**EX.NO:3 DATE: 30.01.2025**

**IMAGE TRANSFORMATION**

**Aim:**

To implement various geometric transformations on an image, including Translation, Rotation, Scaling, Shearing, and Reflection, using Python.

**Algorithm:**

1. Read the input image using OpenCV or PIL.
2. Apply the following transformations:
   * **Translation**: Shift the image by a specified amount in x and y directions.
   * **Rotation**: Rotate the image by a specified angle.
   * **Scaling**: Resize the image using interpolation.
   * **Shearing**: Apply shearing transformation along x and y axes.
   * **Reflection**: Generate mirrored images by flipping along horizontal, vertical, and diagonal axes.
3. Display the original and transformed images using Matplotlib.
4. Save the transformed images if necessary.

**Code**:

import cv2  
import numpy as np  
import matplotlib.pyplot as plt  
import math  
from PIL import Image

img = cv2.imread('A.png',1)  
img = cv2.cvtColor(img, cv2.COLOR\_BGR2RGB)

# Translation

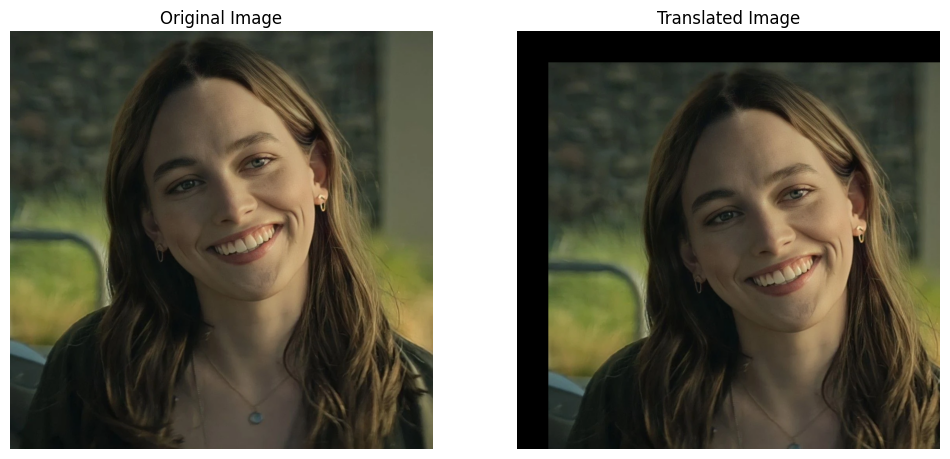
height, width = img.shape[:2]

tx, ty = 80, 80

y\_indices, x\_indices = np.indices((height, width))

ones = np.ones\_like(x\_indices.flatten())  
coords = np.stack([x\_indices.flatten(), y\_indices.flatten(), ones])  
  
translation\_matrix = np.array([  
 [1, 0, tx],  
 [0, 1, ty],  
 [0, 0, 1]  
])  
translated\_coords = translation\_matrix @ coords  
translated\_x, translated\_y = translated\_coords[0].astype(int), translated\_coords[1].astype(int)  
translated\_image = np.zeros((height, width, 3), dtype=np.uint8)

mask = (  
 (translated\_y >= 0) & (translated\_y < height) &  
 (translated\_x >= 0) & (translated\_x < width)  
)  
translated\_image[translated\_y[mask], translated\_x[mask]] = img[y\_indices.flatten()[mask], x\_indices.flatten()[mask]]  
  
output\_image = Image.fromarray(translated\_image)  
plt.figure(figsize=(12, 10))  
plt.subplot(1, 2, 1)  
plt.imshow(img)  
plt.title('Original Image')  
plt.axis('Off')  
plt.subplot(1, 2, 2)  
plt.imshow(output\_image)  
plt.title('Translated Image')  
plt.axis('Off')  
plt.show()





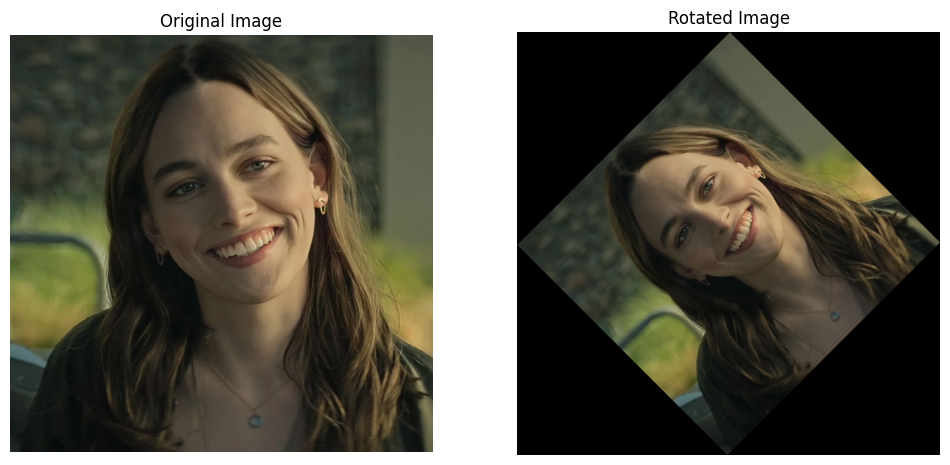
# Rotation

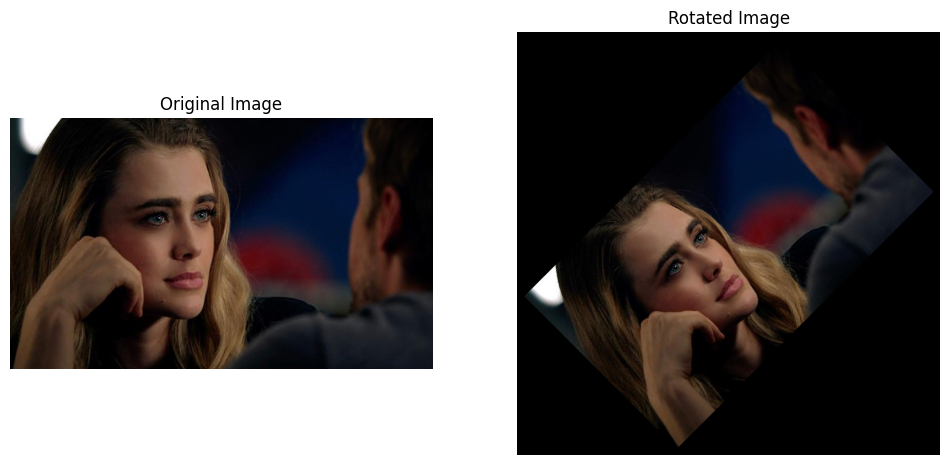
height, width = img.shape[:2]

rotation\_amount\_degree = 45  
rotation\_amount\_rad = np.deg2rad(rotation\_amount\_degree)

new\_height = int(np.ceil(np.sqrt(height\*\*2 + width\*\*2)))  
new\_width = new\_height  
  
y\_indices, x\_indices = np.indices((new\_height, new\_width))

new\_center = (new\_width // 2, new\_height // 2)  
  
x\_indices\_centered = x\_indices - new\_center[0]  
y\_indices\_centered = y\_indices - new\_center[1]  
  
ones = np.ones\_like(x\_indices\_centered)  
coords = np.stack([x\_indices\_centered.flatten(), y\_indices\_centered.flatten(), ones.flatten()])  
  
cos\_a = np.cos(rotation\_amount\_rad)  
sin\_a = np.sin(rotation\_amount\_rad)  
rotation\_matrix = np.array([  
 [cos\_a, -sin\_a, 0],  
 [sin\_a, cos\_a, 0],  
 [0, 0, 1]  
])  
  
rotated\_coords = rotation\_matrix @ coords  
rotated\_x, rotated\_y = rotated\_coords[0] + width // 2, rotated\_coords[1] + height // 2  
  
rotated\_image = np.zeros((new\_height, new\_width, 3), dtype=np.uint8)  
  
mask = (  
 (rotated\_y >= 0) & (rotated\_y < height) &  
 (rotated\_x >= 0) & (rotated\_x < width)  
)  
rotated\_image[y\_indices.flatten()[mask], x\_indices.flatten()[mask]] = img[rotated\_y[mask].astype(int), rotated\_x[mask].astype(int)]  
  
output\_image = Image.fromarray(rotated\_image)  
plt.figure(figsize=(12, 10))  
plt.subplot(1, 2, 1)  
plt.imshow(img)  
plt.title('Original Image')  
plt.axis('Off')  
plt.subplot(1, 2, 2)  
plt.imshow(output\_image)  
plt.title('Rotated Image')  
plt.axis('Off')  
plt.show()





# Scaling

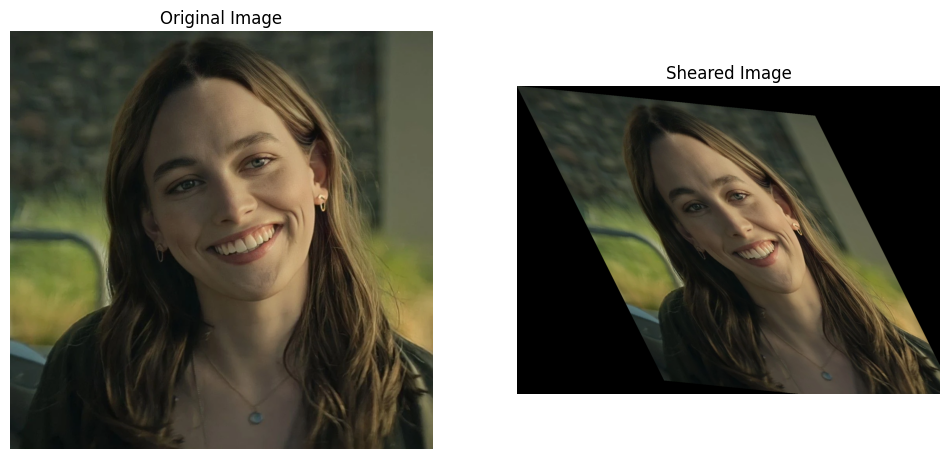
image = cv2.imread("A.png")  
rows, cols, channels = image.shape  
  
scale\_x, scale\_y = 2.0, 2.0  
  
new\_width = int(cols \* scale\_x)  
new\_height = int(rows \* scale\_y)  
  
x\_new = np.linspace(0, cols - 1, new\_width)  
y\_new = np.linspace(0, rows - 1, new\_height)  
  
x\_old, y\_old = np.meshgrid(np.floor(x\_new / scale\_x).astype(int),  
 np.floor(y\_new / scale\_y).astype(int))  
  
x\_old = np.clip(x\_old, 0, cols - 1)  
y\_old = np.clip(y\_old, 0, rows - 1)  
  
scaled\_image = image[y\_old, x\_old]  
  
plt.figure(figsize=(12, 10))  
plt.subplot(1, 2, 1)  
plt.imshow(img)  
plt.title('Original Image')  
plt.axis('Off')  
plt.subplot(1, 2, 2)  
plt.imshow(cv2.cvtColor(scaled\_image, cv2.COLOR\_BGR2RGB))  
plt.title("Scaled Image (2x)")  
plt.axis('Off')  
plt.show()





# Shearing

def shear\_image(image\_path, shear\_x=0, shear\_y=0):  
 image = Image.open(image\_path)  
 original\_width, original\_height = image.size  
 img\_array = np.array(image)  
   
 new\_width = original\_width + int(shear\_x \* original\_height)  
 new\_height = original\_height + int(shear\_y \* original\_width)  
 sheared\_image\_array = np.zeros((new\_height, new\_width, img\_array.shape[2]), dtype=np.uint8)  
   
 for i in range(new\_height):  
 for j in range(new\_width):  
 orig\_x = int(j - shear\_x \* i)  
 orig\_y = int(i - shear\_y \* j)  
   
 if 0 <= orig\_x < original\_width and 0 <= orig\_y < original\_height:  
 sheared\_image\_array[i, j] = img\_array[orig\_y, orig\_x]  
   
 sheared\_image = Image.fromarray(sheared\_image\_array)  
 return sheared\_image  
  
sheared\_image = shear\_image("A.png", shear\_x=0.5, shear\_y=0.1)   
  
plt.figure(figsize=(12, 10))  
plt.subplot(1, 2, 1)  
plt.imshow(img)  
plt.title('Original Image')  
plt.axis('Off')  
plt.subplot(1, 2, 2)  
plt.imshow(sheared\_image)  
plt.title('Sheared Image')  
plt.axis('Off')  
plt.show()

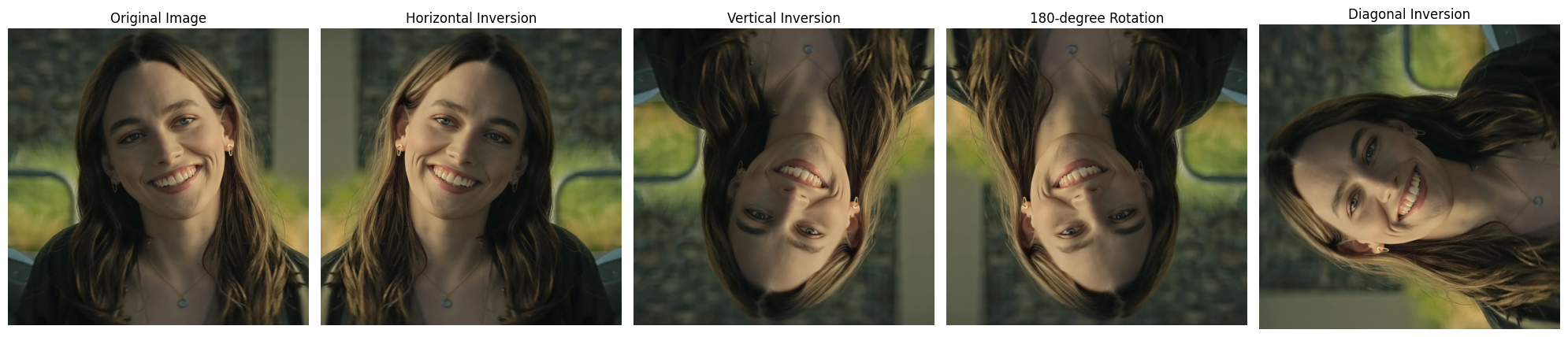




# Reflection

def apply\_inversions(image\_path):  
 image = Image.open(image\_path)  
 img\_array = np.array(image)  
   
 horizontal\_inversion = np.flip(img\_array, axis=1)  
  
 vertical\_inversion = np.flip(img\_array, axis=0)  
  
 rotated\_180 = np.flip(img\_array, axis=(0, 1))  
  
 diagonal\_inversion = np.transpose(img\_array, axes=(1, 0, 2))

return img\_array, horizontal\_inversion, vertical\_inversion, rotated\_180, diagonal\_inversion  
  
image\_path = "A.png"  
original, horizontal, vertical, rotated\_180, diagonal = apply\_inversions(image\_path)  
  
fig, axes = plt.subplots(1, 5, figsize=(20, 10))  
  
axes[0].imshow(original)  
axes[0].set\_title("Original Image")  
axes[0].axis('off')   
axes[1].imshow(horizontal)  
axes[1].set\_title("Horizontal Inversion")  
axes[1].axis('off')  
  
axes[2].imshow(vertical)  
axes[2].set\_title("Vertical Inversion")  
axes[2].axis('off')  
  
axes[3].imshow(rotated\_180)  
axes[3].set\_title("180-degree Rotation")  
axes[3].axis('off')  
  
axes[4].imshow(diagonal)  
axes[4].set\_title("Diagonal Inversion")  
axes[4].axis('off')  
  
plt.tight\_layout()  
plt.show()





**Inference**

1. The geometric transformations were successfully applied to the input image.
2. The transformed images demonstrate the correct implementation of translation, rotation, scaling, shearing, and reflection.
3. The output visualizations confirm the expected results of each transformation.
4. This implementation can be extended for more complex transformations in image processing tasks.

**Result:**

The transformations were successfully applied, and the output images accurately reflect translation, rotation, scaling, shearing, and reflection.