# Smart Search System for Analytics Vidhya Courses

## Project Overview

The project involves creating a Smart Search System to assist users in finding relevant free courses on the Analytics Vidhya platform. The system leverages web scraping, data processing, and a machine learning-based semantic search engine to enhance user experience. The final product is deployed on Huggingface Spaces.

---

## Methodology

### 1. Data Collection

Data was collected using web scraping techniques from the Analytics Vidhya course platform. The scraping process involved extracting the following details for each course:

- **Course Name**
- **Course URL**
- **Lesson Count**
- **Price**
- **Description**
- **Instructor**
- **Rating**
- **Difficulty Level**

**Tools and Libraries Used**

- **Python Libraries**: requests, BeautifulSoup (for web scraping), csv, and openpyxl (for saving data).
- **Scraping Code**:
  - A multi-page scraper was implemented to handle pagination and fetch course details.
  - Additional requests were made to course-specific URLs to gather detailed information.

### 2. Data Cleaning and Preprocessing

After scraping, the data underwent cleaning and preprocessing to ensure consistency and usability for embedding generation.

- **Steps Taken**:

    - Filling missing values for fields like lesson_count, instructor, rating, and difficulty_level.
    - Normalizing text fields (e.g., converting to lowercase).
    - Combining the course_name and description fields into a single field (text) for semantic embeddings.
- **Libraries Used**:
    - pandas and numpy for data manipulation.

## 3. Embedding Generation

To enable semantic search, embeddings were generated for the textual data using a pre-trained transformer model.

**Steps Taken:**

1. **Model Selection**:
    - Chosen Model: all-MiniLM-L6-v2 from the sentence-transformers library.
    - This model is optimized for semantic similarity tasks and offers a balance between performance and speed.
2. **Embedding Creation**:
    - Generated embeddings for each course's text field.
    - Stored the embeddings as numpy arrays for integration with the search engine.
3. **Libraries Used**:
    - sentence-transformers and torch for embedding generation.

## 4. Smart Search Implementation

The search engine uses a K-Nearest Neighbors (KNN) approach to retrieve courses based on query embeddings.

**Steps Taken:**

1. **Query Handling**:
    - User-provided queries are converted into embeddings using the same transformer model.
2. **Search Algorithm**:
    - Utilized Scikit-learn's NearestNeighbors to compute similarity scores between query and course embeddings.
    - Returned the top-N most relevant courses.

3. **Output**:
   - Results include course name, URL, description, rating, and difficulty level, along with relevance scores.

## 5. Deployment

The final application was deployed on Huggingface Spaces using Gradio for the user interface.

**Steps Taken:**

1. **Gradio Interface**:
   - Designed an input field for search queries and an output area to display search results.
2. **Deployment Workflow**:
   - Project files (including code, cleaned dataset, and model dependencies) were organized in a Git repository.
   - Pushed to Huggingface Spaces for public access.
3. **URL**:
   - A publicly accessible link was generated for the deployed application.

## 6. Evaluation

- **Testing**:
  - Verified the scraper's accuracy by manually cross-checking with the website.
  - Ensured search relevance by testing multiple user queries.
- **Performance**:
  - Validated the system's responsiveness and scalability on Huggingface Spaces.

---

# Key Features

1. **Comprehensive Search**:
   - Users can search using keywords or natural language queries.
2. **Detailed Course Information**:
   - Outputs include essential course details and direct links for access.
3. **Scalable Deployment**:
   - Hosted on Huggingface Spaces for easy accessibility.

## Project Repository Structure

```
project-directory/
 ├── scraper.py            # Script for web scraping
 ├── data/
|    ├── courses_detailed.csv  # Raw scraped data
|    ├── cleaned_courses.csv   # Processed data
 ├── app.py                # Gradio application code
 ├── requirements.txt        # Dependencies
 ├── README.md             # Project documentation
└── huggingface_repo/       # Git repository for Huggingface
```

## Future Improvements

1. **Advanced Ranking**:
    - Incorporate user feedback to improve ranking metrics.
2. **Real-time Updates**:
    - Automate scraping to keep course data up-to-date.
3. **Personalization**:
    - Add user-specific recommendations based on past searches.

## Conclusion

This project demonstrates the integration of data scraping, natural language processing, and machine learning techniques to build a user-friendly search system. By hosting on Huggingface Spaces, the system is accessible, scalable, and ready for real-world application.