Camera Calibration:-

Defined number of corners across horizontal and vertical direction, ran a for loop across all the given images, converted into grayscale images and by using '**findchessboardcorners**' method from OpenCv found the corners. For each successful corner append the corners to the imgpoints list.

Code is like below:

```
img = mpimg.imread(image)
    gray = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)
    ret, corners = cv2.findChessboardCorners (gray, (nx, ny), None)
    if ret==True:
        objpoints.append (objp)
        imgpoints.append(corners)
```

After finding the object points and image points, computed the camera matrix and distortion coefficients using opencv '**calibrateCamera**' method. Find the below code:

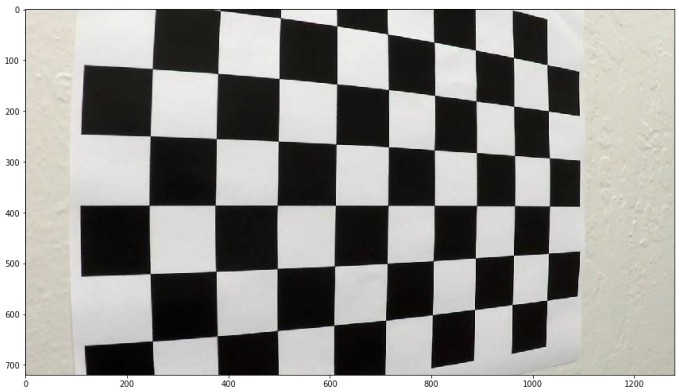
```
ret, mtx, dist, rvecs, tvecs = cv2.calibrateCamera (objpoints,
imgpoints, gray.shape[::-1],None, None)
```

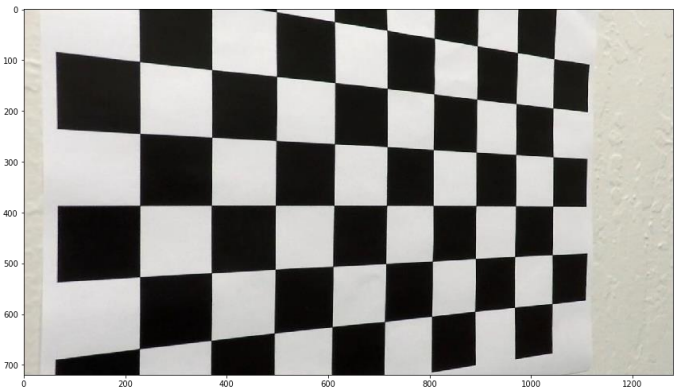
Distortion Correction:

After finding the camera matrix and distortion coefficients we can undistort the image using opencv method '**undistort**'

```
dist = cv2.undistort(image,Cameramatrix,distortioncoeffts,None,mtx)
```

Please find below original image and undistorted image:





Pipeline:-

Threshold and Binary Image:

Used color gradients, sobel along 'X' and directional threshold to get the final combined binary image. For the color gradient used the channels R, G from RGB color space, L, S channels from HLS color space and applied suitable thresholds to detect the lane lines. Used the same logic as described in the tutorial like:

```
//Get the corresponding channel from color space //
        R = img[:,:,0]

//Define color threshold condition //
R_cond = (R>threshold[0]) &(R<threshold[1])

//Apply the sobel along x direction using directional threshold //
sobelx_binary = abs_sobel_thresh(gray, 'x', 10, 200)
direction_binary = dir_threshold(gray, thresh=(np.pi/6, np.pi/2))
//Combined sobel along X and directional threshold Condition//
combined_binary = ((sobelx_binary == 1) & (direction_binary == 1))

    //Combine all conditions //
color_combined[(r_g_condition & l_condition) & (s_condition | combined_binary)] = 1
```

Please find the original image and combined thresholded image

Perspective Transform:

Define source coordinates and destination coordinates manually

Source Points:

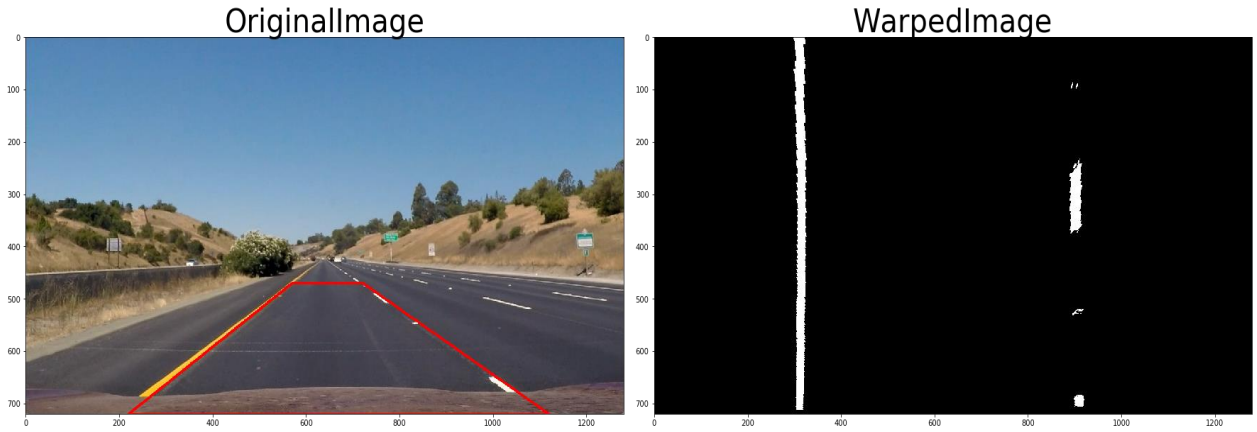
leftbottm = [220,720]
rightbottom = [1120, 720]
leftop = [570, 470]
righttop = [722, 470]

Destination Points:-
leftbottm = [320,720]
rightbottom = [920, 720]
leftop = [320, 5]
righttop = [920, 1]

Converted both points to Numpy array of type float32, got the perspective using source and destination points, perform warp perspective .Find the below code:

```
src = np.float32([leftbottm,rightbottom,righttop,leftop])
dst = np.float32([leftbottm,rightbottom,righttop,leftop])
img_p = cv2.getPerspectiveTransform(src, dst)
warped_image = cv2.warpPerspective(combined_binary, img_p, gray.shape[::-1] ,
flags=cv2.INTER_LINEAR)
```

Find below original image and warped image



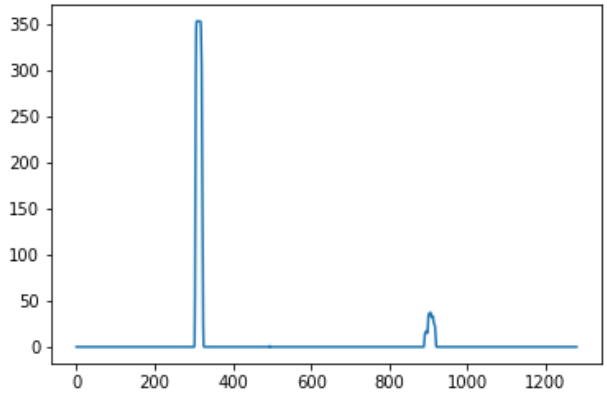
Here warp perspective performed for the previous combined binary image result.

Lane Finding:

Lane pixels are found by using the histogram technique. Histogram technique is if we calculate the histogram of the all the pixel values from the combined binary image there will be only two spikes which are lane lines. So initial x_base coordinates for left lane and right lane is found by using the histogram coordinates. Corresponding y coordinates is found by using the margin. Defined '10' windows to draw around the lanes by fitting a 2nd degree polynomial. Code is like below:

//Calculate the Histogram //

```
histogram = np.sum(warped_image[warped_image.shape[0]//2:,:], axis=0)
```



Two spikes in the above image indicates two lanes of the road.

Leftx base and Rightx base calculated like below:

```
midpoint = np.int(histogram.shape[0]/2)
leftx_base = np.argmax(histogram[:midpoint])
rightx_base = np.argmax(histogram[midpoint:]) + midpoint
```

Run a for loop for the number of windows :

for window in n_windows:

//Calculate window coordiantes //

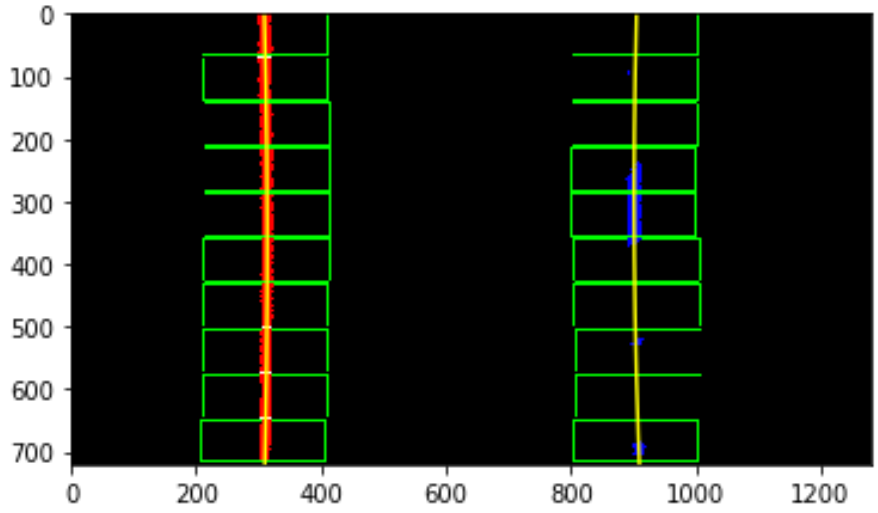
```
win_y_low = warped_image.shape[0] - (window+1)*window_height
    win_y_high = warped_image.shape[0] - window*window_height -5
    win_xleft_low = leftx_current - margin
    win_xleft_high = leftx_current + margin
    win_xright_low = rightx_current - margin
    win_xright_high = rightx_current + margin
```

If there are any nonzero pixel values in the windows, get all those indices. Code below:
good_left_inds = ((nonzeroy >= win_y_low) & (nonzeroy < win_y_high) & (nonzerox >= win_xleft_low) & (nonzerox < win_xleft_high)).nonzero()[0]
good_right_inds = ((nonzeroy >= win_y_low) & (nonzeroy < win_y_high) & (nonzerox >= win_xright_low) & (nonzerox < win_xright_high)).nonzero()[0]

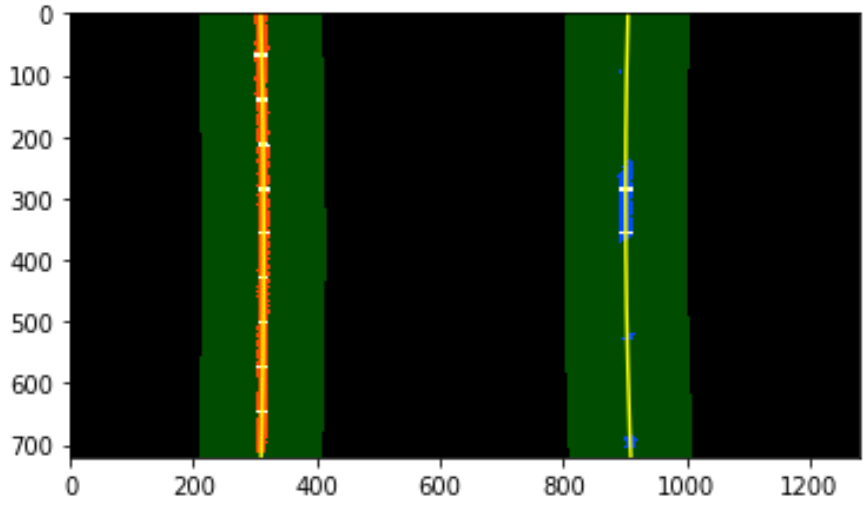
//Apply a second order polynomial //

```
left_fit = np.polyfit(lefty, leftx, 2)
right_fit = np.polyfit(righty, rightx, 2)
```

Final Output image is like below:



Visualize the image by recasting the found x and y points into usable format for cv2.fillpoly



Calculate the Curvature Radius:

Radius is calculated by using the same logic given in the tutorial.

```
ym_per_pix = 30/720
xm_per_pix = 3.7/700
ploty = np.linspace(0, 719, num=720    y_eval = np.max(ploty)
fit_cr = np.polyfit(ploty*ym_per_pix, values*xm_per_pix, 2)
curverad     =     ((1     +     (2*fit_cr[0]*y_eval*ym_per_pix     +
fit_cr[1])**2)**1.5) / np.absolute(2*fit_cr[0])
```

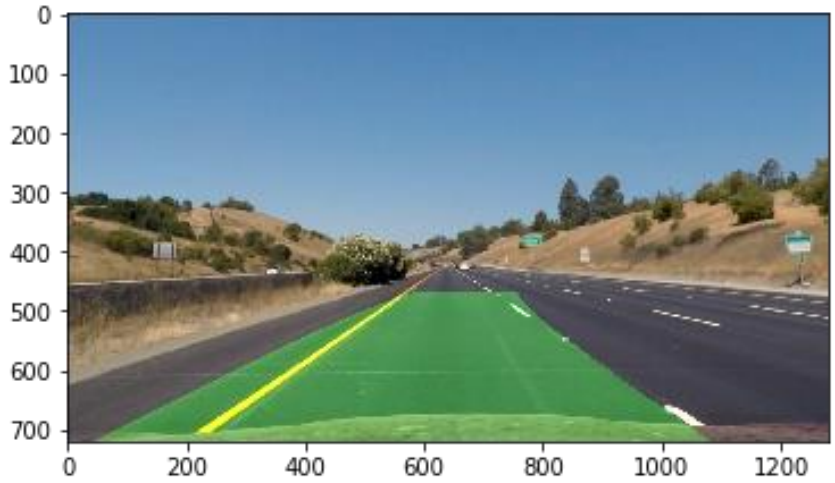
Offset values from the centre is calculated by using the image shape as below:

```
lane_center = (rightx[719] + leftx[719])/2
xm_per_pix = 3.7/700
center_offset_pixels = abs(gray.shape[0]/2 - lane_center)
center_offset_mtrs = xm_per_pix*center_offset_pixels
```

Final Output Image:

Final output image is obtained by performing the inverse perspective and combining with the original image.

PFB final output image:



Output Images and Video:-

All output images are contained in the folder '**output'** and the project video is saved as '**project_video_output.mp4**'.

Further Improvements:-

Pipeline may fail for the extreme cases of more shadowing, cases of roads with tunnels because code unable to detect lines in this case and the case with more number of sharp turns like in the challenge video case.

Improvements needed for averaging over more number of frames, more robust binary thresholded image and handling sharp turns by taking perspective transform over small section of the image because turns are happening very fast.