

Investigating Data Parsing and Visualization for Enhanced Insights

Marlon Carney Crowe

N00212147

***Report submitted in partial fulfilment of the requirements for the BSc (Hons) in
Creative Computing at the Institute of Art, Design and Technology (IADT).***

Declaration of Authorship

The incorporation of material without formal and proper acknowledgement (even with no deliberate intent to cheat) can constitute plagiarism.

If you have received significant help with a solution from one or more colleagues, you should document this in your submitted work and if you have any doubt as to what level of discussion/collaboration is acceptable, you should consult your lecturer or the Programme Chair.

WARNING: Take care when discarding program listings lest they be copied by someone else, which may well bring you under suspicion. Do not leave copies of your own files on a hard disk where they can be accessed by others. Be aware that removable media, used to transfer work, may also be removed and/or copied by others if left unattended.

Plagiarism is considered to be an act of fraudulence and an offence against Institute discipline.

Alleged plagiarism will be investigated and dealt with appropriately by the Institute. Please refer to the Institute Handbook for further details of penalties.

The following is an extract from the B.Sc. in Computing (Hons) course handbook. Please read carefully and sign the declaration below

Collusion may be defined as more than one person working on an individual assessment. This would include jointly developed solutions as well as one individual giving a solution to another who then makes some changes and hands it up as their own work.

Declaration

I am aware of the Institute's policy on plagiarism and certify that this thesis is my own work.

Signed: Marlon Carney Crowe

Date: 29/03/2025

Failure to complete and submit this form may lead to an investigation into your work.

Abstract

Application Programming Interfaces (APIs) play a highly important role in modern software development, APIs serve as a way to easily pull and send information from anywhere in the world. Data visualization takes advantage of the power of APIs, in the sense that as data sets get larger, the benefits of data visualization become clearer, providing insights into the data and making it more accessible to a broader audience. This project demonstrates how users can supply an API URL endpoint and receive a response in the form of the requested data being displayed on screen. Furthermore, this document shows how users can filter and export the raw response data from the API and create visualizations and dashboards from the collected data.

Acknowledgements

I would like to thank the supervisor of this project, Cyril Conolly for providing invaluable guidance and feedback during this project. Cyril's knowledge of data visualization and expertise in Tableau and R helped greatly in the development of this project.

I want to express my gratitude to the individuals and organizations who host free APIs, their generosity made this project possible.

I would also like to extend a thank you to the R community for providing and maintaining free packages which greatly enhance visualization.

Contents

Abstract	3
Acknowledgements	4
Contents	5
1. Introduction	7
2. Research.....	9
2.1 Application Programming Interface	9
2.1.1 What is an application programming interface?	9
2.1.2 How to use an API	9
2.1.3 Filtering data from APIs	10
2.1.4 Why API architecture design is important	11
2.2 Data visualization.....	15
2.2.1 The need for visualization because of Big Data.....	15
2.2.2 Why data visualization is important.....	16
2.2.3 The issues of data visualization	16
2.2.4 Business Intelligence	18
3. Requirements	20
3.1 Similar applications	21
4. Design.....	23
5. Implementation.....	26
5.1 Creating the Front-end	26
5.2 Visualizing and Developing in Tableau.....	34

5.3 Visualizing and developing in R.....	41
6. Testing and Analysis	51
6.1 Functional Testing	51
6.2 User Testing.....	52
6.3 Insights and learnings	53
7. Project Management.....	55
7.1 Project Management Tools	55
7.1.1 GitHub.....	55
7.1.2 Miro	55
7.1.3 Figma	55
7.2 Communication.....	56
8. Conclusion.....	57
References.....	60

1. Introduction

The project aims to create a flexible platform that allows unsupported third-party APIs to integrate seamlessly into software that supports data visualization e.g., using proprietary software such as Tableau and an open-source environment such as R.

APIs operate on four fundamental functions, “GET”, which retrieves data, “POST”, which sends data, “PUT”, which updates existing data and “DELETE”, which deletes data.

Traditional dashboards are built with a specific data type in mind, which limits their flexibility. This project aims to solve this problem by providing functionality to allow the user to filter out any unwanted data from the API of their choice and choose what to display from within the application.

Since the 1970s, business intelligence dashboards have played a critical role in assisting businesses make informed decisions. Initially, this technology was designed for business analysts with specialized knowledge in data analysis. However, the rise of data has brought the evolution of modern dashboards that are becoming increasingly more user-friendly and can empower not just businesses but also day-to-day people.

Development in the field of informational dashboards is the use of artificial intelligence and machine learning to provide more advanced, predictive insights and actions based on the data. Applications such as Grafana and Kibana are offering real-time dashboards for monitoring systems for industries such as finance and healthcare.

The major technologies that made this project possible were React, for the front-end which handles retrieving data from APIs and Tableau and R for data visualization. A number of challenges were identified and solved in the development of this project such as parsing and filtering the data from APIs, exporting the data correctly without the loss of information and learning the software Tableau and R.

Testing methods such as functional testing were carried out to see how the application was functioning and to see if it was operating as expected. Things were highlighted during this process of testing such as error handling and communication to the user, in this area, the application was lacking.

We believe that too much time was spent developing areas of the project, such as the data filtration system. If less time was spent there, more emphasis could have been placed on creating interactive visualization in the R environment. Allowing us to broaden our understanding and skills that R offers with it. As the project developed, the initial target audience evolved. Initially our target audience were tech-savvy people aged 20 and above, but to really get the most out of this application, you need knowledge of Tableau and R to create visualization from the data, so the target audience shifted towards developers.

Despite the discussed challenges, the project did achieve its goal by revising initial goals and narrowing the scope of the overall project.

2. Research

2.1 Application Programming Interface

2.1.1 What is an application programming interface?

Santoro, M., Vaccari, L., Mavridis, D., Smith, R., Posada, M., & Gattwinkel, D. (2019), An application programming interface (API) is a set of rules and protocols which enables different software applications to communicate with each other. It defines methods and data formats that applications can use to request and exchange data. APIs serve as the middleman that allow the integration of different systems to be able to work together. Cooksey, B. (2014), APIs are designed in mind to make the process of sharing data and services much easier, allowing for developers to build applications that take advantage of these platforms. As a result of this, APIs have become a fundamental piece of technology to modern software development, and it is used in everything from mobile applications to complex systems for businesses, for example, Netflix, Amazon, Spotify and Facebook all use APIs to enhance their services.

2.1.2 How to use an API

Hall, D. (2024, August 1), To begin using an API you first need an endpoint. Endpoints are URLs that provide access to certain information on a server, these endpoints are usually provided by the developer. Once you have your endpoint, you give it to your API client and hit send.



Figure 1 Example of a GET request

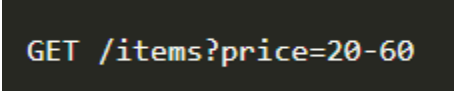
The four main request methods of an API client are GET, POST, PUT and DELETE. The GET method retrieves information, the POST method creates new data, the PUT

method updates a current piece of data, and the DELETE method deletes data. The request methods POST, PUT and DELETE will likely require authentication to be used in most cases.

2.1.3 Filtering data from APIs

Parameswaran, V. (2023, April 25), to filter the data from an API, it refers to the process of limiting what information is sent back to your client based on the criteria provided by the user.

For example, if you wanted to only retrieve data about items that were in the price range from 20-60, the request may look like this:

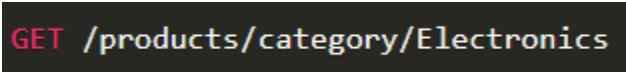


```
GET /items?price=20-60
```

Figure 2 Example of query parameters on a GET request

In an API, parameters can be used to filter data by adding the filtering condition to the endpoint URL.

For example, an endpoint to retrieve products by the category type of “Electronic”, the API endpoint would look like this:



```
GET /products/category/Electronics
```

Figure 3 Example of query parameters on a GET request

In figure 3, the API would send back a response with a list of products of category “Electronic”.

As a developer it is important to know how to use an API client effectively, filtering helps limit the amount of data retrieved, returns only relevant information which allows for faster processing speeds due to the smaller amount of data and improves readability.

2.1.4 Why API architecture design is important

a. What is API architecture?

“API architecture refers to the design and structure of Application Programming Interfaces, specifying how different API components should interact and communicate. The API architecture consists of rules that determine functionalities to provide to the API consumers.” (Catchpoint, 2024).

In API architecture there are four important layers to consider, the interaction layer, application layer, integration layer and the data layer.

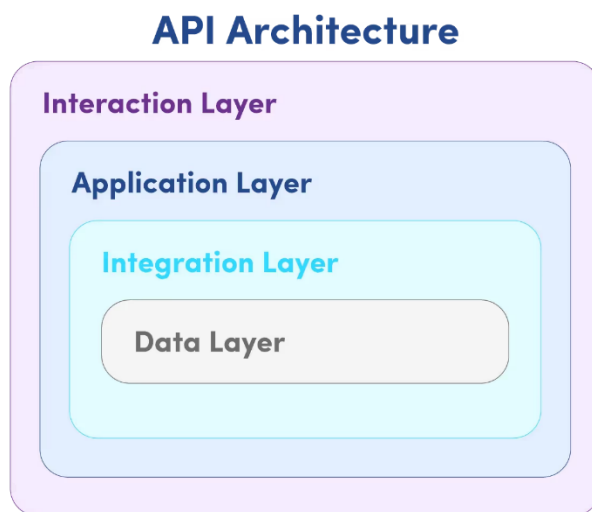


Figure 4 Architecture layers of an API

According to Catchpoint, 2024, the data layer controls the storing, retrieving and manipulation of data. It usually includes databases, data storage, and data components. The main objective of the data layer is to ensure that the data is stored and retrieved quickly.

Catchpoint, 2024, the integration layer increases interoperability by managing the integration of multiple systems and services. The integration layer operates between the data layer and the application layer, providing communication and coordination of data. It also ensures tasks such as data transformation, validation and keeping a steady flow of information between different components.

According to Astera, 2024, the core of the API architecture is the application layer. This layer deals with computing and processing the data and logic, it controls the functionality and behavior of the API and application. For example, components such as algorithms, microservices and business logic are in this layer.

Akana, 2019, the interaction layer is where the application is where the customers, partners and employees interact with the services and data. Astera, 2024, this layer establishes a consistent way of exposing and consuming APIs, regardless of the type of platform and technology it may be using. This reduces complexity and makes collaboration easier across different applications.

b. Key considerations when designing an API

To build an efficient and effective API, there are a few considerations to consider when designing an API:

1. Versioning

According to Catchpoint, (2024), implementing versioning allows for backwards compatibility and smooth transitions when making changes to the API. It allows applications that use older versions of the API to still operate and at the same time, allows the API to evolve without disrupting current users, this creates a reliable development environment.

2. Documentation and Discoverability

(Murphy, L., Kery, M. B., Alliyu, O., Macvean, A., & Myers, B. A. 2018), explain how the first impression is important for customers looking to utilize an API service. Clear documentation on how to interact with the service will go a long way to gaining users and retaining existing users. Poor documentation will deter potential customers.

3. Consistency

(Kotstein, n.d.), discusses that consistency in API design such as naming conventions, data formats, and response messages help users understand how to interact with the API.

4. Rate Limiting

According to Kotstein, limiting the rate at which users can send and receive data to the API is an important factor to consider, especially if the service is free to use. For example, a limit rate of 100 requests per day, this helps ensure that the performance of the service is kept stable and maintains availability of the service.

v. Types of APIs

Stoplight. (n.d.), public APIs are available for anyone to use with little restriction, they may require you to login for authentication and require you to provide a key for the API. An API key is a code used to identify the user or application. Public APIs are usually easy to access because they are intended to be used by the public, thus they are designed in an intuitive manner. Due to public APIs being free to use, often a rate-limit is imposed on the user, limiting the number of requests they can send to the API daily. An example of a public API is Google Maps API, it allows you to create maps to show any location, add customs graphics, interactive custom data layers and much more.

This article also found that many public APIs follow the OpenAPI standard. Previously known as Swagger, the OpenAPI standard is like a public API but follows specifications for writing a public API, using guidelines for things such as endpoint naming conventions, data formats, and error handling. The goal of the guidelines is to streamline the process of getting started and working with an API as a developer, by automating some tasks and removing the need to read through a complex code base.

API Micro Official. (2023, February 17), private APIs are intended to be used within an organization, they are not designed to be used externally because they are made in a way that meets the organization's specific needs. Private APIs can be used to communicate between different departments or to grant access to develop applications within an organization. Amazon S3 API is an example of a private API, it is used by Amazon Web Services to provide internal storage and retrieval of data.

API Micro Official also mentioned that partner APIs are another type of APIs which is used, it is designed for a particular group of developers, such as business partners or customers. Partner APIs are usually used to provide access to data or services that are not consumed by the public. For example, a shipping company provides a partner API that allows third parties to track shipments in real-time. Then an e-commerce company integrates this API into their application and allows customers to keep track of their orders.

vi. The impact of APIs on the industry

According to Benzell, S., Lagarda, G., & Van Alstyne, M. W. (2017), Since the adoption of APIs, firms have experienced a decrease in costs by an estimate of \$420 million annually, this indicates APIs can streamline operations and decrease inefficiencies.

This paper also found that adopting an API into a business can increase a firm's market capitalization by 12.7%, indicating that the market perceives API adoption as a strong indicator of future growth and profitability.

Wulf, J., & Blohm, I. (2020) found that incorporating an API into your service enables faster development and deployment of services, allowing companies to quickly respond to the market and customers' needs. This kind of flexibility is critical in the digital world.

Wulf and Blohm also discussed how being able to communicate between different applications easily allows organizations to further specialize their own applications by utilizing these APIs provided by other services. This specialization can lead to increased quality of the application and gaining a new audience.

APIs allow for greater productivity by allowing programmers to work more efficiently, it enables modular coding, which is the practice of dividing code into smaller, independent easy to read components that perform certain tasks. It makes it easier to access data and software, allows for greater reusability, and integration.

This modularity leads to long-term productivity gains for firms as they keep on reducing internal inefficiencies and become more innovative.

2.2 Data visualization

2.2.1 The need for visualization because of Big Data

a. What is Big Data

According to Pence, H. E. (2014), when the name “Big Data” is mentioned, it refers to the massive amounts of data being generated at any moment from many sources, such as social media, transactions, sensors and much more. The term describes data that is too large or complex to be processed sufficiently by traditional methods.

In Big Data, there are some key characteristics, these are often referred to as the “Vs in Big Data”. According to Sas (N.D), volume is a key characteristic of Big Data. Volume refers to all the data collected by organizations from various sources, such as images, videos and audio. Before, storing all this data would be far too expensive, but nowadays the cost of storage has become cheaper, and there are more efficient ways of storing vast amounts of data such as data lakes.

Sas also determined that velocity is another key metric, with the growth of the internet, data is pouring into businesses faster than ever. This velocity of data must be handled in a timely fashion and efficiently. RFID tags, which are small electronic devices that store information and communicate with one another, sensors and smart meters are the driving forces behind dealing with this vast amount of data in real-time.

Chen, M. (2024, March 11) also discusses that veracity is another key metric to be aware of, veracity is how truthful the data is, and can you rely on it? The basis of veracity is bound to other functional concepts, such as the quality of the data and the integrity. These all combine and become the responsibility of an organization that wishes to be a reliable source of high-quality, accurate and reliable data.

Value is another characteristic Chen, M. discusses; data has an innate value to a business. Until that data is analyzed, and key correlations are identified, it has no value. This value can be internal, such as how tasks within an organization may be optimized, or external, such as altering the homepage of a website to maximize customer engagement.

b. Big Data visualization

Salesforce (2020) mentions that because of the large quantity of information being captured, it is too much for the average person to parse that information effectively. Big data visualization captures the long process of analyzing the data and allows for users to easily comprehend the data.

Sigma Solve (2023, August 24) discusses how big data visualization leads to faster decision making. Due to the analytical process being accelerated, you can use the data sooner, this is a big advantage for a business, because it will allow them to operate quicker than their competitors.

2.2.2 Why data visualization is important

You can use data in many ways and interpret meaningful things, visualizing the data enhances communication and gives you knowledge of something you may not have interpreted through raw data, leading to new insights and developments.

Data visualization also leads to patterns, it is a skill to recognize these patterns and derive meaningful correlations from these patterns. By seeing data plotted, it can reinforce data driven decision making and reach a larger audience of people because the visualization allows a broader range of people to be able to understand it.

2.2.3 The issues of data visualization

According to Stobierski, T., using the wrong type of chart to display your data can lead to interpreting the data in the wrong way. The data type dictates the format of the visualization. The two most important factors are whether the data is qualitative, meaning the data describes or categorizes something or quantitative,

meaning it can be measured. These two factors tend to have preference to certain types of graphs, for example qualitative data is suitable for bar graphs and quantitative is better represented using histograms.

Stobierski also found that adding too many variables to your visualization is harmful. You should consider what variables will help you communicate the data effectively and determine any variables that are unnecessary to visualize.

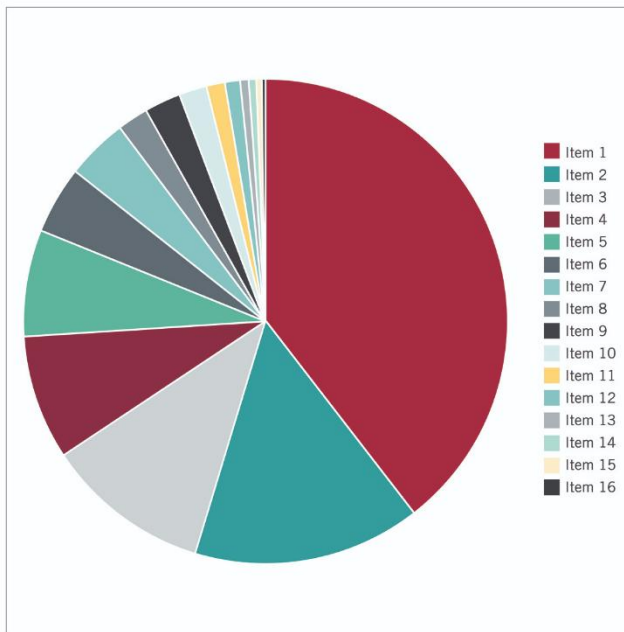


Figure 5 Pie chart with too many variables

As you can see in figure 5, by having too many variables in the pie chart, it makes the difference between values hard to grasp.

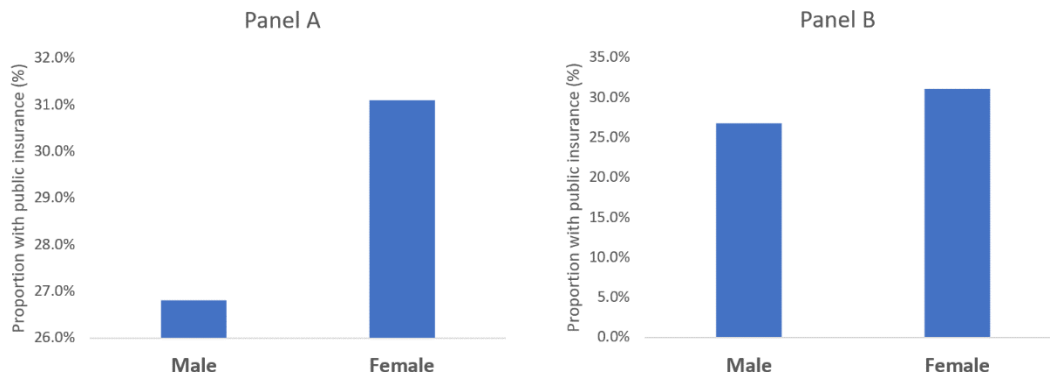


Figure 6 Showcasing the deception of truncating the y-axis

Yau, N. (2017), In figure 6, at first glance in Panel A, it appears that females have a much higher percentage of having public insurance. In reality, if you take a closer look, the difference is only a few percent. This is the effect of truncating data on the y-axis. This means only starting the visualization of data at a minimum value. Due to the percentage of males being near the minimum value, it displays it in a manner which at a first look their value is far lower than the females. In Panel B the y-axis is not truncated, so the value starts at zero, giving a much better depiction of the data.

2.2.4 Business Intelligence

a. What is business intelligence

Elena, C. (2011) claims that business intelligence (BI) refers to the process of using computer-based technology to parse raw data into meaningful information. This information is used to deploy more efficient strategic, insightful decision-making.

Khan, R. A., & Quadri, S. M. (2012) also found that BI is used to improve the quality and time efficiency of information, allowing managers to have a clearer understanding of their business' position relative to their competitors.

b. Business intelligence dashboards

Orlovskiy, D., & Kopp, A. (2020, December), The primary goal of business intelligence dashboards is to provide users with a compact view of key performance indicators (KPIs) and relevant data metrics that encourage data-driven decision-making. Hansoti, B. (2010), They gather and display data from different sources and display it on a singular interface, allowing users to quickly track performance, analyze patterns, and make informed decisions based on these metrics. Dashboards usually provide visualization in the forms graphs and charts to help users understand complicated data quickly.

The primary goal of a BI dashboard is to encourage data driven decision making by presenting data in a digestible format. They help organizations track performance against goals they have set and identify areas that are inefficient. According to Morris, A. (2021, April 16), companies such as Coca-Cola, Tesla, Uber, Walmart, Twitter and many more all take advantage of business intelligence.

3. Requirements

This chapter will outline the key functional and non-functional requirements for the application, identifying any essential features. We also have researched and observed similar applications and will point out the positives and negatives of the applications.

Functional:

API Data Retrieval:

- Allow users to enter an API endpoint URL
- The application can send GET requests to the specificized endpoint
- The application will display the API data back to the user in a readable format

Data filtering:

- The application will allow users to filter for specified data from the response

Exporting data:

- The application will have a function that allows you to export the raw API data to an XLS file.

Non-functional:

- The application will provide a clean and intuitive UI to make it easy to understand how to use the service.
- Developing skills in software such as Tableau and R to visualize data
- Research potential competitors and draw the pros and cons of their services
- To use the application a user needs access to the internet

3.1 Similar applications

Grafana

“Grafana is an open-source solution for running data analytics with the help of metrics that give us an insight into the complex infrastructure and massive amount of data that our services deal with, with the help of customizable dashboards” (Shivang, 2024).



Figure 7 Dashboard in Grafana

Pros:

- Customizable and supports different data sources
- Has real-time visualization and monitoring of data
- APIs can be integrated through HTTP or JSON plugins
- The software is open-source so there are many useful plugins created by the community

Cons:

- Requires technical knowledge and is not straight-forward to setup
- Better suited for data monitoring rather than visualization
- Integrating the data of APIs is complicated

Redash

“Redash is an open-source data collaboration platform that enables you to connect to any data source, visualize data and share it” (Milhomem, 2021).

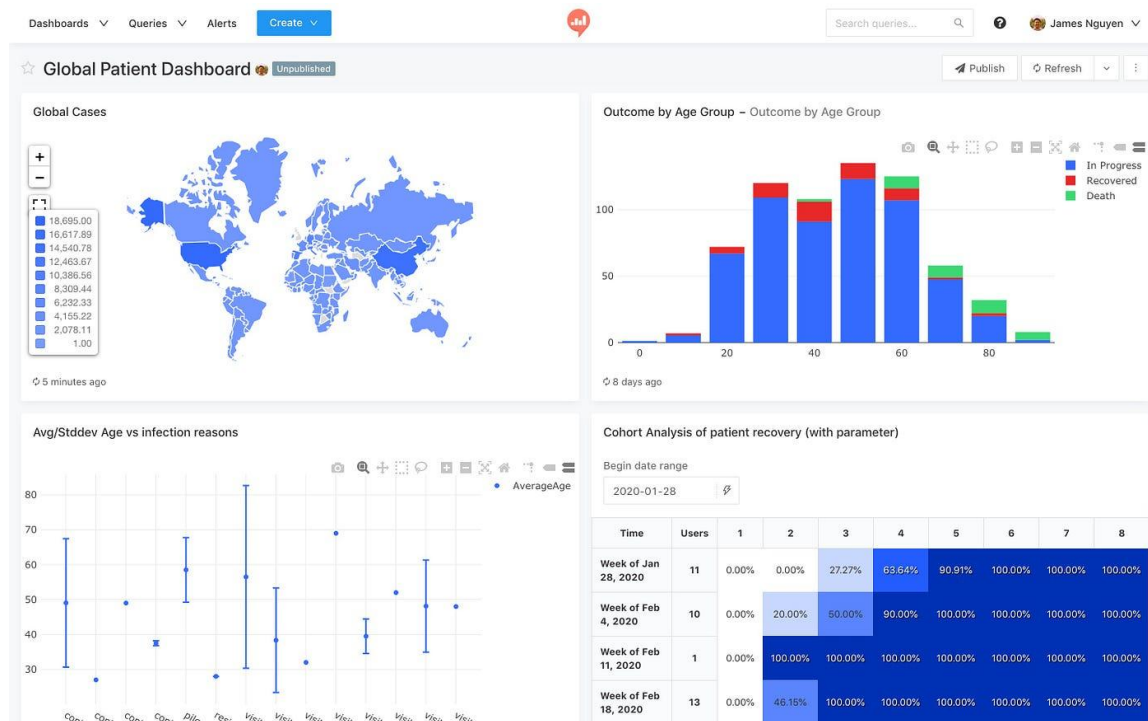


Figure 8 Dashboard in Redash

Pros:

- Open-source allowing for plugin creation by the community
- Supports data from APIs and allows user to input custom queries
- Allows for collaboration

Cons:

- Requires knowledge of SQL to manipulate data
- Installation and maintenance can be troublesome

4. Design

In this chapter, we discuss the design process of the website and mention principles followed during the creation of visualizations.

To envision the look and feel of the website, a prototype wireframe was created in the program Figma before development of the frontend started. “Figma is a collaborative web application for interface design, with additional offline features enabled by desktop applications for macOS and Windows” (Wikipedia 2025). There are benefits to creating a wireframe before developing and designing the website, it helps establish a clear structure, layout and flow of the website. You can indirectly control how the user will interact with the website by how the layout is structured. Having a rough wireframe can also help the transfer from low-fidelity to high-fidelity design of the website, which is exactly what happened in the development process of the website.

These wireframes in Figma also make it extremely easy to test different color palettes and alternate layouts, making the design process more efficient.

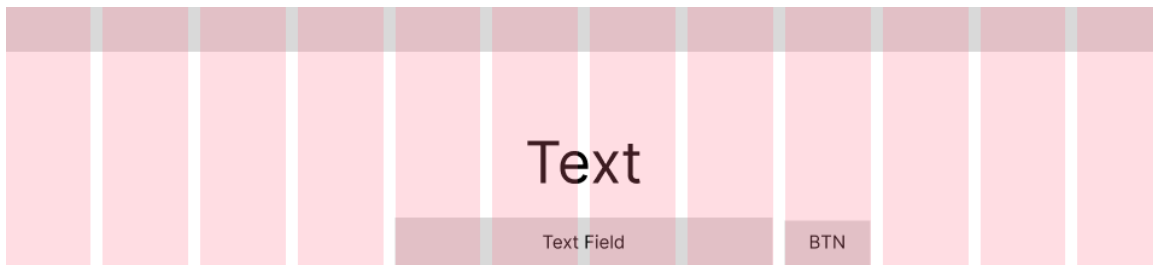


Figure 9 Snippet of a low-fidelity wireframe in Figma

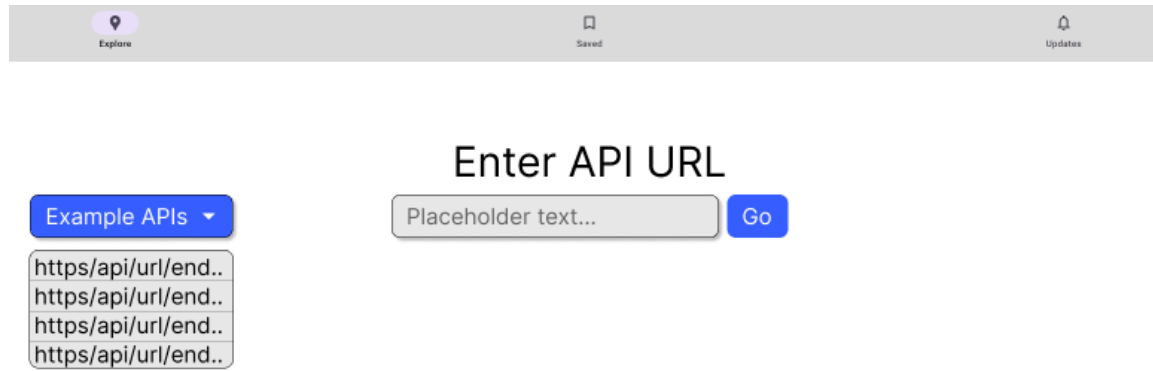


Figure 10 Snippet of a high-fidelity wireframe in Figma

(<https://www.figma.com/design/TpzGsWwOXnjNRTzylt54Q/major-project-design?node-id=0-1&t=LfmZo9zL19D8g9FD-1>)

The element of design is important for visualizations as well, providing a readable graph is necessary as you do not want to confuse your audience and have them misinterpret the data being visualized. While creating graphs, it can be easy to get carried away by trying to enhance the aesthetics of the plot. It is important to maintain a balance between aesthetics and readability; this was kept in mind when creating any graph during the development of the project.

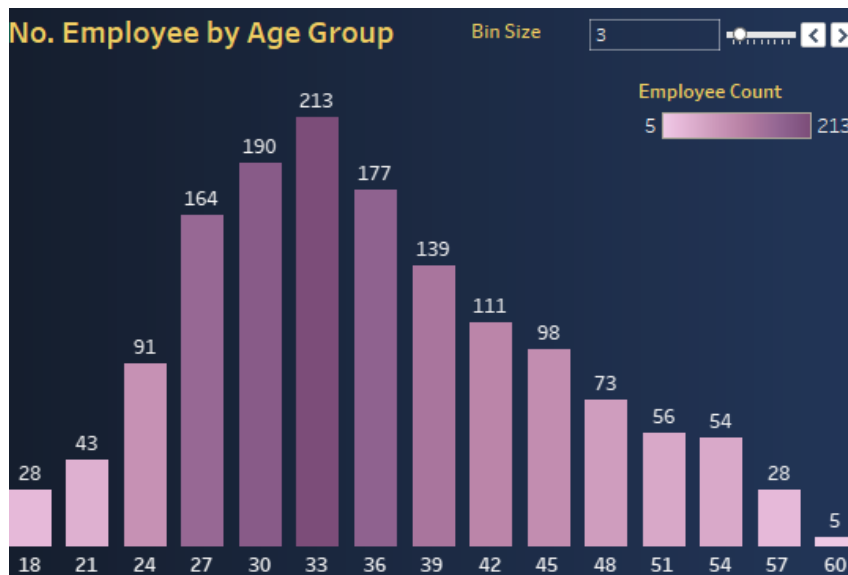


Figure 11 Visualization in Tableau

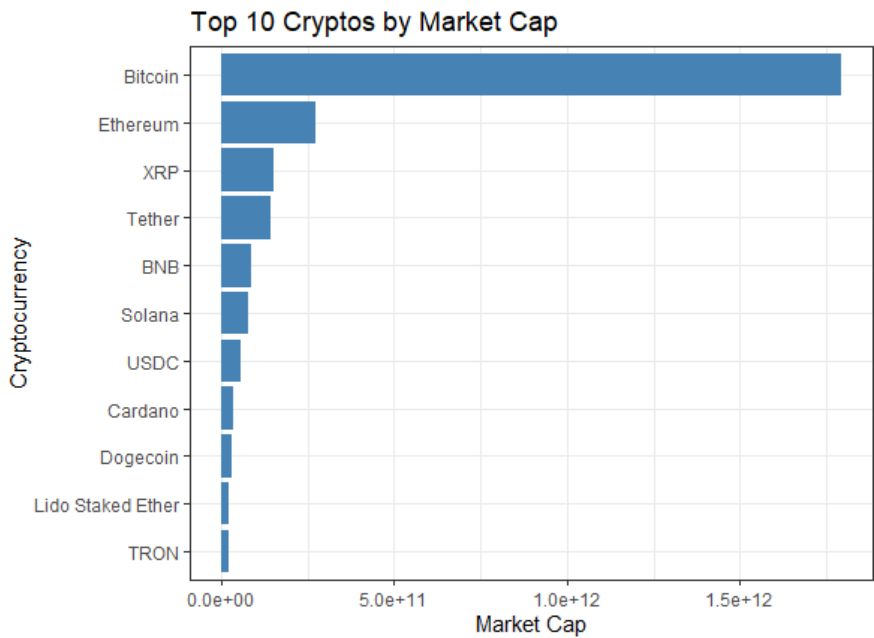


Figure 12 Visualization in R

5. Implementation

5.1 Creating the Front-end

The first step of this application was to get a response from an API. To do this, axios was used, axios is a promised-based HTTP Client for Node.js and the browser. Once you get the response, the response is then stored in a variable to be used.

```
useEffect(() => {  
  axios  
    .get(`https://restcountries.com/v3.1/name/ireland?fullText=true`)  
    .then((response) => {  
      console.log(response.data[0]);  
      setCountry(response.data[0]);  
    });  
});
```

Figure 13 Receiving a response from the API

In the application we want to allow the user to be able to request information from any API, so we needed to get rid of the hardcoded API URL that was passing to the axios.get function. To achieve this, a function called 'handleAPI' was created, this created an input field to allow the user to type in their own API URL and created also new variable called 'inputValue' to track the value entered. Whatever is entered into the input field is now what 'inputValue' is equal to, this value is passed to the axios.get function in the newly created handleAPI method. To commence the request, a button was created, which when clicked, calls the handleAPI method.

```
const handleAPI = () => {  
  axios  
    .get(inputValue)  
    .then((response) => {  
      console.log("API data retrieved from search: ", response.data);  
      setApi(response.data[0]);  
    });  
};
```

Figure 14 Improved get method to allow any value

```
<input
  type="text"
  placeholder="Enter API URL"
  value={inputValue}
  onChange={(e) => setInputValue(e.target.value)}
/>
<button onClick={handleAPI}>GO</button>
```

Figure 15 Button to commence the API request

When the user does a GET request, it is important to display that response back to them. To do this, we took the response, which is in JSON format and applied 'stringify' to it. This takes any value and puts it into a string so it can be displayed to the user.

```
<h3>API Data:</h3>
<pre>
  {JSON.stringify(api, null, 2)}
</pre>
```

Figure 16 Displaying the response back to the user

In figure 16, 'stringify' is given three arguments, the first one is the value, which is passed to the API response data, the second is the replacer, this is null so no filtering or transformation is applied, and the third is the space parameter, after two spaces it add an indentation to make the response not appear all on one line for readability.

```
{
  "name": {
    "common": "Ireland",
    "official": "Republic of Ireland",
    "nativeName": {
      "eng": {
        "official": "Republic of Ireland",
        "common": "Ireland"
      }
    }
  }
}
```

Figure 17 Snippet of the response displayed to the user

Now that features to receive and display data had been implemented, a function which allows the user to export that data was needed so the user can make use of the data, for example exporting numerical data to an excel file to use it in a software such as Tableau which can easily visualize data.

The first step was to find out a way to export the data to an excel file, this was achieved by installing a third-party library called 'SheetJS'. This enables easy reading, writing and manipulation of spreadsheet files directly in the browser.

```
const worksheet = XLSX.utils.json_to_sheet(csvData);
const workbook = XLSX.utils.book_new();
XLSX.utils.book_append_sheet(workbook, worksheet, "data")
XLSX.writeFile(workbook, "api_data.xlsx", { compression: true });
```

Figure 18 Code to export data as to an excel file

Referring to figure 18, the code exports the data provided in the parameters of 'worksheet', creates a new sheet called 'data' and then writes the file.

Being able to filter which data you would like to export is important, so we started to develop a method that the user could choose which data they would like to export.

```
<input
  type="text"
  placeholder="Enter Filter Conditions"
  value={filterInputValue}
  onChange={(e) => setFilterInputValue(e.target.value)}
/>
<button type="button" onClick={handleFilter}>Filter</button>
```

Figure 19 Input field to enter filtering conditions

Referring to figure 19, to allow the user to choose what they would like to filter specifically, an input field was created, the value of the input field is set to 'setFilterInputValue', which is used for the 'handleFilter' method, which can then be called by clicking a button.

```
const handleFilter = () => {

  var newFilter = filterInputValue.split(",");
  filterCriteria.push(...newFilter)
```

Figure 20 handleFilter method

In reference to figure 20, a variable called 'newFilter' was created, which contains the values of the input field, and the values are separated by a comma. For example, if a user entered 'population,temperature', these two values would be separated and pushed into an array as separate elements.

Next, the filtered data needed to be stored, and we also needed to access the properties using dynamic variable names. To do this, bracket annotations were used so we could use our own variable names to access the nested properties.

```
const filteredData = {
  filter1: response.data[0][filterCriteria[0]],
  filter2: response.data[0][filterCriteria[1]],
```

Figure 21 Storing filtered data and accessing properties using the filter conditions

In reference to figure 21, we are going to the response data then accessing the specific property entered by the user by passing the value from the array which stores the input values from the input field.



Figure 22 Input field to enter filter conditions

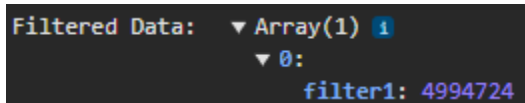


Figure 23 Console view when running the handleAPI function

We also wanted to give the user the ability to export the raw data from the API response, to do this, the API response was passed to the SheetJS function which exports the data. An error was encountered while implementing this, SheetJS does not allow exporting data which are objects. To overcome this, before exporting the data, it gets wrapped in an array.

```
const exportRawData = () => {  
  let dataToExport = api;  
  
  if (!Array.isArray(api)) {  
    if (typeof api === 'object' && api !== null) {  
      dataToExport = [api]; // Wrap object in an array  
    } else {  
      console.error("API data is not an array or object");  
      return;  
    }  
  }  
}
```

Figure 24 Function to export raw API response

In reference to figure 24, currently this code has a bug, where data nested inside of an object is not exported, it currently exports the key name but not the values associated with it, so some values will be missing.

The aim of this is to be as flexible as possible, which means we do not want any hardcoded variables if possible. We changed 'filterCriteria' to be a 'useState' variable which allows us to track changes made to it. This was necessary because 'filterCriteria' is an array, where elements can be added and removed, so it is necessary to track this variable so changes can be made when needed. So now when the user enters values and runs the filter, it is displayed back to the user.

```
(key, value) => {  
  if (key === "" || key === filterCriteria[0] || key === filterCriteria[1]) {  
    if (key === filterCriteria[0] || key === filterCriteria[1]) {
```

Figure 25 Conditional to check if key has same value as filterCriteria elements

In reference to figure 25, keys are now being checked if they are equal to the element in the 'filterCriteria' array. This does need additional work, as you can see, we are only checking the first two elements, what if a user has three filters? More of these conditionals could be added, but it would be best to dynamically add these conditionals depending on how many filter conditions there are.

```
for (let i = 0; i < filterCriteria.length; i++) {  
  filteredData[filterCriteria[i]] = response.data[0][filterCriteria[i]];  
}
```

Figure 26 Conditional to check if key has same value as filterCriteria elements

In reference to figure 26, before the 'filteredData' object had two hardcoded key value pairs, similar to the issue that was just addressed in figure 25, we needed to dynamically create these key value pairs. To achieve this, a for loop was created which iterates over the 'filterCriteria' array inside the '.then' inside of the 'handleAPI' function, and for each element inside the array, it creates a key value pair using the dynamic variables from the 'filterCriteria' array.

At this point, we had developed the application to the point where we had reduced the hardcoding enough to where any API should work. But we started to run into a problem when using other APIs, after receiving the response, trying to filter the data resulted in the page reloading to its default state. To solve this problem, we added a parameter named 'event' to the 'handleAPI' and 'handleFilter' functions. Within both of these functions we also added the line 'event.preventDefault()', this prevents the default actions of functions which solved my problem of the page reloading back to the default state.

```
const handleAPI = (event) => {    const handleFilter = (event) => {  
  event.preventDefault();          event.preventDefault();
```

Figure 27 event.preventDefault() added to prevent default actions

```
.then((response) => {  
    let data = response.data; //by default, access data normally  
  
    if(Array.isArray(response.data)) {  
        data = response.data[0]; //if response data is an array, access first element  
    }  
})
```

Figure 28 Conditional to check if response data is an array

In reference to figure 28, an issue was discovered when using different APIs, sometimes the response data is nested inside of an array, so to solve this, a conditional statement was added to the ‘handleAPI’ function. By default, the data is set to be equal to ‘response.data’, if the response is an array, we change the ‘data’ variable to access the first element of the array by adding ‘[0]’ to the end of ‘data’. This further improves flexibility of the application.

Sometimes the response data can be nested inside of an array called “data”, so we added a conditional to check if the response contained a nested property called “data”, and if there was, the data to be exported was set to “response.data”.

```
if (api && api.data && Array.isArray(api.data)) {  
    dataToExport = api.data;  
}
```

Figure 29 Conditional to check if response contains nested property "data"

```
const flattenObject = (obj, parent = '', res = {}) => {  
    for (let key in obj) {  
        let propName = parent ? parent + '.' + key : key;  
        if (typeof obj[key] == 'object' && obj[key] !== null && !Array.isArray(obj[key])) {  
            flattenObject(obj[key], propName, res);  
        } else {  
            res[propName] = obj[key];  
        }  
    }  
    return res;  
}
```

Figure 30 Code snippet which flattens objects

In reference to figure 30, the 'flattenObject' function is used to transform nested objects into a flat structure. When exporting data from APIs, we would run into issues with some of the data being missing due to data being nested in multiple properties. By flattening the object, we make sure all the nested properties can be accessed as top-level properties, making it easier to access the data.

The first line of the 'flattenObject' function defines the function and takes three parameters, 'obj', the object that will be flattened, 'parent', a string to defining the parent key to build key names and 'res', an object to store the flattened key-value pairs.

The rest of the code begins a for loop which will iterate over each key in the object, defines a new variable called 'propName' which builds the property name for the flattened object. If the parent has a value, it concatenates the parent, a dot and the key name, otherwise it will just use the key name. Then a conditional statement checks whether the value of the current key is an object, not null and not an array. If the value is an object, the function calls itself with the value of the current key, the property name and the result of the object. Lastly, an 'else' statement is defined so if the value is not an object, it adds the key-value pair to the result object using the property name that was built. And after that last conditional statement the resulting object is returned.

```
const flattenedData = dataToExport.map(item => flattenObject(item));  
  
const worksheet = XLSX.utils.json_to_sheet(flattenedData);
```

Figure 31 Creating a new array called 'flattenedData'

In reference to figure 31, this snippet of code created a new array called 'flattenedData', each element inside this array is a flattened version of the correlated element in the 'dataToExport' array. We then pass 'flattenedData' as the parameter in the function which exports data in JSON format to a worksheet in excel.

5.2 Visualizing and Developing in Tableau

At this point, we were satisfied with the functionality of the application, so we decided to start to focus on the visualization aspect of the data. We began to develop in Tableau, taking data exported from my application and making interesting visualizations by utilizing the calculated field, which allows you to write code to show, manipulate or highlight data based on conditional statements.

The first dataset we developed with was data based on cryptocurrency. It contained information on the current price, price change percentage in the past 24 hours, market cap etc.

```
AVG([Market Cap Change 24H]) > 200000000
```

OR

```
AVG([Market Cap Change 24H]) < -7200000
```

Figure 32 Conditional checking if market cap is above or below a specified value

In reference to figure 32, we wanted a graph that displayed the top 10 and bottom 10 cryptocurrencies for market cap. To do this, we created conditional statements that only show cryptocurrencies above 200 million and below negative 7.2 million which results in showing the top and bottom 10 currencies.

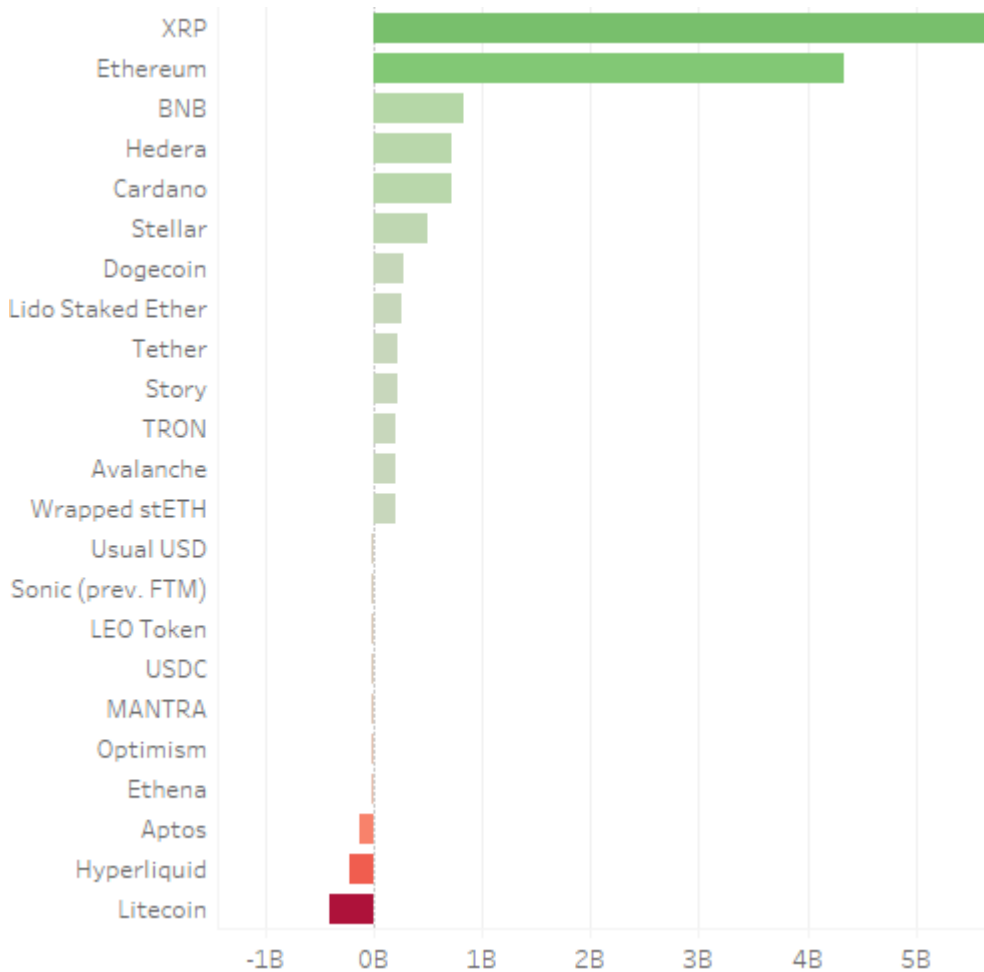


Figure 33 Visualization of market change in 24hr

```

IF SUM([Price Change Percentage 24H]) > 3 THEN "Green"
ELSEIF SUM([Price Change Percentage 24H]) >= 0 THEN "Yellow"
ELSE "Red"
END

```

Figure 34 Conditional statements to assign strings

In reference to figure 34, we wanted a chart which colored the bars based on their value. We created a calculated field which contained three conditional statements, where if the value was greater than three it would tag the currency with “Green”, greater or equal to 0, it would be tagged with “Yellow”, and if below zero, it would be tagged with “Red”. This code does not actually color the bars though; in Tableau it is not possible to color charts based on conditional statements. To achieve the

colors, we dragged the calculated field onto the color mark and selected the colors for the string values “Green”, “Yellow” and “Red”

```
IF [Name] = [Focus Name] THEN "Focus"  
ELSE "Other"  
END
```

Figure 35 Conditional statement to assign the string "Focus"

In reference to figure 35, we also wanted a filter so you could type in the cryptocurrency name, and it would highlight the bar on the chart. To do this, we wrote a conditional which checked in the name of the currency was equal to the parameter “Focus Name”, if it was it would assign the currency the tag “Focus”, if not it would assign the tag “Other”.

Name

Focus Name

Properties

Data type

String ▼

Current value

Bitcoin

Figure 36 Custom parameter

In reference to figure 36, we created a parameter called “Focus Name”, assigned the data type of string to it and gave it a default value “Bitcoin”. With this parameter created all we had to do now to highlight bars was to drag the calculated field “Focus” on the color mark and assign the color to it.

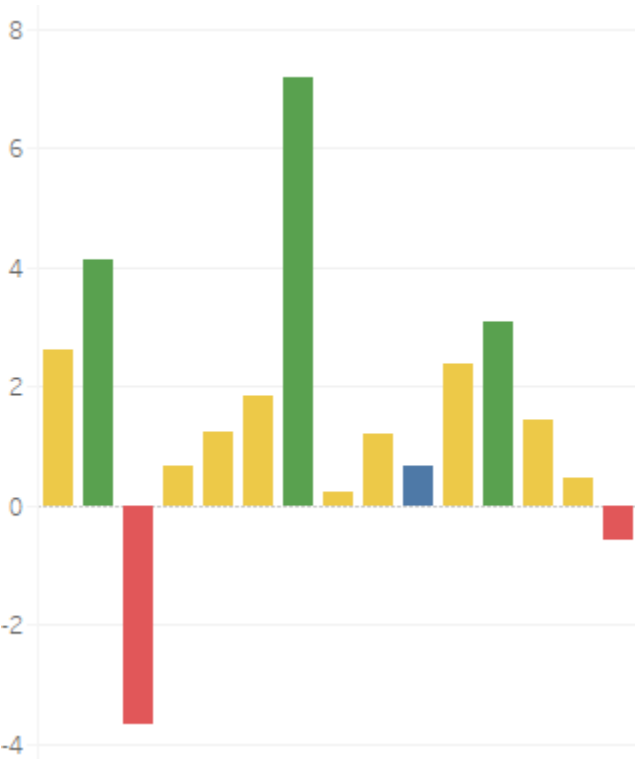


Figure 37 Resulting graph with calculated fields applied and highlighting Bitcoin

Referring to figures 34, 35 and 36, we thought it would look nice in a dashboard to be able to find the price of a coin by typing its name into a filter, so we applied the same logic used in the referred figures to achieve that.

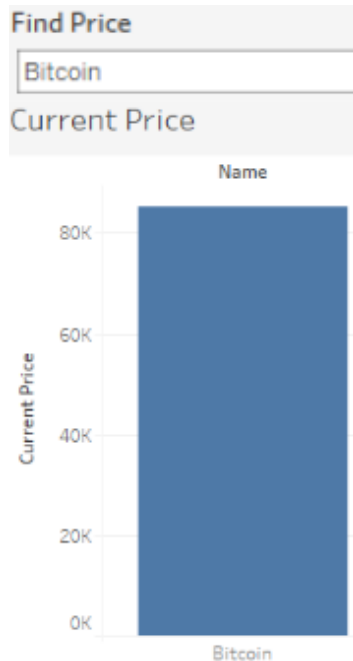


Figure 38 Filtering for "Bitcoin"

Now that we had familiarized with Tableau, we wanted to take another dataset and create a more visually appealing dashboard. The dataset contained information whether the employee had left the company or not. we wanted to use this data to create some graphs that visualize the attrition rate of employees based on gender, department and age group.

First, we needed to calculate the count of the attrition, in the dataset, it was not a numeric value it was either tagged with "yes" or "no", to create the total sum of attrition rate we needed to create a calculation field.

Attrition Count

```
IF [Attrition] ="Yes" THEN 1 ELSE 0 END
```

Figure 39 Conditional statement to assign a 1 or 0 based on attrition value

In reference to figure 39, this calculation assigns "1" if attrition is equal to "yes" and a "0" if it is not.

Next, we wanted to have a percentage value of the attrition rate of employees, to do this we took the total sum of the attrition count from the calculated field in figure 39 and divided it by the sum of employees to get the percentage.

Attrition Rate

`SUM([Attrition Count])/SUM([Employee Count])`

Figure 40 Conditional statement for attrition rate of employees

Active Employees

`SUM([Employee Count])-SUM([Attrition Count])`

Figure 41 Conditional statement to find number of active employees

In reference to figure 41, we also wanted the sum of the total active employees; to return this value we took the sum of employee count and subtracted it by the sum of attrition count.



Figure 42 Creating a dual-axis

In reference to figure 42, to create a “lollipop” visualization of attrition by gender we duplicated the entry into the columns field, because we wanted a horizontal bar representing the value and a circle representing the value. To combine both, we selected the “Attrition Count” tag and enabled dual axis. This combines both of the visualizations into one graph, resulting in the lollipop chart.

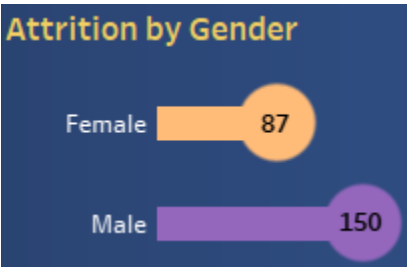


Figure 43 Resulting visualization

For the next dataset, we chose a sales-oriented dataset. For the first visualization we wanted to create a chart that displayed the maximum number of sales and minimum number of sales from either the current year of sales or the previous year of sales. We wanted to depict this using two line-graphs and circles marking the maximum and minimum.

To obtain the values of the maximum and minimum, we needed to create two calculated fields beforehand that find the total current year sales and the total previous year sales.

Total Current Year Sales

```
IF YEAR([Order Date])={MAX(YEAR([Order Date]))} THEN [Sales] END
```

Figure 44 Conditional statement to find total current year sales

In reference to figure 44, this calculated field checks if the order date is equal to the maximum year in the dataset, if it is true, it returns sales, otherwise it returns null. To get the previous year was simple, we took the same calculation and subtracted 1 from the order date value.

Total Previous Year Sales

```
IF YEAR([Order Date])={MAX(YEAR([Order Date]))}-1 THEN [Sales] END
```

Figure 45 Conditional statement to find previous total year sales

Min Max Month

Results are computed along Table (across).

```
IF SUM([Total Current Year Sales])=WINDOW_MAX(SUM([Total Current Year Sales])) THEN SUM([Total Current Year Sales])
ELSEIF SUM([Total Current Year Sales])=WINDOW_MIN(SUM([Total Current Year Sales])) THEN SUM([Total Current Year Sales])
ELSE NULL
END
```

Figure 46 Conditional statement to find the maximum and minimum sales of the current year sales

In reference to figure 46, with the previously calculated fields created, we can now determine the maximum and minimum values for the sales. In this calculation field, the first conditional checks if the total current year sales are equal to the current maximum value in the current window, if it is, it returns the maximum sum. The second conditional check applies the same logic but for the minimum value within the window.

With these calculation fields done, we could visualize the data, we also made the graph a dual-axis chart with circles to the maximum and minimum sales.



Figure 47 Resulting visualization

I repeated this process to visualize total profit and total quantity.

5.3 Visualizing and developing in R

With this, we were satisfied with the work and progress in Tableau, this allowed us to explore R, another data visualization software like Tableau, but it is more coding oriented. R is an open-source programming language and software environment designed for statistical computing and data visualization. Over the years R has gained popularity and has become one of the most widely used tools for data analytics and research applications. One of R's biggest strengths is its flexibility, through the packages created by the community via the Comprehensive R Archive Network (CRAN), users can install these packages and create professional quality visualizations.

In the following section, we will detail how R was used in this project to visualize data, highlighting packages used to enhance these visualizations throughout the process.

To familiarize myself with the R environment we used a dataset that comes installed with R and made a few visualizations. Beforehand, we had to install some packages such as "tidyverse" and "ggplot" to generate the graphs.

The first dataset was a biochemical oxygen demand dataset (BOD), "the BOD data frame has 6 rows and 2 columns giving the biochemical oxygen demand versus time in an evaluation of water quality."

```
ggplot(BOD, aes(Time, demand))+  
  geom_point(size=3)+  
  geom_line(color = "red")
```

Figure 48 Code the visualization in R

In reference to figure 48, the function “ggplot” is taking two arguments, first is the data source and second is “aes” which itself can take multiple arguments, in this case it is just the x-axis data and the y-axis data. The code with “geom” is creating the geometry of the visualizations, the first one is plotting all the data with points, and the second one is plotting the data with lines.

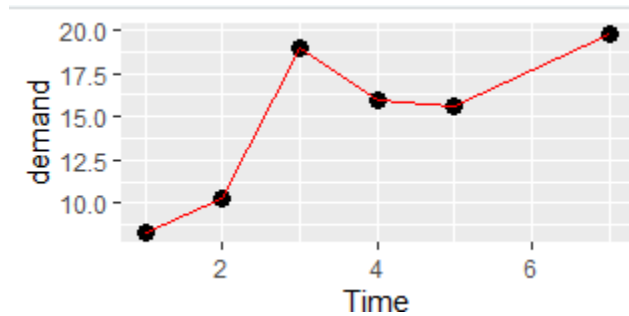


Figure 49 Resulting visualization

The second dataset we used was carbon dioxide uptake in grass plants, “the data frame has 84 rows and 5 columns of data from an experiment on the cold tolerance of the grass species *Echinochloa crus-galli*”.

```
ggplot(CO2, aes(conc, uptake,  
                colour = Treatment)) +  
  geom_point(size = 3, alpha = 0.5) +  
  geom_smooth(method = lm, se = F) +  
  facet_wrap(~Type) +  
  labs(title = "Concentration of CO2")
```

Figure 50 Code snippet to create the visualization

For this visualization, we added the colour argument to “aes” and set the colour value to be equal to “treatment”, which will help identify what data is the “chilled” versus “non-chilled”.

I added the code with “geom_smooth” to create a linear regression line, denoted by “method = lm”. The fifth line in figure 50 creates multiple plots based on the values in the “Type” column. Each unique value gets its own plot, in this case there are two values, Quebec and Mississippi. And the final line of code creates a title for the plot.

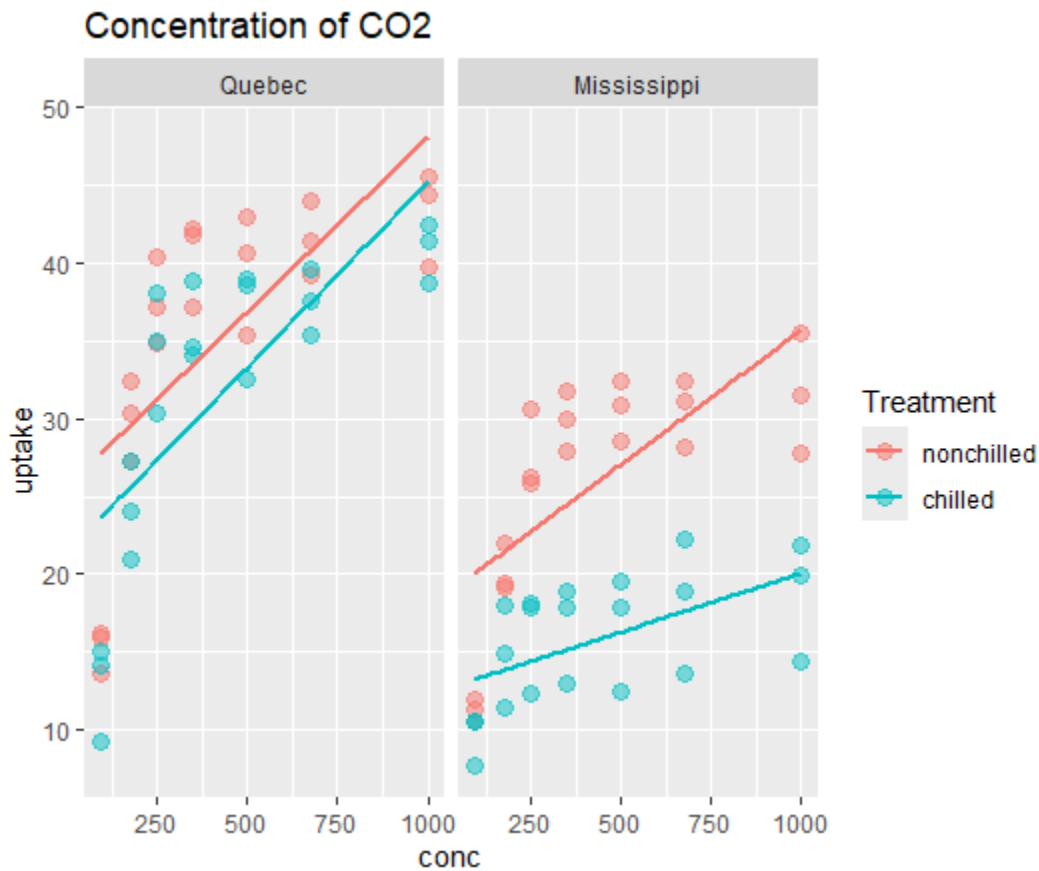


Figure 51 Resulting visualization

After we familiarized ourselves with the R Studio environment, we wanted to take the cryptocurrency dataset we extracted from my website and used in Tableau and try creating the same visualizations in R Studio to see how it would compare.

The first visualization we wanted to recreate was a horizontal bar chart displaying the price change of cryptocurrencies in the past 24 hours.

```
10 p1 <- ggplot(CryptoData, aes(x = reorder(name, price_change_percentage_24h),
11                               y = price_change_percentage_24h,
12                               fill = price_change_percentage_24h)) +
13   geom_col() +
14   scale_fill_gradientn(colors = c("red", "green"),
15                       values = scales::rescale(c(-10, -5, 0, 5, 14))) +
16   labs(x = "Name",
17        y = "Price Change Percentage (24h)",
18        title = "24h Price Change of Cryptocurrencies") +
19   coord_flip() +
20   theme_bw() +
21   theme(axis.text.y = element_text(size = 8))
```

Figure 52 Code to create the visualization

In reference to figure 52, the first argument given to “ggplot” is “CryptoData”, which is the dataset exported from the frontend website created. The graph is sorted by descending by using the “reorder” function on line 10. The color is based on the current price as shown on line 12. To achieve a gradient color scale, on line 14, we give gradient function two colors to use, red and green. On line 15, we define values at which the colors transition and we use rescale to scale the value between 0 and 1 so they within the ggplot color scale.

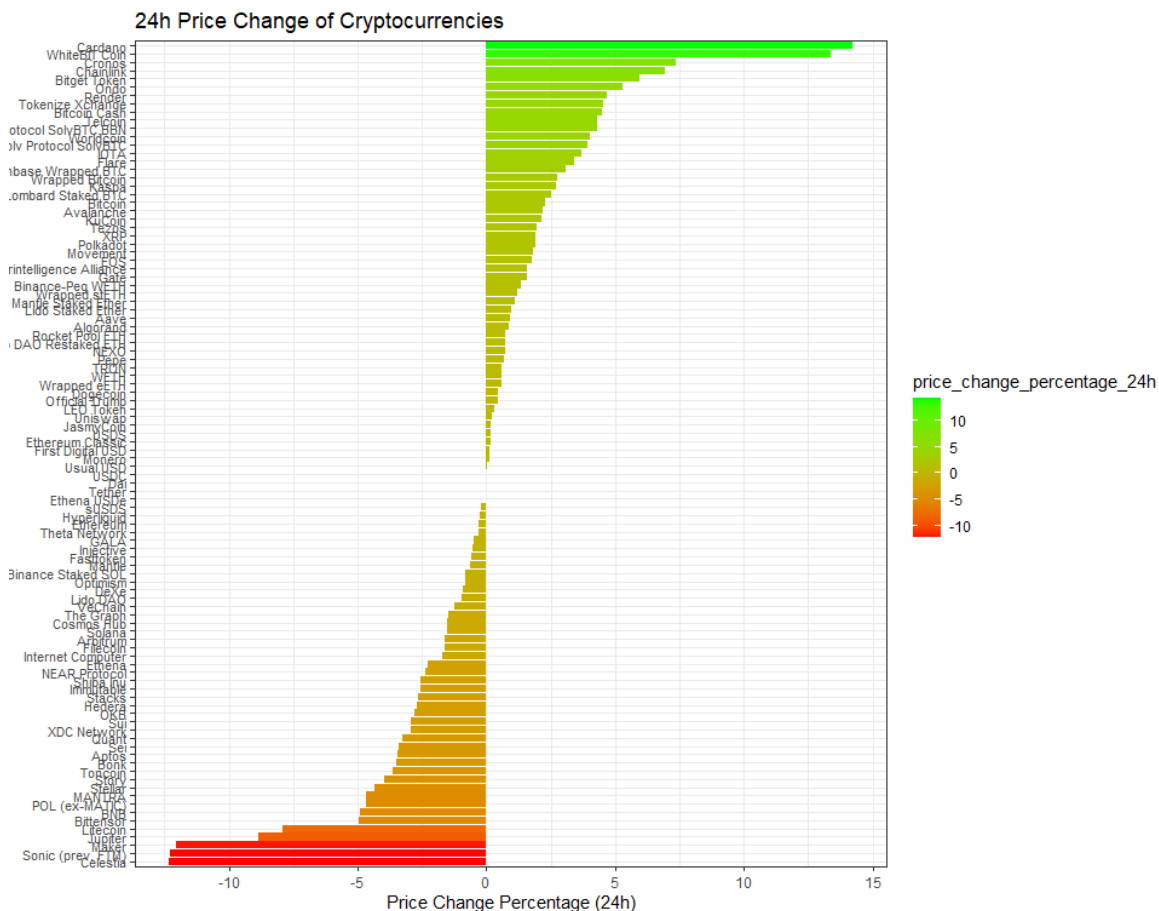


Figure 53 Resulting visualization

```
p2 <- CryptoData %>%
  filter(market_cap > 20000000000) %>%
  arrange(desc(market_cap)) %>%
  ggplot(aes(x = reorder(name, market_cap), y = market_cap)) + #Sorting
  geom_col(fill = "steelblue") +
  labs(x = "cryptocurrency", y = "Market cap", title = "Top 10 Cryptos by Market Cap") +
  coord_flip() +
  theme_bw()
```

Figure 54 Code to create the visualization

In reference to figure 54, we first apply the pipe operator represented by “%>%” to the dataset. This allows us to filter the data and when doing this, “ggplot” now knows my data source, so we do not need to declare it in the first parameter. We order the data by using the “arrange” method and by defining which column we want to sort, which is market cap. Then when we define the plot with “ggplot”, we use the reorder method which sorts the categorical data on the x-axis. We color the bar chart by using the fill argument in “geom_col”, and then we display the appropriate labels with the “labs” method and flip the coordinates of the chart to create a horizontal bar chart.

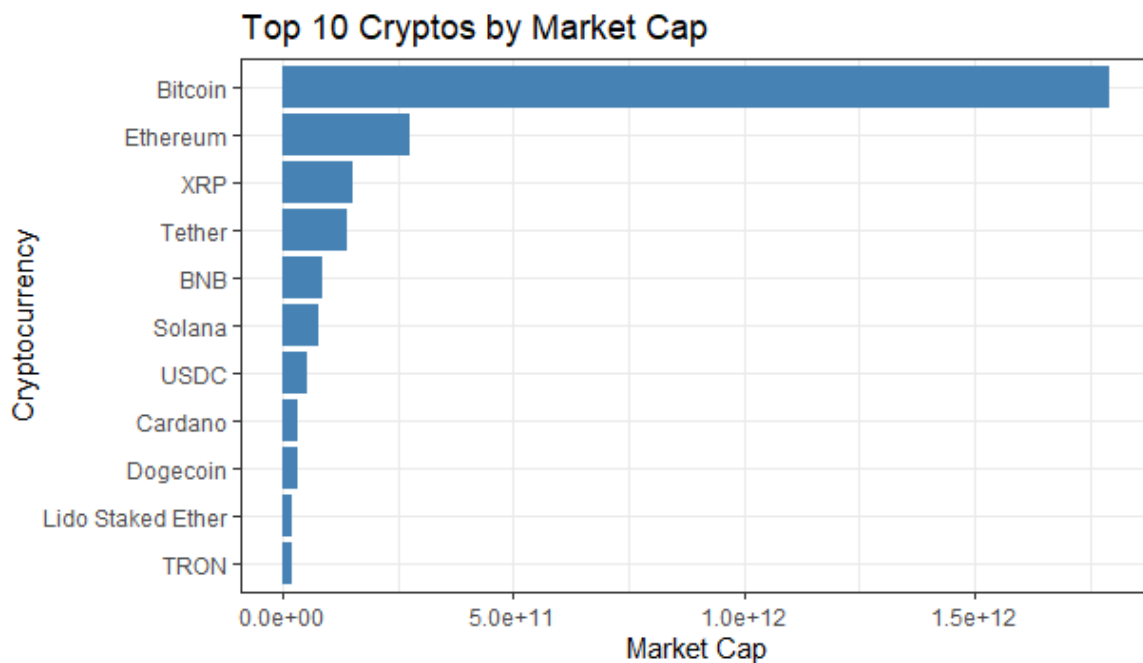


Figure 55 Resulting visualization

```
p3 <- ggplot(CryptoData, aes(area = market_cap, fill = market_cap, label = name)) +
  geom_treemap() +
  geom_treemap_text(size = 10, color = "white", place = "center") +
  scale_fill_gradient(low = "purple", high = "blue") +
  labs(title = "Treemap of Cryptocurrencies by Market Cap")
```

Figure 56 Code to create the visualization

In reference to figure 56, we wanted to create a tree map of the market cap, to create a plot of a tree map we first had to install the package “treemapify”. To display the labels in the tree map we added the “label” argument to aesthetic of ggplot. To ensure that the text generated is readable, we increased the size and defined the positioning of the text to center. We added the scale gradient to color the chart, if the value was low, it was assigned purple, if it was high, it was assigned blue. Unfortunately, the gradient effect was not visible as Bitcoin held such a large

majority of the market cap that the rest of the cryptocurrencies were all the same gradient of purple.

Treemap of Cryptocurrencies by Market Cap

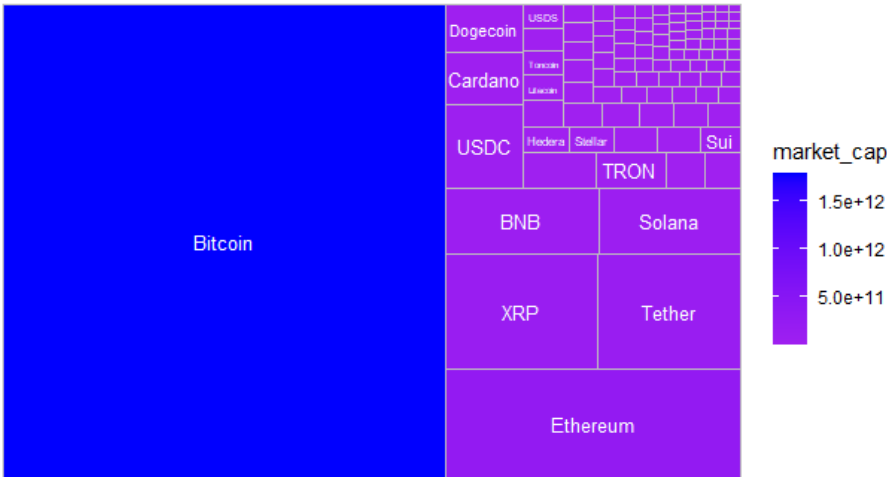


Figure 57 Resulting visualization

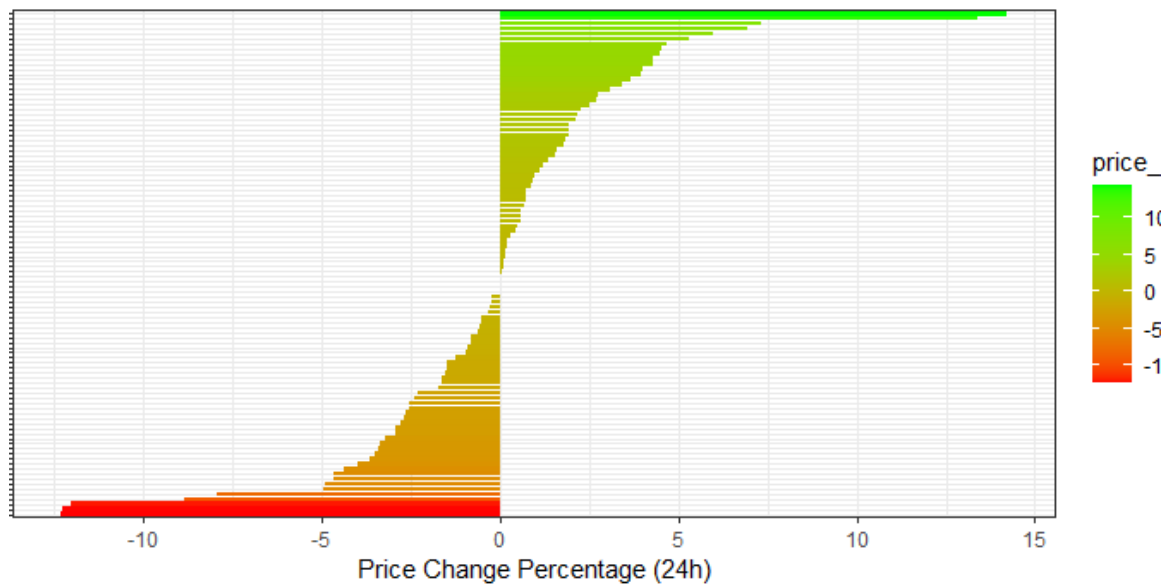
In figures 52, 54 and 56, at the beginning of each of the snippets of code we would begin by storing the following code by creating a variable named “p” followed by a number. So, with three graphs, we had p1, p2 and p3, these represented each of the plots. To assemble all of these plots into a dashboard, we installed a package called “patchwork” which allows you to do this by defining each plot with a variable. You can then print the dashboard in a customizable layout.

```
43 dashboard <- p1 / (p2 | p3)
44 print(dashboard)
```

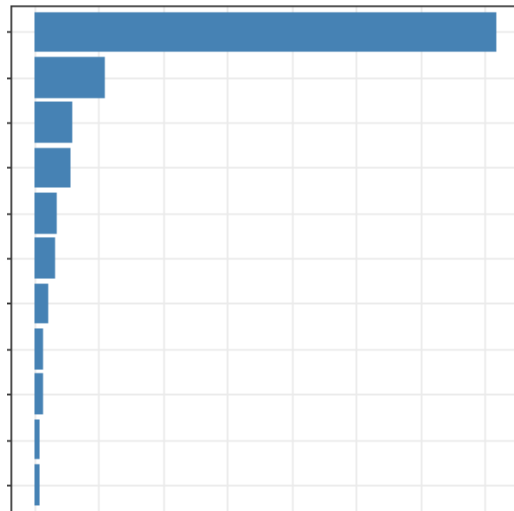
Figure 58 Printing the dashboard

In reference to figure 58, on line 43, we can define the layout of the dashboard. P1 will be printed on the top by itself and P2 and P3 will be printed side by side, this is denoted by the “|”.

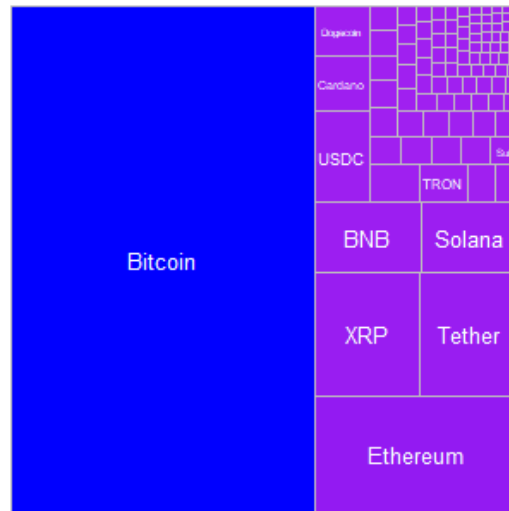
24h Price Change of Cryptocurrencies



Top 10 Cryptos by Market Cap



Treemap of Cryptocurrencies by Market Cap

*Figure 59 Result of the dashboard*

Next, we wanted to revisit the visualization which finds the price of a cryptocurrency using Shiny. Shiny is an R package which allows you to build web applications from within R.

```

79 library(shiny)
80 library(shinydashboard)
81
82 # creating the shiny UI elements
83 ui <- dashboardPage(
84   dashboardHeader(title = "Cryptocurrency Price Viewer"),
85   dashboardSidebar(
86     selectInput("name", "Select a Cryptocurrency:", choices = unique(CryptoData$name))
87   ),
88   dashboardBody(
89     valueBoxOutput("price_box")

```

Figure 60 Defining the Shiny user-interface

Referring to figure 60, first, Shiny and its dashboard library is loaded, on lines 83, the user-interface (UI) is being defined. The ‘dashboardPage’ is the container, all the UI elements will be placed inside of it. On line 84, the title is defined and on line 85, the sidebar is created. Inside the sidebar, a selection menu is provided, this will contain all the cryptocurrency names. On line 86, the first argument of “selectInput” is “name”, this is the input ID, this will be used to refer to the selection later. The second argument is the label text of the selection menu, the third argument, “choices”, will define the dropdown options. “Choices” is assigned all the unique values from the name column of the CryptoData data frame, the data is being accessed by using the \$ operator. On line 88, the dashboard body content is defined, we create a placeholder UI, where a value box will be rendered, this is where the information about the cryptocurrency will be displayed.

```

93 server <- function(input, output) {
94   output$price_box <- renderValueBox({
95     price <- CryptoData %>% filter(name == input$name) %>% pull(current_price)
96
97     valueBox(
98       subtitle = paste("Current Price of", input$name),
99       value = paste0("$", format(round(price, 2), big.mark = ",")),
100      color = "blue"

```

Figure 61 Creating the Shiny server

In figure 61, on line 93, the server is defined with two arguments, input and output. Input will contain the inputs from UI, for example the name of the cryptocurrency from the dropdown menu. Output is what is sent to the UI, this will be the “valueBoxOutput” which was defined in figure 60. On line 94, an interactive output called “price_box” is created, this will appear wherever “renderValueBox” is used, inside this function is what will be displayed. On line 95, a variable called “price” is defined, we start by using the pipe operator (%>%) on the CryptoData dataset. Next, we filter the dataset where the column is equal to the user’s selection from the menu. Then, using the “pull” function we can get the price of the cryptocurrency by providing the column name “current_price” to the function.

With the output defined, on line 97, a “valueBox” is created, this creates a card-like component display the output. I add a subtitle to the card and use the “paste” function, in R this allows you to concatenate multiple pieces of text together, so this allows us to combine a fixed piece of text with input value selected from the dropdown menu. On line 99, I use the “value” parameter, to display the price, then use “paste0” which adds text together without a space. The price is then formatted, rounding to only display two decimals and the argument “big.mark” with the value of “,” is used to include commas in the value where necessary. Lastly the color of the card is defined on line 100.

```
105 # runs the shiny app
106 shinyApp(ui, server)
```

Figure 62 Code to run the Shiny application

Referring to figure 62, when this code is executed the Shiny application is launched, connecting the UI and server functions that were defined.

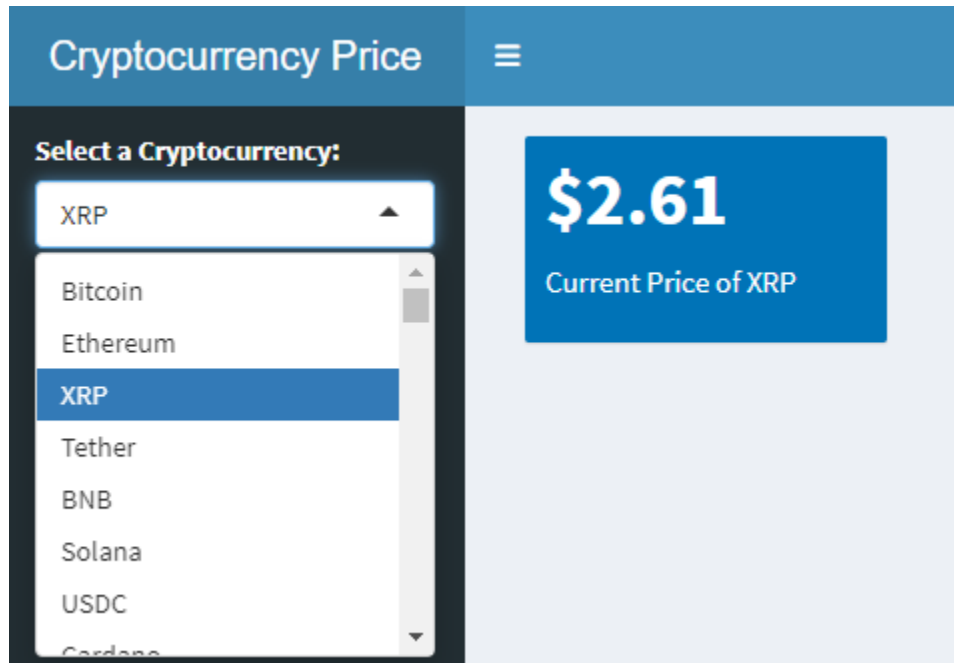


Figure 63 Result the Shiny dashboard

In summary, this section discussed the process of creating the front-end and developing in the software Tableau and R. Implementation of features were discussed and how they would be approached, any issues or challenges that arose during the development were identified and discussed, documenting how features evolved as a result of these challenges during the development.

6. Testing and Analysis

6.1 Functional Testing

A number of functional tests were carried out to identify if there were any unexpected outcomes and to see if the application was operating as planned.

	Action	Inputs	Expected Output	Actual Output	Test Result	Test Comments
1	Click "Go" with no value entered into the text field for the API URL	N/A	Error message	No error message	Fail	Error messages provide use feedback for developers and users
2	Enter a valid API URL and click "Go"	API URL	Response data displayed on screen	Response data was displayed	Pass	Adding a success popup could be beneficial
3	Export raw data of a response	Enter valid URL, click "Go" then click "Export Raw Data"	File download begins	File was downloaded	Pass	Implementing a popup saying "Data successfully exported" could be a nice confirmation for users
4	Clicking "Export Raw Data" with no valid API URL entered	Click "Go" with no input then click "Export Raw Data"	A blank excel file is downloaded	No file is downloaded	Pass	A popup saying "Cannot export raw data" would be a nice error

						message to have
5	Use the filter feature	Enter a valid API URL and enter "population" as the filter	"population" is displayed with its respective value	"population" was displayed and so was its value	Pass	This filter feature is inconsistent, it works on some APIs and does not on others

From these functional tests a conclusion was drawn that there is a lack of error handling and a lack of communication to the user through popups. Without error messages, users can be left confused and frustrated, less likely to return to your service. The addition of error messages can allow a user to figure out the issue and correct it resulting in a better experience.

6.2 User Testing

User testing was conducted both in-person and online, a total of 5 users were tested. To create a realistic testing environment, before the test began, we gave the users the tasks to be completed. Once the test began, we were not able to help the user.

The following tasks were given to each user during the testing process:

1. Select and copy the second URL from the "Example API URLs" button, paste the URL copied into the text field and click "Go".
2. Once a response is received from the API, export the raw API data.
3. Read through the response displayed on screen and enter a filter condition into the text field.

These tasks were chosen as they represent core functions of the application and will provide useful insight from observing how the user interacts with these features.

Common themes emerged from the user testing, such as reducing the number of actions to be performed by the user. This is most prominent for task 1, which requires a total of 6 actions. Another theme was that there is no confirmation popup for when you export the raw API data, the download just begins the moment you click the button.

The final theme that emerged in all the user tests was using the filter feature, for instance some of the users found it difficult to read through the API response and they were not clear on how to use the filter.

6.3 Insights and learnings

We believe that the overall user testing experience was a positive one, no complaints were received from the users on how the test was conducted and appreciated how we refrained from assisting them during the test as this created an authentic testing experience, this also helps us as the data collected is realistic and represents a user's genuine experience.

As discussed in section 6.2, three main themes were identified from the testing, the following themes were:

1. Reduce the number of actions required for the user to send a GET response.
2. No confirmation to begin the download when exporting the raw API data.
3. Unclear on how to use the filter system.

To address theme 1, the process of selecting an example URL can be streamlined by instead of having to click the button to open the dropdown and highlight the URL etc., it should simply enter the URL into the text field when you click on the URL.

The solution to theme 2 is simple, when clicking on the button to export the raw API data, before the download process begins a popup should appear, asking the user if they are sure they want to download the file with a "Yes" or "No". Additionally, we

can also allow the user to name the file, currently the name of the file is hard-coded, which means file name cannot be changed unless the code itself is altered.

As for theme 3, an information icon could be added in the section where the filter feature is on the website. When hovering over the icon, a brief explanation on how to use the filter feature would popup. A user pointed out that it would be much more convenient to be able to click on the data displayed and filter it that way. This could be done by rendering tick boxes next to each key of data to select what you want to filter instead of having to type it out. This approach would require an entire rework of the filter system and would most likely prove to be difficult to implement, but would be extremely beneficial in the long-term for users in the future.

The information collected from the testing allowed us to see that the lack of communication was a major flaw, which was creating a worse experience for the users. We would like to address these problems by implementing the solutions mentioned in each of the themes that were identified.

7. Project Management

7.1 Project Management Tools

7.1.1 *GitHub*

GitHub is a service to host and share your code online and has tools to allow for collaboration. To use GitHub, you can create a repository to commit and push your code with meaningful comments, you can also clone the repository to different machines if you work from multiple locations.

GitHub was mainly used when developing the front-end and was not used when developing in Tableau and R, files were saved to cloud storage to be accessed from anywhere.

7.1.2 *Miro*

Miro is an online, collaborative whiteboard platform that allows teams to visually brainstorm, create and share ideas through a real-time collaboration.

Miro was used to collect resources for the research section, documenting the development on the project along with meaningful comments and a kanban board was placed to track any bugs with the application along with the proposed solution to fix them.

7.1.3 *Figma*

Similar to Miro, Figma is a collaborative, web-based design tool mainly used to create prototypes for user interfaces (UI) for applications. In this project Figma was used to prototype the UI for the front-end application as discussed in section 4.

7.2 Communication

To ensure that the project was progressing steadily, meetings were scheduled weekly with the project's supervisor. This helped greatly as there were times where progress was slower than expected and the supervisor provided valuable information and feedback to assist the progress of the project. We believe that communication was sufficiently exercised and was effective throughout the whole process of this project and learned that without regularly scheduled meetings the project could easily derail.

8. Conclusion

Initially at the start of this project, a large amount was dedicated to developing the filter system to allow users to filter the API response data by typing in words into a text field such as “population, country” for example, which would output only those two keys and their respective values. As we began to test this function with more APIs, it became clear that it would be extremely difficult and require even more time to make this system work on all APIs. The issue lies in how API responses can differ from each other, nesting data differently which causes issues with the code and parsing through the data structure. It also became evident that the filtering would most likely be a feature used by few as exporting the raw data is the easiest thing to do and software like Tableau, allows you to import the data and you can drag in what data you want without the need for filtering it out specifically.

A vast amount of time was also dedicated to the exportation of the data, specifically the formatting of it. We applied a number of techniques so that the data would be formatted as expected, such as in “long form”, this means that each column represents a category of data. This format is very beneficial to software like Tableau, as you would only have to drag in two columns of data to create a visualization. But like the systems previously discussed, making sure this works on every API response would require a considerable amount of additional time and work.

While a proficiency was developed in R, I would have liked to spend more time developing in R, we would have liked to create more visualizations utilizing more packages to create more in-depth, complex, interactive graphs. If I were to start this project again, the R environment would have played a more significant role and the development process in the R environment would have begun much sooner.

The target audience for this application was advertised to be aimed at people familiar with technology aged 20 and above, but the application became more of a tool that a developer would use instead. There are some improvements that could be made to the application to achieve the original target audience such as establishing a link from the website to the two software we spent time developing in, Tableau and R. As it currently is, the user who exports the data would have to already know what to do with it and how to use it, but for example, if a feature on the

website was developed to open the exported data in one of the software or direct them to an installation of the software.

Throughout this project, a variety of skills were developed such as learning how to create visualizations and aesthetically pleasing dashboards in Tableau, although it may look simple on the surface, you can create complex graphs that require calculated fields, specific formatting, joining data and custom axes to achieve the visualization. We learned how to develop in the R environment, compared to Tableau, R is more flexible and far more coding oriented. Learning how to develop in an open-source environment is a valuable skill as we were constantly learning about and installing useful packages created by the community of R.

My ability to read data also got better. Over the course of the project, I looked at many datasets and can now identify variables from datasets that can create useful visualizations.

As someone who was new to Tableau and R, I believe that I did well to learn both of the software environments and develop visualizations. Particularly for R, custom packages were installed made by the community to achieve visualizations created adding a layer of complexity on top of already trying learn the software and the language of R. Developing in Tableau was beneficial as, patterns in data visualization were identified, understanding which data should be on either the x-axis or y-axis, and what graph would be best suited to represent the data to create the visualization. So, when development started in R, visualizations were quickly realized due to the skills gained from Tableau.

The scope of some features such as the filter system, the data exportation and trying to make the application flexible enough to work with every API were too ambitious. Upon reflection, supporting a few core APIs to work with the application would have been a wiser decision. This would allow us to continuously update the application and add support for new APIs based on user feedback.

As for future development, the project can be taken further by refining and developing some additional features such as rethinking how the data filtration

system works and allowing users to supply a key to the API GET request allowing for more flexibility of the overall application.

The application would also benefit by streamlining the process of introducing users to visualization software such as Tableau and R. As R is free to use this could be easily implemented as we could give a short introduction to R and provide a download link to the software, but Tableau requires an activation key to be used for an extended period of time. This could be overcome by contacting the Tableau team, discussing the purpose of our application and proposing that users who download Tableau through our referral link receive a free 6-month trial of Tableau.

In section 6.3, three themes were identified and solutions to solving those were discussed. If given more time, we would have liked to implement these solutions as they would further enhance the application, creating a better experience for the user.

More time would have liked to be spent developing in R, specifically using the Shiny package which allows you to create interactive plots with filters.

References

(2025). Youtube.com. <https://www.youtube.com/@datatutorials1>

(2025). Youtube.com. <https://www.youtube.com/@RProgramming101>

“Big Data: What It Is and Why It Matters.” Sas.com, 2024, www.sas.com/en_ie/insights/big-data/what-is-big-data.html. Accessed 3 Dec. 2024.

API Architecture Patterns and Best Practices. (2024). Catchpoint.com. <https://www.catchpoint.com/api-monitoring-tools/api-architecture>

API Micro Official. (2023, February 17). Public vs Partner vs Private vs Composite APIs: What Are the Differences? Medium. <https://apimicro.medium.com/public-vs-partner-vs-private-vs-composite-apis-what-are-the-differences-e312c8657ac9>

Benzell, S., Lagarda, G., & Van Alstyne, M. W. (2017). The impact of APIs in firm performance. Boston University Questrom School of Business Research Paper, (2843326).

Chen, M. (2024, March 11). What Is Big Data? Oracle.com; Oracle. <https://www.oracle.com/ie/big-data/what-is-big-data/>

Cooksey, B. (2014). An introduction to APIs.

Elena, C. (2011). Business intelligence. *Journal of Knowledge Management, Economics and Information Technology*, 1(2), 1-12.

Grafana OSS: Leading Observability Tool for Visualizations & Dashboards. Grafana Labs. (n.d.-b). <https://grafana.com/oss/grafana/>

Hall, D. (2024, August 1). How to use an API: Just the basics. TechnologyAdvice. <https://technologyadvice.com/blog/information-technology/how-to-use-an-api/>

Hansoti, B. (2010). Business intelligence dashboard in decision making.

Kabacoff, R. (2025). *Chapter 3 Introduction to ggplot2 | Modern Data Visualization with R*. Github.io. <https://rkabacoff.github.io/datavis/IntroGGPLOT.html>

Khan, R. A., & Quadri, S. M. (2012). Business intelligence: an integrated approach. *Business Intelligence Journal*, 5(1), 64-70.

Kotstein, S. (n.d.). Which RESTful API Design Rules Are Important and How Do They Improve Software Quality? A Delphi Study with Industry Experts. Retrieved December 12, 2024, from <https://arxiv.org/pdf/2108.00033>

Mastering API Architecture: A Comprehensive Guide | Astera. (2024). Astera. https://doi.org/1053553860/l7WGCJv0_V0QxOmv9gM

Milhomem. (2021, June 14). *What is Redash?*. Medium. <https://milmeninos.medium.com/what-is-redash-1715e105e778>

Morris, A. (2021, April 16). 23 Case Studies and Real-World Examples of How Business Intelligence Keeps Top Companies Competitive. Oracle NetSuite. <https://www.netsuite.com/portal/resource/articles/business-strategy/business-intelligence-examples.shtml>

Murphy, L., Kery, M. B., Alliyu, O., Macvean, A., & Myers, B. A. (2018, October). API designers in the field: Design practices and challenges for creating usable APIs. In 2018 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC) (pp. 249-258). IEEE.

Orlovskiy, D., & Kopp, A. (2020, December). A Business Intelligence Dashboard Design Approach to Improve Data Analytics and Decision Making. In IT&I (pp. 48-59).

Parameswaran, V. (2023, April 25). Rest API design: Filtering, sorting, and Pagination. Atatus Blog - For DevOps Engineers, Web App Developers and Server Admins. <https://www.atatus.com/blog/rest-api-design-filtering-sorting-and-pagination/#query-parameters>

Pence, H. E. (2014). What is Big Data and Why is it Important? - Harry E. Pence, 2014. Journal of Educational Technology Systems. <https://doi.org/10.2190/ET.43.2>

Salesforce. (2020). Salesforce. <https://www.salesforce.com/ca/hub/analytics/why-use-big-data-visualization/#:~:text=Advantages%20of%20Big%20Data%20Visualization%20Tools&text=Big%20data%20visualization%20cuts%20out,interactive%2C%20visually%2Dpresented%20dashboards>.

Santoro, M., Vaccari, L., Mavridis, D., Smith, R., Posada, M., & Gattwinkel, D. (2019). Web application programming interfaces (apis): General purpose standards, terms and European commission initiatives. Eur. Commission, Luxembourg, Luxembourg, UK, Tech. Rep. JRC118082.

Shiny - Welcome to Shiny. (2025). Shiny. <https://shiny.posit.co/r/getstarted/shiny-basics/lesson1/>

Shiny components. Shiny. (n.d.). <https://shiny.posit.co/r/components/>

Shiny dashboard. (n.d.). <https://rstudio.github.io/shinydashboard/index.html>

Shivang. (2024, June 9). *What is Grafana? why use it? everything you should know about it*. Scaleyourapp. <https://scaleyourapp.com/what-is-grafana-why-use-it-everything-you-should-know-about-it/>

Stobierski, T. (2021, January 28). Bad data visualization: 5 examples of misleading data. Business Insights Blog. <https://online.hbs.edu/blog/post/bad-data-visualization>

Stoplight. (n.d.). Types of APIs: Types of API calls & REST API protocol. <https://stoplight.io/api-types>

Unwin, A. (2020). Why is data visualization important? What is important in data visualization? Harvard Data Science Review, 2(1), 1.

What Is API Architecture? | Akana by Perforce. (2019). Akana. <https://www.akana.com/blog/api-architecture#layers-of-api-architecture>

Why Data Visualization Is Important in Big Data? - Sigma Solve. (2023, August 24). Sigma Solve INC. - Leverage Our Technology Consulting to Empower Your Business : Hire an Enterprise Solutions Expert. <https://www.sigmasolve.com/blog/why-data-visualization-is-important-in-big-data/>

Wulf, J., & Blohm, I. (2020). Fostering value creation with digital platforms: A unified theory of the application programming interface design. Journal of Management Information Systems, 37(1), 251-281.

Yau, N. (2015, August 31). Bar Chart Baselines Start at Zero. FlowingData. <https://flowingdata.com/2015/08/31/bar-chart-baselines-start-at-zero/>

Yau, N. (2017). How to Spot Visualization Lies. Guides: Tips and suggestions for.