UNIVERSITY OF WESTMINSTER

IIT INFORMATICS INSTITUTE OF TECHNOLOGY

5DATA005W Data Engineering

Coursework 2

Lecturer Name: Ms. Yaalini Balathasan

- **Name : W A Sandasmi Dewni**
- **Blackboard Name : Wijesuriya Dewni**
- **IIT Index Number : 20231373**
- **UOW Index : W20521491**

Table of Contents

1. Objectives

1. Database for Content-Based Image Retrieval (CBIR) System

The purpose of this database is to retrieve 50 images from the database called “CBIR” using NoSQL on MongoDB. This CBIR system helps overcome image mining issues such as inconsistent pixel presentation, unmatching colours and other features. (Munjal, 2018)

CBIR comprises the following steps:

1. Image collection
2. Feature extraction.
3. Database implementation.
4. The visual features are extracted from the image database and stored in the feature matrix.
5. The features of the query image are matched from the image database using some similarity-matching methods.
6. At the end, the desired image is retrieved. (Munjal, 2018)

2. Databases for Sentiment Analysis Models

The purpose of Sentimental Analysis is to tag 50 text files according to their sentiment, such as positive, negative, and neutral. (Pascual, 2022) The data is loaded to a NoSQL database called ‘NLP’. NLP consists of the following steps.

1. Data Collection
2. Text Preprocessing
3. Sentiment Scoring
4. Machine Learning Models
5. Polarity Extraction (HyScaler, 2024)

2. Data Sources

1. Database for Content-Based Image Retrieval (CBIR) System

The raw 50 images were extracted from multiple sources such as Unsplash, Pixabay and Pexels.
The link to the Google Drive containing the unprocessed image file:
https://drive.google.com/drive/folders/1w5ya2k2aJOaxnXwIiYTnau2t6vB4Fs5y?usp=drive_link
I ensured the images covered numerous subjects such as animals, fashion, transport, food, devices, products, nature, etc. They also cover different styles and characteristics.

2. Databases for Sentiment Analysis Models

The raw 50 text files were extracted from two sources.
The link to the Google Drive containing the unprocessed text file:
https://drive.google.com/drive/folders/1iEQY5-mT1QuODd7cFJZLv9F6ZAh2whdv?usp=drive_link

45 out of 50 text files were randomly collected from the data file named '20_newsgroup' used for the tutorial in Week 7 (link: https://www.cs.cmu.edu/afs/cs/project/theo-20/www/data/news20.html)

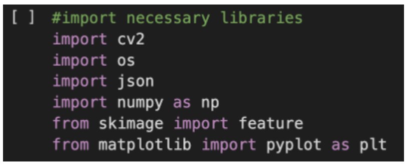
The other five files that contain IMDB reviews were obtained from this data source: https://www.kaggle.com/datasets/lakshmi25npathi/imdb-dataset-of-50k-movie-reviews

These 50 textual data files cover categories such as movie reviews, customer reviews, political and religious views, sports comments, product reviews, etc.

3. Methodology

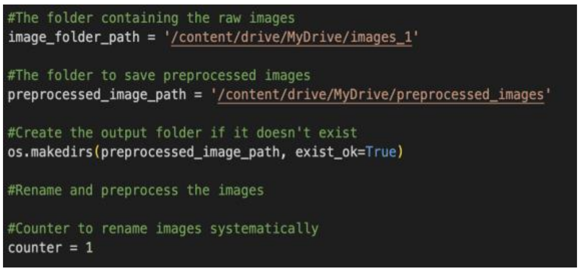
1. Database for Content-Based Image Retrieval (CBIR) System

1. Image Preprocessing



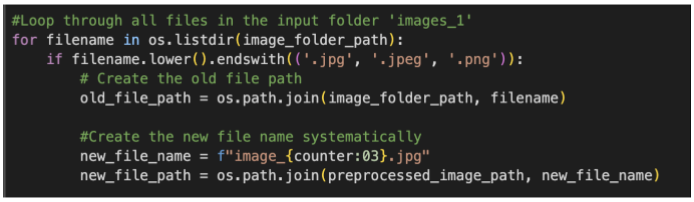
```
[ ] #import necessary libraries
    import cv2
    import os
    import json
    import numpy as np
    from skimage import feature
    from matplotlib import pyplot as plt
```

Necessary Python libraries were imported on Google Colab to start the preprocessing of images.



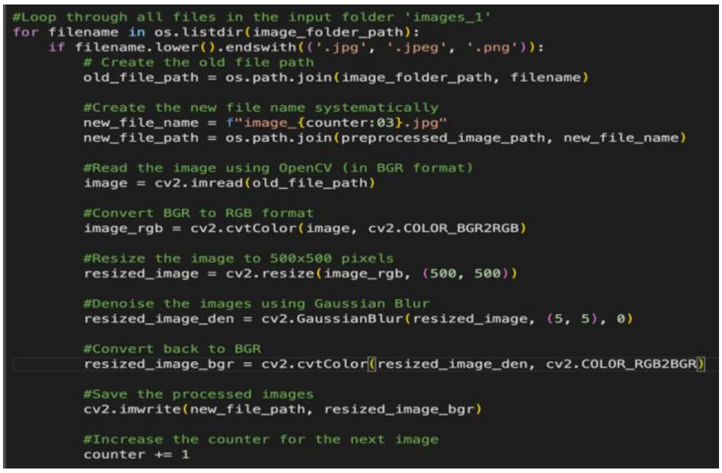
```
#The folder containing the raw images
image_folder_path = '/content/drive/MyDrive/images_1'

#The folder to save preprocessed images
preprocessed_image_path = '/content/drive/MyDrive/preprocessed_images'

#Create the output folder if it doesn't exist
os.makedirs(preprocessed_image_path, exist_ok=True)

#Rename and preprocess the images

#Counter to rename images systematically
counter = 1
```

The file named ‘images_1’, containing raw, unprocessed images is specified and the path for the folder is mentioned. The preprocessed images are to be saved in the specified path named ‘preprocessed_image_path’. The files were saved in the Google Drive connected to Colab. ‘os.makedirs’ ensures that the selected folder exists.



```
#Loop through all files in the input folder 'images_1'
for filename in os.listdir(image_folder_path):
    if filename.lower().endswith(('.jpg', '.jpeg', '.png')):
        # Create the old file path
        old_file_path = os.path.join(image_folder_path, filename)

        #Create the new file name systematically
        new_file_name = f"image_{counter:03}.jpg"
        new_file_path = os.path.join(preprocessed_image_path, new_file_name)
```

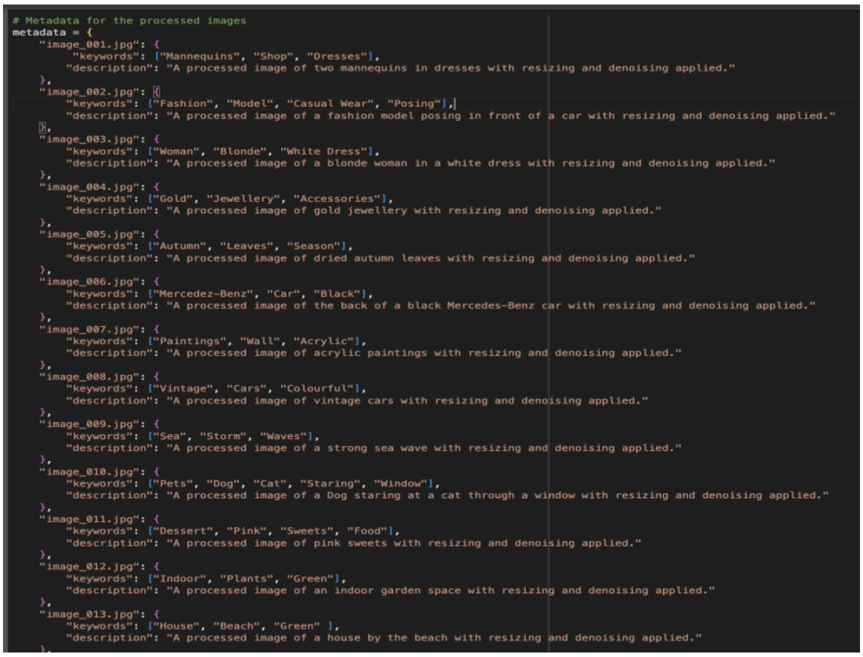
The ‘for’ loop iterates through all the unprocessed images that end with ‘jpg’, ‘jpeg’, and ‘png’. Then the images are renamed systematically (e.g: image_001.jpg, image_002.jpg, etc.) and saved to the new file path created using ‘os.path.join’.



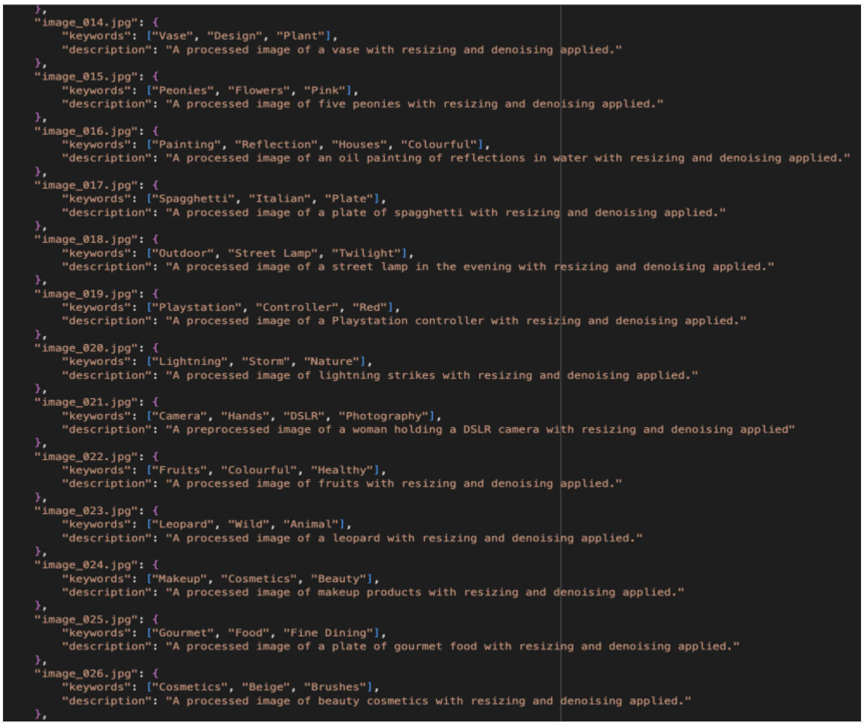
```python
#Loop through all files in the input folder 'images_1'
for filename in os.listdir(image_folder_path):
    if filename.lower().endswith(('.jpg', '.jpeg', '.png')):
        # Create the old file path
        old_file_path = os.path.join(image_folder_path, filename)

        #Create the new file name systematically
        new_file_name = f"image_{counter:03}.jpg"
        new_file_path = os.path.join(preprocessed_image_path, new_file_name)

        #Read the image using OpenCV (in BGR format)
        image = cv2.imread(old_file_path)

        #Convert BGR to RGB format
        image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

        #Resize the image to 500x500 pixels
        resized_image = cv2.resize(image_rgb, (500, 500))

        #Denoise the images using Gaussian Blur
        resized_image_den = cv2.GaussianBlur(resized_image, (5, 5), 0)

        #Convert back to BGR
        resized_image_bgr = cv2.cvtColor(resized_image_den, cv2.COLOR_RGB2BGR)

        #Save the processed images
        cv2.imwrite(new_file_path, resized_image_bgr)

        #Increase the counter for the next image
        counter += 1
```

The 'cv2.imread' function reads the images in BGR format using OpenCV and then converts them to the RGB format using the function 'cv2.cvtColor', to prepare the images for processing.
'cv2.resize' resizes the images to 500x500 pixels as instructed.
'cv2.GaussianBlur' denoises the images using Gaussian Blur for further processing. Then, the function 'cv2.cvtColor' is reused to convert the denoised images to BGR format to save the file. The line 'Counter += 1' ensures that the counter for the next image is increased.

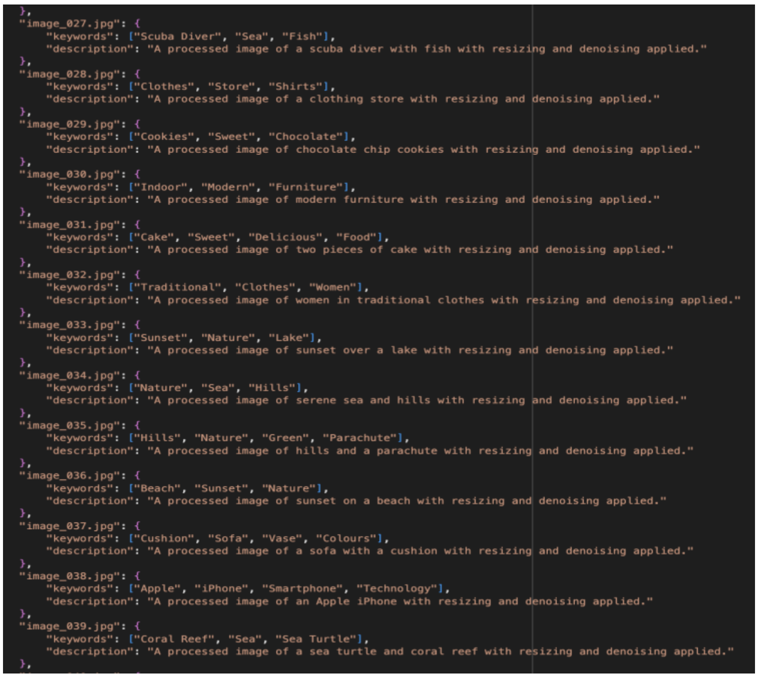
2. Image Annotation



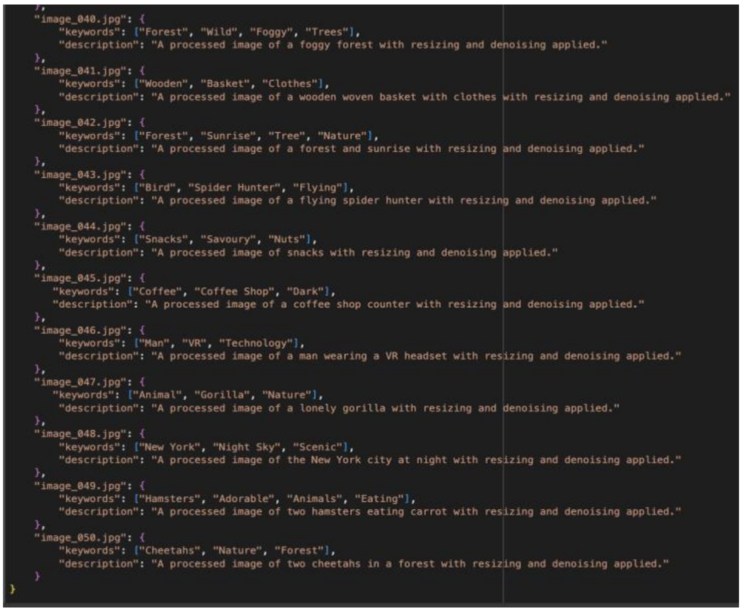
```python
# Metadata for the processed images
metadata = {
    "image_001.jpg": {
          "keywords": ["Mannequins", "Shop", "Dresses"],
        "description": "A processed image of two mannequins in dresses with resizing and denoising applied."
    },
    "image_002.jpg": {
        "keywords": ["Fashion", "Model", "Casual Wear", "Posing"],
        "description": "A processed image of a fashion model posing in front of a car with resizing and denoising applied."
    },
    "image_003.jpg": {
        "keywords": ["Woman", "Blonde", "White Dress"],
        "description": "A processed image of a blonde woman in a white dress with resizing and denoising applied."
    },
    "image_004.jpg": {
        "keywords": ["Gold", "Jewellery", "Accessories"],
        "description": "A processed image of gold jewellery with resizing and denoising applied."
    },
    "image_005.jpg": {
        "keywords": ["Autumn", "Leaves", "Season"],
        "description": "A processed image of dried autumn leaves with resizing and denoising applied."
    },
    "image_006.jpg": {
        "keywords": ["Mercedez-Benz", "Car", "Black"],
        "description": "A processed image of the back of a black Mercedes-Benz car with resizing and denoising applied."
    },
    "image_007.jpg": {
        "keywords": ["Paintings", "Wall", "Acrylic"],
        "description": "A processed image of acrylic paintings with resizing and denoising applied."
    },
    "image_008.jpg": {
        "keywords": ["Vintage", "Cars", "Colourful"],
        "description": "A processed image of vintage cars with resizing and denoising applied."
    },
    "image_009.jpg": {
        "keywords": ["Sea", "Storm", "Waves"],
        "description": "A processed image of a strong sea wave with resizing and denoising applied."
    },
    "image_010.jpg": {
        "keywords": ["Pets", "Dog", "Cat", "Staring", "Window"],
        "description": "A processed image of a Dog staring at a cat through a window with resizing and denoising applied."
    },
    "image_011.jpg": {
        "keywords": ["Dessert", "Pink", "Sweets", "Food"],
        "description": "A processed image of pink sweets with resizing and denoising applied."
    },
    "image_012.jpg": {
        "keywords": ["Indoor", "Plants", "Green"],
        "description": "A processed image of an indoor garden space with resizing and denoising applied."
    },
    "image_013.jpg": {
        "keywords": ["House", "Beach", "Green" ],
        "description": "A processed image of a house by the beach with resizing and denoising applied."
    },
```



```
    },
    "image_014.jpg": {
        "keywords": ["Vase", "Design", "Plant"],
        "description": "A processed image of a vase with resizing and denoising applied."
    },
    "image_015.jpg": {
        "keywords": ["Peonies", "Flowers", "Pink"],
        "description": "A processed image of five peonies with resizing and denoising applied."
    },
    "image_016.jpg": {
        "keywords": ["Painting", "Reflection", "Houses", "Colourful"],
        "description": "A processed image of an oil painting of reflections in water with resizing and denoising applied."
    },
    "image_017.jpg": {
        "keywords": ["Spagghetti", "Italian", "Plate"],
        "description": "A processed image of a plate of spagghetti with resizing and denoising applied."
    },
    "image_018.jpg": {
        "keywords": ["Outdoor", "Street Lamp", "Twilight"],
        "description": "A processed image of a street lamp in the evening with resizing and denoising applied."
    },
    "image_019.jpg": {
        "keywords": ["Playstation", "Controller", "Red"],
        "description": "A processed image of a Playstation controller with resizing and denoising applied."
    },
    "image_020.jpg": {
        "keywords": ["Lightning", "Storm", "Nature"],
        "description": "A processed image of lightning strikes with resizing and denoising applied."
    },
    "image_021.jpg": {
        "keywords": ["Camera", "Hands", "DSLR", "Photography"],
        "description": "A preprocessed image of a woman holding a DSLR camera with resizing and denoising applied"
    },
    "image_022.jpg": {
        "keywords": ["Fruits", "Colourful", "Healthy"],
        "description": "A processed image of fruits with resizing and denoising applied."
    },
    "image_023.jpg": {
        "keywords": ["Leopard", "Wild", "Animal"],
        "description": "A processed image of a leopard with resizing and denoising applied."
    },
    "image_024.jpg": {
        "keywords": ["Makeup", "Cosmetics", "Beauty"],
        "description": "A processed image of makeup products with resizing and denoising applied."
    },
    "image_025.jpg": {
        "keywords": ["Gourmet", "Food", "Fine Dining"],
        "description": "A processed image of a plate of gourmet food with resizing and denoising applied."
    },
    "image_026.jpg": {
        "keywords": ["Cosmetics", "Beige", "Brushes"],
        "description": "A processed image of beauty cosmetics with resizing and denoising applied."
    },
```



```
  },
  "image_027.jpg": {
      "keywords": ["Scuba Diver", "Sea", "Fish"],
      "description": "A processed image of a scuba diver with fish with resizing and denoising applied."
  },
  "image_028.jpg": {
      "keywords": ["Clothes", "Store", "Shirts"],
      "description": "A processed image of a clothing store with resizing and denoising applied."
  },
  "image_029.jpg": {
      "keywords": ["Cookies", "Sweet", "Chocolate"],
      "description": "A processed image of chocolate chip cookies with resizing and denoising applied."
  },
  "image_030.jpg": {
      "keywords": ["Indoor", "Modern", "Furniture"],
      "description": "A processed image of modern furniture with resizing and denoising applied."
  },
  "image_031.jpg": {
      "keywords": ["Cake", "Sweet", "Delicious", "Food"],
      "description": "A processed image of two pieces of cake with resizing and denoising applied."
  },
  "image_032.jpg": {
      "keywords": ["Traditional", "Clothes", "Women"],
      "description": "A processed image of women in traditional clothes with resizing and denoising applied."
  },
  "image_033.jpg": {
      "keywords": ["Sunset", "Nature", "Lake"],
      "description": "A processed image of sunset over a lake with resizing and denoising applied."
  },
  "image_034.jpg": {
      "keywords": ["Nature", "Sea", "Hills"],
      "description": "A processed image of serene sea and hills with resizing and denoising applied."
  },
  "image_035.jpg": {
      "keywords": ["Hills", "Nature", "Green", "Parachute"],
      "description": "A processed image of hills and a parachute with resizing and denoising applied."
  },
  "image_036.jpg": {
      "keywords": ["Beach", "Sunset", "Nature"],
      "description": "A processed image of sunset on a beach with resizing and denoising applied."
  },
  "image_037.jpg": {
      "keywords": ["Cushion", "Sofa", "Vase", "Colours"],
      "description": "A processed image of a sofa with a cushion with resizing and denoising applied."
  },
  "image_038.jpg": {
      "keywords": ["Apple", "iPhone", "Smartphone", "Technology"],
      "description": "A processed image of an Apple iPhone with resizing and denoising applied."
  },
  "image_039.jpg": {
      "keywords": ["Coral Reef", "Sea", "Sea Turtle"],
      "description": "A processed image of a sea turtle and coral reef with resizing and denoising applied."
  },
```

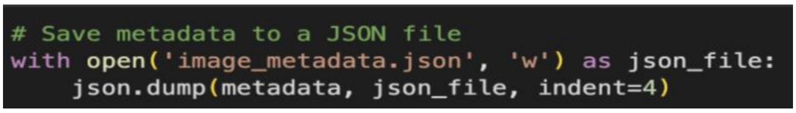


```
    },
    "image_040.jpg": {
        "keywords": ["Forest", "Wild", "Foggy", "Trees"],
        "description": "A processed image of a foggy forest with resizing and denoising applied."
    },
    "image_041.jpg": {
        "keywords": ["Wooden", "Basket", "Clothes"],
        "description": "A processed image of a wooden woven basket with clothes with resizing and denoising applied."
    },
    "image_042.jpg": {
        "keywords": ["Forest", "Sunrise", "Tree", "Nature"],
        "description": "A processed image of a forest and sunrise with resizing and denoising applied."
    },
    "image_043.jpg": {
        "keywords": ["Bird", "Spider Hunter", "Flying"],
        "description": "A processed image of a flying spider hunter with resizing and denoising applied."
    },
    "image_044.jpg": {
        "keywords": ["Snacks", "Savoury", "Nuts"],
        "description": "A processed image of snacks with resizing and denoising applied."
    },
    "image_045.jpg": {
       "keywords": ["Coffee", "Coffee Shop", "Dark"],
       "description": "A processed image of a coffee shop counter with resizing and denoising applied."
    },
    "image_046.jpg": {
        "keywords": ["Man", "VR", "Technology"],
        "description": "A processed image of a man wearing a VR headset with resizing and denoising applied."
    },
    "image_047.jpg": {
       "keywords": ["Animal", "Gorilla", "Nature"],
        "description": "A processed image of a lonely gorilla with resizing and denoising applied."
    },
    "image_048.jpg": {
        "keywords": ["New York", "Night Sky", "Scenic"],
        "description": "A processed image of the New York city at night with resizing and denoising applied."
    },
    "image_049.jpg": {
        "keywords": ["Hamsters", "Adorable", "Animals", "Eating"],
        "description": "A processed image of two hamsters eating carrot with resizing and denoising applied."
    },
    "image_050.jpg": {
        "keywords": ["Cheetahs", "Nature", "Forest"],
        "description": "A processed image of two cheetahs in a forest with resizing and denoising applied."
    }
}
```

The processed 50 images are manually annotated with metadata using the following key elements:

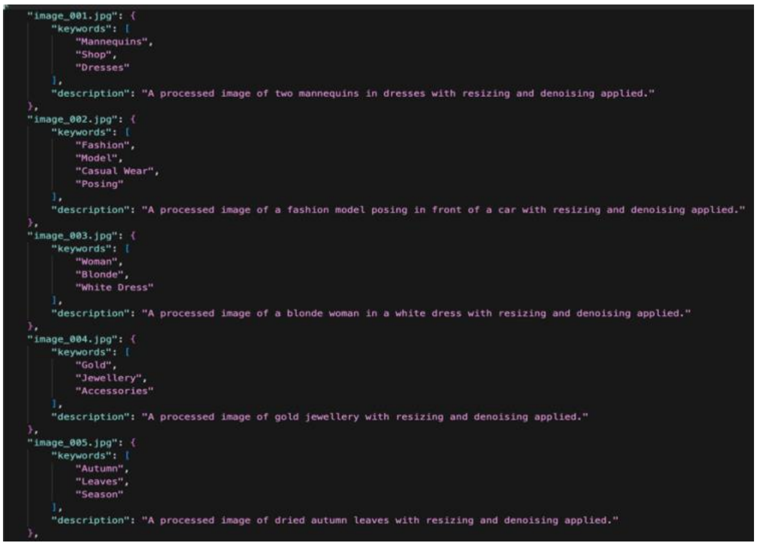
- Image Filename: The name of the processed image file (e.g. image_001.jpg, image_002.jpg, etc.)
- Keywords: Describes each image based on its subjects and characteristics. (e.g; Animal, Scenic, Spagghetti)
- Description: Provides additional details of each image and the steps used to process it.

This information is the ground truth for the evaluation of the CBIR system.



```
# Save metadata to a JSON file
with open('image_metadata.json', 'w') as json_file:
    json.dump(metadata, json_file, indent=4)
```

The metadata above is saved to a JSON file named ‘image_metadata.json’

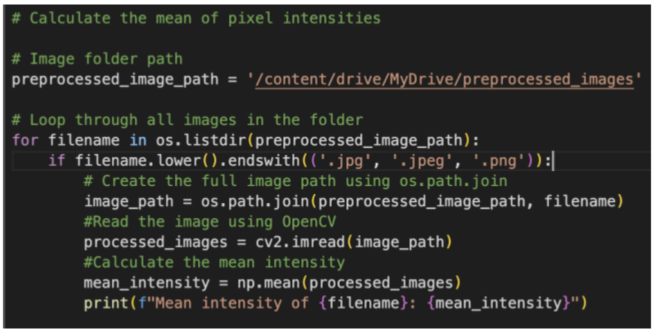


```
    "image_001.jpg": {
        "keywords": [
            "Mannequins",
            "Shop",
            "Dresses"
        ],
        "description": "A processed image of two mannequins in dresses with resizing and denoising applied."
    },
    "image_002.jpg": {
        "keywords": [
            "Fashion",
            "Model",
            "Casual Wear",
            "Posing"
        ],
        "description": "A processed image of a fashion model posing in front of a car with resizing and denoising applied."
    },
    "image_003.jpg": {
        "keywords": [
            "Woman",
            "Blonde",
            "White Dress"
        ],
        "description": "A processed image of a blonde woman in a white dress with resizing and denoising applied."
    },
    "image_004.jpg": {
        "keywords": [
            "Gold",
            "Jewellery",
            "Accessories"
        ],
        "description": "A processed image of gold jewellery with resizing and denoising applied."
    },
    "image_005.jpg": {
        "keywords": [
            "Autumn",
            "Leaves",
            "Season"
        ],
        "description": "A processed image of dried autumn leaves with resizing and denoising applied."
    },
```

The metadata for the first five processed images in the JSON file is shown in the screenshot above.

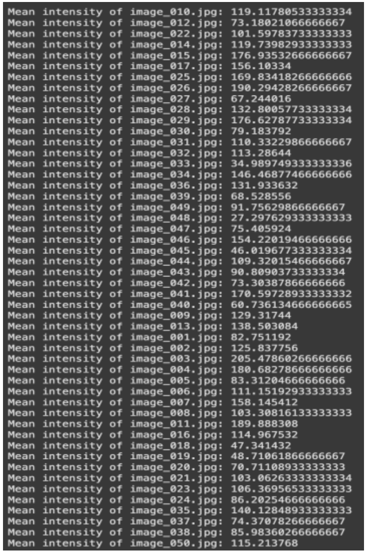
3. Image Feature Extraction

- ### The Mean of Pixel Intensities



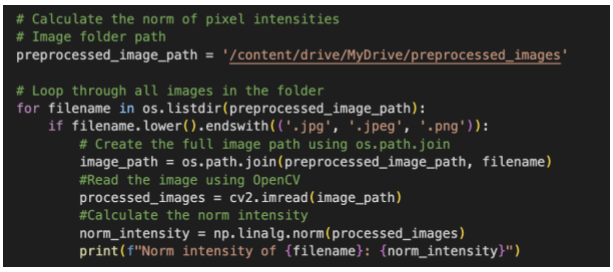
```
# Calculate the mean of pixel intensities

# Image folder path
preprocessed_image_path = '/content/drive/MyDrive/preprocessed_images'

# Loop through all images in the folder
for filename in os.listdir(preprocessed_image_path):
    if filename.lower().endswith(('.jpg', '.jpeg', '.png')):
        # Create the full image path using os.path.join
        image_path = os.path.join(preprocessed_image_path, filename)
        #Read the image using OpenCV
        processed_images = cv2.imread(image_path)
        #Calculate the mean intensity
        mean_intensity = np.mean(processed_images)
        print(f"Mean intensity of {filename}: {mean_intensity}")
```

The code snippet above calculates the mean of pixel intensities. It iterates through all 50 processed images using a 'for' loop. Then, the image path is created using 'os.path.join'. The images are read using 'OpenCV' and the mean intensity for each image is computed using 'np.mean'.
The output:



```
Mean intensity of image_010.jpg: 119.11780533333334
Mean intensity of image_012.jpg: 73.18021066666667
Mean intensity of image_022.jpg: 101.59783733333333
Mean intensity of image_014.jpg: 119.73982933333333
Mean intensity of image_015.jpg: 176.93532666666667
Mean intensity of image_017.jpg: 156.10334
Mean intensity of image_025.jpg: 169.83418266666666
Mean intensity of image_026.jpg: 190.29428266666667
Mean intensity of image_027.jpg: 67.244016
Mean intensity of image_028.jpg: 132.80057733333334
Mean intensity of image_029.jpg: 176.62787733333334
Mean intensity of image_030.jpg: 79.183792
Mean intensity of image_031.jpg: 110.33229866666667
Mean intensity of image_032.jpg: 113.28644
Mean intensity of image_033.jpg: 34.989749333333336
Mean intensity of image_034.jpg: 146.46877466666666
Mean intensity of image_036.jpg: 131.933632
Mean intensity of image_039.jpg: 68.528556
Mean intensity of image_049.jpg: 91.75629866666667
Mean intensity of image_048.jpg: 27.297629333333333
Mean intensity of image_047.jpg: 75.405924
Mean intensity of image_046.jpg: 154.22019466666666
Mean intensity of image_045.jpg: 46.019677333333334
Mean intensity of image_044.jpg: 109.32015466666667
Mean intensity of image_043.jpg: 90.80903733333334
Mean intensity of image_042.jpg: 73.30387866666666
Mean intensity of image_041.jpg: 170.59728933333332
Mean intensity of image_040.jpg: 60.736134666666665
Mean intensity of image_009.jpg: 129.31744
Mean intensity of image_013.jpg: 138.503084
Mean intensity of image_001.jpg: 82.751192
Mean intensity of image_002.jpg: 125.837756
Mean intensity of image_003.jpg: 205.47860266666666
Mean intensity of image_004.jpg: 180.68278666666666
Mean intensity of image_005.jpg: 83.31204666666666
Mean intensity of image_006.jpg: 111.15192933333333
Mean intensity of image_007.jpg: 158.145412
Mean intensity of image_008.jpg: 103.30816133333333
Mean intensity of image_011.jpg: 189.888308
Mean intensity of image_016.jpg: 114.967532
Mean intensity of image_018.jpg: 47.341432
Mean intensity of image_019.jpg: 48.71061866666667
Mean intensity of image_020.jpg: 70.71108933333333
Mean intensity of image_021.jpg: 103.06263333333334
Mean intensity of image_023.jpg: 106.36956533333333
Mean intensity of image_024.jpg: 86.20254666666666
Mean intensity of image_035.jpg: 140.12848933333333
Mean intensity of image_037.jpg: 74.37078266666667
Mean intensity of image_038.jpg: 85.98360266666667
Mean intensity of image_050.jpg: 115.213768
```

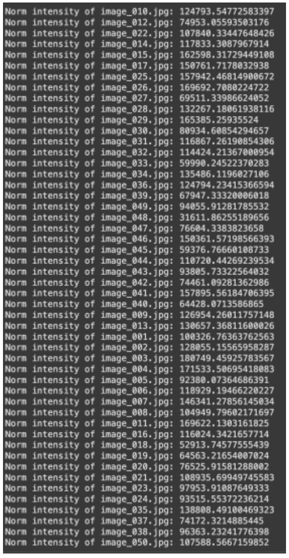
- ### The Norm of Pixel Intensities



```
# Calculate the norm of pixel intensities
# Image folder path
preprocessed_image_path = '/content/drive/MyDrive/preprocessed_images'

# Loop through all images in the folder
for filename in os.listdir(preprocessed_image_path):
    if filename.lower().endswith(('.jpg', '.jpeg', '.png')):
        # Create the full image path using os.path.join
        image_path = os.path.join(preprocessed_image_path, filename)
        #Read the image using OpenCV
        processed_images = cv2.imread(image_path)
        #Calculate the norm intensity
        norm_intensity = np.linalg.norm(processed_images)
        print(f"Norm intensity of {filename}: {norm_intensity}")
```

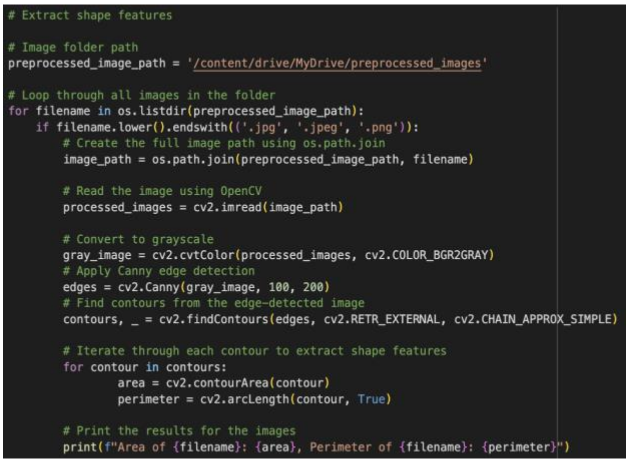
The code snippet above calculates the norm of pixel intensities. It iterates through all 50 processed images using a 'for' loop. Then, the image path is created using 'os.path.join'. The images are read using 'OpenCV' and the mean intensity for each image is computed using 'np.linalg.norm'.

The output:



```
Norm intensity of image_010.jpg: 124793.54772583397
Norm intensity of image_012.jpg: 74953.05593503176
Norm intensity of image_022.jpg: 107840.33447648426
Norm intensity of image_014.jpg: 117833.3087967914
Norm intensity of image_015.jpg: 162598.31729449108
Norm intensity of image_017.jpg: 150761.7178032938
Norm intensity of image_025.jpg: 157942.46814900672
Norm intensity of image_026.jpg: 169692.7080224722
Norm intensity of image_027.jpg: 69511.33986624052
Norm intensity of image_028.jpg: 132267.18061938116
Norm intensity of image_029.jpg: 165385.25935524
Norm intensity of image_030.jpg: 80934.60854294657
Norm intensity of image_031.jpg: 116867.26190854306
Norm intensity of image_032.jpg: 114424.21367000954
Norm intensity of image_033.jpg: 59990.24522370283
Norm intensity of image_034.jpg: 135486.1196027106
Norm intensity of image_036.jpg: 124794.23415366594
Norm intensity of image_039.jpg: 67947.33320006018
Norm intensity of image_049.jpg: 94055.91281785532
Norm intensity of image_048.jpg: 31611.86255189656
Norm intensity of image_047.jpg: 76604.3383823658
Norm intensity of image_046.jpg: 150361.57198566393
Norm intensity of image_045.jpg: 59376.76660108733
Norm intensity of image_044.jpg: 110720.44269239534
Norm intensity of image_043.jpg: 93805.73322564032
Norm intensity of image_042.jpg: 74461.09281362986
Norm intensity of image_041.jpg: 157895.56184706395
Norm intensity of image_040.jpg: 64428.0713586865
Norm intensity of image_009.jpg: 126954.26011757148
Norm intensity of image_013.jpg: 130657.36811600026
Norm intensity of image_001.jpg: 100326.76363762563
Norm intensity of image_002.jpg: 128055.15565958287
Norm intensity of image_003.jpg: 180749.45925783567
Norm intensity of image_004.jpg: 171533.50695418083
Norm intensity of image_005.jpg: 92380.07364686391
Norm intensity of image_006.jpg: 118929.19466220227
Norm intensity of image_007.jpg: 146341.27856145034
Norm intensity of image_008.jpg: 104949.79602171697
Norm intensity of image_011.jpg: 169622.1303161825
Norm intensity of image_016.jpg: 116024.3421657714
Norm intensity of image_018.jpg: 52913.74577555439
Norm intensity of image_019.jpg: 64563.21654007024
Norm intensity of image_020.jpg: 76525.91581288002
Norm intensity of image_021.jpg: 108935.69949745583
Norm intensity of image_023.jpg: 97953.91087649333
Norm intensity of image_024.jpg: 93515.55372236214
Norm intensity of image_035.jpg: 138808.49100469323
Norm intensity of image_037.jpg: 74172.3214885445
Norm intensity of image_038.jpg: 96363.23241776398
Norm intensity of image_050.jpg: 107588.5667159852
```

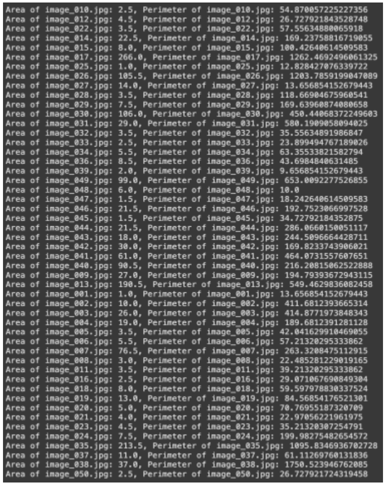
- ## The Extraction of Shape Features



```
# Extract shape features

# Image folder path
preprocessed_image_path = '/content/drive/MyDrive/preprocessed_images'

# Loop through all images in the folder
for filename in os.listdir(preprocessed_image_path):
    if filename.lower().endswith(('.jpg', '.jpeg', '.png')):
        # Create the full image path using os.path.join
        image_path = os.path.join(preprocessed_image_path, filename)

        # Read the image using OpenCV
        processed_images = cv2.imread(image_path)

        # Convert to grayscale
        gray_image = cv2.cvtColor(processed_images, cv2.COLOR_BGR2GRAY)
        # Apply Canny edge detection
        edges = cv2.Canny(gray_image, 100, 200)
        # Find contours from the edge-detected image
        contours, _ = cv2.findContours(edges, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

        # Iterate through each contour to extract shape features
        for contour in contours:
                area = cv2.contourArea(contour)
                perimeter = cv2.arcLength(contour, True)

        # Print the results for the images
        print(f"Area of {filename}: {area}, Perimeter of {filename}: {perimeter}")
```

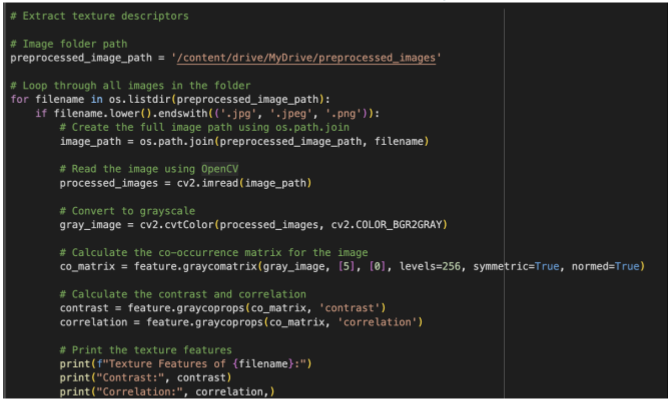
The code snippet above extracts the shape features such as ‘Area’, which is the total number of pixels enclosed by a contour and ‘Perimeter’, which is the length of the contour boundary. It iterates through all 50 processed images using a ‘for’ loop. Then, the image path is created using ‘os.path.join’. The images are read using ‘OpenCV’, and then the ‘cv2.cvtColor’ function converts the images to grayscale, and the edges are detected using ‘cv2.Canny’. The area of each contour is calculated using ‘cv2.contour’, and the perimeter is calculated using ‘cv2.arcLength’

The output:



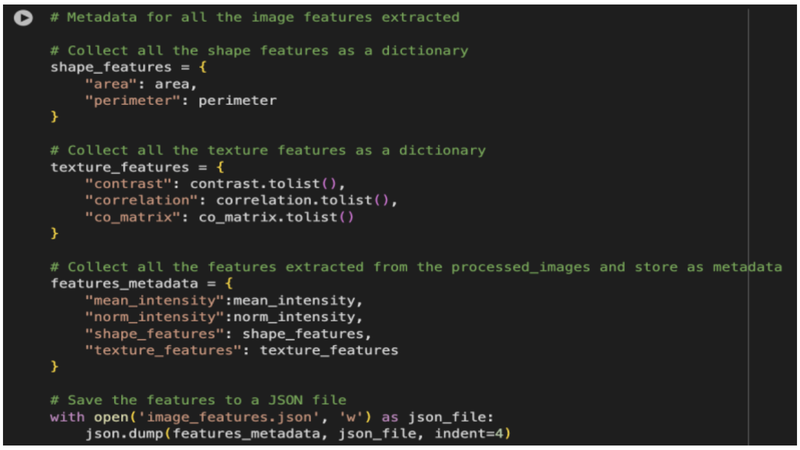
```
Area of image_010.jpg: 2.5, Perimeter of image_010.jpg: 54.870057225227356
Area of image_012.jpg: 4.5, Perimeter of image_012.jpg: 26.727921843528748
Area of image_022.jpg: 3.5, Perimeter of image_022.jpg: 57.55634880065918
Area of image_014.jpg: 22.5, Perimeter of image_014.jpg: 169.23758816719055
Area of image_015.jpg: 8.0, Perimeter of image_015.jpg: 100.42640614509583
Area of image_017.jpg: 266.0, Perimeter of image_017.jpg: 1262.4692496061325
Area of image_025.jpg: 1.0, Perimeter of image_025.jpg: 12.828427076339722
Area of image_026.jpg: 105.5, Perimeter of image_026.jpg: 1203.7859199047089
Area of image_027.jpg: 14.0, Perimeter of image_027.jpg: 13.656854152679443
Area of image_028.jpg: 3.5, Perimeter of image_028.jpg: 118.66904675960541
Area of image_029.jpg: 7.5, Perimeter of image_029.jpg: 169.63960874080658
Area of image_030.jpg: 106.0, Perimeter of image_030.jpg: 450.44068372249603
Area of image_031.jpg: 29.0, Perimeter of image_031.jpg: 580.1909058094025
Area of image_032.jpg: 3.5, Perimeter of image_032.jpg: 35.55634891986847
Area of image_033.jpg: 2.5, Perimeter of image_033.jpg: 23.899494767189026
Area of image_034.jpg: 5.5, Perimeter of image_034.jpg: 63.35533821582794
Area of image_036.jpg: 8.5, Perimeter of image_036.jpg: 43.6984840631485
Area of image_039.jpg: 2.0, Perimeter of image_039.jpg: 9.656854152679443
Area of image_049.jpg: 99.0, Perimeter of image_049.jpg: 653.0092277526855
Area of image_048.jpg: 6.0, Perimeter of image_048.jpg: 10.0
Area of image_047.jpg: 1.5, Perimeter of image_047.jpg: 18.242640614509583
Area of image_046.jpg: 21.5, Perimeter of image_046.jpg: 192.7523066997528
Area of image_045.jpg: 1.5, Perimeter of image_045.jpg: 34.72792184352875
Area of image_044.jpg: 21.5, Perimeter of image_044.jpg: 286.0660150051117
Area of image_043.jpg: 18.0, Perimeter of image_043.jpg: 244.5096664428711
Area of image_042.jpg: 30.0, Perimeter of image_042.jpg: 169.8233743906021
Area of image_041.jpg: 61.0, Perimeter of image_041.jpg: 464.0731557607651
Area of image_040.jpg: 90.5, Perimeter of image_040.jpg: 216.20815062522888
Area of image_009.jpg: 27.0, Perimeter of image_009.jpg: 194.79393672943115
Area of image_013.jpg: 190.5, Perimeter of image_013.jpg: 549.4629836082458
Area of image_001.jpg: 1.0, Perimeter of image_001.jpg: 13.656854152679443
Area of image_002.jpg: 10.0, Perimeter of image_002.jpg: 411.6812393665314
Area of image_003.jpg: 26.0, Perimeter of image_003.jpg: 414.8771973848343
Area of image_004.jpg: 19.0, Perimeter of image_004.jpg: 189.6812391281128
Area of image_005.jpg: 3.5, Perimeter of image_005.jpg: 42.041629910469055
Area of image_006.jpg: 5.5, Perimeter of image_006.jpg: 57.21320295333862
Area of image_007.jpg: 76.5, Perimeter of image_007.jpg: 263.3208475112915
Area of image_008.jpg: 3.0, Perimeter of image_008.jpg: 22.485281229019165
Area of image_011.jpg: 3.5, Perimeter of image_011.jpg: 39.21320295333862
Area of image_016.jpg: 2.5, Perimeter of image_016.jpg: 29.071067690849304
Area of image_018.jpg: 8.0, Perimeter of image_018.jpg: 59.597978830337524
Area of image_019.jpg: 13.0, Perimeter of image_019.jpg: 84.5685417652130l
Area of image_020.jpg: 5.0, Perimeter of image_020.jpg: 70.76955187320709
Area of image_021.jpg: 4.0, Perimeter of image_021.jpg: 22.97056221961975
Area of image_023.jpg: 4.5, Perimeter of image_023.jpg: 35.21320307254791
Area of image_024.jpg: 7.5, Perimeter of image_024.jpg: 199.98275482654572
Area of image_035.jpg: 213.5, Perimeter of image_035.jpg: 1095.8346936702728
Area of image_037.jpg: 11.0, Perimeter of image_037.jpg: 61.11269760131836
Area of image_038.jpg: 37.0, Perimeter of image_038.jpg: 1750.523946762085
Area of image_050.jpg: 2.5, Perimeter of image_050.jpg: 26.727921724319458
```

- ### The Extraction of Texture Descriptors



```
# Extract texture descriptors

# Image folder path
preprocessed_image_path = '/content/drive/MyDrive/preprocessed_images'

# Loop through all images in the folder
for filename in os.listdir(preprocessed_image_path):
    if filename.lower().endswith(('.jpg', '.jpeg', '.png')):
        # Create the full image path using os.path.join
        image_path = os.path.join(preprocessed_image_path, filename)

        # Read the image using OpenCV
        processed_images = cv2.imread(image_path)

        # Convert to grayscale
        gray_image = cv2.cvtColor(processed_images, cv2.COLOR_BGR2GRAY)

        # Calculate the co-occurrence matrix for the image
        co_matrix = feature.graycomatrix(gray_image, [5], [0], levels=256, symmetric=True, normed=True)

        # Calculate the contrast and correlation
        contrast = feature.graycoprops(co_matrix, 'contrast')
        correlation = feature.graycoprops(co_matrix, 'correlation')

        # Print the texture features
        print(f"Texture Features of {filename}:")
        print("Contrast:", contrast)
        print("Correlation:", correlation,)
```

The code snippet above extracts the texture descriptors such as ‘The Gray Level Co-occurrence Matrix’ (GLCM), ‘Contrast’ and the ‘Correlation’. It iterates through all 50 processed images using a ‘for’ loop. Then, the image path is created using ‘os.path.join’. The images are read using ‘OpenCV’, then the ‘cv2.cvtColor’ function converts the images to grayscale. The GLCM is calculated using ‘feature.graycomatrix’. The contrast and correlation of each picture are calculated using ‘feature.graycoprops’.

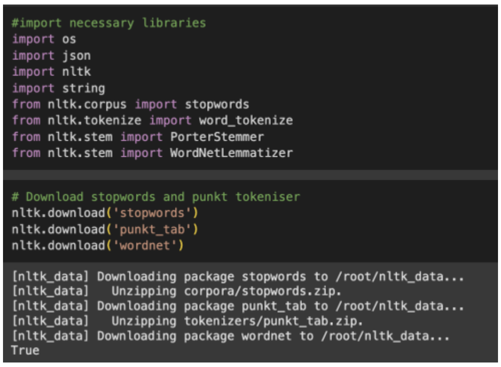


```
# Metadata for all the image features extracted

# Collect all the shape features as a dictionary
shape_features = {
    "area": area,
    "perimeter": perimeter
}

# Collect all the texture features as a dictionary
texture_features = {
    "contrast": contrast.tolist(),
    "correlation": correlation.tolist(),
    "co_matrix": co_matrix.tolist()
}

# Collect all the features extracted from the processed_images and store as metadata
features_metadata = {
    "mean_intensity":mean_intensity,
    "norm_intensity":norm_intensity,
    "shape_features": shape_features,
    "texture_features": texture_features
}

# Save the features to a JSON file
with open('image_features.json', 'w') as json_file:
    json.dump(features_metadata, json_file, indent=4)
```

All the extracted image features above are stored in their respective dictionaries, collected as metadata and saved to a JSON file named ‘image_features.json’.

2. *Databases for Sentiment Analysis Models*

1. Data Preprocessing

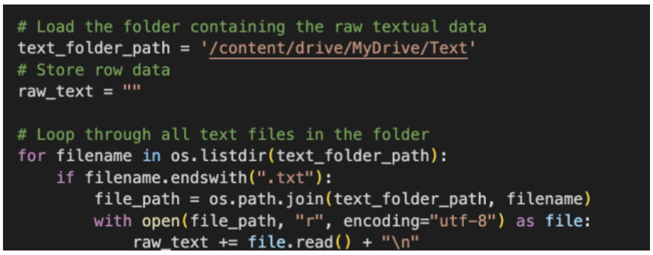


```
#import necessary libraries
import os
import json
import nltk
import string
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import PorterStemmer
from nltk.stem import WordNetLemmatizer
```

```
# Download stopwords and punkt tokeniser
nltk.download('stopwords')
nltk.download('punkt_tab')
nltk.download('wordnet')
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
[nltk_data] Downloading package punkt_tab to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt_tab.zip.
[nltk_data] Downloading package wordnet to /root/nltk_data...
True
```

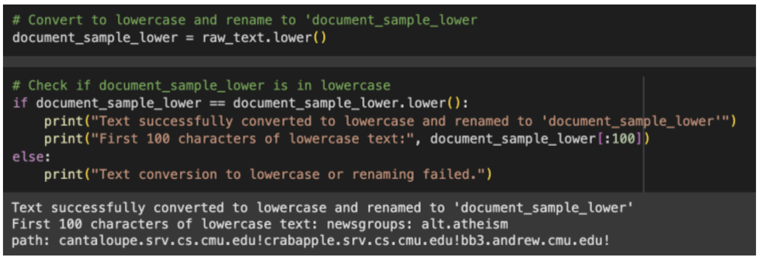
Necessary Python libraries and modules were imported on Google Colab to start the preprocessing of textual data.



```
# Load the folder containing the raw textual data
text_folder_path = '/content/drive/MyDrive/Text'
# Store row data
raw_text = ""

# Loop through all text files in the folder
for filename in os.listdir(text_folder_path):
    if filename.endswith(".txt"):
        file_path = os.path.join(text_folder_path, filename)
        with open(file_path, "r", encoding="utf-8") as file:
            raw_text += file.read() + "\n"
```

The file named ‘Text’, containing raw, unprocessed textual data is specified, and the path for the folder is mentioned. The file was saved in Google Drive connected to Colab. It contains 50 text documents in ‘.txt’ format. The variable ‘raw_text’ stores the combined content of all 50 text files. The ‘for’ loop iterates through all the unprocessed texts that end with ‘.txt’ and ‘os.listdir’ returns files and directories in the specified path. This code snippet combines all the text files in the ‘text’ folder into one text data for further processing.

- ### Text Normalisation (Lowercasing)



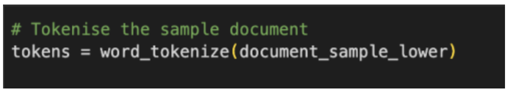
```
# Convert to lowercase and rename to 'document_sample_lower
document_sample_lower = raw_text.lower()
```

```
# Check if document_sample_lower is in lowercase
if document_sample_lower == document_sample_lower.lower():
    print("Text successfully converted to lowercase and renamed to 'document_sample_lower'")
    print("First 100 characters of lowercase text:", document_sample_lower[:100])
else:
    print("Text conversion to lowercase or renaming failed.")
```

```
Text successfully converted to lowercase and renamed to 'document_sample_lower'
First 100 characters of lowercase text: newsgroups: alt.atheism
path: cantaloupe.srv.cs.cmu.edu!crabapple.srv.cs.cmu.edu!bb3.andrew.cmu.edu!
```

This code snippet iterates through each document file using ‘if’ and ‘else’, and converts all the text documents to lowercase and then saves them to a folder named ‘document_sample_lower’. The output of the first 100 characters, shows that it has been successfully converted to lowercase.

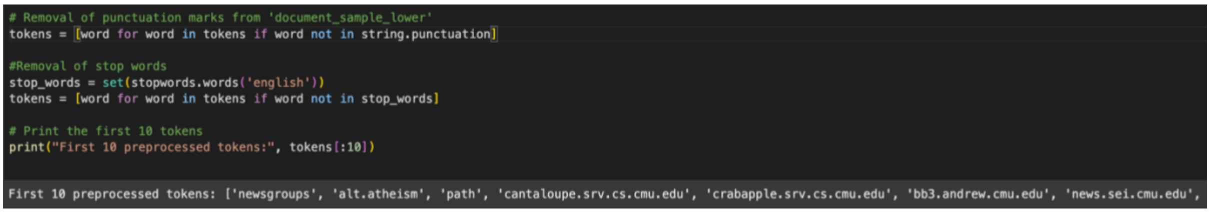
- ### Tokenisation



```
# Tokenise the sample document
tokens = word_tokenize(document_sample_lower)
```

The ‘word_tokenize’ function from NLTK is used to split the texts in ‘document_sample_lower’ into individual words or tokens.

- ### Removal of Punctuation and Stop Words

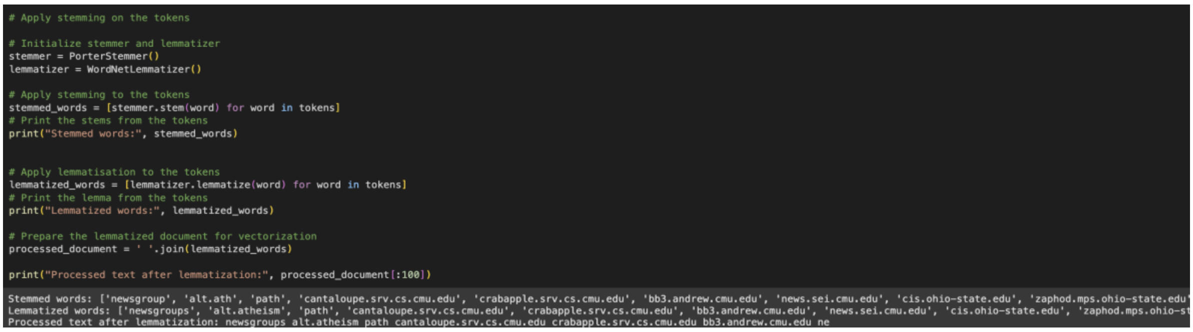


```
# Removal of punctuation marks from 'document_sample_lower'
tokens = [word for word in tokens if word not in string.punctuation]

#Removal of stop words
stop_words = set(stopwords.words('english'))
tokens = [word for word in tokens if word not in stop_words]

# Print the first 10 tokens
print("First 10 preprocessed tokens:", tokens[:10])
```

```
First 10 preprocessed tokens: ['newsgroups', 'alt.atheism', 'path', 'cantaloupe.srv.cs.cmu.edu', 'crabapple.srv.cs.cmu.edu', 'bb3.andrew.cmu.edu', 'news.sei.cmu.edu',
```

This first code filters tokens to eliminate unnecessary punctuation marks using the ‘string.punctuation’ function. The second code uses ‘stopwords.words’ from NLTK to remove common words, such as ‘the’ and ‘and’. The output shows that the codes have successfully removed punctuation marks and stop words. Both these code lines help focus on the meaningful content of the texts by eliminating unnecessary characters.

- ### Application of Stemming and Lemmatisation on the Tokens



```
# Apply stemming on the tokens

# Initialize stemmer and lemmatizer
stemmer = PorterStemmer()
lemmatizer = WordNetLemmatizer()

# Apply stemming to the tokens
stemmed_words = [stemmer.stem(word) for word in tokens]
# Print the stems from the tokens
print("Stemmed words:", stemmed_words)


# Apply lemmatisation to the tokens
lemmatized_words = [lemmatizer.lemmatize(word) for word in tokens]
# Print the lemma from the tokens
print("Lemmatized words:", lemmatized_words)

# Prepare the lemmatized document for vectorization
processed_document = ' '.join(lemmatized_words)

print("Processed text after lemmatization:", processed_document[:100])
```

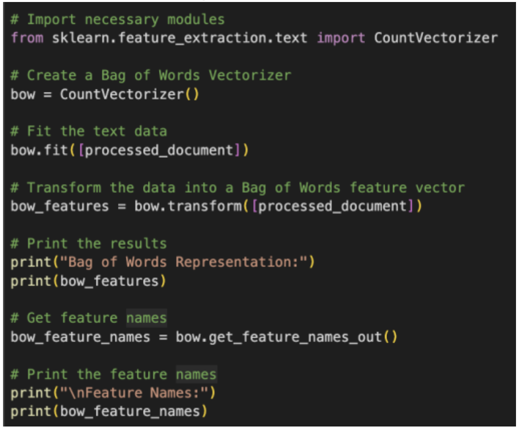
```
Stemmed words: ['newsgroup', 'alt.ath', 'path', 'cantaloupe.srv.cs.cmu.edu', 'crabapple.srv.cs.cmu.edu', 'bb3.andrew.cmu.edu', 'news.sei.cmu.edu', 'cis.ohio-state.edu', 'zaphod.mps.ohio-state.edu'
Lemmatized words: ['newsgroups', 'alt.atheism', 'path', 'cantaloupe.srv.cs.cmu.edu', 'crabapple.srv.cs.cmu.edu', 'bb3.andrew.cmu.edu', 'news.sei.cmu.edu', 'cis.ohio-state.edu', 'zaphod.mps.ohio-st
Processed text after lemmatization: newsgroups alt.atheism path cantaloupe.srv.cs.cmu.edu crabapple.srv.cs.cmu.edu bb3.andrew.cmu.edu ne
```

The code snippets above apply stemming and lemmatisation on the tokens using the functions ‘PorterStemmer()” and ‘WordNetLemmatizer()’. It helps reduce words to their root forms (e.g. Cancelling to Cancel). The lemmatized document is saved as ‘processed_document’ for text vectorisation.

2. Text Vectorisation

- ### Bag of Words

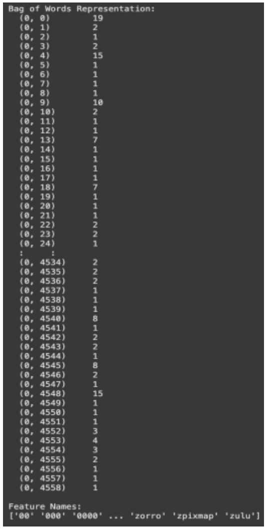
This method converts text documents into numerical feature vectors, which can be used for various machine-learning tasks. It focuses on the frequency of each word occurrence. (Hackers Realm, 2023)



```
# Import necessary modules
from sklearn.feature_extraction.text import CountVectorizer

# Create a Bag of Words Vectorizer
bow = CountVectorizer()

# Fit the text data
bow.fit([processed_document])

# Transform the data into a Bag of Words feature vector
bow_features = bow.transform([processed_document])

# Print the results
print("Bag of Words Representation:")
print(bow_features)

# Get feature names
bow_feature_names = bow.get_feature_names_out()

# Print the feature names
print("\nFeature Names:")
print(bow_feature_names)
```

This code snippet converts the texts into a sparse matrix of word counts using ‘CountVectorizer()’ and prints the result using a variable named ‘bow_feature_names’.

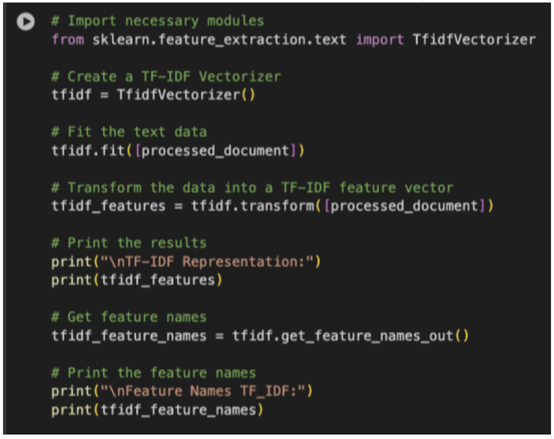
The output:



```
Bag of Words Representation:
  (0, 0)	19
  (0, 1)	2
  (0, 2)	1
  (0, 3)	2
  (0, 4)	15
  (0, 5)	1
  (0, 6)	1
  (0, 7)	1
  (0, 8)	1
  (0, 9)	10
  (0, 10)	2
  (0, 11)	1
  (0, 12)	1
  (0, 13)	7
  (0, 14)	1
  (0, 15)	1
  (0, 16)	1
  (0, 17)	1
  (0, 18)	7
  (0, 19)	1
  (0, 20)	1
  (0, 21)	1
  (0, 22)	2
  (0, 23)	2
  (0, 24)	1
  :	:
  (0, 4534)	2
  (0, 4535)	2
  (0, 4536)	2
  (0, 4537)	1
  (0, 4538)	1
  (0, 4539)	1
  (0, 4540)	8
  (0, 4541)	1
  (0, 4542)	2
  (0, 4543)	2
  (0, 4544)	1
  (0, 4545)	8
  (0, 4546)	2
  (0, 4547)	1
  (0, 4548)	15
  (0, 4549)	1
  (0, 4550)	1
  (0, 4551)	1
  (0, 4552)	3
  (0, 4553)	4
  (0, 4554)	3
  (0, 4555)	2
  (0, 4556)	1
  (0, 4557)	1
  (0, 4558)	1

Feature Names:
['00' '000' '0000' ... 'zorro' 'zpixmap' 'zulu']
```

- **Term Frequency-Inverse Document Frequency (TF-IDF)**

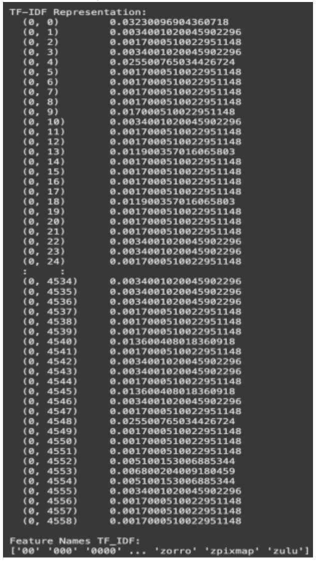
This method evaluates the importance of a word in a document relative to a collection of documents. It quantifies the significance or relevance of string representations (words, phrases, lemmas, etc.) in a document amongst a collection of documents. (Hackers Realm, 2023)



```
# Import necessary modules
from sklearn.feature_extraction.text import TfidfVectorizer

# Create a TF-IDF Vectorizer
tfidf = TfidfVectorizer()

# Fit the text data
tfidf.fit([processed_document])

# Transform the data into a TF-IDF feature vector
tfidf_features = tfidf.transform([processed_document])

# Print the results
print("\nTF-IDF Representation:")
print(tfidf_features)

# Get feature names
tfidf_feature_names = tfidf.get_feature_names_out()

# Print the feature names
print("\nFeature Names TF_IDF:")
print(tfidf_feature_names)
```

This code snippet highlights the representation of important texts in the document using 'TfidfVectorizer()' and prints the output using a variable named 'tfidf_feature_names'.

The output:

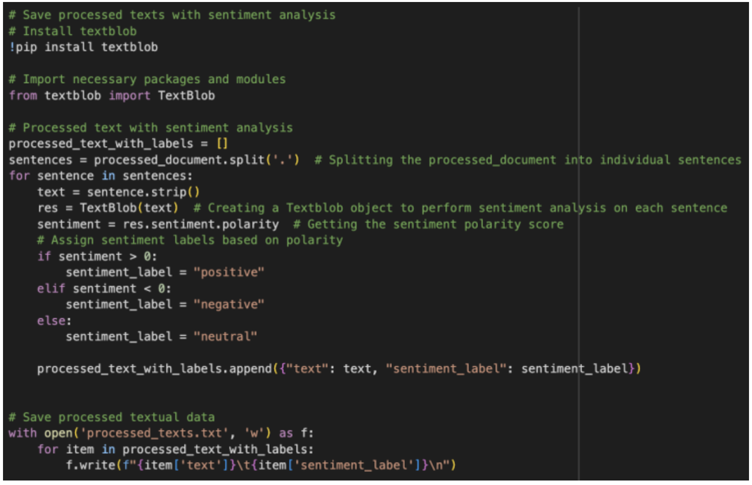


```
TF-IDF Representation:
  (0, 0)	0.03230096904360718
  (0, 1)	0.003400102004590229
  (0, 2)	0.0017000510022951148
  (0, 3)	0.003400102004590229
  (0, 4)	0.025500765034426724
  (0, 5)	0.0017000510022951148
  (0, 6)	0.0017000510022951148
  (0, 7)	0.0017000510022951148
  (0, 8)	0.0017000510022951148
  (0, 9)	0.017000510022951148
  (0, 10)	0.003400102004590229
  (0, 11)	0.0017000510022951148
  (0, 12)	0.0017000510022951148
  (0, 13)	0.011900357016065803
  (0, 14)	0.0017000510022951148
  (0, 15)	0.0017000510022951148
  (0, 16)	0.0017000510022951148
  (0, 17)	0.0017000510022951148
  (0, 18)	0.011900357016065803
  (0, 19)	0.0017000510022951148
  (0, 20)	0.0017000510022951148
  (0, 21)	0.0017000510022951148
  (0, 22)	0.003400102004590229
  (0, 23)	0.003400102004590229
  (0, 24)	0.0017000510022951148
  :	:
  (0, 4534)	0.003400102004590229
  (0, 4535)	0.003400102004590229
  (0, 4536)	0.003400102004590229
  (0, 4537)	0.0017000510022951148
  (0, 4538)	0.0017000510022951148
  (0, 4539)	0.0017000510022951148
  (0, 4540)	0.013600408018360918
  (0, 4541)	0.0017000510022951148
  (0, 4542)	0.003400102004590229
  (0, 4543)	0.003400102004590229
  (0, 4544)	0.0017000510022951148
  (0, 4545)	0.013600408018360918
  (0, 4546)	0.003400102004590229
  (0, 4547)	0.0017000510022951148
  (0, 4548)	0.025500765034426724
  (0, 4549)	0.0017000510022951148
  (0, 4550)	0.0017000510022951148
  (0, 4551)	0.0017000510022951148
  (0, 4552)	0.005100153006885344
  (0, 4553)	0.006800204009180459
  (0, 4554)	0.005100153006885344
  (0, 4555)	0.003400102004590229
  (0, 4556)	0.0017000510022951148
  (0, 4557)	0.0017000510022951148
  (0, 4558)	0.0017000510022951148

Feature Names TF_IDF:
['00' '000' '0000' ... 'zorro' 'zpixmap' 'zulu']
```

3. Metadata and Labelling

• Labelling

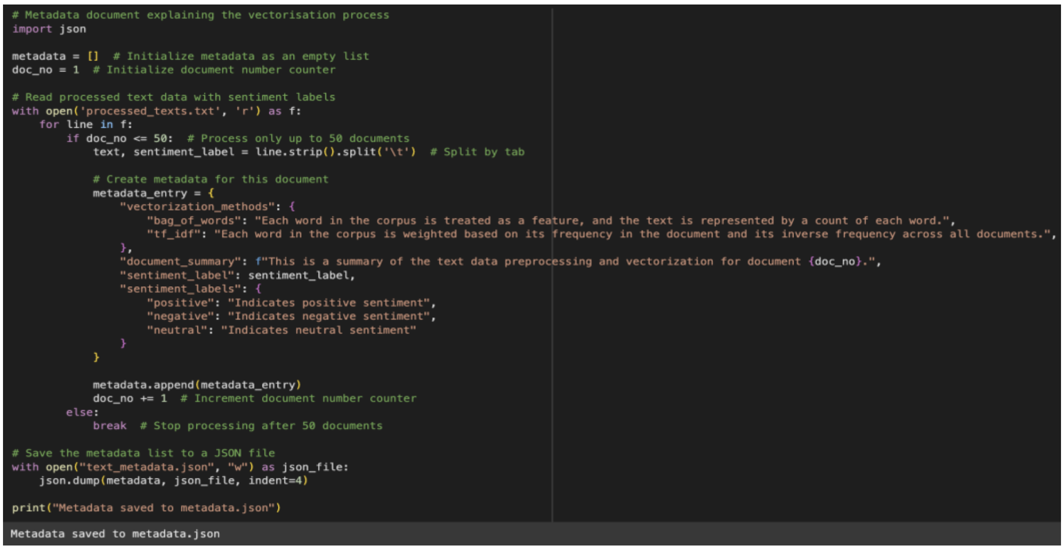
The text was automatically labelled with sentiment labels (e.g., positive, negative, neutral) using ‘TextBlob’. TextBlob is a Python library used for Natural Language Processing (NLP). It relies on NLTK (Natural Language Toolkit). It uses polarity and subjectivity for sentimental labelling. (Mohit, 2021)



```python
# Save processed texts with sentiment analysis
# Install textblob
!pip install textblob

# Import necessary packages and modules
from textblob import TextBlob

# Processed text with sentiment analysis
processed_text_with_labels = []
sentences = processed_document.split('.')  # Splitting the processed_document into individual sentences
for sentence in sentences:
    text = sentence.strip()
    res = TextBlob(text)  # Creating a Textblob object to perform sentiment analysis on each sentence
    sentiment = res.sentiment.polarity  # Getting the sentiment polarity score
    # Assign sentiment labels based on polarity
    if sentiment > 0:
        sentiment_label = "positive"
    elif sentiment < 0:
        sentiment_label = "negative"
    else:
        sentiment_label = "neutral"

    processed_text_with_labels.append({"text": text, "sentiment_label": sentiment_label})


# Save processed textual data
with open('processed_texts.txt', 'w') as f:
    for item in processed_text_with_labels:
        f.write(f"{item['text']}\t{item['sentiment_label']}\n")
```

The processed document is split into individual sentences. Then a for loop is used to iterate through each sentence to create a TextBlob object to perform sentimental analysis on each sentence. Then, the ‘res.sentiment.polarity’ function is used to assign sentiment labels based on its polarity score. For example, if the sentiment polarity score is below 0, the sentiment label is ‘negative’. The processed text file with the sentimental labels generated by TextBlob is saved as ‘processed_texts.txt’ at the end.

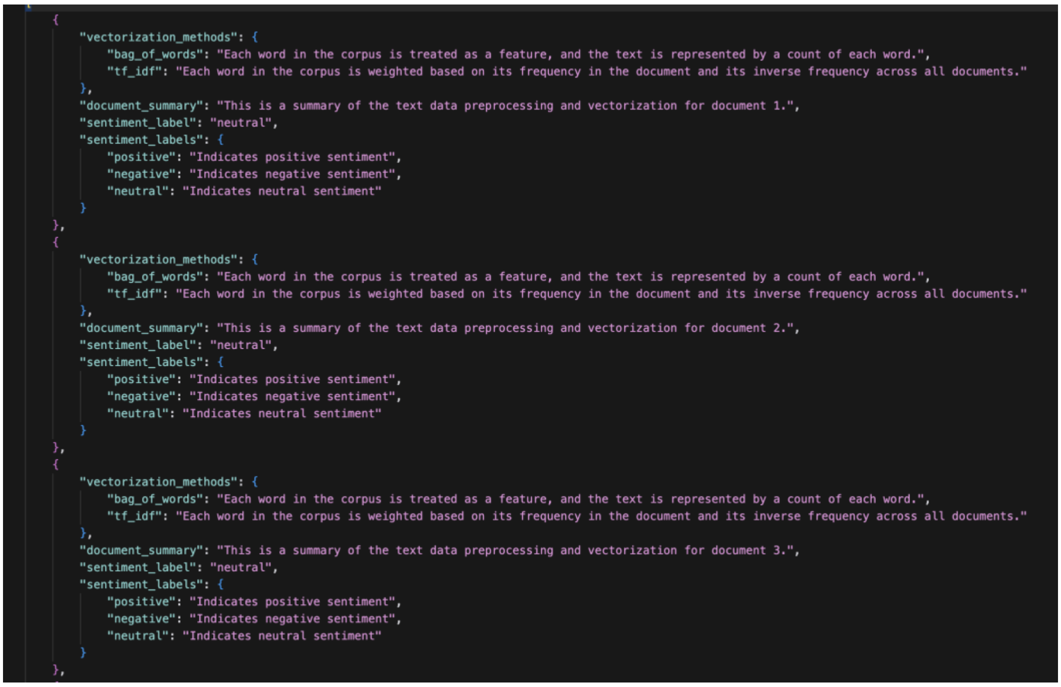
• Metadata



```python
# Metadata document explaining the vectorisation process
import json

metadata = []  # Initialize metadata as an empty list
doc_no = 1  # Initialize document number counter

# Read processed text data with sentiment labels
with open('processed_texts.txt', 'r') as f:
    for line in f:
        if doc_no <= 50:  # Process only up to 50 documents
            text, sentiment_label = line.strip().split('\t')  # Split by tab

            # Create metadata for this document
            metadata_entry = {
                "vectorization_methods": {
                    "bag_of_words": "Each word in the corpus is treated as a feature, and the text is represented by a count of each word.",
                    "tf_idf": "Each word in the corpus is weighted based on its frequency in the document and its inverse frequency across all documents.",
                },
                "document_summary": f"This is a summary of the text data preprocessing and vectorization for document {doc_no}.",
                "sentiment_label": sentiment_label,
                "sentiment_labels": {
                    "positive": "Indicates positive sentiment",
                    "negative": "Indicates negative sentiment",
                    "neutral": "Indicates neutral sentiment"
                }
            }

            metadata.append(metadata_entry)
            doc_no += 1  # Increment document number counter
        else:
            break  # Stop processing after 50 documents

# Save the metadata list to a JSON file
with open("text_metadata.json", "w") as json_file:
    json.dump(metadata, json_file, indent=4)

print("Metadata saved to metadata.json")
```

```
Metadata saved to metadata.json
```

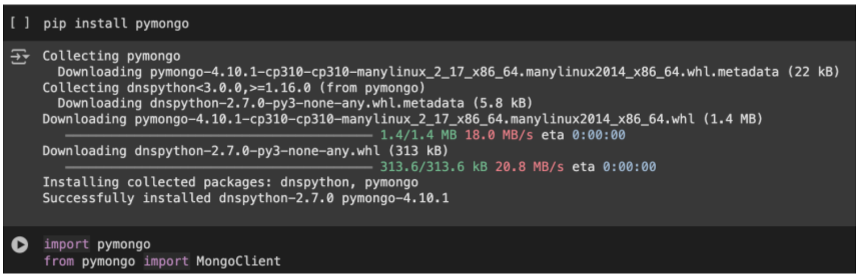
This code snippet generates a metadata file in JSON format that explains the vectorisation process used for sentiment analysis. It includes information about the sentiment labels assigned to each document using TextBlob. It iterates through the 50 text documents with the sentiment labels and, creates a dictionary named ‘metadata_entry’ to store the metadata for the current document. This dictionary contains information about the vectorisation methods used (Bag of Words and TF-IDF), a summary of the document, the sentiment label assigned to the document, and an explanation of the sentiment labels. The metadata file is saved as ‘text_metadata.json’ at the end.



```
    {
        "vectorization_methods": {
            "bag_of_words": "Each word in the corpus is treated as a feature, and the text is represented by a count of each word.",
            "tf_idf": "Each word in the corpus is weighted based on its frequency in the document and its inverse frequency across all documents."
        },
        "document_summary": "This is a summary of the text data preprocessing and vectorization for document 1.",
        "sentiment_label": "neutral",
        "sentiment_labels": {
            "positive": "Indicates positive sentiment",
            "negative": "Indicates negative sentiment",
            "neutral": "Indicates neutral sentiment"
        }
    },
    {
        "vectorization_methods": {
            "bag_of_words": "Each word in the corpus is treated as a feature, and the text is represented by a count of each word.",
            "tf_idf": "Each word in the corpus is weighted based on its frequency in the document and its inverse frequency across all documents."
        },
        "document_summary": "This is a summary of the text data preprocessing and vectorization for document 2.",
        "sentiment_label": "neutral",
        "sentiment_labels": {
            "positive": "Indicates positive sentiment",
            "negative": "Indicates negative sentiment",
            "neutral": "Indicates neutral sentiment"
        }
    },
    {
        "vectorization_methods": {
            "bag_of_words": "Each word in the corpus is treated as a feature, and the text is represented by a count of each word.",
            "tf_idf": "Each word in the corpus is weighted based on its frequency in the document and its inverse frequency across all documents."
        },
        "document_summary": "This is a summary of the text data preprocessing and vectorization for document 3.",
        "sentiment_label": "neutral",
        "sentiment_labels": {
            "positive": "Indicates positive sentiment",
            "negative": "Indicates negative sentiment",
            "neutral": "Indicates neutral sentiment"
        }
    },
```

The metadata for the first three processed text files in the JSON file is shown in the screenshot above.

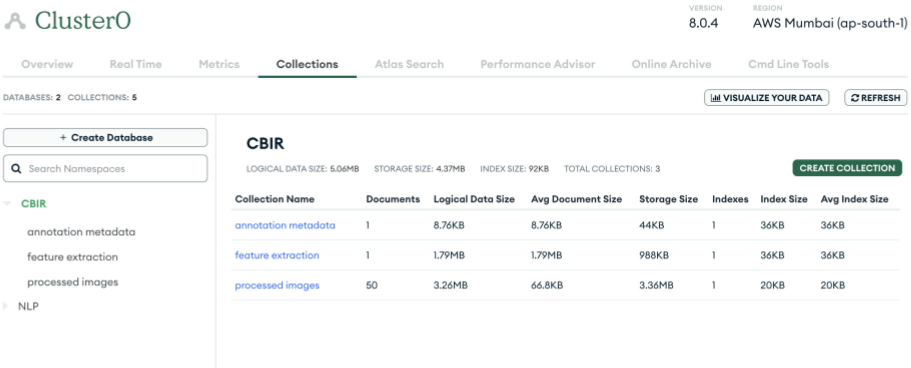
4. Storage and Retrieval



```
[ ] pip install pymongo

    Collecting pymongo
      Downloading pymongo-4.10.1-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (22 kB)
    Collecting dnspython<3.0.0,>=1.16.0 (from pymongo)
      Downloading dnspython-2.7.0-py3-none-any.whl.metadata (5.8 kB)
    Downloading pymongo-4.10.1-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (1.4 MB)
       ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.4/1.4 MB 18.0 MB/s eta 0:00:00
    Downloading dnspython-2.7.0-py3-none-any.whl (313 kB)
       ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 313.6/313.6 kB 20.8 MB/s eta 0:00:00
    Installing collected packages: dnspython, pymongo
    Successfully installed dnspython-2.7.0 pymongo-4.10.1

    import pymongo
    from pymongo import MongoClient
```

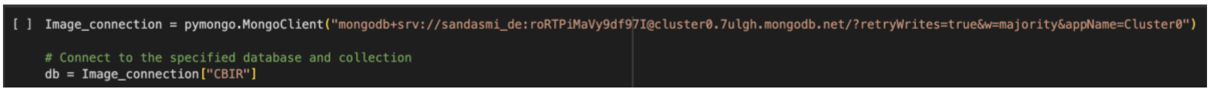
MongoDB was used as NoSQL database for CBIR and NLP since it's more convenient and flexible than other databases, helps handle large unstructured data, and combines data from multiple documents into a single query. MongoDB also has a better sharding ability as it can break up a collection into subsets of data and store them across multiple shards. (GUVI Geek, 2020)

1. NoSQL Database - CBIR



Collection Name	Documents	Logical Data Size	Avg Document Size	Storage Size	Indexes	Index Size	Avg Index Size
annotation metadata	1	8.76KB	8.76KB	44KB	1	36KB	36KB
feature extraction	1	1.79MB	1.79MB	988KB	1	36KB	36KB
processed images	50	3.26MB	66.8KB	3.36MB	1	20KB	20KB

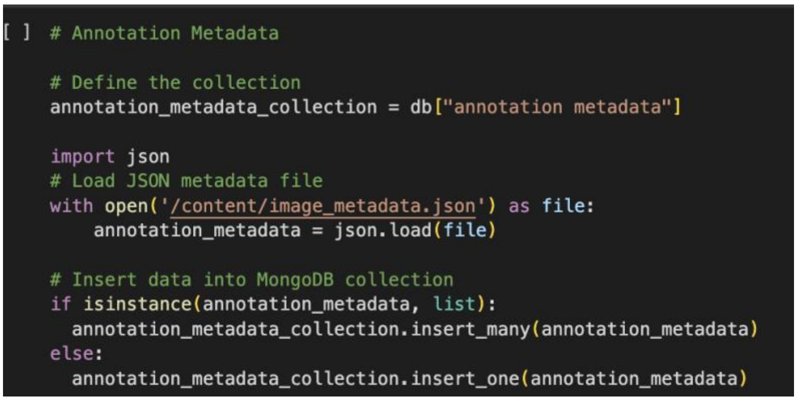
'Cluster0' is the cluster created to enter a database named 'CBIR' and its collections: 'annotation metadata', 'feature extractions', and 'processed images'.



```
[ ] Image_connection = pymongo.MongoClient("mongodb+srv://sandasmi_de:roRTPiMaVy9df97I@cluster0.7ulgh.mongodb.net/?retryWrites=true&w=majority&appName=Cluster0")

    # Connect to the specified database and collection
    db = Image_connection["CBIR"]
```

The connection string obtained from the cluster is used to connect MongoDB Atlas to Google Colab and create the image connection using 'pymongo'. The string contains the username which is **sandasmi_de,** and the password which is **roRTPiMaVy9df97I**

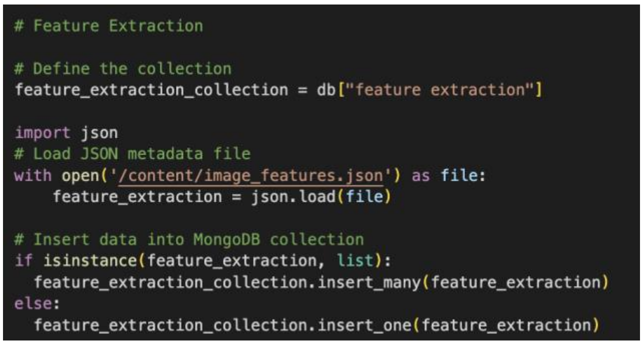
- ### Annotation Metadata



```
[ ] # Annotation Metadata

    # Define the collection
    annotation_metadata_collection = db["annotation metadata"]

    import json
    # Load JSON metadata file
    with open('/content/image_metadata.json') as file:
        annotation_metadata = json.load(file)

    # Insert data into MongoDB collection
    if isinstance(annotation_metadata, list):
      annotation_metadata_collection.insert_many(annotation_metadata)
    else:
      annotation_metadata_collection.insert_one(annotation_metadata)
```

This code snippet loads the JSON file named 'image_metadata.json' and inserts it into the 'annotation metadata' collection in the MongoDB database.

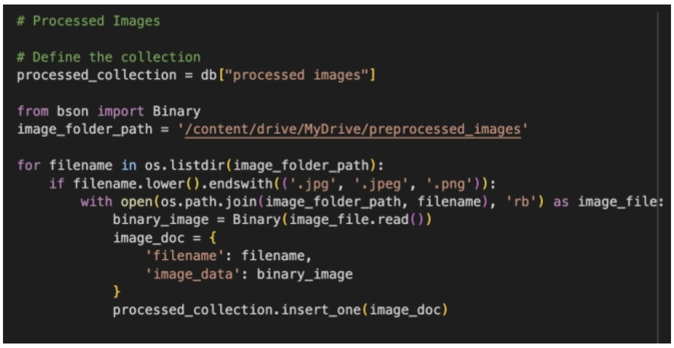
- **Feature Extraction**



```
# Feature Extraction

# Define the collection
feature_extraction_collection = db["feature extraction"]

import json
# Load JSON metadata file
with open('/content/image_features.json') as file:
    feature_extraction = json.load(file)

# Insert data into MongoDB collection
if isinstance(feature_extraction, list):
  feature_extraction_collection.insert_many(feature_extraction)
else:
  feature_extraction_collection.insert_one(feature_extraction)
```

This code snippet loads the JSON file named ‘image_features.json’ and inserts it into the ‘feature extraction’ collection in the MongoDB database, storing the extracted features for each image.

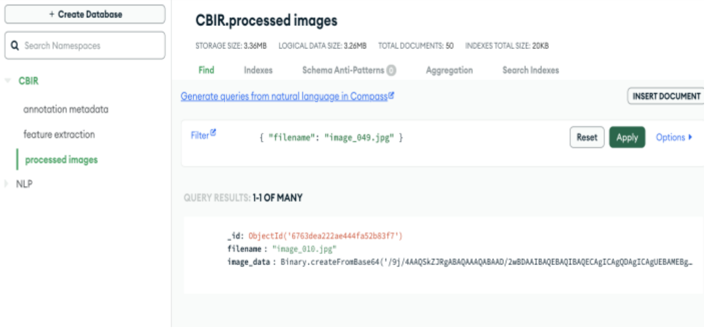
- **Processed Images**



```
# Processed Images

# Define the collection
processed_collection = db["processed images"]

from bson import Binary
image_folder_path = '/content/drive/MyDrive/preprocessed_images'

for filename in os.listdir(image_folder_path):
    if filename.lower().endswith(('.jpg', '.jpeg', '.png')):
        with open(os.path.join(image_folder_path, filename), 'rb') as image_file:
            binary_image = Binary(image_file.read())
            image_doc = {
                'filename': filename,
                'image_data': binary_image
            }
            processed_collection.insert_one(image_doc)
```

This code snippet loads the JSON file and inserts it into the ‘processed images’ collection in the MongoDB database. It stores the binary data of each preprocessed image using ‘bson’.

Queries

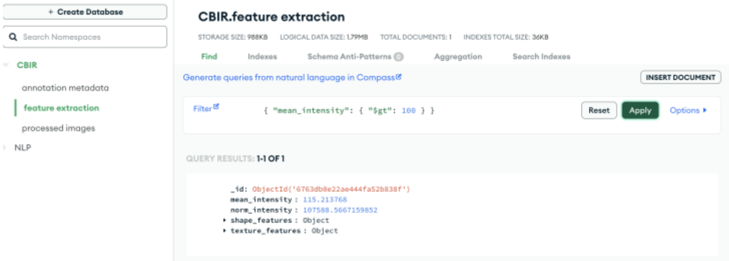


```
{ "filename": "image_049.jpg" }
```

QUERY RESULTS: 1-1 OF MANY

```
_id: ObjectId('6763dea222ae444fa52b83f7')
filename : "image_010.jpg"
image_data : Binary.createFromBase64('/9j/4AAQSkZJRgABAQAAAQABAAD/2wBDAAIBAQEBAQIBAQECAgICAgQDAgICAgUEBAMEBg…
```

- Checks for a specific document using its file name.



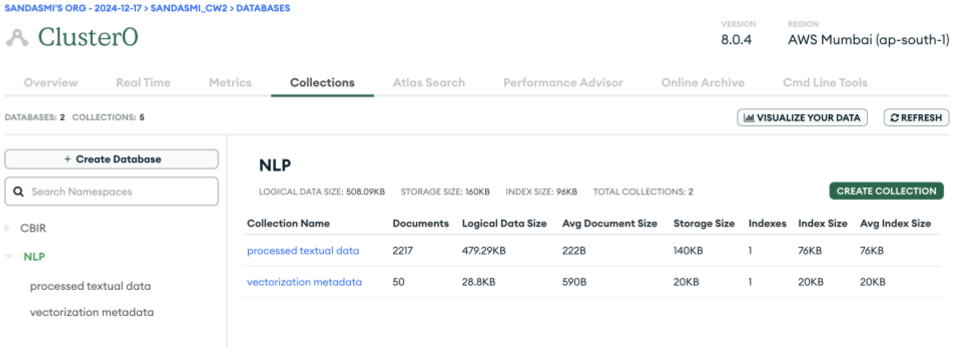
```
{ "mean_intensity": { "$gt": 100 } }
```

QUERY RESULTS: 1-1 OF 1

```
_id: ObjectId('6763db8e22ae444fa52b838f')
mean_intensity : 115.213768
norm_intensity : 107588.5667159852
▸ shape_features : Object
▸ texture_features : Object
```

- Checks for documents with mean intensity greater than 100.

2. NoSQL Database - Sentiment Analysis Models (NLP)



SANDASMI'S ORG - 2024-12-17 > SANDASMI_CW2 > DATABASES

Cluster0

VERSION 8.0.4 REGION AWS Mumbai (ap-south-1)

DATABASES: 2 COLLECTIONS: 5

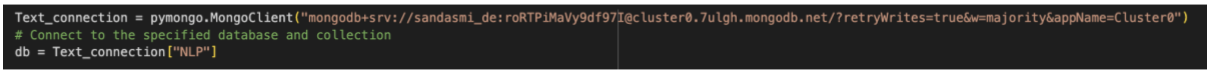
CBIR
NLP
processed textual data
vectorization metadata

NLP

LOGICAL DATA SIZE: 508.09KB STORAGE SIZE: 160KB INDEX SIZE: 96KB TOTAL COLLECTIONS: 2

Collection Name	Documents	Logical Data Size	Avg Document Size	Storage Size	Indexes	Index Size	Avg Index Size
processed textual data	2217	479.29KB	222B	140KB	1	76KB	76KB
vectorization metadata	50	28.8KB	590B	20KB	1	20KB	20KB

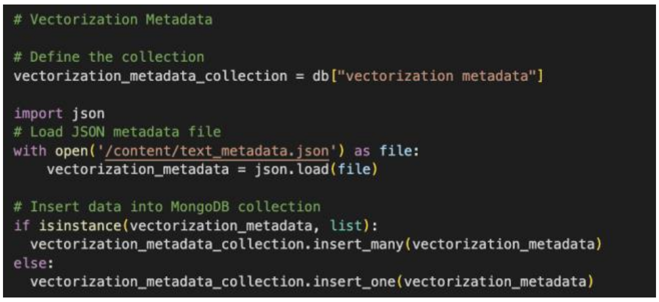
‘Cluster0’ is the cluster created to enter a database named ‘NLP’ and its collections: ‘processed textual data’ and ‘vectorization metadata’.



```
Text_connection = pymongo.MongoClient("mongodb+srv://sandasmi_de:roRTPiMaVy9df97I@cluster0.7ulgh.mongodb.net/?retryWrites=true&w=majority&appName=Cluster0")
# Connect to the specified database and collection
db = Text_connection["NLP"]
```

The connection string obtained from the cluster is used to connect MongoDB Atlas to Google Colab and create the text connection using ‘pymongo’. The string contains the username which is **sandasmi_de,** and the password which is **roRTPiMaVy9df97I**

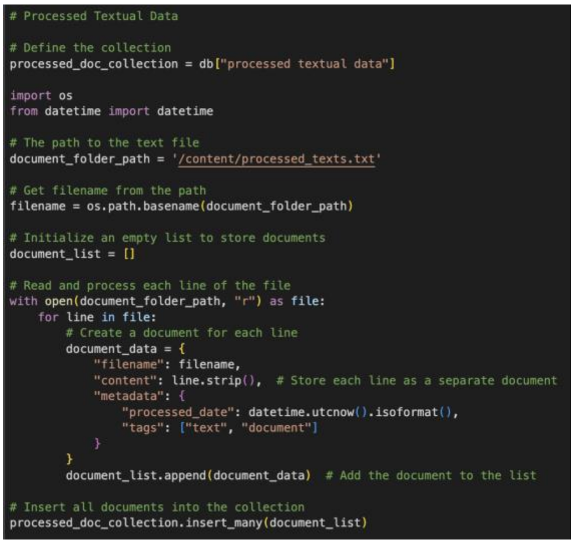
- ### Vectorization Metadata



```
# Vectorization Metadata

# Define the collection
vectorization_metadata_collection = db["vectorization metadata"]

import json
# Load JSON metadata file
with open('/content/text_metadata.json') as file:
    vectorization_metadata = json.load(file)

# Insert data into MongoDB collection
if isinstance(vectorization_metadata, list):
  vectorization_metadata_collection.insert_many(vectorization_metadata)
else:
  vectorization_metadata_collection.insert_one(vectorization_metadata)
```

This code snippet loads the JSON file named ‘text_metadata.json’ and inserts it into the ‘vectorisation metadata’ collection in the MongoDB database.

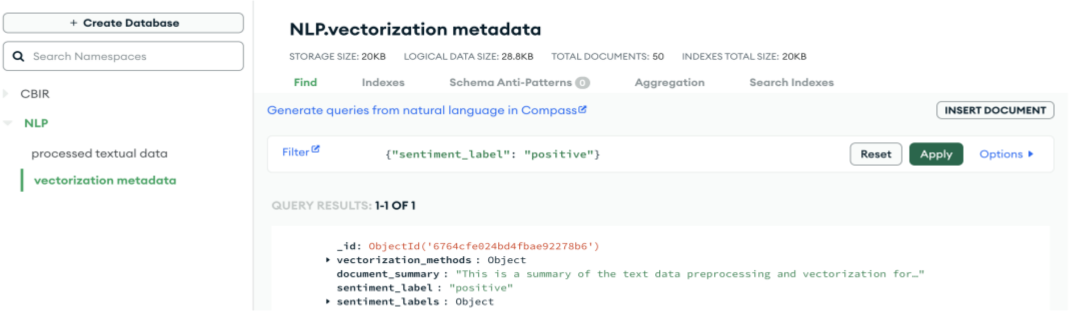
- ### Processed Textual Data



```
# Processed Textual Data

# Define the collection
processed_doc_collection = db["processed textual data"]

import os
from datetime import datetime

# The path to the text file
document_folder_path = '/content/processed_texts.txt'

# Get filename from the path
filename = os.path.basename(document_folder_path)

# Initialize an empty list to store documents
document_list = []

# Read and process each line of the file
with open(document_folder_path, "r") as file:
    for line in file:
        # Create a document for each line
        document_data = {
            "filename": filename,
            "content": line.strip(),  # Store each line as a separate document
            "metadata": {
                "processed_date": datetime.utcnow().isoformat(),
                "tags": ["text", "document"]
            }
        }
        document_list.append(document_data)  # Add the document to the list

# Insert all documents into the collection
processed_doc_collection.insert_many(document_list)
```

This code snippet reads the preprocessed textual data from the file ‘processed_texts.txt’, stores it into a list of documents, and then inserts those documents into a MongoDB collection named ‘processed textual data’.

Queries



NLP.vectorization metadata

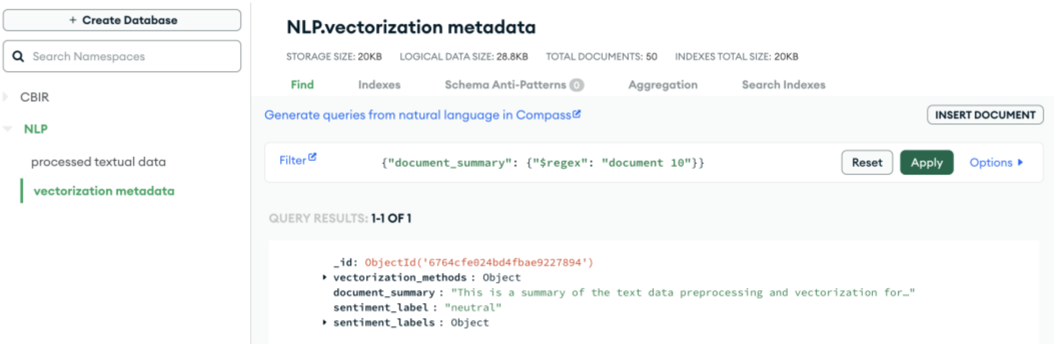
STORAGE SIZE: 20KB LOGICAL DATA SIZE: 28.8KB TOTAL DOCUMENTS: 50 INDEXES TOTAL SIZE: 20KB

```
{"sentiment_label": "positive"}
```

QUERY RESULTS: 1-1 OF 1

```
  _id: ObjectId('6764cfe024bd4fbae92278b6')
▸ vectorization_methods : Object
  document_summary : "This is a summary of the text data preprocessing and vectorization for…"
  sentiment_label : "positive"
▸ sentiment_labels : Object
```

- Checks for documents in ‘vectorization metadata’ where the sentiment label is ‘positive’.



NLP.vectorization metadata

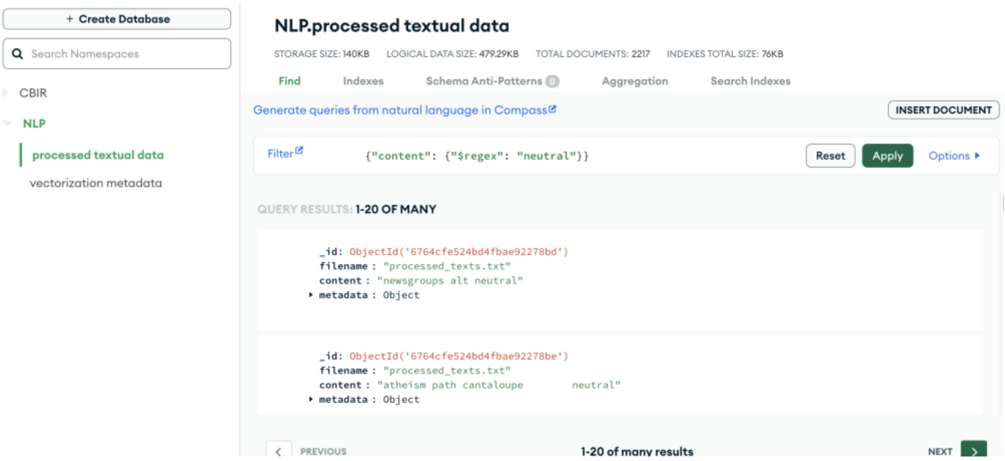
STORAGE SIZE: 20KB LOGICAL DATA SIZE: 28.8KB TOTAL DOCUMENTS: 50 INDEXES TOTAL SIZE: 20KB

```
{"document_summary": {"$regex": "document 10"}}
```

QUERY RESULTS: 1-1 OF 1

```
  _id: ObjectId('6764cfe024bd4fbae9227894')
▸ vectorization_methods : Object
  document_summary : "This is a summary of the text data preprocessing and vectorization for…"
  sentiment_label : "neutral"
▸ sentiment_labels : Object
```

- Retrieves vectorisation details and sentiment label of ‘document 10’.



NLP.processed textual data

STORAGE SIZE: 140KB LOGICAL DATA SIZE: 479.29KB TOTAL DOCUMENTS: 2217 INDEXES TOTAL SIZE: 76KB

```
{"content": {"$regex": "neutral"}}
```

QUERY RESULTS: 1-20 OF MANY

```
  _id: ObjectId('6764cfe524bd4fbae92278bd')
  filename : "processed_texts.txt"
  content : "newsgroups alt neutral"
▸ metadata : Object

  _id: ObjectId('6764cfe524bd4fbae92278be')
  filename : "processed_texts.txt"
  content : "atheism path cantaloupe        neutral"
▸ metadata : Object
```

1-20 of many results

- Checks for documents with the word ‘neutral’ in the content.

5. References

- Munjal, M.N. (2018). *A Deep Study of Content Based Image Retrieval System Using Sentiment Analysis*. [online] ResearchGate. Available at: https://www.researchgate.net/publication/341411954_A_Deep_Study_of_Content_Based_Image_Retrieval_System_using_Sentiment_Analysis [Accessed 18 Dec. 2024].

- Pascual, F. (2022). *Getting Started with Sentiment Analysis using Python*. [online] huggingface.co. Available at: https://huggingface.co/blog/sentiment-analysis-python.

- HyScaler. (2024). *Sentiment Analysis: A Step-by-Step Guide - HyScaler*. [online] Available at: https://hyscaler.com/insights/sentiment-analysis-guide/

- GUVI Geek (2020). *MongoDB vs. MySQL: Which Database Should You Learn? - GUVI Blogs*. [online] GUVI Blogs. Available at: https://www.guvi.in/blog/mongodb-vs-mysql-which-is-the-best-to-learn/ [Accessed 18 Dec. 2024].

- Hackers Realm (2023). *Feature Extraction of Text Data using Bag of Words | NLP | Python*. [online] Hackers Realm. Available at: https://www.hackersrealm.net/post/bag-of-words-python [Accessed 19 Dec. 2024].

- Hackers Realm (2023). *Feature Extraction using Term Frequency - Inverse Document Frequency (TF-IDF) | NLP | Python*. [online] Hackers Realm. Available at: https://www.hackersrealm.net/post/tf-idf-python [Accessed 19 Dec. 2024].

- Mohit (2021). *Sentiment Analysis with Textblob and Vader in Python*. [online] Analytics Vidhya. Available at: https://www.analyticsvidhya.com/blog/2021/10/sentiment-analysis-with-textblob-and-vader/.