This is the write-up report for project1 Finding Lane Lines on the Road.

Change in Helper Function

There are 3 functions with 1 additional import added:

```
from scipy import stats
def draw_lines_improve(img, lines, color=[255, 0, 255], thickness=10):
def column(matrix, i):
def filter(img):
```

"draw_lines_improve()" is the function modified, improved from original "draw_line()" function
"import stats" imports stat library to use stat regression model to estimate slope and intercept of lines. In order to use this import, scipy needed to be installed.
"column()" function is to aid some logics in "draw_lines_improve()" when to separate x-coordinates and y-coordinates out of lines.
"filter()" function is the additional improvement to solve optional challenge by modifying image/video by detecting yellow/white line from converting BGR to HSV and filtering.

Test Images

This section has been the primary place to debug using an image or a video capture to see where in pipeline seem to produce bad result. Logics here are similar as in pipeline.

Build a Lane Finding Pipeline

Pipeline logics consist of what's covered in the class – grayscale, Gaussian, canny edge, masking, hough line and weighted (draw hough lines on original image/video)

Test on Videos

This section was simply to focus on make the existing logic work.
Had to fix many syntax errors, install missing libraries/packages.

Improve the draw_lines() function

At the beginning, averaging line values were used. It was good enough in "solidWhiteRight.mp4" but was too unstable in "solidYellowLeft.mp4."
Thus, regression model is used instead as mentioned in "import stats" above.

Also, following logic condition is used to distinguish whether the line is left line or right line.

```
for line in lines:
    for x1,y1,x2,y2 in line:
        slope = (y2-y1)/(x2-x1)
    if slope > 0.5 and slope < 1.5 and x1 > half_line and x2 > half_line:
        right_lines.append([x1,y1,x2,y2])
    elif slope < -0.25 and slope > -1.25 and x1 < half_line and x2 < half_line:
        left_lines.append([x1,y1,x2,y2])
```

Slope enforcement conditions are for filtering out lines with extreme slope values, under an assumption the slope of the road lane in the frames are consistent as camera mounting location and there is no sudden turning.

Half_line is simply a divider that distinguish whether the line is on the left or on the right of the image/video.

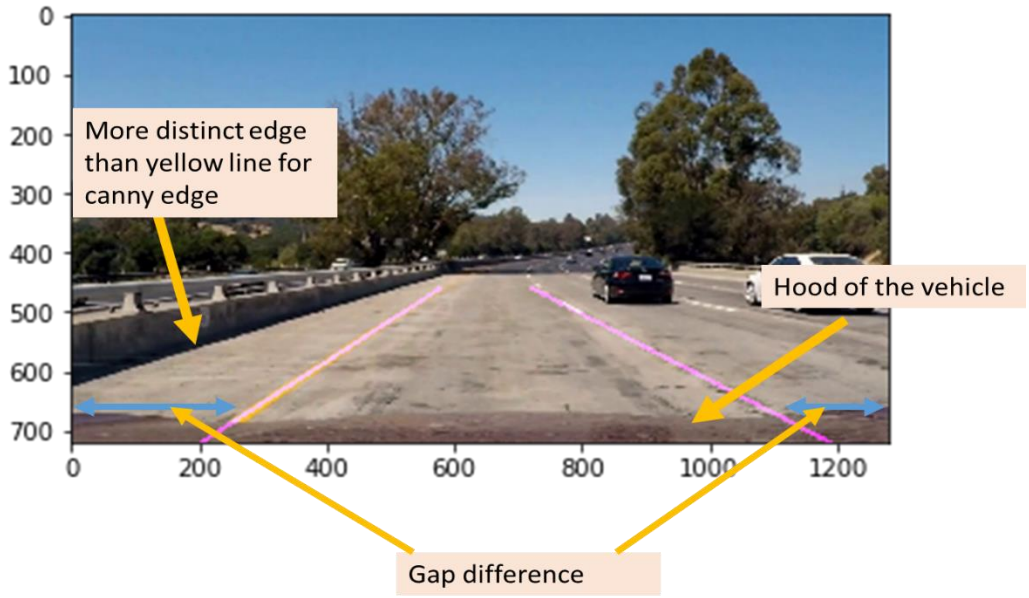
One tricky part was, since y-axis increased as to the bottom in this system.
This is different from traditional xy-axis system as y-axis is inverted,
making it quite counter-intuitive and confusing.

Optional Challenge

One of the issues found during the debugging process was the size difference of the video. The size of "challenge.mp4" is 720P (1280x720) compared to 980x540 in other videos.
This made masking the region of interest and draw line functions to go off.

There were several other issues found including (see the capture image below for some of issues):

1. The hood of vehicle takes about 50 pixels of the bottom of video.
2. The road is curved to the right and slanted slightly, creating the gap difference between lines and edge of the video.
3. There is central reservation divider on the left has more distinct edge than yellow lane.
4. There is a section of the video where the white lane on the right almost disappears out of region of interest.
5. Some debris on the road are very white
6. Road color/shadow changes



To resolve these issues,
Several if conditions were used to adjust values base on different sizes.
For example in "pipeline()" logic,

```
vertices = np.array([[(50,y2-50),(x1, y1), (x2, y1), (imshape[1]-50,y2-50)]], dtype=np.int32)
if imshape[1] == 1280:
    vertices = np.array([[(300,y2-70),(x1, 450), (x2, 450), (imshape[1]-100,y2-70)]], dtype=np.int32)
```

As you can see, there are some adjustment of variables/hard code values to adjust vertices for the region of interest.

Similar condition is applied in "draw_lines_improve()" function when determining the line coorinates to draw base on regression values.

Conclusion and some thoughts

The code logic is not perfect as the solid lines are little shaking in "challenge.mp4" video.
Sure it could be improved further at least on the videos given, and I will continue refining.

Also, it is expected to be vulnerable in cases such as – when there is white truck is on the front, there is no road lanes exist, light reflection from mirror blurring the camera focus, different weather conditions, etc. However, I know this can be improved as sensor fusion and deep learning comes into play.