Machine learning algorithm for detecting thyroid-related disorders to increase thyroid illness diagnosis accuracy

Advance Machine Learning / Spring 2024 / Prof. Ashok Kumar Patel

By Group – 11

Ajeet Singh – 48
Manas Vishal – 58
Rohith Kumar – 13

Problem Statement:

Machine learning techniques have been increasingly used in the medical field to improve the accuracy of diagnoses and treatment plans. The use of an ML algorithm for detecting thyroid-related disorders to increase thyroid illness diagnosis accuracy. This approach involves using data-based decisions for diagnosing thyroid dysfunction, which is a classification problem that can be solved using Machine Learning techniques.

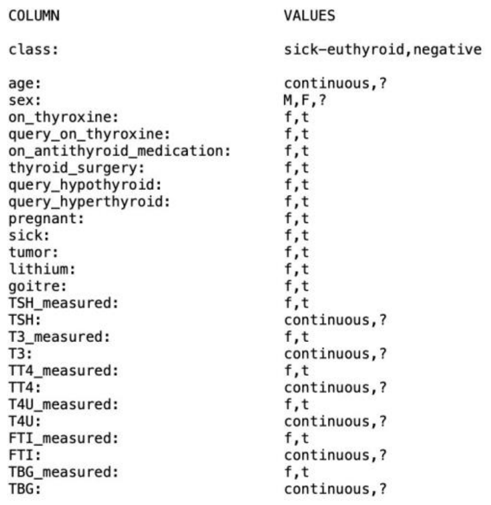
About the Datasets:

The dataset contains records of thyroid diagnoses from the Garvan Institute, collected between 1984 and early 1987. Each record consists of 25 attribute values, followed by a class. The attributes include various medical conditions, patient characteristics, and laboratory measurements related to thyroid function. The class contains two possible values: negative and sick-euthyroid.

Observations: 3163

Features: 25

Class: 1 (with two distinct values)



COLUMN	VALUES
class:	sick-euthyroid,negative
age:	continuous,?
sex:	M,F,?
on_thyroxine:	f,t
query_on_thyroxine:	f,t
on_antithyroid_medication:	f,t
thyroid_surgery:	f,t
query_hypothyroid:	f,t
query_hyperthyroid:	f,t
pregnant:	f,t
sick:	f,t
tumor:	f,t
lithium:	f,t
goitre:	f,t
TSH_measured:	f,t
TSH:	continuous,?
T3_measured:	f,t
T3:	continuous,?
TT4_measured:	f,t
TT4:	continuous,?
T4U_measured:	f,t
T4U:	continuous,?
FTI_measured:	f,t
FTI:	continuous,?
TBG_measured:	f,t
TBG:	continuous,?

Source:

Quinlan, Ross. (1987). Thyroid Disease. UCI Machine Learning Repository. https://doi.org/10.24432/C5D010.

Solution Approach:

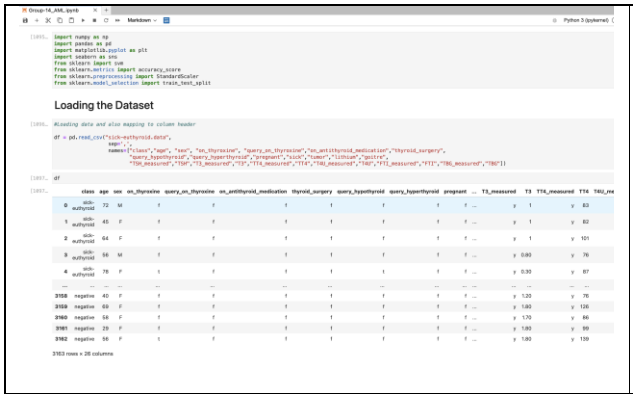
Given the complexity and the nature of the thyroid disease dataset, which includes a mix of continuous and categorical variables, and considering the need for accurate prediction of multiple classes of thyroid conditions, a combination of machine learning algorithms would be beneficial. The choice of algorithms should be based on their ability to handle both types of data and their performance in classification tasks, especially in the context of imbalanced classes.

1. Random Forest (RF): Random Forest is a powerful ensemble learning method that can handle both continuous and categorical data. It is known for its ability to avoid overfitting and its robustness to outliers.
2. Gradient Boosting Machine (GBM): GBM is another ensemble method that builds a series of weak learners (typically decision trees) to form a strong predictive model. It is particularly effective in handling imbalanced datasets and can capture complex patterns in the data.
3. Support Vector Machine (SVM): SVMs are effective in high-dimensional spaces and are versatile as different Kernel functions can be specified for the decision function. They are particularly useful when the dataset is not linearly separable.
4. AdaBoost: AdaBoost is an ensemble method that combines multiple weak learners to create a strong learner. It is effective in handling noisy data and can improve the accuracy of the model.
5. Deep Learning Models (ANN): For a more complex and nuanced understanding of the data, deep learning models like ANN networks can be used. These models can capture complex patterns and dependencies in the data, which might be beneficial for predicting thyroid conditions.

It would be beneficial to experiment with these algorithms, possibly in combination, and use cross-validation to evaluate their performance. The choice of the best algorithm or combination of algorithms would depend on the specific characteristics of the dataset and the performance metrics (accuracy, precision, recall, etc.) that are most relevant to the problem at hand.

Loading Dataset:

Load the csv file of thyroid dataset. Since the file does not have columns names and so we need to specify while loading it into dataframe. It has 25 features, 1 class and 3163 observations. It does not show any missing values initially, but we do see some unknow values like '?'.



Group-14_AML.ipynb

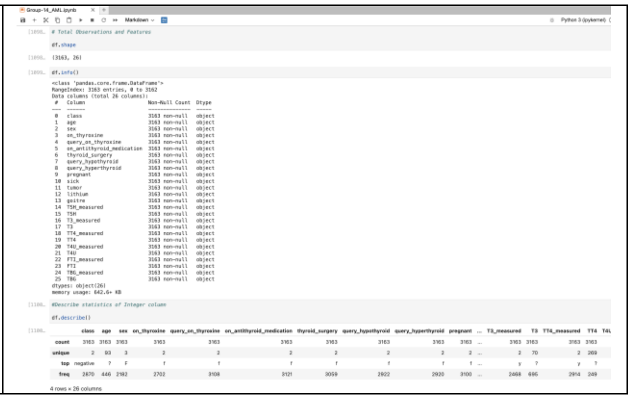
```
[1095… import numpy as np
       import pandas as pd
       import matplotlib.pyplot as plt
       import seaborn as sns
       from sklearn import svm
       from sklearn.metrics import accuracy_score
       from sklearn.preprocessing import StandardScaler
       from sklearn.model_selection import train_test_split
```

Loading the Dataset

```
[1096… #Loading data and also mapping to column header

       df = pd.read_csv("sick-euthyroid.data",
                        sep=',',
                        names=["class","age", "sex", "on_thyroxine", "query_on_thyroxine","on_antithyroid_medication","thyroid_surgery",
                               "query_hypothyroid","query_hyperthyroid","pregnant","sick","tumor","lithium","goitre",
                               "TSH_measured","TSH","T3_measured","T3","TT4_measured","TT4","T4U_measured","T4U","FTI_measured","FTI","TBG_measured","TBG"])

[1097… df
```

	class	age	sex	on_thyroxine	query_on_thyroxine	on_antithyroid_medication	thyroid_surgery	query_hypothyroid	query_hyperthyroid	pregnant	...	T3_measured	T3	TT4_measured	TT4	T4U_me
0	sick-euthyroid	72	M	f	f	f	f	f	f	f	...	y	1	y	83	
1	sick-euthyroid	45	F	f	f	f	f	f	f	f	...	y	1	y	82	
2	sick-euthyroid	64	F	f	f	f	f	f	f	f	...	y	1	y	101	
3	sick-euthyroid	56	M	f	f	f	f	f	f	f	...	y	0.80	y	76	
4	sick-euthyroid	78	F	t	f	f	f	t	f	f	...	y	0.30	y	87	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
3158	negative	40	F	f	f	f	f	f	f	f	...	y	1.20	y	76	
3159	negative	69	F	f	f	f	f	f	f	f	...	y	1.80	y	126	
3160	negative	58	F	f	f	f	f	f	f	f	...	y	1.70	y	86	
3161	negative	29	F	f	f	f	f	f	f	f	...	y	1.80	y	99	
3162	negative	56	F	t	f	f	f	f	f	f	...	y	1.80	y	139	

3163 rows × 26 columns



Group-14_AML.ipynb

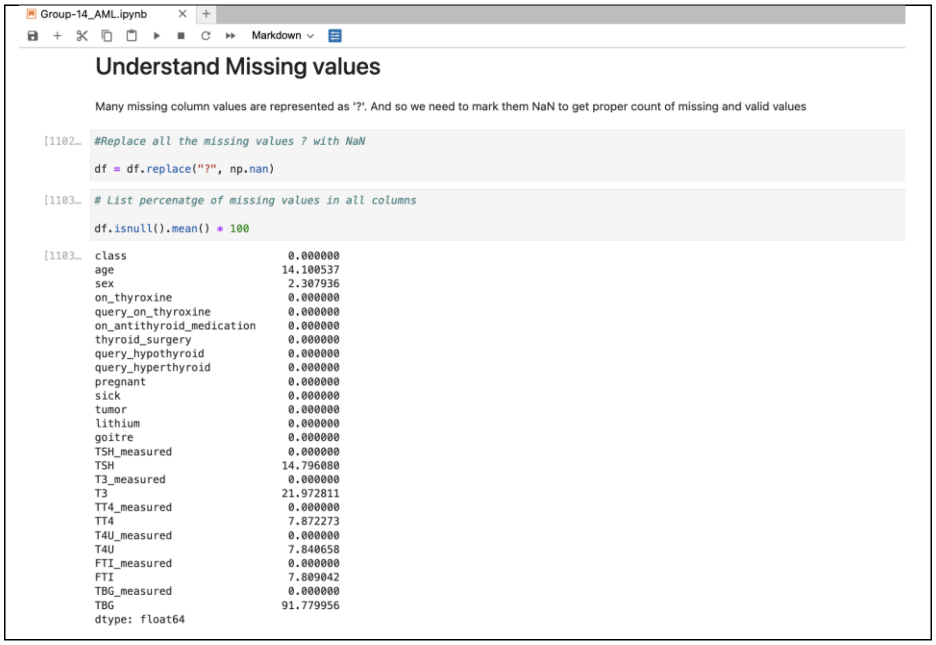
```
[1098… # Total Observations and Features

       df.shape

[1098… (3163, 26)

[1099… df.info()

       <class 'pandas.core.frame.DataFrame'>
       RangeIndex: 3163 entries, 0 to 3162
       Data columns (total 26 columns):
        #   Column                     Non-Null Count  Dtype
       ---  ------                     --------------  -----
        0   class                      3163 non-null   object
        1   age                        3163 non-null   object
        2   sex                        3163 non-null   object
        3   on_thyroxine               3163 non-null   object
        4   query_on_thyroxine         3163 non-null   object
        5   on_antithyroid_medication  3163 non-null   object
        6   thyroid_surgery            3163 non-null   object
        7   query_hypothyroid          3163 non-null   object
        8   query_hyperthyroid         3163 non-null   object
        9   pregnant                   3163 non-null   object
        10  sick                       3163 non-null   object
        11  tumor                      3163 non-null   object
        12  lithium                    3163 non-null   object
        13  goitre                     3163 non-null   object
        14  TSH_measured               3163 non-null   object
        15  TSH                        3163 non-null   object
        16  T3_measured                3163 non-null   object
        17  T3                         3163 non-null   object
        18  TT4_measured               3163 non-null   object
        19  TT4                        3163 non-null   object
        20  T4U_measured               3163 non-null   object
        21  T4U                        3163 non-null   object
        22  FTI_measured               3163 non-null   object
        23  FTI                        3163 non-null   object
        24  TBG_measured               3163 non-null   object
        25  TBG                        3163 non-null   object
       dtypes: object(26)
       memory usage: 642.6+ KB

[1100… #Describe statistics of Integer column

       df.describe()
```

	class	age	sex	on_thyroxine	query_on_thyroxine	on_antithyroid_medication	thyroid_surgery	query_hypothyroid	query_hyperthyroid	pregnant	...	T3_measured	T3	TT4_measured	TT4	T4U
count	3163	3163	3163	3163	3163	3163	3163	3163	3163	3163	...	3163	3163	3163	3163	
unique	2	93	3	2	2	2	2	2	2	2	...	2	70	2	269	
top	negative	?	F	f	f	f	f	f	f	f	...	y	?	y	?	
freq	2870	446	2182	2702	3108	3121	3059	2922	2920	3100	...	2468	695	2914	249	

4 rows × 26 columns

Understanding Missing Values:

We need to replace the unknow values '?' with NaN so that we can understand correct stats of the missing numbers. And as you can see in below percentages of missing values, there are some missing values in age, sex, TSH, T3, TT4, T4U, FTI and TBG. But class values do not have any missing values which is good.



Group-14_AML.ipynb

Understand Missing values

Many missing column values are represented as '?'. And so we need to mark them NaN to get proper count of missing and valid values

```
[1102… #Replace all the missing values ? with NaN

       df = df.replace("?", np.nan)

[1103… # List percenatge of missing values in all columns

       df.isnull().mean() * 100

[1103… class                         0.000000
       age                          14.100537
       sex                           2.307936
       on_thyroxine                  0.000000
       query_on_thyroxine            0.000000
       on_antithyroid_medication     0.000000
       thyroid_surgery               0.000000
       query_hypothyroid             0.000000
       query_hyperthyroid            0.000000
       pregnant                      0.000000
       sick                          0.000000
       tumor                         0.000000
       lithium                       0.000000
       goitre                        0.000000
       TSH_measured                  0.000000
       TSH                          14.796080
       T3_measured                   0.000000
       T3                           21.972811
       TT4_measured                  0.000000
       TT4                           7.872273
       T4U_measured                  0.000000
       T4U                           7.840658
       FTI_measured                  0.000000
       FTI                           7.809042
       TBG_measured                  0.000000
       TBG                          91.779956
       dtype: float64
```

Handling Missing Values and Encoding Categorical Values:

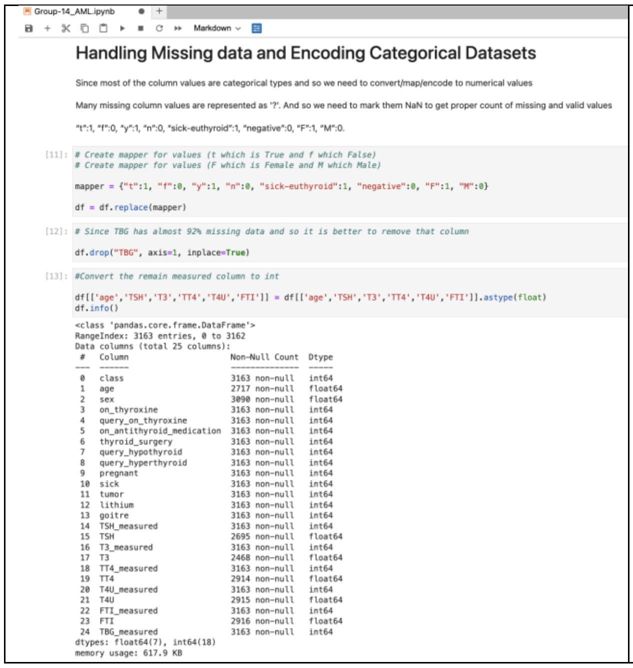
- We can remove the TBG column as it has 91.77 % of missing values and will not be helpful for training model.
- All other missing values should be replaced with the Mean value of that column.
- Since most of the column values are categorical types and so we need to convert/map/encode to numerical values as listed below:

  'sick-euthyroid': 1
  'negative': 0

  't': 1
  'f': 0
  'y': 1
  'n': 0

  'F': 1
  'M': 0

- Convert the remaining measured features from Object to Float



Handling Missing data and Encoding Categorical Datasets

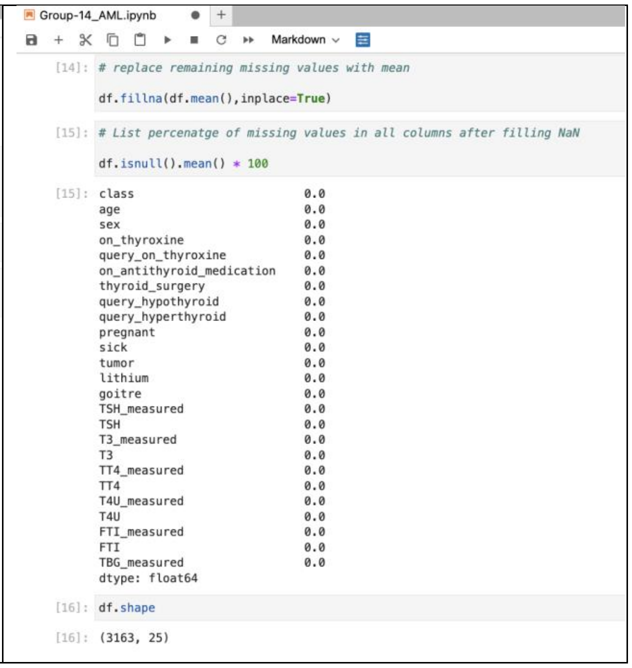
Since most of the column values are categorical types and so we need to convert/map/encode to numerical values

Many missing column values are represented as '?'. And so we need to mark them NaN to get proper count of missing and valid values

"t":1, "f":0, "y":1, "n":0, "sick-euthyroid":1, "negative":0, "F":1, "M":0.

```
[11]: # Create mapper for values (t which is True and f which False)
      # Create mapper for values (F which is Female and M which Male)

      mapper = {"t":1, "f":0, "y":1, "n":0, "sick-euthyroid":1, "negative":0, "F":1, "M":0}

      df = df.replace(mapper)

[12]: # Since TBG has almost 92% missing data and so it is better to remove that column

      df.drop("TBG", axis=1, inplace=True)

[13]: #Convert the remain measured column to int

      df[['age','TSH','T3','TT4','T4U','FTI']] = df[['age','TSH','T3','TT4','T4U','FTI']].astype(float)
      df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3163 entries, 0 to 3162
Data columns (total 25 columns):
 #   Column                     Non-Null Count  Dtype
---  ------                     --------------  -----
 0   class                      3163 non-null   int64
 1   age                        2717 non-null   float64
 2   sex                        3090 non-null   float64
 3   on_thyroxine               3163 non-null   int64
 4   query_on_thyroxine         3163 non-null   int64
 5   on_antithyroid_medication  3163 non-null   int64
 6   thyroid_surgery            3163 non-null   int64
 7   query_hypothyroid          3163 non-null   int64
 8   query_hyperthyroid         3163 non-null   int64
 9   pregnant                   3163 non-null   int64
 10  sick                       3163 non-null   int64
 11  tumor                      3163 non-null   int64
 12  lithium                    3163 non-null   int64
 13  goitre                     3163 non-null   int64
 14  TSH_measured               3163 non-null   int64
 15  TSH                        2695 non-null   float64
 16  T3_measured                3163 non-null   int64
 17  T3                         2468 non-null   float64
 18  TT4_measured               3163 non-null   int64
 19  TT4                        2914 non-null   float64
 20  T4U_measured               3163 non-null   int64
 21  T4U                        2915 non-null   float64
 22  FTI_measured               3163 non-null   int64
 23  FTI                        2916 non-null   float64
 24  TBG_measured               3163 non-null   int64
dtypes: float64(7), int64(18)
memory usage: 617.9 KB
```



```
[14]: # replace remaining missing values with mean

      df.fillna(df.mean(),inplace=True)

[15]: # List percenatge of missing values in all columns after filling NaN

      df.isnull().mean() * 100
```

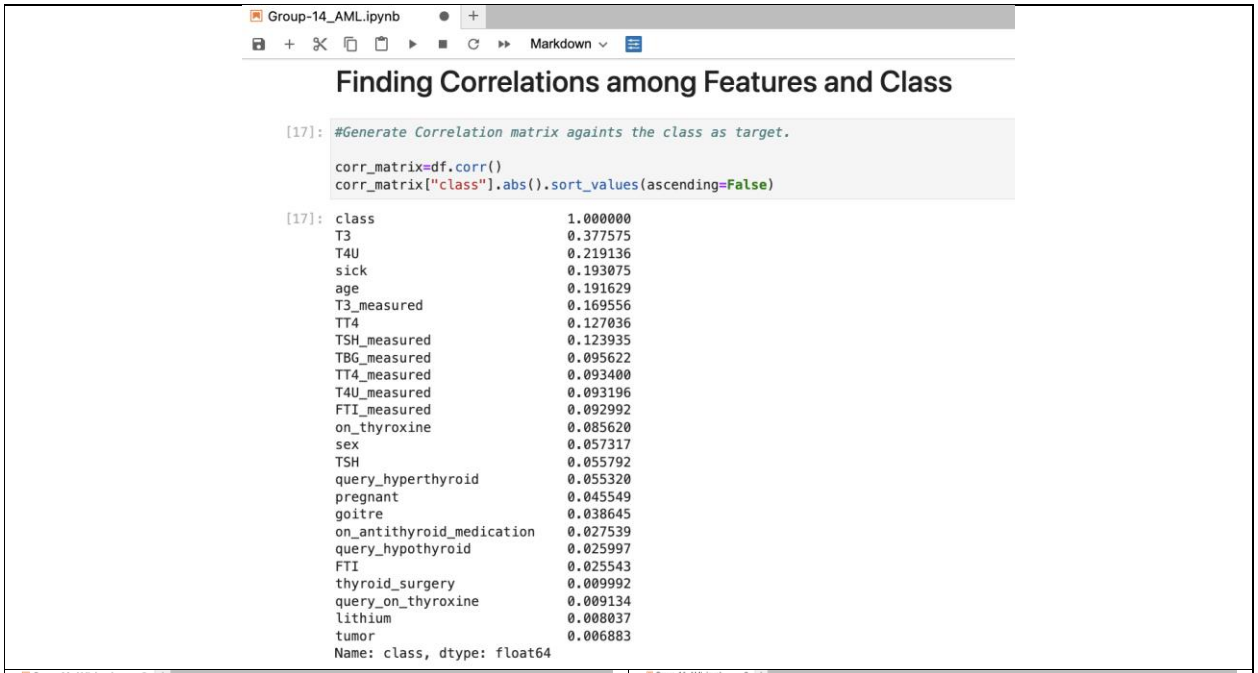
```
[15]: class                        0.0
      age                          0.0
      sex                          0.0
      on_thyroxine                 0.0
      query_on_thyroxine           0.0
      on_antithyroid_medication    0.0
      thyroid_surgery              0.0
      query_hypothyroid            0.0
      query_hyperthyroid           0.0
      pregnant                     0.0
      sick                         0.0
      tumor                        0.0
      lithium                      0.0
      goitre                       0.0
      TSH_measured                 0.0
      TSH                          0.0
      T3_measured                  0.0
      T3                           0.0
      TT4_measured                 0.0
      TT4                          0.0
      T4U_measured                 0.0
      T4U                          0.0
      FTI_measured                 0.0
      FTI                          0.0
      TBG_measured                 0.0
      dtype: float64
```

```
[16]: df.shape

[16]: (3163, 25)
```

Finding Correlations between Features and Class:

- There is strong correlation for T3, T4U, sick, age, T3_measured, TT4, TSH_measured, TBG_measured.

- Some of the features are not very well correlated. like, Tumor, lithium, thyroid_surgery, query_on_thyroxine, FTI, query_hypothyroid, on_antothyroid_medication, goitre.

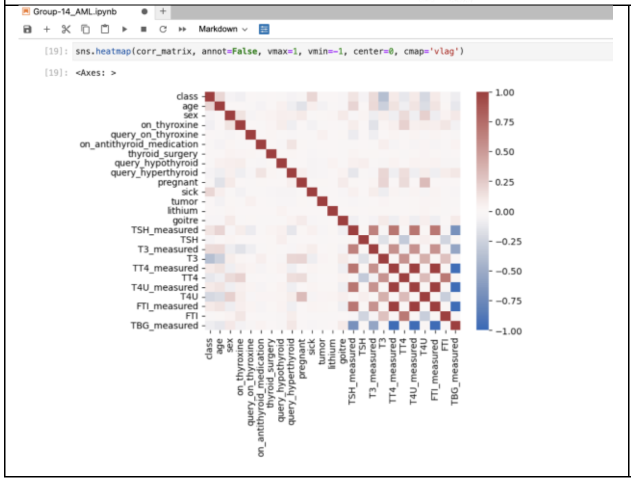


Group-14_AML.ipynb

Finding Correlations among Features and Class

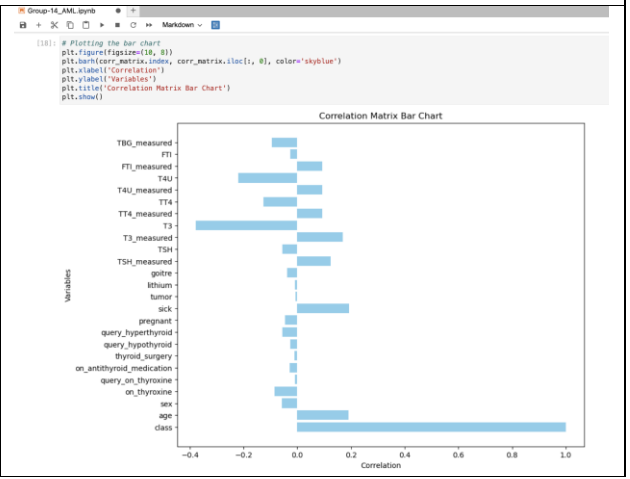
```
[17]: #Generate Correlation matrix againts the class as target.

      corr_matrix=df.corr()
      corr_matrix["class"].abs().sort_values(ascending=False)
```

```
[17]: class                        1.000000
      T3                           0.377575
      T4U                          0.219136
      sick                         0.193075
      age                          0.191629
      T3_measured                  0.169556
      TT4                          0.127036
      TSH_measured                 0.123935
      TBG_measured                 0.095622
      TT4_measured                 0.093400
      T4U_measured                 0.093196
      FTI_measured                 0.092992
      on_thyroxine                 0.085620
      sex                          0.057317
      TSH                          0.055792
      query_hyperthyroid           0.055320
      pregnant                     0.045549
      goitre                       0.038645
      on_antithyroid_medication    0.027539
      query_hypothyroid            0.025997
      FTI                          0.025543
      thyroid_surgery              0.009992
      query_on_thyroxine           0.009134
      lithium                      0.008037
      tumor                        0.006883
      Name: class, dtype: float64
```



Group-14_AML.ipynb

```
[19]: sns.heatmap(corr_matrix, annot=False, vmax=1, vmin=-1, center=0, cmap='vlag')

[19]: <Axes: >
```



Group-14_AML.ipynb

```
[18]: # Plotting the bar chart
      plt.figure(figsize=(10, 8))
      plt.barh(corr_matrix.index, corr_matrix.iloc[:, 0], color='skyblue')
      plt.xlabel('Correlation')
      plt.ylabel('Variables')
      plt.title('Correlation Matrix Bar Chart')
      plt.show()
```

Dividing Datasets into Training and Testing sets:

- We divided datasets into Train set and Test set. Making sure each set represent proper proportion of each Classes

  Train Set: (2530, 25)
  0 2296
  1 234

  Test Set: (633, 25)
  0 574
  1 59

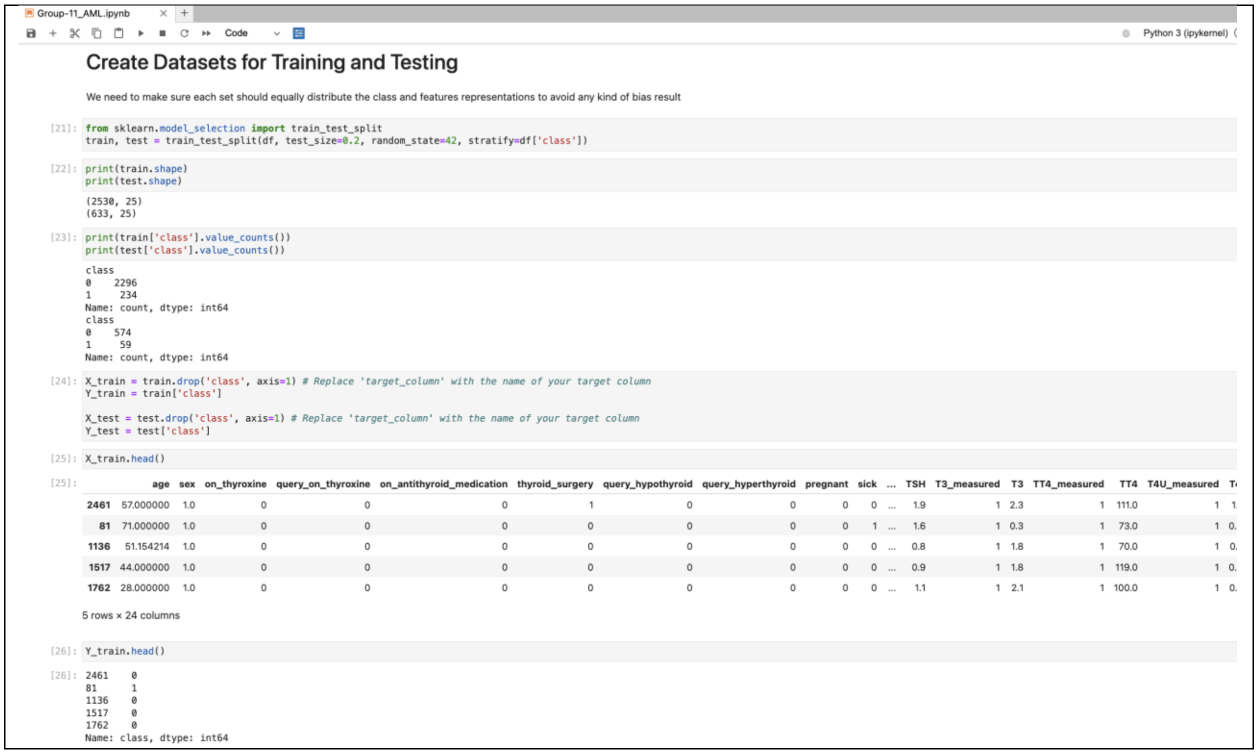
- Created X and Y variables for each dataset to use into Model Function

  X_train = hold all features values for train set

  Y_train = hold only class values for train set

  X_test = hold all features values for test set

  Y_test = hold only class values for test set



Create Datasets for Training and Testing

We need to make sure each set should equally distribute the class and features representations to avoid any kind of bias result

```
[21]: from sklearn.model_selection import train_test_split
      train, test = train_test_split(df, test_size=0.2, random_state=42, stratify=df['class'])
```

```
[22]: print(train.shape)
      print(test.shape)
```

```
(2530, 25)
(633, 25)
```

```
[23]: print(train['class'].value_counts())
      print(test['class'].value_counts())
```

```
class
0    2296
1     234
Name: count, dtype: int64
class
0    574
1     59
Name: count, dtype: int64
```

```
[24]: X_train = train.drop('class', axis=1) # Replace 'target_column' with the name of your target column
      Y_train = train['class']

      X_test = test.drop('class', axis=1) # Replace 'target_column' with the name of your target column
      Y_test = test['class']
```

```
[25]: X_train.head()
```

[25]:

	age	sex	on_thyroxine	query_on_thyroxine	on_antithyroid_medication	thyroid_surgery	query_hypothyroid	query_hyperthyroid	pregnant	sick	...	TSH	T3_measured	T3	TT4_measured	TT4	T4U_measured	T
2461	57.000000	1.0	0	0	0	1	0	0	0	0	...	1.9	1	2.3	1	111.0	1	1.
81	71.000000	1.0	0	0	0	0	0	0	0	1	...	1.6	1	0.3	1	73.0	1	0.
1136	51.154214	1.0	0	0	0	0	0	0	0	0	...	0.8	1	1.8	1	70.0	1	0.
1517	44.000000	1.0	0	0	0	0	0	0	0	0	...	0.9	1	1.8	1	119.0	1	0.
1762	28.000000	1.0	0	0	0	0	0	0	0	0	...	1.1	1	2.1	1	100.0	1	0.

5 rows × 24 columns

```
[26]: Y_train.head()
```

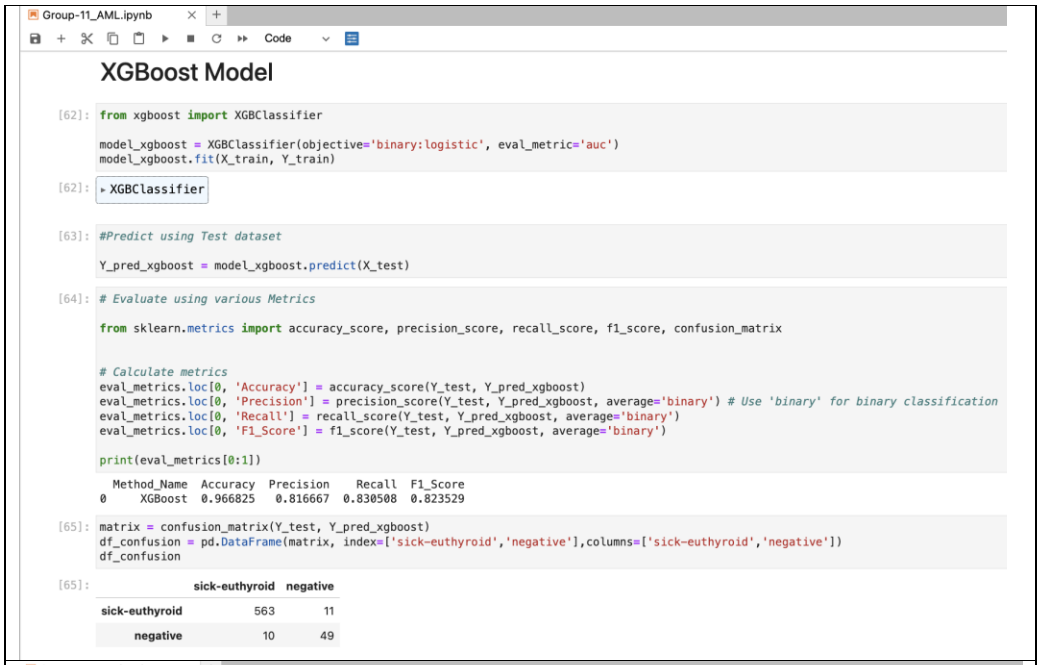
```
[26]: 2461    0
      81      1
      1136    0
      1517    0
      1762    0
      Name: class, dtype: int64
```

Training Using different ML Models:

- We will train and test datasets using different models as listed below. And will capture the evaluation metrics for comparison and understand which model is better.
    - Network Model
        - ANN
    - Statistical Model
        - SVC
    - Tree Based Model
        - XGBoost
        - GBM
        - Random Forest
        - AdaBoost

- We are going to follow the steps below to train/test/evaluate for each model.

    - Import required model's library
    - Train Model using Training set
        - trained_model = model.fit(X_train,Y_Train)
    - Predict Model using Testing set
        - Y_predict = trained_model.predict(X_test)
    - Evaluate Model Performance
        - accuracy_score(Y_test, Y_predict)
        - precision_score(Y_test, Y_predict)
        - recall_score(Y_test, Y_predict)
        - f1_score(Y_test, Y_predict)
    - Also create Confusion Matrix and plot a heatmap
        - confusion_matrix(Y_test, Y_predict)



XGBoost Model

[62]:
```
from xgboost import XGBClassifier

model_xgboost = XGBClassifier(objective='binary:logistic', eval_metric='auc')
model_xgboost.fit(X_train, Y_train)
```

[62]:
```
▸ XGBClassifier
```

[63]:
```
#Predict using Test dataset

Y_pred_xgboost = model_xgboost.predict(X_test)
```

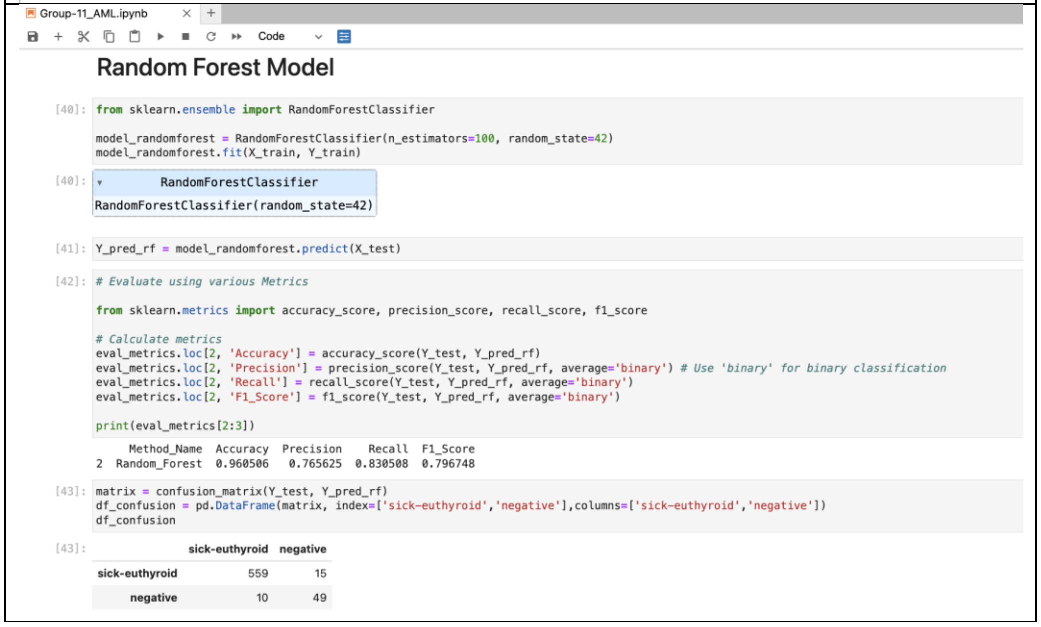
[64]:
```
# Evaluate using various Metrics

from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix


# Calculate metrics
eval_metrics.loc[0, 'Accuracy'] = accuracy_score(Y_test, Y_pred_xgboost)
eval_metrics.loc[0, 'Precision'] = precision_score(Y_test, Y_pred_xgboost, average='binary') # Use 'binary' for binary classification
eval_metrics.loc[0, 'Recall'] = recall_score(Y_test, Y_pred_xgboost, average='binary')
eval_metrics.loc[0, 'F1_Score'] = f1_score(Y_test, Y_pred_xgboost, average='binary')

print(eval_metrics[0:1])
```

```
  Method_Name  Accuracy  Precision    Recall  F1_Score
0     XGBoost  0.966825   0.816667  0.830508  0.823529
```

[65]:
```
matrix = confusion_matrix(Y_test, Y_pred_xgboost)
df_confusion = pd.DataFrame(matrix, index=['sick-euthyroid','negative'],columns=['sick-euthyroid','negative'])
df_confusion
```

[65]:

	sick-euthyroid	negative
sick-euthyroid	563	11
negative	10	49



Random Forest Model

[40]:
```
from sklearn.ensemble import RandomForestClassifier

model_randomforest = RandomForestClassifier(n_estimators=100, random_state=42)
model_randomforest.fit(X_train, Y_train)
```

[40]:
```
▾        RandomForestClassifier
RandomForestClassifier(random_state=42)
```

[41]:
```
Y_pred_rf = model_randomforest.predict(X_test)
```

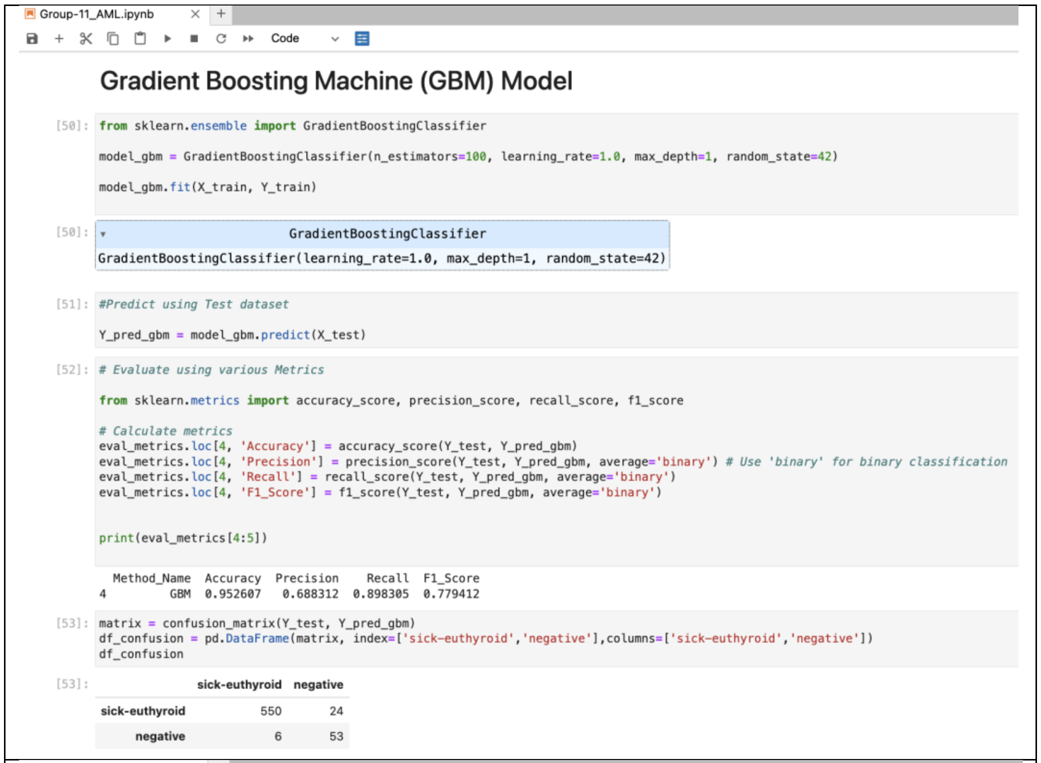
[42]:
```
# Evaluate using various Metrics

from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

# Calculate metrics
eval_metrics.loc[2, 'Accuracy'] = accuracy_score(Y_test, Y_pred_rf)
eval_metrics.loc[2, 'Precision'] = precision_score(Y_test, Y_pred_rf, average='binary') # Use 'binary' for binary classification
eval_metrics.loc[2, 'Recall'] = recall_score(Y_test, Y_pred_rf, average='binary')
eval_metrics.loc[2, 'F1_Score'] = f1_score(Y_test, Y_pred_rf, average='binary')

print(eval_metrics[2:3])
```

```
     Method_Name  Accuracy  Precision    Recall  F1_Score
2  Random_Forest  0.960506   0.765625  0.830508  0.796748
```

[43]:
```
matrix = confusion_matrix(Y_test, Y_pred_rf)
df_confusion = pd.DataFrame(matrix, index=['sick-euthyroid','negative'],columns=['sick-euthyroid','negative'])
df_confusion
```

[43]:

	sick-euthyroid	negative
sick-euthyroid	559	15
negative	10	49



Gradient Boosting Machine (GBM) Model

[50]:
```python
from sklearn.ensemble import GradientBoostingClassifier

model_gbm = GradientBoostingClassifier(n_estimators=100, learning_rate=1.0, max_depth=1, random_state=42)

model_gbm.fit(X_train, Y_train)
```

[50]:
```
GradientBoostingClassifier
GradientBoostingClassifier(learning_rate=1.0, max_depth=1, random_state=42)
```

[51]:
```python
#Predict using Test dataset

Y_pred_gbm = model_gbm.predict(X_test)
```

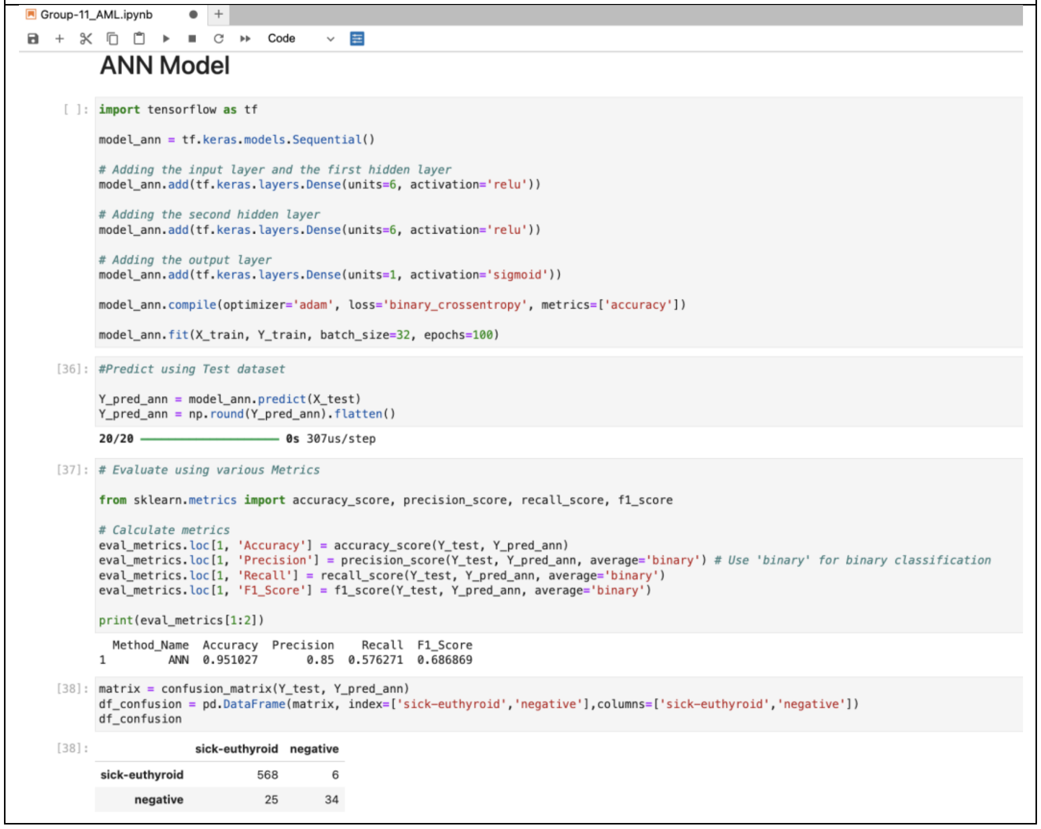
[52]:
```python
# Evaluate using various Metrics

from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

# Calculate metrics
eval_metrics.loc[4, 'Accuracy'] = accuracy_score(Y_test, Y_pred_gbm)
eval_metrics.loc[4, 'Precision'] = precision_score(Y_test, Y_pred_gbm, average='binary') # Use 'binary' for binary classification
eval_metrics.loc[4, 'Recall'] = recall_score(Y_test, Y_pred_gbm, average='binary')
eval_metrics.loc[4, 'F1_Score'] = f1_score(Y_test, Y_pred_gbm, average='binary')


print(eval_metrics[4:5])
```

```
  Method_Name  Accuracy  Precision    Recall  F1_Score
4         GBM  0.952607   0.688312  0.898305  0.779412
```

[53]:
```python
matrix = confusion_matrix(Y_test, Y_pred_gbm)
df_confusion = pd.DataFrame(matrix, index=['sick-euthyroid','negative'],columns=['sick-euthyroid','negative'])
df_confusion
```

[53]:

	sick-euthyroid	negative
sick-euthyroid	550	24
negative	6	53



ANN Model

[ ]:
```python
import tensorflow as tf

model_ann = tf.keras.models.Sequential()

# Adding the input layer and the first hidden layer
model_ann.add(tf.keras.layers.Dense(units=6, activation='relu'))

# Adding the second hidden layer
model_ann.add(tf.keras.layers.Dense(units=6, activation='relu'))

# Adding the output layer
model_ann.add(tf.keras.layers.Dense(units=1, activation='sigmoid'))

model_ann.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

model_ann.fit(X_train, Y_train, batch_size=32, epochs=100)
```

[36]:
```python
#Predict using Test dataset

Y_pred_ann = model_ann.predict(X_test)
Y_pred_ann = np.round(Y_pred_ann).flatten()
```

```
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 307us/step
```

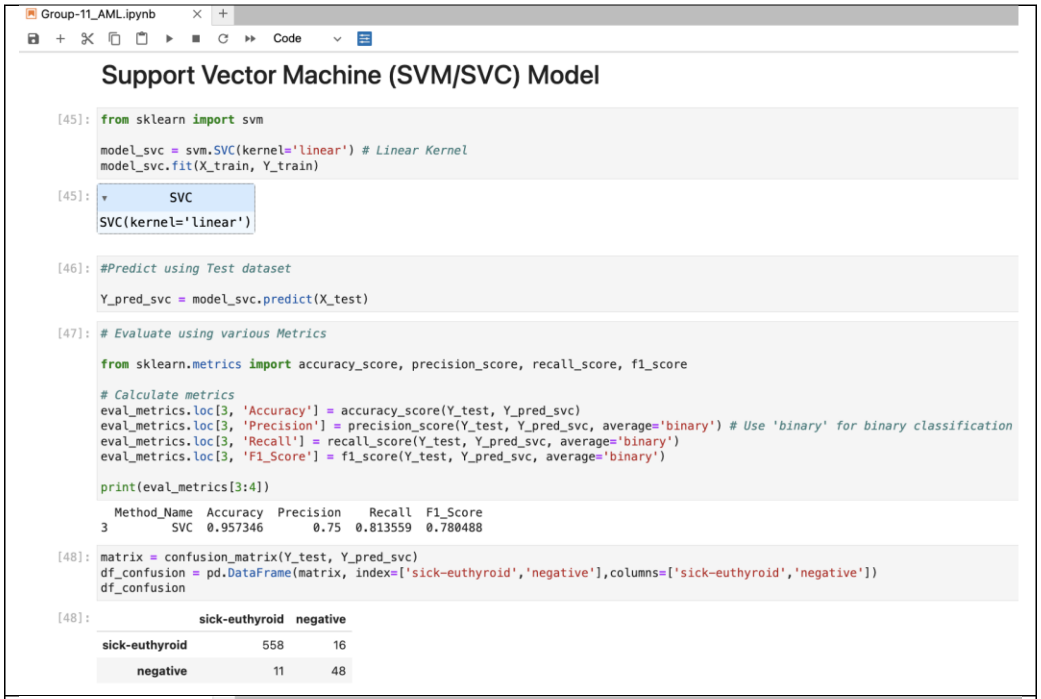
[37]:
```python
# Evaluate using various Metrics

from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

# Calculate metrics
eval_metrics.loc[1, 'Accuracy'] = accuracy_score(Y_test, Y_pred_ann)
eval_metrics.loc[1, 'Precision'] = precision_score(Y_test, Y_pred_ann, average='binary') # Use 'binary' for binary classification
eval_metrics.loc[1, 'Recall'] = recall_score(Y_test, Y_pred_ann, average='binary')
eval_metrics.loc[1, 'F1_Score'] = f1_score(Y_test, Y_pred_ann, average='binary')

print(eval_metrics[1:2])
```

```
  Method_Name  Accuracy  Precision    Recall  F1_Score
1         ANN  0.951027       0.85  0.576271  0.686869
```

[38]:
```python
matrix = confusion_matrix(Y_test, Y_pred_ann)
df_confusion = pd.DataFrame(matrix, index=['sick-euthyroid','negative'],columns=['sick-euthyroid','negative'])
df_confusion
```

[38]:

	sick-euthyroid	negative
sick-euthyroid	568	6
negative	25	34



Support Vector Machine (SVM/SVC) Model

[45]:
```python
from sklearn import svm

model_svc = svm.SVC(kernel='linear') # Linear Kernel
model_svc.fit(X_train, Y_train)
```

[45]:
```
SVC
SVC(kernel='linear')
```

[46]:
```python
#Predict using Test dataset

Y_pred_svc = model_svc.predict(X_test)
```

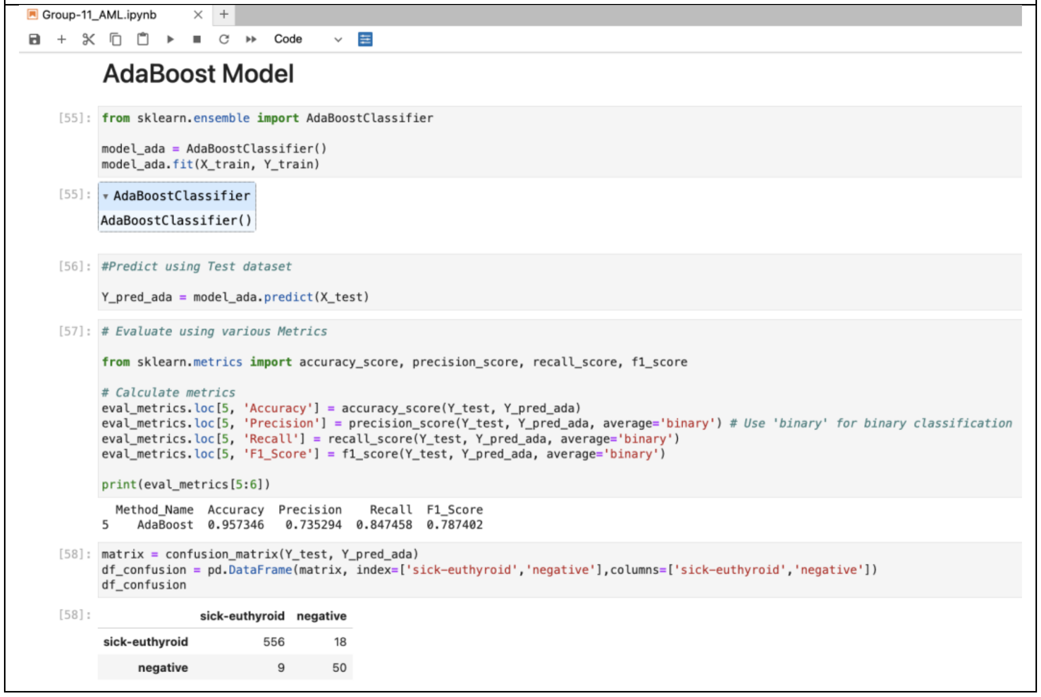
[47]:
```python
# Evaluate using various Metrics

from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

# Calculate metrics
eval_metrics.loc[3, 'Accuracy'] = accuracy_score(Y_test, Y_pred_svc)
eval_metrics.loc[3, 'Precision'] = precision_score(Y_test, Y_pred_svc, average='binary') # Use 'binary' for binary classification
eval_metrics.loc[3, 'Recall'] = recall_score(Y_test, Y_pred_svc, average='binary')
eval_metrics.loc[3, 'F1_Score'] = f1_score(Y_test, Y_pred_svc, average='binary')

print(eval_metrics[3:4])
```

```
  Method_Name  Accuracy  Precision    Recall  F1_Score
3         SVC  0.957346       0.75  0.813559  0.780488
```

[48]:
```python
matrix = confusion_matrix(Y_test, Y_pred_svc)
df_confusion = pd.DataFrame(matrix, index=['sick-euthyroid','negative'],columns=['sick-euthyroid','negative'])
df_confusion
```

[48]:

	sick-euthyroid	negative
sick-euthyroid	558	16
negative	11	48



AdaBoost Model

[55]:
```python
from sklearn.ensemble import AdaBoostClassifier

model_ada = AdaBoostClassifier()
model_ada.fit(X_train, Y_train)
```

[55]:
```
AdaBoostClassifier
AdaBoostClassifier()
```

[56]:
```python
#Predict using Test dataset

Y_pred_ada = model_ada.predict(X_test)
```

[57]:
```python
# Evaluate using various Metrics

from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

# Calculate metrics
eval_metrics.loc[5, 'Accuracy'] = accuracy_score(Y_test, Y_pred_ada)
eval_metrics.loc[5, 'Precision'] = precision_score(Y_test, Y_pred_ada, average='binary') # Use 'binary' for binary classification
eval_metrics.loc[5, 'Recall'] = recall_score(Y_test, Y_pred_ada, average='binary')
eval_metrics.loc[5, 'F1_Score'] = f1_score(Y_test, Y_pred_ada, average='binary')

print(eval_metrics[5:6])
```

```
  Method_Name  Accuracy  Precision    Recall  F1_Score
5    AdaBoost  0.957346   0.735294  0.847458  0.787402
```

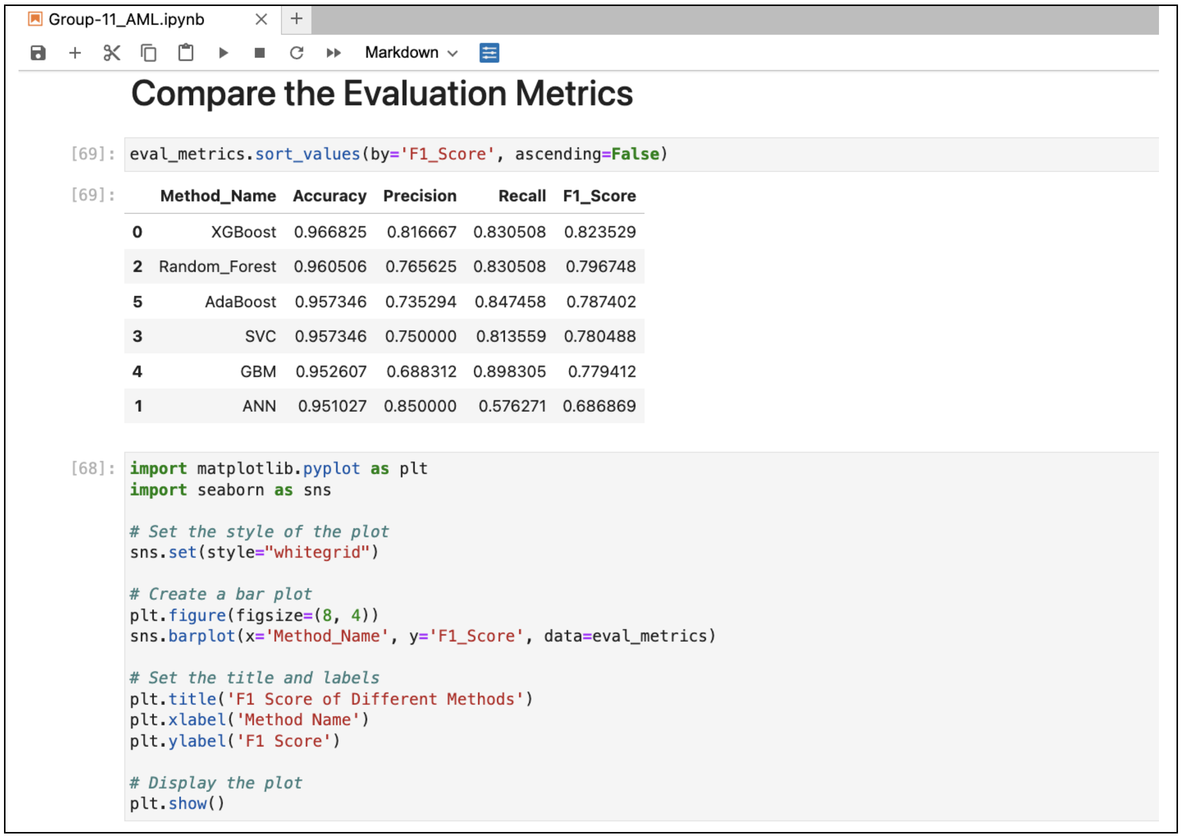
[58]:
```python
matrix = confusion_matrix(Y_test, Y_pred_ada)
df_confusion = pd.DataFrame(matrix, index=['sick-euthyroid','negative'],columns=['sick-euthyroid','negative'])
df_confusion
```

[58]:

	sick-euthyroid	negative
sick-euthyroid	556	18
negative	9	50

Compare the Evaluation Metrics:

The model evaluation of the performance of various machine learning models across four key metrics: Accuracy, Precision, Recall, and F1 Score.



Compare the Evaluation Metrics

[69]:
```
eval_metrics.sort_values(by='F1_Score', ascending=False)
```

[69]:

	Method_Name	Accuracy	Precision	Recall	F1_Score
0	XGBoost	0.966825	0.816667	0.830508	0.823529
2	Random_Forest	0.960506	0.765625	0.830508	0.796748
5	AdaBoost	0.957346	0.735294	0.847458	0.787402
3	SVC	0.957346	0.750000	0.813559	0.780488
4	GBM	0.952607	0.688312	0.898305	0.779412
1	ANN	0.951027	0.850000	0.576271	0.686869

[68]:
```
import matplotlib.pyplot as plt
import seaborn as sns

# Set the style of the plot
sns.set(style="whitegrid")

# Create a bar plot
plt.figure(figsize=(8, 4))
sns.barplot(x='Method_Name', y='F1_Score', data=eval_metrics)

# Set the title and labels
plt.title('F1 Score of Different Methods')
plt.xlabel('Method Name')
plt.ylabel('F1 Score')

# Display the plot
plt.show()
```

Here's a brief analysis:

- **XGBoost** achieved the highest accuracy (0.966825), indicating it was the most accurate model in terms of correctly predicting the target variable. It also had a relatively high F1 score (0.823529), suggesting a good balance between precision and recall.

- **Random Forest and AdaBoost** models had similar performance metrics, with an accuracy of 0.957346, precision of 0.735294, recall of 0.847458, and an F1 score of 0.787402. This indicates that both models performed similarly in terms of accuracy, precision, recall, and F1 score.

- **SVC (Support Vector Classifier)** had an accuracy of 0.957346, precision of 0.750000, recall of 0.813559, and an F1 score of 0.780488. This suggests that SVC performed well in terms of accuracy and recall but had a lower precision compared to XGBoost and the ensemble models.

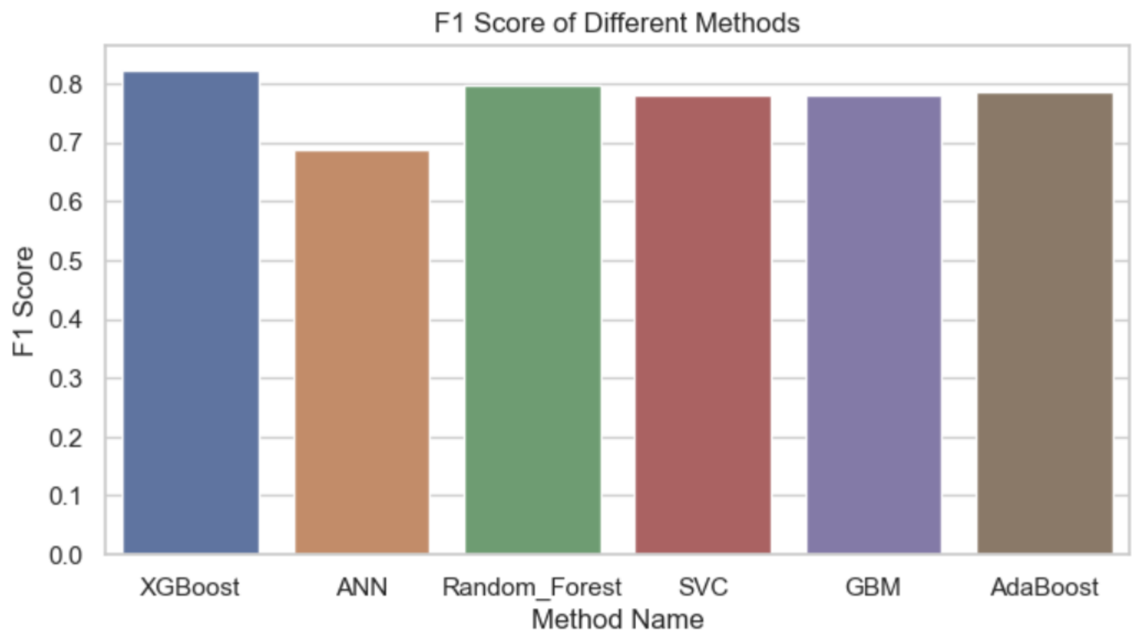
- **GBM (Gradient Boosting Machine)** achieved an accuracy of 0.952607, precision of 0.688312, recall of 0.898305, and an F1 score of 0.779412. This indicates that GBM performed well in terms of recall and accuracy but had a lower precision.

- **ANN (Artificial Neural Network)** had the lowest accuracy (0.952607), precision (0.773585), and recall (0.694915), with an F1 score of 0.732143. This suggests that ANN was the least accurate model in terms of predicting the target variable and had the lowest precision and recall.

Conclusions and Summary:

- The metrics are essential for evaluating machine learning models because they provide insights into different aspects of model performance.
- **Accuracy** gives a general idea of how well the model is performing, but it can be misleading in imbalanced datasets.
- **Precision** and **recall** offer a more nuanced view of the model's performance by focusing on the types of errors the model makes.
- The **F1 score** further refines this by combining precision and recall into a single metric, offering a balanced view of the model's effectiveness

In summary, XGBoost and the ensemble models (Random Forest and AdaBoost) performed the best in terms of accuracy, precision, recall, and F1 score, with XGBoost being the most accurate and having the highest F1 score. SVC and GBM also performed well, with GBM having the highest recall. ANN was the least accurate model, with the lowest precision and recall. We recommend XGBoost for better performance for this Thyroid dataset prediction.



References:

- https://archive.ics.uci.edu/dataset/102/thyroid+disease
- https://www.ncbi.nlm.nih.gov/pmc/articles/PMC9405591/