

# ITCS-6100 BIG DATA FOR COMPUTATIONAL ADVANTAGE

## DELIVERABLE 3

### PROJECT GROUP 8 TEAM MEMBERS:

- Aneela Gannarapu
- Preetham Garre
- Rachana Goli
- Sai Rithwik Reddy Bolla
- Susmitha Dalli

### COMMUNICATION PLAN:

All our team members discussed the perspectives and insights through emails and exchanged our ideas. We met via Google meet and worked on finishing the tasks accordingly.

The project's repository can be accessed on GitHub using URL that's given below:  
<https://github.com/rachanagoli/BigDataGroup8>

### DATA SET SELECTION :

We selected the data set from Kaggle active competitions officially maintained by Jet Propulsion Laboratory of California Institute of Technology which is an organization under NASA. In this Dataset all kinds of Data related to Asteroid is included to analyze.

Link to our dataset: [Asteroid Dataset](#)

The major attributes implemented to perform analysis within the dataset are detailed as below:

- SPK-ID: Object primary SPK-ID
- Object ID: Object internal database ID
- Object fullname: Object full name/designation
- pdes: Object primary designation
- name: Object IAU name
- NEO: Near-Earth Object (NEO) flag
- PHA: Potentially Hazardous Asteroid (PHA) flag
- H: Absolute magnitude parameter
- Diameter: object diameter (from equivalent sphere) km Unit
- Albedo: Geometric albedo
- Diameter\_sigma: 1-sigma uncertainty in object diameter km Unit
- Orbit\_id: Orbit solution ID

- Epoch: Epoch of osculation in modified Julian day form
- Equinox: Equinox of reference frame
- e: Eccentricity
- a: Semi-major axis au Unit
- q: perihelion distance au Unit
- i: inclination; angle with respect to x-y ecliptic plane
- tp: Time of perihelion passage TDB Unit
- moid\_ld: Earth Minimum Orbit Intersection Distance Unit

## **BUSINESS PROBLEM OR OPPORTUNITY:**

The asteroid dataset offers valuable insights and opportunities for multiple domains, including astronomy and space exploration. It contains information on the materialistic characteristics and the orbital details of more than 17,000 asteroids. Moreover, detecting and monitoring potentially dangerous asteroids can help avert catastrophic incidents, such as an asteroid collision with Earth. The dataset also facilitates the identification of valuable resources, like water and metals, on asteroids, which can pave the way for future asteroid mining expeditions. Consequently, the asteroid dataset provides critical data for diverse scientific and commercial use cases in the field of space exploration.

## **RESEARCH OBJECTIVES:**

The main aim was to provide a follow-up based on demand and assist in reaching the business goals. To carry out the solution, we decided to use AWS technologies. We planned to execute relevant designs and algorithms to find out the flow in demand that was most appropriate in examining the dataset.

Evaluating the performance of different machine learning algorithms for predicting whether an asteroid is potentially hazardous or not. Identify any patterns or trends in the probability of an asteroid colliding with the Earth based on its orbit class and eccentricity. Investigate the potential impacts of asteroids that are bound to the solar system and examine how they are distributed amongst the planets. Identify any areas the diameter of an asteroid in the solar system.

## **ANALYTICS AND MACHINE LEARNING:**

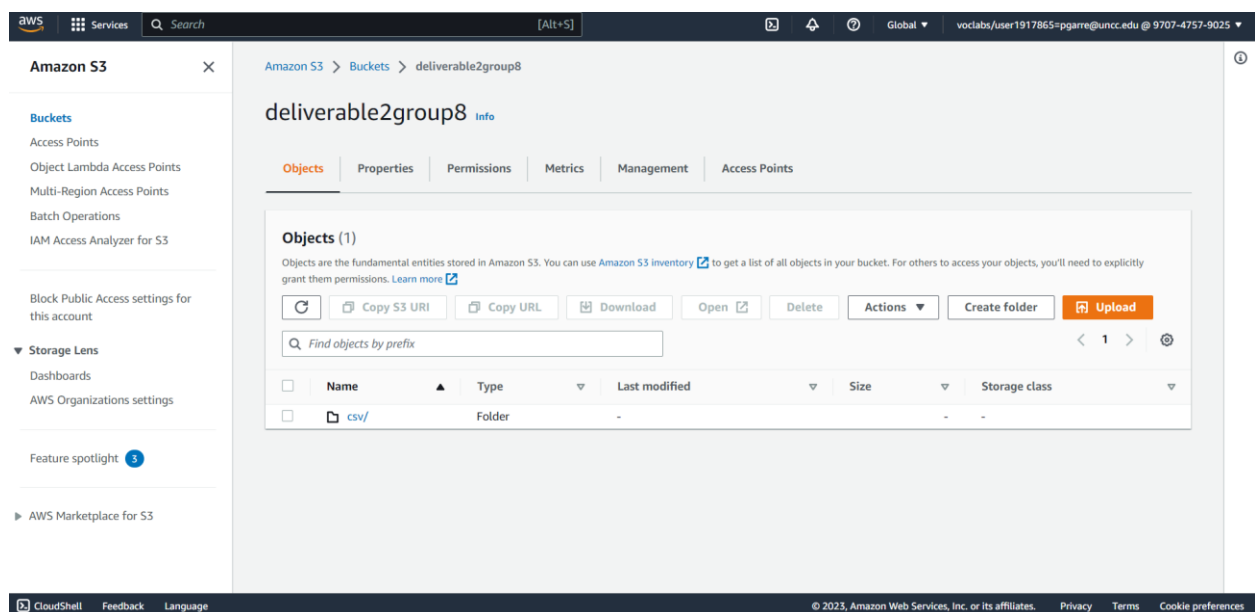
The Asteroid dataset comprises of 45 features that offer extensive insights into various aspects of asteroids. Prior to constructing an analytical model, the data underwent cleaning procedures, which involved removing null values and special characters. To filter out any outliers, the Z-score approach was employed, which rates each data point based on its originality and magnitude. The dataset was split into two sections, with 80% of the data assigned to the training set and 20% allocated to the testing set.

Various models were generated to make predictions on the asteroid dataset such as Random Forest, XGBoost, Decision Tree, Naïve Bayes, Logistic Regression, and K-Nearest Neighbors. To evaluate the effectiveness of the models, standard metrics such as Accuracy and F1 score were used along with Precision and Recall. Accuracy measures the proportion of correct predictions made by the model, while MSE (Mean Squared Error) assesses how well the model's predictions agree with the data.

Analytics and machine learning in Amazon are powered by machine learning technologies such as SageMaker. The dataset underwent recognition using Amazon Athena and Glue before being further processed by SageMaker to leverage the best ML tools for its structure and produce the most accurate results. The dataset's schema was meticulously examined to ensure optimal use of ML tools, and Amazon SageMaker was utilized to obtain the results.

## Implementation:

The data was first uploaded to an S3 bucket, and then the AWS S3 copy command was used to load it into a Jupyter notebook instance. Subsequently, the data was merged into a single data frame.



After the dataset was loaded in the terminal, it was loaded into a data frame thereby proving the established connection.

```
Deliverable3 Group 8

In [28]: import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler
from scipy.stats import zscore
import plotly.express as px
import plotly.graph_objects as go
from plotly.subplots import make_subplots

In [4]: df = pd.read_csv('/home/ec2-user/dataset.csv', low_memory=False)
```

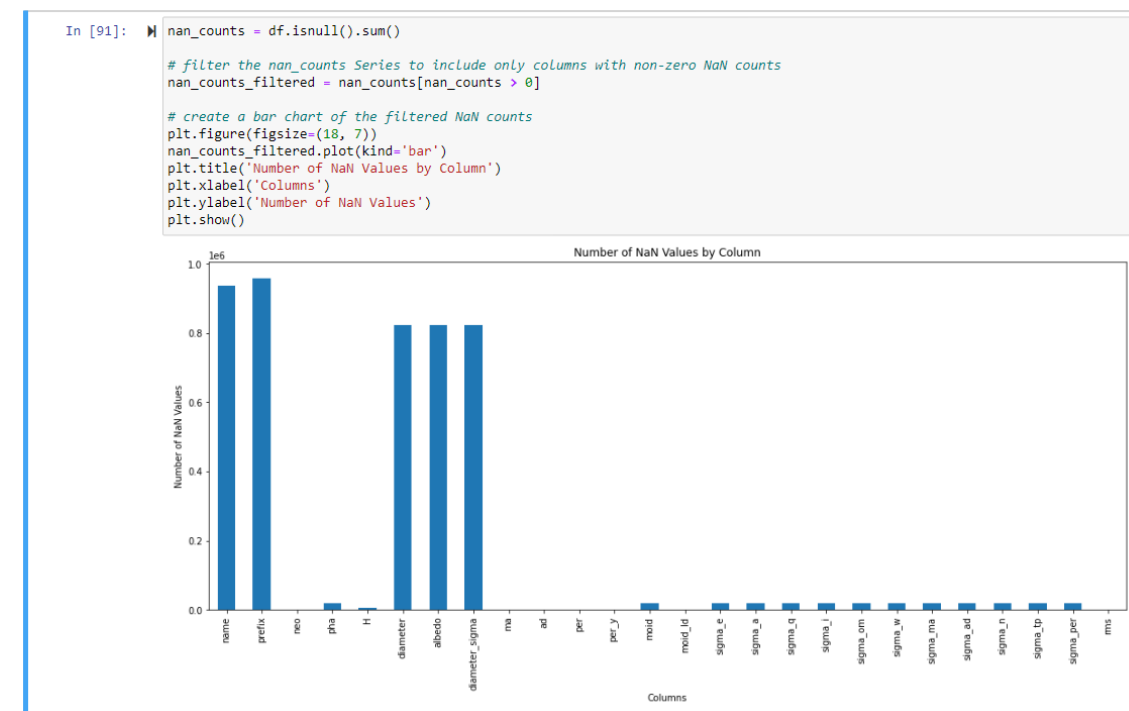
## Understanding the dataset.

```
In [5]: df.shape
Out[5]: (958524, 45)

In [6]: df.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 958524 entries, 0 to 958523
Data columns (total 45 columns):
#   Column              Non-Null Count  Dtype
---  --
0    id                   958524 non-null  object
1    spkid                958524 non-null  int64
2    full_name            958524 non-null  object
3    pdes                 958524 non-null  object
4    name                 22064 non-null   object
5    prefix              18 non-null      object
6    neo                  958520 non-null  object
7    pha                  938603 non-null  object
8    H                    952261 non-null  float64
9    diameter             136209 non-null  float64
10   albedo               135103 non-null  float64
11   diameter_sigma       136081 non-null  float64
12   orbit_id             958524 non-null  object
13   epoch                958524 non-null  float64
14   epoch_mjd            958524 non-null  int64
15   epoch_cal            958524 non-null  float64
16   equinox              958524 non-null  object
17   e                    958524 non-null  float64
18   a                    958524 non-null  float64
19   q                    958524 non-null  float64
20   i                    958524 non-null  float64
21   om                   958524 non-null  float64
22   w                    958524 non-null  float64
23   ma                   958523 non-null  float64
24   ad                   958520 non-null  float64
25   n                    958524 non-null  float64
26   tp                   958524 non-null  float64
27   tp_cal               958524 non-null  float64
28   per                  958520 non-null  float64
29   per_y                958523 non-null  float64
30   moid                 938603 non-null  float64
31   moid_id              958397 non-null  float64
32   sigma_e              938602 non-null  float64
33   sigma_a              938602 non-null  float64
34   sigma_q              938602 non-null  float64
35   sigma_i              938602 non-null  float64
36   sigma_om             938602 non-null  float64
37   sigma_w              938602 non-null  float64
38   sigma_ma             938602 non-null  float64
39   sigma_ad             938598 non-null  float64
40   sigma_n              938602 non-null  float64
41   sigma_tp             938602 non-null  float64
42   sigma_per            938598 non-null  float64
43   class                958524 non-null  object
44   rms                  958522 non-null  float64
dtypes: float64(33), int64(2), object(10)
memory usage: 329.1+ MB
```

## Data Pre-processing:

The null values have been identified and processed to clean the data.



The duplicate values have been searched but found none.

```
In [12]: duplicate = df[df.duplicated()]
print("Duplicate Rows : ", len(duplicate))
duplicate

Duplicate Rows : 0

Out[12]:
```

id	spkid	full_name	pdes	name	prefix	neo	pha	H	diameter	...	sigma_i	sigma_om	sigma_w	sigma_ma	sigma_ad	sigma_n	sigma_tp	sigma_pe
----	-------	-----------	------	------	--------	-----	-----	---	----------	-----	---------	----------	---------	----------	----------	---------	----------	----------

0 rows x 45 columns

The missing values found maximum in the respective columns have been searched to drop accordingly since the columns with huge proportions of missing values are insignificant in performing analysis.

```
In [92]: # Columns with missing values in Percentage
missing_cols = df.isna().mean() * 100
missing_cols = missing_cols[missing_cols > 0]
print("Percentage of missing values:\n", missing_cols)

Percentage of missing values:
name          97.698128
prefix        99.998122
neo           0.000417
pha           2.078300
H             0.653400
diameter      85.789714
albedo        85.905100
diameter_sigma 85.803068
ma            0.000104
ad            0.000417
per           0.000417
per_y         0.000104
moid         2.078300
moid_ld       0.013250
sigma_e       2.078404
sigma_a       2.078404
sigma_q       2.078404
sigma_i       2.078404
sigma_om      2.078404
sigma_w       2.078404
sigma_ma      2.078404
sigma_ad      2.078821
sigma_n       2.078404
sigma_tp      2.078404
sigma_per     2.078821
rms           0.000209
dtype: float64
```

The correlations between the attributes and decision attribute have been identified using a heat map visualization.



The insignificant columns have been dropped to de-clutter the dataset.

```
In [30]: # Drop Columns to remain with Osculating Orbital Elements
df = df[['spkid', 'full_name', 'orbit_id',
        'e', 'a', 'q', 'i', 'n', 'tp', 'per', 'per_y',
        'class',
        'rms']].copy()
df.shape

Out[30]: (958524, 13)
```

The remaining columns have been renamed for improving data understanding.

```
In [31]: df = df.rename(columns = {
        'e': 'Eccentricity',
        'a': 'Semi_Major_Axis',
        'q': 'PerihelionDistance',
        'i': 'Inclination',
        'M': 'MeanAnomaly',
        'tp': 'Time_Perihelion_Passage',
        'n': 'Mean_Motion',
        'Q': 'Aphelion_Distance',
        'full_name': 'FullName',
        'spkid': 'SPKID',
        'class': 'Classification',
        'per': 'Period_Days',
        'per_y': 'Period_Years'
    }).copy()
df.head(10)
```

```
Out[31]:
```

	SPKID	FullName	orbit_id	Eccentricity	Semi_Major_Axis	PerihelionDistance	Inclination	Mean_Motion	Time_Perihelion_Passage	Period_Days	Period
0	2000001	1 Ceres	JPL 47	0.076009	2.769165	2.558684	10.594067	0.213885	2.458239e+06	1683.145703	4.1
1	2000002	2 Pallas	JPL 37	0.229972	2.773841	2.135935	34.832932	0.213345	2.458321e+06	1687.410992	4.1
2	2000003	3 Juno	JPL 112	0.256936	2.668285	1.982706	12.991043	0.226129	2.458446e+06	1592.013769	4.1
3	2000004	4 Vesta	JPL 35	0.088721	2.361418	2.151909	7.141771	0.271609	2.458248e+06	1325.432763	3.1
4	2000005	5 Astraea	JPL 114	0.190913	2.574037	2.082619	5.367427	0.238661	2.458926e+06	1508.414421	4.1
5	2000006	6 Hebe	JPL 89	0.203219	2.424533	1.931822	14.739653	0.261073	2.459649e+06	1378.924506	3.1
6	2000007	7 Iris	110	0.230145	2.387375	1.837933	5.521598	0.267192	2.459422e+06	1347.347071	3.1
7	2000008	8 Flora	JPL 118	0.155833	2.201415	1.858362	5.889081	0.301753	2.459149e+06	1193.029574	3.1
8	2000009	9 Metis	JPL 116	0.123300	2.386189	2.091972	5.576494	0.267391	2.458911e+06	1346.343282	3.1
9	2000010	10 Hygiea	JPL 96	0.112117	3.142435	2.790114	3.831786	0.176931	2.459776e+06	2034.688644	5.1

Understanding the dimensional distribution of data attributes.

```
# Figure with two subplots
fig, axes = plt.subplots(ncols=2, figsize=(10, 5))

# Count plot of the "neo" variable on the left subplot
sns.countplot(x='neo', data=df, ax=axes[0])
axes[0].set_title('Near Earth Objects')

# Add percentage labels to the left subplot
total = float(len(df.neo))
for p in axes[0].patches:
    percentage = '{:.1f}%'.format(100 * p.get_height()/total)
    x = p.get_x() + p.get_width() / 2 - 0.1
    y = p.get_y() + p.get_height() + 3
    axes[0].annotate(percentage, (x, y))

# Count plot of the "pha" variable on the right subplot
sns.countplot(x='pha', data=df, ax=axes[1])
axes[1].set_title('Potentially Hazardous Asteroids')

# Add percentage labels to the right subplot
total = float(len(df.pha))
for p in axes[1].patches:
    percentage = '{:.1f}%'.format(100 * p.get_height()/total)
    x = p.get_x() + p.get_width() / 2 - 0.1
    y = p.get_y() + p.get_height() + 3
    axes[1].annotate(percentage, (x, y))

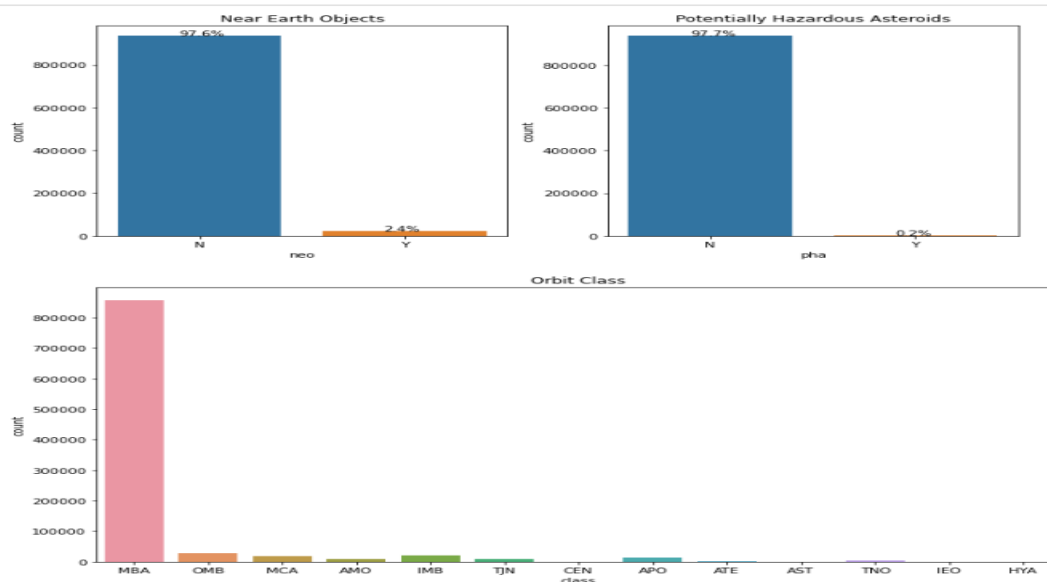
# Adjust the spacing between the subplots
fig.tight_layout()

# Show the plot
plt.show()

# Count plot for Orbit Class
fig, ax = plt.subplots(figsize=(12, 7))
sns.countplot(x='class', data=df, ax=ax)
plt.title('Orbit Class')

plt.show()
```

Visualized results of above analysis



From the above, we can understand that the data distribution is very irregular for each of the attributes chosen. The accuracy is highly manipulated since the data is biased to a particular value local to the corresponding attribute. To obtain efficient result analysis resampling the train data is essential. Later the test data can be used to evaluate each model's performance. The below code indicates the splitting of data into test and train data subsets.

```

In [101]: from sklearn.model_selection import train_test_split

# Split train, validation, and test sets
x = one_hot_encoded_data.drop('pha', axis=1)
y = one_hot_encoded_data['pha'].to_frame()
x_train, x_test, y_train, y_test = train_test_split(
    x, y, test_size=0.4, random_state=100, stratify=y)
x_val, x_test, y_val, y_test = train_test_split(
    x_test, y_test, test_size=0.5, random_state=100, stratify=y_test)
print("Shape of original dataset :", one_hot_encoded_data.shape)
print("Shape of x_train set", x_train.shape)
print("Shape of y_train set", y_train.shape)
print("Shape of x_validation set", x_val.shape)
print("Shape of y_validation set", y_val.shape)
print("Shape of x_test set", x_test.shape)
print("Shape of y_test set", y_test.shape)

Shape of original dataset : (932335, 46)
Shape of x_train set (559401, 45)
Shape of y_train set (559401, 1)
Shape of x_validation set (186467, 45)
Shape of y_validation set (186467, 1)
Shape of x_test set (186467, 45)
Shape of y_test set (186467, 1)

```

Under sampling and oversampling are techniques used in case of imbalanced classification of our dataset as the model tends to be biased towards the majority class and performs poorly on the minority class. Under sampling is used to reduce the number of instances in the majority class, whereas oversampling is used to increase the number of instances in the minority class. With the use of these, we can balance the number of instances in both classes and improve the performance of the model. This can lead to better accuracy, precision, recall, and F1 score for both classes.

```

In [102]: from sklearn.preprocessing import StandardScaler

# Normalizing the features
# Normalizing after splitting could prevent leaking information about the validation set into the train set
# StandardScaler() is useful in classification and Normalizer() is useful in regression
x_train = StandardScaler().fit_transform(x_train)
x_val = StandardScaler().fit_transform(x_val)
x_test = StandardScaler().fit_transform(x_test)

In [103]: # Imbalance in target variable
y_train.value_counts()

Out[103]: pha
0      558161
1       1240
dtype: int64

```

SMOTE (Synthetic Minority Over-sampling Technique) resampling is performed on the dataset to address the class imbalance problem. SMOTE resampling generates synthetic examples for the minority class by creating new instances that are combinations of existing minority class instances. This helps to balance the dataset and improve the performance of the machine learning model in predicting both classes. By oversampling the minority class in this way, the model is exposed to more diverse examples of that class and is less likely to miss important patterns in the data that are specific to that class.



```
In [104]: from imblearn.over_sampling import SMOTE
```

```
# Data Upsampling - SMOTE
x_train_us, y_train_us = SMOTE(
    sampling_strategy=0.5, random_state=100).fit_resample(x_train, y_train)
y_train_us.value_counts()
```

```
Out[104]: pha
0      558161
1      279080
dtype: int64
```

```
In [105]: from imblearn.under_sampling import RandomUnderSampler
```

```
# Data Undersampling - Random Undersampling
random_under_sampling = RandomUnderSampler(random_state=100)
x_train_us_rus, y_train_us_rus = random_under_sampling.fit_resample(x_train_us, y_train_us)

y_train_us_rus.value_counts()
```

```
Out[105]: pha
0      279080
1      279080
dtype: int64
```

From the below visualization, the data dimension distribution is uniform for chosen data attributes.

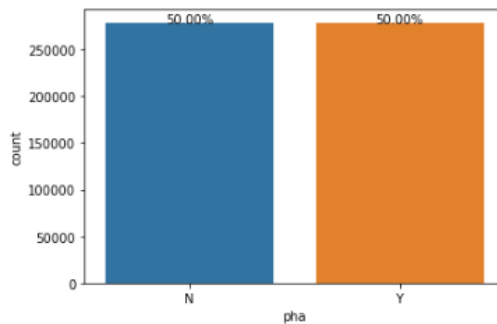
```
In [106]: from imblearn.over_sampling import SMOTE
```

```
# Data Upsampling - SMOTE
x_train_SMOTE, y_train_SMOTE = SMOTE(
    sampling_strategy=0.5, random_state=100).fit_resample(x_train, y_train)
y_train_SMOTE.value_counts()

# Data Undersampling - Random Undersampling
random_under_SAMPLING = RandomUnderSampler(random_state=100)
x_train_us_UNDER, y_train_us_UNDER = random_under_SAMPLING.fit_resample(x_train_SMOTE, y_train_SMOTE)

y_train_us_UNDER['pha'] = y_train_us_UNDER['pha'].map({0: 'N', 1: 'Y'})

ax = sns.countplot(x="pha", data=y_train_us_UNDER)
total = float(len(y_train_us_UNDER))
for p in ax.patches:
    height = p.get_height()
    ax.text(p.get_x()+p.get_width()/2., height + 3, '{:.2f}%'.format(100*height/total), ha="center")
plt.show()
```



## Execution of Machine Learning Models:

### Random Forest Classifier:

```
In [120]: from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import precision_score, recall_score, f1_score, classification_report

rfc = RandomForestClassifier(class_weight='balanced', random_state=100)
# Skip Hyperparameter Tuning part because parameter with default value get the highest accuracy of model

rfc.fit(x_train_us_rus, y_train_us_rus)

# Predict for validation set
y_val_pred = rfc.predict(x_val)

# Metrics
precision_rfc, recall_rfc, fscore_rfc, support_rfc = precision_recall_fscore_support(
    y_val, y_val_pred, average='macro')
# Calculate the accuracy of the model
accuracy_rfc = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy_rfc)

print(classification_report(y_val, y_val_pred))
```

```
Accuracy: 0.9980908149967554
```

	precision	recall	f1-score	support
0	1.00	1.00	1.00	186054
1	0.98	0.43	0.60	413
accuracy			1.00	186467
macro avg	0.99	0.71	0.80	186467
weighted avg	1.00	1.00	1.00	186467

### Logistic Regression:

```
In [121]: from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, precision_recall_fscore_support

from sklearn.linear_model import LogisticRegression
from sklearn.metrics import precision_score, recall_score, f1_score, classification_report
lr = LogisticRegression()

lr.fit(x_train_us_rus, y_train_us_rus)
# Predict for validation set
y_val_pred = lr.predict(x_val)

# Metrics
precision_lr, recall_lr, fscore_lr, support_lr = precision_recall_fscore_support(
    y_val, y_val_pred, average='macro')
# Calculate the accuracy of the model
accuracy_lr = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy_lr)
print(classification_report(y_val, y_val_pred))
```

```
Accuracy: 0.9980908149967554
```

	precision	recall	f1-score	support
0	1.00	0.99	1.00	186054
1	0.24	1.00	0.39	413
accuracy			0.99	186467
macro avg	0.62	1.00	0.69	186467
weighted avg	1.00	0.99	1.00	186467

## Decision Tree Classifier:

```
In [129]: M dtc=DecisionTreeClassifier(class_weight='balanced', random_state=100)

dtc.fit(x_train_us_rus, y_train_us_rus)
# Predict for validation set
y_val_pred=dtc.predict(x_val)

# Metrics
precision_dtc, recall_dtc, fscore_dtc, support_dtc = precision_recall_fscore_support(y_val, y_val_pred, average='macro')
print(classification_report(y_val, y_val_pred))

# Calculate the accuracy of the model
accuracy_dtc = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy_dtc)
```

	precision	recall	f1-score	support
0	1.00	1.00	1.00	186054
1	0.94	0.43	0.59	413
accuracy			1.00	186467
macro avg	0.97	0.72	0.80	186467
weighted avg	1.00	1.00	1.00	186467

Accuracy: 0.9980908149967554

## Naïve Bayes Classifier:

```
In [123]: M gnb=GaussianNB()

gnb.fit(x_train_us_rus, y_train_us_rus)
# Predict for validation set
y_val_pred=gnb.predict(x_val)

# Metrics
precision_gnb, recall_gnb, fscore_gnb, support_gnb=precision_recall_fscore_support(y_val, y_val_pred, average='macro')
print (classification_report(y_val, y_val_pred))

# Calculate the accuracy of the model
accuracy_gnb = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy_gnb)
```

	precision	recall	f1-score	support
0	1.00	1.00	1.00	186054
1	0.00	0.00	0.00	413
accuracy			1.00	186467
macro avg	0.50	0.50	0.50	186467
weighted avg	1.00	1.00	1.00	186467

Accuracy: 0.9980908149967554

## XGBoost Classifier:

```
In [124]: M from xgboost import XGBClassifier

xgbc = XGBClassifier(max_depth=10, learning_rate=0.1,
                     n_estimators=1000, eval_metric='mlogloss', random_state=100)

# Train the model on the training set
xgbc.fit(x_train, y_train)

# Make predictions on the testing set
y_pred = xgbc.predict(x_test)

# Calculate precision, recall, and f1 score
precision_xgbc = precision_score(y_test, y_pred)
recall_xgbc = recall_score(y_test, y_pred)
fscore_xgbc = f1_score(y_test, y_pred)

# Print precision, recall, and f1 score
print(f"Precision: {precision_xgbc:.2f}")
print(f"Recall: {recall_xgbc:.2f}")
print(f"F1 score: {fscore_xgbc:.2f}")

# Calculate the accuracy of the model
accuracy_xgbc = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy_xgbc)

# Print classification report
print(classification_report(y_test, y_pred))
```

Precision: 0.97  
Recall: 0.14  
F1 score: 0.25  
Accuracy: 0.9980908149967554

	precision	recall	f1-score	support
0	1.00	1.00	1.00	186054
1	0.97	0.14	0.25	413
accuracy			1.00	186467
macro avg	0.98	0.57	0.62	186467
weighted avg	1.00	1.00	1.00	186467

## K-Neighbors Classifier:

```
In [125]: knc=KNeighborsClassifier(n_neighbors=1)
knc.fit(x_train_us_rus, y_train_us_rus)
# Predict for test set
y_test_pred=knc.predict(x_test)

# Metrics
precision_knc, recall_knc, fscore_knc, support_knc=precision_recall_fscore_support(y_test, y_test_pred, average='macro')

# Calculate the accuracy of the model
accuracy_knc = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy_knc)

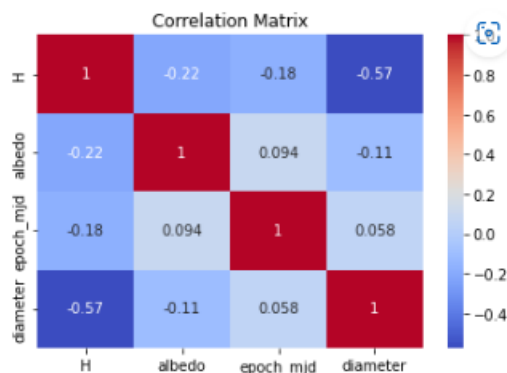
# Print the classification report
print(classification_report(y_test, y_pred))
```

Accuracy: 0.9980906149967554

	precision	recall	f1-score	support
0	1.00	1.00	1.00	186054
1	0.97	0.14	0.25	413
accuracy			1.00	186467
macro avg	0.98	0.57	0.62	186467
weighted avg	1.00	1.00	1.00	186467

## Diameter relation with other attributes:

```
# Plot the correlation matrix as a heatmap
sns.heatmap(corr, cmap='coolwarm', annot=True)
plt.title('Correlation Matrix')
plt.show()
```



## Asteroids in the dataset by orbit class is as follows:

```
In [5]: #The breakdown of the asteroids in the dataset by orbit class is as follows:
def summarize_by_class( df ):
    OrbitClass = {
        'AMO' : 'Amor',
        'APO' : 'Apollo',
        'AST' : 'Asteroid (other)',
        'ATE' : 'Aten',
        'CEN' : 'Centaur',
        'HYA' : 'Hyperbolic Asteroid',
        'IEO' : 'Atira',
        'IMB' : 'Inner Main-belt Asteroid',
        'MBA' : 'Main-belt Asteroid',
        'MCA' : 'Mars Crossing Asteroid',
        'OMB' : 'Outer Main-belt Asteroid',
        'TJN' : 'Jupiter Trojan',
        'TNO' : 'TransNeptunian Object'
    }

    _ = df['class'].value_counts()
    return pd.DataFrame({
        'class': _.index,
        'count': _,
        'orbit_class' : _.index.map( OrbitClass )
    }).reset_index(drop=True)

summarize_by_class(df)
```

Out[5]:

	class	count	orbit_class
0	MBA	855954	Main-belt Asteroid
1	OMB	28355	Outer Main-belt Asteroid
2	IMB	20360	Inner Main-belt Asteroid
3	MCA	18685	Mars Crossing Asteroid
4	APO	12687	Apollo
5	AMO	8457	Amor
6	TJN	8221	Jupiter Trojan
7	TNO	3468	TransNeptunian Object
8	ATE	1729	Aten
9	CEN	506	Centaur
10	AST	76	Asteroid (other)
11	IEO	22	Atira
12	HYA	4	Hyperbolic Asteroid

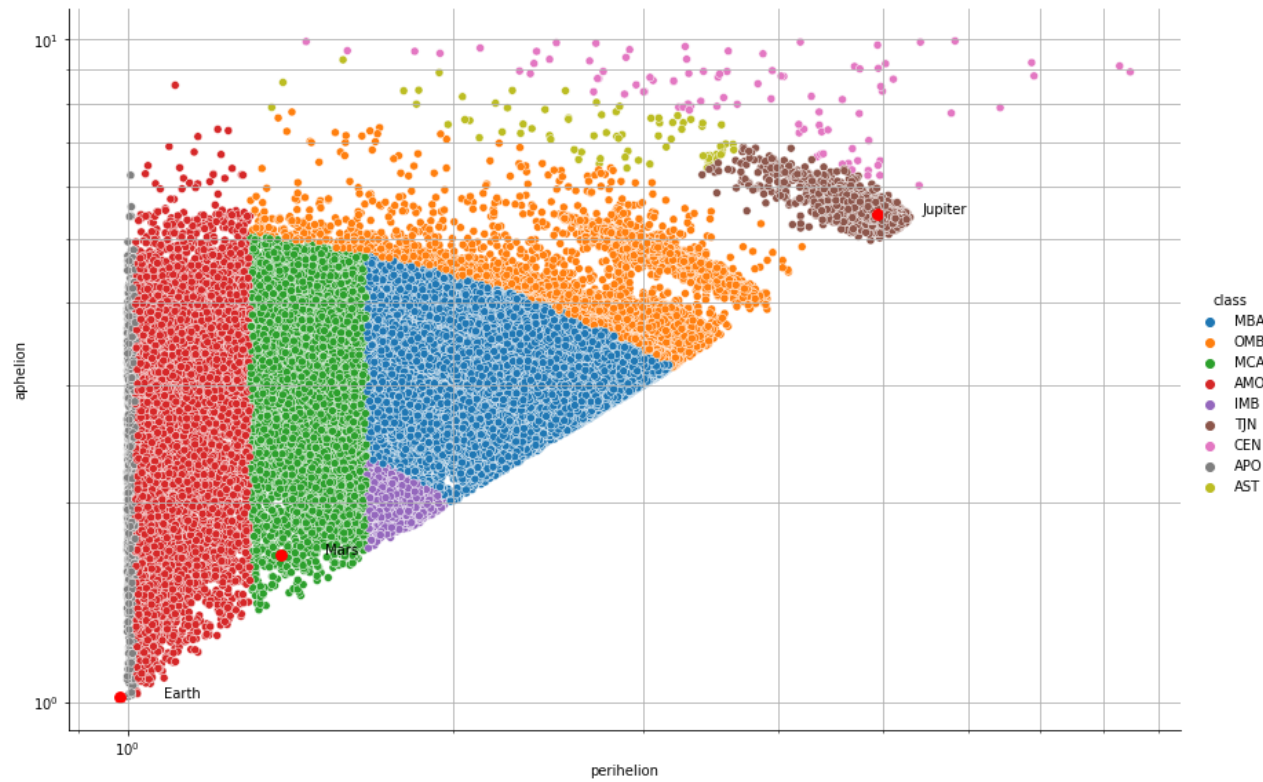
## Distribution Throughout the Solar System:

```
In [8]: ▶ def plot_planets(names=planets.keys(), params=(1,1,200)):
          ax = plt.gca()
          scale_x, scale_y, size = params
          for name in names:
              orbit = planets[name]
              perihelion = to_perihelion( orbit.a, orbit.e)
              aphelion = to_aphelion( orbit.a, orbit.e)
              plt.scatter( perihelion, aphelion, c='red', s=size)
              plt.text( scale_x*perihelion, scale_y*aphelion, name)

          sns.relplot( data=ellipticals, x='perihelion', y='aphelion', hue='class', height=8, aspect=12/8)
          plot_planets(params=(1.1,0.9,40))
          plt.minorticks_on()
          plt.grid(which='both')
          plt.xscale('log')
          plt.yscale('log')
```



## Asteroid classification near Earth

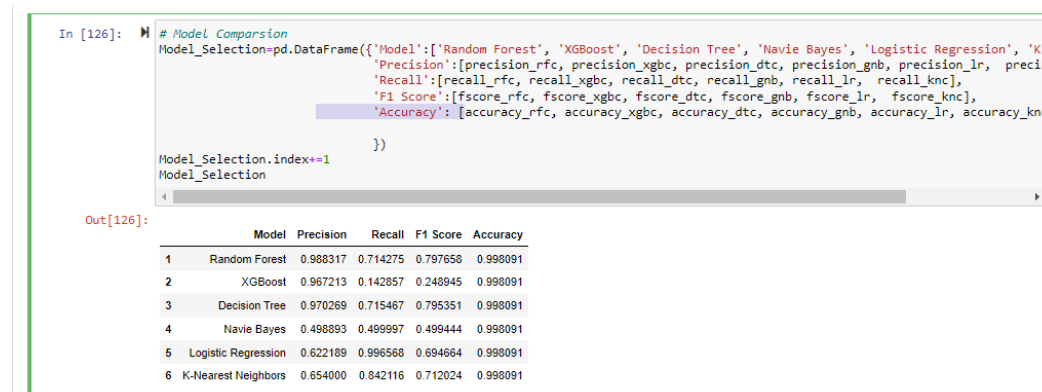


## EVALUATION AND OPTIMIZATION:

F1 score and accuracy metrics are used to evaluate the performance of classification models. F1 score is a harmonic mean of precision and recall, which means it considers both false positives and false negatives. On the other hand, accuracy measures the proportion of correctly classified instances among all instances.

While accuracy is a good metric for balanced datasets, it can be misleading when dealing with imbalanced datasets like ours, where one class is much more prevalent than the other. In such cases, a model that always predicts the majority class can still achieve high accuracy but fails to detect instances of the minority class.

Therefore, F1 score, and accuracy are combinedly used to determine model efficiency to get a more complete picture of a model's performance. A good model should have high accuracy and an F1 score, indicating that it is able to classify instances correctly while also considering false positives and false negatives. However, if a model performs well in terms of accuracy but poorly in terms of F1 score, it may indicate that the model is not good at detecting instances of the minority class, which is an important consideration in imbalanced datasets.



The screenshot shows a Jupyter Notebook cell with the following code and output:

```
In [126]: # Model Comparison
Model_Selection=pd.DataFrame({'Model':['Random Forest', 'XGBoost', 'Decision Tree', 'Navie Bayes', 'Logistic Regression', 'K-Nearest Neighbors'],
                              'Precision':[precision_rfc, precision_xgbc, precision_dtc, precision_gnb, precision_lr, precision_knc],
                              'Recall':[recall_rfc, recall_xgbc, recall_dtc, recall_gnb, recall_lr, recall_knc],
                              'F1 Score':[f1score_rfc, f1score_xgbc, f1score_dtc, f1score_gnb, f1score_lr, f1score_knc],
                              'Accuracy':[accuracy_rfc, accuracy_xgbc, accuracy_dtc, accuracy_gnb, accuracy_lr, accuracy_knc]})
Model_Selection.index+=1
Model_Selection
```

Out[126]:

	Model	Precision	Recall	F1 Score	Accuracy
1	Random Forest	0.988317	0.714275	0.797658	0.998091
2	XGBoost	0.967213	0.142857	0.248945	0.998091
3	Decision Tree	0.970269	0.715467	0.795351	0.998091
4	Navie Bayes	0.498893	0.499997	0.499444	0.998091
5	Logistic Regression	0.622189	0.996568	0.694664	0.998091
6	K-Nearest Neighbors	0.654000	0.842116	0.712024	0.998091

## RESULTS:

Based on Accuracy, all the models perform very well, with an almost similar accuracy score for all the models. So, accuracy alone is not sufficient to differentiate between the models. Based on F1 Score, the Random Forest model performs the best with an F1 score of 0.797658, followed closely by the Decision Tree model with an F1 score of 0.795351. The other models have lower F1 scores, with the XGBoost model having the lowest F1 score of 0.248945.

The diameter of the asteroid is closely related to the H, albedo, epoch\_mjd

The orbit class attribute i.e a three-character code is used to understand the distribution of asteroids in the solar system and the clear classification near the earth.

Firstly, full set of bound orbits we are interested in understanding the likelihood of planetary collisions. full set of bound orbits and later those whose perihelion lies inside of the Earth's orbit. Based on the eccentricity of the each of the asteroid a visualization is developed that clearly shows number of asteroids near the earth along with their classification based on their class value.

## **FUTURE WORK AND COMMENTS:**

### **1. What was unique about the data? Did you have to deal with imbalance? What data cleaning did you do? Outlier treatment? Imputation?**

The asteroid dataset contains comprehensive information about different aspects of asteroids such as their orbits, physical properties, and mineralogical composition. It also includes information about the asteroids' potential impact hazards and their likelihood of colliding with Earth. One unique aspect of this dataset is its class imbalance, meaning that there are significantly more examples of non-hazardous asteroids than hazardous ones. To deal with this imbalance, techniques such as under sampling and over sampling are used to balance the classes. Under sampling involves randomly removing examples from the majority class, while over sampling involves creating new examples in the minority class. As for data cleaning, the dataset contains null or missing values, which are dealt with through imputation or removal. Outliers are also present in some of the features, and these are treated using Z-score approach. Additionally, the categorical variables are encoded.

### **2. Did you create any new additional features / variables?**

Although no new variables were created, numerous new features were identified. Initially, AWS Athena and AWS Glue were used to locate the data in AWS Quicksight dashboards. Regression models, such as linear regression, were used to display the data results during the Analytics/ML modeling phase. AWS SageMaker was used to generate the final outcomes.

### **3. What was the process you used for evaluation? What was the best result?**

The process used for evaluation involved training the models on the training data, and then evaluating their performance on the validation set. The best model was then selected based on its performance on the validation set, and its performance was further tested on the test set to ensure that the model was not overfitting. The evaluation metrics used were precision, recall, F1 score, and accuracy. Precision is the ratio of true positives to the total number of positive predictions and measures the model's ability to correctly identify positive cases. Recall is the ratio of true positives to the total number of actual positive cases and measures the model's ability to correctly identify all positive cases. F1 score is the harmonic mean of precision and recall and provides a balanced measure of the model's performance. Accuracy is the proportion of correct predictions to the total number of predictions and measures the overall performance of the model.

The best result was achieved by the Random Forest model, which had an accuracy of 0.998 and an F1 score of 0.797. The model had a precision of 0.988 and a recall of 0.714, indicating that it was able to correctly identify a large proportion of positive cases while minimizing false positives.

### **4. What were the problems you faced? How did you solve them?**

The bias in the asteroid dataset is particularly due to the presence of imbalanced attributes like "pha" (potential hazardous asteroid), "neo" (near-earth object), and "H" (absolute magnitude



parameter used to estimate the size of the asteroid). As the dataset is biased, it could lead to inaccurate or unfair predictions.

To address the issue of bias, several approaches are taken. First, the dataset was thoroughly analyzed to identify any biases that exist. This involved examining the distribution of the imbalanced attributes and looking for patterns or correlations with other variables. The bias is detected, and resampling, data augmentation. SMOTE resampling technique has been used to uniformly distribute the dimensions of the data attributes. In addition, the bias is mitigated through careful feature selection and converting the data type of attribute to uniform data representation. Addressing bias in the asteroid dataset is crucial to ensure accurate and fair predictions, and it required a combination of careful analysis, appropriate algorithm selection, and appropriate preprocessing techniques.

## **5. What future work would you like to do?**

One potential avenue is to explore more advanced machine learning algorithms, such as neural networks, to see if they can improve the accuracy of the classification results. Another potential area of investigation is to incorporate additional features, such as the size or shape of the asteroids, to see if they can improve the model's predictive power. Finally, exploring the use of ensemble methods, which combine the predictions of multiple models, to further improve the accuracy might be beneficial.

## **6. Instructions for individuals that may want to use your work**

To reproduce our findings, it is important for interested parties to install necessary Python libraries, including NumPy, Pandas, Matplotlib, Sns, and Sklearn, which are extensively used in this project. Additionally, the dataset should undergo thorough data cleansing, transformation, and preparation to remove most of the outliers, ensuring the accuracy of machine learning models and visualizations. The dataset is sourced from Kaggle, and those who wish to use our work must set up an appropriate AWS environment with the necessary credentials. AWS services may incur charges that need to be paid. After loading the Kaggle data into AWS, the user must configure AWS SageMaker to run the Python notebook containing the code. The GitHub repository contains all other requirements for the procedure.