Judith Osuna Ramírez

Use of artificial intelligence in the prediction of response in patients with COVID-19 pneumonia treated with low-dose anti-inflammatory radiotherapy

Final Degree Project

directed by Dr. Victor Hernandez Masgrau
and Dr. Meritxell Arenas Prat

Biomedical Engineering Degree

UNIVERSITAT ROVIRA i VIRGILI

Tarragona

2024

Dedicated
to my parents, Ramón and Rosario,
who have supported and encouraged me throughout
my entire academic journey.
To my brother Javier, to Marina,
and to my nephew Oliver, who are all so special to me.
To my university schoolmates, especially Gabriela and Òscar,
who have become significant individuals in my life.
To my lifelong friend Alba, who has been by my side since childhood.
To Dani, my best chance.
To my supervisors Víctor and Meritxell,
for believing in me.
To Alberto, the hospital IT specialist
who helped me to get started in the world of radiomic data.
To Paula, my work partner-in-crime.
To Marta and Raquel,
for their unwavering support and assistance in every possible way.
To Hatem, the professor who has given me the most confidence
and support in all the technical aspects of this work.
Lastly, to Amsterdam, for providing me with the most wonderful views
during most of the creation of this project.

Abstract

This project presents a multi-omic predictive model designed to predict the response to low-dose pulmonary radiotherapy in patients suffering from pneumonia caused by COVID-19. Clinical, metabolomic, and radiomic characteristics were acquired from 50 patients with varying responses to the same treatment. Using supervised machine learning models, the best classifier for predicting the response to this treatment was studied.

The results highlight that predictive models for binary classification achieve improved results when dealing with imbalanced response data and employing advanced feature selection methods.

This approach offers substantial advancements in predicting responses to a certain treatment, marking a significant step towards precision and personalized medicine through the use of Artificial Intelligence as a clinical support system.

Keywords: COVID-19; low-dose radiotherapy; Machine Learning; multi-omics.

Resumen

Este proyecto presenta un modelo predictivo multi-ómico diseñado para predecir la respuesta a la radioterapia pulmonar de baja dosis en pacientes que sufren neumonía causada por COVID-19. Se adquirieron características clínicas, metabolómicas y radiómicas de 50 pacientes con respuestas variables al mismo tratamiento. Mediante modelos supervisados de aprendizaje automático, se estudió cuál era el mejor clasificador para predecir la respuesta a este tratamiento.

Los resultados ponen de manifiesto que los modelos predictivos de clasificación binaria obtienen mejores resultados cuando se tratan los datos de respuesta desequilibrados y se emplean métodos avanzados de selección de características.

Esta aproximación ofrece avances sustanciales en la predicción de respuestas a un determinado tratamiento, marcando un paso significativo hacia la medicina de precisión y personalizada mediante el uso de la Inteligencia Artificial como sistema de apoyo clínico.

Palabras clave: COVID-19; radioterapia; aprendizaje supervisado; multi-ómica.

Resum

Aquest projecte presenta un model predictiu multi-òmic dissenyat per a predir la resposta a la radioteràpia pulmonar de baixa dosi en pacients que sofreixen pneumònia causada per COVID-19. Es van adquirir característiques clíniques, metabolòmiques i radiòmiques de 50 pacients amb respostes variables al mateix tractament. Mitjançant models supervisats d'aprenentatge automàtic, es va estudiar quin era el millor classificador per a predir la resposta a aquest tractament.

Els resultats posen de manifest que els models predictius de classificació binària obtenen millors resultats quan es tracten les dades de resposta desequilibrats i es fan servir mètodes avançats de selecció de característiques.

Aquesta aproximació ofereix avanços substancials en la predicció de respostes a un determinat tractament, marcant un pas significatiu cap a la medicina de precisió i personalitzada mitjançant l'ús de la Intel·ligència Artificial com a sistema de suport clínic.

Paraules clau: COVID-19; radioteràpia; aprenentatge supervisat; multi-òmica.

Table of Contents

Abbreviations

AI	Artificial Intelligence
AUC	Area Under the Curve
bpm	breaths per minute
CM	Confusion Matrix
CT	Computed Tomography
CTV	Clinical Target Volume
CV	Cross-Validation
COVID	COronaVIrus Disease 2019
DICOM	Digital Imaging and Communications in Medicine
DT	Decision Tree
EHR	Electronic Health Record
FiO_2	Fraction of Inspired Oxygen
FN	False Negatives
FP	False Positives
GDS	Global Deterioration Scale
GLCM	Gray-Level Co-Occurrence Matrix
GLDM	Gray-Level Dependence Matrix
GLRLM	Gray-Level Run-Length Matrix
GLSZM	Gray-Level Size Zone Matrix
GOT	Glutamic Oxaloacetic Transaminase
GPT	Glutamic Pyruvic Transaminase
HUSJR	Hospital Universitari Sant Joan de Reus
KDE	Kernel Density Estimate
LBP	Local Binary Patterns
LD	Low Dose
LR	Logistic Regression
ML	Machine Learning
MIQ	Mutual Information Quotient

mRMR	minimum Redundancy Maximum Relevance
NGTDM	Neighborhood Gray-Tone Difference Matrix
NIfTI	Neuroimaging Informatics Technology Initiative
PaO_2	Partial Pressure of Oxygen
PM	Predictive Model
PTV	Planning Target Volume
RF	Random Forest
ROC	Receiver Operating Characteristic
ROI	Region Of Interest
rpm	respirations per minute
RT	Radiotherapy
SMOTE	Synthetic Minority Oversampling TEchnique
SoC	Standard of Care
SpO_2	Peripheral Capillary Oxygen Saturation
TN	True Negatives
TP	True Positives
VIF	Variance Inflation Factor
WHO	World Health Organization
YAML	YAML Ain't Markup Language

1. Introduction

1.1 Background

Throughout the COVID-19 pandemic, the world faced an unprecedented challenge. The outbreak began in December of 2019 in a city called Wuhan, in China [1]. Thereafter, as illustrated in the Figure 1 provided by the World Health Organization (WHO), the coronavirus spread to every corner of the globe, with some countries being more severely affected than others, but no nation was completely spared. Since the WHO declared the global pandemic situation on the 11th of March 2020, over 775 million cases have been reported worldwide [2].

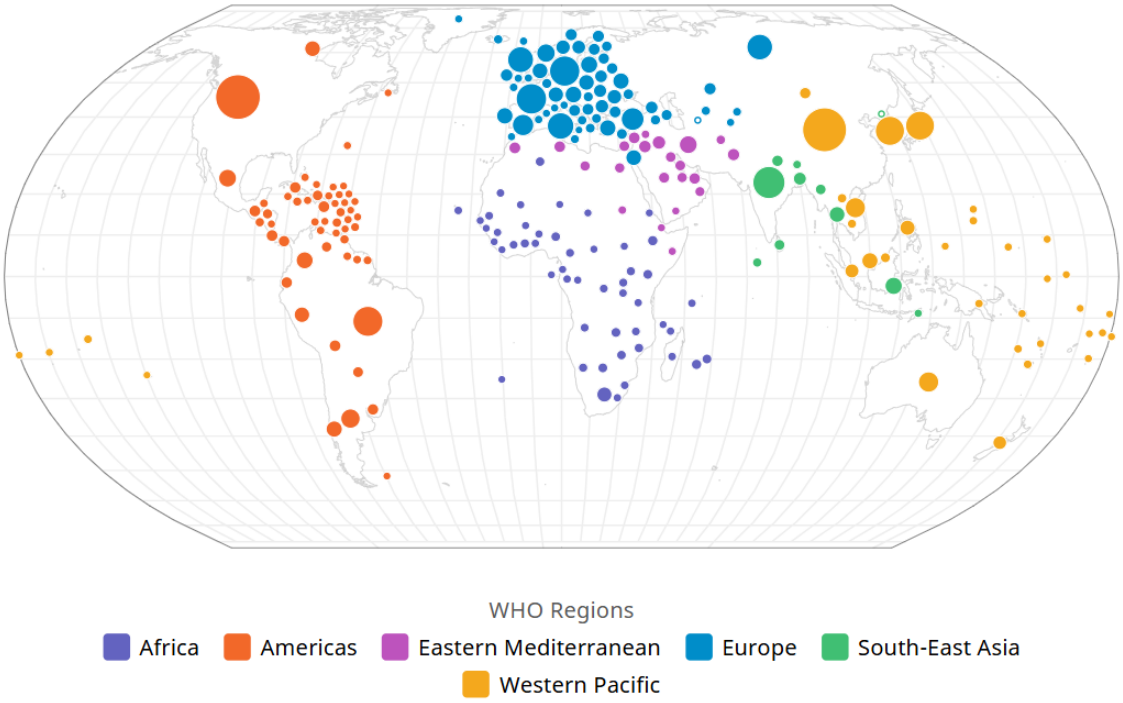


Figure 1: Total cumulative number of COVID-19 cases reported to the World Health Organization [2].

Some people were asymptomatic carriers of the virus, while others experienced mild symptoms similar to a common cold. However, many people, especially the elderly and those with pre-existing medical conditions, required prolonged hospitalization due to severe complications such as bilateral pneumonia. For these patients, the standard treatments administered included antiviral and anti-inflammatory drugs, as well as mechanical ventilation. In cases where patients were not candidates for invasive mechanical ventilation and pharmacological treatments were not sufficient to bring about recovery, an innovative approach offered new hope. This treatment consisted of a clinical trial, named IPACOVID, of total lung Low Dose Radiotherapy (LD-RT) to improve patients' oxygenation [3].

1.2 IPACOVID clinical trial

IPACOVID clinical trial is a Phase 3 non-randomized ambispective study involving 100 participants, comprising 50 control patients and 50 cases. The mean age of both cohorts was 85 years, and they were matched by sex, comorbidities, and SAFI values. The control cohort received only the standard of care (which includes antiviral or anti-inflammatory drugs). The experimantal cohort received the standard of care plus a single radiation dose of 0.5 Gy to the entire thorax [3].

As demonstrated by *Arenas et al.*[4], radiotherapy administered at low doses (LD-RT) modulates the inflammatory response with efficacy, producing an anti-inflammatory effect. This means that we can apply LD-RT to treat benign diseases such as osteoarthritis, humeral epi-condylitis, scapular–humeral periarthritis, or heel spurs.

Therefore, the clinical trial hypothesized that applying LD-RT could improve patients' respiratory function by reducing lung inflammation caused by COVID-19 pneumonia, thereby decreasing the mortality rate. The trial results showed that patients in the experimental cohort who received LD-RT in addition to Standard of Care (SoC) experienced improvements in respiratory parameters, shorter hospitalization periods, reduced 1-month mortality rates, and prolonged overall survival compared to those who received SoC alone.

1.3 Application of Artificial Intelligence in analysing patient response to LD-RT

In this project, we focused on analysing the responses to the LD-RT pulmonary treatment in the experimental cohort previously mentioned. While the vast majority of those fifty patients responded positively and recovered with the help of the medication, there were some for whom this treatment was ineffective. Our goal was to study and identify the relationships between variables in patients who responded positively and those who did not. This is helpful to better determine which patients are likely to be suitable candidates for this treatment in the future.

To achieve this, we conducted a hybrid study that integrated clinical, metabolomic, and radiomic data to develop a highly accurate and effective predictive model (PM) for assessing responses to LD-RT treatment. After the data was collected and cleaned, we applied several supervised ML algorithms to identify the one that yields the best results.

1.4 Multiomic data

All data used in this study was collected from the fifty patients participating in the IPACOVID clinical trial conducted at Hospital Universitari Sant Joan de Reus, Spain.

While many researchers and practitioners have focused on enhancing the quality of models (such as the architecture and automated feature selection) to improve the results, there are limited efforts towards improving the data quality. Nonetheless, the performance of a ML model is upper bounded by the quality of the data used [5].

As indicated above, this project combines data from various sources to gather the maximum amount of information about each patient. The following sections will explain each type of data and provide further details.

1.4.1 Clinical data

Clinical data is derived from the patient's historical information, some of which is stored in the hospital's EHR (Electronic Health Record). In this context, it encompasses personal details about the patient, including pre-existing conditions, functional assessments and respiratory parameters [6].

1.4.2 Metabolomic data

"The suffix omics refers specifically to the whole of something. Therefore, the omics sciences are those sciences that allow us to study a large number of molecules, that are involved in the functioning of an organism." [7]

In this context, metbolomic data pertains to metabolites and cellular components present in the blood that have been shown to be important in COVID-19 patients [6], as determined through blood analysis.

1.4.3 Radiomic data

Radiomics has been defined as an area of study focused on extracting quantitative metrics, known as radiomic features, from medical images. These features reflect various aspects of tissue and lesion characteristics, such as heterogeneity and shape. When used alone or together with demographic, histologic, genomic, or proteomic information, radiomic features can aid in addressing clinical challenges [8].

In this study, our medical images were simulation CT scans stored in DICOM format and carried out before the LD-RT treatment.

1.5 Clinical applicability of predictive models

One of the primary advantages of Aritificial Intelligence (AI) in the medical field is its capacity to process and analyse large volumes of clinical data with a speed and accuracy that far surpass human capabilities. This enables the identification of patterns and trends that might otherwise go unnoticed. In this specific context, AI can evaluate a multitude of clinical, metabolomic, and imaging variables to more accurately predict which patients will respond favorably to the treatment. This not only optimizes medical resources but can also significantly improve patient outcomes.

Nevertheless, this advanced tool also presents several challenges. For an AI model to be installed in a clinical setting, it must demonstrate that it ensures data security and privacy, is free from bias, and has proven efficacy [9]. In other words, it must be a reliable model. Furthermore, the implementation of this technology needs that healthcare professionals possess a thorough understanding of its functionality and limitations. It is essential that they do not place blind trust in its efficacy and remember that the final decision should always be theirs. AI should be regarded as a decision-support tool, not a replacement for clinical judgment.

2. Objectives

The main objective of this project is to develop a PM capable of determining the response to the LD-RT treatment in patients with COVID-19 pneumonia.

To achieve this, clinical and metabolomic data were collected, while radiomic features were extracted from CT images. To address the issue of data imbalance—where positive responses were more frequent than negative ones—data augmentation techniques were employed. Subsequently, two different techniques were evaluated for preprocessing and selecting non-correleted variables for model training. Finally, multiple supervised ML models were developed and compared using various evaluation metrics to assess their efficacy.

The goal is to identify the best combination of preprocessing techniques and model training methods to create an effective algorithm for predicting which future COVID-19 patients are likely to benefit from LD-RT treatment.

3. Materials and methods

The steps used to prepare the data to the PM are illustrated in Figure 2 and described in the following sections.

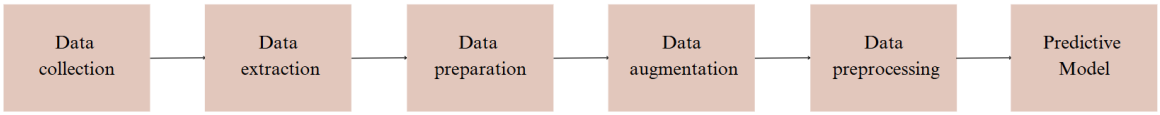


Figure 2: Diagram showing the methodology workflow.

3.1 Data collection

Both the clinical and metabolomic databases used in the present study were collected from the IPACOVID prospective clinical trial [3] and described by *Piqué et al.* [6].

Radiomic databases were not available in those datasets and were created from scratch from the CT basal images of the 50 patients.

Before proceeding, we needed to process each of the fifty CT images individually to isolate the target area that outlines the lung volume. In radiotherapy, target volumes typically include a margin around the lungs to ensure accurate radiation delivery. For our purposes, though, we are only interested in extracting and analyzing the specific lung information.

The steps to obtain the desired CT images are the following:

1. **Lung Segmentation Wizard:** The software used in the Radiotherapy department of HUSJR to prescribe treatments to patients is called *Eclipse* [10] and included a tool designed to automatically delineate the lungs. Nevertheless, this tool proved ineffective for our images, as COVID-19 patients had mucus and fluid in their lungs, resulting in abnormal lung structures. This necessitated an additional processing step.

2. **Manual adjustment of the segmentation:** With the drawing tools, we could paint and erase specific areas using the desired brush size, correcting any parts that the segmentation tool either missed or incorrectly highlighted. This permits to manually contour and refine each section of the CT scan.

3. **Post processing Tool:** This tool allowed us to remove parts of the drawing that were smaller than a specified threshold (0.5 cm^2, in this case) and retain the 'n' largest segments, in this instance, 2. This process ensures that the resulting structure accurately represents the lungs by eliminating any accidental or irrelevant portions.

Once everything had been verified, we could proceed to download the resulting anonymized image, which included the structures: Body, PTV, CTV, and, ultimately, our target structure, z_lungs, as shown in Figure 3; where PTV (Planning Target Volume) is the volume including the CTV with an additional margin to account for uncertainties in treatment delivery and CTV (Clinical Target Volume) is the volume encompassing the tumor or treatment area, along with any surrounding tissues that may require treatment.

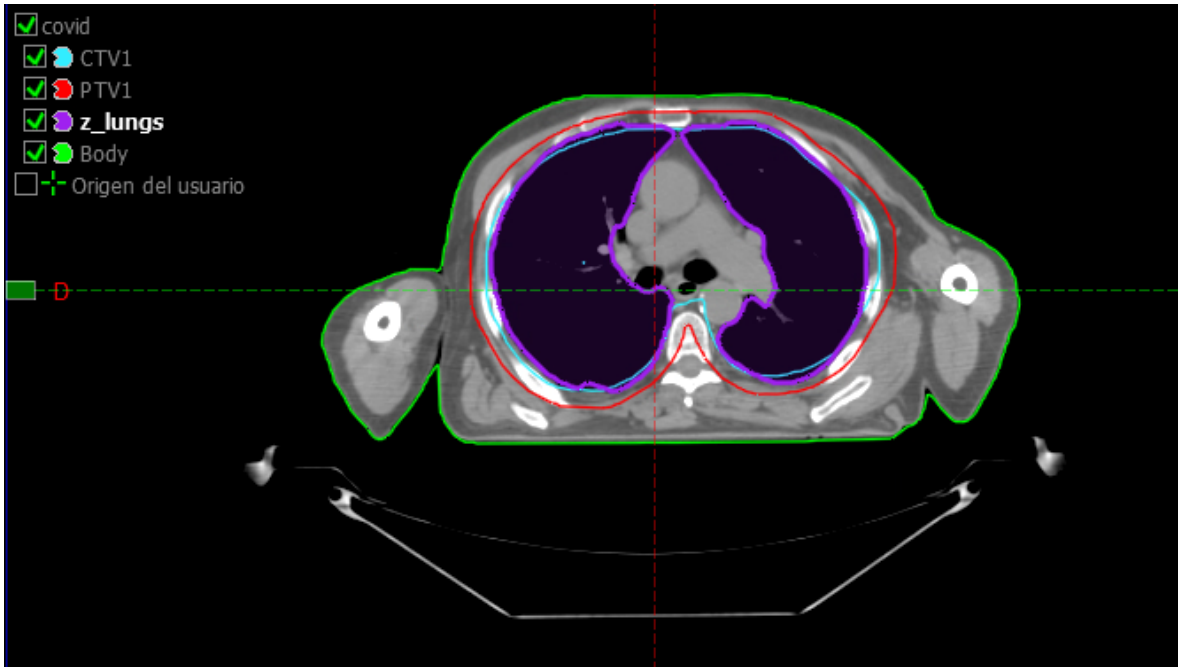


Figure 3: Computed Tomography Structures

The resulting ‘z_lungs’ structure from the CT scan above is representated in 3D in Figure 4.

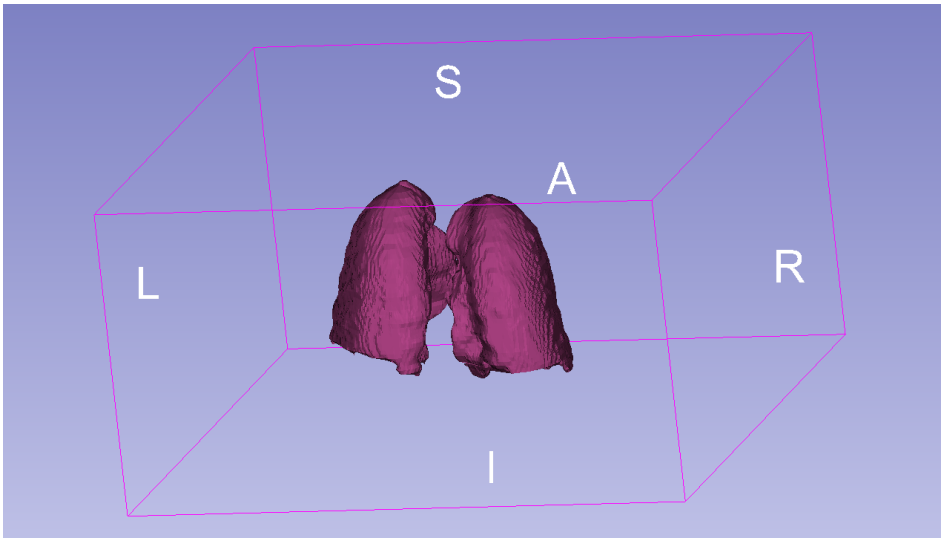


Figure 4: Our target structure: ´z_lungs’

3.2 Data extraction

At this point, we needed to select the appropriate clinical and metabolomic features for this algorithm, as those used in the article from *Piqué et al.* [6] also took into account clinical and metabolomic data after performing the CT scan. In our case, this information is redundant since the prediction focuses on finding differences between patients before treatment, specifically at the time of the CT scan.

Clinical data

Starting with the clinical data, we were left with the following features selected as appropriate for the model, given that they focus on data obtained before the LD-RT treatment and reflect the relevant information for response prediction at the time of the CT scan.

Table 3.1 provides a brief description of the clinical variables selected, detailing the units used in the database, as well as the normal value ranges or their coding into binary and categorical variables.

Variable	Units	Description	Normal Range/ Codification
Age	Years	The patient's age at the time of the CT scan.	Elderly people
Sex	Unitless	The gender of the patient.	0 (Male) or 1 (Female)
Neurological diseases	Unitless	Presence of previous neurological diseases.	1 (Presence) or 0 (Absence)
Cardiovascular diseases	Unitless	Presence of previous cardiovascular diseases.	1 (Presence) or 0 (Absence)
Respiratory diseases	Unitless	Presence of previous respiratory diseases.	1 (Presence) or 0 (Absence)
Other diseases	Unitless	Presence of other diseases not classified in the previous categories.	1 (Presence) or 0 (Absence)
Barthel scale	Scale	Assessment of the patient's ability to perform activities of daily living.	From 0 (total dependence) to 5 (total independence)

Variable	Units	Description	Normal Range/ Codification
GDS	Scale	Measurement to assess the level of cognitive decline in individuals with dementia.	From 0 (no depression) to 6 (severe depression)
Days symptoms	Days	Number of days since the onset of symptoms up to the time of the CT scan.	Variable based on clinical context
Days hospitalization	Days	Number of days the patient remained admitted in the hospital from their admission to their discharge.	Variable based on clinical context
SAFI	Ratio	SpO_2/FiO_2.	Greater than 400
PAFI	Ratio	PaO_2/FiO_2.	Greater than 300
Pulmonary affectation	Scale	Measurement of pulmonary affectation.	From 1 (5-25%) to 4 (more than 75%)
PO_2 venous	mmHg	Partial pressure of oxygen in venous blood.	40-50 mmHg
Oxygen saturation	Percentage	Percentage of oxygen saturation in the blood.	95-100%
Liters oxygen	Liters	Amount of oxygen administered to the patient.	Variable based on patient needs
Cardiac frequency	bpm	Number of heartbeats per minute.	60-100 bpm
Respiratory frequency	rpm	Number of breaths per minute.	12-20 rpm
CURB-65 scale	Score	Score for assessing the severity of pneumonia and the need for hospitalization.	From 0 (low risk) to 5 (high risk)

Table 3.1: Description of Clinical Variables.
GDS: Global Deterioration Scale.

The variable nomenclature has been coded according to the guidelines provided by *Piqué et al.*[6]. Additionally, some descriptions and annotated values have been verified using [11], [12], and [13].

Therewith, a thorough description of these features is provided in Table 3.2, including basic statistics such as the mean values, dispersion or standard deviation, as well as maximum and minimum measures. Additionally, an analysis of the total number of missing values is

performed, allowing us to understand the extent of missing data and decide on management strategies such as imputation or removal. Finally, the types of variables are analyzed to identify any categorical variables within the data that need to be processed for model training.

	Mean	Std	Min	Max	Missing values (%)	Type
Age	84,7	6,58	60	96	0	Integer
Sex	0,5	0,50	0	1	0	Binary
Neurological diseases	0,3	0,45	0	1	0	Binary
Cardiovascular diseases	0,8	0,39	0	1	0	Binary
Respiratory diseases	0,4	0,49	0	1	0	Binary
Other diseases	0,9	0,33	0	1	0	Binary
Barthel scale	3,5	1,13	1	5	0	Categorical
GDS	2,1	1,45	1	6	0	Categorical
Days symptoms	5,1	1,64	2	7	0	Integer
Days hospitalization	18,6	13,90	0	57	0	Integer
SAFI	280,9	93,02	90	471	0	Integer
PAFI	262,3	99,05	59	628	4	Integer
Pulmonary affectation	3,1	0,76	1	4	2	Categorical
PO_2 venous	41,9	12,88	16	82,5	32	Float
Oxygen saturation	93,2	2,98	86	99	2	Integer
Liters oxygen	6,9	4,57	2	15	4	Integer
Cardiac frecuency	101,6	37,23	61	329	2	Integer
Respiratory frecuency	20,5	3,73	15	30	4	Integer
CURB-65 scale	2,9	0,68	2	4	0	Integer

Table 3.2: Clinical data description.
Std: Standard Deviation;
GDS: Global Deterioration Scale.

Considering that the values pertain to severely ill patients, we had to adopt a more flexible approach when identifying outliers, as all variables might be affected by COVID-19. In this context, we did not observe any values that were clearly out of range, and therefore, there was insufficient reason to exclude them based on extreme values alone. Nonetheless, we noted that one variable, *PO_2 venous*, contained a higher percentage of missing values (32%). This variable, along with the three categorical variables (*Barthel scale*, *GDS*, *Pulmonary affectation*), needed to be addressed in the data preparation section to ensure robustness in our analysis. Additionally, we observed a balanced distribution of gender among the observed patients, which suggests that our model is unlikely to introduce gender bias.

Metabolomic data

Continuing with the metabolomic data, the same description of these data is provided in the Table 3.3 below.

Variable	Units	Description	Normal Range
IL6	pg/mL	A pro-inflammatory cytokine involved in immune response and inflammation.	10 pg/mL
Ferritin	ng/mL	A protein that stores iron in the body; its level reflects iron reserves.	- Men: 30-300 ng/mL - Women: 30-200 ng/mL
PCR	mg/L	A marker of acute inflammation in the body.	10 mg/L
LDH	U/L	An enzyme involved in cellular metabolism; can indicate tissue damage or disease.	140-280 U/L
CD4	cells/µL	A type of T cell crucial for immune response; used to monitor immune status.	500-1,500 cells/µL
CD8	cells/µL	T cells that help eliminate infected and tumor cells.	200-1,000 cells/µL
DimerD	ng/mL	A fragment of fibrin breakdown; used to assess the presence of thrombosis.	500 ng/mL
Creatinin	mg/dL	A waste product from muscle metabolism, used to assess kidney function.	- Men: 0.7-1.3 mg/dL - Women: 0.6-1.1 mg/dL
GOT	U/L	A liver enzyme that helps evaluate liver function and damage.	10-40 U/L
GPT	U/L	A liver enzyme that also assesses liver function and damage.	7-56 U/L
Hemoglobin	g/dL	A protein in red blood cells that carries oxygen.	- Men: 14-17 g/dL - Women: 12-16 g/dL
Leukocytes	cells/µL	White blood cells that help fight infections.	4,000-11,000 cells/µL
Lymphocytes	cells/µL	A type of white blood cell essential for the immune response.	1,000-4,000 cells/µL
Platelets	cells/µL	Blood cells involved in clotting.	150,000-450,000 cells/µL

Table 3.3: Description of metabolomic variables. U: Units; GOT: Glutamic Oxaloacetic Transaminase; GPT: Glutamic Pyruvic Transaminase.

Using the information provided in [14], Table 3.3 has been completed.

The subsequent table presents a comprehensive description of the metabolomic data used, including the key statistical measures, percentage of missing values and the type of each variable 3.4.

	Mean	Std	Min	Max	Missing values (%)	Type
IL6	64.4	114.22	0	627	4	Float
Ferritin	1349.6	1836.44	42.7	8836	2	Float
PCR	7.9	6.79	0.3	32.8	0	Float
LDH	315.4	137.30	168	800	2	Float
CD4	240.7	151.91	0	827	8	Float
CD8	162.3	223.15	0	1523	8	Float
DimerD	2196.4	4074.35	220	26680	0	Float
Viral_load_baseline	1.7	1.13	0.4	4.89	52	Float
RNA_total_concentration	383.7	46.30	259.2	487.2	8	Float
Glucose	160.3	81.92	43	390	0	Float
Creatinin	885.2	6250.36	0.15	44198	0	Float
GOT	31.3	22.29	8	137	0	Integer
GPT	24.3	20.83	6	97	0	Integer
Hemoglobin	36178.3	17117.35	10	44450	0	Integer
Leukocytes	9407.4	5396.68	2460	27910	0	Integer
Lymphocytes	1062.2	1617.70	200	9400	0	Integer
Platelets	222634.0	95107.09	10700	583000	0	Integer

Table 3.4: Metabolomic data description.
Std: Standard Deviation; GOT: Glutamic Oxaloacetic Transaminase;
GPT: Glutamic Pyruvic Transaminase.

Table 3.4 provides a detailed statistical overview of the metabolomic data. In this table, outliers were evident in several variables, specifically *Ferritin*, *DimerD*, *Creatinin*, and *Hemoglobin*, as their values significantly exceed the normal range established in Table 3.3. Furthermore, *Viral load baseline* exhibited a substantial proportion of missing values, with 52% of its data being unknown. Additionally, the analysis revealed that all variables were numerical, with no categorical variables requiring preprocessing before the predictive modeling phase.

Radiomic data

Lastly, the extraction of radiomic features from CT images was a more complex process than the previous data extraction and consisted of several steps. Instead of using the 3D Slicer software [15], which was employed in previous works [16] and provided limited feature extraction with much information left undetected, a direct extraction was performed using the Python *PyRadiomics* library [17].
In the following paragraphs, the steps for extracting radiomic features will be explained.

1. **Images format transformation from DICOM to NIfTi**

   To perform radiomic analysis with tools like PyRadiomics, converting images from DICOM (Digital Imaging and Communications in Medicine) to NIfTI (Neuroimaging Informatics Technology Initiative) format was essential. NIfTI, designed for neurological imaging, simplifies handling 3D and 4D images by consolidating data into one or two files, facilitating data management and integration with segmentation masks. In contrast, DICOM's complexity, involving multiple files and extensive metadata, can complicate processing and reduce efficiency. Therefore, converting to NIfTI enhances compatibility with processing tools, improves performance, and supports more efficient analysis, especially in neuroimaging [18].

   In this work, the thoracic image were converted to NIfTI format. Additionally, the contour created in the previous process, named *'z_lungs'*, which exclusively delineated the lung regions, was also converted to NIfTI format to serve as the mask. This resulted in two files for each patient: *'Image.nii'* and *'Mask.nii'*, where *'Mask.nii'* represented the segmentation of the lung regions.

   The code for this transformation is provided in Annexes: Code 1.

2. **Configuration file for feature extraction**

   When extracting features from medical or scientific images, a YAML (YAML Ain't Markup Language) configuration file is crucial. YAML is a human-readable and simple data format designed to be easy for humans to understand and write while being easily interpretable by machines.

   In this particular project, the YAML file outlined the application of various image filters, including *original images*, *wavelet transforms*, as well as *square*, *square root*, *logarithm*, *exponential*, *gradient*, and *local binary patterns (LBP 2D and 3D)*. The configuration also detailed the types of features to be extracted, covering *shape-based*

metrics, *first-order statistics*, and texture-related features such as the *gray level co-occurrence matrix*, *gray level run length matrix*, *gray level size zone matrix*, *gray level dependence matrix*, and *neighboring gray tone difference matrix*. This setup ensured a comprehensive analysis by applying multiple image transformations and extracting a broad range of features to fully capture different aspects of the image data.

The YAML configuration file is shown in Annexes: Code 2.

3. **Feature extraction using PyRadiomics**

	Number of features
Original	107
Wavelet	744
Square	93
Square Root	93
Logarithm	93
Exponential	93
Gradient	93
LBP_2D	93
LBP_3D	279
Total radiomic features:	**1688**

Table 3.5: Radiomic Feature Filters and their corresponding number of extracted features. LBP: Local Binary Patterns.

Using the *PyRadiomics* library, 1688 features were extracted from the CT images to quantify characteristics relevant to predicting treatment responses.

Shape-based features were applied exclusively to the original CT images. These features describe geometric properties such as volume, surface area, and sphericity. Since they are focused solely on the size and form of the segmented region, shape-based features are limited in number and do not capture variations in pixel intensity or texture.

In contrast, texture features extracted using various filters were more numerous and provided a more comprehensive analysis. These features capture patterns and distributions of gray levels, analyzing the relationships and variations in pixel intensities within the segmented region.

Among these, the wavelet filter significantly increased the number of extracted features by decomposing the image into eight sub-bands, each representing different levels of detail and orientation—specifically, low-low, low-high, high-low, and high-high frequencies in both horizontal and vertical directions. This decomposition allowed for texture features to be extracted from each sub-band, effectively multiplying the number of features by eight.

Similarly, the LBP 3D filter contributed to the large number of features extracted, yielding a total of 279 features. This is attributed to its extensive exploration of local texture patterns across various spatial scales in three dimensions.

The code implementated to extract radiomic features is appended in Annexes, in Code 3.

3.3 Data preparation

As mentioned by *Shrivastava et al.*, 'If the data is inadequate, or contains extraneous and irrelevant information, Machine Learning (ML) algorithms may produce less accurate and less understandable results. Thus, data preparaton is an important step in the ML process.' [19]

1. **Outliers:** The first step was to analyse our clinical and metabolomic data for any obvious outliers. Outliers may arise from human error during manual data entry or due to instrumental errors. In both cases, the most appropriate approach is to manually remove these data points when it is certain they are incorrect and treat them as null values [20]. This ensures that they do not adversely affect the accuracy of our model. For radiomic features, identifying outliers was not applicable, as there are no established criteria for defining outliers in this type of data. Therefore, our outlier analysis was focused solely on clinical and metabolomic data.

2. **Columns with fixed values:** Variables that remain constant across patients do not contribute to differentiating them. Therefore, removing these columns was an important step to take before proceeding with preprocessing. This ensures that these columns are not considered from the outset and do not affect feature selection.

3. **Missing values:** When working with data, it is crucial to properly handle missing values, as they can negatively impact the performance of a PM. A straightforward option is to replace them with the mean (for numerical variables) or the mode (for categorical variables) of the column. However, this method does not account for relationships between variables and can introduce bias if many values are missing. A more advanced approach is the *Iterative Imputer* function from the Scikit-learn library [21]. This technique, as its name suggests, iteratively predicts missing values by treating the column with missing values as the target variable in each iteration while the other columns, whether they have missing values or not, serve as predictors to train a regression model. Finally, through successive iterations, this process refines the imputations based on the relationships learned from the other columns, leading to more accurate and realistic imputations.

   Additionally, we have used a threshold of 20% of missing values to consider that a column should be removed, as they can severely impact data quality and introduce bias, compromising the effectiveness of the model. This is exemplified by the *Viral load baseline* variable, as shown in Figure 5 and quantitatively observed in Table 3.4, where a significant proportion of the data was missing, indicated by white areas.

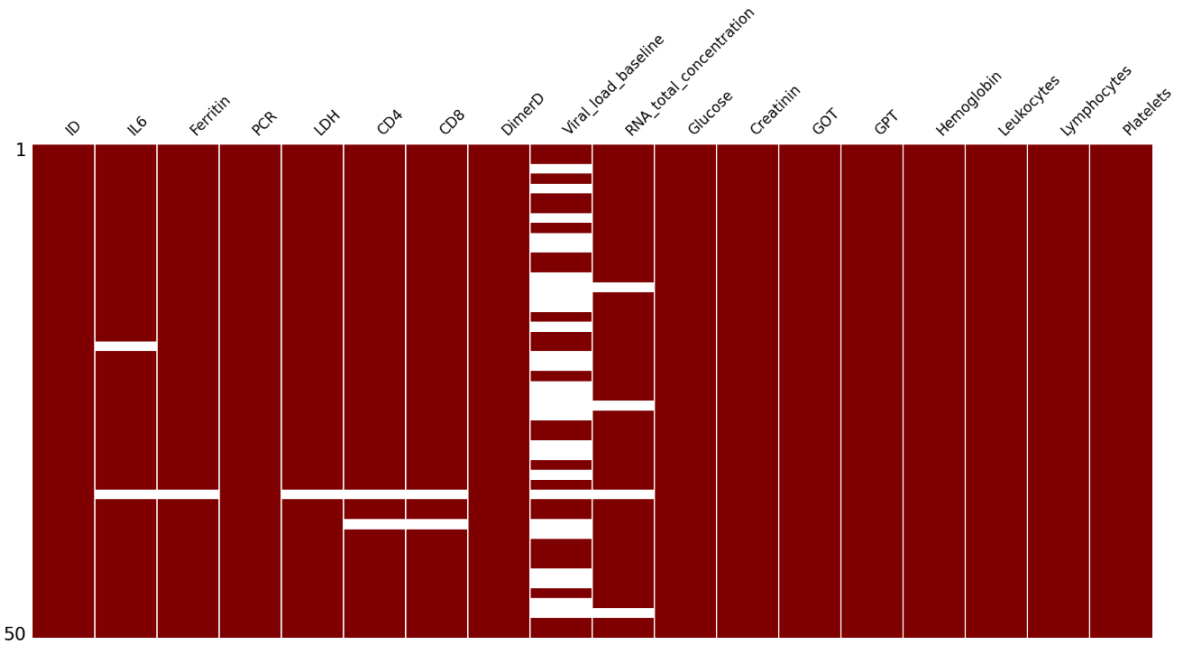


Figure 5: Missing values representation in Metabolomic data

Moreover, it was important to reassess the percentage of missing values after removing outliers, as this can potentially increase the proportion of missing data beyond the 20% threshold.

4. **Categorical features:** Transforming categorical variables is a crucial step in preparing data for ML models, as many of these models require numerical inputs and cannot directly process categorical data. One commonly used technique for this transformation is *one-hot encoding* from the Scikit-learn library [22]. This technique converts a categorical variable into multiple binary columns, with each column corresponding to a possible category. In each record, one of these columns will take the value 1 to indicate the presence of that specific category, while the other columns are set to 0. By applying this transformation, categorical variables (in Table 6a) were effectively converted into numerical data (in Table 6b), which enhances the models' ability to interpret and process the different categories present in the dataset.

	GDS
Patient 1	2
Patient 2	5

(a) Categorical feature

	GDS 1	GDS 2	GDS 3	GDS 4	GDS 5	GDS 6
Patient 1	0	1	0	0	0	0
Patient 2	0	0	0	0	1	0

(b) Categorical feature transformed binary columns

Figure 6: Two tables showing the categorical features transformation steps.

5. **Normalization:** Data normalization is crucial to ensure that all features contribute equally to the ML model, preventing features with larger scales from dominating the learning process. In this case, the *MinMaxScaler* from the Scikit-learn library [23] method was used, which transforms each feature to a defined range, typically between 0 and 1, according to the formula shown in Equation 3.1. This normalization ensures that all features are on a comparable scale, preventing biases and facilitating a balanced contribution of each feature during model training.

$$X_{norm} = \frac{X - X_{min}}{X_{max} - X_{min}} \tag{3.1}$$

Data preparation codes are added in Annexes: Codes 4 (for clinical data) and 5 (for metabolomic data). Radiomic data only needed to be normalized as values where free from outliers, missing values and categorical features.

3.4 Data augmentation

As shown in Figure 7, in our dataset, we had 34 patients who exhibited a positive response to the treatment, meaning they recovered, demonstrating the treatment's efficacy. However, the remaining 16 patients, out of a total of 50, showed no signs of improvement within a three-day period and subsequently passed away during this time frame. As we can see, our dataset was significantly imbalanced between the classes. If we had trained the model with this data, we would have obtained a model biased towards the majority class, as demonstrated in previous research [24], which highlights the importance of balancing imbalanced data to improve classification accuracy in ML.

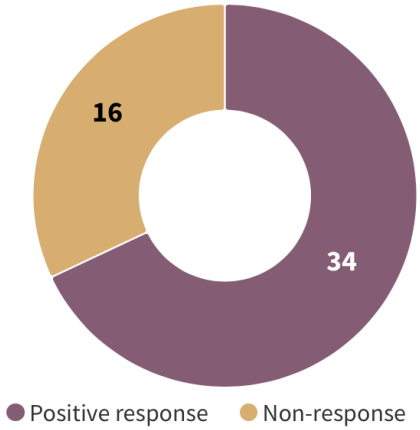


Figure 7: Distribution of treatment response between classes.

One common approach to addressing imbalanced data classes is to reduce the size of the majority class to match that of the minority class. Nevertheless, given the limited number of patients in our dataset, this method was not ideal, as it would result in a significant loss of valuable information.

An alternative and more suitable strategy involved augmenting the minority class by generating synthetic data based on the existing information. This approach allowed us to maintain the integrity of the dataset while mitigating the effects of class imbalance, leading to a more balanced and representative dataset for model training.

1. **Clinical and metabolomic data augmentation:**

   To balance clinical and metabolomic data, which are numerical in nature, a well-known oversampling method called *SMOTE (Synthetic Minority Oversampling TEchnique)* was employed as it has been demonstrated that in low-dimensional data, the use of this technique is beneficial [25].

SMOTE operates by randomly selecting nearby neighbours for each sample in the minority class, using a distance metric such as Euclidean distance. The number of selected neighbors is an adjustable parameter that influences the diversity of the generated synthetic samples. To create a new sample, a nearby neighbor is chosen, and the difference between the features of the original sample and those of the neighbor is calculated. This difference is then multiplied by a random number between 0 and 1, and the resulting value is added to the original features, thereby generating a new synthetic sample that lies somewhere between the original sample and its neighbour. This process is repeated until the minority class reaches the same size as the majority class [26].

When augmenting data, we needed to observe that the synthetic data followed the same Gaussian distribution as the original data. This is important because it ensures that the synthetic data accurately represents the same characteristics and maintains the same variability as the original dataset.
Figure 8 illustrates the Gaussian distribution observed for the metabolomic variable *Creatinine*.

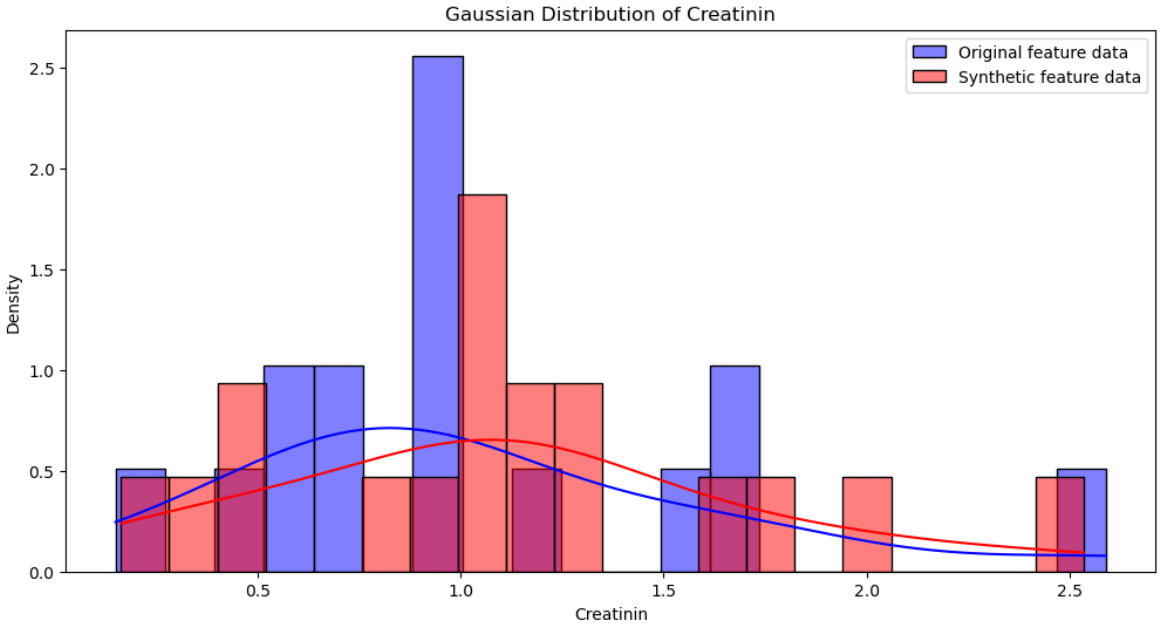
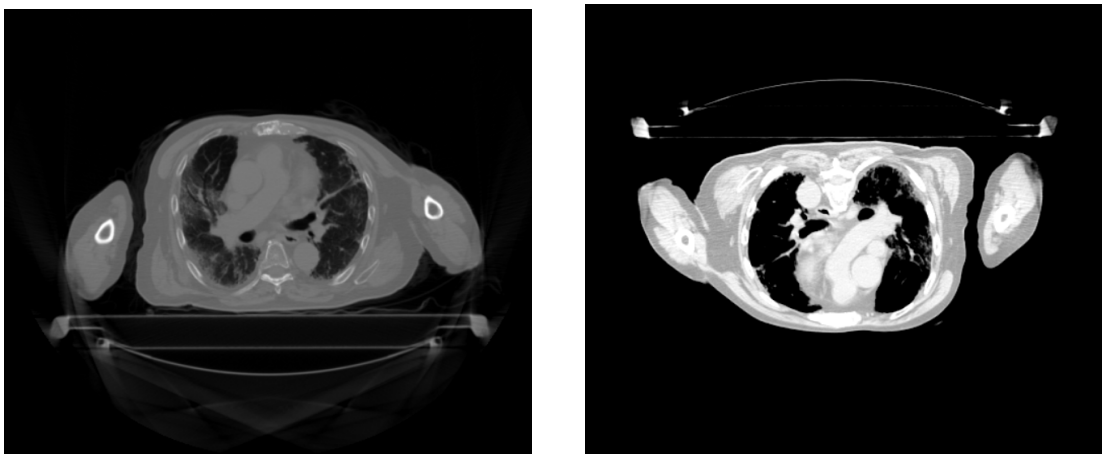


Figure 8: Gaussian distribution with Creatinin feature.

The codes used to augment clinical and metabolimic data are included in Annexes: Code 6.

2. **Image augmentation:**

   In the context of image data augmentation, we needed to generate synthetic images rather than merely creating synthetic features from the original features extracted, as this approach generally yields better model performance. To achieve this, we utilized the *PyTorch* library to apply various transformations to the original images (see Figure 9a), thereby creating new, synthetic variations (see Figure 9b).



(a) Original CT scan. (b) Synthetic CT scan.

Figure 9: Original and synthetic Computed Tomography scans comparison.

The transformation process involved several random alterations to the images, including the addition of *noise*, changes in *brightness*, and *flipping*.

Additionally, it was imperative to adjust the mask associated with each image in accordance with the flipping transformations applied to the synthetic images. This adjustment ensures that the features extracted from the transformed images matches information from the lungs. In Figure 10 an example of a mask image produced through these transformations is provided below (see Figure 10a), alongside its original counterpart (see Figure 10a) for comparison.

(a) Original Mask. (b) Synthetic Mask.

Figure 10: Original and synthetic mask comparison.

The codes performed to obtain synthetic images are appended in Annexes in Code 7.

3.5 Data preprocessing

Selecting only the most relevant features decreases the model's complexity, which in turn speeds up training and makes the model more interpretable. Additonally, it helps eliminate noise from irrelevant features, confirming that the model focuses on the most informative data.
Two different methods for selecting the necessary features were applied and compared:

- **Manual selection:** The feature extraction process was conducted manually, involving the deliberate selection of individual methods to filter and refine the dataset. This approach required careful consideration and choice of each technique to guarantee the most relevant features were identified and retained for further analysis.

    1. **Correlation evaluation:** Manually selecting features involved evaluating the correlation between features and the target variable to identify and remove those with weaker relationships. This approach helps to reduce the risk of biased predictions that may arise due to feature correlation. In this process, we used Pearson's correlation coefficient r given by the following Equation 3.2:

$$r = \frac{\sum_{i=1}^{n} \left(\mathrm{X}_i - \bar{\mathrm{X}}\right)\left(\mathrm{Y}_i - \bar{\mathrm{Y}}\right)}{\sqrt{\left(\sum_{i=1}^{n} \left(\mathrm{X}_i - \bar{\mathrm{X}}\right)^2\right)\left(\sum_{i=1}^{n} \left(\mathrm{Y}_i - \bar{\mathrm{Y}}\right)^2\right)}} \tag{3.2}$$

    Where:

    – X and Y are two features in the dataset.
    – $\bar{\mathrm{X}}$ and $\bar{\mathrm{Y}}$ are the means of X and Y.

    2. **Variance Inflator Factor (VIF) calculation:** Additionally, calculating the VIF (see Equation 3.3) was essential for statistically measuring the extent of multi-collinearity among predictor variables. Features with a VIF exceeding a threshold of 5 indicate excessive correlation with other features and should be removed to improve model interpretability [27].

$$\mathrm{VIF_i} = \frac{1}{1 - \mathrm{R_i^2}} \tag{3.3}$$

    Where:

    – $\mathrm{R_i^2}$ is the value from regressing the i-th variable on all the other variables.

    This approach ensures that the remaining features contribute valuable and distinct information to the model.

    The code for manual feature selection is included in the Annexes: Code 8.

- **Automatic selection:** In parallel with the manual feature selection process, an automatic feature selection method was implemented using the mRMR (minimum Redundancy Maximum Relevance) algorithm, specifically utilizing the MIQ (Mutual Information Quotient) variant. The mRMR method filters features considering both the relevance for predicting the outcome variable and the redundancy within the selected features, performing less memory consumption and time required to achieve computation efficiency [28].

  The code for automatic feature selection is shown in Annexes: Code 9.

3.6 Predictive models

Once the data was cleaned, preprocessed, and features were selected, the next step was to implement various predictive models (PMs) based on the input data and the three classifiers used. The steps to be followed in this phase are illustrated in the Figure 11:

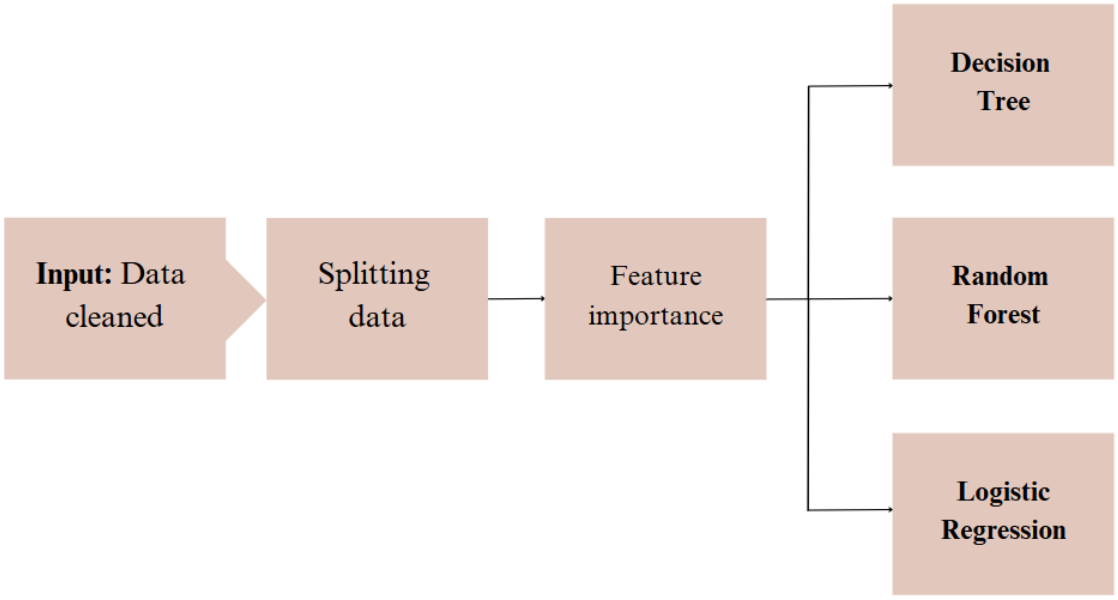


Figure 11: Predictive Model workflow representation.

Splitting data

Before training the models, the dataset was split into two subsets: 90% for training and 10% for testing. Due to the limited amount of data, *Cross-Validation (CV)* was implemented within the 90% training subset to maximize data utilization and provide a more reliable estimate of the model's generalization capability.

CV involves dividing the training set into multiple smaller subsets called *folds*. This process was repeated several times (in this case, 5 folds), where the model was trained on a subset of the folds and validated on the remaining fold (which corresponded to 20% of the training data). Each fold was used as the validation set once throughout the process.

Additionally, the separate testing set was used to evaluate the performance of the best PM on unseen data.

A visual representation of this process is provided in Figure 12:

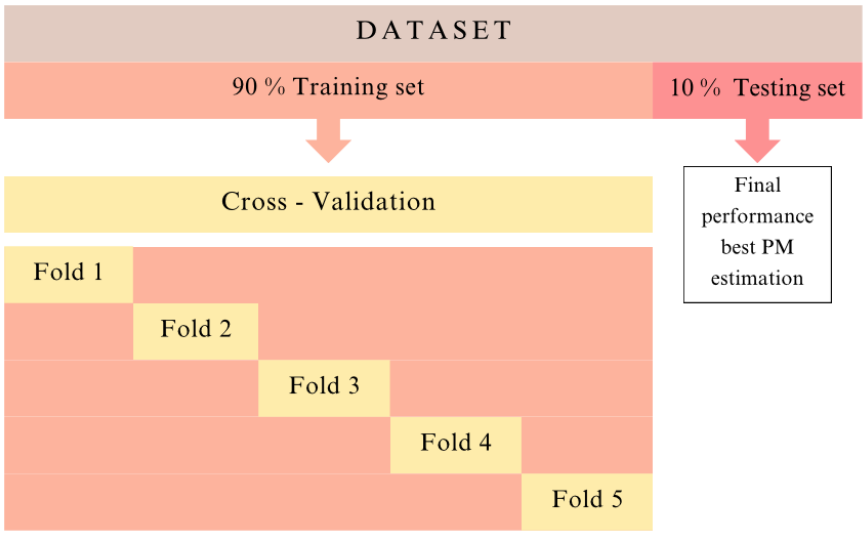


Figure 12: Splitting data process representation.
PM: Predictive Model.

Feature Importance

To enhance the accuracy and efficiency of the models, feature importance weighting was conducted using a forest of trees (Random Forest). This model is particularly effective for classification tasks and offers valuable insights into the significance of each feature in predicting the target variable. By evaluating the weights assigned to each feature [29], we could identify the most impactful ones, 9 in this case, allowing us to reduce the dimensionality of the dataset and potentially improve model performance.

Model Implementation

In this analysis, three widely recognized classification models were employed due to their effectiveness in supervised learning tasks: *Random Forest (RF)*, *Logistic Regression (LR)*, and *Decision Trees (DT)*. These models are well-known for their robust performance in classification tasks, making them suitable candidates for our comparative study. The models were evaluated and compared using various performance metrics to determine which one provided the best results for our specific application.

- **DT** is a classification model that works by splitting the data into subsets based on feature values that best separate the classes. At each node, the algorithm chooses a feature and a threshold that maximizes class separation (using criteria like Gini impurity or entropy). This process continues until the data is sufficiently pure (mostly one class) at the leaf nodes, where the final class prediction is made. DTs are easy to interpret but can overfit the data if the tree becomes too complex [30].

- **RF** is an ensemble classification method that builds multiple DTs using random subsets of the data and features. Each tree in the forest makes its own classification, and the final output is determined by majority voting among all the trees. This approach reduces the variance and overfitting associated with individual decision trees, leading to more accurate and stable predictions, though it sacrifices some interpretability [31].

- **LR** is a linear classification model that predicts the probability that a given input belongs to a specific class. It does this by fitting a linear equation to the input features and applying a logistic (sigmoid) function to convert the output into a probability between 0 and 1. Based on this probability, the model assigns the input to one of the two classes. LR is simple and effective for linearly separable data but may struggle with more complex, non-linear relationships [32].

Finally, the model that demonstrated the best performance was selected to make the final predictions on the test set using OPTUNA as an hyperparameter tuning optimization [33], aiming to maximize specificity and generalization capability for future applications.

All the PM codes are provided in Annexes: Codes 10 (RF PM code), 12 (DT PM code), 11 (LR PM code) and 13 (Best PM testing).

3.7 Model validation

The models' performance was evaluated using the following metrics:

1. **Sensitivity:** Also referred to as, recall, this metric indicates the proportion of True Positives (TP) that the model has correctly identified. It is calculated as:

$$\textbf{Sensitivity} = \frac{\text{TP}}{\text{TP + FN}} \tag{3.4}$$

2. **Specificity:** Specificity measures the proportion of True Negatives (TN) that the model has correctly identified. The formula is:

$$\textbf{Specificity} = \frac{\text{TN}}{\text{TN + FP}} \tag{3.5}$$

3. **Confusion Matrix (CM):** The confusion matrix is a table that visually summarizes the performance of the classification model by showing TP, False Positives (FP), TN, and False Negatives (FN). An example of a CM is in the Table 3.6:

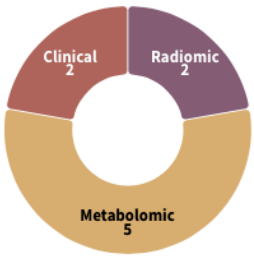
	Positive Prediction	**Negative Prediction**
Actual Positive	TP	FN
Actual Negative	FP	TN

Table 3.6: Confusion Matrix Representation

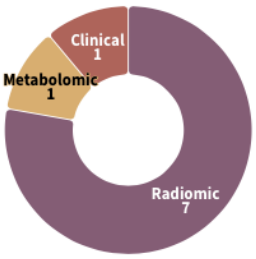
4. Results

In this section, the efficacy on predicting the response of COVID-19 patients treated with LD-RT was evaluated. For this, data augmentation techniques, various feature selection methods and three different classification algorithms were implemented.

Original Data Results



(a) Features extracted manually



(b) Features extracted automatically

Figure 11: Feature Importance Types in Original Data.

For the original data, which was imbalanced and lacked data augmentation techniques, the number of manually selected features from each group based on their importance is represented in Figure 11a. Additionally, for the automatic feature selection, the selected data type representation is visualized in Figure 11b. The features, ordered by their importance in predicting the target variable, are shown in Annexes in supplementary Figures 15 and 16, respectively.

The results obtained using only the original imbalanced data are presented as follows in Table 4.1, distinguishing between manual and automatic feature selection:

	Features extracted manually		**Features extracted automatically**	
	Sensitivity	**Specificity**	**Sensitivity**	**Specificity**
Decision Tree	0.65	0.43	0.71	0.50
Random Forest	0.87	0.57	0.90	0.43
Logistic Regression	0.87	0.57	1.00	0.00

Table 4.1: Imbalanced data results

We observed that sensitivity, which is the rate of true positives, was generally high, with the RF classifier using automatically extracted features performing the best in this metric. However, specificity remained very low across all results obtained.

Augmented Data Results

Next, we examined the results obtained with the augmented data, using a similar approach as before.

(a) Features extracted manually

(b) Features extracted automatically

Figure 12: Feature Importance Types in Augmented Data.

The data type of the important features identified were, in the case of manually selected features, the represented in Figure 12a and for the automatically selected features (see Figure 12b). The corresponding features importance, ordered by their weight in predicting the target variable, are shown in Annexes in supplementary Figures 17 and 18, respectively.

All radiomic features selected are explained in Annexes: Table 8.

The results obtained with this data input are as follows:

	Features extracted manually		**Features extracted automatically**	
	Sensitivity	**Specificity**	**Sensitivity**	**Specificity**
Decision Tree	0.68	0.77	0.74	0.77
Random Forest	0.84	0.77	0.81	0.83
Logistic Regression	0.68	0.70	0.67	0.60

Table 4.2: Augmented data results

With data augmentation to address the class imbalance issue, better results were achieved compared to not addressing the class disparity. The best performance, as before, was obtained with the RF classifier using automatically selected features, achieving a sensitivity of 0.81 and a specificity of 0.83.

Testing Set Results

As mentioned earlier, the best-performing model underwent hyperparameter tuning to maximize its effectiveness, utilizing 10% of the data as a testing set to evaluate performance on unseen data. This model achieved sensitivity and specificity values of 1.00 and 0.50, respectively.

All CM from the three classifiers are represented in Annexes in Figures 27 to 31.

5. Discussion

The current study introduces a novel approach by integrating clinical, metabolomic, and radiomic data to predict responses to LD-RT treatment in patients with COVID-19 pneumonia. Although predictive modeling using such diverse data sources is not entirely new, our study provides unique insights by focusing on the specific application of these techniques to predict LD-RT response of COVID-19 pneumonia patients.

A critical element of this study was the use of the PyRadiomics library for extracting radiomic features from CT images. While 3D Slicer also implements PyRadiomics, it lacks the extensive image filtering capabilities available through the library itself. The direct use of PyRadiomics enabled us to extract a greater number of features through multiple image transformations, revealing patterns that would otherwise remain undetectable. This advanced feature extraction process significantly contributed to enabling automation and scalabilty as well as it allowed to identifying impactful features and enhancing model performance.

Our analysis revealed that radiomic features were predominantly selected across various models, largely due to their higher quantity compared to clinical and metabolomic features. Conversely, clinical features were less emphasized in most PM, suggesting that, despite their potential relevance, they did not contribute as significantly to the model's performance in this context. This suggests that although clinical variables are currently the primary factors used to plan RT treatment and predict patient response, other data, such as metabolomic and radiomic features, should also be considered.

Our findings reveal a notable trade-off between sensitivity and specificity across different models. Specifically, the RF classifier, when applied to the original imbalanced data with automatically extracted features, achieved a high sensitivity of 0.97 but exhibited low specificity at 0.50. Given its low specificity, this model was deemed unsuitable as we aim to accurately identify TP and avoid administering treatment to patients who are unlikely to benefit from it. To address the class imbalance, a data augmentation methodology was applied, which improved the results: the RF classifier achieved a sensitivity of 0.81 and a specificity of 0.83. This enhancement underscores the effectiveness of data augmentation in improving model performance and ensuring better identification of negative cases.
In both cases, the best-performing model utilized automatically selected features using the mRMR method, which also demonstrated a lower emphasis on clinical features than the manual method. Moreover, this approach not only yielded general superior results in terms of sensitivity and specificity but also significantly reduced the time required for feature filtering, compared to manual feature selection.

When evaluated on the test set, the RF classifier achieved a sensitivity of 1.00 but a specificity of only 0.50, highlighting the impact of the limited amount of training data on the model's ability to generalize to unseen data. The small dataset size was a significant constraint in our study, which is a common challenge in ML applications, particularly in specialized healthcare studies.

One notable limitation of the current study is the restricted number of data samples, which impacts the model's performance and generalizability. Despite this limitation, we have demonstrated that predictive models can be useful and have significant potential. The high sensitivity observed indicates that the model has the capability to correctly identify responses to LD-RT treatment in a substantial proportion of cases. However, the low specificity suggests that improvements are needed to reduce FP.

At this juncture, future work should undoubtedly focus on expanding the dataset by collaborating with additional hospital centers willing to administer LD-RT to their most severely affected COVID-19 patients. This expansion is essential to enhance the results and develop a reliable and unbiased predictive model. Once this significant step is achieved, the next phase involves implementing the model in a practical clinical setting. Specifically, this entails integrating the model as a clinical support system for radiotherapy departments through a web-based implementation hosted on the hospital's server. This will provide doctors with a valuable tool for improving patient care and treatment decision-making.

6. Conclusions

This Final Degree Project emphasizes the intricate and sensitive nature of PMs within health data analysis. It has become evident that even minor oversights or inconsistencies in the dataset can significantly alter the results of these models. For PM to achieve their full potential, they require input data that is not only reliable and clean but also well-balanced and sufficiently voluminous. The quality and quantity of the data directly impact the efficacy and accuracy of the model's predictions.

The primary objective of this project was to develop a PM capable of determining the response to LD-RT treatment in patients with COVID-19 pneumonia. Although the results were constrained in the test set due to the low dimensionality of the data, reliable outcomes were achieved in the training set. Nonetheless, they did not achieve the level of specificity in the test set, necessary for implementation in clinical practice.

Despite these challenges, the insights provided by PMs can be invaluable, depending on the context and situation. The ability to foresee potential outcomes before they occur represents a transformative advancement in medicine. However, it is crucial to acknowledge that such models are not infallible and that some degree of error is inevitable. For this reason, they should never replace clinical decisions but rather support them.

To fully realize the potential of predictive modeling in healthcare, there is an urgent need to advance the integration of technological and biomedical engineering expertise into healthcare environments. Professionals from diverse fields, including telecommunications engineers, computer scientists, and physicists, should collaborate closely with healthcare practitioners. This interdisciplinary approach is essential for driving innovations and breakthroughs in medical data analysis and for paving the way toward a more effective and efficient future in healthcare. Moreover, leveraging these predictive models can lead to significant cost savings by optimizing resource allocation, reducing unnecessary treatments, and improving patient outcomes.

In conclusion, while the road to integrating predictive modeling into healthcare is fraught with challenges, it holds the potential of revolutionizing healthcare. By ensuring high-quality data, fostering interdisciplinary collaboration, investing in technological advancements, AI can become an essential component of future medical practice.

References

[1] N. Zhu, D. Zhang, W. Wang, *et al.*, "A novel coronavirus from patients with pneumonia in china, 2019," *New England journal of medicine*, vol. 382, no. 8, pp. 727–733, 2020.

[2] W. H. Organization, *Total cumulative number of covid-19 cases reported to who*, https://data.who.int/dashboards/covid19/cases, Accessed on the 10th of April, 2020, 2024.

[3] M. Arenas, B. Piqué, L. Torres-Royo, *et al.*, "Treatment of covid-19 pneumonia with low-dose radiotherapy plus standard of care versus standard of care alone in frail patients: The seor-gicor ipacovid comparative cohort trial," *Strahlentherapie und Onkologie*, vol. 199, no. 9, pp. 847–856, 2023.

[4] M. Arenas, S. Sabater, V. Hernández, *et al.*, "Anti-inflammatory effects of low-dose radiotherapy," *Strahlentherapie und Onkologie*, pp. 1–7, 2012.

[5] A. Jain, H. Patel, L. Nagalapatti, *et al.*, "Overview and importance of data quality for machine learning tasks," in *Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining*, 2020, pp. 3561–3562.

[6] B. Piqué, K. Peña, F. Riu, *et al.*, "Sars-cov-2 serum viral load and prognostic markers proposal for covid-19 pneumonia in low-dose radiation therapy treated patients," *Journal of clinical medicine*, vol. 12, no. 3, p. 798, 2023.

[7] K. B. Méndez-Rodríguez, M. J. Santoyo-Treviño, K. Saldaña-Villanueva, M. Rodríguez-Aguilar, R. Flores-Ramírez, and F. J. Pérez-Vázquez, "Metabolómica como nueva herramienta para el diagnóstico oportuno en enfermedades no transmisibles," *Revista de Salud Ambiental*, vol. 19, no. 2, pp. 109–115, 2019.

[8] M. E. Mayerhoefer, A. Materka, G. Langs, *et al.*, "Introduction to radiomics," *Journal of Nuclear Medicine*, vol. 61, no. 4, pp. 488–495, 2020.

[9] K. Lekadir, A. Feragen, A. J. Fofanah, *et al.*, *Future-ai: International consensus guideline for trustworthy and deployable artificial intelligence in healthcare*, 2024. arXiv: 2309.12325 [cs.CY]. [Online]. Available: https://arxiv.org/abs/2309.12325.

[10] Varian Medical Systems, *Eclipse treatment planning system*, [Accessed: 4-Aug-2024], 2024. [Online]. Available: https://www.varian.com/products/radiotherapy/treatment-planning/eclipse.

[11] S. Venegas, M. Cortés, L. Flores, *et al.*, "Correlación de spo2/fio2 versus pao2/fio2 para monitoreo de la oxigenación en pacientes con trauma de tórax," *Med Crit*, vol. 32, no. 4, pp. 201–207, 2018. DOI: 10.35366/TI184E.

[12] Lab Tests Guide, *Po2 normal values*, [Accessed: 4-Aug-2024], 2024. [Online]. Available: https://www.labtestsguide.com/po2#po2-normal-values.

[13] M. Moser, M. Lehofer, G. Hildebrandt, M. Voica, S. Egner, and T. Kenner, "Phase- and frequency coordination of cardiac and respiratory function," *Biological Rhythm Research*, vol. 26, no. 1, pp. 100–111, 1995.

[14] MSD Manual, *Pruebas de sangre: Valores normales*, [Accessed: 4-Aug-2024], 2024. [Online]. Available: https://www.msdmanuals.com/es/professional/recursos/valores-normales-de-laboratorio/pruebas-de-sangre-valores-normales.

[15] 3D Slicer, *Documentation: Radiomics extension*, [Accessed: 4-Aug-2024], 2024. [Online]. Available: https://www.slicer.org/wiki/Documentation/4.8/Extensions/Radiomics.

[16] J. P. F. Oliver, *Development and evaluation of machine learning models for COVID-19 related research*, Final Degree Project, Universitat Rovira i Virgili, 2023. [Online]. Available: https://hdl.handle.net/20.500.11797/TFG7026.

[17] J. J. Van Griethuysen, A. Fedorov, C. Parmar, *et al.*, "Computational radiomics system to decode the radiographic phenotype," *Cancer research*, vol. 77, no. 21, e104–e107, 2017.

[18] X. Li, P. S. Morgan, J. Ashburner, J. Smith, and C. Rorden, "The first step for neuroimaging data analysis: Dicom to nifti conversion," *Journal of neuroscience methods*, vol. 264, pp. 47–56, 2016.

[19] H. Shrivastava and S. Sridharan, "Conception of data preprocessing and partitioning procedure for machine learning algorithm," *International Journal of Recent Advances in Engineering & Technology (IJRAET)*, vol. 1, no. 3, pp. 2347–2812, 2013.

[20] J.-F. Beaumont and L.-P. Rivest, "Dealing with outliers in survey data," in *Handbook of statistics*, vol. 29, Elsevier, 2009, pp. 247–279.

[21] S.-l. developers, *Sklearn.impute.iterativeimputer*, Accessed: 2024-08-20, 2024. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.impute.IterativeImputer.html.

[22] Scikit-learn Developers, *Sklearn.preprocessing.onehotencoder*, Accessed: 2024-08-20, 2024. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.OneHotEncoder.html.

[23] Scikit-learn, *Sklearn.preprocessing.minmaxscaler*, Accessed: 2024-08-03, 2024. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html.

[24] V. S. Spelmen and R. Porkodi, "A review on handling imbalanced data," in *2018 international conference on current trends towards converging technologies (ICCTCT)*, IEEE, 2018, pp. 1–11.

[25] R. Blagus and L. Lusa, "Smote for high-dimensional class-imbalanced data," *BMC bioinformatics*, vol. 14, pp. 1–16, 2013.

[26] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "Smote: Synthetic minority over-sampling technique," *Journal of artificial intelligence research*, vol. 16, pp. 321–357, 2002.

[27] J. I. Daoud, "Multicollinearity and regression analysis," in *Journal of Physics: Conference Series*, IOP Publishing, vol. 949, 2017, p. 012 009.

[28] Z. Zhao, R. Anand, and M. Wang, "Maximum relevance and minimum redundancy feature selection methods for a marketing machine learning platform," in *2019 IEEE international conference on data science and advanced analytics (DSAA)*, IEEE, 2019, pp. 442–452.

[29] Scikit-Learn, *Example of feature importance with random forest*, [Accessed: 4-Aug-2024], 2024. [Online]. Available: https://scikit-learn.org/stable/auto_examples/ensemble/plot_forest_importances.html.

[30] Scikit-learn, *Decisiontreeclassifier*, Accessed: 2024-08-25, 2024. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html.

[31] Scikit-learn, *Randomforestclassifier*, Accessed: 2024-08-25, 2024. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html.

[32] Scikit-learn, *Logisticregression*, Accessed: 2024-08-25, 2024. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html.

[33] O. Authors, *Optuna: A hyperparameter optimization framework*, Accessed: 2024-08-03, 2024. [Online]. Available: https://optuna.readthedocs.io/en/v4.0.0/.

[34] J. Osuna, *Radiomics-ai*, Accessed: 2024-08-04, 2023. [Online]. Available: https://github.com/Judith0suna/Radiomics-AI.

Annexes

I. Feature Importance Ranking

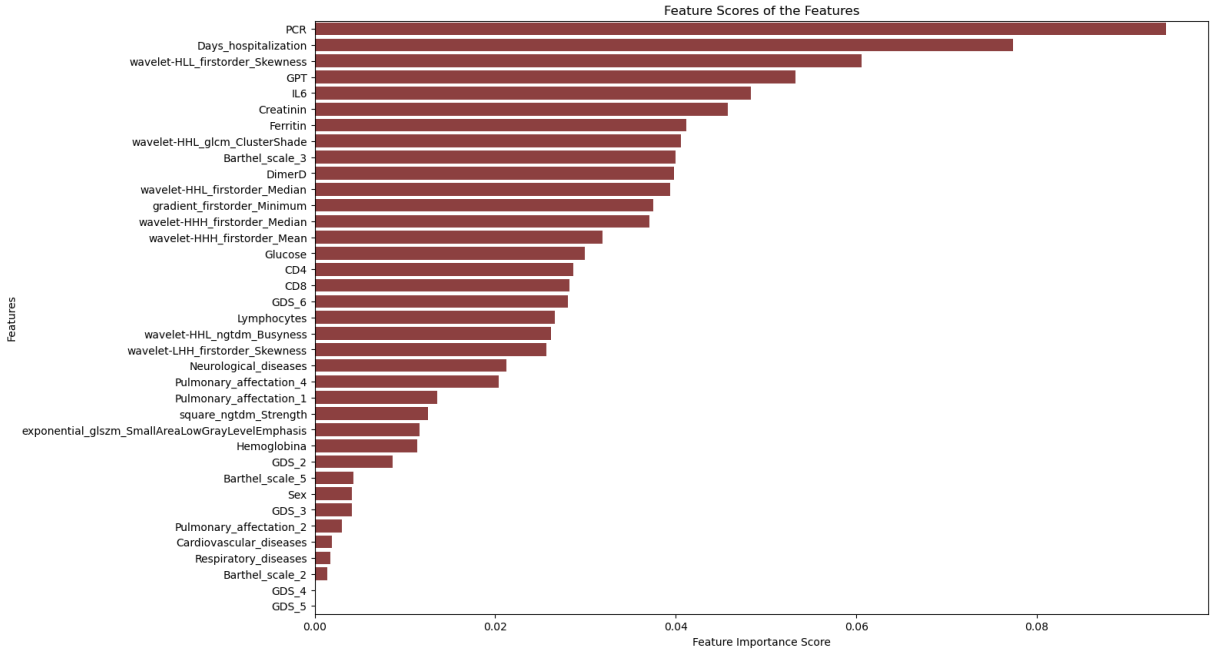


Figure 15: Feature importance with original and manually selected data

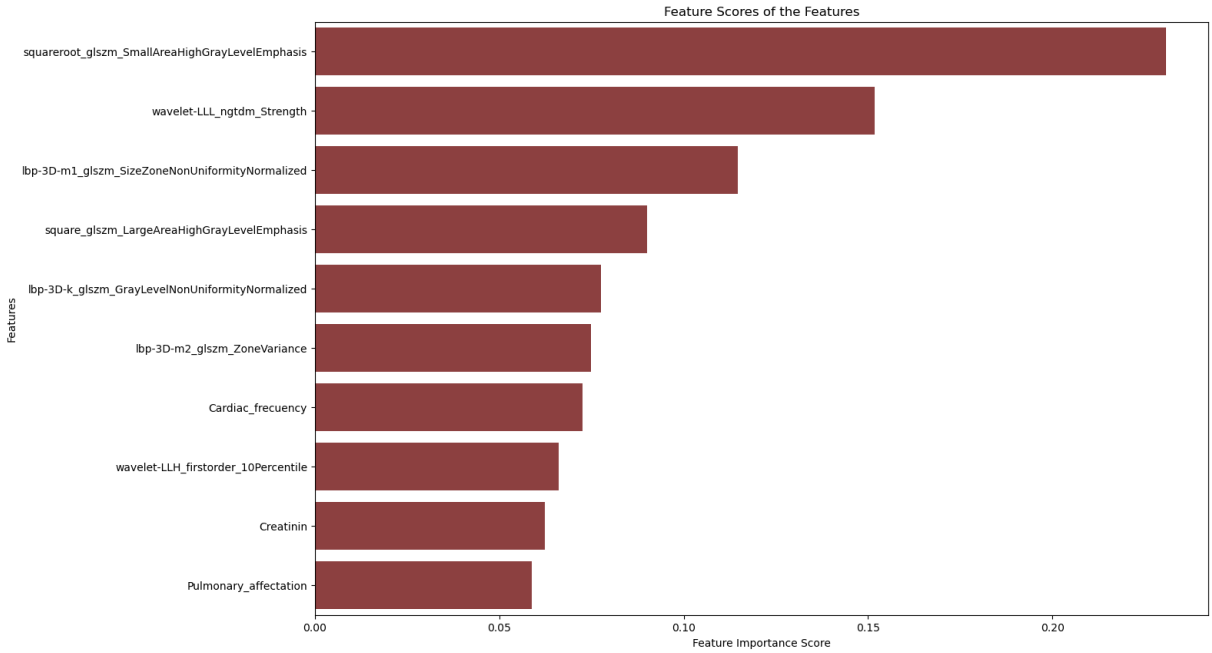


Figure 16: Feature importance with original and automatically selected data

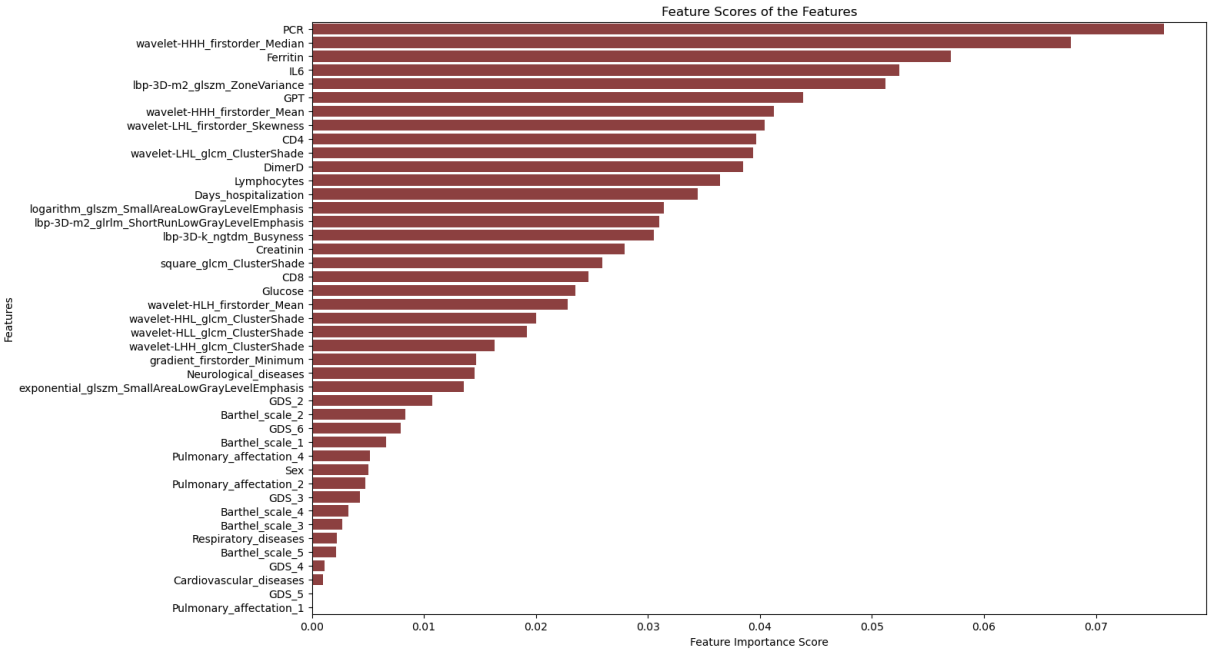


Figure 17: Feature importance with augmented and manually selected data

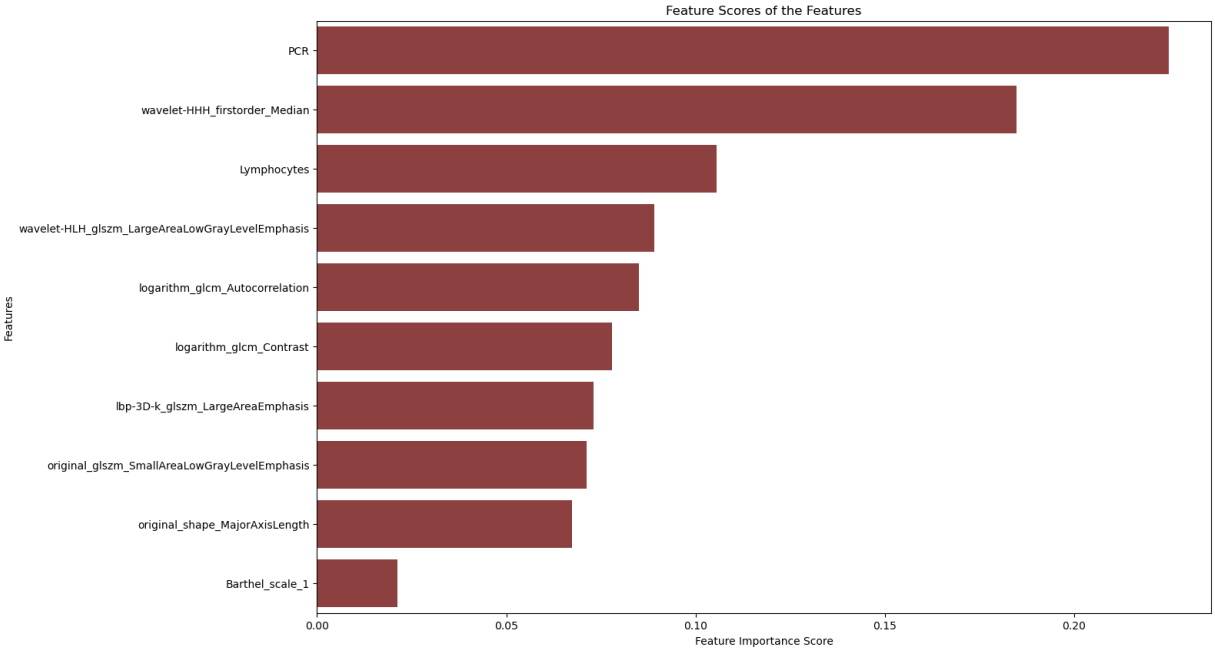


Figure 18: Feature importance with augmented and automatically selected data

II. Confusion Matrices

Decision Tree CM

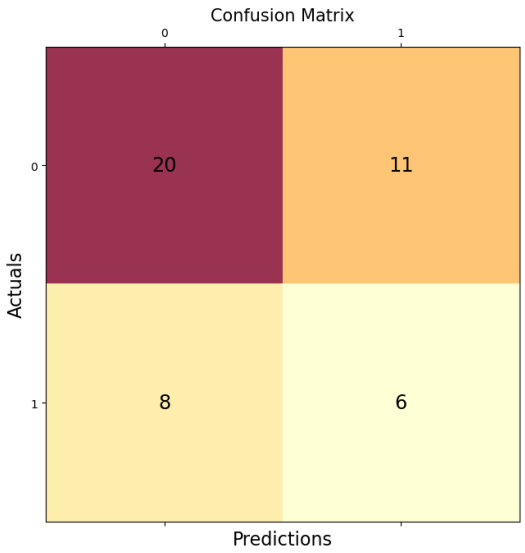


Figure 19: Confusion Matrix original data extracted manually

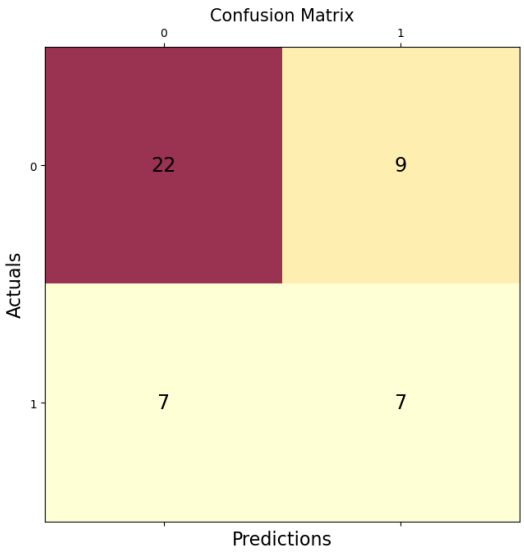


Figure 20: Confusion Matrix original data extracted automatically

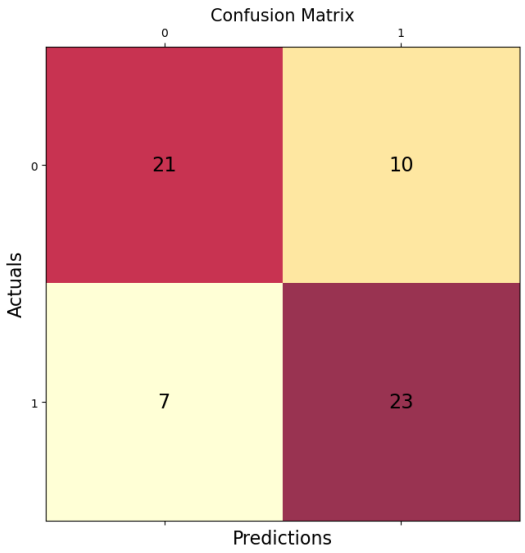


Figure 21: Confusion Matrix augmented data extracted manually

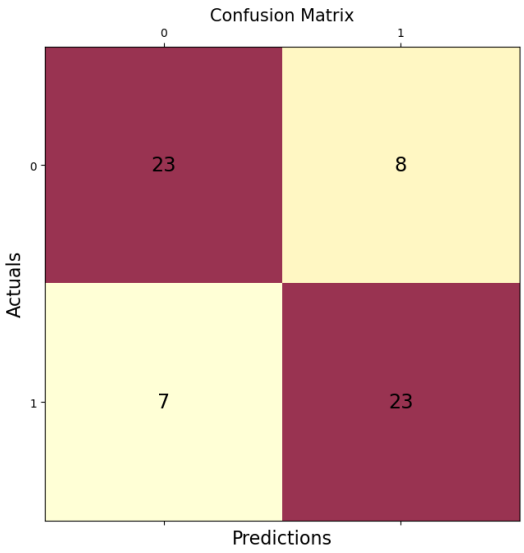


Figure 22: Confusion Matrix augmented data extracted automatically

Random Forest CM

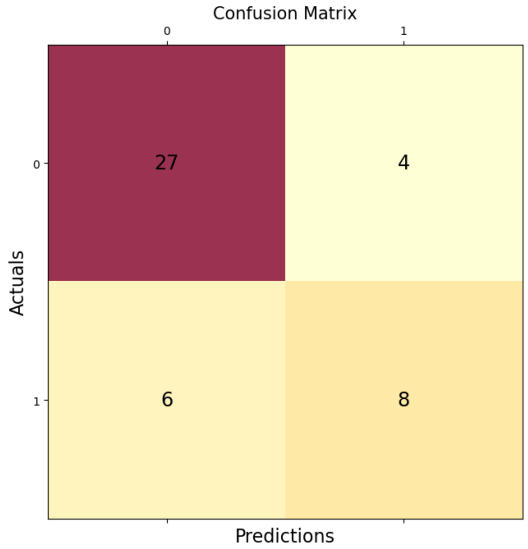


Figure 23: Confusion Matrix original data extracted manually

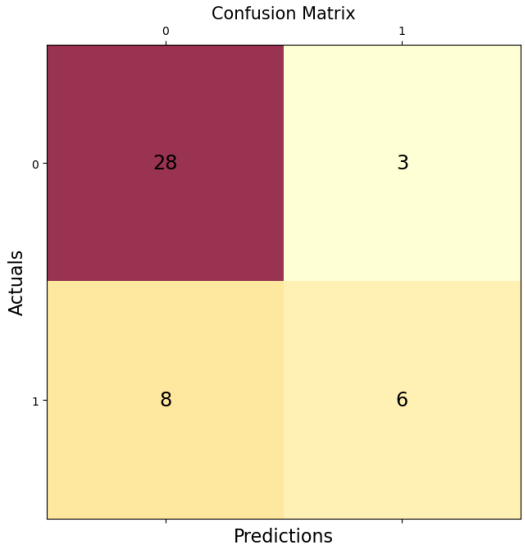


Figure 24: Confusion Matrix original data extracted automatically

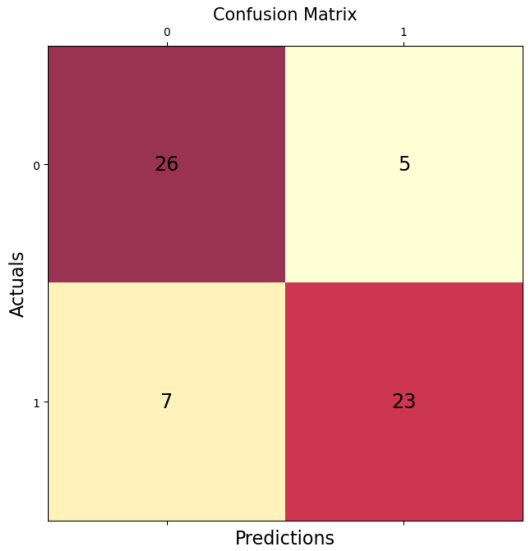


Figure 25: Confusion Matrix augmented data extracted manually

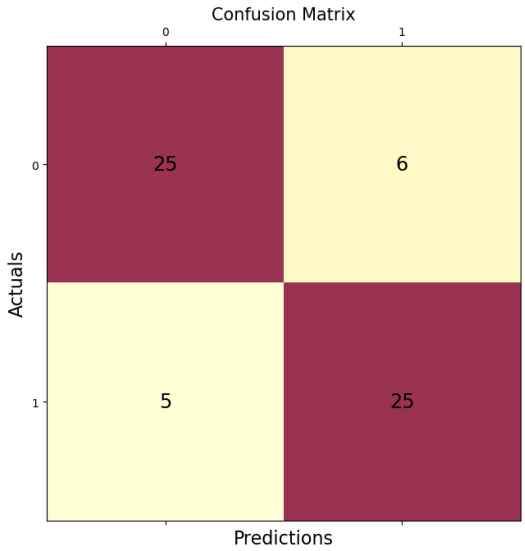


Figure 26: Confusion Matrix augmented data extracted automatically

Logistic Regression CM

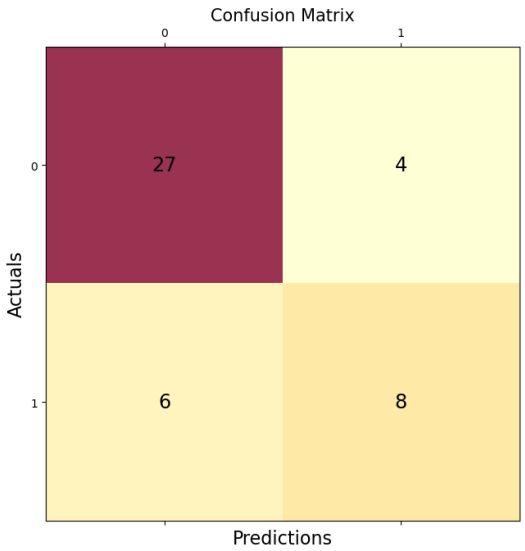


Figure 27: Confusion Matrix original data extracted manually

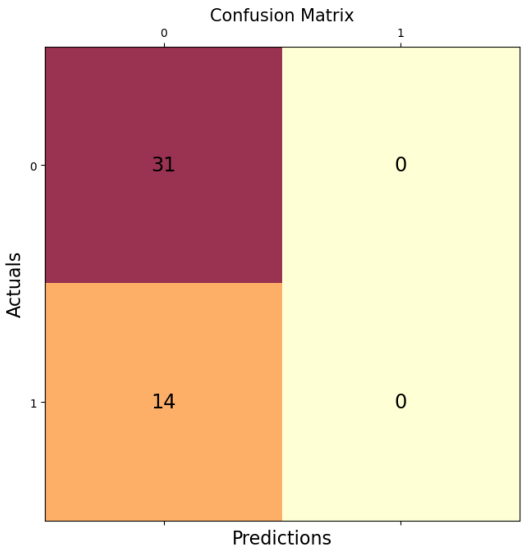


Figure 28: Confusion Matrix original data extracted automatically

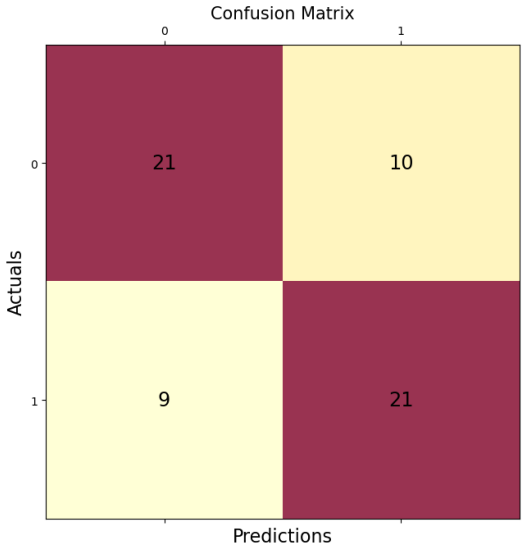


Figure 29: Confusion Matrix augmented data extracted manually

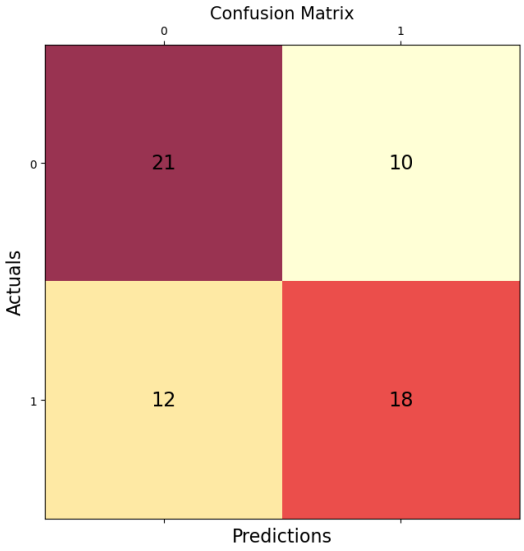


Figure 30: Confusion Matrix augmented data extracted automatically

Validation set CM

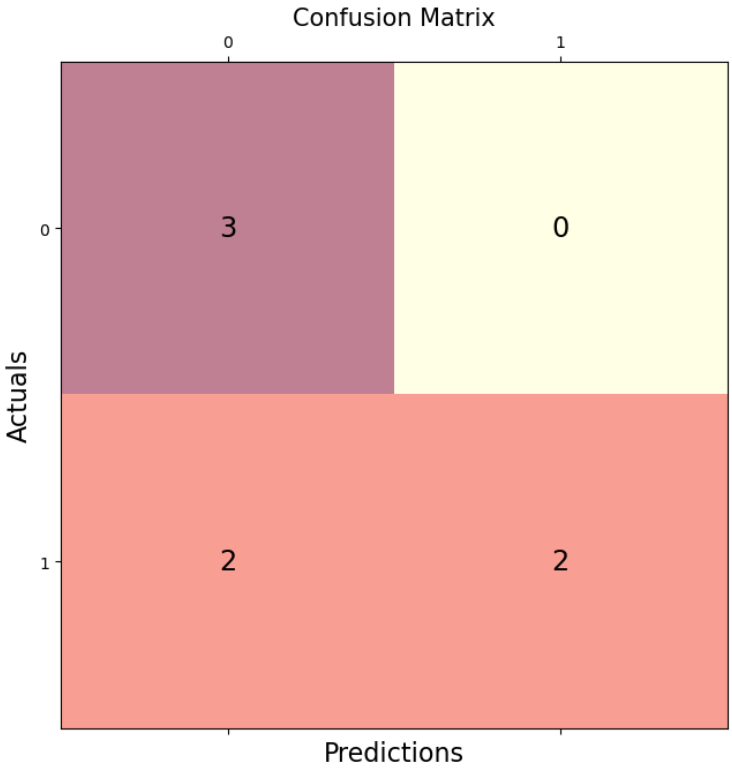


Figure 31: Confusion Matrix validation set

III. Radiomic features

Feature	Description
Wavelet HLL first order skewness	Measures the skewness of pixel intensity distribution in the HLL wavelet component.
Wavelet HHL glcm clustershade	Quantifies the shading effect of pixel clusters in the HHL wavelet component using GLCM.
Squareroot glszm small area high gray level emphasis	Emphasizes high gray levels in small areas of the GLSZM matrix.
Wavelet LLL ngtdm strength	Measures the strength of pixel gray-tone differences in the LLL wavelet component using NGTD.
Lbp 3d m1 glszm size zone non uniformity normalized	Assesses non-uniformity of zone sizes in the GLSZM using LBP 3D method M1.
Square glszm large area high gray level emphasis	Emphasizes large areas with high gray levels in the GLSZM matrix.
Lbp 3d k glszm gray level non uniformity normalized	Evaluates non-uniformity of gray levels in GLSZM with LBP 3D method K.
Lbp 3d m3 glszm zone variance	Measures variance of zones in GLSZM using LBP 3D method M3.
Wavelet hhh first order median	Computes the median of pixel intensities in the HHH wavelet component.
Lbp 3d m2 glszm zone variance	Evaluates variance of zones in GLSZM using LBP 3D method M2.
Wavelet hhh first order mean	Computes the mean of pixel intensities in the HHH wavelet component.
Wavelet lhl first order skewness	Measures the skewness of pixel intensity distribution in the LHL wavelet component.
Wavelet hhh first order median	(Repeated) Computes the median of pixel intensities in the HHH wavelet component.
Wavelet hlh glszm large area low gray level emphasis	Emphasizes large areas with low gray levels in GLSZM using HLH wavelet component.
Logarithm glcm autocorrelation	Measures pixel intensity correlation in the GLCM matrix using logarithmic transformation.
Logarithm glcm contrast	Quantifies the contrast of pixel intensities in the GLCM matrix using logarithmic transformation.
Lbp 3d k glszm large area emphasis	Emphasizes large areas in GLSZM using LBP 3D method K.
Original glszm small area low gray level emphasis	Emphasizes low gray levels in small areas of the GLSZM matrix.
Original shape major axis length	Measures the length of the major axis in the shape of the ROI.

Table 8: Radiomic feature selected description

IV. Codes

All codes used in this work are provided below and are available in the public repository: https://github.com/Judith0suna/Radiomics-AI [34].

From DICOM to Nifti

```
from DicomRTTool.ReaderWriter import DicomReaderWriter
import os
import SimpleITK as sitk

# Main directory containing DICOM data
main_directory = "C:/Users/jor14/Desktop/Not_treatment_worked"
# Directory where the output NIfTI files will be saved
output_directory = "C:/Users/jor14/Desktop/Not_treatment_worked_nii"

# Instantiate the DicomReaderWriter with a description and argument to get the maximum
Dicom_reader = DicomReaderWriter(description='Images', arg_max=True)

# Iterate through each sub-directory in the main directory
for subdir, dirs, files in os.walk(main_directory):
    for dir in dirs:
        current_path = os.path.join(subdir, dir)
        print(f"Processing {current_path}...")

        # Process DICOM data in the current directory
        Dicom_reader.walk_through_folders(current_path)
        all_rois = Dicom_reader.return_rois(print_rois=False)
        # Set the contour names and their associations
        Dicom_reader.set_contour_names_and_associations (contour_names=['z_lungs'])

        # Determine which indexes have all the required ROIs
        indexes = Dicom_reader.which_indexes_have_all_rois()

        if indexes:
            # Select the last index that contains all the required ROIs
            pt_indx = indexes[-1]
            Dicom_reader.set_index(pt_indx)
            # Retrieve images and masks for the selected index
            Dicom_reader.get_images_and_mask()

            # Prepare the output subdirectory based on the current directory's name
            subfolder_name = os.path.basename(current_path)
            output_subdir = os.path.join(output_directory, subfolder_name)
            print(f"Saving images in {output_subdir}")
            # Create the output subdirectory if it does not exist
            if not os.path.exists(output_subdir):
                os.makedirs(output_subdir)

            # Retrieve the image and mask arrays
            image = Dicom_reader.ArrayDicom
            mask = Dicom_reader.mask
            # Retrieve the SimpleITK image handles
            dicom_sitk_handle = Dicom_reader.dicom_handle
            mask_sitk_handle = Dicom_reader.annotation_handle

            # Save the images and masks as NIfTI files
```

```
            sitk.WriteImage(dicom_sitk_handle, os.path.join(output_subdir, 'Image.nii'))
            sitk.WriteImage(mask_sitk_handle, os.path.join(output_subdir, 'Mask.nii'))

            print(f"Saved Image and Mask in {output_subdir}")
        else:
            print(f"No suitable indexes found in {current_path}. Skipping...")

print("Processing completed.")
```

Code 1: Code for transforming DICOM images to NIfTI format

File 'parameters.yaml'

```
# This is an example of a parameters file
# It is written according to the YAML-convention (www.yaml.org) and is checked by the code for consistency.
# Three types of parameters are possible and reflected in the structure of the document:
#
# Parameter category:
#   Setting Name: <value>
#
# The three parameter categories are:
# - setting: Setting to use for preprocessing and class specific settings. if no <value> is specified, the value for
#   this setting is set to None.
# - featureClass: Feature class to enable, <value> is list of strings representing enabled features. If no <value> is
#   specified or <value> is an empty list ('[]'), all features for this class are enabled.
# - imageType: image types to calculate features on. <value> is custom kwarg settings (dictionary). if <value> is an
#   empty dictionary ('{}'), no custom settings are added for this input image.
#
# Some parameters have a limited list of possible values. Where this is the case, possible values are listed in the
# package documentation

# Settings to use, possible settings are listed in the documentation (section "Customizing the extraction").
setting: {correctMask: True}

# Image types to use: "Original" for unfiltered image, for possible filters, see documentation.
imageType:
  Original: {} # for dictionaries / mappings, None values are not allowed, '{}' is interpreted as an empty dictionary
  Wavelet: {}
  Square: {}
  SquareRoot: {}
  Logarithm: {}
  Exponential: {}
  Gradient: {}
  LBP2D: {}
  LBP3D: {}
# Featureclasses, from which features must be calculated. If a featureclass is not mentioned, no features are calculated
# for that class. Otherwise, the specified features are calculated, or, if none are specified, all are calculated (excluding redundant/deprecated features).
```

```
featureClass:
  # redundant Compactness 1, Compactness 2 an Spherical Disproportion features are disabled by default, they can be
  # enabled by specifying individual feature names (as is done for glcm) and including them in the list.
  shape:
  firstorder: # specifying an empty list has the same effect as specifying nothing.
  glcm:
  glrlm: # for lists none values are allowed, in this case, all features are enabled
  glszm:
  gldm:  # contains deprecated features, but as no individual features are specified, the deprecated features are not enabled
  ngtdm:
```

Code 2: Parameters file needed tu run the radiomic features extraction code

Radiomic features extraction

```python
from radiomics import featureextractor
import os
import pandas as pd
import warnings

# Suppress warnings for cleaner output
warnings.filterwarnings('ignore')

def radiomic_features_extraction(dataDir, params):
    # Initialize an empty DataFrame to store the results
    df = pd.DataFrame()

    # Count the number of subdirectories in the main directory
    count = 0
    for path in os.listdir(dataDir):
        if os.path.isdir(os.path.join(dataDir, path)):
            count += 1

    # Iterate through each subdirectory to extract radiomic features
    for i in range(1, count + 1):
        # Define the paths for the image and mask files
        imagePath = os.path.join(dataDir, str(i).zfill(2) + "/" + "Image.nii")
        labelPath = os.path.join(dataDir, str(i).zfill(2) + "/" + "Mask.nii")

        print(imagePath)
        print(labelPath)

        # Initialize the feature extractor with the provided parameters
        extractor = featureextractor.RadiomicsFeatureExtractor(params)
        extractor.enableAllFeatures()  # Enable all available features

        try:
            # Execute feature extraction and store the results
            result = extractor.execute(imagePath, labelPath)

            # Append the results to the DataFrame
            df = df._append(result, ignore_index=True)
        except Exception as e:
            # Print any errors that occur during feature extraction
```

```
            print(f"Error processing case {i}: {e}")

    # Drop columns containing "diagnostics" in their names
    for i in df.columns.values:
        if "diagnostics" in i:
            df.drop([i], axis=1, inplace=True)

    return df

# Define directories and parameter file
dataDir = "C:/Users/jor14/Desktop/Treatment_worked_Nifti/"
params = "C:/Users/jor14/Desktop/parameters.yaml"

# Extract features from the first dataset
df_worked = radiomic_features_extraction(dataDir, params)

# Display the first 34 rows of the extracted features
df_worked.head(34)

# Define directories and parameter file for the second dataset
dataDir = "C:/Users/jor14/Desktop/Not_treatment_worked_Nifti/"
params = "C:/Users/jor14/Desktop/parameters.yaml"

# Extract features from the second dataset
df_not_worked = radiomic_features_extraction(dataDir, params)

# Display the first 16 rows of the extracted features
df_not_worked.head(16)

print("TIME TO CONCATENATE")

# Assign a result label to each DataFrame
df_worked = df_worked.assign(Result=1)
df_not_worked = df_not_worked.assign(Result=0)
# Concatenate the two DataFrames
radiomic_features = pd.concat([df_worked, df_not_worked])
# Reset the index of the concatenated DataFrame
radiomic_features.index = range(radiomic_features.shape[0])

# Display the first few rows of the combined DataFrame
radiomic_features.head()

# Save the combined DataFrame to an Excel file
radiomic_features.to_excel('radiomic_features_excel.xlsx', index=False)
print("DONE")
```

Code 3: Code for extracting radiomic features from NIfTI images

Clinical data preparation

```
import pandas as pd
import numpy as np
from sklearn.experimental import enable_iterative_imputer
from sklearn.impute import IterativeImputer

# Load the data from the Excel file
df_clinical = pd.read_excel('clinical_features_excel_id.xlsx')
```

```

# Rename 'Gender' to 'Sex' and convert 'MALE' to 1 and others to 0
df_clinical.rename(columns={'Gender': 'Sex'}, inplace=True)
df_clinical['Sex'] = np.where(df_clinical['Sex'] == 'MALE', 1, 0)

# Separate the 'ID' column
ids = df_clinical['ID']  # Save the IDs separately
df_without_id = df_clinical.drop(columns=['ID'])

# Apply imputation to fill missing values
imputer = IterativeImputer(random_state=0)
imputed_data = imputer.fit_transform(df_without_id)
df_imputed_no_id = pd.DataFrame(imputed_data, columns=df_without_id.columns)

# Exclude 'PO2_venous' from integer conversion
column_to_exclude = 'PO2_venous'
df_imputed_no_id[df_imputed_no_id.columns.difference ([column_to_exclude])] =
    df_imputed_no_id[
    df_imputed_no_id.columns.difference([column_to_exclude])
].apply(np.floor).astype(int)

# Add the 'ID' column back to the imputed DataFrame
df_imputed_with_id = df_imputed_no_id.copy()
df_imputed_with_id['ID'] = ids.values

# List of columns to transform with one-hot encoding
columns_to_transform = ['GDS', 'Barthel_scale', 'Pulmonary_affectation']

# Create dummies/one-hot encoding for the specified columns
df_one_hot_encoded = pd.get_dummies(df_imputed_with_id, columns=columns_to_transform,
    prefix=columns_to_transform)

# Replace any boolean values with integers (False = 0, True = 1)
df_one_hot_encoded.replace({False: 0, True: 1}, inplace=True)

# Merge the encoded DataFrame with the results DataFrame based on 'ID'
df_combined_clinical = pd.merge(df_one_hot_encoded, dfresult, on='ID', how='outer')

# Display the first few rows of the combined DataFrame
print(df_combined_clinical.head())

# Save the resulting DataFrame with 'ID' to a new Excel file
df_combined_clinical.to_excel('clinical_ID.xlsx', index=False)

print("File 'clinical_ID.xlsx' created successfully.")
```

Code 4: Code for filling in missing values and processing categorical characteristics of clinical data

Metabolomic data preparation

```
import pandas as pd
from sklearn.experimental import enable_iterative_imputer
from sklearn.impute import IterativeImputer

# Load the data from the Excel file
df_metabolomics = pd.read_excel('metabolomic_features_id.xlsx')
```

```

# Separate the 'ID' column
ids = df_metabolomics['ID']  # Save the IDs separately
df_without_id = df_metabolomics.drop(columns=['ID'])

# Apply imputation to fill missing values
imputer = IterativeImputer(random_state=0)
imputed_data = imputer.fit_transform(df_without_id)
df_imputed_no_id = pd.DataFrame(imputed_data, columns=df_without_id.columns)

# Add the 'ID' column back to the imputed DataFrame
df_imputed_with_id = df_imputed_no_id.copy()
df_imputed_with_id['ID'] = ids.values

# Display the first few rows of the imputed DataFrame
print(df_imputed_with_id.head())

# Save the imputed DataFrame with 'ID' to a new Excel file
df_imputed_with_id.to_excel('metabolomic_ID.xlsx', index=False)

print("File 'metabolomic_ID.xlsx' created successfully.")
```

Code 5: Code to fill in missing values in metabolomics data

Clinical and Metabolomic data augmentation

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from imblearn.over_sampling import SMOTE

# Load the data from the Excel file
df = pd.read_excel('metabolomic.xlsx')

# Separate features (X) and the target variable (y)
X = df.drop('Result', axis=1)  # Replace 'Result' with the name of your target column
y = df['Result']

# Apply SMOTE to balance the data
smote = SMOTE(sampling_strategy='auto', random_state=42)
X_resampled, y_resampled = smote.fit_resample(X, y)

# Combine the resampled features and target variable into a DataFrame
df_resampled = pd.concat([pd.DataFrame(X_resampled, columns=X.columns),
    pd.DataFrame(y_resampled, columns=['Result'])], axis=1)

# Save the balanced DataFrame to a new Excel file
df_resampled.to_excel('metabolomic_augmented.xlsx', index=False)

print("The data has been balanced and saved in 'metabolomic_augmented.xlsx'")

# Plot the distribution of a selected feature
feature = 'LDH'  # Replace 'LDH' with the column name you want to plot

# Original data (Result=0)
original_data = df[df['Result'] == 0][feature]

```

```
# Synthetic data (Result=0, after SMOTE)
synthetic_data = df_resampled[(df_resampled['Result'] == 0) &
    (~df_resampled.index.isin(df.index))][feature]

# Plotting the distributions
plt.figure(figsize=(12, 6))

# Histogram and KDE of original data
sns.histplot(original_data, kde=True, color='blue', stat="density", label='Original
    (Result=0)', bins=20)

# Histogram and KDE of synthetic data
sns.histplot(synthetic_data, kde=True, color='red', stat="density", label='Synthetic
    (SMOTE)', bins=20)

plt.title(f'Gaussian Distribution of {feature} - Original vs Synthetic Data')
plt.xlabel(feature)
plt.ylabel('Density')
plt.legend()
plt.show()
```

Code 6: Code to increase the unbalanced negative response on clinical and metabolomic data

CT images augmentation

```
import nibabel as nib
import numpy as np
import matplotlib.pyplot as plt
import os
import shutil
import torch
from skimage import exposure


# Directory containing the numbered subfolders
base_dir = 'C:/Users/jor14/Desktop/Not_treatment_worked_Nifti/'  # Change this to your
    base directory path

# Create a folder to save synthetic images and masks
output_dir = 'synthetic_images_real'
if os.path.exists(output_dir):
    shutil.rmtree(output_dir)  # Remove the existing directory if it already exists
os.makedirs(output_dir)

# Function to add random noise to an image
def add_random_noise(image_np, noise_level=0.1):
    noise = np.random.normal(loc=0, scale=noise_level, size=image_np.shape)
    return image_np + noise


# Function to adjust brightness and contrast of an image
def adjust_brightness_contrast(image_np, brightness_factor=1.0, contrast_factor=1.0):
    # Adjust brightness
    image_np = image_np * brightness_factor
    # Adjust contrast
    p2, p98 = np.percentile(image_np, (2, 98))
```

```
    image_np = exposure.rescale_intensity(image_np, in_range=(p2, p98)) * contrast_factor
    return np.clip(image_np, 0, 255)

# Function to apply random flipping to an image and its mask
def apply_random_flipping(image_np, mask_np):
    if np.random.rand() > 0.5:
        image_np = np.flip(image_np, axis=0)
        mask_np = np.flip(mask_np, axis=0)
    if np.random.rand() < 0.5:
        image_np = np.flip(image_np, axis=1)
        mask_np = np.flip(mask_np, axis=1)
    return image_np, mask_np

# Function to generate synthetic images from an original image and mask
def generate_synthetic_image(image_np, mask_np):
    # Add random noise
    noisy_image = add_random_noise(image_np)

    # Adjust brightness and contrast
    brightness_factor = np.random.uniform(0.8, 1.2)  # Random brightness factor
    contrast_factor = np.random.uniform(0.8, 1.2)    # Random contrast factor
    final_image = adjust_brightness_contrast(noisy_image, brightness_factor,
        contrast_factor)

    # Apply random flipping
    final_image, final_mask = apply_random_flipping(final_image, mask_np)

    return final_image, final_mask

num_original_images = 16 # Number of original images to generate
num_synthetic_images = 18  # Number of synthetic images to generate

# Store original images and masks
original_images = []
original_masks = []

# Load 16 original images and their masks
for i in range(1, 17):
    folder_path = os.path.join(base_dir, f'{i:02d}')
    image_path = os.path.join(folder_path, 'Image.nii')
    mask_path = os.path.join(folder_path, 'Mask.nii')

    # Load image and mask
    image = nib.load(image_path)
    mask = nib.load(mask_path)

    # Convert data to numpy arrays
    image_data_np = image.get_fdata()
    mask_data_np = mask.get_fdata()

    # Convert numpy arrays to PyTorch tensors
    image_data_tensor = torch.from_numpy(image_data_np.astype(np.float32))
    mask_data_tensor = torch.from_numpy(mask_data_np.astype(np.float32))

    # Store tensors and affine matrices in lists
    original_images.append((image_data_tensor, image.affine))
    original_masks.append((mask_data_tensor, mask.affine))

# Generate and save synthetic images and masks
```

```
for i in range(1, num_synthetic_images + 1):
    # Select a random original image to generate a synthetic image
    idx = (i - 1) % num_original_images
    original_image_tensor, affine_image = original_images[idx]
    original_mask_tensor, affine_mask = original_masks[idx]

    # Convert tensors to numpy arrays
    original_image_np = original_image_tensor.numpy()
    original_mask_np = original_mask_tensor.numpy()

    # Generate synthetic data
    synthetic_image_np, synthetic_mask_np = generate_synthetic_image(original_image_np,
        original_mask_np)

    # Create a subfolder for each synthetic image
    output_subdir = os.path.join(output_dir, f'{i:02d}')
    os.makedirs(output_subdir)

    # Create NIfTI images for synthetic data
    synthetic_image = nib.Nifti1Image(synthetic_image_np, affine_image)
    synthetic_mask = nib.Nifti1Image(synthetic_mask_np, affine_mask)

    # Save synthetic images and masks in the corresponding subfolder
    nib.save(synthetic_image, os.path.join(output_subdir, 'Image.nii'))
    nib.save(synthetic_mask, os.path.join(output_subdir, 'Mask.nii'))

# Compare intensity distributions of original and synthetic images
original_sample = original_images[0][0].numpy().flatten()
synthetic_sample = synthetic_data_np.flatten()

plt.figure(figsize=(12, 6))
plt.hist(original_sample, bins=50, alpha=0.5, label='Original')
plt.hist(synthetic_sample, bins=50, alpha=0.5, label='Synthetic')
plt.legend(loc='upper right')
plt.title('Intensity Distribution')
plt.xlabel('Intensity')
plt.ylabel('Frequency')
plt.show()
```

Code 7: Code for augmenting unbalanced Computed Tomography Images

Preprocessing all data manually

```
# Import necessary libraries
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

# For data imputation and scaling
from sklearn.experimental import enable_iterative_imputer
from sklearn.impute import IterativeImputer
from sklearn.preprocessing import StandardScaler

# For calculating Variance Inflation Factor (VIF)
from statsmodels.stats.outliers_influence import variance_inflation_factor

```

```
import warnings
warnings.filterwarnings("ignore")

# Load the dataset
df_radiomic = pd.read_excel('clinical_nulls.xlsx')

# Remove columns with constant values
df_radiomic = df_radiomic.loc[:, (df_radiomic != df_radiomic.iloc[0]).any()]

# List to store names of columns with object data type
object_columns = []

# Check the data type of each column
for col in df_radiomic.columns:
    print(col, ': ', df_radiomic[col].dtype)
    if df_radiomic[col].dtype == 'object':
        object_columns.append(col)

# Check if there are any object type columns and print them
if object_columns:
    print("\nThe following variables are of type object in the DataFrame:")
    for col in object_columns:
        print(col)
else:
    print("\nThere are no object-type variables in the DataFrame.")

# Extract the independent variables (X) and dependent variable (y)
X = df_radiomic.drop('Result', axis=1)
y = df_radiomic.Result

# Display the distribution of the target variable
print(y.value_counts())

# Calculate the correlation matrix
corr_df = df_radiomic.corr(method='pearson')
print(corr_df.head())

# Plot the correlation matrix
sns.set(rc={'figure.figsize':(11.7, 8.27)})
plt.figure(2)
sns.heatmap(corr_df)
plt.show(block=False)

# Identify highly correlated variables
highly_correlated_vars = set()
threshold = 0.75
target_variable = "Result"

for i in range(len(corr_df.columns)):
    for j in range(i):
        if abs(corr_df.iloc[i, j]) > threshold:
            var1 = corr_df.columns[i]
            var2 = corr_df.columns[j]
            corr_with_target_var1 = abs(df_radiomic[var1].corr(df_radiomic
            [target_variable]))
            corr_with_target_var2 = abs(df_radiomic[var2].corr(df_radiomic
            [target_variable]))
            if corr_with_target_var1 < corr_with_target_var2:
                highly_correlated_vars.add(var1)
```

```
            else:
                highly_correlated_vars.add(var2)

# Drop highly correlated variables
df_radiomic.drop(columns=highly_correlated_vars, inplace=True)
X.drop(columns=highly_correlated_vars, inplace=True)

# Recalculate the correlation matrix if needed
correlation_matrix_updated = df_radiomic.corr(method='pearson')

# Function to calculate and filter by VIF
def calculate_vif(x, thresh=5):
    output = x.copy()
    while True:
        vif = [variance_inflation_factor(output.values, i) for i in range(output.shape[1])]
        max_vif = max(vif)
        if max_vif > thresh:
            max_index = vif.index(max_vif)
            output = output.drop(output.columns[max_index], axis=1)
        else:
            break
    return output

# Select features based on VIF
selected_features = calculate_vif(X)
print(selected_features.head())

# Get the names of the selected columns
selected_features_columns = selected_features.columns

# Update df_radiomic to only contain the selected columns and the 'Result' column
df_radiomic_cleaned = df_radiomic[list(selected_features_columns) + ['Result']]

# Initialize the standard scaler
scaler = StandardScaler()

# Select the columns to standardize (all except 'Result')
columns_to_scale = df_radiomic_cleaned.drop(columns='Result')

# Apply standardization to the selected columns
df_radiomic_cleaned[columns_to_scale.columns] = scaler.fit_transform(columns_to_scale)

# Save the cleaned and standardized DataFrame to an Excel file
df_radiomic_cleaned.to_excel('MetabolicData.xlsx', index=False)
```

Code 8: Code to prepare manually all data for the Predictive Model

Preprocessing all data automatically

```
import pandas as pd
import pymrmr
from sklearn.preprocessing import MinMaxScaler

# Load the dataset from an Excel file
data = pd.read_excel('C:/Users/jor14/Desktop/radiomics&metabolomic&clinical.xlsx')

# Assuming the 'Result' column is the target variable
```

```
X = data.drop(columns=['Result'])  # Features (all columns except 'Result')
y = data['Result']  # Target variable

# Scale the features to a range between 0 and 1
scaler = MinMaxScaler()
X_scaled = scaler.fit_transform(X)  # Fit and transform the features

# Concatenate the scaled features and the target variable for use in pymrmr
data_for_mrmr = pd.concat([y, pd.DataFrame(X_scaled, columns=X.columns)], axis=1)

# Select the top 10 features using the mRMR method with the 'MIQ' criterion
selected_features = pymrmr.mRMR(data_for_mrmr, 'MIQ', 10)  # 'MIQ' stands for Mutual
    Information Quotient

# Add the target variable to the list of selected features
selected_features_with_target = ['Result'] + selected_features

# Create a new DataFrame with only the selected features and the target variable
selected_features_df = data[selected_features_with_target]

# Save the resulting DataFrame to a new Excel file
selected_features_df.to_excel('C:/Users/jor14/Desktop/ALLpreprocessed_auto.xlsx',
    index=False)

# Display the first few rows of the resulting DataFrame
print("DataFrame with the selected features and the target variable:")
print(selected_features_df.head())
```

Code 9: Code to select features automatically for the Predictive Model

Predictive Model with RF

```
# Import necessary libraries
import pandas as pd  # For data manipulation
import numpy as np  # For numerical operations
import matplotlib.pyplot as plt  # For plotting graphs
import seaborn as sns  # For data visualization

import warnings  # To ignore warnings
warnings.filterwarnings("ignore")

from sklearn.model_selection import train_test_split, cross_val_predict, StratifiedKFold
    # For data splitting and cross-validation
from sklearn.ensemble import RandomForestClassifier  # For Random Forest model
from sklearn.metrics import confusion_matrix, recall_score, f1_score, precision_score,
    accuracy_score, roc_curve, auc  # For model evaluation metrics
from sklearn.model_selection import learning_curve  # For plotting learning curves

# Load the dataset
df = pd.read_excel('C:/Users/jor14/Desktop/allpreprocessed_auto.xlsx')

# Separate the dataset into features (X) and target variable (y)
X = df.drop('Result', axis=1)
y = df['Result']
print(X.shape)

```

```
# Split the dataset into training and testing sets with stratification to maintain class proportions
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.1, train_size=0.9, stratify=y, random_state=42)

# Convert target variables to arrays for compatibility with scikit-learn
y_train = y_train.values
y_test = y_test.values

# Convert feature datasets to arrays for compatibility with scikit-learn
X_train = X_train.values
X_test = X_test.values

# Extract the feature names
feature_names = X.columns
print(feature_names)

# Initialize and train the Random Forest Classifier
clf = RandomForestClassifier(n_estimators=100, random_state=42)
clf.fit(X_train, y_train)

# Calculate feature importance scores
feature_scores = pd.Series(clf.feature_importances_, index=X.columns).sort_values(ascending=False)

# Select the top n features based on their importance scores
n = 9
selected_features = feature_scores.index.values[:n]
print(selected_features)

# Get the indices of the selected features
cols_index = [X.columns.tolist().index(col) for col in selected_features]

# Plot the feature importance scores
f, ax = plt.subplots(figsize=(15, 10))
sns.barplot(x=feature_scores.values, y=feature_scores.index, ax=ax, color='maroon', alpha=0.8)
ax.set_title("Feature scores of the features")
ax.set_yticklabels(feature_scores.index)
ax.set_xlabel("Feature importance score")
ax.set_ylabel("Features")
plt.show()

# Reduce the training and testing datasets to the selected features
X_train = X_train[:, cols_index]
X_test = X_test[:, cols_index]

#-------------------------------------------------------------
# MODIFY THIS PART OF THE CODE FOR LR AND DT PM
#-------------------------------------------------------------
# Implement cross-validation with stratified folds
cv = StratifiedKFold(n_splits=5)
pred_cv = cross_val_predict(clf, X_train, y_train, cv=cv)
#-------------------------------------------------------------

# Calculate ROC curve and AUC
fpr, tpr, thresholds = roc_curve(y_train, pred_cv)
roc_auc = auc(fpr, tpr)

```

```
# Function to print the confusion matrix in a readable format
def print_confusion_matrix(y_true, y_pred):
    cm = confusion_matrix(y_true, y_pred)
    print('True positive = ', cm[1][1])
    print('False positive = ', cm[0][1])
    print('False negative = ', cm[1][0])
    print('True negative = ', cm[0][0])

# Generate and print the confusion matrix
conf_matrix = confusion_matrix(y_train, pred_cv)
conf_matrix = np.flipud(conf_matrix)
conf_matrix = np.fliplr(conf_matrix)

# Print various model evaluation metrics
print('Accuracy: {:.2f}'.format(accuracy_score(y_train, pred_cv)))
print('Precision: {:.2f}'.format(precision_score(y_train, pred_cv)))
print('Recall: {:.2f}'.format(recall_score(y_train, pred_cv)))
print('F1 Score: {:.2f}'.format(f1_score(y_train, pred_cv)))
print('AUC: {:.2f}'.format(auc(fpr, tpr)))
print('Confusion Matrix:')
print(conf_matrix)
print_confusion_matrix(y_train, pred_cv)

# Extract confusion matrix components for specificity calculation
tn, fp, fn, tp = confusion_matrix(y_train, pred_cv).ravel()

# Calculate specificity
specificity = tn / (tn + fp)
print(f'Specificity: {specificity:.2f}')

# Plot ROC curve
plt.figure()
plt.plot(fpr, tpr, color='maroon', lw=2, label='AUC = %0.2f' % roc_auc)
plt.plot([0, 1], [0, 1], color='gray', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend(loc="lower right")
plt.show()

# Plot the confusion matrix as a heatmap
fig, px = plt.subplots(figsize=(7.5, 7.5))
px.matshow(conf_matrix, cmap=plt.cm.YlOrRd, alpha=0.8)
for m in range(conf_matrix.shape[0]):
    for n in range(conf_matrix.shape[1]):
        px.text(x=m, y=n, s=conf_matrix[n, m], va='center', ha='center', size='xx-large')

# Set labels and title for the confusion matrix plot
plt.xlabel('Predictions', fontsize=16)
plt.ylabel('Actuals', fontsize=16)
plt.title('Confusion Matrix', fontsize=15)
plt.show()

# Generate learning curves
train_sizes, train_scores, valid_scores = learning_curve(clf, X, y, cv=5, scoring='f1',
    train_sizes=np.linspace(0.1, 1.0, 10))

```

```
# Calculate the mean and standard deviation of the training and validation scores
train_mean = np.mean(train_scores, axis=1)
train_std = np.std(train_scores, axis=1)
valid_mean = np.mean(valid_scores, axis=1)
valid_std = np.std(valid_scores, axis=1)

# Plot the learning curves
plt.figure(figsize=(10, 6))
plt.plot(train_sizes, train_mean, 'o-', color='purple', label='Training')
plt.plot(train_sizes, valid_mean, 'o-', color='red', label='Cross-Validation')

# Fill between the standard deviation bands for training and validation curves
plt.fill_between(train_sizes, train_mean - train_std, train_mean + train_std,
    color='purple', alpha=0.1)
plt.fill_between(train_sizes, valid_mean - valid_std, valid_mean + valid_std, color='red',
    alpha=0.1)

# Set plot title and labels
plt.title('Learning Curves')
plt.xlabel('Training Set Size')
plt.ylabel('F1 Score')
plt.legend(loc='best')
plt.show()
```

Code 10: Code to train and evaluate the Random Forest classifier

Predictive Model with LR

```
#-----------------------------------------------------------------
# Import necessary libraries
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import StratifiedKFold, cross_val_predict

# Initialize the Logistic Regression model
lr = LogisticRegression(random_state=42)

# Perform Stratified K-Fold Cross-Validation and get predictions
cv = StratifiedKFold(n_splits=5)
pred_cv = cross_val_predict(lr, X_train, y_train, cv=cv, method='predict')
#-----------------------------------------------------------------
```

Code 11: Code to modify the Predictive Model with Logistic Regression

Predictive Model with DT

```
#-----------------------------------------------------------------
# Import necessary libraries
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import StratifiedKFold, cross_val_predict

# Initialize the Decision Tree model
dt = DecisionTreeClassifier(random_state=42)

# Perform Stratified K-Fold Cross-Validation and get predictions
```

```
cv = StratifiedKFold(n_splits=5)
pred_cv = cross_val_predict(dt, X_train, y_train, cv=cv, method='predict')
#-------------------------------------------------------------
```

Code 12: Code to modify the Predictive Model with Decision Tree

Best PM with OPTUNA and testing

```

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings

from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split, cross_val_score, cross_val_predict,
    StratifiedKFold
from sklearn.metrics import confusion_matrix, recall_score, f1_score, precision_score,
    accuracy_score, roc_curve, auc
import optuna

# Suppress warnings
warnings.filterwarnings("ignore")

# Load the dataset
df = pd.read_excel('C:/Users/jor14/Desktop/ALLaugmented_featureauto.xlsx')

# Extract features (independent variables) and target (dependent variable)
X = df.drop('Result', axis=1)
y = df['Result']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.1, train_size=0.9,
    stratify=y, random_state=42)

# Convert variables to arrays
y_train = y_train.values
X_test = X_test.values
X_train = X_train.values
y_test = y_test.values

# Get the feature names
feature_names = X.columns
print(feature_names)

# Initialize and fit the RandomForest model
clf = RandomForestClassifier(n_estimators=100, random_state=42)
clf.fit(X_train, y_train)

# Calculate feature importance scores
feature_importance_scores = pd.Series(clf.feature_importances_,
    index=X.columns).sort_values(ascending=False)
n=10

# Display feature importance scores
print(feature_importance_scores)
```

```
46
47 # Select the top n features
48 selected_features = feature_importance_scores.index.values[:n]
49 print(selected_features)
50
51 # Get the indices of the selected features
52 cols_index = [X.columns.tolist().index(col) for col in selected_features]
53
54 # Plot the feature importance scores
55 plt.figure(figsize=(15, 10))
56 sns.barplot(x=feature_importance_scores.values, y=feature_importance_scores.index,
       color='maroon', alpha=0.8)
57
58 # Set plot title and labels
59 plt.title("Feature Importance Scores")
60 plt.xlabel("Importance Score")
61 plt.ylabel("Features")
62
63 # Show the plot
64 plt.show()
65
66 # Reduce the training and testing sets to the selected features
67 X_train = X_train[:, cols_index]
68 X_test = X_test[:, cols_index]
69
70 # Objective function for hyperparameter tuning with Optuna
71 def objective(trial):
72     # Define the hyperparameters to tune
73     n_estimators = trial.suggest_int('n_estimators', 10, 200)
74     max_depth = trial.suggest_int('max_depth', 2, 32)
75     min_samples_split = trial.suggest_int('min_samples_split', 2, 10)
76     min_samples_leaf = trial.suggest_int('min_samples_leaf', 1, 4)
77     bootstrap = trial.suggest_categorical('bootstrap', [True, False])
78
79     # Create the model with the suggested hyperparameters
80     model = RandomForestClassifier(
81         n_estimators=n_estimators,
82         max_depth=max_depth,
83         min_samples_split=min_samples_split,
84         min_samples_leaf=min_samples_leaf,
85         bootstrap=bootstrap,
86         random_state=42
87     )
88
89     # Evaluate the model using cross-validation
90     cv = StratifiedKFold(n_splits=5)
91     scores = cross_val_score(model, X, y, cv=cv, scoring='f1')
92
93     return scores.mean()
94
95 # Create and optimize the study with Optuna
96 study = optuna.create_study(direction='maximize')
97 study.optimize(objective, n_trials=50)
98
99 # Print the best parameters and best score found by Optuna
100 print(f"Best Parameters: {study.best_params}")
101 print(f"Best Score (F1): {study.best_value}")
102
103 # Train the best model with the selected features and parameters
```

```
best_params = study.best_params
best_model = RandomForestClassifier(**best_params, random_state=42)
best_model.fit(X_train, y_train)

# Evaluate the model on the training set using cross-validation predictions
y_probs = cross_val_predict(best_model, X_train, y_train, cv=StratifiedKFold(n_splits=5),
    method="predict")
fpr, tpr, thresholds = roc_curve(y_train, y_probs)
roc_auc = auc(fpr, tpr)

# Define function to print confusion matrix details
def print_confusion_matrix(y_true, y_pred):
    cm = confusion_matrix(y_true, y_pred)
    print('True Positive = ', cm[1][1])
    print('False Positive = ', cm[0][1])
    print('False Negative = ', cm[1][0])
    print('True Negative = ', cm[0][0])

# Print model evaluation metrics
print('Accuracy: {:.2f}'.format(accuracy_score(y_train, y_probs)))
print('Precision: {:.2f}'.format(precision_score(y_train, y_probs)))
print('Recall: {:.2f}'.format(recall_score(y_train, y_probs)))
print('F1 Score: {:.2f}'.format(f1_score(y_train, y_probs)))
print('AUC: {:.2f}'.format(roc_auc))

# Convert probabilities to binary predictions
y_pred = (y_probs >= 0.5).astype(int)

# Confusion matrix for the training set
cm_train = confusion_matrix(y_train, y_pred)
print('Confusion Matrix:')
print(cm_train)
print_confusion_matrix(y_train, y_pred)

# Plot ROC curve
plt.figure()
plt.plot(fpr, tpr, color='maroon', lw=2, label='AUC = %0.2f' % roc_auc)
plt.plot([0, 1], [0, 1], color='gray', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend(loc="lower right")
plt.show()

# Plot the confusion matrix
fig, ax = plt.subplots(figsize=(7.5, 7.5))
ax.matshow(cm_train, cmap=plt.cm.YlOrRd, alpha=0.8)
for i in range(cm_train.shape[0]):
    for j in range(cm_train.shape[1]):
        ax.text(x=i, y=j, s=cm_train[j, i], va='center', ha='center', size='xx-large')

# Set labels and title
plt.xlabel('Predictions', fontsize=16)
plt.ylabel('Actuals', fontsize=16)
plt.title('Confusion Matrix', fontsize=15)
plt.show()

```

```
# Plot learning curves
train_sizes, train_scores, valid_scores = learning_curve(best_model, X_train, y_train,
    cv=5, scoring='f1', train_sizes=np.linspace(0.1, 1.0, 10))

# Calculate mean and standard deviation of the scores
train_mean = np.mean(train_scores, axis=1)
train_std = np.std(train_scores, axis=1)
valid_mean = np.mean(valid_scores, axis=1)
valid_std = np.std(valid_scores, axis=1)

# Plot the learning curves
plt.figure(figsize=(10, 6))
plt.plot(train_sizes, train_mean, 'o-', color='purple', label='Training')
plt.plot(train_sizes, valid_mean, 'o-', color='red', label='Cross-Validation')

# Fill between the standard deviation bands
plt.fill_between(train_sizes, train_mean - train_std, train_mean + train_std,
    color='purple', alpha=0.1)
plt.fill_between(train_sizes, valid_mean - valid_std, valid_mean + valid_std, color='red',
    alpha=0.1)

# Set plot title and labels
plt.title('Learning Curves')
plt.xlabel('Training Set Size')
plt.ylabel('F1 Score')
plt.legend(loc='best')
plt.show()

# Evaluate the model on the test set
clf = RandomForestClassifier(n_estimators=100, random_state=42)
clf.fit(X_train, y_train)
pred_test = clf.predict(X_test)

fpr_test, tpr_test, thresholds_test = roc_curve(y_test, pred_test)
roc_auc_test = auc(fpr_test, tpr_test)

# Print test set evaluation metrics
print('Accuracy: {:.2f}'.format(accuracy_score(y_test, pred_test)))
print('Precision: {:.2f}'.format(precision_score(y_test, pred_test)))
print('Recall: {:.2f}'.format(recall_score(y_test, pred_test)))
print('F1 Score: {:.2f}'.format(f1_score(y_test, pred_test)))
print('AUC: {:.2f}'.format(roc_auc_test))

# Confusion matrix for the test set
cm_test = confusion_matrix(y_test, pred_test)
print('Confusion Matrix:')
print(cm_test)
print_confusion_matrix(y_test, pred_test)

# Plot ROC curve for the test set
plt.figure()
plt.plot(fpr_test, tpr_test, color='maroon', lw=2, label='AUC = %0.2f' % roc_auc_test)
plt.plot([0, 1], [0, 1], color='gray', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend(loc="lower right")
```

```
plt.show()

# Plot the confusion matrix for the test set
fig, ax = plt.subplots(figsize=(7.5, 7.5))
ax.matshow(cm_test, cmap=plt.cm.YlOrRd, alpha=0.5)
for i in range(cm_test.shape[0]):
    for j in range(cm_test.shape[1]):
        ax.text(x=i, y=j, s=cm_test[j, i], va='center', ha='center', size='xx-large')

# Set labels and title
plt.xlabel('Predictions', fontsize=16)
plt.ylabel('Actuals', fontsize=16)
plt.title('Confusion Matrix', fontsize=15)
plt.show()
```

Code 13: Code to test the best Predictive Model