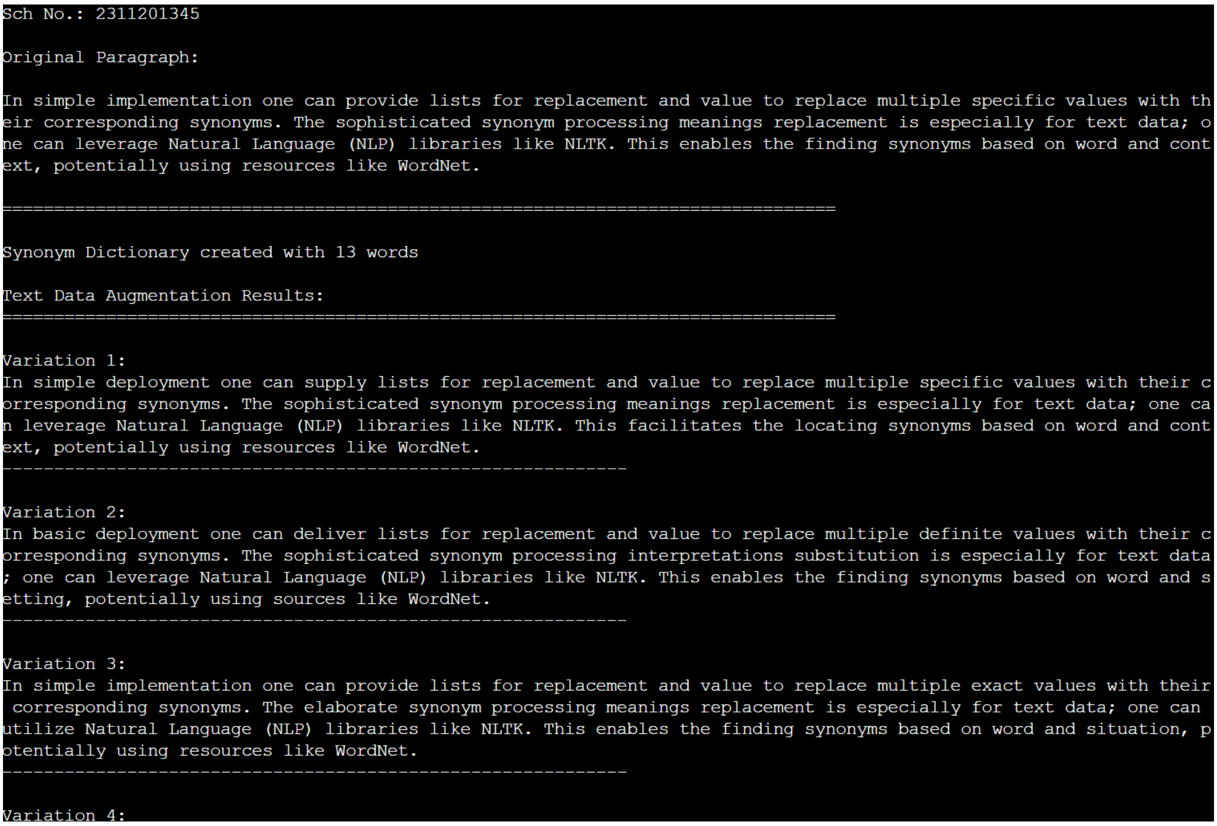
ASSIGNMENT: 3

SUBJECT - DATA SCEINCE
CODE – CS 358
NAME - ANURAG SINGH DHAKAD
SCH NO. - 2311201345

ANSWER 1:

```
import random
import re

original_paragraph = (
    "In simple implementation one can provide lists for replacement and value "
    "to replace multiple specific values with their corresponding synonyms. "
    "The sophisticated synonym processing meanings replacement is especially "
    "for text data; one can leverage Natural Language (NLP) libraries like NLTK. "
    "This enables the finding synonyms based on word and context, potentially "
    "using resources like WordNet."
)

synonym_dictionary = {
    'simple': ['basic', 'elementary', 'straightforward', 'easy', 'uncomplicated'],
    'implementation': ['execution', 'application', 'realization', 'development',
'deployment'],
    'provide': ['supply', 'offer', 'furnish', 'deliver', 'give'],
    'replacement': ['substitution', 'exchange', 'swap', 'change', 'alternative'],
    'specific': ['particular', 'exact', 'precise', 'definite', 'distinct'],
    'sophisticated': ['advanced', 'complex', 'refined', 'elaborate', 'intricate'],
    'leverage': ['utilize', 'exploit', 'harness', 'employ', 'use'],
    'libraries': ['packages', 'modules', 'frameworks', 'toolkits', 'collections'],
    'enables': ['allows', 'permits', 'facilitates', 'makes possible', 'empowers'],
    'finding': ['discovering', 'locating', 'identifying', 'detecting', 'searching'],
    'meanings': ['definitions', 'interpretations', 'senses', 'significance', 'semantics'],
    'context': ['situation', 'environment', 'setting', 'circumstances', 'background'],
    'resources': ['tools', 'materials', 'assets', 'sources', 'references']
}

def preserve_word_case(original, replacement):
    if original.isupper():
        return replacement.upper()
    elif original.islower():
        return replacement.lower()
    elif original.istitle():
        return replacement.capitalize()
    else:
        return replacement

def replace_word_with_synonym(word, synonym_dict):
    word_lower = word.lower()
    if word_lower in synonym_dict:
        synonym = random.choice(synonym_dict[word_lower])
        return preserve_word_case(word, synonym)
    return word

def synonym_replacement_augmentation(text, synonym_dict,
replacement_prob=0.3, num_variations=5):
    def augment_single_text(text, synonym_dict, replacement_prob):
        tokens = re.findall(r'\b\w+\b|\W+', text)
        augmented_tokens = []
        for token in tokens:
            if re.match(r'^\b\w+\b$', token):
                if token.lower() in synonym_dict and random.random() <
replacement_prob:
                    augmented_tokens.append(replace_word_with_synonym(token,
synonym_dict))
                else:
                    augmented_tokens.append(token)
            else:
                augmented_tokens.append(token)
        return ''.join(augmented_tokens)
    return [augment_single_text(text, synonym_dict, replacement_prob) for _ in
range(num_variations)]

def advanced_synonym_replacement(text, synonym_dict, num_replacements=None,
preserve_case=True):
    words = re.findall(r'\b\w+\b', text)
    replaceable_words = [w for w in words if w.lower() in synonym_dict]
    if not replaceable_words:
        return text, {}
    max_replace = min(len(replaceable_words), 5)
    if num_replacements is None:
        num_replacements = random.randint(1, max_replace)
    else:
        num_replacements = min(num_replacements, max_replace)
    words_to_replace = random.sample(replaceable_words, num_replacements)
    augmented_text = text
    replacements_made = {}
    for word in words_to_replace:
        synonym = random.choice(synonym_dict[word.lower()])
        if preserve_case:
            synonym = preserve_word_case(word, synonym)
        pattern = r'\b' + re.escape(word) + r'\b'
        augmented_text, count = re.subn(pattern, synonym, augmented_text,
count=1)
        if count > 0:
            replacements_made[word] = synonym
    return augmented_text, replacements_made

if __name__ == "__main__":
    print("Sch No.: 2311201345")
    print("\nOriginal Paragraph:\n")
    print(original_paragraph)
    print("\n" + "=" * 80 + "\n")
    print("Synonym Dictionary created with", len(synonym_dictionary), "words")
    print("\nText Data Augmentation Results:")
    print("=" * 80)
    augmented_texts = synonym_replacement_augmentation(
        original_paragraph, synonym_dictionary, replacement_prob=0.4,
num_variations=5
    )
    for i, augmented_text in enumerate(augmented_texts, 1):
        print(f"\nVariation {i}:")
        print(augmented_text)
        print("-" * 60)
    print("\nAdvanced Synonym Replacement with Change Tracking:")
    print("=" * 80)
    for i in range(3):
        augmented, changes = advanced_synonym_replacement(
            original_paragraph, synonym_dictionary, num_replacements=4,
preserve_case=True
        )
        print(f"\nAdvanced Variation {i + 1}:")
        print("Changes made:")
        for original, replacement in changes.items():
            print(f" '{original}' → '{replacement}'")
        print("\nAugmented text:")
        print(augmented)
        print("-" * 60)
```

OUTPUT :-



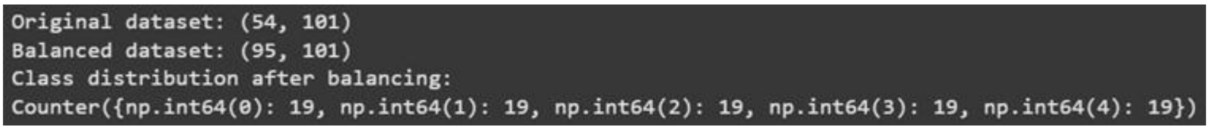
```
Sch No.: 2311201345

Original Paragraph:

In simple implementation one can provide lists for replacement and value to replace multiple specific values with th
eir corresponding synonyms. The sophisticated synonym processing meanings replacement is especially for text data; o
ne can leverage Natural Language (NLP) libraries like NLTK. This enables the finding synonyms based on word and cont
ext, potentially using resources like WordNet.

================================================================================

Synonym Dictionary created with 13 words

Text Data Augmentation Results:
================================================================================

Variation 1:
In simple deployment one can supply lists for replacement and value to replace multiple specific values with their c
orresponding synonyms. The sophisticated synonym processing meanings replacement is especially for text data; one ca
n leverage Natural Language (NLP) libraries like NLTK. This facilitates the locating synonyms based on word and cont
ext, potentially using resources like WordNet.
------------------------------------------------------------

Variation 2:
In basic deployment one can deliver lists for replacement and value to replace multiple definite values with their c
orresponding synonyms. The sophisticated synonym processing interpretations substitution is especially for text data
; one can leverage Natural Language (NLP) libraries like NLTK. This enables the finding synonyms based on word and s
etting, potentially using sources like WordNet.
------------------------------------------------------------

Variation 3:
In simple implementation one can provide lists for replacement and value to replace multiple exact values with their
 corresponding synonyms. The elaborate synonym processing meanings replacement is especially for text data; one can
utilize Natural Language (NLP) libraries like NLTK. This enables the finding synonyms based on word and situation, p
otentially using resources like WordNet.
------------------------------------------------------------

Variation 4:
```

ANSWER 2:

CODE:

```
print("Scholar No.: 2311201345")  # <-- now prints in output


from imblearn.over_sampling import BorderlineSMOTE
from sklearn.neighbors import NearestNeighbors
from collections import Counter
import pandas as pd
import numpy as np


# Load and prepare data
df   =   pd.read_csv('ECG_data.csv')
X = df.iloc[:, :-1].values    # Features (101 columns)
y = df.iloc[:, -1].values.astype(int)  # Labels (5 classes)


# Apply Borderline-SMOTE
borderline_smote = BorderlineSMOTE(
   random_state=42,
   k_neighbors=3,     # Reduced due to small dataset
   m_neighbors=10,    # Neighbors to identify borderline instances
   kind='borderline-1' # Borderline-1 variant
)

# Transform the data
X_resampled, y_resampled = borderline_smote.fit_resample(X, y)

# Save balanced dataset
balanced_df = pd.DataFrame(X_resampled)
balanced_df['class'] = y_resampled
balanced_df.to_csv('ECG_data_balanced.csv',  index=False)

# Print details
print(f"Original dataset: {X.shape}")
print(f"Balanced dataset: {X_resampled.shape}")
print("Class distribution after balancing:")
print(Counter(y_resampled))
```

OUTPUT:



```
Original dataset: (54, 101)
Balanced dataset: (95, 101)
Class distribution after balancing:
Counter({np.int64(0): 19, np.int64(1): 19, np.int64(2): 19, np.int64(3): 19, np.int64(4): 19})
```

PSEUDOCODE:

```
INPUT:
   Dataset (X, y)      # X = features, y = class labels
   k_neighbors        # number of nearest neighbors to consider
   m_neighbors         # number of neighbors to decide if borderline
```

```
  N_synthetic        # number of synthetic samples to generate

OUTPUT:
  Balanced dataset (X_new, y_new)

--------------------------------------------------------------

Step 1: Identify minority classes
  class_counts = count_samples_per_class(y)
  minority_classes = classes with fewer samples than majority

Step 2: Find borderline samples
  borderline_list = []
  For each sample xi in minority_classes:
    neighbors = find_m_neighbors(xi, X)  # nearest m neighbors
    different_class_count = count of neighbors with different label
    same_class_count = m - different_class_count

    IF (different_class_count == 0):
      # Safe sample (all neighbors same class)
      continue
    ELSE IF (different_class_count == m):
      # Noisy sample (all neighbors different class)
      continue
    ELSE:
      # Risky sample (mixed neighbors)
      borderline_list.add(xi)
```

```
Step 3: Generate synthetic samples
   synthetic_samples = []
   While len(synthetic_samples) < N_synthetic:
      Select xi from borderline_list
      neighbors_same_class = find_k_neighbors(xi, X with same label)
      Choose neighbor xj randomly from neighbors_same_class
      random_factor = random(0,1)
      new_sample = xi + random_factor * (xj - xi)
      synthetic_samples.add(new_sample with label of xi)

Step 4: Build final dataset
   X_new = concatenate(X, synthetic_samples)
   y_new = concatenate(y, labels_of_synthetic_samples)

RETURN (X_new, y_new)
```

ANSWER 3:

CODE:

```
#!/usr/bin/env python3
"""
Data Augmentation Script for Image Processing
===========================================
This script applies two data augmentation operations on an input image:
1. Random rotation (10-80 degrees, left or right)
2. Random shift (5% of width and height)
```

```
It generates 9 augmented images and displays them in a 3x3 grid.

Requirements:
- PIL (Pillow)
- numpy
- matplotlib

Author: Computer Science Student
Date: September 25, 2025
"""

import numpy as np
import matplotlib.pyplot as plt
from PIL import Image
import random
import os
import json
from datetime import datetime


class ImageAugmenter:
    """Class to handle image augmentation operations"""

    def __init__(self, seed=42):
        """
        Initialize the augmenter with a random seed for reproducibility.
        Args:
            seed (int): Random seed for reproducible results
        """
        self.seed = seed
        random.seed(seed)
        self.augmentation_log = []

    def load_image(self, image_path):
        """
        Load image from file path.
        Args:
            image_path (str): Path to the input image
        Returns:
            numpy.ndarray: Image as numpy array
        """
        try:
            img = Image.open(image_path)
            return np.array(img)
        except Exception as e:
            print(f"Error loading image: {e}")
            return None

    def rotate_image(self, image_array, angle):
        """
        Rotate image by specified angle.
        Args:
            image_array (numpy.ndarray): Input image
            angle (float): Rotation angle in degrees (positive = clockwise)
        Returns:
            numpy.ndarray: Rotated image
        """
        img_pil = Image.fromarray(image_array)
        rotated = img_pil.rotate(angle, expand=True, fillcolor=(255, 255, 255))
        return np.array(rotated)

    def shift_image(self, image_array, shift_x_ratio, shift_y_ratio):
        """
        Shift image by specified ratios.
        Args:
            image_array (numpy.ndarray): Input image
            shift_x_ratio (float): Horizontal shift ratio (-1.0 to 1.0)
            shift_y_ratio (float): Vertical shift ratio (-1.0 to 1.0)
        Returns:
            numpy.ndarray: Shifted image
        """
        height, width = image_array.shape[:2]
        shift_x = int(width * shift_x_ratio)
        shift_y = int(height * shift_y_ratio)

        # Create a new image with white background
        new_image = np.full_like(image_array, 255)

        # Calculate source and destination coordinates
        src_y_start = max(0, -shift_y)
        src_y_end = min(height, height - shift_y)
        src_x_start = max(0, -shift_x)
        src_x_end = min(width, width - shift_x)
        dst_y_start = max(0, shift_y)
        dst_y_end = dst_y_start + (src_y_end - src_y_start)
        dst_x_start = max(0, shift_x)
        dst_x_end = dst_x_start + (src_x_end - src_x_start)

        # Copy the shifted region
        if (dst_y_end > dst_y_start) and (dst_x_end > dst_x_start):
            new_image[dst_y_start:dst_y_end, dst_x_start:dst_x_end] = \
                image_array[src_y_start:src_y_end, src_x_start:src_x_end]

        return new_image

    def generate_augmented_images(self, image_array, num_images=9):
        """
        Generate multiple augmented images.
        Args:
            image_array (numpy.ndarray): Input image
            num_images (int): Number of augmented images to generate
        Returns:
            list: List of augmented images
        """
        augmented_images = []
        self.augmentation_log = []

        print(f"Generating {num_images} augmented images...")
        print("-" * 50)
```

```
for i in range(num_images):
    # Random rotation
    rotation_angle = random.uniform(10, 80)
    if random.choice([True, False]):
        rotation_angle = -rotation_angle # Left rotation

    rotated_img = self.rotate_image(image_array, rotation_angle)

    # Random shift
    shift_x = random.uniform(-0.05, 0.05)
    shift_y = random.uniform(-0.05, 0.05)
    final_img = self.shift_image(rotated_img, shift_x, shift_y)

    # Store image and log
    augmented_images.append(final_img)
    self.augmentation_log.append({
        "image_number": i + 1,
        "rotation_degrees": round(rotation_angle, 1),
        "shift_x_ratio": round(shift_x, 3),
        "shift_y_ratio": round(shift_y, 3)
    })

    print(f"Image {i+1:2d}: Rotation = {rotation_angle:6.1f}°, "
          f"Shift = ({shift_x:6.3f}, {shift_y:6.3f})")

print(f"\nSuccessfully generated {len(augmented_images)} augmented images")
return augmented_images
```

```
def display_images(self, original_image, augmented_images, save_plot=False):
    """
    Display original and augmented images in a grid.
    Args:
        original_image (numpy.ndarray): Original image
        augmented_images (list): List of augmented images
        save_plot (bool): Whether to save the plot as PNG
    """
    # Original image
    plt.figure(figsize=(8, 6))
    plt.imshow(original_image)
    plt.title('Original Image', fontsize=14, fontweight='bold')
    plt.axis('off')
    if save_plot:
        plt.savefig('original_image.png', dpi=300, bbox_inches='tight')
    plt.show()

    # 3x3 grid for augmented images
    fig, axes = plt.subplots(3, 3, figsize=(15, 15))
    fig.suptitle('Data Augmented Images (3x3 Grid)', fontsize=16, fontweight='bold')

    for i, ax in enumerate(axes.flat):
        if i < len(augmented_images):
            ax.imshow(augmented_images[i])
            ax.set_title(f'Augmented Image {i+1}', fontsize=12)
            ax.axis('off')

    plt.tight_layout()
```

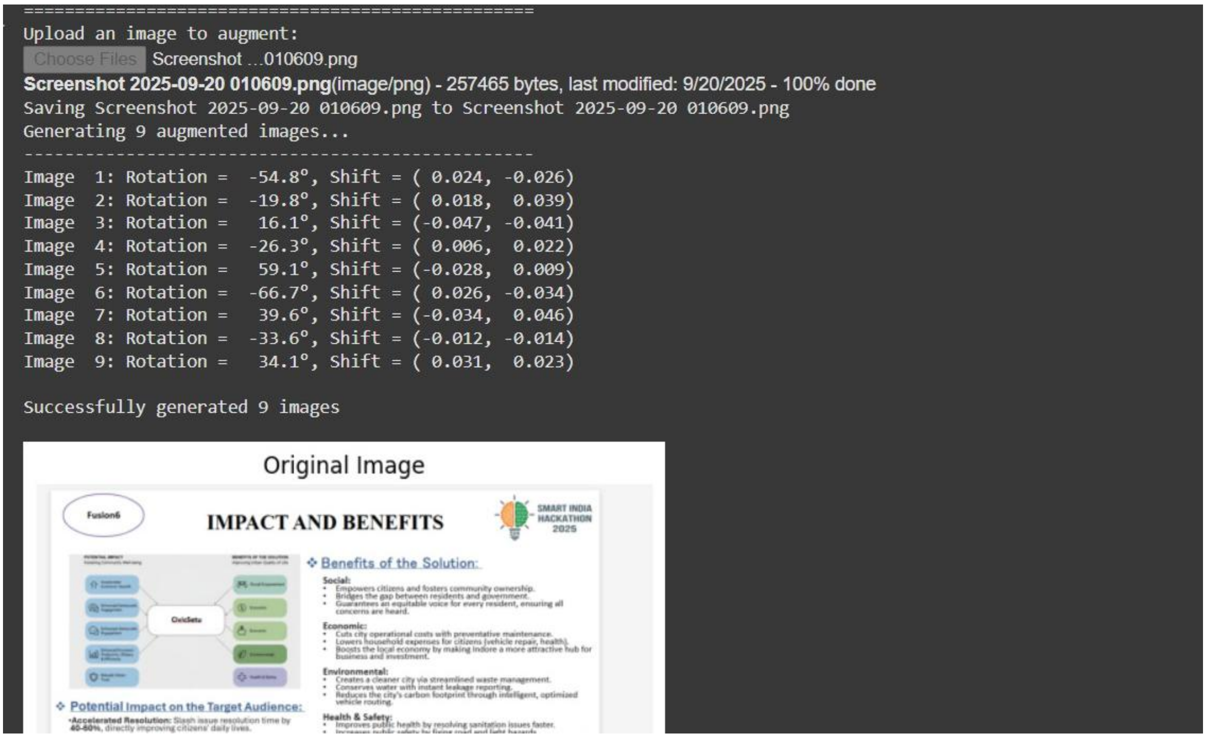
```
        if save_plot:
            plt.savefig('augmented_images_grid.png', dpi=300, bbox_inches='tight')
        plt.show()

    def save_augmentation_log(self, filename="augmentation_log.json"):
        """
        Save augmentation parameters to JSON file.
        Args:
            filename (str): Output filename for the log
        """
        log_data = {
            "timestamp": datetime.now().isoformat(),
            "seed": self.seed,
            "total_images": len(self.augmentation_log),
            "operations": [
                "Random rotation (10-80 degrees, left or right)",
                "Random shift (5% of width and height)"
            ],
            "parameters": self.augmentation_log
        }
        with open(filename, 'w') as f:
            json.dump(log_data, f, indent=2)

        print(f"\nAugmentation log saved to: {filename}")


def main():
    """Main function to run the data augmentation pipeline"""
```

```
# Scholar No
print("Scholar No.: 2311201345")

# Configuration
INPUT_IMAGE = "cat.png"  # Change this to your image path
NUM_AUGMENTED_IMAGES = 9
RANDOM_SEED = 42
SAVE_PLOTS = True

print("=" * 60)
print("  IMAGE DATA AUGMENTATION PIPELINE")
print("=" * 60)
print(f"Input Image: {INPUT_IMAGE}")
print(f"Number of Augmented Images: {NUM_AUGMENTED_IMAGES}")
print(f"Random Seed: {RANDOM_SEED}")
print("=" * 60)

# Initialize augmenter
augmenter = ImageAugmenter(seed=RANDOM_SEED)

# Load original image
print("\nLoading image...")
original_image = augmenter.load_image(INPUT_IMAGE)
if original_image is None:
    print(f"Failed to load image: {INPUT_IMAGE}")
    return

print(f"Image loaded successfully!")
```

```
    print(f" - Dimensions: {original_image.shape}")
    print(f" - Data type: {original_image.dtype}")

    # Generate augmented images
    print("\nStarting augmentation process...")
    augmented_images = augmenter.generate_augmented_images(
        original_image, NUM_AUGMENTED_IMAGES
    )

    # Display results
    print("\nDisplaying results...")
    augmenter.display_images(original_image, augmented_images, SAVE_PLOTS)

    # Save augmentation log
    augmenter.save_augmentation_log()

    print("\nData augmentation pipeline completed successfully!")
    print("\nFiles created:")
    if SAVE_PLOTS:
        print("  - original_image.png")
        print("  - augmented_images_grid.png")
        print("  - augmentation_log.json")


# Additional utility function for batch processing
def batch_augment_images(image_folder, output_folder, num_augmentations=9):
    """
    Apply augmentation to multiple images in a folder.
```

```
Args:
    image_folder (str): Path to folder containing input images
    output_folder (str): Path to folder for saving augmented images
    num_augmentations (int): Number of augmentations per image
"""
if not os.path.exists(output_folder):
    os.makedirs(output_folder)

augmenter = ImageAugmenter()
image_extensions = {'.png', '.jpeg', '.jpg', '.bmp', '.tiff'}
image_files = [f for f in os.listdir(image_folder)
               if os.path.splitext(f.lower())[1] in image_extensions]

print(f"Found {len(image_files)} images to process")

for img_file in image_files:
    print(f"\nProcessing: {img_file}")
    img_path = os.path.join(image_folder, img_file)
    original_image = augmenter.load_image(img_path)
    if original_image is None:
        continue

    augmented_images = augmenter.generate_augmented_images(
        original_image, num_augmentations
    )

    # Save augmented images
    base_name = os.path.splitext(img_file)[0]
    for i, aug_img in enumerate(augmented_images):
        output_filename = f"{base_name}_aug_{i+1:02d}.png"
        output_path = os.path.join(output_folder, output_filename)
        aug_img_pil = Image.fromarray(aug_img.astype('uint8'))
        aug_img_pil.save(output_path)
    print(f"  Saved {len(augmented_images)} augmented images")



if _name__== "_main_":
    main()
```

OUTPUT:



```
==================================================
Upload an image to augment:
Choose Files Screenshot ...010609.png
Screenshot 2025-09-20 010609.png(image/png) - 257465 bytes, last modified: 9/20/2025 - 100% done
Saving Screenshot 2025-09-20 010609.png to Screenshot 2025-09-20 010609.png
Generating 9 augmented images...
--------------------------------------------------
Image  1: Rotation =  -54.8°, Shift = ( 0.024, -0.026)
Image  2: Rotation =  -19.8°, Shift = ( 0.018,  0.039)
Image  3: Rotation =   16.1°, Shift = (-0.047, -0.041)
Image  4: Rotation =  -26.3°, Shift = ( 0.006,  0.022)
Image  5: Rotation =   59.1°, Shift = (-0.028,  0.009)
Image  6: Rotation =  -66.7°, Shift = ( 0.026, -0.034)
Image  7: Rotation =   39.6°, Shift = (-0.034,  0.046)
Image  8: Rotation =  -33.6°, Shift = (-0.012, -0.014)
Image  9: Rotation =   34.1°, Shift = ( 0.031,  0.023)

Successfully generated 9 images
```

Original Image

Fusion6

IMPACT AND BENEFITS

SMART INDIA HACKATHON 2025

❖ Benefits of the Solution:

Social:
- Empowers citizens and fosters community ownership.
- Bridges the gap between residents and government.
- Guarantees an equitable voice for every resident, ensuring all concerns are heard.

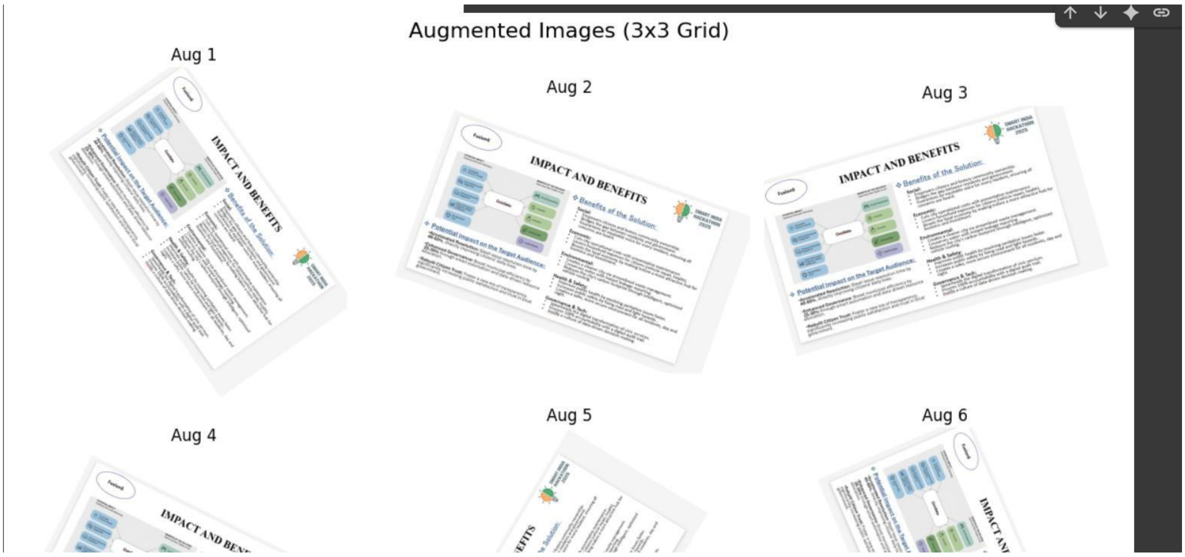
Economic:
- Cuts city operational costs with preventative maintenance.
- Lowers household expenses for citizens (vehicle repair, health).
- Boosts the local economy by making Indore a more attractive hub for business and investment.

Environmental:
- Creates a cleaner city via streamlined waste management.
- Conserves water with instant leakage reporting.
- Reduces the city's carbon footprint through intelligent, optimized vehicle routing.

Health & Safety:
- Improves public health by resolving sanitation issues faster.
- Increases public safety by fixing road and light hazards.

❖ Potential Impact on the Target Audience:

•**Accelerated Resolution:** Slash issue resolution time by **40-60%**, directly improving citizens' daily lives.



Augmented Images (3x3 Grid)
Aug 1
Aug 2
Aug 3
Aug 4
Aug 5
Aug 6
IMPACT AND BENEFITS
Benefits of the Solution:
Potential Impact on the Target Audience: