Best Price Flight Recommender

Project Overview

The Best Price Flight Recommender aims to provide users with personalized flight recommendations at the best prices. By scraping flight data from Expedia, processing and cleaning the data, integrating with OpenAI for intelligent filtering, and providing a user-friendly interface, the system helps travelers make better-informed decisions.

Components

1. Web Crawler (Data Extraction)

Purpose: Fetch flight details (e.g., airlines, times, durations, prices) from Expedia.

- **Technologies:**
    - Python: requests, BeautifulSoup for static pages
    - Selenium for handling dynamic content (JavaScript-driven pages)

Sample Code (Static Example):

```
import requests
from bs4 import BeautifulSoup

def fetch_flight_data(url):
    response = requests.get(url)
    if response.status_code == 200:
        soup = BeautifulSoup(response.content, "html.parser")
        # Extraction logic for flight details goes here
        return flight_data
    return None
```

Selenium Example (Dynamic Content):

```python
from selenium import webdriver
from selenium.webdriver.chrome.options import Options
from selenium.webdriver.common.by import By
import time, csv

def scrape_flights(origin, destination, depart_date, return_date, output_csv):
    url = (
        f"https://www.expedia.com/Flights-Search?flight-type=on&mode=search&trip=roundtrip
        f"&leg1=from:{origin},to:{destination},departure:{depart_date}TANYT"
        f"&leg2=from:{destination},to:{origin},departure:{return_date}TANYT"
        "&options=cabinclass:economy"
        f"&fromDate={depart_date}&toDate={return_date}"
        f"&d1={depart_date}&d2={return_date}&passengers=adults:1"
    )

    chrome_options = Options()
    chrome_options.add_argument("--headless")
    chrome_options.add_argument("--no-sandbox")
    chrome_options.add_argument("--disable-dev-shm-usage")

    driver = webdriver.Chrome(options=chrome_options)
    driver.get(url)
    time.sleep(10)  # Wait for page to load

    flight_cards = driver.find_elements(By.CSS_SELECTOR, "li[data-test-id='offer-listing']
    results = []
    for card in flight_cards:
        airline = card.find_element(By.CSS_SELECTOR, "[data-test-id='airline-name']").text
        departure_time = card.find_element(By.CSS_SELECTOR, "[data-test-id='departure-time
        arrival_time = card.find_element(By.CSS_SELECTOR, "[data-test-id='arrival-time']")
        duration = card.find_element(By.CSS_SELECTOR, "[data-test-id='duration']").text if
        price = card.find_element(By.CSS_SELECTOR, "[data-test-id='listing-price-dollars']

        results.append([airline, departure_time, arrival_time, duration, price])

    driver.quit()

    with open(output_csv, "w", newline="", encoding="utf-8") as f:
        writer = csv.writer(f)
        writer.writerow(["Airline", "Departure Time", "Arrival Time", "Duration", "Price"]
        writer.writerows(results)
```

2. Data Processing (Cleaning & Standardization)

Purpose: Ensure consistent and clean data for recommendation logic.

- **Tasks:**
    - Remove irrelevant fields
    - Convert price formats to numerical values
    - Handle missing values and data type conversions

Sample Code:

```
import pandas as pd

def clean_flight_data(raw_data):
    df = pd.DataFrame(raw_data)
    df['price'] = df['price'].apply(lambda x: float(x.replace('$', '')))
    # Additional cleaning steps as needed
    return df
```

3. Integration with OpenAI (Recommendation Logic)

Purpose: Use OpenAI's models (e.g., GPT-4 or gpt-3.5-turbo) to generate personalized recommendations.

- **Process:**
    1. Provide clean flight data as context
    2. Prompt OpenAI to suggest the best flights based on user preferences (budget, duration, airlines)
    3. Parse the model's response for actionable recommendations

Sample Code:

```python
import openai
import os

openai.api_key = os.getenv("OPENAI_API_KEY")

def get_recommendations(prompt):
    response = openai.Completion.create(
        model="gpt-4",
        prompt=prompt,
        max_tokens=150
    )
    return response.choices[0].text.strip()
```

4. Recommendation Engine (Filtering & Ranking)

Purpose: Filter and rank flights based on user-defined criteria (e.g., max budget, max duration).

Sample Code:

```python
def filter_recommendations(flight_data, budget=None, max_duration=None):
    # Assume flight_data is a list of dicts with keys like 'price' and 'duration'
    filtered_flights = [
        flight for flight in flight_data
        if (budget is None or flight['price'] <= budget) and
           (max_duration is None or flight['duration'] <= max_duration)
    ]
    # Additional sorting or ranking can be applied here
    return filtered_flights
```

5. User Interface (Frontend)

Purpose: Provide a simple web-based interface to input details (origin, destination, dates, budget) and display recommendations.

- **Technology:** Flask or Streamlet

Sample Flask Code:

```python
from flask import Flask, render_template, request

app = Flask(__name__)

@app.route('/', methods=['GET', 'POST'])
def home():
    if request.method == 'POST':
        # Extract user inputs
        # Fetch and process data, generate recommendations
        # Render results
        pass
    return render_template('index.html')

if __name__ == '__main__':
    app.run(debug=True)
```

Workflow Summary

1. **User Input:** User specifies origin, destination, travel dates, budget, etc.
2. **Web Crawler:** Data is scraped from Expedia.
3. **Data Processing:** Data is cleaned and standardized.
4. **OpenAI Analysis:** Model generates recommendations from processed data.
5. **Recommendation Engine:** Filters and refine results based on user preferences.
6. **User Interface:** Displays the final recommended flights.