Research Methods in Data Science

COURSE INSTRUCTOR
(DR. Mohammed K Halawani)

SUBMITTED BY:

Anas Mohammed Alsubhi
Mohammed Khaled Altufayhi
Nawaf Abdulrhman Alageel
Albadar Ibrahim Almaymani
Abdullah Mansour Habit

DEPARTMENT OF (Data Science)
COLLEGE OF COMPUTING
UMM AL-QURA UNIVERSITY

2024

Introduction

Big data analytics has become crucial in understanding and interpreting vast amounts of data generated across various domains. The ability to efficiently collect, clean, and analyze this data is essential for gaining insights and making informed decisions. One such area where big data plays a vital role is the study of near-Earth objects (NEOs). These celestial bodies, which come close to Earth's orbit, pose potential risks and offer opportunities for scientific research. In this research, we focus on automating the collection and preparation of big data related to NEOs using NASA APIs. By developing an automated, reusable data collection methodology, we aim to streamline the data acquisition process, making it efficient and scalable for future analysis and machine learning applications.

Background

NASA's APIs provide a wealth of data on various space-related phenomena, including information on NEOs. These data are invaluable for scientific research, risk assessment, and educational purposes. However, manually collecting and processing such large datasets is labor-intensive and prone to errors. Automated data collection addresses these challenges by ensuring consistent, accurate, and timely acquisition of data.

Near-Earth objects are small solar system bodies whose orbits bring them into close proximity to Earth. Monitoring and analyzing NEOs is essential for several reasons. Firstly, it helps in assessing the potential impact risks these objects might pose to Earth. Secondly, studying NEOs provides insights into the early solar system's conditions and the processes that have shaped it. Finally, NEOs can serve as potential targets for future space missions aimed at resource extraction or exploration.

The NASA NEO Feed API is a valuable tool that provides detailed information on NEOs, including their orbits, sizes, and potential close approaches to Earth. By leveraging this API, we can automate the data collection process, ensuring that we have up-to-date and comprehensive datasets for analysis.

Methodologies for Data Collection

API Selection and Integration:

We chose the NASA NEO Feed API due to its comprehensive coverage of near-Earth objects, including detailed information on their characteristics and orbits.

Automated Data Collection and Reusability:

The following Python code snippets demonstrate how the data collection process is automated and reusable:

```python
import requests

def get_neo_data(api_key, start_date, end_date):
    base_url = "https://api.nasa.gov/neo/rest/v1/feed"

    params = {
        'api_key': api_key,
        'start_date': start_date,
        'end_date': end_date
    }

    response = requests.get(base_url, params=params)

    if response.status_code == 200:
        data = response.json()
        return data['near_earth_objects']
    else:
        print(f"Error: {response.status_code}")
        return None
```

This function encapsulates the logic to interact with the NASA API, making it reusable for different date ranges and API keys. It handles API requests and returns the data in JSON format.

Loop to Automate Data Collection

```python
from datetime import datetime, timedelta

api_key = "your_api_key"

start_date = datetime.strptime("2010-01-01", "%Y-%m-%d")
end_date = datetime.strptime("2024-01-31", "%Y-%m-%d")

all_neo_data = {}
total_records_fetched = 0
max_records = 500000
days_per_request = 7

while total_records_fetched < max_records:
    next_end_date = start_date + timedelta(days=days_per_request)
    if next_end_date > end_date:
        next_end_date = end_date

    neo_data = get_neo_data(api_key, start_date.strftime("%Y-%m-%d"), next_end_date.strftime("%Y-%m-%d"))

    if neo_data:
        for date in neo_data:
            if date not in all_neo_data:
                all_neo_data[date] = []
            all_neo_data[date].extend(neo_data[date])
            total_records_fetched += len(neo_data[date])

        print(f"Fetched {total_records_fetched} records so far.")

        if total_records_fetched >= max_records:
            break

    start_date = next_end_date + timedelta(days=1)
```

This loop automates the data collection process by fetching data in chunks (7-day intervals). It ensures that data collection continues until the specified maximum number of records is reached, making the process scalable and efficient.

Function to Save Data to CSV

```python
import csv

def save_to_csv(data, filename="neo_data.csv"):
    if not data:
        print("No data to save.")
        return

    with open(filename, mode='w', newline='') as file:
        writer = csv.writer(file)

        # Write the header
        keys = set()
        for date in data:
            for neo in data[date]:
                keys.update(neo.keys())
        writer.writerow(keys)

        # Write the data rows
        for date in data:
            for neo in data[date]:
                writer.writerow([neo.get(key, '') for key in keys])

    print(f"Data saved to {filename}")
```

This function saves the collected NEO data to a CSV file. By parameterizing the filename and dynamically generating the header from the data, it ensures reusability and adaptability to different datasets.

Data Cleaning and Preparation

```python
import pandas as pd

# Load the collected data
neo_df = pd.read_csv('neo_data.csv')

# Remove duplicates
neo_df.drop_duplicates(inplace=True)

# Handle missing values by forward filling
neo_df.fillna(method='ffill', inplace=True)

# Convert date columns to datetime objects
neo_df['close_approach_date'] = pd.to_datetime(neo_df['close_approach_data_close_approach_date'])

# Normalize column names for consistency
neo_df.columns = [col.replace('.', '_') for col in neo_df.columns]

# Extract relevant columns for analysis
columns_of_interest = [
    'name', 'id', 'absolute_magnitude_h', 'estimated_diameter_meters_min',
    'estimated_diameter_meters_max', 'is_potentially_hazardous_asteroid',
    'close_approach_date', 'relative_velocity_kilometers_per_hour',
    'miss_distance_kilometers', 'orbiting_body'
]
neo_df = neo_df[columns_of_interest]

# Handle outliers in numerical columns
for col in ['absolute_magnitude_h', 'estimated_diameter_meters_min', 'estimated_diameter_meters_max']:
    neo_df[col] = neo_df[col].apply(lambda x: x if x > 0 else None)
neo_df.dropna(inplace=True)

# Save the cleaned data
neo_df.to_csv('cleaned_neo_data.csv',index=False)
```

Remove Duplicates: Ensures that each record is unique.
Handle Missing Values: Uses forward filling to propagate the last valid observation forward.
Convert Date Columns: Ensures that date columns are in a consistent datetime format for easier analysis.
Normalize Column Names: Replaces periods in column names with underscores for consistency and ease of use in downstream processes.
Extract Relevant Columns: Focuses on columns that are crucial for analysis.
Handle Outliers: Ensures that numerical columns contain only valid positive values and removes rows with invalid values.

Types of Data Collected

The code collects data on near-Earth objects (NEOs) from the NASA NEO Feed API. This data includes detailed information about each NEO, such as its physical characteristics, orbital parameters, and close-approach details. Here are the key types of data collected by this code:

Name and ID:

name: The name of the NEO.
id: A unique identifier for the NEO.

Physical Characteristics:

absolute_magnitude_h: The absolute magnitude (brightness) of the NEO.
estimated_diameter_meters_min: The estimated minimum diameter of the NEO in meters.
estimated_diameter_meters_max: The estimated maximum diameter of the NEO in meters.
is_potentially_hazardous_asteroid: A boolean indicating whether the NEO is considered potentially hazardous.

Orbital Data and Close Approach Information:

close_approach_date: The date of the NEO's closest approach to Earth.
relative_velocity_kilometers_per_hour: The relative velocity of the NEO at the time of close approach, in kilometers per hour.
miss_distance_kilometers: The distance at which the NEO will miss Earth during its closest approach, in kilometers.
orbiting_body: The primary body that the NEO is orbiting (usually the Sun).

Potential Uses of the Data

The collected and cleaned NEO data can be utilized in various applications:

1. **Scientific Research**: Researchers can study the characteristics, orbits, and potential risks of near-Earth objects, contributing to our understanding of these celestial bodies.

2. **Risk Assessment**: Governments and space agencies can use the data to assess the potential threats posed by NEOs to Earth and develop strategies to mitigate those risks.

3. **Machine Learning**: The data can be used to train machine learning models to predict the trajectories and potential impacts of NEOs, enhancing our predictive capabilities.

4. **Educational Purposes**: Educators can use the data to teach students about space science, data analysis, and the importance of monitoring near-Earth objects.

5. **Public Awareness**: The data can be shared with the public to raise awareness about NEOs and the measures being taken to monitor and protect Earth from potential impacts.

Conclusion

This research demonstrates the effectiveness of using the NASA NEO Feed API for automated and reusable data collection. The methodology ensures scalability and consistency, enabling efficient handling of big data challenges. The automated scripts for data collection and cleaning provide a robust and reusable framework, ensuring that data is continuously updated and prepared for analysis. This approach not only saves time and effort but also improves data accuracy and reliability. The collected and cleaned data is prepared for further analysis and machine learning applications, providing a solid foundation for ongoing research in the field of near-Earth objects.