Applying Web Technologies to Mine Data for Models of Advancement Over Time in Space Exploration Technology

A Thesis Proposal
for the Degree of
Master of Science in Information Science (MSIS)
University of Arkansas at Little Rock

By
Peng-Hung Tsai
2020

ABSTRACT

The development of web technologies has progressed rapidly in the past few decades. As a result, these technologies have increasingly gained popularity among researchers as an instrument for data analysis and visualization to aid their work. The purpose of this project is to pursue a new method for mining time-based data to perform curve fitting, using web client technology. This requires comparing the advantages and disadvantages of different JavaScript software designs to determine which is best. Based on the results, we emphasize developing a client-side web application and demonstrate its use through fitting Moore's Law to space mission data. The tool can act as a foundation for more advanced work using data from other domains.

1. INTRODUCTION

The term curve fitting describes a method that determines a model that provides the best fit curve to the specific values in a data set. It is widely used in various disciplines, such as economics, social sciences, data mining, etc. There are many software tools that can be used for curve fitting. Among the popular choices, a frequently used tool by researchers has been a spreadsheet program, such as Excel or Google Sheets, due to its simplicity and versatility in data analysis and visualization. Spreadsheets do a good job of presenting data clearly, which makes it easy to identify data quality problems, such as missing or duplicate values. Creating formulas to perform numerical calculations and manipulate data to generate the graphs in a spreadsheet is also very straightforward. However, one drawback of spreadsheets is their inflexibility in enabling us to add functionality we might need for advanced analysis without using plugins or supported programming languages. For example, Excel needs Visual Basic scripting knowledge while Google Sheets needs Google Apps Script in order to use their functionalities to the greatest extent possible. Figuring out how to use these tools can be quite challenging.

For the past 20 years, the evolution of web technologies has given rise to the widespread development of web applications. The growing trend toward client-

side web applications is driving a more responsive and interactive user experience through an immediate content update from within the browser with fewer resources required from the server. It has been greatly increasing the possibility of performing curve fitting directly on the browser.

In this project, we will develop a client-side web application and demonstrate its use through fitting Moore's law to space mission data. This work will not only illustrate the use of a client-side web application in providing an interactive curve fitting tool but also act as a foundation for more advanced work using data from other domains. The proposal is organized in the following sequence. First, we investigate various modern web technologies that might be useful for our web application development. Second, we evaluate each technology using the metrics from ISO/IEC 25010 to determine how it would perform to suit our needs. Third, a brief review of relevant literature on the studies of modeling advancement in space exploration technology will be given. The fourth part will describe how the built tool can contribute to making conducting research more efficient and timely. Lastly, a timeline for completion of the project is provided.

2. REVIEW OF TECHNOLOGIES

The overall research program requires that a set of data records be loaded and analyzed. Furthermore, it is desirable that the user could modify the data records in their own personal copy. It is also important that the system does not depend on maintenance of a special server running unique code, because that would present a permanent resource sink such that if the needed resources were ever interrupted the system would become unavailable. In this, the system is unlike the popular commercial paradigm of cloud computing in which cloud based

systems are maintainable because the services they provide yield a flow of funds that can be used for maintenance. With these considerations in mind we next examine a number of different strategies for storing the data records.

Screenshot

Among different methods to present data, taking a screenshot of the data and graphs in a spreadsheet and placing the images on the Web would take the least time to implement. Despite this, whenever there is an update in data or graphs, new copies of images will have to be uploaded to replace the old ones. Because images only serve the purpose of displaying the content, this method is restricted to presenting data only. Unless the data is also made available for download, users will not have access to the data for data editing or is.

HTML Table

HTML (HyperText Markup Language) can be used to create a table~~-like~~ structure of columns and rows, an HTML table, to display data on the Web. Its basic structure is a table element, which uses the <table> and </table> tags to mark the start and end of the table and contains some number of rows between them. An HTML table can be used to present text, graphs, or sets of data in a tabular format. Basically, anything that could be put into a spreadsheet could go in a table.

We can control how content appears in an HTML table by linking a CSS (Cascading Style Sheets)style sheet to the HTML code. The size and position of the table and the size of each column and row can also be specified. Although HTML tables only allow for data presentation but not data editing or analysis, if

users need data for their analysis, they could simply select and copy the data they want and then paste it into a spreadsheet, or alternatively use Excel to directly extract data from the web page. This is an advantage compared to a screenshot, which does not allow convenient access to the data. Such access is needed, however, to visualize the data with charts and graphs. We will need to use some JavaScript graphing libraries for this, which will be described later.

Alternatively, users could edit the HTML table directly in the HTML file, or edit it using an editor made for editing HTML files. Then they would have to save the HTML file on their local machine for future use. However it would be a burden on the user to have to do this.

HTML Form

An HTML form, defined by the <FORM> and </FORM> tags in a web page, is used to collect data inputted by users and thus, unlike screenshot and HTML Tables, allows for interactivity within the page. The process of creating an HTML Form set up with initial data is similar to the process of creating an HTML Table. However, as HTML only builds a form's structure, a form on its own is not especially useful but often used with HTML Tables and some programming languages, such as JavaScript and PHP, to create a dynamic web page that enables users to input data, which is then sent to a web server for processing or used to add new data on the client-side. A form's appearance can also be controlled by adding some CSS to its HTML code.

A possible disadvantage of this method is that it is not clear if the user would have trouble saving the edited page on their machine for later use. However, a potential advantage of this approach is that the form submit button could be set

up to tell the server about the updates to the form data and the server could then, if it was constructed properly, make a record of the changes that the research team could use to curate and perhaps update the default data in the form in the web page provided by the server.

JSON

JSON, or JavaScript Object Notation, is a language-independent, text-based syntax that is derived from JavaScript for storing and exchanging data. JSON is commonly used as a data interchange format with most programming languages to transport data between browsers and servers. JSON data can be saved in its own file with the .json extension or stored in an HTML file and accessed using the dot notation (.). It can also be retrieved from a server through JavaScript codes into a web application and converted to an HTML table in the browser. The following example shows a simple JSON format containing two name/value pairs:

```
{
  "Make": "Nissan",
  "Price": 10000
}
```

JSON has some advantages. JSON syntax is simple and easy to work with as it includes no more than commas, curly braces, square brackets, and name/value pairs data. Machines also can handle JSON data easily, so if JSON is used on the server side, users can get a fast response from the server. Moreover, the fact that JSON is recognized natively by JavaScript makes it well-suited to being used

in JavaScript-based web applications. However, its flexibility and lack of schemas limit the ability to verify that the data is correct. If the data is of poor quality, the application using the data can fail. Another issue with the JSON approach is the situation where the user wishes to edit and save the data, then reload it later. The JSON data would need to be stored on the user's local machine, for example in the JavaScript localStorage data repository. This is certainly possible, but the stored data is relatively inaccessible to the user except through the web page that it is stored/read within, making it unclear how the user might move the stored data to another machine.

Note that JSON for storing data is compatible with the HTML table and form methods of presenting it in a web page, and so is not an alternative to them, but rather an option to use or not use with them. Similarly, JSON is not a full solution but needs to be used in conjunction with presenting the data using an HTML table or form or perhaps some other presentation method. Thus, JSON would be potentially useful for the research team as a way to edit and view the raw data, perhaps preferable to reading raw HTML or depending on a commercial HTML editor with the necessary functionality. The JSON file would probably be invisible to the user, unless they tried to change machines in which case they would be stuck or need to be sophisticated users with knowledge of browser-specific intricacies (and they still would likely not need to see the actual JSON data).

Google Sheets

Google Sheets is a cloud-based spreadsheet software that has very similar functionalities to Excel. Anyone who has a Google account can access it easily with any Internet browser and work quickly in entering and analyzing data with

little to no programming required. Users can also conduct complicated calculations, such as solving what-if types questions, without redoing the calculations. A distinct feature of Google Sheets is that it enables a user to work on it simultaneously with other users and decide whether they can just view or edit. This ensures that data security is kept in check.

Despite being limited in customizability, its graphing functionality is easy to operate for simple visualizations with various chart types, which can be configured for automatic updates from source data. Making further customizations to the chart can be done with some JavaScript coding and functions.

Sophisticated processing of data could be done in the web page that serves as a front end to the Google sheet, but that processing could also be done directly in the Google sheet. Comparing these two options, many users are likely to be more comfortable seeing the space mission data in a web page than in a Google sheet. One reason is that the web page approach better supports explanatory text and content that places the data in context. Also, the web page approach gives users the ability to perform advanced data simulations or modeling without any knowledge of coding or expertise in using plugin tools.

Open-source software libraries

Tables display tabular data with multiple rows and columns. Nearly all websites use tables to present data. In addition to using HTML Tables, various open-source JavaScript libraries are available to achieve this purpose. However, unlike standard HTML tables that do no more than providing a basic layout of data, many of them provide advanced features for direct data manipulation, like

filtering, sorting, dynamic editing, etc., in a table-like interface. Using these libraries can quickly transform almost any data source to a beautiful interactive table with a short length of JavaScript code.

Since comparing each of them is difficult and not a part of this project, we look at ~~the~~ two libraries, DataTables and Handsontable, that would best fit our project as both are easy to use and enable responsive interaction. They also have detailed documentation on using their functionalities. They are compatible with major JavaScript libraries including jQuery, which is used to make DOM manipulation easier. Best of all, they both support client-side processing, which allows us to implement curve fitting directly in the browser.

DataTables is a very flexible library. It supports inline editing and allows us to easily add filtering, sorting, searching, and pagination to pure HTML tables. Additionally, it offers a variety of extensions to add more advanced interaction controls.

Handsontable has a very intuitive interface because it provides Excel-like table editing. Users can simply double-click on a cell via a mouse to edit its value and also perform all other major Excel tasks, such as copy, remove, merge cells, insert a row or column, etc. With Handsontable, anyone with experience working with Excel or Google Sheets will find similar spreadsheet experience. It is free for non-commercial purposes.

Graphing libraries

Visualizing data via charts or graphs helps people analyze data and understand data more easily. It also enables identifying patterns within data sets that we could not see if simply looking at a table. Our goal is to find the most

appropriate visualization tool for our curve-fitting method development needs.

There are many different JavaScript graphing libraries that can be utilized to create versatile charts and graphs. Some are simpler or faster, while some are more difficult to learn with a steep learning curve. Some are free while others are paid for. It is outside our present scope to compare all of them comprehensively, and sometimes it may be best to complement one with another. For our project, we consider these factors:

1. Is the line chart type available to choose from?
2. Is the library compatible with major JavaScript frameworks such as jQuery?
3. Is the chart responsive and can it be customized?
4. Is the library available free of charge for research use?

The assessment results of these factors lead us to two options, amCharts and Chart.js, both of which would work best for our project. We will see next how they might be integrated with our design.

amCharts is a simple yet flexible graphing library that has great customization functionality. It is compatible with major JavaScript frameworks such as jQuery, Angular, React, etc. In addition, it can work with Handsontable, an Excel-like interface, so that users who are comfortable with using spreadsheets can easily enter and manipulate data for their charts on the web page. It is free for any use as well although a small mark "JS chart by amCharts" will be added onto the graph.

Chart.js is known for its simplicity and user friendliness. It allows users to create responsive, interactive, and customizable charts and graphs in a short

time. It also provides integrations with jQuery, Angular, React, etc. It is a free open-source library.

Server-side vs. client-side web applications

According to Wikipedia, a web application is a client–server computer program that the client (the user) runs in a web browser. It can include scripts that are run on the client's browser or on the web server. Server-side scripts, such as PHP and ASP.NET, are the codes that are executed by servers and used to handle the processing and storage of data, while client-side scripts, such as HTML and JavaScript, are codes that are executed by browsers and used to display data to users. This allows users to interact with a web page. Some examples of web applications include interactive games, online surveys, shopping carts, online banking, etc.

Conventionally in a web application, the user sends a request to the server, and then the server processes the data before sending a response back to the user. In this flow, the majority of the processing of the data is done at the server end. Nowadays, JavaScript frameworks have brought a trend toward client-side web application development, which shifts much processing to the browser. When the flow simply occurs on the browser, the response to the user's action becomes faster as a trip to the server is not needed. Also, resources on the server are much less used. However, because client-side scripts are mostly written in JavaScript along with HTML and CSS, the browser must support or not block the use of these languages in order to run client-side scripts.

Custom client-side applications with JavaScript

The aforementioned investigations of various web technologies have led us to propose a web tool by taking advantage of the strengths of each of them. Because Handsontable has been incorporated by amCharts into its product design and they work together very smoothly, we chose to use both as our table and graphing library respectively for our development work. One potential drawback to note, however, is that if Handsontable or amCharts goes out of business, our application will stop working.

We also use the jQuery library to employ a useful method, getJSON(), that allows us to easily retrieve JSON format data from a Google sheet for storing space mission data. Given that we prefer not to need a special physical server to store our data and to execute code, and the tool that we develop requires a fast response from the browser, we determined to develop a client-side rather than server-side application.

The reasons that we use Google Sheets as the backend to store our data are many and obvious. It is available for free in the cloud and can be easily accessed with any web browser. Users on Excel can find themselves comfortable manipulating data using Google Sheets because of its similarity with Excel and user-friendly interface. Data security can also be assured by limiting access to the data to appropriate people while keeping the data public. If users want to create their own copy or download the data that they can then edit, it will only require several clicks to achieve it. Above all, without any backend programming knowledge, we can pull the data in JSON format from Google Sheets simply using a few JavaScript scripts.

Before we extract the JSON format data from our Google spreadsheet, we need to change the document settings and create the spreadsheet URL by taking

the following steps:

1. Open the Google spreadsheet file, and then click on the “Share” button on the top right corner. Click "Advanced", and then click "Change..." under "Who has access". Select "On - Public on the web", and then click "Save". On the pop-up window of "Sharing settings", make sure "Public on the web - Anyone on the Internet can find and view" is shown under "Who has access", and then click "Done".

2. Go to “File”. Click “Publish to the web”, and then click "Publish".

3. Get the spreadsheet ID (bolded below) of the spreadsheet from the URL in the address bar.

   https://docs.google.com/spreadsheets/d/**1ILB9lUsCnZ7bh3UX9nJ-enyfr-Cqjj543xfXPmdG9Zc**/edit#gid=846789546

4. Replace the spreadsheetID in the following URL string with the ID acquired in the previous step.

   https://spreadsheets.google.com/feeds/worksheets/**spreadsheetID**/public/basic?alt=json

   The resulted URL will look like this:

   https://spreadsheets.google.com/feeds/worksheets/**1ILB9lUsCnZ7bh3UX9nJ-enyfr-Cqjj543xfXPmdG9Zc**/public/basic?alt=json

5. Open a new browser using the new URL above to get the JSON format data, and find the child element “entry” to get the worksheet ID: oe05mr4, as highlighted below.

"entry": [
{
"id": {
"$t": "https://spreadsheets.google.com/feeds/worksheets/1ILB9lUsCnZ7bh3UX9nJ-enyfr-Cqjj543xfXPmdG9Zc/public/basic/**oe05mr4**"
},...

6. Replace spreadsheetID and worksheetID in the following URL string with the spreadsheet ID and worksheet ID respectively.

   https://spreadsheets.google.com/feeds/list/**spreadsheetID**/**worksheetID**/public/values?alt=json

   The resulted URL will look like this:

   https://spreadsheets.google.com/feeds/list/**1ILB9lUsCnZ7bh3UX9nJ-enyfr-Cqjj543xfXPmdG9Zc**/**oe05mr4**/public/values?alt=json

7. Place this URL inside the getJSON() method in our application script as shown below.

   $.getJSON('**https://spreadsheets.google.com/feeds/list/1ILB9lUsCnZ7bh3UX9nJ-enyfr-Cqjj543xfXPmdG9Zc/oe05mr4/public/values?alt=json**', function(data) {}

The specific steps that we took and the corresponding scripts that we used to build our preliminary application are shown in the Appendix.

3. EVALUATION OF METHODS

In this section, we assess each method according to the criteria adapted from ISO/IEC 25010, which is a widespread international quality standard derived from ISO/IEC 9126. Both standards were prepared by the joint technical committee from ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) to evaluate software quality.

The standard defines eight quality attributes, which are further broken into sets of sub-attributes, in its product quality model: functional suitability, performance efficiency, compatibility, maintainability, portability, reliability, security, and usability. Among them, we consider six attributes that are suitable for the evaluation of these methods and specify these six attributes as follows:

1. Functional suitability: The level to which the method meets functional requirements.
2. Performance efficiency: The duration of the response and processing times of the method when performing its functions.
3. Compatibility: The level to which the method can work with other technologies to perform its function.
4. Maintainability: How easily the method can be modified or updated.
5. Reliability: How well the method performs desired functions for a specified period of time.
6. Usability: How effectively and efficiently users can use the method to complete the task.

We use the following indicators to assess each method.

3 – Effective

2 – Moderately Effective

1 – Not Very Effective or Ineffective

The evaluation results are shown in the table below.

	Screenshot	**HTML Table**	**HTML Form**	**JSON**	**Google Sheets**	**Open-Source Software Library**	**Graphing Library**	**Server-Side Web Application**	**Client-Side Web Application**
	Constituents of Web Application Architecture								
Functional suitability	1	1	1	1	3	1	1	3	3
Performance efficiency	1	3	3	3	3	3	3	2	3
Compatibility	2	3	2	2	3	2	2	3	3
Maintainability	1	1	1	1	2	1	1	3	3
Reliability	3	3	3	3	3	3	3	3	3
Usability	1	2	1	1	3	2	1	3	3
Total	**9**	**13**	**11**	**11**	**17**	**12**	**11**	**17**	**18**

4. REVIEW OF LITERATURE

Many technologies have been proved to conform to the general form of Moore's law, which states that the rate of advances in technologies will increase exponentially over time, despite at different rates of change (Farmer & Lafond 2016). Another law, Wright's law (also called the power law or experience curve), which has also been proposed by many researchers, provides comparable

approximations to the pace of technological improvement (Nagy et al. 2013; Magee et al. 2014; Lafond et al. 2018; Berleant et al. 2019). While these laws have served as guides for technology progress in various domains, few studies have demonstrated that space exploration technology follows similar patterns. Berleant et al. (2019), for instance, employed spacecraft lifespan as a proxy for advancement in space exploration technology and found an exponential trend similar to Moore's law and Wright's law. They chose to use a single proxy rather than multiple variables to avoid the potential of overfitting risk. Ultimately an extrapolatable curve with a perfect fitting to the data will have limited value if it has no predictive power (Kodali et al. 2018). Our study will extend their research to include a larger set of variables to pursue the most general space exploration trend curve without overfitting through splitting the data into training and test sets for goodness-of-fit testing.

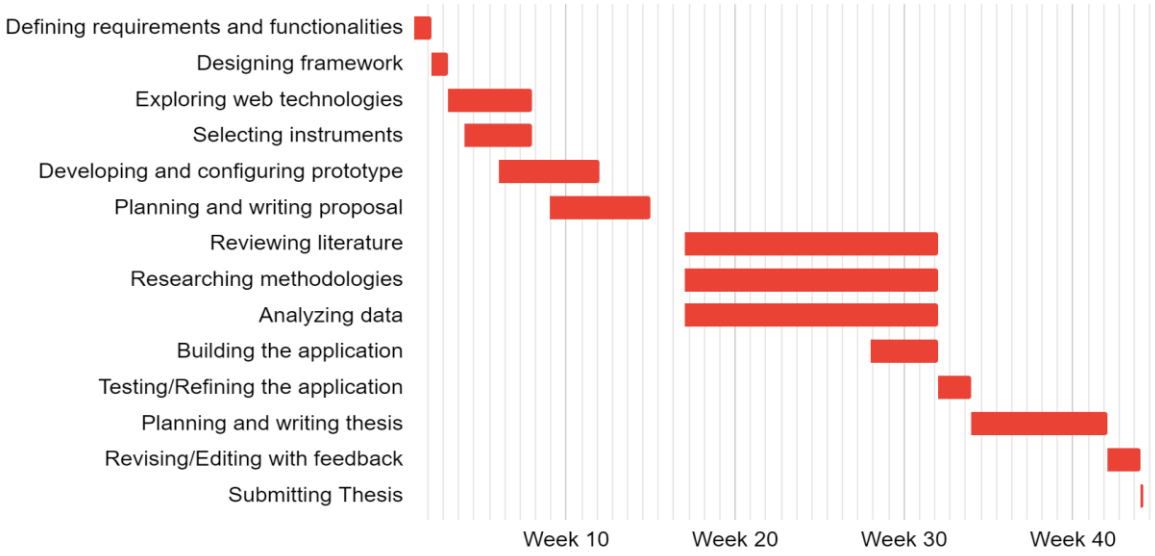
5. INTENDED CONTRIBUTIONS

This project will provide a new approach to curve fitting using a custom client-side web application. The application will allow users with any technical background to easily explore and manipulate data for curve fitting by changing a set of parameters and comparing different fit results graphically and numerically on the same screen in real time. These accessibility and interactive features will greatly improve how users understand and analyze data and ultimately aid research. Moreover, by demonstrating its use through fitting Moore's Law to space mission data, our results will contribute to the literature related to the advancement in space exploration. After all, understanding how technological performance in space exploration improves over time will provide better foresight

in formulating technology policy as well as planning unmanned exploration missions to distant objects (Hall et al. 2017). Although we only focus on using space mission data in this research, the new tool and the methodology for curve fitting could be applied to different data sets from other domains.

6. TIMELINE FOR COMPLETION OF THE PROJECT

Activities	Start Date	End Date	Start on Week
Defining requirements and functionalities	1/21/2020	1/27/2020	1
Designing framework	1/28/2020	2/3/2020	2
Exploring web technologies	2/4/2020	3/9/2020	3
Selecting instruments	2/11/2020	3/9/2020	4
Developing and configuring prototype	2/25/2020	4/6/2020	6
Planning and writing proposal	3/17/2020	4/27/2020	9
Reviewing literature	5/12/2020	8/24/2020	17
Researching methodologies	5/12/2020	8/24/2020	17
Analyzing data	5/12/2020	8/24/2020	17
Building the application	7/28/2020	8/24/2020	28
Testing/Refining the application	8/25/2020	9/7/2020	32
Planning and writing thesis	9/8/2020	11/2/2020	34
Revising/Editing with feedback	11/3/2020	11/16/2020	42
Submitting Thesis	11/17/2020	11/17/2020	44


Defining requirements and functionalities
Designing framework
Exploring web technologies
Selecting instruments
Developing and configuring prototype
Planning and writing proposal
Reviewing literature
Researching methodologies
Analyzing data
Building the application
Testing/Refining the application
Planning and writing thesis
Revising/Editing with feedback
Submitting Thesis
Week 10
Week 20
Week 30
Week 40

BIBLIOGRAPHY

amCharts (2019). *amCharts V4 Tutorial.* Retrieved 2020-03-02 from https://www.amcharts.com/docs/v4/

Berleant, D., Kodali, V., Segall, R., Aboudja, H., & Howell, M. (2019). Moore's law, Wright's law, and the countdown to exponential space. *The Space Review*, 2019, Paper 3632/1.

Chart.js (2019). *Chart.js V2.9.3 Tutorial.* Retrieved 2020-02-24 from https://www.chartjs.org/docs/latest/

DataTables (2020). *DataTables 1.10+ Tutorial.* Retrieved 2020-02-08 from https://datatables.net/manual/

Hall, C., Berleant, D., Segall, R., & Lu, S. (2017). Steps toward quantifying advancement in space exploration. *Proceedings of WMSCI.*

Handsontable (2020). *Handsontable 7.4.2 Tutorial.* Retrieved 2020-03-02, from https://handsontable.com/docs/7.4.2/tutorial-introduction.html

Howell, M., Kodali, V., Kreinovich, V., Aboudja, H., Batthula, V.J.R., Segall, R., & Berleant, D. (2019). Trends in technology development in the US and USSR during the Space Race. *The Space Review*, 2019, Paper 3840/1.

Howell, M., Kodali, V., Segall, R., Aboudja, H., & Berleant, D. (2019). Moore’s Law

and Space Exploration: New Insights and Next Steps. *Journal of the Arkansas Academy of Science*, 73(Article 6):13-17.

International Organization for Standardization. (2011). *_Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — System and software quality models_(ISO Standard No. 25010:2011(en)).* Retrieved 2020-04-10, from https://www.iso.org/obp/ui/#iso:std:iso-iec:25010:ed-1:v1:en

Kodali, V., Duggirala, R., Segall, R., Aboudja, H., & Berleant, D. (2018). Travel to extraterrestrial destinations over time: some exploratory analyses of mission data. *Journal of the Arkansas Academy of Science*, 72(Article 13):58-66.

Lafond, F., Bailey, A.G., Bakker, J.D., Rebois, D., Zadourian, R., McSharry, P., & Farmer, J.D. (2018). How well do experience curves predict technological progress? A method for making distributional forecasts. *Technological Forecasting and Social Change*, 128:104-117.

Magee, C.L., Basnet, S., Funk, J.L., & Benson, C.L. (2014). Quantitative empirical trends in technical performance. *Technological Forecasting and Social Change,* 104:237-246.

Nagy, B., Farmer, J.D., Bui, Q.M., & Trancik, J.E. (2013). Statistical basis for predicting technological progress. *PLoS One*, 8(2):e52669.

Rohit Boggarapu, R. (2016). *Interactive JavaScript Charts Using Data from Google Sheets.* Retrieved 2020-02-01, from https://www.sitepoint.com/interactive-javascript-charts-using-data-from-google-sheets/

W3Schools (2020). *CSS Tutorial.* Retrieved 2020-02-12, from https://www.w3schools.com/css/default.asp

W3Schools (2020). *HTML Tutorial.* Retrieved 2020-02-05, from https://www.w3schools.com/html/default.asp

W3Schools (2020). *JavaScript Tutorial.* Retrieved 2020-02-08, from https://www.w3schools.com/js/default.asp

W3Schools (2020). *JS JSON Tutorial.* Retrieved 2020-02-07, from https://www.w3schools.com/js/js_json_intro.asp

W3Techs (2020). *Usage statistics of client-side programming languages for websites.* Retrieved 2020-02-04, from https://w3techs.com/technologies/overview/client_side_language

Wikipedia (2020). *Comparison of JavaScript charting libraries.* Retrieved 2020-02-24, from https://en.wikipedia.org/wiki/Comparison_of_JavaScript_charting_libraries

Wikipedia (2020). *ISO/IEC 9126.* Retrieved 2020-04-10, from

https://en.wikipedia.org/wiki/ISO/IEC_9126

Wikipedia (2020). *jQuery.* Retrieved 2020-02-22, from https://en.wikipedia.org/wiki/JQuery#cite_note-:0-4

Wikipedia (2020). *Web application.* Retrieved 2020-02-22, from https://en.wikipedia.org/wiki/Web_application

APPENDIX

```
// HTML code

<!doctype html>
<html lang="en-US">

<head>
<meta charset="utf-8">
<title>My Project</title>

// Import the Handsontable CSS files.

<link rel="stylesheet" type="text/css"
href="https://cdn.jsdelivr.net/npm/handsontable@latest/dist/handsontable.full.min.
css">
<link rel="stylesheet" type="text/css"
href="https://handsontable.com/static/css/main.css">

// Use CSS for styling the web page.

<style>
@import url(https://fonts.googleapis.com/css?family=Lato);

body {
  font-family: Lato;
}

#chartdiv1 {
  width: 100%;
  height: 300px;
  border: 1px solid #ccc;
```

```
}

#chartdiv2 {
  width: 100%;
  height: 300px;
  border: 1px solid #ccc;
}
</style>
</head>

<body>
<div>
    <h4>Used Car Listings</h4>

    // Create a button linking to the Google Sheets data file that will open in another web page after clicking it.

    <a class="button butn"
    href="https://docs.google.com/spreadsheets/d/1ILB9lUsCnZ7bh3UX9nJ-enyfr-Cqjj543xfXPmdG9Zc/edit#gid=846789546" target="_blank">
    Click here to see the source data
    </a>
</div>

// Create a <div> container where the Handsontable table is rendered.

<div id="hTable"></div>

<div id="export-buttons" class="visible">

    // Create a button element for downloading the table in CSV format.
```

```
    <button id="export-csv" class="btn size-medium bg-blue text-white shadow hover-moveup" style="float: center; margin-top: 15px;">
            Export to a .csv file
    </button>
</div>

<br></br>

// Create a <div> where the chart one is rendered.

<div id="chartdiv1"></div>

<br></br>

// Create a <div> where the chart two is rendered.

<div id="chartdiv2"></div>

// Include the jQuery, Handsontable, and amCharts libraries subsequently via CDNs:

<!-- jQuery -->
<script src="https://code.jquery.com/jquery-3.4.1.min.js"></script>

<!-- Handsontable -->
<script src="https://cdn.jsdelivr.net/npm/handsontable@latest/dist/handsontable.full.min.js"></script>

<!-- amCharts -->
<script src="https://www.amcharts.com/lib/3/amcharts.js"></script>
```

```
<script src="https://www.amcharts.com/lib/3/serial.js"></script>

<script src="https://www.amcharts.com/lib/3/themes/light.js"></script>

<script
src="https://www.amcharts.com/lib/3/plugins/dataloader/dataloader.min.js"></scri
pt>

<script
src="https://www.amcharts.com/lib/3/plugins/tools/bestFitLine/bestFitLine.min.js"
></script>


<script>


// Build a function for creating an interactive chart using the
AmCharts.makeChart() method.

// Reference: https://docs.amcharts.com/3/javascriptcharts/AmSerialChart


function createChart1(cardata) {

    return AmCharts.makeChart("chartdiv1", {

            "type": "serial",

            "theme": "light",

            "dataProvider": cardata,

            "marginTop": 10,

            "valueAxes": [{

                    "gridAlpha": 0.07,

                    "position": "left",

                    "title": "Price"

            }],

            "graphs": [{

                    "lineThickness": 1,

                    "bullet": "round",

                    "title": "Price",

                    "valueField": "price"

            }],
```

```
            "chartCursor": {},
            "categoryField": "brand",
            "categoryAxis": {
                "startOnAxis": true
            },
            "legend": {
                "position": "top",
            },
    });
}

// Build another function for creating the second amCharts chart.

function createChart2(cardata) {
    return AmCharts.makeChart("chartdiv2", {
            "type": "serial",
            "theme": "light",
            "dataProvider": cardata,
            "marginTop": 10,
            "valueAxes": [{
                "gridAlpha": 0.07,
                "position": "left",
                "title": "Year"
            }],
            "graphs": [{
                "lineThickness": 1,
                "bullet": "diamond",
                "title": "Year",
                "valueField": "year"
            }],
            "chartCursor": {},
```

```
            "categoryField": "brand",
            "categoryAxis": {
                "startOnAxis": true
            },
            "legend": {
                "position": "top",
            },
    });
}
```

// Fetch the Google Sheets data in JSON format using the getJSON() method with the spreadsheet URL for creating the Handsontable table and amCharts charts.

```
$.getJSON('https://spreadsheets.google.com/feeds/list/1ILB9lUsCnZ7bh3UX9nJ-enyfr-Cqjj543xfXPmdG9Zc/oe05mr4/public/values?alt=json', function(data) {

    // Store the data in a new variable sheetData.

    var sheetData = data.feed.entry;

    // Iterate over each data instance in sheetData and extract each value of the instances out of it. Then store the extracted values in an array: arry.

    var arry = [];
    for (var i = 0; i < sheetData.length; i++) {
        var objson = [
                sheetData[i]['gsx$no']['$t'],
                sheetData[i]['gsx$brand']['$t'],
                sheetData[i]['gsx$year']['$t'],
                sheetData[i]['gsx$price']['$t']
            ];
```

```
            arry.push(objson);
    }

    var dataCar = arry, // Assign arry to dataCar, which will be used by Handsontable and amCharts as a data source.
            chart1,
            chart2;

    // Start Handsontable.
    // Get the <div id="hTable"></div> element and assign it to tableElement, which will be passed as a first argument to the Handsontable constructor.

    var tableElement = document.querySelector('#hTable');

    // Set appropriate values for the constructor options and assign them to tableSetting, which will be passed as a second argument to the Handsontable constructor.
    // Reference: https://handsontable.com/docs/7.4.2/Options.html

    var tableSetting = {
                data: dataCar,
                height: 200,
                maxRows: 600,
                stretchH: 'all',
                width: 650,
                autoColumnSize: {
                        samplingRatio: 23
                },
                autoWrapRow: true,
                colHeaders: [
                        'No',
                        'Brand',
```

```
            'Year',
            'Price'
        ],
        columnSorting: {
            indicator: true
        },
        contextMenu: true,
        dropdownMenu: true,
        filters: true,
        formulas: true,
        licenseKey: 'non-commercial-and-evaluation',
        manualColumnMove: true,
        manualColumnResize: true,
        manualRowMove: true,
        manualRowResize: true,
        mergeCells: true,
        minSpareRows: 0,
        multiColumnSorting: {
            indicator: true
        },

        // Add the afterInit hook to create the amCharts chart when the Handsontable is initiated.
        // Reference: https://handsontable.com/docs/7.4.2/Hooks.html#event:afterInit

        "afterInit": function(firstTime) {
            chart1 = createChart1(
                formatChartData(this.getData())
            );
            chart2 = createChart2(
```

```
                    formatChartData(this.getData())
                );
            },

            // Add the afterChange hook to track each change in Handsontable and update the charts.
            // Reference: https://handsontable.com/docs/7.4.2/Hooks.html#event:afterChange

            "afterChange": function(changes, source) {
                if (changes === null)
                    return;
                chart1.dataProvider = formatChartData(this.getData());
    // Update the first chart with the formatChartData() function when one or more cells have been changed.
                chart1.validateData();
                chart2.dataProvider = formatChartData(this.getData());
    // Update the second chart.
                chart2.validateData();
            },
    };

    // Construct a Handsontable table using both tableElement and tableSetting.

    var myTable = new Handsontable(tableElement, tableSetting);

    // Get the <button id="export-csv" ...></button> element and assign it to button,

    var button = document.getElementById("export-csv");

    // Access to the exportFile plugin using the getPlugin method
    // Reference: https://handsontable.com/docs/7.4.2/ExportFile.html
```

```
    var exportPlugin = myTable.getPlugin("exportFile");


    // Apply the addEventListener method on button to build a function for
downloading the table.
    // Reference: https://handsontable.com/docs/7.4.2/EventManager.html


    button.addEventListener("click", function(event) {
        exportPlugin.downloadFile("csv", {
            bom: false,
            columnDelimiter: ',',
            columnHeaders: true,
            rowDelimiter: '\r\n',
            rowHeaders: false,
            exportHiddenColumns: true,
            exportHiddenRows: true,
            fileExtension: 'csv',
            filename: 'Table_[YYYY]-[MM]-[DD]',
            mimeType: 'text/csv'
        });
    });


    // Create a function to get the data from Handsontable for construcing or
updating the amCharts charts.


    function formatChartData(tableData) {
        var chartData = [];
        for(var i = 0; i < tableData.length; i++) {
            chartData.push({
                "no":           tableData[i][0],
                "brand":        tableData[i][1],
```

```
                    "year":         tableData[i][2],
                    "price":        tableData[i][3]
                });
            }
            return chartData;
    }
    })

</script>

</body>
</html>
```