

5DATA005W

Data Engineering

COURSEWORK

Module Leader:

Dr Habeeb Balogun

Full Name: Sithuli Nayanama Kothalawala

Blackboard Name: Sithuli Kothalawala

IIT ID: 20230336

UOW ID: 20521288

UOW username : W2052128

Table of Contents

Objectives : Purpose of Databases	3
Content-Based Image Retrieval System (CBIR) :	3
Sentiment Analysis Model Training (NLP) :	3
Data Sources	4
Content-Based Image Retrieval System (CBIR) :	4
Sentiment Analysis Model Training (NLP) :	7
Methodology	8
Content-Based Image Retrieval System (CBIR) :	8
a)Image Collection.....	8
b) Image processing	9
c) Image Annotation	11
d) Image Feature Extraction.....	12
e) Database Design	16
Database for sentiment analysis models (NLP)	21
a)Data Collection	22
b) Data preprocessing.....	23
c)Text Vectorization	25
d)Metadata and labelling.....	29
e) Database creation	31
Storage and retrieval	34
Database for content-based image retrieval system (CBIR)	34
Database for sentiment analysis models	37
MongoDB username and password	39
References.....	39

Objectives : Purpose of Databases

The primary purpose of the databases created for this coursework:

Content-Based Image Retrieval System (CBIR) :

The objective of the CBIR database is to store and manage a collection of processed images along with their metadata and extracted features. CBIR database enables searches based on image content rather than textual descriptions and supports various applications including visual search engines, digital libraries and e-commerce platforms. In this case CBIR database is created to store data of a pet shop.

Sentiment Analysis Model Training (NLP) :

The objective of the sentiment analysis database is to provide a structured repository of text data labelled with sentiment categories such as positive, negative or neutral. This data is essential for training and evaluating machine learning models that can detect and interpret emotional tones in text.

Data Sources

Content-Based Image Retrieval System (CBIR) :

To obtain images I have mainly used Google search engine and Pinterest and a few other websites. Here are the raw data sources for the 50 images separately.

image_001.jpg - <https://en.wikipedia.org/wiki/Thrianta>

image_002.jpg - <https://www.petassure.com/new-newsletters/loveable-lovebirds/>

image_003.jpg - <https://pin.it/3nfglt4xx>

image_004.jpg - <https://images.app.goo.gl/47iSgiC8xvaaqwwP7>

image_005.jpg - - <https://www.ubuy.com.lk/en/product/41PRH3V4G-pedigree-complete-nutrition-adult-dry-dog-food-roasted-chicken-rice-vegetable-flavor-dog-kibble-30-lb-bag>

image_006.jpg - <https://www.pinterest.com/pin/1037446464143118402/>

image_007.jpg - <https://www.pinterest.com/pin/184929128440605479/>

image_008.jpg - <https://www.pinterest.com/pin/278097345733381814/>

image_010.jpg - <https://www.pinterest.com/pin/436849232584191689/>

image_011.jpg - <https://www.pinterest.com/pin/982347737506757417/>

image_012.jpg - <https://www.pinterest.com/pin/1015421047230731796/>

image_013.jpg - <https://www.pinterest.com/pin/26740191529386899/>

image_014.jpg - <https://www.pinterest.com/pin/66780006965521146/>

image_015.jpg - <https://www.pinterest.com/pin/351491945932789456/>

image_016.jpg - <https://www.pinterest.com/pin/638174209689781734/>

image_017.jpg - <https://www.pinterest.com/pin/436567757642130195/>

image_018.jpg - <https://nativecollars.com/products/biothane-two-tone-collar-mix-match>

image_019.jpg - <https://www.pinterest.com/pin/982347737507146704/>

image_020.jpg - <https://www.pinterest.com/pin/2462974791071832/>

image_021.jpg - <https://www.pinterest.com/pin/180707003789790146/>

image_022.jpg - <https://www.pinterest.com/pin/1148206867475373037/>

image_023.jpg - <https://www.pinterest.com/pin/986851337087573609/>

image_024.jpg - <https://www.pinterest.com/pin/741545894928422210/>

image_025.jpg - <https://www.pinterest.com/pin/556546466453599994/>

image_026.jpg - <https://www.pinterest.com/pin/845762005015654487/>

image_027.jpg - <https://www.pinterest.com/pin/1072630836234203884/>

image_028.jpg - <https://www.pinterest.com/pin/893190538711772725/>

image_29.jpg - <https://www.pinterest.com/pin/724657396312946530/>

image_030.jpg - <https://www.pinterest.com/pin/1122451907124248014/>

image_031.jpg - <https://www.pinterest.com/pin/880594533375384364/>

image_032.jpg - <https://www.pinterest.com/pin/76068681191638750/>

image_033.jpg - <https://lafeber.com/pet-birds/species/red-lory/>

image_034.jpg - <https://www.pinterest.com/pin/903956956434510082/>

image_035.jpg - <https://www.pinterest.com/pin/48202658504944014/>

image_036.jpg - <https://www.pinterest.com/pin/947726315331337107/>

image_037.jpg - <https://www.pinterest.com/pin/1022035709193146758/>

image_038.jpg - <https://www.pinterest.com/pin/186829084534312146/>

image_039.jpg - <https://www.pinterest.com/pin/815503445059893999/>

image_040.jpg - <https://www.pinterest.com/pin/366691594678806321/>

image_041.jpg - <https://pangovet.com/pet-breeds/birds/white-bellied-caique/>

image_042.jpg - <https://www.pinterest.com/pin/12455336456313483/>

image_043.jpg - <https://www.pinterest.com/pin/1048212882013872577/>

image_044.jpg - <https://www.pinterest.com/pin/51369251995444059/>

image_045.jpg - <https://www.pinterest.com/pin/683632418468775318/>

image_046.jpg - <https://www.pinterest.com/pin/403212972872086821/>

image_047.jpg - <https://www.pinterest.com/pin/328199891594632841/>

image_048.jpg - <https://www.pinterest.com/pin/592716000950200309/>

image_049.jpg - <https://www.whiskas.com.ph/cat-food-products/wet-cat-food/tuna-cat-food-pouch-adult-1>

image_50.jpg - <https://www.pinterest.com/pin/786863366151668931/>

Sentiment Analysis Model Training (NLP) :

To obtain textual data I used a few data sources. I mostly used AliExpress to obtain customer reviews. Other than that, I have used a few datasets from Kaggle. Following are the data sources;

<https://www.aliexpress.com/>

<https://best.aliexpress.com/>

<https://www.kaggle.com/datasets/mrmars1010/iphone-customer-reviews-nlp>

<https://www.kaggle.com/datasets/michelhatab/hotel-reviews-bookingcom>

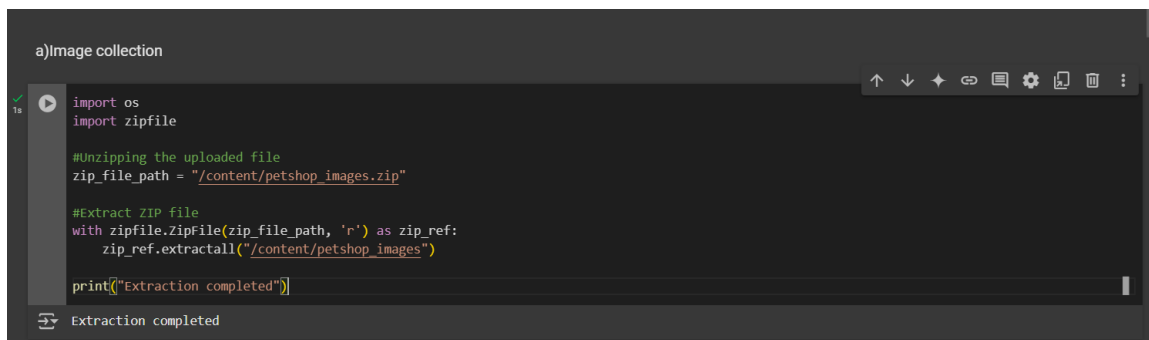
<https://www.kaggle.com/datasets/vigneshwarsofficial/reviews>

Methodology

Content-Based Image Retrieval System (CBIR) :

a)Image Collection

Initially 50 images were gathered under the theme ‘pet shop’. Images of pets, pet accessories, pet food, pet toys and etc were gathered from multiple sources into a folder.



```
a)Image collection

import os
import zipfile

#Unzipping the uploaded file
zip_file_path = "/content/petshop_images.zip"

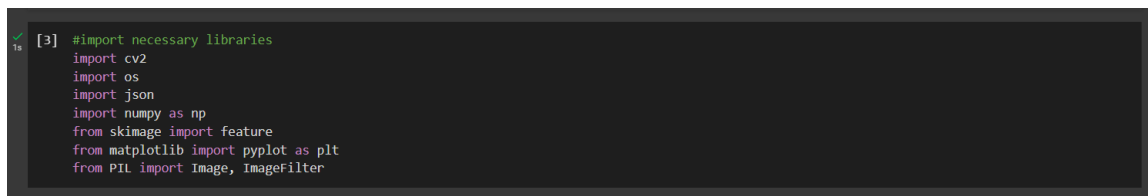
#Extract ZIP file
with zipfile.ZipFile(zip_file_path, 'r') as zip_ref:
    zip_ref.extractall("/content/petshop_images")

print("Extraction completed")
```

Extraction completed

The ZIP file of the image folder ‘petshop_images’ is uploaded and unzipped. The ZIP file is opened in “read” mode and then the opened ZIP file object is assigned to the variable zip_ref and extracted.

Then all the necessary libraries are imported



```
[3] #import necessary libraries
import cv2
import os
import json
import numpy as np
from skimage import feature
from matplotlib import pyplot as plt
from PIL import Image, Imagefilter
```


b) Image processing

```
b)Image preprocessing

import os
from PIL import Image, ImageFilter
import matplotlib.pyplot as plt

# Define paths
input_folder = "/content/petshop_images/petshop_images" # Directly use the extraction path
output_folder = "/content/processed_images"
os.makedirs(output_folder, exist_ok=True)

processed_images = []

# Sort files lexicographically (consistent across sessions)
file_list = sorted([f for f in os.listdir(input_folder) if f.lower().endswith(('.jpg', '.jpeg', '.png'))])
```

After importing the necessary libraries, the file paths are defined. Input folder specifies the directory containing original images. Output folder specifies where processed images will be saved. `os.makedirs()` creates the folder if it doesn't exist.

`file_list` is a sorted list of image filenames with extensions `.jpg`, `.jpeg`, `.png` in the input folder ensuring consistent order across the sessions.

Then the images are loaded and are converted to RGB mode

```
for idx, filename in enumerate(file_list):
    image_path = os.path.join(input_folder, filename)
    with Image.open(image_path) as img:
        # Convert image to RGB mode
        img_rgb = img.convert('RGB')

    # Display the RGB converted image (for the first image only)
    if idx == 0:
        plt.imshow(img_rgb)
        plt.title("RGB Converted Image")
        plt.axis('off')
        plt.show()
```

The first RGB converted image is displayed as follows

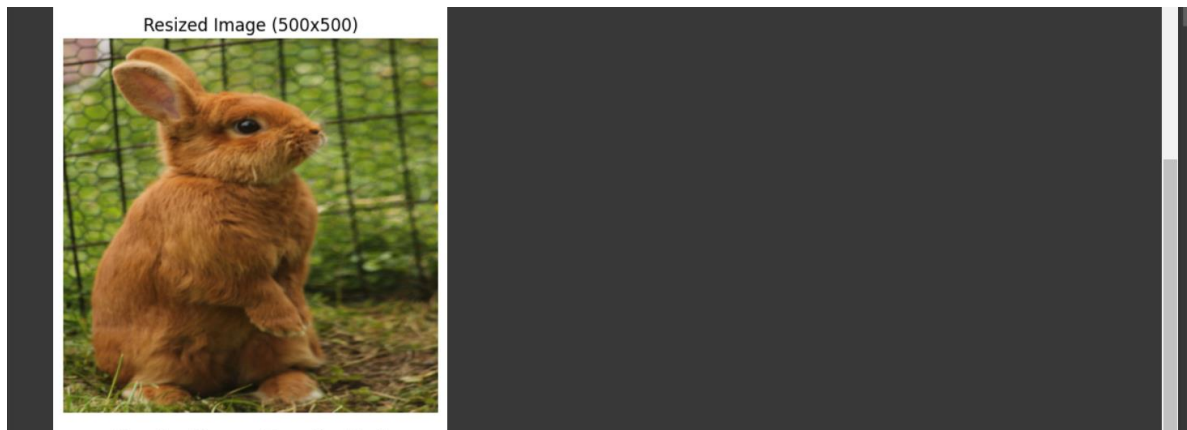


Then the images are resized to 500*500 pixels using `.resize()`

```
# Resize to 500x500
img_resized = img_rgb.resize((500, 500))

# Display the resized image (for the first image only)
if idx == 0:
    plt.imshow(img_resized)
    plt.title("Resized Image (500x500)")
    plt.axis('off')
    plt.show()
```

The first resized image is displayed as follows



Then Gaussian blur is applied to reduce the noise of the image using `.filter(ImageFilter.GaussianBlur(1))`

```
# Apply Gaussian blur for denoising
img_denoised = img_resized.filter(ImageFilter.GaussianBlur(1))

# Display the denoised image (for the first image only)
if idx == 0:
    plt.imshow(img_denoised)
    plt.title("Denoised Image (Gaussian Blur)")
    plt.axis('off')
    plt.show()
```

The blurred image for the first image is displayed as follows.



```

# Save with a systematic name
new_name = f"image_{idx + 1:03d}.jpg"
output_path = os.path.join(output_folder, new_name)
img_denoised.save(output_path)
processed_images.append(output_path)

print(f"Preprocessed {len(processed_images)} images.")

```

```

Preprocessed 50 images.
/usr/local/lib/python3.10/dist-packages/PIL/Image.py:1054: UserWarning: Palette images with Transparency expressed in bytes should be converted to RGBA images
  warnings.warn(

```

At the end , it print the total number of images processed and save in the output folder.

c) Image Annotation

The metadata dictionary is manually created to store annotations for 50 images.

Each image filename (e.g., image_001.jpg) acts as a key. For every image keywords, which is a list of attributes related to the image and the description which is a detailed textual description is given.

Then metadata is saved into a JSON file named image_metadata.json

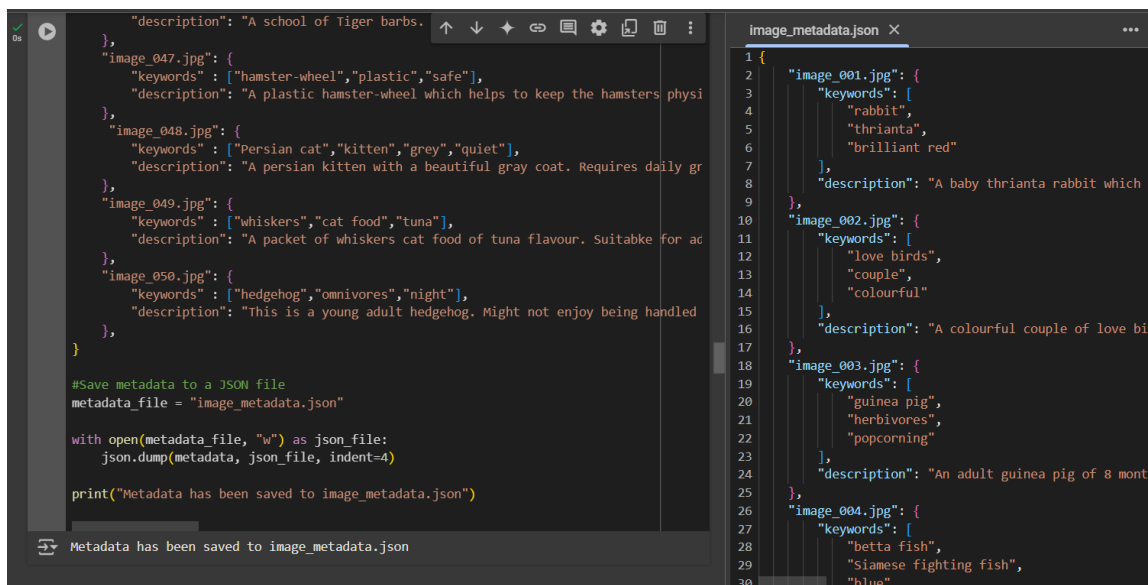
```

c)Image Annotation

import json

#metadata for processed images
metadata = {
    "image_001.jpg": {
        "keywords": ["rabbit", "thrianta", "brilliant red"],
        "description": "A baby thrianta rabbit which is a domestic rabbit species and is brilliant red in colour and weighs around 2-3 pounds"
    },
    "image_002.jpg": {
        "keywords": ["love birds", "couple", "colourful"],
        "description": "A colourful couple of love birds. One is a Dutch Blue Lovebird and the other one is a Lillian's love bird. The couple or"
    },
    "image_003.jpg": {
        "keywords": ["guinea pig", "herbivores", "popcorning"],
        "description": "An adult guinea pig of 8 months old. Still in need of the necessary vitamins. Should never be fed highly processed food"
    },
    "image_004.jpg": {
        "keywords": ["betta fish", "Siamese fighting fish", "blue", "bowl"],
        "description": "A blue betta fish also known as Siamese fighting fish. The fish is 2 inches long and has the potential to grow about and"
    },
    "image_005.jpg": {
        "keywords": ["dog food", "pedigree", "adult"],
        "description": "Roasted chicken and vegetable faloured pedigree packets containing 100% complete and balanced food for adult dogs"
    },
    "image_006.jpg": {
        "keywords": ["parakeet", "budgerigar", "couple", "intelligent"],
        "description": "A couple of budgergars also known as commonly parakeets. One of the birds has a beautiful mix of sky blue and white fea

```



```
},
    "description": "A school of Tiger barbs."
},
    "image_047.jpg": {
        "keywords": ["hamster-wheel", "plastic", "safe"],
        "description": "A plastic hamster-wheel which helps to keep the hamsters physi
    },
    "image_048.jpg": {
        "keywords": ["Persian cat", "kitten", "grey", "quiet"],
        "description": "A persian kitten with a beautiful gray coat. Requires daily gr
    },
    "image_049.jpg": {
        "keywords": ["whiskers", "cat food", "tuna"],
        "description": "A packet of whiskers cat food of tuna flavour. Suitabke for ac
    },
    "image_050.jpg": {
        "keywords": ["hedgehog", "omnivores", "night"],
        "description": "This is a young adult hedgehog. Might not enjoy being handled
    },
}

#Save metadata to a JSON file
metadata_file = "image_metadata.json"

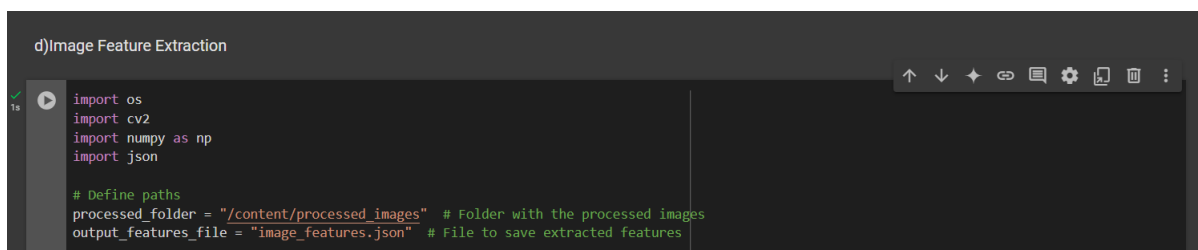
with open(metadata_file, "w") as json_file:
    json.dump(metadata, json_file, indent=4)

print("Metadata has been saved to image_metadata.json")
```

Metadata has been saved to image_metadata.json

```
1 {
2   "image_001.jpg": {
3     "keywords": [
4       "rabbit",
5       "thrianta",
6       "brilliant red"
7     ],
8     "description": "A baby thrianta rabbit which
9   },
10  "image_002.jpg": {
11    "keywords": [
12      "love birds",
13      "couple",
14      "colourful"
15    ],
16    "description": "A colourful couple of love bi
17  },
18  "image_003.jpg": {
19    "keywords": [
20      "guinea pig",
21      "herbivores",
22      "popcorning"
23    ],
24    "description": "An adult guinea pig of 8 mont
25  },
26  "image_004.jpg": {
27    "keywords": [
28      "beta fish",
29      "Siamese fighting fish",
30      "blue"
```

d) Image Feature Extraction



```
d)Image Feature Extraction

import os
import cv2
import numpy as np
import json

# Define paths
processed_folder = "/content/processed_images" # Folder with the processed images
output_features_file = "image_features.json" # File to save extracted features
```

First the necessary libraries are imported. Then the paths are defined as ;

`processed_folder` : path to the folder containing processed images

`output_features_file` : The name of the JSON file where extracted features will be saved.

Feature Extraction Function

The `extracted_features()` function extracts several features from an image.

```

# Function to calculate features for an image
def extract_features(image_path):
    image = cv2.imread(image_path)
    image_gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY) # Converts to grayscale

    # Feature 1: Mean pixel intensity
    mean_intensity = np.mean(image_gray)

    # Feature 2: Norm of pixel intensities
    norm_intensity = np.linalg.norm(image_gray)

    # Feature 3: Texture descriptor (Histogram for simplicity)
    glcm = cv2.calcHist([image_gray], [0], None, [256], [0, 256]).flatten()
    texture_desc = glcm.tolist()

```

Load and Convert to grayscale

- cv2.imread is used to load the image
- It is converted to grayscale with cv2.cvtColor()

Feature 1 : Mean Pixel Intensity

- Computes the average intensity of the grayscale image using np.mean()

Feature 2 : Norm of pixel intensities

- Computes the Euclidean norm (magnitude) of the pixel intensities using np.linalg.norm()

Feature 3 : Texture Descriptor

- Calculates a histogram of pixel intensity values using cv2.calcHist()
- Flattens the histogram into a 1D list using .flatten() and converts it to a Python list

```
# Feature 4: Shape features
_, thresholded = cv2.threshold(image_gray, 128, 255, cv2.THRESH_BINARY)
contours, _ = cv2.findContours(thresholded, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

shape_features = []
for contour in contours:
    # Area
    area = cv2.contourArea(contour)

    # Perimeter
    perimeter = cv2.arcLength(contour, True)

    # Centroid
    moments = cv2.moments(contour)
    if moments["m00"] != 0:
        cx = int(moments["m10"] / moments["m00"])
        cy = int(moments["m01"] / moments["m00"])
    else:
        cx, cy = 0, 0

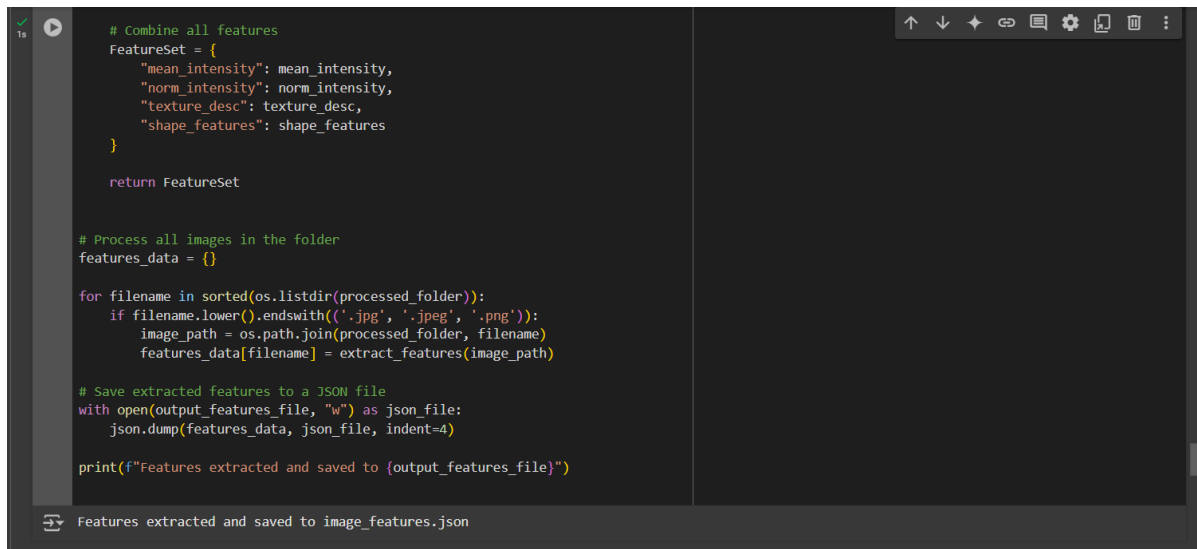
    # Bounding Box
    x, y, w, h = cv2.boundingRect(contour)

    # Append the shape features dictionary
    shape_features.append({
        "area": area,
        "perimeter": perimeter,
        "centroid": [cx, cy],
        "bounding_box": [x, y, w, h]
    })
```

Feature 4 : Shape features

- Thresholding : Segments the image into a binary (black and white) using `cv2.threshold()`
- Contours : Detects contours (edges) in the binary image using `cv2.findContours()`
- For each contour Area, Perimeter, Centroid (determines the centre of the shape using image moments) and Bounding Box (finds the smallest rectangle enclosing the contour) are calculated.

All the features are combined into a dictionary called FeatureSet



```
# Combine all features
FeatureSet = {
    "mean_intensity": mean_intensity,
    "norm_intensity": norm_intensity,
    "texture_desc": texture_desc,
    "shape_features": shape_features
}

return FeatureSet

# Process all images in the folder
features_data = {}

for filename in sorted(os.listdir(processed_folder)):
    if filename.lower().endswith(('.jpg', '.jpeg', '.png')):
        image_path = os.path.join(processed_folder, filename)
        features_data[filename] = extract_features(image_path)

# Save extracted features to a JSON file
with open(output_features_file, "w") as json_file:
    json.dump(features_data, json_file, indent=4)

print(f"Features extracted and saved to {output_features_file}")
```

Features extracted and saved to image_features.json

Then a dictionary called `features_data` was initialized to store the extracted features for all the images. Each image's filename will be the key and the extracted feature will be the value. All the files in the `processed_folder` is listed. The if condition ensures only image files with those extensions are processed. Then a full image path is constructed as `image_path`. `extracted_features()` function is called with `image_path` as input.

Then the extracted features are saved into a JSON file.

e) Database Design

```
e) Database Design

! pip install "pymongo[srv]"

Collecting pymongo[srv]
  Downloading pymongo-4.10.1-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (22 kB)
WARNING: pymongo 4.10.1 does not provide the extra 'srv'
Collecting dnspython<3.0.0,>=1.16.0 (from pymongo[srv])
  Downloading dnspython-2.7.0-py3-none-any.whl.metadata (5.8 kB)
  Downloading dnspython-2.7.0-py3-none-any.whl (313 kB)
    313.6/313.6 kB 6.8 MB/s eta 0:00:00
  Downloading pymongo-4.10.1-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (1.4 MB)
    1.4/1.4 MB 31.5 MB/s eta 0:00:00
Installing collected packages: dnspython, pymongo
Successfully installed dnspython-2.7.0 pymongo-4.10.1

[10] import pymongo
      from pymongo import MongoClient

[11] imagedb_connection = pymongo.MongoClient("mongodb+srv://sithulikothalawala:VFV7uY4izXtmalF9@cluster0.hvv2h.mongodb.net/?retryWrites=true&w=major
      db = imagedb_connection["CBIR"]
```

To allow Python to interact with MongoDB ‘pymongo’ library was installed. Then the necessary libraries are imported.

Then the connection to MongoDB is created and CBIR database is selected.

```
[12] #Test the connection
      try:
          print("Collections in CBIR database:", db.list_collection_names())
      except Exception as e:
          print(f'Connection failed: {e}')

Collections in CBIR database: ['image_processed', 'petshop_metadata', 'feature_image']
```

It is checked if the connection was successful by listing the collections in the CBIR database. Prints an error message if the connection fails.

Inserting Image Metadata into petshop_metadata Collection.

```
[ ] # Storing image metadata JSON file into 'petshop_metadata' collection
petshop_metadata_collection = db['petshop_metadata']

# Storing image metadata JSON file into 'petshop_metadata' collection
petshop_metadata_collection = db['petshop_metadata']
with open('/content/image_metadata.json') as file: # Path to image metadata file
    petshop_meta_data = json.load(file)

# Insert the metadata into the MongoDB collection
if isinstance(petshop_meta_data, list):
    petshop_metadata_collection.insert_many(petshop_meta_data)
else:
    petshop_metadata_collection.insert_one(petshop_meta_data)

print("Inserted image metadata into 'petshop_metadata' collection.")
```

↗ Inserted image metadata into 'petshop_metadata' collection.

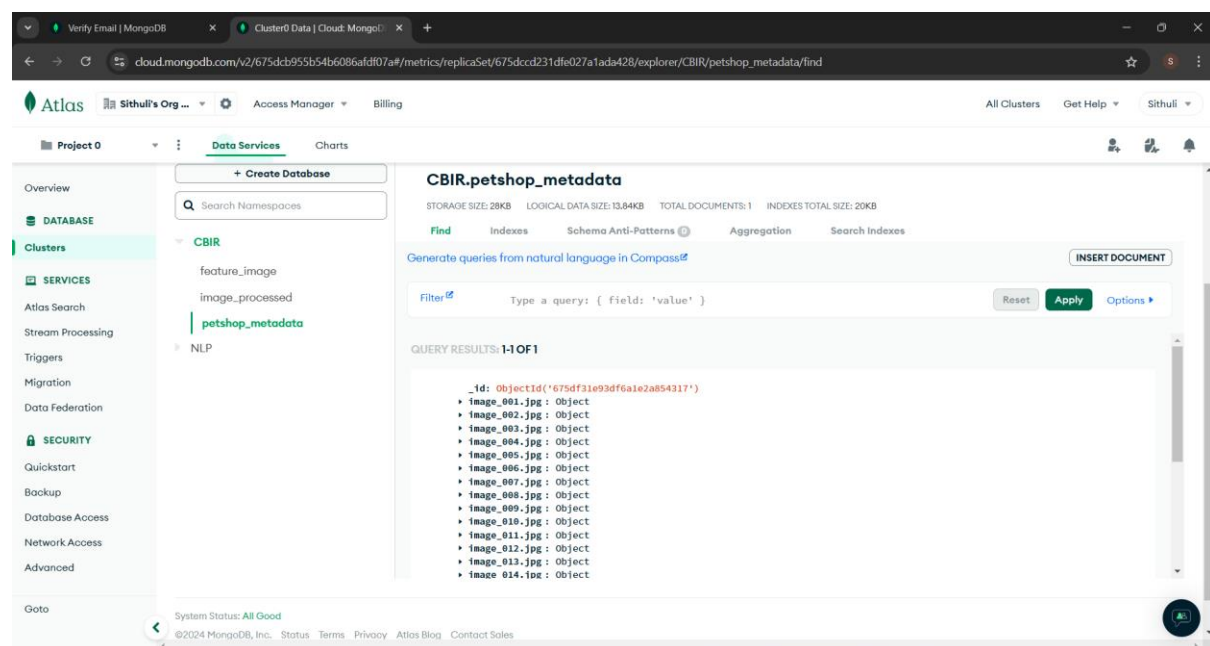
petshop_metadata_collection accesses the 'petshop_metadata' collection in the database. Then the image_metadata.json file is read.

Checks if the data is a list:

- Inserts multiple documents with 'insert_many'.
- Otherwise, inserts a single document with 'insert_one'

A statement is printed to display that the image_metadata is saved to petshop_metadata

Following is the output in MongoDB showing petshop_metadata



Inserting Image features into feature_image Collection

```
# Storing image features JSON file into 'feature_image' collection
feature_image_collection = db['feature_image']

with open('/content/image_features.json') as file: # Path to image features file
    image_feature_data = json.load(file)

# Insert the features into the MongoDB collection
if isinstance(image_feature_data, list):
    feature_image_collection.insert_many(image_feature_data)
else:
    feature_image_collection.insert_one(image_feature_data)

print("Inserted image features into 'feature_image' collection.")
```

Inserted image features into 'feature_image' collection.

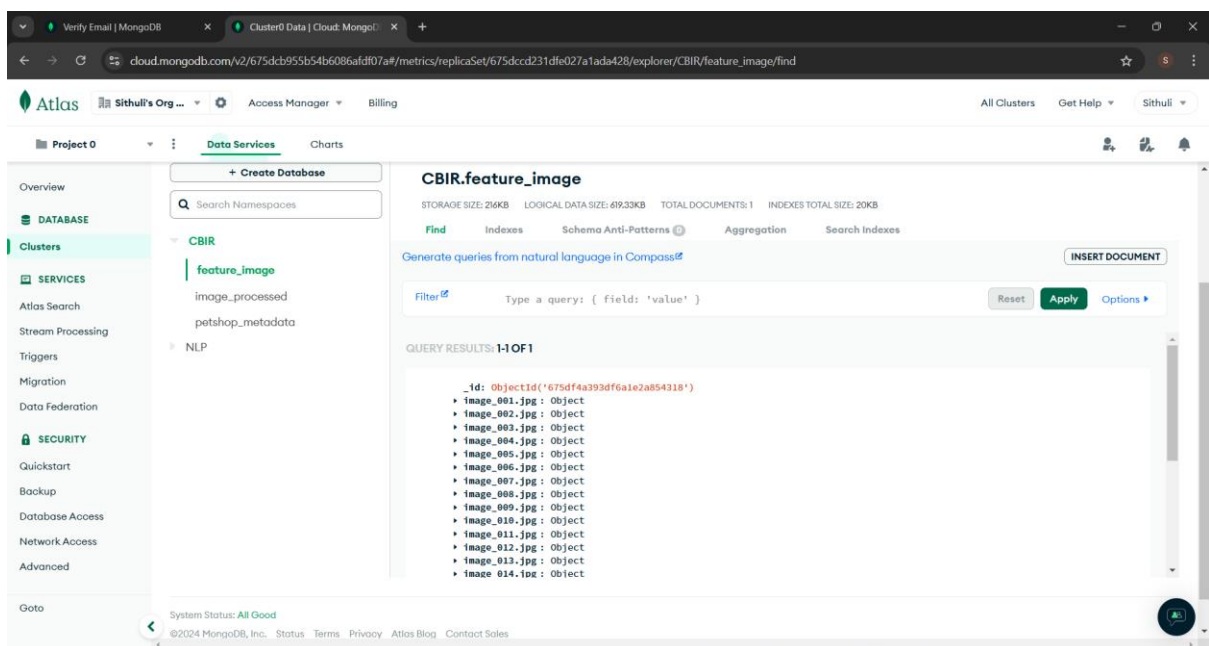
Feature_image_collection accesses the 'feature_image' collection in the database. Then the image_features.json file is read.

Checks if the data is a list:

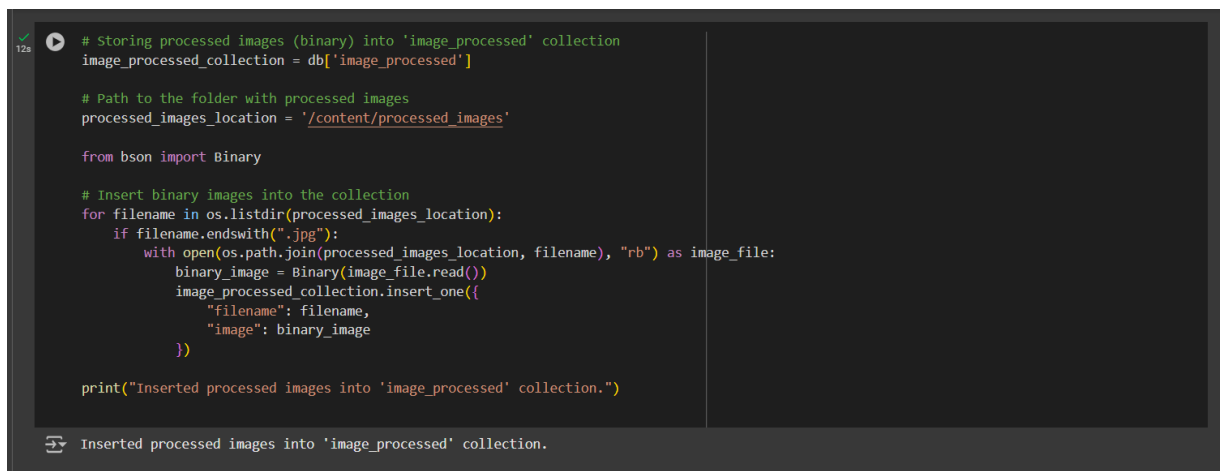
- Inserts multiple documents with 'insert_many'.
- Otherwise, inserts a single document with 'insert_one'

A statement is printed to display that the image_features is saved to feature_image

Following is the output in MongoDB showing feature_image



Insert Binary Processed Images into images_processed Collection



```
# Storing processed images (binary) into 'image_processed' collection
image_processed_collection = db['image_processed']

# Path to the folder with processed images
processed_images_location = '/content/processed_images'

from bson import Binary

# Insert binary images into the collection
for filename in os.listdir(processed_images_location):
    if filename.endswith(".jpg"):
        with open(os.path.join(processed_images_location, filename), "rb") as image_file:
            binary_image = Binary(image_file.read())
            image_processed_collection.insert_one({
                "filename": filename,
                "image": binary_image
            })

print("Inserted processed images into 'image_processed' collection.")
```

Inserted processed images into 'image_processed' collection.

`db['image_processed']` selects the `image_processed` collection from the MongoDB database to store the processed images.

`Processed_images_location` is the folder where the processed images are stored.

Binary allows the image file data to be stored as binary format in MongoDB (uses BSON format)

All the files in that folder are listed and filters only files that have `.jpg` extensions (processed images)

Folder path and filename are combined to get the full path.

Then the binary images are inserted into the MongoDB collection.

Once the loop is complete a confirmation statement is printed.

Below is how it is shown in MongoDB.

Verify Email | MongoDB Cluster0 Data | Cloud Mongo... cloud.mongodb.com/v2/675dcb955b54b6086afdf07a#/metrics/replicaSet/675dccb231dfe027a1ada428/explorer/CBIR/image_processed/find

Atlas Sithuli's Org ... Access Manager Billing All Clusters Get Help Sithuli

Project 0 Data Services Charts

Overview DATABASE Clusters SERVICES Atlas Search Stream Processing Triggers Migration Data Federation SECURITY Quickstart Backup Database Access Network Access Advanced Goto

+ Create Database Search Namespaces

CBIR feature_image image_processed potshop_metadata NLP

CBIR.image_processed STORAGE SIZE: 1.32MB LOGICAL DATA SIZE: 1.28MB TOTAL DOCUMENTS: 50 INDEXES TOTAL SIZE: 20KB Find Indexes Schema Anti-Patterns Aggregation Search Indexes

Generate queries from natural language in Compass

INSERT DOCUMENT

Filter Type a query: { field: 'value' } Reset Apply Options

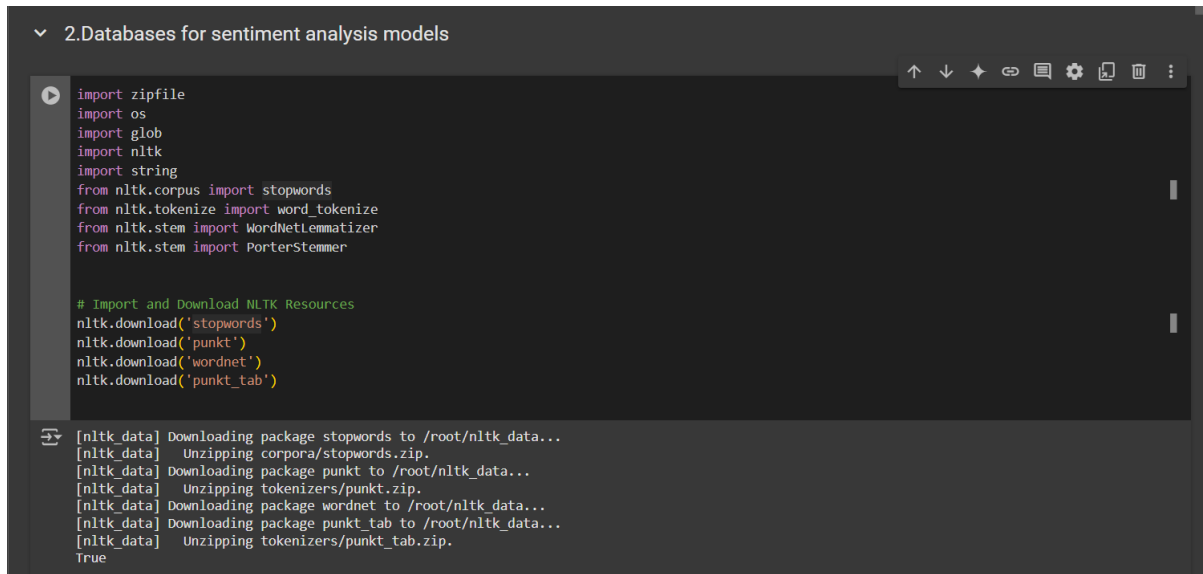
QUERY RESULTS: 1-20 OF MANY

```
{ "_id": ObjectId("675e6cf34dfd46996ca77f35"),
  "filename": "image_017.jpg",
  "image": Binary.createFromBase64('/9j/4AAQSkZJRgABAQAAQABAAQ//GA7Q13FQVRPUjogZ2Z0tan8LZyB2M54wICh1c2luZyB3SkcgS1BFYyB2OTU...') }
```

PREVIOUS 1-20 of many results NEXT

System Status: All Good ©2024 MongoDB, Inc. Status Terms Privacy Atlas Blog Contact Sales

Database for sentiment analysis models (NLP)



```
2.Databases for sentiment analysis models

import zipfile
import os
import glob
import nltk
import string
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import WordNetLemmatizer
from nltk.stem import PorterStemmer

# Import and Download NLTK Resources
nltk.download('stopwords')
nltk.download('punkt')
nltk.download('wordnet')
nltk.download('punkt_tab')

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

Firstly, the necessary libraries are imported. nltk (Natural Language Toolkit) is a library for text processing and NLP tasks. Then NLTK resources are imported and downloaded.

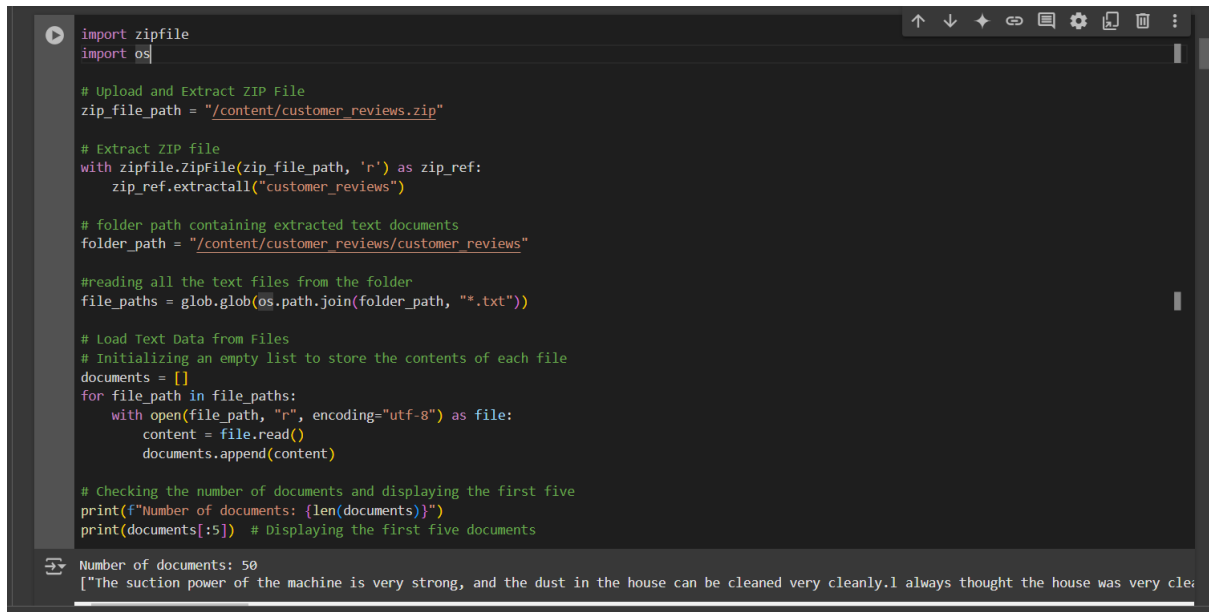
stopwords : A list of commonly used words that are often removed during text preprocessing.

word_tokenize : Splits a sentence into individual words (tokens)

WordNetLemmatizer : Reduces words to their base form

PorterStemmer : Reduces words to their stem form by chopping off suffixes.

a)Data Collection



```
import zipfile
import os

# Upload and Extract ZIP File
zip_file_path = "/content/customer_reviews.zip"

# Extract ZIP file
with zipfile.ZipFile(zip_file_path, 'r') as zip_ref:
    zip_ref.extractall("customer_reviews")

# folder path containing extracted text documents
folder_path = "/content/customer_reviews/customer_reviews"

#reading all the text files from the folder
file_paths = glob.glob(os.path.join(folder_path, "*.txt"))

# Load Text Data from Files
# Initializing an empty list to store the contents of each file
documents = []
for file_path in file_paths:
    with open(file_path, "r", encoding="utf-8") as file:
        content = file.read()
        documents.append(content)

# Checking the number of documents and displaying the first five
print(f"Number of documents: {len(documents)}")
print(documents[:5]) # Displaying the first five documents
```

Number of documents: 50
["The suction power of the machine is very strong, and the dust in the house can be cleaned very cleanly.I always thought the house was very cle;

After importing the required libraries zip file path is defined. Then all the content is extracted into the “customer_reviews” folder. folder_path specifies the directory containing the extracted text files.

glob.glob() finds all files with the extension *.txt in the specified folder. file_paths contains a list of paths to all the .txt files.

A list called documents is created to store text content of each file in the folder, read each file line by line, appends the content to the list and prints the total number of files along with first five for verification.

b) Data preprocessing

```
b) Data preprocessing

import os
import string
from nltk.tokenize import word_tokenize

# Define input and output folder paths
input_folder = "/content/customer_reviews/customer_reviews" # Folder containing raw reviews
output_folder = "/content/processed_customer_reviews" # Folder to save preprocessed reviews

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

Initially necessary libraries are imported.

input_folder contains the raw text files (customer reviews)

output_folder is the folder where the processed reviews will be saved.

Defining the preprocessing function

```
def process_all_files(input_folder, output_folder):
    # Calculate the number of digits for zero-padding
    total_files = len([f for f in os.listdir(input_folder) if f.endswith(".txt")])
    num_digits = len(str(total_files))

    # examples for the first three files
    examples = []

    for i, filename in enumerate(sorted(os.listdir(input_folder)), 1):
        if filename.endswith(".txt"):
            file_path = os.path.join(input_folder, filename)

            with open(file_path, 'r', encoding='utf-8') as file:
                text = file.read()

            # Tokenize, remove punctuation, and convert to lowercase
            tokens = [word for word in word_tokenize(text.lower()) if word not in string.punctuation]
```

First the code counts how many .txt files are in the input folder. It calculates the number of digits for zero-padding file names based on the total count.

An empty list is created to store example outputs for the first three files for display purpose.

enumerate loops through all files assigning an index i (starting at 1) and the file name. if condition ensures only files ending with .txt are processed.

The file content are read as a string

i.Convert to Lowercase : text.lower() ensures all text is lowercase.

ii.Tokenize : word_tokenize splits the text into tokens

```
# examples for the first three files
if i <= 3:
    examples.append({
        "file_name": filename,
        "lowercase_text": text.lower(),
        "tokens": tokens
    })

# Write processed tokens to output file
output_file_path = os.path.join(output_folder, f"{i:0>{num_digits}}_preprocessed_customer_reviews.txt")
with open(output_file_path, 'w', encoding='utf-8') as output_file:
    for token in tokens:
        output_file.write(token + "\n")
```

For the first three files code saves the file name, the lowercased text and the tokens.

Files are named like 01_preprocessed_customer_reviews.txt where 01 is a padded using the calculated num_digits.

```
# Print examples for the first three files
print("\n--- Sample Outputs (First 3 Files) ---")
for i, example in enumerate(examples):
    print(f"\nFile: {example['file_name']}")
    print(f"Lowercased Text:\n{example['lowercase_text'][:100]}...")
    print(f"Tokens:\n{example['tokens'][:10]}...")

    print("\nProcessed files and saved.")

# Run the preprocessing function
process_all_files(input_folder, output_folder)
```

```
--- Sample Outputs (First 3 Files) ---

File: textual_data_01.txt
Lowercased Text:
overall evaluation: very good. fabric material: comfortable without being heavy. style: classic and ...
Tokens:
['overall', 'evaluation', 'very', 'good', 'fabric', 'material', 'comfortable', 'without', 'being', 'heavy']...

File: textual_data_02.txt
Lowercased Text:
ordered 3 keyfob, works perfectly, the program works, everything is easy and simple. comes with a ba...
Tokens:
['ordered', '3', 'keyfob', 'works', 'perfectly', 'the', 'program', 'works', 'everything', 'is']...

File: textual_data_03.txt
Lowercased Text:
this is my second suitcase i buy anti-theft style, it only has two pockets, a hole for the headphone...
Tokens:
['this', 'is', 'my', 'second', 'suitcase', 'i', 'buy', 'anti-theft', 'style', 'it']...
```

After processing all files, the script print

- The first 100 characters of lowercased text
- The first 10 tokens for the first 3 files

Calls the process_all_files function, passing the input and output folder paths.

c)Text Vectorization

i.Bag of words

```
c) Text vectorization

i. Bag of words

import os
from sklearn.feature_extraction.text import CountVectorizer

def bag_of_words_vectorization(input_folder):
    # Load all the preprocessed text files
    documents = [] # List to store all text content
    file_names = [] # List to keep track of file names

    # Process files in sorted order
    for filename in sorted(os.listdir(input_folder)):
        if filename.endswith(".txt"): # Only process text files
            file_path = os.path.join(input_folder, filename)
            with open(file_path, 'r', encoding='utf-8') as file:
                file_content = file.read().strip() # Read and remove leading/trailing whitespace
            if file_content: # Check if the file is not empty
                documents.append(file_content) # Add content to the list
                file_names.append(filename) # Track file names
```

documents is the list to store the content of each text file.

file_names is a list to track file names.

After ensuring files are processed in order the content are read, white spaces are removed and appended it to the documents.

```
# Bag of Words (BoW) Vectorization
print("\n--- Bag of Words (BoW) Representation ---")
bow_vectorizer = CountVectorizer() # Initialize the BoW vectorizer
bow_matrix = bow_vectorizer.fit_transform(documents) # Learn vocabulary and transform text

# Print the shape of the BoW matrix
print(f"BoW Matrix Shape: {bow_matrix.shape}")
total_words = len(bow_vectorizer.vocabulary_)
print(f"Total number of unique words in the vocabulary: {total_words}")

# Print the vocabulary (words and their corresponding indices)
print("\nVocabulary (All words and their indices):")
for word, index in bow_vectorizer.vocabulary_.items():
    print(f"{word}: {index}")
```

CountVectorizer converts text data into sparse matrix of words and it tokenizes text, builds a vocabulary and counts the frequency of each word.

fit_transform learns vocabulary and transforms the documents list into the BoW matrix.

Matrix Shape shows the BoW matrix dimensions (number_of_documents, unique_words)

Vocabulary is the total number of unique words in all files combined.

Then the code prints each word in the vocabulary and it's assigned an index

Displaying a sample BoW Vector

```
# Display a sample document's BoW representation
print("\n--- Sample BoW Representation for First Document ---")
sample_bow = bow_matrix[0].toarray() # Convert first row to array
print(f"File: {file_names[0]}")
print(f"BoW Vector:\n{sample_bow}")

# Define the folder containing preprocessed text files
input_folder = "/content/processed_customer_reviews" # Preprocessed folder path

# Perform Bag of Words vectorization
bag_of_words_vectorization(input_folder)
```

`Bow_matrix[0]` extracts the BoW vector of the first document

`.toarray()` converts the sparse matrix row into a standard array for easy readability.

Then it prints the file name of the document and its corresponding Bag of Words vector

```
--- Bag of Words (BoW) Representation ---
BoW Matrix Shape: (50, 1240)
Total number of unique words in the vocabulary: 1240

Vocabulary (All words and their indices):
overall: 808
evaluation: 365
very: 1170
good: 458
fabric: 385
material: 691
comfortable: 245
without: 1221
being: 138
heavy: 492
style: 1054
classic: 223
and: 79
timeless: 1119
comfort: 244
level: 649
extremely: 383
it: 605
can: 185
be: 130
considered: 259
great: 463
shopping: 984
experience: 381
ordered: 802
keyfob: 622
works: 1226
perfectly: 822
the: 1096
```

ii. Term frequency-inverse document frequency

```
ii. Term frequency-inverse document frequency

from sklearn.feature_extraction.text import TfidfVectorizer
import os

def tfidf_vectorization(input_folder):
    # Loading preprocessed text files
    documents = [] # List to store all text
    file_names = [] # List to keep track of file names
```

TfidfVectorizer converts text data into a TF-IDF matrix. os is for file and directory handing.

documents is list to store content of each text file

file_names is list to store the corresponding file names.

```
for filename in sorted(os.listdir(input_folder)):
    if filename.endswith(".txt"): # Process text files only
        file_path = os.path.join(input_folder, filename)
        with open(file_path, 'r', encoding='utf-8') as file:
            documents.append(file.read()) # Add content to the list
            file_names.append(filename)
```

Loop reads each .txt file in sorted order. documents appends content of each file. file_names stores the file names for references

TF – IDF Vectorization

```
# TF-IDF Vectorization
print("\n--- Term Frequency-Inverse Document Frequency (TF-IDF) Representation ---")
tfidf_vectorizer = TfidfVectorizer()
tfidf_matrix = tfidf_vectorizer.fit_transform(documents) # Transform text into numerics
print(f"TF-IDF Matrix Shape: {tfidf_matrix.shape}")

# Get feature names and their corresponding indices
feature_names = tfidf_vectorizer.get_feature_names_out()

# Get vocabulary and corresponding term frequencies
vocabulary = tfidf_vectorizer.vocabulary_
terms = list(vocabulary.keys())
```

The vectorizer is initialized and TF-IDF matrix is generated and the shape of it is displayed(number of documents, unique words).

feature_names is a list of words in the vocabulary.

vocabulary_ is a dictionary of words as keys and their corresponding indices as values.
vocabulary contains keys and their indices.

terms extracts the list of words from the vocabulary.

```

# Output term frequencies
formatted_words = [f"{word:<15}: {tfidf_matrix[:, vocabulary[word]].sum():>5}" for word in terms]

# Print the formatted list of terms with their total term frequencies
for formatted_word in formatted_words:
    print(formatted_word)

return tfidf_matrix, tfidf_vectorizer

# Run the TF-IDF vectorization
input_folder = "/content/processed_customer_reviews" # Folder containing preprocessed reviews
tfidf_matrix, tfidf_vectorizer = tfidf_vectorization(input_folder)

```

formatted_words calculates the sum of TF – IDF scores for each term across all documents.

‘tfidf_matrix[:, vocabulary[word]].sum()’ sums the TF – IDF values for a specific word.

Then each word is printed along with its total TF – IDF score across all documents.

the function is executed with the folder containing the preprocessed reviews.

```

--- Term Frequency-Inverse Document Frequency (TF-IDF) Representation ---
TF-IDF Matrix Shape: (50, 1240)
overall      : 0.4235042348824531
evaluation   : 0.2308165853221446
very         : 2.5752067179497957
good         : 1.7731392865925868
fabric       : 0.4082550466418571
material     : 0.8862169553424033
comfortable  : 0.6946812076598352
without      : 0.30016440010257606
being        : 0.27672069656797416
heavy        : 0.2308165853221446
style        : 0.38516008625341247
classic      : 0.2308165853221446
and          : 3.7850294081338425
timeless     : 0.2308165853221446
comfort      : 0.33832751619316315
level        : 0.2308165853221446
extremely    : 0.2308165853221446
it           : 4.273029432707462
can          : 0.7697583909062762
be           : 1.0250181243042982
considered   : 0.2308165853221446
great        : 0.6576513470738524
shopping     : 0.3619209853887607
experience   : 0.3446896413368218
ordered      : 0.3176702867004185
keyfob       : 0.20074651036788357
works        : 0.7320145588282638
perfectly    : 0.20074651036788357
the          : 6.357143209869836
program      : 0.20074651036788357
everything    : 0.6208215887822497
is           : 3.780782418357377

```

d) Metadata and labelling

```
d) Metadata and labelling

import os
import json
from sklearn.feature_extraction.text import CountVectorizer
from textblob import TextBlob # For sentiment analysis

def create_metadata_and_labeling(input_folder, output_metadata_file):
    """
    Function to create metadata for preprocessed text files.
    - Performs word count.
    - Assigns sentiment labels (positive, negative, neutral).
    - Includes the top 5 words for each document based on frequency.

    Args:
        input_folder (str): Path to folder containing preprocessed text files.
        output_metadata_file (str): Path to output JSON metadata file.
    """

    documents = [] # To store text content for vectorization
    metadata = [] # To store metadata for each file
    file_names = [] # To track file names
```

The function `creat_metadata_and_labeling` performs three tasks. It reads the preprocessed text files from the specified folder, analyze metadata including word count, sentiment analysis (positive, negative or neutral) and the top five words and also saves metadata for each file into a JSON file.

Loading Processed Text Files

```
[ ] # Load all preprocessed text files and perform sentiment analysis
print("Loading files and performing sentiment analysis...\n")
for filename in sorted(os.listdir(input_folder)):
    if filename.endswith(".txt"): # Process only .txt files
        file_path = os.path.join(input_folder, filename)
        with open(file_path, 'r', encoding='utf-8') as file:
            file_content = file.read().strip() # Read and strip whitespace

        if file_content: # Skip empty files
            # Add text content for vectorization
            documents.append(file_content)
            file_names.append(filename)

            # Sentiment analysis using TextBlob
            sentiment_polarity = TextBlob(file_content).sentiment.polarity
            if sentiment_polarity > 0:
                sentiment_label = "Positive"
            elif sentiment_polarity < 0:
                sentiment_label = "Negative"
            else:
                sentiment_label = "Neutral"

            # Add metadata for the current file
            metadata.append({
                "file_name": filename,
                "word_count": len(file_content.split()), # Total words in the document
                "sentiment_label": sentiment_label,
                "top_words": None # Placeholder for now
            })
```

The function iterates through all .txt files in the `input_folder`.

It reads the file contents, skip empty files and :

- Counts the total words using `len(file_content.split())`.
- Calculates setiment polarity using TextBlob :
 - positive polarity – “Positive”
 - negative polarity – “Negative”

- Zero polarity – “Neutral”
- And adds this metadata to a list.

The documents are BoW vectorized and top 5 words are extracted.

```
[ ]
# Vectorize the documents using Bag of Words
print("Performing Bag of Words (BoW) vectorization...\n")
vectorizer = CountVectorizer()
bow_matrix = vectorizer.fit_transform(documents) # Transform documents into BoW matrix
vocab = vectorizer.get_feature_names_out() # Extract vocabulary

# Extract top 5 words for each document
print("Extracting top words for each document...\n")
for i, doc_vector in enumerate(bow_matrix):
    word_counts = doc_vector.toarray().flatten() # Flatten the vector for current document
    top_word_indices = word_counts.argsort()[-5:][::-1] # Get indices of top 5 words (sorted)
    top_words = [vocab[idx] for idx in top_word_indices if word_counts[idx] > 0] # Filter non-zero words
    metadata[i]["top_words"] = top_words # Update top words in metadata

# Save metadata to a JSON file
with open(output_metadata_file, 'w', encoding='utf-8') as json_file:
    json.dump(metadata, json_file, indent=4)

print(f"Metadata successfully saved to '{output_metadata_file}'!")
print(f"Processed {len(metadata)} files.")

# Define input and output paths
input_folder = "/content/processed_customer_reviews" # Path to preprocessed text files
output_metadata_file = "text_metadata.json" # Output JSON file for metadata

# Call the function to create metadata
create_metadata_and_labelling(input_folder, output_metadata_file)
```

Saves the metadata for all processed files as a formatted JSON file (output_metadata_file) with details for file:

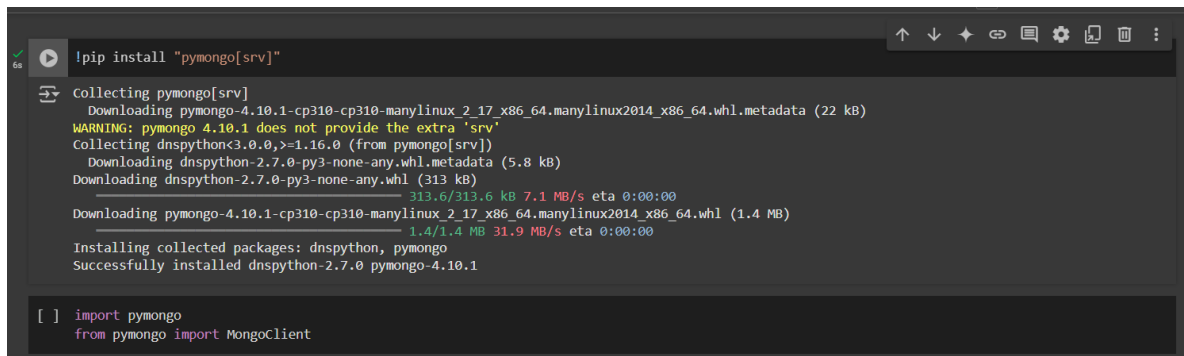
- file name
- word count
- sentiment label
- top 5 words

input_folder contains preprocessed text files.

output_metadata_file is where the metadata will be saved.

create_metadata_and_labelling is called to execute the pipeline.

e) Database creation



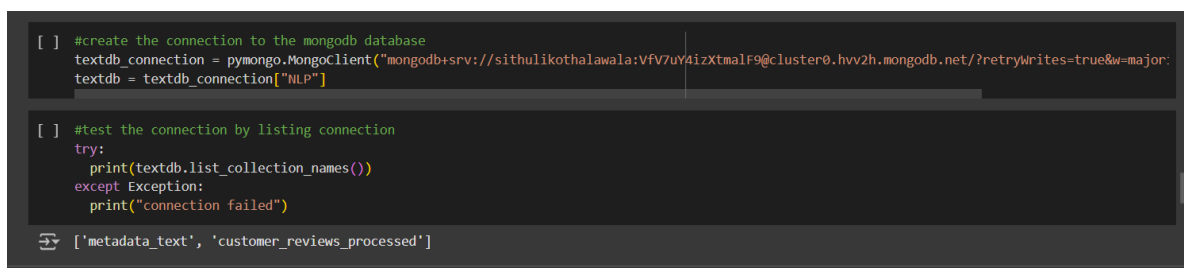
```
!pip install "pymongo[srv]"

Collecting pymongo[srv]
  Downloading pymongo-4.10.1-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (22 kB)
  WARNING: pymongo 4.10.1 does not provide the extra 'srv'
  Collecting dnspython<3.0.0,>=1.16.0 (from pymongo[srv])
  Downloading dnspython-2.7.0-py3-none-any.whl.metadata (5.8 kB)
  Downloading dnspython-2.7.0-py3-none-any.whl (313 kB)
    313.6/313.6 kB 7.1 MB/s eta 0:00:00
  Downloading pymongo-4.10.1-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (1.4 MB)
    1.4/1.4 MB 31.9 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
```

‘pymongo’ library is installed which is used to interact with MongoDB databases.

‘pymongo and MongoClient libraries are imported which are required to establish a connection to a MongoDB database and perform operations.



```
[ ] #create the connection to the mongodb database
    textdb_connection = pymongo.MongoClient("mongodb+srv://sithulikothalawala:VfV7uY4izXtmaIf9@cluster0.hvv2h.mongodb.net/?retrywrites=true&w=major:
    textdb = textdb_connection["NLP"]

[ ] #test the connection by listing connection
    try:
        print(textdb.list_collection_names())
    except Exception:
        print("connection failed")

[ ] ['metadata_text', 'customer_reviews_processed']
```

A connection is created to MongoDB database using the connection string

‘textdb’ refers to the database within the MongoDB cluster called NLP.

Then the connection to the database is tested by listing the names of all connections in the NLP database.

Inserting text metadata into metadata_text collection

```
[ ] #load text metadata to text metadata collection

#Defining the collection
text_metadata_collection = textdb["metadata_text"]

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

#inserting data into MongoDB collection
if isinstance(customer_reviews_data, list):
    text_metadata_collection.insert_many(customer_reviews_data)
else:
    text_metadata_collection.insert_one(customer_reviews_data)

[ ] print("text metadata inserted successfully")

text metadata inserted successfully
```

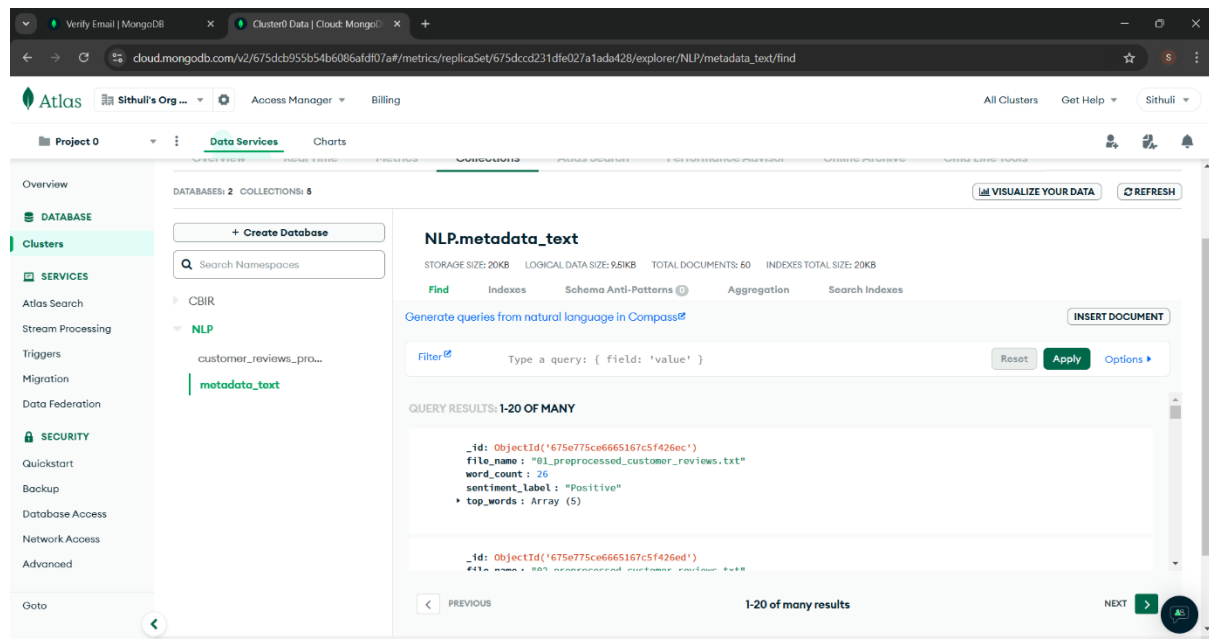
text_metadata_collection accesses the 'metadata_text' collection in the database. Then the text_metadata.json file is read into customer_review_data.

Then the loaded JSON data is inserted into "metadata_text" collection in MongoDB.

If customer_reviews_data is a list the insert_many method is used to insert all documents at once.

Otherwise, if it is a single document the insert_one method is used.

Following is how the metadata is inserted in metadata_text collection.



Inserting processed_customer_reviews into the collection

```
[ ] processed_text_doc = textdb["customer_reviews_processed"]

processed_folder_location = "/content/processed_customer_reviews"

from bson import Binary

for filename in os.listdir(processed_folder_location):
    if filename.endswith(".txt"):
        with open(os.path.join(processed_folder_location, filename), "rb") as text_file:
            binary_text = Binary(text_file.read())
            processed_text_doc.insert_one({"filename": filename, "text": binary_text})

print("processed text documents inserted successfully")
```

processed text documents inserted successfully

customer_reviews_processed is the collection where text files will be stored.

processed_folder_location is where preprocessed .txt files are stored

os.listdir() lists all files in processed_folder_location

if condition ensures only .txt files are processed.

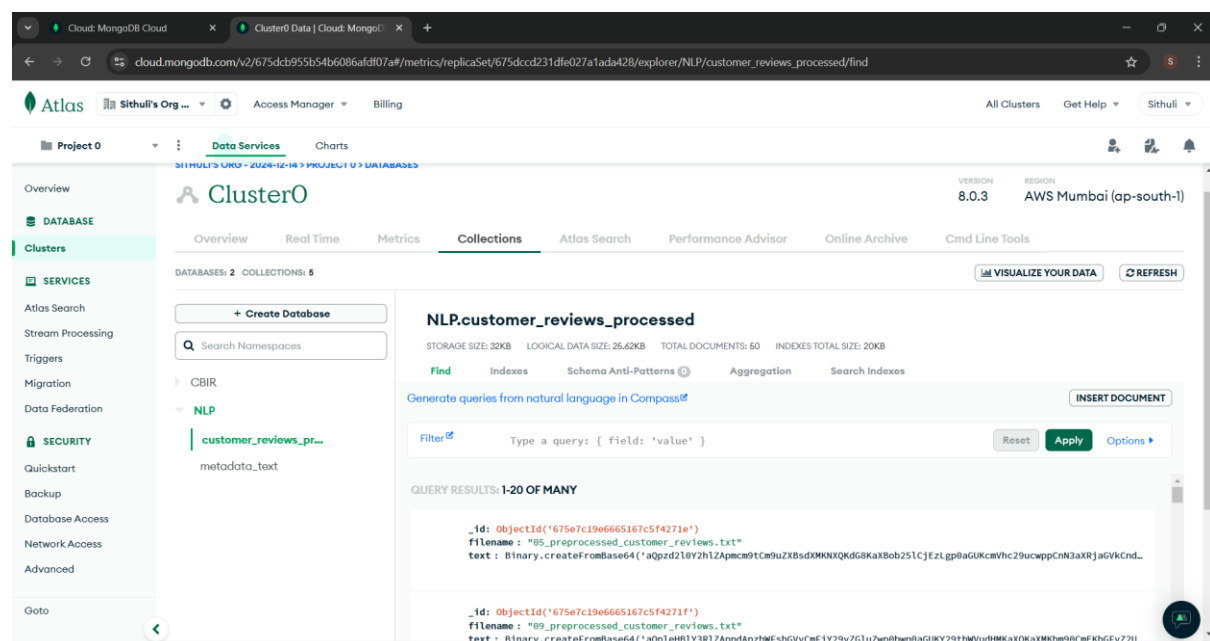
The text file is opened in binary read mode

Binary() converts content into binary format to ensure the data can be stored in MongoDB.

insert_one() inserts a single document into the MongoDB collection

When files are inserted a success message is printed.

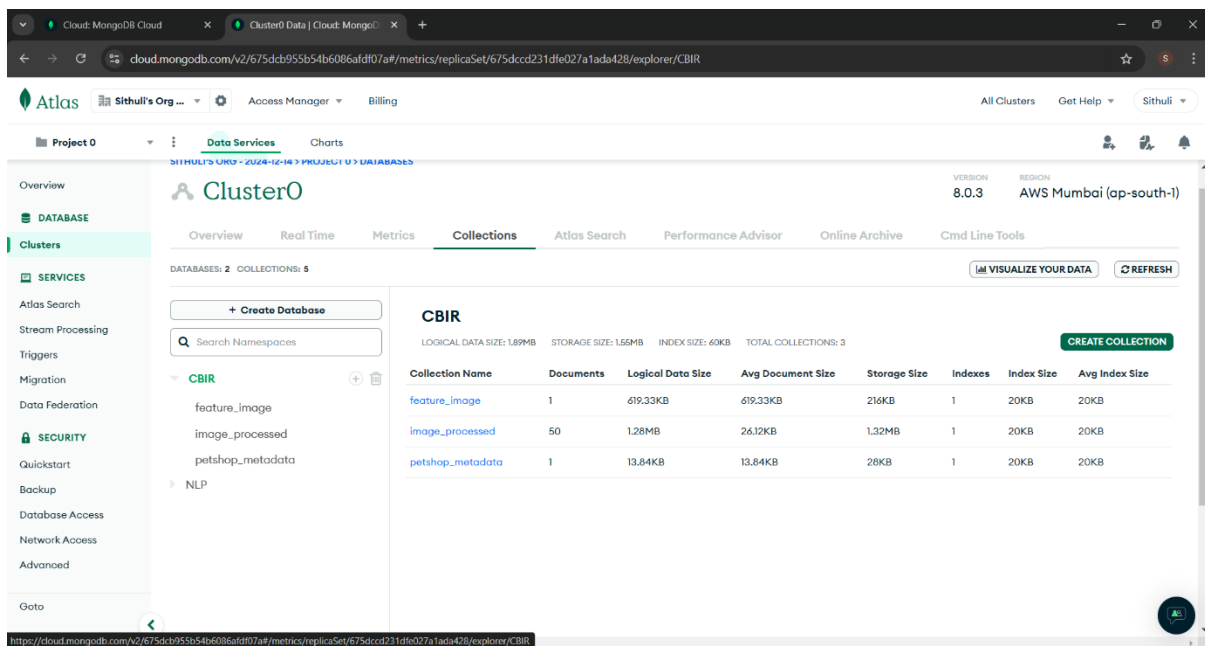
Below is how it is inserted in MongoDB



Storage and retrieval

Database for content-based image retrieval system (CBIR)

The CBIR database in MongoDB Atlas is used for managing image-related data for tasks like Content-Based Image Retrieval. It has three collections: `feature_image` for image features or metadata, `image_processed` for processed image data and `petshop_metadata` for image descriptions or labels. The largest collection, `image_processed`, contain 50 documents reach averaging about 26KB, while all collections have small 20KB indexes for efficient searches.



The screenshot shows the MongoDB Atlas interface for a ClusterO database. The 'Collections' tab is selected, displaying a table of collections for the 'CBIR' database. The table includes columns for Collection Name, Documents, Logical Data Size, Avg Document Size, Storage Size, Indexes, Index Size, and Avg Index Size. The collections listed are `feature_image`, `image_processed`, and `petshop_metadata`.

Collection Name	Documents	Logical Data Size	Avg Document Size	Storage Size	Indexes	Index Size	Avg Index Size
<code>feature_image</code>	1	619.33KB	619.33KB	216KB	1	20KB	20KB
<code>image_processed</code>	50	1.28MB	26.12KB	1.32MB	1	20KB	20KB
<code>petshop_metadata</code>	1	13.84KB	13.84KB	28KB	1	20KB	20KB

Cloud: MongoDB Cloud Cluster0 Data | Cloud: Mongo...

cloud.mongodb.com/v2/675dcb955b54b6086afdf07a#/metrics/replicaSet/675dcd231dfe027a1ada428/explorer/CBIR/feature_image/find

Atlas | Sithuli's Org ... | Access Manager | Billing

All Clusters | Get Help | Sithuli

Project 0 | Data Services | Charts

Overview | DATABASE | Clusters | SERVICES | Atlas Search | Stream Processing | Triggers | Migration | Data Federation | SECURITY | Quickstart | Backup | Database Access | Network Access | Advanced | Goto

Cluster0

Overview | Real Time | Metrics | Collections | Atlas Search | Performance Advisor | Online Archive | Cmd Line Tools

DATABASES: 2 | COLLECTIONS: 5

+ Create Database

Search Namespaces

CBIR

feature_image

image_processed

petshop_metadata

NLP

CBIR.feature_image

STORAGE SIZE: 26KB | LOGICAL DATA SIZE: 49.33KB | TOTAL DOCUMENTS: 1 | INDEXES TOTAL SIZE: 20KB

Find | Indexes | Schema Anti-Patterns | Aggregation | Search Indexes

Generate queries from natural language in Compass

INSERT DOCUMENT

Filter | Type a query: { field: 'value' } | Reset | Apply | Options

QUERY RESULTS: 1-OF 1

```
{
  "_id": ObjectId("675d4f4a393df6a1e2a854318"),
  "image_001.jpg": Object,
  "image_002.jpg": Object,
  "image_003.jpg": Object,
  "image_004.jpg": Object,
  "image_005.jpg": Object,
  "image_006.jpg": Object,
  "image_007.jpg": Object,
  "image_008.jpg": Object
}
```

Cloud: MongoDB Cloud Cluster0 Data | Cloud: Mongo...

cloud.mongodb.com/v2/675dcb955b54b6086afdf07a#/metrics/replicaSet/675dcd231dfe027a1ada428/explorer/CBIR/image_processed/find

Atlas | Sithuli's Org ... | Access Manager | Billing

All Clusters | Get Help | Sithuli

Project 0 | Data Services | Charts

Overview | DATABASE | Clusters | SERVICES | Atlas Search | Stream Processing | Triggers | Migration | Data Federation | SECURITY | Quickstart | Backup | Database Access | Network Access | Advanced | Goto

Cluster0

Overview | Real Time | Metrics | Collections | Atlas Search | Performance Advisor | Online Archive | Cmd Line Tools

DATABASES: 2 | COLLECTIONS: 5

+ Create Database

Search Namespaces

CBIR

feature_image

image_processed

petshop_metadata

NLP

CBIR.image_processed

STORAGE SIZE: 1.32MB | LOGICAL DATA SIZE: 1.28MB | TOTAL DOCUMENTS: 50 | INDEXES TOTAL SIZE: 20KB

Find | Indexes | Schema Anti-Patterns | Aggregation | Search Indexes

Generate queries from natural language in Compass

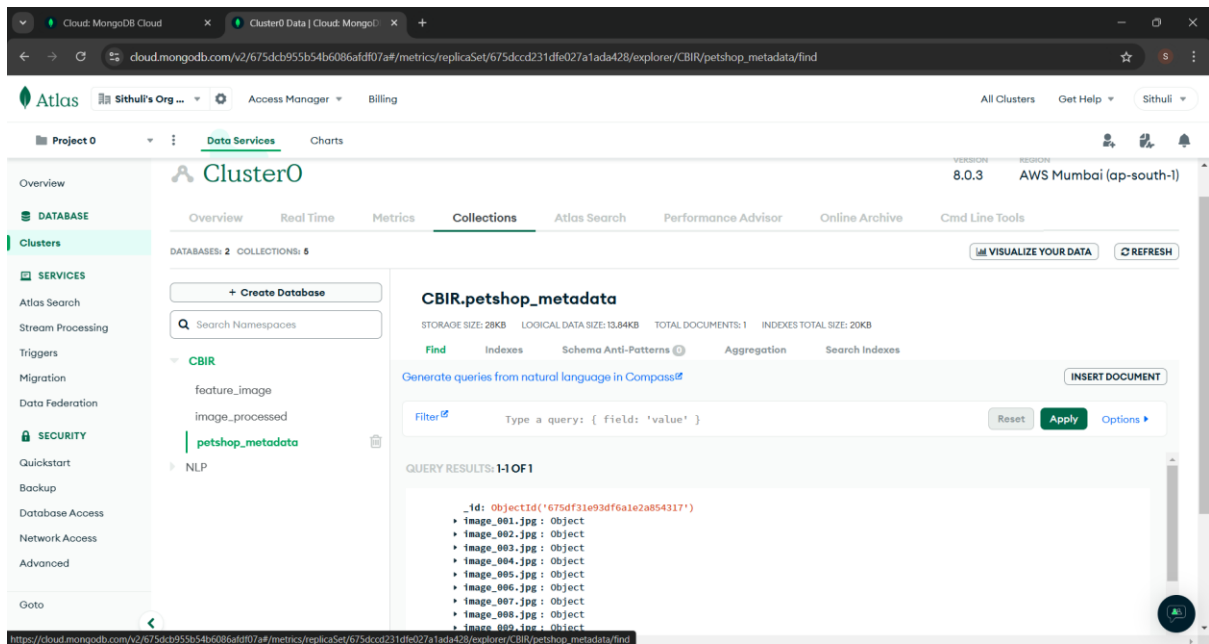
INSERT DOCUMENT

Filter | Type a query: { field: 'value' } | Reset | Apply | Options

QUERY RESULTS: 1-20 OF MANY

```
{
  "_id": ObjectId("675e6cf34dfd46996ca77f35"),
  "filename": "image_017.jpg",
  "image": Binary.createFromBase64(' /9j /4AAQSkZJRgABAQAAQABAAQ /gA7Q1JFQVRPUjogZ2QtanB1ZyB2M54wICh1c2luZyB3SkcgS1BFb20T...
}

{
  "_id": ObjectId("675e6cf44dfd46996ca77f36"),
  "filename": "image_007.jpg",
  "image": Binary.createFromBase64(' /9j /4AAQSkZJRgABAQAAQABAAQ /2wBDAAGCBgcGBQgHBwcJCQgKDBQNDAsLDBkSEwUHR0FHh6aHBwgJC4nIC...
```



Querying a specific Image by Filename.

```
[14] # Query an image to check if the processed images were loaded to the database
query = {"filename": "image_006.jpg"}
result = image_processed_collection.find_one(query)

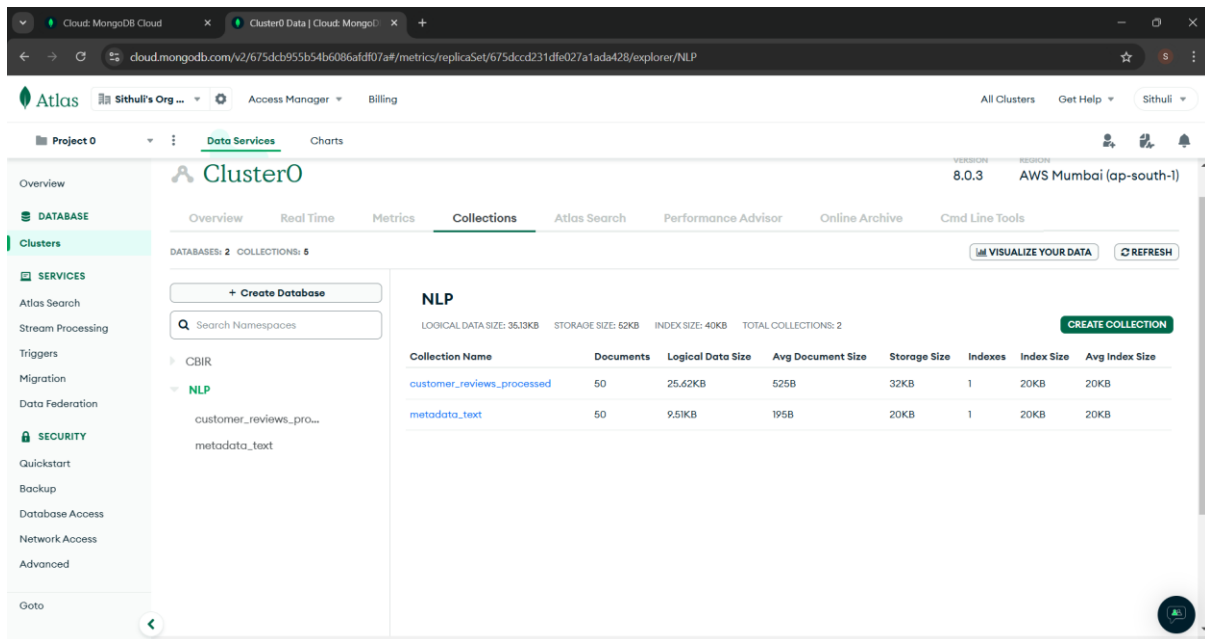
# Print the result
if result:
    print("Image found:", result["filename"])
else:
    print("Image not found in the database.")

Image found: image_006.jpg
```

Queries the image processed collection for a document where the filename is 'image_006.jpg'. If the document is found prints the filename otherwise prints "Image not found".

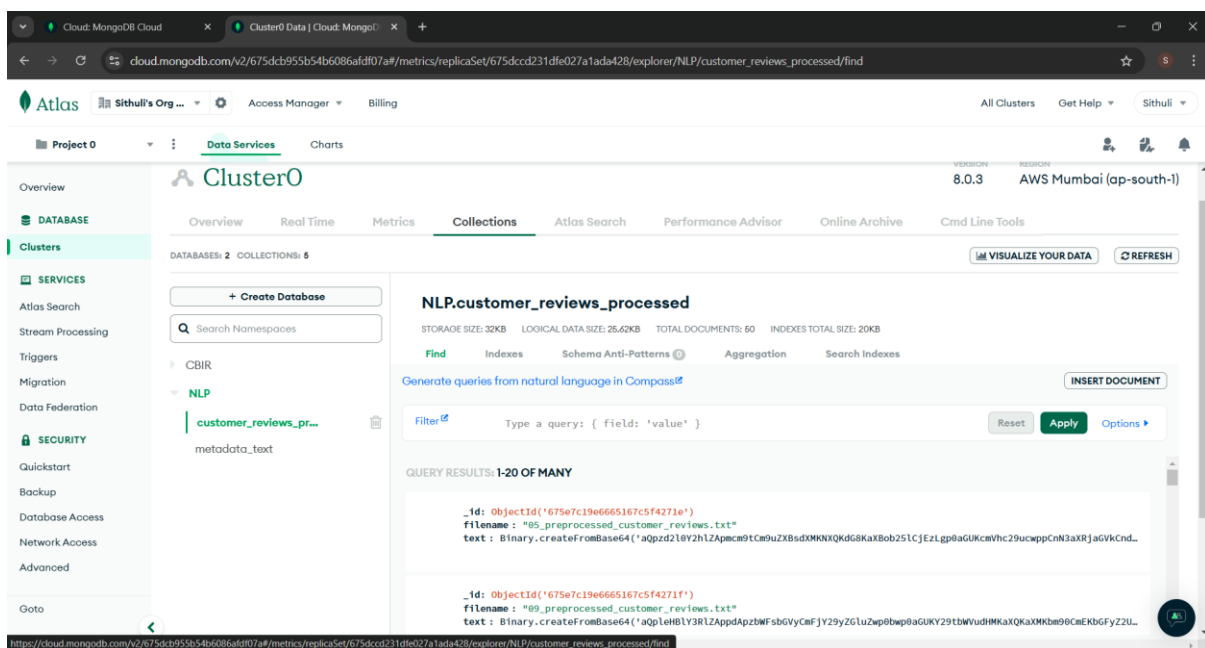
Database for sentiment analysis models

The NLP database handles text data for Natural Language Processing tasks. It has two collections: `customer_reviews_processed`, which stores cleaned and preprocessed customer reviews, and `metadata_text`, which contains metadata like word counts, sentimental labels or important words. Both collections are small, with 50 documents each, and their compact size reflects the nature of text data. Each collection also includes a small index for faster access and queries.



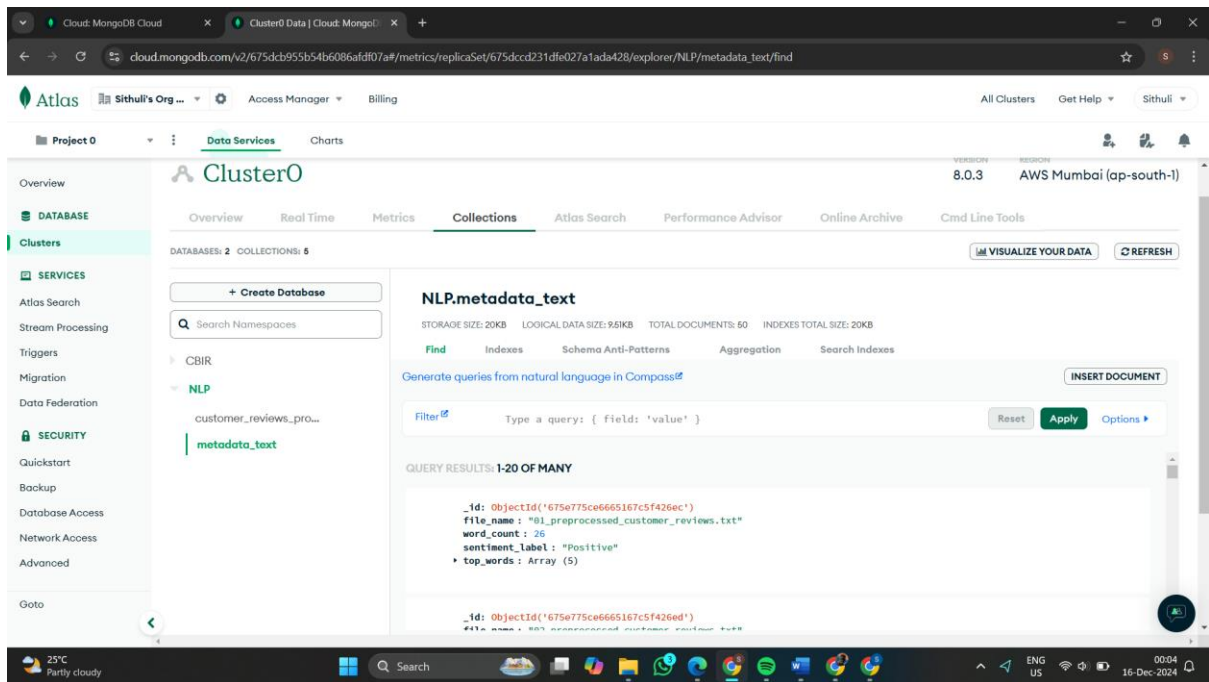
The screenshot shows the MongoDB Atlas interface for a Cluster0 instance. The 'Collections' tab is selected, displaying a table of collections within the 'NLP' database. The table has columns for Collection Name, Documents, Logical Data Size, Avg Document Size, Storage Size, Indexes, Index Size, and Avg Index Size. Two collections are listed: 'customer_reviews_processed' and 'metadata_text', both containing 50 documents.

Collection Name	Documents	Logical Data Size	Avg Document Size	Storage Size	Indexes	Index Size	Avg Index Size
customer_reviews_processed	50	25.62KB	525B	32KB	1	20KB	20KB
metadata_text	50	9.51KB	195B	20KB	1	20KB	20KB

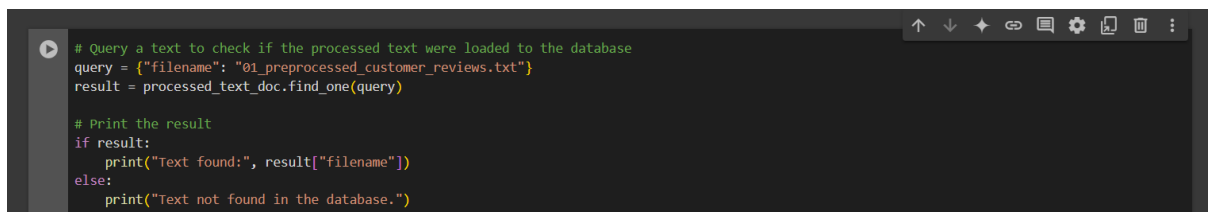


The screenshot shows the MongoDB Atlas interface for the 'NLP.customer_reviews_processed' collection. The 'Find' tab is selected, displaying a query editor and the results of a query. The query results show two documents, each with an '_id', 'filename', and 'text' field. The 'text' field contains a base64-encoded string.

```
{ "_id": "675e7c19e6665167c5f4271e", "filename": "B5_preprocessed_customer_reviews.txt", "text": "Binary.createFromBase64('aQpzd2l0Y2h1ZApwcm9tcm9uZXBsdXMKNXQ6dG8KaX8ob251c1JlZlgsbGUKcmVhc29ucwppcnN3aXRjaGVkCnd..." }
{ "_id": "675e7c19e6665167c5f4271f", "filename": "B9_preprocessed_customer_reviews.txt", "text": "Binary.createFromBase64('aQplehBY3RlZApwcm9tcm9uZXBsdXMKNXQ6dG8KaX8ob251c1JlZlgsbGUKcmVhc29ucwppcnN3aXRjaGVkCnd..." }
```



Querying a specific customer review by file name.



Queries the processed text collection for a document where the filename is '01_preprocessed_customer_reviews'

If the document is found prints the filename otherwise prints "Text not found".

MongoDB username and password

sithulikothalawala

VfV7uY4izXtmalF9

References

Vicky (2019). *Restaurant Customer Reviews*. [online] Kaggle.com. Available at: <https://www.kaggle.com/datasets/vigneshwarsofficial/reviews> [Accessed 12 Dec. 2024].

Mars_1010 (2023). *Apple iPhone Customer Reviews*. [online] Kaggle.com. Available at: <https://www.kaggle.com/datasets/mrmars1010/iphone-customer-reviews-nlp> [Accessed 12 Dec. 2024].

Lakshmipathi N (2019). *IMDB Dataset of 50K Movie Reviews*. [online] Kaggle.com. Available at: <https://www.kaggle.com/datasets/lakshmi25npathi/imdb-dataset-of-50k-movie-reviews> [Accessed 12 Dec. 2024].

Hatab, M. (2023). *Hotel Reviews Booking.com*. [online] Kaggle.com. Available at: <https://www.kaggle.com/datasets/michelhatab/hotel-reviews-bookingcom> [Accessed 12 Dec. 2024].

<https://www.facebook.com/thesprucepets> (2024). *Learn All About Pet Tortoises*. [online] The Spruce Pets. Available at: <https://www.thesprucepets.com/tortoises-as-pets-1237258> [Accessed 10 Dec. 2024].

Thegoodypet.com. (2024). *The Goody Pet - Because Every Pet Deserves More*. [online] Available at: <https://www.thegoodypet.com/> [Accessed 15 Dec. 2024].

Pinterest. (2023). *Pinterest*. [online] Available at: <https://www.pinterest.com/> [Accessed 15 Dec. 2024].

Reference - to, C. (2009). *breed of rabbit*. [online] Wikipedia.org. Available at: <https://en.wikipedia.org/wiki/Thrianta> [Accessed 12 Dec. 2024].

Reference – Petassure.com. (2018). *Loveable Lovebirds!* [online] Available at: <https://www.petassure.com/new-newsletters/loveable-lovebirds/> [Accessed 12 Dec. 2024].

