A Major-Project Report

on

Model for Diabetes Prediction

Submitted in partial fulfillment of the requirements

for the award of degree of

BACHELOR OF TECHNOLOGY

in

Information Technology

by

A.Satya Anusha(18WH1A1201)

M.Harini (18WH1A1233)

D.Harshitha (18WH1A1253)

K.Divya (18WH1A1258)

Under the esteemed guidance of

Dr.P.Kayal

Associate Professor

VISHNU
UNIVERSAL LEARNING

Department of Information Technology

BVRIT HYDERABAD College of Engineering for Women

Rajiv Gandhi Nagar, Nizampet Road, Bachupally, Hyderabad – 500090

(Affiliated to Jawaharlal Nehru Technological University Hyderabad)

(NAAC 'A' Grade & NBA Accredited- ECE, EEE, CSE IT)

June, 2022

DECLARATION

We hereby declare that the work presented in this project entitled “ **Model for Diabetes Prediction**” submitted towards completion of the major project in IV year II sem of B.Tech IT at “BVRIT HYDERABAD College of Engineering for Women”, Hyderabad is an authentic record of our original work carried out under the esteem guidance of **Dr.P.Kayal,Associate Professor** , Department of IT.

A.Satya Anusha (18WH1A1201)

M.Harini (18WH1A1233)

D.Harshitha (18WH1A1253)

K.Divya (18WH1A1258)

BVRIT HYDERABAD

College of Engineering for Women

Rajiv Gandhi Nagar, Nizampet Road, Bachupally, Hyderabad – 500090

(Affiliated to Jawaharlal Nehru Technological University Hyderabad)

(NAAC 'A' Grade & NBA Accredited- ECE, EEE, CSE IT)

CERTIFICATE

This is to certify that the Major-project report on **"Model for Diabetes Prediction"** is a bonafide work carried out by **A.Satya Anusha (18WH1A1201), M.Harini (18WH1A1233), D.Harshitha (18WH1A1253)** and **K.Divya (18WH1A1258)** in the partial fulfillment for the award of B.Tech degree in **Information Technology, BVRIT HYDERABAD College of Engineering for Women, Bachupally, Hyderabad** affiliated to Jawaharlal Nehru Technological University, Hyderabad under my guidance and supervision.

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

Internal Guide	**Head of the Department**
Dr.P.Kayal	**Dr. Aruna Rao S L**
Associate Professor	**Professor & HoD**
Department of IT	**Department of IT**

<u>ACKNOWLEDGEMENT</u>

We would like to express our profound gratitude and thanks to **Dr. K. V. N. Sunitha, Principal, BVRIT HYDERABAD** for providing the working facilities in the college.

Our sincere thanks and gratitude to **Dr. Aruna Rao S L, Professor & Head, Department of IT, BVRIT HYDERABAD** for all the timely support, constant guidance and valuable suggestions during the period of our project.

We are extremely thankful and indebted to our internal guide, **Dr.P.Kayal, Associate Professor, Department of IT, BVRIT HYDERABAD** for her constant guidance, encouragement and moral support throughout the project.

Finally, we would also like to thank our Project Coordinators **Dr.P.S.Latha Kalyampudi, Associate Professor, Dr.P.Kayal, Associate Professor**, all the faculty and staff of the IT Department who helped us directly or indirectly, parents and friends for their cooperation in completing the project work.

A.Satya Anusha (18WH1A1201)

M.Harini (18WH1A1233)

D.Harshitha (18WH1A1253)

K.Divya (18WH1A1258)

ABSTRACT

Diabetes mellitus is a chronic disease characterized by hyperglycemia. It may cause many complications. According to the growing morbidity in recent years, in 2040, the world's diabetic patients will reach 642 million, which means that one of the ten adults in the future is suffering from diabetes. There is no doubt that this alarming figure needs great attention. With the rapid development of the latest technology , many aspects of medical health can be improved . In this research, we focused on models to determine whether a patient admitted to an ICU has been diagnosed with a particular type of diabetes, Diabetes mellitus. The dataset contains 130,157 patients data, where 70 percent are used for training and 30 percent are used for testing. Since the dataset has 371 attributes, we used feature engineering to optimize the data and then implemented various Machine Learning algorithms like XGB, Random forest and LightGBM to predict diabetes mellitus which resulted in AUC score 0.86. For improving results we have applied Hyper Parameter tuning which resulted in 0.872 AUC score.

LIST OF FIGURES

LIST OF ABBREVIATIONS

Abbreviation	**Meaning**
ICU	Intensive Care Unit
ML	Machine Learning
GBM	Gradient Boosting Machine
SVM	Support Vector Machine
T1D	Type 1 Diabetes
T2D	Type 2 Diabetes
DT	Decision Tree
RF	Random Forest
PCA	Principal Component Analysis
ACC	Accuracy
PIDD	Pima Indian Diabetes Database
UCI	UC Irvine
DLPD	Deep Learning for Predicting Diabetes
NAN	Not A Number
EDA	Exploratory Data Analysis
KNN	K Nearest Neighbour
BG	Blood Glucose
MLP	Multi Layer Perceptron
LR	Logistic Regression
LSTM	Long Short-Term Memory
MA	Moving Averages
LR	Linear Regression
SNPG37	Single Nucleotide Polymorphism Gene 37

Abbreviation	**Meaning**
GBDT	Gradient Boosted Decision Tree
EFB	Exclusive Feature Bundling
GOSS	Gradient-based One Side Sampling

CONTENTS

1. Introduction

Data is a collection of information. One purpose of Data Science is to structure data, making it interpretable and easy to work with.Before data can be analyzed, it must be imported/extracted.We must clean the data in order to perform the analysis.Data science encompasses preparing data for analysis, including cleansing, aggregating, and manipulating the data to perform advanced data analysis. Analytic applications and data scientists can then review the results to uncover patterns and enable business leaders to draw informed insights.

Data science is an interdisciplinary field that extracts knowledge and insights from many structural and unstructured data, using scientific methods, data mining techniques, machine-learning algorithms, and big data. The healthcare industry generates large datasets of useful information on patient demography, treatment plans, results of medical examinations, insurance, etc. The data collected from the Internet of Things (IoT) devices attract the attention of data scientists. Data science provides aid to process, manage, analyze, and assimilate the large quantities of fragmented, structured, and unstructured data created by healthcare systems. This data requires effective management and analysis to acquire factual results. The process of data cleansing, data mining, data preparation, and data analysis used in healthcare applications is reviewed and discussed in the article. The article provides an insight into the status and prospects of big data analytics in healthcare, highlights the advantages, describes the frameworks and techniques used, briefs about the challenges faced currently, and discusses viable solutions. Data science and big data analytics can provide practical insights and aid in the decision-making of strategic decisions concerning the health system. It helps build a comprehensive view of patients, consumers, and clinicians. Data-driven decision-making opens up new possibilities to boost healthcare quality.

Data Science is a combination of multiple disciplines that uses statistics, data analysis, and machine learning to analyze data and to extract knowledge and insights from it. It is about data gathering, analysis and decision-making.Data Science is about finding patterns in data, through analysis, and make future predictions.Data Science benefits from a synergy arising from an efficiency in having the “It’s much more efficient to have the same mind with the product sense decide what metrics are relevant, the programming sense to implement tracking, and the statistical sense to provide analysis.” Critical to this is what is referred to in Data Science parlance as ‘product’. Thus the product (or context) of health is a major unique focus point of this project.

Health Data Science is the science and art of generating data-driven solutions through comprehension of complex real-world health problems, employing critical thinking and analytics to derive knowledge from data. Health Data Science is an emergent discipline, arising at the intersection of statistics, computer science, and health. Healthcare is an important domain for predictive analytics. It is one of the most popular topics in health analytics. A predictive model uses historical data, learns from it, finds patterns and generates accurate predictions from it. It finds various correlations and association of symptoms, finds habits, diseases and then makes meaningful predictions.

Predictive Analytics is playing an important role in improving patient care, chronic disease management and increasing the efficiency of supply chains and pharmaceutical logistics. Population health management is becoming an increasingly popular topic in predictive analytics. It is a data-driven approach focusing on prevention of diseases that are commonly prevalent in society. With data science, hospitals can predict the deterioration in patient’s health and provide preventive measures and start an early treatment that will assist in reducing the risk of the further aggravation of patient health. Furthermore, predictive analytics plays an important role in monitoring the logistic supply of hospitals and pharmaceutical. Data Science helps in advancing healthcare facilities and processes. It helps boost productivity in diagnosis and treatment and enhances the workflow of healthcare systems. The ultimate goals of the healthcare system are to ease the workflow of the healthcare system, reduce the risk of treatment failure, provide proper treatment on time, avoid unnecessary emergencies due to the non-availability of doctors and reduce the waiting time of patients. The best way to change health care is to identify risks and recommend prevention programs before health risks become a major problem. By wearing it with other tracking devices that pay attention to historical patterns and genetic information, you may be able to see the problem before it gets out of hand. A combination of science, technology, and medicine in the dynamic digital age has unveiled new data systems to improve statistics, improve healthcare and drug delivery, and improve health information reporting on clinical decisions. Data science in health care has seen the latest and most rapid progress in 3 ways.First way by using big data with a combination of large and complex data sets includes electronic medical records, social media, genomic information, and digital body data from wireless health devices. Second, with new open-access efforts that seek to utilize the availability of clinical trials, research, and citizen science sources for data sharing.Lastly, In the analysis techniques, especially of big data, including machine learning and artificial intelligence that can improve systematic and unstructured data analysis. As new data sets are being developed, analyzed, and growing available, a number of key questions arise, including few points like the quality of informal data processing, use of unsaved methods in data processing with traditional software and hardware lead to data fragmentation and non-productive analysis etc.

Predictive analytics in healthcare offers many benefits and, as the industry amasses more and more data, the algorithms will improve in accuracy. However, these types of tools bring their own risks that medical facilities will need to address before unlocking their full potential. Deloitte describes three main risk factors associated with implementing predictive analytics solutions in healthcare: Gaining doctors' acceptance. Healthcare providers are finding themselves in a constant need to advance their computer skills. And with predictive analytics, they will not only need to access dashboards but also keep capturing and processing patient data. It can be a challenge to find the balance between patient care and data collection during appointments. To overcome this challenge, one can involve doctors in the algorithm development process to make sure the solution addresses their needs. Some doctors may no longer put a lot of thought into making decisions as they believe predictive analytics is responsible for the consequences. To minimize this risk, everyone involved needs to understand that some decisions made by analytical tools are not binding but merely a suggestion. Clinicians still need to think them through and discuss with patients if necessary. Algorithm bias and lack of regulations. There are several types of algorithm bias that can impact predictive analytics' performance on particular datasets. What makes things harder is that there are no explicit regulations governing algorithm development. At the moment, the responsibility of producing fair tools falls on the vendor's shoulders and healthcare facilities are counting on the vendor's goodwill. To reduce the risk of bias, vendors can use feedback loops to improve their tools and eliminate any bias that still gets in. Also, organizations that are using predictive analytics in healthcare need to conduct regular audits to make sure the algorithms are still relevant.

Predictive analytics has become a key piece of any health analytics strategy. Today, it's a critical tool for measuring, aggregating, and making sense of behavioral, psychosocial, and biometric data that until recently was not available or exceedingly hard to capture. At the individual level, predictive analytics can help health providers deliver the right care to the right patient at the right time. On a larger scale, it can enable health systems to identify and understand larger trends, leading to improved population health strategies. In one example, researchers developed a model of how Ebola is spread using big data analytics and massive amounts of data, including information from social media and search engines. Individuals who have potentially been exposed to Ebola can enter their symptoms into a mobile app, which uses geocoordinates to check whether the person has been in proximity to someone from a community in which Ebola has been active. Not only can predictive analytics enhance care, but it can also dramatically reduce costs. For example, more accurate prediction models for patient length of stay and readmission rates enable hospitals to avoid penalties and reduce operational expenses. By tapping into electronic health records and predictive analytics, providers can flag patients who are likely to miss an appointment. Once identified, those patients could be reminded or otherwise supported in keeping their appointment. The enormous potential of predictive analytics includes helping identify patients at risk for chronic conditions, developing evidence-based best practices, and proactively spotting potential obstacles to care plan adherence. Data can help clinicians remain one step ahead of events, delivering proactive care to patients before their health becomes critical.

The global predictive analytics in healthcare market keeps growing and is expected to hit $7.8 billion by 2025. It provides plenty of opportunities for healthcare providers and health tech companies. The effects of certain biological factors such as genome structure or clinical variability are taken into account to predict the occurrence of specific diseases. Common causes include prediction of disease progression or prevention to reduce the risk and side effects. The main benefit is to improve the quality of life of patients and the quality of medical conditions. As we amass more and more data, predictive analytics will become both more common and more accurate.

Currently, many predictive models use traditional statistical methods, such as logistic regression, which are useful and can provide insightful results. However, AI and machine learning methods such as random forests, when implemented appropriately, can provide more accurate predictions. Ultimately, as more feature-rich data is collected and as the collection process itself improves, predictive analytics can take advantage of deep learning algorithms that can make better use of large and complex data sets. For example, deep learning algorithms can be used for image recognition, whether that be images gleaned from MRIs or other types of imaging technology, to automatically detect certain features. In contrast, a machine learning-based approach would require the radiologist to extract all the features from the image first. Deep learning simplifies the process by detecting all the features automatically, without requiring extra work by the radiologist. One current drawback is that predictive analytics is not able to provide insight into what might happen after an intervention or other change, which can be frustrating for researchers and clinicians who want to understand how patients might fare after a new treatment or as a result of a new hospital procedure. We expect predictive analytics will advance past this challenge, allowing researchers to forecast the future in a much more expansive and holistic way. Predictive analytics will also be able to benefit from the massive leaps in computer processing power that make chewing through complicated algorithms possible. Someday soon, predicting the future with accuracy under a variety of conditions will be the norm, not a fortune-teller's trick.

1.1 Objective

Diabetes is a common chronic disease and poses a great threat to human health. The characteristic of diabetes is that the blood glucose is higher than the normal level, which is caused by defective insulin secretion or its impaired biological effects, or both . Diabetes can lead to chronic damage and dysfunction of various tissues, especially eyes, kidneys, heart, blood vessels and nerves.Therefore, how to quickly and accurately diagnose and analyze diabetes is a topic worthy of studying. In medicine, the diagnosis of diabetes is according to fasting blood glucose, glucose tolerance, and random blood glucose levels. The earlier the diagnosis is obtained, the much easier we can control it. Machine learning can help people make a preliminary judgment about diabetes mellitus according to their daily physical examination data, and it can serve as a reference for doctors.The objective of this project is to make use of significant features, design a prediction algorithm using Machine learning and find the optimal classifier to give the closest result comparing to clinical outcomes.

1.2 Problem Definition

The evolution in the digital era has led to the confluence of healthcare and technology resulting in the emergence of newer data-related applications. The constant hyperglycemia of diabetes is related to long-haul harm, brokenness, and failure of various organs, particularly the eyes, kidneys, nerves, heart, and veins. The main aim is to make use of significant features, design a prediction algorithm using Machine learning and find the optimal classifier to give the closest result comparing to clinical outcomes. Doctors rely on common knowledge for treatment. When common knowledge is lacking, studies are summarized after some number of cases have been studied. But this process takes time, whereas if machine learning is used, the patterns can be identified earlier.Analyzing the details and understanding the patterns in the data can help in better decision-making resulting in a better quality of patient care. It can aid to understand the trends to The proposed method aims to focus on improvise the outcome of medical care, life expectancy, early detection, and identification of diabetis milletus disease at an initial stage and required treatment at an affordable cost.

2. Literature Survey

2.1 Related Work

The main objective of this research paper[1] is to make use of significant features, design a prediction algorithm using Machine learning and find the optimal classifier to give the closest result compared to clinical outcomes. The proposed method aims to focus on selecting the attributes that fail in early detection of Diabetes Mellitus using Predictive analysis. The result shows the decision tree algorithm and the Random forest has the highest specificity of 98.20 percent and 98.00 percent, respectively holds best for the analysis of diabetic data. Naïve Bayesian outcome states the best accuracy of 82.30 percent. The research also generalizes the selection of optimal features from the dataset to improve the classification accuracy.

In this paper [2] the author used decision trees, random forest and neural network to predict diabetes mellitus. The dataset is the hospital physical examination data in Luzhou, China. It contains 14 attributes. Five-fold cross validation is used to examine the models. The paper contains the data of randomly selected 68994 healthy people and diabetic patients' data, respectively as a training set. In this study, principal component analysis (PCA) and minimum redundancy maximum relevance (mRMR) to reduce the dimensionality. The results showed that prediction with random forest could reach the highest accuracy (ACC = 0.8084) when all the attributes were used.

The motive of this study [3] is to design a model which can prognosticate the likelihood of diabetes in patients with maximum accuracy. Therefore three machine learning classification algorithms namely Decision Tree, SVM and Naive Bayes are used in this experiment to detect diabetes at an early stage. Experiments are performed on Pima Indians Diabetes Database (PIDD) which is sourced from UCI machine learning repository. The performances of all the three algorithms are evaluated on various measures like Precision, Accuracy, F-Measure, and Recall. Accuracy is measured over correctly and incorrectly classified instances. The results obtained show Naive Bayes outperforms with the highest accuracy of 76.30 percent compared to other algorithms.

The paper [4] shows that the model is mainly built using the hidden layers of a deep neural network and uses dropout regularization to prevent overfitting. The parameters have been tuned and used the binary cross-entropy loss function, which obtained a deep neural network prediction model with high accuracy. The experimental results show the effectiveness and adequacy of the proposed DLPD (Deep Learning for Predicting Diabetes) model. The best training accuracy of the diabetes type data set is 94.02174 percent, and the training accuracy of the Pima Indians diabetes data set is 99.4112 percent.The experimental results show the improvements of our proposed model over the state-of-the-art methods.

The paper Machine Learning Based Diabetes Classification and Prediction for Healthcare Applications [5] , uses a machine learning based approach that has been proposed for the classification, early-stage identification, and prediction of diabetes. Furthermore, it also presents an IoT-based hypothetical diabetes monitoring system for a healthy and affected person to monitor his blood glucose (BG) level. For diabetes classification, three different classifiers have been employed, i.e., random forest (RF), multilayer perceptron (MLP), and logistic regression (LR). For predictive analysis, long short-term memory (LSTM) has been employed, moving averages (MA), and linear regression (LR). For experimental evaluation, a benchmark PIMA Indian Diabetes dataset is used. During the analysis, it is observed that MLP outperforms other classifiers with 86.08 percent of accuracy and LSTM improves the significant prediction with 87.26 percent accuracy of diabetes. Moreover, a comparative analysis of the proposed approach is also performed with existing state-of-the-art techniques, demonstrating the adaptability of the proposed approach in many public healthcare applications.

In this paper [6] , supervised machine-learning algorithms like Support Vector Machine (SVM), Naive Bayes classifier and LightGBM to train on the actual data of 520 diabetic patients and potential diabetic patients aged 16 to 90 for the early prediction of diabetes. Through comparative analysis of classification and recognition accuracy, the performance of the support vector machine is the best.

This paper[7] talks about how the accuracy of the model changes with the increase in the complexity of the problem. They also explained that . LGBM - Light Gradient Boosting Algorithm is one such algorithm that can be used as it depends on decision tree algorithms and it can be used in predicting the accuracy to attain the desired results. With the existing PIMA Indian Dataset the accuracy is calculated as 95.20 percent using LGBM Algorithm . Therefore by using the LGBM classifiers, we can develop a data model for diabetes detection and prediction.

In this study [8], the data of Tianchi Precision Medical Competition-artificial intelligence assisted diabetes genetic risk prediction rematch is selected to construct the LightGBM prediction model and compare with Random Forest and XGBoost on the ROC curve. The results show that the AUC of LightGBM is 85.2 percent. Compared with other prediction models, the LightGBM prediction model has more advantages and a better classification effect.Important features are constructed by LightGBM and analyzed statistically. It was found that when the value of single nucleotide polymorphism gene 37 (SNP37) is 3, it could inhibit the single nucleotide polymorphism gene 34 (SNP34), resulting in reduced risk of disease. When the value of single nucleotide polymorphism gene 34 (SNP34) and single nucleotide polymorphism gene 37 (SNP37) is 2 at the same time, they may promote each other and increase the risk of disease. Pregnant women with high insulin resistance (VAR00007) and older women are at increased risk of developing gestational diabetes. The results of this study indicate that LightGBM and other machine learning techniques play an important role in the auxiliary diagnosis of gestational diabetes and the mining of risk factors.

3. System Design

3.1 Proposed Design

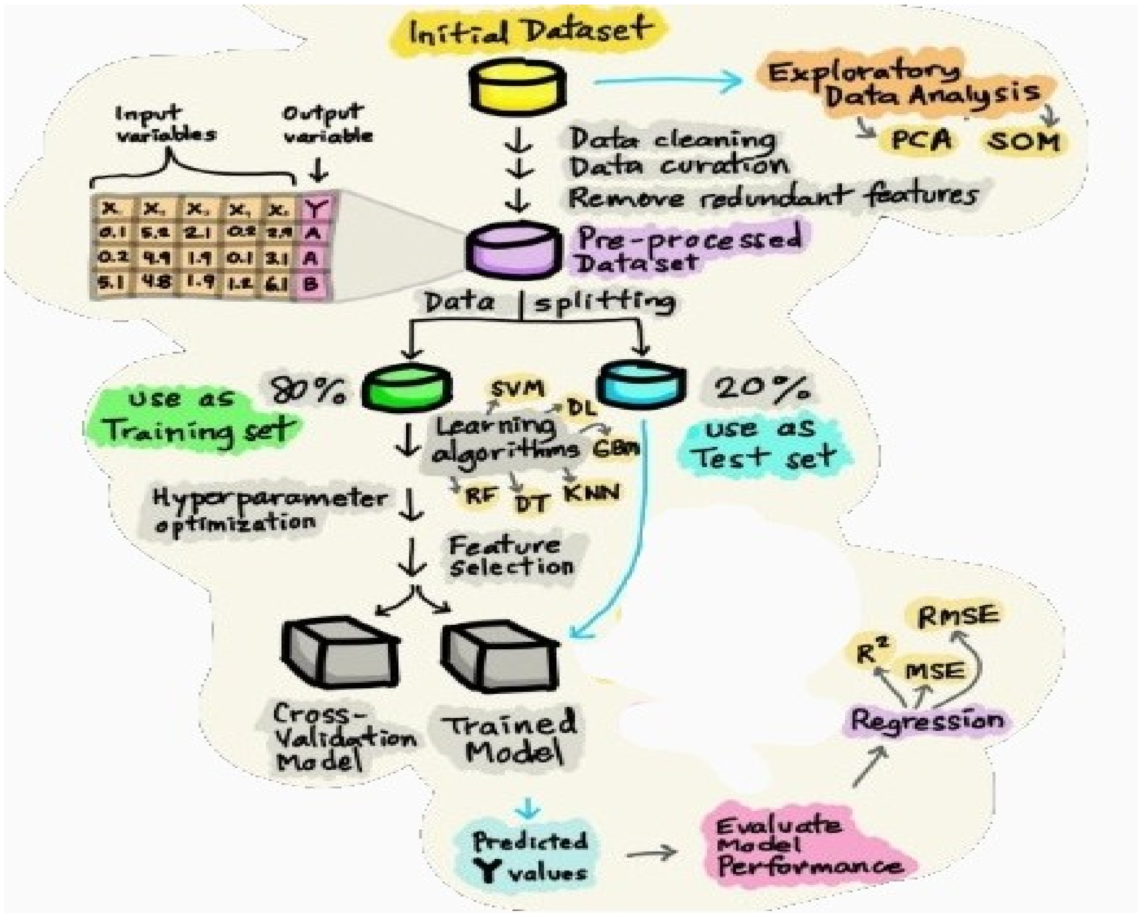


Figure 3.1: Model Design

Our project describes the design and implementation of a software system to improve the management of diabetes using a machine learning approach and to demonstrate and evaluate its effectiveness in controlling diabetes. The proposed approach for this management system handles the various factors that affect the health of people with diabetes by combining multiple machine learning algorithms.

The proposed framework factors the diabetes management problem into subgoals: understanding the dataset, performing exploratory analysis that includes data cleaning, data curation and then removing of redundant features, splitting the pre-processed dataset into training and testing set, implementing learning algorithms, applying hyper parameterized optimization, performing feature selection and cross validation and finally evaluating the model performance. The model achieved specified goals by predicting with high accuracy.

3.2 Software and Hardware Requirements

3.2.1 Hardware Requirement

1.A minimum of 2.16 GHz processor is required for the application.

2.A minimum of 4 GB RAM is required for the application.

3.The Application 64-bit architecture.

3.2.2 Software Requirement

It requires a 64-bit Ubuntu/Windows Operating System.

3.3 Libraries

3.3.1 Numpy

Numpy is one of the most commonly used packages for scientific computing in Python. It provides a multidimensional array object, as well as variations such as masks and matrices, which can be used for various math operations.

3.3.2 Sklearn

Scikit-learn (Sklearn) is the most useful and robust library for machine learning in Python. It provides a selection of efficient tools for machine learning and statistical modeling including classification, regression, clustering and dimensionality reduction via a consistence interface in Python.

3.3.3 Pandas

Pandas is an open-source library that is made mainly for working with relational or labeled data both easily and intuitively. It provides various data structures and operations for manipulating numerical data and time series. This library is built on top of the NumPy library.

3.3.4 Pickle

The pickle module implements binary protocols for serializing and de-serializing a Python object structure. “Pickling” is the process whereby a Python object hierarchy is converted into a byte stream, and “unpickling” is the inverse operation, whereby a byte stream (from a binary file or bytes-like object) is converted back into an object hierarchy.

3.3.5 Matplotlib

Matplotlib is a cross-platform, data visualization and graphical plotting library for Python and its numerical extension NumPy. As such, it offers a viable open source alternative to MATLAB. Developers can also use matplotlib’s APIs (Application Programming Interfaces) to embed plots in GUI applications.

3.3.6 Seaborn

Seaborn is a library that uses Matplotlib underneath to plot graphs. It will be used to visualize random distributions. It provides a high-level interface for drawing attractive and informative statistical graphics.

3.3.7 Catboost

CatBoost is an algorithm for gradient boosting on decision trees. It is developed by Yandex researchers and engineers, and is used for search, recommendation systems, personal assistant, self-driving cars, weather prediction and many other tasks at Yandex and in other companies, including CERN, Cloudflare, Careem taxi. It is in open-source and can be used by anyone.

3.3.8 Lightgbm

LightGBM is a gradient boosting framework that uses tree based learning algorithms. It is designed to be distributed and efficient with the following advantages like faster training speed and higher efficiency, lower memory usage, better accuracy, support of parallel, distributed, and GPU learning, and capable of handling large-scale data.

3.3.9 Xgboost

LightGBM is a gradient boosting framework that uses tree based learning algorithms. It is designed to be distributed and efficient with the following advantages like faster training speed and higher efficiency, lower memory usage, better accuracy, support of parallel, distributed, and GPU learning, and capable of handling large-scale data.

3.3.10 Streamlit

Streamlit is an open-source Python library that makes it easy to create and share beautiful, custom web apps for machine learning and data science. In just a few minutes you can build and deploy powerful data apps.

4. Methodology

4.1 Architecture

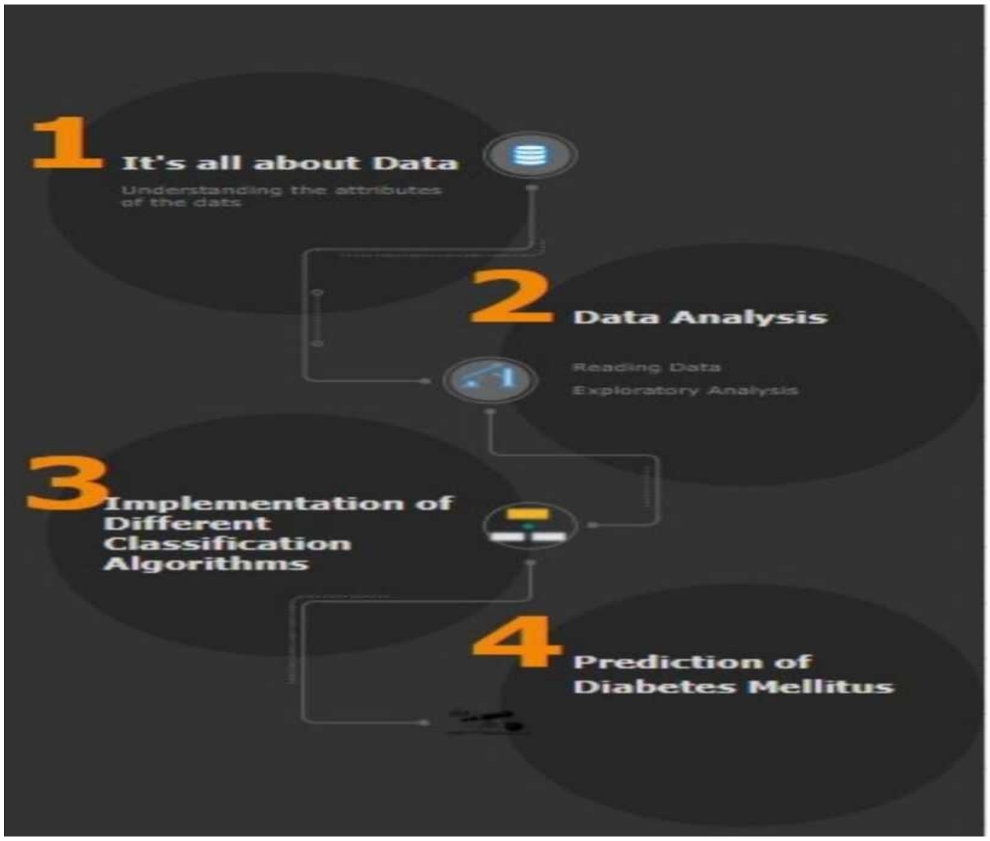

1
It's all about Data
Understanding the attributes of the dats
2
Data Analysis
Reading Data
Exploratory Analysis
3
Implementation of Different Classification Algorithms
4
Prediction of Diabetes Mellitus

Figure 4.1: Architecture

The flow of the project goes initially understanding the dataset, exploring the attributes present in the dataset for focusing on necessary features. Next analysis phase comes where preprocessing of data happens by removing the missing values, replacing the NaN values with the appropriate ones etc. Learning algorithms like random forest, linear regression, Lgbm, xg-boost, etc to predict whether the person is suffering from diabetes or not.

4.2 Modules

4.2.1 Exploratory Data Analysis

Exploratory Data Analysis (EDA) is an approach to analyzing datasets to summarize their main characteristics, often with visual methods. EDA is used for seeing what the data can tell us before the modeling task.It is used to discover trends, patterns, or to check assumptions with the help of statistical summary and graphical representations.

4.2.2 Data Preprocessing

Data preprocessing is a process of preparing the raw data and making it suitable for a machine learning model. It is the first and crucial step while creating a machine learning model. Real-world data generally contains noises, missing values, and maybe in an unusable format which cannot be directly used for machine learning models. Data preprocessing is required tasks for cleaning the data and making it suitable for a machine learning model which also increases the accuracy and efficiency of a machine

4.2.3 Feature Extraction

Feature engineering is the process of selecting, manipulating, and transforming raw data into features that can be used in supervised learning.Feature engineering is a machine learning technique that leverages data to create new variables that aren't in the training set. It can produce new features for both supervised and unsupervised learning, with the goal of simplifying and speeding up data transformations while also enhancing model accuracy. Feature engineering is required when working with machine learning models. Regardless of the data or architecture, a terrible feature will have a direct impact on your model.

4.2.4 Correlation

Correlation explains how one or more variables are related to each other. These variables can be input data features which have been used to forecast our target variable. Correlation, statistical technique which determines how one variables moves/changes in relation with the other variable.is a statistical measure that expresses the extent to which two variables are linearly related (meaning they change together at a constant rate). It's a common tool for describing simple relationships without making a statement about cause and effect.

4.2.5 Hyper Parameter Tuning

In machine learning, hyperparameter optimization or tuning is the problem of choosing a set of optimal hyperparameters for a learning algorithm. A hyperparameter is a parameter whose value is used to control the learning process. By contrast, the values of other parameters (typically node weights) are learned.The same kind of machine learning model can require different constraints, weights or learning rates to generalize different data patterns. These measures are called hyperparameters, and have to be tuned so that the model can optimally solve the machine learning problem. Hyperparameter optimization finds a tuple of hyperparameters that yields an optimal model which minimizes a predefined loss function on given independent data.The objective function takes a tuple of hyperparameters and returns the associated loss

4.3 Algorithms

As the model can be built using machine learning algorithms, there are various algorithms which give good accuracy for our model. The different algorithms which we implemented for training our model are logistic regression,KNN, XGBoost, Random Forest and Light GBM.

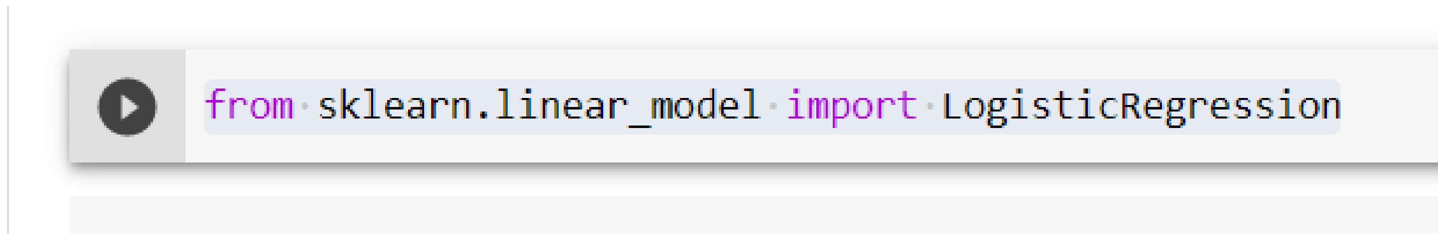
4.3.1 Logistic Regression

Logistic regression is a statistical model that in its basic form uses a logistic function to model a binary dependent variable, although many more complex extensions exist. In regression analysis, logistic regression (or logit regression) is estimating the parameters of a logistic model (a form of binary regression). It is used in statistical software to understand the relationship between the dependent variable and one or more independent variables by estimating probabilities using a logistic regression equation. This type of analysis can help you predict the likelihood of an event happening or a choice being made.

Steps in Logistic Regression:

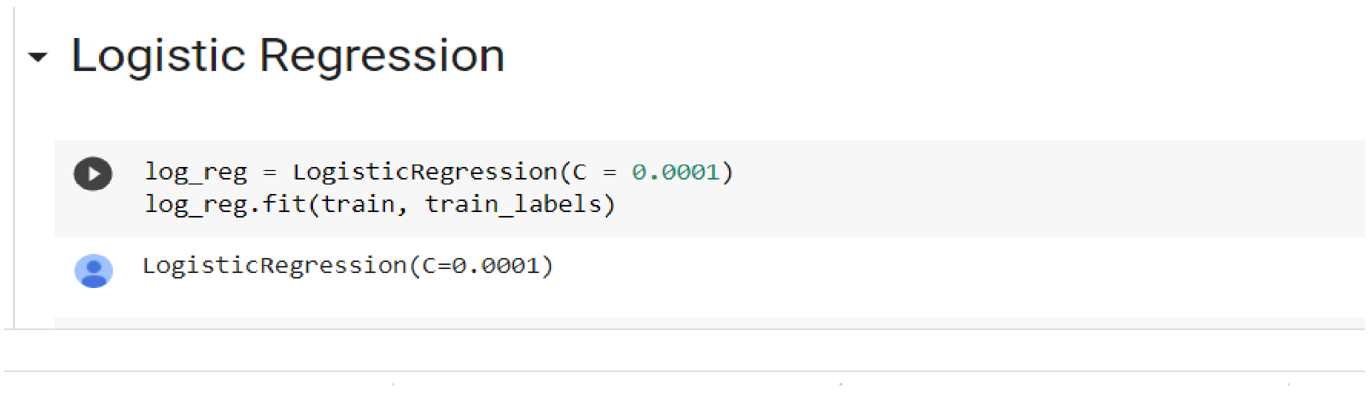
- Data Pre-processing step
- Fitting Logistic Regression to the Training set
- Predicting the test result
- Test accuracy of the result(Creation of Confusion matrix)
- Visualizing the test set result.

In our model to implement logistic regression we need to import classifier as shown in fig 4.2 and fit the classifier as shown in fig 4.3



```
from sklearn.linear_model import LogisticRegression
```

Figure 4.2: Logistic Regression Import



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

```
LogisticRegression(C=0.0001)
```

Figure 4.3: Logistic Regression Model Fit

4.3.2 K Nearest Neighbours

K-Nearest Neighbour is one of the simplest Machine Learning algorithms based on Supervised Learning technique. K-NN algorithm assumes the similarity between the new case/data and available cases and put the new case into the category that is most similar to the available categories. K-NN algorithm stores all the available data and classifies a new data point based on the similarity. This means when new data appears then it can be easily classified into a well suite category by using K- NN algorithm. K-NN algorithm can be used for Regression as well as for Classification but mostly it is used for the Classification problems. K-NN is a non-parametric algorithm, which means it does not make any assumption on underlying data.It is also called a lazy learner algorithm because it does not learn from the training set immediately instead it stores the dataset and at the time of classification, it performs an action on the dataset. KNN algorithm at the training phase just stores the dataset and when it gets new data, then it classifies that data into a category that is much similar to the new data.

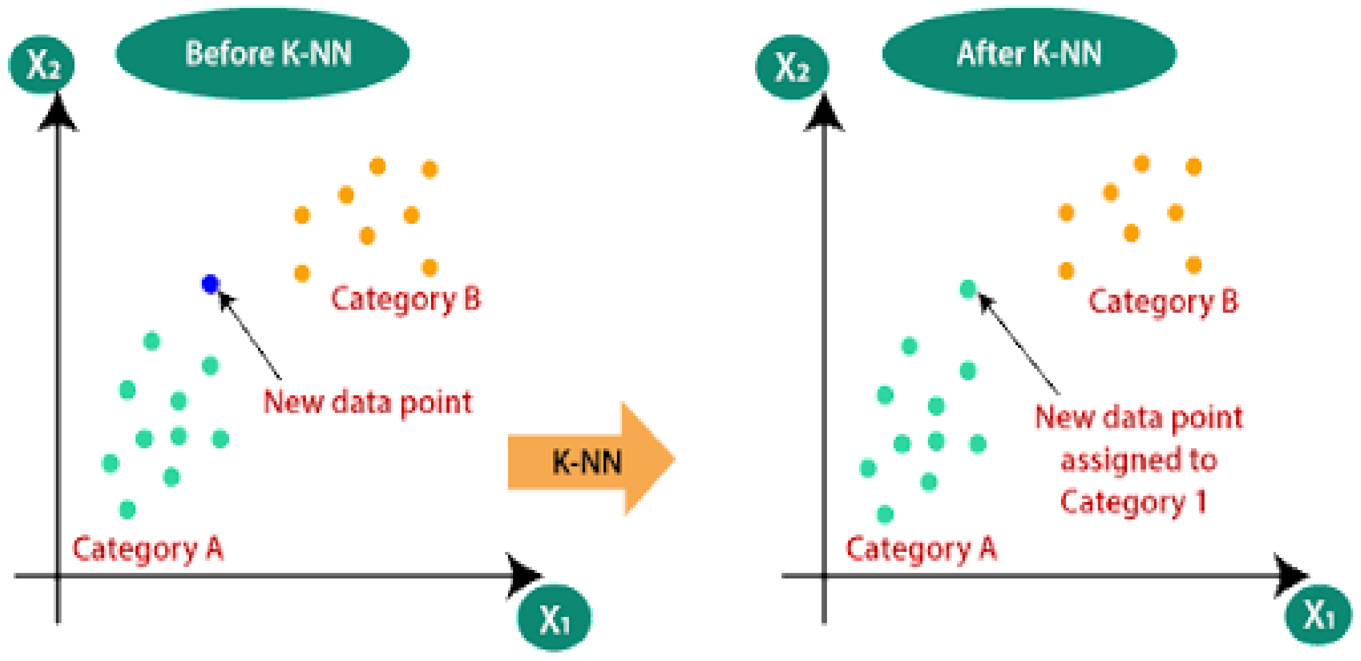
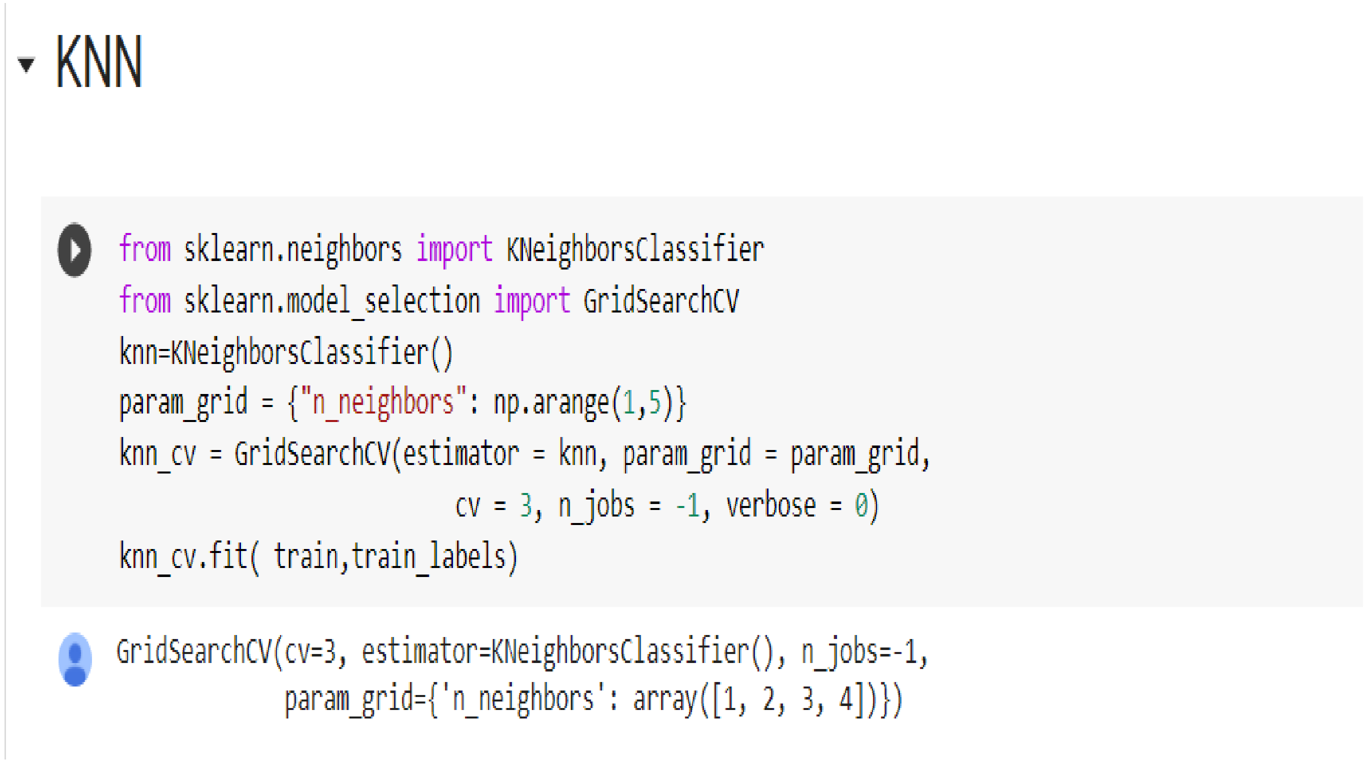


Figure 4.4: K Nearest Neightbour Algorithm

The K-NN working can be explained on the basis of the below algorithm:

- Select the number K of the neighbors
- Calculate the Euclidean distance of K number of neighbors
- Take the K nearest neighbors as per the calculated Euclidean distance.
- Among these k neighbors, count the number of the data points in each category.
- Assign the new data points to that category for which the number of the neighbor is maximum.
- Our model is ready.

In our model to implement KNN we need to import classifier as fit the classifier as shown in fig 4.5



KNN

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import GridSearchCV
knn=KNeighborsClassifier()
param_grid = {"n_neighbors": np.arange(1,5)}
knn_cv = GridSearchCV(estimator = knn, param_grid = param_grid,
                          cv = 3, n_jobs = -1, verbose = 0)
knn_cv.fit( train,train_labels)
```

```
GridSearchCV(cv=3, estimator=KNeighborsClassifier(), n_jobs=-1,
             param_grid={'n_neighbors': array([1, 2, 3, 4])})
```

Figure 4.5: KNN import and model fitting

4.3.3 XGBoost

XGBoost is an optimized distributed gradient boosting library designed to be highly efficient, flexible and portable. It implements machine learning algorithms under the Gradient Boosting framework. XGBoost provides a parallel tree boosting (also known as GBDT, GBM) that solve many data science problems in a fast and accurate way.XGBoost is an algorithm that has recently been dominating applied machine learning and Kaggle competitions for structured or tabular data. XGBoost is an implementation of gradient boosted decision trees designed for speed and performance.XGBoost, which stands for Extreme Gradient Boosting, is a scalable, distributed gradient-boosted decision tree (GBDT) machine learning library. It provides parallel tree boosting and is the leading machine learning library for regression, classification, and ranking problems
The boosting ensemble technique consists of three simple steps:

- An initial model F0 is defined to predict the target variable y. This model will be associated with a residual (y – F0)
- A new model h1 is fit to the residuals from the previous step.
- Now, F0 and h1 are combined to give F1, the boosted version of F0.
- Among these k neighbors, count the number of the data points in each category.

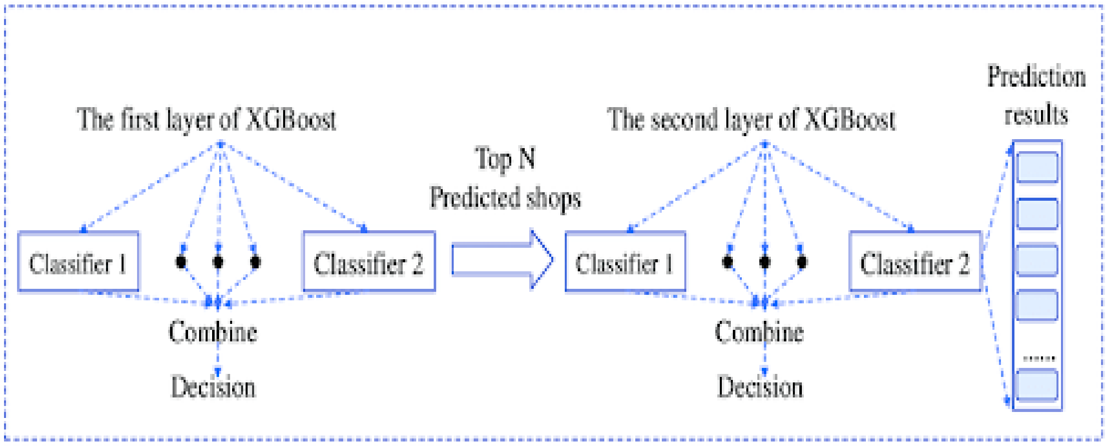
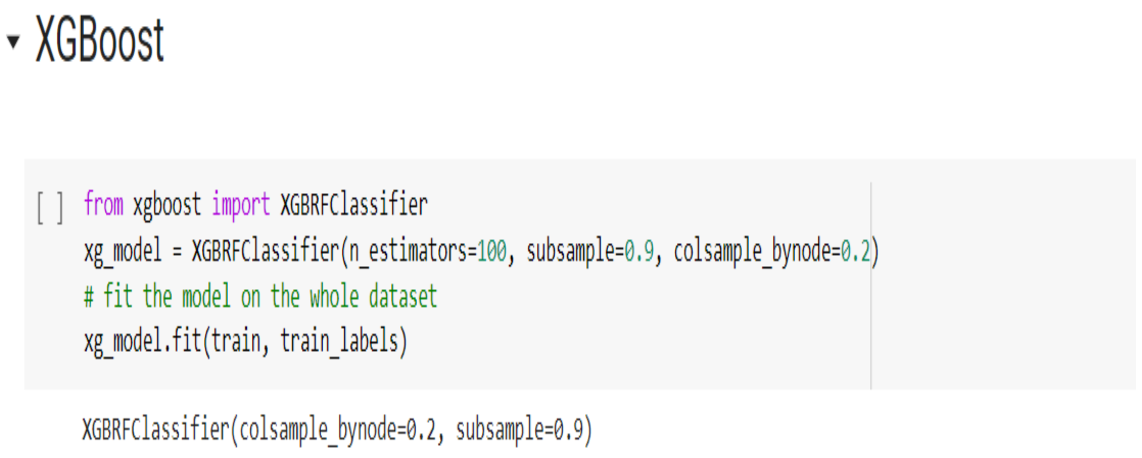


Figure 4.6: XG Boost Algorithm

In our model to implement XGBoost we need to import classifier and fit the classifier as shown in fig 4.7



```
from xgboost import XGBRFClassifier
xg_model = XGBRFClassifier(n_estimators=100, subsample=0.9, colsample_bynode=0.2)
# fit the model on the whole dataset
xg_model.fit(train, train_labels)
```

```
XGBRFClassifier(colsample_bynode=0.2, subsample=0.9)
```

Figure 4.7: XGBoost import and model Fitting

4.4.4 Random Forest

Random Forest is a popular machine learning algorithm that belongs to the supervised learning technique. It can be used for both Classification and Regression problems in ML. It is based on the concept of ensemble learning, which is a process of combining multiple classifiers to solve a complex problem and to improve the performance of the model.Random Forest is a classifier that contains a number of decision trees on various subsets of the given dataset and takes the average to improve the predictive accuracy of that dataset.Ïnstead of relying on one decision tree, the random forest takes the prediction from each tree and based on the majority votes of predictions, and it predicts the final output.The greater number of trees in the forest leads to higher accuracy and prevents the problem of overfitting.

The below diagram explains the working of the Random Forest algorithm:

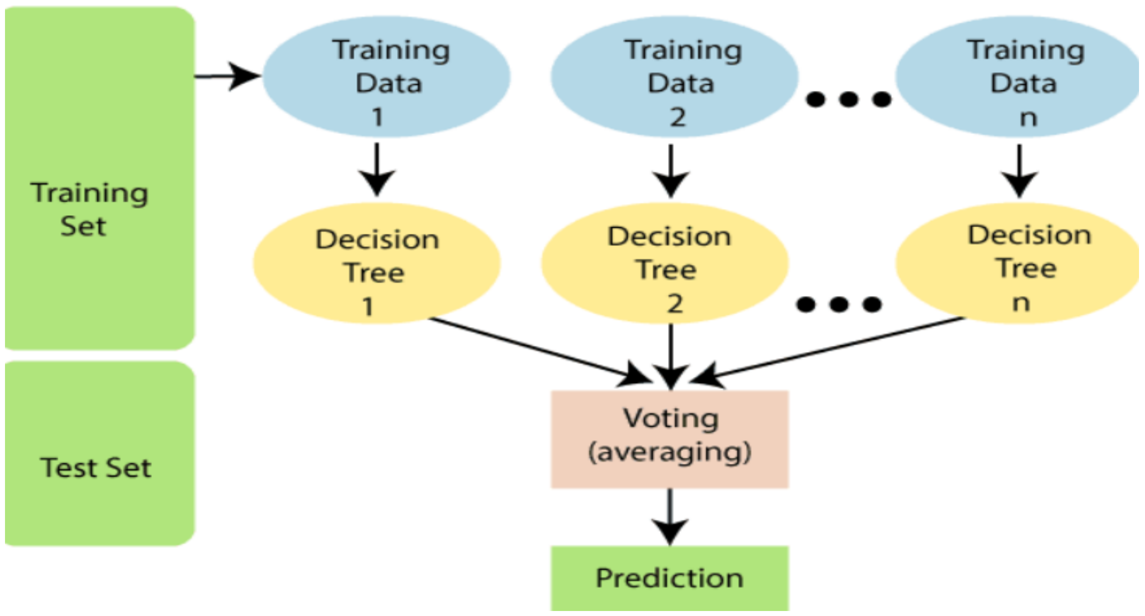


Figure 4.8: Random Forest Algorithm

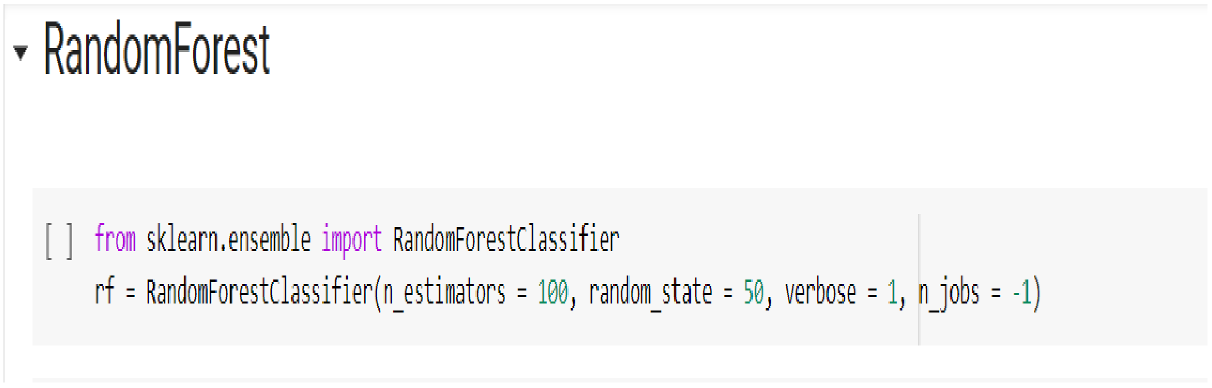
The reasons why we used random forest algorithm are:

- It takes less training time as compared to other algorithms.
- It predicts output with high accuracy, even for the large dataset it runs efficiently.
- It can also maintain accuracy when a large proportion of data is missing

The Working process can be explained in the below steps:

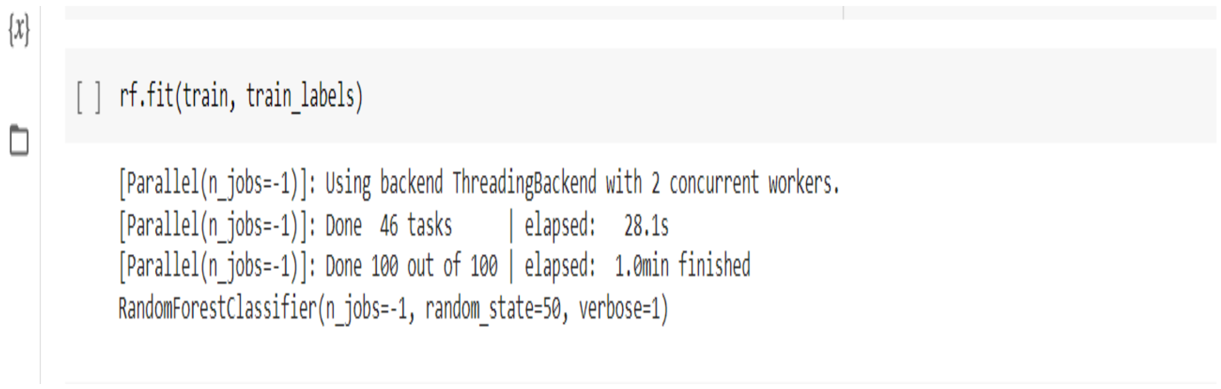
- Select random K data points from the training set.
- Build the decision trees associated with the selected data points (Subsets).
- Choose the number N for decision trees that you want to build.
- Repeat Step 1 2.
- For new data points, find the predictions of each decision tree, and assign the new data points to the category that wins the majority votes.

In our model to implement random forest we need to import classifier as shown in fig 4.9 and fit the classifier as shown in fig 4.10



```
[ ] from sklearn.ensemble import RandomForestClassifier
    rf = RandomForestClassifier(n_estimators = 100, random_state = 50, verbose = 1, n_jobs = -1)
```

Figure 4.9: Random Forest import



```
[ ] rf.fit(train, train_labels)

    [Parallel(n_jobs=-1)]: Using backend ThreadingBackend with 2 concurrent workers.
    [Parallel(n_jobs=-1)]: Done  46 tasks      | elapsed:   28.1s
    [Parallel(n_jobs=-1)]: Done 100 out of 100 | elapsed:  1.0min finished
    RandomForestClassifier(n_jobs=-1, random_state=50, verbose=1)
```

Figure 4.10: Random Forest Model fitting

4.4.5 Light GBM

Light GBM is a fast, distributed, high-performance gradient boosting framework based on a decision tree algorithm, used for ranking, classification and many other machine learning tasks to increase the efficiency of the model and reduce memory usage. It uses two novel techniques: Gradient-based One Side Sampling and Exclusive Feature Bundling (EFB) which fulfills the limitations of histogram-based algorithm that is primarily used in all GBDT (Gradient Boosting Decision Tree) frameworks. The two techniques of GOSS and EFB described below form the characteristics of LightGBM Algorithm. They comprise together to make the model work efficiently and provide it a cutting edge over other GBDT frameworks. LightGBM splits the tree leaf-wise as opposed to other boosting algorithms that grow tree level-wise. It chooses the leaf with maximum delta loss to grow. Since the leaf is fixed, the leaf-wise algorithm has lower loss compared to the level-wise algorithm. Leaf-wise tree growth might increase the complexity of the model and may lead to overfitting in small datasets. Below is a diagrammatic representation of Leaf-Wise Tree Growth:

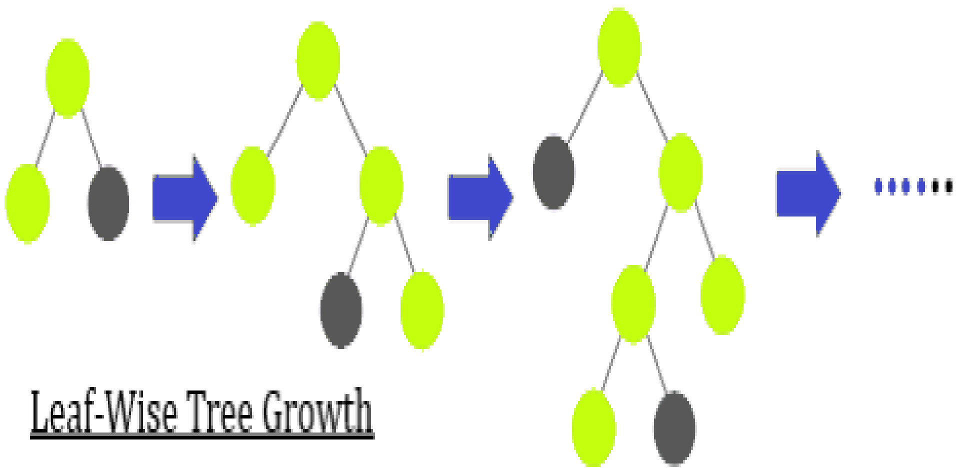
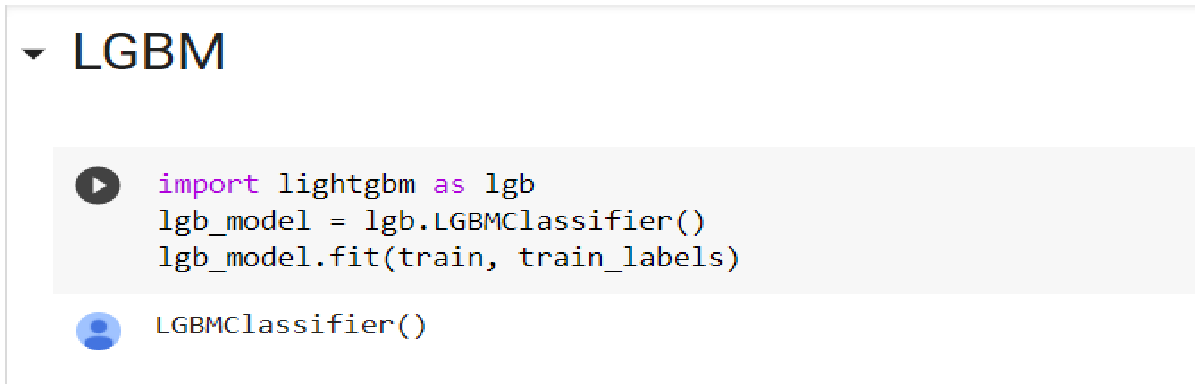


Figure 4.11: Light GBM Algorithm

In our model to implement Light GBM we need to import classifier and fit the classifier as shown in fig 4.12



LGBM

```
import lightgbm as lgb
lgb_model = lgb.LGBMClassifier()
lgb_model.fit(train, train_labels)
```

```
LGBMClassifier()
```

Figure 4.12: Light GBM Import and Model Fitting

5. Partial Implementation and Results

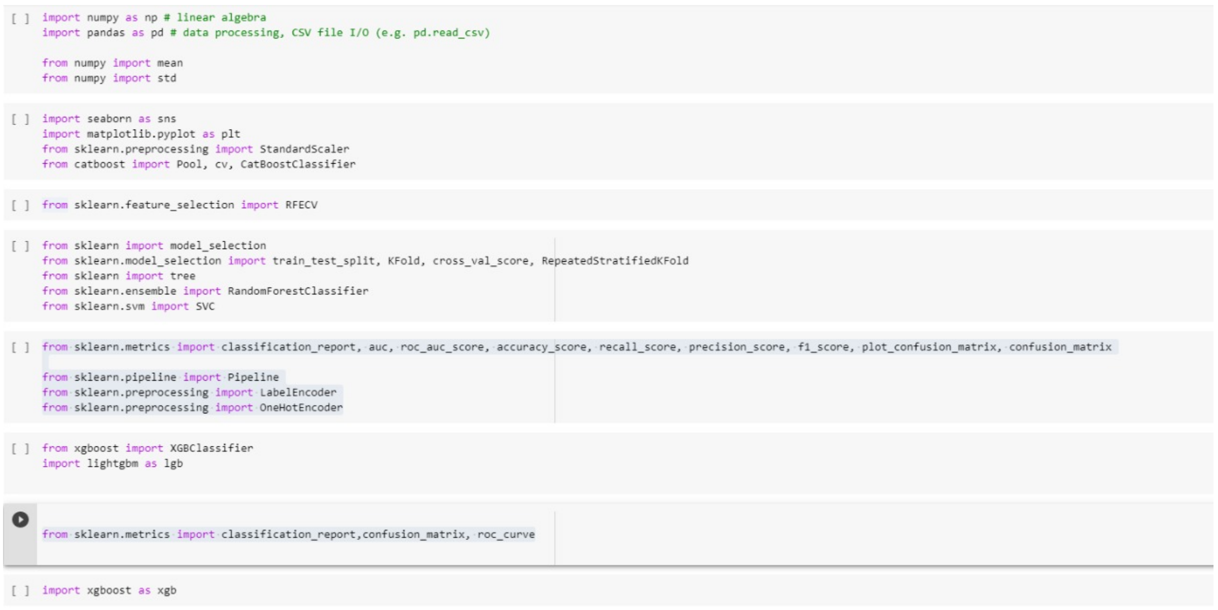
5.1 Implementation

Since our model can be built using machine learning algorithms, so the steps for partially implementing this model are:

- Import the necessary libraries
- Reading the Data
- Exploratory Data Analysis
- Visualizing the Data
- Handling Missing values
- Preprocessing the Data
- Encoding Categorical Values
- Splitting the data
- Algorithm Implementation

Importing Libraries

The necessary libraries for building the model can be imported by using import keyword is shown in the below fig 5.1

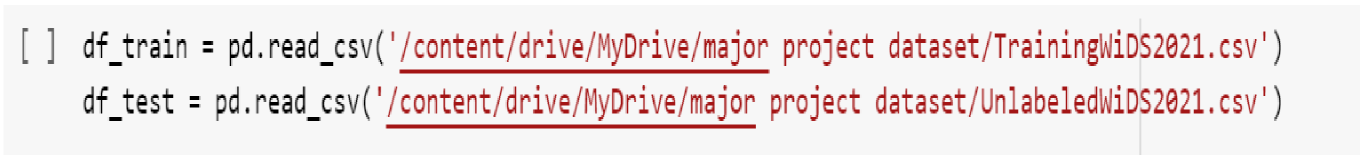


```
[ ] import numpy as np # linear algebra
    import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

    from numpy import mean
    from numpy import std

[ ] import seaborn as sns
    import matplotlib.pyplot as plt
    from sklearn.preprocessing import StandardScaler
    from catboost import Pool, cv, CatBoostClassifier

[ ] from sklearn.feature_selection import RFECV

[ ] from sklearn import model_selection
    from sklearn.model_selection import train_test_split, KFold, cross_val_score, RepeatedStratifiedKFold
    from sklearn import tree
    from sklearn.ensemble import RandomForestClassifier
    from sklearn.svm import SVC

[ ] from sklearn.metrics import classification_report, auc, roc_auc_score, accuracy_score, recall_score, precision_score, f1_score, plot_confusion_matrix, confusion_matrix

    from sklearn.pipeline import Pipeline
    from sklearn.preprocessing import LabelEncoder
    from sklearn.preprocessing import OneHotEncoder

[ ] from xgboost import XGBClassifier
    import lightgbm as lgb


    from sklearn.metrics import classification_report,confusion_matrix, roc_curve

[ ] import xgboost as xgb
```

Figure 5.1: Importing Libraries

Reading the Data

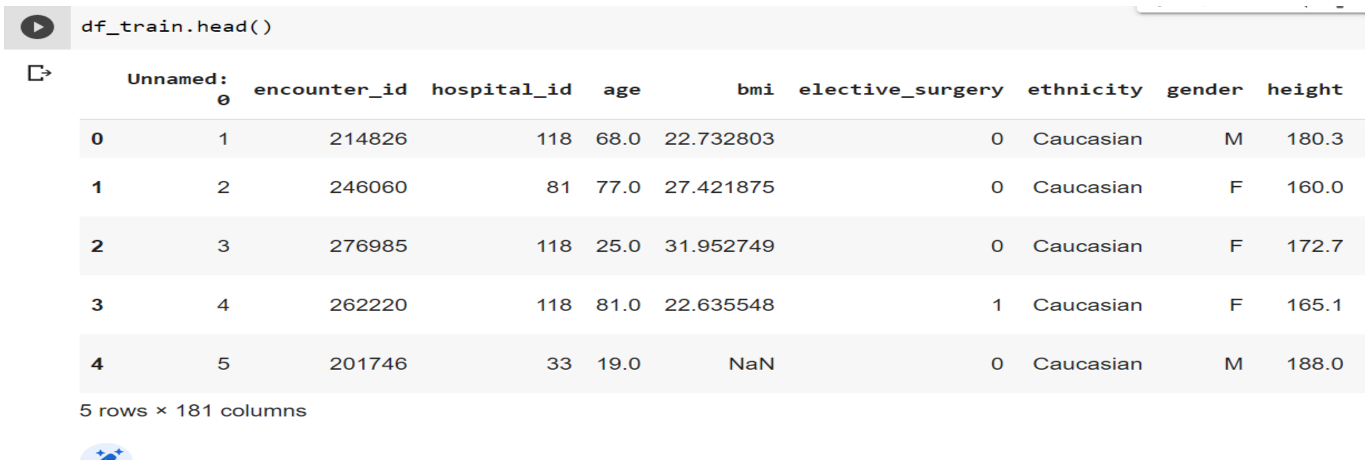
The data can be read by using read csv method from pandas as shown in the fig 5.2



```
[ ] df_train = pd.read_csv('/content/drive/MyDrive/major project dataset/TrainingWiDS2021.csv')
    df_test = pd.read_csv('/content/drive/MyDrive/major project dataset/UnlabeledWiDS2021.csv')
```

Figure 5.2: Reading Data

The read data can be seen by using head() function as shown in the fig 5.3



```
df_train.head()
```

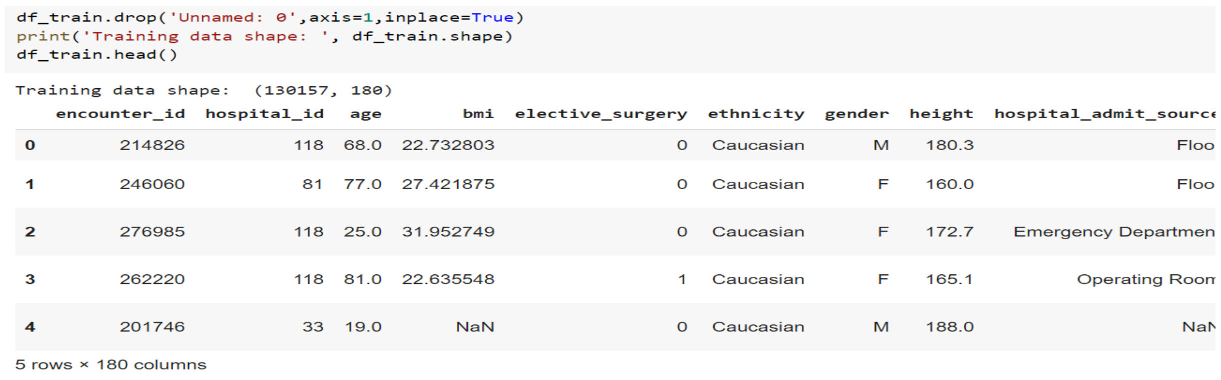
	Unnamed: 0	encounter_id	hospital_id	age	bmi	elective_surgery	ethnicity	gender	height
0	1	214826	118	68.0	22.732803	0	Caucasian	M	180.3
1	2	246060	81	77.0	27.421875	0	Caucasian	F	160.0
2	3	276985	118	25.0	31.952749	0	Caucasian	F	172.7
3	4	262220	118	81.0	22.635548	1	Caucasian	F	165.1
4	5	201746	33	19.0	NaN	0	Caucasian	M	188.0

5 rows × 181 columns

Figure 5.3: Read Data

Exploratory Data Analysis

Exploratory Data Analysis include understanding the data, finding the target column, checking the missing values, removing the unneccessary columns from the data as shown in the below figure 5.4 and figure 5.5



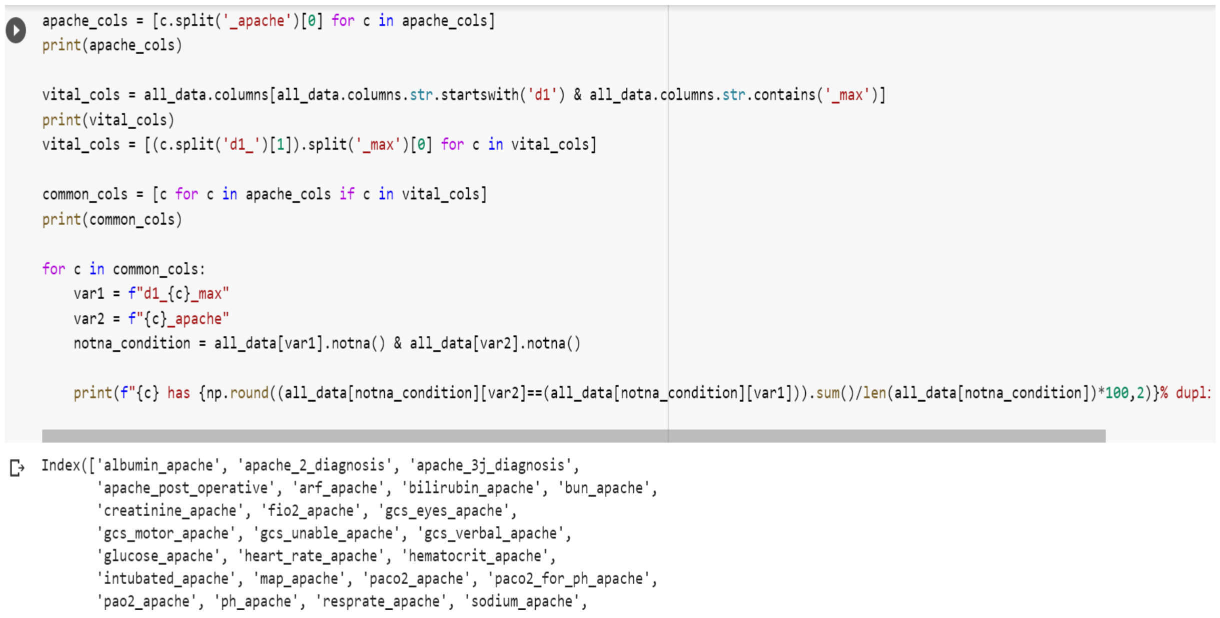
```
df_train.drop('Unnamed: 0',axis=1,inplace=True)
print('Training data shape: ', df_train.shape)
df_train.head()
```

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

	encounter_id	hospital_id	age	bmi	elective_surgery	ethnicity	gender	height	hospital_admit_sourc
0	214826	118	68.0	22.732803	0	Caucasian	M	180.3	Floo
1	246060	81	77.0	27.421875	0	Caucasian	F	160.0	Floo
2	276985	118	25.0	31.952749	0	Caucasian	F	172.7	Emergency Departmen
3	262220	118	81.0	22.635548	1	Caucasian	F	165.1	Operating Roon
4	201746	33	19.0	NaN	0	Caucasian	M	188.0	Nan

5 rows × 180 columns

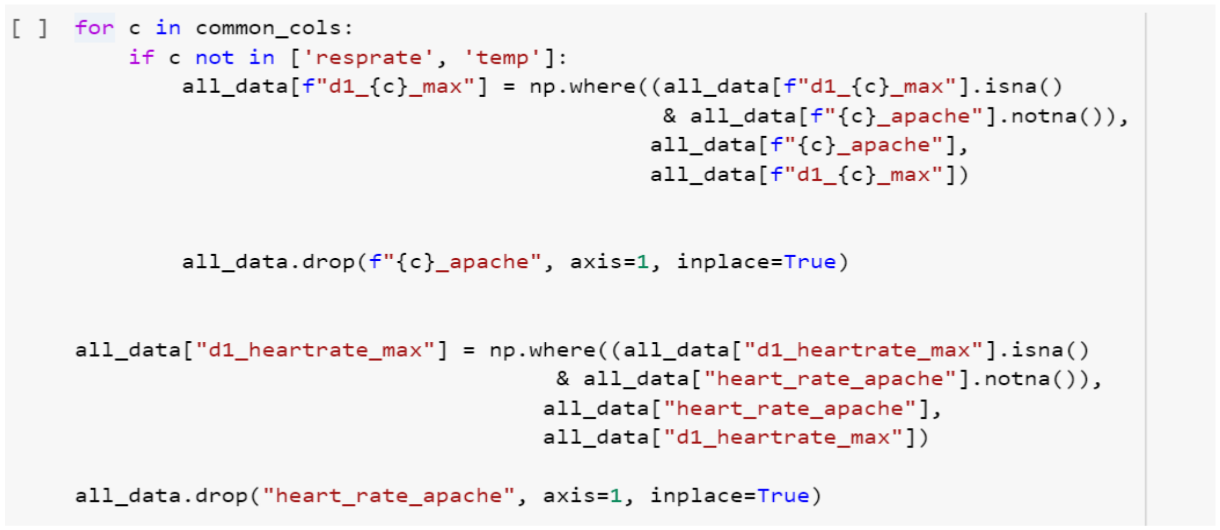
Figure 5.4: Dropping unnecessary columns



```
apache_cols = [c.split('_apache')[0] for c in apache_cols]
print(apache_cols)

vital_cols = all_data.columns[all_data.columns.str.startswith('d1') & all_data.columns.str.contains('_max')]
print(vital_cols)
vital_cols = [(c.split('d1_')[1]).split('_max')[0] for c in vital_cols]

common_cols = [c for c in apache_cols if c in vital_cols]
print(common_cols)

for c in common_cols:
    var1 = f"d1_{c}_max"
    var2 = f"{c}_apache"
    notna_condition = all_data[var1].notna() & all_data[var2].notna()

    print(f"{c} has {np.round((all_data[notna_condition][var2]==(all_data[notna_condition][var1])).sum()/len(all_data[notna_condition])*100,2)}% dupl:
```

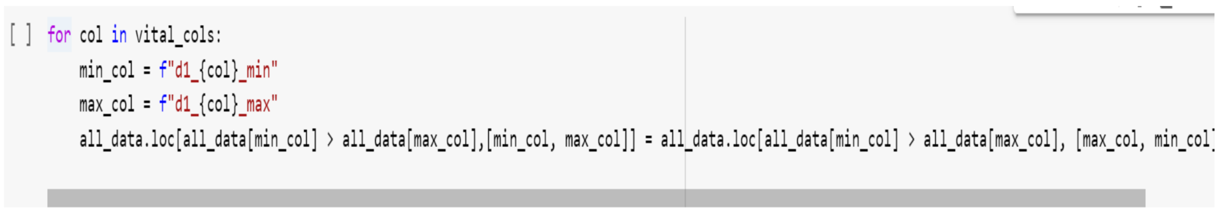
```
Index(['albumin_apache', 'apache_2_diagnosis', 'apache_3j_diagnosis',
       'apache_post_operative', 'arf_apache', 'bilirubin_apache', 'bun_apache',
       'creatinine_apache', 'fio2_apache', 'gcs_eyes_apache',
       'gcs_motor_apache', 'gcs_unable_apache', 'gcs_verbal_apache',
       'glucose_apache', 'heart_rate_apache', 'hematocrit_apache',
       'intubated_apache', 'map_apache', 'paco2_apache', 'paco2_for_ph_apache',
       'pao2_apache', 'ph_apache', 'resprate_apache', 'sodium_apache',
```

Figure 5.5: Getting Apache columns

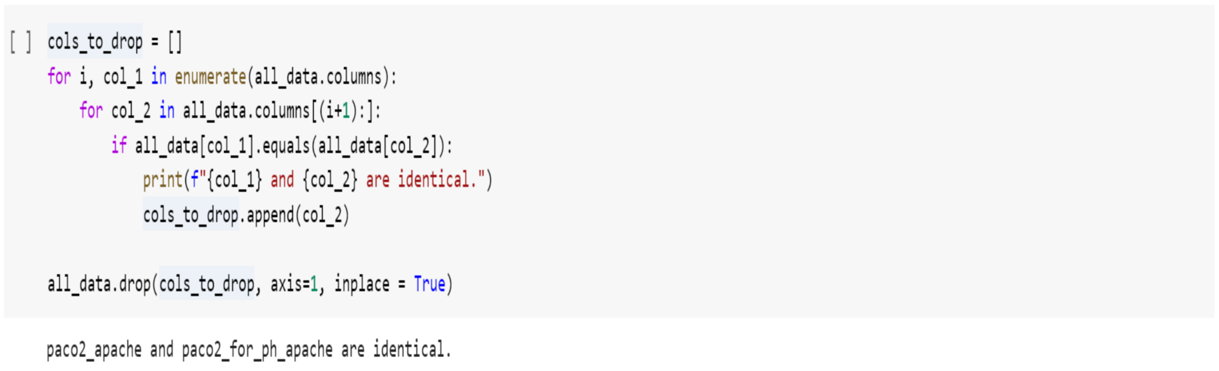


```
[ ]  for c in common_cols:
         if c not in ['resprate', 'temp']:
             all_data[f"d1_{c}_max"] = np.where((all_data[f"d1_{c}_max"].isna()
                                                  & all_data[f"{c}_apache"].notna()),
                                                 all_data[f"{c}_apache"],
                                                 all_data[f"d1_{c}_max"])


             all_data.drop(f"{c}_apache", axis=1, inplace=True)


     all_data["d1_heartrate_max"] = np.where((all_data["d1_heartrate_max"].isna()
                                               & all_data["heart_rate_apache"].notna()),
                                              all_data["heart_rate_apache"],
                                              all_data["d1_heartrate_max"])

     all_data.drop("heart_rate_apache", axis=1, inplace=True)
```

Figure 5.6: Dropping common columns

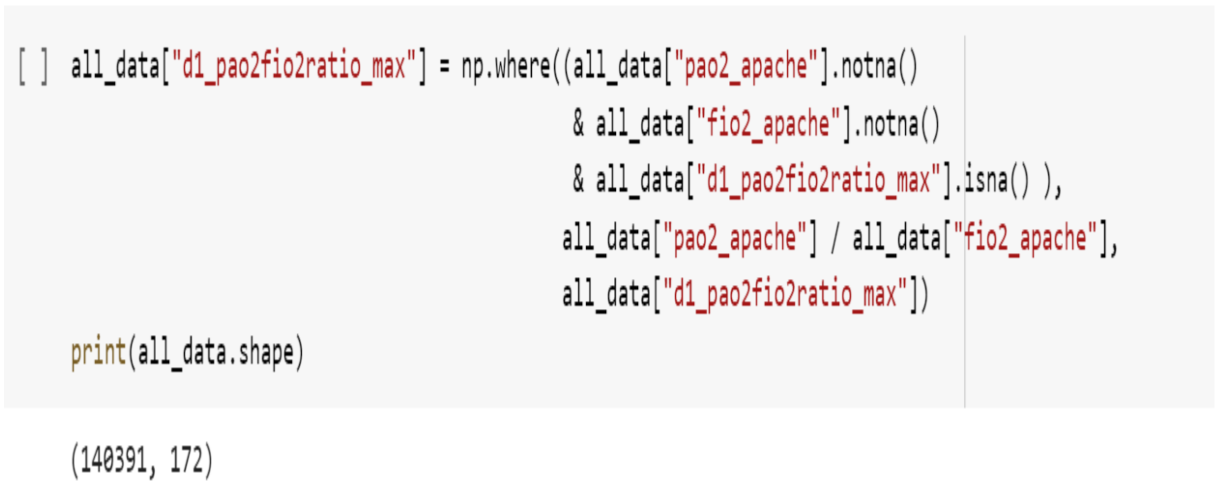


```
[ ] for col in vital_cols:
        min_col = f"d1_{col}_min"
        max_col = f"d1_{col}_max"
        all_data.loc[all_data[min_col] > all_data[max_col],[min_col, max_col]] = all_data.loc[all_data[min_col] > all_data[max_col], [max_col, min_col]
```



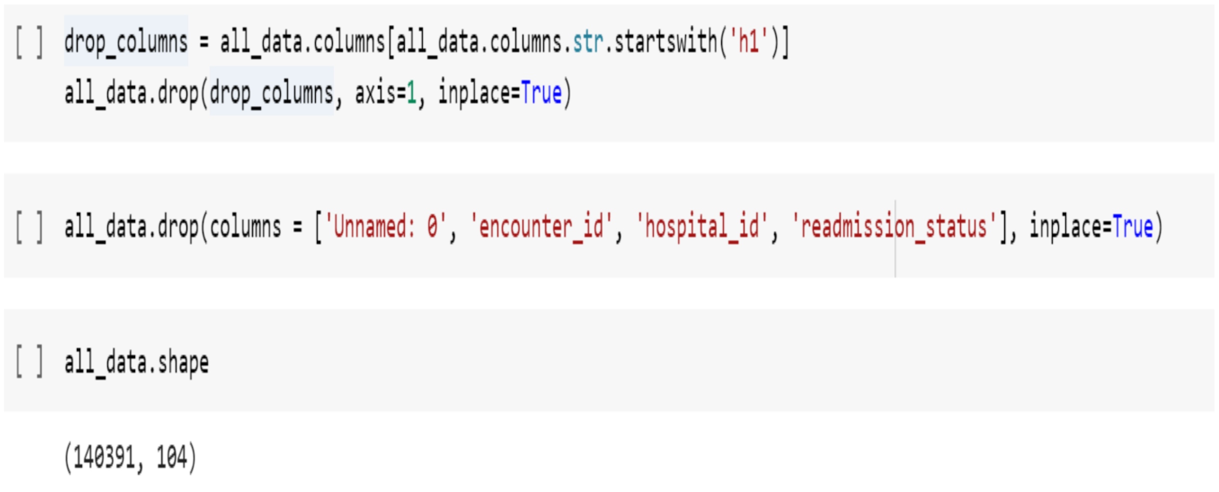
```
[ ] cols_to_drop = []
    for i, col_1 in enumerate(all_data.columns):
        for col_2 in all_data.columns[(i+1):]:
            if all_data[col_1].equals(all_data[col_2]):
                print(f"{col_1} and {col_2} are identical.")
                cols_to_drop.append(col_2)

    all_data.drop(cols_to_drop, axis=1, inplace = True)

    paco2_apache and paco2_for_ph_apache are identical.
```

Figure 5.7: Dropping identical columns



```
[ ] all_data["d1_pao2fio2ratio_max"] = np.where((all_data["pao2_apache"].notna()
                                                 & all_data["fio2_apache"].notna()
                                                 & all_data["d1_pao2fio2ratio_max"].isna() ),
                                                all_data["pao2_apache"] / all_data["fio2_apache"],
                                                all_data["d1_pao2fio2ratio_max"])

    print(all_data.shape)

    (140391, 172)
```

Figure 5.8: Handling missing columns

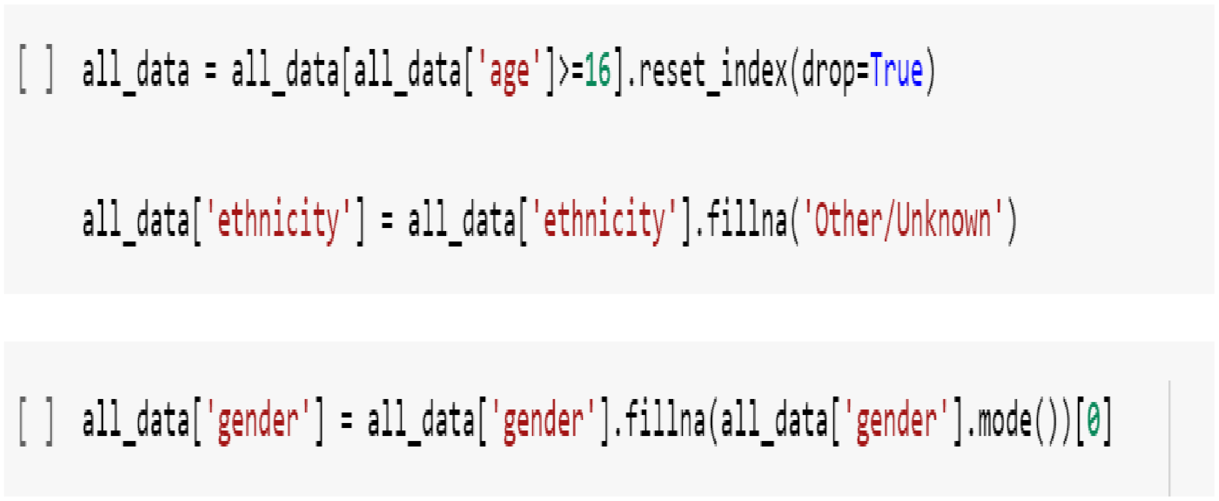


```
[ ] drop_columns = all_data.columns[all_data.columns.str.startswith('h1')]
    all_data.drop(drop_columns, axis=1, inplace=True)
```

```
[ ] all_data.drop(columns = ['Unnamed: 0', 'encounter_id', 'hospital_id', 'readmission_status'], inplace=True)
```

```
[ ] all_data.shape

    (140391, 104)
```

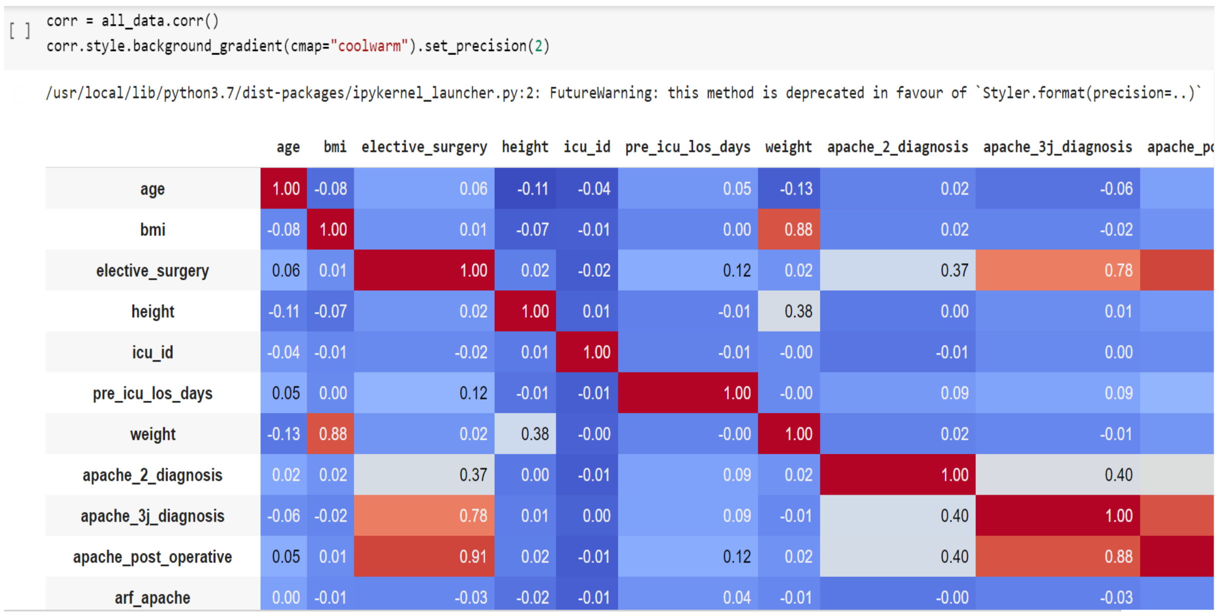
Figure 5.9: Drop unwanted columns



```
[ ] all_data = all_data[all_data['age']>=16].reset_index(drop=True)

    all_data['ethnicity'] = all_data['ethnicity'].fillna('Other/Unknown')
```

```
[ ] all_data['gender'] = all_data['gender'].fillna(all_data['gender'].mode())[0]
```

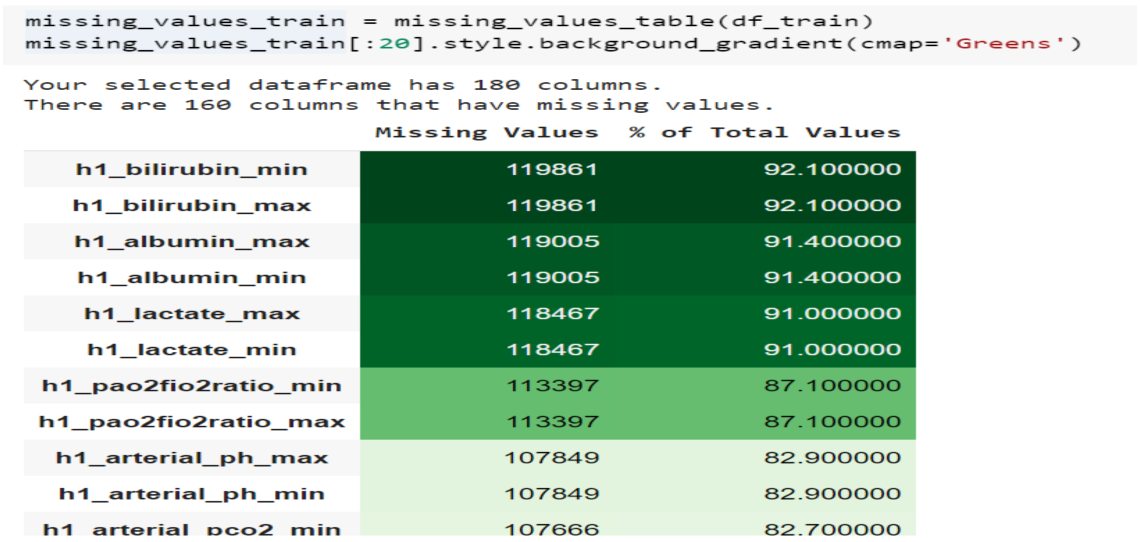
Figure 5.10: Filling nulls



```
[ ] corr = all_data.corr()
    corr.style.background_gradient(cmap="coolwarm").set_precision(2)

/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:2: FutureWarning: this method is deprecated in favour of `Styler.format(precision=..)`
```

	age	bmi	elective_surgery	height	icu_id	pre_icu_los_days	weight	apache_2_diagnosis	apache_3j_diagnosis	apache_pc
age	1.00	-0.08	0.06	-0.11	-0.04	0.05	-0.13	0.02	-0.06	
bmi	-0.08	1.00	0.01	-0.07	-0.01	0.00	0.88	0.02	-0.02	
elective_surgery	0.06	0.01	1.00	0.02	-0.02	0.12	0.02	0.37	0.78	
height	-0.11	-0.07	0.02	1.00	0.01	-0.01	0.38	0.00	0.01	
icu_id	-0.04	-0.01	-0.02	0.01	1.00	-0.01	-0.00	-0.01	0.00	
pre_icu_los_days	0.05	0.00	0.12	-0.01	-0.01	1.00	-0.00	0.09	0.09	
weight	-0.13	0.88	0.02	0.38	-0.00	-0.00	1.00	0.02	-0.01	
apache_2_diagnosis	0.02	0.02	0.37	0.00	-0.01	0.09	0.02	1.00	0.40	
apache_3j_diagnosis	-0.06	-0.02	0.78	0.01	0.00	0.09	-0.01	0.40	1.00	
apache_post_operative	0.05	0.01	0.91	0.02	-0.01	0.12	0.02	0.40	0.88	
arf_apache	0.00	-0.01	-0.03	-0.02	-0.01	0.04	-0.01	-0.00	-0.03	

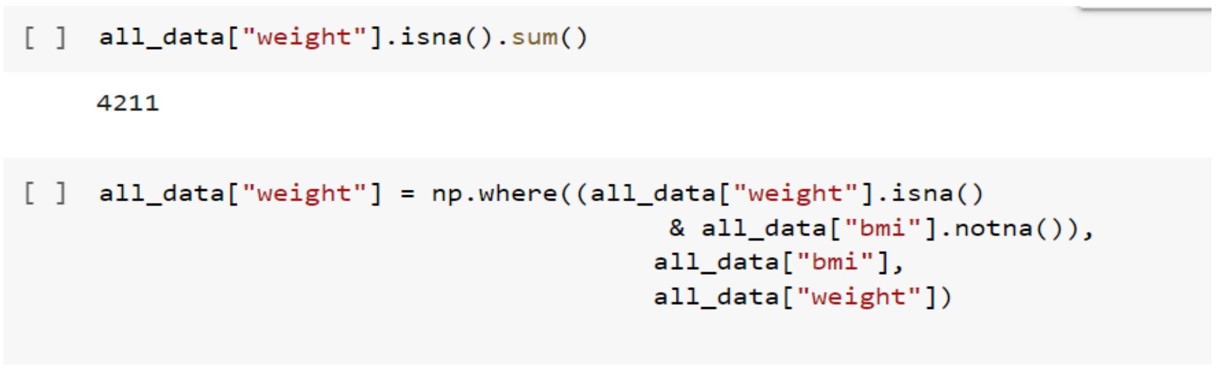
Figure 5.11: Correlated columns



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

Your selected dataframe has 180 columns.
There are 160 columns that have missing values.
```

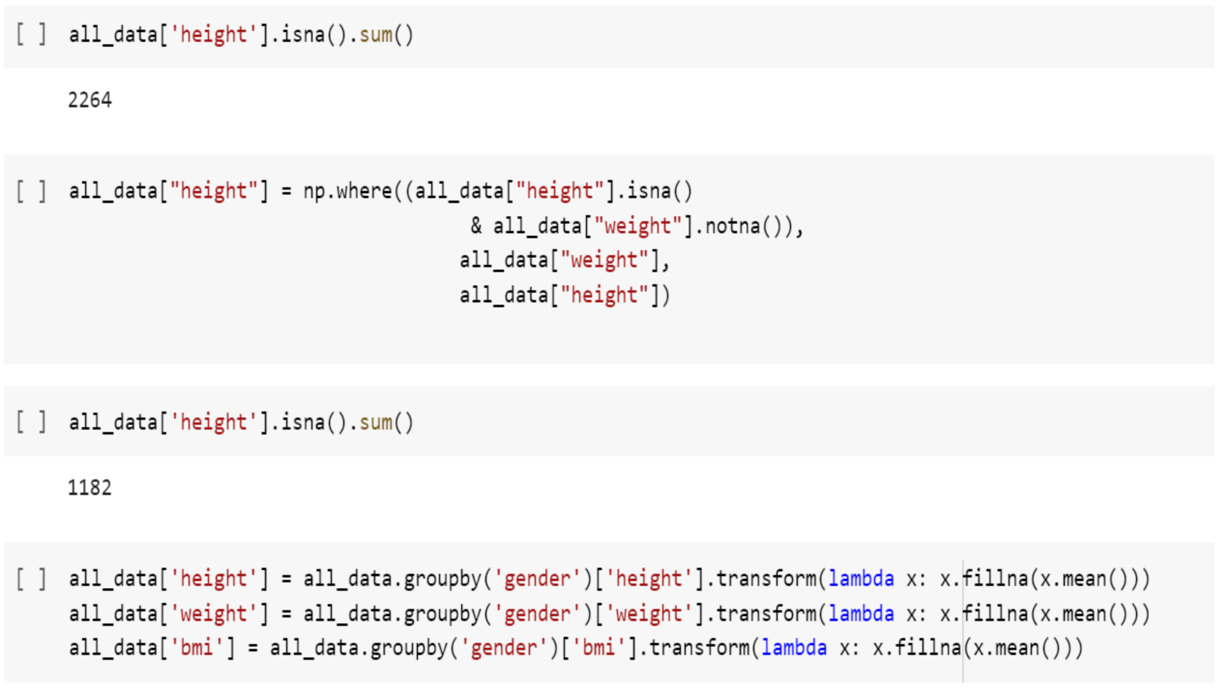
	Missing Values	% of Total Values
h1_bilirubin_min	119861	92.100000
h1_bilirubin_max	119861	92.100000
h1_albumin_max	119005	91.400000
h1_albumin_min	119005	91.400000
h1_lactate_max	118467	91.000000
h1_lactate_min	118467	91.000000
h1_pao2fio2ratio_min	113397	87.100000
h1_pao2fio2ratio_max	113397	87.100000
h1_arterial_ph_max	107849	82.900000
h1_arterial_ph_min	107849	82.900000
h1_arterial_pco2_min	107666	82.700000

Figure 5.12: Finding missing data



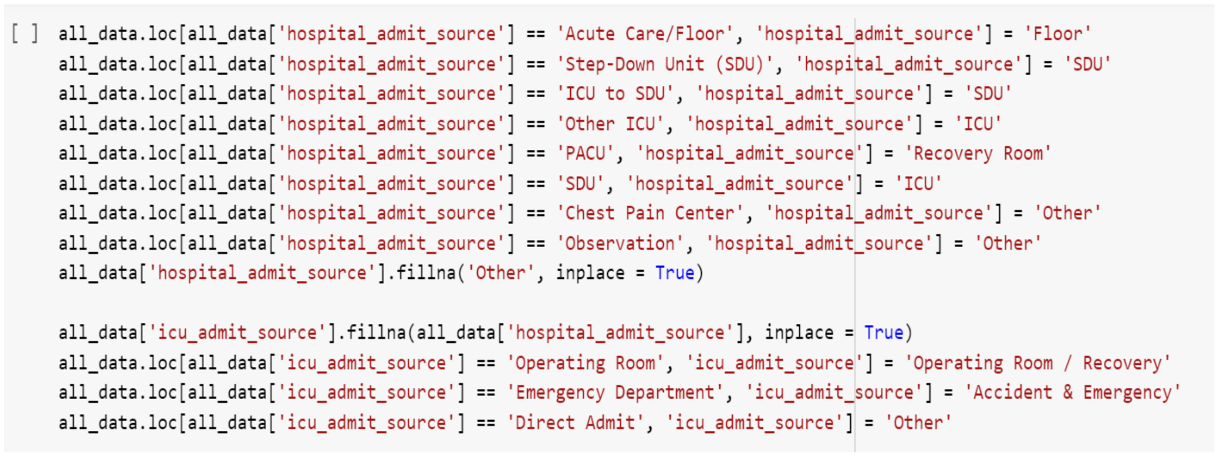
```
[ ] all_data["weight"].isna().sum()

    4211

[ ] all_data["weight"] = np.where((all_data["weight"].isna()
                                   & all_data["bmi"].notna()),
                                  all_data["bmi"],
                                  all_data["weight"])
```

Figure 5.13: Filling weight columns



```
[ ] all_data['height'].isna().sum()

    2264

[ ] all_data["height"] = np.where((all_data["height"].isna()
                                   & all_data["weight"].notna()),
                                  all_data["weight"],
                                  all_data["height"])

[ ] all_data['height'].isna().sum()

    1182

[ ] all_data['height'] = all_data.groupby('gender')['height'].transform(lambda x: x.fillna(x.mean()))
    all_data['weight'] = all_data.groupby('gender')['weight'].transform(lambda x: x.fillna(x.mean()))
    all_data['bmi'] = all_data.groupby('gender')['bmi'].transform(lambda x: x.fillna(x.mean()))
```

Figure 5.14: Filling null columns

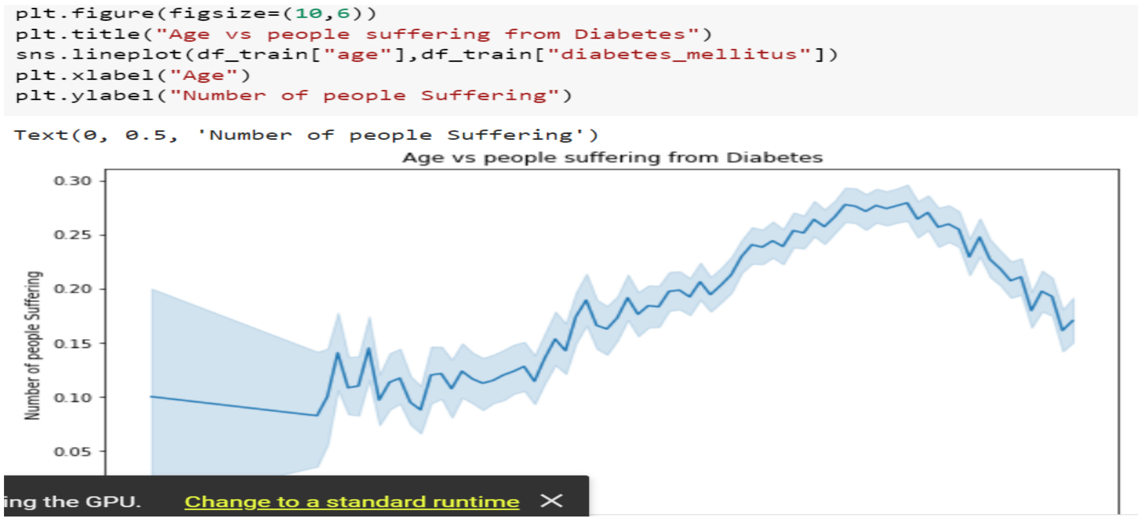


```
[ ]  all_data.loc[all_data['hospital_admit_source'] == 'Acute Care/Floor', 'hospital_admit_source'] = 'Floor'
     all_data.loc[all_data['hospital_admit_source'] == 'Step-Down Unit (SDU)', 'hospital_admit_source'] = 'SDU'
     all_data.loc[all_data['hospital_admit_source'] == 'ICU to SDU', 'hospital_admit_source'] = 'SDU'
     all_data.loc[all_data['hospital_admit_source'] == 'Other ICU', 'hospital_admit_source'] = 'ICU'
     all_data.loc[all_data['hospital_admit_source'] == 'PACU', 'hospital_admit_source'] = 'Recovery Room'
     all_data.loc[all_data['hospital_admit_source'] == 'SDU', 'hospital_admit_source'] = 'ICU'
     all_data.loc[all_data['hospital_admit_source'] == 'Chest Pain Center', 'hospital_admit_source'] = 'Other'
     all_data.loc[all_data['hospital_admit_source'] == 'Observation', 'hospital_admit_source'] = 'Other'
     all_data['hospital_admit_source'].fillna('Other', inplace = True)

     all_data['icu_admit_source'].fillna(all_data['hospital_admit_source'], inplace = True)
     all_data.loc[all_data['icu_admit_source'] == 'Operating Room', 'icu_admit_source'] = 'Operating Room / Recovery'
     all_data.loc[all_data['icu_admit_source'] == 'Emergency Department', 'icu_admit_source'] = 'Accident & Emergency'
     all_data.loc[all_data['icu_admit_source'] == 'Direct Admit', 'icu_admit_source'] = 'Other'
```

Figure 5.15: Replacing columns

Visualizing the Data

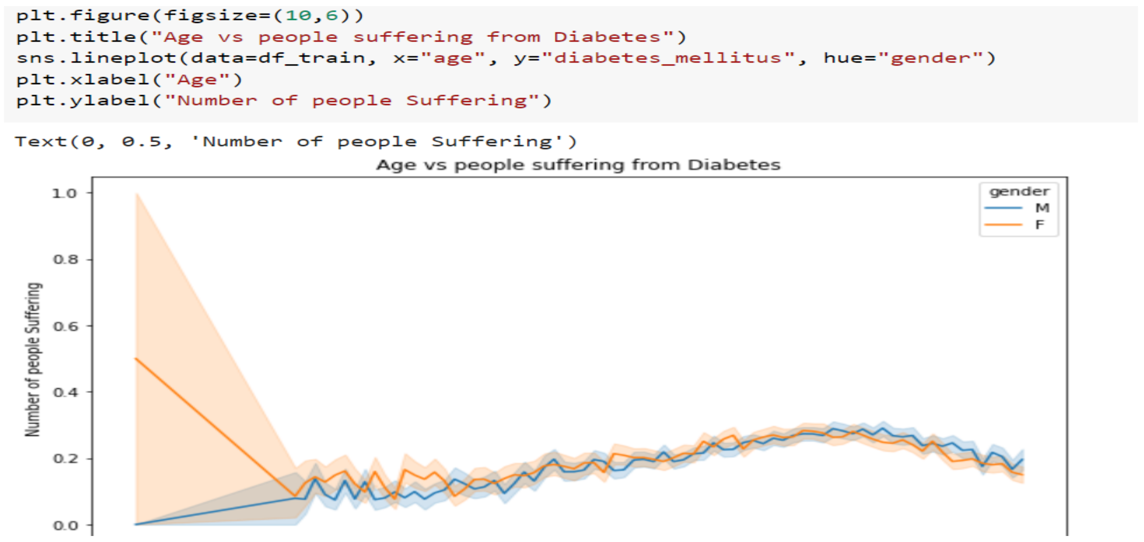
The data can be visualized for finding the target people by using matplotlib library as shown in the fig 5.16 and fig 5.17



```
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')
```

Figure 5.16: Visualizing number of diabetic suffered people



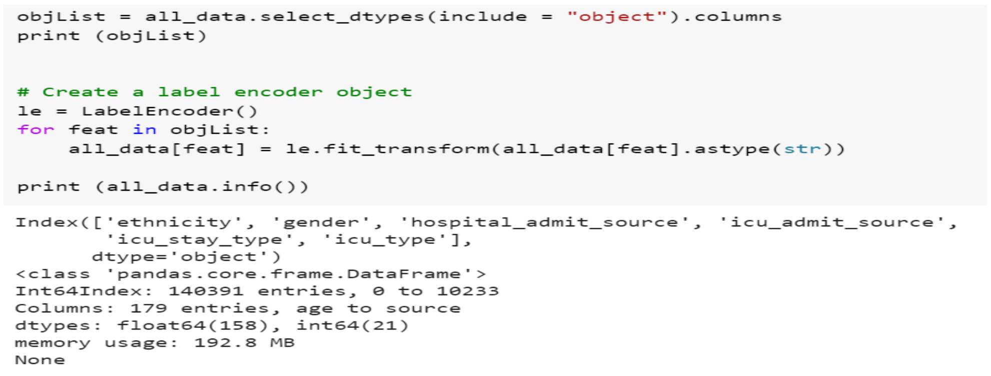
```
plt.figure(figsize=(10,6))
plt.title("Age vs people suffering from Diabetes")
sns.lineplot(data=df_train, x="age", y="diabetes_mellitus", hue="gender")
plt.xlabel("Age")
plt.ylabel("Number of people Suffering")
```

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

Figure 5.17: Visualizing Age wise Diabetes suffering

Encoding the Categorical Data

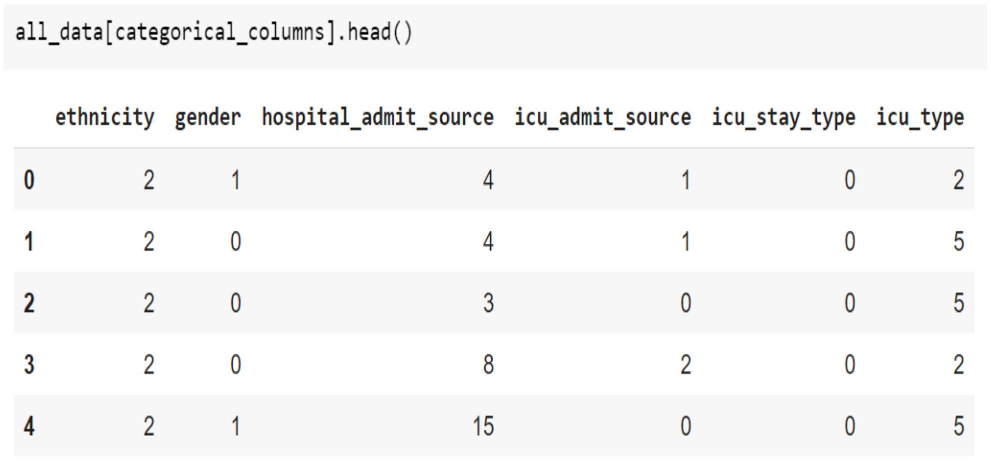
Encoding categorical values with a technique called ”label encoding”, which allows you to convert each value in a column to a number as shown in fig 5.18 and fig 5.19



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


# Create 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
```

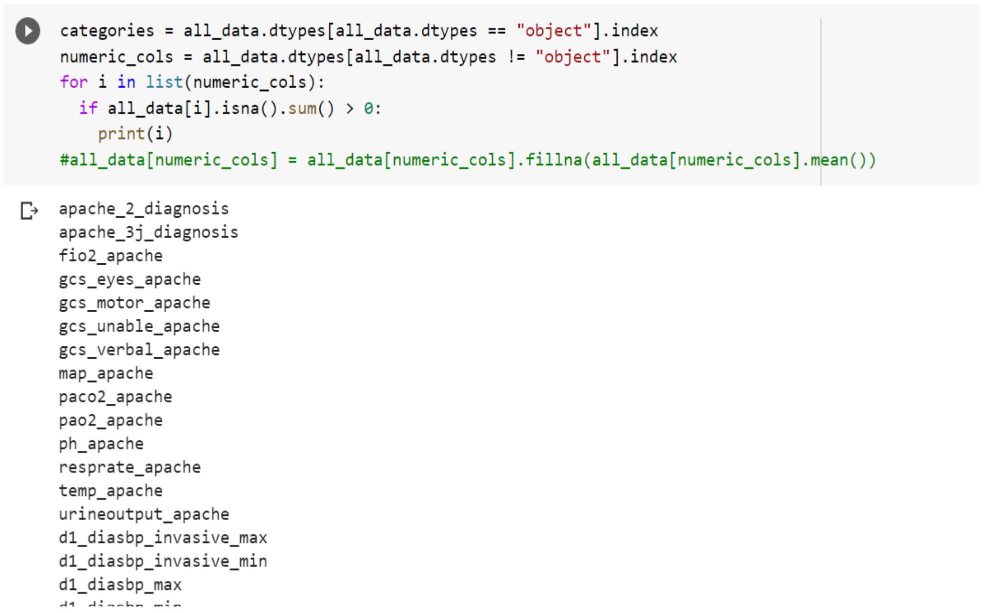
Figure 5.18: Code for Encoding Categorical Data



```
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

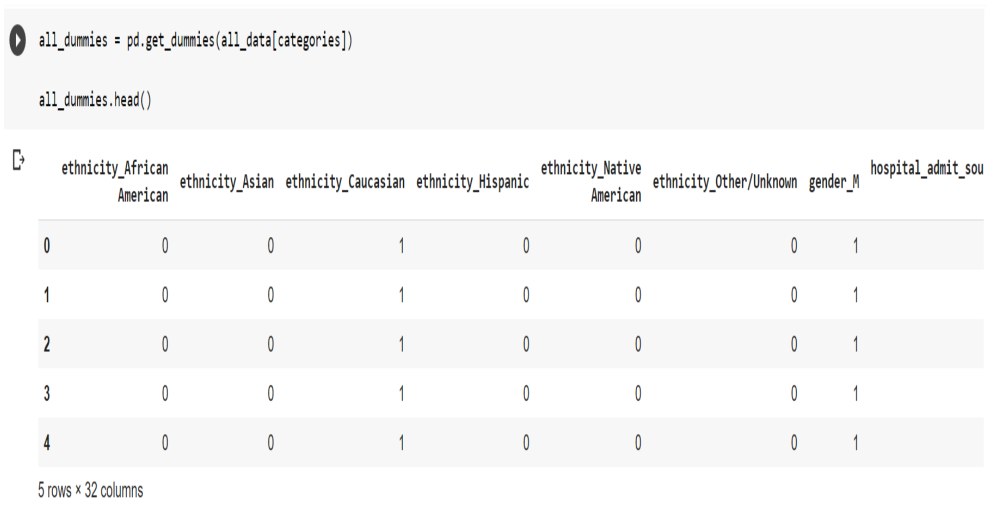
Figure 5.19: Encoded data



```
categories = all_data.dtypes[all_data.dtypes == "object"].index
numeric_cols = all_data.dtypes[all_data.dtypes != "object"].index
for i in list(numeric_cols):
  if all_data[i].isna().sum() > 0:
    print(i)
#all_data[numeric_cols] = all_data[numeric_cols].fillna(all_data[numeric_cols].mean())
```

```
apache_2_diagnosis
apache_3j_diagnosis
fio2_apache
gcs_eyes_apache
gcs_motor_apache
gcs_unable_apache
gcs_verbal_apache
map_apache
paco2_apache
pao2_apache
ph_apache
resprate_apache
temp_apache
urineoutput_apache
d1_diasbp_invasive_max
d1_diasbp_invasive_min
d1_diasbp_max
```

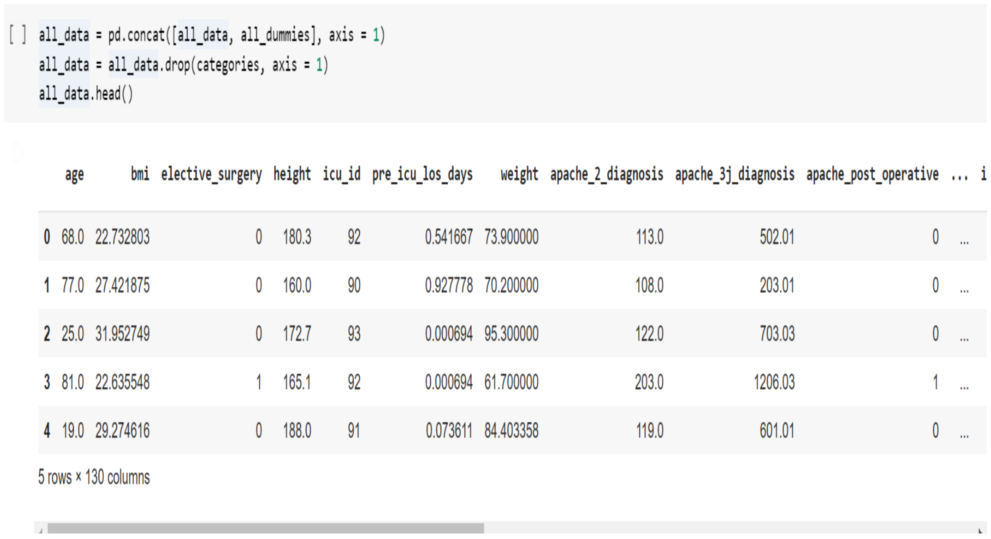
Figure 5.20: Getting categorical data



```python
all_dummies = pd.get_dummies(all_data[categories])

all_dummies.head()
```

```
   ethnicity_African ethnicity_Asian ethnicity_Caucasian ethnicity_Hispanic ethnicity_Native ethnicity_Other/Unknown gender_M hospital_admit_sou
            American                                                                American
0                  0               0                   1                  0                0                       0        1
1                  0               0                   1                  0                0                       0        1
2                  0               0                   1                  0                0                       0        1
3                  0               0                   1                  0                0                       0        1
4                  0               0                   1                  0                0                       0        1
5 rows × 32 columns
```

Figure 5.21: Get dummies



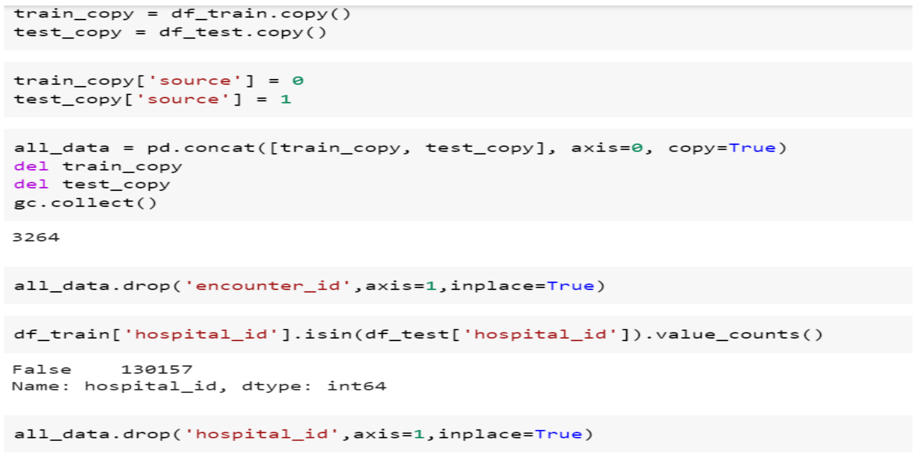
```python
all_data = pd.concat([all_data, all_dummies], axis = 1)
all_data = all_data.drop(categories, axis = 1)
all_data.head()
```

```
   age        bmi elective_surgery height icu_id pre_icu_los_days    weight apache_2_diagnosis apache_3j_diagnosis apache_post_operative ... i
0 68.0  22.732803                0  180.3     92         0.541667 73.900000              113.0              502.01                     0 ...
1 77.0  27.421875                0  160.0     90         0.927778 70.200000              108.0              203.01                     0 ...
2 25.0  31.952749                0  172.7     93         0.000694 95.300000              122.0              703.03                     0 ...
3 81.0  22.635548                1  165.1     92         0.000694 61.700000              203.0             1206.03                     1 ...
4 19.0  29.274616                0  188.0     91         0.073611 84.403358              119.0              601.01                     0 ...
5 rows × 130 columns
```

Figure 5.22: Dropping categorical data

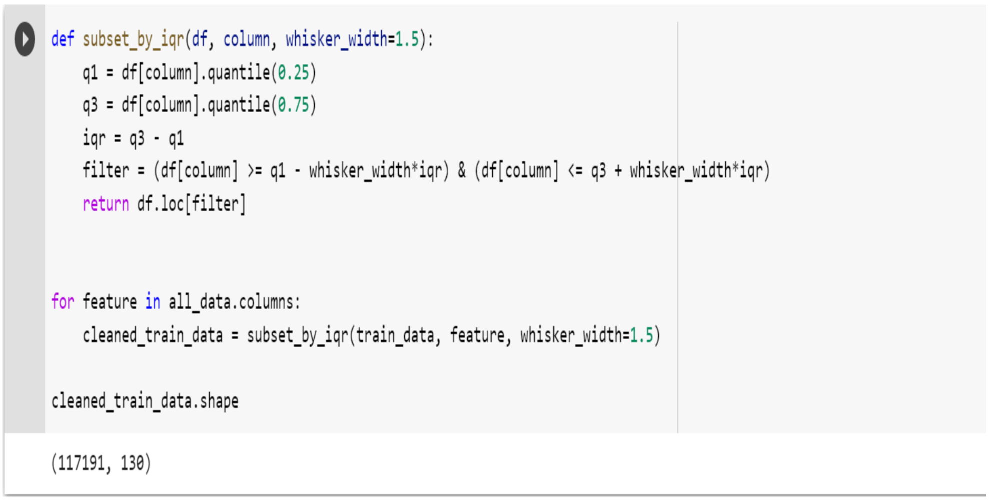
Preprocessing the Data

Data preprocessing can refer to manipulation or dropping of data before it is used in order to ensure or enhance performance as shown in the fig 5.23.



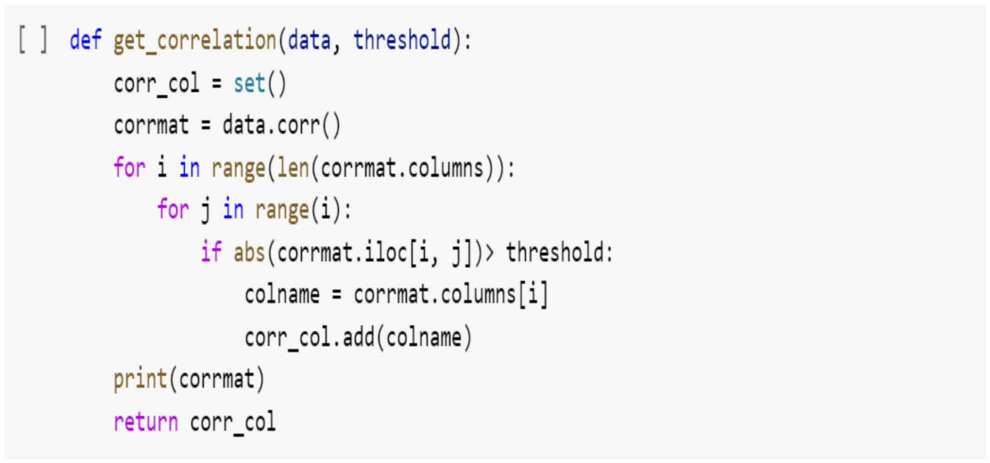
```
train_copy = df_train.copy()
test_copy = df_test.copy()

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()

3264

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

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

False     130157
Name: hospital_id, dtype: int64

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

Figure 5.23: Preprocessing Data



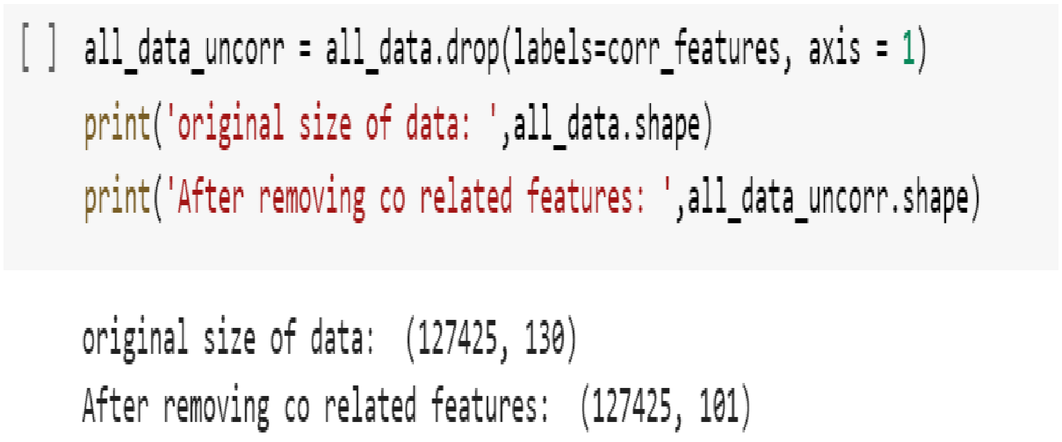
```
def subset_by_iqr(df, column, whisker_width=1.5):
    q1 = df[column].quantile(0.25)
    q3 = df[column].quantile(0.75)
    iqr = q3 - q1
    filter = (df[column] >= q1 - whisker_width*iqr) & (df[column] <= q3 + whisker_width*iqr)
    return df.loc[filter]


for feature in all_data.columns:
    cleaned_train_data = subset_by_iqr(train_data, feature, whisker_width=1.5)

cleaned_train_data.shape

(117191, 130)
```

Figure 5.24: Cleaning data



```
[ ] def get_correlation(data, threshold):
        corr_col = set()
        corrmat = data.corr()
        for i in range(len(corrmat.columns)):
            for j in range(i):
                if abs(corrmat.iloc[i, j])> threshold:
                    colname = corrmat.columns[i]
                    corr_col.add(colname)
        print(corrmat)
        return corr_col
```

Figure 5.25: Getting correlated data

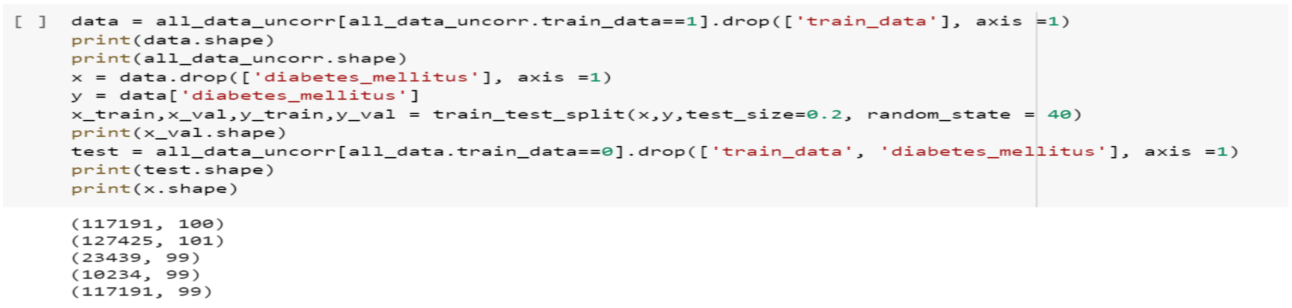


```
[ ] all_data_uncorr = all_data.drop(labels=corr_features, axis = 1)
    print('original size of data: ',all_data.shape)
    print('After removing co related features: ',all_data_uncorr.shape)

    original size of data:  (127425, 130)
    After removing co related features:  (127425, 101)
```

Figure 5.26: Dropping correlated data

Splitting the Data

We need to split a dataset into train and test sets to evaluate how well our machine learning model performs. The train set is used to fit the model, the statistics of the train set are known. The second set is called the test data set, this set is solely used for predictions as shown in fig 5.27



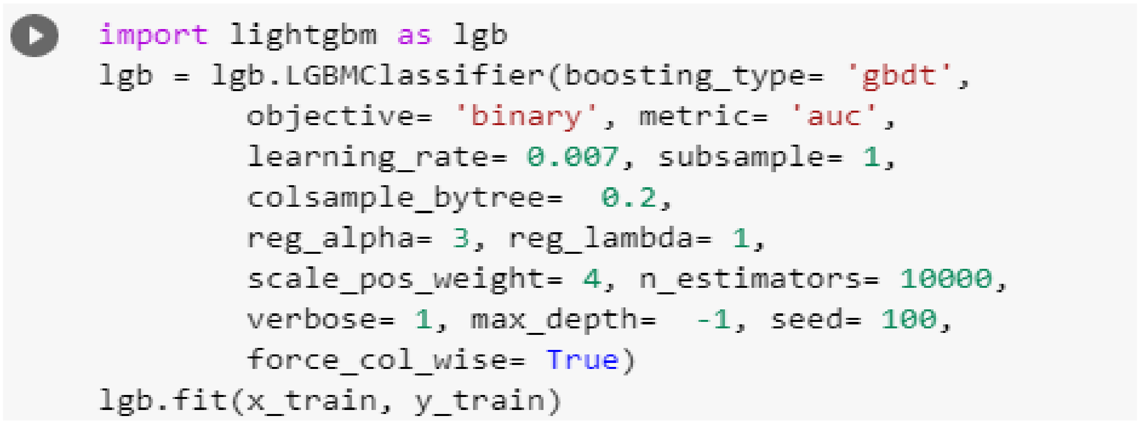
```
[ ]  data = all_data_uncorr[all_data_uncorr.train_data==1].drop(['train_data'], axis =1)
     print(data.shape)
     print(all_data_uncorr.shape)
     x = data.drop(['diabetes_mellitus'], axis =1)
     y = data['diabetes_mellitus']
     x_train,x_val,y_train,y_val = train_test_split(x,y,test_size=0.2, random_state = 40)
     print(x_val.shape)
     test = all_data_uncorr[all_data.train_data==0].drop(['train_data', 'diabetes_mellitus'], axis =1)
     print(test.shape)
     print(x.shape)

     (117191, 100)
     (127425, 101)
     (23439, 99)
     (10234, 99)
     (117191, 99)
```

Figure 5.27: Splitting of Data

Algorithm Implementation

The model can be built by using various machine learning algorithms. The algorithms that we used for building this model is Logistic Regression, KNN, XG boost, Random Forest and Light GBM. We can fit the model by using these algorithms as shown in the figures 4.2,4.4,4.6,4.8,4.11.

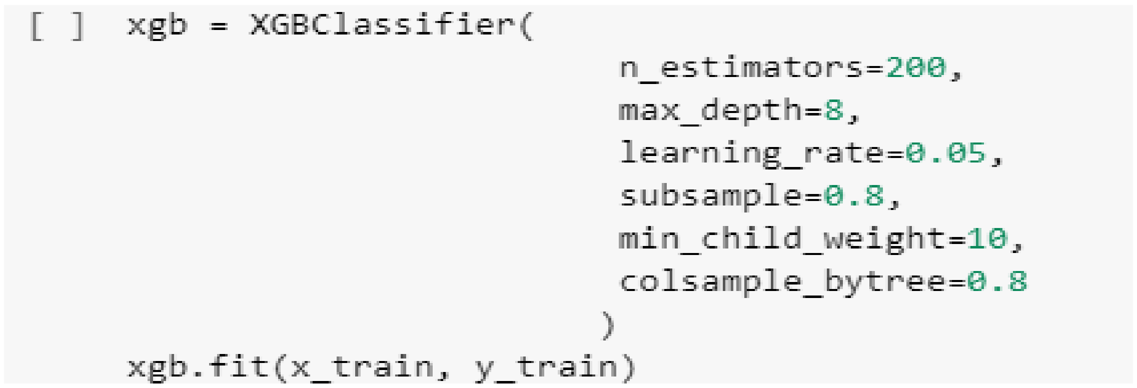
In our model to improve Light GBM performance hyper parameter tuning is implemented and fitted with the classifier as shown in fig 5.28



```
import lightgbm as lgb
lgb = lgb.LGBMClassifier(boosting_type= 'gbdt',
        objective= 'binary', metric= 'auc',
        learning_rate= 0.007, subsample= 1,
        colsample_bytree=  0.2,
        reg_alpha= 3, reg_lambda= 1,
        scale_pos_weight= 4, n_estimators= 10000,
        verbose= 1, max_depth=  -1, seed= 100,
        force_col_wise= True)
lgb.fit(x_train, y_train)
```

Figure 5.28: LightGBM with Hyper Parameter Tuning

In our model to improve XGB performance hyper parameter tuning is implemented and fitted with the classifier as shown in fig 5.29



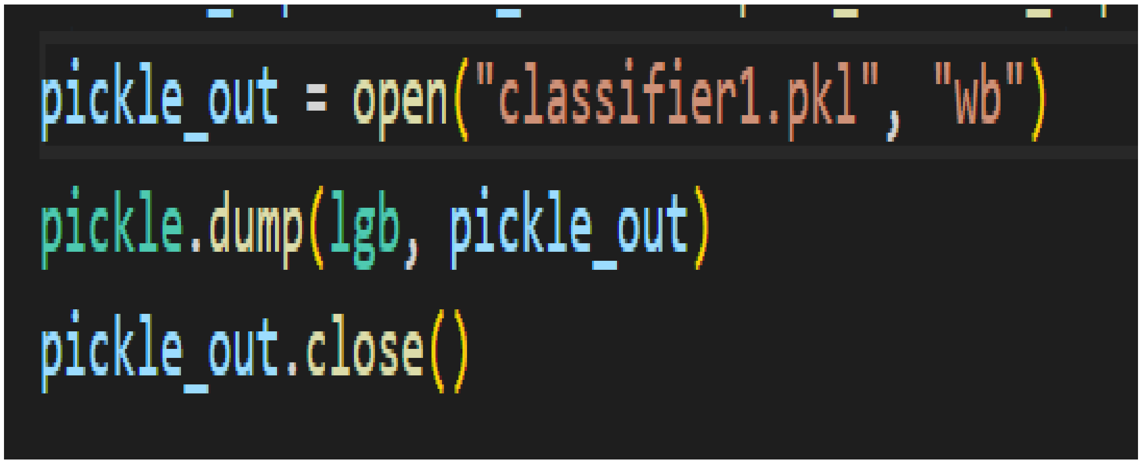
```
[ ]  xgb = XGBClassifier(
                            n_estimators=200,
                            max_depth=8,
                            learning_rate=0.05,
                            subsample=0.8,
                            min_child_weight=10,
                            colsample_bytree=0.8
                           )
     xgb.fit(x_train, y_train)
```

Figure 5.29: XGB with Hyper Parameter Tuning

Streamlit Implementation

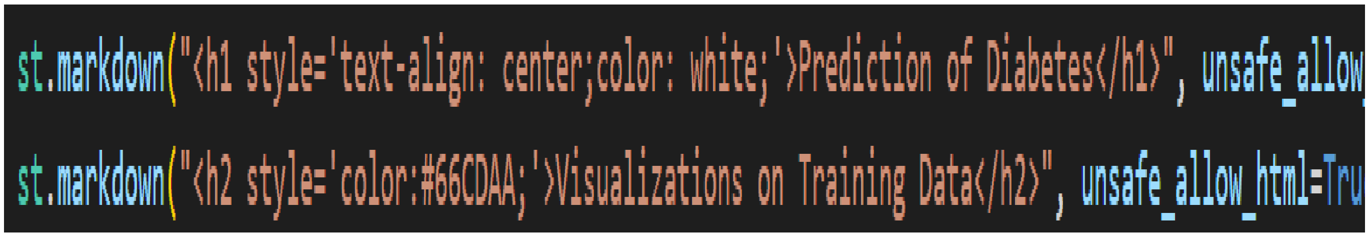
Streamlit is used for making our model an interactive Data Science Dashboard.

The streamlit implementation is shown in fig 5.30,5.31,5.32 and 5.33.



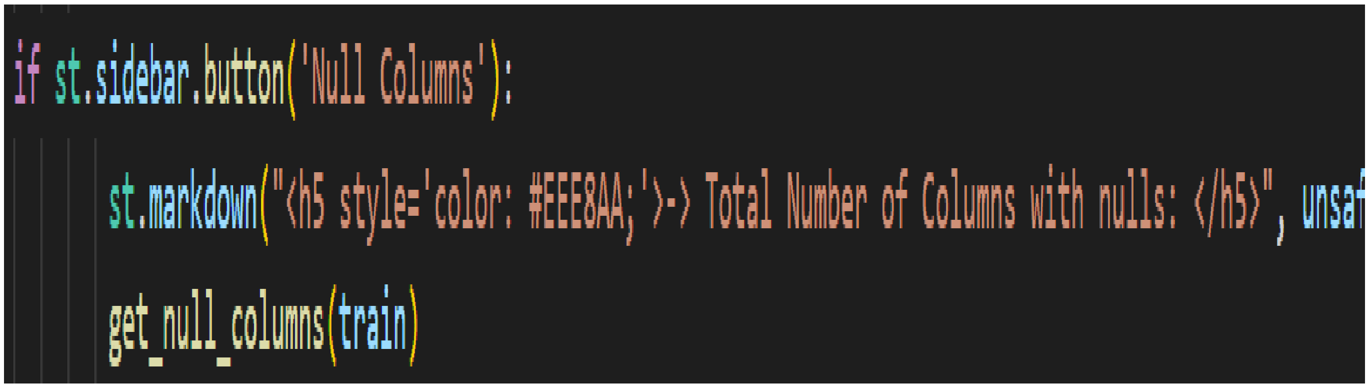
```
pickle_out = open("classifier1.pkl", "wb")
pickle.dump(lgb, pickle_out)
pickle_out.close()
```

Figure 5.30: Loading classifier



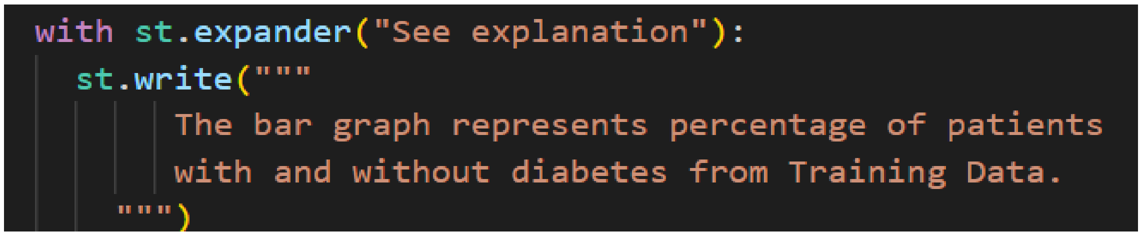
```
st.markdown("<h1 style='text-align: center;color: white;'>Prediction of Diabetes</h1>", unsafe_allow
st.markdown("<h2 style='color:#66CDAA;'>Visualizations on Training Data</h2>", unsafe_allow_html=Tru
```

Figure 5.31: Streamlit title implementation



```
if st.sidebar.button('Null Columns'):
        st.markdown("<h5 style='color: #EEE8AA;'>-> Total Number of Columns with nulls: </h5>", unsaf
        get_null_columns(train)
```

Figure 5.32: Streamlit button implementation



```
with st.expander("See explanation"):
    st.write("""
            The bar graph represents percentage of patients
            with and without diabetes from Training Data.
        """)
```

Figure 5.33: Streamlit expander implementation

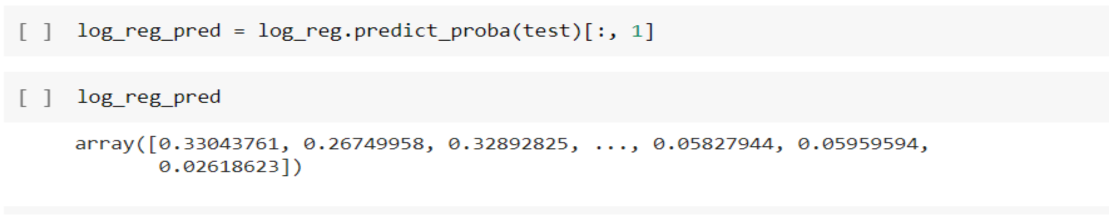
5.2 Results

The Results of the Model by using various machine learning algorithms can be considered in two phases:

- Prediction
- Accuracy Score

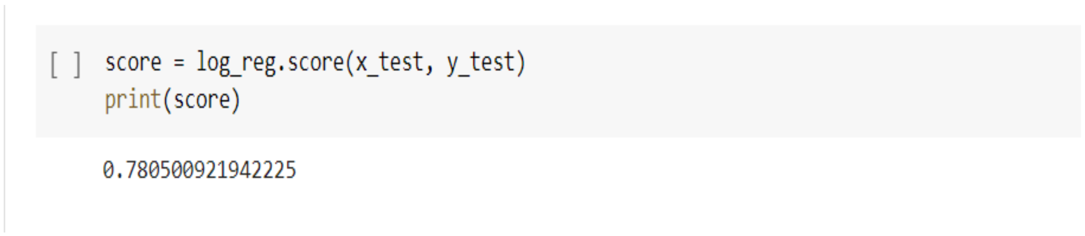
Predictions and Results using Logistic Regression

By using the logistic regression the predictions are shown in fig 5.34 and with this predictions the accuracy of this algorithm can be measured by score function as shown in fig 5.35 and the accuracy is value 78 percent.



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

[ ]  log_reg_pred

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

Figure 5.34: Predictions using Logistic Regression



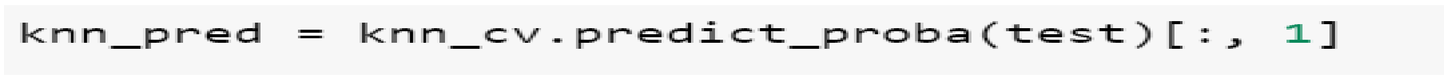
```
[ ]  score = log_reg.score(x_test, y_test)
     print(score)

     0.780500921942225
```

Figure 5.35: Accuracy Score using Logistic Regression

Predictions and Results using KNN

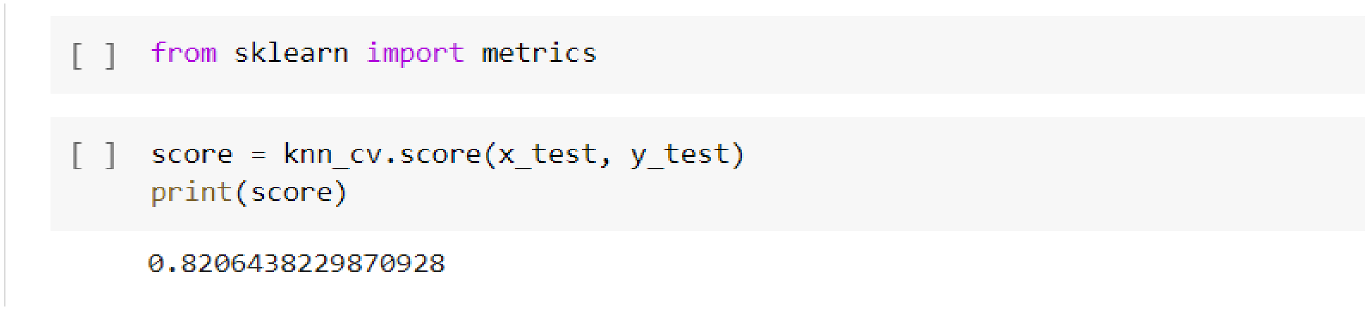
So in order to have more accuracy we implemented KNN algorithm

By using the KNN the predictions are shown in fig 5.36 and with these predictions the accuracy of this algorithm can be measured by score function as shown in fig 5.37 and the accuracy is value 82 percent.



```
knn_pred = knn_cv.predict_proba(test)[:, 1]
```

Figure 5.36: Predictions using KNN

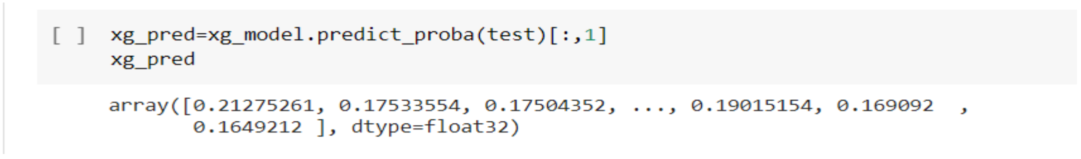


```
[ ]  from sklearn import metrics

[ ]  score = knn_cv.score(x_test, y_test)
     print(score)

     0.8206438229870928
```

Figure 5.37: Accuracy Score using KNN

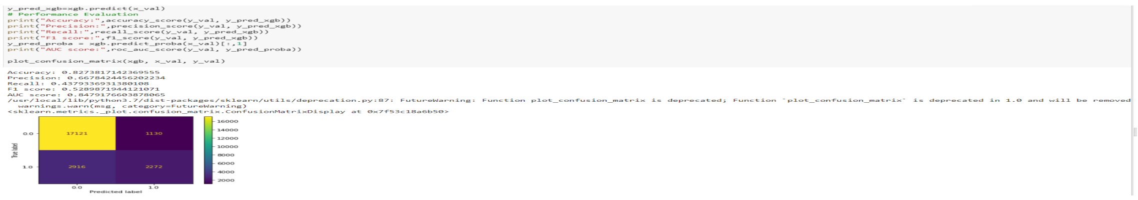
Predictions and Results using XG Boost

So in order to have more accuracy and better performance we implemented XGBoost algorithm By using the XGBoost the predictions are shown in fig 5.38 and with these predictions the accuracy of this algorithm can be measured by score function as shown in fig 5.39 and the accuracy is value 79 percent.



```
[ ]  xg_pred=xg_model.predict_proba(test)[:,1]
     xg_pred

     array([0.21275261, 0.17533554, 0.17504352, ..., 0.19015154, 0.169092  ,
            0.1649212 ], dtype=float32)
```

Figure 5.38: Predictions using XGBoost

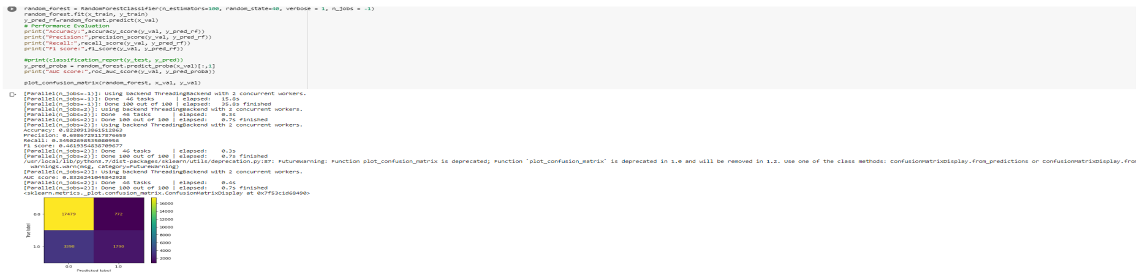


```
y_pred_xgb=xgb.predict(x_val)
# Performance Evaluation
print("Accuracy:",accuracy_score(y_val, y_pred_xgb))
print("Precision:",precision_score(y_val, y_pred_xgb))
print("Recall:",recall_score(y_val, y_pred_xgb))
print("F1 score:",f1_score(y_val, y_pred_xgb))
y_pred_proba = xgb.predict_proba(x_val)[:,1]
print("AUC score:",roc_auc_score(y_val, y_pred_proba))

plot_confusion_matrix(xgb, x_val, y_val)

Accuracy: 0.8273817142369555
Precision: 0.6678424456202234
Recall: 0.4379336931380108
F1 score: 0.5289871944121071
AUC score: 0.8479176603878065
/usr/local/lib/python3.7/dist-packages/sklearn/utils/deprecation.py:87: FutureWarning: Function plot_confusion_matrix is deprecated; Function `plot_confusion_matrix` is deprecated in 1.0 and will be removed
  warnings.warn(msg, category=FutureWarning)
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7f53c18a6b50>
```

Figure 5.39: Accuracy score using XGBoost

Predictions and Results using Random Forest

But the accuracy decreased and the performance of XGBoost is low on this dataset. So in order to have more accuracy and better performance we implemented Random Forest algorithm By using the Random Forest the predictions and with these predictions the accuracy of this algorithm can be measured by score function as shown in fig 5.40

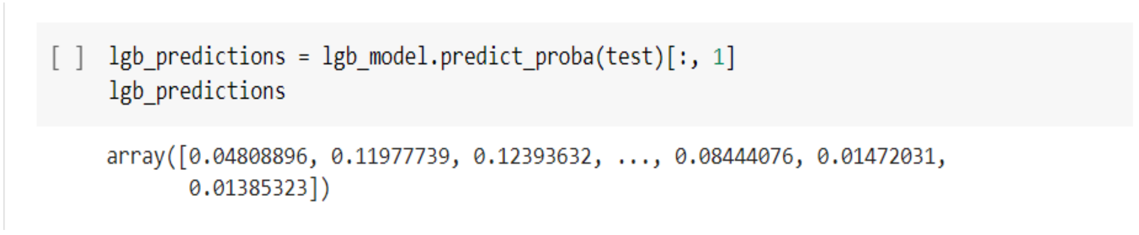


```
random_forest = RandomForestClassifier(n_estimators=100, random_state=40, verbose = 1, n_jobs = -1)
random_forest.fit(x_train, y_train)
y_pred_rf=random_forest.predict(x_val)
# Performance Evaluation
print("Accuracy:",accuracy_score(y_val, y_pred_rf))
print("Precision:",precision_score(y_val, y_pred_rf))
print("Recall:",recall_score(y_val, y_pred_rf))
print("F1 score:",f1_score(y_val, y_pred_rf))

#print(classification_report(y_test, y_pred))
y_pred_proba = random_forest.predict_proba(x_val)[:,1]
print("AUC score:",roc_auc_score(y_val, y_pred_proba))

plot_confusion_matrix(random_forest, x_val, y_val)

[Parallel(n_jobs=-1)]: Using backend ThreadingBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done  46 tasks      | elapsed:   15.8s
[Parallel(n_jobs=-1)]: Done 100 out of 100 | elapsed:   35.8s 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
[Parallel(n_jobs=2)]: Using backend ThreadingBackend with 2 concurrent workers.
Accuracy: 0.8220913861512863
Precision: 0.6986729117876659
Recall: 0.34502698535080956
F1 score: 0.46193548387096777
[Parallel(n_jobs=2)]: Done  46 tasks      | elapsed:    0.3s
[Parallel(n_jobs=2)]: Done 100 out of 100 | elapsed:    0.7s finished
/usr/local/lib/python3.7/dist-packages/sklearn/utils/deprecation.py:87: FutureWarning: Function plot_confusion_matrix is deprecated; Function `plot_confusion_matrix` is deprecated in 1.0 and will be removed in 1.2. Use one of the class methods: ConfusionMatrixDisplay.from_predictions or ConfusionMatrixDisplay.fro
  warnings.warn(msg, category=FutureWarning)
[Parallel(n_jobs=2)]: Using backend ThreadingBackend with 2 concurrent workers.
AUC score: 0.8326241045842928
[Parallel(n_jobs=2)]: Done  46 tasks      | elapsed:    0.4s
[Parallel(n_jobs=2)]: Done 100 out of 100 | elapsed:    0.7s finished
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7f53c1d68490>
```

Figure 5.40: Predictions and Accuracy score using Random Forest

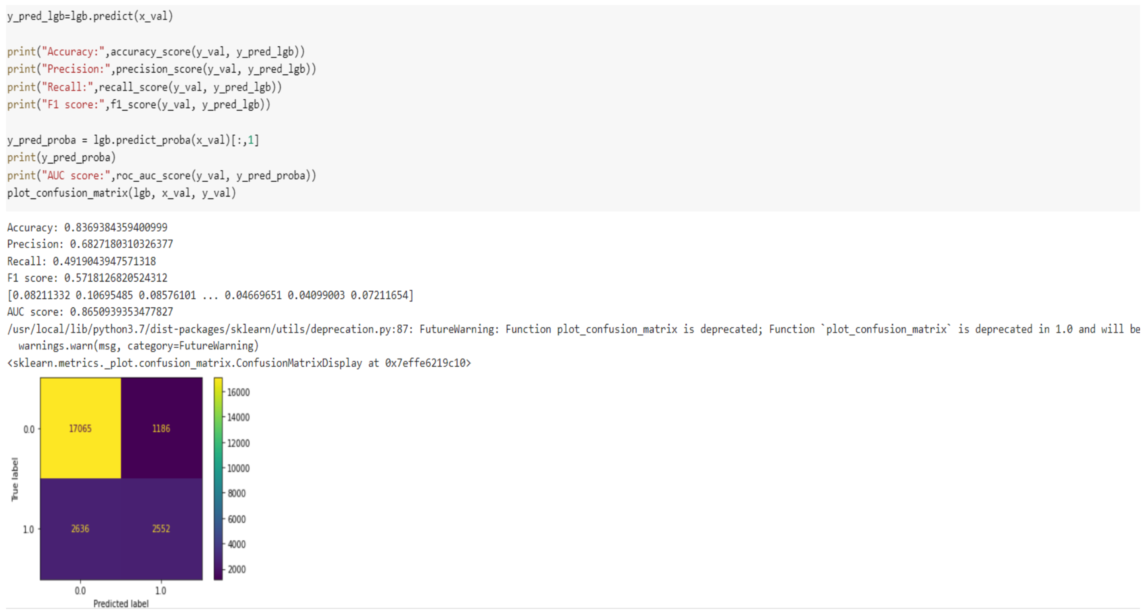
Predictions and Results using Light GBM

Even with this algorithm the predictions are not accurate so we implemented a Light GBM algorithm. By using the Light GBM the predictions are shown in fig 5.41 and with these predictions the accuracy of this algorithm can be measured by score function as shown in fig 5.42 and the accuracy is value 84 percent.



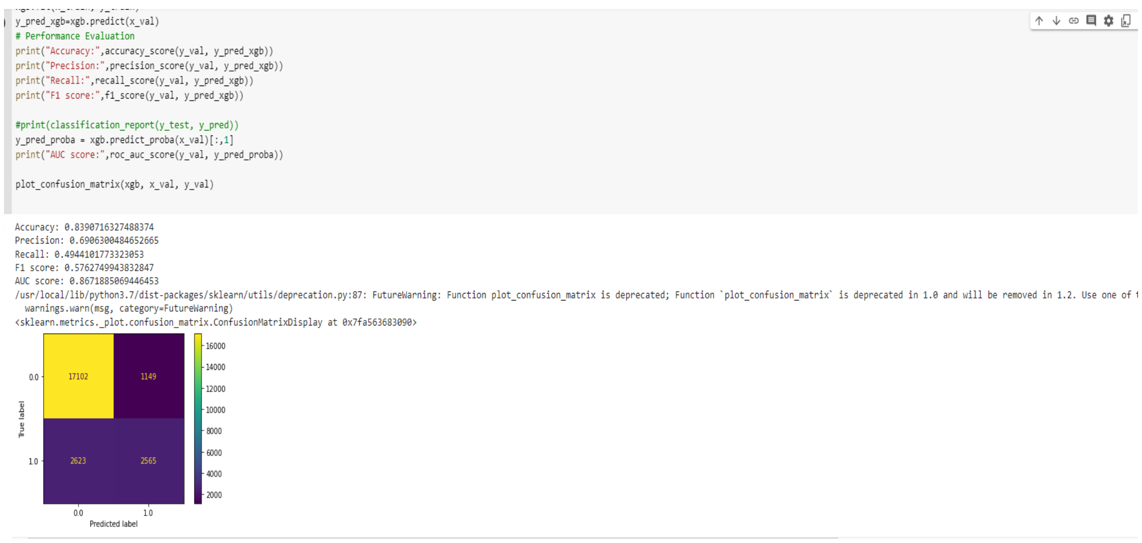
```
[ ] lgb_predictions = lgb_model.predict_proba(test)[:, 1]
    lgb_predictions

    array([0.04808896, 0.11977739, 0.12393632, ..., 0.08444076, 0.01472031,
           0.01385323])
```

Figure 5.41: Predictions using Light GBM



```
y_pred_lgb=lgb.predict(x_val)

print("Accuracy:",accuracy_score(y_val, y_pred_lgb))
print("Precision:",precision_score(y_val, y_pred_lgb))
print("Recall:",recall_score(y_val, y_pred_lgb))
print("F1 score:",f1_score(y_val, y_pred_lgb))

y_pred_proba = lgb.predict_proba(x_val)[:,1]
print(y_pred_proba)
print("AUC score:",roc_auc_score(y_val, y_pred_proba))
plot_confusion_matrix(lgb, x_val, y_val)

Accuracy: 0.8369384359400999
Precision: 0.6827180310326377
Recall: 0.4919043947571318
F1 score: 0.5718126820524312
[0.08211332 0.10695485 0.08576101 ... 0.04669651 0.04099003 0.07211654]
AUC score: 0.8650939353477827
/usr/local/lib/python3.7/dist-packages/sklearn/utils/deprecation.py:87: FutureWarning: Function plot_confusion_matrix is deprecated; Function `plot_confusion_matrix` is deprecated in 1.0 and will be
  warnings.warn(msg, category=FutureWarning)
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7effe6219c10>
```

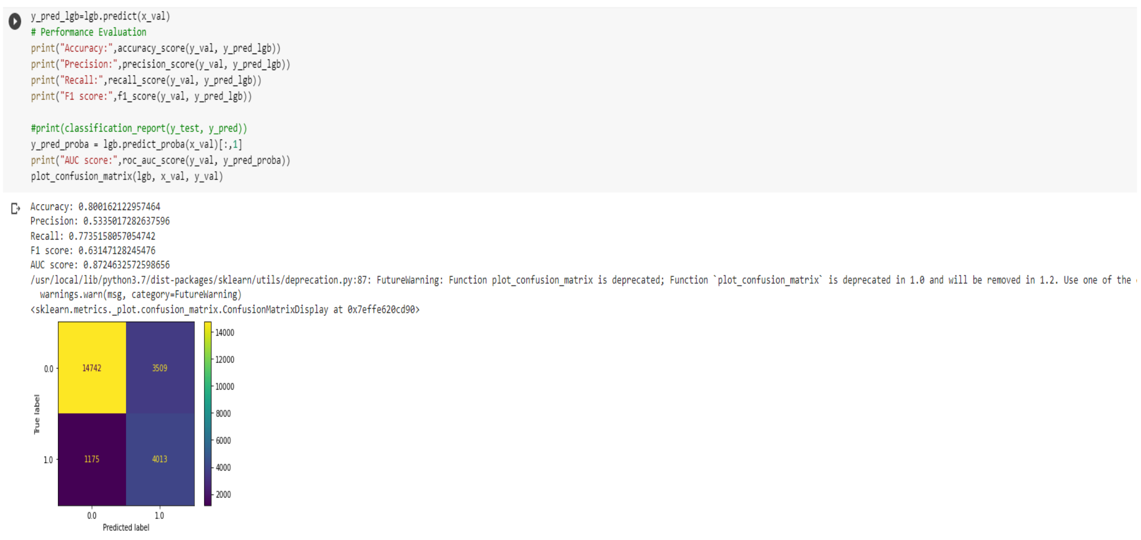
Figure 5.42: Accuracy Score using Light GBM



```
y_pred_xgb=xgb.predict(x_val)
# Performance Evaluation
print("Accuracy:",accuracy_score(y_val, y_pred_xgb))
print("Precision:",precision_score(y_val, y_pred_xgb))
print("Recall:",recall_score(y_val, y_pred_xgb))
print("F1 score:",f1_score(y_val, y_pred_xgb))

#print(classification_report(y_test, y_pred))
y_pred_proba = xgb.predict_proba(x_val)[:,1]
print("AUC score:",roc_auc_score(y_val, y_pred_proba))

plot_confusion_matrix(xgb, x_val, y_val)
```

```
Accuracy: 0.8390716327488374
Precision: 0.6906300484652665
Recall: 0.4944101773323053
F1 score: 0.5762749943832847
AUC score: 0.8671885069446453
/usr/local/lib/python3.7/dist-packages/sklearn/utils/deprecation.py:87: FutureWarning: Function plot_confusion_matrix is deprecated; Function `plot_confusion_matrix` is deprecated in 1.0 and will be removed in 1.2. Use one of t
  warnings.warn(msg, category=FutureWarning)
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7fa563683090>
```

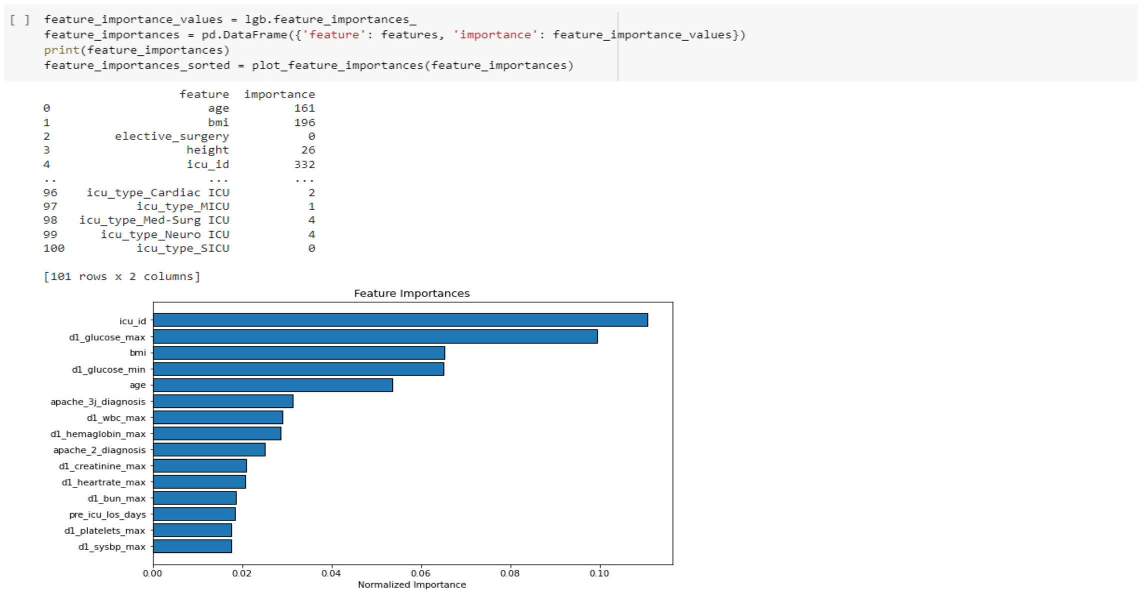
Figure 5.43: Accuracy Score using XGB with Hyper Parameter Tuning



```
y_pred_lgb=lgb.predict(x_val)
# Performance Evaluation
print("Accuracy:",accuracy_score(y_val, y_pred_lgb))
print("Precision:",precision_score(y_val, y_pred_lgb))
print("Recall:",recall_score(y_val, y_pred_lgb))
print("F1 score:",f1_score(y_val, y_pred_lgb))

#print(classification_report(y_test, y_pred))
y_pred_proba = lgb.predict_proba(x_val)[:,1]
print("AUC score:",roc_auc_score(y_val, y_pred_proba))
plot_confusion_matrix(lgb, x_val, y_val)
```

```
Accuracy: 0.800162122957464
Precision: 0.5335017282637596
Recall: 0.7735158057054742
F1 score: 0.63147128245476
AUC score: 0.8724632572598656
/usr/local/lib/python3.7/dist-packages/sklearn/utils/deprecation.py:87: FutureWarning: Function plot_confusion_matrix is deprecated; Function `plot_confusion_matrix` is deprecated in 1.0 and will be removed in 1.2. Use one of the
  warnings.warn(msg, category=FutureWarning)
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7effe620cd90>
```

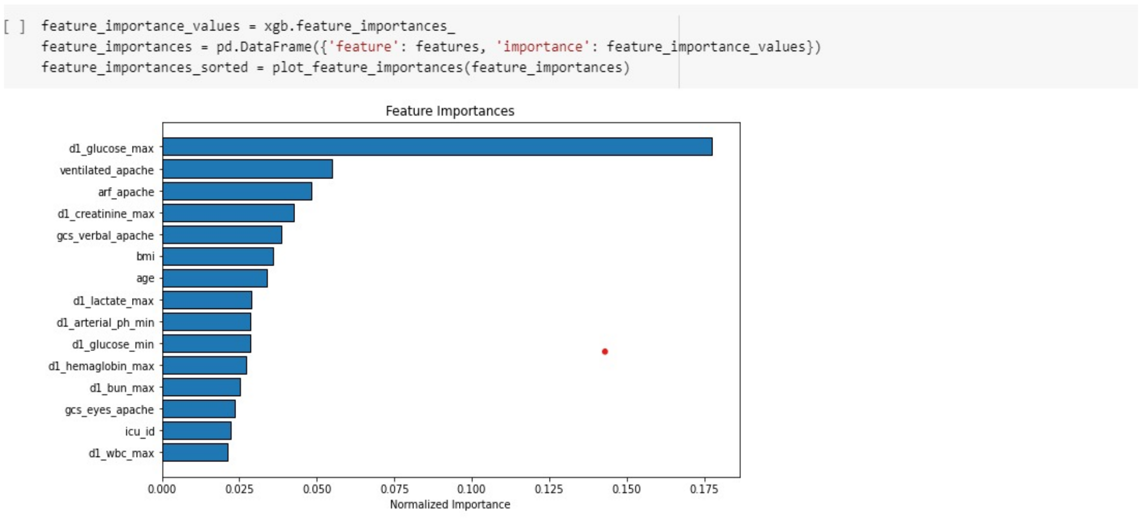
Figure 5.44: Accuracy Score using Light GBM with Hyper Parameter Tuning



```
[ ]  feature_importance_values = lgb.feature_importances_
     feature_importances = pd.DataFrame({'feature': features, 'importance': feature_importance_values})
     print(feature_importances)
     feature_importances_sorted = plot_feature_importances(feature_importances)
```

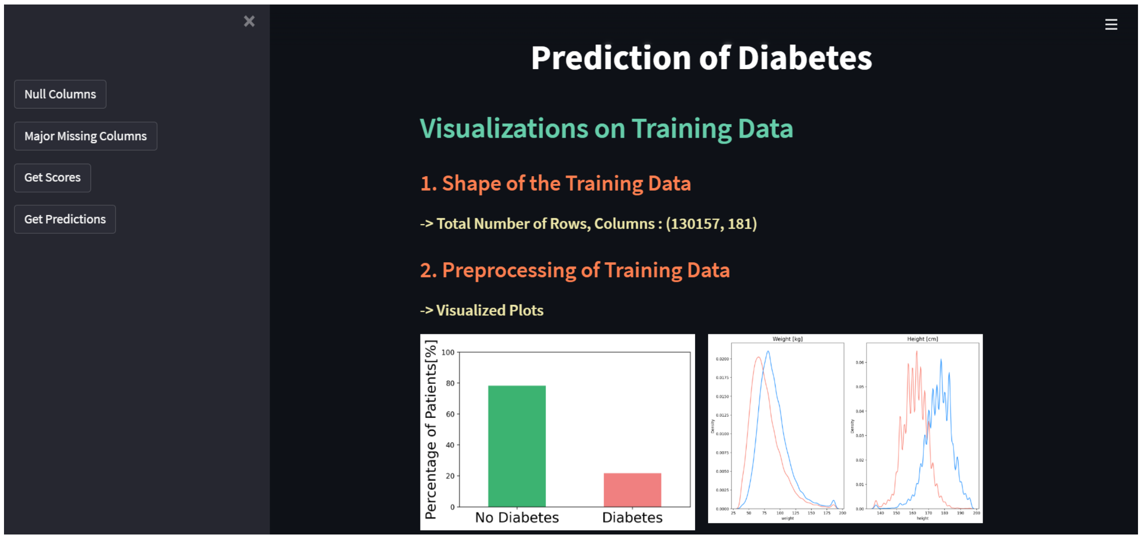
```
                  feature  importance
0                     age         161
1                     bmi         196
2         elective_surgery           0
3                  height          26
4                  icu_id         332
..                    ...         ...
96     icu_type_Cardiac ICU           2
97           icu_type_MICU           1
98      icu_type_Med-Surg ICU           4
99        icu_type_Neuro ICU           4
100          icu_type_SICU           0

[101 rows x 2 columns]
```

Figure 5.45: Important Features by LightGBM



```
[ ]  feature_importance_values = xgb.feature_importances_
     feature_importances = pd.DataFrame({'feature': features, 'importance': feature_importance_values})
     feature_importances_sorted = plot_feature_importances(feature_importances)
```

Figure 5.46: Important Features by XGB



Prediction of Diabetes

Visualizations on Training Data

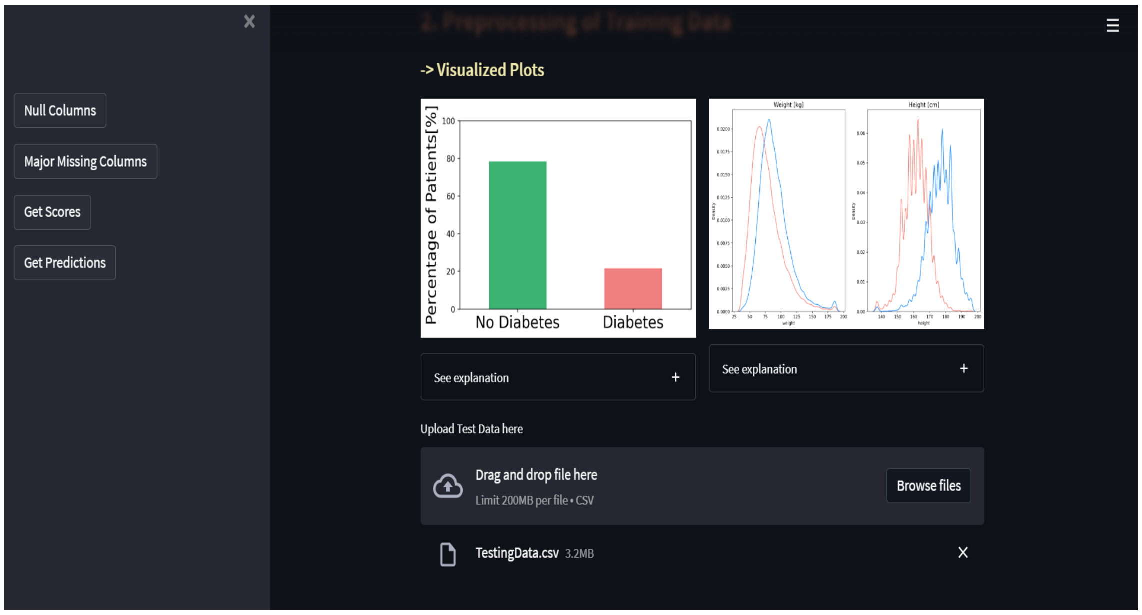
1. Shape of the Training Data

-> Total Number of Rows, Columns : (130157, 181)

2. Preprocessing of Training Data

-> Visualized Plots

Figure 5.47: Dashboard

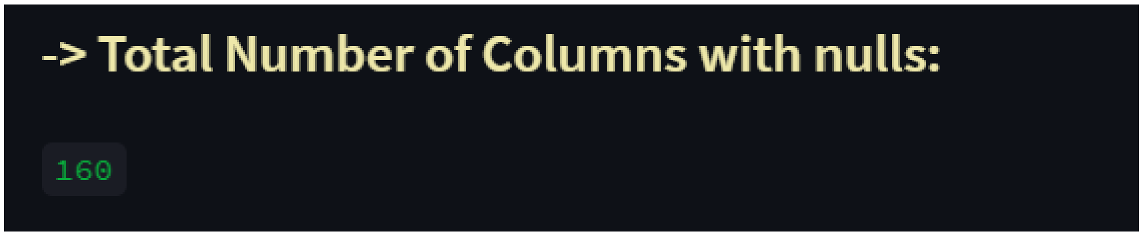


-> Visualized Plots

Upload Test Data here

TestingData.csv 3.2MB

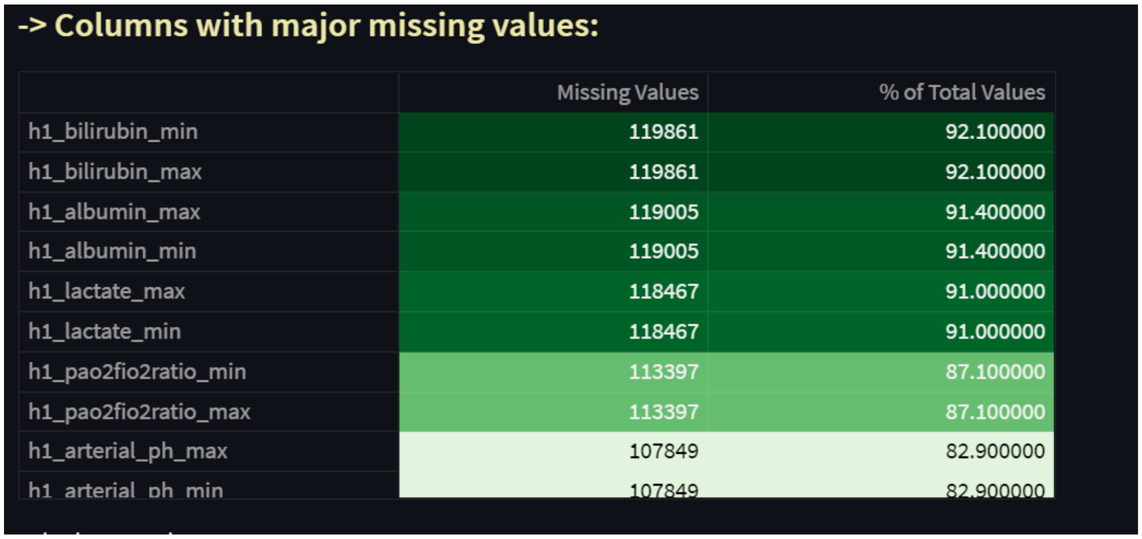
Figure 5.48: Uploading file



-> Total Number of Columns with nulls:

```
160
```

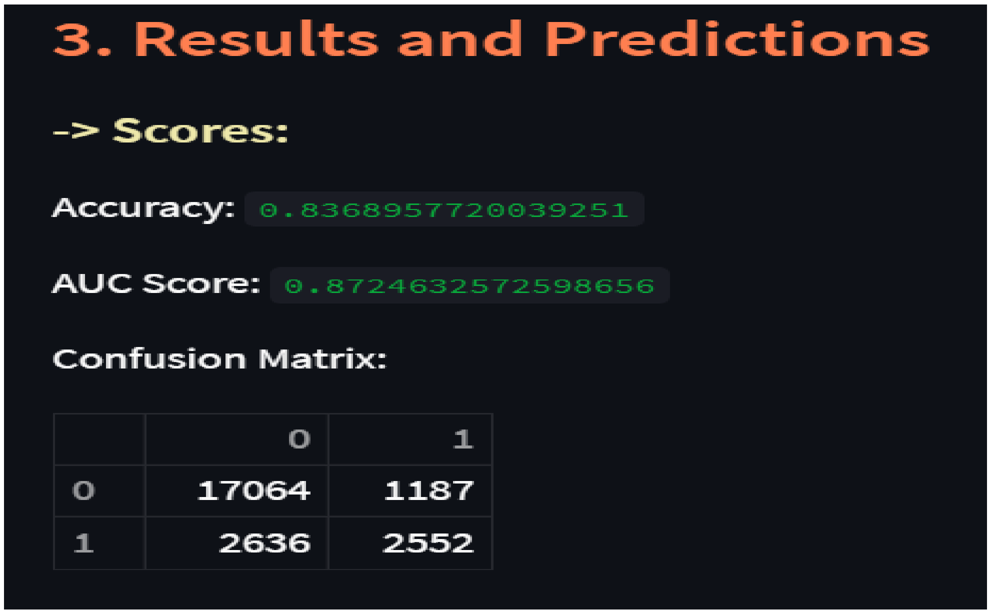
Figure 5.49: Displaying nulls



-> Columns with major missing values:

	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

Figure 5.50: Displaying missing value percentages



3. Results and Predictions

-> Scores:

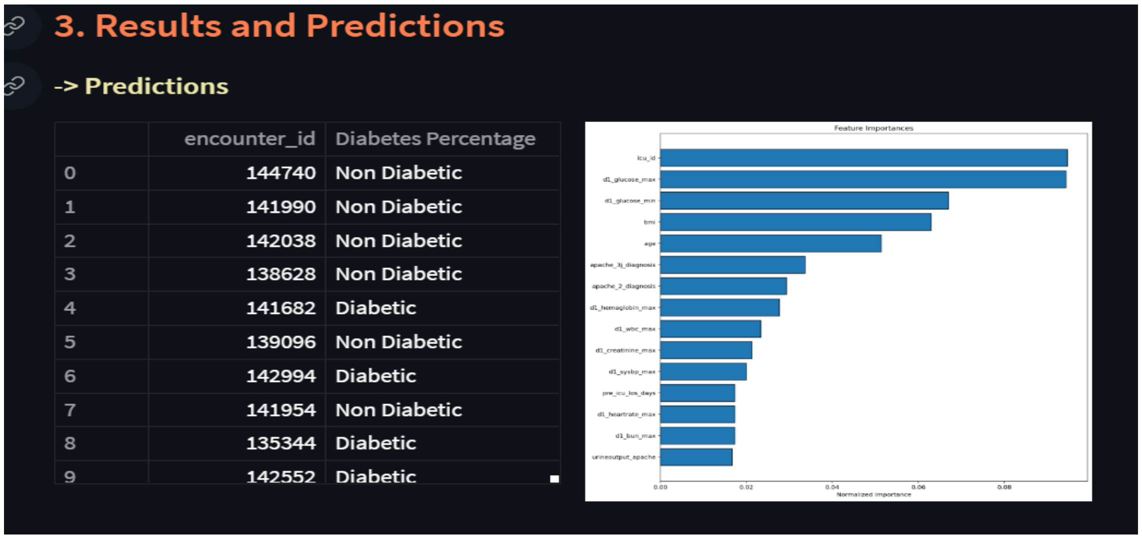
Accuracy: `0.8368957720039251`

AUC Score: `0.8724632572598656`

Confusion Matrix:

	0	1
0	17064	1187
1	2636	2552

Figure 5.51: Scores and Results



3. Results and Predictions

-> Predictions

	encounter_id	Diabetes Percentage
0	144740	Non Diabetic
1	141990	Non Diabetic
2	142038	Non Diabetic
3	138628	Non Diabetic
4	141682	Diabetic
5	139096	Non Diabetic
6	142994	Diabetic
7	141954	Non Diabetic
8	135344	Diabetic
9	142552	Diabetic

Figure 5.52: Predictions

6. Conclusion

The main aim of this project was to design and implement Diabetes Prediction Using Machine Learning Methods and Performance Analysis of that methods and it has been achieved successfully. The Experimental results can assist health care to take early prediction and make early decision to cure diabetes and save humans life. As of now the risk of diabetes was predicted using different algorithms like XGBoost and Light GBM. LightGBM classifier performs better on integer encoded categorical values.Some techniques like resampling, extraction of indicators using the blood pressure columns into new features, missing value handling of BMI and Blood. pressure columns, 5-fold cross validation are also the major reasons behind the better accuracy. A multicenter study with more variables can make a large difference in efficiency of results.implementation of feature engineering with use of domain knowledge extract features from raw data. The motivation is to use these extra features to improve the quality of results. Hyper Parameter Optimization helps in choosing hyper parameters which are used in controlling the learning process. A combination of both feature engineering and Hyper Parameter Optimization can result in best accurate results. Streamlit is used to build a interactive Data Science web application with all visualization part and predictions.

References

1. N. Sneha Tarun Gangil , “Analysis of diabetes mellitus for early prediction using optimal features selection” Journal of Big Data, 2019.

2. Quan Zou1, Kaiyang Qu, Yamei Luo, Dehui Yin, “Predicting Diabetes Mellitus With Machine Learning Techniques”Front. Genet, 2018.

3. DeeptiSisodia, aDilip SinghSisodia “Prediction of Diabetes using Classification Algorithms” Procedia Computer Science Volume 132 2018.

4. Diabetes prediction model based on an enhanced deep neural network, Huaping Zhou, Raushan Myrzashova Rui Zheng, EURASIP Journal on Wireless Communications and Networking volume 2020, Article number: 148 ,2020

5. Machine Learning Based Diabetes Classification and Prediction for Healthcare Applications, Umair Muneer Butt , Sukumar Letchmunan ,Mubashir Ali ,Fadratul Hafinaz Hassan, Volume 2021,

6. Research on Diabetes Prediction Method Based on Machine Learning, Jingyu Xue,a, Fanchao Min,b, Fengying, Journal of Physics: Conference Series, 2020

7. LGBM Classifier based Technique for Predicting Type-2 Diabetes, B. Shamreen Ahamed Dr. Meenakshi Sumeet Arya, European Journal of Molecular Clinical Medicine, Volume 08, Issue 03, 2021.

8. Prediction of Gestational Diabetes Based on LightGBM, Fan Hou, ZhiXhang Cheng, 2020