In [1]:

```
!pip install --upgrade "protobuf<=3.20.1" --user
```

```
Requirement already satisfied: protobuf<=3.20.1 in c:\users\user\downloads\n
ew folder\lib\site-packages (3.20.1)
```

In [3]:

```
mp_drawing = mp.solutions.drawing_utils
mp_pose = mp.solutions.pose
pose = mp_pose.Pose(min_detection_confidence=0.5, min_tracking_confidence=0.5)
```

In [2]:

```
import cv2
import math
import numpy as np
import mediapipe as mp
from time import time
import matplotlib.pyplot as plt
```

In [3]:

```
mp_pose=mp.solutions.pose
pose=mp_pose.Pose(static_image_mode=True,min_detection_confidence=0.5,min_tracking_confiden
mp_drawing=mp.solutions.drawing_utils
```

In [ ]:

In [4]:

```
def detectPose(image, pose, display=True):

    # Create a copy of the input image.
    output_image = image.copy()

    # Convert the image from BGR into RGB format.
    imageRGB = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

    # Perform the Pose Detection.
    results = pose.process(imageRGB)

    # Retrieve the height and width of the input image.
    height, width, _ = image.shape

    # Initialize a list to store the detected landmarks.
    landmarks = []

    # Check if any landmarks are detected.
    if results.pose_landmarks:

        # Draw Pose landmarks on the output image.
        mp_drawing.draw_landmarks(image=output_image, landmark_list=results.pose_landmarks,
                                  connections=mp_pose.POSE_CONNECTIONS)

        # Iterate over the detected landmarks.
        for landmark in results.pose_landmarks.landmark:

            # Append the landmark into the list.
            landmarks.append((int(landmark.x * width), int(landmark.y * height),
                              (landmark.z * width)))

    # Check if the original input image and the resultant image are specified to be display
    if display:

        # Display the original input image and the resultant image.
        plt.figure(figsize=[22,22])
        plt.subplot(121);plt.imshow(image[:,:,::-1]);plt.title("Original Image");plt.axis('
        plt.subplot(122);plt.imshow(output_image[:,:,::-1]);plt.title("Output Image");plt.a

        # Also Plot the Pose landmarks in 3D.
        mp_drawing.plot_landmarks(results.pose_world_landmarks, mp_pose.POSE_CONNECTIONS)

    # Otherwise
    else:

        # Return the output image and the found landmarks.
        return output_image, landmarks
```

In [ ]:

```

```

In [5]:

```
def calculateAngle(landmark1,landmark2,landmark3):
 x1,y1,_=landmark1
 x2,y2,_=landmark2
 x3,y3,_=landmark3

 angle=math.degrees(math.atan2(y3-y2,x3-x2)- math.atan2(y1-y2,x1-x2))

 if angle<0:
   angle+=360

 return angle
```

In [6]:

```
angle=calculateAngle((558,326,0),(642,333,0),(718,321,0))
print(f'The calculated angle is {angle}')
```

```
The calculated angle is 166.26373169437744
```

In [ ]:

```

```

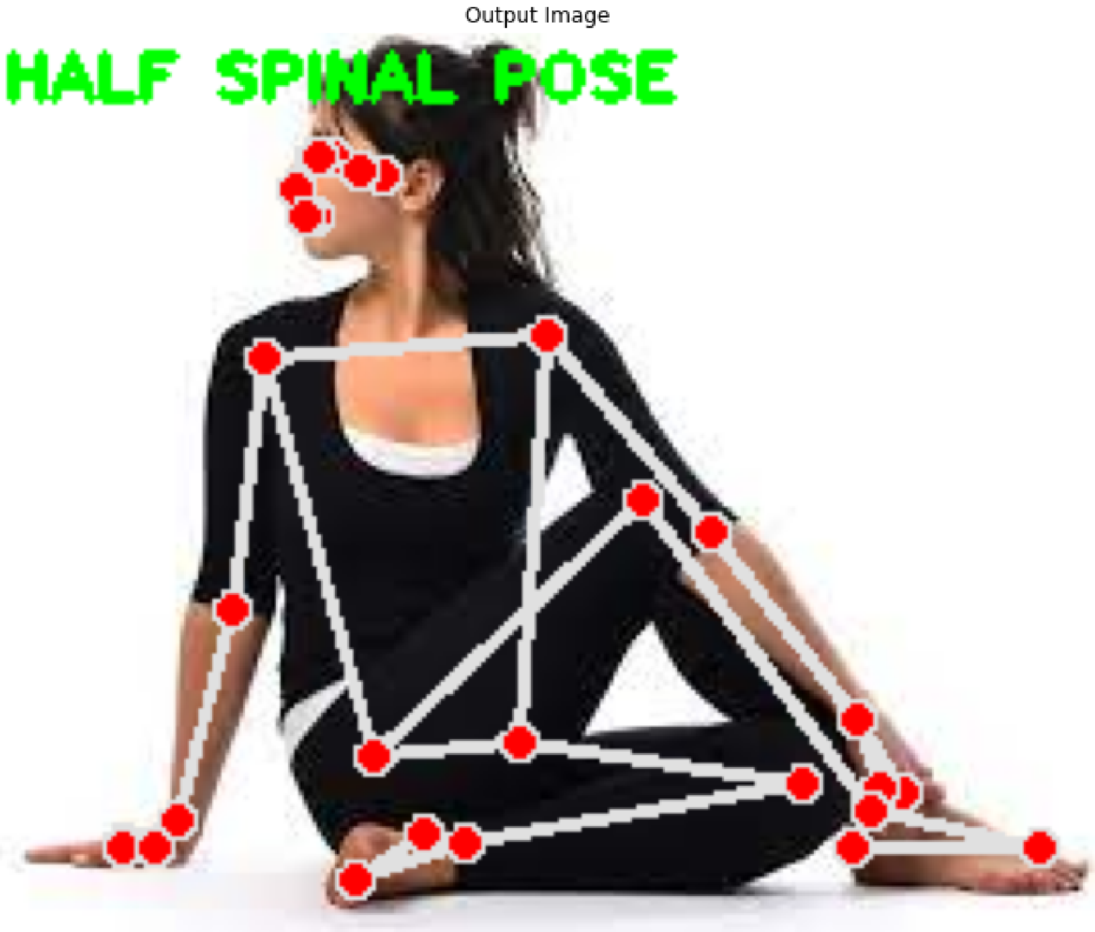
In [300]:

```
def classifyPose(landmarks, output_image, display=False):

    # Initialize the label of the pose. It is not known at this stage.
    label = 'UNKNOWN POSE'

    # Specify the color (Red) with which the label will be written on the image.
    color = (0, 0, 255)

    # Calculate the required angles.
    #--------------------------------------------------------------------------------------

    # Get the angle between the left shoulder, elbow and wrist points.
    left_elbow_angle = calculateAngle(landmarks[mp_pose.PoseLandmark.LEFT_SHOULDER.value],
                                      landmarks[mp_pose.PoseLandmark.LEFT_ELBOW.value],
                                      landmarks[mp_pose.PoseLandmark.LEFT_WRIST.value])

    # Get the angle between the right shoulder, elbow and wrist points.
    right_elbow_angle = calculateAngle(landmarks[mp_pose.PoseLandmark.RIGHT_SHOULDER.value]
                                       landmarks[mp_pose.PoseLandmark.RIGHT_ELBOW.value],
                                       landmarks[mp_pose.PoseLandmark.RIGHT_WRIST.value])

    # Get the angle between the left elbow, shoulder and hip points.
    left_shoulder_angle = calculateAngle(landmarks[mp_pose.PoseLandmark.LEFT_ELBOW.value],
                                         landmarks[mp_pose.PoseLandmark.LEFT_SHOULDER.value
                                         landmarks[mp_pose.PoseLandmark.LEFT_HIP.value])

    # Get the angle between the right hip, shoulder and elbow points.
    right_shoulder_angle = calculateAngle(landmarks[mp_pose.PoseLandmark.RIGHT_HIP.value],
                                          landmarks[mp_pose.PoseLandmark.RIGHT_SHOULDER.val
                                          landmarks[mp_pose.PoseLandmark.RIGHT_ELBOW.value]

    # Get the angle between the left hip, knee and ankle points.
    left_knee_angle = calculateAngle(landmarks[mp_pose.PoseLandmark.LEFT_HIP.value],
                                     landmarks[mp_pose.PoseLandmark.LEFT_KNEE.value],
                                     landmarks[mp_pose.PoseLandmark.LEFT_ANKLE.value])

    # Get the angle between the right hip, knee and ankle points
    right_knee_angle = calculateAngle(landmarks[mp_pose.PoseLandmark.RIGHT_HIP.value],
                                      landmarks[mp_pose.PoseLandmark.RIGHT_KNEE.value],
                                      landmarks[mp_pose.PoseLandmark.RIGHT_ANKLE.value])

    left_heel_angle=calculateAngle(landmarks[mp_pose.PoseLandmark.LEFT_ANKLE.value],
                                      landmarks[mp_pose.PoseLandmark.LEFT_HEEL.value],
                                      landmarks[mp_pose.PoseLandmark.LEFT_FOOT_INDEX.value]

    right_heel_angle=calculateAngle(landmarks[mp_pose.PoseLandmark.RIGHT_ANKLE.value],
                                      landmarks[mp_pose.PoseLandmark.RIGHT_HEEL.value],
                                      landmarks[mp_pose.PoseLandmark.RIGHT_FOOT_INDEX.value

    right_hip_angle=calculateAngle(landmarks[mp_pose.PoseLandmark.RIGHT_SHOULDER.value],
                                      landmarks[mp_pose.PoseLandmark.RIGHT_HIP.value],
                                      landmarks[mp_pose.PoseLandmark.RIGHT_KNEE.value])

    left_hip_angle=calculateAngle(landmarks[mp_pose.PoseLandmark.LEFT_SHOULDER.value],
                                      landmarks[mp_pose.PoseLandmark.LEFT_HIP.value],
                                      landmarks[mp_pose.PoseLandmark.LEFT_KNEE.value])
    if right_shoulder_angle > 20 and right_shoulder_angle < 70:

                if right_elbow_angle > 180 and right_elbow_angle < 200:

                    if right_knee_angle > 260 and right_knee_angle < 280:

                        label = 'HALF SPINAL POSE'


    if right_shoulder_angle > 120 and right_shoulder_angle < 160:

                if right_elbow_angle > 150 and right_elbow_angle < 180:

                    if right_knee_angle > 230 and right_knee_angle < 270:

                        label = 'BOW POSE'



    if right_shoulder_angle > 180 and right_shoulder_angle < 220:

                if right_elbow_angle > 140 and right_elbow_angle < 180:

                    if right_knee_angle > 300 and right_knee_angle < 340:
                        # Specify the label of the pose that is Warior II pose.
                        label = 'CHILD POSE'



    if right_shoulder_angle > 180 and right_shoulder_angle < 220:

                if right_elbow_angle > 140 and right_elbow_angle < 180:

                    if right_knee_angle > 300 and right_knee_angle < 340:

                        label = 'CHILD POSE'



    if left_shoulder_angle > 10 and left_shoulder_angle < 40:

                if left_hip_angle > 230 and left_hip_angle < 270:

                    if left_knee_angle > 170 and left_knee_angle < 190:

                        label = 'LEGS UP THE WALL POSE'


    if left_shoulder_angle > 110 and left_shoulder_angle < 150:

                if left_hip_angle > 90 and left_hip_angle < 120:

                    if left_knee_angle > 140 and left_knee_angle < 180:

                        label = 'WHEEL POSE'


      # Check if the pose is classified successfully
    if label != 'UNKNOWN POSE':

        # Update the color (to green) with which the label will be written on the image.
        color = (0, 255, 0)

    # Write the label on the output image.
    cv2.putText(output_image, label, (10, 15),cv2.FONT_HERSHEY_PLAIN, 1, color, 2)

    # Check if the resultant image is specified to be displayed.
    if display:

        # Display the resultant image.
        plt.figure(figsize=[12,15])
        plt.imshow(output_image[:,:,::-1]);plt.title("Output Image");plt.axis('off');

    else:

        # Return the output image and the classified label.
        return output_image, label
```

In [ ]:

In [301]:

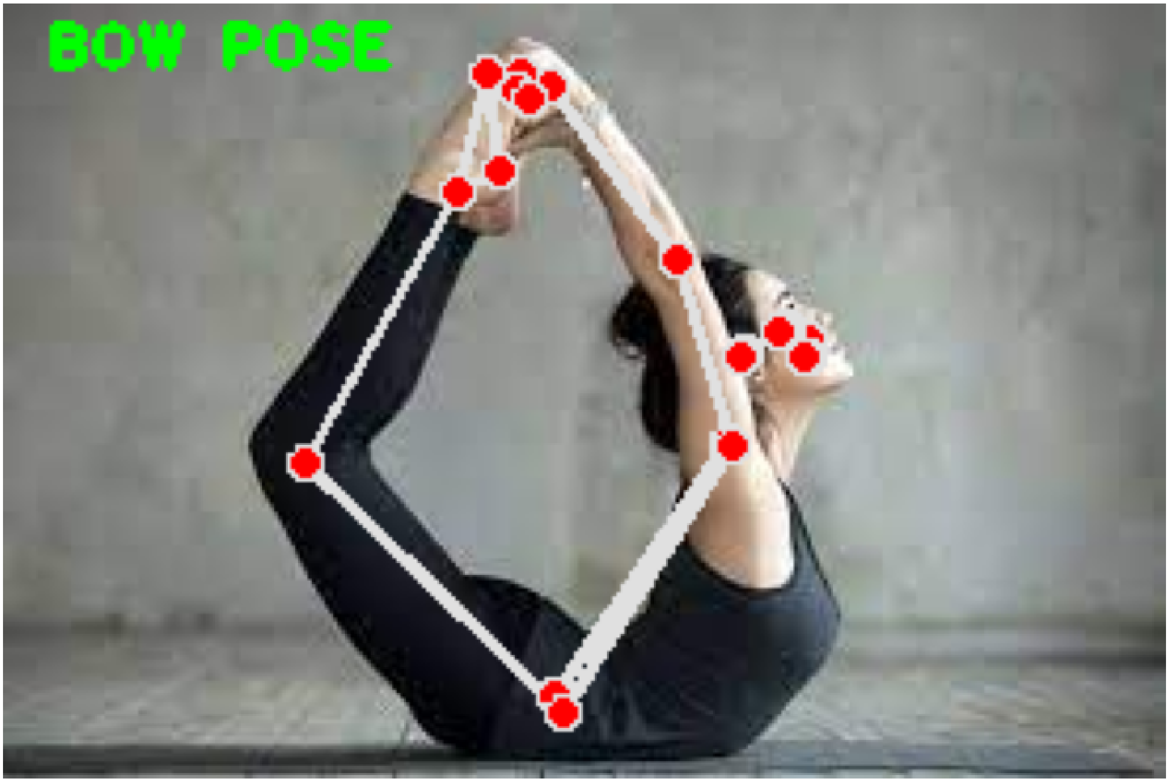
```
image=cv2.imread("half spinal twist.jpg")
output_image,landmarks=detectPose(image,pose,display=False)
if landmarks:
    classifyPose(landmarks,output_image,display=True)
```



In [302]:

```
image=cv2.imread("bow pose.jpg")
output_image,landmarks=detectPose(image,pose,display=False)
if landmarks:
    classifyPose(landmarks,output_image,display=True)
```

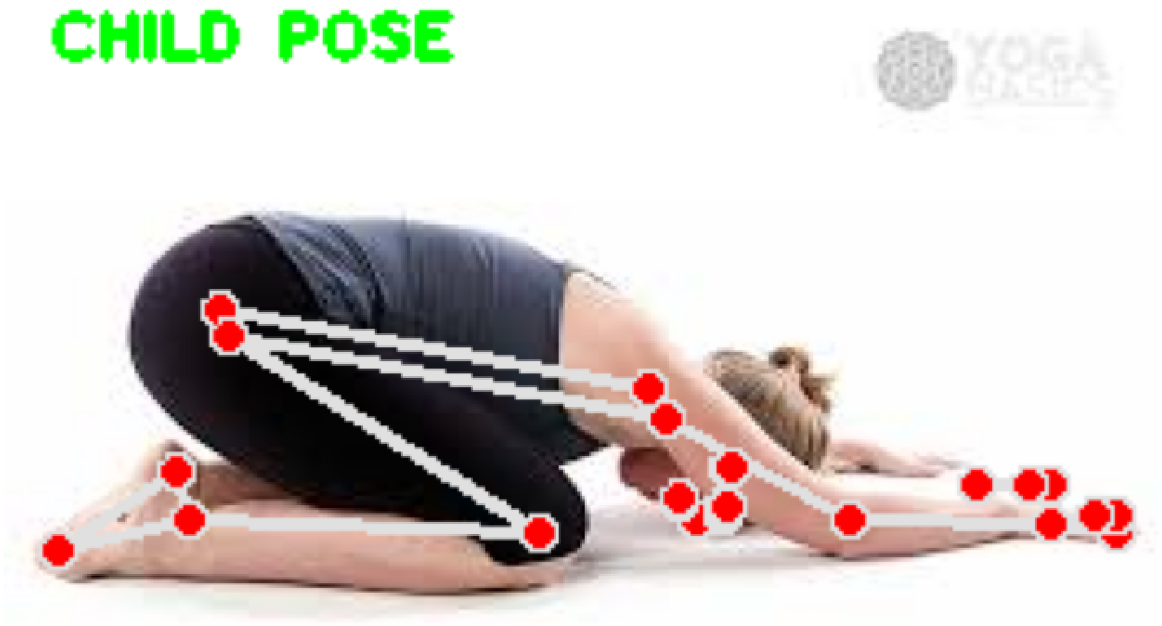
Output Image



In [303]:

```
image=cv2.imread("child pose.jpg")
output_image,landmarks=detectPose(image,pose,display=False)
if landmarks:
    classifyPose(landmarks,output_image,display=True)
```

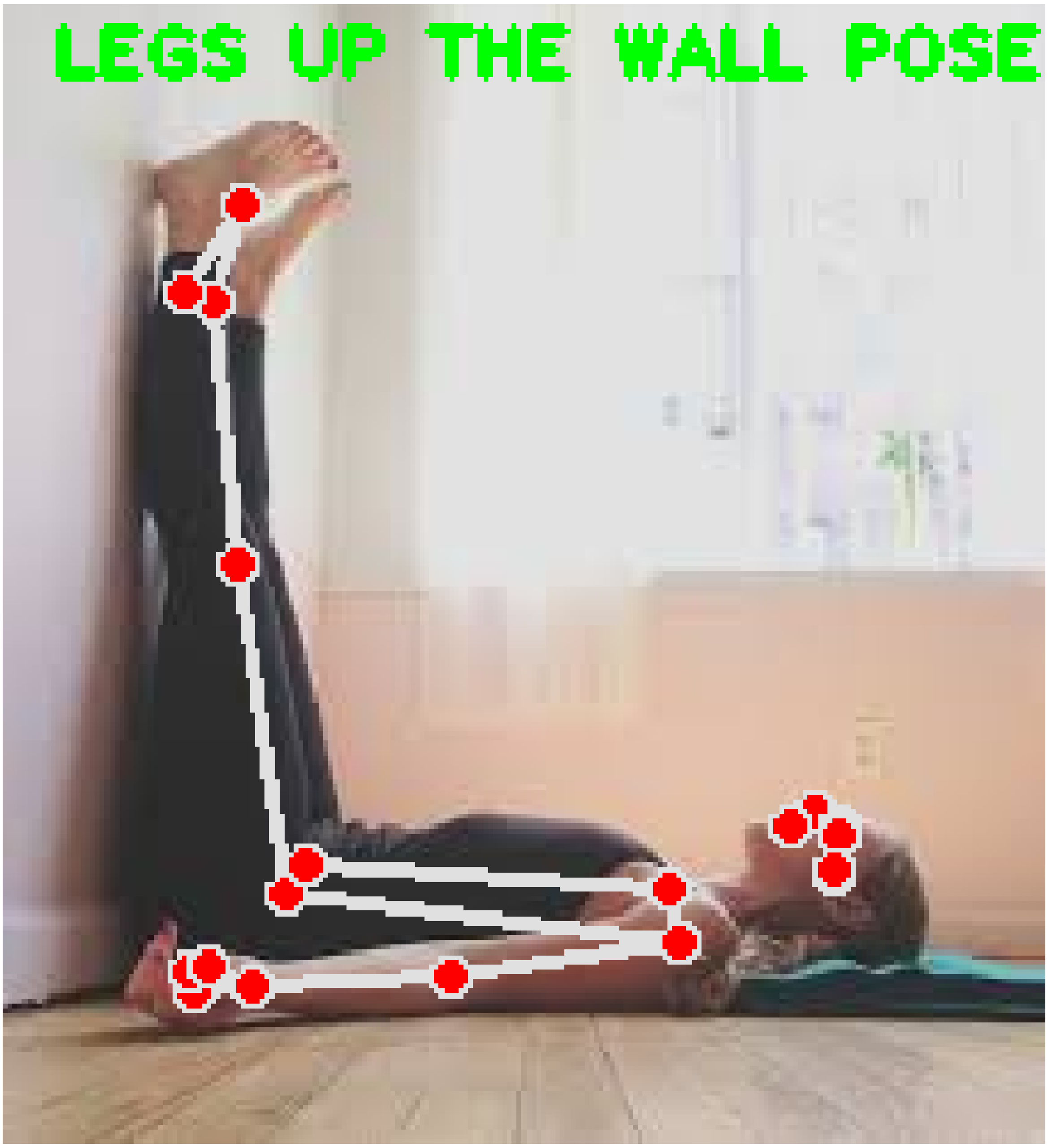
Output Image



In [304]:

```
image=cv2.imread("legs up the wall pose.jpg")
output_image,landmarks=detectPose(image,pose,display=False)
if landmarks:
    classifyPose(landmarks,output_image,display=True)
```

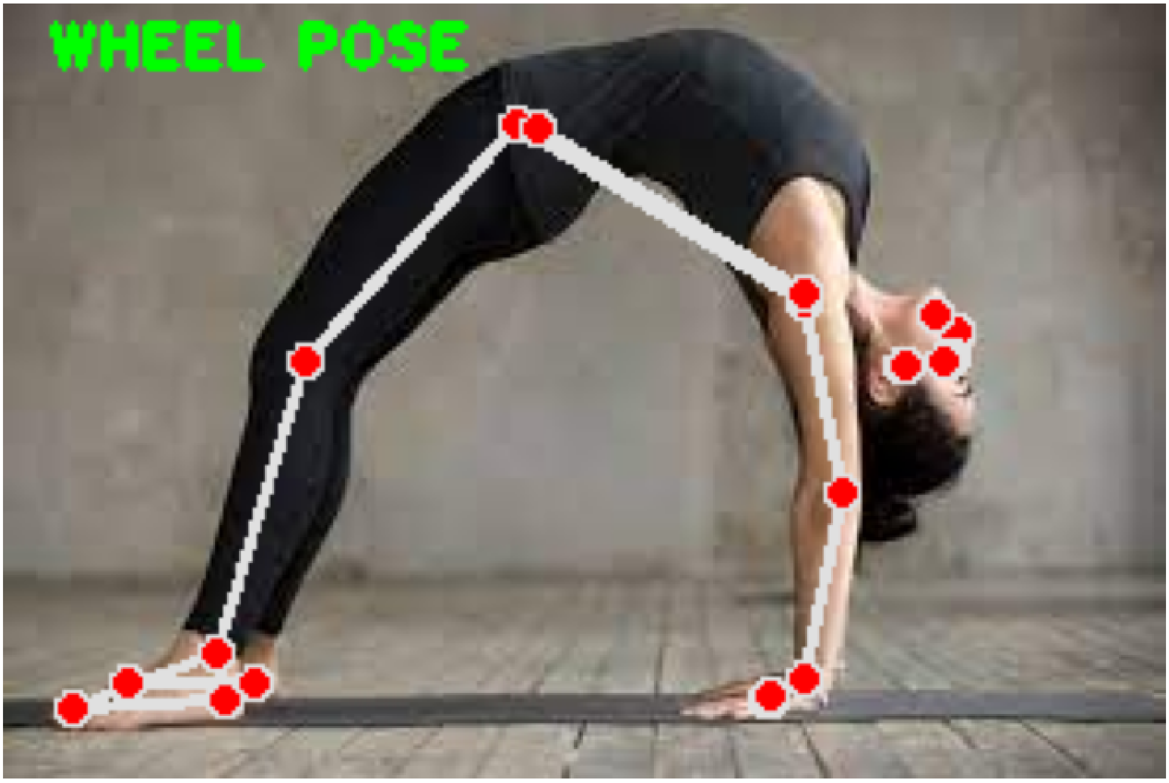
Output Image



In [305]:

```
image=cv2.imread("wheel pose.jpg")
output_image,landmarks=detectPose(image,pose,display=False)
if landmarks:
    classifyPose(landmarks,output_image,display=True)
```

Output Image



In [ ]:

```
```