Predicting Hazardous Nearest Earth Objects (1910-2024)

Ahsan Khan

Introduction to Machine Learning: Supervised Learning

University of Colorado Boulder

Dataset Overview

- We're working with the NASA | Nearest Earth Objects (1910-2024) dataset from Kaggle.
- This dataset contains over 338,000 records of NEOs, each described by various parameters such as size, velocity, and distance from Earth.

Project Objective

The primary objective of our project is to develop a machine learning model capable of predicting the "is_hazardous" classification of NEOs

Methodology

- Data Cleaning and Exploratory Data Analysis (EDA)
- Feature Engineering
- Model Building
- Model Evaluation
- Hyperparameter Tuning
- Conclusion and Recommendations

Load the Data

```
# Load the dataset
file_path = 'nearest-earth-objects(1910-2024).csv'
neo_df = pd.read_csv(file_path)

# Display the first few rows of the dataset
neo_df.head()
```

[1]:

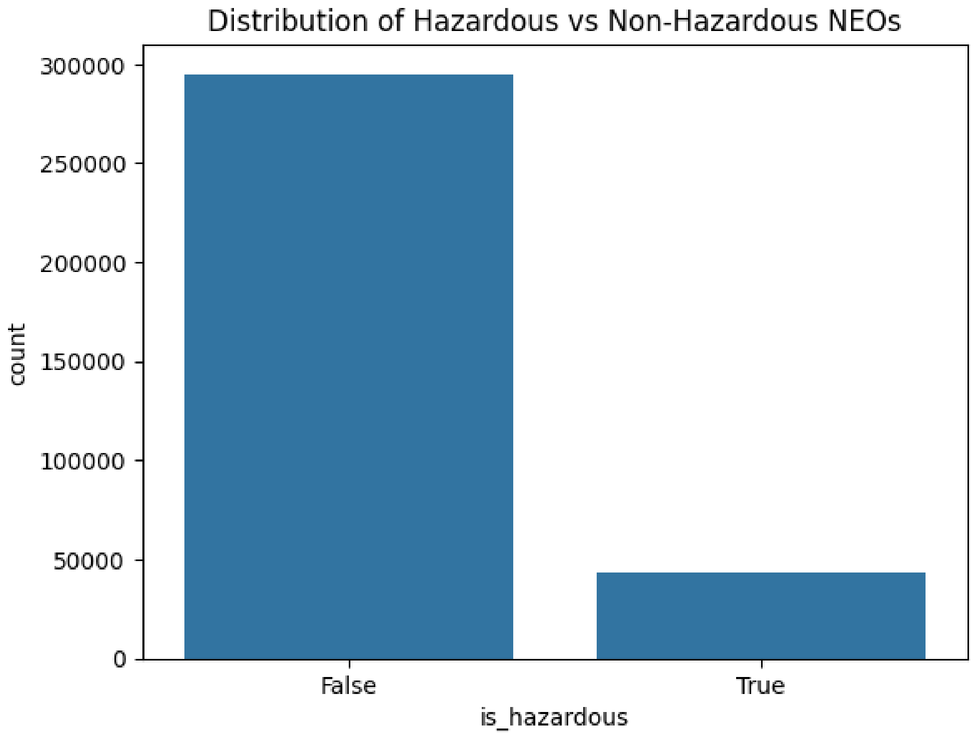
	neo_id	name	absolute_magnitude	estimated_diameter_min	estimated_diameter_max	orbiting_body	relative_velocity	miss_distance	is_hazardous
0	2162117	162117 (1998 SD15)	19.14	0.394962	0.883161	Earth	71745.401048	5.814362e+07	False
1	2349507	349507 (2008 QY)	18.50	0.530341	1.185878	Earth	109949.757148	5.580105e+07	True
2	2455415	455415 (2003 GA)	21.45	0.136319	0.304818	Earth	24865.506798	6.720689e+07	False
3	3132126	(2002 PB)	20.63	0.198863	0.444672	Earth	78890.076805	3.039644e+07	False
4	3557844	(2011 DW)	22.70	0.076658	0.171412	Earth	56036.519484	6.311863e+07	False

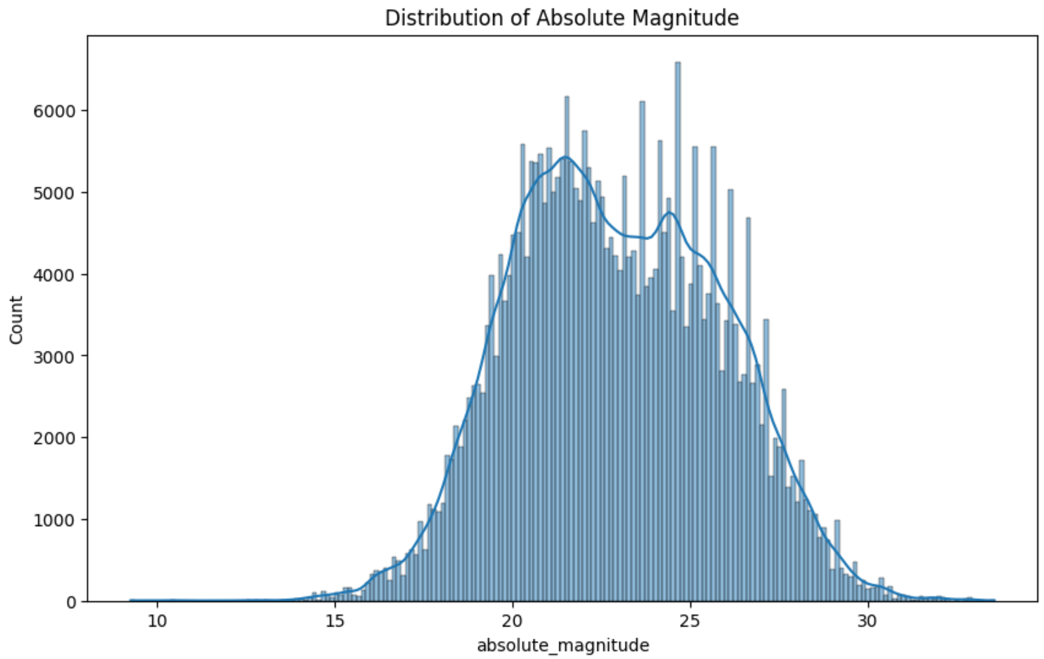
Data Cleaning and Exploratory Data Analysis (EDA)

```python
# Check the data types and missing values
neo_df.info()

# Summary statistics of the dataset
neo_df.describe()

# Check for missing values
missing_values = neo_df.isnull().sum()
print("Missing values in each column:\n", missing_values)

# Visualize the distribution of the 'is_hazardous' feature
sns.countplot(x='is_hazardous', data=neo_df)
plt.title('Distribution of Hazardous vs Non-Hazardous NEOs')
plt.show()

# Visualize the distribution of absolute magnitude
plt.figure(figsize=(10, 6))
sns.histplot(neo_df['absolute_magnitude'], kde=True)
plt.title('Distribution of Absolute Magnitude')
plt.show()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 338199 entries, 0 to 338198
Data columns (total 9 columns):
 #   Column                  Non-Null Count   Dtype
---  ------                  --------------   -----
 0   neo_id                  338199 non-null  int64
 1   name                    338199 non-null  object
 2   absolute_magnitude      338171 non-null  float64
 3   estimated_diameter_min  338171 non-null  float64
 4   estimated_diameter_max  338171 non-null  float64
 5   orbiting_body           338199 non-null  object
 6   relative_velocity       338199 non-null  float64
 7   miss_distance           338199 non-null  float64
 8   is_hazardous            338199 non-null  bool
dtypes: bool(1), float64(5), int64(1), object(2)
memory usage: 21.0+ MB
Missing values in each column:
 neo_id                     0
name                      0
absolute_magnitude       28
estimated_diameter_min   28
estimated_diameter_max   28
orbiting_body             0
relative_velocity         0
miss_distance             0
is_hazardous              0
dtype: int64
```

Data Cleaning and Exploratory Data Analysis (EDA)



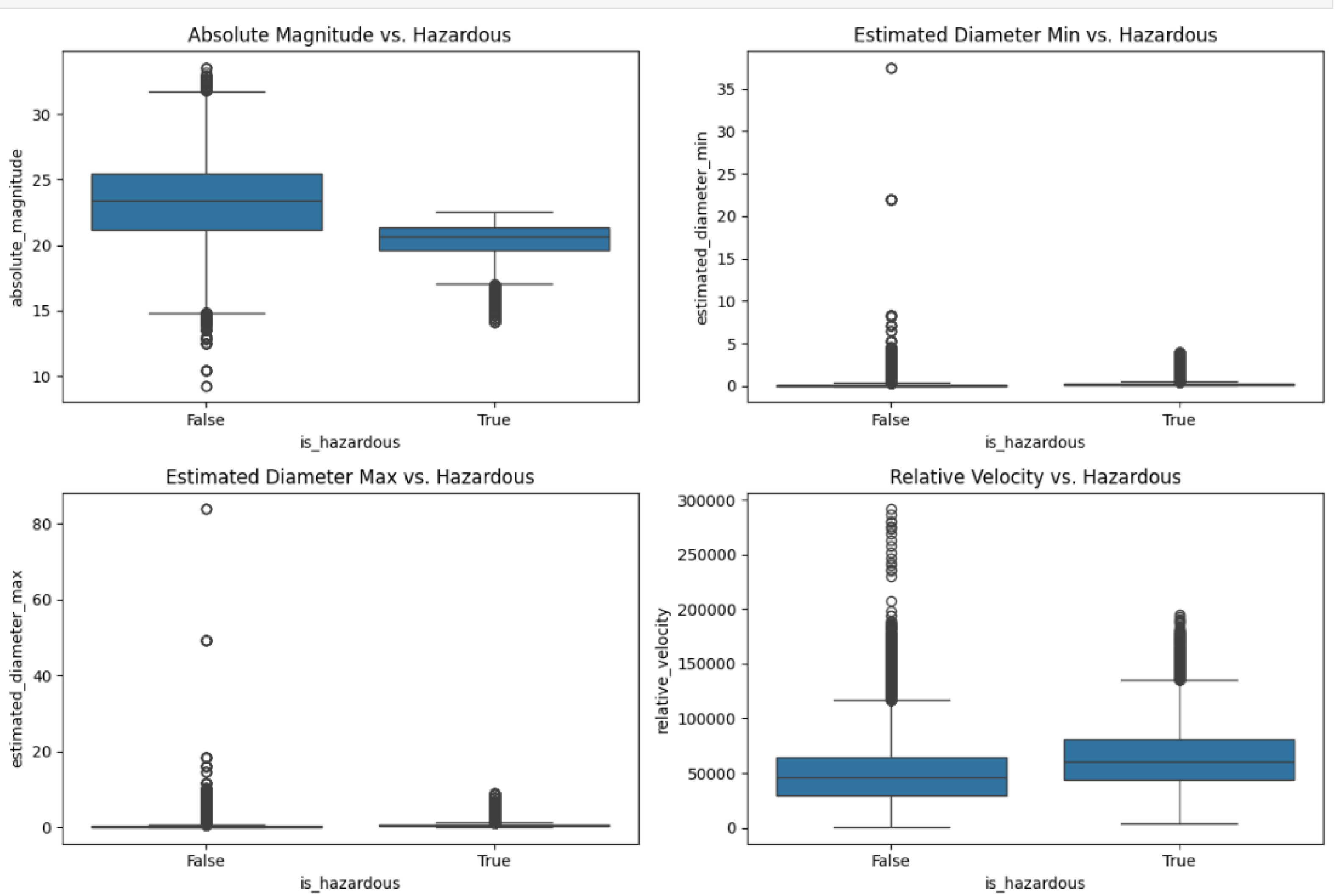


Data Cleaning

```
[3]: # Remove rows with missing values
neo_df_cleaned = neo_df.dropna()

# Verify that missing values have been removed
print("Missing values after cleaning:\n", neo_df_cleaned.isnull().sum())

# Confirm the shape of the cleaned dataset
print("Shape of the cleaned dataset:", neo_df_cleaned.shape)
```

```
Missing values after cleaning:
 neo_id                    0
name                      0
absolute_magnitude        0
estimated_diameter_min    0
estimated_diameter_max    0
orbiting_body             0
relative_velocity         0
miss_distance             0
is_hazardous              0
dtype: int64
Shape of the cleaned dataset: (338171, 9)
```

Feature Engineering & Selection



Feature Summary

```
Summary of Absolute Magnitude by Hazardous Status:
                  count       mean       std    min    25%    50%    75%    max
is_hazardous
False          295009.0  23.315579  2.881367   9.25  21.20  23.39  25.43  33.58
True            43162.0  20.314378  1.341423  14.10  19.62  20.62  21.36  22.54
Summary of Estimated Diameter Min by Hazardous Status:
                  count      mean       std       min       25%       50%  \
is_hazardous
False          295009.0  0.138021  0.311454  0.000511  0.021805  0.055790
True            43162.0  0.293083  0.296549  0.082519  0.142087  0.199781

                    75%        max
is_hazardous
False          0.152952  37.545248
True           0.316632   4.023046
Summary of Estimated Diameter Max by Hazardous Status:
                  count      mean       std       min       25%       50%  \
is_hazardous
False          295009.0  0.308624  0.696433  0.001143  0.048757  0.124750
True            43162.0  0.655353  0.663103  0.184519  0.317717  0.446725

                    75%        max
is_hazardous
False          0.342011  83.953727
True           0.708011   8.995804
Summary of Relative Velocity by Hazardous Status:
                  count          mean           std          min           25%  \
is_hazardous
False          295009.0  49171.347009  25657.848391   203.346433  29336.460874
True            43162.0  63968.941094  27748.694685  3888.602813  43602.125119

                        50%           75%            max
is_hazardous
False          45692.856505  64533.249206  291781.106613
True           59967.023231  80436.509676  194676.462159
```

Data Preprocessing

```python
# Drop unnecessary columns
neo_df_model = neo_df_cleaned.drop(columns=['neo_id', 'name', 'orbiting_body'])

# Split the dataset into features and target variable
X = neo_df_model.drop(columns=['is_hazardous'])
y = neo_df_model['is_hazardous']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Print shapes of the datasets
print(f"Training data shape: X_train: {X_train.shape}, y_train: {y_train.shape}")
print(f"Testing data shape: X_test: {X_test.shape}, y_test: {y_test.shape}")
```

```
Training data shape: X_train: (236719, 5), y_train: (236719,)
Testing data shape: X_test: (101452, 5), y_test: (101452,)
```

Training Set: 236,719 samples

Testing Set: 101,452 samples

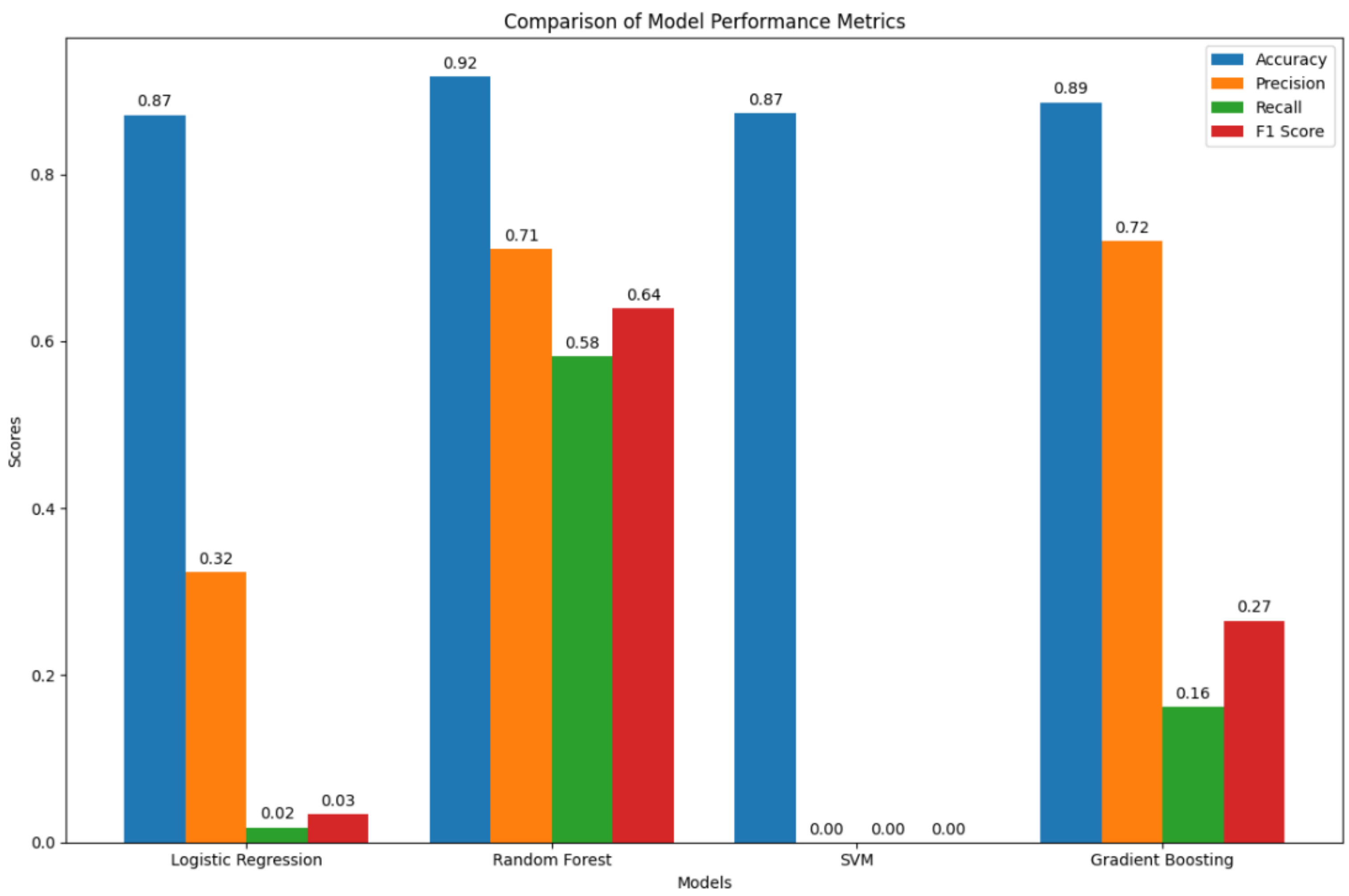
Features: 5 features (absolute_magnitude, estimated_diameter_min, estimated_diameter_max, relative_velocity, miss_distance)

Model Building and Evaluation

```
# Initialize models
log_reg = LogisticRegression(max_iter=1000)
random_forest = RandomForestClassifier()
svm = SVC()
gradient_boosting = GradientBoostingClassifier()

# Train models
log_reg.fit(X_train, y_train)
random_forest.fit(X_train, y_train)
svm.fit(X_train, y_train)
gradient_boosting.fit(X_train, y_train)

# Predict on test set
y_pred_log_reg = log_reg.predict(X_test)
y_pred_rf = random_forest.predict(X_test)
y_pred_svm = svm.predict(X_test)
y_pred_gb = gradient_boosting.predict(X_test)

# Evaluate models
def evaluate_model(y_true, y_pred, model_name):
    accuracy = accuracy_score(y_true, y_pred)
    precision = precision_score(y_true, y_pred)
    recall = recall_score(y_true, y_pred)
    f1 = f1_score(y_true, y_pred)
    print(f"{model_name} Performance:")
    print(f"Accuracy: {accuracy:.4f}")
    print(f"Precision: {precision:.4f}")
    print(f"Recall: {recall:.4f}")
    print(f"F1 Score: {f1:.4f}")
    print()

# Print evaluation results
evaluate_model(y_test, y_pred_log_reg, "Logistic Regression")
evaluate_model(y_test, y_pred_rf, "Random Forest")
evaluate_model(y_test, y_pred_svm, "SVM")
evaluate_model(y_test, y_pred_gb, "Gradient Boosting")
```

```
Logistic Regression Performance:
Accuracy: 0.8710
Precision: 0.3230
Recall: 0.0179
F1 Score: 0.0339

Random Forest Performance:
Accuracy: 0.9168
Precision: 0.7094
Recall: 0.5806
F1 Score: 0.6386

SVM Performance:
Accuracy: 0.8734
Precision: 0.0000
Recall: 0.0000
F1 Score: 0.0000

Gradient Boosting Performance:
Accuracy: 0.8860
Precision: 0.7194
Recall: 0.1627
F1 Score: 0.2654
```

Model Performance Comparison



Model Improvement Strategies

- Hyperparameter Tuning
- Training the Final Model with Best Parameters
- Comprehensive Model Evaluation

Hyperparameter Tuning

```python
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV

# Initialize the Random Forest model
rf = RandomForestClassifier(random_state=42)

# Initialize GridSearchCV with the model, parameter grid, and other settings
grid_search = GridSearchCV(estimator=rf,
                           param_grid=param_grid,
                           cv=3,                    # Number of cross-validation folds
                           scoring='accuracy', # Evaluation metric
                           n_jobs=-1,               # Use all available cores
                           verbose=2)               # Verbosity level

# Fit GridSearchCV to the training data
grid_search.fit(X_train, y_train)

# Get the best parameters and the best score
best_params = grid_search.best_params_
best_score = grid_search.best_score_

print("Best Parameters:\n", best_params)
print("Best Score:\n", best_score)
```

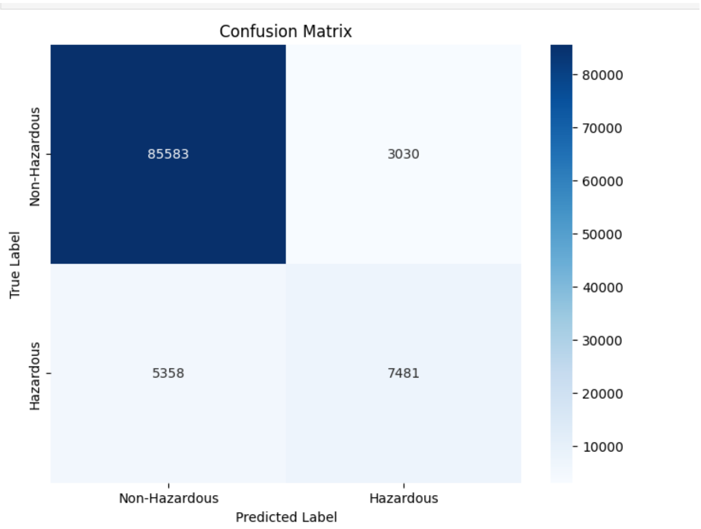
```
Fitting 3 folds for each of 1080 candidates, totalling 3240 fits
Best Parameters:
 {'bootstrap': True, 'max_depth': 40, 'max_features': 'sqrt', 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 200}
Best Score:
 0.9098931632882374
```

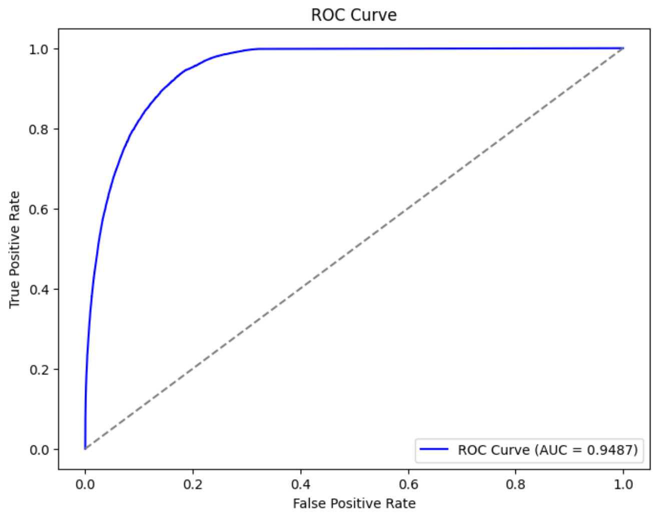
Final Random Forest Model Training and Evaluation with Best Parameters

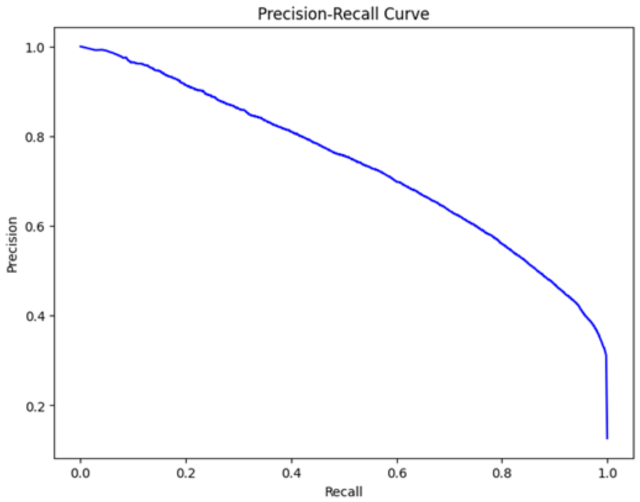
```python
# Define the best parameters
best_params = {
    'bootstrap': True,
    'max_depth': 40,
    'max_features': 'sqrt',
    'min_samples_leaf': 1,
    'min_samples_split': 2,
    'n_estimators': 200
}

# Train the Random Forest model with the best parameters
final_model = RandomForestClassifier(**best_params, random_state=42)
final_model.fit(X_train, y_train)

# Predict on the test set
y_pred = final_model.predict(X_test)
y_pred_proba = final_model.predict_proba(X_test)[:, 1]

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
roc_auc = roc_auc_score(y_test, y_pred_proba)

# Print the evaluation metrics
print("Final Model Performance:")
print(f"Accuracy: {accuracy:.4f}")
print(f"Precision: {precision:.4f}")
print(f"Recall: {recall:.4f}")
print(f"F1 Score: {f1:.4f}")
print(f"ROC AUC: {roc_auc:.4f}")

# Confusion Matrix
conf_matrix = confusion_matrix(y_test, y_pred)
print("\nConfusion Matrix:")
print(conf_matrix)
```

```
Final Model Performance:
Accuracy: 0.9173
Precision: 0.7117
Recall: 0.5827
F1 Score: 0.6408
ROC AUC: 0.9487

Confusion Matrix:
[[85583  3030]
 [ 5358  7481]]
```

Final Model Evaluation Visualizations







Conclusion

Our improved Random Forest model shows strong performance in predicting hazardous NEOs. The high accuracy and AUC indicate that the model is reliable overall. However, there's still room for improvement, particularly in reducing false negatives.