# SUMMER INTERNSHIP REPORT

**Computer Science and Engineering**

**JIIT Noida**

**B TECH – Ist Year**

**(2-Credit Internship)**

**Travel Buddy: Travel Recommendation System**

**Submitted by**

| | |
|---|---|
| Karvy Singh | 2401030234 |
| Tanishka Singh | J9924012100097 |
| Harshit Shukla | 2401200047 |
| Jayant Singh | J9924010400089 |
| Prasoon Sharma | J9924012100051 |

**Internship Duration:** June 9 - July 5, 2025

**Submitted to**

Dr. Neetu Singh

Dr. Imraan Rasheed

Dr. Kirti Aggarwal

## Declaration

I hereby declare that the Summer Internship Report titled “Travel Buddy: Travel Recommendation System” is the outcome of my own work, carried out under the guidance of-
Dr. Neetu Singh
Dr. Imraan Rasheed
Dr. Kirti Aggarwal
and has not been submitted elsewhere for any other academic purpose.

| | |
|---|---|
| Name of Student: | Karvy Singh |
| Signature: | ____________________ |
| Date: | ____________________ |

## Certificate

This is to certify that Karvy Singh, Roll No. 2401030234, has successfully completed the Summer Internship Project titled “Travel Buddy: Travel Recommendation System” during the period June 9 – July 5, 2025 under my supervision.

| | |
|---|---|
| Mentor’s Name: | |
| Designation: | |
| Department: | |
| Organisation: | |
| Signature: | |

## Acknowledgement

I express my sincere gratitude to my mentor-
Dr. Neetu Singh
Dr. Imraan Rasheed
Dr. Kirti Aggarwal
for their invaluable guidance and support throughout this internship. I also thank Jaypee Institute of Information Technology for the opportunity to work on this project, and my family and friends for their encouragement.

# Contents

## 1 Introduction

Travelers often find it overwhelming to search for suitable tourist spots due to the abundance of options available. The diversity in preferences and lack of personalized suggestions further complicates their decision-making process, so for more personalized recommendations, so we made recommendation system that uses IR+ML to give you details of travel destinations fromthe our dataset consisting of 3000 entries, along with some pictures from net.

## 2 Objectives of the Internship

- Understand Core IR Concepts
  - Study and implement ranking and retrieval metrics.
  - Analyze search engine behavior and document relevance.
- Apply Machine Learning to IR Tasks
  - Use supervised models to improve ranking or classification.
  - Train and evaluate models on real-world datasets.
- Explore Feature Engineering and Model Evaluation
  - Extract meaningful features (e.g., TF-IDF).
  - Evaluate models with appropriate metrics (e.g., MAP, NDCG, F1).
- Build End-to-End IR or ML Pipelines
  - Preprocessing, model training, evaluation, and reporting.
  - Automate workflows where possible (e.g., using scikit-learn or PyTorch).
- Gain Practical Experience
  - Work with real datasets (text, tabular, etc.).
  - Contribute to team projects or research-oriented tasks.

## 3 Company/Problem Overview

Sites like MakeMyTrip focus on hotels, prices, journey but we lack sites recommending which place to visit as per your interest, for that we need to use old-school google search, which does not allow you to put in filters for more personalized recommendations, so we made recommendation system that uses IR+ML to give you details of travel destinations from the our dataset, with photos from the net, along with simple sober and yet better interface for interaction.

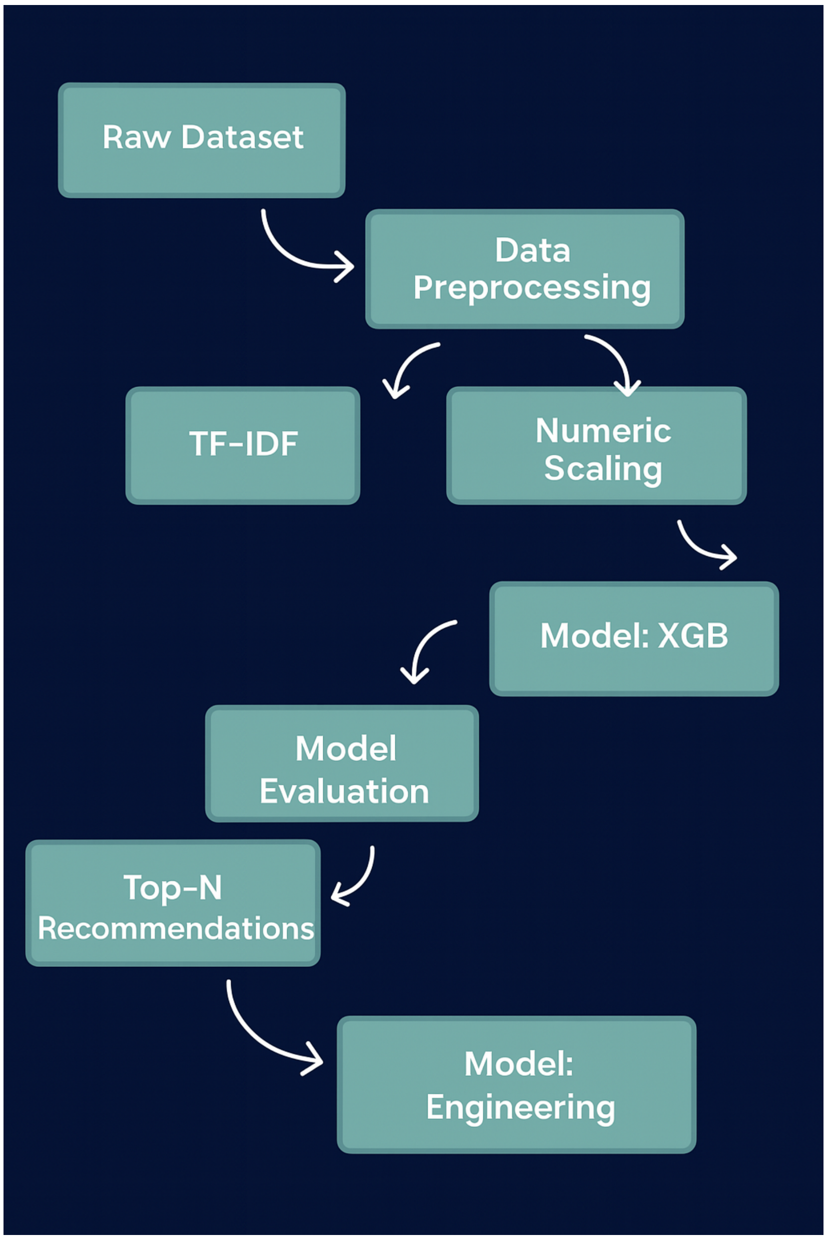
## 4 Tools and Technologies Used

- Neovim editor
- Python
- HTML,CSS,JavaScript
- GitHub for VCS
  - flask==2.3.3
  - numpy==1.26.4
  - pandas==2.2.2
  - scikit-learn==1.4.2
  - scipy==1.13.1
  - xgboost==2.0.3
  - requests==2.32.3
- Deployment app

## 5 Weekly Progress Summary

- Week 1 (June 9–15): Foundations of Information Retrieval Studied IR systems, text preprocessing (tokenization, stopword removal, stemming, lemmatization), inverted index construction, retrieval models (Boolean, Vector Space, TF-IDF), similarity metrics (Cosine, Jaccard, Euclidean), and evaluation metrics (Precision, Recall, F1-score, MAP, NDCG).
- Week 2 (June 16–21): Machine Learning for IR Explored the role of ML in IR (ranking, classification, recommendations), implemented Naive Bayes for document classification, used Scikit-learn for model building, and studied feature engineering (lexical, semantic, behavioral).
- Week 3 (June 23–29): Learning to Rank and Recommendation Systems Learned about learning-to-rank approaches (pointwise, pairwise, listwise), RankNet and LambdaMART, embedding-based retrieval (word/sentence embeddings), and recommendation techniques (content-based, collaborative filtering).
- Week 4 (June 30–5): Project Week Built an end-to-end IR+ML system including text preprocessing, TF-IDF ranking, document classification, and evaluation using IR metrics. Integrated concepts from previous weeks into a working prototype.

## 6 System Design / Workflow / Architecture



# 7 Implementation Details

1. **Overview**
   - A travel recommendation web application built using **Flask**.
   - Uses **Information Retrieval (IR)** and **Machine Learning (ML)** for ranked recommendations.
   - Supports keyword-based and filter-based search with optional ML-based re-ranking.
2. **Technology Stack**
   - **Frontend:** HTML, CSS (Bootstrap), JavaScript
   - **Backend:** Flask (Python)
   - **ML Model:** XGBoost Regressor (`XGBRegressor`)
   - **IR Techniques:** TF-IDF vectorization, Cosine Similarity
   - **Dataset:** 3000+ entries (Kaggle)
   - **Image Search:** Google Programmable Search Engine
3. **Functional Modules**
   (a) **User Interface**
      i. **Search Form (Landing Page)**
         - Input Fields:
           – City, Place, Minimum Rating, Maximum Duration, Month
         - Checkbox: `Apply Hard Filters`
         - IR/ML Settings:
           – Keywords (e.g., “lake”, “heritage”)
           – Top N Results to retrieve
      ii. **Results Page**
         - Shows top N results ranked by:
           – IR-only (TF-IDF + Cosine Similarity)
           – ML re-ranking (predicted rating)
         - Each result includes:
           – Place Name, City, Rating, IR Score, ML Score
           – Best Time to Visit, Duration, View Details button
         - Displays evaluation metrics for IR vs IR+ML
      iii. **Details Page**

- Displays:
  - Rating, Best Time, Duration, Distance from City
  - 5 Images (via Google Search)
  - City and Place Descriptions
  - Back to Results link

4. **Information Retrieval (IR) Pipeline**
   - **Vectorization:** TF-IDF on text fields (e.g., name, description)
   - **Similarity:** Cosine similarity between query and dataset entries
   - **Ranking:** Based on similarity scores
   - **Filtering:** Applied if hard filters are enabled:
     - Filters by city, rating, duration, and month

5. **Machine Learning (ML) Pipeline**
   - **Model:** `XGBRegressor`
   - **Target:** User rating
   - **Features:**
     - Place description, City, Month, Duration, TF-IDF vector
   - **Re-ranking Process:**
     - Top N IR results fed to ML model
     - Model predicts ML score
     - Results sorted by predicted ML score

6. **Evaluation Metrics**
   - **NDCG@N:** Measures ranking quality
   - **Precision@N:** Relevance in top-N results
   - **Recall@N:** Coverage of relevant items
   - **MAP@N:** Mean of average precision
   - **F-measure@N:** Harmonic mean of precision and recall

7. **External API Integration**
   - Google Programmable Search API used to fetch 5 images per place
   - Query is constructed using place and city names
   - Images displayed on the details page

8. **Navigation Flow**

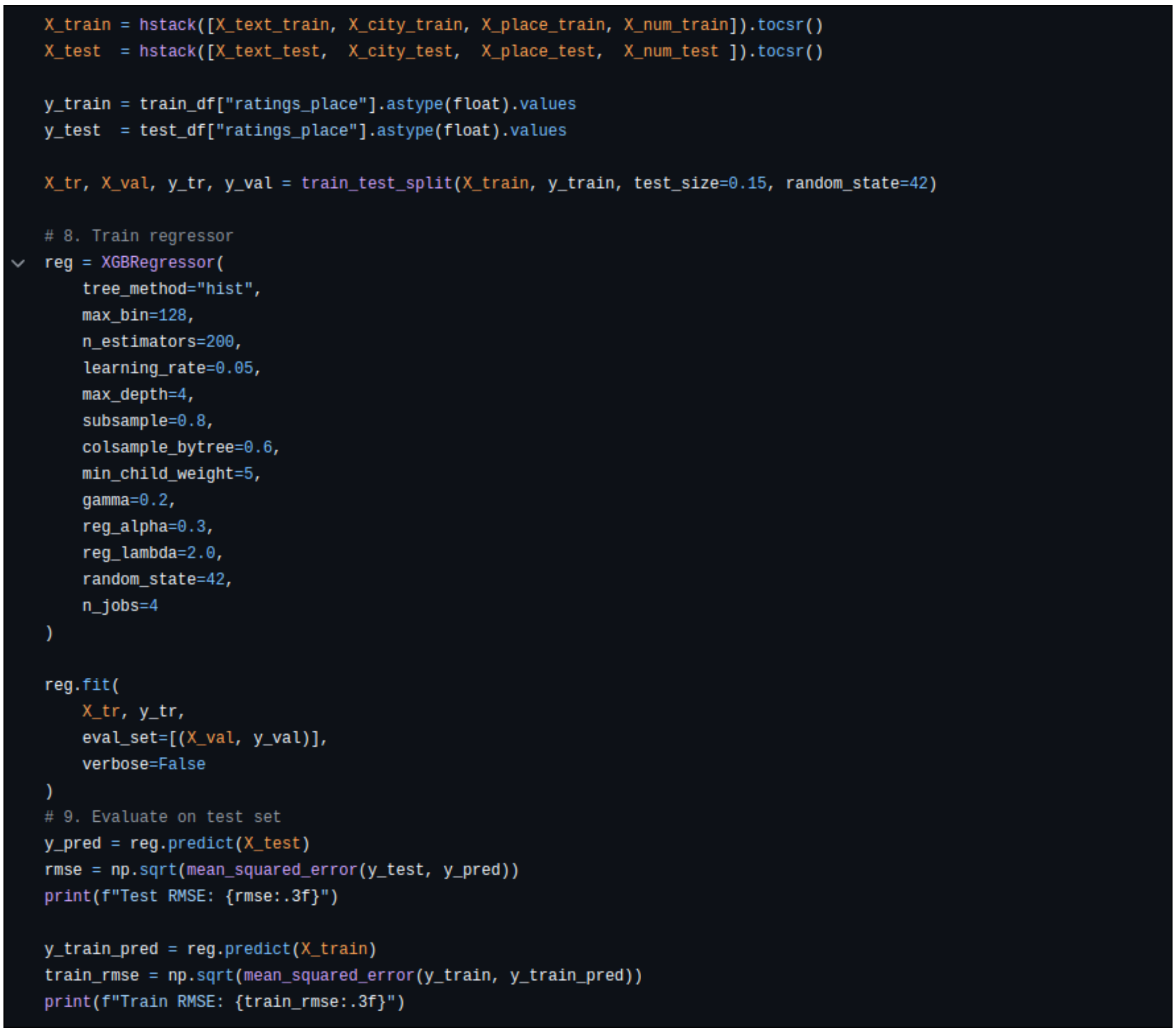
   (a) Landing Page: Search form submission

   (b) Results Page: Ranked recommendations and metrics

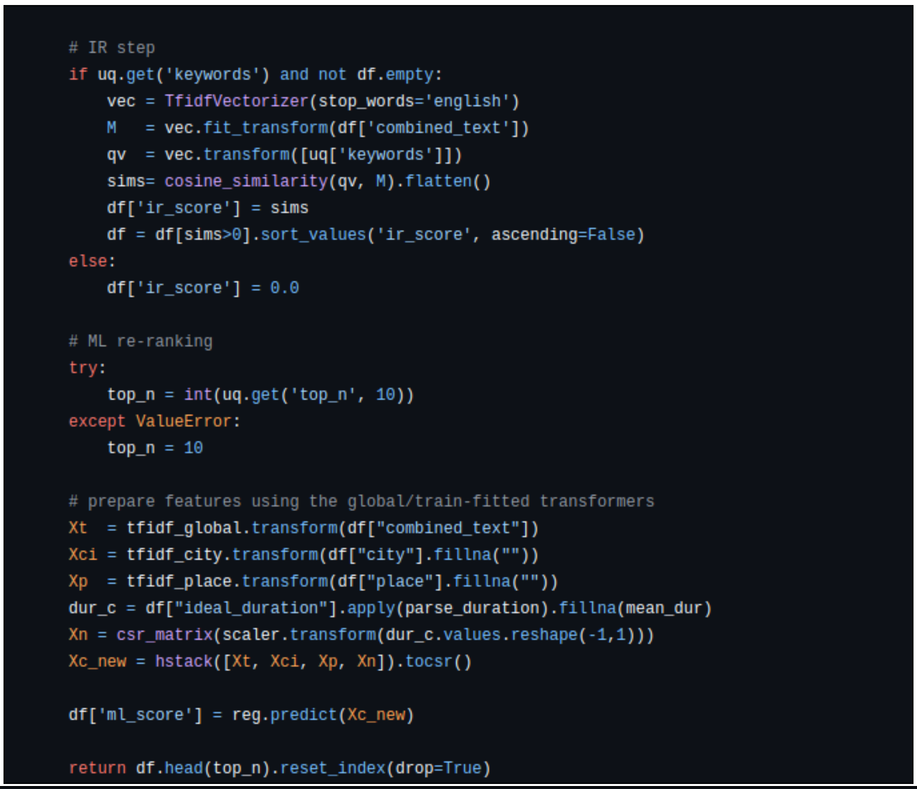
   (c) Details Page: Rich information and images

   (d) Back links to navigate between pages

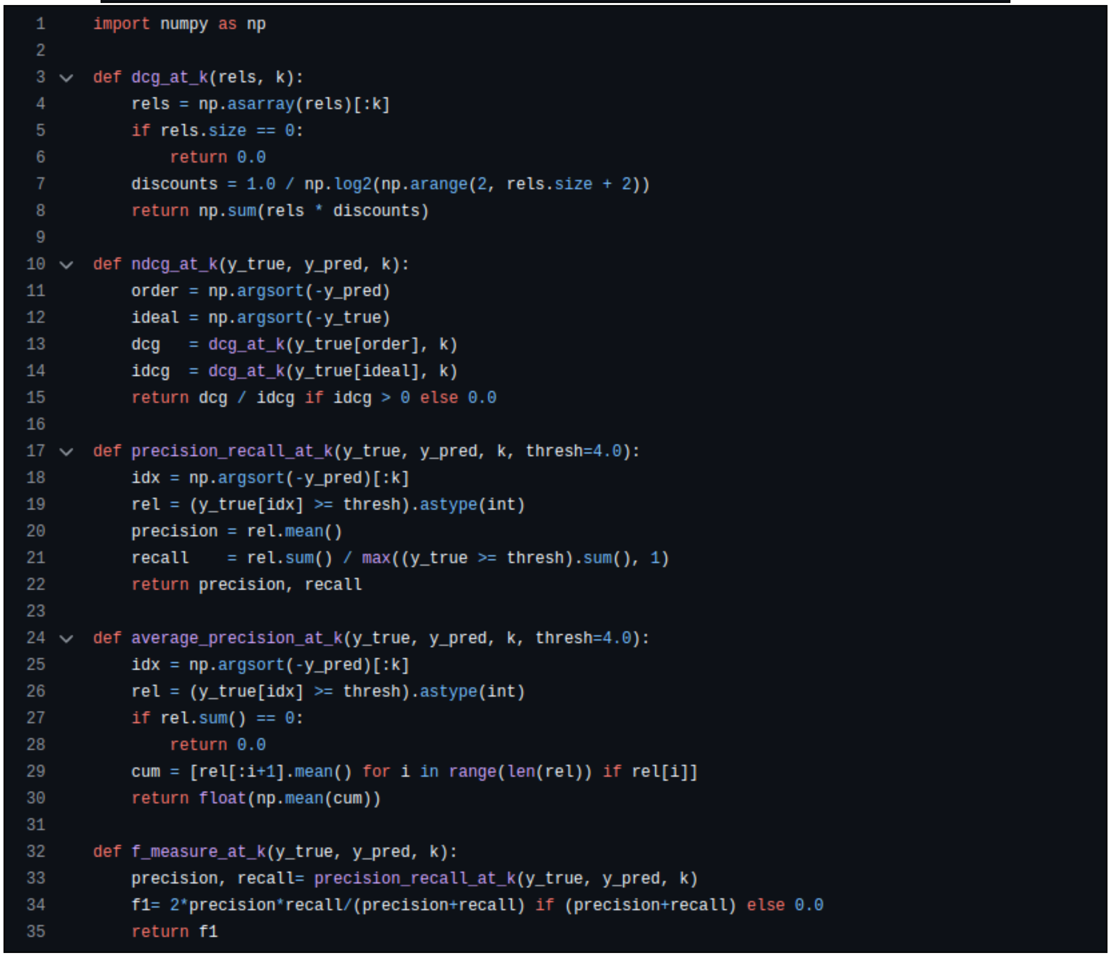
## 8 Screenshots / Code Snapshots / Output Samples



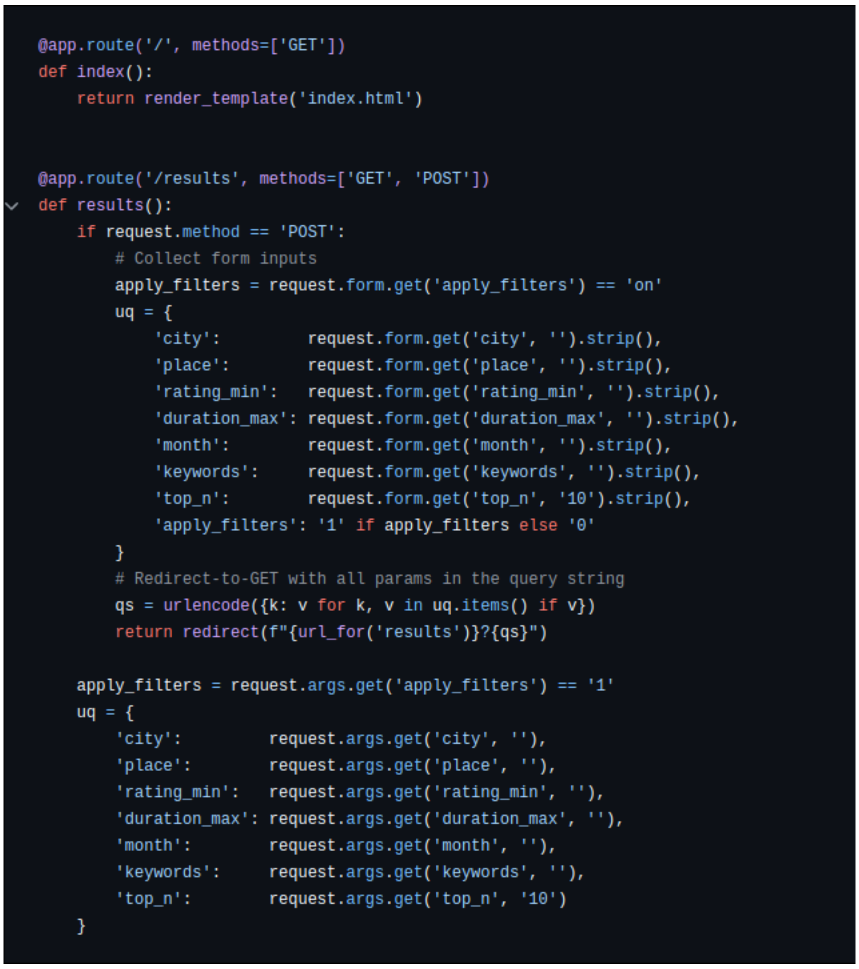
```
X_train = hstack([X_text_train, X_city_train, X_place_train, X_num_train]).tocsr()
X_test  = hstack([X_text_test,  X_city_test,  X_place_test,  X_num_test ]).tocsr()

y_train = train_df["ratings_place"].astype(float).values
y_test  = test_df["ratings_place"].astype(float).values

X_tr, X_val, y_tr, y_val = train_test_split(X_train, y_train, test_size=0.15, random_state=42)

# 8. Train regressor
reg = XGBRegressor(
    tree_method="hist",
    max_bin=128,
    n_estimators=200,
    learning_rate=0.05,
    max_depth=4,
    subsample=0.8,
    colsample_bytree=0.6,
    min_child_weight=5,
    gamma=0.2,
    reg_alpha=0.3,
    reg_lambda=2.0,
    random_state=42,
    n_jobs=4
)

reg.fit(
    X_tr, y_tr,
    eval_set=[(X_val, y_val)],
    verbose=False
)
# 9. Evaluate on test set
y_pred = reg.predict(X_test)
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
print(f"Test RMSE: {rmse:.3f}")

y_train_pred = reg.predict(X_train)
train_rmse = np.sqrt(mean_squared_error(y_train, y_train_pred))
print(f"Train RMSE: {train_rmse:.3f}")
```



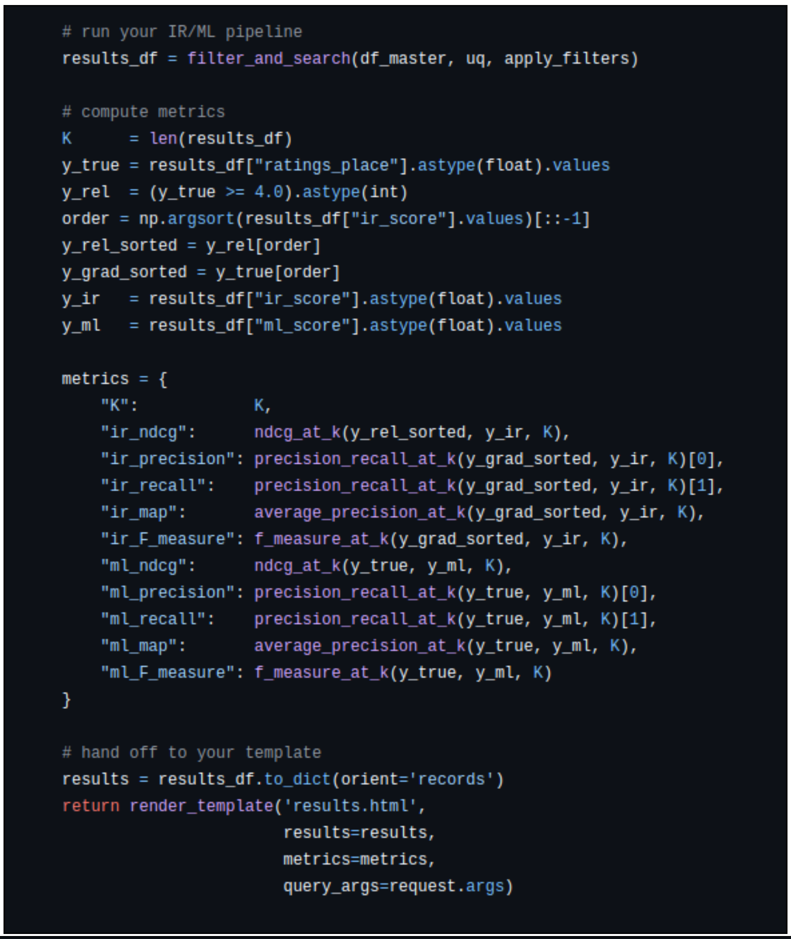
```python
    # IR step
    if uq.get('keywords') and not df.empty:
        vec = TfidfVectorizer(stop_words='english')
        M   = vec.fit_transform(df['combined_text'])
        qv  = vec.transform([uq['keywords']])
        sims= cosine_similarity(qv, M).flatten()
        df['ir_score'] = sims
        df = df[sims>0].sort_values('ir_score', ascending=False)
    else:
        df['ir_score'] = 0.0

    # ML re-ranking
    try:
        top_n = int(uq.get('top_n', 10))
    except ValueError:
        top_n = 10

    # prepare features using the global/train-fitted transformers
    Xt  = tfidf_global.transform(df["combined_text"])
    Xci = tfidf_city.transform(df["city"].fillna(""))
    Xp  = tfidf_place.transform(df["place"].fillna(""))
    dur_c = df["ideal_duration"].apply(parse_duration).fillna(mean_dur)
    Xn = csr_matrix(scaler.transform(dur_c.values.reshape(-1,1)))
    Xc_new = hstack([Xt, Xci, Xp, Xn]).tocsr()

    df['ml_score'] = reg.predict(Xc_new)

    return df.head(top_n).reset_index(drop=True)
```



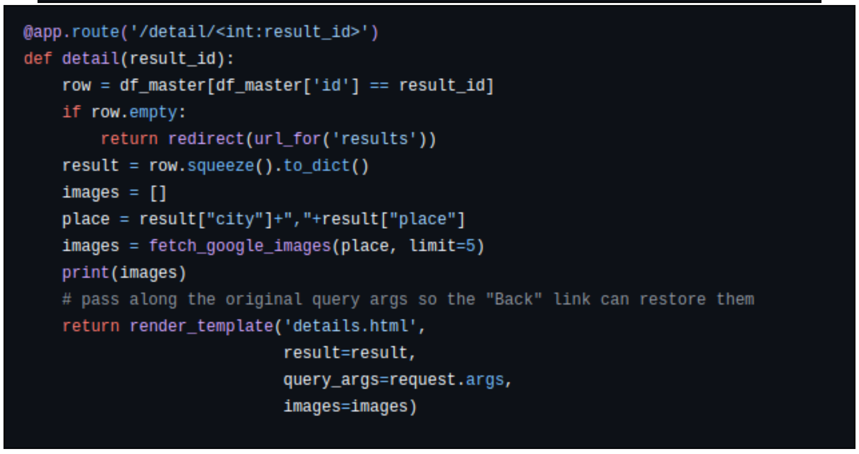
```python
import numpy as np

def dcg_at_k(rels, k):
    rels = np.asarray(rels)[:k]
    if rels.size == 0:
        return 0.0
    discounts = 1.0 / np.log2(np.arange(2, rels.size + 2))
    return np.sum(rels * discounts)

def ndcg_at_k(y_true, y_pred, k):
    order = np.argsort(-y_pred)
    ideal = np.argsort(-y_true)
    dcg   = dcg_at_k(y_true[order], k)
    idcg  = dcg_at_k(y_true[ideal], k)
    return dcg / idcg if idcg > 0 else 0.0

def precision_recall_at_k(y_true, y_pred, k, thresh=4.0):
    idx = np.argsort(-y_pred)[:k]
    rel = (y_true[idx] >= thresh).astype(int)
    precision = rel.mean()
    recall    = rel.sum() / max((y_true >= thresh).sum(), 1)
    return precision, recall

def average_precision_at_k(y_true, y_pred, k, thresh=4.0):
    idx = np.argsort(-y_pred)[:k]
    rel = (y_true[idx] >= thresh).astype(int)
    if rel.sum() == 0:
        return 0.0
    cum = [rel[:i+1].mean() for i in range(len(rel)) if rel[i]]
    return float(np.mean(cum))

def f_measure_at_k(y_true, y_pred, k):
    precision, recall= precision_recall_at_k(y_true, y_pred, k)
    f1= 2*precision*recall/(precision+recall) if (precision+recall) else 0.0
    return f1
```



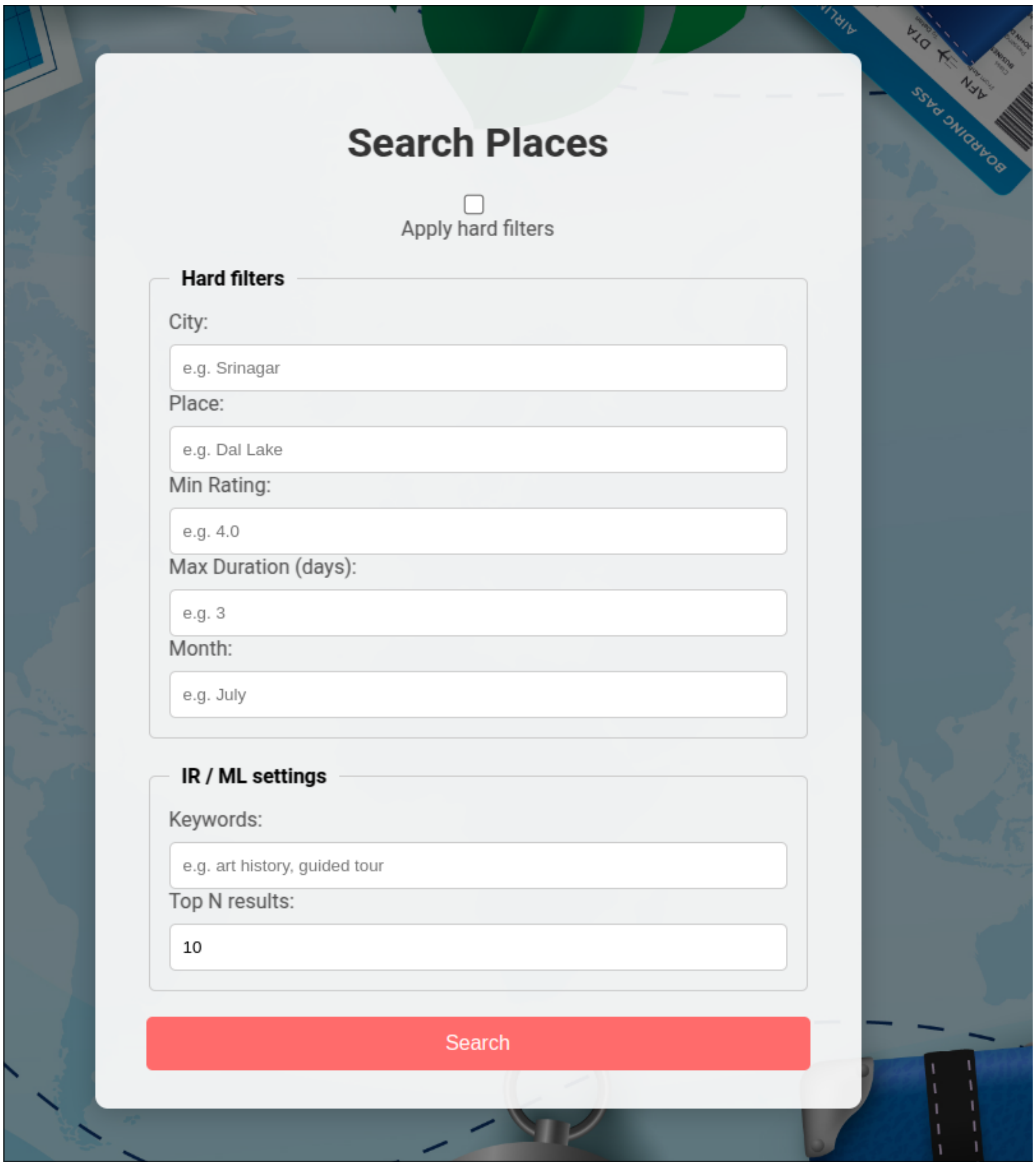
```python
@app.route('/', methods=['GET'])
def index():
    return render_template('index.html')


@app.route('/results', methods=['GET', 'POST'])
def results():
    if request.method == 'POST':
        # Collect form inputs
        apply_filters = request.form.get('apply_filters') == 'on'
        uq = {
            'city':         request.form.get('city', '').strip(),
            'place':        request.form.get('place', '').strip(),
            'rating_min':   request.form.get('rating_min', '').strip(),
            'duration_max': request.form.get('duration_max', '').strip(),
            'month':        request.form.get('month', '').strip(),
            'keywords':     request.form.get('keywords', '').strip(),
            'top_n':        request.form.get('top_n', '10').strip(),
            'apply_filters': '1' if apply_filters else '0'
        }
        # Redirect-to-GET with all params in the query string
        qs = urlencode({k: v for k, v in uq.items() if v})
        return redirect(f"{url_for('results')}?{qs}")

    apply_filters = request.args.get('apply_filters') == '1'
    uq = {
        'city':         request.args.get('city', ''),
        'place':        request.args.get('place', ''),
        'rating_min':   request.args.get('rating_min', ''),
        'duration_max': request.args.get('duration_max', ''),
        'month':        request.args.get('month', ''),
        'keywords':     request.args.get('keywords', ''),
        'top_n':        request.args.get('top_n', '10')
    }
```



```python
    # run your IR/ML pipeline
    results_df = filter_and_search(df_master, uq, apply_filters)

    # compute metrics
    K      = len(results_df)
    y_true = results_df["ratings_place"].astype(float).values
    y_rel  = (y_true >= 4.0).astype(int)
    order = np.argsort(results_df["ir_score"].values)[::-1]
    y_rel_sorted = y_rel[order]
    y_grad_sorted = y_true[order]
    y_ir   = results_df["ir_score"].astype(float).values
    y_ml   = results_df["ml_score"].astype(float).values

    metrics = {
        "K":            K,
        "ir_ndcg":      ndcg_at_k(y_rel_sorted, y_ir, K),
        "ir_precision": precision_recall_at_k(y_grad_sorted, y_ir, K)[0],
        "ir_recall":    precision_recall_at_k(y_grad_sorted, y_ir, K)[1],
        "ir_map":       average_precision_at_k(y_grad_sorted, y_ir, K),
        "ir_F_measure": f_measure_at_k(y_grad_sorted, y_ir, K),
        "ml_ndcg":      ndcg_at_k(y_true, y_ml, K),
        "ml_precision": precision_recall_at_k(y_true, y_ml, K)[0],
        "ml_recall":    precision_recall_at_k(y_true, y_ml, K)[1],
        "ml_map":       average_precision_at_k(y_true, y_ml, K),
        "ml_F_measure": f_measure_at_k(y_true, y_ml, K)
    }

    # hand off to your template
    results = results_df.to_dict(orient='records')
    return render_template('results.html',
                           results=results,
                           metrics=metrics,
                           query_args=request.args)
```



```python
@app.route('/detail/<int:result_id>')
def detail(result_id):
    row = df_master[df_master['id'] == result_id]
    if row.empty:
        return redirect(url_for('results'))
    result = row.squeeze().to_dict()
    images = []
    place = result["city"]+","+result["place"]
    images = fetch_google_images(place, limit=5)
    print(images)
    # pass along the original query args so the "Back" link can restore them
    return render_template('details.html',
                           result=result,
                           query_args=request.args,
                           images=images)
```



Search Places

☐ Apply hard filters

**Hard filters**

City: e.g. Srinagar

Place: e.g. Dal Lake

Min Rating: e.g. 4.0

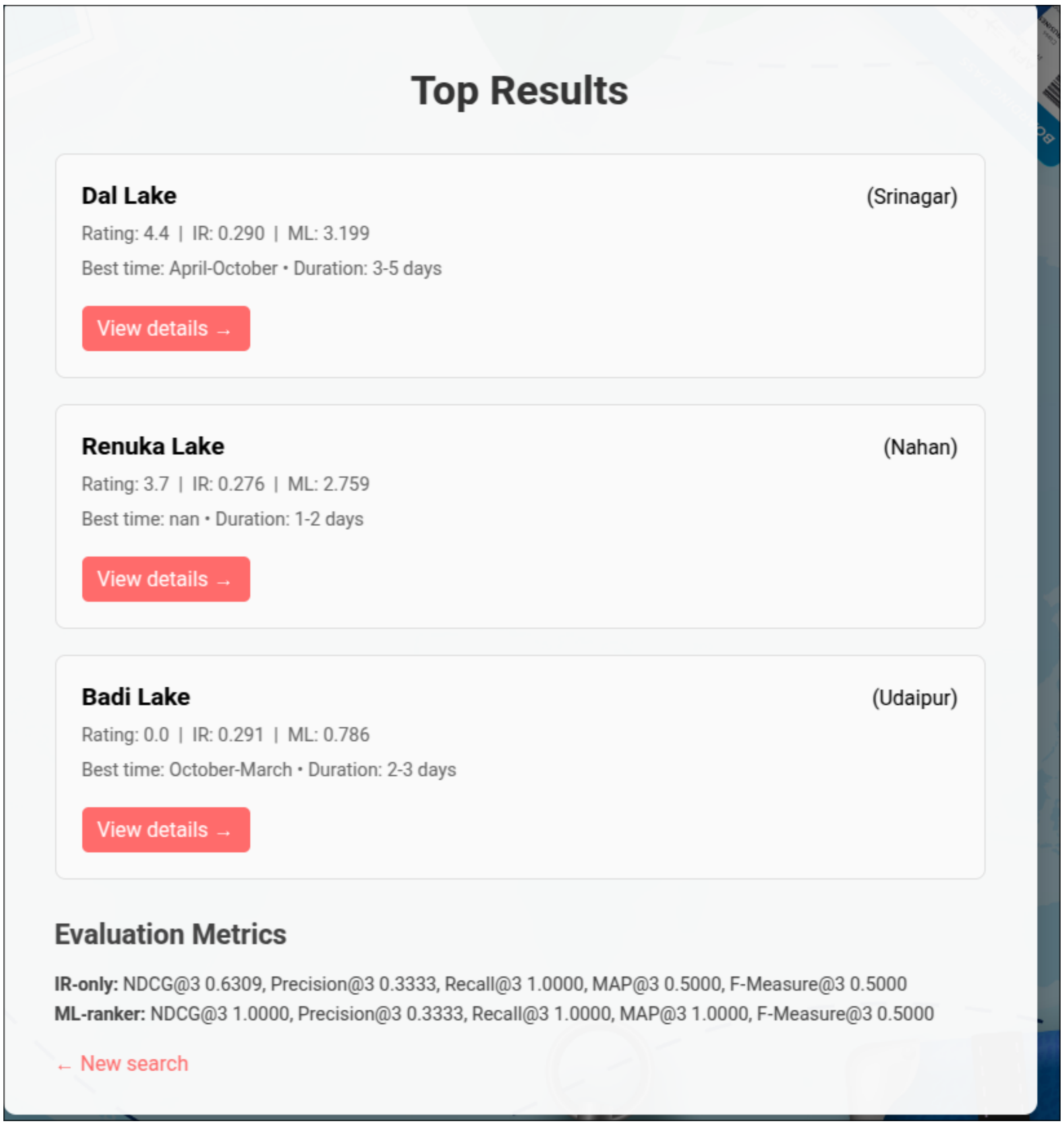
Max Duration (days): e.g. 3

Month: e.g. July

**IR / ML settings**

Keywords: e.g. art history, guided tour

Top N results: 10



Top Results

**Dal Lake** (Srinagar)

Rating: 4.4 | IR: 0.290 | ML: 3.199

Best time: April-October • Duration: 3-5 days

**Renuka Lake** (Nahan)

Rating: 3.7 | IR: 0.276 | ML: 2.759

Best time: nan • Duration: 1-2 days

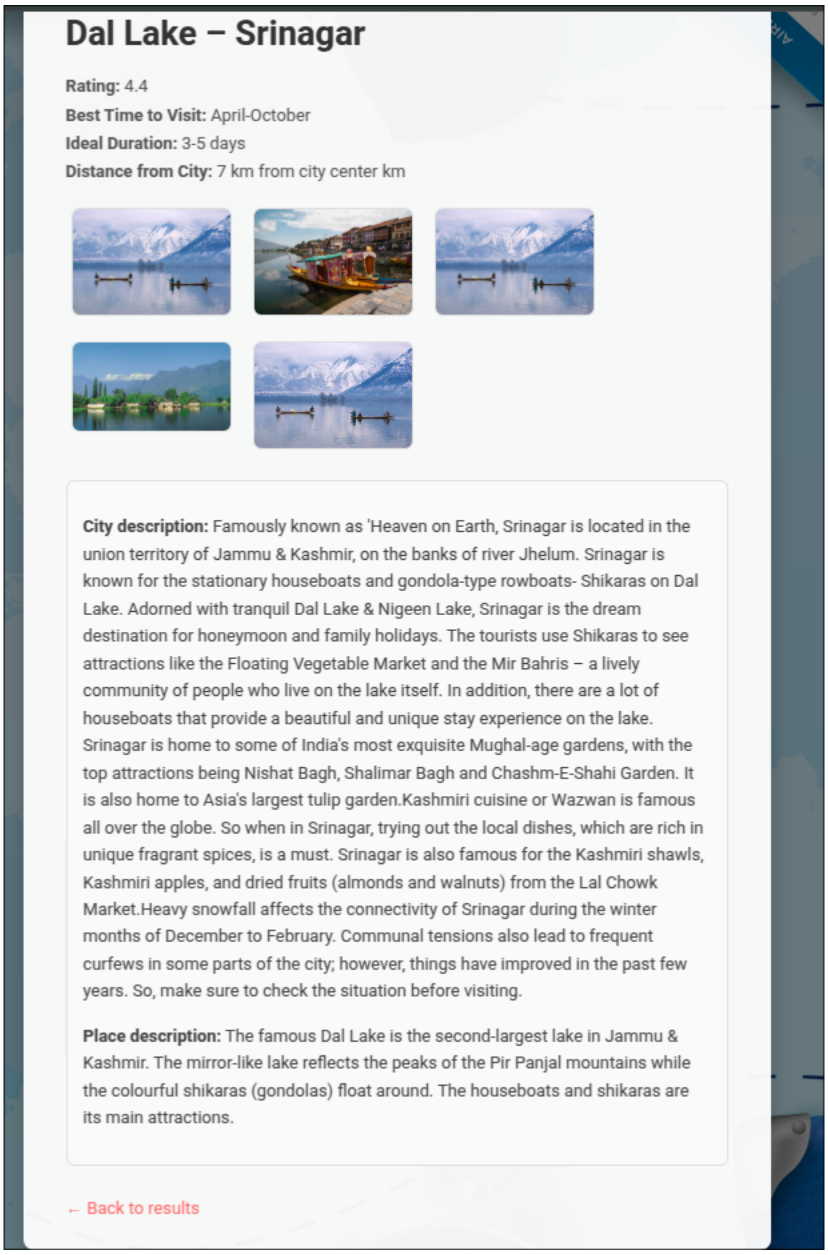
**Badi Lake** (Udaipur)

Rating: 0.0 | IR: 0.291 | ML: 0.786

Best time: October-March • Duration: 2-3 days

Evaluation Metrics

**IR-only:** NDCG@3 0.6309, Precision@3 0.3333, Recall@3 1.0000, MAP@3 0.5000, F-Measure@3 0.5000
**ML-ranker:** NDCG@3 1.0000, Precision@3 0.3333, Recall@3 1.0000, MAP@3 1.0000, F-Measure@3 0.5000



Dal Lake – Srinagar

**Rating:** 4.4
**Best Time to Visit:** April-October
**Ideal Duration:** 3-5 days
**Distance from City:** 7 km from city center km

**City description:** Famously known as 'Heaven on Earth, Srinagar is located in the union territory of Jammu & Kashmir, on the banks of river Jhelum. Srinagar is known for the stationary houseboats and gondola-type rowboats- Shikaras on Dal Lake. Adorned with tranquil Dal Lake & Nigeen Lake, Srinagar is the dream destination for honeymoon and family holidays. The tourists use Shikaras to see attractions like the Floating Vegetable Market and the Mir Bahris – a lively community of people who live on the lake itself. In addition, there are a lot of houseboats that provide a beautiful and unique stay experience on the lake. Srinagar is home to some of India's most exquisite Mughal-age gardens, with the top attractions being Nishat Bagh, Shalimar Bagh and Chashm-E-Shahi Garden. It is also home to Asia's largest tulip garden.Kashmiri cuisine or Wazwan is famous all over the globe. So when in Srinagar, trying out the local dishes, which are rich in unique fragrant spices, is a must. Srinagar is also famous for the Kashmiri shawls, Kashmiri apples, and dried fruits (almonds and walnuts) from the Lal Chowk Market.Heavy snowfall affects the connectivity of Srinagar during the winter months of December to February. Communal tensions also lead to frequent curfews in some parts of the city; however, things have improved in the past few years. So, make sure to check the situation before visiting.

**Place description:** The famous Dal Lake is the second-largest lake in Jammu & Kashmir. The mirror-like lake reflects the peaks of the Pir Panjal mountains while the colourful shikaras (gondolas) float around. The houseboats and shikaras are its main attractions.

# 9 Challenges Faced and Solutions

1. **No Rating CSV Available:** The dataset lacked user-specific ratings for direct supervised training. We addressed this by training an `XGBRegressor` to predict ratings using extracted features, and used these predicted scores for re-ranking IR results.

2. **Missing or Inconsistent Fields:** Several entries had incomplete data (e.g., duration, best time). We handled this with preprocessing steps such as median imputation, placeholder tokens for text fields, and conditional logic to clean and standardize records.

3. **Feature Imbalance (Text vs Numeric):** TF-IDF features had significantly higher dimensionality compared to numeric ones. We applied numeric scaling and feature engineering to ensure balanced contribution to the ML model.

4. **Lack of User Feedback for Evaluation:** In absence of real user interactions, we used offline evaluation metrics (NDCG@3, Precision@3, etc.) to validate model performance and maintained a modular structure to enable future feedback integration.

5. **Google Programmable Search Engine:** It gave us pictures from facebook and instagram which were not available without login and thus on html page there came no pics but alt text, we added a check in the getimg.py file to prevent this from happening by checking if it starts with '/image'.

## 10 Key Learnings

1. **Foundations of Information Retrieval**
   - Understood how IR systems function, focusing on preprocessing techniques such as tokenization, stopword removal, stemming, and lemmatization.
   - Built an inverted index for efficient document retrieval.
   - Applied retrieval models including Boolean, Vector Space, and TF-IDF.
   - Used similarity metrics like Cosine, Jaccard, and Euclidean distance to assess document relevance.
   - Evaluated system performance using metrics such as Precision, Recall, F1-score, MAP, and NDCG.
2. **Machine Learning for IR**
   - Explored how ML enhances IR through classification and ranking tasks.
   - Implemented Naive Bayes for document classification using Scikit-learn.
   - Conducted feature engineering using lexical, semantic, and behavioral features to improve model effectiveness.
3. **Learning to Rank and Recommendations**
   - Studied learning-to-rank methods: pointwise, pairwise, and listwise approaches.
   - Learned ranking models such as RankNet and LambdaMART.
   - Applied embedding-based retrieval using word and sentence embeddings for semantic search.
   - Developed recommendation systems using content-based and collaborative filtering techniques.
4. **End-to-End System Development**
   - Designed and implemented a full IR+ML pipeline, including preprocessing, ranking, classification, and evaluation.
   - Integrated diverse models into a functional travel recommendation application.
   - Focused on user experience by building a clean, intuitive interface for personalized destination suggestions.

## 11 Conclusion

Over four weeks, we explored the core principles and practical applications of Information Retrieval (IR) and Machine Learning (ML) to build a travel recommendation system that goes beyond conventional platforms like MakeMyTrip. Unlike traditional travel sites that focus mainly on logistics, our system focuses on personalized destination recommendations based on user interests, using techniques like TF-IDF ranking, classification models, and similarity metrics. By integrating IR techniques with ML-driven classification and ranking algorithms, we created a user-friendly prototype that retrieves relevant travel destinations and displays them with images and contextual insights, enhancing user experience and engagement.

## References

[1] Dataset. https://www.kaggle.com/datasets/naqibahmedkadri/famous-indian-tourist-places

[2] Flask Documentation. https://flask.palletsprojects.com/

[3] XGBoost Documentation. https://xgboost.readthedocs.io/

[4] NumPy Documentation. https://numpy.org/doc/

[5] Pandas Documentation. https://pandas.pydata.org/docs/

[6] SciPy Documentation. https://docs.scipy.org/doc/scipy/

[7] Scikit-learn Documentation. https://scikit-learn.org/stable/documentation.html

# 12 Appendix

This appendix includes additional information, datasets, evaluation summaries, and architectural references that support the implementation and evaluation of the Travel Recommendation System.

## A.1 Dataset Overview

- **Source:** Public Kaggle dataset containing 3000+ tourist places across various Indian cities.
- **Fields Used:** Place name, City, Description, Tags, Best time to visit, Ideal duration, Distance from city, and Ratings.
- **Preprocessing:** Missing values were handled via imputation or exclusion, and categorical/text data was encoded for IR and ML pipelines.

## A.2 IR and ML Pipeline Configuration

- **Text Vectorization:** TF-IDF on place descriptions and tags.
- **Similarity Measure:** Cosine similarity used to rank initial results.
- **Re-ranking Model:** `XGBRegressor`, trained to predict rating scores.
- **Top-N Selection:** User-specified value used to limit final output set.

## A.3 Evaluation Metrics

- Metrics used for comparing IR-only vs IR+ML ranking:
  - NDCG@3 (Normalized Discounted Cumulative Gain)
  - Precision@3
  - Recall@3
  - MAP@3 (Mean Average Precision)
  - F-Measure@3
- Evaluation was performed using simulated user relevance rankings across multiple test queries.

## A.4 Example Query Scenarios

1. **Query:** "lake"
   **Filters:** City = Srinagar, Rating $\geq 4.0$
   **Top Results:** Dal Lake, Wular Lake, Nigeen Lake

2. **Query:** "heritage fort"
   **Filters:** Duration $\leq 2$ days
   **Top Results:** Gwalior Fort, Mehrangarh Fort, Jaisalmer Fort

## A.5 System Architecture Summary

- User mits search via Flask frontend.
- TF-IDF computes cosine similarity scores over the dataset.
- Top-N IR results passed to ML model for re-ranking using predicted ratings.
- Final results rendered with evaluation metrics and a link to a detailed view (images, descriptions, etc.).
- External image data fetched using Google Programmable Search API.

## A.6 Additional Notes

- Image API queries were tuned using a combination of place name + city name to ensure quality.
- Backend allows modular extension for user feedback collection and real-time re-training.