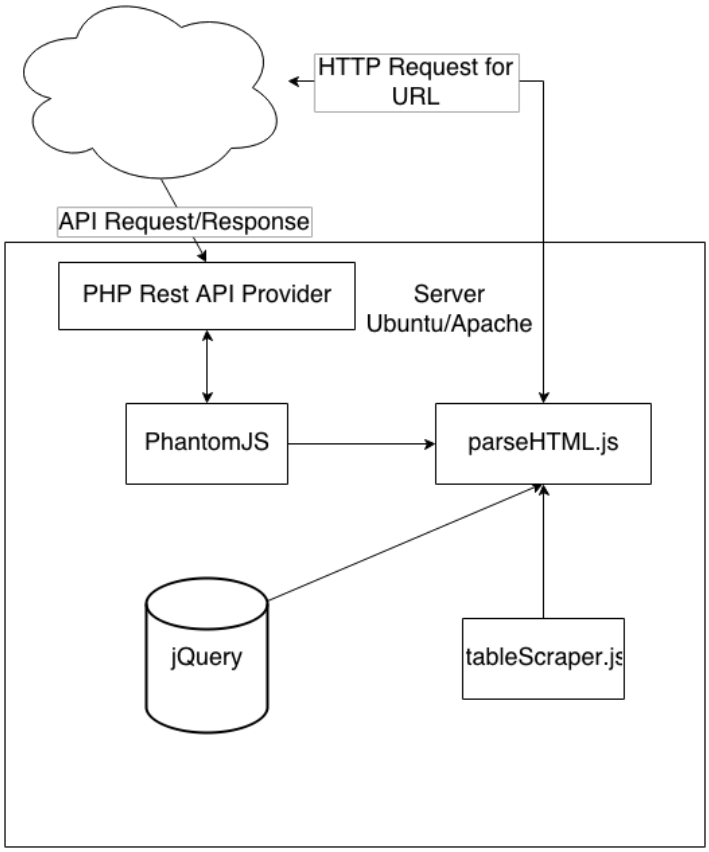

Figure 4: Server-Side Architecture

The first step in creating visualizations for a given URL is to scrape and parse that URL for data. The Instant Visualization tool scrapes URLs for tabular data, and returns this data to the client for visualization. First, the server receives a REST (representational state transfer) API call from a client, which contains the URL to parse. REST is a standard for creating APIs using a web server and HTTP GET, POST, PUT and DELETE to perform the API functions. The API endpoint is a PHP script that will validate the request. If it is valid it will then launch a PhantomJS process to do the parsing/scraping of the URL.

PhantomJS is a headless browser with a scriptable interface that allows us to request and virtually render a web page which can then be parsed for data. Being able to parse a rendered page is a key feature of the product. Modern websites use JavaScript to dynamically load and generate content which is not visible in the underlying HTML of a given page. Traditional scrapers would read this code and search for the desired content; however JavaScript content would be unavailable to these methods. PhantomJS works around this by rendering a page virtually and then executing JavaScript code which traverses the DOM (Document Object Model) and finds tabular data. In a normal browser websites from different URLs are sandboxed and cannot interact with each other for security purposes such as restricting XSS (cross site scripting) attacks. PhantomJS bypasses this issue.

Once the PhantomJS process is started, it first loads a JavaScript file (parseHTML.js in Figure 4) that provides a basic framework for the table scraping/parsing. This includes error handling, URL loading, and loading the table scraper logic. Once a URL has been requested and loaded, PhantomJS then injects our scraping code into the requested URL along with the current version of jQuery (jQuery and tableScraper.js in Figure 4). The table scraping logic makes use of jQuery's powerful selectors to quickly find table tags that exist in the DOM.

Once a table has been found the scraper begins to parse the table to ensure that it is a valid table. The table will be rejected if the table is missing cells, makes use of either HTML rowspan or colspan attributes. Any of these three represent tables that are incomplete or tables that are being used for structure and not data. Tables are also checked for a variety of configurations. For example, some tables may not have a body tag or head tag but still may have a header row. If a table is found to be valid, all of the table's cells will be collected and returned. After all possible data including a table's head and caption are recorded, that table will be saved and the scraper will continue on to the next table and once all valid tables have been found the data will be returned to the PHP API as a JSON string.

This string is returned to the PHP script handling the API request and will be returned to the client as the response to the request. Some pages may have no usable tables or the URL provided is invalid. In these cases PhantomJS will still generate a valid response and it is up to the client to verify that the request did return data. Once the data has been passed on to the client the Parser has completed its job, and from here the Analyzer will begin its work on the client.

Visualization Rendering

```
{
     "Type": ("Line" | "Bar" | "Scatter" | "Pie" | "Tree"  | "Bubble")
     "DataColumns": [independent, (dependent)*],
     "Score": (Number) Bigger is better,
     "VisTitle": (string)
}
```

Figure 5: Visualization Object Definition

```
{
     "Visualizations": Visualization[]
     "Data":
     {
          "ColumnLabel": [col], (Default empty string)
          "Caption": (String), (Default empty string)
          "ColumnType": (Integer | Float | String ),
          "ColumnUnique": [Percents], (0.00 to 1.00)
          "Cols" : (Integer),
          "DataSetScore" : (sum of all ColumnUnique values * Rows * Cols),
          "Rows" : (Integer),
          "Values": [rows][col],
          "MetaData":[row][col] (Default empty string)
     }
}
```

Figure 6: Data Object Definition

The final stage in the Instant Visualization tool pipeline is the Visualizer module. This component is responsible for drawing the visualization to the web page based on the instructions (Figure 5) and data (Figure 6) it received from the Controller, which were originally produced by the Analyzer.

The Visualizer needed to be able to produce a set of six different graph types: bar chart, line graph, scatter plot, bubble chart, pie chart, and a basic treemap. It also needed to be able to present either one or two data sets, so as to allow the user to make comparisons between various parts of the data. Additionally, given the wide range of potential inputs the Instant Visualization tool could receive, the Visualizer needed to be able to draw visualizations regardless of the size of the data set. A final requirement of the Visualizer

was the need to be very fast in order to be able to respond to user customizations and to provide interactive visualizations.

To accommodate all of the requirements of the Visualizer module, the Instant Visualization tool utilizes D3.js. D3.js, which stands for Data-Driven Documents, is an open source JavaScript library created by Mike Bostock, and it provides the ability to bind data to elements of an HTML page's DOM, and then manipulate the elements based on that data. The library makes use of HTML, CSS, and an XML-based vector graphics markup language called SVG (Scalable Vector Graphics). Being a JavaScript library, all of D3's operations are performed in the web browser, which makes it very fast and capable of both handling large data sets and providing interactivity.

```
1  svg.selectAll("rect")
2          .data(rectData)
3          .enter()
4          .append("rect")
5          .attr("class", "treemap-section")
6          .attr("fill", "blue")
7          .style("stroke", "black")
8          .attr("height", function(d) { return d; })
9          .attr("width", 10)
10         .attr("x", function(d, i) { return i*12; })
11         .attr("y", function(d) { return yScale(d); })
12         .style("stroke", "black")
13         .on("mouseover", function() {
14             d3.select(this)
15                 .attr("fill", "orange");
16         })
17         .on("mouseout", function(d, i)
18         {
19             d3.select(this)
20                 .attr("fill", "blue");
21         });
```

Figure 7: D3 Method Chain.

The usual D3 flow consists of a method chain (shown in Figure 7), the initial step of which is the selection of elements in the HTML document (Figure 7, line 1). This selection may be empty if the requested elements do not exist. The data being visualized, which is stored in an array, is then bound to the selection (Figure 7, line 2). Next, placeholders are added for missing elements (Figure 7, line 3), and then SVG elements are created for each data

point in the data set (Figure 7, line 4). Finally, various attributes, including position, dimensions, style, and interaction behaviors are defined (Figure 7, lines 5-21).

Each of the visualization types produced by the Visualizer follow this pattern. Some types handle the preparation of the data prior to this flow differently, however. For example, the Treemap needs to calculate all of the attributes of the rectangles before it adds the SVG rectangle elements to the DOM. The scatter plot data, however, needs no additional manipulation prior to rendering, and is used with the D3 methods as is.

Interactivity is simple to add to specific SVG elements, such as implementing hovering behavior for a point in a scatter plot or a bar in a bar chart. For the line graph, however, the user is able to move a vertical guideline across the visualization. When the line intersects with any data points, tooltips pop up. Since these interactions do not involve direct intersection between the element and the mouse cursor, the standard interaction events won't suffice. Thus, the line graph definition stores information about each point, then compares the mouse position to those points every time the mouse cursor moves. The tooltips for the points within a specified x distance from the line are then displayed. In a case such as this, some features of D3 will not be applicable, and additional work may be required without the use of D3.

Design and Implementation Decisions

D3.js JavaScript Library for Visualizations

One of the main decisions in the design of the Instant Visualization tool was how the tool would produce visualizations. Given that the tool was intended to be web-based, and that it was to produce interactive visualizations, it was logical to pursue a JavaScript-based solution. JavaScript would enable responsive and dynamically-updating visualizations with which the user could interact. For this reason, the tool does not utilize some common statistical and visualization languages such as R. D3.js is a JavaScript library that provides HTML document manipulation functionality and is widely used for data visualization applications. D3 stands for Data-Driven Documents, and it is a powerful library that enables the binding of data to elements of a web page's DOM (Document Object Model). Featuring a smooth learning curve and providing clean visuals that respond to user-input quickly, D3.js offered many benefits with little to no downsides, and thus proved to be a great choice for the Instant Visualization tool's graphical needs.

Tablesorter jQuery Plugin for Data Table Creation

Another requirement was to display the scraped data alongside the graph in a table. While producing a simple HTML table would meet these needs, the tool instead utilizes the Tablesorter jQuery plug-in. By using a plug-in the tool limits the possibilities for naming conflicts with existing code and it provides a simple API to get some powerful results. The tool gains the ability to sort the table dynamically, as well as to improve the display of the table with little effort required by the developers. While there are table manipulation libraries that provide much more power and interaction, these extra features are out of the project scope and provide a poor trade off for programmer time to feature gain. Tablesorter provides a nice balance between simplicity and the quality of results.

jQuery for Table Parsing and DOM Manipulation

In recent years jQuery has become a standard part of web development. Large companies such as Microsoft and Google have elected to use jQuery as a JavaScript library of choice. Boasting a powerful selection of tools, jQuery enables easy manipulation of the DOM, producing a powerful and clean web page. For this project, the Instant Visualization tool utilizes jQuery for several reasons, including dynamically rendering and manipulating data, which was a core requirement of the project. Next, to make the page more visually appealing the tool needed to provide effects such as fading, loading notifiers and the dynamic resizing of content. These tasks are ideal use cases for jQuery. In addition, the tool also used jQuery for parsing tables, since the provided selectors and DOM traversal functions make parsing existing HTML much simpler.

Performing Parsing on the Server

The Instant Visualization tool performs the parsing of HTML data tables on the server with the aid of PhantomJS. Modern websites are heavy users of JavaScript on the client-side for rendering the final HTML page. Some of these pages may have their entire page generated dynamically, including tables, backed by underlying JavaScript data. Traditional scraping would prevent the tool from seeing any part of such tables, and in a real world case, the data would either be in a separate file or possibly loaded from the server through an API/AJAX(asynchronous request for JSON/XML) call. It is because of these factors that the tool must scrape the pages as they are rendered and not the underlying file. With this in mind it would be ideal to scrape the data in the browser, but security policies do not allow pages from different sites to interact without special conditions provided by both servers.

Thus, the tool needs to do the scraping in a headless browser on the server, which can be accomplished via PhantomJS, a JavaScript-based headless WebKit.

Performing Data Analysis and Visualization Rendering in the Client

Other than the Parser module, the Instant Visualization tool performs all of its operations client-side. The primary motivation behind this approach is that it enables a quick and responsive experience for the user. The tool makes use of one request-response pair when it communicates with the server-side Parser, and subsequently has no need to make any further requests. Additionally, by carrying out most operations client-side, the load on the server is reduced, which increases performance in cases where the server is being contacted by a larger number of clients. Finally, the fact that the Visualizer module operates client-side means the tool can provide dynamic, responsive visualizations with which the user can interact.

Modularizing the system architecture

The Instant Visualization tool is designed in a modular fashion. It's component parts (Parser, Analyzer, Controller, Visualizer) are decoupled such that any individual component could be replaced by another implementation. For example, different parsing modules could be used to extract data from different sources, and alternate visualization modules could be implemented that utilize different external graphing libraries. This makes the tool easy to adapt for various applications and simple to update to provide new features.

Hosting copies of external libraries on the server

The Instant Visualization tool's server hosts copies of D3.js, jQuery, Tablesorter, and PhantomJS, rather than linking to the libraries' external locations. This provides the tool with two benefits. First, it prevents linking errors that could occur in the event that connections to the libraries cannot be made. This approach eliminates the need to make those connections and thus eliminates the potential corresponding errors. The second benefit is that it ensures the Instant Visualization tool will continue to work in the future, even if the libraries change or deprecate in some manner that would negatively affect the performance of the tool. With this approach, the tool will always be utilizing the same library code and any errors that could be caused by the changing of that code are avoided.

Results

The Newmont Instant Visualization tool has been tested on Firefox (version 30), Chrome (version 35), QupZilla 1.6.5 (using WebKit 534), and Internet Explorer 11. No differences were found in the tool across the various browsers. All features worked as expected regardless of the client in which it was running.

The tool's aim is to handle any valid and accessible URL, and as such, the potential number of URLs that could be provided to the system is extremely large. Additionally, there are a variety of ways in which an HTML table can be formed. The tool has been given a wide range of URLs with varying table formats as test cases, and it proved to work well within the project requirements.

The goals of the project were successfully met. The tool consists of a front-end web page allowing the user to enter a URL into a textbox. The tool then verifies the provided URL is accessible and, if it is, scrapes the web page located at the URL for tabular data. If useable data is found and scraped, each set of tabular data is analyzed and estimations are made indicating which portions of each data set are most useful for visualization. Based on the analysis, an interactive graph visualizing the data is produced, and additional information is displayed based on the user interaction. Finally, the user is able to select alternate visualizations of the data if they are not satisfied with initial graph. The product is hosted on a machine that is controlled by the client.

Throughout development, the core aspects of the design remained intact. However, items such as the user interface and the level of control the user has over the data changed over time. Such additions are listed below.

- Alongside the visualization, the user is presented with a data table displaying the information scraped from the source URL. The user is able to interact with this table, altering the table title, column titles, and even individual cell values. Additionally, the user can delete entire rows of the table. These changes are reflected in the visualization, which updates as the table is altered.

- The user is also able to change the color pallette of the visualization by choosing a color scheme from a selection of presets displayed on the web page.

- The user interface evolved over the course of development. At first, the interface simply presented data from a single table. It was then refactored to display data from multiple tables. The interface then changed a final time to accommodate the inclusion of the data table alongside the visualization.

Due to the modular nature of the project new components can easily be added such as new visualization types or new parsers for other data sources such as CSV or a database server. Given the scope of URLs and tables the tool could encounter, testing the system is an ongoing process. Other areas for future work include expanding support for older browsers such as ones that are not HTML 5 compatible and allowing the user to save and share the dataset as a whole and not just a static visualization.