CLOUD x LAB

Continuing Education Centre

NEAR EARTH OBJECTS CLASSIFICATION

Capstone project for,

Executive PG Certificate Program in Data Science by IIT Roorkee

By: Uday Marwah

Email: udaymarwah88@gmail.com

THE DATASET

- The dataset is of the NASA Near Earth Objects.
  - https://www.kaggle.com/datasets/sameepvani/nasa-nearest-earth-objects
- The dataset compiles the list of NASA certified asteroids that are classified as near earth objects.
- These near earth objects could potentially be harmful to the planet, thus we shall attempt to build a model that could classify these bodies as harmful or not based on some features that we will attempt to identify using data science techniques.

PRELIMINARY WORK AND GETTING THE DATA

We first get the data from the source at Kaggle into our working notebook

In [1]:

```
# This Python 3 environment comes with many helpful analytics libraries installed
# It is defined by the kaggle/python Docker image: https://github.com/kaggle/docker-python
# For example, here's several helpful packages to load

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

# Input data files are available in the read-only "../input/" directory
# For example, running this (by clicking run or pressing Shift+Enter) will list all files under the input directory

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

# You can write up to 20GB to the current directory (/kaggle/working/) that gets preserved as output when you create a ver
# You can also write temporary files to /kaggle/temp/, but they won't be saved outside of the current session
```

```
/kaggle/input/nasa-nearest-earth-objects/neo_v2.csv
/kaggle/input/nasa-nearest-earth-objects/neo.csv
```

We import some basic commonly used libraries and set paths to the data.

Importing the data as a pandas data frame and checking the same for what kinds of values are contained within.

In [2]:
```
1  df = pd.read_csv('../input/nasa-nearest-earth-objects/neo_v2.csv')
2  df    # checking out the data as a pandas DataFrame
```

Out[2]:

	id	name	est_diameter_min	est_diameter_max	relative_velocity	miss_distance	orbiting_body	sentry_object	absolute_magnitude	hazardous
0	2162635	162635 (2000 SS164)	1.198271	2.679415	13569.249224	5.483974e+07	Earth	False	16.73	False
1	2277475	277475 (2005 WK4)	0.265800	0.594347	73588.726663	6.143813e+07	Earth	False	20.00	True
2	2512244	512244 (2015 YE18)	0.722030	1.614507	114258.692129	4.979872e+07	Earth	False	17.83	False
3	3596030	(2012 BV13)	0.096506	0.215794	24764.303138	2.543497e+07	Earth	False	22.20	False
4	3667127	(2014 GE35)	0.255009	0.570217	42737.733765	4.627557e+07	Earth	False	20.09	True
...	...	...	...	...	...	...	...	...	...	...
90831	3763337	(2016 VX1)	0.026580	0.059435	52078.886692	1.230039e+07	Earth	False	25.00	False
90832	3837603	(2019 AD3)	0.016771	0.037501	46114.605073	5.432121e+07	Earth	False	26.00	False
90833	54017201	(2020 JP3)	0.031956	0.071456	7566.807732	2.840077e+07	Earth	False	24.60	False
90834	54115824	(2021 CN5)	0.007321	0.016370	69199.154484	6.869206e+07	Earth	False	27.80	False
90835	54205447	(2021 TW7)	0.039862	0.089133	27024.455553	5.977213e+07	Earth	False	24.12	False

90836 rows × 10 columns

Exploring the dataset

```
In [3]:  1  df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 90836 entries, 0 to 90835
Data columns (total 10 columns):
 #   Column              Non-Null Count  Dtype  
---  ------              --------------  -----  
 0   id                  90836 non-null  int64  
 1   name                90836 non-null  object 
 2   est_diameter_min    90836 non-null  float64
 3   est_diameter_max    90836 non-null  float64
 4   relative_velocity   90836 non-null  float64
 5   miss_distance       90836 non-null  float64
 6   orbiting_body       90836 non-null  object 
 7   sentry_object       90836 non-null  bool   
 8   absolute_magnitude  90836 non-null  float64
 9   hazardous           90836 non-null  bool   
dtypes: bool(2), float64(5), int64(1), object(2)
memory usage: 5.7+ MB
```

```
In [4]:  1  # checking for any null values
         2  df.isnull().sum()
```

```
Out[4]: id                  0
name                0
est_diameter_min    0
est_diameter_max    0
relative_velocity   0
miss_distance       0
orbiting_body       0
sentry_object       0
absolute_magnitude  0
hazardous           0
dtype: int64
```

The dataset has 90836 bodies, and 10 features about each body with 0 null values, hence our dataset is already pretty clean.

The features are, as follows:

- Id: unique identifier for each body
- Name: Name given by NASA to the body
- est_diameter_min: Minimum estimated diameter (in kilometers)
- est_diameter_max: Maximum estimated diameter (in kilometers)
- relative_velocity: Velocity of body, relative to Earth
- miss_distance: Distance in kilometers missed to the last near earth pass
- orbiting_body: Planet that the body orbits
- sentry_object: Boolean for whether the object is included in sentry (NASA's automated collision monitoring system)
- absolute_magnitude: Describes intrinsic luminosity
- Hazardous: Boolean feature that shows whether asteroid is harmful or not

Hence we shall have take hazardous as our target variable. Therefore our task is a simple binary classification of a near earth body as either hazardous or not.

PRE-PROCESSING THE DATA

- Now, we shall process the data to make it useable by our models.
- We decide to drop the following features:
  - Orbiting_body: We drop this column as the entire dataset is of objects orbiting the Earth and all values in this feature are Earth, so this doesn't influence our target variable.
  - sentry_object: We drop this column as it is a Boolean variable for whether the object is included in NASA's sentry monitoring system or not. Hence it is not relevant for our classification task.

In [5]:
```
# Dropping some columns
# orbiting_body; as it has only single value as entire dataset is of objects orbiting near earth only
# sentry_object; as it is irrelevant to our classification use case
df.drop(['orbiting_body', 'sentry_object'], axis=1)
```

Out[5]:

	id	name	est_diameter_min	est_diameter_max	relative_velocity	miss_distance	absolute_magnitude	hazardous
0	2162635	162635 (2000 SS164)	1.198271	2.679415	13569.249224	5.483974e+07	16.73	False
1	2277475	277475 (2005 WK4)	0.265800	0.594347	73588.726663	6.143813e+07	20.00	True
2	2512244	512244 (2015 YE18)	0.722030	1.614507	114258.692129	4.979872e+07	17.83	False
3	3596030	(2012 BV13)	0.096506	0.215794	24764.303138	2.543497e+07	22.20	False
4	3667127	(2014 GE35)	0.255009	0.570217	42737.733765	4.627557e+07	20.09	True
...	...	...	...	...	...	...	...	...
90831	3763337	(2016 VX1)	0.026580	0.059435	52078.886692	1.230039e+07	25.00	False
90832	3837603	(2019 AD3)	0.016771	0.037501	46114.605073	5.432121e+07	26.00	False
90833	54017201	(2020 JP3)	0.031956	0.071456	7566.807732	2.840077e+07	24.60	False
90834	54115824	(2021 CN5)	0.007321	0.016370	69199.154484	6.869206e+07	27.80	False
90835	54205447	(2021 TW7)	0.039862	0.089133	27024.455553	5.977213e+07	24.12	False

90836 rows × 8 columns

Separating the numeric data and checking it's summary to get an estimate idea of how the data is distributed.

In [6]:
```
# Seperating numeric data
numeric_df = ["est_diameter_min","est_diameter_max","relative_velocity","miss_distance","absolute_magnitude"]
df[numeric_df].describe()
```

Out[6]:

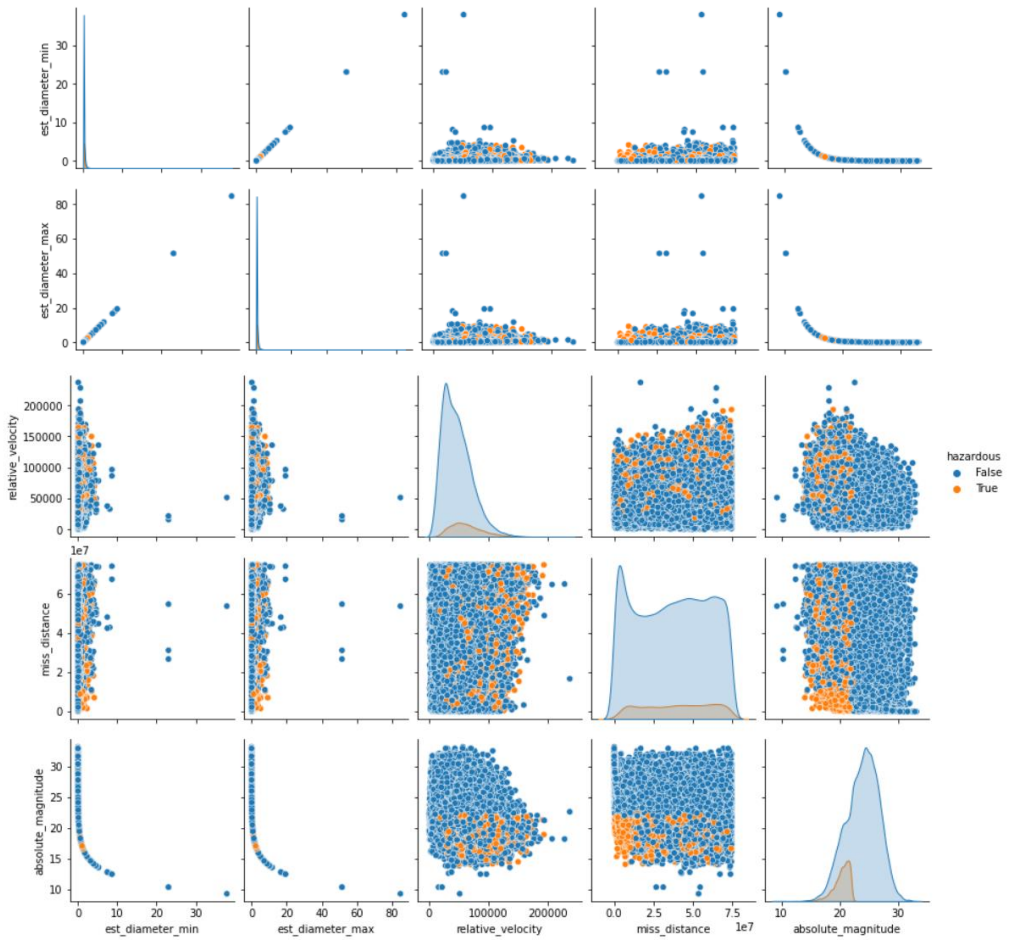
	est_diameter_min	est_diameter_max	relative_velocity	miss_distance	absolute_magnitude
count	90836.000000	90836.000000	90836.000000	9.083600e+04	90836.000000
mean	0.127432	0.284947	48066.918918	3.706655e+07	23.527103
std	0.298511	0.667491	25293.296961	2.235204e+07	2.894086
min	0.000609	0.001362	203.346433	6.745533e+03	9.230000
25%	0.019256	0.043057	28619.020645	1.721082e+07	21.340000
50%	0.048368	0.108153	44190.117890	3.784658e+07	23.700000
75%	0.143402	0.320656	62923.604633	5.654900e+07	25.700000
max	37.892650	84.730541	236990.128088	7.479865e+07	33.200000

VISUALISING THE RELATIONSHIPS BETWEEN THE NUMERICAL DATA

We now try to find relationships between the various numerical features by using visualisation techniques. We will do so using the seaborn and matplot libraries.

In [7]:
```
import seaborn as sns
import matplotlib.pyplot as plt
```

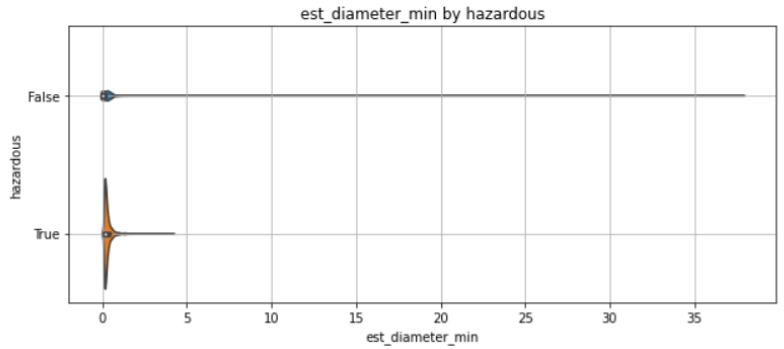
In [8]:
```
# create pairplot to plot multiple pairwise bivariate distributions with hazardous
fig1 = sns.pairplot(df[numeric_df+['hazardous']],hue = 'hazardous')
```

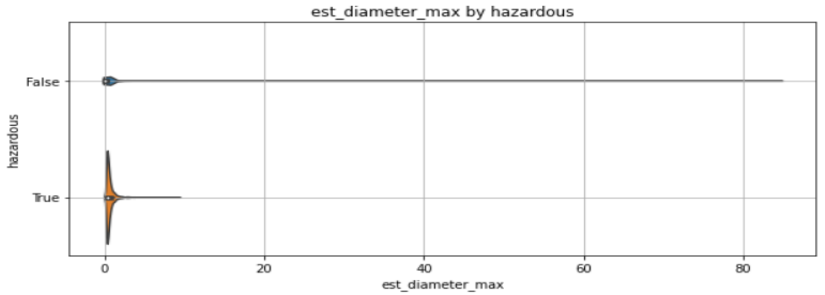


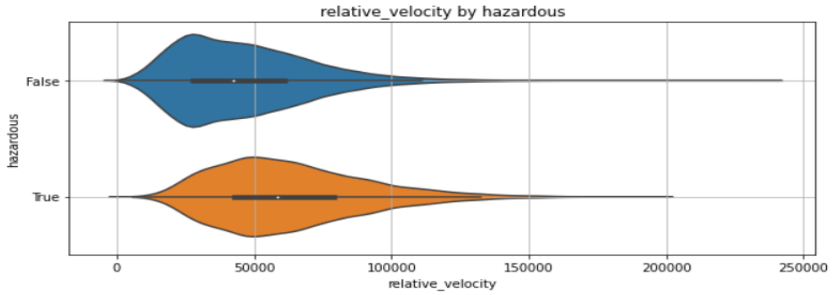
- As seen the maximum and the minimum estimated diameters have a linear relationship
- The miss_distance has hazardous bodies clustering towards the lower values for miss_distance, i.e. the lower the miss_distance the more hazardous the object could be.
- The estimated diameters of the bodies and their absolute magnitude are have a logarithmic relationship.

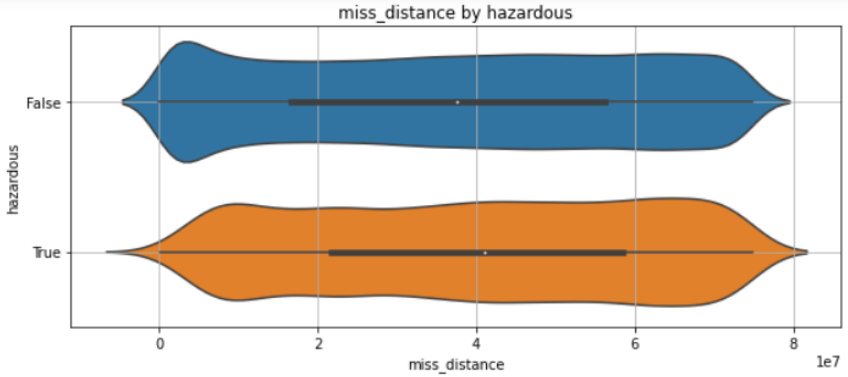
We now use violin plots to check the distributions of hazardous variable (which is our target variable), relative to other numeric data.

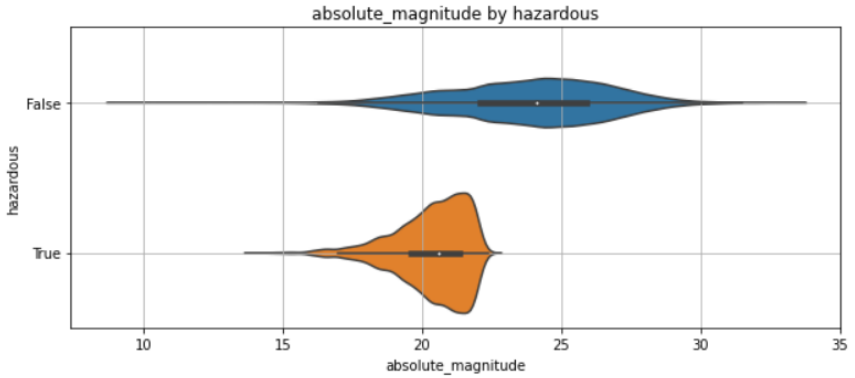
In [9]:
```
# Using violinplot to check hazardous relative to other numeric data
for k in numeric_df:
    plt.figure(figsize=(10,4))
    sns.violinplot(data=df, x=k, y='hazardous', orient='h')
    title = k + ' by hazardous'
    plt.title(title)
    plt.grid()
```











- As we can observe, the hazardous variable is heavily dependent and concentrated around the minimum and maximum diameter variables.
- It also seems to be nearly normally distributed around the relative_velocity variable, with a slight negative skewness.
- And seems to be very widely distributed for the miss_distance variable.

Hence we shall pick the est_diameter_min and the relative_velocity features as our features of interest that we shall use for classification of our target feature (i.e. whether the object is potentially hazardous or not).

Note: Here we could've picked the est_diameter_max instead of the min estimate, this should not make a significant difference in our classification.

Picking only est_diameter_min and relative_velocity features

And splitting the data into training and testing set

For training our models we will split the dataset into train and test datasets, we shall do so by conducting the 80-20 split, that is, 80% of the data in training set and 20% in the testing set.

Using the train_test_split method in the sklearn.model_selection we split the dataset into:

- X_train: Training set predictor variables
- y_train: Training set target variable
- X_test: Test set predictor variables
- y_test: Test set target variables

In [10]:
```
# Spliting into x and y, droping the irrelevant features
x = df.drop(["id","name",'est_diameter_max',"est_diameter_max",'hazardous',"miss_distance","absolute_magnitude", "orbiting
y = df.hazardous.astype('int')    # converting hazardous to int
print(x.shape, y.shape)
```

```
(90836, 2) (90836,)
```

In [11]:
```
# Train test spliting
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(x,y, test_size=0.2, random_state=42)
```

In [12]:
```
X_train
```

Out[12]:

	est_diameter_min	relative_velocity
35538	0.038420	91103.489666
40393	0.192555	28359.611312
58540	0.004619	107351.426865
61670	0.015295	21423.536884
11435	0.011603	69856.053840
...	...	...
6265	0.211132	88209.754856
54886	0.035039	58758.452153
76820	0.211132	52355.509176
860	0.282199	50527.379563
15795	0.075258	22527.647871

72668 rows × 2 columns

MODELLING

For modelling we shall use the hit and trial method as the number of predictor variables are only 2. We shall train a few common classifying machine learning models and compare their accuracies.

The model is fitted to the training data, and the accuracy of the model is calculated by fitting it to the predictor features in the training data and comparing the predicted values of y with the known values of y.

The accuracy score is the ratio of True Positives and True Negatives to the total number of target variables.

- The models we shall use are:
  - XGBoost Classifier
  - KNN
  - Random Forest Classifier
  - Gaussian Naïve Bayes Classifier
  - Decision Tree Classifier

In [13]:

```
# Importing libraries
import plotly.express as px
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from xgboost import XGBClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.linear_model import SGDClassifier
from sklearn.tree import DecisionTreeClassifier
```

XGBoost Classifier:

We fit the training data to the XGBClassifier, then use this model to make predictions on our X_test predictor features and calculate accuracy score by using the accuracy score method from sklearn.metrics

In [14]:
```
XGBC = XGBClassifier()
XGBC.fit(X_train, y_train)
XGBC_pred = XGBC.predict(X_test)
Acc_XGBC = round(accuracy_score(XGBC_pred, y_test) * 100,2)
print(Acc_XGBC)
```

```
91.31
```

KNN Classifier:

We similarly, fit the data into the KNN Classifier with out n_neighbours set to 3. We then fit the model to the training data, use this model to make predictions on our test predictor features and compute it's accuracy using the accuracy score method.

In [15]:
```
KNN = KNeighborsClassifier(n_neighbors = 3)
KNN.fit(X_train, y_train)
KNN_pred = KNN.predict(X_test)
Acc_KNN = round(accuracy_score(KNN_pred, y_test) * 100, 2)
print(Acc_KNN)
```

```
88.01
```

Random Forest Classifier:

We also fit the training data to a Random Forest Classifier. Using this model to make predictions on the X_test predictor variables and calculate its accuracy.

In [16]:

```
RF = RandomForestClassifier()
RF.fit(X_train, y_train)
RF_pred = RF.predict(X_test)
Acc_RF = round(accuracy_score(RF_pred, y_test) * 100, 2)
print(Acc_RF)
```

```
89.69
```

Gaussian Naïve Bayes Classifier:

We also fit the training data to a Gaussian Naïve Bayes Classifier. Using this model to make predictions on the X_test predictor variables and calculate its accuracy.

In [17]:

```
GNB = GaussianNB()
GNB.fit(X_train, y_train)
GNB_pred = GNB.predict(X_test)
Acc_GNB = round(accuracy_score(GNB_pred,y_test) * 100, 2)
print(Acc_GNB)
```

```
90.14
```

Decision Tree Classifier:

We also fit the training data to a Decision Tree Classifier. Using this model to make predictions on the X_test predictor variables and calculate its accuracy.

In [18]:

```
DTC = DecisionTreeClassifier()
DTC.fit(X_train, y_train)
DTC_pred = DTC.predict(X_test)
Acc_DTC = round(accuracy_score(DTC_pred, y_test) * 100, 2)
print(Acc_DTC)
```

```
89.11
```

CONCLUSION

Compiling the accuracies of the models we have trained:

In [19]:
```
models = pd.DataFrame({
    'Model': ['Random Forest', 'XG Boost', 'Gaussian Naive Bayes',
              'KNeighborsClassifier', 'DecisionTreeClassifier'],
    'Score': [Acc_RF, Acc_XGBC, Acc_GNB,
              Acc_KNN, Acc_DTC]})
models.sort_values(by='Score', ascending=False)
```

Out[19]:

	Model	Score
1	XG Boost	91.31
2	Gaussian Naive Bayes	90.14
0	Random Forest	89.69
4	DecisionTreeClassifier	89.11
3	KNeighborsClassifier	88.01

For our purposes, the XGBoost Classifier is best performing and we shall deploy that for our classification of near earth objects as potentially hazardous or not.