Bhuvi Ghosh
60009210191

IPCV-II Miniproject

```
[ ] import cv2
    import mediapipe as mp
    import numpy as np
    import pandas as pd
    import pickle
    from sklearn.preprocessing import StandardScaler

    # Setup mediapipe instance
    mp_drawing = mp.solutions.drawing_utils
    mp_pose = mp.solutions.pose
```

Taking Video input to create input data and data preprocessing

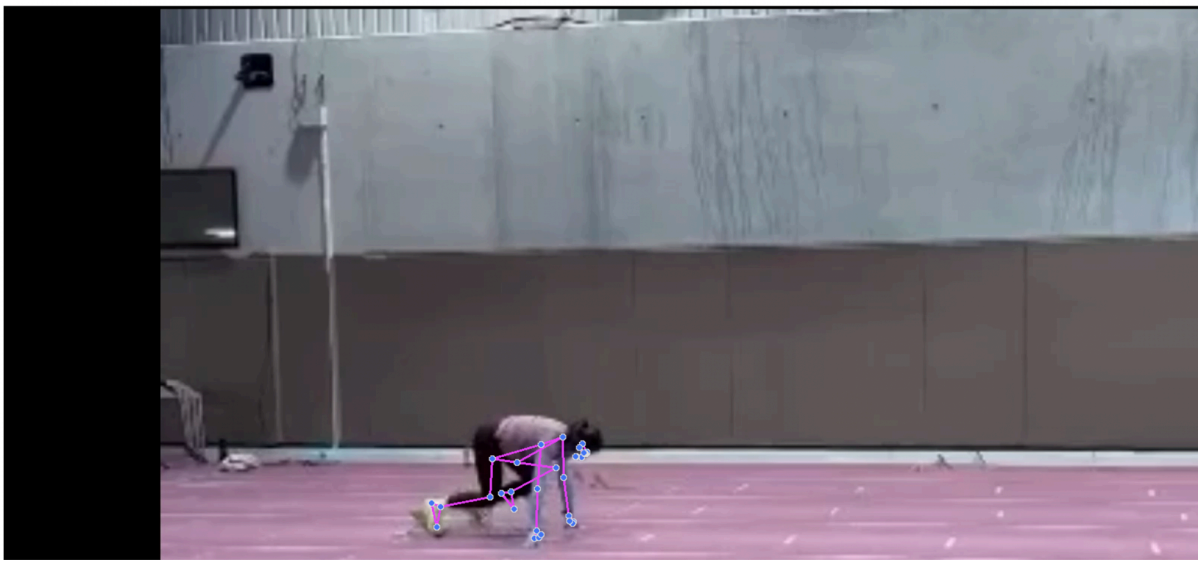
```
[ ] from google.colab.patches import cv2_imshow
```

```
import cv2
import mediapipe as mp
import numpy as np
import pandas as pd

# Function to calculate angle between three points given their coordinates
def calculate_angle(a, b, c):
    a = np.array(a)
    b = np.array(b)
    c = np.array(c)

    radians = np.arctan2(c[1]-b[1], c[0]-b[0]) - np.arctan2(a[1]-b[1], a[0]-b[0])
    angle_degrees = np.abs(np.degrees(radians))

    if angle_degrees > 180:
        angle_degrees = 360 - angle_degrees

    return angle_degrees

cap = cv2.VideoCapture(r"/content/test.mp4")
```

```
# Setup mediapipe instance
mp_pose = mp.solutions.pose
mp_drawing = mp.solutions.drawing_utils

# Create an empty dictionary to store the angles for all frames
angles_dict = {
    'Frame': [],
    'Left Hip Angle': [],
    'Left Knee Angle': [],
    'Left Ankle Angle': [],
    'Right Hip Angle': [],
    'Right Knee Angle': [],
    'Right Ankle Angle': []
}

with mp_pose.Pose(min_detection_confidence=0.5, min_tracking_confidence=0.5) as pose:
    frame_count = 0

    while cap.isOpened():
        ret, frame = cap.read()

        # Check if frame is successfully read
        if not ret:
            break

        # Recolor image to RGB
        image = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
        image.flags.writeable = False

        # Make detection
        results = pose.process(image)

        # Recolor back to BGR
        image.flags.writeable = True
        image = cv2.cvtColor(image, cv2.COLOR_RGB2BGR)

        # Extract landmarks
        try:
            landmarks = results.pose_landmarks.landmark
```

```
            left_hip = [landmarks[mp_pose.PoseLandmark.LEFT_HIP.value].x, landmarks[mp_pose.PoseLandmark.LEFT_HIP.value].y]
            left_knee = [landmarks[mp_pose.PoseLandmark.LEFT_KNEE.value].x, landmarks[mp_pose.PoseLandmark.LEFT_KNEE.value].y]
            left_ankle = [landmarks[mp_pose.PoseLandmark.LEFT_ANKLE.value].x, landmarks[mp_pose.PoseLandmark.LEFT_ANKLE.value].y]
            left_foot = [landmarks[mp_pose.PoseLandmark.LEFT_FOOT_INDEX.value].x, landmarks[mp_pose.PoseLandmark.LEFT_FOOT_INDEX.value].y]
            right_foot = [landmarks[mp_pose.PoseLandmark.RIGHT_FOOT_INDEX.value].x, landmarks[mp_pose.PoseLandmark.RIGHT_FOOT_INDEX.value].y]
            # Get the landmark points for the right side
            right_hip = [landmarks[mp_pose.PoseLandmark.RIGHT_HIP.value].x, landmarks[mp_pose.PoseLandmark.RIGHT_HIP.value].y]
            right_knee = [landmarks[mp_pose.PoseLandmark.RIGHT_KNEE.value].x, landmarks[mp_pose.PoseLandmark.RIGHT_KNEE.value].y]
            right_ankle = [landmarks[mp_pose.PoseLandmark.RIGHT_ANKLE.value].x, landmarks[mp_pose.PoseLandmark.RIGHT_ANKLE.value].y]

            # Calculate the angles
            left_hip_angle = calculate_angle(left_ankle, left_hip, left_knee)
            left_knee_angle = calculate_angle(left_hip, left_knee, left_ankle)
            left_ankle_angle = calculate_angle(left_knee, left_ankle, left_foot)
            right_hip_angle = calculate_angle(right_ankle, right_hip, right_knee)
            right_knee_angle = calculate_angle(right_hip, right_knee, right_ankle)
            right_ankle_angle = calculate_angle(right_knee, right_ankle, right_foot)

            # Append the angles to the dictionary
            angles_dict['Frame'].append(frame_count)
            angles_dict['Left Hip Angle'].append(left_hip_angle)
            angles_dict['Left Knee Angle'].append(left_knee_angle)
            angles_dict['Left Ankle Angle'].append(left_ankle_angle)
            angles_dict['Right Hip Angle'].append(right_hip_angle)
            angles_dict['Right Knee Angle'].append(right_knee_angle)
            angles_dict['Right Ankle Angle'].append(right_ankle_angle)

        except:
            pass

        # Render detections
        image_with_landmarks = cv2.cvtColor(image, cv2.COLOR_RGB2BGR)
        mp_drawing.draw_landmarks(image_with_landmarks, results.pose_landmarks, mp_pose.POSE_CONNECTIONS,
                                  mp_drawing.DrawingSpec(color=(245, 117, 66), thickness=2, circle_radius=2),
                                  mp_drawing.DrawingSpec(color=(245, 66, 230), thickness=2, circle_radius=2)
                                  )
        cv2_imshow(image_with_landmarks)

        if cv2.waitKey(10) & 0xFF == ord('q'):
            break
```

```
[ ]     cap.release()
        cv2.destroyAllWindows()

    # Convert the dictionaries to dataframes
    angles_df = pd.DataFrame(angles_dict)

    # Display the angles dataframe
    print("Angles DataFrame:")
    print(angles_df)
```



```
[ ] import numpy as np

    inp = []
    for i in range(len(angles_df)):
        row = angles_df.iloc[i].values
        # Convert the first element to an integer
        row[0] = int(row[0])
        a = [row]
        inp.append(a)

    # Convert the list of lists to a list of 2D NumPy arrays
    inp = [np.array(a) for a in inp]

    print(inp[0])
```

```
[[ 82.          57.7597934  128.80788866  29.31418939 110.74220177
  114.96826495]]
```

Loading input data into pickle file:

> Model used: Logistic regression

```
[ ] import pickle

    # Open the pickle file for reading
    with open('/content/best_logistic_regression_model.pkl', 'rb') as file:
        model = pickle.load(file)
```

```
[ ] scaler = StandardScaler()
    inp = np.vstack(inp)
```

```
[ ] print(inp)
```

```
[[ 82.          57.7597934  128.80788866  29.31418939 110.74220177
  114.96826495]
 [ 55.          77.17321782 115.16562734  25.83267907 111.753417
  115.94889152]
 [ 48.          79.07775226 119.68912872  25.01476889 111.24179873
  114.40148821]
 ...
 [ 16.         145.58650337 154.94817005  44.26311088 111.30015733
   96.7007075 ]
 [  7.         165.02244598 167.8048513   54.14809309  93.19672364
  112.79764608]
 [ 22.         137.59209141 107.85049327  37.80584998 125.63448926
  146.90383827]]
```

```
[ ] out = []
    for i in range(len(inp)):
        b = model.predict([inp[i]])
        out.append(b)
```

```
[ ] print(out)
```

```
[array([0]), array([0]), array([0]), array([0]), array([0]), array([0]), array([0]), array([0]), array([0]), array([0]), array([0]), array([0]), a
```

```
[ ] out = [int(arr[0]) for arr in out]
```

```
print(out)
```

```
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
```

```
[ ] count_one = out.count(1)
    count_zero = out.count(0)

    if count_one > count_zero:
        print("Person has injury")
    else:
        print("Person won't have an injury")
```

```
[ ] print(out)
```

```
[array([0]), array([0]), array([0]), array([0]), array([0]), array([0]), array([0]), array([0]), array([0]), array([0]), array([0]), array
```

```
[ ] out = [int(arr[0]) for arr in out]
```

```
[ ] print(out)
```

```
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
```

```
count_one = out.count(1)
count_zero = out.count(0)

if count_one > count_zero:
    print("Person has injury")
else:
    print("Person won't have an injury")
```

```
Person won't have an injury
```