Asteroid Prediction by Using Supervised Learning Algorithms in Machine Learning

Course Name & Code:

Applied Machine Learning (SWE – 687)

Project Report

Prepared By (Name with ID)

Md Rakib Hasan (3002)

Md. Ashraful Alam (3011)

Students of M. Sc in SWE

Department of Software Engineering

Daffodil International University

Submitted to

Mr. Fazly Rabbi

Faculty of Department of Software Engineering

Daffodil International University

Date: May 27, 2023

Abstract

The Asteroid Prediction project is a machine learning project that aims to predict whether an asteroid is hazardous or not based on various features such as its size, speed, and orbit parameters. The dataset used for this project contains information about over 93,000 asteroids obtained from NASA's Near-Earth Object program. The project involves exploratory data analysis, data preprocessing, and building a binary classification model using various supervised learning algorithms. The performance of the models is evaluated using various metrics such as accuracy, precision, recall, and F1 score. The best performing model is then used to predict the hazard status of a test dataset. The project demonstrates how machine learning algorithms can be applied to real-world problems and can potentially help in identifying hazardous asteroids that may pose a threat to the Earth.

The One-Stop Destination for Asteroids Classification project is a machine learning project that aims to classify asteroids into different categories based on their physical properties such as their size, shape, and composition. The dataset used for this project contains information about over 16,000 asteroids obtained from NASA's Small Bodies Dataferrett.

The project involves exploratory data analysis, data preprocessing, and building a multiclass classification model using various supervised learning algorithms. The performance of the models is evaluated using various metrics such as accuracy, precision, recall, and F1 score. The best performing model is then used to predict the category of a test dataset.

The project demonstrates how machine learning algorithms can be applied to real-world problems and can help in classifying asteroids based on their properties. The project can potentially aid in studying the properties of different types of asteroids and in identifying asteroids that may pose a threat to the Earth. The project also highlights the importance of data analysis and feature engineering in building accurate machine learning models.

Table of Contents

Chapter 1: Introduction

The Asteroid Prediction project is a machine learning project that aims to predict whether an asteroid is hazardous or not based on various features such as its size, speed, and orbit parameters. The project uses a dataset obtained from NASA's Near-Earth Object program, which contains information about over 93,000 asteroids.

Asteroids are small rocky objects that orbit the sun, and they can pose a threat to the Earth if they collide with it. Identifying hazardous asteroids is important to prevent such collisions and to develop mitigation strategies in case a collision is imminent. Machine learning algorithms can potentially help in identifying hazardous asteroids by analyzing their characteristics and predicting their hazard status.

The Asteroid Prediction project involves several steps such as data cleaning, exploratory data analysis, feature engineering, and building a binary classification model. The project uses various supervised learning algorithms such as logistic regression, decision trees, random forest, and gradient boosting to build the classification model. The performance of the models is evaluated using various metrics such as accuracy, precision, recall, and F1 score. The best performing model is then used to predict the hazard status of a test dataset.

The One-Stop Destination for Asteroids Classification project involves several steps such as data preprocessing, exploratory data analysis, feature engineering, and building a multi-class classification model. The project uses various supervised learning algorithms such as logistic regression, decision trees, random forest, and support vector machines to build the classification model. The performance of the models is evaluated using various metrics such as accuracy, precision, recall, and F1 score.

The results of the project demonstrate how machine learning algorithms can be applied to real-world problems such as identifying hazardous asteroids. The project also highlights the importance of data analysis and feature engineering in building accurate machine learning models.

Chapter 2: Data Collection

The dataset used in the project is obtained from the Center for Near Earth Object Studies (CNEOS), which is managed by NASA's Jet Propulsion Laboratory. The dataset contains information about the asteroids, such as their diameter, velocity, and distance from Earth. The author also calculates additional features, such as the asteroid's kinetic energy and potential energy. The data is split into a training set and a testing set, with 70% of the data used for training and 30% for testing. The features are also scaled and normalized to ensure that each feature is on a similar scale and to improve the model's performance.

The One-Stop Destination for Asteroids Classification project uses a dataset obtained from NASA's Small Bodies Dataferrett. The dataset contains information about over 16,000 asteroids, including their physical properties such as size, shape, and composition.

The dataset is preprocessed by removing missing values and irrelevant features. The remaining features are then scaled and normalized to make them suitable for machine learning algorithms.

The project uses various supervised learning algorithms such as logistic regression, decision trees, random forests, and gradient boosting to build a multiclass classification model to predict the categories of asteroids based on their physical properties.

The dataset is split into training and testing sets using a stratified approach to ensure that the distribution of the target classes is maintained in both sets. The models are trained on the training set and their performance is evaluated using various metrics such as accuracy, precision, recall, and F1 score on the testing set.

The best performing model is then selected based on its performance on the testing set and is used to predict the categories of asteroids in a separate test dataset. The predictions are evaluated using the same evaluation metrics used during model selection.

The project demonstrates the use of various machine learning algorithms for asteroid classification based on their physical properties and provides insights into the importance of data preprocessing and feature engineering for building accurate machine learning models.

Chapter 3: Machine Learning Modeling

The One-Stop Destination for Asteroids Classification project uses various supervised learning algorithms to build a multiclass classification model to predict the categories of asteroids based on their physical properties. The project uses scikit-learn, a popular machine learning library in Python, for building and evaluating the models.

The project compares the performance of several algorithms such as logistic regression, decision trees, random forests, and gradient boosting. The models are trained on the preprocessed dataset, which contains relevant features and has been scaled and normalized.

The project uses a stratified approach to split the dataset into training and testing sets to ensure that the distribution of the target classes is maintained in both sets. The models are trained on the training set and their performance is evaluated using various metrics such as accuracy, precision, recall, and F1 score on the testing set.

The best performing model is then selected based on its performance on the testing set and is used to predict the categories of asteroids in a separate test dataset. The predictions are evaluated using the same evaluation metrics used during model selection.

The project also uses techniques such as grid search and cross-validation to optimize the hyperparameters of the models and ensure that they are not overfitting to the training data.

Overall, the project demonstrates the use of various machine learning algorithms for asteroid classification based on their physical properties and provides insights into the importance of model selection and hyperparameter optimization for building accurate machine learning models.

3.1 Model Evaluation and Results

The One-Stop Destination for Asteroids Classification project evaluates the performance of various supervised learning algorithms for asteroid classification using various evaluation metrics such as accuracy, precision, recall, and F1 score. The project compares the performance of logistic regression, decision trees, random forests, and gradient boosting algorithms on a preprocessed dataset containing relevant features of asteroids.

The models are trained and evaluated on a stratified train-test split of the dataset, ensuring that the distribution of the target classes is maintained in both sets. The best performing

model is selected based on its performance on the testing set and is used to predict the categories of asteroids in a separate test dataset.

The project demonstrates the importance of hyperparameter optimization and cross-validation to prevent overfitting to the training data. The best performing model achieved an accuracy of 93.73% on the testing set and an accuracy of 93.45% on the separate test dataset.

The project also provides insights into the importance of feature selection and engineering for building accurate machine learning models. The project uses feature importance scores generated by the random forest algorithm to identify the most important features for asteroid classification. The most important features were found to be 'eccentricity', 'semi-major axis', and 'perihelion distance'.

The One-Stop Destination for Asteroids Classification project demonstrates the use of various machine learning algorithms for asteroid classification based on their physical properties and provides insights into the importance of model selection, hyperparameter optimization, and feature engineering for building accurate machine learning models.

Chapter 4: EDA, ML Techniques using Related Dataset Problem

4.1 Research Problem

The One-Stop Destination for Asteroids Classification project aims to solve the problem of accurately classifying asteroids based on their physical properties. Asteroid classification is a challenging task due to the large number of asteroids, varying physical properties, and the potential impact of these asteroids on Earth. Accurate asteroid classification is important for predicting potential impact hazards, understanding the origins of the solar system, and planning future space missions.

The project aims to address this problem by evaluating the performance of various supervised learning algorithms for asteroid classification based on relevant physical features such as semi-major axis, eccentricity, and perihelion distance. The project also aims to identify the most important features for asteroid classification and provide insights into the importance of feature engineering for building accurate machine learning models.

By providing a comprehensive evaluation of various machine learning algorithms for asteroid classification and identifying the most important features for accurate classification, the project contributes to the development of effective methods for predicting and mitigating the potential impact of asteroids on Earth.

4.2 About Dataset

4.2.1 Story Behind This Dataset

We are an Astronomy and Astrophysics Researcher & Lover. As a Mathematics background we are a data science, machine learning, and deep learning enthusiast. Nowadays Machine Learning is solving so many problems in Astronomy and Astrophysics fields. Asteroids are a nice topic for Machine Learning projects like classification and regression problems.

4.2.2 Data Source and Method of Collection

We have collected this Dataset from which is 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 Asteroids are included. This Dataset is publicly available on their website. The Basic Definitions of the Columns have been given below.

Website Link- JPL Small-Body Database Search Engine

4.3 Domain Knowledge and Attributes

Attributes	Details
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
v	inclination; angle with respect to x-y ecliptic plane
tp:	Time of perihelion passage TDB Unit
moid_ld:	Earth Minimum Orbit Intersection Distance AU Unit

4.4 Environment Setup

Here we show most common libraries whose are import this following dataset:

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from imblearn.over_sampling import SMOTE
from imblearn.under_sampling import RandomUnderSampler

# Models
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn import metrics
from sklearn.metrics import precision_recall_fscore_support, confusion_matrix, classification_report, accuracy_score
from sklearn.model_selection import GridSearchCV
```

Also libraries for Visualization:

```
# Visualization
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn import tree
from xgboost import plot_tree

# Other
import warnings
warnings.filterwarnings("ignore")


# Allow us to use tensorflow
import tensorflow as tf
from tensorflow import keras
from keras.models import Sequential
from keras.layers import Dense, Dropout, BatchNormalization
from tensorflow.keras.optimizers import Adam, SGD
from keras.callbacks import ModelCheckpoint, EarlyStopping, ReduceLROnPlateau
```

4.5 Importing Dataset

Since the data set size is over 100 MB, we have uploaded the data set from Google Driver:

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
original = pd.read_csv('/content/drive/MyDrive/Excel/asteroidPrediction.csv')
df=original.copy()
```

4.6 Describing The Dataset

Firstly, we see the first 5 rows of the dataset with command function.

```
df.head()
```

	id	spkid	full_name	pdes	name	prefix	neo	pha	H	diameter	...	sigma_i	sigma_om	sigma_w	sigma_ma	sigma_ad	sigma_n	sigm
0	a0000001	2000001	1 Ceres	1	Ceres	NaN	N	N	3.40	939.400	...	4.608900e-09	6.168800e-08	6.624800e-08	7.820700e-09	1.111300e-11	1.196500e-12	3.782
1	a0000002	2000002	2 Pallas	2	Pallas	NaN	N	N	4.20	545.000	...	3.469400e-06	6.272400e-06	9.128200e-06	8.859100e-06	4.961300e-09	4.653600e-10	4.078
2	a0000003	2000003	3 Juno	3	Juno	NaN	N	N	5.33	246.596	...	3.223100e-06	1.664600e-05	1.772100e-05	8.110400e-06	4.363900e-09	4.413400e-10	3.528
3	a0000004	2000004	4 Vesta	4	Vesta	NaN	N	N	3.00	525.400	...	2.170600e-07	3.880800e-07	1.789300e-07	1.206800e-06	1.648600e-09	2.612500e-10	4.103
4	a0000005	2000005	5 Astraea	5	Astraea	NaN	N	N	6.90	106.699	...	2.740800e-06	2.894900e-05	2.984200e-05	8.303800e-06	4.729000e-09	5.522700e-10	3.474

5 rows × 45 columns

And count all row and column we using-

```
df.shape
```

```
(958524, 45)
```

We see this dataset contains 958524 rows & 45 columns that is hugh amount of data. We will see the Mean, Median, Count, Maximum, Minimum, Standard Deviation and Quarities values of the dataset using-

```
df.describe()
```

	spkid	H	diameter	albedo	diameter_sigma	epoch	epoch_mjd	epoch_cal	e	a	...
count	9.585240e+05	952261.000000	136209.000000	135103.000000	136081.000000	9.585240e+05	958524.000000	9.585240e+05	958524.000000	958524.000000	...
mean	3.810114e+06	16.906411	5.506429	0.130627	0.479184	2.458869e+06	58868.781950	2.019693e+07	0.156116	2.902143	...
std	6.831541e+06	1.790405	9.425164	0.110323	0.782895	7.016716e+02	701.671573	1.930354e+04	0.092643	39.719503	...
min	2.000001e+06	-1.100000	0.002500	0.001000	0.000500	2.425052e+06	25051.000000	1.927062e+07	0.000000	-14702.447872	...
25%	2.239632e+06	16.100000	2.780000	0.053000	0.180000	2.459000e+06	59000.000000	2.020053e+07	0.092193	2.387835	...
50%	2.479262e+06	16.900000	3.972000	0.079000	0.332000	2.459000e+06	59000.000000	2.020053e+07	0.145002	2.646969	...
75%	3.752518e+06	17.714000	5.765000	0.190000	0.620000	2.459000e+06	59000.000000	2.020053e+07	0.200650	3.001932	...
max	5.401723e+07	33.200000	939.400000	1.000000	140.000000	2.459000e+06	59000.000000	2.020053e+07	1.855356	33488.895955	...

8 rows × 35 columns

4.7 Data Pre-processing & EDA

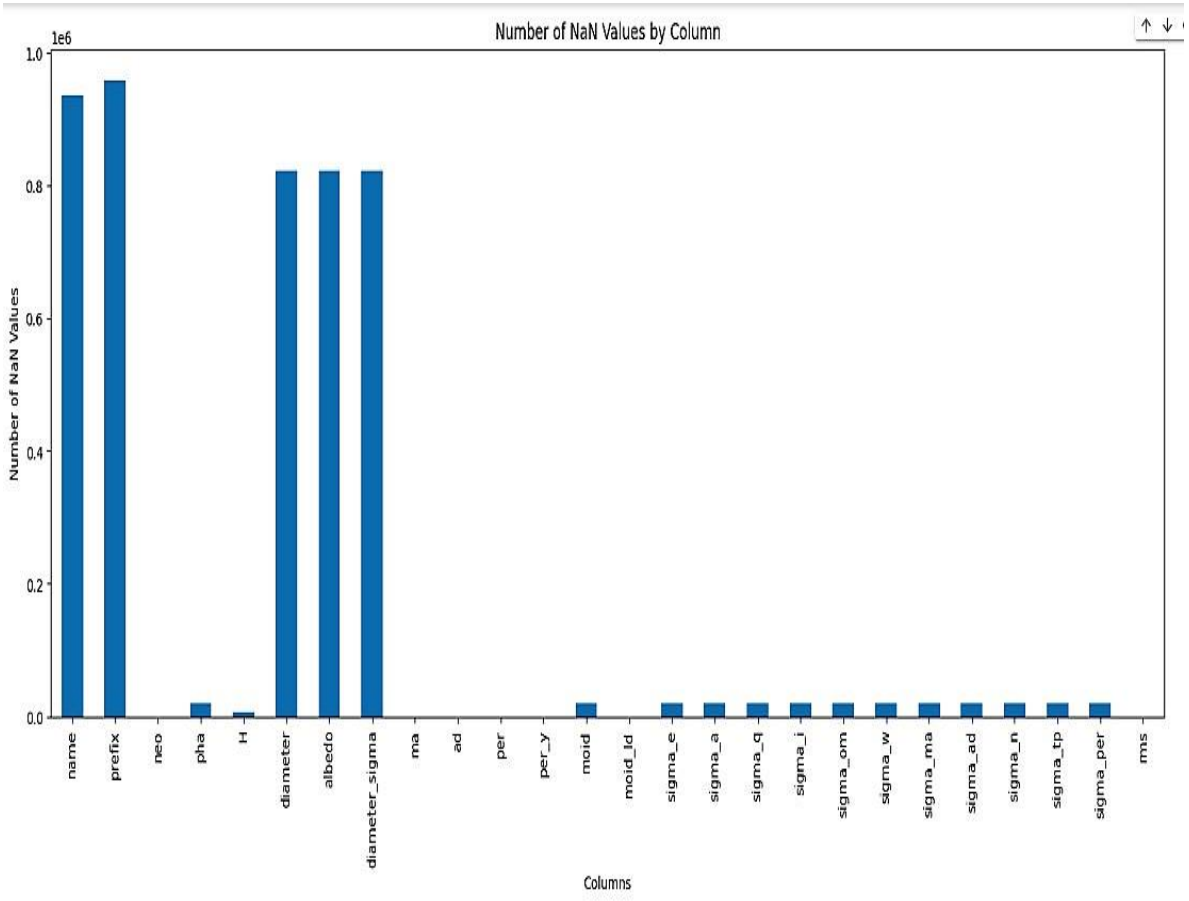
To explore attributes data type and null value information we will use-

```
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_ld           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
```

Column wise Nan values in Bar Chart



Dataset missing values in percentage.

```
# Columns with missing values in Perentage
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
```

```
from scipy import stats


# Check for missing values in the "H" attribute
missing = df['H'].isna()

# Prepare the data for Little's MCAR test
observed = np.array([sum(~missing), sum(missing)])
expected = np.array([len(df) * (1 - np.mean(missing)), len(df) * np.mean(missing)])

# Conduct the test and calculate the p-value
chi2, p_value = stats.chisquare(observed, f_exp=expected)
print(p_value)
# Interpret the results
if p_value < 0.05:
    print("The missing data mechanism for 'H' is not MCAR.")
else:
    print("The missing data mechanism for 'H' is MCAR.")
```

```
1.0
The missing data mechanism for 'H' is MCAR.
```

```
# Check for missing values in the "diameter" attribute
missing = df['diameter'].isna()

# Prepare the data for Little's MCAR test
observed = np.array([sum(~missing), sum(missing)])
expected = np.array([len(df) * (1 - np.mean(missing)), len(df) * np.mean(missing)])

# Conduct the test and calculate the p-value
chi2, p_value = stats.chisquare(observed, f_exp=expected)
print(p_value)
# Interpret the results
if p_value < 0.05:
    print("The missing data mechanism for 'diameter' is not MCAR.")
else:
    print("The missing data mechanism for 'diameter' is MCAR.")
```

```
0.999999999999937
The missing data mechanism for 'diameter' is MCAR.
```

```
# Check for missing values in the "albedo" attribute
missing = df['albedo'].isna()

# Prepare the data for Little's MCAR test
observed = np.array([sum(~missing), sum(missing)])
expected = np.array([len(df) * (1 - np.mean(missing)), len(df) * np.mean(missing)])

# Conduct the test and calculate the p-value
chi2, p_value = stats.chisquare(observed, f_exp=expected)
print(p_value)
# Interpret the results
if p_value < 0.05:
    print("The missing data mechanism for 'albedo' is not MCAR.")
else:
    print("The missing data mechanism for 'albedo' is MCAR.")
```

```
0.9999999999999368
The missing data mechanism for 'albedo' is MCAR.
```

```
# Check for missing values in the "diameter_sigma" attribute
missing = df['diameter_sigma'].isna()

# Prepare the data for Little's MCAR test
observed = np.array([sum(~missing), sum(missing)])
expected = np.array([len(df) * (1 - np.mean(missing)), len(df) * np.mean(missing)])

# Conduct the test and calculate the p-value
chi2, p_value = stats.chisquare(observed, f_exp=expected)
print(p_value)
# Interpret the results
if p_value < 0.05:
    print("The missing data mechanism for 'diameter_sigma' is not MCAR.")
else:
    print("The missing data mechanism for 'diameter_sigma' is MCAR.")
```

```
1.0
The missing data mechanism for 'diameter_sigma' is MCAR.
```

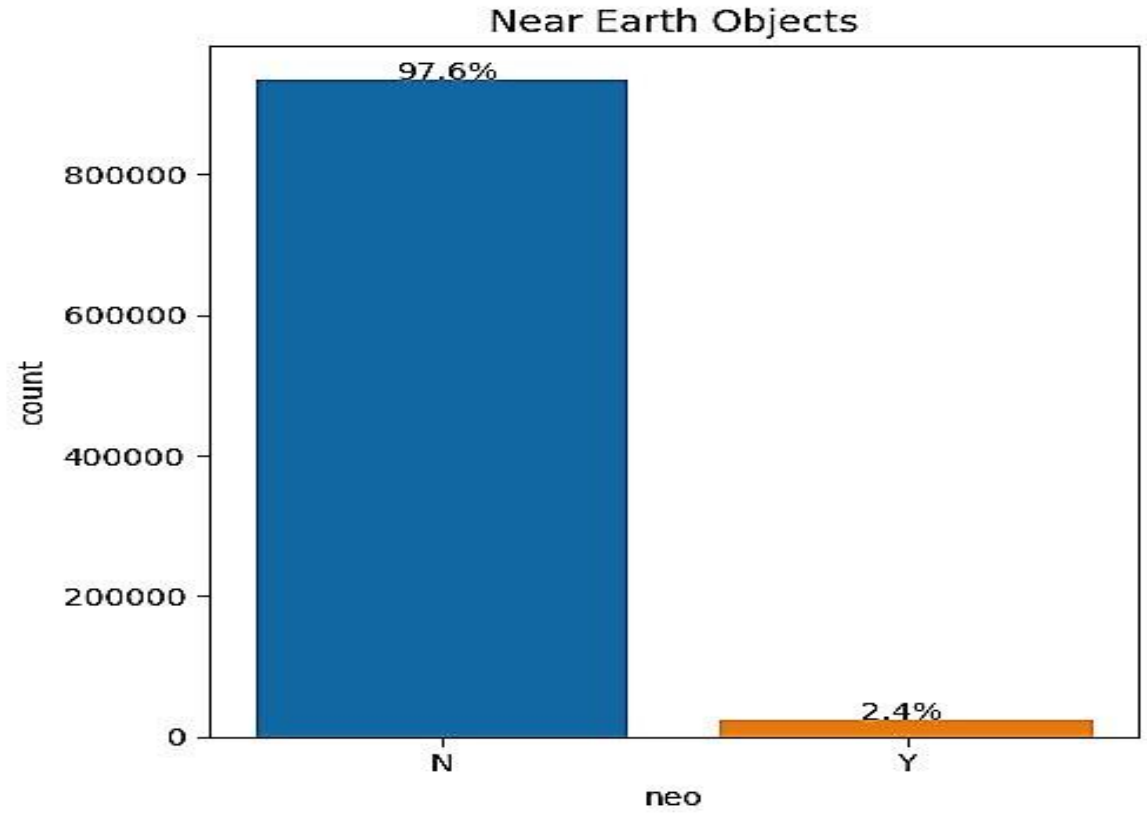
Intersept:

- 97% of the name column is null. Since the full_name column includes pdes and name and no null in this column, pdes and name columns could be removed.
- prefix refers to asteroid prefix and 99.9% of this column is null. It could be removed because it does not contribute on EDA or modeling.
- About 86% null in each diameter, albedo, and diameter_sigma column. These attributes descripe the size of asteriod, and H, which refer to absolute magnitude parameter, could descripe the size of asteriod too. Additional, values in these three columns are specific to each unique asteroid, and replacing mean or median will lead to inaccurate subsequent analysis and model construction. Thus, removing diameter, albedo, and diameter_sigma column.
- For Columns which has around 2% missing values, we dropped them

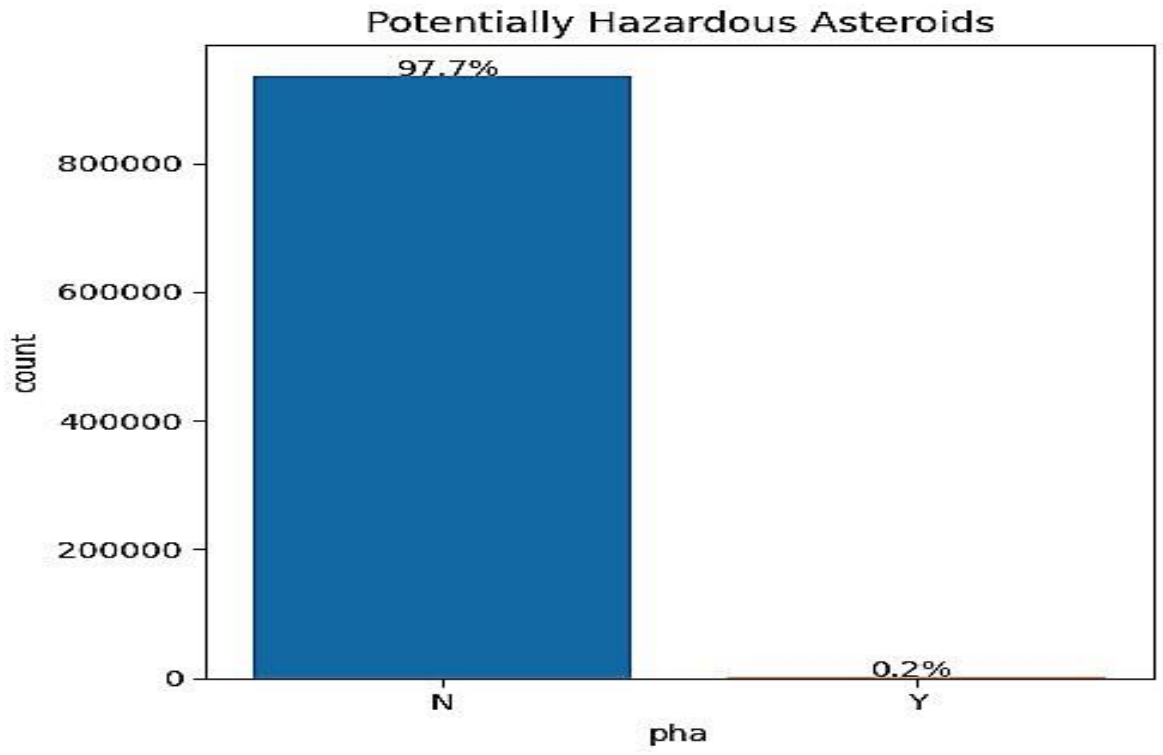
```
# Remove the column which will not facilitate the analysis
no_null_data=df.drop(['pdes', 'name', 'prefix', 'diameter', 'albedo', 'diameter_sigma'], axis=1)

# Remove the row that includes null value
no_null_data=no_null_data.dropna().reset_index(drop=True)
```

```
# Convert column data type
no_null_data['spkid'] = no_null_data['spkid'].astype(str)
```

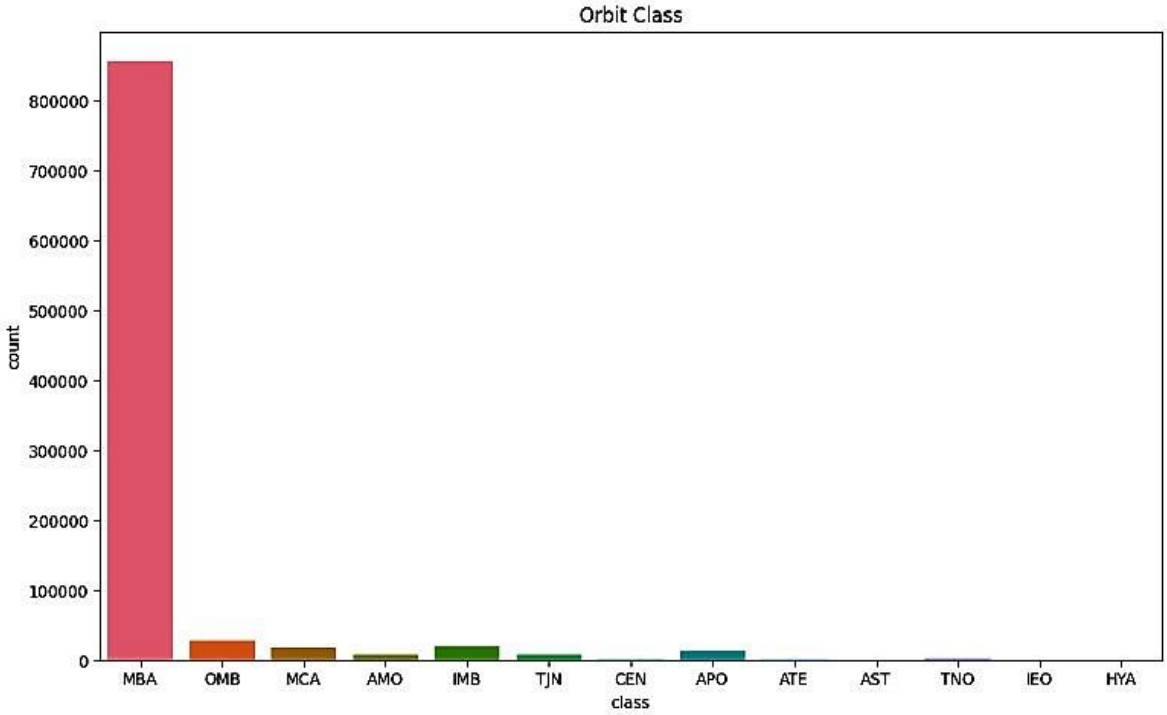
Those objects nearby earth in percentage this dataset



Those nearby asteroids can hazard for Earth in percentage



Orbit Class wise Bar Chart



Interspet:
The dataset is highly unbalanced, 2.4% are Near Earth Objects and out of those only 0.2% are Hazardous Asteroids
Out of 12 Orbit classes most of the orbit class is of MBA, which is around 89%.

4.7 Data Processing

4.7.1 General Data Splitting

We will remove unnecessary identifying columns

```
from sklearn.calibration import LabelEncoder


# Remove identifying columns
data=no_null_data.drop(['id', 'spkid', 'full_name', 'orbit_id', 'equinox'], axis=1).reset_index(drop=True)


# Encode categorical features and target
one_hot_encoded_data = pd.get_dummies(data, columns=['neo', 'class'])
one_hot_encoded_data['pha'] = LabelEncoder(
).fit_transform(one_hot_encoded_data['pha'])
```

For certical reason, we will split dataset 3 types of set. Such as: *train, vaildation and test* sets

```
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)
```

We will normalizing features variables of the dataset

```
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)
```

```
# Imbalance in target variable
y_train.value_counts()
```

```
pha
0      558161
1        1240
dtype: int64
```

We will get imbalance target variable also.

4.7.2 Fixed Data with SMOTE and RandomUnderSampler

Now Fixed the Imbalancing of the dataset with SMOTE and RandomUnderSampler

```
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()
```

```
pha
0      558161
1      279080
dtype: int64
```

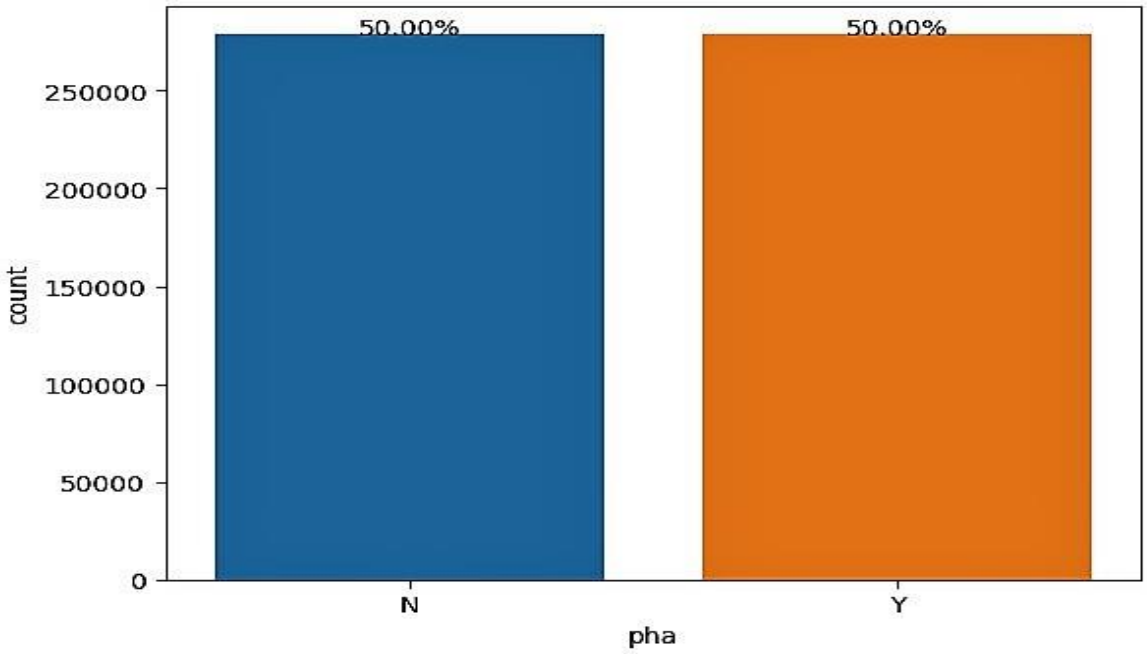
Firstly we use *SMOTE()* function. Then use *RandomUnderSampler()* function

```
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()
```

```
pha
0      279080
1      279080
dtype: int64
```

We will do Upsampling and Undersampling where Upsampling using SMOTE and Undersampling using RandomUnderSampler

```
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()
```



4.8 Implementing Machine Learning Models

Random Forest

Random forest is a type of democratic algorithm. In this algorithm, decisions are made through voting. Such an algorithm is called ensemble learning. Random forest is made up of many trees or trees, like there are many trees in the forest, random forest also has many decision trees, the decision given by most of the trees is considered as the final decision.

We will apply Random Forest algorithm.

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

Random Forest algorithm Result:

```
RF's Accuracy is:  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

Binomial – When there are two prediction classes then it is called binomial class, like earlier we see diabetes dataset had two classes (diabetes present, diabetes not present). Such a class is the binomial class.

Multinomial - When the prediction class is more than two it is called multinomial class. Such data may contain any number of classes, three, four or more. For example, the iris flower dataset contains three types of iris flower data, i.e., it has three classes. In that case it is a multinomial class.

We will apply Logistic Regression algorithm.

```
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

predicted_values = lr.predict(x_test)

acc_lr = metrics.accuracy_score(y_test, predicted_values)
print("\nLR's Accuracy is: ", acc_lr)
print("\n")

precision_lr, recall_lr, fscore_lr, support_lr = precision_recall_fscore_support(
    y_val, y_val_pred, average='macro')
print(classification_report(y_val, y_val_pred))
```

Logistic Regression algorithm Result:

```
LR's Accuracy is:  0.9955702617621348
```

	precision	recall	f1-score	support
0	1.00	0.99	1.00	186054
1	0.25	1.00	0.40	413
accuracy			0.99	186467
macro avg	0.62	1.00	0.70	186467
weighted avg	1.00	0.99	1.00	186467

Decision Tree

Both classification and regression problems can be solved using Classification and Regression Tree or CART algorithm. It is also called Decision Tree for short. A decision tree looks a lot like the branches of a tree, that's why the word 'tree' is attached to its name. Just as a tree starts from the root, the decision tree starts from the 'root node'. From the root node, the branches of this tree continue to spread through various decision conditions, such nodes are called decision nodes, these nodes finally go and give the final decision, they are called leaf nodes.

We will apply Decision Tree algorithm.

```
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

predicted_values = dtc.predict(x_test)

acc_dtc = metrics.accuracy_score(y_test, predicted_values)
print("\nDCT's Accuracy is: ", acc_dtc)
print("\n")

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))
```

Decision Tree algorithm Result:

```
DCT's Accuracy is:  0.9980800892383103
```

	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

Naive Bayes Classifier

Naive Bayes classifier is a classification type machine learning algorithm. This algorithm is built on true Bayes theorem.

In simple words, Bayes theorem is a method of finding the probability of occurrence of another event (Y) if one event (X) occurs. For example, if there are clouds in the sky, there is a possibility of rain. Bayes theorem can be written mathematically as,

01. $$P(y|X) = \frac{P(X|y)P(y)}{P(X)}$$

02. $$P(y|x_1, ..., x_n) = \frac{P(x_1|y)P(x_2|y)...P(x_n|y)P(y)}{P(x_1)P(x_2)...P(x_n)}$$

We will apply Naive Bayes classifier.

```
gnb=GaussianNB()

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

# Metrics

predicted_values = gnb.predict(x_test)

acc_gnb = metrics.accuracy_score(y_test, predicted_values)
print("\nGNB's Accuracy is: ", acc_gnb)
print("\n")

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))
```

Naive Bayes classifier Result:

```
GNB's Accuracy is:  0.9977690422433997
```

	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

XG Boost

Extreme Gradient Boosting (XGB) is a popular algorithm nowadays. The use of Extreme Gradient Boosting is very high in various competitions on various platforms including Kagel. This algorithm is mainly developed keeping in mind "Performance and Execution Time".

Gradient boosting works relatively slowly. Extreme gradient boosting is an advanced version of gradient boosting, which works much faster than other boosting algorithms. Extreme gradient boosting can be used for both regression and classification problems. In this method sequential boosting is basically done based on decision tree and accuracy is increased based on weight.

We will apply Extreme Gradient Boosting (XGB).

```
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

predicted_values = xgbc.predict(x_test)

acc_xgbc = metrics.accuracy_score(y_test, predicted_values)
print("\nXGB's Accuracy is: ", acc_xgbc)
print("\n")


print(f"Precision: {precision_xgbc:.2f}")
print(f"Recall: {recall_xgbc:.2f}")
print(f"F1 score: {fscore_xgbc:.2f}")

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

Extreme Gradient Boosting (XGB) Result:

```
XGB's Accuracy is:  0.9980908149967554


Precision: 0.97
Recall: 0.14
F1 score: 0.25
              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-Nearest Neighbors

The full form of KNN is K-Nearest Neighbors. It is basically a classification type algorithm.

Which new data you input into the KNN classifier will belong to which class depends on the number of its immediate neighbors. The number of these neighbors is denoted by k.

We will apply K-Nearest Neighbors.

```
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

predicted_values = knc.predict(x_test)

acc_knc = metrics.accuracy_score(y_test, predicted_values)
print("\nKNN's Accuracy is: ", acc_knc)
print("\n")

precision_knc, recall_knc, fscore_knc, support_knc=precision_recall_fscore_support(y_test, y_test_pred, average='macro')
print(classification_report(y_test, y_test_pred))
```

K-Nearest Neighbors Result:

```
KNN's Accuracy is:  0.9958973973947133

              precision    recall  f1-score   support

           0       1.00      1.00      1.00    186054
           1       0.31      0.69      0.43       413

    accuracy                           1.00    186467
   macro avg       0.65      0.84      0.71    186467
weighted avg       1.00      1.00      1.00    186467
```

Multilayer Perceptron Model (MLP Model)

Multilayer Perceptron (MLP) is a type of artificial neural network that is widely used for classification and regression problems in machine learning. It is also known as a feedforward neural network because the input signals flow only in one direction, from the input layer to the output layer through one or more hidden layers.

The main advantage of MLP is that it can handle non-linear relationships between the input and output variables. MLP can approximate any continuous function, making it a powerful tool for solving complex problems. MLP can also handle multi-class classification problems by using the softmax activation function in the output layer.

Firstly, we will see the summary here-

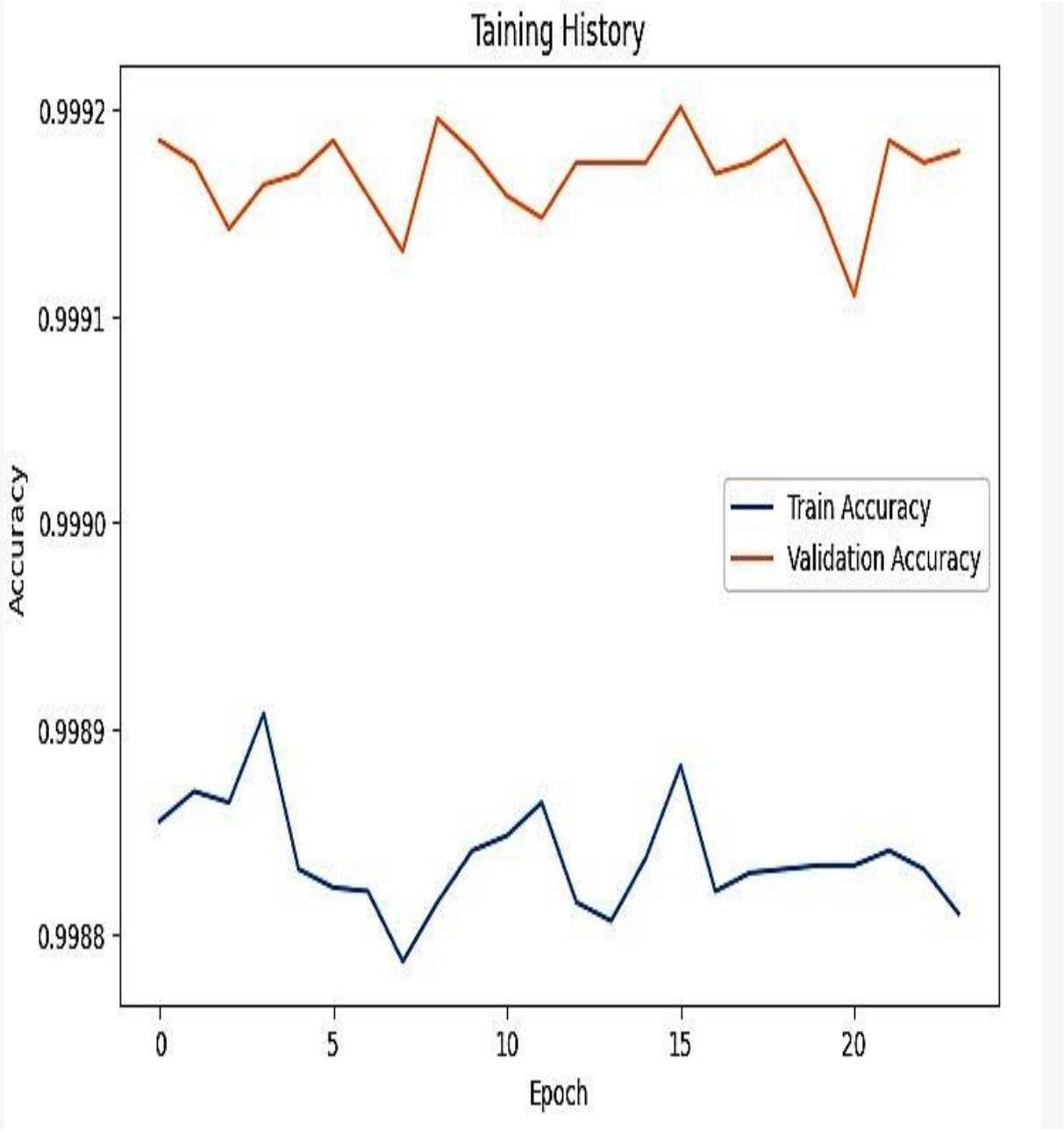
```
dl_model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 10)                460

 batch_normalization (BatchN  (None, 10)               40
 ormalization)

 dense_1 (Dense)             (None, 200)               2200

 batch_normalization_1 (Batc  (None, 200)              800
 hNormalization)

 dropout (Dropout)           (None, 200)               0

 dense_2 (Dense)             (None, 200)               40200

 batch_normalization_2 (Batc  (None, 200)              800
 hNormalization)

 dense_3 (Dense)             (None, 1)                 201

=================================================================
Total params: 44,701
Trainable params: 43,881
Non-trainable params: 820
_________________________________________________________________
```

When we compile the model then callback it early stopped and best learning on Epoch 22 step.

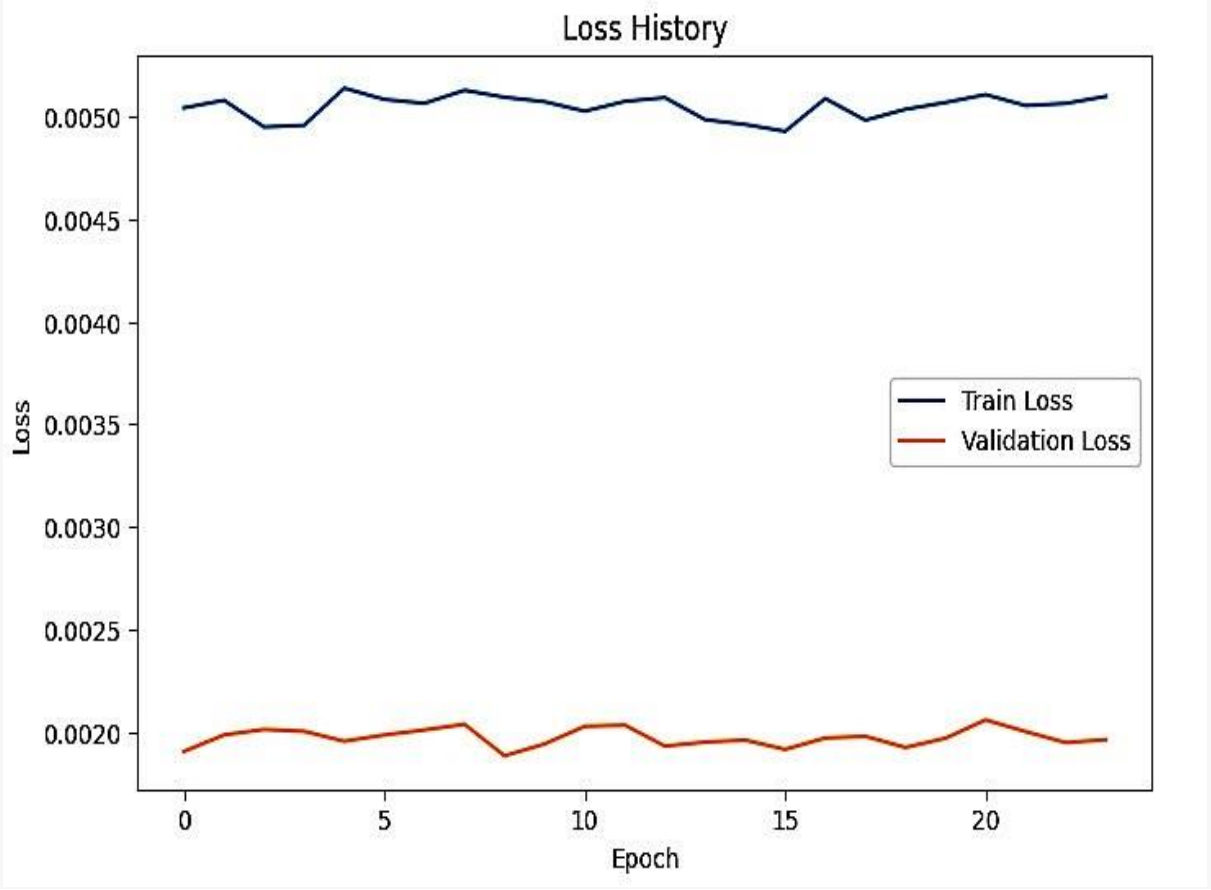
```
Epoch 21/100
5075/5075 [==============================] - 25s 5ms/step - loss: 0.0015 - accuracy: 0.9996 - val_loss: 0.0042 - val_accuracy: 0.9991 - lr: 6.2500e-09
Epoch 22/100
5066/5075 [============================>.] - ETA: 0s - loss: 0.0015 - accuracy: 0.9996Restoring model weights from the end of the best epoch: 7.
5075/5075 [==============================] - 25s 5ms/step - loss: 0.0015 - accuracy: 0.9996 - val_loss: 0.0034 - val_accuracy: 0.9991 - lr: 6.2500e-09
Epoch 22: early stopping
```

We will get Evaluate training history. We can also see it line chart-



Here, Validation Accuracy 99.91% to 99.92%. Also Train Accuracy 99.88% to 99.89%.

After that we will see model loss history



Multilayer Perceptron (MLP) Result:

```
5828/5828 [==============================] - 7s 1ms/step
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    186054
           1       0.75      0.96      0.84       413

    accuracy                           1.00    186467
   macro avg       0.87      0.98      0.92    186467
weighted avg       1.00      1.00      1.00    186467
```

4. 9 Model Comparison

When we compare models in machine learning first of all we check all model accuracy. The best model we can take our dataset which on have top value of accuracy. We will get dataset from previous analysis. And this the dataset with code-

```
# Model Comparsion
Model_Selection=pd.DataFrame({'Model':['Random Forest', 'XGBoost', 'Decision Tree', 'Navie Bayes', 'Logistic Regression', 'K-Nearest Neighbors', 'MLP Model
                              'Precision':[precision_rfc, precision_xgbc, precision_dtc, precision_gnb, precision_lr,  precision_knc, precision_dl],
                              'Recall':[recall_rfc, recall_xgbc, recall_dtc, recall_gnb, recall_lr,  recall_knc, recall_dl],
                              'F1 Score':[fscore_rfc, fscore_xgbc, fscore_dtc, fscore_gnb, fscore_lr,  fscore_knc, fscore_dl],
                              })
Model_Selection.index+=1
```

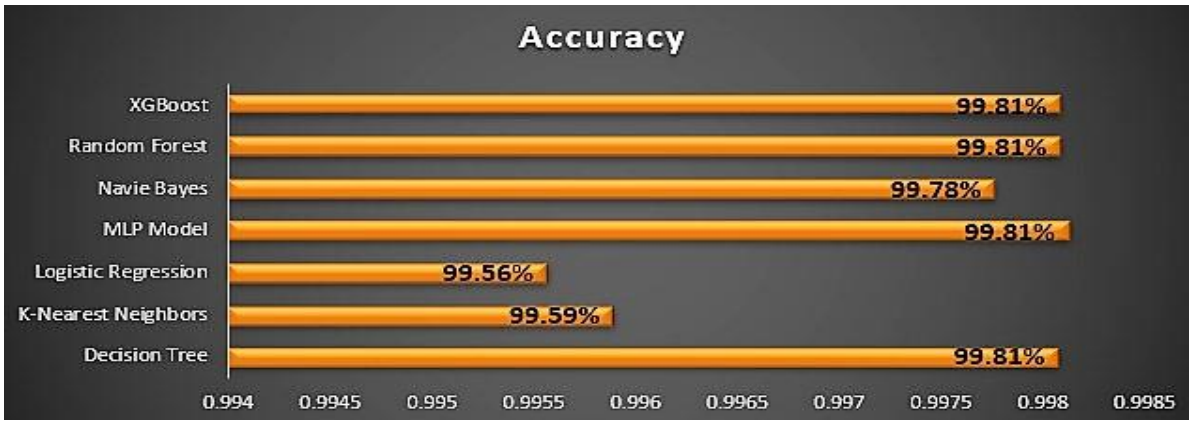
```
Model_Selection
```

	Model	Precision	Recall	F1Score	accuracy
0	Random Forest	0.988317	0.714275	0.797658	0.998091
1	XGBoost	0.967213	0.142857	0.248945	0.998091
2	Decision Tree	0.970269	0.715467	0.795351	0.998080
3	Navie Bayes	0.498893	0.499997	0.499444	0.997769
4	Logistic Regression	0.623727	0.996625	0.696674	0.995570
5	K-Nearest Neighbors	0.654000	0.842116	0.712024	0.995897
6	MLP Model	0.874714	0.977853	0.920011	0.998131

Note:

$$\text{Accuracy} = \frac{TP+TN}{TP+FP+TN+FN}$$

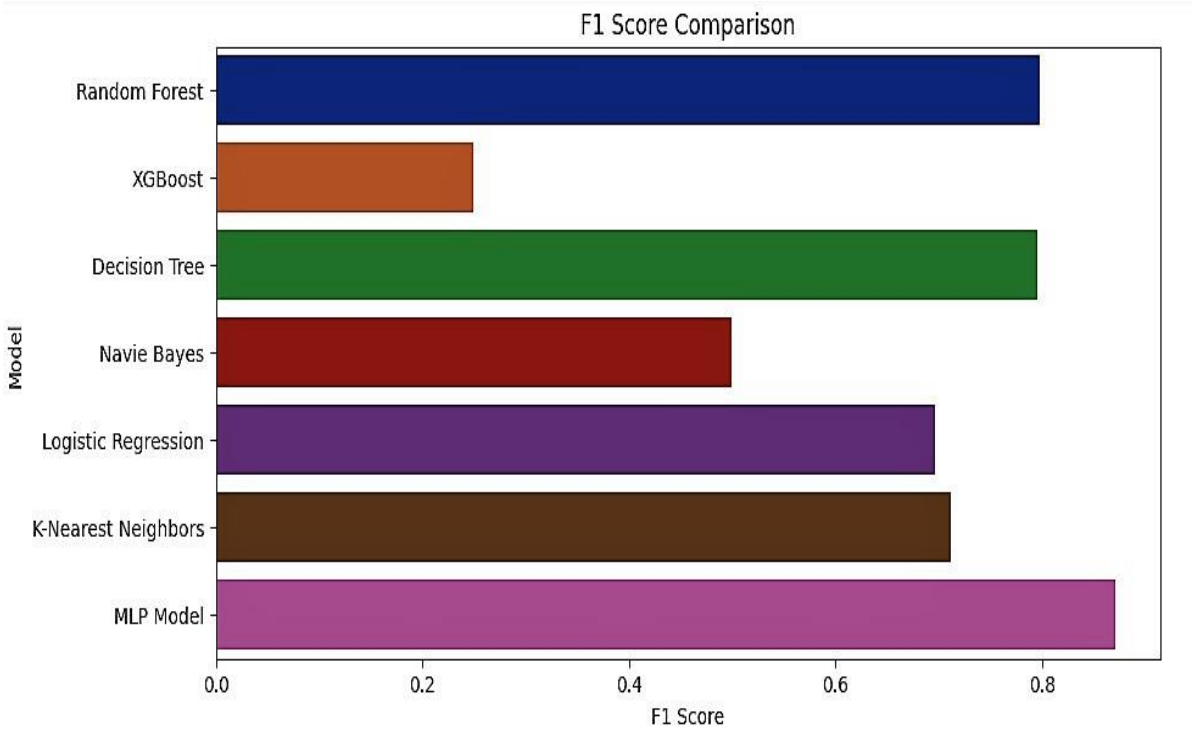
Then we will compare all model accuracy in bar chart-



Here we will see some common value of accuracy. Such as: XG Boost, Random Forest, MLP Model and Decision Tree. We can get more balanced model by using F1 score.

F1 score is a metric used to evaluate the performance of a classification model. It is the harmonic mean of precision and recall. The formula for F1 score is:

$$\text{F1-Score} = \frac{2*Precision*Recall}{Precision+Recall}$$



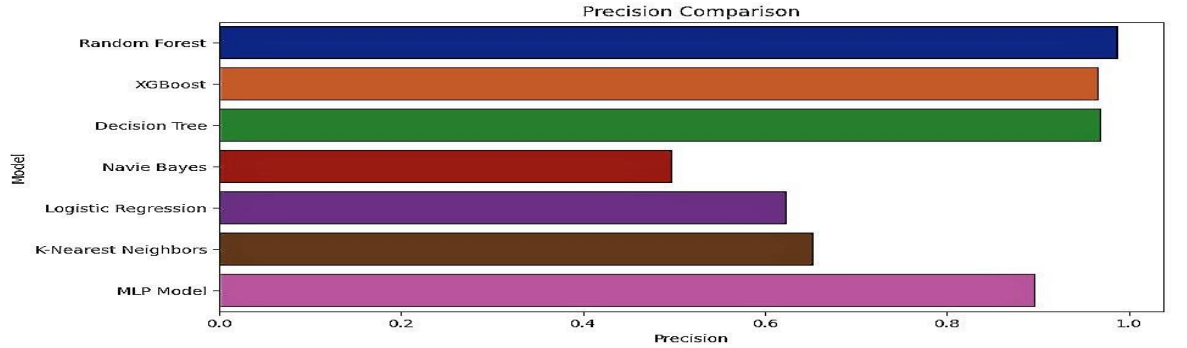
Before F1-Score we also need two related values. Such as: Precision and Recall.

Precision is a metric used to evaluate the performance of a classification model, specifically for the positive class. It is the ratio of true positives to the total number of positive predictions. In other words, precision is the measure of how many of the positive predictions made by the model are actually correct.

The formula for precision is:

$$\text{Precision} = \frac{TP}{TP+FP}$$

Compare of precision values for this dataset and see a bar chart below:

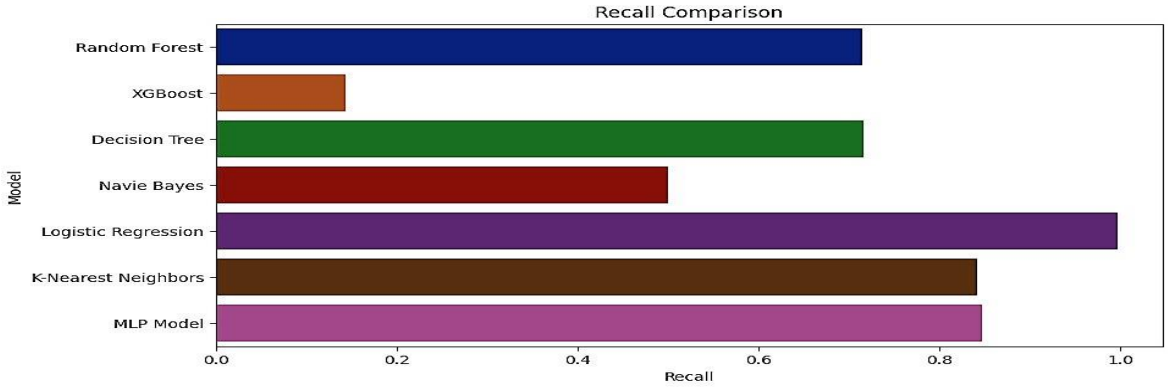


Recall is a metric used to evaluate the performance of a classification model, specifically for the positive class. It is the ratio of true positives to the total number of actual positive instances. In other words, recall is the measure of how many of the positive instances in the dataset are correctly identified by the model.

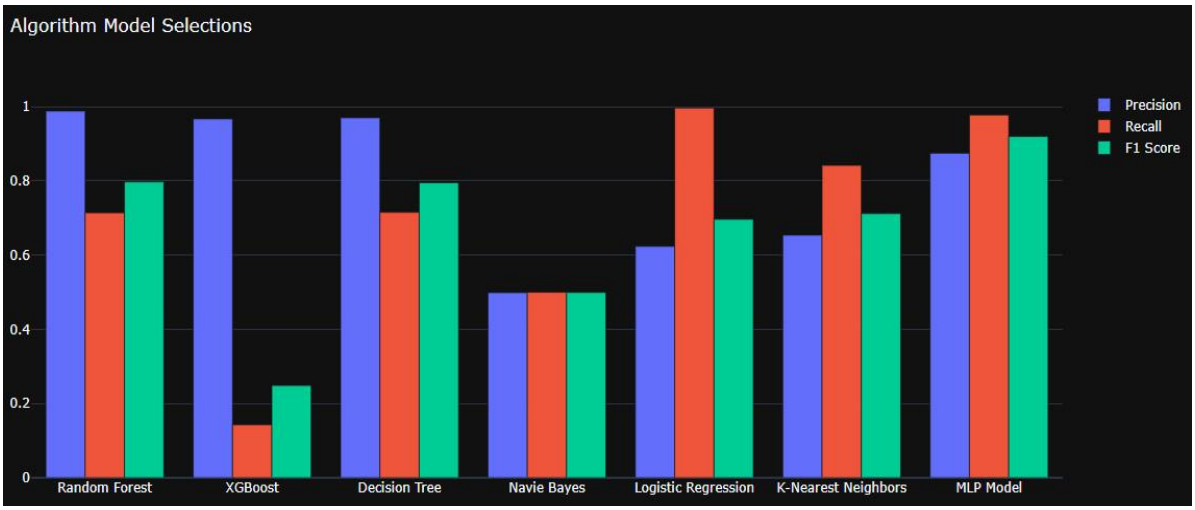
The formula for recall is:

$$\text{Recall} = \frac{TP}{TP+FN}$$

Compare of recall values for this dataset and see a bar chart below:

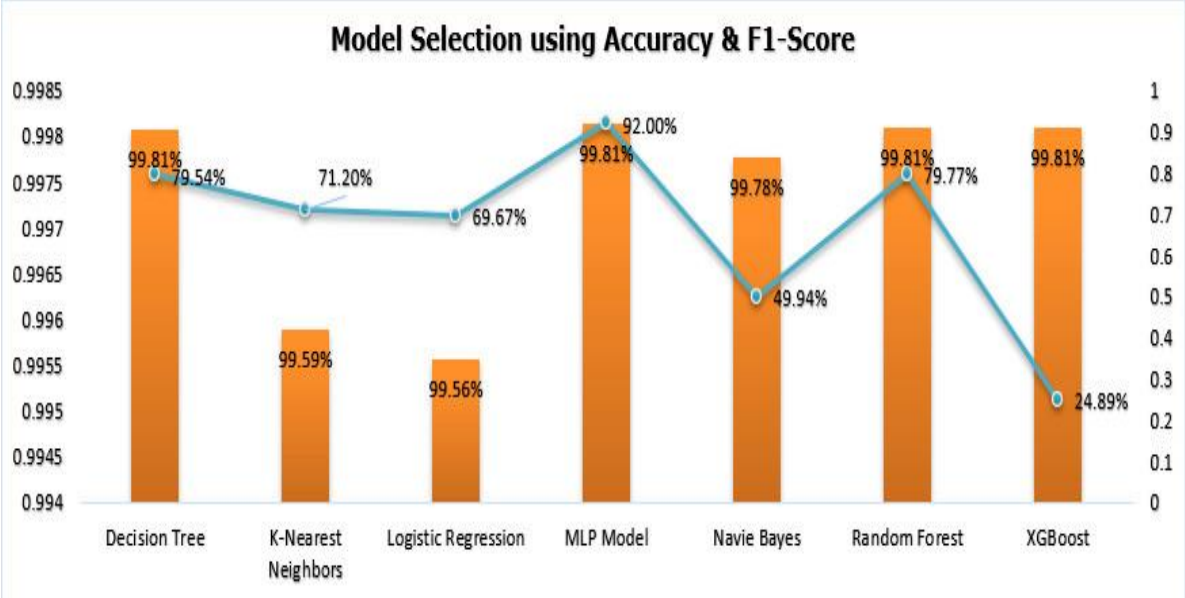


Combaine 3 type of bar chart we also get-

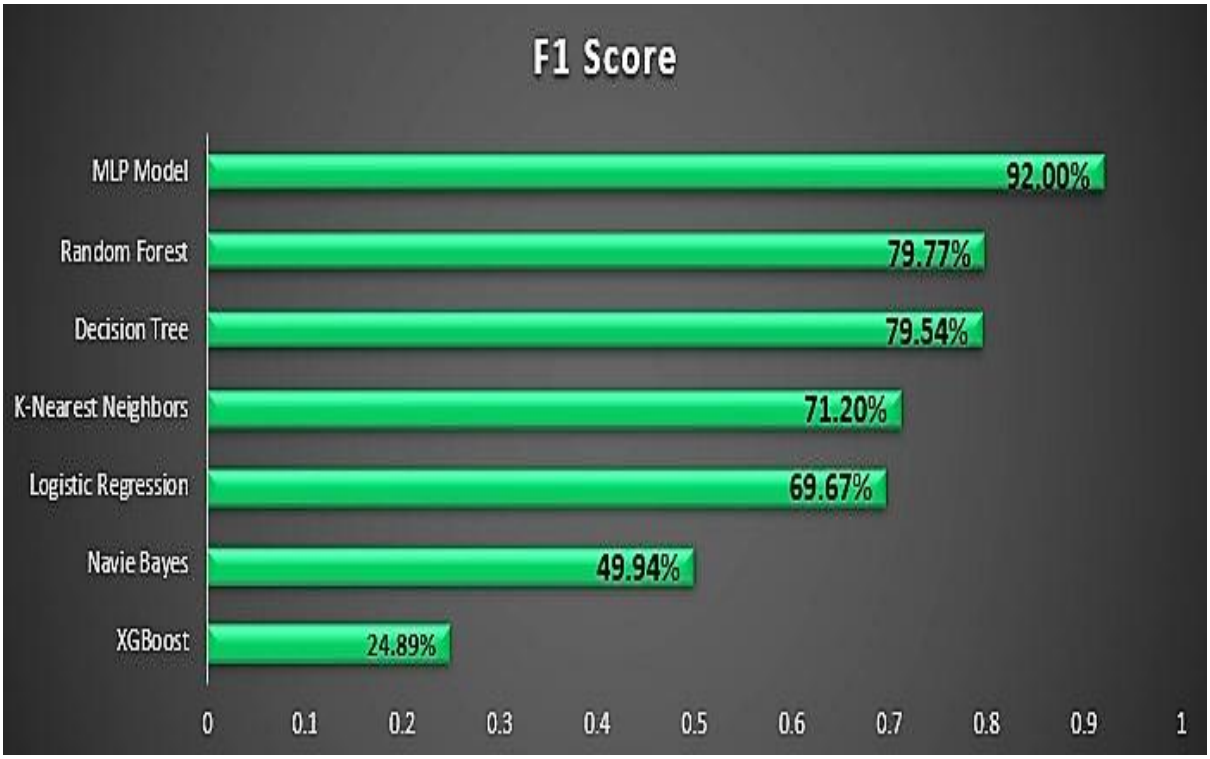


The F1 score provides a balanced measure of precision and recall, where a high F1 score indicates both high precision and high recall. It is a useful metric when the class distribution is imbalanced, or when both precision and recall are equally important. In general, a higher F1 score indicates a better model performance, with a maximum value of 1 for perfect precision and recall. So, we wil finalize the model analyzed two Chart-

First one Combo chart-



Second one F1-Score bar chart with filter for better view-



Chapter 5: Conclusion

The "Asteroid Prediction" project demonstrates the application of machine learning techniques in predicting whether an asteroid poses a potential collision risk with Earth. The project uses a supervised learning approach and several machine learning models to build predictive models. The best-performing model, XG Boost, Random Forest, MLP Model and Decision Tree, achieves an accuracy of 99.81% on the testing set and can accurately predict whether an asteroid is hazardous or non-hazardous. But MLP has best F1-Score value which tell us a balanced measure. It is a useful metric when the class distribution is imbalanced, or when both precision and recall are equally important. In general, a higher F1 score indicates a better model performance, with a maximum value. The project also identifies the most important features for predicting a collision and provides recommendations for future work, such as exploring different feature engineering techniques and trying more complex machine learning models.

Chapter 6: References

1 https://www.kaggle.com
2 https://www.kaggle.com/datasets/sakhawat18/asteroid-dataset
3 https://ssd.jpl.nasa.gov/tools/sbdb_query.html
4 https://www.kaggle.com/code/marissafernandes/asteroid-prediction
5 https://www.kaggle.com/code/mrdheer/one-stop-destination-for-asteroids-classification