Disney Cinema Through the Ages

INFORMATION VISUALIZATION BY CHARLES ANDREWS

Project Overview

The visualization centers on visualizing the key concepts, fan and critic reception, and box office earnings of the **seven eras of Disney movies**, which are as follows:

1. Golden Age (1937-1942)
2. Wartime Era (1943-1949)
3. Silver Age (1950-1969)
4. Bronze Age (1970-1988)
5. Disney Renaissance (1989-1999)
6. Revival Era (2010-Present)

This way of organizing the history of Disney was something I learned about after reading a blog post on the topic. Growing up around Disney's wide array of content, I have always been interested in the progression of the films since Walt Disney's initial animated films. While at first I intended to focus the visualization on animated works, I decided to expand the scope to all theatrical films, or high-budget movies made for the big screen. These live action and sometimes live action-animated hybrid films are just as valuable to the brand and history of Disney in the eyes of those who grew up watching them.

Gathering the Data

Stumbling across Sameer Patel's Kaggle dataset (see Figure 1), I now had a reliable list of Disney film movie titles and release dates ranging from 1937 to August 2021. To ensure the list was up to date with recent films such as *Encanto*, I consulted Wikipedia to gather and input the release dates and film names of projects that were produced between September 2021 and now. With this now complete csv of Disney films, I would be able to use the array of release dates and film names to collect data concerning the films from APIs.

Once I had the film information, I needed to determine exactly what I wished to communicate about the topic of historical Disney cinema. After completing a demo about **OMBD**, a RESTful API which delivers movie information, I imagined using this dataset to search for each films' **IMBD rating**, which comprises fan opinion, and **Metacritic rating**, which comprises critic opinion. This would allow me to compare the viewpoints of fans and critics, which is a relationship that often does not align when it comes to personally nostalgic media like Disney films.

Through writing a Python script, I was able to request a fan score, critic score, and box office earning for each movie in my dataset. This process was centered around using a **for loop** to iterate through the csv, requesting this data using the parameters of title, referred to as 't' in

the API, and release date year, referred to as ‘y’ in the API. It was important to request the movie using its release date year as well because Disney has produced many remakes of earlier films in its history under the same title.

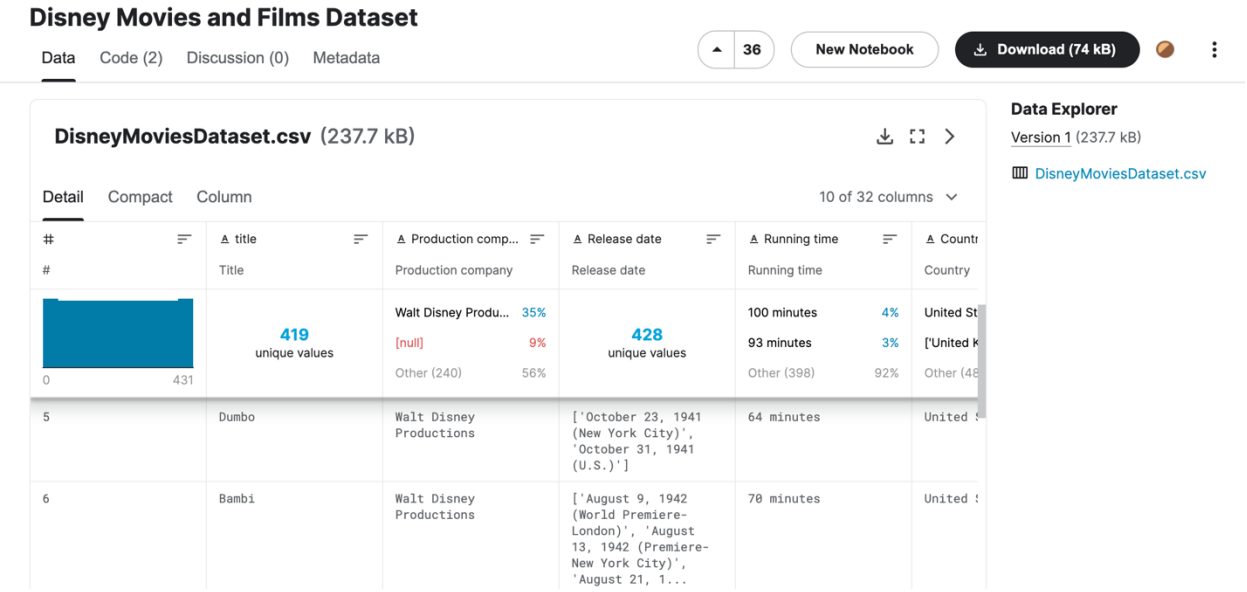

Figure 1 **Disney Movies and Films Dataset from Sameer Patel on Kaggle**

Upon receiving OMBD’s response for each film, I converted it into a json so I could access the data point’s fields easily. Then, I used a **try, except, else python structure** to test if the response was successful and contained a ‘imbdRating’, ‘Metascore’, and ‘BoxOffice’ field. If these fields did not exist when I tried to access them in the **try clause**, the **except clause** would note the current index in ‘indiciesToRemove’ so the row could be deleted after the for loop of requests ended. A placeholder String was placed in the progressively accumulated arrays of data, which collect the information in the **else clause** when no error occurs, to ensure the indexing of these lists matches the indexing of the csv. After the for loop, these accumulated lists are then added as new columns of data within the pandas data frame that is used throughout the script. Then, the script enters another for loop to delete the data frame rows that had unsuccessful responses using the saved list of ‘indiciesToRemove’ as a reference. Finally, the data frame is saved locally as a csv to be used in the d3.js based visualization.

Visualization Ideation

Before writing my OMBD python script, I made an initial mockup (see Figure 2) of what I envisioned for the visualization.

The treemap, seen at the bottom of the mockup, visualizes the distribution of data across the Disney movie eras to illustrate the brand’s film evolution at a high level. In the final visualization this **treemap was swapped out for a horizontal stacked bar chart** to

showcase the fan ratings and critic ratings in an orderly bar format (see Figure 3), rather than the non-aligned visualization format seen in the treemap's era partitions. This stacked format encourages comparing variables by era. Additionally, rather than provide a drop down to change the variable distribution that is being visualized, I decided to create **two separate stacked bar charts** that display average ratings and box office earnings respectively without the unnecessary distraction of user interaction.

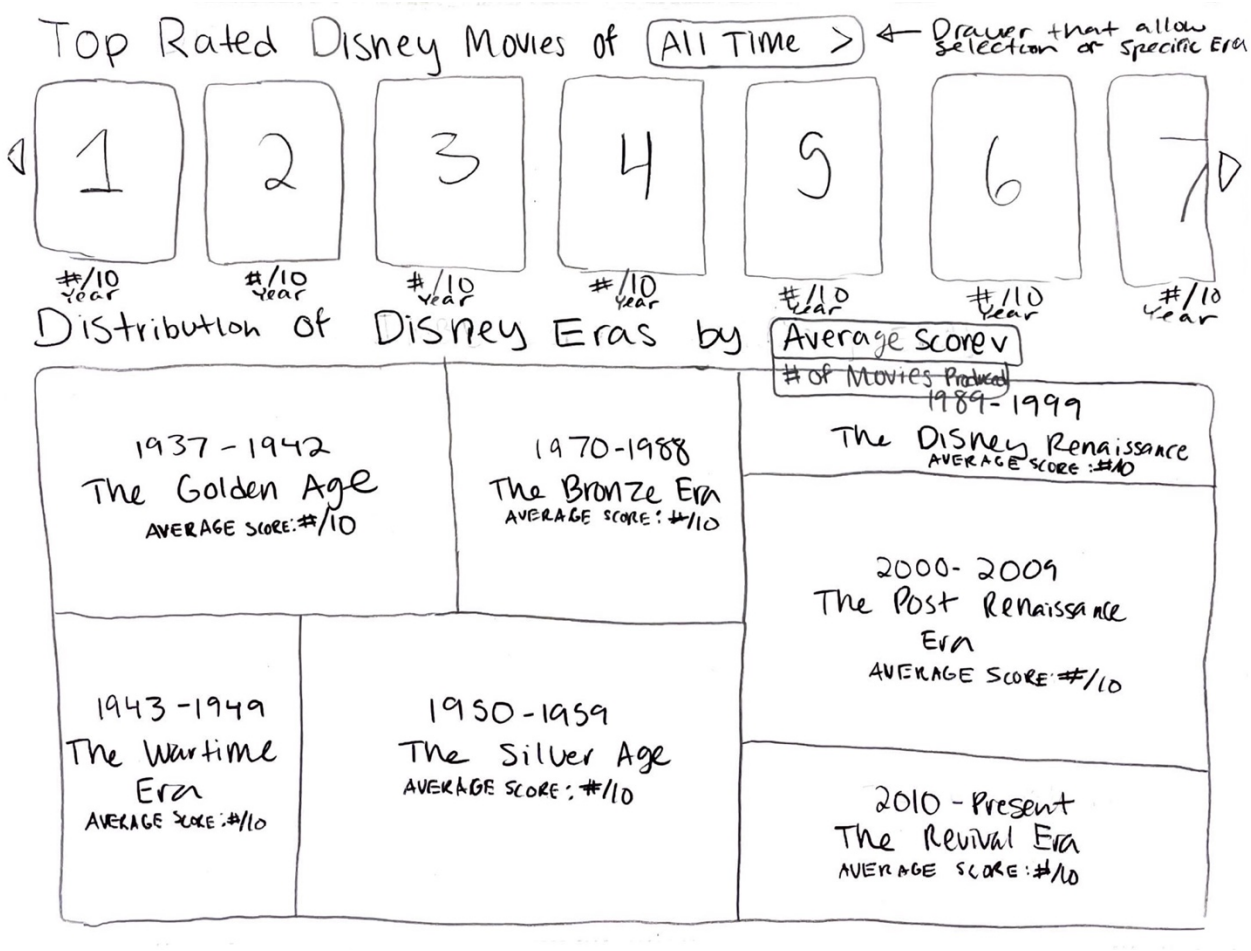

Figure 2 **Early Wireframe for Visualization**

The top section of the mockup, which allows users to explore the posters and data of top rated Disney movies in ranked order based on a selected timeframe, was faithfully reproduced in the final visualization (see Figure 4). However, the final tool is **ranked by fan ratings** specifically because that is the only data field that almost all of the successful responses from OMBD had (except one movie). I also decided to showcase more information about each movie in its caption: the critic rating, release year, and box office earnings. Furthermore, the amount of overlapping fan ratings across the films made the mockup's plan to label the movies with a specific rank unnecessary. Lastly, I decided to move this section to the bottom the page so that the ranked movie posters could be added as rows, rather than as a click through element with arrows on either side, as seen in the mockup. This allows the

user to scroll rather than force them to click the next or previous button repeatedly. This change also naturally adapted the page for mobile interactions and screen widths.

Lastly, I decided to add an unplanned introduction to the visualization to help introduce the topic of the seven eras of Disney cinema (see Figure 5). This introduction includes a timeline with short blurbs about each era's notable themes, techniques, and public reception.

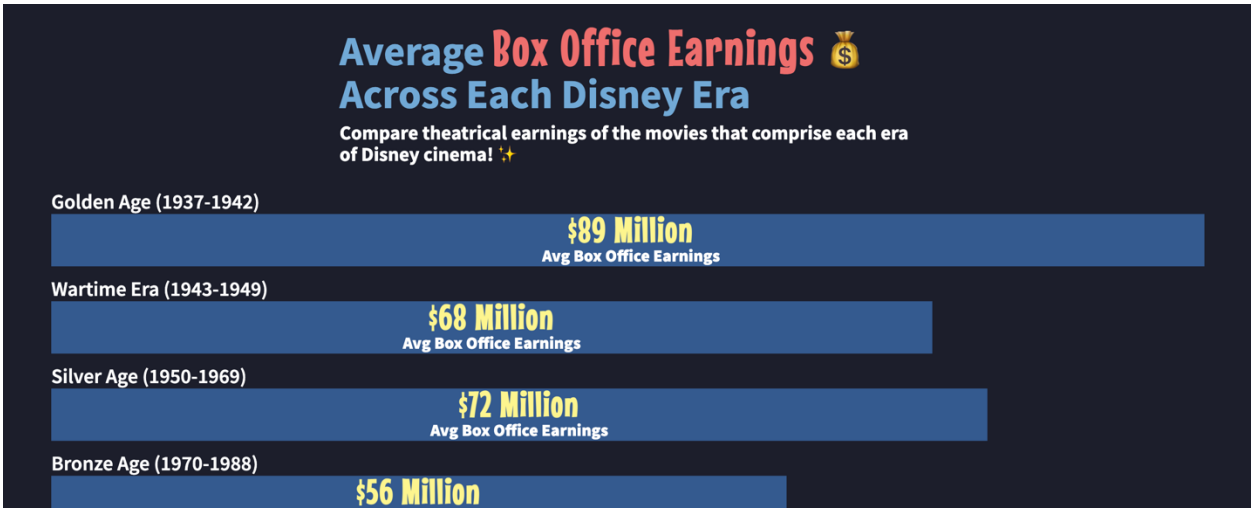

Figure 3 **Final Horizontally Stacked Bar Chart**

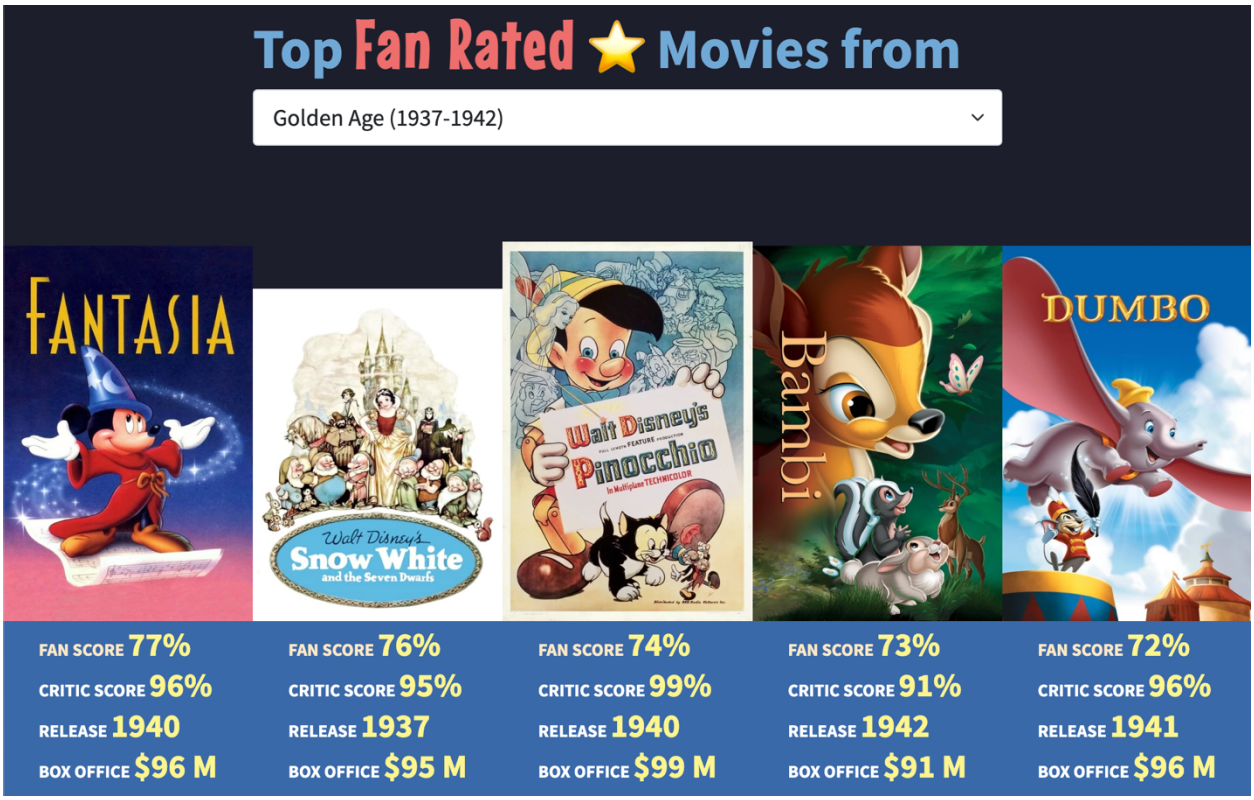

Figure 4 **Final Top Fan Rated Movies Exploration Tool**

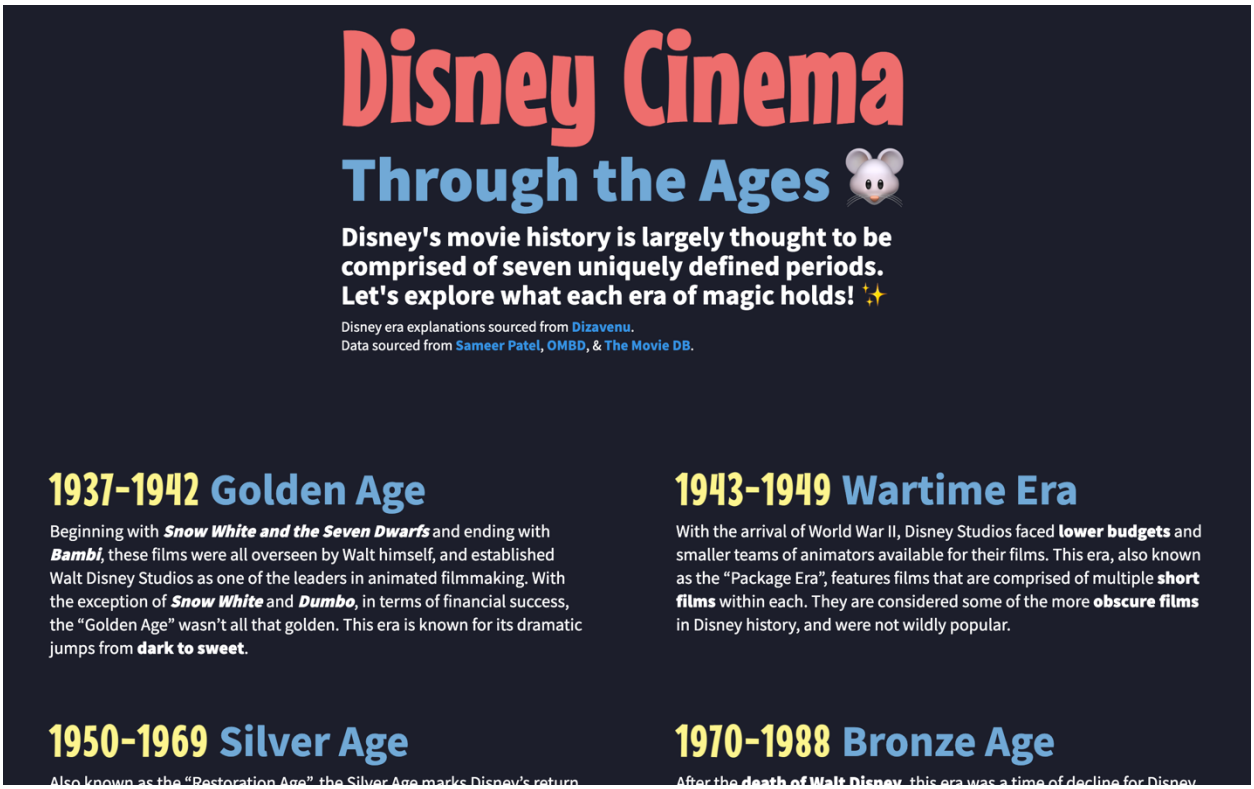

Figure 5 **Disney Era Timeline**

Development Process

To begin the development process for the web-based visualization, I sorted the dataset into a list of the custom class **EraSummary**. Having the data in chunks by Disney Era allowed me to easily feed the necessary information into the bar charts and top fan rated movies tool. An EraSummary holds a calculated **averageFanRating**, **averageCriticRating**, and **averageBoxOffice**, as well another custom class **Era**. On construction, an Era conveniently initializes and holds the **start and end year** of the Era, the **nickname** for the era, and an **eraIndexValue** to match the Era with its corresponding option in the Era selection dropdown located in the top fan rated movies tool.

A subsection of the data retrieved from OMBD returned "N/A" for the critic rating and box office earning fields within the response json. These N/A values were then translated to the csv that is used for the d3.js. I did not want to delete these movie titles from the list because they still had a fan rating, which is all that is required for the top fan rated movies tool's ranking, and this would mean eliminating a sizeable portion of my movie titles. To work around these values, I made each **N/A equal to 0** at the start of the code, which allowed me to essentially skip them when calculating the averages for each era. 0 is not a value that is found in any fan rating, critic rating, or box office earning value, so it worked well as a placeholder. When I came across these values in formatting the captions for the posters, I

used my **ifNaNThenBreakElseSpanFormat()** function to determine if the given value is equal to the 0 placeholder. If it was a placeholder value, then I **hid the placeholder text** to show that the value is not available (see Figure 6).

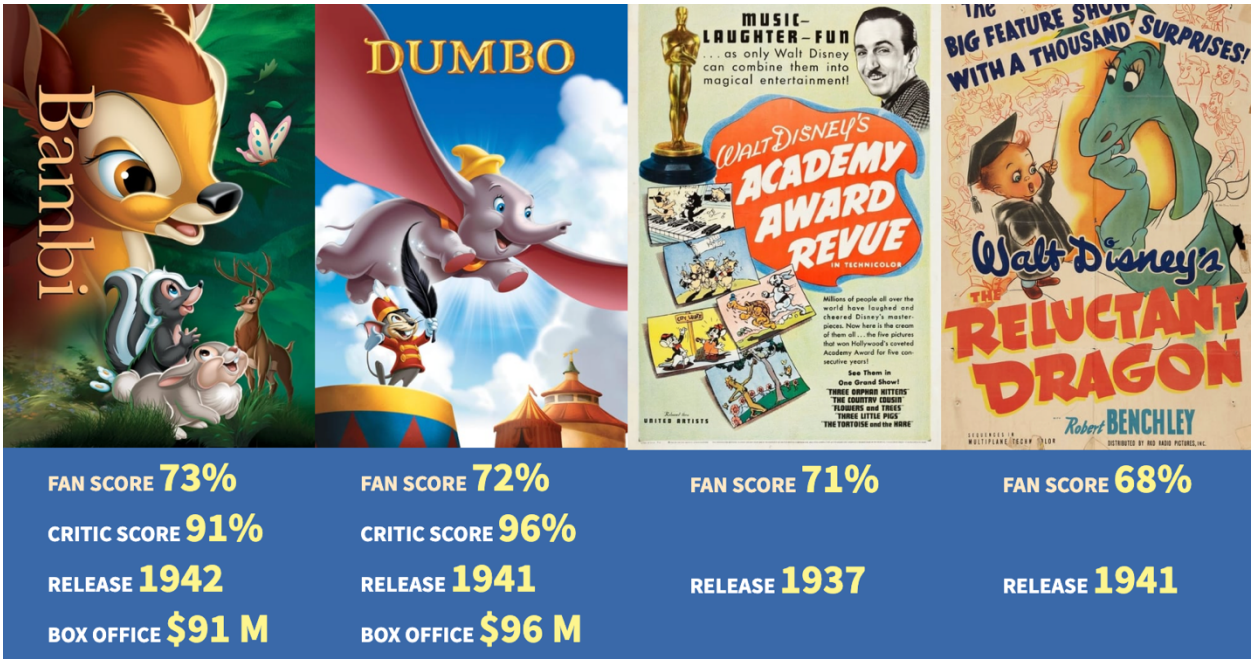

Figure 6 **Movies with Placeholder (N/A) Values Hide Text for Missing Information**

In creating the bar charts, I wanted to ensure that viewers were able to see the bars on the screen together to encourage comparison between Disney Eras. To accomplish this I used my **updateChartSize()** function to **update the svg width and height** to be about the dimensions of the browser window initially and after any window resize event. The function also reformats the bars and text to fit in the new svg dimensions. To format the text to appear inside of the bars as a caption, I **mapped its font-sizes to the screen height** so that the annotations filled up the height of the bar in an aesthetically appealing manner. I also positioned the text vertically centered using the yScale bandwidth and horizontally centered using the xScale and data value associated with the bar. Lastly, I decided to remove the axes on the graph because they were not super helpful in communicating scale for the bars that were father away from the axes. Each bar's value and scale are already adequately communicated through its internally placed text label.

My final challenge in coding this visualization was retrieving the movie posters for the top fan rated movies tool from the **Movie DB API** using JavaScript and jQuery. Initially, as well as upon selection from the Bootstrap drop down element, the **setMoviePostersAccordingToEra()** function retrieves and formats the movie poster for each movie in the currently selected Era (default Era is Golden Age). This function iterates through the movies in the csv that fall within the current Era year range, after sorting the data from highest fan rating to lowest fan rating.

With each iteration, it sends the necessary information to retrieve the poster and format its caption to the **addPoster()** function. This function first **creates a new div slot** for the current movie title in the **#poster-container**. Creating this div before requesting the movie poster was vital to preventing out of order posters due to **API lagging**, as it provides a set slot for the poster to be placed when received. Previously, posters were erroneously being appended to this div in the order they were received, rather than in their ranked order. Moreover, similar to handling the instance where OMBD did not return a successful response, I had to check the movie poster response from the Movie DB, which takes the form of a list of urls, to determine if it had results. In the event that it had no results, this meant that the API did not have a poster to provide so I hid the movie's div by setting the display attribute to none. This also prevented erroring when trying to access a url that does not exist. The movies that did not have a poster were rather obscure so removing them did not majorly affect the visualization's integrity. Otherwise, if the requested poster existed, I added an image element with the url of the first result and its formatted caption div to its slot. The parent #poster-container has its CSS Grid's **grid-template-column** attribute in the format of **repeat(number-of-columns-desired, equal-positioning-unit)** to effortlessly position the poster divs in consistent row lengths.

Overall, this project allowed me to explore the world of APIs to gather data in the form of numbers and texts, but also images. I was happy to be able to put together a Disney-inspired webpage that used its gathered data to illustrate key progressions in Disney cinema. APIs are a powerful concept, and they are definitely something I hope to explore and learn more about in the future!