REPORT

1. EXPLORATORY DATA ANALYSIS :

1.1 DATA INSPECTION :

- ➢ The data is imported as df and its shape is (4534, 24) .The shape (4534,24) says there are **4534** data points and each data point has **24** different features.
- ➢ It can be seen that there is one int64 feature called "**Name**" , 3 object class Ordinal Features and all the other features being float64 dtype.
- ➢ The df.describe() function provides a comprehensive statistical summary of the dataset, giving insights into the **central tendency, dispersion, and shape** of the distribution of the data.

This data includes **mean ,standard deviation ,minimum,maximums ,Quartiles** of all the features which helps analyze how the data is varied over features.

	count	mean	std	min	25%	50%	75%	max
Name	4534.0	3.268624e+06	5.517954e+05	2.000433e+06	3.092344e+06	3.513224e+06	3.691155e+06	3.781897e+06
Epoch Date Close Approach	3280.0	1.178921e+12	1.986535e+11	7.889472e+11	1.014365e+12	1.202458e+12	1.354954e+12	1.473318e+12
Relative Velocity km per hr	3033.0	5.051697e+04	2.653014e+04	1.207815e+03	3.043742e+04	4.696825e+04	6.521035e+04	1.606815e+05
Miles per hour	3668.0	3.131246e+04	1.638618e+04	7.504891e+02	1.884339e+04	2.895942e+04	4.033194e+04	9.984123e+04
Miss Dist. (Astronomical)	3933.0	2.582210e-01	1.460704e-01	1.778761e-04	1.358070e-01	2.652806e-01	3.870335e-01	4.998841e-01
Miss Dist. (lunar)	3417.0	1.007099e+02	5.693874e+01	1.876690e-01	5.287751e+01	1.042615e+02	1.504344e+02	1.943596e+02
Miss Dist. (kilometers)	3166.0	3.842441e+07	2.207442e+07	2.660989e+04	1.950318e+07	3.987901e+07	5.769962e+07	7.478160e+07
Miss Dist. (miles)	3882.0	2.391178e+07	1.357595e+07	1.653462e+04	1.246634e+07	2.476464e+07	3.581782e+07	4.646713e+07
Jupiter Tisserand Invariant	2802.0	5.126265e+00	1.197144e+00	2.367000e+00	4.179250e+00	5.102500e+00	6.043000e+00	9.025000e+00
Epoch Osculation	3007.0	2.457720e+06	9.248399e+02	2.450936e+06	2.458000e+06	2.458000e+06	2.458000e+06	2.458000e+06
Semi Major Axis	3346.0	1.358242e+00	4.653126e-01	6.159204e-01	9.900082e-01	1.223551e+00	1.626350e+00	2.568553e+00
Asc Node Longitude	3438.0	1.721858e+02	1.030559e+02	1.940674e-03	8.328843e+01	1.738952e+02	2.536353e+02	3.599059e+02
Perihelion Arg	3400.0	1.851489e+02	1.034172e+02	6.917625e-03	9.610051e+01	1.924201e+02	2.730680e+02	3.599931e+02
Aphelion Dist	3719.0	1.898115e+00	8.344986e-01	8.037653e-01	1.260397e+00	1.590005e+00	2.331365e+00	4.662158e+00
Perihelion Time	2970.0	2.457741e+06	9.155433e+02	2.450100e+06	2.457826e+06	2.457976e+06	2.458109e+06	2.458706e+06
Mean Anomaly	3616.0	1.827352e+02	1.077570e+02	3.191491e-03	8.706667e+01	1.890511e+02	2.780387e+02	3.599180e+02
Mean Motion	3026.0	7.610904e-01	3.377451e-01	2.393633e-01	4.838406e-01	7.376844e-01	1.002870e+00	1.946801e+00
approach_year	3715.0	2.006921e+03	6.292621e+00	1.995000e+03	2.002000e+03	2.008000e+03	2.013000e+03	2.016000e+03
approach_month	3006.0	6.474385e+00	3.462926e+00	1.000000e+00	3.000000e+00	7.000000e+00	1.000000e+01	1.200000e+01
approach_day	3991.0	1.489652e+01	5.717762e+00	1.000000e+00	8.000000e+00	1.500000e+01	2.200000e+01	2.200000e+01

We can observe one particular thing from Count that a good amount of data contains NULL values and should be carefully handled. Other observations include a few things like Mean Motion being less than 1 and Epoch Date Closest Approach bearing a factor of 10^{12} indicates a huge requirement for Scaling the data before feeding it to the Classifier.

Dataset Contains overall **23.32%** Null Values.

Filling of Null Values:

- ➢ There are **three** categorical features:

- **Relative Velocity km per sec** categorical feature depends on '**Relative Velocity km per hr**' feature. **Orbital Period** feature categorical values depend on **Semi major Axis**(T^2 proportional to a^3). Thus Their Null Values are filled using them.

➢ Null values of features **Epoch Date Close Approach, Epoch Osculation, approach_year, approach_month, approach_day, Perihelion Time,Orbit Uncertainity** are filled with **forward values**.

➢ Null Values of features **Miss dist.(Astronomical), Miss dist.(lunar),Miss dist.(kilometers),Miss dist.(miles)** are filled with **unit conversion factors** as they are the same thing only different due to different units.

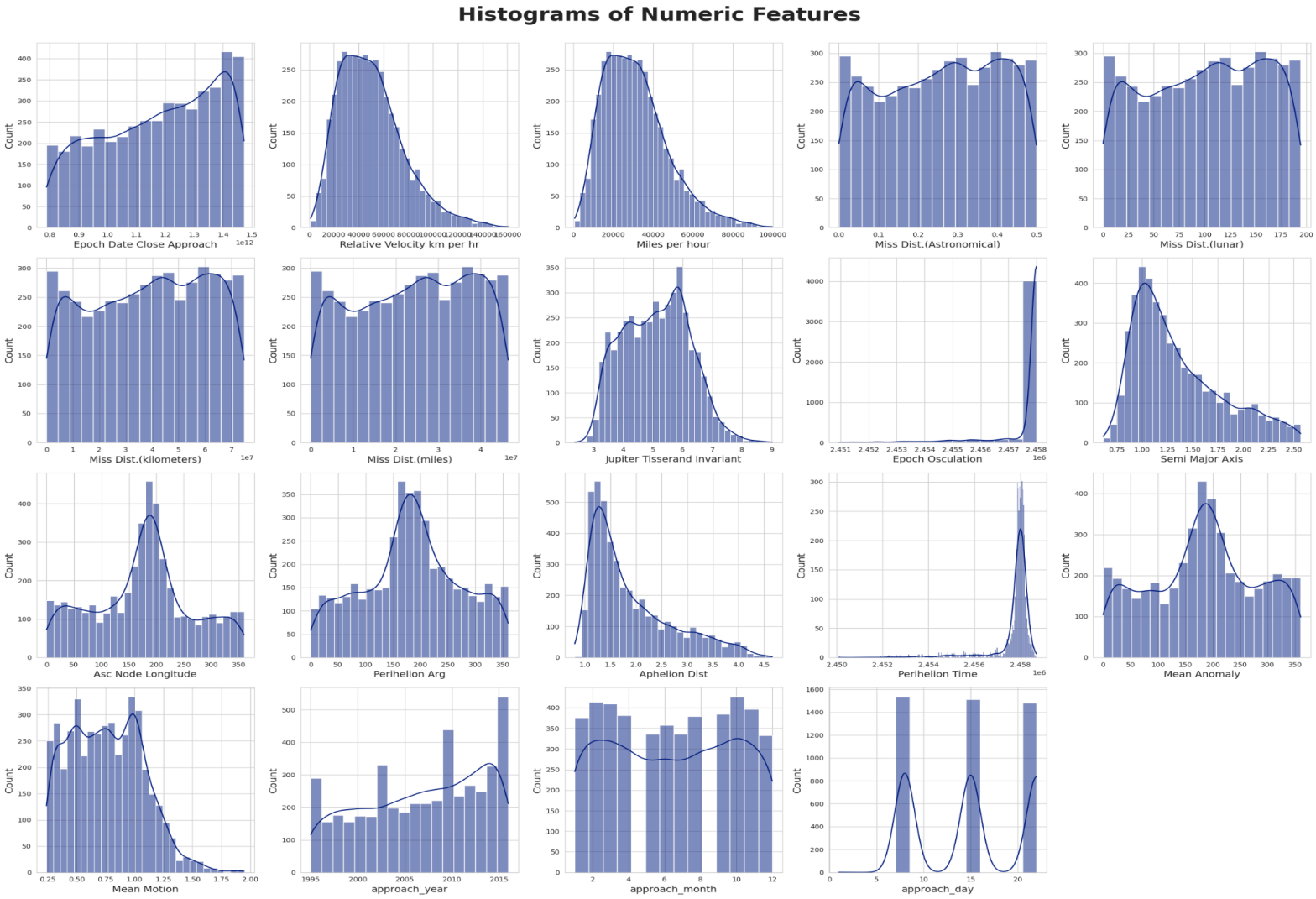
➢ null values of features **Jupiter Tisserand Invariant,Semi Major Axis,Asc Node Longitude,Perihelion Arg,Mean Anomaly,Mean Motion,Aphelion Dist** are filled using the **k-nearest neighbors approach**.(scikit-learn's **KNNImputer** is used for this task.

Discussing How this process differs for numerical and categorical columns

Filling null values in the dataset (**imputation**) is different for **Numerical** and **categorical Features**. For Numerical features averaging,summing,multiplication and other mathematical operations can be performed to fill the null values. On the other hand for Categorical features mathematical operations are not meaningful, it requires other methods to fill them like filling with mode,most frequent or category prediction.

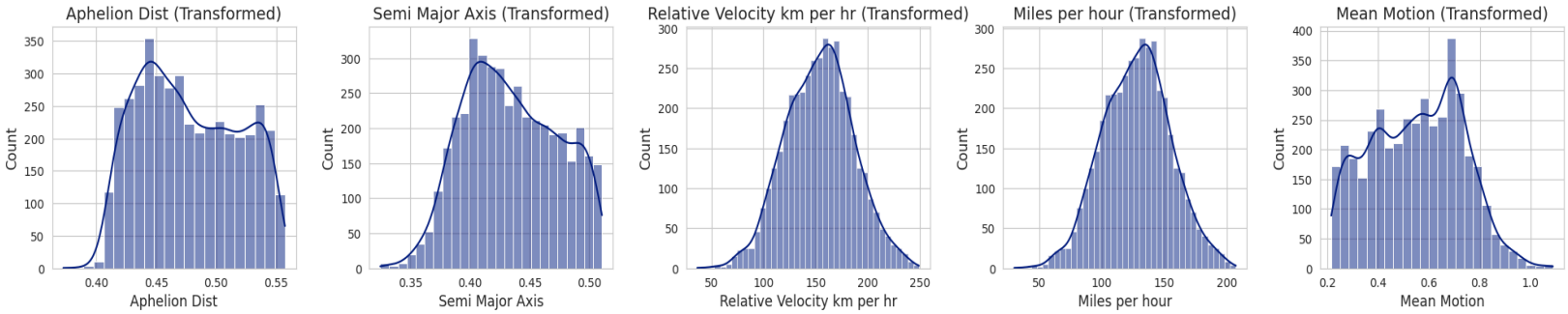
1.2 STATISTICAL INFERENCE:

Distribution of the features:



Since the features like Aphelion Dist,Semi Major Axis,Relative Velocity km per hr,Miles per hour,Mean Motion are skewed, they are **Normalized** by using **YeoJohnson Method** which is applicable for both Positively and Negatively Skewed data.

Histograms of Transformed Features (Yeo-Johnson)



Removing **Skewness** in features of the dataset is very Important. Many machine learning algorithms assume that the input features are **normally distributed**. Highly skewed features can violate this assumption, leading to suboptimal model performance. Skewed distributions often indicate the presence of **outliers**. Transforming skewed data can **reduce the impact** of these outliers, making the model less sensitive to extreme values. Thus Reducing skewness helps in building more accurate, interpretable, and robust models. It ensures that the data aligns better with the assumptions of many machine learning algorithms, leading to **better performance** and **more reliable predictions**.

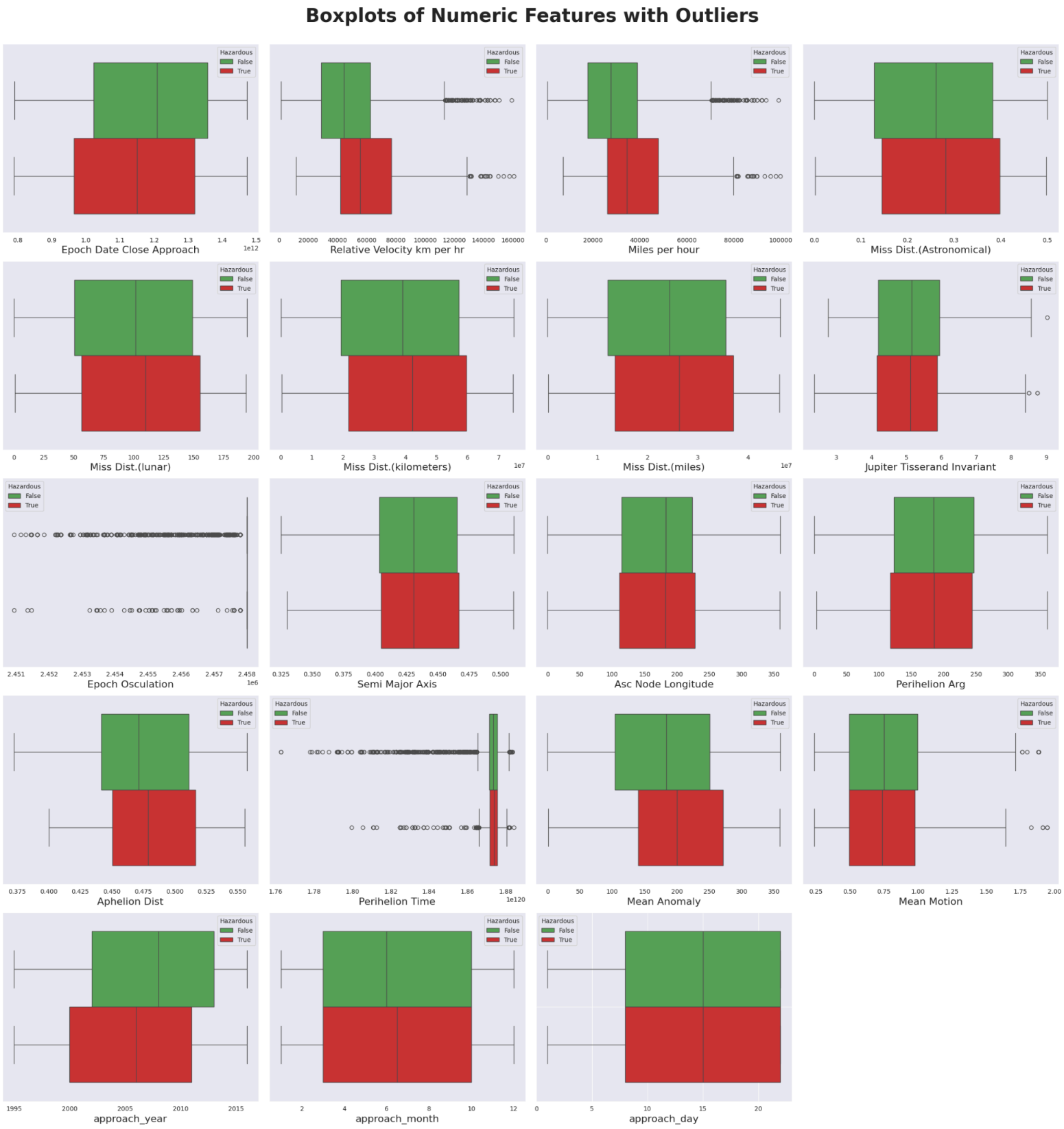
IDENTIFICATION OF OUTLIERS:

Outliers can disproportionately influence the results of statistical analyses and machine learning models, leading to inaccurate predictions. Models trained on datasets with outliers may be less robust and generalize poorly to new, unseen data. Removing outliers can lead to a more stable model that performs consistently across different datasets.
Outliers can be identified using **boxplot** and **Z-score**.

In **Boxplot**, Box Represents the **interquartile range (IQR),** which contains the **middle 50%** of the data. The edges of the box are the **first quartile (Q1) and the third quartile (Q3)**. Lines extending from the box to the smallest and largest values within **1.5 times the IQR** from the lower and upper quartiles, respectively are whiskers. Points that fall outside the **whiskers**, typically represented as individual dots are outliers.

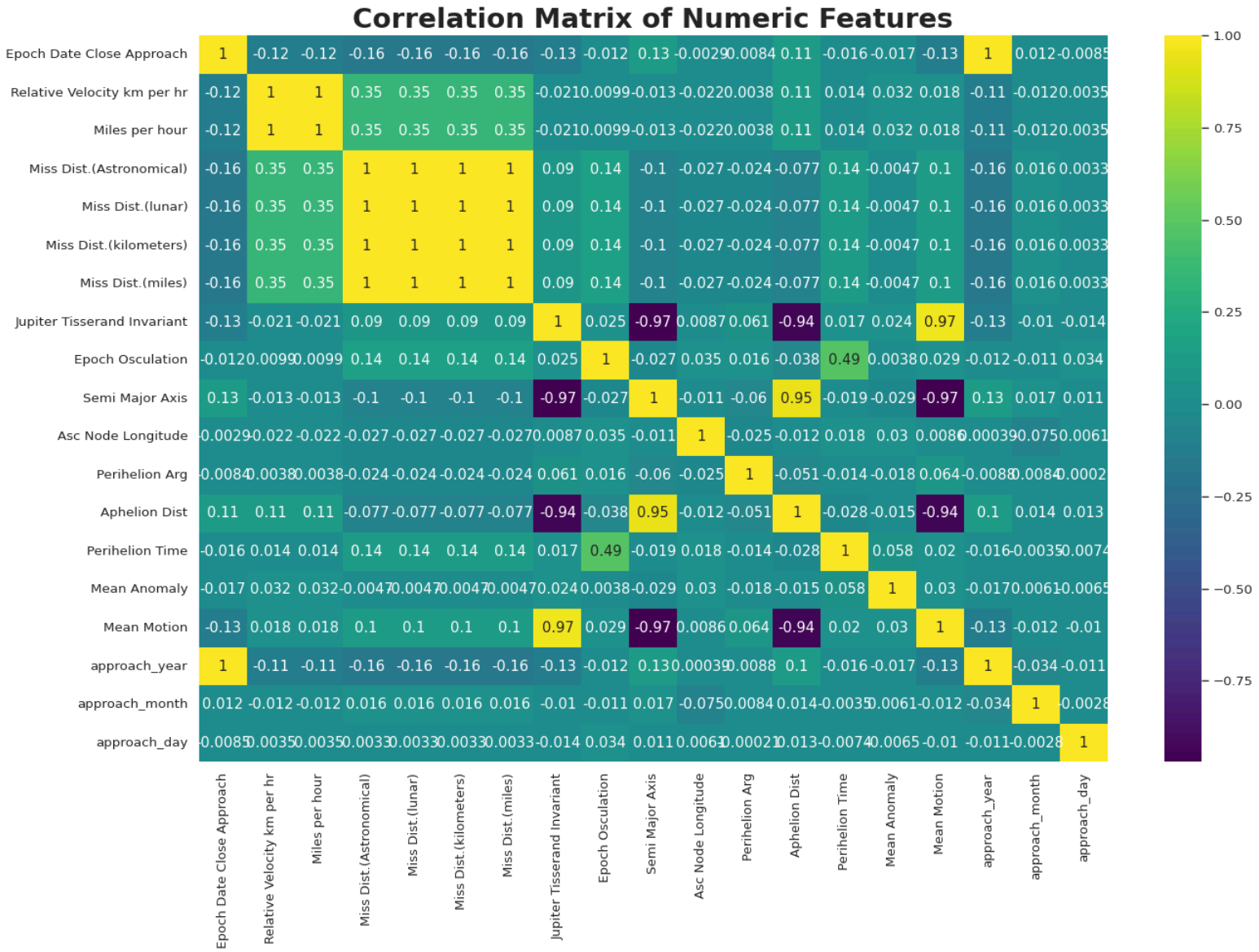
Z-score on other hand measures how far each data point is from the mean, in terms of standard deviations. **$Z=(x-\mu)/\sigma$** .A data point is considered an outlier if its Z-Score is significantly high or low (usually greater than 3 or less than -3).

Identification of Outliers using Boxplot:



From The Plot we can see that the Features **'Relative Velocity km per hr'',''Miles per hour'',''Epoch Osculation'',''Perihelion Time'** have outliers. But there are 23 features and only 4 have outliers. Thus we can avoid these outliers. Removing them will lead to a lot of loss of data so we are not Removing the outliers for these 4 features.

Correlation among the features:



This **correlation matrix** reveals the relationships between the numeric features of a dataset.

A few key observations:

Perfect correlations:

- **"Relative Velocity km per hr"** and "**Miles per hour**" are perfectly correlated (1.0), indicating they are unit conversions.
- All "**Miss Distance**" variations (**astronomical, lunar, kilometers, miles**) are also perfectly correlated (1.0), reflecting different units of the same measurement.

Strong correlations:

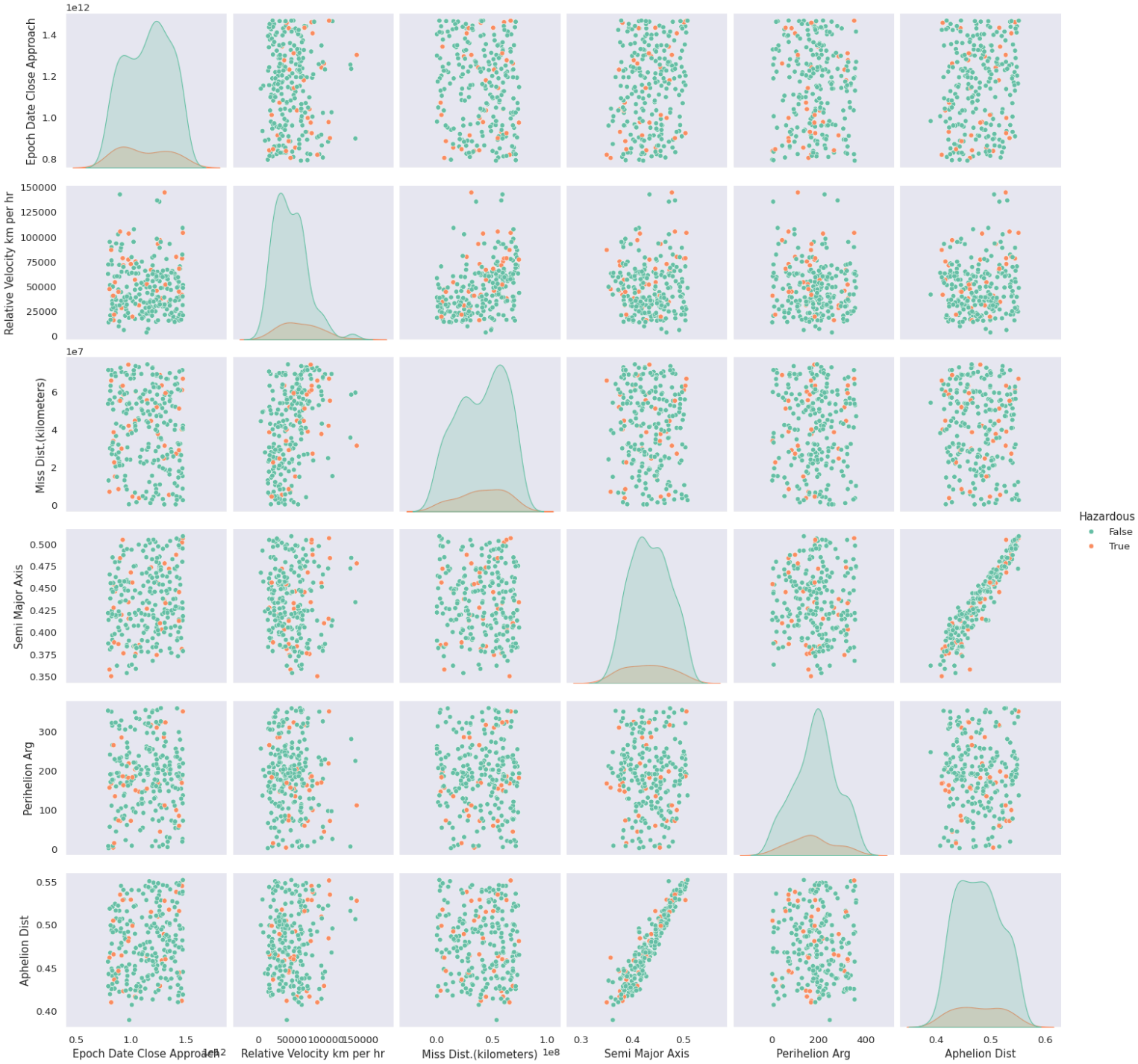
- **"Aphelion Distance" and "Semi Major Axis"** have a strong positive correlation (**0.95**), as both describe orbital properties.
- "**Mean Motion" and "Semi Major Axis**" have a strong negative correlation **(-0.96)**, consistent with orbital mechanics where a larger semi-major axis results in slower mean motion.

In summary, many features are either directly related by **unit conversions** or **derived physical properties** (like mean motion and semi-major axis), while others are **largely independent**.

1.3 VISUALIZATION :

Pairplot:

As there are a lot of features (more than 20) if all the features are plotted then the graph becomes too clumsy ,so to avoid that some of the important and that are independent from one other are considered for plotting.



In the Pair Plot we can see that there is a **linear Relationship** between **Semi Major** Axis and **Aphelion dist.** In other Plots randomly Scattered Plots suggest Little correlation between features.

In a pair plot, diagonal and off-diagonal plots serve different purposes in visualizing relationships and distributions within a dataset.

Diagonal Plots:

The diagonal plots display the distribution of each individual variable in the dataset.They often use kernel density estimates (KDE) to visualize how the data is distributed for each feature. You can see if the variable is normally distributed, skewed (positively or negatively), bimodal, etc.Helps identify the range of values and potential outliers in each variable.Mean and median can be inferred from the shape of the distribution.

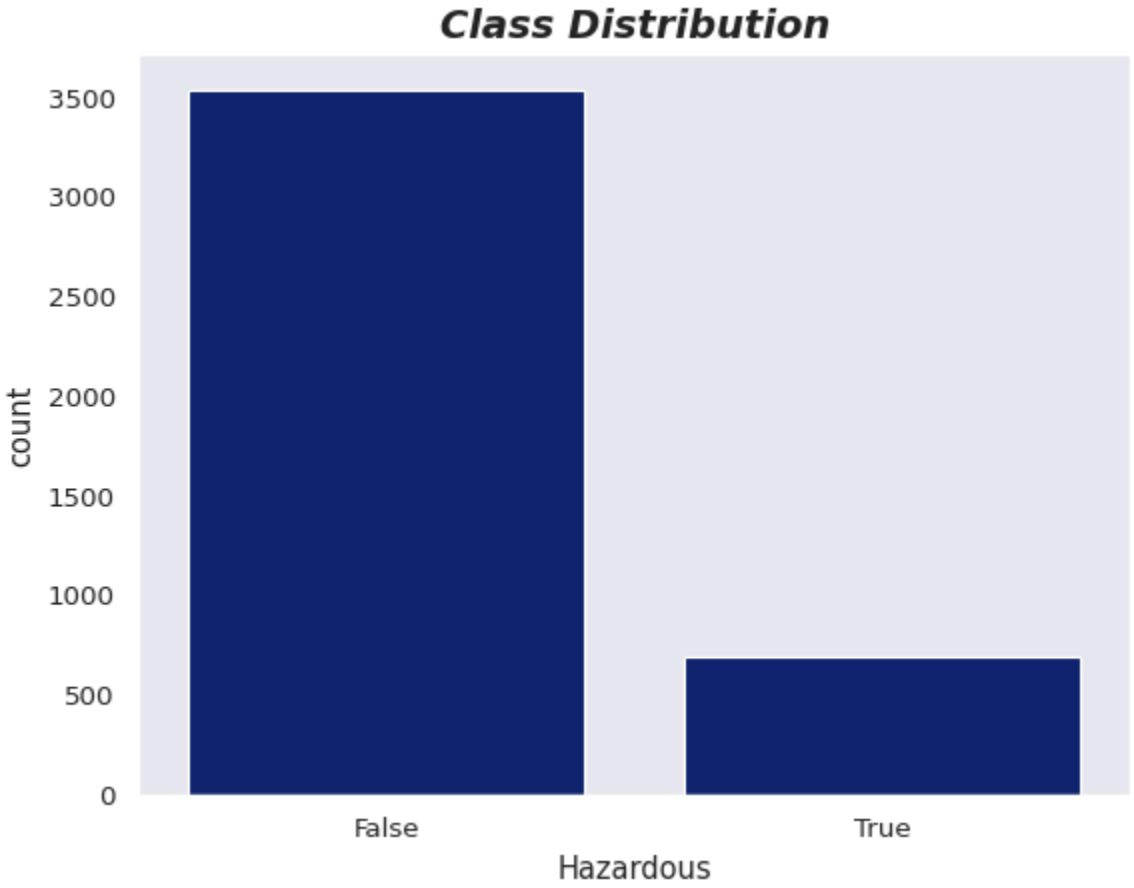
Off-Diagonal Plots:

The off-diagonal plots depict the relationships between pairs of variables, usually using scatter plots or other types of bivariate plots.You can assess the strength and direction of relationships between variables. For example, a positive correlation will show a trend where one variable increases as the other does.Identify clusters, or groups within the data. If a hue parameter is included, these plots can show how different categories (e.g., Hazardous in the given dataset) influence the relationships between numerical variables.

1.4 TACKLING CLASS IMBALANCE :

Tackling class imbalance is crucial in machine learning and statistical modeling.

In the given dataset the **Majority** class is **'False' 83.63%** and the **Minority** class **'True' is 16.37%**.
So our Dataset has **Class Imbalance**.



To tackle such problem We can Increase the number of samples in the minority class by duplicating instances or generating synthetic samples using libraries like **SMOTE**.Also Training Algorithms which we can use are Ensemble like boosting**(Xgboost,Catboost,etc.)** or **SVM(Support Vector Machine)** which can deal with class imbalance.For our Problem Statement We will be using **SVM**. Also we will be using Methods for giving more priority to Minority class during Training.

Discussing the implication of class imbalance on model performance :

Class imbalance can significantly impact model performance in various ways.Models tend to predict the majority class more often thus bias towards Majority Class.The model may achieve high overall accuracy but perform poorly on the minority class, resulting in a high number of false negatives.It can also lead to misleading evaluation metrics. So as we Required to Predict whether an asteroid is hazardous or not , with True class being minority it can be biased to False class and model may predict asteroid non-Hazardous even though it is Hazardous which is dangerous in real-world situations. So dealing with class imbalance is important.

2. NUMERICAL INTERPRETATION AND MATHEMATICAL ANALYSIS :

2.1 Feature Engineering :

- For Combining the **approach_date, month, and year** features into a single feature representing the day of the year we used **pd.to_datetime** function of pandas library.

```
df2['approach_date'] = pd.to_datetime(df2[['approach_year',
'approach_month', 'approach_day']].rename(columns={
    'approach_year': 'year',
    'approach_month': 'month',
    'approach_day': 'day'
}))

df3 = df2.drop(['approach_year', 'approach_month', 'approach_day','Miss
Dist.(miles)','Miss Dist.(kilometers)','Miss Dist.(lunar)'] ,axis=1)
```

- Ratio of **Miss Distance vs. Semi-major axis** is calculated and also the **'Time Until Approach'** feature based on the difference between the **'Epoch Date Close Approach'** and the current date is being created.

```
df3['MD_to_SMA_ratio'] = df3['Miss Dist.(Astronomical)'] / df3['Semi Major
Axis']

current_time = pd.Timestamp.now()
current_epoch_date = int(current_time.timestamp() )

df3['Time_till_Approach'] = df3['Epoch Date Close Approach'] -
current_epoch_date
```

Calculating the eccentricity of the orbit, average orbital velocity, and orbital period using Kepler's Law

- ***Eccentricity*** **:**Eccentricity measures an orbit's deviation from circularity, ranging from 0 to 1. A value of 0 indicates a circular orbit, while values closer to 1 signify more elongated or elliptical orbits. It is calculated using the perihelion and aphelion distances.
- ***Kepler's Third Law***: The orbital period (T) is the time required for an object to complete one full orbit around a central body. It depends on the semi-major axis and the mass of the central body; a longer orbital period corresponds to a larger semi-major axis.
- ***Average Orbital Velocity (v)***: Orbital velocity is the speed needed to maintain a stable orbit. It varies with distance from the central body, being highest at perihelion and lowest at aphelion.

Code for above Calculation is given below:

```
r_p = 2 * df['Semi Major Axis'] - df['Aphelion Dist']
df3['eccentricity'] = (df['Aphelion Dist'] - r_p) / (df['Aphelion Dist'] +
r_p)

mu = 1.327e20    #(G*M)
AU_to_meters = 1.496e11
sma_au = df['Semi Major Axis'] * AU_to_meters
df3['Orbital Period (seconds)'] = 2 * np.pi * np.sqrt((sma_au** 3) / mu)

df3['Orbital Velocity (m/s)'] = np.sqrt(mu/ df['Semi Major Axis'])
```

Calculating the heliocentric distance, escape velocity, and specific orbital energy.

- ***Heliocentric Distance Calculation***: Heliocentric distance measures the distance from an object (e.g., a planet) to the Sun, based on the Sun's center. It is crucial for understanding the gravitational forces on celestial bodies in the solar system.

Escape Velocity: Escape velocity is the minimum speed required for an object to break free from a celestial body's gravitational influence without additional propulsion. It depends on the body's mass and radius.

Specific Orbital Energy: Specific orbital energy is the total mechanical energy (kinetic + potential) per unit mass of an orbiting body. It indicates the orbit type: negative for bound orbits (e.g., planets) and zero or positive for unbound orbits (e.g., comets).

Code for above Calculation is given below:

```
df3['Perihelion Dist'] = df['Semi Major Axis'] * (1 - df3['eccentricity'])

df3['Escape Velocity at Perihelion (km/s)'] = np.sqrt(2 * mu /
(df3['Perihelion Dist']*AU_to_meters))/1000.0
```

```
df3['Escape Velocity at Aphelion (km/s)'] = np.sqrt(2 * mu / df3['Aphelion Dist']*AU_to_meters)/1000.0

df3['Specific Orbital Energy'] = -mu / (2 * df['Semi Major Axis']*AU_to_meters)
```

Calculating the Specific Angular Momentum and velocity at Perihelion and Aphelion:

- ***Specific Angular Momentum:*** The specific angular momentum hhh measures an orbiting body's rotational motion, calculated using the gravitational constant GGG, the mass MMM of the central body, the semi-major axis aaa, and the eccentricity eee. Higher specific angular momentum indicates a more stable orbit, balancing gravitational attraction with the body's inertia.
- ***Velocity at Perihelion:*** The velocity at perihelion vpv_pvp is the speed of an orbiting body at its closest approach to the central body. This speed increases due to stronger gravitational pull as the body nears the central mass, resulting in higher velocities at perihelion than at other points in the orbit.
- ***Velocity at Aphelion:*** The velocity at aphelion vav_ava is the speed of an orbiting body at its farthest distance from the central body. The weaker gravitational force at this distance leads to a lower velocity compared to perihelion. This illustrates Kepler's laws, where an orbiting body moves faster when closer to the central mass and slower when farther away.

Code for above Calculation is given below:

```
df3['Specific Angular Momentum'] = np.sqrt(mu * df['Semi Major Axis'] * (1 - (df3['eccentricity']**2)))
a=1/ df['Semi Major Axis']
b=2 / df3['Perihelion Dist']
c=2 / df['Aphelion Dist']
df3['Velocity at Perihelion'] = np.sqrt(((mu*b) - a))
df3['Velocity at Aphelion'] = np.sqrt( ((mu*c) - a))
```

Calculating Synodic Period and Mean Motion

- ***Synodic Period :*** The time interval between consecutive alignments of a celestial body as viewed from Earth. It accounts for both the orbit of the celestial body and the orbit of Earth itself
- ***Mean Motion :*** The average rate at which a celestial object travels along its orbit, expressed as the angle (in radians) it covers per unit of time, typically per year. It indicates how fast the object moves in its elliptical path around a central body.

Code for above Calculation is given below:

```
P_earth = 365.25 * 86400

df3['Synodic Period (seconds)'] = 1 / (1 / P_earth - 1 / df3['Orbital Period (seconds)'])
```

```
df3['Mean Motion (radians/sec)'] = 2 * np.pi / df3['Orbital Period
(seconds)']
```

2.2 Additional Features :

For identifying whether a asteroid is Hazardous or not additional feature we have chosen are **Velocity Ratio, Orbital Inclination and Orbital stability ratio**. These Features can be important in predicting the nature of Asteroid as they tell a lot about it as Explained below.

- ***Velocity Ratio***: Measures an asteroid's speed relative to its escape velocity. A higher ratio suggests faster movement relative to its gravitational binding energy, increasing collision risk. Perihelion, the closest point to the Sun, is where escape velocity is most relevant due to stronger gravitational pull.
- ***Orbital Inclination:*** The tilt of an asteroid's orbit relative to the ecliptic plane. Higher inclination indicates a more eccentric, potentially hazardous orbit.
- ***Orbital Stability Ratio:*** Assesses orbit stability by comparing the semi-major axis to eccentricity. A stable ratio helps predict long-term orbital behavior.

Code for above Calculation is given below:

```
df3['Velocity Ratio'] = (df3['Relative Velocity km per hr']*60)/df3['Escape
Velocity at Perihelion (km/s)']
df3['Orbital Inclination'] = (df['Asc Node Longitude'] + df['Perihelion
Arg'])%360
df3['Orbital Stability Ratio'] = df3['Semi Major Axis']/df3['eccentricity']
```

3. HANDLING BINNED VALUES :

Modifying binned (or categorical) features into numeric values is often necessary because many machine learning algorithms require **numerical inputs** to work effectively. Most machine learning models, especially linear models (like Linear Regression, Logistic Regression) and tree-based models (like Decision Trees, Random Forests), expect numerical input. They cannot process categorical data directly unless it is converted to numeric.

Categorical data can be broadly classified into two types: **ordinal** and **nominal**.

- **Ordinal data** are categorical variables that have a clear, meaningful order or ranking among their categories.
- **Nominal data** are categorical variables that do not have any intrinsic order or ranking. They are used to label categories that are mutually exclusive and do not follow a specific sequence.

In our Dataset we have three Categorical Features:**Relative Velocity km per sec,Orbital Period,Orbit Uncertainity .** Also our dataset has the target **'Hazardous'** as categorical(True and False).

Relative Velocity km per sec,Orbital Period,Orbit Uncertainity features data that follows an order or we can say ranking (like very slow,slow,fast and very fast, low,medium and high). So for such a feature we used sklearn **OrdinalEncoder()**. The code for that is given below.

```
ord_data=[['Very Slow','Slow','Fast','Very Fast']]

ord_data1=[['Low','Medium','High']]

from sklearn.preprocessing import OrdinalEncoder

le=OrdinalEncoder(categories=ord_data)

le1=OrdinalEncoder(categories=ord_data1)

le2=OrdinalEncoder(categories=ord_data1)

le.fit(df3[['Relative Velocity km per sec']])

le1.fit(df3[['Orbital Period']])

le2.fit(df3[['Orbit Uncertainity']])

df3['Relative Velocity km per sec']=le.transform(df3[['Relative Velocity km per sec']])

df3['Orbital Period']=le1.transform(df3[['Orbital Period']])

df3['Orbit Uncertainity']=le2.transform(df3[['Orbit Uncertainity']])
```

On the other hand Target Column **'Hazardous'** has nothing related to order or ranking(True and False don't show any order) and is thus nominal. So for modifying it we used sklearn's **LabelEncoder()**. Code for it is given below.

```
from sklearn.preprocessing import LabelEncoder

label_encoder = LabelEncoder()

encoded_labels = label_encoder.fit_transform(df3['Hazardous'])

df3['Hazardous'] = encoded_labels
```

4. HAZARDOUS CLASSIFICATION :

For making a Classifier for predicting whether an asteroid is Hazardous or not we will be using Support Vector Machine(Classification) as it works well for categorical and imbalance dataset.

- First we did the **Feature Scaling (Normalisation)** using **MinMaxScalar()**.This fixes the values of every feature between 0 and 1 which helps the model to train faster and efficiently.

```
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()
X_scaled = scaler.fit_transform(X)
```

- As our dataset is Imbalance we used **SMOTE** technique to Increase the **Minority** class data.Instead of duplicating existing samples, it generates new synthetic samples by interpolating between the selected sample and its neighbors. This is done by taking a random point between the feature values of the original sample and its neighbor. This Leads to improved performance

```
smote = SMOTE()
X_resampled, y_resampled = smote.fit_resample(X_scaled, y)
```

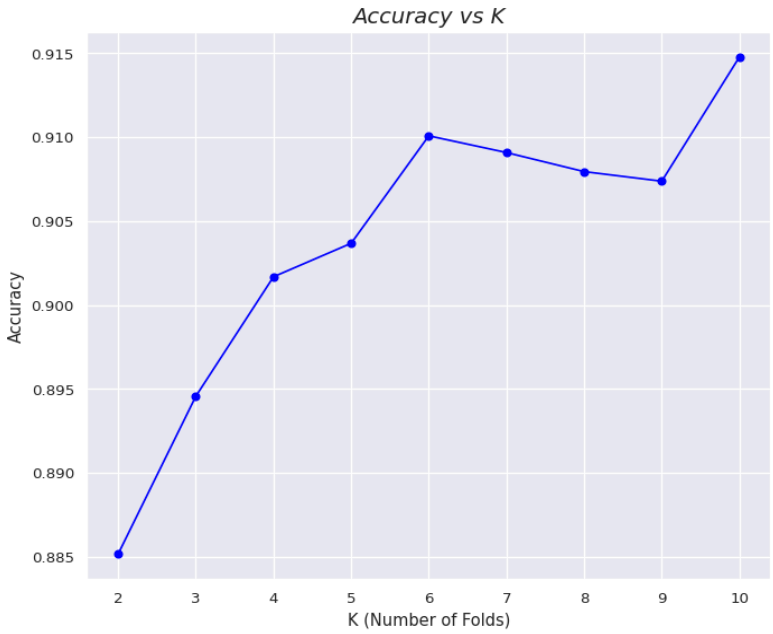
- We then used **K-Fold Cross Validation to get** the best K for **Hyperparameter** tuning.
  K-Fold Cross Validation is a resampling procedure used to evaluate machine learning models on a limited data sample. The process involves splitting the dataset into K number of folds (subsets). It ensures that the performance metrics of your model are reliable and not dependent on a specific split of your data. It provides a better estimate of how the model would perform on unseen data. Thus knowling the best K for model training is important.
  We ran the **K-Fold Cross Validation** Algorithm for k=2 to 10 and recorded the **accuracy and loss** for each. The k with better accuracy and loss is chosen for Extracting Important features from dataset and for hyperparameter Tuning. The code for K-fold cross validation is given below. We have used **XGboost** as it is faster and more efficient.

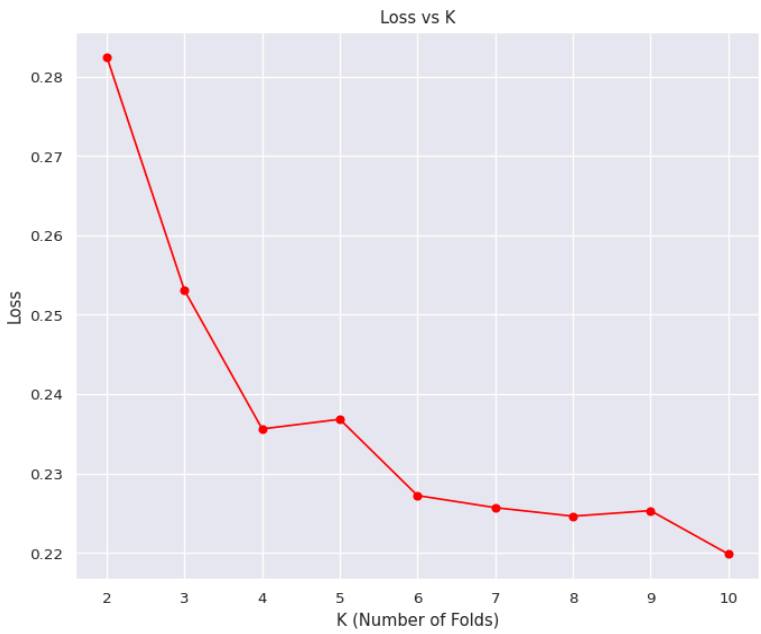
```
import xgboost as xgb
kf_scores = {}
for k in range(2, 11):
    kf = KFold(n_splits=k, shuffle=True, random_state=42)
    model = xgb.XGBClassifier(scale_pos_weight=len(y_resampled[y_resampled
== 0]) / len(y_resampled[y_resampled == 1]) ,random_state=42)

    losses = []
    accuracies = []
```

```
    for train_index, val_index in kf.split(X_resampled):
        X_train, X_val = X_resampled[train_index], X_resampled[val_index]
        y_train, y_val = y_resampled[train_index], y_resampled[val_index]

        model.fit(X_train,y_train,eval_set=[(X_val, y_val)], verbose=False)

        y_pred = model.predict(X_val)
        accuracy = accuracy_score(y_val, y_pred)
        loss = log_loss(y_val, model.predict_proba(X_val))

        accuracies.append(accuracy)
        losses.append(loss)

    kf_scores[k] = {'accuracy': np.mean(accuracies), 'loss':
np.mean(losses)}

k_values = list(kf_scores.keys())
accuracies = [kf_scores[k]['accuracy'] for k in k_values]
losses = [kf_scores[k]['loss'] for k in k_values]
```

We Then Plotted the Graphs of accuracy and Losses

```
plt.figure(figsize=(14, 6))

plt.subplot(1, 2, 1)
plt.plot(k_values, accuracies, marker='o', color='b')
plt.title('Accuracy vs K')
plt.xlabel('K (Number of Folds)')
plt.ylabel('Accuracy')
plt.grid(True)

plt.subplot(1, 2, 2)
plt.plot(k_values, losses, marker='o', color='r')
plt.title('Loss vs K')
plt.xlabel('K (Number of Folds)')
plt.ylabel('Loss')
plt.grid(True)

plt.tight_layout()
plt.show()
best_k=np.argmax(accuracies) + 2
```





- After Getting the best K , we used it for Extracting Important Features**(Using XGboost Mode**l) which can be used for Hyperparameter tuning. We are doing this because doing hyperparameter tuning on entire dataset features will take a long time. So to make it **Faster** we are doing this.

```
from xgboost import XGBClassifier
from sklearn.model_selection import KFold, train_test_split
from sklearn.metrics import accuracy_score
import numpy as np

k = best_k
kf = KFold(n_splits=k, shuffle=True, random_state=42)
model = XGBClassifier(scale_pos_weight=1, random_state=42)

feature_importances = []

for train_index, val_index in kf.split(X_resampled):
    X_train, X_val = X_resampled[train_index], X_resampled[val_index]
    y_train, y_val = y_resampled[train_index], y_resampled[val_index]


    model.fit(X_train, y_train)
    feature_importances.append(model.feature_importances_)


avg_importance = np.mean(feature_importances, axis=0)
```

```
top_features = np.argsort(avg_importance)[-10:]
```

- After Extracting Important features, for getting the best hyperparameter for training the model using SVM we split the dataset into half and did **RandomSearchCv** on **50% of data to train Faster**

```
X_tune, _, y_tune, _ = train_test_split(X_resampled, y_resampled,
test_size=0.5, random_state=42)
```

Why choose SVC over other classifiers?

SVM is preferred because it handles high-dimensional data efficiently, making it suitable for datasets with a large number of features. It performs well with both linear and non-linear separations using kernel functions, allowing flexibility in decision boundaries. SVM excels when there's a clear margin of separation between classes, minimizes overfitting, and is robust to outliers by maximizing the margin between the decision boundary and the closest data points (support vectors). Its ability to manage high-dimensional spaces makes it ideal for complex datasets.

```
from sklearn.svm import SVC
from sklearn.model_selection import RandomizedSearchCV


X_top = X_tune[:, top_features]

param_grid = {
    'C': [0.1, 1, 10, 100],
    'gamma': [1, 0.1, 0.01, 0.001],
    'kernel': ['rbf', 'linear']
}

svc = SVC(probability=True, random_state=42)
random_search = RandomizedSearchCV(svc, param_grid, n_iter=10, cv=k,
verbose=2, n_jobs=-1)


random_search.fit(X_top, y_tune)
print("Best Parameters for SVM:", random_search.best_params_)
best_para=random_search.best_params_
```

These best Parameters are **not necessary to be the best**. There can be a possibility that better parameters than them exist. Reason is that we are **not doing randomSearchCv on the entire dataset** and also using only some important feature.So it's not necessary that these parameters are best.

- We have split the dataset into Training and testing dataset.

```
np.random.seed(42)
from sklearn.model_selection import train_test_split
```

```
X_train_resampled, X_test, y_train_resampled, y_test =
train_test_split(X_resampled, y_resampled, test_size=0.2, random_state=42)
```

We Trained the Model using **SVM(classification)** and using the hyperparameters. We printed the Classification Report for the **Test dataset.** We did hyperparameter tuning to make performance better as we found that RandomSearchCv failed to give better parameters.(Reason explained above)

```
from sklearn.metrics import classification_report, roc_auc_score
svc=SVC(kernel='rbf' ,C=1 ,gamma=10)
svc.fit(X_train_resampled,y_train_resampled)
y_pred1=svc.predict(X_test)
print(classification_report(y_test,y_pred1))
roc_auc = roc_auc_score(y_test, y_pred1)
print(f'AUC: {roc_auc:.2f}')
```

Classification Report

```
              precision    recall  f1-score   support

           0       0.97      1.00      0.98       735
           1       1.00      0.97      0.98       671

    accuracy                           0.98      1406
   macro avg       0.98      0.98      0.98      1406
weighted avg       0.98      0.98      0.98      1406

AUC: 0.98
```

We got Accuracy of **98% and F1 score as 0.98 i.e 98% on our Test Dataset.**

For Checking whether our model has **overfitted** or not we Also calculated the **f1 score** for both train and test dataset.

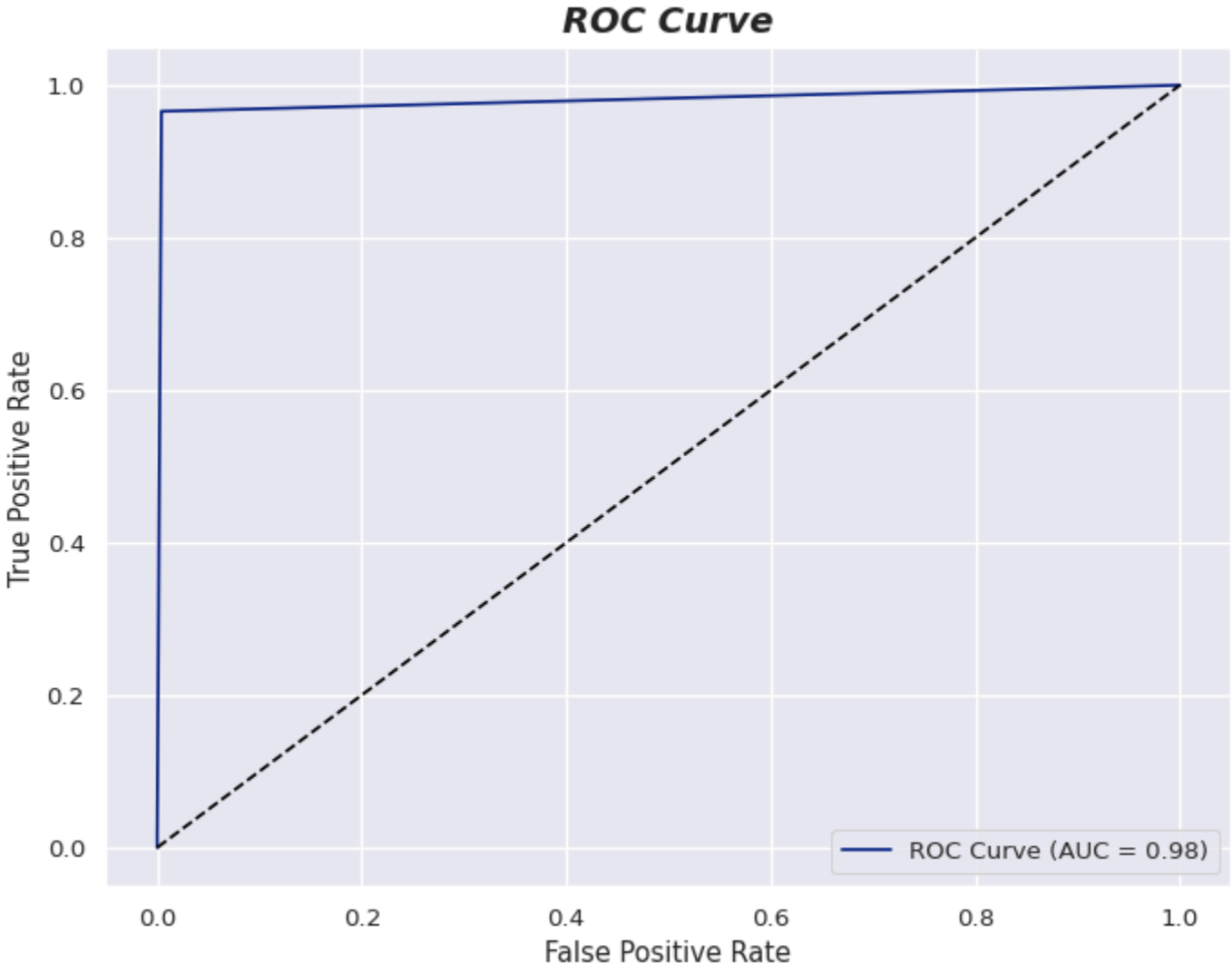
```
from sklearn.metrics import classification_report, roc_auc_score, f1_score
y_pred_train=svc.predict(X_train_resampled)
train_f1 = f1_score(y_train_resampled, y_pred_train)
test_f1 = f1_score(y_test, y_pred1)
print(f'Train F1 Score: {train_f1:.2f}')
print(f'Test F1 Score: {test_f1:.2f}')
print(train_f1-test_f1)
```

We got the difference of F1 Score of train and test as 0.01966 i.e is1.966% which shows there is no overfitting .

```
Train F1 Score: 1.00
Test F1 Score: 0.98
0.01966717095310133
```

We Plotted the ROC Curve to Quantify the performance of our model.

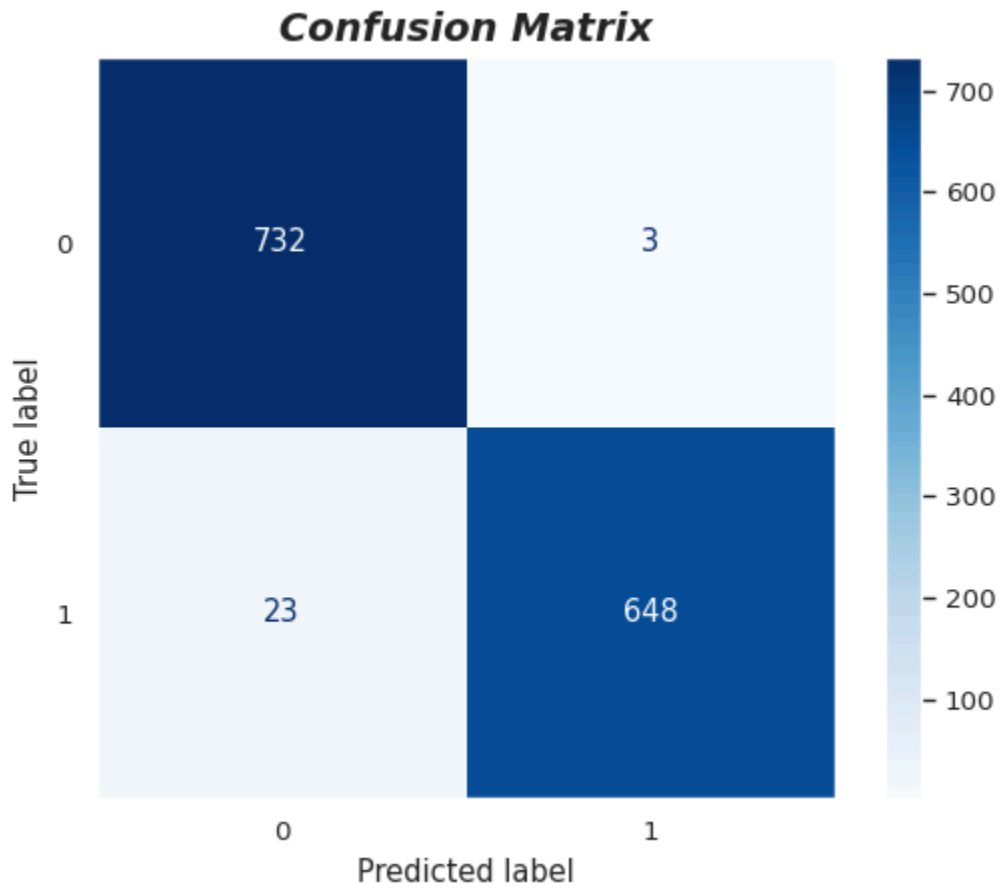
```
from sklearn.metrics import roc_curve
import matplotlib.pyplot as plt

fpr, tpr, thresholds = roc_curve(y_test, y_pred1)

plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, label=f'ROC Curve (AUC = {roc_auc:.2f})')
plt.plot([0, 1], [0, 1], 'k--')  # Diagonal line
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend(loc='lower right')
plt.grid()
plt.show()
```



- The **ROC Curve** value close to **1 (0.98)** shows that our model is performing well on our test dataset.

We Plotted the **Confusion Matrix** to quantify the model performance.

```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

cm = confusion_matrix(y_test, y_pred1)
disp = ConfusionMatrixDisplay(confusion_matrix=cm)
disp.plot(cmap='Blues')
plt.title('Confusion Matrix')
plt.show()
```



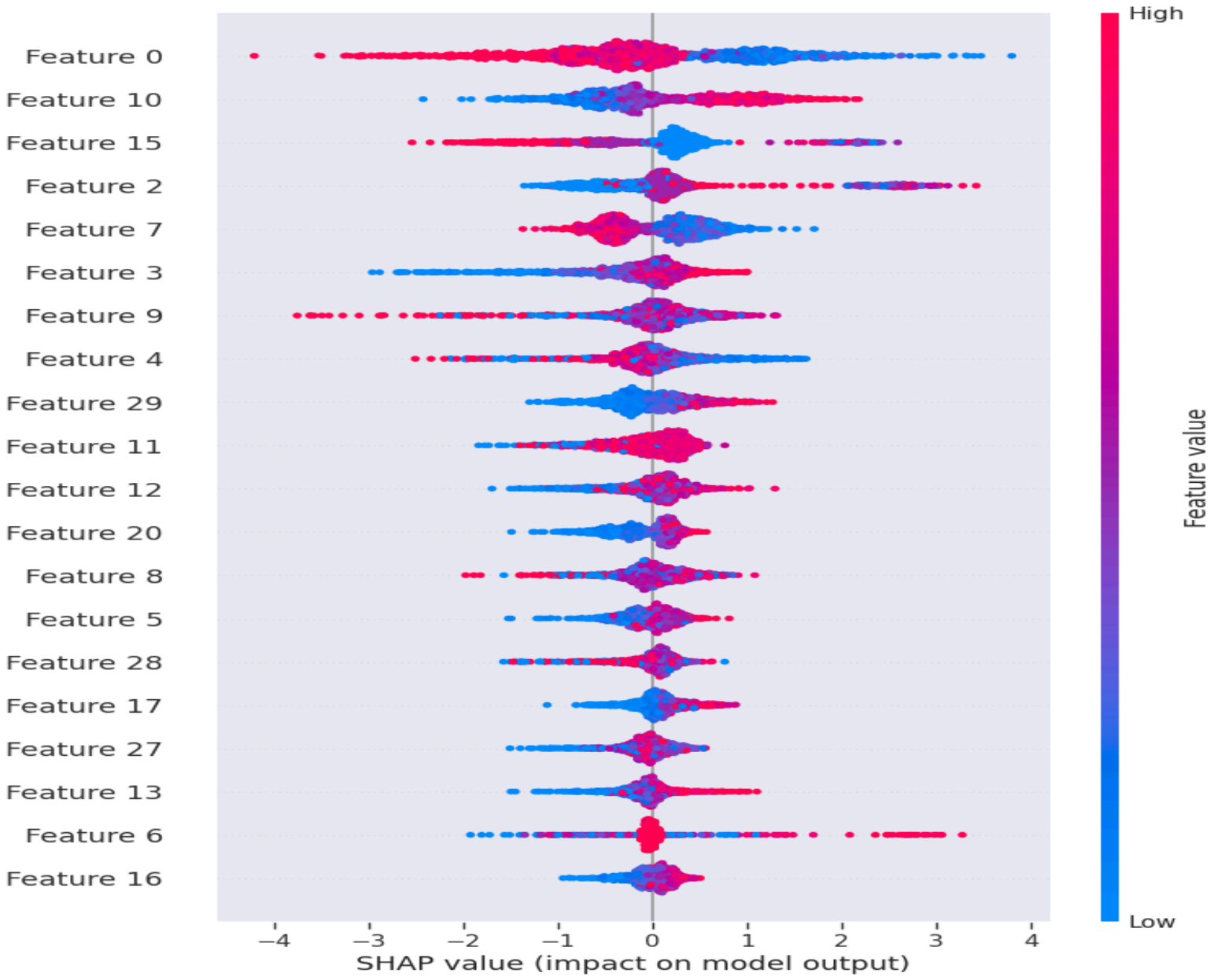
From The Confusion Matrix we can tell that our model **732** outputs as False which are actually False. Models predict **648** output as True which are actually True. Model predicts **23** outputs as False which are Actually True and Predicts **3** outputs as True which are actually False. Overall we can see our model is performing well on the Test Set.

For Visualizing the importance of each feature we used **Shap** and **Permutation Importance.**
Code for shap is:

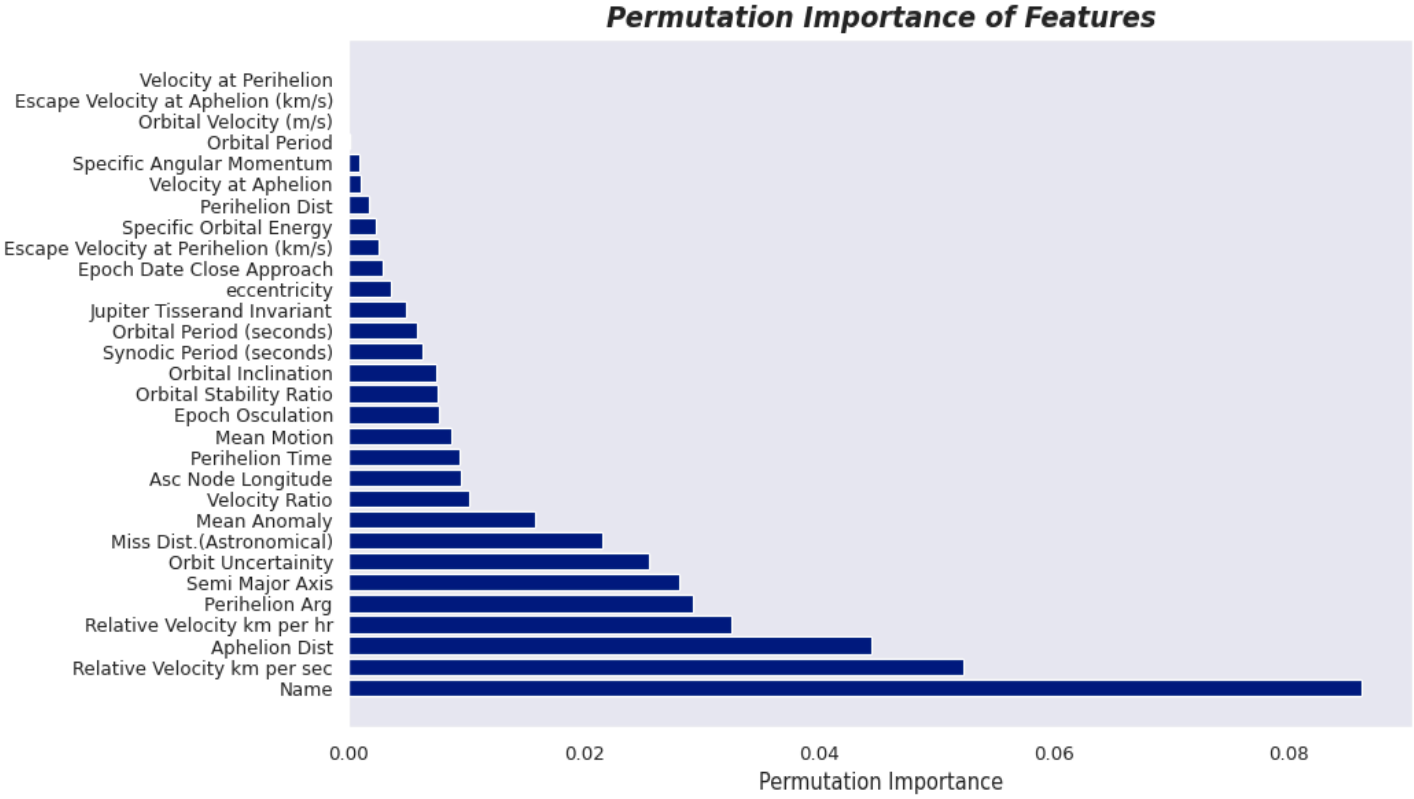
```
import shap

# Creating a SHAP explainer using the trained model
```

```
explainer = shap.Explainer(model)
# Calculating SHAP values for the test set
shap_values = explainer(X_test)
# Generating a summary plot of SHAP values to visualize feature importance
shap.summary_plot(shap_values, X_test)
```

SHAP calculates the importance of a feature by evaluating how much the model's prediction would change if that feature were removed or its value was changed.Higher absolute SHAP values indicate that the feature has a larger impact on the model's output.
SHAP values can be positive or negative, showing whether a feature pushes the prediction higher or lower. For example, if the SHAP value of a feature is positive, it means that this feature is contributing positively to the prediction. If it is negative, it means the feature is reducing the prediction value.



Permutation Importance measures Important features by randomly shuffling a particular Feature keeping other features fixed and observing performance of the model. If performance changes a lot it means that feature is important.
In our dataset we can see the importance feature as follows:



5. ANOMALY DETECTION :

- ***Using an Inbuilt Library (IsolationForest) :***
  Choosing the Algorithm: We chose **Isolation Forest**, an efficient algorithm for detecting anomalies by isolating data points through random partitions.
  Parameter Tuning:

  - *Contamination*: This parameter controls the proportion of the dataset expected to be anomalies. I set this based on domain knowledge or experimentation. For example, if I expect 8% of the data to be anomalous, I set `contamination=0.08`.
  - `n_estimators`: This defines the number of trees in the forest. I used 100 estimators for better accuracy.

  Application:

  - We fit the Isolation Forest model on the feature set of my dataset, excluding any target labels or identifiers.

- The algorithm assigns each data point an anomaly score, with higher scores indicating a higher likelihood of being an outlier.

```
from sklearn.ensemble import IsolationForest

iso_forest=IsolationForest(contamination=0.08,n_estimators=100,random_state=42
)
df3['if_anomaly_library'] = iso_forest.fit_predict(X_scaled)
df3['if_anomaly_library'] = df3['if_anomaly_library'].apply(lambda x: 1 if
x==-1 else 0)

num_of_anomalies_lib = df3['if_anomaly_library'].sum()
print(f"Number of anomalies detected by IsolationForest:
{num_of_anomalies_lib}")
```

Explanation:

- The Isolation Forest model was applied to the dataset. The contamination parameter was set to 0.08, assuming 8% of the data could be anomalies.
- The model predicted anomalies and the results were stored as a new column "`if_anomaly_library`". A value of 1 indicates an anomaly, and 0 indicates normal data.
- The total number of anomalies detected was printed at the end.

- ***Using a Custom made Algorithm(Z-Score approach) :***

Choosing the Algorithm: We chose a custom Z-Score-based approach, where anomalies are identified based on how many standard deviations a data point is from the mean of the dataset.

Parameter Tuning:

- *Threshold:* We set a Z-Score threshold (e.g., 4 standard deviations). Any data point with an absolute Z-Score greater than this threshold is considered an anomaly. The threshold can be adjusted based on the distribution of data.

Application:

- For each feature in the dataset, we calculate the Z-Score for every data point. Z-Score is defined as: $Z=(Xi-\mu)/\sigma$ where Xi is the data point, μ is the mean, and σ is the standard deviation of the feature.
- If the absolute Z-Score of any data point exceeds the threshold (e.g., $|Z| > 4$), it is flagged as an anomaly.

```
def custom_z_score(X):
```

```
    z_scores = np.zeros(X.shape)

    for feature in range(X.shape[1]):

        mean = np.mean(X[:, feature])

        std_dev = np.std(X[:, feature])

        if std_dev == 0:

            z_scores[:, feature] = 0

        else:

            z_scores[:, feature] = (X[:, feature] - mean) / std_dev

    return z_scores

threshold = 4

z_scores = custom_z_score(X_scaled, threshold)

df3['if_anomaly_custom'] = (np.abs(z_scores) >
threshold).any(axis=1).astype(int)

num_anomalies_custom = df3['if_anomaly_custom'].sum()

print(f"Number of anomalies detected by custom Z-score method:
{num_anomalies_custom}")
```

Explanation:

- The Z-Score for each feature was calculated using the mean and standard deviation of the dataset.
- A threshold of 4 standard deviations was used to identify outliers. Any data point with an absolute Z-Score greater than 4 was flagged as an anomaly.
- The anomalies were stored in the **"`if_anomaly_custom`"** column, where 1 indicates an anomaly and `0` indicates normal data.
- Finally, the total number of anomalies detected using this custom Z-Score method was printed.

- ***Observations and Analysis :***

Anomalies Detected by Individual Algorithms **:**After applying both the **Isolation Forest** (inbuilt library) and the **custom Z-Score** method for anomaly detection, the following results were obtained:

- **Isolation Forest**:

    - Number of anomalies detected: 337
- **Custom Z-Score Method**:
    - Number of anomalies detected: 178

Here, we can observe that **Isolation Forest** detected slightly more anomalies compared to the custom Z-Score method. This is likely due to the Isolation Forest's ability to better capture anomalies in multi-dimensional data, while the Z-Score method is more focused on statistical outliers.

Combined Anomaly Detection Results : To further analyze the results, we compared both algorithms by identifying the number of common anomalies flagged. The results are as follows:

- **Anomalies detected by both algorithms**: 80

  Reason: This can be attributed to the different underlying assumptions of the two methods. Isolation Forest isolates anomalies based on the data's structure, while Z-Score relies on deviations from the mean.

***Confusion Matrix Analysis* :**To understand how the two algorithms compare, we plotted a **Confusion Matrix** to evaluate the overlap between the anomalies detected by both methods:

True Positives (TP): Number of data points flagged as anomalies by both **Isolation Forest** and **Z-Score**.
False Positives (FP): Data points flagged as anomalies by **Isolation Forest** but considered normal by **Z-Score**.
False Negatives (FN): Data points flagged as anomalies by **Z-Score** but considered normal by **Isolation Forest**.
True Negatives (TN): Data points considered normal by both algorithms.