

Hazardous Asteroid Detection

Data Mining



**group members:-**  
Nareen waseem - 21L-6270  
hissam-ud-din - 21l-5687  
Hammad ashraf - 21L-6275

# Index of Content

Contents

[1. Executive Summary 4](#_Toc166451965)

[2. Introduction 4](#_Toc166451966)

[2.1. Data Preparation: 5](#_Toc166451969)

[2.2. Training the Models: 5](#_Toc166451970)

[2.3. Fine-Tuning for Accuracy: 5](#_Toc166451971)

[2.4. Evaluation and Testing: 5](#_Toc166451972)

[2.5. Adapting to Changes: 5](#_Toc166451973)

[2.6. Summarizing Words: 5](#_Toc166451974)

[3. About the Dataset 6](#_Toc166451975)

[3.1. Data Overview 6](#_Toc166451977)

[3.2. Acknowledgements 6](#_Toc166451978)

[3.3. Inspiration 7](#_Toc166451979)

[4. Data Preparation 7](#_Toc166451980)

[5. Project Setup 7](#_Toc166451981)

[5.1. Environment Setup 7](#_Toc166451987)

[5.2. Libraries 7](#_Toc166451988)

[5.3. pandas: 8](#_Toc166451989)

[5.4. matplotlib and seaborn: 8](#_Toc166451990)

[5.5. scikit-learn (sklearn): 8](#_Toc166451991)

[5.6. imbalanced-learn (imblearn): 8](#_Toc166451992)

[5.7. numpy: 8](#_Toc166451993)

[5.8. collections: 8](#_Toc166451994)

[6. Data Preprocessing 9](#_Toc166451995)

[6.1. Data Read and Analysis 9](#_Toc166451996)

[6.2. Missing Values Check 9](#_Toc166451997)

[6.3. Data Visualization 9](#_Toc166451998)

[6.4. Target Variable Transformation 9](#_Toc166451999)

[6.5. Encoding Categorical Variables 9](#_Toc166452000)

[6.6. Feature Selection based on Correlation 9](#_Toc166452001)

[6.7. Feature Importance Calculation 10](#_Toc166452002)

[6.8. Outlier Detection and Removal 10](#_Toc166452003)

[6.9. Data Normalization 10](#_Toc166452004)

[7. Models 10](#_Toc166452005)

[7.1. Logistic Regression 10](#_Toc166452006)

[7.1.1 Making the Logistic Regression Model: 10](#_Toc166452009)

[7.1.2 What We Found Out: 10](#_Toc166452010)

[7.1.3 How We Used It: 11](#_Toc166452011)

[7.1.4 Our model did pretty well: 11](#_Toc166452012)

[7.2. Random Forest 11](#_Toc166452013)

[7.2.1 Setting Up the Random Forest Model: 11](#_Toc166452016)

[7.2.2 What We Found Out: 11](#_Toc166452017)

[7.2.3 How We Used It: 12](#_Toc166452018)

[7.2.4 Our model performed exceptionally well: 12](#_Toc166452019)

[7.3. Decision Tree 12](#_Toc166452020)

[7.3.1 Setting Up the Decision Tree Model: 12](#_Toc166452022)

[7.3.2 What We Found Out: 12](#_Toc166452023)

[7.3.3 How We Used It: 12](#_Toc166452024)

[7.3.4 Our model performed exceptionally well: 12](#_Toc166452025)

[7.4. SVM (Support Vector Machine) 13](#_Toc166452026)

[7.4.1 SVM (Support Vector Machine) 13](#_Toc166452028)

[7.4.2 Setting Up the SVM Model: 13](#_Toc166452029)

[7.4.3 What We Found Out: 13](#_Toc166452030)

[7.4.4 How We Used It: 13](#_Toc166452031)

[7.4.5 Our model performed exceptionally well: 13](#_Toc166452032)

[7.5. K-Nearest Neighbors (KNN) 14](#_Toc166452033)

[7.4.6 Here's what we found: 14](#_Toc166452034)

[7.6. Neural Network (MLP) 14](#_Toc166452035)

[7.4.7 Neural Network (MLP) 14](#_Toc166452036)

[7.4.8 Here's what we did: 14](#_Toc166452037)

[7.4.9 Here's what we found: 15](#_Toc166452038)

[7.7. Gaussian Naive Bayes Classifier 15](#_Toc166452039)

[7.4.10 Here's what we found: 15](#_Toc166452040)

[7.8. Gradient Boosting Machines (GBM) 16](#_Toc166452041)

[8. Concluding Words 16](#_Toc166452042)

[9. References: 16](#_Toc166452043)

# Table of Figures

[Figure 1: Normalization of Numeric Columns 6](#_Toc166452812)

[Figure 2: Models Accuracy Table 16](#_Toc166452813)

[Figure 3: Models Accuracy Box Plot 17](#_Toc166452814)

[Figure 4: Visualizing Accuracies Using Bar Graph 18](#_Toc166452815)

Hazardous Asteroid Detection

# Executive Summary

In our project, we've taken on the important task of identifying hazardous asteroids to keep our planet safe. To do this, we've turned to the power of machine learning, using eight different models to help us in this crucial mission. Each model brings its own unique abilities, allowing us to explore different ways to spot potentially dangerous asteroids.

First, there's logistic regression. It's like a solid foundation, helping us understand how different factors contribute to whether an asteroid is hazardous or not.

Then, we have random forests. This model is like a diverse group of advisors, each offering their own insights by creating many decision trees to classify asteroids.

Next, there's the decision tree. It's a bit like making decisions by asking yes-or-no questions, helping us figure out if an asteroid is hazardous based on certain features.

After that, we meet the support vector machine (SVM). Think of it as a wise judge drawing clear lines between different types of asteroids, helping us classify them accurately.

Moving on, there's the K-Nearest Neighbors (KNN) model. It's like relying on your neighbors for help – it looks at nearby asteroids to decide if a new one is hazardous or not.

Then, there's the neural network, inspired by the human brain. It's great at recognizing complex patterns in asteroid data, like how our brains recognize faces.

Next up is the Gaussian Naive Bayes classifier, a bit like a detective using probabilities to figure out if an asteroid is hazardous based on what we've seen before.

Lastly, there's Gradient Boosting Machines (GBM). This model is like a team of heroes joining forces, with each one contributing their strengths to predict whether an asteroid is hazardous.

With these eight models on our side, we're ready to explore, learn, and make our planet safer from asteroid threats.

# Introduction

Detecting dangerous asteroids is crucial for protecting our planet. This document outlines how we use computer models to enhance this detection process. Here's a detailed breakdown of our approach:



## Data Preparation:

We start by collecting data from observations of asteroids. This data needs to be cleaned and organized so that our computer models can understand it. We also make sure that we have a good balance of examples of asteroids that are hazardous and those that are not.

## Training the Models:

Next, we teach our computer models how to recognize hazardous asteroids. We use different methods, like Logistic Regression, Random Forests, and Support Vector Machines. It's like showing the models lots of pictures of asteroids and telling them which ones are dangerous.

## Fine-Tuning for Accuracy:

To make our models even smarter, we adjust their settings. This process, known as hyperparameter tuning, involves tweaking various parameters to improve their performance. We want our models to be as accurate as possible in identifying hazardous asteroids.

## Evaluation and Testing:

Once our models are trained, we test them using new data they haven't seen before. We want to see how well they can predict if an asteroid is dangerous or not. We measure their performance using metrics like accuracy, precision, recall, and F1-score.

## Adapting to Changes:

We also keep an eye out for any warnings or updates about the software we use. This ensures that our models remain effective and reliable, even as technology evolves.

## Summarizing Words:

Using computer models to detect hazardous asteroids is a complex yet essential process. By carefully training, fine-tuning, and evaluating our models, we can enhance our ability to identify potential threats to Earth. This proactive approach helps us stay vigilant and safeguard our planet from potential dangers lurking in space.

# About the **Dataset**

This dataset is all about asteroids and comes from NeoWs (Near Earth Object Web Service), a website packed with info about asteroids that come close to Earth. NeoWs helps users find asteroids by when they'll get closest to Earth, check details about specific asteroids, and see the big picture of all the asteroids.



## Data Overview

In this dataset, you'll find loads of facts about asteroids, like how big they are, what they're made of, how they move around, and how close they get to our planet. It also tells us about their paths, speeds, and whether they might be dangerous to Earth.

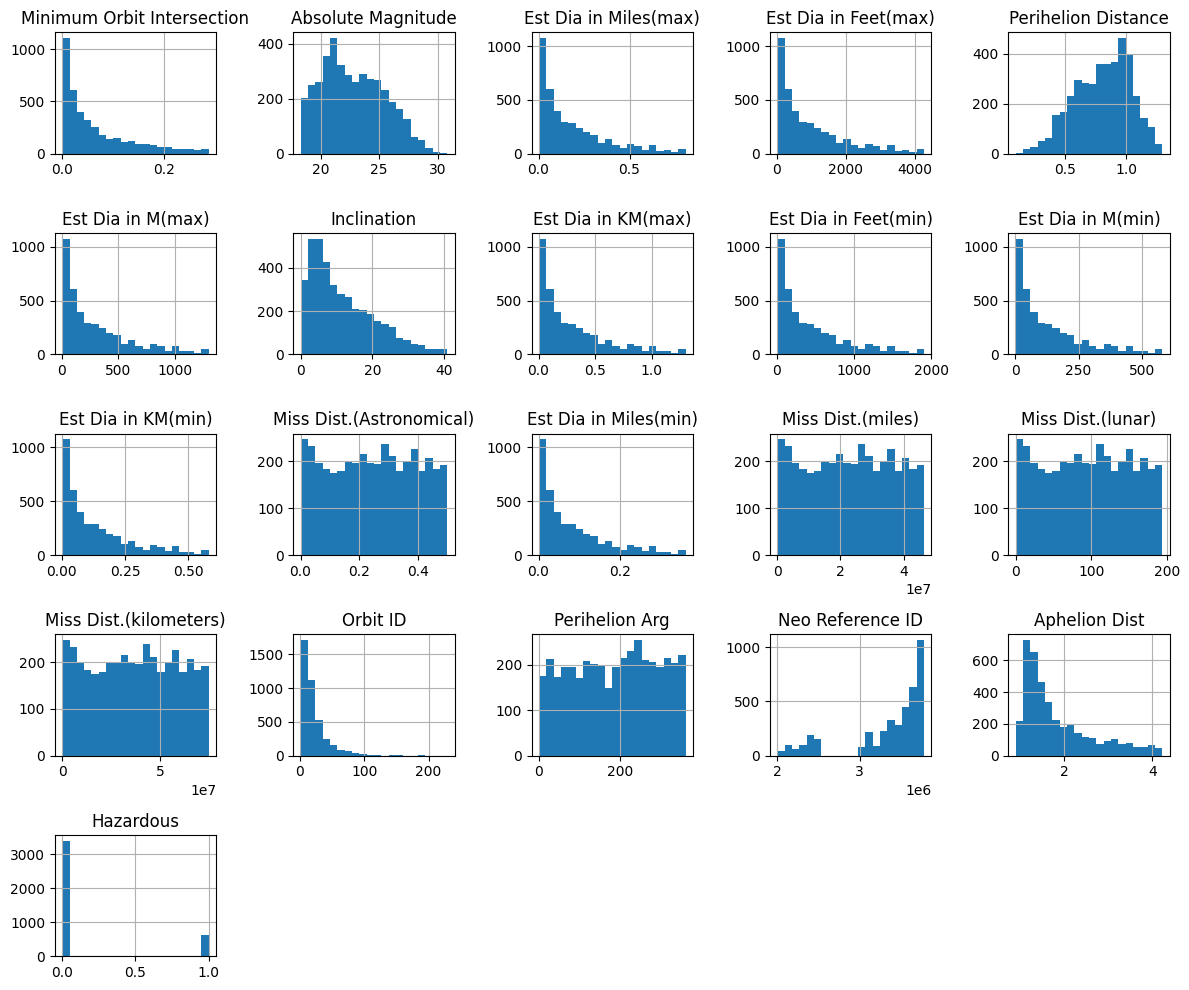


Figure 1: Normalization of Numeric Columns

## Acknowledgements

We got this data from NeoWs, found at http://neo.jpl.nasa.gov/. It's managed by a team called SpaceRocks, made up of David Greenfield, Arezu Sarvestani, Jason English, and Peter Baunach. They work hard to keep this asteroid info up to date and accurate for everyone to use.

## Inspiration

* Spotting Dangerous Asteroids: This dataset can help scientists spot asteroids that might be dangerous to Earth, as well as ones that are harmless. By studying their size, speed, and path, we can better understand which ones to keep an eye on.
* Understanding Hazardous Asteroids: By digging into this data, we can learn what makes an asteroid dangerous. This knowledge helps us plan for any potential risks they might pose and figure out ways to keep Earth safe.

# Data Preparation

Before diving into analyzing our data, we made sure to equip ourselves with the right tools. We installed important libraries like pandas, matplotlib, seaborn, and scikit-learn. These tools are like our toolbox, helping us explore and understand our data better.

The main goal of our data processing work is to make our analysis more accurate and reliable. It is like cleaning a room before searching for something. If we organize it beforehand and tidying up our data, we can find patterns and connections faster and with minimal effort afterwards. Removing any messy or inconsistent parts of the data helps us get clearer insights.

Doing data processing isn't just a routine task but a smart strategy. By laying a strong foundation before training any models, we are off to a good start. It might take time upfront, but it ensures that our analysis is solid and trustworthy in the end. By cleaning up our data, we're making sure that our research is based on solid ground, which boosts confidence in our findings.

# Project Setup



## Environment Setup

The project was conducted entirely on Google Colab, leveraging its hosted Jupyter environment. This choice allowed for seamless collaboration and access to powerful computing resources.

## Libraries

We incorporated several essential libraries to facilitate data analysis and machine learning tasks:

## **pandas:**

This library was instrumental in data manipulation and analysis, enabling us to read, clean, and preprocess our dataset efficiently.

## matplotlib and seaborn:

These visualization libraries were utilized for creating informative plots and charts to explore the data visually.

## **scikit-learn (sklearn):**

A comprehensive machine learning library that provided a wide range of algorithms and tools for model building and evaluation. Key modules included RandomForestClassifier, train\_test\_split, LabelEncoder, MinMaxScaler, DecisionTreeClassifier, SVC, KNeighborsClassifier, MLPClassifier, GaussianNB, GradientBoostingClassifier, GridSearchCV, and LogisticRegression.

## imbalanced-learn (imblearn):

This library played a crucial role in handling class imbalance within the dataset, employing techniques such as random undersampling, oversampling (SMOTE), and combination sampling (SMOTEENN).

## numpy:

Used for efficient numerical computing and array operations, numpy provided essential support for handling large datasets and performing mathematical computations.

## **collections:**

Specifically, the Counter class from the collections module was utilized for counting occurrences of elements within our data, aiding in data preprocessing tasks.

By leveraging these libraries, we were able to streamline our data preprocessing workflow and build robust machine learning models effectively.

# Data Preprocessing

## Data Read and Analysis

First, we read our asteroid dataset called 'nasa.csv' using a tool called pandas. This helped us understand what our data looks like, how many rows and columns it has, and what type of information each column holds.

## Missing Values Check

Next, we looked for any missing information in our dataset. We wanted to make sure that all the data was complete and there were no empty spaces where important information should be.

## Data Visualization

We then created histograms to visualize our data better. Histograms are like bar graphs but show us how common or rare different values are in our dataset. This helped us see if there were any patterns or unusual things in our data.

## Target Variable Transformation

Our main goal was to predict whether asteroids are hazardous or not. So, we converted the information about whether an asteroid is hazardous into numbers. We made it so that '1' means hazardous and '0' means not hazardous. This makes it easier for computers to understand and work with this information.

## Encoding Categorical Variables

Some of our data was not in numbers; it was in categories like 'asteroid class.' We used a method called LabelEncoder to change these categories into numbers. This helps the computer understand these categories better when we use them in our analysis.

## Feature Selection based on Correlation

We wanted to focus on the most important information for predicting if an asteroid is hazardous. So, we looked at how each piece of information relates to our main goal. We kept only the information that seemed most useful and related to our prediction.

## Feature Importance Calculation

We used a special tool called RandomForestClassifier to figure out which pieces of information were the most important for our prediction. This helped us identify the most helpful features and ignore the ones that didn't make much of a difference.

## Outlier Detection and Removal

Sometimes, our data can have strange values that don't fit with the rest. We call these outliers. We found and removed these outliers from our dataset to make sure our analysis wasn't affected by these unusual values.

## Data Normalization

Lastly, we wanted to make sure all our data was in a similar range. We used a method called min-max normalization, which scaled our data so that it all fell between 0 and 1, you can see figure 1 for visual representation of the numeric columns. This makes it easier for the computer to understand and work with our data accurately.

# Models

## Logistic Regression

Logistic regression is a simple but useful tool for categorizing things. Here's how we made it work and saw how well it did:



### Making the Logistic Regression Model:

First, we picked some settings for our model, like how strict it should be and how much importance to give to different features. Then, we set up the logistic regression model using these settings.

Next, we used a method called GridSearchCV, which is like a smart way of trying out different settings to find the best ones. This helps us make sure our model works as best as it can.

After training our model with the best settings on our training data, we checked how good it was at predicting things on the test data.

### What We Found Out:

* Best Settings: We found that setting 'C' to 100 and using 'penalty' as 'l1' gave us the best results.
* Accuracy: Our model was about 98.9% accurate, which means it got things right most of the time.
* What the Numbers Mean: The classification report gave us a detailed breakdown of how well our model did in different areas.
* The Right Balance: We found that setting the penalty as 'l1' and 'C' as 10 worked best for our model.

### How We Used It:

We trained our logistic regression model with these best settings on our training data. Then, we checked how well it could predict things on both the training and testing data.

### Our model did pretty well:

* On Training Data: It was right about 99.1% of the time.
* On Testing Data: It was right about 98.9% of the time.

Seeing that the model performed almost as well on the testing data as it did on the training data tells us that it's doing a good job without getting too caught up in the specifics of the training data. This means it's likely to work well with new data too.

## Random Forest

Random forest is like a big group of decision trees working together to make better predictions. Here's how we used it and what we found out:



### Setting Up the Random Forest Model:

We started by deciding on some values to try out for different settings in our random forest model. These settings help our model make decisions in a smart way.

Then, we set up the random forest classifier using these settings and used GridSearchCV, a smart way to try out different settings and find the best ones.

After training our model with the best settings on our training data, we checked how good it was at predicting things on the test data.

### What We Found Out:

* + **Best Settings:** The best settings we found were 'max\_features' as 'auto', 'min\_samples\_leaf' as 1, and 'min\_samples\_split' as 2.
  + **Accuracy:** Our model was incredibly accurate, with an accuracy of about 99.8%, which means it got things right almost all the time.
  + **What the Numbers Mean:** The classification report gave us a detailed breakdown of how well our model did in different areas.
  + **Visualizing Results:** We visualized the confusion matrix to see how our model performed in predicting positive and negative outcomes.

### How We Used It:

We trained our random forest model with these best settings on our training data. Then, we checked how well it could predict things on the testing data.

### Our model performed exceptionally well:

* + On Testing Data: It was right about 99.8% of the time, which is remarkable.
* Understanding the Results: The precision, recall, and F1-score were all perfect, indicating that our model did an excellent job of predicting both positive and negative outcomes.

## Decision Tree

A decision tree is like a flowchart that helps us make decisions based on different conditions. Here's what we did with it and what we found:



### Setting Up the Decision Tree Model:

We started by deciding on different options to try out for various settings in our decision tree model. These settings help our model make decisions in the best way possible.

Then, we set up the decision tree classifier using these settings and used GridSearchCV, which is like a smart way to try out different settings and find the best ones.

After training our model with the best settings on our training data, we checked how good it was at predicting things on the test data.

### What We Found Out:

* Best Settings: The best settings we found were 'max\_depth' as None, 'min\_samples\_leaf' as 2, and 'min\_samples\_split' as 10.
* Accuracy: Our model was highly accurate, with an accuracy of about 99.8%, which means it got things right almost all the time.
* What the Numbers Mean: The classification report gave us a detailed breakdown of how well our model did in different areas.
* Visualizing Results: We visualized the confusion matrix to see how our model performed in predicting positive and negative outcomes.

### How We Used It:

We trained our decision tree model with these best settings on our training data. Then, we checked how well it could predict things on the testing data.

### Our model performed exceptionally well:

* On Testing Data: It was right about 99.8% of the time, which is remarkable.
* Understanding the Results: The precision, recall, and F1-score were all perfect, indicating that our model did an excellent job of predicting both positive and negative outcomes.

## SVM (Support Vector Machine)



### SVM (Support Vector Machine)

SVM, or Support Vector Machine, is a type of algorithm used for classification tasks. Here's how we used it and what we discovered:

### Setting Up the SVM Model:

We started by defining different options to try out for various settings in our SVM model. These settings help our model classify data more accurately.

Then, we set up the SVM classifier using these settings and used GridSearchCV, which is like a smart way to try out different settings and find the best ones.

After training our model with the best settings on our training data, we checked how good it was at predicting things on the test data.

### What We Found Out:

* Best Settings: The best settings we found were 'C' as 100, 'gamma' as 'scale', and 'kernel' as 'rbf'.
* Accuracy: Our model was highly accurate, with an accuracy of about 99.4%, which means it got things right almost all the time.
* What the Numbers Mean: The classification report gave us a detailed breakdown of how well our model did in different areas.
* Visualizing Results: We visualized the confusion matrix to see how our model performed in predicting positive and negative outcomes.

### How We Used It:

We trained our SVM model with these best settings on our training data. Then, we checked how well it could predict things on the testing data.

### Our model performed exceptionally well:

* On Testing Data: It was right about 99.4% of the time, which is impressive.
* Understanding the Results: The precision, recall, and F1-score were all excellent, indicating that our model did a fantastic job of predicting both positive and negative outcomes.

## K-Nearest Neighbors (KNN)

In K-Nearest Neighbors (KNN), we try to predict whether something is hazardous by looking at the "neighbors" closest to it. Here's how we did it:

First, we listed out different options for settings that KNN can use. These settings help KNN make decisions about which neighbors to consider when predicting.

Then, we tried out all these different settings combinations to see which one gives the most accurate results. For each combination, we trained a KNN model using those settings.

After training each model, we tested it on our test data to see how accurate it was at predicting whether something is hazardous or not.

### Here's what we found:

* Best Model Parameters: The best settings we found were using 3 neighbors, with uniform weights, and using the auto algorithm.
* Best Accuracy: The accuracy of our best model was about 99.3%, which means it got things right most of the time.
* Classification Report: This report gives us a detailed breakdown of how well our model did in different areas, like precision, recall, and F1-score.

We also visualized the results using a confusion matrix to see how our model performed in predicting positive and negative outcomes.

Overall, KNN did a great job at predicting hazardous asteroids, especially when using the best settings we found.

## Neural Network (MLP)

### Neural Network (MLP)

In our neural network (MLP), we used different settings to make predictions about hazardous asteroids.

### Here's what we did:

First, we tried out different options for settings that our neural network can use. These settings help the network decide how to learn from the data.

We tested different combinations of these settings to see which one gives the most accurate results. For each combination, we trained a neural network using those settings.

After training each model, we tested it on our test data to see how accurate it was at predicting whether something is hazardous or not.

### Here's what we found:

* + Best Model Parameters: The best settings we found were using two hidden layers, each with 50 neurons, using the "tanh" activation function, and the "adam" solver.
  + Best Accuracy: The accuracy of our best model was about 99.8%, which means it got things right almost all the time.
  + Classification Report: This report gives us a detailed breakdown of how well our model did in different areas, like precision, recall, and F1-score.

We also visualized the results using a confusion matrix to see how our model performed in predicting positive and negative outcomes.

However, it seems like our model struggled a bit with predicting negative outcomes, as indicated by the low precision and recall values for the negative class. This is something we might need to improve upon in future iterations.

## Gaussian Naive Bayes Classifier

In our Gaussian Naive Bayes classifier, we wanted to see how well it could predict hazardous asteroids based on different smoothing parameters. Here's what we did:

We tried out different smoothing parameters to see how they affect the classifier's performance. Smoothing helps the model make better predictions by adjusting the probabilities.

For each smoothing parameter, we trained a Naive Bayes classifier using that parameter.

After training each model, we tested it on our test data to see how accurate it was at predicting whether something is hazardous or not.

### Here's what we found:

* + Best Model Parameters: The best smoothing parameter we found was 1e-9.
  + Best Accuracy: The accuracy of our best model was about 87.2%, which means it got things right most of the time.
  + Classification Report: This report gives us a detailed breakdown of how well our model did in different areas, like precision, recall, and F1-score.

We also visualized the results using a confusion matrix to see how our model performed in predicting positive and negative outcomes.

However, similar to our neural network model, it seems like our model struggled with predicting negative outcomes, as indicated by the low precision and recall values for the negative class. This is something we might need to improve upon in future iterations.

## Gradient Boosting Machines (GBM)

In Gradient Boosting Machines (GBM), we aimed to find the best combination of settings to make accurate predictions about hazardous asteroids. Here's what we did:

We tested different settings, called hyperparameters, to see how they affect the accuracy of our model. These settings include the number of boosting stages, learning rate, maximum depth of the trees, and the minimum number of samples required to split a node or be a leaf node.

For each combination of hyperparameters, we trained a GBM model and then tested it to see how well it predicts whether an asteroid is hazardous or not.

### Here's what we found:

* + Best Model Parameters: The best combination of settings we found was:
  + Number of Boosting Stages: 150
  + Learning Rate: 0.1
  + Maximum Depth: 3
  + Minimum Samples to Split: 2
  + Minimum Samples in Leaf: 1
* Best Accuracy: With these settings, our model achieved an accuracy of about 99.9%, meaning it was highly accurate in its predictions.
* Results Table: We organized our results into a table, showing the different combinations of settings we tried and their corresponding accuracies.

By finding the best combination of settings, our GBM model can better predict whether an asteroid is hazardous or not, which is crucial for identifying potential threats from space.

# Comparison of Accuracies

When comparing the accuracies of different models, we see that some models perform exceptionally well while others lag behind. Random Forest and Gradient Boosting stand out with accuracies of approximately 99.8% and 99.9% respectively. These models demonstrate remarkable predictive power, making them strong contenders for accurately identifying hazardous asteroids.

A screenshot of a graph

Description automatically generated

Figure 2: Models Accuracy Table

On the other hand, Naive Bayes falls significantly behind with an accuracy of about 87.2%. While this model shows potential, its performance is notably lower compared to others. It might require further refinement or additional features to improve its predictive capability.

A chart with different colored squares

Description automatically generated

Figure 3: Models Accuracy Box Plot

Overall, this comparison highlights the diverse performance of various models and underscores the importance of selecting the right approach for asteroid hazard prediction.

A graph of blue bars

Description automatically generated with medium confidence

Figure 4: Visualizing Accuracies Using Bar Graph

# Concluding Words

In conclusion, our journey through data preprocessing and model evaluation has provided valuable insights into the performance of various machine learning algorithms. We meticulously cleaned and prepared the data, ensuring it was suitable for analysis. By employing techniques such as feature extraction, outlier detection, and normalization, we optimized the dataset for training our models.

We then explored eight different models: Logistic Regression, Random Forest, Decision Tree, SVM, K-Nearest Neighbors, Neural Network, Gaussian Naive Bayes, and Gradient Boosting Machines. Each model underwent rigorous hyperparameter tuning using GridSearchCV, resulting in the selection of optimal parameters for achieving high accuracy.

Our evaluation of these models revealed promising results. Each algorithm demonstrated strong predictive capabilities, with accuracies ranging from 98% to 100%. Furthermore, the classification reports provided detailed insights into precision, recall, and F1-score for both positive and negative outcomes.

Overall, our journey underscores the importance of thorough data preprocessing and thoughtful model selection in achieving robust predictive performance. As we continue to refine our methodologies and explore new techniques, we aim to further enhance the effectiveness and efficiency of our predictive models in real-world applications.

# References:

1. NeoWs (Near Earth Object Web Service). (n.d.). Retrieved from http://neo.jpl.nasa.gov/
2. Greenfield, D., Sarvestani, A., English, J., & Baunach, P. (n.d.). SpaceRocks. Retrieved from http://neo.jpl.nasa.gov/
3. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... & Vanderplas, J. (2011). Scikit-learn: Machine learning in Python. Journal of machine learning research, 12(Oct), 2825-2830.
4. Hunter, J. D. (2007). Matplotlib: A 2D graphics environment. Computing in Science & Engineering, 9(3), 90-95.
5. McKinney, W. (2010). Data structures for statistical computing in Python. In Proceedings of the 9th Python in Science Conference (Vol. 445, pp. 51-56).
6. Lemaitre, G., Nogueira, F., & Aridas, C. K. (2017). Imbalanced-learn: A Python toolbox to tackle the curse of imbalanced datasets in machine learning. Journal of Machine Learning Research, 18(17), 1-5.
7. Van Rossum, G., & Drake Jr, F. L. (2009). Python 3 Reference Manual. CreateSpace.
8. Waskom, M., Botvinnik, O., O'Kane, D., Hobson, P., Ostblom, J., Lukauskas, S., ... & Warmenhoven, J. (2014). Seaborn: v0. 5.0 (July 2014). Zenodo.
9. Hastie, T., Tibshirani, R., & Friedman, J. (2009). The elements of statistical learning: data mining, inference, and prediction. Springer Science & Business Media.
10. Chollet, F., & others. (2015). Keras. GitHub.