– Independent Work Report Spring 2022 –

Visual Statistics: Exploring a New Way of Teaching Unintuitive Math

Alex Baroody
Adviser: David Walker

Abstract

This project implemented a web-based tutorial focused on providing supplemental material on unintuitive statistical concepts in a visual manner. The goal of this project was to help students and people of all levels of statistical knowledge further their understanding of difficult statistical concepts, focused on type I and type II errors. It has three simple, visually focused, and easy-to-use tutorials on these errors, each targeted towards a different level of prior knowledge. The Basic tutorial focuses on introducing key words and ideas, the Intermediate tutorial focuses on Conditional Probability and Bayes' Theorem, and the Advanced tutorial focuses on formally defining type I and type II errors, as well as formal hypothesis testing. In testing, the vast majority of users had positive experiences, only giving criticism on minor technical issues. This paper describes the research, design process, technical implementation, and evaluation of this tool. These tutorials can be found at `https://visual-stats.herokuapp.com/`.

1. Introduction

Statistics is often taught in confusing and verbose ways. This makes the more complicated aspects of the field especially difficult for visual learners and those who do not have a solid understanding of supporting mathematical concepts. The goal of this project is to make a set of tutorials that act as a supplement to classroom instruction on an area of Statistics that I find especially challenging and important: hypothesis testing. I designed and implemented three separate tutorials at different levels of statistical depth, aiming to have a way for all curious or struggling users to learn Statistics, no matter their level of prior experience. The Basic tutorial assumes very little prior statistical and mathematical knowledge, while the Advanced tutorial assumes a much more complete understanding of high school-level mathematical concepts. Ideally, this tool will provide users with an intuitive way to learn a few complicated statistical concepts primarily through visual examples.

To make this project, I used JavaScript, React, HTML, and CSS. Due to the limitations of popular animation libraries, the visual animations are manually programmed using the user's scroll height. To evaluate my project, I spoke to 10 potential users, including people from my Independent Work seminar, and took notes on both their experiences using and learning from the tool. The research, design, implementation, and evaluation processes are detailed in the following sections, along with potential future work on the current tool and potential extensions upon the broader idea of visual Statistics education.

2. Related Work

2.1. Related Statistical Work

This project was inspired by a Google PAIR tool.[1] This "explorable" looks into how researchers measure fairness in Artificial Intelligence and Machine Learning models [16]. Specifically, it looks into how one can tell if a model performs equally well when used on different types of people. This "explorable" has the same scrolling-slide system as my tool, with a grid representation of the population as a visual tool. Seeing this, I was able to improve upon their design and adapt it for teaching statistical concepts. I will discuss this design process in greater detail in Section 3.

I gathered information regarding type I and type II errors, as well as for the rest of the content of the tutorials, from a handful of different sources. For the Basic tutorial, I consulted the glossary of Split.io. This website gave easy and intuitive definitions that I paraphrased for my explanations. Specifically, I was alerted to the existence of the positive and negative predictive values, which I posit in the tutorial as the most meaningful metric to use to interpret diagnostic test results [3]. While this glossary had definitions that were easy to read and interpret, it didn't include any examples with

[1]PAIR is a division of the tech company that focuses on ethics, morality, and equality in AI models, created by Google and other entities.

real numbers. Because of this, I was able to expand upon their definitions and add my own examples so that users would be able to grasp the concepts and definitions both abstractly and concretely.

To develop the the Advanced tutorial, I consulted two main sources. First, I used an article by Pritha Bhandari to get an idea of the concept and to understand how an introductory tutorial would present the information [17]. This article also had a number of good examples that I consulted when I was integrating the lesson on type I and II errors into my running example. Bhandari's article was helpful, but it did not provide a sufficiently advanced explanation of the concept; for this, I turned to a paper by Banerjee et. al. published through the National Library of Medicine [11]. This paper had in-depth information about hypothesis testing and both types of errors, far more than I could wish to include in my tutorials. However, reading this paper and digesting the information was the most effective way for me to teach the concepts through a different medium: because my knowledge base was now large, I could work to distill that information down to the most important and applicable ideas to give to my users.

The final relevant piece of related statistical work was a CDC study by Brihn et. al. on the performance of a COVID-19 antigen test [12]. This study attempted to determine the test's sensitivity and specificity based on empirical data, finding that the test had a very high specificity but a relatively low sensitivity. I chose to include this study in the tutorial because I found it to be relevant to users' lives, as the COVID-19 pandemic has significantly impacted so many. This study would put the running example from the tutorial in context and provide a vote of confidence in our public health system, given that the numbers in the example, while clean, do not shine well on the test being used.

2.2. Tools and Libraries

The most important tool that I used is React, an open-source rendering tool created by Meta [9]. This tool allows for the specification of custom HTML elements through JSX, which allows for the writing of HTML code directly into JavaScript code. Further, React allows for the dynamic updating of elements, such that the entire page does not have to be re-rendered for one single element to change. This makes smoother, lighter applications that feel responsive to the user. Learning to use

React, at least in the capacity that I used it, was not difficult: there is extensive documentation and it is rather intuitive. Further, the developers have set up a start framework called `create-react-app`, through which someone with no experience with React can stand up a local development server without writing a single line of code. For someone like me, for whom setup is often the most complicated and difficult task faced when completing a project, this tool was very useful. This, paired with the other documentation on the React website, made starting development smooth and easy.

The next tool that I installed Nodemon, a no-setup package that will automatically reload your server when you save code to file [5]. The benefits of this tool are self-explanatory: not having to manually restart the server from the terminal after every change vastly speeds up the development process. Further, it makes it much easier to make small formatting and layout changes, as there is no need to stop to use the terminal in between editing code and viewing results in the browser. This tool is a must-have for any web development project that uses Node.

Another tool that necessary for completing this project was MathJax, specifically an updated version called better-react-mathjax [1]. This package parses basic LaTeX commands and renders the text as it would be rendered by LaTeX. This functionality was imperative to my project, as both in-line and stand-alone equations made many appearances in my designs. This package defines a React component with a parsing functionality: It parses strings intended to be displayed as simple HTML text and finds any instances of LaTeX commands. One of the downsides of MathJax, however, is that JavaScript and LaTeX both use the character '\' to mean different things: specifically, it denotes an escaped character in JavaScript and the start of a command in LaTeX. Therefore, any LaTeX command must start with an escaped backslash, '\\' in order to be correctly parsed by MathJax.

Different browsers have different standards for how they interpret JavaScript, HTML, and CSS. Two of the most popular browsers, Google Chrome and Apple Safari, interpret scrolling differently: Chrome supports 'smooth' scrolling, while Safari does not.[2] This meant that Chrome users would

[2]Safari *does* in fact support smooth scrolling, it just requires more code than Chrome: the Google browser only needs a JSON object passed to the `scroll` function with the behavior property set to `smooth`.

have the page nicely transition from slide to slide, while Safari users were faced with an abrupt jump. To solve this issue, I used poly-fill code from Dustan Kasten [14]. A poly-fill is a way of converting all instances of a given function to instances of a similar function without the source programmer having to do any manual work. This is often needed when considering compatibility with older versions of browsers and JavaScript. However, this specific poly-fill took a single line of Chrome-specific smooth scrolling code, wherever it appeared in my code base, and converted it to Safari-specific smooth scrolling code, when necessary.

The last main tool I used was not a neatly bundled package, but some code from a tutorial on CodePen. This code was originally written by Eric Terwan at MIT, and it is the code that displays and controls the "Hamburger Menu" in the top left corner of the page [7]. This is code I have used in the past, and both the code and the output the code produces are easy to use and adapt. While the functionality of this code in my code base comes directly from Mr. Terwan, I updated some of the styling and positioning (as well as the content of the menu, of course) in order to better fit my project.

Part of my research process was dedicated to finding a pre-packaged animation library, but I was not able to find one that fit my needs. One library that I considered was react-spring, an extremely well-regarded animation library for React [8]. This library bases its animations on the properties and motion of a spring. These manifest as intuitive, sinusoidal animations that offer fresh, interesting alternatives to standard linear animations. With countless combinations of options available, it seemed like the perfect way to animate the visual aspects of my tool. However, there was a glaring omission from every popular animation library, including react-spring: the ability to animate based on scroll height. I could not find a package that created animations as a function of scroll height so that the animated objects would change evenly as the user scrolled. It seems simple enough: based on my experience creating this tool, there is only a small amount of work needed to convert time-based animations into scroll-based animations. However, I could not find a good package that would accomplish this for me, so I had to manually animate every transition.

3. Approach

3.1. Global Design Considerations

In designing this tool, I aimed to be simple and elegant and to create a tool that was easy and intuitive to use. I wanted the user to know exactly how to get around the website with little or no prompting, and I wanted the colors to be contrasting but easy on the eyes. Ideally, every aspect of the design of my tool would serve the user's experience in learning the material, without any extra frills.

When organizing the content of the tutorials, I considered the Principle of Chunking, as defined by Lidwell et al. [15]. I employed this in two ways: the first is chunking in the text section itself, both deciding what information goes on which slide and how to divide up the information on a given slide. My goal with this set of chunking was to give information in digestible pieces. I also placed a text chunk and a visual chunk on every slide in order to promote engagement and connection for the user. I found in many Statistics tutorials that the ratio of paragraphs to visuals is incredibly high, leaving learners to drown in a sea of text. I wanted the user to be able to learn equally from reading and visualising the math. This is known as the Picture Superiority Effect, which Lidwell et al. concisely define to mean that "pictures are remembered better than words."

I also heeded the advice of Lidwell et al. when it came to distinguishing keywords and section headers. The authors discuss the idea of Highlighting, and how bolding text is more effective than underlining or italicizing it. I also chose to use boldface over changing the color of the text in order to maintain simplicity.

As stated in the Section 2, I drew inspiration from the Google PAIR tutorial on measuring fairness. However, I saw a few ways to improve upon the design in order to keep students more engaged and provide a more entertaining and informative experience. The most important aspect was to target multiple audiences with multiple levels of tutorials. Rather than aim to have one tutorial be accessible to the entire population—and risk having it be either too complex or too simple—I divided the tutorial into three sections. This way, people with less statistical experience would not feel overwhelmed by a difficult tutorial, and those with more experience would still be able to learn something from using the tool. This also meant that there was more total information provided, so someone could learn a great amount of fundamental Statistics in one place. Ideally, the tool is designed in a way that many types of users—struggling high school students or college students, exceptional elementary or middle school students, or curious adults—can learn from it.

The next aspect that could use improvement was the transitions between states in the visual portion of the tutorial. In tools like this, there are multiple ways to transition between ideas, each with a unique set of benefits and drawbacks. A slideshow is the most basic of these, where each idea is on its own "slide" and the user clicks to trigger an instant transition to that slide. This is easy to code and familiar to the user; however, it does not command the user's engagement like other methods. The slideshow can be upgraded slightly by making smooth transitions from slide to slide: this increases engagement, but does not string the ideas together very well. One way of accomplishing this task is to keep the visual aspect on the screen the entire time: indeed, the PAIR tool did just this. This means that the user sees a connection between every idea and its predecessor in the form of the visual element. When exploring the Google PAIR tool, I noticed that the visual changed entirely after passing a specified point on the page using a timed animation. That is, letting this point on the page be x units from the top of the page, the visual would be in one state at $x-1$ units and a different state at $x+1$ units, with no ability for the user to control that animation. While this has no bearing on the content of the tutorial, I believe that it makes the experience less in-depth and interactive, making the user less engaged. Because of this, I settled on a scroll-based animation system that would keep a single visual element on screen for as long as is pertinent to the example.

The final aspect that needed consideration was navigating the page. Normally, a website has many pages, so intra-page navigation is done with simple user scrolling and inter-page navigation is done with buttons and links. However, I chose to make a single page site in order to make the user feel like they were always performing the same task: advancing from tutorial to tutorial was no different than advancing within a tutorial, so the barrier for entry into the harder tutorials should be lower. Therefore, it is only necessary to navigate only page. In the PAIR tutorial, user scrolling was the only way to advance from slide to slide. This meant that the user had to work to get the page in a state that they were comfortable reading and learning from and it meant that the site could enter states where elements were misaligned, had strange opacity, or were partially cut off more easily. To improve upon this, I made my tool more explicitly slide-based, with multiple different advancement options. Manual scrolling is still possible, but discouraged. The main ways to navigate are using the up and down arrow keys, as well as the clickable up and down arrows that are always on screen. This way, the user does not have to worry about arriving in a confusing state of the site. Further, it more explicitly denotes which text belongs to which state of the visual, which also includes equations, graphs, and tables, expanding on the definition of "visual" used in the PAIR tutorial.

3.2. Local Design Considerations

While the majority of the time spent on this project was focused on coding, the user's experience is spent mostly on reading the textual part of the tutorials. Because of this, I took care when designing the tutorials, focusing mostly on creating understandable, keyword focused tutorials that closely matched the accompanying visuals. Below are discussions of the writing process for each of the three tutorials.

3.2.1. Basic Tutorial For the Basic tutorial, I focused entirely on the providing the most utility to the user while assuming as little mathematical and statistical knowledge as possible. The goal for this section was to allow for any adult to be able to garner useful information in a small amount of time. Because of this, I spent time choosing the values for the population, sensitivity, and specificity that would create the simplest example for the user. I also did my best to assume no prior knowledge and explain each new term in as plain language as possible. Finally, I wanted to leave the user with some information that they could use in assessing test results in their own lives with the Positive and Negative Predictive Values, the primary focus of this tutorial. Again, these values were meant to be easily understood as well as useful to the user.

3.2.2. Intermediate Tutorial The Intermediate tutorial starts to delve into the origins of the values brought up in the Basic tutorial. While the latter is meant to give people tools to use in very similar situations, the former is meant to give people the tools to use in many more situations. While certainly applicable to the example posited in this tutorial, and to hypothesis testing in general, there are countless other applications of conditional probability and Bayes' Theorem. By teaching them in a rigid context, I aimed to solidify the user's understanding of the concepts so that they could apply them in other settings. In this tutorial, I spent less time introducing the example than in the Basic tutorial, assuming that the user is familiar with the concepts from either previously taking the Basic tutorial or from some other avenue. For both the broad idea of conditional probability and the relatively specific idea of Bayes' Theorem, I first introduce them in a general sense and then I introduce them in the context of the example the user knows. The visual component of the site aids in this: in the conceptual introduction, the visual component is the fundamental equation related to the topic (Equation 1 for conditional probability, Equation 2 for Bayes' Theorem), while in the concrete, exemplary introduction, the visual component is the familiar grid.

$$P\{A|B\} = \frac{P\{A \cap B\}}{P\{B\}} \tag{1}$$

$$P\{A|B\} = \frac{P\{B|A\} \cdot P\{A\}}{P\{B\}} \tag{2}$$

3.2.3. Advanced Tutorial The Advanced tutorial is designed to go beyond the visual examples and put users on track to learn more about advanced, research level Statistics. However, as this is a large category of Mathematics, my goal was to give a good introduction and good opportunities for further reading. In order to maintain a connection to the rest of the site, I still introduce the main visual example and circle back to it by the end of the tutorial; however, the middle delves more into fundamental hypothesis testing. Again, my goal is to be a jumping off point, so I synthesized information from a number of introductory sources about hypothesis testing. I attempted to stick to the goal of the site, so I was as concise as possible and included what I hope to be helpful visuals to go along with the text. Next, I do some real world analysis of COVID-19 diagnostic rapid tests and discuss the results of these real researchers in the context of the toy example to which they have been introduced. Finally, I reconnect with that example and discuss how, with the doctored sensitivity and specificity numbers of the example, the test does not produce statistically significant results.

4. Implementation

Before I begin with the specifics of the implementation, it is important to introduce the concept of DOM trees. Discussing this concept in the general sense will make understanding my use of the DOM tree in the implementation much easier.

The DOM (Document Object Model) tree is the abstract representation of HTML elements on a page. It is organized in the same way as any other tree in Computer Science: there is a root with some number of children, and each of those children can have some other number of children. You can view the DOM tree for this site in the Appendix. There is some DOM specific language and concepts that are key to the following sections: specifically, the notions of a Parent, a Child, a Descendant, the Document, and the Root [13].

A node is called a Parent node if it has any number of child nodes. A node is a Child node when it has a single parent node. A child cannot have more than one parent, but a parent can have any number of children. When discussing pairs of nodes, there is a parent-child relationship if the child node is exactly one level lower than the parent. If there is more than one level of separation between nodes, the higher node is called the parent node and the lower node is called a descendant node of that parent. The Document is the true root of the DOM tree. It has no parent node and one child node (the root). As its name implies, this is analogous to a `.html` document. The root is the only child of the Document, and is equivalent to the `<html>` tag in a `.html` document. The root usually has 1-3 children, the header, body, and footer (or some subset of the three).

4.1. Basic Elements

Because I designed the website to have only one page, I needed a way to organize and differentiate between the header slide and each tutorial. Further, I needed a way to organize the content inside each tutorial in a standardized, intelligible way. I decided to create a number of customized HTML and React elements with standardized styling to create this organization.

The first element that the user sees is the header, which contains the title, subtitle, site explanation, and navigation arrows. As this element is displayed only once, it can have very specific styling. Further, as each child element only exists in one place, they can also be specifically styled. This header slide was specifically built to be viewed by all users.

The rest of the site consists of three custom-built React components with the same basic structure. The parent element of all three components is a basic body element, which is styled very generally to encapsulate the spacing and sizing ideas for the tutorial internals. This body has two main children: the visual—discussed in depth in Section 4.2—and textual components. The textual component for each tutorial has a variable number of tutorial sections, which each have a variable number of paragraphs. These paragraphs are the chunks of text discussed in Section 3.1. As the visual and textual components exist in every slide, it might seem like it would be a good idea to have a certain number of slide elements, each with a textual child and a visual child. However, because the visual transitions are animated, I needed to raise the visual component outside of the slides to be a direct child of the tutorial itself.

As stated above, each slide has a textual child. This text element is generally styled so that each and every slide has the same text formatting and style. Further, I only need to update the CSS code in one place in order to change the styling for every slide, which saves a ton of time. Inside each text element are the text paragraphs of the slide, all wrapped in a MathJax React component. I stored the text itself in a separate file in order to avoid clutter in the main files of my app. This textual content is stored in a dictionary and imported into the tutorial files.

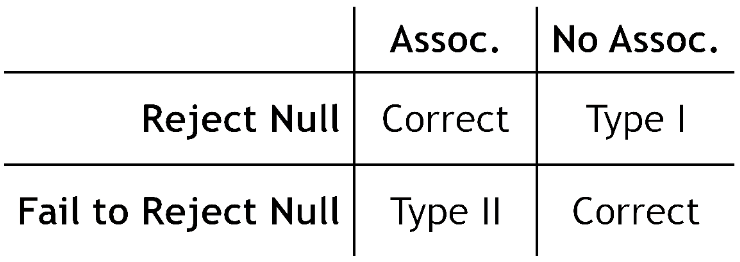
One feature of CSS that I tried to take advantage of is the "Cascading" nature of the language (the 'C' in CSS). The cascading property of CSS means that a style property defined for an element is automatically applied to all of its descendant elements unless explicitly overwritten. In practice, I could define a property at a high level in the DOM tree and have it apply to many elements in the tree, or I could define a property at a low level in the DOM tree and have it apply to only a few elements. I took advantage of both strategies in different places on the site: for the header, properties like the larger header text size and the centering of the text were defined at a low level because the styling was specific to the header and would not appear in other places. On the contrary, properties like text color and size were defined at the root of the DOM tree, such that they were automatically applied to each and every element on the site. This meant that I only had to change them in a few places (like in the header) rather than specify the same property in every element except for those in the header.

4.2. Visual Elements

There are a handful of elements that make up the visual portion of the tutorials. The most important is the grid representing the sample population used throughout all 3 tutorials; however, the other visual elements were also necessary to producing an engaging tool.

First, consider the simplest set of elements: the equations used in all three tutorials. As you can see in Figures 6, 7, and 8 in the Appendix, the math is contained in a `div` element. It consists of a MathJax React component for every equation necessary for that tutorial, with the equation inline in the HTML code. For the Basic tutorial, both equations were displayed at the same time; however, for the Intermediate and Advanced tutorials, each equation needed to be displayed at a different stage of the tutorial. To accomplish this—that is, to make sure only the correct equation was displayed at any given point of the tutorial—I used the display property of a `div` element. Setting the display property of an element to `none` means that element is not rendered on the screen at all: it does not even take up space. This is very convenient when working with multiple elements of similar size that are children of the same parent element, like these equations: as long as only one of them has a display property set to `auto` rather than `none` at all times, I could style the parent element like it had only one child.

Figure 8 shows that the math component in the Advanced tutorial also contains an HTML table element (the final version of this table can be seen in Figure 1). Ideally, this would have been a LaTeX table in a MathJax component in order to match the styling of the other mathematical



	Assoc.	No Assoc.
Reject Null	Correct	Type I
Fail to Reject Null	Type II	Correct

Figure 1: A table depicting how type I and type II errors are made. This table is color corrected for this paper (on the website, the text and dividing lines are white against a navy background).

visual elements, but MathJax does not support tables, as it is designed to handle only inline LaTeX commands.The majority of the time spent coding this element was focused on styling, as there are many ways to place dividing lines in an HTML table. Ultimately, I settled on specifying cell borders based on their position in the table. For example, the cell in the top left of the table needed borders on the bottom and right sides of the cell (but not the top or left side of the cell). Therefore, I set the border width property for that cell to be `0 0.5vw 0.75vh 0`.[3] This means the right and bottom edges of this cell have a border weight of 0.5% of the width and 0.75% of the height respectively,[4] while the left and top edges have a weight of 0. Once I learned how to style the table in this way, finishing the table proved simple and styling and positioning the table in the larger site was effectively the same as it was for any other element.

The final auxiliary visual element was an image representing an acceptable p-value in a null hypothesis test [10]. This image needed some preprocessing, but once that was completed, positioning was equivalent to other elements. That preprocessing was done in Apple Preview [6] and consisted of stripping the image of labels, changing its color, and sizing it to fit correctly on the page.

Figure 2 shows the main visual element of my site. This grid represents the population that is

[3] CSS labels sides of a rectangular element like a compass: North comes first, followed by East, South, and West. It may be helpful to think of the popular mnemonic device "Never Eat Soggy Waffles," or your regional equivalent.

[4] the unit of measurement `vw` means "visual width": `1vw` is equivalent to 1% of the width of the user's screen. Similarly, `1vh` ("visual height") is equivalent to 1% of the height of the user's screen. I chose to use these units in order to promote usability on all types of screens.

used for examples throughout all three tutorials. I chose 1600 avatars because it is a square number that plays well with values for sensitivity and specificity that were both realistic and resulted in simple calculations. Especially for the beginner tutorial, it was paramount that the calculations were as simple as possible—while maintaining some semblance of realism—so that users would not be deterred by complicated decimals or fractions. A population of 1600 was the perfect sweet spot because it represented the most manageable population that would still result in a realistic example for the users to work through and apply to the real world.

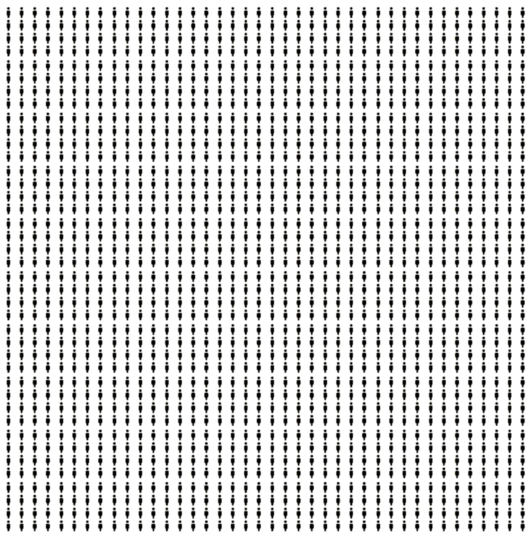


Figure 2: The full, 1600 avatar grid. It is comprised of one hundred 16 avatar cells, and, like Figure 1, is color corrected for this paper.

This grid is comprised of 10 rows and 10 columns, with every cell having 16 avatars. I originally had 40 rows and 40 columns, which meant that each cell had a singular avatar. While this certainly allowed for more flexibility, I never needed to move fewer than 16 avatars at a time (save for transitioning to the 'About' section, which I will touch on later). Further, moving fewer elements is much more efficient: even basic transitions were slow when I had a 40 by 40 grid. The 10 by 10 grid meant that, at every user scroll input—using the on screen arrow keys, the keyboard arrow keys, or the scroll wheel—at most 100 different elements must be considered, a 16$\times$ reduction.

The internal structure of the grid also helped in making transitions more efficient. Consider

Figure 9 in the Appendix: the grid is broken first into 10 rows, which are each broken into 10 cells. This means that when I needed to move an entire row of cells rather than specific cells inside that row, I only needed to consider 1 row rather than all 10 cells. While this resulted in a further 10× reduction, it occasionally also meant that there were both cell and row transitions applied at the same time. This structure also meant that moving the grid was simple, even in a complex state, as I only needed to move the grid element and all child elements would move along with it.

The only other serious pitfall of this grid architecture was the final transition, from all 1600 avatars to a singular avatar, representing me, the creator of the project. If I had used a 40 by 40 grid, this transition would have been very simple: specify the correct row and column in the grid and translate the element appropriately. However, as performance was important to making this a usable tool, I opted for a more complicated final transition that was faster overall. This transition, which I will discuss in depth in Section 4.4, involved hiding a single avatar behind the grid until the time came to keep it on the page while the grid left the screen.

4.3. Hamburger Menu

The hamburger menu gets it is name from it is unique shape, which can be, albeit abstractly, interpreted to resemble a hamburger. The DOM Tree representation of this menu can be found in Figure 5 in the Appendix and the code for this menu was adapted from a pure CSS hamburger menu developed by Erik Terwan in 2015 [7]. This menu utilizes a hidden checkbox element, the state of which Terwan uses to change the position of the menu. This hidden checkbox has the visibility property set to `hidden`, which means that none of it is visible features are rendered on screen but it still takes up space. This is crucially different than having the display property set to `none` due to the fact that Terwan's element still takes up space on the page. He leverages this feature to layer it on top of the visible menu button, so that, when the user clicks on the menu, they are actually checking or un-checking a checkbox.

Terwan uses a small set of animations to transform the top and bottom spans of the menu into an ×, which is used as the 'exit' button for the menu. This transformation is in place, so the hidden checkbox remains in the same location and is un-checked when a user clicks on the ‘exit’ button.

When the checkbox is in the ‘checked’ state, the menu is displayed on screen. This menu is a simple list of link elements with custom text embedded (see Figure 5). Each of these links, rather than connecting to a separate page, call an `onClick()` function that prompts the site to scroll to a new part of the page. As stated in Section 2.2, Terwan created this menu using only CSS and HTML. However, I converted that code to make this menu a React component to more seamlessly fit into the flow of my site. Because of this, I was able to pass the appropriate heights in as props. Props are analogous to arguments passed to a function: the component is written without knowing the values of these props, and they are inserted at runtime [9]. This way, I could determine the appropriate scroll destination in the main App file and would have to worry about keeping and updating constants in multiple files.

Finally, I added a drop shadow to the menu such that it appears as if the menu is sliding out on top of the rest of the page. Given that the rest of the page is designed to be entirely 2-dimensional, this small bit of depth creates an impressive effect.

4.4. Animations

The crux of this project was transitioning from slide to slide. In order to differentiate my tutorials from other widely accessible ones, I needed to keep the user engaged. The static visual elements discussed in Section 4.3 were designed to do this, but most engagement was intended to come from the animations. In Section 3.1, I discussed the pitfalls of the Google PAIR tutorial; now I will discuss the implementation of those design improvements. To do this, I will divide the animations into a few categories: core grid transformations and other, miscellaneous transformations.

4.4.1. Basic Framework Before digging into specific animations, however, it is important to discuss the basic framework for transitions. These transitions are managed by a master function, `onPageScroll()`, which delegates to nested auxiliary functions depending on the user’s scroll height, h_s.[5] The first layer of nesting, which is in the main `App.js` file, delegates based on coarse

[5]the user’s scroll height is defined as the number of pixels between the top of the user’s current window and the top of the page. If the user is at the top of the page, $h_s = 0$; if the user has scrolled 100 pixels from the top of the page, $h_s = 100$. It is important to note that this value is always positive, even though the user is scrolling "down".

location: it uses a switch statement to determine if the user is on the home page or in the Basic, Intermediate, or Advanced tutorial.

This switch statement was a surprising sticking point in the development of this tool. In JavaScript, switch statements compare only equality, which works well when a switch statement is needed for numbers or strings. However, if a switch based on an interval—the interval of heights that represent a single slide, in our case—is needed, it gets tricky: it would be completely senseless to define every value in that interval as a different case and have them each run the same code. Because the switch statement compares only on equality, it is necessary to determine some way to set up an equation such that one side is a boolean statement that is true when h_s is in the correct interval, and false otherwise. For this outer switch statement, those intervals are the tutorials:

$$\text{Home Page: } 0 < h_s \leq h_{\text{basic}} \tag{3}$$

$$\text{Basic tutorial: } h_{\text{basic}} < h_s \leq h_{\text{intermediate}} \tag{4}$$

$$\text{Intermediate tutorial: } h_{\text{intermediate}} < h_s \leq h_{\text{advanced}} \tag{5}$$

$$\text{Advanced tutorial: } h_{\text{advanced}} < h_s \tag{6}$$

Indeed, that reveals the answer: switching on the boolean value `true`. Then, the cases of the switch statement are the boolean statements in Equations 3-6. I further simplified the cases by taking advantage of the structure of switch statements: cases are considered for equality in the order of appearance in the code. By breaking at the end of every executed case, only one will ever be executed. That means it is only necessary to check the upper limit for h_s: as there are no gaps in the intervals, it must be true that h_s is greater than the current interval's lower bound if the current case has been reached.

Once the main file has determined which tutorial the user is viewing, it calls a function imported from the file specific to that tutorial. This function is called `onPageScroll*()`, where `*` is the name of the tutorial.[6] The main function passes the user's scroll height, as well as pointers to

[6]If you are reading the source code along side this paper, note that the names 'Intermediate' and 'Advanced' are replaced with 'Inter' and 'Adv' in the code for brevity.

the elements each tutorial uses. I chose to initialize a pointer to every element on page load for performance: if, for example, I reinitialized a pointer to an element every time I needed to use it, the browser would be destroying and recreating pointers left and right. This way, a singular pointer is maintained throughout the user's time on the page.

This nested `onPageScroll*()` function employs a similar switch statement to the one previously discussed, but, rather than the intervals being coarsely defined as tutorials, they are finely defined as 'slides'. Each slide is exactly the height of the user's window.[7] Each case in this switch statement calls a unique transformation function, passing in the user's scroll height and any relevant element pointers. Those transformation functions all fall into one of two categories, which are each discussed below. However, they all operate according to the same basic idea: use h_s to find a value $\varepsilon \in [0,1)$, where $\varepsilon = 0$ when the user is exactly at the start of the slide (i.e. the slide is centered on the user's screen) and $\varepsilon \to 1$ when the user approaches the end of the slide. Then, move elements according to ε and a constant factor.

4.4.2. Core Grid Transformations The majority of the slides feature transformations from one state of the grid to another. To accomplish this, we first need to find the appropriate cell(s) or row(s) of the grid component and then move them to the correct position using ε. As discussed in Section 4.2, it is always more efficient to move a row than 10 cells, so I always moved a row, if possible. There are multiple possible ways to move an element in CSS: either move its actual position or apply a translation transformation. I chose to apply a transformation because it meant that I could always maintain an idea of the grid's original position: therefore, if I needed to reset the grid (like at the beginning of a new tutorial) I could just reset the transformation to 0. Transformations in CSS are relatively simple, but they require string concatenation. To create a transformation, you add a string to the transform property of an element. This string contains as many type-value pairs as the programmer wants, where the type is the type of transform (translateX, translateY, rotate, etc.) and the value is the magnitude of that transform.

Sometimes, I also needed to change the background color of a cell or set of cells. To do this, I

[7]this value is easily accessible as the `innerHeight` property of the standard window object.

used the background-opacity and background-color properties of an element. If the background was initially empty, I initialized a final, constant set of `rgb` values and set the opacity based on ε, such that the color would fade in over the course of the transition.[8] On the other hand, if I was changing colors in a transition, I used ε in order to change the `rgb` values themselves over the course of the transition, holding the opacity constant.

4.4.3. Other Transformations Sometimes, I needed to move the grid off of the screen for one or multiple slides so that it could be replaced with one or multiple other visual elements (see Section 4.2 for more on these auxiliary elements). These animations proved relatively simple compared to the grid transformations. The first reason is that I did not have to worry about moving a specific set of cells or rows: I just needed to move the entire grid. Second, I did not need to worry about moving the grid in the X direction, only in the Y direction. Finally, computing how far the grid needed to move was also very easy: I needed to move it exactly $\varepsilon \cdot h_{\text{window}}$, where h_{window} is the height of the user's window. These same principles also held for the auxiliary visual element that was replacing the grid on screen: the only difference was that it needed to start off screen and transition to having no transformations.

The final transition to discuss is the transition from the end of the Advanced tutorial to the 'About' section. This is unique because it "hides" an avatar behind the grid, which then pops out of the grid and becomes larger. To implement this transition, I created a single avatar with a background that matched the background of the page. I kept this element hidden using the visibility property until the last moment—seven-eighths of the way through previous transition—when I displayed the single avatar and it was camouflaged by the avatars in the grid. Then I simply moved the grid off screen using the same principles as above, while keeping the single avatar in place. Because the visual element has its position property set to `absolute`, this required no code and the avatar just stays in place as the rest of the page scrolls around it. Finally, I applied a scaling transformation using ε so that the avatar would slowly grow from the size of the avatars in the grid to a reasonable size on it is own.

[8] I chose to fade from 0 to 0.5 opacity so that the transition would be noticeable and aesthetically pleasing but the final color would not be distracting.

4.5. Server and Heroku

One important technical aspect of this site was making it available to the world. During development, I hosted the site on my local machine: the code ran locally, and was accessed by the browser using the `localhost` IP. This process was very simple, made easier by a descriptive tutorial tailored towards making a React app [2]. Effectively, this process consists of telling React what you want to render: in this case, I made the entire site a custom React component, so that all React needed to render was that component. This made development quick and easy, especially when I started using Nodemon. However, there was no infrastructure set up so that users on different computers could access my app.

Heroku is a hosting service for small to mid-sized apps [4]. It follows a "freemium" account structure, allowing users to host apps on their servers for free, as long as they agree to slower boot speeds and lower server priority. For a small research project like this, this free option is perfect; in the case that I choose to scale this product after the semester, I will either upgrade my Heroku account or move off the platform altogether.[9] React is such a popular framework for building sites that Heroku has a large amount of support for React apps. Indeed, there is even a no-setup way to create a React app through Heroku that makes it very easy to host on Heroku. All that is required is creating an app on the Heroku site and connecting it to the source code via GitHub. From there, Heroku can auto-deploy or manually deploy a given git branch, and it automatically deploys it as a production version! This is one of many reasons why React is a beginner-friendly framework.

5. Evaluation

In order to evaluate the quality of my tool, I chose to both consider my goals at the beginning of the semester and ask others to use the product without having used it throughout development. The former, which I will call Self Evaluation, consisted of taking a microscope to my work while rereading my project proposal from February. The latter, User Evaluation, consisted of two rounds:

[9] Heroku makes money off of people overpaying for the premium features: these same features are available for cents on the dollar at companies designed for larger apps, like Amazon Web Services.

a focus group style session with fellow Computer Science majors in my Independent Work seminar and a broad call for users of all types to evaluate my project using a Google Form.

5.1. Self Evaluation

Looking back at my project proposal, the final set of tutorials is a good proof of concept for the tool I laid out. My goal was

> to provide a number of simple, intuitive tutorials about type I and type II errors, accessible to all levels of statistical familiarity. While using one continuous example, I want to be able to educate those with little to no mathematical experience, those with high school math experience, and those with a college-level of statistical knowledge.

I believe that my final product lives up to this goal, save the end of the last sentence: upon review of the Advanced tutorial, I do not believe it is quite as "advanced" as the proposal suggested. The Advanced tutorial is more advanced than the Intermediate tutorial, but it does not quite live up to the "college-level of statistical knowledge" that I strove for. However, this does not mean the tool is a failure: in fact, this only means there is more room for the project to grow and evolve. The rest of the tool does what I intended: I believe that it is accessible to all users with a basic understanding of fractions and percentages, with paths for as much or as little learning as the user wants. Further, I believe that the tutorials are intuitive and easy to read, use, and understand.

In terms of the technical side of the tool, my self-evaluation is more critical. The tool works very well on screens with similar sizes to my own, but starts to fall apart when a user with a different size screen (a tablet, a phone, or even a monitor or TV) attempts to use it. I have spent time attempting to mitigate these problems, but it is a difficult problem to fix. I will discuss the process of making these improvements in Section 6.1.

5.2. User Evaluation

As stated above, the user evaluation consisted of two phases: a focus group of my seminar mates—all Computer Science majors with backgrounds is the technical aspects of the site—and a set of 6 random users from the University community and beyond, with a wide range of confidence in Statistics. In total, I had 10 users test my product at a thorough level.

5.2.1. Peer Evaluation The focus group style setting was very important to making improvements to the usability of my tool. Because my classmates have similar—if not equal—experience using the development tools that I did, they were able to give specific, accomplishable feedback on the tool. One classmate stated that they "would increase the line height and break long paragraphs into smaller chunks." Indeed, this was a simple fix that made the site much easier to digest, which I had not thought of because I had internalized the contents of the tutorial. Another benefit of the high level of technical proficiency in this setting was reporting bugs with accuracy and specificity. Rather than saying "there were a number of visual bugs that detracted from my experience"—part of a response from a random University Community user—they reported things like "issues with the animations/graphics when the window was resized." As a developer, the latter provides a great starting point for bug fixing, while the former adds little to the process of development.

5.2.2. Other User Evaluation While the broader call for evaluation did mean there were some vague and unhelpful responses, it also meant that I could hear from people more similar to my target audience. While my classmates were very helpful with the technical aspects of the tool, they are all very well versed in Statistics. As upperclassmen in a relatively theoretical and mathematical Computer Science program, they might just barely fall in to the set of users who would benefit from using the Advanced tutorial. While they did their best to follow the spirit of the game and put themselves in the shoes of a more targeted user, it is very difficult to forget one's statistical knowledge for 15 minutes. While my classmates were very helpful with the technical and visual components of the tool, these members of the University Community were helpful in making the content clear and easy to follow. One tester suggested rewording the explanation on the "home page" of the site in order to make it more clear how to navigate the website, while another suggested taking more time to define notation in the Intermediate tutorial. Both of these issues, although minor and simple to fix, were overlooked by myself and my classmates because of our experience with both developing and using similar sites and formal mathematical notation. Another tester suggested labeling the outlined avatars in some sections of the tutorials. This was another minor technical change that I believe adds a lot to the site.

6. Future Work

Although the current state of this tool represents a good proof of concept, there are many areas upon which could be improved. In addition to changes to the tool itself, the process of creating this set of tutorials has also shed light on some future work in adjacent areas of Computer Science and Statistics. In this section, I will discuss potential future work both in further developing this tool in order to more effectively teach Statistics and in making the process of creating similar tool easier for future programmers.

6.1. Relative Sizing, Positioning and Animation

The most glaring and immediate piece of future work is rewriting the animation code to move things relatively rather than absolutely. Currently, the site is styled in a hodge-podge of relative and absolute units. This means that, for those using a screen with different dimensions that mine, the animations are imperfect. This problem stems from the fact that I was inexperienced with using either form of measurement, and defaulted to absolute units because they felt most natural. While I attempt to use relative units by storing and using the height of the users window, this effort proved insubstantial to users who still faced issues.

In order to fix this problem, I spent time looking through the code base in order to diagnose it. Early observation shows that the way I define and use the height of the user's window should cover animation specific sizing and positioning: that is, the animations should be acting in a relative manner. However, upon closer examination, I found that all animations were based on the height of the page alone, rather than both the height and the width of the screen. While this was by design, it means that any horizontal scaling of the site mean that the visual animations do not change in magnitude. After this examination of the code, I spent time re-working the sizing and positioning of some elements. The site is more usable, but the more glaring issues still persist.

Once I am confident in my solution to the problems in Google Chrome, I still might face issues when users attempt to access the site from other browsers. While I attempted to mitigate this effect during development, there still may be other issues yet to be uncovered. This will likely require more poly-filling, either by hand or through packages like the one provided by Mr. Kasten [14]. However, as I know that this is not the main source of current sizing, positioning, and animation bugs, it seems unproductive to consider before solving the known issues.

6.2. Auxiliary Tutorials

In order to make this site a more effective place to learn Statistics, I may choose to develop similar tutorials in other areas of Statistics or Mathematics. One of the stretch goals in my project proposals was to include an example of such a tutorial on Simpson's Paradox; however, creating the animations took longer than I thought, and I wanted to produce a smaller set of fleshed-out tutorials rather than a larger set of poor-quality ones. However, these other tutorials could span many topics, from Simpson's Paradox to Statistical Bias. In designing these tutorials, I would try to keep the main goal of the original tool in mind: I would balance visuals and explanations, attempting to have the latter serve the former rather than the other way around. This would also allow the hamburger menu to become more effective: it is currently included as a nice-to-have, something that users would not sorely miss were it to remain unimplemented. However, it serves a a sign of things to come: creating more tutorials would mean more links, pages, and sections in this menu.

6.3. Scroll Based Animation Library

As mentioned in Section 2.2, one of the most surprising parts of research was not being able to find a Scroll-based animation library. Such a library would have made this project much easier, and my tool may have been able to include auxiliary tutorials. Creating such a library would benefit other users intent on making a similarly styled website, where animations are based on scrolling. Further, there might be other developers who would like this functionality in sections of larger sites. In order to create such a library, I could start from my code for this project: much of the required logic is contained in my work. However, the important step would be abstracting my existing code to fit any situation: with a set of functions, I would need to cover a vast majority of possible transformations.

Another important consideration is the scope and platform in which my library exists. For example, if it is a standard HTML or CSS package, it would be easy to write and very portable, but likely not very powerful. On the other hand, if I built a wrote a React specific library, it would be less portable but likely more powerful, as I could take advantage of React's partial re-rendering [9].

7. Acknowledgements

I would like to thank Prof. David Walker for teaching an extremely useful seminar and seeming to have the answers to my questions before I needed to ask them; thanks must also be extended to my classmates in his seminar, who consistently gave helpful and constructive feedback on my project, from start to finish. I would also like to thank Jérémie Lumbroso for inspiring my love for using technology for education, in any and all forms. Although I no longer work with Jérémie, his influence is still very much felt. Lastly, I would like to thank my parents, Jim and Karen, for opening endless doors so that I could be here and work on such a project.

This paper represents my own work in accordance with University regulations.

Alex M. Baroody '23

References

[1] "better-react-mathjax." [Online]. Available: https://www.npmjs.com/package/better-react-mathjax

[2] "Create React App." [Online]. Available: https://create-react-app.dev/

[3] "False Positive Rate | Split Glossary." [Online]. Available: https://www.split.io/glossary/false-positive-rate/

[4] "Heroku." [Online]. Available: https://heroku.com

[5] "nodemon." [Online]. Available: https://nodemon.io/

[6] "Preview User Guide for Mac." [Online]. Available: https://support.apple.com/guide/preview/welcome/mac

[7] "Pure CSS Hamburger fold-out menu." [Online]. Available: https://codepen.io/erikterwan/details/EVzeRP

[8] "react-spring." [Online]. Available: https://react-spring.io/

[9] "React – A JavaScript library for building user interfaces." [Online]. Available: https://reactjs.org/

[10] "transparent p-value statistical significance statistics null hypothesis test statistic statistical significance." [Online]. Available: https://w7.pngwing.com/pngs/281/694/png-transparent-p-value-statistical-significance-statistics-null-hypothesis-test-statistic-statistical-significance.png

[11] A. Banerjee, U. B. Chitnis, S. L. Jadhav, J. S. Bhawalkar, and S. Chaudhury, "Hypothesis testing, type I and type II errors," *Industrial Psychiatry Journal*, vol. 18, no. 2, pp. 127–131, 2009. [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2996198/

[12] A. Brihn, J. Chang, K. OYong, S. Balter, D. Terashita, Z. Rubin, and N. Yeganeh, "Diagnostic Performance of an Antigen Test with RT-PCR for the Detection of SARS-CoV-2 in a Hospital Setting — Los Angeles County, California, June–August 2020," *MMWR. Morbidity and Mortality Weekly Report*, vol. 70, 2021. [Online]. Available: https://www.cdc.gov/mmwr/volumes/70/wr/mm7019a3.htm

[13] I. Kantor, "DOM tree," Dec. 2021. [Online]. Available: https://javascript.info/dom-nodes

[14] D. Kasten, "Smooth Scroll behavior polyfill," Mar. 2022, original-date: 2013-08-08T03:41:53Z. [Online]. Available: https://github.com/iamdustan/smoothscroll

[15] W. Lidwell, K. Holden, and J. Butler, *Universal Principles of Design*, second edition ed. Rockport Publishers, Jan. 2010.

[16] A. Pearce, "Measuring Fairness," May 2020. [Online]. Available: https://pair.withgoogle.com/explorables/measuring-fairness/

[17] Pritha Bhandari, "Type I and Type II errors," Jan. 2021. [Online]. Available: https://www.scribbr.com/statistics/type-i-and-type-ii-errors/

Appendix

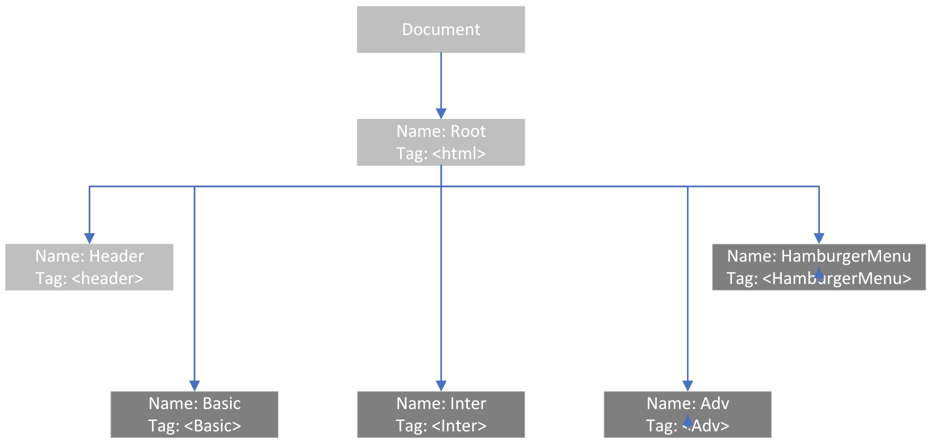


Figure 3: The highest level DOM Tree for this site. Light Grey rectangles are standard HTML Elements, while Darker Grey rectangles represent React Components. The structure of these components are depicted in the following figures.

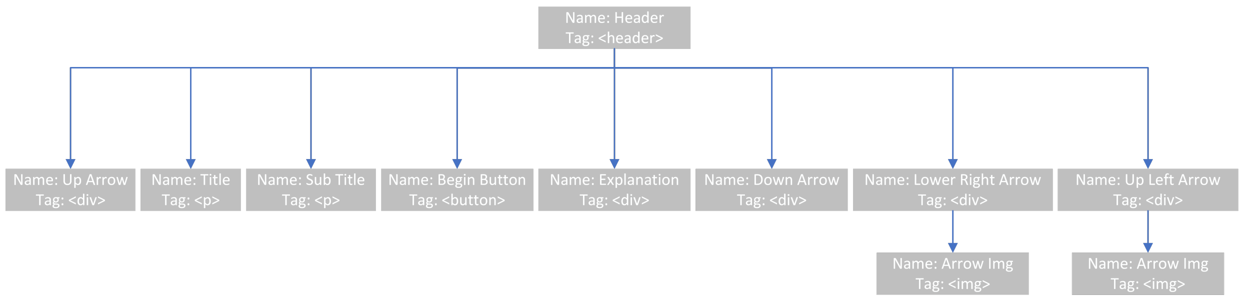


Figure 4: The DOM Tree for the header / home page. This is made entirely out of standard HTML elements.

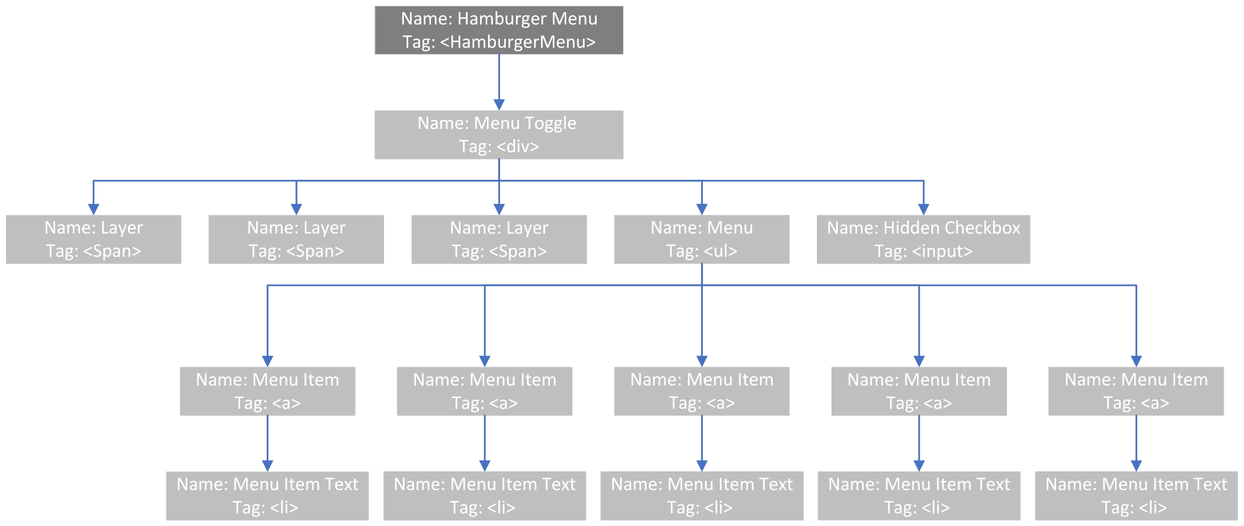


Figure 5: The DOM Tree for the Hamburger Menu. Adapted from a CodePen Project by Erik Terwan [7]. Light Grey rectangles are standard HTML Elements, while Darker Grey rectangles represent React Components.

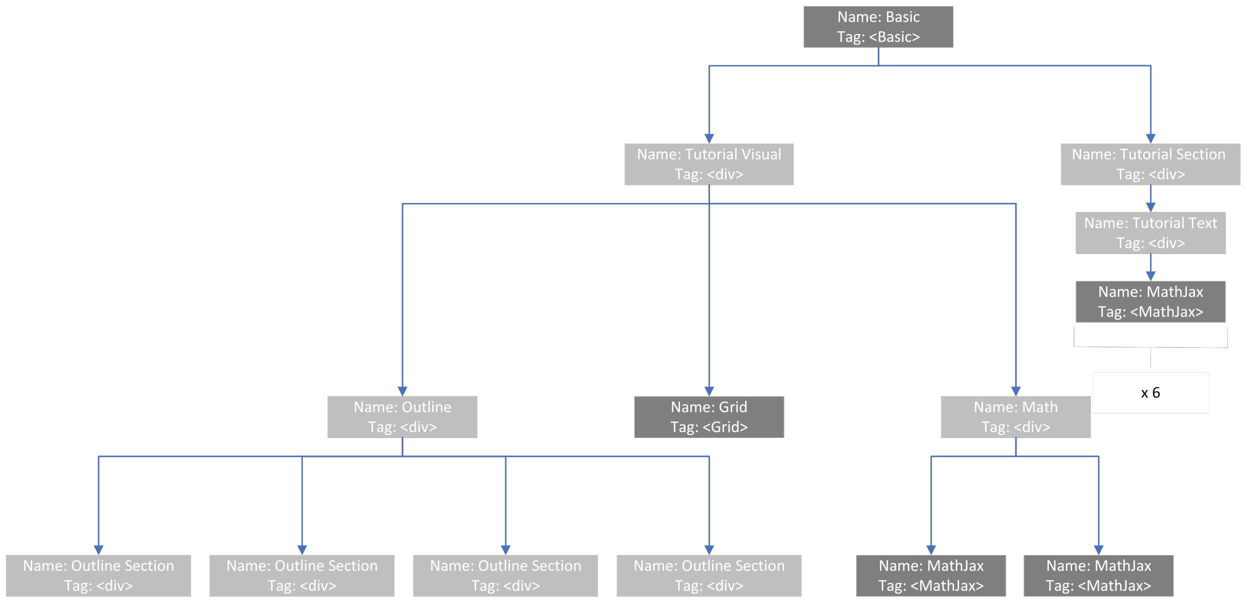


Figure 6: The DOM Tree for the Basic tutorial. Light Grey rectangles are standard HTML Elements, while Darker Grey rectangles represent React Components.

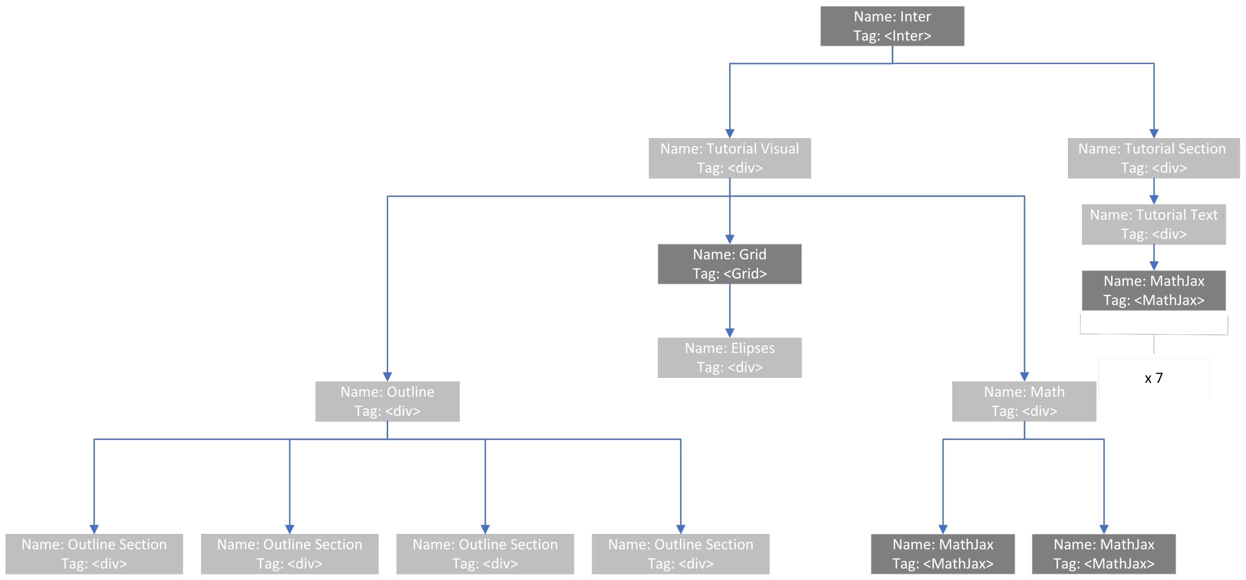


Figure 7: The DOM Tree for the Intermediate tutorial. Light Grey rectangles are standard HTML Elements, while Darker Grey rectangles represent React Components.

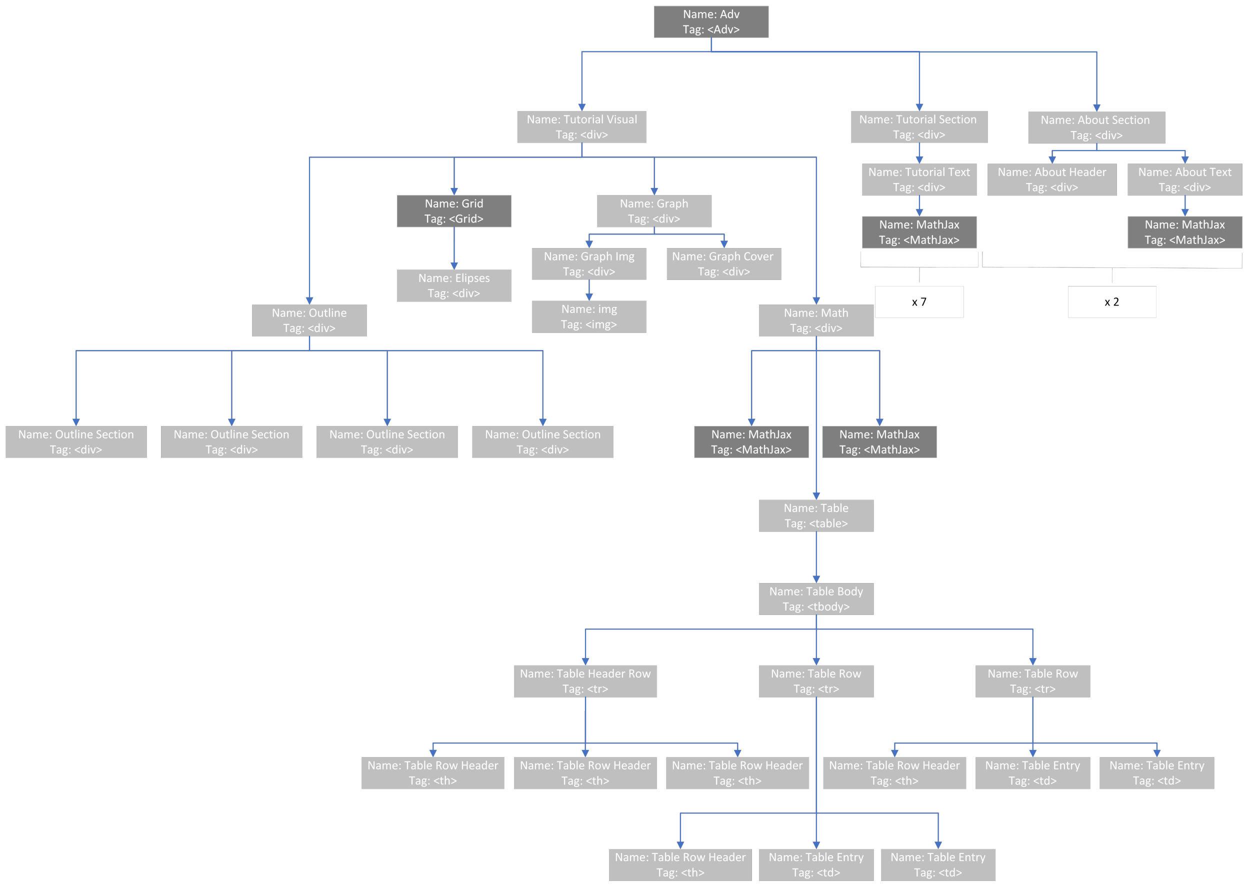


Figure 8: The DOM Tree for the Advanced tutorial. Light Grey rectangles are standard HTML Elements, while Darker Grey rectangles represent React Components.

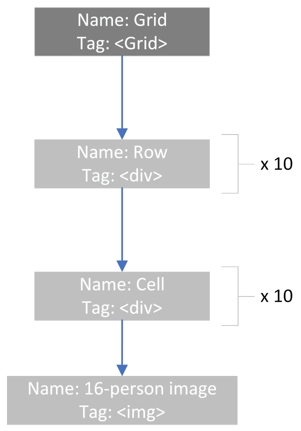


Figure 9: The DOM Tree for the Grid, the main part of the visual section of the site. Light Grey rectangles are standard HTML Elements, while Darker Grey rectangles represent React Components.