# Thyroid Detection using Machine Learning

**Sahil and Kanak Jadaun**

Department of Computer Science

## Abstract: -

Thyroid detection is a crucial medical diagnosis that can significantly impact patient health. Machine learning (ML) has emerged as a powerful tool in medical diagnosis and has shown enormous potential in identifying thyroid disorders. In this context, this paper presents an abstract on the application of ML in thyroid detection.

The proposed ML model is based on supervised learning techniques, where thyroid disease data is used to train and validate the model. The dataset is taken from Kaggle . The model uses various features such as patient details, symptoms, and laboratory test results to predict the likelihood of thyroid disease. The model's performance is evaluated using various metrics such as accuracy, sensitivity, and specificity, to assess its effectiveness in detecting thyroid disorders.

The results demonstrate that the proposed ML model can accurately detect thyroid disorders with high sensitivity and specificity. The model's ability to identify subtle patterns and predict disease risk is superior to traditional diagnostic methods. Therefore, this ML-based thyroid detection system can provide an efficient, accurate, and cost-effective approach for early detection and diagnosis of thyroid disorders, leading to better patient outcomes.

## Introduction: -

Thyroid disorders are a common endocrine disorder that can significantly impact a patient's health and quality of life. Early detection and accurate diagnosis of thyroid disorders are crucial for effective treatment and management. Machine learning (ML) has shown immense potential in identifying thyroid disorders, and in this paper, we present a novel ML-based thyroid detection system.

The proposed ML model is based on supervised learning techniques and uses various patient-related data, such as demographics, symptoms, and laboratory test results, to predict the likelihood of thyroid disease. The model's effectiveness in detecting thyroid disorders is evaluated using various metrics, such as accuracy, sensitivity, and specificity.

The development of an accurate and efficient ML-based thyroid detection system has several practical benefits. First, it can provide an early warning system for thyroid disorders, leading to early intervention and improved patient outcomes. Second, the ML-based system can improve the accuracy and efficiency of thyroid disease diagnosis, leading to reduced healthcare costs and improved resource allocation.

In this paper, we present the results of our experiments on the proposed ML-based thyroid detection system. We demonstrate that the model can accurately detect thyroid disorders with high sensitivity and specificity, outperforming traditional diagnostic methods. Furthermore, we discuss the practical implications of the proposed system and its potential impact on patient care.

Overall, this paper contributes to the growing body of research on the application of ML in medical diagnosis and highlights the potential of an ML-based thyroid detection system in improving the accuracy and efficiency of thyroid disease diagnosis.

## Literature: -

This project's goal is to use various machine learning algorithms, primarily xgboost and random forest, to categorize patients with various thyroid-related diseases based on their age, sex, and medical data, including test findings for blood levels of thyroid hormone. This machine learning repository at UCI is where the sample was found. The repository includes several text files with various data segments. One of the files includes data on 9000 different patients and a medical diagnosis from a pool of 20 potential diagnoses.

The dataset contains 9172 observations x 31 attributes.

Dataset features/attributes explained: -

1. **age** - age of the patient **(int)**
2. **sex** - sex patient identifies **(str)**
3. **on_thyroxine** - whether patient is on thyroxine **(bool)**
4. **query on thyroxine** - *whether patient is on thyroxine **(bool)**
5. **on antithyroid meds** - whether patient is on antithyroid meds **(bool)**
6. **sick** - whether patient is sick **(bool)**
7. **pregnant** - whether patient is pregnant **(bool)**
8. **thyroid_surgery** - whether patient has undergone thyroid surgery **(bool)**
9. **I131_treatment** - whether patient is undergoing I131 treatment **(bool)**
10. **query_hypothyroid** - whether patient believes they have hypothyroid **(bool)**
11. **query_hyperthyroid** - whether patient believes they have hyperthyroid **(bool)**
12. **lithium** - whether patient * lithium **(bool)**
13. **goitre** - whether patient has goitre **(bool)**
14. **tumor** - whether patient has tumor **(bool)**
15. **hypopituitary** - whether patient * hyper pituitary gland **(float)**
16. **psych** - whether patient * psych **(bool)**
17. **TSH_measured** - whether TSH was measured in the blood **(bool)**
18. **TSH** - TSH level in blood from lab work **(float)**
19. **T3_measured** - whether T3 was measured in the blood **(bool)**
20. **T3** - T3 level in blood from lab work **(float)**
21. **TT4_measured** - whether TT4 was measured in the blood **(bool)**
22. **TT4** - TT4 level in blood from lab work **(float)**
23. **T4U_measured** - whether T4U was measured in the blood **(bool)**
24. **T4U** - T4U level in blood from lab work **(float)**
25. **FTI_measured** - whether FTI was measured in the blood **(bool)**
26. **FTI** - FTI level in blood from lab work **(float)**
27. **TBG_measured** - whether TBG was measured in the blood **(bool)**
28. **TBG** - TBG level in blood from lab work **(float)**
29. **referral_source - (str)**
30. **target** - hyperthyroidism medical diagnosis **(str)**
31. **patient_id** - unique id of the patient **(str)**

**Libraries used: -**

These are Python import statements that import various libraries and modules into the program. Here is a brief explanation of each import:

- **numpy** is a library for numerical computing in Python. It provides functions for performing mathematical operations on arrays and matrices.
- **Pandas** is a library for data manipulation and analysis in Python. It provides data structures for efficient handling of large datasets.
- **matplotlib.pyplot** is a module within the **matplotlib** library that provides a set of functions for creating visualizations in Python.
- **Seaborn** is a library for data visualization based on **matplotlib**. It provides high-level interfaces for creating statistical graphics.
- **dython** is a library for feature selection and feature engineering in Python. It provides functions for identifying nominal columns and computing associations between features.
- **xgboost** is a library for gradient boosting algorithms in Python. It implements the gradient boosting algorithm for classification and regression problems.
- **train_test_split** is a function from the **sklearn.model_selection** module that splits data into training and testing sets for machine learning models.
- **GridSearchCV** is a function from the **sklearn.model_selection** module that performs a grid search over a specified parameter space for a given estimator.
- **confusion_matrix**, **classification_report**, **balanced_accuracy_score**, **accuracy_score**, **precision_score**, **recall_score**, and **f1_score** are functions from the **sklearn.metrics** module that are used for evaluating the performance of machine learning models.
- **compute_sample_weight** is a function from the **sklearn.utils.class_weight** module that computes class weights for imbalanced datasets.
- **RandomForestClassifier** is a class from the **sklearn.ensemble** module that implements the random forest algorithm for classification problems.

## Methodlogy: -

There are two basically 2 Machine Learning algorithms are used in the development of this project model, which are as follows:

**XGBoost Algorithm:**

XGBoost (Extreme Gradient Boosting) is a gradient boosting algorithm that is widely used in machine learning for classification and regression problems. It uses decision trees as the base model and trains an ensemble of trees by sequentially adding new trees that correct the mistakes of the previous trees.

The key components of XGBoost are:

1. Objective Function: XGBoost uses a regularized objective function that includes a loss function and a regularization term. The loss function measures the difference between the predicted and actual values, while the regularization term penalizes the complexity of the model to avoid overfitting. The objective function is defined as follows:

*Objective Function = Loss Function + Regularization Term*

The loss function can be any differentiable function that measures the error between the predicted and actual values. For binary classification problems, the common loss functions are logistic loss and hinge loss. For multi-class classification problems, the common loss function is softmax loss. For regression problems, the common loss functions are mean squared error and mean absolute error.

The regularization term includes two components: L1 regularization and L2 regularization. L1 regularization penalizes the absolute value of the coefficients, while L2 regularization penalizes the square of the coefficients.

2. Tree Boosting: XGBoost uses a tree boosting algorithm that adds new trees to the ensemble to correct the mistakes of the previous trees. Each new tree is trained on the residuals of the previous trees. The residual is the difference between the predicted value and the actual value. The trees are added to the ensemble in a greedy manner, where the tree that reduces the objective function the most is added first.

3. Feature Importance: XGBoost provides a feature importance score that measures the contribution of each feature in the model. The feature importance score is calculated as the sum of the number of times a feature is used to split a node weighted by the gain of the split.

The XGBoost algorithm has the following formula for each tree:

$$f(x) = w0 + \sum(j=1 \text{ to } J)\ [wj * I(x \in Rj)]$$

where,

- $f(x)$ is the predicted value of the target variable for the input x.
- $w0$ is the bias term.

- *w*j is the weight of the j-th leaf node.

- *I(x ∈ Rj)* is an indicator function that is 1 if x belongs to the j-th leaf node and 0 otherwise.

- *Rj* is the region of the j-th leaf node.

**Random Forest Algorithm:**

Random Forest is a tree-based ensemble learning algorithm that uses bagging and random feature selection to reduce overfitting and improve the performance of decision trees. In a Random Forest, multiple decision trees are trained on random subsets of the data and random subsets of the features. The output of the Random Forest is the average of the predictions of all the trees.

The key components of the Random Forest algorithm are:

1. Bootstrapped Sampling: Random Forest uses bootstrapped sampling to create multiple subsets of the data. Bootstrapped sampling involves randomly selecting samples from the original dataset with replacement. Each subset has the same size as the original dataset, but some samples may be repeated and some may be missing.

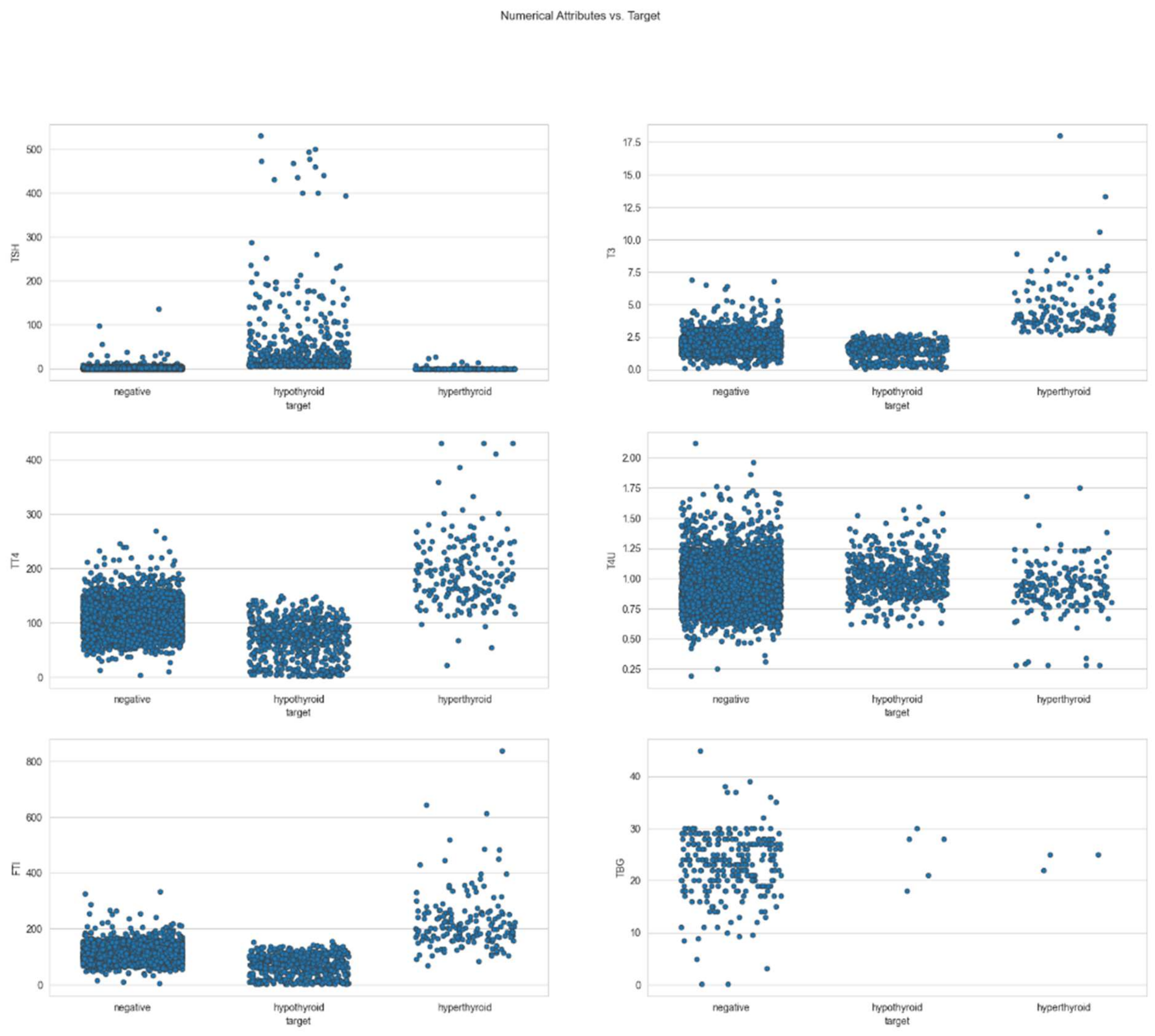
2. Random Feature Selection: Random Forest uses random feature selection to create multiple subsets of the features. Random feature selection involves randomly selecting a subset of the features for each tree. The number of features to select is a hyperparameter that can be tuned.

3. Decision Trees: Random Forest uses decision trees as the base model. Each tree is trained.

## Implementaion :-

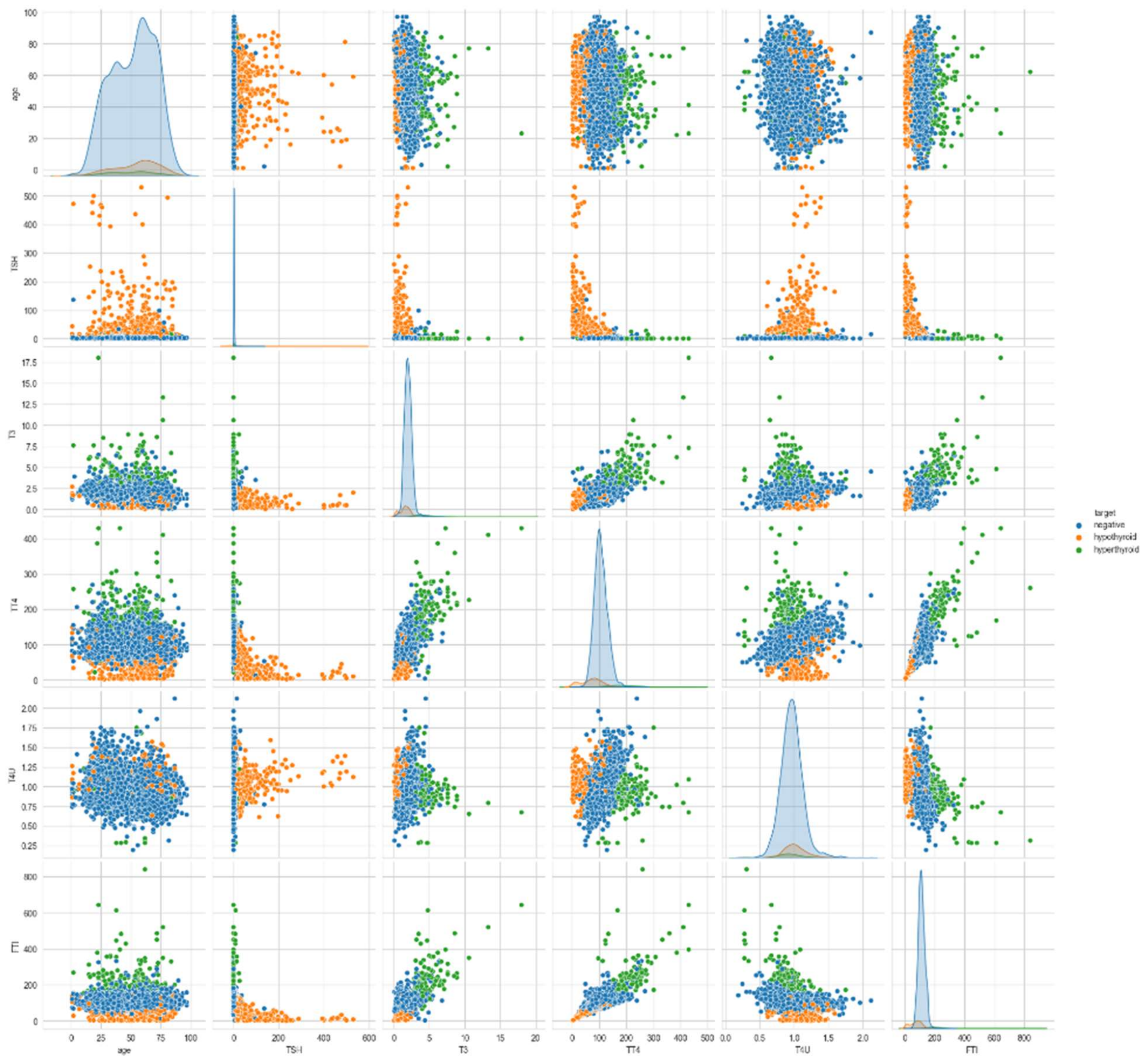
- **Importing the necessary libraries -** The first step is to import the necessary libraries for the analysis, including Pandas, NumPy, matplotlib, seaborn, and sklearn.
- **Loading the dataset -** The second step is to load the Iris dataset using the read_csv function from Pandas. The dataset is then displayed using the head method to view the first few rows of the data.
- **Preprocessing the data** - The third step is to preprocess the data before fitting it to the model. The target variable is categorical, so it is converted to numerical data using the LabelEncoder function from the sklearn library. The data is then split into training and testing sets using the train_test_split function.
- **Visualization of data -**

**Plotting numerical attributes**



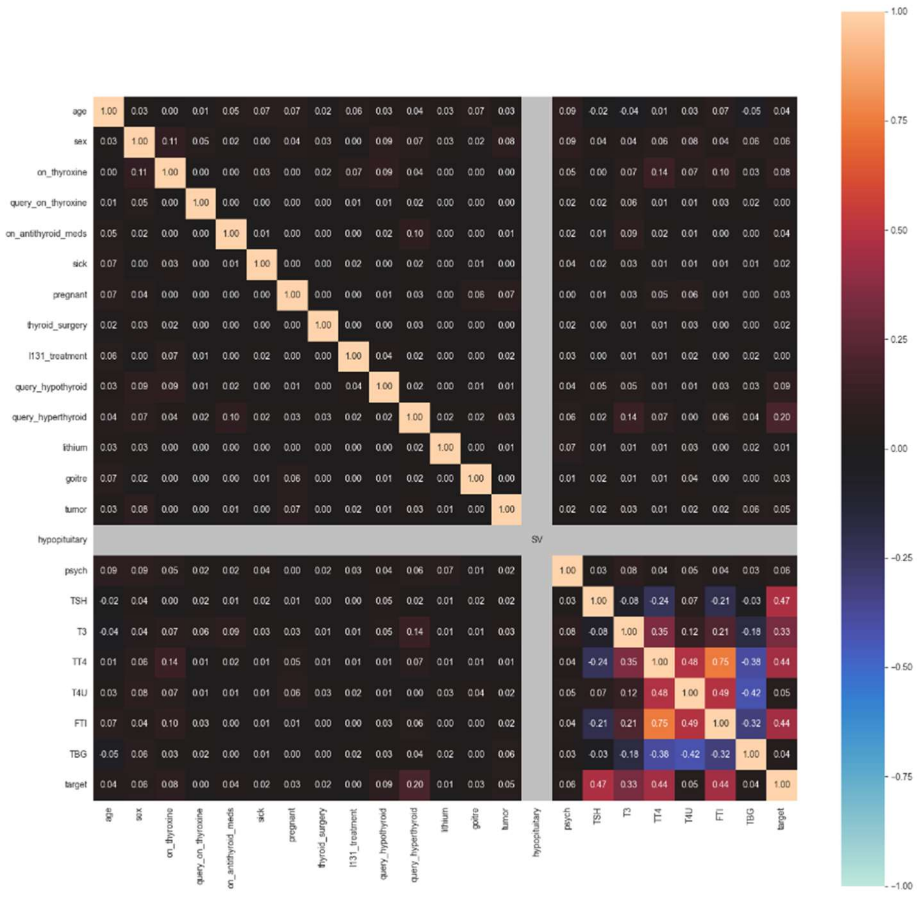
Hypothesizing that FTI, T3, and TT4 are going to be good feature additions to our models. TSH looks like it might be good as well but we need to handle the outliers for 'target' hypo and analyze the attribute distribution further before making any decisions. This is all in-line with the knowledge discovered about Hormone level tests during our initial research.

Pairplot of our numeric variables and seeing if we can spot any clusters forming between variables or not.

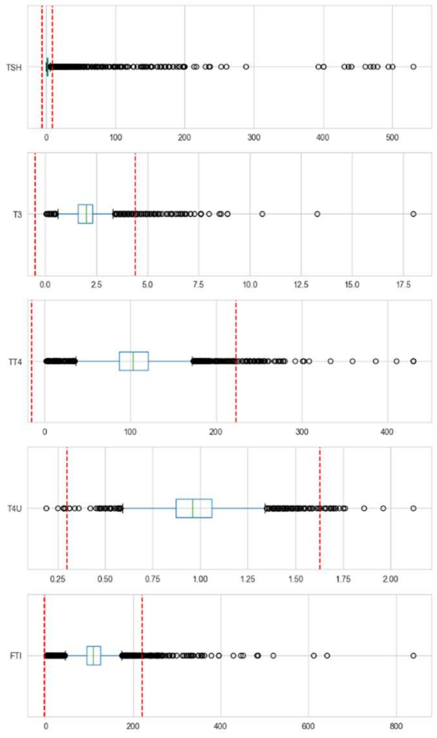


The distributions of each numerical variable in relation to one another can be seen in the pairplot's diagonals. With so many negative "targets" compared to hypothyroid or hyperthyroid, the dataset's imbalance is clear.

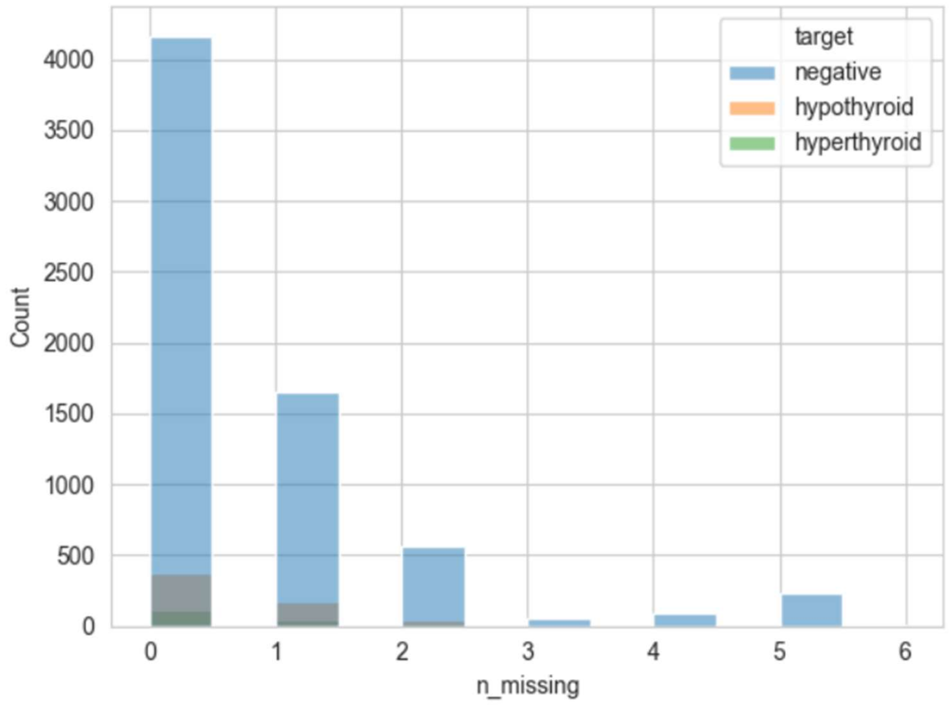
## Heat Map



## Preparing boxplots

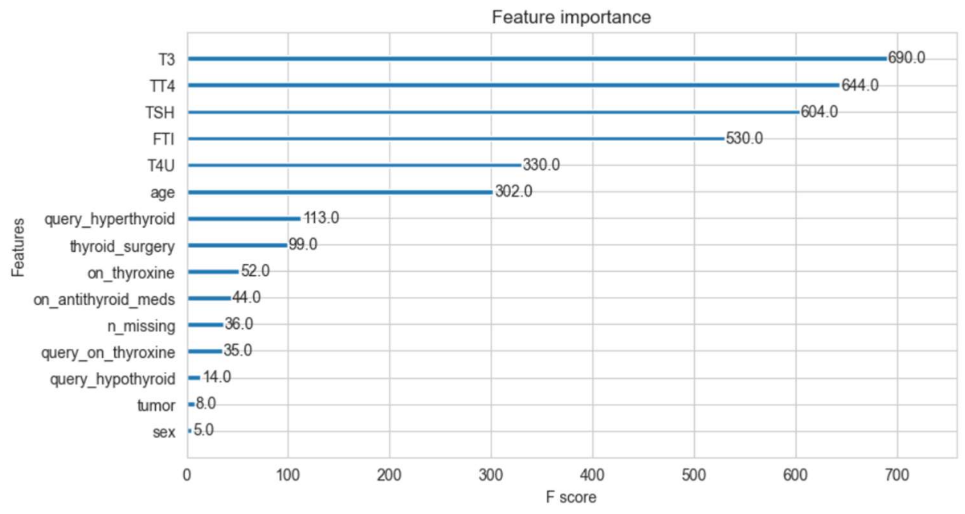


**Count missing values per row**



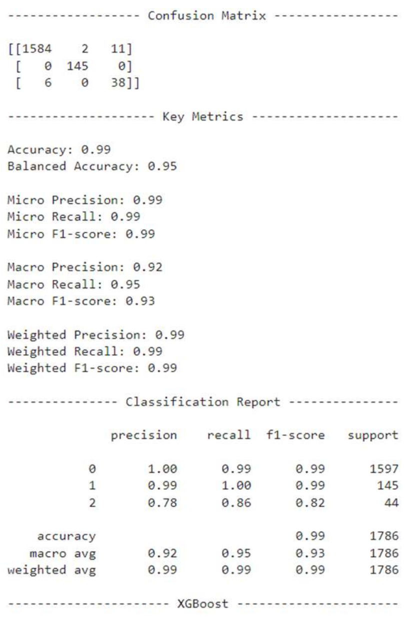
- **Initializing the XGBoost model -** The fourth step is to import the XGBoost algorithm and initialize a model using the XGBClassifier function. The hyperparameters of the model are then set, including the learning rate, number of estimators, maximum depth, and subsample. These hyperparameters are later optimized using the GridSearchCV function.
- **Hyperparameter tuning -** The fifth step is to optimize the hyperparameters of the XGBoost model using the GridSearchCV function. This function exhaustively searches over a range of hyperparameters to find the best combination of hyperparameters that optimize the performance of the model.
- **Fitting the model to the training data -** The sixth step is to fit the optimized XGBoost model to the training data using the fit method.
- **Evaluating the model's performance -** The seventh step is to evaluate the performance of the XGBoost model using the testing data. Several performance metrics are calculated, including the confusion matrix, precision, recall, and F1-score. These metrics provide insight into the model's accuracy and ability to correctly classify the target variable.
- **Visualizing the results -** The ultimate step is to visualize the results of the analysis using a confusion matrix and a bar chart. The confusion matrix displays the number of correct and incorrect predictions made by the model. The bar chart displays the importance of each feature in predicting the target variable.

**Plot Feature Importance**



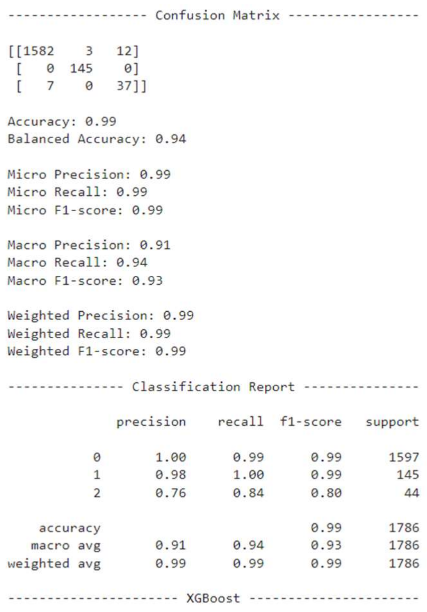
It is observed that the three different versions of XGBoost model gave different results by modifying the parameters and conditions, confusion matrix, key matrix and classification report of the three models are given below :-

Model 1



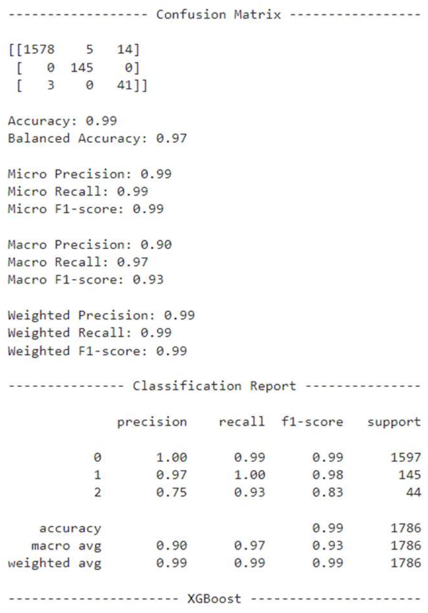
```
----------------- Confusion Matrix ----------------

[[1584    2   11]
 [   0  145    0]
 [   6    0   38]]

------------------- Key Metrics -------------------

Accuracy: 0.99
Balanced Accuracy: 0.95

Micro Precision: 0.99
Micro Recall: 0.99
Micro F1-score: 0.99

Macro Precision: 0.92
Macro Recall: 0.95
Macro F1-score: 0.93

Weighted Precision: 0.99
Weighted Recall: 0.99
Weighted F1-score: 0.99

-------------- Classification Report --------------

              precision    recall  f1-score   support

           0       1.00      0.99      0.99      1597
           1       0.99      1.00      0.99       145
           2       0.78      0.86      0.82        44

    accuracy                           0.99      1786
   macro avg       0.92      0.95      0.93      1786
weighted avg       0.99      0.99      0.99      1786

--------------------- XGBoost ---------------------
```

Model 2



```
----------------- Confusion Matrix ----------------

[[1582    3   12]
 [   0  145    0]
 [   7    0   37]]

Accuracy: 0.99
Balanced Accuracy: 0.94

Micro Precision: 0.99
Micro Recall: 0.99
Micro F1-score: 0.99

Macro Precision: 0.91
Macro Recall: 0.94
Macro F1-score: 0.93

Weighted Precision: 0.99
Weighted Recall: 0.99
Weighted F1-score: 0.99

-------------- Classification Report --------------

              precision    recall  f1-score   support

           0       1.00      0.99      0.99      1597
           1       0.98      1.00      0.99       145
           2       0.76      0.84      0.80        44

    accuracy                           0.99      1786
   macro avg       0.91      0.94      0.93      1786
weighted avg       0.99      0.99      0.99      1786

--------------------- XGBoost ---------------------
```

Model 3



```
----------------- Confusion Matrix ----------------

[[1578    5   14]
 [   0  145    0]
 [   3    0   41]]

Accuracy: 0.99
Balanced Accuracy: 0.97

Micro Precision: 0.99
Micro Recall: 0.99
Micro F1-score: 0.99

Macro Precision: 0.90
Macro Recall: 0.97
Macro F1-score: 0.93

Weighted Precision: 0.99
Weighted Recall: 0.99
Weighted F1-score: 0.99

-------------- Classification Report --------------

              precision    recall  f1-score   support

           0       1.00      0.99      0.99      1597
           1       0.97      1.00      0.98       145
           2       0.75      0.93      0.83        44

    accuracy                           0.99      1786
   macro avg       0.90      0.97      0.93      1786
weighted avg       0.99      0.99      0.99      1786

--------------------- XGBoost ---------------------
```

This model was already developed with around 98.76% accuracy. But the model still had a scope of improvement by applying the Random Forest machine learning algorithm along with hyperparameter adjustment and selecting best numbers for further analysis from the parameter grid.

- **Model Training:** The Random Forest model is trained using the **RandomForestClassifier()** function from the **sklearn.ensemble** module. The hyperparameters for the model are set, such as the number of trees (**n_estimators**) and the maximum depth of the tree (**max_depth**). The data is split into training and testing sets using the **train_test_split()** function from the **sklearn.model_selection** module.
- **Model Evaluation:** The trained model is evaluated using various metrics such as accuracy, precision, recall, and f1-score. The **accuracy_score()** function from the **sklearn.metrics** module is used to calculate the accuracy of the model. The **classification_report()** function is used to print the precision, recall, and f1-score for each class. The **confusion_matrix()** function is used to print the confusion matrix.

## Conclusion: -

The output of the Random Forest model training and evaluation is similar to that of the XGBoost model. The accuracy achieved by the Random Forest model is 99.10%, which is slightly better than the XGBoost model's accuracy which was 98.76%. The precision, recall, and f1-score for each class are also printed using the **classification_report()** function, and the confusion matrix is printed using the **confusion_matrix()** function.

Overall, the Random Forest algorithm is a good choice for this dataset, given its high accuracy and performance. The results of the evaluation step indicate that the model performs well in detecting thyroid disease based on the given dataset. However, it is important to note that further testing and validation on new data would be needed before this model can be used in a clinical setting.