**Camera Calibration:**
This was straight-forward code given to us from the lesson, a couple of simple lines the find the chessboard corners and getting the distortion matrix using cv2.calibrateCamera. Then call cv2.undistort. Example of these are in output_images/undist_ChessBoard*.jpg. Undistorted test images called undist*.jpg.

**Binary Image:**
I used a combination of all of the methods used in the lesson. I used the x, y sobel to find the gradients in the x, y directions with relatively low thresholds. Also discussed in the lesson I took the images in different color spaces (HLS, HSV) and looked at the channels that showed the lane lines the brightest (saturation and value channels) with larger thresholds. Then I just added all of the pixels we found together to form our binary image. These images are binary*.jpg.

**Warped Images:**
This was pretty much given from the lesson, all we had to do was define the area of interest and the destination and call cv2.getPerspectiveTransform and the cv2.warpPerspective. Images saved as warped*.jpg

**Finding Lanes:**
I took the window tracker code from the lesson and used it to find the centers of each window collected for the left and right lines. Then fed those points into np.polyfit to get the coefficients. The tracker code walked across each window and convolved the signal to basically sum up each of the pixel values and take the max point of the convolution as the center. The code for visualizing was also given in the lesson. Saved as windows*.jpg. Then you run the polynomial function through the image to find the points of the lanes, then draw a polygon outlined by the lanes. Un-warp the perspective using the inverse distortion and add the images together. The polygon images are drawn*.jpg and the added together image are withPoly*.jpg.

**Radius of Curvature:**
The radius was calculated using the left and right lane line pixels adjusted for the pixel conversion factors given in the lesson, the equations were implemented and I chose to average the curvature given by the left and right for a single curvature number. The center calculation was just finding the how offset the lane lines were in the frame. Images are radius*.jpg.

**Output:**
Final output test images are radius*.jpg

**Discussion:**
This pipeline didn't work so well on the challenge video and I have a feeling that making a routine to dynamically change the area of interest for the perspective transform would get better results. Perhaps dynamic thresholding for the sobel and color thresholding would be useful, maybe look at the relative darkness/lightness of the image changing the thresholds according to

that value.  One of the things I thought was really interesting was the averaging the lane lines throughout frames in an attempt to smooth the lines.  But if the lines are changing rapidly then this would hurt the performance, so perhaps you could only use the average if the new lines are within a threshold range.