UNIVERSITY OF WESTMINSTER

INFORMATICS INSTITUTE OF TECHNOLOGY

5DATA005C.1

Data Engineering

Name: Sethmika Dias Bellana

UoW ID: w1985751

IIT ID: 20221906

Contents

1. Part 1 - Database for CBIR

1.1. Objective

The objective here is to create a database suitable for content-based image retrieval system. CBIR is a technique that used the visual features of images to find similar images in a large database. Here, through MATLAB a diverse collection of colour images has been collected, pre-processed, annotated with meta data, and extracted relevant image features such as such as colour, pixel intensities, texture, and shape. Then, structured the meta data and extracted features into a JSON file to serve as a foundation for a CBIR system.

A possible business scenario to use this database could be a non-profit organization or a research institution dedicated to wildlife conservation and ecological research could use a vast collection of animal images captured. The organization wants a CBIR database to analyzing these images to extract meaningful features for research. This data could be used for a much larger database of animal images for various purposes such as animal categorization and identification.

1.2. Image Collection

The image collection consists of 50 colour images also known as RGB images. In this case, the image collection is focused on a particular type of object, i.e. animals. The collection consists of 50 colour animal images gathered from 'Pexels' website (Pexels, 2024). The 50 images were downloaded while saving their image address for reference, this has been saved as an Xcel sheet in the name of 'url.xlsx'. The images were downloaded into a folder named 'original_images', then images were then stored in GITHUB repository called 'w1985751_DataEngineering_Coursework2' (w1985751, 2023).

1.3. Image pre-processing

Image pre-processing was carried out to enhance the quality of the image and standardize them. Image pre-processing was crucial because, while observing the images most of them had noise in them, to a certain extent and all the images were in different sizes.

Therefore a 'for loop' was used iterate through every image in the 'orginal_images' folder. As shown in *Figure 1* the path of the 'original_images' folder was given to the OriginalFolder variable and then a list of the images was obtained. The `fullfile` function was used to create the full path to the directory and `dir` function to create a structure array of the image list with the 'jpeg' extension. 'jpeg' extension was specifically used because all the images were already in the same extension. Then the loop was run for the `length` of the ImageList, which was 50 time as the 'original_images' only contained 50 images.

Next inside the 'for loop' the Original Images were read with the `imread` function, for image pre-processing and feature extraction.

```
%Specify the folders
OriginalFolder = 'original_images';
% Get the list of the names of the image files with jpeg extension
ImageList = dir(fullfile(OriginalFolder, '*.jpeg'));

% A loop for image pre-processing and image feature extraction for each 50 images
for i = 1:length(ImageList)
    %% Image Pre-processing
    InputImagePath = fullfile(OriginalFolder, ImageList(i).name);
    OriginalImage = imread(InputImagePath);
```

Figure 1 - For Loop

With the loop each image was resized to 500 x 500 pixels using the `imresize` function, as shown in *Figure 2* and saved it into 'ResizedImage' variable. This was done due to two reasons. Most images had a much larger size and resizing to 500 x 500 pixels was downsizing the to a smaller resolution, this was done to reduce memory usage during the analysis

otherwise the processing large image would take longer periods of time. The second reason was to give all the image a standard size to each image, to get a consistent feature extraction.

```
%% Image Pre-processing
InputImagePath = fullfile(OriginalFolder, ImageList(i).name);
OriginalImage = imread(InputImagePath);

% Resizing the image to 500x500 pixels
ResizedImage = imresize(OriginalImage, [500, 500]);
```

Figure 2 - Image Resizing

Next, when observing the images most of them consisted of noise, and noise is an unwanted variation in pixel values that degrades the quality of the image. Therefore, the images were denoised using the `imgaussfilt` function, as shown in *Figure 3 and* saved to 'DenoisedImage' variable. This function performs a 2-D Gaussian filtering on every image, using a Gaussian smoothing kernel with standard deviation of 1. A much lower value was not chosen as it would not reduce the noise of the image much, on the other hand a higher value was not chosen as it would blur the image much. Therefore, the standard deviation value of '1' was derived after experimenting several times.

```
% Reduce the noise of the image using a Gaussian Filter
DenoisedImage = imgaussfilt(ResizedImage, 1);
```

Figure 3- Image Denoising

To see the image with noise and the denoised image side by side, a human in the loop situation has been deployed. As shown in *Figure 4* this code was used to prompt the user to enter 'yes' or 'no' for the question 'Do you wish to see the original and denoised image side by side?'.

```
%Ask the user if they wish to see the original and denoised image side by side
DenoiseImages = input('Do you wish to see the original and denoised image side by side? (yes/no): ', 's');
DenoiseImages = strcmpi(DenoiseImages, 'yes');
```

Figure 4 - Human in the Loop to view Denoised Image

Then the following output will appear on the command window, as shown in *Figure 5.*

```
Do you wish to see the original and denoised image side by side? (yes/no):
```

Figure 5 - Command Window denoise question

If 'yes' is typed on the command window, the resized image which contains noise and the denoised image will be shown side by side for each image, as shown in *Figure 7* If 'no' is typed then the image will not be shown side by side, but the denoising will take place in both situations. This step was included for the convenience of the user, as now the user has the option to avoid viewing multiple figure windows.

```
if DenoiseImages
   figure();
   subplot(1,2,1);
   imshow(ResizedImage);
   subplot(1,2,2)
   imshow(DenoisedImage);
end
```

Figure 6 - Denoise image in a figure window

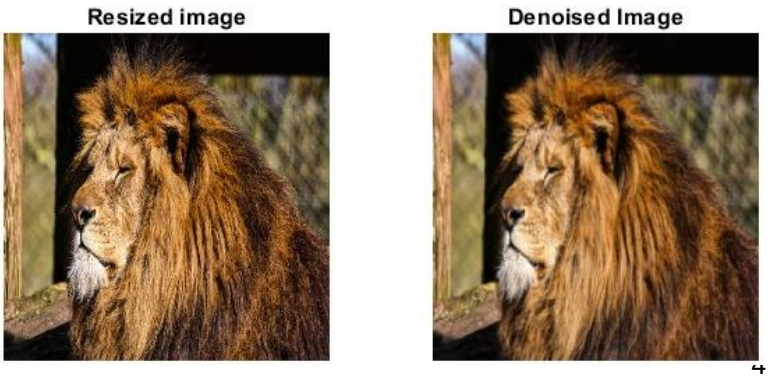

Figure 7 - Resized image with noise vs Denoised image in the figure window of image_22

While observing the original images, none of the images required rotation. Although there could be certain images that could require image rotation when this CBIR system is used for another image dataset. Therefore, like before human interaction was required to confirm if the images required rotation.

```
% Ask the user if they want to rotate the images
RotateImages = input('Do you want to rotate the images? (yes/no): ', 's');
RotateImages = strcmpi(RotateImages, 'yes');
```

Figure 7 - Human in the loop to rotate the image

After the the user press enter from the previously mentioned denoising question, question shown in *Figure 8* will appear in the command window

```
Do you want to rotate the images? (yes/no):
```

Figure 8 - Command Window Rotate question

If 'yes' then the figure window will load the first denoised image, as shown in *Figure 9*, if required the rotation angle(in degrees) must be entered in the command window.

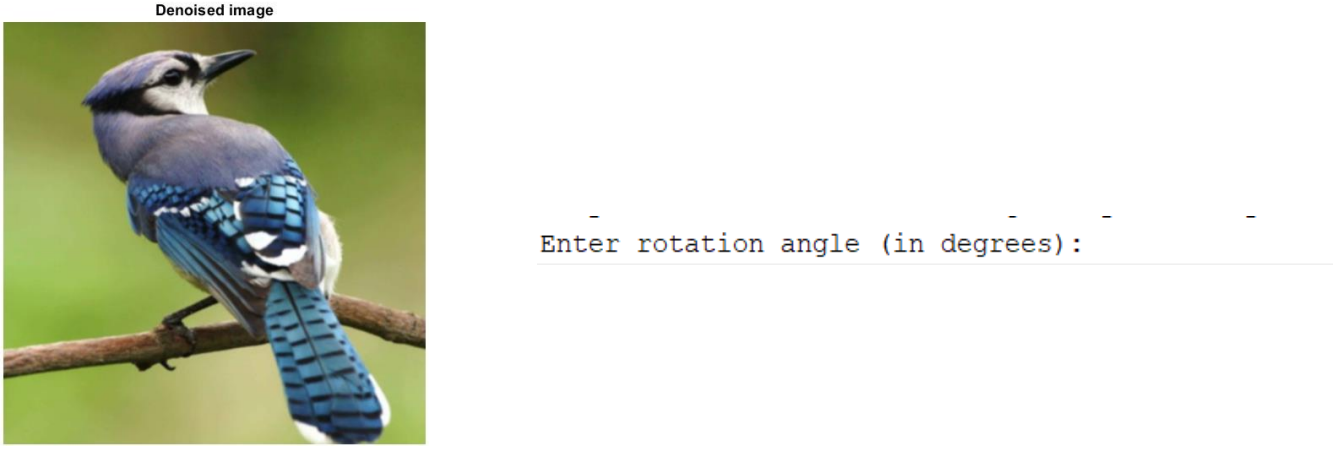

Figure 9

The code shown in *Figure 10,* will rotate the image based on the rotation angle entered. This will continue on for each image in the original_folder and save the image as 'RotatedImage', if there is no rotation 'DenoisedImage' is considered as the 'RotatedImage'.

```
% Rotate the image if the user wants to rotate
if RotateImages
    % Opening an interactive window to rotate the image
    figure(2);
    imshow(DenoisedImage);
    title('Denoised image');

    % Prompt the user to enter the rotation angle
    RotationAngle = input('Enter rotation angle (in degrees): ');
    RotatedImage = imrotate(DenoisedImage, RotationAngle);
    close;
else
    RotatedImage = DenoisedImage;
end
```

Figure 10

'RotatedImage', is the image that has been resized to 500 x 500 pixels, denoised and then rotated if needed. This is the final pre-processed image that is used for feature extraction. Therefore, the final image has been saved systematically in the 'preprocessed_images' folder.

```
% Saving the pre-processed images in 'preprocessed images' folder
OutputFileName = sprintf('image_%02d.jpeg', i);
OutputFilePath = fullfile(PreprocessedFolder, OutputFileName);
imwrite(RotatedImage, OutputFilePath);
```

Figure 11 - Saving the pre-processed images

1.4. Image Annotation

Automatic image annotations require pre-trained models, which is inaccessible to students. Therefore, manual automation was done. After observing each pre-processed image in the folder image each image was annotated with keywords (i.e. animal name and type) and with a short description of the image. These data were put to Xcel file named as 'image_tags.xlsx'. Then this table was loaded into MATLAB using `readtable` function, as shown in Figure *12* and then these image annotations were later added to the JSON file.

```
% Loading the image with metadata (tags)
ImageTags = 'image_tags.xlsx';
TagsTable = readtable(ImageTags);
ImageId = TagsTable.ImageId;
Tags = TagsTable.Tags;
Description = TagsTable.Description;
```

Figure 12 - Image Annotations

1.5. Image Feature Extraction

Images features were then obtained to present the visual content of an image in a quantitative and structured way. In this case colour features, texture features and shape features were gathered and stored in a JSON file.

Firstly, to get the colour features the colour pixel intensities of each channel (Red, Green, Blue) mean and norm were obtained. RGB images' colour channels were obtained with the '(:,:,1)' notation for the first which is the red channel, and so on, as shown in *Figure 13*.

```
%% Image Feature Extraction

% Read the preprocessed image
PreprocessedImage = imread(OutputFilePath);

% MEAN of channel pixel intensity - using mean2 function
RedMean = mean2(PreprocessedImage(:,:,1));
GreenMean = mean2(PreprocessedImage(:,:,2));
BlueMean = mean2(PreprocessedImage(:,:,3));

% NORM of channel pixel intensity - using norm function
RedNorm = norm(double(PreprocessedImage(:,:,1)));
GreenNorm = norm(double(PreprocessedImage(:,:,2)));
BlueNorm = norm(double(PreprocessedImage(:,:,3)));
```

Figure 13 – Colour Features

Next, the texture features such as GLCM, local entropy, local Standard Deviation, local binary pattern (LBP) and histogram of oriented gradients (HOG) features were obtained. For this the images were required, converting each image to grey scale, this was done using the `rgb2gray` function and named the image 'GreyImage'.

Then the GLCM features such as contrast, energy, correlation, and homogeneity were obtained as shown in *Figure 14*. This was obtained using four different offsets to capture the spatial relationships between pixels in different direction. The matrix of [0 1; -1 1; -1 0; -1 -1] was chosen as there is no dominant orientation for the images therefore it will take the pixel relationship horizontally, vertically, and diagonally, and since the images are resized to 500x500 used a smaller offset of 1 pixel displacement to avoid losing information.

```
%*Texture Features*

% GLCM features - statistics of the GLCM
GreyImage = rgb2gray(PreprocessedImage);
Glcm = graycomatrix(GreyImage, 'Offset', [0 1; -1 1; -1 0; -1 -1]);
GlcmStats = graycoprops(Glcm, {'Contrast', 'Energy', 'Correlation', 'Homogeneity'});
```

Figure 14 – GLCM Texture Features

Next the Local entropy was measured to obtain the randomness or disorder of an image, local standard deviation to measure the variation of pixel values and LBP to receive the local texture comparing the intensities of each pixel with the neighbouring pixels. Each of these texture feature's mean was derived to measure the average intensity, standard deviation to measure the variability or smoothness, and the skewness to measure the directionality of a texture.

```
% Entropy
E = entropyfilt(GreyImage); % Local entropy of the image
MeanE = mean(E(:)); % Mean of the local entropy
StdE = std(E(:)); % Standard deviation of the local entropy
SkewE = skewness(E(:)); % Skewness of the local entropy

% Standard deviation
S = stdfilt(GreyImage); % Local standard deviation of the image
MeanS = mean(S(:)); % Mean of the local standard deviation
StdS = std(S(:)); % Standard deviation of the local standard deviation
SkewS = skewness(S(:)); % Skewness of the local standard deviation

% LBP
LBP = extractLBPFeatures(GreyImage); % Extract the LBP features
MeanLBP = mean(LBP); % Mean of the LBP features
StdLBP = std(LBP); % Standard deviation of the LBP features
SkewLBP = skewness(LBP); % Skewness of the LBP features
```

Figure 15 - Other Texture features

Next the shape features of the grey scale images were measured. To simplify the image data by reducing it to binary form, we binarize the image using `imbinarize` function. This is done to give more attention to the shape of the object. The global binarization method was used to binarize the images of animals because most of them have high contrast with the background.

```
% *Shape Features*
% Binarize the image using golbal threshold
BinaryImages = imbinarize(GreyImage, 'global');
```

Figure 16 - Binarize the image

Next the binary image was inverted based on the average of all pixel values in the binary image. Meaning if the image has more white pixels than black pixels using `imcomplement` function the colours will be reversed, otherwise the image will be kept as it is. This was done so that for all most all the image's the object to be black and the background to be white.

```
% Invert the image if mean binary value is greater than 0.5
MeanBinaryImage = mean(BinaryImages(:));
if MeanBinaryImage > 0.5 %% 0.5 was used as it the midpoint of the binary range values
    % Use imcomplement to invert the image
    ImImages = imcomplement(BinaryImages);
else
    ImImages = BinaryImages;
end
```

Figure 17 - Invert Binary Image

The `bwconncomp` function was used to obtain the total number of connected components of the inverted image in order to separate the objects from the background and from each other.

```
% To obtain the total number of connected components
CcImages = bwconncomp(ImImages);
```

Figure 18 - Connected Components

Next measured the gemoatric properties of the image using the function. Area measured the number of pixels in the connnected componets, Centroid measured the the coordinates of the center of mass, Circulirty measured how circular the connected component is, majoraxislength and minoraxislength, Eccentricity measured how elongated the connected component is, Orientation measured the angle between the x-axis and the major axis, FilledArea measured the number of pixels in the connected component, Perimeter measured the length of the boundary of the connected component.

```
% To measure geometric properties of the image
GeoFeatures = regionprops(CcImages, 'Area', 'Centroid', 'Circularity', 'MajorAxisLength', ...
    'MinorAxisLength', 'Eccentricity', 'Orientation', 'FilledArea', 'Perimeter');
% Calculate mean,standard deviation and skewness for each measurement
% Area
MeanArea = mean([GeoFeatures.Area]); % Mean area
StdArea = std([GeoFeatures.Area]);   % Standard deviation of the area
SkewArea = skewness([GeoFeatures.Area]); % Skewness of the area

%Centroid
MeanCentroid = mean([GeoFeatures.Centroid]); % Mean Centroid
StdCentroid = std([GeoFeatures.Centroid]);   % Standard deviation of the Centroid
SkewCentroid = skewness([GeoFeatures.Centroid]); % Skewness of the Centroid

% Circularity
MeanCircularity = mean([GeoFeatures.Circularity]); % Mean circularity
StdCircularity = std([GeoFeatures.Circularity]);   % Standard deviation of the circularity
SkewCircularity = skewness([GeoFeatures.Circularity]); % Skewness of the circularity

% Major Axis Length
MeanMajorAxisLength = mean([GeoFeatures.MajorAxisLength]); % Mean major axis length
StdMajorAxisLength = std([GeoFeatures.MajorAxisLength]);   % Standard deviation of the major axis length
SkewMajorAxisLength = skewness([GeoFeatures.MajorAxisLength]); % Skewness of the major axis length

% Minor Axis Length
MeanMinorAxisLength = mean([GeoFeatures.MinorAxisLength]); % Mean minor axis length
StdMinorAxisLength = std([GeoFeatures.MinorAxisLength]);   % Standard deviation of the minor axis length
SkewMinorAxisLength = skewness([GeoFeatures.MinorAxisLength]); % Skewness of the minor axis length

% Eccentricity
MeanEccentricity = mean([GeoFeatures.Eccentricity]); % Mean eccentricity
StdEccentricity = std([GeoFeatures.Eccentricity]);   % Standard deviation of the eccentricity
SkewEccentricity = skewness([GeoFeatures.Eccentricity]); % Skewness of the eccentricity

% Orientation
MeanOrientation = mean([GeoFeatures.Orientation]); % Mean orientation
StdOrientation = std([GeoFeatures.Orientation]);   % Standard deviation of the orientation
SkewOrientation = skewness([GeoFeatures.Orientation]); % Skewness of the orientation
% Filled Area
MeanFilledArea = mean([GeoFeatures.FilledArea]); % Mean filled area
StdFilledArea = std([GeoFeatures.FilledArea]);   % Standard deviation of the filled area
SkewFilledArea = skewness([GeoFeatures.FilledArea]); % Skewness of the filled area

% Perimeter
MeanPerimeter = mean([GeoFeatures.Perimeter]); % Mean perimeter
StdPerimeter = std([GeoFeatures.Perimeter]);   % Standard deviation of the perimeter
SkewPerimeter = skewness([GeoFeatures.Perimeter]); % Skewness of the perimeter
```

Figure 19 - Shape Features

Finally structure was created to store each image's meta data: image annotations, image features and the relavnt image ID along with the image address.

```
%% Save in JSON File

% Create a structure for the current image
CurrentImageInfo.ImageId = ImageId{i};
CurrentImageInfo.WebImageAddress = WebImageAddress{i};
CurrentImageInfo.GithubOriginalImageAddress = GithubOriginalImageAddress {i};
CurrentImageInfo.GithubPreprocessedImageAddress = GithubPreprocessedImageAddress{i};
CurrentImageInfo.Tags = Tags{i};
CurrentImageInfo.Description = Description{i};
CurrentImageInfo.Size = CcImages.ImageSize;

% Channel pixel intensities
CurrentImageInfo.ColourFeatures.Mean = struct('Red', RedMean, 'Green', GreenMean, 'Blue', BlueMean);
CurrentImageInfo.ColourFeatures.Normalization = struct('Red', RedNorm, 'Green', GreenNorm, 'Blue', BlueNorm);

% Texture Features
CurrentImageInfo.TextureFeatures.GLCM = GlcmStats;
CurrentImageInfo.TextureFeatures.Entropy = struct('Mean', MeanE, 'Standard_Deviation', StdE, 'Skewness', SkewE);
CurrentImageInfo.TextureFeatures.StandardDeviation = struct('Mean', MeanS, 'Standard_Deviation', StdS, 'Skewness', SkewS);
CurrentImageInfo.TextureFeatures.LBP = struct('Mean', MeanLBP, 'Standard_Deviation', StdLBP, 'Skewness', SkewLBP);

% Shape Features
CurrentImageInfo.ShapeFeatures.Area = struct('Mean', MeanArea, 'Standard_Deviation', StdArea, 'Skewness', SkewArea);
CurrentImageInfo.ShapeFeatures.Centroid = struct('Mean', MeanCentroid, 'Standard_Deviation', StdCentroid, 'Skewness', SkewCentroid);
CurrentImageInfo.ShapeFeatures.Circularity = struct('Mean', MeanCircularity, 'Standard_Deviation', StdCircularity, 'Skewness', SkewCircularity);
CurrentImageInfo.ShapeFeatures.MajorAxisLength = struct('Mean', MeanMajorAxisLength, 'Standard_Deviation', StdMajorAxisLength, 'Skewness', SkewMajorAxisLength);
CurrentImageInfo.ShapeFeatures.MinorAxisLength = struct('Mean', MeanMinorAxisLength, 'Standard_Deviation', StdMinorAxisLength, 'Skewness', SkewMinorAxisLength);
CurrentImageInfo.ShapeFeatures.Eccentricity = struct('Mean', MeanEccentricity, 'Standard_Deviation', StdEccentricity, 'Skewness', SkewEccentricity);
CurrentImageInfo.ShapeFeatures.Orientation = struct('Mean', MeanOrientation, 'Standard_Deviation', StdOrientation, 'Skewness', SkewOrientation);
CurrentImageInfo.ShapeFeatures.FilledArea = struct('Mean', MeanFilledArea, 'Standard_Deviation', StdFilledArea, 'Skewness', SkewFilledArea);
CurrentImageInfo.ShapeFeatures.Perimeter = struct('Mean', MeanPerimeter, 'Standard_Deviation', StdPerimeter, 'Skewness', SkewPerimeter);

% Store the current image info in the cell array
FeaturesArray{i} = CurrentImageInfo;
```

Figure 20 - Structure for metadata

Finally the loop was stopped and the collected data in the CurrentImageInfo structure for all the images, was used created a JSON file named 'w1985751_part1.json'.

```
end

% Convert the cell array to JSON format
JsonString = jsonencode(FeaturesArray, 'PrettyPrint', true);

% Write the JSON string to the file
JsonFilePath = 'w1985751_part1.json';
Fid = fopen(JsonFilePath, 'w');
fprintf(Fid, '%s\n', JsonString);
fclose(Fid);
```

Figure 21 - JSON file to save the meta data and features

1.6. Database Design

In here a No SQL database was created to store the JSON file created above. The chosen NoSQL database was MongoDB as it easily facilitates to store JSON files and images as GridFS . Due to the unavailability of licence to use MATLAB to connect to MongoDB python was used.

In the Jupyter Notebook, named as 'w1985751_DataLoading_part1.ipynb', the pymongo package was installed and imported the MongoClient and ServerApi packages in order to connect with MongoDB compass.

```
In [3]: pip install pymongo
        0f226a3cbfb54263d02bb421c7f2adc136520729c2c689c1e5/dnspython-2.4.2-py3-none-any.whl.metadata
          Downloading dnspython-2.4.2-py3-none-any.whl.metadata (4.9 kB)
        Downloading pymongo-4.6.1-cp311-cp311-win_amd64.whl (472 kB)
           ---------------------------------------- 0.0/472.7 kB ? eta -:--:--
           -- ------------------------------------- 30.7/472.7 kB ? eta -:--:--
           --- ------------------------------------ 41.0/472.7 kB 653.6 kB/s eta 0:00:01
           --- ------------------------------------ 41.0/472.7 kB 653.6 kB/s eta 0:00:01
           --- ------------------------------------ 41.0/472.7 kB 653.6 kB/s eta 0:00:01
           --- ------------------------------------ 41.0/472.7 kB 653.6 kB/s eta 0:00:01
           ----- ---------------------------------- 61.4/472.7 kB 192.5 kB/s eta 0:00:03
           --------- ------------------------------ 122.9/472.7 kB 399.4 kB/s eta 0:00:01
           --------- ------------------------------ 122.9/472.7 kB 399.4 kB/s eta 0:00:01
           ------------ --------------------------- 163.8/472.7 kB 392.8 kB/s eta 0:00:01
           ------------ --------------------------- 163.8/472.7 kB 392.8 kB/s eta 0:00:01
           ------------ --------------------------- 163.8/472.7 kB 392.8 kB/s eta 0:00:01
           ---------------- ---------------------- 194.6/472.7 kB 346.5 kB/s eta 0:00:01
           ------------------ -------------------- 225.3/472.7 kB 362.0 kB/s eta 0:00:01
           -------------------- ------------------ 256.0/472.7 kB 413.7 kB/s eta 0:00:01
           ----------------------- --------------- 286.7/472.7 kB 420.8 kB/s eta 0:00:01

In [19]: import json
         from pathlib import Path

In [4]: from pymongo.mongo_client import MongoClient

In [5]: from pymongo.server_api import ServerApi
```

Figure 22 - Python importing packages

Next the connection string was obtained from the MongoDB cloud as shown in *Figure 23* and loaded it to the variable 'uri' in the Jupyter Notebook as shown in *Figure 24.*

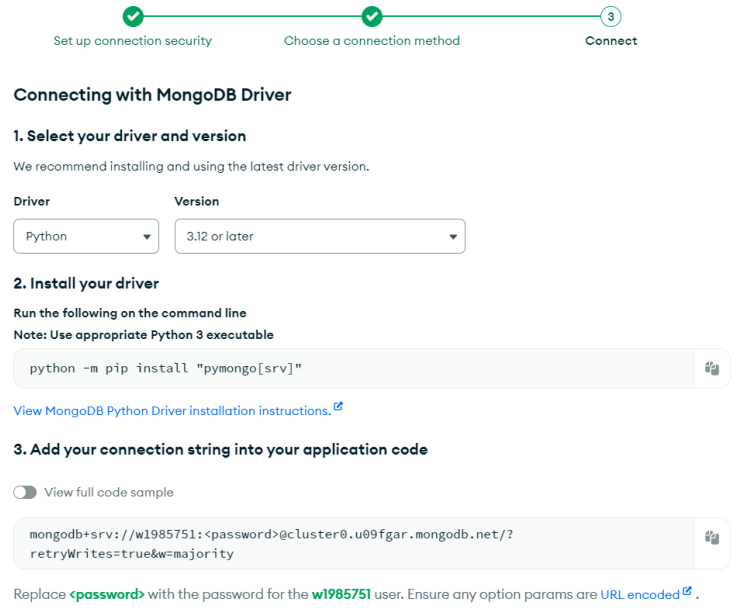

Figure 23

```
In [4]: uri = "mongodb+srv://w1985751:Sethmika2005@cluster0.u09fgar.mongodb.net/?retryWrites=true&w=majority"

In [5]: client = MongoClient(uri)
```

Figure 24

Then a database names ‘CBIR’ was created with a collection named ‘w1985751_JSON_File’ and then the JSON file created using MATLAB named ‘w1985751_part1.json’ was loaded and uploaded to the connection.

```
In [13]: database_name = "CBIR"

In [14]: # Create a database in MongoDB Compass
         db = client[database_name]

In [16]: #collection name
         collection_name = "w1985751_JSON_File"

In [17]: json_file_path = Path("w1985751_part1.json")

In [19]: #Load the json file
         with open(json_file_path, 'r') as file:
             data = json.load(file)

In [20]: # Insert the data into the collection
         db[collection_name].insert_many(data)

Out[20]: InsertManyResult([ObjectId('659cf54f9b0574466679af6c'), ObjectId('659cf54f9b0574466679af6d'), ObjectId('659cf54f9b0574466679af6e'), ObjectId('659cf54f9b0574466679af6f'), ObjectId('659cf54f9b0574466679af70'), ObjectId('659cf54f9b0574466679af71'), ObjectId('659cf54f9b0574466679af72'), ObjectId('659cf54f9b0574466679af73'), ObjectId('659cf54f9b0574466679af74'), ObjectId('659cf54f9b0574466679af75'), ObjectId('659cf54f9b0574466679af76'), ObjectId('659cf54f9b0574466679af77'), ObjectId('659cf54f9b0574466679af78'), ObjectId('659cf54f9b0574466679af79'), ObjectId('659cf54f9b0574466679af7a'), ObjectId('659cf54f9b0574466679af7b'), ObjectId('659cf54f9b0574466679af7c'), ObjectId('659cf54f9b0574466679af7d'), ObjectId('659cf54f9b0574466679af7e'), ObjectId('659cf54f9b0574466679af7f'), ObjectId('659cf54f9b0574466679af80'), ObjectId('659cf54f9b0574466679af81'), ObjectId('659cf54f9b0574466679af82'), ObjectId('659cf54f9b0574466679af83'), ObjectId('659cf54f9b0574466679af84'), ObjectId('659cf54f9b0574466679af85'), ObjectId('659cf54f9b0574466679af86'), ObjectId('659cf54f9b0574466679af87'), ObjectId('659cf54f9b0574466679af88'), ObjectId('659cf54f9b0574466679af89'), ObjectId('659cf54f9b0574466679af8a'), ObjectId('659cf54f9b0574466679af8b'), ObjectId('659cf54f9b0574466679af8c'), ObjectId('659cf54f9b0574466679af8d'), ObjectId('659cf54f9b0574466679af8e'), ObjectId('659cf54f9b0574466679af8f'), ObjectId('659cf54f9b0574466679af90'), ObjectId('659cf54f9b0574466679af91'), ObjectId('659cf54f9b0574466679af92'), ObjectId('659cf54f9b0574466679af93'), ObjectId('659cf54f9b0574466679af94'), ObjectId('659cf54f9b0574466679af95'), ObjectId('659cf54f9b0574466679af96'), ObjectId('659cf54f9b0574466679af97'), ObjectId('659cf54f9b0574466679af98'), ObjectId('659cf54f9b0574466679af99'), ObjectId('659cf54f9b0574466679af9a'), ObjectId('659cf54f9b0574466679af9b'), ObjectId('659cf54f9b0574466679af9c'), ObjectId('659cf54f9b0574466679af9d')], acknowledged=True)
```

After the database was created and the JSON file was pushed the JSON document was shown in the MongoDB compass as shown in *Figure 25.*

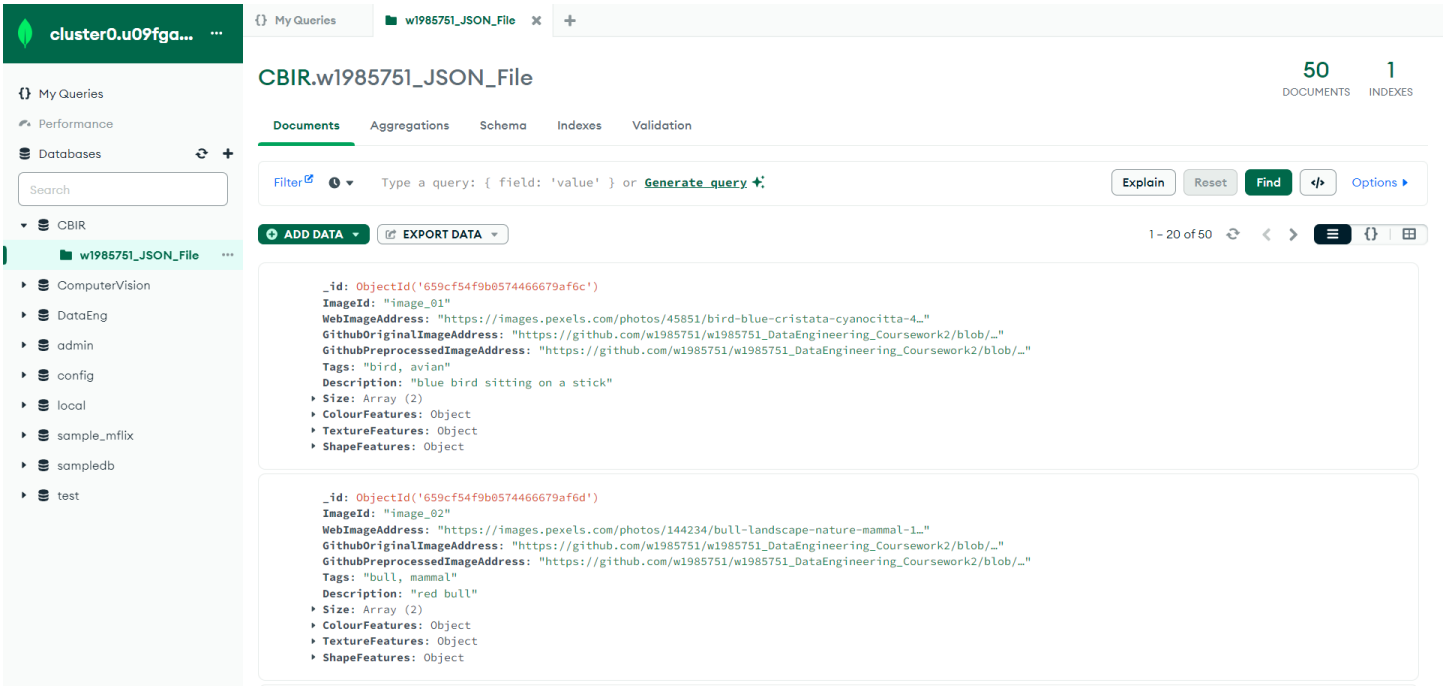

Figure 25 - JSON document in MongoDB compass

As shown above in the JSON file there are 50 objects, each with meta data and features as per the structure created, I using MATLAB as shown in *Figure 20.* Further details of structure is shown in the following *Table 1*:

Variable	Description
ImageId	Identifier for the image
WebImageAddress	Web address of the image
GithubOriginalImageAddress	GitHub address of the original image
GithubPreprocessedImageAddress	GitHub address of the preprocessed image
Tags	Image Annotations of the image
Description	Description of the image
Size	Size information of the image
ColourFeatures.Mean	Mean pixel intensities for Red, Green, and Blue channels
ColourFeatures.Normalization	Normalized pixel intensities for Red, Green, and Blue channels
TextureFeatures.GLCM	Gray Level Co-occurrence Matrix (GLCM) statistics
TextureFeatures.Entropy	Entropy statistics for texture
TextureFeatures.StandardDeviation	Standard deviation statistics for texture
TextureFeatures.LBP	Local Binary Pattern (LBP) statistics
ShapeFeatures.Area	Area statistics for the image
ShapeFeatures.Centroid	Centroid statistics for the image
ShapeFeatures.Circularity	Circularity statistics for the image
ShapeFeatures.MajorAxisLength	Major axis length statistics for the image
ShapeFeatures.MinorAxisLength	Minor axis length statistics for the image
ShapeFeatures.Eccentricity	Eccentricity statistics for the image
ShapeFeatures.Orientation	Orientation statistics for the image
ShapeFeatures.FilledArea	Filled area statistics for the image
ShapeFeatures.Perimeter	Perimeter statistics for the image

Table 1

Next the images were uploaded to MongoDB in using GridFS, this created 2 documents in 2 collections: one file for the image chunk named as 'fs.chunks, shown in *Figure 27* and the other for the meta data named as 'fs.files', shown in *Figure 28*. The python code for it is shown in *Figure 26.*

```
In [34]: collection = client[database_name][collection_name]

In [35]: import os
         import pymongo
         from gridfs import GridFS
         from PIL import Image
         from io import BytesIO

In [42]: # Access the default GridFS collection
         fs = GridFS(db)

         # Folder path containing the images
         images_folder_path = "preprocessed_images"

In [43]: # Iterate through the images in the folder
         for image_filename in os.listdir(images_folder_path):
             image_path = os.path.join(images_folder_path, image_filename)

             # Open and read the image as binary data
             with open(image_path, 'rb') as image_file:
                 image_data = image_file.read()

             # A dictionary to store the image name and type
             metadata = {
                 'filename': image_filename,
                 'content_type': 'image/jpeg',}

             # Store the image in MongoDB using GridFS
             file_id = fs.put(image_data, **metadata)
```

Figure 26

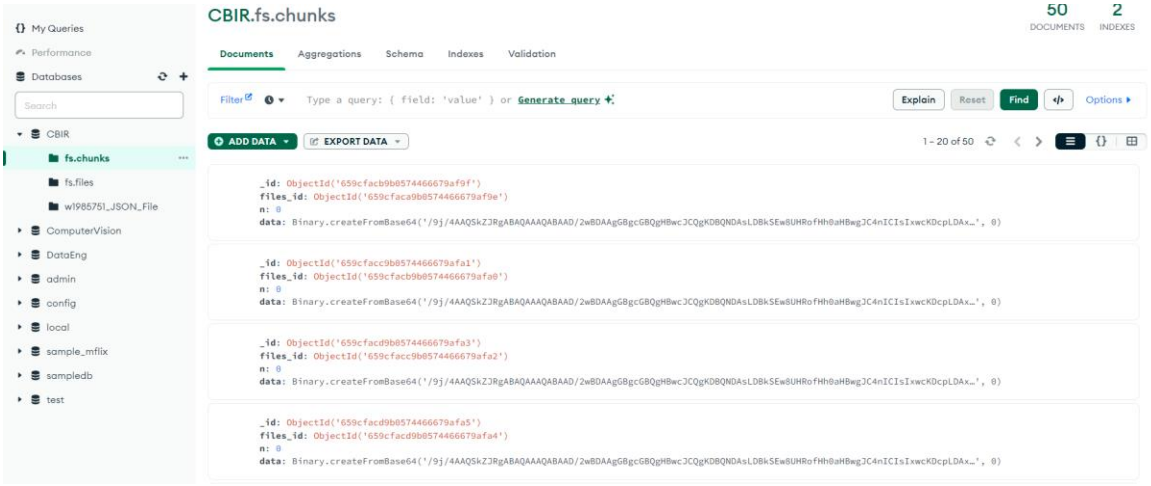

Figure 27 - fs.chunks

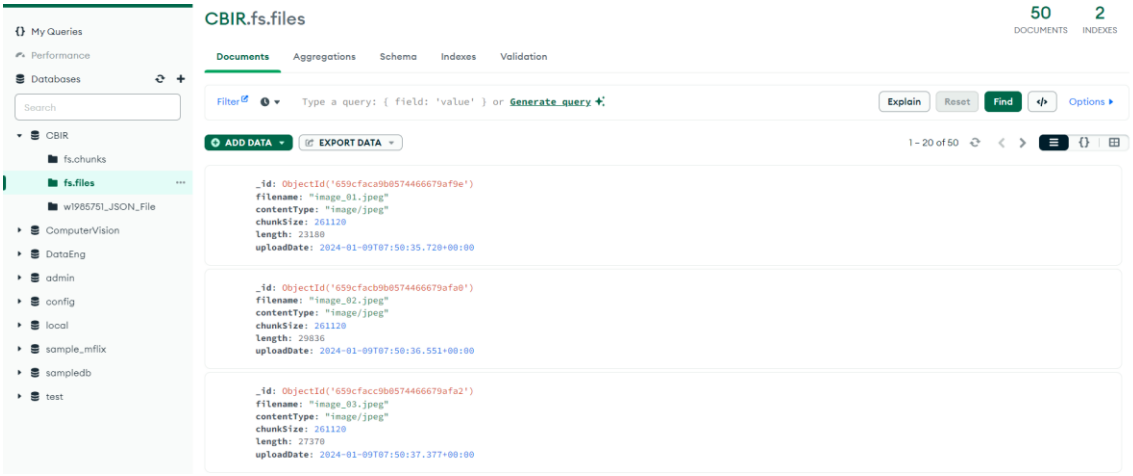

Figure 28 - fs.files

As requested the image feature extraction CSV files, were created by exporting the JSON file document as CSV dump in MongoDB, as shown in Figure 29. All the meta data and features extracted is in the file named 'CBIR.w1985751_JSON_File.csv'. If the colour features, texture features and shape features are required to be in separate CSV, please view 'color_features.csv' , texture_features.csv',

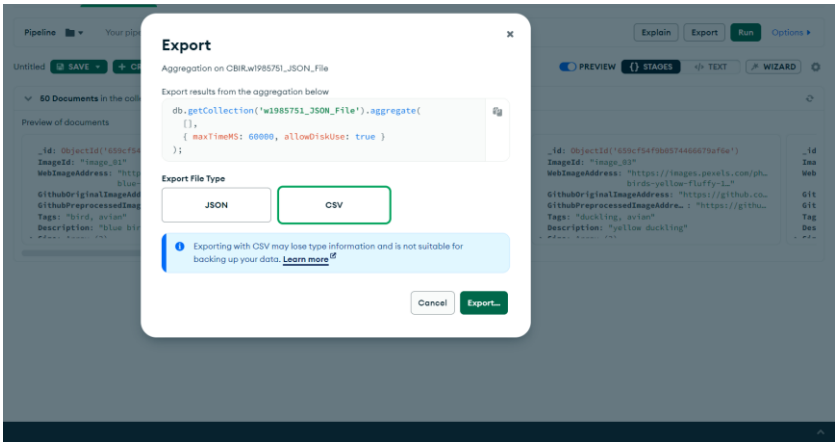

Figure 30 - Export as CSV file

```
import pandas as pd

# Read the JSON file extracted from MongoDB
csv_filename = "CBIR.w1985751_JSON_File.csv"
df = pd.read_csv(csv_filename)

# Extracting the neccsary column
image_id_col = df['ImageId']
color_features = df[['ImageId', 'ColourFeatures.Mean.Red', 'ColourFeatures.Mean.Green', 'ColourFeatures.Mean.Blue',
                     'ColourFeatures.Normalization.Red', 'ColourFeatures.Normalization.Green', 'ColourFeatures.Normalization.Blue

texture_features = df[['ImageId', 'TextureFeatures.GLCM.Contrast[0]', 'TextureFeatures.GLCM.Contrast[1]', 'TextureFeatures.GLCM.
                       'TextureFeatures.GLCM.Correlation[0]', 'TextureFeatures.GLCM.Correlation[1]', 'TextureFeatures.GLCM.Correl
                       'TextureFeatures.GLCM.Energy[0]', 'TextureFeatures.GLCM.Energy[1]', 'TextureFeatures.GLCM.Energy[2]', 'Tex
                       'TextureFeatures.GLCM.Homogeneity[0]', 'TextureFeatures.GLCM.Homogeneity[1]', 'TextureFeatures.GLCM.Homoge
                       'TextureFeatures.Entropy.Mean', 'TextureFeatures.Entropy.Standard_Deviation', 'TextureFeatures.Entropy.Ske
                       'TextureFeatures.StandardDeviation.Mean', 'TextureFeatures.StandardDeviation.Standard_Deviation', 'Texture
                       'TextureFeatures.LBP.Mean', 'TextureFeatures.LBP.Standard_Deviation', 'TextureFeatures.LBP.Skewness']]

shape_features = df[['ImageId', 'ShapeFeatures.Area.Mean', 'ShapeFeatures.Area.Standard_Deviation', 'ShapeFeatures.Area.Skewness'
                     'ShapeFeatures.Centroid.Mean', 'ShapeFeatures.Centroid.Standard_Deviation', 'ShapeFeatures.Centroid.Skewness
                     'ShapeFeatures.Circularity.Mean', 'ShapeFeatures.Circularity.Standard_Deviation', 'ShapeFeatures.Circularity
                     'ShapeFeatures.MajorAxisLength.Mean', 'ShapeFeatures.MajorAxisLength.Standard_Deviation', 'ShapeFeatures.Maj
                     'ShapeFeatures.MinorAxisLength.Mean', 'ShapeFeatures.MinorAxisLength.Standard_Deviation', 'ShapeFeatures.Min
                     'ShapeFeatures.Eccentricity.Mean', 'ShapeFeatures.Eccentricity.Standard_Deviation', 'ShapeFeatures.Eccentric
                     'ShapeFeatures.Orientation.Mean', 'ShapeFeatures.Orientation.Standard_Deviation', 'ShapeFeatures.Orientation
                     'ShapeFeatures.FilledArea.Mean', 'ShapeFeatures.FilledArea.Standard_Deviation', 'ShapeFeatures.FilledArea.Sk
                     'ShapeFeatures.Perimeter.Mean', 'ShapeFeatures.Perimeter.Standard_Deviation', 'ShapeFeatures.Perimeter.Skew

#File names with CSV extension
color_csv_filename = "color_features.csv"
texture_csv_filename = "texture_features.csv"
shape_csv_filename = "shape_features.csv"

# Save each feature to the file name as a csv
color_features.to_csv(color_csv_filename, index=False)
texture_features.to_csv(texture_csv_filename, index=False)
shape_features.to_csv(shape_csv_filename, index=False)
```

Figure 29 - python code used to seprate the csv files

CBIR.w1985751_JSON_File.json, CBIR.fs.files.json and CBIR.fs.chunks.json contains the database dump.

2. Part 2

2.1. Objective

A women's clothing company that operates in the e-commerce industry receives **many** customer reviews on its clothes sold online. Managing and understanding these **customer** reviews can be **crucial** for improving **customer** satisfaction. The objective is to use natural language processing (NLP) techniques to **analyse** and extract valuable insights from customer reviews**.** With this data, data scientists can conduct further analysis, such as grouping similar reviews, building recommendation systems, or developing predictive models for customer **behaviour** all based on the numerical features that is calculated below.

2.2. Data Collection

From the Kaggle website the 'Women's E-Commerce Clothing Reviews.csv' (Kaggle, 2018)was downloaded as shown in *Figure 31*. This dataset contained reviews given by customers for clothes. The attributes of the dataset were as follows, as shown in *Table 2*:

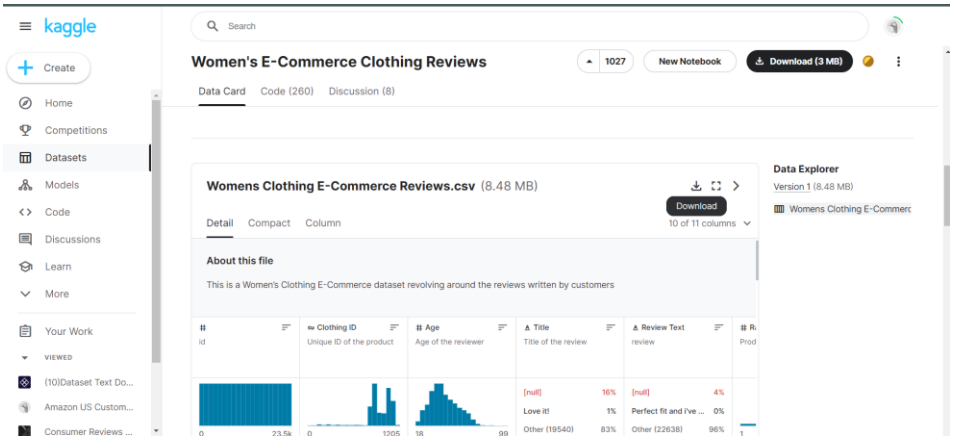

Figure 31 - Kaggle Dataset Download

Index: Unique value for each row
Clothing ID: Integer Categorical variable that refers to the specific piece being reviewed.
Age: Positive Integer variable of the reviewers age.
Title: String variable for the title of the review.
Review Text: String variable for the review body.
Rating: Positive Ordinal Integer variable for the product score granted by the customer from 1 Worst, to 5 Best.
Recommended IND: Binary variable stating where the customer recommends the product where 1 is recommended, 0 is not recommended.
Positive Feedback Count: Positive Integer documenting the number of other customers who found this review positive.
Division Name: Categorical name of the product high level division.
Department Name: Categorical name of the product department name.
Class Name: Categorical name of the product class name.

Table 2 – Metadata of Women's E-Commerce Clothing Reviews.csv'

Sentiment analysis was carried out for a sample of 50 reviews of the customers only, therefore the 'Women's E-Commerce Clothing Reviews.csv' loaded and extracted the first 50 columns of the fifth column, which is 'review Text' column, into a table named 'FinalTable'. These data collection and extraction steps were done in a separate MATLAB file named 'text_document_creation.m'.

```
% Reading the orginial dataset
Dataset=readtable('Womens Clothing E-Commerce Reviews.csv');

% Get a reviews column from the table which is row 5
Col = Dataset(:,5);

% Getting the necessary columns of the dataset
FinalTable = Col(1:50,:);

% Save the reviews column data to a new CSV file
writetable(FinalTable,'reviews.csv');
```

Figure 32 - Load Dataset

To conduct the sentiment analysis, each review was loaded into text documents and stored in a folder called as shown in *Figure 33.* For this process, using a loop each row's data from the FinalTable was loaded as a cell and then converted to a string. Then the string value was written to a text file using the fopen, fprintf, fclose functions. The text documents were saved systemically using the format 'document_01.txt', 'document_02.txt' and so on. With this each cell's data was put into a text document and stored in a folder called 'original_text'.

```
% Create the "preprocessed images" folder if it doesn't exist
folder="original_text";
if ~exist(folder, 'dir')
    mkdir(folder);
end


% Initialize the counter
counter = 1;

% Loop through each row of the table
for i = 1:height(FinalTable)

    % Get the row data as a cell array
    row_data = table2cell(FinalTable(i,:));

    % Convert the cell array to a string
    row_string = strjoin(row_data, ' ');

    % Create the file name based on the counter
    file_name = sprintf('document_%02d.txt',i);

    % Combine the folder path and the file name
    full_name = fullfile(folder, file_name);

    % Write the value to the file
    fid = fopen(full_name,'w'); % Use full_name here
    fprintf(fid,'%s',row_string);
    fclose(fid);

    % Increment the counter
    counter = counter + 1;
end
```

Figure 33 - Loading each review to text documents

2.3. Data pre-processing

In order to use natural language processing (NLP) techniques to extract numerical data, the raw text documents must be cleaned and tokenized.

First using a 'for loop' the 50 text documents in the 'original_text' folder was read and loaded into a cell array called 'TextDataAll' initialized before the loop.

```
% Specifyed the folder path
OriginalTextFolder = 'original_text';
PreprocessedTextFolder = 'preprocessed_text';

% Create the PreprocessedTextFolder if it doesn't exist
if ~exist(PreprocessedTextFolder, 'dir')
    mkdir(PreprocessedTextFolder);
end

% List of the Document names
TextList = dir(fullfile(OriginalTextFolder, '*.txt'));

% Loaded sentiment data from 'sentiments.csv'
SentimentsTable = readtable('sentiment.csv');
Sentiments = SentimentsTable.Sentiment;


% Initialized arrays to store text and corresponding IDs
TextDataAll = cell(length(TextList), 1);
DocIdsAll = cell(length(TextList), 1);

% Loop through each document
for i = 1:length(TextList)
    % Used the actual filename obtained from dir
    FileName = fullfile(OriginalTextFolder, TextList(i).name);

    % Read the text from the document
    Text = fileread(FileName);

    % Stored the text and document ID
    TextDataAll{i} = Text;
    DocIdsAll{i} = sprintf('document_%02d', i);
end
```

Figure 34 - Loading the text documents

Now the cell array named 'TextDataAll', contains the reviews in the 50 text documents in 50 cells. Instead using a loop to pre-process each text document, the cell array allows pre-processing all at once, This way, the code handles multiple documents efficiently.

The text pre-processing was done, firstly by using `lower` function to convert the text to lowercase, and `erasePunctuation` to remove any punctuations and special characters that are not part of words. Since its easier to

analyse and manipulate words the `tokenizedDocument` function was used to split each text into individual words or tokens. In this situation the reviews are very small therefore the tokenization was for word by word not for each sentence. The `removeWords` function eliminated the stop words from the tokenized document. Stop words are common words that do not convey much information or meaning. Since the text documents were very short reviews, only a few stop words were specified in a cell array and removed. The default `stopWord` list was not suitable, as it would make the text documents too vague and render it meaningless.

```
%% Text Preprocessing

% Converted text to lowercase
LowercaseDataAll = lower(TextDataAll);

% Erased punctuation
NpunctDataAll = erasePunctuation(LowercaseDataAll);

% Tokenized the text
TokensAll = tokenizedDocument(NpunctDataAll);

% Removed stop words
StopWords = {'the', 'and', 'is', 'in', 'it', 'to', 'of', 'a', 'for', 'i',...
    'you', 'he', 'she', 'it', 'they', 'them', 'theirs', 'us', 'me'};
FiltTokensAll = removeWords(TokensAll, StopWords);
```

Figure 35 - Data pre-processing

After filtering and tokenizing each text the finally tokenized document was saved in '`FiltTokensAll`'. Next each element of the cell array was converted to a string array using the string function and each element of the string arrays were joined into a single string using the `strjoin` function, as shown in *Figure 36*. Each of these elements were saved in a new text document in the folder 'preprocessed_text' created above shown in *Figure 33*.

```
%% Saving in CSV
for j = 1:length(FiltTokensAll)

    TextArray = string(FiltTokensAll(j));

    % Joining the processed reviews into a single string
    TextArrayStr = strjoin(TextArray, ' ');

    % Saving the processed text document as a single line
    ProcessedTextPath = fullfile('preprocessed_text/', sprintf('pre-processed_document%02d.txt', j));
    writelines(TextArrayStr, ProcessedTextPath);

end
```

Figure 36 - Saving Pre-processed text

2.4. Text vectorization

Text vectorization is the most important step here, it transforms text data into numerical vectors that can be used for machine learning or data analysis. Here two types of text vectorization have been carried out, bag-of-words and term frequency-inverse document-frequency. Although word embeddings were not extracted as it requires a pre-trained model which is not accessible.

Bag of word vectorization creates a vocabulary of unique words from the text data, and then represents each document as a vector of word counts. This has been carried out by the `bagOfWords` function as shown in *Figure 37.*

```
%% Text Vectorization

% Bag-of-Words Vectorization
% Created Bag-of-Words once for filtered tokens
BoWAll = bagOfWords(FiltTokensAll);
% Encoded the documents
VectorizedDocsAll = encode(BoWAll, FiltTokensAll);
VectorsAll = full(VectorizedDocsAll);
```

Figure 37 - Bag-of-words

Term frequency-inverse document frequency (TF-IDF) vectorization is a variation of bag-of-words that assigns weights to the word counts based on their importance in the document. This has been carried out by `tfidf` function as shown in *Figure 38.*

```
%Term Frequency-Inverse Document Frequency
TfidfAll = tfidf(BoWAll, FiltTokensAll);% Calculated TF-IDF once for filtered tokens
```

Figure 38 - TF-IDF

2.5. Metadata and labelling

A JSON document was created containing the extracted text vectorisations vectors and the sentiments for each text document in sperate JSON objects. As shown in *Figure 39* the bag-of-words and term frequency-inverse document frequency features from the pre-processed text data. These features are numerical vectors that represent the frequency and importance of the words in the text documents.

```
%% Create a JSON file to store the meata data, features and sentiments*

% *Extrating the Bag-of-Words*
% Transformed tokens to a cell array
Tok2CellAll = doc2cell(FiltTokensAll);

% Extracted the vocabulary from the Bag-of-Words
ExtBowAll = BoWAll.Vocabulary;

% Created a table with vectors to store Bag-of-Words
VdfTableAll = array2table(VectorsAll, 'VariableNames', ExtBowAll);
VdfStructAll = table2struct(VdfTableAll);

% Converted the Bag-of-Words struct to a cell array
VdfCellAll = cell(length(TextList), 1);
for i = 1:length(TextList)
    VdfCellAll{i, 1} = VdfStructAll(i);
end

% *Extracting Term Frequency-Inverse Document Frequency*
% Created a table with TF-IDF vectors
TfidfTableAll = array2table(full(TfidfAll), 'VariableNames', ExtBowAll);
TfidfStructAll = table2struct(TfidfTableAll);

% Converted the struct to a cell array
TfidfCellAll = cell(length(TextList), 1);
for i = 1:length(TextList)
    TfidfCellAll{i, 1} = TfidfStructAll(i);
end

% Created a review object for the JSON file
ReviewObjectAll = struct('DocumentID', DocIdsAll, ...
    'Sentiment', Sentiments,...
    'Tokens', Tok2CellAll, ...
    'Vectors', VdfCellAll, ...
    'TFIDF', TfidfCellAll);


% Converted the review object to JSON
ReviewEncodeAll = jsonencode(ReviewObjectAll, 'PrettyPrint', true);

% Provided a file name to store the review objects
ReviewJsonFilePathAll = 'w1985751_part2.json';

% Wrote the JSON string to a file
FidAll = fopen(ReviewJsonFilePathAll, 'w');
fprintf(FidAll, ReviewEncodeAll);
fclose(FidAll);
```

Figure 39 - JSON file

Along with the numerical vectors sentiment analysis have been performed manually and included into the Json document. In order to obtain the sentiments another MATLAB code was used in the file ‘sentiment_analysis.m’, as shown in *Figure 40*. With this code the same text pre-processing as shown in *Figure 35*, was conducted and then a figure from the bag-of-words was displayed. As shown in Figure 41, we can observe that the first document has a ‘positive’ sentiment, likewise the sentiments were manually analysed and entered into the ‘sentiment.csv’, this was the loaded into the JSON structure.

```
% Specify the folder
OriginalTextFolder = 'original_text';

% Get the list of the names of the text files
FileList = dir(fullfile(OriginalTextFolder, '*.txt'));

% Loop through the files
for i = 1:length(FileList)
    %% Text pre-processing
    % Get the file name
    FileName = FileList(i).name;

    % File path
    FilePath = fullfile(OriginalTextFolder, FileName);

    % File reading
    Text = fileread(FilePath);

    % Lowercase
    LowerCaseText = lower(Text);

    % Erase punctuation
    NoPunctuationText = erasePunctuation(LowerCaseText);

    % Tokenize the text
    Tokens = tokenizedDocument(NoPunctuationText);

    % Remove the stop words from the list of tokens, using MATLAB's default stopWords list
    StopWords = {'the', 'and', 'is', 'in', 'it', 'to', 'of', 'a', 'for', 'i', ...
        'you', 'he', 'she', 'it', 'they', 'them', 'theirs', 'us', 'me'};
    FilteredTokens = removeWords(Tokens, StopWords);

    BoWAll = bagOfWords(FilteredTokens);
    figure
    wordcloud(BoWAll);

end
```

Figure 40 - Sentiment Analysis

Figure 41 - Sentiment Analysis Bag-Of-Words

As shown above the JSON file named ‘w1985751_part2.json’ contains 50 objects, each with sentiments and features as per the structure created, using MATLAB as shown in *Figure 40.* Further details of structure is shown in the following *Table 3*:

Variable	Description
DocumentID	Unique identifier for each document. It follows the format 'document_XX',
Sentiment	The sentiment label assigned to the document, extracted from the 'sentiments.csv' file.
Tokens	Tokenized representation of the text, where each element corresponds to a token in the pre-processed document.
Vectors	Bag-of-Words representation of the document in vector form, using the vocabulary extracted from the BoW.
TFIDF	TF-IDF (Term Frequency–Inverse Document Frequency) representation of the document using the BoW vocabulary.

Table 3 - w1985751_part2.json file structure

Furthermore another Json file named ‘w1985751_part2_text_sentiment.json’ was created as shown in *Figure 42,* if the user wishes to see the text and its sentiment only.

```
%%  Create a JSON file to store the text and sentiments

%Extracting the tokenized texts
TextStructAll = FiltTokensAll;

% Converted the struct to a cell array
TextCellAll = cell(length(TextList), 1);
for i = 1:length(TextList)
    TextCellAll{i, 1} = FiltTokensAll(i);
end

%Create a text object for the JSON file
TextObjectAll = struct('DocumentID', DocIdsAll, ...
    'Text',TextCellAll,...
    'Sentiment', Sentiments);

% Converted the review object to JSON
TextEncodeAll = jsonencode(TextObjectAll, 'PrettyPrint', true);

% Provided a file name to store the review objects
TextJsonFilePathAll = 'w1985751_part2_text_sentiment.json';

% Wrote the JSON string to a file
TextFidAll = fopen(TextJsonFilePathAll, 'w');
fprintf(TextFidAll, TextEncodeAll);
fclose(TextFidAll);
```

Figure 42 - additional Json file

2.6. Database Creation

As mentioned before since we do not have capability to connect MATLAB to MongoDB, using python language in a Jupyter notebook named 'w1985751_DataLoading_part2.ipynb' the JSON files were pushed to MongoDB.The connection was established and the same libraries were uploaded same as before, shown in *Figure 22, Figure 23* and *Figure 24.*

As shown in *Figure 43* a database named 'SentimentAnalysis' was created and 'w1985751_part2.json' file was stored in the 'w1985751_JSON_TextFile' collection.

```
In [5]: database_name = "SentimentAnalysis"

In [6]: # Create a database in MongoDB Compass
        db = client[database_name]

In [ ]: #To store w1985751_part2.json - with the sentiments and text features

In [7]: #collection name
        collection_name = "w1985751_JSON_TextFile"

In [8]: json_file_path = Path("w1985751_part2.json")

In [9]: #Load the json file
        with open(json_file_path, 'r') as file:
            data = json.load(file)

In [10]: # Insert the data into the collection
         db[collection_name].insert_many(data)

Out[10]: InsertManyResult([ObjectId('659e0dc1c83257a6c43611c8'), ObjectId('659e0dc1c83257a6c43611c9'), ObjectId('659e0dc1c83257a6c43611ca'), ObjectId('659e0dc1c83257a6c43611cb'), ObjectId('659e0dc1c83257a6c43611cc'), ObjectId('659e0dc1c83257a6c43611cd'), ObjectId('659e0dc1c83257a6c43611ce'), ObjectId('659e0dc1c83257a6c43611cf'), ObjectId('659e0dc1c83257a6c43611d0'), ObjectId('659e0dc1c83257a6c43611d1'), ObjectId('659e0dc1c83257a6c43611d2'), ObjectId('659e0dc1c83257a6c43611d3'), ObjectId('659e0dc1c83257a6c43611d4'), ObjectId('659e0dc1c83257a6c43611d5'), ObjectId('659e0dc1c83257a6c43611d6'), ObjectId('659e0dc1c83257a6c43611d7'), ObjectId('659e0dc1c83257a6c43611d8'), ObjectId('659e0dc1c83257a6c43611d9'), ObjectId('659e0dc1c83257a6c43611da'), ObjectId('659e0dc1c83257a6c43611db'), ObjectId('659e0dc1c83257a6c43611dc'), ObjectId('659e0dc1c83257a6c43611dd'), ObjectId('659e0dc1c83257a6c43611de'), ObjectId('659e0dc1c83257a6c43611df'), ObjectId('659e0dc1c83257a6c43611e0'), ObjectId('659e0dc1c83257a6c43611e1'), ObjectId('659e0dc1c83257a6c43611e2'), ObjectId('659e0dc1c83257a6c43611e3'), ObjectId('659e0dc1c83257a6c43611e4'), ObjectId('659e0dc1c83257a6c43611e5'), ObjectId('659e0dc1c83257a6c43611e6'), ObjectId('659e0dc1c83257a6c43611e7'), ObjectId('659e0dc1c83257a6c43611e8'), ObjectId('659e0dc1c83257a6c43611e9'), ObjectId('659e0dc1c83257a6c43611ea'), ObjectId('659e0dc1c83257a6c43611eb'), ObjectId('659e0dc1c83257a6c43611ec'), ObjectId('659e0dc1c83257a6c43611ed'), ObjectId('659e0dc1c83257a6c43611ee'), ObjectId('659e0dc1c83257a6c43611ef'), ObjectId('659e0dc1c83257a6c43611f0'), ObjectId('659e0dc1c83257a6c43611f1'), ObjectId('659e0dc1c83257a6c43611f2'), ObjectId('659e0dc1c83257a6c43611f3'), ObjectId('659e0dc1c83257a6c43611f4'), ObjectId('659e0dc1c83257a6c43611f5'), ObjectId('659e0dc1c83257a6c43611f6'), ObjectId('659e0dc1c83257a6c43611f7'), ObjectId('659e0dc1c83257a6c43611f8'), ObjectId('659e0dc1c83257a6c43611f9')], acknowledged=True)
```

Figure 43 - Database for Json file w1985751_part2.json

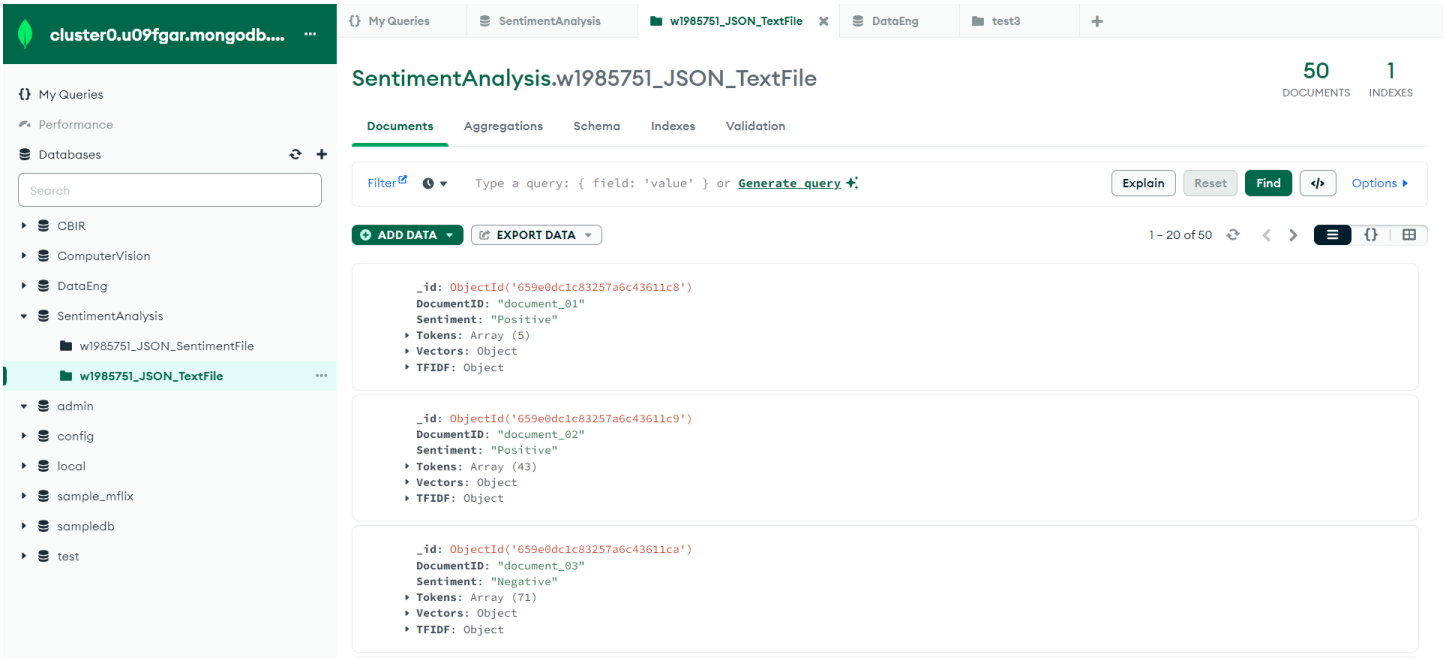

Figure 44 – MongoDB output of w1985751_JSON_TextFile

The ‘w1985751_part2_text_sentiment.json’ was stored in ‘w1985751_JSON_SentimentFile’ as shown in *Figure 44.*

```
In [11]: #To store w1985751_part2_text_sentiment.json - with the sentiments and pre-processed text only

In [16]: #collection name
collection_name = "w1985751_JSON_SentimentFile"

In [17]: json_Sfile_path = Path("w1985751_part2_text_sentiment.json")

In [18]: #Load the json file
with open(json_Sfile_path, 'r') as file:
    data = json.load(file)

In [19]: # Insert the data into the collection
db[collection_name].insert_many(data)

Out[19]: InsertManyResult([ObjectId('659e0ef3c83257a6c436122c'), ObjectId('659e0ef3c83257a6c436122d'), ObjectId('659e0ef3c83257a6c436122
e'), ObjectId('659e0ef3c83257a6c436122f'), ObjectId('659e0ef3c83257a6c4361230'), ObjectId('659e0ef3c83257a6c4361231'), ObjectId
('659e0ef3c83257a6c4361232'), ObjectId('659e0ef3c83257a6c4361233'), ObjectId('659e0ef3c83257a6c4361234'), ObjectId('659e0ef3c83
257a6c4361235'), ObjectId('659e0ef3c83257a6c4361236'), ObjectId('659e0ef3c83257a6c4361237'), ObjectId('659e0ef3c83257a6c436123
8'), ObjectId('659e0ef3c83257a6c4361239'), ObjectId('659e0ef3c83257a6c436123a'), ObjectId('659e0ef3c83257a6c436123b'), ObjectId
('659e0ef3c83257a6c436123c'), ObjectId('659e0ef3c83257a6c436123d'), ObjectId('659e0ef3c83257a6c436123e'), ObjectId('659e0ef3c83
257a6c436123f'), ObjectId('659e0ef3c83257a6c4361240'), ObjectId('659e0ef3c83257a6c4361241'), ObjectId('659e0ef3c83257a6c436124
2'), ObjectId('659e0ef3c83257a6c4361243'), ObjectId('659e0ef3c83257a6c4361244'), ObjectId('659e0ef3c83257a6c4361245'), ObjectId
('659e0ef3c83257a6c4361246'), ObjectId('659e0ef3c83257a6c4361247'), ObjectId('659e0ef3c83257a6c4361248'), ObjectId('659e0ef3c83
257a6c4361249'), ObjectId('659e0ef3c83257a6c436124a'), ObjectId('659e0ef3c83257a6c436124b'), ObjectId('659e0ef3c83257a6c436124
c'), ObjectId('659e0ef3c83257a6c436124d'), ObjectId('659e0ef3c83257a6c436124e'), ObjectId('659e0ef3c83257a6c436124f'), ObjectId
('659e0ef3c83257a6c4361250'), ObjectId('659e0ef3c83257a6c4361251'), ObjectId('659e0ef3c83257a6c4361252'), ObjectId('659e0ef3c83
257a6c4361253'), ObjectId('659e0ef3c83257a6c4361254'), ObjectId('659e0ef3c83257a6c4361255'), ObjectId('659e0ef3c83257a6c436125
6'), ObjectId('659e0ef3c83257a6c4361257'), ObjectId('659e0ef3c83257a6c4361258'), ObjectId('659e0ef3c83257a6c4361259'), ObjectId
('659e0ef3c83257a6c436125a'), ObjectId('659e0ef3c83257a6c436125b'), ObjectId('659e0ef3c83257a6c436125c'), ObjectId('659e0ef3c83
257a6c436125d')], acknowledged=True)
```

Figure 45 - database for w1985751_part2_text_sentiment.json

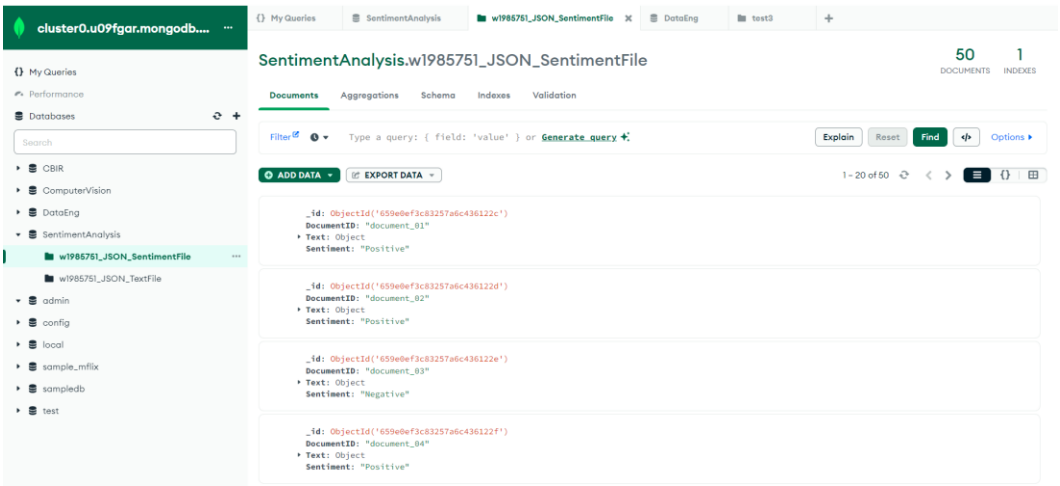

Figure 46 - MongoDB output for w1985751_JSON_SentimentFile

SentimentAnalysis.w1985751_JSON_SentimentFile.json and SentimentAnalysis.w1985751_JSON_TextFile.json files contained the databse dump.

Self Reflection

- If more time was available a more effective method of noise reduction for the images could be explored, as the imgaussfilt function used in the coursework blurred the images and affected their sharpness and contrast.
- The global binarization method had some drawbacks, such as introducing some 'artifacts' and errors in the background, and losing some details or textures in the foreground. Had to conduct multiple experiments on how to apply the binarization, made another loop to use the standard deviation of pixel intensities as a threshold to decide which binarization worked best. Although the "global" method gave the best result. If given the opportunity to redo, would explore more into this topic to binarize the image better.
- Could not save the image address documents as csv files, due to the commas in URL, when saving as a CSV file it divided into many columns.
- Coult not find a method to read all the image extensions only one file extension was possible to read at once, in this situation 'jpeg'.
- There was a difficulty in Sentiment Analysis on how to get the text vectorisation without combining all the text documents, but after carefully studying lecture notes got a more understanding on how to do it.
- After doing image feature processing realised that it's possible to read each image using a loop and then store them in cell array instead of using a loop for image preprocessing and feature extraction. Therefore, this method was then used in sentiment analysis.
- The ideal way to conduct the CBIR and the sentiment analysis would be to use only 1 type of programming language and automate the annotations and sentiment analysis. However due to the unavailability of license to use a pre-trained model and to establish a connection to MongoDB with MATLAB this wasn't possible.
- Since it was important make the reader walk through each and every step of the coursework it was difficult to strictly adhere to the word count.
- The coursework required the unstructured data such as text and image's features were extracted and stored in a NoSQL database. In my opinion this ability displayed properly.

References

Kaggle, 2018. *Women's E-Commerce Clothing Reviews,* s.l.: NICAPOTATO.

Pexels, 2024. s.l.: s.n.

w1985751, 2023. s.l.: s.n.