Thyroid disease data analysis and Prediction

Abdul Basith Mohammed (T00700187)

IFSC 7370 Data Science Technologies

Dr. Elizabeth Pierce

Spring 2023

05/08/2023

Contents

Introduction

The thyroid is one of the most crucial glands of the human body. It is responsible for releasing hormones essential for regulating metabolism, blood pressure, body temperature, and heart rate ("*Thyroid disorders*", 2022). Some of the important hormones released by the Thyroid gland are triiodothyronine (T3), and T4 or thyroxine.

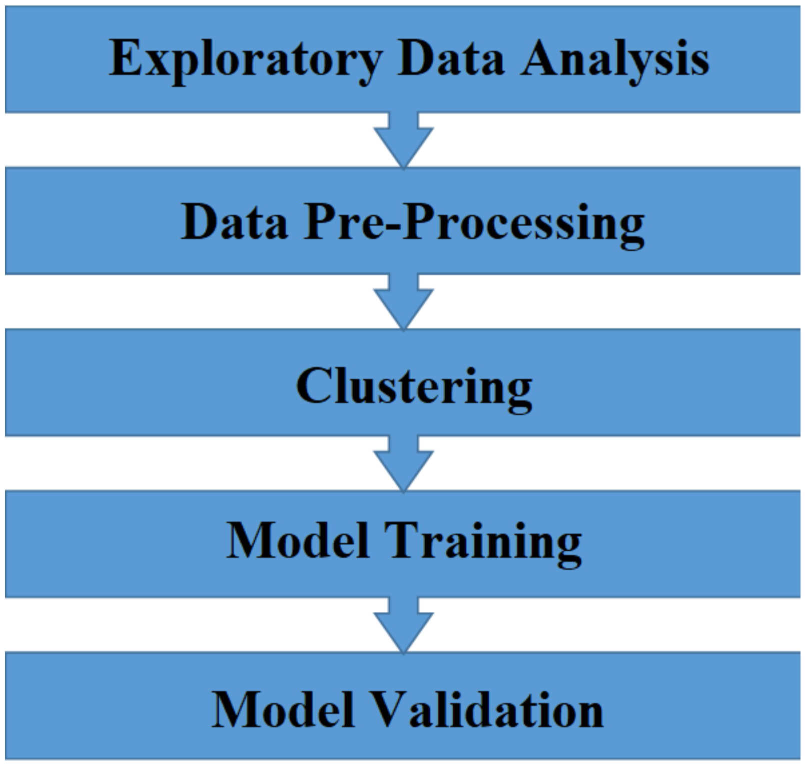
Unfortunately, like other organs/glands of the human body, it does get sick / has diseases that affect its normal functioning. The reasons for thyroid diseases can be plenty including lifestyle, family genetics, nutrient deficiencies, or pregnancies. Primarily, there are two types of thyroid disorders/diseases.

1. Hyperthyroidism – Excess quantity of thyroid hormones produced.

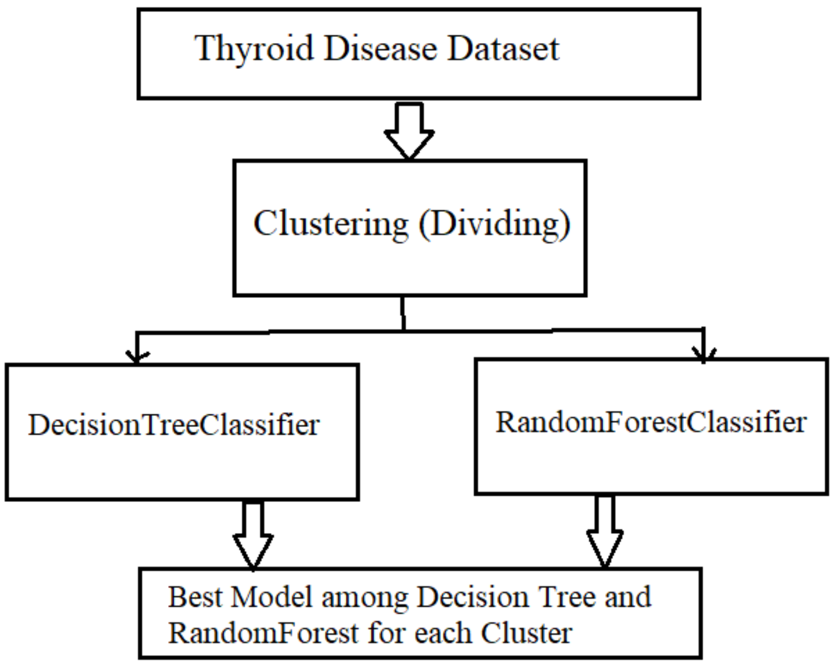
2. Hypothyroidism – Inadequate thyroid hormone production.

Problem Statement:

In this project/report, I am going to use Machine Learning models for predicting the type of thyroid disease. This project is important and interesting to me because I too have hypothyroidism and am curious to know more about it using analysis. This project will be interesting because I am using multiple machine learning algorithms such as K-means clustering and Random Forest for prediction. The flow of the project is illustrated by the below diagrams.



Process Flow Diagram



<u>Process Flow Diagram</u>

Data

The data for this project is taken from the University of California Irvine Machine Learning Repository (https://archive.ics.uci.edu/ml/machine-learning-databases/thyroid-disease/). There are many data files present in the source location. I am using the file named ‘allhypo.data’. After downloading the file, I changed its extension to .csv format. Added the headers using the ‘allbp.names’ text file. The dataset contains 2800 records and 30 columns. The target column is Class, it is a qualitative column having four unique values as mentioned below:

1. Negative
2. compensated hypothyroid
3. primary hypothyroid
4. secondary hypothyroid

This is a **classification** problem because we are predicting the qualitative column.

The columns/features of this dataset:

'age', 'sex', 'on_thyroxine', 'query_on_thyroxine', 'on_antithyroid_medication', 'sick', 'pregnant', 'thyroid_surgery', 'I131_treatment', 'query_hypothyroid', 'query_hyperthyroid', 'lithium','goitre', 'tumor', 'hypopituitary', 'psych', 'TSH_measured', 'TSH','T3_measured', 'T3', 'TT4_measured', 'TT4', 'T4U_measured', 'T4U','FTI_measured', 'FTI', 'TBG_measured', 'TBG', 'referral_source'.

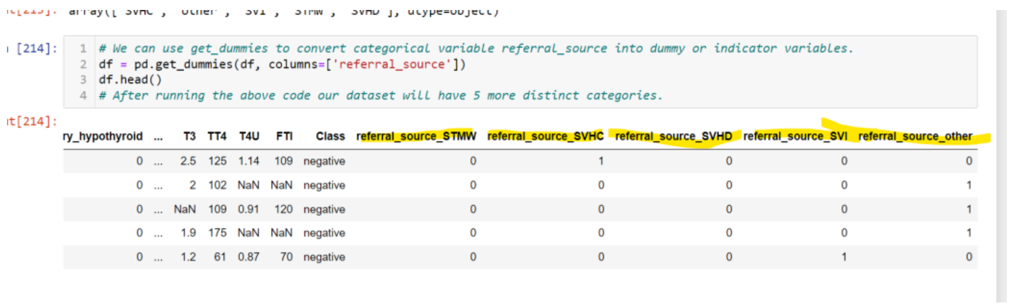
- Thyroxine – (Called T4), TSH, and T3 are thyroid hormones.

- TBG is a protein produced by the liver helpful in the distribution of T4.
- Hypopituitary – Condition of the pituitary gland.
- Hypothyroid and Hyperthyroid are explained in the introduction.
- FTI means Free thyroxine index which is used for calculating the amount of Free Thyroxine in the bloodstream.

Exploratory Data Analysis and Data Pre-Processing

Data cleaning is very important for model training and accuracy. I have done EDA to have a thorough understanding of the dataset and cleaned the null values, and duplicate values, and dropped unnecessary columns.

After carefully looking at the data it can be found that "?" have replaced the null values. So, I have replaced the "?" marks with the np.nan. There are a few columns by the name 'TBG', dummy values denoted with 'measured', I have dropped them. We have many categorical columns that need to be converted to numerical ones before we apply any imputation techniques. I have utilized the map function of Python for mapping F and M to 0 and 1 respectively. Mapped the columns having f and m to 0 and 1 respectively. Used get_dumsmies() to convert categorical variable referral_source into dummy or indicator variables.

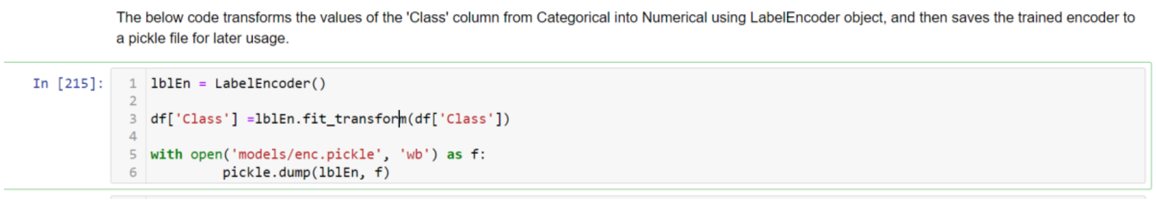


```
t[213]: array(['SVHC', 'other', 'SVI', 'STMW', 'SVHD'], dtype=object)
```

```
 [214]:
# We can use get_dummies to convert categorical variable referral_source into dummy or indicator variables.
df = pd.get_dummies(df, columns=['referral_source'])
df.head()
# After running the above code our dataset will have 5 more distinct categories.
```

```
t[214]:
ry_hypothyroid  ...   T3  TT4   T4U  FTI     Class  referral_source_STMW  referral_source_SVHC  referral_source_SVHD  referral_source_SVI  referral_source_other
             0  ...  2.5  125  1.14  109  negative                     0                     1                     0                    0                      0
             0  ...    2  102   NaN  NaN  negative                     0                     0                     0                    0                      1
             0  ...  NaN  109  0.91  120  negative                     0                     0                     0                    0                      1
             0  ...  1.9  175   NaN  NaN  negative                     0                     0                     0                    0                      1
             0  ...  1.2   61  0.87   70  negative                     0                     0                     0                    1                      0
```

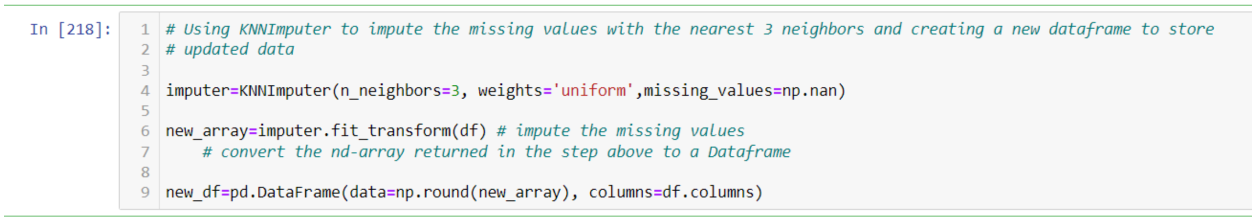
To convert the Class column from Categorical to Numerical, used the LabelEncoder object, and saved the trained encoder to a pickle file for later usage.



The below code transforms the values of the 'Class' column from Categorical into Numerical using LabelEncoder object, and then saves the trained encoder to a pickle file for later usage.

```
In [215]:
lblEn = LabelEncoder()

df['Class'] =lblEn.fit_transform(df['Class'])

with open('models/enc.pickle', 'wb') as f:
        pickle.dump(lblEn, f)
```

After converting all the categorical data into numerical data, used KNNImputer to impute the missing values.

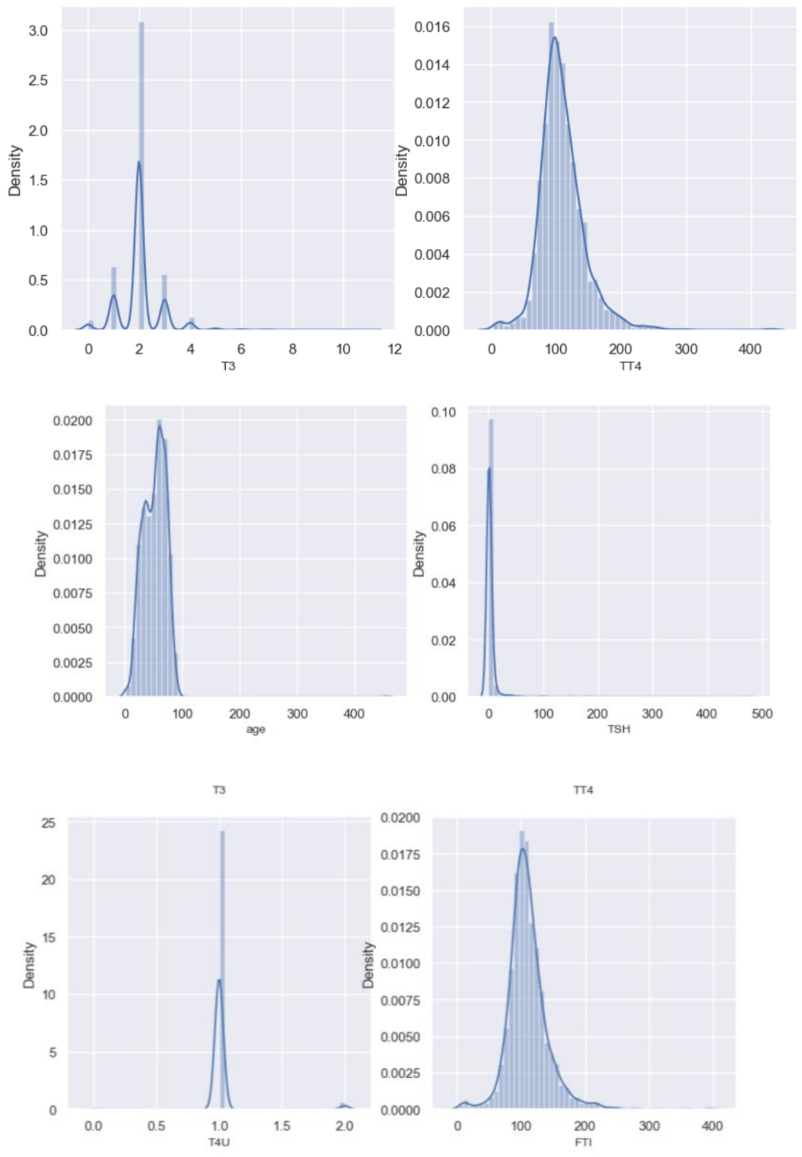


```
In [218]: # Using KNNImputer to impute the missing values with the nearest 3 neighbors and creating a new dataframe to store
          # updated data

          imputer=KNNImputer(n_neighbors=3, weights='uniform',missing_values=np.nan)

          new_array=imputer.fit_transform(df) # impute the missing values
              # convert the nd-array returned in the step above to a Dataframe

          new_df=pd.DataFrame(data=np.round(new_array), columns=df.columns)
```

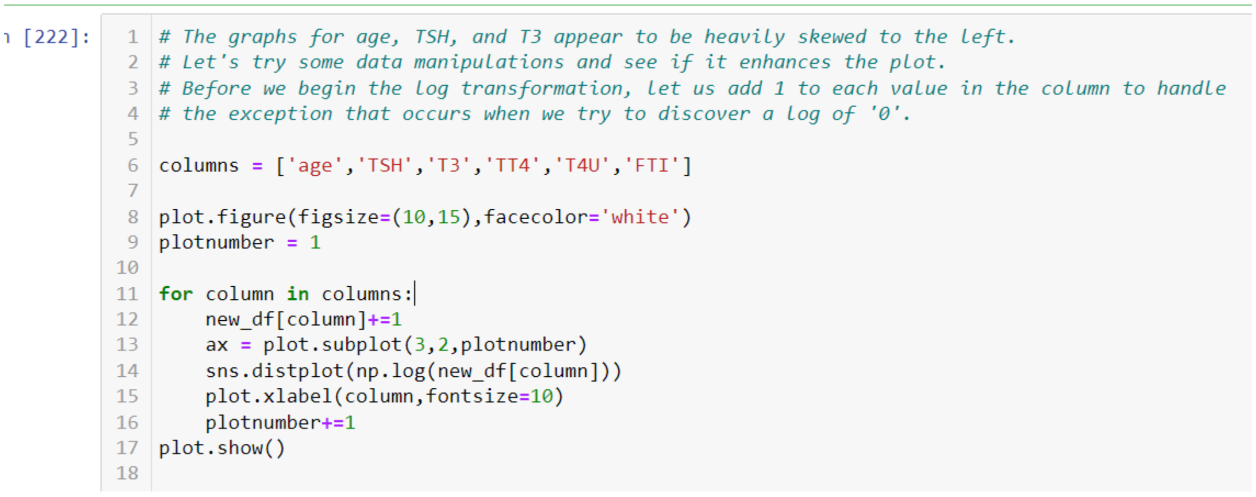
Now the data is cleaned, it is ready for the visualizations.

Visualizations:

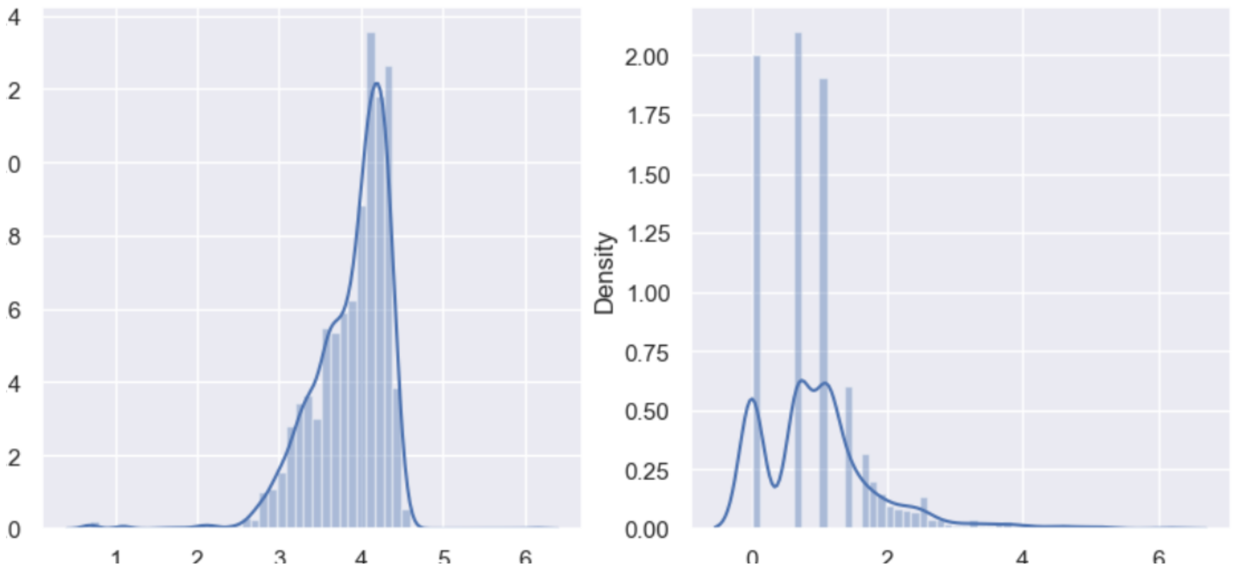
Used the seaborn distplot for plotting the overall distribution of the columns ['age', 'TSH' , 'T3', 'TT4' , 'T4U' ,'FTI']. This graph looks skewed to either left or right.



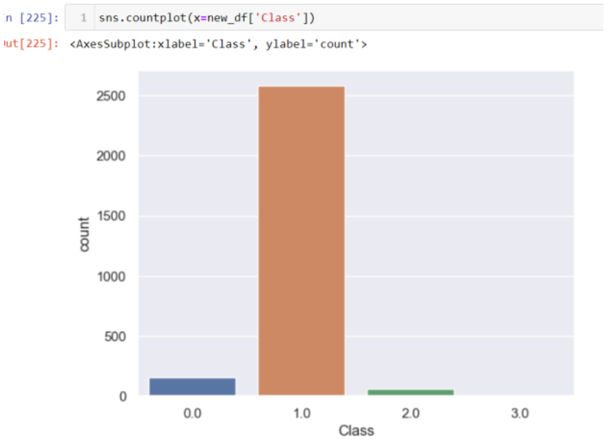
Applying the log transformation to correct it.



```
n [222]:
# The graphs for age, TSH, and T3 appear to be heavily skewed to the left.
# Let's try some data manipulations and see if it enhances the plot.
# Before we begin the log transformation, let us add 1 to each value in the column to handle
# the exception that occurs when we try to discover a log of '0'.

columns = ['age','TSH','T3','TT4','T4U','FTI']

plot.figure(figsize=(10,15),facecolor='white')
plotnumber = 1

for column in columns:
    new_df[column]+=1
    ax = plot.subplot(3,2,plotnumber)
    sns.distplot(np.log(new_df[column]))
    plot.xlabel(column,fontsize=10)
    plotnumber+=1
plot.show()

```

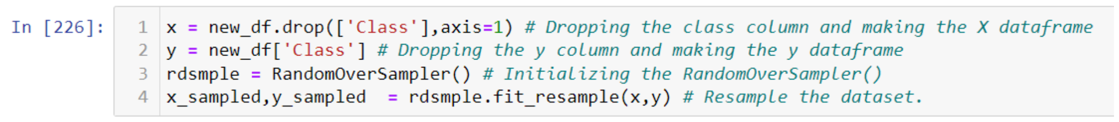


The dataset was heavily imbalanced i.e., the 'Class' column had nearly 2500 occurrences of the same value (negative). So, did some oversampling to balance the data.

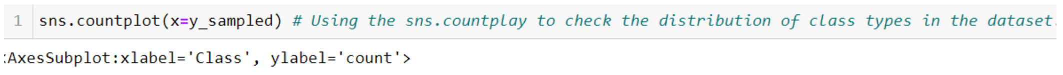


```
n [225]: sns.countplot(x=new_df['Class'])
ut[225]: <AxesSubplot:xlabel='Class', ylabel='count'>
```

The below piece of code does the oversampling and balances the data.

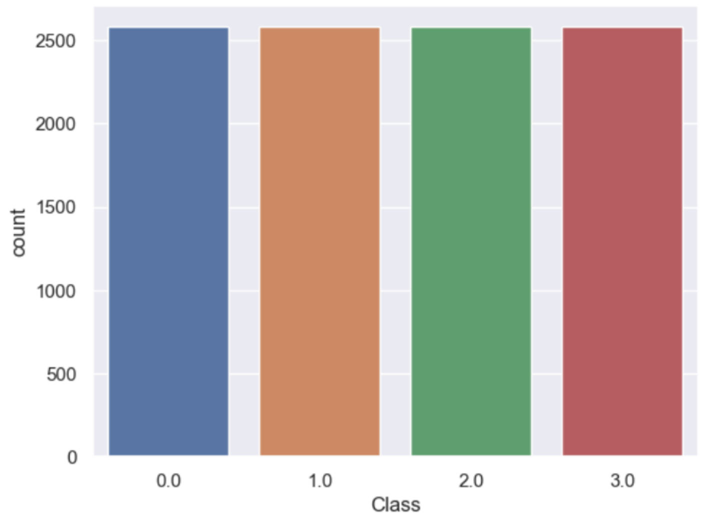


```
In [226]: x = new_df.drop(['Class'],axis=1) # Dropping the class column and making the X dataframe
          y = new_df['Class'] # Dropping the y column and making the y dataframe
          rdsmple = RandomOverSampler() # Initializing the RandomOverSampler()
          x_sampled,y_sampled  = rdsmple.fit_resample(x,y) # Resample the dataset.
```



```
sns.countplot(x=y_sampled) # Using the sns.countplay to check the distribution of class types in the dataset.
```

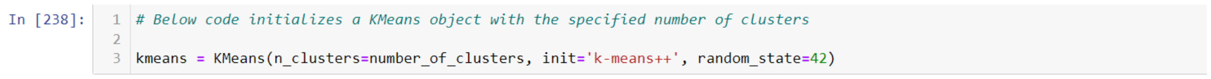
```
:AxesSubplot:xlabel='Class', ylabel='count'>
```



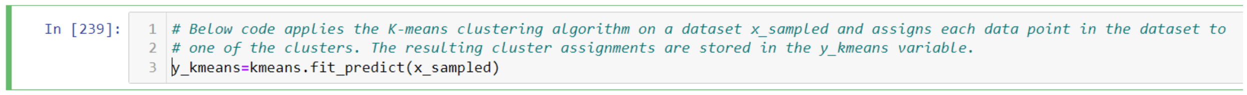
Now the data quality is improved and it can be clustered into different groups.

Clustering

In this phase of my project, I am grouping our data into various clusters using the K-means clustering algorithm. I am using the elbow plot and kneed libraries' knee attribute to get the optimum number of clusters (K) to split the data. After getting the K value, implemented the K Means algorithm onto the pre-processed dataset.



```
In [238]: # Below code initializes a KMeans object with the specified number of clusters

          kmeans = KMeans(n_clusters=number_of_clusters, init='k-means++', random_state=42)
```



```
In [239]: # Below code applies the K-means clustering algorithm on a dataset x_sampled and assigns each data point in the dataset to
          # one of the clusters. The resulting cluster assignments are stored in the y_kmeans variable.
          y_kmeans=kmeans.fit_predict(x_sampled)
```

I am saving all the models that are generated during this training locally on the file system for future predictions. I have created a custom function called model_saving() for it.

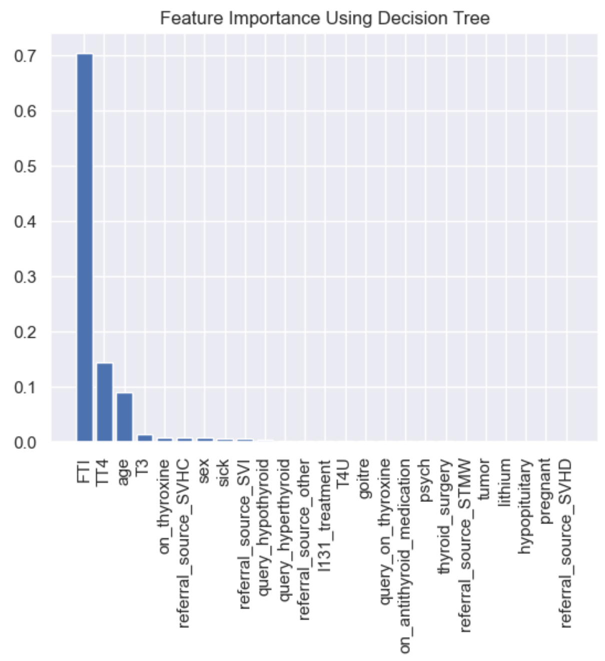
After creating the clusters, I am adding a label column to the data for the identification of the records concerning the cluster numbers.

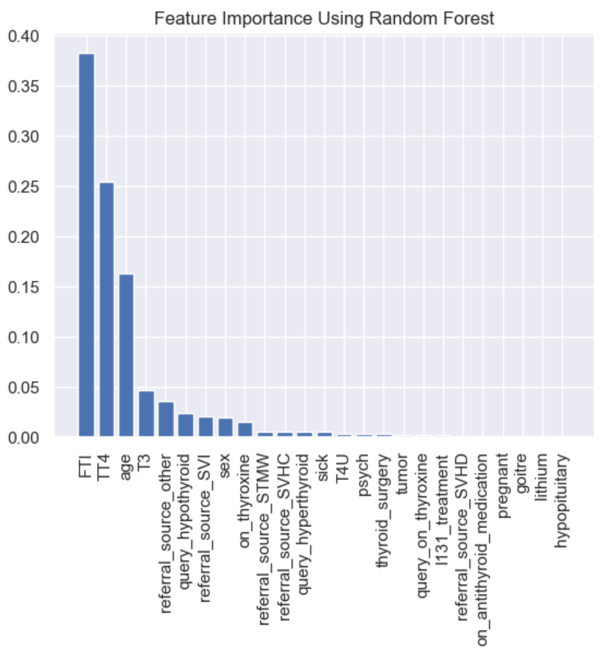
Now, that we have divided our dataset into clusters we can proceed with the model selection/training.

Feature Selection

In this phase, I am selecting only the features that are most important for the prediction of thyroid disease. I am using the Decision Tree and Random Forest classifiers from the Scikit learn library to first shortlist the important features which will be further used in the Model Training/Selection phase.

Below are the graphs of the important features after applying Decision Tree and Random Forest.





After looking at the above figures, I have shortlisted only the below-mentioned 9 features/predictors for better accuracy and less complexity.

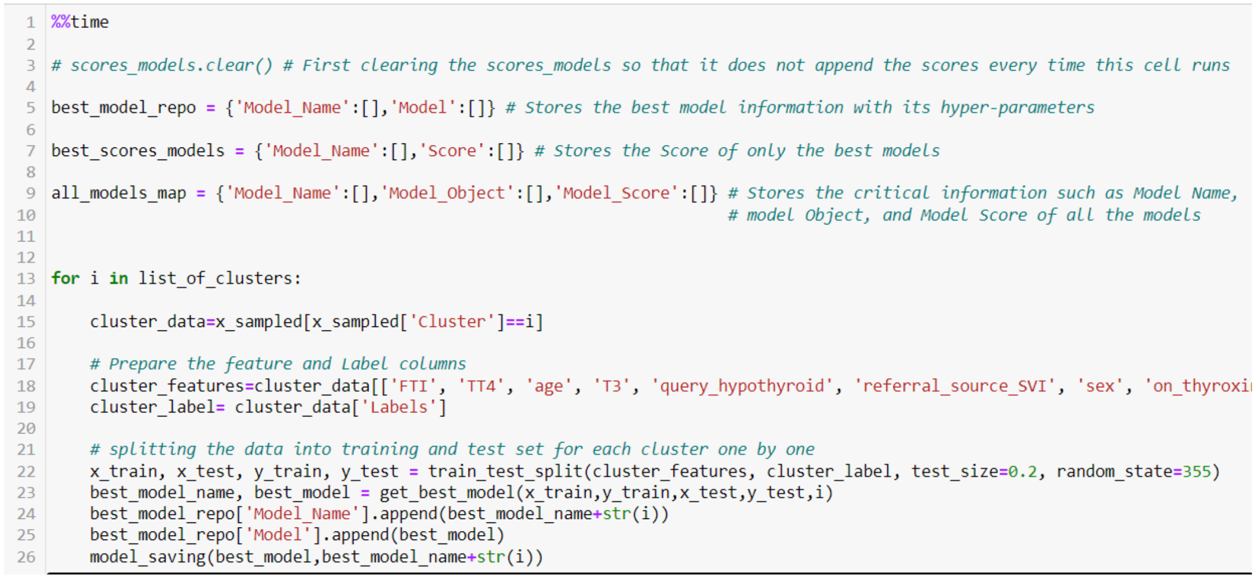
['FTI', 'TT4', 'age', 'T3', 'query_hypothyroid', 'referral_source_SVI', 'sex', 'on_thyroxine', 'referral_source_STMW'].

From those features the top 3 predictors for Thyroid disease are FTI, TT4, and age.

Model Training/Selection

This is the most crucial and exciting phase of the project. RandomForestClassifier and DecisionTreeClassifier are the two models used for the training purpose. I am doing hyperparameter tuning using GridSearchCV() to get the best parameters for increased accuracy and stability of the models. I have designed the code of the model training using modularity i.e., in small blocks. I have created separate functions calling each model and separated the model training. **Detailed code and its explanation is provided in the Python notebook** that I have created for the Model Training named **thyroid_ml_project.ipynb.**

I will give an overview of each function and the data flow in brief here.



```
%%time

# scores_models.clear() # First clearing the scores_models so that it does not append the scores every time this cell runs

best_model_repo = {'Model_Name':[],'Model':[]} # Stores the best model information with its hyper-parameters

best_scores_models = {'Model_Name':[],'Score':[]} # Stores the Score of only the best models

all_models_map = {'Model_Name':[],'Model_Object':[],'Model_Score':[]} # Stores the critical information such as Model Name,
                                                                      # model Object, and Model Score of all the models


for i in list_of_clusters:

    cluster_data=x_sampled[x_sampled['Cluster']==i]

    # Prepare the feature and Label columns
    cluster_features=cluster_data[['FTI', 'TT4', 'age', 'T3', 'query_hypothyroid', 'referral_source_SVI', 'sex', 'on_thyroxi
    cluster_label= cluster_data['Labels']

    # splitting the data into training and test set for each cluster one by one
    x_train, x_test, y_train, y_test = train_test_split(cluster_features, cluster_label, test_size=0.2, random_state=355)
    best_model_name, best_model = get_best_model(x_train,y_train,x_test,y_test,i)
    best_model_repo['Model_Name'].append(best_model_name+str(i))
    best_model_repo['Model'].append(best_model)
    model_saving(best_model,best_model_name+str(i))
```

Code block1:

The above code uses the for loop to get each cluster's data sequentially. It splits the cluster_data into two parts in the ratio of 80:20 (80 % for training and 20 % for testing). It passes the clustered data to the function **get_best_model()** and gets the model (either DecisionTree or Random Forest) which is best for the respective cluster's data.

Method 2: **get_best_model()**

The **get_best_model()** function has two functions **cc**() and **get_best_model_for_random_forest**() which are implementing the Hypertuned DecisionTree and Random Forest, models. The data is **given** to these both functions and the best DecisionTree and Random Forest Model is **returned** for each cluster. The accuracy of both the models is measured using different metrics and the best model is returned to the main code.

Method 3: **get_best_model_for_DecisionTree**()

The **get_best_model_for_DecisionTree** () takes the training data. Uses GridSearchCV() to find the best parameters to implement DecisionTreeClassifier(), fits the classifier, and returns the DecisionTree model.

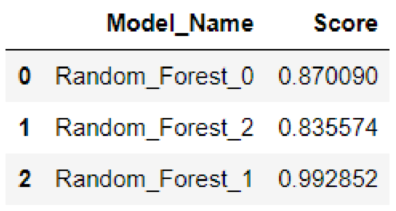
Method 4: **get_best_model_for_random_forest**()

Similar to the get the best model for knn, **get_best_model_for_random_forest**() takes the training data. Uses GridSearchCV() to find the best parameters to implement RandomForestClassifier(), fits the classifier, and returns the RandomForest model.

Conclusion / Results

The below tables and output gives us the accuracy, recall, and F-1 score of both the decision tree and random forest models for each cluster. By looking at these tables we can conclude that the RandomForestClassifiers are best suited for this dataset. The accuracy of the random forest for clusters 1 and 2 is nearly 95 % which is impressive. The accuracy of 98% for cluster 0 suggests that this model might be overfitted because of the data.

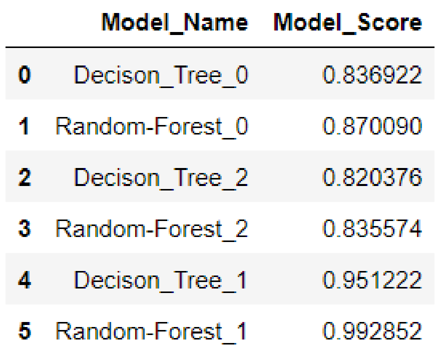
Scores of best models in each cluster:



	Model_Name	Score
0	Random_Forest_0	0.870090
1	Random_Forest_2	0.835574
2	Random_Forest_1	0.992852

The below table illustrates the scores of all the models (RandomForest and Decision Tree). The decision tree model scores are also impressive here.

Scores of all the Models on each cluster:



	Model_Name	Model_Score
0	Decison_Tree_0	0.836922
1	Random-Forest_0	0.870090
2	Decison_Tree_2	0.820376
3	Random-Forest_2	0.835574
4	Decison_Tree_1	0.951222
5	Random-Forest_1	0.992852

Classification_report for Cluster 0 DecisionTreeClassifier()

I am passing each cluster data to the decision tree and calculating the precision, recall, and f1-score using classification_report.

```
              precision    recall  f1-score   support

         0.0       0.62      0.35      0.44        81
         1.0       0.79      0.93      0.86       255
         2.0       1.00      0.44      0.61        16

    accuracy                           0.78       352
   macro avg       0.81      0.57      0.64       352
weighted avg       0.76      0.78      0.75       352
```

Classification_report for Cluster 1 || DecisionTreeClassifier()

I am passing each cluster data to the decision tree and calculating the precision, recall, and f1-score using classification_report

```
              precision    recall  f1-score   support

         0.0       0.93      1.00      0.97        14
         1.0       1.00      0.40      0.57         5
         2.0       1.00      1.00      1.00       449
         3.0       1.00      1.00      1.00       532

    accuracy                           1.00      1000
   macro avg       0.98      0.85      0.88      1000
weighted avg       1.00      1.00      1.00      1000
```

Classification_report for Cluster 2 || DecisionTreeClassifier()

```
precision    recall  f1-score   support

         0.0       0.78      0.62      0.69       414
         1.0       0.55      0.72      0.62       268
         2.0       0.97      1.00      0.98        31

    accuracy                           0.67       713
   macro avg       0.76      0.78      0.76       713
weighted avg       0.70      0.67      0.68       713
```

Cluster-wise, the decision tree model performed pretty well on only Cluster # 1. It was not that good on the other two clusters.

Classification_report for Cluster 0 || RandomForestClassifier()

```
    precision    recall  f1-score   support

         0.0       0.94      0.19      0.31        81
         1.0       0.77      1.00      0.87       255
         2.0       1.00      0.44      0.61        16

    accuracy                           0.78       352
   macro avg       0.90      0.54      0.60       352
weighted avg       0.82      0.78      0.73       352

Score:  0.7840909090909091
```

Classification_report for Cluster 1 || RandomForestClassifier()

```
precision    recall  f1-score   support

         0.0       1.00      0.43      0.60        14
         1.0       0.00      0.00      0.00         5
         2.0       0.98      0.98      0.98       449
         3.0       0.97      1.00      0.99       532

    accuracy                           0.98      1000
   macro avg       0.74      0.60      0.64      1000
weighted avg       0.97      0.98      0.97      1000

Score:  0.977
```

Classification_report for Cluster 2 || RandomForestClassifier()

```
precision      recall         f1-score   support

         0.0       0.66      0.98      0.79       414
         1.0       0.88      0.22      0.35       268
         2.0       0.97      1.00      0.98        31

    accuracy                           0.69       713
   macro avg       0.84      0.73      0.71       713
weighted avg       0.76      0.69      0.63       713

Score:  0.6942496493688639
```

The accuracy and recall of the clusters 0 and 1 are better than 2. But, the F1- score is higher for the cluster 2 RandomForestClassifier()

Execution Time:

DecisionTreeClassifier() models were quicker than RandomForest. DecisionTree took nearly 3 secs for running the cluster whereas RandomForest took almost 20 secs.

Future Work / Critique

In this project, I have created the trained models based on the training data from the CSV files. I have created the below model files and saved them on the file system.

enc.pickle file -- This file holds the encoding information of the class column

kmeans_clustering.sav – This file contains the ML Kmeans model created for the clustering

Decision_Tree(<Cluster_Number>) – The File with this type of name contains the decision tree model

RandomForest((<Cluster_Number>) – The file with this type of name contains the RandomForest model.

By using the above files, I can build pipelines that can take the raw files as the input, process the data by dividing it into clusters, apply hyper-tuned machine learning models on the clusters, and finally predict the outcomes.

The above system/pipeline can then be created into a Flask or any other kind of application or API deployed locally or cloud. This application can be consumed by front-end clients such as web apps, HTTP clients, etc. They can also be used as a part of a much bigger system such as a multi-disease diagnostic/detection system.

Critique:

This was one of the most interesting and learning journeys of my life. Machine learning, Data Science, and other important concepts learned and implemented by me in this project and the entire course will help me become a better professional. I strongly believe that nothing is perfect and there is always an improvement to it. The same applies to my project. This is a new system implemented by me, I am sure that it can be improved in its later versions. More testing I do on this project, the more I may be able to enhance it. Below are some learnings or shortcomings which I believe exist in my dataset or model:

1. The dataset has many columns that do not contribute to the prediction, I have excluded them.
2. The dataset had more qualitative variables, I am new to them and always have trouble with them. I have done good work on handling them but surely know that they can be handled much more efficiently
3. I have implemented KNeighborsClassifier() on this dataset, but it was taking a lot of time to train it when compared to a random forest and decision tree classifiers
4. I could have used some additional models such as SVM, logistic regression, or else Naïve Bayes.

Certification

My Brief Background:

I have a bachelor's degree in Electronics and Communications Engineering. But, due to my interest in Information Technology, I chose to work in the Information Technology industry. Before coming to Arkansas for my Masters, I was a Microsoft-certified Azure Administrator. Even though, I was in IT for so many years, my knowledge of Data Science, Machine Learning, and Python was little. In my first semester (Fall 2022), I realized that without learning Python, I cannot do anything in this field. I quickly learned basic Python to use it in some very basic tasks such as visualization, or at least understanding what is going on in the data field or projects.

Now, coming to this semester (Spring 2023), I am very glad that I took this course. The quick mini-courses provided by Dr. Pierce from CognitiveClass.ai were helpful. Mr. Bruce Bauer's Python classes strengthened my Python skills. I thought that doing a Data Science project for the final assignment is not a cup of tea for me and I might end up wasting time, so I took a certification course from Udemy called **Complete 2022 Data Science & Machine Learning Bootcamp** (https://www.udemy.com/course/python-data-science-machine-learning-bootcamp/). I finished this course last week i.e., 1st week of May 2023, and received a course completion certificate.

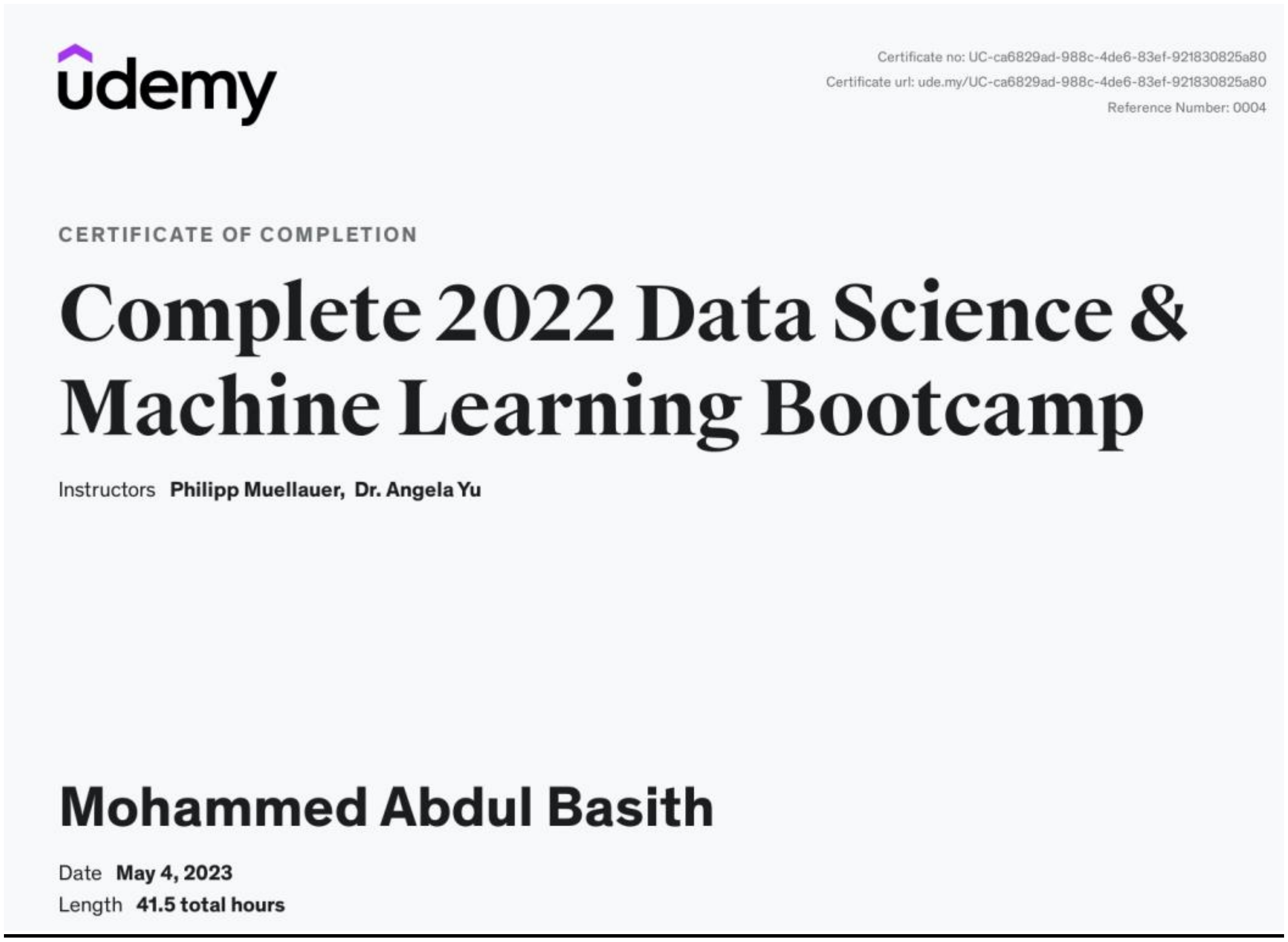
Complete 2022 Data Science & Machine Learning Bootcamp Certificate:

I have attached the certificate as a pdf, the file name is ***DataScience_Certificate_Abdul_Basith.pdf***. I learned many topics related to Data Science specifically Machine learning during this course from **Philipp Muellauer and Dr. Angela Yu**.

Though this course was for 41.5 hours. I have skipped some Python classes because I was taking classes from Mr. Bruce. Overall, I have spent almost 38 hours on this course. These are some of the topics that I have learned from this course.

1. Data Cleaning and pre-processing using Python
2. Regression
    a. Linear Regression
    b. Regularization (Ridge and Lasso)
    c. Random Forest Regression
    d. Support Vector Regressor
3. Classification
    a. Multivariable Regression
    b. Logistic Regression
    c. Decision Tree
    d. KNN
    e. Support Vector Machines (SVM)
    f. XG-Boost
    g. ADA boost
    h. Random Forest Classifier
4. Clustering,
    a. K Means
    b. Hierarchical Clustering
    c. Density-Based Spatial Clustering of Applications with Noise (DBSCAN)

I did work on numerous Python packages namely Scikit Learn, Pandas, NumPy, SciPy, TensorFlow, and Keras.


Certificate no: UC-ca6829ad-988c-4de6-83ef-921830825a80
Certificate url: ude.my/UC-ca6829ad-988c-4de6-83ef-921830825a80
Reference Number: 0004

CERTIFICATE OF COMPLETION

Complete 2022 Data Science & Machine Learning Bootcamp

Instructors **Philipp Muellauer, Dr. Angela Yu**

Mohammed Abdul Basith

Date **May 4, 2023**
Length **41.5 total hours**

Time spent on the Data Science project and certification

As explained in the previous section about my background. The transition from an Administrative background to Data Analytics and Data Science is not an easy task. But I have put some remarkable efforts in the right direction under the guidance of Dr. Pierce. Below is a table describing my time spent on the data science project/certification.

Type of Work	Activity	Approximate Time Spent (Hrs.)
Certification	Data Science & Machine Learning Bootcamp Certificate	38
Project	Problem Formulation and dataset search	3
	Understanding the dataset	2
	Data Cleaning (Learning Imputation and new techniques)	5
	Learning to implement clustering to improve performance and use multiple models on different clusters	4
	Coding involved in implementing the clusters and dealing with multiple classifiers	3
	Learning label encoding	1.5
	Learning about the Seaborn, Matplotlib visualizations	1
	Solving coding errors	2
	Commenting the code	0.5
	Learning and implementing metrics of RandomForest and DecisionTree Classifiers	1
	Code optimizations	2
	Preparing the report and other documentation	3
	Total	66

Time for Certification: 38 Hrs.

Time for Project: 28 Hrs.

References:

Thyroid disorders. (2022, October 28). Johns Hopkins Medicine, based in Baltimore, Maryland. https://www.hopkinsmedicine.org/health/conditions-and-diseases/disorders-of-the-thyroid#:~:text=The%20thyroid%20is%20a%20butterfly,the%20body%20to%20other%20hormones

Appendices

Folder structure and files information:

I have placed all these files on the GitHub at https://github.com/Basith5075/Data_Science_Submission.git

If you clone or download the files from the above github repository you will find contents like this.

Submission (Root Folder):

- **Certification**
  - Practice (Folder Containing all the files used for practice)
  - DataScience_Certificate_Abdul_Basith.pdf (Udemy Data Science Bootcamp Certificate)
- **Thyroid_Project**
  - Thyroid Detection Data Analysis Report.pdf (Project Report document)
  - thyroid_ml_project.ipynb (This is the notebook containing my entire code and extensive comments for better understanding). I have created this notebook using **Jupyter notebook** and tested it.
  - raw_data (Folder containing the raw data file **'raw_data.csv'** used for model training)
  - models (It has the saved model objects from the past training, if you run the notebook thyroid_ml_project.ipynb, all the content of this folder will be deleted and new models will be created)
- 3MT_Abdul_Basith.mp4 – 3 minutes thesis video
- 3MT_Slides_Abdul_Basith.pptx – 3 MT Slides