5DATA005W DATA ENGINEERING COURSEWORK 2 Database documentation

1.Database for content-based image retrieval system

The purpose of this project is to design and implement two databases, one for content based image retrieval system (CBIR) and the other one for sentiment analysis models. Databases are used for organising storing and retrieving, in this case, unstructured data in an efficient manner.

The purpose of this database is to store, retrieve and analyse image data and metadata on nature, animals, summer and travel images. The image database supports content-based image retrieval (CBIR) by storing processed image features, keywords and metadata. It holds metadata of each image that has gone through processing methods (such as resizing, rotation and denoising) and holds mean intensity, norm intensity, texture features and shape features such as area and perimeter data. This database allows users to easily find image data, query and find images based on similar content (from keywords) on CBIR. The CBIR system is retrieves images based on content similarity, from keywords and extracted features, which is ideal when querying large image datasets.

My colour image collection focuses on the topics mentioned, sourced from google images, Pexels, Pixabay and Pinterest, showing multiple styles and diverse subjects.

https://www.pexels.com/search/animal/

https://pixabay.com/images/search/nature/

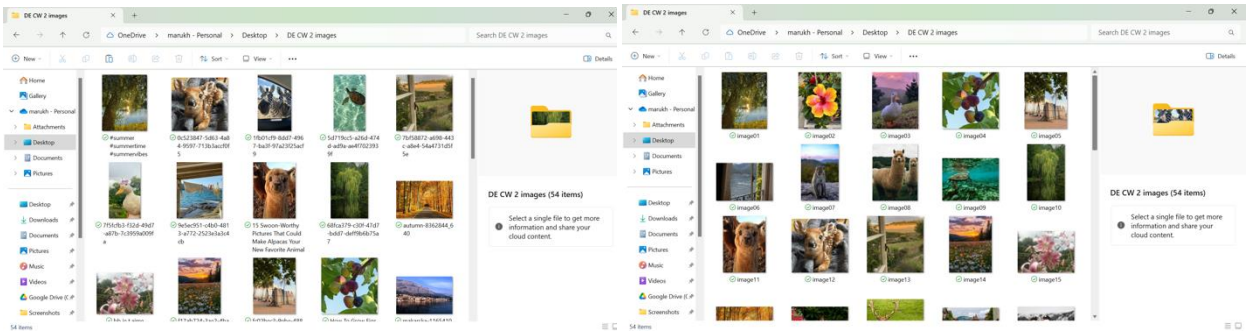
https://uk.pinterest.com/search/pins/?q=animals&rs=typed

Below is the link to my raw image files on google drive:

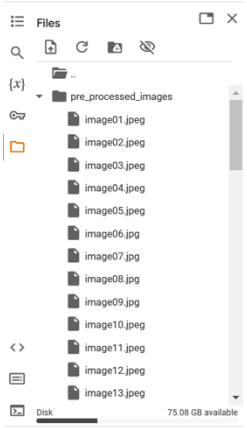
https://drive.google.com/drive/folders/1mqPSQVH_qyTNxObDjbmw7_u5rfegtMID?usp=drive_link

Raw text files:

https://drive.google.com/drive/folders/1Wmr1g-bn_1bW9B0n9o725-uO73OGey13?usp=drive_link

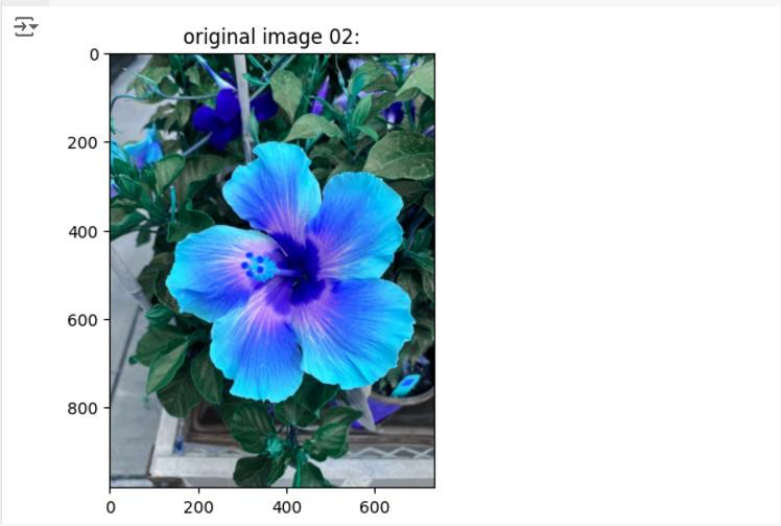


Firstly, I collected and saved all 54 colour preprocessed images into a folder on my desktop, manually systematically changed the name of each different image and then loaded them onto Google Colab into a folder called "pre_processed_images".

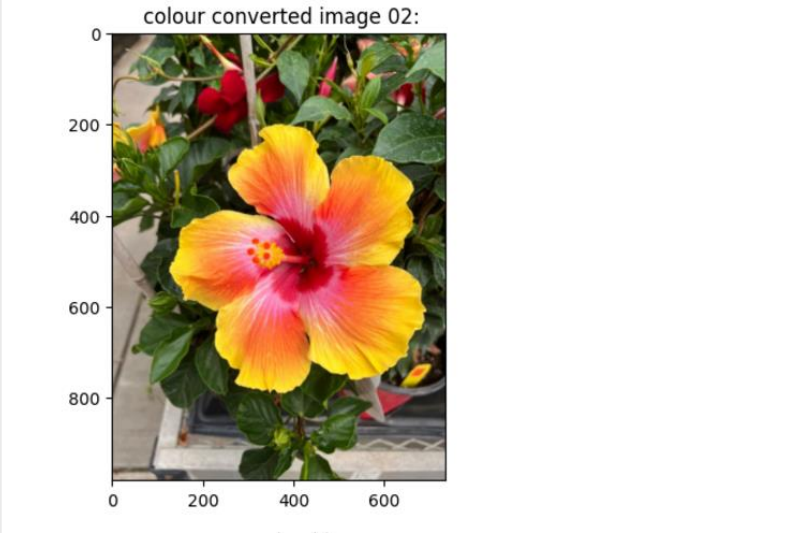


The first step in image pre-processing is converting the image from BGR (blue green red) format to RGB (red green blue) to ensure consistency in image colour. The preprocessing included resizing each image to a uniform size of 500x500 pixels, rotating some of the images by 90 degrees. Finally, Gaussian blur was applied which is putting a blur on each image to smooth the quality and denoise it. I then renamed the processed image and printed it, also saving them into a new seperate folder called processed_images. Below is an example of my code for image 2, printing the initial image and then going through step by step to show the output image.

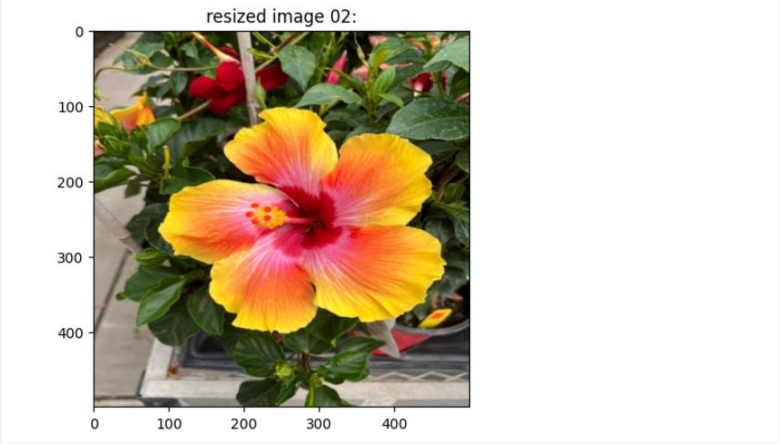
```
image_path ='/content/pre_processed_images/image02.jpeg'
image= cv2.imread(image_path)
plt.imshow(image)
plt.axis()
plt.title("original image 02:")
plt.show()
```



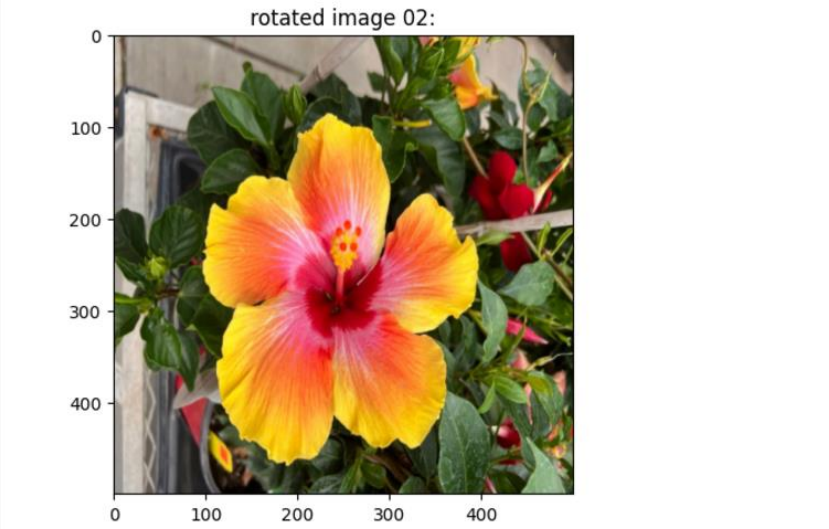

```
image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
plt.imshow(image_rgb)
plt.axis()
plt.title(" colour converted image 02:")
plt.show()
```



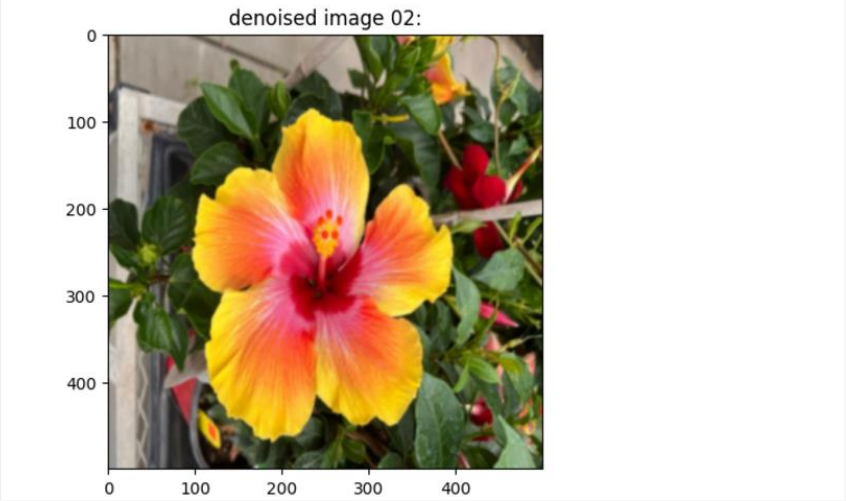

```
image_rgb = cv2.resize(image_rgb, (500, 500))
plt.imshow(image_rgb)
plt.axis()
plt.title("resized image 02:")
plt.show()
```




```
image_rgb= cv2.rotate(image_rgb, cv2.ROTATE_90_CLOCKWISE) #rotating image 90 degrees clockwise
plt.imshow(image_rgb)
plt.axis()
plt.title("rotated image 02:")
plt.show()
```




```
image_rgb = cv2.GaussianBlur(image_rgb, (5, 5), 0)
plt.imshow(image_rgb)
plt.axis()
plt.title("denoised image 02:")
plt.show()
```




```
processed_image_path = os.path.join(processed_folder, 'processed_image02.jpg')
cv2.imwrite(processed_image_path, cv2.cvtColor(image_rgb, cv2.COLOR_RGB2BGR))

#display the image using matplot lib
plt.imshow(image_rgb)
plt.axis()
plt.title("processed image 02:")
plt.show()
```

The next step is image annotation and creating metadata files of each image so the system can associate keywords and a description with each image, which can help with retrieval

and queries. This image info will serve as the basis for evaluating the CBIR system, which means that the system can then be used to retrieve similar images based on content from keywords. I went through each image and added various keywords and then provided a brief description on what was happening in the image. Below is an example of the types of keywords and descriptions I gave for the first 5 images:

```
[40] metadata= {
        "processed_image01.jpg":{
            "keywords": ["nature", "lake", "tree", "scenery"],
            "description": " A processed image of a tree overlooking a lake with resizing and denoising applied to image"
        },
        "processed_image02.jpg":{
            "keywords": ["flower", "nature", "hibiscus", "summer"],
            "description": " A processed image of a hibiscus flower with resizing, rotation and denoising applied to image"
        },

        "processed_image03.jpg":{
            "keywords": ["animals", "duck", "nature"],
            "description": "A processed image of a duck on a scenic hill with resizing and denoising applied to image"
            },

        "processed_image04.jpg":{
            "keywords": ["plant", "tree", "fig", "fruits", "nature"],
            "description": "A processed image of a fig tree with resizing and denoising applied to image"
        },
        "processed_image05.jpg":{
            "keywords": ["summer" "surfboards" "beach" "palm trees"],
            "description": "A processed image of surfboards at a beach  with resizing and denoising applied to image"
        },
```

The next step is to extract relevant feature images from the processed images. I used a for loop for this to efficiently calculate and iterate through all of the images instead of manually processing each one as it would take ages. It also helps with consistency and prevents errors. Feature extraction is finding characteristics of the images to help describe them and find similarities between images.

```
folder = '/content/processed_images'

# creating a for loop so it goes through all images in the folder to display
their features
for i in range(1, 55):  # for the 54 images
    file_name = f"processed_image{i:02}.jpg"  #i:02 =makes sure each file name
has two digits
    image_path = os.path.join(folder, file_name)  # creating the full path
```

I found the mean and the norm intensity, and these features were stored in a structured format . The mean of the channel pixel intensity represents the brightness of an overall image by calculating the mean intensity. The norm of pixel intensity shows the intensity and brightness of the pixels.

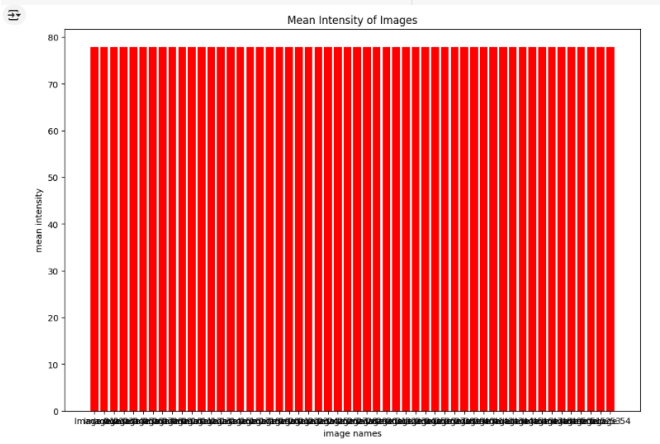
```
image = cv2.imread(image_path)
    mean_intensity = np.mean(image)
    norm_intensity = np.linalg.norm(image)
```

```
    print(f"the features for {file_name}:")
    print("mean intensity is:", mean_intensity)
    print("norm intensity is:", norm_intensity)
    print("")  #  this is just a gap between each image feature to make output
look a bit more neater
```

This is what the output looked like for the first few images:

```
the features for processed_image01.jpg:
mean intensity is: 76.327528
norm intensity is: 80233.37316852632

the features for processed_image02.jpg:
mean intensity is: 104.32922133333334
norm intensity is: 109909.2012617688

the features for processed_image03.jpg:
mean intensity is: 98.53565466666667
norm intensity is: 97090.76965911846

the features for processed_image04.jpg:
mean intensity is: 108.54818666666667
norm intensity is: 111549.14673810822

the features for processed_image05.jpg:
mean intensity is: 108.108516
norm intensity is: 111784.20067701876
```

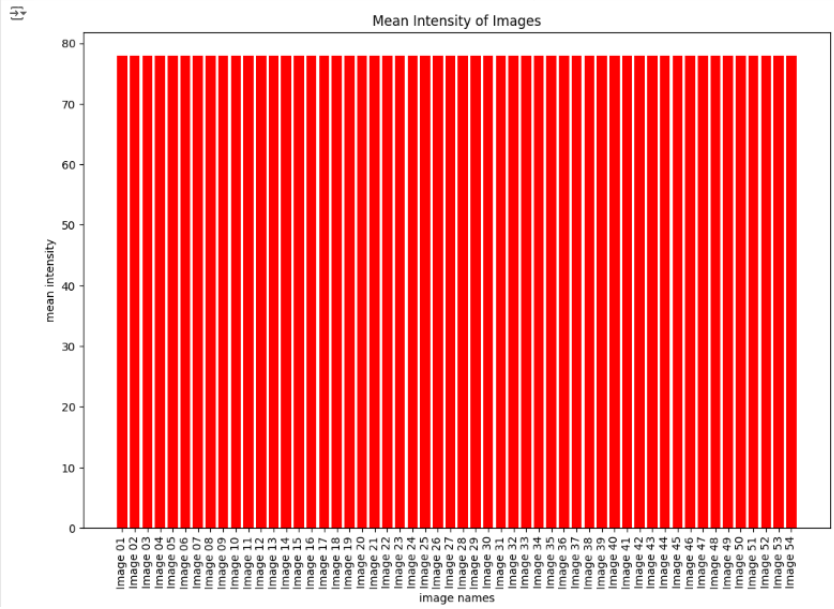
I included a bar chart plot to help visualization and to also show patterns. Initially this is what it looked like, with the x axis labels unreadable, so I made sure to rotate them for readability. Below is the initial bar chart, the code and the updated bar chart.




```
import matplotlib.pyplot as plt

plt.figure(figsize=(12, 8))
```

```
plt.bar([f"Image {i:02}" for i in range(1, 55)], mean_intensity, color='red')
#loops through
plt.xlabel('image names')
plt.ylabel('mean intensity')
plt.title('Mean Intensity of Images')
plt.xticks(rotation=90)
plt.show()
```

Output:



Shape features can be used to compare the structure of different images. Finding the area of the object is the total number of pixels in a boundary of the image and shows the measure of the image size, and the perimeter is the length of the images boundary. I used a for loop again and below is the code i used to find the shape features. I then stored it in a JSON file for retrieval.

```
# first converting the image to grayscale for shape feature extraction
gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

# applying Canny edge detection to find contours
edges = cv2.Canny(gray_image, 100, 200)

# finding contours from the edge detected image
contours, _ = cv2.findContours(edges, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

#iterating through each contour to find the the shape features
```

```
for contour in contours:
  area = cv2.contourArea(contour)
  perimeter = cv2.arcLength(contour, True)  # true means the contour is closed

        # printing the shape features
  print(f"area: {area}, perimeter: {perimeter}")
```

Output:

```
area: 3.0, perimeter: 9.656854152679443
area: 11.5, perimeter: 15.899494767189026
area: 5.0, perimeter: 31.798989415168762
area: 9.5, perimeter: 73.35533845424652
area: 8.0, perimeter: 39.455843687057495
area: 6.0, perimeter: 36.627416491508484
area: 9.0, perimeter: 40.28427064418793
area: 6.5, perimeter: 51.35533833503723
area: 4.5, perimeter: 38.870057106018066
area: 9.5, perimeter: 63.35533809661865
area: 1.5, perimeter: 61.899494767189026
area: 27.5, perimeter: 164.26702558994293
```

Texture descriptor is a feature that shows the texture properties of an image. Gray level co occurrence matrix (glcm) looks at how often the pixel intensities in an image occur. To do this for an image I first imported the required libraries and then defined and read the image. Next I converted the image to grayscale, where pixels have a value between 0-255. Then calculated the glcm to measure the relationship between pixel intensities, using the tutorial week 8 materials and script to guide me. Below is the code I used to extract texture descriptors:

```
    # convert image to grayscale
    gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

    # Compute GLCM and extract contrast
    glcm = graycomatrix(gray_image, distances=[1], angles=[0], levels=256,
symmetric=True, normed=True)
    contrast = graycoprops(glcm, 'contrast')[0, 0]

    print(f"Contrast Feature: {contrast}")
```

Output for the first few images:

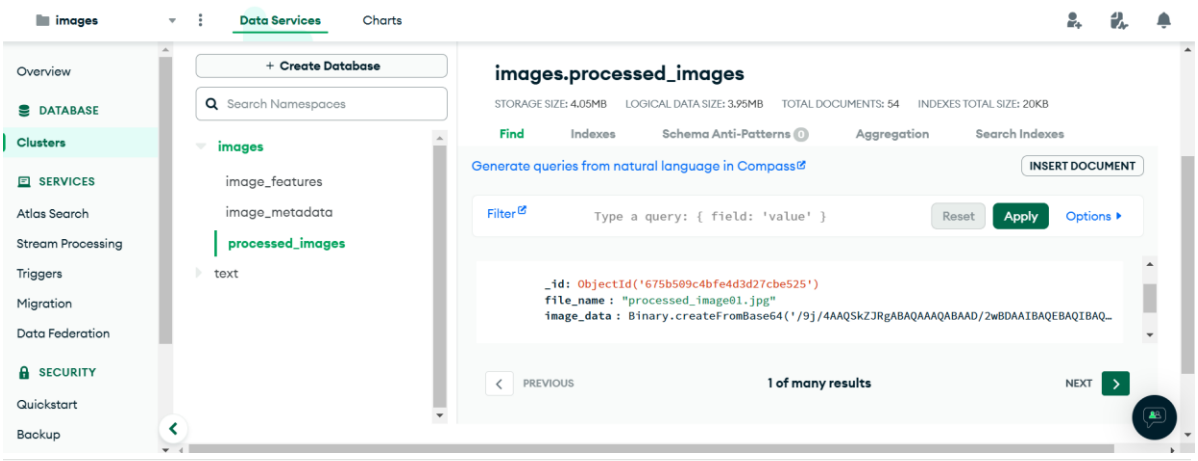
```
    print(f"Contrast Feature: {contrast}")
```

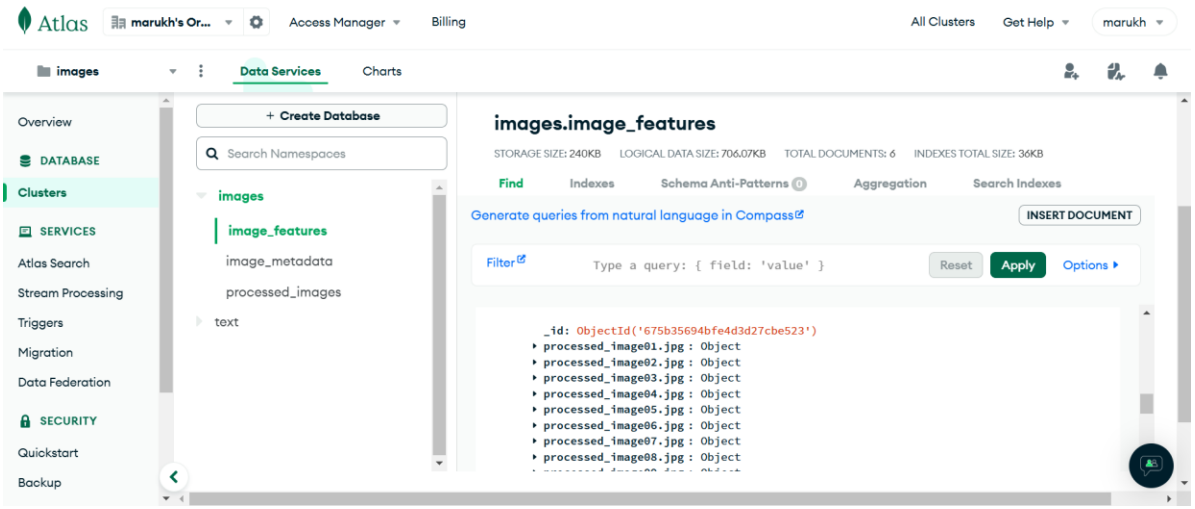
```
Contrast Feature: 65.0948136272545
Contrast Feature: 67.13771543086172
Contrast Feature: 29.82538677354709
Contrast Feature: 36.20655310621243
Contrast Feature: 86.08375551102206
Contrast Feature: 61.79270941883766
Contrast Feature: 13.59161523046092
Contrast Feature: 36.24282965931864
```

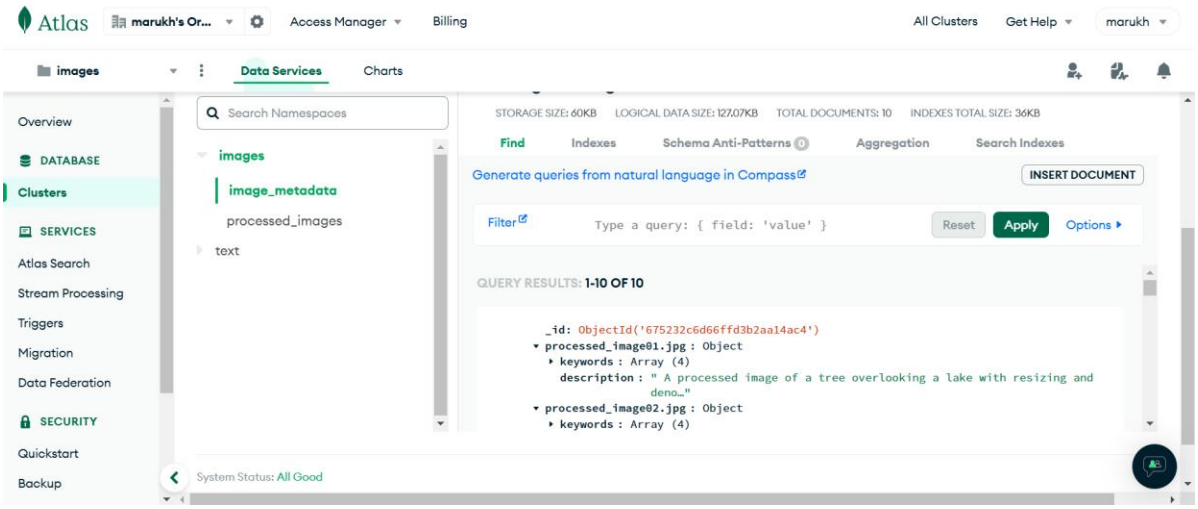
MongoDB store data that is unstructured such as images and texts, and is used because it can easily handle large datasets, is efficient in queries, is flexible and can store a variety of metadata. Firstly, I installed mongo DB, created a cluster that I'm going to be working in and created a image database and three collections to store the pre processed images, image features and the metadata. I then inserted these images and the json files onto Mongo db with my code

To check the database implementation, I ran some sample queries as shown below, which confirmed that my data had been loaded on.

When uploading my images to Mongo DB, my code loops through all my processed images, reads them as binary (rb) and then stores them into the MongoDB processed_images collection in the image database, where each document contains the image name and binary data- I used the week 9 tutorial resources to guide me. I went onto MongoDB to confirm the data has been loaded and the screenshots below show that MongoDB has successfully stored them.







2.Databases for sentiment analysis models

The purpose of this database is to collect textual data- I decided to collect data on customer service reviews on Apple, sourcing my reviews from Trustpilot and from X (links below) to get a diverse range of reviews- age, tone (positive, negative etc) and length wise. I collected these reviews and saved them as files, then saved them into a folder called "text_data"

https://uk.trustpilot.com/review/www.apple.co.uk

https://x.com/search?q=apple%20customer%20service&src=typed_query

After importing the necessary libraries and downloading stopwords, the next step was text preprocessing which is cleaning raw data for analysis. Text normalisation is converting the text to lowercase for consistency, and tokenisation is splitting the text into individual words, which makes the text more manageable when analysing. For some texts that were quite long, I removed stop words, which reduces data noise and focuses on important words. I also removed punctuation in some reviews as they didn't affect the statement and simplifies the text further.

I saved this processed text data as a txt file into a new folder using the with statement: the following website helped me.

https://www.statology.org/with-open-python/

Below is an example of the code I used:

```
with open('/content/text_data/review_text49.txt') as file:
  review = file.read()
  print("Review text 49:")
  print(review)
  print("-")
  review_lower= review.lower()
  tokens= word_tokenize(review_lower)
  stop_words= set(stopwords.words('english'))
tokens = [word for word in tokens if word not in stop_words]
print(tokens)

output_file = '/content/processed_text_data/processed_review_text49.txt'
with open(output_file, 'w') as output_file:
    output_file.write(' '.join(tokens))
```

Text vectorisation is where the text needs to be converted into numerical representations for machine learning models. I used tutorial resources to help me write my code.

Bag of words is used in nlp to convert text data into numerical data, so that it can be processed, it counts the number of times a word is in the document file (frequencies) and represents it in a vector form. Below is the code I used and the output:

```
corpus = [document_2]
print("Review_text_02")
vectorizer_bow = CountVectorizer()
bow_matrix = vectorizer_bow.fit_transform(corpus)
print("bag of words representation:\n", bow_matrix.toarray())
print("vocabulary:", vectorizer_bow.get_feature_names_out())
```

```
Review_text_03
bag of words representation:
 [[1 1 1 2 3 1 2 2 2 1 1 4 1 1 1 1 2 1 1 2 1 1 1 1 4 1 1 1 2 2 1 1 1 1 1 1
  1 1 2 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 3 1 1 6 1 2 2 1 1 2 3 2 2 1
  1 1 1 1 1 1]]
vocabulary: ['14' '2024' 'after' 'an' 'and' 'anything' 'apple' 'are' 'as' 'at' 'away'
 'bought' 'can' 'courteous' 'date' 'different' 'experience' 'explaining'
 'friendly' 'from' 'get' 'had' 'have' 'human' 'in' 'indifferent'
 'interested' 'ipad' 'iphone' 'is' 'john' 'know' 'lacking' 'leeds' 'lewis'
 'mine' 'my' 'need' 'not' 'november' 'of' 'or' 'out' 'person' 'poor'
 'product' 'products' 'rating' 'recently' 'review' 'savvy' 'selling'
 'service' 'should' 'sought' 'staff' 'stars' 'states' 'stores' 'techno'
 'that' 'the' 'their' 'there' 'they' 'totally' 'touch' 'very' 'was' 'we'
 'well' 'were' 'what' 'which' 'whom' 'wife' 'wish' 'with']
```

Term frequency inverse document frequency measures how frequent a word in a document is and how unique and important a word is across all the documents in the corpus; a corpus is a collection of text documents.

```
vectorizer_tfidf = TfidfVectorizer()
tfidf_matrix = vectorizer_tfidf.fit_transform(corpus)
print("\nTF-IDF representation:\n", tfidf_matrix.toarray())
```

Output:

```
TF-IDF representation:
 [[0.06868028 0.06868028 0.06868028 0.13736056 0.20604085 0.06868028
  0.13736056 0.13736056 0.13736056 0.06868028 0.06868028 0.27472113
  0.06868028 0.06868028 0.06868028 0.06868028 0.13736056 0.06868028
  0.06868028 0.13736056 0.06868028 0.06868028 0.06868028 0.06868028
  0.27472113 0.06868028 0.06868028 0.06868028 0.06868028 0.13736056
  0.06868028 0.06868028 0.06868028 0.06868028 0.06868028 0.06868028
  0.06868028 0.06868028 0.13736056 0.06868028 0.13736056 0.06868028
  0.06868028 0.06868028 0.06868028 0.06868028 0.06868028 0.06868028
  0.06868028 0.06868028 0.06868028 0.06868028 0.06868028 0.06868028
  0.06868028 0.06868028 0.06868028 0.06868028 0.20604085 0.06868028
  0.06868028 0.41208169 0.06868028 0.13736056 0.13736056 0.06868028
  0.06868028 0.13736056 0.20604085 0.13736056 0.13736056 0.06868028
  0.06868028 0.06868028 0.06868028 0.06868028 0.06868028 0.06868028]]
Review_text_04
```

The next step was metadata which ensured the extracted features are stored together, and sentiment labelling. In this I described the vectorisation process for each text document with a brief description of what it actually is, and then a document summary. Next I labelled each text document with a sentiment either positive, neutral or negative based on the ratings given and the tone of the review. I did these manually and then save it as one json file.

Below is the start of my code for the metadata and then for the sentiment labels:

```
text_metadata = {
    "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": {
        "document_1": "The pre processing techniques I used on this review text 1
were text normalising, which is converting to lower case so all texts are
uniform, and tokenisation which is splitting the text into separate words. This
text was vectorized using Bag of words to count word frequency, and Term
Frequency-Inverse Document Frequency showed the uniqueness and importance of a
word in relation to all the documents. ",

"sentiment_labels": {
        "document_1": "negative",
        "document_2": "negative",
        "document_3": "negative",
        "document_4": "positive",
        "document_5": "negative",
        "document_6": "positive",

import json

with open("text_metadata.json", "w") as file:
    json.dump(text_metadata, file, indent=4)
```

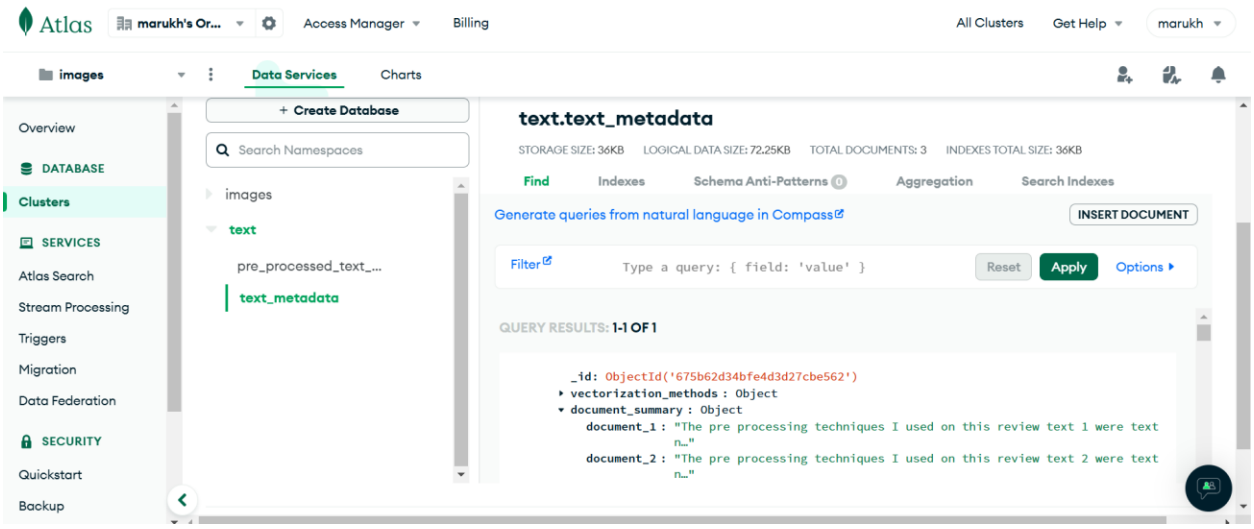
The last step was to store the pre-processed text data and associated sentiment labels in a NoSQL database. MongoDB stores data that is unstructured such as text files and is used

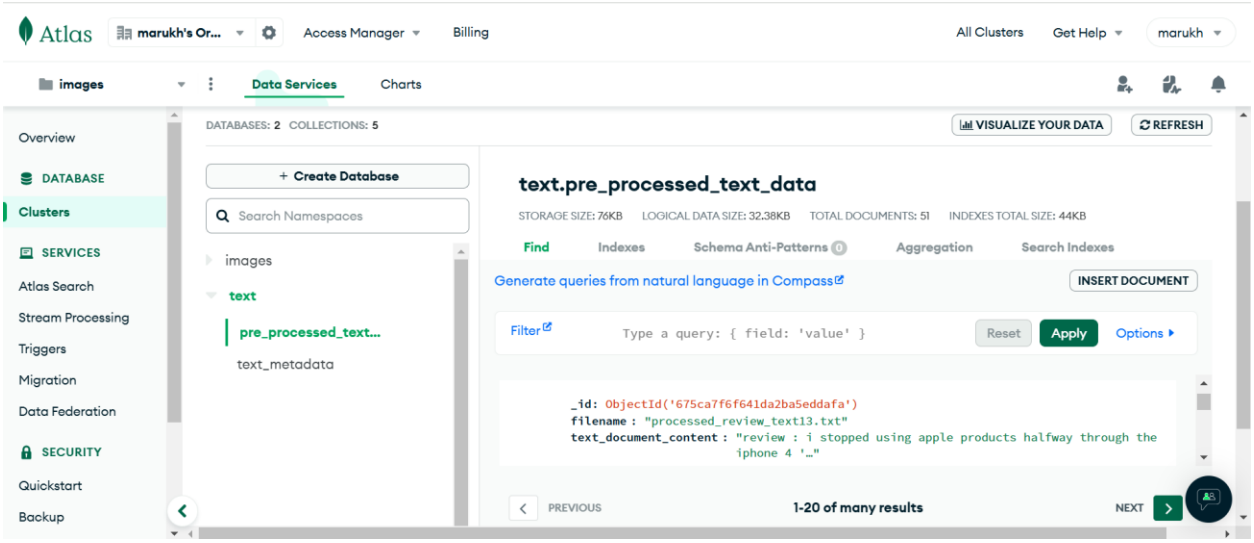
because it can easily handle large datasets, is efficient in queries, is flexible and can store a variety of metadata.

I did this by importing and installing MongoDB on google Colab, creating a database (called text) and then its two collections, text_metadata and the pre_processed_text_data. Next, I inserted the text metadata and sentiment labels, and the text data into the database through code like so, using a for loop and os path:

```
import pymongo
from pymongo import MongoClient
import os

# connect to mongodb
text_connection = pymongo.MongoClient("mongodb+srv://marukh:Marukh2005@cluster0.zxly9.mongodb.net/")

# selecting  database and collection
db = text_connection["text"]
pre_processed_text_collection = db["pre_processed_text_data"]


folder_path = '/content/processed_text_data/'

#loop through all files in the folder
for filename in os.listdir(folder_path):
    file_path = os.path.join(folder_path, filename)  #constructingthe full file path

    # reading the content of each file
    with open(file_path, 'r') as file:
        file_content = file.read()

    # creating a document structure for MongoDB to show the name of the file and the content of the review
    document = {
        "filename": filename,
        "text_document_content": file_content
    }

    pre_processed_text_collection.insert_one(document)  # this inserts the text content into MongoDB

#all files have now been uploaded to MongoDB
```

The screenshots below are from MongoDB which confirm my data has been correctly loaded:





The code below is a query to check the data documents loaded correctly, the code retrieves all the documents in the collection, if the collection is empty then it wont print anything showing the data insertion failed. I also included a cropped screenshot of some of the output it produced (all image text data)

```
#sample query
for doc in pre_processed_text_collection.find():
    print(doc)
```

```
[84] #sample query
     for doc in pre_processed_text_collection.find():
         print(doc)
```

```
{'_id': ObjectId('675ca7f4f641da2ba5eddaf9'), 'filename': 'processed_review_text05.txt', 'text_document_content': 'review : the process was slow ... over a week of cal
{'_id': ObjectId('675ca7f6f641da2ba5eddafa'), 'filename': 'processed_review_text13.txt', 'text_document_content': "review : i stopped using apple products halfway thro
{'_id': ObjectId('675ca7f6f641da2ba5eddafb'), 'filename': 'processed_review_text20.txt', 'text_document_content': 'review : apple have the best customer service possib
{'_id': ObjectId('675ca7f6f641da2ba5eddafc'), 'filename': 'processed_review_text29.txt', 'text_document_content': "review : i stopped using apple products halfway thro
{'_id': ObjectId('675ca7f6f641da2ba5eddafd'), 'filename': 'processed_review_text08.txt', 'text_document_content': 'review : one year ago , i went to the apple store to
{'_id': ObjectId('675ca7f6f641da2ba5eddafe'), 'filename': 'processed_review_text25.txt', 'text_document_content': "review : genuinely advise people think twice relying
{'_id': ObjectId('675ca7f6f641da2ba5eddaff'), 'filename': 'processed_review_text28.txt', 'text_document_content': 'review : received the products i purchased , and i '
{'_id': ObjectId('675ca7f6f641da2ba5eddb00'), 'filename': 'processed_review_text09.txt', 'text_document_content': 'review : ordered new iphone 16 pro max . order journ
{'_id': ObjectId('675ca7f7f641da2ba5eddb01'), 'filename': 'processed_review_text32.txt', 'text_document_content': "review : i regret switching from android to iphone .
{'_id': ObjectId('675ca7f7f641da2ba5eddb02'), 'filename': 'processed_review_text12.txt', 'text_document_content': 'review : went apple store trafford centre purchase r
{'_id': ObjectId('675ca7f7f641da2ba5eddb03'), 'filename': 'processed_review_text31.txt', 'text_document_content': 'review : got phone stolen , went apple store stratfo
{'_id': ObjectId('675ca7f7f641da2ba5eddb04'), 'filename': 'processed_review_text33.txt', 'text_document_content': 'review : after being overcharged for an icloud subsc
{'_id': ObjectId('675ca7f7f641da2ba5eddb05'), 'filename': 'processed_review_text41.txt', 'text_document_content': 'review : amazing service from devon in the leeds bra
{'_id': ObjectId('675ca7f7f641da2ba5eddb06'), 'filename': 'processed_review_text34.txt', 'text_document_content': "review : i was disappointed with my new iphone 14 pr
```