CV Model

```
import numpy as np
import pandas as pd
import cv2
from sklearn.linear_model import RANSACRegressor
from sklearn.preprocessing import PolynomialFeatures

# Libraries needed to edit/save/watch video clips
from matplotlib import pyplot as plt
import matplotlib.lines as mlines

capture = cv2.VideoCapture('test_video.mp4')
while True:
    isTrue, frame = capture.read()
    if not isTrue:
        break

    rgb_image = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
    cv2.imshow('lanes', frame)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

capture.release()
cv2.destroyAllWindows()


def perspective_warp(img, dst_size=(1280, 720),
                     src=np.float32([(0.30, 0.58), (0.70, 0.58), (0.1, 0.8), (1, 0.8)]),
                     dst=np.float32([(0, 0), (1, 0), (0, 1), (1, 1)])):
    img_size = np.float32([(img.shape[1], img.shape[0])])
    src = src * img_size
    dst = dst * np.float32(dst_size)
    M = cv2.getPerspectiveTransform(src, dst)
    warped = cv2.warpPerspective(img, M, dst_size)
    return warped


def sobel_filtering(warped_img):
    hls_image = cv2.cvtColor(warped_img, cv2.COLOR_BGR2HLS)
    h, l, s = cv2.split(hls_image)
    sobel_x_s = cv2.Sobel(s, cv2.CV_64F, 1, 0)
    sobel_x_l = cv2.Sobel(l, cv2.CV_64F, 1, 0)
    abs_sobel_x_s = np.absolute(sobel_x_s)
    abs_sobel_x_l = np.absolute(sobel_x_l)
    s_threshold = 50  # Adjust the threshold values as needed
    l_threshold = 50
    s_binary = np.zeros_like(s, dtype=np.uint8)
    s_binary[(abs_sobel_x_s >= s_threshold)] = 1
    l_binary = np.zeros_like(l, dtype=np.uint8)
    l_binary[(abs_sobel_x_l >= l_threshold)] = 1
```

```
    # Combine binary images for saturation and lightness channels
    combined_binary = cv2.bitwise_or(s_binary, l_binary)
    binary_image = np.array(combined_binary, dtype=np.uint8) * 255
    return binary_image


def sliding_window(binary_image, n_windows=9, margin=100, min_pixels=100):
    # Set height of windows
    window_height = binary_image.shape[0] // n_windows

    # Identify the x and y positions of all nonzero pixels in the image
    nonzero = binary_image.nonzero()
    nonzeroy = np.array(nonzero[0])
    nonzerox = np.array(nonzero[1])

    # Current positions to be updated later for each window
    leftx_current = int(binary_image.shape[1] * 0.25)
    rightx_current = int(binary_image.shape[1] * 0.75)

    # Set the width of the windows +/- margin
    margin = margin

    # Set minimum number of pixels found to recenter window
    minpix = min_pixels

    # Create empty lists to receive left and right lane pixel indices
    left_lane_inds = []
    right_lane_inds = []

    # Step through the windows one by one
    for window in range(n_windows):
        # Identify window boundaries in x and y (and right and left)
        win_y_low = binary_image.shape[0] - (window + 1) * window_height
        win_y_high = binary_image.shape[0] - window * window_height

        win_xleft_low = leftx_current - margin
        win_xleft_high = leftx_current + margin
        win_xright_low = rightx_current - margin
        win_xright_high = rightx_current + margin

        # Identify the nonzero pixels in x and y within the window
        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]
```

```
        # Append these indices to the lists
        left_lane_inds.append(good_left_inds)
        right_lane_inds.append(good_right_inds)

        # If found > minpix pixels, recenter next window on their mean position
        if len(good_left_inds) > minpix:
            leftx_current = int(np.mean(nonzerox[good_left_inds]))
        if len(good_right_inds) > minpix:
            rightx_current = int(np.mean(nonzerox[good_right_inds]))

    # Concatenate the arrays of indices
    left_lane_inds = np.concatenate(left_lane_inds)
    right_lane_inds = np.concatenate(right_lane_inds)

    # Extract pixel coordinates for left and right lanes
    left_x = nonzerox[left_lane_inds]
    left_y = nonzeroy[left_lane_inds]
    right_x = nonzerox[right_lane_inds]
    right_y = nonzeroy[right_lane_inds]

    return left_x, left_y, right_x, right_y


def fit_parabolic_curves(left_x, left_y, right_x, right_y):
    # Fit parabolic curve for left lane
    left_curve = np.polyfit(left_y, left_x, 2)
    left_curve_fn = np.poly1d(left_curve)

    # Fit parabolic curve for right lane
    right_curve = np.polyfit(right_y, right_x, 2)
    right_curve_fn = np.poly1d(right_curve)

    return left_curve_fn, right_curve_fn


capture = cv2.VideoCapture('test_video.mp4')

# Get video properties
fps = capture.get(cv2.CAP_PROP_FPS)
width = int(capture.get(cv2.CAP_PROP_FRAME_WIDTH))
height = int(capture.get(cv2.CAP_PROP_FRAME_HEIGHT))
fourcc = cv2.VideoWriter_fourcc(*'mp4v')
output_video = cv2.VideoWriter('output_video.mp4', fourcc, fps, (width, height), isColor=False)

# Create the structuring element for morphology operation
kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (3, 3))

while True:
    isTrue, frame = capture.read()
    if not isTrue:
        break

    # Apply perspective warp
    warped_img = perspective_warp(frame)
```

```
    # Apply Sobel filtering and thresholding
    binary_image = sobel_filtering(warped_img)

    left_x, left_y, right_x, right_y = sliding_window(binary_image)

    # Fit parabolic curves using RANSAC
    left_curve_fn, right_curve_fn = fit_parabolic_curves(left_x, left_y, right_x, right_y)

    # Generate x and y values for plotting the curves
    plot_y = np.linspace(0, binary_image.shape[0] - 1, binary_image.shape[0])
    left_fit_x = left_curve_fn(plot_y)
    right_fit_x = right_curve_fn(plot_y)

    # Create an empty image for drawing the fitted curves
    out_img = np.dstack((binary_image, binary_image, binary_image)) * 255

    # Draw left curve in red
    for i, j in zip(left_fit_x.astype(int), plot_y.astype(int)):
        cv2.circle(out_img, (i, j), 2, (255, 0, 0), -1)

    # Draw right curve in blue
    for i, j in zip(right_fit_x.astype(int), plot_y.astype(int)):
        cv2.circle(out_img, (i, j), 2, (0, 0, 255), -1)

    # Resize the frame to match the dimensions of out_imgbreak
    frame = cv2.resize(frame, (out_img.shape[1], out_img.shape[0]))

    # Apply curve fitting to the output of sliding window
    frame[left_y, left_x] = [255, 0, 0]  # Left curve in red
    frame[right_y, right_x] = [0, 0, 255]  # Right curve in blue

    # Overlay the output of sliding window on the original frame
    out_img = cv2.addWeighted(frame, 1, out_img, 0.5, 0)

    cv2.imshow('lanes', out_img)

    # Write the frame to the output video
    output_video.write(out_img)

    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

capture.release()
output_video.release()
cv2.destroyAllWindows()
```