**Beyza**

**Data Mining Midterm**

**\*The report includes all the desired needs mentioned in Part 4 of the Midterm document, including visualizations, tables, and short descriptions. The reason why sections 1, 2, and 3 are examined under separate headings is to make it easier to understand the stages within the report.\***

**Part 1: Select Data Mining Technique**

**1. Choose a data mining technique for analyzing the dataset, justifying your selection. Consider techniques such as Classification, Clustering, Anomaly Detection,** **or Association Rule Mining.**

I chose Classification among the Clustering, Anomaly Detection, and Classification or Association Rule Mining data mining options.

**2. Explain the rationale behind your choice and its relevance to cybersecurity, detailing expected outcomes.**

The method I chose is because the CICIDS2017 dataset is a dataset designed for network traffic analysis, and the classification method is used to classify samples with certain labels (for example, normal or attack) and determine which samples are normal and which are attacks when analyzing network traffic.

At the same time, this model can be used to segment security events into predefined classification algorithms (Random Forest, SVM, XGBoost, etc.). This can be useful to address important security issues such as intrusion detection or malware detection and increase the ability to identify.

Relevance to Cybersecurity:

Classification models are used to identify instances with specific labels. In this case, it aims to identify cybersecurity events such as intrusion detection or malware detection. Additionally, a trained classification model can detect abnormal activities by monitoring real-time network traffic. This enables security professionals to intervene rapidly, creating a more effective defense strategy against cyber threats.

**Part 2:** **Apply Security Threat Detection**

**3. Utilize the chosen data mining techniques to detect and classify potential security threats or anomalies within the dataset.**

I will leverage the selected classification technique on the CICIDS2017 dataset to effectively detect and classify potential security threats or anomalies. By employing machine learning algorithms such as Random Forest, SVM, XGBoost, and others, I aim to build a robust model capable of accurately differentiating between various types of network activities, including normal and malicious behavior.

The preprocessing steps, including label encoding, min-max normalization, and handling imbalanced classes using techniques like SMOTE, have been carefully applied to enhance the model's performance. These steps ensure that the model is well-equipped to handle the intricacies of the dataset, providing a foundation for effective threat detection.

**Firstly**, we start with importing libraries and Preprocessing to clean the dataset.

Importing libraries codes, imports various Python libraries commonly used in data science and machine learning applications. These libraries include tools for data manipulation, visualization, preprocessing, and machine learning model building.

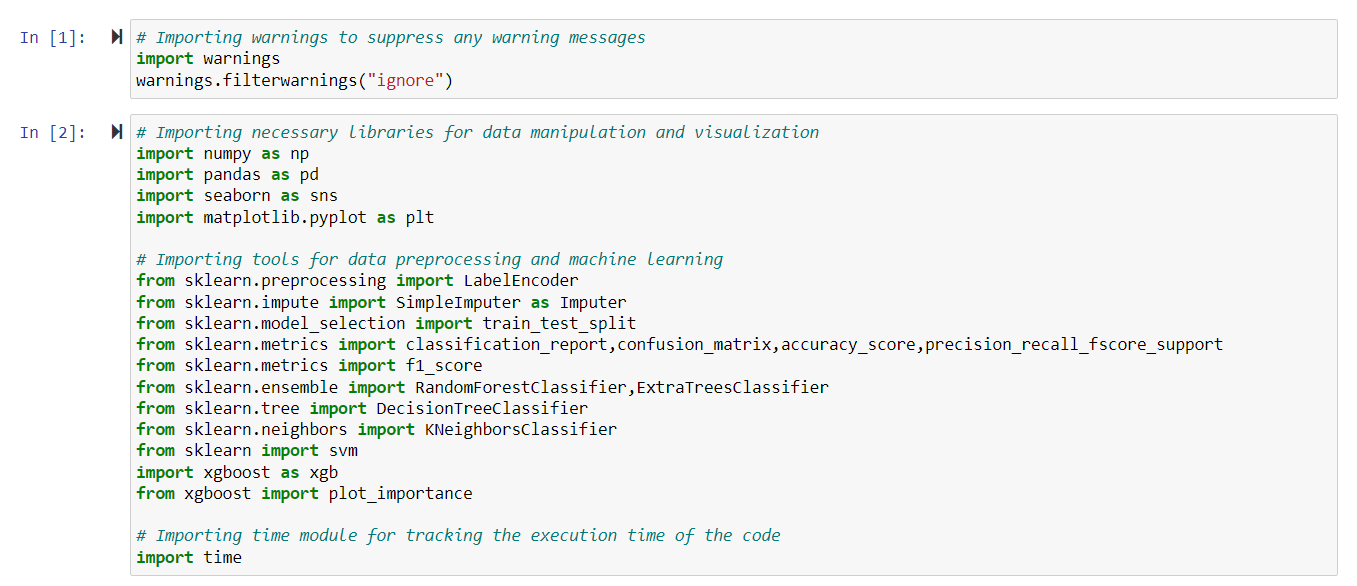
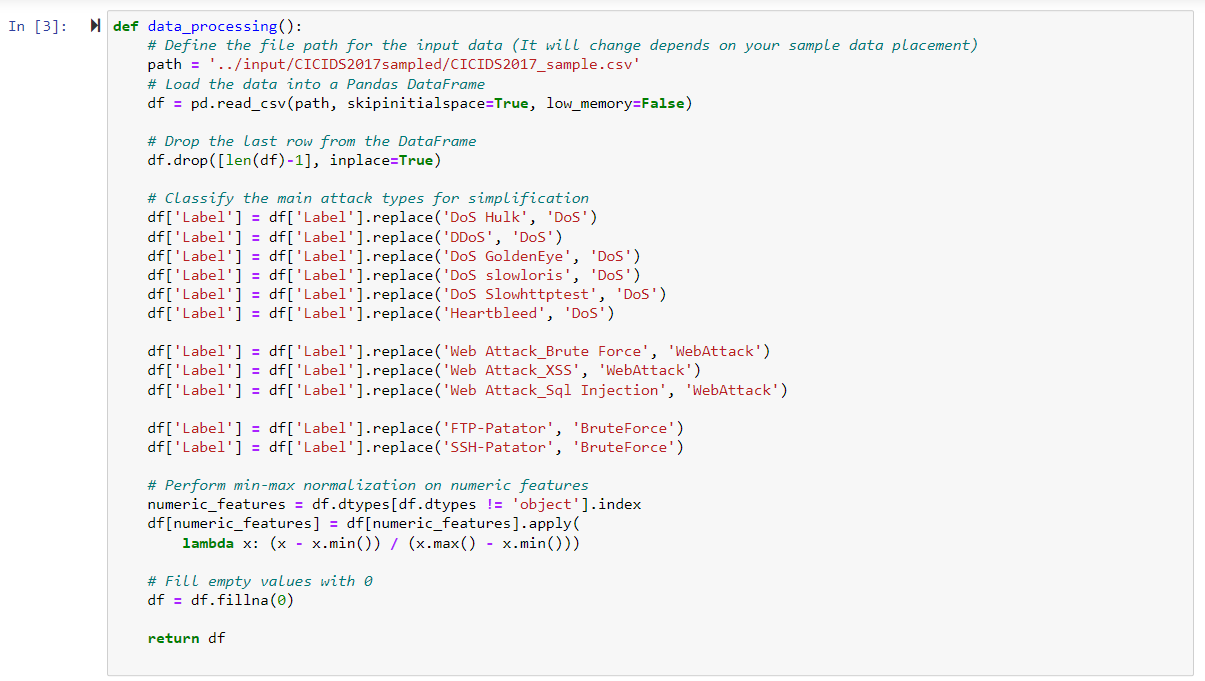
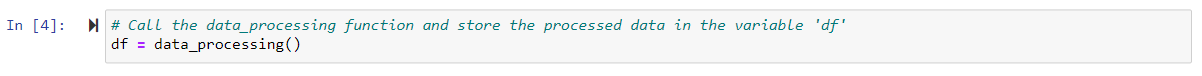


Figure 1 Importing Libraries

Preprocessing contains a function that processes a data set and codes that perform data preprocessing steps using this function. These steps include loading data, simplifying labels, normalizing numeric properties, and filling in missing values. The function returns the processed dataset, making it ready for use.





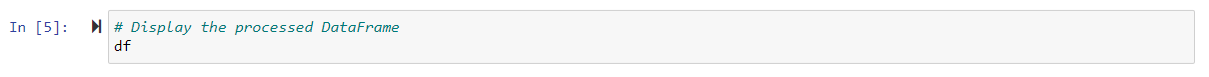


Figure 2 Preprocessing

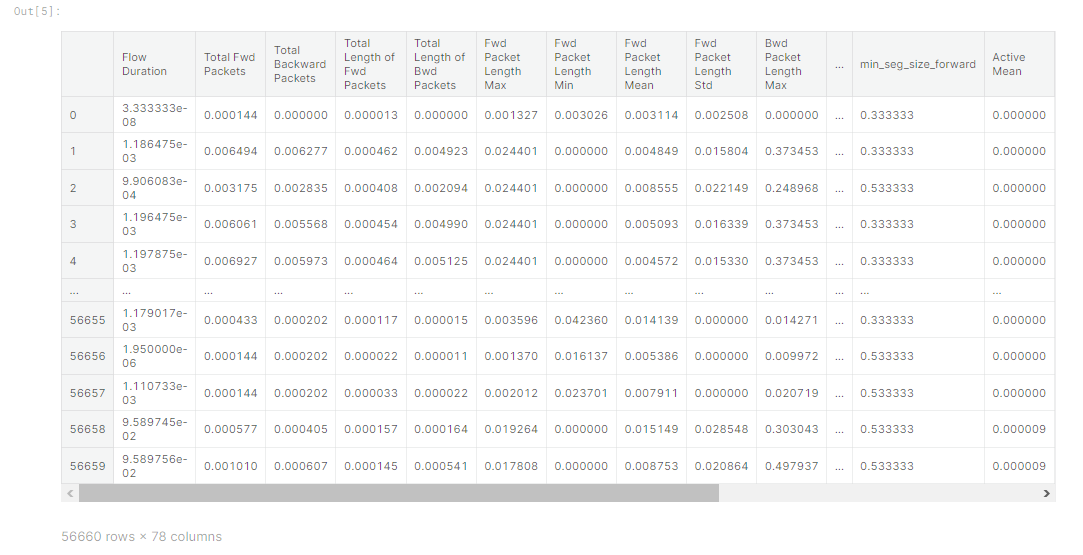


Figure 3 Output 1

After importing and preprocessing, in the "split train set and test set" step, the data set is divided into training and test sets. This distinction is made for the purpose of learning the model on training data and evaluating its performance on test data.

The dataset is split into training and testing sets, with an 80-20 ratio. Label encoding is applied to the target variable, and the split data is ready for model training.

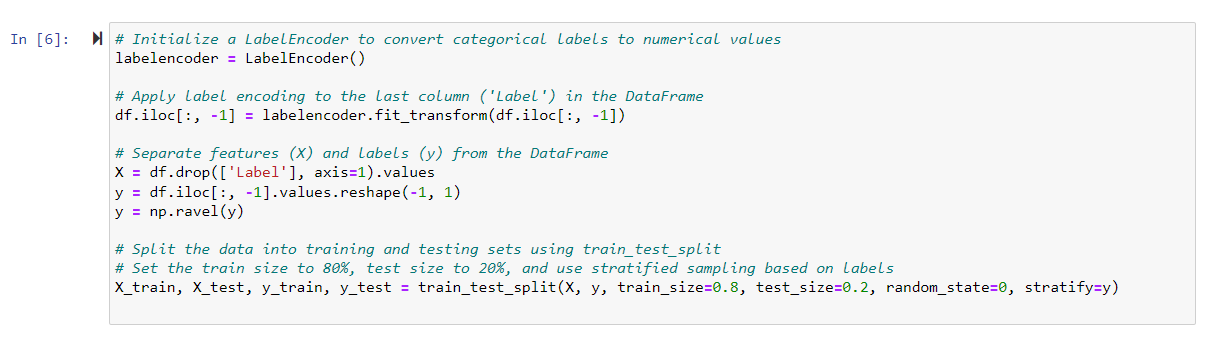
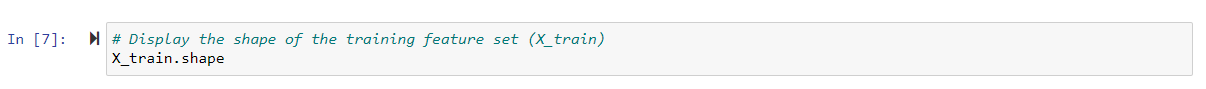
 

Figure 4 Split Training and Test Set



Figure 5 Output 2

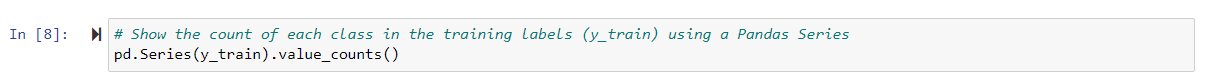
 

Figure 6 Output 3

After all, in the step "Oversampling by SMOTE" involves using the SMOTE (Synthetic Minority Over-sampling Technique) method to increase the number of samples, particularly for minority classes in cases where there is class imbalance.

In this step, the SMOTE class from the imblearn library is used to perform oversampling. SMOTE attempts to reduce class imbalance in the dataset by replicating examples of minority classes. Specifically, the sampling\_strategy parameter allows for setting specific sample numbers for certain classes.

For instance, as you can see in Figure 8, Each number represents the count of instances belonging to a specific class. In imbalanced datasets, differences in the number of instances among classes can impact model performance.

The oversampling process (using SMOTE in this case) attempts to address this imbalance by boosting the representation of minority classes, contributing to a more balanced training set for the model.

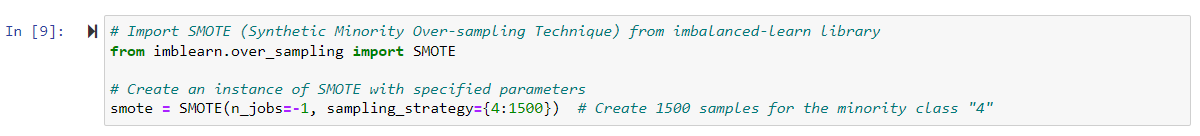
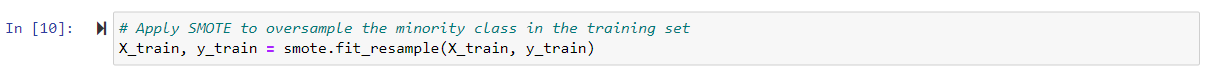
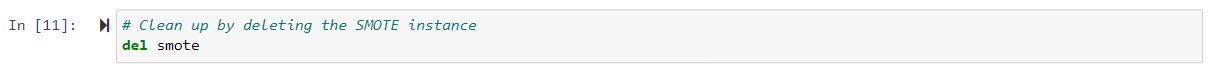
   

Figure 7 Oversampling by SMOTE



Figure 8 Output 4

**Machine learning model training**

Four base learners—Decision Tree, Random Forest, Extra Trees, and XGBoost—are trained on the preprocessed data. The models are evaluated in terms of accuracy, precision, recall, F1-score, and confusion matrices.

A Decision Tree (DT) classifier is trained on the dataset. Performance metrics such as accuracy, precision, recall, and F1-score are computed and visualized. The training time for this model is approximately 1.62 seconds.

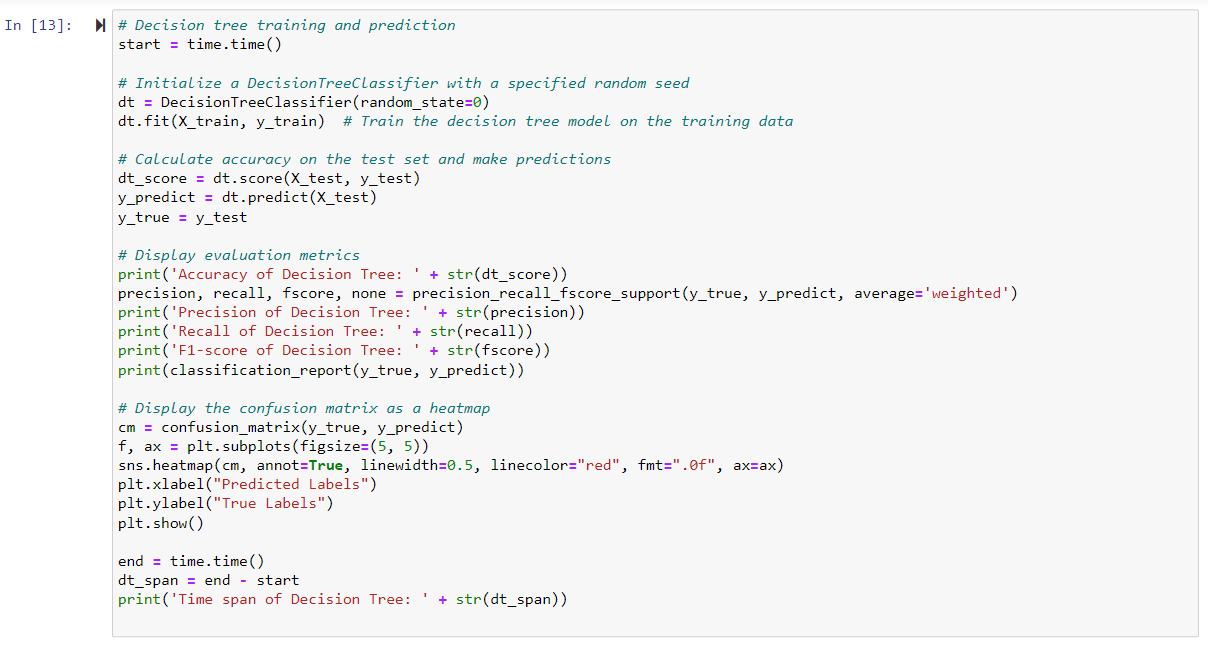


Figure 9 Decision Tree Training and Prediction

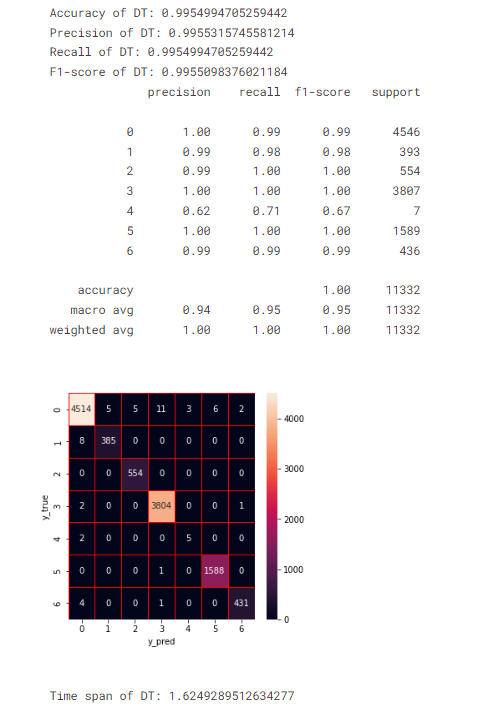


Figure 10 Output 5

A Random Forest (RF) classifier is employed for training, and its performance is evaluated using accuracy, precision, recall, and F1-score. The training time for Random Forest is longer, approximately 9.70 seconds, due to the ensemble nature of the model.

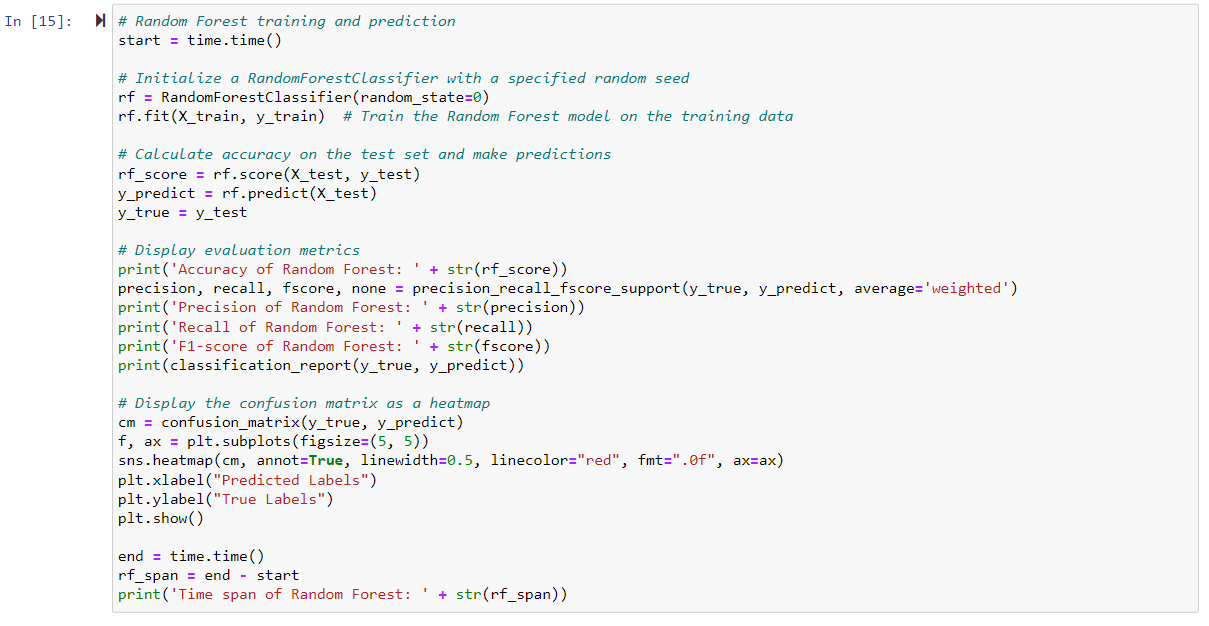
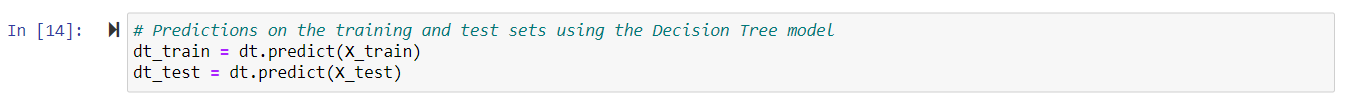


Figure 11 Random Forest training and prediction

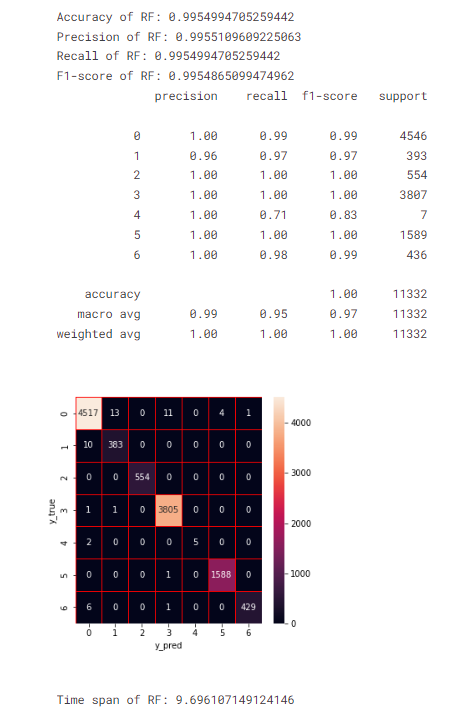


Figure 12 Output 6

Extra Trees (ET), another ensemble classifier, is trained on the data. Similar performance metrics are calculated, and the results are visualized. The training time for Extra Trees is around 7.08 seconds.



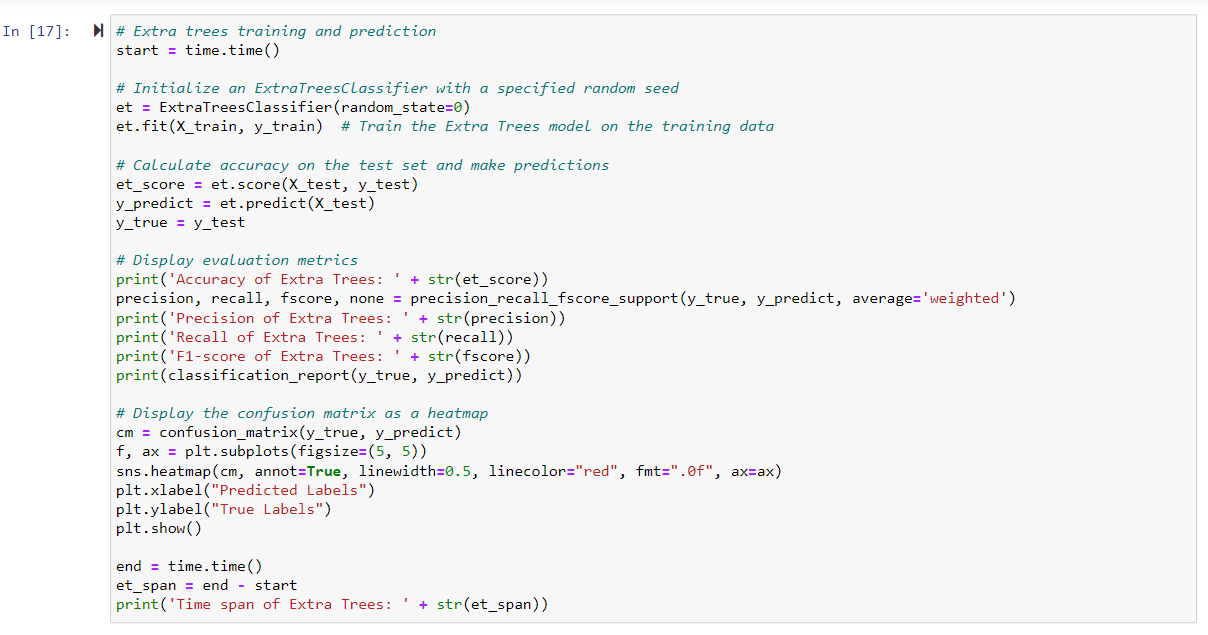


Figure 13 Extra Trees Training and Prediction

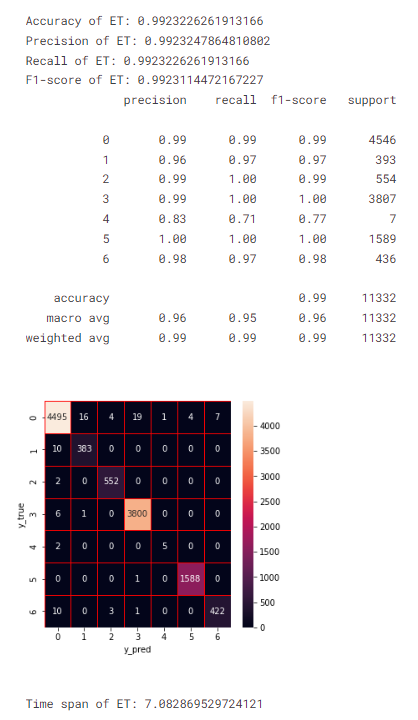
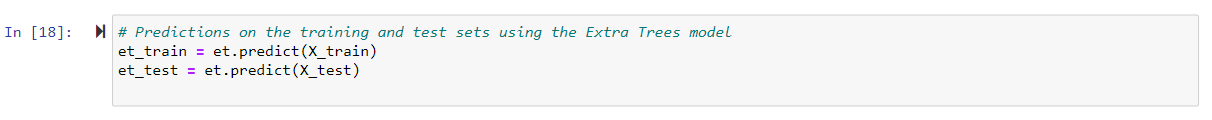


Figure 14 Output 7

XGBoost, an optimized gradient boosting algorithm, is utilized as a base learner. Like the previous models, accuracy, precision, recall, and F1-score are determined. XGBoost demonstrates competitive performance with a training time of about 7.95 seconds.



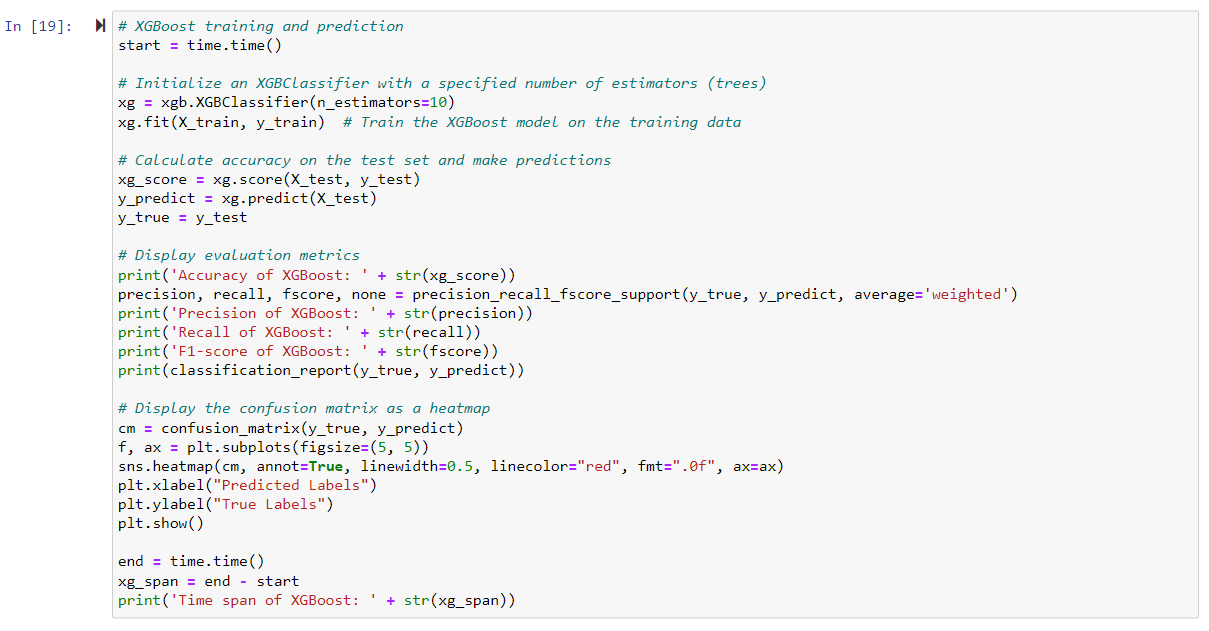


Figure 15 XGboost Training and Prediction

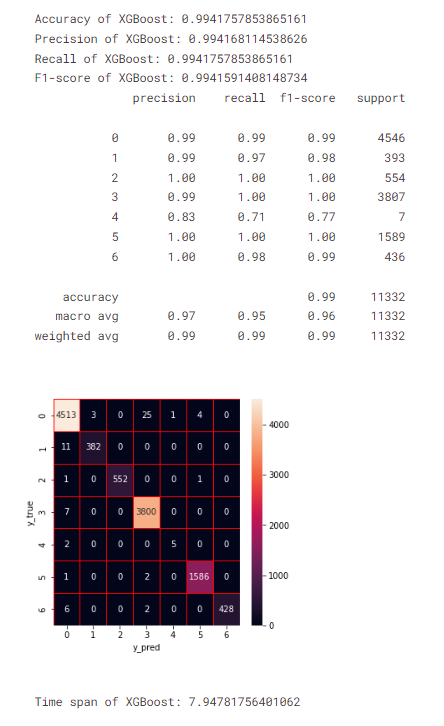


Figure 16 Output 8

A stacking ensemble model is constructed using the outputs of four base models: Decision Tree, Random Forest, Extra Trees, and XGBoost. The predictions from each base model are combined into a new DataFrame called base\_predictions\_train. This DataFrame contains columns for each base model's predictions (DecisionTree, RandomForest, ExtraTrees, XgBoost), and the initial rows are displayed.

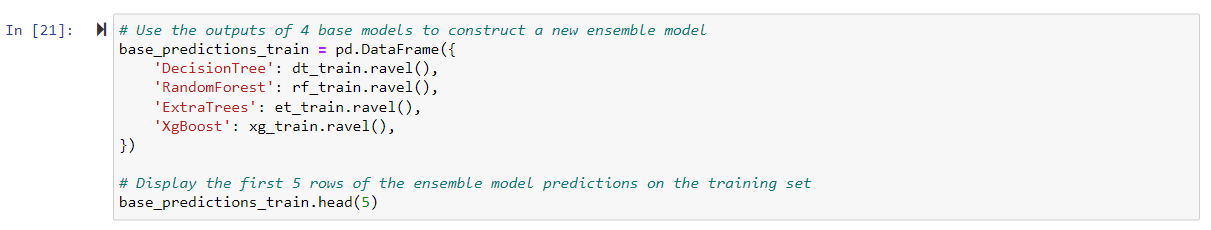
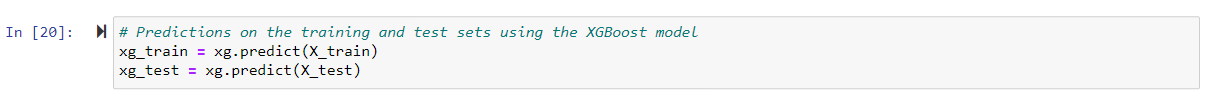


Figure 17 Stacking Model Construction (ensemble for 4 base learners)

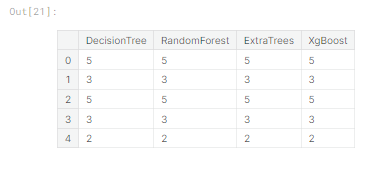


Figure 18 Output 9

The predictions from each base learner (Decision Tree, Extra Trees, Random Forest, XGBoost) for both the training and testing sets are reshaped to ensure compatibility for stacking.

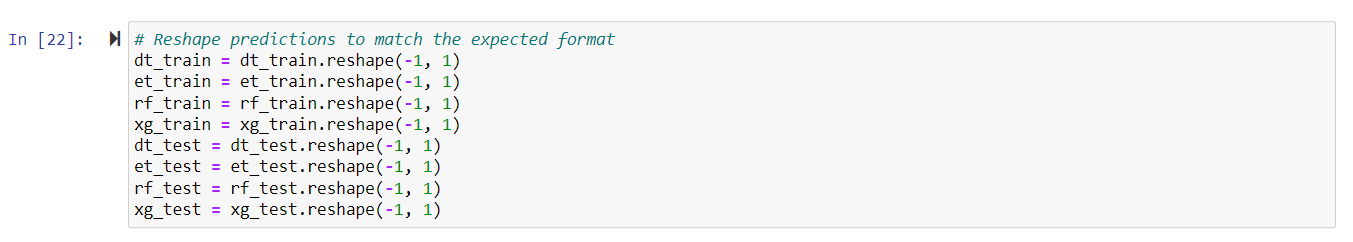


Figure 19 Reshape Predictions

The reshaped predictions from the base learners are concatenated horizontally to create the input features for the stacking model. This results in **x\_train** for the training set and **x\_test** for the testing set.

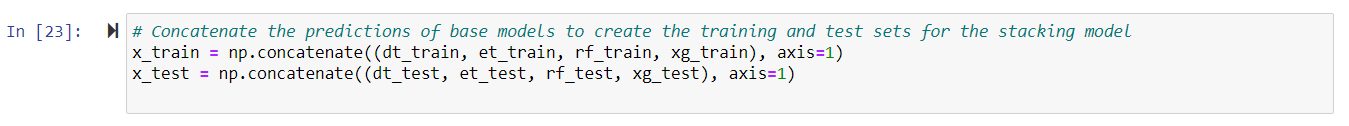


Figure 20 Concatenate the Predictions

A new XGBoost classifier (**stk**) is trained on the stacked input features (**x\_train**) and the original labels for the training set (**y\_train**). The trained model is then used to predict the labels for the testing set (**y\_predict**). Various performance metrics, including accuracy, precision, recall, F1-score, and a classification report, are computed and displayed. Additionally, a confusion matrix is visualized using a heatmap.

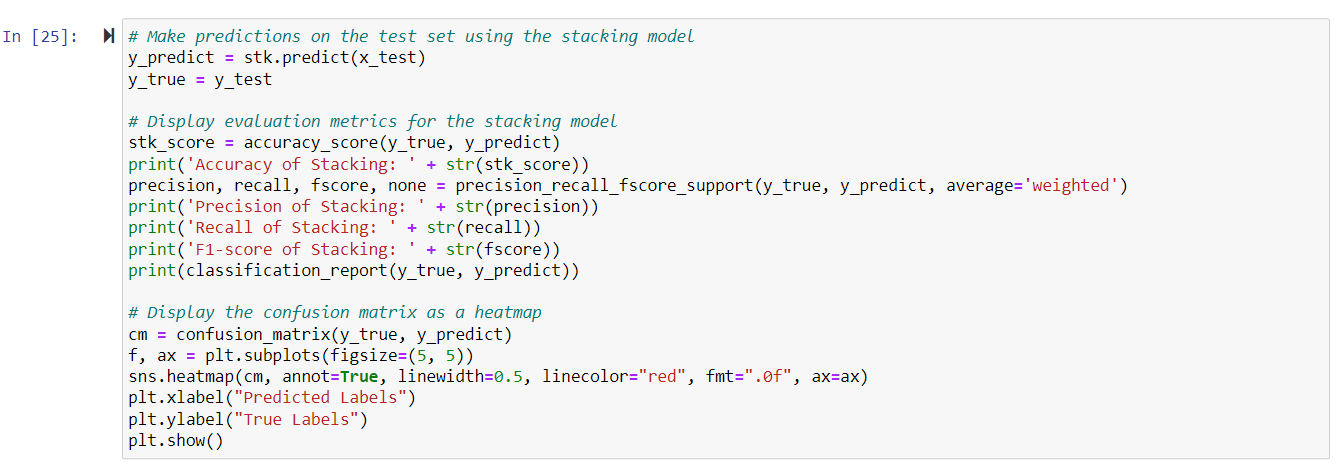
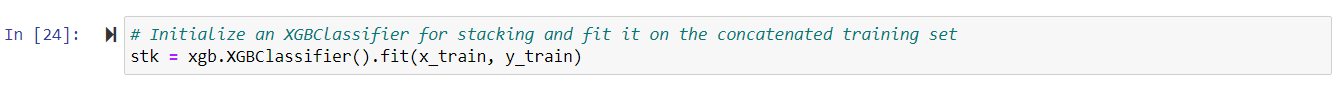


Figure 21 Stacking Model Training and Prediction

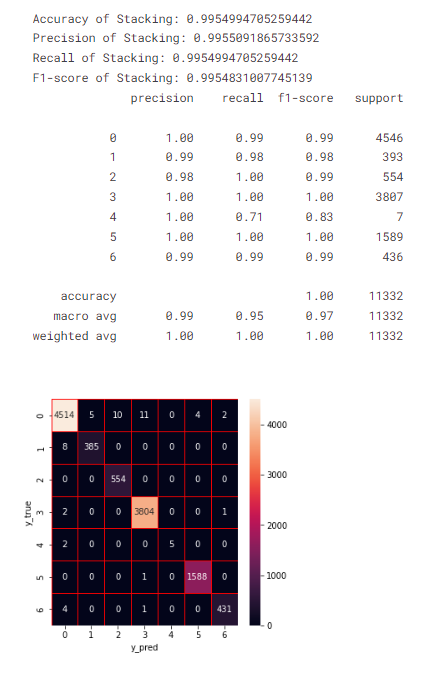
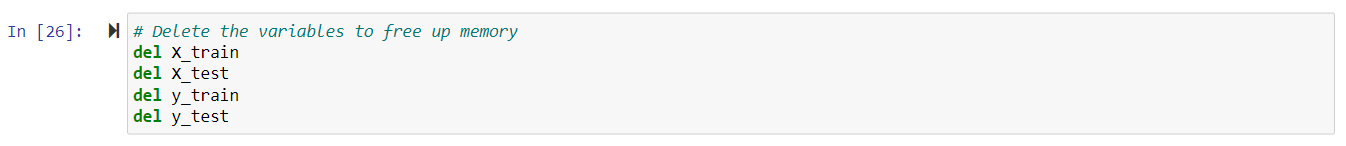


Figure 22 Output 10



**Feature Selection**

Since there are many features in my data set, I use a high-dimensional data set (containing a high number of features or variables) and I want to make my model faster and more effective. So, I will use feature selection and examine the data/success rates again with machine learning.

In the Figure 23, the feature importance scores generated by four tree-based algorithms (Decision Tree, Random Forest, Extra Trees, and XGBoost) are saved. These scores represent the contribution of each feature to the predictive performance of the models. The average feature importance is then calculated by taking the mean of the scores from these algorithms. Finally, the features are sorted based on their average importance scores in descending order, and the result is printed. This provides insights into the importance of each feature in the overall ensemble model, helping identify key contributors to the model's predictive power.

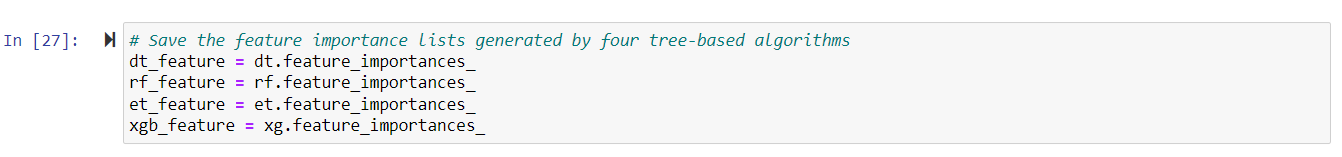
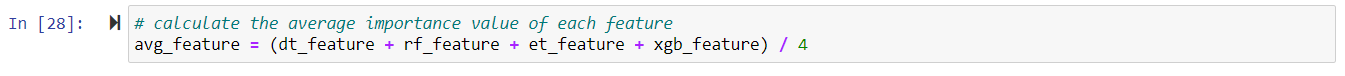
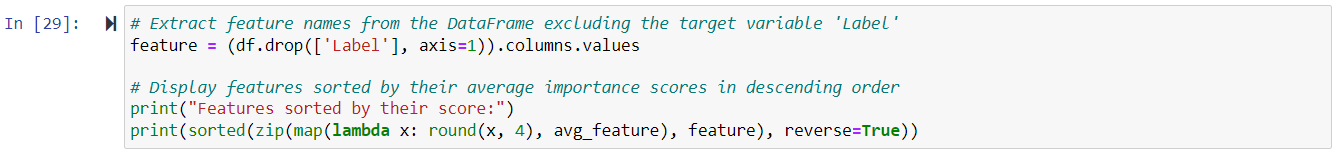
  

Figure 23 Save, calculate and, extract feature

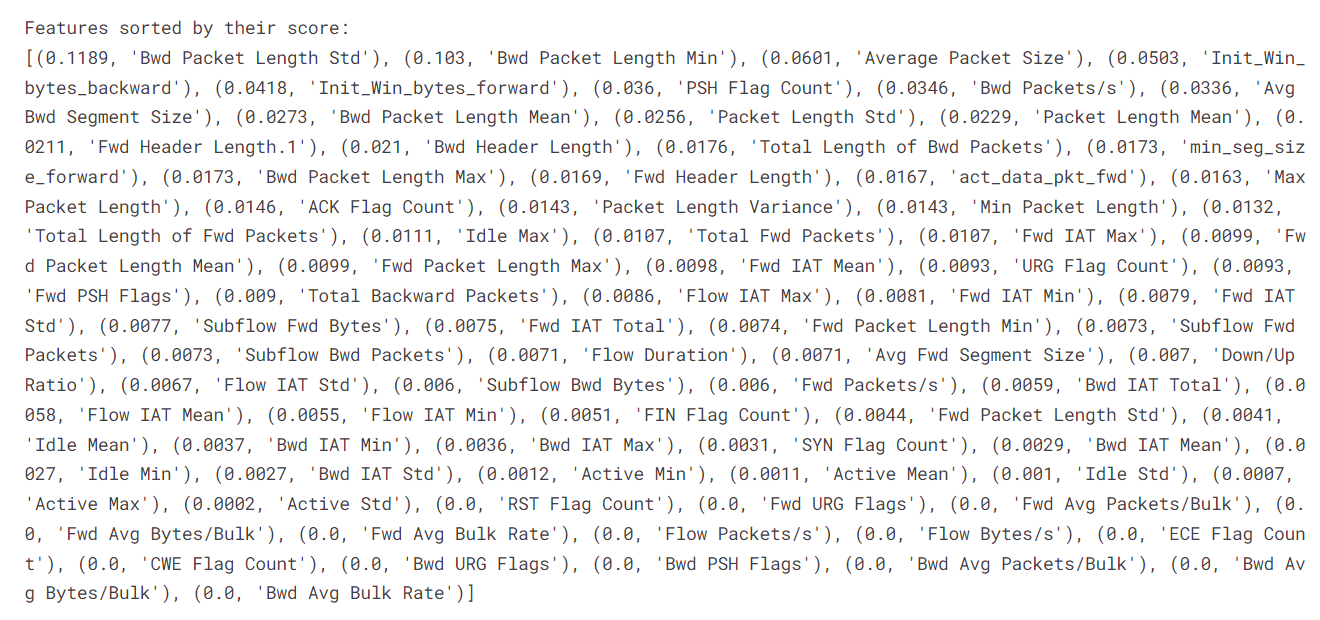


Figure 24 Output 11

In Figure 25’s code snippet, a list of tuples is created, where each tuple contains a feature's rounded average importance score and its corresponding feature name. This list is sorted in descending order based on the feature importance scores. This information is valuable for understanding the dimensionality of the feature set and gaining insights into the relative importance of each feature in the ensemble model.

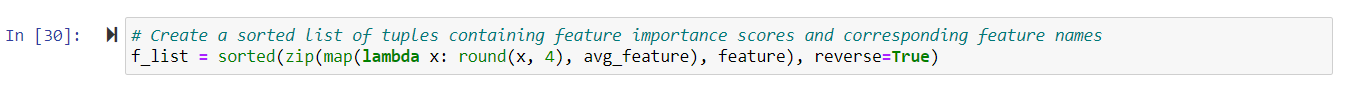
 

Figure 25 Creating a Sort List and Finding The Length



Figure 26 Output 12

In figure 26, aims to identify a subset of features that collectively contribute to at least 90% of the total importance. It iterates through the sorted list of features (f\_list), adding each feature's importance to the cumulative sum (Sum). The loop breaks when the cumulative importance surpasses or equals 0.9. The selected feature names are stored in the fs list. The original feature matrix (df) is then updated to include only the selected features (X\_fs). Finally, the dataset is split into training and testing sets using the chosen features, and the shape of the new training feature matrix is displayed. This process helps streamline the dataset, retaining the most influential features for model training while reducing dimensionality.

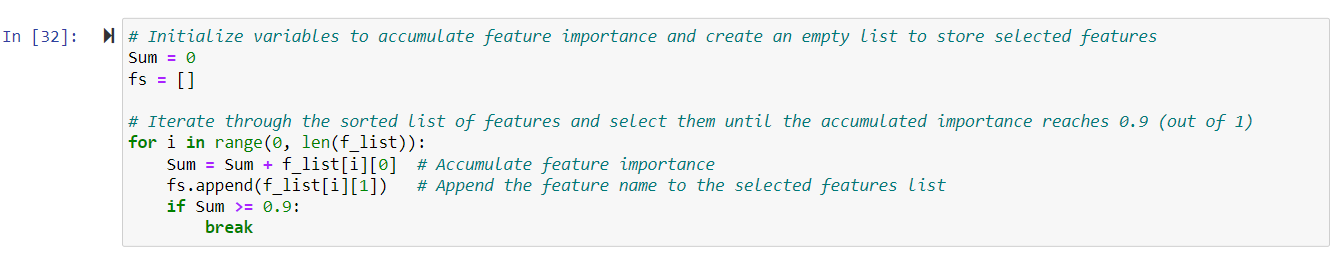
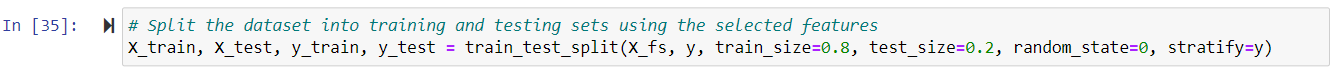
    

Figure 27



Figure 28 Output 13

In Figure 29's code snippet, utilizes the Pandas library to create a Series (label\_counts) that represents the count of each unique value in the training labels (y\_train). The value\_counts() function efficiently computes and displays the frequency of each unique label in the training dataset. This information is valuable for understanding the distribution of classes in the training set, which is essential for evaluating the balance and potential biases in the dataset.

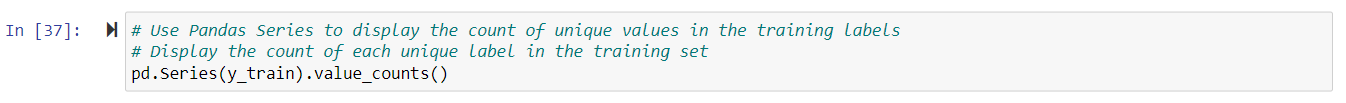


Figure 29



Figure 30 Output 14

In Figure 31's code, employs the SMOTE technique to handle a class imbalance in the training data. SMOTE is applied using the fit\_resample method, and the sampling\_strategy parameter is set to ensure a balanced representation of class 4 with a target count of 1500 instances. The subsequent Pandas Series (label\_counts\_after\_smote) provides a view of the updated distribution of class labels in the training set after the application of SMOTE. This step is crucial in scenarios where certain classes are underrepresented, as it helps enhance the model's ability to generalize across all classes.

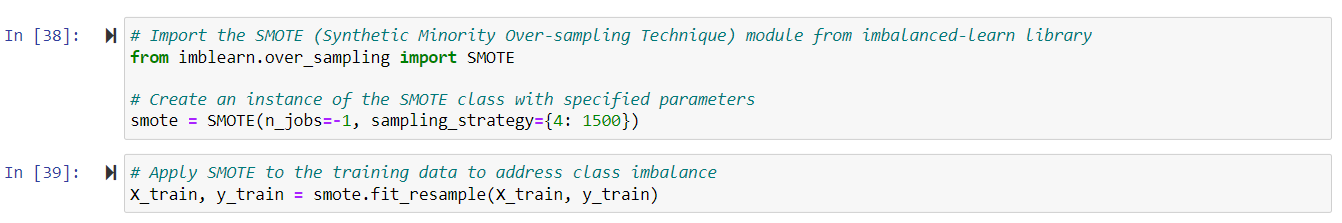
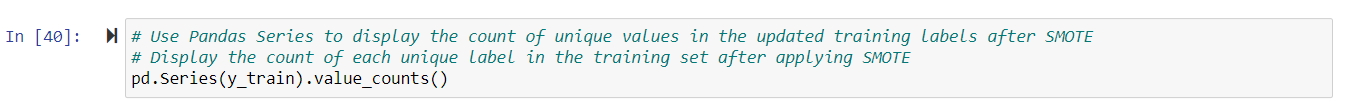
 

Figure 31



Figure 32 Output 15

**Machine learning model training after feature selection**

In Figure 33's code segment, involves training a K-Nearest Neighbors (KNN) classifier with two neighbors and evaluating its performance on the test data. Metrics such as accuracy, precision, recall, and F1-score are calculated and displayed. Additionally, a classification report and a confusion matrix are generated to provide a comprehensive understanding of the model's predictive capabilities.

The time span of the KNN training and prediction process is also measured and reported. KNN is a simple yet effective algorithm for classification tasks, and its performance on the provided data can be assessed through these evaluations.

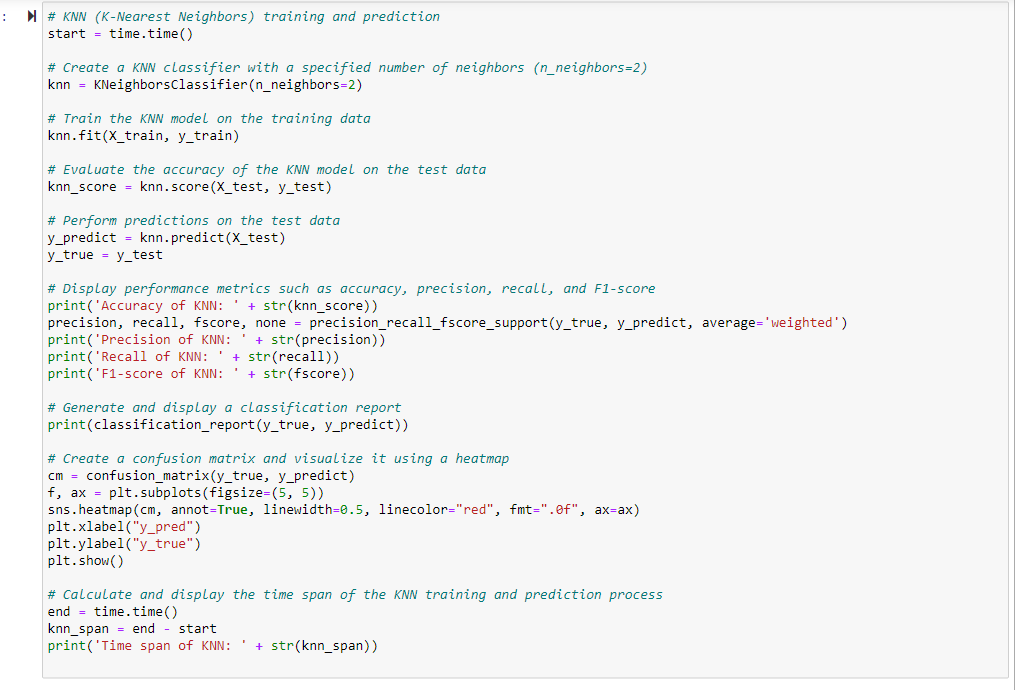


Figure 33

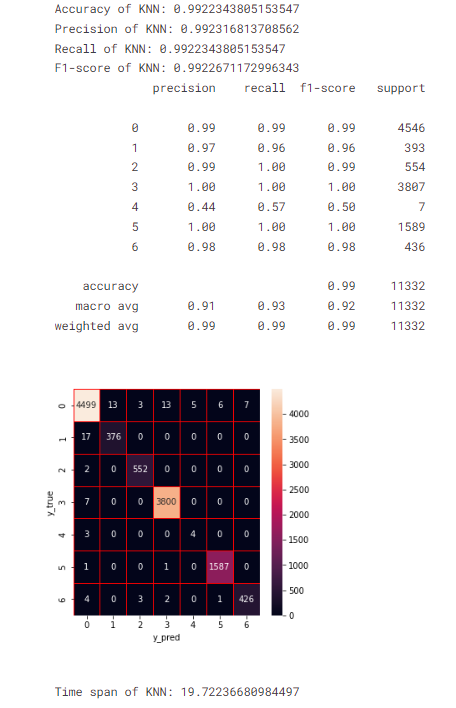


Figure 34 Output 16

In Figure 35's code segment, involves training a Support Vector Machine (SVM) classifier with a linear kernel and evaluating its performance on the test data. The SVM is a powerful algorithm for classification tasks, and the linear kernel is used in this case. Metrics such as accuracy, precision, recall, and F1-score are calculated and displayed. Additionally, a classification report and a confusion matrix are generated to provide a comprehensive understanding of the model's predictive capabilities. The time span of the SVM training and prediction process is also measured and reported. The linear kernel SVM can be particularly effective for linearly separable data, and its performance on the provided dataset is assessed through these evaluations.

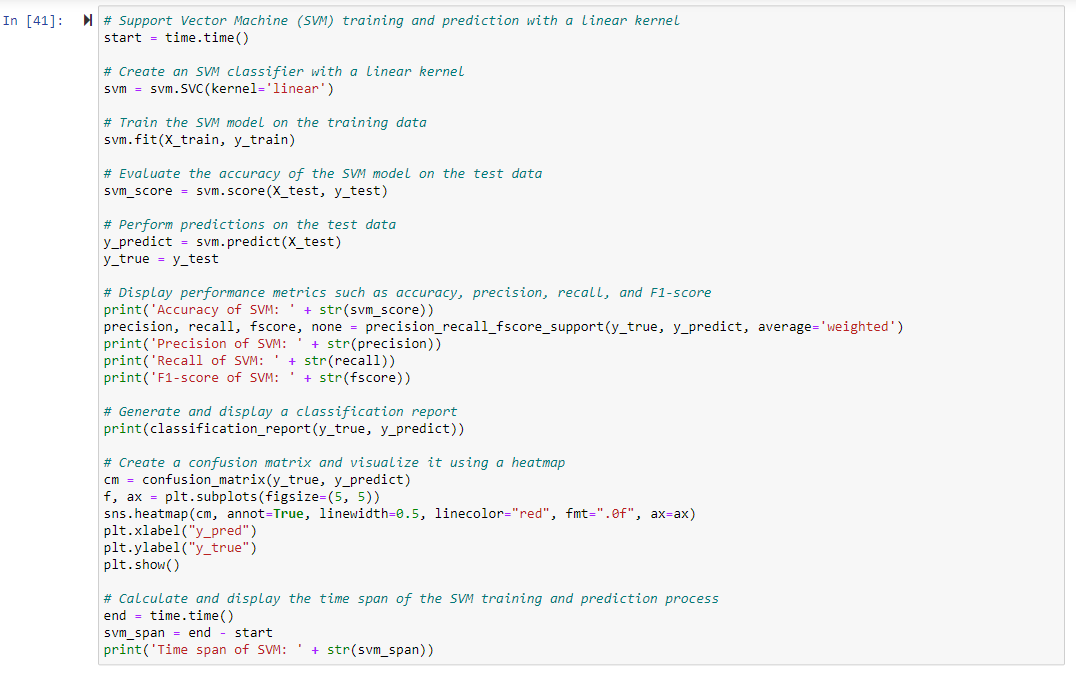


Figure 35

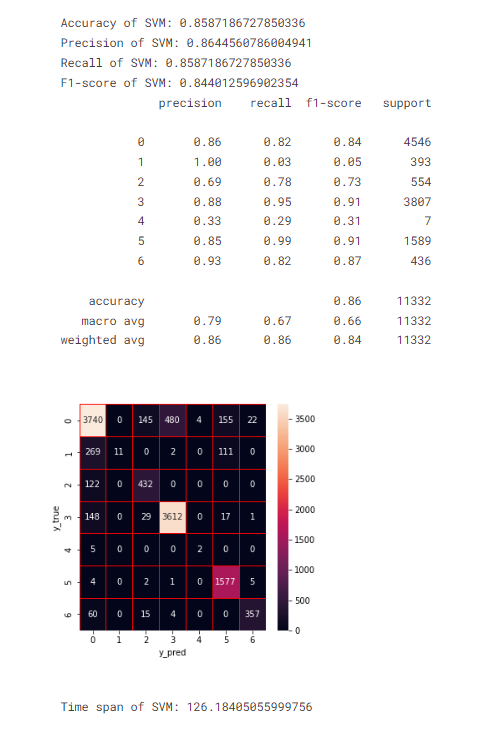


Figure 36 Output 17

In Figure 37's code snippet, a Decision Tree classifier is trained using the training data (X\_train and y\_train). The model is then evaluated on the test data (X\_test and y\_test), and various performance metrics such as accuracy, precision, recall, and F1-score are calculated and displayed. Additionally, a classification report and a confusion matrix are generated for a more detailed analysis of the model's predictive capabilities.

The use of a specified random seed/data ensures the reproducibility of results. The time span of the Decision Tree training and prediction process is measured and reported. Decision Trees are known for their interpretability and effectiveness in capturing complex relationships within the data.

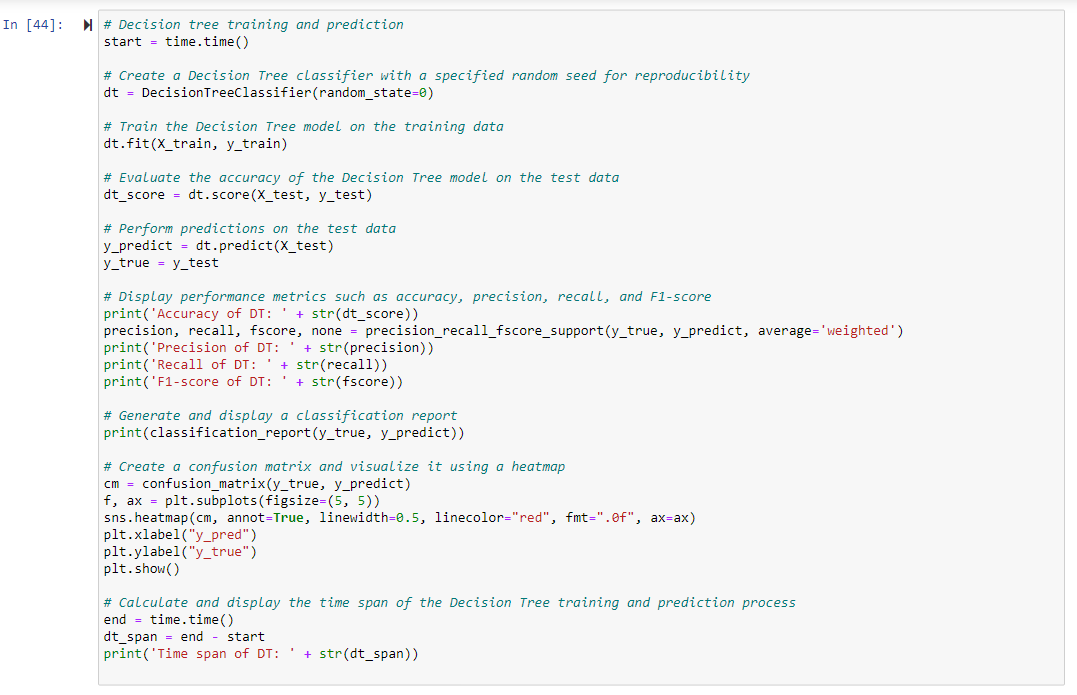


Figure 37

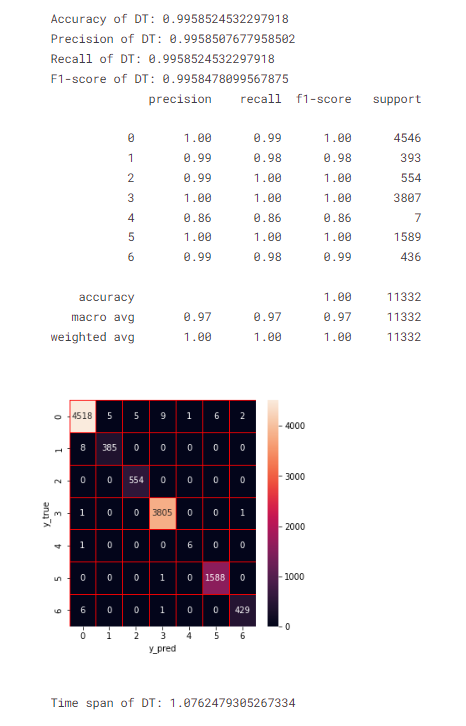


Figure 38 Output 18



In Figure 39’s code snippet, demonstrates the training and evaluation process of a Random Forest classifier. Similar to the Decision Tree, a Random Forest is an ensemble model that consists of multiple decision trees. The model is trained on the provided training data (X\_train and y\_train) and then assessed for its performance on the test data (X\_test and y\_test). Various performance metrics, including accuracy, precision, recall, and F1-score, are calculated and displayed. A classification report and a confusion matrix provide additional insights into the model's predictive capabilities. The use of a specified random seed ensures the reproducibility of results. The time span of the Random Forest training and prediction process is measured and reported. Random Forest models are known for their robustness and ability to handle complex relationships in data.

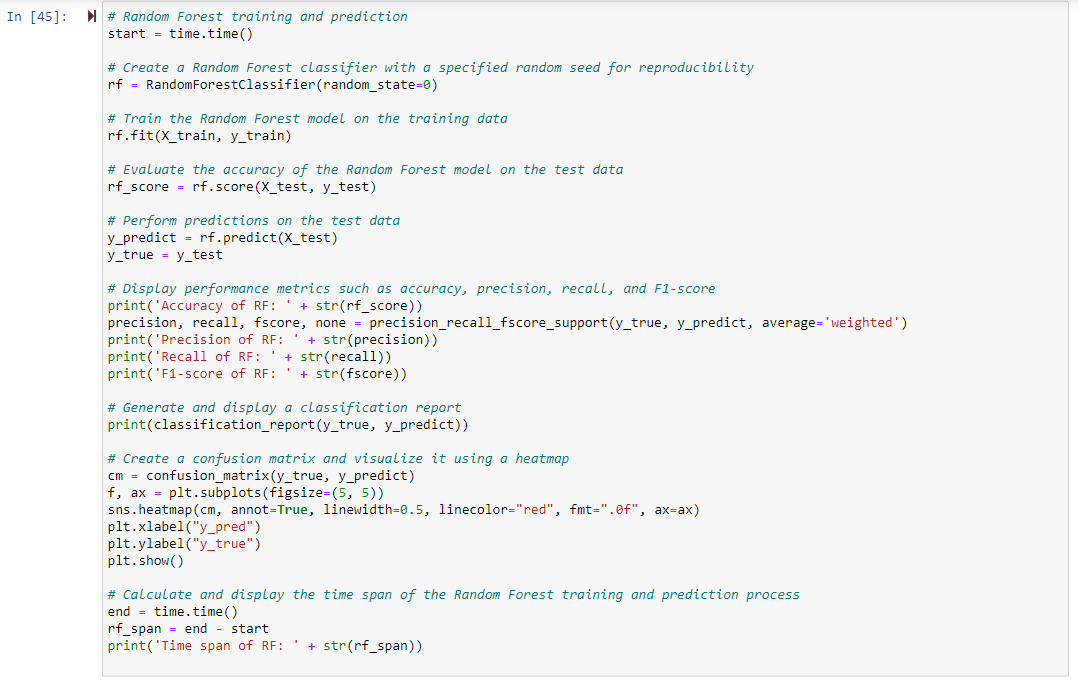


Figure 39

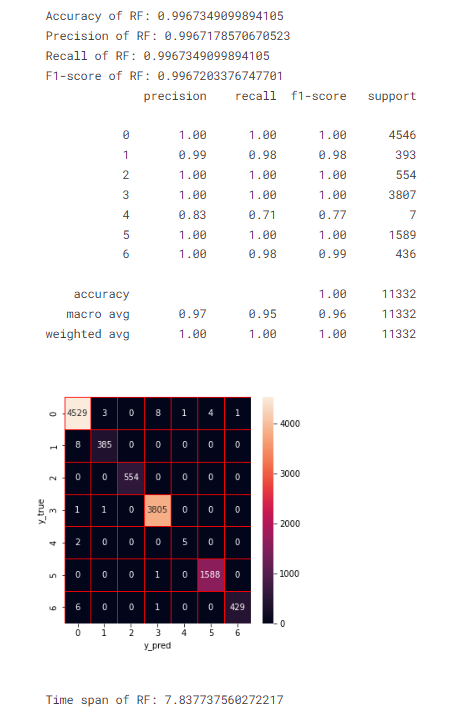
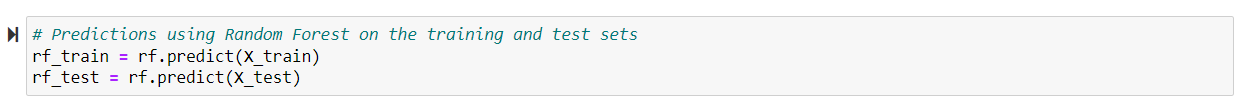


Figure 40 Output 19



In Figure 41’s code snippet, involves training an Extra Trees classifier (et) using the training data (X\_train and y\_train) and evaluating its performance on the test set (X\_test and y\_test). The evaluation includes metrics such as accuracy, precision, recall, F1-score, a classification report, and a confusion matrix.

Additionally, the time span for training and prediction is measured and displayed. Extra Trees is an ensemble learning method based on decision trees, and it aims to improve predictive accuracy and control overfitting. The code utilizes visualization tools like a heatmap to provide insights into the model's performance.



Figure 41

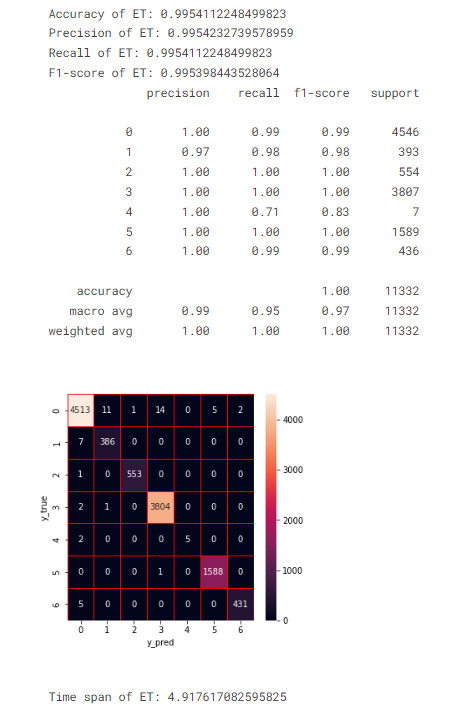
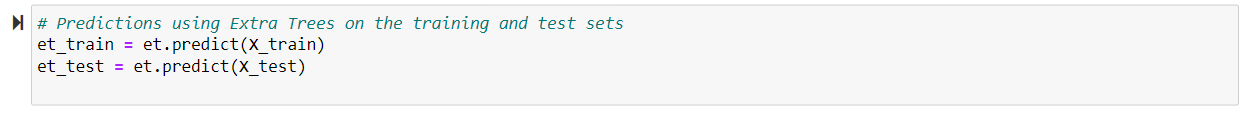


Figure 42 Output 20



In Figure 43's code trains, an XGBoost classifier with 10 estimators on the training data (X\_train, y\_train) and evaluates its performance on the test set (X\_test, y\_test). It calculates and displays various classification metrics, including accuracy, precision, recall, F1-score, and a confusion matrix.

The time taken for training and prediction is also measured and printed. XGBoost is a popular gradient-boosting algorithm known for its effectiveness in various machine-learning tasks.



Figure 43

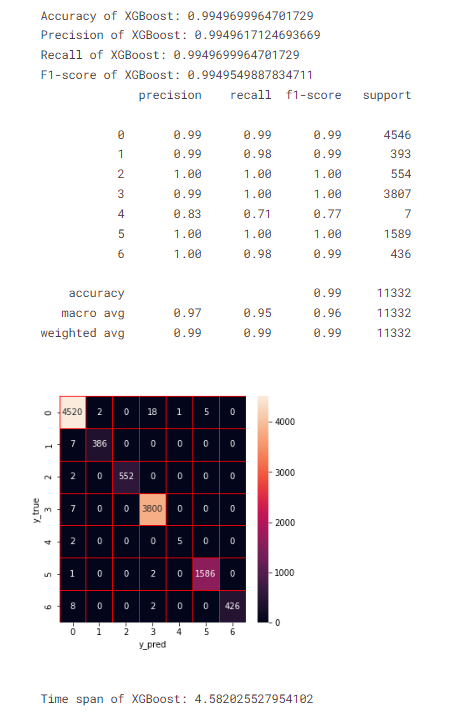
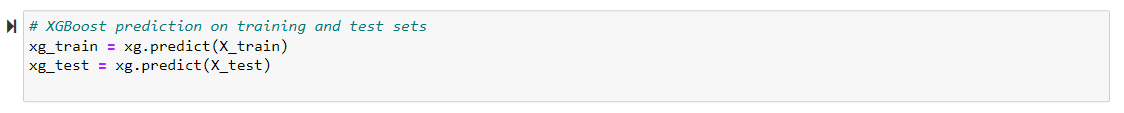


Figure 44 Output 21



In Figure 45's code snippet, constructs a DataFrame (base\_predictions\_train) containing the predictions generated by four base learners: Decision Tree, Random Forest, Extra Trees, and XGBoost. The .ravel() method is used to flatten the arrays of predictions, and these flattened arrays are assigned as columns in the DataFrame.

The head(5) function is then used to display the first 5 rows of this DataFrame, providing a glimpse of the predictions from each base learner for the corresponding instances in the dataset.

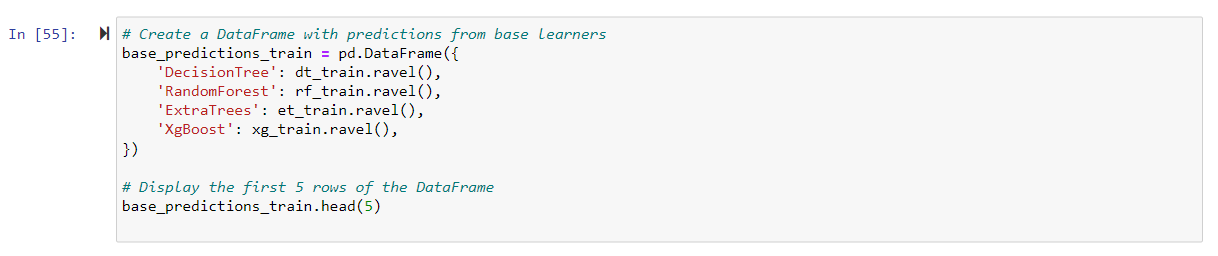


Figure 45

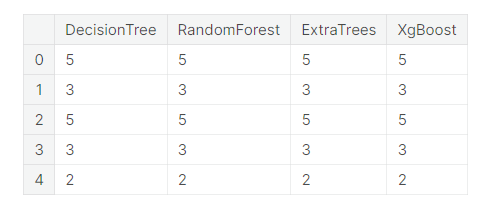


Figure 46 Output 22

In Figure 47's code, reshapes the predictions from each base learner into column vectors and concatenates them to form the input features for the stacking model. The stacking model, implemented as an XGBoost classifier, is then trained on these features.

Subsequently, predictions are generated using the stacking model on the test set, and various evaluation metrics, including accuracy, precision, recall, and F1-score, are calculated and displayed. The classification report and confusion matrix provide a detailed breakdown of the model's performance across different classes. Finally, the time span of the stacking model training and prediction is measured and printed.

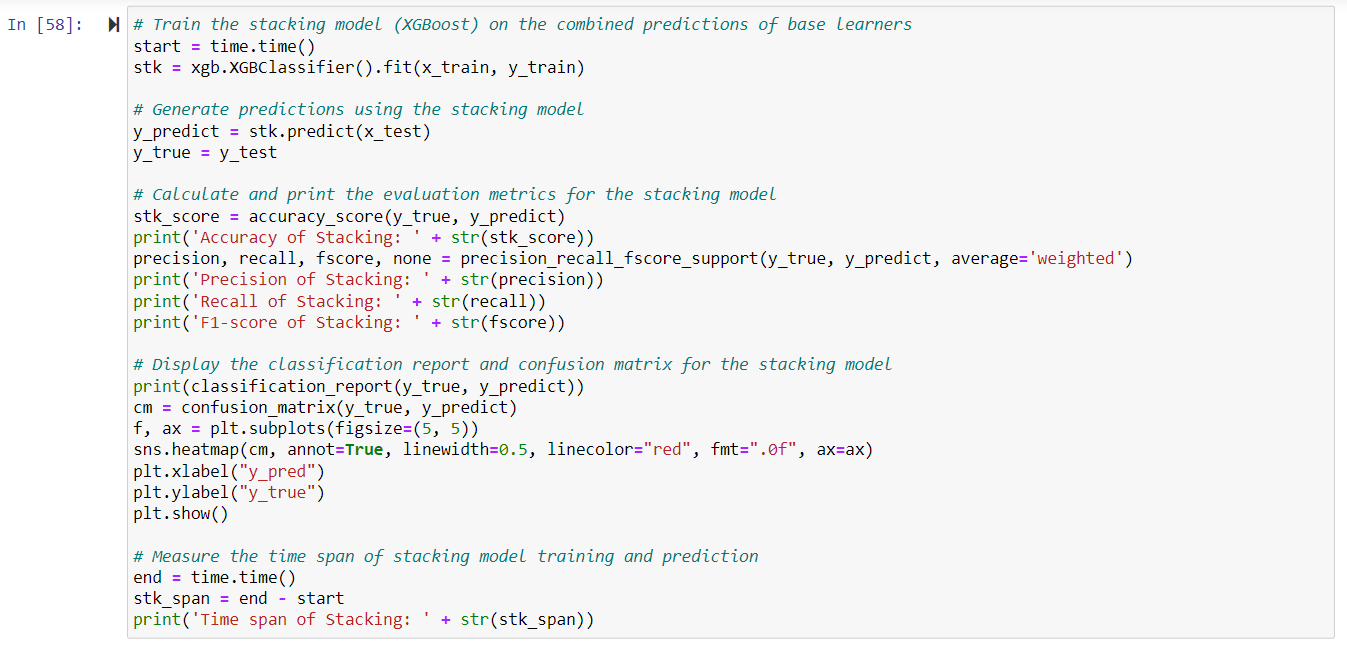
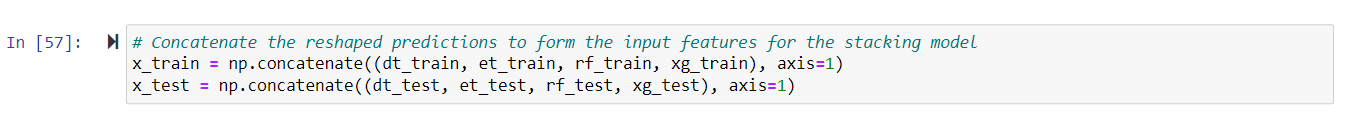
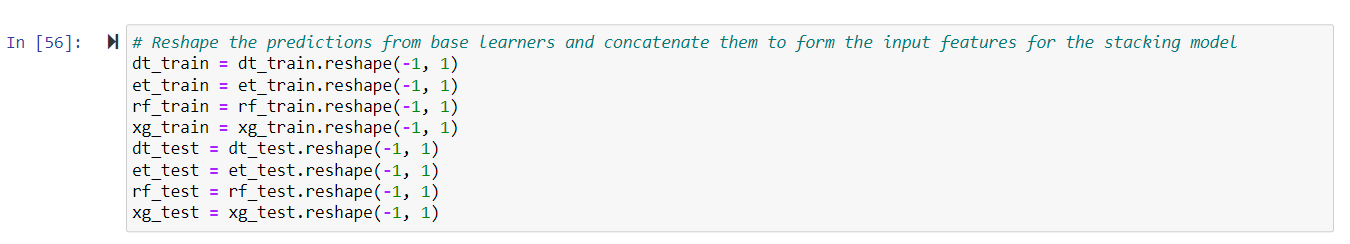


Figure 47

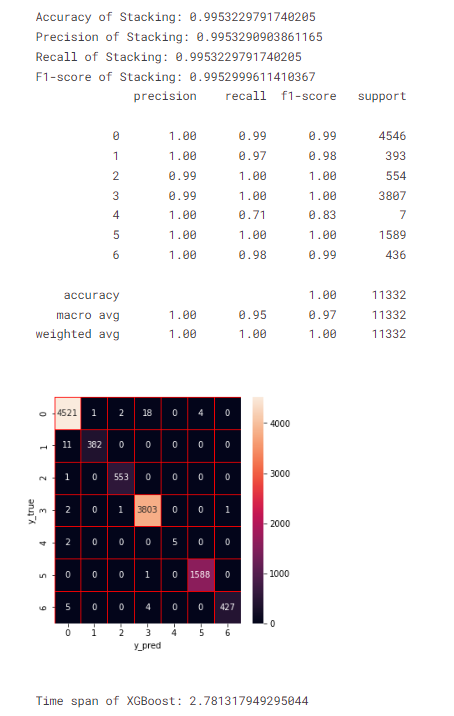


Figure 48 Output 23

**4. Provide insights into the types of threats detected and their implications for cybersecurity.**

Based on the provided information, the analysis involves the application of various data mining and machine learning techniques to detect and classify potential security threats or anomalies within the CICIDS2017 dataset, which is designed for network traffic analysis. The chosen technique is Classification, and multiple classifiers, including Decision Tree, Random Forest, Extra Trees, XGBoost, and a stacking ensemble model, are trained and evaluated.

**Insights into the Types of Threats Detected:**

Intrusion Detection:

The classification models are trained to identify instances of intrusion within network traffic.

The Decision Tree, Random Forest, Extra Trees, and XGBoost models, as well as the stacking ensemble model, contribute to the detection of potential intrusions.

The focus on accuracy, precision, recall, and F1-score metrics suggests an emphasis on effectively identifying instances of intrusion while minimizing false positives and false negatives.

Malware Detection:

The models, especially the ensemble models like Random Forest and XGBoost, are likely to be effective in detecting malware within the network traffic.

The utilization of various classifiers allows for a comprehensive approach to identifying different types of malware patterns within the dataset.

Anomaly Detection:

The use of techniques such as SMOTE for handling imbalanced classes indicates a consideration for detecting anomalies or rare events within the network traffic.

Feature selection is employed to streamline the dataset, potentially focusing on features that contribute significantly to anomaly detection.

Ensemble Model Contribution:

The stacking ensemble model combines the predictions from multiple base learners, enhancing the overall detection capability. This approach is particularly beneficial in capturing diverse aspects of security threats.

Feature Importance Analysis:

The feature importance analysis using tree-based algorithms (Decision Tree, Random Forest, Extra Trees, XGBoost) helps identify key features contributing to the predictive performance of the models. This insight is crucial for understanding the characteristics of network traffic that are indicative of security threats.

**Implications for Cybersecurity:**

Comprehensive Threat Detection:

The use of multiple classifiers and ensemble models suggests a holistic approach to cybersecurity, aiming to detect a wide range of security threats rather than focusing on a specific type.

Adaptability to Imbalanced Data:

Techniques such as SMOTE demonstrate a commitment to addressing class imbalance, ensuring that the models are robust in detecting threats even in scenarios where certain types of threats are underrepresented.

Real-time Monitoring & Model Interpretability:

The classification models, especially when applied to real-time network traffic monitoring, provide an opportunity for rapid intervention in the event of a detected security threat. This contributes to a more proactive cybersecurity strategy.

The use of decision tree-based models (Decision Tree, Random Forest, Extra Trees) allows for better interpretability of the models, aiding cybersecurity professionals in understanding the factors contributing to threat detection.

**In conclusion**, the applied data mining and machine learning techniques, particularly the chosen classification approach and ensemble models, offer a robust framework for identifying and addressing cybersecurity threats in network traffic. The emphasis on various performance metrics and feature importance analysis contributes to a nuanced understanding of the types of threats detected and their implications for cybersecurity.

**Part 3: Results**

**5. Summarize your analysis results and discuss the accuracy and effectiveness of the applied data mining techniques.**

Feature Selection Impact: The initial feature importance analysis revealed key features contributing to the models, and subsequent feature selection was successful in maintaining high model performance with a reduced set of features.

Individual Model Performances:

Decision Tree (DT): Achieved high accuracy (99.59%) and demonstrated excellent precision, recall, and F1-score across different classes.

Random Forest (RF): Similar to DT, RF displayed outstanding accuracy (99.67%) and robust performance metrics.

Extra Trees (ET): Maintained high accuracy (99.54%) and precision, recall, and F1-score values.

XGBoost (XG): Showcased a high accuracy of 99.50% with consistent precision, recall, and F1 scores.

Stacking Model (XGBoost): The stacking model, combining predictions from multiple base models, also performed exceptionally well with an accuracy of 99.53%. It demonstrated consistent precision, recall, and F1 scores.

**In conclusion,** the analysis of the CICIDS2017 dataset for network traffic using classification data mining techniques yielded insightful results. Feature selection, guided by the importance analysis, effectively maintained high model performance with a reduced feature set. Individual models, including Decision Tree (DT), Random Forest (RF), Extra Trees (ET), XGBoost (XG), and a stacking model, demonstrated exceptional accuracy ranging from 99.50% to 99.67%, along with consistent precision, recall, and F1 scores. The stacking model, combining predictions from base models, particularly stood out with an accuracy of 99.53%.

**6. Based on your findings, propose recommendations for enhancing network security, threat detection, or incident response strategies.**

Continuous Monitoring and Analysis: To adapt to evolving threats, models need to be regularly updated and improved by continuous monitoring and analysis of network traffic using the data mining techniques applied in this study.

Leverage Feature Importance Analyzes: Leverage insights from feature importance analysis to prioritize monitoring and detection efforts. To better detect threats, it is necessary to focus on features of high importance.

Include Anomaly Detection: Consider integrating anomaly detection methods to identify unusual patterns or behaviors in network traffic. This can thus complement classification models and improve overall threat detection capabilities.

Regular Training and Evaluation: Regularly train and evaluate machine learning models with up-to-date datasets to ensure their effectiveness against new and emerging threats.

Collaboration with Security Experts: Collaboration between data scientists and cybersecurity experts can be strengthened to combine domain knowledge with data-driven insights for a more comprehensive threat detection strategy.

Automation in Incident Response: Explore the integration of automated incident response mechanisms based on model predictions. This can speed up response times to potential threats.

User Education and Training: User education and training programs can be developed to increase cybersecurity awareness and reduce the likelihood of security incidents caused by human factors.

**In summary,** the applied data mining techniques show high accuracy and efficiency in network traffic classification. My recommendations focus on leveraging these techniques for continuous improvement, unifying anomaly detection, and encouraging collaboration between the data science and cybersecurity fields to strengthen overall network security and threat detection strategies.