African Institute for Mathematical Sciences
CAMEROON

DECISION TREE CLASSIFIER APPLIED ON COVID-19 DATASET

Edited by ***Group 5***

1.	Regine	PIYOU
2.	Michel	EMEL
3.	Joyce	Mukolwe
4.	Gabryella	Ratianarivo
5.	Sinenkhosi	Mamba
6.	Donald	Taboua

Lecturer: Prof O. Olawale Awe

Date of Submission: 27/12/2023

Contents

1 Introduction

A decision tree(DT) is defined in [1] to be a structure that includes a root node, branches and leaf nodes. A more technical and scientific definition of a DT is given in [2] which describes it as a flowchart-like tree structure where an internal node represents a feature or an attribute, the branch represents a decision rule and each leaf node represents the outcome. A graphical representation of a DT is shown in the figure below.

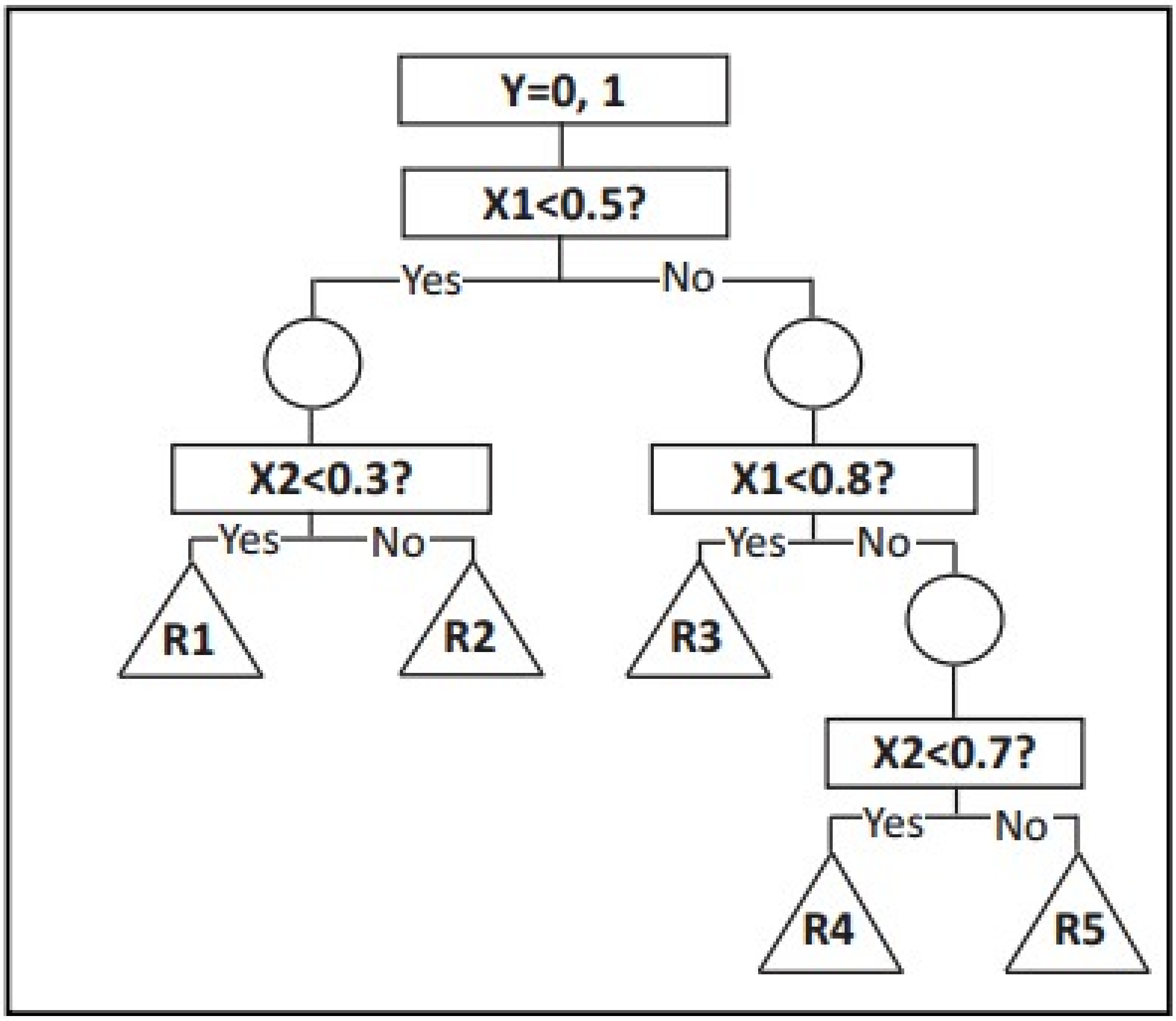


Figure 1: Yan-yan SONG, Ying LU, https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4466856/

It is important that we expand further on the important aspects of a decision tree and they are outlined below.

These aspect are:

- Nodes: These are namely the root node, internal node and leaf node. The root node is the top most part of the decision tree or simply the node that contains the condition to split the data into subsets, an internal node is any node in the decision tree that is not a leaf node, and leaf node is a terminal node in the decision tree
- Branches: These represent classification decision rules of the decision tree.

- Prunning: This is the process of removing nodes in a decision tree that provide less additional information.

The DT algorithm has numerous real life applications across many fields including education, medicine, engineering, biology to name just a few. As seen in [3], the decision tree can be used as a replacement for statistical procedures to find missing data in a class, for diagnosis of diseases in medicine,to improve search engines among many other applications. Due to its easy interpretability, ability to handle missing data, efficiency in handling missing values and robustness to outliers, it has emerged as a go to algorithm in a number of situations. As with anything in real life, as much as DT has its strengths, it also has its weaknesses. The most notable of them all is overfitting. Also, they have a high variance, are unstable in small changes in data, are biased towards dominant classes in a dataset and also are greedy in nature.

For the purposes of this work, we are to use the DT algorithm for the diagnosis of COVID-19. A dataset provided by the Mexican government was used that contains anonymized patient related information including pre-conditions. The objective of this study is to train a Decision Tree model and evaluate its performance in comparison to the logistic regression, which is considered the best basic classification model. To achieve this, we will employ GridSearchCV for hyperparameter tuning and utilize over-sampling techniques to address class imbalance. By doing so, we aim to determine the optimal performance and resampling approach for enhancing classification accuracy using the Decision Tree Classifier.

2 Literature review

The work [4] presents a comparative analysis of decision trees in the assessment of biomedical data and the strengths and weaknesses of each in dealing with different types of data are assessed. The decision tree algorithms studied for this work are Random Tree, REPTree, DecisionStump, SimpleCART, LMT and Regression Trees. These algorthms were found to be very competent in dealing with biomedical data and they achieved quite impressive results in the classification process.

Aksu et al in [5] give a study that compares decision trees produced by data mining algorithms that have been applied in a variety of fields recently based on several criteria. Through the use of 12 independent variables from the PISA 2015 student survey, similar and dissimilar elements of the decision trees derived by different methodologies for identifying the students as successful or failed in terms of science literacy were highlighted in the study. An open-source Java program called Weka was used to evaluate data from 5895 pupils in the 15+ age range who were gathered throughout Turkey. The investigation revealed that the most effective algorithms in terms of proper classification rate were, Logistic Model, Hoeffding Tree, J.48, REPTree and Random Tree.

In the work [6], we see that it is more challenging to apply data mining techniques to geographical data than to classical data. Spatial data mining algorithms need to handle the representation of data as a stack of thematic layers and take into account not only the item of interest but also its neighbors connected by implicit spatial links in order to extract meaningful patterns. A useful decision support tool for geographical data analysis is the application of decision trees for categorization along with visualization tools. This paper's goal is to present and assess a different spatial classification algorithm that complements the thematic-layered data organization. To do this, the C4.5 decision tree algorithm is modified for use with spatial data to create S-C4.5, a spatial classification algorithm that was inspired by the SCART and spatial ID3 algorithms.

This work [7] examines the decision tree algorithm's particular use in business cost control. It stresses that prior to establishing each data mining task, the data mining requirements must be made clear in order to apply data mining technology. To make the data mining realization aim more focused, we can only decide what type of data to choose and what method to employ for data mining by making the needs for the process clear. From the experimental results of this work, it was concluded that the overall tree view is more streamlined to include the manufacturing cost for analysis. Additionally, based on the evaluation outcomes following the decision tree, it was observed that the accuracy of decision tree modeling using optimization methods is higher, reaching levels of up to 95.1%.

The work [8] presents a solution to the problem of shop owners with E-commerce websites having to sort and see the product and the number of orders for that product which becomes very difficult when the orders are varied and consist of several product categories. This work attempts to solve this problem by adding a product category monitoring system function to the e-commerce website database which will automatically record and read the quantity of transactions and other data by incorporating the DT algorithm concept. This system generates a report on the quantity of stock and the best selling number of goods.

The authors of the work [9] present a methodology to create a stock trading method by combining the filter rule and decision tree techniques. Investors frequently employ the filter rule, which is used to provide candidate trading locations. Next, a decision tree method C4.5 is applied to cluster and screen generated points. This work incorporates future knowledge into the criterion for clustering the trade points. The NASDAQ and Taiwan stock markets are cited for the justification of the suggested approach. The proposed trading approach outperforms both the filter rule and the previous method, according to the experimental findings.

Bahzad et al [10] present decision tress as one of the most popular approaches for representing classifiers in data classification. The topic of expanding a decision tree from existing data has been studied by numerous researchers from a variety of disciplines and backgrounds, including statistics, machine learning, and pattern recognition. Decision tree classifiers have been suggested for usage in many different disciplines, including medical disease analysis, text categorization, user smartphone classification, pictures, and many more. This work offers a thorough analysis of the decision trees. In addition, the paper's specifics, including the datasets, algorithms/approaches employed, and results attained are thoroughly assessed and described. Furthermore, every method examined was talked about in order to highlight the writers' topics and determine which classifiers were the most accurate.

A number of decision trees were developed for the classification of hyperspectral data that was collected from a number of corn fields with different tillage, residue and cropping practices in the work [11]. The results show that decision trees could typically distinguish between different residue levels (0.98 classification accuracy) and tillage procedures (0.89 classification accuracy), which are very impressive numbers.

It was brought to light in the work [12] that the C5.0 Decision Tree Algorithm is a quick and quite accurate method for classifying minerals using X-ray intensities from a SEM-EDS without any standardized analytical conditions.

The authors of [13] developed a decision tree model that assists in the prediction of areas that are at risk of ground subsidence in the region of Jeongseon in Korea. They used the CHAID algorithm, QUEST algorithm and also frequency ratio model. All these models achieve impressive performance results, with the CHAID T algorithm achieving 94.01%, the QUEST algorithm achieving 90.37% and the frequency ratio model achieving 86.70% accuracy.

We are undoubtedly at the pinnacle of the data driven decision making era where companies, organizations and governments are using machine learning and data analysis to make sound decisions and even go a step further to making future predictions based on the data. The work [14] proposes a trend analysis technology and a model that can be used for future predictions based on analysis and text mining technologies which are based on decision trees.

Svakumar et al in the work [15] propose an educational data mining model which is to be used to to predict whether a student will drop out or not during the course of studies. This predictions are based on relevant attributes from socio-demographic, academic and institutional data for undergraduates at a university in India. This decision tree model was based on the ID3 algorithm. This proposed model achieved 97.50% accuracy which is a very impressive score.

The paper [16] presents the numerous successful applications of decision trees across many different fields the likes of cyber-security, intrusion detection systems, user authentication in biometrics, stock market and information retrieval to name a few. This work takes a closer look into how decision trees have been used in these different to solve various problems.

This work [17] proposes an interpretable decision set framework that combines descriptive and predictive capabilities. It has been applied to various domains, including healthcare and credit scoring. For instance, in healthcare, the framework has been used to generate interpretable decision sets for predicting patient outcomes and providing insights into the factors influencing the predictions.

It is stated in the work [18] that CatBoost is a gradient boosting framework that handles categorical features effectively. It has found applications in various domains, including recommendation systems and fraud detection. For example, CatBoost has been used to improve the performance of recommendation engines in e-commerce platforms by considering categorical features such as item categories and user preferences.

This work [19] focuses on optimizing hyperparameters of decision trees for imbalanced datasets. It addresses the challenge of handling imbalanced classes in classification tasks. For example, the research has been applied to medical diagnosis, where decision trees are tuned to handle the imbalance between normal and rare disease cases, leading to improved diagnostic accuracy.

JURIKOM et at in [20] present the application of data mining techniques, specifically decision tree classification, in predicting credit customers with the potential to lend for Mega Auto Finance. The research aims to address the problem of identifying potential customers for credit without incurring high operational marketing costs. The chosen data mining technique is classification, specifically using the decision tree algorithm known as C4.5.The study utilizes installment data of Mega Auto Finance loan customers from July 2018, which is processed in Microsoft Excel format. The outcome of the research is an application that assists the Mega Auto Finance Funds Section in identifying credit marketing targets in the future

3 Methodology

3.1 Gini impurity

Since decision trees are sensitive to irrelevant or redundant features, the first step to developing this model is to choose the most relevant features for model training. This can be done by using methods like Boruta for performing the feature selection process.

The most important part after feature selection is the computation of the Gini impurity or Entropy. For the purposes of this work we will use the both the Gini impurity and Entropy method. The Gini impurity is simply a measure of the impurity or disorder in a dataset. In the context of a DT,it is used to evaluate how well a feature separates the data into classes. The Gini impurity is computed using 1 below.

$$\text{Gini impurity} = 1 - \sum_{i=1}^{c} p_i^2 \tag{1}$$

where c is the number of classes, and p_i is the probability of choosing a data point of class i in the node. From this computations.

After choosing the best split using gini impurity, the next step is recursively splitting the dataset into subsets at each node of the tree based on the selected features and split points. This is where the actual tree is developed. If it happens that the resultant tree is too big, then pruning methods are applied to reduce the size of the tree. Then, given a new instance, the decision tree traverses the tree from top to bottom to the leaf node and gives the prediction based on the class or the value associated with the reached leaf.

3.2 Entropy

Entropy on the other hand is defined as the measure of disorder in a set of data points. The main objective of the decision if the decision tree algorithm is to try and minimize the entropy, this in turn results in more pure child nodes after splitting.

Entropy is computed for each class using expression 2 below

$$\text{Entropy}(S) = -\sum_{i=1}^{c} p_i \cdot \log_2(p_i) \tag{2}$$

where c is the number of classes p_i is the probability of choosing a data point of class i in the set S.
After calculating the Entropy for each node, the information gain is then computed for each node still which is given by the formula.

The information gain formula is given by 3 below:

$$\text{Information Gain} = \text{Entropy}(S) - \sum_{i=1}^{k} \frac{|S_i|}{|S|} \cdot \text{Entropy}(S_i) \tag{3}$$

where

- Entropy(S) is the entropy of the parent node.
- k is the number of child nodes after the split.

- $|S_i|$ is the number of data points in the i-th child node.
- $|S|$ is the total number of data points in the parent node.

The split that results in the highest information gain among all candidate splits is considered to be the best split in the data set.

4 Empirical Evidence

4.1 Visualization if dataset before pre-processing

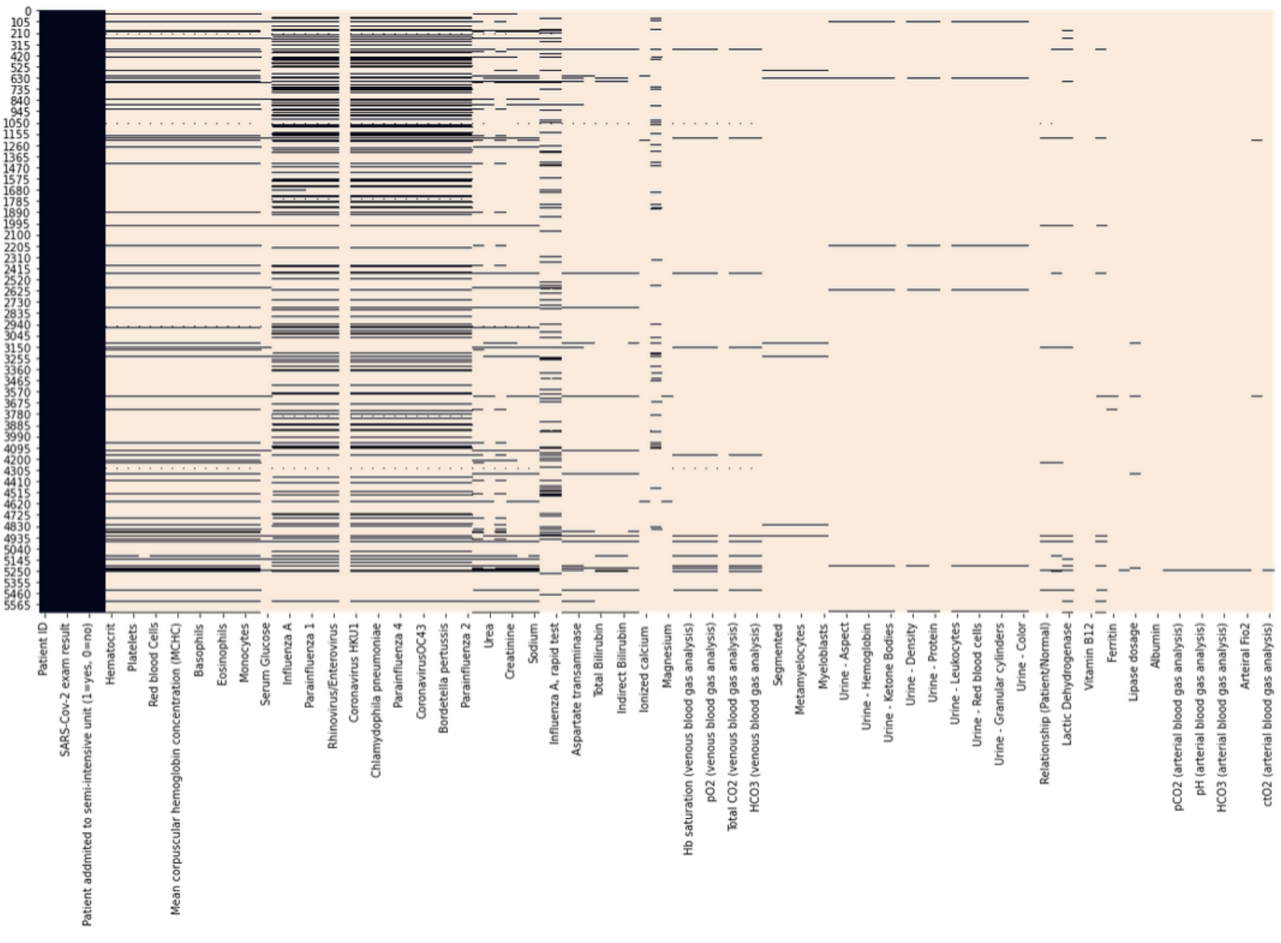


Figure 2: Visualization if dataset before pre-processing.

Figure 2 above gives a visual representation of the data before any data pre-preprocessing is done on the dataset. In this dataset, only 3 features did not have any missing values and these are "Patient ID", "SARS-Cov-2 exam result" and "Patient admitted to semi-intensive unit" and this can be seen in the figure by the thick black color corresponding to these columns. For the features which had some observations shaded black and some left blank, this depicted that there are missing values for that feature. Prior to any pre-processing on the dataset, it had a shape of (5644,111). Three pre-processing and data cleaning methods were employed on the dataset namely

- Handling missing values: This was done by dropping all columns which have proportion of missing values greater than 0.7 and observations which had missing values.

- Label encodage: Encoding negative = 0, positive = 1, non-detect = 0 and detect = 1 in the dataset.
- Features engineering: Creating a new column that groups all the columns of the diseases into 1 column.

After employing all these data pre-processing and cleaning techniques on the data, the shape the became (598, 17)

4.2 Correlation Plot

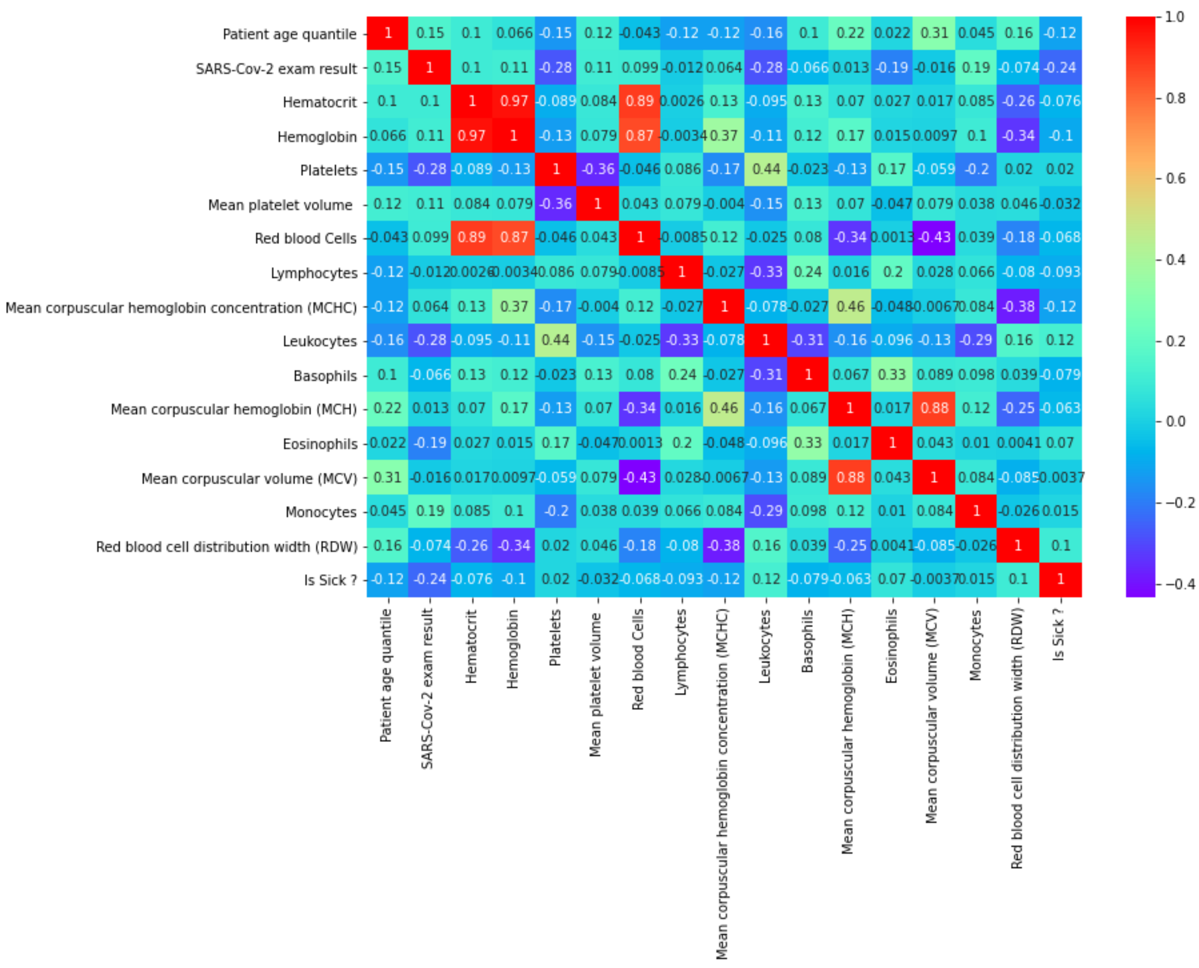


Figure 3: Correlation Plot.

We used a correlation plot to see if there are any correlated features in the dataset. A thick red color on the correlation plot signifies that there is strong positive correlation between those 2 features a thick blue color signifies that there is a strong negative correlation. The lighter the color suggests the correlation is not very strong between those features. From the correlation plot above, it can be seen that the features "Hematocrit" and "Hemoglobin" have a strongest positive correlation in all the features, "Hematocrit" and "Hemoglobin" are positively correlated with "Red blood cells", "Mean corpuscular hemoglobin (MCH)" is positively correlated with "Mean corpuscular volume (MCV)".

4.3 Distribution plots of features

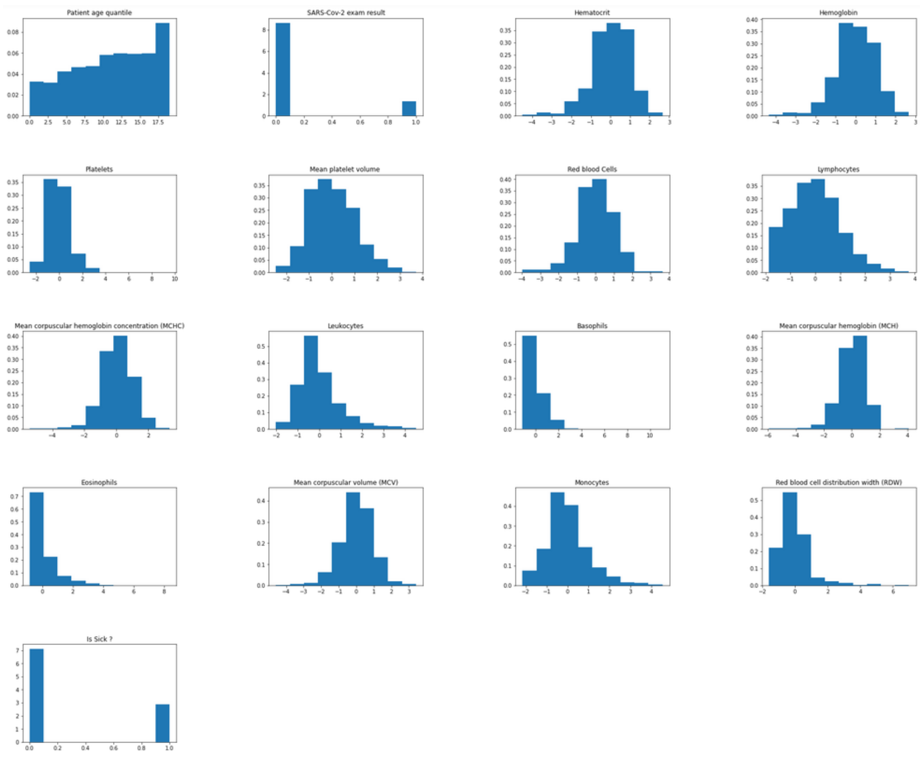


Figure 4: Distribution plots of features.

The plots in 4 show the distribution of data in each of the 17 features that remained after pre-processing stage and these said features are "Patient age quantile", "SARS-Cov-2 exam result", "Hematocrit", "Hemoglobin", "Platelets", "Mean platelet volume", "Red blood Cells", "Lymphocytes", "Mean corpuscular hemoglobin concentration (MCHC)", "Leukocytes", "Basophils", "Mean corpuscular hemoglobin (MCH)", "Eosinophils", "Mean corpuscular volume (MCV)", "Monocytes", "Red blood cell distribution width (RDW)", "Is Sick ?". The target variable is **"SARS-Cov-2 exam result"**.

4.4 Plot of target variable before resampling

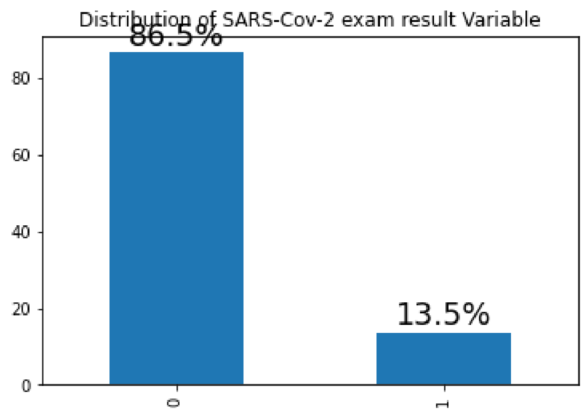


Figure 5: Distribution of target.

Above in 5 is a visualization of the target variable, "SARS-Cov-2 exam result". It clearly visible that we are working with imbalanced data, with "0" being the majority class(Not affected) and "1" being the minority class (affected).

4.5 Plot of target variable after resampling

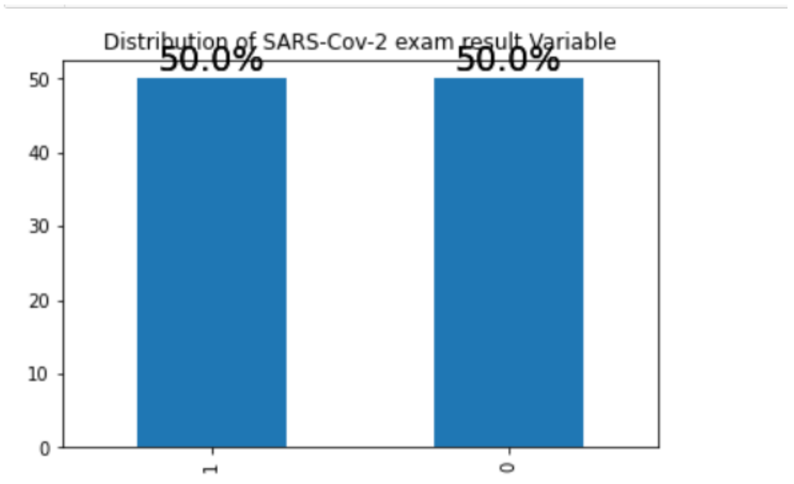


Figure 6: Distribution of target after resampling.

Then, figure 6 shows again the distribution of the target variable but now after resampling (Using SMOTE = Synthetic Minority Oversampling Technique) to make the target balances.

4.6 Perfomance evaluation for each model under different resampling methods

4.6.1 Grid Search Cross Validation

Before training the models , first Grid Search Cross Validation was performed to ascertain the best Hyper-parameters to use for building the models and the results from the Grid Search were summarised in table 1 as shown below. The table shows best parameters for The Decision Tree and Logistic Regression models independently.

Table 1: Best Parameters and Scores for Different Models

Model	Best Parameters	Best Score
DecisionTree	{'criterion':'entropy' ,'max_depth':5, 'min_samples_leaf':4, 'min_samples_split': 2}	0.7249
LogisticRegression	{'C': 100, 'penalty': 'l2'}	0.6756

Based on the parameters obtained from Grid Search cross validation above, different resampling methods were applied on the dataset to make it balanced. Then, both models were trained using data from each resampling method and the performance of both models under each resampling method was assessed and summarized in a table as shown in 2 below order by descending Balanced Accuracy score.

Table 2: Results of Different Resample Methods and Models

Resample Method	Model	Accuracy Score	Balanced Accuracy Score	F1 Score	Precision	Sensitivity	Specificity
SMOTE	DT	0.844444	0.866667	0.658537	0.519231	0.900000	0.833333
BSMOTE	LR	0.850000	0.830000	0.640000	0.533333	0.800000	0.860000
ROS	LR	0.833333	0.846667	0.634146	0.500000	0.866667	0.826667
ADASYN	LR	0.827778	0.830000	0.617284	0.490196	0.833333	0.826667
ADASYN	DT	0.838889	0.810000	0.613333	0.511111	0.766667	0.853333
SMOTE	LR	0.827778	0.816667	0.607595	0.489796	0.800000	0.833333
SMOTEENN	LR	0.783333	0.830000	0.580645	0.428571	0.900000	0.760000
SMOTEENN	DT	0.794444	0.810000	0.574713	0.438596	0.833333	0.786667
BSMOTE	DT	0.788889	0.806667	0.568182	0.431034	0.833333	0.780000
ROS	DT	0.827778	0.763333	0.563380	0.487805	0.666667	0.860000

From this results table, it can be seen that the DT model performs best when using the SMOTE resampling method and LR model performs best when using BSMOTE(BordelineSMOTE) resampling method.

4.7 Confusion Matrices

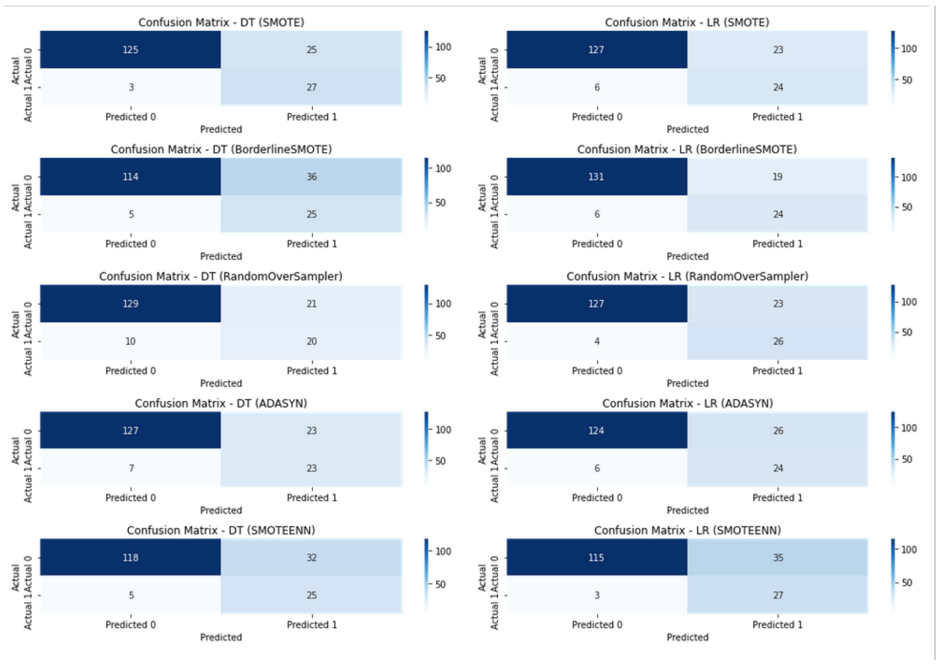


Figure 7: Confusion Matrices

Figure 7 above shows plots of confusion matrices for both Decision Tree and Logistic Regression models for each resampling method. In each confusion matrix.

- **True Negative (TN)**: Is represent by (0,0)
- **True Positive (TP)**: Is represented by (1,1)
- **False Positive (FP)**: Is represented by (0,1)
- **False Negtaive (FN)**: Is represented by (1,0)

In this description abscises is the actual and column is the predicted.

From the confusion matrices above, it is quite evident that the Decision Tree Model performs better for the prediction of the positive class than the negative as well as Logistic Regression model. Because the goal here is to classifier Positive very well than negative, we can say that using SMOTE resampling method this goal is achieve in the best way (just by see the true positive).

4.8 Learning Curve

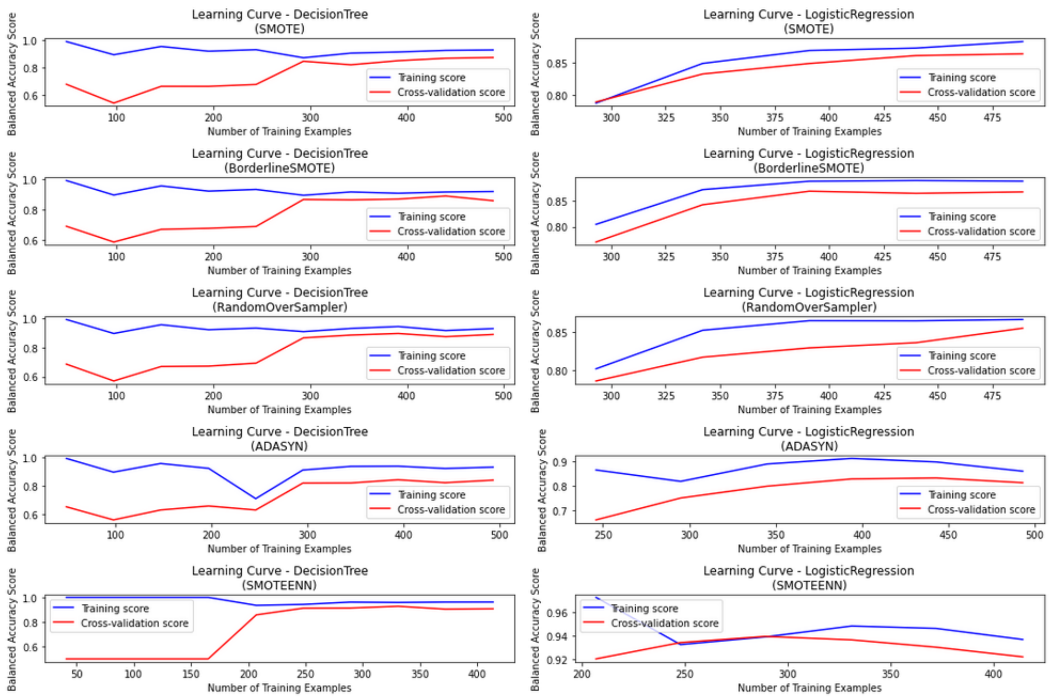


Figure 8: Decision Tree

- From the learning curves in 8, we can see that generally all the accuracy training scores a quite high. This is an indication that the model is learning very well from the underlying patterns in the dataset.
- Also, all learning curves except for the LR model using SMOTEENN resampling are increasing. This indicates that the model is learning very well and the during validation, the accuracy scores for models are improving.
- For most of the models, the learning curves increase and stabilize at high values. This is an indication that the models have learned the patterns in the data without overfitting

4.9 Decision Tree

The decision tree shown in 9 was developed based on the SMOTE resampling method since it was the best performing one for the DT model

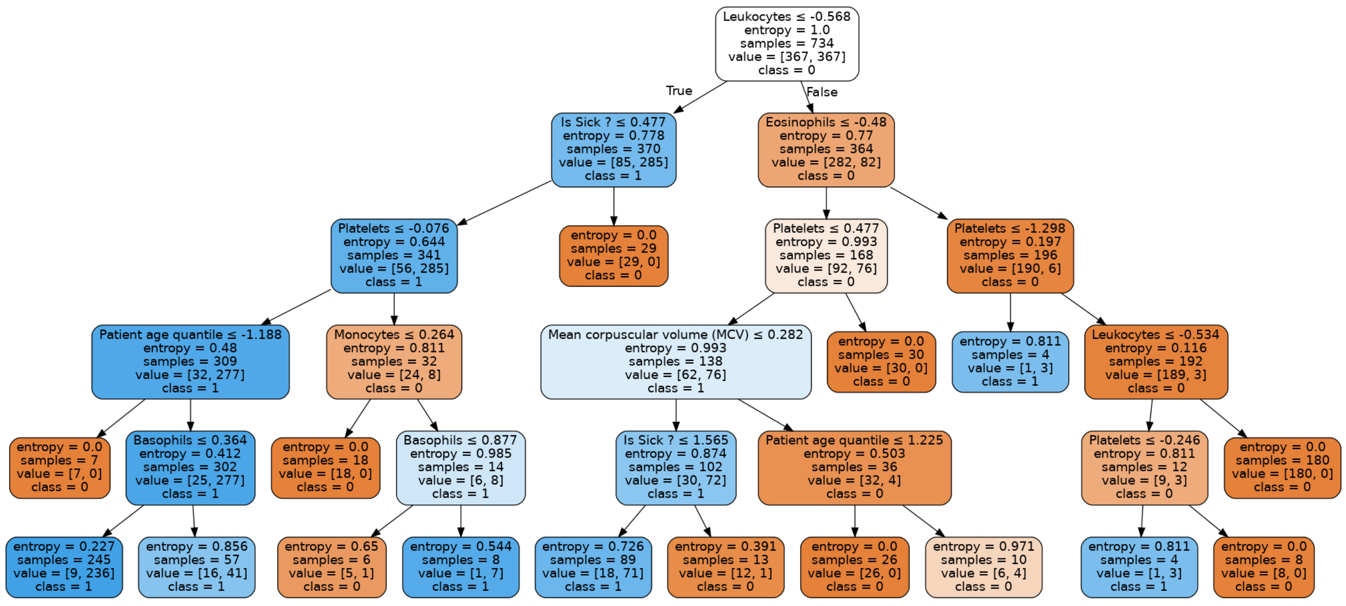


Figure 9: Decision Tree

Interpretation

- At the beginning we have root node which is the attribute with high Information Gain and the best for the splitting.(in this case is Leukocytes)

- Follow by the branches : the decision Rules (True or false).

- Next the decision Node with come after the Topmost node and the further branches.(The depth of decision = 4)

- At the end the leaf node: the final output nodes that do not split further and contain the prediction or outcome (class = 0 for not affected or 1 for affected).

The Max depth of our tree = 5 , and by define this we do the pre-pruning that avoid overfilling of the model.

As the above comparaison of DT with LR we can conclude that our Decision Tree Classifier Model Perform Well with SMOTE resampling Method and hyper-parameter : 'criterion':'entropy' ,'max_depth':5, 'min_samples_leaf':4, 'min_samples_split': 2.

5 Conclusion

Based on the results of this work, we can conclude that decision trees can be used for diagnosis of Covid-19 with a better performance compared to logistic regression when using SMOTE and ADASYN oversampling methods. As return time of test results is of utmost importance in the diagnosis of Covid-19, this work provides strong motivation for the adoption of Machine learning methods, particularly the use of decision trees to provide prompt and trustworthy results for Covid-19 diagnosis which could help save lives. The chosen model for this work was the DT using SMOTE resampling method as it achieved very high accuracy scores of 84.44%, with a sensitivity of 90% and these are quite impressive results.

6 Recommendations

From the results of this work, it is evident that the Decision Tree model performs best for predicting the positive class and this is true also for the logistic regression model. Due to this fact, further investigation was done whereby we find by grid-search the best hyper-parameters under each resampling method. From this, we saw that when the best hyper-parameters were found, its performance for predicting the negative class than the positive was greatly improved yet for logistic regression model the results remain the same. For future works, we recommend looking into developing an ensemble method of the two models like stacking to improve the performance results (Perform well in predicting the positive class as the negative class).

References

[1] N. Singh, Classification by decision tree induction (05 2023).

[2] A. Navlani, The decision tree algorithm (2023 url = https://www.datacamp.com/tutorial/decision-tree-classification-python, note = 2023: 12, 2023,).

[3] A. Navada, A. N. Ansari, S. Patil, B. A. Sonkamble, Overview of use of decision tree algorithms in machine learning, in: 2011 IEEE Control and System Graduate Research Colloquium, 2011, pp. 37–42. `doi:10.1109/ICSGRC.2011.5991826`.

[4] M. B. M. A. R. Fahima Hajjej, Manal Abdullah Alohali, A comparison of decision tree algorithms in the assessment of biomedical data 6 (2022) 1–9. `doi:10.1155/2022/9449497`.

[5] N. Aksu, GÃ¶khan; Dogan, Comparison of decision trees used in data mining 9 (2019) 27.

[6] B. H. i. B. F. Oujdi, S., Decision tree algorithm for spatial data, alternatives and performances 27 (2019) 27. `doi:10.20532/cit.2019.1004651`.

[7] W. H. X. X. X. W. Hong, Sen, Early warning of enterprise financial risk based on decision tree algorithm (07 2022). `doi:10.1155/2022/9182099`.

[8] H. Parlindungan Supriadi, Implementation decision tree algorithm for ecommerce website 24 (2020).

[9] M.-C. Wu, S.-Y. Lin, C.-H. Lin, An effective application of decision tree to stock trading, Expert Systems with Applications 31 (2) (2006) 270–274. `doi:https://doi.org/10.1016/j.eswa.2005.09.026`.
URL `https://www.sciencedirect.com/science/article/pii/S0957417405002137`

[10] B. Charbuty, A. Abdulazeez, Classification based on decision tree algorithm for machine learning, Journal of Applied Science and Technology Trends 2 (01) (2021) 20 – 28. `doi:10.38094/jastt20165`.
URL `https://www.jastt.org/index.php/jasttpath/article/view/65`

[11] C.-C. Yang, S. O. Prasher, P. Enright, C. Madramootoo, M. Burgess, P. K. Goel, I. Callum, Application of decision tree technology for image classification using remote sensing data, Agricultural Systems 76 (3) (2003) 1101–1117. `doi:https://doi.org/10.1016/S0308-521X(02)00051-3`.
URL `https://www.sciencedirect.com/science/article/pii/S0308521X02000513`

[12] E. Akkas, L. Akin, H., H. Artuner, Application of decision tree algorithm for classification and identification of natural minerals using sem eds, Computers and Geosciences 80 (2) (2015) 38–48. `doi:https://doi.org/10.1016/j.eswa.2005.09.026`.
URL `https://www.sciencedirect.com/science/article/pii/S0098300415000722`

[13] S. Lee, I. Park, Application of decision tree model for the ground subsidence hazard mapping near abandoned underground coal mines, Journal of environmental management 127 (2013) 166–176.

[14] J. Kim, M. Hwang, D.-H. Jeong, H. Jung, Technology trends analysis and forecasting application based on decision tree and statistical feature analysis, Expert Systems with Applications 39 (16) (2012) 12618–12625.

[15] S. Sivakumar, S. Venkataraman, R. Selvaraj, Predictive modeling of student dropout indicators in educational data mining using improved decision tree, Indian Journal of Science and Technology 9 (4) (2016) 1–5.

[16] K. Sachdeva, M. Hanmandlu, A. Kumar, Real life applications of fuzzy decision tree, International Journal of Computer Applications 42 (10) (2012) 24–28.

[17] H. Lakkaraju, S. H. Bach, J. Leskovec, Interpretable decision sets: A joint framework for description and prediction, in: Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining, 2016, pp. 1675–1684.

[18] A. V. Dorogush, V. Ershov, A. Gulin, Catboost: gradient boosting with categorical features support, arXiv preprint arXiv:1810.11363 (2018).

[19] R. Mantovani, T. Horvath, R. Cerri, J. Vanschoren, A. de Carvalho, Hyper-parameter tuning of a decision tree induction algorithm, 2016. doi:10.1109/BRACIS.2016.018.

[20] V. Srihith Indla, P. Lakshmi, A. Donald, T. Aditya, T. A. Srinivas, G. Thippanna, A forest of possibilities: Decision trees and beyond 6 (2023) 1–9. doi:10.5281/zenodo.8372196.

7 Appendix

Group_5_ML

December 27, 2023

[33]:
```
## Import useful the pachages
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier# use by boruta and RF
from sklearn.metrics import accuracy_score, balanced_accuracy_score
from sklearn.metrics import  recall_score, precision_score, f1_score,␣
↪matthews_corrcoef # the metrics
from imblearn.over_sampling import RandomOverSampler # for oversampling
from sklearn.metrics import roc_curve, roc_auc_score, confusion_matrix, ␣
↪classification_report
from sklearn.metrics import ConfusionMatrixDisplay ,make_scorer# The␣
↪additionnal metric ROC and fourfold plots
from sklearn.metrics import roc_curve, roc_auc_score
from sklearn.model_selection import GridSearchCV, KFold
from sklearn.model_selection import train_test_split
from imblearn.over_sampling import SMOTE, ADASYN, BorderlineSMOTE,␣
↪RandomOverSampler
from imblearn.combine import SMOTEENN
from imblearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.datasets import make_classification
from boruta import BorutaPy
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.tree import export_graphviz
from six import StringIO
from IPython.display import Image
import pydotplus
from sklearn.model_selection import learning_curve
from sklearn.impute import KNNImputer
```

0.1 Import DataSet and Preprocessing

[34]:
```
#Import Dataset
data = pd.read_excel('dataset.xlsx')
# show the 5 first row
data.head()
```

[34]:
```
         Patient ID  Patient age quantile SARS-Cov-2 exam result  \
0  44477f75e8169d2                    13               negative
1  126e9dd13932f68                    17               negative
2  a46b4402a0e5696                     8               negative
3  f7d619a94f97c45                     5               negative
4  d9e41465789c2b5                    15               negative

   Patient addmited to regular ward (1=yes, 0=no)  \
0                                               0
1                                               0
2                                               0
3                                               0
4                                               0

   Patient addmited to semi-intensive unit (1=yes, 0=no)  \
0                                                  0
1                                                  0
2                                                  0
3                                                  0
4                                                  0

   Patient addmited to intensive care unit (1=yes, 0=no)  Hematocrit  \
0                                                  0            NaN
1                                                  0       0.236515
2                                                  0            NaN
3                                                  0            NaN
4                                                  0            NaN

   Hemoglobin  Platelets  Mean platelet volume  …  \
0         NaN        NaN                   NaN  …
1    -0.02234  -0.517413              0.010677  …
2         NaN        NaN                   NaN  …
3         NaN        NaN                   NaN  …
4         NaN        NaN                   NaN  …

   Hb saturation (arterial blood gases)  pCO2 (arterial blood gas analysis)  \
0                                   NaN                                 NaN
1                                   NaN                                 NaN
2                                   NaN                                 NaN
3                                   NaN                                 NaN
```

```
4                                  NaN                                    NaN

   Base excess (arterial blood gas analysis)  \
0                                        NaN
1                                        NaN
2                                        NaN
3                                        NaN
4                                        NaN

   pH (arterial blood gas analysis)  Total CO2 (arterial blood gas analysis)  \
0                               NaN                                       NaN
1                               NaN                                       NaN
2                               NaN                                       NaN
3                               NaN                                       NaN
4                               NaN                                       NaN

   HCO3 (arterial blood gas analysis)  pO2 (arterial blood gas analysis)  \
0                                 NaN                                NaN
1                                 NaN                                NaN
2                                 NaN                                NaN
3                                 NaN                                NaN
4                                 NaN                                NaN

   Arteiral Fio2  Phosphor  ctO2 (arterial blood gas analysis)
0            NaN       NaN                                 NaN
1            NaN       NaN                                 NaN
2            NaN       NaN                                 NaN
3            NaN       NaN                                 NaN
4            NaN       NaN                                 NaN

[5 rows x 111 columns]
```

[35]:
```
# display the columns
data.columns.to_list()
```

[35]:
```
['Patient ID',
 'Patient age quantile',
 'SARS-Cov-2 exam result',
 'Patient addmited to regular ward (1=yes, 0=no)',
 'Patient addmited to semi-intensive unit (1=yes, 0=no)',
 'Patient addmited to intensive care unit (1=yes, 0=no)',
 'Hematocrit',
 'Hemoglobin',
 'Platelets',
 'Mean platelet volume ',
 'Red blood Cells',
 'Lymphocytes',
 'Mean corpuscular hemoglobin concentration\xa0(MCHC)',
 'Leukocytes',
 'Basophils',
 'Mean corpuscular hemoglobin (MCH)',
 'Eosinophils',
 'Mean corpuscular volume (MCV)',
 'Monocytes',
 'Red blood cell distribution width (RDW)',
 'Serum Glucose',
 'Respiratory Syncytial Virus',
 'Influenza A',
 'Influenza B',
 'Parainfluenza 1',
 'CoronavirusNL63',
 'Rhinovirus/Enterovirus',
 'Mycoplasma pneumoniae',
 'Coronavirus HKU1',
 'Parainfluenza 3',
 'Chlamydophila pneumoniae',
 'Adenovirus',
 'Parainfluenza 4',
 'Coronavirus229E',
 'CoronavirusOC43',
 'Inf A H1N1 2009',
 'Bordetella pertussis',
 'Metapneumovirus',
 'Parainfluenza 2',
 'Neutrophils',
 'Urea',
 'Proteina C reativa mg/dL',
 'Creatinine',
 'Potassium',
 'Sodium',
 'Influenza B, rapid test',
 'Influenza A, rapid test',
 'Alanine transaminase',
 'Aspartate transaminase',
 'Gamma-glutamyltransferase\xa0',
 'Total Bilirubin',
 'Direct Bilirubin',
 'Indirect Bilirubin',
 'Alkaline phosphatase',
 'Ionized calcium\xa0',
 'Strepto A',
 'Magnesium',
 'pCO2 (venous blood gas analysis)',
 'Hb saturation (venous blood gas analysis)',
 'Base excess (venous blood gas analysis)',
 'pO2 (venous blood gas analysis)',
 'Fio2 (venous blood gas analysis)',
 'Total CO2 (venous blood gas analysis)',
 'pH (venous blood gas analysis)',
 'HCO3 (venous blood gas analysis)',
 'Rods #',
 'Segmented',
 'Promyelocytes',
 'Metamyelocytes',
 'Myelocytes',
 'Myeloblasts',
 'Urine - Esterase',
 'Urine - Aspect',
 'Urine - pH',
 'Urine - Hemoglobin',
 'Urine - Bile pigments',
 'Urine - Ketone Bodies',
 'Urine - Nitrite',
 'Urine - Density',
 'Urine - Urobilinogen',
 'Urine - Protein',
 'Urine - Sugar',
 'Urine - Leukocytes',
 'Urine - Crystals',
 'Urine - Red blood cells',
 'Urine - Hyaline cylinders',
 'Urine - Granular cylinders',
 'Urine - Yeasts',
 'Urine - Color',
 'Partial thromboplastin time\xa0(PTT)\xa0',
 'Relationship (Patient/Normal)',
 'International normalized ratio (INR)',
 'Lactic Dehydrogenase',
 'Prothrombin time (PT), Activity',
 'Vitamin B12',
 'Creatine phosphokinase\xa0(CPK)\xa0',
 'Ferritin',
 'Arterial Lactic Acid',
 'Lipase dosage',
 'D-Dimer',
 'Albumin',
 'Hb saturation (arterial blood gases)',
 'pCO2 (arterial blood gas analysis)',
 'Base excess (arterial blood gas analysis)',
 'pH (arterial blood gas analysis)',
 'Total CO2 (arterial blood gas analysis)',
 'HCO3 (arterial blood gas analysis)',
 'pO2 (arterial blood gas analysis)',
 'Arteiral Fio2',
 'Phosphor',
 'ctO2 (arterial blood gas analysis)']
```

```
[36]: # Save a copy of the Dataset before preprocessing
      df = data.copy()
      df.head()
```

```
[36]:          Patient ID  Patient age quantile SARS-Cov-2 exam result  \
      0  44477f75e8169d2                    13               negative
      1  126e9dd13932f68                    17               negative
      2  a46b4402a0e5696                     8               negative
      3  f7d619a94f97c45                     5               negative
      4  d9e41465789c2b5                    15               negative

         Patient addmited to regular ward (1=yes, 0=no)  \
      0                                               0
      1                                               0
      2                                               0
      3                                               0
      4                                               0

         Patient addmited to semi-intensive unit (1=yes, 0=no)  \
      0                                                  0
      1                                                  0
      2                                                  0
      3                                                  0
      4                                                  0

         Patient addmited to intensive care unit (1=yes, 0=no)  Hematocrit  \
      0                                                  0          NaN
      1                                                  0     0.236515
      2                                                  0          NaN
      3                                                  0          NaN
      4                                                  0          NaN

         Hemoglobin  Platelets  Mean platelet volume  …  \
      0         NaN        NaN                   NaN  …
      1    -0.02234  -0.517413              0.010677  …
      2         NaN        NaN                   NaN  …
      3         NaN        NaN                   NaN  …
      4         NaN        NaN                   NaN  …

         Hb saturation (arterial blood gases)  pCO2 (arterial blood gas analysis)  \
      0                                   NaN                                 NaN
```

```
1                                  NaN                                  NaN
2                                  NaN                                  NaN
3                                  NaN                                  NaN
4                                  NaN                                  NaN

   Base excess (arterial blood gas analysis)  \
0                                        NaN
1                                        NaN
2                                        NaN
3                                        NaN
4                                        NaN

   pH (arterial blood gas analysis)  Total CO2 (arterial blood gas analysis)  \
0                               NaN                                      NaN
1                               NaN                                      NaN
2                               NaN                                      NaN
3                               NaN                                      NaN
4                               NaN                                      NaN

   HCO3 (arterial blood gas analysis)  pO2 (arterial blood gas analysis)  \
0                                 NaN                                NaN
1                                 NaN                                NaN
2                                 NaN                                NaN
3                                 NaN                                NaN
4                                 NaN                                NaN

   Arteiral Fio2  Phosphor  ctO2 (arterial blood gas analysis)
0            NaN       NaN                                 NaN
1            NaN       NaN                                 NaN
2            NaN       NaN                                 NaN
3            NaN       NaN                                 NaN
4            NaN       NaN                                 NaN

[5 rows x 111 columns]
```

[37]:
```
# display the shape of the data
data.shape
```

[37]:
```
(5644, 111)
```

[38]:
```
key_columns = ['Patient age quantile', 'SARS-Cov-2 exam result']
```

[39]:
```
missing_rate = df.isna().sum()/df.shape[0]
```

[40]:
```
# By see the distribution of the missing rate
#we notice that missing rate between 0.88 and 0.9 are for blood composite
# and between 0.75 and 0.8 is different viral
```

```
blood_columns = list(df.columns[(missing_rate < 0.9) & (missing_rate > 0.88)])
viral_columns = list(df.columns[(missing_rate < 0.80) & (missing_rate > 0.75)])
```

[41]:
```
# we just focus on the information in the virus(disease) ,
#the blood and  key colummns
df = df[key_columns + blood_columns + viral_columns]
```

[42]:
```
df.shape
```

[42]:
```
(5644, 33)
```

[43]:
```
df[viral_columns]
```

[43]:
```
     Respiratory Syncytial Virus   Influenza A   Influenza B Parainfluenza 1  \
0                            NaN           NaN           NaN             NaN
1                   not_detected  not_detected  not_detected    not_detected
2                            NaN           NaN           NaN             NaN
3                            NaN           NaN           NaN             NaN
4                   not_detected  not_detected  not_detected    not_detected
…                              …             …             …              …
5639                         NaN           NaN           NaN             NaN
5640                         NaN           NaN           NaN             NaN
5641                         NaN           NaN           NaN             NaN
5642                         NaN           NaN           NaN             NaN
5643                         NaN           NaN           NaN             NaN

     CoronavirusNL63 Rhinovirus/Enterovirus Coronavirus HKU1 Parainfluenza 3  \
0                NaN                    NaN              NaN             NaN
1       not_detected               detected     not_detected    not_detected
2                NaN                    NaN              NaN             NaN
3                NaN                    NaN              NaN             NaN
4       not_detected               detected     not_detected    not_detected
…                  …                      …                …               …
5639             NaN                    NaN              NaN             NaN
5640             NaN                    NaN              NaN             NaN
5641             NaN                    NaN              NaN             NaN
5642             NaN                    NaN              NaN             NaN
5643             NaN                    NaN              NaN             NaN

     Chlamydophila pneumoniae    Adenovirus Parainfluenza 4 Coronavirus229E  \
0                         NaN           NaN             NaN             NaN
1                not_detected  not_detected    not_detected    not_detected
2                         NaN           NaN             NaN             NaN
3                         NaN           NaN             NaN             NaN
4                not_detected  not_detected    not_detected    not_detected
…                           …             …               …               …
5639                      NaN           NaN             NaN             NaN
```

```
5640                         NaN           NaN              NaN              NaN
5641                         NaN           NaN              NaN              NaN
5642                         NaN           NaN              NaN              NaN
5643                         NaN           NaN              NaN              NaN

     CoronavirusOC43 Inf A H1N1 2009 Bordetella pertussis Metapneumovirus  \
0                NaN             NaN                  NaN             NaN
1       not_detected    not_detected         not_detected    not_detected
2                NaN             NaN                  NaN             NaN
3                NaN             NaN                  NaN             NaN
4       not_detected    not_detected         not_detected    not_detected
…                  …               …                    …               …
5639             NaN             NaN                  NaN             NaN
5640             NaN             NaN                  NaN             NaN
5641             NaN             NaN                  NaN             NaN
5642             NaN             NaN                  NaN             NaN
5643             NaN             NaN                  NaN             NaN

     Parainfluenza 2
0                NaN
1       not_detected
2                NaN
3                NaN
4       not_detected
…                  …
5639             NaN
5640             NaN
5641             NaN
5642             NaN
5643             NaN

[5644 rows x 17 columns]
```

[44]:
```python
def encodage(df):
    code = {'negative':0,
            'positive':1,
             'True': 1,
            'False':0,
            'not_detected':0,
            'detected':1}

    for col in df.select_dtypes('object').columns:
        df.loc[:,col] = df[col].map(code)

    return df
```

[45]:
```
def feature_engineering(df):
    df['Is Sick ?'] = df[viral_columns].sum(axis=1) >= 1
    df = df.drop(viral_columns, axis=1)
    return df
```

[46]:
```
def imputation(df):
    df = df.dropna(axis=0)
    return  df
```

[47]:
```
def preprocessing(df):



    df = encodage(df)
    df = feature_engineering(df)
    df = imputation(df)

    return df
```

[48]:
```
covid_data = preprocessing(df)
```

[49]:
```
covid_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 598 entries, 1 to 5643
Data columns (total 17 columns):
 #   Column                                            Non-Null Count  Dtype
---  ------                                            --------------  -----
 0   Patient age quantile                              598 non-null    int64
 1   SARS-Cov-2 exam result                            598 non-null    int64
 2   Hematocrit                                        598 non-null    float64
 3   Hemoglobin                                        598 non-null    float64
 4   Platelets                                         598 non-null    float64
 5   Mean platelet volume                              598 non-null    float64
 6   Red blood Cells                                   598 non-null    float64
 7   Lymphocytes                                       598 non-null    float64
 8   Mean corpuscular hemoglobin concentration (MCHC)  598 non-null    float64
 9   Leukocytes                                        598 non-null    float64
 10  Basophils                                         598 non-null    float64
 11  Mean corpuscular hemoglobin (MCH)                 598 non-null    float64
 12  Eosinophils                                       598 non-null    float64
 13  Mean corpuscular volume (MCV)                     598 non-null    float64
 14  Monocytes                                         598 non-null    float64
 15  Red blood cell distribution width (RDW)           598 non-null    float64
 16  Is Sick ?                                         598 non-null    bool
dtypes: bool(1), float64(14), int64(2)
memory usage: 80.0 KB
```

[50]:
```
covid_data['Is Sick ?'] = covid_data['Is Sick ?'].astype(int)
```

[51]:
```
covid_data.head()
```

[51]:
```
    Patient age quantile  SARS-Cov-2 exam result  Hematocrit  Hemoglobin  \
1                     17                       0    0.236515   -0.022340
8                      1                       0   -1.571682   -0.774212
15                     9                       0   -0.747693   -0.586244
18                    11                       0    0.991838    0.792188
22                     9                       0    0.190738   -0.147652

    Platelets  Mean platelet volume   Red blood Cells  Lymphocytes  \
1   -0.517413               0.010677         0.102004     0.318366
8    1.429667              -1.672222        -0.850035    -0.005738
15  -0.429480              -0.213711        -1.361315    -1.114514
18   0.072992              -0.550290         0.542763     0.045436
22  -0.668155               1.020415        -0.127191     0.002791

    Mean corpuscular hemoglobin concentration (MCHC)  Leukocytes  Basophils  \
1                                          -0.950790   -0.094610  -0.223767
8                                           3.331071    0.364550  -0.223767
15                                          0.542882   -0.884923   0.081693
18                                         -0.452899   -0.211488  -0.834685
22                                         -1.249524   -1.132592   0.387152

    Mean corpuscular hemoglobin (MCH)  Eosinophils  \
1                           -0.292269     1.482158
8                            0.178175     1.018625
15                           1.746323    -0.666950
18                           0.334989    -0.709090
22                          -0.083183    -0.709090

    Mean corpuscular volume (MCV)  Monocytes  \
1                        0.166192   0.357547
8                       -1.336024   0.068652
15                       1.668409   1.276759
18                       0.606842  -0.220244
22                       0.566783   2.012129

    Red blood cell distribution width (RDW)  Is Sick ?
1                                 -0.625073          1
8                                 -0.978899          0
15                                -1.067355          1
18                                 0.171035          1
22                                 0.613318          0
```

[52]:
```
covid_data.shape
```

[52]:
```
(598, 17)
```

[53]:
```
covid_data.to_csv('covid_data.csv', index=False)
```

[54]:
```
#importing the dataset
df = pd.read_csv("covid_data.csv")
df.head()
```

[54]:
```
   Patient age quantile  SARS-Cov-2 exam result  Hematocrit  Hemoglobin  \
0                    17                       0    0.236515   -0.022340
1                     1                       0   -1.571682   -0.774212
2                     9                       0   -0.747693   -0.586244
3                    11                       0    0.991838    0.792188
4                     9                       0    0.190738   -0.147652

   Platelets  Mean platelet volume   Red blood Cells  Lymphocytes  \
0  -0.517413              0.010677          0.102004     0.318366
1   1.429667             -1.672222         -0.850035    -0.005738
2  -0.429480             -0.213711         -1.361315    -1.114514
3   0.072992             -0.550290          0.542763     0.045436
4  -0.668155              1.020415         -0.127191     0.002791

   Mean corpuscular hemoglobin concentration (MCHC)  Leukocytes  Basophils  \
0                                         -0.950790   -0.094610  -0.223767
1                                          3.331071    0.364550  -0.223767
2                                          0.542882   -0.884923   0.081693
3                                         -0.452899   -0.211488  -0.834685
4                                         -1.249524   -1.132592   0.387152

   Mean corpuscular hemoglobin (MCH)  Eosinophils  \
0                          -0.292269     1.482158
1                           0.178175     1.018625
2                           1.746323    -0.666950
3                           0.334989    -0.709090
4                          -0.083183    -0.709090

   Mean corpuscular volume (MCV)  Monocytes  \
0                       0.166192   0.357547
1                      -1.336024   0.068652
2                       1.668409   1.276759
3                       0.606842  -0.220244
4                       0.566783   2.012129

   Red blood cell distribution width (RDW)  Is Sick ?
0                                -0.625073          1
1                                -0.978899          0
2                                -1.067355          1
3                                 0.171035          1
4                                 0.613318          0
```

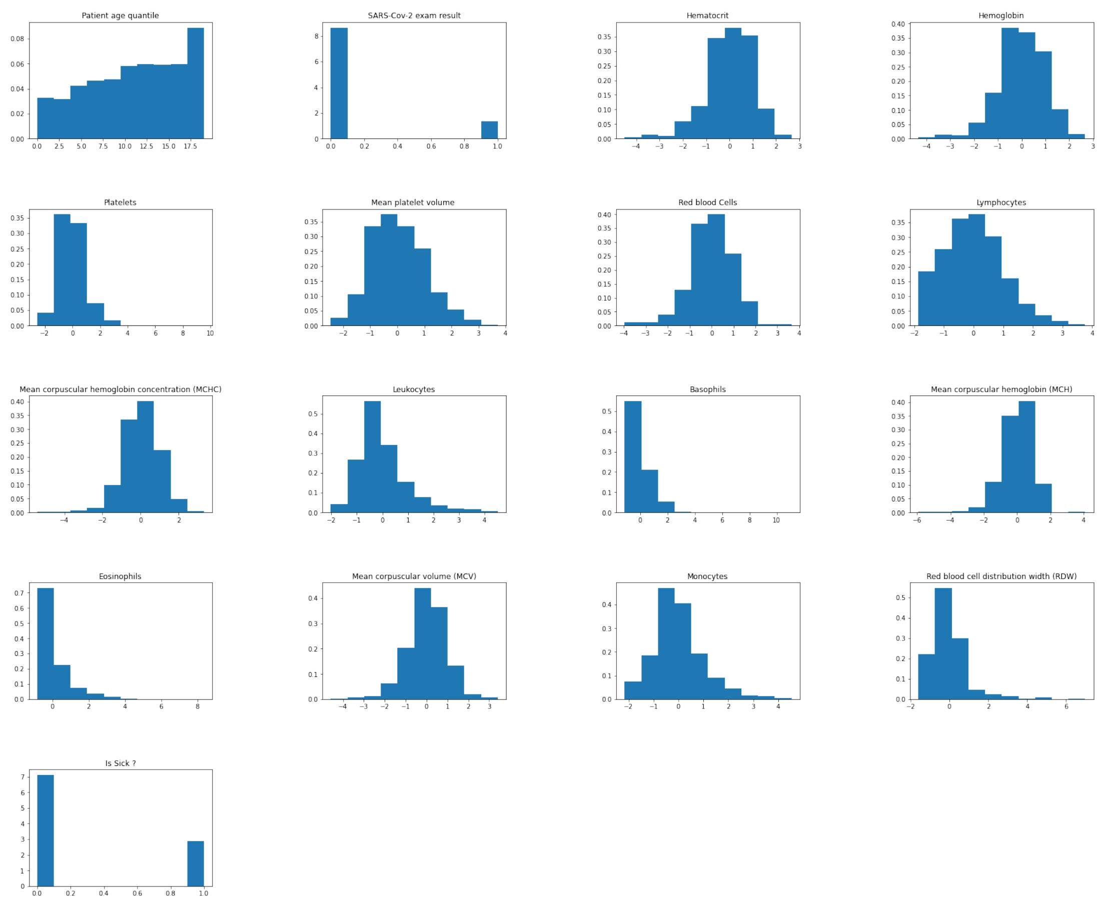
[55]:
```
df.shape
```

[55]:
```
(598, 17)
```

[56]:
```
df.columns
```

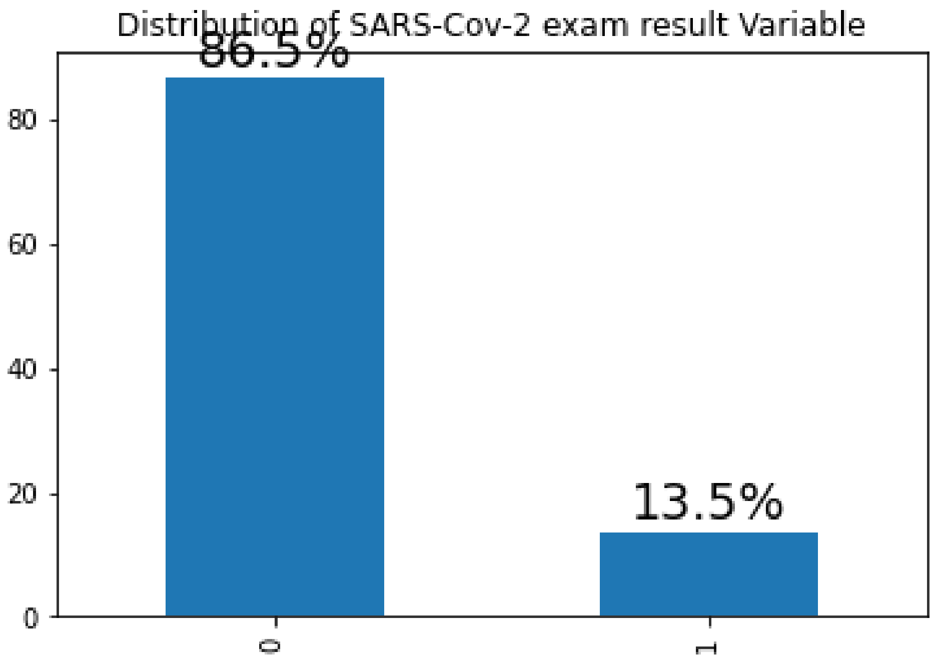
[56]:
```
Index(['Patient age quantile', 'SARS-Cov-2 exam result', 'Hematocrit',
       'Hemoglobin', 'Platelets', 'Mean platelet volume ', 'Red blood Cells',
       'Lymphocytes', 'Mean corpuscular hemoglobin concentration (MCHC)',
       'Leukocytes', 'Basophils', 'Mean corpuscular hemoglobin (MCH)',
       'Eosinophils', 'Mean corpuscular volume (MCV)', 'Monocytes',
       'Red blood cell distribution width (RDW)', 'Is Sick ?'],
      dtype='object')
```

[57]:
```
## Histogramm of the features
# Create the figure and subplots
fig, axs = plt.subplots(5, 4, figsize=(30,25))

# Flatten the axs array to make it easier to iterate through
axs = axs.flatten()

# Loop through each feature and plot a histogram in a separate subplot
for i, col in enumerate(df.columns):
    axs[i].hist(df[col], density = True, bins=10)
    axs[i].set_title(col)

# Hide any unused subplots
for i in range(len(df.columns), 5 * 4):
    axs[i].axis('off')

# Add a main title and adjust the spacing between subplots
#plt.suptitle('Histograms for Multiple Features')
plt.subplots_adjust(hspace=0.6, wspace=0.6)
#fig.suptitle("Histogram for the features", fontsize = 30)

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



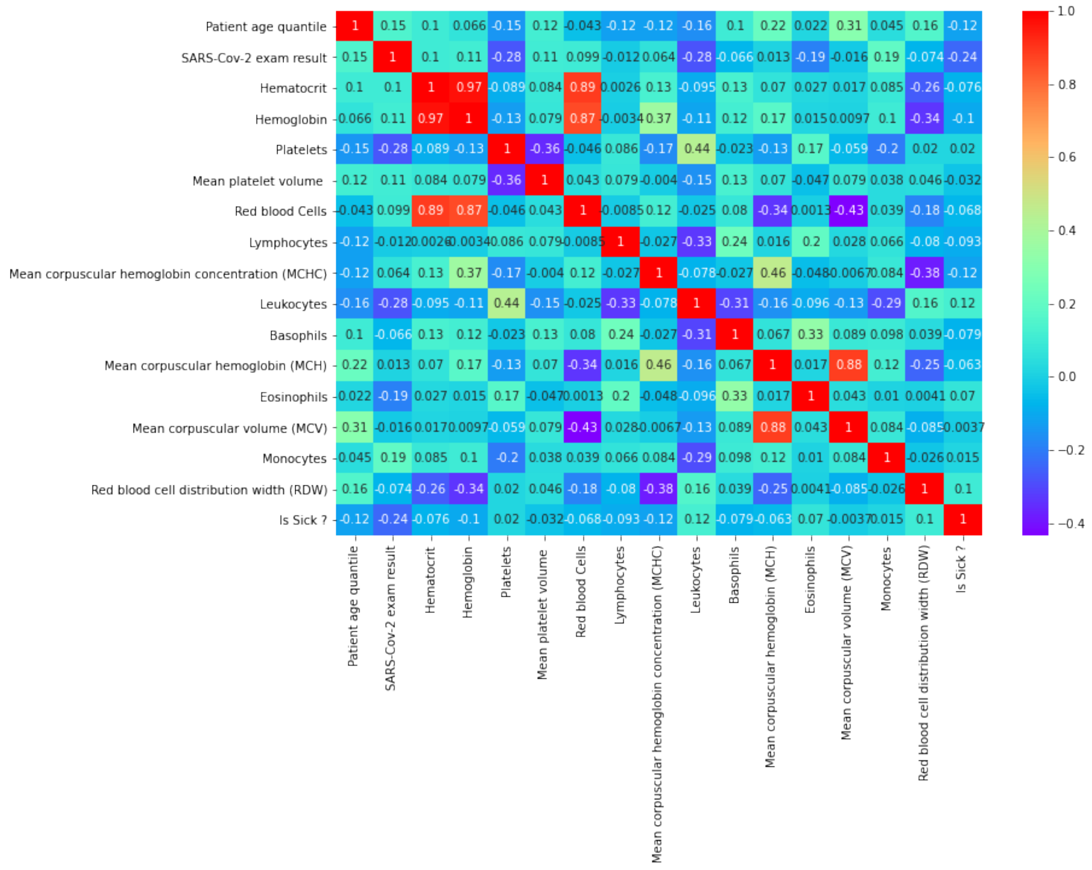
[58]:
```
#To check our target variable()

ax = df["SARS-Cov-2 exam result"].value_counts(normalize=True).mul(100).plot.
↪bar()
for p in ax.patches:
    y = p.get_height()
    x = p.get_x() + p.get_width() / 2

    # Label of bar height
    label = "{:.1f}%".format(y)

    # Annotate plot
    ax.annotate(label, xy=(x, y), xytext=(0, 5), textcoords="offset points",␣
↪ha="center", fontsize=19)
    # Add a title to the plot
ax.set_title('Distribution of SARS-Cov-2 exam result Variable')
plt.savefig('Imbalanced_data.png')
# Remove y axis
```

```
#ax.get_yaxis().set_visible(False)
```

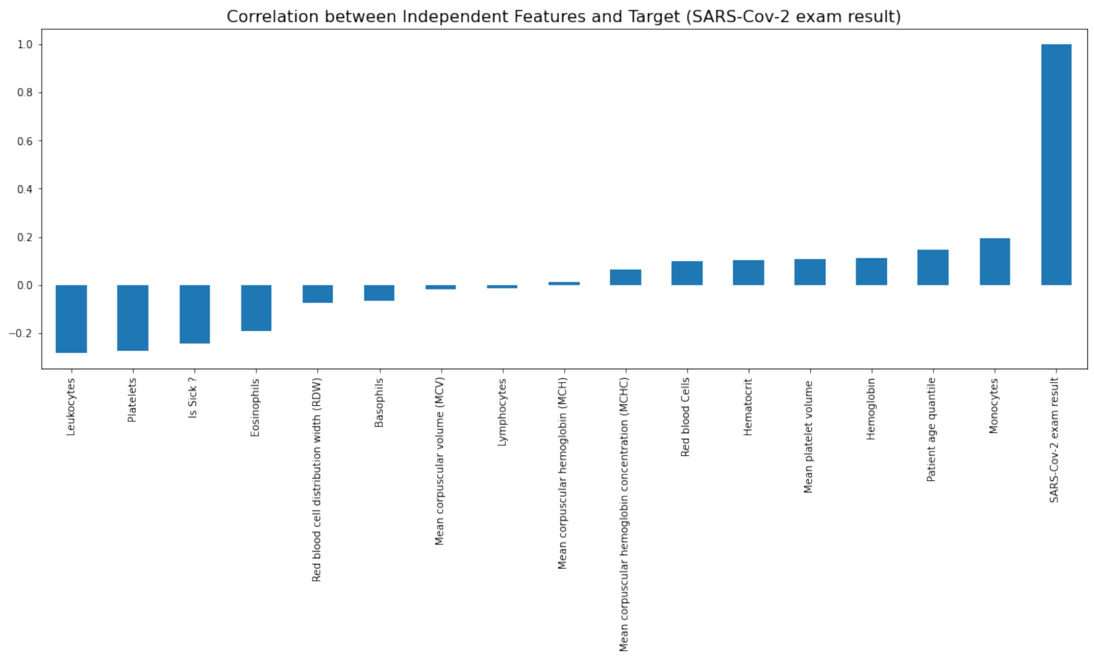


```
[59]: plt.figure(figsize=(12,8))
      sns.heatmap(df.corr(),annot=True,cmap="rainbow")
```

```
[59]: <AxesSubplot:>
```

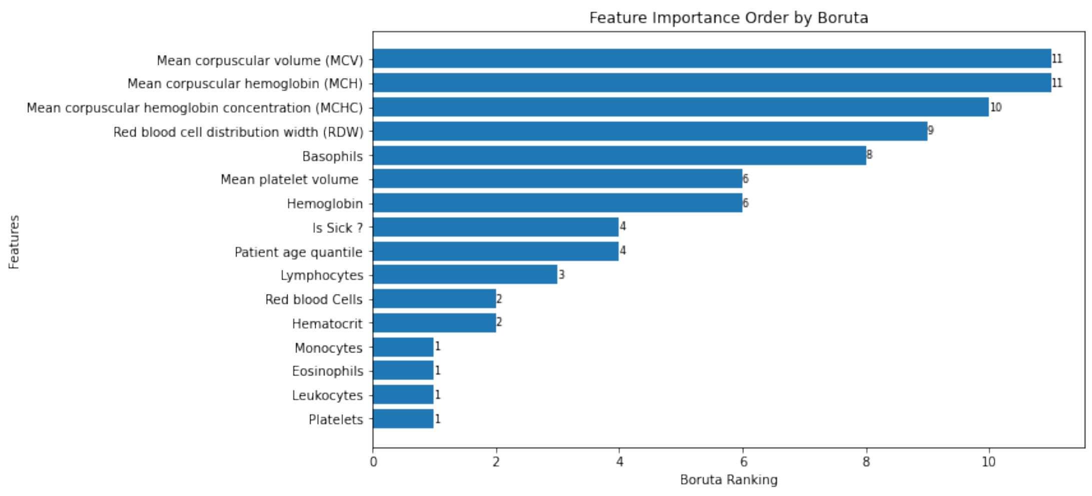


[60]:
```
#Let's find the correlation between the independent and the dependent feature␣
↪(obese)
df.corr()['SARS-Cov-2 exam result'].sort_values().plot(kind='bar', figsize=(18,␣
↪6))
# Add a title to the plot
plt.title('Correlation between Independent Features and Target (SARS-Cov-2 exam␣
↪result)', fontsize=16)

# Show plot
plt.show()
```



[61]:
```
X,y = df.drop('SARS-Cov-2 exam result',axis =1), df['SARS-Cov-2 exam result']
```

[62]:
```
# Initialize a Random Forest classifier
rf = RandomForestClassifier(n_estimators=100, random_state=42)

# Perform Boruta feature selection
boruta_selector = BorutaPy(rf, n_estimators='auto', verbose=0, random_state=42)
boruta_selector.fit(X.values, y.values)

# Get selected features and their rankings
feature_ranks = list(zip(X.columns, boruta_selector.ranking_))
feature_ranks.sort(key=lambda x: x[1])  # Sort by Boruta ranking

# Visualizing important features in sorted order with Boruta rankings as␣
 ↪annotations
import matplotlib.pyplot as plt
plt.figure(figsize=(10, 6))
features_sorted, ranks_sorted = zip(*feature_ranks)
bars = plt.barh(features_sorted, ranks_sorted)
plt.xlabel('Boruta Ranking')
plt.ylabel('Features')
plt.title('Feature Importance Order by Boruta')

# Adding Boruta rankings as annotations on the bars
for bar, rank in zip(bars, ranks_sorted):
    plt.text(bar.get_width(), bar.get_y() + bar.get_height()/2, f'{rank}',
             va='center', ha='left', fontsize=8)

plt.show()
plt.savefig('Features_Importance.png')
# Get selected features in descending order of importance
selected_features = [feature[0] for feature in feature_ranks]
print("Selected Features in Order of Importance:", selected_features)
```



```
Selected Features in Order of Importance: ['Platelets', 'Leukocytes',
'Eosinophils', 'Monocytes', 'Hematocrit', 'Red blood Cells', 'Lymphocytes',
'Patient age quantile', 'Is Sick ?', 'Hemoglobin', 'Mean platelet volume ',
'Basophils', 'Red blood cell distribution width (RDW)', 'Mean corpuscular
hemoglobin concentration\xa0(MCHC)', 'Mean corpuscular hemoglobin (MCH)', 'Mean
corpuscular volume (MCV)']

<Figure size 432x288 with 0 Axes>
```

```
[63]: X_train , X_test, y_train, y_test =␣
       ↪train_test_split(X,y,random_state=120,test_size = 0.3 )
      print('X_train shape = ', X_train.shape)
      print('y_train shape = ', y_train.shape)
      print('X_test shape =', X_test.shape)
      print('y_test shape =', y_test.shape)
```

```
X_train shape =  (418, 16)
y_train shape =  (418,)
X_test shape = (180, 16)
y_test shape = (180,)
```

```
[64]: scaler=StandardScaler()
      X_train_transformed = scaler.fit_transform(X_train)
      X_test_transformed  = scaler.transform(X_test)
```

0.1.1 GridSearchCV

```
[65]: # Define the parameter grids for each model
      param_grids = {
              'DecisionTree': {'criterion': ["gini", "entropy"],
                                "max_depth" : [None,5, 7,10, 15],
                                "min_samples_split": [2,7, 5,10] ,
                               "min_samples_leaf" : [1,2,4]},
              'LogisticRegression': {"penalty" : ['l1', 'l2', "elasticnet"],
                  'C': [0.001,0.01, 0.1, 1, 10, 100]}
          }

      # Create scorers for each metric
      scorers = {
          'Accuracy': make_scorer(accuracy_score),
          'Balanced Accuracy': make_scorer(balanced_accuracy_score),
          'Precision': make_scorer(precision_score, average='macro'),
          'Recall': make_scorer(recall_score, average='macro'),
          'f1': make_scorer(f1_score, average='macro')
      }

          # Initialize models
      models = {
              'DecisionTree': DecisionTreeClassifier(),
              'LogisticRegression': LogisticRegression()
          }

          #Perform GridSearchCV for each model
      for model_name, model in models.items():

              param_grid = param_grids[model_name]
              grid_search = GridSearchCV(estimator=model, param_grid=param_grid, cv=3,
                                         scoring=scorers, refit='Balanced Accuracy' )
              grid_search.fit(X_train_transformed, y_train)

              # Get the best parameters and the best score
              best_params = grid_search.best_params_
              best_score = grid_search.best_score_
              print(f"Model: {model_name}")
              print("Best Parameters:", best_params)
              print("Best Score:", best_score)
```

```
Model: DecisionTree
Best Parameters: {'criterion': 'entropy', 'max_depth': 5, 'min_samples_leaf': 4,
'min_samples_split': 2}
Best Score: 0.7249250626546767
Model: LogisticRegression
Best Parameters: {'C': 100, 'penalty': 'l2'}
Best Score: 0.6756088414307481
```

0.1.2 Resampling method Without any strategy using the best hyperparameter find during the GridSearchCV

```
[66]: resample_methods1 = [SMOTE(random_state=42),
                          BorderlineSMOTE(random_state=42),
                          RandomOverSampler(random_state=42),
                         ADASYN(random_state = 42),
                         SMOTEENN(random_state=42),
                        ]

      models = [('DecisionTree', DecisionTreeClassifier(criterion= 'entropy',
                                                        max_depth= 5,␣
       ↪min_samples_leaf= 4,
                                                        min_samples_split = 2)),
               ('LogisticRegression', LogisticRegression(C= 100, penalty= 'l2'))]

      results = pd.DataFrame(columns=["Resample Method", "Model",
                                      "Accuracy Score",
                                      "Balanced Accuracy Score", "F1 Score"])

      for resample_method in resample_methods1:
          model_results = pd.DataFrame(columns=["Model", "Accuracy Score", "Balanced␣
       ↪Accuracy Score", "F1 Score"])
          x_resampled, y_resampled = resample_method.
       ↪fit_resample(X_train_transformed, y_train)

          for clf_name, clf in tqdm(models):
              clf.fit(x_resampled, y_resampled)
              predictions = clf.predict(X_test_transformed)
              accuracy = accuracy_score(y_test, predictions)
              bal_accuracy = balanced_accuracy_score(y_test, predictions)
              f1score = f1_score(y_test, predictions)
              sensitivity = recall_score(y_test, predictions)
              specificity = recall_score(y_test, predictions, pos_label=0)
              precision = precision_score(y_test, predictions)

              new_row = {"Model": clf_name, "Accuracy Score": accuracy,
                         "Balanced Accuracy Score": bal_accuracy,
                         "F1 Score": f1score,
                        "sensitivity" : sensitivity,
                     "specificity":specificity,
                     "precision" : precision
                     }
              model_results = model_results.append(new_row, ignore_index=True)

          model_results["Resample Method"] = resample_method.__class__.__name__
          results = pd.concat([results, model_results])

      results.sort_values(by="F1 Score", ascending=False, inplace=True)
      results.to_excel("resampling_results.xlsx")
      results
```

```
100%|      | 2/2 [00:00<00:00,  9.34it/s]
100%|      | 2/2 [00:00<00:00, 18.36it/s]
100%|      | 2/2 [00:00<00:00,  9.39it/s]
100%|      | 2/2 [00:00<00:00,  8.74it/s]
100%|      | 2/2 [00:00<00:00, 15.98it/s]
```

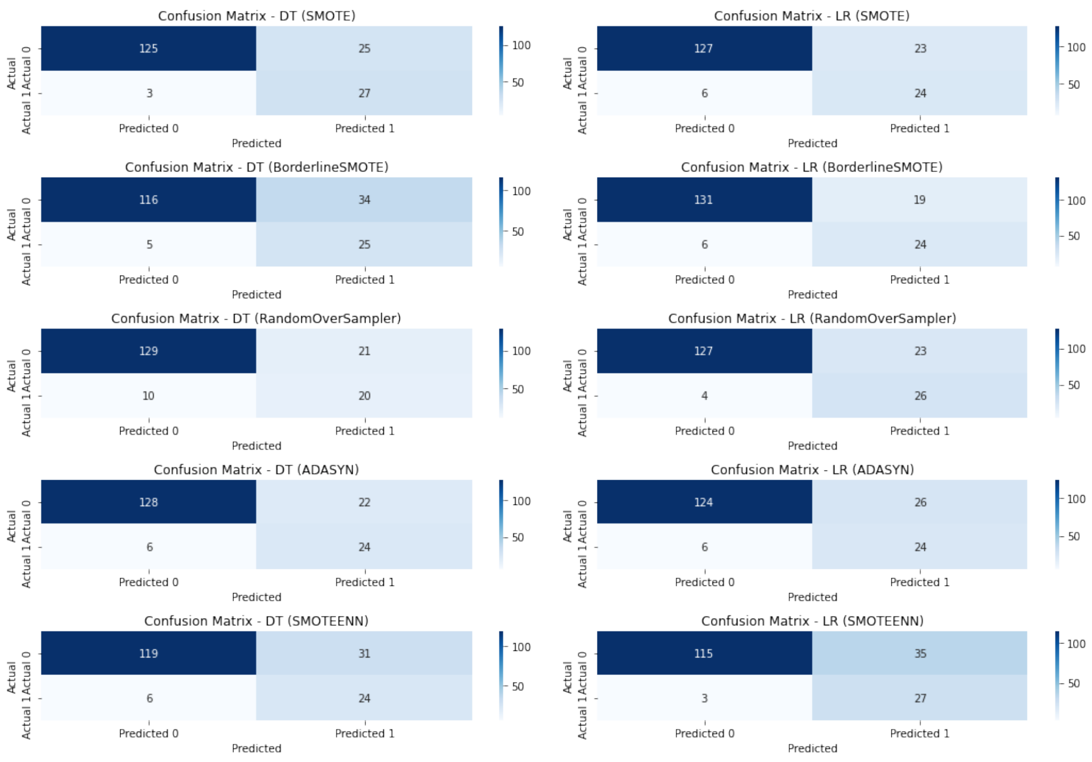
[66]:

```
      Resample Method               Model  Accuracy Score  \
0               SMOTE        DecisionTree        0.844444
1  RandomOverSampler  LogisticRegression        0.850000
1    BorderlineSMOTE  LogisticRegression        0.861111
1               SMOTE  LogisticRegression        0.838889
0              ADASYN        DecisionTree        0.833333
1              ADASYN  LogisticRegression        0.822222
1            SMOTEENN  LogisticRegression        0.788889
0    BorderlineSMOTE        DecisionTree        0.788889
0            SMOTEENN        DecisionTree        0.794444
0  RandomOverSampler        DecisionTree        0.827778

   Balanced Accuracy Score  F1 Score  precision  sensitivity  specificity
0                 0.866667  0.658537   0.519231     0.900000     0.833333
1                 0.856667  0.658228   0.530612     0.866667     0.846667
1                 0.836667  0.657534   0.558140     0.800000     0.873333
1                 0.823333  0.623377   0.510638     0.800000     0.846667
0                 0.820000  0.615385   0.500000     0.800000     0.840000
1                 0.813333  0.600000   0.480000     0.800000     0.826667
1                 0.833333  0.586957   0.435484     0.900000     0.766667
0                 0.806667  0.568182   0.431034     0.833333     0.780000
0                 0.796667  0.564706   0.436364     0.800000     0.793333
0                 0.763333  0.563380   0.487805     0.666667     0.860000
```

0.1.3 Confusion Matrix

[67]:

```
# List of models
models = [('DT', DecisionTreeClassifier(criterion='entropy',
                                              max_depth=5,␣
 ↪min_samples_leaf=4,
                                              min_samples_split=2)),
          ('LR', LogisticRegression(C=100, penalty='l2'))]

# DataFrame to store the results
results = pd.DataFrame(columns=["Resample Method", "Model",
                               "Accuracy Score",
                               "Balanced Accuracy Score", "F1 Score"])

# Create a figure with subplots for confusion matrices
fig, axes = plt.subplots(len(resample_methods1), len(models), figsize=(15, 10))

# Iterate over resampling methods
for i, resample_method in enumerate(resample_methods1):
    x_resampled, y_resampled = resample_method.
 ↪fit_resample(X_train_transformed, y_train)

    # Iterate over models
    for j, (clf_name, clf) in enumerate(models):
        ax = axes[i][j]  # Get the current subplot

        # Fit the model on the resampled data and make predictions
        clf.fit(x_resampled, y_resampled)
        predictions = clf.predict(X_test_transformed)
        accuracy = accuracy_score(y_test, predictions)
        bal_accuracy = balanced_accuracy_score(y_test, predictions)
        f1score = f1_score(y_test, predictions)

        # Confusion matrix
        cm = confusion_matrix(y_test, predictions)
        cm_df = pd.DataFrame(cm, index=['Actual 0', 'Actual 1'],␣
 ↪columns=['Predicted 0', 'Predicted 1'])

        # Plot confusion matrix on the current subplot
        sns.heatmap(cm_df, annot=True, fmt='d', cmap='Blues', ax=ax)
        ax.set_title(f'Confusion Matrix - {clf_name} ({resample_method.
 ↪__class__.__name__})')
        ax.set_xlabel('Predicted')
        ax.set_ylabel('Actual')

# Adjust the spacing between subplots and display the figure
plt.tight_layout()
plt.show()
plt.savefig('Confusion_matrix.png')
```



```
<Figure size 432x288 with 0 Axes>
```

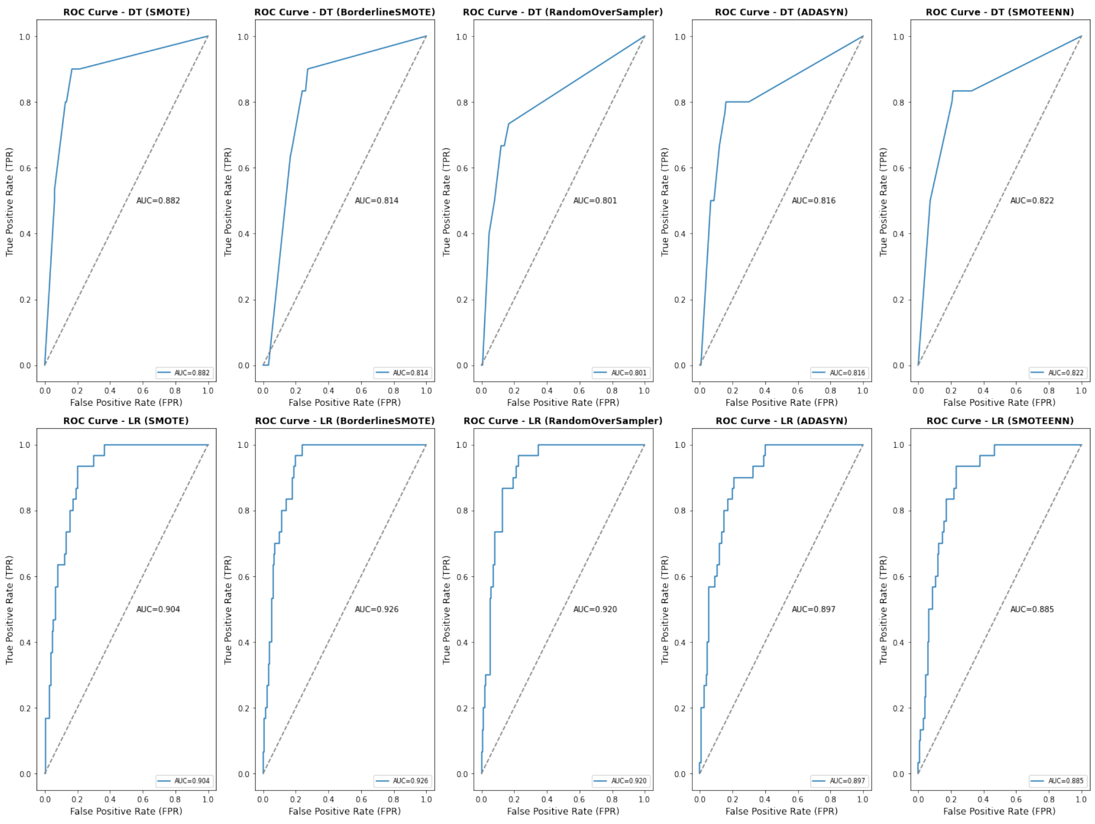
0.1.4 ROC AUC Curve

```
[68]: models = [('DT', DecisionTreeClassifier(criterion='entropy',
                                                max_depth=5,␣
       ↪min_samples_leaf=4,
                                                min_samples_split=2)),
                ('LR', LogisticRegression(C=100, penalty='l2'))]

      result_table = pd.DataFrame(columns=['Resample Method', 'Model', 'fpr', 'tpr',␣
       ↪'auc'])

      # Train the models and record the results
      for resample_method in resample_methods1:
          x_resampled, y_resampled = resample_method.
       ↪fit_resample(X_train_transformed, y_train)

          for clf_name, clf in tqdm(models):
              model = clf.fit(x_resampled, y_resampled)
              if hasattr(model, 'predict_proba'):
                  y_proba = model.predict_proba(X_test_transformed)[:, 1]
```

```
        else:
            # For models without predict_proba, you may need to use␣
↪decision_function or other appropriate method
            y_proba = model.decision_function(X_test_transformed)

        fpr, tpr, _ = roc_curve(y_test, y_proba)
        auc_value = roc_auc_score(y_test, y_proba)

        result_table = result_table.append({'Resample Method': resample_method.
↪__class__.__name__,
                                            'Model': clf_name,
                                            'fpr': fpr,
                                            'tpr': tpr,
                                            'auc': auc_value},␣
↪ignore_index=True)

# Plot ROC curves for each model and resampling method
fig, axs = plt.subplots(len(models), len(resample_methods1), figsize=(20, 15))
for i, resample_method in enumerate(resample_methods1):
    for j, (clf_name, clf) in enumerate(models):
        ax = axs[j, i]
        roc_data = result_table[(result_table['Resample Method'] ==␣
↪resample_method.__class__.__name__) &
                                (result_table['Model'] == clf_name)]
        if not roc_data.empty:
            fpr = roc_data['fpr'].values[0]
            tpr = roc_data['tpr'].values[0]
            auc_value = roc_data['auc'].values[0]

            # Plot ROC curve
            ax.plot(fpr, tpr, label="AUC={:.3f}".format(auc_value))

            # Plot the diagonal line (random classifier)
            ax.plot([0, 1], [0, 1], color='gray', linestyle='--')

            # Set axis labels and title
            ax.set_xlabel("False Positive Rate (FPR)", fontsize=12)
            ax.set_ylabel("True Positive Rate (TPR)", fontsize=12)
            ax.set_title(f'ROC Curve - {clf_name} ({resample_method.__class__.
↪__name__})', fontweight='bold',
                         fontsize=12)

            # Display AUC value near the point (0.5, 0.5)
            ax.text(0.7, 0.5, f'AUC={auc_value:.3f}', ha='center', va='center',␣
↪fontsize=10)
```

```
            # Add legend
            ax.legend(prop={'size': 8}, loc='lower right')
        else:
            ax.axis('off')

# Adjust layout to prevent overlap
plt.tight_layout()
plt.show()
plt.savefig('ROC_AUC_curves.png')
```

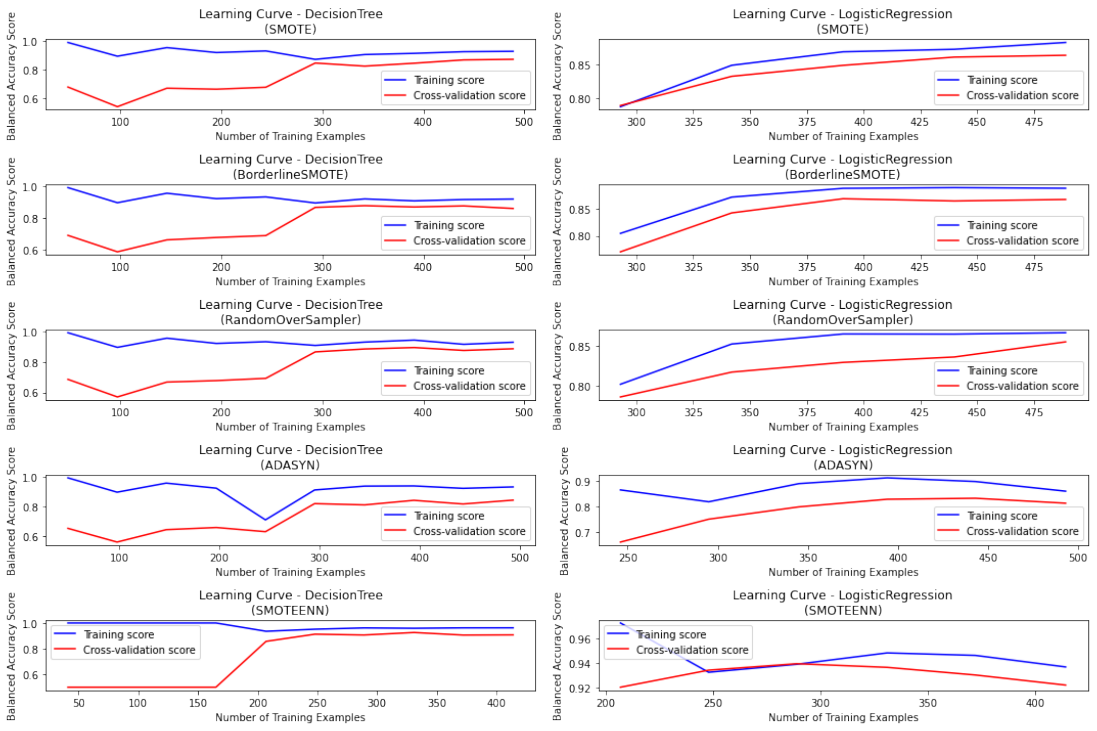
```
100%|      | 2/2 [00:00<00:00, 39.86it/s]
100%|      | 2/2 [00:00<00:00, 13.03it/s]
100%|      | 2/2 [00:00<00:00,  9.08it/s]
100%|      | 2/2 [00:00<00:00, 20.83it/s]
100%|      | 2/2 [00:00<00:00, 22.24it/s]
```



```
<Figure size 432x288 with 0 Axes>
```

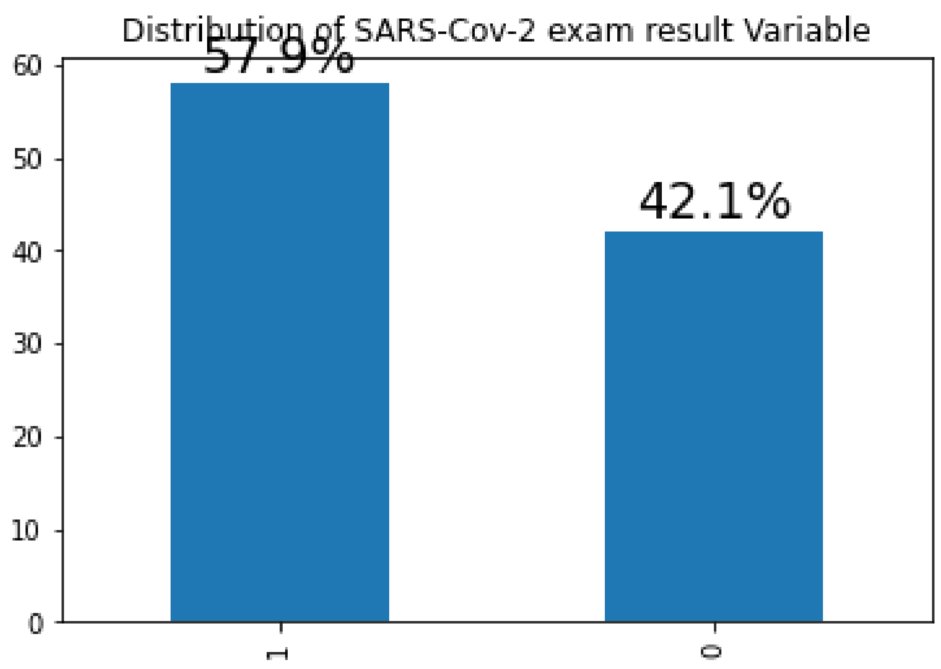
0.1.5 Learning Curve to check if there is not overfiting According to each resampled method

[69]:
```
models = [('DecisionTree', DecisionTreeClassifier(criterion='entropy',
                                                   max_depth=5,␣
 ↪min_samples_leaf=4,
                                                   min_samples_split=2)),
          ('LogisticRegression', LogisticRegression(C=100, penalty='l2'))]

results = pd.DataFrame(columns=["Resample Method", "Model",
                                "Accuracy Score",
                                "Balanced Accuracy Score", "F1 Score"])

fig, axes = plt.subplots(len(resample_methods1), len(models), figsize=(15, 10))

for i, resample_method in enumerate(resample_methods1):
    x_resampled, y_resampled = resample_method.
 ↪fit_resample(X_train_transformed, y_train)

    for j, (clf_name, clf) in enumerate(models):
        ax = axes[i][j]

        train_sizes, train_scores, test_scores = learning_curve(
            clf, x_resampled, y_resampled, cv=3, scoring='balanced_accuracy',␣
 ↪train_sizes=np.linspace(0.1, 1.0, 10)
        )

        train_mean = np.mean(train_scores, axis=1)
        train_std = np.std(train_scores, axis=1)
        test_mean = np.mean(test_scores, axis=1)
        test_std = np.std(test_scores, axis=1)

        # Plot learning curve
        ax.plot(train_sizes, train_mean, label='Training score', color='blue')

        ax.plot(train_sizes, test_mean, label='Cross-validation score',␣
 ↪color='red')


        ax.set_xlabel('Number of Training Examples')
        ax.set_ylabel('Balanced Accuracy Score')
        ax.set_title(f'Learning Curve - {clf_name}\n({resample_method.__class__.
 ↪__name__})')
        ax.legend(loc='best')

plt.tight_layout()
plt.savefig('learning_curves.png')
plt.show()
```



0.1.6 Plot the Tree with the best resampled Method : SMOTE

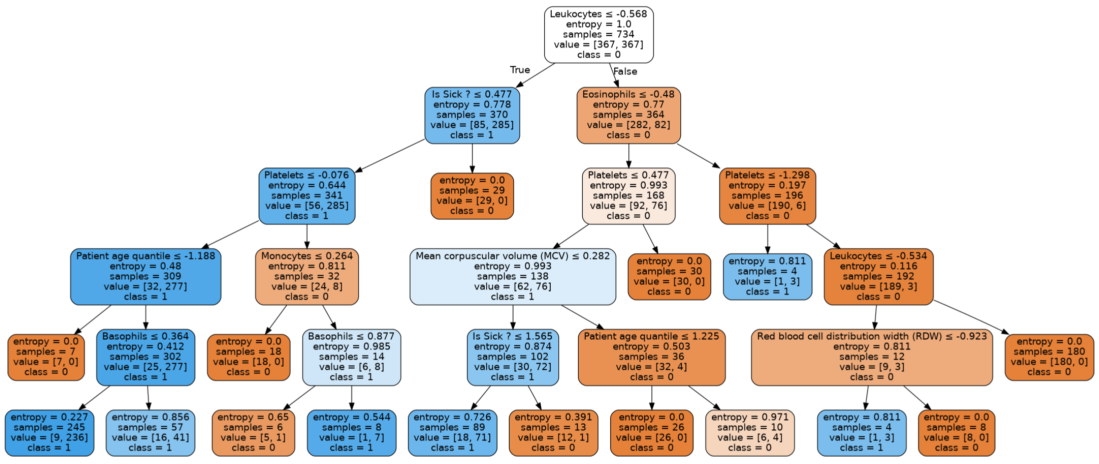
[70]:
```
#To check our target variable()

ax = y_resampled.value_counts(normalize=True).mul(100).plot.bar()
for p in ax.patches:
    y = p.get_height()
    x = p.get_x() + p.get_width() / 2

    # Label of bar height
    label = "{:.1f}%".format(y)

    # Annotate plot
    ax.annotate(label, xy=(x, y), xytext=(0, 5), textcoords="offset points",␣
 ↪ha="center", fontsize=19)
    # Add a title to the plot
ax.set_title('Distribution of SARS-Cov-2 exam result Variable')
plt.savefig('Imbalanced_data.png')
# Remove y axis
#ax.get_yaxis().set_visible(False).value_counts(normalize=True).mul(100).plot.
 ↪bar()
```



[71]:
```
### Draw Decision Tree for the best model
#OverSampled using Smote


resample_method = SMOTE(random_state=42)

x_resampled, y_resampled = resample_method.fit_resample(X_train_transformed,␣
 ↪y_train)

clf = DecisionTreeClassifier(criterion='entropy',
                             max_depth=5,
                             min_samples_leaf=4,min_samples_split=2)

dt = clf.fit(x_resampled, y_resampled)

feature_cols = X_train.columns.tolist()

dot_data = StringIO()
export_graphviz(dt, out_file=dot_data,
                filled=True, rounded=True,
                special_characters=True,feature_names =␣
 ↪feature_cols,class_names=['0','1'])
graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
graph.write_png('fdec_tree_gini.png')
```

```
Image(graph.create_png())
```

[71]:



0.1.7 Further Investigation For each resampled method (add some strategy of re-sampling),

0.1.8 find the best hyperparameter and compute the metric ,

0.1.9 then sort the model by ascending = False Accurracy score

[72]:

```
results = pd.DataFrame(columns=["Resample Method", "Model",
                                "Best Parameters", "Accuracy Score",
                                "Balanced Accuracy Score", "F1 Score",
                                "Recall", "Specificity", "Precision"])

resample_methods2= [SMOTE(random_state=42,sampling_strategy=0.9),
                    BorderlineSMOTE(random_state=42, sampling_strategy=0.8),
                    RandomOverSampler(random_state=42, sampling_strategy= 0.7),
                   ADASYN(random_state = 42, sampling_strategy= 0.8),
                   SMOTEENN(random_state=42, sampling_strategy= 0.7),
                    make_pipeline(SMOTE(random_state=42, sampling_strategy = 0.
 ↪8),
                         ADASYN(sampling_strategy = 0.9))
                  ]

# Iterate over each resampling method
for resample_method in resample_methods2:
    model_results = pd.DataFrame(columns=["Model", "Best Parameters", "Accuracy␣
 ↪Score", "Balanced Accuracy Score", "F1 Score", "Recall", "Specificity",␣
 ↪"Precision"])
    x_resampled, y_resampled = resample_method.
 ↪fit_resample(X_train_transformed, y_train)
```

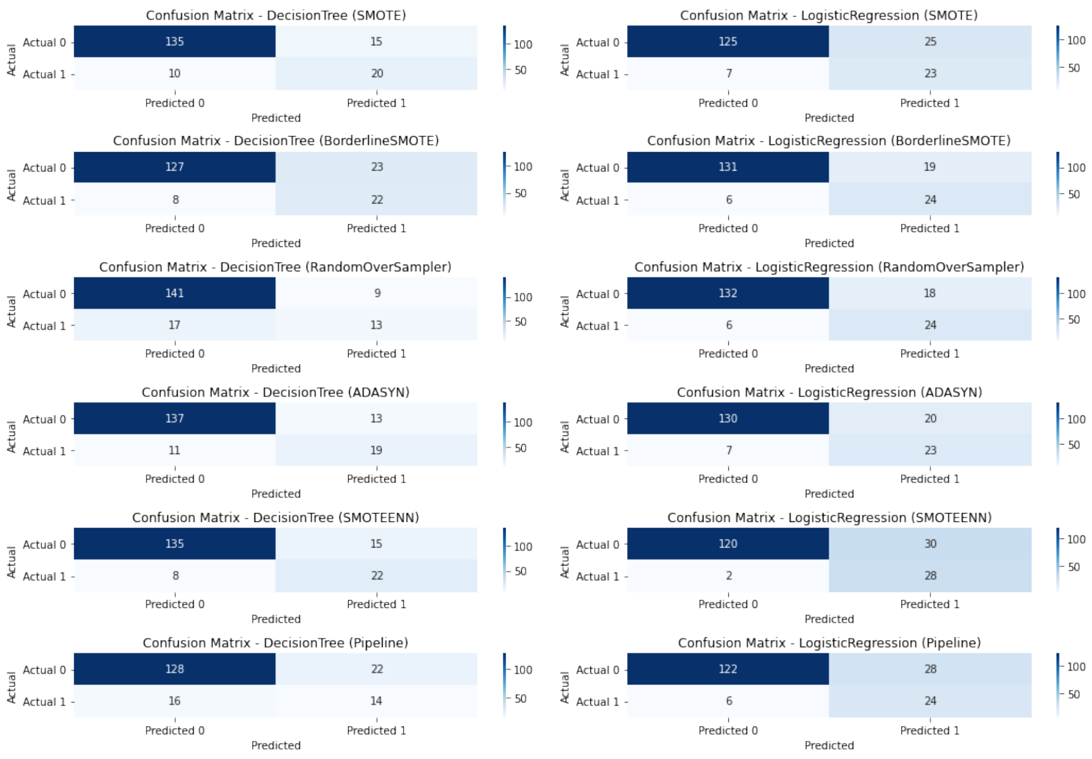
```
    # Initialize models
    models = {
        'DecisionTree': DecisionTreeClassifier(random_state=42),
        'LogisticRegression': LogisticRegression(random_state=42)
    }

    # Iterate over each model
    for model_name, model in models.items():
        param_grid = param_grids[model_name]
        grid_search = GridSearchCV(estimator=model, param_grid=param_grid,␣
 ↪cv=3, scoring=scorers, refit='Balanced Accuracy')
        grid_search.fit(x_resampled, y_resampled)

        # Get the best parameters and the best score
        best_params = grid_search.best_params_
        best_score = grid_search.best_score_

        # Set the best hyperparameters for the model
        clf = model.set_params(**best_params)

        # Train the model and make predictions
        clf.fit(x_resampled, y_resampled)
        predictions = clf.predict(X_test_transformed)

        # Compute metrics using the best hyperparameters
        accuracy = accuracy_score(y_test, predictions)
        bal_accuracy = balanced_accuracy_score(y_test, predictions)
        f1score = f1_score(y_test, predictions)
        recall = recall_score(y_test, predictions)
        specificity = recall_score(y_test, predictions, pos_label=0)
        precision = precision_score(y_test, predictions)

        new_row = {"Model": model_name, "Best Parameters": best_params,␣
 ↪"Accuracy Score": accuracy,
                   "Balanced Accuracy Score": bal_accuracy, "F1 Score":␣
 ↪f1score, "Recall": recall,
                   "Specificity": specificity, "Precision": precision}
        model_results = model_results.append(new_row, ignore_index=True)

    model_results["Resample Method"] = resample_method.__class__.__name__
    results = pd.concat([results, model_results])

results.sort_values(by="Accuracy Score", ascending=False, inplace=True)
results.to_excel("resampling_results2.xlsx")
results
```

[72]:
```
       Resample Method               Model  \
0             SMOTEENN        DecisionTree
1   RandomOverSampler  LogisticRegression
0               ADASYN        DecisionTree
0                SMOTE        DecisionTree
1     BorderlineSMOTE  LogisticRegression
0   RandomOverSampler        DecisionTree
1               ADASYN  LogisticRegression
0     BorderlineSMOTE        DecisionTree
1                SMOTE  LogisticRegression
1             SMOTEENN  LogisticRegression
1             Pipeline  LogisticRegression
0             Pipeline        DecisionTree

                                     Best Parameters  Accuracy Score  \
0  {'criterion': 'entropy', 'max_depth': 7, 'min_…       0.872222
1                         {'C': 0.1, 'penalty': 'l2'}        0.866667
0  {'criterion': 'gini', 'max_depth': None, 'min_…       0.866667
0  {'criterion': 'gini', 'max_depth': 10, 'min_sa…       0.861111
1                         {'C': 0.1, 'penalty': 'l2'}        0.861111
0  {'criterion': 'gini', 'max_depth': None, 'min_…       0.855556
1                         {'C': 0.1, 'penalty': 'l2'}        0.850000
0  {'criterion': 'entropy', 'max_depth': 7, 'min_…       0.827778
1                         {'C': 0.1, 'penalty': 'l2'}        0.822222
1                         {'C': 0.1, 'penalty': 'l2'}        0.822222
1                         {'C': 0.1, 'penalty': 'l2'}        0.805556
0  {'criterion': 'entropy', 'max_depth': 10, 'min…       0.800000

   Balanced Accuracy Score  F1 Score    Recall  Specificity  Precision
0                 0.816667  0.656716  0.733333     0.900000   0.594595
1                 0.840000  0.666667  0.800000     0.880000   0.571429
0                 0.773333  0.612903  0.633333     0.913333   0.593750
0                 0.783333  0.615385  0.666667     0.900000   0.571429
1                 0.836667  0.657534  0.800000     0.873333   0.558140
0                 0.686667  0.500000  0.433333     0.940000   0.590909
1                 0.816667  0.630137  0.766667     0.866667   0.534884
0                 0.790000  0.586667  0.733333     0.846667   0.488889
1                 0.800000  0.589744  0.766667     0.833333   0.479167
1                 0.866667  0.636364  0.933333     0.800000   0.482759
1                 0.816667  0.588235  0.833333     0.800000   0.454545
0                 0.680000  0.454545  0.500000     0.860000   0.416667
```

[73]:
```python
# List of models
models = [('DecisionTree', DecisionTreeClassifier(random_state=42)),
          ('LogisticRegression', LogisticRegression(random_state=42))]

# DataFrame to store the results
```

```
results = pd.DataFrame(columns=["Resample Method", "Model", "Best Parameters"])

# Create a figure with subplots for confusion matrices
fig, axes = plt.subplots(len(resample_methods2), len(models), figsize=(15, 10))

# Iterate over resampling methods
for i, resample_method in enumerate(resample_methods2):
    x_resampled, y_resampled = resample_method.
 ↪fit_resample(X_train_transformed, y_train)

    # Iterate over models
    for j, (model_name, model) in enumerate(models):
        ax = axes[i][j]  # Get the current subplot

        param_grid = param_grids[model_name]
        grid_search = GridSearchCV(estimator=model, param_grid=param_grid,␣
 ↪cv=3, scoring=scorers, refit='Balanced Accuracy')
        grid_search.fit(x_resampled, y_resampled)

        # Get the best parameters
        best_params = grid_search.best_params_

        # Set the best hyperparameters for the model
        clf = model.set_params(**best_params)

        # Fit the model on the resampled data and make predictions
        clf.fit(x_resampled, y_resampled)
        predictions = clf.predict(X_test_transformed)

        # Confusion matrix
        cm = confusion_matrix(y_test, predictions)
        cm_df = pd.DataFrame(cm, index=['Actual 0', 'Actual 1'],␣
 ↪columns=['Predicted 0', 'Predicted 1'])

        # Plot confusion matrix on the current subplot
        sns.heatmap(cm_df, annot=True, fmt='d', cmap='Blues', ax=ax)
        ax.set_title(f'Confusion Matrix - {model_name} ({resample_method.
 ↪__class__.__name__})')
        ax.set_xlabel('Predicted')
        ax.set_ylabel('Actual')

        new_row = {"Resample Method": resample_method.__class__.__name__,␣
 ↪"Model": model_name,
                "Best Parameters": best_params}
        results = results.append(new_row, ignore_index=True)

# Adjust the spacing between subplots and display the figure
```

```
plt.tight_layout()
plt.show()
plt.savefig('Confusion_matrix_resampling2.png')

# Save the results to a file
results.to_excel("confusion_matrix_results2.xlsx")
```



```
<Figure size 432x288 with 0 Axes>
```