Advanced Lane Finding Project

The goals / steps of this project are the following:

- Compute the camera calibration matrix and distortion coefficients given a set of chessboard images.
- Apply a distortion correction to raw images.
- Use color transforms, gradients, etc., to create a thresholded binary image.
- Apply a perspective transform to rectify binary image ("birds-eye view").
- Detect lane pixels and fit to find the lane boundary.
- Determine the curvature of the lane and vehicle position with respect to center.
- Warp the detected lane boundaries back onto the original image.
- Output visual display of the lane boundaries and numerical estimation of lane curvature and vehicle position.

Writeup

This section addresses all the necessary sections called out in the Rubric.

Camera Calibration

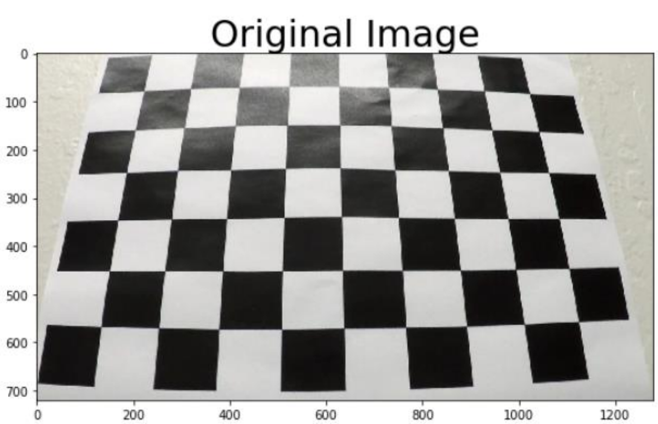
1. Briefly state how you computed the camera matrix and distortion coefficients. Provide an example of a distortion corrected calibration image.

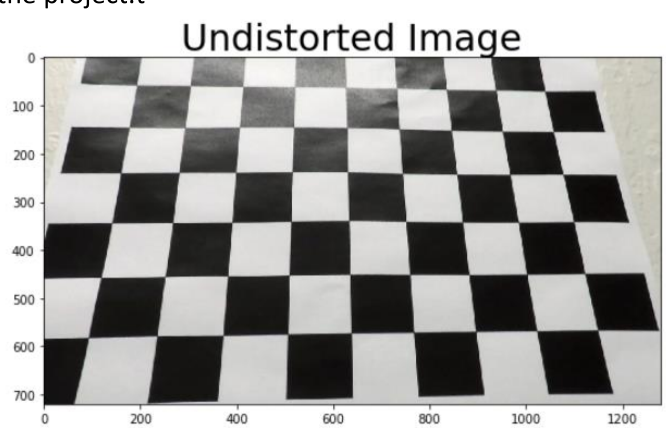
The code begins with just initializing variables and then loading chessboard images from the file. The calibration code then steps through each image if the corners were found, then appends the objpoints and imgpints to their respective lists, and creates a new image with the chessboard corners drawn in. This gets done for all of the images that had chessboard corners found.

Next, I load a random image from the given chessboard images and I load in the object points and img points into the CameraCalibration function which returns the distortion coefficients (dist), the camera matrix (mtx) that I need to transform 3D object points to 2D image points. Also returns the position of the camera in the world with values for the rotation and translation in a vector form (rvecs and tvecs, respectively).

Next, I use the undistort function which takes in a distorted image, the camera matrix and the distortion coefficients. This function gives us a destination image that is undistorted.

Finally, I save and plot the undistorted image and then create a pickle file to save the distortion coefficients and camera matrix information into which can later be recalled and used in the project.t



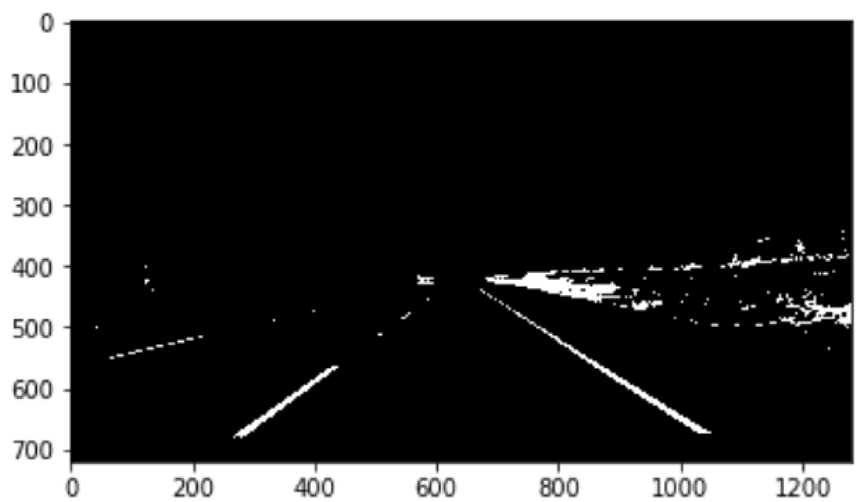


Pipeline

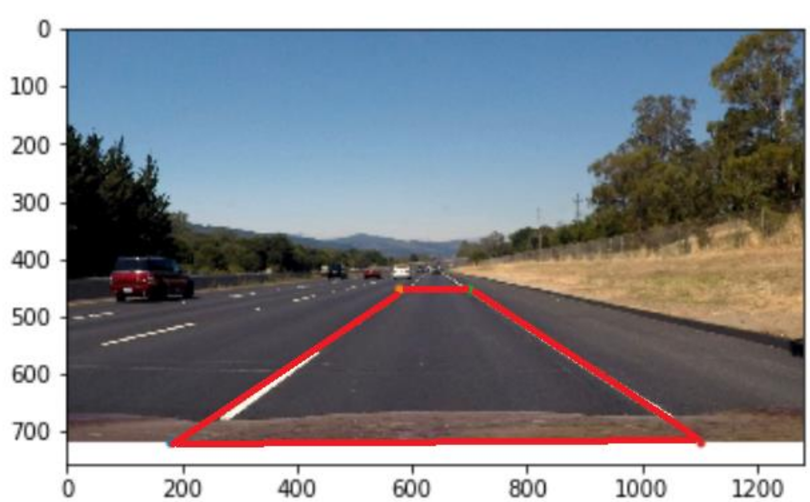
1. Provide an example of a distortion-corrected image.

Here's the original image:

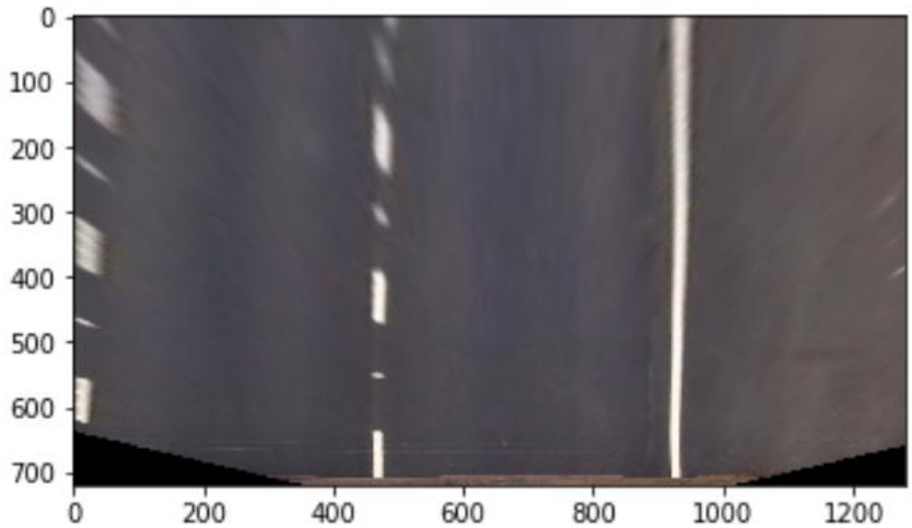
Here's the same image with the filtering/thresholding/masking applied to it:



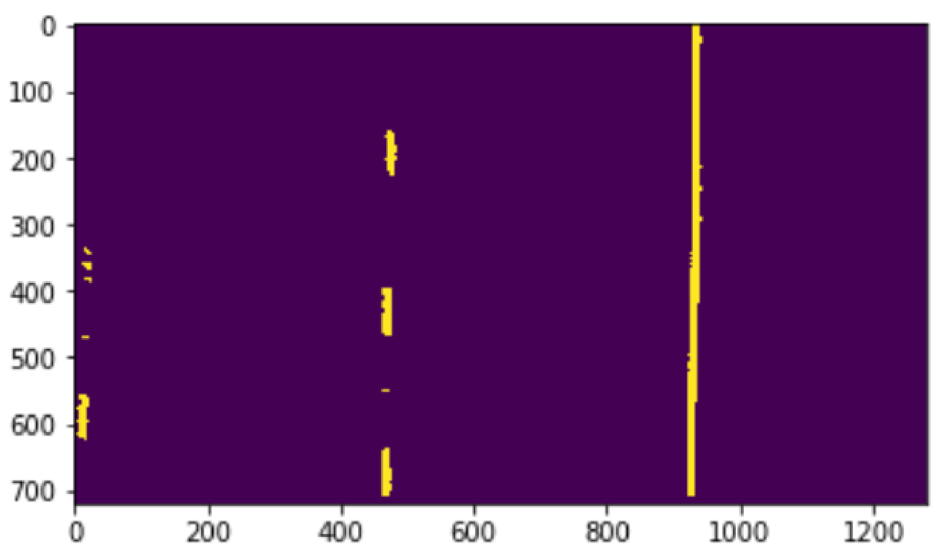
I then do a perspective transform to transform a certain part of the lane in front of the vehicle into a square:



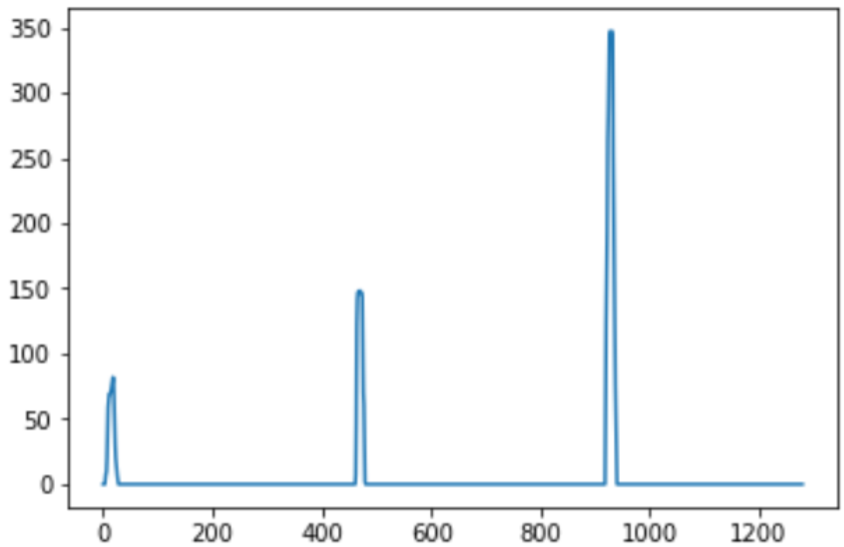
I get this transformed image:



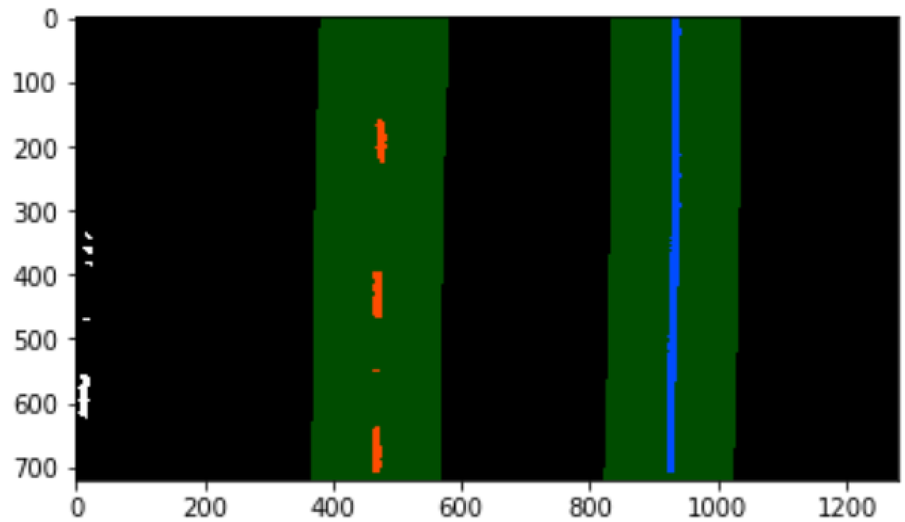
I then take this image and apply the same filtering to it to make it look like this so I can clearly pick out the lane lines:



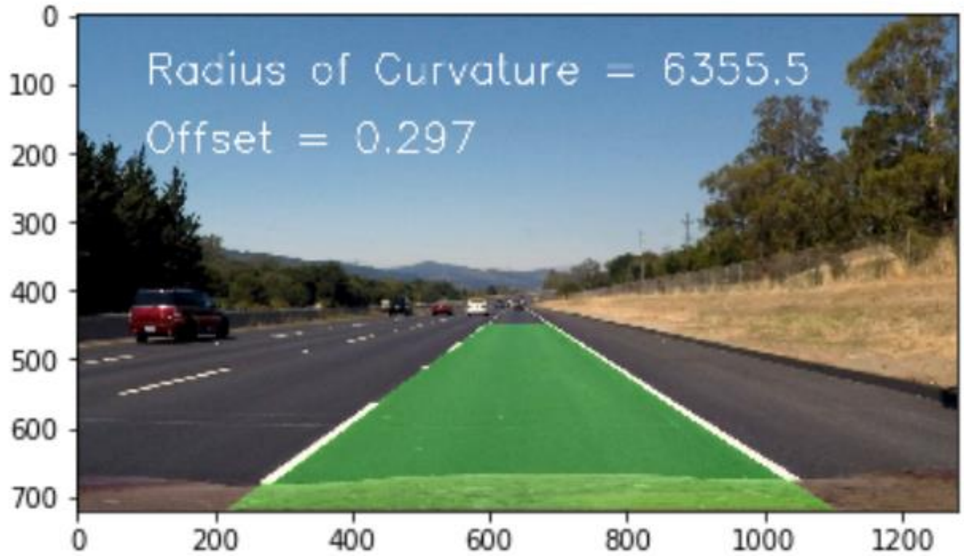
I then use a histogram to figure out the base of the lines to start searching across the window for lanes:



I then pick out the left and right lanes and fit a polynomial line to each lane:



Lastly, I calculate the radius of curvature and lanes offset value and overlay a rectangle over the lane up ahead. This is the final undistorted image:

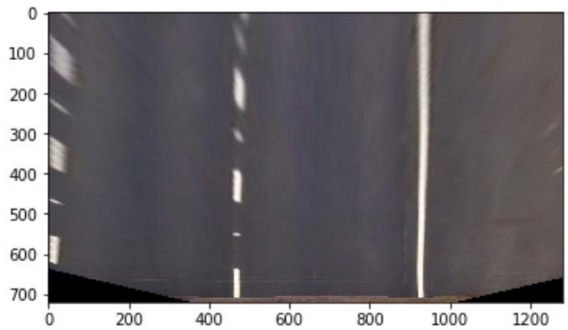


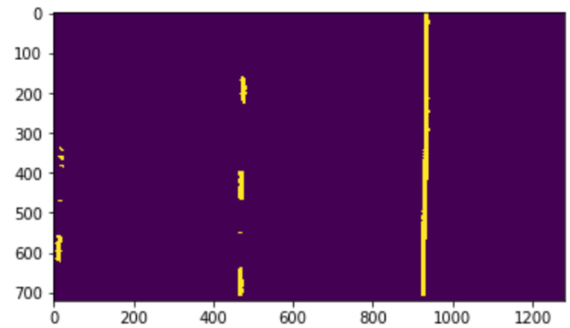
Beginning of pipeline code (more images in zipped file):

```python
def warp(img):
    global imagecount

    img = cv2.undistort(img, mtx, dist, None, mtx)
    origimgout = 'outputvids/origfinal'+str(imagecount)+'.jpg'
    plt.imsave(origimgout,img)
    cv2.waitKey(500)
    #plt.figure()
    #plt.imshow(img)

    imagecount = imagecount + 1

    img_size = (img.shape[1], img.shape[0])
    #print(img_size)
    yoffset=470
    src = np.float32(
        [[180,img.shape[0]],
         [575,yoffset],
         [700,yoffset],
         [1100,img.shape[0]]])

    dst = np.float32(
        [[390, img.shape[0]],
         [430, 0],
         [920, 0],
         [990, img.shape[0]]])

    binary_img = pipeline(img, s_thresh=(90, 255), sx_thresh=(20, 100))
    binaryimgout = 'outputvids/binaryoutfinal'+str(imagecount)+'.jpg'
    plt.imsave(binaryimgout,binary_img)
    #plt.figure()
    #plt.imshow(binary_img)
    cv2.waitKey(500)

```

2. Describe how (and identify where in your code) you used color transforms, gradients or other methods to create a thresholded binary image. Provide an example of a binary image result.

I have used a combination of Red & Green, Saturation & Lightness thresholding in my code to perform color transforms and create binary images that effectively pick out the lane lines. The R/G thresholds are good for picking out the yellow lanes and the S/L thresholds help with avoiding darker regions of the road.





Code:

```
def pipeline(img, s_thresh=(0, 255), sx_thresh=(200, 255)):
    img = np.copy(img)
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    hls = cv2.cvtColor(img, cv2.COLOR_RGB2HLS)
    #hsv_image = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
    s_channel = hls[:,:,2] #for other yellow lane situations
    l_channel = hls[:,:,1]
    #S_hsv = hsv_image[:,:,1] #for shadows
    #V = hsv_image[:,:,2]
    R = img[:,:,0] #for yellow line
    G = img[:,:,1]
    thresh = (190, 255)
    #Vthresh = (200, 255)
    #S_thresh = (200, 255)
    R_binary = np.zeros_like(R)
    G_binary = np.zeros_like(G)
    #S_HSV_binary = np.zeros_like(S_hsv)
    #V_binary = np.zeros_like(V)
    #R and G threshold application
    R_binary[(R > thresh[0]) & (R <= thresh[1])] = 1
    G_binary[(G > thresh[0]) & (G <= thresh[1])] = 1
    #S and V threshold application
    #S_HSV_binary[(S_hsv > S_thresh[0]) & (S_hsv <= S_thresh[1])] = 1
    #V_binary[(V > Vthresh[0]) & (V <= Vthresh[1])] = 1
    # S Threshold color channel
    s_thresh_min = 100
    s_thresh_max = 255
    s_binary = np.zeros_like(s_channel)
    s_binary[(s_channel > s_thresh_min) & (s_channel <= s_thresh_max)] = 1
    # Adding L channel to avoid dark shadows getting detected as lines
    l_thresh_min = 100
    l_thresh_max = 255
    l_binary = np.zeros_like(l_channel)
    l_binary[(l_channel > l_thresh_min) & (l_channel <= l_thresh_max)] = 1
    #Combine the hresholds
    combined_binary = np.zeros_like(R_binary)
    combined_binary[((R_binary == 1) | (G_binary == 1)) &
                    ((l_binary == 1) | (s_binary == 1)) &
                    ((l_binary == 1) | (R_binary == 1)) &
                    ((s_binary == 1) | (R_binary == 1)) &
                    ((l_binary == 1) | (G_binary == 1)) &
                    ((s_binary == 1) | (G_binary == 1))]=1
                    #((R_binary == 1) | (V_binary == 1)) &
                    #((G_binary == 1) | (V_binary == 1)) &
                    #((L_binary == 1) | (V_binary == 1)) &
                    #((S_HSV_binary == 1) | (V_binary == 1))]=1

    binout = 'outputvids/binout'+str(imagecount)+'.jpg'
    plt.imsave(binout,combined_binary)
    cv2.waitKey(500)
    #plt.figure()
    #plt.imshow(combined_binary)

    return combined_binary
```

3. Describe how (and identify where in your code) you performed a perspective transform and provide an example of a transformed image.

The code is pretty straight forward, I got the M and MInv matrices and use those to warp the image. MInv is used later on to unwarp the transformed image after it has been used to detect lane lines.

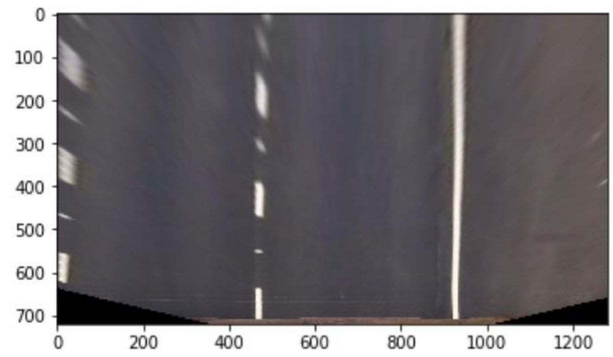
Here's where I apply a perspective transform and then pass that image into the thresholding function:

```
553 def warp(img):
554     global imagecount
555 
556     img = cv2.undistort(img, mtx, dist, None, mtx)
557     origimgout = 'outputvids/origfinal'+str(imagecount)+'.jpg'
558     plt.imsave(origimgout,img)
559     cv2.waitKey(500)
560     #plt.figure()
561     #plt.imshow(img)
562 
563     imagecount = imagecount + 1
564 
565     img_size = (img.shape[1], img.shape[0])
566     #print(img_size)
567     yoffset=470
568     src = np.float32(
569         [[180,img.shape[0]],
570          [575,yoffset],
571          [700,yoffset],
572          [1100,img.shape[0]]])
573 
574     dst = np.float32(
575         [[390, img.shape[0]],
576          [430, 0],
577          [920, 0],
578          [990, img.shape[0]]])
579 
580     binary_img = pipeline(img, s_thresh=(90, 255), sx_thresh=(20, 100))
581     binaryimgout = 'outputvids/binaryoutfinal'+str(imagecount)+'.jpg'
582     plt.imsave(binaryimgout,binary_img)
583     #plt.figure()
584     #plt.imshow(binary_img)
585     cv2.waitKey(500)
586 
```

One thing I had to do was make sure to reduce the height of the top pixels because in my previous submission of this project, I was looking too far down the lane, and it was leading to blurry lane detections.

I also perform image warping on the thresholded image (line 579), this also prevents blurring of the lanes, which later on creates a problem because if the image you are feeding into your histogram search code is blurry, then the code cannot find lane lines.

Image of the perspective transform after it has been applied:



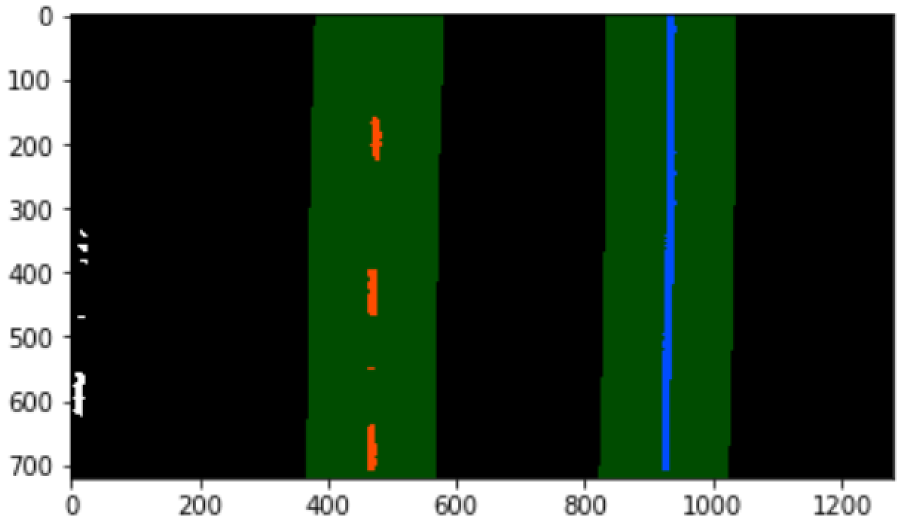
4. Describe how (and identify where in your code) you identified lane-line pixels and fit their positions with a polynomial?

The code I used to identify lane line pixels and then fit a polynomial to them is essentially exactly the same as the code provided in the course content. The thresholding I applied to extract lane lines from the video is powerful enough to do a good job and as such, I didn't have to touch this code:

The code below basically grabs lane line indices from each frame and then fits a polynomial to the x and y pixel positions. Then x and y values for plotting are generated and an image is output with all of this new information on it.

```
    # Extract left and right line pixel positions
    leftx = nonzerox[left_lane_inds]
    lefty = nonzeroy[left_lane_inds]
    rightx = nonzerox[right_lane_inds]
    righty = nonzeroy[right_lane_inds]

    # Fit a second order polynomial to each
    left_fit = np.polyfit(lefty, leftx, 2)
    right_fit = np.polyfit(righty, rightx, 2)

    # Generate x and y values for plotting
    ploty = np.linspace(0, binary_warped.shape[0]-1, binary_warped.shape[0] )
    left_fitx = left_fit[0]*ploty**2 + left_fit[1]*ploty + left_fit[2]
    right_fitx = right_fit[0]*ploty**2 + right_fit[1]*ploty + right_fit[2]

    # Create an image to draw on and an image to show the selection window
    out_img = np.dstack((binary_warped, binary_warped, binary_warped))*255
    window_img = np.zeros_like(out_img)
    # Color in left and right line pixels
    out_img[nonzeroy[left_lane_inds], nonzerox[left_lane_inds]] = [255, 0, 0]
    out_img[nonzeroy[right_lane_inds], nonzerox[right_lane_inds]] = [0, 0, 255]

    # Generate a polygon to illustrate the search window area
    # And recast the x and y points into usable format for cv2.fillPoly()
    left_line_window1 = np.array([np.transpose(np.vstack([left_fitx-margin, ploty]))])
    left_line_window2 = np.array([np.flipud(np.transpose(np.vstack([left_fitx+margin,
                                  ploty])))])
```

Expected output:



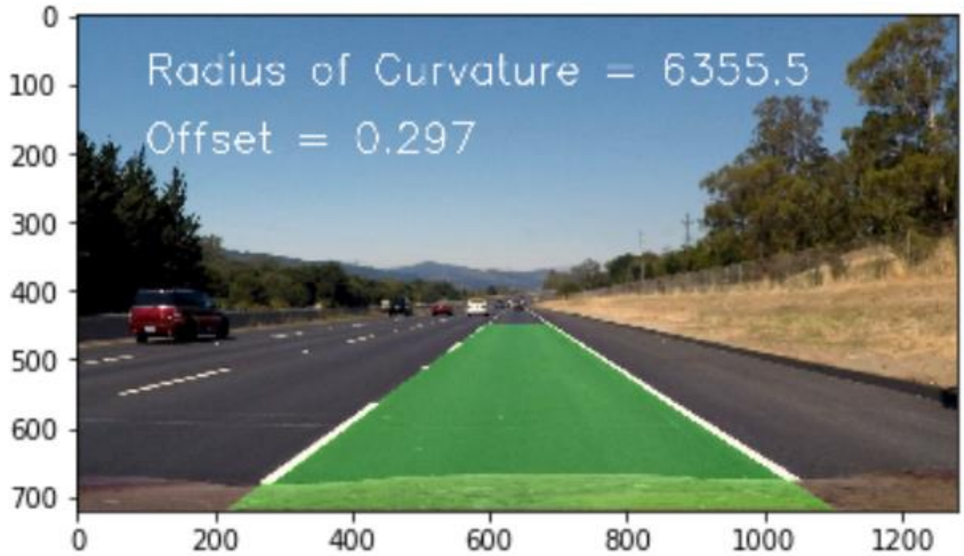
5. Describe how (and identify where in your code) you calculated the radius of curvature of the lane and the position of the vehicle with respect to center.

The code to calculate the radius of curvature for the left and right lane is as given in the course content. Basically, take the meters per pixel conversion to calculate the radius of curvature in the world space, and then take the average of the left and right radii of curvature to get the curvature of the center of the vehicle.

For the offset, I grab the center position of the vehicle by splitting the image into two, and this is the ideal center or the ground truth of the lane (assuming the camera is exactly in the center of the car). Then calculate the center of the lane the vehicle is driving on in the bottom row of the image and divide this by two to get the center of the lane. Then just subtracting these values and taking the absolute value gives the offset result.

```
    centerpos = binary_warped.shape[1]/2
    lane_center = (right_fitx[719] + left_fitx[719])/2
    center_offset_pixels = abs(centerpos - lane_center)

    y_eval = np.max(ploty)

    #Offset Calculation
    # Define conversions in x and y from pixels space to meters
    ym_per_pix = 30/720 # meters per pixel in y dimension
    xm_per_pix = 3.7/700 # meters per pixel in x dimension

    # Fit new polynomials to x,y in world space
    #print(len(ploty),'y')
    #print(len(left_fitx),'x')

    left_fit_cr = np.polyfit(ploty*ym_per_pix, left_fitx*xm_per_pix, 2)
    right_fit_cr = np.polyfit(ploty*ym_per_pix, right_fitx*xm_per_pix, 2)
    # Calculate the new radii of curvature

    left_curverad = ((1 + (2*left_fit_cr[0]*y_eval*ym_per_pix + left_fit_cr[1])**2)**1.5) / np.absolute(2*left_fit_cr[0])
    right_curverad = ((1 + (2*right_fit_cr[0]*y_eval*ym_per_pix + right_fit_cr[1])**2)**1.5) / np.absolute(2*right_fit_cr[0]
    # Now our radius of curvature is in meters
    #print(left_curverad, 'm', right_curverad, 'm')
    #print(int((left_curverad+right_curverad)/2.0+0.5),'avg')
    # Example values: 632.1 m    626.2 m
    curvatureavg = int(left_curverad+right_curverad)/2.0+0.5
    offsetval = (center_offset_pixels*xm_per_pix)*1.00
```

6. Provide an example image of your result plotted back down onto the road such that the lane area is identified clearly.



7. Briefly discuss any problems / issues you faced in your implementation of this project. Where will your pipeline likely fail? What could you do to make it more robust?

In my previous submission of this project, I was requested to resubmit due to the following issues in the video:

To correct these issues, I was told by folks on the Forum to average out some parts of the data being tracked to help bugger dark segments and glitch segments of the video that had missed lane line readings.

So I added averaging on the left and right lane coefficients and then added sanity checks for lane detection, if the left or right lane were not detected correctly, then I would ignore the image and use the previous good image to detect lanes on. Similarly, another sanity check was if the lower part of the lane was detected to be wider than the upper part of the lane, and the radius of curvature of the left lane was detected to be smaller than the radius of curvature for the right lane, then this is also a bad image and I would scrap it and use the last good one. I created a new function called processimage that will re-generate the lane drawings on the img (same code as provided in the course), except I pass in the previous img to generate lane lines on top of that img.
Image of averaging and sanity check section of my pipeline:

```python

    if (len(leftx) < 2200 or len(rightx) < 2250) or (left_curverad < right_curverad and lane_wid_up > lane_wid_dwn) :
        #do this if the image is a failure

        left_lane_inds = prev_left_lane_inds
        right_lane_inds = prev_right_lane_inds
        left_fit = total_avg_left_fit
        right_fit = total_avg_right_fit
        nonzero = prev_binary_warped_img.nonzero()
        nonzeroy = np.array(nonzero[0])
        nonzerox = np.array(nonzero[1])
        result, curvatureavg, offsetval = processimage(img, prev_binary_warped_img, left_fit, right_fit, left_lane_inds, righ

        return result, curvatureavg, offsetval, lane_wid_up, lane_wid_dwn
    else:

        prev_left_fit = left_fit
        prev_right_fit = right_fit

        prev_left_lane_inds = left_lane_inds
        prev_right_lane_inds = right_lane_inds

        prev_binary_warped_img = binary_warped

        if imagecount > 1 :
            Left_newarray = []
            for x, y in zip(total_avg_left_fit,left_fit):
                #print('x',x,'y',y)
                avg = (x + y )/ 2
                #print('x + y / 2', avg)
                Left_newarray.append(avg)
            #print(Left_newarray)
            total_avg_left_fit = Left_newarray

            Right_newarray = []
            for x, y in zip(total_avg_right_fit,right_fit):
                #print('x',x,'y',y)
                avg = (x + y )/ 2
                #print('x + y / 2', avg)
                Right_newarray.append(avg)
            #print(Right_newarray)
            total_avg_right_fit = Right_newarray
        else:
            total_avg_left_fit = left_fit
            total_avg_right_fit = right_fit

        #print('total avg left fit', total_avg_left_fit)
        #print('total avg right fit', total_avg_right_fit)
        #return this image

    #lane draw
    # Create an image to draw the lines on
    warp_zero = np.zeros_like(binary_warped).astype(np.uint8)
    color_warp = np.dstack((warp_zero, warp_zero, warp_zero))
```

Image of processimage function:

```
def processimage(img, binary_warped, left_fit, right_fit, left_lane_inds, right_lane_inds, total_avg_left_fit, nonzeroy, non

    # Extract left and right line pixel positions
    leftx = nonzerox[left_lane_inds]
    lefty = nonzeroy[left_lane_inds]
    rightx = nonzerox[right_lane_inds]
    righty = nonzeroy[right_lane_inds]

    # Generate x and y values for plotting
    ploty = np.linspace(0, binary_warped.shape[0]-1, binary_warped.shape[0] )
    left_fitx = left_fit[0]*ploty**2 + left_fit[1]*ploty + left_fit[2]
    right_fitx = right_fit[0]*ploty**2 + right_fit[1]*ploty + right_fit[2]

    centerpos = binary_warped.shape[1]/2
    lane_center = (right_fitx[719] + left_fitx[719])/2
    center_offset_pixels = abs(centerpos - lane_center)

    y_eval = np.max(ploty)

    #Offset Calculation
    # Define conversions in x and y from pixels space to meters
    ym_per_pix = 30/720 # meters per pixel in y dimension
    xm_per_pix = 3.7/700 # meters per pixel in x dimension

    # Fit new polynomials to x,y in world space
    #print(len(ploty),'y')
    #print(len(left_fitx),'x')

    left_fit_cr = np.polyfit(ploty*ym_per_pix, left_fitx*xm_per_pix, 2)
    right_fit_cr = np.polyfit(ploty*ym_per_pix, right_fitx*xm_per_pix, 2)
    # Calculate the new radii of curvature

    left_curverad = ((1 + (2*left_fit_cr[0]*y_eval*ym_per_pix + left_fit_cr[1])**2)**1.5) / np.absolute(2*left_fit_cr[0])
    right_curverad = ((1 + (2*right_fit_cr[0]*y_eval*ym_per_pix + right_fit_cr[1])**2)**1.5) / np.absolute(2*right_fit_cr[0]
    # Now our radius of curvature is in meters
    #print(left_curverad, 'm', right_curverad, 'm')
    #print(int((left_curverad+right_curverad)/2.0+0.5),'avg')
    # Example values: 632.1 m    626.2 m
    curvatureavg = int(left_curverad+right_curverad)/2.0+0.5
    offsetval = (center_offset_pixels*xm_per_pix)*1.00
    #print(round(offsetval,3),'offset')

    # Create an image to draw on and an image to show the selection window
    out_img = np.dstack((binary_warped, binary_warped, binary_warped))*255
    window_img = np.zeros_like(out_img)
    # Color in left and right line pixels
    out_img[nonzeroy[left_lane_inds], nonzerox[left_lane_inds]] = [255, 0, 0]
    out_img[nonzeroy[right_lane_inds], nonzerox[right_lane_inds]] = [0, 0, 255]

    # Generate a polygon to illustrate the search window area
    # And recast the x and y points into usable format for cv2.fillPoly()
    left_line_window1 = np.array([np.transpose(np.vstack([left_fitx-margin, ploty]))])
    left_line_window2 = np.array([np.flipud(np.transpose(np.vstack([left_fitx+margin,
                                  ploty])))])
```