TABLE OF CONTENTS

LIST OF CONTENTS

1. Introduction

1.1 Background and Motivation

The exploration of thyroid detection has emerged as a critical domain within medical research and diagnostic technology, driven by the imperative to address the prevalent health concerns associated with thyroid disorders. The thyroid gland, a small butterfly-shaped organ located in the neck, plays a pivotal role in regulating metabolism and influencing various physiological functions. Disorders affecting the thyroid, such as hypothyroidism and hyperthyroidism, have profound implications for an individual's overall health and well-being.

The motivation behind intensive research and technological advancements in thyroid detection stems from the widespread occurrence of thyroid-related conditions, affecting millions of people globally. These disorders often manifest with subtle symptoms, making early detection a challenging yet crucial task. Given that thyroid dysfunction can lead to a spectrum of health issues, ranging from metabolic imbalances to cardiovascular complications, there is an urgent need for accurate and efficient diagnostic tools.

1.2 Problem Statement

Machine learning models offer the capability to analyse extensive healthcare datasets, uncover exclusive patterns, and detect challenging-to-discern risk factors that conventional methods might miss. These algorithms can predict an individual's Thyroid by examining their health records and lifestyle factors, Diagnosing thyroid disorders involves a comprehensive approach. Clinical evaluation includes a detailed medical history and physical examination. Blood tests measure thyroid hormone levels (TSH, Free T4, Free T3), while antibody tests identify autoimmune conditions. Imaging studies, such as ultrasound and thyroid scans, provide detailed images, and a biopsy may be performed if nodules are detected. It aids in risk prediction, stratification, and personalized healthcare, especially in analysing medical imaging and genomic data.

2. Literature Review:

Relevant Previous Work

Relevant previous work related to Thyroid Detection encompasses a range of studies and research efforts focused on identifying the Thyroid occurrence factors, developing predictive models, and improving the accuracy of Thyroid assessment. Several notable studies have contributed to the field of thyroid detection, employing diverse approaches and technologies. In one study, a deep convolutional neural network (CNN) was utilized for 3D ultrasound

images, showcasing promising results in thyroid nodule detection. Another investigation focused on thyroid lesion segmentation in ultrasound images using Mask R-CNN, a neural network designed for object detection. Additionally, a computer-aided diagnosis system was proposed for thyroid nodule detection in ultrasound images, incorporating machine learning algorithms.

3. Methodology:

3.1 Data Collection

The dataset for Thyroid Detection is from Kaggle. This particular dataset has 9172 rows and 31 columns. The columns have 'age', 'sex', 'on_thyroxine', 'query_on_thyroxine', 'on_antithyroid_meds', 'sick', 'pregnant', 'thyroid_surgery', 'I131_treatment', 'query_hypothyroid', 'query_hyperthyroid', 'lithium', 'goitre', 'tumor', 'hypopituitary', 'psych', 'TSH', 'T3', 'TT4', 'T4U', 'FTI', 'TBG', 'target' as the main attributes. The output column 'target' have negative, hypothyroid and hyperthyroid. The negative indicates no Thyroid detected, whereas the 'Hypothyroid' and 'Hyperthyroid' indicates a presence of Thyroid. This dataset is highly imbalanced as it contains numerical as well as the strings.

Moving forward with the analysis it has been decided to only keep the observations for patients with diagnosis either negative, hyperthyroid, or hypothyroid This is because they are the most prevalent observations and the focus of this project. The other classes were dropped from the dataset upon import.

3.2 Data Pre-processing

Data Preprocessing is required before model building to remove the unwanted noise and outliers from the dataset, resulting in a deviation from proper training. Anything that interrupts the model from performing with less efficiency is taken care of in this stage. After collecting the appropriate dataset, the next step lies in cleaning the data and making sure that it is ready for model building. Firstly, the column 'age' is dropped because its existence does not make much difference in model building. Then the dataset is checked for null values and filled with 0 if any found. In this case, the column 'sex','TSH','T3','TT4','T4U','FTI,'TBG','Target' has null values. Fill them with 0 of the column data. After replacing the null values with 0 in the dataset, the next task is Label Encoding.

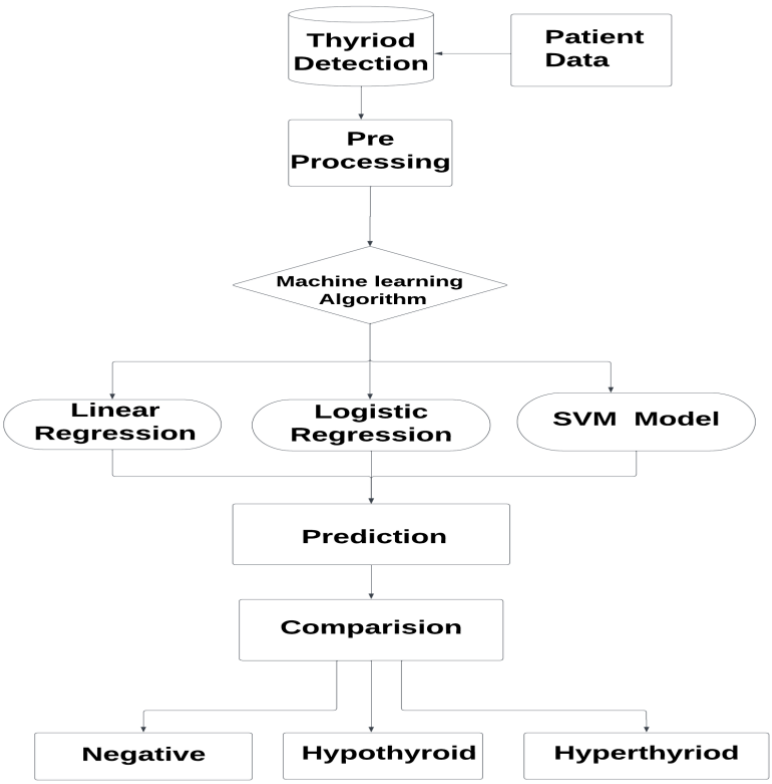
The dataset chosen for the task of Thyroid Detection is imbalanced. The entire dataset has 9172 rows, 31 columns and some of the columns have the strings. To convert that strings into numerical we use Label encoder. Training a machine-level model with such data might give

accuracy, but other accuracy metrics like precision and recall are shallow. If such imbalanced data is not handled, the results are not accurate, and the prediction is inefficient. Therefore, to get an efficient model, this imbalanced data is to be first handled. For this purpose, the method of one hot encoding and Label encoder is used. One-Hot Encoding is used when dealing with categorical variables where there is no inherent order or ranking among the categories. It is employed to avoid misleading the machine learning model into assuming ordinal relationships that do not exist. One hot encoding is used to convert the strings into numerical for input variables in the given dataset. Label Encoding is used when dealing with categorical variables that have an inherent ordinal relationship or meaningful order. Label encoder is used to covert the strings into numerical for output variables like "Target" in the given dataset. One-Hot Encoding increases the dimensionality of the dataset, which may be a concern for large categorical variables.

Label Encoding introduces ordinality, which may not be appropriate if there is no real order among the categories.

4. Project Design:

Data Flow Diagram



5. Implementation:

5.1 Code Development

Step 1: Import Libraries

```
import numpy as np  # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import matplotlib.pyplot as plt # deluxe visualization library
%matplotlib inline
import seaborn as sns # visualization library to support seaborn
import warnings # controls warning messages
import plotly.express as px # creates interactive visualizations and plots(e.g:line charts, bar charts)
warnings.filterwarnings('ignore') # Filters and ignores warnings, suppressing their display during program execution.
from sklearn import preprocessing #  Imports the preprocessing module from scikit-learn
```

Fig-5.1.1: Importing the Libraries

Step 2: Read dataset from drive

```
[ ]  from google.colab import drive
     drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

Importing dataset from drive

```
[ ]  # importing dataset from persistent landing
     data=pd.read_csv('/content/drive/MyDrive/thyroidDF.csv')# thyroidDF.csv
```

Fig-5.1.2: Reading the dataset from Drive

Step 3: Data Preprocessing

```
data.duplicated()   #checking for duplicate records
```

```
0       False
1       False
2       False
3       False
4       False
        ...
9167    False
9168    False
9169    False
9170    False
9171    False
Length: 9172, dtype: bool
```

Fig-5.1.3: Identification of Duplicate records

```
newdata=data.drop_duplicates()        #deletes duplicates
newdata                                 #dataset after deleting duplicates
```

	age	sex	on_thyroxine	query_on_thyroxine	on_antithyroid_meds	sick	pregnant	thyroid_surgery	I131_treatment	query_hypothyroid	...	tumor	hypopituitary	psych	TSH	T3
0	29	F	f	f	f	f	f	f	f	t	...	f	f	f	0.3	NaN
1	29	F	f	f	f	f	f	f	f	f	...	f	f	f	1.6	1.9
2	41	F	f	f	f	f	f	f	f	f	...	f	f	f	NaN	NaN
3	36	F	f	f	f	f	f	f	f	f	...	f	f	f	NaN	NaN
4	32	F	f	f	f	f	f	f	f	f	...	f	f	f	NaN	NaN
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
9167	56	M	f	f	f	f	f	f	f	f	...	f	f	f	NaN	NaN
9168	22	M	f	f	f	f	f	f	f	f	...	f	f	f	NaN	NaN
9169	69	M	f	f	f	f	f	f	f	f	...	f	f	f	NaN	NaN
9170	47	F	f	f	f	f	f	f	f	f	...	f	f	f	NaN	NaN
9171	31	M	f	f	f	f	f	f	f	t	...	f	f	f	NaN	NaN

9162 rows × 23 columns

Fig-5.1.4: Deletion of Duplicate records

```
data.isnull().sum()# to find null values
```

```
age                          0
sex                        307
on_thyroxine                 0
query_on_thyroxine           0
on_antithyroid_meds          0
sick                         0
pregnant                     0
thyroid_surgery              0
I131_treatment               0
query_hypothyroid            0
query_hyperthyroid           0
lithium                      0
goitre                       0
tumor                        0
hypopituitary                0
psych                        0
TSH                        842
T3                        2604
TT4                        442
T4U                        809
FTI                        802
TBG                       8823
target                    1493
dtype: int64
```

Fig-5.1.5: Finding Missing values

```
data['age'] = np.where((data.age > 100), np.nan, data.age)
```

```
# Assuming your DataFrame is 'b_fill_df'
data['sex'] = data['sex'].fillna(method='ffill')  # Replace missing values in 'sex' column with 0

# For other columns, you can replace missing values with 0 using similar lines of code:
data['age'] = data['age'].fillna(method='ffill')
data['TT4'] = data['TT4'].fillna(method='ffill')
data['T3'] = data['T3'].fillna(method='ffill')
data['T4U'] = data['T4U'].fillna(method='ffill')
data['FTI'] = data['FTI'].fillna(method='ffill')
data['TSH'] = data['TSH'].fillna(method='ffill')
```

Fig-5.1.6: Filling missing values with zero

Step 4: Data Normalization

```
import pandas as pd
from sklearn.preprocessing import OneHotEncoder
categorical_cols = [
 'sex',
 'on_thyroxine',
 'query_on_thyroxine',
 'on_antithyroid_meds',
 'sick',
 'pregnant',
 'thyroid_surgery',
 'I131_treatment',
 'query_hypothyroid',
 'query_hyperthyroid',
 'lithium',
 'goitre',
 'tumor',
 'hypopituitary',
 'psych',
 'TSH',
 'T3',
 'TT4',
 'T4U',
 'FTI',
 'TBG']
encoder = OneHotEncoder(sparse=False, drop='first')  # 'drop' parameter removes one of the one-hot encoded columns to avoid multicollinearity
encoded_cols = pd.DataFrame(encoder.fit_transform(data[categorical_cols]), columns=encoder.get_feature_names_out(categorical_cols))
data = pd.concat([data, encoded_cols], axis=1)
data.drop(categorical_cols, axis=1, inplace=True)
```

Fig-5.1.7: Converting the strings into Numerical using One Hot Encoding

```
from sklearn.impute import SimpleImputer

imputer = SimpleImputer(strategy='most_frequent')
data = pd.DataFrame(imputer.fit_transform(data), columns=data.columns)
```

Fig-5.1.8: Imputing missing values

```
from sklearn.preprocessing import LabelEncoder

# Assuming 'df' is your DataFrame and 'target_column' is the name of your target variable column
target_column = 'target'

# Initialize LabelEncoder
label_encoder = LabelEncoder()

# Fit and transform the target variable
data[target_column] = label_encoder.fit_transform(data[target_column])
```

Fig-5.1.9: transforming the values from categorical labels to numerical values

```
data#checking
```

	age	target	sex_M	on_thyroxine_t	query_on_thyroxine_t	on_antithyroid_meds_t	sick_t	pregnant_t	thyroid_surgery_t	I131_treatment_t	...	TBG_100.0	TBG_106.0	TBG_108.0
0	29.0	2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0
1	29.0	2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0
2	41.0	2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0
3	36.0	2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0
4	32.0	2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
9167	56.0	2	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0
9168	22.0	2	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0
9169	69.0	2	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0
9170	47.0	2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0
9171	31.0	2	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0

9172 rows × 1322 columns

Fig-5.1.10: Dataset after the Normalisation

Step 5: Data Visualization

```
# Line plot
plt.plot(data['TSH'],color="#533153")
plt.xlabel("TSH")
plt.ylabel("Levels")
plt.title("Line Plot")
plt.show()
```

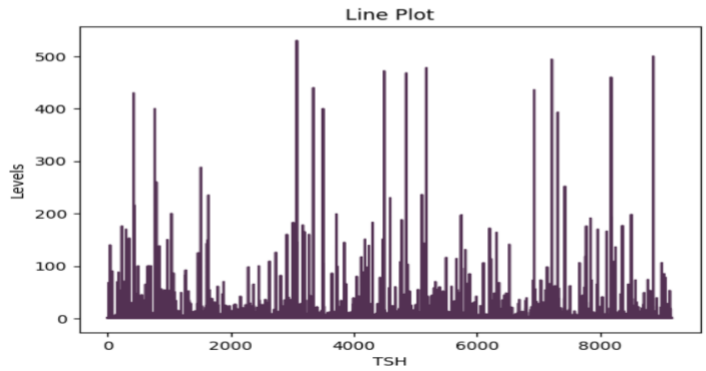


Fig-5.1.11: The line plot is a way to visualize the variation of 'TSH' levels

```
target_dist = pd.DataFrame(data['target'].value_counts())
target_dist.reset_index(inplace=True)
target_dist.columns = ['target', 'count']
fig = px.pie(target_dist, names='target', values='count')
fig.update_traces(marker=dict(colors=["#BEE9E9", "#6C7B8B", "#3A506B"]))
fig.show()
```

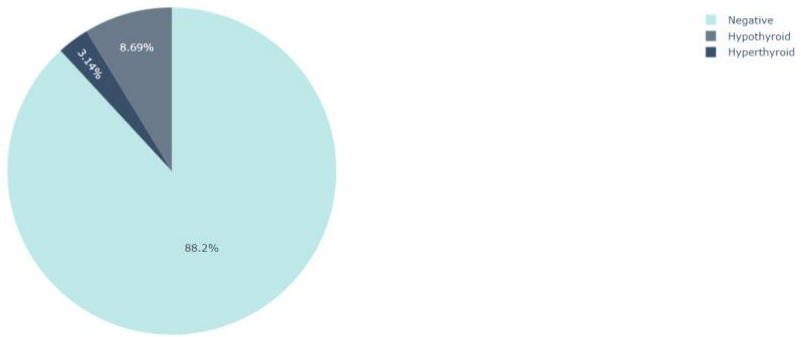


Fig-5.1.12: pie chart to visualize the distribution of values in the 'target' column

Step 6: Train-Test Split

```
x=data
y=data['target']
from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(x,y, test_size=0.33, random_state=89)
```

Fig-5.1.13: split the Data Frame and 'target' into training and testing sets

```python
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score, accuracy_score
from sklearn.model_selection import train_test_split
import numpy as np

# Assuming X_train, X_test, Y_train, Y_test are your training and testing data

# Split the data into training and testing sets
X_train, X_test, Y_train, Y_test = train_test_split(x, y, test_size=0.33, random_state=42)

# Create a linear regression model
model = LinearRegression()

# Fit the model to the training data
model.fit(X_train, Y_train)

# Make predictions on the test set
predictions = model.predict(X_test)

# Convert predictions to discrete classes (assuming a classification scenario)
predictions_classes = np.round(predictions).astype(int)

# Evaluate the model using accuracy (not typical for linear regression)
accuracy = accuracy_score(Y_test, predictions_classes)

print('Accuracy of Linear Regression model:', accuracy)
```

Fig-5.1.14: Linear Regression

```python
from sklearn import metrics
from sklearn.metrics import classification_report
from sklearn.preprocessing import LabelEncoder
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix,ConfusionMatrixDisplay
# Initialize and train the Logistic Regression model
model = LogisticRegression()
model.fit(x_train, y_train)

# Make predictions on the test set
predictions = model.predict(x_test)

# Convert back to original labels for evaluation
predictions_original_labels = label_encoder.inverse_transform(predictions)

# Evaluate the model
accuracy = metrics.accuracy_score(y_test, predictions)
print('The accuracy of the Logistic Regression model is:', accuracy)

# Display the classification report
report = classification_report(y_test, predictions)
print("Classification Report:\n", report)

cm = confusion_matrix(y_test, predictions, labels=model.classes_)
cmap = plt.cm.get_cmap('PuBu')
disp = ConfusionMatrixDisplay(confusion_matrix=cm,display_labels=model.classes_)
disp.plot(cmap =cmap)
plt.show()
```

Fig-5.1.15: Logistic Regression

```python
from sklearn import metrics
from sklearn.metrics import classification_report
from sklearn.preprocessing import LabelEncoder
from sklearn.svm import SVC

# Initialize and train the Logistic Regression model
model = SVC()
model.fit(x_train, y_train)

# Make predictions on the test set
predictions = model.predict(x_test)

# Convert back to original labels for evaluation
predictions_original_labels = label_encoder.inverse_transform(predictions)

# Evaluate the model
accuracy = metrics.accuracy_score(y_test, predictions)
print('The accuracy of the svm model is:', accuracy)

# Display the classification report
report = classification_report(y_test, predictions)
print("Classification Report:\n", report)
```

Fig-5.1.16: SVM Model

5.2 Algorithm Used

Linear Regression

Linear regression is a statistical method used to model the relationship between a dependent variable and one or more independent variables by fitting a linear equation to observed data. The simplest form is simple linear regression, which deals with the relationship between two variables, while multiple linear regression deals with two or more predictors.

Logistic regression

Logistic regression is a statistical method and a type of regression analysis used for predicting the probability of a binary outcome (1 / 0, Yes / No, True / False) based on one or more predictor variables. It's particularly useful when the dependent variable is categorical, and it's commonly employed for classification tasks in machine learning.

SVM Model

Support Vector Machines (SVM) are a type of supervised machine learning algorithm that can be used for classification or regression tasks. incorporating SVM into your thyroid detection project alongside linear and logistic regression can enhance the model's ability to handle non-linear relationships, high-dimensional data, and outliers. It provides a complementary approach that may lead to better overall performance, especially in scenarios where a more complex decision boundary is required.

6. Results and Analysis:

6.1 Results

The evaluation of machine learning model to predict Thyroid Detection is based on performance metrics such as accuracy, precision, recall, f1-score of the logistic regression. Their values for our model are 1.0, 1.0, 1.0, 1.0 respectively.

```
The accuracy of the Logistic Regression model is: 1.0
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        78
           1       1.00      1.00      1.00       209
           2       1.00      1.00      1.00      2740

    accuracy                           1.00      3027
   macro avg       1.00      1.00      1.00      3027
weighted avg       1.00      1.00      1.00      3027
```

Fig-6.1.1: Accuracy of Logistic Regression

```
Accuracy of Linear Regression model: 1.0
```

Fig-6.1.2: Accuracy of Linear Regression

```
The accuracy of the svm model is: 0.905186653452263
Classification Report:
              precision    recall  f1-score   support

           0       0.00      0.00      0.00        78
           1       0.00      0.00      0.00       209
           2       0.91      1.00      0.95      2740

    accuracy                           0.91      3027
   macro avg       0.30      0.33      0.32      3027
weighted avg       0.82      0.91      0.86      3027
```

Fig-6.1.3: Accuracy of SVM Model

6.2 Analysis

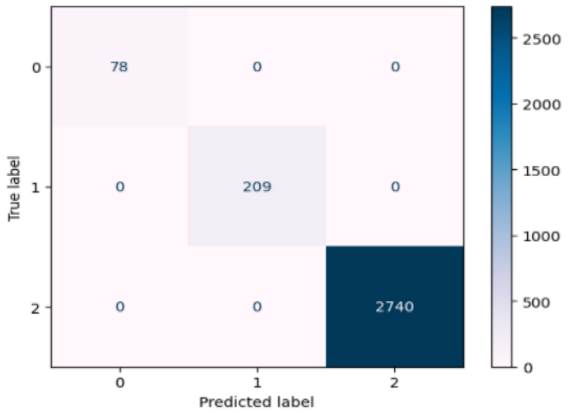


Fig-6.2.1: Confusion Matrix

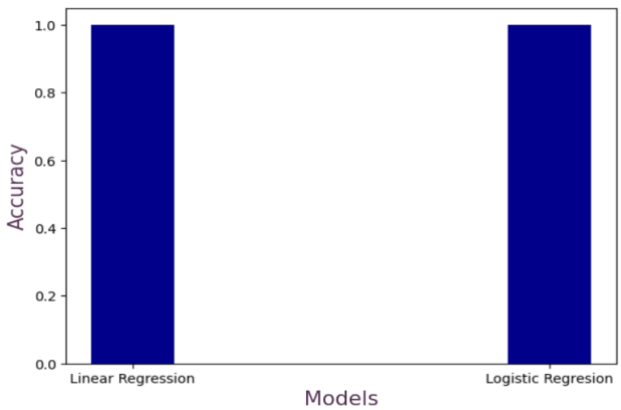


Fig-6.2.2: Comparison between Linear and Logistic Regression

6.3 Performance Evaluation metrics

When evaluating a machine learning model for detecting Thyroid we typically use various performance metrics to assess its effectiveness. Below are some common performance metrics for Thyroid detection:

1.Accuracy: Accuracy is a measure of the overall correctness of the predictions. It calculates the ratio of correctly predicted instances to the total number of instances. However, accuracy might not be the best metric if the data is imbalanced.

2. Precision: Precision is the ratio of true positive predictions to the total number of positive predictions made. It helps in detecting the presence or absence of thyroid abnormalities.

3.Recall (Sensitivity or True Positive Rate): Recall is the ratio of true positive predictions to the sum of true positives and false positives. precision in a thyroid detection project evaluates the model's ability to accurately identify positive cases (presence of a thyroid condition) among the instances it predicts as positive.

4.F1-Score: The F1-Score is the harmonic mean of precision and recall. It provides a balance between precision and recall. It is especially useful when you want to find an optimal balance between false positives and false negatives.

5.Specificity (True Negative Rate): Specificity in thyroid detection evaluates the model's ability to accurately identify cases without a thyroid condition, contributing to the overall assessment of the model's performance in distinguishing between positive and negative instances.

6.Area Under the ROC Curve (AUC-ROC): In thyroid detection, the ROC (Receiver Operating Characteristic) curve is a graphical representation of the trade-off between the true positive rate (sensitivity) and the false positive rate at different classification thresholds. The Area Under the ROC Curve (AUC-ROC) is a metric that quantifies the model's ability to distinguish between positive (presence of a thyroid condition) and negative (absence of a thyroid condition) cases across these various thresholds.

7.Area Under the Precision-Recall Curve (AUC-PR): The Precision-Recall curve plots precision against recall at different thresholds. AUC-PR quantifies the precision-recall trade-off.

8.Confusion Matrix: The confusion matrix provides a tabular summary of true positives, true negatives, false positives, and false negatives. It's helpful for a detailed understanding of model performance.

9.False Positive Rate (FPR): In thyroid detection, the False Positive Rate (FPR) is the ratio of false positive predictions to the total number of actual cases without a thyroid condition. It provides insights into the model's tendency to incorrectly predict the presence of a thyroid condition when it is not actually present.

10.True Negative Rate (TNR): TNR is another term for specificity and measures the model's ability to correctly identify Thyroid cases.

- Comparison with different algorithms

Algorithms used are Logistic Regression and Linear Regression

Logistic regression is a commonly used algorithm for binary classification problems, making it suitable for predicting the presence or absence of Thyroid. An accuracy of 1.00 indicates that the logistic regression model correctly classified 100% of the instances in the dataset. This suggests that the features used in the logistic regression model have a relatively strong relationship with the target variable (presence or absence of Thyroid).The model demonstrates a good ability to distinguish between positive and negative cases, making it a promising tool for Thyroid Detection.

In Linear regression we got the accuracy of 1.00 indicates that the linear regression model correctly classified 100% of the instances in the dataset. Achieving perfect accuracy suggests that the model may have memorized the training data, leading to doubts about its real-world applicability. It's crucial to examine other evaluation metrics, such as precision, recall, and the confusion matrix, to gain a more comprehensive understanding of the model's performance. Furthermore, the appropriateness of linear regression for binary classification tasks should be carefully considered, as logistic regression is more conventionally suited for such scenarios.

Thorough investigation into potential overfitting, data anomalies, and alignment with task requirements is necessary to ensure the reliability of the model's results.

7.Conclusion:

- Summary of Findings

The Thyroid Detection project, implemented in Python, has successfully utilized various data analysis techniques to understand Thyroid Detection better. The project has processed and analysed a vast amount of data related to Thyroid functions, leading to several significant findings about Thyroid disease.

- Achievements

The project has created a tool using Python that can understand and make sense of Thyroid data. This tool has been very useful in finding patterns and trends in the data that we didn’t know about before. Additionally, through this project, we have gained valuable experience and knowledge in analysing medical data using Python.

- Future Work

The next steps for this project include refining the analysis tool to incorporate more advanced machine learning algorithms for better predictive capabilities. Additionally, the team plans to extend the project to analyse other organ functions using similar methodologies. The ultimate goal is to create a comprehensive health analysis platform that can provide valuable insights into various aspects of human health.