|  |  |
| --- | --- |
| **Project Title** | **🚀 NASA Near-Earth Object (NEO) Tracking & Insights using Public API** |
| **Skills take away From This Project** | * + - API Integration and JSON Parsing     - Data Transformation     - Relational Database (SQL) Table Creation and Insertion     - Query Writing for Analytical Insights     - Streamlit Dashboard Development     - Filter-Based Data Interaction |
| **Domain** | Space Research & Astronomical Data Analysis |

### **Business Use Cases**

* **Threat Monitoring**: Analyze which asteroids are potentially hazardous based on velocity, size, and proximity.
* **Date-Based Exploration**: Understand patterns of asteroid occurrences and visits by date.
* **Filtering for Insight**: Help researchers or educators filter based on custom conditions like approach distance, size, or orbiting body.
* **Space Data Democratization**: Deliver an intuitive interface for non-technical users to explore astronomical data.

**Project Approach**

**Step 1: Getting the NASA API Key**

1. Go to:<https://api.nasa.gov>
2. Fill in your name and email, and submit the form to register.
3. Your **API key will be sent to your email address**.
4. The key will look like “qSxX9kz1L7....”
5. Use your key to format the API URL as follows:  
    **https://api.nasa.gov/neo/rest/v1/feed?start\_date=YYYY-MM-DD&end\_date=YYYY-MM-DD&api\_key=YOUR\_KEY**

#### **Step 2 : Extract Data Using NASA's Asteroid API**

### **📌Steps for Fetching NASA NEO Data**

1. Begin with a start date (**2024-01-01**) and an end date 7 days later (**2024-01-07**).
2. **Use Pagination via the 'next' Link**

* After processing the first response, get the 'next' URL from **data['links']['next']**.
* This link points to the next 7-day date range.
* Update your request URL and repeat the process.

1. **Repeat Until 10,000 Records Are Collected**

* Keep looping through the **'next'** links.
* Each response brings data for 7 days
* Append extracted fields into a list.
* Stop when your list reaches **10,000** asteroid records.

**Fields to Extract**

|  |  |  |
| --- | --- | --- |
| **s.no** | **Fields** | **Description** |
| 1 | **id** | Unique ID of the asteroid |
| 2 | **neo\_reference\_id** | NASA’s internal reference ID for the asteroid. |
| 3 | **name** | Official name or designation of the asteroid. |
| 4 | **absolute\_magnitude\_h** | A measure of the asteroid’s brightness |
| 5 | **estimated\_diameter\_min\_km** | Minimum estimated diameter in kilometers |
| 6 | **estimated\_diameter\_max\_km** | Maximum estimated diameter in kilometers |
| 7 | **is\_potentially\_hazardous\_asteroid** | Indicates if the asteroid is considered hazardous (True/False). |
| 8 | **close\_approach\_date** | Date of the asteroid's closest approach. |
| 9 | **relative\_velocity\_kmph** | Speed of the asteroid during approach (in km/h) |
| 10 | **astronomical** | average distance between the Earth and the Sun |
| 11 | **miss\_distance\_km** | Distance by which the asteroid missed Earth (in km). |
| 12 | **miss\_distance\_lunar** | Distance in lunar units (Earth–Moon distances) |
| 13 | **orbiting\_body** | The celestial body the asteroid was orbiting (usually "Earth"). |

🔁 **Note:**

*The dataset includes multiple records for the same asteroid because each entry corresponds to a distinct close approach event.*

### **Step 3 :🧹 Data Cleaning Steps**

1. Extract only the required fields from the JSON response.
2. Ensure all values are properly formatted (e.g., convert numbers stored as strings to int or float).
3. Convert date strings (e.g., "2024-07-15") to proper date objects using Python’s datetime.strptime() method.
4. Handle any missing or null values by setting defaults or skipping incomplete records.(**.get()** method)
5. Prepare the cleaned data for SQL insertion by ensuring it **matches the correct data types** and structure.

**Step 4: Insert NASA Asteroid Data into SQL**

### **✅ SQL Insertion Steps (via Python)**

1. Create a database  
   * Use any SQL engine like **MySQL, PostgreSQL, or SQLite**
   * For MySQL/PostgreSQL: create manually in GUI or CLI
   * For SQLite: a .db file is auto-created
2. Establish a connection using Python  
   * Use mysql.connector or pymysql for MySQL
   * Use psycopg2 for PostgreSQL
   * Use sqlite3 for SQLite(colab)
3. Create a cursor object  
   * Used to **execute SQL commands from Python**
4. Create 2 tables using Python code  
   * **asteroids** — stores general asteroid details
   * **close\_approach** — stores approach event data
5. Insert the parsed data into these tables  
   * Extract data from the JSON response
   * Use INSERT queries to populate both tables

**SQL Tables info**

**🚀 Table 1: asteroids**

This table stores general information about each asteroid.

🔁 **Note:**

*The id field is not unique in your dataset.So, it* ***can’t be used as a primary key*** *directly.*

|  |  |  |
| --- | --- | --- |
| **s.no** | **Fields** | **Datatype** |
| 1 | **id** | INT |
| 2 | **name** | VARCHAR |
| 3 | **absolute\_magnitude\_h** | FLOAT |
| 4 | **estimated\_diameter\_min\_km** | FLOAT |
| 5 | **estimated\_diameter\_max\_km** | FLOAT |
| 6 | **is\_potentially\_hazardous\_asteroid** | BOOLEAN |

### **🌍 Table 2: close\_approach**

This table stores event-based data about each asteroid’s approach to Earth.

|  |  |  |
| --- | --- | --- |
| **s.no** | **Fields** | **DataType** |
| **1.** | **neo\_reference\_id(same as id in asteroids)** | INT |
| **2** | **close\_approach\_date** | DATE |
| **3** | **relative\_velocity\_kmph** | FLOAT |
| **4** | **astronomical(AU)** | FLOAT |
| **5** | **miss\_distance\_km** | FLOAT |
| **6** | **miss\_distance\_lunar** | FLOAT |
| **7** | **orbiting\_body** | VARCHAR |

**🔍 Step 5 : SQL Queries (Display the output in streamlit)**

1. Count how many times each asteroid has approached Earth
2. Average velocity of each asteroid over multiple approaches
3. List top 10 fastest asteroids
4. Find potentially hazardous asteroids that have approached Earth more than 3 times
5. Find the month with the most asteroid approaches
6. Get the asteroid with the fastest ever approach speed
7. Sort asteroids by maximum estimated diameter (descending)
8. An asteroid whose closest approach is getting nearer over time(Hint: Use ORDER BY close\_approach\_date and look at miss\_distance).
9. Display the name of each asteroid along with the date and miss distance of its closest approach to Earth.
10. List names of asteroids that approached Earth with velocity > 50,000 km/h
11. Count how many approaches happened per month
12. Find asteroid with the highest brightness (lowest magnitude value)
13. Get number of hazardous vs non-hazardous asteroids
14. Find asteroids that passed closer than the Moon (lesser than 1 LD), along with their close approach date and distance.
15. Find asteroids that came within 0.05 AU(astronomical distance)

**Along with answering the 15 given questions, learners are encouraged to frame and execute their own SQL queries(5 to 10) to derive additional insights.**

**Step 5: Streamlit UI**

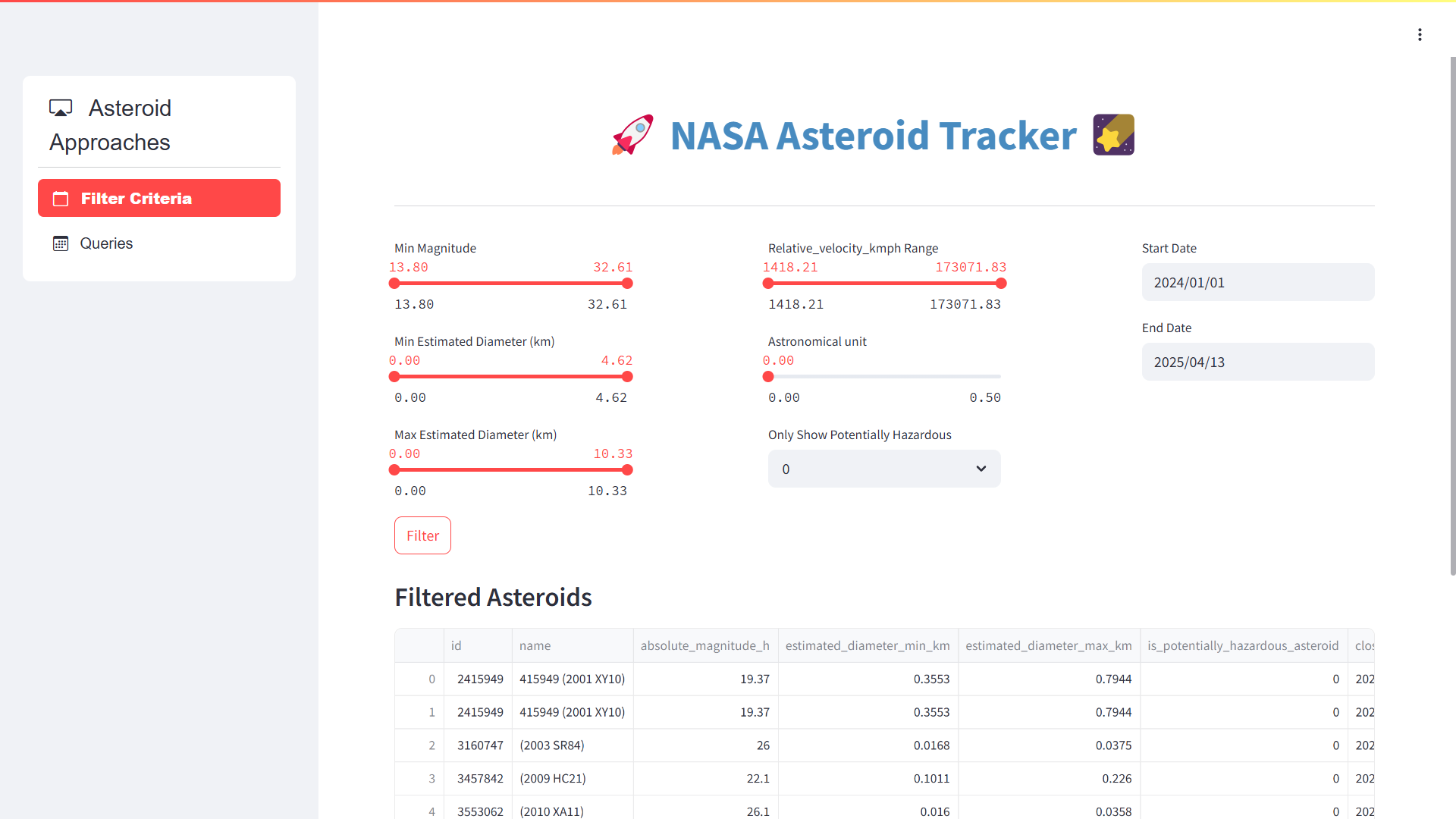
1. Set up Streamlit environment  
    Install required packages and create a Python script
2. Establish connection to the SQL database
3. Create a sidebar dropdown

* Let users select from predefined queries (15 sql queries)
* Display results of the selected query in the main area
* Design main page filters  
  Write a single SQL query which filters the data according to the filter criteria selected by the users.
  + Add input widgets like sliders, date pickers, or number inputs
  + Filters based on:
    - Close approach date
    - Astronomical Units
    - Lunar distances
    - Relative velocity
    - Estimated diameter (min & max)
    - Hazardous state
  + To add these filters in your Streamlit app, you can use:
    - st.date\_input()
    - st.slider()
    - st.selectbox() etc..
  + Display the filtered results as a table

### 

### 

### **Sample Streamlit UI**



### 

### **🛠 Technical Tags**

**API Integration, JSON Parsing, Python, SQL, Data Extraction, Data Transformation, Data Cleaning, Database Insertion, SQL Joins, Streamlit, Data Filtering, Pagination, Astroinformatics, Space Data Analysis, Query Optimization, Scientific Insights**

### **📊 Expected Results**

**✅** Extraction of 10,000 asteroid records using NASA API with proper pagination.

✅ Transformed JSON data with only the required fields for analysis.

✅ Automated SQL table creation and data insertion using Python (**no manual SQL operations**).

✅ A fully functional Streamlit dashboard

### **⏳ Timeline**

The project should be completed and submitted **within 14 days** from the date it is assigned.

|  |  |  |
| --- | --- | --- |
| **Created By** | **Verified By** | **Approved By** |
| Nilofer Mubeen | Shadiya | Nehlath Harmain |

**PROJECT DOUBT CLARIFICATION SESSION ( PROJECT AND CLASS DOUBTS)**

**About Session:** The Project Doubt Clarification Session is a helpful resource for resolving questions and concerns about projects and class topics. It provides support in understanding project requirements, addressing code issues, and clarifying class concepts. The session aims to enhance comprehension and provide guidance to overcome challenges effectively.

**Note: Book the slot at least before 12:00 Pm on the same day**

**Timing: Monday-Saturday (3:30PM to 4:30PM)**