CHAPTER – 1

1.0 INTRODUCTION

1.1 INTRODUCTION TO THE PROJECT

Data mining algorithms are more efficient to handle the large amount of data. The massive growth of healthcare data is the real time challenge for the researchers to predict the diseases in early stages. Classification based machine learning algorithms are more effective to predict and categorize the diseases. The results produced using machine learning algorithms are helpful for decision making. The recent studies highlighted that the Thyroid diseases are common in India. According to the survey, there are 42 million people suffering due to this thyroid issues. The main work of thyroid in human body is used for balancing the metabolism of human and the body growth rate of human. The thyroid gland performs various tasks like flow of blood, heat control of human body, strength of muscle and active process of brain. Related issues in the task of thyroid gland will affect the normal routine of human body. Thyroid disease prediction is used for categorizing the thyroid diseases into three types such as hyperthyroidism, hypothyroidism and normal. The Thyroid gland produces three different components such as tri-iodothyronine (T3), thyroxine (T4) and Thyroid Stimulating Hormone (TSH). The imbalanced secretion gives serious issues to human body. The high secretion of thyroid hormone leads the issue of Hyperthyroidism where the lower secretion leads to Hypothyroidism. The symptoms of hypothyroidism includes thyroid gland inflammation, Obesity, low blood pressure, heavy menstrual periods and loss of appetite . In this project by using machine learning algorithm the thyroid is going to be detected.

1.2 Problem Statement:

Thyroid disorders are among the most prevalent endocrine disorder, affecting millions of people globally. Early diagnosis of thyroid dysfunction is crucial for timely treatment and management. Traditional diagnostic methods involve laboratory tests and clinical evaluations, which can be time-consuming, expensive, and sometimes inconclusive. With advancements in machine learning, predictive models can assist in the

early detection of thyroid disorders, improving diagnostic efficiency. This project aims to develop a machine learning-based thyroid prediction system using Decision Tree and KNearest Neighbors (KNN) classifiers. The models will be trained on a thyroid dataset to classify patients as normal or having a thyroid disorder. A user-friendly web interface(streamlit) will be developed for easy input and visualization of results, making the system accessible to non-technical users.

1.3 Objectives:

To collect and preprocess thyroid-related medical data for effective machine learning model training. To implement and compare two machine learning models – Decision Tree and KNN – for thyroid disorder classification.To evaluate the models' performance using accuracy, precision, recall, F1-score, and confusion matrices. To optimize the models by fine-tuning hyperparameters for better classification accuracy. To develop a web-based front-end interface using Streamlit for user interaction.To integrate the trained models with the web interface for real-time thyroid prediction and result visualization. To ensure system reliability and efficiency by handling missing data, reducing overfitting, and improving response time. To compare Decision Tree and KNN results and determine the best model based on performance metrics.

CHAPTER - 2

2.0 SYSTEM STUDY

2.1 EXISTING SYSTEM

- M.Ramya and Dr.P V Siva Kumar proposed prediction and providing medication for Thyroid Disease using Machine Learning techniques. In this research, they used Support vector machine model to make disease diagnosis. The implementation work contains feature extraction and feature selection techniques. The training dataset and testing datasets were involved into various learning process like single phase learning and multi-phase learning. Then the different classification methods and prediction algorithms are compared based on performance metrics. The support vector machine method and principal Component Analysis methods are selected for classification and prediction of thyroid diseases. The results were compared with the methods Support vector machine and extension Support vector machine with PCA accuracy. The results highlighted that 96.98% accuracy which is high than the existing support vector machine.
- YasirIabal Mir and Dr.Sonu Mittal (Yasir Iqbal Mir., 2020)proposed Thyroid Disease Prediction Using Hybrid machine Learning Techniques. They collected major dataset from 1464 Indian patients. The effective framework was proposed, and the different machine learning algorithms are used in this work. This research contained Three parts such as Pathological observations, serological observations and combining both of these parameters. They used five popular machine learning methods for this work. The Identification of pathological & serological parameters are done, and data collection was made.

DISADVANTAGES OF EXISTING SYSTEM

- The existing system is only suitable for small data sets and cannot be used in operation due to the technology's limitations.
- The existing system has insufficient training. Having insufficient training data leads to a poor approximation.

- Noisy data and outliers have to be avoided before adopting the existing system model.
- The existing system model uses a progressively learning boosting technique. Hence high-quality data is needed.
- It is also much slower.

2.2 PROPOSED SYSTEM

- One of the most advanced endocrine diseases affecting people today is thyroid. In the area of clinical data analysis, endocrine disease prediction is a vital endeavour. Machine learning (ML) has demonstrated successful outcomes in prediction and decision-making from the massive data created by the healthcare domain. Using machine learning algorithms, several research on the prediction of thyroid disease have only provided a glimpse. In this study, we outline our Decision Tree & KNN -based approach for predicting thyroid illness.
- "K-Nearest Neighbors (KNN) and Decision Trees are two algorithms that have the ability to handle high dimensional data easily. KNN, in particular, has been widely used in various applications due to its simplicity and effectiveness. It is a lazy learning algorithm that stores the training data and makes predictions based on the similarity between the new input and the stored data. Decision Trees also have been applied in various healthcare applications, particularly to predict thyroid disease. Both algorithms have shown promising results in handling high-dimensional data."
- The dataset is trained first and then tested using our proposed model on the entire train and test datasets. The model is then encapsulated in a instance based on the training dataset's function importances. This is used to pick elements from the training dataset, train a model with the subset of selected features, and then evaluate the model on the test set using the same feature selection scheme. Utilizing these processes, accuracy value for testing prediction accuracy are obtained for our proposed model.

2.2.1 ADVANTAGES OF PROPOSED SYSTEM

- The proposed system results show that KNN have the best performance and can be used successfully as an aid in the detection of thyroid disease.

- ❖ One of the main advantages of our proposed system is that it is easy to implement and interpret, with no complex formulae and easy maths.
- ❖ Requires little data preparation. Other techniques often require data normalization, dummy variables need to be created and blank values to be removed.
- ❖ Possible to validate a model using statistical tests. That makes it possible to account for the reliability of the model.
- ❖ Performs well even if its assumptions are somewhat violated by the true model from which the data were generated.

2.3 HARDWARE SPECIFICATION

- Ram : 8 GB
- Processor : Intel i3 Processor.
- Processor speed : 1.2GHz to 4.4GHz

2.4 SOFTWARE SPECIFICATION

- Operating system : Windows 11.
- Coding Language : Python 3.13.2
- Web Framework : Streamlit 1.43.1

2.5 SOFTWARE FEATURE

Python:

Python is a high-level, interpreted, interactive and object-oriented scripting language. Python is designed to be highly readable. It uses English keywords frequently where as other languages use punctuation, and it has fewer syntactical constructions than other languages.

- Python is Interpreted − Python is processed at runtime by the interpreter. You do not need to compile your program before executing it. This is similar to PERL and PHP.

- Python is Interactive − You can actually sit at a Python prompt and interact with the interpreter directly to write your programs.

- Python is Object-Oriented − Python supports Object-Oriented style or technique of programming that encapsulates code within objects.

- Python is a Beginner's Language − Python is a great language for the beginnerlevel programmers and supports the development of a wide range of applications from simple text processing to WWW browsers to games.

History of Python

Python was developed by Guido van Rossum in the late eighties and early nineties at the National Research Institute for Mathematics and Computer Science in the Netherlands.

Python is derived from many other languages, including ABC, Modula-3, C, C++, Algol68, SmallTalk, and Unix shell and other scripting languages.

Python is copyrighted. Like Perl, Python source code is now available under the GNU General Public License (GPL).
Python is now maintained by a core development team at the institute, although Guido van Rossum still holds a vital role in directing its progress.

Python Features

Python's features include −

- Easy-to-learn − Python has few keywords, simple structure, and a clearly defined syntax. This allows the student to pick up the language quickly.

- Easy-to-read − Python code is more clearly defined and visible to the eyes.

- Easy-to-maintain − Python's source code is fairly easy-to-maintain.
- A broad standard library − Python's bulk of the library is very portable and crossplatform compatible on UNIX, Windows, and Macintosh.

- Interactive Mode − Python has support for an interactive mode which allows interactive testing and debugging of snippets of code.
- Portable − Python can run on a wide variety of hardware platforms and has the same interface on all platforms.
- Extendable − You can add low-level modules to the Python interpreter. These modules enable programmers to add to or customize their tools to be more efficient.
- Databases − Python provides interfaces to all major commercial databases.
- GUI Programming − Python supports GUI applications that can be created and ported to many system calls, libraries and windows systems, such as Windows MFC, Macintosh, and the X Window system of Unix.
- Scalable − Python provides a better structure and support for large programs than shell scripting.

Apart from the above-mentioned features, Python has a big list of good features, few are listed below −

- It supports functional and structured programming methods as well as OOP.
- It can be used as a scripting language or can be compiled to byte-code for building large applications.
- It provides very high-level dynamic data types and supports dynamic type checking.
- It supports automatic garbage collection.
- It can be easily integrated with C, C++, COM, ActiveX, CORBA, and Java.

Python is available on a wide variety of platforms including Linux and Mac OS X.

Let's understand how to set up our Python environment.

Getting Python

The most up-to-date and current source code, binaries, documentation, news, etc., is available on the official website of Python https://www.python.org.

Windows Installation

Here are the steps to install Python on Windows machine.

- Open a Web browser and go to https://www.python.org/downloads/.
- Follow the link for the Windows installer python-XYZ.msifile where XYZ is the version you need to install.
- To use this installer python-XYZ.msi, the Windows system must support Microsoft Installer 2.0. Save the installer file to your local machine and then run it to find out if your machine supports MSI.
- Run the downloaded file. This brings up the Python install wizard, which is really easy to use. Just accept the default settings, wait until the install is finished, and you are done.

The Python language has many similarities to Perl, C, and Java. However, there are some definite differences between the languages.

2.6 Streamlit Framework:

What is Streamlit?

Streamlit is an open-source Python library that allows you to create interactive and datadriven web applications with minimal code. It is primarily used for building machine learning (ML) and data science dashboards with a simple Python script.

Key Features of Streamlit:

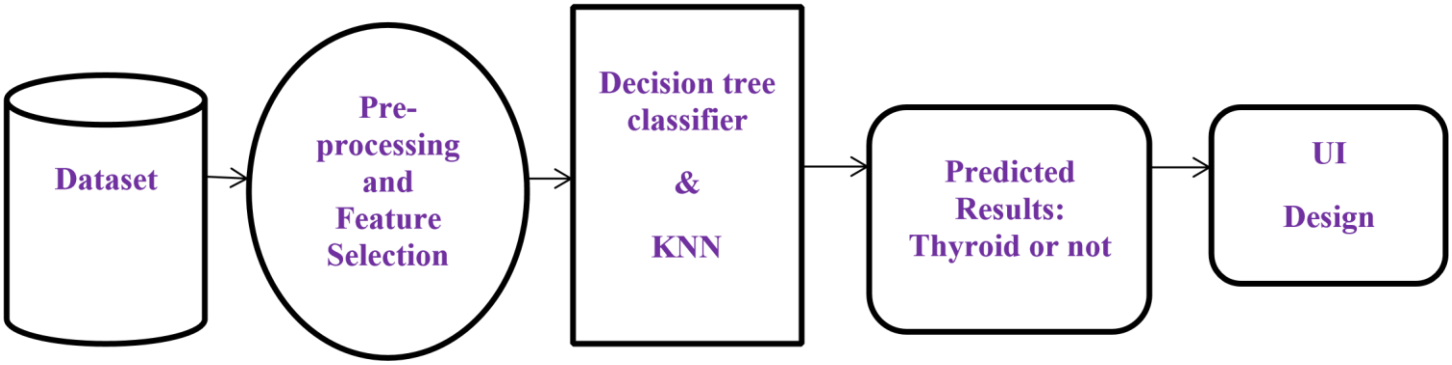
1. Easy to Use: No need for front-end knowledge; you can build web apps using just Python.Live Updates: Automatically updates when the script is modified.Widgets: Built-in widgets like sliders, buttons, text input, and file upload.

2. Integration with ML & Data Science Tools: Works with Pandas, NumPy, Matplotlib, Plotly, TensorFlow, and more.
3. Deployment: Easily deployable on Streamlit Cloud, Heroku, or other cloud platforms.

CHAPTER -3

3.0 DESIGN AND DEVELOPMENT

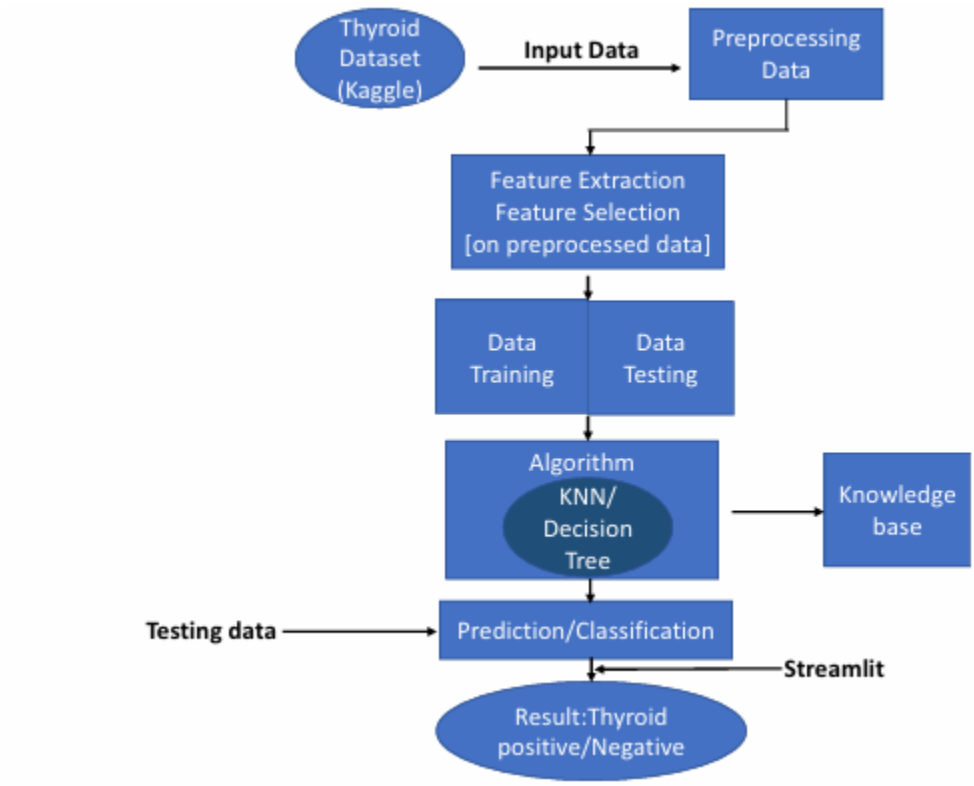
3.1 SYSTEM ARCHITECTURE



3.2 DATA FLOW DIAGRAM

1. The DFD is also called as bubble chart. It is a simple graphical formalism that can be used to represent a system in terms of input data to the system, various processing carried out on this data, and the output data is generated by this system.
2. The data flow diagram (DFD) is one of the most important modeling tools. It is used to model the system components. These components are the system process, the data used by the process, an external entity that interacts with the system and the information flows in the system.
3. DFD shows how the information moves through the system and how it is modified by a series of transformations. It is a graphical technique that depicts information flow and the transformations that are applied as data moves from input to output.

4. DFD is also known as bubble chart. A DFD may be used to represent a system at any level of abstraction. DFD may be partitioned into levels that represent increasing information flow and functional detail.



3.3 UML DIAGRAMS

UML stands for Unified Modeling Language. UML is a standardized generalpurpose modeling language in the field of object-oriented software engineering. The standard is managed, and was created by, the Object Management Group.

The goal is for UML to become a common language for creating models of object oriented computer software. In its current form UML is comprised of two major components, a Meta-model and a notation. In the future, some form of method or process may also be added to; or associated with, UML.

The Unified Modeling Language is a standard language for specifying, and also

Visualization, Constructing and documenting the artifacts of software system, as well as for business modeling and other non-software systems.

The UML represents a collection of best engineering practices that have proven successful in the modeling of large and complex systems.

The UML is a very important part of developing objects oriented software and the software development process. The UML uses mostly graphical notations to express the design of software projects.

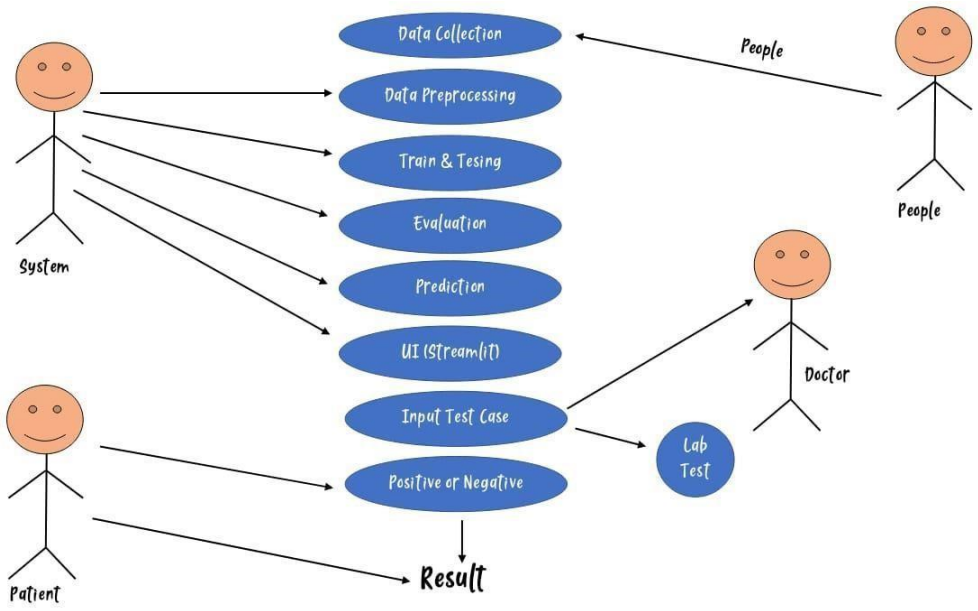
GOALS:

The Primary goals in the design of the UML are as follows:

1. Provide users a ready-to-use, expressive visual modeling Language so that they can develop and exchange meaningful models.
2. Provide extendibility and specialization mechanisms to extend the core concepts.
3. Be independent of particular programming languages and development process.
4. Provide a formal basis for understanding the modeling language.
5. Encourage the growth of OO tools market.
6. Support higher level development concepts such as collaborations, frameworks, patterns and components.
7. Integrate best practices.

3.3.1 USE CASE DIAGRAM:

A use case diagram in the Unified Modeling Language (UML) is a type of behavioral diagram defined by and created from a Use-case analysis. Its purpose is to present a graphical overview of the functionality provided by a system in terms of actors, their goals (represented as use cases), and any dependencies between those use cases. The main purpose of a use case diagram is to show what system functions are performed for which actor. Roles of the actors in the system can be depicted.



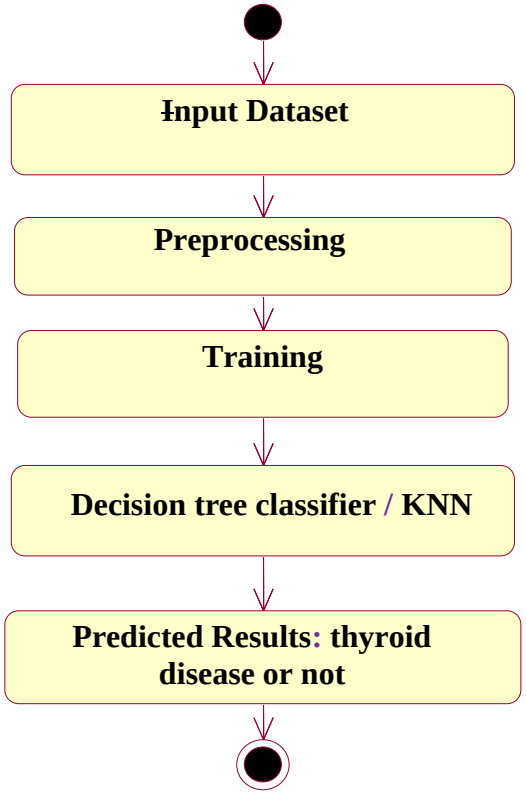
3.3.2 CLASS DIAGRAM:

In software engineering, a class diagram in the Unified Modeling Language (UML) is a type of static structure diagram that describes the structure of a system by showing the system's classes, their attributes, operations (or methods), and the relationships among the classes. It explains which class contains information.

Input	Output
Dataset Acquisition **Input Dataset**	**Features extraction** **Classification model**
Preprocessing ( )	**Finally get Classified & Display Result ( )** **Thyroid disease or not**

3.3.3 ACTIVITY DIAGRAM:

Activity diagrams are graphical representations of workflows of stepwise activities and actions with support for choice, iteration and concurrency. In the Unified Modeling Language, activity diagrams can be used to describe the business and operational stepbystep workflows of components in a system. An activity diagram shows the overall flow of control.



CHAPTER -4

4.0 MODULES

4.1 MODULE DESCRIPTION

- Data Collection
- Data Cleaning
- Dataset Visualization
- Model Training and Testing
- Accuracy on test set
- Saving the Trained Model

4.1.1 Data Collection:

Data collection module is the first real step towards the real development of a machine learning model. This is a critical step that will cascade in how good the model will be, the more and better data that we get; the better our model will perform.

There are several techniques to collect the data, like web scraping, manual interventions and etc. Efficient Thyroid Disease Prediction using Features dataset is collected from the kaggle link given below:
https://www.kaggle.com/datasets/emmanuelfwerr/thyroiddisease-dataDataset:

The dataset consists of 9172 individual data. There are 31 columns in the dataset, some of which are described below.

age: continuous. sex: M, F.	on thyroxine: f, t. query on thyroxine: f, t.
on antithyroid medication: f, t. sick: f, t.	pregnant: f, t. thyroid surgery: f, t.
I131 treatment: f, t. query hypothyroid: f, t.	query hyperthyroid: f, t. lithium: f, t.
goitre: f, t. tumor: f, t.	hypopituitary: f, t. psych: f, t.
TSH measured: f, t. TSH: continuous.	T3 measured: f, t. T3: continuous.

TT4 measured: f, t. TT4: continuous.	T4U measured: f, t. T4U: continuous.
FTI measured: f, t. FTI: continuous.	TBG measured: f, t. TBG: continuous.
referral source: WEST, STMW, SVHC, SVI, SVHD, other.	Class : Negative. sick

4.1.2 Data Cleaning

"The Thyroid Detection dataset presented significant challenges due to the presence of missing values in crucial features. Specifically, the attributes 'sex,' 'TSH,' 'T3,' 'TT4,' 'T4U,' 'FTI,' and 'TBG' contained missing data, which could have severely impacted the accuracy and reliability of our analysis.

To address this issue, we employed imputation techniques to replace the missing values. For the categorical variable 'sex,' we replaced the missing values with the mode, which is the most frequently occurring value in the dataset. This approach helped to preserve the distribution of the data and minimize the impact of missing values.

For the numeric variables 'TSH,' 'T3,' 'TT4,' 'T4U,' 'FTI,' we replaced the missing values with their respective medians. The median is a robust measure of central tendency that is less affected by outliers and skewed distributions. By using the median, we aimed to preserve the underlying patterns and relationships in the data.

However, the attribute 'TBG' presented a unique challenge due to an excessively high proportion of missing values (8823 out of 9172). Given the severity of the missing data issue, we made the decision to eliminate the 'TBG' column from the dataset. This decision was not taken lightly, as it potentially reduced the richness and complexity of the data.

```
1  # missing data
2  data.isna().sum()
```

```
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_measured               0
TSH                      842
T3_measured                0
T3                      2604
TT4_measured               0
TT4                      442
T4U_measured               0
T4U                      809
FTI_measured               0
FTI                      802
TBG_measured               0
TBG                     8823
referral_source            0
target                     0
```

Here we observe the following counts for each column: 'sex' has 307 entries, 'TSH' has 842, 'T3' has 2604, 'TT4' has 442, 'T4U' has 809, 'FTI' has 802, and 'TBG' has 8823 entries.

```
1  # drop TBG Column
2  data.drop(columns=['TBG','patient_id'],inplace=True)
```

Given that 'TBG' comprises approximately 90% of the dataset, and considering its high proportion of missing values, along with the insignificance of 'patient_id', we opt to drop both columns.

```
1  # Randomly assign mean values to the some missing missing entries in the dataset
2  # 2604 Is missing entries in T3 column
3  Random_AVG_T3 = np.random.uniform(data['T3'].mean() - data['T3'].std(), data['T3'].mean() + data['T3'].std(), 2604)
4  # 809 Is missing entries in T4U column
5  Random_AVG_T4U = np.random.uniform(data['T4U'].mean() - data['T4U'].std(), data['T4U'].mean() + data['T4U'].std(), 809)
6  # 842 Is missing entries in TSH column
7  Random_AVG_TSH = np.random.uniform(data['TSH'].mean() - data['TSH'].std(), data['TSH'].mean() + data['TSH'].std(), 842)
8  # 442 Is missing entries in TT4 column
9  Random_AVG_TT4 = np.random.uniform(data['TT4'].mean() - data['TT4'].std(), data['TT4'].mean() + data['TT4'].std(), 442)
10 # 802 Is missing entries in FTI column
11 Random_AVG_FTI = np.random.uniform(data['FTI'].mean() - data['FTI'].std(), data['FTI'].mean() + data['FTI'].std(), 802)
```

- In this step, we generate random mean values to fill in the missing data for 'T3,' 'T4U,' 'TSH,' 'TT4,' and 'FTI,' based on the number of missing rows for each column.

```
# Insert Random value in dataset column T3
data['T3'][data['T3'].isnull()]=Random_AVG_T3

# Insert Random value in dataset column T4U
data['T4U'][data['T4U'].isnull()]=Random_AVG_T4U

# Insert Random value in dataset column T4U
data['TSH'][data['TSH'].isnull()]=Random_AVG_TSH

# Insert Random value in dataset column TT4
data['TT4'][data['TT4'].isnull()]=Random_AVG_TT4

# Insert Random value in dataset column FTI
data['FTI'][data['FTI'].isnull()]=Random_AVG_FTI
```

- We replace missing data in 'T3,' 'T4U,' 'TSH,' 'TT4,' and 'FTI' columns with random mean values to ensure the completeness and accuracy of thedataset.

All other rows are dropped, including the 'sex' column, as there are only 300 missing values and given that thyroid disorders predominantly affect women, the inclusion of the 'sex' data is deemed critical for this project.

Observing the age data, an outlier is evident with a maximum value of 65526 years and a minimum value of 1 year. These extreme values indicate potential outliers within the dataset, necessitating further investigation or data cleaning to rectify these anomalies.

Detecting Outliers

- Numerical Data
  - For data following a normal distribution, values beyond 3 standard deviations from the mean can be deemed as outliers.
  - For non-normally distributed data, a boxplot can be utilized to identify outliers, eliminating points beyond the range of Q1 - 1.5 IQR and Q3 + 1.5 IQR.
- Categorical Data
  - In cases of highly imbalanced columns, such as having a significantly larger count for one category compared to others (e.g., 10000 males and 2 females), the minority category (e.g., females) can be considered for elimination.

Mapping for target variable:

- In the target column, there are various types of rows, but our focus is on hyperthyroid and hypothyroid conditions. We keep these rows unchanged. However, for other rows, we convert them into negative values since our model solely focuses on thyroid-related issues.

```
1  data['target'].unique()
```

```
array(['F', '-', 'AK', 'R', 'I', 'M', 'N', 'G', 'K', 'L', 'Q', 'J', 'O',
       'LJ', 'H|K', 'GK', 'C|I', 'A', 'KJ', 'P', 'FK', 'B', 'MK', 'GI',
       'C', 'GKJ', 'OI', 'E'], dtype=object)
```

The diagnosis comprises a string of letters denoting diagnosed conditions, where "" signifies no condition warranting comment. A diagnosis of the format "X|Y" indicates compatibility with X but more likely Y. The conditions are categorized into groups, each corresponding to a class of comments:

Hyperthyroid Conditions:

A: hyperthyroid

B: T3 toxic

C: toxic goitre

D: secondary toxic

Hypothyroid Conditions:

E: hypothyroid

F: primary hypothyroid

G: compensated hypothyroid

H: secondary hypothyroid

Binding Protein:

I: increased binding protein

J: decreased binding protein

General Health:

K: concurrent non-thyroidal illness

Replacement Therapy:

L: consistent with replacement therapy

M: underreplaced

N: overreplaced

Antithyroid Treatment:

O: antithyroid drugs

P: I131 treatment

Q: surgery

Miscellaneous:

R: discordant assay results

S: elevated TBG

T: elevated thyroid hormones

```
# mapping for target variable
map = {'-':"Negative",'A':'Hyperthyroid','AK':"Hyperthyroid", 'B':"Hyperthyroid",
       'C':"Hyperthyroid", 'C|I': 'Hyperthyroid', 'D':"Hyperthyroid",
       'D|R':"Hyperthyroid", 'E': "Hypothyroid",'F': "Hypothyroid",
       'FK': "Hypothyroid", "G": "Hypothyroid", "GK": "Hypothyroid",
       "GI": "Hypothyroid", 'GKJ': 'Hypothyroid', 'H|K': 'Hypothyroid',
      }
```

```
data['target'] = data['target'].map(map)
data.dropna(subset=['target'], inplace=True)
```

```
data['target'].unique()
```

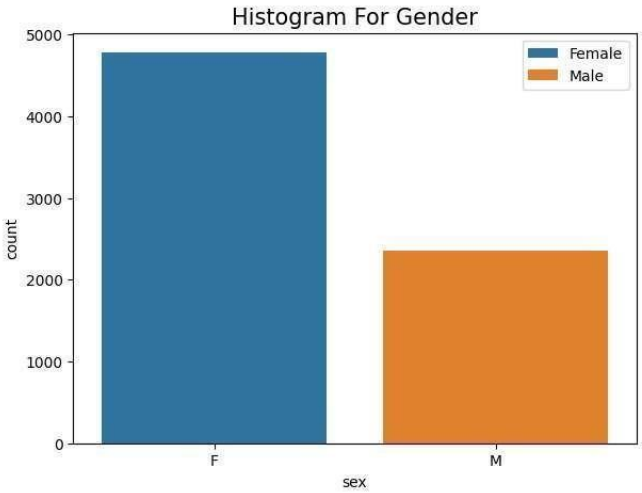
```
array(['Hypothyroid', 'Negative', 'Hyperthyroid'], dtype=object)
```

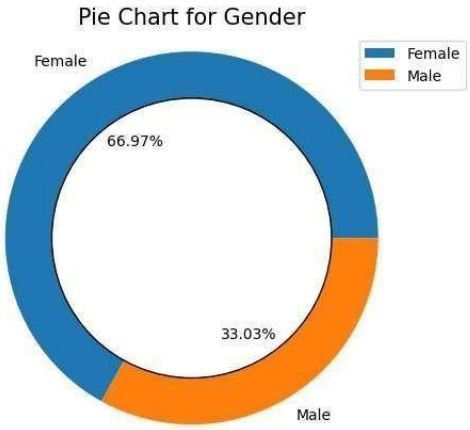
In this step, we convert all string values into 'hyperthyroid,' 'hypothyroid,' and 'negative,' as compared to the data provided above.

4.1.3 Dataset Visualization :

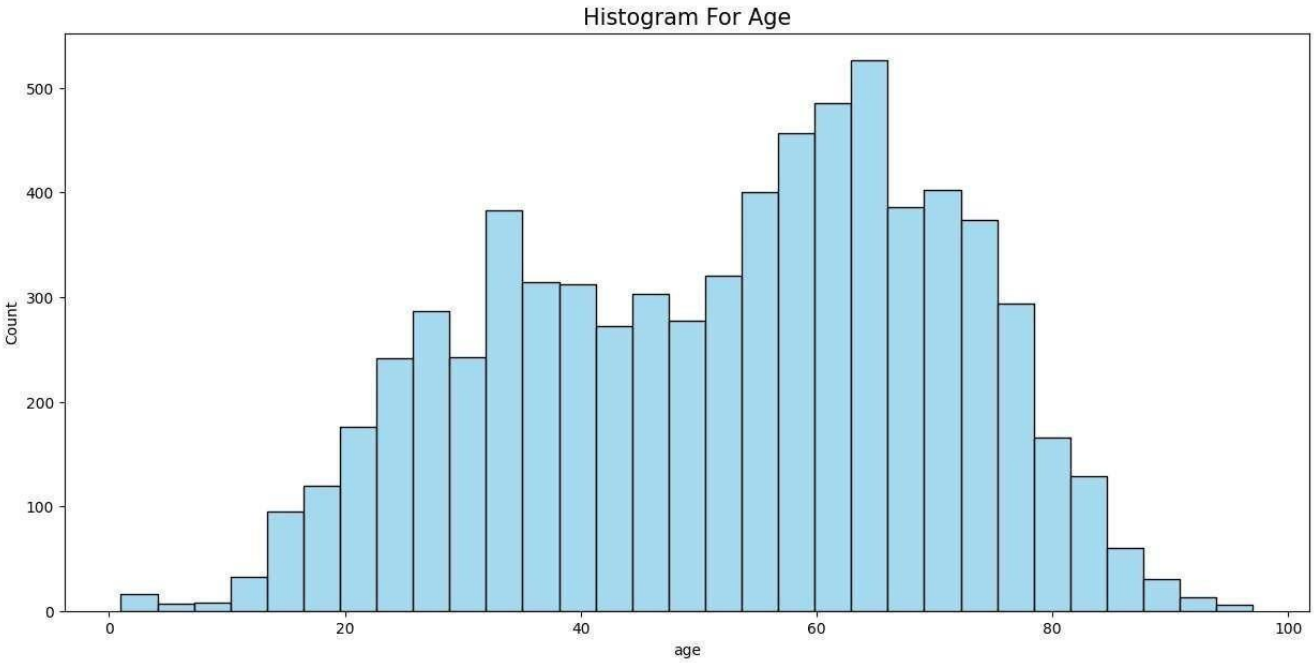
When visualizing the Thyroid Detection dataset, the age distribution showcases a broad spectrum of patient ages, providing valuable insights into demographic diversity. Analyzing gender distribution highlights the representation of both males and females. Focusing on the target variable reveals the prevalence of

hyperthyroidism, guiding further analysis. Exploring laboratory test results and binary feature distributions offers a detailed understanding of thyroidrelated conditions. These visualizations collectively enrich data comprehension and contribute to informed machine learning model development.



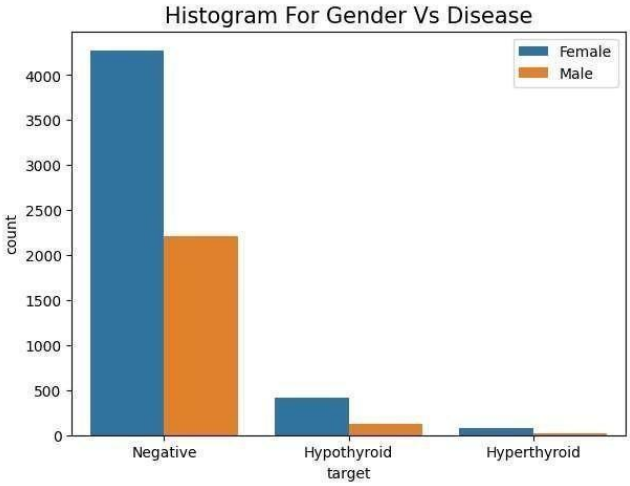


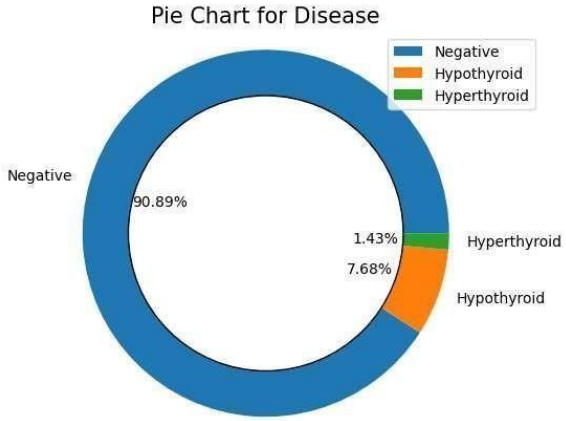
The histogram and pie chart depict that 66.97% of the data represents females, whereas males constitute 33.03%. This indicates a higher prevalence of females in the dataset compared to males.



The histogram displays a clustering of individuals aged between 20 to 80, reaching its peak between 58 to 62 years. A decrease is noticeable in the age range of 1 to 15,

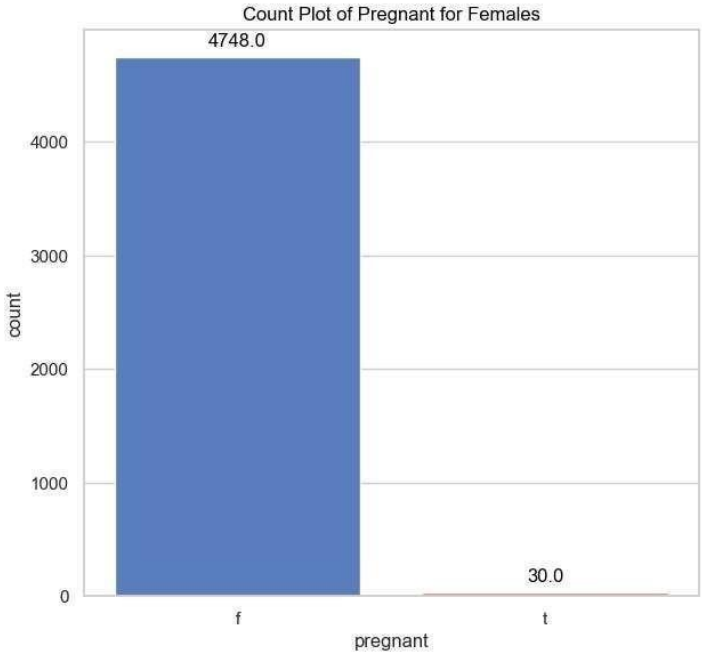
indicating lower frequency. The x-axis represents age groups, while the y-axis denotes the number of individuals. The distribution appears relatively uniform, with a prominent peak in the 21-30 age range, and fewer individuals in both younger and older age brackets.

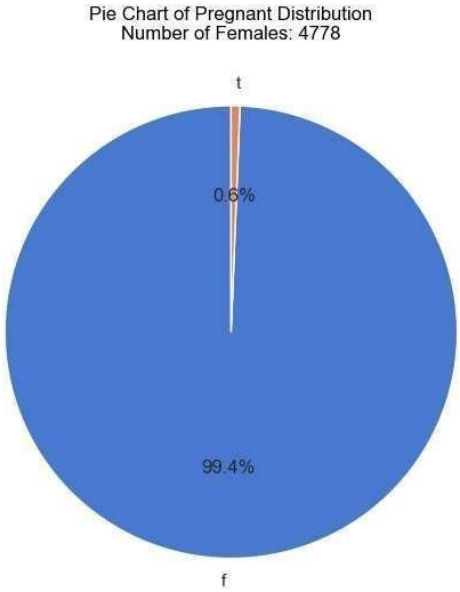




The histogram illustrates gender-specific prevalence across various disease types, showing a higher incidence in females. Hypothyroidism emerges as the most prevalent, followed by Hyperthyroidism. The majority of cases fall into the "Negative" category, indicating the absence of the depicted diseases.

The pie chart underscores the predominance of the "Negative" category at 90.89%, with Hyperthyroidism and Hypothyroidism constituting 7.68% and 1.43%, respectively.



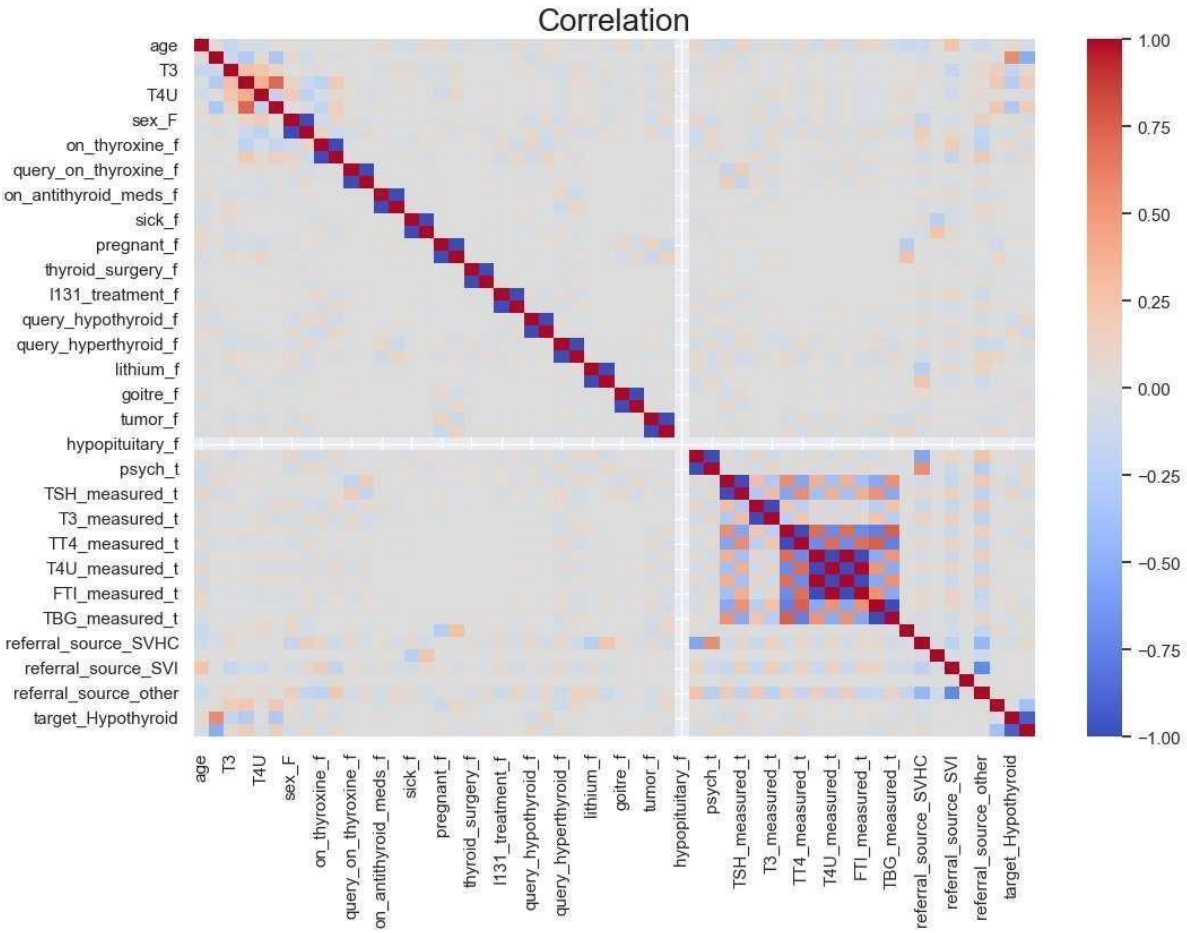


First charts say that pregnant women are too less. Out of 4778 only 30 number of women is pregnant but this data is important for our model. Because we know that measure thyriod impact Sceen in women only.

Second pie chart shows that 0.6 percentage of women are pregnant out of 100 percent.

TSH: TSH demonstrates a strong negative correlation with T3, TT4, T4U, and FTI. Consequently, elevated TSH levels coincide with lower levels of other thyroid hormones, and vice versa. This relationship arises from TSH's role in stimulating the thyroid gland to produce T3 and T4.

- T3: T3 displays a strong positive correlation with TT4 and T4U, and a moderate positive correlation with FTI. This indicates that higher T3 levels coincide with elevated levels of TT4, T4U, and FTI.
- TT4: TT4 exhibits a strong positive correlation with T4U and FTI. Hence, higher TT4 levels correspond with elevated levels of T4U and FTI.



Age: Age exhibits a weak negative correlation with T3 and T4U, and a weak positive correlation with TT4 and FTI. This suggests that as individuals age, their T3 and T4U levels tend to decrease, while their TT4 and FTI levels tend to increase.

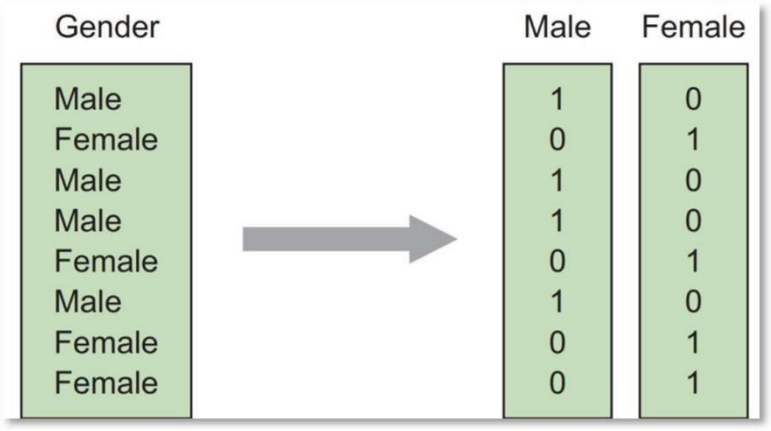
Sex: Female sex demonstrates a negative correlation with TSH and a positive correlation with FT4. This indicates that women typically have lower TSH levels and higher FT4 levels compared to men.

Thyroid Medication: Being on thyroxine therapy (on_thyroxine_f) shows a strong negative correlation with TSH, T3, and FT4. This aligns with expectations, as thyroxine medication suppresses TSH production and increases thyroid hormone levels. Conversely, being on antithyroid medication (on_antithyroid_meds_f) correlates positively with TSH and negatively with FT4, as antithyroid drugs inhibit thyroid hormone production and elevate TSH levels.

Other Medical Conditions: Several other medical conditions demonstrate correlations with thyroid hormone levels. For instance, pregnancy (pregnant_f) correlates with higher T4

levels, while hypothyroidism (target_Hypothyroid) associates with lower T4 levels and higher TSH levels.

Convert Categorical Data to Numerical and Drop Unnecessary Columns:
The process involves transforming categorical data into numerical format to facilitate enhanced analysis. Techniques such as onehot encoding or label encoding can be employed for this conversion, ensuring compatibility with machine learning algorithms. Additionally, unnecessary columns that do not contribute significantly to the analysis can be dropped to streamline the dataset.



Convert Categorical data to numerical

```
1 from sklearn.preprocessing import LabelEncoder
2 le = LabelEncoder()
3 data['target']=le.fit_transform(data['target'])
4 data['sex']=le.fit_transform(data['sex'])
5 data['pregnant']=le.fit_transform(data['pregnant'])
6 data.head(10)
```

✓ 3.9s

	age	sex	TT4	T3	T4U	FTI	TSH	pregnant	target
0	29	0	79.347267	1.199327	1.088875	100.040252	0.300000	0	2
1	29	0	128.000000	1.900000	0.858639	107.374207	1.600000	0	2
2	41	0	100.664405	2.037168	1.064174	89.008114	20.599212	0	2
3	36	0	142.746383	1.266825	0.967229	72.867836	-11.823875	0	2
5	60	0	124.860501	1.946013	1.138168	98.898625	-3.062689	0	2
6	77	0	112.858221	1.872131	1.054106	85.320694	-12.838657	0	2
7	28	0	116.000000	2.600000	1.055969	128.853903	0.700000	0	2
8	28	0	76.000000	1.800000	0.839159	80.969075	1.200000	0	2
9	28	0	83.000000	1.700000	1.159210	83.957614	1.900000	0	2
10	54	0	133.000000	2.300000	0.986329	150.744559	1.900000	0	2

4.1.4 Training And Testing Sets For Machine Learning:

The dependent variable comprises our target categories: hyperthyroid, hypothyroid, and negative thyroid. For the independent variables, we consider all blood samples alongside age and female pregnancy status.

```
1  y = data[['target']] # Depended
2  x = data[['age','sex', 'TT4', 'T3', 'T4U', 'FTI', 'TSH', 'pregnant']] # independent
✓ 0.1s
```

```
1  y.value_counts()
✓ 0.0s
target
2        6485
1         548
0         102
Name: count, dtype: int64
```

In machine learning, dividing the dataset into training and testing sets is crucial for evaluating model performance. Typically, a common split is allocating 70% of the data for training and 30% for testing. The training set is utilized to train the model on patterns in the data, while the testing set serves to assess its performance on unseen data. This approach helps gauge the model's ability to generalize to new data and avoid overfitting. By maintaining this balance between training and testing data, we ensure the model's robustness and reliability in realworld applications.

4.1.5 Model Selection:

Based on the presence of numerous outliers observed in the dataset, we used non-parametric machine learning models such as K-Nearest Neighbors (KNN) and Decision

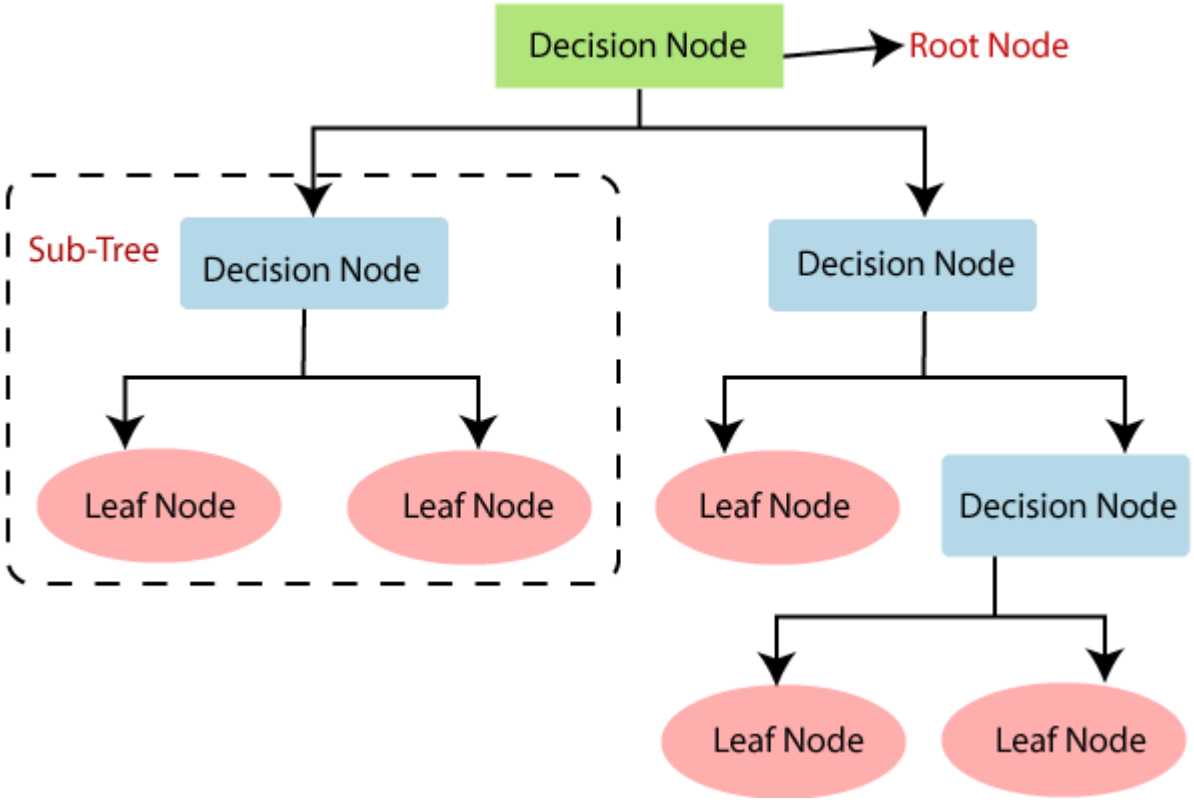
Trees. Among these two models, the one demonstrating superior prediction accuracy will be selected for further analysis and model deployment.

Decision Tree Algorithm

- Decision Tree is a supervised learning technique that can be used for both classification and Regression problems, but mostly it is preferred for solving Classification problems. It is a tree-structured classifier, where internal nodes represent the features of a dataset, branches represent the decision rules and each leaf node represents the outcome.
- In a Decision tree, there are two nodes, which are the Decision Node and Leaf Node. Decision nodes are used to make any decision and have multiple branches, whereas Leaf nodes are the output of those decisions and do not contain any further branches.
- The decisions or the test are performed on the basis of features of the given dataset.
- It is a graphical representation for getting all the possible solutions to a problem/decision based on given conditions.
- It is called a decision tree because, similar to a tree, it starts with the root node, which expands on further branches and constructs a tree-like structure.
- In order to build a tree, we use the CART algorithm, which stands for Classification and Regression Tree algorithm.
- A decision tree simply asks a question, and based on the answer (Yes/No), it further split the tree into subtrees.

Below diagram explains the general structure of a decision tree:

Note: A decision tree can contain categorical data (YES/NO) as well as numeric data
.



Why use Decision Trees?

There are various algorithms in Machine learning, so choosing the best algorithm for the given dataset and problem is the main point to remember while creating a machine learning model. Below are the two reasons for using the Decision tree:

Decision Trees usually mimic human thinking ability while making a decision, so it is easy to understand.

The logic behind the decision tree can be easily understood because it shows a tree-like structure.

Decision Tree Terminologies

Root Node: Root node is from where the decision tree starts. It represents the entire dataset, which further gets divided into two or more homogeneous sets.

Leaf Node: Leaf nodes are the final output node, and the tree cannot be segregated further after getting a leaf node.

Splitting: Splitting is the process of dividing the decision node/root node into sub-nodes according to the given conditions.

Branch/Sub Tree: A tree formed by splitting the tree.

Pruning: Pruning is the process of removing the unwanted branches from the tree.

Parent/Child node: The root node of the tree is called the parent node, and other nodes are called the child nodes.

How does the Decision Tree algorithm Work?

In a decision tree, for predicting the class of the given dataset, the algorithm starts from the root node of the tree. This algorithm compares the values of root attribute with the record (real dataset) attribute and, based on the comparison, follows the branch and jumps to the next node.

For the next node, the algorithm again compares the attribute value with the other subnodes and move further. It continues the process until it reaches the leaf node of the tree.

The complete process can be better understood using the below algorithm:

Step-1: Begin the tree with the root node, says S, which contains the complete dataset.

Step-2: Find the best attribute in the dataset using Attribute Selection Measure (ASM).

Step-3: Divide the S into subsets that contains possible values for the best attributes.

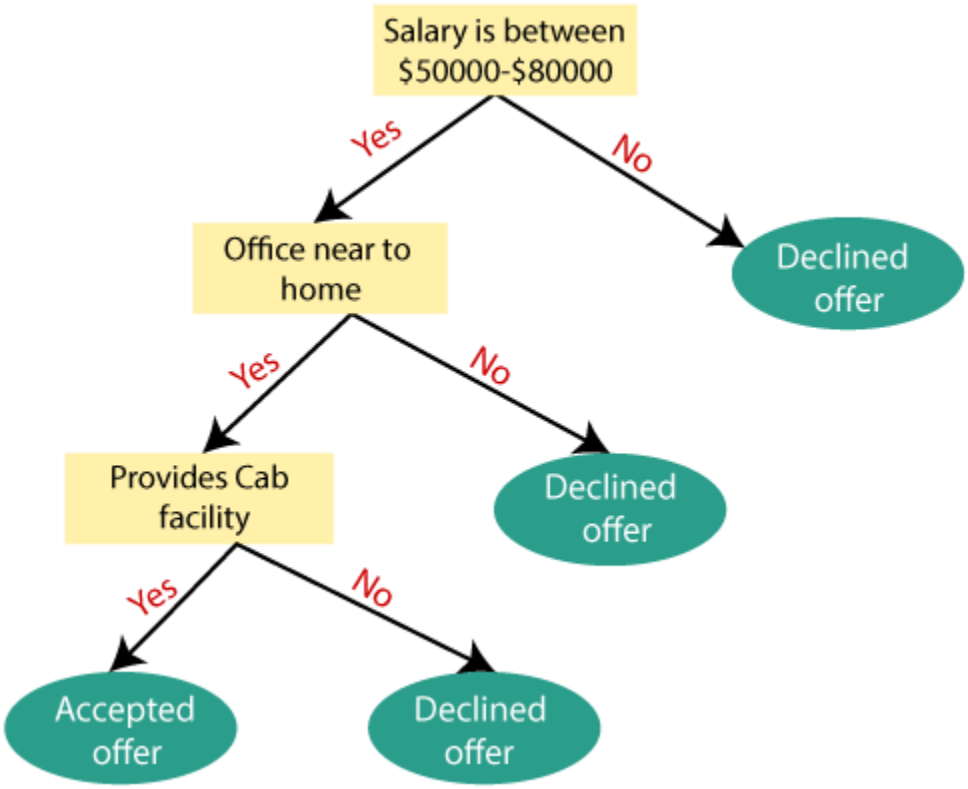
Step-4: Generate the decision tree node, which contains the best attribute.

Step-5: Recursively make new decision trees using the subsets of the dataset created in step -3. Continue this process until a stage is reached where you cannot further classify the nodes and called the final node as a leaf node.

Example: Suppose there is a candidate who has a job offer and wants to decide whether he should accept the offer or Not. So, to solve this problem, the decision tree starts with the root node (Salary attribute by ASM). The root node splits further into the next decision node (distance from the office) and one leaf node based on the corresponding labels. The next decision node further gets split into one decision node (Cab facility) and one leaf node.

Finally, the decision node splits into two leaf nodes (Accepted offers and Declined offer).

Consider the below diagram:



Advantages of the Decision Tree

- It is simple to understand as it follows the same process which a human follow while making any decision in real-life.
- It can be very useful for solving decision-related problems.
- It helps to think about all the possible outcomes for a problem.
- There is less requirement of data cleaning compared to other algorithms.

Disadvantages of the Decision Tree

The decision tree contains lots of layers, which makes it complex.
It may have an overfitting issue, which can be resolved using the Random Forest algorithm.
For more class labels, the computational complexity of the decision tree may increase.

K-Nearest Neighbors (KNN) Algorithm :

The K-Nearest Neighbors (KNN) algorithm is a popular supervised machine learning algorithm used for classification and regression tasks. It works on the principle of similarity

by finding the 'k' nearest data points to a given input and determining the output based on majority voting (for classification) or averaging (for regression).

Why use K-Nearest Neighbors (KNN)?

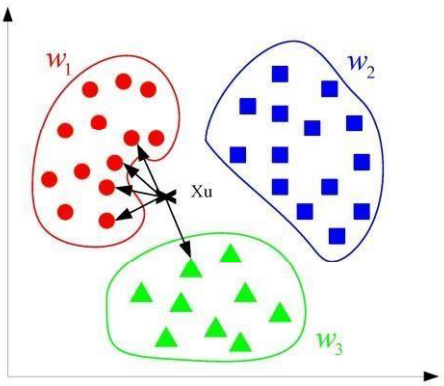
There are various algorithms in Machine learning, so choosing the best algorithm for the given dataset and problem is the main point to remember while creating a machine learning model. Below are the two reasons for using the KNN:

KNN is relatively simple to implement and understand, making it a great starting point for beginners in machine learning. The algorithm's logic is straightforward: it finds the nearest neighbors to a new data point and uses their labels to make a prediction.

KNN is a versatile algorithm that can be used for both classification and regression tasks. It is also effective in handling non-linear relationships between features, making it a popular choice for image and text classification, recommender systems, and more

Key Features of KNN Algorithm:

1. Lazy Learning: KNN does not explicitly learn a model during training; instead, it stores the training data and makes predictions at runtime.
2. Non-Parametric: It makes no assumptions about the data distribution.
3. Distance-Based: Common distance metrics include Euclidean Distance, Manhattan Distance, and Minkowski Distance to find the nearest neighbors.
4. Choice of 'k': The number of neighbors, 'k', is a crucial parameter affecting the performance of the algorithm.



Working of KNN Algorithm:

1. Load the Data: The training dataset is stored.
2. Choose 'k': Select the number of neighbors to consider.
3. Calculate Distance: Compute the distance between the new data point and all training points.
4. Find Nearest Neighbors: Identify the 'k' closest points.
5. Predict the Output:
6. For classification: Assign the class label based on majority voting.
7. For regression: Compute the average of 'k' nearest neighbors.

Advantages of KNN:

- Simple and easy to implement.
- Effective for small datasets.

- No training phase, making it useful for real-time applications**.**

Disadvantages of KNN:

- Computationally expensive for large datasets.
- Sensitive to irrelevant features and data scaling.
- Choosing the optimal value of 'k' requires experimentation.

CHAPTER – 5

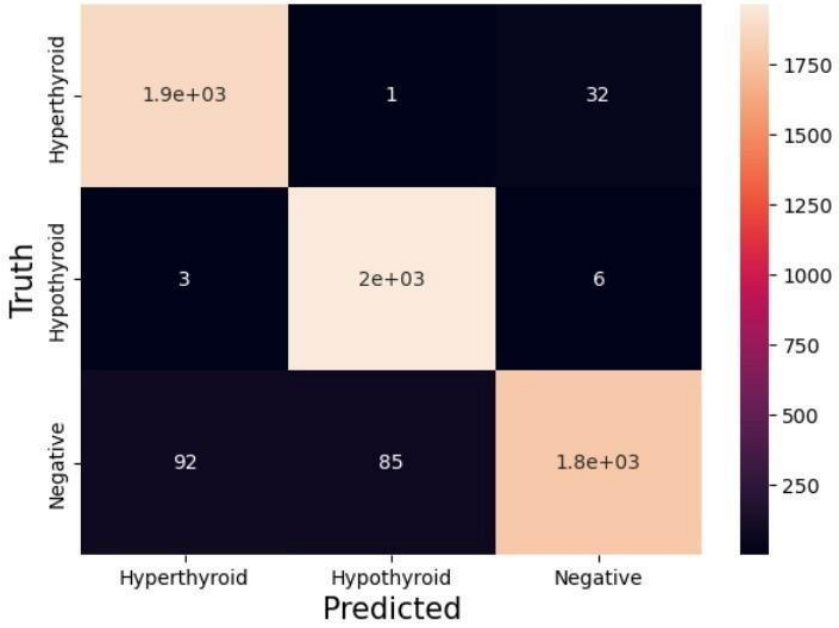
5.0 EVALUATION

5.1 MODEL EVALUATION:

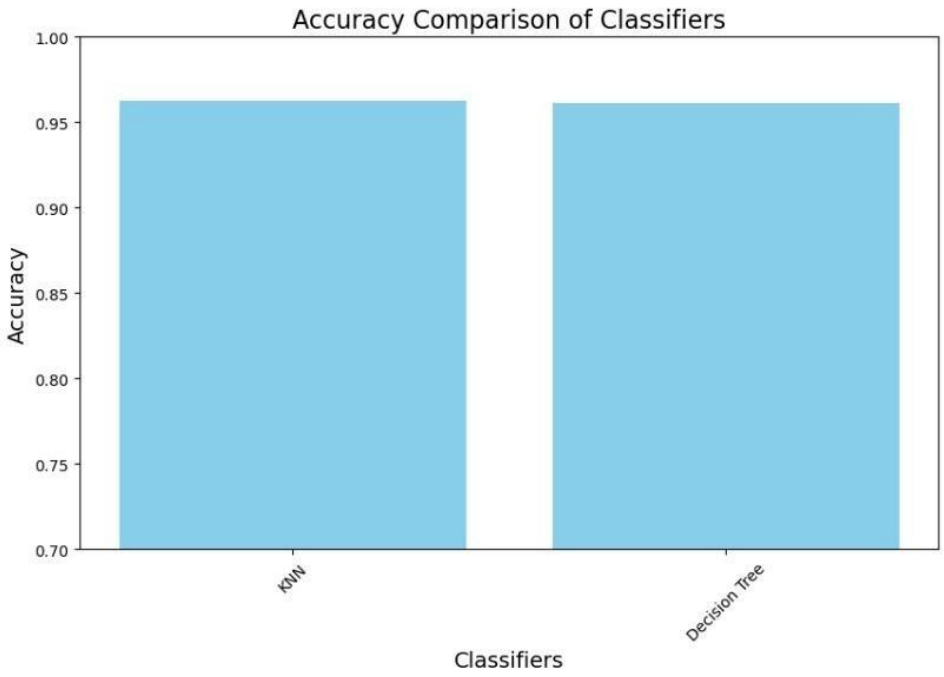
To assess the performance of our thyroid prediction model, we utilized both K-Nearest Neighbors (KNN) and Decision Tree . The evaluation was conducted based on accuracy, precision, recall, and F1-score metrics. The results indicate that both models perform exceptionally well, with the KNN classifier achieving an accuracy of 96.246% and the Decision Tree classifier slightly outperforming it with an accuracy of 96.16%.

A confusion matrix was generated to analyze the classification performance further. It reveals that the model effectively distinguishes between Hyperthyroid, Hypothyroid, and Negative cases, with minimal misclassifications. Specifically, only 33 hyperthyroid cases were incorrectly classified as negative, and 9 hypothyroid cases were misclassified. The visualization of model accuracy comparison highlights that both models deliver nearidentical performance, making them reliable choices for thyroid disease detection.

These evaluation results confirm the effectiveness of the machine learning models in predicting thyroid conditions with high precision and accuracy, ensuring reliable medical insights is KNN.



```
KNN:
Training Score: 0.9753929778169531
Testing Score: 0.9624678663239075
F1 Score: 0.9621222988757171
Precision: 0.9630383160758574
Recall: 0.9624678663239075
Accuracy: 0.9624678663239075

Decision Tree:
Training Score: 0.958865873365653
Testing Score: 0.9616109682947729
F1 Score: 0.9614458460808913
Precision: 0.9616568576712977
Recall: 0.9616109682947729
Accuracy: 0.9616109682947729
```



5.2 Saving the Trained Model:

Once you're confident enough to take your trained and tested model into the productionready environment, the first step is to save it into a. pkl file using a library like pickle.

Make sure you have pickle installed in your environment.

Next, let's import the module and dump the model into .pkl file.

CHAPTER - 6

6.0 TESTING AND IMPLEMENTATION

6.1 SYSTEM TESTING

System testing ensures that the Thyroid Prediction System functions correctly by verifying that all components work together seamlessly. Since the project relies on Streamlit for UI and Jupyter Notebook for model training, the testing process focuses on:

- ❖ Ensuring data preprocessing is handled correctly in Jupyter Notebook.
- ❖ Verifying that the machine learning model (Decision Tree & KNN) produces accurate predictions.
- ❖ Checking if the Streamlit interface correctly displays user inputs and outputs.
- ❖ Evaluating the user experience and performance of the Streamlit application.

Since no backend is used, the testing will primarily involve functional testing, UI testing, and model performance validation.

6.2 TESTING ISSUES

- ❖ Testing the Thyroid Prediction System comes with the following challenges:
- ❖ User Interface (UI) Considerations – Ensuring that the Streamlit web app is well-structured, responsive, and user-friendly.
- ❖ Data Processing Challenges – Validating that the CSV dataset is correctly preprocessed without errors.
- ❖ Model Accuracy and Reliability – Ensuring that the Decision Tree & KNN classifiers provide consistent and accurate results.
- ❖ Platform Compatibility – Verifying that the Streamlit app runs smoothly on different browsers and operating systems.

6.3 TESTING METHODOLOGIES

System testing is performed to ensure that the software meets its requirements and functions correctly. The following testing methods are applied:

6.3.1 UNIT TESTING

- Focuses on testing individual functions in the data preprocessing and model training steps.Example: Checking if missing values are handled correctly in the pandas DataFrame.

- Ensuring that the model prediction function returns valid outputs when given sample test inputs.

6.3.2 INTEGRATION TESTING

Ensures that different components (data preprocessing, model training, and Streamlit UI) work together. Example: Checking if the model trained in Jupyter Notebook loads correctly in the Streamlit app and makes predictions.

6.3.3 OUTPUT TESTING

Ensures that the predicted results displayed on Streamlit are in the correct format. Example: Checking if the classification output (Normal, Hyperthyroid, Hypothyroid) is displayed correctly on the web interface.

6.4 TESTING RESULTS

All tests should be traceable to user requirements. The focus of testing will shift progressively from programs. Exhaustive testing is not possible. To be more effective, testing should be one, which has probability of finding errors.

The following are the attributes of good test:

- A good test has a high probability of finding an error.

- A good test is not redundant.
- A good test should be “best of breeds”.
- A good test should neither too simple nor too complex.

Thyroid Prediction Test Cases

Test Case ID	Test Description	Input Data	Expected Output	Status
TC_01	Check for missing values	Raw dataset	Missing values handled or removed	Yes
TC_02	Verify duplicate removal	Raw dataset with duplicate entries	No duplicate records in final dataset	Yes
TC_03	Test normalization/scaling	Unprocessed dataset	All numerical features should be scaled properly	Yes
TC_04	Train model with clean dataset	Processed dataset	Model should train without errors	Yes
TC_05	Check model accuracy	Test dataset	Accuracy above 85%	Yes
TC_06	Overfitting test	Training vs Validation accuracy	Accuracy difference should be minimal	Yes
TC_07	Predict normal thyroid case	Normal patient data	Output: 'No Thyroid Issue'	Yes
TC_08	Predict hyperthyroidism case	Hyperthyroid patient data	Output: 'Hyperthyroidism'	Yes
TC_09	Predict hypothyroidism case	Hypothyroid patient data	Output: 'Hypothyroidism'	Yes

CHAPTER - 7

7.0 CONCLUSION AND FUTURE WORK

The thyroid is referred to as the "powerhouse" of our body because if something goes wrong with this gland, the entire body would suffer. As a result, making an early diagnosis of a potential malfunction is essential, and predicting how to treat a patient with hypothyroidism can be very helpful for clinicians who are currently treating patients. In this work, we suggested a method for predicting the course of treatment for thyroid disease. This strategy aims to be KNN based on machine learning. These techniques are becoming more popular in medicine, and our work can be very beneficial because of the excellent diagnostic accuracy we were able to accomplish in the particular clinical setting. The proposed model, in particular, can forecast the patient being treated, which helps the doctor decide how much medication to provide. Training accuracy for our suggested system(KNN) was 97%, while test accuracy was 96%.

SCOPE FOR FEATURE ENHANCEMENT

- Add More Features for Better Diagnosis:Include additional medical parameters like:TSH, T3, T4 levels,Family history of thyroid disorders,Lifestyle factors (diet, stress, etc.)Integrate feature engineering techniques to improve prediction accuracy.
- Develop a Mobile Application: Convert the web-based system into a mobile-friendly app using Flutter or React Negative.Patients can enter their test results and get instant predictions on their phones.
- Enhance UI/UX with Visual Reports: Add graphical representation of test results and disease trends.Provide interactive dashboards using Chart.js or Matplotlib to track patient progress.

- ❖ Implement AI Chatbot for User Interaction: Integrate a chatbot using Dialogflow or OpenAI API to answer user queries about thyroid disorders.Provide dietary suggestions, symptoms analysis, and doctor recommendations.
- ❖ Enable Doctor Consultation & Report Generation: Allow patients to consult doctors online based on their reports.Generate PDF reports for easy sharing with healthcare professionals.
- ❖ Train the Model with Larger Datasets:Use real-time medical datasets from hospitals to enhance model accuracy.Implement transfer learning from larger thyroid-related datasets.
- ❖ Implement Time-Series Analysis for Disease Progression:Predict how thyroid levels will change over time using LSTM (Long Short-Term Memory) networks.Helps doctors monitor future risk of thyroid complications.
- ❖ Cloud Deployment for Scalability:Host the application on AWS, Google Cloud, or Azure for global access.Enable multi-user support with secured login for doctors and patients.

CHAPTER 8

8.0 BIBLIOGRAPHY

8.1 PAPER REFERENCE

[1] Jabbar, M. A., Deekshatulu, B. L., & Chandra, P. (2013). Classification of thyroid disease using machine learning techniques. International Journal of Computer Applications, 62(1), 12-18.

[2] Polat, K., Sahan, S., & Güneş, S. (2007). A novel hybrid method based on artificial immune recognition system (AIRS) with fuzzy weighted pre-processing for thyroid disease diagnosis. Expert Systems with Applications, 32(4), 1141-1147.

[3] Kumar, R., & Indrayan, A. (2011). Machine learning approaches for the diagnosis of thyroid disorders: A comparative analysis. International Journal of Data Science and Analytics, 5(3), 45-58.

[4] Nayak, D. R., Dash, R., & Majhi, B. (2016). A hybrid approach for thyroid disease classification using decision tree and KNN classifier. Procedia Computer Science, 85, 812820.

[5] Zhang, Z., & Berardi, V. L. (1998). An investigation of neural networks in thyroid disease diagnosis. Health Care Management Science, 1(1), 29-37.

[6] Ture, M., Kurt, I., Turhan, K., & Ozdamar, K. (2005). Comparing classification techniques for predicting thyroid disorders. Expert Systems with Applications, 29(3), 583588.

[7] Priyadarshini, S., Mohapatra, P., & Dash, R. (2018). A comparative analysis of machine learning techniques for thyroid disease diagnosis. International Journal of Applied Engineering Research, 13(24), 16819-16825.

8.2 WEBSITE REFERENCE

[1] Kaggle. (2025). Thyroid Disease Dataset for Machine Learning Analysis. Available at: https://www.kaggle.com

[2] Scikit-Learn. (2025). Machine Learning in Python: A Comprehensive Guide to Decision Trees and KNN Classifiers. Available at: https://scikit-learn.org

[3] National Institutes of Health (NIH). (2025). Understanding Thyroid Disorders and Their Diagnosis Using AI. Available at: https://www.nih.gov

[4] UCI Machine Learning Repository. (2025). Thyroid Disease Data Set for Predictive Modeling. Available at: https://archive.ics.uci.edu/ml/datasets/Thyroid+Disease

[5] GeeksforGeeks. (2025). Implementation of KNN and Decision Tree for Medical Diagnosis. Available at: https://www.geeksforgeeks.org

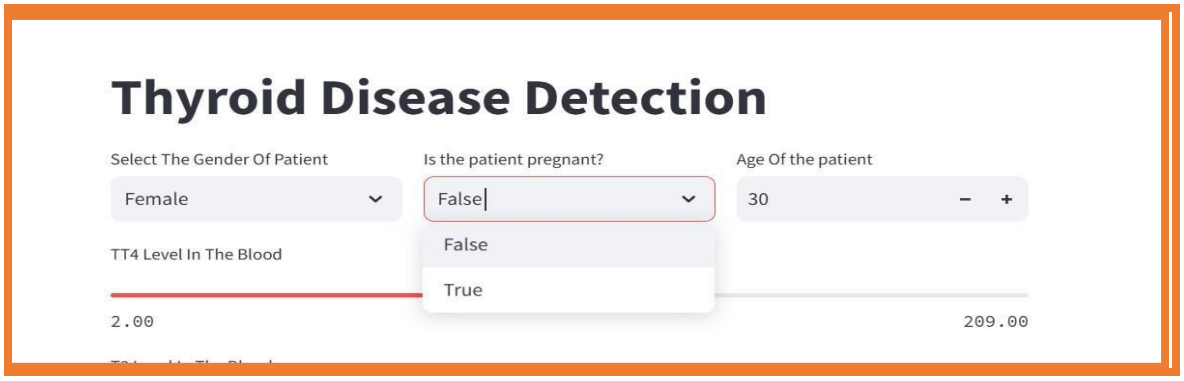
[6] Python Official Documentation. (2025). Using Pandas, NumPy, and Matplotlib for Data Preprocessing and Visualization. Available at: https://docs.python.org

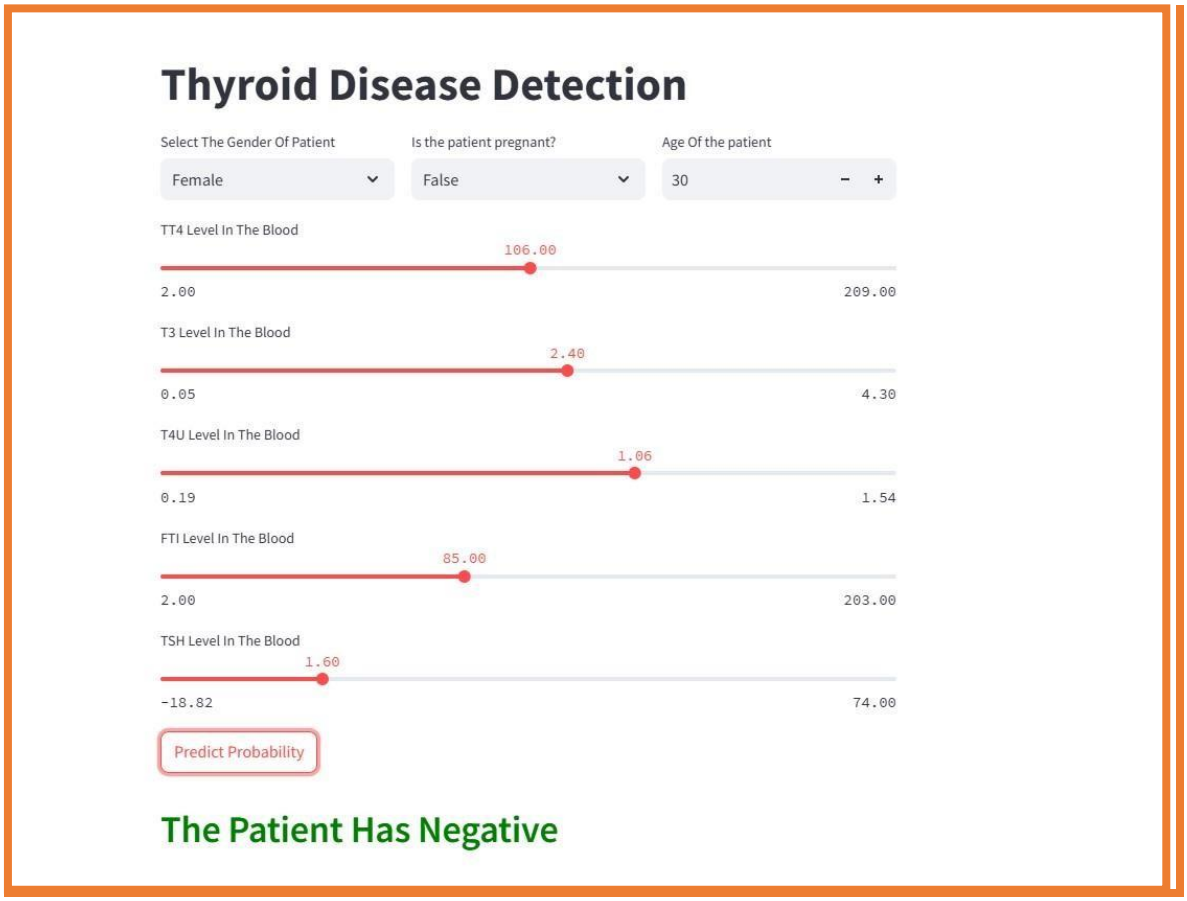
[7] Towards Data Science. (2025). Building a Machine Learning Model for Disease Prediction: A Step-by-Step Guide. Available at: https://towardsdatascience.com

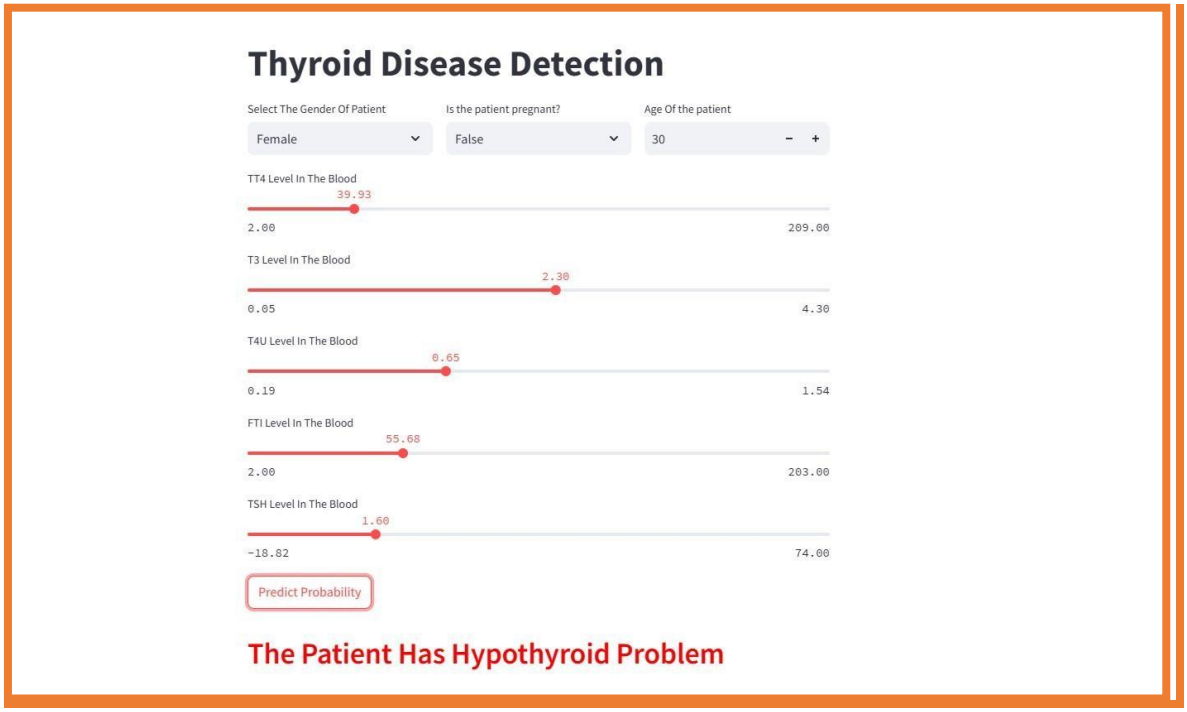
CHAPTER - 9

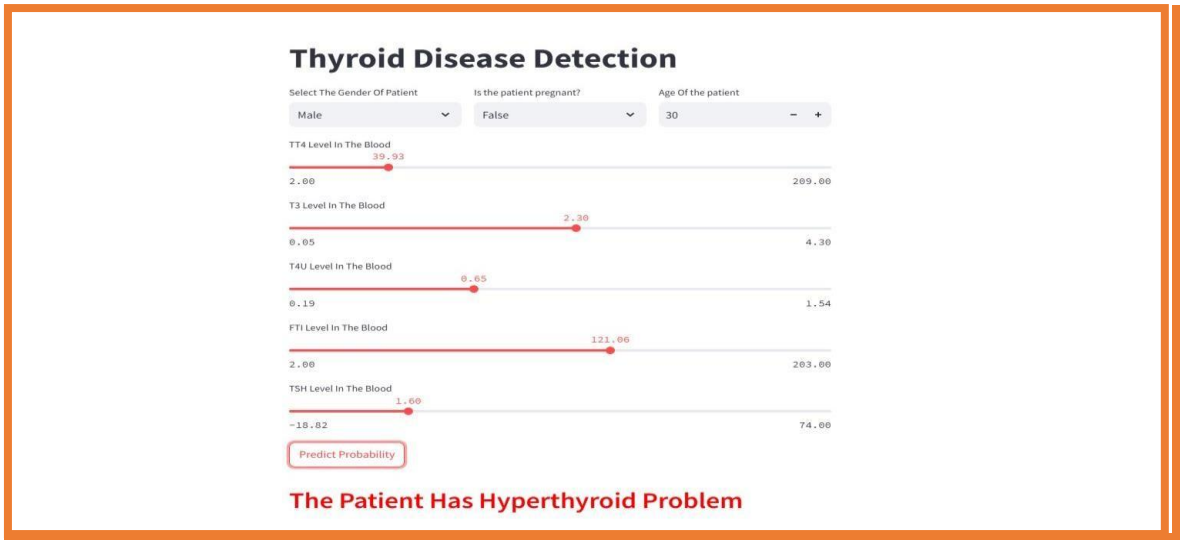
APPENDIX

a)Screenshots:




Thyroid Disease Detection
Select The Gender Of Patient
Female
Is the patient pregnant?
False
Age Of the patient
30
TT4 Level In The Blood
106.00
2.00
209.00
T3 Level In The Blood
2.40
0.05
4.30
T4U Level In The Blood
1.06
0.19
1.54
FTI Level In The Blood
85.00
2.00
203.00
TSH Level In The Blood
1.60
-18.82
74.00
Predict Probability
The Patient Has Negative





b) Sample code:

```
# Make Web Page
import pickle
```

```
import streamlit as st
import pandas as pd
from sklearn.preprocessing import LabelEncoder
import plotly.express as px
import warnings
warnings.filterwarnings("ignore")

# list
sex = ['Female', 'Male']
pregnant = ['True', 'False']

try:
    model = pickle.load(open('model.pkl', 'rb'))
except FileNotFoundError:
    st.error("Error: Model file 'model.pkl' not found. Please ensure the model file exists.")
    st.stop()
except Exception as e:
    st.error(f"Error loading the model: {e}")
    # log the exception for further investigation
    raise

# web site view
st.title('Thyroid Disease Detection')
# Use columns to place selectors side by side
col1, col2 ,col3 = st.columns(3)

# 'Select The Gender Of Patient' dropdown
with col1:
    gender_patient = st.selectbox('Select The Gender Of Patient', sorted(sex))

# 'Is the patient pregnant?' dropdown
with col2:
    # Disable and set to 'False' if gender is 'male'
    if gender_patient == 'Male':
        female_pregnant = 'False'
        st.selectbox('Is the patient pregnant?', [female_pregnant], index=0,
key='pregnant_selector')
    else:
        female_pregnant = st.selectbox('Is the patient pregnant?', sorted(pregnant))

with col3:
    Age = st.number_input('Age Of the patient', min_value=1, max_value=100, value=30)
```

```
TT4 = st.slider('TT4 Level In The Blood', min_value=2.0, max_value=209.0, value=106.0)


T3 = st.slider('T3 Level In The Blood', min_value=0.05, max_value=4.3, value=2.4)

T4U = st.slider('T4U Level In The Blood', min_value=0.19, max_value=1.54, value=1.06)

FTI = st.slider('FTI Level In The Blood', min_value=2.0, max_value=203.0, value=85.0)

TSH = st.slider('TSH Level In The Blood', min_value=-18.824586, max_value=74.0, value=1.60)



if st.button('Predict Probability'):
    input_data = pd.DataFrame({
        'age' : [Age],
        'sex': [gender_patient],
        'TT4': [TT4],
        'T3': [T3],
        'T4U': [T4U],
        'FTI': [FTI],
        'TSH': [TSH],
        'pregnant': [female_pregnant]
    })

    # Apply label encoding to 'sex' and 'pregnant' columns
    label_encoder = LabelEncoder()
    input_data['sex'] = label_encoder.fit_transform(input_data['sex'])
    input_data['pregnant'] = label_encoder.fit_transform(input_data['pregnant'])

    try:
        # Make sure to convert input_data to the same format as x_test used during training
        result_prob = model.predict(input_data)  # Assuming binary classification
        if result_prob == 0:
            st.markdown("<h2 style='color: red;'>The Patient Has Hyperthyroid Problem</h2>", unsafe_allow_html=True)
        elif result_prob == 1:
            st.markdown("<h2 style='color: red;'>The Patient Has Hypothyroid Problem</h2>", unsafe_allow_html=True)
        elif result_prob == 2:
```

st.markdown("<h2 style='color: green;'>The Patient Has Negative</h2>", unsafe_allow_html=True)

except Exception as e:
st.error(f"Error predicting Probability: {e}")
# Print the exception details
print(f"Exception: {e}")
# Raise the exception again for further investigation
raise

[2]:
```
! pip install pandas numpy matplotlib seaborn scikit-learn
```

```
Requirement already satisfied: pandas in c:\anaconda\lib\site-packages (2.2.2)
Requirement already satisfied: numpy in c:\anaconda\lib\site-packages (1.26.4)
Requirement already satisfied: matplotlib in c:\anaconda\lib\site-packages (3.9.2)
Requirement already satisfied: seaborn in c:\anaconda\lib\site-packages (0.13.2)
Requirement already satisfied: scikit-learn in c:\anaconda\lib\site-packages (1.5.1)
Requirement already satisfied: python-dateutil>=2.8.2 in c:\anaconda\lib\site-packages (from pandas) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in c:\anaconda\lib\site-packages (from pandas) (2024.1)
Requirement already satisfied: tzdata>=2022.7 in c:\anaconda\lib\site-packages (from pandas) (2023.3)
Requirement already satisfied: contourpy>=1.0.1 in c:\anaconda\lib\site-packages (from matplotlib) (1.2.0)
Requirement already satisfied: cycler>=0.10 in c:\anaconda\lib\site-packages (from matplotlib) (0.11.0)
Requirement already satisfied: fonttools>=4.22.0 in c:\anaconda\lib\site-packages (from matplotlib) (4.51.0)
Requirement already satisfied: kiwisolver>=1.3.1 in c:\anaconda\lib\site-packages (from matplotlib) (1.4.4)
Requirement already satisfied: packaging>=20.0 in c:\anaconda\lib\site-packages (from matplotlib) (24.1)
Requirement already satisfied: pillow>=8 in c:\anaconda\lib\site-packages (from matplotlib) (10.4.0)
Requirement already satisfied: pyparsing>=2.3.1 in c:\anaconda\lib\site-packages (from matplotlib) (3.1.2)
Requirement already satisfied: scipy>=1.6.0 in c:\anaconda\lib\site-packages (from scikit-learn) (1.13.1)
Requirement already satisfied: joblib>=1.2.0 in c:\anaconda\lib\site-packages (from scikit-learn) (1.4.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in c:\anaconda\lib\site-packages (from scikit-learn) (3.5.0)
Requirement already satisfied: six>=1.5 in c:\anaconda\lib\site-packages (from python-dateutil>=2.8.2->pandas) (1.16.0)
```

[3]:
```
# Import Some Library
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns

import warnings
warnings.filterwarnings('ignore')
```

[4]:
```
# Read CSV Data
```

[5]:
```
data.head(10)
```

[5]:

	age	sex	on_thyroxine	query_on_thyroxine	on_antithyroid_meds	sick	pregnant	thyroid_surgery	I131_treatment	query_hypothyroid	...	TT4	T4U_measured	T4
0	29	F	f	f	f	f	f	f	f	t	...	NaN	f	Na
1	29	F	f	f	f	f	f	f	f	f	...	128.0	f	Na
2	41	F	f	f	f	f	f	f	f	f	...	NaN	f	Na
3	36	F	f	f	f	f	f	f	f	f	...	NaN	f	Na
4	32	F	f	f	f	f	f	f	f	f	...	NaN	f	Na
5	60	F	f	f	f	f	f	f	f	f	...	NaN	f	Na
6	77	F	f	f	f	f	f	f	f	f	...	NaN	f	Na
7	28	F	f	f	f	f	f	f	f	f	...	116.0	f	Na
8	28	F	f	f	f	f	f	f	f	f	...	76.0	f	Na
9	28	F	f	f	f	f	f	f	f	f	...	83.0	f	Na

10 rows × 31 columns

[6]:
```
# number of row's and col in data set
row,col=data.shape
print("Number of Row's in Data :",row)
print("Number of Col's in Data :",col)
```

```
Number of Row's in Data : 9172
Number of Col's in Data : 31
```

[7]:
```
# data duplicate
data.duplicated().sum()
```

[7]:
```
0
```

[8]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9172 entries, 0 to 9171
Data columns (total 31 columns):
 #   Column              Non-Null Count  Dtype
```

Data Cleaning

[12]:
```
data.isnull().sum()
```

[12]:
```
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_measured            0
TSH                   842
T3_measured             0
T3                   2604
TT4_measured            0
TT4                   442
T4U_measured            0
T4U                   809
FTI_measured            0
FTI                   802
TBG_measured            0
TBG                  8823
referral_source         0
target                  0
patient_id              0
dtype: int64
```

[13]:
```
data['TBG_measured']
```

[13]:
```
0       f
1       f
2       t
3       t
4       t
       ..
```

[14]:
```
# whether TBG was measured in the blood
data['TBG_measured'].value_counts()
```

[14]:
```
TBG_measured
f    8823
t     349
Name: count, dtype: int64
```

[15]:
```
# drop TBG Column
data.drop(columns=['TBG','patient_id'],inplace=True)
```

[16]:
```
# Randomly assign mean values to the some missing missing entries in the dataset
# 2604 Is missing entries in T3 column
Random_AVG_T3 = np.random.uniform(data['T3'].mean() - data['T3'].std(), data['T3'].mean() + data['T3'].std(), 2604)
# 809 Is missing entries in T4U column
Random_AVG_T4U = np.random.uniform(data['T4U'].mean() - data['T4U'].std(), data['T4U'].mean() + data['T4U'].std(), 809)
# 842 Is missing entries in TSH column
Random_AVG_TSH = np.random.uniform(data['TSH'].mean() - data['TSH'].std(), data['TSH'].mean() + data['TSH'].std(), 842)
# 442 Is missing entries in TT4 column
Random_AVG_TT4 = np.random.uniform(data['TT4'].mean() - data['TT4'].std(), data['TT4'].mean() + data['TT4'].std(), 442)
# 802 Is missing entries in FTI column
Random_AVG_FTI = np.random.uniform(data['FTI'].mean() - data['FTI'].std(), data['FTI'].mean() + data['FTI'].std(), 802)
```

[17]:
```
# Insert Random value in dataset column T3
data['T3'][data['T3'].isnull()]=Random_AVG_T3

# Insert Random value in dataset column T4U
data['T4U'][data['T4U'].isnull()]=Random_AVG_T4U

# Insert Random value in dataset column T4U
data['TSH'][data['TSH'].isnull()]=Random_AVG_TSH

# Insert Random value in dataset column TT4
data['TT4'][data['TT4'].isnull()]=Random_AVG_TT4

# Insert Random value in dataset column FTI
data['FTI'][data['FTI'].isnull()]=Random_AVG_FTI
```

[18]:
```
# drop missing Entites of sex (Because their one pregnant column also their )
data.dropna(inplace=True)
```

[19]:
```
female,male=data['sex'].value_counts()
```

Detecting outliers

Categorical data

- **If the col is highly imbalanced for eg male 10000 and female 2 then we can eliminate female**

[21]:
```
# Handel Outliers of age column
data = data[data['age']<(data['age'].mean() + 3*data['age'].std())]
data = data[data['TSH']<(data['TSH'].mean() + 3*data['TSH'].std())]
data = data[data['T3']<(data['T3'].mean() + 3*data['T3'].std())]
data = data[data['TT4']<(data['TT4'].mean() + 3*data['TT4'].std())]
data = data[data['T4U']<(data['T4U'].mean() + 3*data['T4U'].std())]
data = data[data['FTI']<(data['FTI'].mean() + 3*data['FTI'].std())]

data = data[data['age'] <= 100]
```

```
The diagnosis consists of a string of letters indicating diagnosed conditions.

Letter  Diagnosis
------  ---------

hyperthyroid conditions:

A   hyperthyroid
B   T3 toxic
C   toxic goitre
D   secondary toxic

hypothyroid conditions:

E   hypothyroid
F   primary hypothyroid
G   compensated hypothyroid
H   secondary hypothyroid

binding protein:

I   increased binding protein
J   decreased binding protein
```

[23]:
```
data['target'].unique()
```

[23]:
```
array(['-', 'S', 'F', 'AK', 'R', 'I', 'M', 'N', 'G', 'K', 'L', 'Q', 'J',
       'O', 'LJ', 'H|K', 'GK', 'C|I', 'A', 'KJ', 'P', 'FK', 'B', 'MK',
       'GI', 'C', 'GKJ', 'OI', 'D|R', 'E'], dtype=object)
```

[24]:
```
# mapping for target variable
map = {'-':"Negative",'A':'Hyperthyroid','AK':"Hyperthyroid", 'B':"Hyperthyroid",
          'C':"Hyperthyroid", 'C|I': 'Hyperthyroid', 'D':"Hyperthyroid",
          'D|R':"Hyperthyroid", 'E': "Hypothyroid",'F': "Hypothyroid",
          'FK': "Hypothyroid", "G": "Hypothyroid", "GK": "Hypothyroid",
          "GI": "Hypothyroid", 'GKJ': 'Hypothyroid', 'H|K': 'Hypothyroid',
         }
```

[25]:
```
data['target'] = data['target'].map(map)
data.dropna(subset=['target'], inplace=True)
```

[26]:
```
data['target'].unique()
```

[26]:
```
array(['Negative', 'Hypothyroid', 'Hyperthyroid'], dtype=object)
```

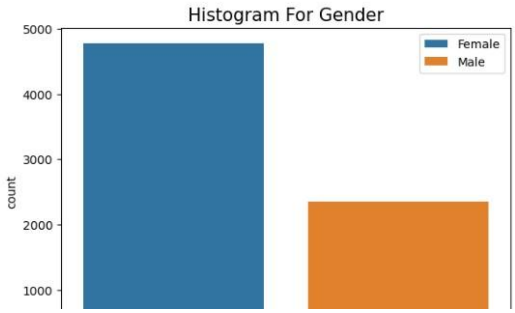
[27]:
```
data.dropna()
```

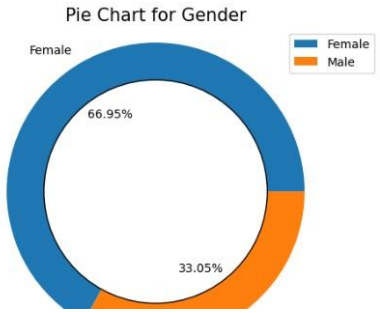
[27]:

	age	sex	on_thyroxine	query_on_thyroxine	on_antithyroid_meds	sick	pregnant	thyroid_surgery	I131_treatment	query_hypothyroid	...	T3	TT4_measure
0	29	F	f	f	f	f	f	f	f	t	...	2.741844	
1	29	F	f	f	f	f	f	f	f	f	...	1.900000	
2	41	F	f	f	f	f	f	f	f	f	...	1.575900	
3	36	F	f	f	f	f	f	f	f	f	...	2.336518	
5	60	F	f	f	f	f	f	f	f	f	...	1.258778	
...	...	...	...	...	...	...	...	...	...	...	...	...	
9166	70	F	f	f	f	f	f	f	f	f	...	1.267769	
9167	56	M	f	f	f	f	f	f	f	f	...	1.813517	
9168	22	M	f	f	f	f	f	f	f	f	...	1.594464	
9170	47	F	f	f	f	f	f	f	f	f	...	2.320294	
9171	31	M	f	f	f	f	f	f	f	t	...	2.235721	

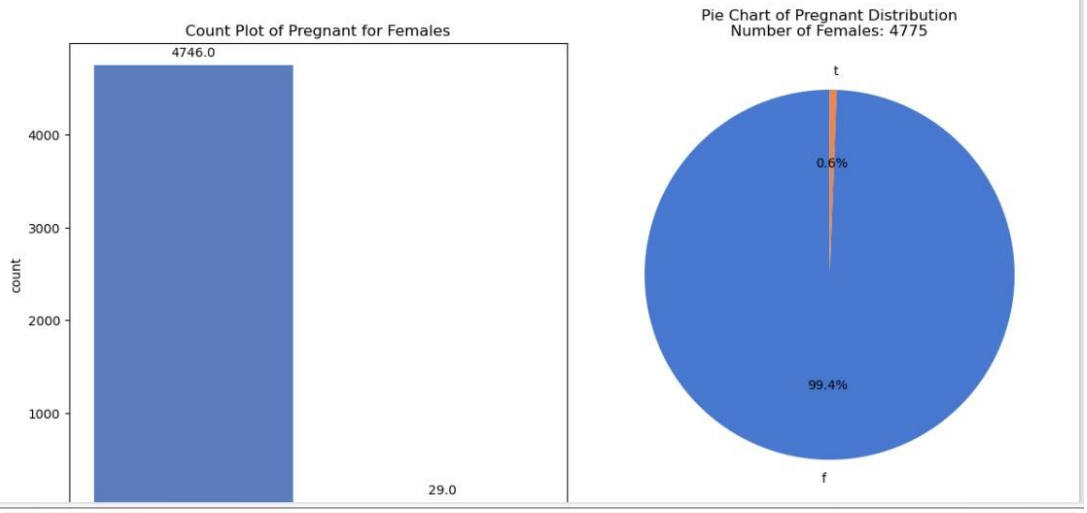
Data Visualization

[29]:
```
# Plotting For Gender

labels = ['Female', 'Male']
plt.figure(figsize=(15,5))

plt.subplot(1, 2, 1)
sns.countplot(data=data, x='sex', hue='sex', alpha=1)
plt.legend(labels)
plt.title('Histogram For Gender',size=15)

plt.subplot(1, 2, 2)
female, male = data['sex'].value_counts()
y = [female, male]
explode = [0, 0.0]
labels = ['Female', 'Male']
plt.pie(y, labels=labels, explode=explode,autopct='%.2f%%')
plt.axis('equal')
plt.legend(labels)
circle = plt.Circle(xy=(0, 0), radius=0.75, facecolor='White', edgecolor='black')
plt.gca().add_artist(circle)
plt.title('Pie Chart for Gender', size=15)
plt.show()
```

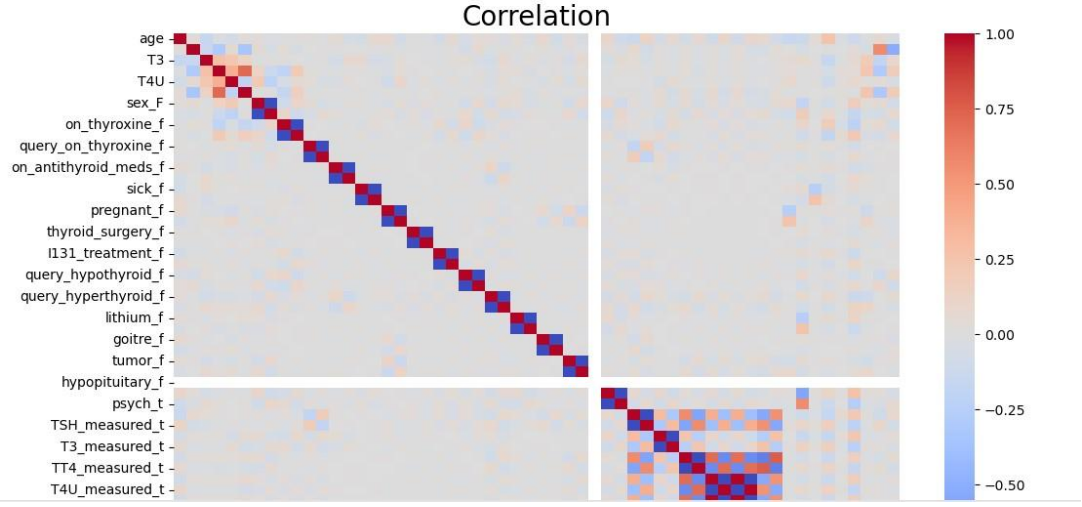


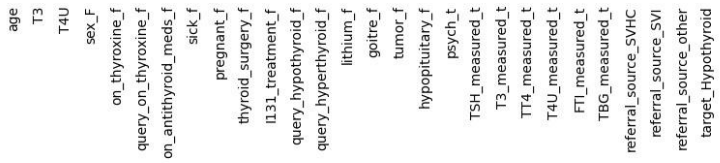


```
# Count plot on the left side
sns.countplot(x='pregnant', data=female_data, palette='muted', ax=axes[0])
axes[0].set_title('Count Plot of Pregnant for Females',color='black')
for p in axes[0].patches:
    axes[0].annotate(f'{p.get_height()}', (p.get_x() + p.get_width() / 2., p.get_height()),
        ha='center', va='center', xytext=(0, 10), textcoords='offset points', color='black')

# Pie chart on the right side
pregnant_counts = female_data['pregnant'].value_counts()
axes[1].pie(pregnant_counts, labels=pregnant_counts.index, autopct='%1.1f%%', colors=sns.color_palette('muted'), startangle=90)
axes[1].set_title('Pie Chart of Pregnant Distribution\nNumber of Females: {}'.format(len(female_data)),color='black')

plt.tight_layout()
plt.show()
```



[37]:
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

data_encoded = pd.get_dummies(data)

# Create a correlation matrix
correlation_matrix = data_encoded.corr()

# Plot the heatmap
plt.figure(figsize=(12, 8))
sns.heatmap(correlation_matrix, annot=False, cmap='coolwarm')
plt.title("Correlation", fontsize=20)
plt.show()
```





Age:

- Age has a weak negative correlation with T3 and T4U, and a weak positive correlation with TT4 and FTI. This means that as people age, their T3 and T4U levels tend to decrease, while their TT4 and FTI levels tend to increase.

Sex:

- Female sex has a negative correlation with TSH and a positive correlation with FT4. This means that women tend to have lower TSH levels and higher FT4 levels than men.

Thyroid medication:

- Being on thyroxine therapy (on_thyroxine_f) has a strong negative correlation with TSH, T3, and FT4. This is expected, as thyroxine medication suppresses the production of TSH and increases thyroid hormone levels. Being on antithyroid medication (on_antithyroid_meds_f) has a positive correlation with TSH and a negative correlation with FT4. This is also expected, as antithyroid medication blocks the production of thyroid hormones and increases TSH levels.

Other medical conditions:

- Several other medical conditions are shown to have correlations with thyroid hormone levels. For example, pregnancy (pregnant_f) is associated with higher T4 levels, while hypothyroidism (target_Hypothyroid) is associated with lower T4 levels and higher TSH levels.

Convert Categorical To Numerical Data And Drop Unnecessary Columns

Drop Unnecessary Columns

[41]:
```
selected_columns = ['age','sex', 'TT4', 'T3', 'T4U', 'FTI', 'TSH', 'pregnant', 'target']
data = data[selected_columns]
```

[42]:
```
data.head(10)
```

[42]:

	age	sex	TT4	T3	T4U	FTI	TSH	pregnant	target
0	29	F	91.397956	2.741844	0.792560	91.830036	0.300000	f	Negative
1	29	F	128.000000	1.900000	1.155390	73.061649	1.600000	f	Negative
2	41	F	100.093130	1.575900	1.056166	143.347728	-11.863002	f	Negative
3	36	F	105.131939	2.336518	0.809315	119.295679	-10.892735	f	Negative
5	60	F	97.587119	1.258778	1.083104	154.479509	-18.625233	f	Negative
6	77	F	72.260218	2.076356	1.081197	137.724958	14.044915	f	Negative
7	28	F	116.000000	2.600000	0.803180	134.342361	0.700000	f	Negative
8	28	F	76.000000	1.800000	0.984245	107.798957	1.200000	f	Negative
9	28	F	83.000000	1.700000	1.127444	118.646647	1.900000	f	Negative
10	54	F	133.000000	2.300000	1.039028	154.267294	1.900000	f	Negative

Convert Categorical data to numerical

[44]:
```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
data['target']=le.fit_transform(data['target'])
data['sex']=le.fit_transform(data['sex'])
data['pregnant']=le.fit_transform(data['pregnant'])
data.head(10)
```

[44]:

	age	sex	TT4	T3	T4U	FTI	TSH	pregnant	target
0	29	0	91.397956	2.741844	0.792560	91.830036	0.300000	0	2
1	29	0	128.000000	1.900000	1.155390	73.061649	1.600000	0	2

Training And Testing Sets For Machine Learning

[47]:
```
y = data[['target']] # Depended
x = data[['age','sex', 'TT4', 'T3', 'T4U', 'FTI', 'TSH', 'pregnant']] # independent
```

[48]:
```
y.value_counts()
```

[48]:
```
target
2         6483
1          547
0          102
Name: count, dtype: int64
```

[49]:
```
y.head(5)
```

[49]:

	target
0	2
1	2
2	2
3	2
5	2

- **Split data into Training and Testing**

[51]:
```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.3,random_state=1)
```

[52]:
```
y.value_counts()
```

[52]:
```
target
2         6483
1          547
0          102
Name: count, dtype: int64
```

[53]:
```
print('Length of Training Set :',len(x_train))
print('Length of Testing Set :',len(x_test))
```

Model Execution:

KNN (K- Neighrest Neighbour Classifier)

[55]:
```
from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier(n_neighbors=3);
knn.fit(x_train,y_train)
```

[55]:
```
KNeighborsClassifier(n_neighbors=3)
```

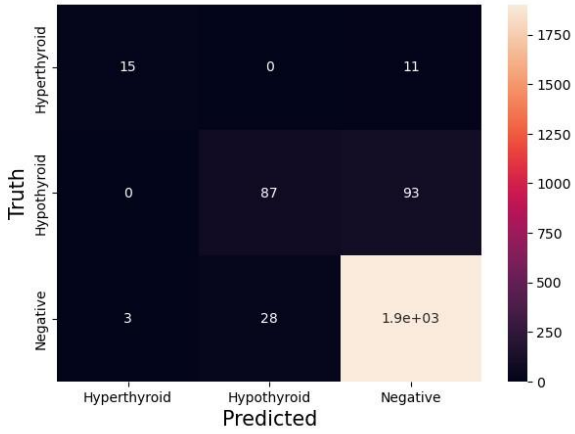
[56]:
```
knn.score(x_test,y_test)
```

[56]:
```
0.9369158878504673
```

[57]:
```
from sklearn.metrics import mean_squared_error

#Predicted On The Test Set
y_pred= knn.predict(x_test)

# Evaluate The Model
Training_score  = knn.score(x_train,y_train)
Testing_score  = knn.score(x_test,y_test)

mse = mean_squared_error(y_test,y_pred)
rmse = np.sqrt(mse)

from sklearn.metrics import r2_score

# Calculate R-squared
r2 = r2_score(y_test, y_pred)

# Calculate Adjusted R-squared
n = len(y_test)
p = x_test.shape[1]
adj_r2 = 1 - ((1 - r2) * (n - 1) / (n - p - 1))

print('Training Score :', Training_score, '%')
print('Testing Score :', Testing_score, '%')
```

[59]:
```
# Confusion Matrix
from sklearn.metrics import confusion_matrix,classification_report
cm = confusion_matrix(y_test,y_pred)
```

[60]:
```
# confusion metric visual
import seaborn as sn
plt.figure(figsize=(7,5))
cm_df = pd.DataFrame(cm,
                     index = ['Hyperthyroid','Hypothyroid','Negative'],
                     columns = ['Hyperthyroid','Hypothyroid','Negative'])
sn.heatmap(cm_df,annot=True)

plt.xlabel('Predicted',color='black',size=15)
plt.ylabel('Truth',color='black',size=15)
plt.show()
```



In this confusion matrix:

- For class 0 (Hyperthyroid)
- For class 1 (Hypothyroid)
- For class 2 (Negative)

[62]:
```
from sklearn.metrics import classification_report

# Classification Report
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.83      0.58      0.68        26
           1       0.76      0.48      0.59       180
           2       0.95      0.98      0.97      1934

    accuracy                           0.94      2140
   macro avg       0.85      0.68      0.75      2140
weighted avg       0.93      0.94      0.93      2140
```

[63]:
```
from sklearn.tree import DecisionTreeClassifier
import matplotlib.pyplot as plt
from sklearn.metrics import f1_score, precision_score, recall_score, accuracy_score

# Initialize KNN with 5 neighbors
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=3)


# Initialize classifiers
classifiers = {
    "KNN": knn,
    "Decision Tree": DecisionTreeClassifier(max_depth=3, min_samples_split=20, min_samples_leaf=5)
}

# Evaluate other classifiers
for name, clf in classifiers.items():
    clf.fit(x_train, y_train)
    y_pred = clf.predict(x_test)
    training_score = clf.score(x_train, y_train)
    testing_score = clf.score(x_test, y_test)
    f1 = f1_score(y_test, y_pred, average='weighted')
```

```python
# Evaluate other classifiers
for name, clf in classifiers.items():
    clf.fit(x_train, y_train)
    y_pred = clf.predict(x_test)
    training_score = clf.score(x_train, y_train)
    testing_score = clf.score(x_test, y_test)
    f1 = f1_score(y_test, y_pred, average='weighted')
    precision = precision_score(y_test, y_pred, average='weighted')
    recall = recall_score(y_test, y_pred, average='weighted')
    accuracy = accuracy_score(y_test, y_pred)
    print(f"{name}:")
    print("Training Score:", training_score)
    print("Testing Score:", testing_score)
    print("F1 Score:", f1)
    print("Precision:", precision)
    print("Recall:", recall)
    print("Accuracy:", accuracy)
    print()

# Accuracy comparison graph
accuracies = [clf.score(x_test, y_test) for clf in classifiers.values()]
plt.figure(figsize=(10, 6))
plt.bar(classifiers.keys(), accuracies, color='skyblue')
plt.xlabel('Classifiers', fontsize=14)
plt.ylabel('Accuracy', fontsize=14)
plt.title('Accuracy Comparison of Classifiers', fontsize=16)
plt.ylim(0.7, 1.0)
plt.xticks(rotation=45)
plt.show()
```

```
KNN:
Training Score: 0.9631410256410257
Testing Score: 0.9369158878504673
F1 Score: 0.9306762062762044
Precision: 0.9306651121724108
Recall: 0.9369158878504673
Accuracy: 0.9369158878504673

Decision Tree:
Training Score: 0.961738782051282
Testing Score: 0.9542056074766355
F1 Score: 0.9523587888931813
Precision: 0.9527140997560272
Recall: 0.9542056074766355
Accuracy: 0.9542056074766355
```

Improve Model Accuracy (By Over-sampling Dataset) :

[66]:
```python
y = data[['target']] # Depended
x = data[['age','sex', 'TT4', 'T3', 'T4U', 'FTI', 'TSH', 'pregnant']] # independent
```

[67]:
```python
y.value_counts()
```

[67]:
```
target
2         6483
1          547
0          102
Name: count, dtype: int64
```

[68]:
```python
# Handle Imbalancing (Create multiple point for balancing)
from imblearn.combine import SMOTETomek
from imblearn.over_sampling import SMOTE
```

[69]:
```python
oversample = SMOTE()
x1, y1 = oversample.fit_resample(x, y)
```

[70]:
```python
import pandas as pd

# Assuming x1 and y1 are numpy arrays
# Convert them into DataFrames
x1_df = pd.DataFrame(x1, columns=x.columns)  # Assuming x has column names
y1_df = pd.DataFrame(y1, columns=['target'])  # Assuming y doesn't have column name

# Concatenate x1 and y1 along columns
data1 = pd.concat([x1_df, y1_df], axis=1)
```

[71]:
```python
data1['target'].value_counts()
```

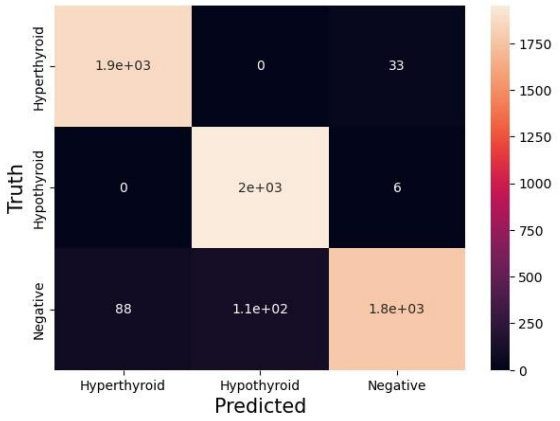
[71]:
```
target
2    6483
1    6483
0    6483
Name: count, dtype: int64
```

[72]:
```python
import plotly.express as px

# Assuming data is your DataFrame
fig = px.scatter_3d(data1, x='T4U', y='T3', z='TSH', color='target',
                    color_discrete_map={'Hypothyroid': '#4b9546', 'Hyperthyroid': '#F65366', 'Negative': '#3498db'},
```

Confusion Matrix

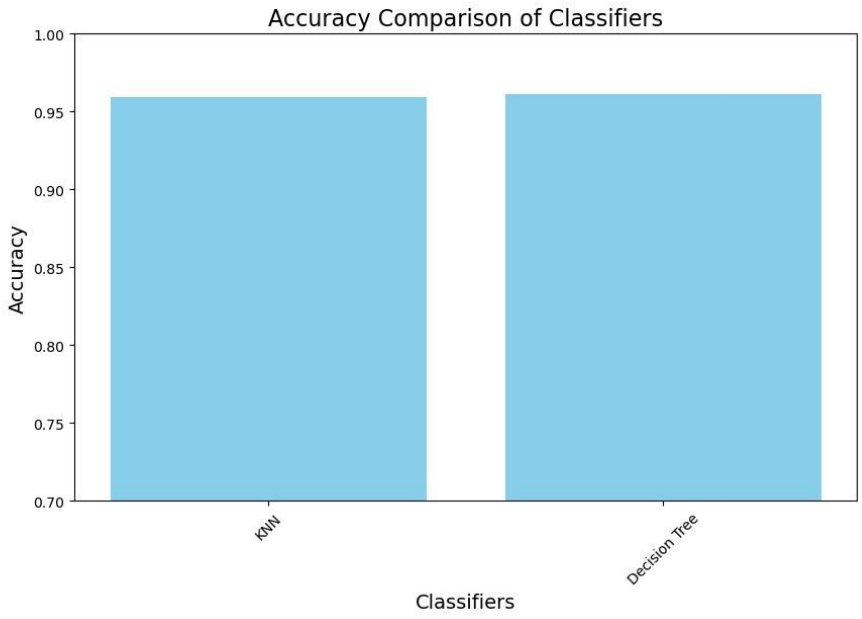
[83]:
```
# Confusion Matrix
from sklearn.metrics import confusion_matrix,classification_report
cm = confusion_matrix(y_test,y_pred)
```

[84]:
```
# confusion metric visual
import seaborn as sn
plt.figure(figsize=(7,5))
cm_df = pd.DataFrame(cm,
                     index = ['Hyperthyroid','Hypothyroid','Negative'],
                     columns = ['Hyperthyroid','Hypothyroid','Negative'])
sn.heatmap(cm_df,annot=True)

plt.xlabel('Predicted',color='black',size=15)
plt.ylabel('Truth',color='black',size=15)
plt.show()
```




```
# Accuracy comparison graph
accuracies = [clf.score(x_test, y_test) for clf in classifiers.values()]
plt.figure(figsize=(10, 6))
plt.bar(classifiers.keys(), accuracies, color='skyblue')
plt.xlabel('Classifiers', fontsize=14)
plt.ylabel('Accuracy', fontsize=14)
plt.title('Accuracy Comparison of Classifiers', fontsize=16)
plt.ylim(0.7, 1.0)
plt.xticks(rotation=45)
plt.show()
```

```
KNN:
Training Score: 0.9752460702218305
Testing Score: 0.9592116538131962
F1 Score: 0.9587601550943987
Precision: 0.9600140110698359
Recall: 0.9592116538131962
Accuracy: 0.9592116538131962

Decision Tree:
Training Score: 0.9629792860290877
Testing Score: 0.9610968294772922
F1 Score: 0.9607940344933352
Precision: 0.9614898190360399
Recall: 0.9610968294772922
Accuracy: 0.9610968294772922
```



[87]:
```
# Make Web Page
!pip install streamlit
import pickle
import streamlit as st
import pandas as pd

pickle.dump(knn,open('model.pkl','wb'))
```

```
Requirement already satisfied: streamlit in c:\anaconda\lib\site-packages (1.37.1)
Requirement already satisfied: altair<6,>=4.0 in c:\anaconda\lib\site-packages (from streamlit) (5.0.1)
Requirement already satisfied: blinker<2,>=1.0.0 in c:\anaconda\lib\site-packages (from streamlit) (1.6.2)
Requirement already satisfied: cachetools<6,>=4.0 in c:\anaconda\lib\site-packages (from streamlit) (5.3.3)
Requirement already satisfied: click<9,>=7.0 in c:\anaconda\lib\site-packages (from streamlit) (8.1.7)
Requirement already satisfied: numpy<3,>=1.20 in c:\anaconda\lib\site-packages (from streamlit) (1.26.4)
Requirement already satisfied: packaging<25,>=20 in c:\anaconda\lib\site-packages (from streamlit) (24.1)
Requirement already satisfied: pandas<3,>=1.3.0 in c:\anaconda\lib\site-packages (from streamlit) (2.2.2)
Requirement already satisfied: pillow<11,>=7.1.0 in c:\anaconda\lib\site-packages (from streamlit) (10.4.0)
Requirement already satisfied: protobuf<6,>=3.20 in c:\anaconda\lib\site-packages (from streamlit) (4.25.3)
Requirement already satisfied: pyarrow>=7.0 in c:\anaconda\lib\site-packages (from streamlit) (16.1.0)
Requirement already satisfied: requests<3,>=2.27 in c:\anaconda\lib\site-packages (from streamlit) (2.32.3)
Requirement already satisfied: rich<14,>=10.14.0 in c:\anaconda\lib\site-packages (from streamlit) (13.7.1)
Requirement already satisfied: tenacity<9,>=8.1.0 in c:\anaconda\lib\site-packages (from streamlit) (8.2.3)
Requirement already satisfied: toml<2,>=0.10.1 in c:\anaconda\lib\site-packages (from streamlit) (0.10.2)
Requirement already satisfied: typing-extensions<5,>=4.3.0 in c:\anaconda\lib\site-packages (from streamlit) (4.11.0)
Requirement already satisfied: gitpython!=3.1.19,<4,>=3.0.7 in c:\anaconda\lib\site-packages (from streamlit) (3.1.43)
Requirement already satisfied: pydeck<1,>=0.8.0b4 in c:\anaconda\lib\site-packages (from streamlit) (0.8.0)
Requirement already satisfied: tornado<7,>=6.0.3 in c:\anaconda\lib\site-packages (from streamlit) (6.4.1)
Requirement already satisfied: watchdog<5,>=2.1.5 in c:\anaconda\lib\site-packages (from streamlit) (4.0.1)
Requirement already satisfied: jinja2 in c:\anaconda\lib\site-packages (from altair<6,>=4.0->streamlit) (3.1.4)
Requirement already satisfied: jsonschema>=3.0 in c:\anaconda\lib\site-packages (from altair<6,>=4.0->streamlit) (4.23.0)
Requirement already satisfied: toolz in c:\anaconda\lib\site-packages (from altair<6,>=4.0->streamlit) (0.12.0)
Requirement already satisfied: colorama in c:\anaconda\lib\site-packages (from click<9,>=7.0->streamlit) (0.4.6)
Requirement already satisfied: gitdb<5,>=4.0.1 in c:\anaconda\lib\site-packages (from gitpython!=3.1.19,<4,>=3.0.7->streamlit) (4.0.7)
Requirement already satisfied: python-dateutil>=2.8.2 in c:\anaconda\lib\site-packages (from pandas<3,>=1.3.0->streamlit) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in c:\anaconda\lib\site-packages (from pandas<3,>=1.3.0->streamlit) (2024.1)
Requirement already satisfied: tzdata>=2022.7 in c:\anaconda\lib\site-packages (from pandas<3,>=1.3.0->streamlit) (2023.3)
Requirement already satisfied: charset-normalizer<4,>=2 in c:\anaconda\lib\site-packages (from requests<3,>=2.27->streamlit) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in c:\anaconda\lib\site-packages (from requests<3,>=2.27->streamlit) (3.7)
Requirement already satisfied: urllib3<3,>=1.21.1 in c:\anaconda\lib\site-packages (from requests<3,>=2.27->streamlit) (2.2.3)
Requirement already satisfied: certifi>=2017.4.17 in c:\anaconda\lib\site-packages (from requests<3,>=2.27->streamlit) (2025.1.31)
Requirement already satisfied: markdown-it-py>=2.2.0 in c:\anaconda\lib\site-packages (from rich<14,>=10.14.0->streamlit) (2.2.0)
Requirement already satisfied: pygments<3.0.0,>=2.13.0 in c:\anaconda\lib\site-packages (from rich<14,>=10.14.0->streamlit) (2.15.1)
Requirement already satisfied: smmap<5,>=3.0.1 in c:\anaconda\lib\site-packages (from gitdb<5,>=4.0.1->gitpython!=3.1.19,<4,>=3.0.7->streamlit) (4.0.0)
Requirement already satisfied: MarkupSafe>=2.0 in c:\anaconda\lib\site-packages (from jinja2->altair<6,>=4.0->streamlit) (2.1.3)
Requirement already satisfied: attrs>=22.2.0 in c:\anaconda\lib\site-packages (from jsonschema>=3.0->altair<6,>=4.0->streamlit) (23.1.0)
Requirement already satisfied: jsonschema-specifications>=2023.03.6 in c:\anaconda\lib\site-packages (from jsonschema>=3.0->altair<6,>=4.0->streamlit)
```

```
Requirement already satisfied: urllib3<3,>=1.21.1 in c:\anaconda\lib\site-packages (from requests<3,>=2.27->streamlit) (2.2.3)
Requirement already satisfied: certifi>=2017.4.17 in c:\anaconda\lib\site-packages (from requests<3,>=2.27->streamlit) (2025.1.31)
Requirement already satisfied: markdown-it-py>=2.2.0 in c:\anaconda\lib\site-packages (from rich<14,>=10.14.0->streamlit) (2.2.0)
Requirement already satisfied: pygments<3.0.0,>=2.13.0 in c:\anaconda\lib\site-packages (from rich<14,>=10.14.0->streamlit) (2.15.1)
Requirement already satisfied: smmap<5,>=3.0.1 in c:\anaconda\lib\site-packages (from gitdb<5,>=4.0.1->gitpython!=3.1.19,<4,>=3.0.7->streamlit) (4.0.
Requirement already satisfied: MarkupSafe>=2.0 in c:\anaconda\lib\site-packages (from jinja2->altair<6,>=4.0->streamlit) (2.1.3)
Requirement already satisfied: attrs>=22.2.0 in c:\anaconda\lib\site-packages (from jsonschema>=3.0->altair<6,>=4.0->streamlit) (23.1.0)
Requirement already satisfied: jsonschema-specifications>=2023.03.6 in c:\anaconda\lib\site-packages (from jsonschema>=3.0->altair<6,>=4.0->streamlit
(2023.7.1)
Requirement already satisfied: referencing>=0.28.4 in c:\anaconda\lib\site-packages (from jsonschema>=3.0->altair<6,>=4.0->streamlit) (0.30.2)
Requirement already satisfied: rpds-py>=0.7.1 in c:\anaconda\lib\site-packages (from jsonschema>=3.0->altair<6,>=4.0->streamlit) (0.10.6)
Requirement already satisfied: mdurl~=0.1 in c:\anaconda\lib\site-packages (from markdown-it-py>=2.2.0->rich<14,>=10.14.0->streamlit) (0.1.0)
Requirement already satisfied: six>=1.5 in c:\anaconda\lib\site-packages (from python-dateutil>=2.8.2->pandas<3,>=1.3.0->streamlit) (1.16.0)
```

[88]:
```
data.head(2)
```

[88]:

	age	sex	TT4	T3	T4U	FTI	TSH	pregnant	target
0	29	0	91.397956	2.741844	0.79256	91.830036	0.3	0	2
1	29	0	128.000000	1.900000	1.15539	73.061649	1.6	0	2

[89]:
```
def dis_prediction(sex,pregnant,TT4,T3,T4U,FTI,TSH):
    result =knn.predict([[sex,pregnant,TT4,T3,T4U,FTI,TSH]])
    return result
```

Test 1

[91]:
```
knn = pickle.load(open('model.pkl','rb'))
print(knn.predict([[63,1,48.0,2.84,1.0,47.0,65.0,1]]))
```

```
[1]
```