REPORT ON Cosmic Collision- Analysing Asteroid Risks with Data

DATA ANALYTICS NSSC'24

TEAM NAME :- TeamByte

PROBLEM & SOLUTION

Overview

Asteroids orbiting the Sun pose potential risks to Earth through possible collisions, which could lead to devastating impacts. To mitigate these risks, data analytics and machine learning techniques will be applied to classify asteroids as hazardous or non-hazardous based on key characteristics such as size, velocity, and orbital proximity to Earth.

Problem Statement

The objective of this analysis is to assess asteroid hazards using a dataset that includes various features. The analysis will involve data exploration, feature engineering, building a classification model, and detecting anomalies to prioritize asteroids for monitoring and potential mitigation.

Key Steps in the Analysis

- **Exploratory Data Analysis (EDA)**
  - Data Inspection: Review and categorize the dataset features, calculate basic statistics, and handle missing values through Simple Imputer from sklearn.

```
data.dtypes
```

Name	int64
Epoch Date Close Approach	float64
Relative Velocity km per sec	object
Relative Velocity km per hr	float64
Miles per hour	float64
Miss Dist.(Astronomical)	float64
Miss Dist.(lunar)	float64
Miss Dist.(kilometers)	float64
Miss Dist.(miles)	float64
Jupiter Tisserand Invariant	float64
Epoch Osculation	float64
Semi Major Axis	float64
Asc Node Longitude	float64
Perihelion Arg	float64
Aphelion Dist	float64
Perihelion Time	float64
Mean Anomaly	float64
Mean Motion	float64
approach_year	float64
approach_month	float64
approach_day	float64
Orbital Period	object
Orbit Uncertainity	object

Identify features that have missing values.

```
data.info()
```

output:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4534 entries, 0 to 4533
Data columns (total 24 columns):
 #   Column                      Non-Null Count  Dtype
---  ------                      --------------  -----
 0   Name                        4534 non-null   int64
 1   Epoch Date Close Approach   3280 non-null   float64
 2   Relative Velocity km per sec  3184 non-null   object
 3   Relative Velocity km per hr   3033 non-null   float64
 4   Miles per hour              3668 non-null   float64
 5   Miss Dist.(Astronomical)    3933 non-null   float64
 6   Miss Dist.(lunar)           3417 non-null   float64
 7   Miss Dist.(kilometers)      3166 non-null   float64
 8   Miss Dist.(miles)           3882 non-null   float64
 9   Jupiter Tisserand Invariant  2802 non-null   float64
 10  Epoch Osculation            3007 non-null   float64
 11  Semi Major Axis             3346 non-null   float64
 12  Asc Node Longitude          3438 non-null   float64
 13  Perihelion Arg              3400 non-null   float64
 14  Aphelion Dist               3719 non-null   float64
 15  Perihelion Time             2970 non-null   float64
 16  Mean Anomaly                3616 non-null   float64
 17  Mean Motion                 3026 non-null   float64
 18  approach_year               3715 non-null   float64
 19  approach_month              3006 non-null   float64
 20  approach_day                3991 non-null   float64
 21  Orbital Period              4004 non-null   object
 22  Orbit Uncertainity          2767 non-null   object
 23  Hazardous                   4534 non-null   bool
dtypes: bool(1), float64(19), int64(1), object(3)
memory usage: 819.3+ KB
```

Calculate and analyze basic statistics for each numerical feature, including range, mean, median, standard deviation, and quartiles.

```
data.describe()
```

	Name	Epoch Date Close Approach	Relative Velocity km per hr	Miles per hour	Miss Dist. (Astronomical)	Miss Dist. (lunar)	Miss Dist. (kilometers)	Miss Dist. (miles)	Jupiter Tisserand Invariant	Epoch Osculation	Semi Major Axis	Asc Node Longitude	Perihel
count	4.534000e+03	3.280000e+03	3033.000000	3668.000000	3933.000000	3417.000000	3.166000e+03	3.882000e+03	2802.000000	3.007000e+03	3346.000000	3438.000000	3400.000
mean	3.268624e+06	1.178921e+12	50516.969113	31312.455735	0.258221	100.709883	3.842441e+07	2.391178e+07	5.126265	2.457720e+06	1.358242	172.185790	185.148
std	5.517954e+05	1.986535e+11	26530.144294	16386.183907	0.146070	56.938739	2.207442e+07	1.357595e+07	1.197144	9.248399e+02	0.465313	103.055919	103.417
min	2.000433e+06	7.889472e+11	1207.814804	750.489149	0.000178	0.187669	2.660989e+04	1.653462e+04	2.367000	2.450936e+06	0.615920	0.001941	0.006
25%	3.092344e+06	1.014365e+12	30437.415189	18843.393552	0.135807	52.877514	1.950318e+07	1.246634e+07	4.179250	2.458000e+06	0.990008	83.288427	96.100
50%	3.513224e+06	1.202458e+12	46968.245275	28959.416222	0.265281	104.261452	3.987901e+07	2.476464e+07	5.102500	2.458000e+06	1.223551	173.895246	192.420
75%	3.691155e+06	1.354954e+12	65210.346095	40331.941346	0.387033	150.434433	5.769962e+07	3.581782e+07	6.043000	2.458000e+06	1.626350	253.635285	273.067
max	3.781897e+06	1.473318e+12	160681.487851	99841.227826	0.499884	194.359650	7.478160e+07	4.646713e+07	9.025000	2.458000e+06	2.568553	359.905890	359.993

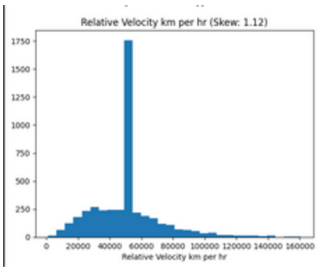
Identify the numerical and categorical features of the dataset to use for further analysis.

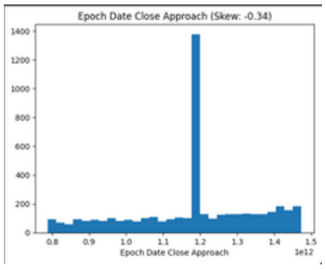
```
[273] categorical_features = []
      numerical_features = []
      for feature in data.columns:
        if (data[feature].dtypes == 'object') or (data[feature].dtypes == 'bool'):
          categorical_features.append(feature)
        else:
          numerical_features.append(feature)
      categorical_features.append('Name')
      numerical_features = np.delete(numerical_features , 0)
      numerical_features
```

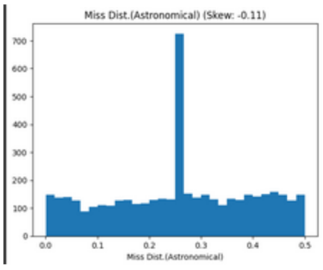
Then we use Simple Imputer from SkLearn for handing missing value

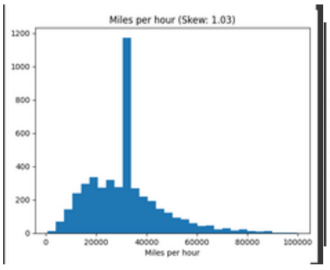
```
#Imputing Strategies
from sklearn.impute import SimpleImputer
imputer_mean = SimpleImputer(missing_values = np.nan, strategy = 'mean')
imputer_mod = SimpleImputer(missing_values = np.nan, strategy = 'most_frequent')
imputer_med = SimpleImputer(missing_values = np.nan, strategy = 'median')
```

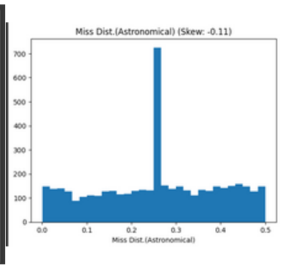
```
for feature in numerical_features:
  plt.hist(df[feature] , bins = 30 )
  plt.xlabel(feature)
  plt.title(f'{feature} (Skew: {skewness_values[feature]:.2f})')
  plt.show()
```





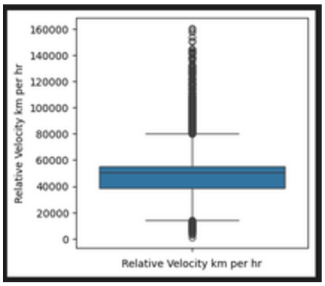


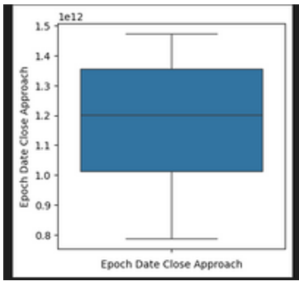


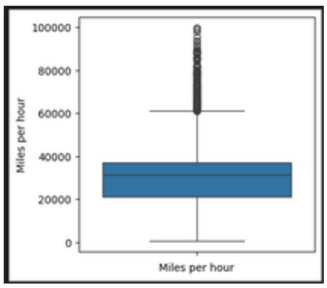


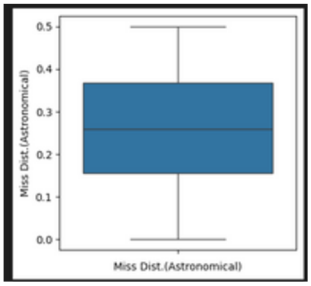
Then we check outliers using Box-plot

```
for feature in numerical_features:
  plt.figure(figsize = (4 ,4))
  sns.boxplot(data[feature])
  plt.xlabel(feature)
  plt.show()
```

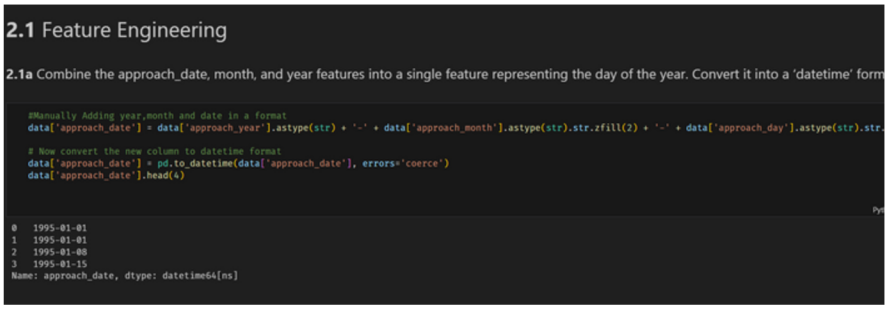








- **Numerical Interpretation and Feature Engineering**
  - Feature Engineering: Create new features such as time until approach, orbital parameters using Kepler's Law, and other orbital characteristics like eccentricity and velocity at key points in the orbit.
  - Additional Features: Introduce innovative features to improve model accuracy.

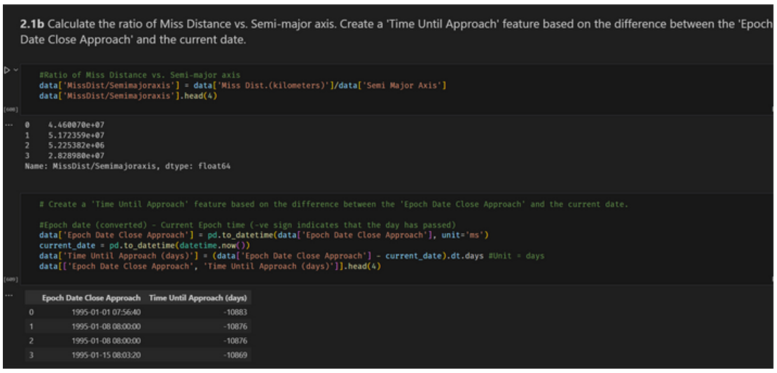


2.1 Feature Engineering

2.1a Combine the approach_date, month, and year features into a single feature representing the day of the year. Convert it into a 'datetime' form

```
#Manually Adding year,month and date in a format
data['approach_date'] = data['approach_year'].astype(str) + '-' + data['approach_month'].astype(str).str.zfill(2) + '-' + data['approach_day'].astype(str).str.

# Now convert the new column to datetime format
data['approach_date'] = pd.to_datetime(data['approach_date'], errors='coerce')
data['approach_date'].head(4)
```

```
0   1995-01-01
1   1995-01-01
2   1995-01-08
3   1995-01-15
Name: approach_date, dtype: datetime64[ns]
```



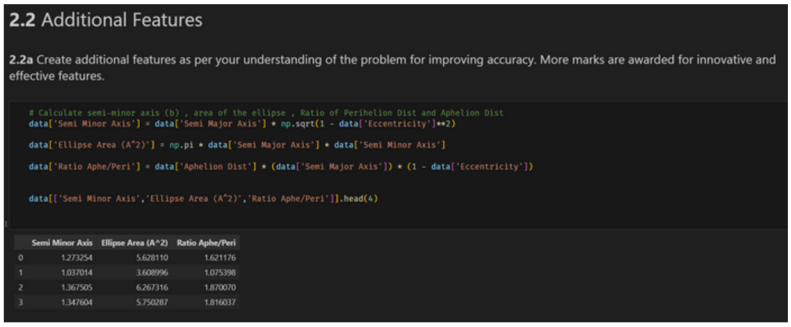
2.1b Calculate the ratio of Miss Distance vs. Semi-major axis. Create a 'Time Until Approach' feature based on the difference between the 'Epoch Date Close Approach' and the current date.

```
#Ratio of Miss Distance vs. Semi-major axis
data['MissDist/Semimajoraxis'] = data['Miss Dist.(kilometers)']/data['Semi Major Axis']
data['MissDist/Semimajoraxis'].head(4)
```

```
0    4.460870e+07
1    5.172359e+07
2    5.225382e+06
3    2.828980e+07
Name: MissDist/Semimajoraxis, dtype: float64
```

```
# Create a 'Time Until Approach' feature based on the difference between the 'Epoch Date Close Approach' and the current date.

#Epoch date (converted) - Current Epoch time (-ve sign indicates that the day has passed)
data['Epoch Date Close Approach'] = pd.to_datetime(data['Epoch Date Close Approach'], unit='ms')
current_date = pd.to_datetime(datetime.now())
data['Time Until Approach (days)'] = (data['Epoch Date Close Approach'] - current_date).dt.days #Unit = days
data[['Epoch Date Close Approach', 'Time Until Approach (days)']].head(4)
```

	Epoch Date Close Approach	Time Until Approach (days)
0	1995-01-01 07:56:40	-10883
1	1995-01-08 08:00:00	-10876
2	1995-01-08 08:00:00	-10876
3	1995-01-15 08:03:20	-10869



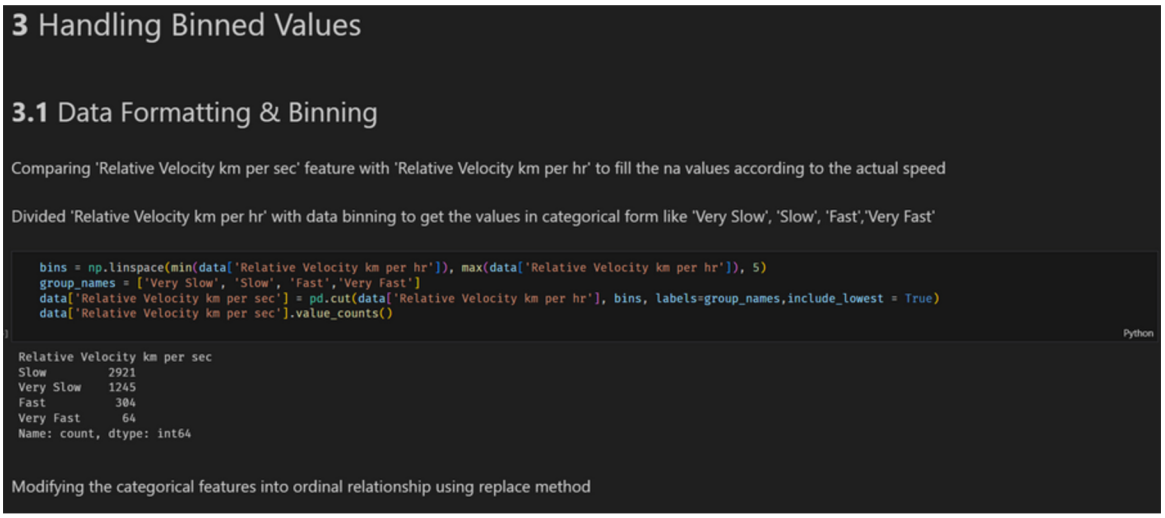
2.2 Additional Features

2.2a Create additional features as per your understanding of the problem for improving accuracy. More marks are awarded for innovative and effective features.

```
# Calculate semi-minor axis (b) , area of the ellipse , Ratio of Perihelion Dist and Aphelion Dist
data['Semi Minor Axis'] = data['Semi Major Axis'] * np.sqrt(1 - data['Eccentricity']**2)

data['Ellipse Area (A^2)'] = np.pi * data['Semi Major Axis'] * data['Semi Minor Axis']

data['Ratio Aphe/Peri'] = data['Aphelion Dist'] * (data['Semi Major Axis']) * (1 - data['Eccentricity'])

data[['Semi Minor Axis','Ellipse Area (A^2)','Ratio Aphe/Peri']].head(4)
```

	Semi Minor Axis	Ellipse Area (A^2)	Ratio Aphe/Peri
0	1.273254	5.628110	1.621176
1	1.037014	3.608996	1.075398
2	1.367505	6.267316	1.870070
3	1.347604	5.750287	1.816037

- **Handling Binned Values**
  - Ordinal Encoding: Convert ordinal categories into numerical values.
  - One-hot Encoding: Apply one-hot encoding for non-ordinal categorical features to prepare the data for classification.



3 Handling Binned Values

3.1 Data Formatting & Binning

Comparing 'Relative Velocity km per sec' feature with 'Relative Velocity km per hr' to fill the na values according to the actual speed

Divided 'Relative Velocity km per hr' with data binning to get the values in categorical form like 'Very Slow', 'Slow', 'Fast','Very Fast'

```
bins = np.linspace(min(data['Relative Velocity km per hr']), max(data['Relative Velocity km per hr']), 5)
group_names = ['Very Slow', 'Slow', 'Fast','Very Fast']
data['Relative Velocity km per sec'] = pd.cut(data['Relative Velocity km per hr'], bins, labels=group_names,include_lowest = True)
data['Relative Velocity km per sec'].value_counts()
```

Python

```
Relative Velocity km per sec
Slow          2921
Very Slow     1245
Fast           304
Very Fast       64
Name: count, dtype: int64
```

Modifying the categorical features into ordinal relationship using replace method

Modifying the categorical features into ordinal relationship using replace method

```
data['Relative Velocity km per sec'] = data['Relative Velocity km per sec'].replace('Very Slow',0).replace('Slow',1).replace('Fast',2).replace('Very Fast',3)
data[['Orbital Period','Orbit Uncertainity']] = data[['Orbital Period','Orbit Uncertainity']].replace('Low',0).replace('Medium',1).replace('High',2)

data[['Relative Velocity km per sec','Epoch Osculation']] = data[['Relative Velocity km per sec','Epoch Osculation']].astype('int64')
data[['approach_year','approach_month','approach_day']] = data[['approach_year','approach_month','approach_day']].astype('int64')
data['Relative Velocity km per sec'].value_counts()
```

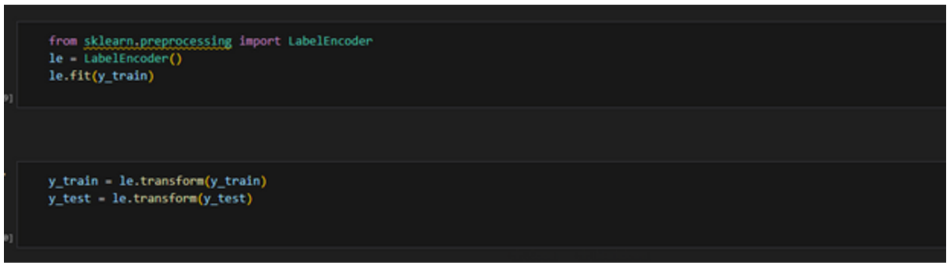
Python

```
Relative Velocity km per sec
1    2921
0    1245
2     304
3      64
Name: count, dtype: int64
```

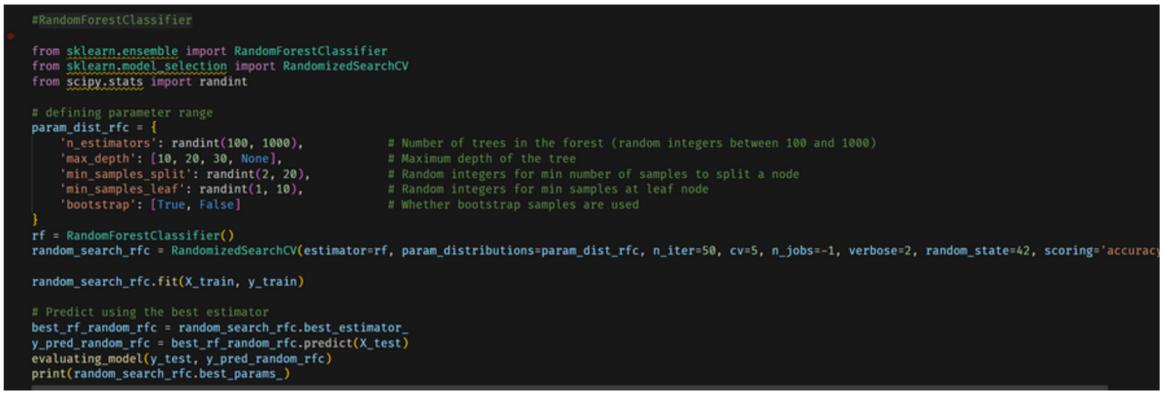
- **Asteroid Hazard Classification**
  - Model Development: Build a machine learning classifier to predict whether an asteroid is hazardous or not.
  - Cross-Validation: Implement K-Fold cross-validation to assess model performance across multiple subsets of data.
  - Hyperparameter Optimization: Tune the classifier's hyperparameters for optimal performance.
  - Model Evaluation: Evaluate the model using metrics such as ROC curves and confusion matrices. Also, assess feature importance using techniques like SHAP values or permutation importance.

```
#Features which are related to time and are not needed in training the model
time_features= ['Name','Epoch Date Close Approach','Epoch Osculation','Perihelion Time','approach_year','approach_month','approach_day','approach_date','T

# Splitting of dataset into X_train, X_test, y_train, y_test

from sklearn.model_selection import train_test_split

X = data.drop(time_features, axis=1).values
y = data['Hazardous'].values

X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.3,random_state=42)
print("Shape of the X_train", X_train.shape)
print("Shape of the y_train", y_train.shape)
print("Shape of the X_test", X_test.shape)
print("Shape of the y_test", y_test.shape)
```

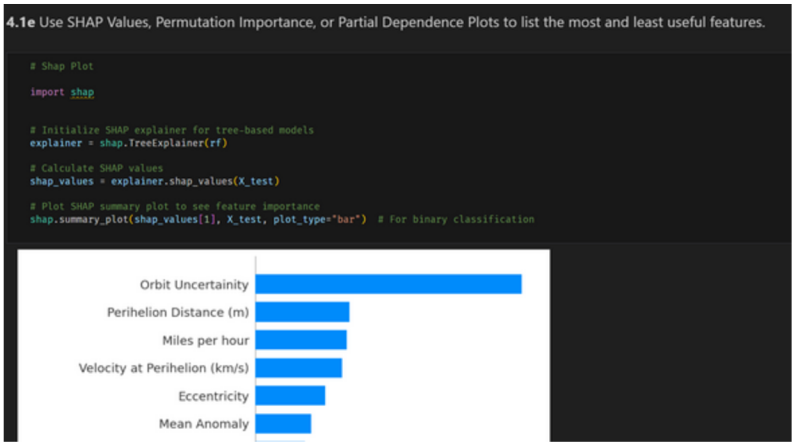
```
Shape of the X_train (3173, 29)
Shape of the y_train (3173,)
Shape of the X_test (1361, 29)
Shape of the y_test (1361,)
```

```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
le.fit(y_train)

y_train = le.transform(y_train)
y_test = le.transform(y_test)
```

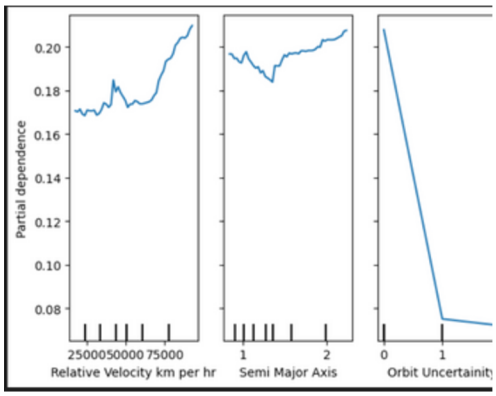


```
#RandomForestClassifier

from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import randint

# defining parameter range
param_dist_rfc = {
    'n_estimators': randint(100, 1000),          # Number of trees in the forest (random integers between 100 and 1000)
    'max_depth': [10, 20, 30, None],             # Maximum depth of the tree
    'min_samples_split': randint(2, 20),         # Random integers for min number of samples to split a node
    'min_samples_leaf': randint(1, 10),          # Random integers for min samples at leaf node
    'bootstrap': [True, False]                   # Whether bootstrap samples are used
}
rf = RandomForestClassifier()
random_search_rfc = RandomizedSearchCV(estimator=rf, param_distributions=param_dist_rfc, n_iter=50, cv=5, n_jobs=-1, verbose=2, random_state=42, scoring='accuracy

random_search_rfc.fit(X_train, y_train)

# Predict using the best estimator
best_rf_random_rfc = random_search_rfc.best_estimator_
y_pred_random_rfc = best_rf_random_rfc.predict(X_test)
evaluating_model(y_test, y_pred_random_rfc)
print(random_search_rfc.best_params_)
```



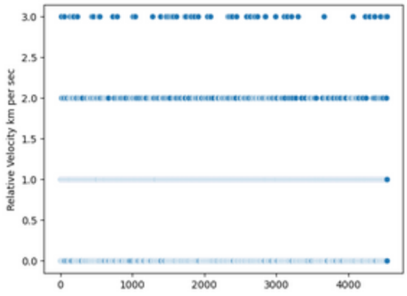
4.1e Use SHAP Values, Permutation Importance, or Partial Dependence Plots to list the most and least useful features.

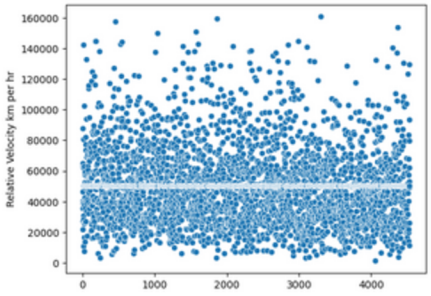
```
# Shap Plot

import shap

# Initialize SHAP explainer for tree-based models
explainer = shap.TreeExplainer(rf)

# Calculate SHAP values
shap_values = explainer.shap_values(X_test)

# Plot SHAP summary plot to see feature importance
shap.summary_plot(shap_values[1], X_test, plot_type="bar")  # For binary classification
```

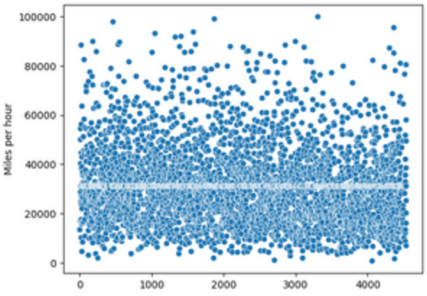



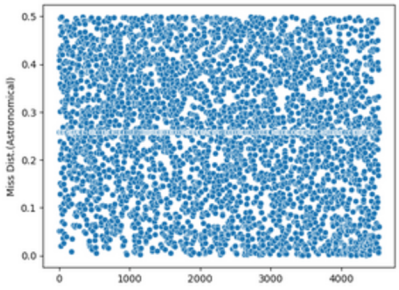
- **Anomaly Detection**
  - Detecting Anomalies: Implement both library-based and custom algorithms for detecting anomalies in asteroid behavior, and compare the results from each method.

```
#to find Anomalies(outliers) features
for feature in X.columns:
  sns.scatterplot(X[feature])
  plt.show()
```









and many more plots

```
from sklearn.ensemble import IsolationForest

#Step 1: Initialize Isolation Forest
iso_forest = IsolationForest(contamination=0.3, random_state=42)#30%

# Step 2: Fit the model and predict anomalies
data['Anomaly_iso'] = iso_forest.fit_predict(data[['Relative Velocity km per hr' , 'Miles per hour'  ,'Semi Major Axis' , 'Jupiter Tisserand Invariant' , 'Mean Motion' , 'Specific A
data['Anomaly_iso'] = data['Anomaly_iso'].apply(lambda x: 1 if x == -1 else 0)
```

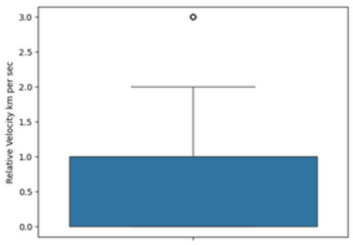
```
data['Anomaly_iso'].value_counts()
```

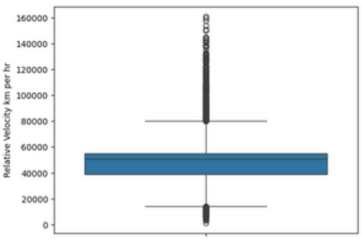
output

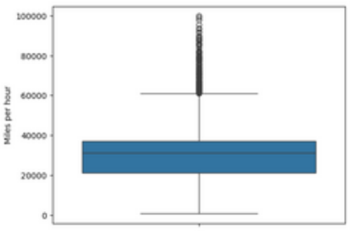
```
Anomaly_iso
0    3174
1    1360
Name: count, dtype: int64
```

Writing your own anomaly detection algorithm. here We will build a custom anomaly detection algorithm using the Z-Score method

```
#using box plot we find outliers
for feature in X.columns:
  sns.boxplot(X[feature])
  plt.show()
```







and many more plots

```
# Step 1: Finding the iqr
per25 =   data[['Relative Velocity km per hr' , 'Miles per hour'  ,'Semi Major Axis' , 'Jupiter Tisserand Invariant' , 'Mean Motion' , 'Specific Angular Momentum (m^2/s)' ,'Perihel
per75 = data[['Relative Velocity km per hr' , 'Miles per hour'  ,'Semi Major Axis' , 'Jupiter Tisserand Invariant' , 'Mean Motion' , 'Specific Angular Momentum (m^2/s)' ,'Perihelio
iqr = per75 - per25
#print(iqr)
#set the limit
upper_limit = per75 + 1.5*iqr
lower_limit = per25 - 1.5*iqr
# Detect anomalies
#print(upper_limit)
#print(lower_limit)
anomalies = (data[['Relative Velocity km per hr' , 'Miles per hour'  ,'Semi Major Axis' , 'Jupiter Tisserand Invariant' , 'Mean Motion' , 'Specific Angular Momentum (m^2/s)' ,'Peri
#Mark anomalies as 1 and normal points as 0
data['Anomaly_iqr'] = anomalies.any(axis=1).astype(int)
```

```
data['Anomaly_iqr'].value_counts()
```

output

```
Anomaly_iqr
0    2750
1    1784
Name: count, dtype: int64
```

Store the results as a new column in the dataset. Print the number of anomalies detected by each method

```
print(f"Number of anomalies detected (isolation Forest): {data['Anomaly_iso'].sum()}")
print(f"Number of anomalies detected (IQR): {data['Anomaly_iqr'].sum()}")
```

output

```
Number of anomalies detected (isolation Forest): 1360
Number of anomalies detected (IQR): 1784
```

Compare the results from both methods by plotting a Confusion Matrix. Print the number of examples flagged by both algorithms

```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
# Generate the confusion matrix
conf_matrix = confusion_matrix(data['Anomaly_iqr'], data['Anomaly_iso'])

# Print the confusion matrix
print("Confusion Matrix:\n", conf_matrix)

# Plot the confusion matrix
ConfusionMatrixDisplay(conf_matrix).plot()
plt.title("Confusion Matrix: IQR method vs Isolation Forest(inbuilt)")
plt.show()

# Print the number of examples flagged by both algorithms
both_flagged = ((data['Anomaly_iqr'] == 1) & (data['Anomaly_iso'] == 1)).sum()
print(f"Number of examples flagged by both algorithms: {both_flagged}")
```

output

```
Confusion Matrix:
 [[2678   72]
 [ 496 1288]]
```