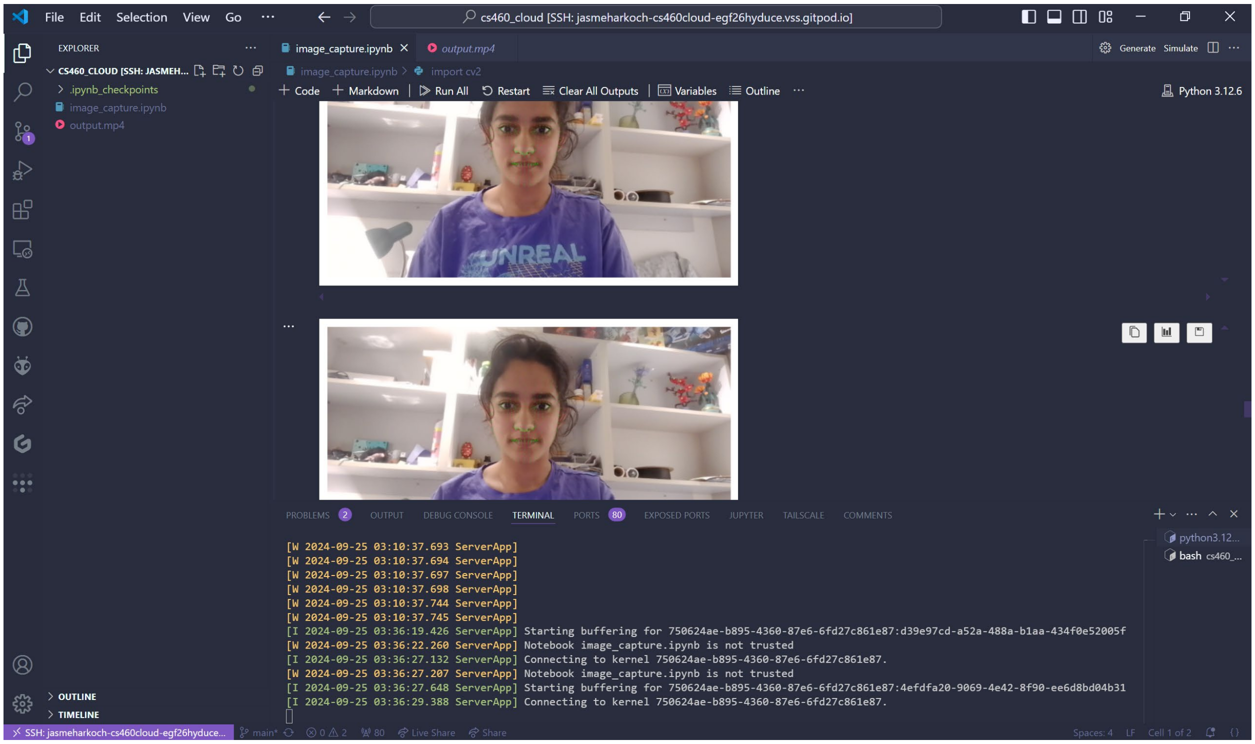


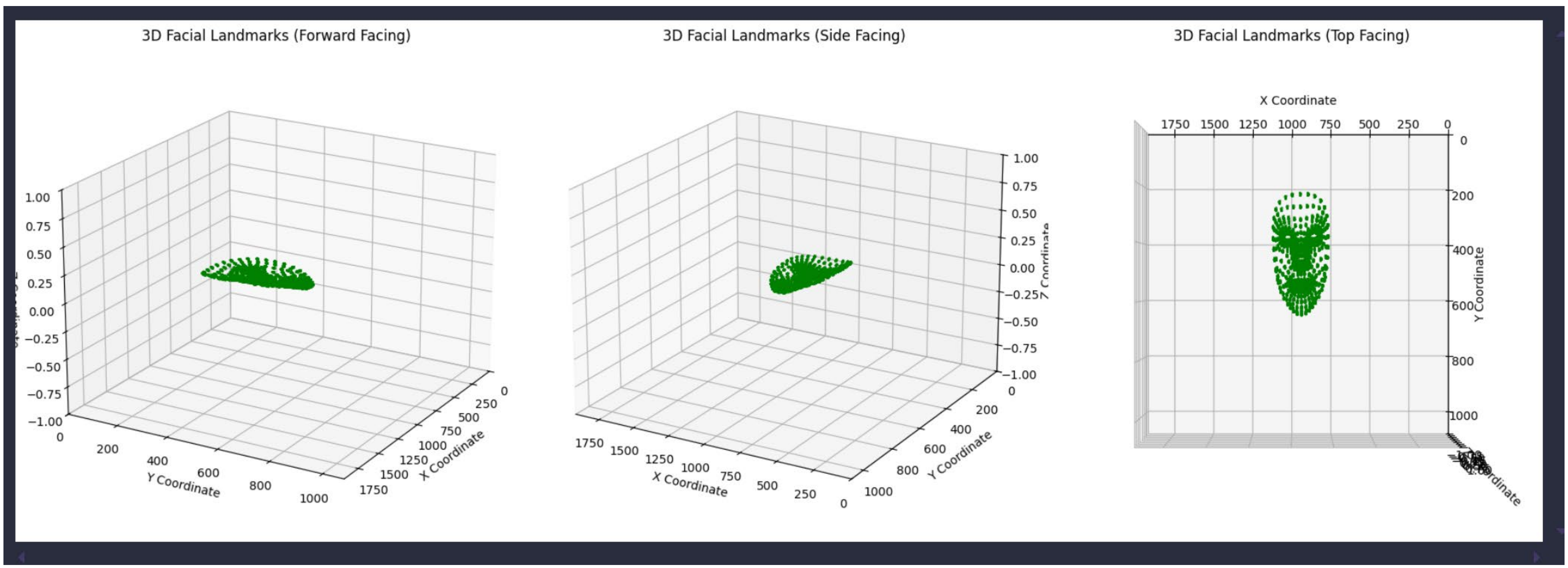
The jupyter notebook held the code for processing the video results to detect facial landmarks using the MediaPipe library to visualize the results and ultimately make a 3d model of my face. The laptop holds the code to take a video and save it to a local file and then through tailscale the video was transferred over to Gitpod for processing.

Here I used cloud setup utilizing Gitpod for development, Tailscale for connecting the local endpoint (laptop camera) and the development kernel (jupyter notebook), and Jupyter Notebooks for data processing and visualization.

Attached is the code I used for it and the results!

```python
import cv2
import mediapipe as mp
import os
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import numpy as np
from sklearn.neighbors import NearestNeighbors

os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'
mp_face_mesh = mp.solutions.face_mesh
face_mesh = mp_face_mesh.FaceMesh(static_image_mode=False, max_num_faces=1)

cap = cv2.VideoCapture('output.mp4')

plt.ion()
frame_count = 0
max_frames = 100
all_points = []

while cap.isOpened() and frame_count < max_frames:
    ret, frame = cap.read()
    if not ret:
        break

    rgb_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
    result = face_mesh.process(rgb_frame)

    if result.multi_face_landmarks:
        for face_landmarks in result.multi_face_landmarks:
            points_3d = []
            for landmark in face_landmarks.landmark:
                x = landmark.x * frame.shape[1]
                y = landmark.y * frame.shape[0]
                z = landmark.z
                points_3d.append([x, y, z])

            all_points.extend(points_3d)
            for landmark in face_landmarks.landmark:
                x = int(landmark.x * frame.shape[1])
                y = int(landmark.y * frame.shape[0])
                cv2.circle(frame, (x, y), 1, (0, 255, 0), -1)

    plt.imshow(cv2.cvtColor(frame, cv2.COLOR_BGR2RGB))
    plt.axis('off')
    plt.show()
    plt.pause(0.001)
    frame_count += 1

cap.release()
plt.ioff()

points = np.array(all_points)
nbrs = NearestNeighbors(n_neighbors=10)
nbrs.fit(points)
distances, indices = nbrs.kneighbors(points)

colors = []
threshold_density = 5
```

```
cap.release()
plt.ioff()

points = np.array(all_points)
nbrs = NearestNeighbors(n_neighbors=10)
nbrs.fit(points)
distances, indices = nbrs.kneighbors(points)

colors = []
threshold_density = 5

for distance in distances:
    if np.sum(distance < 0.05) > threshold_density:
        colors.append('blue')
    else:
        colors.append('green')

fig = plt.figure(figsize=(18, 6))

ax1 = fig.add_subplot(131, projection='3d')
ax1.scatter(points[:, 0], points[:, 1], points[:, 2], c=colors, marker='o', s=1)
ax1.set_xlabel('X Coordinate')
ax1.set_ylabel('Y Coordinate')
ax1.set_zlabel('Z Coordinate')
ax1.set_title('3D Facial Landmarks (Forward Facing)')
ax1.view_init(elev=20, azim=30)
ax1.set_xlim([0, frame.shape[1]])
ax1.set_ylim([0, frame.shape[0]])
ax1.set_zlim([-1, 1])

ax2 = fig.add_subplot(132, projection='3d')
ax2.scatter(points[:, 0], points[:, 1], points[:, 2], c=colors, marker='o', s=1)
ax2.set_xlabel('X Coordinate')
ax2.set_ylabel('Y Coordinate')
ax2.set_zlabel('Z Coordinate')
ax2.set_title('3D Facial Landmarks (Side Facing)')
ax2.view_init(elev=20, azim=120)
ax2.set_xlim([0, frame.shape[1]])
ax2.set_ylim([0, frame.shape[0]])
ax2.set_zlim([-1, 1])

ax3 = fig.add_subplot(133, projection='3d')
ax3.scatter(points[:, 0], points[:, 1], points[:, 2], c=colors, marker='o', s=1)
ax3.set_xlabel('X Coordinate')
ax3.set_ylabel('Y Coordinate')
ax3.set_zlabel('Z Coordinate')
ax3.set_title('3D Facial Landmarks (Top Facing)')
ax3.view_init(elev=90, azim=90)
ax3.set_xlim([0, frame.shape[1]])
ax3.set_ylim([0, frame.shape[0]])
ax3.set_zlim([-1, 1])

plt.tight_layout()
plt.show()
```


3D Facial Landmarks (Forward Facing)
Y Coordinate
X Coordinate
3D Facial Landmarks (Side Facing)
X Coordinate
Y Coordinate
3D Facial Landmarks (Top Facing)
X Coordinate
Y Coordinate