

**UNIVERSITY OF  
WESTMINSTER**



**INFORMATICS  
INSTITUTE OF  
TECHNOLOGY**

## **5DATA005W DATA ENGINEERING COURSEWORK 2**

**Lecturer Name: Ms. Yaalini  
Balathasan**

Name: M S Noor  
UOW ID: w2052142  
IIT No: 20231154

## Contents

Objectives of the Databases .....	3
Images .....	3
Text files .....	3
Question 01 .....	4
Methodology .....	4
1.b)Renaming the Images .....	4
1.b)Resizing the Images .....	5
1.b)Denoising the Images .....	7
1.c) Image Annotation.....	8
1.d)Extracting Image Features .....	9
1.d)Texture features .....	10
1.d)Shape Features .....	11
MongoDB upload process for the images.....	13
Question 02 .....	18
Normalization.....	19
Tokenization.....	20
Vectorization .....	21
Term Frequency Inverse Document.....	22
Text Metadata.....	22
Labeling .....	23

## Objectives of the Databases

### Images

The purpose of making a database of cricket players annotations and images is for the IPL (Indian Premier League) auction to support data-driven decision making, enhance auction processes, and provide actionable insights for all the franchise teams, owners and stakeholders when bidding for players. The aim of this database is to facilitate the franchise owners and management to visualize of players data through high quality images and structured annotations. This will help the franchise to recognize players, compare players and their special skills to make bids for the best players to make their respective teams.

Sources for the Images:

<https://www.gettyimages.com/search/2/image?page=2&phrase=cricket+portrait>

I got all the 51 images of cricket players through the above website.

### Text files

The purpose of getting the amazon customer reviews is to investigate consumer behavior, emotional patterns, product feedback with e-commerce. By utilizing this dataset, I aim to investigate the customer satisfaction, customer key reviews like positive and negative. By analyzing this customer reviews can provide insights into the impact of customer reviews of purchasing patterns and market trends.

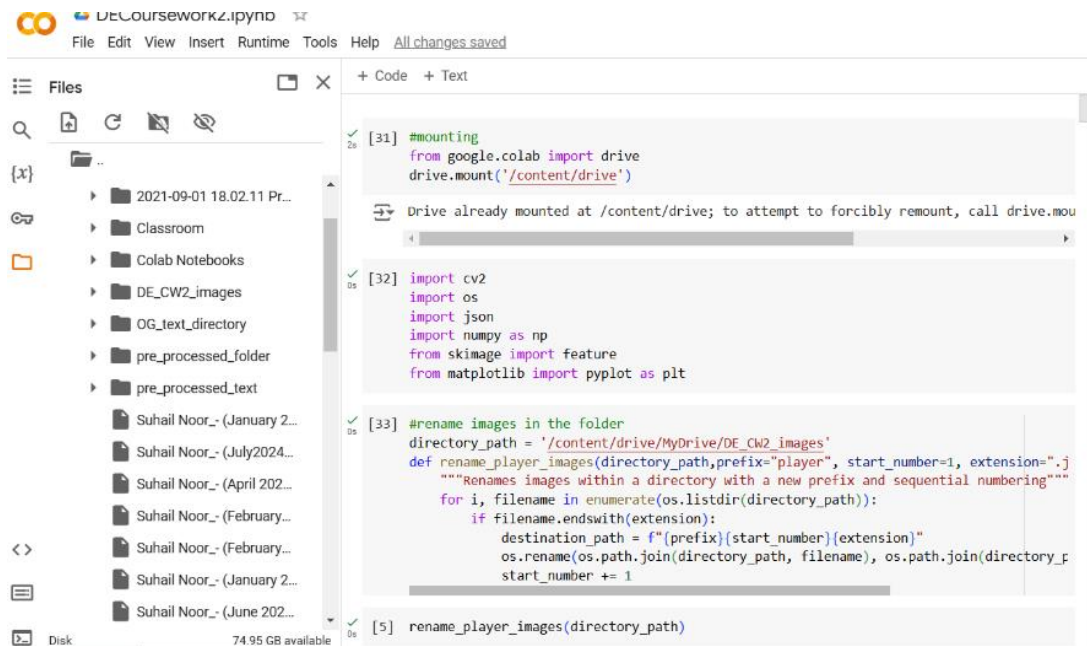
Source for the text files : Amazon online product reviews

<https://www.kaggle.com/datasets/mehmetisik/amazon-review?resource=download>

## Question 01

### Methodology

#### 1.b)Renaming the Images



The screenshot shows a Google Colab notebook interface. On the left, the 'Files' pane displays a directory structure with folders like '2021-09-01 18.02.11 Pr...', 'Classroom', 'Colab Notebooks', 'DE\_CW2\_images', 'OG\_text\_directory', 'pre\_processed\_folder', and 'pre\_processed\_text'. Below these are several files named 'Suhail Noor\_- (January 2...', 'Suhail Noor\_- (July 2024...', 'Suhail Noor\_- (April 202...', 'Suhail Noor\_- (February...', 'Suhail Noor\_- (February...', 'Suhail Noor\_- (January 2...', and 'Suhail Noor\_- (June 202...'. The main code area contains three code cells. Cell [31] mounts the Google Drive. Cell [32] imports necessary libraries: cv2, os, json, numpy, skimage, and matplotlib. Cell [33] defines a function 'rename\_player\_images' that iterates through files in a directory and renames them with a prefix and sequential numbering. The final cell [5] calls the 'rename\_player\_images' function with the directory path.

```
[31] #mounting
from google.colab import drive
drive.mount('/content/drive')

Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mou

[32] import cv2
import os
import json
import numpy as np
from skimage import feature
from matplotlib import pyplot as plt

[33] #rename images in the folder
directory_path = '/content/drive/MyDrive/DE_CW2_images'
def rename_player_images(directory_path,prefix="player", start_number=1, extension=".j
    """Renames images within a directory with a new prefix and sequential numbering"""
    for i, filename in enumerate(os.listdir(directory_path)):
        if filename.endswith(extension):
            destination_path = f"{prefix}{start_number}{extension}"
            os.rename(os.path.join(directory_path, filename), os.path.join(directory_p
            start_number += 1

[5] rename_player_images(directory_path)
```

First, I mounted the Google Drive to the Colab notebook and imported all the necessary libraries for image processing.

To rename the images, with the specified directory (directory\_path), the script in the image defines a function called rename\_player\_images by renaming all the images to the player starting from sequence by defaulting 1. To rename all the images in the folder I have included a loop(os.listdir). (os.rename) function renames the images from the original path to the new path. These functions helped to rename all the images in the folder.

## 1.b)Resizing the Images

```
✓ [34] #Images in the folder resizing
18 pre_processed_folder = '/content/drive/MyDrive/pre_processed_folder'
def resize_images(directory_path, pre_processed_folder, target_size=(500, 500)):
    """Resizes images in the directory_path folder to the target size and saves them t

    # Create the pre_processed_folder if it doesn't exist
    os.makedirs(pre_processed_folder, exist_ok=True)
    for filename in os.listdir(directory_path):
        if filename.endswith(".jpg"):#file extension
            # Read the image
            image_read_path = os.path.join(directory_path, filename)
            img = cv2.imread(image_read_path)

            # Resize the image
            resized_image = cv2.resize(img, target_size)

            # Save the resized image to the pre_processed_folder
            new_processed_path = os.path.join(pre_processed_folder, filename)
            cv2.imwrite(new_processed_path, resized_image)

    # Call the function to resize the images
    resize_images(directory_path,pre_processed_folder)
```

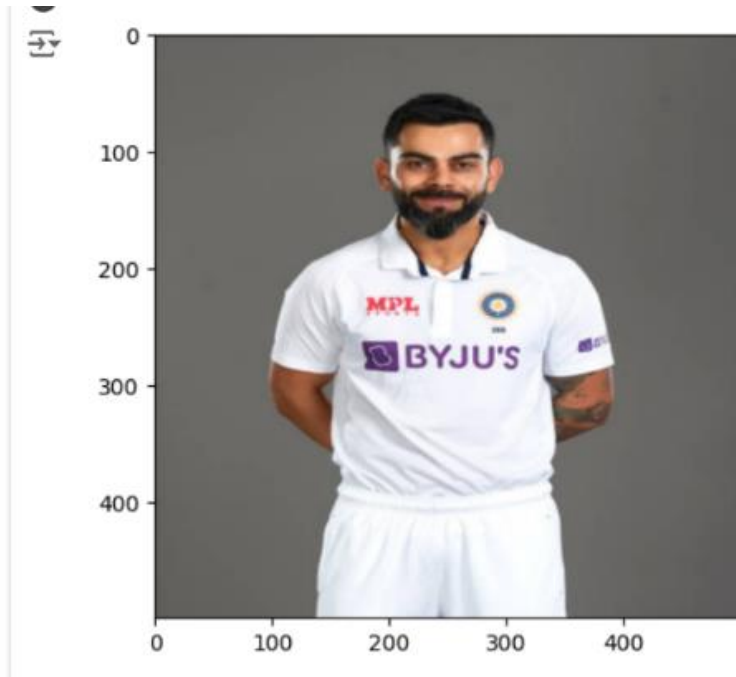
Resize\_images function is to resize all the images in the new folder with the target size of (500x500) pixels using openCV. Pre\_processed\_folder defines the path where the resized images gets saved with only file extension "jpg". To resize all the images in the new folder I have used a loop to the os.listdir(directory\_path), and entered the rest of the codes inside the loop to execute the loop. The above python script shows that from function resize\_image, I have resized the images from directory\_path and save them in the pre\_processed\_folder.

### A sample of resizing the image of player9.jpg

```
▶ #an example of resized image
example_resized_image = cv2.imread('/content/drive/MyDrive/DE_CW2_images/player9.jpg')

# Convert the image to RGB
image_rgb = cv2.cvtColor(example_resized_image, cv2.COLOR_BGR2RGB)
# Now resize the image
resized_image = cv2.resize(image_rgb, (500, 500))
cv2.imwrite('/content/drive/MyDrive/pre_processed_folder/player9.jpg',resized_image)
plt.imshow(resized_image)
plt.axis()
plt.show()
```

In the above code, I have tried to show a sample plot of on resized images in the pre\_process\_folder



Histogram for player9.jpg

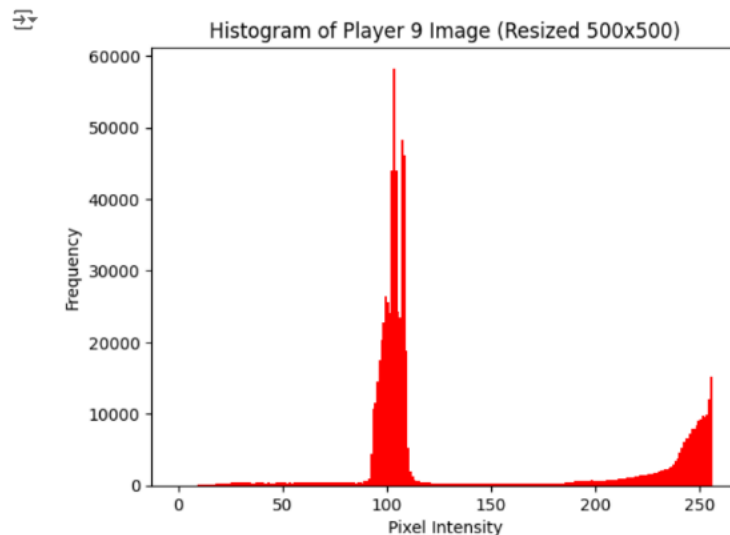
```
#plot histogram for player 9 image
image_path = '/content/drive/MyDrive/pre_processed_folder/player9.jpg'

# Reading the image
Player9Image = cv2.imread(image_path)

# Convert from BGR to RGB
img_rgb = cv2.cvtColor(Player9Image , cv2.COLOR_BGR2RGB)

# Plot the histogram
plt.hist(img_rgb.ravel(), bins=256, range=[0, 256], color='red')
plt.title('Histogram of Player 9 Image (Resized 500x500)')
plt.xlabel('Pixel Intensity')
plt.ylabel('Frequency')
plt.show()
```

This shows a histogram of player which I resized in the pre processing folder by converting the image from BGR to RGB



### 1.b)Denoising the Images

```
✓ 0s #denoise the images
def denoise_images(pre_processed_folder, sigma=(0.1)):
    """Denoises images in the pre_processed_folder directory using Gaussian blur"""

    for filename in os.listdir(pre_processed_folder):
        if filename.endswith(".jpg"):
            # Construct the full path to the image
            image_path = os.path.join(pre_processed_folder, filename)

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

            # Apply Gaussian blur for denoising
            denoised_image = cv2.GaussianBlur(image,(5,5),0)

            # Save the denoised image back to the same location, overwriting the origi
            new_processed_folder= os.path.join(pre_processed_folder,filename)
            cv2.imwrite(image_path, denoised_image)
```

Denoising is done to reduce the noise in in an image. Parameter sigma(0.1) is used to control the amount of blurring the image with a default value. To denoise all the images in the pre\_processed\_folder , I used a loop. If the filename ends with is “.jpg” it will denoise the images. cv2.imread reads the images using OpenCV function. Using Gaussian blur which is a image smoothing technique to denoise the images. This image denoising will make the franchise owners and stakeholders enhance the task of recognizing and analyzing.

## 1.c) Image Annotation

```
0s [35] #Metadata for cricket players
import pandas as pd
excel_file_extract = "/content/drive/MyDrive/data engineering annotation part.xlsx"
json_file = "/content/drive/MyDrive/data engineering annotation part.json"

#Reading the excel file
df = pd.read_excel(excel_file_extract)

#Converting the df(excel file) to JSON
df.to_json(json_file, orient='records', indent=4)
print(f"Excel file has been converted to JSON and saved as {json_file}.")
```

Excel file has been converted to JSON and saved as /content/drive/MyDrive/data engineer

```
0s [36] denoised_image = cv2.GaussianBlur(image,(5,5),0)

# Save the denoised image back to the same location, overwriting the origi
new_processed_folder= os.path.join(pre_processed_folder,filename)
cv2.imwrite(image_path, denoised_image)

+ Code + Text

0s [36] #Metadata for cricket players
import pandas as pd
excel_file_extract = "/content/drive/MyDrive/data engineering annotation part.xlsx"
json_file = "/content/drive/MyDrive/data engineering annotation part.json"

#Reading the excel file
df = pd.read_excel(excel_file_extract)

#Converting the df(excel file) to JSON
df.to_json(json_file, orient='records', indent=4)
print(f"Excel file has been converted to JSON and saved as {json_file}.")

Excel file has been converted to JSON and saved as /content/drive/MyDrive/data engineer

0s [37] from skimage.feature import graycomatrix, graycoprops
from skimage.measure import label, regionprops_table

def extract_image_features(pre_processed_folder):

    image = cv2.imread(pre_processed_folder)
    img_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
```

xt\_folder) data engineering annotation part.json X ...

```
1 [
2 {
3     "Picture number":"player1",
4     "Keywords":"Bhuvaneshwar_Kumar,India,Bowler",
5     "Description":"A right arm medium swing fast
6 },
7 {
8     "Picture number":"player10",
9     "Keywords":"Tim David,Australia,Batsman",
10    "Description":"A right hand middle order pow
11 },
12 {
13    "Picture number":"player11",
14    "Keywords":"Henrich_Klassen,South_Africa,Bats
15    "Description":"A right top order loft batsman
16 },
17 {
18    "Picture number":"player12",
19    "Keywords":"Naseem_Shah,Pakistan,Bowler",
20    "Description":"A right arm fasting bowler"
21 },
22 {
23    "Picture number":"player13",
24    "Keywords":"Glenn_Philips,New_Zealand,Allroun
25    "Description":"A batting allrounder"
26 },
27 {
28    "Picture number":"player14",
```

I have annotated players details for all 50 images through excel sheets manually. I have copied the csv file path to excel file to the variable. To read the excel file I have imported the necessary pandas library to read the csv file. With `pd.read_excel`, stored the data in a dataframe and converted it to a json file. After converting the dataframe to a json file, printed a sentence as "Excel file has been converted to JSON and saved to drive".



## 1.d)Extracting Image Features

```
✓ 0s ▶ from skimage.feature import graycomatrix, graycoprops
from skimage.measure import label, regionprops_table

def extract_image_features(pre_processed_folder):

    image = cv2.imread(pre_processed_folder)
    img_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

    #Intensity colour features
    mean_intensity = np.mean(img_rgb, axis=(0,1))
    norm_intensity = mean_intensity / 255.0

    #color moments
    mean = np.mean(img_rgb, axis=(0,1))
    std = np.std(img_rgb, axis=(0,1))
    skewness = np.mean(np.power(img_rgb - mean,3), axis=(0,1)) / np.power(std, 3)

    #dictionarize the color features
    C_features = {
        'mean_intensity': mean_intensity.tolist(),
        'norm_intensity': norm_intensity.tolist(),
        'mean': mean.tolist(),
        'std': std.tolist(),
        'skewness':skewness.tolist()
    }
    C_moments = np.concatenate([mean,std,skewness])
```

To extract features for the images firstly, I have imported the necessary libraries.(Skimage.feature)library is used to extract texture features and (skimage.measure) is used to analyze images. Next I have defined the extract\_image\_features function to pre\_processed\_folder as input. Through cv2 I have read and converted the images from BGR to RGB.

Next, I extracted colour intensity features.Mean\_intensity calculates the mean average of a each color channel of the entire image.norm\_intensity normalize the intensity from range 0 to 1 by dividing them by 255 pixels.

Then, I calculated the color moments. I calculated mean value, standard deviation, measured the skewness of the color distribution of each color channel and then I dictionarized all the features to store the features.

## 1.d)Texture features

```
✓ 0s #texture features
gray_img = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
glcm = graycomatrix(gray_img, [1], [0, np.pi/4, np.pi/2, 3*np.pi/4], levels=256, nor
texture_features = np.array([
    graycoprops(glcm, 'contrast')[0,0],
    graycoprops(glcm, 'dissimilarity')[0,0],
    graycoprops(glcm, 'homogeneity')[0,0],
    graycoprops(glcm, 'energy')[0,0],
    graycoprops(glcm, 'correlation')[0,0],
    graycoprops(glcm, 'ASM')[0,0]
])
```

For texture features I have converted the RGB image to gray scale. I have extracted few texture features from glcm using graycomatrix. Contrast, dissimilarity, homogeneity, energy, correlation, ASM are the texture feature which is extracted by using graycomatrix. [0,0] index is used to select specific values from graycoprops.

## 1.d)Shape Features

```
✓ 0s ▶ #shape features
gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
ret, thresh = cv2.threshold(gray_image,0,255,cv2.THRESH_BINARY+cv2.THRESH_OTSU)
edges = cv2.Canny(thresh,100,200)
#find contours
contours,_ = cv2.findContours(edges, cv2.RETR_EXTERNAL,cv2.CHAIN_APPROX_SIMPLE)

largest_contour = max(contours, key=cv2.contourArea)

area = cv2.contourArea(largest_contour)
perimeter = cv2.arcLength(largest_contour, True)

#centroid
M = cv2.moments(largest_contour)
if M["m00"] !=0:
    cX = int(M["m10"] / M["m00"])
    cY = int(M["m01"] / M["m00"])
    centroid = (cX, cY)
else:
    centroid = (0,0)

#bounding box
x,y,w,h = cv2.boundingRect(largest_contour)
bounding_box = (x,y,w,h)
shape_features = {
    'area': area,
    'perimeter': perimeter,
    'centroid': centroid,
    'bounding_box' : bounding_box
```

In shaping feature, I have tried to find the largest contour (most distinctive shape). I converted the Image file to grayscale through "cv2.cvtColor". "cv2.threshold" is used to create a binary image from the converted grayscale image file. "cv2.canny" is used to find the edges in the threshold image, this is a crucial step for outlining the shape of the image. "cv2.findContours" is used to find the outlines in the edged images. "cv2.RETR\_EXTERNAL" is a method which finds only the outer boundaries of an image and it will ignore the holes that's inside of an image. I have calculated various shape features like area, perimeter, centroid and bounding box.

Storing all the extracting features

```
}  
#storing all the features  
features = {  
    'C_features':C_features,  
    'texture_features':texture_features.tolist(),  
    'shape_features':shape_features  
}  
return features
```

Here, all the calculated features are stored.

Extracting all the features to Images and creating a JSON file

```
✓ 9s ▶ pre_processed_folder = '/content/drive/MyDrive/pre_processed_folder'  
extracted_features = {}  
  
#substituting all extracted features for all images  
for filename in sorted(os.listdir(pre_processed_folder)):  
    if filename.endswith(".jpg"):  
        image_path = os.path.join(pre_processed_folder, filename)  
        features = extract_image_features(image_path)  
        list_features = {  
            'filename':filename,  
            'C_features':features['C_features'],  
            'texture_features':features['texture_features'],  
            'shape_features':features['shape_features']  
        }  
        extracted_features[filename]=list_features  
  
✓ 0s [39] with open ('image_Feature_extract.json', 'w')as json_file:  
        json.dump(extracted_features,json_file,indent = 4)
```

In the above diagram I have extracted all features applied to images and transferred them to a JSON file called "image\_feature\_extract.json". I have included a (os.listdir) loop to iterate through each filename in pre\_processed\_folder.

## MongoDB upload process for the images

```
✓ 8s !python -m pip install "pymongo[srv]"

Collecting pymongo[srv]
  Downloading pymongo-4.10.1-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl
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 7.0 MB/s eta 0:00:00
  Downloading pymongo-4.10.1-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (
    1.4/1.4 MB 32.1 MB/s eta 0:00:00
Installing collected packages: dnspython, pymongo
Successfully installed dnspython-2.7.0 pymongo-4.10.1

[13] import pymongo
     from pymongo import MongoClient

[14] Image_connection = pymongo.MongoClient("mongodb+srv://msuhailnoor:cricket@clusterde.ed
     db = Image_connection["CBIR"]

[27] try:
     print(db.list_collection_names())
     except Exception as e:
     print(f"connection failed: {e}")

['Image_Metadata', 'Processed_Images', 'Image_Annotation']
```

In the above diagram, I have coded and connected the MongoDB database to colab notebook. MongoDB is a type of database where, we can store our data adjustbally. Also its easy to understand and store our data. Instead of tables and databases we can store our queries as a JSON file in MongoDB. To start the process, I have installed the pymongo[srv] and imported the necessary libraries called pymongo and MongoClient.

Then, created a cluster in MongoDB and created a database called CBIR for images. To connect the MongoDB to colab, we need to copy the path which is specified to my database and paste it to colab to connect the server.

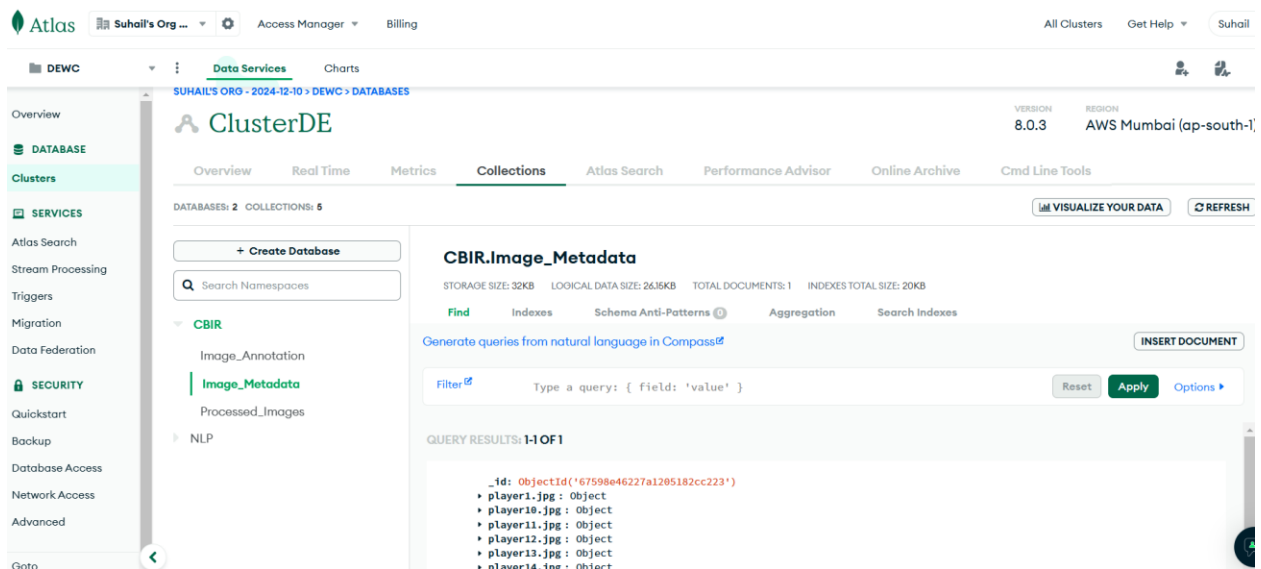
I created three collections under database CBIR in MongoDB. When I connect my MongoDB server to colab I did a testing to check the collection was successful and appeared in the colab.

## Uploading the Image feature extractions to MongoDB

```
#defined the collection
image_metadata_collection = db["Image_Metadata"]

#load Json metadata file
with open('/content/image_Feature_extract.json') as file:
    image_data = json.load(file)

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



In the above figures, I have defined a MongoDB collection “Image\_Metadata” in my MongoDB database. In Image\_Metadata collection I have uploaded the feature extraction JSON file to MongoDB database. The conditional statement specifies if the JSON file contains multiple image metadata’s using insert\_many insert all the data into the collection. If there’s only one entry insert it by using insert\_one to insert it.

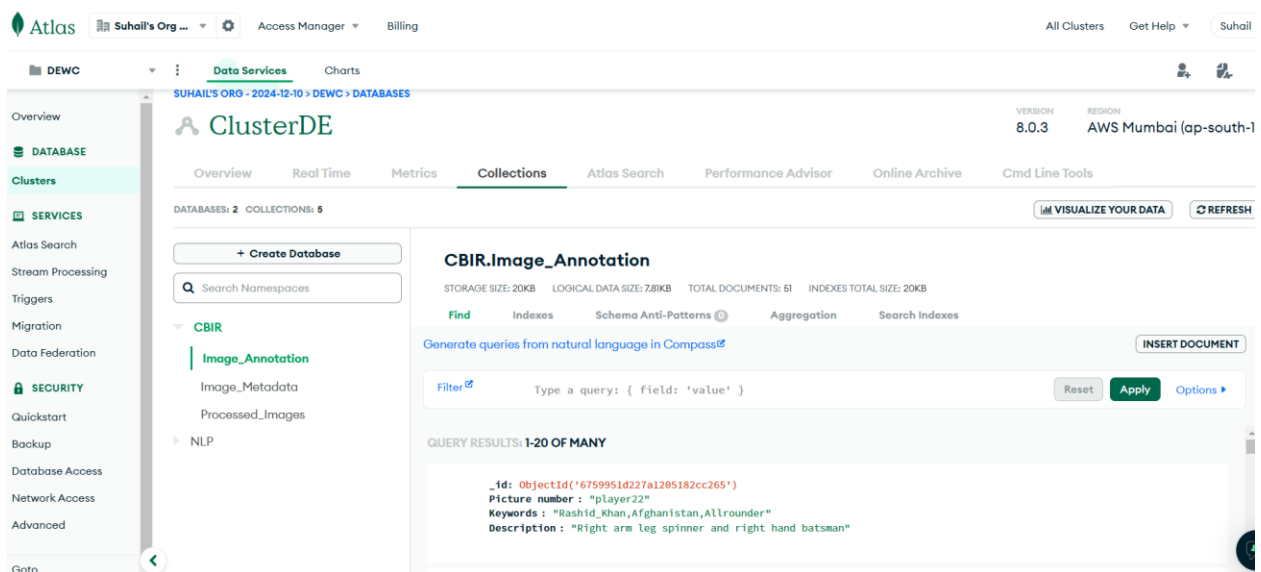
The second figure shows the Image feature extractions which i uploaded to Image Metadata collection.

## Uploading the Image Annotations to MongoDB

```
#defined the collection
image_annotation_collection = db["Image_Annotation"]

#load Json metadata file
with open('/content/drive/MyDrive/data engineering annotation part.json') as file:
    image_data = json.load(file)

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



In the above diagram, I have defined a MongoDB collection called Image\_Annotation. In this collection I have uploaded all the cricket players keywords and descriptions by converting it to a JSON file. I have copied and uploaded the JSON file path to Image\_Annotation.

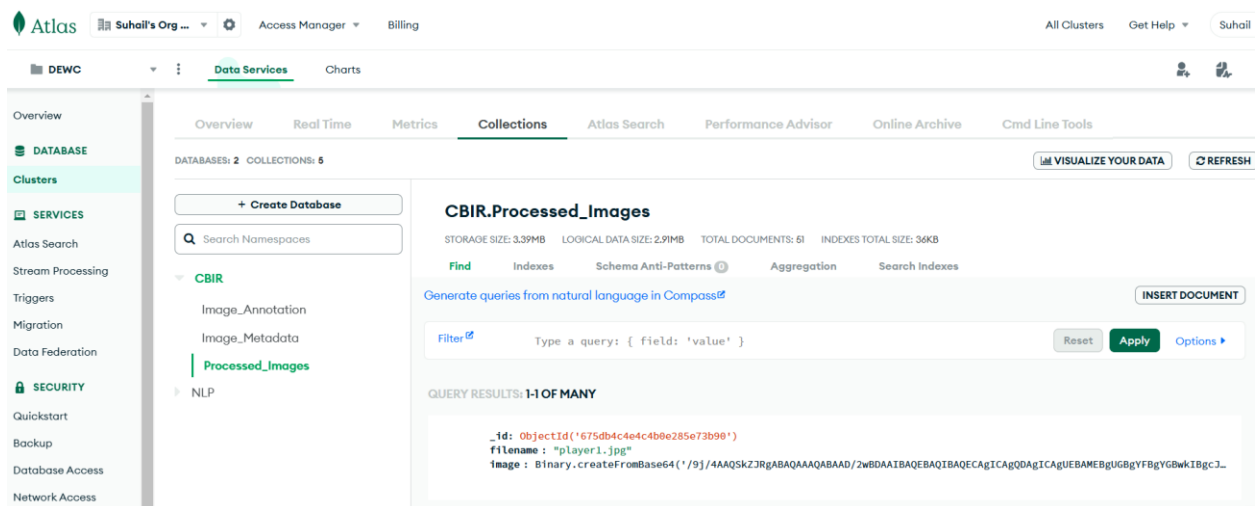
The conditional statement specifies if the JSON file contains multiple image annotations's by using insert\_many insert all the data into the collection. If there's only one Image\_Annotation insert it by using insert\_one to insert it.

## Uploading the Processed Image to MongoDB

```
[ ] processed_collection = db["Processed_Images"]
```

```
from bson import Binary

for filename in os.listdir(pre_processed_folder):
    if filename.endswith(".jpg"):
        with open(os.path.join(pre_processed_folder, filename), "rb") as image_file:
            binary_image = Binary(image_file.read())
            processed_collection.insert_one({
                "filename": filename,
                "image": binary_image
            })
```



In the above figure, I have tried to store image files as binary in MongoDB. It runs through all the files and read them as binary. Also, I have ran a loop to run the files in the folder.



Test check to clarify the query worked in colab

```
#test to check the whether the query is working
query = {"filename": "player9.jpg"}
result = processed_collection.find_one(query)
if result:
    print("Query successful!")
    print(result)
else:
    print("Query failed.")
```

Query successful!  
{ '\_id': ObjectId('675db4cfe4c4b0e285e73bc2'), 'filename': 'player9.jpg' }

This code is used to ensure that the file matches the query. If found a matching document in the processed\_collection , print “Query successful”.

## MongoDB for CBIR

The screenshot displays the MongoDB Atlas web interface. On the left, a sidebar menu shows navigation options like Overview, DATABASE, Clusters, SERVICES, and SECURITY. The main panel is titled 'ClusterDE' and shows the 'Collections' tab for the 'CBIR' database. A table lists three collections: Image\_Annotation, Image\_Metadata, and Processed\_Images, with their respective document counts and sizes. The interface also includes a search bar, a 'Create Database' button, and a 'Create Collection' button.

Collection Name	Documents	Logical Data Size	Avg Document Size	Storage Size	Indexes	Index Size	Avg Index Size
Image_Annotation	51	7.81KB	157B	20KB	1	20KB	20KB
Image_Metadata	1	26.15KB	26.15KB	32KB	1	20KB	20KB
Processed_Images	51	2.91MB	58.46KB	4MB	1	36KB	36KB

The above diagram shows the cluster and the CBIR database I created in MongoDB. Under CBIR database I have created 3 collections called Image\_Annotations, Image\_Metadata and Processed\_Images. In Image\_Annotations , I have uploaded the Player annotations JSON file. In Image\_Metadata collection I have uploaded the Image feature extractions and in Processed\_Images collection , I have uploaded the Pre\_processed\_folder.

## Question 02

### Question 02

```
[ ] import nltk
    import string
    from nltk.corpus import stopwords
    from nltk.tokenize import word_tokenize
    from nltk.stem import PorterStemmer
    from nltk.stem import WordNetLemmatizer
```

```
[ ] nltk.download('punkt')
    nltk.download('stopwords')
    nltk.download('wordnet')
```

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

Import all the necessary libraries for natural languaging process(NLP).

Uploading the csv to colab and generate textfiles

```
[ ] Path_CSV = '/content/drive/MyDrive/DE_customer_review.csv'
    df= pd.read_csv(Path_CSV)

    df_random = df.sample(n=50, random_state=42)
    df_random = df_random.reset_index(drop=True)
```

```
[ ] print(df_random.columns)
```

```
↳ Index(['reviewText'], dtype='object')
```

```
▶ OG_text_directory = '/content/drive/MyDrive/OG_text_directory'
  os.makedirs(OG_text_directory, exist_ok=True)

  for index, row in df_random.iterrows():
      file_name = f'customer_review_{index + 1}.txt'
      file_path = os.path.join(OG_text_directory, file_name)

      with open(file_path, 'w', encoding='utf-8') as file:
          text = row['reviewText']
          file.write(text)
      print(f"Saved {file_name}")
```

```
↳ Saved customer_review_1.txt
  Saved customer_review_2.txt
  Saved customer_review_3.txt
  Saved customer_review_4.txt
  Saved customer_review_5.txt
  Saved customer review 6.txt
```

After downloading the amazon customer review dataset, I have uploaded the csv to my drive to generate 50 text files automatically. To read the csv file, I assigned a path to the variable "Path\_CSV". To read the path I have used Pandas library called df and from "df.sample" I generated 50 random sample text files from the customer review dataset and assigned it to df\_random dataframe. "random\_state=42" is seed for random number generator, which ensures whenever I run the code it give the same number of random samples.

After generating 50 random texfiles ,I assigned it to a new varaibale called OG\_text\_directory and named the textfiles as customer\_review by using f'string.The iterrows loop runs through each row of the df\_random dataframe."w" indicates the file is open for writing."file.write(text)" transfer the extracted texts to the OG\_text\_directory file. Then I asked to print "save" when the text files are saved in the new folder. .


## Normalization

```
pre_processed_text = '/content/drive/MyDrive/pre_processed_text'
os.makedirs(pre_processed_text, exist_ok=True)
#lowercasing all the text files
def lowercasing_text_files(OG_text_directory, pre_processed_text):
    for filename in os.listdir(OG_text_directory):
        if filename.endswith('.txt'):
            file_path = os.path.join(OG_text_directory, filename)
            processed_file_path = os.path.join(pre_processed_text, filename)
            with open(file_path, 'r') as file_path, open(processed_file_path, 'w') as output_file:
                for line in file_path:
                    text = line
                    lowercased_text = line.lower()
                    output_file.write(lowercased_text)
```


```
[ ] lowercasing_text_files(OG_text_directory, pre_processed_text)
```

Normalization is converting a text into a consistent or standard form. There are so many techniques in normalization, but here, I have only used the lowercasing technique. So, lowercasing refers to non-capitalizing the text in the text process,Which all the capital letters in the text file will be lowercased.In the above code I have defined a new function and copied the pre\_processed\_text folder path. Also I have lowercased the original text files and the pre processed text file using this code.

## Tokenization

```
 #tokenization
def tokenize_text_files(pre_processed_text):
    """tokenization all the text files"""
    for filename in os.listdir(pre_processed_text):
        if filename.endswith('.txt'):
            file_path = os.path.join(pre_processed_text, filename)
            with open(file_path, 'r', encoding = 'utf-8') as file:
                text = file.read()
                tokens = word_tokenize(text)
                token_string = ' '.join(tokens)
                with open(file_path, 'w', encoding = 'utf-8') as output_file:
                    for token in tokens:
                        output_file.write(token + '\n')
```

```
[ ] nltk.download('punkt_tab')
    nltk.download('punkt')
    tokenize_text_files(pre_processed_text)
```

```
 [nltk_data] Downloading package punkt_tab to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt_tab.zip.
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
```

Tokenization is a technique of breaking down a sentence texts into individual words. I have ran and downloaded packages,essential libraries to tokenize a word.For tokenization I have defined the tokenize\_text\_files function for pre\_processed\_text folder as input. To tokenize all the text files, I implemented a loop which will run through the all text files in the folder. 'r' indicated that the file is openfor read.I used '/n' which will shape all he tokenized words to new lines by breaking the down the sentences.

## Vectorization

### BoW(Bag of Words)

```
#Vectorization
from sklearn.feature_extraction.text import CountVectorizer
textdata = []
for filename in os.listdir(pre_processed_text):
    if filename.endswith('.txt'):
        file_path = os.path.join(pre_processed_text, filename)
        with open(os.path.join(pre_processed_text, filename), 'r', encoding = 'utf-8') as file:
            textdata.append(' '.join(file.read().split()))

[ ] #bag of words
Bow_vector = CountVectorizer()
Bow_vector.fit(textdata)

CountVectorizer
CountVectorizer()

[ ] Bow_matrix = Bow_vector.transform(textdata)
print(Bow_matrix.toarray())

[[0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 ...
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 2 0]
 [0 0 1 ... 0 0 0]]
```

Vectorization is the process of converting a text to numerical representations. In here, I have presented text vectorization using the CountVectorizer from sklearn. I have taken the Bag of Words (BoW) as the method for vectorization process. When each text file is represented by a vector which indicated the frequency of words in that text file. I have ran a loop to execute the vectorization to all the textfiles. So when the vocabulary is created through the CountVectorizer, it will transfer the text into numerical vectors based on the vocabulary. These vectorized texts can be used for tasks like deep analysis.

In the Bow\_matrix code snippet, I have used the BoW method by transferring the text file to numerical data and show the output in a matrix. Using a matrix to analyze is easier to have a deep investigation on the text file.

## Term Frequency Inverse Document

```
[ ] #Term frequency-inverse document
!pip install scikit-learn
from sklearn.feature_extraction.text import TfidfVectorizer

Requirement already satisfied: scikit-learn in /usr/local/lib/python3.10/dist-packages (1.5.2)
Requirement already satisfied: numpy>=1.19.5 in /usr/local/lib/python3.10/dist-packages (from scikit-learn) (1.26.4)
Requirement already satisfied: scipy>=1.6.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn) (1.13.1)
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn) (1.4.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn) (3.5.0)

vectorizer = TfidfVectorizer()
tfidf_vector = vectorizer.fit_transform(textdata)
print(vectorizer.get_feature_names_out().tolist())

['10', '1080p', '12gb', '15', '16gb', '2014', '25', '28gb', '32', '32gb', '34', '48', '4k', '52gb', '59gbs', '64gb', '8gb', 'about', 'absolutely']

[ ] print(tfidf_vector.toarray())

[[0. 0. 0. ... 0. 0. 0. ]
 [0. 0. 0. ... 0. 0. 0. ]
 [0. 0. 0. ... 0. 0. 0. ]
 ...
 [0. 0. 0. ... 0. 0. 0. ]
 [0. 0. 0. ... 0. 0.16649398 0. ]
 [0. 0. 0.15204496 ... 0. 0. 0. ]]
```

Term Frequency Inverse document is also a numerical statistic way which shows how crucial a word is to a document. This is also a main technique which is used by various parties. In here first, I have installed the scikit-learn and imported tfidfVectorizer. The (tfidf\_vector) variable code is used to perform the Term frequency inverse document on collection of text files. “transform()” can transform the text data into a matrix of TF IDF vector. The value of the element is the TF-IDF sequence for the text file.

## Text Metadata

```
[ ] textfile_metadata = {
    "vectorization_methods":{
        "bag_of_words":"BoW is a method for text vectorization in NLP.It represents text as a numerical vector by counting frequency in each word",
        "tf_idf":"tf-idf is a numerical measure to reflect the importance of a word to a corpus.This measures how frequently a word appears in a corpus"
    },
    "vectorization_process":{
        "tfidf_vector" : " it prints out the list of words it used to represent your text data numerically.",
        "BoW_matrix":"converts text into matrix for analysis."
    },
    "sentiment_labels":{}
}
```

From the textfile\_metadata dictionary, I have included three keys, such as vectorization methods, vectorization process and sentiment labels. First two keys include descriptions.

## Labeling

```
!pip install textblob
from textblob import TextBlob

for filename in os.listdir(pre_processed_text):
    if filename.endswith(".txt"):
        file_path = os.path.join(pre_processed_text, filename)
        with open(file_path, 'r', encoding="utf-8") as file:
            text = file.read()
            blob = TextBlob(text)
            sentiment_polarity = blob.sentiment.polarity

            if sentiment_polarity > 0:
                label = "positive"
            elif sentiment_polarity < 0:
                label = "negative"
            else:
                label = "neutral"

            textfile_metadata["sentiment_labels"][filename] = label

Requirement already satisfied: textblob in /usr/local/lib/python3.10/dist-packages (0.17.1)
Requirement already satisfied: nltk>=3.1 in /usr/local/lib/python3.10/dist-packages (from textblob) (3.9.1)
Requirement already satisfied: click in /usr/local/lib/python3.10/dist-packages (from nltk>=3.1->textblob) (8.1.7)
Requirement already satisfied: joblib in /usr/local/lib/python3.10/dist-packages (from nltk>=3.1->textblob) (1.4.2)
Requirement already satisfied: regex>=2021.8.3 in /usr/local/lib/python3.10/dist-packages (from nltk>=3.1->textblob) (2024.9.11)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages (from nltk>=3.1->textblob) (4.66.6)

[ ] with open('textfile_metadata.json', 'w') as f:
    json.dump(textfile_metadata, f, indent=4)
```

From this code I have tried to sentiment analysis on text file and assign labels to each text. From above code, I have generated the labels to each column through numeric values. I have run a loop to run all the files in the directory. I have extracted the sentiment polarity from TextBlob. Values greater than 0 is positive, Values less than 0 is negative and values with 0 are neutral.

## MongoDB for Text Files(NLP)

```
[ ] text_connection = pymongo.MongoClient('mongodb+srv://msuhailnoor:cricket@clusterde.edsjl.mongodb.net/?retryWrites=true&w=majority&appName=Clusterde')
Dbase = text_connection["NLP"]

!test connection
try:
    print(Dbase.list_collection_names())
except Exception as e:
    print(f"Connection failed: {e}")

['doc_metadata', 'processed_doc']
```

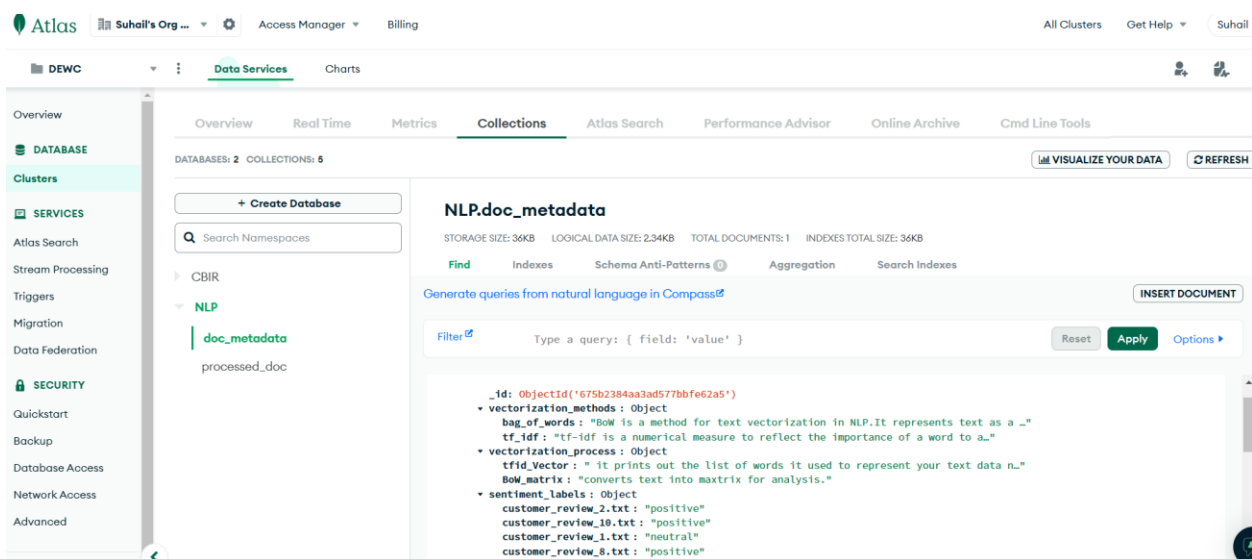
The above code snippet is used to connect the text files to (NLP) data MongoDB database. I have copied the cluster connection path and ran it with a new variable database in colab. And next I tried the test and it was successful which I can see the two collections which I created in the NLP database in MongoDB.

## Uploading Doc Metadata text JSON file to MongoDB

```
doc_metadata_collection = Dbase["doc_metadata"]

#load json metadata text file
with open('/content/textfile_metadata.json') as file:
    doc_data = json.load(file)

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



In the above diagram, I have defined a MongoDB text file collection called doc\_metadata to inset the metadata and the labelings. In this collection I have copied the path texfile\_metadata.json to doc\_metadata.This JSON file includes all the customer review text with the text\_metadata and the labelings with positive,negative and neutral.

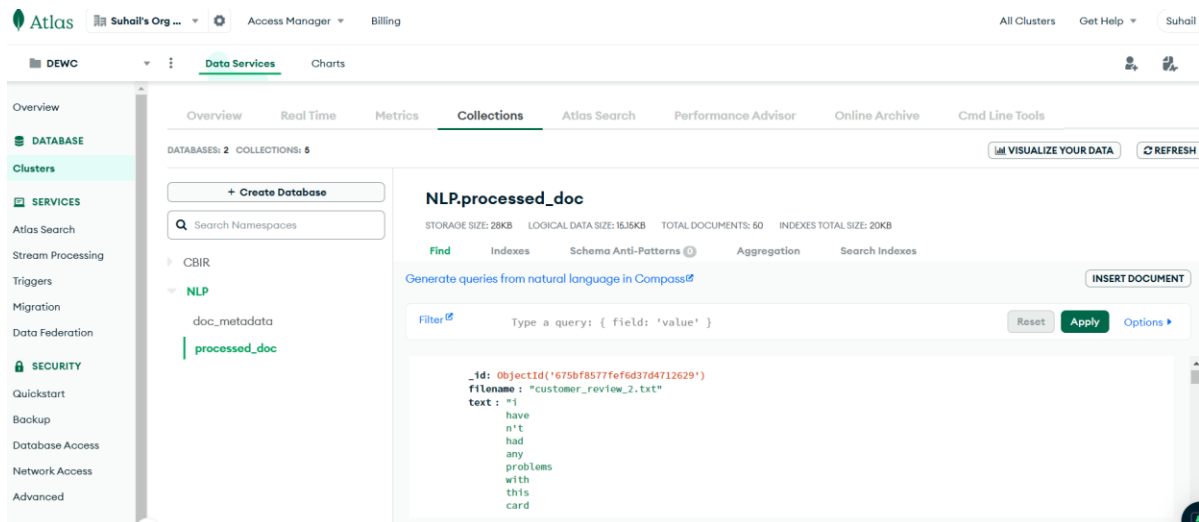
The conditional statement specifies if the JSON file contains multiple text file's by using insert\_many insert all the data into the collection. If there's only one text files insert it by using insert\_one to insert it.



## Uploading Processed Doc file to MongoDB

```
processed_doc_collection = Dbase["processed_doc"]

[ ] import os
    from datetime import datetime
    for filename in os.listdir(pre_processed_text):
        file_path = os.path.join(pre_processed_text, filename)
        if os.path.isfile(file_path):
            with open(file_path, "r") as file:
                text = file.read()
                processed_document = {
                    "filename": filename,
                    "text": text,
                }
                processed_doc_collection.insert_one(processed_document)
```



In the above diagram, I have defined a MongoDB text file collection (NLP) called processed\_doc. Pre\_processed\_text contains all the normalization, tokenization and vectorization features which I used. By, creating a loop I have ran all these features in each and every text file and transferred it to the processed\_doc in Mongo DB.

## MongoDB for NLP

The screenshot displays the ClusterDE MongoDB management interface. The left sidebar contains navigation options: Overview, DATABASE, Clusters, SERVICES, Atlas Search, Stream Processing, Triggers, Migration, Data Federation, SECURITY, Quickstart, Backup, and Database Access. The main panel shows the 'Data Services' tab with a breadcrumb path: SUHAIL'S ORG - 2024-12-10 > DEWC > DATABASES. The 'Collections' tab is active, showing a table of collections for the 'NLP' database. The table has columns: Collection Name, Documents, Logical Data Size, Avg Document Size, Storage Size, Indexes, Index Size, and Avg Index Size. Two collections are listed: 'doc\_metadata' with 1 document and 'processed\_doc' with 50 documents. A 'CREATE COLLECTION' button is visible in the top right of the table area.

Collection Name	Documents	Logical Data Size	Avg Document Size	Storage Size	Indexes	Index Size	Avg Index Size
doc_metadata	1	2.34KB	2.34KB	36KB	1	36KB	36KB
processed_doc	50	15.29KB	314B	28KB	1	20KB	20KB

In the above diagram, I have created another database for Natural Language processing(NLP). Under NLP, I have created 2 collections called doc\_metadata and processed\_doc. In doc\_metadata, I have uploaded the textfile metadata JSON file which contains the metadata and labeling. In processed\_doc, I have uploaded the pre\_processing\_text folder which includes the normalization,tokenization and vectorization.This shows that the texfiles have been successfully uploaded to MongoDB.