☄️ Hazardous Asteroid Classification - NASA JPL Asteroid by Alexander D. Rios

Abstract

The ***problem*** we address is identifying potentially hazardous asteroids for planet Earth. The scope of this project can be interesting for various entities, ranging from astronomical research centers to aerospace agencies.

The ***motivation*** behind this project arises from the difficulty I believe exists in tackling such a topic, which allows me to apply all my mathematical, physical, and statistical knowledge for data study. I recognize that it is one of the most complicated projects I have undertaken due to my limited knowledge in the field of astronomy, but I believe I have the necessary tools to deal with some of the challenges this type of project imposes.

The ***first hypothesis*** I can formulate is that asteroids with orbits closer to Earth's orbit have higher chances of representing a risk for the planet than those further away, and that this is independent of their size.

To verify my hypothesis, I used various methods, both qualitative and quantitative, as well as different designs. Some of the main ones were visualizations like boxplots and statistical descriptions of the data. Key visualizations included boxplots of the `H` and `moid` variables, as well as visualizations of orbits close to Earth and the `neo` vs. `moid` histogram.

Some of the ***extensions of my work*** were calculating the orbital paths of the asteroids through 5 orbital elements or Keplerian elements, the approximate calculation of the asteroid diameters, since this and some other data (`prefix`, `name`, `albedo`, `diameter_sigma`, and `diameter`) had to be excluded from the study due to missing data, i.e., they contained too many null values making them unusable columns. I also had to discard 25,637 records due to extreme outliers, which skewed the useful data. These were

discarded considering the vast amount of data available (1M records), but I took care not to unbalance the data more than it already was. On the other hand, I filled in the missing `H` data with the mean because they are homogeneous values.

Conclusion: Based on the tools used so far, I concluded that the danger of an asteroid lies in the possibility of it intercepting Earth's orbit or not. This means that asteroids with orbits close to Earth's orbit are potentially hazardous, while those with orbits farther away may also be, but with lower probabilities. I also demonstrated that their size does not contribute to their potential danger unless they can intercept Earth's orbit. Clearly, if a massive asteroid were to intercept Earth, it would cause greater damage than a less massive one.

Objective

Throughout human history, there have been countless discussions about the end of the world. One of the main and most plausible causes is the impact of an asteroid. Such an event could be so catastrophic that it threatens to wipe out all existing life on planet Earth.

But from our position as data scientists, what can we do? To answer this question, we have access to a dataset containing information about various asteroids known to humanity. This dataset describes specific physical and temporal characteristics of these asteroids. Based on this, several questions arise:

> **Is there a pattern that allows us to identify potentially hazardous asteroids?**

> **What are the probabilities of an asteroid colliding with Earth in the coming years?**

Business context

An **observatory in Argentina** has detected several asteroids near Earth's orbit. Additionally, it has determined that this weekend, there will be a meteor shower, which consists of debris from asteroids. Fortunately, the observatory has been collecting precise data on these asteroids for several years up to the present.

NASA has hired us to **identify visual patterns** in this data to help **classify** whether these asteroids pose a threat to our ecosystem. The goal is to take preventive actions to **alter their course and avoid a potential impact**, thus preventing the extinction of humanity.

Business problem

Based on the data provided by the observatory, we need to create visualizations to answer the following questions:

- **Are all asteroids near Earth's orbit potentially hazardous?**
- **What type of orbit do most asteroids have?**
- **Is there a relationship between an asteroid's hazard level and its physical size?**
- **Between Mars and Jupiter, there is an asteroid belt. How does Jupiter's massive size affect the orbits of these asteroids?**
- **Are there asteroids with orbits smaller than Earth's that pose a potential threat?**

Analytical context

The observatory has provided us with a dataset in `.CSV` format containing approximately one million records on asteroids. Some of the recorded characteristics include orbital eccentricity, longitude of the descending node, absolute magnitude of the asteroid, among 43 other available features.

The internal index of the dataset is named `id`.

Based on this data, we need to carry out the following tasks:

- **Read and preview the dataset.**
- **Detect and process missing data, determine whether it can be discarded, or otherwise, fill in the gaps.**
- **Detect and process outliers.**
- **Identify relevant features.**
- **Analyze and create visualizations of the data to answer the proposed questions and identify useful patterns.**

Let's get to work! 😊

Installing packages

In [ ]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call
drive.mount("/content/drive", force_remount=True).
```

In [ ]:
```
!pip install plotly==5.9.0 missingno mplcyberpunk kaleido
```

In [ ]:
```
!pip install git+https://github.com/poliastro/poliastro.git@0.16.x --ignore-
```

Importing packages

In [ ]:

```
import pandas as pd
import numpy as np

import matplotlib.pyplot as plt
import matplotlib as mpl

import seaborn as sns

import plotly.graph_objects as go
import plotly.io as pio
import plotly.express as px

import missingno as msno

from astropy.time import Time
from astropy import units as u
from poliastro.bodies import Earth, Sun
from poliastro.twobody import Orbit
from poliastro.plotting.misc import plot_solar_system
from poliastro.frames import Planes

from sklearn.preprocessing import LabelEncoder, StandardScaler, MinMaxScaler
from sklearn.model_selection import train_test_split, GridSearchCV, cross_va
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.feature_selection import SequentialFeatureSelector
from sklearn.metrics import (confusion_matrix, classification_report, DetCurv
                             precision_score, recall_score, f1_score, accurac
from sklearn.model_selection import learning_curve, LearningCurveDisplay
from sklearn.impute import SimpleImputer

from scipy.spatial.transform import Rotation

from statsmodels.stats.outliers_influence import variance_inflation_factor

from imblearn.over_sampling import SMOTE, RandomOverSampler
from imblearn.under_sampling import NearMiss, RandomUnderSampler

import os
import random
import gc
import time

from google.colab import output
import mplcyberpunk

plt.style.use("cyberpunk")
pd.set_option('display.max_columns', 100)
```

Setting up for the reproducibility

In [ ]:
```
seed_value = 42
os.environ['PYTHONHASHSEED'] = str(seed_value)
random.seed(seed_value)
np.random.seed(seed_value)
```

In [ ]:
```
from IPython.display import Javascript
def adjust_display():
  display(Javascript('''google.colab.output.setIframeHeight(0, true, {maxHei
```

📣 About the dataset

This dataset was created by the researcher in Astronomy and Astrophysics, Mir Sakhawat Hossain. It is officially maintained by the Jet Propulsion Laboratory (JPL) of the California Institute of Technology, an organization supervised by NASA. This dataset contains various types of data related to asteroids.

It can be used in Machine Learning projects for both classification and regression tasks.

Column definitions:

- **id**: Internal ID
- **spkid**: Primary ID
- **fullname**: Full designation/name of the object
- **pdes**: Primary designation of the object
- **name**: Object name as per the International Astronomical Union
- **prefix**: Comet prefix
- **neo**: Near-Earth Object (Y/N)
- **pha**: Potentially Hazardous Asteroid (Y/N)
- **H**: Absolute magnitude parameter
- **diameter**: Object diameter (equivalent to a sphere) (km)
- **albedo**: Geometric albedo
- **diameter_sigma**: 1-sigma uncertainty in the object's diameter (km)
- **orbit_id**: Orbit solution ID
- **epoch**: Osculation epoch in Julian day format (TBD)
- **epoch_mjd**: Osculation epoch in Modified Julian day format (TBD)
- **epoch_cal**: Osculation epoch in calendar date/time format (TBD)
- **equinox**: Reference frame equinox
- **e**: Eccentricity
- **a**: Semi-major axis (au)
- **q**: Perihelion distance (au)
- **i**: Inclination. Angle relative to the x-y ecliptic plane (deg)
- **om**: Longitude of the ascending node (deg)
- **w**: Argument of perihelion (deg)
- **ma**: Mean anomaly (deg)

- **ad**: Aphelion distance (au) (also called Q)
- **n**: Mean motion (deg/d)
- **tp**: Time of perihelion passage (TBD)
- **tp_cal**: Time of perihelion passage in calendar date/time format (TBD)
- **per**: Orbital sidereal period (d)
- **per_y**: Orbital sidereal period (years)
- **moid**: Minimum orbit intersection distance with Earth (au)
- **moid_ld**: Minimum orbit intersection distance with Earth (LD)
- **sigma_e**: Eccentricity (1-sigma uncertainty)
- **sigma_a**: Semi-major axis (1-sigma uncertainty) (au)
- **sigma_q**: Perihelion distance (1-sigma uncertainty) (au)
- **sigma_i**: Inclination. Angle relative to the x-y ecliptic plane (1-sigma uncertainty) (deg)
- **sigma_om**: Longitude of the ascending node (1-sigma uncertainty) (deg)
- **sigma_w**: Argument of perihelion (1-sigma uncertainty) (deg)
- **sigma_ma**: Mean anomaly (1-sigma uncertainty) (deg)
- **sigma_ad**: Aphelion distance (1-sigma uncertainty) (au)
- **sigma_n**: Mean motion (1-sigma uncertainty) (deg/d)
- **sigma_tp**: Time of perihelion passage (1-sigma uncertainty) (TBD)
- **sigma_per**: Orbital sidereal period (1-sigma uncertainty) (d)
- **class**: Orbit classification
- **rms**: Normalized orbit fit RMS (arcsec)

👁️ Taking a look at the dataset

In [ ]:
```
source_path = "/content/drive/MyDrive/Coder/Data_Science/Hazardous_Asteroids
dest_path = "/content/drive/MyDrive/Coder/Data_Science/Hazardous_Asteroids/"

df_asteroids = pd.read_csv(source_path+"asteroid_dataset.csv", engine='c', l
df_asteroids.head()
```

Out[ ]:

id	spkid	full_name	pdes	name	prefix	neo	pha	H	diameter	albedo	
a0000001	2000001	1 Ceres	1	Ceres	NaN	N	N	3.40	939.400	0.0900	
a0000002	2000002	2 Pallas	2	Pallas	NaN	N	N	4.20	545.000	0.1010	
a0000003	2000003	3 Juno	3	Juno	NaN	N	N	5.33	246.596	0.2140	
a0000004	2000004	4 Vesta	4	Vesta	NaN	N	N	3.00	525.400	0.4228	
a0000005	2000005	5 Astraea	5	Astraea	NaN	N	N	6.90	106.699	0.2740	

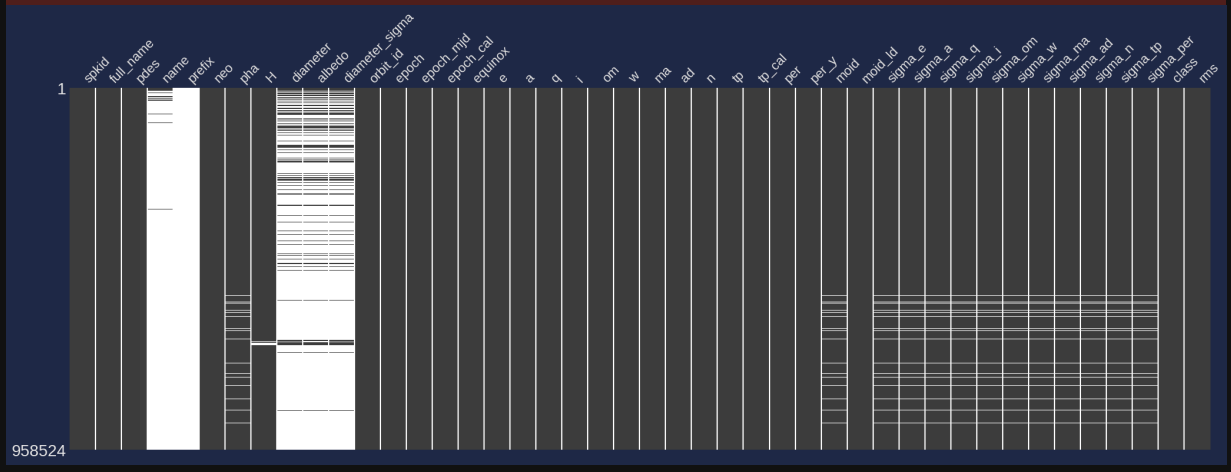
In [ ]:
```
df_asteroids.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 958524 entries, a0000001 to bT3S2678
Data columns (total 44 columns):
 #   Column          Non-Null Count   Dtype  
---  ------          --------------   -----  
 0   spkid           958524 non-null  int64  
 1   full_name       958524 non-null  object 
 2   pdes            958524 non-null  object 
 3   name            22064 non-null   object 
 4   prefix          18 non-null      object 
 5   neo             958520 non-null  object 
 6   pha             938603 non-null  object 
 7   H               952261 non-null  float64
 8   diameter        136209 non-null  float64
 9   albedo          135103 non-null  float64
 10  diameter_sigma  136081 non-null  float64
 11  orbit_id        958524 non-null  object 
 12  epoch           958524 non-null  float64
 13  epoch_mjd       958524 non-null  int64  
 14  epoch_cal       958524 non-null  float64
 15  equinox         958524 non-null  object 
 16  e               958524 non-null  float64
 17  a               958524 non-null  float64
 18  q               958524 non-null  float64
 19  i               958524 non-null  float64
 20  om              958524 non-null  float64
 21  w               958524 non-null  float64
 22  ma              958523 non-null  float64
 23  ad              958520 non-null  float64
 24  n               958524 non-null  float64
 25  tp              958524 non-null  float64
 26  tp_cal          958524 non-null  float64
 27  per             958520 non-null  float64
 28  per_y           958523 non-null  float64
 29  moid            938603 non-null  float64
 30  moid_ld         958397 non-null  float64
 31  sigma_e         938602 non-null  float64
 32  sigma_a         938602 non-null  float64
 33  sigma_q         938602 non-null  float64
 34  sigma_i         938602 non-null  float64
 35  sigma_om        938602 non-null  float64
 36  sigma_w         938602 non-null  float64
 37  sigma_ma        938602 non-null  float64
 38  sigma_ad        938598 non-null  float64
 39  sigma_n         938602 non-null  float64
 40  sigma_tp        938602 non-null  float64
 41  sigma_per       938598 non-null  float64
 42  class           958524 non-null  object 
 43  rms             958522 non-null  float64
dtypes: float64(33), int64(2), object(9)
memory usage: 329.1+ MB
```

🕵️ Searching for missing data

In [ ]:
```
_, ax = plt.subplots(1, 1, figsize=(25, 8))
msno.matrix(df_asteroids, ax=ax);
```

```
/usr/local/lib/python3.11/dist-packages/missingno/missingno.py:61: UserWarnin
g: Plotting a sparkline on an existing axis is not currently supported. To re
move this warning, set sparkline=False.
  warnings.warn(
```



In [ ]:
```
cantidad = df_asteroids.isna().sum()
cantidad = cantidad[cantidad.values > 0]
porcentaje = cantidad * 100 / df_asteroids.shape[0]
df_nulls = pd.DataFrame(data={"Number of missing values":cantidad, "Percenta
df_nulls
```

Out[ ]:

	Number of missing values	Percentage of missing values
prefix	958506	99.998122
name	936460	97.698128
albedo	823421	85.905100
diameter_sigma	822443	85.803068
diameter	822315	85.789714
sigma_per	19926	2.078821
sigma_ad	19926	2.078821
sigma_i	19922	2.078404
sigma_q	19922	2.078404
sigma_a	19922	2.078404
sigma_e	19922	2.078404
sigma_n	19922	2.078404
sigma_ma	19922	2.078404
sigma_w	19922	2.078404
sigma_tp	19922	2.078404
sigma_om	19922	2.078404
moid	19921	2.078300
pha	19921	2.078300
H	6263	0.653400
moid_ld	127	0.013250
per	4	0.000417
ad	4	0.000417
neo	4	0.000417
rms	2	0.000209
per_y	1	0.000104
ma	1	0.000104

Our `target` is `pha`, and as we can see, it has missing values. Fortunately, they only represent 2.07% of the total data.

Without `target` values, these records are not useful for prediction.

Let's take a look at them:

In [ ]:
```
df_asteroids[df_asteroids["pha"].isna()]
```

Out[ ]:

id	spkid	full_name	pdes	name	prefix	neo	pha	H	diameter	albedo
bJ39R00R	3246903	(1939 RR)	1939 RR	NaN	NaN	N	NaN	12.00	NaN	NaN
bJ90O05K	3803913	(1990 OK5)	1990 OK5	NaN	NaN	N	NaN	16.40	NaN	NaN
bJ91R28N	3884244	(1991 RN28)	1991 RN28	NaN	NaN	N	NaN	19.20	NaN	NaN
bJ93T11C	3884246	(1993 TC11)	1993 TC11	NaN	NaN	N	NaN	19.30	NaN	NaN
bJ94A09F	3884247	(1994 AF9)	1994 AF9	NaN	NaN	N	NaN	18.50	NaN	NaN
...	...	...	...	...	...	...	...	...	...	...
bK20K03Q	54017210	(2020 KQ3)	2020 KQ3	NaN	NaN	N	NaN	19.90	NaN	NaN
bK20K03R	54017211	(2020 KR3)	2020 KR3	NaN	NaN	N	NaN	19.40	NaN	NaN
bK20K03V	54017212	(2020 KV3)	2020 KV3	NaN	NaN	N	NaN	19.91	NaN	NaN
bK20K04A	54017209	(2020 KA4)	2020 KA4	NaN	NaN	N	NaN	18.50	NaN	NaN
bK20K04C	54017222	(2020 KC4)	2020 KC4	NaN	NaN	N	NaN	21.70	NaN	NaN

19921 rows × 44 columns

Apparently, these records contain many other columns with missing data, so it will be beneficial to discard them.

In [ ]:
```
df_asteroids = df_asteroids[~df_asteroids["pha"].isna()]
cantidad = df_asteroids.isna().sum()
cantidad = cantidad[cantidad.values > 0]
porcentaje = cantidad * 100 / df_asteroids.shape[0]
df_nulls = pd.DataFrame(data={"Number of missing values":cantidad, "Percenta
df_nulls
```

Out[ ]:

	Number of missing values	Percentage of missing values
prefix	938585	99.998082
name	916539	97.649272
albedo	803500	85.605948
diameter_sigma	802522	85.501751
diameter	802394	85.488114
H	6262	0.667162
sigma_per	5	0.000533
sigma_ad	5	0.000533
neo	4	0.000426
ad	4	0.000426
per	4	0.000426
sigma_w	1	0.000107
sigma_tp	1	0.000107
sigma_n	1	0.000107
sigma_ma	1	0.000107
sigma_e	1	0.000107
sigma_om	1	0.000107
sigma_i	1	0.000107
sigma_q	1	0.000107
sigma_a	1	0.000107
per_y	1	0.000107
ma	1	0.000107
rms	1	0.000107

As we can see, the percentage of missing values in many `features` has significantly decreased.

I will consider that `features` with a missing value percentage greater than 3% can be discarded.

In [ ]:
```
columns_nan = df_nulls[df_nulls["Percentage of missing values"] > 3].index
df_nulls[df_nulls["Percentage of missing values"] > 3]
```

Out[ ]:

	Number of missing values	Percentage of missing values
prefix	938585	99.998082
name	916539	97.649272
albedo	803500	85.605948
diameter_sigma	802522	85.501751
diameter	802394	85.488114

These features will be discarded due to their abundant amount of missing values.

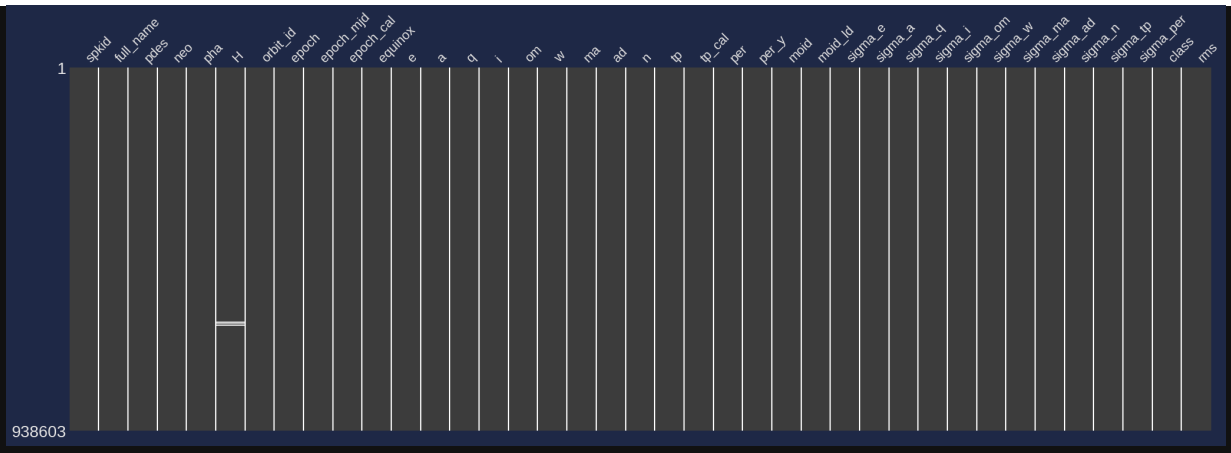
In [ ]:
```
df_asteroids.drop(columns_nan, axis=1, inplace=True)
df_asteroids.head()
```

Out[ ]:

	spkid	full_name	pdes	neo	pha	H	orbit_id	epoch	epoch_mjd	e
id										
a0000001	2000001	1 Ceres	1	N	N	3.40	JPL 47	2458600.5	58600	20
a0000002	2000002	2 Pallas	2	N	N	4.20	JPL 37	2459000.5	59000	20
a0000003	2000003	3 Juno	3	N	N	5.33	JPL 112	2459000.5	59000	20
a0000004	2000004	4 Vesta	4	N	N	3.00	JPL 35	2458600.5	58600	20
a0000005	2000005	5 Astraea	5	N	N	6.90	JPL 114	2459000.5	59000	20

In [ ]:
```
_, ax = plt.subplots(1, 1, figsize=(25, 8))
msno.matrix(df_asteroids, ax=ax);
```

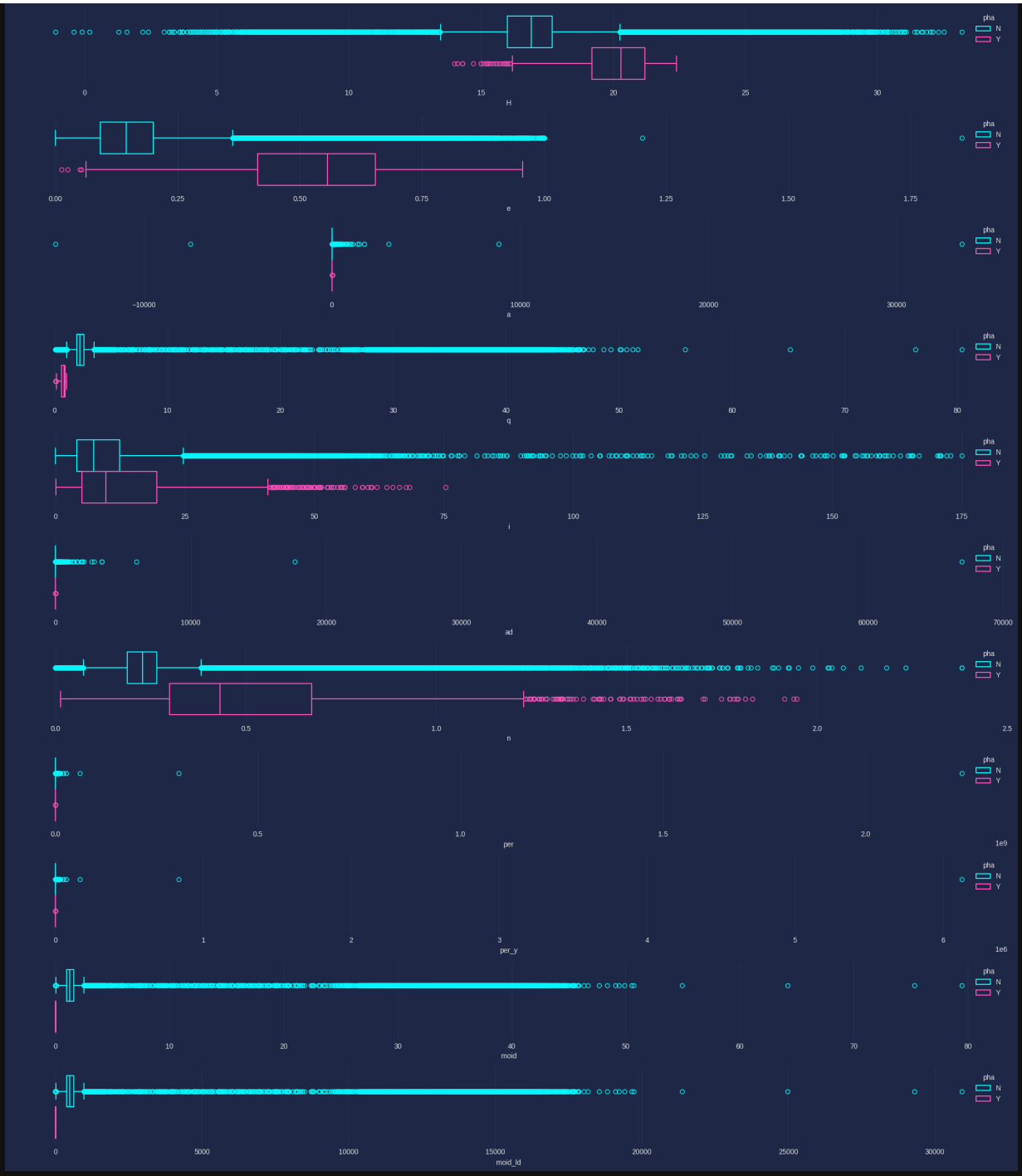
```
/usr/local/lib/python3.11/dist-packages/missingno/missingno.py:61: UserWarnin
g: Plotting a sparkline on an existing axis is not currently supported. To re
move this warning, set sparkline=False.
  warnings.warn(
```



Once the `features` with the highest amount of missing data have been discarded, let's proceed to explore the dataset.

Outliers detection

In [ ]:
```
adjust_display()

columns_numerics = ["H", "e", "a", "q", "i", "ad", "n", "per", "per_y", "moi

fig = plt.figure(figsize=(20, 25))
axs = fig.subplots(len(columns_numerics), 1)
for i, col in enumerate(columns_numerics):
  sns.boxplot(data=df_asteroids, x=col, ax=axs[i], hue="pha", fill=False)
fig.tight_layout(rect=[0, 0.03, 1, 0.95])
```



In [ ]:
```
adjust_display()
df_asteroids[columns_numerics].describe().transpose()
```

Out[ ]:

	count	mean	std	min	25%	50%
H	932341.0	16.889969	1.801386e+00	-1.100000e+00	16.000000	16.900000
e	938603.0	0.156146	9.300582e-02	0.000000e+00	0.092064	0.144865
a	938603.0	2.906752	4.013775e+01	-1.470245e+04	2.388114	2.647342
q	938603.0	2.397725	2.159374e+00	7.051073e-02	1.971656	2.226687
i	938603.0	9.048065	6.646721e+00	7.744220e-03	4.154593	7.404507
ad	938599.0	3.463186	7.252509e+01	6.537730e-01	2.781941	3.047348
n	938603.0	0.236729	7.996385e-02	1.608247e-07	0.189435	0.228817
per	938599.0	5157.181148	2.333335e+06	1.511918e+02	1347.973542	1573.310707
per_y	938602.0	14.119547	6.388312e+03	0.000000e+00	3.690548	4.307484
moid	938603.0	1.415162	2.156731e+00	4.544120e-07	0.979702	1.240850
moid_ld	938603.0	550.738745	8.393348e+02	1.768435e-04	381.270627	482.901594

In [ ]:
```
old_len = len(df_asteroids)
df_asteroids.pha.value_counts()
```

Out[ ]:

pha	count
N	936537
Y	2066

dtype: int64

As we can see in our data, we have many extreme outliers. These will pose a problem for the training of our models. Let's see how we can handle them.

In [ ]:
```
def get_outliers(df, col, tipo="leve"):
  Q1 = df[col].quantile(0.25)
  Q3 = df[col].quantile(0.75)
  IQR = Q3 - Q1
  if tipo == "leve":
    k = 1.5
  else:
    k = 3
  return df[(df[col] >= Q3 + k*IQR) | (df[col] <= Q1 - k*IQR)]

def table_outliers(df, columns):
  df_outliers = pd.DataFrame(columns=["Feature", "Mild outliers", "Percentag
  for col in columns:
    leves = get_outliers(df, col, tipo="leve")
    extremos = get_outliers(df, col, tipo="extremo")
    df_new = pd.DataFrame(data={"Feature":[col],
```

```
                                    "Mild outliers":[len(leves)],
                                    "Percentage of mild outliers": len(leves)*10
                                    "Extreme outliers":[len(extremos)],
                                    "Percentage of extreme outliers": len(extrem
                                    "PHA = Y": [len(extremos[extremos["pha"] ==
                                    "PHA = N": [len(extremos[extremos["pha"] ==
                                    })
    df_outliers = pd.concat((df_outliers, df_new), axis=0, ignore_index = Tr
  return df_outliers

table_outliers(df_asteroids, columns_numerics).sort_values(by=["PHA = Y", "P
```

```
<ipython-input-19-38ba08b9ae9c>:24: FutureWarning: The behavior of DataFrame
concatenation with empty or all-NA entries is deprecated. In a future versio
n, this will no longer exclude empty or all-NA columns when determining the r
esult dtypes. To retain the old behavior, exclude the relevant entries before
the concat operation.
  df_outliers = pd.concat((df_outliers, df_new), axis=0, ignore_index = True)
```

Out[ ]:

	Feature	Mild outliers	Percentage of mild outliers	Extreme outliers	Percentage of extreme outliers	PHA = Y	PHA = N
0	H	39678	4.227346	17160	1.828249	0	17160
9	moid	24551	2.615696	11465	1.221496	0	11465
10	moid_ld	24551	2.615696	11465	1.221496	0	11465
7	per	22113	2.355948	12221	1.302041	1	12220
8	per_y	22116	2.356268	12221	1.302041	1	12220
2	a	24747	2.636578	12186	1.298312	1	12185
3	q	29274	3.118891	11546	1.230126	12	11534
5	ad	29694	3.163638	13404	1.428080	55	13349
4	i	29787	3.173546	1625	0.173130	114	1511
6	n	23985	2.555393	9772	1.041122	851	8921
1	e	24658	2.627096	8761	0.933408	1166	7595

We will address the variables with the highest number of extreme outliers. As we can see, these represent less than 2% of our dataset. With nearly 1M records, removing these outliers will not have a significant impact on our dataset. However, we must ensure not to unbalance our data too much, as we have few positive records of `pha`. According to the table above, we can see that the outliers in `H` are all negative cases of `pha`. Removing these data will also slightly help balance our target.

In [ ]:
```
H_outliers = get_outliers(df_asteroids, "H", "extremos")
df_asteroids.drop(H_outliers.index, axis=0, inplace=True)
table_outliers(df_asteroids, ["e", "a", "q", "i", "ad", "n", "per", "per_y",
```

```
<ipython-input-19-38ba08b9ae9c>:24: FutureWarning: The behavior of DataFrame
concatenation with empty or all-NA entries is deprecated. In a future versio
n, this will no longer exclude empty or all-NA columns when determining the r
esult dtypes. To retain the old behavior, exclude the relevant entries before
the concat operation.
  df_outliers = pd.concat((df_outliers, df_new), axis=0, ignore_index = True)
```

Out[ ]:

	Feature	Mild outliers	Percentage of mild outliers	Extreme outliers	Percentage of extreme outliers	PHA = Y	PHA = N
8	moid	12605	1.367963	7789	0.845305	0	7789
9	moid_ld	12605	1.367963	7789	0.845305	0	7789
6	per	15052	1.633525	8469	0.919102	1	8468
7	per_y	15053	1.633633	8469	0.919102	1	8468
1	a	15875	1.722841	8437	0.915629	1	8436
2	q	15710	1.704935	7870	0.854095	19	7851
4	ad	21096	2.289453	9686	1.051177	72	9614
3	i	28817	3.127377	1467	0.159207	116	1351
5	n	13609	1.476923	3616	0.392428	858	2758
0	e	17665	1.917102	5388	0.584735	1194	4194

Alright, we handled the variable `H`, so we excluded it from the table. Now let's proceed with `moid`, as it does not have any positive values in the target.

In [ ]:
```
moid_outliers = get_outliers(df_asteroids, "moid", "extremos")
df_asteroids.drop(moid_outliers.index, axis=0, inplace=True)
table_outliers(df_asteroids, ["e", "a", "q", "i", "ad", "n", "per", "per_y"]
```

```
<ipython-input-19-38ba08b9ae9c>:24: FutureWarning: The behavior of DataFrame
concatenation with empty or all-NA entries is deprecated. In a future versio
n, this will no longer exclude empty or all-NA columns when determining the r
esult dtypes. To retain the old behavior, exclude the relevant entries before
the concat operation.
  df_outliers = pd.concat((df_outliers, df_new), axis=0, ignore_index = True)
```

Out[ ]:

	Feature	Mild outliers	Percentage of mild outliers	Extreme outliers	Percentage of extreme outliers	PHA = Y	PHA = N
6	per	7369	0.806542	684	0.074864	1	683
7	per_y	7369	0.806542	684	0.074864	1	683
1	a	8260	0.904062	656	0.071800	1	655
2	q	8127	0.889505	161	0.017622	25	136
4	ad	13843	1.515125	2077	0.227329	87	1990
3	i	28201	3.086617	1415	0.154873	118	1297
5	n	13700	1.499474	3632	0.397525	860	2772
0	e	17574	1.923485	5339	0.584357	1199	4140

Since `moid` and `moid_ld` represent the same values but with different scales and we have already addressed them, we can exclude them from the table. The next column to handle is `per`.

In [ ]:
```
per_outliers = get_outliers(df_asteroids, "per", "extremos")
df_asteroids.drop(per_outliers.index, axis=0, inplace=True)
table_outliers(df_asteroids, ["e", "a", "q", "i", "ad", "n"]).sort_values(by
```

```
<ipython-input-19-38ba08b9ae9c>:24: FutureWarning: The behavior of DataFrame
concatenation with empty or all-NA entries is deprecated. In a future versio
n, this will no longer exclude empty or all-NA columns when determining the r
esult dtypes. To retain the old behavior, exclude the relevant entries before
the concat operation.
  df_outliers = pd.concat((df_outliers, df_new), axis=0, ignore_index = True)
```

Out[ ]:

	Feature	Mild outliers	Percentage of mild outliers	Extreme outliers	Percentage of extreme outliers	PHA = Y	PHA = N
1	a	7586	0.830914	4	0.000438	0	4
2	q	7699	0.843292	79	0.008653	25	54
4	ad	13212	1.447145	1408	0.154222	89	1319
3	i	28106	3.078524	1288	0.141078	118	1170
5	n	13496	1.478252	3632	0.397822	860	2772
0	e	17296	1.894476	5141	0.563107	1198	3943

Once again, we exclude `per` and `per_y` as they represent the same variable. Finally, we are left to process the variable `a`.

In [ ]:
```
a_outliers = get_outliers(df_asteroids, "a", "extremos")
df_asteroids.drop(a_outliers.index, axis=0, inplace=True)
table_outliers(df_asteroids, ["e", "q", "i", "ad", "n"]).sort_values(by=["PH
```

```
<ipython-input-19-38ba08b9ae9c>:24: FutureWarning: The behavior of DataFrame
concatenation with empty or all-NA entries is deprecated. In a future versio
n, this will no longer exclude empty or all-NA columns when determining the r
esult dtypes. To retain the old behavior, exclude the relevant entries before
the concat operation.
  df_outliers = pd.concat((df_outliers, df_new), axis=0, ignore_index = True)
```

Out[ ]:

	Feature	Mild outliers	Percentage of mild outliers	Extreme outliers	Percentage of extreme outliers	PHA = Y	PHA = N
1	q	7695	0.842857	78	0.008544	25	53
3	ad	13209	1.446823	1405	0.153894	89	1316
2	i	28105	3.078428	1287	0.140969	118	1169
4	n	13492	1.477821	3628	0.397386	860	2768
0	e	17293	1.894156	5139	0.562891	1198	3941

I will not continue treating outliers on the remaining columns, as this would imply discarding some of the few positive records we have, which would be more harmful.

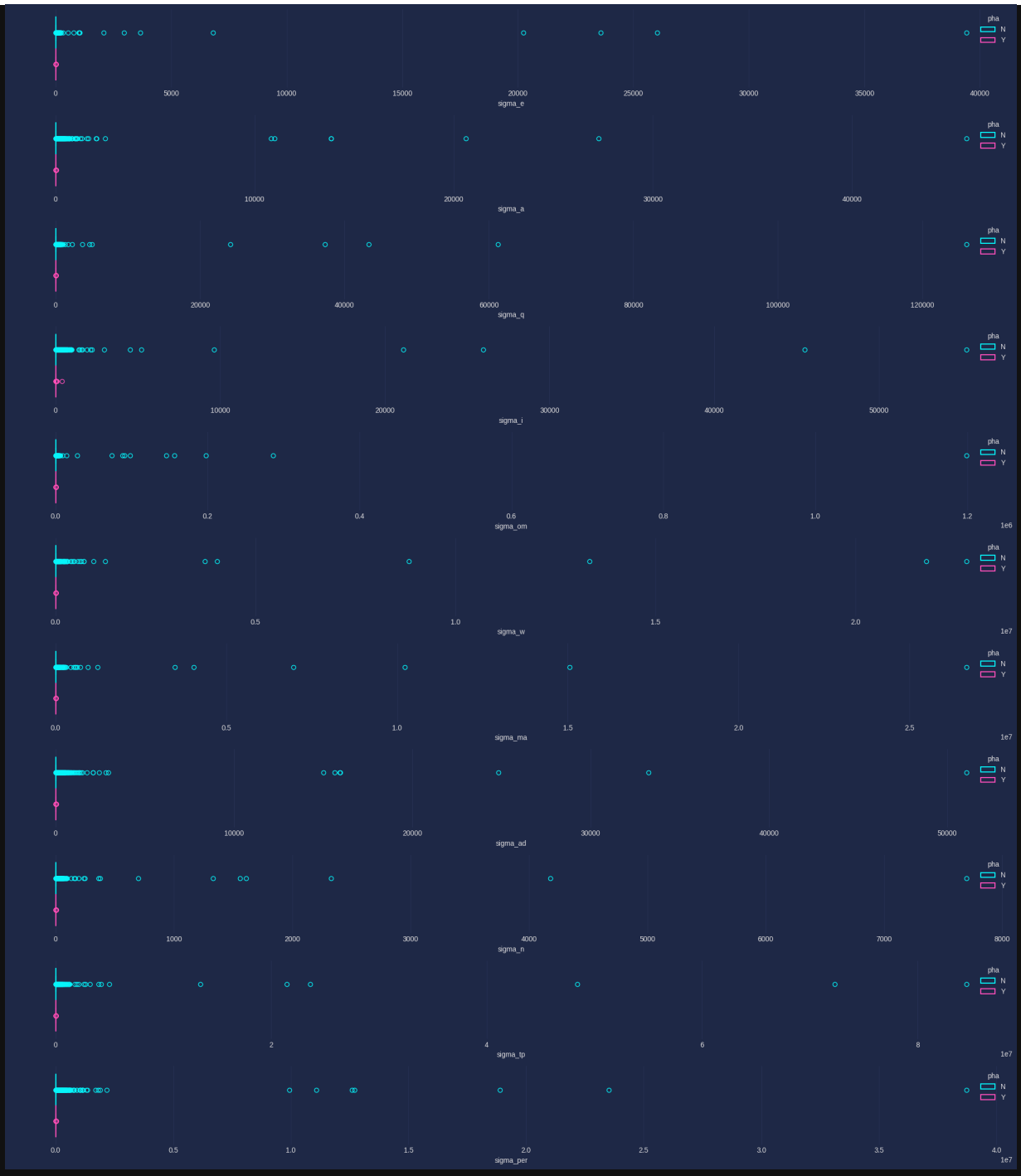
In [ ]:
```
print(f"A total of {old_len - len(df_asteroids)} records have been removed f
```

```
A total of 25637 records have been removed from the original 938603 records.
These records represent 2.73% of our dataset.
```

Uncertainty in measurements

Our dataset contains features that indicate the uncertainty considered during the measurement of a given parameter. A high uncertainty implies a very low confidence in the measurements, and vice versa. To maintain reliable records, we can discard those records with high uncertainties.

In [ ]:
```
adjust_display()

columns_sigma = ["sigma_e", "sigma_a", "sigma_q", "sigma_i", "sigma_om", "si

fig = plt.figure(figsize=(20, 25))
axs = fig.subplots(len(columns_sigma), 1)
for i, col in enumerate(columns_sigma):
  sns.boxplot(data=df_asteroids, x=col, ax=axs[i], hue="pha", fill=False)
fig.tight_layout(rect=[0, 0.03, 1, 0.95])
```



In [ ]:
```
table_outliers(df_asteroids, columns_sigma).sort_values(by=["PHA = Y", "PHA
```

```
<ipython-input-19-38ba08b9ae9c>:24: FutureWarning: The behavior of DataFrame
concatenation with empty or all-NA entries is deprecated. In a future versio
n, this will no longer exclude empty or all-NA columns when determining the r
esult dtypes. To retain the old behavior, exclude the relevant entries before
the concat operation.
  df_outliers = pd.concat((df_outliers, df_new), axis=0, ignore_index = True)
```

Out[ ]:

	Feature	Mild outliers	Percentage of mild outliers	Extreme outliers	Percentage of extreme outliers	PHA = Y	PHA = N
5	sigma_w	169703	18.588096	151966	16.645308	328	151638
4	sigma_om	133127	14.581814	108035	11.833409	367	107668
2	sigma_q	186220	20.397255	172997	18.948898	444	172553
9	sigma_tp	175296	19.200715	159219	17.439751	481	158738
6	sigma_ma	177963	19.492840	162262	17.773061	566	161696
10	sigma_per	171498	18.784708	158174	17.325289	606	157568
3	sigma_i	152375	16.690107	133217	14.591672	617	132600
1	sigma_a	172912	18.939588	159599	17.481374	628	158971
0	sigma_e	188276	20.622455	175568	19.230508	635	174933
7	sigma_ad	174106	19.070371	160759	17.608432	652	160107
8	sigma_n	178137	19.511899	165385	18.115132	802	164583

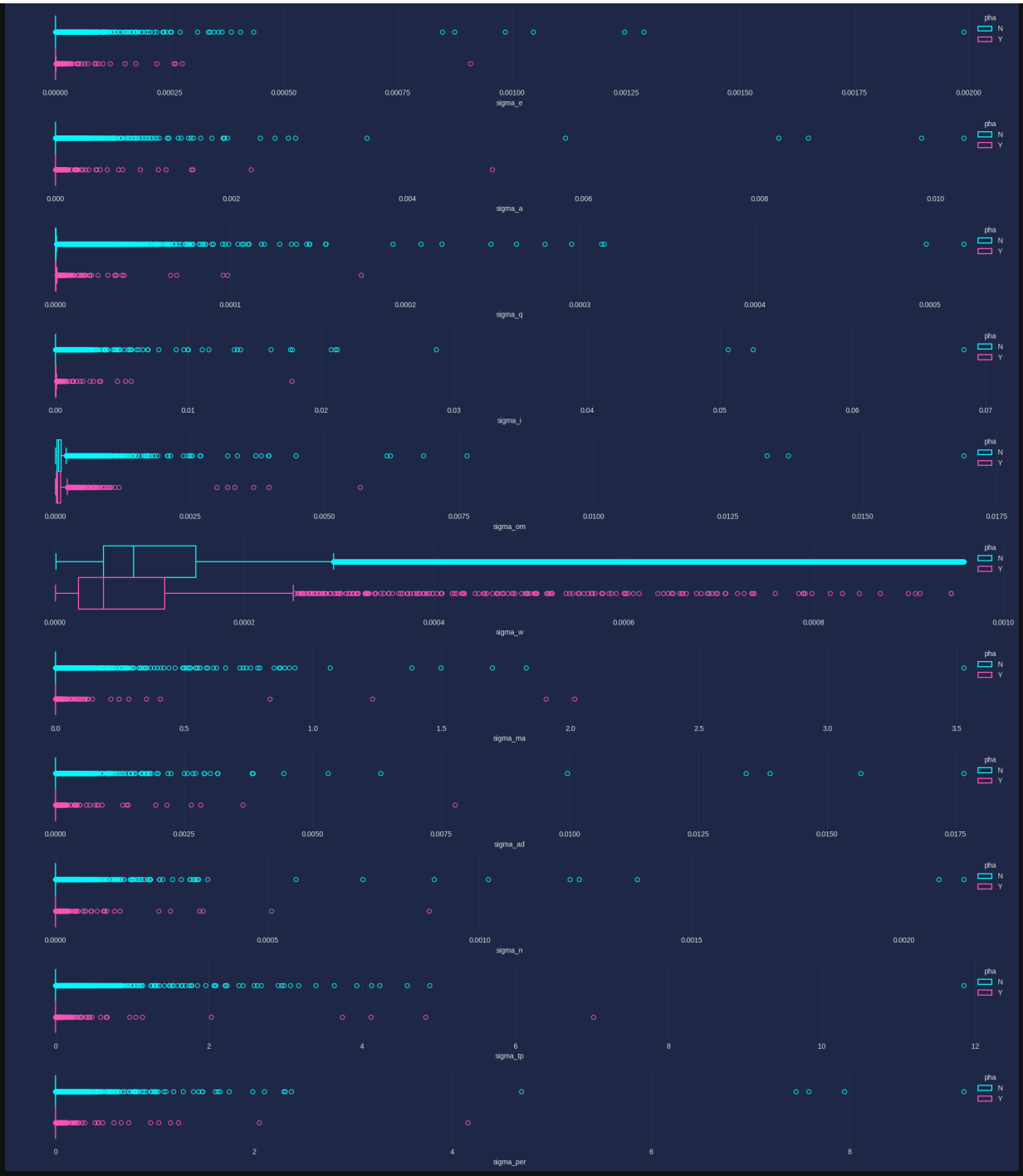
In [ ]:
```
sigw_outliers = get_outliers(df_asteroids, "sigma_w", "extremos")
df_asteroids.drop(sigw_outliers.index, axis=0, inplace=True)
columns_sigma.remove("sigma_w")
table_outliers(df_asteroids, columns_sigma).sort_values(by=["PHA = Y", "PHA
```

```
<ipython-input-19-38ba08b9ae9c>:24: FutureWarning: The behavior of DataFrame
concatenation with empty or all-NA entries is deprecated. In a future versio
n, this will no longer exclude empty or all-NA columns when determining the r
esult dtypes. To retain the old behavior, exclude the relevant entries before
the concat operation.
  df_outliers = pd.concat((df_outliers, df_new), axis=0, ignore_index = True)
```

Out[ ]:

	Feature	Mild outliers	Percentage of mild outliers	Extreme outliers	Percentage of extreme outliers	PHA = Y	PHA = N
4	sigma_om	65317	8.583049	29657	3.897109	114	29543
8	sigma_tp	79213	10.409067	47108	6.190276	226	46882
2	sigma_q	82686	10.865440	56256	7.392378	253	56003
9	sigma_per	64615	8.490802	38907	5.112615	345	38562
5	sigma_ma	81203	10.670565	50174	6.593167	353	49821
1	sigma_a	64868	8.524047	39642	5.209198	377	39265
6	sigma_ad	66274	8.708804	41100	5.400788	436	40664
0	sigma_e	85925	11.291064	59611	7.833246	478	59133
3	sigma_i	53623	7.046386	25237	3.316294	524	24713
7	sigma_n	67878	8.919580	43889	5.767280	640	43249

In [ ]:
```
adjust_display()

columns_sigma = ["sigma_e", "sigma_a", "sigma_q", "sigma_i", "sigma_om", "si

fig = plt.figure(figsize=(20, 25))
axs = fig.subplots(len(columns_sigma), 1)
for i, col in enumerate(columns_sigma):
  sns.boxplot(data=df_asteroids, x=col, ax=axs[i], hue="pha", fill=False)
fig.tight_layout(rect=[0, 0.03, 1, 0.95])
```



In [ ]:
```
print(f"A total of {old_len - len(df_asteroids)} records have been removed f
```

```
A total of 177603 records have been removed from the original 938603 records.
These records represent 18.92% of our dataset.
```

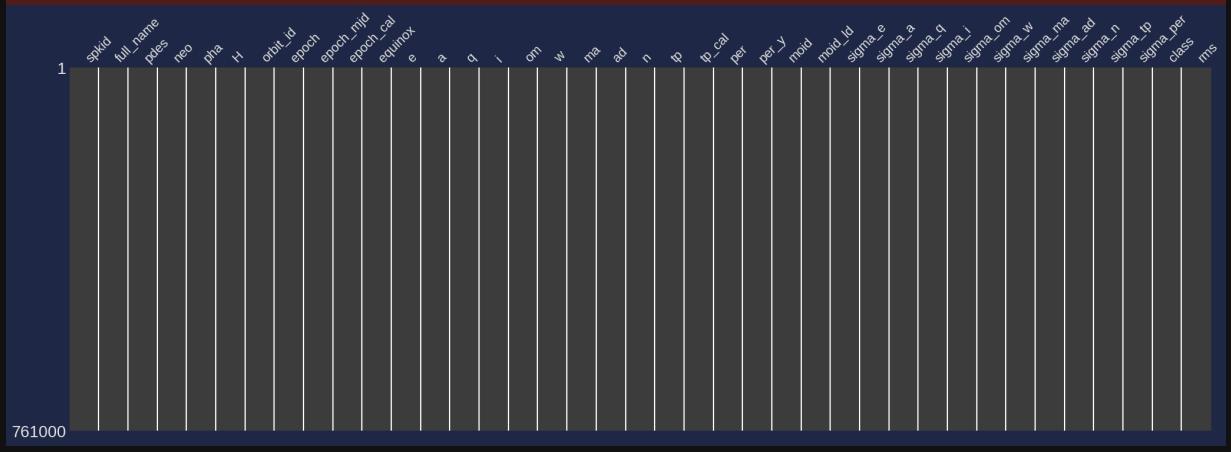
In [ ]:
```
gc.collect()
```

Out[ ]: 39023

Replacing missing data

In [ ]:
```
_, ax = plt.subplots(1, 1, figsize=(25, 8))
msno.matrix(df_asteroids, ax=ax);
```

```
/usr/local/lib/python3.11/dist-packages/missingno/missingno.py:61: UserWarnin
g: Plotting a sparkline on an existing axis is not currently supported. To re
move this warning, set sparkline=False.
  warnings.warn(
```



In [ ]:
```
df_asteroids.isna().sum()
```

Out[ ]:

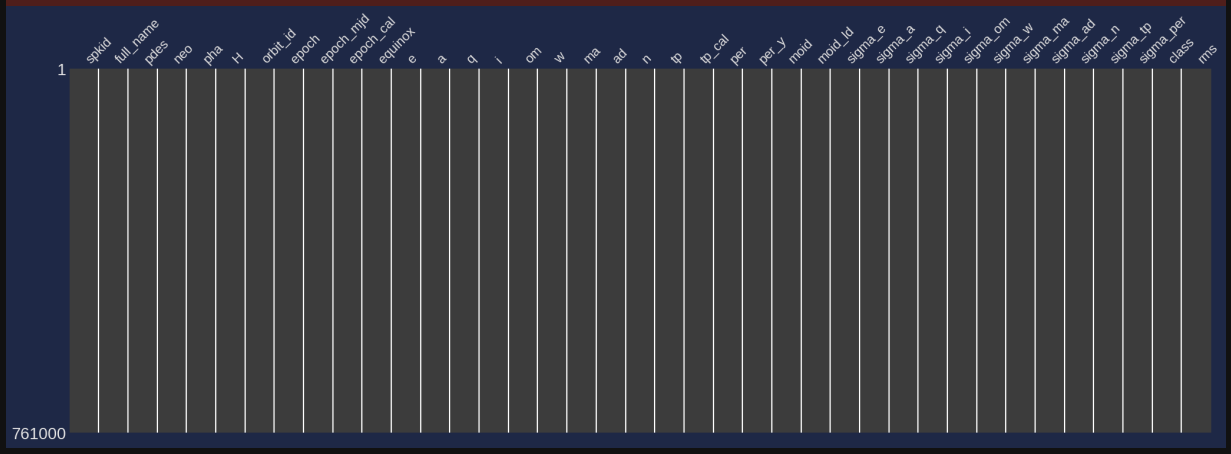
	0
spkid	0
full_name	0
pdes	0
neo	0
pha	0
H	0
orbit_id	0
epoch	0
epoch_mjd	0
epoch_cal	0
equinox	0
e	0
a	0
q	0
i	0
om	0
w	0
ma	0
ad	0
n	0
tp	0
tp_cal	0
per	0
per_y	0
moid	0
moid_ld	0
sigma_e	0
sigma_a	0
sigma_q	0
sigma_i	0
sigma_om	0
sigma_w	0

	0
sigma_ma	0
sigma_ad	0
sigma_n	0
sigma_tp	0
sigma_per	0
class	0
rms	0

dtype: int64

In [ ]:
```
si = SimpleImputer(missing_values=np.nan, strategy='mean')
df_asteroids[df_asteroids.select_dtypes(['number']).columns] = si.fit_transfo
_, ax = plt.subplots(1, 1, figsize=(25, 8))
msno.matrix(df_asteroids, ax=ax);
```

```
/usr/local/lib/python3.11/dist-packages/missingno/missingno.py:61: UserWarnin
g: Plotting a sparkline on an existing axis is not currently supported. To re
move this warning, set sparkline=False.
  warnings.warn(
```



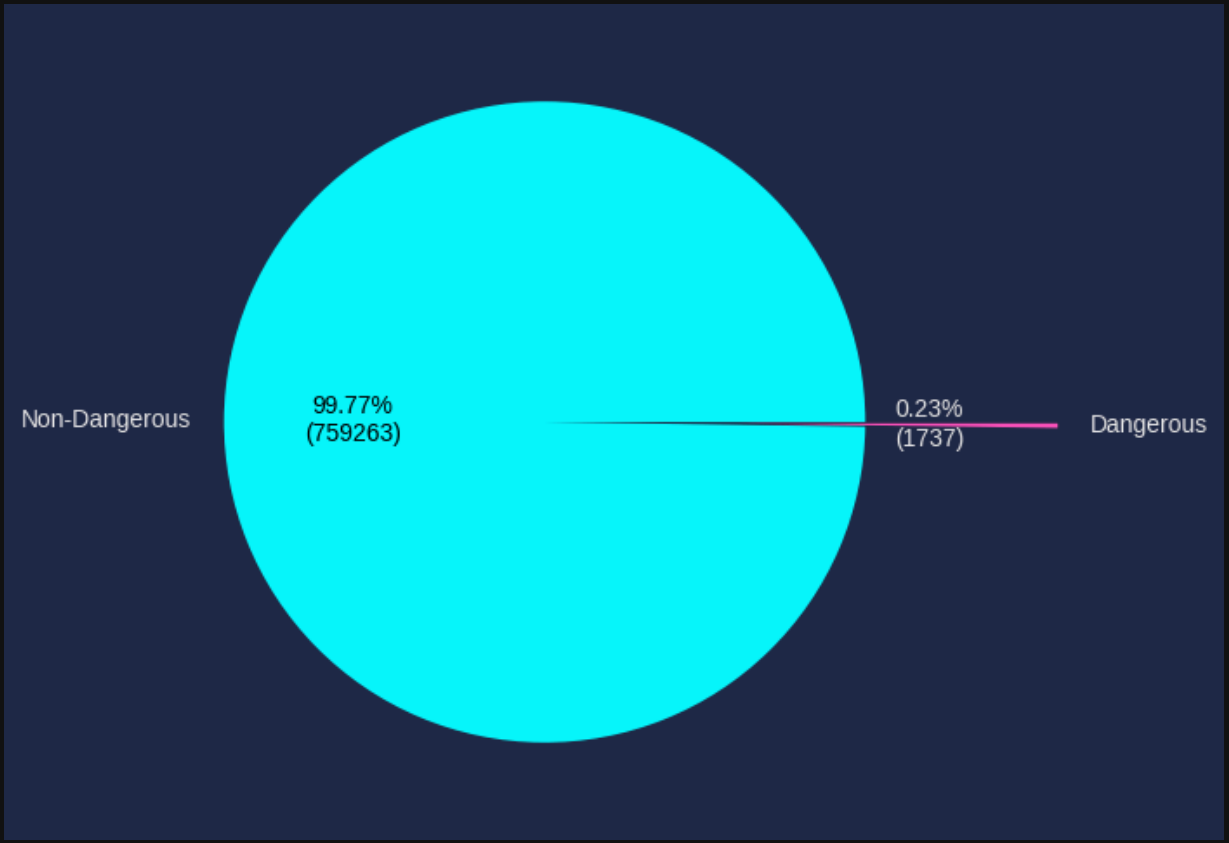
📊 Data visualization

⚖️ Balance of categorical data

By `PHA`

In [ ]:
```
def autopct_format(values):
        def my_format(pct):
            total = sum(values)
            val = int(round(pct*total/100.0))
            return '{:.2f}%\n({v:d})'.format(pct, v=val)
        return my_format
```

```
adjust_display()

cmap = mpl.colormaps['Paired']
plt.figure(figsize=(6, 6))
pha_counts = df_asteroids["pha"].value_counts()
_, _, autotexts = plt.pie(pha_counts, labels=["Non-Dangerous", "Dangerous"],
autotexts[0].set_color('black')
```



As we can observe, our dataset is highly imbalanced with respect to the `target`, meaning we have many records of non-hazardous asteroids compared to the few records of hazardous asteroids. This can pose a problem, and later we will explore how to address it.

By `NEO`, `ORBIT_ID`, `EQUINOX` and `CLASS`

In [ ]:
```
adjust_display()
fig = plt.figure(figsize=(20, 8))
axs = fig.subplots(3, 1)
for i, col in enumerate(["neo", "equinox", "class"]):
  sns.countplot(data=df_asteroids, y=col, ax=axs[i], order=df_asteroids[col]
  for container in axs[i].containers:
    axs[i].bar_label(container, fontsize=7)
mplcyberpunk.add_glow_effects()
```

We have 3 charts, and we'll discuss them briefly:

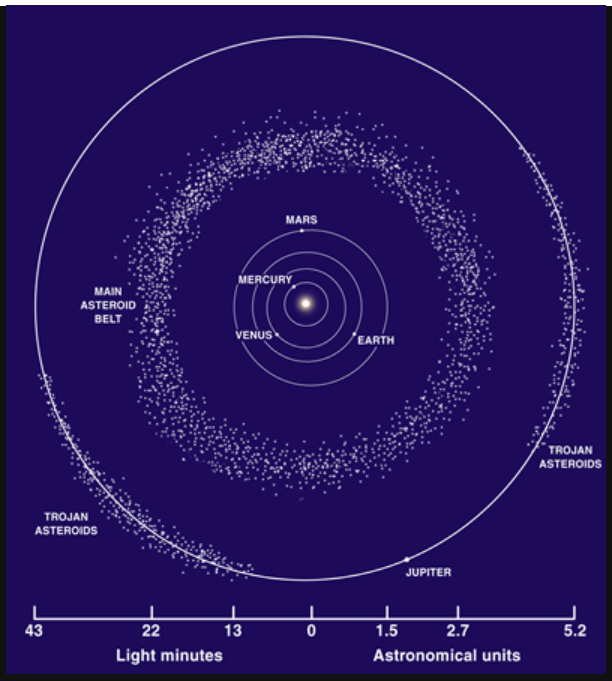
- `neo` is a parameter that defines whether an asteroid is a Near-Earth Object (NEO). As expected, most asteroids are not near the Earth, and this aligns with what we saw earlier. We had observed that there is a large percentage of asteroids that are not dangerous to Earth, so it's normal that most of these asteroids are far from Earth. However, this does not imply that an asteroid far from Earth cannot be potentially dangerous.

- The first thing to highlight is that all the values of `equinox` are `J2000`, so if all values are the same, this feature is not very useful at first glance. But after doing some research, I found the following on this Wikipedia page:

  > *In astronomy, **J2000.0** refers to the Julian date 2451545.0 TT (Terrestrial Time), or January 1, 2000, noon TT. It is equivalent to January 1, 2000, 11:59:27.816 TAI, or January 1, 2000, 11:58:55.816 UTC.*
  >
  > *This date is widely used to indicate a standard instant in time for measuring the positions of celestial bodies and other stellar events. For example, although imperceptible to the naked eye, stars move through space, and it is necessary, to describe their position in the sky, to specify the date to which that position refers. As we can see, most of the asteroids for which we have records are not close to Earth.*

  If we observe our dataset, for example, we'll see that we have 3 features named `epoch`, `epoch_mjd`, and `epoch_cal`, which are times associated with sets of osculating elements, and along with `equinox`, they indicate that they are within the **J2000** reference frame.

- On the other hand, we can observe that most of the asteroids are classified as MBA (Main-belt Asteroid), meaning they are asteroids with orbital elements restricted by (2.0 AU < a < 3.2 AU; q > 1.666 AU).



But what are AU? They are Astronomical Units, created because distances in the Solar System are very large. To compare the average distances between the Sun and the planets, it is more convenient to express them in terms of the average separation between the Earth and the Sun.

1 AU = average distance between the sun and the Earth = 1.496 × 10^8 km

<img src=https://astronomy.swin.edu.au/cms/cpg15x/albums/userpics/AU.gif)

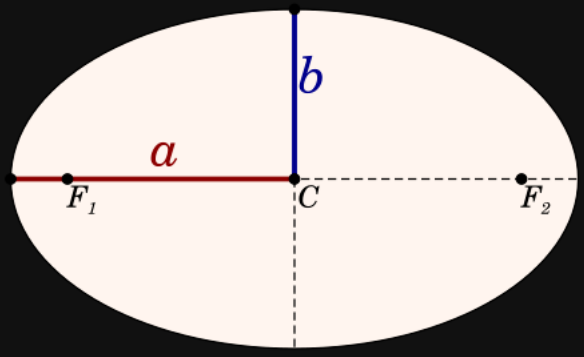
After researching asteroid classification, I found the following table:

Abbreviation	Title	Description
AMO	Amor	Near-Earth asteroid, orbit similar to 1221 Amor (a > 1.0 AU; 1.017 AU < q < 1.3 AU).
APO	Apollo	Near-Earth asteroids with orbits that cross Earth's orbit, similar to 1862 Apollo (a > 1.0 AU; q < 1.017 AU).
AST	Asteroid	The asteroid's orbit does not match any defined class.
ATE	Aten	Near-Earth asteroid, orbit similar to 2062 Aten (a < 1.0 AU; Q > 0.983 AU).
CEN	Centaur	Objects with orbits between Jupiter and Neptune (5.5 AU < a < 30.1 AU).
HYA	Hyperbolic Asteroid	Asteroids in hyperbolic orbits (e > 1.0).
IEO	Interior Earth Object	An asteroid's orbit entirely contained within Earth's orbit (Q < 0.983 AU).
IMB	Inner Main-belt Asteroid	Asteroids with orbital elements restricted by (a < 2.0 AU; q > 1.666 AU).
MBA	Main-belt Asteroid	Asteroids with orbital elements restricted by (2.0 AU < a < 3.2 AU; q > 1.666 AU).

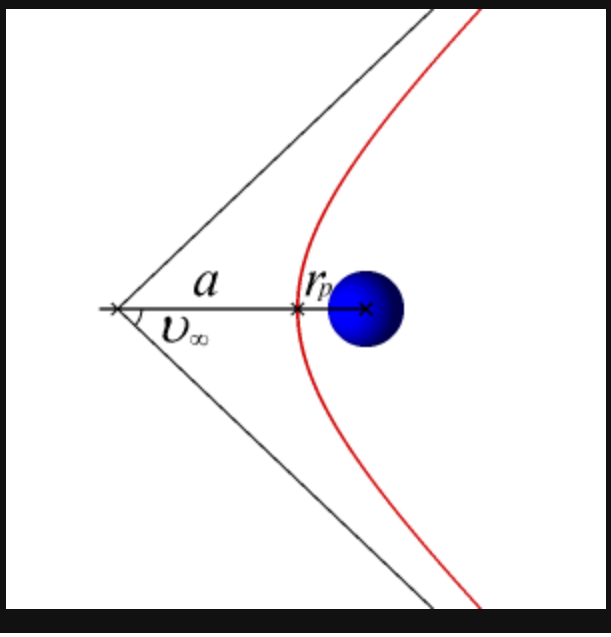
Abbreviation	Title	Description
MCA	Mars-crossing Asteroid	Asteroids that cross Mars's orbit, restricted by (1.3 AU < q < 1.666 AU; a < 3.2 AU).
OMB	Outer Main-belt Asteroid	Asteroids with orbital elements restricted by (3.2 AU < a < 4.6 AU).
PAA	Parabolic Asteroid	Asteroids in parabolic orbits (e = 1.0).
TJN	Jupiter Trojan	Asteroids trapped in Jupiter's Lagrange points L4/L5 (4.6 AU < a < 5.5 AU; e < 0.3).
TNO	TransNeptunian Object	Objects with orbits beyond Neptune (a > 30.1 AU).

Where `a` is the semi-major axis of the orbit, `q` is the perihelion distance, `Q` or `ad` is the aphelion distance, and `e` is the orbital eccentricity. Let's see what these parameters are.

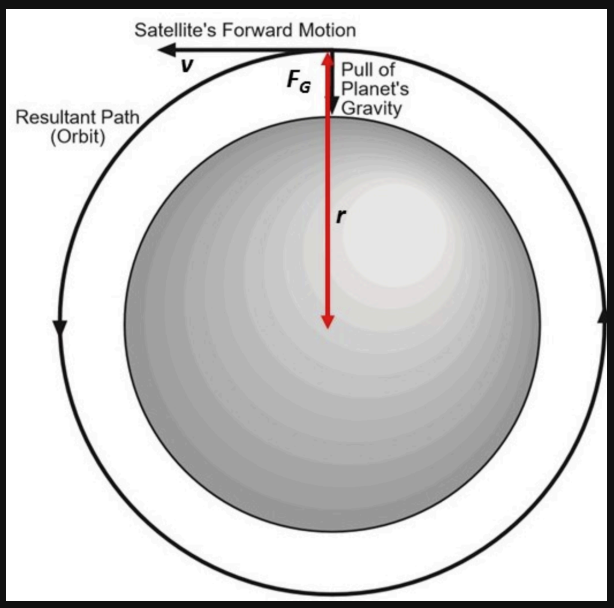
Generally, an orbit follows an elliptical shape (elliptical orbit), and the semi-major axis `a` of an ellipse is as follows:



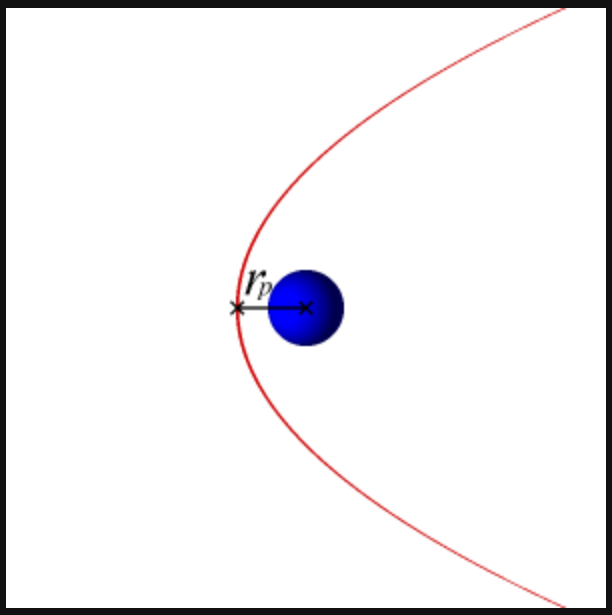
In these cases, `a` > 0.
An orbit can also follow a hyperbolic shape (hyperbolic orbit), where `a` is the distance between the center of the hyperbola and one of its vertices:



Therefore, `a` can be negative or positive.
There are also two special cases of an ellipse: if `a` is equal to the radius of the orbit, we are in the presence of a circular orbit:

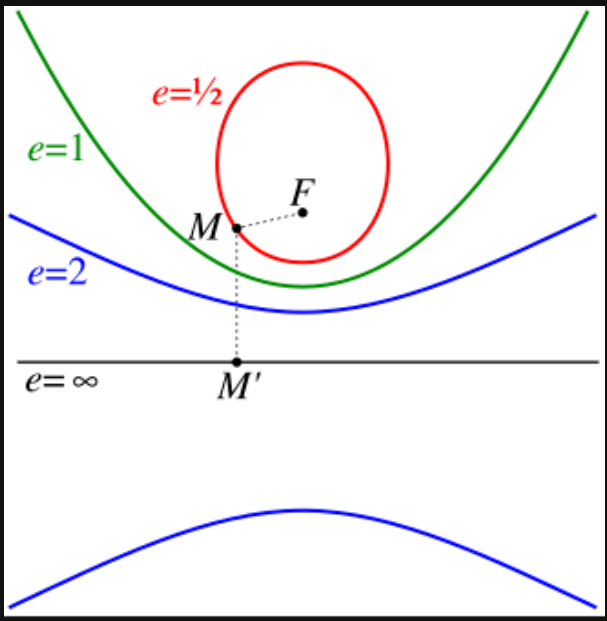


And if `a` is infinitely large, we are in the presence of a parabolic orbit, which is the limiting case between the elliptical orbit (closed) and the hyperbolic orbit (open).



The aphelion is the point in an orbit that is furthest from the Sun (`Q`), and on the other side, the perihelion is the point in an orbit closest to the Sun (`q`):

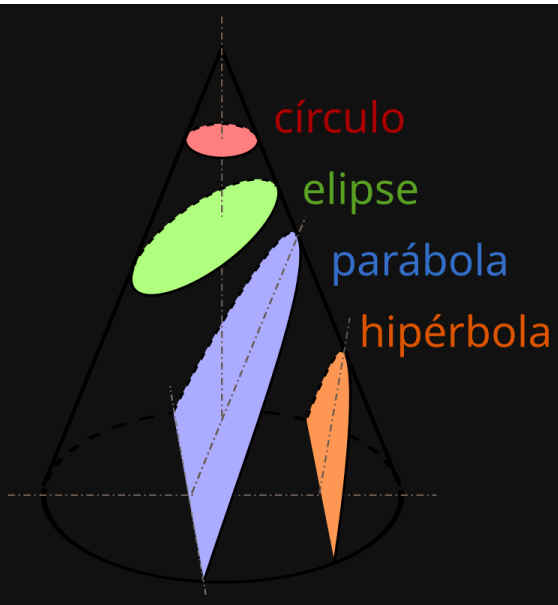
Lastly, eccentricity `e` is a parameter that determines the degree of deviation of a conic section from a circle. Therefore, it defines the shape of the celestial body's orbit:

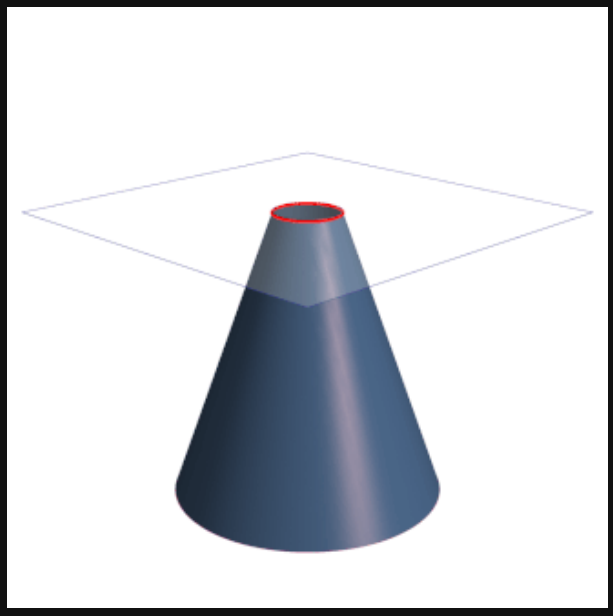


Based on the eccentricity of the orbit, it can be classified following this table:

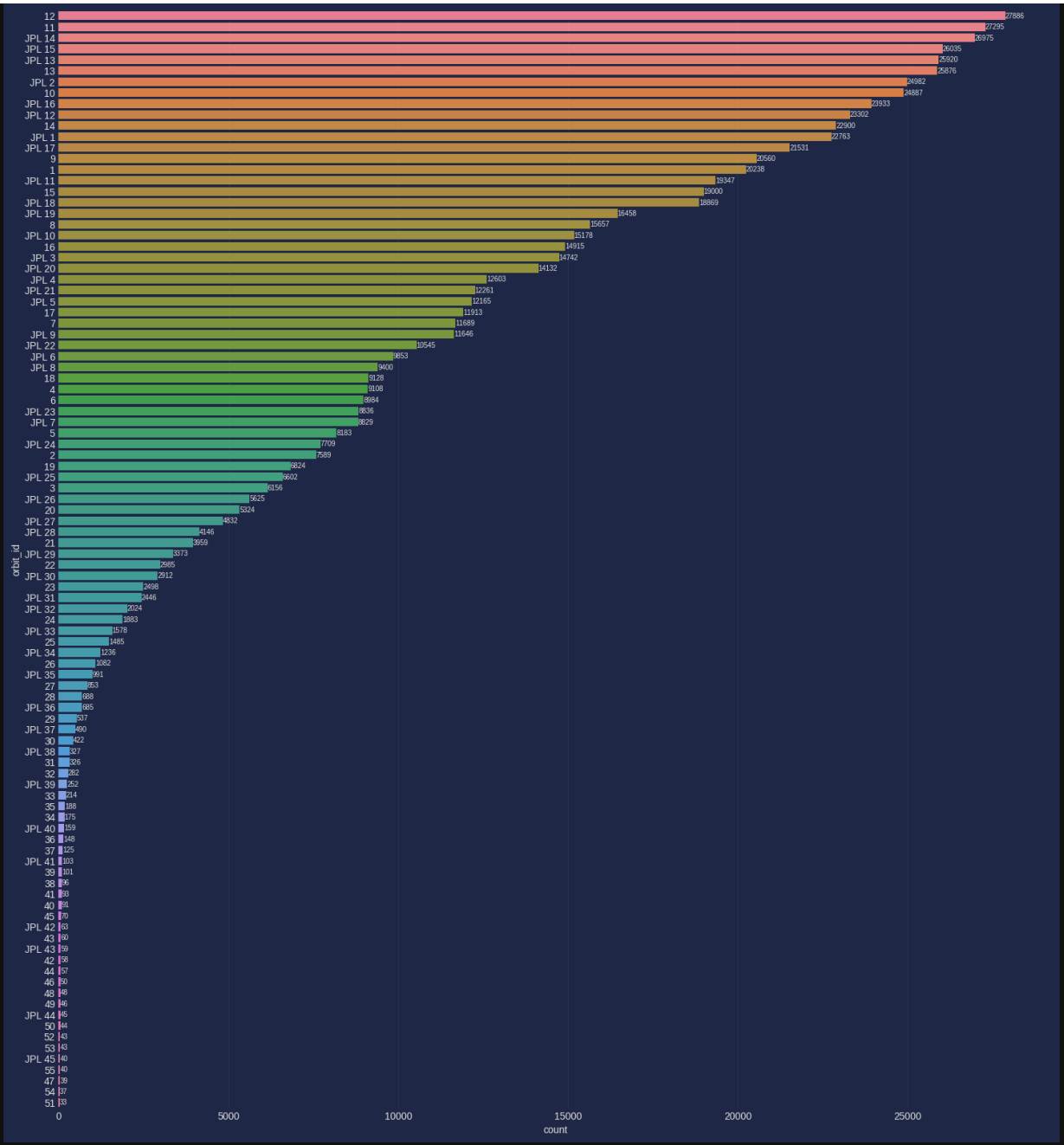
Eccentricity	Orbit Type
0 < e < 1	Elliptical Orbit
e = 0	Circular Orbit
e = 1	Parabolic Orbit
e > 1	Hyperbolic Orbit

And visually, they look like this:





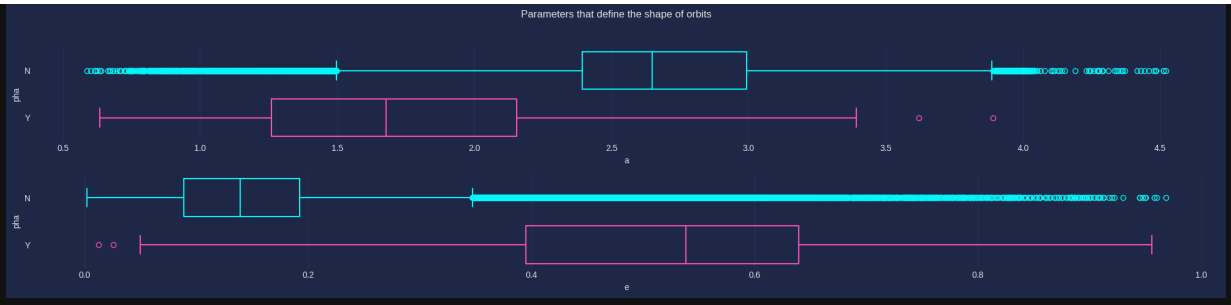
In [ ]:
```
aux = df_asteroids["orbit_id"].value_counts().iloc[:100].reset_index()

adjust_display()
fig = plt.figure(figsize=(18, 20))
ax = sns.barplot(data=aux, y="orbit_id", x="count", hue="orbit_id")
for container in ax.containers:
    ax.bar_label(container, fontsize=7)
```



🔢 Numerical features by `PHA`

Shapes of asteroid orbits according to `a` (semi-major axis of the orbit) and `e` (orbital eccentricity)

In [ ]:
```
adjust_display()
fig = plt.figure(figsize=(20, 5))
axs = fig.subplots(2, 1)
sns.boxplot(data=df_asteroids, y="pha", x="a", ax=axs[0], hue="pha", fill=Fal
sns.boxplot(data=df_asteroids, y="pha", x="e", ax=axs[1], hue="pha", fill=Fal
plt.suptitle("Parameters that define the shape of orbits")
fig.tight_layout(rect=[0, 0.03, 1, 0.95])
```



In [ ]:
```
df_asteroids.loc[ df_asteroids["pha"] == 'Y', ["a", "e"]].describe().transpo
```

Out[ ]:

	count	mean	std	min	25%	50%	75%	max
a	1737.0	1.725794	0.570476	0.635237	1.260581	1.677777	2.154868	3.891236
e	1737.0	0.516548	0.174132	0.012176	0.395379	0.538288	0.639446	0.955945

In [ ]:
```
df_asteroids.loc[df_asteroids["pha"] == 'N', ["a", "e"]].describe().transpos
```

Out[ ]:

	count	mean	std	min	25%	50%	75%	max
a	759263.0	2.67373	0.347634	0.588650	2.394397	2.648678	2.992177	4.521041
e	759263.0	0.14671	0.079136	0.002157	0.088949	0.138947	0.192300	0.968396

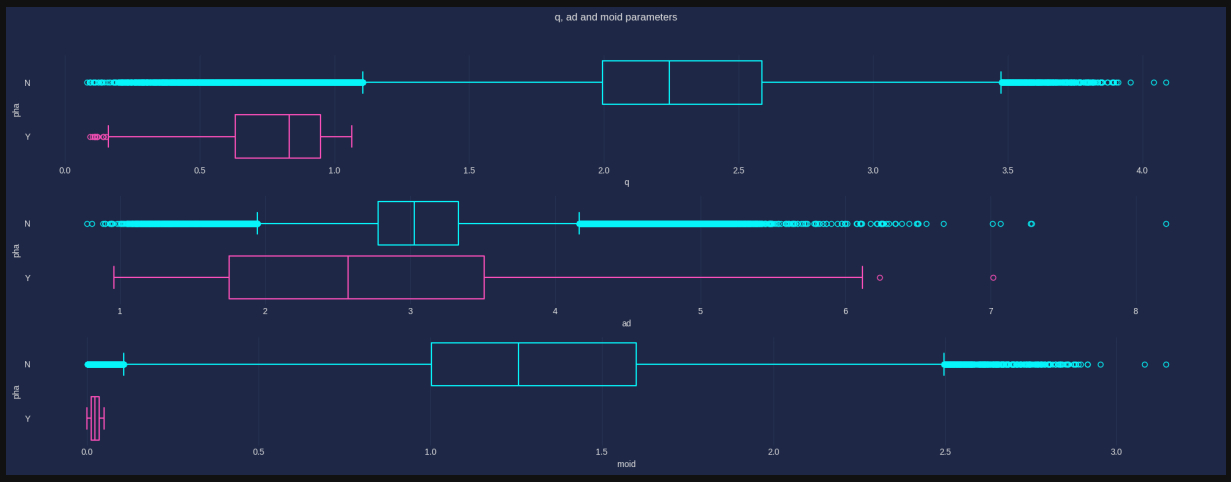
According to these two parameters, we can describe the shape of an asteroid's orbit:

- As we can see, the mean values of **potentially hazardous asteroids (PHA)** indicate that they follow an **elliptical orbit** (0 < e < 1), with a **mean semi-major axis (`a`) tending to 1.77 AU**. This suggests that the farthest point of the asteroid's orbit remains relatively close to its center. If we consider the center of the orbit as the position of the **Sun**, then these asteroids are **very close to Earth's orbit**.
- **Non-hazardous asteroids**, on the other hand, have **elliptical orbits with a tendency toward circular orbits** (e → 0). Their **mean semi-major axis is approximately 2.67 AU**, indicating that they are **farther from the Sun** than potentially hazardous asteroids. As a result, their distance from Earth is also greater.

Distances to the Sun (`q`, `ad`) and intersection with Earth's orbit (`moid`)

In [ ]:
```
adjust_display()
fig = plt.figure(figsize=(20, 8))
axs = fig.subplots(3, 1)
Y = df_asteroids[df_asteroids["pha"] == "Y"]
N = df_asteroids[df_asteroids["pha"] == "N"]
for i, col in enumerate(["q", "ad", "moid"]):
  sns.boxplot(data=df_asteroids, x=col, y="pha", ax=axs[i], hue="pha", fill=F
```

```
plt.suptitle("q, ad and moid parameters")
fig.tight_layout(rect=[0, 0.03, 1, 0.95])
```



In [ ]:
```
df_asteroids.loc[ df_asteroids["pha"] == 'Y', ["q", "ad", "moid"]].describe(
```

Out[ ]:

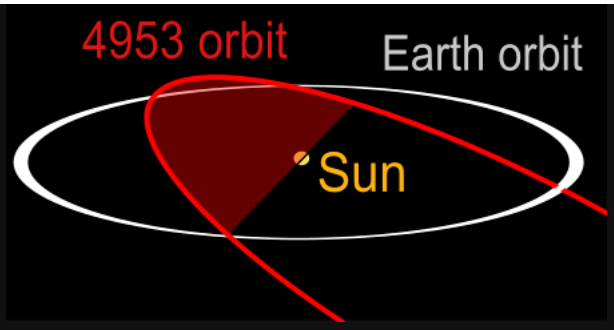
	count	mean	std	min	25%	50%	75%	max
q	1737.0	0.772882	0.216736	0.092924	0.632853	0.833029	0.947844	1.064284
ad	1737.0	2.678706	1.086418	0.956053	1.754060	2.572915	3.511260	7.016173
moid	1737.0	0.023482	0.014392	0.000027	0.011055	0.023233	0.035039	0.049991

In [ ]:
```
df_asteroids.loc[ df_asteroids["pha"] == 'N', ["q", "ad", "moid"]].describe(
```

Out[ ]:

	count	mean	std	min	25%	50%	75%	max
q	759263.0	2.285278	0.385105	0.081820	1.994405	2.243618	2.587117	4.087681
ad	759263.0	3.062181	0.426442	0.773666	2.779532	3.029316	3.334100	8.208638
moid	759263.0	1.299769	0.379235	0.000226	1.002650	1.257760	1.600210	3.143950

- If we look at the graphs, we can conclude that both the perihelion distance (`q`) and the aphelion distance (`ad`) have lower average values for potentially hazardous asteroids compared to non-hazardous ones. This suggests that these asteroids have orbits closer to the Sun, increasing the likelihood of intersecting Earth's orbit.
- To confirm this observation, we can analyze the `moid` variable, which represents the minimum orbit intersection distance (MOID). It is defined as the shortest distance between the closest points of both orbits:



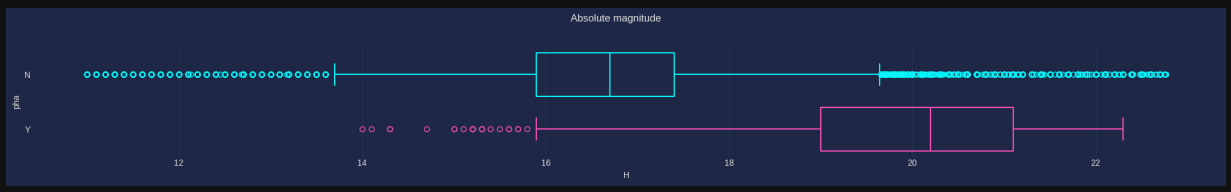
> Orbit of asteroid (4953) 1990 MU in relation to Earth's orbit. Its MOID of 0.0276 AU classifies it as a potentially hazardous object.

Observing this variable, it is expected that potentially hazardous asteroids have lower `moid` values compared to non-hazardous ones, which is precisely what we observe.

Asteroid diameters according to the `H` variable and the `albedo`

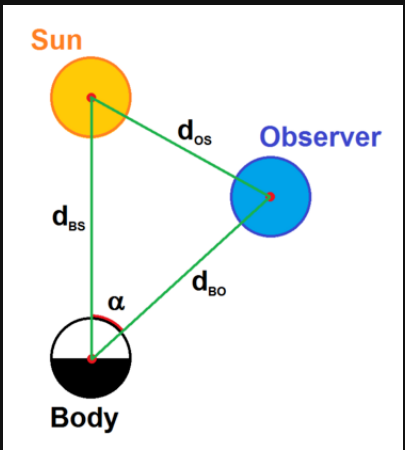
In [ ]:
```
adjust_display()
fig = plt.figure(figsize=(20, 3))
sns.boxplot(data=df_asteroids, y="pha", x="sigma_e", hue="pha", fill=False)
plt.suptitle("Eccentricity")
fig.tight_layout(rect=[0, 0.03, 1, 0.95])
```

In [ ]:
```
adjust_display()
fig = plt.figure(figsize=(20, 3))
sns.boxplot(data=df_asteroids, y="pha", x="H", hue="pha", fill=False)
plt.suptitle("Absolute magnitude")
fig.tight_layout(rect=[0, 0.03, 1, 0.95])
```



- The first point to clarify is that while we do not have data on the `albedo` of the asteroids, since we discarded this data earlier due to its high number of null values, it is generally considered that the albedo falls between 0.25 and 0.05. Based on this, with the absolute magnitude `H`, we can approximate some ranges of asteroid diameters. The absolute magnitude is the magnitude that an observer would see if the asteroid

were at a distance of 1 AU from the Sun, with a phase angle (α) of zero. This angle is formed between the Sun and the Earth as seen from the asteroid:



Once this is understood, we can create the following table:

La magnitud absoluta (H)	Diámetro (km = kilómetros) (m = metros)
15.0	3 km - 6 kilometros
15.5	2 km - 5 kilometros
16.0	2 kilometros - 4 km
16.5	1 km - 3 kilometros
17.0	1 km - 2 kilometros
17.5	1 km - 2 kilometros
18.0	670 m - 1500 m
18.5	530 m - 1200 m
19.0	420 m - 940 m
19.5	330 m - 750 m
20.0	270 m - 590 m
20.5	210 m - 470 m
21.0	170 m - 380 m
21.5	130 m - 300 m
22.0	110 m - 240 m
22.5	85 m - 190 m

From this, we can conclude that the smaller the absolute magnitude, the larger the estimated diameter of the asteroid.

- As mentioned earlier, potentially dangerous asteroids on average have smaller diameters compared to non-dangerous asteroids. This might seem counterintuitive because one would expect larger asteroids to be more dangerous. However, the danger posed by an asteroid is not only dependent on its size, but as we have seen, it mainly depends on the likelihood of it intersecting with Earth's orbit. In other words, an asteroid, no matter how large, does not pose a potential risk if it does not have a chance of colliding with Earth. However, the asteroid's diameter does affect its destructive potential, because if it does collide with Earth, the larger its diameter, the greater the damage it is likely to cause.

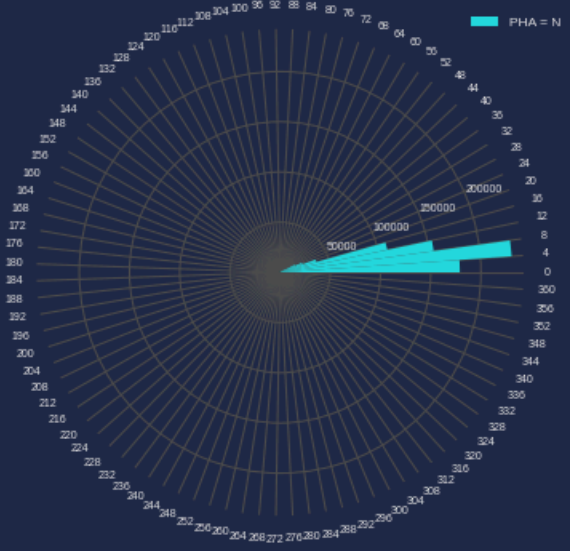
Angular variables

In [ ]:

```
adjust_display()
n=4
theta = np.linspace(0.0, 2 * np.pi, 364//n, endpoint=False)
fig, axs = plt.subplots(nrows=5, ncols=2, figsize=(8, 20), subplot_kw=dict(pol
titles = {'i':"Inclination", 'om':"Longitude of the ascending node", 'w':"Ar
for i, col in enumerate(['i', 'om', 'w', 'ma', 'n']):
  ax = axs[i]
  data = [df_asteroids.loc[df_asteroids["pha"] == 'N', col], df_asteroids.lo

  bars = ax[0].hist(data[0]*np.pi/180, bins=theta, color='#27DADF')
  ax[0].set_xticks(theta)
  ax[0].set_xticklabels(np.arange(0, 364, n))
  ax[0].xaxis.set_tick_params(labelsize=5)
  ax[0].yaxis.set_tick_params(labelsize=5)
  #ax[0].set_thetamax(180)
  ax[0].set_xlabel("Degrees")
  ax[0].set_title(titles[col], y=1.05)
  ax[0].legend(['PHA = N'], bbox_to_anchor=(1.1, 1.05), prop={'size': 6})
  ax[0].grid(color=(0.3, 0.3, 0.3))


  bars = ax[1].hist(data[1]*np.pi/180, bins=theta, color='#E968B6')
  ax[1].set_xticks(theta)
  ax[1].set_xticklabels(np.arange(0, 364, n))
  ax[1].xaxis.set_tick_params(labelsize=5)
  ax[1].yaxis.set_tick_params(labelsize=5)
  #ax[1].set_thetamax(180)
  ax[1].set_xlabel("Degrees")
  ax[1].set_title(titles[col], y=1.05)
  ax[1].legend(['PHA = Y'], bbox_to_anchor=(1.1, 1.05), prop={'size': 6})
  ax[1].grid(color=(0.3, 0.3, 0.3))
plt.suptitle("Angular variables", y=1.0001)
fig.tight_layout()
```

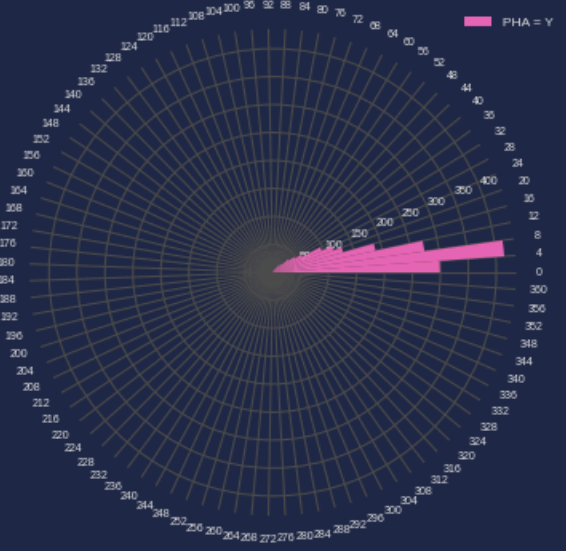
Angular variables

Inclination



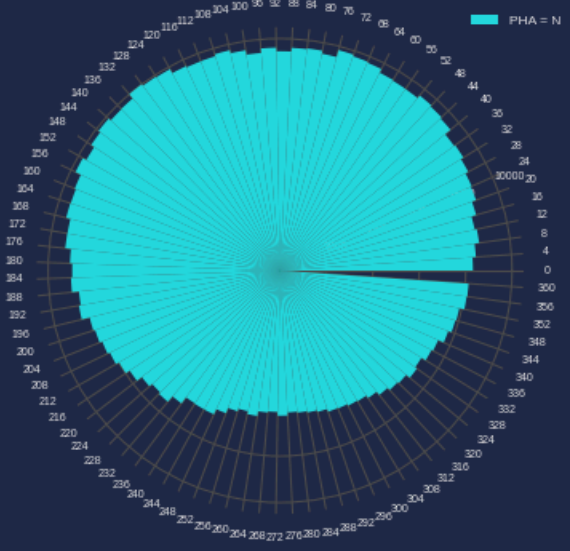
Degrees

Inclination



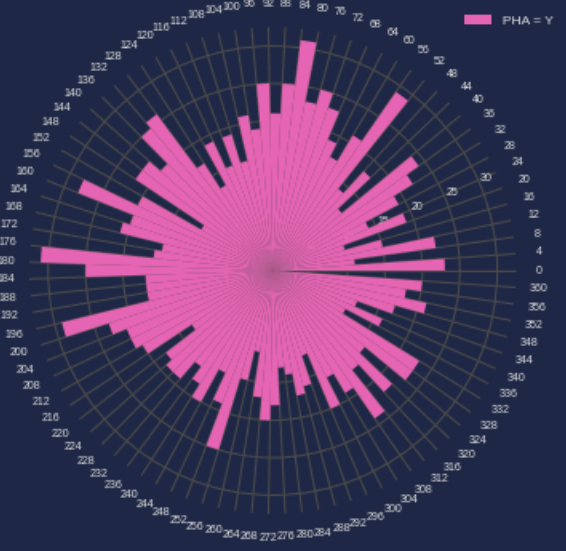
Degrees

Longitude of the ascending node



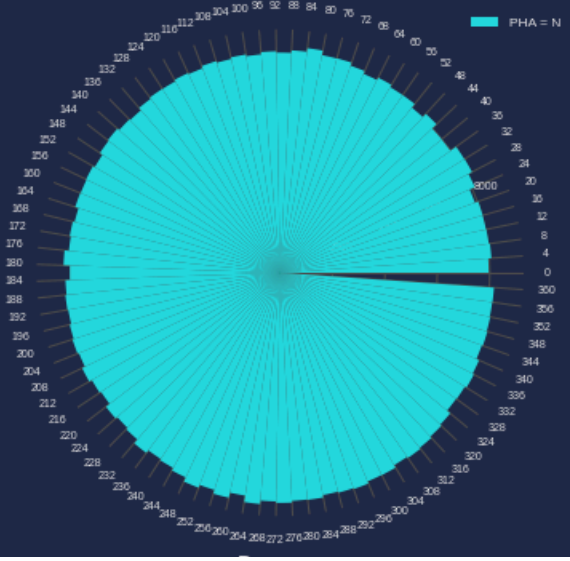
Degrees

Longitude of the ascending node

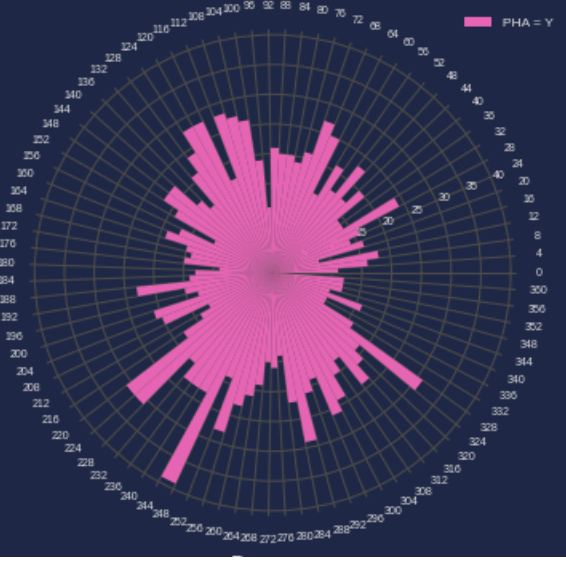


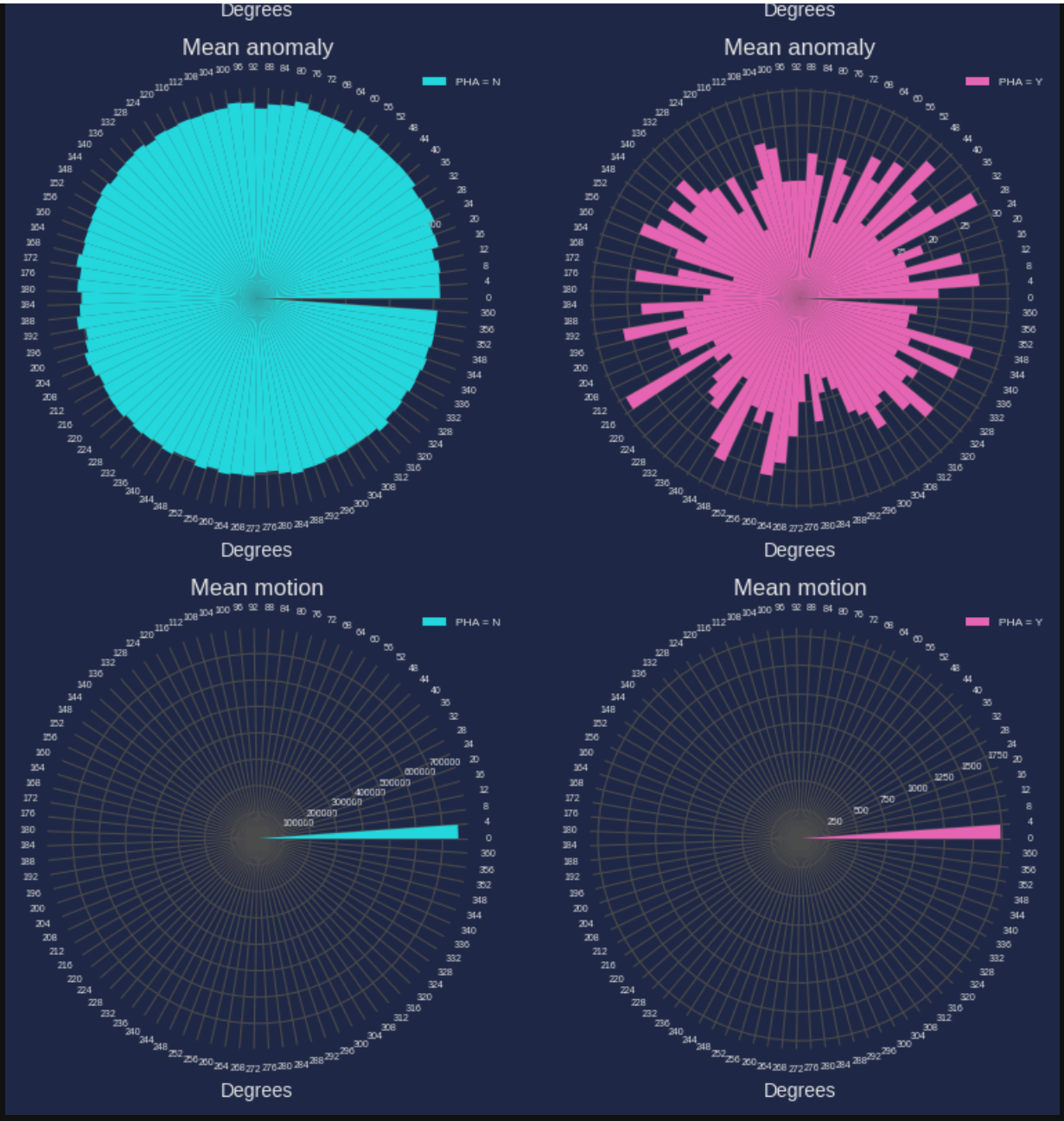
Degrees

Argument of perihelion

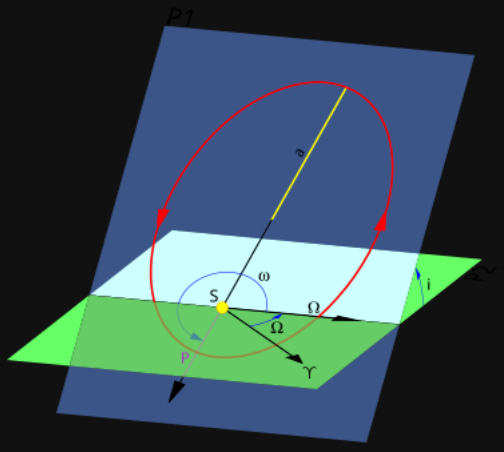


Argument of perihelion

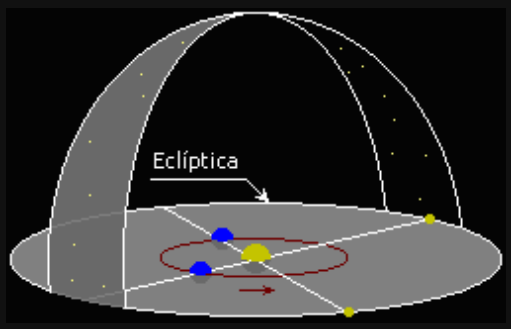




- The `i` variable is the inclination of the asteroid's orbit relative to the ecliptic plane.

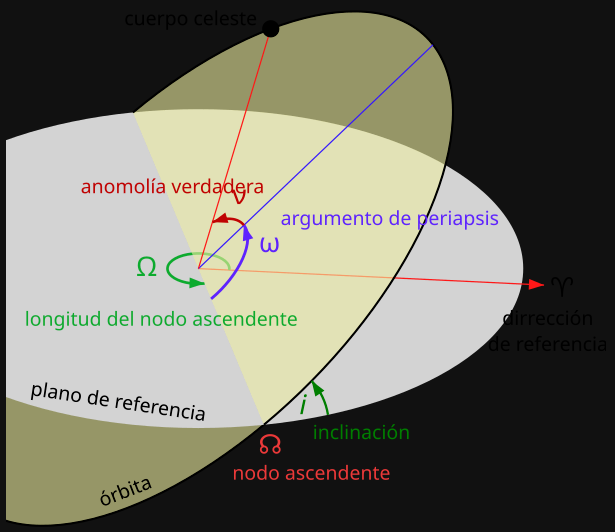


In this diagram, we can also observe the variables `om` and `w`, which we will discuss shortly. The ecliptic plane is the plane defined by the Earth's orbit around the Sun, i.e., it is the plane that the Earth's orbit defines.



From the graph, we can see that the mean value (~9°) of this variable does not differ between potentially dangerous asteroids and those that are not.

- The longitude of the ascending node `om` shows quite a random behavior for potentially dangerous asteroids. However, for asteroids that are not dangerous, we observe a slight inclination towards values between `0° < om < 180°`. The longitude of the ascending node is the angle from a reference direction, called the origin of longitude, to the direction of the ascending node, measured in a reference plane, as shown in the adjacent image:



- The argument of perihelion `w` is the angle from the ascending node to the perihelion, measured in the orbital plane of the object and in the direction of its motion. As we can see, this characteristic shows asteroids distributed evenly between 0° and 360°. However, for potentially dangerous asteroids, there is a slight inclination towards values between 180° and 360°.

- The mean anomaly `ma` is the angle measured from the perihelion, in the direction of motion, at the moment of the epoch. It is defined as the product of the difference between the current time and the epoch when it passed through the perihelion and the mean motion `n`.

Plotting the orbits of the asteroids

Orbits of potentially dangerous asteroids and non-dangerous asteroids

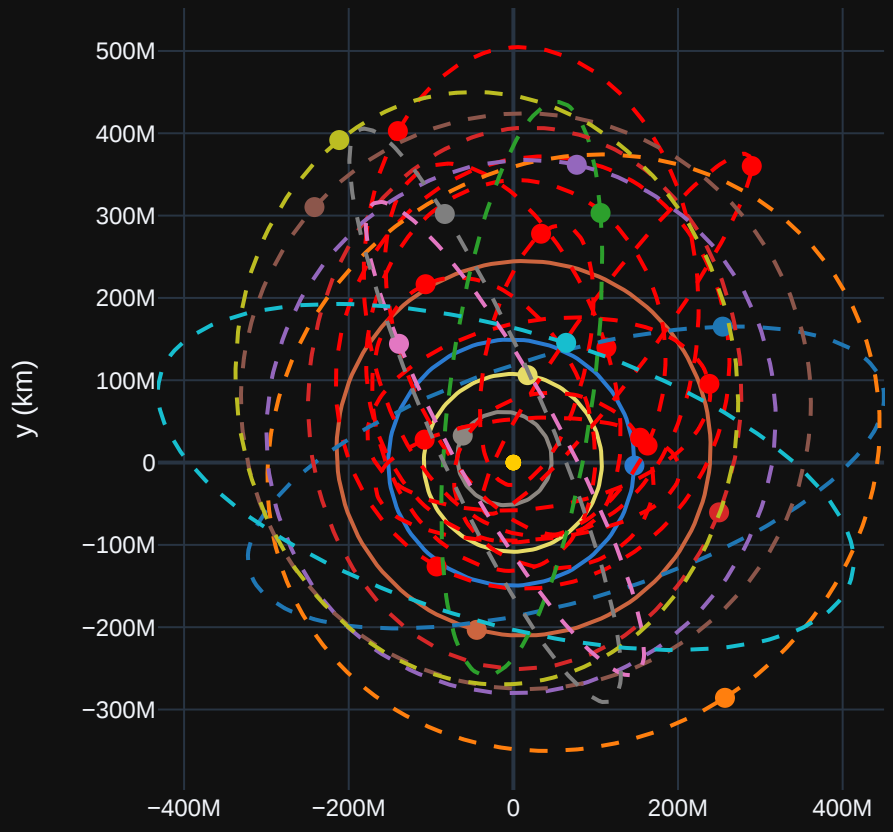
In [ ]:
```
pio.renderers.default = "svg" #"colab"
```

In [ ]:
```
def au2km(x):
  return x*149597870.7

def asteroid_orbit_from_orbital_elements(df, index, object_center="sun"):
  asteroid = df_asteroids.loc[index, :]
  ma = asteroid.ma    #mean anomaly
  e = asteroid.e      #eccentricity
  a = au2km(asteroid.a)      #semi-major axis
  nu = ma + (2*e - (e**3)/4)*np.sin(ma) + (5/4)*(e**2)*np.sin(2*ma) + (13/12
  p = a*(1-e**2)  #semi-latus rectum
  if object_center == "sun":
    obc = Sun
    mu = 1.32712E11 #standard gravitational parameter of the Sun
  elif object_center == "earth":
    obc = Earth
    mu = 3.98600E5 #standard gravitational parameter of the Earth
  omega = asteroid.w
  i = asteroid.i
  epoch = Time([asteroid.epoch], format='jd')

  # According to https://downloads.rene-schwarz.com/download/M001-Keplerian_

  # Transform to perifocal frame
  r_w = (p / (1 + e * np.cos(nu))) * np.array((np.cos(nu), np.sin(nu), 0))
  v_w = ((mu/p)**(1/2)) * np.array((-np.sin(nu), e + np.cos(nu), 0))

  # Rotate the perifocal frame
  R = Rotation.from_euler("ZXZ", [-omega, -i, -omega])
  r_rot = r_w @ R.as_matrix()
  v_rot = v_w @ R.as_matrix()

  # Change units
  r = r_rot * u.km
  v = v_rot * u.km / u.s

  return r, v, epoch, asteroid.full_name, Orbit.from_vectors(obc, r, v, epoc
```

Orbits in 2D

In [ ]:
```
adjust_display()

asteroids_phaY = df_asteroids[df_asteroids["pha"] == 'Y'].index[0:10].tolist
asteroids_phaN = df_asteroids[df_asteroids["pha"] == 'N'].index[0:10].tolist
list_asteroid = [[index, "PHA=Yes"] for index in asteroids_phaY] + [[index,

EPOCH = Time("2000-01-01 12:00:00", scale="tdb")
frame = plot_solar_system(outer=False, epoch=EPOCH, interactive=True)
```

```
#frame._ax.figure.set_size_inches(25, 12)
for i in list_asteroid:
  _, _, epoch, ast_name, orb = asteroid_orbit_from_orbital_elements(df_aster

  if i[1] == "PHA=Yes":
    frame.plot(orb, label=ast_name + " " + i[1], color="red", )
  else:
    frame.plot(orb, label=ast_name.replace("  ", "") + " " + i[1])

frame._figure.update_layout(autosize=False, legend=dict(orientation="h", font
frame.show()
```

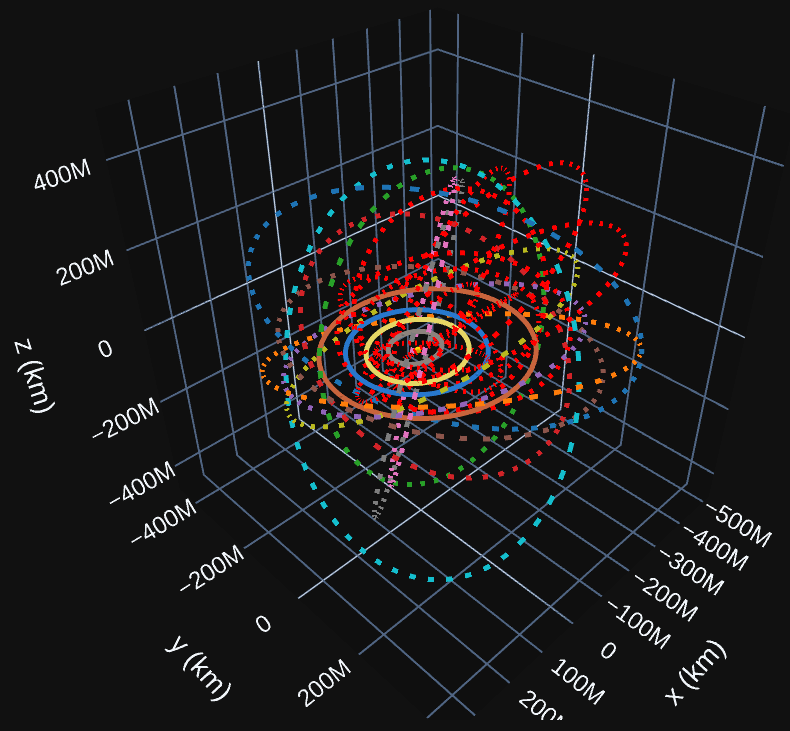


2000-01-01 12:00 (Mercury (☿))
2000-01-01 12:00 (Venus (♀))
2000-01-01 12:00 (Earth (♁))
2000-01-01 12:00 (Mars (♂))
['2020-05-31 00:00'] (1566 Icarus (1949 MA) PHA=Yes)
['2020-05-31 00:00'] (1620 Geographos (1951 RA) PHA=Yes)
['2020-05-31 00:00'] (1862 Apollo (1932 HA) PHA=Yes)
['2020-05-31 00:00'] (1981 Midas (1973 EA) PHA=Yes)
['2020-05-31 00:00'] (2101 Adonis (1936 CA) PHA=Yes)
['2020-05-31 00:00'] (2102 Tantalus (1975 YA) PHA=Yes)
['2020-05-31 00:00'] (2135 Aristaeus (1977 HA) PHA=Yes)
['2020-05-31 00:00'] (2201 Oljato (1947 XC) PHA=Yes)
['2004-08-11 00:00'] (2340 Hathor (1976 UA) PHA=Yes)
['2020-05-31 00:00'] (3122 Florence (1981 ET3) PHA=Yes)
['2020-05-31 00:00'] (155 Scylla PHA=No)
['2020-05-31 00:00'] (157 Dejanira PHA=No)
['2020-05-31 00:00'] (220 Stephania PHA=No)
['2020-05-31 00:00'] (228 Agathe PHA=No)
['2020-05-31 00:00'] (244 Sita PHA=No)
['2020-05-31 00:00'] (249 Ilse PHA=No)
['2020-05-31 00:00'] (254 Augusta PHA=No)
['2020-05-31 00:00'] (262 Valda PHA=No)
['2020-05-31 00:00'] (265 Anna PHA=No)
['2020-05-31 00:00'] (280 Philia PHA=No)

Orbits in 3D

In [ ]:

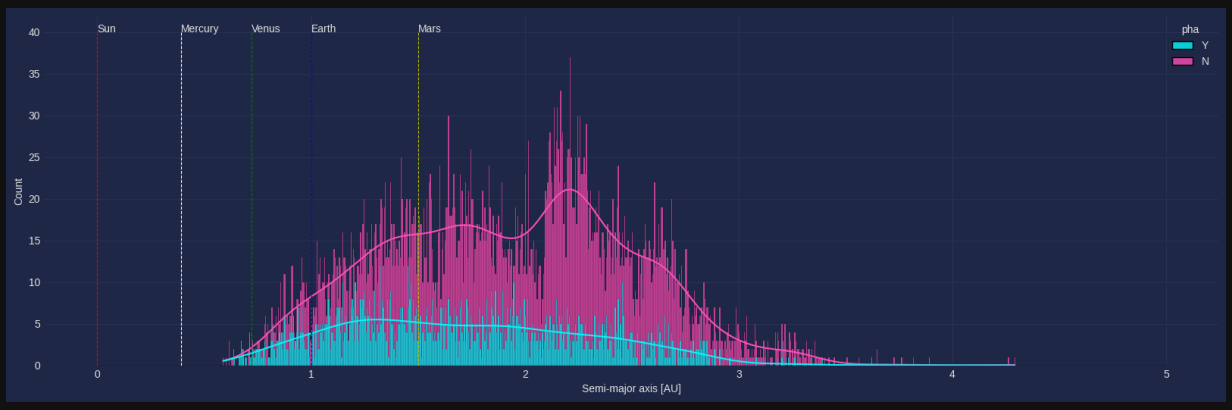
```
adjust_display()
frame = plot_solar_system(outer=False, epoch=EPOCH, interactive=True, use_3d
#frame._ax.figure.set_size_inches(25, 12)
for i in list_asteroid:
  _, _, epoch, ast_name, orb = asteroid_orbit_from_orbital_elements(df_aster

  if i[1] == "PHA=Yes":
    frame.plot(orb, label=ast_name + " " + i[1], color="red")
  else:
    frame.plot(orb, label=ast_name.replace("  ", "") + " " + i[1])

frame._figure.update_layout(autosize=False, legend=dict(orientation="h", font
frame.show()
```



2000-01-01 12:00 (Mercury (☿))
2000-01-01 12:00 (Venus (♀))
2000-01-01 12:00 (Earth (♁))
2000-01-01 12:00 (Mars (♂))
['2020-05-31 00:00'] (1566 Icarus (1949 MA) PHA=Yes)
['2020-05-31 00:00'] (1620 Geographos (1951 RA) PHA=Yes)
['2020-05-31 00:00'] (1862 Apollo (1932 HA) PHA=Yes)
['2020-05-31 00:00'] (1981 Midas (1973 EA) PHA=Yes)
['2020-05-31 00:00'] (2101 Adonis (1936 CA) PHA=Yes)
['2020-05-31 00:00'] (2102 Tantalus (1975 YA) PHA=Yes)
['2020-05-31 00:00'] (2135 Aristaeus (1977 HA) PHA=Yes)
['2020-05-31 00:00'] (2201 Oljato (1947 XC) PHA=Yes)
['2004-08-11 00:00'] (2340 Hathor (1976 UA) PHA=Yes)
['2020-05-31 00:00'] (3122 Florence (1981 ET3) PHA=Yes)
['2020-05-31 00:00'] (155 Scylla PHA=No)
['2020-05-31 00:00'] (157 Dejanira PHA=No)
['2020-05-31 00:00'] (220 Stephania PHA=No)
['2020-05-31 00:00'] (228 Agathe PHA=No)
['2020-05-31 00:00'] (244 Sita PHA=No)
['2020-05-31 00:00'] (249 Ilse PHA=No)
['2020-05-31 00:00'] (254 Augusta PHA=No)
['2020-05-31 00:00'] (262 Valda PHA=No)
['2020-05-31 00:00'] (265 Anna PHA=No)
['2020-05-31 00:00'] (280 Philia PHA=No)

Near-Earth Objects

In [ ]:
```
neos = df_asteroids[df_asteroids["neo"] == 'Y']

fig = plt.figure(figsize=(20, 6))
bin_w = 0.005
neos_min = 0
neos_max = 5
bins = np.arange(neos_min, neos_max, bin_w)
sns.histplot(data=neos, x="a", hue="pha", hue_order=["Y", "N"], kde=True, bi
plt.xlabel("Semi-major axis [AU]")

plt.plot( [0, 0], [40, 0], 'r--', lw=0.8)
plt.text( 0, 40, "Sun")

plt.plot( [0.39, 0.39], [40, 0], 'w--', lw=0.8)
plt.text( 0.39, 40, "Mercury")

plt.plot( [0.72, 0.72], [40, 0], 'g--', lw=0.8)
plt.text( 0.72, 40, "Venus")

plt.plot( [1, 1], [40, 0], 'b--', lw=0.8)
plt.text( 1, 40, "Earth")

plt.plot( [1.5, 1.5], [40, 0], 'y--', lw=0.8)
plt.text( 1.5, 40, "Mars");
```



In [ ]:
```
df_asteroids.groupby(["pha"]).a.mean()
```

Out[ ]:

pha	a
N	2.673730
Y	1.725794

dtype: float64

The NEOs that are potentially dangerous have orbits with lower semi-major axis values.

In [ ]:
```
df_asteroids.loc[df_asteroids["pha"] == "Y", "neo"].value_counts()
```

Out[ ]:

neo	count
Y	1737

dtype: int64

In [ ]:
```
df_asteroids.loc[df_asteroids["neo"] == "Y", "pha"].value_counts()
```

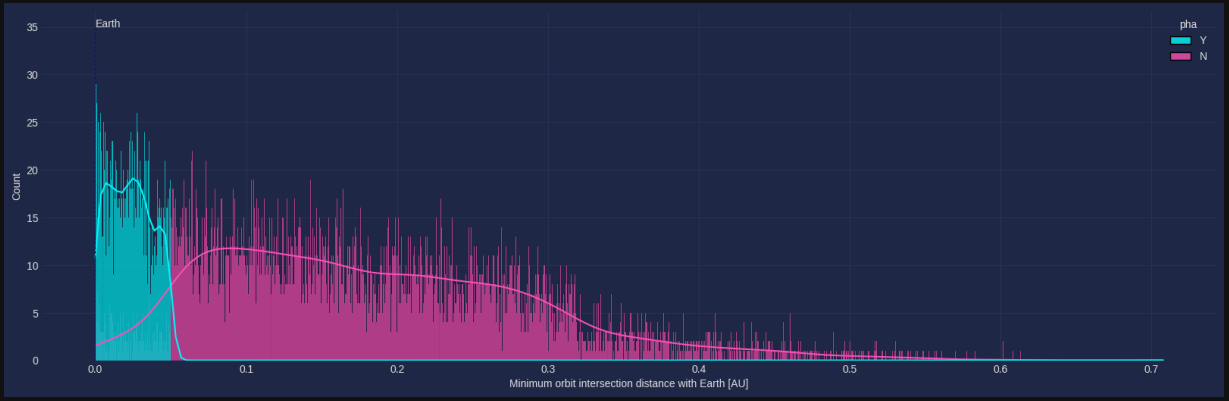
Out[ ]:

pha	count
N	6046
Y	1737

dtype: int64

As we can see, not all near-Earth objects are potentially dangerous, but it can be confirmed that all potentially dangerous asteroids are near-Earth objects.

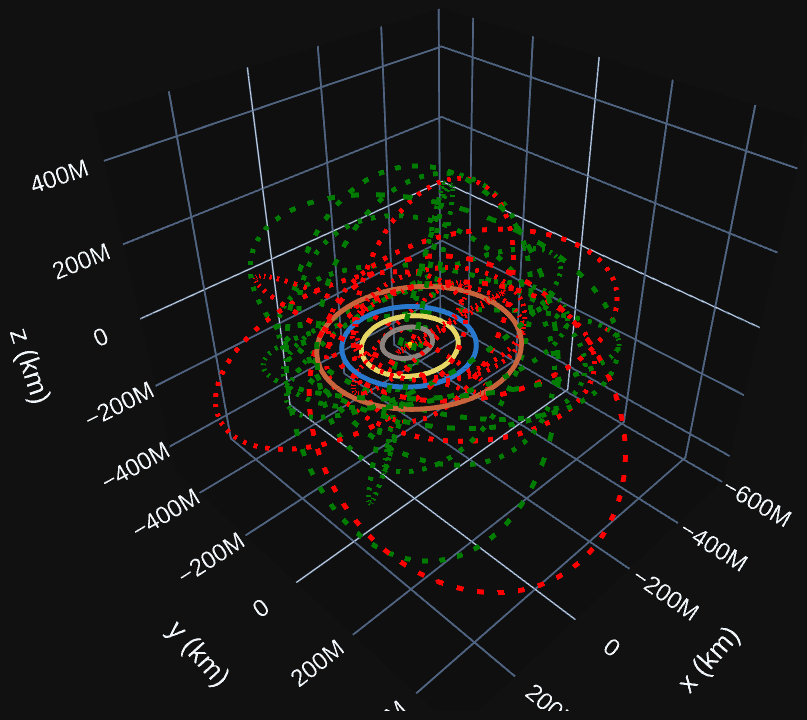
In [ ]:
```
moid = df_asteroids[df_asteroids["neo"] == 'Y']

fig = plt.figure(figsize=(20, 6))
bin_w = 0.0005
moid_min = moid.moid.min()
moid_max = moid.moid.max()
bins = np.arange(moid_min, moid_max, bin_w)
sns.histplot(data=moid, x="moid", hue="pha", kde=True, hue_order=["Y", "N"],
plt.xlabel("Minimum orbit intersection distance with Earth [AU]")
plt.plot( [0, 0], [35, 0], 'b--', lw=0.8)
plt.text( 0, 35, "Earth");
```



- Once again, not all `neos` are potentially dangerous, but those objects that have smaller intersection distances with the Earth's orbit are.

- We can conclude that potentially dangerous asteroids have orbits with a very high probability of intersecting with the Earth's orbit.

In [ ]:
```
adjust_display()

asteroids_neo_Y = df_asteroids[df_asteroids["neo"] == 'Y'].index[0:10].tolis
asteroids_neo_N = df_asteroids[df_asteroids["neo"] == 'N'].index[0:10].tolis
list_asteroid = [[index, "NEO=Yes"] for index in asteroids_neo_Y] + [[index,

frame = plot_solar_system(outer=False, epoch=EPOCH, interactive=True, use_3d
#frame._ax.figure.set_size_inches(25, 12)
for i in list_asteroid:
  _, _, epoch, ast_name, orb = asteroid_orbit_from_orbital_elements(df_aster

  if i[1] == "NEO=Yes":
    frame.plot(orb, label=ast_name + " " + i[1], color="red")
  else:
    frame.plot(orb, label=ast_name.replace("  ", "") + " " + i[1], color="gr

frame._figure.update_layout(autosize=False, legend=dict(orientation="h", font
frame.show()
```



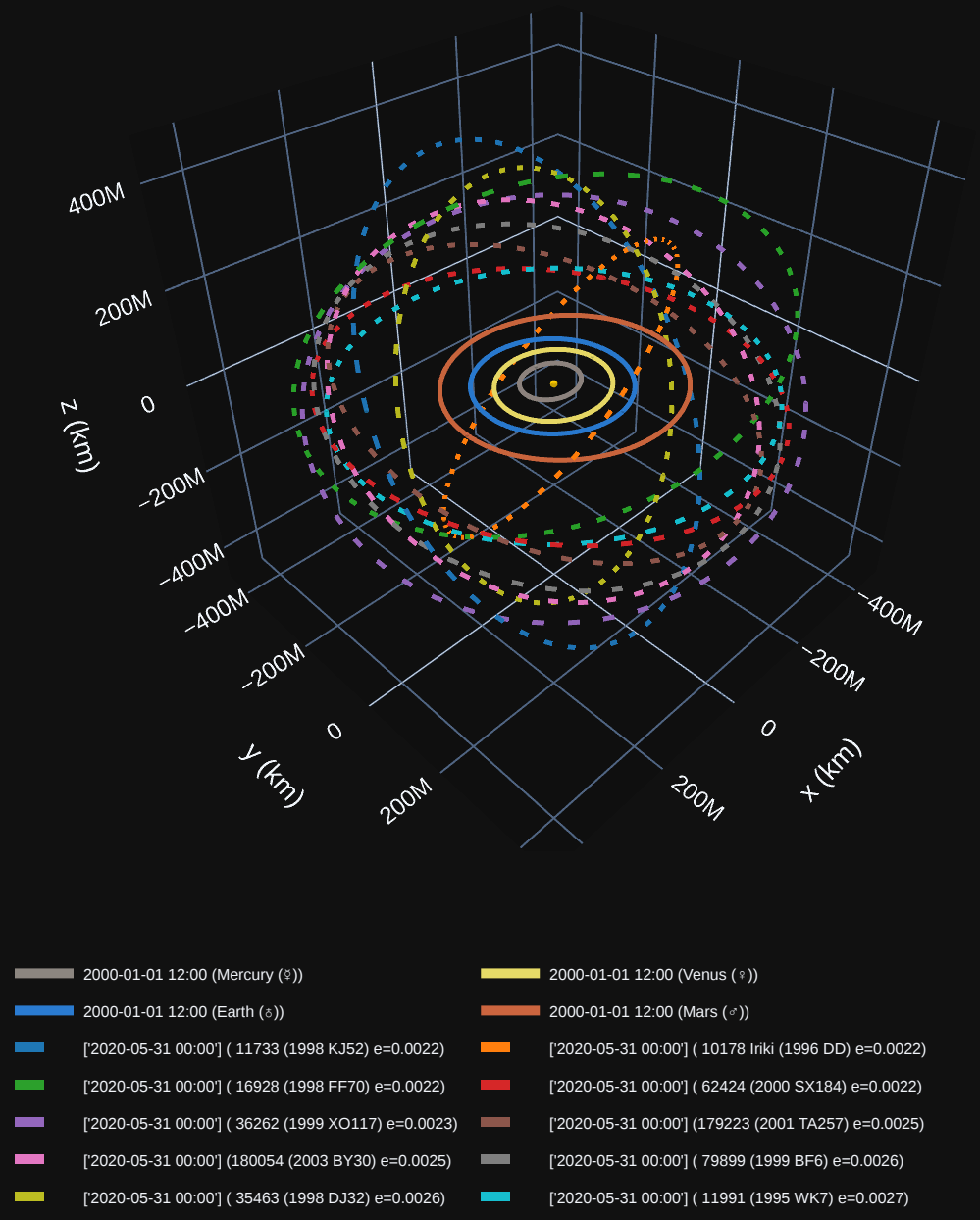
2000-01-01 12:00 (Mercury (☿))
2000-01-01 12:00 (Venus (♀))
2000-01-01 12:00 (Earth (♁))
2000-01-01 12:00 (Mars (♂))
['2020-05-31 00:00'] (719 Albert (1911 MT) NEO=Yes)
['2020-05-31 00:00'] (887 Alinda (1918 DB) NEO=Yes)
['2020-05-31 00:00'] (1221 Amor (1932 EA1) NEO=Yes)
['2020-05-31 00:00'] (1566 Icarus (1949 MA) NEO=Yes)
['2020-05-31 00:00'] (1580 Betulia (1950 KA) NEO=Yes)
['2020-05-31 00:00'] (1620 Geographos (1951 RA) NEO=Yes)
['2020-05-31 00:00'] (1627 Ivar (1929 SH) NEO=Yes)
['2020-05-31 00:00'] (1685 Toro (1948 OA) NEO=Yes)
['2020-05-31 00:00'] (1862 Apollo (1932 HA) NEO=Yes)
['2020-05-31 00:00'] (1863 Antinous (1948 EA) NEO=Yes)
['2020-05-31 00:00'] (155 Scylla NEO=No)
['2020-05-31 00:00'] (157 Dejanira NEO=No)
['2020-05-31 00:00'] (220 Stephania NEO=No)
['2020-05-31 00:00'] (228 Agathe NEO=No)
['2020-05-31 00:00'] (244 Sita NEO=No)
['2020-05-31 00:00'] (249 Ilse NEO=No)
['2020-05-31 00:00'] (254 Augusta NEO=No)
['2020-05-31 00:00'] (262 Valda NEO=No)
['2020-05-31 00:00'] (265 Anna NEO=No)
['2020-05-31 00:00'] (280 Philia NEO=No)

Circular orbits

In [ ]:

```
adjust_display()

circulares = df_asteroids[df_asteroids["e"] < 0.1].sort_values(by='e', ascen
list_asteroid = [[index, f"e={round(df_asteroids.loc[index, 'e'], 4)}"] for

frame = plot_solar_system(outer=False, epoch=EPOCH, interactive=True, use_3d
for i in list_asteroid:
  _, _, epoch, ast_name, orb = asteroid_orbit_from_orbital_elements(df_aster
  frame.plot(orb, label=ast_name + " " + i[1])
  output.clear()
```

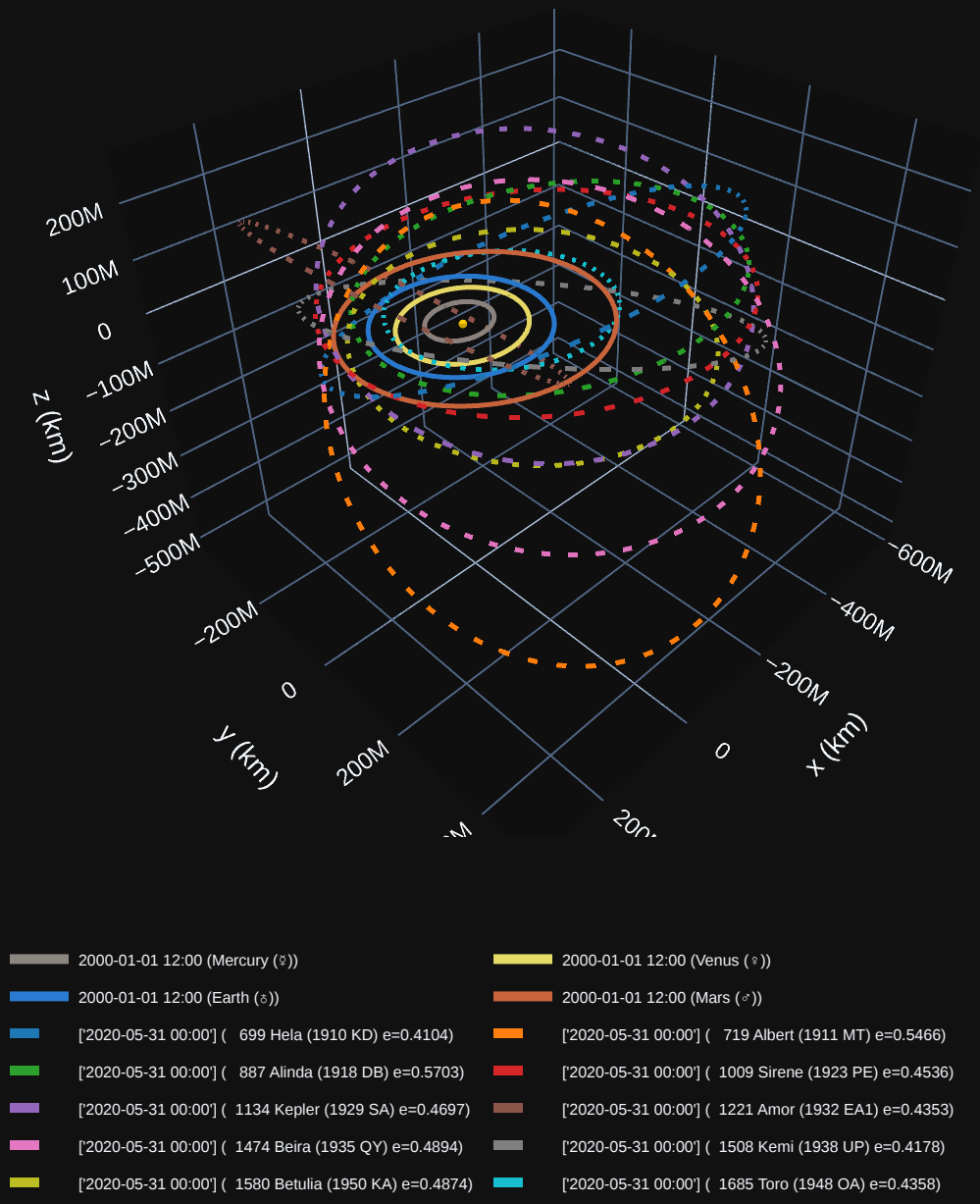
```
frame._figure.update_layout(autosize=False, legend=dict(orientation="h", font
frame.show()
```



Elliptical orbits

In [ ]:
```
adjust_display()

circulares = df_asteroids[(df_asteroids["e"] > 0.4) & (df_asteroids["e"] < 0
list_asteroid = [[index, f"e={round(df_asteroids.loc[index, 'e'], 4)}"] for

frame = plot_solar_system(outer=False, epoch=EPOCH, interactive=True, use_3d
for i in list_asteroid:
  _, _, epoch, ast_name, orb = asteroid_orbit_from_orbital_elements(df_aster
  frame.plot(orb, label=ast_name + " " + i[1])
```

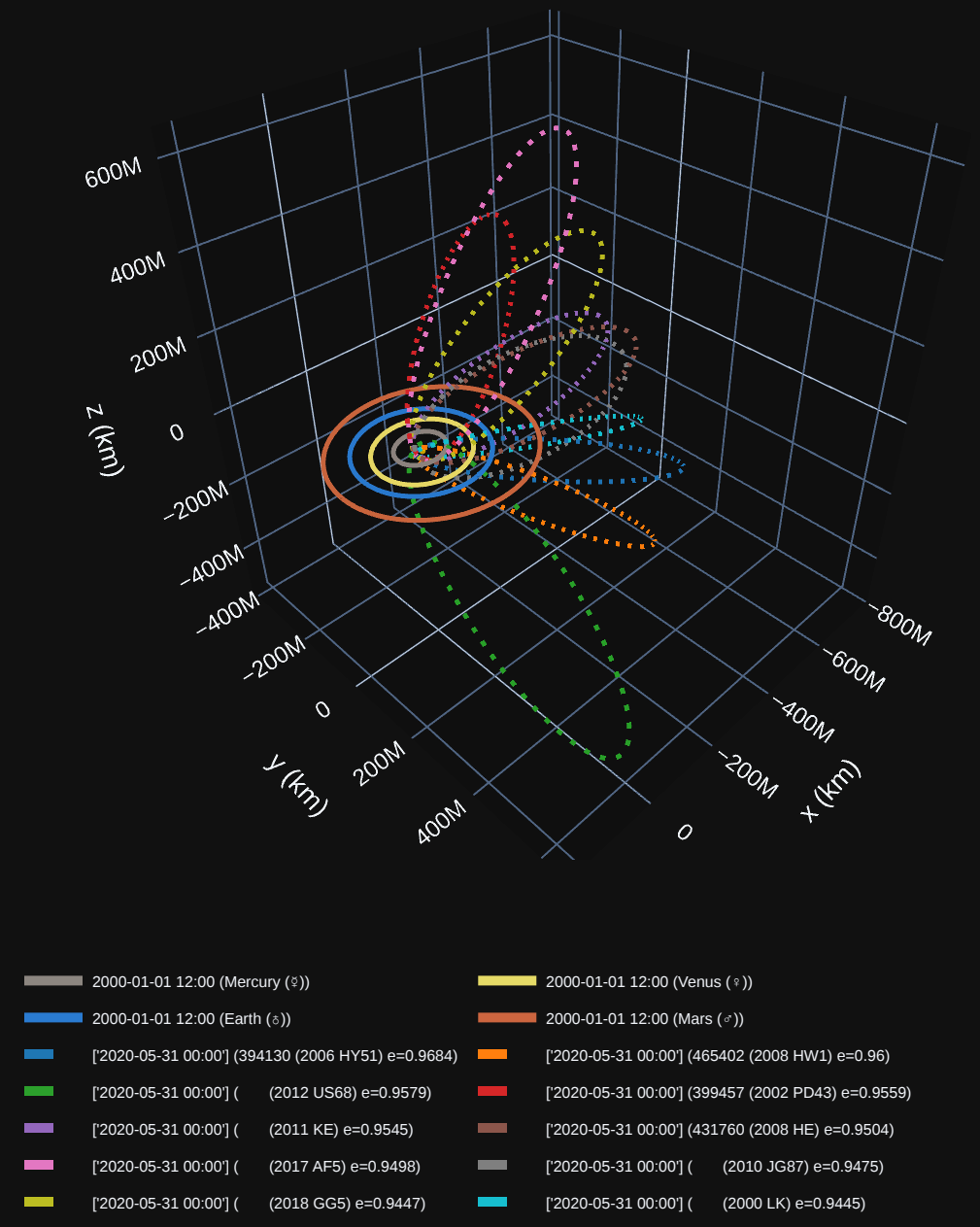
```
  output.clear()

frame._figure.update_layout(autosize=False, legend=dict(orientation="h", font
frame.show()
```



Parabolic and hyperbolic orbits

In [ ]:
```
adjust_display()

hyperbolics = df_asteroids[df_asteroids["e"] > 0.9].sort_values(by='e', asce
list_asteroid = [[index, f"e={round(df_asteroids.loc[index, 'e'], 4)}"] for

frame = plot_solar_system(outer=False, epoch=EPOCH, interactive=True, use_3d
for i in list_asteroid:
```

```
  _, _, epoch, ast_name, orb = asteroid_orbit_from_orbital_elements(df_aster
  frame.plot(orb, label=ast_name + " " + i[1])
  output.clear()

frame._figure.update_layout(autosize=False, legend=dict(orientation="h", font
frame.show()
```



Orbits by classification

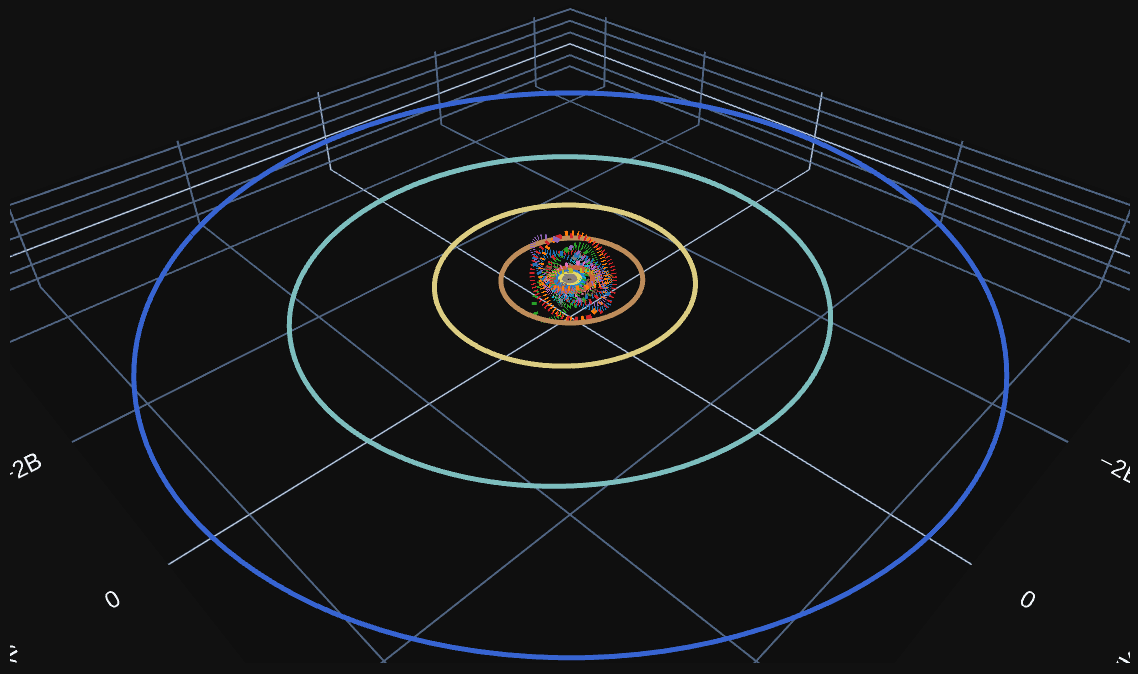
Abbreviation	Title	Description
AMO	Amor	Near-Earth asteroid, orbit similar to 1221 Amor (a > 1.0 AU; 1.017 AU < q < 1.3 AU).

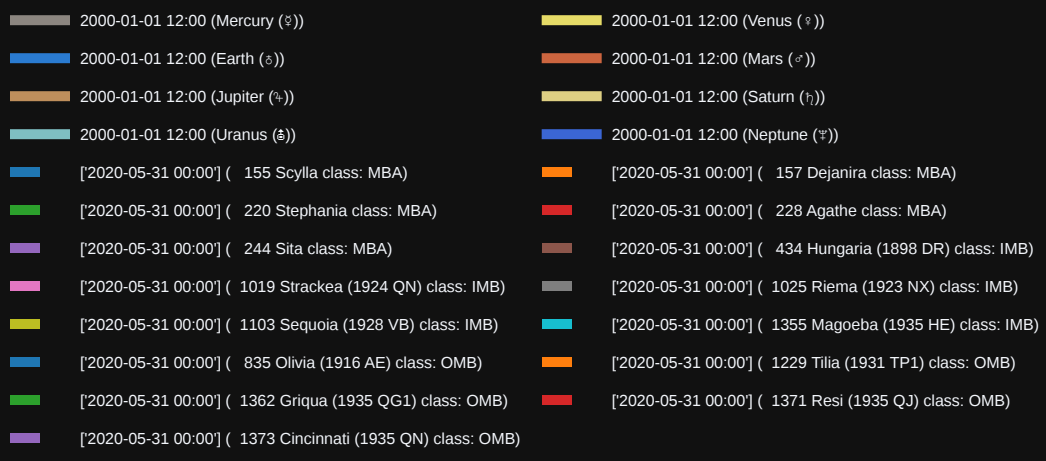
Abbreviation	Title	Description
APO	Apollo	Near-Earth asteroids with orbits that cross Earth's orbit, similar to 1862 Apollo (a > 1.0 AU; q < 1.017 AU).
AST	Asteroid	The asteroid's orbit does not match any defined class.
ATE	Aten	Near-Earth asteroid, orbit similar to 2062 Aten (a < 1.0 AU; Q > 0.983 AU).
CEN	Centaur	Objects with orbits between Jupiter and Neptune (5.5 AU < a < 30.1 AU).
HYA	Hyperbolic Asteroid	Asteroids in hyperbolic orbits (e > 1.0).
IEO	Interior Earth Object	An asteroid's orbit entirely contained within Earth's orbit (Q < 0.983 AU).
IMB	Inner Main-belt Asteroid	Asteroids with orbital elements restricted by (a < 2.0 AU; q > 1.666 AU).
MBA	Main-belt Asteroid	Asteroids with orbital elements restricted by (2.0 AU < a < 3.2 AU; q > 1.666 AU).
MCA	Mars-crossing Asteroid	Asteroids that cross Mars's orbit, restricted by (1.3 AU < q < 1.666 AU; a < 3.2 AU).
OMB	Outer Main-belt Asteroid	Asteroids with orbital elements restricted by (3.2 AU < a < 4.6 AU).
PAA	Parabolic Asteroid	Asteroids in parabolic orbits (e = 1.0).
TJN	Jupiter Trojan	Asteroids trapped in Jupiter's Lagrange points L4/L5 (4.6 AU < a < 5.5 AU; e < 0.3).
TNO	TransNeptunian Object	Objects with orbits beyond Neptune (a > 30.1 AU).

The main asteroid belt

Let's take a look at some of the asteroids that are in the main asteroid belt, which have orbits between the orbits of Mars and Jupiter.

In [ ]:
```
adjust_display()

mba = df_asteroids[df_asteroids["class"] == "MBA"].index[0:5].tolist()
imb = df_asteroids[df_asteroids["class"] == "IMB"].index[0:5].tolist()
omb = df_asteroids[df_asteroids["class"] == "OMB"].index[0:5].tolist()
list_asteroid = [[index, f"class: MBA"] for index in mba] + [[index, f"class

frame = plot_solar_system(outer=True, epoch=EPOCH, interactive=True, use_3d=
for i in list_asteroid:
  _, _, epoch, ast_name, orb = asteroid_orbit_from_orbital_elements(df_aster
  frame.plot(orb, label=ast_name + " " + i[1])
  output.clear()

frame._figure.update_layout(autosize=False, legend=dict(orientation="h", font
frame.show()
```





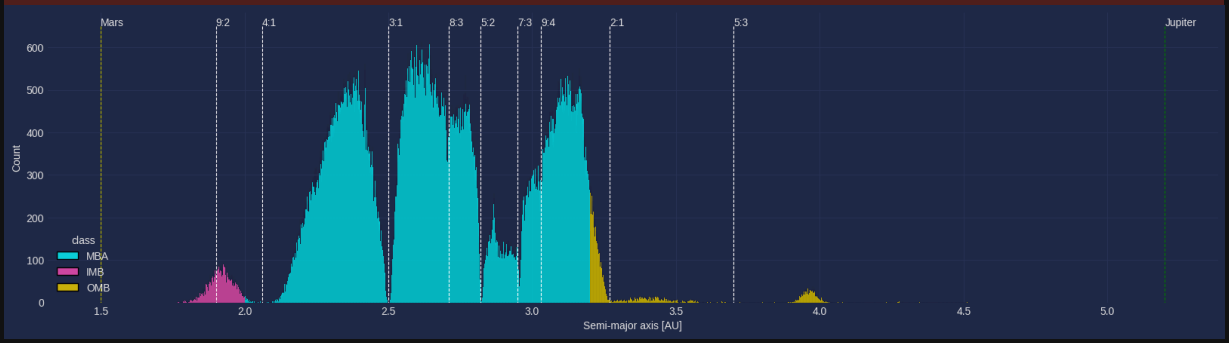
Kirkwood Gaps

The Kirkwood gaps are regions in the asteroid belt where the density of asteroids is notably reduced compared to the average density of the belt. These gaps coincide with orbits whose parameters (their semi-major axis or equivalently their orbital period) have a simple ratio with Jupiter's orbit. It is said that the gaps correspond to the orbital resonances with Jupiter.

In [ ]:
```
gaps = {"9:2":1.9,
        "4:1":2.06,
        "3:1":2.5,
        "5:2":2.82,
        "7:3":2.95,
```

```
        "2:1":3.27,
        "8:3":2.71,
        "9:4":3.03,
        "5:3":3.7,
        }

mainbelt = df_asteroids[df_asteroids["class"].isin(["MBA", "IMB", "OMB", "TJ

fig = plt.figure(figsize=(20, 5))
bin_w = 0.0005
a_min = mainbelt["a"].min()
a_max = mainbelt["a"].max()
bins = np.arange(a_min, a_max, bin_w)
sns.histplot(data=mainbelt, x="a", hue="class", bins=bins, alpha=0.8)
plt.xlabel("Semi-major axis [AU]")

for gap in gaps:
    plt.plot( [gaps[gap], gaps[gap]], [650, 0], 'w--', lw=0.8)
    plt.text( gaps[gap], 650, gap)

plt.plot( [5.2, 5.2], [650, 0], 'g--', lw=0.8)
plt.text( 5.2, 650, "Jupiter")

plt.plot( [1.5, 1.5], [650, 0], 'y--', lw=0.8)
plt.text( 1.5, 650, "Mars");
```

```
/usr/local/lib/python3.11/dist-packages/IPython/core/events.py:89: UserWarnin
g:

Creating legend with loc="best" can be slow with large amounts of data.

/usr/local/lib/python3.11/dist-packages/IPython/core/pylabtools.py:151: UserW
arning:

Creating legend with loc="best" can be slow with large amounts of data.
```



These resonances are identified as `J:A` where J is the number of revolutions of Jupiter and A is the number of revolutions of the asteroid. That is, while Jupiter makes `J` revolutions, the asteroid will make `A` revolutions. This could cause the asteroid to coincide with a maximum approach to Jupiter every `A` orbits, which would lead to a resonant attraction that could eventually expel it from that orbit.

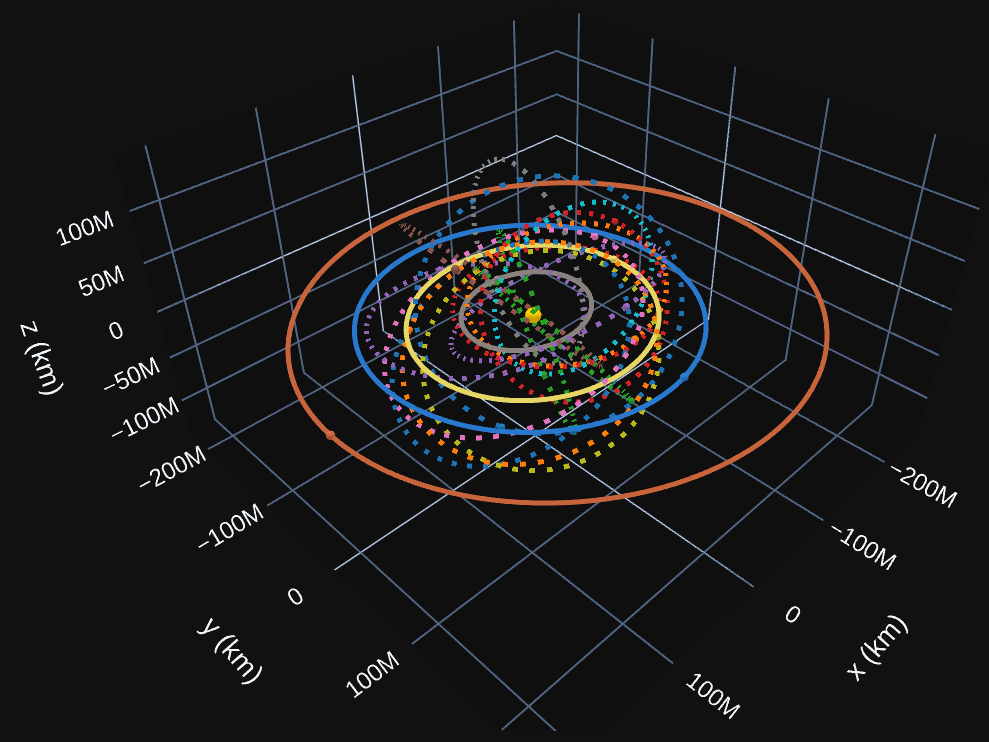
In Jupiter's orbit, we can find the 1:1 resonance. Asteroids that are in this resonance are known as Trojan asteroids.

As we can observe, the orbits of Mars and Jupiter evidently form the boundaries of this asteroid belt.

Orbits completely contained within Earth's orbit

Let’s take a look at some of the asteroids that have orbits completely contained within Earth's orbit.

In [ ]:
```
adjust_display()

mba = df_asteroids[df_asteroids["class"] == "IEO"].index[0:15].tolist()
list_asteroid = [[index, f"class: IEO"] for index in mba]

frame = plot_solar_system(outer=False, epoch=EPOCH, interactive=True, use_3d
for i in list_asteroid:
  _, _, epoch, ast_name, orb = asteroid_orbit_from_orbital_elements(df_aster
  frame.plot(orb, label=ast_name + " " + i[1])

frame._figure.update_layout(autosize=False, legend=dict(orientation="h", font
frame.show()
```



2000-01-01 12:00 (Mercury (☿))
2000-01-01 12:00 (Venus (♀))
2000-01-01 12:00 (Earth (♁))
2000-01-01 12:00 (Mars (♂))
['2020-05-31 00:00'] (163693 Atira (2003 CP20) class: IEO)
['2020-05-31 00:00'] (164294 (2004 XZ130) class: IEO)
['2020-05-31 00:00'] (413563 (2005 TG45) class: IEO)
['2020-05-31 00:00'] (418265 (2008 EA32) class: IEO)
['2020-05-31 00:00'] (434326 (2004 JG6) class: IEO)
['2020-05-31 00:00'] (481817 (2008 UL90) class: IEO)
['2020-05-31 00:00'] ((2006 WE4) class: IEO)
['2020-05-31 00:00'] ((2010 XB11) class: IEO)
['2020-05-31 00:00'] ((2012 VE46) class: IEO)
['2020-05-31 00:00'] ((2013 JX28) class: IEO)
['2020-05-31 00:00'] ((2013 TQ5) class: IEO)
['2020-05-31 00:00'] ((2014 FO47) class: IEO)
['2020-05-31 00:00'] ((2015 DR215) class: IEO)
['2020-05-31 00:00'] ((2017 YH) class: IEO)
['2020-05-31 00:00'] ((2018 JB3) class: IEO)

Feature Engineering

Diameters according to the absolute magnitude

In [ ]:
```
#Source: https://authors.library.caltech.edu/70302/1/aj_152_4_79.pdf
#Other source: https://josevicentediaz.com/tag/asteroide-amor/
#We assume an albedo of 0.0615, a value that we had to discard earlier.
albedo = 0.0615
df_asteroids["diametro"] = (1329/albedo)*np.power(10, -0.4*df_asteroids["H"]
df_asteroids[["diametro"]]
```

Out[ ]:

id	diametro
a0000155	0.715566
a0000157	0.784603
a0000220	0.715566
a0000228	0.284872
a0000244	0.375535
...	...
bK20H07J	0.000460
bK20H08G	0.000221
bK20H10Y	0.000138
bPLS6331	0.000860
bPLS6344	0.000150

761000 rows × 1 columns

Feature elimination

There are some features that are purely for identification purposes. These features will not be useful for our model, as they do not provide relevant information and, in most cases, contain highly varied categorical information.

In [ ]:
```
adjust_display()
columns = df_asteroids.columns.tolist()
df_unique = pd.DataFrame(data={"Feature":columns, "Number of values":np.zero
df_unique["Type"] = ""

for col in columns:
  unique = len(df_asteroids[col].unique())
  df_unique.loc[df_unique["Feature"] == col, "Number of values"] = unique
  df_unique.loc[df_unique["Feature"] == col, "Tipo"]= "Categoric" if df_aste
df_unique
```

Out[ ]:

	Feature	Number of values	Type	Tipo
0	spkid	761000		Numeric
1	full_name	761000		Categoric
2	pdes	761000		Categoric
3	neo	2		Categoric
4	pha	2		Categoric
5	H	4010		Numeric
6	orbit_id	520		Categoric
7	epoch	59		Numeric
8	epoch_mjd	59		Numeric
9	epoch_cal	59		Numeric
10	equinox	1		Categoric
11	e	761000		Numeric
12	a	761000		Numeric
13	q	761000		Numeric
14	i	761000		Numeric
15	om	761000		Numeric
16	w	761000		Numeric
17	ma	761000		Numeric
18	ad	761000		Numeric
19	n	761000		Numeric
20	tp	761000		Numeric
21	tp_cal	760984		Numeric
22	per	761000		Numeric
23	per_y	761000		Numeric
24	moid	256848		Numeric
25	moid_ld	256848		Numeric
26	sigma_e	132893		Numeric
27	sigma_a	146193		Numeric
28	sigma_q	125463		Numeric
29	sigma_i	102263		Numeric
30	sigma_om	131529		Numeric
31	sigma_w	140806		Numeric

	Feature	Number of values	Type	Tipo
32	sigma_ma	142121		Numeric
33	sigma_ad	142096		Numeric
34	sigma_n	125858		Numeric
35	sigma_tp	164680		Numeric
36	sigma_per	155444		Numeric
37	class	8		Categoric
38	rms	44140		Numeric
39	diametro	4010		Numeric

Focus on categorical ceatures (Including `spkid`)

- The features `spkid`, `full_name`, `orbit_id`, and `pdes` have millions of distinct categorical values that provide information not inherent to the potential danger of an asteroid.
- The features `pha`, `neo`, and `class` have fewer categorical values, and later we will see how to handle these features.
- The feature `equinox` has the same value for all records, which is "J2000". We discussed this value earlier, and since it is the same for all records, it will be very irrelevant for our model, as presenting the same value across all records is equivalent to providing no additional information.

In [ ]:
```
df_asteroids.drop(["orbit_id", "full_name", "pdes", "equinox"], axis=1, inpl
df_asteroids.head()
```

Out[ ]:

id	spkid	neo	pha	H	epoch	epoch_mjd	epoch_cal	e	
a0000155	2000155.0	N	N	11.2	2459000.5	59000.0	20200531.0	0.275781	2.759
a0000157	2000157.0	N	N	11.1	2459000.5	59000.0	20200531.0	0.197300	2.579
a0000220	2000220.0	N	N	11.2	2459000.5	59000.0	20200531.0	0.257317	2.348
a0000228	2000228.0	N	N	12.2	2459000.5	59000.0	20200531.0	0.241806	2.201
a0000244	2000244.0	N	N	11.9	2459000.5	59000.0	20200531.0	0.137159	2.174

Redundant features

We can then observe that there are many equivalent features that are possibly scaled by some constant. It would be best to retain one of these features and remove the others, as keeping them would be redundant.

Some of these are:

- `epoch` ⇔ `epoch_mjd` and `epoch_cal`
- `tp` ⇔ `tp_cal`
- `per` ⇔ `per_y`
- `moid` ⇔ `moid_ld`

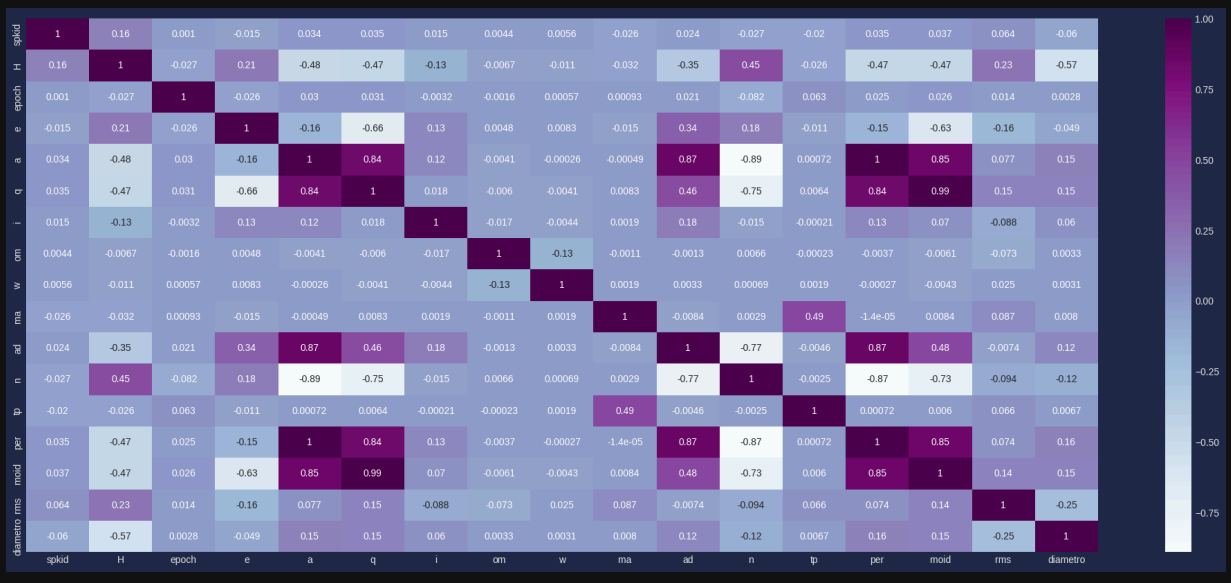
In [ ]:
```
df_asteroids.drop(["epoch_mjd", "epoch_cal", "tp_cal", "per_y", "moid_ld"],
```

The uncertainty features are no longer useful to us, as they are mostly related to the certainty of the measurements rather than the variable we aim to predict.

In [ ]:
```
df_asteroids.drop(columns_sigma, axis=1, inplace=True)
```

Let's look at correlations between features

In [ ]:
```
adjust_display()
numerical_cols = df_asteroids.select_dtypes(["uint8", "float64"]).columns
categorical_cols = df_asteroids.select_dtypes(["object"]).columns

fig = plt.figure(figsize=(25, 10))
sns.heatmap(df_asteroids[numerical_cols].corr(), annot=True, cmap='BuPu', an
```



Detecting and handling multicollinearity between features

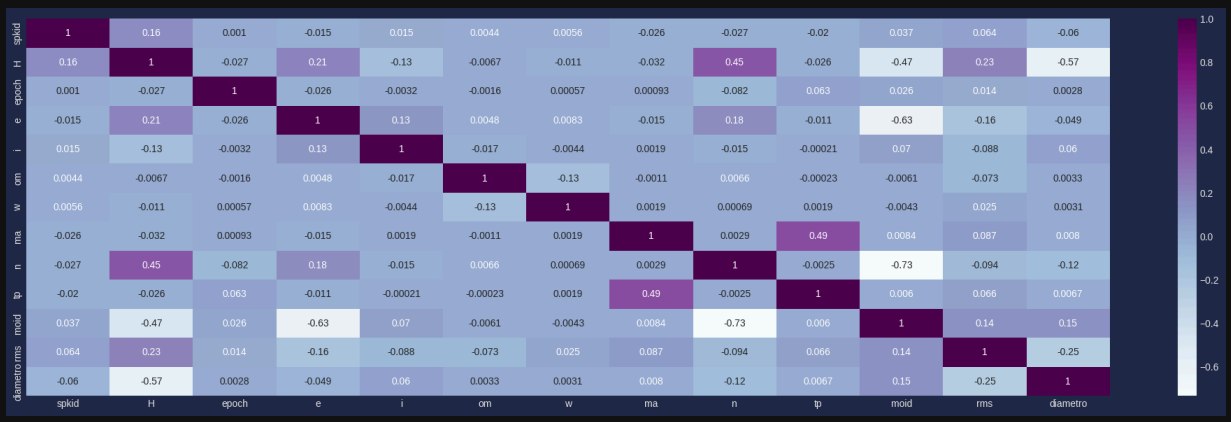
Multicollinearity occurs when the independent variables (predictors) in a regression model are correlated. Independent variables should be, well, independent.

In [ ]:
```
def detect_VIF(df, corr_list):
    df_ = df.copy()
    df_['intercept'] = 1

    with np.errstate(divide='ignore'):
      while(True):
          df_vif = pd.DataFrame(columns=["Features", "VIF"])
          df_vif["Features"] = df_.columns
          df_vif["VIF"] = [variance_inflation_factor(df_.values.astype(np.floa
          df_vif = df_vif[df_vif["Features"] != "intercept"].sort_values("VI

          if df_vif.iloc[0]["VIF"] > 10:
              vif_list = df_vif.loc[df_vif["VIF"] == df_vif["VIF"].max(), "F
              for corr in corr_list:
                if corr in vif_list:
                  df_.drop([corr], axis=1, inplace=True)
                  break
              corr_list.remove(corr)
          else:
              break

      df_.drop(["intercept"], axis=1, inplace=True)
    return df[df_.columns.tolist()], df_vif

corr_list = df_asteroids[numerical_cols].corr().index.tolist()
df_asteroids_select, df_vif = detect_VIF(df_asteroids[numerical_cols], corr_
numeric_cols = df_asteroids_select.columns
df_vif
```

Out[ ]:

	Features	VIF
10	moid	5.448662
8	n	3.333921
3	e	2.574054
1	H	2.258365
12	diametro	1.572130
7	ma	1.331994
9	tp	1.330022
11	rms	1.202403
4	i	1.130980
0	spkid	1.054008
5	om	1.024466
2	epoch	1.020487
6	w	1.019469

In [ ]:
```
adjust_display()

fig = plt.figure(figsize=(25, 7))
sns.heatmap(df_asteroids_select.corr(), annot=True, cmap='BuPu', annot_kws={
```



Separating features from the label

In [ ]:
```
df_asteroids_ = df_asteroids[list(numerical_cols) + list(categorical_cols)]
X = df_asteroids_.drop(["pha"], axis=1)
y = df_asteroids_["pha"]
y.value_counts()
```

Out[ ]:

pha	count
N	759263
Y	1737

dtype: int64

Splitting into training, validation, and test datasets

In [ ]:
```
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.30, stra
X_val, X_test, y_val, y_test = train_test_split(X_val, y_val, test_size=0.25
X_train.shape, X_val.shape, X_test.shape
```

Out[ ]: ((532700, 19), (171225, 19), (57075, 19))

In [ ]:
```
y_train.value_counts()
```

Out[ ]:

pha	count
N	531484
Y	1216

dtype: int64

Encoding categorical features

In [ ]:
```
numerical_cols = X_train.select_dtypes(["uint8", "float64"]).columns
categorical_cols = X_train.select_dtypes(["object"]).columns
```

Encoding label

In [ ]:
```
le = LabelEncoder()
y_train = le.fit_transform(y_train).astype("uint8")
y_val = le.transform(y_val).astype("uint8")
y_test = le.transform(y_test).astype("uint8")
```

Encoding features

In [ ]:
```
from sklearn.preprocessing import OneHotEncoder

ohe = OneHotEncoder(sparse_output=False, handle_unknown="ignore").set_output
ohe.fit_transform(X_train[categorical_cols]).head()
```

Out[ ]:

id	neo_N	neo_Y	class_AMO	class_APO	class_ATE	class_IEO	class_IMB	clas
a0509205	1.0	0.0	0.0	0.0	0.0	0.0	0.0	
a0400500	1.0	0.0	0.0	0.0	0.0	0.0	0.0	
a0476160	1.0	0.0	0.0	0.0	0.0	0.0	0.0	
a0059146	1.0	0.0	0.0	0.0	0.0	0.0	0.0	
a0133669	1.0	0.0	0.0	0.0	0.0	0.0	0.0	

Standardization of features

In [ ]:
```
std = StandardScaler().set_output(transform='pandas')
std.fit_transform(X_train[numerical_cols]).head()
```

Out[ ]:

id	spkid	H	epoch	e	a	q	i	
a0509205	-0.146100	0.965208	0.008119	0.069394	-1.003186	-0.812420	-0.853758	0.7
a0400500	-0.164644	-0.483772	0.008119	-0.024884	1.193311	0.917679	0.445587	-0.7
a0476160	-0.151738	0.321217	0.008119	-0.918697	0.640034	1.031563	0.258119	-1.1
a0059146	-0.222875	-1.208262	0.008119	0.368049	-1.006616	-0.958934	-0.781785	-1.2
a0133669	-0.210162	-1.047264	0.008119	-0.190586	0.245718	0.285911	0.066424	0.0

Pipeline for data preprocessing

In [ ]:
```
from sklearn.compose import make_column_transformer

column_transformer = make_column_transformer(
    # Numerical columns
    (
        StandardScaler().set_output(transform='pandas'),
        numerical_cols
    ),
    # Categorical columns
    (
        OneHotEncoder(sparse_output=False, handle_unknown='ignore').set_outp
        categorical_cols
```

```
    ),
    remainder='passthrough',
    verbose_feature_names_out=False
).set_output(transform='pandas')

X_train = column_transformer.fit_transform(X_train)
X_val = column_transformer.transform(X_val)
X_test = column_transformer.transform(X_test)
X_train.head()
```

Out[ ]:

id	spkid	H	epoch	e	a	q	i	
a0509205	-0.146100	0.965208	0.008119	0.069394	-1.003186	-0.812420	-0.853758	0.7
a0400500	-0.164644	-0.483772	0.008119	-0.024884	1.193311	0.917679	0.445587	-0.7
a0476160	-0.151738	0.321217	0.008119	-0.918697	0.640034	1.031563	0.258119	-1.1
a0059146	-0.222875	-1.208262	0.008119	0.368049	-1.006616	-0.958934	-0.781785	-1.2
a0133669	-0.210162	-1.047264	0.008119	-0.190586	0.245718	0.285911	0.066424	0.0

Handling imbalanced data

Oversampling of the minority class

In [ ]:
```
oversample = SMOTE(sampling_strategy=0.25, random_state=seed_value, k_neighb
X_train, y_train = oversample.fit_resample(X_train, y_train)
np.unique(y_train, return_counts=True)
```

Out[ ]:
```
(array([0, 1], dtype=uint8), array([531484, 132871]))
```

Undersampling of the majority class

In [ ]:
```
from imblearn.under_sampling import RandomUnderSampler
undersample = RandomUnderSampler(random_state=42)
X_train, y_train = undersample.fit_resample(X_train, y_train)
np.unique(y_train, return_counts=True)
```

Out[ ]:
```
(array([0, 1], dtype=uint8), array([132871, 132871]))
```

Feature selection

In [ ]:
```
sfs_forward = SequentialFeatureSelector(estimator=LogisticRegression(),
                                        n_features_to_select=12,
                                        direction="forward",
                                        scoring='roc_auc',
```

```
                                                cv=5).set_output(transform='pandas')

sfs_forward.fit(X_train, y_train)

selected_features = sfs_forward.get_feature_names_out()
selected_features
```

Out[ ]:
```
array(['H', 'i', 'om', 'w', 'ma', 'n', 'moid', 'diametro', 'class_IEO',
       'class_IMB', 'class_MCA', 'class_OMB'], dtype=object)
```

In [ ]:
```
X_train = sfs_forward.transform(X_train)
X_val = sfs_forward.transform(X_val)
X_test = sfs_forward.transform(X_test)
```

Save the training, validation, and testing sets for future use

In [ ]:
```
train = X_train.copy()
test = X_test.copy()
val = X_val.copy()

train["pha"] = y_train
test["pha"] = y_test
val["pha"] = y_val

train.to_csv(dest_path+'train.csv', index=True, index_label="id")
test.to_csv(dest_path+'test.csv', index=True, index_label="id")
val.to_csv(dest_path+'val.csv', index=True, index_label="id")
```

Recovering training, validation, and testing sets

This approach establishes a new starting point, so we don't need to run the entire notebook from scratch again.

In [ ]:
```
train = pd.read_csv(dest_path+'train.csv', engine='c', low_memory=False, ind
test = pd.read_csv(dest_path+'test.csv', engine='c', low_memory=False, index
val = pd.read_csv(dest_path+'val.csv', engine='c', low_memory=False, index_c

X_train, y_train = train.drop(["pha"], axis=1), train["pha"]
X_test, y_test = test.drop(["pha"], axis=1), test["pha"]
X_val, y_val = val.drop(["pha"], axis=1), val["pha"]
```

In [ ]:
```
X_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 265742 entries, 520093 to 664354
Data columns (total 12 columns):
 #   Column     Non-Null Count   Dtype  
---  ------     --------------   -----  
 0   H          265742 non-null  float64
 1   i          265742 non-null  float64
 2   om         265742 non-null  float64
 3   w          265742 non-null  float64
 4   ma         265742 non-null  float64
 5   n          265742 non-null  float64
 6   moid       265742 non-null  float64
 7   diametro   265742 non-null  float64
 8   class_IEO  265742 non-null  float64
 9   class_IMB  265742 non-null  float64
 10  class_MCA  265742 non-null  float64
 11  class_OMB  265742 non-null  float64
dtypes: float64(12)
memory usage: 26.4 MB
```

Model training

When choosing and training a model, we have several tools available to evaluate its performance. One of these tools is the **confusion matrix**, which explicitly shows when one class is confused with another. This allows us to work separately with different types of errors. The confusion matrix looks like this:

		Prediction	
		Negative	Positive
Actual	Negative	TN	FP
	Positive	FN	TP

Where we can observe, for our case study:

- **Negative case**: Harmless asteroid (**AI**)
- **Positive case**: Potentially hazardous asteroid (**PHA**)
- **TN**: The prediction says the asteroid is **AI** when it really is **AI**.
- **TP**: The prediction says the asteroid is **PHA** when it really is **PHA**.
- **FN**: The prediction says the asteroid is **AI** when it really is **PHA**.
- **FP**: The prediction says the asteroid is **PHA** when it really is **AI**.

The worst case scenario is an **FN** prediction, as we would be determining that an asteroid is harmless when it is actually potentially hazardous, ignoring the possibility of an imminent collision in the near future. For this reason, we will aim to reduce this type of prediction.

However, we should not neglect the **FP** errors, as our model would become useless.

The metric that will help us analyze the number of **FN** predictions is known as ***Recall***, such that:

$$Recall = \frac{TP}{TP+FN} \quad \text{if} \quad FN \to 0 \implies Recall \to 1$$

Additionally:

$$Precision = \frac{TP}{TP+FP} \quad \text{if} \quad FP \to 0 \implies Precision \to 1$$

$$F1 = \frac{TP}{TP+\frac{FN+FP}{2}} \quad \text{if} \quad FN, FP \to 0 \implies F1 \to 1$$

In [ ]:
```
def get_best_estimator(estimator, params, X_val, y_val, cv=5):
    grid = GridSearchCV(estimator(), params, scoring=["f1", "recall"], refit=
    grid.fit(X_train, y_train)
    print(f"Best parameters: {grid.best_params_}")
    return grid.best_estimator_
```

Logistic Regression

In [ ]:
```
start = time.time()

lr = LogisticRegression().fit(X_train, y_train)
y_pred_lr = lr.predict(X_test)

end = time.time()
lr_time = end - start
print(f"Total training and prediction time: {lr_time} seconds")
```

```
Total training and prediction time: 0.7285270690917969 seconds
```

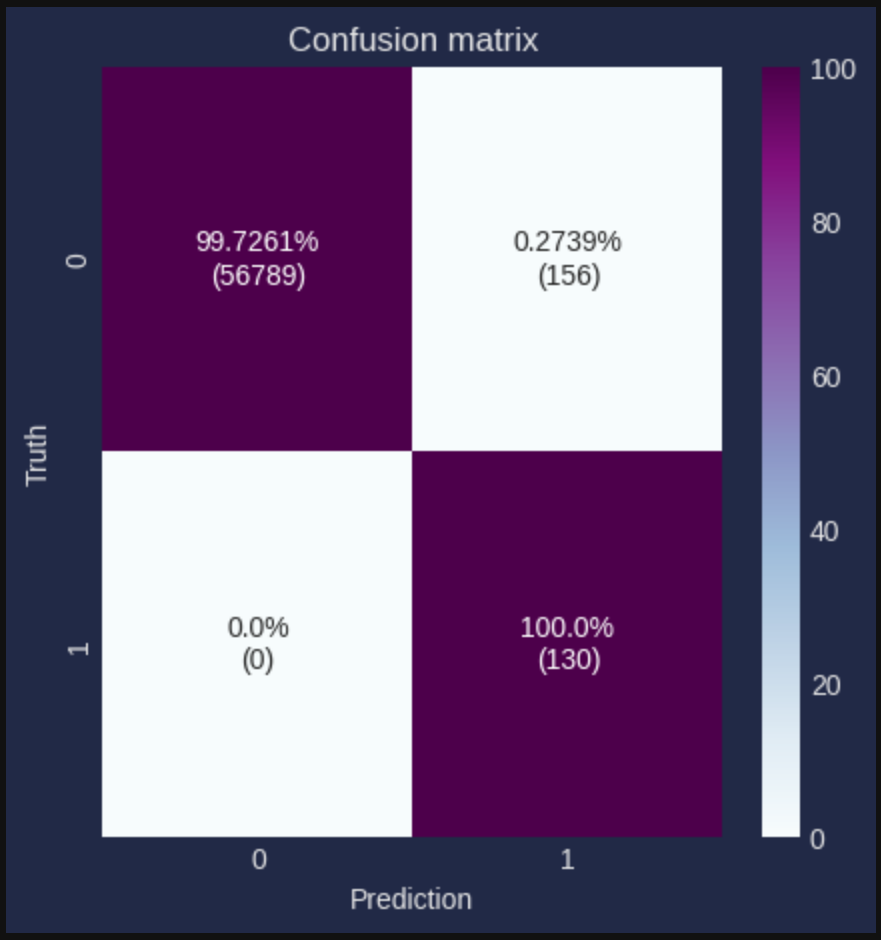
In [ ]:
```
pd.DataFrame(data=classification_report(y_test, y_pred_lr, digits=6, output_d
```

Out[ ]:

	precision	recall	f1-score	support
0	1.000000	0.997261	0.998628	56945.000000
1	0.454545	1.000000	0.625000	130.000000
accuracy	0.997267	0.997267	0.997267	0.997267
macro avg	0.727273	0.998630	0.811814	57075.000000
weighted avg	0.998758	0.997267	0.997777	57075.000000

In [ ]:
```
fig = plt.figure(figsize=(5, 5))

cm_val = confusion_matrix(y_test, y_pred_lr)
cm_pgs = np.round(confusion_matrix(y_test, y_pred_lr, normalize='true')*100,

formatted_text = (np.asarray([f"{pgs}%\n({val})" for val, pgs in zip(cm_val.

sns.heatmap(cm_pgs, annot=formatted_text, fmt='', cmap='BuPu')
```

```
plt.title("Confusion matrix")
plt.xlabel("Prediction")
plt.ylabel("Truth");
```



In [ ]:
```
scores = cross_val_score(lr, X_test, y_test, cv=5)
print(f"Mean accuracy: {scores.mean()}")
```

```
Mean accuracy: 0.9987910643889618
```

Logistic Regression with hyperparameter tuning

In [ ]:
```
start = time.time()

params = {"penalty": ["l2"],
          "C": [300, 400, 500],
          "solver": ["lbfgs", "liblinear"],
          "max_iter": [75, 100, 125],
          "random_state":[42]
          }

lr_best = get_best_estimator(LogisticRegression, params, X_val, y_val)
lr_best.fit(X_train, y_train)
y_pred_lr_best = lr_best.predict(X_test)

end = time.time()
lr_best_time = end - start
print(f"Total training and prediction time: {lr_best_time} seconds")
```

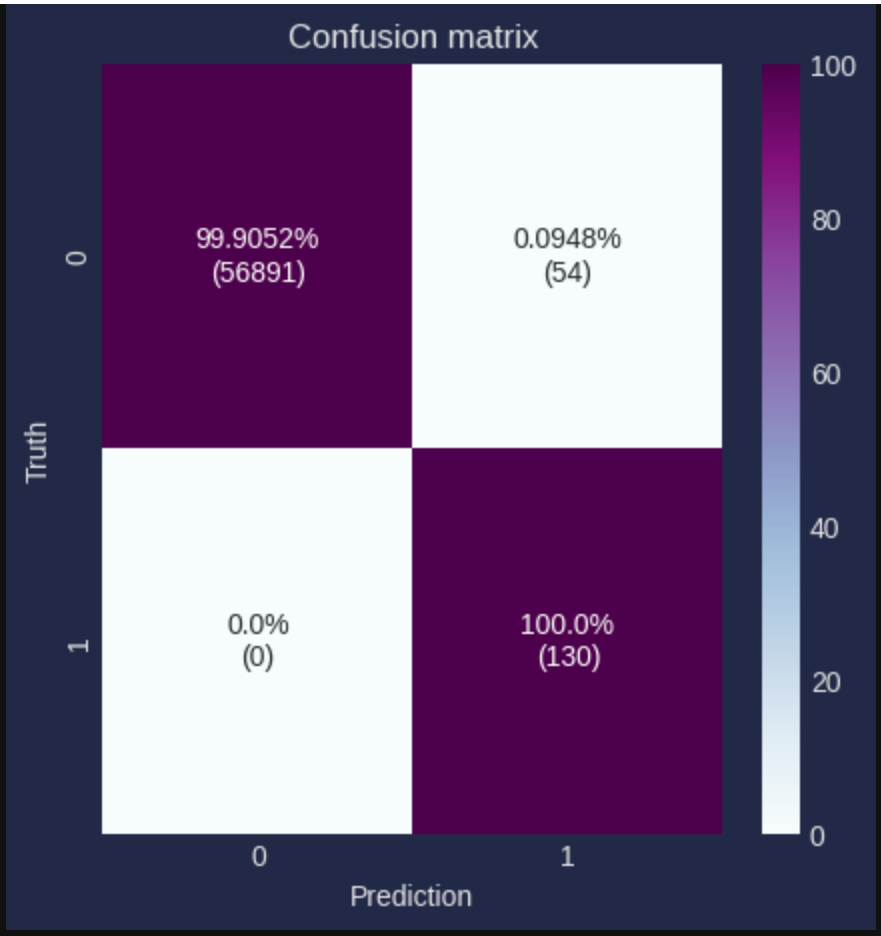
```
Fitting 5 folds for each of 18 candidates, totalling 90 fits
Best parameters: {'C': 400, 'max_iter': 75, 'penalty': 'l2', 'random_state':
42, 'solver': 'lbfgs'}
Total training and prediction time: 80.62852334976196 seconds
```

In [ ]:
```
pd.DataFrame(data=classification_report(y_test, y_pred_lr_best, digits=6, out
```

Out[ ]:

	precision	recall	f1-score	support
0	1.000000	0.999052	0.999526	56945.000000
1	0.706522	1.000000	0.828025	130.000000
accuracy	0.999054	0.999054	0.999054	0.999054
macro avg	0.853261	0.999526	0.913776	57075.000000
weighted avg	0.999332	0.999054	0.999135	57075.000000

In [ ]:
```
fig = plt.figure(figsize=(5, 5))

cm_val = confusion_matrix(y_test, y_pred_lr_best)
cm_pgs = np.round(confusion_matrix(y_test, y_pred_lr_best, normalize='true')

formatted_text = (np.asarray([f"{pgs}%\n({val})" for val, pgs in zip(cm_val.

sns.heatmap(cm_pgs, annot=formatted_text, fmt='', cmap='BuPu')
plt.title("Confusion matrix")
plt.xlabel("Prediction")
plt.ylabel("Truth");
```



In [ ]:
```
scores = cross_val_score(lr_best, X_test, y_test, cv=5)
print(f"Mean accuracy: {scores.mean()}")
```

```
Mean accuracy: 0.9993166885676741
```

K-Nearest Neighbors

In [ ]:
```
start = time.time()

knn = KNeighborsClassifier(n_neighbors=3, leaf_size=50).fit(X_train, y_train)
y_pred_knn = knn.predict(X_test)

end = time.time()
knn_time = end - start
print(f"Total training and prediction time: {knn_time} seconds")
```

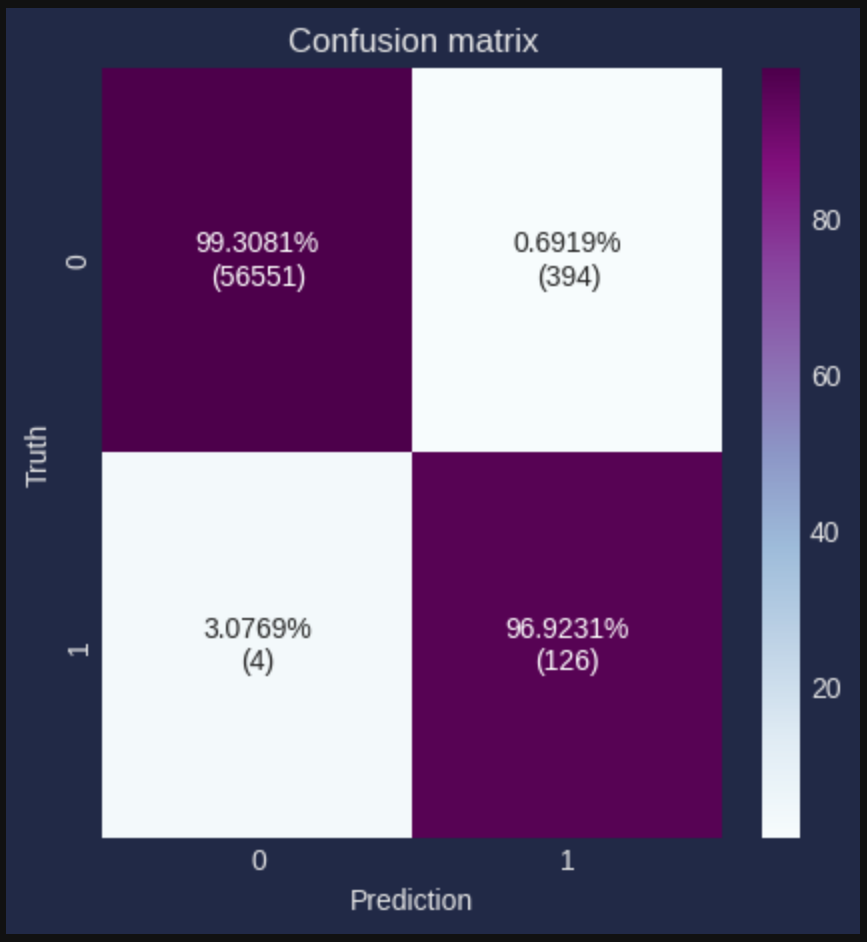
```
Total training and prediction time: 16.01733636856079 seconds
```

In [ ]:
```
pd.DataFrame(data=classification_report(y_test, y_pred_knn, digits=6, output_
```

Out[ ]:

	precision	recall	f1-score	support
0	0.999929	0.993081	0.996493	56945.000000
1	0.242308	0.969231	0.387692	130.000000
accuracy	0.993027	0.993027	0.993027	0.993027
macro avg	0.621118	0.981156	0.692093	57075.000000
weighted avg	0.998204	0.993027	0.995107	57075.000000

In [ ]:
```
fig = plt.figure(figsize=(5, 5))

cm_val = confusion_matrix(y_test, y_pred_knn)
cm_pgs = np.round(confusion_matrix(y_test, y_pred_knn, normalize='true')*100

formatted_text = (np.asarray([f"{pgs}%\n({val})" for val, pgs in zip(cm_val.

sns.heatmap(cm_pgs, annot=formatted_text, fmt='', cmap='BuPu')
plt.title("Confusion matrix")
plt.xlabel("Prediction")
plt.ylabel("Truth");
```



In [ ]:
```
scores = cross_val_score(knn, X_test, y_test, cv=5)
print(f"Mean accuracy: {scores.mean()}")
```

```
Mean accuracy: 0.9974419623302673
```

Random Forest

In [ ]:
```
start = time.time()

rf = RandomForestClassifier(n_estimators=200, max_depth=10, random_state=42).
y_pred_rf = rf.predict(X_test)

end = time.time()
rf_time = end - start
print(f"Total training and prediction time: {rf_time} seconds")
```

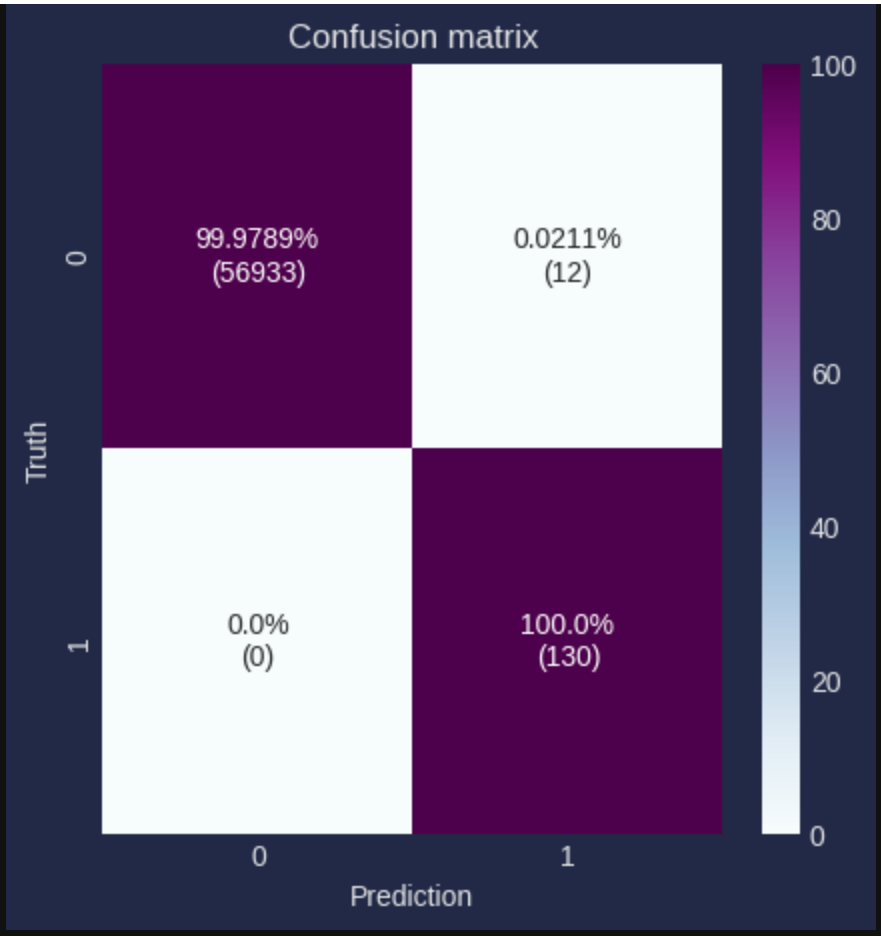
```
Total training and prediction time: 120.76419806480408 seconds
```

In [ ]:
```
pd.DataFrame(data=classification_report(y_test, y_pred_rf, digits=6, output_d
```

Out[ ]:

	precision	recall	f1-score	support
0	1.000000	0.999789	0.999895	56945.00000
1	0.915493	1.000000	0.955882	130.00000
accuracy	0.999790	0.999790	0.999790	0.99979
macro avg	0.957746	0.999895	0.977888	57075.00000
weighted avg	0.999808	0.999790	0.999794	57075.00000

In [ ]:
```
fig = plt.figure(figsize=(5, 5))

cm_val = confusion_matrix(y_test, y_pred_rf)
cm_pgs = np.round(confusion_matrix(y_test, y_pred_rf, normalize='true')*100,

formatted_text = (np.asarray([f"{pgs}%\n({val})" for val, pgs in zip(cm_val.

sns.heatmap(cm_pgs, annot=formatted_text, fmt='', cmap='BuPu')
plt.title("Confusion matrix")
plt.xlabel("Prediction")
plt.ylabel("Truth");
```



In [ ]:
```
scores = cross_val_score(rf, X_test, y_test, cv=5)
print(f"Mean accuracy: {scores.mean()}")
```

```
Mean accuracy: 0.9999123959702146
```

Support Vector Machine

In [ ]:
```
start = time.time()

svc = SVC(C=10, random_state=42).fit(X_train, y_train)
y_pred_svc = svc.predict(X_test)

end = time.time()
svc_time = end - start
print(f"Total training and prediction time: {svc_time} seconds")
```

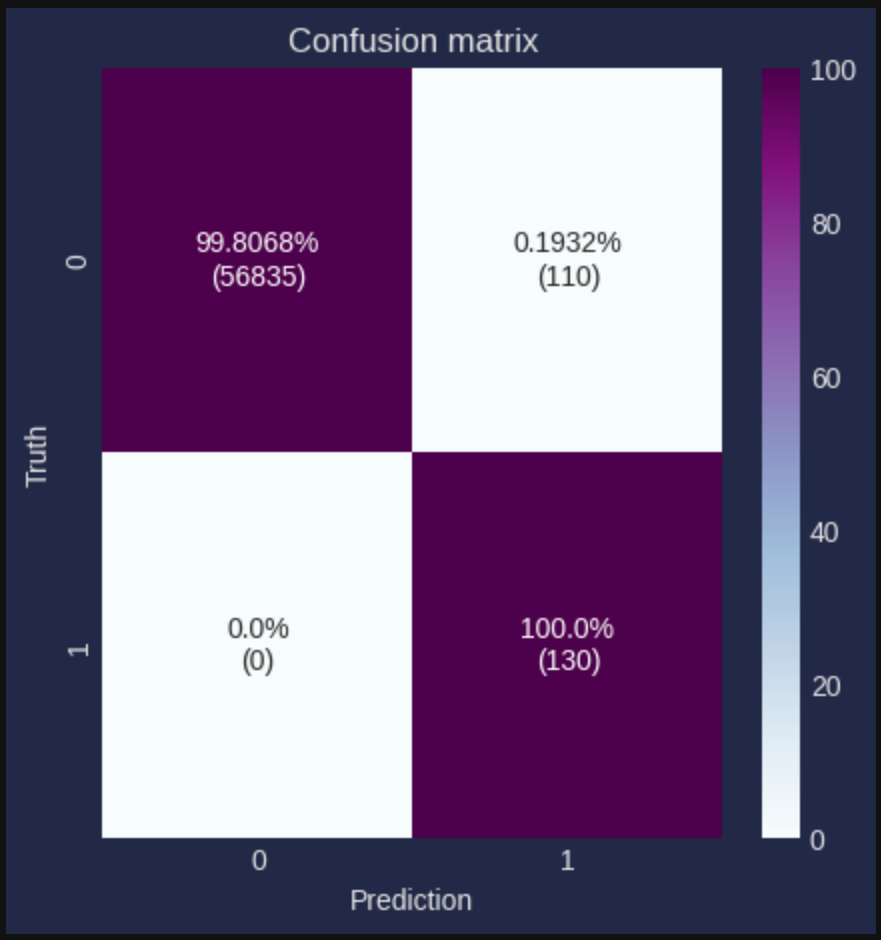
```
Total training and prediction time: 47.52535915374756 seconds
```

In [ ]:
```
pd.DataFrame(data=classification_report(y_test, y_pred_svc, digits=6, output_
```

Out[ ]:

	precision	recall	f1-score	support
0	1.000000	0.998068	0.999033	56945.000000
1	0.541667	1.000000	0.702703	130.000000
accuracy	0.998073	0.998073	0.998073	0.998073
macro avg	0.770833	0.999034	0.850868	57075.000000
weighted avg	0.998956	0.998073	0.998358	57075.000000

In [ ]:
```
fig = plt.figure(figsize=(5, 5))

cm_val = confusion_matrix(y_test, y_pred_svc)
cm_pgs = np.round(confusion_matrix(y_test, y_pred_svc, normalize='true')*100

formatted_text = (np.asarray([f"{pgs}%\n({val})" for val, pgs in zip(cm_val.

sns.heatmap(cm_pgs, annot=formatted_text, fmt='', cmap='BuPu')
plt.title("Confusion matrix")
plt.xlabel("Prediction")
plt.ylabel("Truth");
```



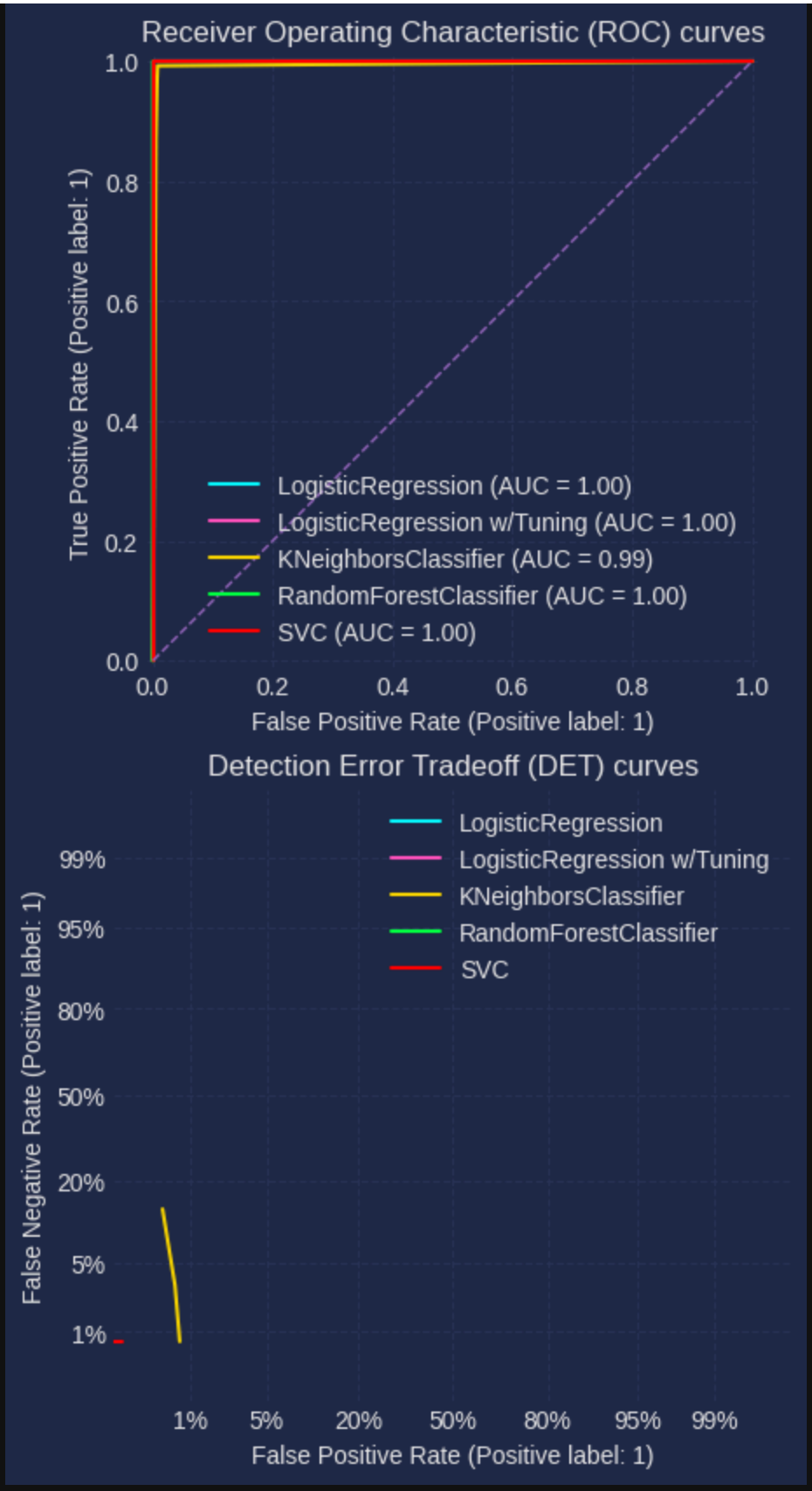
In [ ]:
```
scores = cross_val_score(svc, X_test, y_test, cv=5)
print(f"Mean accuracy: {scores.mean()}")
```

```
Mean accuracy: 0.9980551905387648
```

Comparing the models

ROC and DET curves

In [ ]:
```
adjust_display()
fig, [ax_roc, ax_det] = plt.subplots(2, 1, figsize=(5, 10))

RocCurveDisplay.from_estimator(lr, X_test, y_test, ax=ax_roc)
RocCurveDisplay.from_estimator(lr_best, X_test, y_test, ax=ax_roc, name="Log
RocCurveDisplay.from_estimator(knn, X_test, y_test, ax=ax_roc)
RocCurveDisplay.from_estimator(rf, X_test, y_test, ax=ax_roc)
RocCurveDisplay.from_estimator(svc, X_test, y_test, ax=ax_roc)
ax_roc.plot([0, 1], [0, 1], linestyle='--', lw=1)

DetCurveDisplay.from_estimator(lr, X_test, y_test, ax=ax_det)
DetCurveDisplay.from_estimator(lr_best, X_test, y_test, ax=ax_det, name="Log
DetCurveDisplay.from_estimator(knn, X_test, y_test, ax=ax_det)
DetCurveDisplay.from_estimator(rf, X_test, y_test, ax=ax_det)
DetCurveDisplay.from_estimator(svc, X_test, y_test, ax=ax_det)

ax_roc.set_title("Receiver Operating Characteristic (ROC) curves")
ax_det.set_title("Detection Error Tradeoff (DET) curves")

ax_roc.grid(linestyle="--")
ax_det.grid(linestyle="--")

plt.legend()
plt.show()
```



Comparison table for performance models

In [ ]:
```
def print_scores(y, y_pred, pp_scores=False):
    ac, pr, rc, f1 = accuracy_score(y, y_pred)*100, precision_score(y, y_pre
    if pp_scores == True:
        print(f"Accuracy:{ac}")
        print(f"Precision:{pr}")
        print(f"Recall:{rc}")
        print(f"F1-score:{f1}")
    return {'Accuracy': ac, 'Precision':pr, 'Recall':rc, 'F1-score':f1}

lr_scores = print_scores(y_test, y_pred_lr)
lr_best_scores = print_scores(y_test, y_pred_lr_best)
knn_scores = print_scores(y_test, y_pred_knn)
rf_scores = print_scores(y_test, y_pred_rf)
svc_scores = print_scores(y_test, y_pred_svc)

scores = pd.DataFrame(data=[list(lr_scores.values()),
                            list(lr_best_scores.values()),
                            list(knn_scores.values()),
                            list(rf_scores.values()),
                            list(svc_scores.values()),
                           ], columns=list(lr_scores.keys()))

scores = scores.transpose()
scores = scores.rename(columns={0:"Linear Regression",
                                1:"Linear Regression w/Tuning",
                                2:"K-Nearest Neighbors",
                                3:"Random Forest",
                                4:"C-Support Vector",
                               })
scores.style.highlight_max(color = 'green', axis = 1).highlight_min(color =
```

Out[ ]:

	Linear Regression	Linear Regression w/Tuning	K-Nearest Neighbors	Random Forest	C-Support Vector
Accuracy	99.726675	99.905388	99.302672	99.978975	99.807271
Precision	45.454545	70.652174	24.230769	91.549296	54.166667
Recall	100.000000	100.000000	96.923077	100.000000	100.000000
F1-score	99.777736	99.913501	99.510672	99.979438	99.835827

Comparison table for training time consumption

In [ ]:
```
times = pd.DataFrame(data={"Time taken (seconds)": [lr_time, lr_best_time, k
times = times.sort_values(by=["Time taken (seconds)"])
times.style.highlight_min(color = 'green', axis = 0).highlight_max(color = '
```

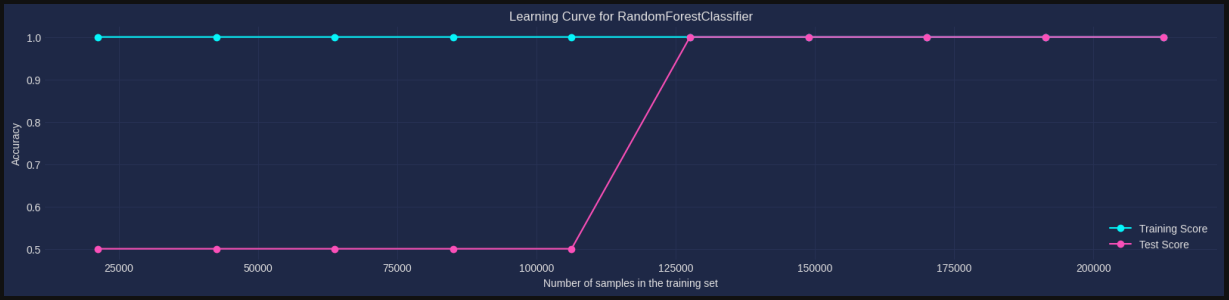
Out[ ]:

	Time taken (seconds)
Logistic Regression	0.728527
K-Nearest Neighbors	16.017336
C-Support Vector	47.525359
Logistic Regression w/tuning	80.628523
Random Forest	120.764198

Analizing the best model

Learning curve

In [ ]:
```
rf_lc = RandomForestClassifier(n_estimators=200, max_depth=10, random_state=4
fig, ax = plt.subplots(nrows=1, ncols=1, figsize=(20, 4))

common_params = {
    "X": X_train,
    "y": y_train,
    "train_sizes": np.linspace(0.1, 1.0, 10),
    "score_type": "both",
    "n_jobs": -1,
    "line_kw": {"marker": "o"},
    "std_display_style": "fill_between",
    "score_name": "Accuracy",
}

LearningCurveDisplay.from_estimator(rf_lc, **common_params, ax=ax)
handles, label = ax.get_legend_handles_labels()
ax.legend(handles[:2], ["Training Score", "Test Score"])
ax.set_title(f"Learning Curve for {rf_lc.__class__.__name__}");
```



In [ ]:
```
print(f"We have determined that the optimal performance is achieved when usi
```

```
We have determined that the optimal performance is achieved when using 49% to
56% of the training set.
```

Random Forest trained with 50~60% of the dataset

In [ ]:
```
X_train_min, _, y_train_min, _ = train_test_split(X_train, y_train, test_siz
print(f"We will use {round(len(X_train_min)*100/len(X_train),2)}% ({len(X_tr
```

```
We will use 45.0% (119583) of the data.
```

In [ ]:
```
start = time.time()

rf = RandomForestClassifier(n_estimators=200, max_depth=10, random_state=42).
y_pred_rf = rf.predict(X_test)

end = time.time()
rf_time = end - start
print(f"Total training and prediction time: {rf_time} seconds")
```

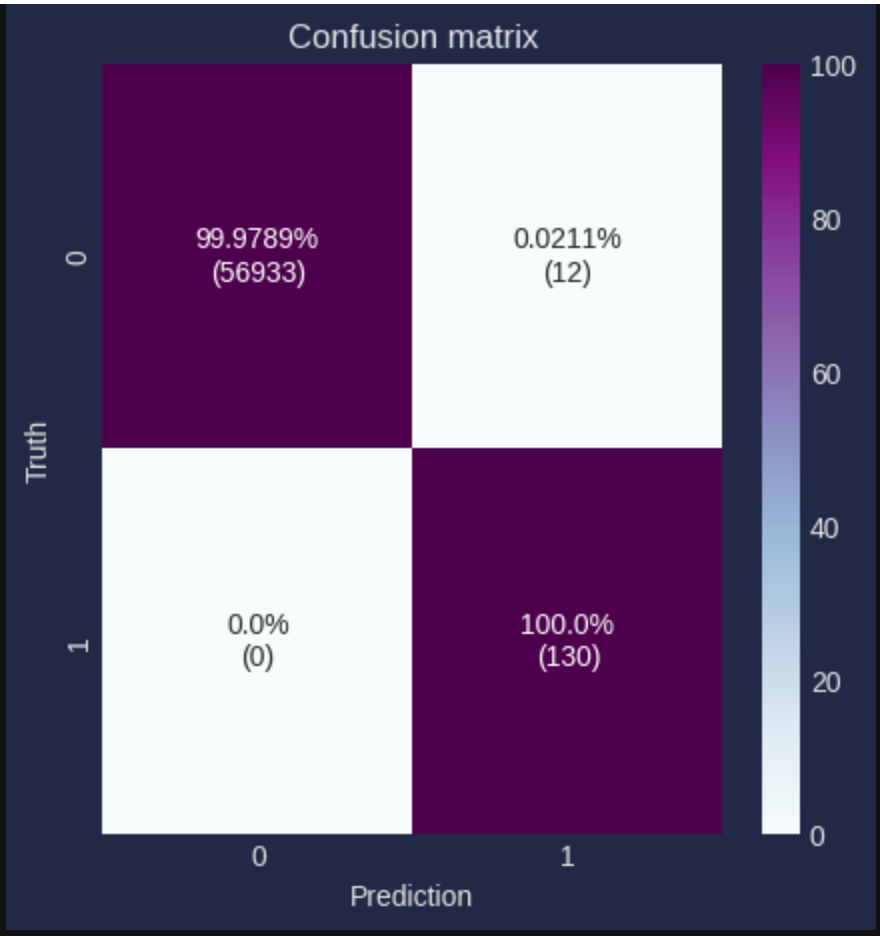
```
Total training and prediction time: 46.309218645095825 seconds
```

In [ ]:
```
pd.DataFrame(data=classification_report(y_test, y_pred_rf, digits=6, output_d
```

Out[ ]:

	precision	recall	f1-score	support
0	1.000000	0.999789	0.999895	56945.00000
1	0.915493	1.000000	0.955882	130.00000
accuracy	0.999790	0.999790	0.999790	0.99979
macro avg	0.957746	0.999895	0.977888	57075.00000
weighted avg	0.999808	0.999790	0.999794	57075.00000

In [ ]:
```
fig = plt.figure(figsize=(5, 5))

cm_val = confusion_matrix(y_test, y_pred_rf)
cm_pgs = np.round(confusion_matrix(y_test, y_pred_rf, normalize='true')*100,

formatted_text = (np.asarray([f"{pgs}%\n({val})" for val, pgs in zip(cm_val.

sns.heatmap(cm_pgs, annot=formatted_text, fmt='', cmap='BuPu')
plt.title("Confusion matrix")
plt.xlabel("Prediction")
plt.ylabel("Truth");
```

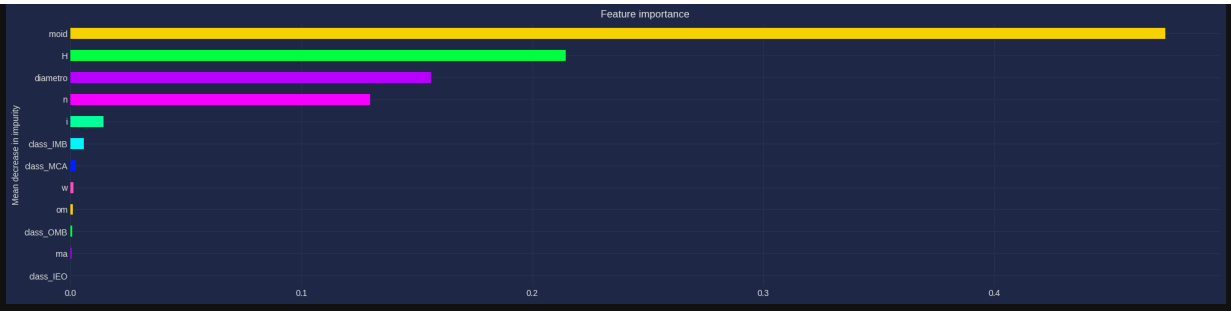


In [ ]:
```
scores = cross_val_score(rf, X_test, y_test, cv=5)
print(f"Mean accuracy: {scores.mean()}")
```

```
Mean accuracy: 0.9999123959702146
```

Important features for our model

In [ ]:
```
adjust_display()
forest_importances = pd.Series(rf.feature_importances_, index=X_train.column
colors = [
    "#00ff9f",
    '#08F7FE',  # teal/cyan
    "#001eff",
    '#FE53BB',  # pink
    '#F5D300',  # yellow
    '#00ff41',  # matrix green
    "#bd00ff",
    "#F600ff"
]
fig, ax = plt.subplots(figsize=(20, 5))
forest_importances.sort_values().plot.barh(color=list(reversed(colors)), ax=
ax.set_title("Feature importance")
ax.set_ylabel("Mean decrease in impurity")
fig.tight_layout()
```

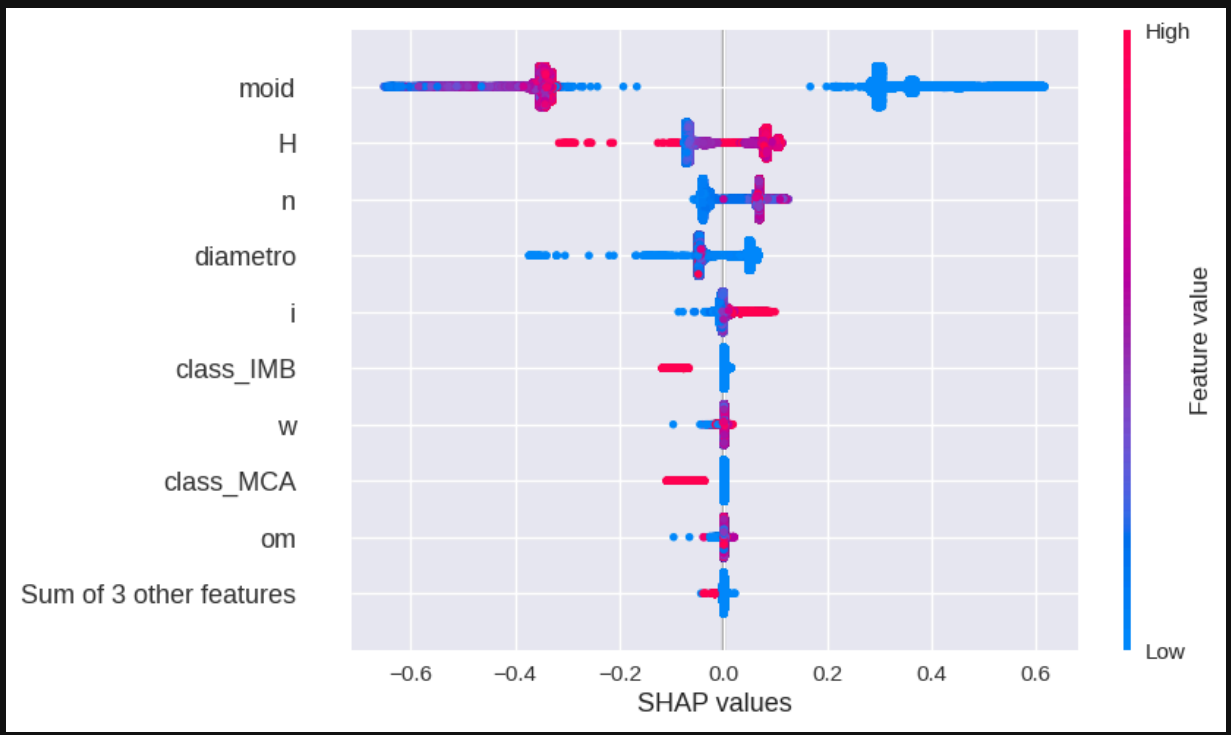


SHAP Values

In [ ]:
```
import shap

shap.initjs()
```

In [ ]:
```
explainer = shap.TreeExplainer(rf)
shap_values = explainer(X_train, y_train)
```

In [ ]:
```
plt.style.use("seaborn-v0_8")
shap.plots.beeswarm(shap_values[:,:,1], show=False)
plt.xlabel(f"SHAP values")
plt.show()
```



According to the diagram presented above, when a point (an instance) is blue, it indicates that its original feature value in the dataset is low, whereas when a point is red, it indicates that the original feature value in the dataset is high.

With this clarified, four situations can arise:

- Red point scattered towards negative SHAP values: For this instance, the feature indicates that the higher its value, the lower the prediction value, as it requires the SHAP value contribution to the prediction to be smaller.
- Red point scattered towards positive SHAP values: On the contrary, for this instance, the feature indicates that the higher its value, the higher the prediction value, as it requires the SHAP value contribution to the prediction to be larger.
- Blue point scattered towards negative SHAP values: For this instance, the feature indicates that the lower its value, the lower the prediction value, as it requires the SHAP value contribution to the prediction to be smaller.
- Blue point scattered towards positive SHAP values: For this instance, the feature indicates that the lower its value, the higher the prediction value, as it requires the SHAP value contribution to the prediction to be larger.

In this way, we can conclude that, for example, an asteroid with a low `moid` value is more likely to be classified as a PHA asteroid.

Saving the pipeline, the label encoder, the feature selector and the best model with CloudPickle

In [ ]:
```
import cloudpickle

with open(dest_path+'HAP_model.bin', 'wb') as f_out:
    cloudpickle.dump((column_transformer, le, sfs_forward, rf), f_out)
```

Conclusions

Based on the previously presented information, we can observe some important aspects. The balance of the data is a critical process for this project due to the significant imbalance in the data. This imbalance will cause a bias in the predictions of the estimators, leaning toward the majority class. If we perform oversampling, it is also important to perform undersampling to avoid artificially inflating our minority class excessively.

Feature selection is extremely helpful in reducing the volume of information that the estimators must analyze, significantly reducing training and prediction times. This selection also helps reduce the noise injected into the estimators by omitting non-redundant features in the prediction.

Regarding the performance of the estimators, several types of estimators were applied, most of which demonstrated excellent performance. However, the estimator that achieved the best results was the Random Forest estimator. Despite this, the time required for its training and

prediction was the slowest, which may need to be considered for certain applications that require minimal processing speeds.

The most relevant feature according to our estimator is `moid`. As we have already seen, this feature indicates the maximum proximity of asteroids to Earth's orbit. We assumed this factor to be important based on the visual and statistical analysis we performed, and the estimator confirms our hypothesis. This does not mean that it is the most important feature overall, but the estimator determines that it is the most useful for classification. This could vary for other estimators.

Finally, we can observe that to achieve the best performance, only about 50~60% of the training set is necessary. Using a larger sample size could lead to overfitting in our estimators.

Thank you for your time! 😃

Converting Notebook to PDF

In [1]:
```
!apt-get install texlive-xetex texlive-fonts-recommended texlive-plain-gener
!pip install pypandoc nbconvert[webpdf]
!playwright install
!playwright install-deps

from google.colab import drive
from IPython.display import clear_output
drive.mount('/content/drive')
clear_output(wait=False)
```

In [ ]:
```
%%capture
!jupyter nbconvert --to webpdf /content/drive/MyDrive/Coder/Data_Science/Haz
```