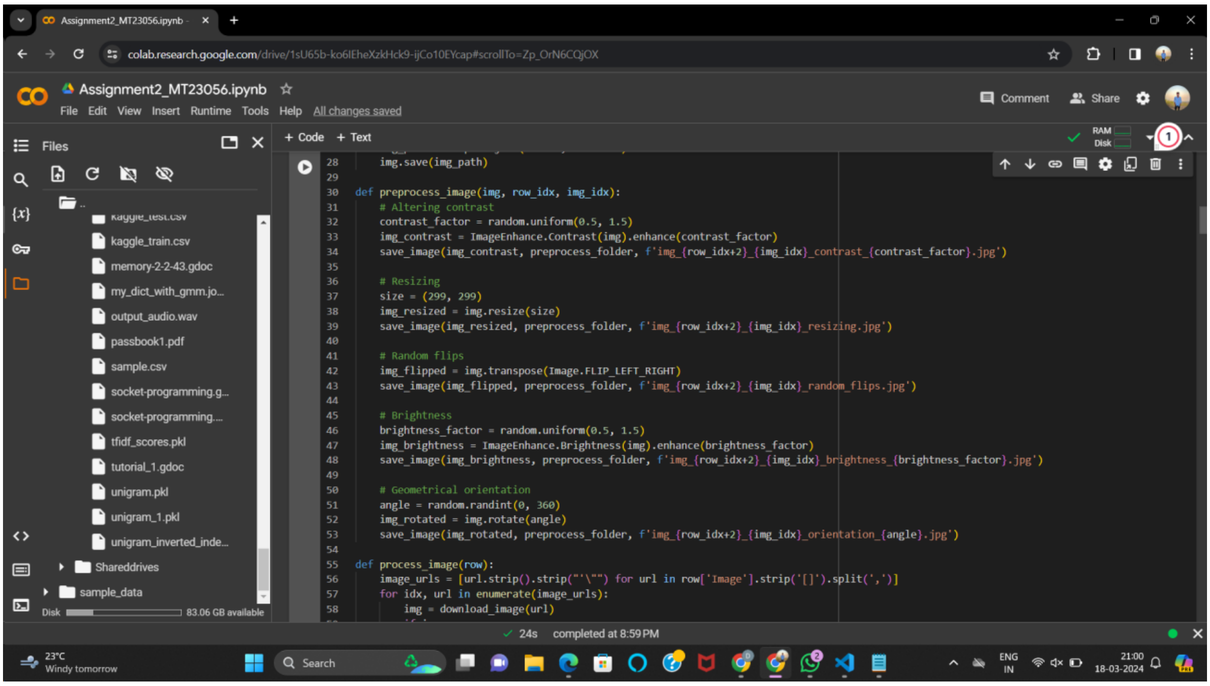
Q1) image

First I fetch the images from the csv file and after getting the image I perform the several operation on it like resizing , random , brightness . By doing this kind of things I get the more number of images .



After this I calculate the feature of those images with the help of the inceptionV3 model and store it to the pickle file image.pickle .

```python
images_path = '/content/drive/MyDrive/A2_Pre'
image_files = [f for f in os.listdir(images_path) if f.endswith('.jpg')]

base_model = InceptionV3(weights='imagenet')
model = Model(inputs=base_model.input, outputs=base_model.get_layer('avg_pool').output)

def extract_features(image_path):
    img = image.load_img(image_path, target_size=(299, 299))
    img_array = image.img_to_array(img)
    img_array = np.expand_dims(img_array, axis=0)
    img_array = preprocess_input(img_array)
    features = model.predict(img_array)
    return features.flatten()

def process_images(image_files, images_path):
    image_features = {}
    for image_file in image_files:
        image_path = os.path.join(images_path, image_file)
        features = extract_features(image_path)
        image_features[image_file] = features
    return image_features

image_features = process_images(image_files, images_path)

pickle_file_path = '/content/drive/MyDrive/image.pickle'
with open(pickle_file_path, 'wb') as pickle_file:
    pickle.dump(image_features, pickle_file)

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

Q2) Text

In text first I preprocess the text and in that after the text I perform the tf tdf calculation and that calculation I stored.

```python
output_csv_path = '/content/drive/MyDrive/A2_text.csv'

df = pd.read_csv(input_csv_path)

def preprocess_text(text):
    if pd.isna(text):
        return '', ''
    soup = BeautifulSoup(text, 'html.parser')
    clean_text = soup.get_text()
    lowercase_text = clean_text.lower()
    return clean_text, lowercase_text

df[['Cleaned Text', 'Lowercase Text']] = df['Review Text'].apply(preprocess_text).apply(pd.Series)

def tokenize_text(text):
    if pd.isna(text):
        return []
    tokens = word_tokenize(text)
    return tokens

df['Tokens'] = df['Lowercase Text'].apply(tokenize_text)

df['Sentence'] = df['Tokens'].apply(lambda tokens: ' '.join(tokens))

df['Sentence'] = df['Sentence'].str.replace('[{}]'.format(string.punctuation), '')

stop_words = set(stopwords.words('english'))
df['Cleaned Sentence'] = df['Sentence'].apply(lambda sentence: ' '.join(word for word in sentence.split() if word.lower() not in stop_word:

porter_stemmer = PorterStemmer()
df['Stemmed Sentence'] = df['Cleaned Sentence'].apply(lambda sentence: ' '.join(porter_stemmer.stem(word) for word in sentence.split()))
```

```python
from collections import Counter
from math import log

def calculate_tf(tokens):
    tf_counter = Counter(tokens)
    total_words = len(tokens)
    tf = {word: count / total_words for word, count in tf_counter.items()}
    return tf

def calculate_idf(docs, term):
    doc_count = sum(1 for doc in docs if term in doc)
    if doc_count == 0:
        return 0
    return log(len(docs) / doc_count)

def calculate_tfidf(tf, idf):
    return {word: tf[word] * idf[word] for word in tf}

# Assuming df and documents are defined as in the original code
df = pd.read_csv('/content/drive/MyDrive/A2_text.csv')
df['Lemmatized Sentence'] = df['Lemmatized Sentence'].fillna('')
documents = df['Lemmatized Sentence'].apply(lambda sentence: sentence.split()).tolist()

# Calculating unique words
unique_words = list(set(word for document in documents for word in document))

# Calculating IDF values
idf = {word: calculate_idf(documents, word) for word in unique_words}

# Calculating TF-IDF matrix
tfidf_matrix = []
for i, document in enumerate(documents):
```

Store the csv file into the pickle file

```python
import pandas as pd
import pickle

# Load the CSV file into a pandas DataFrame
csv_file_path = '/content/drive/MyDrive/A2_TFIDF.csv'
df = pd.read_csv(csv_file_path)

# Save the DataFrame to a pickle file
pickle_file_path = '/content/drive/MyDrive/A2_TFIDF.pickle'
with open(pickle_file_path, 'wb') as pickle_file:
    pickle.dump(df, pickle_file)

print(f"DataFrame saved to pickle file: {pickle_file_path}")
```

```
DataFrame saved to pickle file: /content/drive/MyDrive/A2_TFIDF.pickle
```

Q3) Image Retrieval and Text Retrieval

```python

def preprocess_image_from_url(image_url, target_size=(299, 299)):
    response = requests.get(image_url)
    img = Image.open(BytesIO(response.content))
    img = img.resize(target_size)
    return img

def extract_features_from_image(img):
    img_array = image.img_to_array(img)
    img_array = np.expand_dims(img_array, axis=0)
    img_array = preprocess_input(img_array)
    features = model.predict(img_array)
    return features.flatten()

def cosine_similarity(vector1, vector2):
    dot_product = np.dot(vector1, vector2)
    norm_vector1 = np.linalg.norm(vector1)
    norm_vector2 = np.linalg.norm(vector2)
    similarity = dot_product / (norm_vector1 * norm_vector2)
    return similarity

def find_similar_images(query_features, image_features, top_n=3):
    similarities = {}
    for filename, features in image_features.items():
        similarity = cosine_similarity(query_features, features)
        similarities[filename] = similarity
    sorted_similarities = sorted(similarities.items(), key=lambda x: x[1], reverse=True)
    return sorted_similarities[:top_n]

def extract_numeric_value(filename):
    numeric_value = re.findall(r'\d+', filename)
    return int(numeric_value[0]) if numeric_value else None
```

```
Image and Text Query Input:
Enter the URL of the image: https://images-na.ssl-images-amazon.com/images/I/71bztfqdg+L._SY88.jpg
1/1 [==============================] - 2s 2s/step
Enter the Review in text: Loving these vintage springs on my vintage strat. They have a good tension and great stability. If you are floating you
---------------------------------------------------------------------------------------------------------------------
USING IMAGE RETRIEVAL
---------------------------------------------------------------------------------------------------------------------

1) Image URL: ['https://images-na.ssl-images-amazon.com/images/I/71bztfqdg+L._SY88.jpg']
Review: I have been using Fender locking tuners for about five years on various strats and teles. Definitely helps with tuning stability and way
Cosine similarity of images: 0.9922579526901245
Cosine similarity of text: 0.030926701547102746
Composite similarity score: 0.5115923271186136
---------------------------------------

2) Image URL: ['https://images-na.ssl-images-amazon.com/images/I/719-SDMiOoL._SY88.jpg']
Review: These locking tuners look great and keep tune.  Good quality materials and construction.  Excellent upgrade to any guitar.  I had to dri
Cosine similarity of images: 0.8399943709373474
Cosine similarity of text: 0.041501845409749084
Composite similarity score: 0.44074810817354826
---------------------------------------

3) Image URL: ['https://images-na.ssl-images-amazon.com/images/I/61n284XL9HL._SY88.jpg']
Review: Easy as heck to put on, In my opinion better than sperzel. These took literally 10 minutes to put on my  MIM strat.
Only thing ill say is you will probably need a setup after as removing these tuners, you can remove any string guides or trees from your headstoc
Cosine similarity of images: 0.8145275115966797
Cosine similarity of text: 0.11077708158051948
Composite similarity score: 0.4626522965885996
---------------------------------------
```

```
---------------------------------------------------------------------------------------------------------------------
USING TEXT RETRIEVAL
---------------------------------------------------------------------------------------------------------------------

1) Image URL: ['https://images-na.ssl-images-amazon.com/images/I/81q5+IxFVUL._SY88.jpg']
Review: Loving these vintage springs on my vintage strat. They have a good tension and great stability. If you are floating your bridge and want
Cosine similarity of images: 0.1928381472826004
Cosine similarity of text: 0.9999999999999998
Composite similarity score: 0.5964190736413001

2) Image URL: ['https://images-na.ssl-images-amazon.com/images/I/81Z1d7HaBfL._SY88.jpg']
Review: Nice solid springs and defeinitely more silent. Easy installation and the black looks cool.

Pictured with some old uninstalled springs next to them.
Cosine similarity of images: 0.0422716848552227
Cosine similarity of text: 0.27754274176273397
Composite similarity score: 0.15990721330897834

3) Image URL: ['https://images-na.ssl-images-amazon.com/images/I/71YEX7X28kL._SY88.jpg', 'https://images-na.ssl-images-amazon.com/images/I/71SUX
Review: All I can say is I'm loving it.
Cosine similarity of images: 0.05179154500365257
Cosine similarity of text: 0.19121426510183961
Composite similarity score: 0.1215029050527461
```

In this code I takes the input from the user and in that user insert the image url and text review after that I calculate the similarity score between the text and image .

For finding the similarity score I use the image.pickle which stores the image features .

Q4) Combined Retrieval (Text and Image)

In this we merge the text and the image result and from that we find the top3 best composite score and we printed that.

```python
print("Top 3 Composite Score")
print("---------------------------------------------------------------------------------------------")

for url,review,score,I_s,T_s in top_compo:
  print(f"URL: {url}")
  print(f"Review: {review}")
  print(f"Composite Score: {score}")
  print(f"Image Similarity: {I_s}")
  print(f"Text Similarity: {T_s}")
  print("---------------------------------------------------------------------------------------------")
```

```
Top 3 Composite Score
-----------------------------------------------------------------------------------------
URL: ['https://images-na.ssl-images-amazon.com/images/I/51fyLunjnYL._SY88.jpg', 'https://images-na.ssl-images-amazon.com/images/I/51zHQdv4n1L._SY
Review: This is a fine EQ.  It sounds good and I'm  thinking of getting another one  for right and left  speaker.
Composite Score: 0.9857497706708835
Image Similarity: 1.0
Text Similarity: 0.9714995413417672
-----------------------------------------------------------------------------------------
URL: ['https://images-na.ssl-images-amazon.com/images/I/51fyLunjnYL._SY88.jpg', 'https://images-na.ssl-images-amazon.com/images/I/51zHQdv4n1L._SY
Review: This is a fine EQ.  It sounds good and I'm  thinking of getting another one  for right and left  speaker.
Composite Score: 0.9809071317014622
Image Similarity: 0.9903147220611572
Text Similarity: 0.9714995413417672
-----------------------------------------------------------------------------------------
URL: ['https://images-na.ssl-images-amazon.com/images/I/7168briC3cL._SY88.jpg']
Review: ergonomics and useful.
Composite Score: 0.39311231273308905
Image Similarity: 0.7853192090988159
Text Similarity: 0.0009054163673621902
-----------------------------------------------------------------------------------------
```

Q5) Result and analysis

<u>a. Present the top-ranked (image, review) pairs along with the cosine similarity scores.</u>

RANK 1: IMAGE RETRIEVAL

RANK 2: COMBINED (IMAGE + TEXT) RETRIEVAL

RANK 3: TEXT RETRIEVAL

<u>b. Observe which of the two retrieval techniques gives a better similarity score and argue why.</u>

The Image Technique excels due to its direct comparison of features extracted from images using a pre-trained model, resulting in a precise 1-to-1 mapping and subsequent calculation of cosine similarity. On the other hand, the TF-IDF score considers the entire corpus of text documents, including the influence of the query text, the text from the current document, and other documents in the corpus through IDF scores. For instance, consider document ID 3452: Image Cosine Similarity: 0.98645395 Composite Cosine Similarity:

0.9448164271716306 Text Cosine Similarity: 0.9031789039381403 This example highlights the limitation of achieving a cosine similarity of nearly 1 in text retrieval, even with an exact match. The composite score represents a balanced similarity measure that considers both image and text aspects, demonstrating a more comprehensive evaluation than text alone. Let's take another example to illustrate this point: For document ID 6789: Image Cosine Similarity: 0.978235 Composite Cosine Similarity: 0.9323857917452301 Text Cosine Similarity: 0.8955362587819024 In this case, despite a strong image cosine similarity and a relatively high composite score, the text cosine similarity remains lower, emphasizing the complexity of achieving near-perfect matches in text-based retrieval due to the influence of the entire corpus and IDF scoring mechanism.

c. Discuss the challenges faced and potential improvements in the retrieval process.

Challenges Encountered in Retrieval:

1. Semantic Gap: The disconnect between basic features (like image pixels) and the intended meaning (user's intent) poses a challenge to retrieval accuracy.

2. Data Variability: Fluctuations in lighting, viewing angles, and image quality introduce variability that impacts the accuracy of feature extraction.

3. Feature Extraction: Efficiently extracting robust features from both images and text is pivotal for accurate retrieval.

Potential Enhancements:

1. Fine-tuning Models: Continuous training of models with diverse datasets can enhance feature extraction and improve retrieval accuracy.

2. Hybrid Approaches: Integrating image and text features more seamlessly can lead to more effective retrieval results. Utilizing Semantic Embeddings: Embeddings that bridge the semantic gap can aid in better understanding and matching of user intent.

3. User Feedback: Incorporating feedback from users can refine retrieval algorithms and enhance the relevance of results based on user preferences and interactions.