AI 705: Recommendation Systems

End Term Project

MT2023161 Nishtha Paul

MT2023083 H Anarghya

MT2020147 Anamika Mishra

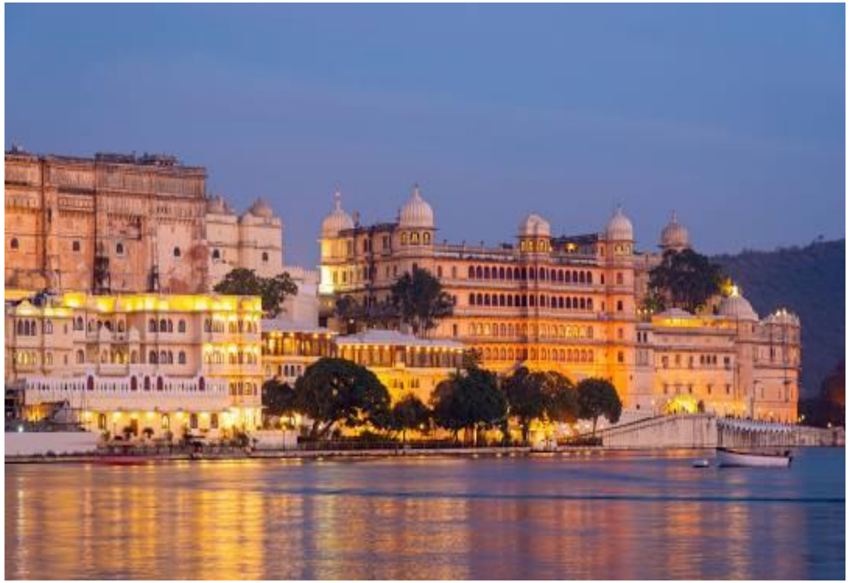
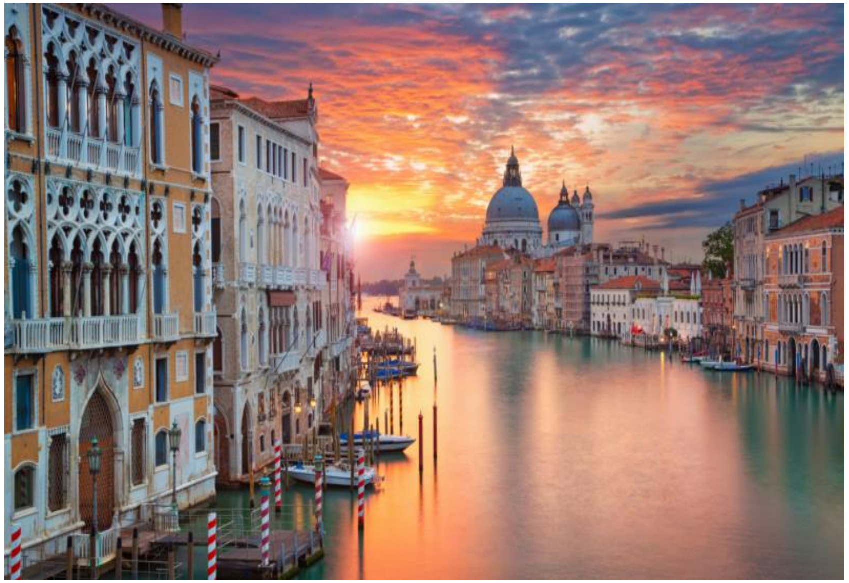
Introduction

- India is a vast land with vibrant culture and ancient traditions. From bustling megacities to serene villages, from the snow-capped peaks of the Himalayas to the serene backwaters of Kerala, India's beauty is unparalleled.
- As a land of innovation and technological prowess, India is a nation constantly evolving while cherishing its ancient heritage. In this huge interconnected world, the exchange of ideas and best practices between diverse cultures will play a major factor to enable sustainable development.
- Many cities in the European countries are renowned for their unique blend of history, culture, and innovation, making them coveted destinations for travelers and admired models for urban development.
- Our aim is to recommend European cities which can be similar to the Indian cities, to take inspiration and identify innovative approaches to urban development.

INDIA
MAJOR CITIES
AFGHANISTAN
TAJIKISTAN
PAKISTAN
CHINA (TIBET)
NEPAL
BHUTAN
BANGLADESH
MYANMAR
SRI LANKA
Arabian Sea
Bay of Bengal
INDIAN OCEAN
JAMMU & KASHMIR
LADAKH
HIMACHAL PRADESH
PUNJAB
CHANDIGARH
UTTARAKHAND
HARYANA
NEW DELHI
RAJASTHAN
UTTAR PRADESH
BIHAR
SIKKIM
ARUNACHAL PRADESH
ASSAM
NAGALAND
MEGHALAYA
MANIPUR
TRIPURA
MIZORAM
JHARKHAND
WEST BENGAL
GUJARAT
MADHYA PRADESH
CHHATTISGARH
ODISHA
DADRA & NAGAR HAVELI AND DAMAN & DIU
MAHARASHTRA
TELANGANA
ANDHRA PRADESH
GOA
KARNATAKA
KERALA
TAMIL NADU
PUDUCHERRY
LAKSHADWEEP (INDIA)
ANDAMAN & NICOBAR ISLANDS (INDIA)
Mumbai
Hyderabad
Bengaluru
Chennai
Kolkata
Lucknow
Jaipur
Bhopal
Srinagar
Shimla
International Boundary
State/UT Boundary
Country Capital
State/UT Capital
Major City
* Amaravati is considered as state capital by the state govt. but has not been notified by the central govt.
Map not to Scale
Copyright © 2020 www.mapsofindia.com
MapsofIndia

Purpose

Do these images look similar?

Were you able to identify?

Udaipur

Venice

- These cities are similar in terms of scenic beauty and geography, and yet, the tourism industry in Venice is much better than in Udaipur.

- India is a developing country, and our purpose is to inspire improvements in Indian cities by learning from the strengths of their European counterparts.

- By exploring the strengths and innovations of European cities, we can identify adaptable solutions by considering India's unique social, cultural, and economic realities which can aid to enhance the quality of life, infrastructure, and overall urban experience in the selected Indian city.

Data Collection

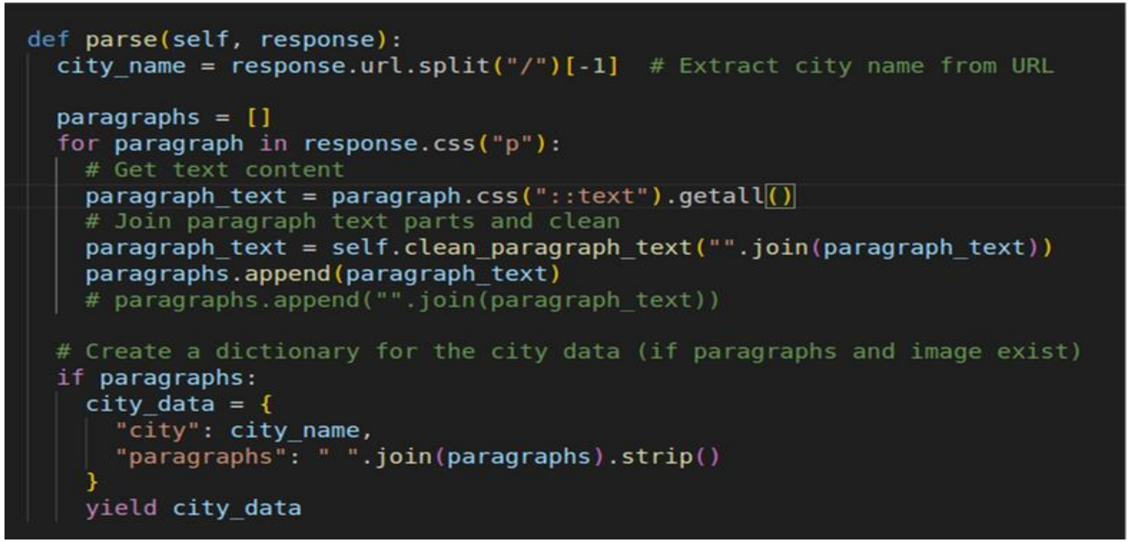
- One of the major challenges in this project was to collect relevant data for every city which can depict the city’s culture and traditions. Such kind of information is usually in the form of text.

- Wikipedia is a free, online encyclopedia, consisting of vast amount of information that is publicly accessible to obtain the latest information.

- The first step was to scrape the Wikipedia webpage for each city.

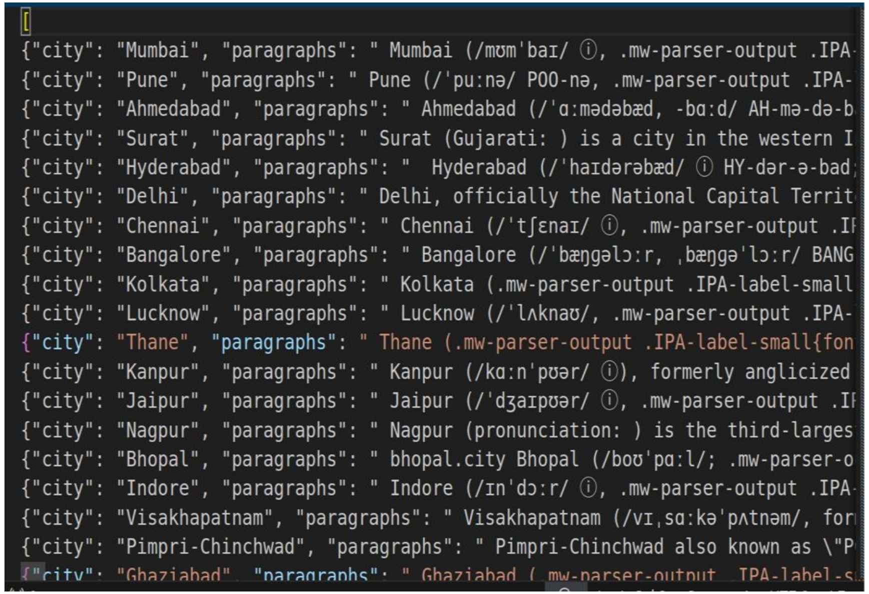
- Scrapy is powerful library that can efficiently crawl through multiple web pages and offers features like handling different website content formats.

- First step was to collect the webpage URLs to all the Indian and European cities.
- We have a total of 430 cities, with 267 Indian cities and 163 European cities.
- While scraping through each webpage, we extract all the paragraph (<p>) elements and extract the text content of each paragraph element.

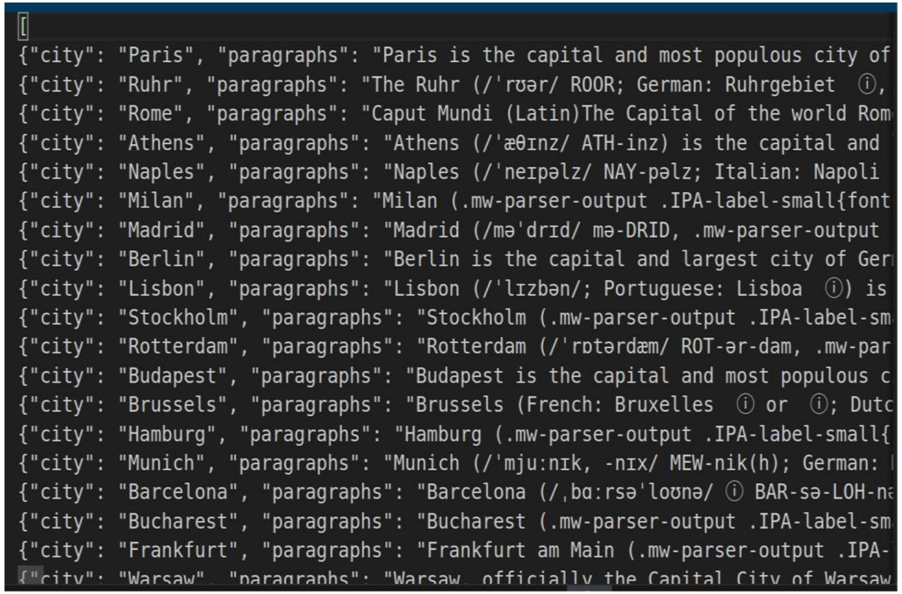


```python
def parse(self, response):
    city_name = response.url.split("/")[-1]  # Extract city name from URL

    paragraphs = []
    for paragraph in response.css("p"):
        # Get text content
        paragraph_text = paragraph.css("::text").getall()
        # Join paragraph text parts and clean
        paragraph_text = self.clean_paragraph_text("".join(paragraph_text))
        paragraphs.append(paragraph_text)
        # paragraphs.append("".join(paragraph_text))

    # Create a dictionary for the city data (if paragraphs and image exist)
    if paragraphs:
        city_data = {
            "city": city_name,
            "paragraphs": " ".join(paragraphs).strip()
        }
        yield city_data
```

- We run the program for Indian cities and European cities to get the JSON containing the scraped data



```
[
{"city": "Mumbai", "paragraphs": " Mumbai (/mʊm'baɪ/ ⓘ, .mw-parser-output .IPA-
{"city": "Pune", "paragraphs": " Pune (/'pu:nə/ POO-nə, .mw-parser-output .IPA-
{"city": "Ahmedabad", "paragraphs": " Ahmedabad (/'ɑ:mədəbæd, -bɑ:d/ AH-mə-də-b
{"city": "Surat", "paragraphs": " Surat (Gujarati: ) is a city in the western I
{"city": "Hyderabad", "paragraphs": "  Hyderabad (/'haɪdərəbæd/ ⓘ HY-dər-ə-bad;
{"city": "Delhi", "paragraphs": " Delhi, officially the National Capital Territ
{"city": "Chennai", "paragraphs": " Chennai (/'tʃɛnaɪ/ ⓘ, .mw-parser-output .IP
{"city": "Bangalore", "paragraphs": " Bangalore (/'bæŋgəlɔ:r, ˌbæŋgə'lɔ:r/ BANG
{"city": "Kolkata", "paragraphs": " Kolkata (.mw-parser-output .IPA-label-small
{"city": "Lucknow", "paragraphs": " Lucknow (/'lʌknaʊ/, .mw-parser-output .IPA-
{"city": "Thane", "paragraphs": " Thane (.mw-parser-output .IPA-label-small{fon
{"city": "Kanpur", "paragraphs": " Kanpur (/kɑ:n'pʊər/ ⓘ), formerly anglicized
{"city": "Jaipur", "paragraphs": " Jaipur (/'dʒaɪpʊər/ ⓘ, .mw-parser-output .IP
{"city": "Nagpur", "paragraphs": " Nagpur (pronunciation: ) is the third-larges
{"city": "Bhopal", "paragraphs": " bhopal.city Bhopal (/boʊ'pɑ:l/; .mw-parser-o
{"city": "Indore", "paragraphs": " Indore (/ɪn'dɔ:r/ ⓘ, .mw-parser-output .IPA-
{"city": "Visakhapatnam", "paragraphs": " Visakhapatnam (/vɪˌsɑ:kə'pʌtnəm/, for
{"city": "Pimpri-Chinchwad", "paragraphs": " Pimpri-Chinchwad also known as \"P
{"city": "Ghaziabad", "paragraphs": " Ghaziabad (.mw-parser-output .IPA-label-s
```

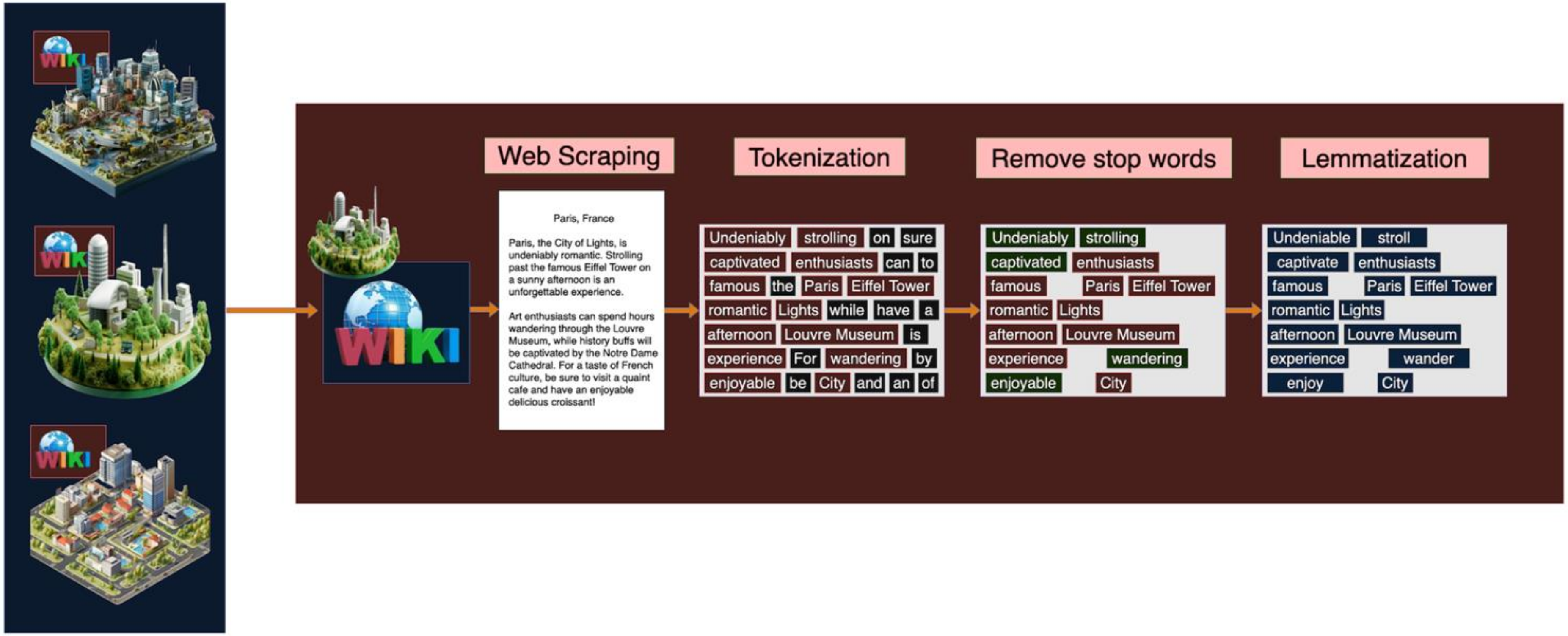


```
[
{"city": "Paris", "paragraphs": "Paris is the capital and most populous city of
{"city": "Ruhr", "paragraphs": "The Ruhr (/'rʊər/ ROOR; German: Ruhrgebiet  ⓘ,
{"city": "Rome", "paragraphs": "Caput Mundi (Latin)The Capital of the world Rom
{"city": "Athens", "paragraphs": "Athens (/'æθɪnz/ ATH-inz) is the capital and
{"city": "Naples", "paragraphs": "Naples (/'neɪpəlz/ NAY-pəlz; Italian: Napoli
{"city": "Milan", "paragraphs": "Milan (.mw-parser-output .IPA-label-small{font
{"city": "Madrid", "paragraphs": "Madrid (/mə'drɪd/ mə-DRID, .mw-parser-output
{"city": "Berlin", "paragraphs": "Berlin is the capital and largest city of Ger
{"city": "Lisbon", "paragraphs": "Lisbon (/'lɪzbən/; Portuguese: Lisboa  ⓘ) is
{"city": "Stockholm", "paragraphs": "Stockholm (.mw-parser-output .IPA-label-sm
{"city": "Rotterdam", "paragraphs": "Rotterdam (/'rɒtərdæm/ ROT-ər-dam, .mw-par
{"city": "Budapest", "paragraphs": "Budapest is the capital and most populous c
{"city": "Brussels", "paragraphs": "Brussels (French: Bruxelles  ⓘ or  ⓘ; Dutc
{"city": "Hamburg", "paragraphs": "Hamburg (.mw-parser-output .IPA-label-small{
{"city": "Munich", "paragraphs": "Munich (/'mju:nɪk, -nɪx/ MEW-nik(h); German:
{"city": "Barcelona", "paragraphs": "Barcelona (/ˌbɑ:rsə'loʊnə/ ⓘ BAR-sə-LOH-n
{"city": "Bucharest", "paragraphs": "Bucharest (.mw-parser-output .IPA-label-sm
{"city": "Frankfurt", "paragraphs": "Frankfurt am Main (.mw-parser-output .IPA-
{"city": "Warsaw", "paragraphs": "Warsaw, officially the Capital City of Warsaw
```

Data Pre-processing

- Once the text data has been scraped for each city, the text is processed using the **nltk** library.
- The sequence of processing is as follows:
    - **Lower case** - Converting the flow of text all into lower case.
    - **Removing Stop words -** Removing words like I, me, etc. which don't add any extra meaning to the sentence.
    - **Punctuation removal -** Removing comma, full stop, apostrophe, etc.
    - **Tokenising -** Converting each word in a sentence into an entry in a list.
    - **Lemmatizing -** Grouping together multiple forms of a word as a single word.

Web Scraping



Paris, France

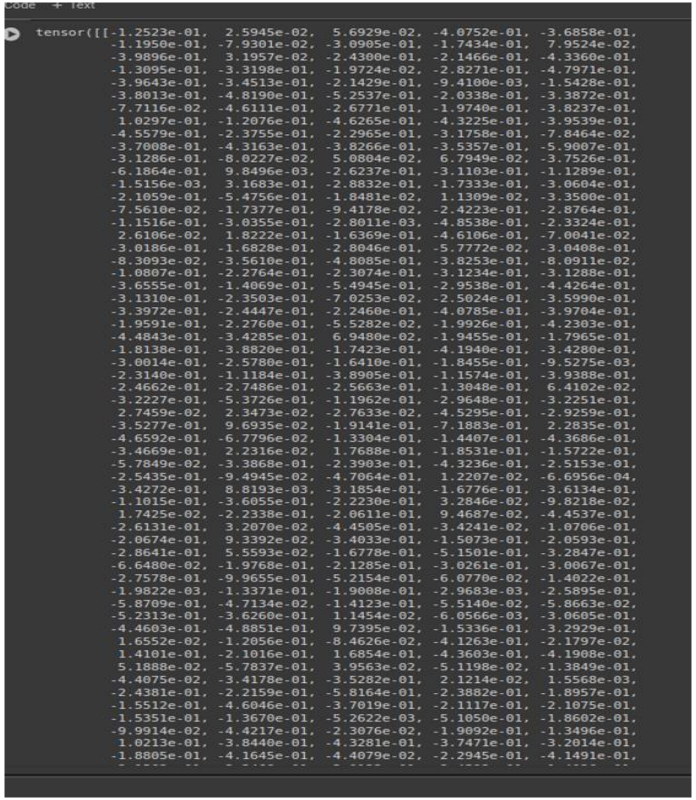
Paris, the City of Lights, is undeniably romantic. Strolling past the famous Eiffel Tower on a sunny afternoon is an unforgettable experience.

Art enthusiasts can spend hours wandering through the Louvre Museum, while history buffs will be captivated by the Notre Dame Cathedral. For a taste of French culture, be sure to visit a quaint cafe and have an enjoyable delicious croissant!

```python
# Preprocess text descriptions
def preprocess_text(text):
    text = text.lower()  # Lowercase
    pattern = r'\((?!(mw-parser-output|English|Bengali pronunciation|Kannada pronunciation|IPA|Gujarati|Hindi))[^)]+\)'
    text = re.sub(pattern, '', text)
    text = re.sub(r'[^\w\s]', '', text)  # Remove punctuation
    # Remove stop words
    stop_words = set(stopwords.words('english'))
    lemmatizer = WordNetLemmatizer()

    words = word_tokenize(text)
    # Lemmatize and remove stop words
    filtered_words = [lemmatizer.lemmatize(word) for word in words if word not in stop_words]
    text = ' '.join(filtered_words)

    return text
```

Generating Text Embeddings

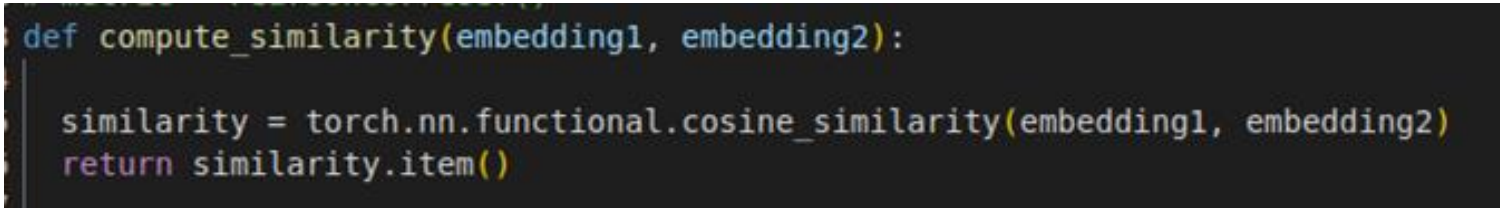
In the recent times, the most popular model to retrieve text embeddings are Large Language Models(LLM). We have used the pretrained BERT model that is available on the HuggingFace platform to obtain the embeddings. It provides an embedding of 768 dimension.



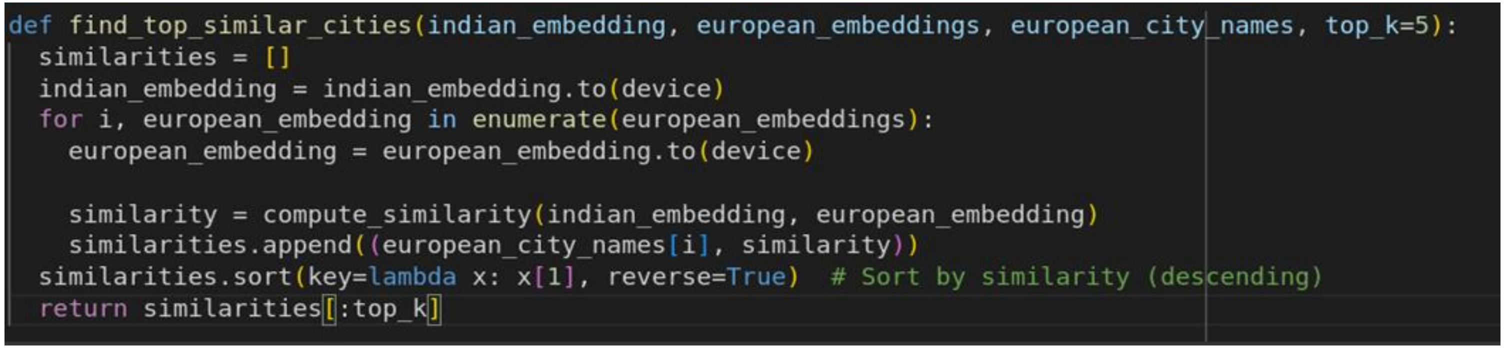
```
tensor([[-1.2523e-01,  2.5945e-02,  5.6929e-02, -4.0752e-01, -3.6858e-01,
         -1.1950e-01, -7.9301e-02, -3.0905e-01, -1.7434e-01,  7.9524e-02,
         -3.9896e-01,  3.1957e-02, -2.4300e-01, -2.1466e-01, -4.3360e-01,
         -1.3095e-01, -3.3198e-01, -1.9724e-02, -2.8271e-01, -4.7971e-01,
         -3.9643e-01, -3.4513e-01, -2.1429e-01, -9.4100e-03, -1.5428e-01,
         -3.8013e-01, -4.8190e-01, -5.2537e-01, -2.0338e-01, -3.3872e-01,
         -7.7116e-02, -4.6111e-01, -2.6771e-01, -1.9740e-01, -3.8237e-01,
          1.0297e-01, -1.2076e-01, -4.6265e-01, -4.3225e-01, -3.9539e-01,
         -4.5579e-01, -2.3755e-01, -2.2965e-01, -3.1758e-01, -7.8464e-02,
         -3.7008e-01, -4.3163e-01, -3.8266e-01, -3.5357e-01, -5.9007e-01,
         -3.1286e-01, -8.0227e-02,  5.0804e-02,  6.7949e-02, -3.7526e-01,
         -6.1864e-01,  9.8496e-03, -2.6237e-01, -3.1103e-01, -1.1289e-01,
         -1.5156e-03,  3.1683e-01, -2.8832e-01, -1.7333e-01, -3.0604e-01,
         -2.1059e-01, -5.4756e-01, -1.8481e-02,  1.1309e-02, -3.3500e-01,
         -7.5610e-02, -1.7377e-01, -9.4178e-02, -2.4223e-01, -2.8764e-01,
         -1.1516e-01, -3.0355e-01, -2.8011e-03, -4.8538e-01, -2.3324e-01,
          2.6106e-02,  1.8222e-01, -1.6369e-01, -4.6106e-01, -7.0041e-02,
         -3.0186e-01, -1.6828e-01, -2.8046e-01, -5.7772e-02, -3.0408e-01,
         -8.3093e-02, -3.5610e-01, -4.8085e-01, -3.8253e-01, -8.0911e-02,
         -1.0807e-01, -2.2764e-01, -2.3074e-01, -3.1234e-01, -3.1288e-01,
         -3.6555e-01, -1.4069e-01, -5.4945e-01, -2.9538e-01, -4.4264e-01,
         -3.1310e-01, -2.3503e-01, -7.0253e-02, -2.5024e-01, -3.5990e-01,
         -3.3972e-01, -2.4447e-01, -2.2460e-01, -4.0785e-01, -3.9704e-01,
         -1.9591e-01, -2.2760e-01, -5.5282e-02, -1.9926e-01, -4.2303e-01,
         -4.4843e-01, -3.4285e-01,  6.9480e-02, -1.9455e-01, -1.7965e-01,
         -1.8138e-01, -3.8820e-01, -1.7423e-01, -4.1940e-01, -3.4280e-01,
         -3.0014e-01, -2.5780e-01, -1.6410e-01, -1.8455e-01, -9.5275e-03,
         -2.3140e-01, -1.1184e-01, -3.8905e-01, -1.1574e-01, -3.9388e-01,
         -2.4662e-01, -2.7486e-01, -2.5663e-01, -1.3048e-01,  6.4102e-02,
         -3.2227e-01, -5.3726e-01, -1.1962e-01, -2.9648e-01, -3.2251e-01,
          2.7459e-02,  2.3473e-02, -2.7633e-02, -4.5295e-01, -2.9259e-01,
         -3.5277e-01,  9.6935e-02, -1.9141e-01, -7.1883e-01,  2.2835e-01,
         -4.6592e-01, -6.7796e-02, -1.3304e-01, -1.4407e-01, -4.3686e-01,
         -3.4669e-01,  2.2316e-02,  1.7688e-01, -1.8531e-01, -1.5722e-01,
         -5.7849e-02, -3.3868e-01, -2.3903e-01, -4.3236e-01, -2.5153e-01,
         -2.5435e-01, -9.4945e-02, -4.7064e-01,  1.2207e-02, -6.6956e-04,
         -3.4272e-01,  8.8193e-03, -3.1854e-01, -1.6776e-01, -3.6134e-01,
         -1.1015e-01, -3.6055e-01, -2.2230e-01,  3.2846e-02, -9.8218e-02,
          1.7425e-02, -2.2338e-01, -2.0611e-01,  9.4687e-02, -4.4537e-01,
         -2.6131e-01,  3.2070e-02, -4.4505e-01, -3.4241e-02, -1.0706e-01,
         -2.0674e-01,  9.3392e-02, -3.4033e-01, -1.5073e-01, -2.0593e-01,
         -2.8641e-01,  5.5593e-02, -1.6778e-01, -5.1501e-01, -3.2847e-01,
         -6.6480e-02, -1.9768e-01, -2.1285e-01, -3.0261e-01, -3.0067e-01,
         -2.7578e-01, -9.9655e-01, -5.2154e-01, -6.0770e-02, -1.4022e-01,
         -1.9822e-03, -1.3371e-01, -1.9008e-01, -2.9683e-03, -2.5895e-01,
         -5.8709e-01, -4.7134e-02, -1.4123e-01, -5.5140e-02, -5.8663e-02,
         -5.2313e-01, -3.6260e-01,  1.1454e-02, -6.0566e-03, -3.0605e-01,
         -4.4603e-01, -4.8851e-01,  9.7395e-02, -1.5336e-01, -3.2929e-01,
          1.6552e-02, -1.2056e-01, -8.4626e-02, -4.1263e-01, -2.1797e-02,
          1.4101e-01, -2.1016e-01,  1.6854e-01, -4.3603e-01, -4.1908e-01,
          5.1888e-02, -5.7837e-01,  3.9563e-02, -5.1198e-02, -1.3849e-01,
         -4.4075e-02, -3.4178e-01, -3.5282e-01,  2.1214e-02,  1.5568e-03,
         -2.4381e-01, -2.2159e-01, -5.8164e-01, -2.3882e-01, -1.8957e-01,
         -1.5512e-01, -4.6046e-01, -3.7019e-01, -2.1117e-01, -2.1075e-01,
         -1.5351e-01, -1.3670e-01, -5.2622e-03, -5.1050e-01, -1.8602e-01,
         -9.9914e-02, -4.4217e-01, -2.3076e-02, -1.9092e-01, -1.3496e-01,
          1.0213e-01, -3.8440e-01, -4.3281e-01, -3.7471e-01, -3.2014e-01,
         -1.8805e-01, -4.1645e-01, -4.4079e-02, -2.2945e-01, -4.1491e-01,
```

Obtaining the European cities

Once we retrieve the embeddings, for each Indian city, we try to obtain the top European cities by computing the cosine similarity with each embedding of the European city.

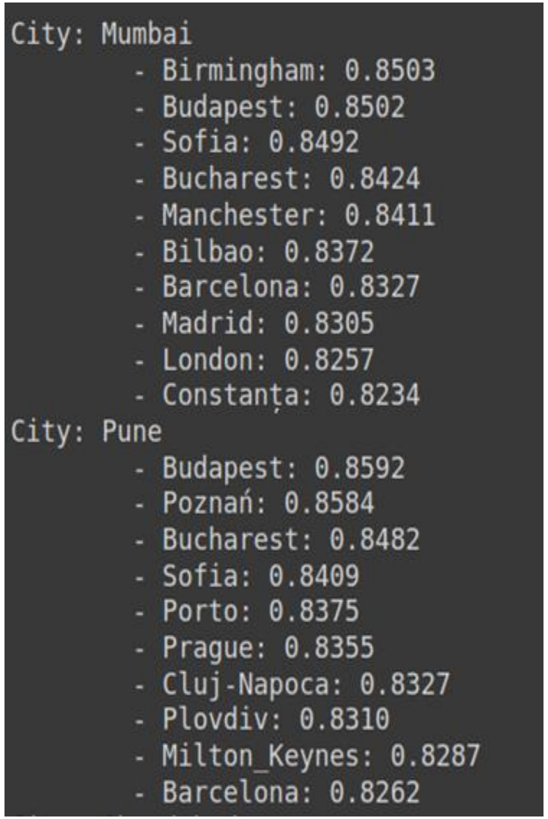


```
def compute_similarity(embedding1, embedding2):

    similarity = torch.nn.functional.cosine_similarity(embedding1, embedding2)
    return similarity.item()
```

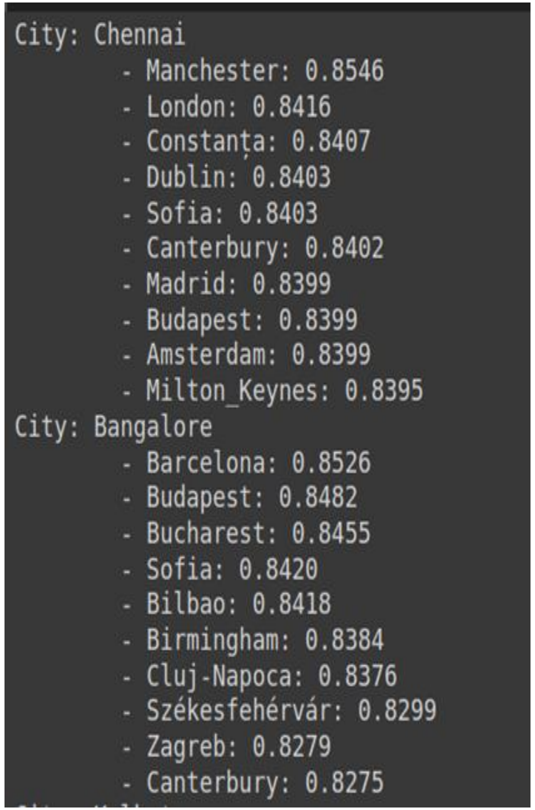


```
def find_top_similar_cities(indian_embedding, european_embeddings, european_city_names, top_k=5):
  similarities = []
  indian_embedding = indian_embedding.to(device)
  for i, european_embedding in enumerate(european_embeddings):
    european_embedding = european_embedding.to(device)

    similarity = compute_similarity(indian_embedding, european_embedding)
    similarities.append((european_city_names[i], similarity))
  similarities.sort(key=lambda x: x[1], reverse=True)  # Sort by similarity (descending)
  return similarities[:top_k]
```

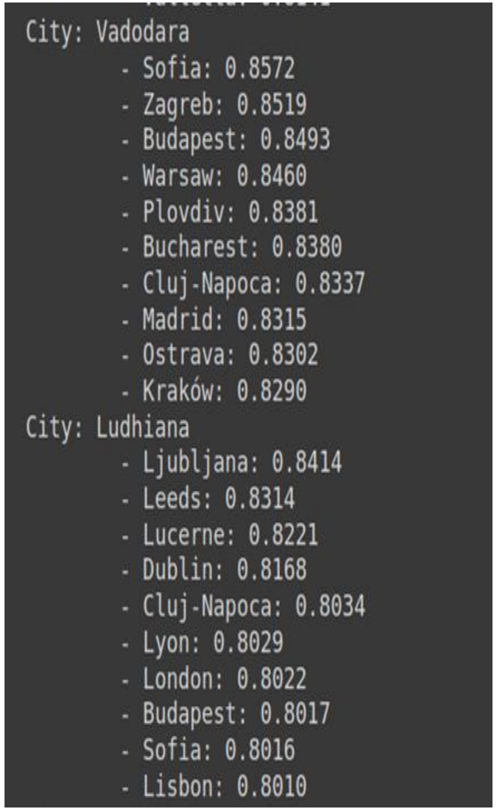
The top 10 European cities for some of the Indian cities are:



```
City: Mumbai
        - Birmingham: 0.8503
        - Budapest: 0.8502
        - Sofia: 0.8492
        - Bucharest: 0.8424
        - Manchester: 0.8411
        - Bilbao: 0.8372
        - Barcelona: 0.8327
        - Madrid: 0.8305
        - London: 0.8257
        - Constanța: 0.8234
City: Pune
        - Budapest: 0.8592
        - Poznań: 0.8584
        - Bucharest: 0.8482
        - Sofia: 0.8409
        - Porto: 0.8375
        - Prague: 0.8355
        - Cluj-Napoca: 0.8327
        - Plovdiv: 0.8310
        - Milton_Keynes: 0.8287
        - Barcelona: 0.8262
```



```
City: Chennai
        - Manchester: 0.8546
        - London: 0.8416
        - Constanța: 0.8407
        - Dublin: 0.8403
        - Sofia: 0.8403
        - Canterbury: 0.8402
        - Madrid: 0.8399
        - Budapest: 0.8399
        - Amsterdam: 0.8399
        - Milton_Keynes: 0.8395
City: Bangalore
        - Barcelona: 0.8526
        - Budapest: 0.8482
        - Bucharest: 0.8455
        - Sofia: 0.8420
        - Bilbao: 0.8418
        - Birmingham: 0.8384
        - Cluj-Napoca: 0.8376
        - Székesfehérvár: 0.8299
        - Zagreb: 0.8279
        - Canterbury: 0.8275
```



```
City: Vadodara
        - Sofia: 0.8572
        - Zagreb: 0.8519
        - Budapest: 0.8493
        - Warsaw: 0.8460
        - Plovdiv: 0.8381
        - Bucharest: 0.8380
        - Cluj-Napoca: 0.8337
        - Madrid: 0.8315
        - Ostrava: 0.8302
        - Kraków: 0.8290
City: Ludhiana
        - Ljubljana: 0.8414
        - Leeds: 0.8314
        - Lucerne: 0.8221
        - Dublin: 0.8168
        - Cluj-Napoca: 0.8034
        - Lyon: 0.8029
        - London: 0.8022
        - Budapest: 0.8017
        - Sofia: 0.8016
        - Lisbon: 0.8010
```

Can we do better?

The next step was to improve the representation(text embeddings) of each city. We wanted to make use of the city's images as well so that we could enhance our text representations.

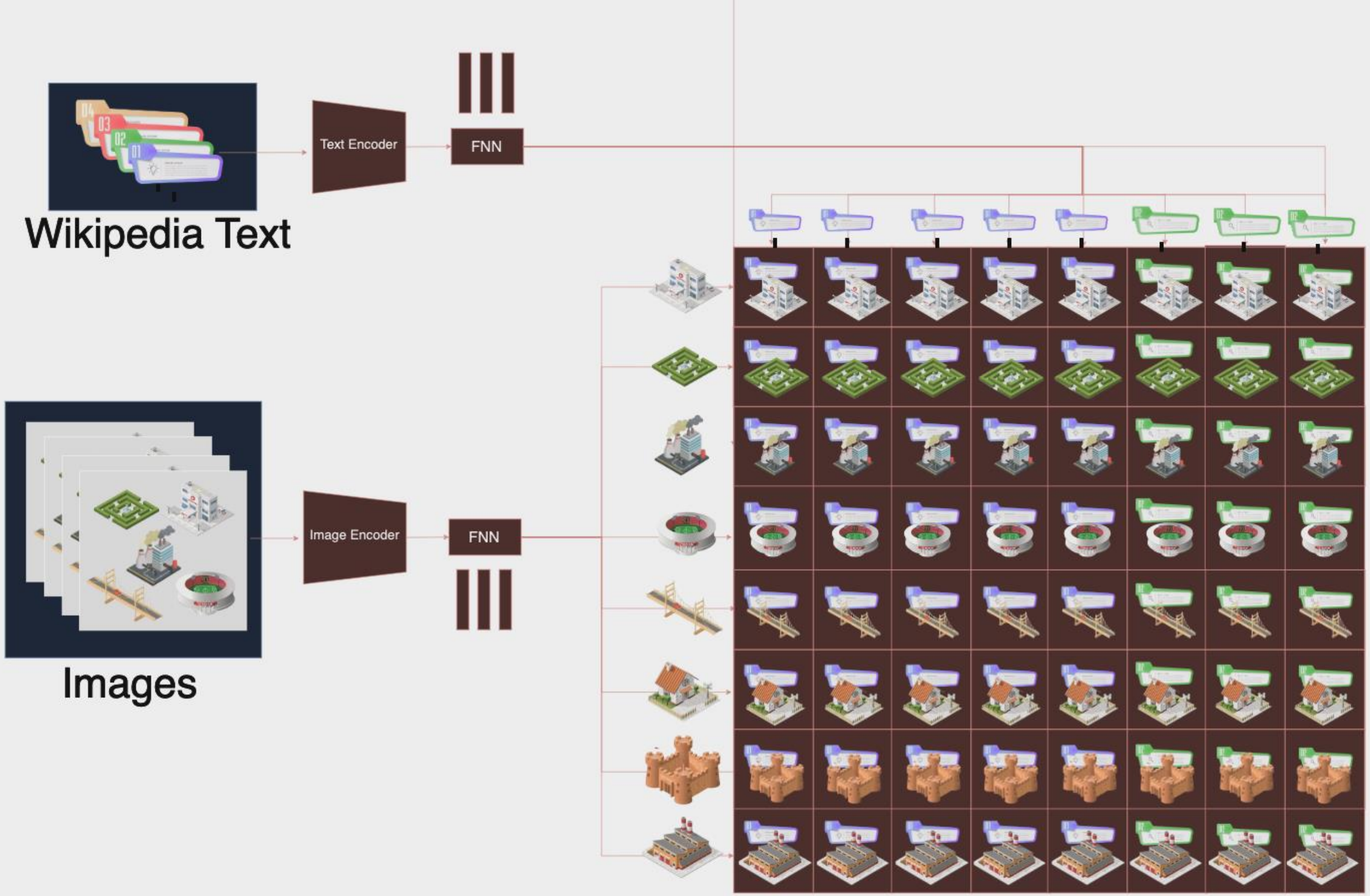
Hence for each city, we collected 5 images that can represent the city adequately. The pretrained ResNet50 model has been used to obtain the image embeddings.

But that's not enough, we have to make sure that the image embeddings and text embeddings for the same city are as close as possible and that for different cities are as far as possible. The text embeddings must be enhanced to represent the city better.

Hence we make use of the concept Contrastive Language-Image Pretraining(CLIP) model.

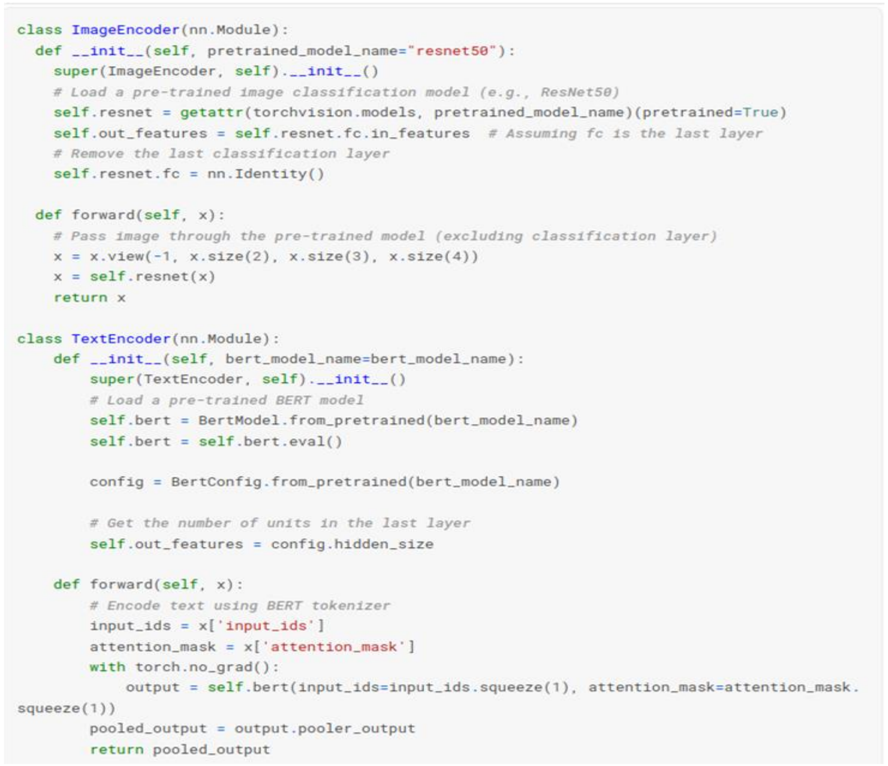
CLIP - Contrastive Language-Image Pretraining model

- CLIP (Contrastive Language-Image Pre-training) is a powerful model developed by OpenAI that learns to understand images and text in a shared space.
- The groundbreaking novel aspect of CLIP is that unlike its predecessors, it does not have to rely on massive, expensive datasets with manual labels, but instead learns from the wealth of text descriptions that already exist alongside images online.
- By leveraging contrastive learning, CLIP learns to associate similar semantics across diverse image-text pairs while discerning dissimilarities.
- This innovative approach enables CLIP to grasp nuanced relationships between visual and textual information, showcasing remarkable generalization, robustness, and adaptability across various downstream tasks.

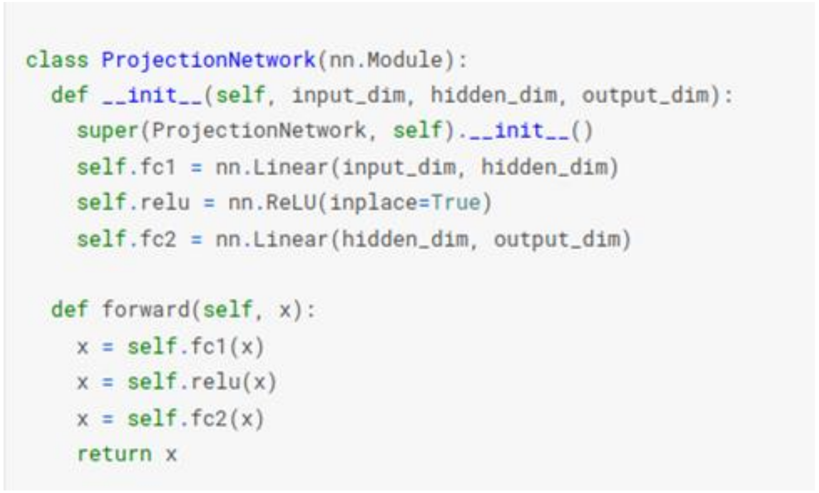

Wikipedia Text
Text Encoder
FNN
Images
Image Encoder
FNN

How does our model work

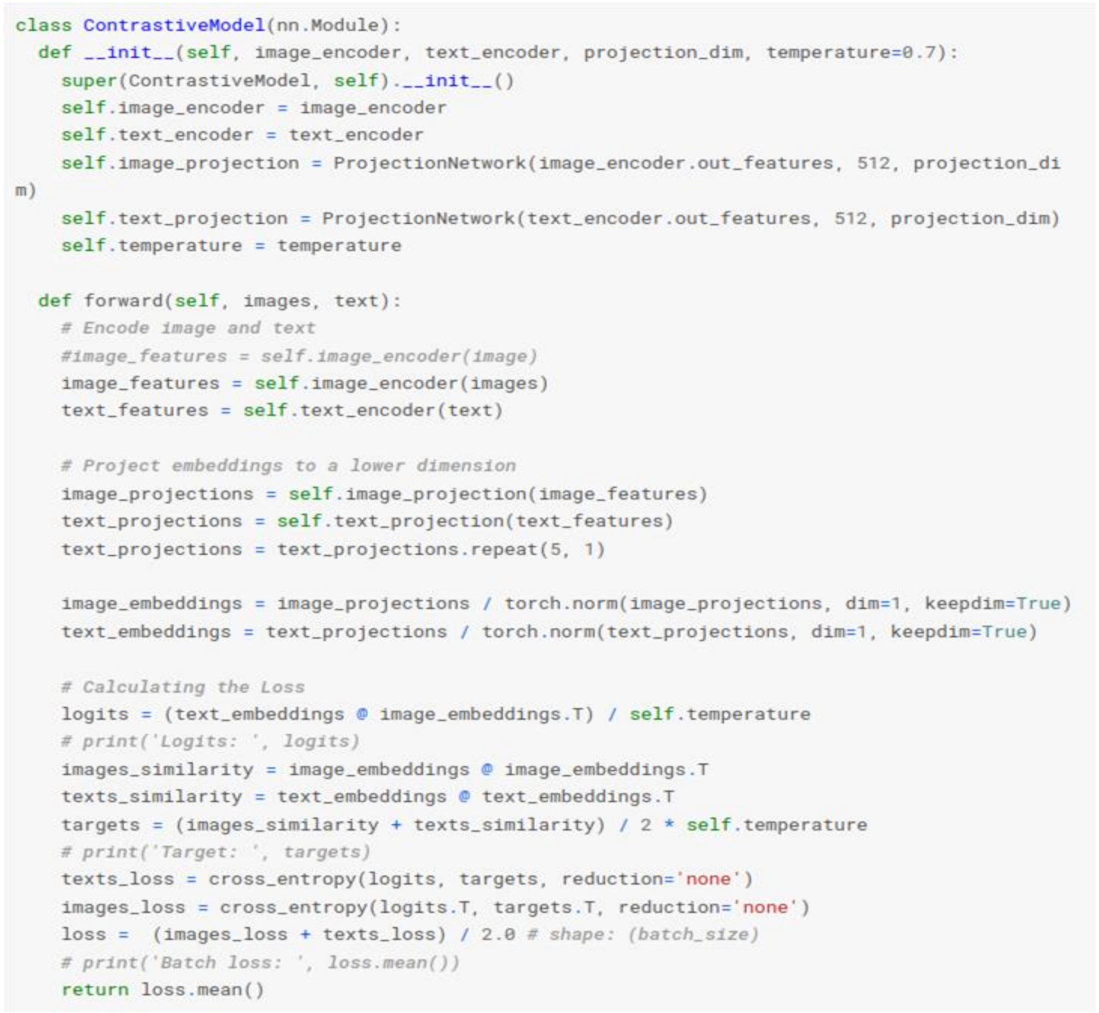
Model architecture:



```
class ImageEncoder(nn.Module):
  def __init__(self, pretrained_model_name="resnet50"):
    super(ImageEncoder, self).__init__()
    # Load a pre-trained image classification model (e.g., ResNet50)
    self.resnet = getattr(torchvision.models, pretrained_model_name)(pretrained=True)
    self.out_features = self.resnet.fc.in_features  # Assuming fc is the last layer
    # Remove the last classification layer
    self.resnet.fc = nn.Identity()

  def forward(self, x):
    # Pass image through the pre-trained model (excluding classification layer)
    x = x.view(-1, x.size(2), x.size(3), x.size(4))
    x = self.resnet(x)
    return x

class TextEncoder(nn.Module):
    def __init__(self, bert_model_name=bert_model_name):
        super(TextEncoder, self).__init__()
        # Load a pre-trained BERT model
        self.bert = BertModel.from_pretrained(bert_model_name)
        self.bert = self.bert.eval()

        config = BertConfig.from_pretrained(bert_model_name)

        # Get the number of units in the last layer
        self.out_features = config.hidden_size

    def forward(self, x):
        # Encode text using BERT tokenizer
        input_ids = x['input_ids']
        attention_mask = x['attention_mask']
        with torch.no_grad():
            output = self.bert(input_ids=input_ids.squeeze(1), attention_mask=attention_mask.
squeeze(1))
        pooled_output = output.pooler_output
        return pooled_output
```



```
class ProjectionNetwork(nn.Module):
  def __init__(self, input_dim, hidden_dim, output_dim):
    super(ProjectionNetwork, self).__init__()
    self.fc1 = nn.Linear(input_dim, hidden_dim)
    self.relu = nn.ReLU(inplace=True)
    self.fc2 = nn.Linear(hidden_dim, output_dim)

  def forward(self, x):
    x = self.fc1(x)
    x = self.relu(x)
    x = self.fc2(x)
    return x
```



```python
class ContrastiveModel(nn.Module):
  def __init__(self, image_encoder, text_encoder, projection_dim, temperature=0.7):
    super(ContrastiveModel, self).__init__()
    self.image_encoder = image_encoder
    self.text_encoder = text_encoder
    self.image_projection = ProjectionNetwork(image_encoder.out_features, 512, projection_di
m)
    self.text_projection = ProjectionNetwork(text_encoder.out_features, 512, projection_dim)
    self.temperature = temperature

  def forward(self, images, text):
    # Encode image and text
    #image_features = self.image_encoder(image)
    image_features = self.image_encoder(images)
    text_features = self.text_encoder(text)

    # Project embeddings to a lower dimension
    image_projections = self.image_projection(image_features)
    text_projections = self.text_projection(text_features)
    text_projections = text_projections.repeat(5, 1)

    image_embeddings = image_projections / torch.norm(image_projections, dim=1, keepdim=True)
    text_embeddings = text_projections / torch.norm(text_projections, dim=1, keepdim=True)

    # Calculating the Loss
    logits = (text_embeddings @ image_embeddings.T) / self.temperature
    # print('Logits: ', logits)
    images_similarity = image_embeddings @ image_embeddings.T
    texts_similarity = text_embeddings @ text_embeddings.T
    targets = (images_similarity + texts_similarity) / 2 * self.temperature
    # print('Target: ', targets)
    texts_loss = cross_entropy(logits, targets, reduction='none')
    images_loss = cross_entropy(logits.T, targets.T, reduction='none')
    loss =  (images_loss + texts_loss) / 2.0 # shape: (batch_size)
    # print('Batch loss: ', loss.mean())
    return loss.mean()
```

- For each of the cities in our dataset, we use the pretrained ResNet model and the BERT model to obtain the image embeddings for all 5 images and the text embeddings.

- We pass these embeddings into two separate neural networks to bring down the dimension size to 500.

- Next, we compute the similarity between these embeddings and set up the binary cross entropy loss.

- Then we back propagate the loss to update the model parameters.

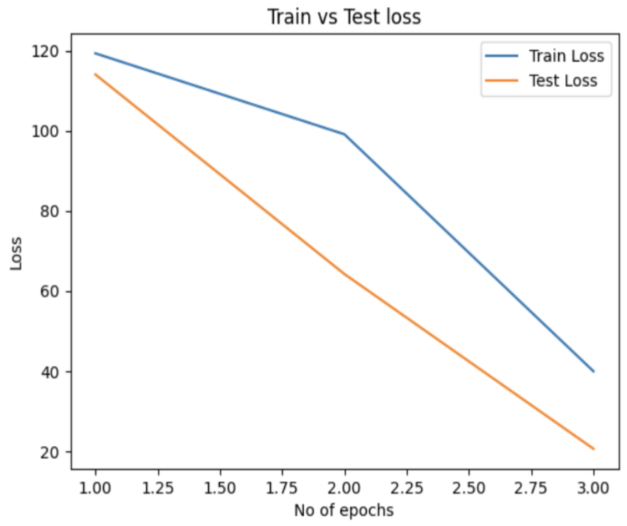
HyperParameter Tuning

HyperParameter Tuning on Learning rate

A good learning rate balances the speed and stability of learning. Too slow, and the model takes forever to learn. Too fast, and it might miss the optimal solution or become unstable. The choice of learning rate can vary depending on the type of gradient descent you're using: batch gradient descent, mini-batch gradient descent, and stochastic gradient descent. Due to the smoother gradient with the whole dataset, BGD can often tolerate slightly higher learning rates compared to SGD. However, it's still important to find a balance to avoid large jumps and overshooting the minimum. To compensate for the noisier gradient, SGD typically requires smaller learning rates compared to BGD. A smaller learning rate helps to average out the fluctuations and prevent the model from jumping around too much. Mini-batch GD can often use learning rates between those of BGD and SGD. The ideal rate depends on the specific mini-batch size. A larger mini-batch size gets closer to BGD behavior, allowing for potentially slightly higher learning rates. There's a general rule that suggests you can increase the learning rate proportionally as you increase the mini-batch size. This is because with a larger batch, the gradient estimate becomes smoother, allowing for potentially larger steps. However, it's not a strict proportionality.

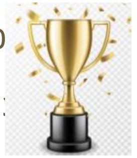
Comparison on Learning Rate

Learning rate = 0.001
Dimension = 10
Number of epochs = 3

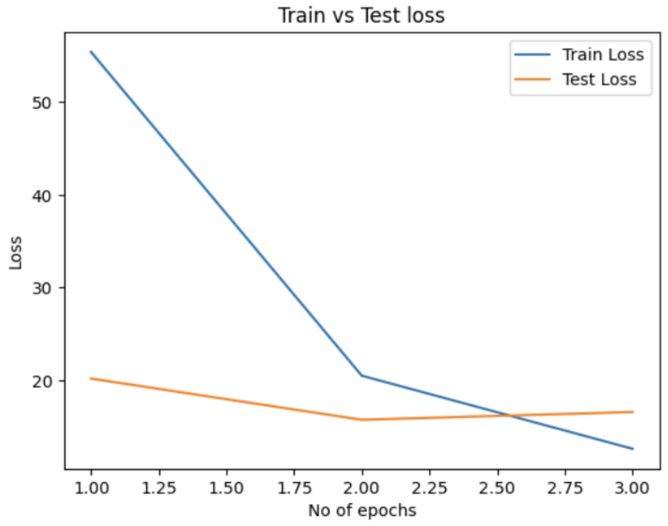


Training Loss = 39.9128
Test Loss = 20.5651

Learning rate = 0.000
Dimension = 10
Number of epochs =



CLEAR WIN
ON LOSS

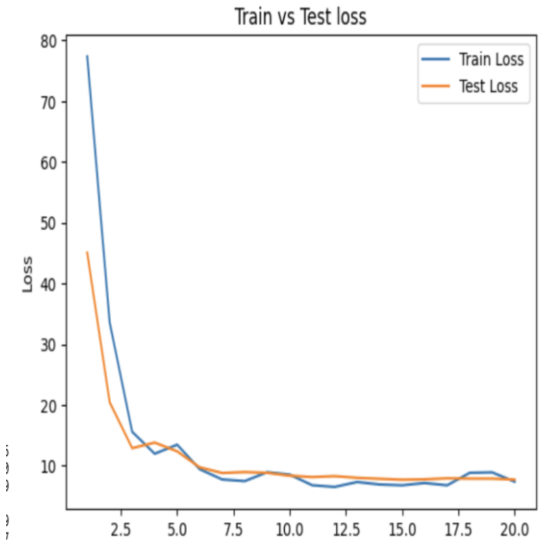


Training Loss = 12.5957
Test Loss = 16.5623

Comparison on Dimension of the Feature Vector

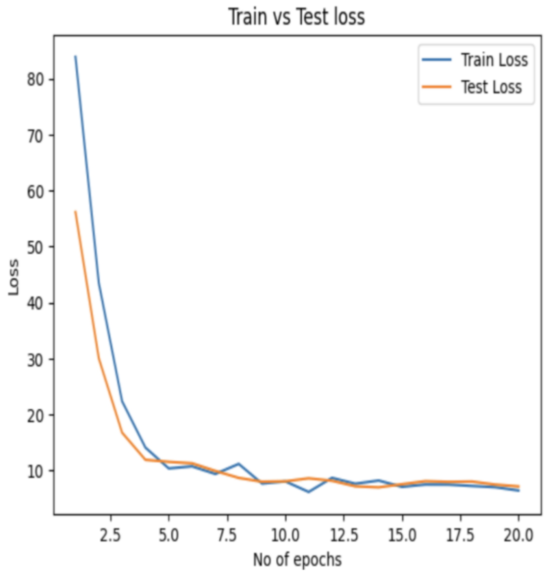
Learning Rate Considered : 0.0001

Dimension = 100
Number of epochs = 20



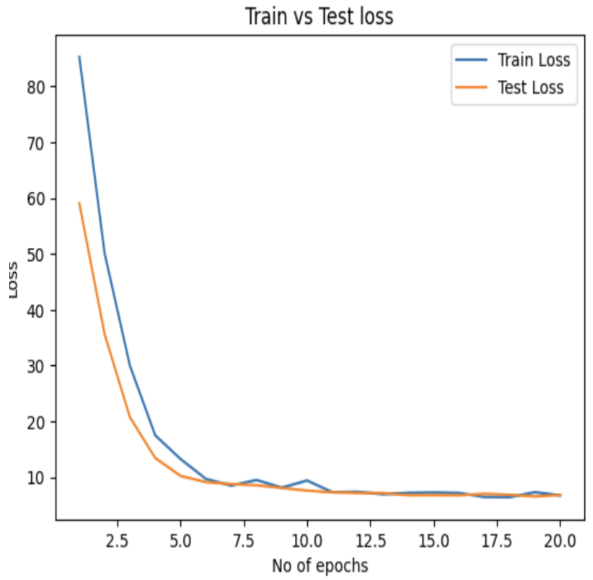
Training Loss : 7.3766
Test Loss : 7.7239

Dimension = 300
Number of epochs = 20



Training Loss : 6.4573
Test Loss : 7.1789

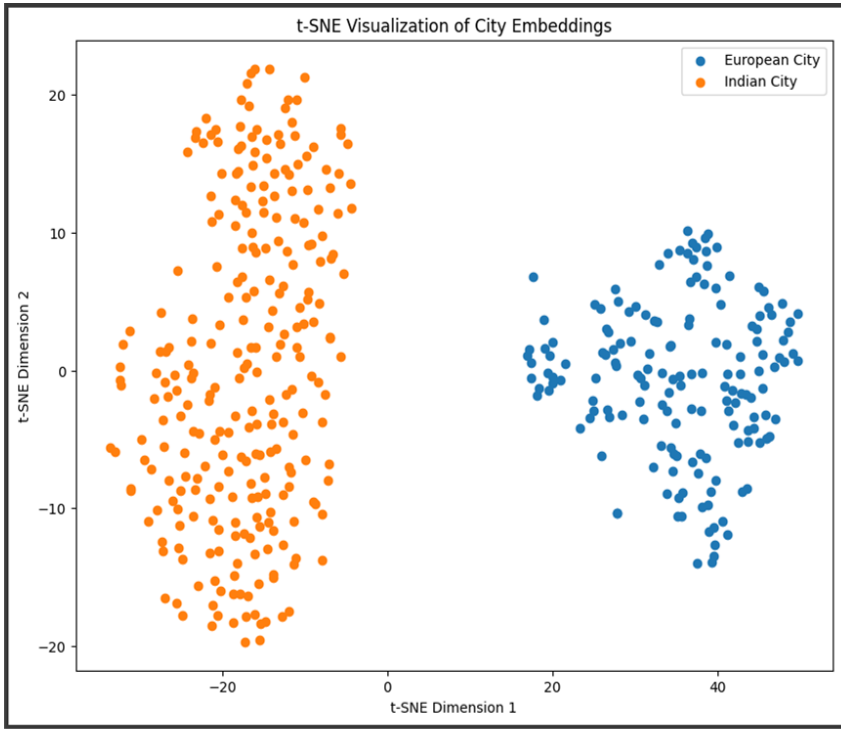
Dimension = 500
Number of epochs = 20



Training Loss : 6.7715
Test Loss : 6.8300

Visualising the city embeddings

- Once the training is complete, we obtain the embeddings for all our cities in our dataset.
- In order to analyze how good or bad the embeddings are, we have used the technique of t-SNE, which is a popular technique used for visualizing high-dimensional data in a low-dimensional space.
- We can observe that the inter class separability of the Indian and European cities is very high, hence we continue to use the model.



Top 4 European cities similar to Ludhiana

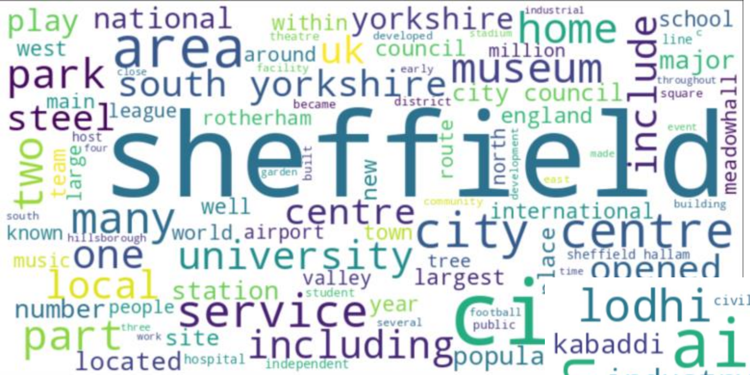
Our model gave highest Pearson Correlation coefficient for Ludhiana (known for Industrial development, textile, educational institutes)

Modified CLIP Model

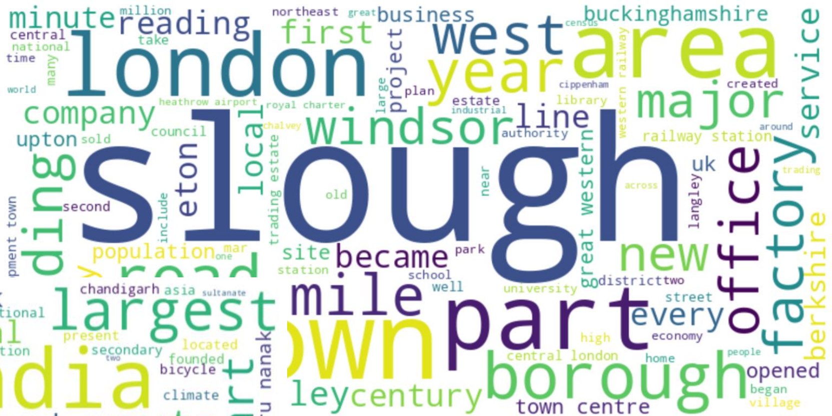
1. Sheffield - Similarity 0.8454
2. Slough - Similarity 0.8356
3. Leeds - Similarity 0.8334
4. Milton Keynes - Similarity 0.8262

Before applying CLIP

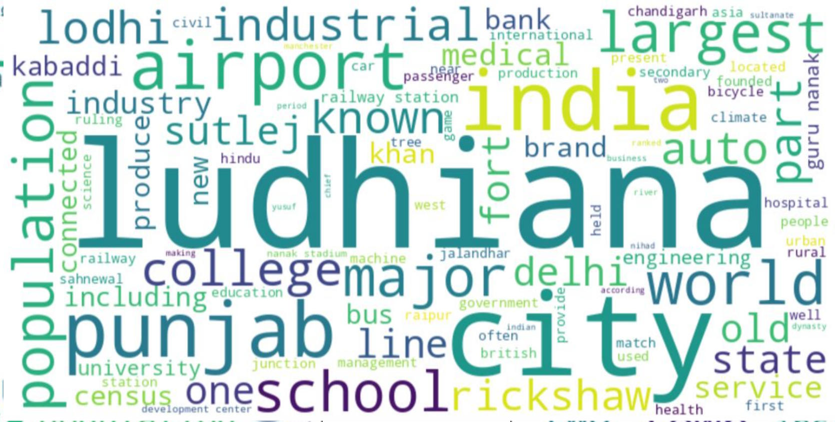
Sheffield was at Rank 25 - Similarity 0.7848
Slough was at Rank 32 - Similarity 0.7798
Leeds was at Rank 2 - Similarity 0.8310
Milton Keynes was at Rank 12 - Similarity 0.7940



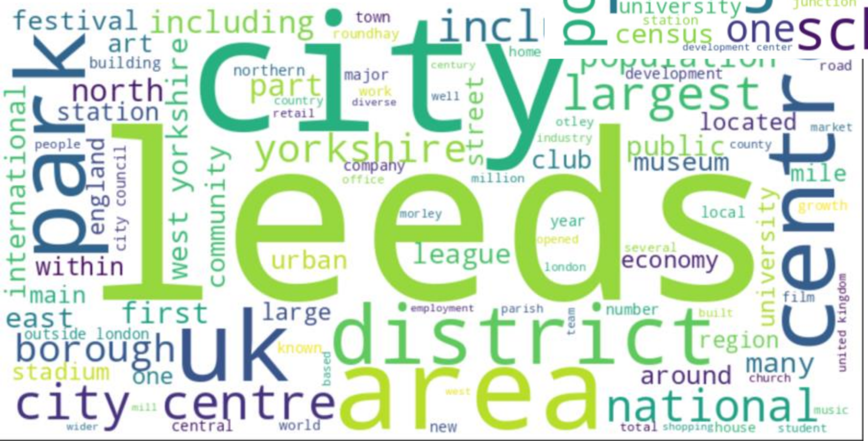
Similarity = 0.8454



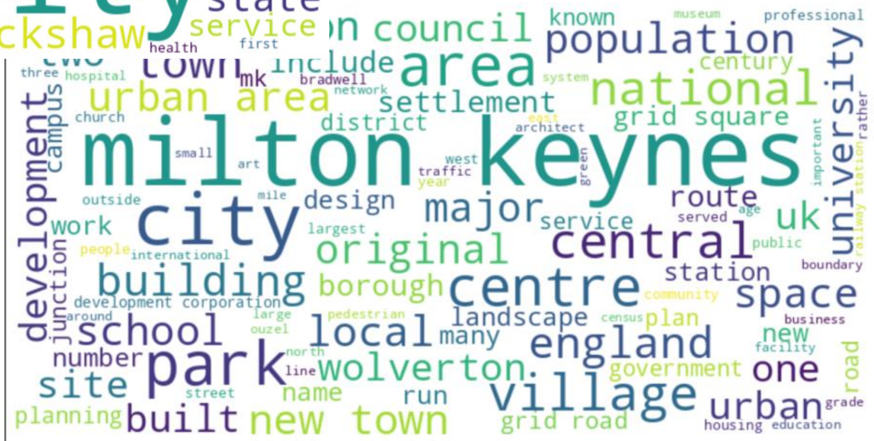
Similarity = 0.8356



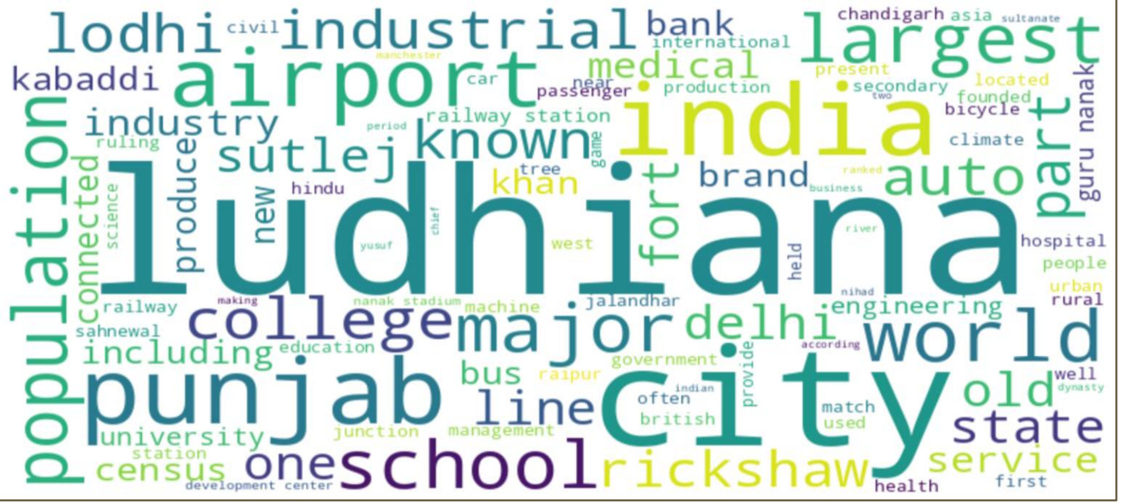
Similarity = 0.8334

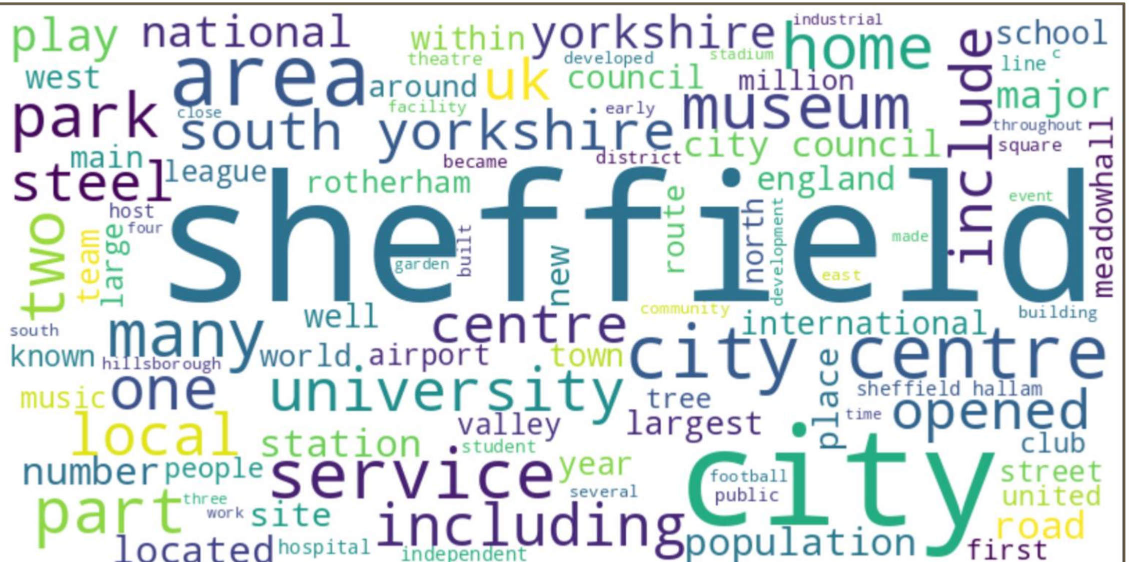


Similarity = 0.8262



Inferences





Common Words

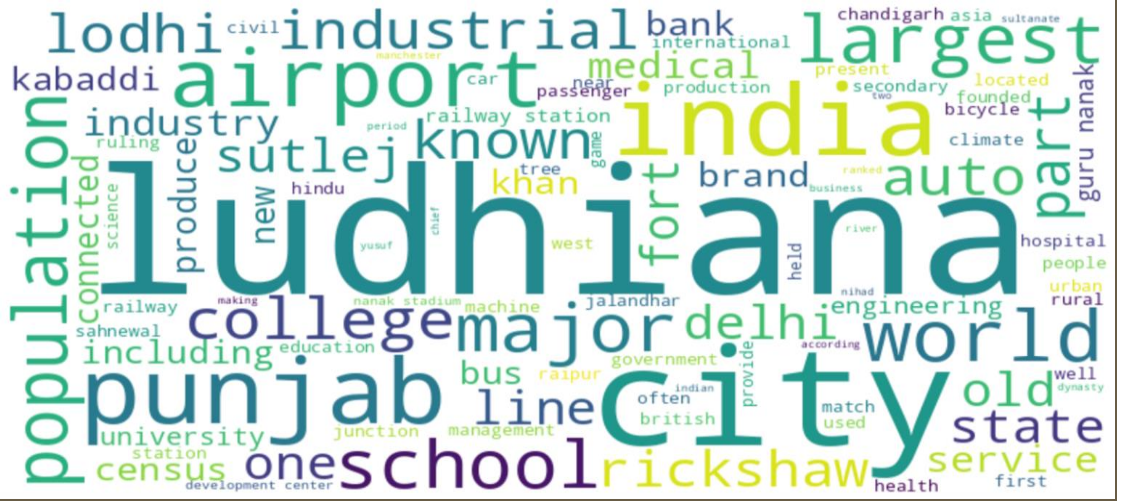
University, college, school, education
Industry, manchester
Stadium
Population
Hospital, medical
Park, bicycle
Airport

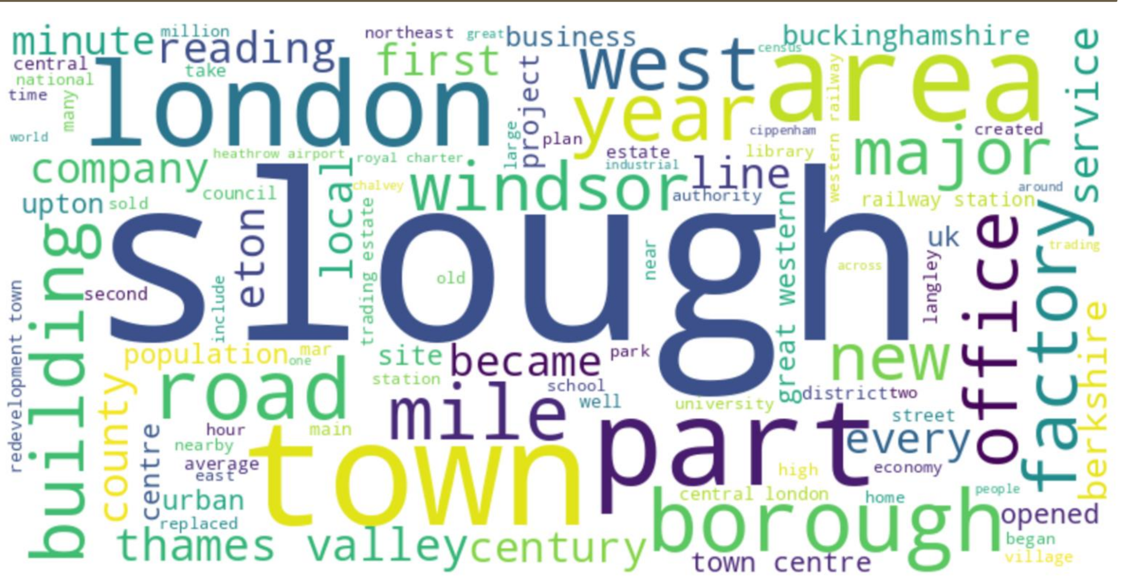
Matched on Center of Education

Sheffield is a center of education and boasts 2 world class univs - The University of Sheffield and Hallam University
Ludhiana has 363 senior secondary, 367 high, 324 middle, 1129 primary

Ludhiana

Sheffield



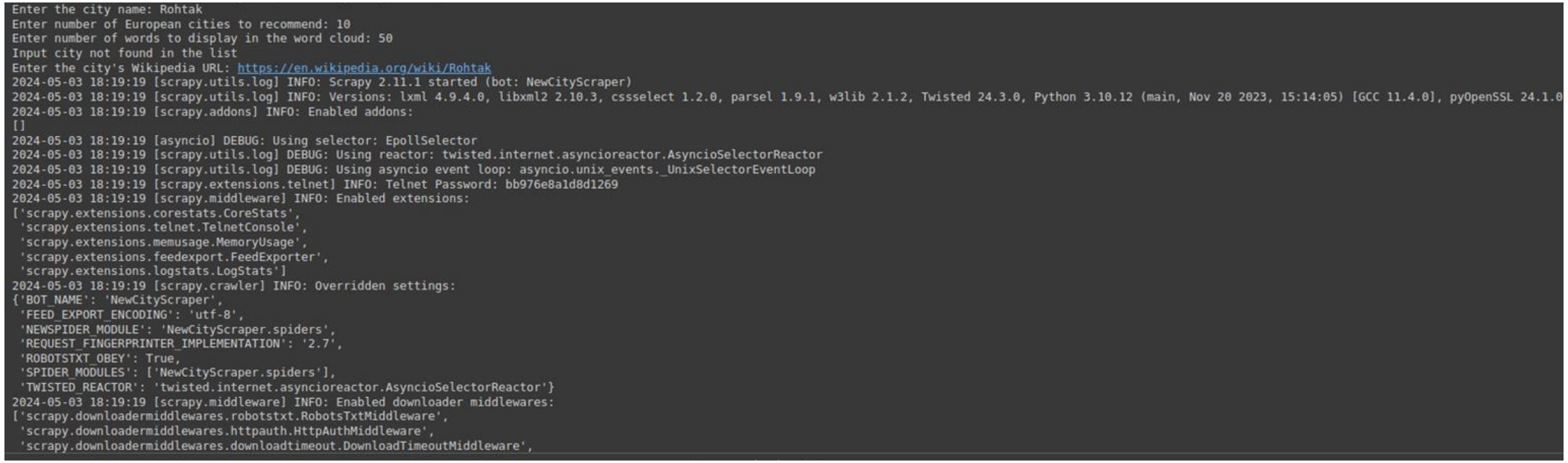


Common Words

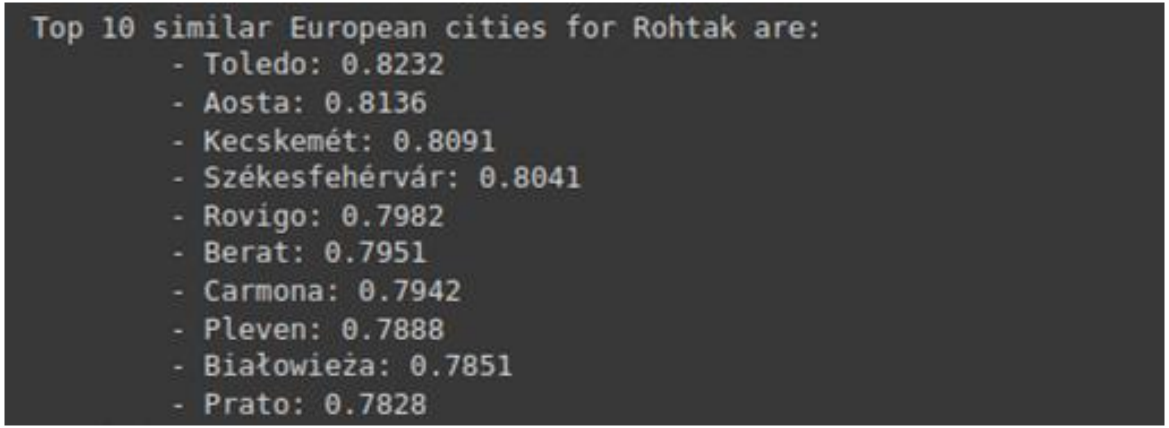
Industrial, manchester, office, factory, business, company, engineering Railway, station, roads

Matched on Industrial estates

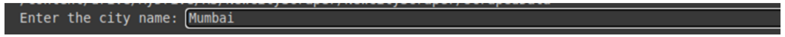
Slough is the largest industrial estate of Europe providing 17,000 jobs cross 400 businesses.
Ludhiana is known for its textile industry



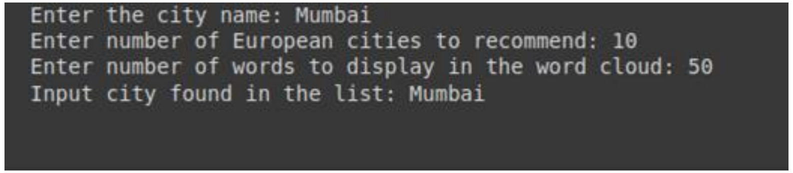
```
Enter the city name: Rohtak
Enter number of European cities to recommend: 10
Enter number of words to display in the word cloud: 50
Input city not found in the list
Enter the city's Wikipedia URL: https://en.wikipedia.org/wiki/Rohtak
2024-05-03 18:19:19 [scrapy.utils.log] INFO: Scrapy 2.11.1 started (bot: NewCityScraper)
2024-05-03 18:19:19 [scrapy.utils.log] INFO: Versions: lxml 4.9.4.0, libxml2 2.10.3, cssselect 1.2.0, parsel 1.9.1, w3lib 2.1.2, Twisted 24.3.0, Python 3.10.12 (main, Nov 20 2023, 15:14:05) [GCC 11.4.0], pyOpenSSL 24.1.0
2024-05-03 18:19:19 [scrapy.addons] INFO: Enabled addons:
[]
2024-05-03 18:19:19 [asyncio] DEBUG: Using selector: EpollSelector
2024-05-03 18:19:19 [scrapy.utils.log] DEBUG: Using reactor: twisted.internet.asyncioreactor.AsyncioSelectorReactor
2024-05-03 18:19:19 [scrapy.utils.log] DEBUG: Using asyncio event loop: asyncio.unix_events._UnixSelectorEventLoop
2024-05-03 18:19:19 [scrapy.extensions.telnet] INFO: Telnet Password: bb976e8a1d8d1269
2024-05-03 18:19:19 [scrapy.middleware] INFO: Enabled extensions:
['scrapy.extensions.corestats.CoreStats',
 'scrapy.extensions.telnet.TelnetConsole',
 'scrapy.extensions.memusage.MemoryUsage',
 'scrapy.extensions.feedexport.FeedExporter',
 'scrapy.extensions.logstats.LogStats']
2024-05-03 18:19:19 [scrapy.crawler] INFO: Overridden settings:
{'BOT_NAME': 'NewCityScraper',
 'FEED_EXPORT_ENCODING': 'utf-8',
 'NEWSPIDER_MODULE': 'NewCityScraper.spiders',
 'REQUEST_FINGERPRINTER_IMPLEMENTATION': '2.7',
 'ROBOTSTXT_OBEY': True,
 'SPIDER_MODULES': ['NewCityScraper.spiders'],
 'TWISTED_REACTOR': 'twisted.internet.asyncioreactor.AsyncioSelectorReactor'}
2024-05-03 18:19:19 [scrapy.middleware] INFO: Enabled downloader middlewares:
['scrapy.downloadermiddlewares.robotstxt.RobotsTxtMiddleware',
 'scrapy.downloadermiddlewares.httpauth.HttpAuthMiddleware',
 'scrapy.downloadermiddlewares.downloadtimeout.DownloadTimeoutMiddleware',
```



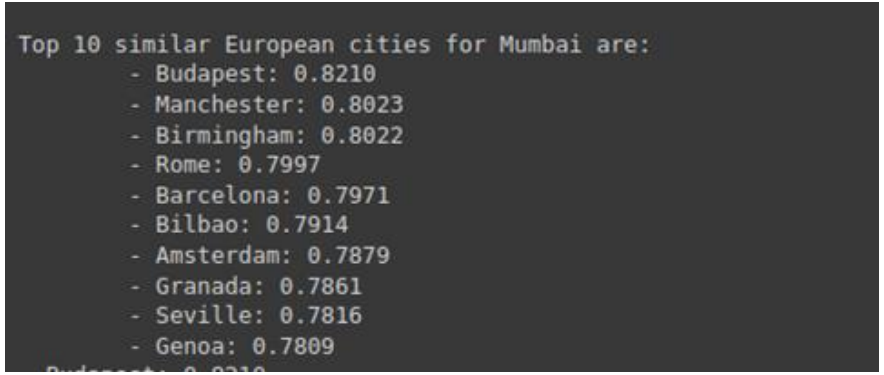
```
Top 10 similar European cities for Rohtak are:
        - Toledo: 0.8232
        - Aosta: 0.8136
        - Kecskemét: 0.8091
        - Székesfehérvár: 0.8041
        - Rovigo: 0.7982
        - Berat: 0.7951
        - Carmona: 0.7942
        - Pleven: 0.7888
        - Białowieża: 0.7851
        - Prato: 0.7828
```



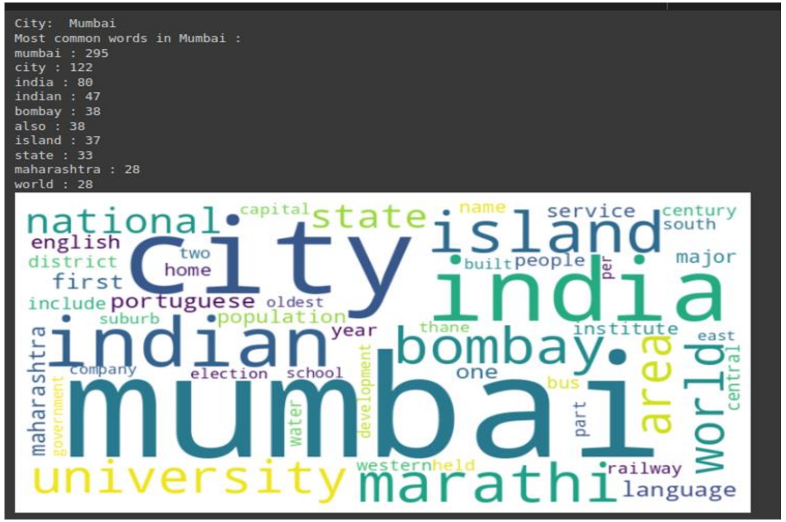
```
Enter the city name: Mumbai
```



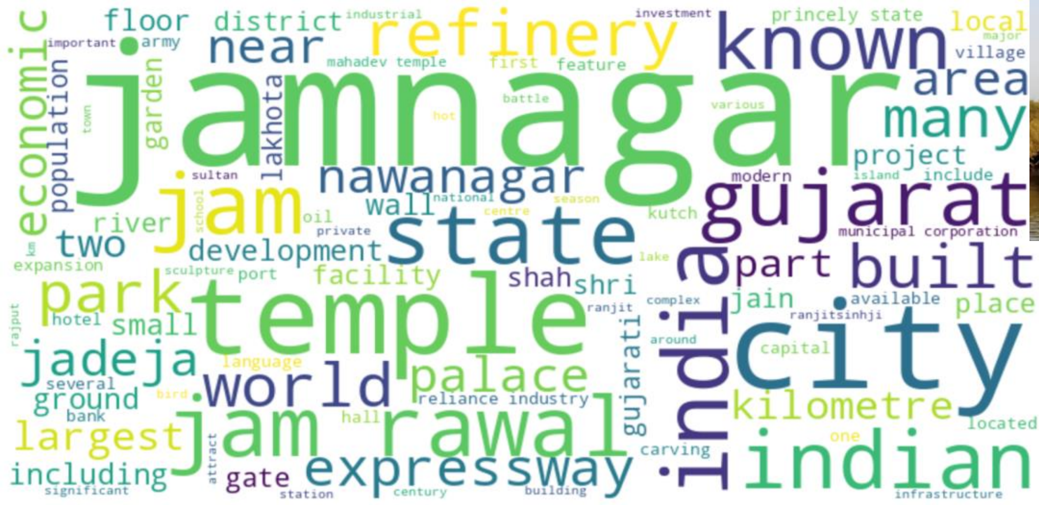
```
Enter the city name: Mumbai
Enter number of European cities to recommend: 10
Enter number of words to display in the word cloud: 50
Input city found in the list: Mumbai
```



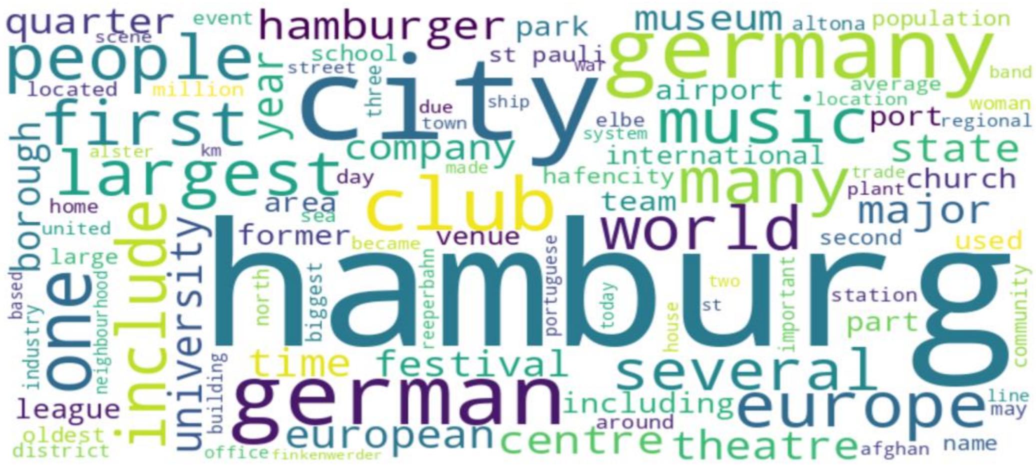
```
Top 10 similar European cities for Mumbai are:
        - Budapest: 0.8210
        - Manchester: 0.8023
        - Birmingham: 0.8022
        - Rome: 0.7997
        - Barcelona: 0.7971
        - Bilbao: 0.7914
        - Amsterdam: 0.7879
        - Granada: 0.7861
        - Seville: 0.7816
        - Genoa: 0.7809
  Budapest: 0.8210
```



```
City:  Mumbai
Most common words in Mumbai :
mumbai : 295
city : 122
india : 80
indian : 47
bombay : 38
also : 38
island : 37
state : 33
maharashtra : 28
world : 28
```



Similarity = 0.6382



THANK YOU