My pipeline for this project was heavily based off of the coursework leading up to the project as a starting point. Since many of the quizzes covered the content, I utilized those code snippets to kick off the pipeline.

It starts off with the greyscale and blurring functions to feed the Canny edge detection algorithm. I used 150 and 180 for lower and upper thresholds, which seemed to pick out good edges on the road. I knew I was going to be masking off the road later on so I didn't care that Canny was picking out treelines and sharp lines on cars.

Next, I utilized masked edges to create a defined 4 sided polygon to trim out the sky , far horizon, and neighboring cars.

Then I utilized the Hough transform, using trial and error to determine optimal values for rho, theta, and threshold. However, I found that the biggest gains happened by keeping minimum line length at 15 pixels, and maximum line gap at 15 pixels. This let me keep identifying dashed lines far into the distance while keeping foreground dashed lines clearly defined as lines.

Now for the fun part. I began to sort out the slopes of all the Hough line segments by positive and negative slopes between 0.5 and 0.8. This eliminates the cat's eyes between dashed lines, and any weird shapes from road debris. However, the algorithm was still pulling in line segments from the wrong sides of the road, so I needed to filter further so that it was only looking at the left and right hand sides of the screen. Finally, with these filters, the slopes were consistently coming out correct, and the "jitter" was kept to a minimum.

Now the hard part was to figure out how to extrapolate lines to the horizon and back to the foreground (hood of the car). I struggled with this for a while but finally, I was able to utilize the centers and slope calculations mentioned in the Youtube Office Hours recording. I found the LH and RH centers for each frame, then extrapolated the lines to 60% of the height of the image, then used the slope to draw out the horizon endpoints. The lower endpoints simply extend past the bottom edge of the image (+300 pixels).

Shortcomings:

I wish I could average across frames. Because of the way that the code is structured, I can't think of an easy way to do this. This would smooth out the video and would prevent the lines from moving too much from dips and bumps in the road.

My code might also get confused if the car went too far left or right, because the slopes would become more vertical, and then get filtered out entirely once they become > 0.8 slope.

Improvements:

I ran out of time to do the challenge MP4. The HSV conversion makes sense so that it would track the yellow lines well even with the light colored pavement and curves.

I should also consider calculating the bottom edge of the image so that it properly draws the lines towards the bottom of the screen instead of blinding going past the edge. Also I probably could have figured out what the highest Hough transform line would have been, and used that as the upper y-value. This would account for rising and falling roads, and it would also shorten lines when the road gets curvy.