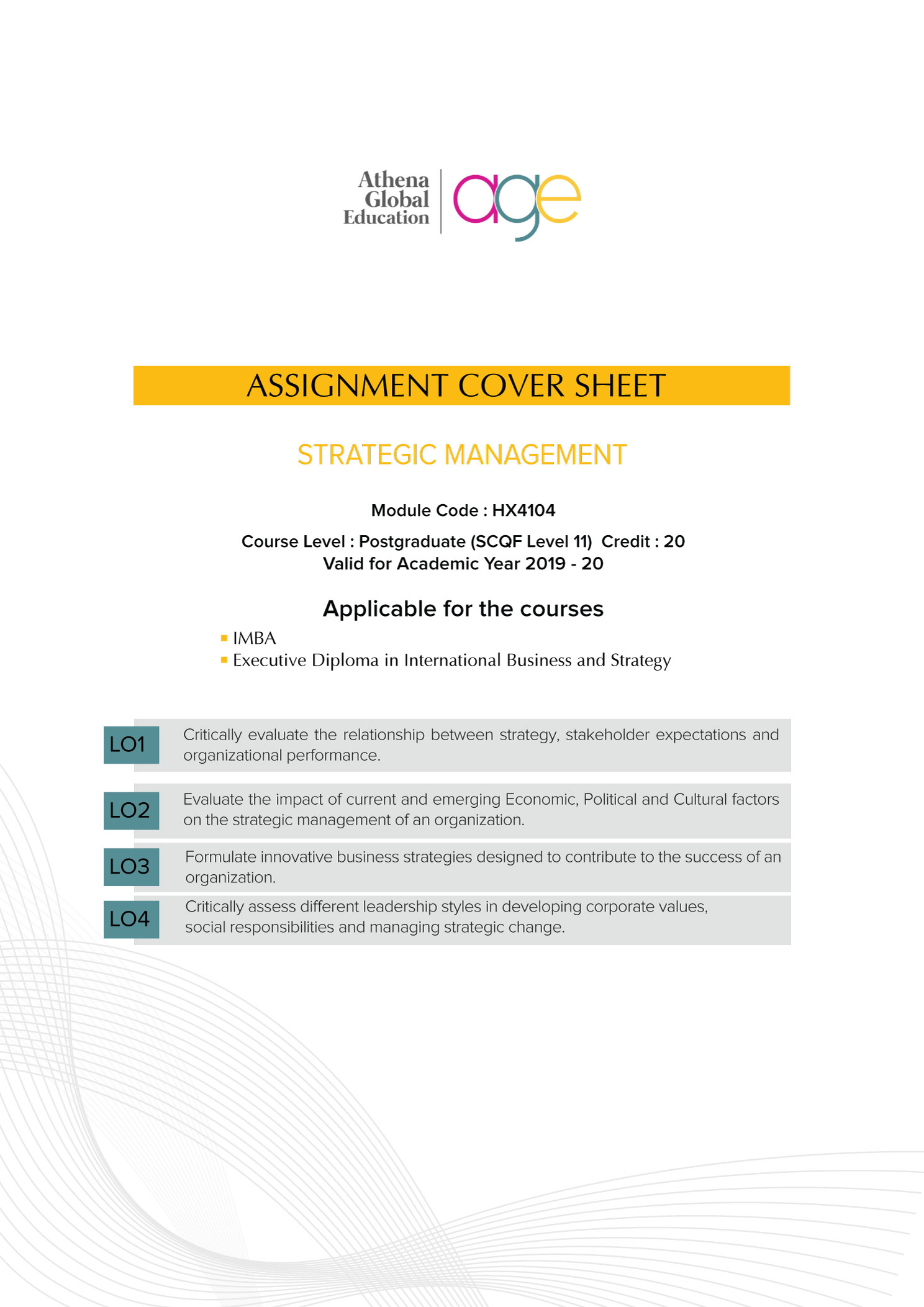
A blue and white logo

Description automatically generated



|  |  |
| --- | --- |
|  | |
| **COURSE/UNIT INFORMATION** | |
| Course | *PG Diploma in AI and ML – UCAM* |
| Course Level | *Postgraduate* |
| Module Name | Dissertation |
| Awarding Body | CIQ/UCAM |
| Module Code |  |
| Faculty |  |
| **ASSIGNMENT INFORMATION** | |
| Full/ Part Assignment | *Full* |
| Assignment brief IV by | *Dr. Vivek Mohan* |
| Assessor |  |
| Assignment due date |  |
| Turnitin Class ID | *35800448* |
| Turnitin Enrolment Key | *AIT115* |
| **TO BE FILLED BY THE STUDENT** | |
| Student Name | *Alaa Ali Abd-Alrazaq* |
| Student ID | *300123350* |
| Email ID | *aaa4027@qatar-med.cornell.edu* |
| Date Submitted | *11/03/2024* |

**ASSESSMENT FEEDBACK**

|  |  |  |
| --- | --- | --- |
| **TO BE FILLED BY THE ASSESSOR** | | |
| Assessment type | Marks | Marks Awarded |
| Project Documentation | 70 |  |
| Viva and Presentation | 30 |  |
| Overall Marks achieved |  | |
| GRADE ACHIEVED |  | |
| **Summative Feedback by Assessor for further improvement** | | |
|  | | |
| **Comments for REDO submission (If applicable)** | | |
|  | | |

**GRADE DESCRIPTORS**

|  |  |
| --- | --- |
| 70% and above  (Distinction) | The Project evaluated is of a high to exemplary standard. The work addresses clearly and articulately the project requirements and thus meets and satisfies all the learning outcomes (either well or in an exemplary way). The work demonstrates: clear knowledge; references to appropriate academic literature; analysis; critical evaluation; and originality of argument. It is structured and presented to a high (or exemplary) standard. Referencing conventions are fully observed. |
| 60 to 69%  (Merit) | The project evaluated is of a good to a high standard. Substantial knowledge, comprehension and analysis is evident throughout. Arguments presented are clear and focussed with a logical structure in place. There is clear evidence of critical evaluation of a wide range of theories/perspectives from academic literature and some independent thought. The work is well-written and addresses well all of the learning outcomes. Referencing conventions are fully observed. |
| 50 to 59%  (Pass) | The project evaluated is of a fair to good standard. Adequate knowledge, comprehension and analysis is evident throughout. The arguments presented have a logical structure and show some critical evaluation in places, although there may be limited evidence of an independent perspective. There is evidence of some good engagement with some of the appropriate literature. Learning outcomes have been largely met and to an appropriate degree. Referencing conventions are observed. |
| 40 to 49%  (Fail/Redo) | The project evaluated is of a basic standard. The arguments presented have some logical structure and are supported by academic literature in most cases. The academic literature used is outside of the suggestions made in the module guide but remains limited. Little critical evaluation is evident, and the work tends more widely towards a descriptive style. Learning outcomes have been addressed in a basic but satisfactory way. Referencing conventions are mostly observed. |
| Fail Grades | |
| 30 to 39%  (Module retake) | The project evaluated is of a limited standard. Limited use of academic literature and as such knowledge and argument is very weak. A simple descriptive style with no evidence of critical evaluation throughout. Over-reliance on simplistic, limited sources. Referencing conventions may not be observed. Some learning outcomes met but in a weak and simplistic way. The work is needs to be developed in greater depth and detail to move to a passable standard at this level of study |
| 29% and Below  (Module retake) | The project evaluated is of an unacceptable standard. There is little or no evidence of knowledge and understanding that is required at this level. Referencing is inadequate or non-existent. The learning outcomes have not been addressed fully and the work requires significant modification to bring it to a passable standard. |

**General Guidelines**

1. Complete the ‘To be filled by the student section’ in the cover page.
2. Project documentation must be submitted as an electronic document in MS word (Use 12 Times New Roman script with 1.5 spacing between lines)
3. Submit the Project in MS word document with the file name being:

First Name Last Name \_ Program Code

Example: John Smith\_DS (DS abbreviated as Data Science)

**PG Diploma Dissertation**

**Overview:**

In this capstone project, students will be required to demonstrate the application of Artificial Intelligence and Machine Learning skills learned throughout the course. The project involves the end-to-end process of addressing a real-world problem using AI. This includes data gathering, cleaning, exploratory data analysis, visualization, and applying machine learning algorithms for solutions.

**Objectives:**

* Apply key concepts, algorithms, and technologies in a real-world context.
* Identify and address a significant challenge using AI and ML techniques.
* Efficiently develop and implement ML models or AI algorithms.
* Handle, preprocess, and visualize large datasets with relevance.
* Measure model performance using appropriate validation strategies.
* Address ethical concerns in AI/ML solution design and implementation.
* Ensure solution scalability and provide deployment strategies.
* Work effectively with team members, if applicable.
* Integrate current AI and ML advancements into the project.
* Maintain records and prepare comprehensive reports or presentations.
* Adapt and learn new AI/ML tools or techniques relevant to the project

**Sample Topics:**

1. **Title:** "Smart Energy Consumption Prediction in Households"

**Description:** Using historical data on energy consumption and related factors like weather conditions and household activities, this project will predict future energy demands. Such predictions can help households optimize their energy usage, contributing to more efficient and environmentally friendly homes.These are just examples, and you are encouraged to develop your own unique project related to your field of interest.

1. **Title**: "Automated Plant Disease Identification through Image Analysis"

**Description**: Farmers often struggle with identifying plant diseases at an early stage. This project seeks to develop a machine learning model that can identify and classify plant diseases based on images of the leaves. Early detection can lead to timely intervention, saving crops and increasing yields.

1. **Title**: "Real-time Anomaly Detection in Financial Transactions"

**Description**: With the increase in online transactions, fraud detection has become paramount. This project aims to develop a system that can identify potentially fraudulent transactions in real-time by analyzing patterns and anomalies, offering an additional layer of security for online financial activities.

1. **Title**: "AI-driven Virtual Personal Shopper"

**Description**: E-commerce platforms offer a vast array of products, often overwhelming consumers. This project intends to create a virtual shopping assistant that uses machine learning to understand user preferences and recommends products tailored to individual tastes and previous shopping behaviors.

1. **Title**: "Predictive Maintenance of Industrial Machines"

**Description**: Unscheduled machine downtimes in industries can lead to substantial financial losses. By analyzing sensor data from industrial machines, this project will predict when a machine is likely to fail or needs maintenance, allowing industries to schedule timely interventions and avoid unforeseen disruptions.

These are just examples, and you are encouraged to develop your own unique project related to your field of interest.

**Documentation:**

The following instructions provide guidance on how to compile the project documentation.

**Artificial Intelligence for Predicting Patients’ Responses to Thyroid Cancer Treatment**

# Abstract

**Background**: Thyroid cancer, one of the most prevalent endocrine malignancies, presents diverse treatment responses among patients, underscoring the necessity for personalized treatment strategies.

**Objective**: This study aims to develop and evaluate AI models that can accurately predict the initial treatment response among patients with well-differentiated thyroid cancer.

**Methods**: Utilizing a dataset comprising demographic and clinicopathological features from a retrospective cohort of 383 patients who received histopathological diagnoses of thyroid cancers from a single medical center, we trained and validated various predictive models, including random forests, support vector machines, and decision trees.

**Results**: Models in all experiments achieved an average accuracy of 72.2%, average precision of 76.2%, average recall of 68%, average F1 score of 70.6%, and average AUC of 80.5%. Multi-layer Perceptron achieved the highest accuracy (81.8%), recall (75%), and F1 score (81.7%) in this study when it was used for predicting binary treatment response using all features, excluding those weakly correlated with the treatment response. However, Naive Bayes attained the highest precision (95.1%) and AUC (88.7%) in this study when it was used for predicting binary treatment response using all features.

**Conclusion**: AI demonstrates satisfactory performance in predicting the response to thyroid cancer treatment, yet there is room for optimization. Healthcare providers should not solely rely on our models; combining them with other tools is advised until further studies validate their optimal performance. Future research should enhance predictive capabilities by including additional features, exploring alternative models, and utilizing larger and balanced datasets.

# Introduction

## Background

Thyroid cancer is one of the most common endocrine malignancies that results from abnormal growth and division of cells in the thyroid, a gland located beneath the voice box at the front of the throat [1]. Worldwide, it is estimated that over 586,202 new cases of thyroid cancer emerged in 2020 (449,000 cases in women and 137,000 in men) [2]. Thyroid cancer includes various types. Papillary thyroid cancer is the most prevalent, often exhibiting slow growth. Follicular thyroid cancer arises from thyroid follicular cells and may spread to nearby tissues (e.g., bones and lungs). Medullary thyroid cancer originates in specialized C cells and may be hereditary. Anaplastic thyroid cancer, though rare, is aggressive and rapidly spreads to the surrounding lymph nodes and distant sites [1, 3, 4]. The vast majority, over 90%, of thyroid cancers are classified as differentiated thyroid cancer; a type of thyroid cancer that retains some features of normal thyroid tissue [5]. Papillary thyroid cancer and follicular thyroid cancer are considered subtypes of differentiated thyroid cancer [3].

The treatment for thyroid cancer often involves a combination of approaches [6]. Surgical intervention, often in the form of a thyroidectomy, stands as the primary method for removing the affected thyroid tissue. After thyroidectomy, radioactive iodine therapy may be used, especially for types of thyroid cancer responsive to iodine, to assist in eliminating any remaining cancer cells. Thyroid hormone therapy (e.g., levothyroxine (LT4)) is commonly prescribed not only to replace lost hormones after thyroid removal but also to suppress thyroid-stimulating hormone, thereby reducing the risk of cancer recurrence. In cases where surgery and radioactive iodine may not be sufficient, external beam radiation therapy may be used. For more aggressive or advanced thyroid cancers, targeted therapies and chemotherapy may be considered to impede cancer cell growth and dissemination. The selection of treatments is highly personalized, taking into account the thyroid cancer type, stage, and individual patient factors [6].

Response to thyroid cancer treatment (usually surgery and radioactive iodine treatment) is categorized into "Excellent," "Indeterminate," "Biochemical incomplete," and "Structural incomplete" [6]. An "Excellent Response" indicates a very favorable outcome with no clinical, biochemical, or structural evidence of disease, meaning that physical examinations, imaging tests, and blood tests, such as thyroglobulin levels (a tumor marker for thyroid cancer), show no signs of the disease [6]. The "Indeterminate Response" category is applied when the results are unclear or uncertain, such as ambiguous imaging findings that cannot definitively be identified as cancer, or detectable thyroglobulin levels without clear evidence of remaining cancer cells [6]. A "Biochemical Incomplete Response" suggests that, despite the absence of cancer on imaging, elevated thyroglobulin levels in the blood imply the likely presence of unseen cancer cells [6]. Lastly, a "Structural Incomplete Response" is characterized by visible evidence of remaining cancer on imaging tests, regardless of thyroglobulin levels, indicating that structural remnants of the disease, like nodules or tumors, persist despite the treatment [6]. Because thyroid cancer varies in type, stage, and how it affects each person, treatment responses can differ significantly. In other words, the effectiveness of thyroid cancer treatment is highly individualized.

## Research Problem, Significance, and Relevance

Many studies showed that Artificial Intelligence (AI) has significant potential in the field of thyroid cancer such as detecting and differentiating between different types of thyroid tumors [7-9], stratifying its risk [10], assessing its prognosis [11, 12], predicting its recurrence [13-15], and predicting its treatment complications [16]. However, very few studies assessed the effectiveness of AI in predicting the response of thyroid cancer treatment [17-19]. These studies also have some limitations. Firstly, they did not consider important features in their models such as current smoking status, past smoking history, history of radiation therapy, presence of goiter, presence of adenopathy, and risk assessment according to American Thyroid Association (ATA) guidelines [17-19]. Secondly, they did not compare the performance of AI in predicting a 4-class treatment response versus a 2-class treatment response [17-19]. Thirdly, the follow-up period after the treatment was short (<10 years) [17-19]. Fourthly, they did not compare the performance of different AI models [17, 18]. Lastly, they focused on a specific type of thyroid cancer (e.g., patients without structural disease) [19].

Given the above-mentioned limitations, it is very important to conduct further research to accurately predict a patient's response to thyroid cancer treatment in order to tailor personalized care and optimize outcomes. Specifically, accurate predictions enable the adjustment of treatment strategies, ensuring that patients receive the most effective interventions. For those predicted to have a lower response to standard treatments, it opens doors to exploring more aggressive therapies sooner. This proactive approach not only enhances the chances of successful outcomes but also minimizes potential side effects and complications and improves healthcare resource allocation. Therefore, there is an urgent need to develop tools for predicting the response of thyroid cancer treatment.

## Project Objectives

The aim of the project is to develop and evaluate AI models that can accurately predict the initial treatment response among patients with well-differentiated thyroid cancer. This aim will be achieved through the following objectives:

1. To find a public dataset related to this topic in order to use it in developing the model.
2. To perform preprocessing for the dataset.
3. To train various AI models using different sets of features.
4. To evaluate the performance of these models in predicting 2 or 4-class responses to treatment.
5. To formulate recommendations for healthcare providers, researchers, and policymakers.

## Project Scope and Constraints

The scope of this project includes the development of AI models for predicting the treatment response among patients diagnosed with well-differentiated thyroid cancer, evaluating them, and providing recommendations for healthcare providers, researchers, and policymakers. However, other outcomes (e.g., detecting thyroid cancer, differentiating its types, and predicting its recurrence) and other populations (patients with non-differentiated thyroid cancer or any other kind of cancers) are out of the scope of this review. Further, the project does not develop AI models for predicting the response of a specific thyroid cancer treatment (e.g., thyroidotomy or radioactive iodine therapy) given that this is not possible using the available data.

The current project may have the following constraints: (1) limited access to large high-quality data, (2) potential unrepresentative or biased data, and (3) lack of access to high-performance computing if the data is massive. The project will consistently consider ethical concerns to ensure data privacy, responsible data usage, and clear interpretability of the model.

By achieving the above-mentioned objectives within the specified scope and taking into account the associated constraints, this project aims to offer practical insights and solutions to enhance the prediction of initial treatment response in patients with well-differentiated thyroid cancer.

# Methodology

Several experiments will be carried out to develop and evaluate AI models for accurately predicting the initial treatment response among patients with well-differentiated thyroid cancer. To be more precise, the following steps will be undertaken:

1. Data acquisition: The following sources and repositories will be searched to find a dataset related to the initial treatment response of patients with well-differentiated thyroid cancer: GitHub, Kaggle, UC Irvine Machine Learning Repository, and Google Dataset Search. The data set should contain important features for predicting the initial treatment response.
2. Data preprocessing: The acquired data will be preprocessed and cleaned by addressing missing values, removing outliers, conducting data visualization and exploratory data analysis (EDA), and performing necessary data encoding and transformations using Python libraries such as Pandas, NumPy, Matplotlib, and Seaborn.
3. Feature engineering: Features that are important for predicting the initial treatment response will be identified and extracted from the data. In this step, new features may be generated, or existing features will be transformed to capture relevant information such as treatment timestamp, patient demographics, treatment specifics, tumor characteristics (e.g., type and size), and laboratory results (hormone levels over time).
4. Data splitting: The dataset will be divided into training and testing sets. The training set will be used for training the AI models, while the testing set will be used to assess their predictive performance. Training test split approach will be followed for splitting the dataset using the Scikit-learn library.
5. Model training: The training data will be used to train several AI models such as logistic regression, decision trees, random forests, or support vector machines based on the characteristics of the data and the study's objective. The model will be trained using the fit function provided by the Scikit-learn (Sklearn) library, and hyperparameters will be adjusted through experimentation or cross-validation techniques.
6. Model evaluation: The performance of trained models in predicting initial treatment response will be assessed using new, unseen data (testing data). The predict function in the Scikit-learn library will be used to test the performance of the models. Several metrics will be used to assess the performance of the models such as accuracy, sensitivity (recall), precision, and AUC. Graphs will be created to visualize the results in an appropriate way.
7. Interpretation and Recommendations: The findings will be carefully interpreted to offer actionable recommendations for enhancing treatment response prediction, potentially informing treatment strategies or personalized interventions.

# Project Implementation

## Data Acquisition and Uploading

Here, the above-mentioned project’s plan was executed step by step starting with the selection of an appropriate dataset. To do so, I searched GitHub, Kaggle, UC Irvine Machine Learning Repository, and Google Dataset Search. Several datasets were found; however, I selected the one that was posted by Shiva Borzooei on UC Irvine Machine Learning Repository [20] for the following reasons: (1) it contains the initial treatment response as a variable, (2) it is a very recent dataset that was posted on the 30th of October 2023, (3) it was not used for the same purpose of this project, and (4) it contains several variables (features) different from those were considered in the previous studies. This dataset was downloaded and saved on my personal computer.

After this, the following libraries were imported as they are necessary for implementing the project:

NumPy (numpy): This Python library is fundamental for numerical computing [21]. It allows users to work with arrays and matrices efficiently and perform various mathematical operations on them. It was imported as np.

Pandas (pandas): This Python library is designed for data manipulation and analysis [22]. It provides a powerful data structure called DataFrame, which makes handling and analyzing structured data much easier. It was imported as pd.

Matplotlib (matplotlib.pyplot): This is a popular library for creating visualizations of data [23]. It offers a wide range of options for creating various plots, charts, and graphs. It was imported as plt.

Seaborn (seaborn): This library builds on top of Matplotlib and provides additional features for creating informative and visually appealing data visualizations [24]. It is especially useful for creating statistical graphics. It was imported as sns.

A white rectangular object with a black border

Description automatically generated

Then, I uploaded the data stored in my computer and created a dataframe called "df".

A screenshot of a computer

Description automatically generated

## Exploratory Data Analysis

After uploading the dataset, I checked its size, column names, data types, and the number of missing values using the functions “df.shape” and “df.info”.

A screenshot of a computer

Description automatically generated

As shown in the figure above, the dataset consists of 383 rows (number of participants/entries) and 17 columns (number of features/variables). The data type was object (categorical) for all variables except for age, which was integer. There are no missing values in all variables. The variables were as follows:

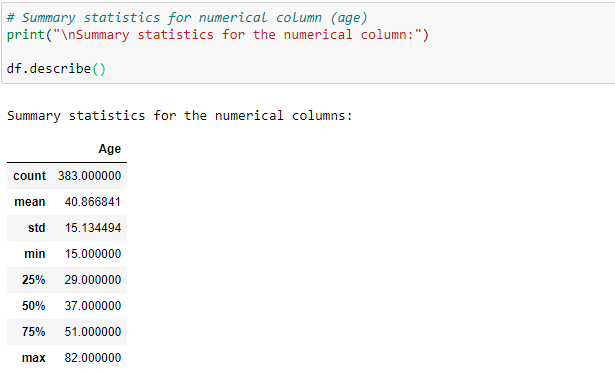
1. Age: The age of the patient at diagnosis in years.
2. Gender: Biological sex of the patient (Female and Male).
3. Smoking: Current smoking status (Yes and No).
4. Hx Smoking: Past smoking history (Yes and No).
5. Hx Radiotherapy: History of radiation therapy to the head and neck region (Yes and No).
6. Thyroid Function: Thyroid function (classified as euthyroid, clinical, or subclinical hypo/ hyperthyroidism).
7. Physical Examination: Presence of goiter (diffuse, single nodular goiter on the left or right lobe, multinodular, or normal).
8. Adenopathy: Presence of adenopathy on physical examination (Yes and No).
9. Pathology: Pathological type of thyroid cancer (papillary, micropapillary, follicular, Hürthle cell).
10. Focality: Either unifocal (tumor is concentrated in one primary site) or multifocal (the presence of multiple distinct cancerous areas within the same organ or tissue).
11. Risk: Risk assessment according to American Thyroid Association (ATA) guidelines (low, intermediate, high)
12. T: The extent and severity of cancer based on the size and extent of the primary tumor (T). This is usually followed by a number (T0, T1, T2, etc.), with higher numbers indicating larger or more invasive tumors.
13. N: The extent and severity of cancer based on the involvement of nearby lymph nodes (N). This is followed by a number (N0, N1, N2, etc.), indicating the presence and extent of lymph node involvement.
14. M: The extent and severity of cancer based on the presence or absence of distant metastasis (M), which refers to the spread of cancer cells to other organs or tissues beyond the primary tumor and its nearby lymph nodes. The M category is denoted as M0 if there is no distant metastasis and M1 if metastasis is present.
15. Stage: An overall stage based on T, N, and M. It is expressed as stages I to IV, with higher stages indicating more advanced cancer.
16. Response: initial treatment response (excellent, biochemical incomplete, structurally incomplete, indeterminate).
17. Recurred: the recurrence of cancer after the initial treatment (Yes and No).

Given that the "Recured" variable was used as the target variable in the original study [13] and the target variable in this project is "Response", I removed the column "Recured".

A screenshot of a computer

Description automatically generated

To explore the data more, summary statistics were carried out for the integer (numerical) variable, which is "Age". The figure below shows the mean age was about 40.9 years with a range of 15-82.



A bar chart was used to check the distribution of age-related data. The figure below shows the age-related data is slightly skewed to the right side, and this is expected given that participants younger than 15 were not eligible in the original study [13].

A graph with blue lines and white text

Description automatically generated with medium confidence

Further, bar charts (count plots) were created to visualize each categorical variable. The bar charts showed that there is an imbalance in categories in all variables, especially the target variable (Treatment response), and this should be taken into account when developing the models.

A computer code on a white background

Description automatically generated

A blue and orange rectangles

Description automatically generated

A screenshot of a graph

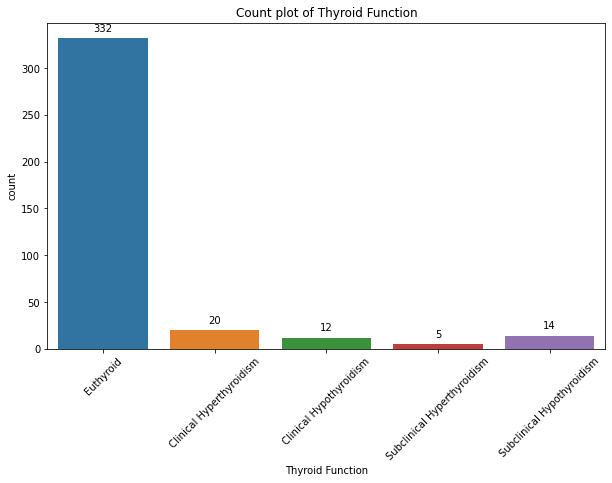
Description automatically generated

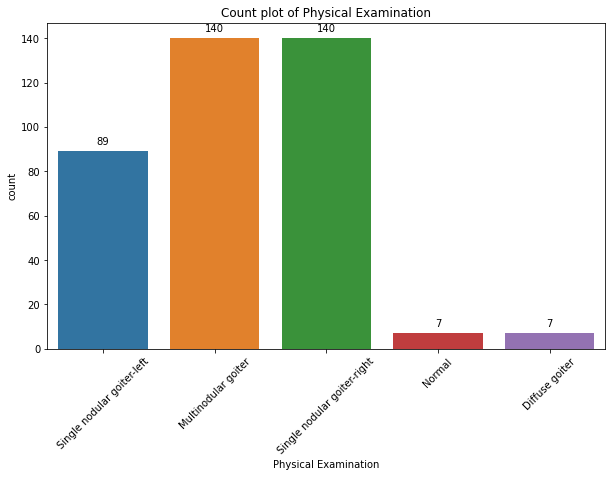
A screenshot of a graph

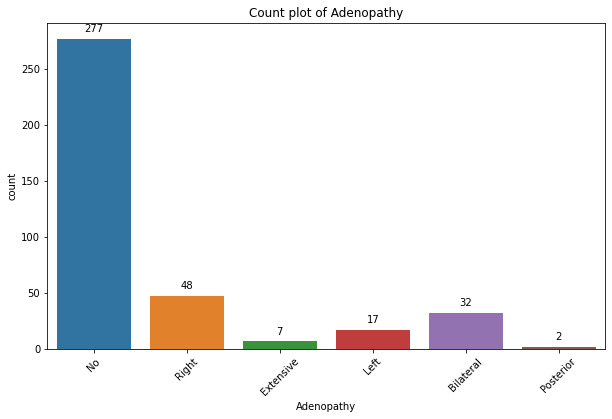
Description automatically generated

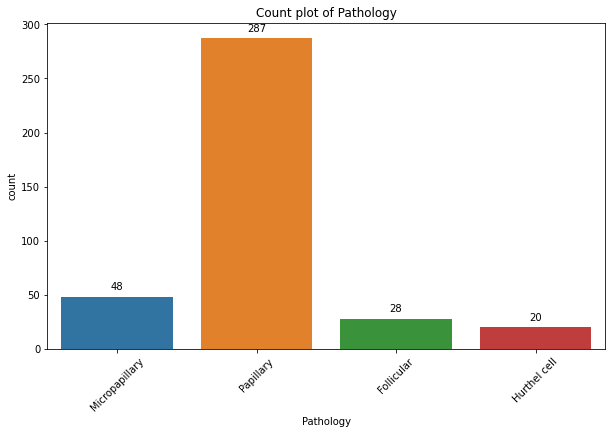
A screenshot of a computer

Description automatically generated









A screenshot of a graph

Description automatically generated

A screenshot of a graph

Description automatically generated

A graph with different colored bars

Description automatically generated

A graph with different colored squares

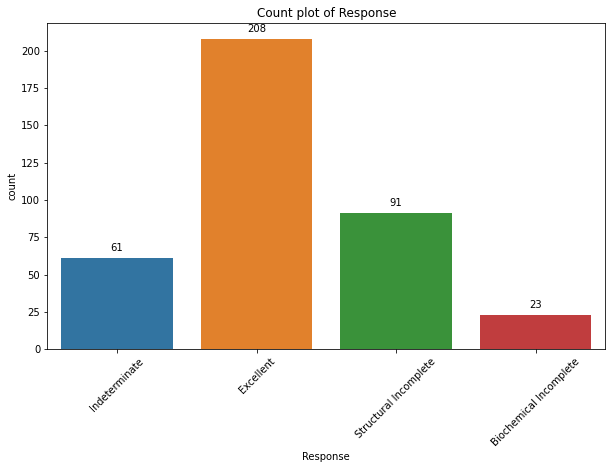
Description automatically generated

A screenshot of a graph

Description automatically generated

A screenshot of a graph

Description automatically generated



Given that the data type in all variables except “Age” was object (categorical), the "map" function was used to encode these variables to become integer variables. LabelEncoder() function could be used to make the process shorter, but I selected the "map" function to know the value assigned to each category, especially for the target variable (treatment response). The following code shows both techniques. As shown in the Table 1, the assigned values for each category are as follows:

Table 1: Assigned values for each category

|  |  |  |
| --- | --- | --- |
| **Variable** | **Category** | **Encoded value** |
| **Gender** | M | 0 |
| F | 1 |
| **Smoking** | No | 0 |
| Yes | 1 |
| **Hx Smoking** | No | 0 |
| Yes | 1 |
| **Hx Radiotherapy** | No | 0 |
| Yes | 1 |
| **Thyroid Function** | Euthyroid | 1 |
| Clinical Hyperthyroidism | 2 |
| Subclinical Hyperthyroidism | 3 |
| Clinical Hypothyroidism | 4 |
| Subclinical Hypothyroidism | 5 |
| **Physical Examination** | Normal | 1 |
| Multinodular goiter | 2 |
| Single nodular goiter-right | 3 |
| Single nodular goiter-left | 4 |
| Diffuse goiter | 5 |
| **Adenopathy** | No | 0 |
| Right | 1 |
| Left | 2 |
| Bilateral | 3 |
| Extensive | 4 |
| Posterior | 5 |
| **Pathology** | Papillary | 1 |
| Micropapillary | 2 |
| Follicular | 3 |
| Hurthel cell | 4 |
| **Focality** | Uni-Focal | 0 |
| Multi-Focal | 1 |
| **Risk** | Low | 1 |
| Intermediate | 2 |
| High | 3 |
| **T** | T1a | 1 |
| T1b | 2 |
| T2 | 3 |
| T3a | 4 |
| T3b | 5 |
| T4a | 6 |
| T4b | 7 |
| **N** | N0 | 1 |
| N1a | 2 |
| N1b | 3 |
| **M** | M0 | 0 |
| M1 | 1 |
| **Stage** | I | 1 |
| II | 2 |
| III | 3 |
| IVA | 4 |
| IVB | 5 |
| **Response** | Excellent | 1 |
| Indeterminate | 2 |
| Structural Incomplete | 3 |
| Biochemical Incomplete | 4 |



Then, I checked the data summary after encoding to ensure that the data type changed as required and no issues happened during the encoding. The data type for all variables is integer and no missing values resulted from this process. A screenshot of a computer

Description automatically generated

A screenshot of a computer screen

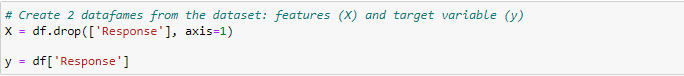
Description automatically generatedThe correlation between the target variable and other variables (features) was calculated to identify which features I should consider when developing the models. The results were shown using a heatmap graph. As shown in the figure below, the target variable has a very low correlation (<0.1) with the features “thyroid function” and “pathology” whereas it has a moderate-to-strong correlation (≥0.30) with the features "Smoking", "Adenopathy", "Focality", "Risk", "T", "N", and "Stage". So, these differences in correlations will be considered when developing the models as I will explain later. It is noteworthy that I did not carry out data cleaning (apart from removing not needed column) given that the dataset was clean, and no missing data was available.

## Model Development and Evaluation

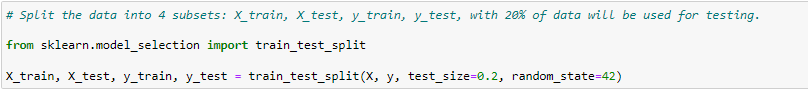
The previous steps showed that there are 2 issues: (1) the number of participants in categories (classes) is imbalanced, especially in the target variable, and (2) some features have low correlation with the target variable. Therefore, to overcome the first issue, I tried to make the number of participants in categories (classes) in the target variable balance using 2 techniques. The first technique is to use sampling strategies such as Synthetic Minority Over-sampling Technique (SMOTE) (i.e., generating additional instances of the minority class to make it more proportionate to the majority class), RandomOverSampler (e.g., unlike SMOTE, not generating synthetic instances but rather replicates existing instances from the minority class to balance the dataset), and RandomUnderSampler (i.e., randomly removes instances from the majority class until a more balanced distribution is achieved). The second technique is to group classes in the target variable into 2 classes: excellent and non-excellent. To overcome the second issue, I assessed the performance of AI models using 3 different sets of features: (1) all features, (2) without weakly correlated features (<0.1), and (3) with only moderately and strongly correlated features (>0.3). Therefore, 9 experiments were carried out to identify the best AI model to predict the treatment response. These experiments are detailed in the following 9 subsections.

### 1. All features & 4 imbalanced classes

In this experiment, I developed and evaluated models using all features to predict 4 imbalanced classes of the target variable. I created 2 dataframes; one (X) contains only the features and the other (y) contains the target variable (treatment response).



Then, I imported train\_test\_split from the Sklearn library [25] to split the data into 4 subsets: X\_train, X\_test, y\_train, and y\_test. The value of parameter test\_size was set to 0.2, which means that 80% of the data will be used for training while the remaining data (20%) will be used for testing. This value was selected because the dataset is small. The value of the parameter “random\_state” was set to 42 (which is arbitrary) to ensure that the dataset is split in the same way every time I run the code. This helps in obtaining consistent and reproducible results.



The following models were developed in this project to select the most appropriate one: logistic regression (LogR), decision trees (DT), random forests (RF), or support vector machines (SVM), Naïve Bayes (NB), K-Nearest Neighbor (KNN), and Multi-layer Perceptron (MLP). All these models are Machine learning models except for MLP, which is a deep learning model. Several performance measures were used to assess the performance of these models: Accuracy, recall (sensitivity), precision, F1 score, ROC curve, and AUC. Thus, these models and metrics were imported from the Sklearn library [25].

A white screen with black text

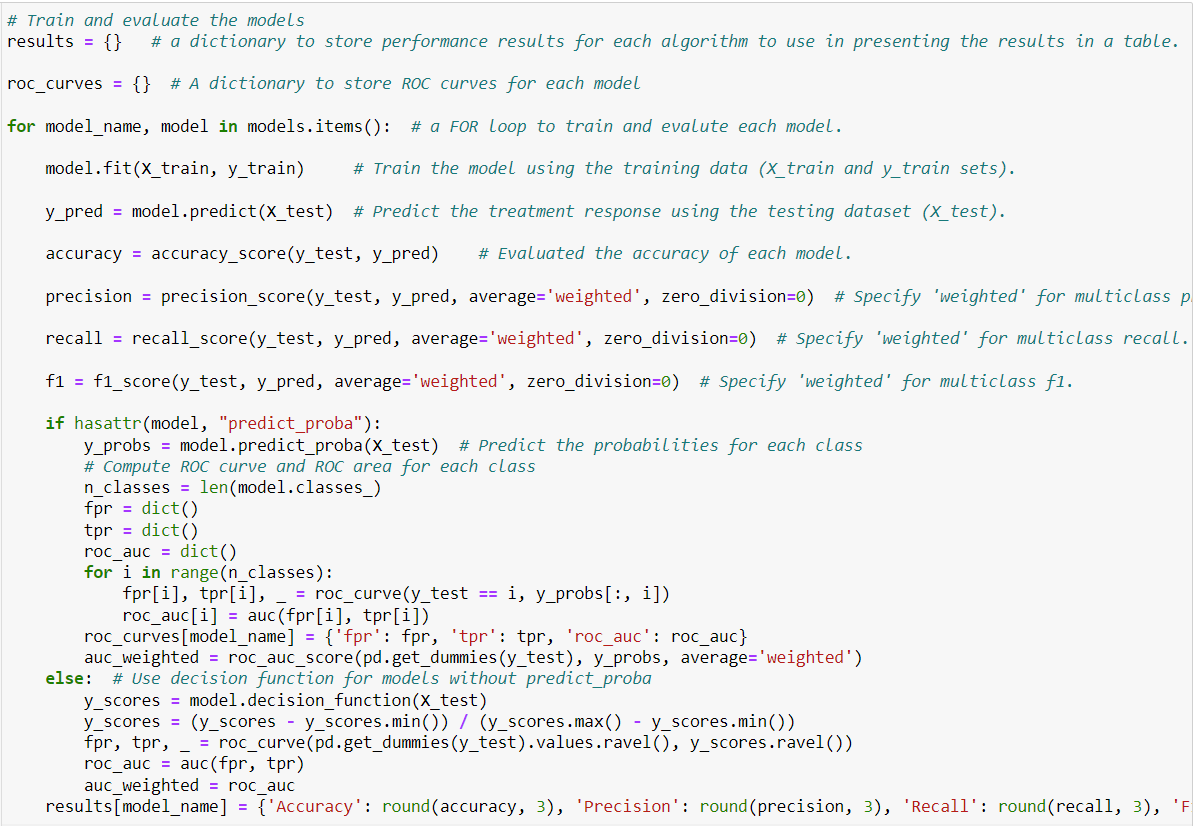
Description automatically generated

A dictionary of imported models was created to use it in training and testing. For all models except for NB and KNN, I set random\_state at 42 to control the randomness of the model's internal random number generator, thereby ensuring the model reproducibility each time I run the code. For LogR and MLP models, I also set max\_iter at 1000, which refers to the maximum number of iterations the model will perform during the training process. For the MLP model, I set hidden\_layer\_sizes at 100, which refers to the number of neurons in each hidden layer of the MLP. In our model, there is one hidden layer with 100 neurons.

A computer code with black text

Description automatically generated with medium confidence

I created 2 dictionaries to help present the results. Specifically, the dictionary “results” was created to store performance results for each algorithm to use in presenting the results in a table. The dictionary “roc\_curves” was created to store ROC curves for each model to help calculate AUC. After that, I created a “for” loop to train (fit) each model using the “fit” function and the training data (X\_train and y\_train sets). Then, the trained models predicted the treatment response using the “predict” function with an unseen testing dataset (X\_test). Lastly, the above-mentioned metrics were calculated to evaluate the performance of each model. For precision, recall, f1, and AUC scores, the average='weighted' parameter is used to compute metrics for each label and then average them based on the number of true instances for each label. This is attributed to the fact this is a multi-class classification problem. For precision, recall, and f1 score, I set the parameter “zero\_division” at 0 to assign a score of 0 when there is a division by zero (i.e., when precision, recall, and/or f1 score is undefined because there are no true positive predictions). The results were rounded to the nearest 3 decimal places and stored in the dictionary “results”.



Results stored in the dictionary “Results” was converted to a dataframe to display them in a tabular format. I used a transpose operation (T) to swap the rows and columns of the dataframe so that each row corresponds to a model, and each column corresponds to a performance metric (e.g., Accuracy, Precision, Recall, F1 Score, AUC).

A screenshot of a computer

Description automatically generated

I also present the results for all algorithms in a bar chart for better comprehension. In the bar chart, the x-axis represents the models while the y-axis represents the performance score. The legend for each metric is shown in the left top corner. The bar width is 0.18 inches. The figure has a width of 13 inches and a height of 8 inches.

A screenshot of a computer code

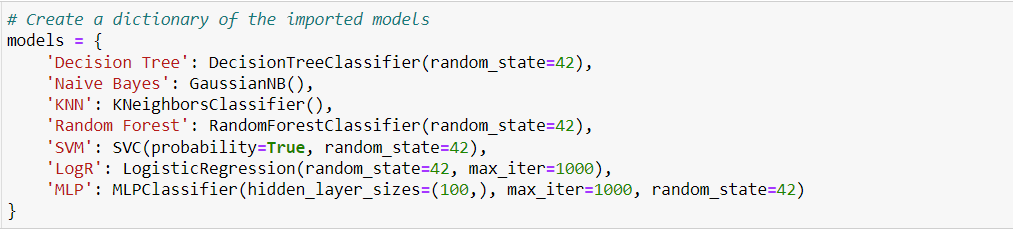
Description automatically generated

A graph of different colored vertical lines

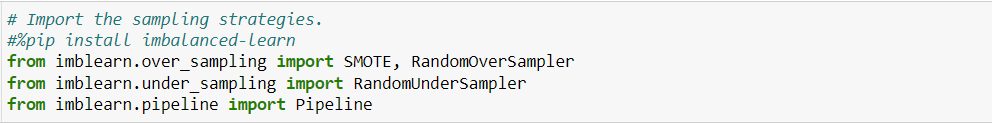
Description automatically generated with medium confidence

### 2. All features & 4 balanced classes

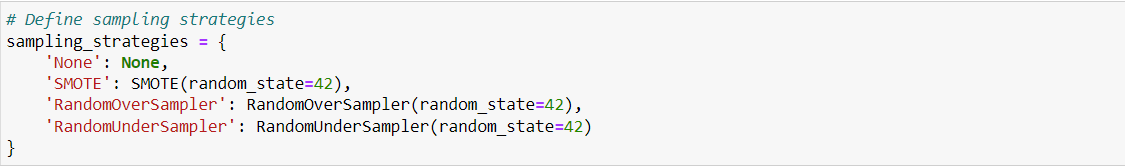
In this experiment, I developed and evaluated models using all features to predict 4 balanced classes of the target variable. The classes in the target variable were balanced using the above-mentioned sampling strategies (i.e., SMOTE, RandomOverSampler, RandomUnderSampler). To this end, a dictionary of imported models was created to use it in training and testing. For all models except for NB and KNN, I set random\_state at 42 to control the randomness of the model's internal random number generator, thereby ensuring the model reproducibility each time I run the code. For LogR and MLP models, I also set max\_iter at 1000, which refers to the maximum number of iterations the model will perform during the training process. For the MLP model, I set hidden\_layer\_sizes at 100, which refers to the number of neurons in each hidden layer of the MLP. In our model, there is one hidden layer with 100 neurons.



I imported the 3 sampling strategies from the imbalanced-learn library [26]. I also imported the Pipeline class from the imblearn.pipeline module to create a workflow that includes resampling techniques and AI models. This helps ensure that the resampling is applied consistently to both the training and testing datasets. It is worth noting that I should install the imbalanced-learn library if it is not already installed.



I created a dictionary named sampling\_strategies to store the imported sampling strategies, each associated with a key. The key “None” is associated with the value “None”, which suggests that no resampling is applied to the data. The key “SMOTE” is associated with the value “SMOTE”, which suggests that SMOTE is applied to the data for oversampling the minority class by generating synthetic samples. The key “RandomOverSampler” is associated with the value “RandomOverSampler”, which suggests that RandomOverSampler is applied to data to randomly duplicate samples from the minority class to balance the class distribution. The key “RandomUnderSampler” is associated with the value “RandomUnderSampler”, which suggests that RandomUnderSampler is applied to data to randomly remove samples from the majority class to balance the class distribution. The parameter “random\_state” in the sampling strategies was set at 42 to control the randomness of the model's internal random number generator, thereby ensuring the model reproducibility each time I run the code.

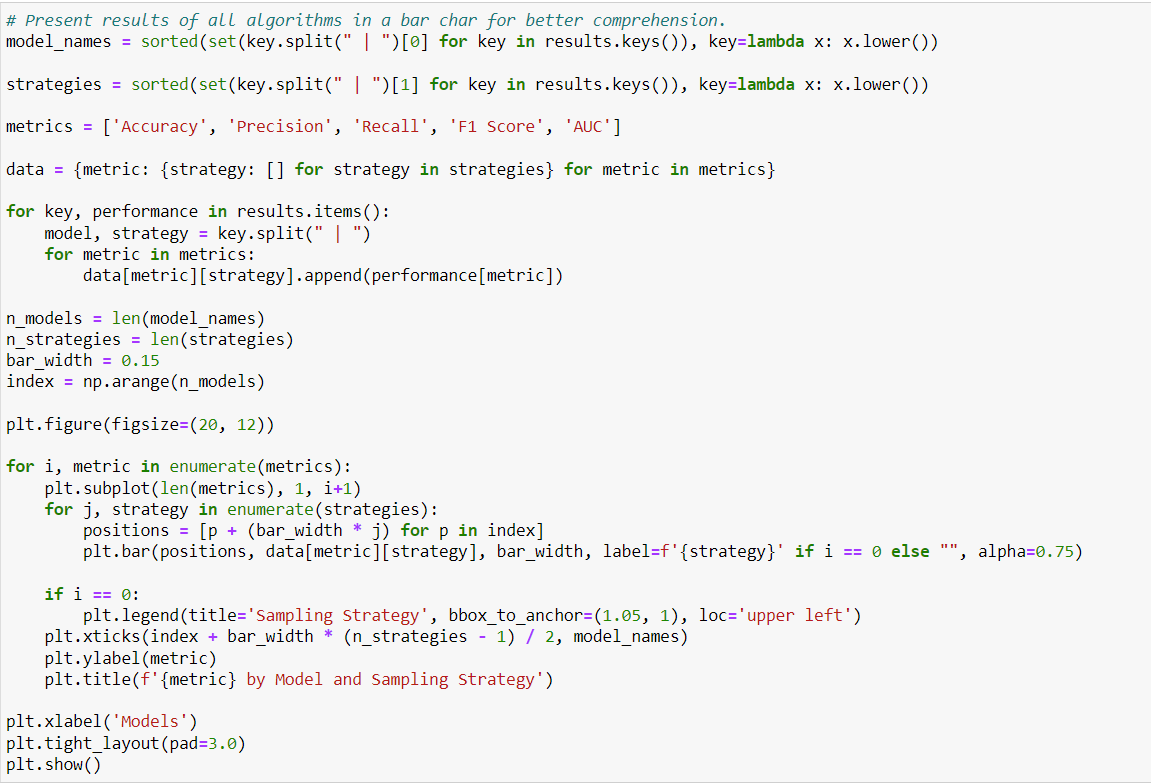


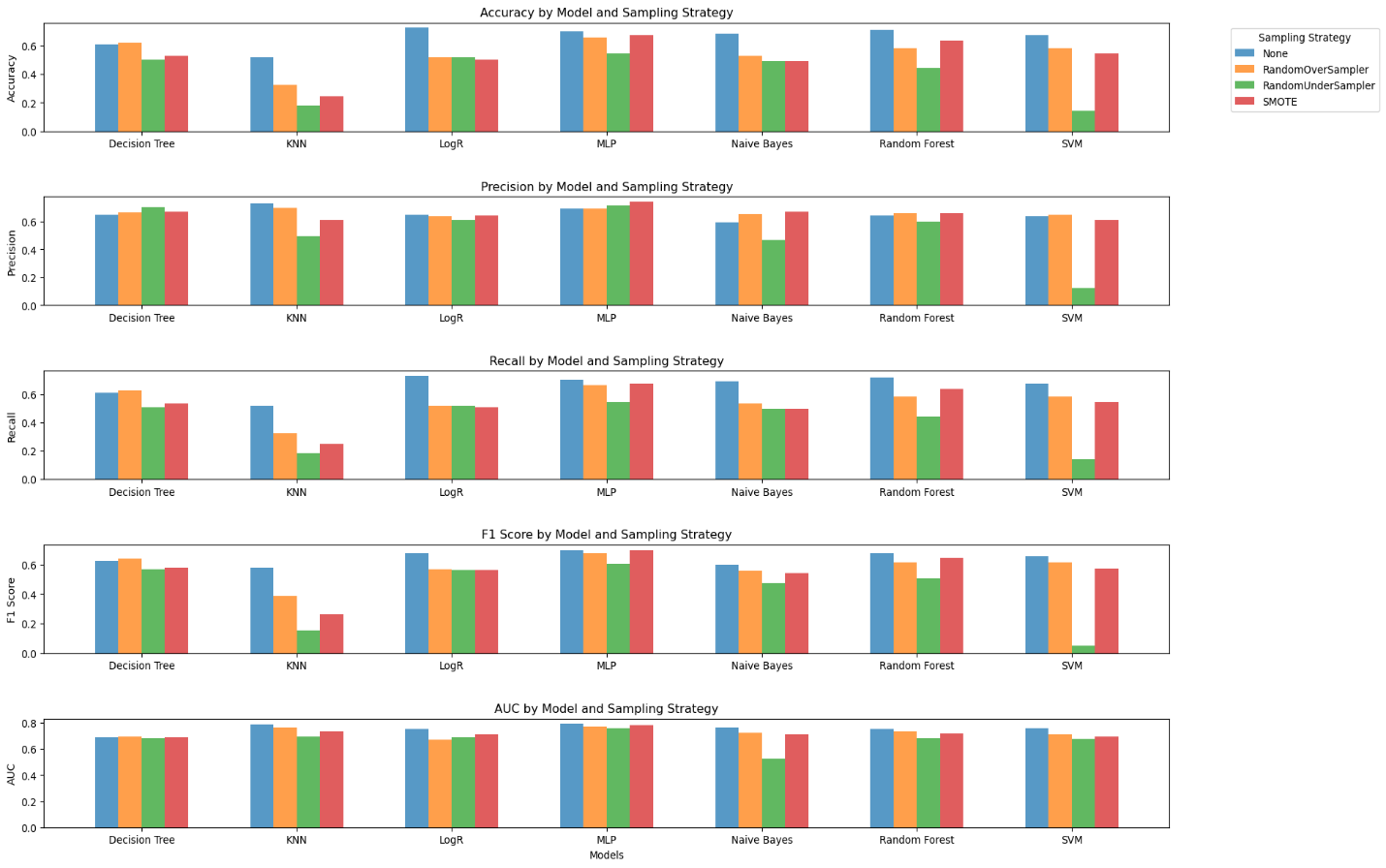
I created an empty dictionary named results that will be used to store the results for each model to present them in a table. A nested loop was created to iterate over each combination of sampling strategy and AI model. The variables strategy\_name and sampler are assigned the key-value pairs from the sampling\_strategies dictionary, where the key is a string representing the name of the sampling strategy, and the value is the corresponding sampler (e.g., SMOTE, RandomOverSampler). Similarly, model\_name and model are assigned the key-value pairs from the models dictionary, where the key is a string representing the name of the AI model, and the value is the actual model object. I also created a pipeline for the current combination of sampling strategy and model. If a sampling strategy is specified (i.e., sampler is not None), the pipeline includes a sampling step followed by a modeling step. If no sampling strategy is specified (i.e., sampler is None), the pipeline only includes the modeling step. The pipeline was fitted to the training data (X\_train and y\_train) and then the fitted pipeline was used to make predictions on the test data (X\_test). The results of these predictions (y\_pred) were used for evaluation. For precision, recall, f1, and AUC scores, the average='weighted' parameter is used to compute metrics for each label and then average them based on the number of true instances for each label. This is attributed to the fact this is a multi-class classification problem. For precision, recall, and f1 score, I set the parameter “zero\_division” at 0 to assign a score of 0 when there is a division by zero (i.e., when precision, recall, and/or f1 score is undefined because there are no true positive predictions). The results for the current model and sampling strategy combination were rounded to the nearest 3 decimal places and stored in the results dictionary.

A screenshot of a computer program

Description automatically generated

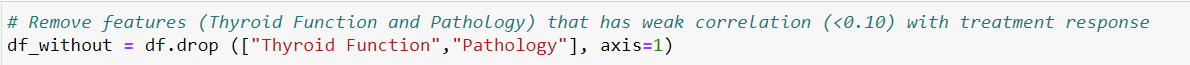
Grouped bar charts were created using Matplotlib [23] to visually compare the performance metrics (e.g., accuracy, precision, recall, F1 score, AUC) of different AI models under various sampling strategies. To do so, unique model names and sampling strategies were extracted from the keys of the results dictionary. The split (" | ") operation is used to separate the model name and strategy in each key. The set ensures uniqueness, and sorted organizes them alphabetically (case-insensitive). A dictionary called “data” was created to store performance metrics for each strategy and metric. The dictionary has the structure data[metric][strategy] = [], where metric is one of the evaluation metrics, and strategy is one of the sampling strategies. A nested loop was created to populate the data dictionary with the corresponding performance metrics for each model, strategy, and metric. I identified some parameters that will be used when creating a group bar chart such as the width of the bar (0.15), number of models and strategies, and figure size (20\*12 inches). I created a nested loop to iterate over each performance metric ('Accuracy', 'Precision', 'Recall', 'F1 Score', 'AUC) using the enumerate function to obtain both the index i and the corresponding metric name. A subplot within the overall figure for each metric was created. The len (metrics), 1 specifies the number of rows and columns in the subplot grid, and i+1 is the index of the current subplot. The inner loop (1) iterates over each sampling strategy using enumerate to obtain both the index j and the strategy name, (2) calculates the x-positions for the bars within the current subplot, (3) plots a bar chart within the current subplot using the calculated positions, the performance data (data[metric][strategy]), and bar width (bar\_width). The legend indicates the sampling strategy, and the x-axis represents the different models. The chart is organized in a grid. The entire chart is then displayed using plt.show().



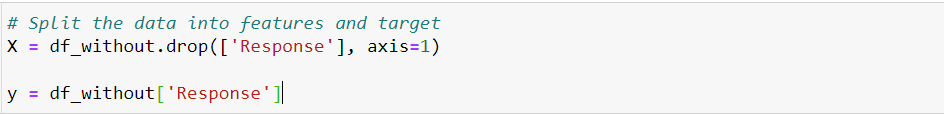


### 3. Without weakly correlated features & 4 imbalanced classes

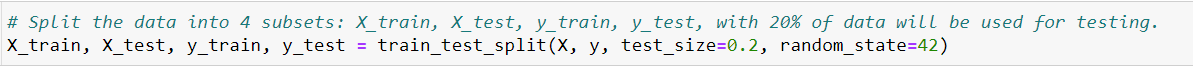
In this experiment, I removed features that have a weak correlation (<0.10) with treatment response (Thyroid Function and Pathology). Then, I developed and evaluated models using the remaining features to predict 4 imbalanced classes of the target variable.



I created 2 dataframes; one (X) contains only the features and the other (y) contains the target variable (treatment response).



Then, the train\_test\_split function from the Sklearn library [25] was used to split the data into 4 subsets: X\_train, X\_test, y\_train, and y\_test. The value of parameter test\_size was set to 0.2, which means that 80% of the data will be used for training while the remaining data (20%) will be used for testing. This value was selected because the dataset is small. The value of the parameter “random\_state” was set to 42 (which is arbitrary) to ensure that the dataset is split in the same way every time I run the code. This helps in obtaining consistent and reproducible results.



A dictionary of imported models was created to use it in training and testing. For all models except for NB and KNN, I set random\_state at 42 to control the randomness of the model's internal random number generator, thereby ensuring the model reproducibility each time I run the code. For LogR and MLP models, I also set max\_iter at 1000, which refers to the maximum number of iterations the model will perform during the training process. For the MLP model, I set hidden\_layer\_sizes at 100, which refers to the number of neurons in each hidden layer of the MLP. In our model, there is one hidden layer with 100 neurons.

A computer code with black text

Description automatically generated with medium confidence

I created 2 dictionaries to help present the results. Specifically, the dictionary “results” was created to store performance results for each algorithm to use in presenting the results in a table. The dictionary “roc\_curves” was created to store ROC curves for each model to help calculate AUC. After that, I created a “for” loop to train (fit) each model using the “fit” function and the training data (X\_train and y\_train sets). Then, the trained models predicted the treatment response using the “predict” function with an unseen testing dataset (X\_test). Lastly, the above-mentioned metrics were calculated to evaluate the performance of each model. For precision, recall, f1, and AUC scores, the average='weighted' parameter is used to compute metrics for each label and then average them based on the number of true instances for each label. This is attributed to the fact this is a multi-class classification problem. For precision, recall, and f1 score, I set the parameter “zero\_division” at 0 to assign a score of 0 when there is a division by zero (i.e., when precision, recall, and/or f1 score is undefined because there are no true positive predictions). The results were rounded to the nearest 3 decimal places and stored in the dictionary “results”.

A screenshot of a computer code

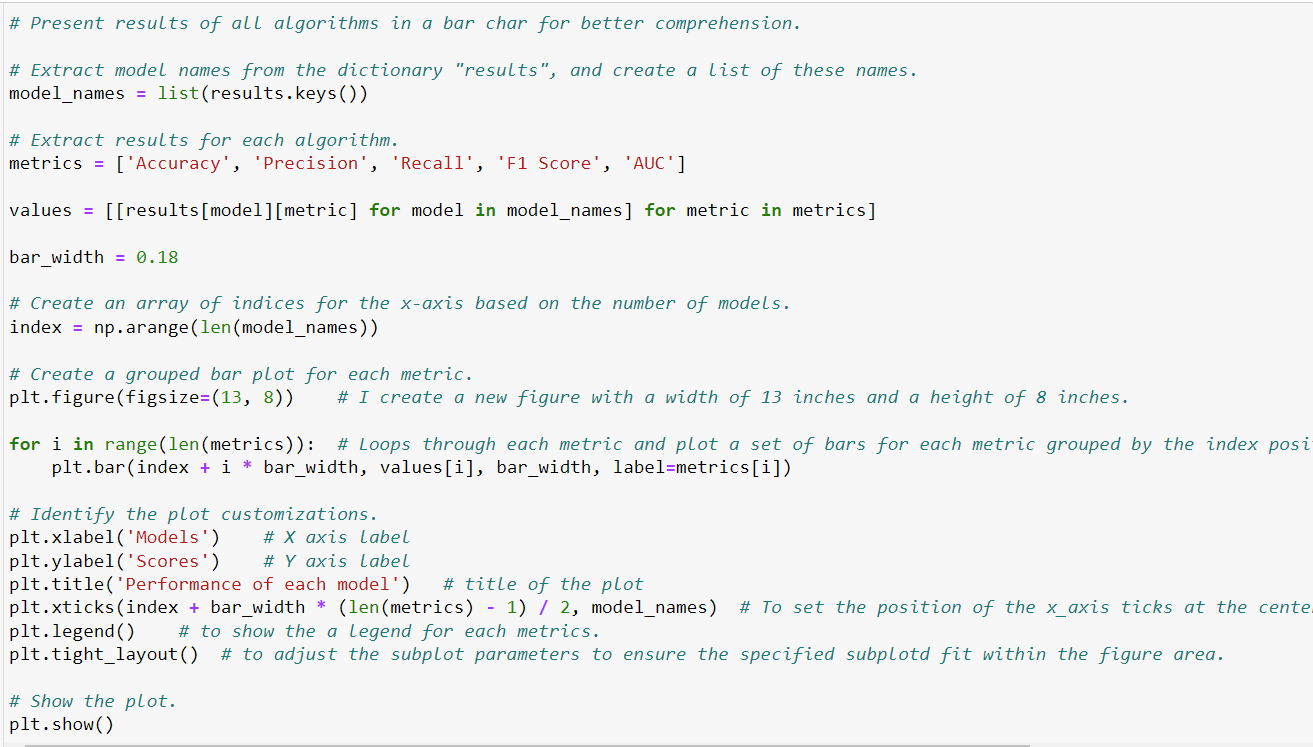
Description automatically generated

Results stored in the dictionary “Results” was converted to a dataframe to display them in a tabular format. I used a transpose operation (T) to swap the rows and columns of the dataframe so that each row corresponds to a model, and each column corresponds to a performance metric (e.g., Accuracy, Precision, Recall, F1 Score, AUC).

A screenshot of a computer

Description automatically generated

I also present the results for all algorithms in a bar chart for better comprehension. In the bar chart, the x-axis represents the models while the y-axis represents the performance score. The legend for each metric is shown in the left top corner. The bar width is 0.18 inches. The figure has a width of 13 inches and a height of 8 inches.



A colorful lines in different colors

Description automatically generated

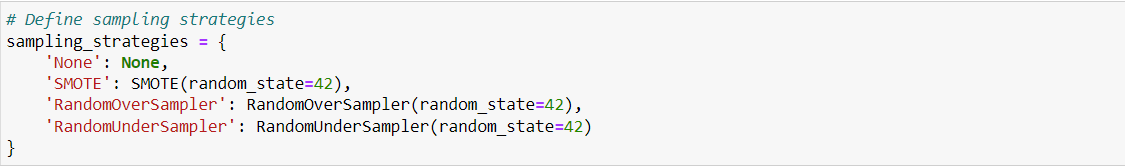
### 4. Without weakly correlated features & 4 balanced classes

In this experiment, I removed features that have a weak correlation (<0.10) with treatment response (Thyroid Function and Pathology). Then, I developed and evaluated models using the remaining features to predict 4 balanced classes of the target variable. The classes in the target variable were balanced using the 3 sampling strategies (i.e., SMOTE, RandomOverSampler, RandomUnderSampler). To this end, a dictionary of imported models was created to use it in training and testing. For all models except for NB and KNN, I set random\_state at 42 to control the randomness of the model's internal random number generator, thereby ensuring the model reproducibility each time I run the code. I also set max\_iter at 1000 for the LogR model and 5000 for the MLP model. This is because the optimization in the MLP model did not converge when using max\_iter less than 5000. For the MLP model, I set hidden\_layer\_sizes at 100, which refers to the number of neurons in each hidden layer of the MLP. In our model, there is one hidden layer with 100 neurons.

A computer code with many numbers

Description automatically generated with medium confidence

I created a dictionary named sampling\_strategies to store the imported sampling strategies, each associated with a key. The key “None” is associated with the value “None”, which suggests that no resampling is applied to the data. The key “SMOTE” is associated with the value “SMOTE”, which suggests that SMOTE is applied to the data for oversampling the minority class by generating synthetic samples. The key “RandomOverSampler” is associated with the value “RandomOverSampler”, which suggests that RandomOverSampler is applied to data to randomly duplicate samples from the minority class to balance the class distribution. The key “RandomUnderSampler” is associated with the value “RandomUnderSampler”, which suggests that RandomUnderSampler is applied to data to randomly remove samples from the majority class to balance the class distribution. The parameter “random\_state” in the sampling strategies was set at 42 to control the randomness of the model's internal random number generator, thereby ensuring the model reproducibility each time I run the code.

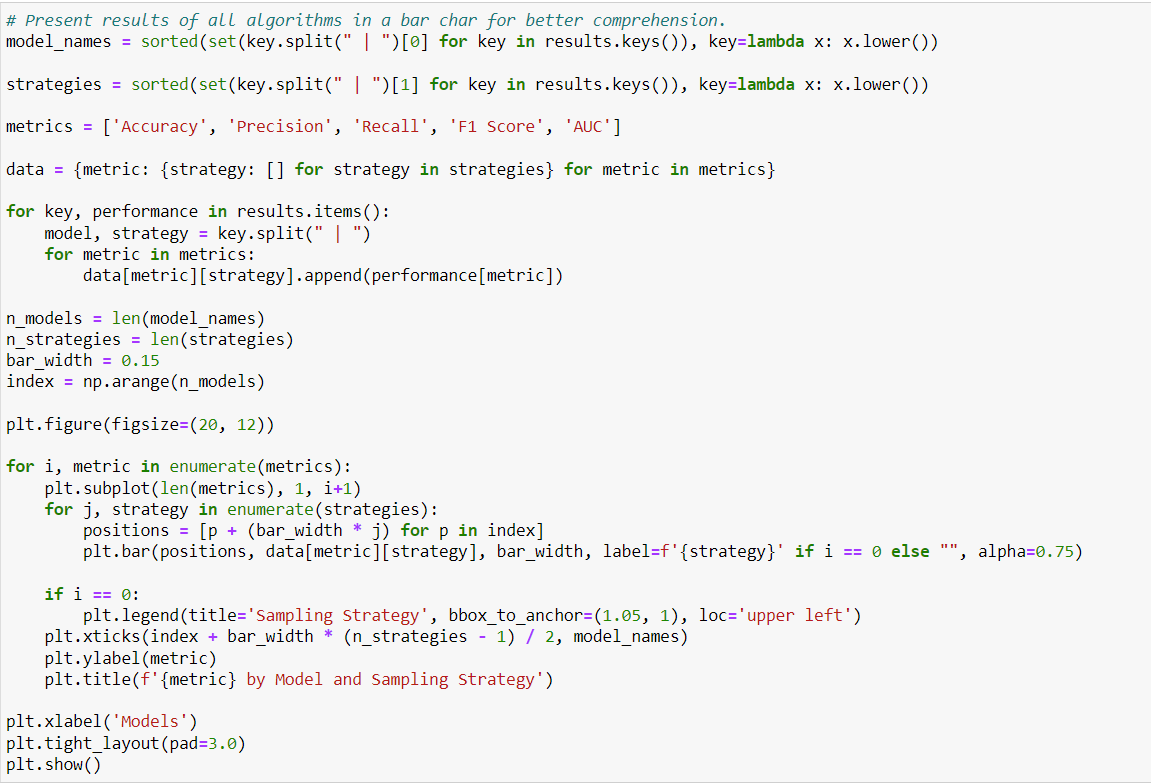


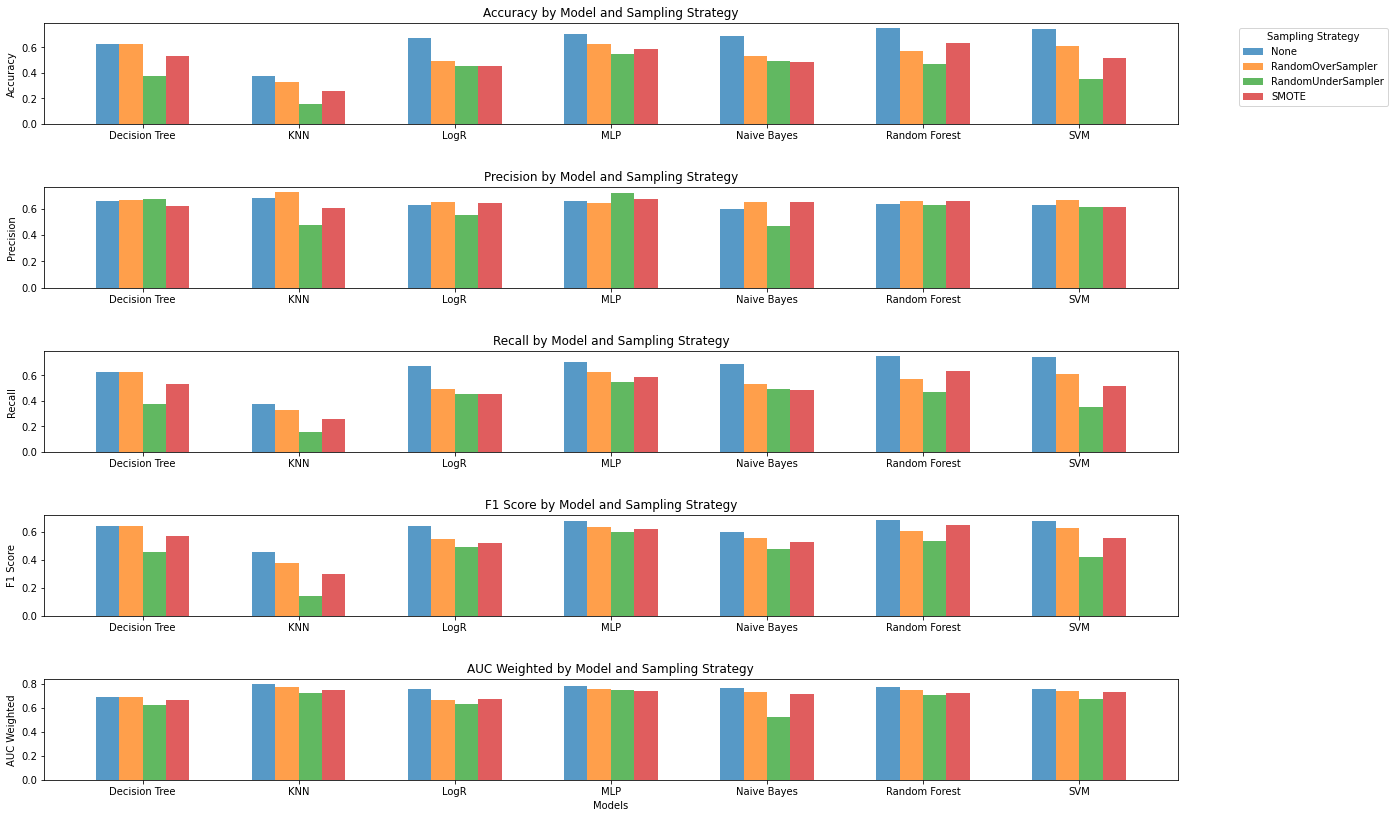
I created an empty dictionary named results that will be used to store the results for each model to present them in a table. A nested loop was created to iterate over each combination of sampling strategy and AI model. The variables strategy\_name and sampler are assigned the key-value pairs from the sampling\_strategies dictionary, where the key is a string representing the name of the sampling strategy, and the value is the corresponding sampler (e.g., SMOTE, RandomOverSampler). Similarly, model\_name and model are assigned the key-value pairs from the models dictionary, where the key is a string representing the name of the AI model, and the value is the actual model object. I also created a pipeline for the current combination of sampling strategy and model. If a sampling strategy is specified (i.e., sampler is not None), the pipeline includes a sampling step followed by a modeling step. If no sampling strategy is specified (i.e., sampler is None), the pipeline only includes the modeling step. The pipeline was fitted to the training data (X\_train and y\_train) and then the fitted pipeline was used to make predictions on the test data (X\_test). The results of these predictions (y\_pred) were used for evaluation. For precision, recall, f1, and AUC scores, the average='weighted' parameter is used to compute metrics for each label and then average them based on the number of true instances for each label. This is attributed to the fact this is a multi-class classification problem. For precision, recall, and f1 score, I set the parameter “zero\_division” at 0 to assign a score of 0 when there is a division by zero (i.e., when precision, recall, and/or f1 score is undefined because there are no true positive predictions). The results for the current model and sampling strategy combination were rounded to the nearest 3 decimal places and stored in the results dictionary.

A screenshot of a computer program

Description automatically generated

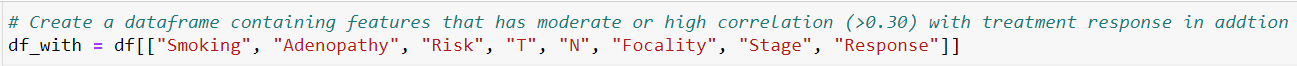
Grouped bar charts were created using Matplotlib [23] to visually compare the performance metrics (e.g., accuracy, precision, recall, F1 score, AUC) of different AI models under various sampling strategies. To do so, unique model names and sampling strategies were extracted from the keys of the results dictionary. The split (" | ") operation is used to separate the model name and strategy in each key. The set ensures uniqueness, and sorted organizes them alphabetically (case-insensitive). A dictionary called “data” was created to store performance metrics for each strategy and metric. The dictionary has the structure data[metric][strategy] = [], where metric is one of the evaluation metrics, and strategy is one of the sampling strategies. A nested loop was created to populate the data dictionary with the corresponding performance metrics for each model, strategy, and metric. I identified some parameters that will be used when creating a group bar chart such as the width of the bar (0.15), number of models and strategies, and figure size (20\*12 inches). I created a nested loop to iterate over each performance metric ('Accuracy', 'Precision', 'Recall', 'F1 Score', 'AUC) using the enumerate function to obtain both the index i and the corresponding metric name. A subplot within the overall figure for each metric was created. The len (metrics), 1 specifies the number of rows and columns in the subplot grid, and i+1 is the index of the current subplot. The inner loop (1) iterates over each sampling strategy using enumerate to obtain both the index j and the strategy name, (2) calculates the x-positions for the bars within the current subplot, (3) plots a bar chart within the current subplot using the calculated positions, the performance data (data[metric][strategy]), and bar width (bar\_width). The legend indicates the sampling strategy, and the x-axis represents the different models. The chart is organized in a grid. The entire chart is then displayed using plt.show().



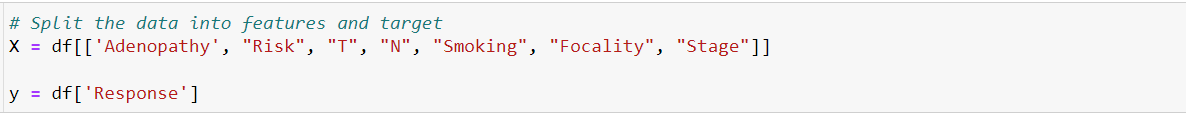


### 5. With moderately-to-strongly correlated features & 4 imbalanced classes

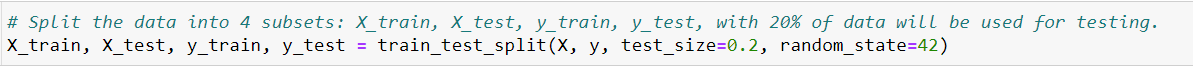
In this experiment, I removed features that have a correlation of less than 0.30 with treatment response. Then, I developed and evaluated models using the remaining features to predict 4 imbalanced classes of the target variable. To do so, I created a dataframe called df\_with containing features that have a moderate or high correlation (≥0.30) with treatment response ("Smoking", "Adenopathy", "Risk", "T", "N", "Focality", "Stage") in addition to the target variable (“Response”).



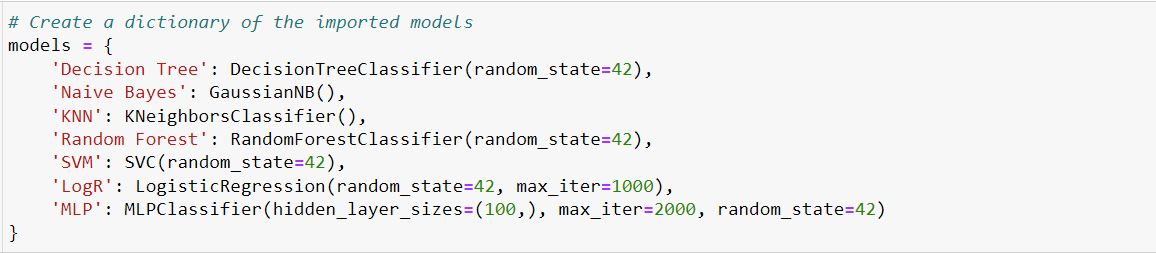
I created 2 dataframes; one (X) contains only the features and the other (y) contains the target variable (treatment response).



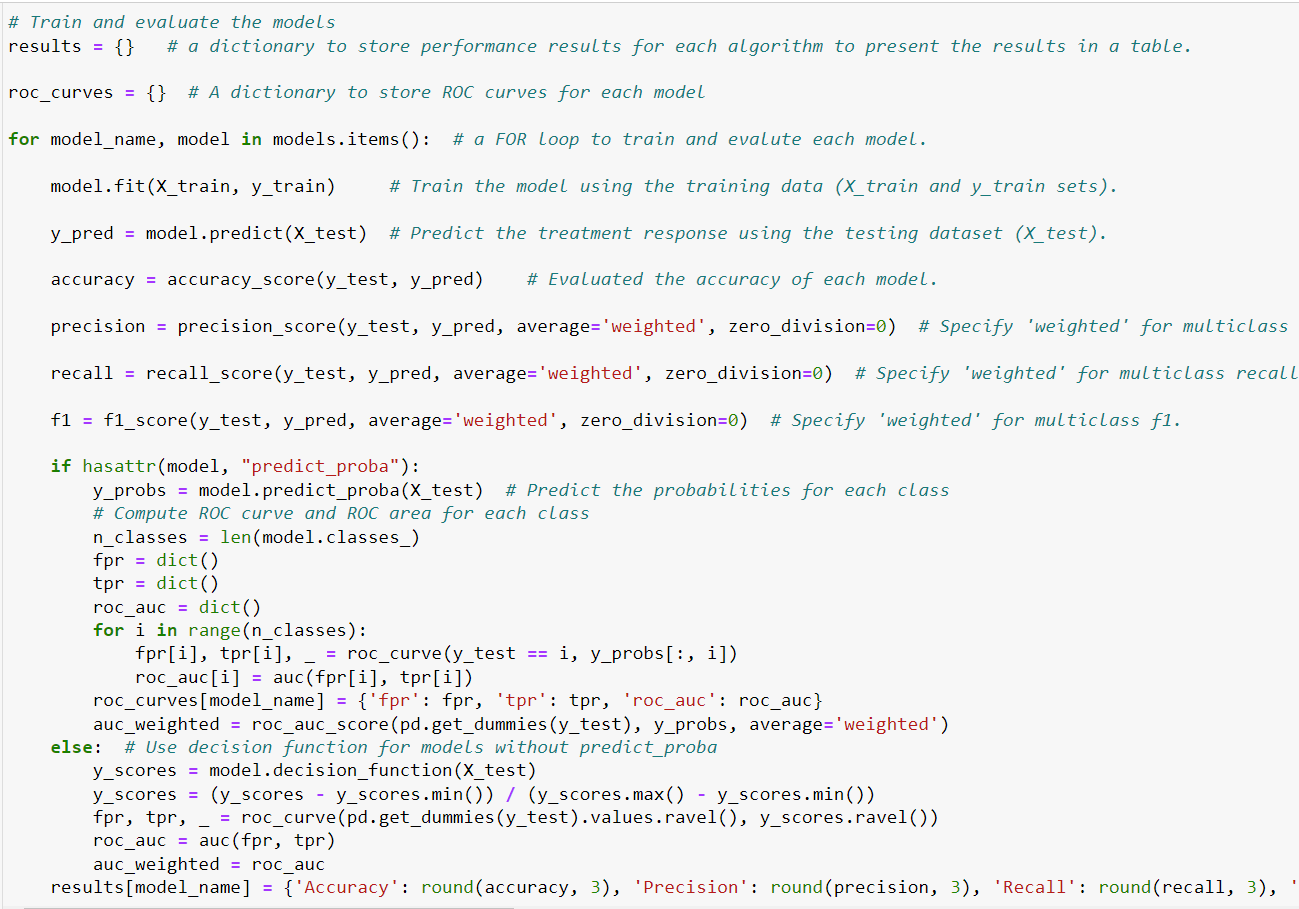
Then, the train\_test\_split function from the Sklearn library [25] was used to split the data into 4 subsets: X\_train, X\_test, y\_train, and y\_test. The value of parameter test\_size was set to 0.2, which means that 80% of the data will be used for training while the remaining data (20%) will be used for testing. This value was selected because the dataset is small. The value of the parameter “random\_state” was set to 42 (which is arbitrary) to ensure that the dataset is split in the same way every time I run the code. This helps in obtaining consistent and reproducible results.



A dictionary of imported models was created to use it in training and testing. For all models except for NB and KNN, I set random\_state at 42 to control the randomness of the model's internal random number generator, thereby ensuring the model reproducibility each time I run the code. I also set max\_iter at 1000 for the LogR model and 2000 for the MLP model. This is because the optimization in the MLP model did not converge when using max\_iter less than 2000. For the MLP model, I set hidden\_layer\_sizes at 100, which refers to the number of neurons in each hidden layer of the MLP. In our model, there is one hidden layer with 100 neurons.



I created 2 dictionaries to help present the results. Specifically, the dictionary “results” was created to store performance results for each algorithm to use in presenting the results in a table. The dictionary “roc\_curves” was created to store ROC curves for each model to help calculate AUC. After that, I created a “for” loop to train (fit) each model using the “fit” function and the training data (X\_train and y\_train sets). Then, the trained models predicted the treatment response using the “predict” function with an unseen testing dataset (X\_test). Lastly, the above-mentioned metrics were calculated to evaluate the performance of each model. For precision, recall, f1, and AUC scores, the average='weighted' parameter is used to compute metrics for each label and then average them based on the number of true instances for each label. This is attributed to the fact this is a multi-class classification problem. For precision, recall, and f1 score, I set the parameter “zero\_division” at 0 to assign a score of 0 when there is a division by zero (i.e., when precision, recall, and/or f1 score is undefined because there are no true positive predictions). The results were rounded to the nearest 3 decimal places and stored in the dictionary “results”.

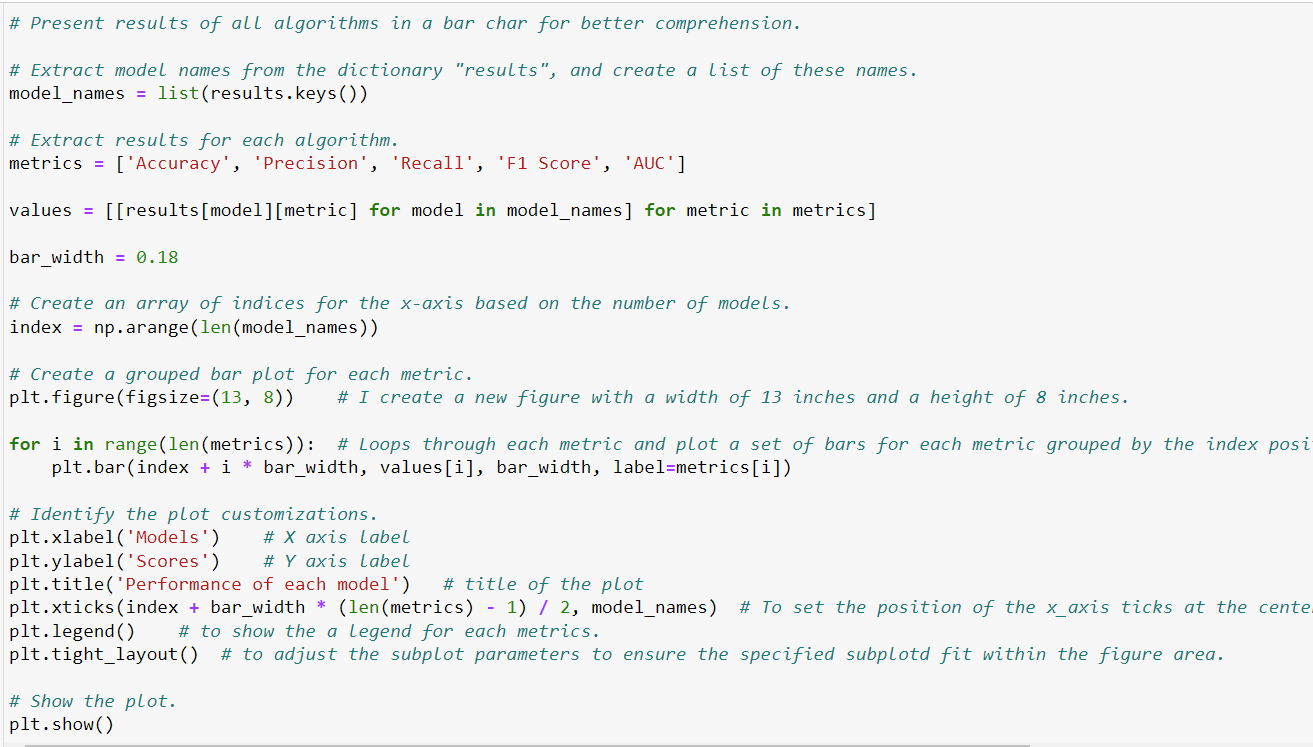


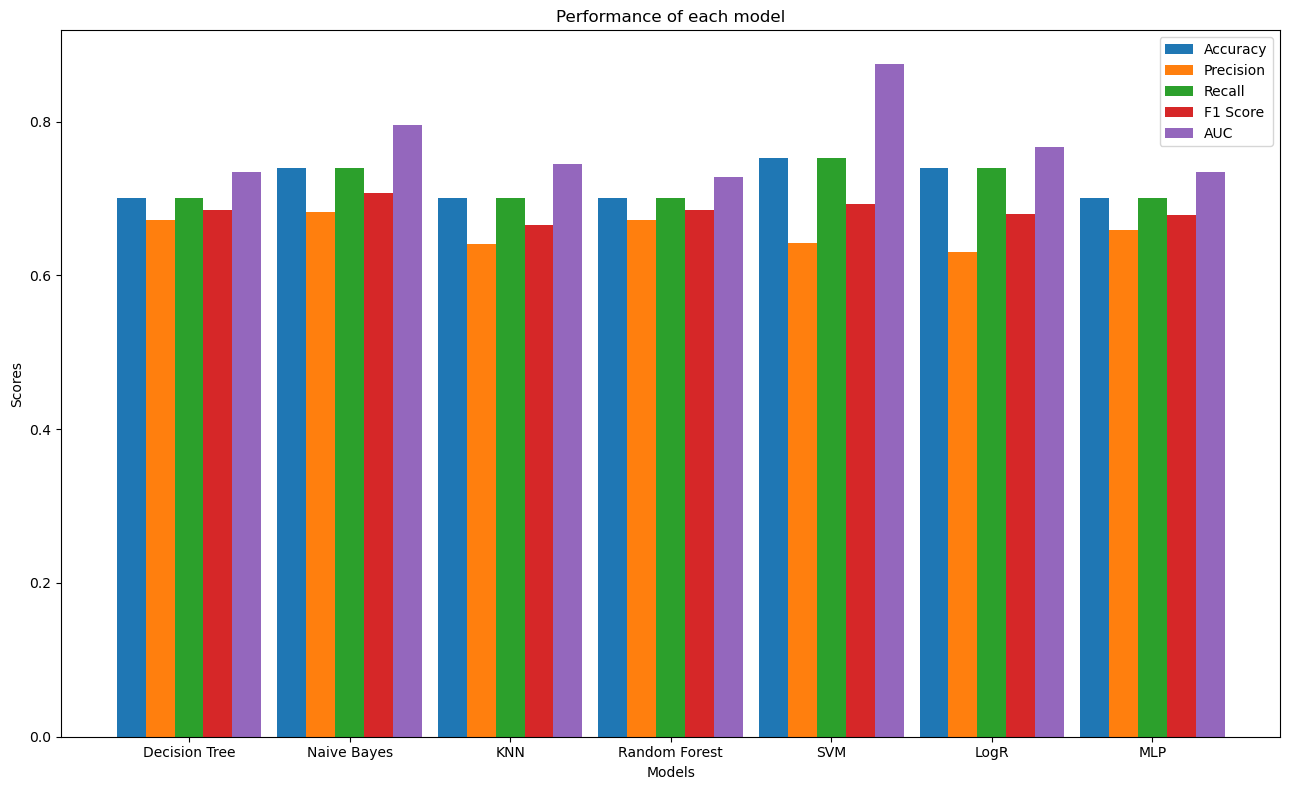
Results stored in the dictionary “Results” was converted to a dataframe to display them in a tabular format. I used a transpose operation (T) to swap the rows and columns of the dataframe so that each row corresponds to a model, and each column corresponds to a performance metric (e.g., Accuracy, Precision, Recall, F1 Score, AUC).

A screenshot of a computer screen

Description automatically generated

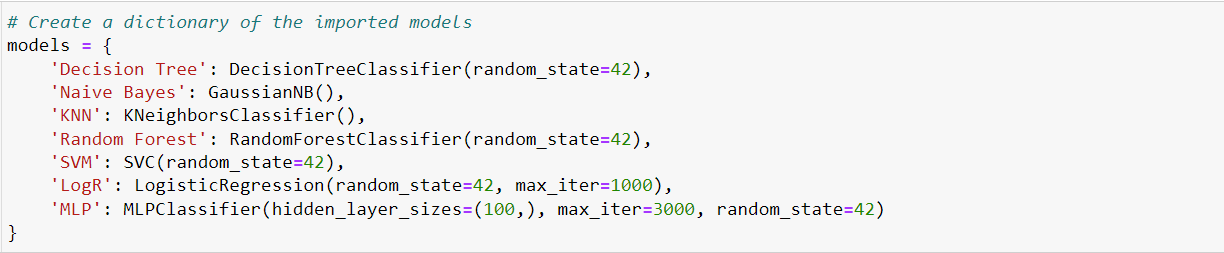
I also present the results for all algorithms in a bar chart for better comprehension. In the bar chart, the x-axis represents the models while the y-axis represents the performance score. The legend for each metric is shown in the right top corner. The bar width is 0.18 inches. The figure has a width of 13 inches and a height of 8 inches.





### 6. With moderately-to-strongly correlated features & 4 imbalanced classes

In this experiment, I removed features that have a correlation of less than 0.30 with treatment response. Then, I developed and evaluated models using the remaining features to predict 4 balanced classes of the target variable. The classes in the target variable were balanced using the 3 sampling strategies (i.e., SMOTE, RandomOverSampler, RandomUnderSampler). To this end, a dictionary of imported models was created to use it in training and testing. For all models except for NB and KNN, I set random\_state at 42 to control the randomness of the model's internal random number generator, thereby ensuring the model reproducibility each time I run the code. I also set max\_iter at 1000 for the LogR model and 3000 for the MLP model. This is because the optimization in the MLP model did not converge when using max\_iter less than 3000. For the MLP model, I set hidden\_layer\_sizes at 100, which refers to the number of neurons in each hidden layer of the MLP. In our model, there is one hidden layer with 100 neurons.



I created a dictionary named sampling\_strategies to store the imported sampling strategies, each associated with a key. The key “None” is associated with the value “None”, which suggests that no resampling is applied to the data. The key “SMOTE” is associated with the value “SMOTE”, which suggests that SMOTE is applied to the data for oversampling the minority class by generating synthetic samples. The key “RandomOverSampler” is associated with the value “RandomOverSampler”, which suggests that RandomOverSampler is applied to data to randomly duplicate samples from the minority class to balance the class distribution. The key “RandomUnderSampler” is associated with the value “RandomUnderSampler”, which suggests that RandomUnderSampler is applied to data to randomly remove samples from the majority class to balance the class distribution. The parameter “random\_state” in the sampling strategies was set at 42 to control the randomness of the model's internal random number generator, thereby ensuring the model reproducibility each time I run the code.

A close-up of a computer code

Description automatically generated

I created an empty dictionary named results that will be used to store the results for each model to present them in a table. A nested loop was created to iterate over each combination of sampling strategy and AI model. The variables strategy\_name and sampler are assigned the key-value pairs from the sampling\_strategies dictionary, where the key is a string representing the name of the sampling strategy, and the value is the corresponding sampler (e.g., SMOTE, RandomOverSampler). Similarly, model\_name and model are assigned the key-value pairs from the models dictionary, where the key is a string representing the name of the AI model, and the value is the actual model object. I also created a pipeline for the current combination of sampling strategy and model. If a sampling strategy is specified (i.e., sampler is not None), the pipeline includes a sampling step followed by a modeling step. If no sampling strategy is specified (i.e., sampler is None), the pipeline only includes the modeling step. The pipeline was fitted to the training data (X\_train and y\_train) and then the fitted pipeline was used to make predictions on the test data (X\_test). The results of these predictions (y\_pred) were used for evaluation. For precision, recall, f1, and AUC scores, the average='weighted' parameter is used to compute metrics for each label and then average them based on the number of true instances for each label. This is attributed to the fact this is a multi-class classification problem. For precision, recall, and f1 score, I set the parameter “zero\_division” at 0 to assign a score of 0 when there is a division by zero (i.e., when precision, recall, and/or f1 score is undefined because there are no true positive predictions). The results for the current model and sampling strategy combination were rounded to the nearest 3 decimal places and stored in the results dictionary.

A screenshot of a computer program

Description automatically generated

Grouped bar charts were created using Matplotlib [23] to visually compare the performance metrics (e.g., accuracy, precision, recall, F1 score, AUC) of different AI models under various sampling strategies. To do so, unique model names and sampling strategies were extracted from the keys of the results dictionary. The split (" | ") operation is used to separate the model name and strategy in each key. The set ensures uniqueness, and sorted organizes them alphabetically (case-insensitive). A dictionary called “data” was created to store performance metrics for each strategy and metric. The dictionary has the structure data[metric][strategy] = [], where metric is one of the evaluation metrics, and strategy is one of the sampling strategies. A nested loop was created to populate the data dictionary with the corresponding performance metrics for each model, strategy, and metric. I identified some parameters that will be used when creating a group bar chart such as the width of the bar (0.15), the number of models and strategies, and figure size (20\*12 inches). I created a nested loop to iterate over each performance metric ('Accuracy', 'Precision', 'Recall', 'F1 Score', 'AUC) using the enumerate function to obtain both the index i and the corresponding metric name. A subplot within the overall figure for each metric was created. The len (metrics), 1 specifies the number of rows and columns in the subplot grid, and i+1 is the index of the current subplot. The inner loop (1) iterates over each sampling strategy using enumerate to obtain both the index j and the strategy name, (2) calculates the x-positions for the bars within the current subplot, (3) plots a bar chart within the current subplot using the calculated positions, the performance data (data[metric][strategy]), and bar width (bar\_width). The legend indicates the sampling strategy, and the x-axis represents the different models. The chart is organized in a grid. The entire chart is then displayed using plt.show().

A screenshot of a computer program

Description automatically generated

A row of colorful books

Description automatically generated with medium confidence

### 7. All features & 2 balanced classes

In this experiment, I developed and evaluated models using all features to predict a binary outcome with balanced classes in the target variable. The classes in the target variable were balanced by grouping classes into 2 subgroups: excellent (comprising excellent class) and non-excellent (comprising biochemical incomplete, structurally incomplete, indeterminate classes). To do so, I converted the values in the 'Response' column to binary values, where 0 is transformed to 0 (Excellent), and any other value is transformed to 1 (Non-Excellent).

A screenshot of a computer

Description automatically generated

I checked whether the transformation was correct by checking the count values of each class.

A white rectangular object with a white border

Description automatically generated

The correlation between the target variable and other variables (features) was calculated to identify which features I should consider when developing the models. The results were shown using a heatmap graph. As shown in the figure below, the target variable has a very low correlation (<0.1) with the features "Thyroid Function", "Pathology", and "Hx Smoking" whereas it has a moderate-to-strong correlation (≥0.30) with the features "Adenopathy", "Focality", "Risk", "T", "N", and "Stage". So, these differences in correlations were considered when developing the models as I did in the previous experiments.

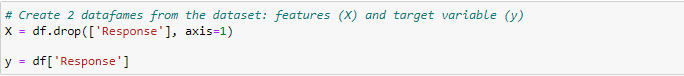
A computer code on a white background

Description automatically generated

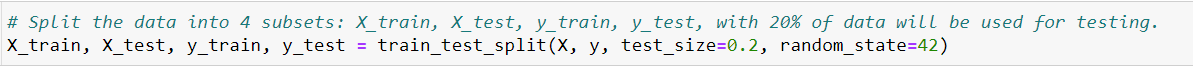
A screenshot of a computer

Description automatically generated

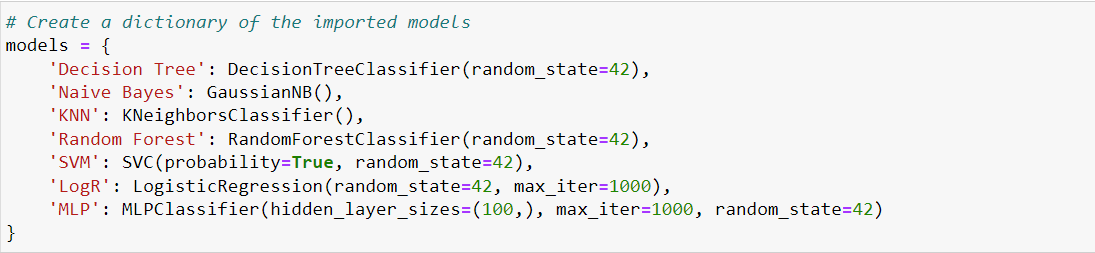
I created 2 dataframes; one (X) contains only the features and the other (y) contains the target variable (treatment response).



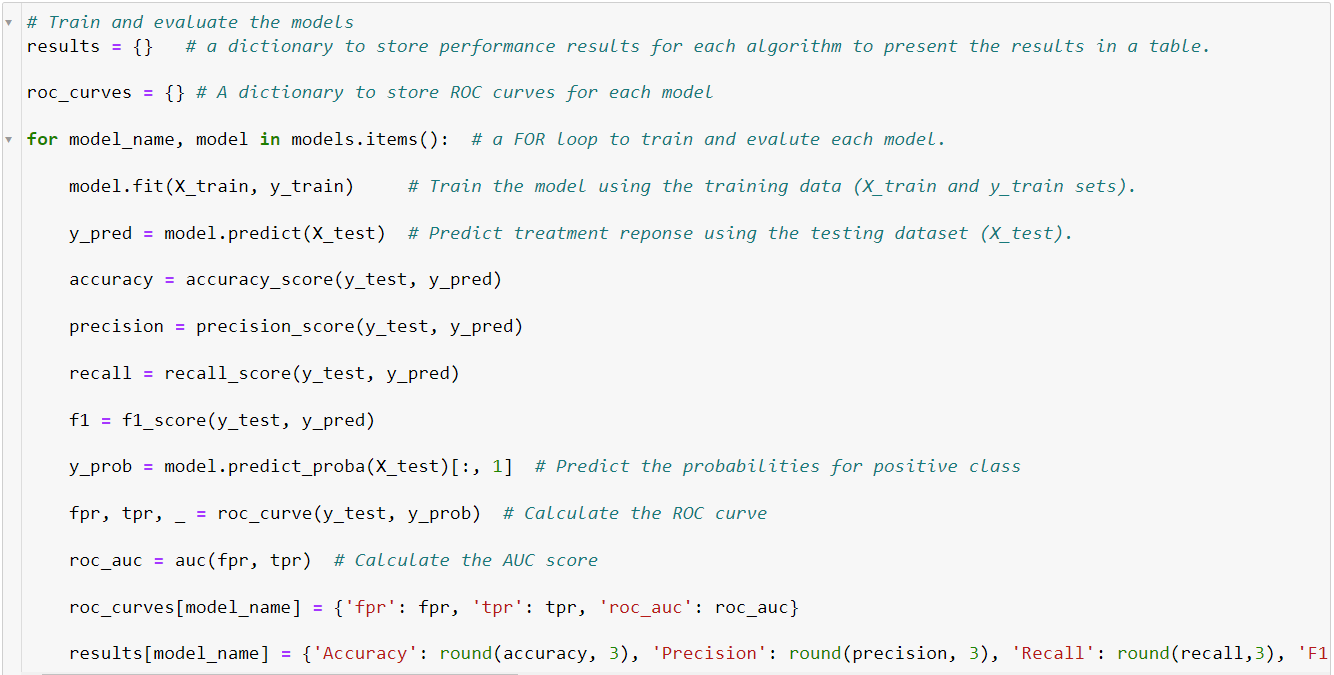
Then, the train\_test\_split function from the Sklearn library [25] was used to split the data into 4 subsets: X\_train, X\_test, y\_train, and y\_test. The value of parameter test\_size was set to 0.2, which means that 80% of the data will be used for training while the remaining data (20%) will be used for testing. This value was selected because the dataset is small. The value of the parameter “random\_state” was set to 42 (which is arbitrary) to ensure that the dataset is split in the same way every time I run the code. This helps in obtaining consistent and reproducible results.



A dictionary of imported models was created to use it in training and testing. For all models except for NB and KNN, I set random\_state at 42 to control the randomness of the model's internal random number generator, thereby ensuring the model reproducibility each time I run the code. For SVM, I set the parameter “probability” at “True” to enable the SVC classifier to provide probability estimates for class assignments in addition to the predicted class labels. For LogR and MLP models, I also set max\_iter at 1000, which refers to the maximum number of iterations the model will perform during the training process. For the MLP model, I set hidden\_layer\_sizes at 100, which refers to the number of neurons in each hidden layer of the MLP. In our model, there is one hidden layer with 100 neurons.



I created 2 dictionaries to help present the results. Specifically, the dictionary “results” was created to store performance results for each algorithm to use in presenting the results in a table. The dictionary “roc\_curves” was created to store ROC curves for each model to help calculate AUC and present ROC curves in a graph. After that, I created a “for” loop to train (fit) each model using the “fit” function and the training data (X\_train and y\_train sets). Then, the trained models predicted the treatment response using the “predict” function with an unseen testing dataset (X\_test). Lastly, the above-mentioned metrics were calculated to evaluate the performance of each model. The results were rounded to the nearest 3 decimal places and stored in the dictionary “results”.

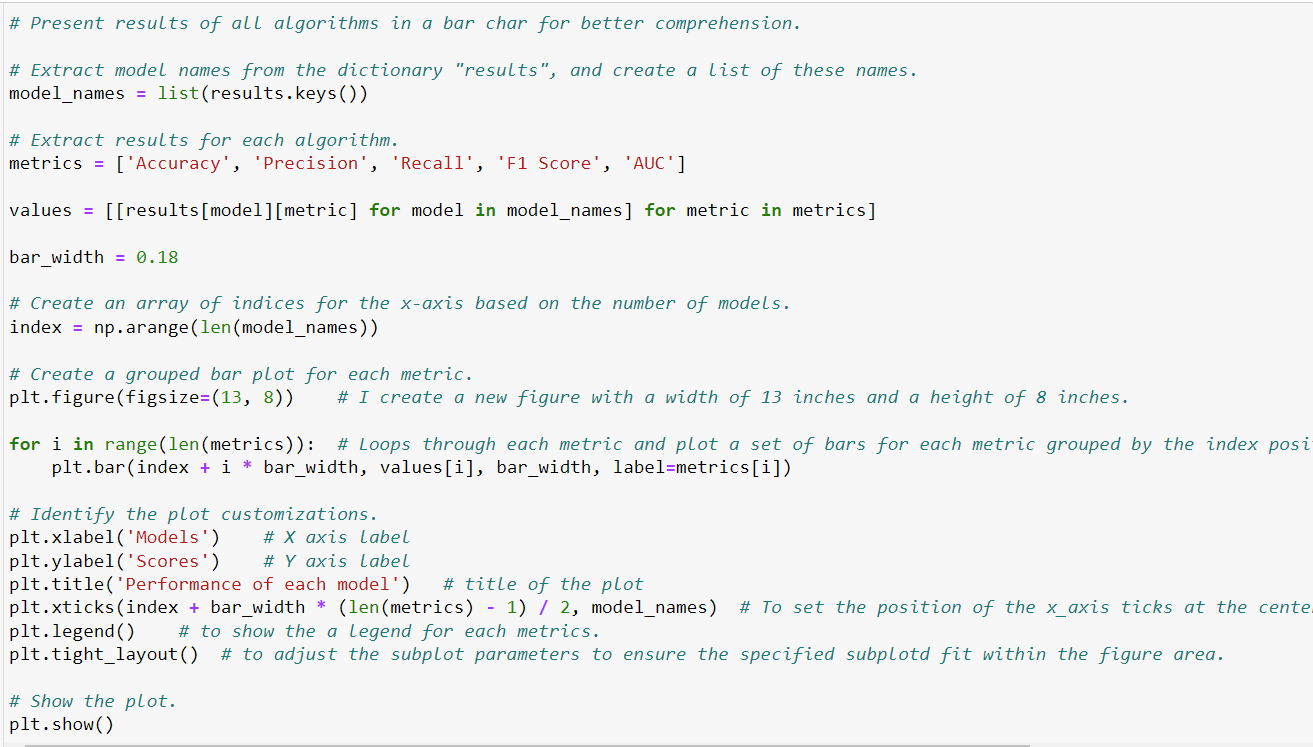


Results stored in the dictionary “Results” was converted to a dataframe to display them in a tabular format. I used a transpose operation (T) to swap the rows and columns of the dataframe so that each row corresponds to a model, and each column corresponds to a performance metric (e.g., Accuracy, Precision, Recall, F1 Score, AUC).

A screenshot of a computer screen

Description automatically generated

I also present the results for all algorithms in a bar chart for better comprehension. In the bar chart, the x-axis represents the models while the y-axis represents the performance score. The legend for each metric is shown in the right top corner. The bar width is 0.18 inches. The figure has a width of 13 inches and a height of 8 inches. Labels of the x-axis and y-axis and the title of the figure were identified.



A graph of different colored vertical lines

Description automatically generated with medium confidence

Further, I presented ROC curve for each model in a line plot with a length of 8 inches and a width of 14 inches. the false positive rate (fpr) was presented on the x-axis and the true positive rate (tpr) was presented on the y-axis. A dashed black line was created to represent the ROC curve for a random classifier. This line serves as a reference for the models' performance. Labels of the x-axis and y-axis and the title of the figure were identified. The legend of the plot indicates the model names and their corresponding AUC values.

A computer screen shot of a code

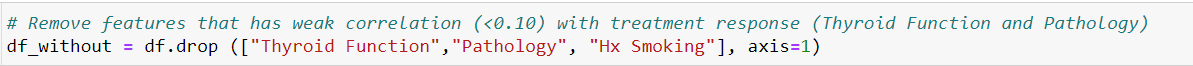
Description automatically generated

A graph of a graph

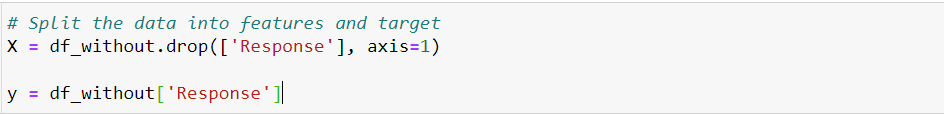
Description automatically generated with medium confidence

### 8. Without weakly correlated features & 2 balanced classes

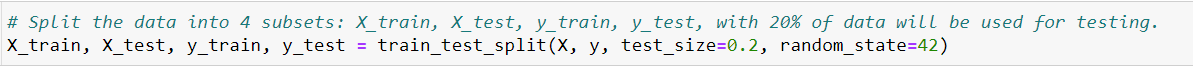
In this experiment, I removed features that have a weak correlation (<0.10) with treatment response (Thyroid Function, Pathology, and Hx Smoking). Then, I developed and evaluated models using the remaining features to predict 2 balanced classes of the target variable.



I created 2 dataframes; one (X) contains only the features and the other (y) contains the target variable (treatment response).



Then, the train\_test\_split function from the Sklearn library [25] was used to split the data into 4 subsets: X\_train, X\_test, y\_train, and y\_test. The value of parameter test\_size was set to 0.2, which means that 80% of the data will be used for training while the remaining data (20%) will be used for testing. This value was selected because the dataset is small. The value of the parameter “random\_state” was set to 42 (which is arbitrary) to ensure that the dataset is split in the same way every time I run the code. This helps in obtaining consistent and reproducible results.



A dictionary of imported models was created to use it in training and testing. For all models except for NB and KNN, I set random\_state at 42 to control the randomness of the model's internal random number generator, thereby ensuring the model reproducibility each time I run the code. For SVM, I set the parameter “probability” at “True” to enable the SVC classifier to provide probability estimates for class assignments in addition to the predicted class labels. For LogR and MLP models, I also set max\_iter at 1000, which refers to the maximum number of iterations the model will perform during the training process. For the MLP model, I set hidden\_layer\_sizes at 100, which refers to the number of neurons in each hidden layer of the MLP. In our model, there is one hidden layer with 100 neurons.

A computer code with text

Description automatically generated with medium confidence

I created 2 dictionaries to help present the results. Specifically, the dictionary “results” was created to store performance results for each algorithm to use in presenting the results in a table. The dictionary “roc\_curves” was created to store ROC curves for each model to help calculate AUC and present ROC curves in a graph. After that, I created a “for” loop to train (fit) each model using the “fit” function and the training data (X\_train and y\_train sets). Then, the trained models predicted the treatment response using the “predict” function with an unseen testing dataset (X\_test). Lastly, the above-mentioned metrics were calculated to evaluate the performance of each model. The results were rounded to the nearest 3 decimal places and stored in the dictionary “results”.

A screenshot of a computer program

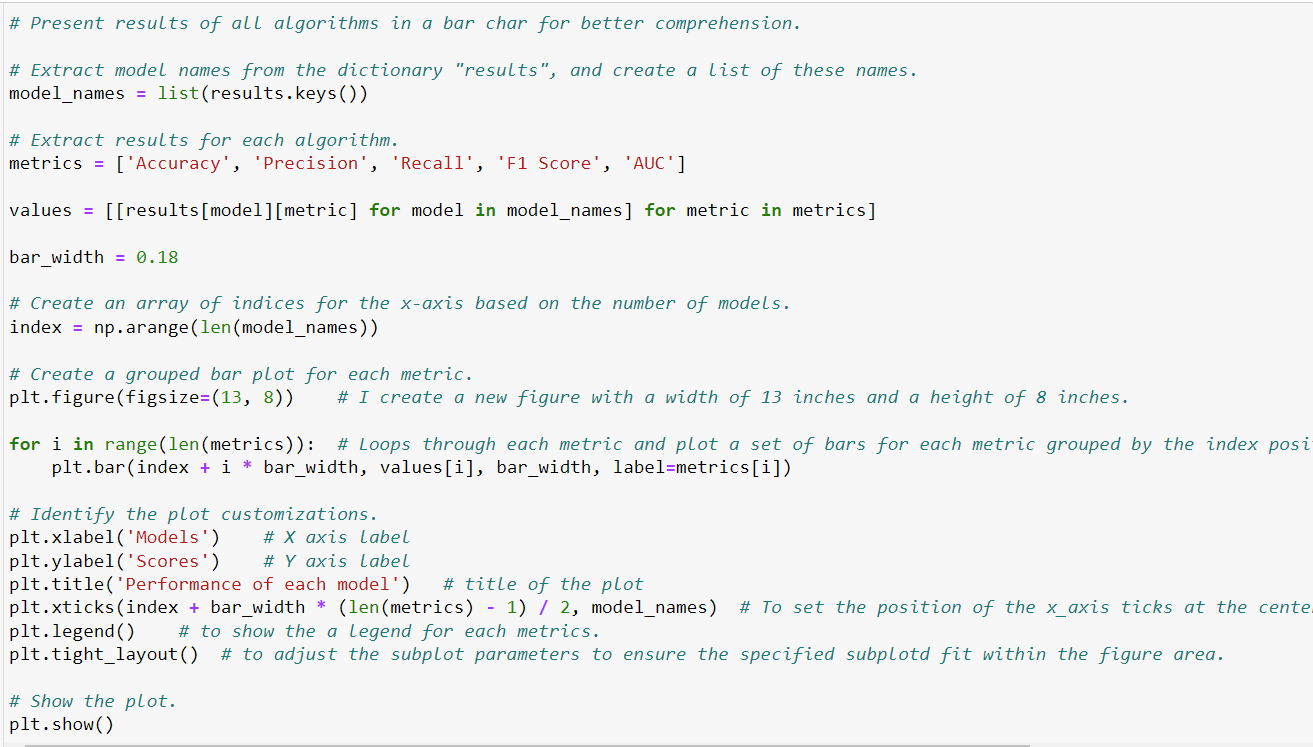
Description automatically generated

Results stored in the dictionary “Results” was converted to a dataframe to display them in a tabular format. I used a transpose operation (T) to swap the rows and columns of the dataframe so that each row corresponds to a model, and each column corresponds to a performance metric (e.g., Accuracy, Precision, Recall, F1 Score, AUC).

A screenshot of a computer

Description automatically generated

I also present the results for all algorithms in a bar chart for better comprehension. In the bar chart, the x-axis represents the models while the y-axis represents the performance score. The legend for each metric is shown in the left top corner. The bar width is 0.18 inches. The figure has a width of 13 inches and a height of 8 inches. Labels of the x-axis and y-axis and the title of the figure were identified.



A graph of different colored vertical lines

Description automatically generated with medium confidence

Further, I presented ROC curve for each model in a line plot with a length of 8 inches and a width of 14 inches. the false positive rate (fpr) was presented on the x-axis and the true positive rate (tpr) was presented on the y-axis. A dashed black line was created to represent the ROC curve for a random classifier. This line serves as a reference for the models' performance. Labels of the x-axis and y-axis and the title of the figure were identified. The legend of the plot indicates the model names and their corresponding AUC values.

A computer screen shot of a code

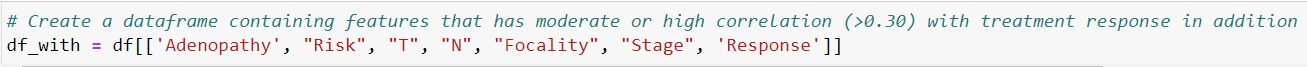
Description automatically generated

A graph of a number of different colored lines

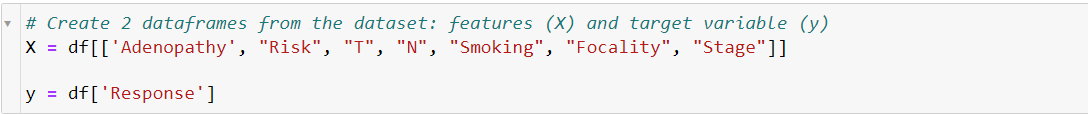
Description automatically generated

### 9. Without moderately-to-strongly correlated features & 2 balanced classes

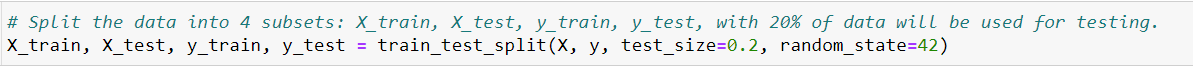
In this experiment, I removed features that have a weak correlation (<0.30) with treatment response. Then, I developed and evaluated models using the remaining features to predict 2 balanced classes of the target variable. To do so, I created a dataframe called df\_with containing features that have a moderate or high correlation (≥0.30) with treatment response ("Smoking", "Adenopathy", "Risk", "T", "N", "Focality", "Stage") in addition to the target variable (“Response”).



I created 2 dataframes; one (X) contains only the features and the other (y) contains the target variable (treatment response).



Then, the train\_test\_split function from the Sklearn library [25] was used to split the data into 4 subsets: X\_train, X\_test, y\_train, and y\_test. The value of parameter test\_size was set to 0.2, which means that 80% of the data will be used for training while the remaining data (20%) will be used for testing. This value was selected because the dataset is small. The value of the parameter “random\_state” was set to 42 (which is arbitrary) to ensure that the dataset is split in the same way every time I run the code. This helps in obtaining consistent and reproducible results.



A dictionary of imported models was created to use it in training and testing. For all models except for NB and KNN, I set random\_state at 42 to control the randomness of the model's internal random number generator, thereby ensuring the model reproducibility each time I run the code. For SVM, I set the parameter “probability” at “True” to enable the SVC classifier to provide probability estimates for class assignments in addition to the predicted class labels. I also set max\_iter at 1000 for the LogR model and 2000 for the MLP model. This is because the optimization in the MLP model did not converge when using max\_iter less than 2000. For the MLP model, I set hidden\_layer\_sizes at 100, which refers to the number of neurons in each hidden layer of the MLP. In our model, there is one hidden layer with 100 neurons.

A computer code with many numbers

Description automatically generated with medium confidence

I created 2 dictionaries to help present the results. Specifically, the dictionary “results” was created to store performance results for each algorithm to use in presenting the results in a table. The dictionary “roc\_curves” was created to store ROC curves for each model to help calculate AUC and present ROC curves in a graph. After that, I created a “for” loop to train (fit) each model using the “fit” function and the training data (X\_train and y\_train sets). Then, the trained models predicted the treatment response using the “predict” function with an unseen testing dataset (X\_test). Lastly, the above-mentioned metrics were calculated to evaluate the performance of each model. The results were rounded to the nearest 3 decimal places and stored in the dictionary “results”.

A screenshot of a computer program

Description automatically generated

Results stored in the dictionary “Results” was converted to a dataframe to display them in a tabular format. I used a transpose operation (T) to swap the rows and columns of the dataframe so that each row corresponds to a model, and each column corresponds to a performance metric (e.g., Accuracy, Precision, Recall, F1 Score, AUC).

A screenshot of a computer

Description automatically generated

I also present the results for all algorithms in a bar chart for better comprehension. In the bar chart, the x-axis represents the models while the y-axis represents the performance score. The legend for each metric is shown in the left top corner. The bar width is 0.18 inches. The figure has a width of 13 inches and a height of 8 inches. Labels of the x-axis and y-axis and the title of the figure were identified.

A screenshot of a computer code

Description automatically generated

A colorful lines in different colors

Description automatically generated

Further, I presented ROC curve for each model in a line plot with a length of 8 inches and a width of 14 inches. the false positive rate (fpr) was presented on the x-axis and the true positive rate (tpr) was presented on the y-axis. A dashed black line was created to represent the ROC curve for a random classifier. This line serves as a reference for the models' performance. Labels of the x-axis and y-axis and the title of the figure were identified. The legend of the plot indicates the model names and their corresponding AUC values.

A computer screen shot of a code

Description automatically generated

A graph of a graph showing the results of a model

Description automatically generated with medium confidence

# Results and Discussion

## Summary of findings

Our experiments showed that AI has a good performance in predicting the response of thyroid cancer treatment, but it is not optimal. Specifically, as shown in Table 2, models in all experiments achieved an average accuracy of 72.2% (39% to 85.7%), average precision of 76.2% (59.5% to 96.3%), average recall of 68% (39% to 79.5%), average F1 score of 70.6% (46.5% to 85%), and average AUC of 80.5% (67% to 91.6%).

Table 2: Summary of overall performance of all models in all experiments

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **Accuracy** | **Precision** | **Recall** | **F1 Score** | **AUC** |
| **Mean** | 0.722 | 0.762 | 0.680 | 0.706 | 0.805 |
| **SD** | 0.083 | 0.123 | 0.088 | 0.082 | 0.064 |
| **Max** | 0.857 | 0.963 | 0.795 | 0.85 | 0.916 |
| **Min** | 0.39 | 0.595 | 0.39 | 0.465 | 0.67 |

In experiments 2, 4, and 6, the group bar charts revealed that the models exhibited superior performance in most results when no sampling strategies were used. As outlined in Table 3, AI models generally exhibited superior performance in predicting a binary treatment response (Excellent and Non-excellent) compared to predicting multi-class treatment responses (Excellent, biochemical incomplete, structurally incomplete, indeterminate classes) in terms of accuracy (76.8% vs.67.6%), precision (87.3% vs. 65.1%), recall (68.3% vs. 67.6%), F1 score (76.1% vs. 65%), and AUC (84.3% vs. 76.8%).

Table 3: Summary of overall performance of all models based on number of classes in the target variable.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **Multi-class prediction** | | | | |
|  | **Accuracy** | **Precision** | **Recall** | **F1 Score** | **AUC** |
| **Mean** | 0.676 | 0.651 | 0.676 | 0.650 | 0.768 |
| **SD** | 0.086 | 0.036 | 0.086 | 0.057 | 0.052 |
| **Max** | 0.753 | 0.728 | 0.753 | 0.707 | 0.875 |
| **Min** | 0.39 | 0.595 | 0.39 | 0.465 | 0.67 |
|  | **Binary-class prediction** | | | | |
| **Mean** | 0.768 | 0.873 | 0.683 | 0.761 | 0.843 |
| **SD** | 0.049 | 0.062 | 0.092 | 0.062 | 0.051 |
| **Max** | 0.857 | 0.963 | 0.795 | 0.85 | 0.916 |
| **Min** | 0.662 | 0.765 | 0.487 | 0.629 | 0.719 |

As presented in Table 4, in general, the performance of AI models in predicting a multi-class treatment response using moderately-to-strongly correlated features was higher than using all features without weakly correlated features or all features in terms of accuracy (72% vs. 64.7% vs. 66.2%), recall (72% vs. 64.7% vs. 66.2%), and F1 score (68.5% vs. 62.2% vs. 64.3%). However, their performance was comparable in terms of precision (65.7% vs. 64.1% vs. 65.6%) and AUC (76.8% vs. 76.5% vs. 77%).

Table 4: Summary of overall performance of all models in predicting multi-class treatment response based on different sets of features.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **All features** | | | | |
|  | **Accuracy** | **Precision** | **Recall** | **F1 Score** | **AUC** |
| **Mean** | 0.662 | 0.656 | 0.662 | 0.643 | 0.770 |
| **SD** | 0.074 | 0.043 | 0.074 | 0.044 | 0.052 |
| **Max** | 0.727 | 0.728 | 0.727 | 0.694 | 0.859 |
| **Min** | 0.519 | 0.595 | 0.519 | 0.578 | 0.687 |
|  | **Without weakly correlated features** | | | | |
| **Mean** | 0.647 | 0.641 | 0.647 | 0.622 | 0.765 |
| **SD** | 0.123 | 0.043 | 0.123 | 0.080 | 0.059 |
| **Max** | 0.753 | 0.707 | 0.753 | 0.705 | 0.858 |
| **Min** | 0.39 | 0.595 | 0.39 | 0.465 | 0.67 |
|  | **With moderately-to-strongly correlated features** | | | | |
| **Mean** | 0.720 | 0.657 | 0.720 | 0.685 | 0.768 |
| **SD** | 0.024 | 0.020 | 0.024 | 0.013 | 0.053 |
| **Max** | 0.753 | 0.683 | 0.753 | 0.707 | 0.875 |
| **Min** | 0.701 | 0.63 | 0.701 | 0.666 | 0.728 |

As exhibited in Table 5, AI models generally showed superior performance in predicting a binary treatment response when utilizing moderately-to-strongly correlated features compared to scenarios involving all features, either with or without weakly correlated features. This superiority was evident in terms of accuracy (78.6% compared to 77.7% and 74.2%), recall (73.4% compared to 65.9% and 65.6%), and F1 score (79.7% compared to 74.6% and 74%). However, their performance showed similarity in precision (87.5% compared to 87.8% and 86.6%) and AUC (84.3% compared to 83.7% and 84.8%).

Table 5: Summary of overall performance of all models in predicting binary treatment response based on different sets of features.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **All features** | | | | |
|  | **Accuracy** | **Precision** | **Recall** | **F1 Score** | **AUC** |
| **Mean** | 0.742 | 0.866 | 0.656 | 0.740 | 0.848 |
| **SD** | 0.054 | 0.068 | 0.108 | 0.070 | 0.066 |
| **Max** | 0.805 | 0.963 | 0.773 | 0.81 | 0.916 |
| **Min** | 0.662 | 0.78 | 0.5 | 0.629 | 0.731 |
|  | **Without weakly correlated features** | | | | |
| **Mean** | 0.777 | 0.878 | 0.659 | 0.746 | 0.837 |
| **SD** | 0.049 | 0.072 | 0.105 | 0.069 | 0.059 |
| **Max** | 0.857 | 0.958 | 0.795 | 0.849 | 0.887 |
| **Min** | 0.727 | 0.765 | 0.487 | 0.644 | 0.719 |
|  | **With moderately-to-strongly correlated features** | | | | |
| **Mean** | 0.786 | 0.875 | 0.734 | 0.797 | 0.843 |
| **SD** | 0.037 | 0.053 | 0.034 | 0.033 | 0.029 |
| **Max** | 0.844 | 0.944 | 0.773 | 0.85 | 0.886 |
| **Min** | 0.727 | 0.811 | 0.682 | 0.741 | 0.808 |

Table 6 shows that the best models for predicting multi-class treatment response are Logistic Regression in terms of accuracy (73.6%) and recall (73.6%), Naive Bayes in terms of precision (70.6%), Random Forest in terms of F1 score (69.5%), and Support Vector Machine in terms of AUC (86.4%).

Table 6: Summary of overall performance of different models in predicting multi-class treatment response.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **Decision Tree** | | | | |
|  | **Accuracy** | **Precision** | **Recall** | **F1 Score** | **AUC** |
| **Mean** | 0.640 | 0.647 | 0.640 | 0.642 | 0.697 |
| **SD** | 0.053 | 0.025 | 0.053 | 0.037 | 0.033 |
| **Max** | 0.701 | 0.672 | 0.701 | 0.685 | 0.734 |
| **Min** | 0.61 | 0.622 | 0.61 | 0.616 | 0.67 |
|  | **Naive Bayes** | | | | |
| **Mean** | 0.550 | **0.706** | 0.550 | 0.583 | 0.794 |
| **SD** | 0.177 | 0.023 | 0.177 | 0.121 | 0.006 |
| **Max** | 0.74 | 0.728 | 0.74 | 0.707 | 0.799 |
| **Min** | 0.39 | 0.683 | 0.39 | 0.465 | 0.788 |
|  | **K-Nearest Neighbors** | | | | |
| **Mean** | 0.692 | 0.636 | 0.692 | 0.655 | 0.752 |
| **SD** | 0.040 | 0.018 | 0.040 | 0.029 | 0.007 |
| **Max** | 0.727 | 0.65 | 0.727 | 0.676 | 0.759 |
| **Min** | 0.649 | 0.616 | 0.649 | 0.622 | 0.745 |
|  | **Random Forest** | | | | |
| **Mean** | 0.705 | 0.688 | 0.705 | **0.695** | 0.769 |
| **SD** | 0.008 | 0.014 | 0.008 | 0.010 | 0.036 |
| **Max** | 0.714 | 0.697 | 0.714 | 0.705 | 0.792 |
| **Min** | 0.701 | 0.672 | 0.701 | 0.685 | 0.728 |
|  | **Support Vector Machine** | | | | |
| **Mean** | 0.710 | 0.611 | 0.710 | 0.629 | **0.864** |
| **SD** | 0.038 | 0.027 | 0.038 | 0.055 | 0.010 |
| **Max** | 0.753 | 0.642 | 0.753 | 0.693 | 0.875 |
| **Min** | 0.688 | 0.595 | 0.688 | 0.597 | 0.858 |
|  | **Logistic Regression** | | | | |
| **Mean** | **0.736** | 0.636 | **0.736** | 0.680 | 0.759 |
| **SD** | 0.020 | 0.007 | 0.020 | 0.005 | 0.007 |
| **Max** | 0.753 | 0.644 | 0.753 | 0.685 | 0.767 |
| **Min** | 0.714 | 0.63 | 0.714 | 0.675 | 0.753 |
|  | **Multi-layer Perceptron** | | | | |
| **Mean** | 0.701 | 0.638 | 0.701 | 0.666 | 0.739 |
| **SD** | 0.026 | 0.020 | 0.026 | 0.012 | 0.018 |
| **Max** | 0.727 | 0.659 | 0.727 | 0.678 | 0.759 |
| **Min** | 0.675 | 0.619 | 0.675 | 0.654 | 0.724 |

As shown in Table 7, the best models in predicting binary treatment response are Multi-layer Perceptron in terms of accuracy (81.8%), recall (75%), and F1 score (81.7%), and Naive Bayes in terms of precision (95.1%) and AUC (88.7%).

Table 7: Summary of overall performance of different models in predicting binary treatment response.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **Decision Tree** | | | | |
|  | **Accuracy** | **Precision** | **Recall** | **F1 Score** | **AUC** |
| **Mean** | 0.727 | 0.785 | 0.692 | 0.735 | 0.755 |
| **SD** | 0.000 | 0.023 | 0.031 | 0.021 | 0.052 |
| **Max** | 0.727 | 0.811 | 0.727 | 0.753 | 0.815 |
| **Min** | 0.727 | 0.765 | 0.667 | 0.712 | 0.719 |
|  | **Naive Bayes** | | | | |
| **Mean** | 0.812 | **0.951** | 0.682 | 0.79 | **0.887** |
| **SD** | 0.046 | 0.010 | 0.129 | 0.085 | 0.001 |
| **Max** | 0.844 | 0.958 | 0.773 | 0.85 | 0.887 |
| **Min** | 0.779 | 0.944 | 0.59 | 0.73 | 0.886 |
|  | **K-Nearest Neighbors** | | | | |
| **Mean** | 0.760 | 0.829 | 0.694 | 0.753 | 0.805 |
| **SD** | 0.028 | 0.001 | 0.112 | 0.066 | 0.005 |
| **Max** | 0.779 | 0.829 | 0.773 | 0.8 | 0.808 |
| **Min** | 0.74 | 0.828 | 0.615 | 0.706 | 0.801 |
|  | **Random Forest** | | | | |
| **Mean** | 0.779 | 0.827 | 0.747 | 0.785 | 0.840 |
| **SD** | 0.018 | 0.003 | 0.004 | 0.001 | 0.011 |
| **Max** | 0.792 | 0.829 | 0.75 | 0.786 | 0.848 |
| **Min** | 0.766 | 0.825 | 0.744 | 0.784 | 0.832 |
|  | **Support Vector Machine** | | | | |
| **Mean** | 0.766 | 0.932 | 0.607 | 0.727 | 0.865 |
| **SD** | 0.055 | 0.025 | 0.170 | 0.117 | 0.018 |
| **Max** | 0.805 | 0.95 | 0.727 | 0.81 | 0.878 |
| **Min** | 0.727 | 0.914 | 0.487 | 0.644 | 0.852 |
|  | **Logistic Regression** | | | | |
| **Mean** | 0.812 | 0.909 | 0.723 | 0.805 | 0.872 |
| **SD** | 0.009 | 0.008 | 0.006 | 0.007 | 0.004 |
| **Max** | 0.818 | 0.914 | 0.727 | 0.81 | 0.874 |
| **Min** | 0.805 | 0.903 | 0.718 | 0.8 | 0.869 |
|  | **Multi-layer Perceptron** | | | | |
| **Mean** | **0.818** | 0.899 | **0.750** | **0.817** | 0.846 |
| **SD** | 0.055 | 0.018 | 0.064 | 0.045 | 0.012 |
| **Max** | 0.857 | 0.912 | 0.795 | 0.849 | 0.854 |
| **Min** | 0.779 | 0.886 | 0.705 | 0.785 | 0.837 |

The highest accuracy, recall, and F1 score across all experiments were 81.8%, 75%, and 81.7%, respectively. These results were achieved by Multi-layer Perceptron when it was used for predicting binary treatment response using all features, excluding those weakly correlated with the treatment response. However, the highest precision and AUC across all experiments were 95.1% and 88.7%, respectively. These results were attained when using Naive Bayes for predicting binary treatment response using all features.

## Comparison with Previous Studies

Our findings align closely with the findings reported in Sa et al.'s study [19]. To elaborate, Sa et al. investigated the performance of six algorithms (Logistic Regression, Support Vector Machine, Random Forest, Neural Networks, Adaptive Boosting, and Gradient Boost) in predicting responses (effective response vs. non-effective response) to radioiodine therapy and thyrotropin (TSH) suppression therapy in patients with differentiated thyroid cancer but without structural disease, based on pre-treatment information [19]. The study showed that Random Forest outperformed other models in predicting responses to radioiodine therapy (accuracy of 81.3%, sensitivity of 79.5%, and AUC of 0.896) and thyrotropin (TSH) suppression therapy (accuracy of 78.7%, sensitivity of 79.7%, and AUC of 0.857).

Our research demonstrated superior predictive capabilities of AI models in predicting treatment response compared to the findings presented in Grani et al.'s study [17]. To be more precise, Grani et al.'s study [17] assessed the effectiveness of 2 decision tree models to predict treatment response among patients with differentiated thyroid cancer. The highest sensitivity achieved by the decision tree models was 49%. This inconsistency in results may be attributed to several factors: (1) this study did not use AI models other than decision tree, (2) the follow-up period was shorter (< 7 years) than the follow-up period in the data used in our study, (3) several features were not used in their models such as smoking status, history of radiation therapy, presence of goiter, presence of adenopathy, and risk assessment according to American Thyroid Association (ATA) guidelines.

## Practical and research implications

Considering that our models did not achieve optimal performance in predicting the response to thyroid cancer treatment, healthcare providers should not rely exclusively on our models for treatment response predictions. Instead, they should utilize our models in combination with other tools and approaches until further studies show optimal performance of the AI models. Our research indicates that there is still potential for enhancing AI's ability to predict treatment responses. As a result, future investigations should focus on refining this predictive capability by incorporating additional features not considered in our study, such as body mass index, tumor size, Thyroglobulin level, family history of thyroid cancer, radioiodine uptake (RAIU%), surgical approach, surgical margins, presurgical cytology, and number of removed lymph nodes. Moreover, exploring alternative models like Adaptive Boosting, Gradient Boost, and Deep Neural Networks, as well as employing larger and more balanced datasets, could contribute to improved predictions. Additionally, it is imperative for future studies to use other validation techniques (e.g., k-fold cross-validation and leave-one-out cross-validation) when splitting a dataset, especially when the dataset size is small. Furthermore, researchers should assess the performance of AI models in predicting responses to various types of thyroid cancer treatments, including total thyroidectomy, partial thyroidectomy (thyroid lobectomy), and thyroidectomy plus Radioactive Iodine Therapy. Prioritizing the prediction of multi-class treatment responses is also essential, given its greater clinical significance compared to predicting binary responses.

## Limitations

Despite the significance of this study, several noteworthy limitations should be acknowledged in this study. Firstly, the generalizability of our models to datasets beyond our specific dataset is uncertain because external validation was impossible due to the absence of comparable data from other institutions. Secondly, the internal validation dataset closely mirrors the training dataset, raising the possibility of potential model overfitting. Thirdly, the study did not use validation techniques other than training-test split to split the dataset. Fourthly, the limited size of the dataset constrained the development of more advanced deep learning techniques. Lastly, this paper cannot provide insights into the performance of AI in predicting the response of specific thyroid cancer treatments, as the dataset lacked information on the types of treatments administered.

# Conclusion

AI demonstrates satisfactory performance in predicting the response to thyroid cancer treatment, yet there is room for optimization. AI models used for predicting binary treatment responses outperformed AI models used for predicting multi-class treatment responses. The highest accuracy, recall, and F1 score in all experiments were achieved by Multi-layer Perceptron when it was used for predicting binary treatment response using all features, excluding those weakly correlated with the treatment response. However, the highest precision and AUC across all experiments were attained when using Naive Bayes for predicting binary treatment response using all features.

While our models fell short of optimal performance in predicting thyroid cancer treatment response, healthcare providers should not rely solely on them. Combining our models with other tools is recommended until further studies establish their optimal performance. Future research should enhance predictive capabilities by including additional features, exploring alternative models, and utilizing larger and balanced datasets. Validation techniques like k-fold and leave-one-out cross-validation are crucial, especially with small datasets. Evaluating AI models across various thyroid cancer treatments and prioritizing multi-class responses remains imperative for comprehensive clinical insights.

# References

1. Nguyen QT, Lee EJ, Huang MG, Park YI, Khullar A, Plodkowski RA. Diagnosis and treatment of patients with thyroid cancer. American health & drug benefits. 2015 Feb;8(1):30-40. PMID: 25964831.

2. Pizzato M, Li M, Vignat J, Laversanne M, Singh D, La Vecchia C, et al. The epidemiological landscape of thyroid cancer worldwide: GLOBOCAN estimates for incidence and mortality rates in 2020. The Lancet Diabetes & Endocrinology. 2022 2022/04/01/;10(4):264-72. doi: <https://doi.org/10.1016/S2213-8587(22)00035-3>.

3. Schneider DF, Chen H. New developments in the diagnosis and treatment of thyroid cancer. CA: A Cancer Journal for Clinicians. 2013 2013/11/01;63(6):373-94. doi: <https://doi.org/10.3322/caac.21195>.

4. Filetti S, Durante C, Hartl D, Leboulleux S, Locati LD, Newbold K, et al. Thyroid cancer: ESMO Clinical Practice Guidelines for diagnosis, treatment and follow-up††Approved by the ESMO Guidelines Committee: February 2008, last update September 2019. This publication supersedes the previously published version—Ann Oncol 2012; 23(Suppl 7): vii110–vii119. Annals of Oncology. 2019 2019/12/01/;30(12):1856-83. doi: <https://doi.org/10.1093/annonc/mdz400>.

5. Kitahara CM, Sosa JA. The changing incidence of thyroid cancer. Nature reviews Endocrinology. 2016 Nov;12(11):646-53. PMID: 27418023. doi: 10.1038/nrendo.2016.110.

6. Haugen BR, Alexander EK, Bible KC, Doherty GM, Mandel SJ, Nikiforov YE, et al. 2015 American Thyroid Association Management Guidelines for Adult Patients with Thyroid Nodules and Differentiated Thyroid Cancer: The American Thyroid Association Guidelines Task Force on Thyroid Nodules and Differentiated Thyroid Cancer. Thyroid : official journal of the American Thyroid Association. 2016 Jan;26(1):1-133. PMID: 26462967. doi: 10.1089/thy.2015.0020.

7. Ma X, Xi B, Zhang Y, Zhu L, Sui X, Tian G, et al. A Machine Learning-based Diagnosis of Thyroid Cancer Using Thyroid Nodules Ultrasound Images. Current Bioinformatics. 2020 //;15(4):349-58. doi: 10.2174/1574893614666191017091959.

8. Bellantuono L, Tommasi R, Pantaleo E, Verri M, Amoroso N, Crucitti P, et al. An eXplainable Artificial Intelligence analysis of Raman spectra for thyroid cancer diagnosis. Scientific Reports. 2023 2023/10/03;13(1):16590. doi: 10.1038/s41598-023-43856-7.

9. Zhu Y-C, Du H, Jiang Q, Zhang T, Huang X-J, Zhang Y, et al. Machine Learning Assisted Doppler Features for Enhancing Thyroid Cancer Diagnosis. 2022;41(8):1961-74. doi: <https://doi.org/10.1002/jum.15873>.

10. Li Y, Wu F, Ge W, Zhang Y, Hu Y, Zhao L, et al. Risk stratification of papillary thyroid cancers using multidimensional machine learning. International Journal of Surgery. 2024;110(1).

11. Mourad M, Moubayed S, Dezube A, Mourad Y, Park K, Torreblanca-Zanca A, et al. Machine Learning and Feature Selection Applied to SEER Data to Reliably Assess Thyroid Cancer Prognosis. Scientific Reports. 2020 2020/03/20;10(1):5176. doi: 10.1038/s41598-020-62023-w.

12. Yang CQ, Gardiner L, Wang H, Hueman MT, Chen D. Creating Prognostic Systems for Well-Differentiated Thyroid Cancer Using Machine Learning. 2019 2019-May-08;10. doi: 10.3389/fendo.2019.00288.

13. Borzooei S, Briganti G, Golparian M, Lechien JR, Tarokhian A. Machine learning for risk stratification of thyroid cancer patients: a 15-year cohort study. European archives of oto-rhino-laryngology : official journal of the European Federation of Oto-Rhino-Laryngological Societies (EUFOS) : affiliated with the German Society for Oto-Rhino-Laryngology - Head and Neck Surgery. 2023 Oct 30. PMID: 37902840. doi: 10.1007/s00405-023-08299-w.

14. Park YM, Lee B-J. Machine learning-based prediction model using clinico-pathologic factors for papillary thyroid carcinoma recurrence. Scientific Reports. 2021 2021/03/02;11(1):4948. doi: 10.1038/s41598-021-84504-2.

15. Kil J, Kim KG, Kim YJ, Koo HR, Park JS. Deep Learning in Thyroid Ultrasonography to Predict Tumor Recurrence in Thyroid Cancers. Taehan Yongsang Uihakhoe chi. 2020 Sep;81(5):1164-74. PMID: 36238043. doi: 10.3348/jksr.2019.0147.

16. Seib CD, Roose JP, Hubbard AE, Suh I. Ensemble machine learning for the prediction of patient-level outcomes following thyroidectomy. American journal of surgery. 2021 Aug;222(2):347-53. PMID: 33339618. doi: 10.1016/j.amjsurg.2020.11.055.

17. Grani G, Gentili M, Siciliano F, Albano D, Zilioli V, Morelli S, et al. A Data-Driven Approach to Refine Predictions of Differentiated Thyroid Cancer Outcomes: A Prospective Multicenter Study. The Journal of clinical endocrinology and metabolism. 2023 Jul 14;108(8):1921-8. PMID: 36795619. doi: 10.1210/clinem/dgad075.

18. Lubin DJ, Tsetse C, Khorasani MS, Allahyari M, McGrath M. Clinical predictors of I-131 therapy failure in differentiated thyroid cancer by machine learning: A single-center experience. World journal of nuclear medicine. 2021 Jul-Sep;20(3):253-9. PMID: 34703393. doi: 10.4103/wjnm.WJNM\_104\_20.

19. Sa R, Yang T, Zhang Z, Guan F. Random Forest for Predicting Treatment Response to Radioiodine and Thyrotropin Suppression Therapy in Patients With Differentiated Thyroid Cancer But Without Structural Disease. The oncologist. 2024 Jan 5;29(1):e68-e80. PMID: 37669005. doi: 10.1093/oncolo/oyad252.

20. Borzooei S, Tarokhian A. Differentiated Thyroid Cancer Recurrence. 2023 [March 09, 2023]; Available from: <https://archive.ics.uci.edu/dataset/915/differentiated+thyroid+cancer+recurrence>.

21. van der Walt S, Colbert SC, Varoquaux G. The NumPy Array: A Structure for Efficient Numerical Computation. Computing in Science & Engineering. 2011;13(2):22-30. doi: 10.1109/MCSE.2011.37.

22. McKinney W, editor. Data structures for statistical computing in python. Proceedings of the 9th Python in Science Conference; 2010: Austin, TX.

23. Hunter JD. Matplotlib: A 2D Graphics Environment. Computing in Science & Engineering. 2007;9(3):90-5. doi: 10.1109/MCSE.2007.55.

24. Waskom ML. seaborn: statistical data visualization. Journal of Open Source Software. 2021;6(60). doi: 10.21105/joss.03021.

25. Pedregosa F, Varoquaux G, Gramfort A, Michel V, Thirion B, Grisel O, et al. Scikit-learn: Machine learning in Python. Journal of Machine Learning Research. 2011;12:2825-30.

26. Lemaître G, Nogueira F, Aridas CK. Imbalanced-learn: A python toolbox to tackle the curse of imbalanced datasets in machine learning. Journal of Machine Learning Research. 2017;18(17):1-5.