A Project Report
on

EARLY PREDICTION OF DIABETES MELLITUS USING INTENSIVE CARE DATA TO IMPROVE CLINICAL DECISIONS

submitted in partial fulfillment of the requirements for the award of the degree of

BACHELOR OF TECHNOLOGY

in

COMPUTER SICENCE AND ENGINEERING

by

17WH1A0519	**Ms. T.SAI SARIKA**
18WH5A0503	**Ms. CH.APARNA**
18WH5A0505	**Ms. Y.KAVYA**

under the esteemed guidance of

Ms. SHANTI GUNNA
Assistant Professor

Department of Computer Science and Engineering
BVRIT HYDERABAD
College of Engineering for Women
(NBA Accredited – EEE, ECE, CSE and IT)
(Approved by AICTE, New Delhi and Affiliated to JNTUH, Hyderabad)
Bachupally, Hyderabad – 500090

June, 2021

DECLARATION

We hereby declare that the work presented in this project entitled **''EARLY PREDICTION OF DIABETES MELLITUS USING INTENSIVE CARE DATA TO IMPROVE CLINICAL DECISIONS''** submitted towards completion of Project Work in IV year of B.Tech., CSE at 'BVRIT HYDERABAD College of Engineering For Women', Hyderabad is an authentic record of our original work carried out under the guidance of Mr. Shanti Gunna, Assistant Professor, Department of CSE.

Sign. with date:

Ms. T.SAI SARIKA

(17WH1A0519)

Sign. with date:

Ms. CH.APARNA

(18WH5A0503)

Sign. with date:

Ms. Y.KAVYA

(18WH5A0505)

BVRIT HYDERABAD
College of Engineering for Women
(NBA Accredited – EEE, ECE, CSE and IT)
(Approved by AICTE, New Delhi and Affiliated to JNTUH, Hyderabad)
Bachupally, Hyderabad – 500090

Department of Computer Science and Engineering

Certificate

This is to certify that the Project Work report on **''EARLY PREDICTION OF DIABETES MELLITUS USING INTENSIVE CARE DATA TO IMPROVE CLINICAL DECISIONS**" is a bonafide work carried out by Ms. T.SAI SARIKA (17WH1A0519) ; Ms. CH.APARNA (18WH5A0503) ; Ms. Y.KAVYA (18WH5A0505) in the partial fulfillment for the award of B.Tech. degree in **Computer Science and Engineering, BVRIT HYDERABAD College of Engineering for Women, Bachupally, Hyderabad**, affiliated to Jawaharlal Nehru Technological University Hyderabad, Hyderabad under my guidance and supervision.

The results embodied in the project work have not been submitted to any other University or Institute for the award of any degree or diploma.

Head of the Department
Dr. Ch.Srinivasulu
Professor and HoD,
Department of CSE

Guide
Mr. Shanti Gunna
Assistant Professor

External Examiner

Acknowledgements

We would like to express our sincere thanks to **Dr. K V N Sunitha, Principal**, **BVRIT HYDERABAD College of Engineering for Women**, for providing the working facilities in the college.

Our sincere thanks and gratitude to our **Dr. Ch.Srinivasulu, Professor**, Department of CSE, **BVRIT HYDERABAD College of Engineering for Women** for all the timely support and valuable suggestions during the period of our project.

We are extremely thankful and indebted to our internal guide, **Mr. Shanti Gunna, Assistant Professor**, Department of CSE**, BVRIT HYDERABAD College of Engineering for Women** for his constant guidance, encouragement and moral support throughout the project.

Finally, we would also like to thank our Project Coordinator, all the faculty and staff of **CSE** Department who helped us directly or indirectly, parents and friends for their cooperation in completing the project work.

Ms. T.SAI SARIKA

(17WH1A0519)

Ms. CH.APARNA

(18WH5A0503)

Ms. Y.KAVYA

(18WH5A0505)

Contents

1. INTRODUCTION

We will build a model to determine whether a patient has been diagnosed with Diabetes Mellitus before within the first 24 hours of being admitted to an Intensive Care Unit (ICU). To improve a patient's outcome in an ICU knowledge about their medical conditions can improve clinical decisions. However, often a patient's medical records are not immediately available due to transfer times. An additional challenge is when a patient is not able to provide such information due to their health condition, e.g. shock or unconsiousness. Therefore, it is important to be able to diagnose whether a patient has chronical diseases based on data that can be gathered within the first 24 hours of being admitted to an ICU.

1.1 Objectives

Getting a rapid understanding of the context of a patient's overall health has been particularly important during the COVID-19 pandemic as healthcare workers around the world struggle with hospitals overloaded by patients in critical condition.
Intensive Care Units (ICUs) often lack verified medical histories for incoming patients. A patient in distress or a patient who is brought in confused or unresponsive may not be able to provide information about chronic conditions such as heart disease, injuries, or diabetes. Medical records may take days to transfer, especially for a patient from another medical provider or system.

Knowledge about chronic conditions such as diabetes can inform clinical decisions about patient care and ultimately improve patient outcomes.

1.2 Methodology

To predict whether a patient has diabetes or not a large collection of patient data or information is required. The data is from MIT's GOSSIS (Global Open Source Severity of Illness Score) initiative. In this section the methodology followed is discussed in detail.

1.2.1 Dataset

Proper and large dataset is required for all classification research during the training and the testing phase. The dataset for the experiment is from the MIT's GOSSIS (Global Open Source Severity of Illness Score) initiative database which contains different patient data and their labels.

Training dataset - 130157 rows, 181 columns
Testing dataset - 10324 rows, 180 columns

MIT's GOSSIS community initiative, with privacy certification from the Harvard Privacy Lab, has provided a dataset of more than 130,000 hospital Intensive Care Unit (ICU) visits from patients, spanning a one-year timeframe. This data is part of a growing global effort and consortium spanning Argentina, Australia, New Zealand, Sri Lanka, Brazil, and more than 200 hospitals in the United States. The dataset contains

1.2.2 The Proposed models:

Logistic regression introduction

Classification techniques are an essential part of machine learning and data mining applications. Approximately 70% of problems in Data Science are classification problems. There are lots of classification problems that are available, but the logistics regression is common and is a useful regression method for solving the binary classification problem. Another category of classification is Multinomial classification, which handles the issues where multiple classes are present in the target variable.

Logistic Regression is one of the most simple and commonly used Machine Learning algorithms for two-class classification. It is easy to implement and can be used as the baseline for any binary classification problem. Its basic fundamental concepts are also constructive in deep learning. Logistic regression describes and estimates the relationship between one dependent binary variable and independent variables.

Logistic Regression

Logistic regression is a statistical method for predicting binary classes. The outcome or target variable is dichotomous in nature. Dichotomous means there are only two possible classes. For example, it can be used for cancer detection problems. It computes the probability of an event occurrence.

It is a special case of linear regression where the target variable is categorical in nature. It uses a log of odds as the dependent variable. Logistic Regression predicts the probability of occurrence of a binary event utilizing a logit function.

Linear Regression Equation:

Where, y is dependent variable and x1, x2 ... and Xn are explanatory variables.

Sigmoid Function:

Apply Sigmoid function on linear regression:

Properties of Logistic Regression:

- The dependent variable in logistic regression follows Bernoulli Distribution.
- Estimation is done through maximum likelihood.
- No R Square, Model fitness is calculated through Concordance, KS-Statistics.

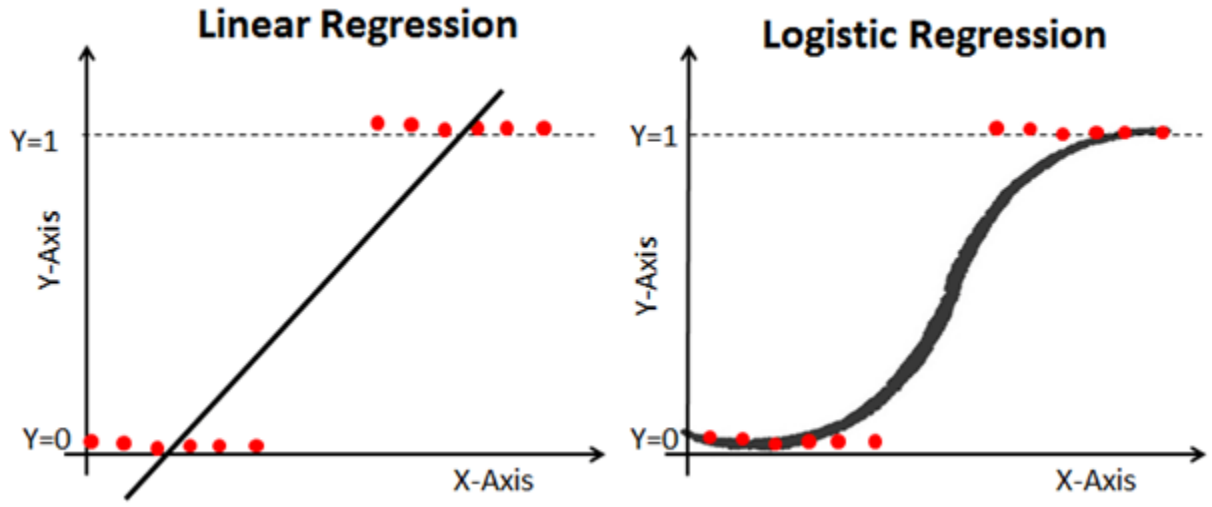


Fig 1.2.2.1: Linear Regression and Logistic Regression Diagram

Linear Regression Vs. Logistic Regression

Linear regression gives you a continuous output, but logistic regression provides a constant output. An example of the continuous output is house price and stock price. Example's of the discrete output is predicting whether a patient has cancer or not, predicting whether the customer will churn. Linear regression is estimated using Ordinary Least Squares (OLS) while logistic regression is estimated using Maximum Likelihood Estimation (MLE) approach.

Random Forest:

Random forest is a flexible, easy to use machine learning algorithm that produces, even without hyper-parameter tuning, a great result most of the time. It is also one of the most used algorithms, because of its simplicity and diversity (it can be used for both classification and regression tasks).

Random forest is a supervised learning algorithm. The "forest" it builds, is an ensemble of decision trees, usually trained with the “bagging” method. The general idea of the bagging method is that a combination of learning models increases the overall result.

HOW RANDOM FOREST WORKS

Random forest is a supervised learning algorithm. The "forest" it builds, is an ensemble of decision trees, usually trained with the “bagging” method. The general idea of the bagging method is that a combination of learning models increases the overall result.

Put simply: random forest builds multiple decision trees and merges them together to get a more accurate and stable prediction.

One big advantage of random forest is that it can be used for both classification and regression problems, which form the majority of current machine learning systems. Let's look at random forest in classification, since classification is sometimes considered the building block of machine learning. Below you can see how a random forest would look like with three trees:

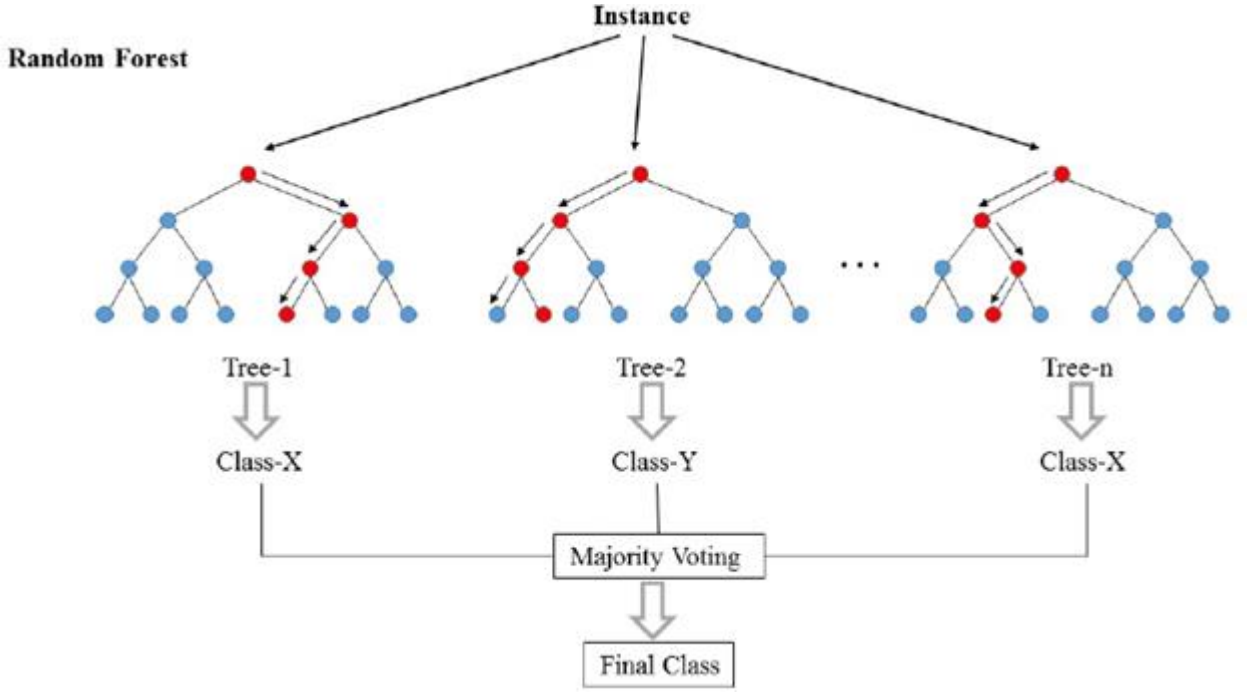


Fig 1.2.2.2: Random Forest Diagram

Random forest has nearly the same hyperparameters as a decision tree or a bagging classifier. Fortunately, there's no need to combine a decision tree with a bagging classifier because you can easily use the classifier-class of random forest. With random forest, you can also deal with regression tasks by using the algorithm's regressor.

Random forest adds additional randomness to the model, while growing the trees. Instead of searching for the most important feature while splitting a node, it searches for the best feature among a random subset of features. This results in a wide diversity that generally results in a better model.

Therefore, in random forest, only a random subset of the features is taken into consideration by the algorithm for splitting a node. You can even make trees more random by additionally using random thresholds for each feature rather than searching for the best possible thresholds (like a normal decision tree does).

One-Hot Encoding

Categorical data are variables that contain label values rather than numeric values.

The number of possible values is often limited to a fixed set.

Categorical variables are often called nominal.

For categorical variables where no such ordinal relationship exists, the integer encoding is not enough.

In fact, using this encoding and allowing the model to assume a natural ordering between categories may result in poor performance or unexpected results (predictions halfway between categories).

In this case, a one-hot encoding can be applied to the integer representation. This is where the integer encoded variable is removed and a new binary variable is added for each unique integer value.

In the "*color*" variable example, there are 3 categories and therefore 3 binary variables are needed. A "1" value is placed in the binary variable for the color and "0" values for the other colors.

For example:

```
1  red,    green,  blue
2  1,              0,              0
3  0,              1,              0
4  0,              0,              1
```

The binary variables are often called "dummy variables" in other fields, such as statistics.

Light Gradient Boosted Machine Algorithm

LightGBM is a gradient boosting framework that uses tree based learning algorithms. It is designed to be distributed and efficient with the following advantages:

- Faster training speed and higher efficiency.
- Lower memory usage.
- Better accuracy.

- Support of parallel, distributed, and GPU learning.
- Capable of handling large-scale data

Gradient boosting refers to a class of ensemble machine learning algorithms that can be used for classification or regression predictive modeling problems.
Ensembles are constructed from decision tree models. Trees are added one at a time to the ensemble and fit to correct the prediction errors made by prior models. This is a type of ensemble machine learning model referred to as boosting.
Models are fit using any arbitrary differentiable loss function and gradient descent optimization algorithm. This gives the technique its name, "*gradient boosting*," as the loss gradient is minimized as the model is fit, much like a neural network.

Light Gradient Boosted Machine, or **LightGBM** for short, is an open-source library that provides an efficient and effective implementation of the gradient boosting algorithm..
LightGBM extends the gradient boosting algorithm by adding a type of automatic feature selection as well as focusing on boosting examples with larger gradients. This can result in a dramatic speedup of training and improved predictive performance.As such, LightGBM has become a de facto algorithm for machine learning competitions when working with tabular data for regression and classification predictive modeling tasks. As such, it owns a share of the blame for the increased popularity and wider adoption of gradient boosting methods in general, along with Extreme Gradient Boosting (XGBoost).

Light Gradient Boosted Machine, or LightGBM for short, is an open-source implementation of gradient boosting designed to be efficient and perhaps more effective than other implementations.
As such, LightGBM refers to the open-source project, the software library, and the machine learning algorithm. In this way, it is very similar to the Extreme Gradient Boosting or XGBoost technique.

Gradient-based One-Side Sampling, or GOSS for short, is a modification to the gradient boosting method that focuses attention on those training examples that result in a larger gradient, in turn speeding up learning and reducing the computational complexity of the method

LightGBM Scikit-Learn API

LightGBM can be installed as a standalone library and the LightGBM model can be developed using the scikit-learn API.

The first step is to install the LightGBM library, if it is not already installed. This can be achieved using the pip python package manager on most platforms; for example:

```
sudo pip install lightgbm
```

You can then confirm that the LightGBM library was installed correctly and can be used by running the following script.

```
# check lightgbm version
import lightgbm
print(lightgbm.__version__)
```

Running the script will print your version of the LightGBM library you have installed.

Your version should be the same or higher. If not, you must upgrade your version of the LightGBM library.

```
2.3.1
```

If you require specific instructions for your development environment, see the tutorial:

- LightGBM Installation Guide

The LightGBM library has its own custom API, although we will use the method via the scikit-learn wrapper classes: LGBMRegressor and LGBMClassifier. This will allow us to use the full suite of tools from the scikit-learn machine learning library to prepare data and evaluate models.

Both models operate the same way and take the same arguments that influence how the decision trees are created and added to the ensemble.
Randomness is used in the construction of the model. This means that each time the algorithm is run on the same data, it will produce a slightly different model.
When using machine learning algorithms that have a stochastic learning algorithm, it is good practice to evaluate them by averaging their performance across multiple runs or repeats of cross-validation. When fitting a final model, it may be desirable to either increase the number of trees until the variance of the model is reduced across repeated evaluations, or to fit multiple final models and average their predictions.

Machine Learning is the fastest growing field in the world. Everyday there will be a launch of bunch of new algorithms, some of those fails and some achieve the peak of success. Today, I am touching one of the most successful machine learning algorithm, Light GBM.Light GBM is a gradient boosting framework that uses tree based learning algorithm.
Light GBM grows tree vertically while other algorithm grows trees horizontally meaning that Light GBM grows tree **leaf-wise** while other algorithm grows level-wise. It will choose the leaf with max delta loss to grow. When growing the same leaf, Leaf-wise algorithm can reduce more loss than a level-wise algorithm.Below diagrams explain the implementation of LightGBM and other boosting algorithms.

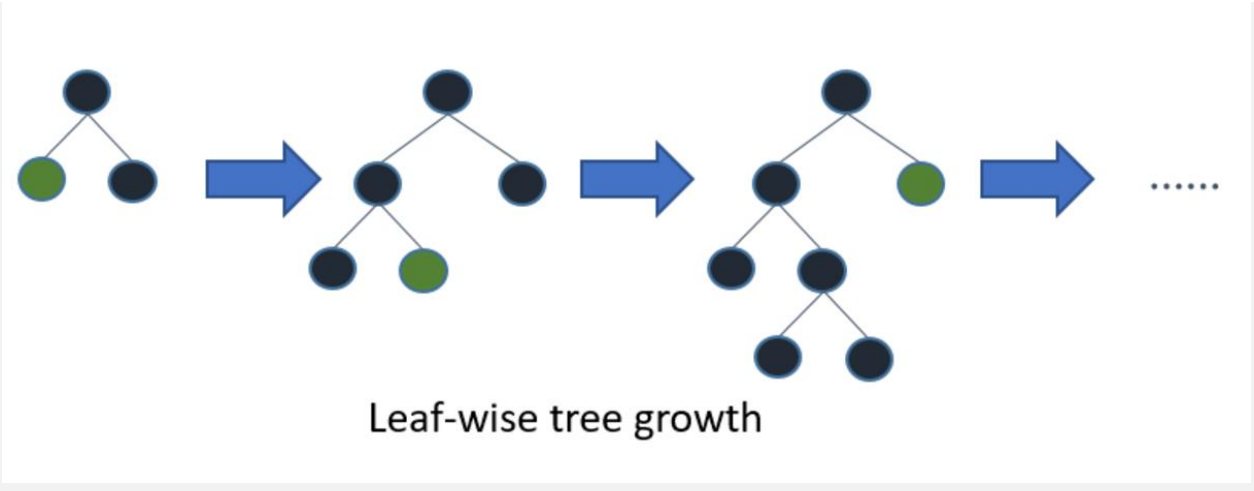


Fig 1.2.2.3: leaf-wise tree growth Diagram

Explains how LightGBM works

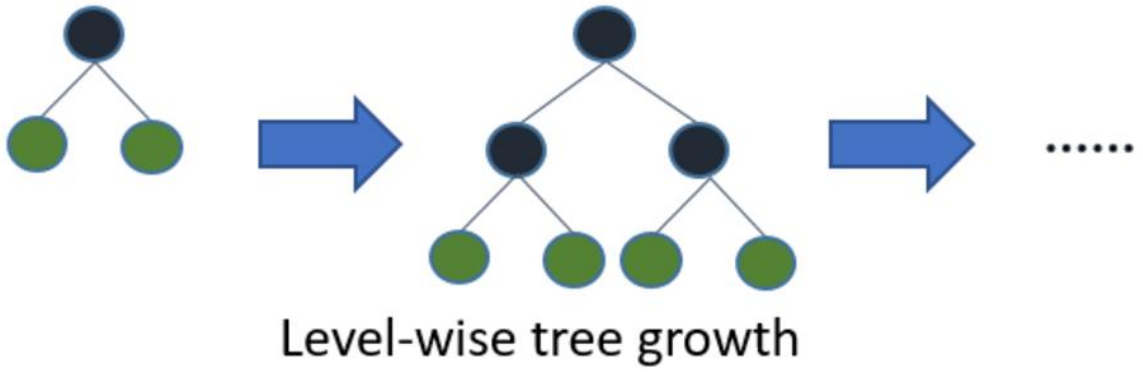


Fig 1.2.2.4: level-wise tree growth Diagram

How other boosting algorithm works

The size of data is increasing day by day and it is becoming difficult for traditional data science algorithms to give faster results. Light GBM is prefixed as 'Light' because of its **high speed.** Light GBM can **handle the large size** of data and **takes lower memory to run**. Another reason of why Light GBM is popular is because it **focuses on accuracy of results**. LGBM also **supports GPU learning** and thus data scientists are widely using LGBM for data science application development.

XGBoost Algorithm

XGBoost is a decision-tree-based ensemble Machine Learning algorithm that uses a Gradient boosting framework. XGBoost is an algorithm that has recently been dominating applied machine learning and for structured or tabular data. XGBoost is an implementation of gradient boosted decision trees designed for speed and performance. In prediction problems involving unstructured data (images, text, etc.) artificial neural networks tend to outperform all other algorithms or frameworks. However, when it comes to small-to-medium structured/tabular data, decision tree based algorithms are considered best-in-class right now.

XGBoost is a software library that you can download and install on your machine, then access from a variety of interfaces. Specifically, XGBoost supports the following main interfaces:

- Command Line Interface (CLI).
- C++ (the language in which the library is written).
- Python interface as well as a model in scikit-learn.
- R interface as well as a model in the caret package.
- Julia.
- Java and JVM languages like Scala and platforms like Hadoop.

XGBoost Features

The library is laser focused on computational speed and model performance, as such there are few frills. Nevertheless, it does offer a number of advanced features.

Model Features

The implementation of the model supports the features of the scikit-learn and R implementations, with new additions like regularization. Three main forms of gradient boosting are supported:

- **Gradient Boosting** algorithm also called gradient boosting machine including the learning rate.
- **Stochastic Gradient Boosting** with sub-sampling at the row, column and column per split levels.
- **Regularized Gradient Boosting** with both L1 and L2 regularization.

System Features

The library provides a system for use in a range of computing environments, not least:

- **Parallelization** of tree construction using all of your CPU cores during training.
- **Distributed Computing** for training very large models using a cluster of machines.
- **Out-of-Core Computing** for very large datasets that don’t fit into memory.
- **Cache Optimization** of data structures and algorithm to make best use of hardware.

Algorithm Features

The implementation of the algorithm was engineered for efficiency of compute time and memory resources. A design goal was to make the best use of available resources to train the model.
Some key algorithm implementation features include:

- **Sparse Aware** implementation with automatic handling of missing data values.
- **Block Structure** to support the parallelization of tree construction.
- **Continued Training** so that you can further boost an already fitted model on new data.

The two reasons to use XGBoost are also the two goals of the project:

1. Execution Speed.
2. Model Performance

Cross-Validation

In machine learning, we couldn't fit the model on the training data and can't say that the model will work accurately for the real data. For this, we must assure that our model got the correct patterns from the data, and it is not getting up too much noise. For this purpose, we use the cross-validation technique.
Cross-validation is a technique in which we train our model using the subset of the data-set and then evaluate using the complementary subset of the data-set.
The three steps involved in cross-validation are as follows :

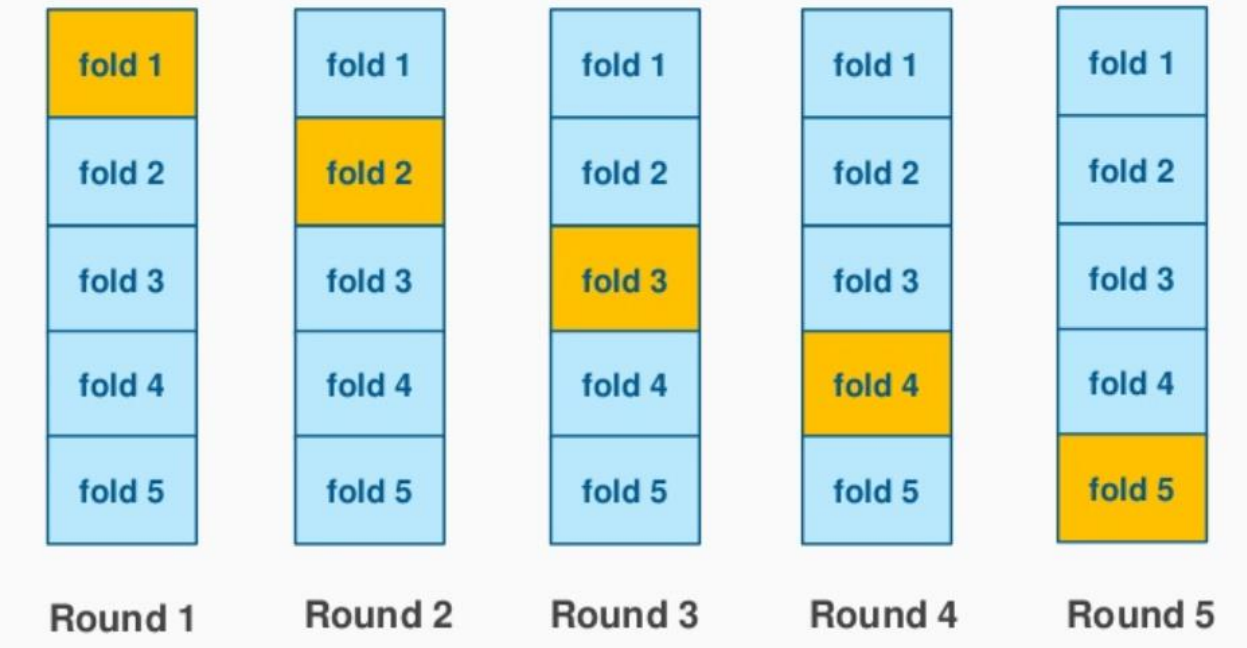
1. Reserve some portion of sample data-set.
2. Using the rest data-set train the model.
3. Test the model using the reserve portion of the data-set.

Stratified K-fold

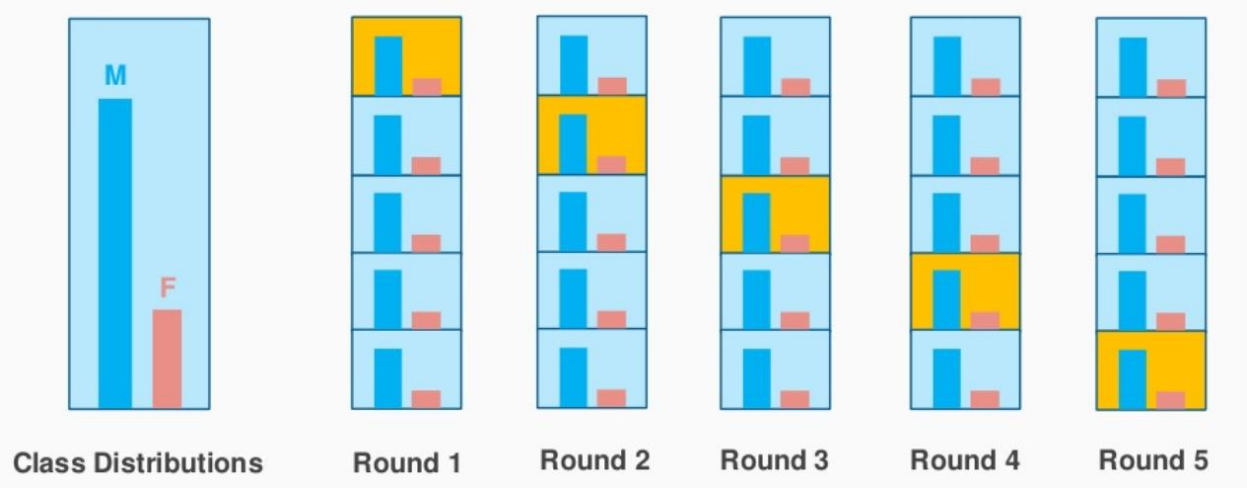
In **stratified k-fold** cross-validation, the **folds** are selected so that the mean response value is approximately equal in all the **folds**. In the case of a dichotomous classification, this means that each **fold** contains roughly the same proportions of the two types of class labels.

Stratified Cross Valiadtion: Splits the data into k folds, making sure each fold is an appropriate representative of the original data. (class distribution, mean, variance, etc)

Example of 5 fold **Cross Validation**:



Example of 5 folds **Stratified Cross Validation**:



2. THEORETICAL ANALYSIS OF THE PROPOSED PROJECT

2.1 Requirements Gathering

2.1.1 Software Requirements

Programming Language : Python 3.6

Dataset : GOSSIS initiative database

Packages : Numpy, Pandas, Os, Klib, Warnings, Matplotlib, Seaborn, Lightgbm, sklearn

Tool : Google Colaboratory

2.1.2 Hardware Requirements

Processor - Intel Core i5

Memory(RAM) - 8 GB

Storage – I TB

2.2 Technologies Description

Python

Python is an interpreted high-level programming language for general-purpose programming. Created by Guido van Rossum and first released in 1991, Python has a design philosophy that emphasizes code readability, notably using significant whitespace.

Python features a dynamic type system and automatic memory management. It supports multiple programming paradigms, including object-oriented, imperative, functional and procedural, and has a large and comprehensive standard library.

- Python is Interpreted − Python is processed at runtime by the interpreter. You do not need to compile your program before executing it. This is similar to PERL and PHP.
- Python is Interactive − you can actually sit at a Python prompt and interact with the interpreter directly to write your programs.

Python also acknowledges that speed of development is important. Readable and terse code is part of this, and so is access to powerful constructs that avoid tedious repetition of code. Maintainability also ties into this may be an all but useless metric, but it does say something about how much code you have to scan, read and/or understand to troubleshoot problems or tweak behaviors. This speed of development, the ease with which a programmer of other languages can pick up basic Python skills and the huge standard library is key to another area where Python excels. All its tools have been quick to implement, saved a lot of time, and several of them have later been patched and updated by people with no Python background - without breaking.

Dataset

The dataset consists of 130,157 patient records along in the TrainingWiDS2021.csv file. This is the data that we will sue to train our model.The UnlabeledWiDS2021.csv file consists of the unlabelled data and will be sued for testing purposes .

SampleSubmissionWiDS2021.csv and the SolutionTemplateWiDS2021.csv provide a template and the submissions should be in this form .

DataDictionaryWiDS2021.csv contains additional information about the dataset.

Numpy

Numpy is a general-purpose array-processing package. It provides a high-performance multidimensional array object, and tools for working with these arrays.

It is the fundamental package for scientific computing with Python. It contains various features including these important ones:

- A powerful N-dimensional array object
- Sophisticated (broadcasting) functions
- Tools for integrating C/C++ and Fortran code
- Useful linear algebra, Fourier transform, and random number capabilities

Besides its obvious scientific uses, Numpy can also be used as an efficient multi-dimensional container of generic data. Arbitrary data-types can be defined using Numpy which allows Numpy to seamlessly and speedily integrate with a wide variety of databases.

Pandas

Pandas is an open-source Python Library providing high-performance data manipulation and analysis tool using its powerful data structures. Python was majorly used for data munging and preparation. It had very little contribution towards data analysis. Pandas solved this problem. Using Pandas, we can accomplish five typical steps in the processing and analysis of data, regardless of the origin of data load, prepare, manipulate, model, and analyze. Python with Pandas is used in a wide range of fields including academic and commercial domains including finance, economics, Statistics, analytics, etc.

Matplotlib

Matplotlib is a Python 2D plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments across platforms. Matplotlib can be used in Python scripts, the Python and IPython shells, the Jupyter Notebook, web application servers, and four graphical user interface toolkits. Matplotlib tries to make easy things easy and hard things possible. You can generate plots, histograms, power spectra, bar charts, error charts, scatter plots, etc., with just a few lines of code. For simple plotting the pyplot module provides a MATLAB-like interface, particularly when combined with IPython. For the power user, you have full control of line styles, font properties, axes properties, etc, via an object oriented interface or via a set of functions familiar to MATLAB users.

Scikit – learn

Scikit-learn provides a range of supervised and unsupervised learning algorithms via a consistent interface in Python. It is licensed under a permissive simplified BSD license and is distributed under many Linux distributions, encouraging academic and commercial use. The library is built

upon the SciPy (Scientific Python) that must be installed before you can use scikit-learn. This stack that includes:

· NumPy: Base n-dimensional array package

· SciPy: Fundamental library for scientific computing

· Matplotlib: Comprehensive 2D/3D plotting

· IPython: Enhanced interactive console

· Sympy: Symbolic mathematics

· Pandas: Data structures and analysis

· Extensions or modules for SciPy care conventionally named SciKits. As such, the module provides learning algorithms and is named scikit-learn.

OS

The **OS** module in **Python** provides functions for interacting with the operating system. This module provides a portable way of using operating system-dependent functionality. The ***os*** and ***os**. path** modules include many functions to interact with the file system.

warnings: Package to handle warnings and 'ignore' is used when we need to filter out all the warnings

Time: Package to handle time manipulation

Seaborn

Seaborn is a Python data visualization library based on matplotlib. It provides a high-level interface for drawing attractive and informative statistical graphics.

Seaborn is a library for making statistical graphics in Python. It builds on top of matplotlib and integrates closely with pandas data structures.

Seaborn helps to explore and understand your data. Its plotting functions operate on dataframes and arrays containing whole datasets and internally perform the necessary semantic mapping and statistical aggregation to produce informative plots. Its dataset-oriented, declarative API lets you focus on what the different elements of your plots mean, rather than on the details of how to draw them.

Google Colaboratory

Google Colaboratory is a free Jupyter notebook environment that runs entirely in the cloud. Most importantly, it does not require a setup and the notebooks that you create can be simultaneously edited by your team members - just the way you edit documents in Google Docs. Colab supports many popular machine learning libraries which can be easily loaded in your notebook.

As we can perform the following using Google Colab.

- Write and execute code in Python
- Document your code that supports mathematical equations
- Create/Upload/Share notebooks
- Import/Save notebooks from/to Google Drive
- Import/Publish notebooks from GitHub
- Import external datasets e.g. from Kaggle
- Integrate PyTorch, TensorFlow, Keras, OpenCV
- Free Cloud service with free GPU

3.DESIGN

3.1 Introduction

Software design sits at the technical kernel of the software engineering process and is applied regardless of the development paradigm and area of application. Design is the first step in the development phase for any engineered product or system. The designer's goal is to produce a model or representation of an entity that will later be built. Beginning, once system requirement have been specified and analyzed, system design is the first of the three technical activities - design, code and test that is required to build and verify software.

The importance can be stated with a single word "Quality". Design is the place where quality is fostered in software development. Design provides us with representations of software that can assess for quality. Design is the only way that we can accurately translate a customer's view into a finished software product or system. Software design serves as a foundation for all the software engineering steps that follow. Without a strong design we risk building an unstable system – one that will be difficult to test, one whose quality cannot be assessed until the last stage.

During design, progressive refinement of data structure, program structure, and procedural details are developed reviewed and documented. System design can be viewed from either technical or project management perspective. From the technical point of view, design is comprised of four activities – architectural design, data structure design, interface design and procedural design.

3.2 Architecture Diagram

Exploratory Data Analysis

It is a way of visualizing, summarizing and interpreting the information that is hidden in rows and column format. EDA is one of the crucial step in data science that allows us to achieve certain insights and statistical measure that is essential for the business continuity, stockholders and data scientists. It performs to define and refine our important features variable selection, that will be used in our model.

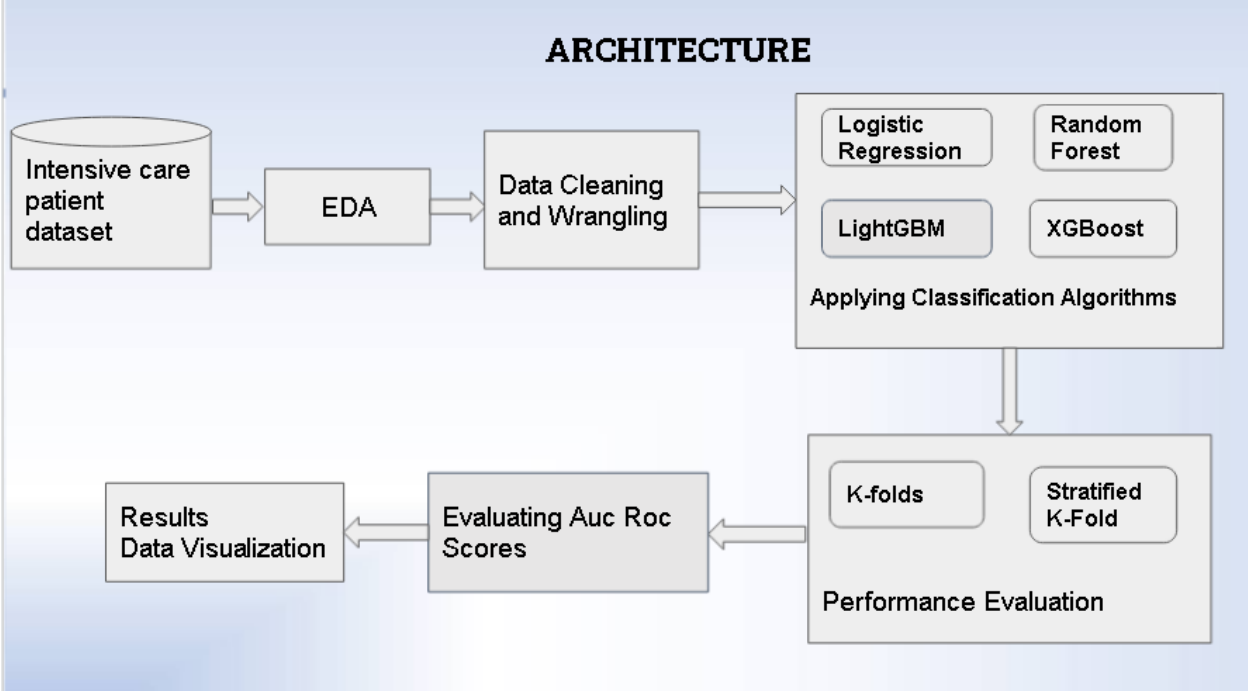


Fig 3.2: Architecture Diagram

Once EDA is complete and insights are drawn, its feature can be used for supervised and unsupervised machine learning modelling. The EDA is executed majorly by Uni-variate visualization, Bi-variate visualization, Multivariate Visualization and Dimensionality reduction. We initially make several hypothesis by looking at the data before we hit the modelling. And its quite a good practice cause that will engage you more with EDA part. EDA helps you in confirming and validating the hypothesis you make. And from here you start your feature engineering part and take a flight to machine learning modelling.

Data Cleaning

We can drop unnecessary information from a dataset using the drop() function, as well as how to set an index for your dataset so that items in it can be referenced easily.

Moreover, you learned how to clean object fields with the .str() accessor and how to clean the entire dataset using the applymap() method. Lastly, we explored how to skip rows in a CSV file and rename columns using the rename() method.

Knowing about data cleaning is very important, because it is a big part of data science. Here we need to have a basic understanding of how Pandas and NumPy can be leveraged to clean datasets!

Data Wrangling

Data wrangling involves processing the data in various formats like - merging, grouping, concatenating etc. for the purpose of analysing or getting them ready to be used with another set of data. Python has built-in features to apply these wrangling methods to various data sets to achieve the analytical goal.

Applying Classification Algorithms

Here we are applying various algorithms like Logistic regression and Random forest, LightGBM, and XGboost and comparing with each other . To obtain the highest accuracy of the training model.

3.3.1 Use Case Diagram

To model a system, the most important aspect is to capture the dynamic behavior. Dynamic behavior means the behavior of the system when it is running/operating.

Only static behavior is not sufficient to model a system rather dynamic behavior is more important than static behavior. In UML, there are five diagrams available to model the dynamic nature and use case diagram is one of them. Now as we have to discuss that the use case diagram is dynamic in nature, there should be some internal or external factors for making the interaction.

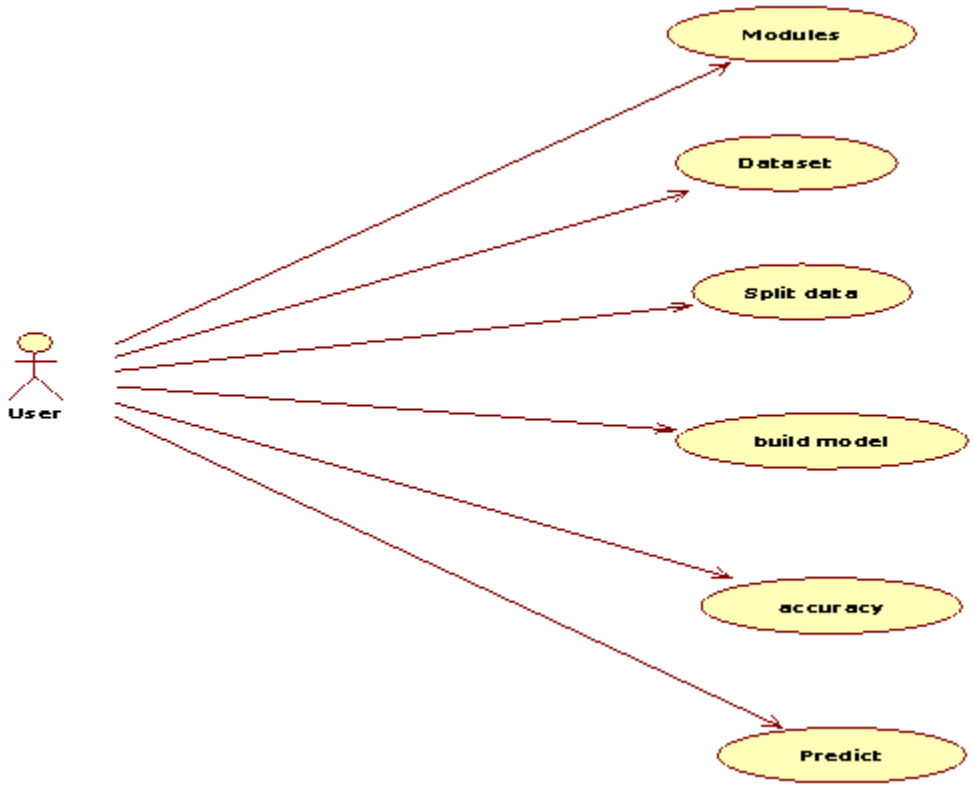


Fig 3.2.1: Use Case Diagram

These internal and external agents are known as actors. Use case diagrams consist of actors, use cases and their relationships. The diagram is used to model the system/subsystem of an application. A single use case diagram captures a particular functionality of a system.

Hence to model the entire system, a number of use case diagrams are used.

3.3.2 Sequence Diagram

Sequence Diagrams Represent the objects participating the interaction horizontally and time vertically. A Use Case is a kind of behavioral classifier that represents a declaration of an offered behavior. Each use case specifies some behavior, possibly including variants that the subject can perform in collaboration with one or more actors. Use cases define the offered behavior of the

subject without reference to its internal structure. These behaviors, involving interactions between the actor and the subject, may result in changes to the state of the subject and communications with its environment. A use case can include possible variations of its basic behavior, including exceptional behavior and error handling.

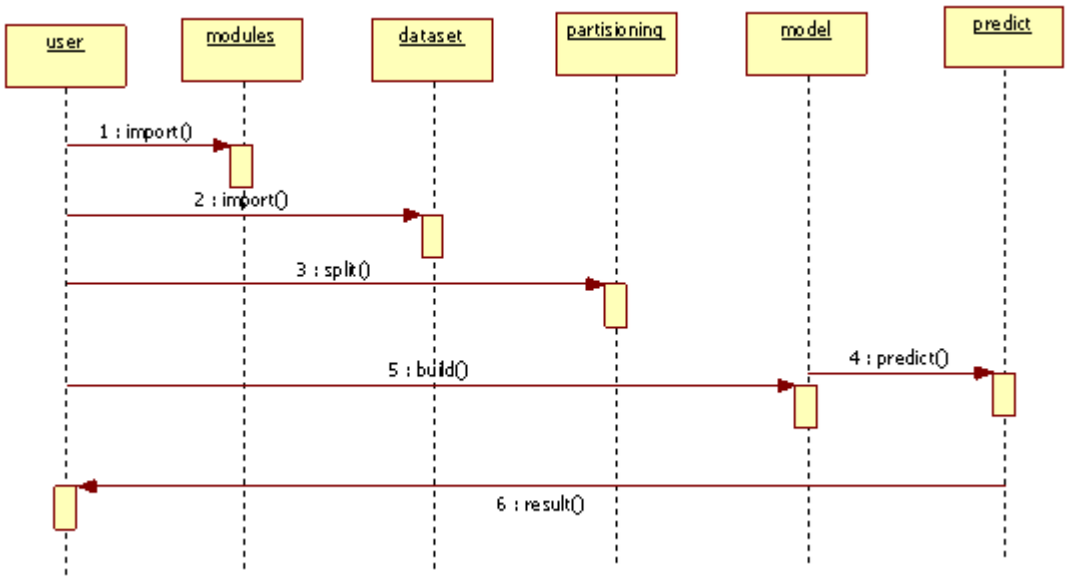


Fig 3.3.2: Sequence Diagram

3.3.3 Activity Diagram

Activity diagrams are graphical representations of Workflows of stepwise activities and actions with support for choice, iteration and concurrency. In the Unified Modeling Language, activity diagrams can be used to describe the business and operational step-by-step workflows of components in a system. An activity diagram shows the overall flow of control.

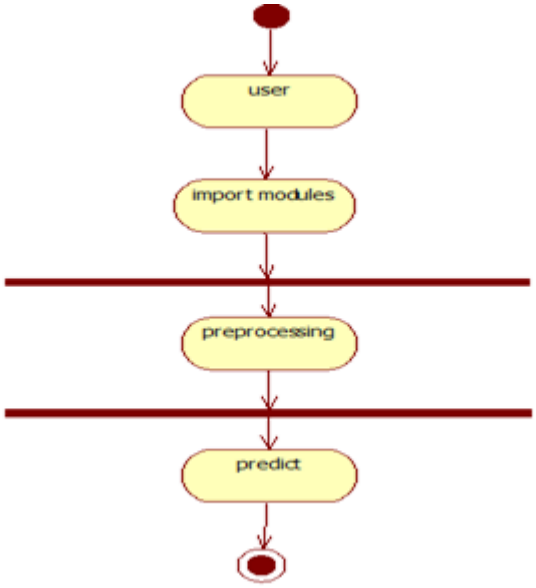


Fig 3.2.3: Activity Diagram

3.3.4 Collaboration Diagram

A collaboration diagram resembles a flowchart that portrays the roles, functionality and behavior of individual objects as well as the overall operation of the system in real time. Objects are shown as rectangles with naming labels inside. These labels are preceded by colons and may be underlined. The relationships between the objects are shown as lines connecting the rectangles. The messages between objects are shown as arrows connecting the relevant rectangles along with labels that define the message sequencing.

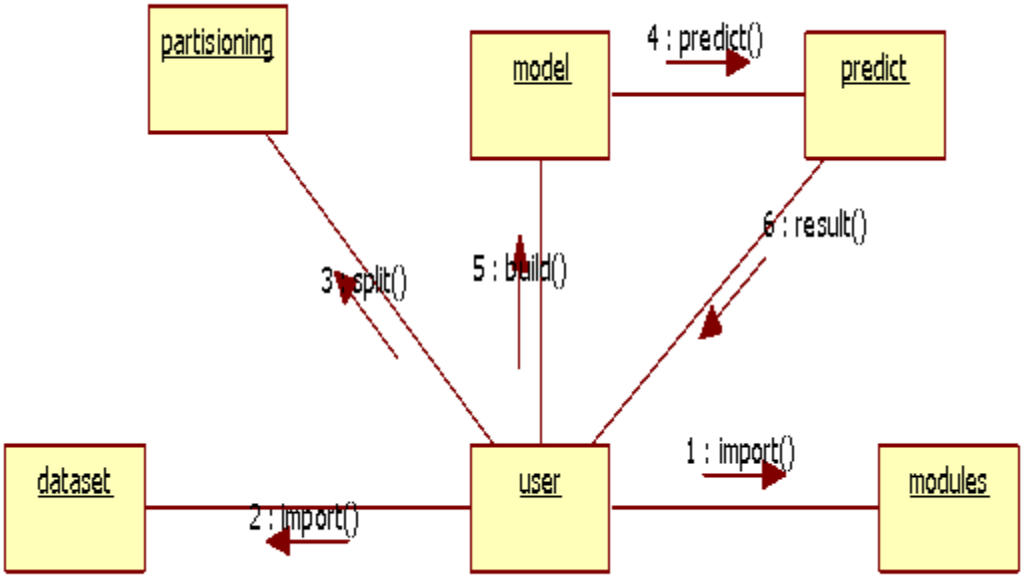
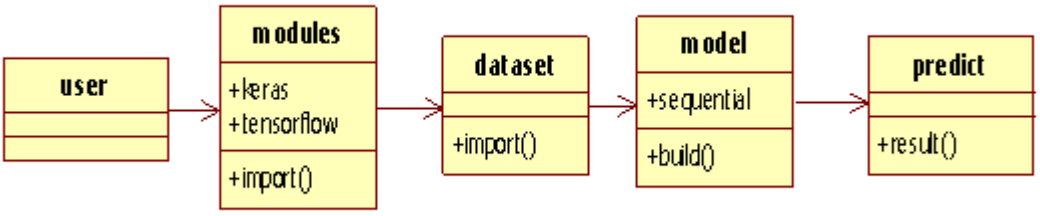


Fig 3.2.4:Collabration Diagram

3.3.5 Class Diagram

The class diagram is the main building block of object-oriented modeling. It is used for general conceptual modeling of the systematic of the application, and for detailed modeling translating the models into programming code. Class diagrams can also be used for data modeling. The classes in a class diagram represent both the main elements, interactions in the application, and the classes to be programmed.



-

Fig 3.2.5: Class Diagram

4. Implementation

Importing the necessary libraries

```
# numpy and pandas for data manipulation
import numpy as np
import pandas as pd

pd.set_option('display.max_rows', 500)
import gc

# sklearn preprocessing
from sklearn.metrics import confusion_matrix, roc_auc_score ,roc_curve,auc
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import classification_report

import lightgbm as lgb

# File system manangement
import os

#eda
!pip install klib
import klib
```

```
# Suppress warnings
import warnings
warnings.filterwarnings('ignore')

# matplotlib and seaborn for plotting
import matplotlib.pyplot as plt
import seaborn as sns


seed = 2357
np.random.seed(seed)
```

Mounting the drive

```
from google.colab import drive
drive.mount('/content/drive')
```

Reading Data

```
# Training data
df_train = pd.read_csv("/content/drive/MyDrive/TrainingWiDS2021.csv")
print('Training data shape: ', df_train.shape)
df_train.head()
```

```
Training data shape:  (130157, 181)
```

	Unnamed: 0	encounter_id	hospital_id	age	bmi	elective_surgery	ethnicity	gender	height	hospital_admit_source	icu_admit_source	i
0	1	214826	118	68.0	22.732803	0	Caucasian	M	180.3	Floor	Floor	
1	2	246060	81	77.0	27.421875	0	Caucasian	F	160.0	Floor	Floor	
2	3	276985	118	25.0	31.952749	0	Caucasian	F	172.7	Emergency Department	Accident & Emergency	
3	4	262220	118	81.0	22.635548	1	Caucasian	F	165.1	Operating Room	Operating Room / Recovery	
4	5	201746	33	19.0	NaN	0	Caucasian	M	188.0	NaN	Accident & Emergency	

5 rows × 181 columns

df_train.drop('Unnamed: 0',axis=1,inplace=True)

print('Training data shape: ', df_train.shape)

df_train.head()

```
Training data shape:  (130157, 180)
```

	encounter_id	hospital_id	age	bmi	elective_surgery	ethnicity	gender	height	hospital_admit_source	icu_admit_source	icu_id
0	214826	118	68.0	22.732803	0	Caucasian	M	180.3	Floor	Floor	92
1	246060	81	77.0	27.421875	0	Caucasian	F	160.0	Floor	Floor	90
2	276985	118	25.0	31.952749	0	Caucasian	F	172.7	Emergency Department	Accident & Emergency	93
3	262220	118	81.0	22.635548	1	Caucasian	F	165.1	Operating Room	Operating Room / Recovery	92
4	201746	33	19.0	NaN	0	Caucasian	M	188.0	NaN	Accident & Emergency	91

5 rows × 180 columns

- The training data has 130157 observations and 180 variables including the TARGET (the label we want to predict).

 # Testing data

df_test = pd.read_csv("/content/drive/MyDrive/UnlabeledWiDS2021.csv")

df_test.drop('Unnamed: 0',axis=1,inplace=True)

print('Testing data shape: ', df_test.shape)

df_test.head()

Testing data shape: (10234, 179)

	encounter_id	hospital_id	age	bmi	elective_surgery	ethnicity	gender	height	hospital_admit_source	icu_admit_source	icu_id	icu_sta
0	144740	10141	72	NaN	0	Caucasian	F	152.4	Floor	Accident & Emergency	82	
1	141990	10141	86	NaN	0	Caucasian	F	175.3	Emergency Department	Accident & Emergency	82	
2	142038	10141	72	NaN	0	Caucasian	F	162.6	Floor	Floor	82	
3	138628	10141	66	NaN	0	Caucasian	M	177.8	Floor	Floor	82	
4	141682	10141	89	NaN	0	Caucasian	M	170.2	Direct Admit	Accident & Emergency	82	

5 rows × 179 columns

- The test set is considerably smaller and lacks a TARGET column.

Exploratory Data Analysis

Column Types

df_train.info(verbose=True, null_counts=True)

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 130157 entries, 0 to 130156
Data columns (total 180 columns):
 #   Column                      Non-Null Count   Dtype
---  ------                      --------------   -----
 0   encounter_id                130157 non-null  int64
 1   hospital_id                 130157 non-null  int64
 2   age                         125169 non-null  float64
 3   bmi                         125667 non-null  float64
 4   elective_surgery            130157 non-null  int64
 5   ethnicity                   128570 non-null  object
 6   gender                      130091 non-null  object
 7   height                      128080 non-null  float64
 8   hospital_admit_source       96959 non-null   object
 9   icu_admit_source            129917 non-null  object
 10  icu_id                      130157 non-null  int64
 11  icu_stay_type               130157 non-null  object
 12  icu_type                    130157 non-null  object
 13  pre_icu_los_days            130157 non-null  float64
 14  readmission_status          130157 non-null  int64
 15  weight                      126694 non-null  float64
 16  albumin_apache              51994 non-null   float64
 17  apache_2_diagnosis          128472 non-null  float64
 18  apache_3j_diagnosis         129292 non-null  float64
 19  apache_post_operative       130157 non-null  int64
 20  arf_apache                  130157 non-null  int64
```

 # Number of each type of column

df_train.dtypes.value_counts()

```
float64    157
int64       17
object       6
dtype: int64
```

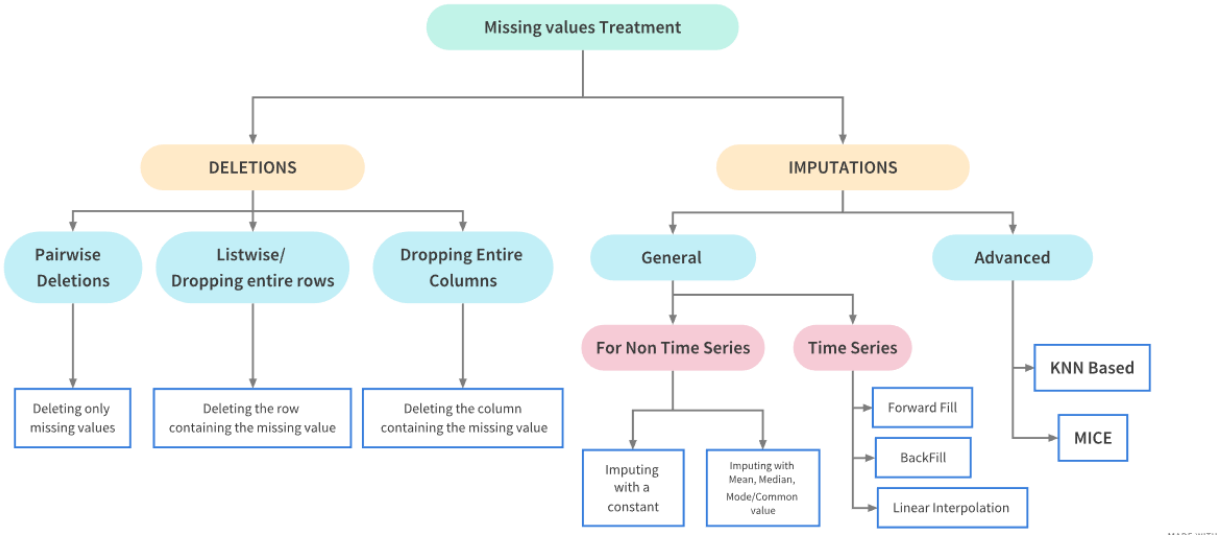
 # Number of unique classes in each object column

df_train.select_dtypes('object').apply(pd.Series.nunique, axis = 0)

```
ethnicity                6
gender                   2
hospital_admit_source    15
icu_admit_source         5
icu_stay_type            3
icu_type                 8
dtype: int64
```

Missing Values

Real world data is messy and often contains a lot of missing values. There could be multiple reasons for the missing values:



```
# Function to calculate missing values by column# Funct

def missing_values_table(df):
    # Total missing values
    mis_val = df.isnull().sum()

    # Percentage of missing values
    mis_val_percent = 100 * df.isnull().sum() / len(df)

    # Make a table with the results
    mis_val_table = pd.concat([mis_val, mis_val_percent], axis=1)

    # Rename the columns
    mis_val_table_ren_columns = mis_val_table.rename(
    columns = {0 : 'Missing Values', 1 : '% of Total Values'})

    # Sort the table by percentage of missing descending
    mis_val_table_ren_columns = mis_val_table_ren_columns[
       mis_val_table_ren_columns.iloc[:,1] != 0].sort_values(
    '% of Total Values', ascending=False).round(1)

    # Print some summary information
    print ("Your selected dataframe has " + str(df.shape[1]) + " columns.\n"
       "There are " + str(mis_val_table_ren_columns.shape[0]) +
         " columns that have missing values.")

    # Return the dataframe with missing information
    return mis_val_table_ren_columns
```

 # Missing values for training data

missing_values_train = missing_values_table(df_train)

missing_values_train[:20].style.background_gradient(cmap='Greens')

	Missing Values	% of Total Values
h1_bilirubin_min	119861	92.100000
h1_bilirubin_max	119861	92.100000
h1_albumin_max	119005	91.400000
h1_albumin_min	119005	91.400000
h1_lactate_max	118467	91.000000
h1_lactate_min	118467	91.000000
h1_pao2fio2ratio_min	113397	87.100000
h1_pao2fio2ratio_max	113397	87.100000
h1_arterial_ph_max	107849	82.900000
h1_arterial_ph_min	107849	82.900000
h1_arterial_pco2_min	107666	82.700000
h1_arterial_pco2_max	107666	82.700000
h1_arterial_po2_max	107445	82.600000
h1_arterial_po2_min	107445	82.600000
h1_hco3_min	106395	81.700000
h1_hco3_max	106395	81.700000
h1_wbc_max	105986	81.400000
h1_wbc_min	105986	81.400000
h1_calcium_min	105921	81.400000
h1_calcium_max	105921	81.400000

 # Missing values for testing data

missing_values_test = missing_values_table(df_test)

missing_values_test[:20].style.background_gradient(cmap='Greens')

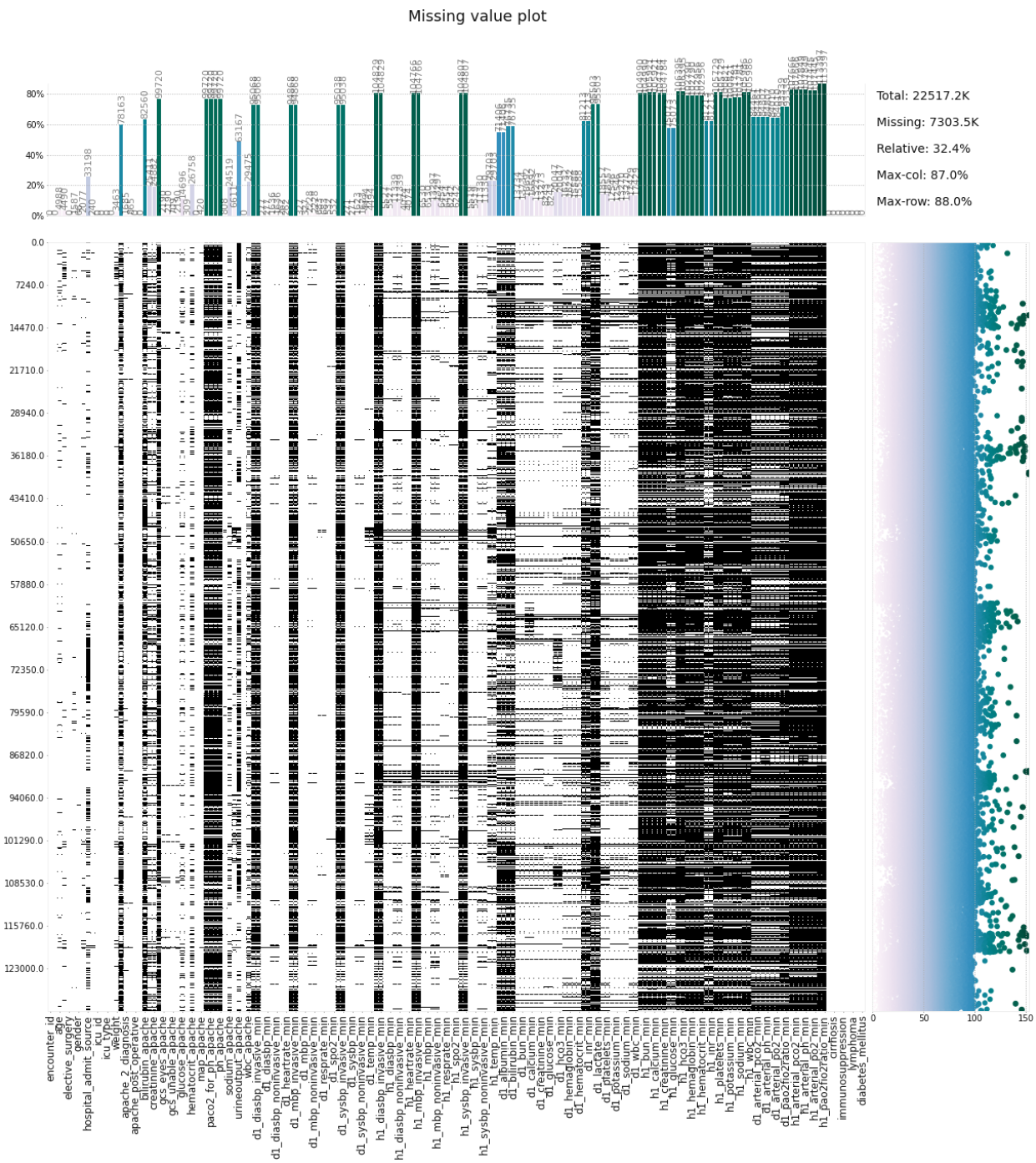
```
Your selected dataframe has 179 columns.
There are 159 columns that have missing values.
```

	Missing Values	% of Total Values
h1_lactate_min	9421	92.100000
h1_lactate_max	9421	92.100000
h1_bilirubin_min	9407	91.900000
h1_bilirubin_max	9407	91.900000
h1_albumin_max	9365	91.500000
h1_albumin_min	9365	91.500000
h1_pao2fio2ratio_min	8812	86.100000
h1_pao2fio2ratio_max	8812	86.100000
h1_arterial_ph_min	8401	82.100000
h1_arterial_ph_max	8401	82.100000
h1_arterial_pco2_min	8347	81.600000
h1_arterial_pco2_max	8347	81.600000
h1_hco3_max	8338	81.500000
h1_hco3_min	8338	81.500000
h1_arterial_po2_min	8306	81.200000
h1_arterial_po2_max	8306	81.200000
h1_calcium_min	8306	81.200000
h1_calcium_max	8306	81.200000
h1_wbc_min	8264	80.800000
h1_wbc_max	8264	80.800000

We can also visualise the missing values instead of looking at the numbers.
Depending upon the percentage of missing values, you can decide to drop the columns which have a very high percentage of missing values and see if that improves the results.Let's drop the empty and single valued columns as well as empty and duplicate rows and visualise the clean dataframe. I'll do it for train but the same needs to be done for the test dataframe as well.

df_cleaned_train = klib.data_cleaning(df_train)

klib.missingval_plot(df_cleaned_train)



The graph above gives a high-level overview of the missing values in a dataset. It pinpoints which columns and rows to examine in more detail.

Top portion of the plot shows the aggregate for each column. Summary statistics is displayed on the right most side.

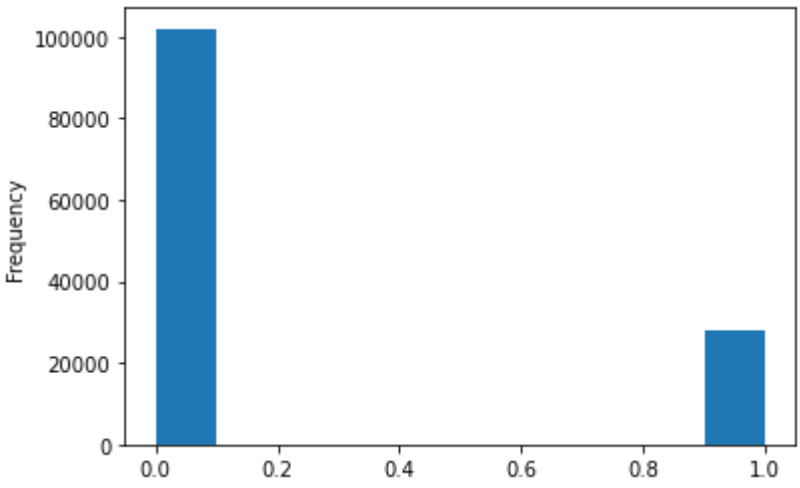
Bottom portion of the plot shows the missing values (black colors) in the DataFrame.

The Target Column

df_train['diabetes_mellitus'].value_counts(normalize=True)

```
0    0.783715
1    0.216285
Name: diabetes_mellitus, dtype: float64
```

df_train['diabetes_mellitus'].astype(int).plot.hist();



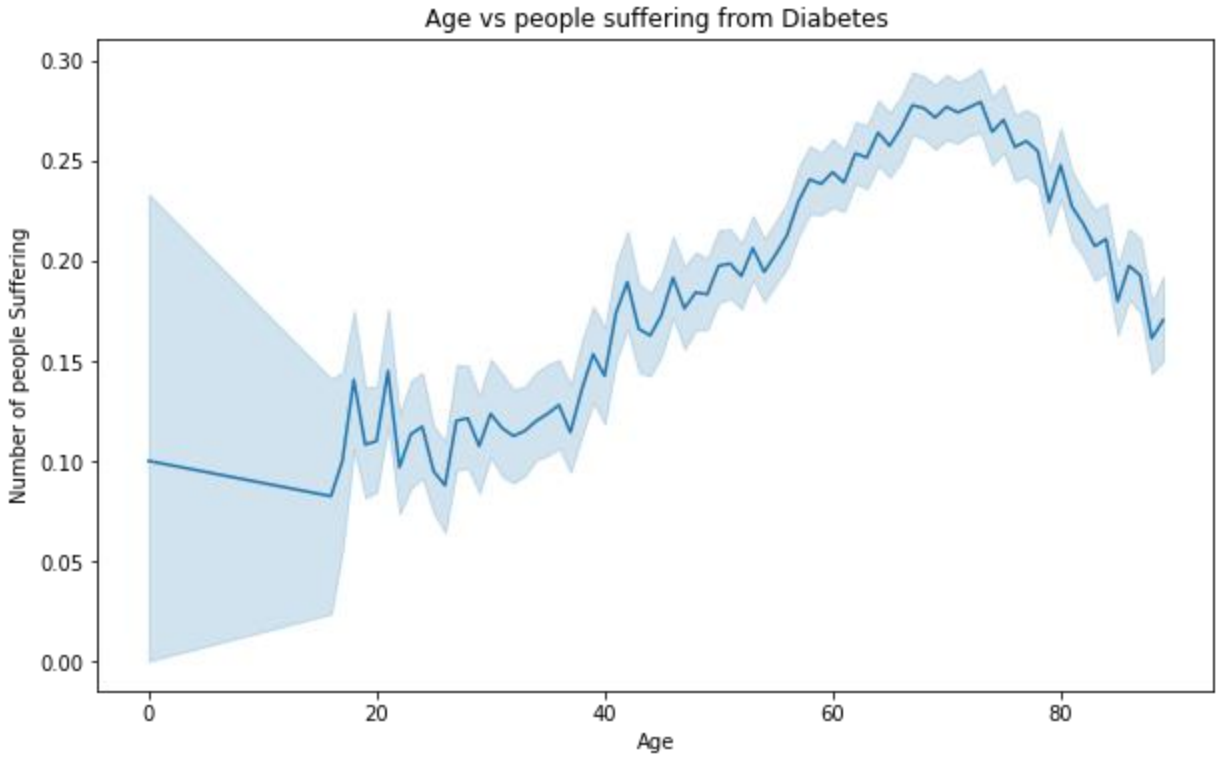
This is an example of imbalanced class problem i.e we where the total number of a class of data (0) is far less than the total number of another class of data (1). Examples include:

- Fraud detection.
- Outlier detection.

plt.figure(figsize=(10,6))

```
plt.title("Age vs people suffering from Diabetes")
sns.lineplot(df_train["age"],df_train["diabetes_mellitus"])
plt.xlabel("Age")
plt.ylabel("Number of people Suffering")
```

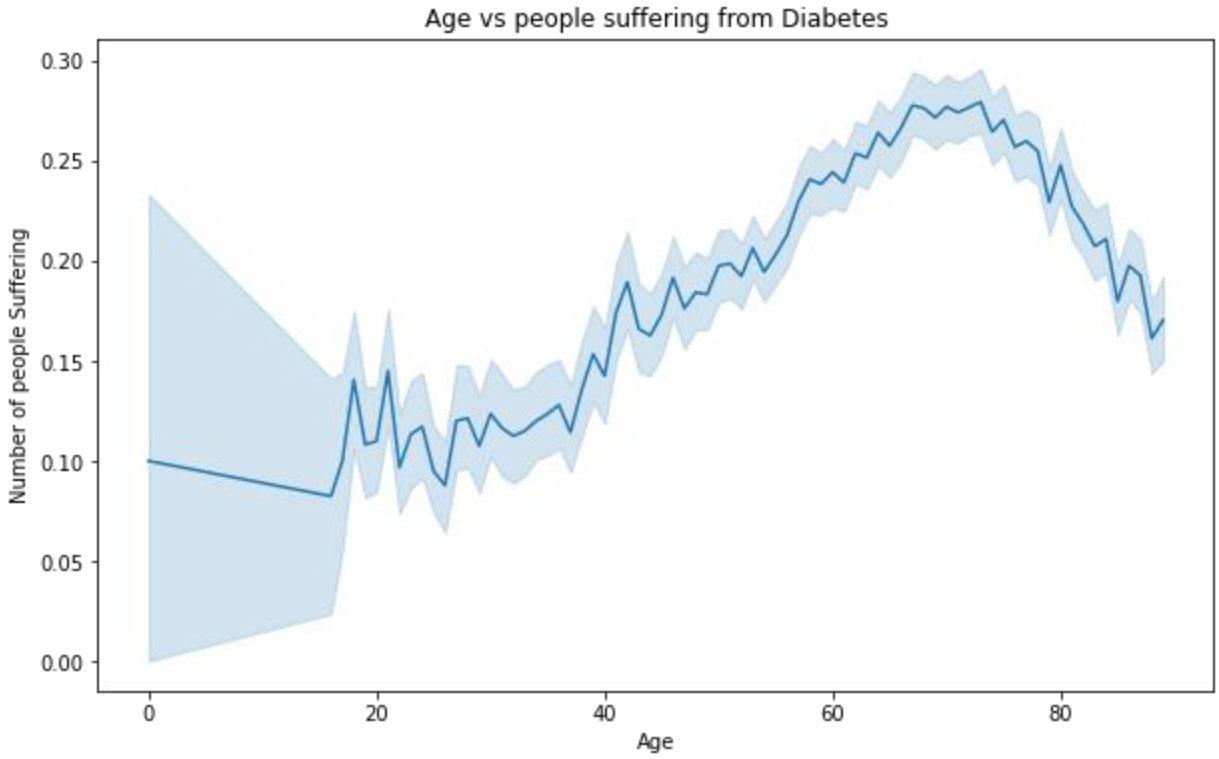
Text(0, 0.5, 'Number of people Suffering')




```
plt.figure(figsize=(10,6))
plt.title("Age vs people suffering from Diabetes")
sns.lineplot(data=df_train, x="age", y="diabetes_mellitus", hue="gender")
plt.xlabel("Age")
plt.ylabel("Number of people Suffering")
```

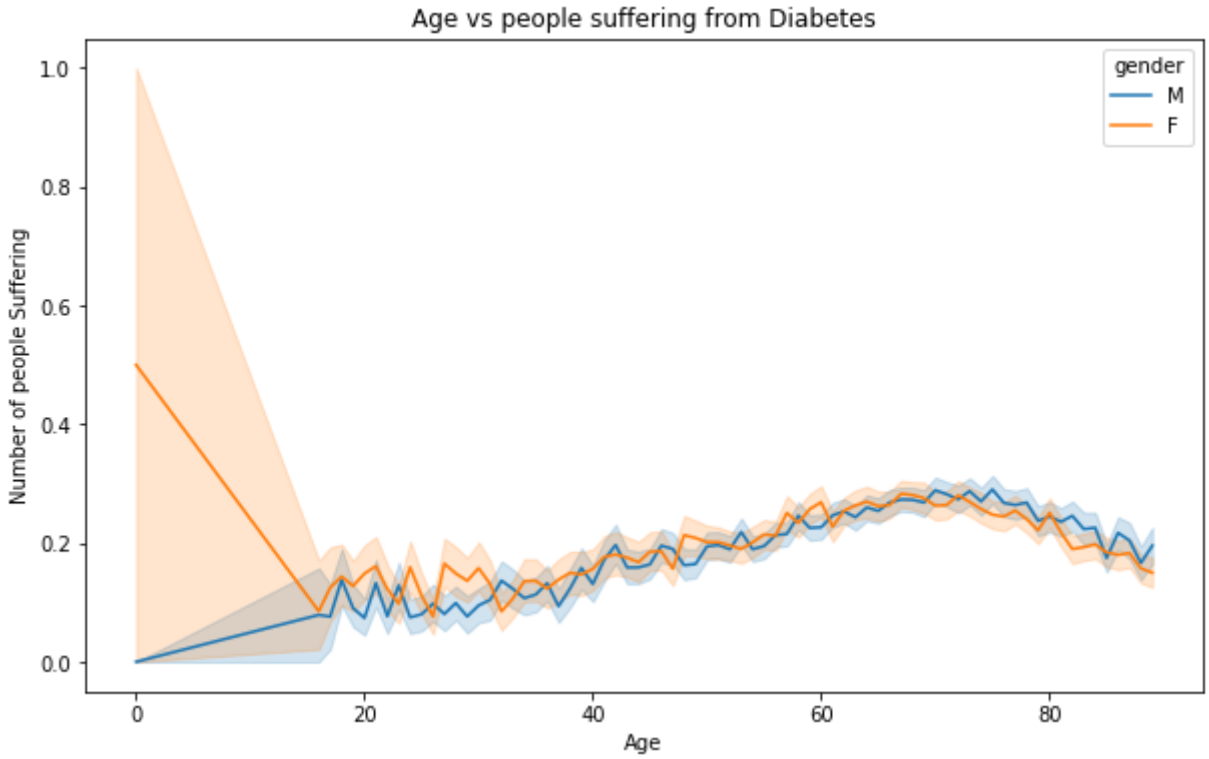
```
Text(0, 0.5, 'Number of people Suffering')
```

Text(0, 0.5, 'Number of people Suffering')



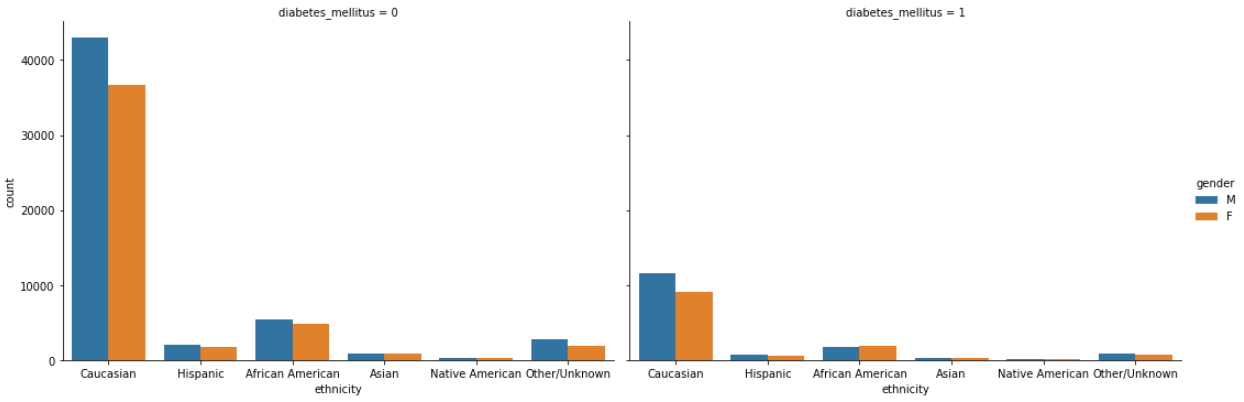

```
plt.figure(figsize=(10,6))
plt.title("Age vs people suffering from Diabetes")
sns.lineplot(data=df_train, x="age", y="diabetes_mellitus", hue="gender")
plt.xlabel("Age")
plt.ylabel("Number of people Suffering")
```

Text(0, 0.5, 'Number of people Suffering')



 # Ethnicity vs Diabetes

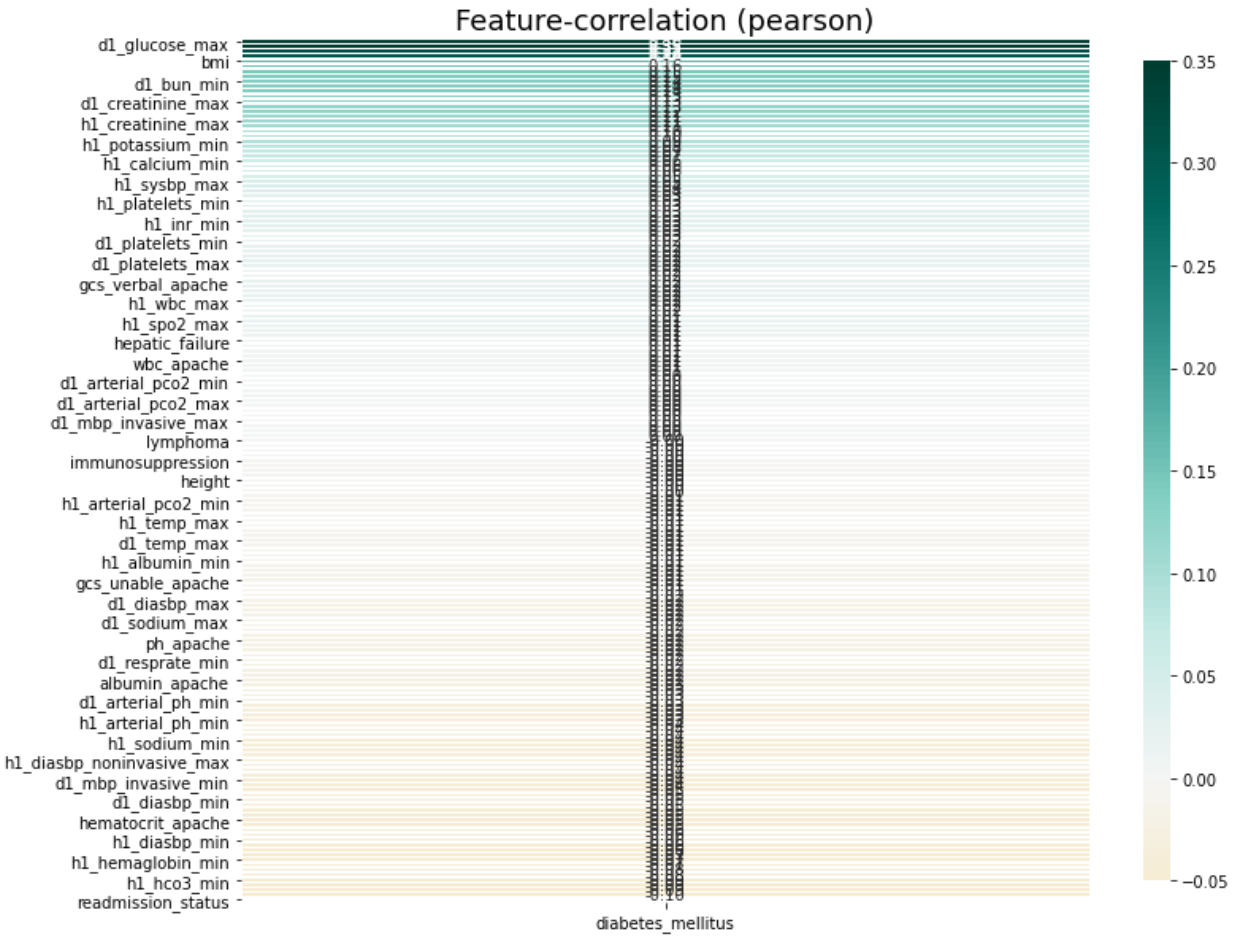
```
g = sns.catplot(x="ethnicity", hue="gender", col="diabetes_mellitus",
                data=df_train, kind="count",
                height=5, aspect=1.5);
```



Correlation Plot

#Display correlation with a target variable of interest

klib.corr_plot(df_train, target='diabetes_mellitus')



Preprocessing Data

combine train and test together to do common feature engineering

train_copy = df_train.copy()

test_copy = df_test.copy()

set up a flag field to distinguish records from training and testing sets in the combined dataset

train_copy['source'] = 0

```
test_copy['source'] = 1

all_data = pd.concat([train_copy, test_copy], axis=0, copy=True)
del train_copy
del test_copy
gc.collect()
```

Output: 87

Dropping unnecessary columns

There were no recurring patient visits, so the encounter_id would not be relevant to our models

```
all_data.drop('encounter_id',axis=1,inplace=True)
```

Checking if there is overlap between the hospitals in the labelled dataset and the hospitals in the unlabelled dataset.

```
df_train['hospital_id'].isin(df_test['hospital_id']).value_counts()
```

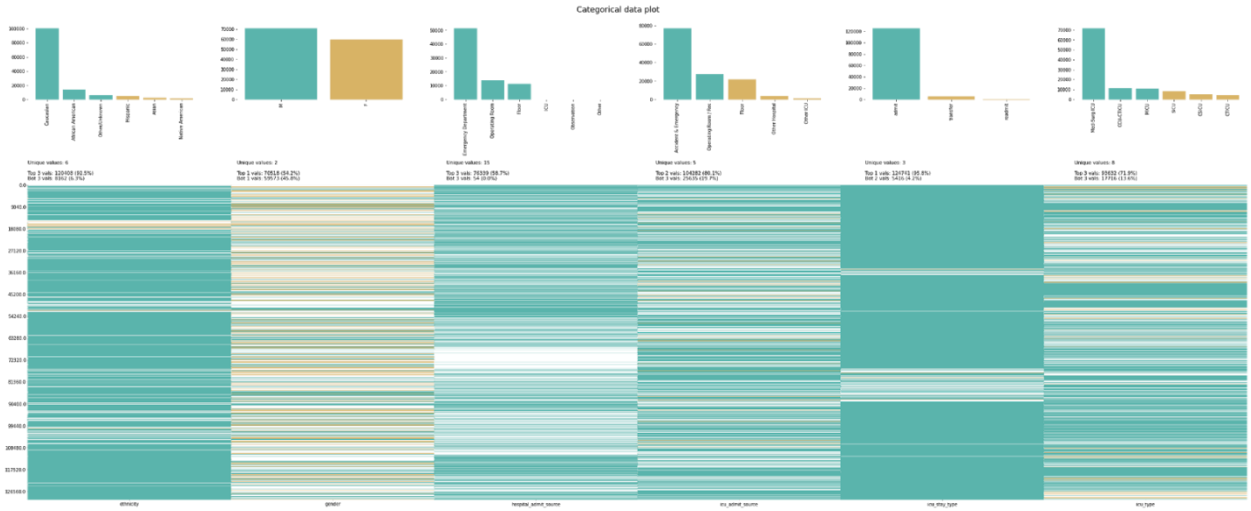
O/P: False 130157
Name: hospital_id, dtype: int64

```
# Dropping hospital id also
all_data.drop('hospital_id',axis=1,inplace=True)
```

Encoding Categorical Variables

Visualising the categorical columns:

klib.cat_plot(df_train, figsize=(50,20))



categorical_columns = all_data.select_dtypes('object').columns

categorical_columns

O/p: Index(['ethnicity', 'gender', 'hospital_admit_source', 'icu_admit_source',
'icu_stay_type', 'icu_type'],
dtype='object')

objList = all_data.select_dtypes(include = "object").columns

print (objList)

Creating a label encoder object

le = LabelEncoder()

for feat in objList:

all_data[feat] = le.fit_transform(all_data[feat].astype(str))

print (all_data.info())

Index(['ethnicity', 'gender', 'hospital_admit_source', 'icu_admit_source',
'icu_stay_type', 'icu_type'],
dtype='object')
<class 'pandas.core.frame.DataFrame'>
Int64Index: 140391 entries, 0 to 10233
Columns: 179 entries, age to source
dtypes: float64(158), int64(21)
memory usage: 192.8 MB
None

all_data[categorical_columns].head()

	ethnicity	gender	hospital_admit_source	icu_admit_source	icu_stay_type	icu_type
0	2	1	4	1	0	2
1	2	0	4	1	0	5
2	2	0	3	0	0	5
3	2	0	8	2	0	2
4	2	1	15	0	0	5

Handling missing values

```
all_data.fillna(-9999, inplace=True)
all_data.isnull().sum()
```

```
age                          0
bmi                          0
elective_surgery             0
ethnicity                    0
gender                       0
height                       0
hospital_admit_source        0
icu_admit_source             0
icu_id                       0
icu_stay_type                0
icu_type                     0
pre_icu_los_days             0
readmission_status           0
weight                       0
albumin_apache               0
apache_2_diagnosis           0
apache_3j_diagnosis          0
apache_post_operative        0
arf_apache                   0
```

---

split the all-data DF into training and testing again

```
# split the all-data DF into training and testing again
training = all_data[all_data['source']==0]
testing = all_data[all_data['source']==1]

del all_data
gc.collect()
```

O/P: 172

```
print(training.shape)
print(testing.shape)
```

Output: (130157, 179)

(10234, 179)

```
testing.drop('diabetes_mellitus',axis=1,inplace=True)
print(testing.shape)
```

Output : (10234, 178)

Baseline

1. Logistic Regression

```
TARGET = 'diabetes_mellitus'
train_labels = training[TARGET]
train = training.drop(columns = [TARGET,'source'])
features = list(train.columns)
test = testing.drop(columns = ['source'])
print('Training data shape: ', train.shape)
print('Testing data shape: ', test.shape)
```

Output:

```
Training data shape:  (130157, 177)
Testing data shape:  (10234, 177)
```

```
log_reg = LogisticRegression(C = 0.0001)
log_reg.fit(train, train_labels)
```

Output : log_reg_pred = log_reg.predict_proba(test)[:, 1]

```
log_reg_pred
```

Output:

```
array([0.33043761, 0.26749958, 0.32892825, ..., 0.05827944, 0.05959594,
       0.02618623])
```

The predictions must be in the format shown in the sample_submission.csv file, where there are only two columns: encounter_id and diabetes_mellitus are present. We will create a dataframe in this format from the test set and the predictions called submit.

```
submit = df_test[['encounter_id']]
submit['diabetes_mellitus'] = log_reg_pred
submit.to_csv('logreg_baseline.csv',index=False)
submit.head()
```

	encounter_id	diabetes_mellitus
0	144740	0.330438
1	141990	0.267500
2	142038	0.328928
3	138628	0.095155
4	141682	0.323641

2. Random Forest

```
rf = RandomForestClassifier(n_estimators = 100, random_state = 50, verbose = 1, n_jobs = -1)
rf.fit(train, train_labels)

feature_importance_values = rf.feature_importances_
feature_importances = pd.DataFrame({'feature': features, 'importance': feature_importance_values})

predictions = rf.predict_proba(test)[:, 1]
```

Output: [Parallel(n_jobs=-1)]: Using backend ThreadingBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done 46 tasks | elapsed: 30.7s

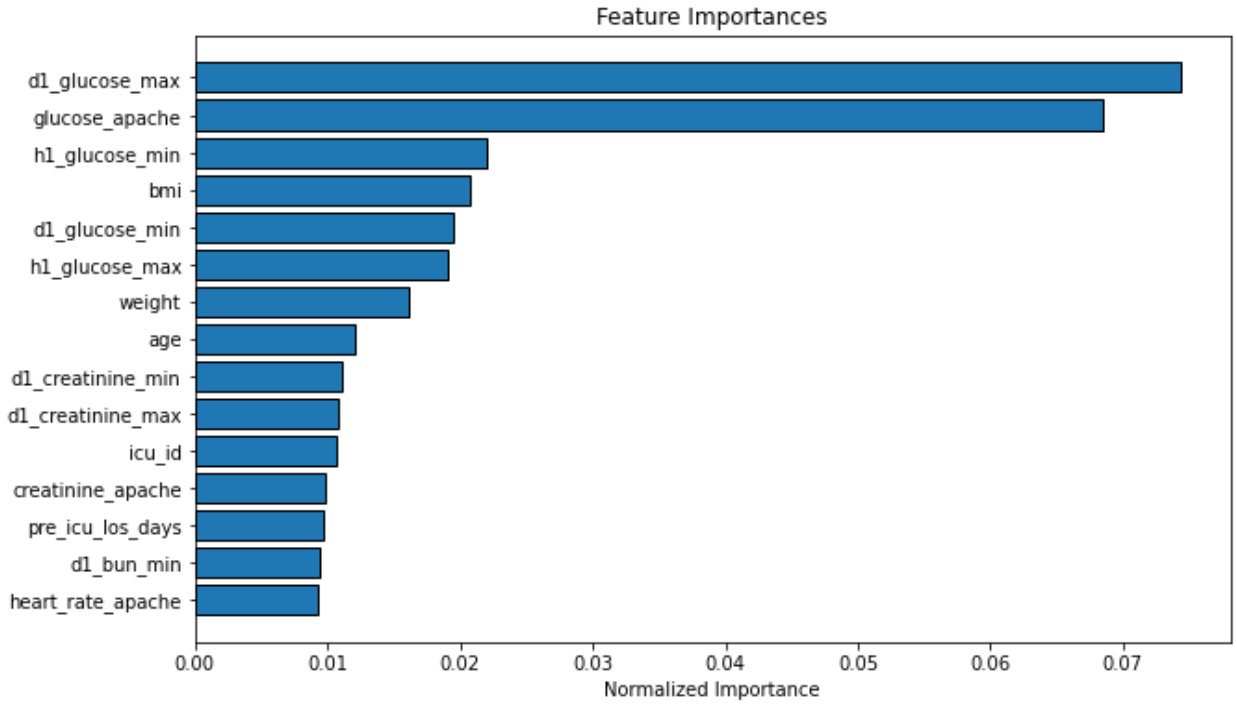
```
[Parallel(n_jobs=-1)]: Done 100 out of 100 | elapsed:  1.1min finished
[Parallel(n_jobs=2)]: Using backend ThreadingBackend with 2 concurrent workers.
[Parallel(n_jobs=2)]: Done  46 tasks      | elapsed:    0.1s
[Parallel(n_jobs=2)]: Done 100 out of 100 | elapsed:    0.3s finished

submit = df_test[['encounter_id']]
submit['diabetes_mellitus'] = predictions

submit.to_csv('random_forest_baseline_domain.csv', index = False)
```

Feature Engineering

Depending on the model, your dataset may contain too many features for it to handle.Hence you can eliminate the ones which donot appear very high on the feature importance leaderboard.

```
def plot_feature_importances(df):
    df = df.sort_values('importance', ascending = False).reset_index()

    # Normalize the feature importances to add up to one
    df['importance_normalized'] = df['importance'] / df['importance'].sum()

    # Make a horizontal bar chart of feature importances
    plt.figure(figsize = (10, 6))
    ax = plt.subplot()

    # Need to reverse the index to plot most important on top
    ax.barh(list(reversed(list(df.index[:15]))),
            df['importance_normalized'].head(15),
            align = 'center', edgecolor = 'k')

    # Set the yticks and labels
```

```
    ax.set_yticks(list(reversed(list(df.index[:15]))))
    ax.set_yticklabels(df['feature'].head(15))

    # Plot labeling
    plt.xlabel('Normalized Importance'); plt.title('Feature Importances')
    plt.show()

    return df

# Show the feature importances for the default features
feature_importances_sorted = plot_feature_importances(feature_importances)
```




```
numerical_columns =
train.columns[~train.columns.isin(categorical_columns)]
```

Evaluating the model

We have created our model but how will we know how accurate it is? We do have a Test dataset but since it doesn't have the Target column, everytime we optimize our model, we will have to submit our predictions to public Leaderboard to assess it accuracy.

Creating a Validation set

Another option would be to create a validation set from the training set. We will hold out a part of the training set during the start of the experiment and use it for evaluating our predictions. We shall use the scikit-learn library's model_selection.train_test_split() function that we can use to split our data

Logistic Regression

```
[ ]  # Logistic Regression

     lr = LogisticRegression()
     lr.fit(train_X, train_y)
     predictions2 = lr.predict_proba(val_X)[:,1]
     roc_auc_score(val_y, predictions2)
```

```
0.6494764066051497
```

Random Forest

```
[ ] # RF

    rf.fit(train_X, train_y)
    predictions3 = rf.predict_proba(val_X)[:,1]
    roc_auc_score(val_y, predictions3)
```

```
[Parallel(n_jobs=-1)]: Using backend ThreadingBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done  46 tasks      | elapsed:   24.0s
[Parallel(n_jobs=-1)]: Done 100 out of 100 | elapsed:   52.1s finished
[Parallel(n_jobs=2)]: Using backend ThreadingBackend with 2 concurrent workers.
[Parallel(n_jobs=2)]: Done  46 tasks      | elapsed:    0.3s
[Parallel(n_jobs=2)]: Done 100 out of 100 | elapsed:    0.7s finished
0.8290978399203652
```

Light GBM

```
[ ] # Light GBM

    d_train=lgb.Dataset(train_X, label=train_y)

    #Specifying the parameter
    params={}
    params['learning_rate']=0.03
    params['boosting_type']='gbdt' #GradientBoostingDecisionTree
    params['objective']='binary' #Binary target feature
    params['metric']='binary_logloss' #metric for binary classification
    params['max_depth']=10

    #train the model
    clf=lgb.train(params,d_train,100) #train the model on 100 epocs

    predictions_lgb = clf.predict(val_X)
    roc_auc_score(val_y, predictions_lgb)
```

```
0.8449149309317078
```

XGBoost

```
[ ] from xgboost import XGBClassifier
    import xgboost as xgb

    model_XGB = XGBClassifier(use_label_encoder=False, eval_metric="auc")
    model_XGB.fit(train_X, train_y)
    print("XGBClassifier : On validation set, the ROC AUC score is ",roc_auc_score(val_y, model_XGB.predict_proba(val_X)[:,1]))

    XGBClassifier : On validation set, the ROC AUC score is  0.8486440311385709
```

Cross Validation

XGBoost with Cross Validation

scores_xgb = cross_val_score(model_XGB, X, y, cv=10, scoring='roc_auc')

scores_xgb.sort()

print('Mean Absolute Score %2f' %(scores_xgb.mean()))

Output: Mean Absolute Score 0.824544

Logistic Regression with Cross Validation

scores_log = cross_val_score(lr, X, y, cv=10, scoring='roc_auc')

scores_log.sort()

print('Mean Absolute Score %2f' %(scores_log.mean()))

Output: Mean Absolute Score 0.643712

Random Forest with Cross Validation

---

scores_rf = cross_val_score(rf, X, y, cv=10, scoring='roc_auc')

scores_rf.sort()

print('Mean Absolute Score %2f' %(scores_rf.mean()))

Output: Mean Absolute Score 0.822345

Stratified K-Fold

Random Forest with Stratified K-Fold

```
kf = StratifiedKFold(n_splits=3,shuffle=True,random_state=seed)
pred_test_full = 0
cv_score =[]
i=1
for train_index,test_index in kf.split(X,y):
  print('{} of KFold {}'.format(i,kf.n_splits))
  xtr,xvl = X.loc[train_index],X.loc[test_index]
  ytr,yvl = y[train_index],y[test_index]

  #model
  lr = RandomForestClassifier(n_estimators = 100, random_state = 50, verbose = 1, n_jobs = -1)
  #lr = LogisticRegression(C = 0.0001)
  lr.fit(xtr,ytr)
  score = roc_auc_score(yvl,lr.predict_proba(xvl)[:,1])
  print('ROC AUC score:',score)
  cv_score.append(score)
  pred_test = lr.predict_proba(test)[:,1]
  pred_test_full +=pred_test
  i+=1


#RandomForest
print('Confusion matrix\n',confusion_matrix(yvl,lr.predict(xvl)))
print('Cv',cv_score,'\nMean cv Score',np.mean(cv_score))
```

```
[Parallel(n_jobs=2)]: Using backend ThreadingBackend with 2 concurrent workers.
[Parallel(n_jobs=2)]: Done  46 tasks      | elapsed:    0.6s
Confusion matrix
 [[32370  1632]
 [ 6257  3126]]
Cv [0.8284436988135645, 0.8256516071493312, 0.8228257419617655]
Mean cv Score 0.8256403493082205
[Parallel(n_jobs=2)]: Done 100 out of 100 | elapsed:    1.2s finished
```

Light GBM with Stratified K-Fold

```
kf = StratifiedKFold(n_splits=3,shuffle=True,random_state=seed)
pred_test_full = 0
cv_score =[]
i=1
for train_index,test_index in kf.split(X,y):
  print('{} of KFold {}'.format(i,kf.n_splits))
  xtr,xvl = X.loc[train_index],X.loc[test_index]
  ytr,yvl = y[train_index],y[test_index]

  d_train=lgb.Dataset(xtr, label=ytr)

  params = {
    'boosting_type': 'gbdt',
    'objective': 'binary',
    'metric': 'auc',
    'learning_rate': 0.01,
    'subsample': 1,
    'colsample_bytree': 0.2,
    'reg_alpha': 3,
    'reg_lambda': 1,
    'scale_pos_weight': 1,
    'n_estimators': 5404,
```

```
    'silent': -1,
    'verbose': -1,
    'max_depth': -1,
    'seed':i + 666,
}


#train the model
clf=lgb.train(params,d_train,100)

 score = roc_auc_score(yvl,clf.predict(xvl))
print('ROC AUC score:',score)
cv_score.append(score)
pred_test = clf.predict(test)
pred_test_full +=pred_test
i+=1


#LGBM
print('Cv',cv_score,'\nMean cv Score',np.mean(cv_score))
```

Output:

Cv [0.8738445874228451, 0.871478480554752, 0.8697095561762789]

Mean cv Score 0.8716775413846253

XGBoost with Stratified K-Fold

```
kf = StratifiedKFold(n_splits=3,shuffle=True,random_state=seed)
pred_test_full = 0
cv_score =[]
```

```
i=1
for train_index,test_index in kf.split(X,y):
  print('{} of KFold {}'.format(i,kf.n_splits))
  xtr,xvl = X.loc[train_index],X.loc[test_index]
  ytr,yvl = y[train_index],y[test_index]

  XGB = XGBClassifier(use_label_encoder=False, eval_metric="auc")
  XGB.fit(xtr, ytr)

  score = roc_auc_score(yvl,XGB.predict_proba(xvl)[:,1])
  print('ROC AUC score:',score)
  cv_score.append(score)
  pred_test = XGB.predict_proba(test)[:,1]
  pred_test_full +=pred_test
  i+=1
print('Cv',cv_score,'\nMean cv Score',np.mean(cv_score))
```

Output: Cv [0.8494461445475375, 0.8486643936069556, 0.8460259605194153]
Mean cv Score 0.8480454995579695

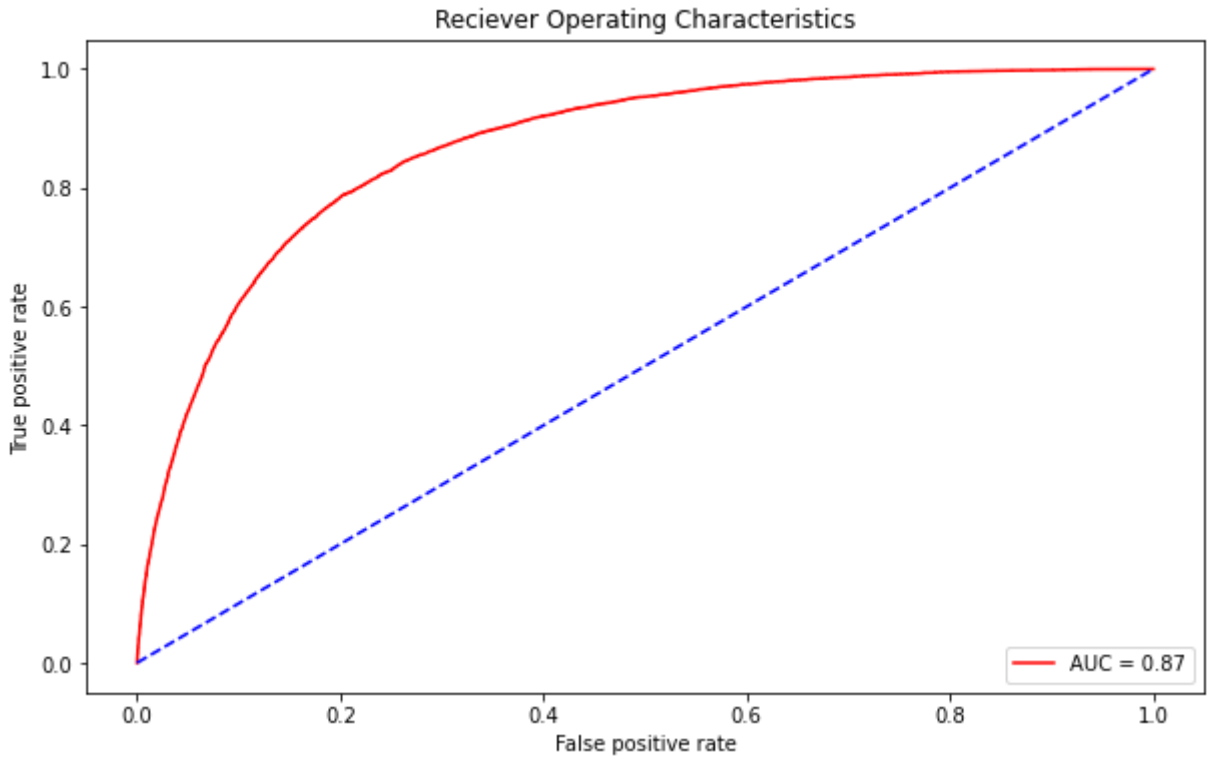
Reciever Operating Characteristics

```
  proba = clf.predict(xvl)
  frp,trp, threshold = roc_curve(yvl,proba)
  roc_auc_ = auc(frp,trp)

  plt.figure(figsize=(10,6))
  plt.title('Reciever Operating Characteristics')
  plt.plot(frp,trp,'r',label = 'AUC = %0.2f' % roc_auc_)
  plt.legend(loc='lower right')
  plt.plot([0,1],[0,1],'b--')
```

plt.ylabel('True positive rate')

plt.xlabel('False positive rate')

Output: Text(0.5, 0, 'False positive rate')



Testing

4.2 Testing

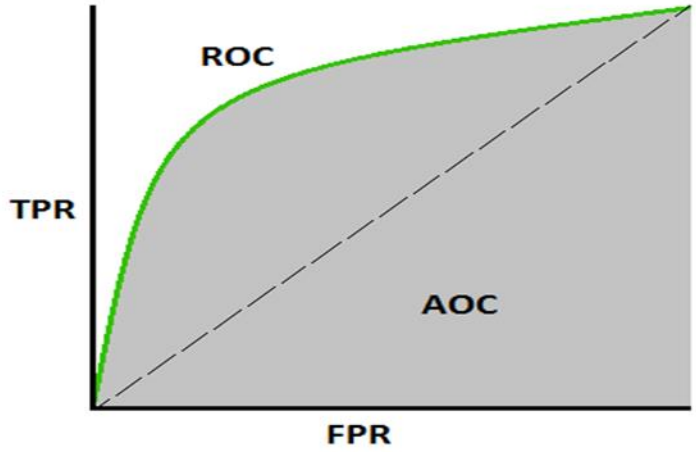
In Machine Learning, performance measurement is an essential task. So when it comes to a classification problem, we can count on an AUC - ROC Curve. When we need to check or visualize the performance of the multi-class classification problem, we use the AUC (Area Under The Curve) ROC (Receiver Operating Characteristics) curve. It is one of the most important evaluation metrics for checking any classification model's performance. It is also written as AUROC (Area Under the Receiver Operating Characteristics)

Why AUC and ROC?

It measures the quality of the model's predictions irrespective of what classification threshold is chosen. AUC - ROC curve is a performance measurement for the classification problems at various threshold settings.

AUC - ROC curve is a performance measurement for the classification problems at various threshold settings. ROC is a probability curve and AUC represents the degree or measure of separability. It tells how much the model is capable of distinguishing between classes. Higher the AUC, the better the model is at predicting 0s as 0s and 1s as 1s. By analogy, the Higher the AUC, the better the model is at distinguishing between patients with the disease and no disease.

The ROC curve is plotted with TPR against the FPR where TPR is on the y-axis and FPR is on the x-axis.



AUC - ROC Curve

Defining terms used in AUC and ROC Curve.

- TPR (True Positive Rate) / Recall /Sensitivity: it is defined as follows

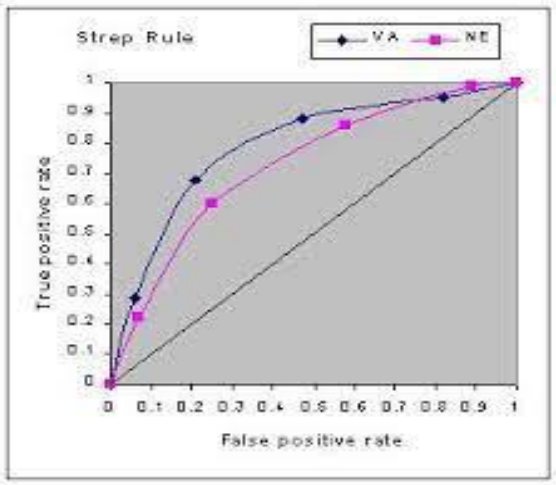
  TPR = TP/TP+FN

- False Positive Rate(FPR): it is defined as follows

FPR = FP/FP+TN

An ROC curve plots TPR vs. FPR at different classification thresholds. Lowering the classification threshold classifies more items as positive, thus increasing both False Positives and True Positives. The following figure shows a typical ROC curve.

Figure 6. Predictions ranked in ascending order of logistic regression score.



AUC represents the probability that a random positive (green) example is positioned to the right of a random negative (red) example.

AUC ranges in value from 0 to 1. A model whose predictions are 100% wrong has an AUC of 0.0; one whose predictions are 100% correct has an AUC of 1.0.

AUC is desirable for the following two reasons:

- AUC is scale-invariant. It measures how well predictions are ranked, rather than their absolute values.
- AUC is classification-threshold-invariant. It measures the quality of the model's predictions irrespective of what classification threshold is chosen.
- However, both these reasons come with caveats, which may limit the usefulness of AUC in certain use cases:

- Scale invariance is not always desirable. For example, sometimes we really do need well calibrated probability outputs, and AUC won't tell us about that.
- Classification-threshold invariance is not always desirable. In cases where there are wide disparities in the cost of false negatives vs. false positives, it may be critical to minimize one type of classification error. For example, when doing email spam detection, you likely want to prioritize minimizing false positives (even if that results in a significant increase of false negatives). AUC isn't a useful metric for this type of optimization.
- ROC curves are frequently used to show in a graphical way the connection/trade-off between clinical sensitivity and specificity for every possible cut-off for a test or a combination of tests. In addition the area under the ROC curve gives an idea about the benefit of using the test(s) in question.
- ROC curves are used in clinical biochemistry to choose the most appropriate cut-off for a test. The best cut-off has the highest true positive rate together with the lowest false positive rate.
- AUC (based on ROC) and overall accuracy seems not the same concept.

Overall accuracy is based on one specific cutpoint, while ROC tries all of the cutpoint and plots the sensitivity and specificity. So when we compare the overall accuracy, we are comparing the accuracy based on some cutpoint. The overall accuracy varies from different cutpoint.

4.2.1 Testing Strategies

Testing is a set of activities that can be planned in advanced and conducted systematically. A strategy for software testing must accommodation low-level tests that are necessary to verify that a small source code segment has been correctly implemented as well as high-level tests that validate major system functions against customer requirements.

Software testing is one element of verification and validation. Verification refers to the set of activities that ensure that software correctly implements as specific function. Validation refers to a different set of activities that ensure that the software that has been built is traceable to customer requirements.

The main objective of software is testing to uncover errors. To fulfill this objective, a series of test steps unit, integration, validation and system tests are planned and executed. Each test step is accomplished through a series of systematic test technique that assist in the design of test cases. With each testing step, the level of abstraction with which software is considered is broadened.

Testing is the only way to assure the quality of software and it is an umbrella activity rather than a separate phase. This is an activity to be performed in parallel with the software effort and one that consists of its own phases of analysis, design, implementation, execution and maintenance.

UNIT TESTING:

This testing method considers a module as single unit and checks the unit at interfaces and communicates with other modules rather than getting into details at statement level. Here the module will be treated as a black box, which will take some input and generate output. Outputs for a given set of input combination are pre-calculated and are generated by the module.

SYSTEM TESTING:

Here all the pre tested individual modules will be assembled to create the larger system and tests are carried out at system level to make sure that all modules are working in synchronous with each other. This testing methodology helps in making sure that all modules which are running perfectly when checked individually are also running in cohesion with other modules. For this testing we create test cases to check all modules once and then generated test combinations of test paths through out the system to make sure that no path is making its way into chaos.

INTEGRATED TESTING

Testing is a major quality control measure employed during software development. Its basic function is to detect errors. Sub functions when combined may not produce than it is desired. Global data structures can represent the problems. Integrated testing is a systematic technique for constructing the program structure while conducting the tests. To uncover errors that are associated with interfacing the objective is to make unit test modules and built a program structure that has been detected by design. In a non - incremental integration all the modules are combined in advance and the program is tested as a whole. Here errors will appear in an end less loop function. In incremental testing the program is constructed and tested in small segments where the errors are isolated and corrected.

Different incremental integration strategies are top – down integration, bottom – up integration, regression testing.

REGRESSION TESTING

Each time a new module is added as a part of integration as the software changes. Regression testing is an actually that helps to ensure changes that do not introduce unintended behavior as additional errors.

Regression testing maybe conducted manually by executing a subset of all test cases or using automated capture play back tools enables the software engineer to capture the test case and results for subsequent playback and compression. The regression suit contains different classes of test cases.

A representative sample to tests that will exercise all software functions.

Additional tests that focus on software functions that are likely to be affected by the change.

4. CONCLUSION AND FUTURE SCOPE

These application helps doctors to identify whether the patient having Diabetes or not. The main theme of our project is to improve a patient's outcome in an ICU knowledge about their medical conditions can improve clinical decisions. Using a public dataset that the information about chronic diseases based on data that can be gathered within the first 24 hours of being admitted to an ICU. We train a model by using LightGBM algorithm.

The future enhancement of this application is

• To increase the accuracy.

References

- Analysis of diabetes mellitus for early prediction using optimal features selection - https://journalofbigdata.springeropen.com/articles/10.1186/s40537-019-0175-6

- Prediction of Diabetes Using Machine Learning Algorithms in Healthcare- https://ieeexplore.ieee.org/document/8748992

- Predictive models for diabetes mellitus using machine learning techniques - https://bmcendocrdisord.biomedcentral.com/articles/10.1186/s12902-019-0436-6