TRAVEL RECOMMENDATION SYSTEM

INDEX

ABSTRACT

Tourism plays a pivotal role in bolstering a nation's economic landscape. However, despite its significance, there remains a conspicuous absence of a comprehensive platform catering to the personalized needs of tourists seeking information on attractions. The establishment of a system adept at furnishing tailored and precise insights into local attractions, culinary delights, and shopping avenues stands poised to revolutionize the tourist experience. In determining item similarities, the cosine similarity method is employed, facilitating a nuanced understanding of user preferences. Furthermore, the adoption of a model-based collaborative filtering strategy enhances recommendation accuracy. A weighted hybridization technique is deployed to amalgamate the outcomes of both methodologies seamlessly. Through meticulous data collection encompassing tourist attractions and user preferences, the efficacy of this approach is substantiated. Empirical findings demonstrate that this hybrid approach outperforms conventional content-based and collaborative filtering methods in isolation, underscoring its potential to elevate the tourist experience to unprecedented levels of satisfaction and engagement.In today's fast-paced world, travel has become an integral part of our lives, offering us opportunities for exploration, relaxation, and cultural immersion. With the advent of technology, travelers now have access to a vast array of information and resources to plan their journeys. However, the abundance of options can often lead to decision paralysis and overwhelm, hindering the overall travel experience.

To address this challenge, we present a cutting-edge Travel Recommendation System designed to streamline the travel planning process and enhance the overall journey. By leveraging advanced algorithms and data analytics, our system aims to provide personalized recommendations tailored to each traveler's preferences, interests, and budget constraints. Whether it's discovering hidden gems in a bustling city, embarking on a culinary adventure, or unwinding in a tranquil getaway, our platform endeavors to cater to diverse travel aspirations.

Our recommendation system utilizes machine learning algorithms to analyze user preferences, past travel history, and demographic data to deliver personalized recommendations. By understanding each traveler's unique preferences, we ensure that recommendations are highly relevant and aligned with individual interests. development and deployment of a personalized recommendation system tailored for travelers.

Problem Statement :

The problem we're addressing is the overwhelming amount of travel-related information available online, making it challenging for travellers to find recommendations that align with their preferences and interests. Traditional travel websites often provide generic suggestions that may not cater to individual needs, leading to frustration and dissatisfaction among users.

OBJECTIVE:

1. Develop a recommendation engine capable of understanding user preferences and behavior.
2. Utilize machine learning techniques to analyze user data and generate personalized recommendations.
3. Deploy the recommendation system as a web interface for user interaction and feedback.
4. Evaluate the effectiveness of the system through user testing and feedback mechanisms.
5. Incorporate real-time updates and feedback mechanisms to continuously improve recommendation accuracy and relevance.

SIGNIFICANCE :

Our personalized recommendation system addresses the need for efficient and tailored travel recommendations in an era of information overload. By leveraging AI and data science techniques, we aim to enhance the travel experience for users, saving them time and effort while ensuring they discover destinations and activities that truly match their preferences.

Objectives:

Personalization:

Objective: Tailor recommendations based on individual user preferences, behavior, and past travel history.

Goal: Increase user engagement and satisfaction by providing relevant and personalized travel suggestions.

Accuracy:

Objective: Deliver accurate recommendations that match users' interests and preferences.

Goal: Minimize irrelevant suggestions and enhance trust in the recommendation system.

Efficiency:

Objective: Optimize system performance to provide timely recommendations, even with large datasets.

Goal: Reduce latency in generating recommendations and improve overall system responsiveness.

Diversity:

Objective: Offer diverse and varied travel suggestions to cater to different user preferences and interests.

Goal: Ensure recommendations cover a wide range of destinations, activities, and accommodation options.

User Engagement:

Objective: Encourage user interaction and exploration through intuitive user interfaces and engaging features.

Goal: Increase the time spent on the platform and the frequency of return visits by providing compelling recommendations and content.

Sustainability:

Objective: Promote sustainable travel practices by recommending eco-friendly accommodations, transportation options, and activities.

Goal: Encourage users to make environmentally conscious travel choices and reduce their carbon footprint.

Adaptability:

Objective: Continuously adapt and improve recommendation algorithms based on user feedback and evolving preferences.

Goal: Enhance the system's ability to adapt to changing user needs and trends in the travel industry.

Trustworthiness:

Objective: Establish trust and credibility with users by providing transparent explanations for recommendation decisions.

Goal: Increase user confidence in the system's recommendations by offering clear rationales and explanations for suggested destinations and activities.

Accessibility:

Objective: Ensure the recommendation system is accessible to users across different devices and platforms.

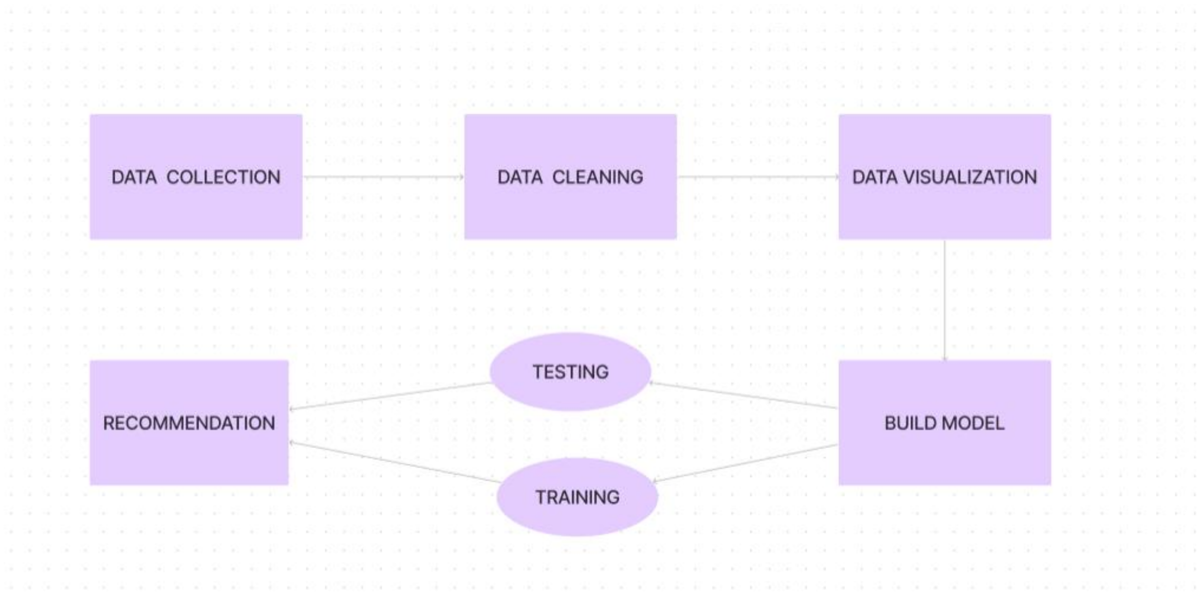
Goal: Enable users to access travel recommendations seamlessly from desktops, smartphones, and other devices, improving overall user convenience.

Monetization (optional):

Objective: Generate revenue through affiliate marketing, sponsored content, or premium subscription models.

Goal: Explore opportunities to monetize the recommendation system while maintaining user trust and satisfaction.

PROJECT PROCESS FLOW:



1. Initially, dataset is collected from Kaagle.
2. In our project, there are three datasets which are rating,users and tourism.
3. We merge the required data into a single dataset to perform analysis.
4. Dataset is already cleaned,so there is no need to perform data pre processing.
5. Data visualization is done to get the insights from data.
6. The model is built, two different models are build namely content based and collaborative based filtering.
7. Dataset is splitted into training data and testing data.
8. Finally, the model is saved for future purpose.
9. By using streamlit, the web application is developed.
10. The application takes the user current place and recommend five destinations to visit.

METHODOLOGY

The methodology for developing the personalized travel recommendation system comprises six key stages:

1.Data Collection and Preprocessing:

- Collect user preferences through a form presented to users, capturing information such as vacation type, preferred destinations, and specific interests.
- Preprocess the collected data to handle missing values, normalize data, and encode categorical variables.

2.Data Extraction:

- Extract relevant information from the preprocessed data, including user preferences, profiles, and previous travel experiences.
- Use this extracted data as the foundation for generating personalized recommendations.

3.Establishing User-Place Relations:

- Analyze user preferences and previous experiences to establish relationships between users and travel destinations.
- Understand individual preferences and interests to determine the suitability of different travel destinations for each user.

4.Data Splitting:

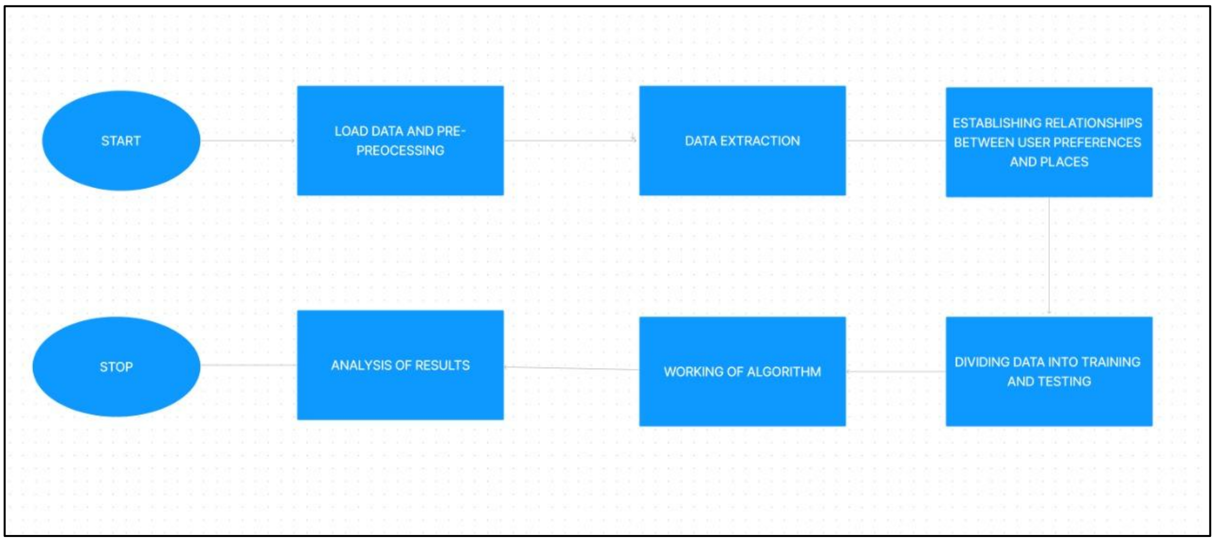
- Divide the processed data into training and test sets.
- Utilize the training set to train the machine learning model and the test set to evaluate its performance.

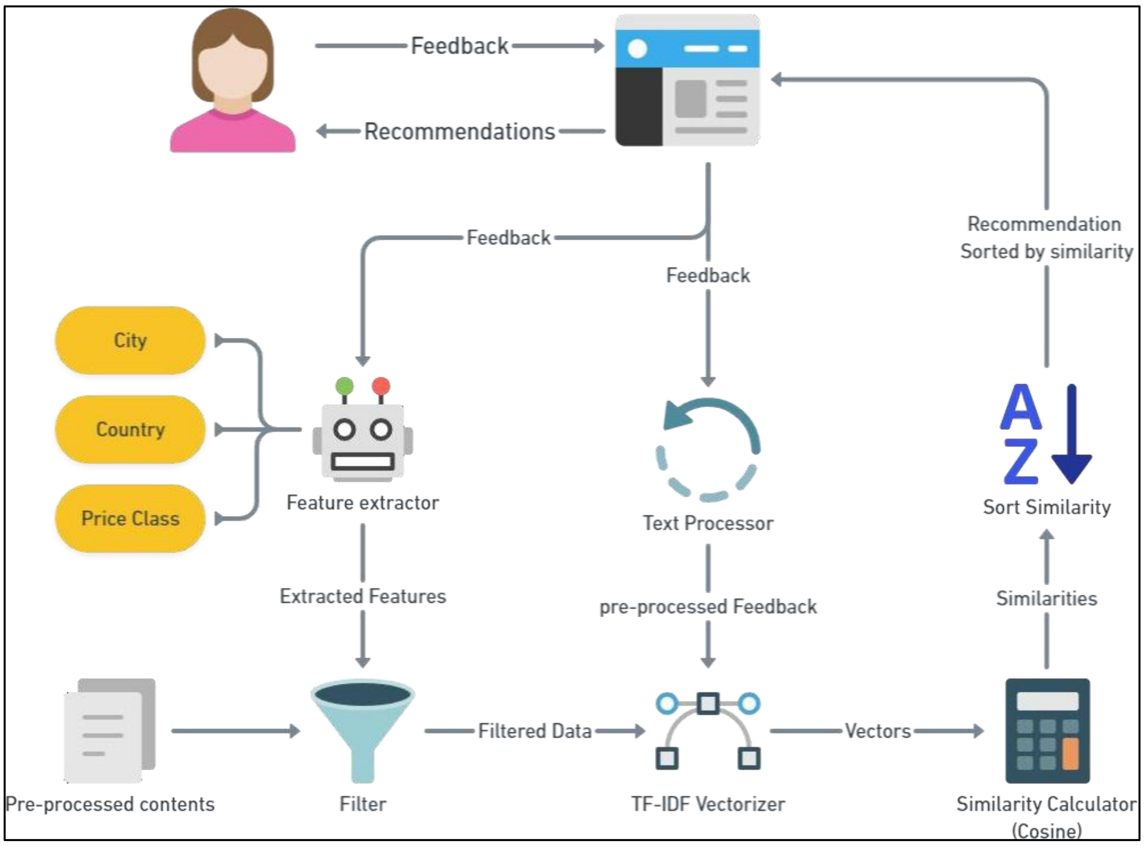
5.Algorithm Implementation:

- Employ machine learning techniques such as cosine similarity and) to analyze user data and generate personalized recommendations.
- Utilize the extracted features and preferences to identify the best attractions and places for each user.

6.Evaluation and Analysis:

- Analyze the results obtained from the recommendation system.
- Evaluate the accuracy and effectiveness of the system by comparing the recommended attractions with the user's actual preferences and feedback.
- Assess the performance of the system and identify areas for improvement based on the analysis.


START
LOAD DATA AND PRE-PREOCESSING
DATA EXTRACTION
ESTABLISHING RELATIONSHIPS BETWEEN USER PREFERENCES AND PLACES
DIVIDING DATA INTO TRAINING AND TESTING
WORKING OF ALGORITHM
ANALYSIS OF RESULTS
STOP


Feedback
Recommendations
Feedback
Feedback
Recommendation Sorted by similarity
City
Country
Price Class
Feature extractor
Text Processor
Sort Similarity
Extracted Features
pre-processed Feedback
Similarities
Pre-processed contents
Filter
Filtered Data
TF-IDF Vectorizer
Vectors
Similarity Calculator (Cosine)

SOFTWARE REQUIREMENTS

1.NumPy:

- Used for numerical computing and handling large arrays of data efficiently.
- Provides support for mathematical operations on arrays, such as matrix multiplication and element-wise operations.
- Essential for data preprocessing and manipulation tasks in machine learning.

2.Pandas:

- Enables data manipulation and analysis through its DataFrame and Series data structures.
- Facilitates tasks like data cleaning, filtering, aggregation, and transformation.
- Useful for handling structured data, such as user preferences and travel history, in the recommendation system.

3.Seaborn:

- Offers high-level functions for creating informative and visually appealing statistical plots.
- Simplifies the process of visualizing data distributions, relationships, and patterns.
- Enhances the interpretability of data analysis results and model insights through effective visualization.

4.Matplotlib:

- Provides a wide range of plotting functions for creating static, interactive, and publication-quality visualizations.
- Supports customization options for controlling plot appearance, layout, and labeling.
- Enables the visualization of data trends, correlations, and recommendations generated by the system.

5.scikit-learn (sklearn):

- Offers a comprehensive suite of machine learning algorithms and tools for building recommendation models.
- Includes modules for preprocessing data, feature selection, model evaluation, and hyperparameter tuning.
- Supports various algorithms such as collaborative filtering, content-based filtering, and hybrid models for personalized recommendations.

6.Jupyter Notebook:

- Provides an interactive environment for data exploration, model prototyping, and sharing research findings.
- Facilitates the creation of executable documents containing code, visualizations, and explanatory text.
- Enables iterative development and collaboration among team members during the project lifecycle.

7.Git:

- Enables version control and collaboration among developers working on the project.
- Tracks changes to the codebase, facilitates code review, and ensures version history management.
- Integrates with platforms like GitHub or GitLab for hosting repositories and managing project workflows.

8.Keras:

- Simplifies the implementation of deep learning models, including neural collaborative filtering, using high-level APIs.
- Allows for rapid prototyping and experimentation with different architectures and hyperparameters.
- Integrates seamlessly with TensorFlow, enabling efficient training and deployment of recommendation models.

9.JOBLIB:

Joblib is a Python library that provides utilities for saving and loading Python objects, especially large NumPy arrays, efficiently to and from disk. It is particularly useful for saving and loading trained machine learning models, which often involve large parameter sets or complex data structures.In this project, after comparing accuracy the SVM got highest accuracy so that model is saved using joblib.

10.STREAMLIT:

Streamlit is an open-source Python library that allows you to create interactive web applications for data science and machine learning projects with ease. It is designed to simplify the process of building and sharing data-centric web apps, enabling data scientists and machine learning engineers to quickly prototype and deploy their projects without requiring web development expertise. In this project, streamlit is used to design the web application.

RECOMMENDATION SYSTEM

A recommendation system is a software application or algorithm that suggests items to users based on their preferences, interests, or past behavior. These systems are widely used in various domains such as e-commerce, streaming services, social media, and online content platforms to help users discover relevant items and enhance their overall experience.

A machine learning algorithm known as a recommendation system combines information about users and products to forecast a user's potential interests. These systems are used in a wide range of applications, such as e-commerce, social media, and entertainment, to provide personalized recommendations to users.

There are several types of recommendation systems, including:

1. **Content-based filtering:** This type of system uses the characteristics of items that a user has liked in the past to recommend similar items.
2. **Collaborative filtering:** This type of system uses the past behaviour of users to recommend items that similar users have liked.
3. **Hybrid:** To generate suggestions, this kind of system combines content-based filtering and collaborative filtering techniques.
4. **Matrix Factorization:** Using this method, the user-item matrix is divided into two lower-dimension matrices that are then utilized to generate predictions.
5. **Deep Learning:** To train the user and item representations that are subsequently utilized to generate recommendations, these models make use of neural networks.

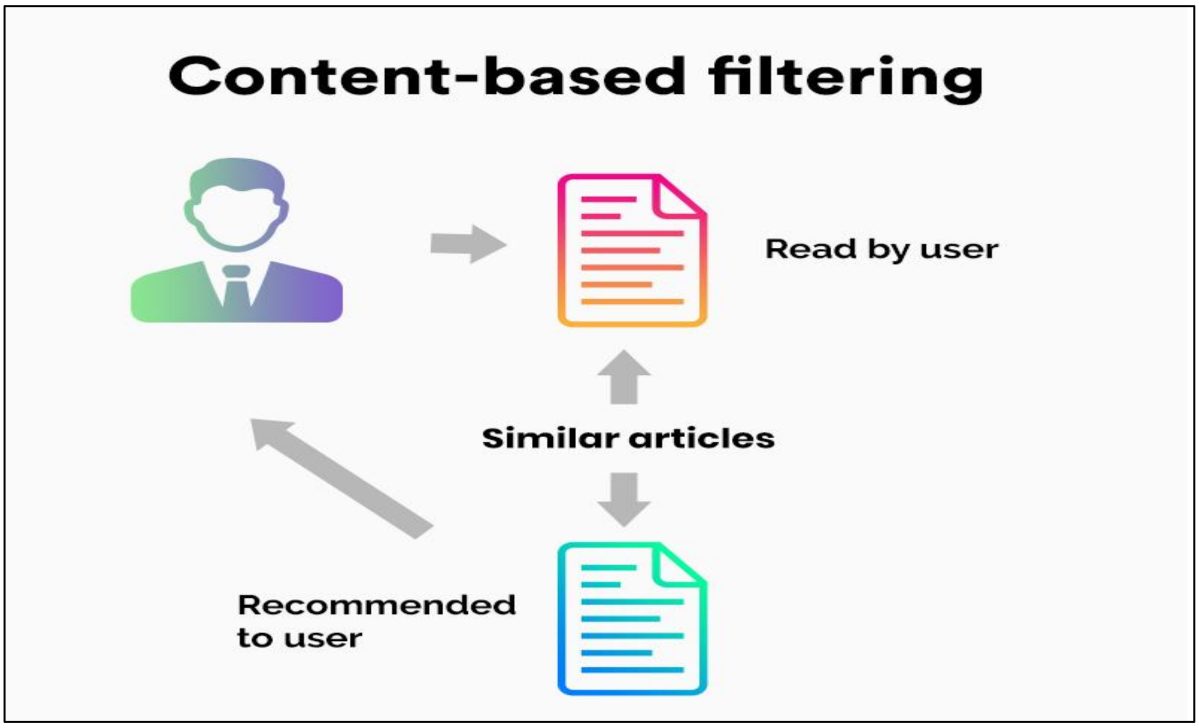
The choice of which type of recommendation system to use depends on the specific application and the type of data available.

Python Recommendation Systems employs a data-driven methodology to offer customers tailored recommendations. It uses user data and algorithms to forecast and suggest goods, services, or content that a user is probably going to find interesting. These systems are essential in applications where users may become overwhelmed by large volumes of information, such as social media, streaming services, and e-commerce. Building recommendation systems is a common use for Python because of its modules and machine learning frameworks.

CLASSIFICATION OF RECOMMENDER SYSTEM

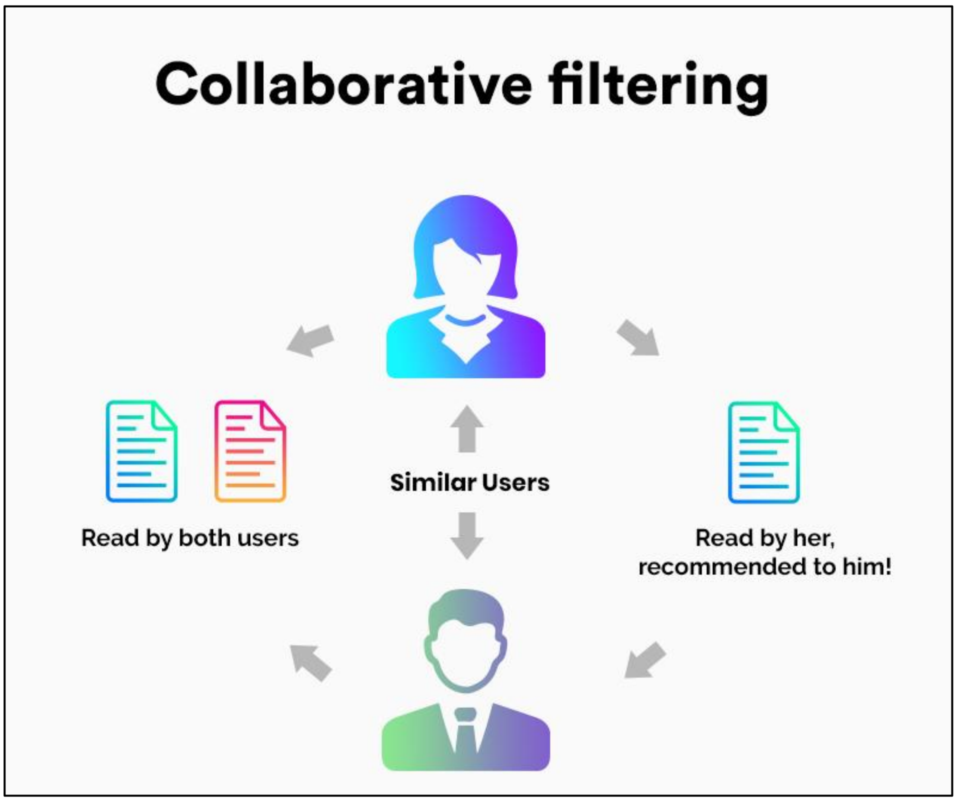
1.CONTENT BASED :

- These systems analyze the content or characteristics of items that the user has interacted with in the past.
- Recommendations are based on similarities between the characteristics of items and the user's profile.
- Recommendations focus on items that are similar to those the user has shown interest in before.
- One challenge of content-based systems is overspecialization, where recommendations become too similar and may not cater to diverse user interests.
- Another challenge is the limitation to textual information, which may overlook contextual, visual, or semantic information that could improve recommendations.



2.COLLOBORATIVE :

- Collaborative filtering systems recommend items based on similarities between users' preferences.
- These systems do not rely on the content of items but rather on the behavior and preferences of users.
- Recommendations are based on the preferences of users with similar interests or behavior.
- Collaborative filtering algorithms can be memory-based (user-based or item-based) or model-based.
- Challenges with collaborative systems include sparsity (lack of user-item interactions), scalability, and the "cold start" problem for new items.



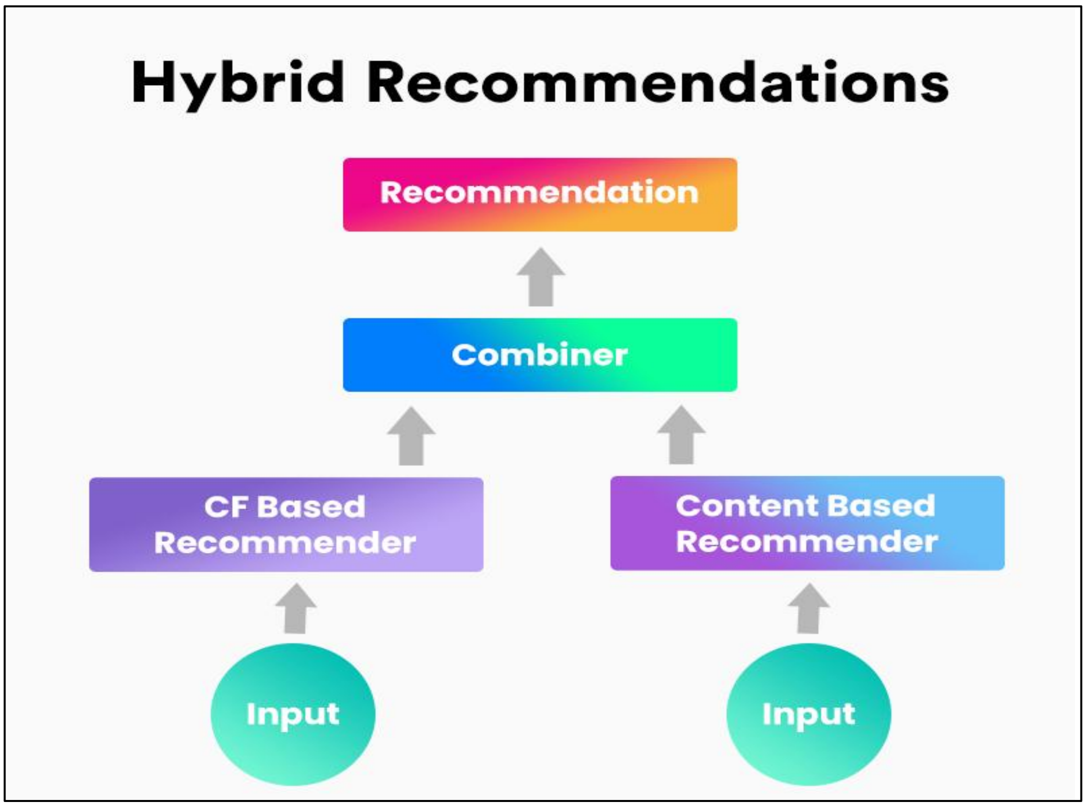
3.Hybrid Recommender Systems:

Hybrid recommender systems integrate two or more recommendation techniques to provide more accurate and diverse recommendations. By combining the strengths of different approaches, hybrid systems aim to overcome the limitations of individual methods and enhance the overall recommendation quality. There are several common hybrid approaches:

Weighted Hybrid: This approach assigns weights to recommendations generated by different techniques and combines them to produce a final recommendation list. The weights can be static or dynamically adjusted based on user feedback or performance metrics.

Switching Hybrid: In this approach, different recommendation techniques are applied based on certain conditions or user characteristics. For example, content-based recommendations may be used when sufficient user data is available, while collaborative filtering may be employed for new users.

Feature Combination: Hybrid systems can combine features extracted from various recommendation techniques to enhance the recommendation process. For instance, content-based features such as item attributes can be combined with collaborative filtering features like user-item interactions to create a more comprehensive user profile.



COUNT VECTORIZER:

CountVectorizer is a feature extraction technique used in natural language processing (NLP) and text mining tasks. It is primarily used to convert a collection of text documents into a matrix of token counts.

Tokenization:

First, the text data is tokenized, meaning it's divided into individual words or terms. This process involves breaking down the text into smaller units, typically words or phrases, called tokens.

Vocabulary Building:

CountVectorizer then builds a vocabulary of all unique tokens (words) present in the entire corpus of documents. Each unique token becomes a feature in the vocabulary.

Counting Occurrences:

For each document in the corpus, CountVectorizer counts the occurrences of each token in the document. This results in a matrix representation where each row corresponds to a document and each column corresponds to a token in the vocabulary. The matrix stores the count of each token in each document.

Vectorization:

Finally, the matrix is created where each row represents a document, and each column represents the frequency of a token in that document. This matrix is typically sparse, meaning most of its entries are zeros since not all tokens appear in all documents.

CountVectorizer has several parameters that can be customized based on the specific requirements of the task:

Tokenization Options: It allows specifying options for tokenization such as handling of punctuation, case sensitivity, n-grams (sequences of adjacent words), etc.

Vocabulary Size: You can limit the size of the vocabulary by specifying parameters like max_features, which only keeps the most frequent tokens.

Stop Words: CountVectorizer can optionally filter out common stop words (like "and", "the", "is") which may not be informative for many tasks.

Token Patterns: It allows specifying regular expressions to define what constitutes a token. This can be useful for handling special cases like numbers, URLs, or custom patterns.

COSINE SIMILARITY

Cosine similarity is a metric used to measure the similarity between two non-zero vectors in an inner product space. In the context of natural language processing (NLP) and information retrieval, cosine similarity is often used to assess the similarity between documents or text-based features.

Here's how cosine similarity works:

Vector Representation: First, each document or text is represented as a vector in a high-dimensional space, where each dimension corresponds to a feature or term. Typically, these vectors are constructed based on some feature extraction method like CountVectorizer or TF-IDF (Term Frequency-Inverse Document Frequency).

Normalization: Before calculating the cosine similarity, it's common practice to normalize the vectors to unit length. This normalization step ensures that the similarity measure is unaffected by the length of the vectors, focusing solely on the direction of the vectors.

Calculation: Cosine similarity is calculated as the cosine of the angle between the two vectors. Mathematically, it's defined as the dot product of the two vectors divided by the product of their magnitudes (or lengths). The formula for cosine similarity between vectors

$$\text{Cosine\_similarity}(a,b) = a.b/(||a||.||b||)$$

Interpretation: Cosine similarity ranges from -1 to 1. A cosine similarity of 1 indicates that the vectors point in the same direction, meaning they are perfectly similar. A cosine similarity of -1 indicates exactly opposite directions, meaning they are dissimilar. A cosine similarity of 0 indicates orthogonality, implying no similarity.

Cosine similarity is widely used in various NLP tasks such as document similarity, information retrieval, clustering, and recommendation systems. It's computationally efficient and provides a robust measure of similarity that is particularly useful when dealing with high-dimensional sparse data like text documents.

SPARSE MATRIX AND DENSE MATRIX

In the context of linear algebra and data representation, sparse and dense matrices are two types of matrices that differ in terms of the density of their elements.

Sparse Matrix:

A sparse matrix is a matrix in which most of the elements are zero. In other words, it has a low ratio of non-zero elements to total elements.

Sparse matrices are common in situations where the data is inherently sparse or when dealing with high-dimensional data where most elements are zero.

Examples of situations where sparse matrices arise include representing adjacency matrices of large graphs, term-document matrices in natural language processing, and representing certain types of data in scientific computing.

Sparse matrices are typically stored in a format optimized for memory efficiency, such as the Compressed Sparse Row (CSR) format, Compressed Sparse Column (CSC) format, or Coordinate List (COO) format.

Dense Matrix:

A dense matrix is a matrix in which most of the elements are non-zero. In other words, it has a high ratio of non-zero elements to total elements.

Dense matrices are common in situations where the data is dense or when dealing with low-dimensional data where most elements are non-zero.

Examples of situations where dense matrices arise include representing small graphs, small datasets in machine learning, and performing basic linear algebra operations.

Dense matrices are typically stored in a straightforward manner, where each element is explicitly stored in memory as a separate value.

In summary, sparse matrices have a low density of non-zero elements and are often used to represent large, sparse datasets efficiently, whereas dense matrices have a high density of non-zero elements and are typically used for smaller, dense datasets or computations. The choice between using sparse or dense matrices depends on the specific characteristics of the data and the computational tasks being performed.

RMSE(ROOT MEAN SQUARED ERROR)

RMSE stands for Root Mean Square Error, and it's a commonly used metric to evaluate the performance of recommendation systems, particularly in the context of rating predictions. RMSE measures the average deviation of predicted ratings from the actual ratings provided by users.

Here's how RMSE is calculated in the context of a recommendation system:

Prediction: The recommendation system predicts ratings for items (e.g., movies, products) that a user hasn't rated based on the user's historical ratings and other factors such as item features, user demographics, or collaborative filtering.

Actual Ratings: For each user, there are actual ratings available for some items in the dataset. These ratings are typically provided by users in the form of explicit ratings (e.g., star ratings) or implicit feedback (e.g., purchase history, clicks).

Calculation: RMSE is calculated by taking the square root of the average of the squared differences between predicted ratings and actual ratings. Mathematically, it can be expressed as:

$$RMSE = \sqrt{\frac{1}{n}\sum_{j=1}^{n}(y_j - \hat{y}_j)^2}$$

Interpretation: RMSE provides a measure of the average error between predicted ratings and actual ratings. A lower RMSE indicates better performance, as it means the predicted ratings are closer to the actual ratings. Conversely, a higher RMSE indicates poorer performance, suggesting larger discrepancies between predicted and actual ratings.

RMSE is widely used in recommendation systems because it provides a quantitative measure of prediction accuracy, making it easy to compare the performance of different recommendation algorithms or parameter settings. However, it's important to note that RMSE may not capture all aspects of recommendation quality, especially in cases where user preferences are diverse or when dealing with implicit feedback data. Therefore, it's often used in conjunction with other metrics such as precision, recall, or diversity to provide a more comprehensive evaluation of recommendation system performance.

ENCODING IN COLLABORATIVE

In collaborative filtering, encoding refers to the process of representing users, items (such as products or movies), or their interactions in a numerical format suitable for processing by machine learning algorithms. This encoding is necessary to convert raw data, such as user-item interaction histories, into a format that can be used to train recommendation models.

There are several approaches to encoding in collaborative filtering:

One-Hot Encoding: One common method is one-hot encoding, where categorical variables such as user IDs or item IDs are converted into binary vectors. Each dimension of the vector represents a unique user or item, and only one dimension is set to 1 while all others are set to 0 to indicate the presence of a specific user or item.

Embedding: Another approach is embedding, where categorical variables are mapped to continuous vectors of a lower dimensionality. This mapping is learned during the training process, allowing the model to capture the latent relationships between users and items. Embeddings are typically initialized randomly and updated iteratively through backpropagation during training.

Sparse Matrix Representation: In some cases, interactions between users and items are represented as a sparse matrix, where rows correspond to users, columns correspond to items, and each entry represents the interaction strength (e.g., rating, purchase, click). This matrix can then be processed directly by collaborative filtering algorithms or transformed into a more compact representation using techniques like matrix factorization.

The choice of encoding method depends on factors such as the nature of the data, the size of the dataset, and the requirements of the recommendation model. Regardless of the specific approach, the goal of encoding in collaborative filtering is to represent user-item interactions in a format that facilitates the learning of meaningful patterns and relationships by the recommendation model.

SOURCE CODE

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
tourism=pd.read_csv('tourism.csv')
rating=pd.read_csv('rating.csv')
user=pd.read_csv('user.csv')
tourism.head()
rating.head()
user.head()
num_places = len(tourism['Place_Id'].unique())
print("Number of places in the dataset:", num_places)

num_users = len(user['User_Id'].unique())
print("Number of users:", num_users)

num_ratings = len(rating)
print("Total number of ratings:", num_ratings)
tourism.shape
rating.shape
user.shape
tourism.info()
rating.info()
user.info()
tourism.isnull().sum()
rating.isnull().sum()
user.isnull().sum()
tourism_all = np.union1d(tourism['Place_Id'].unique(), rating['Place_Id'].unique())
total_tourism = len(tourism_all)
print("Total number of tourism:",total_tourism)
all_tourism_rate = rating.copy()
all_tourism_rate
all_tourism                                                                     =
pd.merge(all_tourism_rate,tourism[["Place_Id","Place_Name","Description","City","Categor
y"]],on='Place_Id', how='left')
all_tourism
all_tourism['city_category'] = all_tourism[['City','Category']].agg(' '.join,axis=1)
all_tourism
all_tourism.describe()
all_tourism.isnull().sum()
prep = all_tourism.drop_duplicates(subset='Place_Id')
```

```
Prep=place_id, place_name, place_category, place_desc, place_city, city_category = prep[['Place_Id','Place_Name','Category', 'Description', 'City', 'city_category']].values.T.tolist()
tourism_new = pd.DataFrame({
   "id": prep['Place_Id'],
   "name": prep['Place_Name'],
   "category": prep['Category'],
   "description": prep['Description'],
   "city": prep['City'],
   "city_category": prep['city_category']
})
tourism_new
op_10 = tourism_new['id'].value_counts().reset_index()[0:10]
top_10 = pd.merge(top_10,prep[['Place_Id','Place_Name']], how='left', left_on='index', right_on='Place_Id')
plt.figure(figsize=(8,5))
sns.barplot('Place_Id', 'Place_Name', data=top_10)
plt.title('Creating a visualization of tourism with the highest number of ratings.'", pad=20)
plt.ylabel("The number of ratings.")
plt.xlabel("The location name.")
plt.show()
sns.countplot(y='Category', data=prep)
plt.title('Comparison of the Number of Tourism Categories in Bandung City', pad=20)
plt.show()
plt.figure(figsize=(5,3))
sns.boxplot(user['Age']);
plt.title('Distribution of User Ages', pad=20)
plt.show()
City_of_Origin= user['Location'].apply(lambda x : x.split(',')[0])
plt.figure(figsize=(8,6))
sns.countplot(y=City_of_Origin)
plt.title('The Number of Cities of Origin of Users')
plt.show()
ds = tourism_new
ds.sample(5)
from sklearn.feature_extraction.text import CountVectorizer
cv = CountVectorizer()
cv.fit(ds['city_category'])
print("Features Name: ", list(cv.vocabulary_.keys()))
cv_matrix = cv.transform(ds['city_category'])
cv_matrix.shape
cv_matrix.todense()
df = pd.DataFrame(
   cv_matrix.todense(),
   columns=list(cv.vocabulary_.keys()),index=ds['name'])
```

```
    df.sample(5)
from sklearn.metrics.pairwise import cosine_similarity
cosine = cosine_similarity(cv_matrix)
cosine
cosine_sim = pd.DataFrame(cosine,index=ds['name'],columns=ds['name'])
cosine_sim.sample(5,axis=1).sample(10,axis=0)
def
tourism_recommendations(place_name,similarity_data=cosine_sim,items=ds[['name','categor
y','description','city']],k=5):
    index = similarity_data.loc[:,place_name].to_numpy().argpartition(range(-1,-k,-1))

    closest = similarity_data.columns[index[-1:-(k+2):-1]]

    closest = closest.drop(place_name,errors='ignore')

    return pd.DataFrame(closest).merge(items).head(k)
tourism_recommendations("SnowBay Waterpark")
import pandas as pd
import numpy as np
from zipfile import ZipFile
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from pathlib import Path
df = rating
df
user_ids = df.User_Id.unique().tolist()

user_to_user_encoded = {x:i for i, x in enumerate(user_ids)}

user_encoded_to_user = {i: x for i, x in enumerate(user_ids)}
place_ids = df.Place_Id.unique().tolist()

place_to_place_encoded = {x: i for i, x in enumerate(place_ids)}

place_encoded_to_place = {x: i for x, i in enumerate(place_ids)}
df['user'] = df.User_Id.map(user_to_user_encoded)

df['place'] = df.Place_Id.map(place_to_place_encoded)
num_users = len(user_to_user_encoded)
num_place = len(place_encoded_to_place)
df['Place_Ratings'] = df['Place_Ratings'].values.astype(np.float32)
min_rating = min(df['Place_Ratings'])
max_rating= max(df['Place_Ratings'])
```

```
print('Number of User: {}, Number of Place: {}, Min Rating: {}, Max Rating: {}'.format(
    num_users, num_place, min_rating, max_rating
))
df = df.sample(frac=1,random_state=42)
df
x = df[['user','place']].values
y = df['Place_Ratings'].apply(lambda x:(x-min_rating)/(max_rating-min_rating)).values
train_indices = int(0.8 * df.shape[0])
x_train,x_val,y_train,y_val = (
    x[:train_indices],
    x[train_indices:],
    y[:train_indices],
    y[train_indices:]
)
print(x,y)
class RecommenderNet(tf.keras.Model):
  def __init__(self, num_users, num_place, embedding_size, **kwargs):
    super(RecommenderNet, self).__init__(**kwargs)
    self.num_users = num_users
    self.num_place = num_place
    self.embedding_size = embedding_size
    self.user_embedding = layers.Embedding(
        num_users,
        embedding_size,
        embeddings_initializer = 'he_normal',
        embeddings_regularizer = keras.regularizers.l2(1e-6)
    )
    self.user_bias = layers.Embedding(num_users, 1)
    self.place_embedding = layers.Embedding(
        num_place,
        embedding_size,
        embeddings_initializer = 'he_normal',
        embeddings_regularizer = keras.regularizers.l2(1e-6)
    )
    self.place_bias = layers.Embedding(num_place, 1)

  def call(self, inputs):
    user_vector = self.user_embedding(inputs[:,0])
    user_bias = self.user_bias(inputs[:, 0])
    place_vector = self.place_embedding(inputs[:, 1])
    place_bias = self.place_bias(inputs[:, 1])

    dot_user_place = tf.tensordot(user_vector, place_vector, 2)
    x = dot_user_place + user_bias + place_bias

    return tf.nn.sigmoid(x)

 model = RecommenderNet(num_users, num_place, 100)
model.compile(
    loss = tf.keras.losses.BinaryCrossentropy(),
    optimizer = keras.optimizers.Adam(learning_rate=0.001),
    metrics=[tf.keras.metrics.RootMeanSquaredError()]
)
history = model.fit(
    x = x_train,
    y = y_train,
    batch_size = 8,
    epochs = 100,
    validation_data = (x_val, y_val),
)
plt.plot(history.history['root_mean_squared_error'])
plt.plot(history.history['val_root_mean_squared_error'])
plt.title('model_metrics')
plt.ylabel('root_mean_squared_error')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
place_df = tourism_new
df = pd.read_csv('rating.csv')

user_id = df.User_Id.sample(1).iloc[0]
place_visited_by_user = df[df.User_Id == user_id]

place_not_visited                                                           =
place_df[~place_df['id'].isin(place_visited_by_user['Place_Id'].values)]['id']
place_not_visited = list(
    set(place_not_visited)
    .intersection(set(place_to_place_encoded.keys()))
)

place_not_visited = [[place_to_place_encoded.get(x)] for x in place_not_visited]
user_encoder = user_to_user_encoded.get(user_id)
user_place_array = np.hstack(
    ([[user_encoder]] * len(place_not_visited), place_not_visited)
)
ratings = model.predict(user_place_array).flatten()
```

```
top_ratings_indices = ratings.argsort()[-10:][::-1]
recommended_place_ids = [
    place_encoded_to_place.get(place_not_visited[x][0]) for x in top_ratings_indices
]

print('Showing recommendations for users: {}'.format(user_id))

print('Place with high ratings from user :')


top_place_user = (
    place_visited_by_user.sort_values(
        by = 'Place_Ratings',
        ascending=False
    )
    .head(5)
    .Place_Id.values
)

place_df_rows = place_df[place_df['id'].isin(top_place_user)]
pd.DataFrame(place_df_rows)
print('Top 10 place recommendation')
recommended= place_df[place_df['id'].isin(recommended_place_ids)]
recommended
import joblib
joblib.dump(cosine_sim,'COSINE_SIMILARITY.pkl')
items=ds[['name','category','description','city']]
joblib.dump(items,'DATA_ITEMS.pkl')
import joblib
import pandas as pd
cosine=joblib.load('COSINE_SIMILARITY.pkl')
item=joblib.load('DATA_ITEMS.pkl')
def tourism_recommendations(place_name):
    try:
        k=5
        similarity_data=cosine
        items=item
        index = similarity_data.loc[:,place_name].to_numpy().argpartition(range(-1,-k,-1))
        closest = similarity_data.columns[index[-1:-(k+2):-1]]
        closest = closest.drop(place_name,errors='ignore')
        df=pd.DataFrame(closest).merge(items).head(k)
        return df
    except KeyError:
        return "Sorry!No data found"
```

```
input_data=input("Enter the location:")
print(input_data)
tourism_recommendations(input_data)

import streamlit as st
import pandas as pd
import joblib

# Load precomputed data
cosine_sim = joblib.load('COSINE_SIMILARITY.pkl')
ds = joblib.load('DATA_ITEMS.pkl')

# Function to recommend places
def                           tourism_recommendations(place_name,similarity_data=cosine_sim,
items=ds[['name','category','description','city']], k=5):
    index = similarity_data.loc[:, place_name].to_numpy().argsort()[::-1][:k]
    closest = similarity_data.columns[index[-1:-(k+2):-1]]
    closest = closest.drop(place_name, errors='ignore')
    return pd.DataFrame(closest).merge(items).head(k)

# Streamlit app
def main():
    st.title('Travel Recommendation System')
    st.sidebar.title('Explore Your Next Destination')

    # Input field for location
    place = st.sidebar.text_input("Enter your location here")

    # Button to generate recommendations
    recommend_button = st.sidebar.button("Generate Recommendations")

    # Display recommendations if button clicked
    if recommend_button and place:
        recommendations = tourism_recommendations(place)
        if recommendations is not None and not recommendations.empty:
            st.subheader("Top Recommendations")
            st.dataframe(recommendations.style.highlight_max(axis=0))
        else:
            st.error("Sorry! No recommendations found.")

if __name__ == '__main__':
    main()
```

EXPLANATION

Content based

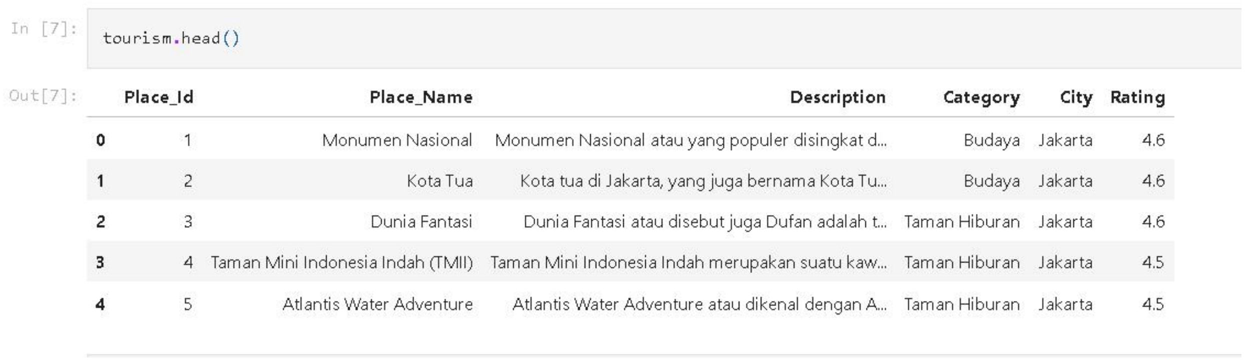
The tourism_recommendations function is designed to provide recommendations for tourism destinations based on similarity data. Taking the name of a specific place as input, the function utilizes similarity data, often in the form of cosine similarity scores, to identify destinations that exhibit similar characteristics or appeal. It first retrieves the similarity scores for the specified place from the similarity data, efficiently identifying the k most similar destinations using a partitioning technique. Next, it extracts the names of these similar destinations, excluding the input place to avoid recommending it to itself. Subsequently, the function merges this list of similar destinations with additional information about tourism destinations, such as name, category, description, and city, stored in the items DataFrame. Finally, it returns a DataFrame containing the top k recommended destinations based on their similarity to the input place. This function is valuable for assisting tourists in discovering new destinations that align with their interests or preferences, enhancing their travel experiences by offering personalized recommendations.

Collaborative based

The RecommenderNet class defines a neural network-based recommendation model using TensorFlow/Keras. Upon initialization, it sets up parameters such as the number of users, number of places, and embedding size. The class includes embedding layers for both users and places, which are initialized with He normal initializer and regularized using L2 regularization to prevent overfitting. In the call method, the model takes inputs containing user and place IDs and extracts corresponding embeddings and biases. It then computes the dot product between user and place embeddings, adds user and place biases, and applies the sigmoid activation function to generate the final prediction. This model architecture enables the system to learn the latent representations of users and places based on their interactions, thereby making recommendations by predicting the likelihood of user engagement with different places. Overall, the RecommenderNet class encapsulates a neural network model tailored for recommendation tasks, providing a flexible and customizable framework for building recommendation systems

SCREENSHOTS

Tourism dataset

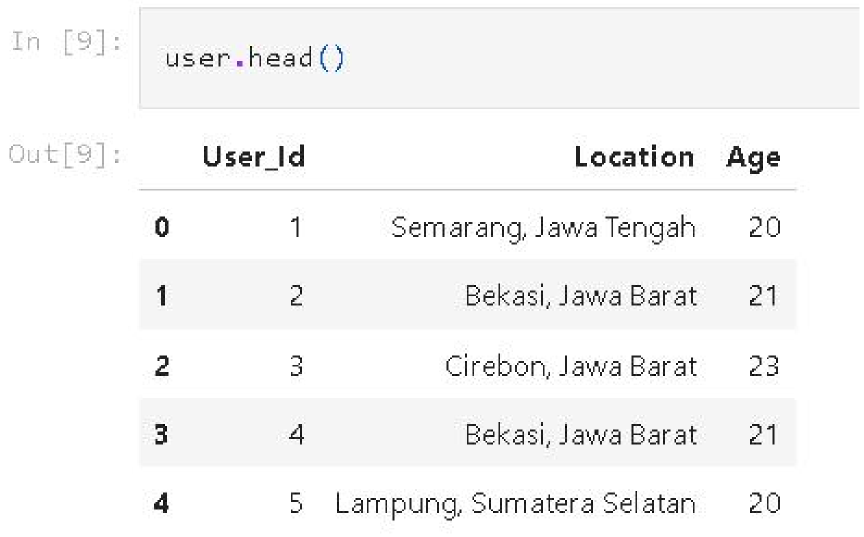


```
In [7]: tourism.head()
```

Out[7]:

	Place_Id	Place_Name	Description	Category	City	Rating
0	1	Monumen Nasional	Monumen Nasional atau yang populer disingkat d...	Budaya	Jakarta	4.6
1	2	Kota Tua	Kota tua di Jakarta, yang juga bernama Kota Tu...	Budaya	Jakarta	4.6
2	3	Dunia Fantasi	Dunia Fantasi atau disebut juga Dufan adalah t...	Taman Hiburan	Jakarta	4.6
3	4	Taman Mini Indonesia Indah (TMII)	Taman Mini Indonesia Indah merupakan suatu kaw...	Taman Hiburan	Jakarta	4.5
4	5	Atlantis Water Adventure	Atlantis Water Adventure atau dikenal dengan A...	Taman Hiburan	Jakarta	4.5

User dataset

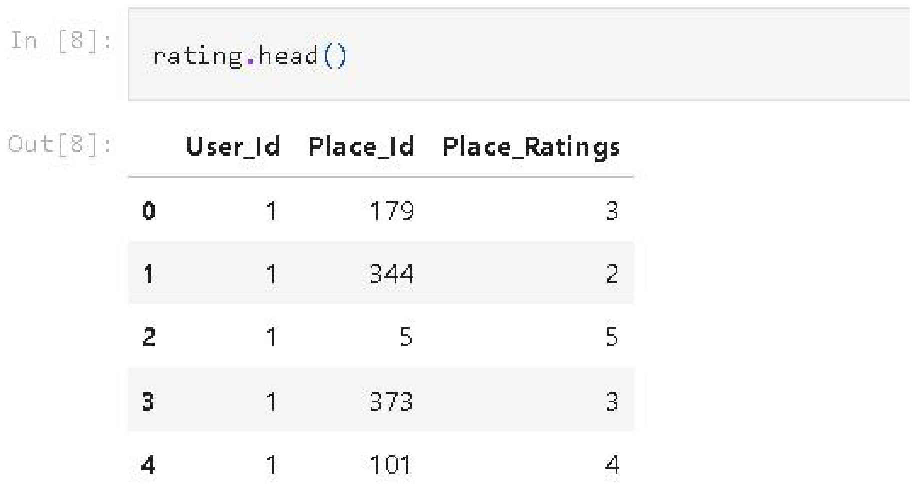


```
In [9]: user.head()
```

Out[9]:

	User_Id	Location	Age
0	1	Semarang, Jawa Tengah	20
1	2	Bekasi, Jawa Barat	21
2	3	Cirebon, Jawa Barat	23
3	4	Bekasi, Jawa Barat	21
4	5	Lampung, Sumatera Selatan	20

Rating dataset

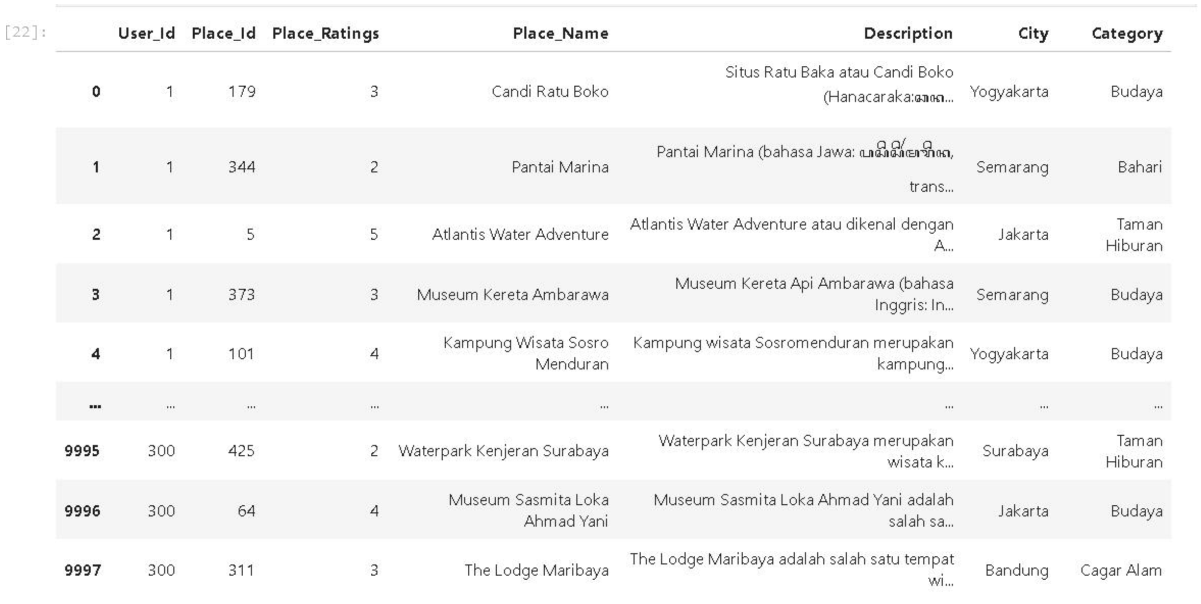


```
In [8]: rating.head()
```

Out[8]:

	User_Id	Place_Id	Place_Ratings
0	1	179	3
1	1	344	2
2	1	5	5
3	1	373	3
4	1	101	4

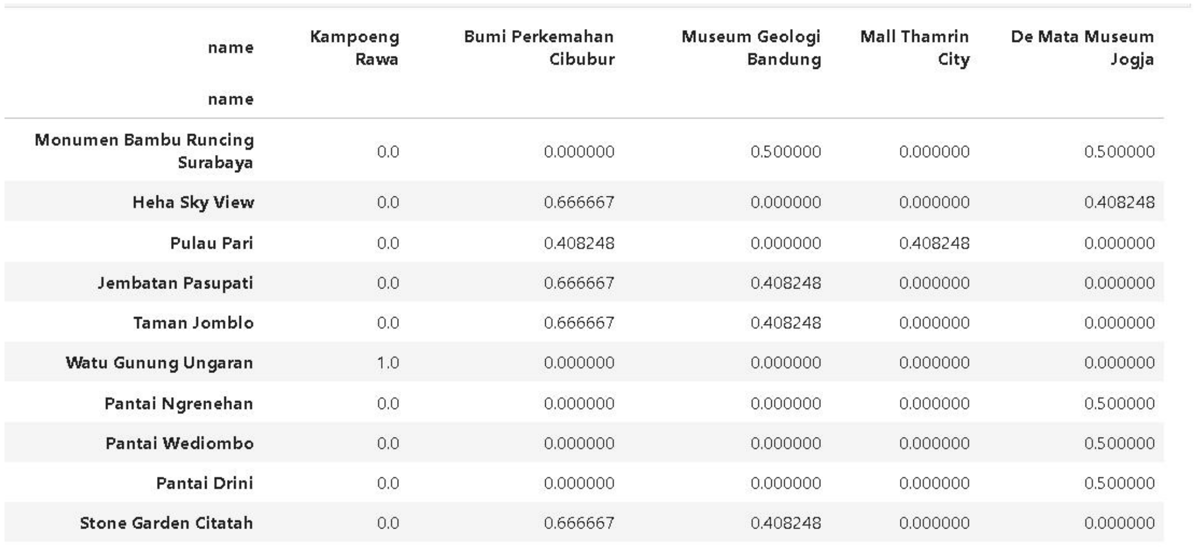
After merging



[22]:

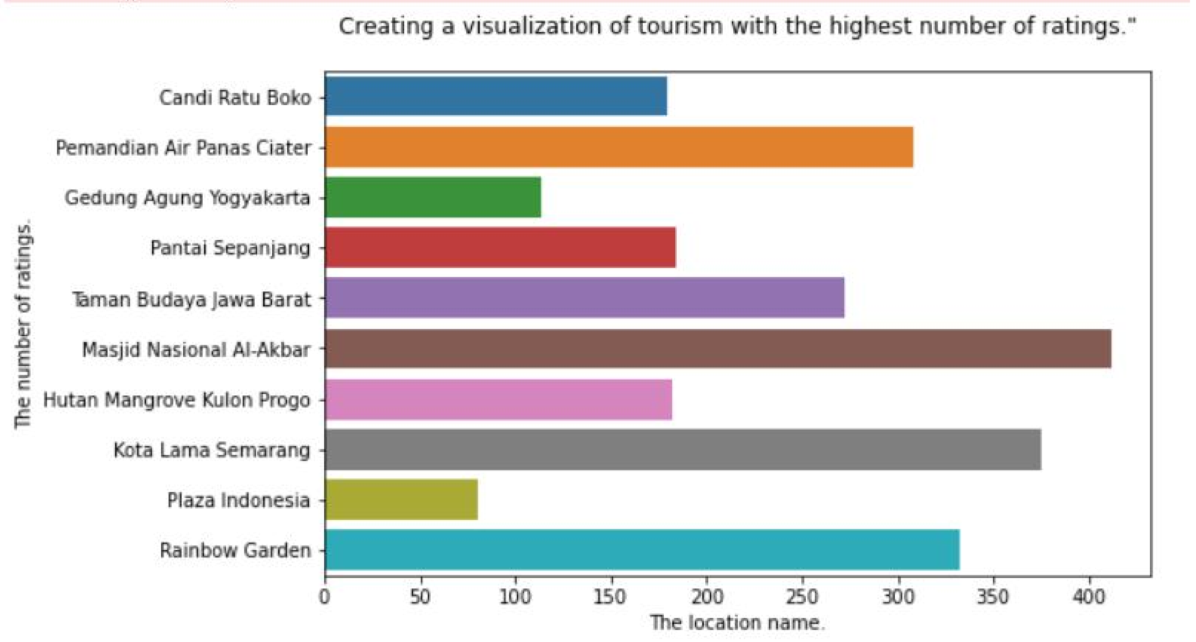
	User_Id	Place_Id	Place_Ratings	Place_Name	Description	City	Category
0	1	179	3	Candi Ratu Boko	Situs Ratu Baka atau Candi Boko (Hanacaraka:[illegible]...	Yogyakarta	Budaya
1	1	344	2	Pantai Marina	Pantai Marina (bahasa Jawa: [illegible], trans...	Semarang	Bahari
2	1	5	5	Atlantis Water Adventure	Atlantis Water Adventure atau dikenal dengan A...	Jakarta	Taman Hiburan
3	1	373	3	Museum Kereta Ambarawa	Museum Kereta Api Ambarawa (bahasa Inggris: In...	Semarang	Budaya
4	1	101	4	Kampung Wisata Sosro Menduran	Kampung wisata Sosromenduran merupakan kampung...	Yogyakarta	Budaya
...	...	...	...	...	...	...	...
9995	300	425	2	Waterpark Kenjeran Surabaya	Waterpark Kenjeran Surabaya merupakan wisata k...	Surabaya	Taman Hiburan
9996	300	64	4	Museum Sasmita Loka Ahmad Yani	Museum Sasmita Loka Ahmad Yani adalah salah sa...	Jakarta	Budaya
9997	300	311	3	The Lodge Maribaya	The Lodge Maribaya adalah salah satu tempat wi...	Bandung	Cagar Alam

Applying cosine similarity

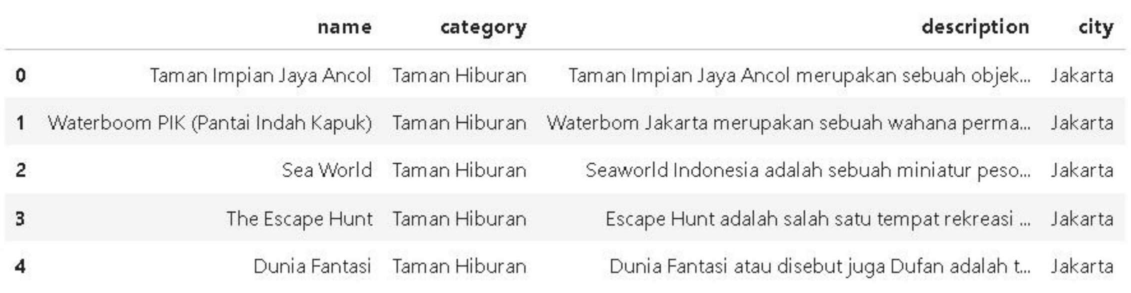


name	Kampoeng Rawa	Bumi Perkemahan Cibubur	Museum Geologi Bandung	Mall Thamrin City	De Mata Museum Jogja
name					
Monumen Bambu Runcing Surabaya	0.0	0.000000	0.500000	0.000000	0.500000
Heha Sky View	0.0	0.666667	0.000000	0.000000	0.408248
Pulau Pari	0.0	0.408248	0.000000	0.408248	0.000000
Jembatan Pasupati	0.0	0.666667	0.408248	0.000000	0.000000
Taman Jomblo	0.0	0.666667	0.408248	0.000000	0.000000
Watu Gunung Ungaran	1.0	0.000000	0.000000	0.000000	0.000000
Pantai Ngrenehan	0.0	0.000000	0.000000	0.000000	0.500000
Pantai Wediombo	0.0	0.000000	0.000000	0.000000	0.500000
Pantai Drini	0.0	0.000000	0.000000	0.000000	0.500000
Stone Garden Citatah	0.0	0.666667	0.408248	0.000000	0.000000

Visualization

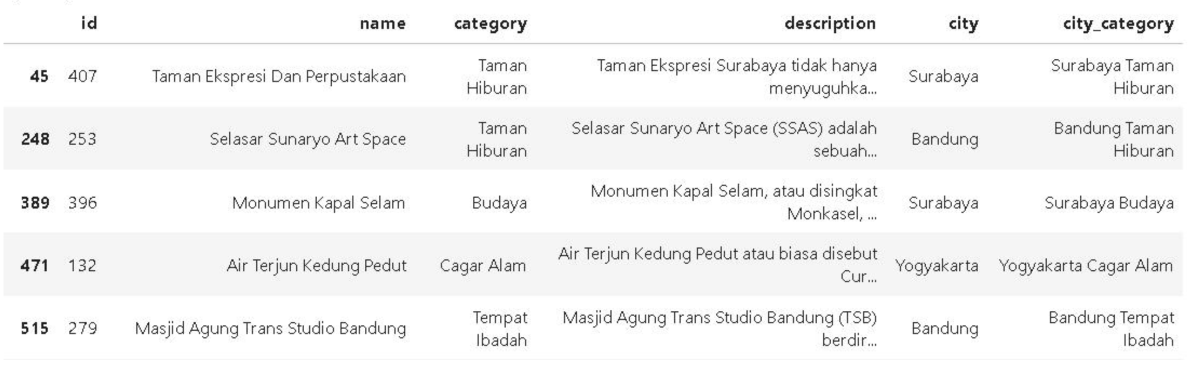


Content based



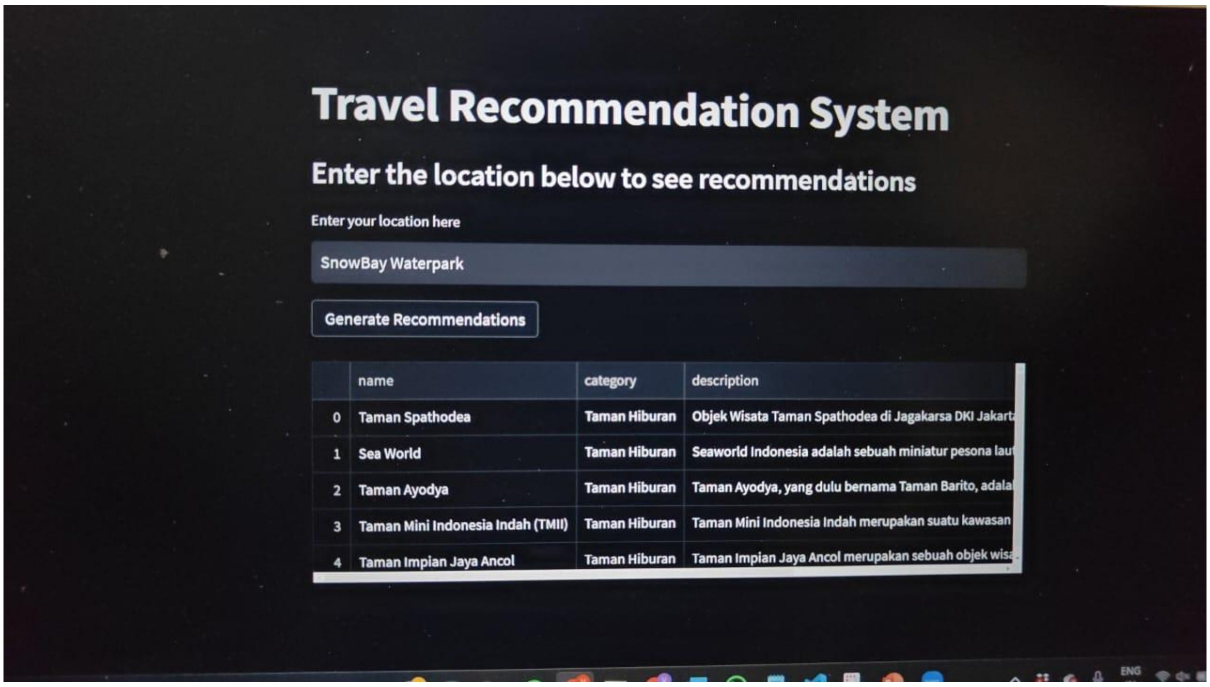
	name	category	description	city
0	Taman Impian Jaya Ancol	Taman Hiburan	Taman Impian Jaya Ancol merupakan sebuah objek...	Jakarta
1	Waterboom PIK (Pantai Indah Kapuk)	Taman Hiburan	Waterbom Jakarta merupakan sebuah wahana perma...	Jakarta
2	Sea World	Taman Hiburan	Seaworld Indonesia adalah sebuah miniatur peso...	Jakarta
3	The Escape Hunt	Taman Hiburan	Escape Hunt adalah salah satu tempat rekreasi ...	Jakarta
4	Dunia Fantasi	Taman Hiburan	Dunia Fantasi atau disebut juga Dufan adalah t...	Jakarta

Collaborative



	id	name	category	description	city	city_category
45	407	Taman Ekspresi Dan Perpustakaan	Taman Hiburan	Taman Ekspresi Surabaya tidak hanya menyuguhka...	Surabaya	Surabaya Taman Hiburan
248	253	Selasar Sunaryo Art Space	Taman Hiburan	Selasar Sunaryo Art Space (SSAS) adalah sebuah...	Bandung	Bandung Taman Hiburan
389	396	Monumen Kapal Selam	Budaya	Monumen Kapal Selam, atau disingkat Monkasel, ...	Surabaya	Surabaya Budaya
471	132	Air Terjun Kedung Pedut	Cagar Alam	Air Terjun Kedung Pedut atau biasa disebut Cur...	Yogyakarta	Yogyakarta Cagar Alam
515	279	Masjid Agung Trans Studio Bandung	Tempat Ibadah	Masjid Agung Trans Studio Bandung (TSB) berdir...	Bandung	Bandung Tempat Ibadah

Web application



Travel Recommendation System

Enter the location below to see recommendations

Enter your location here

SnowBay Waterpark

	name	category	description
0	Taman Spathodea	Taman Hiburan	Objek Wisata Taman Spathodea di Jagakarsa DKI Jakart
1	Sea World	Taman Hiburan	Seaworld Indonesia adalah sebuah miniatur pesona lau
2	Taman Ayodya	Taman Hiburan	Taman Ayodya, yang dulu bernama Taman Barito, adala
3	Taman Mini Indonesia Indah (TMII)	Taman Hiburan	Taman Mini Indonesia Indah merupakan suatu kawasan
4	Taman Impian Jaya Ancol	Taman Hiburan	Taman Impian Jaya Ancol merupakan sebuah objek wis

CONCLUSION

In conclusion, the development of a travel recommendation system leveraging both content-based and collaborative filtering techniques holds significant promise in enhancing the travel experience for users. By combining the strengths of these two approaches, we have created a system that offers personalized recommendations based on both the intrinsic characteristics of travel items and the preferences of similar users. Integrating content-based filtering allows our system to consider the intrinsic attributes of travel destinations and activities. This leads to more personalized recommendations tailored to users' specific interests, preferences, and constraints. Collaborative filtering complements this by leveraging collective user behaviour to identify similarities and patterns, further enhancing recommendation . By incorporating both content-based and collaborative filtering, our system can provide a diverse pool of recommendations. Content-based filtering ensures that recommendations are relevant and aligned with users' stated preferences, while collaborative filtering introduces serendipity by suggesting items liked by similar users but not yet encountered by the current user. Ultimately, the success of our travel recommendation system hinges on user engagement and satisfaction. By providing relevant, diverse, and personalized recommendations, we aim to enhance the overall travel experience for users, encouraging them to explore new destinations, discover hidden gems, and create unforgettable memories.

FUTURE WORK

While the travel recommendation system utilizing content-based and collaborative filtering techniques has shown promising results, there are several avenues for future work and improvement:

1. **Integration of Additional Data Sources:** Incorporating additional data sources such as social media activity, user reviews, and real-time travel data could further enhance recommendation accuracy and relevance. Analyzing user-generated content and sentiment analysis can provide valuable insights into users' preferences and sentiments towards specific destinations or attractions.
2. **Advanced Machine Learning Techniques:** Exploring advanced machine learning techniques such as deep learning and reinforcement learning can lead to more sophisticated recommendation models. Deep learning models can capture complex patterns and relationships in user data, while reinforcement learning can optimize recommendations based on user feedback and interactions over time.
3. **Dynamic and Context-Aware Recommendations:** Developing a system that can adapt to changing user preferences and contextual factors such as location, time of day, and weather conditions can provide more personalized and timely recommendations. Incorporating contextual information into the recommendation process can improve user satisfaction and engagement.
4. **Evaluation and User Feedback:** Conducting thorough evaluations and gathering user feedback are essential for iteratively improving the recommendation system. Implementing A/B testing and user surveys can help assess the effectiveness of different recommendation algorithms and features. Incorporating user feedback into the recommendation process can further refine the system and enhance user satisfaction.
5. **Enhanced Visualization and User Interface:** Improving the visualization and user interface of the recommendation system can enhance user experience and usability. Interactive visualizations, personalized dashboards, and intuitive navigation can make it easier for users to explore recommendations, customize their preferences, and plan their travel itineraries.
6. **Collaboration with Industry Partners:** Collaborating with industry partners such as travel agencies, airlines, and hotels can provide access to additional data sources and domain expertise. Partnering with industry stakeholders can also help validate the effectiveness of the recommendation system in real-world settings and identify opportunities for commercialization and deployment.
7. **Addressing Privacy and Security Concerns:** Ensuring the privacy and security of user data is paramount in developing a recommendation system. Implementing robust data privacy measures such as anonymization, encryption, and access controls can protect user confidentiality and build trust in the system.
8. **Scaling and Deployment:** Optimizing the recommendation system for scalability and deploying it in production environments is essential for real-world applications. tools can streamline the deployment process and ensure seamless performance under high load conditions.

PROJECT LINKS :

P. Venkata Mohan : https://github.com/Venkatmohan07/Travel-Recommender

B. Bhavya : https://github.com/BBhavya02/Travel-System

Pottimurthi Jaswanth : https://github.com/jaswanthlucky/Travel_systems

SUBMITTED BY

P.VENKATA MOHAN

B.BHAVYA

P.JASWANTH