project-asteroids

October 31, 2023

1 Summary

In this study, we utilized the NASA Near-Earth Object Web Service (NeoWs) API to gather data on asteroids. The retrieved asteroid information, presented in JSON format, was then stored in a MongoDB database for further analysis. Leveraging the pymongo library, a series of operations were performed to extract valuable insights from the dataset.

The focus of this analysis centered around two key aspects: asteroid diameter and velocity. These findings provide a deeper understanding of the characteristics and behavior of these celestial objects, contributing to our knowledge of near-Earth asteroids.

2 Requirements and connection

[151]:
```
# import packages
import pymongo
import pprint as pp
import pandas as pd
import requests
import numpy as np
import matplotlib.pyplot as plt
import numpy as np
from scipy.optimize import curve_fit
from sklearn.metrics import r2_score
from sklearn.cluster import KMeans
# pandas configuration
pd.set_option('display.precision', 2)
```

Details about the API and database setup

[71]:
```
API_URL = "https://api.nasa.gov/neo/rest/v1/feed?
↪start_date=2015-09-07&end_date=2015-09-08&api_key=DEMO_KEY"
CNX_STR = "mongodb+srv://valleandrea:nosql_andrea@cluster0.puqrqtt.mongodb.net"
DB_NAME = "db_asteroids"
COLL_NAME = "asteroids"
```

3 ELT Process

3.1 Extract

[75]:
```
# Conncetion to the database
client = pymongo.MongoClient(CNX_STR)
db = client["asteroids_db"]
```

[78]:
```
# 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({})
```

[78]: <pymongo.results.DeleteResult at 0x2155840dbe0>

3.2 Load

[81]:
```
# 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
```

[86]:
```
# Load the dataset
near_earth_objects = []
for i in range(5):
    url = f"https://api.nasa.gov/neo/rest/v1/neo/browse?
 ↪page={i}&size=20&api_key=DEMO_KEY"
    r = requests.get(url)
    data = r.json()
    near_earth_objects.extend(data['near_earth_objects'])
```

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

[90]:
```python
# 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: 100
```

[95]:
```python
# Printing a sample of a document in the collection
r = col.aggregate([{"$limit":5}])
pd.DataFrame(r)
```

[95]:
```
                        _id       id                   name  \
0  65416a61d2dd94a9fbceb84d  2000433      433 Eros (A898 PA)
1  65416a61d2dd94a9fbceb84e  2000719    719 Albert (A911 TB)
2  65416a61d2dd94a9fbceb84f  2000887    887 Alinda (A918 AA)
3  65416a61d2dd94a9fbceb850  2001036  1036 Ganymed (A924 UB)
4  65416a61d2dd94a9fbceb851  2001221    1221 Amor (1932 EA1)

                                    nasa_jpl_url  absolute_magnitude_h  \
0  http://ssd.jpl.nasa.gov/sbdb.cgi?sstr=2000433                 10.41
1  http://ssd.jpl.nasa.gov/sbdb.cgi?sstr=2000719                 15.59
2  http://ssd.jpl.nasa.gov/sbdb.cgi?sstr=2000887                 13.88
3  http://ssd.jpl.nasa.gov/sbdb.cgi?sstr=2001036                  9.26
4  http://ssd.jpl.nasa.gov/sbdb.cgi?sstr=2001221                 17.37

                                  estimated_diameter  \
0  {'kilometers': {'estimated_diameter_min': 22.0…
1  {'kilometers': {'estimated_diameter_min': 2.02…
2  {'kilometers': {'estimated_diameter_min': 4.45…
3  {'kilometers': {'estimated_diameter_min': 37.3…
4  {'kilometers': {'estimated_diameter_min': 0.89…

   is_potentially_hazardous_asteroid  \
0                              False
1                              False
2                              False
3                              False
4                              False

                                 close_approach_data  \
0  [{'close_approach_date': '1900-12-27', 'close_…
```

```
1  [{'close_approach_date': '1909-08-21', 'close_…
2  [{'close_approach_date': '1910-01-04', 'close_…
3  [{'close_approach_date': '1910-02-25', 'close_…
4  [{'close_approach_date': '1900-03-08', 'close_…

                                       orbital_data  is_sentry_object
0  {'orbit_id': '659', 'orbit_determination_date'…             False
1  {'orbit_id': '257', 'orbit_determination_date'…             False
2  {'orbit_id': '441', 'orbit_determination_date'…             False
3  {'orbit_id': '1123', 'orbit_determination_date…             False
4  {'orbit_id': '113', 'orbit_determination_date'…             False
```

3.3 Transformation

[108]:
```
# Extract the average realtive speed based on the osservations

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)
```

[108]:

	_id	id	name	relativeSpeedAvg
0	65416a61d2dd94a9fbceb84d	2000433	433 Eros (A898 PA)	5.06
1	65416a61d2dd94a9fbceb84e	2000719	719 Albert (A911 TB)	6.20
2	65416a61d2dd94a9fbceb84f	2000887	887 Alinda (A918 AA)	10.09
3	65416a61d2dd94a9fbceb850	2001036	1036 Ganymed (A924 UB)	13.98
4	65416a61d2dd94a9fbceb851	2001221	1221 Amor (1932 EA1)	10.46

[107]:

```
# Extract the  minimum distance based on the osservations
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)
```

[107]:

	_id	id	name	missDistance_min
0	65416a61d2dd94a9fbceb84d	2000433	433 Eros (A898 PA)	0.15
1	65416a61d2dd94a9fbceb84e	2000719	719 Albert (A911 TB)	0.21
2	65416a61d2dd94a9fbceb84f	2000887	887 Alinda (A918 AA)	0.08
3	65416a61d2dd94a9fbceb850	2001036	1036 Ganymed (A924 UB)	0.03
4	65416a61d2dd94a9fbceb851	2001221	1221 Amor (1932 EA1)	0.11

3.4 Datastructure

4 Orbital analysis

Calculate the distribution of the orbits

[119]:
```
dist_orbits = col.aggregate([{
        '$group': {
            '_id': '$orbital_data.orbit_class.orbit_class_type',
            'count': {'$sum': 1}
        }
    }])
pd.DataFrame(dist_orbits)
```

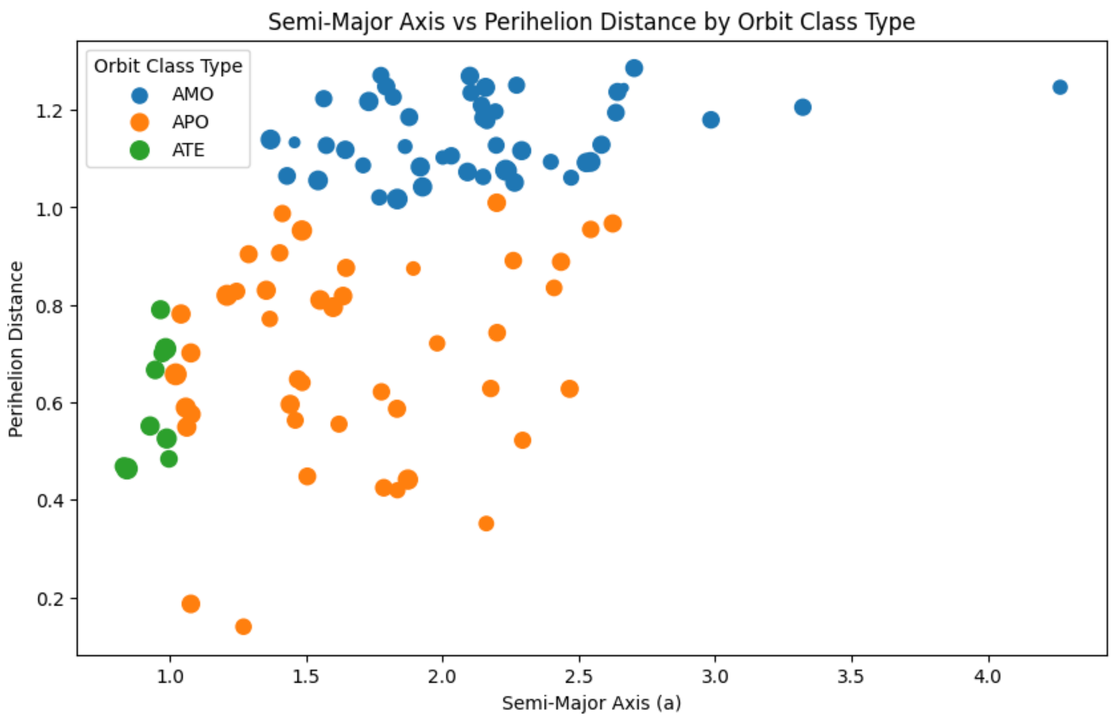
[119]:

	_id	count
0	AMO	47
1	APO	44

```
2  ATE      9
```

[120]:
```
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()
```



[25]:
```
# Diameter speed relation
```

[130]:
```
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,
            "relativeSpeed_Avg": 1,
            "missDistance_min": 1
```

```
        }
    }
]

result = col.aggregate(pipeline)
df = pd.DataFrame(list(result))

estimated_diameter_min = []
orbit_class_type_list = []

for index, row in df.iterrows():
    orbit_class_type_list.
 ↪append(row["orbital_data"]["orbit_class"]["orbit_class_type"])
    estimated_diameter = row["absolute_magnitude_h"]
    estimated_diameter_min.append(estimated_diameter)

plt.figure(figsize=(10, 6))
marker_size = 20 + 100 * np.array(estimated_diameter_min)

for i in range(len(df)):
    plt.scatter(
        df["relativeSpeed_Avg"].iloc[i],
        df["missDistance_min"].iloc[i],
        label=orbit_class_type_list[i],
        s=marker_size[i],
        alpha=0.5
    )

plt.xlabel('Relative Speed (km/s)')
plt.ylabel('Miss Distance (astronomical units)')
plt.title('Relative Speed vs Miss Distance')
plt.legend(title='Orbit Class Type', loc='upper right', bbox_to_anchor=(1.3, 1))
plt.show()
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
File ~\OneDrive - Hochschule␣
 ↪Luzern\Desktop\AsteroidsDB\venv\lib\site-packages\pandas\core\indexes\base.py
 ↪3790, in Index.get_loc(self, key)
   3789 try:
-> 3790     return self._engine.get_loc(casted_key)
   3791 except KeyError as err:

File index.pyx:152, in pandas._libs.index.IndexEngine.get_loc()

File index.pyx:181, in pandas._libs.index.IndexEngine.get_loc()
```

```
File pandas\_libs\hashtable_class_helper.pxi:7080, in pandas._libs.hashtable.
↪PyObjectHashTable.get_item()

File pandas\_libs\hashtable_class_helper.pxi:7088, in pandas._libs.hashtable.
↪PyObjectHashTable.get_item()

KeyError: 'orbital_data.orbit_class.orbit_class_type'

The above exception was the direct cause of the following exception:

KeyError                                  Traceback (most recent call last)
Cell In[130], line 18
     15 df = pd.DataFrame(list(result))
     17 # Extract unique orbit class types
---> 18 unique_orbit_classes = df["orbital_data.orbit_class.orbit_class_type"].
↪unique()
     20 plt.figure(figsize=(10, 6))
     21 marker_size = 20 + 100 * np.array(estimated_diameter_min)

File ~\OneDrive - Hochschule␣
↪Luzern\Desktop\AsteroidsDB\venv\lib\site-packages\pandas\core\frame.py:3893,␣
↪in DataFrame.__getitem__(self, key)
   3891 if self.columns.nlevels > 1:
   3892     return self._getitem_multilevel(key)
-> 3893 indexer = self.columns.get_loc(key)
   3894 if is_integer(indexer):
   3895     indexer = [indexer]

File ~\OneDrive - Hochschule␣
↪Luzern\Desktop\AsteroidsDB\venv\lib\site-packages\pandas\core\indexes\base.py
↪3797, in Index.get_loc(self, key)
   3792     if isinstance(casted_key, slice) or (
   3793         isinstance(casted_key, abc.Iterable)
   3794         and any(isinstance(x, slice) for x in casted_key)
   3795     ):
   3796         raise InvalidIndexError(key)
-> 3797     raise KeyError(key) from err
   3798 except TypeError:
   3799     # If we have a listlike key, _check_indexing_error will raise
   3800     #  InvalidIndexError. Otherwise we fall through and re-raise
   3801     #  the TypeError.
   3802     self._check_indexing_error(key)

KeyError: 'orbital_data.orbit_class.orbit_class_type'
```

[143]:
```python
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 Nasa:", count)
```

```
The number of dangerous asteroids from Nasa: 43
```

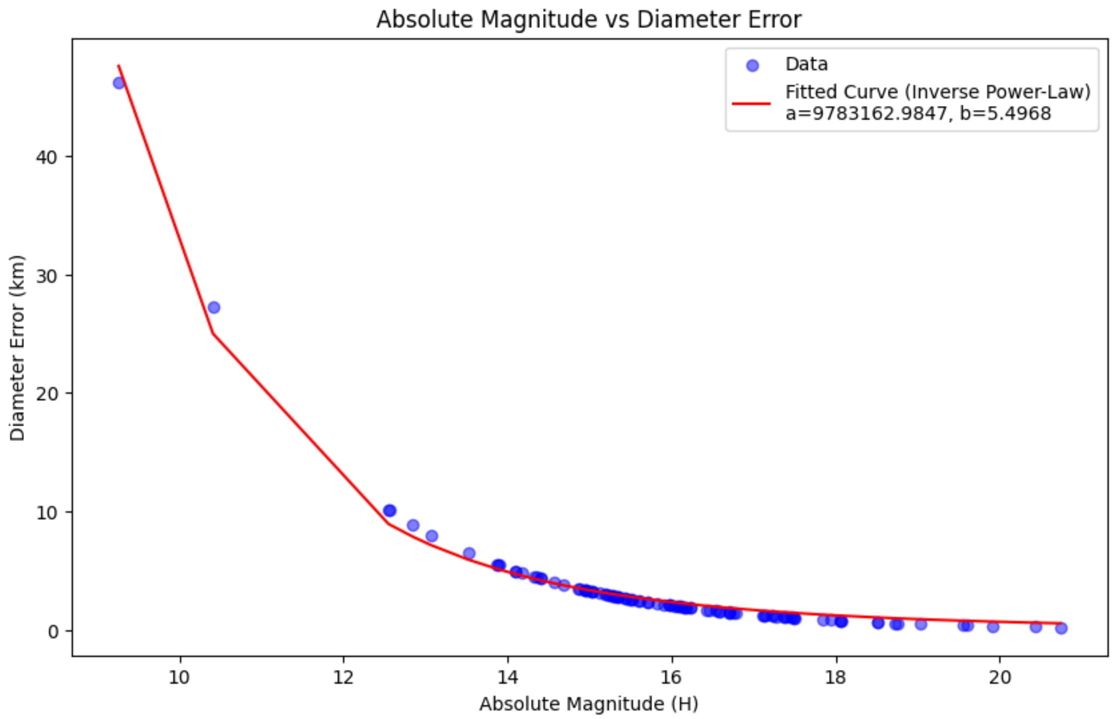
[142]:
```python
pipeline = [
    {
        "$match": {
            "is_potentially_hazardous_asteroid": True,
        }
    },
    {
        "$count": "count"
    }
]

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: 28
```

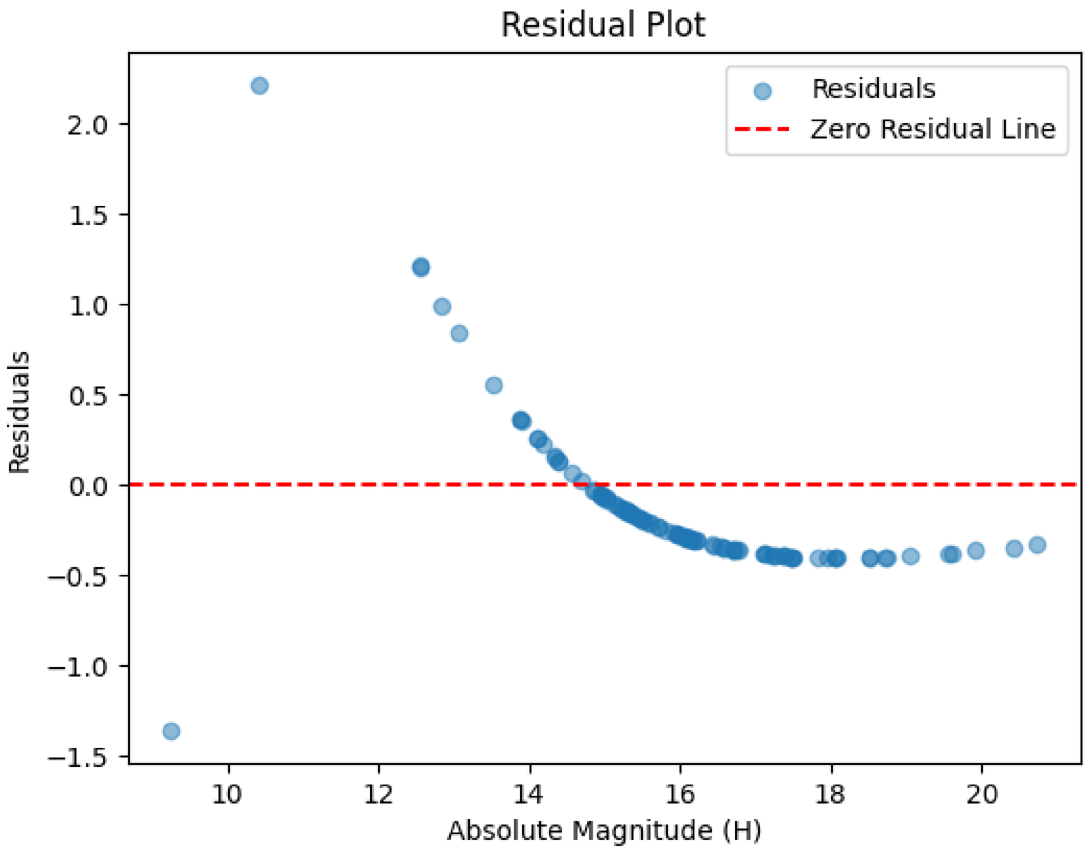
Function of absolute error of the diameter based on the velocity

[153]:
```python
pipeline = [
    {
        "$addFields": {
            "diameter_error": {
                "$subtract": [
                    "$estimated_diameter.kilometers.estimated_diameter_max",
                    "$estimated_diameter.kilometers.estimated_diameter_min"
                ]
            }
        }
    }
]
```

```python
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

# Calculate the fitted y values
y_fit_power_law = inverse_power_law(x_data_sorted, a_opt, b_opt)

# Create the combined plot
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:.4f}, b={b_opt:.4f}')

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



[155]:
```
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:.4f}')
```




```
R-squared value: 0.9930
```