Near Earth Objects JSON Analysis

Introduction

This report uses JavaScript and Node to analyse data from NASA about Near Earth Objects (NEOs) in five separate steps. The JSON data contains 202 objects with information about the properties of each NEO including the observed magnitude (h_mag), the minimum orbit intersection distance (MOID), the perihelion distance (q), the aphelion distance (Q), the orbital period (period_yr), the orbital inclination (i_deg), whether the NEO is a potentially hazardous asteroid (PHA) and the orbit class. This report details how the data was loaded, analysed using functions, reformatted, saved and tested for performance.

Step 1: Loading JSON data

After downloading the data, the file was read into a variable in Node.js using the code provided, declaring "data" as a global variable. This file was then capsuled within a function that parsed the data into a JavaScript object using "JSON.parse".

Step 2: Basic functions

A total of four functions were used to complete the programming exercises in step 2. These functions are used to retrieve the properties of a specific NEO to help with data analysis. The "findNeoByIndex" function retrieves an individual NEO based on the index. It takes the index as the argument and stores the JSON data as a variable locally. It utilises an "if" and "else" conditional statement. The "filterByOrbitClass" function returns a collection of NEOs according to the class using ".filter". It accounts for classes that have an asterisk by using the "or" logical operator. The "filterByPha" function also returns a collection of NEOs but according to whether they are a PHA. This is declared as a constant to be accessible for future functions. Finally the "findNeoByDesignation" function retrieves an individual NEO based on the designation. It takes the designation as the argument and stores the JSON data as a variable locally. It uses a "for" loop and "if" statement to iterate through each NEO until the designation is found before returning the results. "console.table" can be used to log all the results of these functions in a format that is easy to identify each property. Some initial functions were inefficient or faulty and replaced by these newer functions. The first designation function did not work so a loop was used instead. The display functions were also replaced so data comparison was easier.

Step 3: Analysis

To determine the characteristics of NEOs of certain types, a data analysis compared all of the orbital properties of NEOs of different types. To do this analysis, new functions were created taking arguments that were the result of functions in step 2 (eg. "phaNeos", "apolloNeos"). New function, "filterByPhaAndOrbitClass", accepts both PHA and class type as arguments with ".filter" to allow for better data comparison. The "displayAllNeos" function uses an "if" statement and "for" loop to return all NEOs. When displayed using "console.table" it gives a quick overall view and side by side comparison. In order to calculate the maximum, minimum and average values for the orbits of different NEOs, three different functions were created. "extractPhaPropertyValues" and "extractClassPropertyValues" both return the orbital values for NEOs of a certain type using "for" loops. "calculateStats" then takes these values and returns the maximum, minimum and average values for each orbital value using “for” loops and “if” statements.

The original functions to display all NEOs or certain types had nice headings but were replaced with better formatting for data comparison. The original function to calculate maximums, minimums and averages of orbits involved too many subsets of functions and was cumbersome to use for each type of NEO, thus was replaced with a more efficient function.

Being able to identify PHAs is one of the most important uses of this dataset. All PHA orbit properties apart from the observed magnitude, were on average smaller than non-PHAs. PHAs are classified based on how close they can potentially get to Earth; h_mag ≤ 22.0 and MOID ≤ 0.05AU (Center for Near-Earth Object Studies, n.d.). In the observed data, these matched the given definition (h_mag: 18 - 21.9, MOID: 0.05 - 0.0002AU). While the other orbit values may be important features of PHAs, they are not defining characteristics. In the data, Apollos are the most likely asteroid class to be PHAs whereas Amors are further from Earth’s orbit so are less likely to be PHAs.

The ability to identify different classes of asteroids and comets is also important when considering the data. According to the CNEOS. (n.d.), asteroids are classed based on their perihelion, aphelion and semi major axis (a)[1]. The Apollos orbits (q: 0.14 - 1.01AU, Q: 1.08 - 9.89AU) correspond to the given definition of Apollos by CNEOS. (n.d.), which defines them as having an Earth crossing orbit of a > 1AU and q < 1.017AU. On average, Amors have the lowest observed magnitude, largest MOID, longest perihelion distance (1.02 - 1.3AU), longest aphelion distance (1.45 - 41.78AU), longest orbital period and largest orbital inclination of the three classes of asteroids in the data. Again the findings correspond to the given definition of an asteroid that does not cross the Earth’s orbit with a > 1AU and 1.017 < q < 1.3AU (CNEOS, n.d.). Atens, on average, have the largest observed magnitude, smallest MOID, shortest perihelion distance (0.31 - 0.85AU), shortest aphelion distance (1.04 - 1.6AU), shortest orbital period and smallest orbital inclination compared to other asteroid classes in the data. This supports CNEOS. (n.d.) definition of an asteroid with an Earth crossing orbit with a < 1AU and Q > 0.983AU.

Comets[2] are defined based on their orbital periods and orbital inclinations (Wikipedia, n.d.). The single Encke-type Comet in the data had the lowest aphelion distance, shortest orbital period (5.49 years) and smallest orbital inclination on average which matches the definition given by NASA. (n.d.), which states these comets have the shortest orbital periods. The Jupiter-family Comets data confirmed they have shorter orbital periods (< 20 years) and lower orbital inclinations (≤ 30 degrees) compared to Halley-type Comets (NASA, n.d., Cosmos, n.d., Wikipedia, n.d.). Halley-type Comets in the data have the lowest MOID and perihelion compared to other comets. The orbital periods (23.19 - 23.56 years) and orbital inclinations (38.69 - 147.05 degrees) found match the given definitions according to Cosmos. (n.d.) and Wikipedia. (n.d.) of having orbital periods between 20-200 years and orbital inclinations from 0 to more than 90

[1] Semi major axis are not included in the dataset
[2] No recorded observed magnitudes for any comets in the data

degrees. Parabolic comets had the highest orbital inclinations on average (≥ 96.48 degrees) however it is not defined by this. It is known to have a long orbital period ≥1000 yrs which is not recorded in the data (Wikipedia., n.d.). Undefined comets in the data have the highest MOID, perihelion distance, aphelion distance and orbital period

Step 4: Changing JSON format
To rearrange the original data into arrays sorted by NEO class type a new function was created. “returnClassArrays” utilised the “filterByOrbitClass” function to extract arrays for each different class type. These arrays were then declared as a new constant “newJSON” which was parsed into a JSON string using “JSON.stringify”. Finally the data was saved into a new file called “NEOWISE_classes.json” using “fs.writeFileSync”. Although the data first appears disorganised in the new file, this was fixed by utilising the “format document” option (option + shift + F on Mac), displaying each class array neatly.

Step 5: Unit tests
Unit tests using the JavaScript testing framework Jest were created to test four of the functions created during the programming exercises. The first three functions, “findNeoByIndex”, “findNeoByDesignation” and "extractClassPropertyValues”, were tested with a mix of “.toBe” and “.toEqual” matchers to check for exact equality and the value of the returned objects. Valid and invalid responses were tested for the “findNeoByIndex” and “findNeoByDesignation” functions. Finally the “calculateStats” function was tested with “.toMatchObject” to check if the returned object matched a subset of properties of an object. Results from the tests are provided in figure 1.

Conclusion
This report demonstrates the practical application of programming and JavaScript fundamentals in order to analyse extensive data sets like NEOs from NASA. Although the NEO data is seemingly large and complicated, it can be broken down and sorted with simple functions and logical operations. The definitions of each NEO criteria then can be reverse engineered using data analysis. The final part of this assignment tests practical debugging skills, emphasising the importance of problem solving skills in real world scenarios.

References

Center for Near-Earth Object Studies. (CNEOS) (n.d.). *Neo Basics*. NASA Jet Propulsion Laboratory. https://cneos.jpl.nasa.gov/about/neo_groups.html

Cosmos. (n.d.). *Halley-Type Comets.* Swinburne University of Technology. https://astronomy.swin.edu.au/cosmos/h/halley-type+comets

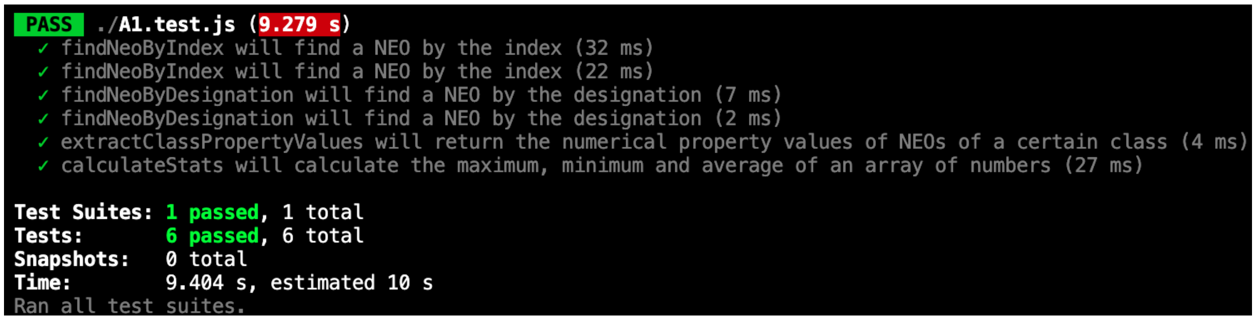
NASA. (n.d.). *2P/Encke*. Retrieved February 27, 2024, from https://science.nasa.gov/solar-system/comets/2p-encke/

NASA. (n.d.). *9P/Tempel 1*. Retrieved February 27, 2024, from https://science.nasa.gov/solar-system/comets/9p-tempel-1/

Wikipedia. (n.d.). *Comet.* Retrieved February 27, 2024, from https://en.wikipedia.org/wiki/Comet#Short_period

Wikipedia. (n.d.). *List of near-parabolic comets.* Retrieved February 27, 2024, from https://en.wikipedia.org/wiki/List_of_near-parabolic_comets

Figure 1



```
 PASS  ./A1.test.js (9.279 s)
  ✓ findNeoByIndex will find a NEO by the index (32 ms)
  ✓ findNeoByIndex will find a NEO by the index (22 ms)
  ✓ findNeoByDesignation will find a NEO by the designation (7 ms)
  ✓ findNeoByDesignation will find a NEO by the designation (2 ms)
  ✓ extractClassPropertyValues will return the numerical property values of NEOs of a certain class (4 ms)
  ✓ calculateStats will calculate the maximum, minimum and average of an array of numbers (27 ms)

Test Suites: 1 passed, 1 total
Tests:       6 passed, 6 total
Snapshots:   0 total
Time:        9.404 s, estimated 10 s
Ran all test suites.
```

Note: Results of unit testing showing six out of six tests passing