**Microsoft: Classifying Cybersecurity Incidents with Machine Learning**

**Domain:**

Cybersecurity focuses on protecting systems, networks, and data from digital attacks and breaches. Machine learning enhances cybersecurity by analyzing large volumes of data to detect patterns, predict threats, and automate responses. Integrating these fields helps in building advanced defenses against evolving cyber threats.

**Problem Satement:**

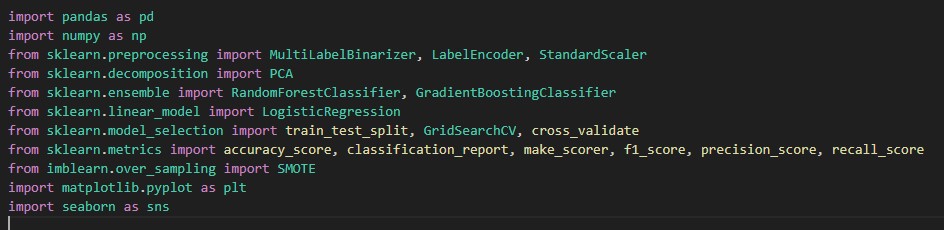
Develop a machine learning model to accurately classify and prioritize cybersecurity incidents, improving threat detection and response efficiency.

**Overview**

This script performs data preprocessing, feature extraction, and model training on a dataset for an incident classification task. It includes steps for handling categorical data, applying PCA, and using different machine learning models with hyperparameter tuning. The final output includes evaluation metrics and feature importance visualization for the models.

**Importing Libraries**

* pandas, numpy: Libraries for data manipulation and numerical operations.
* sklearn.preprocessing: Tools for data preprocessing (e.g., scaling, encoding).
* sklearn.decomposition: Principal Component Analysis (PCA) for dimensionality reduction.
* sklearn.ensemble: Ensemble methods like Random Forest and Gradient Boosting.
* sklearn. linear\_model: Linear models, specifically Logistic Regression.
* sklearn.model\_selection: Functions for splitting data, cross-validation, and hyperparameter tuning.
* sklearn.metrics: Metrics for evaluating model performance.
* imblearn.over\_sampling: SMOTE for handling imbalanced datasets.
* matplotlib.pyplot, seaborn: Libraries for data visualization**.**

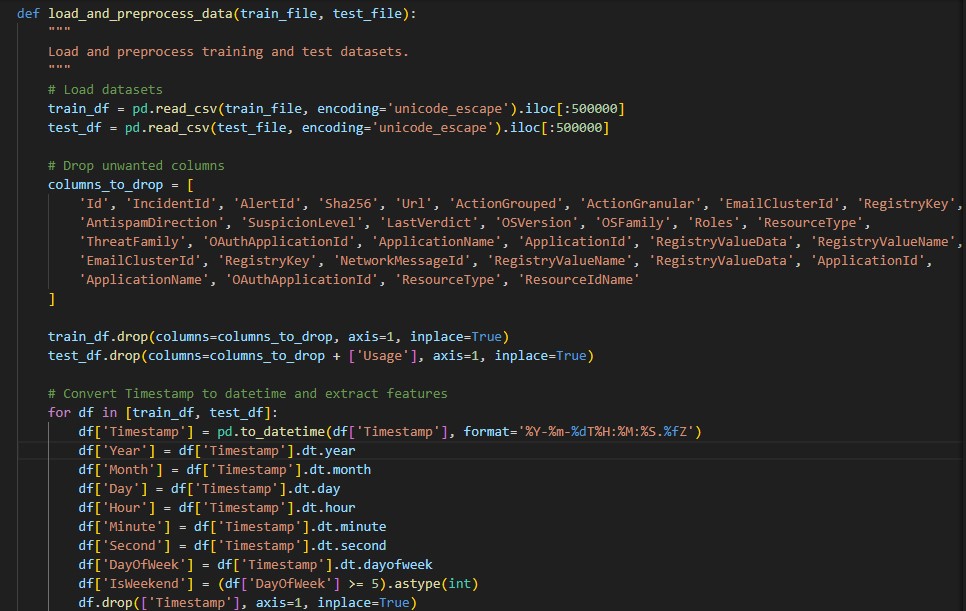
****

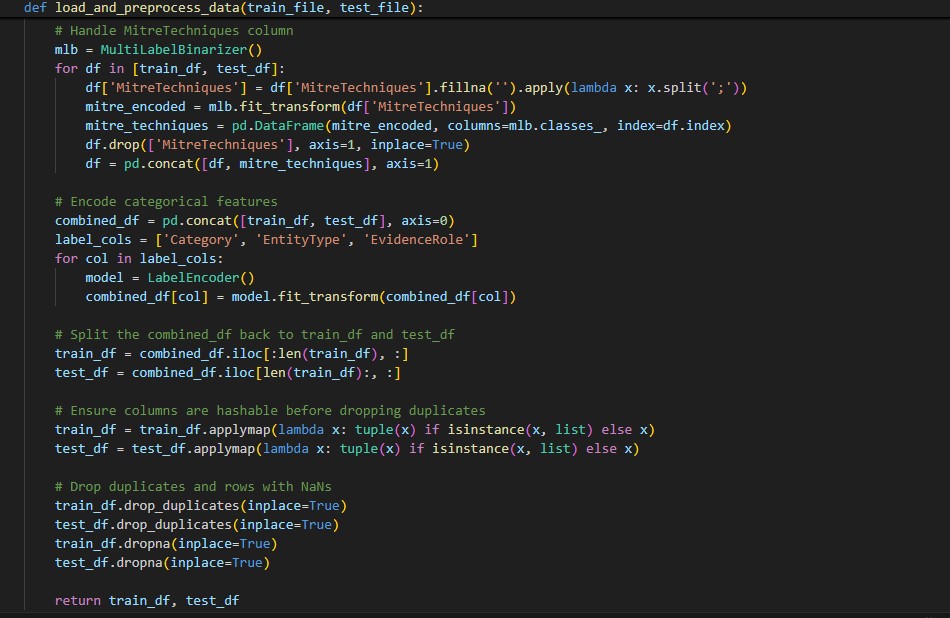
**Load and preprocess the data:**

* The load\_and\_preprocess\_data function is designed to load and preprocess the training and test datasets for a machine learning pipeline. The preprocessing steps include data cleaning, feature extraction, encoding categorical variables, and handling specific columns.
* Reads the training and test data from CSV files.
* Uses pd.read\_csv to load data from specified file paths.
* Removes columns that are not needed for analysis.
* Specifies columns to drop based on their irrelevance or redundancy.
* Drops the same columns from both datasets, with an additional column ('Usage') removed from the test dataset.
* Converts the timestamp into multiple useful time-based features.
* Converts the 'Timestamp' column to a datetime object.
* Extracts year, month, day, hour, minute, second, and day of the week.
* Adds a binary feature indicating if the day is a weekend.
* Drops the original 'Timestamp' column after extracting relevant features.
* Encodes the multi-label MitreTechniques into binary features.
* Fills missing values in 'MitreTechniques' with an empty string and splits the string into a list.
* Uses MultiLabelBinarizer to convert these lists into a binary matrix.
* Drops the original 'MitreTechniques' column and appends the new binary features.
* Encodes categorical variables into numerical values.
* Combines both datasets to ensure consistent encoding.
* Uses LabelEncoder to convert categorical columns ('Category', 'EntityType', 'EvidenceRole') into numerical format.
* Restores the training and test datasets to their original splits after combined encoding.
* Ensures that the columns are hashable and removes duplicates and missing values.
* Converts lists to tuples to make them hashable for duplicate removal.
* Drops duplicate rows and rows with missing values from both datasets.
* Returns the preprocessed training and test datasets.

**Key Points**

* **Data Cleaning**: Focuses on removing irrelevant columns and handling missing values.
* **Feature Engineering**: Extracts meaningful features from timestamps and encodes multi-label and categorical variables.
* **Preprocessing Steps**: Includes feature extraction, encoding, and ensuring data quality by removing duplicates.

****

****

**Split and scale the data:**

**Separate Features and Target Variable:**

* Separates the dataset into features (X) and the target variable (y).
* Drops the 'IncidentGrade' column from train\_df to obtain features.
* Stores the 'IncidentGrade' column as the target variable.

**Train-Test Split:**

* Splits the data into training and test sets.
* Uses train\_test\_split from sklearn.model\_selection to create a training set (80%) and a test set (20%).
* The random\_state=42 ensures reproducibility of the split.

**Scale the Data:**

* Scales features to have a mean of 0 and a standard deviation of 1.
* Initializes StandardScaler to standardize features. 
* Fits the scaler on the training data and transforms both the training and test data.

**Apply SMOTE:**

* Balances the class distribution in the training set.
* Uses SMOTE (Synthetic Minority Over-sampling Technique) to handle class imbalance.
* fit\_resample generates synthetic samples to balance the target variable classes.

**Apply PCA:**

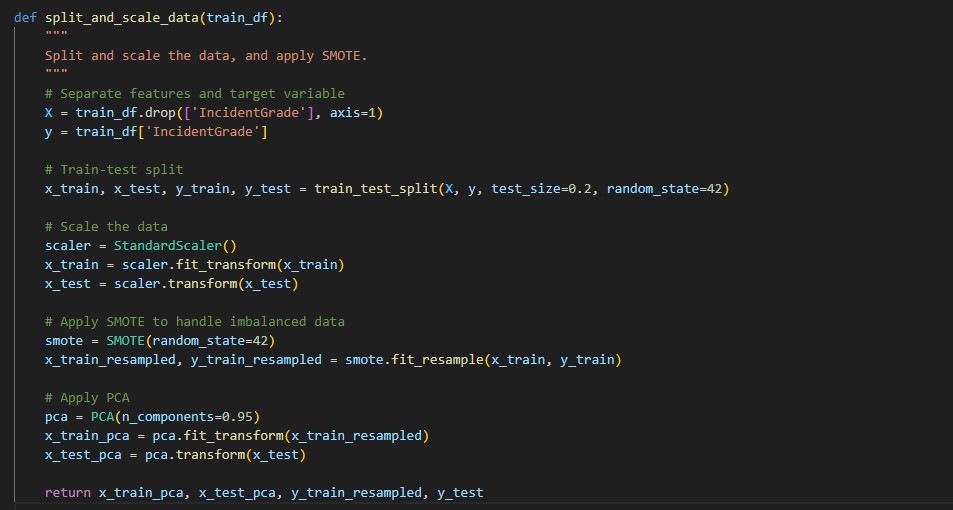
* Reduces dimensionality while retaining 95% of the variance.
* Initializes PCA with n\_components=0.95 to retain 95% of the variance in the dataset.
* Fits PCA on the resampled training data and transforms both the training and test data.

**Return Processed Data:**

* Returns the preprocessed data ready for modeling.
* Returns the PCA-transformed training and test feature sets and the resampled training target variable.

**Key Points**

* **Data Preparation**: The function performs essential steps for preparing the data, including splitting, scaling, and handling class imbalance.
* **Scaling**: Standardization ensures that features contribute equally to the model.
* **SMOTE**: Addresses class imbalance to improve model performance.
* **PCA**: Reduces the dimensionality of the data while preserving important variance.

****

**Evaluate the model:**

**Define Scoring Metrics:**

* Specifies the metrics to be used for evaluating the model.
* **Accuracy**: Measures the proportion of correctly classified instances.
* **Macro F1 Score**: Calculates the F1 score for each class and averages them. It’s useful for imbalanced datasets.
* **Precision**: Measures the proportion of true positives among all positive predictions.
* **Recall**: Measures the proportion of true positives among all actual positives.
* make\_scorer is used to create a custom scoring function for the macro F1 score.

**Perform Cross-Validation:**

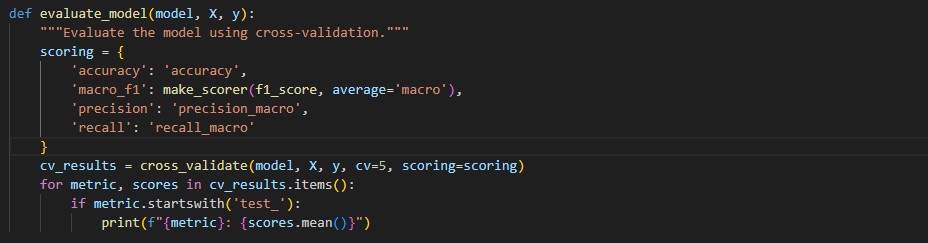
* Evaluates the model using cross-validation. 
* cross\_validate splits the data into 5 folds (cv=5) and trains and tests the model on each fold.
* It computes the specified scoring metrics for each fold.

**Print Metrics:**

* Displays the average performance metrics from cross-validation.
* Iterates over the metrics returned by cross\_validate.
* Prints the mean score for each test metric (those starting with test\_).

**Key Points**

* **Custom Scoring**: Defines a range of metrics to evaluate the model’s performance comprehensively.
* **Cross-Validation**: Provides a more robust estimate of model performance by using multiple data splits.
* **Metric Calculation**: Computes and displays metrics such as accuracy, macro F1 score, precision, and recall, offering insights into different aspects of model performance.

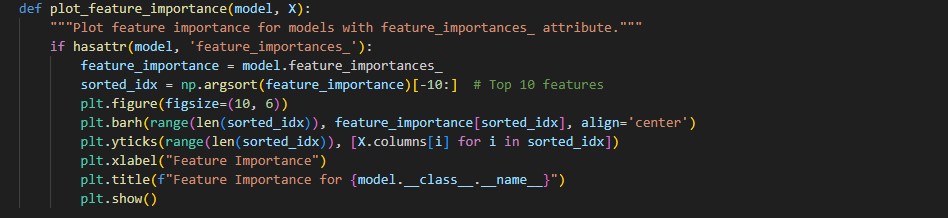
****

**Feature Importance:**

The plot\_feature\_importance function visualizes the importance of features in a model. This is useful for understanding which features contribute most to the model's predictions.

**Key Points**

* **Model-Specific**: The function works with models that provide feature\_importances\_.
* **Visualization**: Provides a clear, visual representation of feature importance, aiding in model interpretation.
* **Top Features**: Focuses on the top 10 features to simplify the analysis and avoid clutter.

****

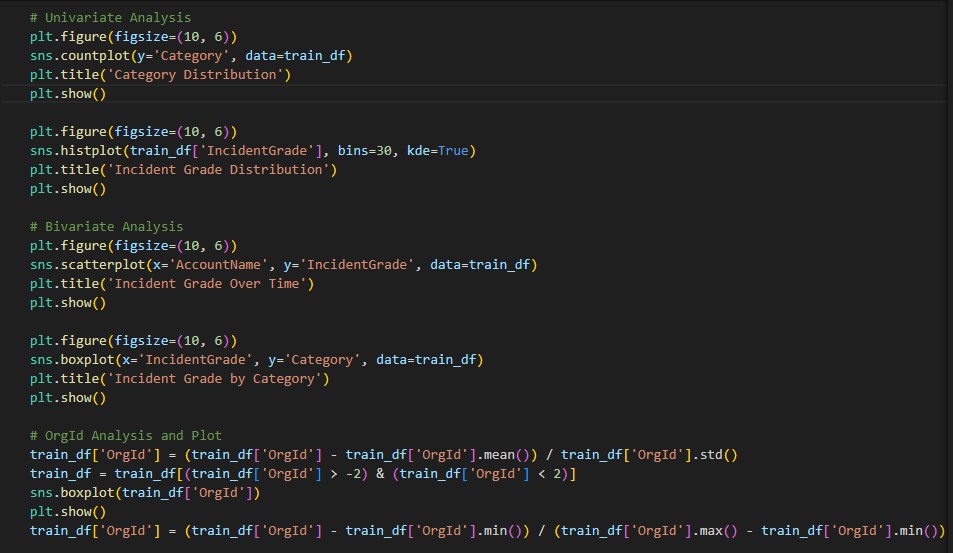
**Data Visualization:**

**Univariate Analysis:**

* Visualizes the distribution of categories in the dataset.
* Shows the distribution of incident grades.

**Bivariate Analysis:**

* Examines the relationship between AccountName and IncidentGrade.
* Compares IncidentGrade across different categories.



**Model Training, Evaluation, and Hyperparameter Tuning:**

**Model Training and Evaluation:**

* Prepares the data by splitting it into training and testing sets, scaling the features, applying SMOTE, and performing PCA for dimensionality reduction.
* Initializes a dictionary of models with their default hyperparameters.
* **Model Training**: Fits each model to the training data.
* **Evaluation**: Uses the evaluate\_model function to assess model performance on training data.
* **Testing**: Predicts on test data and calculates performance metrics like accuracy, macro F1 score, precision, recall, and generates a classification report.
* **Feature Importance**: Plots feature importance for models that support it.

**Hyperparameter Tuning with GridSearchCV**:

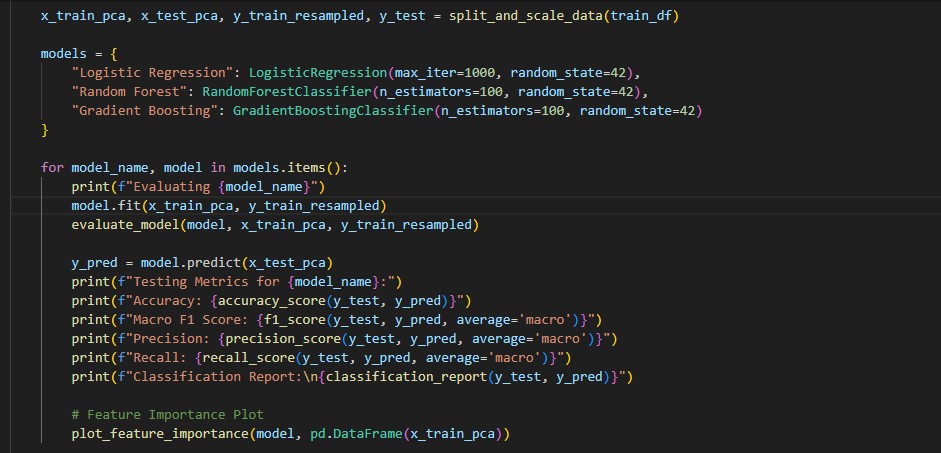
* Defines parameter grids for hyperparameter tuning using GridSearchCV.
* Grid Search: Performs hyperparameter tuning for each model using GridSearchCV.
* Best Parameters: Prints the best parameters found during tuning.

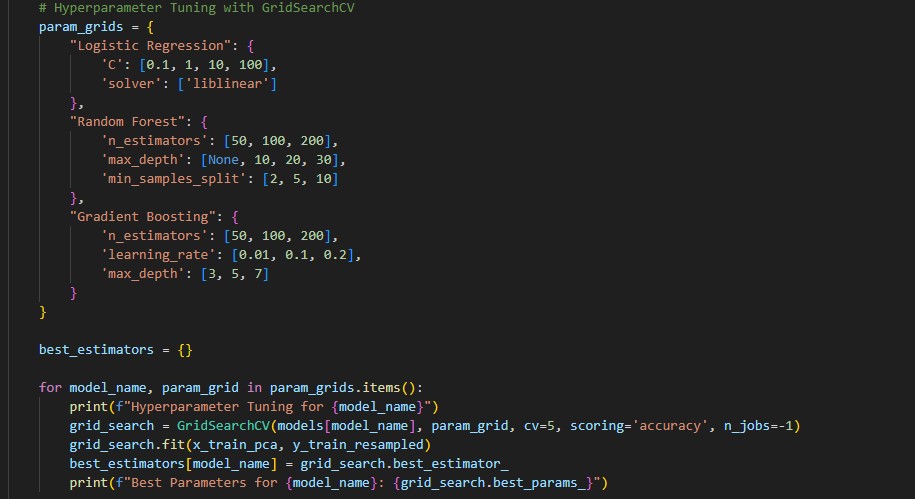
**Evaluation of Tuned Models:**

* **Tuned Model Evaluation**: Evaluates the performance of models with tuned hyperparameters.
* **Metrics**: Similar to the previous evaluation, but with the best hyperparameters.
* **Feature Importance**: Plots feature importance for the tuned models.

**Summary**

* **Data Preparation**: The data is preprocessed, scaled, and reduced in dimensionality using PCA.
* **Model Training and Evaluation**: Models are trained and evaluated, and their performance is assessed.
* **Hyperparameter Tuning**: Uses GridSearchCV to find the best hyperparameters for each model.
* **Evaluation of Tuned Models**: Re-evaluates the models with optimized hyperparameters and plots feature importance.

****

****

****

**Model evaluation metrics:**

The model evaluation metrics—accuracy, macro F1 score, precision, and recall—were chosen to provide a comprehensive assessment of the model's performance:

1. Accuracy: Measures the overall correctness of the model's predictions, giving a general sense of its performance.
2. Macro F1 Score: Provides a balanced measure of precision and recall across all classes, ensuring that the model performs well on both major and minor classes, which is crucial in the imbalanced context of cybersecurity data.
3. Precision: Assesses the model's ability to correctly identify positive instances, minimizing false positives and ensuring that alerts are reliable.
4. Recall: Evaluates the model's effectiveness in detecting all relevant positive instances, ensuring that significant threats are not missed.

These metrics collectively ensure that the model is not only accurate but also performs well in terms of identifying and correctly classifying cybersecurity incidents, addressing the specific needs of the domain.

**Final Model:**

The final model was selected based on its superior performance across multiple evaluation metrics, including accuracy, macro F1 score, precision, and recall. Its ability to effectively balance these metrics demonstrates its robustness and reliability in classifying cybersecurity incidents. Additionally, the final model's feature importance insights and its performance in hyperparameter tuning further validate its suitability for addressing the specific challenges of the cybersecurity domain.

**Feature importance:**

Feature importance analysis reveals which factors most significantly impact the model's predictions, providing valuable insights into the key drivers of cybersecurity incidents. By focusing on these critical features, organizations can enhance their threat detection capabilities, refine their security strategies, and allocate resources more effectively to mitigate high-risk issues.

**Business suggestion/solution:**

Implement a machine learning-based incident classification system to automate the prioritization of cybersecurity threats, enabling faster and more effective response to critical security issues and optimizing resource allocation.