CSE 587B - Data Intensive Computing
Project 1
Phase 1

Aditya Verma - averma27
Deepa Umesh - deepaume
Meghna Verma - mverma4
Pavithran Gnanasekaran - pgnanase

Title - Thyroid disease classification

Domain - Healthcare

Problem Statement

Thyroidism, including both Hypothyroidism and Hyperthyroidism, are life threatening diseases that affect millions of people every year around the world. It is imperative that it is detected early and accurately, which will help in the prompt treatment of the disease.

This can be a challenging process as it requires examining a wide range of factors, such as previous medical history, current body conditions, and blood tests.

In the scope of our problem statement, we have chosen to detect the major two thyroid diseases - hyper and hypo thyroidism. We also cover negative cases, as in, if neither of the two diseases is present.

There can also be cases where other diseases can affect the hormones, triggering a spike in one or more of the chemical compounds. These cases mean that the patient doesn't have either hyper or hypo thyroidism, but could have some other disease. Thus, in the scope of the problem statement, we will classify such cases as negative.

Thyroidism can be caused by a wide range of symptoms , some of which are natural, like age, sex, sickness, and others which can be detected by blood tests, like the levels of certain chemical compounds in the blood.

Impact On The Domain

We believe that making data driven decisions can greatly help medical professionals, and also the patients. Using a solution like the one presented in this project, doctors and patients both can get a fair chance of the likelihood of either of the diseases being present, or being absent altogether. This could help fast-track the more meticulous and thorough methods of the detection of the disease if detected, and possibly get the confirmation and begin the treatment. In many medical cases, early treatment is critical in order to treat the patient properly without any side effects.

Using this solution as a benchmark, other researchers can also possibly come up with solutions to this problem, and also come up with similar data-driven approaches for other diseases.

Data Source

We found a dataset from Kaggle, which aligned with our problem statement

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

This dataset contains **31 columns and 9172 rows**

Following are the columns present in the dataset, with short descriptions and datatypes:

1. **age** - age of the patient **(int)**
2. **sex** - sex patient identifies **(str)**
3. **on_thyroxine** - whether patient is on thyroxine **(bool)**
4. **query on thyroxine** - whether patient is on thyroxine **(bool)**
5. **on antithyroid meds** - whether patient is on antithyroid meds **(bool)**
6. **sick** - whether patient is sick **(bool)**
7. **pregnant** - whether patient is pregnant **(bool)**
8. **thyroid_surgery** - whether patient has undergone thyroid surgery **(bool)**
9. **I131_treatment** - whether patient is undergoing I131 treatment **(bool)**
10. **query_hypothyroid** - whether patient believes they have hypothyroid **(bool)**
11. **query_hyperthyroid** - whether patient believes they have hyperthyroid **(bool)**
12. **lithium** - whether patient lithium **(bool)**
13. **goitre** - whether patient has goitre **(bool)**
14. **tumor** - whether patient has tumor **(bool)**
15. **hypopituitary** - whether patient hyperpituitary gland **(float)**
16. **psych** - whether patient psych **(bool)**
17. **TSH_measured** - whether TSH was measured in the blood **(bool)**
18. **TSH** - TSH level in blood from lab work **(float)**
19. **T3_measured** - whether T3 was measured in the blood **(bool)**
20. **T3** - T3 level in blood from lab work **(float)**
21. **TT4_measured** - whether TT4 was measured in the blood **(bool)**
22. **TT4** - TT4 level in blood from lab work **(float)**
23. **T4U_measured** - whether T4U was measured in the blood **(bool)**
24. **T4U** - T4U level in blood from lab work **(float)**

25. **FTI_measured** - whether FTI was measured in the blood **(bool)**
26. **FTI** - FTI level in blood from lab work **(float)**
27. **TBG_measured** - whether TBG was measured in the blood **(bool)**
28. **TBG** - TBG level in blood from lab work **(float)**
29. **referral_source** - Source of referral of the patient**(str)**
30. **target** - hyperthyroidism medical diagnosis **(str)**
31. **patient_id** - unique id of the patient **(str)**

Note : DOI Citation for this data source is currently not available on the Kaggle site.

Data Overview

First, we examine the initial state of the dataset using the shape, info and describe commands.

Shape command gives an output of : **(9172,31)**

Info command:

```
<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
```

Here, we can see the metadata of the dataset, where we observe the non null count, and data types. We see that most of the columns are of object type, or floats. On observing the data, we find that the object data type corresponds to strings stored as ‘f’ and ‘t’, corresponding to False and True. We will take care of this in the data cleaning step.

We also see that some columns have lesser non null values than others, indicating the presence of null values in the data, which have to be preprocessed.

Describe command:

	age	TSH	T3	TT4	T4U	FTI	TBG	patient_id
count	9172.000000	8330.000000	6568.000000	8730.000000	8363.000000	8370.000000	349.000000	9.172000e+03
mean	73.555822	5.218403	1.970629	108.700305	0.976056	113.640746	29.870057	8.529473e+08
std	1183.976718	24.184006	0.887579	37.522670	0.200360	41.551650	21.080504	7.581969e+06
min	1.000000	0.005000	0.050000	2.000000	0.170000	1.400000	0.100000	8.408010e+08
25%	37.000000	0.460000	1.500000	87.000000	0.860000	93.000000	21.000000	8.504090e+08
50%	55.000000	1.400000	1.900000	104.000000	0.960000	109.000000	26.000000	8.510040e+08
75%	68.000000	2.700000	2.300000	126.000000	1.065000	128.000000	31.000000	8.607110e+08
max	65526.000000	530.000000	18.000000	600.000000	2.330000	881.000000	200.000000	8.701190e+08

Here, we see the basic stats of the numerical columns. This can be useful later, when trying to derive insights.
We can see that the stats are also printed for the patient id column, which is just a unique integer assigned to every patient.

Data Cleaning/Processing

Data cleaning is imperative to any data driven approach. For our dataset, we followed the following steps:

1. Imputing missing values
2. Outlier detection and imputing
3. Map and encode categorical columns (target)
4. Drop columns
5. Normalizing numerical columns

Imputing missing values

Initially we examine the status of the number of null values present in each column. This will give us a fair idea on which columns to focus on to impute values.

age	0
sex	307
on_thyroxine	0
query_on_thyroxine	0
on_antithyroid_meds	0
sick	0
pregnant	0
thyroid_surgery	0
I131_treatment	0
query_hypothyroid	0
query_hyperthyroid	0
lithium	0
goitre	0
tumor	0
hypopituitary	0
psych	0
TSH_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

For the column ‘sex’ we can do a simple median replacement, where we first categorically encode the genders, and impute the missing values with the median of the categorical values, and then map the genders back.

For the columns ‘TSH’, ‘T3’, ‘TT4’, ‘T4U’, ‘FTI’, ‘TBG’, which are observations we get from blood tests, there are also accompanying columns of whether they were measured or not. Using this, we infer that if the test was measured and the value was ‘t’, the value is present

in its numerical column, and if not, then no value is present. Thus, for these columns, we can impute 0 when the indicator column has the value ‘f’.

Outlier detection and imputing

Outliers are detected in numerical columns. We will take the standard threshold of 3 taking the plus/minus 2 standard deviations from the mean value.

After performing the outlier detection, we find the number of values replaced :

```
Count of replaced outliers in each column defaultdict(<class 'dict'>, {'TSH': 102, 'T3': 70, 'TT4': 75, 'T4U': 17, 'FTI': 68, 'TBG': 235})
```

These are accepted outlier values as the total count of the dataset is more than 9000.

Map/Encode target variables

The target variable is mapped as text values. We needed to deep dive into the intricacies of the hyper and hypo thyroidism to map the output to either hyper, hypo or negative classes.

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

Based on the above categorization, we are grouping the target column into 3 groups which are:

1. Hypothyroid conditions
2. Hypothyroid conditions
3. Negative

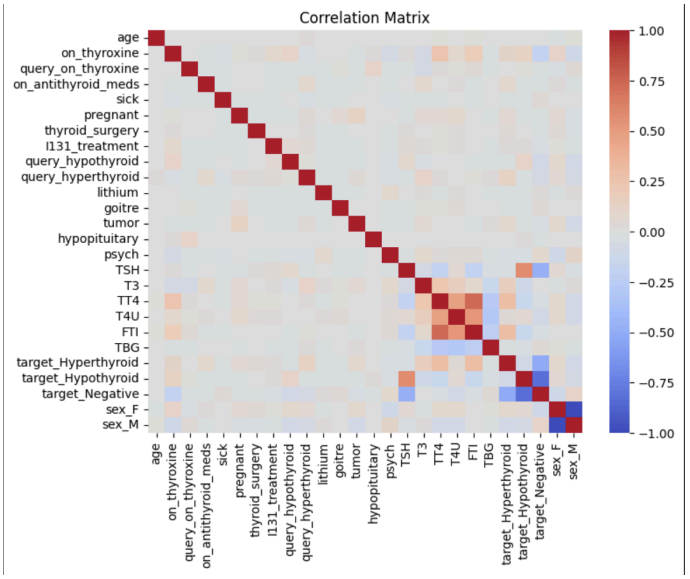
- Straight away - ABCD is mapped to hyperthyroid, and EFGH is mapped to hypothyroid.
- Rows with target J,R,S, and T have no variance in their attributes, and will hence not contribute to the downstream modeling.
- There was only one row with the target I, and that corresponds to Binding protein so it was dropped.
- For classes L and M, which come under replacement therapy, our research showed us that the test results in hypothyroidism.
- For class N, we found out that it maps to hyperthyroidism.
- For classes O,P, and Q, the research showed us that it maps to hyperthyroidism.
- For target variables where there were multiple classes, for ex. AK, GKJ or H|K, we assume that the diagnosis is the superseding class. For these examples, the diagnoses are class A, class G, class H, which is hyper, hypo, hypo respectively.

Dropping columns

Initially, we dropped the columns 'patient_id', 'referral_source', as they will not be involved in any kind of statistical modeling.

Then, when the data imputation of the columns ‘TSH’, ‘T3’, ‘TT4’, ‘T4U’, ‘FTI’, ‘TBG’ was done, now there was no need of the data existence indicator columns of ‘TSH_measured’, ‘T3_measured’, ‘TT4_measured’, ‘T4U_measured’, ‘FTI_measured’, ‘TBG_measured’, as they are now redundant. Hence, these columns were dropped.

Then we plotted a correlation matrix to find the the degree of variability between variables:



Here, we can see that the columns: 'age', 'query_on_thyroxine', 'sick', 'thyroid_surgery', 'I131_treatment', 'lithium', 'goitre', 'tumor', 'hypopituitary' have the least correlation with the rest of the variables, and hence were dropped.

Normalizing numerical columns

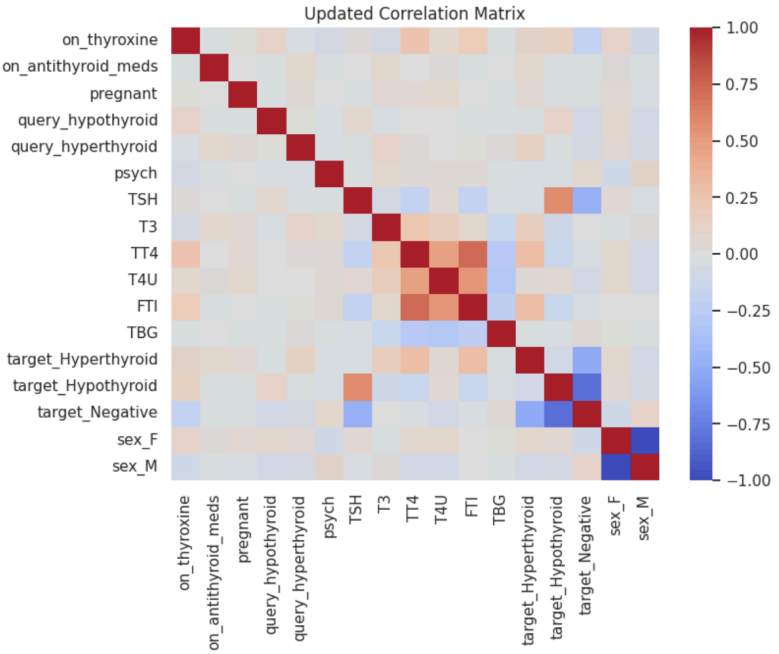
Numerical columns should be normalized to bring them all into a standard scale throughout the data. The scale we have chosen is 0 to 1. We normalize the columns by finding the maximum and minimum values and replacing the values with the formula:

(value - min value) / (max value - min value)

This brings the entire data into a standard 0-1 scale.

Exploratory Data Analysis (EDA)

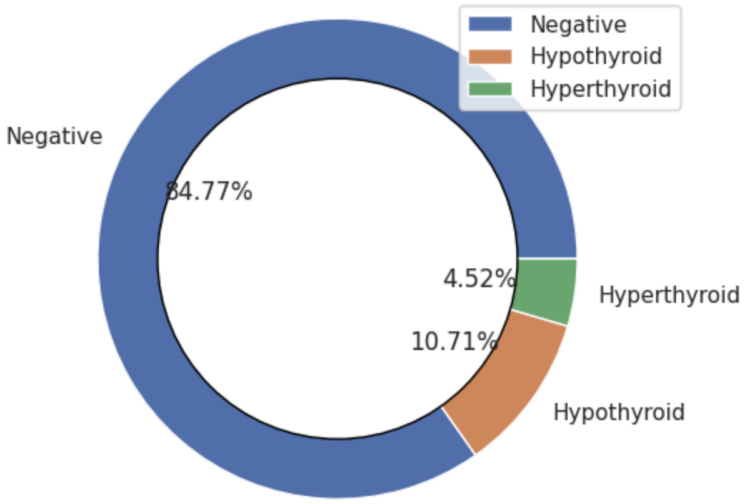
The first step of the EDA process has already been covered in the dropping columns section, where we use it to infer which features to drop. After dropping, we plotted another one.



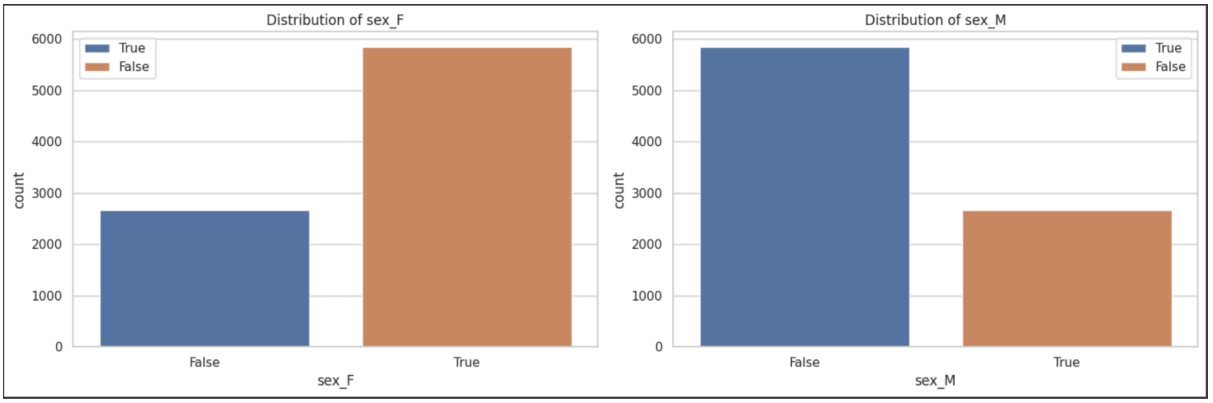
Here, we can see that all the remaining features have either a positive or a negative correlation. We can use this later while designing a model.

Another EDA process we did was feature engineering columns where the values are ‘t’ or ‘f’ strings. We replaced these values with boolean True and False values. This will help in the future mathematical modeling.

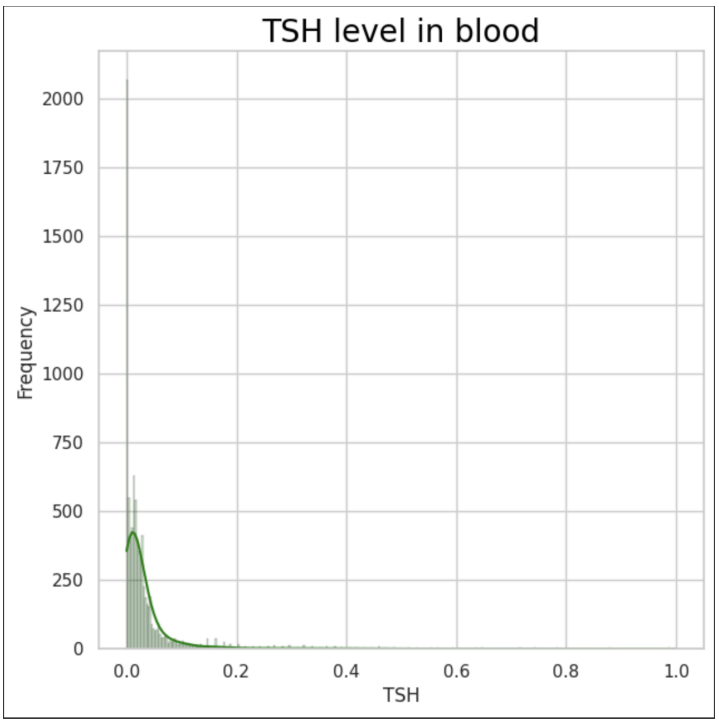
Now, let’s move on to some visualizing:



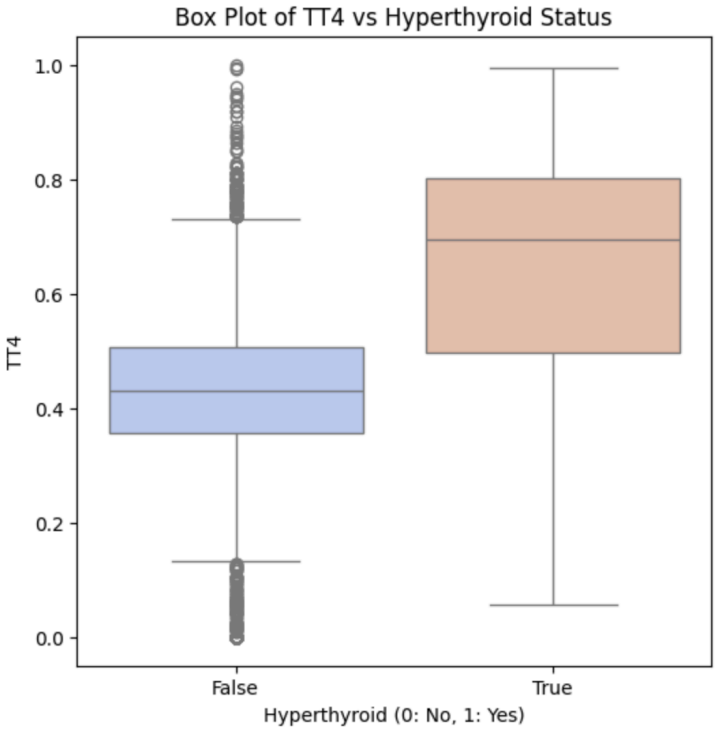
The above graph shows the distribution of classes in the target variable. We can see that most of the samples are classified as negative after the custom class mapping.



From these graphs, we can see a clear distribution where the dataset has more female samples than male samples. This shows there might be some bias in our findings later on in the models.

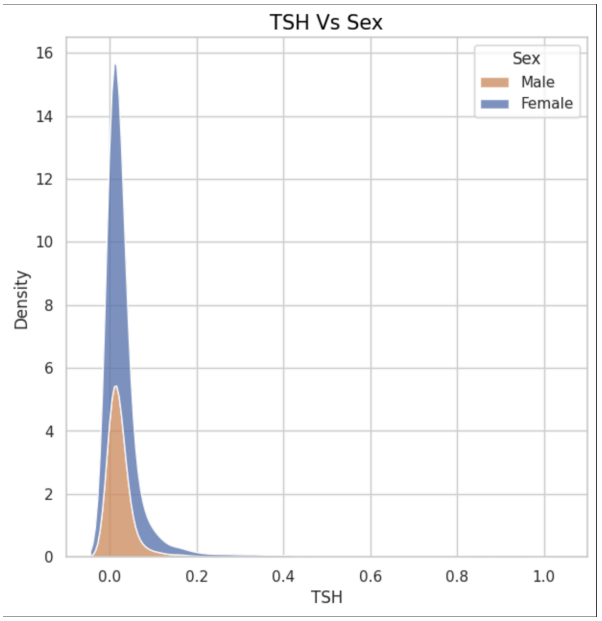


This graph shows the variance of the TSH in blood among the samples. We can see that this turns out to be a right skewed Bernoulli distribution, which suggests there could be some underlying factors/ bias affecting the levels.

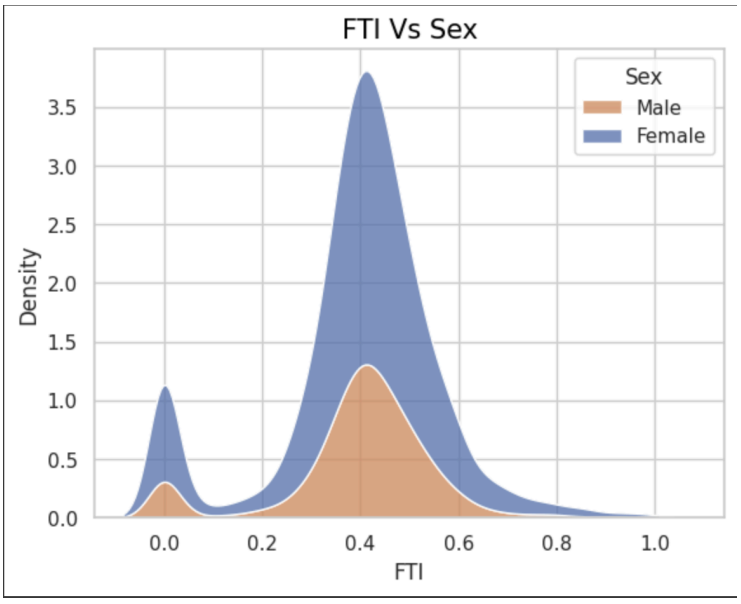


The median TT4 value is higher in individuals with hyperthyroidism (orange) compared to those without (blue). There are no outliers here, showing good normalization. This indicates that patients with hyperthyroidism generally have elevated levels of TT4.

In the non-hyperthyroid (blue) part of the box plot, we consider the other two classes, and see that the median is not skewed. The presence of outliers here show that there are abnormal levels of TT4 in their blood, which might indicate the presence of hypothyroidism or neither.



This graph shows the density distribution between the TSH compound and the sexes. We can see that it is higher for females, and also steeper. It is an indicator of high availability of the compound in females.



This graph shows the density distribution between the FTI compound and the sexes. It is observed that there are 2 peaks, where one can be considered a local maxima and the other can be the global maxima. It shows that the distribution varies evenly and similarly between males and females, with it being more prevalent in females.

Phase 2

Continuing on the data science pipeline, after the data preprocessing, EDA, we must apply various techniques to extract inferences from our data. The main motive is to derive insights, make predictions, or classify data points into different buckets.

Our problem statement involves performing multiclass classification between 3 classes in the dataset: Hyperthyroid, Hypothyroid, and Negative.

To align with this, we have chosen to apply 5 Machine Learning Models to the data. These are:

- Linear Regression
- Decision Tree
- Neural Network
- Gradient Boosting (XGBoost)
- Support Vector Machine

The following are the findings:

1. **Linear Regression**

   The rationale behind choosing this model:
   Linear regression offers a straightforward and efficient approach. Its simplicity and interpretability make it valuable for understanding the underlying relationships between features and the target classes. By training a model and examining the model's coefficients, we can gain valuable insights into which features are most influential in the classification decision. This knowledge can guide further analysis, feature engineering, or the selection of more complex models if necessary. Linear regression is the starting point for many machine learning models. In our case, this is a good starting point for us to implement as we have several independent features and only 3 target classes, and the prediction can come from a regression-based model.

   Evaluation metrics and results:

```
Training Loss: 0.0617
Training Accuracy: 0.8780

Test Results:
Test Accuracy: 0.8655
              precision    recall  f1-score   support

           0       0.00      0.00      0.00        85
           1       0.86      0.26      0.40       190
           2       0.87      1.00      0.93      1428

    accuracy                           0.87      1703
   macro avg       0.58      0.42      0.44      1703
weighted avg       0.82      0.87      0.82      1703
```

2. **Decision Tree**

   The rationale behind choosing this model:
   We chose Decision Trees because of their strong classification performance and their capability to model complex, non-linear relationships effectively. They are particularly well-suited for medical datasets like thyroid diagnosis, where understanding the rationale behind predictions is crucial. Decision Trees provide human-readable decision rules, making them highly interpretable and valuable for clinical decision-making. They also handle missing or imputed data efficiently and excel in capturing feature interactions, allowing us to identify the most significant factors influencing the diagnosis. This combination of high accuracy, interpretability, and reliability makes them a strong choice for healthcare applications.

   Evaluation metrics and results:

```
Accuracy: 0.9530240751614797
              precision    recall  f1-score   support

           0       0.71      0.69      0.70        85
           1       0.90      0.82      0.85       190
           2       0.97      0.99      0.98      1428

    accuracy                           0.95      1703
   macro avg       0.86      0.83      0.85      1703
weighted avg       0.95      0.95      0.95      1703
```

3. **Neural Network**

   The rationale behind choosing this model:
   We chose Neural Networks for the thyroid dataset because they excel at capturing complex, non-linear relationships and subtle patterns, which are crucial for accurate classification in medical problems like thyroid diagnosis. Their ability to learn deep feature representations allows them to identify intricate dependencies between features that may not be apparent with simpler models, leading to high accuracy in complex multiclass classification tasks. Additionally, Neural Networks are versatile, adaptable to a wide range of applications, and can handle large datasets effectively while minimizing overfitting with proper regularization. These advantages make them an excellent choice for extracting meaningful insights and delivering reliable predictions in healthcare scenarios.

   Evaluation metrics and results:

```
Epoch [45/50], Loss: 0.0197, Accuracy: 0.9213
Epoch [46/50], Loss: 0.0197, Accuracy: 0.9212
Epoch [47/50], Loss: 0.0197, Accuracy: 0.9206
Epoch [48/50], Loss: 0.0197, Accuracy: 0.9222
Epoch [49/50], Loss: 0.0197, Accuracy: 0.9220
Epoch [50/50], Loss: 0.0197, Accuracy: 0.9216
Test Accuracy: 0.9113
```

4. **XGBoost**

   The rationale behind choosing this model:
   We chose XGBoost for the thyroid dataset because it is a powerful and versatile

algorithm, particularly well-suited for structured data. Its ability to handle missing data, model complex relationships, and provide valuable feature importance insights makes it an ideal choice for medical applications. XGBoost's robust performance, efficiency, and scalability have made it a popular choice in machine learning competitions and real-world applications, including healthcare. By leveraging XGBoost's strengths, we can build a highly accurate and reliable thyroid diagnosis model that contributes to improved patient care and clinical decision-making.

Evaluation metrics and results:

```
Accuracy: 0.9757433489827856
              precision    recall  f1-score   support

           0       0.90      0.79      0.84        58
           1       0.93      0.93      0.93       135
           2       0.99      0.99      0.99      1085

    accuracy                           0.98      1278
   macro avg       0.94      0.90      0.92      1278
weighted avg       0.98      0.98      0.98      1278
```

5. **Support Vector Machine**
   To make it work for multiclass classification, we used the One-v-One strategy (OvO). It involves creating a binary classifier for every possible pair of classes.

   The rationale behind choosing this model:
   We chose the Support Vector Machine (SVM) for the thyroid dataset because it is effective at handling complex multiclass classification tasks, especially when the data has a clear margin of separation. SVM is good at finding optimal decision boundaries between classes, even in high-dimensional spaces, making it a strong choice for medical datasets where accuracy is crucial. Its ability to handle non-linear relationships through the kernel trick allows it to model complex patterns in the data. Additionally, SVM is robust to overfitting with proper regularization, ensuring reliable predictions. These advantages make SVM a good tool for accurate and efficient thyroid diagnosis.

   Evaluation metrics and results:

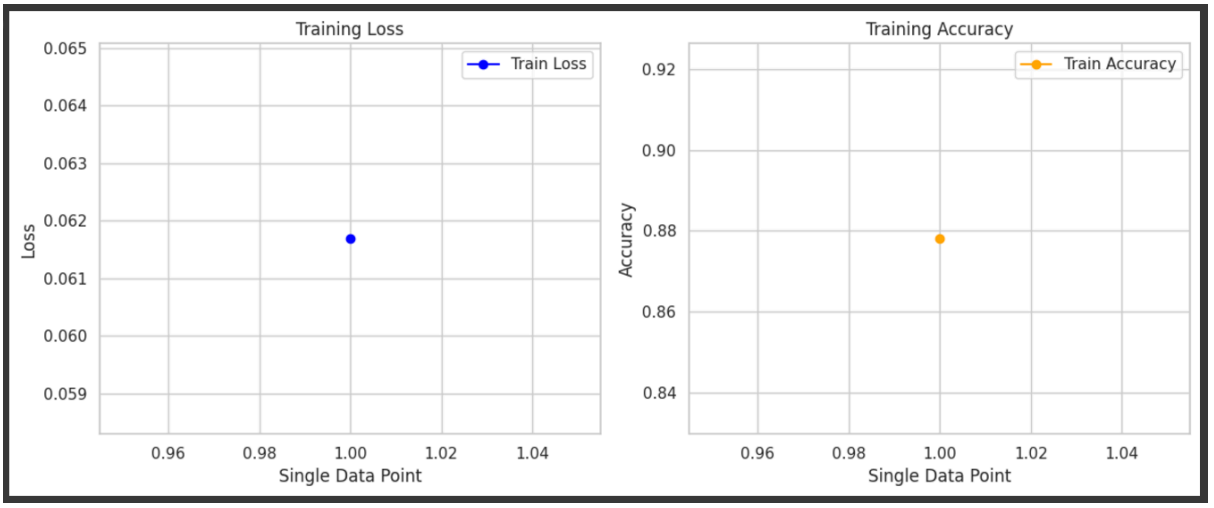
```
Test Accuracy: 0.8385202583675866
              precision    recall  f1-score   support

           0       0.00      0.00      0.00        85
           1       0.00      0.00      0.00       190
           2       0.84      1.00      0.91      1428

    accuracy                           0.84      1703
   macro avg       0.28      0.33      0.30      1703
weighted avg       0.70      0.84      0.76      1703
```

The best-performing model out of all the ones tried above is **XGBoost**

Visualisations

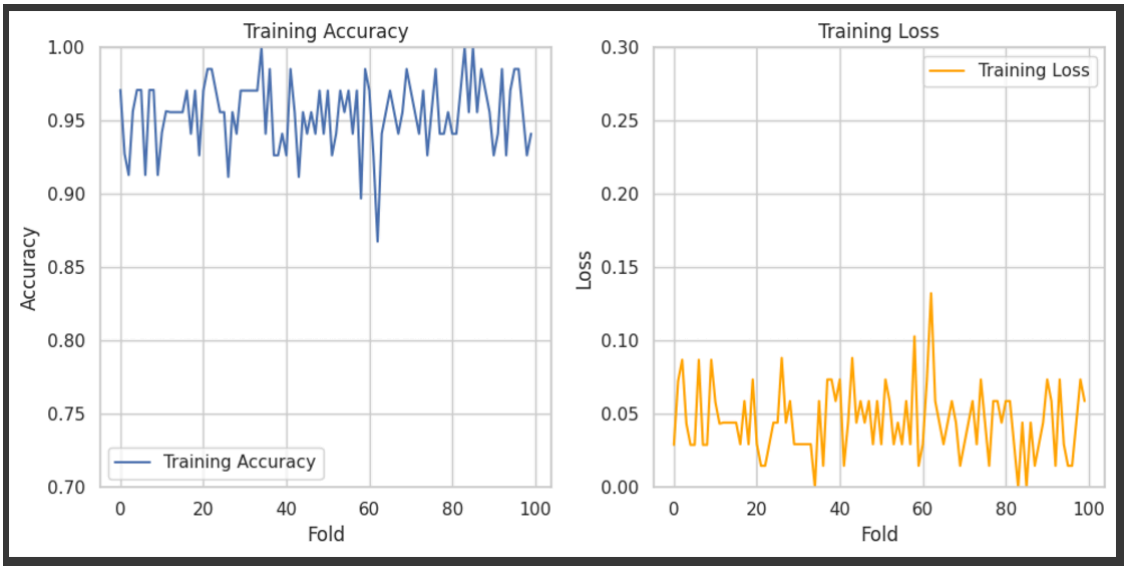
The following visualisations were plotted after the analysis of these different models:

- Linear Regression



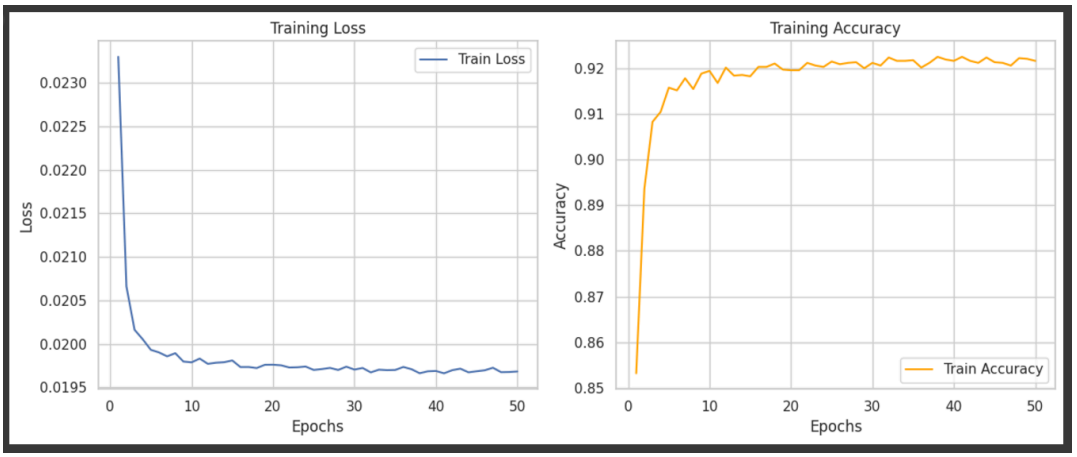
In this plot, we can see that the Training loss is reasonably low and the Training accuracy indicates that the model performs well with the training data i.e. it correctly predicts outputs for most training examples. This means that the overall model performance is considerably good.

- Decision Tree



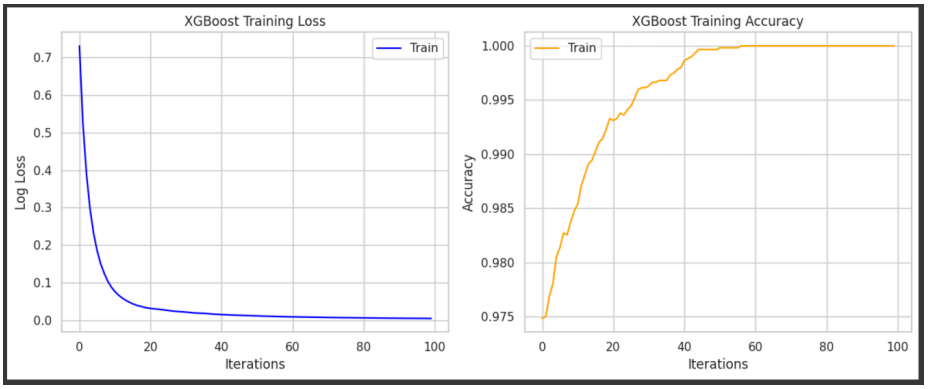
In the training accuracy plot, we can see that the values are high, mostly above 90%, with occasional dips around or below 80%. This variation in accuracy across folds suggests some fluctuations in model performance, likely due to differences in the training data used in each fold.
In the training loss plot, we can see that the loss is generally low (below 0.1 for most folds), with occasional spikes up to around 0.25. This variability reflects differences in how well the model fits each fold of the data.

- Neural Network



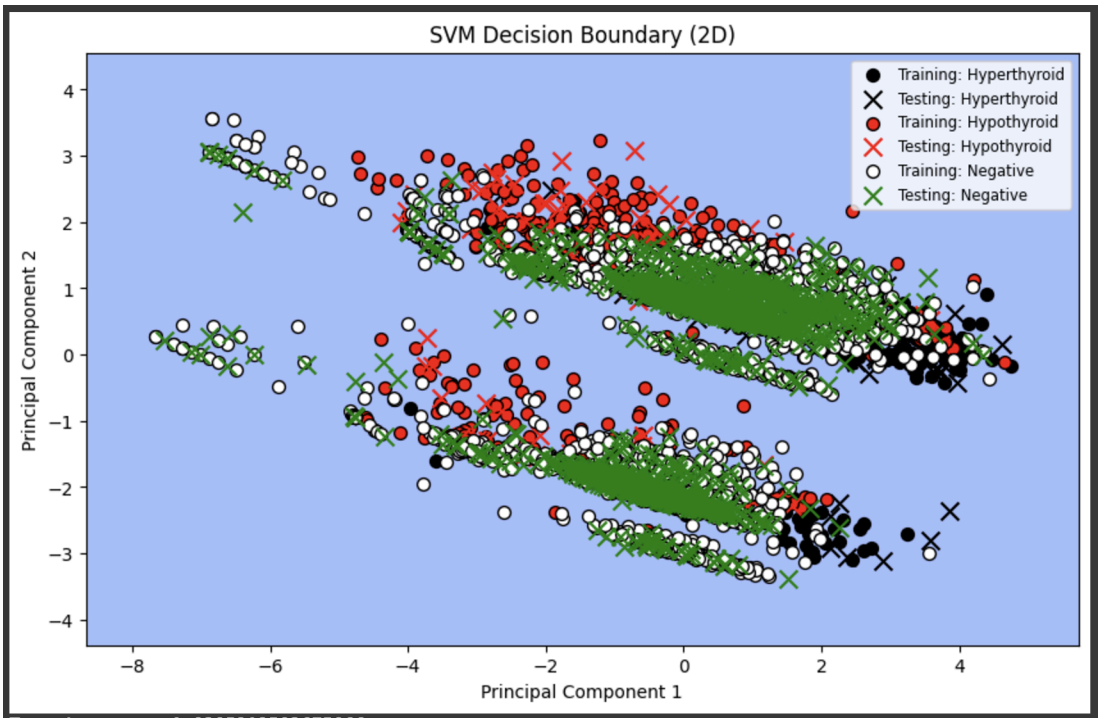
In the training Loss plot, we can see that the loss starts high (~0.023) and decreases rapidly within the first 10 epochs. After around 10 epochs, the loss stabilizes and converges near 0.0195, indicating the model has effectively minimized its error on the training data.
In the training accuracy plot, we see that the accuracy starts at ~85% and increases rapidly in the first few epochs, stabilizing around 92%. Beyond epoch 10, there is little to no noticeable improvement in accuracy, suggesting the model has learned the training data well.

- XGBoost



In the training loss plot, the loss function shows rapid convergence, dropping sharply from 0.7 to near 0.1 in the first 20 iterations, then steadily decreasing to almost 0 by iteration 100, indicating strong model confidence in its predictions.
In the training accuracy plot the accuracy increases rapidly from 0.975 to approximately 0.995 within the first 20 iterations, eventually reaching and maintaining near-perfect accuracy of 1.0 after 60 iterations, this learning curve suggests that the model has found an optimal solution without getting stuck in local minima.

- Support Vector Machine

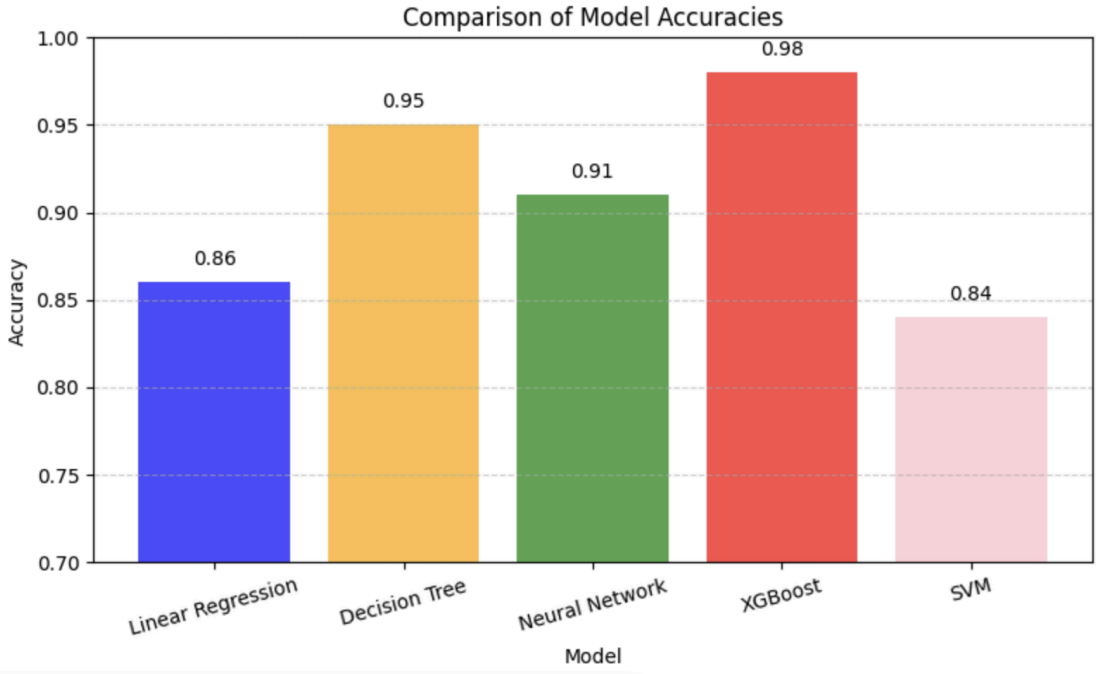


From this plot we can infer that the SVM has successfully captured the non-linear relationship between the three classes as there appears to be some overlap between classes in the central region where the bands converge.
We can also see that the model maintains consistent classification patterns for both training and testing data points.
To visualize the SVM, we used Principal Component Analysis to extract 2 primary components and then plotted a graph between them. We used PCA to reduce the dimensionality to 2 features so that the visualisation can be straightforward. Now using the legend on the graph, we can see the classification between training and testing samples for all 3 classes.

- Comparison of accuracies across models

MODEL	TRAINING LOSS	TRAINING ACCURACY	TEST ACCURACY	PRECISION	RECALL	F1-SCORE	OBSERVATION
Linear Regression	0.0617	0.8780	0.8655	0.58	0.42	0.44	Performed well overall but struggled with class imbalance.
Decision Tree	NA	NA	0.9530	0.86	0.83	0.85	Excellent performance with high precision, recall, and F1 scores. Also handled class imbalance effectively.
Neural Network	0.0197	0.9216	0.9113	NA	NA	NA	High training and test accuracy with just a slight gap between the two.
XGBoost	NA	NA	0.9757	0.94	0.90	0.92	Outstanding performance, especially for class 2. Minimal error, and handled class imbalance effectively.
Support Vector Machine	NA	NA	0.8385	0.28	0.33	0.30	Poor performance for minority classes (0 and 1) and a strong bias toward the majority class (2).

Visual Representation:



As we can see from the graph plot and the table, the best-performing model to predict thyroid cases is **XGBoost**.

REFERENCES

- **Linear Regression:** https://scikit-learn.org/dev/modules/generated/sklearn.linear_model.LinearRegression.html
- **Decision Tree:** https://scikit-learn.org/dev/modules/generated/sklearn.tree.DecisionTreeClassifier.html
- **Pytorch:** https://pytorch.org/docs/stable/index.html
- **XGBoost:** https://xgboost.readthedocs.io/en/stable/python/python_api.html
- **SVM:** https://scikit-learn.org/1.5/modules/generated/sklearn.svm.SVC.html#sklearn.svm.SVC
- **PCA:** https://scikit-learn.org/dev/modules/generated/sklearn.decomposition.PCA.html