Date	21-11-2023
Project Name	Endocrine Elegance: Classifying Thyroid Disorders with Precision
Team Id	591830

Endocrine Elegance: Classifying Thyroid Disorders with Precision

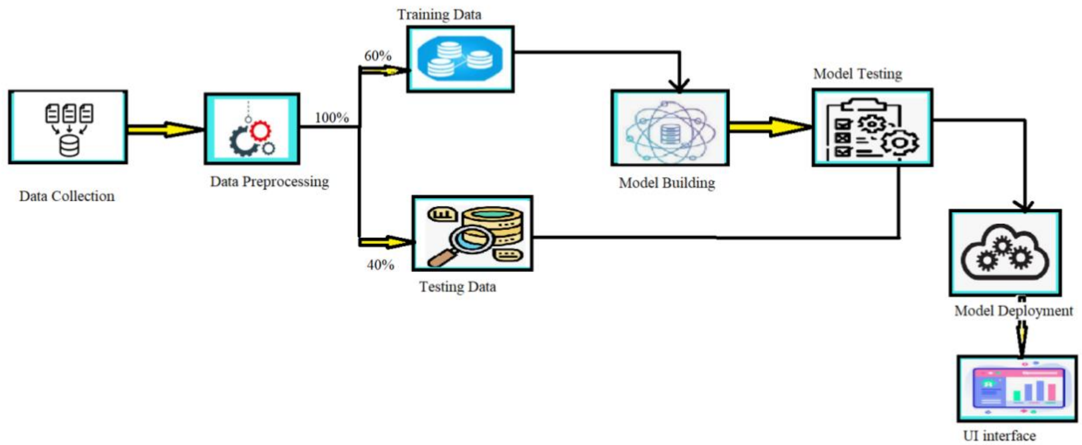
Introduction:

Endocrine Elegance is a web app using advanced machine learning models to accurately classify thyroid disorders.It leverages Random Forest, SVC, XGBoost, and ANN algorithms for precise diagnoses.Thyroid disorders impact well-being, but diagnosing them accurately can be challenging.

The app learns patterns from a large dataset of thyroid disorder cases.The user-friendly interface, built with Flask ML, allows easy access for healthcare professionals.Users input patient data, including symptoms and test results, for analysis.The models analyze the data and provide high-precision thyroid disorder classification.Endocrine Elegance reduces subjectivity and enhances diagnostic accuracy.It can be accessed remotely, enabling informed decisions regardless of location.

This app has the potential to revolutionize endocrinology and improve patient care worldwide.

Technical Architecture:



Prerequisites:

Anaconda Navigator is a free and open-source distribution of the Python and R programming languages for data science and machine learning related applications. It can be installed on Windows, Linux, and macOS. Conda is an open-source, cross-platform, package management system. Anaconda comes with so very nice tools like JupyterLab, Jupyter Notebook, QtConsole, Spyder, Glueviz, Orange, Rstudio, Visual Studio Code.

For this project, we will be using Jupyter notebook and Spyder.

1. To build Machine learning models you must require the following packages

- Numpy:

It is an open-source numerical Python library. It contains a multidimensional array and matrix data structures and can be used to perform mathematical operations

- Scikit-learn:

It is a free machine learning library for Python. It features various algorithms like support vector machine, random forests, and k-neighbors, and it also supports Python numerical and scientific libraries like NumPy and SciPy

- Flask:

Web framework used for building Web applications

- Python packages:

Open anaconda prompt as administrator

Type “pip install numpy” and click enter.

Type “pip install pandas” and click enter.

Type “pip install scikit-learn” and click enter.

Type “pip install tensorflow==2.3.2” and click enter.

Type “pip install keras==2.3.1” and click enter.

Type “pip install Flask” and click enter.

- Machine Learning Concepts

✓ Machine Learning Models: Utilize advanced machine learning algorithms like Random Forest, Support Vector Machines (SVM), XGBoost, or Artificial Neural Networks (ANN) to analyze data and make predictions for the web app.

✓ Training and Testing: Train the machine learning models using labeled datasets to learn patterns and relationships, and evaluate their performance through testing to ensure accuracy and reliability.

✓ Feature Engineering: Extract and select relevant features from the input data to improve the performance and efficiency of the machine learning models in making predictions.

✓ Model Deployment: Deploy the trained machine learning models within the web app framework, such as Flask ML, to provide real-time predictions and functionality to users.

Project Objectives:

By the end of this project you will:

✓ Develop a robust machine learning model to accurately classify thyroid disorders based on diverse patient data.

✓ Enhance precision in diagnosis by integrating advanced algorithms that analyze hormonal, imaging, and clinical parameters.

✓ Implement a user-friendly interface for healthcare professionals to input patient data and receive reliable predictions for thyroid conditions.

✓ Strive for high sensitivity and specificity to ensure Endocrine Elegance's efficacy in supporting clinicians with precise thyroid disorder classifications.

Project Flow:

- ✓ **Data Collection and Preprocessing:** Gather diverse datasets encompassing hormonal, imaging, and clinical information. Employ thorough preprocessing techniques to ensure data quality and uniformity.

- ✓ **Model Development:** Design and train a sophisticated machine learning model, leveraging advanced algorithms to analyze and interpret the integrated data for precise classification of thyroid disorders.

- ✓ **Interface Implementation:** Develop an intuitive user interface, enabling healthcare professionals to input patient data seamlessly. Ensure real-time interaction with the trained model and provide clear diagnostic outputs.

- ✓ **Validation and Optimization:** Rigorously validate the model's performance using independent datasets. Fine-tune algorithms and parameters for optimal precision, sensitivity, and specificity, ensuring Endocrine Elegance's reliability in clinical settings.

To accomplish this, we have to complete all the activities and tasks listed below

Data Collection.

1. Download the dataset
2. Importing the libraries
3. Read the Dataset

Data Preprocessing.

1. Checking for null values
2. Splitting the data x and y
3. Converting the Data Type
4. Handling Categorical Values
5. Splitting data into train and test
6. Handling Imbalanced Data
7. Applying StandardScaler
8. Performing Feature Importance
9. Selecting Output Columns

Exploratory Data Analysis

1. Descriptive statistical
2. Visual Analysis

Model Building

1. Training the model in multiple algorithms
2. Testing the model

Performance Testing & Hyperparameter Tuning

1. Testing model with multiple evaluation metrics
2. Comparing model accuracy before & after applying hyperparameter tuning

Model Deployment

1. Save the best model
2. Integrate with Web Framework

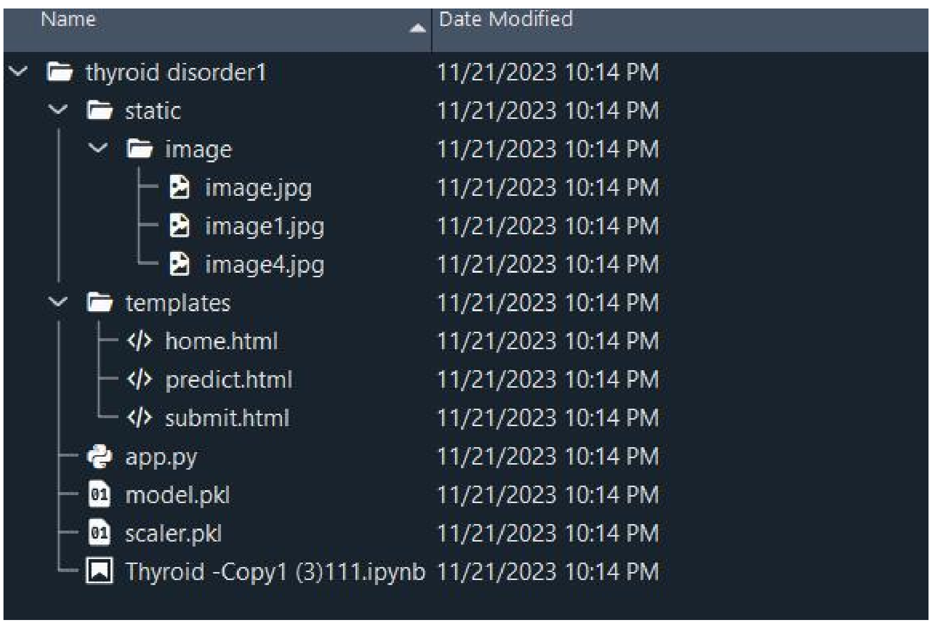
Project Demonstration & Documentation

1. Record explanation Video for project end to end solution
2. Project Documentation-Step by step project development procedure

Project Structure:

Create a Project folder which contains files as shown below

We use spyder platform to use this project structure



- We are building a Flask Application that needs HTML pages stored in the templates folder and a python script app.py for server side scripting

- we need the model which is saved and the saved model in this content is a model1.pkl

- templates folder contains home.html,predict.html ,submit.html pages.

Milestone 1: Data Collection

There are many popular open sources for collecting the data. Eg: kaggle.com,UCI repository, etc.

In this project, we have used thyroid data. This data is downloaded from kaggle.com. Please refer to the link given below to download the dataset.

Link: https://www.kaggle.com/datasets/emmanuelfwerr/thyroid-disease-data

1.1 Importing the libraries

Import the necessary libraries as shown in the image.

Importing the libraries

In [2]:

```
# Importing the required libraries
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score
```

1.2 Read the Dataset

Our dataset format might be in .csv, excel files, .txt, .json, etc. We can read the dataset with the help of pandas.

In pandas, we have a function called read_csv() to read the dataset. As a parameter, we have to give the directory of the csv file.

In [3]:
```
# Loading the dataset
data = pd.read_csv('thyroidDF.csv')
```

In [4]:
```
data
```

Out[4]:

	age	sex	on_thyroxine	query_on_thyroxine	on_antithyroid_meds	sick	pregnant	thyroi
0	29	F	f	f	f	f	f	
1	29	F	f	f	f	f	f	
2	41	F	f	f	f	f	f	
3	36	F	f	f	f	f	f	
4	32	F	f	f	f	f	f	
...	...	...	...	...	...	...	...	
9167	56	M	f	f	f	f	f	
9168	22	M	f	f	f	f	f	
9169	69	M	f	f	f	f	f	
9170	47	F	f	f	f	f	f	
9171	31	M	f	f	f	f	f	

9172 rows × 31 columns

In [54]:
```
data.shape
```

Out[54]: (9172, 23)

Milestone 2: Data Preprocessing

As we have understood how the data is, let's pre-process the collected data.

The download data set is not suitable for training the machine learning model as it might have so much randomness so we need to clean the dataset properly in order to fetch good results. This activity includes the following steps.

• Handling missing values

• Descriptive analysis

• Splitting the dataset as x and y

• Handling Categorical Values

• Checking Correlation

• Converting Data Type

• Splitting dataset into training and test set

• Handled Imbalanced Data

• Applying Standard Scaler

2.1: Checking for null values

Removing handling values

```
data["sex"].fillna(data["sex"].mode()[0],inplace=True)
data["TSH"].fillna(data["TSH"].mean(),inplace=True)
data["T3"].fillna(data["T3"].mean(),inplace=True)
data["TT4"].fillna(data["TT4"].mean(),inplace=True)
data["T4U"].fillna(data["T4U"].mean(),inplace=True)
data["FTI"].fillna(data["FTI"].mean(),inplace=True)
data["TBG"].fillna(data["TBG"].mean(),inplace=True)
```

```
data.isnull().sum()
```

```
]:  age                      0
    sex                      0
    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                      0
    T3                       0
    TT4                      0
    T4U                      0
    FTI                      0
    TBG                      0
    target                   0
    dtype: int64
```

2.2: Splitting the data x and y

Splitting the data x and y

2.2 Splitting the data X and Y

```
x=data.iloc[:,0:-1] #independent Columns
y=data.iloc[:,-1] #dependent Columns
```

```
x.head()
```

8]:

	age	sex	on_thyroxine	query_on_thyroxine	on_antithyroid_meds	sick	pregnant	thyroid_s
0	29	F	f	f	f	f	f	
1	29	F	f	f	f	f	f	
2	41	F	f	f	f	f	f	
3	36	F	f	f	f	f	f	
4	32	F	f	f	f	f	f	

5 rows × 22 columns

```
y.head()
```

9]:

```
0    -
1    -
2    -
3    -
4    S
Name: target, dtype: object
```

2.3: Converting the Data Type

```
x.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9172 entries, 0 to 9171
Data columns (total 22 columns):
 #   Column               Non-Null Count  Dtype  
---  ------               --------------  -----  
 0   age                  9172 non-null   int64  
 1   sex                  9172 non-null   float64
 2   on_thyroxine         9172 non-null   float64
 3   query_on_thyroxine   9172 non-null   float64
 4   on_antithyroid_meds  9172 non-null   float64
 5   sick                 9172 non-null   float64
 6   pregnant             9172 non-null   float64
 7   thyroid_surgery      9172 non-null   float64
 8   I131_treatment       9172 non-null   float64
 9   query_hypothyroid    9172 non-null   float64
 10  query_hyperthyroid   9172 non-null   float64
 11  lithium              9172 non-null   float64
 12  goitre               9172 non-null   float64
 13  tumor                9172 non-null   float64
 14  hypopituitary        9172 non-null   float64
 15  psych                9172 non-null   float64
 16  TSH                  9172 non-null   float64
 17  T3                   9172 non-null   float64
 18  TT4                  9172 non-null   float64
 19  T4U                  9172 non-null   float64
 20  FTI                  9172 non-null   float64
 21  TBG                  9172 non-null   float64
dtypes: float64(21), int64(1)
memory usage: 1.5 MB
```

2.4: Handling Categorical Values

As we can see our dataset has categorical data we must convert the categorical data to integer encoding or binary encoding.

To convert the categorical features into numerical features we use encoding techniques. There are several techniques but in our project we are using Ordinal Encoding and Label Encoding.

- In our project, categorical features are x and y values.
- Here, applying Ordinal Encoding on x values.

2.4 Handling Categorical Values

```
from sklearn.preprocessing import OrdinalEncoder

# Assuming X is your DataFrame
categorical_columns = x.select_dtypes(include=['object']).columns

# Initialize OrdinalEncoder
ordinal_encoder = OrdinalEncoder()

# Fit and transform the categorical columns
x[categorical_columns] = ordinal_encoder.fit_transform(x[categorical_columns]

# Now X contains the ordinal-encoded values for categorical columns
```

```
x.head()
```

23]:

	age	sex	on_thyroxine	query_on_thyroxine	on_antithyroid_meds	sick	pregnant	thyroid_s
0	29	0.0	0.0	0.0	0.0	0.0	0.0	
1	29	0.0	0.0	0.0	0.0	0.0	0.0	
2	41	0.0	0.0	0.0	0.0	0.0	0.0	
3	36	0.0	0.0	0.0	0.0	0.0	0.0	
4	32	0.0	0.0	0.0	0.0	0.0	0.0	

5 rows × 22 columns

- applying Label Encoding on y(Independent variable) value.

```
from sklearn.preprocessing import LabelEncoder

# Assuming y is your target variable
le = LabelEncoder()
y_encoded = le.fit_transform(y)

# Now, Y_encoded contains the numeric labels for your target variable
```

```
unique_values = y.unique()
print(unique_values)
```

```
['-' 'S' 'F' 'AK' 'R' 'I' 'M' 'N' 'G' 'K' 'A' 'KJ' 'L' 'MK' 'Q' 'J' 'C|I'
 'O' 'LJ' 'H|K' 'D' 'GK' 'MI' 'P' 'FK' 'B' 'GI' 'C' 'GKJ' 'OI' 'D|R' 'E']
```

```
# Mapping between original labels and encoded labels
label_mapping = dict(zip(le.classes_, range(len(le.classes_))))
print(label_mapping)
#encoded values
```

```
{'-': 0, 'A': 1, 'AK': 2, 'B': 3, 'C': 4, 'C|I': 5, 'D': 6, 'D|R': 7, 'E':
8, 'F': 9, 'FK': 10, 'G': 11, 'GI': 12, 'GK': 13, 'GKJ': 14, 'H|K': 15,
'I': 16, 'J': 17, 'K': 18, 'KJ': 19, 'L': 20, 'LJ': 21, 'M': 22, 'MI': 23,
'MK': 24, 'N': 25, 'O': 26, 'OI': 27, 'P': 28, 'Q': 29, 'R': 30, 'S': 31}
```

2.5: Splitting data into train and test

Now let's split the Dataset into train and test sets

Changes: first split the dataset into x and y and then split the data set

Here x and y variables are created. On x variable, data is passed with dropping the target variable. And my target variable is passed. For splitting training and testing data we are using the train_test_split() function from sklearn. As parameters, we are passing x, y, test_size, random_state.

```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test = train_test_split(x_Scaled,y,test_size =0.2,ra
```

```
print(x_train.shape,x_test.shape,y_train.shape,y_test.shape)
```

```
(7337, 22) (1835, 22) (7337,) (1835,)
```

2.6: Handling Imbalanced Data

2.5 Handling Imbalanced Data

```
# Assuming y_encoded is your label-encoded target variable
y = y_encoded.astype(float)
y
```

```
3]: array([ 0.,  0.,  0., ..., 16.,  0.,  0.])
```

2.7: Applying Standard Scaler

• Scaling the features makes the flow of gradient descent smooth and helps algorithms quickly reach the minima of the cost function.

• Without scaling features, the algorithm may be biased toward the feature which has values higher in magnitude. it brings every feature in the same range and the model uses every feature wisely.

• Here, we have the data in array format and we are making it dataframe.

Applying StandardScaler

```
from sklearn.preprocessing import MinMaxScaler
ms=MinMaxScaler()
```

```
x_Scaled=ms.fit_transform(x)
```

```
x_Scaled=pd.DataFrame(ms.fit_transform(x),columns=x.columns)
```

```
x_Scaled.head()
```

	age	sex	on_thyroxine	query_on_thyroxine	on_antithyroid_meds	sick	pregnant	thyr
0	0.000427	0.0	0.0	0.0	0.0	0.0	0.0	
1	0.000427	0.0	0.0	0.0	0.0	0.0	0.0	
2	0.000610	0.0	0.0	0.0	0.0	0.0	0.0	
3	0.000534	0.0	0.0	0.0	0.0	0.0	0.0	
4	0.000473	0.0	0.0	0.0	0.0	0.0	0.0	

5 rows × 22 columns

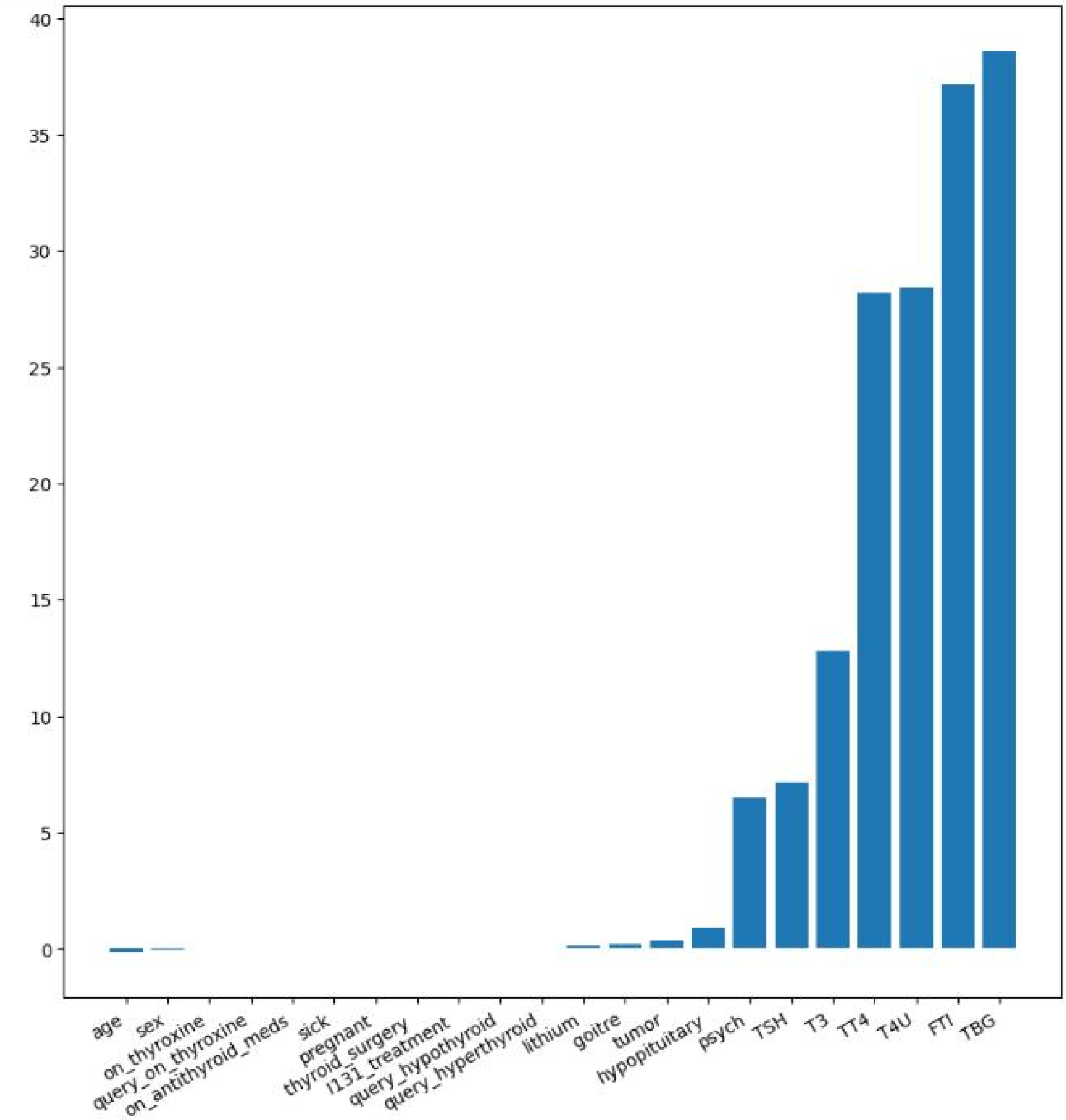
2.8: Performing Feature Importance

• The idea behind permutation feature importance is simple. The feature importance is calculated by noticing the increase or decrease in error when we permute the values of a feature.

• If permuting the values causes a huge change in the error, it means the feature is important for our model.

```python
# To display the importance scores
importance_scores = pd.Series(results.importances_mean, index=x_test.columns)
print("\nPermutation Feature Importance:")
print(importance_scores.sort_values(ascending=False))
```

```
Permutation Feature Importance:
T3                       38.571929
FTI                      37.182925
TT4                      28.414427
TSH                      28.192544
TBG                      12.822716
T4U                       7.179820
on_thyroxine              6.487885
age                       0.908594
on_antithyroid_meds       0.328944
sex                       0.217011
thyroid_surgery           0.151109
query_hyperthyroid        0.043944
psych                     0.042174
tumor                     0.041817
sick                      0.025249
I131_treatment            0.019408
goitre                    0.008491
pregnant                  0.006051
hypopituitary             0.000000
query_on_thyroxine       -0.001647
lithium                  -0.048012
query_hypothyroid        -0.164220
dtype: float64
```



2.9: Selecting Output Columns

Before we have this many columns

```
x.head()
```

	age	sex	on_thyroxine	query_on_thyroxine	on_antithyroid_meds	sick	pregnant	thyroid_s
0	29	0.0	0.0	0.0	0.0	0.0	0.0	
1	29	0.0	0.0	0.0	0.0	0.0	0.0	
2	41	0.0	0.0	0.0	0.0	0.0	0.0	
3	36	0.0	0.0	0.0	0.0	0.0	0.0	
4	32	0.0	0.0	0.0	0.0	0.0	0.0	

After Performing Feature Importance by using 'Permutation Importance' we are dropping some columns which are not important for 'target'.

```
x_test.head()
```

	age	sex	on_thyroxine	query_on_thyroxine	on_antithyroid_meds	sick	pregnant	thyroi
624	72	1.0	0.0	0.0	0.0	0.0	0.0	
6458	57	0.0	0.0	0.0	0.0	0.0	0.0	
3128	82	1.0	0.0	0.0	0.0	0.0	0.0	
5501	58	0.0	0.0	0.0	0.0	0.0	0.0	
9070	80	1.0	0.0	0.0	0.0	0.0	0.0	

5 rows × 22 columns

```
x_bal.head()
```

	goitre	tumor	hypopituitary	psych	TSH	T3	TT4	T4U	FTI	
0	0.0	0.0	0.0	0.0	0.000557	0.106999	0.178429	0.373174	0.127604	0.148
1	0.0	0.0	0.0	0.0	0.003009	0.103064	0.210702	0.373174	0.127604	0.148
2	0.0	0.0	0.0	0.0	0.009837	0.106999	0.178429	0.373174	0.127604	0.054
3	0.0	0.0	0.0	0.0	0.009837	0.106999	0.178429	0.373174	0.127604	0.129
4	0.0	0.0	0.0	0.0	0.009837	0.106999	0.178429	0.373174	0.127604	0.179

Milestone 3: Exploratory Data Analysis

3.1: Descriptive analysis

Descriptive analysis is to study the basic features of data with the statistical process. Here pandas have a worthy function called describe. With this described function we can find mean, std, min, max and percentile values of continuous features.

3.1 Descriptive analysis

```
data.describe()
```

]:

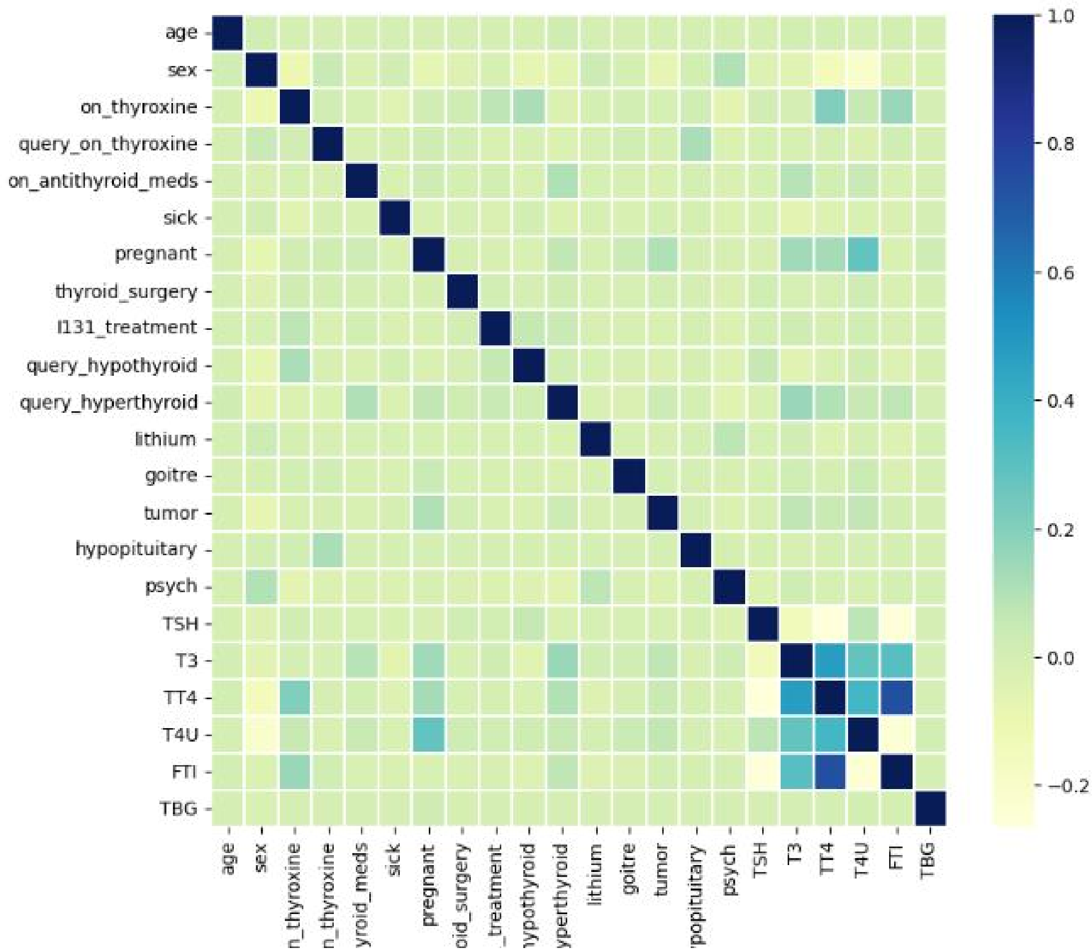
	age	TSH	T3	TT4	T4U	FTI	
count	9172.000000	9172.000000	9172.000000	9172.000000	9172.000000	9172.000000	9172.0
mean	73.555822	5.218403	1.970629	108.700305	0.976056	113.640746	29.8
std	1183.976718	23.047102	0.751073	36.607295	0.191319	39.693254	4.1
min	1.000000	0.005000	0.050000	2.000000	0.170000	1.400000	0.1
25%	37.000000	0.590000	1.700000	88.000000	0.870000	95.000000	29.8
50%	55.000000	1.600000	1.970629	106.000000	0.976056	112.000000	29.8
75%	68.000000	3.700000	2.200000	124.000000	1.050000	126.000000	29.8
max	65526.000000	530.000000	18.000000	600.000000	2.330000	881.000000	200.0

3.2: Visual analysis

Visual analysis is the process of using visual representations, such as charts, plots, and graphs, to explore and understand data. It is a way to quickly identify patterns, trends, and outliers in the data, which can help to gain insights and make informed decisions.

3.2 Visual analysis

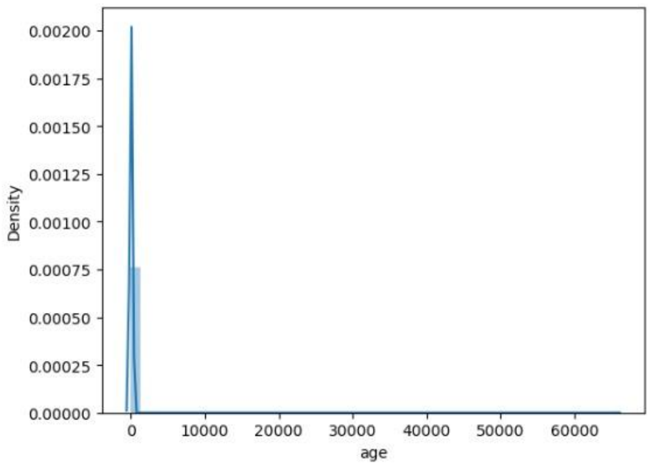
```python
import seaborn as sns
import matplotlib.pyplot as plt  # Importing the necessary module

corrmat = x.corr()
f, ax = plt.subplots(figsize=(9,8))  # Corrected the function name to subplot
sns.heatmap(corrmat, ax=ax, cmap="YlGnBu", linewidths=0.1)
plt.show()
```



In [17]:
```
import seaborn as sns
sns.distplot(data["age"])
```

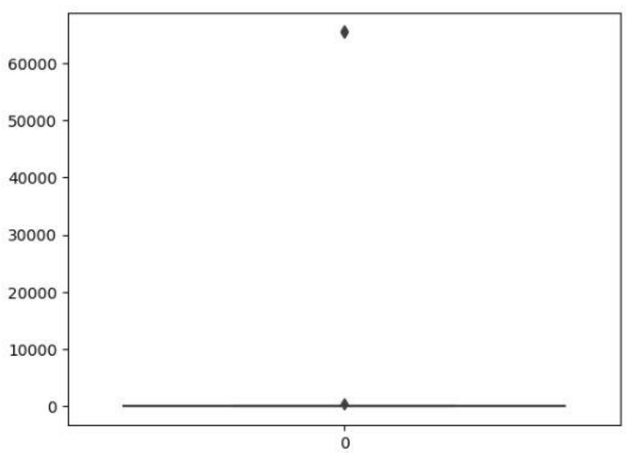
```
C:\Users\ASUS\AppData\Local\Temp\ipykernel_2132\3892360786.py:2: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(data["age"])
```

Out[17]: <Axes: xlabel='age', ylabel='Density'>



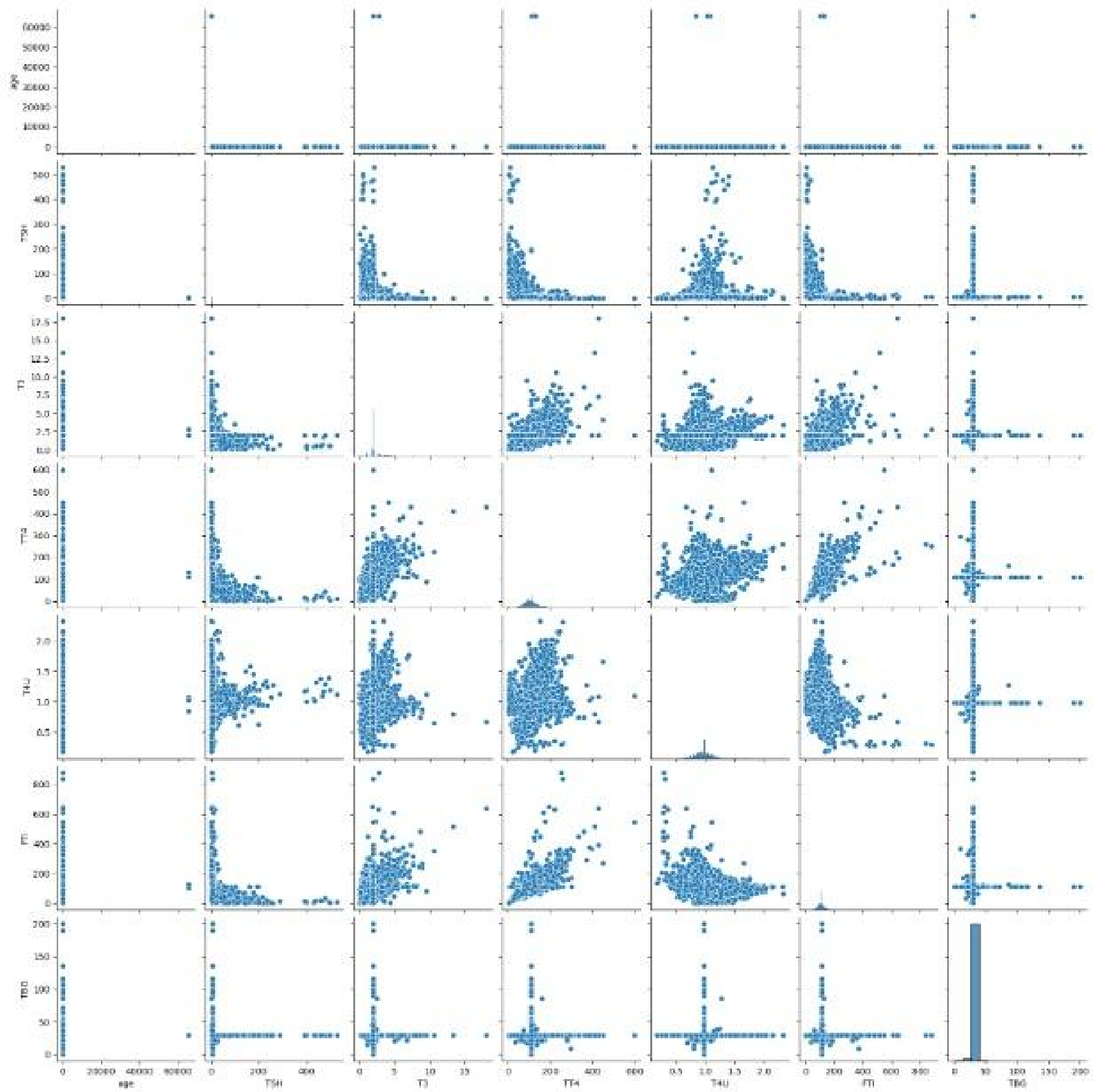
In [19]:
```
sns.boxplot(data.age)
```

Out[19]: <Axes: >

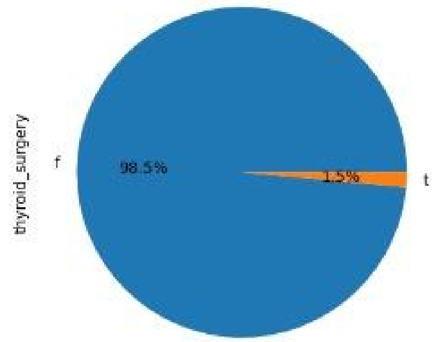


In [21]:
```
sns.pairplot(data)
```

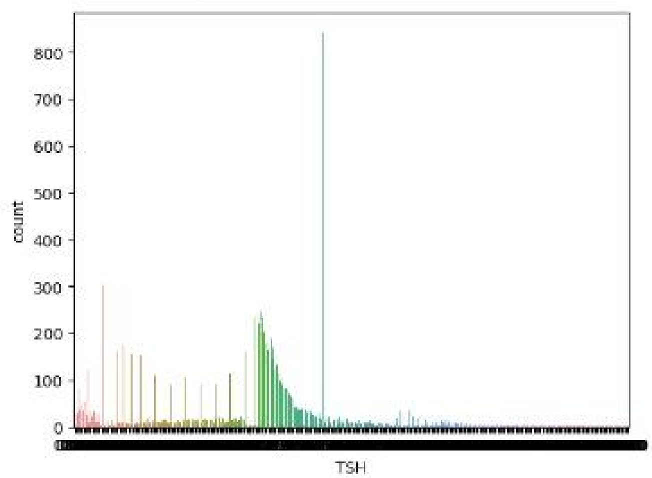
Out[21]: <seaborn.axisgrid.PairGrid at 0x13c9fa90f50>



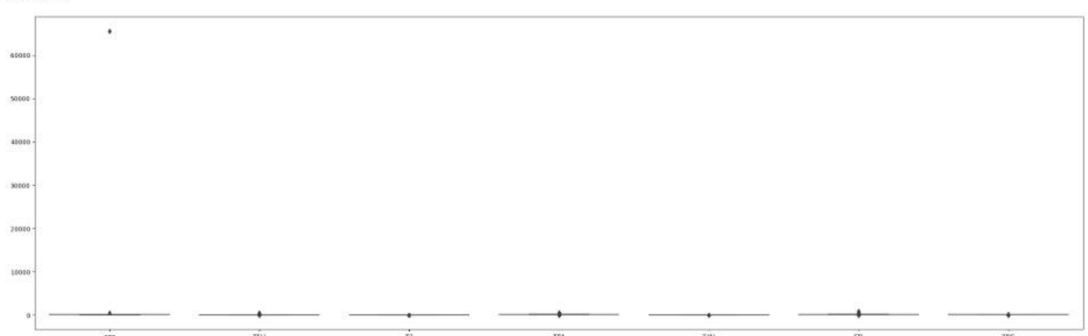
In [22]:
```
data.thyroid_surgery.value_counts().plot(kind="pie",autopct="%1.1f%%")
```
Out[22]: <Axes: ylabel='thyroid_surgery'>



In [23]:
```
sns.countplot(x="TSH",data=data)
```
Out[23]: <Axes: xlabel='TSH', ylabel='count'>



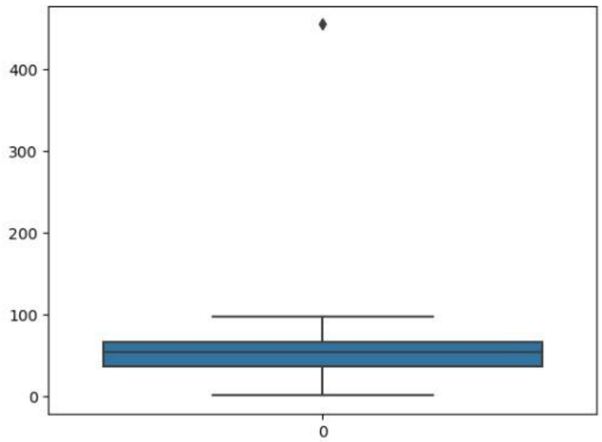
In [25]:
```
import matplotlib.pyplot as plt
plt.figure(figsize=(30,9))
sns.boxplot(data)
```
Out[25]: <Axes: >



```
In [26]: from scipy import stats
         import numpy as np
         z_scores = stats.zscore(data['age'])
         df_cleaned = data[(np.abs(z_scores) <=3)]
```

```
In [27]: sns.boxplot(df_cleaned['age'])
```

```
Out[27]: <Axes: >
```

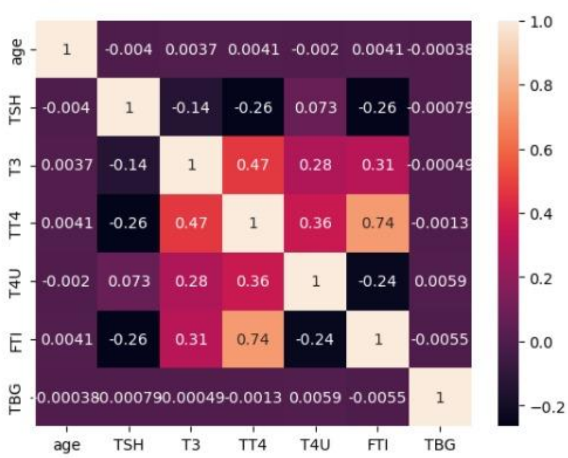


2.1: Checking Correlation.

Here, I'm finding the correlation using HeatMap. It visualizes the data in 2-D coloured maps making use of colour variations. It describes the related variables in the form of colours instead of numbers; it will be plotted on both axes.

Here, there is no correlation between columns.

```
In [18]: sns.heatmap(data.corr(),annot=True)

C:\Users\ASUS\AppData\Local\Temp\ipykernel_2132\2578434383.py:1: FutureWarning: The default value of numeric_only in DataFrame.
corr is deprecated. In a future version, it will default to False. Select only valid columns or specify the value of numeric_on
ly to silence this warning.
  sns.heatmap(data.corr(),annot=True)

Out[18]: <Axes: >
```



Milestone 4: Model Building

4.1: Random Forest Classifier Model

A function named Random Forest Classifier Model is created and train and test data are passed as the parameters. Inside the function, the Random Forest Classifier algorithm is initialized and training data is passed to the model with the .fit() function. Test data is predicted with the .predict() function and saved in a new variable. For evaluating the model, accuracy_score and classification report is done.

1.1: Random Forest Classifier Model

```
In [56]: from sklearn.ensemble import RandomForestClassifier
         rfc=RandomForestClassifier()

In [57]: rfc.fit(x_train,y_train)

Out[57]: ▾ RandomForestClassifier
         RandomForestClassifier()

In [58]: y_pred=rfc.predict(x_test)

In [59]: y_pred

Out[59]: array([ 0.,  0., 18., ...,  0., 18.,  0.])

In [60]: y_test

Out[60]: array([ 0.,  0., 18., ...,  0., 18.,  0.])

In [61]: TBG=pd.DataFrame({"Actual_TBG":y_test,"Predicted _TBG":y_pred})

In [62]: TBG

Out[62]:
```

	Actual_TBG	Predicted _TBG
0	0.0	0.0
1	0.0	0.0
2	18.0	18.0
3	0.0	0.0
4	18.0	18.0
...	...	...
1830	0.0	0.0
1831	0.0	0.0
1832	0.0	0.0
1833	18.0	18.0
1834	0.0	0.0

1835 rows × 2 columns

```
In [66]: # Import necessary libraries
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report

# Assume 'X' is your feature matrix, and 'y' is your target variable (labels)
# Replace this with your actual data
# X, y = ...

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42)

# Create a Random Forest Classifier model
rf_classifier = RandomForestClassifier(n_estimators=100, random_state=42)

# Train the model
rf_classifier.fit(X_train, y_train)

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

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.2f}")

# Display classification report
print("Classification Report:")
print(classification_report(y_test, y_pred))
```

```
Accuracy: 0.93
Classification Report:
              precision    recall  f1-score   support

         0.0       0.95      0.97      0.96      1328
         1.0       0.62      0.71      0.67        21
         2.0       0.78      0.70      0.74        10
         3.0       0.00      0.00      0.00         4
         9.0       0.90      0.95      0.93        40
        10.0       0.00      0.00      0.00         1
        11.0       0.97      1.00      0.99        69
        12.0       0.00      0.00      0.00         1
        13.0       0.86      1.00      0.92         6
        16.0       0.83      0.70      0.75        82
        17.0       1.00      0.50      0.67        12
        18.0       0.89      0.92      0.91       106
        19.0       1.00      0.50      0.67         2
        20.0       0.65      0.54      0.59        28
        22.0       1.00      1.00      1.00        25
        24.0       1.00      1.00      1.00         6
        25.0       0.68      0.85      0.76        20
        26.0       1.00      0.50      0.67         4
        29.0       0.67      0.67      0.67         3
        30.0       0.71      0.60      0.65        45
```

```
In [67]: from sklearn.model_selection import train_test_split
         from sklearn.ensemble import RandomForestClassifier
         from sklearn.metrics import accuracy_score, classification_report

         # Assume 'X' is your feature matrix, and 'y' is your target variable (labels)
         # Replace this with your actual data
         # X, y = ...

         # Split the data into training, validation, and testing sets
         X_train, X_temp, y_train, y_temp = train_test_split(x, y, test_size=0.3, random_state=42)
         X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5, random_state=42)

         # Create a Random Forest Classifier model with hyperparameter adjustments
         rf_classifier = RandomForestClassifier(n_estimators=100, max_depth=10, min_samples_split=5, min_samples_leaf=2, random_state=42)

         # Train the model on the training set
         rf_classifier.fit(X_train, y_train)

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

         # Evaluate the model on the test set
         accuracy_test = accuracy_score(y_test, y_test_pred)
         print(f"Testing Accuracy: {accuracy_test:.2f}")

         # Make predictions on the validation set
         y_val_pred = rf_classifier.predict(X_val)

         # Evaluate the model on the validation set
         accuracy_val = accuracy_score(y_val, y_val_pred)
         print(f"Validation Accuracy: {accuracy_val:.2f}")

         # Optionally, you can also print the training accuracy
         y_train_pred = rf_classifier.predict(X_train)
         accuracy_train = accuracy_score(y_train, y_train_pred)
         print(f"Training Accuracy: {accuracy_train:.2f}")

         # Display classification report for the test set
         print("Classification Report for Test Set:")
         print(classification_report(y_test, y_test_pred))

         Testing Accuracy: 0.92
         Validation Accuracy: 0.90
         Training Accuracy: 0.94
```

4.2: XGBClassifier model

A function named XGBClassifier model is created and train and test data are passed as the parameters. Inside the function, the XGBClassifier algorithm is initialized and training data is passed to the model with the .fit() function. Test data is predicted with the .predict() function and saved in a new variable. For evaluating the model, the accuracy score and classification report is done.

In [68]:
```python
# Import necessary libraries
from xgboost import XGBClassifier

# Create an XGBClassifier model
xgb_classifier = XGBClassifier(random_state=42)

# Train the model
xgb_classifier.fit(X_train, y_train)

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

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.2f}")

# Display classification report
print("Classification Report:")
print(classification_report(y_test, y_pred))
```

```
Accuracy: 0.93
Classification Report:
              precision    recall  f1-score   support

         0.0       0.97      0.97      0.97      1014
         1.0       0.88      0.75      0.81        20
         2.0       1.00      0.67      0.80         6
         3.0       0.00      0.00      0.00         3
         4.0       0.00      0.00      0.00         1
         6.0       1.00      1.00      1.00         2
         9.0       0.95      0.92      0.93        38
        10.0       0.00      0.00      0.00         1
        11.0       0.86      0.96      0.91        52
        12.0       0.50      0.50      0.50         2
        13.0       0.88      1.00      0.93         7
        16.0       0.84      0.87      0.85        54
        17.0       0.60      0.50      0.55         6
        18.0       0.84      0.92      0.88        74
        19.0       1.00      0.50      0.67         2
        20.0       0.70      0.78      0.74        18
        22.0       0.88      1.00      0.93        14
        24.0       0.00      0.00      0.00         3
        25.0       0.71      0.71      0.71        14
        26.0       1.00      0.50      0.67         2
        28.0       0.00      0.00      0.00         1
        29.0       0.00      0.00      0.00         2
        30.0       0.83      0.74      0.78        27
        31.0       0.93      1.00      0.96        13

    accuracy                           0.93      1376
   macro avg       0.64      0.60      0.61      1376
weighted avg       0.93      0.93      0.93      1376
```

```python
In [69]: # Import necessary libraries
from xgboost import XGBClassifier
from sklearn.metrics import accuracy_score, classification_report
from sklearn.model_selection import train_test_split

# Assuming you have your data in X and y

# Split the data into training and testing sets
X_train, X_temp, y_train, y_temp = train_test_split(x, y, test_size=0.3, random_state=42)
X_test, X_val, y_test, y_val = train_test_split(X_temp, y_temp, test_size=0.5, random_state=42)

# Create an XGBClassifier model with some hyperparameter adjustments
xgb_classifier = XGBClassifier(
    learning_rate=0.01,
    max_depth=3,
    min_child_weight=1,
    gamma=0.1,
    subsample=0.8,
    colsample_bytree=0.8,
    reg_alpha=0.1,
    reg_lambda=0.1,
    random_state=42
)

# Train the model on the training set
xgb_classifier.fit(X_train, y_train)

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

# Evaluate the model on the test set
accuracy_test = accuracy_score(y_test, y_test_pred)
print(f"Testing Accuracy: {accuracy_test:.2f}")

# Make predictions on the validation set
y_val_pred = xgb_classifier.predict(X_val)

# Evaluate the model on the validation set
accuracy_val = accuracy_score(y_val, y_val_pred)
print(f"Validation Accuracy: {accuracy_val:.2f}")

# Print the training accuracy
y_train_pred = xgb_classifier.predict(X_train)
accuracy_train = accuracy_score(y_train, y_train_pred)
print(f"Training Accuracy: {accuracy_train:.2f}")

# Display classification report for the test set
print("Classification Report for Test Set:")
print(classification_report(y_test, y_test_pred))
```

```
Testing Accuracy: 0.89
Validation Accuracy: 0.91
Training Accuracy: 0.92
```

4.3: SVC model

A function named SVC model is created and train and test data are passed as the parameters. Inside the function, the SVC algorithm is initialized and training data is passed to the model with .fit() function. Test data is predicted with the .predict() function and saved in a new variable. For evaluating the model, the accuracy score and classification report is done.

In [70]:
```
# Import necessary libraries
from sklearn.svm import SVC

# Create an SVC model
svc_classifier = SVC(kernel='rbf', random_state=42)  # 'rbf' stands for radial basis function, a common choice

# Train the model
svc_classifier.fit(X_train, y_train)

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

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.2f}")

# Display classification report
print("Classification Report:")
print(classification_report(y_test, y_pred))
```

```
Accuracy: 0.74
Classification Report:
              precision    recall  f1-score   support

         0.0       0.74      1.00      0.85       997
         1.0       0.38      0.19      0.25        16
         2.0       0.00      0.00      0.00         5
         3.0       0.00      0.00      0.00         3
         4.0       0.00      0.00      0.00         1
         5.0       0.00      0.00      0.00         1
         9.0       0.82      0.44      0.57        32
        11.0       0.00      0.00      0.00        53
        13.0       0.00      0.00      0.00         4
        15.0       0.00      0.00      0.00         1
        16.0       0.00      0.00      0.00        62
        17.0       0.00      0.00      0.00         8
        18.0       0.00      0.00      0.00        74
        19.0       0.00      0.00      0.00         1
        20.0       0.00      0.00      0.00        18
        22.0       0.00      0.00      0.00        21
        23.0       0.00      0.00      0.00         1
        24.0       0.00      0.00      0.00         3
        25.0       0.00      0.00      0.00        17
        26.0       0.00      0.00      0.00         2
        28.0       0.00      0.00      0.00         1
        29.0       0.00      0.00      0.00         4
        30.0       0.00      0.00      0.00        33
        31.0       0.00      0.00      0.00        18

    accuracy                           0.74      1376
   macro avg       0.08      0.07      0.07      1376
weighted avg       0.56      0.74      0.63      1376
```

```
In [71]: # Import necessary libraries
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, classification_report
from sklearn.model_selection import train_test_split

# Assuming you have your data in X and y

# Split the data into training and testing sets
X_train, X_temp, y_train, y_temp = train_test_split(x, y, test_size=0.3, random_state=42)
X_test, X_val, y_test, y_val = train_test_split(X_temp, y_temp, test_size=0.5, random_state=42)

# Create an SVC model
svc_classifier = SVC(kernel='rbf', random_state=42)  # 'rbf' stands for radial basis function, a common choice

# Train the model on the training set
svc_classifier.fit(X_train, y_train)

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

# Evaluate the model on the test set
accuracy_test = accuracy_score(y_test, y_test_pred)
print(f"Testing Accuracy: {accuracy_test:.2f}")

# Make predictions on the validation set
y_val_pred = svc_classifier.predict(X_val)

# Evaluate the model on the validation set
accuracy_val = accuracy_score(y_val, y_val_pred)
print(f"Validation Accuracy: {accuracy_val:.2f}")

# Optionally, you can also print the training accuracy
y_train_pred = svc_classifier.predict(X_train)
accuracy_train = accuracy_score(y_train, y_train_pred)
print(f"Training Accuracy: {accuracy_train:.2f}")

# Display classification report for the test set
print("Classification Report for Test Set:")
print(classification_report(y_test, y_test_pred))

Testing Accuracy: 0.74
Validation Accuracy: 0.75
Training Accuracy: 0.76
```

4.4 ANN Model

Artificial Neural Networks (ANN) are multi-layer fully-connected neural nets.They consist of an input layer, multiple hidden layers, and an output layer. Every node in one layer is connected to every other node in the next layer. We make the network deeper by increasing the number of hidden layers

In [72]:
```python
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neural_network import MLPClassifier

# Load the dataset
data = pd.read_csv('thyroidDF.csv')


# Split the dataset into training, validation, and testing sets
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42)
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=0.25, random_state=42)

# Standardize the features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_val = scaler.transform(X_val)
X_test = scaler.transform(X_test)

# Define the MLP classifier
mlp = MLPClassifier(hidden_layer_sizes=(64, 32), activation='relu', max_iter=100, random_state=42)

# Train the model
epochs = 50
for epoch in range(epochs):
    mlp.partial_fit(X_train, y_train, classes=np.unique(y_train))

    # Evaluate the model on the validation set
    val_accuracy = mlp.score(X_val, y_val)
    print(f"Epoch {epoch+1}/{epochs}, Validation Accuracy: {val_accuracy}")

# Test the model
test_accuracy = mlp.score(X_test, y_test)
print(f"Test Accuracy: {test_accuracy}")


train_accuracy = mlp.score(X_train, y_train)
print(f"Train Accuracy: {train_accuracy}")
```

```
Epoch 1/50, Validation Accuracy: 0.6719346049046322
Epoch 2/50, Validation Accuracy: 0.7286103542234332
Epoch 3/50, Validation Accuracy: 0.7395095367847412
Epoch 4/50, Validation Accuracy: 0.753133514986376
Epoch 5/50, Validation Accuracy: 0.761307901907357
Epoch 6/50, Validation Accuracy: 0.7705722070844687
Epoch 7/50, Validation Accuracy: 0.7798365122615804
Epoch 8/50, Validation Accuracy: 0.7863760217983651
Epoch 9/50, Validation Accuracy: 0.7950953678474114
Epoch 10/50, Validation Accuracy: 0.8049046321525886
Epoch 11/50, Validation Accuracy: 0.8092643051771117
Epoch 12/50, Validation Accuracy: 0.8147138964577657
Epoch 13/50, Validation Accuracy: 0.8158038147138964
Epoch 14/50, Validation Accuracy: 0.8163487738419618
```

```
Epoch 14/50, Validation Accuracy: 0.8163487738419618
Epoch 15/50, Validation Accuracy: 0.8228882833787466
Epoch 16/50, Validation Accuracy: 0.8245231607629427
Epoch 17/50, Validation Accuracy: 0.8272479564032698
Epoch 18/50, Validation Accuracy: 0.8267029972752044
Epoch 19/50, Validation Accuracy: 0.8283378746594006
Epoch 20/50, Validation Accuracy: 0.8316076294277929
Epoch 21/50, Validation Accuracy: 0.8343324250681199
Epoch 22/50, Validation Accuracy: 0.8343324250681199
Epoch 23/50, Validation Accuracy: 0.8359673024523161
Epoch 24/50, Validation Accuracy: 0.8354223433242507
Epoch 25/50, Validation Accuracy: 0.8359673024523161
Epoch 26/50, Validation Accuracy: 0.8365122615803815
Epoch 27/50, Validation Accuracy: 0.8392370572207084
Epoch 28/50, Validation Accuracy: 0.8397820163487738
Epoch 29/50, Validation Accuracy: 0.8425068119891008
Epoch 30/50, Validation Accuracy: 0.8452316076294278
Epoch 31/50, Validation Accuracy: 0.8468664850136239
Epoch 32/50, Validation Accuracy: 0.8479564032697547
Epoch 33/50, Validation Accuracy: 0.849591280653951
Epoch 34/50, Validation Accuracy: 0.8501362397820164
Epoch 35/50, Validation Accuracy: 0.8517711171662126
Epoch 36/50, Validation Accuracy: 0.852316076294278
Epoch 37/50, Validation Accuracy: 0.8517711171662126
Epoch 38/50, Validation Accuracy: 0.8544959128065395
Epoch 39/50, Validation Accuracy: 0.8555858310626703
Epoch 40/50, Validation Accuracy: 0.8572207084468665
Epoch 41/50, Validation Accuracy: 0.8583106267029973
Epoch 42/50, Validation Accuracy: 0.8588555858310627
Epoch 43/50, Validation Accuracy: 0.8599455040871935
Epoch 44/50, Validation Accuracy: 0.8599455040871935
Epoch 45/50, Validation Accuracy: 0.8599455040871935
Epoch 46/50, Validation Accuracy: 0.8615803814713896
Epoch 47/50, Validation Accuracy: 0.862125340599455
Epoch 48/50, Validation Accuracy: 0.8637602179836512
Epoch 49/50, Validation Accuracy: 0.8637602179836512
Epoch 50/50, Validation Accuracy: 0.862125340599455
Test Accuracy: 0.8528610354223434
Train Accuracy: 0.9002181025081788
```

Testing the model

Testing the model

```
rf_classifier.predict([[32,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,5.218403,1.970629,108.700305,0.976056,
```

```
C:\Users\Leena\anaconda3\lib\site-packages\sklearn\base.py:420: UserWarning: X does not have valid feature names, but Random
ForestClassifier was fitted with feature names
  warnings.warn(
```

```
1]: array([31.])
```

```
svc_classifier.predict([[32,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,5.218403,1.970629,108.700305,0.976056
```

```
C:\Users\Leena\anaconda3\lib\site-packages\sklearn\base.py:420: UserWarning: X does not have valid feature names, but SVC wa
s fitted with feature names
  warnings.warn(
```

```
2]: array([0.])
```

Milestone 5: Performance Testing & Hyperparameter Tuning

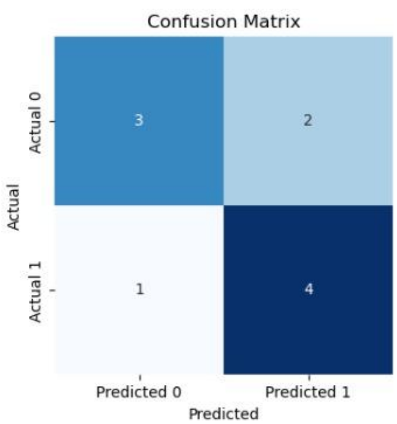
5.1 Testing model with multiple evaluation metrics

For comparing the above four models, the compareModel function is defined.

Performance Metrics

In [76]:
```python
# Import necessary libraries
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
import matplotlib.pyplot as plt
import seaborn as sns

# Sample data (replace this with your actual data)
true_labels = [1, 0, 1, 1, 0, 1, 0, 0, 1, 0]
predicted_labels = [1, 0, 1, 1, 0, 0, 1, 0, 1, 1]

# Calculate accuracy
accuracy = accuracy_score(true_labels, predicted_labels)
print(f'Accuracy: {accuracy:.2f}')

# Generate confusion matrix
conf_matrix = confusion_matrix(true_labels, predicted_labels)

# Display confusion matrix using seaborn
plt.figure(figsize=(4, 4))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', cbar=False,
            xticklabels=['Predicted 0', 'Predicted 1'],
            yticklabels=['Actual 0', 'Actual 1'])
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()

# Display classification report
class_report = classification_report(true_labels, predicted_labels)
print('Classification Report:\n', class_report)
```

```
Accuracy: 0.70
```

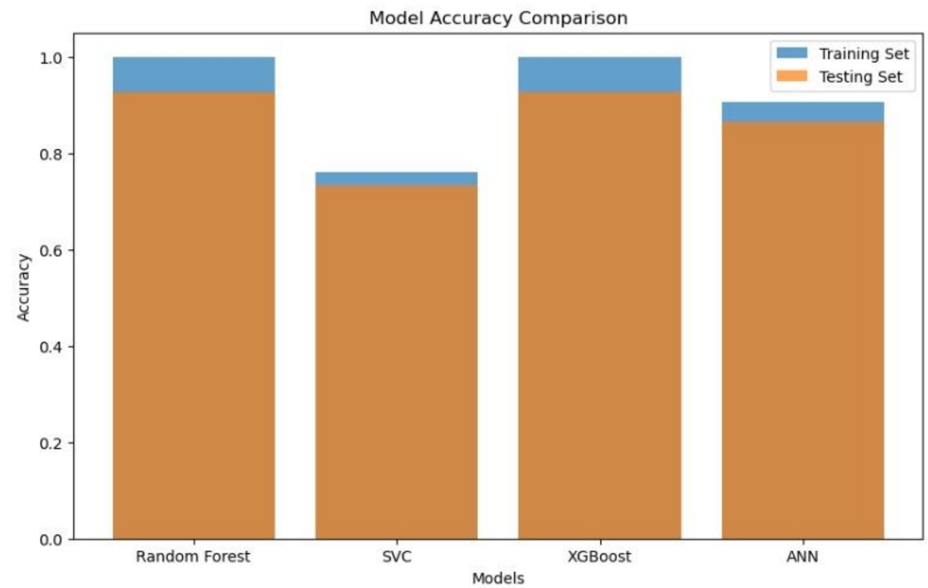



```
Classification Report:
               precision    recall  f1-score   support

           0       0.75      0.60      0.67         5
           1       0.67      0.80      0.73         5

    accuracy                           0.70        10
   macro avg       0.71      0.70      0.70        10
weighted avg       0.71      0.70      0.70        10
```

5.2 Comparing model accuracy before & after applying hyperparameter tuning

In [79]:

```python
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from xgboost import XGBClassifier
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt
from sklearn.neural_network import MLPClassifier

# Load your dataset (replace 'your_dataset.csv' with your actual file)
# Ensure that your dataset includes features and a target variable
df = pd.read_csv('thyroidDF.csv')



# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42)

# Initialize models
rf_model = RandomForestClassifier(random_state=42)
svc_model = SVC(random_state=42)
xgb_model = XGBClassifier(random_state=42)
ann_model = MLPClassifier(random_state=42, max_iter=500)  # You might need to adjust max_iter based on your data

# List of models
models = [rf_model, svc_model, xgb_model, ann_model]

# Lists to store accuracy scores
accuracy_scores_train = []
accuracy_scores_test = []

# Train and evaluate each model
for model in models:
    model.fit(X_train, y_train)

    # Training set accuracy
    y_train_pred = model.predict(X_train)
    accuracy_train = accuracy_score(y_train, y_train_pred)
    accuracy_scores_train.append(accuracy_train)

    # Testing set accuracy
    y_test_pred = model.predict(X_test)
    accuracy_test = accuracy_score(y_test, y_test_pred)
    accuracy_scores_test.append(accuracy_test)

# Model names for plotting
model_names = ['Random Forest', 'SVC', 'XGBoost', 'ANN']

# Plotting
plt.figure(figsize=(10, 6))
plt.bar(model_names, accuracy_scores_train, label='Training Set', alpha=0.7)
plt.bar(model_names, accuracy_scores_test, label='Testing Set', alpha=0.7)

plt.xlabel('Models')
plt.ylabel('Accuracy')
plt.title('Model Accuracy Comparison')
plt.legend()
plt.show()
```



After applying hyper parameter tuning

Compare the model

```python
import pandas as pd
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from xgboost import XGBClassifier
from sklearn.neural_network import MLPClassifier
import matplotlib.pyplot as plt
```

```python
# Hyperparameter grids
param_grids = {
    'Random Forest': {'n_estimators': [50, 100, 200], 'max_depth': [None, 10, 20]},
    'SVC': {'C': [1, 10, 100], 'kernel': ['linear', 'rbf']},
    'XGBoost': {'n_estimators': [50, 100, 200], 'max_depth': [3, 6, 9]},
    'ANN': {'hidden_layer_sizes': [(50,), (100,), (50, 50)], 'alpha': [0.0001, 0.001, 0.01]}
}
```

```python
models = {'Random Forest': rf_model, 'SVC': svc_model, 'XGBoost': xgb_model, 'ANN': ann_model}
```

```python
y_train = y_train.astype(int)
```

```python
# Hyperparameter tuning and training
for model_name, model in models.items():
    param_grid = param_grids[model_name]
    grid_search = GridSearchCV(model, param_grid, cv=5, scoring='accuracy', n_jobs=-1)
```

```python
grid_search.fit(X_train, y_train)
```

```
C:\Users\Leena\anaconda3\lib\site-packages\sklearn\model_selection\_split.py:700: UserWarning: The least populated class
as only 1 members, which is less than n_splits=5.
  warnings.warn(
```

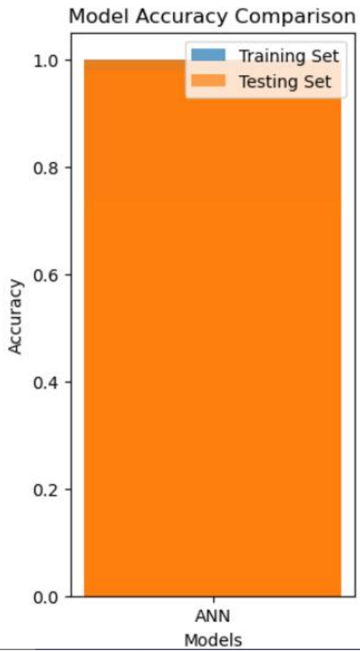
```
GridSearchCV
estimator: MLPClassifier
MLPClassifier
```

```python
best_model = grid_search.best_estimator_
```

```python
 # Evaluate on training set
train_acc = best_model.score(X_train, y_train)
accuracy_scores_train.append(train_acc)

    # Evaluate on test set
test_acc = best_model.score(X_test, y_test)
accuracy_scores_test.append(test_acc)

    # Print results for each model
print(f'{model_name} - Best Parameters: {grid_search.best_params_}')
print(f'{model_name} - Training Accuracy: {train_acc:.4f}')
print(f'{model_name} - Test Accuracy: {test_acc:.4f}')
print()
```

```
ANN - Best Parameters: {'alpha': 0.001, 'hidden_layer_sizes': (50, 50)}
ANN - Training Accuracy: 0.9849
ANN - Test Accuracy: 0.9738
```

```
plt.figure(figsize=(3, 6))
plt.bar(model_name, accuracy_scores_train, label='Training Set', alpha=0.7)
plt.bar(model_name, accuracy_scores_test, label='Testing Set', alpha=0.7)

plt.xlabel('Models')
plt.ylabel('Accuracy')
plt.title('Model Accuracy Comparison')
plt.legend()
plt.show()
```



Saving the model as thyroid1_model.pkl

Milestone 6: Model Deployment

6.1:Save the best model

Saving the best model after comparing its performance using different evaluation metrics means selecting the model with the highest performance and saving its weights and configuration. This can be useful in avoiding the need to retrain the model every time it is needed and also to be able to use it in the future.

```
In [77]: import pickle
         pickle.dump(rf_classifier,open('model.pkl','wb'))
         pickle.dump(ms,open('scaler.pkl','wb'))
```

6.2 Building Html pages:

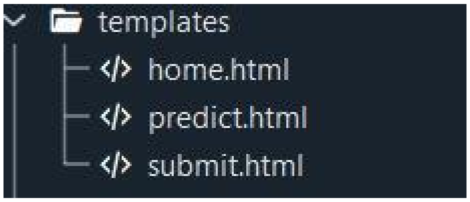
For this project project create three HTML files namely

• home.html

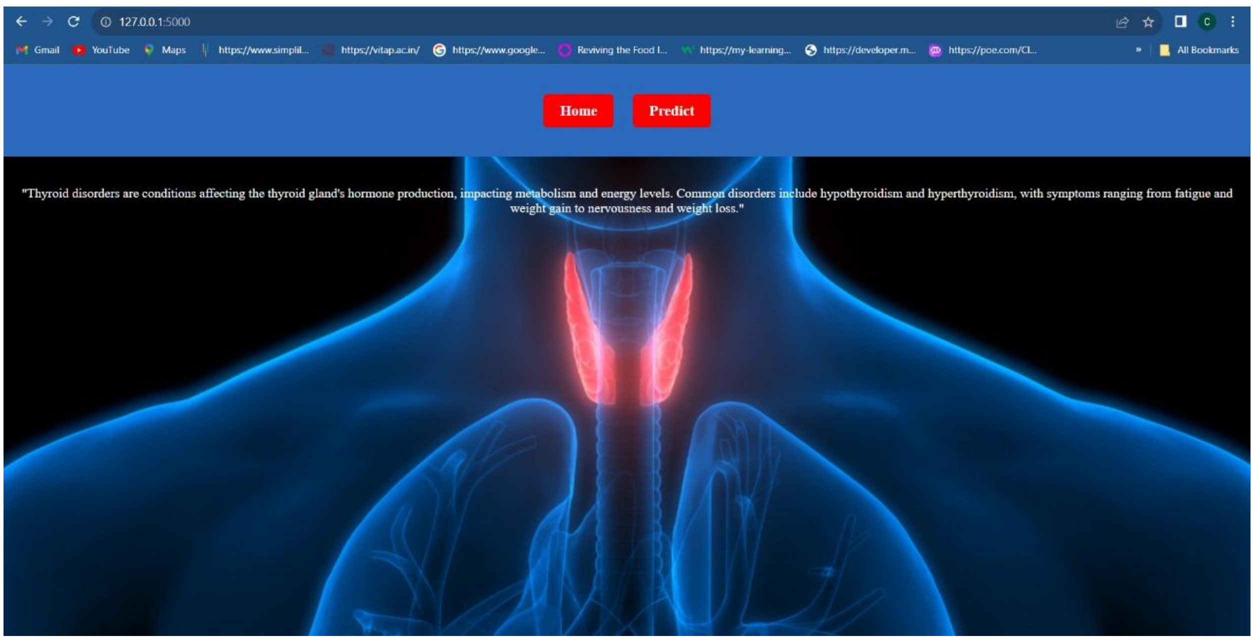
• predict.html

• submit.html

and save them in the templates folder.



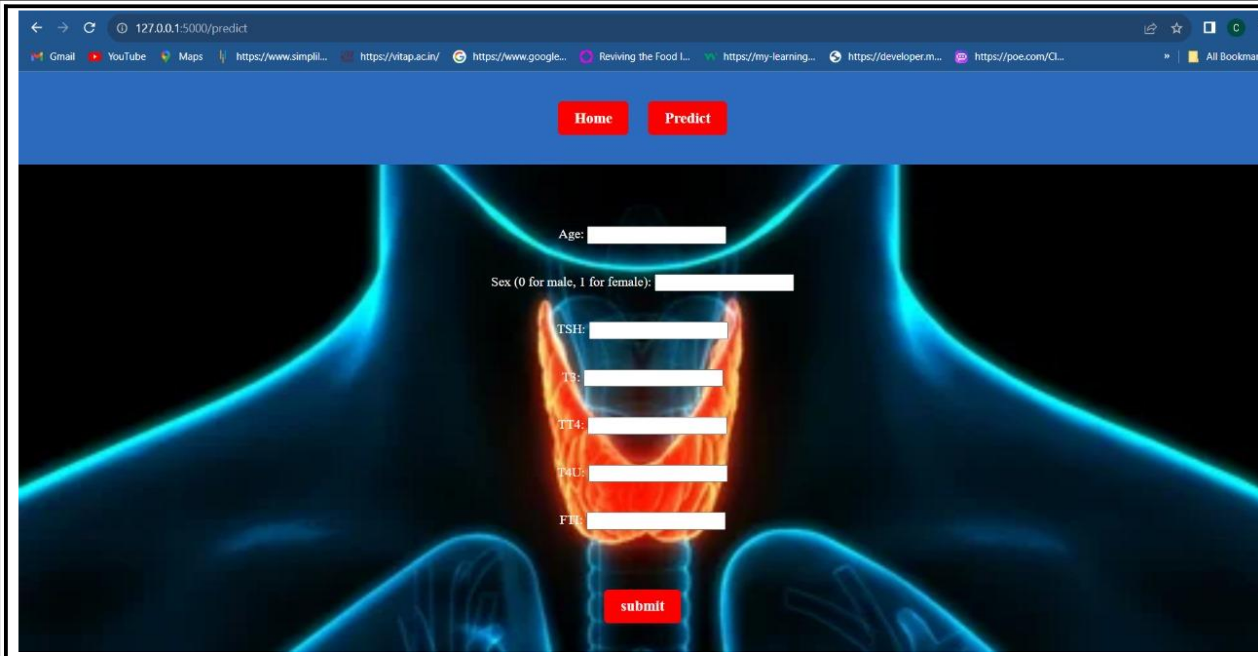
Let’s see how our home.html page looks like:



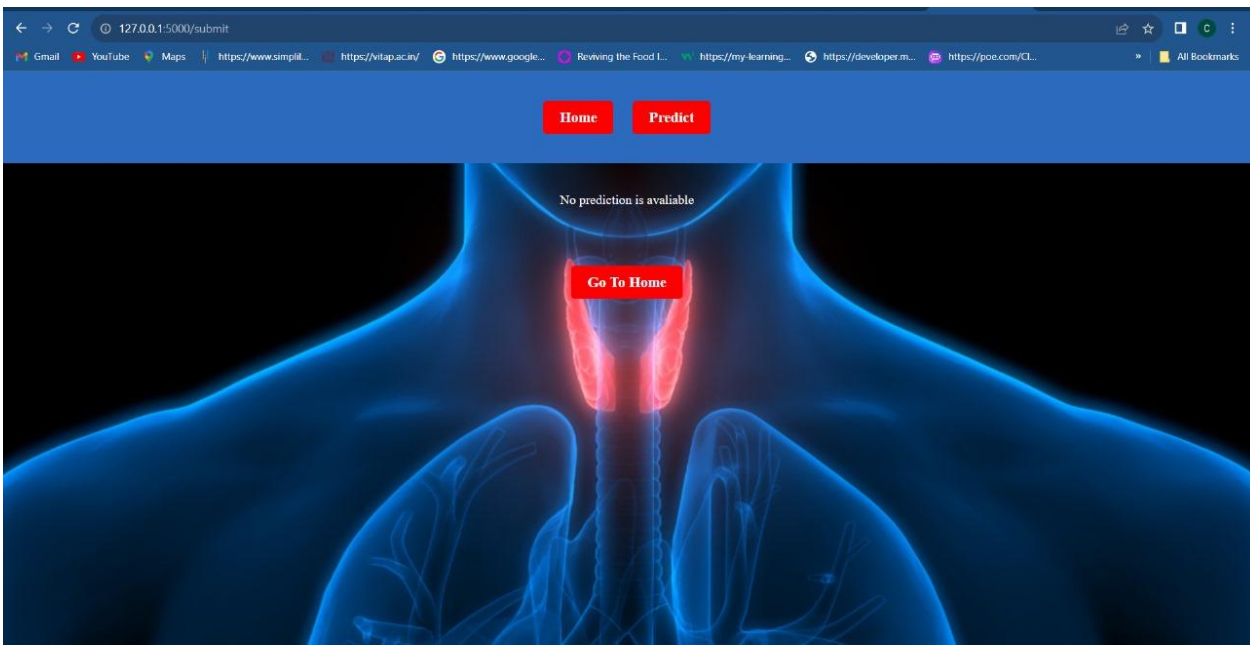
Now when you click on predict button from top right corner you will get

redirected to predict.html

Let's look how our predict.html file looks like:



Now when you click on submit button from left bottom corner you will get redirected to submit.html



6.3: Build Python code:

Import the libraries

```
from flask import Flask, render_template, request
import pandas as pd
import pickle
```

Load the saved model. Importing the flask module in the project is mandatory. An object of Flask class is our WSGI application. Flask constructor takes the name of the current module (__name__) as argument.

```
app = Flask(__name__)

# Load the pre-trained model
model = pickle.load(open("model.pkl",'rb'))
```

Render HTML page:

```
@app.route('/')
def home():
    return render_template('home.html')
```

Here we will be using a declared constructor to route to the HTML page which we have created earlier.

In the above example, '/' URL is bound with the home.html function. Hence, when the home page of the web server is opened in the browser, the html page will be rendered. Whenever you enter the values from the html page the values can be retrieved using POST Method.

Retrieves the value from UI:

```python
@app.route('/predict', methods=['GET', 'POST'])
def predict():
    prediction = None
    if request.method == 'POST':
        # Get input features from the form

        features = [float(request.form['age']),
                    float(request.form['sex']),
                    float(request.form['tsh']),
                    float(request.form['t3']),
                    float(request.form['tt4']),
                    float(request.form['t4u']),
                    float(request.form['fti'])]

        # Convert the features to a DataFrame
        input_data = pd.DataFrame([features])

        # Make predictions using the model
        prediction = model.predict(input_data)[0]

        return render_template('predict.html', prediction=prediction)

    return render_template('predict.html', prediction=None)

@app.route('/submit')
def submit():
    if request.method == 'POST':
        # Get the prediction from the form data
        prediction = request.form.get('prediction')

        return render_template('submit.html', prediction=prediction)
    else:
        # If the form is not submitted, render the page without the prediction
        return render_template('submit.html', prediction=None)
```

6.4: Run the application

When you run the “app.py”

File this window will open in the console or output terminal.Copy the URL given in the form http://127.0.0.1:5000 and paste it in the browser

```
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
 * Restarting with watchdog (windowsapi)
```

Here we are routing our app to predict() function. This function retrieves all the values from the HTML page using Post request. That is stored in an array. This array is passed to the model.predict() function. This function returns the prediction. And this prediction value will be rendered to the text that we have mentioned in the submit.html page earlier.

Main Function:

```python
if __name__ == '_main_':
    app.run(debug=True)
```