

ASTEROIDS - NEOWS

Table of Contents

- [1 Summary](#)
- [2 Requirements-and-connection](#)
- [3 ELT Process](#)
 - [3.1 Extract](#)
 - [3.2 Load](#)
 - [3.3 Transform](#)
 - [3.4 Datastructure](#)
- [4 Orbital analysis](#)
 - [4.1 Distribution of orbit class types](#)
 - [4.2 Identifying clustering features](#)
 - [4.3 Comparison of potentially hazardous asteroids](#)
 - [4.4 Analysis of absolute error in diameter estimation](#)
- [5 Conclusions](#)

Summary

In this study, we leveraged NASA's Near-Earth Object Web Service (NeoWs) API to collect numerous asteroid data. This information, structured in JSON format, was efficiently stored in a MongoDB database to facilitate a comprehensive analysis. Our analysis was guided by the use of the pymongo library, which allowed us to perform a series of operations aimed at extracting valuable information from the asteroid dataset.

Our investigation delved into multiple crucial attributes of asteroids, including estimated diameter ranges, minimum approach distances, relative velocities, absolute magnitude (H), and the assessment of potential hazards. The primary findings of our study unveiled that NASA's definition of potential hazard appears to be less conservative than the widely accepted threshold of 0.05 astronomical units (AU) or less, in conjunction with an absolute magnitude (H) of 22.0. Additionally, our analysis demonstrated an inverse power-law relationship between the absolute error in estimated diameter and the absolute magnitude (H). Moreover, we identified that parameters such as perihelion distance and semi-major axis serve as distinguishing factors for grouping various orbital classes.

These results provide valuable insights into the world of near-Earth asteroids and their characteristics, offering a deeper understanding of the underlying relationships among key attributes.

Requirements and connection

To conduct this analysis, specific packages need to be imported for the study to run effectively.

```
In [1]: # import packages
import pymongo
import pprint as pp
import pandas as pd
import requests
import matplotlib.pyplot as plt
import numpy as np
from scipy.optimize import curve_fit
from sklearn.metrics import r2_score

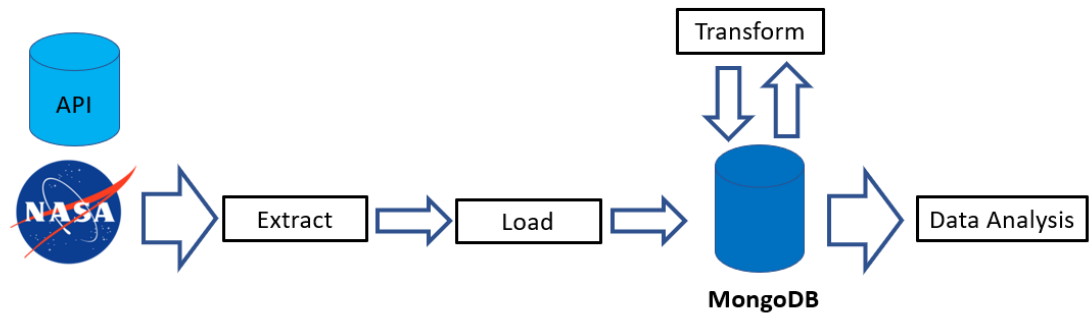
# pandas configuration
pd.set_option('display.precision', 2)
pd.set_option('max_colwidth', 50)
```

As mentioned above, we use a NoSQL database, MongoDB, to efficiently store JSON data retrieved from the API. In the following sections, we will provide a complete overview of the API specification and database configuration for this purpose.

```
In [2]: API_URL = "https://api.nasa.gov/neo/rest/v1/neo/browse?page=1&size=20&api_key=DEMO
CNX_STR = "mongodb+srv://valleandrea:nosql_andrea@cluster0.puqrqt.mongodb.net"
DB_NAME = "db_asteroids"
COLL_NAME = "asteroids"
```

ELT Process

The approach chosen for this procedure is ETL, which stands for "Extract, Load, Transform". It is a series of steps that begin with extracting the data from the API, follow by loading the data into the database, and culminate with applying the essential transformations. Next, the transformed data is retrieved from the database for further analysis. The diagram below provides a concise overview of this process.



Extract

The extraction phase begins with the creation of a database connection, making sure that the details required for the database connection are configured correctly. This step also involves verifying that a suitable collection exists within the database in which the asteroid records will be stored. If the collection does not exist, the extraction process involves creating the collection to accommodate the incoming data. This step lays the foundation for the effective storage and management of the asteroid dataset, enabling subsequent data retrieval and analysis.

```
In [3]: # Connetion to the database
client = pymongo.MongoClient(CNX_STR)
db = client["asteroids_db"]
```

```
In [4]: # Specify a collection or create it if it does not already exist
collection_name = "asteroids"

if collection_name in db.list_collection_names():
    col = db.get_collection(collection_name)
else:
    col = db.create_collection[collection_name]

# Clean up the collection
col.delete_many({})
```

```
Out[4]: <pymongo.results.DeleteResult at 0x1e05c6c5dc0>
```

Subsequently, we extracted data from the API, with each API request providing information on 20 asteroids. To optimize our data retrieval process, we had the flexibility to control the number of asteroids retrieved by specifying the number of batches. In this specific instance, we chose to limit our data retrieval to 10 batches, ensuring an efficient and manageable dataset for our analysis.

```
In [5]: # Fetcher for the API
def fetcher(data):
    asteroids = {
        'id': int(data['id']),
```

```

        'name': data['name'],
        'nasa_jpl_url': data['nasa_jpl_url'],
        'absolute_magnitude_h': data['absolute_magnitude_h'],
        'estimated_diameter': data['estimated_diameter'],
        'is_potentially_hazardous_asteroid':
            data['is_potentially_hazardous_asteroid'],
        'close_approach_data': data['close_approach_data'],
        'orbital_data': data['orbital_data'],
        'is_sentry_object': data['is_sentry_object']
    }

    return asteroids

```

```

In [6]: # Fetch from API
near_earth_objects = []

number_batches = 10

for i in range(number_batches):
    url = f"https://api.nasa.gov/neo/rest/v1/neo/browse?page={i}&size=20&api_key="
    r = requests.get(url)
    data = r.json()
    near_earth_objects.extend(data['near_earth_objects'])

list_asteroids = []
for object in near_earth_objects:
    list_asteroids.append(fetcher(object))

```

Load

The data were efficiently loaded into the default collection, where each asteroid was recorded as an individual document.

```

In [7]: # Insert into the collection
col.insert_many(list_asteroids)

# Count elements in the collection
document_count = col.count_documents({})
print(f'Number of documents in the collection: {document_count}')

```

Number of documents in the collection: 200

An initial dataset sample is printed to offer an initial glimpse of its structure.

```

In [24]: # Printing a sample of a document in the collection
projection = {
    "_id": 1,
    "id": 1,
    "name": 1,
    "nasa_jpl_url": 1,
    "absolute_magnitude_h": 1,
    "estimated_diameter": 1,

```

```

        "is_potentially_hazardous_asteroid": 1,
        "is_sentry_object": 1
    }
}
pp.pprint(list(col.aggregate([{"$limit":1}, {"$project": projection}])))

[{'_id': ObjectId('654217cc5ec6259d1ed0bf74'),
  'absolute_magnitude_h': 10.41,
  'estimated_diameter': {'feet': {'estimated_diameter_max': 161445.1600989368,
                                   'estimated_diameter_min': 72200.4705239119},
                          'kilometers': {'estimated_diameter_max': 49.2084832235,
                                           'estimated_diameter_min': 22.0067027115},
                          'meters': {'estimated_diameter_max': 49208.4832234845,
                                       'estimated_diameter_min': 22006.7027114738},
                          'miles': {'estimated_diameter_max': 30.5767244291,
                                      'estimated_diameter_min': 13.6743268705}},
  'id': 2000433,
  'is_potentially_hazardous_asteroid': False,
  'is_sentry_object': False,
  'name': '433 Eros (A898 PA)',
  'nasa_jpl_url': 'http://ssd.jpl.nasa.gov/sbdb.cgi?sstr=2000433'}]

```

Transform

In the transformation step, we extract specific features that are essential for our subsequent analyses. One of the initial steps in this phase involves calculating the average relative velocity, a critical metric derived from available observational data. This average relative velocity provides valuable insights into the dynamics of these celestial objects and serves as a key parameter for further investigation.

```

In [9]: # Extract the average relative speed based on the observations

pipeline = [
    {
        "$unwind": "$close_approach_data"
    },
    {
        "$addFields": {
            "relativeSpeed": {
                "$toDouble":
                    "$close_approach_data.relative_velocity.kilometers_per_second"
            }
        }
    },
    {
        "$group": {
            "_id": "$_id",
            "relativeSpeedAvg": { "$avg": "$relativeSpeed" }
        }
    },
    {
        "$project": {
            "_id": 1,
            "relativeSpeedAvg": 1
        }
    }
]

```

```

    }
  }
]

result = col.aggregate(pipeline)
for doc in result:
    id = doc["_id"]
    relative_speed_avg = doc["relativeSpeedAvg"]
    col.update_one({"_id": id}, {"$set": {"relativeSpeedAvg": relative_speed_avg}}]

pipeline = [
    {"$limit": 5},
    {
        "$project": {
            "_id": 1,
            "id": 1,
            "name": 1,
            "relativeSpeedAvg": 1
        }
    }
]
r = col.aggregate(pipeline)
pd.DataFrame(r)

```

Out[9]:

	_id	id	name	relativeSpeedAvg
0	654217cc5ec6259d1ed0bf74	2000433	433 Eros (A898 PA)	5.06
1	654217cc5ec6259d1ed0bf75	2000719	719 Albert (A911 TB)	6.20
2	654217cc5ec6259d1ed0bf76	2000887	887 Alinda (A918 AA)	10.09
3	654217cc5ec6259d1ed0bf77	2001036	1036 Ganymed (A924 UB)	13.98
4	654217cc5ec6259d1ed0bf78	2001221	1221 Amor (1932 EA1)	10.46

Another crucial feature we extract during the transformation phase is the minimum missing distance, expressed in astronomical units (AU). This parameter provides essential information about the approach of an asteroid to our planet during its orbit. By quantifying the minimum miss distance in astronomical units, we gain a more intuitive understanding of the potential risks associated with these celestial objects.

In [10]: *# Extract the minimum distance based on the observations*

```

pipeline = [
    {
        "$unwind": "$close_approach_data"
    },
    {
        "$addFields": {
            "missDistance": {
                "$toDouble": "$close_approach_data.miss_distance.astronomical"
            }
        }
    }
],

```

```

    {
        "$group": {
            "_id": "$_id",
            "missDistance_min": { "$min": "$missDistance" }
        }
    },
    {
        "$project": {
            "_id": 1,
            "missDistance_min": 1
        }
    }
]

result = col.aggregate(pipeline)
for doc in result:
    unique_id = doc["_id"]
    col.update_one({"_id": unique_id},
                   {"$set": {"missDistance_min": doc["missDistance_min"]}})

pipeline = [
    {"$limit": 5},
    {
        "$project": {
            "_id": 1,
            "id": 1,
            "name": 1,
            "missDistance_min": 1
        }
    }
]
r = col.aggregate(pipeline)
pd.DataFrame(r)

```

Out[10]:

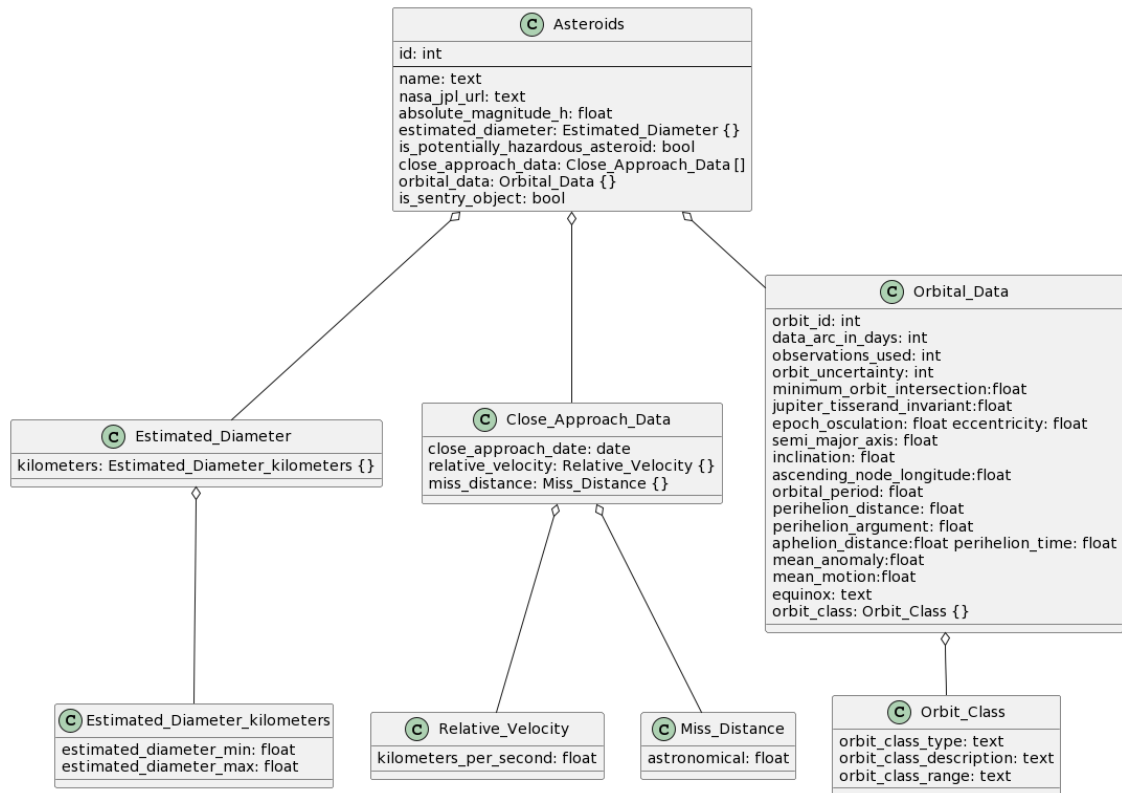
	_id	id	name	missDistance_min
0	654217cc5ec6259d1ed0bf74	2000433	433 Eros (A898 PA)	0.15
1	654217cc5ec6259d1ed0bf75	2000719	719 Albert (A911 TB)	0.21
2	654217cc5ec6259d1ed0bf76	2000887	887 Alinda (A918 AA)	0.08
3	654217cc5ec6259d1ed0bf77	2001036	1036 Ganymed (A924 UB)	0.03
4	654217cc5ec6259d1ed0bf78	2001221	1221 Amor (1932 EA1)	0.11

Datastructure

The structure of documents within the database has a hierarchical structure, consisting of JSON objects and nested lists. At the highest level is the "Asteroids" class, which serves as a general container for all asteroid-related information. Just below this top level, we find three key sublayers, "Estimated Diameter", "Approach Data" and "Orbital Data". These

sublevels, in turn, house other nested JSON objects, providing a structured organization of the data.

It is critical to note that the diagram presented here illustrates only the most pertinent features, chosen specifically for our study



Orbital analysis

Distribution of orbit class types

The examination of asteroid distribution according to orbit classes has revealed the presence of three primary categories:

- AMO: Amor orbits.
- APO: Apollo orbits.
- ATE: Atira orbits.

The data underscores that the majority of asteroids are distributed among the AMO and APO classes, providing valuable insights into the prevalence of these orbit classes among near-Earth asteroids.

```
In [11]: dist_orbits = col.aggregate([
        '$group': {
```



```

        '_id': '$orbital_data.orbit_class.orbit_class_type',
        'count': {'$sum': 1}
    }
    })
pd.DataFrame(dist_orbits)

```

```

Out[11]:
   _id  count
0  AMO     93
1  APO     98
2  ATE      9

```

Identifying clustering features

This section explores the essential characteristics required for asteroid grouping, with special attention to their orbit classes. The results of the analysis reveal that the inclusion of the semi-major axis and distance to perihelion as parameters is adequate for the grouping process. These two attributes offer valuable insights for distinguishing different groups of asteroids based on their orbital characteristics. In addition, the size of the data points in our visualizations is related to absolute magnitude (H), providing viewers with an additional dimension of information, where larger points correspond to higher absolute magnitudes and smaller points to lower values. This visual representation helps to understand the relationships between asteroid features.

```

In [17]: pipeline = [
    {
        "$project": {
            "absolute_magnitude_h": 1,
            "orbital_data.perihelion_distance": 1,
            "orbital_data.semi_major_axis": 1,
            "orbital_data.orbit_class.orbit_class_type": 1,
        }
    }
]

cursor = col.aggregate(pipeline)

data = []

for entry in cursor:
    orbital_data = entry.get("orbital_data")
    absolute_magnitude_h = entry.get("absolute_magnitude_h")

    if orbital_data:
        perihelion_distance = orbital_data.get("perihelion_distance")
        semi_major_axis = orbital_data.get("semi_major_axis")
        orbit_class = orbital_data.get("orbit_class")

        if perihelion_distance and semi_major_axis and orbit_class:

```

```

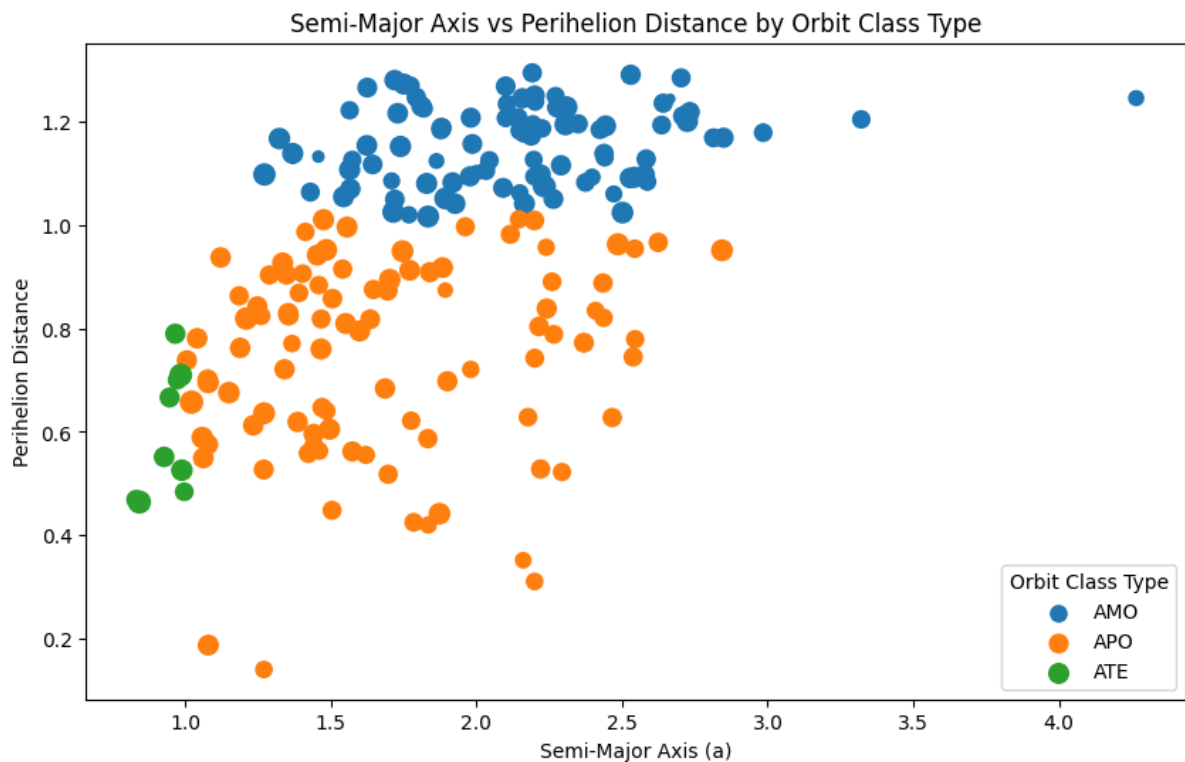
        perihelion_distance = float(perihelion_distance)
        semi_major_axis = float(semi_major_axis)
        absolute_magnitude_h = float(absolute_magnitude_h)
        orbit_class_type = orbit_class.get("orbit_class_type")
        data.append({
            "perihelion_distance": perihelion_distance,
            "semi_major_axis": semi_major_axis,
            "orbit_class_type": orbit_class_type,
            "absolute_magnitude_h": absolute_magnitude_h
        })

df = pd.DataFrame(data)
min_h = df["absolute_magnitude_h"].min()
max_h = df["absolute_magnitude_h"].max()
df["normalized_absolute_magnitude_h"] = ((df["absolute_magnitude_h"] - min_h)
                                          / (max_h - min_h))

plt.figure(figsize=(10, 6))
for orbit_class_type, group in df.groupby("orbit_class_type"):
    marker_size = 20 + 100 * group["normalized_absolute_magnitude_h"]
    plt.scatter(
        group["semi_major_axis"],
        group["perihelion_distance"],
        label=orbit_class_type,
        s=marker_size,
    )

plt.xlabel('Semi-Major Axis (a)')
plt.ylabel('Perihelion Distance')
plt.title('Semi-Major Axis vs Perihelion Distance by Orbit Class Type')
plt.legend(title='Orbit Class Type')
plt.show()

```



Comparison of potentially hazardous asteroids

In this section, we conduct a comprehensive comparison of the criteria established by NASA's Near-Earth Object Web Service (NeoWs) to identify potentially hazardous asteroids (PHAs). These criteria include two basic parameters: minimum distance and absolute magnitude (H). NASA's definition of PHA includes a MOID of 0.05 astronomical units (AU) or less and an absolute magnitude (H) not exceeding 22.0.

Our analysis of the data set, particularly in relation to the "is_potentially_hazardous_asteroid" attribute, yields a noteworthy revelation. It is apparent that NASA's standards for identifying potentially hazardous asteroids are considerably less conservative than the widely accepted and commonly used PHA criteria.

```
In [13]: pipeline = [
    {
        "$match": {
            "absolute_magnitude_h": { "$lt": 22 },
            "missDistance_min": { "$lt": 0.05 }
        }
    },
    {
        "$count": "count"
    }
]

result = next(col.aggregate(pipeline), {"count": 0})
count = result["count"]
print("The number of dangerous asteroids from PHA standards:", count)
```

The number of dangerous asteroids from PHA standards: 84

```
In [14]: pipeline = [
    {
        "$match": {
            "is_potentially_hazardous_asteroid": True,
        }
    },
    {
        "$count": "count"
    }
]

result = next(col.aggregate(pipeline), {"count": 0})
count = result["count"]
print("The number of dangerous asteroids from Nasa:", count)
```

The number of dangerous asteroids from Nasa: 48

Analysis of absolute error in diameter estimation

In this section, we investigate the intriguing relationship between the absolute error in asteroid diameter estimation and the absolute magnitude (H) of the asteroid. This relationship reveals a characteristic trend, reminiscent of an inverse power law, indicating that as the absolute magnitude (H) changes, the absolute error in diameter estimation follows a discernible trend.

In addition, our analysis extends to examining the residuals derived from the curve-fitting process. The exceptionally high R-squared value of 0.99 underscores the robustness and accuracy of the model. This strong correlation, particularly evident in the range of absolute magnitudes between 12 and 22, underscores the model's accuracy in capturing variations in diameter estimation error.

```
In [15]: pipeline = [
    {
        "$addFields": {
            "diameter_error": {
                "$subtract": [
                    "$estimated_diameter.kilometers.estimated_diameter_max",
                    "$estimated_diameter.kilometers.estimated_diameter_min"
                ]
            }
        }
    }
]

result = col.aggregate(pipeline)

# Extract data for plotting
absolute_magnitudes = []
diameter_errors = []

for doc in result:
    absolute_magnitudes.append(doc["absolute_magnitude_h"])
    diameter_errors.append(doc["diameter_error"])

# Define the inverse power-law function
def inverse_power_law(x, a, b):
    return a * np.power(x, -b)

# Sort data for fitting
data_sorted = sorted(zip(absolute_magnitudes, diameter_errors))
x_data_sorted, y_data_sorted = zip(*data_sorted)

# Perform the curve fit
params, covariance = curve_fit(inverse_power_law, x_data_sorted, y_data_sorted)
a_opt, b_opt = params

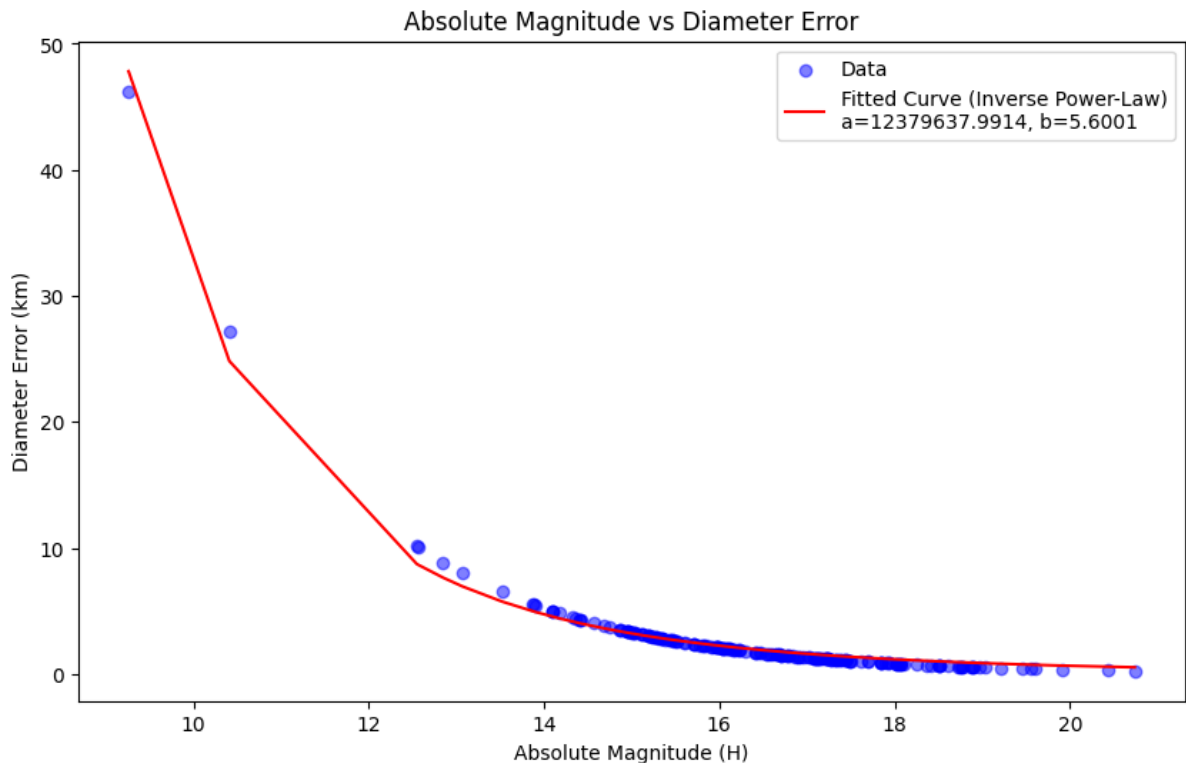
y_fit_power_law = inverse_power_law(x_data_sorted, a_opt, b_opt)
```

```
plt.figure(figsize=(10, 6))

# Scatter plot for data points
plt.scatter(absolute_magnitudes, diameter_errors, c='blue', alpha=0.5, label='Data')

# Fitted curve (Inverse Power-Law)
plt.plot(x_data_sorted, y_fit_power_law, 'r-',
         label=f'Fitted Curve (Inverse Power-Law)\na={a_opt:.1f}, b={b_opt:.1f}')

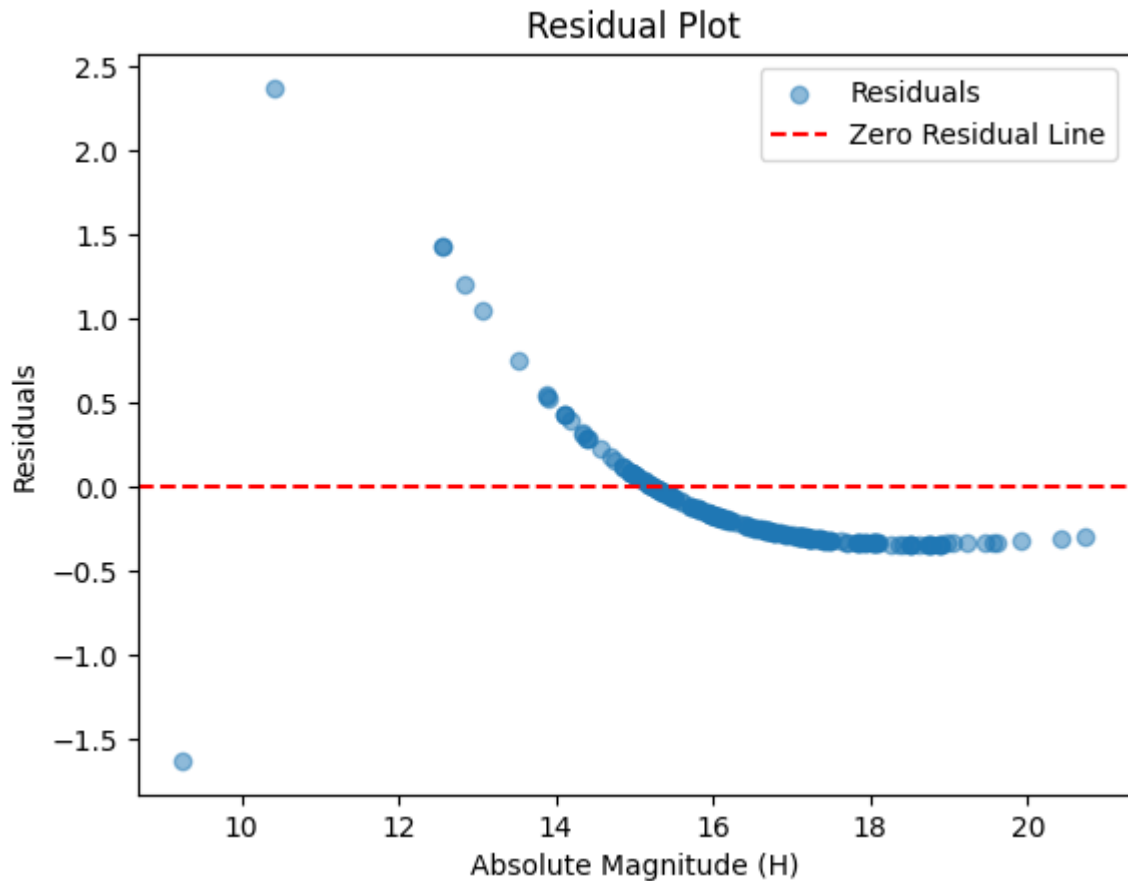
plt.xlabel('Absolute Magnitude (H)')
plt.ylabel('Absolute Diameter Error (km)')
plt.title('Absolute Magnitude vs Absolute Diameter Error')
plt.legend()
plt.show()
```



```
In [16]: r_squared = r2_score(y_data_sorted, y_fit_power_law)
residuals = y_data_sorted - y_fit_power_law

# Plot the residuals
plt.scatter(x_data_sorted, residuals, alpha=0.5, label='Residuals')
plt.axhline(y=0, color='r', linestyle='--', label='Zero Residual Line')
plt.xlabel('Absolute Magnitude (H)')
plt.ylabel('Residuals')
plt.title('Residual Plot')
plt.legend()
plt.show()

print(f'R-squared value: {r_squared:.2f}')
```



R-squared value: 0.9907

Conclusions

In this comprehensive study, NASA's Near-Earth Object Web Service (NeoWs) API was leveraged to collect valuable asteroid data. Leveraging the capabilities of a NoSQL database, MongoDB, we successfully stored this data for subsequent analysis. The ETL (Extract, Load, Transform) process facilitated the seamless transition of data from the API to the database, making it readily available for in-depth exploration.

Our investigation delved into several key aspects of asteroids, including estimated minimum and maximum diameters, minimum distances from Earth, relative velocities, absolute magnitude (H), and assessment of potential asteroid hazard. Specifically, our comparison of NASA's potential hazard standards revealed that they are less conservative than commonly used limits, with the potential hazard threshold set at a minimum orbit intersection distance (MOID) of 0.05 astronomical units (au) or less and an absolute magnitude (H) of 22.0.

One of the highlights of our analysis was the identification of an interesting inverse power-law relationship between the absolute error in the estimated asteroid diameter and the absolute magnitude (H) of the asteroid. The associated residuals demonstrated the

accuracy of the model, especially within the absolute magnitude range of 12 to 22, with a high R-squared value of 0.99. These results provide valuable insights into the accuracy of asteroid diameter estimates.

Additionally, our exploration uncovered that the Semi-Major Axis and Perihelion Distance are sufficient features to cluster asteroids based on their orbit class, simplifying the categorization of these celestial objects.

In summary, this study offers a deeper understanding of the characteristics and classification of asteroids, as well as a robust analysis of the accuracy of asteroid diameter estimations. The rich dataset and the insights gained from this analysis lay the groundwork for future research and further our understanding of these fascinating celestial bodies.