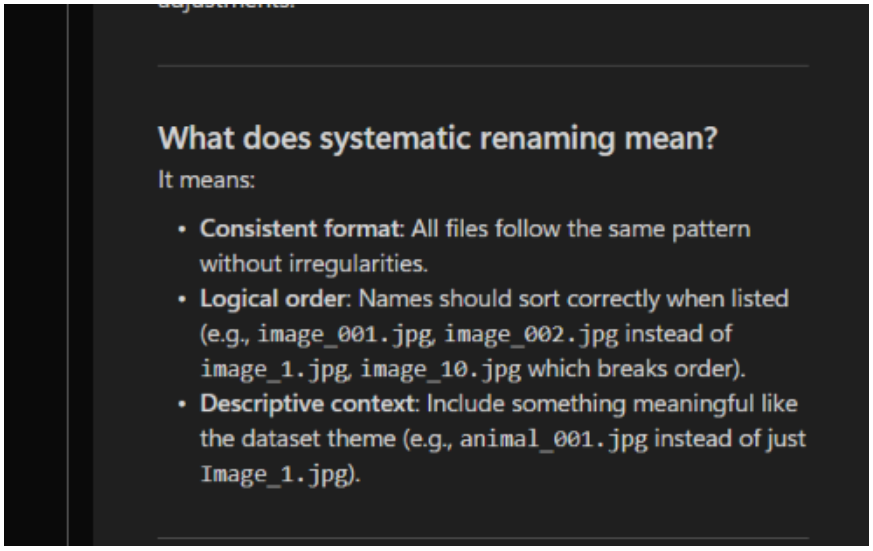
Content-Based Image Retrieval & Sentiment Analysis Database Report

Contents

Artificial Intelligence Use Declaration

I used Perplexity Ai to help break down complex tasks that were unclear to me so I fully understood the tasks, and I utilised its deep research feature to learn about models, industry practises for coding and documenting projects. In addition, I used it to help plan and outline the structure of my documentation and gain an understanding of how long each task would take me to stay on track. I used Gemini Ai as a last resort at times to explain errors in my code to me when I couldn't understand the syntax errors. I also used Claude Ai to review my code to get a different perspective to ensure my code wasn't overly filled with # comments and print statements.



Here ChatGPT was used to clarify what was meant by systematic renaming.

I confirm that I have not used Ai to generate content that I am presenting as my own work and I remain fully responsible for the code and conclusions in this submission.

Objectives

CBIR database:

The purpose of the database for content based image retrieval database is to store animal images along with their visual features allowing users to search and retrieve images based on visual content.

Sentiment analysis database:

The purpose for the sentiment analysis database is to store patient reviews from The Mission Practise and classify the sentiment conveyed in the reviews. This will help the practise gauge public sentiment towards their services and make informed decisions based on insights extracted from the processed reviews and metadata stored within the database.

Data Sources

Download these files to be able to run the code and also conduct tests in my database.

Animals file:

Link to download original animal images: Animals

After downloading you need to create a folder named Animals and upload the images to there.

Link to Animal_features folder: Animal_features

The Mission Practise file:

Link to download original text data: THE MISSION PRACTICE (Review Audit Reviews).csv

Link to text vectorization and labelling Json file:
text_vectorization_and_labelling_metadata.json

Methodology

In my code I ensured to consistently use print statements in order to track the flow of my code and also implemented try except techniques in order to handle errors.

CBIR database

First, I imported all libraries that will be used in the process of creating the database

```
[6]
Os
import os #write and read files
import json #creating Json files
import numpy as np #image feature extraction
from skimage import feature # image processing
from os import listdir #file directory use
from PIL import Image # image processing tool
import matplotlib.pyplot as plt # to show images
import shutil #file directories
import cv2 #Open CV
```

Os - The os module will be frequently used as it allows me to be able to iterate through all the images in my folders which will allow me to have images with the same effects.

Listdir is from the os module, which will be frequently used in returning filenames in my dataset.

JSON - This is crucial to have as it will me to be able to share extract features from images and a structured metadata into my database.

NumPy as np – This will come in handy during feature vector tasks and can be used to calculate different measures in my images.

Skimage.feature – This is imported because it contains functions that can be used to compute GLCM and contrast.

PIL - the pillow library is another library that will be frequently used because I would need to open my raw image files and ensure they have a consistent format.

Matplotlib – This library was imported as optional just in case I plan to retrieve images based on their relevancy.

Shutil – This library was imported as it helps copy, move and remove files and directories.

Cv2 – OpenCV is commonly used in CBIR pipelines for converting images from BGR to RGB and applying a variety of features.

Explanation of libraries in depth

A) Image Collection:

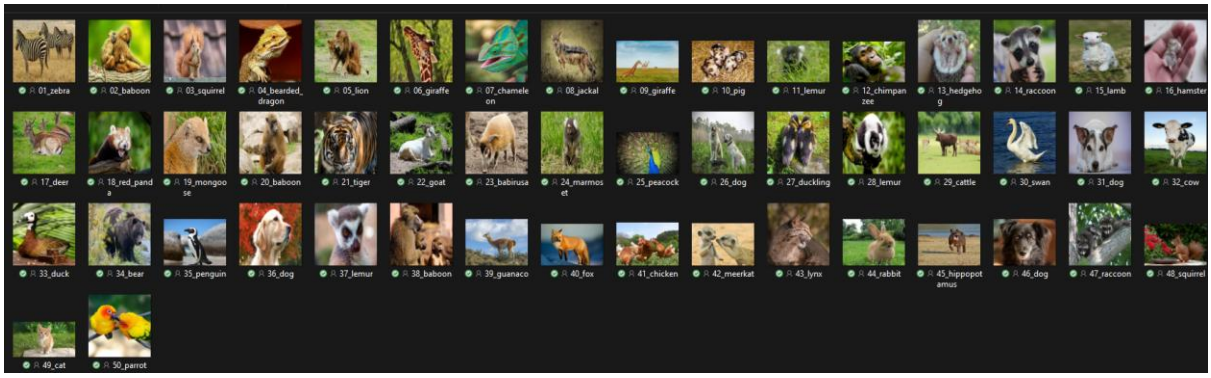
Bing image downloader.ext library was used to download animal images from Bing . I decided to go this route because it was the fastest way to collect 50 images and specified the file type to be jpg have a uniform size of 500 x 500. I downloaded the images into my computer ensure I could put it back onto google colab after runtime disconnected and check what the images all looked like.

```
2+from bing_image_downloader import downloader # Imports the corr
+Code cell <jyxseLzR-t2n>
+# %% [code]
1+search_queries = ["Animals"]
2
3+#dim and file type specified
4+for query in search_queries:
5+  downloader.download(query, limit=50, output_dir='Animals_data
+Execution output from 17 Dec 2025 11:39
+19KB
+    Stream
+        [%] Downloading Images to /content/Animals_dataset/Anim
```

```
adult_filter_off=True, resize_dim=(500, 500), file_type='jpg')
```

Here is code from my revision history as proof of bing downloader ext was used to download images.

50 Animal images:



I decided to collect animal images because I would be able to class different animal types such as mammal, bird and reptile

B) Image Preprocessing:

for loop to iterate through every image in the folder using ‘os.listdir’ and opened each image.

```
#Loop through every file in source folder
for filename in os.listdir(src):
  if filename.endswith('.jpg'):
      try:
        #open image
        img_path = os.path.join(src, filename)
        with Image.open(img_path) as img:
```

To preprocess the animal images, I applied four steps:

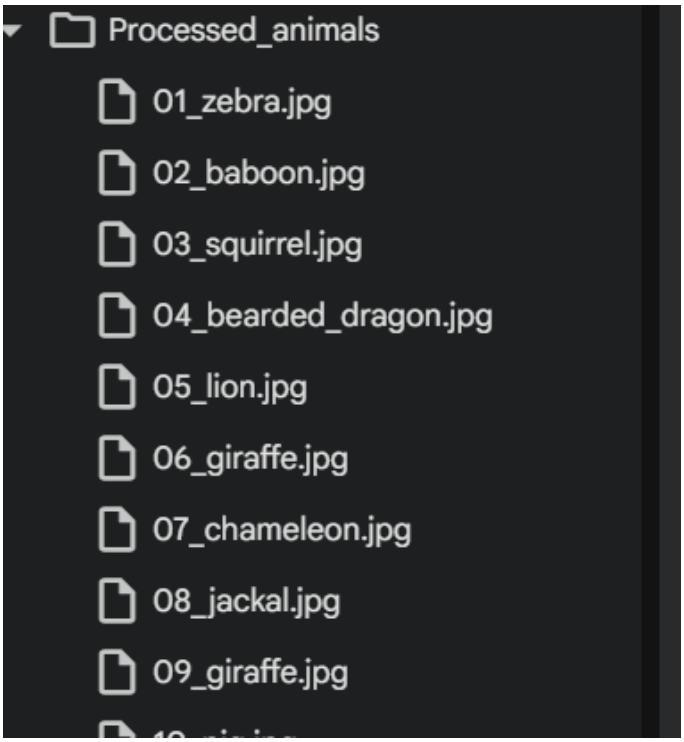
1) RGB Conversion: All images in source folder were converted to RGB format this meant that my dataset now contained a consistent format to so that features within the images can be extracted easily.

2) Resizing: Images were resized to 500x500 pixels. Having a uniform size for images in my dataset is good for consistency and helps the database handle a lot of images easier. If I didn’t do this then it would mean my images would be in different types of format and the extracted information wouldn’t be reliable and it could cause issues in retrieving image or extract features.

3) Denoising: Gaussian Blur with a 5x5 kernel was applied to reduce noise whilst preserving the edges. I intentionally chose a 5x5 instead of 3x3 because of its ability to smooth the noise but preserves the edges of my images.

4) Normalization: Pixel values were normalized to the 0-1 range to improve consistency across images with different brightness levels, then converted back to uint8 (0-255) so it would be in a suitable format to store in the database.

After processing images, I saved them into the Processed_animals folder to instead of the raw folder to manage workflow.



Here is the location of processed images

I iterated through each image checking for 500 x 500 pixels and RGB to ensure the process was successful

```
01_zebra.jpg is 500x500 (OK)
08_jackal.jpg is 500x500 (OK)
04_bearded_dragon.jpg is 500x500 (OK)
33_duck.jpg is 500x500 (OK)
45_hippopotamus.jpg is 500x500 (OK)
42_meerkat.jpg is 500x500 (OK)
23_babirusa.jpg is 500x500 (OK)
20_baboon.jpg is 500x500 (OK)
31_dog.jpg is 500x500 (OK)
47_raccoon.jpg is 500x500 (OK)
18_red_panda.jpg is 500x500 (OK)
15_lamb.jpg is 500x500 (OK)
24_marmoset.jpg is 500x500 (OK)
50_parrot.jpg is 500x500 (OK)
09_giraffe.jpg is 500x500 (OK)
48_squirrel.jpg is 500x500 (OK)
29_cattle.jpg is 500x500 (OK)
12_chimpanzee.jpg is 500x500 (OK)
44_rabbit.jpg is 500x500 (OK)
35_penguin.jpg is 500x500 (OK)
05_lion.jpg is 500x500 (OK)
28_lemur.jpg is 500x500 (OK)
46_dog.jpg is 500x500 (OK)
```

All processed images were in 500 x 500 pixels.

```
01_zebra.jpg → Mode: RGB
08_jackal.jpg → Mode: RGB
04_bearded_dragon.jpg → Mode: RGB
33_duck.jpg → Mode: RGB
45_hippopotamus.jpg → Mode: RGB
42_meerkat.jpg → Mode: RGB
23_babirusa.jpg → Mode: RGB
20_baboon.jpg → Mode: RGB
31_dog.jpg → Mode: RGB
47_raccoon.jpg → Mode: RGB
18_red_panda.jpg → Mode: RGB
15_lamb.jpg → Mode: RGB
24_marmoset.jpg → Mode: RGB
50_parrot.jpg → Mode: RGB
09_giraffe.jpg → Mode: RGB
48_squirrel.jpg → Mode: RGB
29_cattle.jpg → Mode: RGB
12_chimpanzee.jpg → Mode: RGB
44_rabbit.jpg → Mode: RGB
35_penguin.jpg → Mode: RGB
05_lion.jpg → Mode: RGB
28_lemur.jpg → Mode: RGB
46_dog.jpg → Mode: RGB
40_fox.jpg → Mode: RGB
```

All images were in the RGB mode.

C) Image Annotation:
For Image annotation I decided to go the manual route by writing the animal type, keywords and description. After I saved all annotations into a Json File I decided to not use an Ai tool because I was unable to find a tool that could automatically annotate images whilst meeting the criteria of keywords and description.

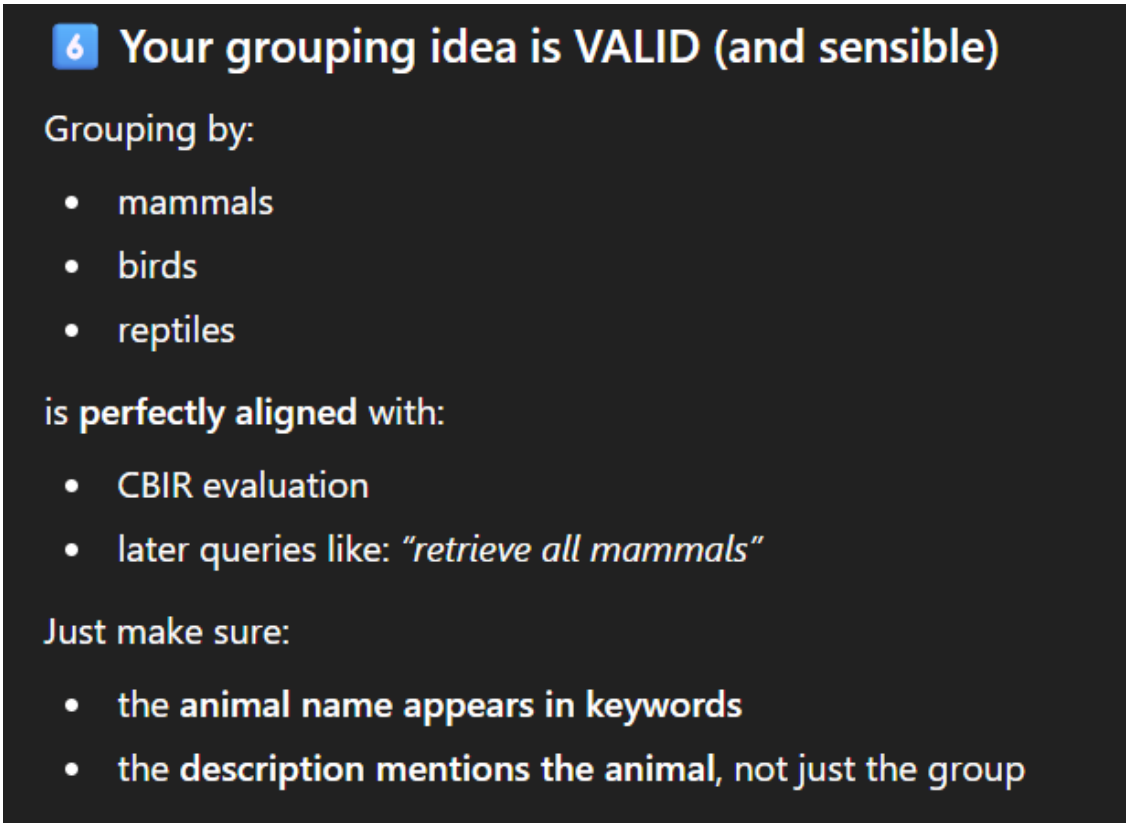

```
[
  {
    "filename": "01_zebra.jpg",
    "animal_type": "mammal",
    "keywords": [
      "mammal",
      "zebra",
      "savannah",
      "group",
      "standing"
    ],
    "description": "zebras standing on dry grassland"
  },
  {
    "filename": "02_baboon.jpg",
    "animal_type": "mammal",
    "keywords": [
      "mammal",
      "baboon",
      "hugging",
      "closeup"
    ],
    "description": "two baboons"
  },
  {
    "filename": "03_squirrel.jpg",
    "animal_type": "mammal",
    "keywords": [
      "mammal",
      "squirrel",
      "orange fur",
      "sitting",
      "closeup"
    ],
    "description": "orange squirrel"
```


Here is what the format of my metadata Json file looks like.

Structure Rationale:
Animal type: I decided my animals would be grouped by the 3 animal types I had in my dataset which was mammal, bird and reptile. It would be beneficial when I store the dataset as I would be able to search for retrieve based on their animal type.

Keyword: The keyword also includes animal types first, and everything else was based on camera shot, environment, sitting position and sometimes a specific trait the animal had. Having this structure would help me separate animals in case I had the same animals but different images of them.

Description was included in the structure to provide a human readable explanation of the image content.



ChatGPT helped with coming up with ideas of how I could structure my metadata.

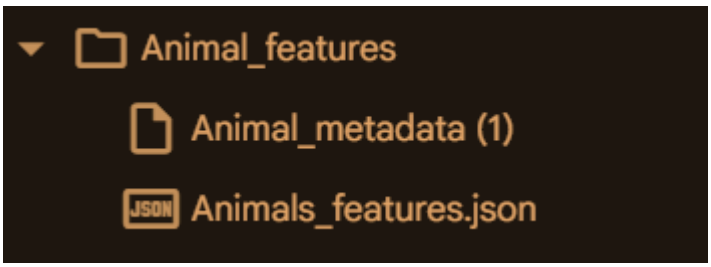
D) Image Feature Extraction:
Since computers cannot see what humans see, I need to turn the images into a small set of meaningful numbers that describe how it looks to the computer. The features I decided to extract are:

Mean & Norm intensity: This captures the colour information of the image which will enables a computer to understand whether the image is dark or bright and if there is a specific colour that dominates the image. Open cv was used to convert them to RGB and they were converted to a float data type for the rgb analysis.

Convert to grayscale: This was implemented to prepare the image for area calculation and texture feature analysis.

Compute area: This measures how much of the image is occupied by the animal by separating the animal from the background which is done by applying a threshold and counting how many pixels belongs to the animals.
Doing so allows the computer to understand the size of the animal and whether the animal is close or far away in the image.

Texture feature extraction: Using GLCM technique from the skimage library I calculated the texture contrast of the image.



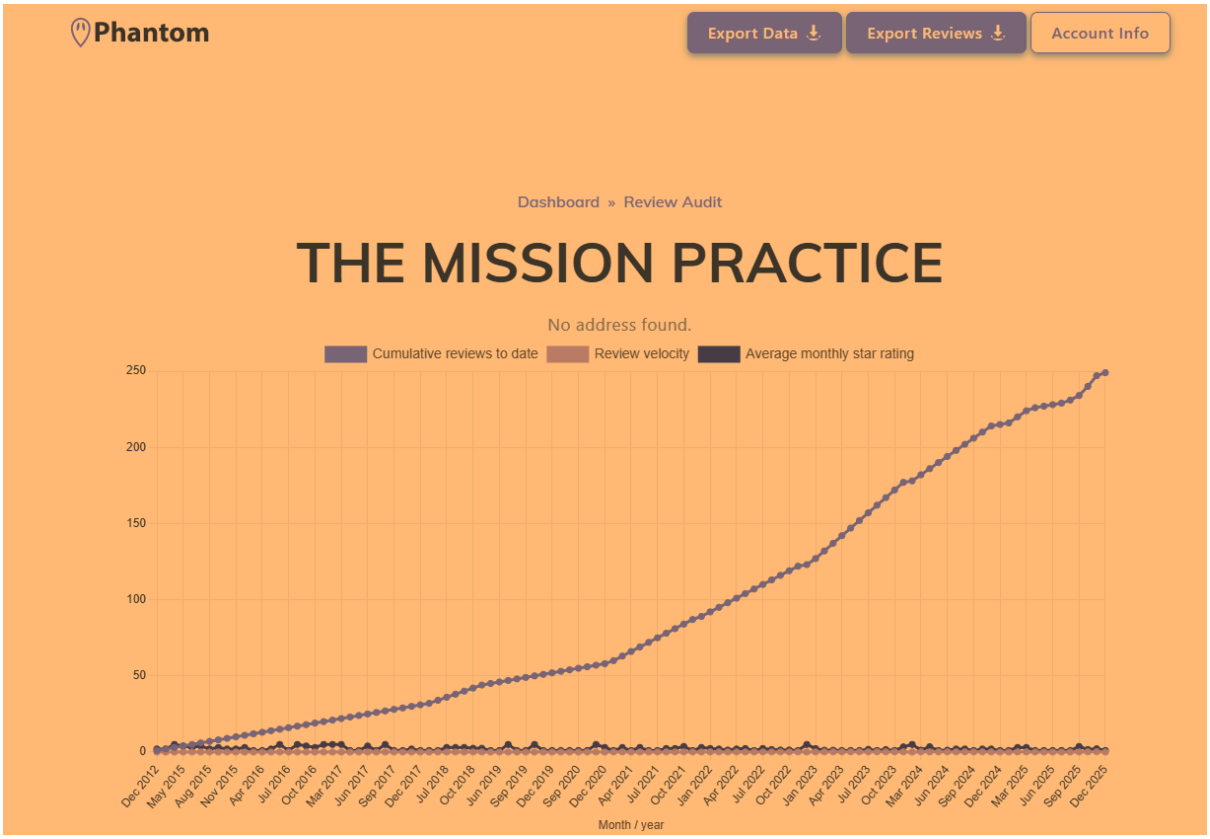
Feature was saved as a JSON file called “Animals_features” under the Animal_features folder.

```
[
  {
    "filename": "01_zebra.jpg",
    "intensity_features": {
      "mean_intensity": 125.7781,
      "norm_intensity": 119077.9148
    },
    "shape_features": {
      "area": 161911.5
    },
    "texture_features": {
      "contrast": 231.1272
    }
  },
  {
    "filename": "02_baboon.jpg",
    "intensity_features": {
      "mean_intensity": 117.5242,
      "norm_intensity": 118959.6565
    },
    "shape_features": {
      "area": 131428.0
    },
    "texture_features": {
      "contrast": 88.3621
    }
  },
  {
    "filename": "03_squirrel.jpg",
    "intensity_features": {
      "mean_intensity": 134.1765,
      "norm_intensity": 121290.7423
    },
    "shape_features": {
      "area": 159492.0
    }
```


Here is a display of the structure which will be implemented to the database allowing images to be retrieved through queries that is based on their features.

Sentiment analysis database

A) Textual data collection:
After exporting the patient reviews into a csv, I ingested the dataset it into a pandas data frame and which allowed me to manipulate the dataset by specifying the columns, Star Rating and Review Content and filtered to 50 rows without any missing values.



Proof of using Phantom google extension to export google reviews.

	Star rating	Review content
1	1	Booked my unwell baby an appointment at 8:30pm...
2	1	This practice's negligence put me in hospital ...
3	5	Been at this practice for a decade now. I'm re...
4	1	The reception manager Deborah Turner and the r...
5	1	The reception manager Ms Deborah Turner is the...
6	1	Find it really odd how a non liscend medical p...
7	1	I have been a patient with this practice since...
8	5	I have been attending the mission practise for...
9	1	The worst GP ever I seen they book my appointm...
10	1	They never pick up the phone or respond worst ...

Sample image of data frame.

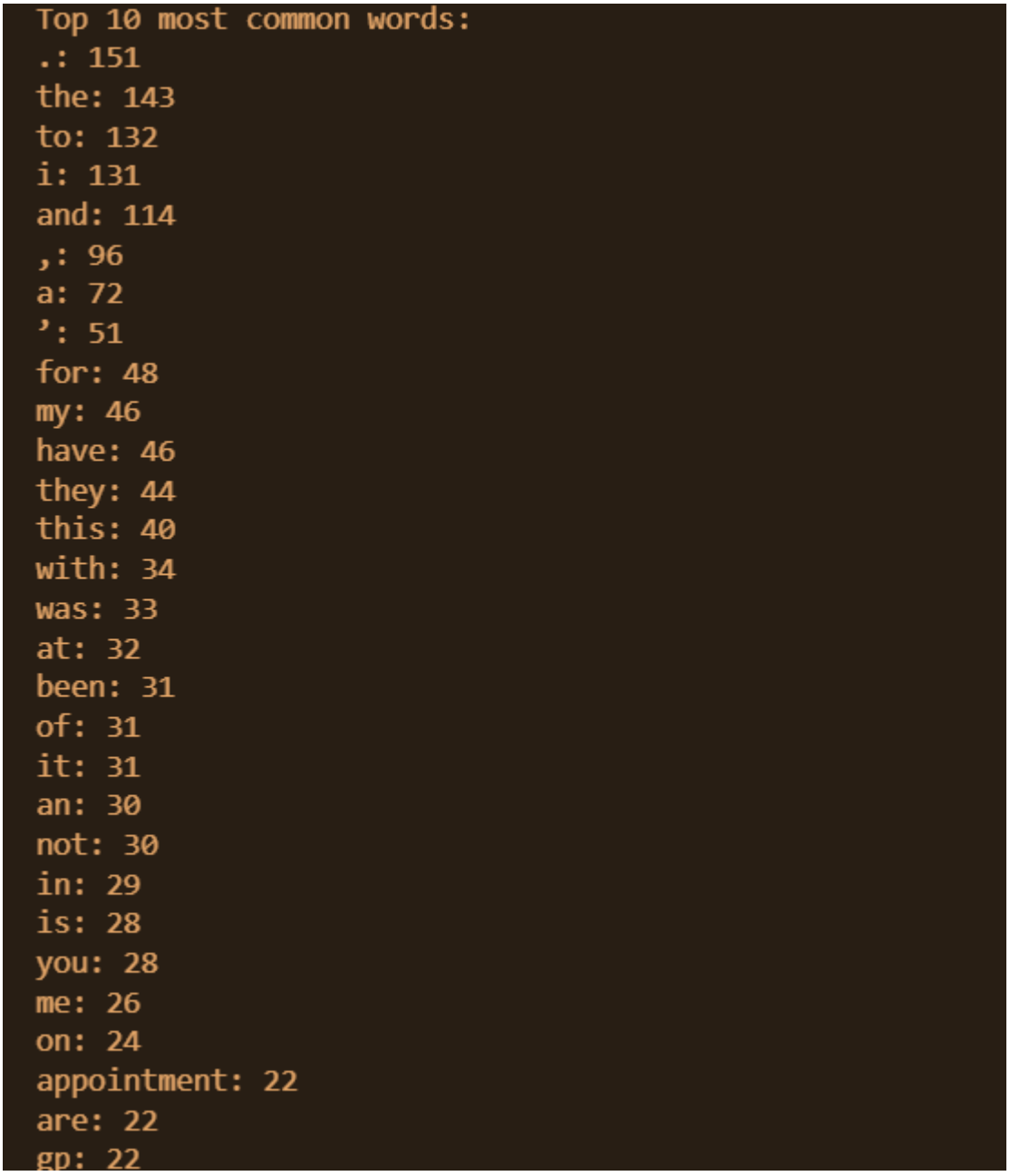
B) Textual data preprocessing:

For textual data preprocessing NLTK library was imported, preprocessing the data is essentially data cleaning so I could later proceed with sentiment analysis. The techniques I applied were:

Removal of punctuation & stop words: By removing punctuation and stop words the words in the dataset are standardised and cleaner for sentiment analysis.

Lowercase and word tokenize applying a lowercase to the dataset was intentional as I noticed people would demonstrate their emotions by writing in capitals and change to lowercase in their reviews. Word tokenize is essential as it allows model to understand language.

To measure the effectiveness of my textual data cleaning I frequently flatten the list of tokenize words into a list to count the frequency of words.


Top 10 most common words:
.: 151
the: 143
to: 132
i: 131
and: 114
,: 96
a: 72
’: 51
for: 48
my: 46
have: 46
they: 44
this: 40
with: 34
was: 33
at: 32
been: 31
of: 31
it: 31
an: 30
not: 30
in: 29
is: 28
you: 28
me: 26
on: 24
appointment: 22
are: 22
gp: 22
First check

```
Top 10 most common words:
.: 151
,: 96
’: 51
appointment: 22
gp: 22
service: 19
practice: 18
call: 18
phone: 16
care: 16
n't: 16
reception: 15
receptionist: 15
get: 15
staff: 14
!: 14
time: 13
surgery: 12
doctor: 12
never: 11
need: 10
always: 9
back: 9
dr: 9
even: 9
doctors: 8
told: 8
one: 8
mission: 8
```

Second check

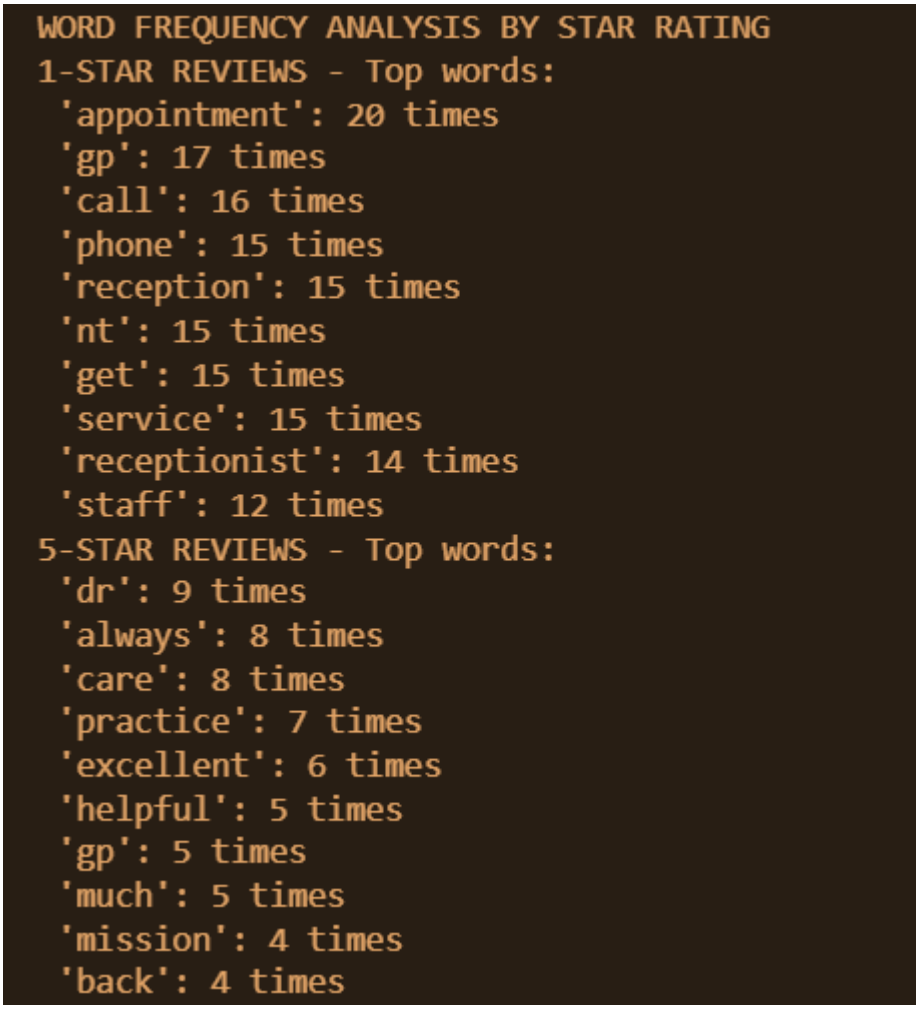
```
Top 10 most common words:
appointment: 22
gp: 22
service: 19
practice: 18
call: 18
phone: 16
care: 16
nt: 16
reception: 15
receptionist: 15
get: 15
staff: 14
time: 13
surgery: 12
doctor: 12
never: 11
need: 10
always: 9
back: 9
dr: 9
even: 9
doctors: 8
told: 8
one: 8
mission: 8
s: 8
could: 7
```

Final check

After cleaning the dataset, the only reviews that I had were reviews with a star rating of 1 or 5. This potentially meant there was an opportunity to classify reviews as just positive or negative based on their star rating.

```
Star Rating Distribution:
Star rating
1    37
5    13
Name: count, dtype: int64
```

Here the star rating distribution of the reviews in my dataset is shown.



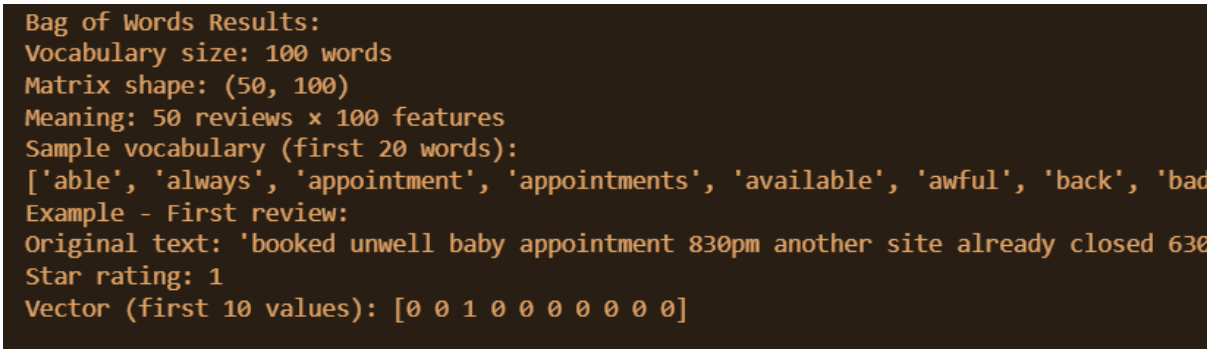
This illustrates that 5 star reviews have words like helpful, care and excellent whilst 1 star reviews only show words that are just associated with a GP implying that 5 star reviews were just positive and 1 star reviews were just negative. By checking this it meant I could classify the reviews based on their star rating.

C) Text vectorisation:
For text vectorisation I joined the tokens back into text so they could be converted into numerical arrays which allowed me to apply the methods:

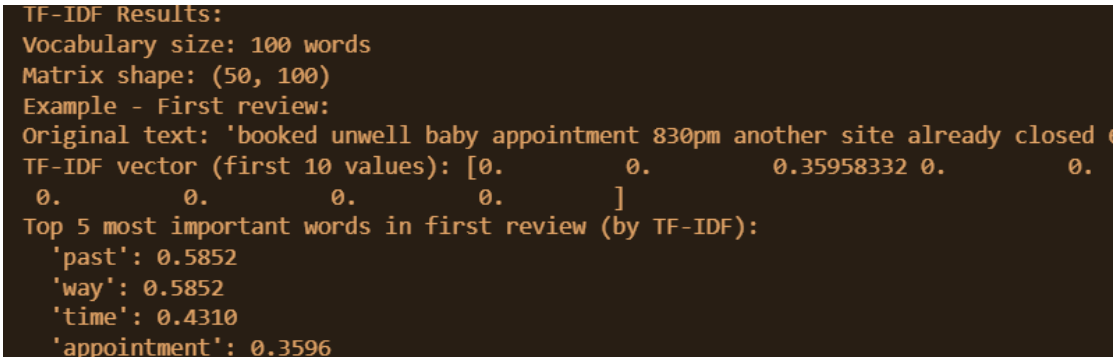
Bag of Words: This provides a simple way to count word occurrences in a numerical representation.

TF- IDF: Allows to extract distinctive words within the dataset.

These two methods are crucial for sentiment analysis patterns to be identified and understand the nuances within the reviews.



Here the result for BoWs is shown. This process was done by importing Count Vectorizer from sklearn.feature_extraction.text.



Here is the result of TF -IDF vectorizer. This process was done by importing TfidfVectorizer from sklearn.feature_extraction.text

```
Comparison: BoW vs TF-IDF Word Importance
1-STAR REVIEWS (37 reviews analyzed):
BoW - Most frequent words (total counts):
  'appointment': 20 times
  'gp': 17 times
  'call': 16 times
  'get': 15 times
  'nt': 15 times
  'phone': 15 times
  'reception': 15 times
  'service': 15 times
  'receptionist': 14 times
  'staff': 12 times
TF-IDF - Most important words (average scores):
  'service': 0.1260
  'reception': 0.0942
  'appointment': 0.0912
  'get': 0.0868
  'nt': 0.0773
  'gp': 0.0772
  'call': 0.0760
  'receptionist': 0.0627
  'staff': 0.0624
  'phone': 0.0588
```

Comparison of 1-star reviews on both methods

```
5 STAR REVIEWS (13 reviews analyzed):
BoW  Most frequent words (total counts):
  'dr': 9 times
  'always': 8 times
  'care': 8 times
  'practice': 7 times
  'excellent': 6 times
  'gp': 5 times
  'helpful': 5 times
  'much': 5 times
  'back': 4 times
  'feel': 4 times
TF-idf Most important words (average scores):
  'excellent': 0.1372
  'dr': 0.1337
  'always': 0.1298
  'care': 0.1207
  'helpful': 0.1018
  'practice': 0.0860
  'much': 0.0826
  'thank': 0.0764
  'nurse': 0.0728
  'service': 0.0703
```

Comparison of 5-star reviews on both methods

Despite both methods being used for different purposes the results are nearly identical even. From this result I was able to understand that 5 star reviews were revolved around the patients' interactions with the doctors and receiving help they needed. On the other hand, results for 1 star reviews implied that patients negative reviews primarily came from receptionist and trying to get in touch with the GP and I could tell this because terms like "phone call", "receptionist", "appointment" were the most frequent.

D) Metadata and labelling:
I decided to go with the sentimental labels positive and negative, by going through the text vectorisation process and manually reviewing the dataset myself solidified this decision. To integrate a sentiment label, I created a copy of my data frame and implemented a new column named "sentiment_label".

I created a metadata document by implementing a dictionary containing the information on the dataset, vectorization methods and sentiment labels.


```
{
  "dataset_info": {
    "dataset_name": "Mission Practice GP Reviews - Sentiment Analysis",
    "source": "Google Reviews for The Mission Practice GP surgery",
    "total_reviews": 50,
    "date_created": "2025-12-25",
    "collection_method": "Exported using Phantom browser extension"
  },
  "vectorization_methods": {
    "Bag_of_words": "Each document is represented as a vector of word counts over the selected vocabulary.",
    "Tf_idf": "Each document is represented as a TF-IDF weighted vector, emphasizing words that are distinctive for that review."
  },
  "sentiment_labels": {
    "description": "Sentiment labels assigned to classify review sentiment",
    "labeling_method": "Star rating based classification",
    "label_column": "sentiment_label",
    "source_column": "Star rating",
    "labeling_rules": {
      "positive": {
        "criteria": "Star rating = 5",
        "count": 13
      },
      "negative": {
        "criteria": "Star rating = 1",
        "count": 37
      }
    }
  }
}
```


Here is a display of the metadata structure.

Perplexity Ai was used to help me understand what could be improved for metadata file

```
Metadata file created: text_vectorization_metadata.json
```

I saved the metadata as JSON file so it could be stored in the database.

MongoDB Storage and Retrieval

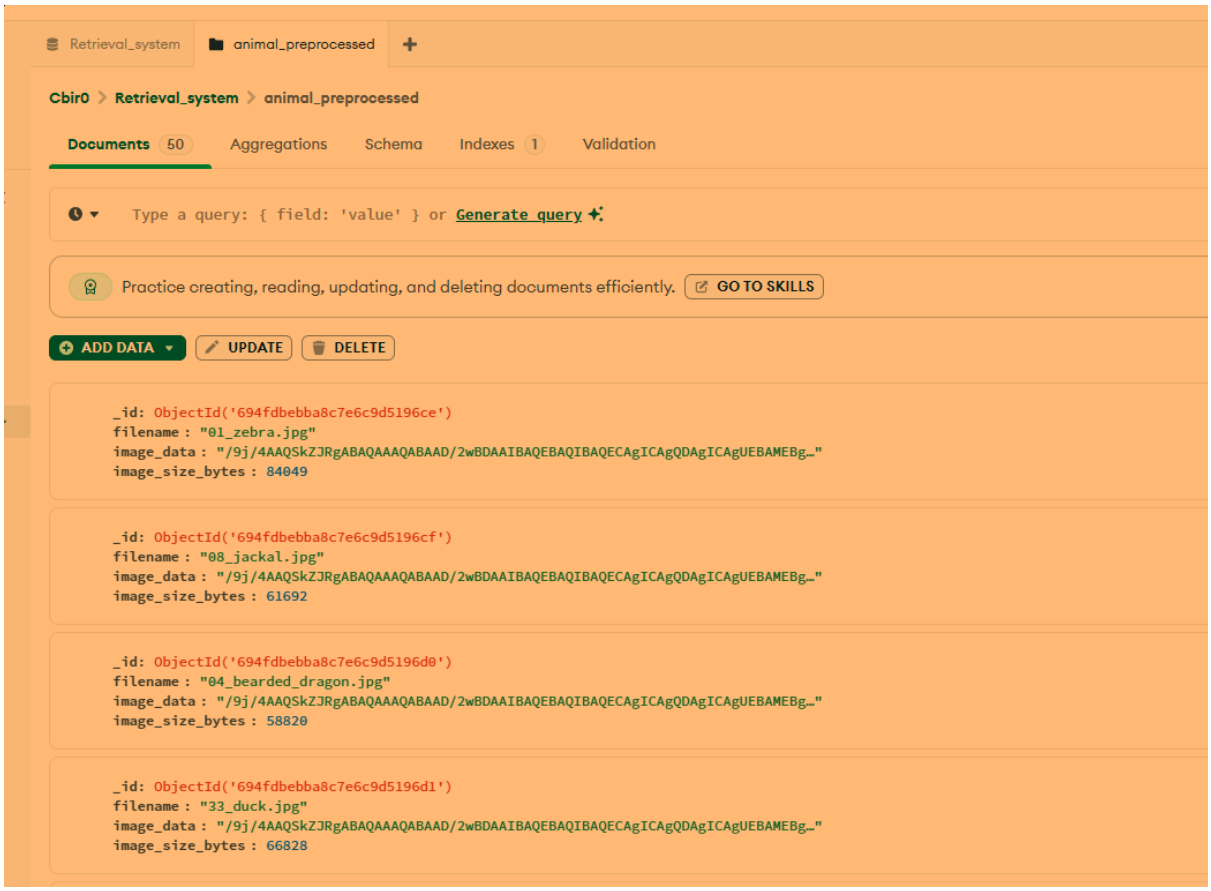
For the database storage I decided to go with MongoDB because it meets the criteria of being a NoSQL document database that can store JSON file documents. MongoDB is also able to integrate well with python which is done by install pymongo and importing Mongo Client and It gave me a URL which allows me to connect to my specific database and collection. This allowed me to insert my JSON documents. Moreover, MongoDB allows me to be able to run tests by writing queries to conduct tests to allowing me to verify if the database works as intended then last, I ensured to close the client.

CBIR MongoDB

The database was named “Retrieval_system” and I created two collection named “animal_metadata” and “animal_preprocessed” this helped organise and differentiate what is contained within each collection and database.

Storage set up display.

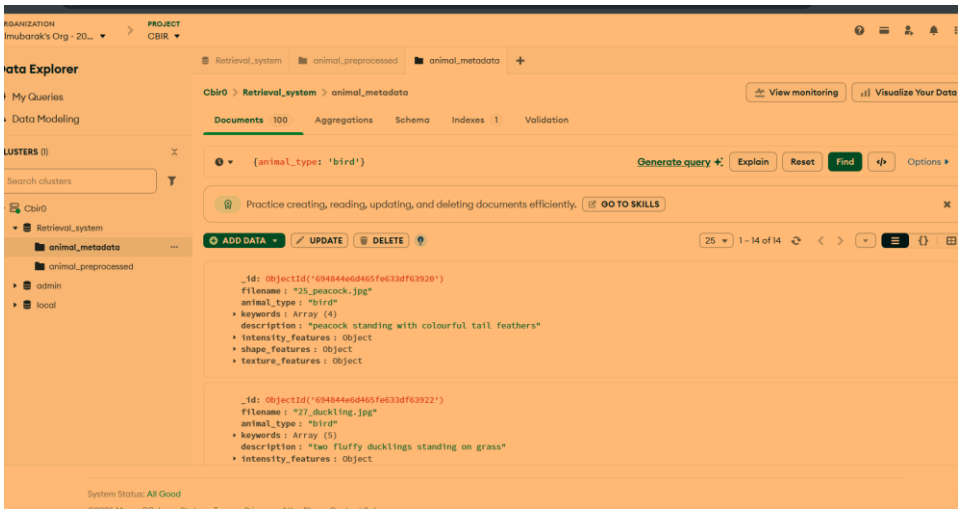
Metadata and feature extraction Json file inserted successfully.



Pre-processed animals inserted successfully.

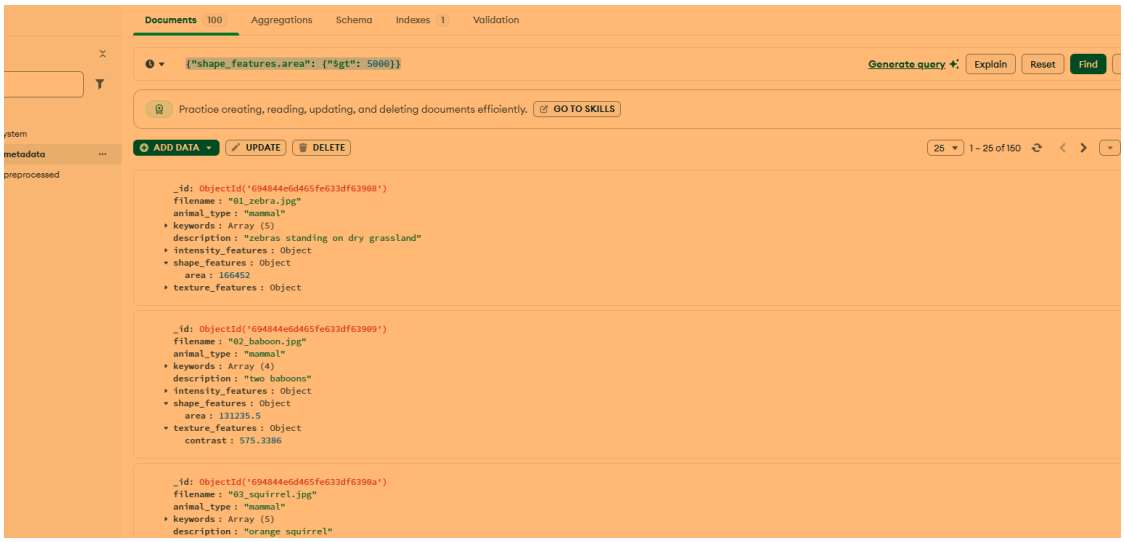
Sample Queries:

Finding images by animal type:



As you can see the query by animal type was successful and only birds were shown.

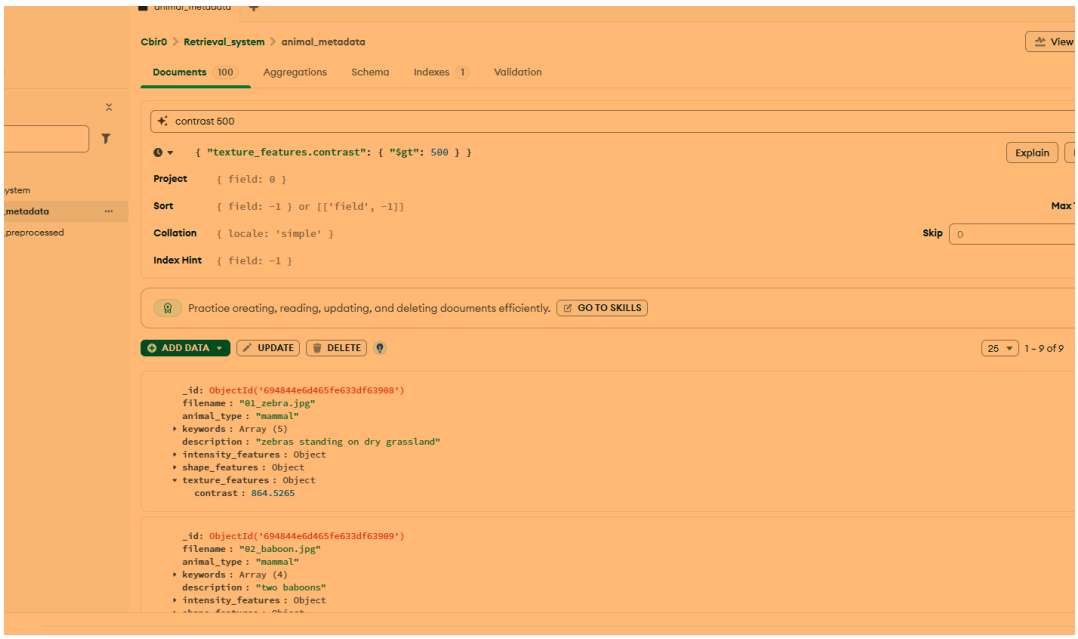
Find images with a shape feature area greater than 500:



Here the image shows the query was able to filter to only images with an area greater than 500.

The $gt is a MongoDB operator meaning greater than.

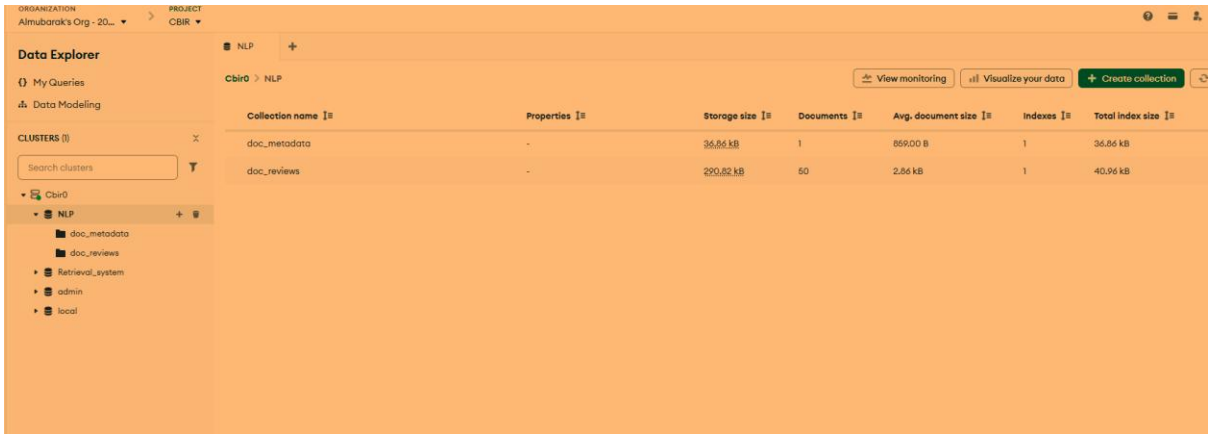
Find images with a contrast greater than 500:



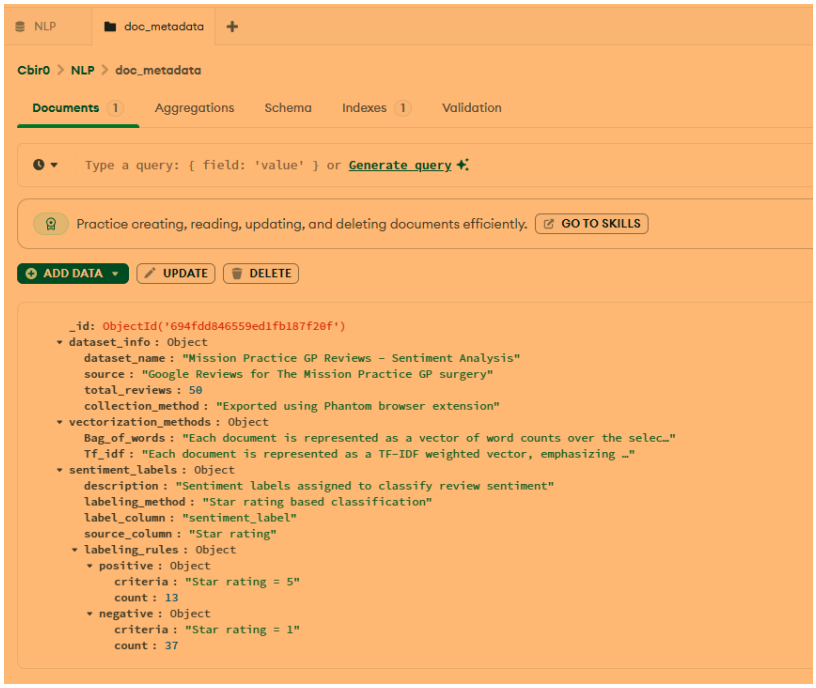
This query was successful and only images with a contrast greater than 500 were shown.

Sentiment analysis MongoDB

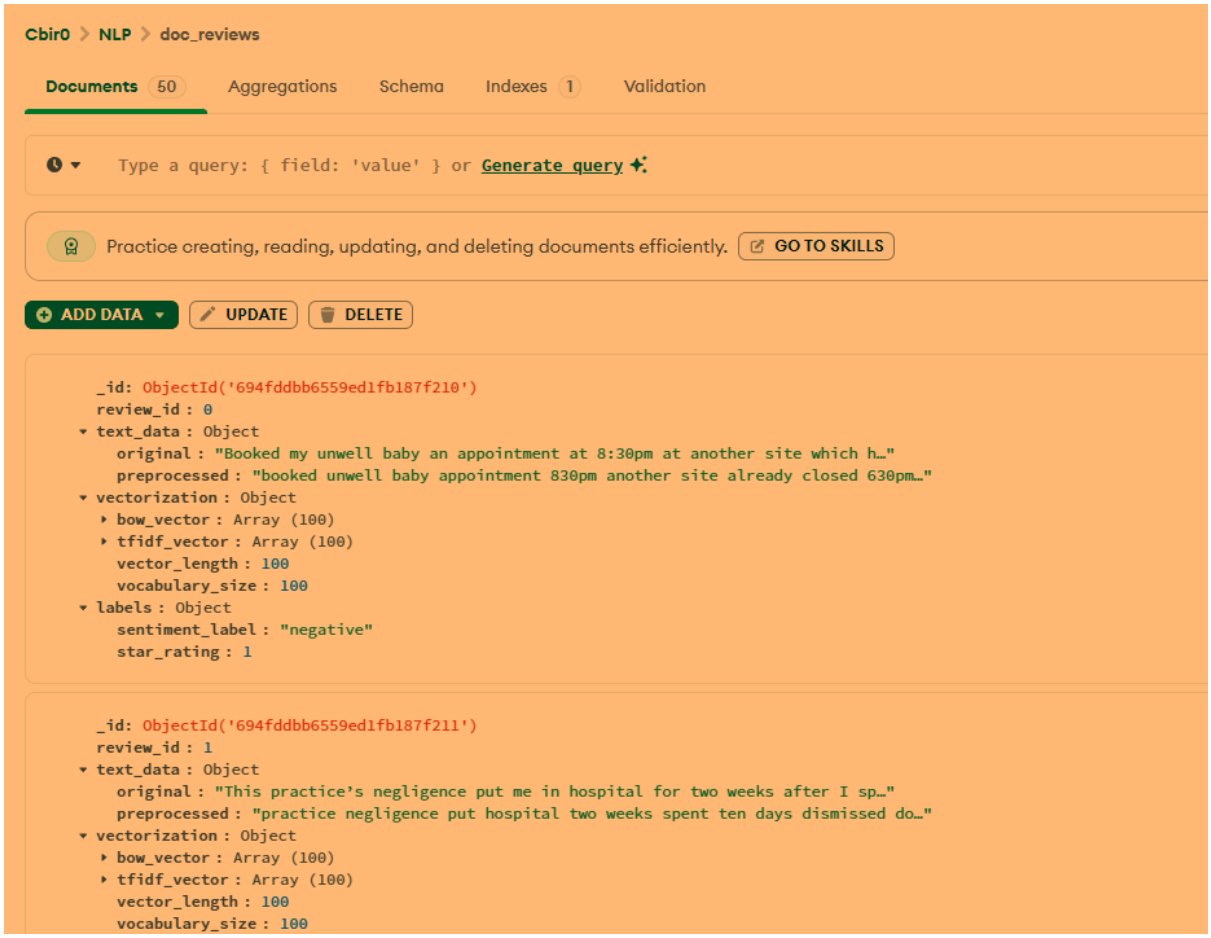
The database was named “NLP” and created two collections named “doc_metadata” and “doc_reviews” to organise where metadata and reviews will be stored.



Display of database set up.



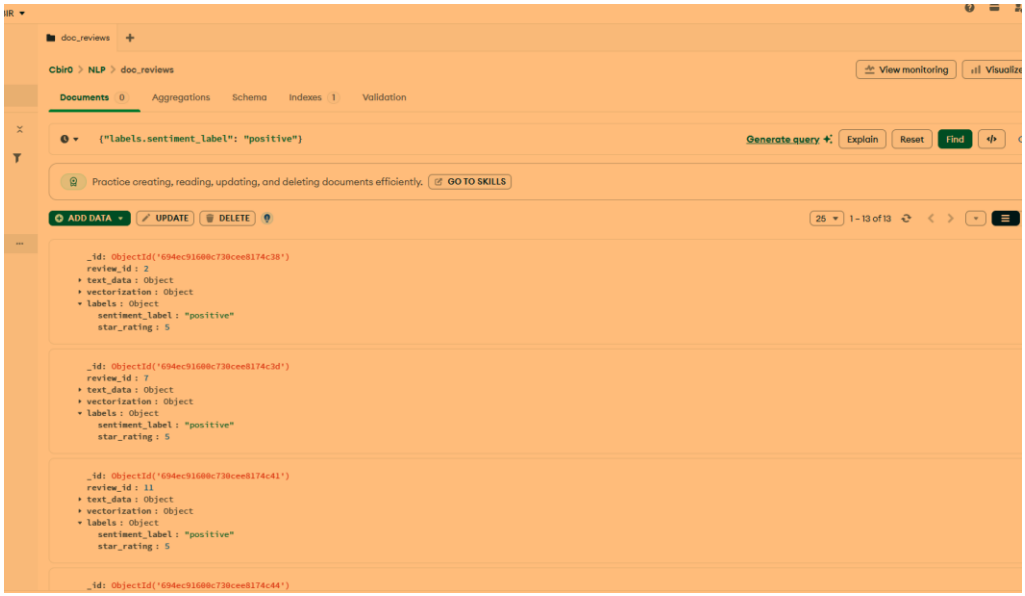
Display of what is stored inside doc_metadata collection.



Display of what is stored inside doc_reviews collection.

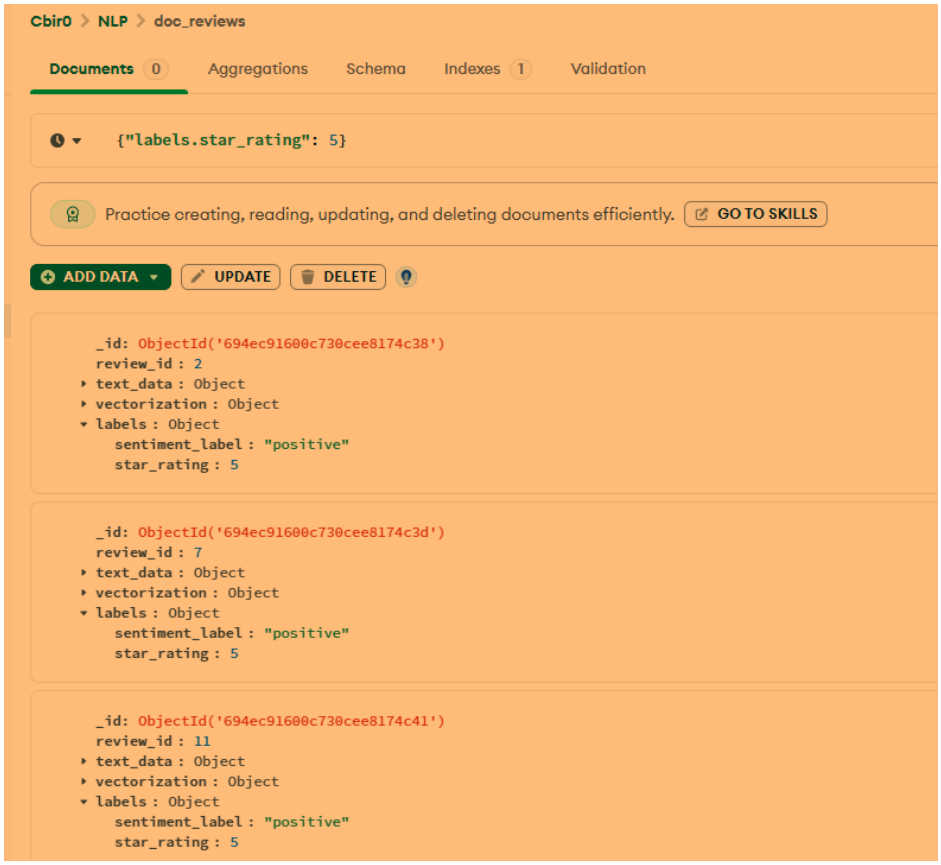
Sample Queries:

Filtered to only show reviews with a positive sentiment label



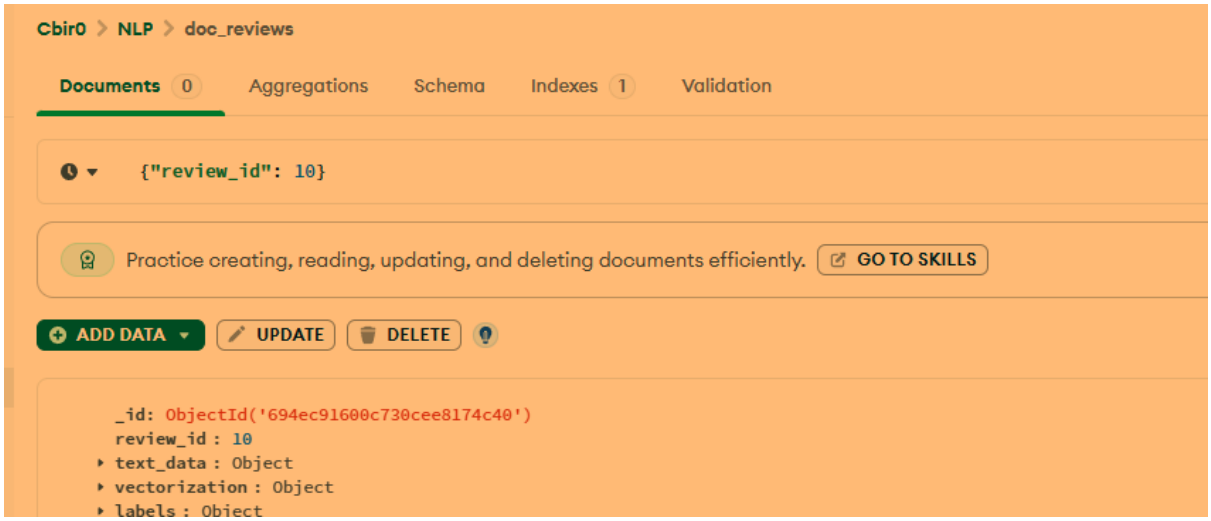
Query successful.

Apply filtering to show reviews based on star rating:



Query successful.

Search specific ID:

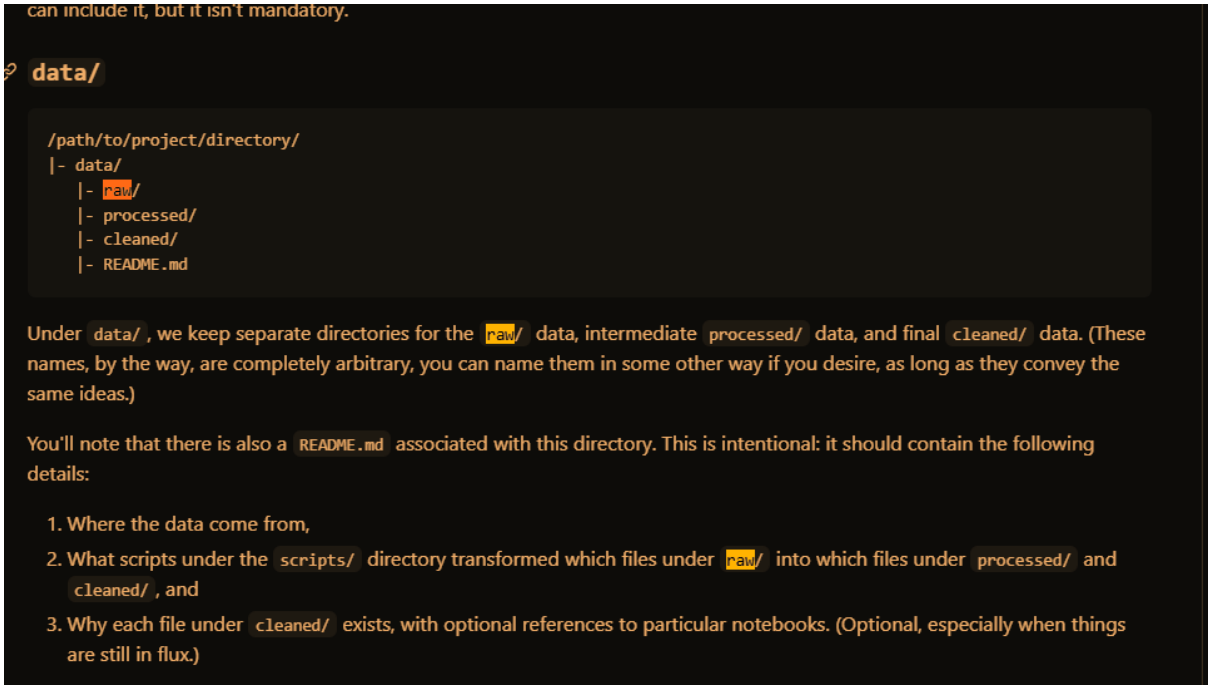


Query Successful.

Reflections

For CBIR data collection I had 10 Images that weren't the kind of images I wanted , so I decided to manually download animal images and replace the ones that weren't up to standard with them. Downloading the images into my computer allowed me to see all of them if I didn't do so I wouldn't have known the limitations that come with using Bing image downloader.

In this project I learnt a lot from Ai about considering good practises to follow writing code how I write my code and structure my report as these are essential skills that would be beneficial to start developing as I will be collaborating with people and they will be going through my project and need to have some level of understanding.


can include it, but it isn't mandatory.

data/

```
/path/to/project/directory/
|- data/
   |- raw/
   |- processed/
   |- cleaned/
   |- README.md
```

Under `data/` , we keep separate directories for the `raw/` data, intermediate `processed/` data, and final `cleaned/` data. (These names, by the way, are completely arbitrary, you can name them in some other way if you desire, as long as they convey the same ideas.)

You'll note that there is also a `README.md` associated with this directory. This is intentional: it should contain the following details:

1. Where the data come from,
2. What scripts under the `scripts/` directory transformed which files under `raw/` into which files under `processed/` and `cleaned/` , and
3. Why each file under `cleaned/` exists, with optional references to particular notebooks. (Optional, especially when things are still in flux.)