NASA PLANETOID STRATIFICATION USING MACHINE LEARNING

A Project Report submitted to

Rajiv Gandhi University of Knowledge and Technologies

SRIKAKULAM

In partial fulfillment of the requirements for the Award of the degree of

BACHELOR OF TECHNOLOGY IN COMPUTER SCIENCE AND ENGINEERING

Submitted By

4th Year B. Tech 2nd semester

S.Sravani (s170062)

SK.Grace Jessy Moon(s171159)

N.Prasanna(s170191)

Under the Esteemed Guidance of

Asst. Prof. S.SATEESH KUMAR SIR

CERTIFICATE

This is to certify that the report entitled "**Nasa Planetoid Stratification Using Machine Learning**" Submitted by **S.Sravani(s170062), SK.Grace Jessy Moon(s171159), N.Prasanna(s170191)** in partial fulfillment of the requirements for the award of Bachelor of Technology in Computer Science is a bonafide work carried out by them under my supervision and guidance.

The report has not been submitted previously in part or full to this or any other University or Institution for the award of any Degree or Diploma.

Asst. Prof. S.SATEESH KUMAR SIR
Project Guide

Asst. Prof. CH.SATEESH KUMAR SIR
Project Coordinator

DECLARATION

We, S.Sravani, SK.Grace Jessy Moon, N.Prasanna hereby declare that this report entitled **"Nasa Planetoid Stratification Using Machine Learning"** submitted by us under the guidance and supervision of **S.Sateesh Kumar** is a bonafide work. Carried out by us during the year 2022-2023 in partial fulfillment of the requirements for the Major Project in Computer Science and Engineering. We further declare that this dissertation has not been submitted elsewhere for any Degree. The matter embodied in this dissertation report has not been submitted elsewhere for any other Degree. Furthermore, the technical details furnished in various chapters of this thesis are purely relevant to the above project and there is no deviation from the theoretical point of view for design, development and implementation.

Date :

Place :

S.Sravani (s170062)

SK.Grace Jessy Moon (s1701159)

N.Prasanna (s170191)

ACKNOWLEDGEMENT

We would like to articulate my profound gratitude and indebtedness to our project guide **Mr. S.Sateesh kumar sir,** Assistant Professor who has always been a constant motivation and guiding factor throughout the project name. It has been a great pleasure for me to get an opportunity to work under his guidance and complete the thesis work successfully. We extend our sincere thanks to Asst. Prof. **M. Roopa Mam**, M. Tech, Head of the Computer Science and Engineering Department, for her constant encouragement throughout the project. We are also grateful to other members of the department without their support our work would have been carried out so successfully. We thank one and all who have rendered help to us directly or indirectly in the completion of our thesis work.

Project Associate/ Project Members

S.Sravani (s170062)

SK.Grace Jessy Moon (s1701159)

N.Prasanna (s170191)

ABSTRACT

Asteroids are the leftovers from the formation of our solar system about 4.6 billion years ago. There exist millions of asteroids and the vast majority of known asteroids orbit within the central asteroid belt located between the orbits of mars and jupiter. With an asteroid hitting the earth, dust and smoke rising in the atmosphere prevents sunlight from reaching our world and causes the total temperature to drop. This event can lead to the death of many living organisms

The only way to eliminate the threat of the asteroid hitting the earth is to divert them from their course. Many organizations, primarily NASA, perform regular scans of the sky to identify celestial bodies at risk of hitting our earth. But before diverting the asteroid from its path it is much needed to find out whether it is precarious or not. In earlier times astronomers used to use ground-space-based telescopes to detect the threat from celestial bodies. As technology is evolving day by day we want to detect the precarity using machine learning algorithms like logistic regression, decision tree, XGBoost, Random forest.

This type of stratification helps in enhancing the efforts of NASA there by providing more accurate results that helps to prevent the planet from hazardous asteroids.

Keywords:

Asteroids, asteroid belt, precarious, ground-space-based-telescope, Machine learning, logistic regression, decision tree, XGBoosting.

INDEX

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 Introduction

Asteroids are one of the most discussed materials of space. Asteroids are leftovers from the formation of our solar system about 4.6 billion years ago. There exist millions of asteroids and the vast majority of known asteroids orbit within the central asteroid belt located between the orbits of mars and jupiter. Generally asteroids are amorphous means they don't have specific shape and size. Most of the asteroids are located between the planets Mars and Jupiter called the asteroid belt. Asteroids are located in the asteroid belt because the gravity of Jupiter might keep a planet from forming in the area. Asteroids are one of the most important materials to find the characteristics of the solar systems. So it is very valuable to scientists. It is divided into various classes based on size, color & composition and their location.based on color & composition, the asteroids are different classes known as C-type(Chondrite), S-type(Stony), M-type(metallic) and based on location also asteroids are classified into different classes known as Main belt asteroids, trojan asteroids and Near Earth asteroids. But all are not hazardous, only the asteroids moving near the earth are potentially hazardous. They are of four types. But all are not hazardous also. Those are known as Atens, Atiras, Amors, Apollos. They are divided by various features such as semi major axis, eccentricity etc. So the hazardous and non-hazardous quality is also depending on some attributes such as its diameter, relative velocity, magnitude, inclination etc. It has been seen that now-a-days, machine learning is the one of the most important techniques for predicting or classifying the dataset. So here, I use some of the machine learning models and later compare those results to show which one of them is the more accurate for classifying the asteroids as hazardous or non-hazardous.

1.2 Problem statement

The data is about Asteroids-NeoWs(Near Earth Object Web Service) is a RESTful(API) web service for near earth Asteroid information.With NeoWs a user can : Search for Asteroids based on their closest approach date to Earth, Lookup a specific Asteroid with its NASA JPL small body id,as well as browse the overall data-set.

1.3 Scope of the project

Hazardous asteroids need to divert their path. Before diverting the asteroid from its path it is much needed to find out whether it is precarious or not. In earlier times astronomers used to use ground-space-based telescopes to detect the threat from celestial bodies. As technology is evolving day by day we want to detect the precarity using machine learning algorithms. This type of stratification helps in enhancing the efforts of NASA there by providing more accurate results that helps to prevent the planet from hazardous asteroids

1.4 Definitions & Acronyms

- **NumPy:** NumPy is a library for the Python programming language. It provides support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to perform operations on these arrays.
- **Pandas:** Pandas is an open-source library that is made mainly for working with relational or labeled data both easily and intuitively. It provides various data structures and operations for manipulating numerical data and time series. This library is built on top of the NumPy library. Pandas is fast and it has high performance & productivity for users.
- **Sk learn:** Scikit-learn (Sklearn) is the most useful and robust library for machine learning in Python. It provides a selection of efficient tools for machine learning and statistical modeling including classification, regression, clustering and dimensionality reduction via a consistent interface in Python.
- **Keras:** Keras is an open-source high-level Neural Network library, which is written in Python is capable enough to run on Theano, TensorFlow, or CNTK. It was developed by one of the Google engineers, Francois Chollet. It is made user-friendly, extensible, and modular for facilitating faster experimentation with deep neural networks. It not only supports Convolutional Networks and Recurrent Networks individually but also their combination.
- **Missingno:** Missingno is a Python library that provides the ability to understand the distribution of missing values through informative visualizations. The visualizations can be in the form of heat maps or bar charts. With this library, it is possible to observe where the missing values have occurred and to check the correlation of the columns containing

the missing with the target column. Missing values are better handled once the dataset is fully explored. Let us now implement this and find out how it helps us pre-process the data better.

CHAPTER 2

LITERATURE SURVEY

2.1 Collecting the information:

Planetoids are leftovers from the formation of our solar system about 4.6 billion years ago. Generally Planetoids are amorphous. Most asteroids are located in an area between the orbits of Mars and Jupiter called the Asteroid Belt. Asteroids help in finding the characteristics of the solar system. There are many approaches has presented by many researchers.The former approaches which were doing using Logistic Regression Algorithm are not exactly accurate and that's the reason why astronomers are still using ground-space-based telescopes to detect the threat.so,to increase the accuracy we want to train a model with a collection of machine learning algorithms like Decision Tree,Random Forest,XGBoosting etc.

In this process we want to use the dataset collected by NEOWS(Near Earth Object Web Service) consisting of 4687 data instances(rows) and 40(columns) which gives the detailed nature of every asteroid identified by the space research organizations.

The main features include:

1.Neo Reference ID': This feature denotes the reference ID assigned to an asteroid.

2.'Name': This feature denotes the name given to an asteroid.

3.'Absolute Magnitude': This feature denotes the absolute magnitude of an asteroid. An asteroid's absolute magnitude is the visual magnitude an observer would record if the asteroid were placed 1 Astronomical Unit (AU) away, and 1 AU from the Sun and at a zero phase angle.

4.'Est Dia in KM(min)': This feature denotes the estimated diameter of the asteroid in kilometers (KM).

5.'Est Dia in M(min)': This feature denotes the estimated diameter of the asteroid in meters(M).

6.'Relative Velocity km per sec': This feature denotes the relative velocity of the asteroid in kilometers per second.

7.Relative Velocity km per hr': This feature denotes the relative velocity of the asteroid in kilometers per hour.

8.'Orbiting Body': This feature denotes the planet around which the asteroid is revolving.

9.'Jupiter Tisserand Invariant': This feature denotes the Tisserand's parameter for the asteroid. Tisserand's parameter (or Tisserand's invariant) is a value calculated from several orbital elements(semi-major axis, orbital eccentricity, and inclination) of a relatively small object and a more substantial' perturbing body'. It is used to distinguish different kinds of orbits.

10.‘Eccentricity’: This feature denotes the value of eccentricity of the asteroid’s orbit. Just like many other bodies in the solar system, the realms made by asteroids are not perfect circles, but ellipses. The axis marked eccentricity is a measure of how far from circular each orbit is: the smaller the eccentricity number, the more circular the realm.
11.‘Semi Major Axis’: This feature denotes the value of the Semi Major Axis of the asteroid’s orbit. As discussed above, the realm of an asteroid is elliptical rather than circular. Hence, the Semi Major Axis exists.
12.‘Orbital Period’: This feature denotes the value of the orbital period of the asteroid. Orbital period refers to the time taken by the asteroid to make one full revolution around its orbiting body.
13.‘Perihelion Distance’: This feature denotes the value of the Perihelion distance of the asteroid. For a body orbiting the Sun, the point of least distance is the perihelion.
14.‘Aphelion Dist’: This feature denotes the value of Aphelion distance of the asteroid. For a body orbiting the Sun, the point of greatest distance is the aphelion.
15.‘Hazardous’: This feature denotes whether the asteroid is hazardous or not.

2.2 Study

After the detailed study it has been noticed that the time taken for threat detection was a major issue in previous studies and because of that lead time decreases,a threat to earth increases.Hence there is a need for improvement in the speed of the detection and more focus on reliability of the model.
In this paper[1],the authors tried to make some predictions over the combinations of orbital parameters for yet undiscovered and potentially hazardous asteroids by machine learning techniques. For this reason, they have used the Support Vector Machine algorithm with the kernel RBF. By this approach, the boundaries of the potentially hazardous asteroid groups in 2-D and 3-D can be easily understood. By this algorithm, they have achieved an accuracy of over 90%.

In this paper[2] , authors have provided classification of asteroids, which are observed by VISTA-VHS survey. They have used some statistical methods to classify the 18,265 asteroids. They used a probabilistic method, KNN method and a statistical method to classify those. Later, they compared the algorithms’ accuracy. They have classified the asteroids into V, S and A types. They did it on the 18265 asteroids and test set consists of over 6400 asteroids.

CHAPTER 3

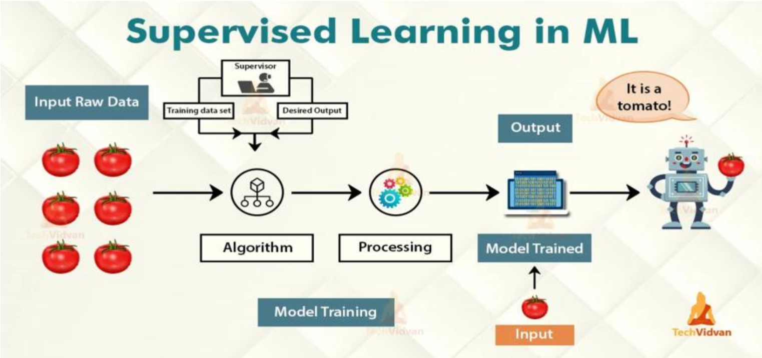
TECHNOLOGY OVERVIEW

3.1 Machine Learning

Machine Learning is a subfield of Artificial Intelligence, which is broadly defined as the capability of machines to imitate intelligent human behavior. Artificial Intelligence systems are used to perform complex tasks in a way that is similar to how humans solve problems. The process of learning begins with the observations or data, such as examples, direct experience, or instruction, in order to look for patterns in the data and make better decisions in the future based on the examples that we provide. The primary aim is to allow the computers to allow automatically without human intervention or assistance and adjust actions accordingly. In machine Learning the text is considered as a sequence of keywords; instead, an approach based on semantic analysis mimics the human ability to understand the meaning of a text.

<u>Supervised Learning</u>

It is a type of machine learning method in which we provide sample labeled data to the machine learning system in order to train it., and on that basis, it predicts the output. The system provides a model using labeled data to understand the datasets and learn about each data, once the training and processing are done then we test the model by providing sample data to check whether it is providing the exact output or not. Supervised Learning can be grouped further into two categories of algorithms: Classification and Regression



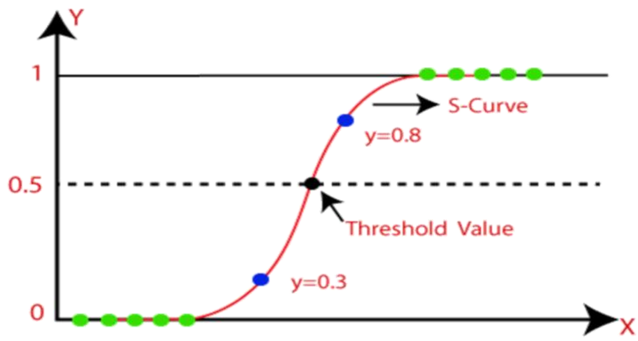
Fig[1] : Supervised learning algorithm

Models in Supervised Learning are:

1.Logistic Regression:

Logistic regression is a supervised learning algorithm used to calculate or predict the probability of binary event occurring. Here instead of fitting a regression line, we fit an “S” shaped logistic function, which predicts maximum values of 1 or 0. The S shaped curve is known as sigmoid function. In this logistic regression we use the concept of threshold value, which defines the probability either 0 or 1.

from sklearm.linear_model import LogisticRegression



Fig[2] : Logistic regression

2.Gaussian Naive Bayes algorithm:

Naïve Bayes theorem is also a supervised algorithm, which is based on the Bayes theorem and used to solve classification problems. It is one of the most simple and effective classification algorithms in Machine Learning which enables us to build various ML models for quick predictions. It is a probabilistic classifier that means it predicts on the basis of probability of an object.

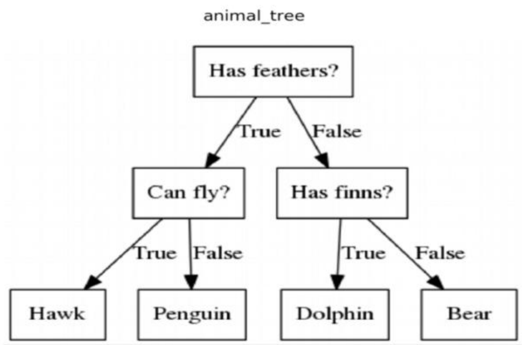
$$P(X|Y=c) = \frac{1}{\sqrt{2\pi\sigma_c^2}} e^{\frac{-(x-\mu_c)^2}{2\sigma_c^2}}$$

3.Decision Tree:

Decision tree algorithms fall under the category of supervised learning. They can be used to solve both regression and classification problems.

Decision tree uses the tree representation to solve the problem in which each leaf node corresponds to a class label and attributes are represented on the internal node of the tree.

from sklearn.tree import DecisionTreeClassifier as DTC

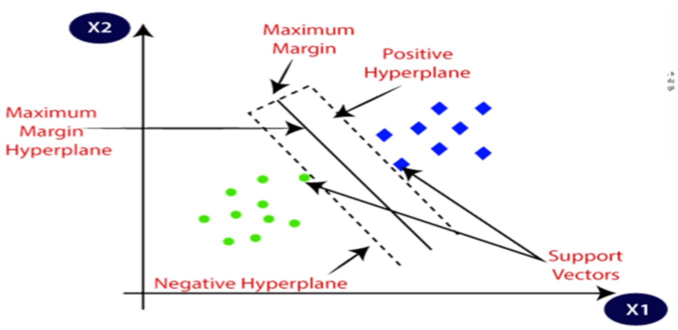


Fig[3] : Decision tree example

4. Support Vector Machine:

Support Vector Machine or SVM is one of the most popular Supervised Learning algorithms, which is used for Classification as well as Regression problems. However, primarily, it is used for Classification problems in Machine Learning. The goal of the SVM algorithm is to create the best line or decision boundary that can segregate n-dimensional space into classes so that we can easily put the new data point in the correct category in the future.This best decision boundary is called a hyperplane.

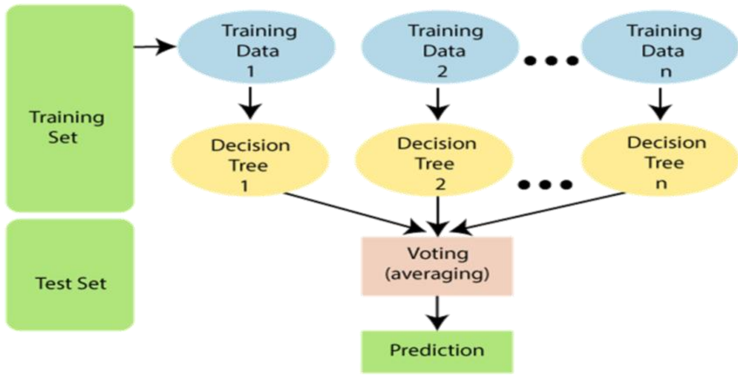
from sklearn.svm import SVC # "Support vector classifier



Fig[4] : Support vector Machine

5. Random Forest:

Random Forest is a popular machine learning algorithm that belongs to the supervised learning technique. It can be used for both Classification and Regression problems in ML. It is based on the concept of ensemble learning, which is a process of combining multiple classifiers to solve a complex problem and to improve the performance of the model. "Random Forest is a classifier that contains a number of decision trees on various subsets of the given dataset and takes the average to improve the predictive accuracy of that dataset." The greater number of trees in the forest leads to higher accuracy and prevents the problem of overfitting.

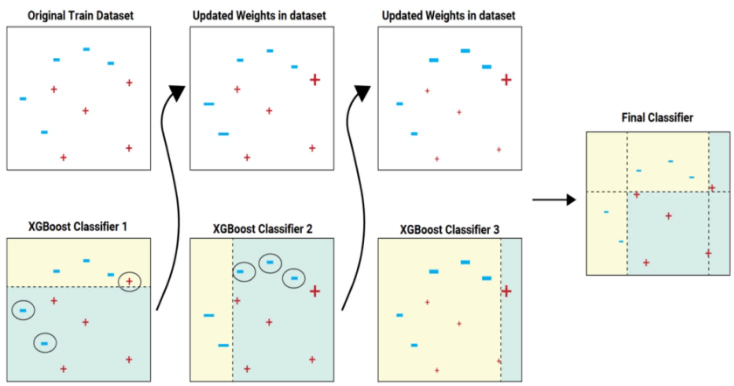


Fig[5] : random forest example

6.XGBoost:

XGBoost builds one tree at a time so that each data pertaining to the decision tree is taken into account and the data is filled in if there is any missing data. This helps developers to work with gradient algorithms along with the decision tree algorithm for better results. If we want to explore more about decision trees and gradients, XGBoost is a good option.

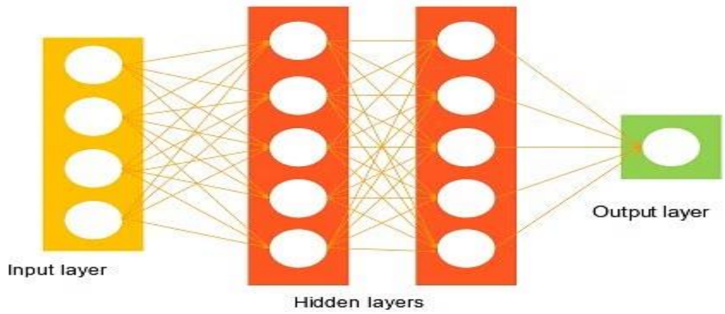
from xgboost import XGBClassifier



Fig[6] : XGBoosting classifier

7. Neural Networks:

Neural networks are a set of algorithms, modeled loosely after the human brain, that are designed to recognize patterns. They interpret sensory data through a kind of machine perception, labeling or clustering raw input. The patterns they recognize are numerical, contained in vectors, into which all real-world data, be it images, sound, text or time series, must be translated.ANN replicates a biological neuron. Input to a neuron - input layer, Neuron - hidden layer, Output to the next neuron - output layer



Fig[7] : Artificial neural network

Applications of Machine Learning:

- Agriculture
- Banking
- Computer vision
- Credit- card fraud detection
- Economics
- Financial market analysis
- Handwriting recognition

Correlation Matrix:

A correlation matrix is simply a table which displays the correlation coefficients for different variables. The matrix depicts the correlation between all the possible pairs of values in a table. It is a powerful tool to summarize a large dataset and to identify and visualize patterns in the given data. Correlation coefficient describes how strong the variables are related to each other. The formula for finding the correlation coefficient between the variables are:

$$Cor(X,Y) = \frac{Cov(X,Y)}{s_x s_y}$$

Fig[8] : correlation coefficient formula

Sx, Sy are the standard deviations of variables x and y respectively.

3.2 Evaluation metrics:

Evaluation metrics are used to evaluate the performance of the model that was trained. The most commonly used evaluation metrics are:

1. Confusion Matrix:

It is a matrix used for evaluating the performance of a classification model.

n = total predictions	Actual: No	Actual: Yes
Predicted: No	True Negative	False Positive
Predicted: Yes	False Negative	True Positive

Fig[9] : confusion matrix

True Negative: these are the cases when the actual class of the datapoint was false and predicted is also false.

True Positive: these are the cases when the actual class of the datapoint was true and predicted is also true.

False Positive: these are the cases when the actual class of the datapoint was false and predicted is true.

False Negative:these are the cases when the actual class of the datapoint was true and predicted is false.

2. Accuracy:

Accuracy is one metric for evaluating classification models. Informally, accuracy is the fraction of predictions our model got right. Formally, accuracy has the following definition:

$$Accuracy = \frac{True_{positive} + True_{negative}}{True_{positive} + True_{negative} + False_{positive} + False_{negative}}$$

Fig[10] : Accuracy formula

3. Precision:

Precision is the ratio of no.of True Positives to the total number of predicted positives. It measures, out of the total predicted positive, how many are actually positive.

$$Precision = \frac{True_{positive}}{True_{positive} + False_{positive}}$$

Fig[11] : precision formula

4. Recall:

Recall is the ratio of no.of True positives to the total number of actual positives. It measures out of the total actual positives, how many are predicted as True positives.

$$\boldsymbol{Recall} = \frac{\boldsymbol{True_{positive}}}{\boldsymbol{True_{positive}} + \boldsymbol{False_{negative}}}$$

Fig[12] : recall formula

5. F1 Score:

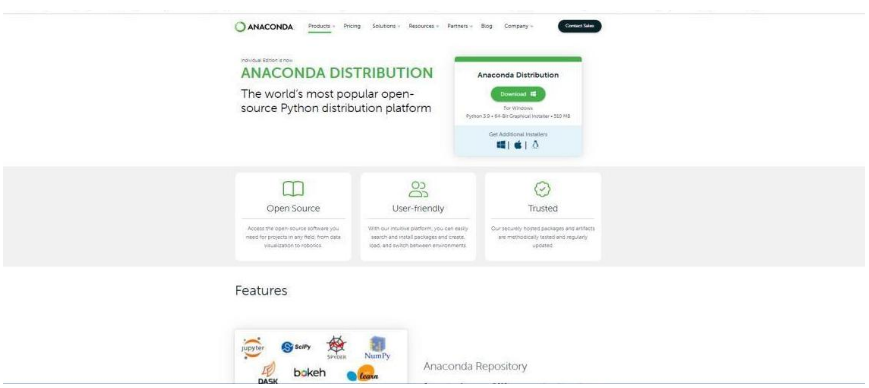
F1 score is a metric that measures a model’s accuracy. It combines the precision and recall scores of a model. The accuracy metric computes how many times a model made a correct prediction across the entire dataset.

$$\begin{aligned}\text{F1 Score} &= \frac{2}{\frac{1}{\text{Precision}} + \frac{1}{\text{Recall}}} \\ &= \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}\end{aligned}$$

Fig[13] : F1 score formula

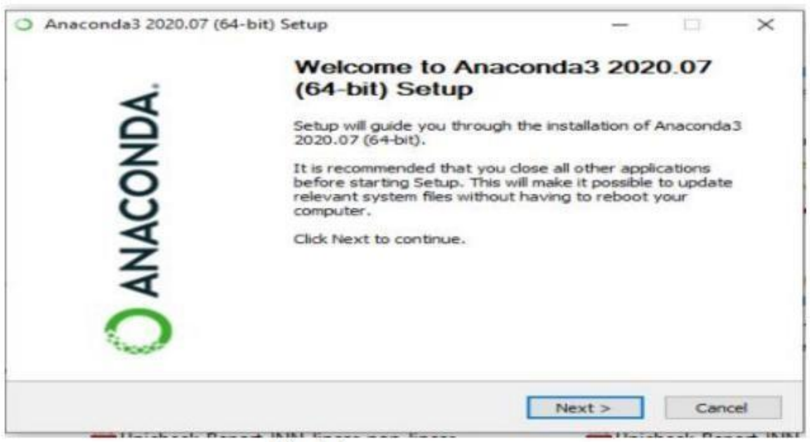
3.3 Installing Anaconda through chrome

1 .Install Anaconda



Fig[14] : Step 1

2. Download .exe File.



Fig[15] : Step 2

3. After installation click on Launch Jupyter Notebook

Fig[16] : Step 3

CHAPTER 4

SYSTEM ANALYSIS

4.1 Existing System

NASA, performs regular scans of the sky to identify celestial bodies at risk of hitting our earth. But before the diverting the asteroid from its path it is much needed to find out whether it is precarious or not

4.2 Proposed System

we want to detect the precarity using machine learning algorithms like Logistic regression, Decision tree, XGBoosting, Random forest,Neural Networks.

This type of stratification helps in enhancing the efforts of NASA there by providing more accurate results that helps to prevent the planet from hazardous asteroids.

4.3 System Requirements

4.3.1 Hardware Components

Processor	:	I3/Intel Processor/AMD Processor
Hard disk	:	10GB
RAM	:	2GB(Minimum)
Keyboard	:	Standard Windows Keyboard
Mouse	:	Two or Three Button

4.3.2 Software Components

Operating System	:	Windows 7, 8, 9, 10 and 11
IDE	:	Anaconda
Language	:	python
Libraries	:	Numpy, Panda, sk learn, Keras, missingno, warnings,

CHAPTER 5

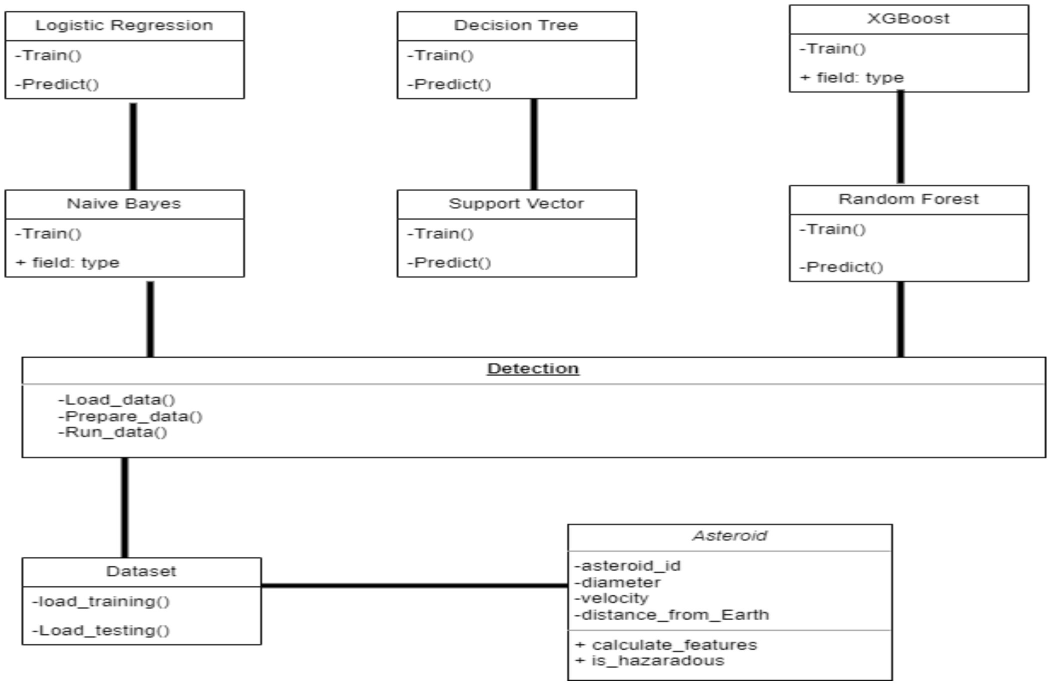
SYSTEM DESIGN

5.1 Design of the system

Unified Modelling Language (UML) was created in 1995 by using merging diagramming conventions used by three application development methodologies: OMT by James Rumbaugh, Objector by Invar Jacobson and the Brooch procedure by using Grady Brooch. Previous to this time, these three amigos, together with a few dozen other practitioners had promoted competing methodologies for systematic program development, each and every with its possess system of diagramming conventions. The methodologies adopted a sort of cookbook sort of pushing an application task via a succession of life cycle stages, culminating with a delivered and documented software. One purpose of UML was once to slash the proliferation of diagramming techniques by way of standardizing on an original modelling language, as a result facilitating verbal exchange between builders. It performed that goal in 1997 when the (international) Object administration team (OMG)adopted it as a commonplace. Some critics don't forget that UML is a bloated diagramming language written by means of a committee. That said, I do not forget it to be the nice manner to be had today for documenting object-oriented program progress. It has been and is fitting more and more utilized in industry and academia. Rational Rose is a pc Aided program Engineering (CASE)software developed by way of the Rational organization underneath the course of Brooch, Jacobson and Rumbaugh to support application progress using UML. Rational Rose is always complex due to its mission of wholly supporting UML. Furthermore, Rational Rose has countless language extensions to Ada, C++, VB, Java, J2EE, and many others. Rational Rose supports ahead and reverse engineering to and from these langue ages. However, Rational Rose does now not aid some usual design tactics as knowledge drift diagrams and CRC cards, due to the fact that these will not be a part of UML. Considering that Rational Rose has so many capabilities it is a daunting task to master it. Happily, loads can be executed making use of only a small subset of these capabilities. These notes are designed to introduce beginner builders into making productive use of the sort of subset.

5.1.1 Class Diagram:

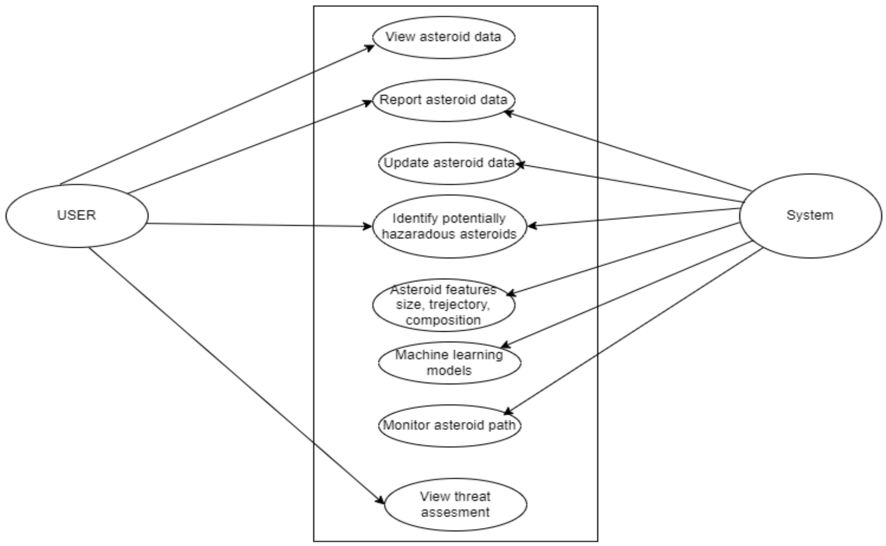
Class diagram in the Unified Modelling Language (UML), is a kind of static structure diagram hat describes the constitution of a process through showing the system's classes, their attributes, and the relationships between the class. The motive of a class diagram is to depict the classes.



Fig[17] : class diagram

5.1.2 Use Case Diagram:

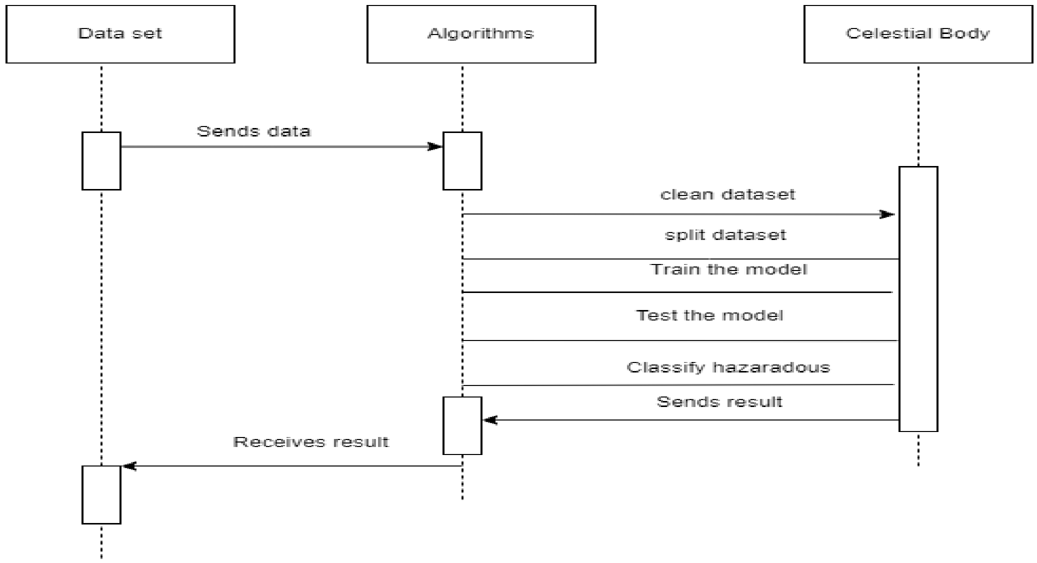
It is a visual representation of what happens when an actor interacts with a system. A use case diagram captures the functional aspects of a system. The system is shown as a rectangle with the name of the system inside ,the actors are shown as stick figures, the use cases are shown as solid bordered ovals labeled with the name of the use case and relationships are lines or arrows between actor and use cases.



Fig[18] : use case diagram

5.1.3 Sequence Diagram:

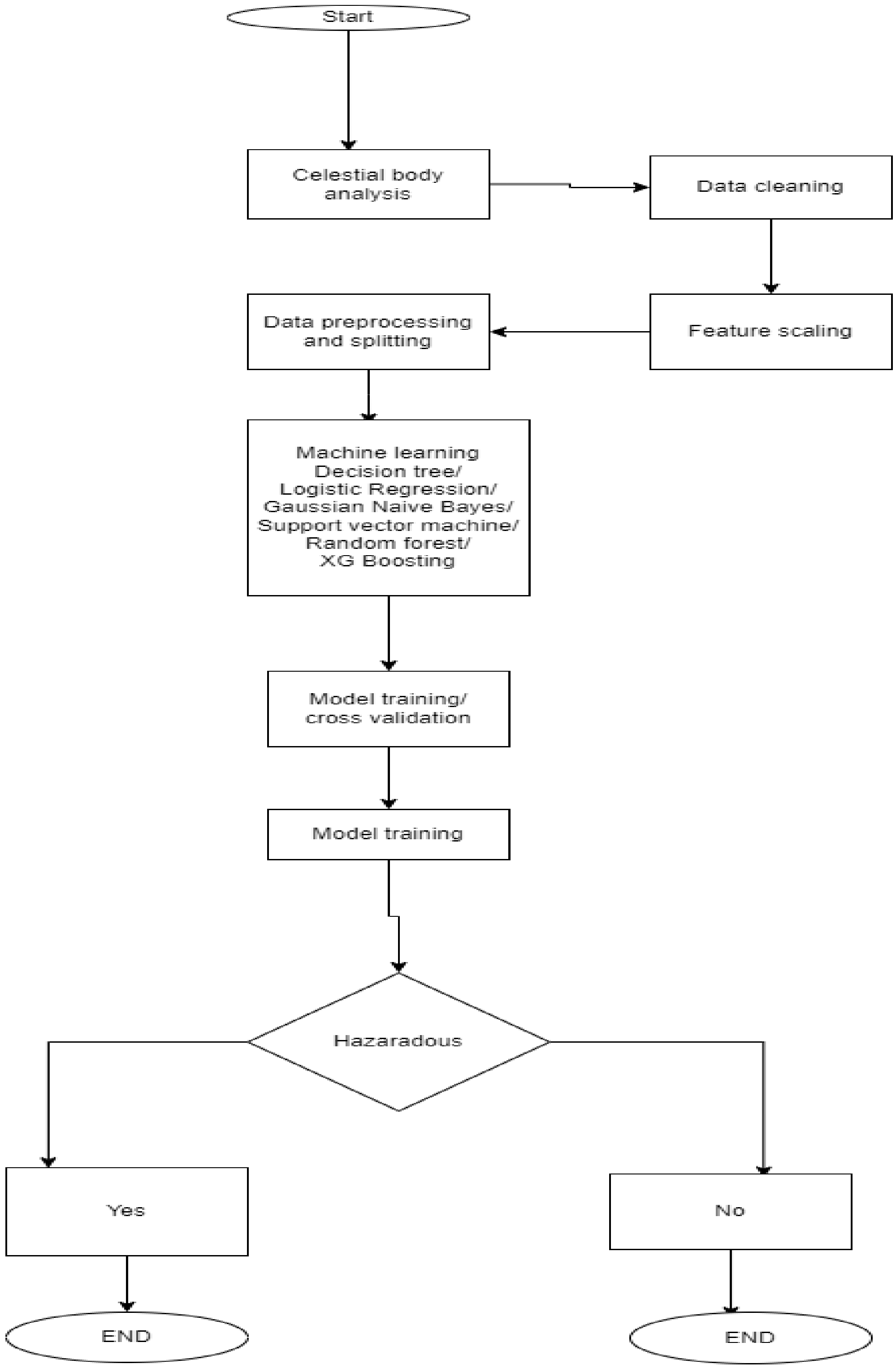
A sequence diagram in Unified Modelling Language (UML) is one variety of interaction diagram that suggests how methods operate with one another and in what order. It is a construct of a Message Sequence Chart. Sequence diagrams are quite often referred to as event-hint diagrams, event situations, and timing diagrams. A sequence diagram suggests, as parallel vertical traces (lifelines), special systems or objects that are residing at the same time, and, as horizontal arrows, the messages exchanged between them, within the order the place they occur.



Fig[19] : sequence diagram

5.1.4 DFD Diagrams:

A data flow diagram or bubble chart (DFD) is a graphical representation of the "flow" of data through an information system, modeling its process aspects. Often they are a preliminary step used to create an overview of the system which can later be elaborated. DFD's can also be used for the visualization of data processing (structured design). A DFD shows what kinds of information will be input to and output from the system, where the data will come from and go to, and where the data will be stored. It does not show information about the timing of processes, or information about whether processes will operate in sequence or in parallel (which is shown on a flowchart).The primitive symbols used for constructing DFD's are: Symbols used in DFD.



Fig[20] : DFD diagram

CHAPTER 6

WORKING & DESIGN CONCEPT

To implement Planetoid Stratification it involves two steps. First we have to train the model and test the model with data. For training the model we need to train the model by a dataset we have collected from the kaggle and the dataset is called NASA.CSV which contains the overall information about the asteroid and its features. We need to import all the Necessary packages such as numpy,pandas, matplotlib, seaborn, keras, warnings, missingno, sklearn. We have to pass all the parameters to model through the nasa.csv file. The nasa.csv will be given as input to the model. As the nasa.csv file consists of many unnecessary features, we will preprocess the data and perform data cleaning on the dataset. After we have started the main implementation of the model there we have trained our model with various machine learning algorithms i.eLogistic Regression, Decision trees, Random Forests, Support vector machine, XGBoosting, Gaussian Naive Bayes, Neural Networks. After the completion of the training, we have splitted the dataset into training dataset and testing dataset.

CHAPTER 7

SOURCE CODE

import packages

```
import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns
```

1.Gaussian Naive Bayes algorithm:

```
[ ] import pandas as pd
    import numpy as np
    import matplotlib.pyplot as plt
    import seaborn as sns
```

```
df = pd.read_csv('C:\\Users\\kanak\\OneDrive\\Desktop\\mpr\\nasa.csv')
```

```
[ ] df.head()
```

	Neo Reference ID	Name	Absolute Magnitude	Est Dia in KM(min)	Est Dia in KM(max)	Est Dia in M(min)	Est Dia in M(max)	Est Dia in Miles(min)	Est Dia in Miles(max)	Est Dia in Feet(min)	...	Asc Node Longitude	Orbital Period	Perihelion Distance	Perihelion Arg
0	3703080	3703080	21.6	0.127220	0.284472	127.219879	284.472297	0.079051	0.176763	417.388066	...	314.373913	609.599786	0.808259	57.257470
1	3723955	3723955	21.3	0.146068	0.326618	146.067964	326.617897	0.090762	0.202951	479.225620	...	136.717242	425.869294	0.718200	313.091975
2	2446862	2446862	20.3	0.231502	0.517654	231.502122	517.654482	0.143849	0.321655	759.521423	...	259.475979	643.580228	0.950791	248.415038
3	3092506	3092506	27.4	0.008801	0.019681	8.801465	19.680675	0.005469	0.012229	28.876199	...	57.173266	514.082140	0.983902	18.707701
4	3514799	3514799	21.6	0.127220	0.284472	127.219879	284.472297	0.079051	0.176763	417.388066	...	84.629307	495.597821	0.967687	158.263596

5 rows × 40 columns

Fig[21] : loading the dataset and printing head values

```
[ ] df.shape

(4687, 40)

[ ] df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4687 entries, 0 to 4686
Data columns (total 40 columns):
 #   Column                        Non-Null Count  Dtype
---  ------                        --------------  -----
 0   Neo Reference ID              4687 non-null   int64
 1   Name                          4687 non-null   int64
 2   Absolute Magnitude            4687 non-null   float64
 3   Est Dia in KM(min)            4687 non-null   float64
 4   Est Dia in KM(max)            4687 non-null   float64
 5   Est Dia in M(min)             4687 non-null   float64
 6   Est Dia in M(max)             4687 non-null   float64
 7   Est Dia in Miles(min)         4687 non-null   float64
 8   Est Dia in Miles(max)         4687 non-null   float64
 9   Est Dia in Feet(min)          4687 non-null   float64
 10  Est Dia in Feet(max)          4687 non-null   float64
 11  Close Approach Date           4687 non-null   object
 12  Epoch Date Close Approach     4687 non-null   int64
 13  Relative Velocity km per sec  4687 non-null   float64
 14  Relative Velocity km per hr   4687 non-null   float64
 15  Miles per hour                4687 non-null   float64
 16  Miss Dist.(Astronomical)      4687 non-null   float64
 17  Miss Dist.(lunar)             4687 non-null   float64
 18  Miss Dist.(kilometers)        4687 non-null   float64
 19  Miss Dist.(miles)             4687 non-null   float64
 20  Orbiting Body                 4687 non-null   object
 21  Orbit ID                      4687 non-null   int64
```

Fig[22] : printing the information of dataset

```
] df = df.drop(['Neo Reference ID', 'Name', 'Orbit ID', 'Close Approach Date','Epoch Date Close Approach', 'Orbit Determination Date'] , axis = 1)

] df.head()
```

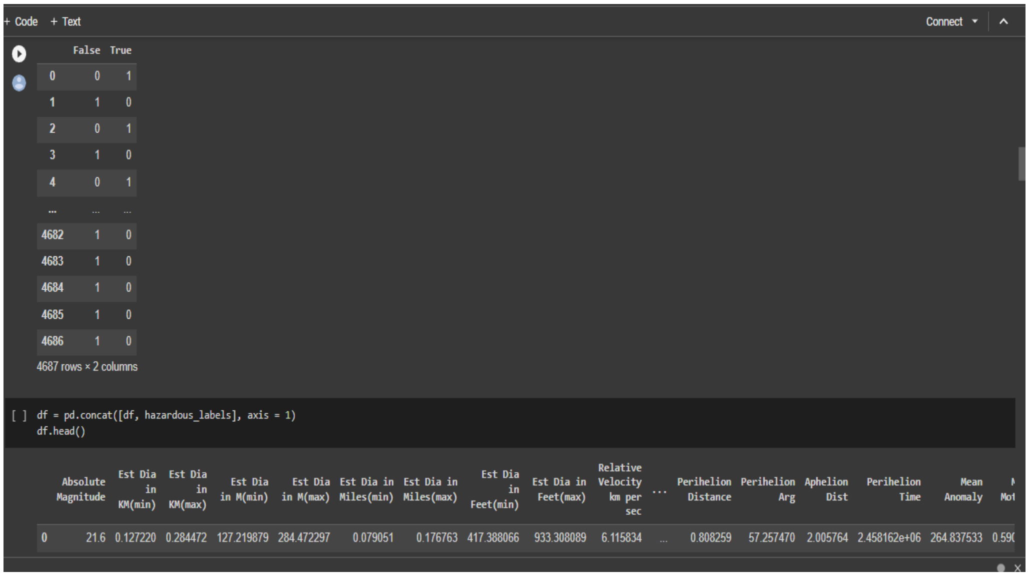
	Absolute Magnitude	Est Dia in KM(min)	Est Dia in KM(max)	Est Dia in M(min)	Est Dia in M(max)	Est Dia in Miles(min)	Est Dia in Miles(max)	Est Dia in Feet(min)	Est Dia in Feet(max)	Relative Velocity km per sec	...	Asc Node Longitude	Orbital Period	Perihelion Distance	Perihelion Arg	Aphelion Dist	Perih
0	21.6	0.127220	0.284472	127.219879	284.472297	0.079051	0.176763	417.388066	933.308089	6.115834	...	314.373913	609.599786	0.808259	57.257470	2.005764	2.45816
1	21.3	0.146068	0.326618	146.067964	326.617897	0.090762	0.202951	479.225620	1071.581063	18.113985	...	136.717242	425.869294	0.718200	313.091975	1.497352	2.45779
2	20.3	0.231502	0.517654	231.502122	517.654482	0.143849	0.321655	759.521423	1698.341531	7.590711	...	259.475979	643.580228	0.950791	248.415038	1.966857	2.45812
3	27.4	0.008801	0.019681	8.801465	19.680675	0.005469	0.012229	28.876199	64.569144	11.173874	...	57.173266	514.082140	0.983902	18.707701	1.527904	2.45790
4	21.6	0.127220	0.284472	127.219879	284.472297	0.079051	0.176763	417.388066	933.308089	9.840831	...	84.629307	495.597821	0.967687	158.263596	1.483543	2.45781

5 rows × 34 columns

```
] #one hot encoding

] hazardous_labels = pd.get_dummies(df['Hazardous'])
  hazardous_labels
```

	False	True
0	0	1
1	1	0

Fig[23] : one hot encoding on target variables



+ Code + Text Connect

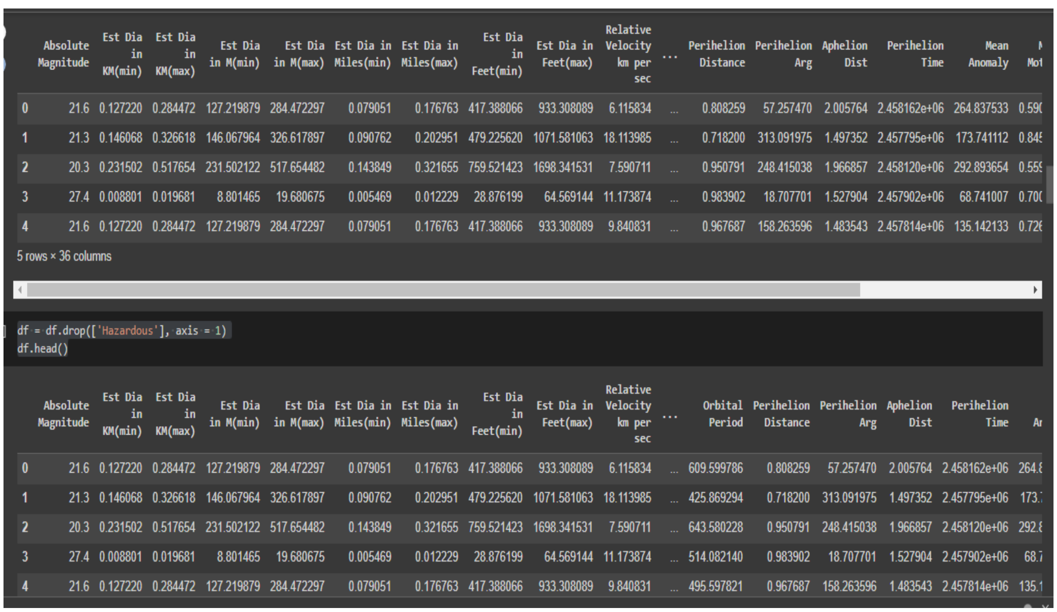
	False	True
0	0	1
1	1	0
2	0	1
3	1	0
4	0	1
...	...	...
4682	1	0
4683	1	0
4684	1	0
4685	1	0
4686	1	0

4687 rows × 2 columns

```
[ ] df = pd.concat([df, hazardous_labels], axis = 1)
    df.head()
```

	Absolute Magnitude	Est Dia in KM(min)	Est Dia in KM(max)	Est Dia in M(min)	Est Dia in M(max)	Est Dia in Miles(min)	Est Dia in Miles(max)	Est Dia in Feet(min)	Est Dia in Feet(max)	Relative Velocity km per sec	...	Perihelion Distance	Perihelion Arg	Aphelion Dist	Perihelion Time	Mean Anomaly	M Mot
0	21.6	0.127220	0.284472	127.219879	284.472297	0.079051	0.176763	417.388066	933.308089	6.115834	...	0.808259	57.257470	2.005764	2.458162e+06	264.837533	0.59(

Fig[24] : concatenate the labeled variables



	Absolute Magnitude	Est Dia in KM(min)	Est Dia in KM(max)	Est Dia in M(min)	Est Dia in M(max)	Est Dia in Miles(min)	Est Dia in Miles(max)	Est Dia in Feet(min)	Est Dia in Feet(max)	Relative Velocity km per sec	...	Perihelion Distance	Perihelion Arg	Aphelion Dist	Perihelion Time	Mean Anomaly	M Mot
0	21.6	0.127220	0.284472	127.219879	284.472297	0.079051	0.176763	417.388066	933.308089	6.115834	...	0.808259	57.257470	2.005764	2.458162e+06	264.837533	0.59(
1	21.3	0.146068	0.326618	146.067964	326.617897	0.090762	0.202951	479.225620	1071.581063	18.113985	...	0.718200	313.091975	1.497352	2.457795e+06	173.741112	0.84
2	20.3	0.231502	0.517654	231.502122	517.654482	0.143849	0.321655	759.521423	1698.341531	7.590711	...	0.950791	248.415038	1.966857	2.458120e+06	292.893654	0.55
3	27.4	0.008801	0.019681	8.801465	19.680675	0.005469	0.012229	28.876199	64.569144	11.173874	...	0.983902	18.707701	1.527904	2.457902e+06	68.741007	0.70(
4	21.6	0.127220	0.284472	127.219879	284.472297	0.079051	0.176763	417.388066	933.308089	9.840831	...	0.967687	158.263596	1.483543	2.457814e+06	135.142133	0.72

5 rows × 36 columns

```
df = df.drop(['Hazardous'], axis = 1)
df.head()
```

	Absolute Magnitude	Est Dia in KM(min)	Est Dia in KM(max)	Est Dia in M(min)	Est Dia in M(max)	Est Dia in Miles(min)	Est Dia in Miles(max)	Est Dia in Feet(min)	Est Dia in Feet(max)	Relative Velocity km per sec	...	Orbital Period	Perihelion Distance	Perihelion Arg	Aphelion Dist	Perihelion Time	Ar
0	21.6	0.127220	0.284472	127.219879	284.472297	0.079051	0.176763	417.388066	933.308089	6.115834	...	609.599786	0.808259	57.257470	2.005764	2.458162e+06	264.8
1	21.3	0.146068	0.326618	146.067964	326.617897	0.090762	0.202951	479.225620	1071.581063	18.113985	...	425.869294	0.718200	313.091975	1.497352	2.457795e+06	173.
2	20.3	0.231502	0.517654	231.502122	517.654482	0.143849	0.321655	759.521423	1698.341531	7.590711	...	643.580228	0.950791	248.415038	1.966857	2.458120e+06	292.8
3	27.4	0.008801	0.019681	8.801465	19.680675	0.005469	0.012229	28.876199	64.569144	11.173874	...	514.082140	0.983902	18.707701	1.527904	2.457902e+06	68.7
4	21.6	0.127220	0.284472	127.219879	284.472297	0.079051	0.176763	417.388066	933.308089	9.840831	...	495.597821	0.967687	158.263596	1.483543	2.457814e+06	135.1

Fig[25] : feature selection

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4687 entries, 0 to 4686
Data columns (total 35 columns):
 #   Column                        Non-Null Count  Dtype
---  ------                        --------------  -----
 0   Absolute Magnitude            4687 non-null   float64
 1   Est Dia in KM(min)            4687 non-null   float64
 2   Est Dia in KM(max)            4687 non-null   float64
 3   Est Dia in M(min)             4687 non-null   float64
 4   Est Dia in M(max)             4687 non-null   float64
 5   Est Dia in Miles(min)         4687 non-null   float64
 6   Est Dia in Miles(max)         4687 non-null   float64
 7   Est Dia in Feet(min)          4687 non-null   float64
 8   Est Dia in Feet(max)          4687 non-null   float64
 9   Relative Velocity km per sec  4687 non-null   float64
 10  Relative Velocity km per hr   4687 non-null   float64
 11  Miles per hour                4687 non-null   float64
 12  Miss Dist.(Astronomical)      4687 non-null   float64
 13  Miss Dist.(lunar)             4687 non-null   float64
 14  Miss Dist.(kilometers)        4687 non-null   float64
 15  Miss Dist.(miles)             4687 non-null   float64
 16  Orbiting Body                 4687 non-null   object
 17  Orbit Uncertainity            4687 non-null   int64
 18  Minimum Orbit Intersection    4687 non-null   float64
 19  Jupiter Tisserand Invariant   4687 non-null   float64
 20  Epoch Osculation              4687 non-null   float64
 21  Eccentricity                  4687 non-null   float64
 22  Semi Major Axis               4687 non-null   float64
 23  Inclination                   4687 non-null   float64
 24  Asc Node Longitude            4687 non-null   float64
 25  Orbital Period                4687 non-null   float64
 26  Perihelion Distance           4687 non-null   float64
```

Fig[26] : information of the dataset

```
#Observing Unique Values in Orbiting Body and Equinox
```

```
df['Orbiting Body'].value_counts()
```

```
Earth    4687
Name: Orbiting Body, dtype: int64
```

```
df['Equinox'].value_counts()
```

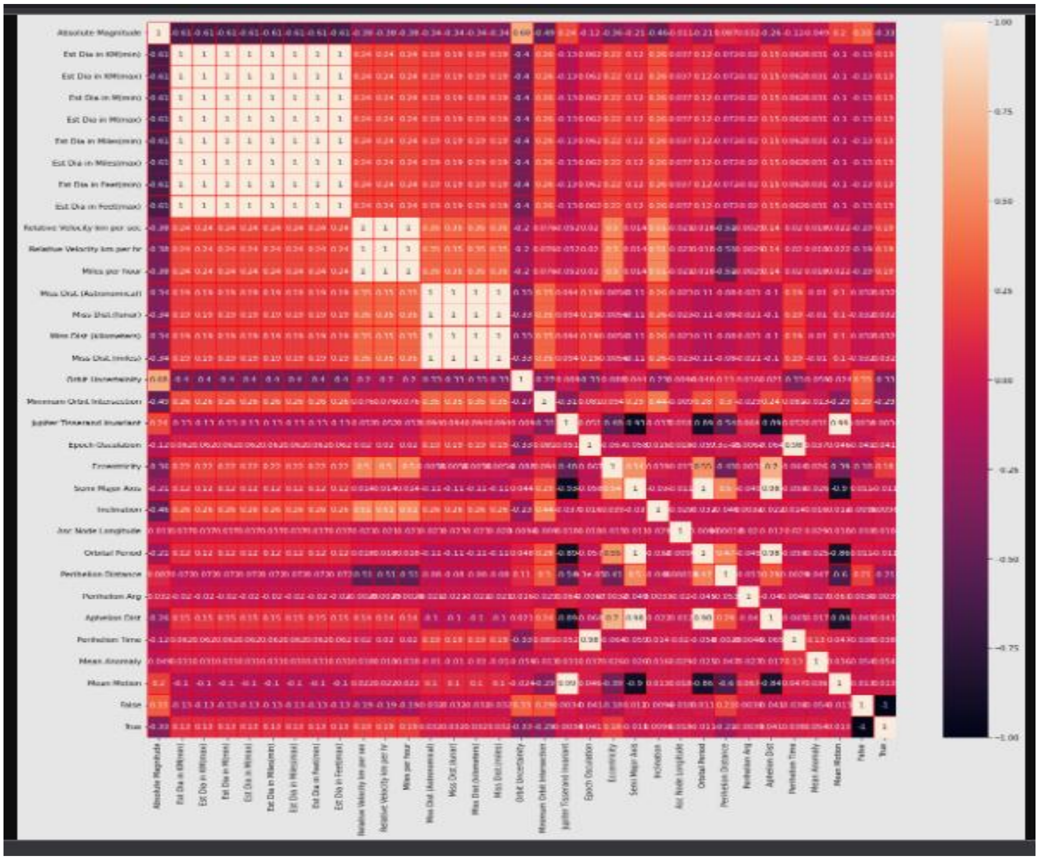
```
J2000    4687
Name: Equinox, dtype: int64
```

```
#Only single unique value, so both can be dropped
df = df.drop(['Orbiting Body', 'Equinox'], axis = 1)
```

```
df.head()
```

	Absolute Magnitude	Est Dia in KM(min)	Est Dia in KM(max)	Est Dia in M(min)	Est Dia in M(max)	Est Dia in Miles(min)	Est Dia in Miles(max)	Est Dia in Feet(min)	Est Dia in Feet(max)	Relative Velocity km per sec	...	Asc Node Longitude	Orbital Period	Perihelion Distance	Perihelion Arg	Aphelion Dist	Perih
0	21.6	0.127220	0.284472	127.219879	284.472297	0.079051	0.176763	417.388066	933.308089	6.115834	...	314.373913	609.599786	0.808259	57.257470	2.005764	2.4581
1	21.3	0.146068	0.326618	146.067964	326.617897	0.090762	0.202951	479.225620	1071.581063	18.113985	...	136.717242	425.869294	0.718200	313.091975	1.497352	2.4577
2	20.3	0.231502	0.517654	231.502122	517.654482	0.143849	0.321655	759.521423	1698.341531	7.590711	...	259.475979	643.580228	0.950791	248.415038	1.966857	2.4581
3	27.4	0.008801	0.019681	8.801465	19.680675	0.005469	0.012229	28.876199	64.569144	11.173874	...	57.173266	514.082140	0.983902	18.707701	1.527904	2.4579
4	21.6	0.127220	0.284472	127.219879	284.472297	0.079051	0.176763	417.388066	933.308089	9.840831	...	84.629307	495.597821	0.967687	158.263596	1.483543	2.4578

Fig[27] : drop the unnecessary features



Fig[28] : heat map of the dataset

```python
df = df.drop(['Est Dia in KM(max)', 'Est Dia in M(min)', 'Est Dia in M(max)', 'Est Dia in Miles(min)'
             ,'Est Dia in Miles(max)', 'Est Dia in Feet(min)', 'Est Dia in Feet(max)',
             'Relative Velocity km per hr', 'Miles per hour', 'Miss Dist.(lunar)',
             'Miss Dist.(kilometers)', 'Miss Dist.(miles)'], axis = 1)
```

```python
df.head()
```

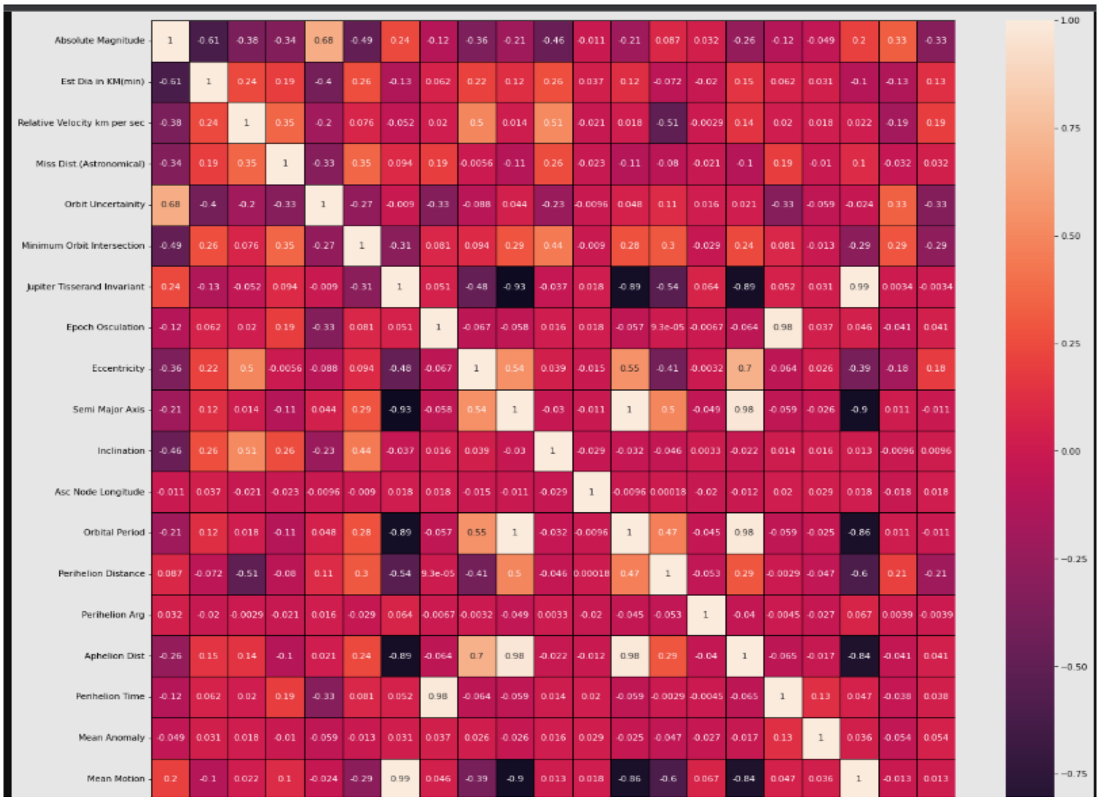
	Absolute Magnitude	Est Dia in KM(min)	Relative Velocity km per sec	Miss Dist.(Astronomical)	Orbit Uncertainity	Minimum Orbit Intersection	Jupiter Tisserand Invariant	Epoch Osculation	Eccentricity	Semi Major Axis	...	Asc Node Longitude	Orbital Period	Perihelion Distance	Perihelion Arg	Aphelic Dis
0	21.6	0.127220	6.115834	0.419483	5	0.025282	4.634	2458000.5	0.425549	1.407011	...	314.373913	609.599786	0.808259	57.257470	2.00576
1	21.3	0.146068	18.113985	0.383014	3	0.186935	5.457	2458000.5	0.351674	1.107776	...	136.717242	425.869294	0.718200	313.091975	1.49735
2	20.3	0.231502	7.590711	0.050956	0	0.043058	4.557	2458000.5	0.348248	1.458824	...	259.475979	643.580228	0.950791	248.415038	1.96685
3	27.4	0.008801	11.173874	0.285322	6	0.005512	5.093	2458000.5	0.216578	1.255903	...	57.173266	514.082140	0.983902	18.707701	1.5279(
4	21.6	0.127220	9.840831	0.407832	1	0.034798	5.154	2458000.5	0.210448	1.225615	...	84.629307	495.597821	0.967687	158.263596	1.4835

5 rows × 21 columns

```python
#Final Heatmap
plt.figure(figsize = (20,20))
sns.heatmap(df.corr(),annot = True,linewidths=0.5, linecolor='black')
```

<AxesSubplot:>

Fig[29] : drop the features



Fig[30] : heat map of new data set

```
[ ] #Drop the False Column, True is enough for classification

[ ] df.drop([False], axis = 1, inplace = True)

df.head()
```

	Absolute Magnitude	Est Dia in KM(min)	Relative Velocity km per sec	Miss Dist.(Astronomical)	Orbit Uncertainity	Minimum Orbit Intersection	Jupiter Tisserand Invariant	Epoch Osculation	Eccentricity	Semi Major Axis	Inclination	Asc Node Longitude	Orbital Period	Perihelion Distance	Perihelion Arg
0	21.6	0.127220	6.115834	0.419483	5	0.025282	4.634	2458000.5	0.425549	1.407011	6.025981	314.373913	609.599786	0.808259	57.257470
1	21.3	0.146068	18.113985	0.383014	3	0.186935	5.457	2458000.5	0.351674	1.107776	28.412996	136.717242	425.869294	0.718200	313.091975
2	20.3	0.231502	7.590711	0.050956	0	0.043058	4.557	2458000.5	0.348248	1.458824	4.237961	259.475979	643.580228	0.950791	248.415038
3	27.4	0.008801	11.173874	0.285322	6	0.005512	5.093	2458000.5	0.216578	1.255903	7.905894	57.173266	514.082140	0.983902	18.707701
4	21.6	0.127220	9.840831	0.407832	1	0.034798	5.154	2458000.5	0.210448	1.225615	16.793382	84.629307	495.597821	0.967687	158.263596

```
[ ] df.describe()
```

	Absolute Magnitude	Est Dia in KM(min)	Relative Velocity km per sec	Miss Dist.(Astronomical)	Orbit Uncertainity	Minimum Orbit Intersection	Jupiter Tisserand Invariant	Epoch Osculation	Eccentricity	Semi Major Axis	Inclination	Asc Node Longitude	Orbital Period	Pe I
count	4687.000000	4687.000000	4687.000000	4687.000000	4687.000000	4687.000000	4687.000000	4.687000e+03	4687.000000	4687.000000	4687.000000	4687.000000	4687.000000	468

Fig[31] : describe the dataset

```
[ ] #Model Buliding
    x = df.drop([True], axis = 1)
    y = df[True].astype(int)

[ ] from sklearn.model_selection import train_test_split
    x_train, x_test, y_train, y_test = train_test_split(x, y, random_state = 0 , test_size = 0.4)

    from sklearn.naive_bayes import GaussianNB
    clf = GaussianNB()
    clf.fit(x_train, y_train)
    cm_train, cm_test = classifiers(clf, 'Naive Bayes')

    Accuracy of Naive Bayes for Test Set = 0.9152
    Accuracy of Naive Bayes for Train Set = 0.8986486486486487

[ ] from sklearn.metrics import accuracy_score,classification_report,confusion_matrix

    predictions = clf.predict(x_test)
    acc = accuracy_score(y_test, predictions)
    print(str(np.round(acc*100, 2))+'%')
```

Fig[32] : label encoding

2.Support Vector Machine algorithm:

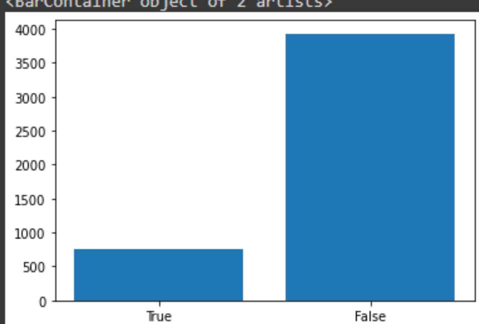
```
import warnings
warnings.filterwarnings("ignore")
```

```
#loading dataset
nasa = pd.read_csv("C:\\Users\\kanak\\OneDrive\\Desktop\\mpr\\nasa.csv")
del nasa['Name']
del nasa['Neo Reference ID']
del nasa["Close Approach Date"]
del nasa ["Orbit Determination Date"]
del nasa["Orbiting Body"]
del nasa['Est Dia in Feet(max)']
del nasa['Est Dia in Feet(min)']
del nasa['Est Dia in M(max)']
del nasa['Est Dia in Miles(max)']
del nasa['Est Dia in Miles(min)']
del nasa['Miles per hour']
del nasa['Relative Velocity km per sec']
del nasa['Equinox']
del nasa['Epoch Date Close Approach']
del nasa['Miss Dist.(Astronomical)']
del nasa['Miss Dist.(lunar)']
del nasa['Epoch Osculation']
del nasa['Perihelion Time']
```

Fig[33] : loading the dataset

```
import matplotlib.pyplot as plt
true_samples = np.sum(nasa['Hazardous'] == True)
false_samples = np.sum(nasa['Hazardous']==False)
plt.bar(["True","False"],height = [true_samples,false_samples])
```

```
<BarContainer object of 2 artists>
```




```
[ ] #Extracting out the True samples using boolean mask

nasa_true_mask = nasa['Hazardous'] ==True
nasa_true = nasa[nasa_true_mask]
len(nasa_true)
```

Fig[33] : visualizing the dataset

```
755

[ ] np.array(nasa).shape

(4687, 22)

[ ] bootstrap = nasa_true.sample(3177,replace=True)

    nasa = nasa.append(bootstrap)

from sklearn.preprocessing import StandardScaler
X,y = nasa.iloc[:,0:21],nasa.iloc[:,21]

Scaler = sklearn.preprocessing.StandardScaler()
Scaler.fit(X)
X = Scaler.transform(X)

binary = sklearn.preprocessing.LabelBinarizer(pos_label=1,neg_label=0)
lb = binary.fit(y)
y = lb.transform(y)

[ ] x_train,x_test,y_train,y_test = train_test_split(X,y,test_size=0.4,shuffle=True)
```

Fig[34] : splitting the dataset

```
+ Code  + Text

[ ] binary = sklearn.preprocessing.LabelBinarizer(pos_label=1,neg_label=0)
    lb = binary.fit(y)
    y = lb.transform(y)

[ ] x_train,x_test,y_train,y_test = train_test_split(X,y,test_size=0.4,shuffle=True)

[ ] from sklearn import svm

    svm = sklearn.svm.SVC().fit(x_train,y_train)

[ ] print("for svm classifier: ",svm.score(x_test,y_test))

    for svm classifier:  0.9577240940877304

[ ] y_pred =svm.predict(x_test)

from sklearn import metrics
cm = confusion_matrix(y_test, y_pred)
accuracy=metrics.accuracy_score(y_test, y_pred)
print(accuracy)
```

Fig[35] : training the dataset

3. Neural Networks:

```
[ ] np.array(nasa).shape

    (11041, 22)

    bootstrap = nasa_true.sample(3177,replace=True)

    nasa = nasa.append(bootstrap)

[ ] nasa.shape

    (14218, 22)

[ ] from sklearn.preprocessing import StandardScaler
    X,y = nasa.iloc[:,0:21],nasa.iloc[:,21]

    Scaler = sklearn.preprocessing.StandardScaler()
    Scaler.fit(X)
    X = Scaler.transform(X)

    binary = sklearn.preprocessing.LabelBinarizer(pos_label=1,neg_label=0)
    lb = binary.fit(y)
    y = lb.transform(y)
```

Fig[36] : preprocessing the data

```
[ ] x_train,x_test,y_train,y_test = train_test_split(X,y,test_size=0.2,shuffle=True)

    from keras.layers import Dense,Input
    from keras.models import Model

    def asteroid(input_shape=(21,)):

      X_input = Input(shape=input_shape)
      X = Dense(units=16,activation='relu')(X_input)
      X = Dense(units=8,activation="relu")(X)
      X = Dense(units=1,activation='sigmoid')(X)

      model = Model(inputs=X_input,outputs=X)

      return model

[ ] asteroid = asteroid()

    import keras.backend as K
    def f1(y_true, y_pred): #taken from old keras source code
        true_positives = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
        possible_positives = K.sum(K.round(K.clip(y_true, 0, 1)))
        predicted_positives = K.sum(K.round(K.clip(y_pred, 0, 1)))
        precision = true_positives / (predicted_positives + K.epsilon())
```

Fig[37] : working with keras library

```python
def f1(y_true, y_pred): #taken from old keras source code
    true_positives = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
    possible_positives = K.sum(K.round(K.clip(y_true, 0, 1)))
    predicted_positives = K.sum(K.round(K.clip(y_pred, 0, 1)))
    precision = true_positives / (predicted_positives + K.epsilon())
    recall = true_positives / (possible_positives + K.epsilon())
    f1_val = 2*(precision*recall)/(precision+recall+K.epsilon())
    return f1_val,precision,recall
def precision(y_true, y_pred): #taken from old keras source code
    true_positives = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
    possible_positives = K.sum(K.round(K.clip(y_true, 0, 1)))
    predicted_positives = K.sum(K.round(K.clip(y_pred, 0, 1)))
    precision = true_positives / (predicted_positives + K.epsilon())
    recall = true_positives / (possible_positives + K.epsilon())
    return precision
def recall(y_true, y_pred): #taken from old keras source code
    true_positives = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
    possible_positives = K.sum(K.round(K.clip(y_true, 0, 1)))
    predicted_positives = K.sum(K.round(K.clip(y_pred, 0, 1)))
    precision = true_positives / (predicted_positives + K.epsilon())
    recall = true_positives / (possible_positives + K.epsilon())
    return recall
```

```python
asteroid.compile(optimizer='adam',loss="binary_crossentropy",metrics=['accuracy',f1,precision,recall])
```

Fig[38] : defining evolutionary metrics

```
#Training Neural Network

asteroid.fit(x=x_train,y=y_train,epochs=100)

Epoch 1/100
356/356 [==============================] - 7s 5ms/step - loss: 0.4472 - accuracy: 0.8378 - f1: 0.8917 - precision: 0.8564 - recall: 0.9297
Epoch 2/100
356/356 [==============================] - 2s 5ms/step - loss: 0.1500 - accuracy: 0.9492 - f1: 0.9659 - precision: 0.9430 - recall: 0.9898
Epoch 3/100
356/356 [==============================] - 2s 5ms/step - loss: 0.0884 - accuracy: 0.9715 - f1: 0.9808 - precision: 0.9665 - recall: 0.9954
Epoch 4/100
356/356 [==============================] - 2s 5ms/step - loss: 0.0668 - accuracy: 0.9797 - f1: 0.9861 - precision: 0.9751 - recall: 0.9974
Epoch 5/100
356/356 [==============================] - 2s 5ms/step - loss: 0.0537 - accuracy: 0.9839 - f1: 0.9891 - precision: 0.9797 - recall: 0.9986
Epoch 6/100
356/356 [==============================] - 2s 5ms/step - loss: 0.0446 - accuracy: 0.9868 - f1: 0.9908 - precision: 0.9830 - recall: 0.9989
Epoch 7/100
356/356 [==============================] - 2s 5ms/step - loss: 0.0374 - accuracy: 0.9885 - f1: 0.9922 - precision: 0.9852 - recall: 0.9993
Epoch 8/100
356/356 [==============================] - 2s 5ms/step - loss: 0.0317 - accuracy: 0.9908 - f1: 0.9937 - precision: 0.9883 - recall: 0.9993
Epoch 9/100
356/356 [==============================] - 2s 5ms/step - loss: 0.0272 - accuracy: 0.9922 - f1: 0.9947 - precision: 0.9898 - recall: 0.9996
Epoch 10/100
356/356 [==============================] - 2s 5ms/step - loss: 0.0239 - accuracy: 0.9931 - f1: 0.9952 - precision: 0.9911 - recall: 0.9994
Epoch 11/100
356/356 [==============================] - 2s 4ms/step - loss: 0.0218 - accuracy: 0.9937 - f1: 0.9956 - precision: 0.9923 - recall: 0.9989
Epoch 12/100
356/356 [==============================] - 1s 4ms/step - loss: 0.0183 - accuracy: 0.9954 - f1: 0.9969 - precision: 0.9942 - recall: 0.9995
Epoch 13/100
356/356 [==============================] - 1s 4ms/step - loss: 0.0169 - accuracy: 0.9955 - f1: 0.9970 - precision: 0.9944 - recall: 0.9997
```

Fig[39] : training the model

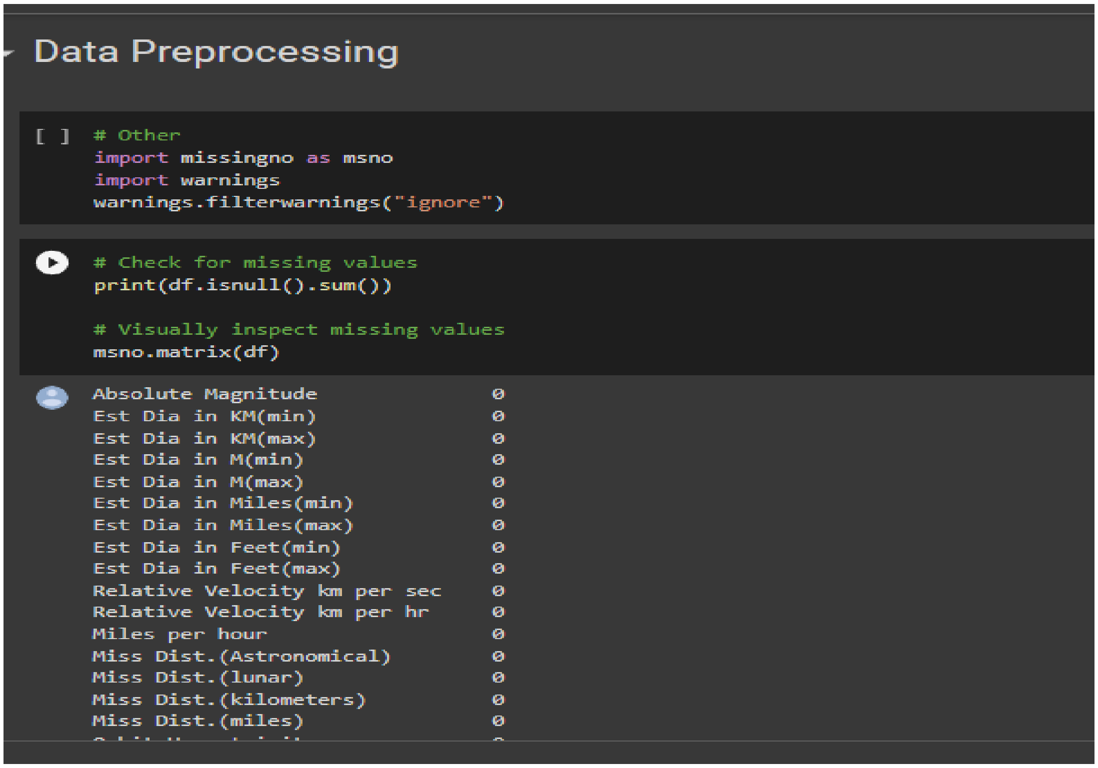
```
[ ] y_pred = asteroid.predict(x=x_test)

    89/89 [==============================] - 1s 3ms/step

    print(y_pred)

    [[6.0137763e-21]
     [1.0000000e+00]
     [4.8886095e-27]
     ...
     [1.0000000e+00]
     [9.9996167e-01]
     [9.9934292e-01]]

[ ] for i in range(len(y_pred)):

      if (y_pred[i] >= 0.5):
        y_pred[i]=1
      else :
        y_pred[i] = 0
    matrix = sklearn.metrics.confusion_matrix(y_test, y_pred)
    print(matrix)
    import seaborn
    seaborn.heatmap(matrix,annot=True,fmt='d')

    [[ 792    8]
     [   0 2044]]
    <AxesSubplot:>
```

Fig[40] : printing confusion matrix

```
[ ] for i in range(len(y_pred)):

      if (y_pred[i] >= 0.5):
        y_pred[i]=1
      else :
        y_pred[i] = 0
    matrix = sklearn.metrics.confusion_matrix(y_test, y_pred)
    print(matrix)
    import seaborn
    seaborn.heatmap(matrix,annot=True,fmt='d')

    [[ 792    8]
     [   0 2044]]
    <AxesSubplot:>
```

Fig[41] : visualizing the confusion matrix

4. Random Forest:



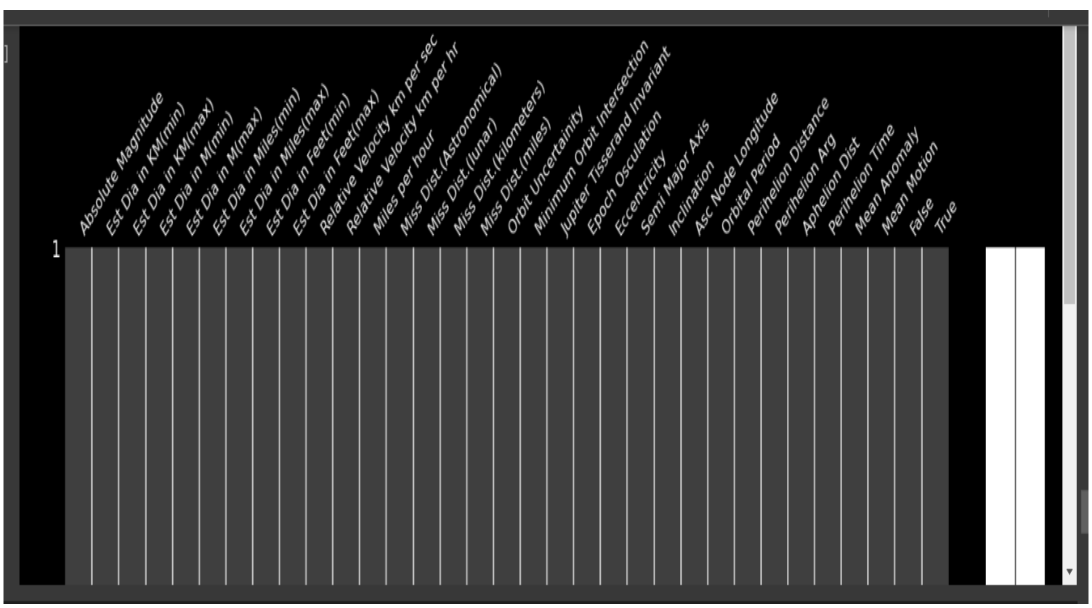
Data Preprocessing

```
[ ] # Other
    import missingno as msno
    import warnings
    warnings.filterwarnings("ignore")
```

```
# Check for missing values
print(df.isnull().sum())

# Visually inspect missing values
msno.matrix(df)
```

```
Absolute Magnitude                0
Est Dia in KM(min)                0
Est Dia in KM(max)                0
Est Dia in M(min)                 0
Est Dia in M(max)                 0
Est Dia in Miles(min)             0
Est Dia in Miles(max)             0
Est Dia in Feet(min)              0
Est Dia in Feet(max)              0
Relative Velocity km per sec      0
Relative Velocity km per hr       0
Miles per hour                    0
Miss Dist.(Astronomical)          0
Miss Dist.(lunar)                 0
Miss Dist.(kilometers)            0
Miss Dist.(miles)                 0
```

Fig[42] : data preprocessing



Fig[43] : visualizing dataset as a matrix

```python
#Heatmap of Correlation
plt.figure(figsize = (20,20))
sns.heatmap(df.corr(),annot = True)
```

<AxesSubplot:>

Fig[44] : visualizing heat map

```python
df = df.drop(['Est Dia in KM(max)', 'Est Dia in M(min)', 'Est Dia in M(max)', 'Est Dia in Miles(min)'
             ,'Est Dia in Miles(max)', 'Est Dia in Feet(min)', 'Est Dia in Feet(max)',
             'Relative Velocity km per hr', 'Miles per hour', 'Miss Dist.(lunar)',
             'Miss Dist.(kilometers)', 'Miss Dist.(miles)'], axis = 1)
```

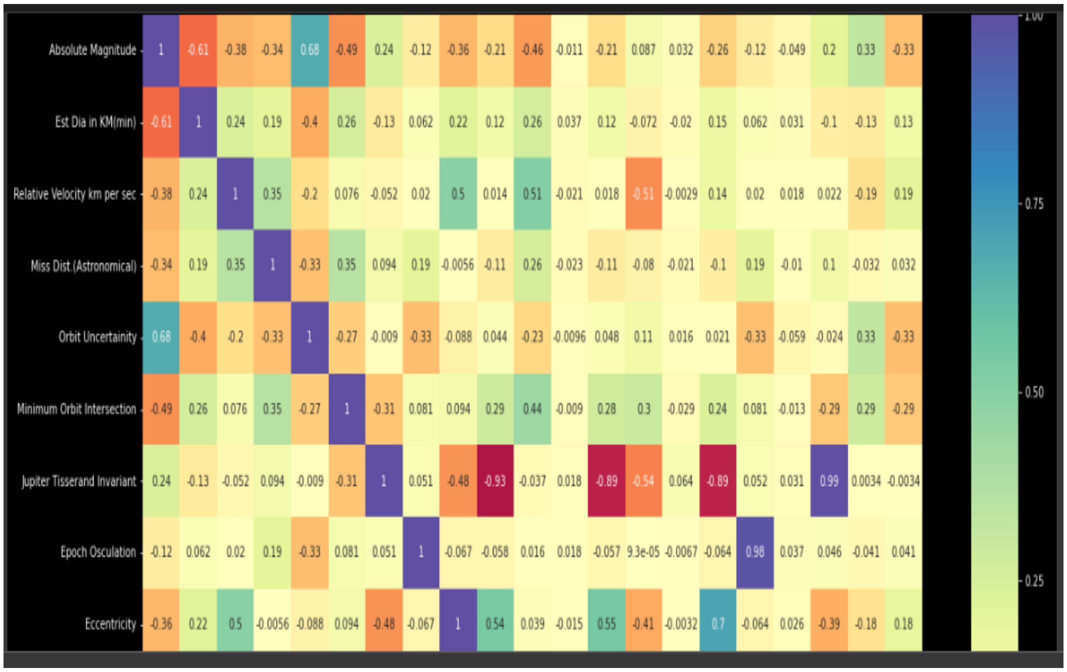
```python
df.head()
```

	Absolute Magnitude	Est Dia in KM(min)	Relative Velocity km per sec	Miss Dist.(Astronomical)	Orbit Uncertainity	Minimum Orbit Intersection	Jupiter Tisserand Invariant	Epoch Osculation	Eccentricity	Semi Major Axis	...	Asc Node Longitude	Orbital Period	Perihelion Distance	Perihelion Arg	Aphelic Dis
0	21.6	0.127220	6.115834	0.419483	5	0.025282	4.634	2458000.5	0.425549	1.407011	...	314.373913	609.599786	0.808259	57.257470	2.00576
1	21.3	0.146068	18.113985	0.383014	3	0.186935	5.457	2458000.5	0.351674	1.107776	...	136.717242	425.869294	0.718200	313.091975	1.49735
2	20.3	0.231502	7.590711	0.050956	0	0.043058	4.557	2458000.5	0.348248	1.458824	...	259.475979	643.580228	0.950791	248.415038	1.96685
3	27.4	0.008801	11.173874	0.285322	6	0.005512	5.093	2458000.5	0.216578	1.255903	...	57.173266	514.082140	0.983902	18.707701	1.52790
4	21.6	0.127220	9.840831	0.407832	1	0.034798	5.154	2458000.5	0.210448	1.225615	...	84.629307	495.597821	0.967687	158.263596	1.48354

5 rows × 21 columns

```python
#Final Heatmap
plt.figure(figsize = (20,20))
sns.heatmap(df.corr(),annot = True,cmap="Spectral")
```

Fig[45] : dropping unnecessary features



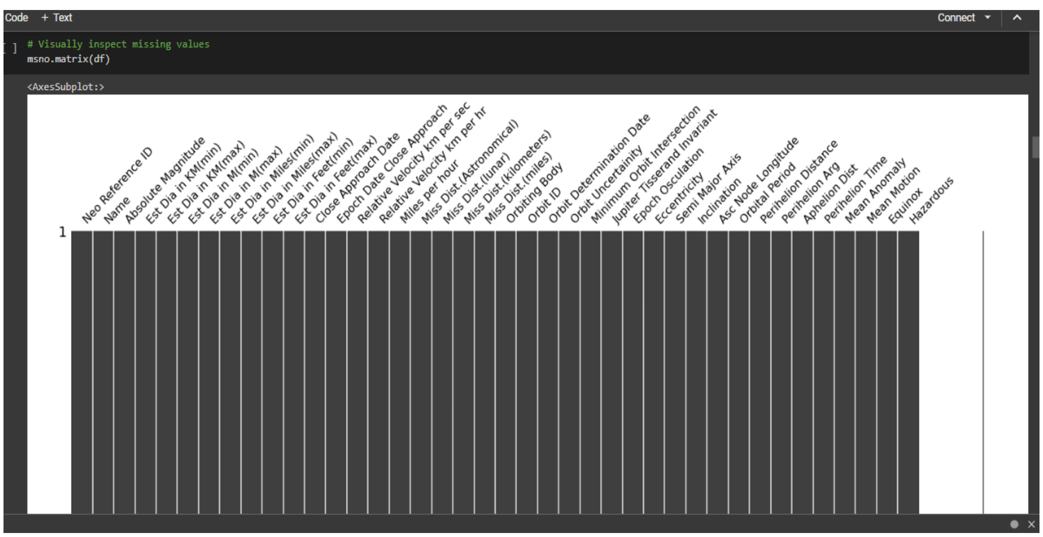
Fig[46] : visualizing heat map for new dataset

5. Logistic Regression:

```
# Check for missing values
print(df.isnull().sum())
```

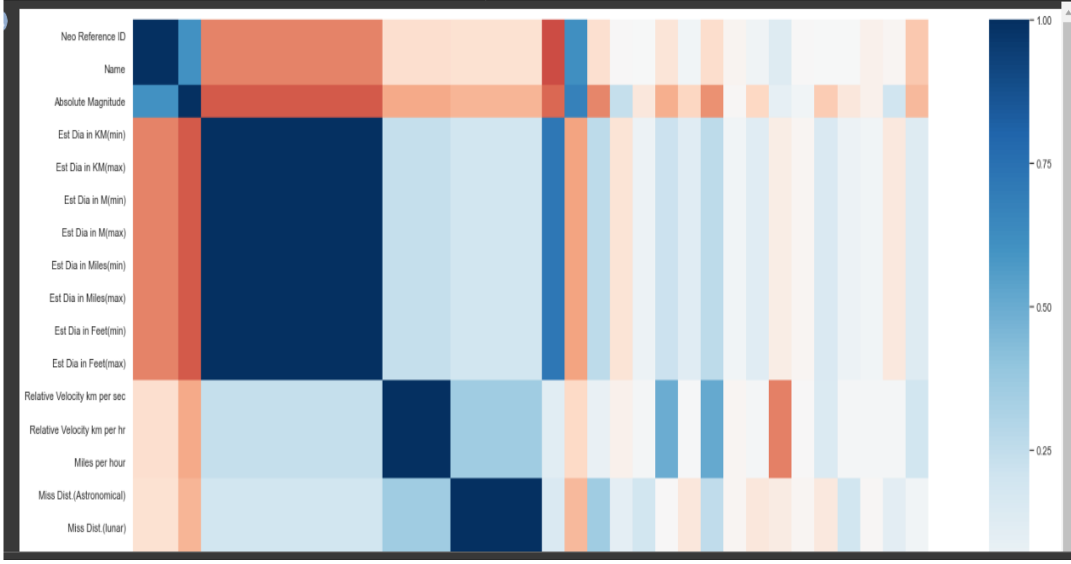
```
Neo Reference ID                 0
Name                             0
Absolute Magnitude               0
Est Dia in KM(min)               0
Est Dia in KM(max)               0
Est Dia in M(min)                0
Est Dia in M(max)                0
Est Dia in Miles(min)            0
Est Dia in Miles(max)            0
Est Dia in Feet(min)             0
Est Dia in Feet(max)             0
Close Approach Date              0
Epoch Date Close Approach        0
Relative Velocity km per sec     0
Relative Velocity km per hr      0
Miles per hour                   0
Miss Dist.(Astronomical)         0
Miss Dist.(lunar)                0
Miss Dist.(kilometers)           0
Miss Dist.(miles)                0
Orbiting Body                    0
Orbit ID                         0
Orbit Determination Date         0
Orbit Uncertainity               0
Minimum Orbit Intersection       0
Jupiter Tisserand Invariant      0
Epoch Osculation                 0
Eccentricity                     0
Semi Major Axis                  0
Inclination                      0
Asc Node Longitude               0
```

Fig[47] : checking for missing values



Fig[48] : visualizing the dataset as a matrix

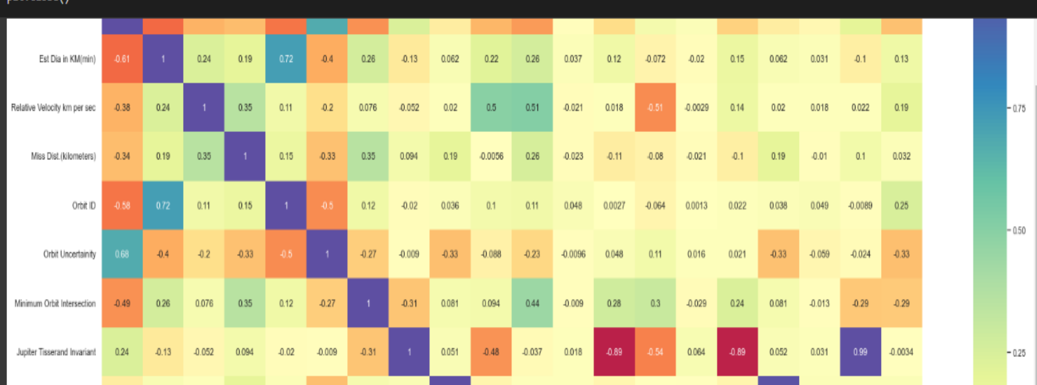
```python
#Checking visually for feature correlation
sns.set(rc={'figure.figsize':(30,20)})
sns.heatmap(df.corr(), vmin=-1, vmax=1, cmap="RdBu", annot=False,square=True)
plt.show()
plt.close()
```



Fig[49] : visualizing the correlation matrix

```python
# Dropping completely correlated features and datetime features
df = df.drop(['Est Dia in M(min)', 'Est Dia in M(max)', 'Est Dia in Miles(min)', 'Est Dia in Miles(max)', 'Est Dia in Feet(min)', 'Est Dia in Feet(max)', 'Est Dia in KM(max)',
              'Relative Velocity km per hr', 'Miles per hour',
              'Miss Dist.(Astronomical)', 'Miss Dist.(lunar)', 'Miss Dist.(miles)',
              'Semi Major Axis',
              'Neo Reference ID', 'Name',
              'Close Approach Date', 'Epoch Date Close Approach', 'Orbit Determination Date'],axis=1)
```

```python
# Plotting feature correlation with reduced dataset
sns.set(rc={'figure.figsize':(30,20)})
sns.heatmap(df.corr(), vmin=-1, vmax=1, cmap="Spectral", annot=True)
plt.show()
plt.close()
```



Fig[50] : visualizing the correlation with coefficients

```
[ ] # Encoding the target variable
    from sklearn.preprocessing import LabelEncoder
    l_enc = LabelEncoder()
    df['hazardous'] = l_enc.fit_transform(df.Hazardous)
    print('Hazardous == True -> 1')
    print('Hazardous == False -> 0\n')

    Hazardous == True -> 1
    Hazardous == False -> 0

[ ] # Checking if the other categorical features need to be encoded
    print(df['Orbiting Body'].unique())
    print(df['Equinox'].unique())
    print('\n')

    ['Earth']
    ['J2000']

    # Removing them since there is only a single value that is identical across all observations
    df = df.drop(['Orbiting Body', 'Equinox', 'Hazardous'], axis=1)

[ ] sns.set(rc={'figure.figsize':(30,20)})
    sns.heatmap(df.corr(), vmin=-1, vmax=1, cmap="Spectral", annot=True,annot_kws={'fontsize':10, 'fontweight':'bold'},square=True)
    plt.show()
    plt.close()
```

Fig[51] : drop the features

Fig[52] : heat map for new features

```
X= df.drop(columns = ['hazardous'], axis=1)
print(X)
```

```
      Absolute Magnitude  Est Dia in KM(min)  Relative Velocity km per sec  \
0                 21.600            0.127220                      6.115834
1                 21.300            0.146068                     18.113985
2                 20.300            0.231502                      7.590711
3                 27.400            0.008801                     11.173874
4                 21.600            0.127220                      9.840831
...                  ...                 ...                           ...
4682              23.900            0.044112                     22.154265
4683              28.200            0.006089                      3.225150
4684              22.700            0.076658                      7.191642
4685              21.800            0.116026                     11.352090
4686              19.109            0.400641                     35.946852

      Miss Dist.(kilometers)  Orbit ID  Orbit Uncertainity  \
0               6.275369e+07        17                   5
1               5.729815e+07        21                   3
2               7.622912e+06        22                   0
3               4.268362e+07         7                   6
4               6.101082e+07        25                   1
...                      ...       ...                 ...
4682            6.187511e+06         4                   8
4683            9.677324e+05         2                   6
4684            9.126775e+06        17                   6
4685            3.900908e+07         6                   5
4686            6.916986e+07        13                   6

      Minimum Orbit Intersection  Jupiter Tisserand Invariant  \
0                       0.025282                        4.634
1                       0.186935                        5.457
2                       0.043058                        4.557
3                       0.005512                        5.093
```

Fig[53] : printing the data of all the features

```
y= df[['hazardous']].astype(int)
print(y)
```

```
      hazardous
0             1
1             0
2             1
3             0
4             1
...         ...
4682          0
4683          0
4684          0
4685          0
4686          0

[4687 rows x 1 columns]
```

```
#ONE HOT ENCODING
from sklearn import preprocessing

encoder = preprocessing.OneHotEncoder(sparse=False)
enc_df = pd.DataFrame(encoder.fit_transform(df[['hazardous']]))
enc_df.columns = encoder.get_feature_names_out(['hazardous'])
d = df.join(enc_df)
d
```

	Absolute Magnitude	Est Dia in KM(min)	Relative Velocity km per sec	Miss Dist. (kilometers)	Orbit ID	Orbit Uncertainity	Minimum Orbit Intersection	Jupiter Tisserand Invariant	Epoch Osculation	Eccentricity	...	Orbital Period	Perihelion Distance	Perihelion Arg	Aphelion Dist	Periheli Ti
0	21.600	0.127220	6.115834	6.275369e+07	17	5	0.025282	4.634	2458000.5	0.425549	...	609.599786	0.808259	57.257470	2.005764	2.458162e+

Fig[54] : one hot encoding

```
[ ] # Splitting the data into Train and Test

[ ] from sklearn.model_selection import train_test_split
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.40, random_state=0)

    print(y_train.shape)
    print(y_test.shape)

    (2812, 1)
    (1875, 1)

[ ] lg = LogisticRegression()
    lg = lg.fit(X_train,y_train)

[ ] pred = lg.predict(X_test)

[ ] score = lg.score(X_test, y_test)
    score

    0.8373333333333334

[ ] from sklearn.metrics import accuracy_score,classification_report,confusion_matrix

    predictions = lg.predict(X_test)
    acc = accuracy_score(y_test, predictions)
    print(str(np.round(acc*100, 2))+'%')

    83.73%
```

Fig[55] : training the model

6. XGBoost:

```
[ ] np.array(nasa).shape

    (4687, 22)

    #Using Bootstrapping to balance dataset
    bootstrap = nasa_true.sample(3177,replace=True)

    nasa = nasa.append(bootstrap)

[ ] nasa.shape

    (7864, 22)

[ ] #Labelling the Data
    X,y = nasa.iloc[:,0:21],nasa.iloc[:,21]

    Scaler = sklearn.preprocessing.StandardScaler()
    Scaler.fit(X)
    X = Scaler.transform(X)

    binary = sklearn.preprocessing.LabelBinarizer(pos_label=1,neg_label=0)
    lb = binary.fit(y)
    y = lb.transform(y)

[ ] #Splitting the data into train and test

    x_train,x_test,y_train,y_test = train_test_split(X,y,test_size=0.4,shuffle=True)

[ ] # fitting to sgd model
```

Fig[56] : labeling and splitting the dataset

```
] # fitting to sgd model
  fitted_sgd = SGDClassifier(loss='log').fit(x_train,y_train)

] print("for sgd classifier: ",fitted_sgd.score(x_test,y_test))

  for sgd classifier:  0.9291163382072473

] y_pred = fitted_sgd.predict(x_test)

] #!pip install xgboost

] from xgboost import XGBClassifier
  xgb = XGBClassifier(n_estimators=128,bootstrap=False)
  xgb_fit = xgb.fit(x_train,y_train)
  y_pred1 = xgb.predict(x_test)
  recall = sklearn.metrics.recall_score(y_test, y_pred1)
  precision = sklearn.metrics.precision_score(y_test,y_pred1)
  f1_score = (2*precision*recall)/(precision+recall)
  print("The Precision is: ",precision)
  print("The Recall is: ",recall)
  print("The F1 Score is: ",f1_score)
  print("The Accuracy is:",xgb_fit.score(x_test,y_test))

  [13:40:06] WARNING: C:/buildkite-agent/builds/buildkite-windows-cpu-autoscaling-group-i-0fc7796c793e6356f-1/xgboost/xgboost-ci-windows/src/learner.cc:767:
  Parameters: { "bootstrap" } are not used.

  The Precision is:  0.9993493819128172
  The Recall is:  1.0
  The F1 Score is:  0.9996745850959974
  The Accuracy is: 0.9996821360457724
```

Fig[56] : training the model

7. Decision tree:

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import missingno as msno
```

```python
df = pd.read_csv('C:\\Users\\kanak\\OneDrive\\Desktop\\mpr\\nasa.csv')
```

```python
df.head()
```

	Neo Reference ID	Name	Absolute Magnitude	Est Dia in KM(min)	Est Dia in KM(max)	Est Dia in M(min)	Est Dia in M(max)	Est Dia in Miles(min)	Est Dia in Miles(max)	Est Dia in Feet(min)	...	Asc Node Longitude	Orbital Period	Perihelion Distance	Perihelion Arg	Aphelion Dist	Periheli Ti
0	3703080	3703080	21.6	0.127220	0.284472	127.219879	284.472297	0.079051	0.176763	417.388066	...	314.373913	609.599786	0.808259	57.257470	2.005764	2.458162e
1	3723955	3723955	21.3	0.146068	0.326618	146.067964	326.617897	0.090762	0.202951	479.225620	...	136.717242	425.869294	0.718200	313.091975	1.497352	2.457795e
2	2446862	2446862	20.3	0.231502	0.517654	231.502122	517.654482	0.143849	0.321655	759.521423	...	259.475979	643.580228	0.950791	248.415038	1.966857	2.458120e
3	3092506	3092506	27.4	0.008801	0.019681	8.801465	19.680675	0.005469	0.012229	28.876199	...	57.173266	514.082140	0.983902	18.707701	1.527904	2.457902e
4	3514799	3514799	21.6	0.127220	0.284472	127.219879	284.472297	0.079051	0.176763	417.388066	...	84.629307	495.597821	0.967687	158.263596	1.483543	2.457814e

5 rows × 40 columns

```python
df.shape
```

```
(4687, 40)
```

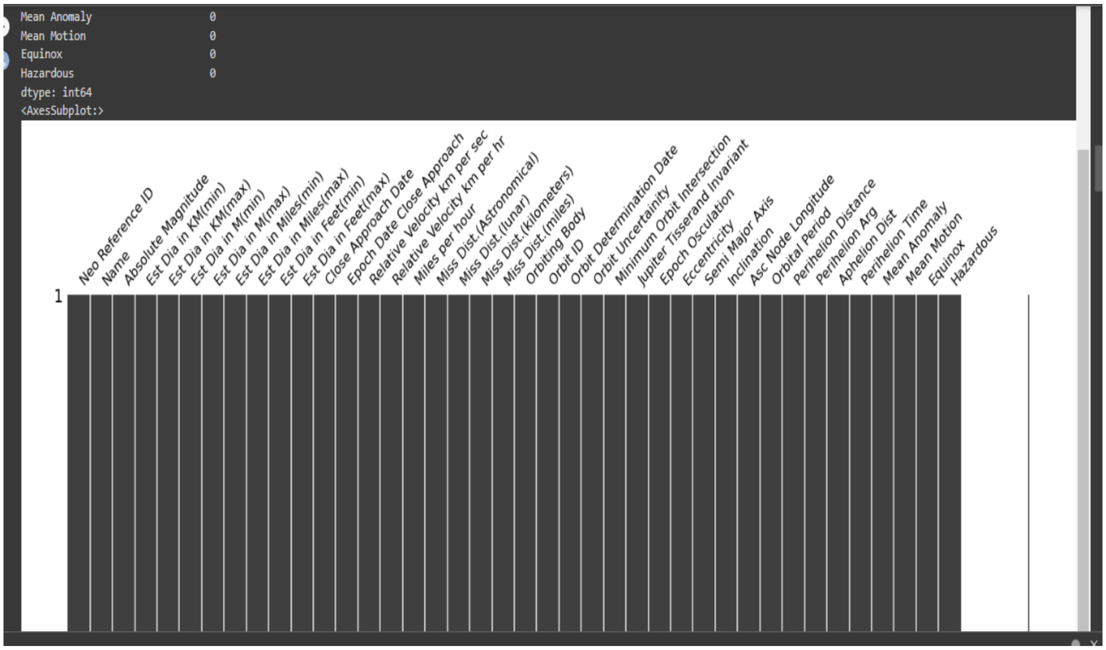
Fig[57] : importing the libraries

```python
from sklearn.tree import DecisionTreeClassifier, plot_tree
```

```python
# Check for missing values
print(df.isnull().sum())

# Visually inspect missing values
msno.matrix(df)
```

```
Neo Reference ID                0
Name                            0
Absolute Magnitude              0
Est Dia in KM(min)              0
Est Dia in KM(max)              0
Est Dia in M(min)               0
Est Dia in M(max)               0
Est Dia in Miles(min)           0
Est Dia in Miles(max)           0
Est Dia in Feet(min)            0
Est Dia in Feet(max)            0
Close Approach Date             0
Epoch Date Close Approach       0
Relative Velocity km per sec    0
Relative Velocity km per hr     0
Miles per hour                  0
Miss Dist.(Astronomical)        0
Miss Dist.(lunar)               0
Miss Dist.(kilometers)          0
Miss Dist.(miles)               0
Orbiting Body                   0
Orbit ID                        0
Orbit Determination Date        0
Orbit Uncertainity              0
```

Fig[58] : printing the dataset as a matrix



Fig[59] : dataset as a matrix

Fig[60] : heatmap for dataset

```python
# Dropping completely correlated features and datetime features
df = df.drop(['Est Dia in M(min)', 'Est Dia in M(max)', 'Est Dia in Miles(min)', 'Est Dia in Miles(max)', 'Est Dia in Feet(min)', 'Est Dia in Feet(max)', 'Est Dia in KM(max)',
              'Relative Velocity km per hr', 'Miles per hour',
              'Miss Dist.(Astronomical)', 'Miss Dist.(lunar)', 'Miss Dist.(miles)',
              'Semi Major Axis',
              'Neo Reference ID', 'Name',
              'Close Approach Date', 'Epoch Date Close Approach', 'Orbit Determination Date'],axis=1)

# Plotting feature correlation with reduced dataset
sns.set(rc={'figure.figsize':(30,20)})
sns.heatmap(df.corr(), vmin=-1, vmax=1, cmap="Spectral", annot=True)
plt.show()
plt.close()
```

Fig[61] : heatmap for new dataset

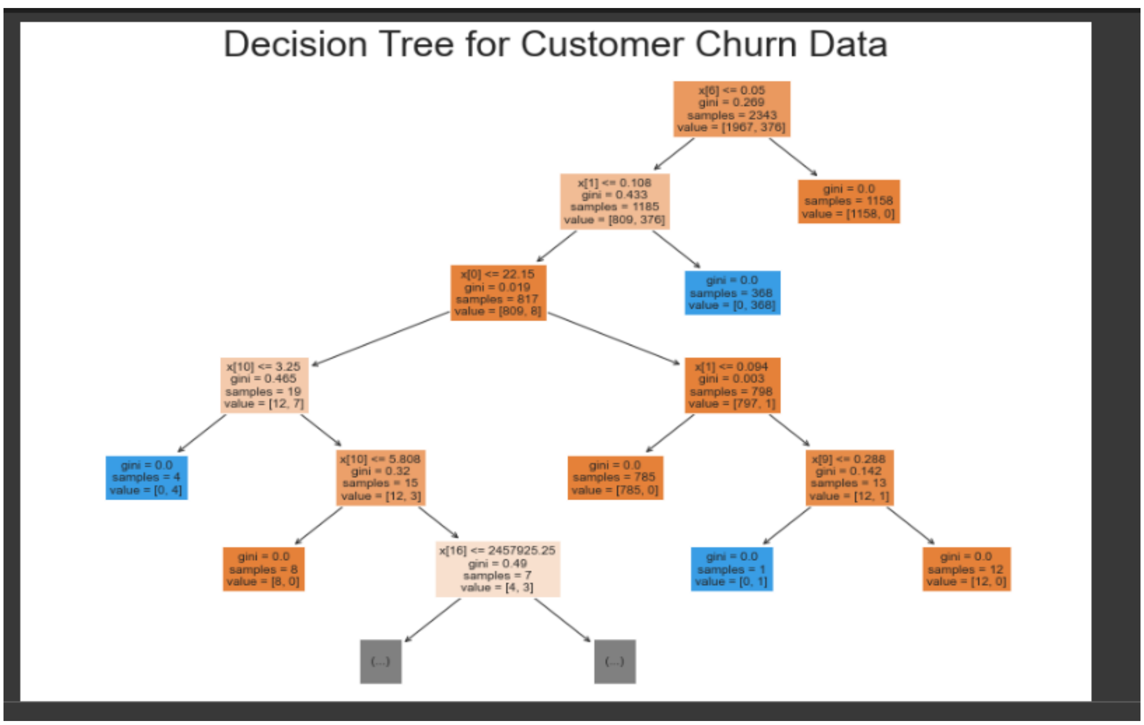
```python
# Encoding the target variable
from sklearn.preprocessing import LabelEncoder
l_enc = LabelEncoder()
df['hazardous'] = l_enc.fit_transform(df.Hazardous)
print('Hazardous == True -> 1')
print('Hazardous == False -> 0\n')

# Checking if the other categorical features need to be encoded
print(df['Orbiting Body'].unique())
print(df['Equinox'].unique())
print('\n')
# Removing them since there is only a single value that is identical across all observations
df = df.drop(['Orbiting Body', 'Equinox', 'Hazardous'], axis=1)

# Check after all the changes
print(df.info())
df.head()
```

```
Hazardous == True -> 1
Hazardous == False -> 0

['Earth']
['J2000']


<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4687 entries, 0 to 4686
Data columns (total 20 columns):
 #   Column                        Non-Null Count  Dtype
---  ------                        --------------  -----
 0   Absolute Magnitude            4687 non-null   float64
 1   Est Dia in KM(min)            4687 non-null   float64
```

Fig[62] : label encoding

```
[ ] X= df.drop(columns = ['hazardous'], axis=1)
    y= df[['hazardous']]

[ ] from sklearn.model_selection import train_test_split
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.50, random_state=0)

[ ] from sklearn.tree import DecisionTreeClassifier
    clf = DecisionTreeClassifier()
    clf = clf.fit(X_train,y_train)

[ ] pred1 = clf.predict(X_test)

[ ] from sklearn import tree
    s = plt.figure(figsize=(15,10))
    tree.plot_tree(clf, max_depth=5, filled=True, fontsize=10)
    plt.title("Decision Tree for Customer Churn Data", fontsize=30)
    plt.show()
```

Fig[63] : training dataset



Fig[63] : visualizing the dataset

CHAPTER 8

RESULTS

1. Gaussian Naive Bayes Algorithm:

```
In [46]: y_pred=clf.predict(x_test)
         input_list=x_test
         for i in range(10):
             print("testing  "+str(i)+"--------",y_pred[i])

         testing  0-------- 0
         testing  1-------- 0
         testing  2-------- 0
         testing  3-------- 1
         testing  4-------- 1
         testing  5-------- 0
         testing  6-------- 0
         testing  7-------- 0
         testing  8-------- 1
         testing  9-------- 0

In [47]: from sklearn.metrics import accuracy_score,classification_report,confusion_matrix

         predictions = clf.predict(x_test)
         acc = accuracy_score(y_test, predictions)
         print(str(np.round(acc*100, 2))+'%')

         91.52%

In [48]: print(classification_report(y_test,predictions))

                       precision    recall  f1-score   support

                    0       0.96      0.93      0.95      1570
                    1       0.71      0.82      0.76       305

             accuracy                           0.92      1875
            macro avg       0.83      0.88      0.85      1875
         weighted avg       0.92      0.92      0.92      1875

In [49]: print(confusion_matrix(y_test,predictions))

         [[1465  105]
          [  54  251]]
```

Fig[64] : results of Gaussian Naive Bayes algorithm

2.Support Vector Machine algorithm:

```
y_pred =svm.predict(x_test)
input_list=x_test
for i in range(5):
    print("testing  "+str(i)+"--------",y_pred[i])
```

```
testing  0-------- 1
testing  1-------- 0
testing  2-------- 0
testing  3-------- 1
testing  4-------- 0
```

```
from sklearn.metrics import classification_report,accuracy_score,confusion_matrix
cm = confusion_matrix(y_test, y_pred)
accuracy=accuracy_score(y_test, y_pred)
confusion_mat=confusion_matrix(y_test,y_pred)
print(accuracy)
```

```
0.9425630810092962
```

```
print(confusion_mat)
```

```
[[1100   78]
 [  10 1172]]
```

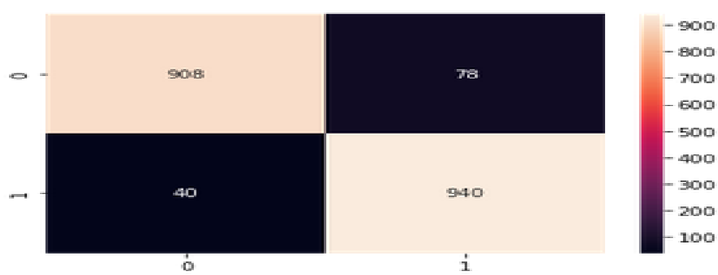
Fig[65] : results of Support Vector Machine

3.Neural Networks:

```
[63]: for i in range(len(y_pred)):

        if (y_pred[i] >= 0.5):
          y_pred[i]=1
        else :
          y_pred[i] = 0
      matrix = sklearn.metrics.confusion_matrix(y_test, y_pred)
      print(matrix)
      import seaborn
      seaborn.heatmap(matrix,annot=True,fmt='d')
```

```
[[908  78]
 [ 40 940]]
```

```
t[63]: <AxesSubplot:>
```




```
[64]: from sklearn.metrics import classification_report,accuracy_score
      accuracy_score(y_test,y_pred)
```

```
t[64]: 0.9399796541200407
```

Fig[66] : results of Neural Networks

4. Random Forest:

```
In [89]: pred2 = rf.predict(x_test)
         input_list=x_test
         for i in range(5):
             print("testing  "+str(i)+"--------",pred2[i])

         testing  0-------- 0
         testing  1-------- 0
         testing  2-------- 0
         testing  3-------- 1
         testing  4-------- 0

In [90]: from sklearn.metrics import classification_report,accuracy_score,confusion_matrix
         accuracy_score(y_test,pred2)

Out[90]: 0.9893390191897654

In [91]: confusion_matrix(pred2,y_test)

Out[91]: array([[387,   4],
                [  1,  77]], dtype=int64)
```

Fig[67] : result of Random Forest

5.Logistic Regression:

```
In [57]: pred = lg.predict(x_test)
         input_list=x_test
         for i in range(10):
             print("testing  "+str(i)+"--------",pred[i])

         testing  0-------- 0
         testing  1-------- 0
         testing  2-------- 0
         testing  3-------- 0
         testing  4-------- 0
         testing  5-------- 0
         testing  6-------- 0
         testing  7-------- 0
         testing  8-------- 0
         testing  9-------- 0

In [59]: from sklearn.metrics import accuracy_score,classification_report,confusion_matrix

         predictions = lg.predict(x_test)
         acc = accuracy_score(y_test, predictions)
         print(str(np.round(acc*100, 2))+'%')

         84.54%

In [60]: print(classification_report(y_test,predictions))

                       precision    recall  f1-score   support

                    0       0.85      1.00      0.92       793
                    1       0.00      0.00      0.00       145

             accuracy                           0.85       938
            macro avg       0.42      0.50      0.46       938
         weighted avg       0.71      0.85      0.77       938

In [61]: print(confusion_matrix(y_test,predictions))

         [[793   0]
          [145   0]]
```

Fig[68] : results of Logistic Regression

6. XGBoosting algorithm:

```
n [21]: from xgboost import XGBClassifier
        xgb = XGBClassifier(n_estimators=128,bootstrap=False)
        xgb_fit = xgb.fit(x_train,y_train)
        y_pred1 = xgb.predict(x_test)
        recall = sklearn.metrics.recall_score(y_test, y_pred1)
        precision = sklearn.metrics.precision_score(y_test,y_pred1)
        f1_score = (2*precision*recall)/(precision+recall)
        print("The Precision is: ",precision)
        print("The Recall is: ",recall)
        print("The F1 Score is: ",f1_score)
        print("The Accuracy is:",xgb_fit.score(x_test,y_test))

        [16:14:18] WARNING: C:/buildkite-agent/builds/buildkite-windows-cpu-autoscaling-group-i-0fc7796c793e6356f-1/xgboost/xgboost-
        ows/src/learner.cc:767:
        Parameters: { "bootstrap" } are not used.

        The Precision is:  0.9993610223642172
        The Recall is:  0.9993610223642172
        The F1 Score is:  0.9993610223642172
        The Accuracy is: 0.9993642720915448

n [23]: y_pred1 = xgb.predict(x_test)
        input_list=x_test
        for i in range(10):
            print("testing  "+str(i)+"--------",y_pred1[i])

        testing  0-------- 1
        testing  1-------- 0
        testing  2-------- 1
        testing  3-------- 0
        testing  4-------- 1
        testing  5-------- 1
        testing  6-------- 0
        testing  7-------- 1
        testing  8-------- 0
        testing  9-------- 0
```

Fig[69] : results of XGBoosting

7. Decision Tree algorithm:

```
In [28]: pred1 = clf_id.predict(X_test)
         input_list=X_test
         for i in range(10):
             print("testing  "+str(i)+"--------",pred1[i])

         testing  0-------- 0
         testing  1-------- 0
         testing  2-------- 0
         testing  3-------- 0
         testing  4-------- 1
         testing  5-------- 0
         testing  6-------- 0
         testing  7-------- 0
         testing  8-------- 0
         testing  9-------- 0

In [29]: from sklearn.metrics import accuracy_score,classification_report,confusion_matrix

         predictions = clf.predict(X_test)
         acc = accuracy_score(y_test, predictions)
         print(str(np.round(acc*100, 2))+'%')

         82.24%

In [30]: print(classification_report(y_test,predictions))

                       precision    recall  f1-score   support

                    0       0.85      0.95      0.90      3924
                    1       0.37      0.15      0.21       754

             accuracy                           0.82      4678
            macro avg       0.61      0.55      0.56      4678
         weighted avg       0.78      0.82      0.79      4678
```

Fig[70] : results of Decision Tree

CHAPTER 9

CONCLUSION AND FUTURE WORK

In this project, we have built a Machine learning model which can show the probability value of a planet(earth) that may be hit by an asteroid. Based on the data we have collected from the space research center , we have classified whether the asteroid is precarious or not. At the end we have classified whether the asteroid is hazardous or not . These values may change in the future according to asteroid revolving actions or based on the technology growth . The model which predicts low accuracy, may have high accuracy in the future and vice versa . So, our model needs continuous data to get the inputs on a daily basis. In the future , based on the continuous data we get from the space research center, we will build software which can be used by the researchers for predicting the characteristics of an asteroid. That software will collect the data from the research center continuously, so our model which is implemented in that software will show the predicted probability and then the researchers can perform actions according to the output.

CHAPTER 10

REFERENCES

[1] https://ieeexplore.ieee.org/document/9697222

[2] https://ieeexplore.ieee.org/document/9753945

[3] https://ui.adsabs.harvard.edu/abs/2020CoBAO..67..329C/abstract

[4] https://medium.com/analytics-vidhya/detecting-potentially-hazardous-asteroids-using-deEp-learning-part-1-9873a0d97ac8

[5] https://cneos.jpl.nasa.gov/about/neo_groups.html

[6] https://astronomy.swin.edu.au/cosmos/p/Potentially+Hazardous+Asteroids

[7] https://fscj.pressbooks.pub/introductionastronomy/chapter/asteroid-classification/

[8] https://weather.com/en-IN/india/space/news/2021-07-01-classification-of-asteroids-interactions-with-earth-and-more

[9] https://www.kaggle.com/datasets/brsdincer/asteroid-classification-for-hazardous-prediction

[10] https://www.iasabhiyan.com/what-are-asteroids-and-how-are-they-grouped/