University of the Philippines Manila
College of Arts and Sciences
Department of Physical Sciences and Mathematics

SLEvival: Predicting Mortality Risk in Systemic Lupus Erythematosus Patients with Explainable Machine Learning

A special problem in partial fulfillment
of the requirements for the degree of
Bachelor of Science in Computer Science
Submitted by:

Ivan R. Baluyut

June 2023

Permission is given for the following people to have access to this SP:

Available to the general public	Yes
Available only after consultation with author/SP adviser	No
Available only to those bound by confidentiality agreement	No

ACCEPTANCE SHEET

The Special Problem entitled "SLEvival: Predicting Mortality Risk in Systemic Lupus Erythematosus Patients with Explainable Machine Learning" prepared and submitted by Ivan R. Baluyut in partial fulfillment of the requirements for the degree of Bachelor of Science in Computer Science has been examined and is recommended for acceptance.

Geoffrey A. Solano, Ph.D.
Adviser

EXAMINERS:

	Approved	**Disapproved**
1. Avegail D. Carpio, M.Sc.	_____	_____
2. Richard Bryann L. Chua, Ph.D. (*cand.*)	_____	_____
3. Perlita E. Gasmen, M.Sc. (*cand.*)	_____	_____
4. Ma. Sheila A. Magboo, Ph.D. (*cand.*)	_____	_____
5. Vincent Peter C. Magboo, M.D.	_____	_____
6. Marbert John C. Marasigan, M.Sc. (*cand.*)	_____	_____

Accepted and approved as partial fulfillment of the requirements for the degree of Bachelor of Science in Computer Science.

Vio Jianu C. Mojica, M.Sc.
Unit Head
Mathematical and Computing Sciences Unit
Department of Physical Sciences and Mathematics

Marie Josephine M. De Luna, Ph.D.
Chair
Department of Physical Sciences and Mathematics

Maria Constancia O. Carrillo, Ph.D.
Dean
College of Arts and Sciences

Abstract

Systemic Lupus Erythematosus (SLE) is an autoimmune disease with unknown causes and no current cure. While Lupus Low Disease Activity State (LLDAS), an attainable treat-to-target goal in SLE, has been associated with reduced damage accrual and decreased mortality risk, the number of deaths remains significantly high. Among of these deaths have been found to be influenced by demographic and clinical factors such as race, sex, infection, and disease activity. Most studies conducted in SLE were statistical analyses and machine learning approach seems to be very limited on the topic. On the other hand, machine learning have been widely utilized in modern healthcare for various disease prediction studies. Additionally, the Asia Pacific Lupus Collaboration (APLC) cohort provides a dataset that has been commonly included in SLE works. Hence, this study proposes the use of machine learning in creating a prediction system for mortality risk in SLE patients. Label Encoder, Ordinal Encoder, One Hot Encoder, Single Imputation, and Multiple Imputation by Chained Equations (MICE) were applied to create the imputed dataset. Synthetic Minority Oversampling Technique (SMOTE), Recursive Feature Elimination with Cross-Validation (RFECV), and Standard Scaler were further applied to produce 15 more dataset variations. Random Forest, XGBoost, Support Vector Machine, and Logistic Regression were trained on the 16 datasets—developing a total of 64 models. Using AUROC as the main metric, results have shown that the XGBoost configured on the SMOTE dataset was the best performing model with an AUROC of 85.1%. Integrating Local Interpretable Model-agnostic Explanations (LIME) with the best XGBoost, a web application was built that allows a user to input real patient health data and view the mortality risk prediction outcome with explanations firsthand.

Keywords: SLE, lupus, mortality risk, prediction, patient health data, machine learning, APLC, MICE, SMOTE, RFECV, standard scaler, XGBoost, LIME

Contents

List of Figures

List of Tables

I. Introduction

A. Background of the Study

Systemic Lupus Erythematosus (SLE) is a type of autoimmune disease that causes inflammation in different parts of the body - affecting mostly women (9 out of 10 of lupus patients are women).[1] Autoimmune diseases occur when the immune system attacks the healthy tissues and organs of the body - causing widespread inflammation and damage in multiple organs. In most cases, SLE is difficult to detect since it often mimics the symptoms of other illnesses. Moreover, it was found that the effects and symptoms of Lupus vary from person to person - some could be mild, some could be fatal. While there are several developments in healthcare, SLE has no current cure and mortality rate remains significantly high.[2] And even if a person does survive Lupus, the disease might have already caused chronic damages to the body - potentially affecting the quality of life of the patient for the rest of one's time.

To reduce the mortality risk and damage-accrual among SLE patients, researchers and physicians developed a treat-to-target goal which is the attainment of LLDAS. Lupus Low Disease Activity State (LLDAS) is considered as an improved state or better condition for SLE patients. LLDAS is commonly defined with the standard definition used in Asia Pacific Lupus Collaboration (APLC) cohort. Additionally, LLDAS attainment has been acknowledged to reduce mortality risk and damage-accrual.[3][4]

Moreover, machine learning models have been widely used in modern healthcare for various disease prediction studies - from chronic kidney disease diagnosis to mortality risk in post-cardiac surgery, and other known diseases.[5][6] Explainable machine learning methodologies have also been found essential in healthcare as it aids in explaining the factors critical for the detection and mortality of diseases such as Alzheimer's and acute myocardial infarction.[7][8] While there are various literatures

regarding SLE detection and association between LLDAS and SLE mortality, machine learning implementation seems to be lacking in SLE prediction studies - having few attempts to unfold the key factors leading to deaths.

On the other hand, the APLC cohort provides a dataset that is commonly used in SLE studies. The dataset contains the health data (demographics, medication, clinical profile, and health-related quality of life) of $4,106$ SLE patients collected from May 1, 2013, and December 31, 2020. The said health data are a combination of categorical and continuous data. However, the dataset contains significant amount of missing values and shows imbalance in the mortality class (e.g., out of 3811 patients, 80 have died and $3,731$ were alive).[9]

Hence, this study proposes an explainable machine learning approach to predict the mortality risk in SLE patients and identify the most contributing features to the probability of a patient's risk of death by using the APLC dataset in the development of the models and the system as a whole.

B. Statement of the Problem

Lupus affects at least five million people around the world and detection of this disease is difficult and costly as its symptoms are often mistaken for other diseases and combination of tests are needed for diagnosis. Up to this date where healthcare continues to progress, the cure for lupus is still not far from a myth. And even with the development of a treat-to-target plan i.e., LLDAS which is associated with reduced damage accrual and improved quality of life, the number of patient deaths is still significant.

Moreover, machine learning implementations are very limited in SLE diagnosis and mortality studies. The development of a system dedicated for predicting mortality risk in SLE can be useful for physicians in augmenting their medical decisions and treatment plans.

C. Objectives of the Study

C..1 General Objectives

The study aims to provide a support tool for the medical-decision making of physicians through a web application integrated with a system that predicts the mortality risk in SLE patients and identifies the critical factors associated to the prediction outcome using machine learning models with explainable machine learning developed from the APLC dataset.

C..2 Specific Objectives

1. Explore the APLC dataset and prepare the data for analysis and modeling by performing necessary data preprocessing methods.

    (a) Apply feature encoding techniques such as Label Encoding, Ordinal Encoding, and One Hot Encoding to transform categorical data into numerical data.

    (b) Use Single Imputation (SI) and Multiple Imputation by Chained Equations (MICE) to handle the missing values for categorical and continuous variables respectively.

    (c) Use Synthetic Minority Oversampling Technique (SMOTE) to handle imbalance in the dataset.

    (d) Apply feature selection using Recursive Feature Elimination with Cross-Validation (RFECV) for possible reduction of dimensionality in the dataset.

    (e) Apply feature scaling using Standard Scaler to standardize the data.

2. Develop four machine learning models - Random Forest, Support Vector Machine, Extreme Gradient Boosting, and Logistic Regression - to predict SLE mortality risk using patient health data from the APLC cohort dataset.

(a) Train and optimize the four models by performing randomized search hyperparameter tuning with cross-validation.

(b) Test and evaluate the performance of the four models using AUROC and other classification metrics such as accuracy, precision, recall, and F1 score.

(c) Show feature importance of the best performing model.

(d) Integrate model agnostic explanation method such as LIME with the best performing model to determine most contributing features in predicting SLE mortality risk.

3. Develop a web application integrated with the best performing model and model explainer, and provide a user interface to allow the physician to:

(a) Input patient health-related data such as demographic information, pathology results, medication, diagnosis details, etc.

(b) View prediction results.

(c) View explanations (feature contributions) of the predicted mortality risk.

D. Significance of the Project

The study proposes a machine learning approach in the prediction of SLE mortality which is lacking in existing works. The study incorporates explainability techniques to provide better understanding of the outcomes - identifying key factors highly associated with the prediction outcome. Development of a web application dashboard is also included in the study - allowing doctors to better visualize the underlying associations and giving them an additional support in their medical decisions and plans. Moreover, communicating the chances of survival (or death) to patients can help in reducing the anxiety of the patient and possibly rethinking of their medical decisions (for the doctor as well).

E. Scope and Limitations

1. The intended users are rheumatologists who specializes in the diagnosis and treatment of diseases that affects muscles and joints such as SLE.

2. The study uses available dataset from the APLC cohort.

3. The developed system only predicts whether the patient is at risk of mortality or not (i.e., classify as "Yes" and "No") and provides the most contributing features to the prediction.

4. The developed system does not include and consider the relevance of survival time in the outcome (i.e., how much time the patient has left until he/she dies).

5. The developed system should only serve as a support tool and should not be used as the main basis of medical decision.

F. Assumptions

1. The user is a rheumatologist or a doctor with sufficient knowledge on SLE and the variables being asked.

2. The user has access on the health data of the patient used as subject on the system.

3. The user will enter only correct inputs.

II. Review of Related Literature

The causes of SLE are unknown and it currently has no cure. Various studies have shown that LLDAS (and remission in some cases) is associated with decreased damage accrual and improved health-related quality of life (HRQoL) in patients. While there are existing works on the association between LLDAS (or other factors) and mortality risk of SLE patients, there has been few to no study that offers a way to predict the mortality risk of the patients and explain the contribution of different factors on such outcomes.

A. Cause of Mortality in SLE

Systemic Lupus Erythematosus (SLE) is the most common type of lupus - an autoimmune disease that causes widespread inflammation and tissue damage in affected organs which can be life-threatening in some cases. While anyone can be at risk for SLE, various studies have shown that demographics is associated with the person's susceptibility to SLE. In terms of geography, ethnicity, and age, incidence and prevalence estimates were highest in North America, Black Ethnicity, and middle adulthood respectively.[10] Moreover, women were more frequently affected than men regardless of age and racial group.

This association also holds true in the mortality risk of an SLE patient. An analysis on the 2002-2004 population-based data from Georgia Lupus Registry suggested that Blacks have higher mortality than Whites while there were no significant differences in terms of sex.[11] In other studies, female sex, younger age, and low socioeconomic status were found to be associated with high mortality.[12][13][14] Meanwhile, medical information are also significant factors. In a 5-year retrospective analysis using 53 death records in India, patient mortality was attributed due to high disease activity and concomitant infection (e.g., bacterial pneumonia, tuberculosis).[15] Alar-

con et. al. also concluded that disease activity and disease damage are important determinants of mortality in SLE patients.[14] Moreover, patient-reported outcomes were also proved to predict SLE mortality - independent of demographics and disease activity/damage.[16]

B. Definition and Validation of LLDAS

Despite the progress in the medical industry in the past years, there has been little to zero evidence of a cure for SLE.[2] Meanwhile, treat-to-target strategies were found to improve outcomes in single-organ diseases. One such strategy is treating low disease activity in rheumatoid arthritis. And so, researchers and physicians have adopted this approach in treating SLE. Lupus Low Disease Activity State (LLDAS) is an attainable treat-to-target goal and a meaning SLE outcome measure which provides significant protection against disease flare and damage accrual.[3][4] As the name suggests, LLDAS is a better or an improved condition of lupus.

To determine if a patient has achieved LLDAS, doctors use standard criteria. In their study, Franklyn et. al. generated the following consensus definition of LLDAS which was also referenced as the proposed definition of Asia Pacific Lupus Collaboration cohort in later works:[17]

1. SLE Disease Activity Index (SLEDAI)-2K ≤ 4, with no activity in major organ systems (renal, central nervous system (CNS), cardiopulmonary, vasculitis, fever) and no haemolytic anaemia or gastrointestinal activity.

2. No new lupus disease activity compared with the previous assessment.

3. A Safety of Estrogens in Lupus Erythematosus National Assessment (SELENA)-SLEDAI physician global assessment (scale 0–3) ≤ 1.

4. A current prednisolone (or equivalent) dose ≤ 7.5 mg daily.

5. Well tolerated standard maintenance doses of immunosuppressive drugs and approved biological agents.

C. LLDAS and SLE Mortality Risk

Since LLDAS has shown positive results on improving the quality of life, it is believed that it also helps in decreasing the mortality risk of an SLE patient. In a prospective cohort study, Kandane-Rathnayake et. al. analyzed data from 3,811 patients with SLE in the Asia Pacific Lupus Collaboration (APLC) cohort collected from 2013 to 2020. Throughout their study, they found out that 43 of the 80 participants who died and 3,035 of the 3,731 participants who were alive at the end of the analysis have attained LLDAS at least once.[9] Patients who died had significantly higher disease activity than those who stayed alive. Additionally, 22 patients who died and 1,966 patients who stayed alive achieved LLDAS at least 50% of their observed time. The researchers concluded that LLDAS for at least 50% of observed time is associated with reduced risk of mortality.

Sharma et. al. also assessed the impact of LLDAS-50 on damage accrual and mortality. They defined LLDAS using the standard APLC definition except, they did not consider the physician global assessment scale.[18] Using data from the Tromsø Lupus Cohort, they discovered that LLDAS-50 was associated with significant reduction in risk of having severe damage, and mortality after correcting for age and sex.

D. Machine Learning in Disease Prediction Studies

Up to this date, few to no machine learning models have been developed for mortality risk prediction in SLE patients and/or patients who attained LLDAS. Meanwhile, various studies have used machine learning models in the medical field - predicting susceptibility, diagnosis, and mortality of patients with certain illnesses. Two of such models are the Random Forest (RF) and Support Vector Machine (SVM) which are

found to be applied most frequently and have the highest accuracy in disease prediction studies. Pontiveros et. al. identified RF as the best performing model with AUC of 92.26% in predicting patient's susceptibility to SLE using single-nucleotide polymorphisms.[19] In the study of Park et. al, RF performed the best in predicting incidence of Alzheimer's disease using health data, followed by SVM.[20] Additionally, RF also obtained good accuracy and AUC in prediction of heart and cardiovascular disease respectively.[21][22]

In predicting in-hospital mortality of patients with acute kidney injury, RF and SVM outperformed Artificial Neural Network (ANN) and the customized model.[23] After performing 10-fold cross validation, RF reached an accuracy of 89%, followed by Gradient Boosting (84%), SVM (82%), and NN (80%) in prediction of post-cardiac surgery mortality risk.[5] In predicting mortality of COVID-19 patients, RF also performed the best against the same models.[24]

On the other hand, Extreme Gradient Boosting (XGBoost) has also been used for its high performance in several studies such as chronic kidney disease diagnosis, diabetes prediction, etc.[6][25] Among the machine learning models used in predicting mortality in patients with spontaneous coronary artery dissection, XGBoost obtained the highest mean accuracy but fell short in AUC against RF and SVM.[26] XGBoost with Bayesian optimization also outperformed RF in heart disease prediction with a classification accuracy of 91.8%.[27]

Moreover, Neural Networks (NN) have also been used in other healthcare studies. Specifically, Recurrent Neural Networks (RNN) were used to predict chronic damage in SLE - obtaining an AUC of 0.77 when tested on clinical data sequences.[28] In another study about mortality risk prediction in COVID-19 patients, NN also performed well along with RF and SVM, with an accuracy of 89.98%.[29]

E. Explainable Machine Learning in Healthcare

Machine learning models have been widely used in disease diagnosis and/or mortality prediction. However, these models focuses on classification only with limited interpretability - making it difficult for physicians to understand and trust the results. That is why explainable machine learning (XML), or explainable artificial intelligence (XAI), has been essential in the medical industry as it helps interpret the process on how a model arrived at such decision. XML techniques can be used to determine the significant factors that led to the prediction outcomes in disease studies - providing explanations to physicians to better support their medical decisions.

In their study, Ibrahim et. al. designed an XGBoost model for prediction of Acute Myocardial Infarction (AMI) and obtained a high accuracy of 97.5%. To examine the contribution of different features on the prediction, the researchers analyzed Shapley values and found that age and two ECG components (ACCI and QRS duration) were the most crucial variable in predicting onset of AMI.[7] Similarly, in a study on early prediction of prognosis in acute kidney injury, SHAP was able to determine the four most contributing variables: GCS, blood urea nitrogen, cumulative urine output, and age. Local Interpretable Model-Agnostic Explanations (LIME) algorithm was also used to explain the individual predictions of death.[30] On a separate study, SHapley Additive exPlanations (SHAP) was used to calculate feature contributions in RF model outcomes and find the most and least influential factor in detection and prediction of Alzheimer's disease.[8] SHAP was also used in predicting acute gastrointestinal bleed mortality.[31]

Additionally, different XAI methods were used in various disease prediction studies such as RF with SMOTE and ADASYN, XGBoost with SHAP, etc. which helped the different models achieve maximum accuracy.[32]

By discussing and reviewing the related literatures, the following have been found

and understood:

1. SLE has no cure and mortality is influenced by demographic and clinical factors such as race, sex, infection, and disease activity.

2. LLDAS is an attainable treat-to-target goal in SLE associated with reduced damage accrual and decreased mortality risk.

3. Machine learning models such as RF, SVM, and XGBoost are useful in disease prediction studies.

4. XML is essential in explaining key factors that contributes to prediction outcomes in disease diagnosis and mortality.

Since existing predictive models are limited in mortality risk prediction in SLE, this study attempts to fill in the current gaps by implementing machine learning models on available datasets from APLC cohort to predict mortality risk in SLE patients. Explainable machine learning will also be utilized to determine particular key features related to the risk outcome. Moreover, a web application will be developed to provide an intuitive, simple user interface for physicians' use - making the prediction system more accessible.

III. Theoretical Framework

A. Systemic Lupus Erythematosus

Systemic Lupus Ertyhematosus (SLE), the most common among the other types of lupus (cutaneous lupus, drug-induced lupus, and neonatal lupus), is an autoimmune disease where the immune system attacks its own tissues. This leads to extensive inflammation in damaged organs such as the joints, skin, brain, lungs, kidneys, and blood vessels. The causes of SLE are unknown but studies have shown that SLE affects mostly women, adults, and people of Black/Hispanic/Asian ethnicity. Although there is no current cure for SLE, various medical interventions such as Lupus Low Disease Activity State (LLDAS) help regulate the disease.[1]

The mortality risk of an SLE patient simply refers to one's likelihood of death within a particular amount of time.

B. APLC Cohort

Asia Pacific Lupus Collaboration cohort consists of patient data from Australia, China, Hong Kong, Indonesia, Japan, Malaysia, Philippines, Singapore, Taiwan and Thailand. Data of adults (ages 18 and above) who met either the 1997 American College of Rheumatology (ACR) modified classification criteria for SLE or the 2012 Systemic Lupus International Collaborating Clinics (SLICC) classification criteria are collected using standardized case report form. Additionally, APLC proposes LLDAS as a treat-to-target endpoint in SLE.[33]

C. Patient Health Data

Patient health data are personal information which includes demographics, medical history, medication, laboratory results, etc. that is used by healthcare professionals to determine appropriate care for a certain individual. Patient data from APLC

which is used in this study, consists of the following:[9]

C..1 Demographics

Demographic information refers to the characteristics of an individual. This includes age, sex, ethnicity, socioeconomic status, level of education, income, etc. Particularly in APLC cohort, details about smoking, family history of SLE, and disease duration are also included.

C..2 Medications

Medication contains information about the administered medicines or treatments performed on patients to treat their diseases. In APLC cohort, data about medication is comprised of prednisolone doses, antimalarials, immunosuppressants, biologics (rituximab and belimumab), etc.

C..3 Clinical Profile

Clinical profile is commonly comprised of information about disease such as diagnoses, symptoms, comorbidities, laboratory results, etc. In APLC cohort, information such as SLEDAI-2K score, PGA score, disease flare, organ damage, date and cause of death, pathology data (creatinine, neutrophils, lymphocites, etc.), and other disease-related data are also included.

C..4 Health-Related Quality of Life

Health-Related Quality of Life (HRQoL) refers to the perceived physical and mental state of an individual which is assessed through a 36-item health survey form. Scores for the two components is included in the APLC cohort dataset.

D. Data Preprocessing

Data preprocessing is the method of preparing a set of data before using them for analysis and modeling. Cleaning and transforming raw data into a more suitable format is essential so that machine learning models can use the data for classification. While a model was able to predict a syndrome using incomplete, imbalanced medical data in one study, preprocessing data is still essential in improving the performance of the model.[34]

D..1 Data Encoding

Data or feature encoding is the process of converting categorical data into numerical data before they can be processed by machine learning models. There are various encoding techniques for different types of categorical data.[35]

1. Label Encoding

    - Label encoding is used for nominal data where categories don't have any specific order or ranking between them. This technique is commonly used for encoding target variables. For example, the classes of gender namely "Male" and "Female" will be encoded into 1 and 0 respectively (according to the alphabetical order).

2. Ordinal Encoding

    - Ordinal encoding is used for ordinal data where categories have a natural order or ranking between them. For example, the classes of education level namely *"Primary", "Secondary", and "Tertiary"* will be encoded into 0, 1, and 2 respectively.

3. One Hot Encoding

- One Hot Encoding transforms nominal data especially multi-class or multi-categorical variables into their binary representations. For example, one hot encoding will create new columns for each class of the variable country namely *"Philippines", "Japan", and "Singapore"* and will assign a value of 0 or 1 to indicate the presence or absence of the category in a particular row.

D..2 Data Imputation

Data imputation is the process of generating new data to fill in the missing values in a variable by using various algorithms.[36]

1. Single Imputation

   - Single Imputation is the easiest and most basic data imputation technique where missing values are replaced with the mean, median, or mode value of a column. The scikit-learn library provides an implementation of this algorithm through the *SimpleImputer* class.

2. Multiple Imputation by Chained Equations

   - Multiple Imputation by Chained Equations (MICE) is a multivariate imputation technique that fills in missing data of one variable using information from the other variables through an iterative series of predictive models. Iterations will run until a specific convergence criteria has been met. MICE have shown better performance than single imputation since it considers the other variables on the dataset and predicts a new value from them rather than using only the information from the specified variable. The scikit-learn library provides an implementation of this algorithm through the *IterativeImputer* class.

D..3 Synthetic Minority Oversampling Technique (SMOTE)

Synthetic Minority Oversampling Technique (SMOTE) is a data augmentation algorithm used to handle imbalance in a dataset. To oversample the minority class, SMOTE creates new data which are synthesized from the existing data. In a more technical sense, SMOTE selects a random row from the minority class and finds the k of the nearest neighbors of that example. Then, a neighbor is chosen at random and a new, synthetic data is created from a random point between the two examples in the feature space. The imbalanced-learn library provides an implementation of the algorithm through the *SMOTE* class.[37]

D..4 Recursive Feature Elimination with Cross-Validation (RFECV)

Feature selection is a technique used to reduce input variables by identifying the most relevant features and eliminating noise in data - increasing the prediction power of the model. One such method is the Recursive Feature Elimination with Cross-Validation (RFECV). RFECV works by fitting a model multiple times, removing the weakest features, then selecting the best subset (i.e., dataset with optimal number of chosen features) based on the cross-validation scores. By recursively eliminating small number of features for each loop, RFECV prevents the existence of collinearity in the model. The scikit-learn library provides an implementation of this algorithm through the *RFECV* class.[38]

D..5 Standard Scaling

The scale and distribution of data are important concepts in machine learning as differences in the scales across input variables may result to problems in the modeling process such as high generalization error. In simpler terms, large input values (e.g., thousands of units) can make a model learn and be sensitive to large weight values - becoming unstable when dealing with a dataset that contains a mix of high and low

values. To rescale the distribution of the values in a dataset, data standardization is performed so that the mean and standard deviation of the data becomes 0 and 1 respectively. Technically, standard scaling subtracts the mean from the values of each input variable and divides them by their standard deviation. The scikit-learn library provides an implementation of this algorithm through the *StandardScaler* class.[39]

E. Machine Learning

Machine learning is a branch of computer science and a subfield of artificial intelligence which focuses on the development of systems with the ability to adapt and learn patterns in data using statistical models and algorithms instead of being directly programmed. Epidemiologic applications of ML include clinical decision support tools using diagnostic and prognostic predictive models, genome-wide association studies, text mining for electronic health records, forecasting of infectious disease, etc.[40]

E..1 Supervised Learning

Supervised learning is a subcategory of ML where labeled data sets are used in training the model that makes predictions - providing both input and the desired output. Classification, under this type of learning, is the standard in epidemiologic practice.

1. **Random Forest**

   Random Forest (RF) uses ensemble learning and combines several randomized decision trees and aggregates their prediction outcomes to decide the final classification.[41] RF solves the overfitting issue in decision tree algorithm, produces reasonable prediction without hyper-parameter tuning, and shows excellent performance in classification studies.[42] Moreover, RF offers a way to determine variable importance.

2. **Support Vector Machine**

   Support Vector Machine (SVM) is used in classification and regression problems where an optimal boundary called hyperplane is used to separate the observations into different classes. SVM shows low misclassification rate and performs well with high-dimensional data. Additionally, SVM is versatile as different kernel functions can be used for decision function.[43]

3. **Extreme Gradient Boosting**

   Extreme Gradient Boosting (XGBoost), an implementation of the gradient boosted decision trees, uses sequential ensemble method to create a precise model. The algorithm starts with a weak classifier (i.e., a predictor slightly better than random guessing), makes prediction, and increases the weight of misclassified data points which are fed on to the next classifier – repeating the steps until best classifier is produced.[44] XGBoost generally obtains high accuracy as it reduces the misclassification rate. Moreover, can train on very large datasets as it is parallelizable to GPUs.

4. **Logistic Regression**

   Logistic Regression (LR) is commonly used in binary classification - identifying and explaining the relationship between one categorical dependent variable and one or more independent variables. Specifically, Binary LR is used in predicting dichotomous variables such as die or not die, present or absent, etc. In LR, the outcome probability (dependent variable) is bounded by 0 and 1.

E..2 Model Optimization

Model Optimization is the process of improving the quality and overall performance of the model through methods such as cross-validation and hyperparameter tuning.

1. **Cross Validation**

   Cross-validation is an approach where model is trained using a subset of the data-set (training set) and then evaluated using the complementary subset (validation set). This gives a measure of how good the classifier is in predicting unseen data.[45]

2. **Hyperparameter Tuning**

   Hyperparameter tuning is a trial-and-error process where different combinations of hyperparameters are used in training the model to find the ones that optimize the model. In particular, random search will be used to find the best combination of hyperparameters.

E..3 Performance Evaluation

Performance evaluation is done to assess the efficiency of the model using various metrics - determining how accurate the classifier is in predicting outcomes of the data set.

1. **Confusion Matrix**

   Confusion matrix is a classification metric which shows the total number of correct and incorrect predictions. The following measure can be obtained from the matrix:[46]

   (a) *Accuracy* - overall correctness calculated by the sum of correct predictions over total number of predictions.

   (b) *Sensitivity or Recall* - proportion of true positives and false negatives.

   (c) *Specificity* - proportion of true negatives and false positives.

   (d) *Precision* - proportion of true positives and false positives.

   (e) *F1 Score* - the 'harmonic mean' of precision and recall.

2. **AUC/AUROC Curve**

   Area Under the Receiver Operating Characteristic (AUROC) curve provides the total performance measure against all classification thresholds. The ROC curve plots true positive rate in the y-axis and false positive rate in the x-axis - the closer the curve is to the upper left corner, the better the efficiency of the classifier.[47]

F. Explainable Machine Learning

Machine learning models have a tendency to act as black box in prediction studies where outcomes are produced without giving explanations - making it difficult to trust the model. Explainable ML uncovers the black box and reveals the decision-making process of the model. In healthcare especially in disease and risk prediction studies, ML interpretability is essential to show the different features (in this case, clinical data) that are critical in the disease detection - potentially providing unseen association and giving additional support to the medical decision-making of physicians.[48]

To add interpretability, model-agnostic explanation methods are used together with the machine learning models.

F..1 Local Interpretable Model-agnostic Explanation

LIME focuses on training local surrogate models for explaining individual predictions. It aims to test what will happen to the predictions if a model is given variations of the data. For example, LIME was used to explain the results of Random Forest in predicting whether the number of bicycles rented from the Capital-Bikeshare company will be above or below the average on a certain day. Particularly, temperature and weather condition (dataset obtained from the UCI Machine Learning Repository) were considered in the test. As shown in Figure 1, warmer temperature and good weather situation are found to have positive effect on the prediction.

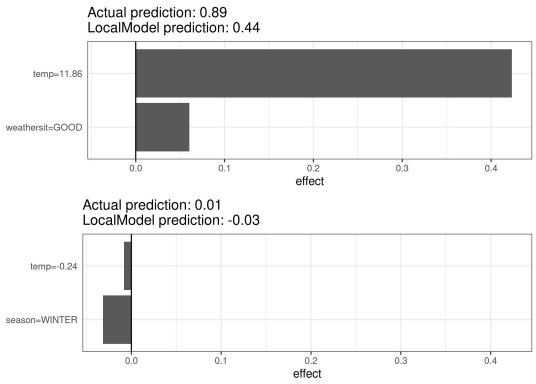


Figure 1: Sample LIME explanation for two instances of the bike rental dataset.

G. Web Application

Web application is a software system built to perform certain tasks over the Internet and accessed through web browsers. A web application has two components: frontend which handles the user interface, and backend which is responsible for the processes and functions needed in the application. In healthcare, web applications are used by patients and hospital staffs to make appointments, obtain laboratory results, access medical records, etc.

G..1 Django

Django is a web framework based on the Python programming language that is mainly responsible for the backend in building web applications.

G..2 HTML

HTML or HyperText Markup Language is used in creating the structure of the web pages.

G..3 CSS

CSS or Cascading Style Sheets provides a mechanism for adding custom style to the web pages.

IV. Design and Implementation

The main goals of this study is to create a prediction system using machine learning models, implement explainable ML techniques to assess feature contributions, and develop a web application that integrates the system for the mortality risk of SLE patients. A web dashboard is designed to make the system more accessible for physicians − allowing them to provide inputs and see the outputs firsthand.

A. The Dataset

Variable	Baseline (recruitment)	Routine Follow ups	Annual Visits
APLC ID			
Visit date	✓	✓	✓
Demographics	✓		
Date of Birth			
Gender			
Ethnicity			
Year of SLE onset			
Year of Diagnosis			
Smoking at recruitment			
Family History of SLE			
Educational Level			
Diagnosis Criteria	✓		
ACR criteria			
SLICC Classification criteria			
Pathology Data	✓	✓	✓
Pathology data collection date			
Creatinine			
eGFR			
Albumin			
CRP			
C3			
C4			
Urine Protein/creatinine ratio			
Haemoglobin			
White Cell Count (WCC)			
Platelet			
Neutrophils			
Lymphocytes			
ESR			
Anti-dsDNA			
Urine WBC			
Urine RBC			
Medication types and doses	✓	✓	✓
Prednisolone			
Hydroxychloroquine			
Chloroquine			
Methotrexate			
Azathioprine			
Mycophenolate mofetil			
Mycophenolic acid			
Leflunomide			
Cyclosporin			
Tacrolimus			
Mizoribine			
Cyclophosphamide			
Rituximab			
Belimumab			
Physician Global Assessment (PGA)	✓	✓	✓
SLE Disease Activity Index – 2k (SLEDAI – 2k)	✓	✓	✓
Mild/Moderate Flare Index	✓	✓	✓
Severe Flare Index	✓	✓	✓
SLICC Damage Index	✓		✓
SF-36v2 (Health Related Quality of Life survey)	✓		✓
Death	✓	✓	✓

Figure 2: Summary of data items included in the SLE dataset from APLC cohort.

The SLE dataset generated by the APLC cohort consists of patient health data that are categorized into demographics, pathology results, medication, and other SLE-related information. Adults (ages 18 and above) from Australia and Asian countries who met either the modified 1997 ACR or the 2012 SLICC classification criteria were enrolled in the cohort. With their consent, the patients were routinely followed up at 3− to 6−monthly intervals, hence collecting their data from May 1, 2013 to December 31, 2020 using standardized case report form.

The original dataset is comprised of 411 features (columns) and a total of $42,355$ visits (rows) from the $4,106$ patients enrolled in the cohort—some patients having multiple visits throughout the observation period. The dataset contains a combination of categorical and continuous data. A snapshot of the data dictionary is shown in Figure 2. The dataset will undergo preprocessing to handle reported missing values and data imbalance.[9]

B. System Design

B..1 Context Diagram

The context diagram shown in Figure 3 presents the general overview of the interaction between the user (physician) and the web application (*SLEvival*). The physician will input patient health data into the *SLEvival* interface, then the web application will output the calculated mortality risk of the patient along with the explanations.

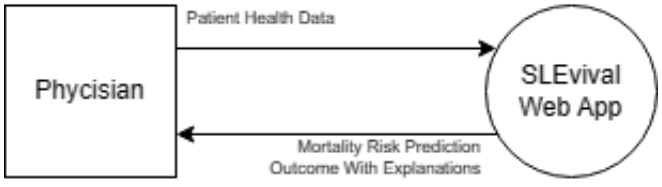


Figure 3: Context Diagram

B..2 Schematic Diagram

After acquiring the data from the APLC cohort, initial data preprocessing is performed on the dataset. In this phase, the priority is to explore and reduce the dimensionality of the dataset by filtering out some rows and columns based on the criteria or requirements of the study. Exploring the dataset includes finding out the initial size, identifying data types, checking missing values and imbalance, and analyzing some values per feature to have a grasp of the information that the dataset provides. Once the observation is done, unnecessary rows and columns will be dropped so that only relevant data are left for use in the modeling process. Then, the initially preprocessed dataset is split first into train set and test set with $70:30$ ratio before moving on to the main phase of data preprocessing. This is done to avoid data leakage that may result to overfitting.

The first preprocessing technique is data or feature encoding since the dataset contains categorical data. To convert the categorical into numerical values, Label Encoding, Ordinal Encoding, and One Hot Encoding are used. Data encoding is applied first on the train set, and then on the test set. The second technique to be performed is data imputation since the dataset contains missing values. Data that are *Not Missing At Random (NMAR)* are replaced with 0 while those that are *Missing At Random (MAR)* are filled in using Single Imputation and MICE. Single Imputation is used for the categorical variables and MICE for the continuous variables. Again, data imputation is applied first on the train set, and then on the test set. Then, the imputed train dataset and test set (as necessary) will further undergo preprocessing such as oversampling, feature selection, and feature scaling using SMOTE, RFECV, and Standard Scaling respectively to produce more datasets for modeling. This is done to see if handling class imbalance, reducing dimensionality, and standardizing the data significantly improves the performance of the models. In total, there are 16 preprocessed datasets (1 imputed dataset + 15 SMOTE/RFECV/StandardScaler

dataset) and 44 trained models.

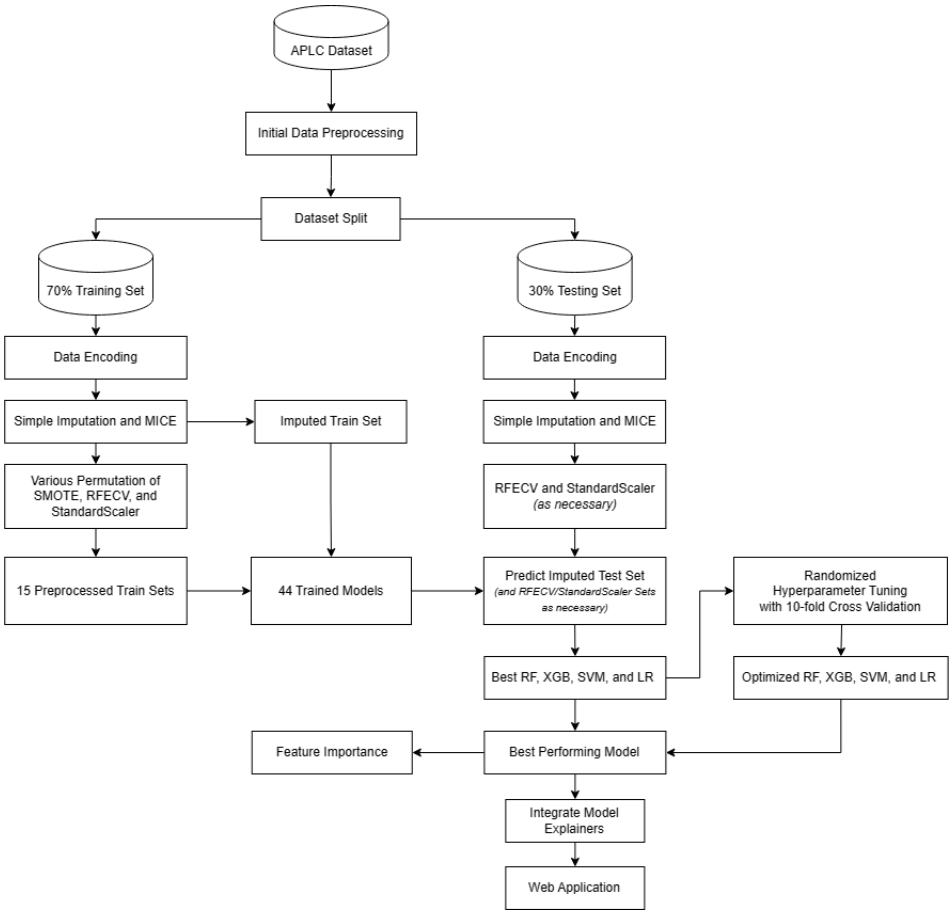


Figure 4: Schematic diagram of the workflow of web application development with model explainers.

Next, the trained models (RF, XGB, SVM, and LR) predict the test set and will be evaluated using the performance metrics such as accuracy, precision, recall, f1 score, and AUROC. The best four models RFbest, XGBbest, SVMbest, and LRbest models are selected to undergo Randomized Hyperparameter Tuning with 10−fold Cross Validation for further optimization. Then, the best models are compared to the optimized models. The model with the highest metrics is the best performing model and will be used with the LIME model explainer. LIME will use the best performing model to show the prediction probability and feature contributions. Moreover, feature importance is extracted from the best model.

Lastly, the best performing model and LIME is integrated to the web application. The web application provides a user interface to allow the physicians to input patient health data in a form and view the mortality risk prediction results along with the explanations.

B..3 Use-Case Diagram

The Use-Case Diagram shown in Figure 5 presents all functionalities of the web application that a physician can perform. The physician (especially rheumatologists) should be able to input patient health data in a form and view the prediction results upon submission. Results include the mortality risk prediction (*'Yes'* if the patient is at risk, otherwise *'No'*) and explanations about the most contributing features to the outcome.

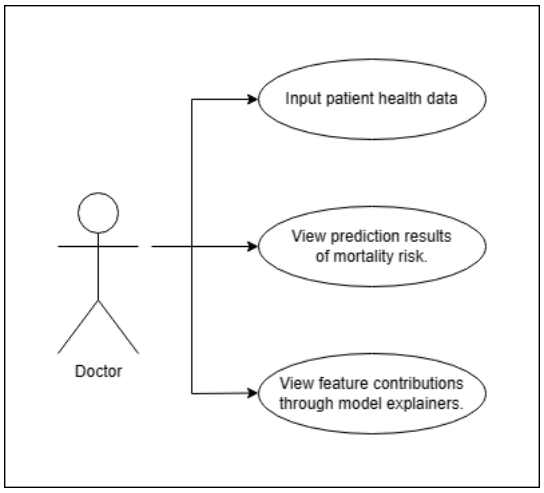


Figure 5: Use-Case Diagram

C. System Architecture

The Python programming language is used for data assessment, preprocessing, and the whole modeling process (training, testing, predicting and showing explanations). The *scikit-learn* and *imbalanced-learn* library, which are Python-based modules, pro-

vide useful packages for machine learning such as preprocessing, classification, model selection, performance evaluation, and hyperparameter tuning.[49] The Python *pandas* and *numpy* library also provides useful tools for machine learning studies primarily in data analysis and manipulation. Moreover, *matplotlib* and *seaborn* were used for visualizations while *joblib, pickle,* and *dill* were imported to save the pipelines of the final models.

On the other hand, the web application containing the user interface for the final prediction system is implemented using the Django framework—a web framework also built on the Python programming language—for the backend, and HTML and CSS for the frontend.

D. Technical Architecture

The deployment of the web application in a server is not within the scope of the study. Hence, the system is run through the *localhost* of the computer. Listed below are the minimum system requirements to run the system:

1. Any operating system but preferably Windows OS
2. Any internet browser (Firefox, Chrome, Edge, etc.)
3. Must have Python 3 installed
4. Must have Django and other dependencies installed

However, if the web application should be deployed in the future, the system will be accessible via a link on any platform (computer or smartphone) as long as a web browser is installed on the device. For this, the minimum system requirements are:

1. Any internet browser (Firefox, Chrome, Safari, etc.)
2. Stable internet connection

V. Results

A. Dataset

The original dataset contains 411 columns (variables) and a total of 42,355 rows (patient visits) from the 4,106 patients enrolled in the cohort—which means that the dataset contains the data of each visit of each patient. Upon observing the target variable (i.e., *deceased_Pt*) which indicates whether the patient have died or was still alive after the last visit, it was found that there were 787 death instances, among of which are 316 deaths directly caused by SLE. The latter was chosen as the final set of data to be used in the study. As for the variables, the dataset includes columns for individual criterions (e.g., SLICC and ACR) but also provides separate columns for the summary of the other variables (e.g., SLICC score and ACR score). Redundant columns are also present such as in medication, there are two columns for each medicine—one for the prescription (*Yes* if prescribed, otherwise *No*) and one for the dosage. For the two cases, the latter along with the other variables were selected for use in modeling as they provide more concise information and more distinct values. After filtering out the irrelevant rows according to the objectives of the study and dropping the unnecessary features as deemed by the related literature, the dataset now has an initial size of 316 rows by 84 columns.

To further reduce the dimensions of the dataset, the percentage and type of missingness in each feature that consists missing values was identified as shown in Table 1. A total of 17 columns (highlighted in red) that contain more than 20% of missing data that are MAR were dropped from the dataset. Data that are NMAR were replaced with a value of 0. Moreover, the number of unique values per variable was also distinguished. Three features particularly under the medication category contains only 1 distinct value and was removed from the dataset. On the other hand, it was found that the target variable is also comprised of only 1 unique value. It was

Feature	Missingness Percentage	Type
Visit Duration	0.126582	NMAR
Ethnicity	0.003165	MAR
SLE Family History	0.003165	MAR
Educational Level	0.003165	MAR
Smoking Status	0.003165	MAR
Creatinine	0.101266	MAR
eGFR	0.243671	MAR
Albumin	0.395570	MAR
CRP	0.601266	MAR
C3	0.155063	MAR
C4	0.509494	MAR
Haemoglobin	0.091772	MAR
White Cell Count	0.091772	MAR
Platelets	0.094937	MAR
Neutrophils	0.117089	MAR
Lymphocytes	0.117089	MAR
Erythrocyte Sedimentation Rate	0.250000	MAR
Anti dsDNA Result	0.167722	MAR
Urine Protein-Creatinine Ratio	0.645570	MAR
Urine WBC	0.202532	MAR
Urine RBC	0.202532	MAR
Urine Comments	0.680380	MAR
Time-Adjusted Mean SLEDAI	0.025316	NMAR
AMS	0.126582	NMAR
PGA	0.034810	MAR
Time-Adjusted Mean PGA	0.025316	NMAR
AM-PGA	0.126582	NMAR
Time-Adjusted Mean Prednisolone	0.126582	NMAR
SF36v2 PF (NBS)	0.721519	MAR
SF36v2 RP (NBS)	0.721519	MAR
SF36v2 BP (NBS)	0.721519	MAR
SF36v2 GH (NBS)	0.721519	MAR
SF36v2 VT (NBS)	0.721519	MAR
SF36v2 SF (NBS)	0.721519	MAR
SF36v2 RE (NBS)	0.721519	MAR
SF36v2 MH (NBS)	0.721519	MAR
SF36v2 PCS	0.721519	MAR
SF36v2 MCS	0.721519	MAR
LLDAS All Categories	0.056962	MAR
LLDAS Percentage	0.126582	NMAR
Time-Adjusted Mean LLDAS	0.025316	NMAR

Table 1: Features with missing data.

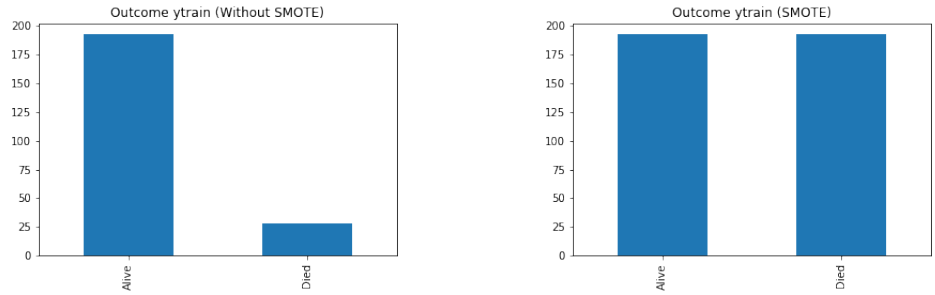
observed that when the patient have died, he/she was identified as dead (i.e., having a *"Yes"* value in *deceased_Pt*) on all of his/her visits. However, it is assumed in the study that only the last visit implies the patient's death, hence the patient should still be alive on the previous visits. To solve this, feature engineering was applied on *deceased_Pt* so that only the last visit of the patient has a *"Yes"* value while the other previous visits have a *"No"* value as shown in Table 2. After performing the initial preprocessing, the dataset now has a size of 316 rows by 64 columns (21 categorical,

43 continuous) and is ready for split.

ID_new	visit_no	deceased_Pt (Before)	deceased_Pt (After)
1027	1	Yes	No
1027	2	Yes	No
1027	3	Yes	No
1027	4	Yes	No
1027	5	Yes	No
1027	6	Yes	No
1027	7	Yes	No
1027	8	Yes	No
1027	9	Yes	No
1027	10	Yes	Yes

Table 2: Patient 1027's *deceased_Pt* attributes before and after feature engineering.

The resulting dataset was split into 70% train set and 30% test set. Feature encoding was performed first to convert categorical values into numerical data. Label Encoder was used for the target variable and predictors containing nominal data while Ordinal Encoder was used on the other predictors containing ordinal data. Specifically, One Hot Encoder was used for the variable *country* as it has 7 unique values. This resulted to 7 new columns while dropping the original *country* column—creating the final size of 316 rows by 69 columns for both the train set and test set. Data imputation was applied next to fill in data that are MAR and to prepare the dataset for modeling. Single Imputation was used to replace missing values with the most frequent value on the categorical features while MICE was used to fill in missing data on the continuous variables.



(a) Before SMOTE (b) After SMOTE

Figure 6: Total number of *Alive/Dead* instances of the target variable.

Three preprocessing techniques such as data augmentation, feature selection, and

feature scaling were further applied on the imputed dataset to create more variations for modeling. Particularly, SMOTE was used to handle the imbalance present on the dataset—creating new synthetic data to oversample the minority class. Figure 6 shows the relationship of the minority and majority class in the target variable before and after performing SMOTE respectively. Meanwhile, RFECV was applied to reduce the number of input variables—this process decreases the computational cost for the models and removes the noise in the data. Additionally, Standard Scaling was performed to standardize the data so that extreme instances, and/or huge gaps between values in the dataset would not have much weight on how the model would interpret such data. All three algorithms mentioned above can potentially increase the performance of machine learning models. Hence, all possible arrangements of preprocessing were considered in the study to see which method or order of methods yields the best performing model. As shown in Table 3, a total of 16 different datasets were produced for use in the modeling process.

SI and MICE only		
SMOTE only	RFECV only	Standard Scaler only
SMOTE-RFECV	RFECV-SMOTE	Standard Scaler-SMOTE
SMOTE-RFECV-Standard Scaler	RFECV-SMOTE-Standard Scaler	Standard Scaler-SMOTE-RFECV
SMOTE-Standard Scaler	RFECV-Standard Scaler	Standard Scaler-RFECV
SMOTE-Standard Scaler-RFECV	RFECV-Standard Scaler-SMOTE	Standard Scaler-RFECV-SMOTE

Table 3: Different variations of the preprocessed datasets

B. Model Performance

The models Random Forest, Extreme Gradient Boosting, Support Vector Machine, and Logistic Regression were trained and tested on the 16 different dataset variations, hence producing a total of 64 differently configured models. For the performance evaluation, various scoring metrics were used such as accuracy, precision, recall, f1-score, and AUROC. Other metrics were utilized to capture the predictive power of the model in all aspects rather than just the accuracy that only evaluates the model's ability to predict true values which sometimes may just be a result of

Random Forest					
Dataset	**Accuracy**	**Precision**	**Recall**	**F1-Score**	**AUROC**
SI and MICE	0.873684	0.699438	0.606928	0.631783	0.606928
SMOTE	0.905263	0.81737	0.696285	0.736842	0.696285
RFECV	0.873684	0.704023	0.64257	0.664706	0.64257
StandardScaler	0.894737	0.825549	0.618976	0.658764	0.618976
Extreme Gradient Boosting					
Dataset	**Accuracy**	**Precision**	**Recall**	**F1-Score**	**AUROC**
SI and MICE	0.863158	0.675711	0.636546	0.652015	0.636546
SMOTE	0.926316	0.827861	0.850904	0.838788	0.850904
RFECV	0.863158	0.663149	0.600904	0.619883	0.600904
StandardScaler	0.852632	0.652941	0.630522	0.640152	0.630522
Support Vector Machine					
Dataset	**Accuracy**	**Precision**	**Recall**	**F1-Score**	**AUROC**
SI and MICE	0.873684	0.436842	0.5	0.466292	0.5
SMOTE	0.578947	0.598889	0.723394	0.521169	0.723394
RFECV	0.873684	0.436842	0.5	0.466292	0.5
StandardScaler	0.873684	0.436842	0.5	0.466292	0.5
Logistic Regression					
Dataset	**Accuracy**	**Precision**	**Recall**	**F1-Score**	**AUROC**
SI and MICE	0.863158	0.663149	0.600904	0.619883	0.600904
SMOTE	0.821053	0.622917	0.648092	0.633038	0.648092
RFECV	0.873684	0.695055	0.571285	0.590517	0.571285
StandardScaler	0.873684	0.704023	0.64257	0.664706	0.64257

Table 4: Performance of the four models on the four base datasets

random guessing. Particularly, AUROC was one of the main metrics used as it considers both the True Positive Rate (TPR) and False Positive Rate (FPR). As shown in Table 4, the SMOTE, RFECV, and StandardScaler datasets have varying effects on the performance of the models compared to the imputed dataset. Generally, the recall and AUROC of all four models (including the other metrics for XGBoost) on the SMOTE dataset is higher than the imputed dataset. On the other hand, the four models did not perform any better on the RFECV- and StandardScaler-only datasets. Moreover, the performance of RF, XGB, SVM, and LR on the other 12 variations of preprocessed datasets are shown in Tables 5, 6, 7, and 8 respectively. Highlighted in yellow are the best metrics for each of the four models. For XGBoost, the best metrics were obtained when SMOTE was performed first, then RFECV and Standard Scaler for SVM. For RF and LR, the best metrics were achieved when StandardScaler was applied first, then SMOTE, and then RFECV. It is observable that the SMOTE-only configured XGBoost has the highest scores for all metrics while the other three best models also include SMOTE in their configurations. Hence, it can be inferred that SMOTE is the most effective on increasing the performance of the models.

Random Forest					
Dataset	**Accuracy**	**Precision**	**Recall**	**F1-Score**	**AUROC**
SMOTE-RFECV	0.905263	0.81737	0.696285	0.736842	0.696285
SMOTE-RFECV-StandardScaler	0.536842	0.572898	0.663655	0.481647	0.663655
SMOTE-StandardScaler	0.368421	0.558571	0.602912	0.356369	0.602912
SMOTE-StandardScaler-RFECV	0.484211	0.561404	0.633534	0.442448	0.633534
RFECV-SMOTE	0.821053	0.622917	0.648092	0.633038	0.648092
RFECV-SMOTE-StandardScaler	0.747368	0.582746	0.641566	0.588745	0.641566
RFECV-StandardScaler	0.873684	0.704023	0.64257	0.664706	0.64257
RFECV-StandardScaler-SMOTE	0.810526	0.627706	0.677711	0.64375	0.677711
StandardScaler-SMOTE	0.873684	0.708824	0.678213	0.691558	0.678213
StandardScaler-SMOTE-RFECV	0.894737	0.764706	0.725904	0.742965	0.725904
StandardScaler-RFECV	0.873684	0.699438	0.606928	0.631783	0.606928
StandardScaler-RFECV-SMOTE	0.894737	0.764706	0.725904	0.742965	0.725904

Table 5: Performance metrics of RF on the other 12 preprocessed datasets

Extreme Gradient Boosting					
Dataset	**Accuracy**	**Precision**	**Recall**	**F1-Score**	**AUROC**
SMOTE-RFECV	0.894737	0.762263	0.832831	0.790564	0.832831
SMOTE-RFECV-StandardScaler	0.347368	0.55465	0.590863	0.3385	0.590863
SMOTE-StandardScaler	0.431579	0.550342	0.603414	0.402377	0.603414
SMOTE-StandardScaler-RFECV	0.452632	0.573558	0.651104	0.424511	0.651104
RFECV-SMOTE	0.831579	0.649525	0.689759	0.664903	0.689759
RFECV-SMOTE-StandardScaler	0.705263	0.578205	0.653112	0.572072	0.653112
RFECV-StandardScaler	0.842105	0.614341	0.588855	0.598478	0.588855
RFECV-StandardScaler-SMOTE	0.757895	0.604286	0.683233	0.614026	0.683233
StandardScaler-SMOTE	0.863158	0.694184	0.707831	0.700606	0.707831
StandardScaler-SMOTE-RFECV	0.884211	0.738743	0.755522	0.746667	0.755522
StandardScaler-RFECV	0.863158	0.675711	0.636546	0.652015	0.636546
StandardScaler-RFECV-SMOTE	0.873684	0.719136	0.749498	0.732645	0.749498

Table 6: Performance metrics of XGBoost on the other 12 preprocessed datasets

Support Vector Machine					
Dataset	**Accuracy**	**Precision**	**Recall**	**F1-Score**	**AUROC**
SMOTE-RFECV	0.621053	0.548246	0.60492	0.511429	0.60492
SMOTE-RFECV-StandardScaler	0.852632	0.696248	0.773092	0.722917	0.773092
SMOTE-StandardScaler	0.663158	0.592105	0.700301	0.565714	0.700301
SMOTE-StandardScaler-RFECV	0.810526	0.641667	0.713353	0.661788	0.713353
RFECV-SMOTE	0.557895	0.514825	0.533133	0.456699	0.533133
RFECV-SMOTE-StandardScaler	0.747368	0.598384	0.677209	0.605263	0.677209
RFECV-StandardScaler	0.905263	0.951087	0.625	0.674286	0.625
RFECV-StandardScaler-SMOTE	0.8	0.602187	0.636044	0.613408	0.636044
StandardScaler-SMOTE	0.336842	0.58	0.620482	0.332106	0.620482
StandardScaler-SMOTE-RFECV	0.863158	0.685606	0.672189	0.678469	0.672189
StandardScaler-RFECV	0.884211	0.941489	0.541667	0.54585	0.541667
StandardScaler-RFECV-SMOTE	0.863158	0.685606	0.672189	0.678469	0.672189

Table 7: Performance metrics of SVM on the other 12 preprocessed datasets

Logistic Regression					
Dataset	**Accuracy**	**Precision**	**Recall**	**F1-Score**	**AUROC**
SMOTE-RFECV	0.789474	0.61	0.665663	0.624209	0.665663
SMOTE-RFECV-StandardScaler	0.694737	0.588954	0.682731	0.577778	0.682731
SMOTE-StandardScaler	0.715789	0.597782	0.694779	0.59421	0.694779
SMOTE-StandardScaler-RFECV	0.726316	0.616815	0.736446	0.615504	0.736446
RFECV-SMOTE	0.810526	0.611946	0.642068	0.623016	0.642068
RFECV-SMOTE-StandardScaler	0.621053	0.578841	0.676205	0.534314	0.676205
RFECV-StandardScaler	0.873684	0.699438	0.606928	0.631783	0.606928
RFECV-StandardScaler-SMOTE	0.757895	0.57175	0.611948	0.578267	0.611948
StandardScaler-SMOTE	0.831579	0.649525	0.689759	0.664903	0.689759
StandardScaler-SMOTE-RFECV	0.821053	0.651316	0.719378	0.672347	0.719378
StandardScaler-RFECV	0.863158	0.675711	0.636546	0.652015	0.636546
StandardScaler-RFECV-SMOTE	0.8	0.632883	0.707329	0.651612	0.707329

Table 8: Performance metrics of LR on the other 12 preprocessed datasets

To further increase the predictive power of the best models, random search hyperparameter tuning with 10-fold cross-validation was performed. Table 9 shows the list of hyperparameters and corresponding sets of values that were configured for each model. The comparisons of the metrics between the best models and the optimized models are shown in Table 10. It is evident that the scores of the fine-tuned models were not any higher than the models with default parameters. Highlighted in green is the best performing model—XGBoost configured on the SMOTE dataset—having a 92.63% accuracy, 82.79% precision, 85.1% recall, 83.89% f1-score, and 85.1% AUROC.

Random Forest	
Hyperparameter	**Values**
n_estimators	20–500
max_depth	5, 10, 15
max_features	0.1, 1, 3, 5
min_samples_leaf	20–200
Extreme Gradient Boosting	
Hyperparameter	**Values**
objective	'binary:logistic', 'reg:gamma'
n_estimators	100–500
max_depth	3, 5, 7, 9, 10
eta	0.01, 0.1, 0.2, 0.5
reg_alpha	0, 0.01, 0.05, 0.1, 0.5, 1
gamma	0.01, 0.1, 1, 3
Support Vector Machine	
Hyperparameter	**Values**
C	0.01–10
kernel	'rbf', 'poly', 'sigmoid', 'linear'
gamma	0.01–10
Logistic Regression	
Hyperparameter	**Values**
C	0.01–10
penalty	'l1', 'l2'
solver	'lbfgs','liblinear'
max_iter	500, 1000, 5000, 10000

Table 9: List of hyperparameters with their corresponding sets of values for each of the four models

Model	**Dataset**	**Accuracy**	**Precision**	**Recall**	**F1-Score**	**AUROC**
RF-best	StandardScaler-SMOTE-RFECV	0.894737	0.764706	0.725904	0.742965	0.725904
RF-tuned	StandardScaler-SMOTE-RFECV	0.789474	0.61	0.665663	0.624209	0.665663
XGB-best	**SMOTE only**	**0.926316**	**0.827861**	**0.850904**	**0.838788**	**0.850904**
XGB-tuned	SMOTE only	0.894737	0.762263	0.832831	0.790564	0.832831
SVM-best	SMOTE-RFECV-StandardScaler	0.852632	0.696248	0.773092	0.722917	0.773092
SVM-tuned	SMOTE-RFECV-StandardScaler	0.768421	0.624861	0.7249	0.638158	0.7249
LR-best	StandardScaler-SMOTE-RFECV	0.821053	0.651316	0.719378	0.672347	0.719378
LR-tuned	StandardScaler-SMOTE-RFECV	0.694737	0.603571	0.718373	0.590091	0.718373

Table 10: Performance comparison of the default best models and fine-tuned models

C. Best Performing Model

The confusion matrix of the best performing model, XGBoost, is shown in Table 11. The SMOTE-configured XGBoost model was able to accurately predict 88 true labels out of the 95 values in the test set—only misclassifying 7 labels. This gives the XGBoost model an accuracy of 92.63%, a sensitivity of 75%, and a specificity of 95.18% which is considered good–excellent performance in machine learning.

		Prediction Outcome		
		0	1	**total**
Actual Value	0	79	4	83
	1	3	9	12
	total	82	13	

Table 11: Confusion Matrix of the best XGBoost model

Moreover, feature importance was also extracted from the best XGBoost to identify which particular features are the most critical to the prediction calculation of the model. For the importance, two different metrics were used, gain and frequency. Gain indicates the contribution of a feature on the model by calculating the improvement in accuracy that it brings to the branches it is on for each tree—the higher the value, the more important the feature is in generating a prediction.[50] As shown in Figure 7, the variable *SLICC_atleast1_clinical* is the most important and most contributing feature to the prediction process of the model when gain was used, followed by *SF_no_visit* and others. Frequency, on the other hand, simply specifies the number of times the variable was split on in the model to arrive at a prediction. For frequency score as presented in Figure 8, it appears that *urine_WBC* is the most important feature, followed by *IS_noyes* and *PGA* which are also the only two variables that

belongs to the top 10 of most importance when gain was used. It is revealed in the next section that the feature contributions produced by the LIME interpreter are highly similar to those of the frequency importance.

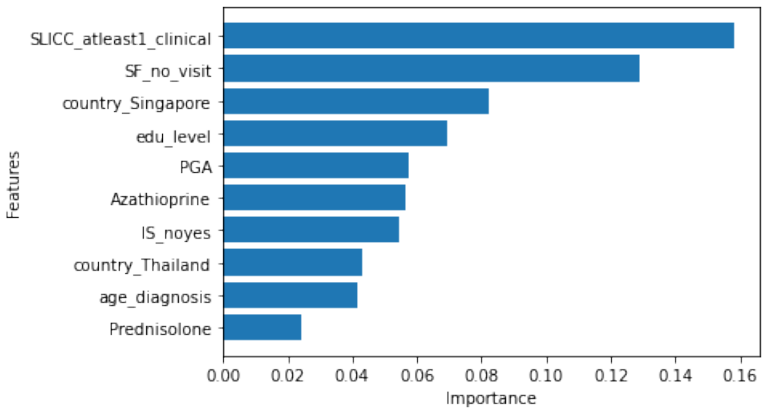


Figure 7: 10 most important features for the best XGBoost based on gain.

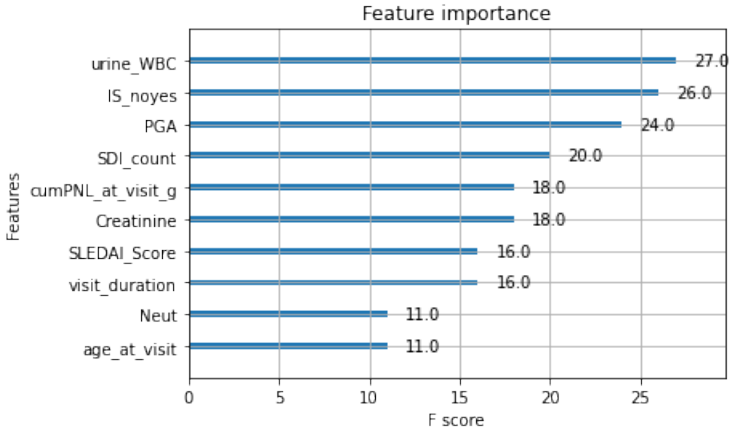


Figure 8: 10 most important features for the best XGBoost based on frequency.

Furthermore, Table 12 lists down all features and their corresponding importance (based on gain) to the model. Since gain implies the feature's relative contribution to the accuracy, the 22 features of 0 importance—the ones highlighted in red—were removed from the dataset. This reduced the number of input variables to 47 which is used in the final prediction system. To ensure that the performance of the XGBoost model did not decrease after removing the unimportant features, the model was trained and tested again on the feature-selected SMOTE dataset. The resulting met-

rics were the same with that of the original SMOTE-configured XGBoost—a 92.63% accuracy, 82.79% precision, 85.1% recall, 83.89% f1-score, and 85.1% AUROC.

Feature	Importance	Feature	Importance	Feature	Importance
visit_duration	0.004667617	Plt	0.011242542	Mycophenolate	0.008417905
GDP_3cat	0	Neut	0.009357261	Mycophenolic_acid	0
age_at_visit	0.011524904	Lymph	0.003819145	MMF_MPA_noyes	0.011305996
gender	0	anti_dsDNA_result	0.002046324	Ciclosporin	0
ethnicity_bin	0	urine_WBC	0.009830874	Cyclophosphamide	0
SLE_fam_history	0	urine_RBC	0.006890139	IS_noyes	0.054636296
edu_level	0.069242433	SLEDAI_Score	0.017226687	Rituximab_noyes	0
smoking_status	0	TAM_SLEDAI	0.003510811	biologics_noyes	0
age_diagnosis	0.041673217	AMS_visit	0.000449982	Belimumab_noyes	0
disease_duration_at_visit	0.013609252	PGA	0.057330307	SDI_TOTALscore_CF	0.005039091
ACR_Immuno	0	TAM_PGA	0.008077955	overall_damage_accrual_score	0.019220246
ACR_ANA	0	AM_PGA_visit	0.003788004	damage_accrual_noyes	0
ACR_number	0.001223248	SF_no_visit	0.128730282	SDI_count	0.007615144
SLICC_nephritis_plus_immuno	0.001062575	any_flare_no_visit	0.003879901	LLDAS_All_5cat	0
SLICC_atleast1_clinical	0.158224612	MF_no_visit	0.006533192	LLDAS_percent_v	0.001052852
SLICC_atleast1_immuno	0	TAM_PNL_visit	0.002727249	TAM_LLDAS	0.008786581
SLICC_clinical_Immuno	0.012690044	Prednisolone	0.024309998	country_Australia	0
SLICC_4more_clinical_Immuno	0	cumPNL_at_visit_g	0.020691073	country_Indonesia	0.003850699
SLICC_number	0.007565049	Hydroxychloroquine	0.008062962	country_Malaysia	0.005653972
Creatinine	0.007598653	Chloroquine	0	country_Philippines	0
C3	0.002486271	AM_noyes	0.021900916	country_Singapore	0.082101181
Hb	0.002104514	Methotrexate	0	country_Taiwan	0
WCC	0.008599261	Azathioprine	0.056570359	country_Thailand	0.043072395

Table 12: Feature importance of the best XGBoost model

D. Model Explainer

Since XGBoost performed best among RF, LR, and SVM, the model (particularly the feature-selected SMOTE-configured model) is integrated with LIME to provide explanations about feature contributions, thus uncovering the blackbox to see how the model arrived with its prediction. Sample model explanations are shown in Figures 9 and 10 where predictions of XGBoost on two different test cases are explained by LIME. The red bars or the ones on the left side from the middle line show the feature contributions on the probability that the patient is predicted to be in class 0 - that is the patient will likely live. On the other hand, the green bars or the ones on the right side from the middle line show the feature contributions on the probability that the patient is predicted to be in class 1 - that is the patient is likely at risk of mortality. It can be observed that the 10 most contributing features of both explanations are

almost similar—*urine_WBC* as the most contributing for the patient classified as class 0 while *SLEDAI_Score* for the class 1 patient.

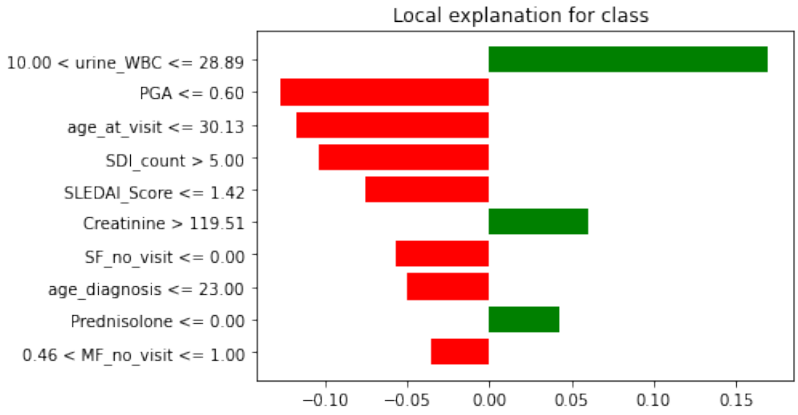


Figure 9: Feature contributions on the patient predicted as class 0.

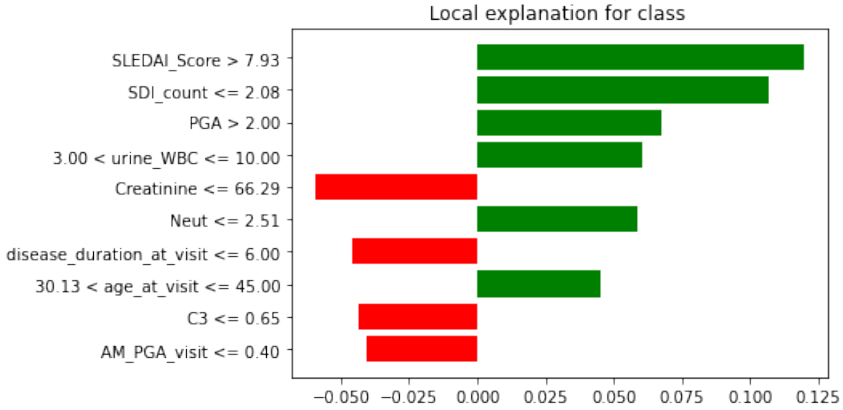


Figure 10: Feature contributions on the patient predicted as class 1.

E. Web Application

The web application built from Django framework is composed of 4 pages. The Home page or the landing page is the first interface that the user will face upon opening the application as shown in Figure 11. It contains the title of the study to give an idea on what the web app is all about. The 'Get Started' button in the Home page will redirect the user to the Dashboard page when clicked. The Dashboard page contains more information about the web application including the XGBoost model

and LIME as shown in Figure 12. It also contains a *Let's Predict* button which will redirect the user to the prediction page. The Prediction page, as seen in Figure 13, is where the user is prompt to input a total of 47 patient medical data comprised of demographics, pathology results, medication, and other SLE-related data. To see the results, the *Predict* is clicked and the application redirects to the Results page. Figure 14 shows a snapshot of the Results page which contains the prediction of XGBoost along with the explanations provided by LIME. In order to go back to the previous pages, the user may use the *Home* and *Dashboad* buttons on the navigation bar.

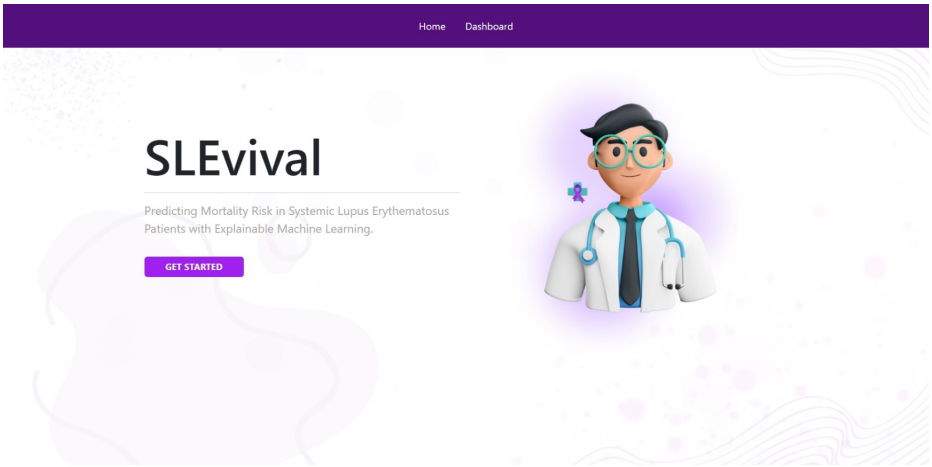


Figure 11: Home page of the SLEvival web application.

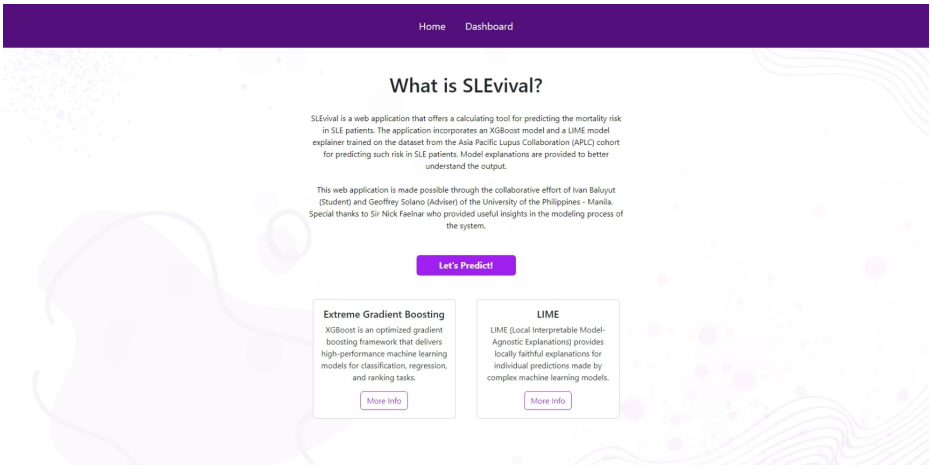


Figure 12: Dashboard page of the SLEvival web application.

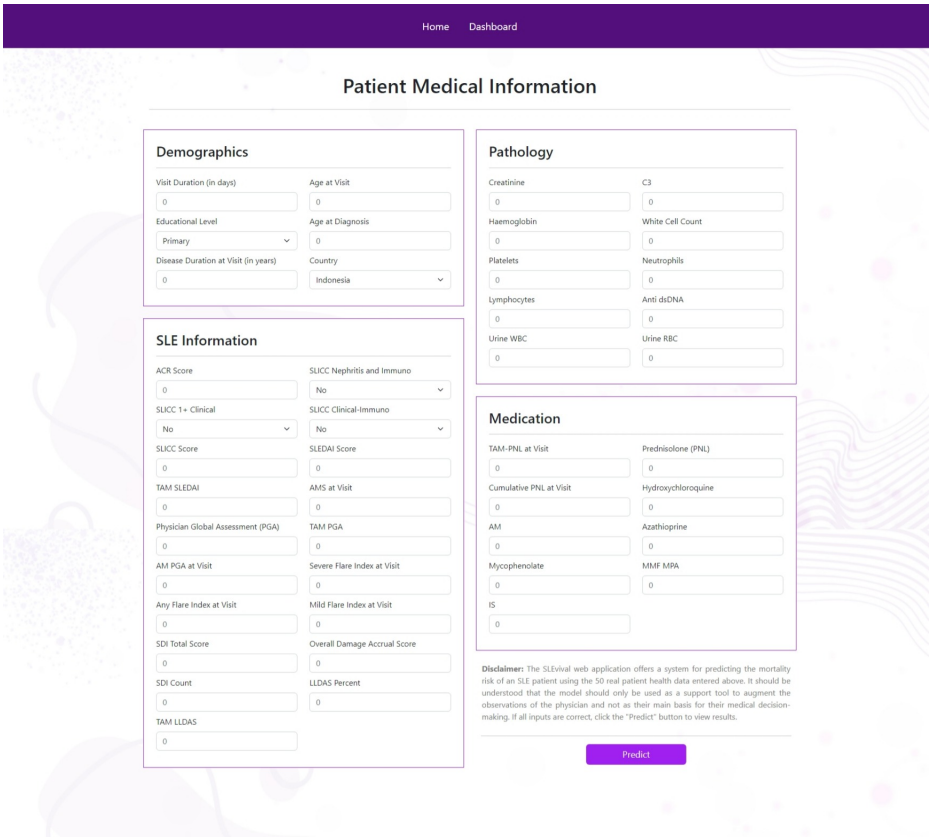


Figure 13: Prediction page of the SLEvival web application.

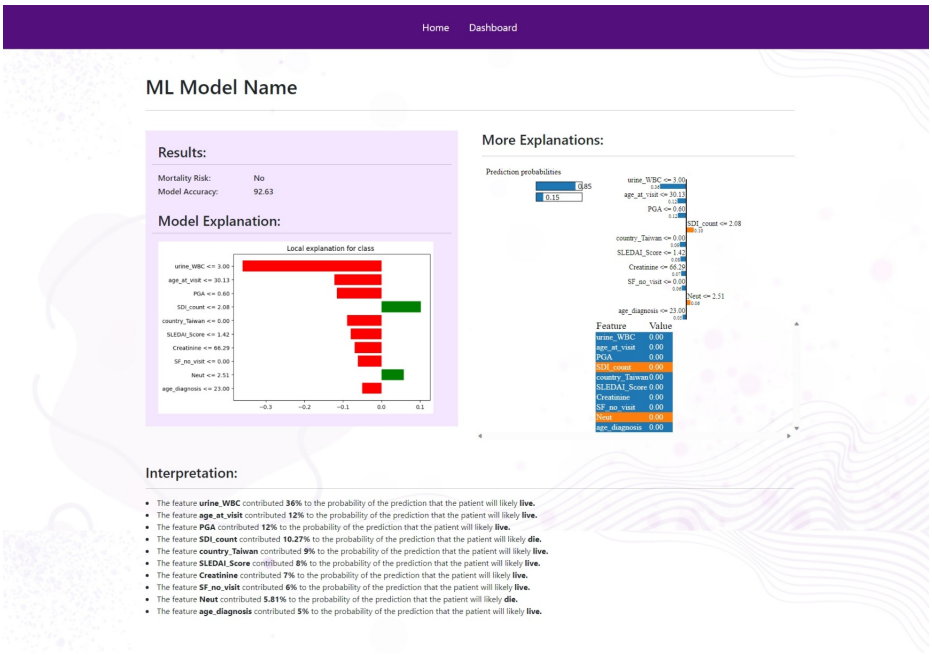


Figure 14: Results page of the SLEvival web application.

VI. Discussions

A. Objectives

At the end of this paper, all objectives of the study have been achieved. The APLC dataset was initially filtered to remove irrelevant data and unnecessary variables—reducing the size of the original dataset to 316 rows x 64 columns. Various preprocessing techniques such as data encoding (Label Encoder, Ordinal Encoder, and One Hot Encoder), data imputation (SI and MICE), data augmentation (SMOTE), feature selection (RFECV), and feature scaling (Standard Scaler) were applied to produce different datasets for modeling. Random Forest, Extreme Gradient Boosting, Support Vector Machine, and Logistic Regression were trained and tested on each of the 16 dataset variations—producing a total of 64 differently configured models. For performance evaluation, AUROC was the main metric looked upon together with accuracy, precision, recall, and f1-score. It was found that SMOTE was the most effective among the other two preprocessing algorithms, RFECV and StandardScaler. The model with the highest metrics for each of the four models underwent random search hyperparameter tuning with 10-fold cross validation to further increase their performance. The best performing model, XGBoost configured on the SMOTE dataset, among the other best and optimized models was then used in LIME to provide explanations about the contributions of the features on the prediction outcome. Moreover, feature importance was also extracted from the final XGBoost model where *SLICC_atleast1_clinical* was identified as the most critical feature in the prediction mechanism. Finally, LIME and XGBoost were integrated in the web application where a user is prompt to input patient health data and view the results of the prediction firsthand.

By accomplishing the objectives mentioned above, the study was able to fulfill its significance in the medical field—to contribute a new machine learning approach

which is scarce in SLE studies and to provide an actual mortality risk prediction system that physicians can use as a support tool in their medical decision-making.

B. Challenges

The first challenge of the modeling process was deciding on what approach to use on the dataset. The original dataset contains over $40,000$ data entries, however an extreme imbalance is present. The approach used in this study is to extract all the patients who were identified as dead - having a class 1 or *yes* in the target variable. Since each patient has multiple entries (i.e., multiple visits) in the dataset, each visit was considered as an individual entry or an individual patient. Then through feature engineering, modify the *deceasedPt* column such that the first to second to the last visit, the patient is alive, while on the last visit, the patient is dead. Here, we assume that the data from the last visit of the patient was the cause of the patient's death after that last visit. In this way, the model can see patterns such as changes in medication, changes in laboratory results, etc. as to why the patient is alive or dead - hence, being capable of predicting the mortality risk in SLE patients. However, time-related causes such as instances where the changes in the data from the $5th$ visit may have contributed in the mortality of the patient, was not considered in the current approach.

The next challenge was on how to handle the missing values. The MICE algorithm used in this study, that is Iterative Imputer with Random Forest Regressor as the estimator, takes a lot of computational time even with small and shortened dataset. On datasets containing thousands of entries, the algorithm takes several hours to complete its execution and can even last a day. Moreover, the imputing the dataset through the algorithm didn't have positive effect on the performance of the models. This was observed when the XGBoost was initially ran on the original dataset and then trained and tested on the imputed dataset. The XGBoost performed better

on the former than the latter. K-Nearest Neighbor Imputation was also applied in this study but did not produce any better results. Another imputation technique, MissForest, was used and gave slightly better results. However, it was not continued to use in the study as there were conflicts in libraries and dependencies—when the version of the library was changed according to MissForest, the other algorithms failed to run, and vice versa.

Lastly, it was observed that the SMOTE introduces new values to the dataset even if the data is categorical. For example, if the feature contains only 0 and 1 values, the SMOTE will produce values between 0 and 1 which does not make sense since the data is categorical. To solve this, values of the categorical features were rounded into a whole number after performing SMOTE. Meanwhile, a variation of SMOTE called SMOTENC, has an inner mechanism that handles categorical data separately. However, the results from the dataset produced from SMOTENC were not better than the SMOTE - hence, the latter was used in the final modeling.

Moreover, the feature selection algorithm, RFECV with Random Forest Classifier as the estimator, did not have much effect on reducing the noise on the dataset neither on increasing the performance of the models when it was put first in the order of preprocessing. While Pearson correlation feature selection is the easiest and most common feature selection algorithm (i.e., drop the predictors with low correlation to the target or high correlation to the other predictors), it can not be used as it only works for continuous variable whereas the dataset in the study is a mix of categorical and continuous features.

C. Contributions

Most of the existing literature in SLE studies, such as the association of demographics and health characteristics to the patient's mortality risk, are purely statistical analyses. And while there were machine learning approach—although very limited—

in SLE, most of these were research on diagnosis and susceptibility to the disease. This current study bridges the gap between the two sides of the existing works in SLE by incorporating the technology of machine learning in the prediction of mortality risk in SLE.

Specifically, the study provides an actual prediction system that a physician can use firsthand to input patient health data and view the results real-time. The system is composed of an XGBoost model trained on the APLC dataset and a LIME model explainer to provide explanations—both of which are integrated on the final, user-facing web application.

VII. Conclusions

The study introduces a machine learning approach in predicting the mortality risk among SLE patients. Initial size of $42,355$ rows by 411 columns of the APLC dataset was reduced to 316 by 64 after filtering out irrelevant data and dropping unnecessary columns including variables with high percentage of missingness. Label, Ordinal, and One Hot Encoders were used to convert the 21 categorical features (including the target variable) into numerical data. Single Imputation and MICE was used to fill in missing values in the categorical and continuous variables respectively. Three preprocessing algorithms, SMOTE, RFECV, and Standard Scaler, were further applied on the imputed dataset and all possible permutations were considered—creating a total of 16 different variations of preprocessed datasets.

Four machine learning models, Random Forest, Extreme Gradient Boosting, Logistic Regression, and Support Vector Machine were trained and tested on the 16 dataset variations—developing a total of 64 differently configured models. After the initial evaluation of the models' performance on the 4 base datasets (imputed, SMOTE-only, RFECV-only, and Standard Scaler-only), it was found that SMOTE relatively increases the recall and AUROC (including the other metrics for XGBoost), while RFECV and Standard Scaler alone did not improve the performance of the models. Further evaluation on the remaining 12 dataset variations was done to see if there is an improvement on the models' performance when configured on various permutations of preprocessing. It appeared that XGBoost obtained the best metrics when SMOTE was performed first, then RFECV and Standard Scaler for SVM. On the other hand, RF and LR performed best when Standard Scaler was applied first, then SMOTE, and then RFECV. Additionally, random search hyperparameter tuning with 10-fold cross-validation was conducted on the four best models but have failed to increase their performance.

At the end of the modeling process, the SMOTE-only-configured XGBoost ob-

tained the highest performance metrics among the other models—having a 92.63% accuracy, 82.79% precision, 85.1% recall, 83.89% f1-score, and 85.1% AUROC. Feature importance of the best XGBoost were obtained as well using two metrics, gain and frequency. For gain, *SLICC_atleast1_clinical* was identified as the most important feature while *urine_WBC* for frequency. A total of 22 variables of 0 importance based on gain was removed on the final dataset—reducing the number of input variables of the model to 47. Moreover, the top important features based on frequency were highly similar to the most contributing features on the sample explanations provided by the LIME interpreter. Finally, a web application containing the final prediction system was built on the Django framework. The web application integrated with XGBoost and LIME consists of four pages: home, dashboard, prediction, and results—allowing a user to input a total of 47 real patient health data and view the XGBoost prediction outcome (i.e., *'Yes'* if the patient is at risk of mortality, otherwise *'No'*) along with the LIME explanations which indicates the contribution of each feature to the prediction firsthand.

VIII. Recommendations

Although a machine learning model (i.e., XGBoost) trained on the produced datasets achieved high performance with—what are considered as—excellent metrics, it is still necessary to explore other algorithms for each preprocessing methods to possibly further increase the performance of the models. Future works may consider other approach on how to use the vast data provided by the APLC dataset such as including time (if possible) in the analysis of mortality risk. Datasets from other cohorts can also be used instead of the APLC cohort to obtain more information and more data about the mortality of SLE patients. Future studies can also consider using other data imputation technique such as MissForest, since the Iterative Imputer with Random Forest Regressor used in this study takes a lot of computational time even with the shortened dataset. Other feature selection algorithms may also be explored since the RFECV which is used in this study didn't reduced the dimensionality of the dataset and didn't have much contribution to the performance of the models. Moreover, to handle data imbalance, future researches can try applying class weights instead of using SMOTE to oversample the minority class.

On the other hand, future works may also consider using other model explainers aside from LIME such as Shapley Values. This could provide entirely different explanations for the machine learning models used which can then be compared to the LIME results in this study.

IX. Bibliography

[1] "Systemic lupus erythematosus (sle)," Jul 2022.

[2] C. Rosario, L. Seguro, C. Vasconcelos, and Y. Shoenfeld, "Is there a cure for systemic lupus erythematosus?," *Lupus*, vol. 22, no. 5, pp. 417–421, 2013.

[3] E. F. Morand, T. Trasieva, A. Berglind, G. G. Illei, and R. Tummala, "Lupus low disease activity state (lldas) attainment discriminates responders in a systemic lupus erythematosus trial: post-hoc analysis of the phase iib muse trial of anifrolumab," *Annals of the rheumatic diseases*, vol. 77, no. 5, pp. 706–713, 2018.

[4] V. Golder, R. Kandane-Rathnayake, M. Huq, H. T. Nim, W. Louthrenoo, S. F. Luo, Y.-J. J. Wu, A. Lateef, S. Sockalingam, S. V. Navarra, *et al.*, "Lupus low disease activity state as a treatment endpoint for systemic lupus erythematosus: a prospective validation study," *The Lancet Rheumatology*, vol. 1, no. 2, pp. e95–e102, 2019.

[5] Y. Fan, J. Dong, Y. Wu, M. Shen, S. Zhu, X. He, S. Jiang, J. Shao, and C. Song, "Development of machine learning models for mortality risk prediction after cardiac surgery," *Cardiovascular Diagnosis and Therapy*, vol. 12, no. 1, p. 12, 2022.

[6] A. Ogunleye and Q.-G. Wang, "Xgboost model for chronic kidney disease diagnosis," *IEEE/ACM transactions on computational biology and bioinformatics*, vol. 17, no. 6, pp. 2131–2140, 2019.

[7] L. Ibrahim, M. Mesinovic, K.-W. Yang, and M. A. Eid, "Explainable prediction of acute myocardial infarction using machine learning and shapley values," *IEEE Access*, vol. 8, pp. 210410–210417, 2020.

[8] S. El-Sappagh, J. M. Alonso, S. Islam, A. M. Sultan, and K. S. Kwak, "A multilayer multimodal detection and prediction model based on explainable artificial intelligence for alzheimer's disease," *Scientific reports*, vol. 11, no. 1, pp. 1–26, 2021.

[9] R. Kandane-Rathnayake, V. Golder, W. Louthrenoo, Y.-H. Chen, J. Cho, A. Lateef, L. Hamijoyo, S.-F. Luo, Y.-J. J. Wu, S. V. Navarra, *et al.*, "Lupus low disease activity state and remission and risk of mortality in patients with systemic lupus erythematosus: a prospective, multinational, longitudinal cohort study," *The Lancet Rheumatology*, vol. 4, no. 12, pp. e822–e830, 2022.

[10] F. Rees, M. Doherty, M. J. Grainge, P. Lanyon, and W. Zhang, "The worldwide incidence and prevalence of systemic lupus erythematosus: a systematic review of epidemiological studies," *Rheumatology*, vol. 56, no. 11, pp. 1945–1961, 2017.

[11] S. S. Lim, C. G. Helmick, G. Bao, J. Hootman, R. Bayakly, C. Gordon, and C. Drenkard, "Racial disparities in mortality associated with systemic lupus erythematosus—fulton and dekalb counties, georgia, 2002–2016," *Morbidity and Mortality Weekly Report*, vol. 68, no. 18, p. 419, 2019.

[12] S. Bernatsky, J.-F. Boivin, L. Joseph, S. Manzi, E. Ginzler, D. Gladman, M. Urowitz, P. Fortin, M. Petri, S. Barr, *et al.*, "Mortality in systemic lupus erythematosus," *Arthritis & Rheumatism: Official Journal of the American College of Rheumatology*, vol. 54, no. 8, pp. 2550–2557, 2006.

[13] E. Y. Yen, M. Shaheen, J. M. Woo, N. Mercer, N. Li, D. K. McCurdy, A. Karlamangla, and R. R. Singh, "46-year trends in systemic lupus erythematosus mortality in the united states, 1968 to 2013: a nationwide population-based study," *Annals of internal medicine*, vol. 167, no. 11, pp. 777–785, 2017.

[14] G. S. Alarcón, G. McGwin, Jr, H. M. Bastian, J. Roseman, J. Lisse, B. J. Fessler, A. W. Friedman, and J. D. Reveille, "Systemic lupus erythematosus in three ethnic groups. viii. predictors of early mortality in the lumina cohort," *Arthritis Care & Research: Official Journal of the American College of Rheumatology*, vol. 45, no. 2, pp. 191–202, 2001.

[15] G. Bharath, P. Kumar, N. Makkar, P. Singla, M. Soneja, A. Biswas, and N. Wig, "Mortality in systemic lupus erythematosus at a teaching hospital in india: A 5-year retrospective study," *Journal of Family Medicine and Primary Care*, vol. 8, no. 7, p. 2511, 2019.

[16] D. R. Azizoddin, M. Jolly, S. Arora, E. Yelin, and P. Katz, "Patient-reported outcomes predict mortality in lupus," *Arthritis care & research*, vol. 71, no. 8, pp. 1028–1035, 2019.

[17] K. Franklyn, C. S. Lau, S. V. Navarra, W. Louthrenoo, A. Lateef, L. Hamijoyo, C. S. Wahono, S. Le Chen, O. Jin, S. Morton, *et al.*, "Definition and initial validation of a lupus low disease activity state (lldas)," *Annals of the rheumatic diseases*, vol. 75, no. 9, pp. 1615–1621, 2016.

[18] C. Sharma, W. Raymond, G. Eilertsen, and J. Nossent, "Association of achieving lupus low disease activity state fifty percent of the time with both reduced damage accrual and mortality in patients with systemic lupus erythematosus," *Arthritis care & research*, vol. 72, no. 3, pp. 447–451, 2020.

[19] M. J. Pontiveros, G. A. Solano, C. A. Tee, and M. L. Tee, "Explainable machine learning applied to single-nucleotide polymorphisms for systemic lupus erythematosus prediction," in *2020 11th International Conference on Information, Intelligence, Systems and Applications (IISA*, pp. 1–8, IEEE, 2020.

[20] J. H. Park, H. E. Cho, J. H. Kim, M. M. Wall, Y. Stern, H. Lim, S. Yoo, H. S. Kim, and J. Cha, "Machine learning prediction of incidence of alzheimer's disease using large-scale administrative health data," *NPJ digital medicine*, vol. 3, no. 1, pp. 1–7, 2020.

[21] Y. K. Singh, N. Sinha, and S. K. Singh, "Heart disease prediction system using random forest," in *International Conference on Advances in Computing and Data Sciences*, pp. 613–623, Springer, 2016.

[22] L. Yang, H. Wu, X. Jin, P. Zheng, S. Hu, X. Xu, W. Yu, and J. Yan, "Study of cardiovascular disease prediction model based on random forest in eastern china," *Scientific reports*, vol. 10, no. 1, pp. 1–8, 2020.

[23] K. Lin, Y. Hu, and G. Kong, "Predicting in-hospital mortality of patients with acute kidney injury in the icu using random forest model," *International journal of medical informatics*, vol. 125, pp. 55–61, 2019.

[24] E. Jamshidi, A. Asgary, N. Tavakoli, A. Zali, S. Setareh, H. Esmaily, S. H. Jamaldini, A. Daaee, A. Babajani, M. A. S. Kashi, *et al.*, "Using machine learning to predict mortality for covid-19 patients on day 0 in the icu," *Frontiers in digital health*, vol. 3, 2021.

[25] M. Li, X. Fu, and D. Li, "Diabetes prediction based on xgboost algorithm," in *IOP conference series: materials science and engineering*, vol. 768, p. 072093, IOP Publishing, 2020.

[26] C. Krittanawong, H. U. H. Virk, A. Kumar, M. Aydar, Z. Wang, M. P. Stewart, and J. L. Halperin, "Machine learning and deep learning to predict mortality in patients with spontaneous coronary artery dissection," *Scientific reports*, vol. 11, no. 1, pp. 1–10, 2021.

[27] K. Budholiya, S. K. Shrivastava, and V. Sharma, "An optimized xgboost based diagnostic system for effective prediction of heart disease," *Journal of King Saud University-Computer and Information Sciences*, 2020.

[28] F. Ceccarelli, M. Sciandrone, C. Perricone, G. Galvan, F. Morelli, L. N. Vicente, I. Leccese, L. Massaro, E. Cipriano, F. R. Spinelli, *et al.*, "Prediction of chronic damage in systemic lupus erythematosus by using machine-learning models," *PLoS One*, vol. 12, no. 3, p. e0174200, 2017.

[29] M. Pourhomayoun and M. Shakibi, "Predicting mortality risk in patients with covid-19 using machine learning to help medical decision-making," *Smart Health*, vol. 20, p. 100178, 2021.

[30] C. Hu, Q. Tan, Q. Zhang, Y. Li, F. Wang, X. Zou, and Z. Peng, "Application of interpretable machine learning for early prediction of prognosis in acute kidney injury," *Computational and Structural Biotechnology Journal*, vol. 20, pp. 2861–2870, 2022.

[31] F. Deshmukh and S. S. Merchant, "Explainable machine learning model for predicting gi bleed mortality in the intensive care unit," *Official journal of the American College of Gastroenterology— ACG*, vol. 115, no. 10, pp. 1657–1668, 2020.

[32] S. Muneer and M. A. Rasool, "Aa systematic review: Explainable artificial intelligence (xai) based disease prediction," *International Journal of Advanced Sciences and Computing*, vol. 1, no. 1, pp. 1–6, 2022.

[33] R. Kandane-Rathnayake, V. Golder, W. Louthrenoo, S.-F. Luo, Y.-J. Jan Wu, Z. Li, Y. An, A. Lateef, S. Sockalingam, S. V. Navarra, *et al.*, "Development of the asia pacific lupus collaboration cohort," *International journal of rheumatic diseases*, vol. 22, no. 3, pp. 425–433, 2019.

[34] A. P. Hassler, E. Menasalvas, F. J. García-García, L. Rodríguez-Mañas, and A. Holzinger, "Importance of medical data preprocessing in predictive modeling and risk factor discovery for the frailty syndrome," *BMC medical informatics and decision making*, vol. 19, no. 1, pp. 1–17, 2019.

[35] T. Firdose, "Understanding categorical encoding techniques: Ordinal, one-hot, and label encoding," May 2023.

[36] M. J. Azur, E. A. Stuart, C. Frangakis, and P. J. Leaf, "Multiple imputation by chained equations: what is it and how does it work?," *International journal of methods in psychiatric research*, vol. 20, no. 1, pp. 40–49, 2011.

[37] J. Brownlee, "Smote for imbalanced classification with python," Mar 2021.

[38] K. Choudhury, "Feature engineering - recursive feature elimination with cross-validation," Sep 2020.

[39] J. Brownlee, "How to use standardscaler and minmaxscaler transforms in python," Aug 2020.

[40] Q. Bi, K. E. Goodman, J. Kaminsky, and J. Lessler, "What is machine learning? a primer for the epidemiologist," *American journal of epidemiology*, vol. 188, no. 12, pp. 2222–2239, 2019.

[41] G. Biau and E. Scornet, "A random forest guided tour," *Test*, vol. 25, no. 2, pp. 197–227, 2016.

[42] "Introduction to random forest in machine learning."

[43] "1.4. support vector machines."

[44] "Xgboost," Jul 2022.

[45] P. Gupta, "Cross-validation in machine learning," Jun 2017.

[46] J. Brownlee, "What is a confusion matrix in machine learning," Aug 2020.

[47] "Classification: Roc curve and auc — machine learning;— google developers."

[48] C. Molnar, "Interpretable machine learning," Dec 2022.

[49] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in python," *J. Mach. Learn. Res.*, vol. 12, p. 2825–2830, nov 2011.

[50] A. Abu-Rmileh, "Be careful when interpreting your features importance in xgboost!," Sep 2021.

X. Appendix

A. Source Code

source–code/modeling–files/initial preprocessing.py

```
#------------------------------ IMPORTANT REMINDER ------------------------------#
#| This code is a cleaned python version of the actual notebook          |
#| file (.ipynb) that was used in the modeling process of the study.     |
#| Hence, the original outputs are not displayed. The code is formatted |
#| for a cleaner and more organized presentation in the manuscript.      |
#------------------------------------ END -------------------------------------#

# Importing Necessary Libraries
import pandas as pd
import numpy as np
pd.options.display.max_rows = 4200

# Reading the original dataset
df = pd.read_excel('APLC 13-20 shared MASTER dataset 050522.xlsx')

# Showing the first five rows
df.head()

# Checking variable types
print(df.dtypes)
print()

# Checking initial size of the original dataset
print("Initial size of the dataset:")
print(df.shape)
print()

# Selecting only patients who died
df1 = df.copy()
df1 = df1.loc[df['deceased_Pt'] == "yes"]
print("Size of dataset after selecting only dead patients:")
print(df1.shape)

# Checking the number of patients who met certain criteria
print("Total no. of patients who have SLE")
print(df1[(df1.ACR_fulfilled == "yes") | (df1.SLICC_fulfilled == "yes")].shape[0])
print()
print("Total no. of patients whose death is SLE related:")
print(sum(df1['death_SLE_related'] == "yes"))
print()
print("Total no. of patients whose death is not SLE related:")
print(sum(df1['death_SLE_related'] == "no"))
print()

# FEATURE ENGINEERING
# Replace the values under 'deceased_Pt' column with "no"
# if it isn't the last visit of the patient.
df1.sort_values (['ID_new', 'visit_no '], inplace=True)
grouped = df1.groupby('ID_new')
df1['deceased_Pt'] = grouped['deceased_Pt'].transform(lambda x: x.mask(x.shift(-1).notnull(), 'no'))
print(df1 [['ID_new','visit_no ','deceased_Pt']])

# Selecting only patients whose death is SLE related
df_sle = df1.copy()
df_sle = df_sle.loc[df_sle['death_SLE_related'] == "yes"]
print("Size after selecting only patients whose death is SLE related:")
print(df_sle.shape)

# Removing unnecessary columns from the dataset
to_drop = ['ID_new', 'ACR_fulfilled', 'SLICC_fulfilled ','visit_no ','total_visits ','visit_date ','visit_yr ','total_duration_days ','
    GDP_2020','birth_yr','enrol_yr ','age_enrol ','path_date ','race ',' RaceOther',
        'death_date ','death_SLE_related','death_infection ','death_cardio ','death_malignancy','tertiary_edu ','current_smoker ','
    incept_cohort ',
        'ACR_malar_rash','ACR_discoid_rash','ACR_oral_ulcers','ACR_Photosensitivity','ACR_arthritis','ACR_serositis','
    ACR_renal','ACR_neurologic','ACR_haemato',
        'SLICC_crit1_AClupus_any','SLICC_crit1_malarR','SLICC_crit1_bullous','SLICC_crit1_necrolysis','
    SLICC_crit1_maculopapularR','SLICC_crit1_photosensitiveR','SLICC_crit1_sub_cutaneous',
        'SLICC_crit2_CClupus_any','SLICC_crit2_discoidR','SLICC_crit2_localised','SLICC_crit2_generalised ','
    SLICC_crit2_hypertrophic','SLICC_crit2_panniculitis ',' SLICC_crit2_mucosal','SLICC_crit2_tumidus',
        'SLICC_crit2_chiblains ',' SLICC_crit2_discoid_lichen_overl ',' SLICC_crit3_oral_nasal_ulcers ','
    SLICC_crit4_nonscar_alopecia','SLICC_crit5_synovitis_tender ',' SLICC_crit5_synovitis ',' SLICC_crit5_tenderness',
        'SLICC_crit6_serositis_any ',' SLICC_crit6_pleuritic_pain ',' SLICC_crit6_pleural_effusions ',' SLICC_crit6_pleural_rub ','
    SLICC_crit6_pericardial_pain ',' SLICC_crit6_pericardial_effusion ',' SLICC_crit6_pericardial_rub ',
        'SLICC_crit6_pericarditis ',' SLICC_crit7_renal_any','SLICC_crit7_renal_highProtein','SLICC_crit7_renal_RBCcasts','
    SLICC_crit8_neurologic_any','SLICC_crit8_seizures','SLICC_crit8_psychosis','SLICC_crit8_mononeuritis',
        'SLICC_crit8_myelitis ',' SLICC_crit8_PorCneuropathy','SLICC_crit8_confusional_state','SLICC_crit9_Hemolytic_anemia','
    SLICC_crit10_leuko_lympho','SLICC_crit10_Leukopenia4000','SLICC_crit10_Lymphopenia1000','SLICC_crit11_thrombocytopenia
    ',
```

```
            'SLEDAI_Seizure_8','SLEDAI_Psychosis_8','SLEDAI_OrganicBrainSyndrome_8','SLEDAI_VisualDisturbance_8','
        SLEDAI_CranialNerveDisorder_8','SLEDAI_LupusHeadache_8','SLEDAI_CVA_8','SLEDAI_Vasculitis_8','SLEDAI_Arthritis_4','
        SLEDAI_Myositis_4',
            'SLEDAI_UrinaryCasts_4','SLEDAI_Haematuria_4','SLEDAI_Proteinuria_4','SLEDAI_Pyuria_4','SLEDAI_Rash_2','
        SLEDAI_Alopecia_2','SLEDAI_MucosalUlcers_2','SLEDAI_Pleurisy_2','SLEDAI_Pericarditis_2','SLEDAI_LowComplement_2',
            'SLEDAI_IncreasedDNAbinding_2','SLEDAI_Fever_1','SLEDAI_Thrombocytopaenia_1','SLEDAI_Leucopaenia_1','
        SF_CNS_SLE','SF_vasculitis','SF_nephritis','SF_myositis','SF_thrombocytopenia','SF_hemolytic_anaemia','SF_PNL_Increase',
            'SF_CYM_AZA_MTX_LEF_Cellcept','SF_hospitalisation','SF_PGA_increase','SF_increase_SLEDAI_12','
        MF_increase_SLEDAI_3','MF_rash','MF_ulcers','MF_pleuritis','MF_pericarditis','MF_arthritis','MF_fever','MF_PNL_Increase','
        MF_NSAID_or_HCQ',
            'MF_PGA_increase','SDI_cataract','SDI_cataract_CF','SDI_Retinal','SDI_Retinal_CF','SDI_Cognitive','SDI_Cognitive_CF','
        SDI_Seizures','SDI_Seizures_CF','SDI_CVA','SDI_CVA_CF','SDI_neuropathy','SDI_neuropathy_CF','SDI_Transverse_myelitis',
            'SDI_Transverse_myelitis_CF','SDI_low_eGFR','SDI_low_eGFR_CF','SDI_Proteinuria_over35','SDI_Proteinuria_over35_CF','
        SDI_ESRD','SDI_ESRD_CF','SDI_Pul_hypertension','SDI_Pul_hypertension_CF','SDI_Pul_fibrosis',
            'SDI_Pul_fibrosis_CF ',' SDI_Shrinking_lung','SDI_Shrinking_lung_CF','SDI_Pleural_fibrosis ',' SDI_Pleural_fibrosis_CF ','
        SDI_Pulmonary_infarction','SDI_Pulmonary_infarction_CF','SDI_Angina_or_CAB','SDI_Angina_or_CAB_CF',
            'SDI_MI','SDI_MI_CF','SDI_Cardiomyopathy','SDI_Cardiomyopathy_CF','SDI_Valvular_disease_CF','SDI_Pericarditis','
        SDI_Pericarditis_CF','SDI_Claudication','SDI_Claudication_CF','SDI_Minor_TissueLoss','SDI_Minor_TissueLoss_CF',
            'SDI_Significant_TissueLoss ',' SDI_Significant_TissueLoss_CF','SDI_Venous','SDI_Venous_CF','SDI_Infarction_bowel','
        SDI_Infarction_bowel_CF','SDI_Mesenteric','SDI_Mesenteric_CF','SDI_peritonitis','SDI_peritonitis_CF',
            'SDI_Stricture ',' SDI_Stricture_CF','SDI_Muscle_atrophy','SDI_Muscle_atrophy_CF','SDI_Deforming_arthritis','
        SDI_Deforming_arthritis_CF','SDI_Avascular_necrosis','SDI_Avascular_necrosis_CF','SDI_Osteoporosis','SDI_Osteoporosis_CF',
            'SDI_Osteomyelitis','SDI_Osteomyelitis_CF','SDI_Scarring_alopecia','SDI_Scarring_alopecia_CF','SDI_panniculum','
        SDI_panniculum_CF','SDI_Skin_ulceration','SDI_Skin_ulceration_CF','SDI_Premature_gonadal_fail','
        SDI_Premature_gonadal_fail_CF',
            'SDI_PGfail_bin','SDI_Diabetes','SDI_Diabetes_CF','SDI_Diabetes_bin','SDI_Malignancy','SDI_Malignancy_CF','
        SDI_Malignancy_bin','SDI_Valvular_disease',
            'Urine_prot_creat_ratio_Luo ',' urine_comments_STRING','PNL_ever','HydroxyChol_ever','Chloroquine_ever','AM_ever','
        Methotrexate_ever','Azathioprine_ever','Mycophenolate_ever','Myco_acid_ever','MMF_MPA_ever',
            'Leflunomide_ever ',' Ciclosporin_ever ',' Tacrolimus_ever ',' Mizoribine_ever ',' Cyclophosphamide_ever','IS_ever','
        Rituximab_ever',' biologics_ever ',' Belimumab_ever',
            'Immuno1_highANA','Immuno2_anti_dsDNA','Immuno3_anti_Sm','Immuno4_APL_Ab_any','Immuno4_i_anti_coag','
        Immuno4_ii_anti_cardiolipin','Immuno4_iii_falseP_RPR','Immuno4_iv_B2GP','Immuno5_low_Compl_any','Immuno5_ii_LowC3','
        Immuno5_iii_LowC4',
            'Immuno5_iv_LowCH50','Immuno6_Coombs','LLDAS_ever','LLDAS_percent','LLDAS_50_v','SF36_some_missing','
        disease_duration_at_enrol','anti_dsDNA_UL','anti_dsDNA_noyes','MF_noyes','MF_ever','SF_ever','SF_noyes','MF_no_total','
        SF_no_total',
            ' any_flare_ever ',' any_flare_noyes ',' any_flare_total ',' cumPNL_at_visit','cumPNL_t','cumPNL_t_g','SF36_performed','
        SF36_complete','SF36_GeneralHealth','SF36_GeneralHealth_compared1yr','SF36_Vigorous_activities','SF36_Moderate_activities',
            ' SF36_Lifting_groceries ',' SF36_Climbing_several_stairs ',' SF36_Climbing_one_stairs ','SF36_Bending_kneeling_stooping','
        SF36_Walking_miles','SF36_Walking_several_yards','SF36_Walking_100_yards','SF36_Bathing_dresssing','
        SF36_Cut_down_work_due_PhysicalH',
            'SF36_Accomplish_less_due_Physica','SF36_OtherActivites_due_Physical','SF36_difficulty_working_due_Phys','
        SF36_Cut_down_work_due_Emotional','SF36_Accomplish_less_due_Emotion','SF36_Noactivities_due_EmotionalH','
        SF36_Interference_social_activit',
            'SF36_bodily_pain','SF36_pain_interfere_normalwork','SF36_Feel_fullOFlife ',' SF36_Very_nervous','SF36_Down_dumps','
        SF36_calm_peaceful','SF36_LotOFenergy','SF36_downhearted_depressed','SF36_Feel_worn_out','SF36_Happy','SF36_Tired',
            ' SF36_Interfere_social_activities ',' SF36_sick_easier ',' SF36_healthy_as_anyone','SF36_Expect_worse_health','
        SF36_excellent_health',' PF',' RP',' BP',' GH',' VT',' SF',' RE',' MH',' overall_damage_accrual_bin',' SDI_last_score ',' last_damage_bin
        ',' SDI_ever',
            'TAM_PNL','PNL_noyes','total_duration_yrs','SDI_noyes','SDI_TOTALscore','SDI_baseline_score','baseline_damage_bin','
        yr_SLE_onset','yr_SLE_diagnosis',
            ]

df_reduc = df_sle.copy()
df_reduc.drop(to_drop, axis=1, inplace=True)
print("Size after initial dropping of columns:")
print(df_reduc.shape)
print()

# Checking the percentage of missing values in each column
print(df_reduc.isnull().mean())

# Dropping columns with more than 20% missing data
# that are Missing At Random (MAR)
drop_na_cols = ['CRP', 'urine_comments',
                'eGFR','Alb','C4','ESR',' Urine_prot_creat_ratio ',
                'PF_NBS','RP_NBS','BP_NBS','GH_NBS','VT_NBS','SF_NBS',
                'RE_NBS','MH_NBS','PCS','MCS',]

df_reduc.drop(drop_na_cols, axis=1, inplace=True)
print("Size after dropping columns with >20% missing data:")
print(df_reduc.shape)
print()

# Checking number of unique values for each column
df_reduc.nunique()

# Dropping columns that are redundant or have only 1 unique value
to_drop = ['HydroxyChol_noyes','Chloroquine_noyes','Methotrexate_noyes','Azathioprine_noyes',
           'Mycophenolate_noyes','Myco_acid_noyes','Ciclosporin_noyes','Cyclophosphamide_noyes',
           'Leflunomide_noyes','Leflunomide','Tacrolimus_noyes','Tacrolimus',
           'Mizoribine_noyes ','Mizoribine ',' Rituximab','Belimumab',]

df_reduc.drop(to_drop, axis=1, inplace=True)
print("Size after dropping redundant columns or those which have only one unique value:")
print(df_reduc.shape)
print()

# Imputing features with missing data that are Not Missing At Random (NMAR)
nmar = ['visit_duration ',' TAM_SLEDAI','AMS_visit','TAM_PGA',
        'AM_PGA_visit','TAM_PNL_visit','LLDAS_percent_v','TAM_LLDAS']
```

```
for col in nmar:
    df_reduc[col] = df_reduc[col].fillna(0)
print(df_reduc.isnull().sum())

# Exporting the shortened and filtered dataset for use in modeling
df_reduc.to_csv("shortened_dataset.csv")
```

source–code/modeling–files/final modeling.py

```
#------------------------------ IMPORTANT REMINDER ------------------------------#
#| This code is a cleaned python version of the actual notebook          |
#| file (.ipynb) that was used in the modeling process of the study.     |
#| Hence, the original outputs are not displayed. The code is formatted |
#| for a cleaner and more organized presentation in the manuscript.      |
#------------------------------------ END -------------------------------------#

# Importing Necessary Libraries
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier, RandomForestRegressor
from sklearn import svm
from sklearn.linear_model import LogisticRegression, BayesianRidge
from sklearn.metrics import accuracy_score, confusion_matrix, ConfusionMatrixDisplay, classification_report, roc_auc_score,
    precision_score, recall_score, f1_score
import xgboost as xgb
from sklearn.preprocessing import LabelEncoder, OrdinalEncoder, OneHotEncoder
from sklearn.model_selection import RandomizedSearchCV, cross_val_score
from scipy.stats import randint, uniform
from sklearn.experimental import enable_iterative_imputer
from sklearn.impute import IterativeImputer, SimpleImputer
import matplotlib.pyplot as plt
from imblearn.over_sampling import SMOTE
from collections import Counter
from sklearn.datasets import make_classification
from sklearn.feature_selection import RFE, RFECV
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import StratifiedKFold
from sklearn.preprocessing import StandardScaler, MinMaxScaler
import joblib
import pickle
pd.options.display.max_rows = 4200

# Reading the initially-preprocessed dataset
df = pd.read_csv('shortened_dataset.csv')

# Showing the first five rows
df.head()

# Checking variable types
print(df.dtypes)
print()

# Checking initial size of the shortened dataset
print("Initial size of the dataset:")
print(df.shape)
print()

# Splitting the dataset into 70% train set and 30% test set
df_reduc = df.copy()
X = df_reduc.drop('deceased_Pt', axis=1)
y = df_reduc['deceased_Pt']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42, stratify=y)

# Checking the size of the train and test sets
print("X_train", X_train.shape)
print("X_test", X_test.shape)
print()

# Checking the number of unique values for each column
for col_name in X_train.columns:
    if X_train[col_name].dtypes == 'object':
        print(col_name, X_train[col_name].nunique())

print()
for col_name in X_test.columns:
    if X_test[col_name].dtypes == 'object':
        print(col_name, X_test[col_name].nunique())


# START OF FEATURE ENCODING
    ================================================================================
# Manually converting values of smoking_status to numerical values
X_train["smoking_status"] = X_train["smoking_status"].replace('never', 0)
X_train["smoking_status"] = X_train["smoking_status"].replace('ex', 1)
X_train["smoking_status"] = X_train["smoking_status"].replace('current', 2)
X_test["smoking_status"] = X_test["smoking_status"].replace('never', 0)
X_test["smoking_status"] = X_test["smoking_status"].replace('ex', 1)
X_test["smoking_status"] = X_test["smoking_status"].replace('current', 2)
```

```
79
80 # Manually converting values of LLDAS_All_5cat to numerical values
81 X_train["LLDAS_All_5cat"] = X_train["LLDAS_All_5cat"].replace('not in LLDAS', 0)
82 X_train["LLDAS_All_5cat"] = X_train["LLDAS_All_5cat"].replace('in LLDAS', 1)
83 X_test["LLDAS_All_5cat"] = X_test["LLDAS_All_5cat"].replace('not in LLDAS', 0)
84 X_test["LLDAS_All_5cat"] = X_test["LLDAS_All_5cat"].replace('in LLDAS', 1)
85
86 # Encoding other categorical features using LabelEncoder and OrdinalEncoder
87 for col_name in X_train.columns:
88     if X_train[col_name].dtypes == 'object':
89         if X_train[col_name].nunique() <= 2 and col_name != 'country':
90             series = X_train[col_name]
91             le = LabelEncoder()
92             X_train[col_name] = pd.Series(le.fit_transform(series[series.notnull()]),
93                                           index=series[series.notnull()].index)
94         else:
95             if col_name != 'country':
96                 oe = OrdinalEncoder()
97                 column_data = X_train[col_name].values.reshape(-1, 1)
98                 X_train[col_name] = oe.fit_transform(column_data)
99
100 for col_name in X_test.columns:
101     if X_test[col_name].dtypes == 'object':
102         if X_test[col_name].nunique() <= 2 and col_name != 'country':
103             series = X_test[col_name]
104             le = LabelEncoder()
105             X_test[col_name] = pd.Series(le.fit_transform(series[series.notnull()]),
106                                          index=series[series.notnull()].index)
107         else:
108             if col_name != 'country':
109                 oe = OrdinalEncoder()
110                 column_data = X_test[col_name].values.reshape(-1, 1)
111                 X_test[col_name] = oe.fit_transform(column_data)
112
113 # Encoding the values of 'country' using OneHotEncoder
114 enc = OneHotEncoder(sparse=False)
115 transformed_df = pd.DataFrame(enc.fit_transform(X_train[['country']]),
116                               columns = enc.get_feature_names_out(),
117                               index = X_train.index)
118 X_train = pd.concat([X_train.drop('country',axis=1),transformed_df],axis=1)
119
120 transformed_df1 = pd.DataFrame(enc.fit_transform(X_test[['country']]),
121                               columns = enc.get_feature_names_out(),
122                               index = X_test.index)
123 X_test = pd.concat([X_test.drop('country',axis=1),transformed_df1],axis=1)
124
125 # Encoding the target variable using LabelEncoder
126 y_train_le = LabelEncoder()
127 y_train = y_train_le.fit_transform(y_train)
128 y_test_le = LabelEncoder()
129 y_test = y_test_le.fit_transform(y_test)
130
131 # END OF FEATURE ENCODING
      ================================================================================
132
133
134 # Checking size of the dataset after feature encoding
135 print(X_train.shape)
136 print(X_test.shape)
137 print()
138
139 # Initially training and testing the XGBoost model on the unimputed dataset
140 # Note: XGBoost can handle missing values
141 xgb_cls = xgb.XGBClassifier()
142 xgb_cls.fit(X_train, y_train)
143 XGB_pred = xgb_cls.predict(X_test)
144 XGBaccuracy,XGBaucroc = accuracy_score(y_test, XGB_pred),roc_auc_score(y_test, XGB_pred, average='macro')
145 XGBprecision,XGBrecall,XGBf1 = precision_score(y_test, XGB_pred, average='macro'),recall_score(y_test, XGB_pred, average='
      macro'),f1_score(y_test, XGB_pred, average='macro')
146
147 scoreDF = pd.DataFrame(columns=['accuracy','precision','recall','f1 '," roc_auc","model","Preprocessing"])
148 data =[XGBaccuracy,XGBprecision,XGBrecall,XGBf1,XGBaucroc,"XGB","None",]
149 scoreDF = pd.concat([scoreDF, pd.DataFrame([data], columns=scoreDF.columns)], ignore_index=True)
150 scoreDF
151
152
153 # START OF DATA IMPUTATION
      ================================================================================
154 # Checking percentage of missingness in each column
155 print("X_train null", X_train.isnull().mean())
156 print()
157 print("X_test null", X_test.isnull().mean())
158 print()
159
160 # Getting the list of categorical features
161 catcols = []
162 for col_name in X_train.columns:
163     if X_train[col_name].nunique() <= 3:
164         catcols.append(col_name)
165 print(catcols)
166 print()
167
```

```
# Using Single Imputation for categorical features
for col in catcols:
    imputer = SimpleImputer(strategy='most_frequent')
    imputer.fit(X_train[col].values.reshape(-1, 1))
    X_train[col] = imputer.transform(X_train[col].values.reshape(-1, 1))

    imputer.fit(X_test[col].values.reshape(-1, 1))
    X_test[col] = imputer.transform(X_test[col].values.reshape(-1, 1))

# Checking the number of missing values in categorical features after the imputation
for col in catcols:
    print(col, X_train[col].isnull().sum())
    print(col, X_test[col].isnull().sum())
print()

# Using MICE with RandomForestRegressor for continuous features
imp_br1 = IterativeImputer(estimator=RandomForestRegressor(),initial_strategy='most_frequent', max_iter=100, imputation_order='
    random', random_state=42)
imputed_data_br1 = imp_br1.fit_transform(X_train)
imputed_X_train = pd.DataFrame(imputed_data_br1, columns=X_train.columns)

imputed_data_br2 = imp_br1.fit_transform(X_test)
imputed_X_test = pd.DataFrame(imputed_data_br2, columns=X_test.columns)

# Checking percentage of missingness after the imputation
print(imputed_X_train.isnull().sum().sum())
print(imputed_X_test.isnull().sum().sum())
print()

# END OF DATA IMPUTATION
    ================================================================================


# Checking duplicate rows
print(imputed_X_train.duplicated().sum())
print(imputed_X_test.duplicated().sum())
print()

# Checking class imbalance on the target variable
class_nobalance = pd.Series(y_train).value_counts().plot.bar()
class_nobalance.set_title("Outcome ytrain (Without SMOTE)")
plt.gca().set_xticklabels(['Alive','Died']);
class_nobalance.figure.savefig('class_nobalance.png')

# ================================================================
# | Data augmentation, Feature selection, and Feature Scaling |
# | are further performed to create various datasets.         |
# ================================================================

# START OF DATA AUGMENTATION as the base algorithm
    ================================================================================
# SMOTE only
sm = SMOTE(random_state = 42)
X_train_re, y_train_re = sm.fit_resample(imputed_X_train, y_train.ravel())

# SMOTE, then RFECV
clf = RFECV(RandomForestClassifier(), cv=StratifiedKFold(10), scoring='roc_auc')
clf = clf.fit(X_train_re, y_train_re)
selected_features = clf.get_support(1)
X_train_rese = X_train_re[X_train_re.columns[selected_features]]
X_test_rese = imputed_X_test[imputed_X_test.columns[selected_features]]

# SMOTE, then RFECV, then StandardScaler
X_train_resesc = X_train_rese.copy()
X_test_resesc = X_test_rese.copy()

cont_cols = []
for col_name in X_train_resesc.columns:
    if X_train_resesc[col_name].nunique() > 3:
        cont_cols.append(col_name)

for col in cont_cols:
    sc = StandardScaler()
    sc.fit(X_train_resesc[col].values.reshape(-1, 1))
    X_train_resesc[col] = sc.transform(X_train_resesc[col].values.reshape(-1, 1))

    sc.fit(X_test_resesc[col].values.reshape(-1, 1))
    X_test_resesc[col] = sc.transform(X_test_resesc[col].values.reshape(-1, 1))

# SMOTE, then StandardScaler
X_train_resc = X_train_re.copy()
X_test_resc = imputed_X_test.copy()

cont_cols = []
for col_name in X_train_resc.columns:
    if X_train_resc[col_name].nunique() > 3:
        cont_cols.append(col_name)

for col in cont_cols:
    sc = StandardScaler()
    sc.fit(X_train_resc[col].values.reshape(-1, 1))
```

```
256         X_train_resc[col] = sc.transform(X_train_resc[col].values.reshape(-1, 1))
257
258         sc.fit(X_test_resc[col].values.reshape(-1, 1))
259         X_test_resc[col] = sc.transform(X_test_resc[col].values.reshape(-1, 1))
260
261 # SMOTE, then StandardScaler, then RFECV
262 clf1 = RFECV(RandomForestClassifier(), cv=StratifiedKFold(10), scoring='roc_auc')
263 clf1 = clf1.fit(X_train_resc, y_train_re)
264 selected_features = clf1.get_support(1)
265 X_train_rescse = X_train_resc[X_train_resc.columns[selected_features]]
266 X_test_rescse = X_test_resc[X_test_resc.columns[selected_features]]
267
268 # END OF DATA AUGMENTATION as the base algorithm
        ==========================================================================


269
270
271 # START OF TRAINING AND TESTING OF THE 4 MODELS ON THE IMPUTED DATASET AND 5 SMOTE-FIRST
        DATASETS ============================
272 scoreDF = pd.DataFrame(columns=['accuracy','precision','recall','f1 '," roc_auc","model","dataset"])
273
274 # 0. Imputed only
275 rf = RandomForestClassifier()
276 rf.fit(imputed_X_train, y_train)
277 RF_pred = rf.predict(imputed_X_test)
278 RFaccuracy,RFaucroc = accuracy_score(y_test, RF_pred),roc_auc_score(y_test, RF_pred, average='macro')
279 RFprecision,RFrecall,RFf1 = precision_score(y_test, RF_pred, average='macro'),recall_score(y_test, RF_pred, average='macro'),
        f1_score(y_test, RF_pred, average='macro')
280 data =[RFaccuracy,RFprecision,RFrecall,RFf1,RFaucroc,"RF","imputed",]
281 scoreDF = pd.concat([scoreDF, pd.DataFrame([data], columns=scoreDF.columns)], ignore_index=True)
282
283 xgb_cls = xgb.XGBClassifier()
284 xgb_cls.fit(imputed_X_train, y_train)
285 XGB_pred = xgb_cls.predict(imputed_X_test)
286 XGBaccuracy,XGBaucroc = accuracy_score(y_test, XGB_pred),roc_auc_score(y_test, XGB_pred, average='macro')
287 XGBprecision,XGBrecall,XGBf1 = precision_score(y_test, XGB_pred, average='macro'),recall_score(y_test, XGB_pred, average='
        macro'),f1_score(y_test, XGB_pred, average='macro')
288 data =[XGBaccuracy,XGBprecision,XGBrecall,XGBf1,XGBaucroc,"XGB","imputed",]
289 scoreDF = pd.concat([scoreDF, pd.DataFrame([data], columns=scoreDF.columns)], ignore_index=True)
290
291 lr = LogisticRegression(max_iter=10000)
292 lr.fit(imputed_X_train, y_train)
293 LR_pred = lr.predict(imputed_X_test)
294 LRaccuracy,LRaucroc = accuracy_score(y_test, LR_pred),roc_auc_score(y_test, LR_pred, average='macro')
295 LRprecision,LRrecall,LRf1 = precision_score(y_test, LR_pred, average='macro'),recall_score(y_test, LR_pred, average='macro'),
        f1_score(y_test, LR_pred, average='macro')
296 data =[LRaccuracy,LRprecision,LRrecall,LRf1,LRaucroc,"LR","imputed",]
297 scoreDF = pd.concat([scoreDF, pd.DataFrame([data], columns=scoreDF.columns)], ignore_index=True)
298
299 svc=svm.SVC()
300 svc.fit(imputed_X_train, y_train)
301 SVM_prediction = svc.predict(imputed_X_test)
302 SVMaccuracy,SVMaucroc = accuracy_score(y_test, SVM_prediction),roc_auc_score(y_test, SVM_prediction, average='macro')
303 SVMprecision,SVMrecall,SVMf1 = precision_score(y_test, SVM_prediction, average='macro'),recall_score(y_test, SVM_prediction,
        average='macro'),f1_score(y_test, SVM_prediction, average='macro')
304 data =[SVMaccuracy,SVMprecision,SVMrecall,SVMf1,SVMaucroc,"SVM","imputed",]
305 scoreDF = pd.concat([scoreDF, pd.DataFrame([data], columns=scoreDF.columns)], ignore_index=True)
306
307
308 # 1. SMOTE only
309 rf2 = RandomForestClassifier()
310 rf2.fit(X_train_re, y_train_re)
311 RF_pred = rf2.predict(imputed_X_test)
312 RFaccuracy,RFaucroc = accuracy_score(y_test, RF_pred),roc_auc_score(y_test, RF_pred, average='macro')
313 RFprecision,RFrecall,RFf1 = precision_score(y_test, RF_pred, average='macro'),recall_score(y_test, RF_pred, average='macro'),
        f1_score(y_test, RF_pred, average='macro')
314 data =[RFaccuracy,RFprecision,RFrecall,RFf1,RFaucroc,"RF","SMOTE",]
315 scoreDF = pd.concat([scoreDF, pd.DataFrame([data], columns=scoreDF.columns)], ignore_index=True)
316
317 xgb_cls_best = xgb.XGBClassifier()
318 xgb_cls_best.fit(X_train_re, y_train_re)
319 XGB_pred = xgb_cls_best.predict(imputed_X_test)
320 XGBaccuracy,XGBaucroc = accuracy_score(y_test, XGB_pred),roc_auc_score(y_test, XGB_pred, average='macro')
321 XGBprecision,XGBrecall,XGBf1 = precision_score(y_test, XGB_pred, average='macro'),recall_score(y_test, XGB_pred, average='
        macro'),f1_score(y_test, XGB_pred, average='macro')
322 data =[XGBaccuracy,XGBprecision,XGBrecall,XGBf1,XGBaucroc,"XGB","SMOTE",]
323 scoreDF = pd.concat([scoreDF, pd.DataFrame([data], columns=scoreDF.columns)], ignore_index=True)
324
325 lr = LogisticRegression(max_iter=10000)
326 lr.fit(X_train_re, y_train_re)
327 LR_pred = lr.predict(imputed_X_test)
328 LRaccuracy,LRaucroc = accuracy_score(y_test, LR_pred),roc_auc_score(y_test, LR_pred, average='macro')
329 LRprecision,LRrecall,LRf1 = precision_score(y_test, LR_pred, average='macro'),recall_score(y_test, LR_pred, average='macro'),
        f1_score(y_test, LR_pred, average='macro')
330 data =[LRaccuracy,LRprecision,LRrecall,LRf1,LRaucroc,"LR","SMOTE",]
331 scoreDF = pd.concat([scoreDF, pd.DataFrame([data], columns=scoreDF.columns)], ignore_index=True)
332
333 svc=svm.SVC()
334 svc.fit(X_train_re, y_train_re)
335 SVM_prediction = svc.predict(imputed_X_test)
336 SVMaccuracy,SVMaucroc = accuracy_score(y_test, SVM_prediction),roc_auc_score(y_test, SVM_prediction, average='macro')
337 SVMprecision,SVMrecall,SVMf1 = precision_score(y_test, SVM_prediction, average='macro'),recall_score(y_test, SVM_prediction,
```

```
        average='macro'),f1_score(y_test, SVM_prediction, average='macro')
data =[SVMaccuracy,SVMprecision,SVMrecall,SVMf1,SVMaucroc,"SVM","SMOTE",]
scoreDF = pd.concat([scoreDF, pd.DataFrame([data], columns=scoreDF.columns)], ignore_index=True)


# 2. SMOTE, then RFECV
rf5 = RandomForestClassifier()
rf5.fit(X_train_rese, y_train_re)
RF_pred = rf5.predict(X_test_rese)
RFaccuracy,RFaucroc = accuracy_score(y_test, RF_pred),roc_auc_score(y_test, RF_pred, average='macro')
RFprecision,RFrecall,RFf1 = precision_score(y_test, RF_pred, average='macro'),recall_score(y_test, RF_pred, average='macro'),
        f1_score(y_test, RF_pred, average='macro')
data =[RFaccuracy,RFprecision,RFrecall,RFf1,RFaucroc,"RF","SMOTE-RFECV",]
scoreDF = pd.concat([scoreDF, pd.DataFrame([data], columns=scoreDF.columns)], ignore_index=True)

xgb_cls = xgb.XGBClassifier()
xgb_cls.fit(X_train_rese, y_train_re)
XGB_pred = xgb_cls.predict(X_test_rese)
XGBaccuracy,XGBaucroc = accuracy_score(y_test, XGB_pred),roc_auc_score(y_test, XGB_pred, average='macro')
XGBprecision,XGBrecall,XGBf1 = precision_score(y_test, XGB_pred, average='macro'),recall_score(y_test, XGB_pred, average='
        macro'),f1_score(y_test, XGB_pred, average='macro')
data =[XGBaccuracy,XGBprecision,XGBrecall,XGBf1,XGBaucroc,"XGB","SMOTE-RFECV",]
scoreDF = pd.concat([scoreDF, pd.DataFrame([data], columns=scoreDF.columns)], ignore_index=True)

lr = LogisticRegression(max_iter=10000)
lr.fit(X_train_rese, y_train_re)
LR_pred = lr.predict(X_test_rese)
LRaccuracy,LRaucroc = accuracy_score(y_test, LR_pred),roc_auc_score(y_test, LR_pred, average='macro')
LRprecision,LRrecall,LRf1 = precision_score(y_test, LR_pred, average='macro'),recall_score(y_test, LR_pred, average='macro'),
        f1_score(y_test, LR_pred, average='macro')
data =[LRaccuracy,LRprecision,LRrecall,LRf1,LRaucroc,"LR","SMOTE-RFECV",]
scoreDF = pd.concat([scoreDF, pd.DataFrame([data], columns=scoreDF.columns)], ignore_index=True)

svc=svm.SVC()
svc.fit(X_train_rese, y_train_re)
SVM_prediction = svc.predict(X_test_rese)
SVMaccuracy,SVMaucroc = accuracy_score(y_test, SVM_prediction),roc_auc_score(y_test, SVM_prediction, average='macro')
SVMprecision,SVMrecall,SVMf1 = precision_score(y_test, SVM_prediction, average='macro'),recall_score(y_test, SVM_prediction,
        average='macro'),f1_score(y_test, SVM_prediction, average='macro')
data =[SVMaccuracy,SVMprecision,SVMrecall,SVMf1,SVMaucroc,"SVM","SMOTE-RFECV",]
scoreDF = pd.concat([scoreDF, pd.DataFrame([data], columns=scoreDF.columns)], ignore_index=True)


# 3. SMOTE, then RFECV, then StandardScaler
rf5 = RandomForestClassifier()
rf5.fit(X_train_resesc, y_train_re)
RF_pred = rf5.predict(X_test_resesc)
RFaccuracy,RFaucroc = accuracy_score(y_test, RF_pred),roc_auc_score(y_test, RF_pred, average='macro')
RFprecision,RFrecall,RFf1 = precision_score(y_test, RF_pred, average='macro'),recall_score(y_test, RF_pred, average='macro'),
        f1_score(y_test, RF_pred, average='macro')
data =[RFaccuracy,RFprecision,RFrecall,RFf1,RFaucroc,"RF","SMOTE-RFECV-StandardScaler",]
scoreDF = pd.concat([scoreDF, pd.DataFrame([data], columns=scoreDF.columns)], ignore_index=True)

xgb_cls = xgb.XGBClassifier()
xgb_cls.fit(X_train_resesc, y_train_re)
XGB_pred = xgb_cls.predict(X_test_resesc)
XGBaccuracy,XGBaucroc = accuracy_score(y_test, XGB_pred),roc_auc_score(y_test, XGB_pred, average='macro')
XGBprecision,XGBrecall,XGBf1 = precision_score(y_test, XGB_pred, average='macro'),recall_score(y_test, XGB_pred, average='
        macro'),f1_score(y_test, XGB_pred, average='macro')
data =[XGBaccuracy,XGBprecision,XGBrecall,XGBf1,XGBaucroc,"XGB","SMOTE-RFECV-StandardScaler",]
scoreDF = pd.concat([scoreDF, pd.DataFrame([data], columns=scoreDF.columns)], ignore_index=True)

lr = LogisticRegression(max_iter=10000)
lr.fit(X_train_resesc, y_train_re)
LR_pred = lr.predict(X_test_resesc)
LRaccuracy,LRaucroc = accuracy_score(y_test, LR_pred),roc_auc_score(y_test, LR_pred, average='macro')
LRprecision,LRrecall,LRf1 = precision_score(y_test, LR_pred, average='macro'),recall_score(y_test, LR_pred, average='macro'),
        f1_score(y_test, LR_pred, average='macro')
data =[LRaccuracy,LRprecision,LRrecall,LRf1,LRaucroc,"LR","SMOTE-RFECV-StandardScaler",]
scoreDF = pd.concat([scoreDF, pd.DataFrame([data], columns=scoreDF.columns)], ignore_index=True)

svc=svm.SVC()
svc.fit(X_train_resesc, y_train_re)
SVM_prediction = svc.predict(X_test_resesc)
SVMaccuracy,SVMaucroc = accuracy_score(y_test, SVM_prediction),roc_auc_score(y_test, SVM_prediction, average='macro')
SVMprecision,SVMrecall,SVMf1 = precision_score(y_test, SVM_prediction, average='macro'),recall_score(y_test, SVM_prediction,
        average='macro'),f1_score(y_test, SVM_prediction, average='macro')
data =[SVMaccuracy,SVMprecision,SVMrecall,SVMf1,SVMaucroc,"SVM","SMOTE-RFECV-StandardScaler",]
scoreDF = pd.concat([scoreDF, pd.DataFrame([data], columns=scoreDF.columns)], ignore_index=True)


# 4. SMOTE, then StandardScaler
rf4 = RandomForestClassifier()
rf4.fit(X_train_resc, y_train_re)
RF_pred = rf4.predict(X_test_resc)
RFaccuracy,RFaucroc = accuracy_score(y_test, RF_pred),roc_auc_score(y_test, RF_pred, average='macro')
RFprecision,RFrecall,RFf1 = precision_score(y_test, RF_pred, average='macro'),recall_score(y_test, RF_pred, average='macro'),
        f1_score(y_test, RF_pred, average='macro')
data =[RFaccuracy,RFprecision,RFrecall,RFf1,RFaucroc,"RF","SMOTE-StandardScaler",]
scoreDF = pd.concat([scoreDF, pd.DataFrame([data], columns=scoreDF.columns)], ignore_index=True)

xgb_cls = xgb.XGBClassifier()
```

```
xgb_cls.fit(X_train_resc, y_train_re)
XGB_pred = xgb_cls.predict(X_test_resc)
XGBaccuracy,XGBaucroc = accuracy_score(y_test, XGB_pred),roc_auc_score(y_test, XGB_pred, average='macro')
XGBprecision,XGBrecall,XGBf1 = precision_score(y_test, XGB_pred, average='macro'),recall_score(y_test, XGB_pred, average='
    macro'),f1_score(y_test, XGB_pred, average='macro')
data =[XGBaccuracy,XGBprecision,XGBrecall,XGBf1,XGBaucroc,"XGB","SMOTE-StandardScaler",]
scoreDF = pd.concat([scoreDF, pd.DataFrame([data], columns=scoreDF.columns)], ignore_index=True)

lr = LogisticRegression(max_iter=10000)
lr.fit(X_train_resc, y_train_re)
LR_pred = lr.predict(X_test_resc)
LRaccuracy,LRaucroc = accuracy_score(y_test, LR_pred),roc_auc_score(y_test, LR_pred, average='macro')
LRprecision,LRrecall,LRf1 = precision_score(y_test, LR_pred, average='macro'),recall_score(y_test, LR_pred, average='macro'),
    f1_score(y_test, LR_pred, average='macro')
data =[LRaccuracy,LRprecision,LRrecall,LRf1,LRaucroc,"LR","SMOTE-StandardScaler",]
scoreDF = pd.concat([scoreDF, pd.DataFrame([data], columns=scoreDF.columns)], ignore_index=True)

svc=svm.SVC()
svc.fit(X_train_resc, y_train_re)
SVM_prediction = svc.predict(X_test_resc)
SVMaccuracy,SVMaucroc = accuracy_score(y_test, SVM_prediction),roc_auc_score(y_test, SVM_prediction, average='macro')
SVMprecision,SVMrecall,SVMf1 = precision_score(y_test, SVM_prediction, average='macro'),recall_score(y_test, SVM_prediction,
    average='macro'),f1_score(y_test, SVM_prediction, average='macro')
data =[SVMaccuracy,SVMprecision,SVMrecall,SVMf1,SVMaucroc,"SVM","SMOTE-StandardScaler",]
scoreDF = pd.concat([scoreDF, pd.DataFrame([data], columns=scoreDF.columns)], ignore_index=True)


# 5. SMOTE, then StandardScaler, then RFECV
rf4 = RandomForestClassifier()
rf4.fit(X_train_rescse, y_train_re)
RF_pred = rf4.predict(X_test_rescse)
RFaccuracy,RFaucroc = accuracy_score(y_test, RF_pred),roc_auc_score(y_test, RF_pred, average='macro')
RFprecision,RFrecall,RFf1 = precision_score(y_test, RF_pred, average='macro'),recall_score(y_test, RF_pred, average='macro'),
    f1_score(y_test, RF_pred, average='macro')
data =[RFaccuracy,RFprecision,RFrecall,RFf1,RFaucroc,"RF","SMOTE-StandardScaler-RFECV",]
scoreDF = pd.concat([scoreDF, pd.DataFrame([data], columns=scoreDF.columns)], ignore_index=True)

xgb_cls = xgb.XGBClassifier()
xgb_cls.fit(X_train_rescse, y_train_re)
XGB_pred = xgb_cls.predict(X_test_rescse)
XGBaccuracy,XGBaucroc = accuracy_score(y_test, XGB_pred),roc_auc_score(y_test, XGB_pred, average='macro')
XGBprecision,XGBrecall,XGBf1 = precision_score(y_test, XGB_pred, average='macro'),recall_score(y_test, XGB_pred, average='
    macro'),f1_score(y_test, XGB_pred, average='macro')
data =[XGBaccuracy,XGBprecision,XGBrecall,XGBf1,XGBaucroc,"XGB","SMOTE-StandardScaler-RFECV",]
scoreDF = pd.concat([scoreDF, pd.DataFrame([data], columns=scoreDF.columns)], ignore_index=True)

lr = LogisticRegression(max_iter=10000)
lr.fit(X_train_rescse, y_train_re)
LR_pred = lr.predict(X_test_rescse)
LRaccuracy,LRaucroc = accuracy_score(y_test, LR_pred),roc_auc_score(y_test, LR_pred, average='macro')
LRprecision,LRrecall,LRf1 = precision_score(y_test, LR_pred, average='macro'),recall_score(y_test, LR_pred, average='macro'),
    f1_score(y_test, LR_pred, average='macro')
data =[LRaccuracy,LRprecision,LRrecall,LRf1,LRaucroc,"LR","SMOTE-StandardScaler-RFECV",]
scoreDF = pd.concat([scoreDF, pd.DataFrame([data], columns=scoreDF.columns)], ignore_index=True)

svc=svm.SVC()
svc.fit(X_train_rescse, y_train_re)
SVM_prediction = svc.predict(X_test_rescse)
SVMaccuracy,SVMaucroc = accuracy_score(y_test, SVM_prediction),roc_auc_score(y_test, SVM_prediction, average='macro')
SVMprecision,SVMrecall,SVMf1 = precision_score(y_test, SVM_prediction, average='macro'),recall_score(y_test, SVM_prediction,
    average='macro'),f1_score(y_test, SVM_prediction, average='macro')
data =[SVMaccuracy,SVMprecision,SVMrecall,SVMf1,SVMaucroc,"SVM","SMOTE-StandardScaler-RFECV",]
scoreDF = pd.concat([scoreDF, pd.DataFrame([data], columns=scoreDF.columns)], ignore_index=True)

# Save and display the table of performance metrics
scoreDF.to_csv('SMOTE-first.csv')
scoreDF

# END OF TRAINING AND TESTING OF THE 4 MODELS ON THE IMPUTED DATASET AND 5 SMOTE-FIRST DATASETS
    ============================


# START OF FEATURE SELECTION as the base algorithm
    ================================================================================
# 1. RFECV only
clf = RFECV(RandomForestClassifier(), cv=StratifiedKFold(10), scoring='f1')
clf = clf.fit(imputed_X_train, y_train)
selected_features = clf.get_support(1)
X_train_se = imputed_X_train[imputed_X_train.columns[selected_features]]
X_test_se = imputed_X_test[imputed_X_test.columns[selected_features]]

# 2. RFECV, then SMOTE
sm = SMOTE(random_state = 42)
X_train_sere, y_train_sere = sm.fit_resample(X_train_se, y_train.ravel())

# 3. RFECV, then SMOTE, then StandardScaler
X_train_seresc = X_train_sere.copy()
X_test_seresc = X_test_se.copy()

cont_cols = []
for col_name in X_train_seresc.columns:
    if X_train_seresc[col_name].nunique() > 3:
```

```
            cont_cols.append(col_name)

for col in cont_cols:
    sc = StandardScaler()
    sc.fit(X_train_seresc[col].values.reshape(-1, 1))
    X_train_seresc[col] = sc.transform(X_train_seresc[col].values.reshape(-1, 1))

    sc.fit(X_test_seresc[col].values.reshape(-1, 1))
    X_test_seresc[col] = sc.transform(X_test_seresc[col].values.reshape(-1, 1))

# 4. RFECV, then StandardScaler
X_train_sesc = X_train_se.copy()
X_test_sesc = X_test_se.copy()

cont_cols = []
for col_name in X_train_sesc.columns:
    if X_train_sesc[col_name].nunique() > 3:
        cont_cols.append(col_name)

for col in cont_cols:
    sc = StandardScaler()
    sc.fit(X_train_sesc[col].values.reshape(-1, 1))
    X_train_sesc[col] = sc.transform(X_train_sesc[col].values.reshape(-1, 1))

    sc.fit(X_test_sesc[col].values.reshape(-1, 1))
    X_test_sesc[col] = sc.transform(X_test_sesc[col].values.reshape(-1, 1))

# 5. RFECV, then StandardScaler, then SMOTE
sm = SMOTE(random_state = 42)
X_train_sescre, y_train_sescre = sm.fit_resample(X_train_sesc, y_train.ravel())

# END OF FEATURE SELECTION as the base algorithm
    ================================================================================


# START OF TRAINING AND TESTING OF THE 4 MODELS ON THE 5 RFECV-FIRST DATASETS
    ==============================================
scoreDF = pd.DataFrame(columns=['accuracy','precision','recall','f1 "," roc_auc","model","dataset"])

# 1. RFECV only
rf = RandomForestClassifier()
rf.fit(X_train_se, y_train)
RF_pred = rf.predict(X_test_se)
RFaccuracy,RFaucroc = accuracy_score(y_test, RF_pred),roc_auc_score(y_test, RF_pred, average='macro')
RFprecision,RFrecall,RFf1 = precision_score(y_test, RF_pred, average='macro'),recall_score(y_test, RF_pred, average='macro'),
    f1_score(y_test, RF_pred, average='macro')
data =[RFaccuracy,RFprecision,RFrecall,RFf1,RFaucroc,"RF","RFECV",]
scoreDF = pd.concat([scoreDF, pd.DataFrame([data], columns=scoreDF.columns)], ignore_index=True)

xgb_cls = xgb.XGBClassifier()
xgb_cls.fit(X_train_se, y_train)
XGB_pred = xgb_cls.predict(X_test_se)
XGBaccuracy,XGBaucroc = accuracy_score(y_test, XGB_pred),roc_auc_score(y_test, XGB_pred, average='macro')
XGBprecision,XGBrecall,XGBf1 = precision_score(y_test, XGB_pred, average='macro'),recall_score(y_test, XGB_pred, average='
    macro'),f1_score(y_test, XGB_pred, average='macro')
data =[XGBaccuracy,XGBprecision,XGBrecall,XGBf1,XGBaucroc,"XGB","RFECV",]
scoreDF = pd.concat([scoreDF, pd.DataFrame([data], columns=scoreDF.columns)], ignore_index=True)

lr = LogisticRegression(max_iter=10000)
lr.fit(X_train_se, y_train)
LR_pred = lr.predict(X_test_se)
LRaccuracy,LRaucroc = accuracy_score(y_test, LR_pred),roc_auc_score(y_test, LR_pred, average='macro')
LRprecision,LRrecall,LRf1 = precision_score(y_test, LR_pred, average='macro'),recall_score(y_test, LR_pred, average='macro'),
    f1_score(y_test, LR_pred, average='macro')
data =[LRaccuracy,LRprecision,LRrecall,LRf1,LRaucroc,"LR","RFECV",]
scoreDF = pd.concat([scoreDF, pd.DataFrame([data], columns=scoreDF.columns)], ignore_index=True)

svc=svm.SVC()
svc.fit(X_train_se, y_train)
SVM_prediction = svc.predict(X_test_se)
SVMaccuracy,SVMaucroc = accuracy_score(y_test, SVM_prediction),roc_auc_score(y_test, SVM_prediction, average='macro')
SVMprecision,SVMrecall,SVMf1 = precision_score(y_test, SVM_prediction, average='macro'),recall_score(y_test, SVM_prediction,
    average='macro'),f1_score(y_test, SVM_prediction, average='macro')
data =[SVMaccuracy,SVMprecision,SVMrecall,SVMf1,SVMaucroc,"SVM","RFECV",]
scoreDF = pd.concat([scoreDF, pd.DataFrame([data], columns=scoreDF.columns)], ignore_index=True)


# 2. RFECV, then SMOTE
rf = RandomForestClassifier()
rf.fit(X_train_sere, y_train_sere)
RF_pred = rf.predict(X_test_se)
RFaccuracy,RFaucroc = accuracy_score(y_test, RF_pred),roc_auc_score(y_test, RF_pred, average='macro')
RFprecision,RFrecall,RFf1 = precision_score(y_test, RF_pred, average='macro'),recall_score(y_test, RF_pred, average='macro'),
    f1_score(y_test, RF_pred, average='macro')
data =[RFaccuracy,RFprecision,RFrecall,RFf1,RFaucroc,"RF","RFECV-SMOTE",]
scoreDF = pd.concat([scoreDF, pd.DataFrame([data], columns=scoreDF.columns)], ignore_index=True)

xgb_cls = xgb.XGBClassifier()
xgb_cls.fit(X_train_sere, y_train_sere)
XGB_pred = xgb_cls.predict(X_test_se)
XGBaccuracy,XGBaucroc = accuracy_score(y_test, XGB_pred),roc_auc_score(y_test, XGB_pred, average='macro')
```

```
XGBprecision,XGBrecall,XGBf1 = precision_score(y_test, XGB_pred, average='macro'),recall_score(y_test, XGB_pred, average='macro'),f1_score(y_test, XGB_pred, average='macro')
data =[XGBaccuracy,XGBprecision,XGBrecall,XGBf1,XGBaucroc,"XGB","RFECV-SMOTE",]
scoreDF = pd.concat([scoreDF, pd.DataFrame([data], columns=scoreDF.columns)], ignore_index=True)

lr = LogisticRegression(max_iter=10000)
lr.fit(X_train_sere, y_train_sere)
LR_pred = lr.predict(X_test_se)
LRaccuracy,LRaucroc = accuracy_score(y_test, LR_pred),roc_auc_score(y_test, LR_pred, average='macro')
LRprecision,LRrecall,LRf1 = precision_score(y_test, LR_pred, average='macro'),recall_score(y_test, LR_pred, average='macro'),f1_score(y_test, LR_pred, average='macro')
data =[LRaccuracy,LRprecision,LRrecall,LRf1,LRaucroc,"LR","RFECV-SMOTE",]
scoreDF = pd.concat([scoreDF, pd.DataFrame([data], columns=scoreDF.columns)], ignore_index=True)

svc=svm.SVC()
svc.fit(X_train_sere, y_train_sere)
SVM_prediction = svc.predict(X_test_se)
SVMaccuracy,SVMaucroc = accuracy_score(y_test, SVM_prediction),roc_auc_score(y_test, SVM_prediction, average='macro')
SVMprecision,SVMrecall,SVMf1 = precision_score(y_test, SVM_prediction, average='macro'),recall_score(y_test, SVM_prediction, average='macro'),f1_score(y_test, SVM_prediction, average='macro')
data =[SVMaccuracy,SVMprecision,SVMrecall,SVMf1,SVMaucroc,"SVM","RFECV-SMOTE",]
scoreDF = pd.concat([scoreDF, pd.DataFrame([data], columns=scoreDF.columns)], ignore_index=True)


# 3. RFECV, then SMOTE, then StandardScaler
rf = RandomForestClassifier()
rf.fit(X_train_seresc, y_train_sere)
RF_pred = rf.predict(X_test_seresc)
RFaccuracy,RFaucroc = accuracy_score(y_test, RF_pred),roc_auc_score(y_test, RF_pred, average='macro')
RFprecision,RFrecall,RFf1 = precision_score(y_test, RF_pred, average='macro'),recall_score(y_test, RF_pred, average='macro'),f1_score(y_test, RF_pred, average='macro')
data =[RFaccuracy,RFprecision,RFrecall,RFf1,RFaucroc,"RF","RFECV-SMOTE-StandardScaler",]
scoreDF = pd.concat([scoreDF, pd.DataFrame([data], columns=scoreDF.columns)], ignore_index=True)

xgb_cls = xgb.XGBClassifier()
xgb_cls.fit(X_train_seresc, y_train_sere)
XGB_pred = xgb_cls.predict(X_test_seresc)
XGBaccuracy,XGBaucroc = accuracy_score(y_test, XGB_pred),roc_auc_score(y_test, XGB_pred, average='macro')
XGBprecision,XGBrecall,XGBf1 = precision_score(y_test, XGB_pred, average='macro'),recall_score(y_test, XGB_pred, average='macro'),f1_score(y_test, XGB_pred, average='macro')
data =[XGBaccuracy,XGBprecision,XGBrecall,XGBf1,XGBaucroc,"XGB","RFECV-SMOTE-StandardScaler",]
scoreDF = pd.concat([scoreDF, pd.DataFrame([data], columns=scoreDF.columns)], ignore_index=True)

lr = LogisticRegression(max_iter=10000)
lr.fit(X_train_seresc, y_train_sere)
LR_pred = lr.predict(X_test_seresc)
LRaccuracy,LRaucroc = accuracy_score(y_test, LR_pred),roc_auc_score(y_test, LR_pred, average='macro')
LRprecision,LRrecall,LRf1 = precision_score(y_test, LR_pred, average='macro'),recall_score(y_test, LR_pred, average='macro'),f1_score(y_test, LR_pred, average='macro')
data =[LRaccuracy,LRprecision,LRrecall,LRf1,LRaucroc,"LR","RFECV-SMOTE-StandardScaler",]
scoreDF = pd.concat([scoreDF, pd.DataFrame([data], columns=scoreDF.columns)], ignore_index=True)

svc=svm.SVC()
svc.fit(X_train_seresc, y_train_sere)
SVM_prediction = svc.predict(X_test_seresc)
SVMaccuracy,SVMaucroc = accuracy_score(y_test, SVM_prediction),roc_auc_score(y_test, SVM_prediction, average='macro')
SVMprecision,SVMrecall,SVMf1 = precision_score(y_test, SVM_prediction, average='macro'),recall_score(y_test, SVM_prediction, average='macro'),f1_score(y_test, SVM_prediction, average='macro')
data =[SVMaccuracy,SVMprecision,SVMrecall,SVMf1,SVMaucroc,"SVM","RFECV-SMOTE-StandardScaler",]
scoreDF = pd.concat([scoreDF, pd.DataFrame([data], columns=scoreDF.columns)], ignore_index=True)


# 4. RFECV, then StandardScaler
rf = RandomForestClassifier()
rf.fit(X_train_sesc, y_train)
RF_pred = rf.predict(X_test_sesc)
RFaccuracy,RFaucroc = accuracy_score(y_test, RF_pred),roc_auc_score(y_test, RF_pred, average='macro')
RFprecision,RFrecall,RFf1 = precision_score(y_test, RF_pred, average='macro'),recall_score(y_test, RF_pred, average='macro'),f1_score(y_test, RF_pred, average='macro')
data =[RFaccuracy,RFprecision,RFrecall,RFf1,RFaucroc,"RF","RFECV-StandardScaler",]
scoreDF = pd.concat([scoreDF, pd.DataFrame([data], columns=scoreDF.columns)], ignore_index=True)

xgb_cls = xgb.XGBClassifier()
xgb_cls.fit(X_train_sesc, y_train)
XGB_pred = xgb_cls.predict(X_test_sesc)
XGBaccuracy,XGBaucroc = accuracy_score(y_test, XGB_pred),roc_auc_score(y_test, XGB_pred, average='macro')
XGBprecision,XGBrecall,XGBf1 = precision_score(y_test, XGB_pred, average='macro'),recall_score(y_test, XGB_pred, average='macro'),f1_score(y_test, XGB_pred, average='macro')
data =[XGBaccuracy,XGBprecision,XGBrecall,XGBf1,XGBaucroc,"XGB","RFECV-StandardScaler",]
scoreDF = pd.concat([scoreDF, pd.DataFrame([data], columns=scoreDF.columns)], ignore_index=True)

lr = LogisticRegression(max_iter=10000)
lr.fit(X_train_sesc, y_train)
LR_pred = lr.predict(X_test_sesc)
LRaccuracy,LRaucroc = accuracy_score(y_test, LR_pred),roc_auc_score(y_test, LR_pred, average='macro')
LRprecision,LRrecall,LRf1 = precision_score(y_test, LR_pred, average='macro'),recall_score(y_test, LR_pred, average='macro'),f1_score(y_test, LR_pred, average='macro')
data =[LRaccuracy,LRprecision,LRrecall,LRf1,LRaucroc,"LR","RFECV-StandardScaler",]
scoreDF = pd.concat([scoreDF, pd.DataFrame([data], columns=scoreDF.columns)], ignore_index=True)

svc=svm.SVC()
svc.fit(X_train_sesc, y_train)
```

```
SVM_prediction = svc.predict(X_test_sesc)
SVMaccuracy,SVMaucroc = accuracy_score(y_test, SVM_prediction),roc_auc_score(y_test, SVM_prediction, average='macro')
SVMprecision,SVMrecall,SVMf1 = precision_score(y_test, SVM_prediction, average='macro'),recall_score(y_test, SVM_prediction,
    average='macro'),f1_score(y_test, SVM_prediction, average='macro')
data =[SVMaccuracy,SVMprecision,SVMrecall,SVMf1,SVMaucroc,"SVM","RFECV-StandardScaler",]
scoreDF = pd.concat([scoreDF, pd.DataFrame([data], columns=scoreDF.columns)], ignore_index=True)


# 5. RFECV, then StandardScaler, then SMOTE
rf = RandomForestClassifier()
rf.fit(X_train_sescre, y_train_sescre)
RF_pred = rf.predict(X_test_sesc)
RFaccuracy,RFaucroc = accuracy_score(y_test, RF_pred),roc_auc_score(y_test, RF_pred, average='macro')
RFprecision,RFrecall,RFf1 = precision_score(y_test, RF_pred, average='macro'),recall_score(y_test, RF_pred, average='macro'),
    f1_score(y_test, RF_pred, average='macro')
data =[RFaccuracy,RFprecision,RFrecall,RFf1,RFaucroc,"RF","RFECV-StandardScaler-SMOTE",]
scoreDF = pd.concat([scoreDF, pd.DataFrame([data], columns=scoreDF.columns)], ignore_index=True)

xgb_cls = xgb.XGBClassifier()
xgb_cls.fit(X_train_sescre, y_train_sescre)
XGB_pred = xgb_cls.predict(X_test_sesc)
XGBaccuracy,XGBaucroc = accuracy_score(y_test, XGB_pred),roc_auc_score(y_test, XGB_pred, average='macro')
XGBprecision,XGBrecall,XGBf1 = precision_score(y_test, XGB_pred, average='macro'),recall_score(y_test, XGB_pred, average='
    macro'),f1_score(y_test, XGB_pred, average='macro')
data =[XGBaccuracy,XGBprecision,XGBrecall,XGBf1,XGBaucroc,"XGB","RFECV-StandardScaler-SMOTE",]
scoreDF = pd.concat([scoreDF, pd.DataFrame([data], columns=scoreDF.columns)], ignore_index=True)

lr = LogisticRegression(max_iter=10000)
lr.fit(X_train_sescre, y_train_sescre)
LR_pred = lr.predict(X_test_sesc)
LRaccuracy,LRaucroc = accuracy_score(y_test, LR_pred),roc_auc_score(y_test, LR_pred, average='macro')
LRprecision,LRrecall,LRf1 = precision_score(y_test, LR_pred, average='macro'),recall_score(y_test, LR_pred, average='macro'),
    f1_score(y_test, LR_pred, average='macro')
data =[LRaccuracy,LRprecision,LRrecall,LRf1,LRaucroc,"LR","RFECV-StandardScaler-SMOTE",]
scoreDF = pd.concat([scoreDF, pd.DataFrame([data], columns=scoreDF.columns)], ignore_index=True)

svc=svm.SVC()
svc.fit(X_train_sescre, y_train_sescre)
SVM_prediction = svc.predict(X_test_sesc)
SVMaccuracy,SVMaucroc = accuracy_score(y_test, SVM_prediction),roc_auc_score(y_test, SVM_prediction, average='macro')
SVMprecision,SVMrecall,SVMf1 = precision_score(y_test, SVM_prediction, average='macro'),recall_score(y_test, SVM_prediction,
    average='macro'),f1_score(y_test, SVM_prediction, average='macro')
data =[SVMaccuracy,SVMprecision,SVMrecall,SVMf1,SVMaucroc,"SVM","RFECV-StandardScaler-SMOTE",]
scoreDF = pd.concat([scoreDF, pd.DataFrame([data], columns=scoreDF.columns)], ignore_index=True)

# Display the table of performance metrics
scoreDF.to_csv('RFECV-first.csv')
scoreDF

# END OF TRAINING AND TESTING OF THE 4 MODELS ON THE 5 RFECV-FIRST DATASETS
    ================================================


# START OF FEATURE SCALING as the base algorithm
    ================================================================================


# 1. StandardScaler only
X_train_sc = imputed_X_train.copy()
X_test_sc = imputed_X_test.copy()

cont_cols = []
for col_name in X_train_sc.columns:
    if X_train_sc[col_name].nunique() > 3:
        cont_cols.append(col_name)

for col in cont_cols:
    sc = StandardScaler()
    sc.fit(X_train_sc[col].values.reshape(-1, 1))
    X_train_sc[col] = sc.transform(X_train_sc[col].values.reshape(-1, 1))

    sc.fit(X_test_sc[col].values.reshape(-1, 1))
    X_test_sc[col] = sc.transform(X_test_sc[col].values.reshape(-1, 1))

# 2. StandardScaler, then SMOTE
sm = SMOTE(random_state = 42)
X_train_scre, y_train_scre = sm.fit_resample(X_train_sc, y_train.ravel())

# 3. StandardScaler, then SMOTE, then RFECV
clf = RFECV(RandomForestClassifier(), cv=StratifiedKFold(10), scoring='roc_auc')
clf = clf.fit(X_train_scre, y_train_scre)
selected_features = clf.get_support(1)
X_train_screse = X_train_scre[X_train_scre.columns[selected_features]]
X_test_screse = X_test_sc[X_test_sc.columns[selected_features]]

# 4. StandardScaler, then RFECV
clf = RFECV(RandomForestClassifier(), cv=StratifiedKFold(10), scoring='roc_auc')
clf = clf.fit(X_train_sc, y_train)
selected_features = clf.get_support(1)
X_train_scse = X_train_sc[X_train_sc.columns[selected_features]]
X_test_scse = X_test_sc[X_test_sc.columns[selected_features]]

# 5. StandardScaler, then RFECV, then SMOTE
```

```
753 sm = SMOTE(random_state = 42)
754 X_train_scsere, y_train_scsere = sm.fit_resample(X_train_scse, y_train.ravel())
755
756 # END OF FEATURE SCALING as the base algorithm
        ================================================================================

757
758
759 # START OF TRAINING AND TESTING OF THE 4 MODELS ON THE 5 STANDARDSCALER-FIRST DATASETS
        ======================================
760 scoreDF = pd.DataFrame(columns=['accuracy','precision','recall','f1 '," roc_auc","model","dataset"])
761
762 # 1. StandardScaler only
763 rf4 = RandomForestClassifier()
764 rf4.fit(X_train_sc, y_train)
765 RF_pred = rf4.predict(X_test_sc)
766 RFaccuracy,RFaucroc = accuracy_score(y_test, RF_pred),roc_auc_score(y_test, RF_pred, average='macro')
767 RFprecision,RFrecall,RFf1 = precision_score(y_test, RF_pred, average='macro'),recall_score(y_test, RF_pred, average='macro'),
        f1_score(y_test, RF_pred, average='macro')
768 data =[RFaccuracy,RFprecision,RFrecall,RFf1,RFaucroc,"RF","StandardScaler",]
769 scoreDF = pd.concat([scoreDF, pd.DataFrame([data], columns=scoreDF.columns)], ignore_index=True)
770
771 xgb_cls = xgb.XGBClassifier()
772 xgb_cls.fit(X_train_sc, y_train)
773 XGB_pred = xgb_cls.predict(X_test_sc)
774 XGBaccuracy,XGBaucroc = accuracy_score(y_test, XGB_pred),roc_auc_score(y_test, XGB_pred, average='macro')
775 XGBprecision,XGBrecall,XGBf1 = precision_score(y_test, XGB_pred, average='macro'),recall_score(y_test, XGB_pred, average='
        macro'),f1_score(y_test, XGB_pred, average='macro')
776 data =[XGBaccuracy,XGBprecision,XGBrecall,XGBf1,XGBaucroc,"XGB","StandardScaler",]
777 scoreDF = pd.concat([scoreDF, pd.DataFrame([data], columns=scoreDF.columns)], ignore_index=True)
778
779 lr = LogisticRegression(max_iter=10000)
780 lr.fit(X_train_sc, y_train)
781 LR_pred = lr.predict(X_test_sc)
782 LRaccuracy,LRaucroc = accuracy_score(y_test, LR_pred),roc_auc_score(y_test, LR_pred, average='macro')
783 LRprecision,LRrecall,LRf1 = precision_score(y_test, LR_pred, average='macro'),recall_score(y_test, LR_pred, average='macro'),
        f1_score(y_test, LR_pred, average='macro')
784 data =[LRaccuracy,LRprecision,LRrecall,LRf1,LRaucroc,"LR","StandardScaler",]
785 scoreDF = pd.concat([scoreDF, pd.DataFrame([data], columns=scoreDF.columns)], ignore_index=True)
786
787 svc=svm.SVC()
788 svc.fit(X_train_sc, y_train)
789 SVM_prediction = svc.predict(X_test_sc)
790 SVMaccuracy,SVMaucroc = accuracy_score(y_test, SVM_prediction),roc_auc_score(y_test, SVM_prediction, average='macro')
791 SVMprecision,SVMrecall,SVMf1 = precision_score(y_test, SVM_prediction, average='macro'),recall_score(y_test, SVM_prediction,
        average='macro'),f1_score(y_test, SVM_prediction, average='macro')
792 data =[SVMaccuracy,SVMprecision,SVMrecall,SVMf1,SVMaucroc,"SVM","StandardScaler",]
793 scoreDF = pd.concat([scoreDF, pd.DataFrame([data], columns=scoreDF.columns)], ignore_index=True)
794
795
796 # 2. StandardScaler, then SMOTE
797 rf4 = RandomForestClassifier()
798 rf4.fit(X_train_scre, y_train_scre)
799 RF_pred = rf4.predict(X_test_sc)
800 RFaccuracy,RFaucroc = accuracy_score(y_test, RF_pred),roc_auc_score(y_test, RF_pred, average='macro')
801 RFprecision,RFrecall,RFf1 = precision_score(y_test, RF_pred, average='macro'),recall_score(y_test, RF_pred, average='macro'),
        f1_score(y_test, RF_pred, average='macro')
802 data =[RFaccuracy,RFprecision,RFrecall,RFf1,RFaucroc,"RF","StandardScaler-SMOTE",]
803 scoreDF = pd.concat([scoreDF, pd.DataFrame([data], columns=scoreDF.columns)], ignore_index=True)
804
805 xgb_cls = xgb.XGBClassifier()
806 xgb_cls.fit(X_train_scre, y_train_scre)
807 XGB_pred = xgb_cls.predict(X_test_sc)
808 XGBaccuracy,XGBaucroc = accuracy_score(y_test, XGB_pred),roc_auc_score(y_test, XGB_pred, average='macro')
809 XGBprecision,XGBrecall,XGBf1 = precision_score(y_test, XGB_pred, average='macro'),recall_score(y_test, XGB_pred, average='
        macro'),f1_score(y_test, XGB_pred, average='macro')
810 data =[XGBaccuracy,XGBprecision,XGBrecall,XGBf1,XGBaucroc,"XGB","StandardScaler-SMOTE",]
811 scoreDF = pd.concat([scoreDF, pd.DataFrame([data], columns=scoreDF.columns)], ignore_index=True)
812
813 lr = LogisticRegression(max_iter=10000)
814 lr.fit(X_train_scre, y_train_scre)
815 LR_pred = lr.predict(X_test_sc)
816 LRaccuracy,LRaucroc = accuracy_score(y_test, LR_pred),roc_auc_score(y_test, LR_pred, average='macro')
817 LRprecision,LRrecall,LRf1 = precision_score(y_test, LR_pred, average='macro'),recall_score(y_test, LR_pred, average='macro'),
        f1_score(y_test, LR_pred, average='macro')
818 data =[LRaccuracy,LRprecision,LRrecall,LRf1,LRaucroc,"LR","StandardScaler-SMOTE",]
819 scoreDF = pd.concat([scoreDF, pd.DataFrame([data], columns=scoreDF.columns)], ignore_index=True)
820
821 svc=svm.SVC()
822 svc.fit(X_train_scre, y_train_scre)
823 SVM_prediction = svc.predict(X_test_sc)
824 SVMaccuracy,SVMaucroc = accuracy_score(y_test, SVM_prediction),roc_auc_score(y_test, SVM_prediction, average='macro')
825 SVMprecision,SVMrecall,SVMf1 = precision_score(y_test, SVM_prediction, average='macro'),recall_score(y_test, SVM_prediction,
        average='macro'),f1_score(y_test, SVM_prediction, average='macro')
826 data =[SVMaccuracy,SVMprecision,SVMrecall,SVMf1,SVMaucroc,"SVM","StandardScaler-SMOTE",]
827 scoreDF = pd.concat([scoreDF, pd.DataFrame([data], columns=scoreDF.columns)], ignore_index=True)
828
829
830 # 3. StandardScaler, then SMOTE, then RFECV
831 rf4 = RandomForestClassifier()
832 rf4.fit(X_train_screse, y_train_scre)
833 RF_pred = rf4.predict(X_test_screse)
```

```
RFaccuracy,RFaucroc = accuracy_score(y_test, RF_pred),roc_auc_score(y_test, RF_pred, average='macro')
RFprecision,RFrecall,RFf1 = precision_score(y_test, RF_pred, average='macro'),recall_score(y_test, RF_pred, average='macro'),
    f1_score(y_test, RF_pred, average='macro')
data =[RFaccuracy,RFprecision,RFrecall,RFf1,RFaucroc,"RF","StandardScaler-SMOTE-RFECV",]
scoreDF = pd.concat([scoreDF, pd.DataFrame([data], columns=scoreDF.columns)], ignore_index=True)

xgb_cls = xgb.XGBClassifier()
xgb_cls.fit(X_train_screse, y_train_scre)
XGB_pred = xgb_cls.predict(X_test_screse)
XGBaccuracy,XGBaucroc = accuracy_score(y_test, XGB_pred),roc_auc_score(y_test, XGB_pred, average='macro')
XGBprecision,XGBrecall,XGBf1 = precision_score(y_test, XGB_pred, average='macro'),recall_score(y_test, XGB_pred, average='
    macro'),f1_score(y_test, XGB_pred, average='macro')
data =[XGBaccuracy,XGBprecision,XGBrecall,XGBf1,XGBaucroc,"XGB","StandardScaler-SMOTE-RFECV",]
scoreDF = pd.concat([scoreDF, pd.DataFrame([data], columns=scoreDF.columns)], ignore_index=True)

lr = LogisticRegression(max_iter=10000)
lr.fit(X_train_screse, y_train_scre)
LR_pred = lr.predict(X_test_screse)
LRaccuracy,LRaucroc = accuracy_score(y_test, LR_pred),roc_auc_score(y_test, LR_pred, average='macro')
LRprecision,LRrecall,LRf1 = precision_score(y_test, LR_pred, average='macro'),recall_score(y_test, LR_pred, average='macro'),
    f1_score(y_test, LR_pred, average='macro')
data =[LRaccuracy,LRprecision,LRrecall,LRf1,LRaucroc,"LR","StandardScaler-SMOTE-RFECV",]
scoreDF = pd.concat([scoreDF, pd.DataFrame([data], columns=scoreDF.columns)], ignore_index=True)

svc=svm.SVC()
svc.fit(X_train_screse, y_train_scre)
SVM_prediction = svc.predict(X_test_screse)
SVMaccuracy,SVMaucroc = accuracy_score(y_test, SVM_prediction),roc_auc_score(y_test, SVM_prediction, average='macro')
SVMprecision,SVMrecall,SVMf1 = precision_score(y_test, SVM_prediction, average='macro'),recall_score(y_test, SVM_prediction,
    average='macro'),f1_score(y_test, SVM_prediction, average='macro')
data =[SVMaccuracy,SVMprecision,SVMrecall,SVMf1,SVMaucroc,"SVM","StandardScaler-SMOTE-RFECV",]
scoreDF = pd.concat([scoreDF, pd.DataFrame([data], columns=scoreDF.columns)], ignore_index=True)


# 4. StandardScaler, then RFECV
rf4 = RandomForestClassifier()
rf4.fit(X_train_scse, y_train)
RF_pred = rf4.predict(X_test_scse)
RFaccuracy,RFaucroc = accuracy_score(y_test, RF_pred),roc_auc_score(y_test, RF_pred, average='macro')
RFprecision,RFrecall,RFf1 = precision_score(y_test, RF_pred, average='macro'),recall_score(y_test, RF_pred, average='macro'),
    f1_score(y_test, RF_pred, average='macro')
data =[RFaccuracy,RFprecision,RFrecall,RFf1,RFaucroc,"RF","StandardScaler-RFECV",]
scoreDF = pd.concat([scoreDF, pd.DataFrame([data], columns=scoreDF.columns)], ignore_index=True)

xgb_cls = xgb.XGBClassifier()
xgb_cls.fit(X_train_scse, y_train)
XGB_pred = xgb_cls.predict(X_test_scse)
XGBaccuracy,XGBaucroc = accuracy_score(y_test, XGB_pred),roc_auc_score(y_test, XGB_pred, average='macro')
XGBprecision,XGBrecall,XGBf1 = precision_score(y_test, XGB_pred, average='macro'),recall_score(y_test, XGB_pred, average='
    macro'),f1_score(y_test, XGB_pred, average='macro')
data =[XGBaccuracy,XGBprecision,XGBrecall,XGBf1,XGBaucroc,"XGB","StandardScaler-RFECV",]
scoreDF = pd.concat([scoreDF, pd.DataFrame([data], columns=scoreDF.columns)], ignore_index=True)

lr = LogisticRegression(max_iter=10000)
lr.fit(X_train_scse, y_train)
LR_pred = lr.predict(X_test_scse)
LRaccuracy,LRaucroc = accuracy_score(y_test, LR_pred),roc_auc_score(y_test, LR_pred, average='macro')
LRprecision,LRrecall,LRf1 = precision_score(y_test, LR_pred, average='macro'),recall_score(y_test, LR_pred, average='macro'),
    f1_score(y_test, LR_pred, average='macro')
data =[LRaccuracy,LRprecision,LRrecall,LRf1,LRaucroc,"LR","StandardScaler-RFECV",]
scoreDF = pd.concat([scoreDF, pd.DataFrame([data], columns=scoreDF.columns)], ignore_index=True)

svc=svm.SVC()
svc.fit(X_train_scse, y_train)
SVM_prediction = svc.predict(X_test_scse)
SVMaccuracy,SVMaucroc = accuracy_score(y_test, SVM_prediction),roc_auc_score(y_test, SVM_prediction, average='macro')
SVMprecision,SVMrecall,SVMf1 = precision_score(y_test, SVM_prediction, average='macro'),recall_score(y_test, SVM_prediction,
    average='macro'),f1_score(y_test, SVM_prediction, average='macro')
data =[SVMaccuracy,SVMprecision,SVMrecall,SVMf1,SVMaucroc,"SVM","StandardScaler-RFECV",]
scoreDF = pd.concat([scoreDF, pd.DataFrame([data], columns=scoreDF.columns)], ignore_index=True)


# 5. StandardScaler, then RFECV, then SMOTE
rf4 = RandomForestClassifier()
rf4.fit(X_train_scsere, y_train_scsere)
RF_pred = rf4.predict(X_test_scse)
RFaccuracy,RFaucroc = accuracy_score(y_test, RF_pred),roc_auc_score(y_test, RF_pred, average='macro')
RFprecision,RFrecall,RFf1 = precision_score(y_test, RF_pred, average='macro'),recall_score(y_test, RF_pred, average='macro'),
    f1_score(y_test, RF_pred, average='macro')
data =[RFaccuracy,RFprecision,RFrecall,RFf1,RFaucroc,"RF","StandardScaler-RFECV-SMOTE",]
scoreDF = pd.concat([scoreDF, pd.DataFrame([data], columns=scoreDF.columns)], ignore_index=True)

xgb_cls = xgb.XGBClassifier()
xgb_cls.fit(X_train_scsere, y_train_scsere)
XGB_pred = xgb_cls.predict(X_test_scse)
XGBaccuracy,XGBaucroc = accuracy_score(y_test, XGB_pred),roc_auc_score(y_test, XGB_pred, average='macro')
XGBprecision,XGBrecall,XGBf1 = precision_score(y_test, XGB_pred, average='macro'),recall_score(y_test, XGB_pred, average='
    macro'),f1_score(y_test, XGB_pred, average='macro')
data =[XGBaccuracy,XGBprecision,XGBrecall,XGBf1,XGBaucroc,"XGB","StandardScaler-RFECV-SMOTE",]
scoreDF = pd.concat([scoreDF, pd.DataFrame([data], columns=scoreDF.columns)], ignore_index=True)

lr = LogisticRegression(max_iter=10000)
```

```
lr.fit(X_train_scsere, y_train_scsere)
LR_pred = lr.predict(X_test_scse)
LRaccuracy,LRaucroc = accuracy_score(y_test, LR_pred),roc_auc_score(y_test, LR_pred, average='macro')
LRprecision,LRrecall,LRf1 = precision_score(y_test, LR_pred, average='macro'),recall_score(y_test, LR_pred, average='macro'),
    f1_score(y_test, LR_pred, average='macro')
data =[LRaccuracy,LRprecision,LRrecall,LRf1,LRaucroc,"LR","StandardScaler-RFECV-SMOTE",]
scoreDF = pd.concat([scoreDF, pd.DataFrame([data], columns=scoreDF.columns)], ignore_index=True)

svc=svm.SVC()
svc.fit(X_train_scsere, y_train_scsere)
SVM_prediction = svc.predict(X_test_scse)
SVMaccuracy,SVMaucroc = accuracy_score(y_test, SVM_prediction),roc_auc_score(y_test, SVM_prediction, average='macro')
SVMprecision,SVMrecall,SVMf1 = precision_score(y_test, SVM_prediction, average='macro'),recall_score(y_test, SVM_prediction,
    average='macro'),f1_score(y_test, SVM_prediction, average='macro')
data =[SVMaccuracy,SVMprecision,SVMrecall,SVMf1,SVMaucroc,"SVM","StandardScaler-RFECV-SMOTE",]
scoreDF = pd.concat([scoreDF, pd.DataFrame([data], columns=scoreDF.columns)], ignore_index=True)

# Display the table of performance metrics
scoreDF.to_csv('StandardScaler-first.csv')
scoreDF

# END OF TRAINING AND TESTING OF THE 4 MODELS ON THE 5 STANDARDSCALER-FIRST DATASETS
    ====================================


#
    ================================================================================================

# | Hyperparameter tuning will be performed on the best versions of each of the four models.      |
# | Note: The hyperparameter values used below for each model were the ones obtained in the study. |
# | The values should be changed accordingly when the code is executed in the future since the    |
# | hyperparameter values are changing depending on the run.                                      |
#
    ================================================================================================


# START OF HYPERPARAMETER TUNING FOR RANDOM FOREST
    ============================================================================
models = {
    'Random Forest': (RandomForestClassifier(),
                      {'n_estimators': randint(20, 500),
                       'max_depth': [5, 10, 15],
                       'max_features' : [0.1, 1.0, 3.0, 5.0],
                       'min_samples_leaf' : randint(20, 200),
                       'bootstrap': [True]}),
}

for model_name, (model, param_dist) in models.items():
    # Randomized hyperparameter search using RandomizedSearchCV
    random_search = RandomizedSearchCV(model, param_distributions=param_dist, n_iter=100, cv=StratifiedKFold(10), scoring='
      roc_auc')
    random_search.fit(X_train_scsere, y_train_scsere)
    print(f"Model: {model_name}")
    print("Best parameters:", random_search.best_params_)
    print("Best cross-validation score:", random_search.best_score_)

    # Cross-validation using the best model
    best_model = random_search.best_estimator_
    cv_scores = cross_val_score(best_model, X_train_scsere, y_train_scsere, cv=StratifiedKFold(10), scoring='roc_auc')
    print("Cross-validation scores:", cv_scores)
    print("Mean cross-validation score:", cv_scores.mean())
    print("-----------------------------------------")

# Evaluating the Best RF and Optimized RF
scoreDF = pd.DataFrame(columns=['accuracy','precision','recall','f1 "," roc_auc","model","dataset"])

rf4 = RandomForestClassifier()
rf4.fit(X_train_screse, y_train_scre)
RF_pred = rf4.predict(X_test_screse)
RFaccuracy,RFaucroc = accuracy_score(y_test, RF_pred),roc_auc_score(y_test, RF_pred, average='macro')
RFprecision,RFrecall,RFf1 = precision_score(y_test, RF_pred, average='macro'),recall_score(y_test, RF_pred, average='macro'),
    f1_score(y_test, RF_pred, average='macro')
data =[RFaccuracy,RFprecision,RFrecall,RFf1,RFaucroc,"RF","StandardScaler-SMOTE-RFECV",]
scoreDF = pd.concat([scoreDF, pd.DataFrame([data], columns=scoreDF.columns)], ignore_index=True)

rf4 = RandomForestClassifier(bootstrap=True, max_depth=10, max_features=0.1, min_samples_leaf=20, n_estimators=422)
rf4.fit(X_train_screse, y_train_scre)
RF_pred = rf4.predict(X_test_screse)
RFaccuracy,RFaucroc = accuracy_score(y_test, RF_pred),roc_auc_score(y_test, RF_pred, average='macro')
RFprecision,RFrecall,RFf1 = precision_score(y_test, RF_pred, average='macro'),recall_score(y_test, RF_pred, average='macro'),
    f1_score(y_test, RF_pred, average='macro')
data =[RFaccuracy,RFprecision,RFrecall,RFf1,RFaucroc,"RF","StandardScaler-SMOTE-RFECV",]
scoreDF = pd.concat([scoreDF, pd.DataFrame([data], columns=scoreDF.columns)], ignore_index=True)

# END OF HYPERPARAMETER TUNING FOR RANDOM FOREST
    ============================================================================


# START OF HYPERPARAMETER TUNING FOR EXTREME GRADIENT BOOSTING
    ============================================================
```

```
models = {
    'XGBoost': (xgb.XGBClassifier(),
                {'objective' : ['binary:logistic ','reg:gamma','survival:cox'],
                 'n_estimators': randint(100, 500),
                 'max_depth': [3, 5, 7, 9, 10],
                 'eta' : [0.01, 0.1, 0.2, 0.5],
                 'reg_alpha' : [0, 0.01, 0.05, 0.1, 0.5, 1.0],
                 'gamma' : [0.01, 0.1, 1, 3]}),
}

# Perform randomized hyperparameter search and cross-validation for each model
for model_name, (model, param_dist) in models.items():
    # Randomized hyperparameter search using RandomizedSearchCV
    random_search = RandomizedSearchCV(model, param_distributions=param_dist, n_iter=100, cv=StratifiedKFold(10), scoring='roc_auc')
    random_search.fit(X_train_re, y_train_re)

    print(f"Model: {model_name}")
    print("Best parameters:", random_search.best_params_)
    print("Best cross-validation score:", random_search.best_score_)

    # Cross-validation using the best model
    best_model = random_search.best_estimator_
    cv_scores = cross_val_score(best_model, X_train_re, y_train_re, cv=StratifiedKFold(10), scoring='roc_auc')
    print("Cross-validation scores:", cv_scores)
    print("Mean cross-validation score:", cv_scores.mean())
    print("-----------------------------------------")

# Evaluating the Best XGBoost and Optimized XGBoost
xgb_cls_best = xgb.XGBClassifier()
xgb_cls_best.fit(X_train_re, y_train_re)
XGB_pred = xgb_cls_best.predict(imputed_X_test)
XGBaccuracy,XGBaucroc = accuracy_score(y_test, XGB_pred),roc_auc_score(y_test, XGB_pred, average='macro')
XGBprecision,XGBrecall,XGBf1 = precision_score(y_test, XGB_pred, average='macro'),recall_score(y_test, XGB_pred, average='macro'),f1_score(y_test, XGB_pred, average='macro')
data =[XGBaccuracy,XGBprecision,XGBrecall,XGBf1,XGBaucroc,"XGB","SMOTE",]
scoreDF = pd.concat([scoreDF, pd.DataFrame([data], columns=scoreDF.columns)], ignore_index=True)

xgb_cls_tuned = xgb.XGBClassifier(eta=0.5, gamma=0.01, max_depth=3, n_estimators=207, objective='binary:logistic', reg_alpha=0.05)
xgb_cls_tuned.fit(X_train_re, y_train_re)
XGB_pred = xgb_cls_tuned.predict(imputed_X_test)
XGBaccuracy,XGBaucroc = accuracy_score(y_test, XGB_pred),roc_auc_score(y_test, XGB_pred, average='macro')
XGBprecision,XGBrecall,XGBf1 = precision_score(y_test, XGB_pred, average='macro'),recall_score(y_test, XGB_pred, average='macro'),f1_score(y_test, XGB_pred, average='macro')
data =[XGBaccuracy,XGBprecision,XGBrecall,XGBf1,XGBaucroc,"XGB","SMOTE",]
scoreDF = pd.concat([scoreDF, pd.DataFrame([data], columns=scoreDF.columns)], ignore_index=True)

# END OF HYPERPARAMETER TUNING FOR EXTREME GRADIENT BOOSTING ================================================================


# START OF HYPERPARAMETER TUNING FOR SUPPORT VECTOR MACHINE ================================================================
models = {
    'SVM': (svm.SVC(),
            {'C': uniform(0.01, 10),
             'kernel': ['rbf', 'poly', 'sigmoid', 'linear'],
             'gamma' : uniform(0.01, 10),})
}

for model_name, (model, param_dist) in models.items():
    # Randomized hyperparameter search using RandomizedSearchCV
    random_search = RandomizedSearchCV(model, param_distributions=param_dist, cv=StratifiedKFold(10), scoring='roc_auc')
    random_search.fit(X_train_resesc, y_train_re)
    print(f"Model: {model_name}")
    print("Best parameters:", random_search.best_params_)
    print("Best cross-validation score:", random_search.best_score_)

    # Cross-validation using the best model
    best_model = random_search.best_estimator_
    cv_scores = cross_val_score(best_model, X_train_resesc, y_train_re, cv=StratifiedKFold(10), scoring='roc_auc')
    print("Cross-validation scores:", cv_scores)
    print("Mean cross-validation score:", cv_scores.mean())
    print("-----------------------------------------")

# Evaluating the Best SVM and Optimized SVM
svc=svm.SVC()
svc.fit(X_train_resesc, y_train_re)
SVM_prediction = svc.predict(X_test_resesc)
SVMaccuracy,SVMaucroc = accuracy_score(y_test, SVM_prediction),roc_auc_score(y_test, SVM_prediction, average='macro')
SVMprecision,SVMrecall,SVMf1 = precision_score(y_test, SVM_prediction, average='macro'),recall_score(y_test, SVM_prediction, average='macro'),f1_score(y_test, SVM_prediction, average='macro')
data =[SVMaccuracy,SVMprecision,SVMrecall,SVMf1,SVMaucroc,"SVM","SMOTE-RFECV-StandardScaler",]
scoreDF = pd.concat([scoreDF, pd.DataFrame([data], columns=scoreDF.columns)], ignore_index=True)

svc=svm.SVC(C=7.25, gamma=2.8, kernel='poly')
svc.fit(X_train_resesc, y_train_re)
SVM_prediction = svc.predict(X_test_resesc)
SVMaccuracy,SVMaucroc = accuracy_score(y_test, SVM_prediction),roc_auc_score(y_test, SVM_prediction, average='macro')
SVMprecision,SVMrecall,SVMf1 = precision_score(y_test, SVM_prediction, average='macro'),recall_score(y_test, SVM_prediction, average='macro'),f1_score(y_test, SVM_prediction, average='macro')
```

```
data =[SVMaccuracy,SVMprecision,SVMrecall,SVMf1,SVMaucroc,"SVM","SMOTE-RFECV-StandardScaler",]
scoreDF = pd.concat([scoreDF, pd.DataFrame([data], columns=scoreDF.columns)], ignore_index=True)

# END OF HYPERPARAMETER TUNING FOR SUPPORT VECTOR MACHINE
      ================================================================


# START OF HYPERPARAMETER TUNING FOR LOGISTIC REGRESSION
      ================================================================================
models = {
    'Logistic Regression': (LogisticRegression(),
                            {'C': uniform(0.01, 10),
                             'penalty': ['l1', 'l2'],
                             'solver' : ['lbfgs','liblinear'],
                             'max_iter' : [500, 1000, 5000, 10000]}),
}

for model_name, (model, param_dist) in models.items():
    # Randomized hyperparameter search using RandomizedSearchCV
    random_search = RandomizedSearchCV(model, param_distributions=param_dist, cv=StratifiedKFold(10), scoring='roc_auc')
    random_search.fit(X_train_rescse, y_train_re)
    print(f"Model: {model_name}")
    print("Best parameters:", random_search.best_params_)
    print("Best cross-validation score:", random_search.best_score_)

    # Cross-validation using the best model
    best_model = random_search.best_estimator_
    cv_scores = cross_val_score(best_model, X_train_rescse, y_train_re, cv=StratifiedKFold(10), scoring='roc_auc')
    print("Cross-validation scores:", cv_scores)
    print("Mean cross-validation score:", cv_scores.mean())
    print("-----------------------------------------")

# Evaluating the Best LR and Optimized LR
lr = LogisticRegression(max_iter=10000)
lr.fit(X_train_screse, y_train_scre)
LR_pred = lr.predict(X_test_screse)
LRaccuracy,LRaucroc = accuracy_score(y_test, LR_pred),roc_auc_score(y_test, LR_pred, average='macro')
LRprecision,LRrecall,LRf1 = precision_score(y_test, LR_pred, average='macro'),recall_score(y_test, LR_pred, average='macro'),
      f1_score(y_test, LR_pred, average='macro')
data =[LRaccuracy,LRprecision,LRrecall,LRf1,LRaucroc,"LR","StandardScaler-SMOTE-RFECV",]
scoreDF = pd.concat([scoreDF, pd.DataFrame([data], columns=scoreDF.columns)], ignore_index=True)

lr = LogisticRegression(C=0.22, max_iter=10000, penalty='l1', solver='liblinear')
lr.fit(X_train_screse, y_train_scre)
LR_pred = lr.predict(X_test_screse)
LRaccuracy,LRaucroc = accuracy_score(y_test, LR_pred),roc_auc_score(y_test, LR_pred, average='macro')
LRprecision,LRrecall,LRf1 = precision_score(y_test, LR_pred, average='macro'),recall_score(y_test, LR_pred, average='macro'),
      f1_score(y_test, LR_pred, average='macro')
data =[LRaccuracy,LRprecision,LRrecall,LRf1,LRaucroc,"LR","StandardScaler-SMOTE-RFECV",]
scoreDF = pd.concat([scoreDF, pd.DataFrame([data], columns=scoreDF.columns)], ignore_index=True)

# END OF HYPERPARAMETER TUNING FOR LOGISTIC REGRESSION
      ================================================================


# Saving and displaying the table of performance metrics for both best and tuned versions of the 4 models
scoreDF.to_csv("best_vs_tuned.csv")
scoreDF

#
      ============================================================================================

# | In the study, the best performing model was the SMOTE-only-configured XGBoost |
#
      ============================================================================================


# Saving the best XGBoost model including the data used in its configuration.
X_train_re.to_csv('X_train_resampled.csv')
imputed_X_test.to_csv('X_test.csv')
y_train_re.tofile('y_train_resampled.csv', sep = ',')
y_test.tofile('y_test.csv', sep = ',')

joblib.dump(xgb_cls_best, "XGBest.joblib")
```

source–code/modeling–files/best model.py

```
#---------------------------- IMPORTANT REMINDER ----------------------------#
#| This code is a cleaned python version of the actual notebook          |
#| file (.ipynb) that was used in the modeling process of the study.     |
#| Hence, the original outputs are not displayed. The code is formatted |
#| for a cleaner and more organized presentation in the manuscript.      |
#---------------------------------- END ---------------------------------#

# Importing Necessary Libraries
import pandas as pd
import numpy as np
from sklearn.metrics import accuracy_score, confusion_matrix, ConfusionMatrixDisplay, classification_report, roc_auc_score,
      precision_score, recall_score, f1_score
import xgboost as xgb
```

```
13 from xgboost import plot_importance
14 from scipy.stats import randint, uniform
15 import joblib
16 import pickle
17 import dill
18 import lime.lime_tabular
19 from lime.lime_tabular import LimeTabularExplainer
20 import base64
21 import io
22 from io import BytesIO
23 from PIL import Image
24 import matplotlib.pyplot as plt
25 from numpy import genfromtxt
26
27 pd.options.display.max_rows = 4200
28
29
30 # Loading the SMOTE dataset
31 X_train = pd.read_csv("X_train_resampled.csv")
32 X_train.drop('Unnamed: 0', axis=1, inplace=True)
33 y_train = genfromtxt('y_train_resampled.csv', delimiter=',')
34 X_test = pd.read_csv("X_test.csv")
35 X_test.drop('Unnamed: 0', axis=1, inplace=True)
36 y_test = genfromtxt('y_test.csv', delimiter=',')
37
38 # Loading the best XGBoost model
39 xgbest = joblib.load("XGBest.joblib")
40
41 # Testing the saved XGBoost model to check if performance is the same
42 XGB_pred = xgbest.predict(X_test)
43 XGBaccuracy,XGBaucroc = accuracy_score(y_test, XGB_pred),roc_auc_score(y_test, XGB_pred, average='macro')
44 XGBprecision,XGBrecall,XGBf1 = precision_score(y_test, XGB_pred, average='macro'),recall_score(y_test, XGB_pred, average='
       macro'),f1_score(y_test, XGB_pred, average='macro')
45 print(XGBaccuracy)
46 print(XGBprecision)
47 print(XGBrecall)
48 print(XGBf1)
49 print(XGBaucroc)
50
51 # Getting the confusion matrix of the saved XGBoost model
52 predictions = xgbest.predict(X_test)
53 cm = confusion_matrix(y_test, predictions, labels=xgbest.classes_)
54 disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=xgbest.classes_)
55 disp.plot()
56 plt.show()
57
58 # START OF FEATURE IMPORTANCE
       ======================================================================
59 # Getting the feature importance
60 ftr_imp = xgbest.feature_importances_
61 print(ftr_imp)
62
63 # Exporting the feature importance to a csv file
64 ftr_imp_list = pd.DataFrame(columns=['Feature','Importance'])
65 for i in range(69):
66     data = [X_train.columns[i], ftr_imp[i]]
67     ftr_imp_list = pd.concat([ftr_imp_list, pd.DataFrame([data], columns=ftr_imp_list.columns)], ignore_index=True)
68
69 ftr_imp_list.to_csv('feature_importance.csv')
70
71 # Plotting the Top 10 important features based on the 'gain' metric.
72 sorted_idx = ftr_imp.argsort()
73 reduced_idx = sorted_idx[-10:]
74 plt.barh(X_train.columns[reduced_idx], ftr_imp[reduced_idx])
75 plt.xlabel("Importance")
76 plt.ylabel("Features")
77
78 # Plotting the Top 10 important features based on the 'frequency' metric.
79 plot_importance(xgbest,max_num_features=10)
80 plt.show()
81
82 # END OF FEATURE IMPORTANCE
       ======================================================================
83
84
85 # Dropping the columns with 0 importance based on gain metric except for the country features
86 to_drop = []
87 for index, row in ftr_imp_list.iterrows():
88     if row['Importance'] == 0:
89         if row['Feature'] != 'country_Australia' and row['Feature'] != 'country_Philippines' and row['Feature'] != 'country_Taiwan
          ':
90             to_drop.append(row['Feature'])
91
92 selected_X_train = X_train.copy()
93 selected_X_test = X_test.copy()
94 selected_X_train.drop(to_drop, axis=1, inplace=True)
95 selected_X_test.drop(to_drop, axis=1, inplace=True)
96 print(selected_X_train.shape)
97 print(selected_X_test.shape)
98 print()
99
100 # Training and testing a new XGBoost model on the selected-SMOTE dataset
```

```
# Note: The metrics scores should be the same with the saved best XGBoost model
selected_xgb = xgb.XGBClassifier()
selected_xgb.fit(selected_X_train, y_train)
selected_XGB_pred = selected_xgb.predict(selected_X_test)
sXGBaccuracy,sXGBaucroc = accuracy_score(y_test, selected_XGB_pred),roc_auc_score(y_test, selected_XGB_pred, average='macro')
sXGBprecision,sXGBrecall,sXGBf1 = precision_score(y_test, selected_XGB_pred, average='macro'),recall_score(y_test,
    selected_XGB_pred, average='macro'),f1_score(y_test, selected_XGB_pred, average='macro')
print(sXGBaccuracy)
print(sXGBprecision)
print(sXGBrecall)
print(sXGBf1)
print(sXGBaucroc)
print()

# Getting the confusion matrix of the new XGBoost model
predictions = selected_xgb.predict(selected_X_test)
cm = confusion_matrix(y_test, predictions, labels=selected_xgb.classes_)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=selected_xgb.classes_)
disp.plot()
plt.show()

# Saving the new XGBoost model including the updated data used in the configurations.
selected_X_train.to_csv('X_train_final.csv')
selected_X_test.to_csv('X_test_final.csv')
joblib.dump(selected_xgb, "XGBest_final.joblib")

# Creating a LIME model explainer using the same data from the new XGBoost model
explainer_lime = lime.lime_tabular.LimeTabularExplainer(selected_X_train.values,
                                                        feature_names=selected_X_train,
                                                        verbose=True,
                                                        class_names =['',''],
                                                        mode='classification')
# Saving the LIME model
with open('LIME_final.pkl', 'wb') as f:
    dill.dump(explainer_lime, f)

# Loading the new XGBoost model
xgbest_final = joblib.load("XGBest_final.joblib")

# Testing the new saved XGBoost model
XGB_pred = xgbest_final.predict(selected_X_test)
XGBaccuracy,XGBaucroc = accuracy_score(y_test, XGB_pred),roc_auc_score(y_test, XGB_pred, average='macro')
XGBprecision,XGBrecall,XGBf1 = precision_score(y_test, XGB_pred, average='macro'),recall_score(y_test, XGB_pred, average='
    macro'),f1_score(y_test, XGB_pred, average='macro')

scoreDF2 = pd.DataFrame(columns=['accuracy','precision','recall','f1',"roc_auc","model","dataset"])
data =[XGBaccuracy,XGBprecision,XGBrecall,XGBf1,XGBaucroc,"XGB","SMOTE",]
scoreDF2 = pd.concat([scoreDF2, pd.DataFrame([data], columns=scoreDF2.columns)], ignore_index=True)
scoreDF2

# Loading the saved LIME model
with open('LIME_final.pkl', 'rb') as f:
    explainer_lime_saved = dill.load(f)

# Testing the saved LIME model
datapoint = selected_X_test.iloc[7].values
extracted_df = pd.DataFrame(datapoint.reshape(1,-1), columns = selected_X_train.columns)

exp_lime = explainer_lime_saved.explain_instance(extracted_df.iloc[0],
                                                 xgbest_final.predict_proba,
                                                 num_features = 10)

results1 = exp_lime.as_html(labels=None,
                            predict_proba = True,
                            show_predicted_value=True)

results2 = exp_lime.as_list()

# Showing the plot of explanations provided by the LIME model
plot = exp_lime.as_pyplot_figure()
plt.rcParams["figure.figsize"] = [100, 50]
plt.rcParams["figure.autolayout"] = True
plt.figure(plot)
img_buf = io.BytesIO()
plt.savefig(img_buf,
            format='png',
            bbox_inches = 'tight')
im = Image.open(img_buf)
im.show(title="XGBoost Explanation")
img_buf.close()
```

source–code/django–files/home.html

```
{% load static %}
<html>
    <head>
        <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.2.3/dist/css/bootstrap.min.css" rel="stylesheet" integrity="sha384-
        rbsA2VBKQhggwzxH7pPCaAqO46MgnOM80zW1RWuH61DGLwZJEdK2Kadq2F9CUG65" crossorigin="anonymous">
        <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.2.3/dist/js/bootstrap.bundle.min.js" integrity="sha384-
        kenU1KFdBIe4zVF0s0G1M5b4hcpxyD9F7jL+jjXkk+Q2h455rYXK/7HAuoJl+0I4" crossorigin="anonymous"></script>
```

```
        <title>Predicting SLE Mortality Risk</title>
    </head>
    <body style="background-image: url({% static 'docpurp1.png' %}); background-size: 100%;">
        <nav class="navbar navbar-expand-lg navbar-light p-4" style="background-color:#53107d;">
            <div class="col-md-2 d-flex justify-content-left">
            </div>
            <div class="col-md-8 d-flex justify-content-center">
                <a href="{% url 'home' %}" class="navbar-brand" style="color: white; margin: 0px 20px;">Home</a>
                <a href="{% url 'dashboard' %}" class="navbar-brand" style="color: white; margin: 0px 20px;">Dashboard</a>
            </div>
            <div class="col-md-2 d-flex justify-content-end">
            </div>
        </nav>
        <div class="container">
            <div class="row align-items-center">
                <div class="col-md-6 justify-content-center">
                    <h1 style="font-size:100px;">SLEvival</h1>
                    <hr>
                    <p style="color:darkgrey; font-size: 24px;">
                        Predicting Mortality Risk in Systemic Lupus Erythematosus Patients with Explainable Machine Learning.
                    </p>
                    <br>
                    <a href="{% url 'dashboard' %}" class="btn" style="background-color:#A020F0; color: white; font-size:large;
    width:200px; vertical-align:middle"><b>GET STARTED</b></a>
                </div>
                <div class="col-md-6 d-flex justify-content-center">
                    <img src ="{% static 'docpurple.png' %}" width="75%">
                </div>
            </div>
        </div>

    </body>
</html>
```

source–code/django–files/dashboard.html

```
{% load static %}
<html>
    <head>
        <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.2.3/dist/css/bootstrap.min.css" rel="stylesheet" integrity="sha384-
    rbsA2VBKQhggwzxH7pPCaAqO46MgnOM80zW1RWuH61DGLwZJEdK2Kadq2F9CUG65" crossorigin="anonymous">
        <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.2.3/dist/js/bootstrap.bundle.min.js" integrity="sha384-
    kenU1KFdBIe4zVF0s0G1M5b4hcpxyD9F7jL+jjXkk+Q2h455rYXK/7HAuoJl+0I4" crossorigin="anonymous"></script>
        <title>Predicting SLE Mortality Risk</title>
    </head>
    <body style="background-image: url({% static 'docpurp1.png' %}); background-size: 100%;">
        <nav class="navbar navbar-expand-lg navbar-light p-4" style="background-color:#53107d;">
            <div class="col-md-2 d-flex justify-content-left">
            </div>
            <div class="col-md-8 d-flex justify-content-center">
                <a href="{% url 'home' %}" class="navbar-brand" style="color: white; margin: 0px 20px;">Home</a>
                <a href="{% url 'dashboard' %}" class="navbar-brand" style="color: white; margin: 0px 20px;">Dashboard</a>
            </div>
            <div class="col-md-2 d-flex justify-content-end">

            </div>
        </nav>

        <div>
            <div class="container justify-content-center" style="margin-top: 50px;">
                <div class="row justify-content-center" >
                        <h1 style="text-align: center;">What is SLEvival?</h1>
                        <div style="width: 50%; text-align: center;">
                            <br>
                            <a style="color:black; text-align: center;">
                                SLEvival is a web application that offers a calculating tool for predicting the mortality risk in
    SLE patients.
                                The application incorporates an XGBoost model and a LIME model explainer trained on the dataset
     from the Asia
                                Pacific Lupus Collaboration (APLC) cohort for predicting such risk in SLE patients. Model
    explanations are provided
                                to better understand the output.
                            </a>
                            <br><br>
                            <a style="color:black; text-align: center;">
                                This web application is made possible through the collaborative effort of Ivan Baluyut (Student)
    and Geoffrey Solano (Adviser)
                                of the University of the Philippines – Manila. Special thanks to Sir Nick Faelnar who provided
    useful insights in the modeling process of the system.
                            </a>
                            <br><br><br>
                            <a href="{% url 'prediction' %}" class="btn" style="background-color:#A020F0; color: white;font-
    size:large; width:200px; vertical-align:middle; text-align: center;"><b>Let's Predict!</b></a>
                        </div>
                </div>
                <br><br>
                <div class="row justify-content-center">
                    <div class="col-lg-3 col-sm-12 d-flex justify-content-center">
                        <div class="card text-center" style="width: 18rem;">
                            <div class="card-body">
```

```
                                <h5 class="card-title">Extreme Gradient Boosting</h5>
                                <p class="card-text">
                                  XGBoost is an optimized gradient boosting framework that delivers high-performance machine learning models for
                                   classification , regression , and ranking tasks.
                                </p>
                                <a href="https://xgboost.readthedocs.io/en/stable/" class="btn" style="background-color: white; border: 1px solid #A020F0; color: #A020F0;">More Info</a>
                            </div>
                        </div>
                    </div>
                    <div class="col-lg-3 col-sm-12 d-flex justify-content-center">
                        <div class="card text-center" style="width: 18rem;">
                            <div class="card-body">
                                <h5 class="card-title">LIME</h5>
                                <p class="card-text">
                                  LIME (Local Interpretable Model-Agnostic Explanations) provides locally
                                   faithful explanations for individual predictions made by complex machine learning models.
                                </p>
                                <a href="https://lime-ml.readthedocs.io/en/latest/" class="btn" style="background-color: white; border: 1px solid #A020F0; color: #A020F0;">More Info</a>
                            </div>
                        </div>
                    </div>
                </div>
            </div>
        </div>
        <br><br><br>
    </body>
</html>
```

source–code/django–files/prediction.html

```
{% load static %}
<html>
    <head>
        <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.2.3/dist/css/bootstrap.min.css" rel="stylesheet" integrity="sha384-rbsA2VBKQhggwzxH7pPCaAqO46MgnOM80zW1RWuH61DGLwZJEdK2Kadq2F9CUG65" crossorigin="anonymous">
        <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.2.3/dist/js/bootstrap.bundle.min.js" integrity="sha384-kenU1KFdBIe4zVF0s0G1M5b4hcpxyD9F7jL+jjXkk+Q2h455rYXK/7HAuoJl+0I4" crossorigin="anonymous"></script>
        <title>Predicting SLE Mortality Risk</title>
    </head>
    <body style="background-image: url({% static 'docpurp1.png' %}); background-size: 100%;">
        <nav class="navbar navbar-expand-lg navbar-light p-4" style="background-color:#53107d;">
            <div class="col-md-2 d-flex justify-content-left">
            </div>
            <div class="col-md-8 d-flex justify-content-center">
                <a href="{% url 'home' %}" class="navbar-brand" style="color: white; margin: 0px 20px;">Home</a>
                <a href="{% url 'dashboard' %}" class="navbar-brand" style="color: white; margin: 0px 20px;">Dashboard</a>
            </div>
            <div class="col-md-2 d-flex justify-content-end">

            </div>
        </nav>

        <div>
            <form action="" method="post">
                {% csrf_token %}
                <div class="container justify-content-center" style="width: 70%; margin-top: 50px;">
                    <div class="row" style="text-align: center;">
                        <h1>Patient Medical Information</h1>
                    </div>
                    <hr>
                    <div class="row gx-5">
                        <div class="col-lg-6">
                            <br>
                            <div class="row">
                                <div class="container" style=" padding: 25px; background-color: white; border: 1px solid #A020F0;">
                                    <h3>Demographics</h3>
                                    <hr>
                                    <div class="row">
                                        <div class="col-6 mb-2">
                                            <label for="formGroupExampleInput" class="form-label">Visit Duration</label>
                                            <input type="number" step="any" name="data1" class="form-control" placeholder="0" aria-label="First name" required>
                                        </div>
                                        <div class="col-6 mb-2">
                                            <label for="formGroupExampleInput" class="form-label">Age at Visit</label>
                                            <input type="number" step="any" name="data3" class="form-control" placeholder="0" aria-label="First name" required>
                                        </div>
                                        <div class="col-6 mb-2">
                                            <label for="formGroupExampleInput" class="form-label">Educational Level</label>
                                            <select class="form-select" name="data7" required>
                                                <option value="0">Primary</option>
                                                <option value="1">Secondary</option>
                                                <option value="2">Tertiary</option>
                                            </select>
                                        </div>
```

```
53                                      <div class="col-6 mb-2">
54                                          <label for="formGroupExampleInput" class="form-label">Age Diagnosis</label>
55                                          <input type="number" step="any" name="data9" class="form-control" placeholder
          ="0" aria-label="Last name" required>
56                                      </div>
57                                      <div class="col-6 mb-2">
58                                          <label for="formGroupExampleInput" class="form-label">Disease Duration at Visit
          </label>
59                                          <input type="number" step="any" name="data10" class="form-control" placeholder
          ="0" aria-label="First name" required>
60                                      </div>
61                                      <div class="col-6 mb-2">
62                                          <label for="formGroupExampleInput" class="form-label">Country</label>
63                                          <select class="form-select" name="country" required>
64                                              <option value="Indonesia">Indonesia</option>
65                                              <option value="Malaysia">Malaysia</option>
66                                              <option value="Singapore">Singapore</option>
67                                              <option value="Thailand">Thailand</option>
68                                              <option value="Austrailia">Australia</option>
69                                              <option value="Philippines">Philippines</option>
70                                              <option value="Others">Others</option>
71                                          </select>
72                                      </div>
73                                  </div>
74                              </div>
75                          </div>
76                          <br>
77                          <div class="row">
78                              <div class="container" style=" padding: 25px; background-color: white; border: 1px solid #A020F0
          ;">
79                                  <h3>SLE Information</h3>
80                                  <hr>
81                                  <div class="row">
82                                      <div class="col-6 mb-2">
83                                          <label for="formGroupExampleInput" class="form-label">ACR Score</label>
84                                          <input type="number" step="any" name="data13" class="form-control" placeholder
          ="0" aria-label="First name" required>
85                                      </div>
86                                      <div class="col-6 mb-2">
87                                          <label for="formGroupExampleInput" class="form-label">SLICC Nephritis and
          Immuno</label>
88                                          <select class="form-select" name="data14" id="" required>
89                                              <option value="0">No</option>
90                                              <option value="1">Yes</option>
91                                          </select>
92                                      </div>
93                                      <div class="col-6 mb-2">
94                                          <label for="formGroupExampleInput" class="form-label">SLICC 1+ Clinical</label
          >
95                                          <select class="form-select" name="data15" id="" required>
96                                              <option value="0">No</option>
97                                              <option value="1">Yes</option>
98                                          </select>
99                                      </div>
100                                     <div class="col-6 mb-2">
101                                         <label for="formGroupExampleInput" class="form-label">SLICC Clinical-Immuno
          </label>
102                                         <select class="form-select" name="data17" id="" required>
103                                             <option value="0">No</option>
104                                             <option value="1">Yes</option>
105                                         </select>
106                                     </div>
107                                     <div class="col-6 mb-2">
108                                         <label for="formGroupExampleInput" class="form-label">SLICC Score</label>
109                                         <input type="number" step="any" name="data19" class="form-control" placeholder
          ="0" aria-label="First name" required>
110                                     </div>
111                                     <div class="col-6 mb-2">
112                                         <label for="formGroupExampleInput" class="form-label">SLEDAI Score</label>
113                                         <input type="number" step="any" name="data30" class="form-control" placeholder
          ="0" aria-label="First name" required>
114                                     </div>
115                                     <div class="col-6 mb-2">
116                                         <label for="formGroupExampleInput" class="form-label">TAM SLEDAI</label>
117                                         <input type="number" step="any" name="data31" class="form-control" placeholder
          ="0" aria-label="First name" required>
118                                     </div>
119                                     <div class="col-6 mb-2">
120                                         <label for="formGroupExampleInput" class="form-label">AMS at Visit</label>
121                                         <input type="number" step="any" name="data32" class="form-control" placeholder
          ="0" aria-label="First name" required>
122                                     </div>
123                                     <div class="col-6 mb-2">
124                                         <label for="formGroupExampleInput" class="form-label">PGA</label>
125                                         <input type="number" step="any" name="data33" class="form-control" placeholder
          ="0" aria-label="First name" required>
126                                     </div>
127                                     <div class="col-6 mb-2">
128                                         <label for="formGroupExampleInput" class="form-label">TAM PGA</label>
129                                         <input type="number" step="any" name="data34" class="form-control" placeholder
          ="0" aria-label="First name" required>
130                                     </div>
```

```
                                    <div class="col-6 mb-2">
                                        <label for="formGroupExampleInput" class="form-label">AM PGA at Visit</label>
                                        <input type="number" step="any" name="data35" class="form-control" placeholder
        ="0" aria-label="First name" required>
                                    </div>
                                    <div class="col-6 mb-2">
                                        <label for="formGroupExampleInput" class="form-label">SF Score at Visit</label>
                                        <input type="number" step="any" name="data36" class="form-control" placeholder
        ="0" aria-label="First name" required>
                                    </div>
                                    <div class="col-6 mb-2">
                                        <label for="formGroupExampleInput" class="form-label">Any Flare Score at Visit</
        label>
                                        <input type="number" step="any" name="data37" class="form-control" placeholder
        ="0" aria-label="First name" required>
                                    </div>
                                    <div class="col-6 mb-2">
                                        <label for="formGroupExampleInput" class="form-label">MF Score at Visit</label>
                                        <input type="number" step="any" name="data38" class="form-control" placeholder
        ="0" aria-label="First name" required>
                                    </div>
                                    <div class="col-6 mb-2">
                                        <label for="formGroupExampleInput" class="form-label">SDI Total Score</label>
                                        <input type="number" step="any" name="data56" class="form-control" placeholder
        ="0" aria-label="First name" required>
                                    </div>
                                    <div class="col-6 mb-2">
                                        <label for="formGroupExampleInput" class="form-label">Overall Damage Accrual
        Score</label>
                                        <input type="number" step="any" name="data57" class="form-control" placeholder
        ="0" aria-label="First name" required>
                                    </div>

                                    <div class="col-6 mb-2">
                                        <label for="formGroupExampleInput" class="form-label">SDI Count</label>
                                        <input type="number" step="any" name="data59" class="form-control" placeholder
        ="0" aria-label="First name" required>
                                    </div>

                                    <div class="col-6 mb-2">
                                        <label for="formGroupExampleInput" class="form-label">LLDAS Percent</label>
                                        <input type="number" step="any" name="data61" class="form-control" placeholder
        ="0" aria-label="First name" required>
                                    </div>
                                    <div class="col-6 mb-2">
                                        <label for="formGroupExampleInput" class="form-label">TAM LLDAS</label>
                                        <input type="number" step="any" name="data62" class="form-control" placeholder
        ="0" aria-label="First name" required>
                                    </div>
                                </div>
                            </div>
                        </div>
                        <div class="col-lg-6">
                            <br>
                            <div class="row">
                                <div class="container" style=" padding: 25px; background-color: white; border: 1px solid #A020F0
        ;">
                                    <h3>Pathology</h3>
                                    <hr>
                                    <div class="row">
                                        <div class="col-6 mb-2">
                                            <label for="formGroupExampleInput" class="form-label">Creatinine</label>
                                            <input type="number" step="any" name="data20" class="form-control" placeholder
        ="0" aria-label="First name" required>
                                        </div>
                                        <div class="col-6 mb-2">
                                            <label for="formGroupExampleInput" class="form-label">C3</label>
                                            <input type="number" step="any" name="data21" class="form-control" placeholder
        ="0" aria-label="First name" required>
                                        </div>
                                        <div class="col-6 mb-2">
                                            <label for="formGroupExampleInput" class="form-label">Hb</label>
                                            <input type="number" step="any" name="data22" class="form-control" placeholder
        ="0" aria-label="First name" required>
                                        </div>
                                        <div class="col-6 mb-2">
                                            <label for="formGroupExampleInput" class="form-label">WCC</label>
                                            <input type="number" step="any" name="data23" class="form-control" placeholder
        ="0" aria-label="First name" required>
                                        </div>
                                        <div class="col-6 mb-2">
                                            <label for="formGroupExampleInput" class="form-label">Plt</label>
                                            <input type="number" step="any" name="data24" class="form-control" placeholder
        ="0" aria-label="First name" required>
                                        </div>
                                        <div class="col-6 mb-2">
                                            <label for="formGroupExampleInput" class="form-label">Neut</label>
                                            <input type="number" step="any" name="data25" class="form-control" placeholder
        ="0" aria-label="First name" required>
                                        </div>
                                        <div class="col-6 mb-2">
```

```
                                        <label for="formGroupExampleInput" class="form-label">Lymph</label>
                                        <input type="number" step="any" name="data26" class="form-control" placeholder
    ="0" aria-label="First name" required>
                                    </div>
                                    <div class="col-6 mb-2">
                                        <label for="formGroupExampleInput" class="form-label">Anti dsDNA</label>
                                        <input type="number" step="any" name="data27" class="form-control" placeholder
    ="0" aria-label="First name" required>
                                    </div>
                                    <div class="col-6 mb-2">
                                        <label for="formGroupExampleInput" class="form-label">Urine WBC</label>
                                        <input type="number" step="any" name="data28" class="form-control" placeholder
    ="0" aria-label="First name" required>
                                    </div>
                                    <div class="col-6 mb-2">
                                        <label for="formGroupExampleInput" class="form-label">Urine RBC</label>
                                        <input type="number" step="any" name="data29" class="form-control" placeholder
    ="0" aria-label="First name" required>
                                    </div>
                                </div>
                            </div>
                        </div>
                        <br>
                        <div class="row">
                            <div class="container" style=" padding: 25px; background-color: white; border: 1px solid #A020F0
    ;">
                                <h3>Medication</h3>
                                <hr>
                                <div class="row">
                                    <div class="col-6 mb-2">
                                        <label for="formGroupExampleInput" class="form-label">TAM-PNL at Visit</label
    >
                                        <input type="number" step="any" name="data39" class="form-control" placeholder
    ="0" aria-label="First name" required>
                                    </div>
                                    <div class="col-6 mb-2">
                                        <label for="formGroupExampleInput" class="form-label">Prednisolone (PNL)</label
    >
                                        <input type="number" step="any" name="data40" class="form-control" placeholder
    ="0" aria-label="First name" required>
                                    </div>
                                    <div class="col-6 mb-2">
                                        <label for="formGroupExampleInput" class="form-label">Cumulative PNL at Visit
    </label>
                                        <input type="number" step="any" name="data41" class="form-control" placeholder
    ="0" aria-label="First name" required>
                                    </div>
                                    <div class="col-6 mb-2">
                                        <label for="formGroupExampleInput" class="form-label">Hydroxychloroquine</label
    >
                                        <input type="number" step="any" name="data42" class="form-control" placeholder
    ="0" aria-label="First name" required>
                                    </div>
                                    <div class="col-6 mb-2">
                                        <label for="formGroupExampleInput" class="form-label">AM</label>
                                        <input type="number" step="any" name="data44" class="form-control" placeholder
    ="0" aria-label="First name" required>
                                    </div>
                                    <div class="col-6 mb-2">
                                        <label for="formGroupExampleInput" class="form-label">Azathioprine</label>
                                        <input type="number" step="any" name="data46" class="form-control" placeholder
    ="0" aria-label="First name" required>
                                    </div>
                                    <div class="col-6 mb-2">
                                        <label for="formGroupExampleInput" class="form-label">Mycophenolate</label>
                                        <input type="number" step="any" name="data47" class="form-control" placeholder
    ="0" aria-label="First name" required>
                                    </div>
                                    <div class="col-6 mb-2">
                                        <label for="formGroupExampleInput" class="form-label">MMF MPA</label>
                                        <input type="number" step="any" name="data49" class="form-control" placeholder
    ="0" aria-label="First name" required>
                                    </div>
                                    <div class="col-6 mb-2">
                                        <label for="formGroupExampleInput" class="form-label">IS</label>
                                        <input type="number" step="any" name="data52" class="form-control" placeholder
    ="0" aria-label="First name" required>
                                    </div>

                                </div>
                            </div>
                        </div>
                        <br>
                        <div class="row">
                            <a style="color:grey; text-align: justify;">
                                <b>Disclaimer:</b> The SLEvival web application offers a system for predicting the mortality
    risk of an SLE patient
                                using the 50 real patient health data entered above.
                                It should be understood that the model should only be used as a support tool to augment the
    observations of the physician
                                and not as their main basis for their medical decision-making.
                                If all inputs are correct, click the "Predict" button to view results.
```

```
                                </a>
                                <div class="container" style=" padding: 10px;">
                                    <hr>
                                    <div class="row justify-content-center">
                                        <input type="submit" name="submit" class="btn" style="background-color:#A020F0;
        color: white; font-size:large; width:200px; vertical-align:middle; text-align: center;" value="Predict">
                                    </div>
                                </div>
                            </div>
                        </div>
                    </div>
                    <br>
                </div>
            </form>
        </div>
        <br><br><br><br>
    </body>
</html>
```

source–code/django–files/results.html

```
{% load static %}
<html>
    <head>
        <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.2.3/dist/css/bootstrap.min.css" rel="stylesheet" integrity="sha384-
        rbsA2VBKQhggwzxH7pPCaAqO46MgnOM80zW1RWuH61DGLwZJEdK2Kadq2F9CUG65" crossorigin="anonymous">
        <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.2.3/dist/js/bootstrap.bundle.min.js" integrity="sha384-
        kenU1KFdBIe4zVF0s0G1M5b4hcpxyD9F7jL+jjXkk+Q2h455rYXK/7HAuoJl+0I4" crossorigin="anonymous"></script>
        <title>Predicting SLE Mortality Risk</title>
    </head>
    <body style="background-image: url({% static 'docpurp1.png' %}); background-size: 100%;">
        <nav class="navbar navbar-expand-lg navbar-light p-4" style="background-color:#53107d;">
            <div class="col-md-2 d-flex justify-content-left">
            </div>
            <div class="col-md-8 d-flex justify-content-center">
                <a href="{% url 'home' %}" class="navbar-brand" style="color: white; margin: 0px 20px;">Home</a>
                <a href="{% url 'dashboard' %}" class="navbar-brand" style="color: white; margin: 0px 20px;">Dashboard</a>
            </div>
            <div class="col-md-2 d-flex justify-content-end">

            </div>
        </nav>

        <div>
            <div class="container justify-content-center" style="margin-top: 50px;">
                <div class="row">
                    <h1>ML Model Name</h1>
                </div>
                <hr>
                <br>
                <div class="row gx-5">
                    <div class="col-lg-6">
                        <div class="container" style=" padding: 25px; background-color: #f5e6ff;">
                            <div class="row">
                                <h3>Results:</h3>
                                <hr>
                                <div class="row">
                                    <div class="col-4">
                                        <h6>Mortality Risk:</h6>
                                        <h6>Model Accuracy:</h6>
                                    </div>
                                    <div class="col-4">
                                        <h6>{{XGB_pred}}</h6>
                                        <h6>{{XGBaccuracy}}</h6>
                                    </div>
                                </div>
                            </div>
                            <br>
                            <div class="row">
                                <h3>Model Explanation:</h3>
                                <hr>
                                <div class="row">
                                    <img src="{{ image64 }}">
                                </div>
                            </div>
                        </div>
                    </div>
                    <div class="col-lg-6">
                        <h3>More Explanations:</h3>
                        <hr>
                        <div class="row">
                            {% autoescape off %}
                            {{results1}}
                            {% endautoescape %}
                        </div>
                    </div>
                </div>
                <br><br>
                <h3>Interpretation:</h3>
                <hr>
```

```
68                 <div>
69                     {% for key, value in feat_imp.items %}
70                         {% if value > 0 %}
71                             <li>The feature <b>{{ key }}</b> contributed <b>{{ value }}%</b> to the
72                                 probability of the prediction that the patient will likely <b>die.</b>
73                             </li>
74                         {% elif value < 0 %}
75                             <li>The feature <b>{{ key }}</b> contributed <b>{% widthratio value 1 -1 %}%</b> to the
76                                 probability of the prediction that the patient will likely <b>live.</b>
77                             </li>
78                         {% else %}
79                             <li>The feature <b>{{ key }}</b> contributed <b>{{ value }}%</b> to the
80                                 probability of the prediction.
81                             </li>
82                         {% endif %}
83                     {% endfor %}
84                 </div>
85             </div>
86         </div>
87         <br><br><br>
88     </body>
89 </html>
```

source–code/django–files/views.py

```
1  from django.shortcuts import render, redirect
2  import pandas as pd
3  import numpy as np
4  from sklearn.metrics import accuracy_score
5  from numpy import genfromtxt
6  import base64
7  import io
8  from io import BytesIO
9  from PIL import Image
10 import matplotlib.pyplot as plt
11 import joblib
12 import dill
13 import random
14 from django.core.cache import cache
15 from django.http import HttpResponse
16
17 def Convert(tup, di):
18     for a, b in tup:
19         a = int(a)
20         di.setdefault(a,[]).append(b)
21     return di
22
23 def fig_to_base64(fig):
24     buff = BytesIO()
25     fig.save(buff, format="PNG")
26     img_str = base64.b64encode(buff.getvalue())
27     img_str = img_str.decode("utf-8")
28     buff.close()
29     return img_str
30
31 def home(request):
32     from django.core.cache import cache
33     from django.contrib.sessions.backends.db import SessionStore
34     cache.clear()
35     random.seed(123)
36     np.random.seed(123)
37     session = SessionStore(session_key=request.session.session_key)
38     session.flush()
39     return render(request, 'home.html')
40
41 def dashboard(request):
42     return render(request, 'dashboard.html')
43
44 def prediction(request):
45     if request.method == "POST":
46             # Converting input for Country to data items
47             country = request.POST.get('country')
48             data63, data64, data65, data66, data67, data68, data69 = 0,0,0,0,0,0,0
49
50             if country == 'Indonesia':
51                 data64 = 1
52             elif country == 'Malaysia':
53                 data65 = 1
54             elif country == 'Singapore':
55                 data67 = 1
56             elif country == 'Thailand':
57                 data69 = 1
58             elif country == 'Australia':
59                 data63 = 1
60             elif country == 'Philippines':
61                 data66 = 1
62             else:
63                 data68 = 1
64
65             # Getting all inputs from the web app
```

```
66          datapoint = np.array([request.POST.get('data1'),request.POST.get('data3'),request.POST.get('data7'),request.POST.get
    ('data9'),
67                                request.POST.get('data10'),request.POST.get('data13'),request.POST.get('data14'),request.POST.
    get('data15'),
68                                request.POST.get('data17'),request.POST.get('data19'),request.POST.get('data20'),request.POST.
    get('data21'),
69                                request.POST.get('data22'),request.POST.get('data23'),request.POST.get('data24'),request.POST.
    get('data25'),
70                                request.POST.get('data26'),request.POST.get('data27'),request.POST.get('data28'),request.POST.
    get('data29'),
71                                request.POST.get('data30'),request.POST.get('data31'),request.POST.get('data32'),request.POST.
    get('data33'),
72                                request.POST.get('data34'),request.POST.get('data35'),request.POST.get('data36'),request.POST.
    get('data37'),
73                                request.POST.get('data38'),request.POST.get('data39'),request.POST.get('data40'),request.POST.
    get('data41'),
74                                request.POST.get('data42'),request.POST.get('data44'),request.POST.get('data46'),request.POST.
    get('data47'),
75                                request.POST.get('data49'),request.POST.get('data52'),request.POST.get('data56'),request.POST.
    get('data57'),
76                                request.POST.get('data59'),request.POST.get('data61'),request.POST.get('data62'),
77                                data63, data64, data65, data66, data67, data68, data69])
78
79
80          #loading the models and data
81          xgbest = joblib.load("predict_sle_mortality_app/static/XGBest_final.joblib")
82          with open('predict_sle_mortality_app/static/LIME_final.pkl', 'rb') as f:
83              explainer_lime = dill.load(f)
84          X_train = pd.read_csv('predict_sle_mortality_app/static/X_train_final.csv')
85          X_test = pd.read_csv('predict_sle_mortality_app/static/X_test_final.csv')
86          y_train = genfromtxt('predict_sle_mortality_app/static/y_train_final.csv', delimiter=',')
87          y_test = genfromtxt('predict_sle_mortality_app/static/y_test_final.csv', delimiter=',')
88
89          X_train.drop('Unnamed: 0', axis=1, inplace=True)
90          X_test.drop('Unnamed: 0', axis=1, inplace=True)
91
92          ts = datapoint.astype(float)
93          extracted_df = pd.DataFrame(ts.reshape(1,-1), columns = X_train.columns)
94
95          # Testing the saved XGBoost model
96          XGB_pred = xgbest.predict(extracted_df)
97          XGB_predtest = xgbest.predict(X_test)
98          XGBaccuracy = accuracy_score(y_test, XGB_predtest)
99          XGBaccuracy = round(XGBaccuracy*100, 2)
100         print("XGB Accuracy:", XGBaccuracy)
101
102         if(XGB_pred[0]==0):
103             XGB_pred = "No"
104         else:
105             XGB_pred = "Yes"
106
107
108         # Generating explanations through the saved LIME interpreter
109         exp_lime = explainer_lime.explain_instance(ts,
110                                                    xgbest.predict_proba,
111                                                    num_features = 10)
112
113         results1 = exp_lime.as_html(labels=None,
114                                     predict_proba = True,
115                                     show_predicted_value=True)
116
117         results2 = exp_lime.as_list()
118
119         # Formatting the features and their contributions for interpretation
120         feat_imp = {}
121         for i in range(0,len(results2)):
122             feat = results2[i][0]
123             pr = results2[i][1]
124             feature = feat.split(" ")
125             prob = round(pr*100, 2)
126             if len(feature) > 3:
127                 feat_imp.update({feature[2]:prob})
128             else:
129                 feat_imp.update({feature[0]:prob})
130
131         # Displaying the explanations as pyplot figure
132         plot = exp_lime.as_pyplot_figure()
133         plt.figure(plot)
134         plt.xlabel('', fontsize=18)
135         plt.ylabel('', fontsize=16)
136         img_buf = io.BytesIO()
137         plt.savefig(img_buf, format='png',bbox_inches = 'tight')
138         im = Image.open(img_buf)
139         image64 = fig_to_base64(im)
140
141         # Cache debugging for the plots
142         response = HttpResponse(content_type='image/png')
143         response.write(img_buf.getvalue())
144         cache.clear()
145
146         return render(request, "results.html", {'XGBaccuracy' : XGBaccuracy, 'XGB_pred' : XGB_pred,
147                                                 'image64' : image64, 'results1' : results1, 'feat_imp' : feat_imp,
```

```
148                                                 'response' : response})
149
150     return render(request, "prediction.html")
151
152 def results(request):
153      return render(request, "results.html")
```

source–code/django–files/urls.py

```
1 from django.urls import path
2 from . import views
3
4 urlpatterns = [
5     path('', views.home, name='home'),
6     path('dashboard/', views.dashboard, name='dashboard'),
7     path('prediction/', views.prediction, name='prediction'),
8     path('results/', views.results, name='results'),
9 ]
```

XI. Acknowledgment

I hereby offer my sincere gratitude and utmost appreciation to the following who have made significant contributions in the development and accomplishment of this study:

To my parents, Benson and Cristina, who have always provided so that I can focus on finishing this paper. I can't imagine having been able to accomplish this without the delicious foods you make and the comfortable rides you offer whenever I go to and come home from Manila. I can't thank you enough for the privilege, care, and understanding that you have given me. You're the greatest Mommy and Daddy I could ever ask for, so this one's for you!

To my amigos, 'mga kasamang nagpuyat kagabi', co-advisees, and friends from highschool who have always helped me emotionally and mentally to continue working on this project. Thank you for answering my programming-related questions, for letting me get a rest every now and then, for sharing the same experience, for listening to my 3am thoughts and rants, and for everything else in between. You guys played a huge part on this success, and on my college life in general.

To my adviser, Sir Geoff, for giving me the topic of the study and for being patient with me from the very start. And to my consultant, Sir Nick, who have provided useful insights in the modeling process of this study. Thank you for always pushing me beyond my capabilities and helping me understand the key concepts necessary to achieve the best for this study.

To DOST-SEI for the unwavering trust and confidence placed in me, enough to offer me a slot in their generous scholarship program and support me through financial aids from the beginning up to the very end of my college life. I am forever grateful for the transformative impact your organization has had on my educational journey.

To the creators and casts of my favorite anime and dramas, especially Kimetsu No Yaiba, for allowing me to have a breather every once in a while. Watching the

amazing productions and listening to the great soundtracks really helped me relax and refresh before preparing to work again on this paper.

To the maple syrup I always have on my blueberry pancake for always giving me just the right amount of sweetness to boost my energy and continue my studies.

And to all who have imparted their knowledge, expertise, and support, no matter how big or small. Thank you for inspiring me enough to get up to this very point of my college journey. I am beyond grateful.