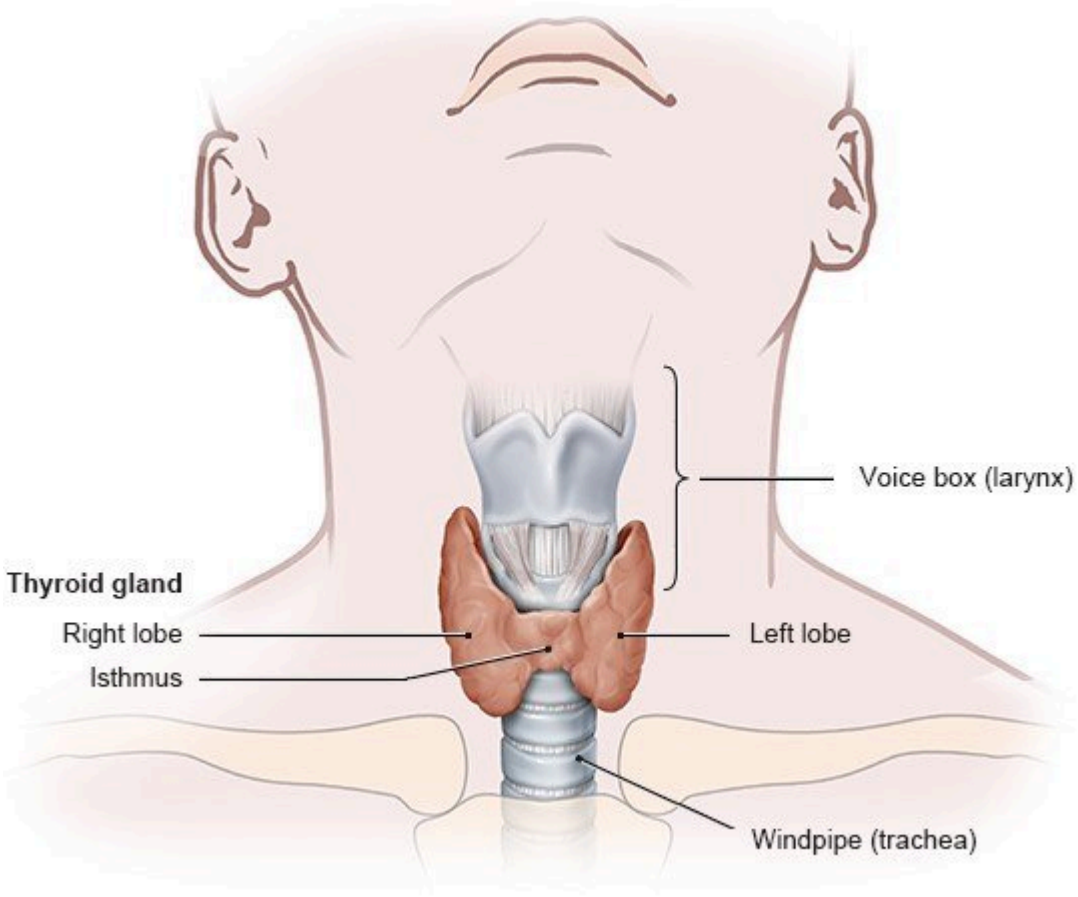
Thyroid Disease Detection



By :

- Muhammad Arslan Khalid
- Zainab Nadeem

Objective Statement:

The primary goal of this project is to employ machine learning techniques to improve the accuracy and efficiency of thyroid disorder detection. By utilizing advanced algorithms and data analytics, the aim is to develop a robust system capable of analyzing diverse medical data, including thyroid function tests and imaging results. The main focus is on creating a machine learning-based solution that can aid healthcare professionals in the early and accurate identification of thyroid disorders, leading to timely interventions and enhanced patient outcomes.

Thyroid Detection Dataset Overview:

The Thyroid Detection dataset, obtained from the UCI Machine Learning Repository and accessible on [Kaggle]., consists of **9172 observations** with **31 attributes**. The dataset is

intended for the development and evaluation of machine learning models for detecting thyroid disorders. Each observation represents a patient, and the attributes include demographic information, medical history, laboratory test results, and the target variable indicating the presence of hyperthyroidism.

Key Attributes:

1. Demographic Information:

- **Age:** Age of the patient (integer).
- **Sex:** Gender identification of the patient (string).

2. Medical History:

- **On_thyroxine:** Boolean indicating whether the patient is on thyroxine.
- **Query_on_thyroxine:** Boolean indicating queries regarding thyroxine usage.
- **On_antithyroid_meds:** Boolean indicating whether the patient is on antithyroid medications.
- **Sick:** Boolean indicating whether the patient is sick.
- **Pregnant:** Boolean indicating whether the patient is pregnant.
- **Thyroid_surgery:** Boolean indicating whether the patient has undergone thyroid surgery.
- **I131_treatment:** Boolean indicating whether the patient is undergoing I131 treatment.
- **Query_hypothyroid:** Boolean indicating the patient's belief of having hypothyroidism.
- **Query_hyperthyroid:** Boolean indicating the patient's belief of having hyperthyroidism.
- **Lithium:** Boolean indicating whether the patient uses lithium.
- **Goitre:** Boolean indicating whether the patient has goitre.
- **Tumor:** Boolean indicating whether the patient has a tumor.
- **Hypopituitary:** Float value indicating a condition related to the hyperpituitary gland.
- **Psych:** Boolean indicating a psychological condition.

3. Laboratory Test Results:

- **TSH_measured:** Boolean indicating whether TSH was measured.
- **TSH:** Float value representing the TSH level in the blood.
- **T3_measured:** Boolean indicating whether T3 was measured.
- **T3:** Float value representing the T3 level in the blood.
- **TT4_measured:** Boolean indicating whether TT4 was measured.
- **TT4:** Float value representing the TT4 level in the blood.
- **T4U_measured:** Boolean indicating whether T4U was measured.
- **T4U:** Float value representing the T4U level in the blood.
- **FTI_measured:** Boolean indicating whether FTI was measured.
- **FTI:** Float value representing the FTI level in the blood.

- **TBG_measured:** Boolean indicating whether TBG was measured.
- **TBG:** Float value representing the TBG level in the blood.

4. Other Attributes:

- **Referral_source:** String indicating the source of patient referral.
- **Target:** String indicating the medical diagnosis of hyperthyroidism.
- **Patient_id:** String representing a unique identifier for each patient.

Import Libraries:

In [ ]:
```
import pandas as pd #Pandas for data manipulation and analysis.
import numpy as np  #NumPy for numerical operations.
import matplotlib.pyplot as plt #Matplotlib for data visualization.
%matplotlib inline
import seaborn as sns #Seaborn for statistical data visualization.
import warnings #Warnings to suppress any warnings generated during code executi
warnings.filterwarnings('ignore')
```

In [ ]:
```
# Read/Import Dataset
data = pd.read_csv("Thyroid.csv")
```

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

Out[ ]:

	age	sex	on_thyroxine	query_on_thyroxine	on_antithyroid_meds	sick	pregnant	th
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	60	F	f	f	f	f	f	
6	77	F	f	f	f	f	f	
7	28	F	f	f	f	f	f	
8	28	F	f	f	f	f	f	
9	28	F	f	f	f	f	f	

10 rows × 31 columns

Summary of Data:

The Thyroid Detection dataset comprises a comprehensive collection of 9172 observations, each representing an individual patient. Sourced from the UCI Machine Learning Repository and provided on Kaggle, this dataset is specifically curated for the

development and evaluation of machine learning models focused on thyroid disorder detection, with a primary emphasis on hyperthyroidism.

Key Characteristics:

1. **Attributes Variety:**
   - The dataset encompasses a wide array of attributes, ranging from demographic details (age, sex) to medical history indicators (medication usage, surgeries, pregnancy) and crucial laboratory test results (TSH, T3, TT4, T4U, FTI).
2. **Boolean and Numeric Variables:**
   - Boolean variables are used to represent binary conditions (e.g., whether a patient is on medication, has undergone surgery, or if a specific test was measured).
   - Numeric variables include float values representing laboratory test results, providing quantitative insights into the patient's thyroid function.
3. **Target Variable:**
   - The dataset includes a target variable, "Target," indicating the medical diagnosis of hyperthyroidism. This binary classification is crucial for training machine learning models to distinguish between individuals with and without hyperthyroidism.
4. **Patient Identifier:**
   - Each observation is associated with a unique patient identifier, enabling individualized tracking and analysis.
5. **Referral Source:**
   - The "Referral_source" attribute indicates the source from which the patient was referred, providing additional contextual information.

In [ ]:
```
#  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
```

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

Out[ ]:
```
0
```

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

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9172 entries, 0 to 9171
Data columns (total 31 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   age                   9172 non-null   int64
 1   sex                   8865 non-null   object
 2   on_thyroxine          9172 non-null   object
 3   query_on_thyroxine    9172 non-null   object
 4   on_antithyroid_meds   9172 non-null   object
 5   sick                  9172 non-null   object
 6   pregnant              9172 non-null   object
 7   thyroid_surgery       9172 non-null   object
 8   I131_treatment        9172 non-null   object
 9   query_hypothyroid     9172 non-null   object
 10  query_hyperthyroid    9172 non-null   object
 11  lithium               9172 non-null   object
 12  goitre                9172 non-null   object
 13  tumor                 9172 non-null   object
 14  hypopituitary         9172 non-null   object
 15  psych                 9172 non-null   object
 16  TSH_measured          9172 non-null   object
 17  TSH                   8330 non-null   float64
 18  T3_measured           9172 non-null   object
 19  T3                    6568 non-null   float64
 20  TT4_measured          9172 non-null   object
 21  TT4                   8730 non-null   float64
 22  T4U_measured          9172 non-null   object
 23  T4U                   8363 non-null   float64
 24  FTI_measured          9172 non-null   object
 25  FTI                   8370 non-null   float64
 26  TBG_measured          9172 non-null   object
 27  TBG                   349 non-null    float64
 28  referral_source       9172 non-null   object
 29  target                9172 non-null   object
 30  patient_id            9172 non-null   int64
dtypes: float64(6), int64(2), object(23)
memory usage: 2.2+ MB
```

In [ ]:
```
data.count()
```

Out[ ]:
```
age                    9172
sex                    8865
on_thyroxine           9172
query_on_thyroxine     9172
on_antithyroid_meds    9172
sick                   9172
pregnant               9172
thyroid_surgery        9172
I131_treatment         9172
query_hypothyroid      9172
query_hyperthyroid     9172
lithium                9172
goitre                 9172
tumor                  9172
hypopituitary          9172
psych                  9172
TSH_measured           9172
TSH                    8330
T3_measured            9172
T3                     6568
TT4_measured           9172
TT4                    8730
T4U_measured           9172
T4U                    8363
FTI_measured           9172
FTI                    8370
TBG_measured           9172
TBG                     349
referral_source        9172
target                 9172
patient_id             9172
dtype: int64
```

In [ ]:
```
data.describe()
```

Out[ ]:

	age	TSH	T3	TT4	T4U	FTI
count	9172.000000	8330.000000	6568.000000	8730.000000	8363.000000	8370.000000
mean	73.555822	5.218403	1.970629	108.700305	0.976056	113.640746
std	1183.976718	24.184006	0.887579	37.522670	0.200360	41.551650
min	1.000000	0.005000	0.050000	2.000000	0.170000	1.400000
25%	37.000000	0.460000	1.500000	87.000000	0.860000	93.000000
50%	55.000000	1.400000	1.900000	104.000000	0.960000	109.000000
75%	68.000000	2.700000	2.300000	126.000000	1.065000	128.000000
max	65526.000000	530.000000	18.000000	600.000000	2.330000	881.000000

Data Cleaning:

The Thyroid Detection dataset exhibits missing values in critical features such as 'sex,' 'TSH,' 'T3,' 'TT4,' 'T4U,' 'FTI,' and 'TBG.' To address this, imputation techniques were

applied:

- **Missing 'sex' Values:** Missing 'sex' values were replaced with the mode of the 'sex' column.
- **Numeric Values:** Missing numeric values ('TSH,' 'T3,' 'TT4,' 'T4U,' 'FTI') were replaced with their respective medians.

Due to an extensive number of missing values in the 'TBG' column (8823 out of 9172), this column was dropped. The resulting dataset is now cleaned, allowing for robust analysis and machine learning model development, ensuring accurate detection of thyroid disorders.

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

Out[ ]:
```
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
```

In [ ]:
```
data['TBG_measured']
```

Out[ ]:
```
0       f
1       f
2       t
3       t
4       t
       ..
9167    f
9168    f
9169    f
9170    f
9171    f
Name: TBG_measured, Length: 9172, dtype: object
```

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

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

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

In [ ]:
```
# 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

# 809 Is missing entries in T4U column
Random_AVG_T4U = np.random.uniform(data['T4U'].mean() - data['T4U'].std(), data[

# 842 Is missing entries in TSH column
Random_AVG_TSH = np.random.uniform(data['TSH'].mean() - data['TSH'].std(), data[
```

In [ ]:
```
# 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
```

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

In [ ]:
```
# Male & Female Count
female,male=data['sex'].value_counts()

print('Total Male   : {}'.format(male))
print('Total Female : {}'.format(female))
```

```
Total Male   : 2593
Total Female : 5496
```

In [ ]:
```
# number of row's and col in data set after drop
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 : 8089
Number of Col's in Data : 29
```

In [ ]:
```
# Different refferral sources
data['referral_source'].value_counts()
```

Out[ ]:
```
referral_source
other    4625
SVI      2271
SVHC      899
STMW      226
SVHD       65
WEST        3
Name: count, dtype: int64
```

Some Bool Column in dataset:

- **T : True**
- **F : False**

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

Out[ ]:
```
TBG_measured
f    8059
t      30
Name: count, dtype: int64
```

In [ ]:
```
# on_thyroxine - whether patient is on thyroxine
data['on_thyroxine'].value_counts()
```

Out[ ]:
```
on_thyroxine
f    6942
t    1147
Name: count, dtype: int64
```

In [ ]:
```
# query on thyroxine - *whether patient is on thyroxine
data['query_on_thyroxine'].value_counts()
```

Out[ ]:
```
query_on_thyroxine
f    7940
t     149
Name: count, dtype: int64
```

In [ ]:
```
# on antithyroid meds - whether patient is on antithyroid meds
data['on_antithyroid_meds'].value_counts()
```

Out[ ]:
```
on_antithyroid_meds
f    7984
t     105
Name: count, dtype: int64
```

In [ ]:
```
# sick - whether patient is sick
data['sick'].value_counts()
```

Out[ ]:
```
sick
f    7781
t     308
Name: count, dtype: int64
```

In [ ]:
```
# pregnant - whether patient is pregnant
data['pregnant'].value_counts()
```

Out[ ]:
```
pregnant
f    7989
t     100
Name: count, dtype: int64
```

In [ ]:
```
# thyroid_surgery - whether patient has undergone thyroid surgery
data['thyroid_surgery'].value_counts()
```

Out[ ]:
```
thyroid_surgery
f    7967
t     122
Name: count, dtype: int64
```

In [ ]:
```
# I131_treatment - whether patient is undergoing I131 treatment
data['I131_treatment'].value_counts()
```

Out[ ]:
```
I131_treatment
f    7935
t     154
Name: count, dtype: int64
```

In [ ]:
```
# query_hypothyroid - whether patient believes they have hypothyroid
data['query_hypothyroid'].value_counts()
```

Out[ ]:
```
query_hypothyroid
f    7531
t     558
Name: count, dtype: int64
```

In [ ]:
```
# query_hyperthyroid - whether patient believes they have hyperthyroid
data['query_hyperthyroid'].value_counts()
```

Out[ ]:
```
query_hyperthyroid
f    7528
t     561
Name: count, dtype: int64
```

In [ ]:
```
# lithium - whether patient * lithium
data['lithium'].value_counts()
```

Out[ ]:
```
lithium
f    8000
t      89
Name: count, dtype: int64
```

In [ ]:
```
# goitre - whether patient has goitre
data['goitre'].value_counts()
```

Out[ ]:
```
goitre
f    8009
t      80
Name: count, dtype: int64
```

In [ ]:
```
# tumor - whether patient has tumor
data['tumor'].value_counts()
```

Out[ ]:
```
tumor
f    7893
t     196
Name: count, dtype: int64
```

In [ ]:
```
# psych - whether patient * psych
data['psych'].value_counts()
```

Out[ ]:
```
psych
f    7694
t     395
Name: count, dtype: int64
```

In [ ]:
```
# T3_measured - whether T3 was measured in the blood
data['T3_measured'].value_counts()
```

Out[ ]:
```
T3_measured
t    5931
f    2158
Name: count, dtype: int64
```

In [ ]:
```
# TT4_measured - whether TT4 was measured in the blood
data['TT4_measured'].value_counts()
```

Out[ ]:
```
TT4_measured
t    8089
Name: count, dtype: int64
```

In [ ]:
```
# T4U_measured - whether T4U was measured in the blood
data['T4U_measured'].value_counts()
```

Out[ ]:
```
T4U_measured
t    8086
f       3
Name: count, dtype: int64
```

In [ ]:
```
# FTI_measured - whether FTI was measured in the blood
data['FTI_measured'].value_counts()
```

Out[ ]:
```
FTI_measured
t    8089
Name: count, dtype: int64
```

In [ ]:
```
# Transpose
# Transposing the summary statistics for categorical data, improves readability
data.describe(include='O').T
```

Out[ ]:

	count	unique	top	freq
sex	8089	2	F	5496
on_thyroxine	8089	2	f	6942
query_on_thyroxine	8089	2	f	7940
on_antithyroid_meds	8089	2	f	7984
sick	8089	2	f	7781
pregnant	8089	2	f	7989
thyroid_surgery	8089	2	f	7967
I131_treatment	8089	2	f	7935
query_hypothyroid	8089	2	f	7531
query_hyperthyroid	8089	2	f	7528
lithium	8089	2	f	8000
goitre	8089	2	f	8009
tumor	8089	2	f	7893
hypopituitary	8089	2	f	8087
psych	8089	2	f	7694
TSH_measured	8089	2	t	7655
T3_measured	8089	2	t	5931
TT4_measured	8089	1	t	8089
T4U_measured	8089	2	t	8086
FTI_measured	8089	1	t	8089
TBG_measured	8089	2	f	8059
referral_source	8089	6	other	4625
target	8089	30	-	5957

In [ ]:
```
data['TSH'].head(200)
```

Out[ ]:
```
18     68.000000
19      1.500000
20     -3.874467
21      1.200000
22      5.900000
          ...
241     0.050000
242     0.600000
243     1.200000
244     0.050000
245    11.000000
Name: TSH, Length: 200, dtype: float64
```

An anomaly is observed in the age data, with a **maximum value of 65526 years and a minimum value of 1 year**. These extreme values indicate the possible presence of outliers in the dataset, suggesting the need for further investigation or data cleaning to address these anomalies.

Detecting outliers:

Numerical Data:

- if the data is following normal distribution, anything beyond 3SD mean 3SD can be considered as an outlier
- if the data does not follow normal distribution, using boxplot we can eliminate points beyond Q1 1.5 IQR and Q3 + 1.5 IQR

Categorical data:

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

In [ ]:
```
# Handel Outliers of age column
data = data[data['age']<(data['age'].mean() + 3*data['age'].std())]

# Handle Outlier of TSH, T3, TT4, T4U, FTI
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 is represented by a string of letters indicating specific conditions. A diagnosis of "-" implies no condition needing comment. For diagnoses of the form "X|Y", it suggests compatibility with X but a higher likelihood of Y.

The conditions are categorized into various 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

In [ ]:
```
# returns the unique values present in the target column of the data DataFrame
data['target'].unique()
```

Out[ ]:
```
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)
```

In [ ]:
```
# mapping for target variable - Reduces the number of unique classes, making the
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',
          }
```

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

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

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

In [ ]:
```
# Minimal Missing Data, Non-Essential Data , Quick Preprocessing
data.dropna()
```

Out[ ]:

	age	sex	on_thyroxine	query_on_thyroxine	on_antithyroid_meds	sick	pregnant
18	63	F	t	f	f	t	f
19	36	F	f	f	f	f	f
20	40	F	f	t	f	f	f
21	40	F	f	f	f	f	f
22	40	F	f	f	f	f	f
...	...	...	...	...	...	...	...
9166	70	F	f	f	f	f	f
9167	56	M	f	f	f	f	f
9168	22	M	f	f	f	f	f
9170	47	F	f	f	f	f	f
9171	31	M	f	f	f	f	f

6501 rows × 29 columns

Data Visualization :

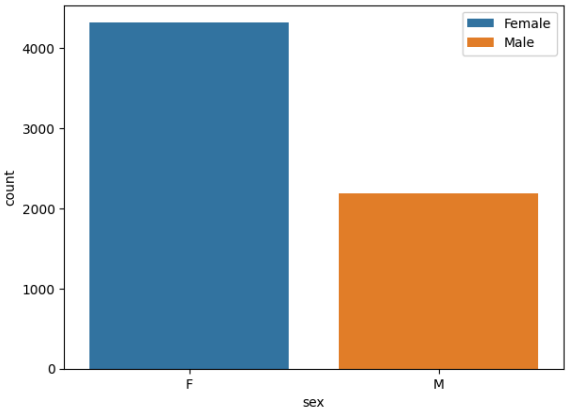
Visualizing the Thyroid Detection dataset offers valuable insights into various aspects. The distribution of patient ages reveals a diverse demographic composition, while examining gender distribution provides insights into the representation of males and females. Exploring the target variable sheds light on the prevalence of hyperthyroidism within the dataset, guiding further analysis. Additionally, visualizing laboratory test results and binary feature distributions provides a deeper understanding of thyroid-related conditions. These visualizations collectively enhance data comprehension and inform subsequent machine learning model development.

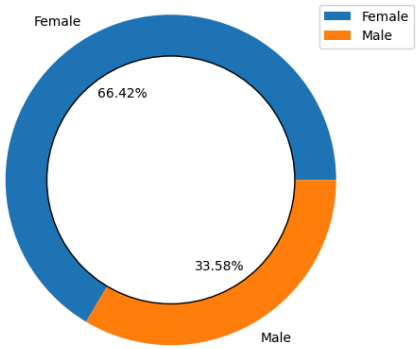
Plotting For Gender:

In [ ]:

```
# count plot (histogram) - show the frequency distribution of the 'sex'
# Particularly useful for identifying any potential imbalance in the gender dist

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)

# Pie Chart -  show the proportion of each gender category
# Illustrates the percentage of male and female instances.

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.show()
```

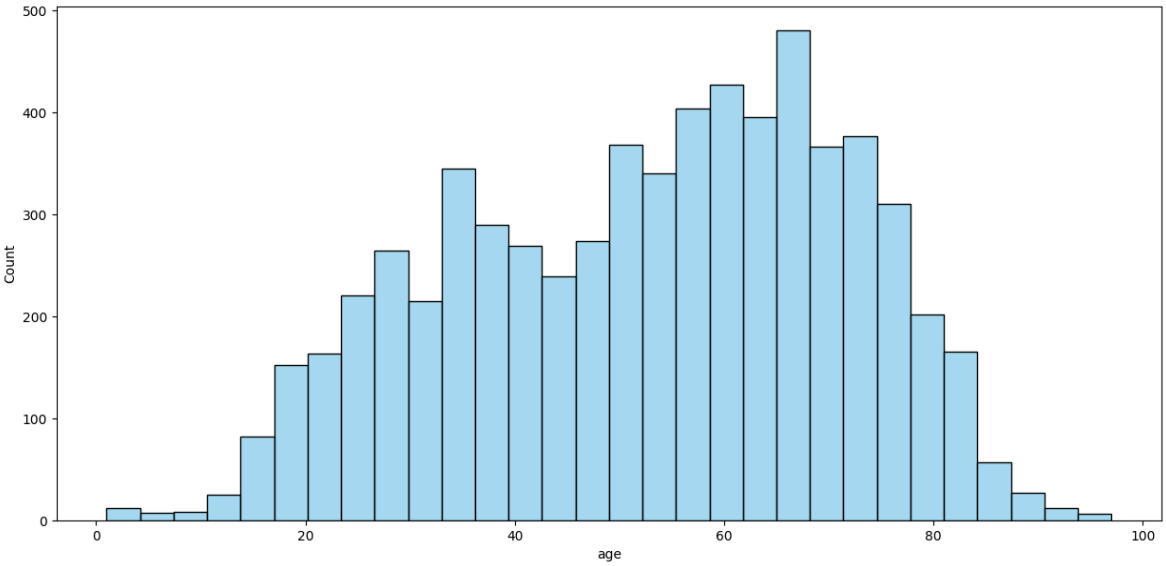




- The histogram and pie chart reveal that **females** constitute **66.42%** of the dataset, whereas **males** make up **33.58%**. This indicates a higher representation of females compared to males within the dataset.

Plotting for Age:

In [ ]:
```
plt.figure(figsize=(15,7))
sns.histplot(data['age'],kde=False,color='skyblue')
plt.show()
```



- The histogram demonstrates a concentration of individuals between the **ages of 20 and 80**, with a peak occurring around **58 to 62 years**. There is a noticeable decline in the frequency of individuals within the **1 to 15 age** range, indicating lower representation. The x-axis represents age groups, while the y-axis indicates the corresponding number of individuals. Overall, the distribution appears relatively

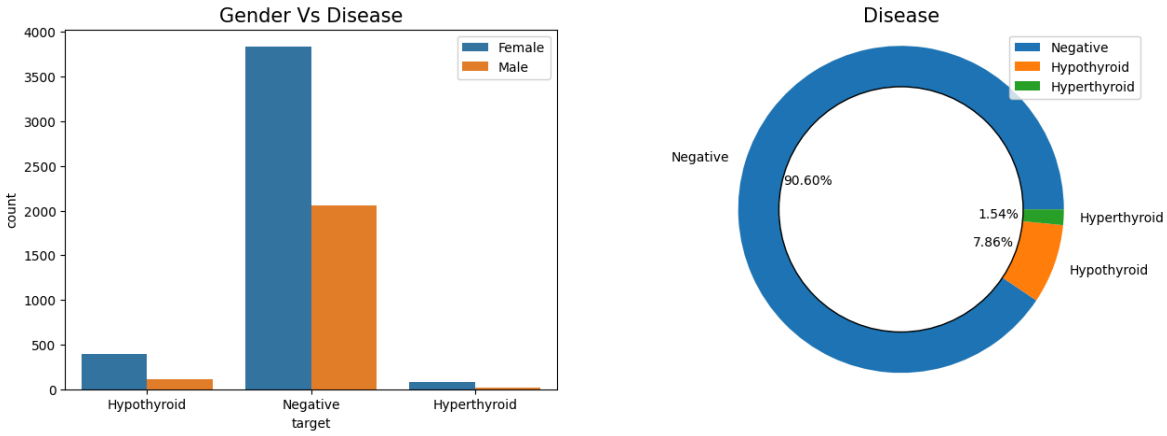
even, with a peak in the **21-30 age** range and fewer individuals in both younger and older age categories.

In [ ]:
```
data['target'].value_counts()
```

Out[ ]:
```
target
Negative        5890
Hypothyroid      511
Hyperthyroid     100
Name: count, dtype: int64
```

Plotting for Disease & Gender vs Disease:

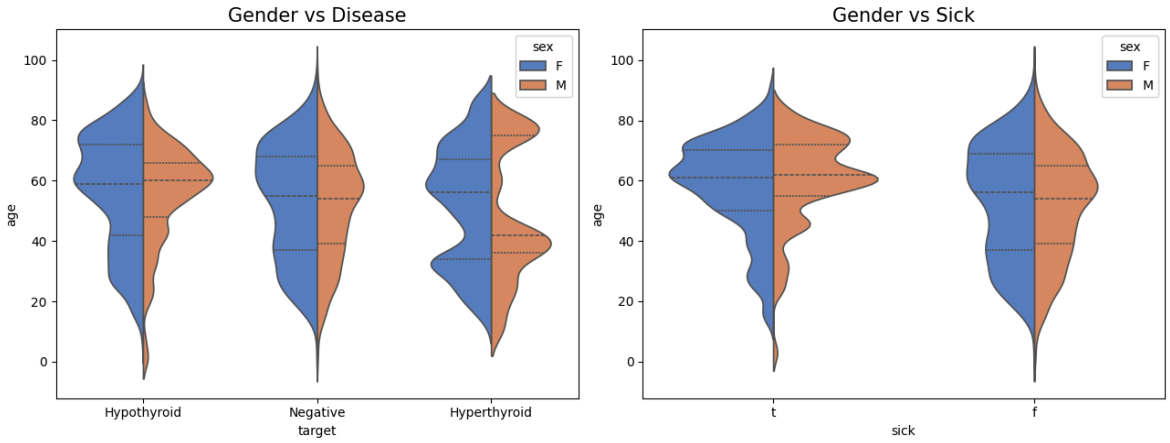
In [ ]:
```
labels = ['Female','Male']
plt.figure(figsize=(15,5))

# Gender Vs Disease
plt.subplot(1, 2, 1)
sns.countplot(data=data, x='target', hue='sex', alpha=1)
plt.legend(labels)
plt.title('Gender Vs Disease',size=15)

plt.subplot(1, 2, 2)
Negative,Hypothyroid,Hyperthyroid = data['target'].value_counts()
y = [Negative,Hypothyroid,Hyperthyroid]
explode = [0, 0.0,0]
labels = ['Negative', 'Hypothyroid','Hyperthyroid']

# Disease
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('Disease', size=15)
plt.show()
```



- The histogram illustrates the gender-specific distribution across different disease types, revealing a higher prevalence in females. Hypothyroidism exhibits the highest prevalence, followed by Hyperthyroidism. The majority of cases fall into the "Negative" category, indicating the absence of the depicted diseases.

- The pie chart highlights that the **"Negative"** category comprises **90.6%** of the dataset, with Hyperthyroidism and Hypothyroidism accounting for **7.86% and 1.54%**, respectively.

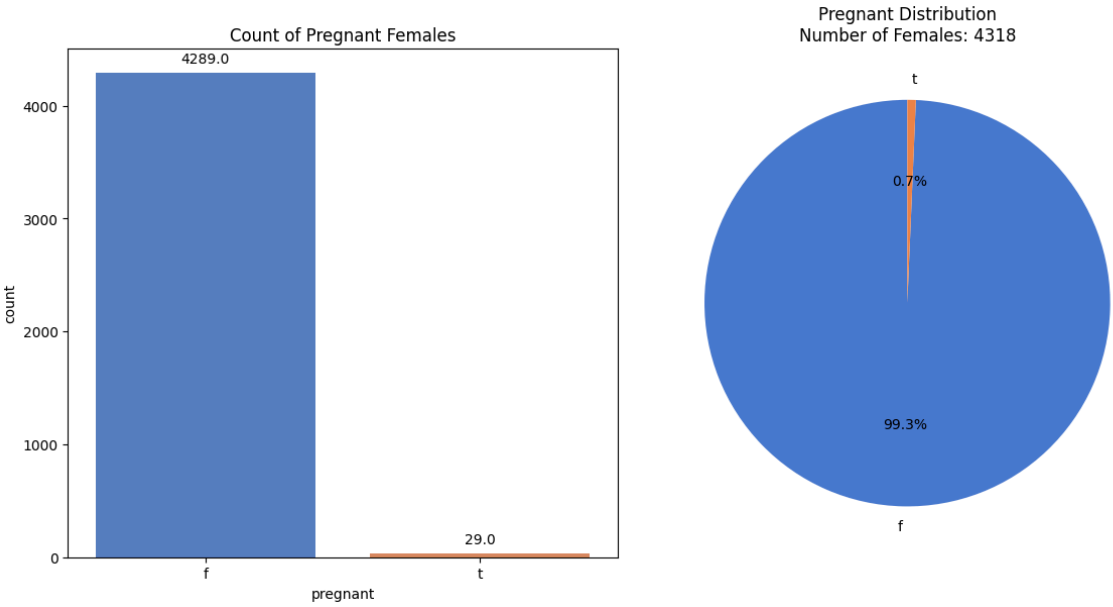
In [ ]:
```
# Violin Plot - comparing the distribution of a numeric variable across differen
# showing both the density and the summary statistics (median, quartiles)

plt.figure(figsize=(13, 5))

# Gender vs Disease
plt.subplot(1, 2, 1)
sns.violinplot(data=data, x="target", y="age", hue="sex", split=True, bw=.2, inn
plt.title('Gender vs Disease', size=15)

# Gender vs Sick
plt.subplot(1, 2, 2)
sns.violinplot(data=data, x="sick", y="age", hue="sex", split=True, bw=.2, inner
plt.title('Gender vs Sick', size=15)

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



Plotting for Pregnant Distribution:

In [ ]:
```
import seaborn as sns
import matplotlib.pyplot as plt

# Assuming 'data' is your DataFrame
female_data = data[data['sex'] == 'F']

fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(12, 6))

# Count of Pregnant Females
sns.countplot(x='pregnant', data=female_data, palette='muted', ax=axes[0])
axes[0].set_title('Count of Pregnant 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
        ha='center', va='center', xytext=(0, 10), textcoords='offset points', co

# Pregnant Distribution\nNumber of Females
pregnant_counts = female_data['pregnant'].value_counts()
axes[1].pie(pregnant_counts, labels=pregnant_counts.index, autopct='%1.1f%%', co
axes[1].set_title('Pregnant Distribution\nNumber of Females: {}'.format(len(fema
```

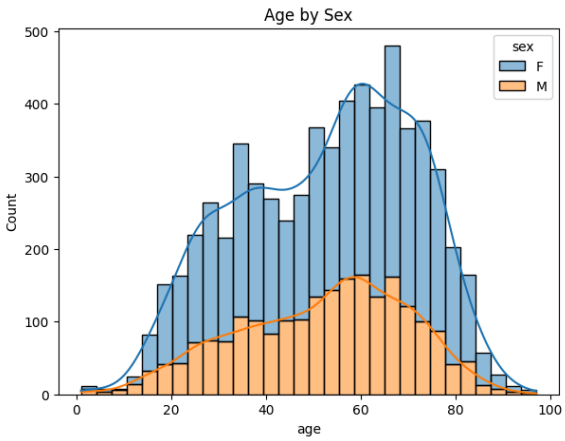
```
plt.tight_layout()
plt.show()
```

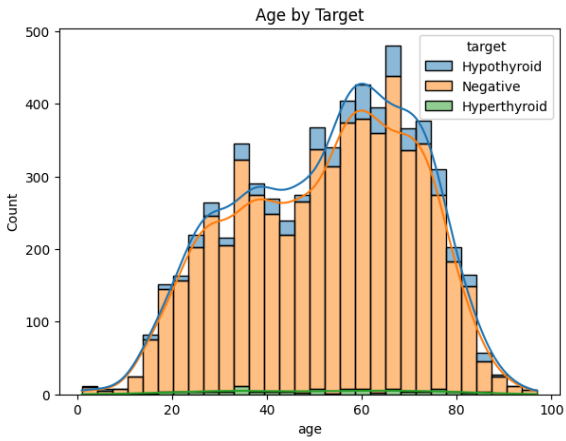


- The count plot illustrates the distribution of pregnant women across different age ranges. Each bar represents the count of pregnant women within a specific age group. It reveals that the highest number of pregnant women falls within the **25-29**age range, surpassing other age categories.
- The pie chart presents the percentage distribution of pregnant women across racial categories. It indicates that the majority of pregnant women are **white (66.5%)**, followed by **Hispanic (15.1%)**, **Black (12.2%)**, and **Asian (4.2%)**.

Plotting for Age by Sex & Target:

In [ ]:
```
# Stacked Histogram - comparing the distributions of multiple groups
plt.figure(figsize=(15,5))

# Age by Sex
plt.subplot(1, 2, 1)
sns.histplot(data=data, x="age", hue="sex", multiple="stack", kde=True)
plt.title('Age by Sex')

# Age by Target
plt.subplot(1, 2, 2)
sns.histplot(data=data, x="age", hue="target", multiple="stack", kde=True)
plt.title('Age by Target')

plt.show()
```





Stacked Histogram for Age by Sex

- Within the **20-30 age group**, approximately **200 females and 100 males** are represented. The histogram illustrates a consistent trend of higher female counts across most age brackets, except for the **0-10 age group**. Notably, the gender gap is more pronounced in younger age brackets, particularly in the **20-30 and 30-40 groups**, with a relatively smaller margin in older age groups.

Stacked Histogram for Age by Target

- In the **20-30 age** bracket, around 100 individuals belong to the blue target group, 50 to the green target group, and 25 to the orange target group. The histogram reveals varied distributions across different age groups concerning target categories. For instance, the blue target group is predominant in younger age brackets **(20-30 and 30-40)**, while the green target group prevails in older age groups **(50-60 and 60-70)**.

Violin Plot: Gender vs Disease

The violin plot for females with Hypothyroidism exhibits a broader body and longer tails compared to the plot for males with the same condition. This suggests a higher prevalence of Hypothyroidism among females, with a more varied severity level across the female population. Overall, the visualization indicates a gender-based disparity in disease prevalence, with females generally experiencing a wider range of disease severity compared to males.

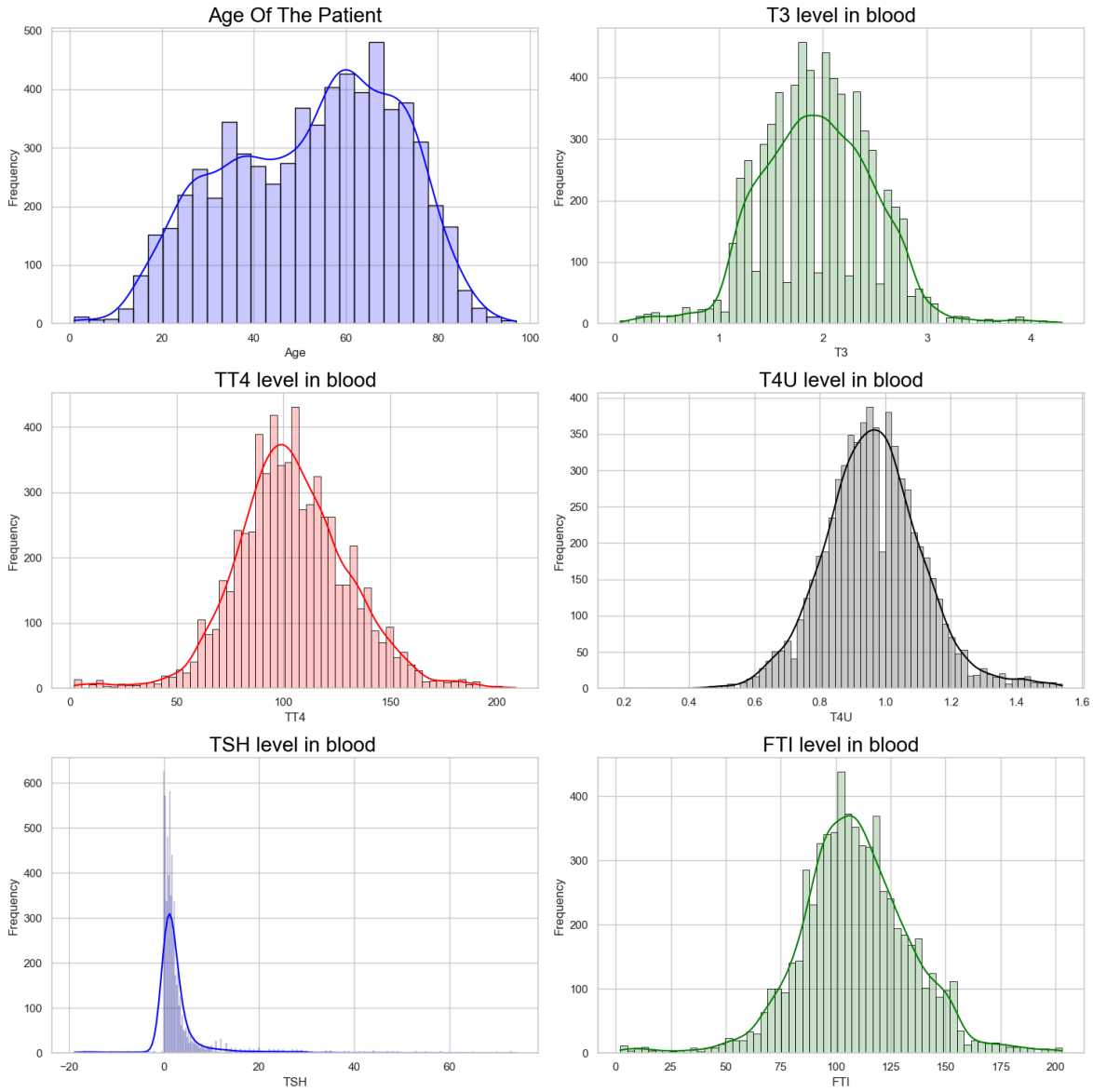
Violin Plot: Age vs Disease

In the age-specific violin plots for the "Negative" category, the plot for the **20-30 age group** displays a broader body compared to the same category in the **60-70 age group**. This observation implies a higher proportion of disease-free individuals in the younger age bracket. The visualization suggests that the distribution of diseases varies across different age cohorts, with younger individuals exhibiting a broader spectrum of health statuses compared to older individuals.

Plotting for distribution of key blood test features:

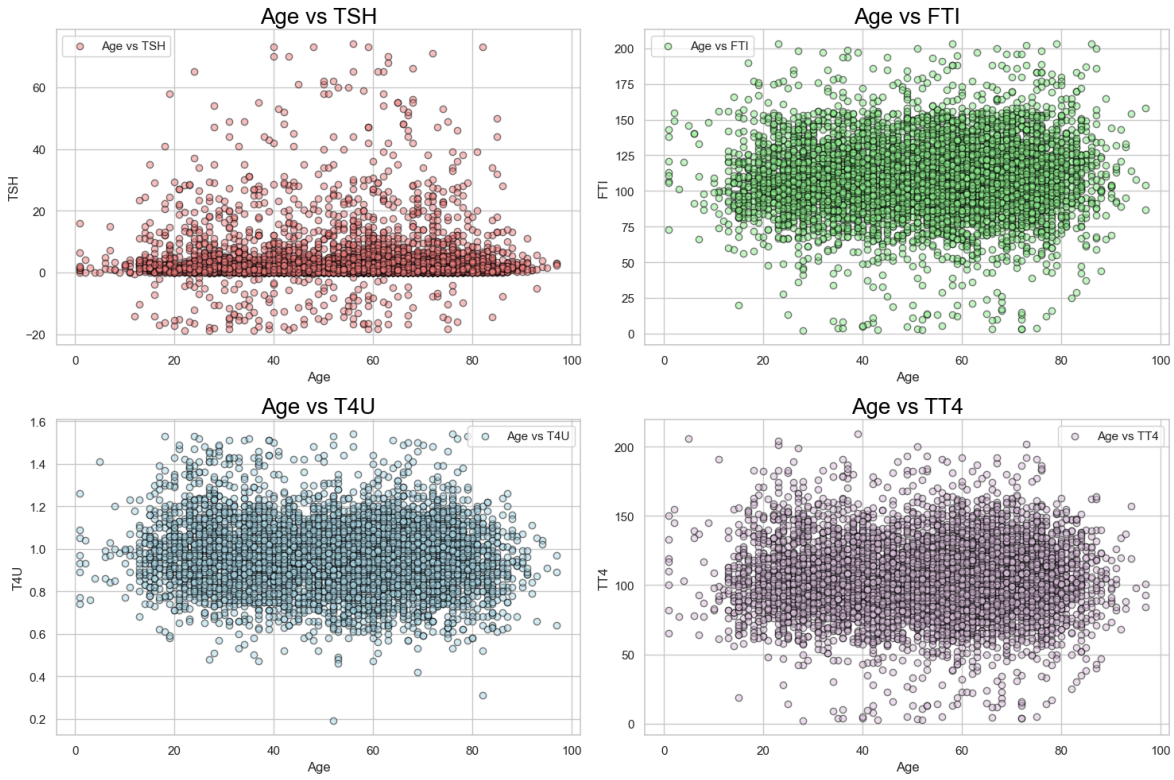
In [ ]:

```
# KDE(kernel density estimate) provides a smooth curve that represents the distr

plt.figure(figsize=(15,15))
sns.set(style='whitegrid')

# Age of the Patient
plt.subplot(3, 2, 1)
sns.histplot(data['age'], kde=True, color='blue', edgecolor='black',alpha=0.2)
plt.title('Age Of The Patient', color='black', fontsize=20)
plt.xlabel('Age')
plt.ylabel('Frequency')

# T3 level in blood
plt.subplot(3, 2, 2)
sns.histplot(data['T3'], kde=True, color='green', edgecolor='black',alpha=0.2)
plt.title('T3 level in blood', color='black', fontsize=20)
plt.xlabel('T3')
plt.ylabel('Frequency')

# TT4 level in blood
plt.subplot(3, 2, 3)
sns.histplot(data['TT4'], kde=True, color='red', edgecolor='black',alpha=0.2)
plt.title('TT4 level in blood', color='black', fontsize=20)
plt.xlabel('TT4')
plt.ylabel('Frequency')

# T4U level in blood
plt.subplot(3, 2, 4)
sns.histplot(data['T4U'], kde=True, color='black', edgecolor='black',alpha=0.2)
plt.title('T4U level in blood', color='black', fontsize=20)
plt.xlabel('T4U')
plt.ylabel('Frequency')

# TSH level in blood
plt.subplot(3, 2, 5)
sns.histplot(data['TSH'], kde=True, color='blue', edgecolor='black',alpha=0.2)
plt.title('TSH level in blood ', color='black', fontsize=20)
plt.xlabel('TSH')
plt.ylabel('Frequency')

# FTI level in blood
plt.subplot(3, 2, 6)
sns.histplot(data['FTI'], kde=True, color='green', edgecolor='black',alpha=0.2)
plt.title('FTI level in blood', color='black', fontsize=20)
plt.xlabel('FTI')
plt.ylabel('Frequency')

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



Plotting for Age vs Blood Test Features:

In [ ]:
```
import matplotlib.pyplot as plt
# Scatter plot - to show the relationship between two continuous variables
# identify any patterns or correlations

columns_to_plot = ['TSH', 'FTI', 'T4U', 'TT4']

plt.figure(figsize=(15, 10))
light_colors = ['lightcoral', 'lightgreen', 'lightblue', 'thistle']

for i, (column, color) in enumerate(zip(columns_to_plot, light_colors), start=1)
    plt.subplot(2, 2, i)
    plt.scatter(data['age'], data[column], label=f'Age vs {column}', color=color
    plt.xlabel('Age')
    plt.ylabel(column)
    plt.title(f'Age vs {column}', color='black', fontsize=20)
    plt.legend()
plt.tight_layout()
plt.show()
```



Age vs TSH:

- A mild positive correlation is noticeable between age and TSH levels, indicating that older individuals tend to exhibit slightly higher TSH levels. However, there is considerable variation within age groups.

Age vs FTI:

- No discernible correlation emerges between age and FTI levels. The relationship between age and FTI levels appears inconsistent, showing no clear trend.

Age vs T4U:

- A slight negative correlation is observed between age and T4U levels, suggesting that older individuals tend to have slightly lower T4U levels. Nonetheless, there is notable variability in T4U levels within each age group.

Age vs TT4:

- A subtle negative correlation exists between age and TT4 levels, indicating that older individuals tend to exhibit slightly lower TT4 levels. Nevertheless, there is considerable variation in TT4 levels across different age groups.

Plot of Age vs T3 with Target Labels

In [ ]:
```
import plotly.express as px
fig1 = px.scatter(data, x='age', y='T3', color='target', color_discrete_map={'Hy

fig1.show()
```

Plot of Age vs FTI with Target Labels

In [ ]:
```
fig1 = px.scatter(data, x='age', y='FTI', color='target', color_discrete_map={'H
fig1.show()
```

Plot of Age vs T4U with Target Labels

In [ ]:
```
fig1 = px.scatter(data, x='age', y='T4U', color='target', color_discrete_map={'H
fig1.show()
```

Plot of Age vs TT4 with Target Labels

In [ ]:
```
fig1 = px.scatter(data, x='age', y='TT4', color='target', color_discrete_map={'H

fig1.show()
```

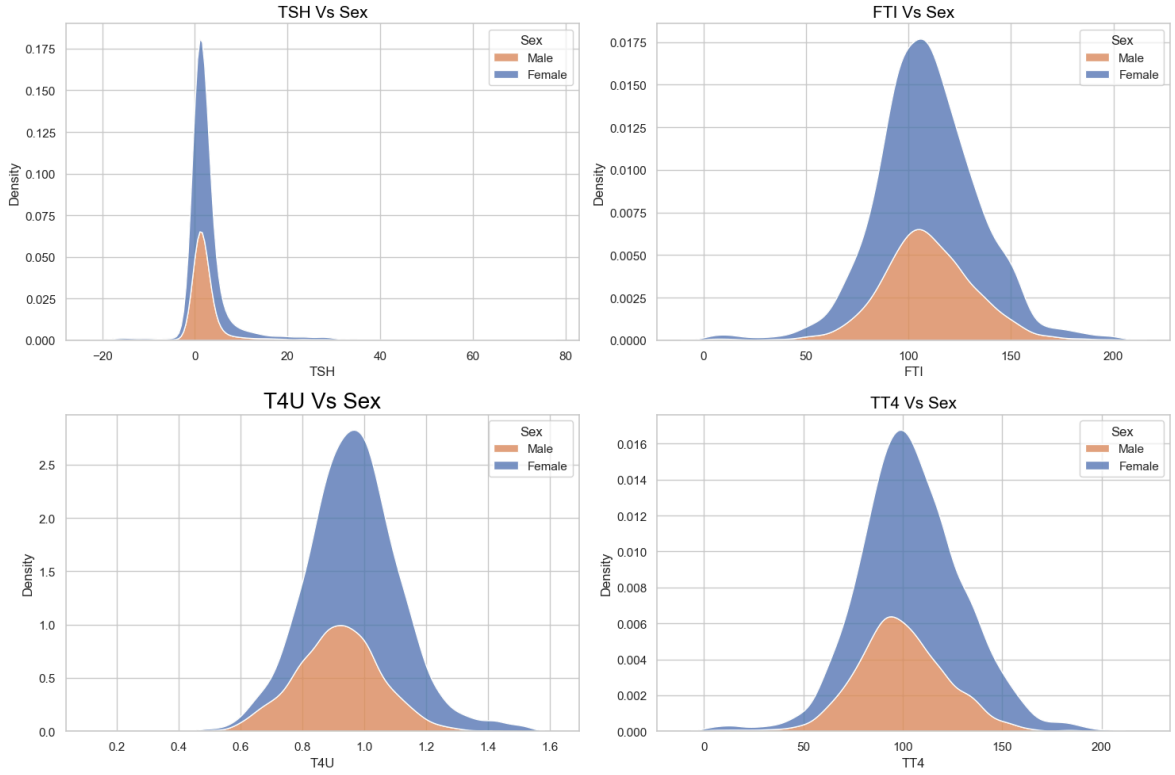
3D Scatter Plot of Blood Test Features by Age and Thyroid Disease

In [ ]:
```
import plotly.express as px

# Assuming data is your DataFrame
fig = px.scatter_3d(data, x='T4U', y='T3', z='TSH', color='target',
                    color_discrete_map={'Hypothyroid': '#4b9546', 'Hyperthyroid'
                    size='age', opacity=0.7, hover_data=['age', 'target'])

# Set the layout to make it full page
fig.update_layout(width=800, height=800)

# Show the plot
fig.show()
```

Plotting for Blood Test Features vs Sex:

In [ ]:
```
import matplotlib.pyplot as plt
import seaborn as sns

# Density Plot: shows the distribution of a continuous variable.
# The area under the curve represents the probability distribution of the data p

plt.figure(figsize=(15, 10))
sns.set(style='whitegrid')

# TSH Vs Sex
plt.subplot(2, 2, 1)
sns.kdeplot(data=data, x="TSH", hue="sex", multiple="stack", fill=True)
plt.title("TSH Vs Sex", fontsize=15, color='black')
plt.legend(title='Sex', labels=['Male', 'Female'])

# FTI Vs Sex
plt.subplot(2, 2, 2)
sns.kdeplot(data=data, x='FTI', hue="sex", multiple="stack", fill=True)
plt.title("FTI Vs Sex", fontsize=15, color='black')
plt.legend(title='Sex', labels=['Male', 'Female'])

# T4U Vs Sex
plt.subplot(2, 2, 3)
sns.kdeplot(data=data, x='T4U', hue="sex", multiple="stack", fill=True)
```

```
plt.title("T4U Vs Sex", fontsize=20, color='black')
plt.legend(title='Sex', labels=['Male', 'Female'])

# TT4 Vs Sex
plt.subplot(2, 2, 4)
sns.kdeplot(data=data, x="TT4", hue="sex", multiple="stack", fill=True)
plt.title("TT4 Vs Sex", fontsize=15, color='black')
plt.legend(title='Sex', labels=['Male', 'Female'])

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



- In the first chart, we observe the density distribution of thyroid-stimulating hormone **(TSH) levels** categorized by sex. The x-axis represents TSH levels, while the y-axis denotes the density. Males are depicted in blue, and females in pink. The chart illustrates higher TSH levels among females compared to males.

- The second chart displays the density distribution of free thyroxine index **(FTI) levels** by sex. The x-axis signifies FTI levels, and the y-axis indicates density. Males are represented in blue, and females in pink. The visualization indicates higher FTI levels in females than in males.

- In the third chart, we explore the density distribution of total thyroxine uptake **(T4U) levels** based on sex. The x-axis portrays T4U levels, while the y-axis depicts density. Males are depicted in blue, and females in pink. The chart suggests higher T4U levels among females compared to males.

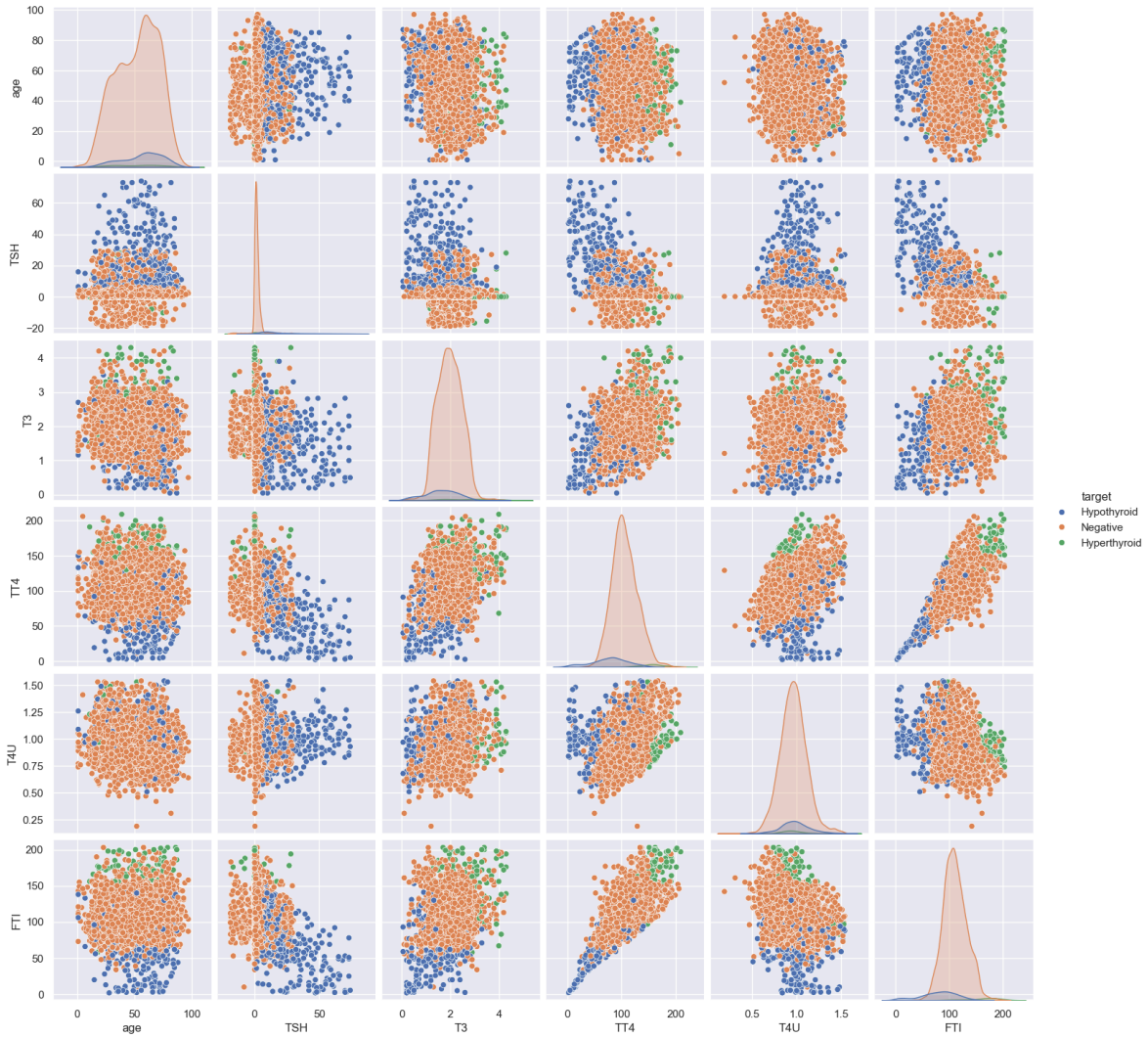
- Lastly, the fourth chart showcases the density distribution of total thyroxine **(TT4) levels** categorized by sex. The x-axis represents TT4 levels, and the y-axis signifies density. Males are shown in blue, and females in pink. The visualization suggests higher TT4 levels in females compared to males.

Plot of Age Distribution by Gender and Thyroid Disease Status:

In [ ]:
```
fig4 = px.violin(data, x='sex', y='age', color='target', points='all', facet_col
                 color_discrete_map={'Hypothyroid': '#4b9546', 'Hyperthyroid': '#
fig4.show()
```

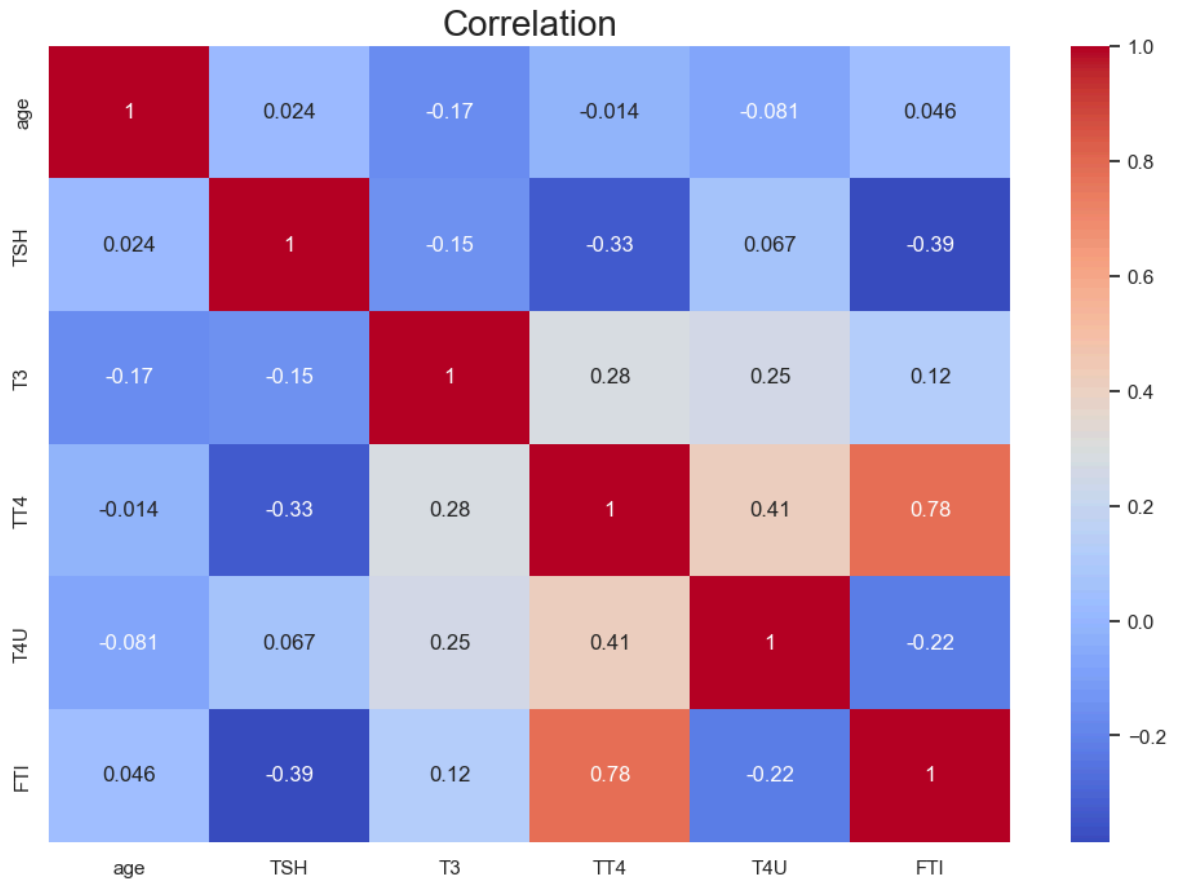
Plotting for Exploring Relationships between Numerical Features and Target Classes:

In [ ]:
```
import seaborn as sns
import matplotlib.pyplot as plt

# Pair plot - comprehensive visualization of the relationships between different
# as well as their distribution concerning the target variable.

# Assuming 'data' is your DataFrame containing numerical and categorical columns
sns.set_style('darkgrid')

# Replace 'data' with your actual DataFrame
selected_columns = data.select_dtypes(include=['number']).columns.tolist() + ['t
sns.pairplot(data[selected_columns], hue='target')

plt.show()
```



Plotting for Correlation of Numerical Features

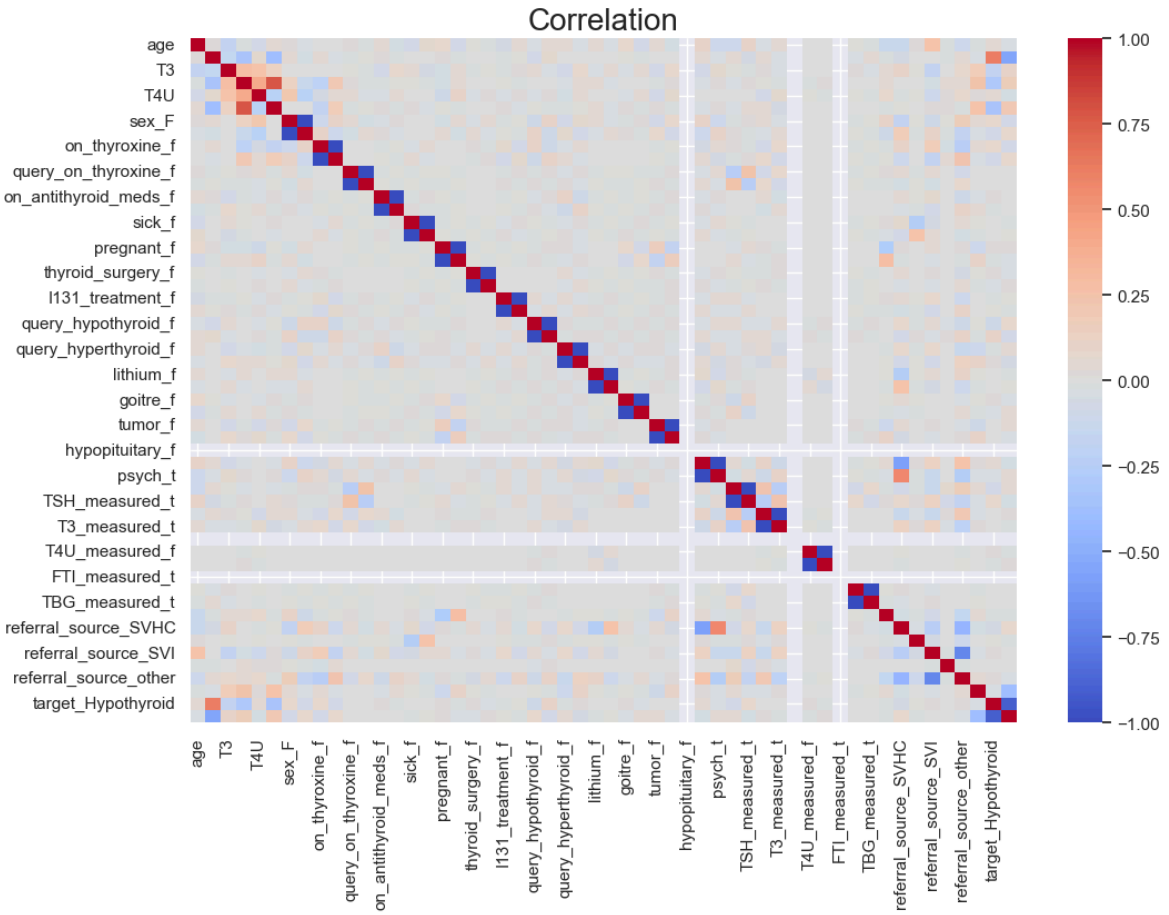
In [ ]:
```
# Heat Map - visualize the correlation matrix of the encoded dataset
# gain insights into the relationships within your data, aiding in effective dat
# which features to include or exclude from your model
# 'coolwarm' color map is used to represent the correlation coefficients. colors
# (annot=False) to make the heatmap less cluttered. If turned on, it would displ

plt.figure(figsize=(12,8))
sns.heatmap(data.select_dtypes(include=['number']).corr(),annot=True,cmap='coolw
plt.title("Correlation",fontsize=20)
plt.show()
```



- **Age:** There's a *weak negative correlation between age and TSH and T3 levels*, while there's a *weak positive correlation with TT4, T4U, and FTI levels*. This suggests that as individuals age, their TSH and T3 levels typically decrease, whereas their TT4, T4U, and FTI levels tend to increase.
- **TSH:** TSH exhibits a *strong negative correlation with T3, TT4, T4U, and FTI*. This implies that higher TSH levels correspond to lower levels of other thyroid hormones, and vice versa. TSH stimulates the production of T3 and T4 by the thyroid gland, hence the inverse relationship.
- **T3:** T3 shows a *strong positive correlation with TT4 and T4U*, and a *moderate positive correlation with FTI*. This indicates that *higher T3 levels often coincide with elevated TT4, T4U, and FTI levels*.
- **TT4:** TT4 demonstrates a *strong positive correlation with T4U and FTI*. This implies that *higher TT4 levels typically align with increased T4U and FTI levels*.

Plotting for Correlation of Encoded Features

In [ ]:
```
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 exhibits a **weak negative correlation with T3 and T4U**, while **weakly positively correlated with TT4 and FTI**. This implies that as individuals age, their T3 and T4U levels typically decrease, whereas their TT4 and FTI levels tend to rise.

Sex:

- Female gender is **negatively correlated with TSH and positively correlated with FT4**. This suggests that women generally have lower TSH levels and higher FT4 levels compared to men.

Thyroid Medication:

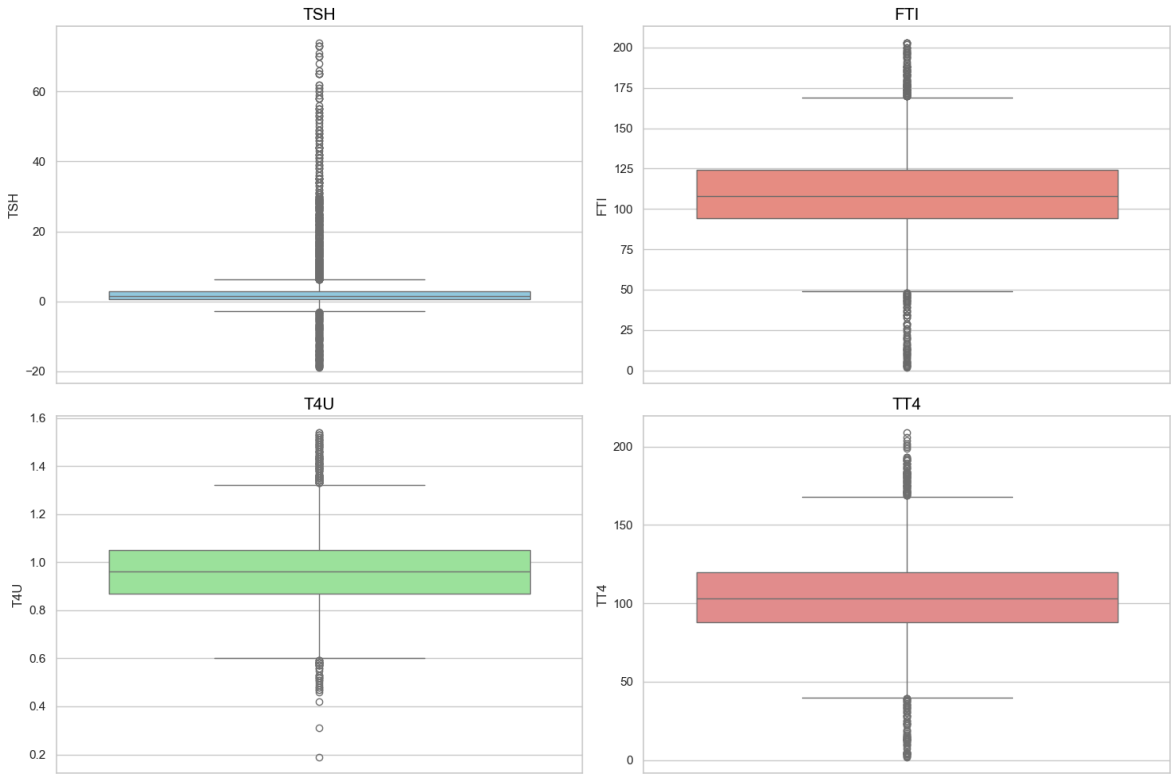
- Use of thyroxine therapy (on_thyroxine_f) is **strongly negatively correlated with TSH, T3, and FT4**. This aligns with expectations, as thyroxine medication suppresses TSH production and elevates thyroid hormone levels. Conversely, taking antithyroid medication (on_antithyroid_meds_f) is positively correlated with TSH and negatively correlated with FT4, reflecting its role in inhibiting thyroid hormone production and elevating 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**, whereas hypothyroidism **(target_Hypothyroid) associates with lower T4 levels and higher TSH levels.**

Plotting for Distribution of Thyroid Function Test Results:

In [ ]:
```
# useful for visualizing the distribution of the numerical variables in your dat
# Outlier Detection, Distribution Understanding, Comparative Analysis
# Make informed decisions about preprocessing and handling outliers, ultimately

plt.figure(figsize=(15, 10))
sns.set(style='whitegrid')

plt.subplot(2, 2, 1)
sns.boxplot(data=data, y="TSH", color='skyblue')
plt.title("TSH", fontsize=15, color='black')

plt.subplot(2, 2, 2)
sns.boxplot(data=data, y='FTI', color='salmon')
plt.title("FTI", fontsize=15, color='black')

plt.subplot(2, 2, 3)
sns.boxplot(data=data, y='T4U', color='lightgreen')
plt.title("T4U", fontsize=15, color='black')

plt.subplot(2, 2, 4)
sns.boxplot(data=data, y="TT4", color='lightcoral')
plt.title("TT4", fontsize=15, color='black')

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



TSH:

- In the euthyroid group, the median *TSH level appears significantly lower compared to the hyperthyroid group*. This aligns with the typical pattern where TSH levels are suppressed among individuals with hyperthyroidism.

FTI, T4U, and TT4:

- Within the hyperthyroid group, *median levels of FTI, T4U, and TT4 exhibit elevation compared to the euthyroid group*. This conforms to the expected rise in these thyroid hormone levels among individuals with hyperthyroidism.

Spread of data:

- Examination of the boxplots reveals a wider spread of data in the hyperthyroid group across all four thyroid hormone levels in comparison to the euthyroid group. This variability indicates a broader range of thyroid hormone levels among individuals diagnosed with hyperthyroidism.

Convert Categorical To Numerical Data And Drop Unnecessary Columns

- Convert categorical data into numerical format to facilitate analysis and compatibility with machine learning algorithms. This can be achieved through techniques such as one-hot encoding or label encoding, enhancing the dataset for further analysis.

Drop Unnecessary Columns

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

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

Out[ ]:

	age	sex	TT4	T3	T4U	FTI	TSH	pregnant	target
18	63	F	48.0	1.892490	1.02	47.0	68.000000	f	Hypothyroid
19	36	F	90.0	2.400000	1.06	85.0	1.500000	f	Negative
20	40	F	79.0	1.454813	0.94	84.0	-3.874467	f	Negative
21	40	F	104.0	2.300000	1.08	96.0	1.200000	f	Negative
22	40	F	88.0	2.100000	0.84	105.0	5.900000	f	Negative
23	77	F	107.0	2.400000	1.13	95.0	0.050000	f	Negative
25	77	F	113.0	2.854427	1.07	106.0	0.400000	f	Negative
27	51	F	93.0	2.100000	0.87	106.0	0.050000	f	Negative
28	75	F	157.0	1.600000	0.89	176.0	0.050000	f	Hyperthyroid
29	56	M	80.0	1.600000	0.62	129.0	0.200000	f	Negative

Convert Categorical data to numerical

In [ ]:
```
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)
```

Out[ ]:

	age	sex	TT4	T3	T4U	FTI	TSH	pregnant	target
18	63	0	48.0	1.892490	1.02	47.0	68.000000	0	1
19	36	0	90.0	2.400000	1.06	85.0	1.500000	0	2
20	40	0	79.0	1.454813	0.94	84.0	-3.874467	0	2
21	40	0	104.0	2.300000	1.08	96.0	1.200000	0	2
22	40	0	88.0	2.100000	0.84	105.0	5.900000	0	2
23	77	0	107.0	2.400000	1.13	95.0	0.050000	0	2
25	77	0	113.0	2.854427	1.07	106.0	0.400000	0	2
27	51	0	93.0	2.100000	0.87	106.0	0.050000	0	2
28	75	0	157.0	1.600000	0.89	176.0	0.050000	0	0
29	56	1	80.0	1.600000	0.62	129.0	0.200000	0	2

- 0 - Female
- 1- Male

Target

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

In [ ]:
```
print('Size of data frame after convert numeric data set :',data.shape)
print('Rows :',data.shape[0])
print('Columns :',data.shape[1])
```

```
Size of data frame after convert numeric data set : (6501, 9)
Rows : 6501
Columns : 9
```

Training And Testing Sets

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

- **Split data into Training and Testing**

In [ ]:
```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.2,random_state=
```

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

```
Length of Training Set : 5200
Length of Testing Set : 1301
```

Model Execution:

- Given the presence of numerous outliers observed in the dataset via the boxplot analysis, the choice of a non-parametric machine learning model such as KNN,RF and SVM are justified.

KNN (K-Nearest Neighbors) Classifier:

- The KNN algorithm is a flexible and intuitive approach widely utilized in thyroid detection tasks. It functions by classifying data points according to the majority class among their k-nearest neighbors in the feature space. Renowned for its simplicity and capability to handle non-linear relationships, KNN emerges as a valuable tool for discerning patterns and making precise predictions in thyroid detection projects.

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

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

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

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

Out[ ]:
```
0.9415833973866257
```

In [ ]:
```
#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)

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

```
Training Score : 0.9673076923076923 %
Testing Score : 0.9415833973866257 %
```

In [ ]:
```
r = knn.predict(x_test)
r[1]
```

Out[ ]:
```
2
```

KNN Result Analysis:

- The high **training score of 96.7%** demonstrates the model's proficiency in learning patterns from the training data, indicating a strong fit.
- With a **testing score of 94%**, the model exhibits excellent performance on unseen data, reflecting its ability to generalize well to new instances.

Random Forest Classifier:

- The Random Forest algorithm is a robust and versatile machine learning technique widely applied in thyroid detection tasks. It operates by constructing a multitude of decision trees during training and outputting the mode of the classes for classification tasks. This ensemble method excels in handling high-dimensional data and mitigating overfitting, making it a reliable and accurate tool for identifying patterns and making predictions in thyroid detection projects. The algorithm's ability to provide feature importance also enhances interpretability and aids in understanding the underlying factors influencing thyroid conditions.

In [ ]:
```
from sklearn.ensemble import RandomForestClassifier
import numpy as np

# Initialize the Random Forest model
rf = RandomForestClassifier(n_estimators=100, random_state=1)

# Train the model
rf.fit(x_train, y_train.values.ravel())
```

Out[ ]:
```
RandomForestClassifier
RandomForestClassifier(random_state=1)
```

In [ ]:
```
rf.score(x_test,y_test)
```

Out[ ]:
```
0.9600307455803229
```

In [ ]:
```
# Evaluate the model
Training_score = rf.score(x_train, y_train)
Testing_score = rf.score(x_test, y_test)

# Predict on the test set
y_pred = rf.predict(x_test)

# Print the results
print('Training Score :', Training_score, '%')
print('Testing Score :', Testing_score, '%')
```

```
Training Score : 1.0 %
Testing Score : 0.9600307455803229 %
```

In [ ]:
```
# Example of predicting with the Random Forest model
r = rf.predict(x_test)
print(r[1])
```

```
2
```

Random Forest Classifier Result Analysis :

- The **training score of 100%** demonstrates the model's impeccable ability to learn patterns from the training data, indicating a perfect fit.
- With a **testing score of 96%**, the model showcases outstanding performance on unseen data, reflecting its superior capability to generalize to new instances.

Support Vector Machine (SVM) Classifier:

- The SVM classifier is a powerful and versatile model widely used in various classification tasks, including thyroid detection. It works by finding the optimal hyperplane that separates data points of different classes in the feature space. Known for its effectiveness in handling high-dimensional data and non-linear relationships, the SVM classifier is a robust tool for identifying patterns and making accurate predictions in thyroid detection projects.

In [ ]:
```
from sklearn.svm import SVC

# Initialize the SVM classifier
svm = SVC(kernel='linear')

# Train the model
svm.fit(x_train, y_train)
```

Out[ ]:
```
SVC
SVC(kernel='linear')
```

In [ ]:
```
svm.score(x_test,y_test)
```

Out[ ]:
```
0.9308224442736357
```

In [ ]:
```
# Evaluate the model
Training_score_svm = svm.score(x_train, y_train)
Testing_score_svm = svm.score(x_test, y_test)

# Predict on the test set
y_pred_svm = svm.predict(x_test)

print('Training Score:', Training_score_svm, '%')
print('Testing Score:', Testing_score_svm, '%')
```

```
Training Score: 0.926923076923077 %
Testing Score: 0.9308224442736357 %
```

In [ ]:
```
r = svm.predict(x_test)
print(r[1])
```

```
2
```

Support Vector Machine (SVM) Classifier Result Analysis:

- The **training score of 92.7%** indicates that the model effectively learned the patterns present in the training data.
- With a **testing score of 93%**, the SVM model demonstrates reliable performance on unseen data, suggesting good generalization capability.

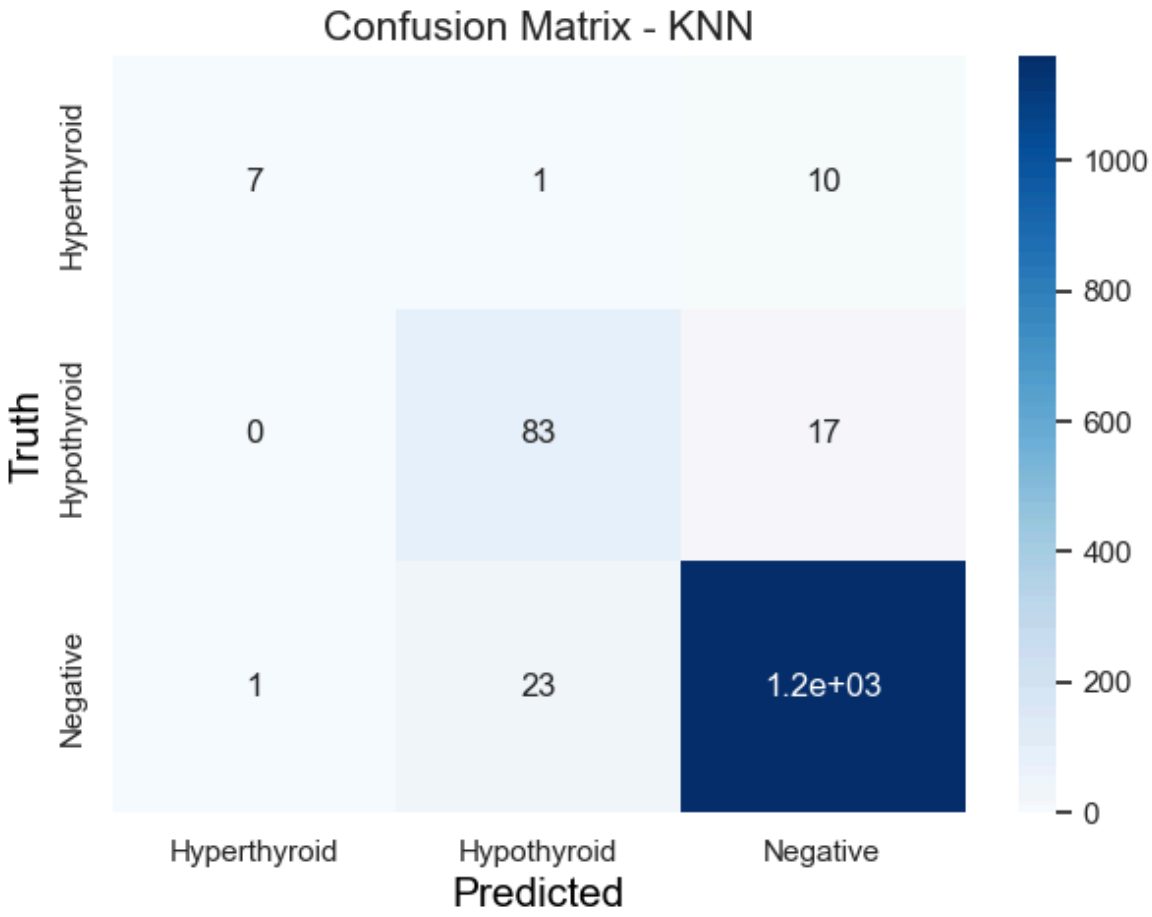
Confusion Matrix Visualization:

A confusion matrix is an essential instrument for assessing the efficacy of a classification model. It offers a detailed breakdown of true positives, true negatives, false positives, and false negatives, providing valuable insights into the precision, recall, and overall accuracy of the model. By visually representing the alignment between predicted and actual classes, confusion matrices facilitate model refinement and pinpoint areas for enhancement in machine learning endeavors.

In [ ]:
```
# Confusion Matrix for KNN
from sklearn.metrics import confusion_matrix
cm_knn = confusion_matrix(y_test, y_pred)

# Confusion Matrix for Random Forest
y_pred_rf = rf.predict(x_test)
cm_rf = confusion_matrix(y_test, y_pred_rf)

# Confusion Matrix for SVM
y_pred_svm = svm.predict(x_test)
cm_svm = confusion_matrix(y_test, y_pred_svm)
```

In [ ]:
```
import seaborn as sns
import matplotlib.pyplot as plt
```

In [ ]:
```
# Confusion Matrix for KNN
cm_knn_df = pd.DataFrame(cm_knn,
                         index=['Hyperthyroid', 'Hypothyroid', 'Negative'],
                         columns=['Hyperthyroid', 'Hypothyroid', 'Negative'])

plt.figure(figsize=(7, 5))
sns.heatmap(cm_knn_df, annot=True, cmap='Blues')
plt.xlabel('Predicted', color='black', size=15)
plt.ylabel('Truth', color='black', size=15)
plt.title('Confusion Matrix - KNN', fontsize=15)
plt.show()
```

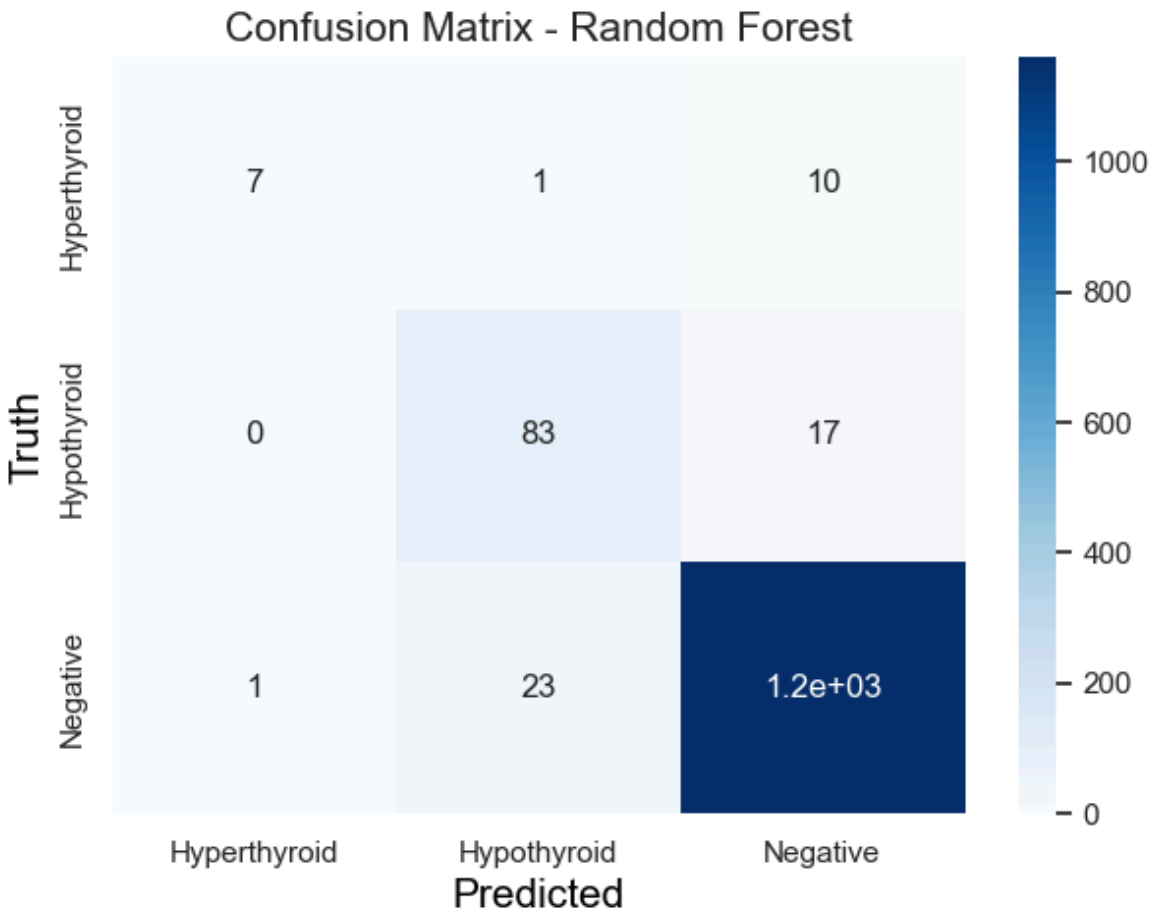


KNN Confusion matrix Analysis:

- For class 0 (Hyperthyroid), the model misclassified 10 instances as Negative.
- For class 1 (Hypothyroid), the model erroneously predicted 17 instances as Negative.
- For class 2 (Negative), the model made 1 misclassifications as Hyperthyroid and 23 misclassifications as Hypothyroid.

In [ ]:
```
# Confusion Matrix for Random Forest
cm_rf_df = pd.DataFrame(cm_rf,
                        index=['Hyperthyroid', 'Hypothyroid', 'Negative'],
                        columns=['Hyperthyroid', 'Hypothyroid', 'Negative'])

plt.figure(figsize=(7, 5))
sns.heatmap(cm_rf_df, annot=True, cmap='Blues')
```

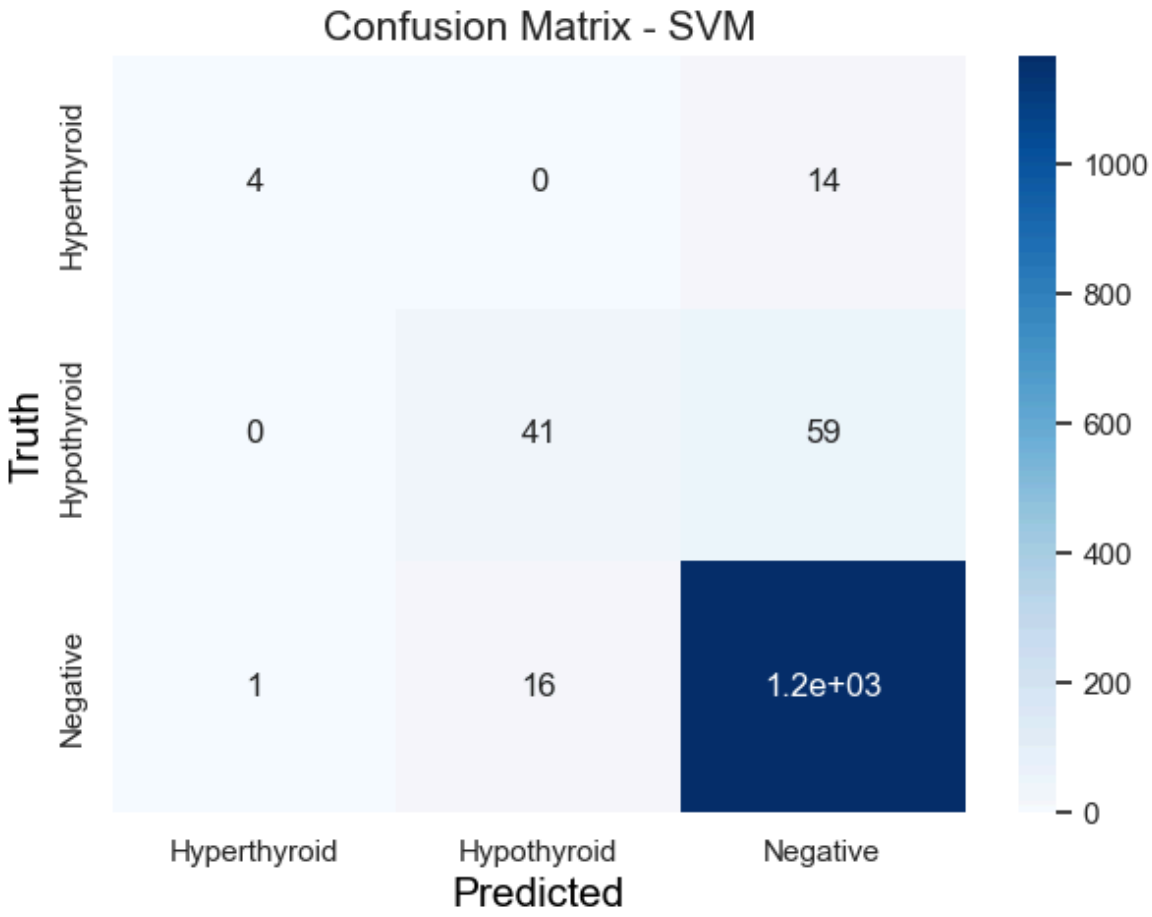
```
plt.xlabel('Predicted', color='black', size=15)
plt.ylabel('Truth', color='black', size=15)
plt.title('Confusion Matrix - Random Forest', fontsize=15)
plt.show()
```



Randon Forest Confusion matrix Analysis:

- For class 0 (Hyperthyroid), the model misclassified 10 instances as Negative.
- For class 1 (Hypothyroid), the model erroneously predicted 17 instances as Negative.
- For class 2 (Negative), the model made 1 misclassifications as Hyperthyroid and 23 misclassifications as Hypothyroid.

In [ ]:
```
# Confusion Matrix for SVM
cm_svm_df = pd.DataFrame(cm_svm,
                         index=['Hyperthyroid', 'Hypothyroid', 'Negative'],
                         columns=['Hyperthyroid', 'Hypothyroid', 'Negative'])

plt.figure(figsize=(7, 5))
sns.heatmap(cm_svm_df, annot=True, cmap='Blues')
plt.xlabel('Predicted', color='black', size=15)
plt.ylabel('Truth', color='black', size=15)
plt.title('Confusion Matrix - SVM', fontsize=15)
plt.show()
```



SVM Confusion matrix Analysis:

- For class 0 (Hyperthyroid), the model misclassified 14 instances as Negative.
- For class 1 (Hypothyroid), the model erroneously predicted 59 instances as Negative.
- For class 2 (Negative), the model made 1 misclassifications as Hyperthyroid and 16 misclassifications as Hypothyroid.

Conclusion

Model Performance:

1. **K-Nearest Neighbors (KNN):**
    - **Training Score:** 96.7%
    - **Testing Score:** 94%
    - **Misclassifications:** Notable misclassifications, especially in distinguishing between Hyperthyroid and Hypothyroid from Negative.
2. **Random Forest Classifier:**
    - **Training Score:** 100%
    - **Testing Score:** 96%
    - **Misclassifications:** Fewer errors compared to other models; demonstrates the best overall performance.
3. **Support Vector Machine (SVM):**
    - **Training Score:** 92.7%

- **Testing Score:** 93%
- **Misclassifications:** Higher errors observed, particularly for cases misclassified as Negative.

Overall Conclusion:

- **Best Model:** Random Forest, with the highest accuracy and best generalization.
- **Recommendation:** Random Forest model is recommended for deployment due to its robust performance and lower misclassification rate.
- **Insights:** Analysis reveals potential areas for improvement in feature selection and preprocessing to enhance model performance and accuracy.

Model Saving & Testing

In [ ]:
```
import pickle

# Save the models
pickle.dump(knn, open('knn_model.pkl', 'wb'))
pickle.dump(rf, open('rf_model.pkl', 'wb'))
pickle.dump(svm, open('svm_model.pkl', 'wb'))
```

In [ ]:
```
# Load the models
knn_model = pickle.load(open('knn_model.pkl', 'rb'))
rf_model = pickle.load(open('rf_model.pkl', 'rb'))
svm_model = pickle.load(open('svm_model.pkl', 'rb'))
```

In [ ]:
```
# Model Prediction function

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

In [ ]:
```
# Define class labels and their corresponding names
class_labels = {
    0: "Hyperthyroid",
    1: "Hypothyroid",
    2: "Negative"
}
```

In [ ]:
```
# # Define model names and corresponding models
models = {
    "KNN": knn_model,
    "Random Forest": rf_model,
    "SVM": svm_model
}

# Test data: a list of test instances
user_data_list = [
    [63, 1, 48.0, 2.84, 1.0, 47.0, 65.0, 1],
    [20, 0, 4.8, 2.8, 1.0, 7.0, 1.4, 1],
    # Add more test instances here
]

# Iterate through each user data and make predictions
```

```python
for user_data in user_data_list:
    print(f"Testing on data: {user_data}")
    for model_name, model in models.items():
        prediction = model.predict([user_data])
        class_label = class_labels[prediction[0]]
        print(f"{model_name} Prediction: {prediction[0]} - {class_label}")
    print("\n")
```

```
Testing on data: [63, 1, 48.0, 2.84, 1.0, 47.0, 65.0, 1]
KNN Prediction: 1 - Hypothyroid
Random Forest Prediction: 1 - Hypothyroid
SVM Prediction: 1 - Hypothyroid


Testing on data: [20, 0, 4.8, 2.8, 1.0, 7.0, 1.4, 1]
KNN Prediction: 1 - Hypothyroid
Random Forest Prediction: 2 - Negative
SVM Prediction: 2 - Negative
```