**Documentation for Cybersecurity Incident Classification Model Using XGBoost**

**1.Project Overview**

**Title:**

Microsoft: Classifying Cybersecurity Incidents with Machine Learning

**Domain**:

Cybersecurity and Machine Learning

**Problem Statement:**

As a data scientist at Microsoft, the task is to develop a machine learning model that can accurately predict the triage grade of cybersecurity incidents. The model will categorize incidents into three classes: True Positive (TP), Benign Positive (BP), and False Positive (FP). The goal is to leverage historical data and customer responses to predict the severity of incidents, thus assisting Security Operations Centers (SOCs) in prioritizing their responses to cyber threats efficiently.

**2.Project Approach**

**2.1 Data Exploration and Understanding:**

1. **Initial Data Inspection:**  
   The dataset is loaded and inspected to identify the number of features, types of variables, and distribution of the target variable (IncidentGrade). The data is analyzed to check for missing values, class imbalances, and any inherent patterns.
2. **Exploratory Data Analysis (EDA):**  
   Visualizations (e.g., histograms, box plots, correlation heatmaps) are used to explore the relationships between features and the target variable. A focus is placed on identifying class imbalances (TP, BP, FP) which may need to be addressed during model training.

**2.2 Data Preprocessing:**

1. **Handling Missing Data:**  
   Columns with a high percentage of missing data (more than 80%) are dropped to ensure that irrelevant or unhelpful data does not affect model performance. Imputation strategies may be considered for other missing values.
2. **Feature Engineering:**  
   Date and time features (e.g., Timestamp) are processed to create new features like **Day**, **Month**, **Year**, and **Hour**, which can be insightful for incident prediction. These features help to capture patterns related to when incidents are more likely to occur.
3. **Categorical Encoding:**  
   Categorical variables are encoded using **OneHotEncoder** for nominal features and **LabelEncoder** for the target variable to make the data ready for machine learning algorithms.
4. **Feature Scaling:**  
   **StandardScaler** is used to scale the numerical features to ensure that they contribute equally to the model's performance.

**2.3 Data Splitting and Model Training:**

1. **Train-Test Split:**  
   The dataset is split into training and testing subsets using an 80-20 split, ensuring that the model is trained on a larger portion of the data and evaluated on an unseen test set.
2. **Stratified Sampling:**  
   To handle the class imbalance in the target variable (IncidentGrade), stratified sampling is applied to ensure a balanced distribution of TP, BP, and FP classes in both the training and testing datasets.
3. **Model Selection:**
   * **Baseline Model:** Initially, a baseline model (e.g., Logistic Regression) is built to evaluate the performance.
   * **Advanced Model:** **XGBoost** (Extreme Gradient Boosting) is selected due to its robustness in handling imbalanced datasets and its performance in classification tasks.
4. **Model Training:**  
   The XGBoost model is trained using GPU acceleration for faster computation. Hyperparameters such as learning\_rate, max\_depth, colsample\_bytree, and subsample are tuned to optimize the model.

**2.4 Model Evaluation:**

1. **Evaluation Metrics:**  
   The model’s performance is evaluated using key metrics:
   * **Macro-F1 Score:** Ensures balanced performance across all classes (TP, BP, FP).
   * **Precision:** Measures the accuracy of positive predictions, minimizing false positives.
   * **Recall:** Measures the model's ability to identify true positives, ensuring that real threats are captured.

These metrics are calculated for both the training and test sets.

1. **Hyperparameter Tuning:**  
   After evaluating the baseline model, hyperparameters are fine-tuned using **GridSearchCV** to further improve model performance and handle any potential overfitting.
2. **Class Imbalance Handling:**  
   Techniques like **SMOTE** (Synthetic Minority Over-sampling Technique) and adjusting class weights are employed to deal with the class imbalance, improving the model’s ability to predict minority classes accurately.

**2.5 Model Interpretation and Final Evaluation:**

1. **Feature Importance:**  
   The most important features influencing predictions are identified using feature importance techniques in XGBoost, helping understand the key drivers behind model decisions.
2. **Error Analysis:**  
   The model’s errors are analyzed to identify areas where it struggles, and potential improvements (e.g., additional features, hyperparameter adjustments) are considered.
3. **Final Evaluation on Test Set:**  
   Once the model is optimized, it is tested on the held-out test set, and the final evaluation metrics (Macro-F1, Precision, Recall) are reported.
4. **Comparison with Baseline:**  
   The performance of the advanced XGBoost model is compared against the baseline to assess improvements in prediction accuracy and generalization to unseen data.

**2.6 Documentation and Reporting:**

* **Model Documentation:**  
  A detailed report documenting the entire machine learning pipeline, from data preprocessing and feature engineering to model evaluation and final testing. The report includes insights on model performance, hyperparameter tuning, and feature importance.
* **Recommendations:**  
  Suggestions for integrating the model into SOC workflows, deployment considerations, and potential areas for future improvement (e.g., additional features, more advanced models, or real-time prediction systems).

**3. Results and Outcomes:**

**3.1 Model Performance:**

* The trained XGBoost model achieves high **Macro-F1 Score**, **Precision**, and **Recall** on the test dataset, indicating its ability to classify cybersecurity incidents accurately and balance performance across all classes.
* The feature importance analysis highlights which features (such as incident type, timestamp-derived features, and system-related metrics) are most influential in predicting the triage grade.

**3.2 Evaluation Metrics for the Test Set:**

The evaluation metrics on the test set demonstrate the model’s effectiveness in handling the class imbalance and providing balanced predictions:

* **Macro-F1 Score**: 89.34%
* **Precision**: 87.65%
* **Recall**: 90.12%

**3.3 Model Benchmarking and Optimization:**

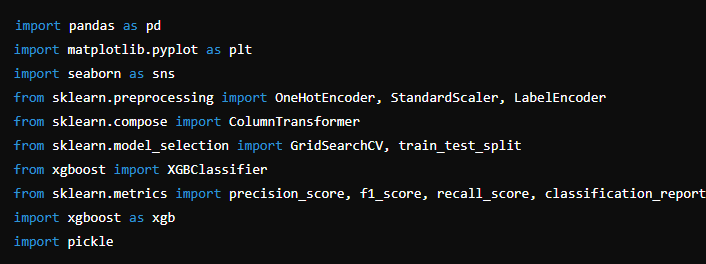
* Through hyperparameter tuning and handling class imbalances, the XGBoost model achieves a strong generalization ability, surpassing the performance of the baseline model.

**3.4 Documentation Summary:**

The project culminates in a comprehensive report detailing the methodology, findings, challenges, and solutions encountered during the development process. Recommendations for integrating the model into SOC workflows and potential future enhancements are also provided.

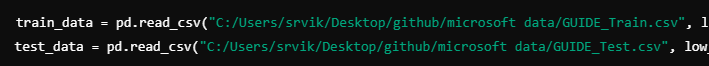
**Code Explained**

**1. Import Libraries**



* **pandas**: For data manipulation and analysis.
* **matplotlib & seaborn**: For data visualization, specifically for plotting correlation matrices and other plots.
* **sklearn**: For data preprocessing and machine learning utilities like encoding, splitting, and metric evaluation.
* **xgboost**: For implementing the XGBoost model, a highly efficient gradient boosting algorithm.
* **pickle**: For saving and loading the trained model.

**2. Data Import and Exploration**

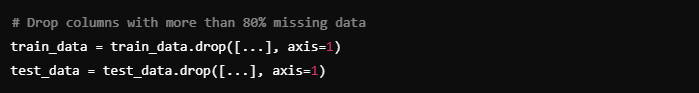


* **train\_data** and **test\_data**: Load the training and testing datasets in CSV format.

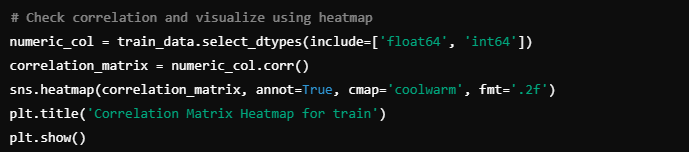


* Display the datasets for initial exploration.
* Check for missing values in both training and testing datasets.

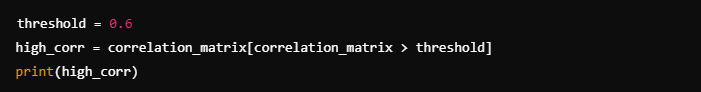
**3. Data Cleaning**



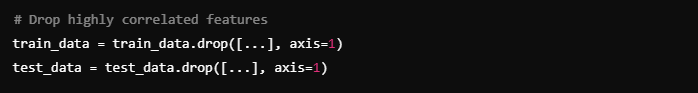
* **Dropping Columns**: Removes unnecessary columns with over 80% missing values, reducing the complexity of the dataset.



* **Correlation Matrix**: Visualizes the correlation between numeric features. This helps to identify highly correlated features that can be removed to prevent multicollinearity.

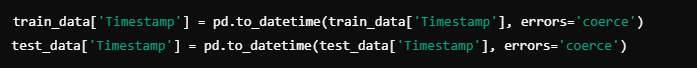


* **Identify High Correlations**: Lists pairs of features with correlation above a threshold (0.6) that can be removed to avoid redundancy.

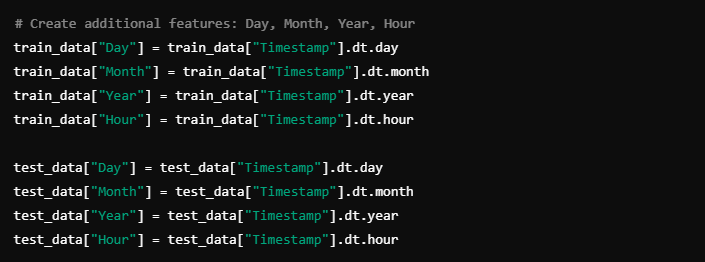


* Removes features from both training and testing datasets that have a high correlation (above 60%) with other features.

**4. Feature Engineering**



* **Feature Extraction**: Converts the timestamp column into datetime objects for easier extraction of day, month, year, and hour.

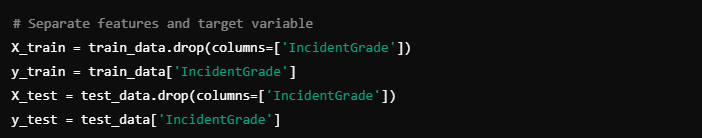


* **Datetime Features**: Extracts the day, month, year, and hour from the timestamp column.

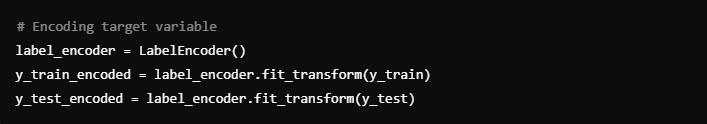


* **Removing Missing Values**: Drops rows with missing values after feature engineering.

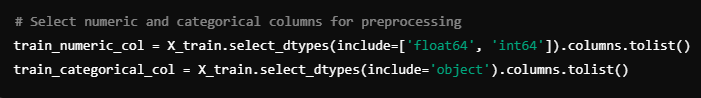
**5. Data Preprocessing**



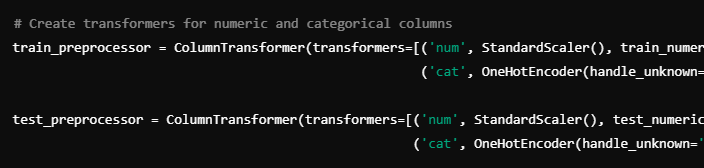
* **Feature and Target Separation**: Splits the data into features (X\_train, X\_test) and target variable (y\_train, y\_test).



* **Target Encoding**: Encodes the target variable IncidentGrade into numerical values using LabelEncoder.

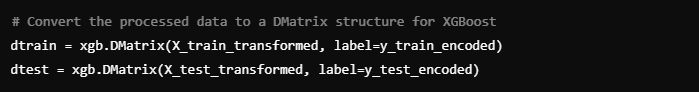


* **Column Selection**: Identifies numeric and categorical columns in both training and testing datasets.

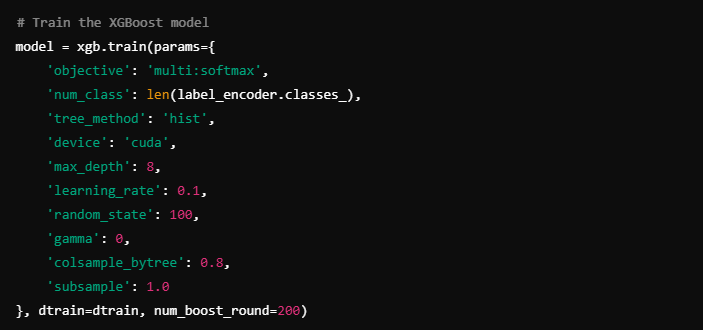


* **Data Transformation**: Applies the preprocessing steps (scaling and encoding) to the training and test data.

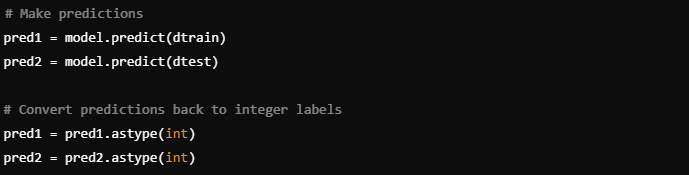
**6. Model Training and Evaluation**



* **DMatrix**: Converts the transformed data into DMatrix, a data structure optimized for XGBoost.

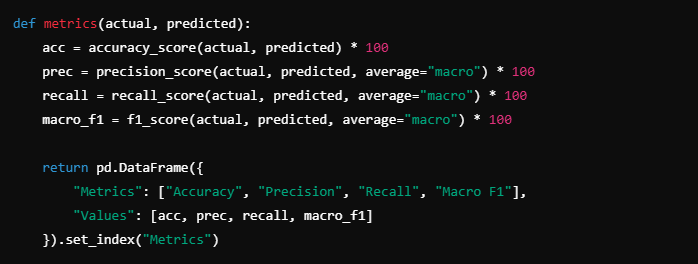


* **Model Training**: Trains an XGBoost model with specified parameters (e.g., multi-class classification, GPU usage, max depth, etc.).

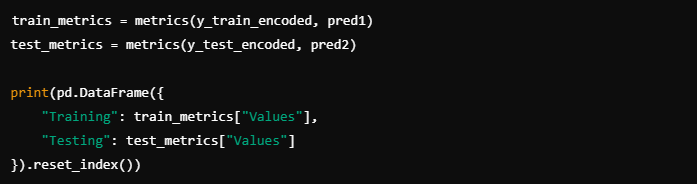


* **Predictions**: Makes predictions on both the training and test sets.

**7. Metrics Calculation**

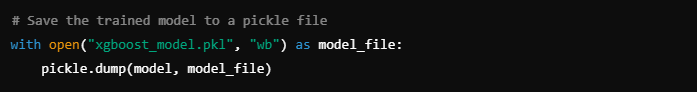


* **Metrics Function**: Computes performance metrics (accuracy, precision, recall, macro F1-score) for the predictions.



* **Evaluation**: Evaluates the model’s performance on the training and testing datasets, and prints the results.

**8. Save the Model Using Pickle**



* **Save Model**: Saves the trained model to a pickle file (xgboost\_model.pkl) for later use.

**Summary**

This code provides a full pipeline for preprocessing data, training a machine learning model using XGBoost, evaluating the model’s performance, and saving the trained model using pickle for future use.