In this project, I analyzed the phenomena of a damped pendulum and modeled the oscillatory motion of the system. I explored the forces that act on a string pendulum and how different factors and environments would manipulate the system. I found this project fulfilling because I was not only able to better understand how these pendulums work, I also improved my coding ability and learned a new aspect of it via animation. While I did not achieve all that I set out to do, this leaves me with possible additions and areas that I may further study so that I may improve my physics and coding knowledge.

To begin this project, I was interested in better understanding how drag and damping forces impact objects and was intrigued by the idea of modeling these systems so that I may see what happens and so I may practice my derivations. I intended on starting from a single, simple damped pendulum, then hoped to explore what different combinations of damped systems would look like. In my proposal, I outline how I would have liked to model a coupled pendulum (where two string pendulums would be connected by a spring) because this would challenge my understanding and produce really interesting, superimposed graphs; changing certain values, like the starting positions of the pendulums, would have produced unique results too. Nevertheless, I am pleased with my ability to apply physics concepts to code. I believe I am set up in a positive direction to investigate the more complicated situations. As I explained in my proposal, I was familiar with the behavior of the curve I needed and in my project I worked through why this type of bounded cosine curve made sense through the use of forces. Ultimately, the constants and coefficients were arbitrary, but I wanted to choose values that were realistic and accurate to what I may experience on the surface of Earth. When researching more about this equation, I learned that the drag coefficient is a dimensionless quantity that depends on the speed of the object, geometric factors, and the density of the liquid the object was moving in. I learned that a typical value for a car moving through air is between 0.3 and 0.35. I also found more information on other coefficient values for more shapes. In my project, I opted to use the value I found for a sphere (0.47). To be more accurate to the real model I created in conjunction with my simulation, I could have picked values that corresponded more to the shapes and weights of the real objects I used to test this phenomena.

Furthermore, I mentioned how I would collect real data of this phenomena at home and use it within my project. While I did not explicitly extract the data I collected and compare this with my generated data, I did draw an indirect comparison along the way. Before I began coding, I created an apparatus in my dorm floor with materials I had around. I used nearly massless floss as a string and a bag of nuts tied to an extended door handle to mimic what I wanted to simulate. This was fun because I got to improvise a good test that limited errors and was realistic. This also allowed me to work more with features of LoggerPro I rarely had experience with, such as the capture video feature. I took a video on my laptop and used each frame to mark data points. This was challenging at first because I needed to teach myself how to use the software and turn the number of frames into usable seconds. Eventually, I produced graphs of the horizontal position and tangential velocity of the pendulum (bag of nuts) versus time. Unique to LoggerPro was the function to curve fit and decide which type of fit you wanted to use. From my derivations, I decided to use a trigonometric cosine curve that was bounded by exponential decaying curves, and this turned out to be a really great fit for the data, which I was pleased with. In essence, I collected data through LoggerPro, but did not end up using it. I generated my own data that matched this phenomena and it suited my purposes better because

I am able to easily control the parameters and generate data for any simulation I want. It may have been a good idea to animate my collected data side-by-side with my generated data, but I ran out of time. In terms of data filtering and cleaning, there was really no need to do so for my generated data because there were no duplications once I went through everything and there were no outliers due to the equations I used to create the data. For my collected data, there are more reasons to suspect errors because I needed to choose points in the video to record. To do so, it was reasonable to choose the same relative spot on the pendulum to get an accurate understanding of the whole body's motion. This invites slight fluctuations. Luckily, this form of data collection was forgiving because I was able to go back if I made a mistake and retake the points I collected. Additionally, if one of my takes for the video was bad (i.e. the object bumped into something or the release was bad) I could easily retake the video because I knew that the experiment was not done correctly. For my simulation, once I knew what the shape of my graph looked like, I could easily compare my pendulum object with the graph and see if I made mistakes in how I gave the pendulum data.

For my code, I used Numpy like I said, but I also used Matplotlib and its animation feature (with FFMpegWriter). Numpy was a great tool for me to create usable data that could be manipulated with arithmetic. This helped with generating more data and eventually plotting it with Matplotlib. Once I was exposed to this package more in class, I was convinced that this would be the best way to approach plotting and animating my pendulum and curve in real time. Thanks to discussions, office hours, and supplemental lessons, I was able to teach myself how to properly use animation features and troubleshoot problems I ran into. What was really frustrating was how I knew how to plot curves, but could not plot my final equation for position with respect to time. I believe this was due to how one of my parameters for the equation included imaginary numbers; in order to graph the damped pendulum, I needed to find a new frequency value ω', and the the discriminant in the value was less than zero. Despite this initial confusion, I thought about using the differential equations I solved for earlier. I explored how others used special packages from Scipy to create ODE variables, but I opted for a simpler approach. I basically used Euler's formula to help me generate new data points in an array to then plot. This ended up being a great alternative. Maybe I can explore more about the certain Scipy methods others used to see if they are better for more complicated systems or if my approach has greater error. Next, when it came to actually animating a "real-time" curve, I made things more complicated than they should have but eventually realized how simple the method was. There were many different approaches to animation I found and many arguments that could be called upon, so it felt daunting at first. The hardest part about the animations was knowing what I wanted to feed into the objects I created and the animation/update function. Numerous trials were unsuccessful due to my misunderstanding of my own data. For example, when I was creating the pendulum, I thought about the x and y positions of the pendulum in terms of the updated theta values I created. However, problems rose because I forgot that the x data I created already resembled the x component of the angles in my theta data, so I kept taking the sine of the sine of the angles! Hopefully this makes sense. Before I realized this, my pendulum was not following the same position as my curve. Once I made sense of it, I had success in my simulation. From this issue, I learned that I must understand what my data represents before doing anything with it.