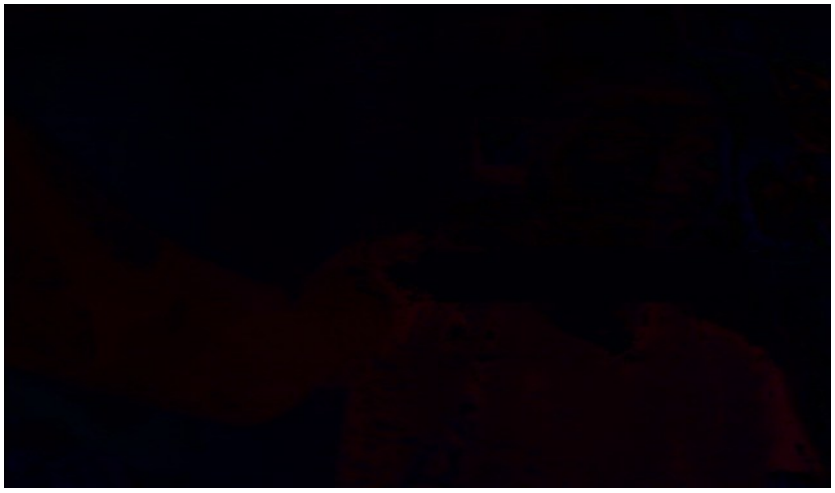
Code 1:

```
import cv2
# For USB webcam (index 0 = first camera)
 cap = cv2.VideoCapture(0)
 while True:
    ret, frame = cap.read()
    if not ret:
        break
    cv2.imshow("Camera Stream", frame)
    if cv2.waitKey(1) & 0xFF == ord("q"):
        break
cap.release()
cv2.destroyAllWindows()
```

Input:

Output:



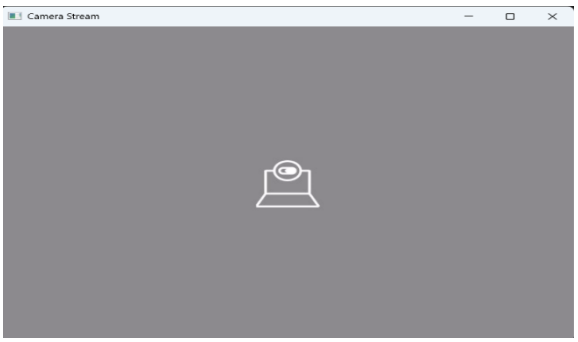
Logic:

1. This program accesses and shows live video from the computer's webcam using OpenCV.
2. The camera is started with the cv2.VideoCapture(0) command, where "0" denotes the webcam device that is used by default.
3. Every frame is recorded inside the loop using cap.read() and shown in a window called "Camera Stream."
4. Until the user hits the "q" key to end the loop, the program keeps displaying the live feed.
5. After exiting, cv2.destroyAllWindows() is used to close all display windows and cap.release() is used to release the camera.

Code 2:

```
import cv2
import os
cap = cv2.VideoCapture(0)
# Create output folder
os.makedirs("frames", exist_ok=True)
frame_count = 0
while True:
    ret, frame = cap.read()
    if not ret:
        break
    # Show stream
    cv2.imshow("Camera Stream", frame)
    # Save frame
    filename = f"frames/frame_{frame_count:06d}.jpg"
    cv2.imwrite(filename, frame)
    frame_count += 1
    if cv2.waitKey(1) & 0xFF == ord("q"):
        break
cap.release()
cv2.destroyAllWindows()
```

Input:



Output:

Logic:

1. This program captures live video from the webcam using OpenCV.
2. It shows the video on the screen and saves each frame as a separate image in a folder called **"frames"**.
3. The program keeps running until the user presses the **'q' key**, after which the camera stops and the windows close.
4. Each frame is automatically numbered so they can be used later for analysis or creating a video.

Code 3:

```
import cv2
# Read an image
img = cv2.imread(r"C:\Users\leela\OneDrive\Desktop\Cv\img1.jpg")   # replace with your file path
# Check if image loaded successfully
if img is None:
    print("Error: Could not read image.")
else:
    # Show the image in a window
    cv2.imshow("My Image", img)
    # Wait until a key is pressed, then close
    cv2.waitKey(0)
    cv2.destroyAllWindows()
```

Input:

Output:

Logic:

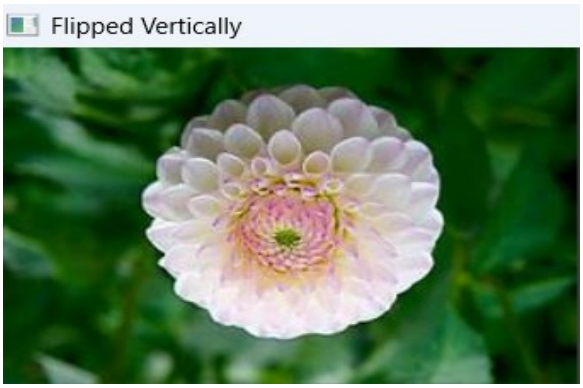
1. The program reads an image from the specified file path and displays it on the screen using OpenCV.
2. It first checks whether the image is loaded correctly. If the image is valid, it opens a window showing the image with a title. If the image is invalid, it displays an error message saying "Could not read the image."
3. The window remains open until the user presses any key, after which the program closes the window.

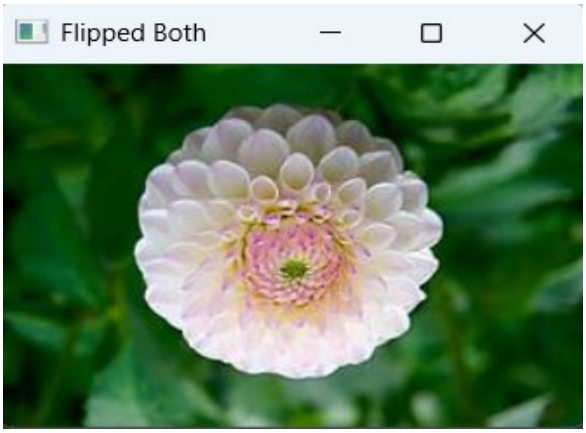
Code 4:

```
import cv2
# Read image
img = cv2.imread(r"C:\Users\leela\OneDrive\Desktop\Cv\img2.jpg")
if img is None:
    print("Error: Could not read image.")
else:
    # Flip vertically (0), horizontally (1), or both (-1)
    flip_vertical = cv2.flip(img, 0) #0 is for vertical flip
    flip_horizontal = cv2.flip(img, 1) #1 for horizontal flip
    flip_both = cv2.flip(img, -1) #-1 for both
  # Show results
    cv2.imshow("Original", img)
    cv2.imshow("Flipped Vertically", flip_vertical)
    cv2.imshow("Flipped Horizontally", flip_horizontal)
    cv2.imshow("Flipped Both", flip_both)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
```

input:

Output:





Logic:

This program loads an image and shows how to **flip it in different directions** using OpenCV. It creates three flipped versions:

- **Vertical flip (0):** turns the image upside down.
- **Horizontal flip (1):** mirrors the image left to right.
- **Both (-1):** flips the image vertically and horizontally.

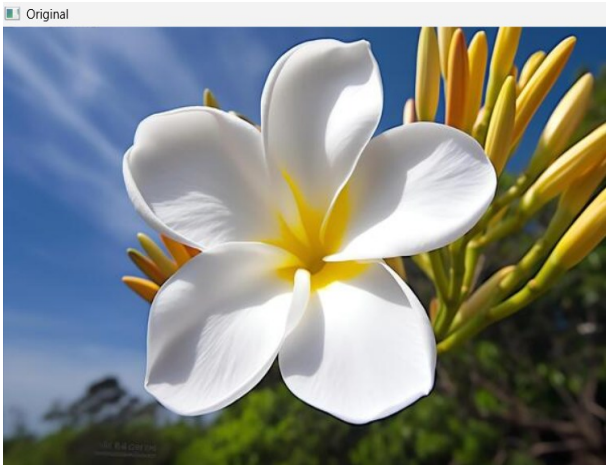
All the flipped images, along with the original, are displayed in separate windows. The windows stay open until the user presses any key, then all windows close.

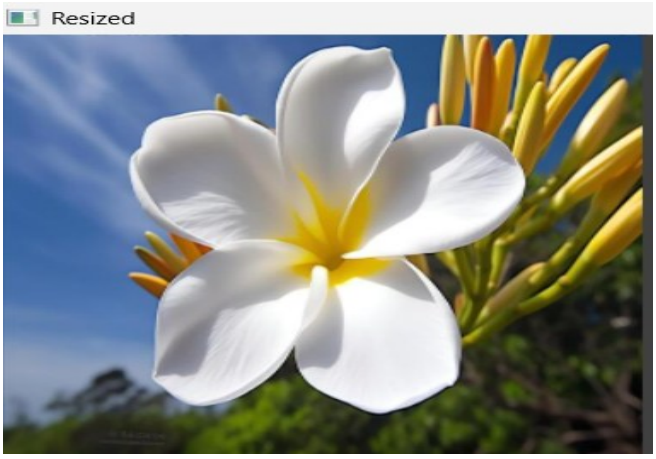
Code 5:

```
import cv2
# Load an image
img = cv2.imread(r"C:\Users\leela\OneDrive\Desktop\Cv\img3.jpg")   # Replace with your
file path
# Check if the image loaded correctly
if img is None:
    print("Error: Could not read image.")
    exit()
# Resize image (width=300, height=300)
resized = cv2.resize(img, (300, 300))
# Show both
cv2.imshow("Original", img)
cv2.imshow("Resized", resized)
# Wait for a key press
cv2.waitKey(0)
cv2.destroyAllWindows()
# Optional: Save the resized image
cv2.imwrite("resized_output.jpg", resized)
```

Input:

Output:





Logic:

1. This program loads an image from the computer and changes its size using OpenCV. It first checks whether the image is loaded correctly.
2. Using cv2.resize(), the image is resized to **300×300 pixels**.
3. Both the **original** and **resized** images are displayed in separate windows. The windows remain open until the user presses any key. Finally, the resized image is saved as **"resized_output.jpg"**.

Code 6:

```
import cv2

# Load an image

img = cv2.imread(r"C:\Users\leela\OneDrive\Desktop\Cv\img1.jpg")   # Replace with your file path

# Check if image loaded correctly

if img is None:

    print("Error: Could not read image.")

    exit()

# Convert to grayscale

gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

# Show both
```

```
cv2.imshow("Original", img)
cv2.imshow("Grayscale", gray)
# Wait until a key is pressed
cv2.waitKey(0)
cv2.destroyAllWindows()
# Optional: Save grayscale image
cv2.imwrite("grayscale_output.jpg", gray)
```

Input:

Output:

Logic:

1. This program loads a color image and converts it into **grayscale** using OpenCV. It first checks whether the image is loaded correctly.
2. The cv2.cvtColor() function changes the image from color (BGR) to grayscale, removing all color information. Both the **original color** image and the **grayscale** version are displayed in separate windows.
3. The windows stay open until the user presses any key. Finally, the grayscale image is saved as **"grayscale_output.jpg"**.

Code 7:

```
import cv2
# Load an image
img = cv2.imread(r"C:\Users\leela\OneDrive\Desktop\Cv\img4.jpg")   # Replace with your file path
# Check if image loaded correctly
if img is None:
    print("Error: Could not read image.")
    exit()
# Apply Gaussian Blur (15x15 kernel) mostly prefer odd number
#as kernel size. More the size of kernel more stronger blur
#less the no of size less stronger the kernel
blur = cv2.GaussianBlur(img, (15, 15), 0)
# Show both
cv2.imshow("Original", img)
cv2.imshow("Blurred", blur)
# Wait for key press
cv2.waitKey(0)
cv2.destroyAllWindows()
# Optional: Save the blurred image
cv2.imwrite("blurred_output.jpg", blur)
```

Input:

Output:

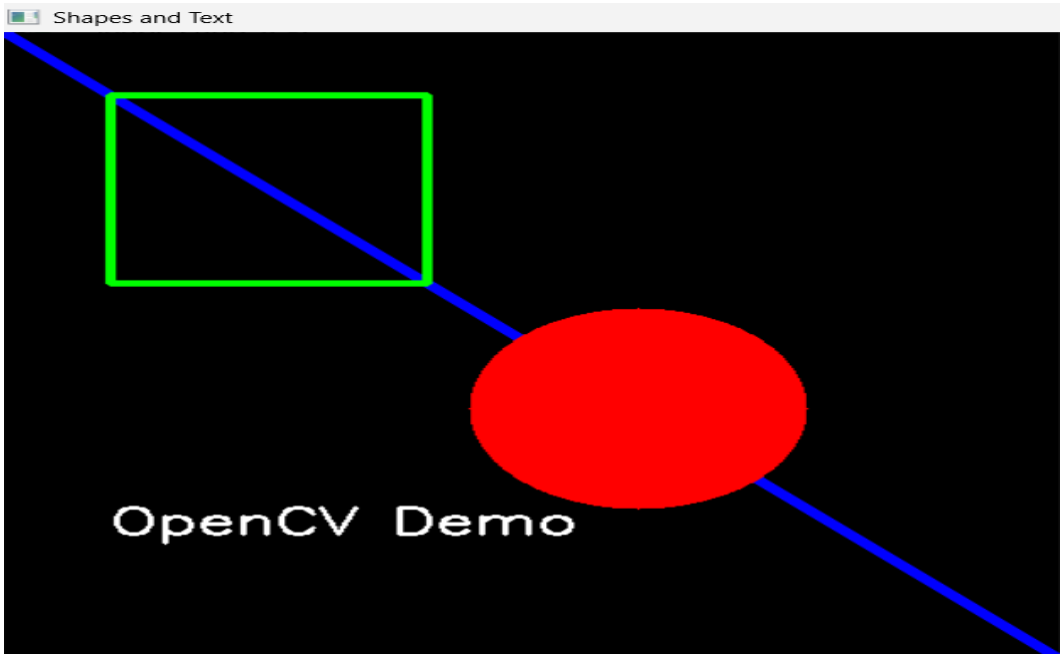
Logic:

1. This program applies a **Gaussian Blur** effect to an image using OpenCV. It first loads the image and checks if it is read correctly.
2. The cv2.GaussianBlur() function is used with a **15×15 kernel** to make the image smooth and reduce details or noise. A larger kernel makes the blur stronger, while a smaller one gives a lighter blur.
3. Both the **original** and **blurred** images are shown in separate windows. After pressing any key, all windows close. The blurred image is also saved as **"blurred_output.jpg"**.

Code 8:

```
import cv2
import numpy as np
img = np.zeros((500, 500, 3), dtype="uint8")
# Draw shapes
cv2.line(img, (0, 0), (500, 500), (255, 0, 0), 5)
cv2.rectangle(img, (50, 50), (200, 200), (0, 255, 0), 3)
cv2.circle(img, (300, 300), 80, (0, 0, 255), -1)
# Add text
cv2.putText(img, "OpenCV Demo", (50, 400), cv2.FONT_HERSHEY_SIMPLEX, 1, (255,
255, 255), 2)
cv2.imshow("Shapes and Text", img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Output:



Logic:

1. This program creates a **blank image** and draws different shapes and text on it using OpenCV.
2. It first makes a black image of size **500×500 pixels** using NumPy. Then, a **blue line**, a **green rectangle**, and a **red filled circle** are drawn using OpenCV drawing functions.
3. The text **"OpenCV Demo"** is added at the bottom of the image in white color.
4. Finally, the image is displayed in a window titled **"Shapes and Text"** until any key is pressed. This code shows how to draw basic shapes and add text to images in OpenCV.

Code 9:

```
import cv2

img = cv2.imread(r"C:\Users\leela\OneDrive\Desktop\Cv\flower.jpg", 0) # Load in grayscale_, thresh = cv2.threshold(img, 127, 255, cv2.THRESH_BINARY)

cv2.imshow("Original", img)

cv2.imshow("Thresholded", thresh)

cv2.waitKey(0)

cv2.destroyAllWindows()
```

Input:

Output:

Logic:

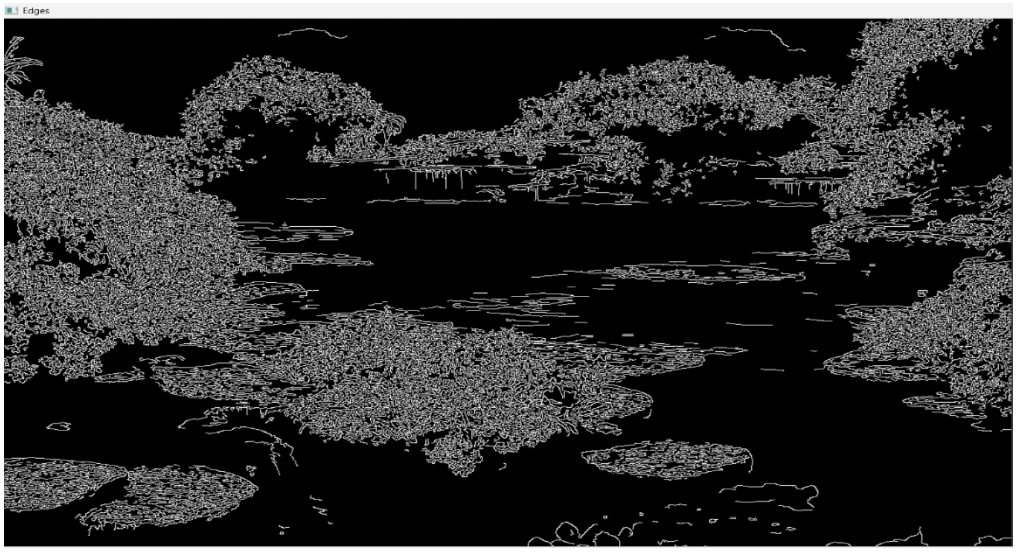
1. This program converts a grayscale image into a **black and white** image using thresholding in OpenCV.
2. The image is first loaded in grayscale. Then, the cv2.threshold() function is used with a **threshold value of 127** — pixels brighter than 127 become **white (255)**, and darker pixels become **black (0)**.
3. Both the **original grayscale** and **thresholded binary** images are displayed in separate windows.
4. The windows stay open until any key is pressed. This technique is useful for separating objects from the background in an image.

Code 10:

```
import cv2
img = cv2.imread(r"C:\Users\leela\OneDrive\Desktop\Cv\img7.jpg", 0)
edges = cv2.Canny(img, 100, 200)
cv2.imshow("Edges", edges)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Input:

Output:



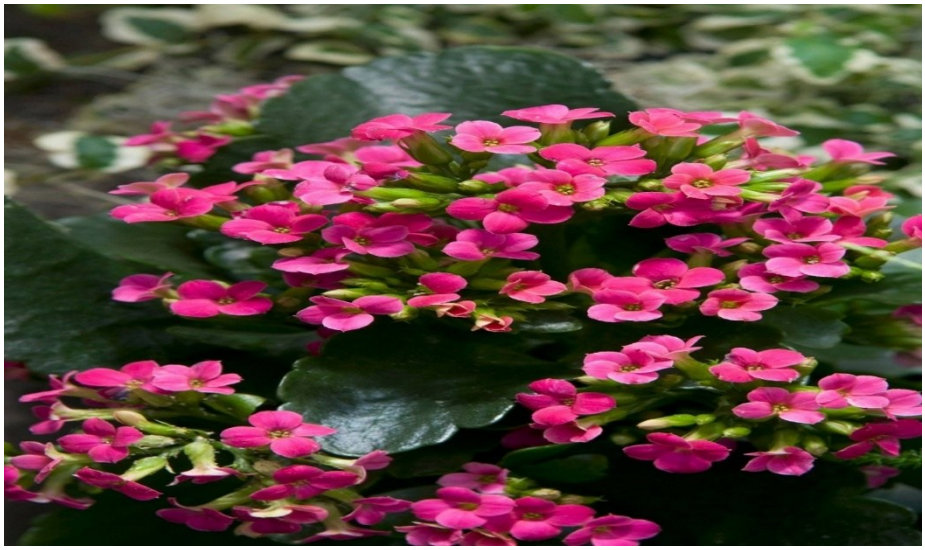
Logic:

1. This program detects the **edges** in an image using the **Canny Edge Detection** algorithm.
2. First, the image is loaded in **grayscale** mode. Then, cv2.Canny() is applied with two threshold values (100 and 200) — these control how strong an edge must be to appear in the result.
3. The output shows **only the boundaries and outlines** of objects in the image, highlighting areas where pixel intensity changes sharply.
4. This method is commonly used in **object detection, image analysis, and computer vision applications**.

Code 11:

```
import cv2
# Load pre-trained classifier
face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades +
"haarcascade_frontalface_default.xml")

img = cv2.imread("img5.jpg")
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

faces = face_cascade.detectMultiScale(gray, 1.1, 4)

for (x, y, w, h) in faces:
    cv2.rectangle(img, (x, y), (x+w, y+h), (255, 0, 0), 2)

cv2.imshow("Face Detection", img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Input:



Output:

Logic:

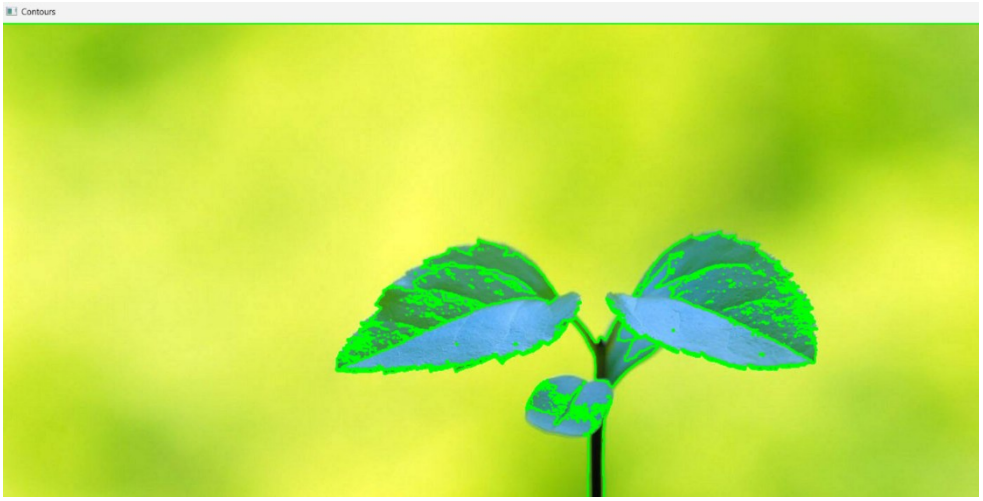
1. This program detects **faces** in an image using a **pre-trained Haar Cascade classifier** provided by OpenCV.
2. The image is first read and converted to **grayscale**, since face detection works faster and better on grayscale images.
3. The detectMultiScale() function scans the image to find face-like regions based on the trained pattern.
4. For every face detected, a **blue rectangle** is drawn around it. The final image is displayed, showing all detected faces.

Code 12:

```
import cv2

img = cv2.imread("img9.jpg")

gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

# Threshold

_, thresh = cv2.threshold(gray, 127, 255, cv2.THRESH_BINARY)

# Find contours

contours, _ = cv2.findContours(thresh, cv2.RETR_TREE,
cv2.CHAIN_APPROX_SIMPLE)

# Draw contours

cv2.drawContours(img, contours, -1, (0, 255, 0), 2)

cv2.imshow("Contours", img)

cv2.waitKey(0)

cv2.destroyAllWindows()
```

Input:

Output:



Logic:

1. The program reads the image and converts it to **grayscale** for easier processing.
2. It applies **thresholding** to convert the image into **black and white**, making it easier to identify object boundaries.
3. The findContours() function detects the **edges and outlines** of objects in the image.
4. The detected contours are then drawn in **green color** on the original image. Finally, the image with all contours is displayed.

Code13:

```
import cv2
img = cv2.imread("img10.jpg!d")
hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
# Define blue color range
lower_blue = (100, 150, 0)
upper_blue = (140, 255, 255)
mask = cv2.inRange(hsv, lower_blue, upper_blue)
result = cv2.bitwise_and(img, img, mask=mask)
cv2.imshow("Original", img)
cv2.imshow("Mask", mask)
cv2.imshow("Filtered", result)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Input:

Output:

Logic:

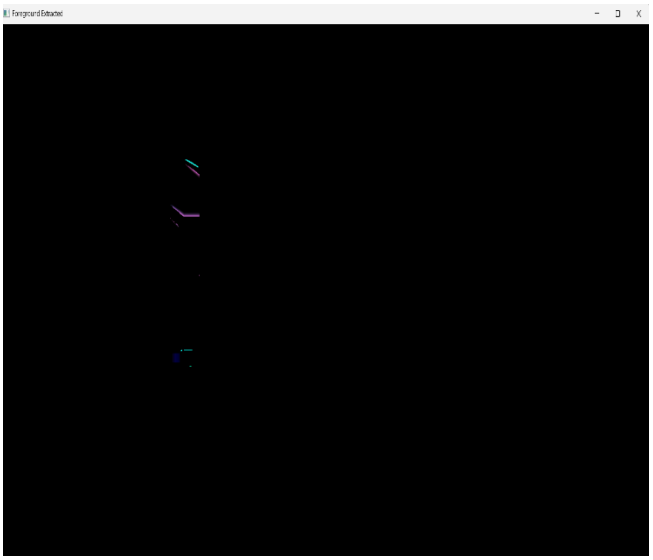
1. Reads the image and converts it to HSV color space (better for color detection).
2. Defines the HSV range for blue color.
3. Creates a mask that highlights only blue areas.
4. Uses bitwise_and() to extract and display only the blue-colored parts of the image.
5. Shows the original image, the mask, and the filtered (blue-only) result.

Code 14:

```
import cv2
import numpy as np
img = cv2.imread("img11.webp")
mask = np.zeros(img.shape[:2], np.uint8)
bgdModel = np.zeros((1, 65), np.float64)
fgdModel = np.zeros((1, 65), np.float64)
rect = (50, 50, 400, 500)  # ROI
cv2.grabCut(img, mask, rect, bgdModel, fgdModel, 5, cv2.GC_INIT_WITH_RECT)
mask2 = np.where((mask == 2) | (mask == 0), 0, 1).astype("uint8")
result = img * mask2[:, :, np.newaxis]
cv2.imshow("Original", img)
cv2.imshow("Foreground Extracted", result)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Input:

Output:



Logic:

1. Reads the image and creates an initial mask (all zeros).
2. Defines a rectangle (ROI) around the main object.
3. Uses GrabCut, an iterative segmentation algorithm, to separate the foreground (main object) from the background using color and edge information.
4. Updates the mask to keep only the foreground pixels.
5. Multiplies the mask with the original image to display only the extracted object, removing the background.
6. Shows the original image and the foreground-extracted result.

Code 15:

```
import cv2
cap = cv2.VideoCapture(0)
while True:
    ret, frame = cap.read()
    hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
    # Blue color range
    lower_blue = (100, 150, 0)
    upper_blue = (140, 255, 255)
    mask = cv2.inRange(hsv, lower_blue, upper_blue)
    result = cv2.bitwise_and(frame, frame, mask=mask)
    cv2.imshow("Frame", frame)
    cv2.imshow("Mask", mask)
    cv2.imshow("Tracked", result)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break
cap.release()
cv2.destroyAllWindows()
```

Output:

Code 16:

```
import cv2
import numpy as np
img = cv2.imread("img12.jpg", 0)
_, thresh = cv2.threshold(img, 127, 255, cv2.THRESH_BINARY_INV)
kernel = np.ones((5, 5), np.uint8)
erosion = cv2.erode(thresh, kernel, iterations=1)
dilation = cv2.dilate(thresh, kernel, iterations=1)
cv2.imshow("Original", thresh)
cv2.imshow("Erosion", erosion)
cv2.imshow("Dilation", dilation)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Input:

Output:

Logic:

1. Reads the image in grayscale.
2. Applies binary inverse thresholding to convert it into black and white.
3. Creates a 5×5 kernel (structuring element).
4. Erosion removes small white noises and shrinks white regions.
5. Dilation enlarges white areas and fills small holes.
6. Displays the thresholded, eroded, and dilated images.

File:1_preprocess.py

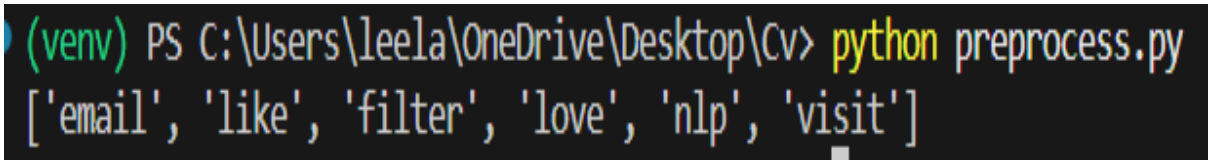
```
import re
from typing import List
from sklearn.feature_extraction.text import ENGLISH_STOP_WORDS
import spacy
nlp = spacy.load("en_core_web_sm")  # small, fast English model
def basic_clean(text: str) -> str:
    # lower, strip urls/emails/@mentions/hashtags, keep letters/numbers/space/apostrophe
    text = text.lower()
    text = re.sub(r"(http\S+|www\.\S+)", " ", text)
    text = re.sub(r"\S+@\S+", " ", text)
    text = re.sub(r"[@#]\w+", " ", text)
    text = re.sub(r"[^a-z0-9\s']", " ", text)
    text = re.sub(r"\s+", " ", text).strip()
    return text
def tokenize_stop_lemma(text: str) -> List[str]:
    doc = nlp(text)
    out = []
    for tok in doc:
        if tok.is_space or tok.is_punct:
            continue
        lemma = tok.lemma_.lower().strip()
        if len(lemma) < 3:          # drop very short tokens
            continue
        if lemma in ENGLISH_STOP_WORDS:  # sklearn's built-in stoplist
            continue
        out.append(lemma)
    return out
```

```
def preprocess(text: str) -> List[str]:
    return tokenize_stop_lemma(basic_clean(text))
if __name__ == "__main__":
    s = "Emails like help@site.com are filtered. I’m LOVING NLP!!! Visit https://x.y."
    print(preprocess(s))
```

Input:

s = "Emails like help@site.com are filtered. I’m LOVING NLP!!! Visit https://x.y."

Output:



Logic:

1. basic_clean() → Converts text to lowercase, removes URLs, emails, mentions, hashtags, and special characters, keeping only letters, numbers, spaces, and apostrophes.

2. tokenize_stop_lemma() → Uses spaCy to tokenize the cleaned text. Converts words to their lemmas (base forms). Removes stop words, punctuation, and short tokens (less than 3 characters).

3. preprocess() → Combines cleaning and tokenization steps to return a final list of meaningful words.

4. When run, it processes a sample sentence and prints the cleaned, lemmatized word list.

File:2_classify_tfidf.py

```
from sklearn.model_selection import train_test_split
from sklearn.pipeline import Pipeline
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix
import numpy as np
# tiny demo dataset (positive/negative sentiment)
texts = [
    "I loved this movie, fantastic acting and great story",
    "This film was terrible and boring",
    "Absolutely wonderful experience, highly recommend",
    "Worst acting ever, do not watch",
    "It was okay, some parts were fun",
    "I hated the plot, very disappointing",
    "Brilliant direction and superb cast",
    "Not good, waste of time",
    "Enjoyable and engaging from start to finish",
    "Awful soundtrack and weak story"
]
labels = np.array([1,0,1,0,1,0,1,0,1,0])  # 1=pos, 0=neg
X_train, X_test, y_train, y_test = train_test_split(texts, labels, test_size=0.3, random_state=42, stratify=labels)
pipe = Pipeline([
    ("tfidf", TfidfVectorizer(ngram_range=(1,2), min_df=1)),
    ("clf",   LogisticRegression(max_iter=1000))
])
pipe.fit(X_train, y_train)
y_pred = pipe.predict(X_test)
```

```
print("Classification report:\n", classification_report(y_test, y_pred, digits=4))
print("Confusion matrix:\n", confusion_matrix(y_test, y_pred))
# try the model
samples = ["pretty good but slow in places", "utterly awful, I want my time back"]
print("Predictions:", pipe.predict(samples))
print("Class probabilities:", pipe.predict_proba(samples))
```

Output:

```
(venv) PS C:\Users\leela\OneDrive\Desktop\Cv> python classify_tfidf.py

    accuracy                         0.3333         3
   macro avg     0.1667    0.5000    0.2500         3
weighted avg     0.1111    0.3333    0.1667         3

Confusion matrix:
 [[0 2]
 [0 1]]
Predictions: [1 1]
Class probabilities: [[0.42652933 0.57347067]
 [0.46719711 0.53280289]]
```

Logic:

1.Creates a small dataset of movie reviews labeled as positive (1) or negative (0).

2. Splits data into training (70%) and testing (30%) sets.

3. Builds a pipeline:

TfidfVectorizer → converts text into numerical features using TF-IDF (word importance).
LogisticRegression → learns to classify reviews as positive or negative.

4. Trains (fit) the model on the training data.

5. Predicts labels on the test data and prints: Classification report (precision, recall, f1-score). Confusion matrix (correct vs. wrong predictions).

6. Tests the model on new sample reviews and shows predicted sentiment and class probabilities.

File:3_tune_grid.py

```
from sklearn.model_selection import GridSearchCV
from sklearn.pipeline import Pipeline
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression
import numpy as np
texts = [
    "excellent movie with great acting",
    "terrible plot and awful pacing",
    "loved every moment, fantastic!",
    "boring and predictable",
    "superb cinematography and direction",
    "weak script and bad acting",
    "what a masterpiece",
    "not good at all",
    "brilliant experience overall",
    "do not recommend"
]
y = np.array([1,0,1,0,1,0,1,0,1,0])
pipe = Pipeline([
    ("tfidf", TfidfVectorizer()),
    ("clf", LogisticRegression(max_iter=1000))
])
param_grid = {
    "tfidf__ngram_range": [(1,1),(1,2)],
    "tfidf__min_df": [1,2],
    "tfidf__analyzer": ["word", "char_wb"],
    "clf__C": [0.25, 1.0, 4.0]  # regularization strength
}
```

```
search = GridSearchCV(pipe, param_grid, cv=3, n_jobs=-1, scoring="f1")
search.fit(texts, y)
print("Best params:", search.best_params_)
print("Best CV score (f1):", search.best_score_)
best_model = search.best_estimator_
print("Sample prediction:", best_model.predict(["not a great movie but had moments"]))
```

Output:

```
(venv) PS C:\Users\leela\OneDrive\Desktop\Cv> python tune_grid.py
Best params: {'clf__C': 4.0, 'tfidf__analyzer': 'char_wb', 'tfidf__min_df': 1, 'tfidf__ngram_range': (1, 2)}
Best CV score (f1): 0.7222222222222222
Sample prediction: [1]
```

Logic:

1. A small dataset of movie reviews is created with labels — 1 (positive) and 0 (negative).

2. A pipeline is built with two steps:

TfidfVectorizer → converts text to numerical TF-IDF features.

LogisticRegression → classifies text sentiment.

3. A parameter grid (param_grid) defines combinations of tuning options:

Different ngram ranges (single words or word pairs).
Minimum word frequency (min_df).
Feature extraction method (word vs. char_wb).
Logistic Regression's regularization strength (C).

4. GridSearchCV tests all parameter combinations using 3-fold cross-validation and selects the best based on F1-score.

5. Prints the best parameters, best cross-validation score, and uses the best model to predict sentiment for a new sample sentence.

File:4_tfidf_demo.py

```
from sklearn.feature_extraction.text import TfidfVectorizer
import pandas as pd
# Sample documents
docs = [
    "machine learning is fun",
    "deep learning advances machine intelligence",
    "artificial intelligence and machine learning"
]
# Create TF-IDF model
tfidf = TfidfVectorizer()
X = tfidf.fit_transform(docs)
# Convert to DataFrame
tfidf_df = pd.DataFrame(X.toarray(), columns=tfidf.get_feature_names_out())
print("Vocabulary:", tfidf.get_feature_names_out())
print("\nTF-IDF Matrix:")
print(tfidf_df.round(3))
```

Output:

```
(venv) PS C:\Users\leela\OneDrive\Desktop\Cv> python tfidf_demo.py
Vocabulary: ['advances' 'and' 'artificial' 'deep' 'fun' 'intelligence' 'is' 'learning'
 'machine']

TF-IDF Matrix:
   advances    and  artificial   deep    fun  intelligence     is  learning  machine
0     0.000  0.000       0.000  0.000  0.609          0.00  0.609     0.360    0.360
1     0.552  0.000       0.000  0.552  0.000          0.42  0.000     0.326    0.326
2     0.000  0.552       0.552  0.000  0.000          0.42  0.000     0.326    0.326
```

Logic:

1. Defines a small list of text documents.

2. Uses TfidfVectorizer to:

Learn the vocabulary (all unique words).

Compute TF-IDF scores for each word in each document (importance of a word relative to the document and corpus).

3. Converts the resulting sparse matrix into a Pandas DataFrame for easy viewing.

4. Prints the vocabulary and the TF-IDF values for each document.

File:5_spacy_ner_pos.py

```
import spacy
from pprint import pprint
nlp = spacy.load("en_core_web_sm")
text = ("Apple is opening a new office in Bengaluru next quarter. "
        "Tim Cook met Karnataka officials on September 3, 2025 to discuss expansion.")
doc = nlp(text)
print("\nNamed Entities (text, label):")
for ent in doc.ents:
    print(f"{ent.text:<25}  -> {ent.label_}"
print("\nPart-of-Speech & Lemmas:")
for token in doc:
    if not token.is_space:
        print(f"{token.text:<15} POS={token.pos_:<5}  Lemma={token.lemma_}")
print("\nNoun chunks (base NPs):")
pprint([chunk.text for chunk in doc.noun_chunks])
```

Output:

```
(venv) PS C:\Users\leela\OneDrive\Desktop\Cv> python spacy_ner_pos.py

Named Entities (text, label):
Apple                          -> ORG
Bengaluru                      -> GPE
next quarter                   -> DATE
Tim Cook                       -> PERSON
Karnataka                      -> GPE
September 3, 2025              -> DATE

Part-of-Speech & Lemmas:
Apple             POS=PROPN   Lemma=Apple
is                POS=AUX     Lemma=be
opening           POS=VERB    Lemma=open
a                 POS=DET     Lemma=a
new               POS=ADJ     Lemma=new
office            POS=NOUN    Lemma=office
in                POS=ADP     Lemma=in
Bengaluru         POS=PROPN   Lemma=Bengaluru
next              POS=ADJ     Lemma=next
quarter           POS=NOUN    Lemma=quarter
.                 POS=PUNCT   Lemma=.
Tim               POS=PROPN   Lemma=Tim
Cook              POS=PROPN   Lemma=Cook
met               POS=VERB    Lemma=meet
Karnataka         POS=PROPN   Lemma=Karnataka
officials         POS=NOUN    Lemma=official
on                POS=ADP     Lemma=on
September         POS=PROPN   Lemma=September
3                 POS=NUM     Lemma=3
,                 POS=PUNCT   Lemma=,
2025              POS=NUM     Lemma=2025
to                POS=PART    Lemma=to
discuss           POS=VERB    Lemma=discuss
expansion         POS=NOUN    Lemma=expansion
.                 POS=PUNCT   Lemma=.
```

```
Noun chunks (base NPs):
['Apple',
 'a new office',
 'Bengaluru',
 'Tim Cook',
 'Karnataka officials',
 'September',
 'expansion']
(venv) PS C:\Users\leela\OneDrive\Desktop\Cv>
```

Logic:

1. Loads the small English model (en_core_web_sm) in spaCy.

2. Processes the text to create a Doc object that contains linguistic annotations.

3. Named Entity Recognition (NER): Extracts entities like organizations, dates, locations, and prints their text and type.

4. Part-of-Speech (POS) tagging & Lemmatization: Prints each token's POS tag (noun, verb, etc.) and lemma (base form).

5. Noun Chunks: Extracts base noun phrases (grouped nouns with modifiers) from the text.

File:6_nb_classify.py

```
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import classification_report
# Tiny sentiment dataset
texts = [
    "I love this movie",
    "This film was awful",
    "Amazing performance and great story",
    "Boring and too long",
    "Fantastic acting",
    "Terrible direction"
]
labels = [1, 0, 1, 0, 1, 0]  # 1=positive, 0=negative
# Split data
X_train, X_test, y_train, y_test = train_test_split(texts, labels, test_size=0.3, random_state=42)
# Bag of Words + Naive Bayes
vectorizer = CountVectorizer()
X_train_bow = vectorizer.fit_transform(X_train)
X_test_bow = vectorizer.transform(X_test)
clf = MultinomialNB()
clf.fit(X_train_bow, y_train)
y_pred = clf.predict(X_test_bow)
print("Classification Report:\n", classification_report(y_test, y_pred))
```

Output:

```
(venv) PS C:\Users\leela\OneDrive\Desktop\Cv> python nb_classify.py
Classification Report:
              precision    recall  f1-score   support

           0       0.50      1.00      0.67         1
           1       0.00      0.00      0.00         1

    accuracy                           0.50         2
   macro avg       0.25      0.50      0.33         2
weighted avg       0.25      0.50      0.33         2
```

Logic:

1. Defines a small dataset of movie reviews labeled as positive (1) or negative (0).

2. Splits data into training (70%) and testing (30%) sets.

3. Uses CountVectorizer to convert text into Bag-of-Words representation (counts of each word in the vocabulary).

4. Trains a Multinomial Naive Bayes classifier on the BoW features from training data.

5. Predicts labels for the test data.

6. Prints a classification report showing precision, recall, f1-score, and accuracy.

File:7_similarity_demo

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
# Example documents
docs = [
    "I love machine learning and NLP",
    "NLP and machine learning are amazing",
    "Cooking recipes are fun to try",
]
# TF-IDF representation
vectorizer = TfidfVectorizer()
X = vectorizer.fit_transform(docs)
# Compute cosine similarity
sim_matrix = cosine_similarity(X)
print("Cosine Similarity Matrix:\n", sim_matrix)
```

Output:

```
(venv) PS C:\Users\leela\OneDrive\Desktop\Cv> python similarity_demo.py
Cosine Similarity Matrix:
 [[1.         0.64424596 0.        ]
 [0.64424596 1.         0.12413287]
 [0.         0.12413287 1.        ]]
(venv) PS C:\Users\leela\OneDrive\Desktop\Cv>
```

Logic:

1. Defines a list of documents.

2. Uses TfidfVectorizer to convert documents into TF-IDF vectors, capturing the importance of words in each document.

3. Computes cosine similarity between all pairs of TF-IDF vectors, which measures how similar two documents are based on content.

4. Prints the similarity matrix, where values range from 0 (no similarity) to 1 (identical).

File:8_topic_modeling.py

```
from gensim import corpora, models
# Example dataset
docs = [
    "I love deep learning and natural language processing",
    "Artificial intelligence is the future",
    "Cooking and baking are my hobbies",
    "I enjoy trying new recipes in the kitchen",
    "Machine learning and AI are closely related"
]
# Tokenize
texts = [doc.lower().split() for doc in docs]
# Create dictionary & corpus
dictionary = corpora.Dictionary(texts)
corpus = [dictionary.doc2bow(text) for text in texts]
# Train LDA model (2 topics)
lda_model = models.LdaModel(corpus, num_topics=2, id2word=dictionary, passes=10)
# Show topics
for idx, topic in lda_model.print_topics(-1):
    print(f"Topic {idx}: {topic}")
```

Output:

```
(venv) PS C:\Users\leela\OneDrive\Desktop\Cv> python topic_modeling.py
Topic 0: 0.094*"and" + 0.092*"are" + 0.056*"learning" + 0.055*"ai" + 0.055*"machine" + 0.055*"related" + 0.055*"closely" + 0.055*"hobbies" + 0.
055*"my" + 0.055*"baking"
Topic 1: 0.072*"i" + 0.071*"the" + 0.043*"language" + 0.043*"natural" + 0.043*"love" + 0.043*"processing" + 0.043*"kitchen" + 0.043*"deep" + 0.
043*"recipes" + 0.043*"in"
```

Logic:

1. Defines a list of documents.

2. Tokenizes each document into lowercase words.

3. Creates a dictionary (mapping of unique words to IDs) and a corpus (bag-of-words representation for each document).

4. Trains an LDA model to discover 2 latent topics from the corpus (passes=10 controls training iterations).

5. Prints the topics, showing the most important words for each topic.