#**Finding Lane Lines on the Road**
The goal of this project is to make a pipeline that finds lane lines on the road and use this pipeline for processing images and videos.

### 1. PIPELINE:

My pipeline consisted of 6 steps. Namely
(i) to convert color image to greyscale.
(ii) to apply Gaussian noise to the converted greyscale image.
(iii) to use canny edge detector/ canny edge transform to detect the edges in the given image.
(iv) to mark the region of interest (ROI) and to retain that part of the edge-image inside the ROI and neglecting the rest.
(v) to draw lines on the edge-image that marks the lanes of the road. (using hough transform).
(vi) to create a weighted image that consists both the initial input image and the line marked image. I.E. to show the lane lines marked on the original image. (output image)

Steps (i), (ii) and (iii)are self-explanatory. We use the functions
"cv2.cvtColor(input_image, cv2.COLOR_RGB2GRAY)"
"cv2.Canny(gray_image, low_threshold, high_threshold)" and
"cv2.GaussianBlur(edge_image, (kernel_size, kernel_size), 0)" respectively.

For step (iv): we initially used triangle function to create a mask that is triangle in shape. But in practice it does not perform feasible and hence we introduced the polygon function
"cv2.fillPoly(dummy_image, vertices_of_the_polygon, ignore_mask_color)". Here the ROI is marked based on the 4 vertex co-ordinates we provide for our polygon. This vertex selection is done manually looking into the images and the videos.

Step (v) plays the major role in our finding lanes project. Its function is to draw the line indicating the detection of lanes.
(I) Initially, we only marked the lane lines based on our edge-image and ROI. Though successful in detecting and marking the lanes on the road, it only marked the lanes as it is. I.E. the broken lane lines were marked as broken only. For a computer to understand the lane, it sees it as the area between the two lines segments. When we have broken lines, it is quite difficult for a machine to comprehend. Hence, we draw a straight line even over the broken lane line. This is done by calculating the slope of the lines produced and using the best fit slope 'M' using the openCV function "cv2.fitLine()". Thus, irrespective of being white or yellow, or solid/broken lane line, we have successfully detected and have marked the lanes of the road.

### 2. Potential short comings:

*One short coming that I have seen is while performing the lane detection on the 'challenge.mp4' video. Here we have curvy lane lines, whereas we have programmed to draw only straight lines. So, this is a shortcoming in our project.
*Another is that, we know that the rules to be followed for yellow and white, solid and broken lines are different. But we have programmed to draw lane lines irrespective of the above differences.

### 3. Suggestions to improve:

* Instead of manually marking the vertices of the polygon for our ROI, we can develop an algorithm to automatically markdown the ROI based on the input.
* While we function to draw only straight line for different 'M' values, we have to consider non-linear lane lines also.
* We can use color masks to differentiate the white and yellow lane lines so that we can instruct the machine more precariously.