Indian Historical Events 'Guess the Day?' Bot Report

This report summarizes the Jupyter Notebook, which implements an interactive chatbot designed to provide Indian historical events for a given date. The notebook demonstrates the use of an external API to fetch data and a simple command-line interface for user interaction.

1. Project Objective

The primary objective of this project is to create a simple, interactive bot that allows users to query Indian historical events by providing a specific date (month and day). The bot aims to provide interesting historical facts in a user-friendly command-line interface.

2. Setup and API Integration

The notebook relies on the requests library to interact with an external API for historical data.

- **Library Import**: The requests library is imported for making HTTP requests to the API.

  import requests

- **API Key**: A placeholder for the API-Ninjas key is provided. This key is essential for authenticating requests to the historical events API.

  API_NINJAS_KEY = 'DKSSLvH25MYASW8O5Gu4pw==4kyCeGGGY0xDf8hI' # Placeholder

  (Note: The actual API key is a placeholder and should be replaced with a valid key by the user.)
- **get_indian_events Function**: This function is responsible for fetching historical events.
  - It constructs a URL to the API-Ninjas historical events endpoint, filtering for events related to 'india' for a given month and day.
  - It includes a try-except block for robust error handling during API calls.
  - If events are found, it returns up to the first three events formatted as "{year}:

{event}".
    - If no specific Indian events are found for the date, it attempts a fallback query for general Indian historical events (without a specific date) to provide some information.
    - It handles API errors (non-200 status codes) and general exceptions during the fetch process.

```
def get_indian_events(month, day):
    url = 
f'https://api.api-ninjas.com/v1/historicalevents?month={month}&day={day}&text
=india'
    headers = {'X-Api-Key': API_NINJAS_KEY}
    try:
        response = requests.get(url, headers=headers)
        if response.status_code == 200:
            events = response.json()
            if events:
                return [f"{event['year']}: {event['event']}" for event in events[:3]]
            else:
                # Fallback to general Indian historical events if date-specific not found
                fallback_url = 'https://api.api-ninjas.com/v1/historicalevents?text=india'
                fallback_response = requests.get(fallback_url, headers=headers)
                if fallback_response.status_code == 200:
                    fallback_events = fallback_response.json()
                    if fallback_events:
                        return [f"{event['year']}: {event['event']}" for event in 
fallback_events[:3]]
                return ["No Indian historical events found for this date."]
        else:
            return [f"Error: {response.status_code} - {response.text}"]
    except Exception as e:
        return [f"Error fetching events: {e}"]
```

3. Bot Interaction Logic

The main function orchestrates the user interaction, input handling, and event display.

- **Welcome Message**: The bot greets the user and provides instructions on how to interact.
- **Input Loop**: It enters an infinite loop to continuously accept user input until an exit command is given.
- **Exit Commands**: Users can type 'bye', 'exit', or 'quit' to terminate the bot.
- **Date Parsing and Validation**:

    - It attempts to parse the user's input as a month-day (MM-DD) format.
    - It includes basic validation to ensure the month is between 1 and 12 and the day is between 1 and 31.
    - ValueError is caught if the input format is incorrect.
- **Event Display**: If a valid date is provided, get_indian_events is called, and the retrieved events are printed to the console.

```
def main():
    print("Welcome to 'Guess the Day?' Bot!")
    print("Type a date in MM-DD format to learn about Indian historical events.")
    print("Type 'bye', 'exit', or 'quit' to exit.\n")
    while True:
        user_input = input("You: ").strip()
        if user_input.lower() in ["bye", "exit", "quit"]:
            print("Bot: Goodbye! Hope you learned something interesting.")
            break
        try:
            month, day = map(int, user_input.split('-'))
            if 1 <= month <= 12 and 1 <= day <= 31:
                indian_events = get_indian_events(month, day)
                print("Bot: Indian Historical Events:")
                for event in indian_events:
                    print(f"    • {event}")
                print()
            else:
                print("Bot: Please enter a valid date (MM-DD, month 1-12, day 1-31).\n")
        except ValueError:
                print("Bot: I didn't understand that. Please enter the date in MM-DD format.\n")

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

4. Example Interactions

The notebook demonstrates several successful interactions:

- **Input: 12-04**
    - Output: 1971: Indo-Pakistani War of 1971: The Indian Navy attacks the Pakistan Navy and Karachi.
- **Input: 01-26**

    - Output: Multiple events for January 26th, including 1930: The Indian National Congress declares 26 January as Independence Day... and 1950: The Constitution of India comes into force...
- **Input: 08-15**
    - Output: Events for August 15th, such as 1947: India gains Independence from British rule...
- **Input: bye**
    - Output: Bot: Goodbye! Hope you learned something interesting.

5. Conclusion and Future Enhancements

The notebook provides a functional and interactive bot for retrieving Indian historical events. It effectively demonstrates API integration, basic input validation, and a conversational flow.

Possible future enhancements include:

- **More Robust Date Validation**: Implement more comprehensive date validation (e.g., checking for valid days in specific months, leap years).
- **Natural Language Understanding**: Integrate a more advanced NLP library or model to understand natural language queries (e.g., "What happened on Christmas Day in India?") instead of strict MM-DD format.
- **Expanded Event Details**: Allow users to ask for more details about a specific event if multiple are returned.
- **Persistent Chat History**: Implement a way to remember past conversations or user preferences.
- **Graphical User Interface (GUI)**: Develop a simple GUI using libraries like Tkinter, PyQt, or even a web-based interface using Flask/Django or Gradio (as seen in previous notebooks) for a more visual experience.
- **Error Handling Refinements**: Provide more user-friendly error messages for API issues or invalid inputs.
- **API Key Management**: Suggest more secure ways to handle API keys in a production environment (e.g., environment variables, secret management services).
- **Broader Historical Context**: Allow users to query for historical events from other countries or general world history.