8 errors15 warnings

Department of Computer Science

Mortality Risk Prediction of COVID-19 and CICU Patients

CS842 - Introduction to Data Science

Guided By:

Dr. Alireza Manashty

Prepared By:

Srushti Vaghani - 200444489 - srushtivaghani@uregina.ca

Neel Patel - 200451256 - neelpatel@uregina.ca

Harsh Patel - 200445899 - hpp711@uregina.ca

Role of Team Members

Srushti Vaghani
Email: srushtivaghani@uregina.ca
Expertise: Extensive knowledge of basic Python programming and Data pre-processing.
Role: Data Cleaning, Conduction of Project Analysis, Visualization UI Design and Project Report.

Harsh Patel
Email: hpp711@uregina.ca
Expertise: Intermediate knowledge in Python programming and feature engineering
Role: Data Exploration and Cleaning, Feature Engineering, Project Report.

Neel Patel
Email: neelpatel@uregina.ca
Expertise: basic knowledge of Django Framework and intermediate knowledge in python programming.
Role: Data visualization, Data Cleaning and Normalization, Feature Selection, Project Report, Project Hosting.

Acknowledgments

Working on this project has been a great opportunity for us. We gained experience and acquired quite a bit of knowledge of designing and analyzing a data science project. We would like to express our deepest gratitude to our course project supervisor, Dr. Alireza Manashty, for his patience and guidance throughout the semester. He imparted valuable information, motivated us, and gave advice whenever required. We would also like to thank everyone surrounding us for the continuous support and for creating a good environment for us to work on this project.

Abstract

The most trending crisis of 2020 has been the coronavirus pandemic. The virus is infecting people ceaselessly. Even after vaccination new variant of COVID-19 has started affecting the world. The rising cases of COVID-19 are resulting in increasing demand for hospital care. When the pandemic was at peak in some countries, hospitals in those countries were having tough times managing the patients. Many seriously ill patients were left unattended and therefore died. Hence, it becomes very important to take quick decisions on how to allocate the available resources efficiently. Likewise, There have been many deaths due to various organ failure in the patients who have been treated in the CICU (Cardiac Intensive Unit Care). The dysfunction of patients' organ is usually within six human body systems : Repository, Cardiovascular, Hepatic, Coagulation, Renal and Neurological systems. Predict the survival status of ICU (Intensive Care Unit) patients at the end of their hospital stay is important for various clinical and organizational tasks. This project will help in solving this problem by proposing and implementing a solution. We have used classification technique to predict a patient's likely condition to be and Regression technique to count number of days to die which depends on his/her pre-existing conditions. The algorithms that have been used in this project are Logistic Regression, Decision Tree, Random Forest, Neural Network, Linear Regression, Support Vector Machine and Gradient Boosting.

Table of Contents

List of Figures

1 Introduction

The ongoing COVID-19 pandemic has shaken up the world. It has affected the lives of people in various ways. The smooth functioning of the world has been disrupted. Companies and individuals are suffering financial losses. This pandemic has also resulted in many deaths. Because of the easy transmissibility of the virus, it is difficult to control its spread efficiently. Even after finding many COVID-19 vaccines still people are getting affected as COVID is not eradicated. Moreover, due to new variant of COVID-19, the the current covid patient ratio is more than 136 million and death rate is more than 2 million in all over the world. Hence, the number of cases are increasing daily and exponentially.Since, every patient cannot be treated in a hospital due to less capacity of hospitals to accommodate them also the high number of cases.Patients are admitted only if the infection is serious or if the patient has co-morbidities. Prioritizing patients is very important as it will ensure that the ones who need the utmost attention are taken care of. As a result, it will bring down the death rate and more lives will be saved.

Likewise, there have been many deaths due to various organ failures in the CICU(Cardiac Intensive Care Unit) patient. To prevent death from organ dysfunction we can use Sequential Organ Failure Assessment score (SOFA). Basically, SOFA is the simple and objective mortality prediction score that is used to track a person's status during the stay in an intensive care unit (ICU) to determine the extent of a person's organ function or rate of organ failure. The SOFA score is depended on six different scores, the six scores are respiratory, cardiovascular, hepatic, coagulation, renal and neurological systems. Prediction of the survival status of ICU patients is very useful for various clinical tasks. Therefore, using patient's SOFA score we can predict his/her condition and give medications accordingly that will lead to decrease the fatality rate for patients with multiple organ dysfunction.

This report talks about a data science project in which the mortality risk of COVID-19 and CICU patients is predicted based on their pre-existing health conditions using a classification and Regression model. The details of the datasets that has been chosen for the project have been discussed thoroughly. Basic analytics of the data has also been shown. The report has a full explanation of the pre-processing and model development stages. The model with the best performance has been selected and a user interface has been developed for getting the input values of the features and displaying the predicted outcome of a patient's condition. The programs written for implementing the project have been written in Python using Google Colab notebook and the web application has been made using flask framework. The report also includes discussion of relevant literature and concludes the findings of the project.

2 Problem Statement

2.1 COVID-19 PATIENT PRE-CONDITION DATASET

The COVID-19 pandemic has affected many people and caused millions of deaths worldwide. There being some vaccines available now, but still not each and every patient are getting it due to less availability of the vaccine. Therefore, patients have to be treated with upmost care using different combination of the available medicines. Moreover, the new variant of corona virus spreads across the society more easily and very rapidly, which has more chances of people affected of covid-19. In addition, covid-19 has affected more than 136 million people in the world. Also, the mortality rate is increasing day by day and more than 2 million people died globally. As the disease is very contagious, there are very high number of the cases compared to healthcare personnel and various equipment, and hence it is a great challenge for the healthcare system to attend to each and every patient, which further results into high mortality rate.

2.2 PREDICTIVE VALUE OF INDIVIDUAL SEQUENTIAL ORGAN FAILURE ASSESSMENT SUB SCORES FOR MORTALITY IN THE CARDIAC INTENSIVE CARE UNIT

On the other hand, there is a noticeable death rate of organ failure patients who are being treated in ICU. However, some patients can even survive multiple organ failures if appropriate medical care can be given on right time. Mortality is dependent on various factors like severity of organ failure, how many organ failures are there, age, various readings of other organ's functionality, and many more. Considering these many complex factors, it is highly challenging task to determine severity of patient's health admitted in ICU, and hence sometimes appropriate medical care could not be given to a more critical patient, which further results into the death of the patient. Sequential Organ Failure Assessment score (SOFA) is used to determine the severity of organ dysfunction and to prevent death from organ failure. The SOFA score is based on six different scores, such as the respiratory, cardiovascular, liver, coagulation, renal and neurological systems.

3 Solution Overview

There has to be a way to classify the patients based on the risk factor so that more attention is given to the patients which are at high risk. Determining the patients who are more likely to die and treating them on time might help to bring down the death rate. This problem can be solved by using classification methods having high accuracy rate which can classify the patients into two categories, i.e., whether patient could die or not. Also, if patient is at the high risk of death, then using regression methods we can predict the number of days to die for a patient, so that proper medical attention could be given. Predicting mortality among patients with COVID-19 or patients with organs dysfunction admitted in an ICU, present with a spectrum of complications is very difficult, hindering the prognostication and management of the disease. The aim of this project is to develop accurate prediction models to predict both disease mortality rate using unbiased computational methods, and identify the clinical features most predictive of this outcome. For that, we have selected a research paper which uses RFE model(Recursive Feature Elimination) for feature selection and train model using those feature only. We will try to solve our mortality prediction problem for both the problems. According to the paper, we can predict patient's death using his/her selected features by RFE model. Apart from this, we can also predict number of days in which patient may die.

After trying these models on our datasets, we will also apply some baseline models such as decision tree, logistic regression and linear regression on our datasets and will compare the accuracy of these models with the models in selected three papers. The best model for both classification and regression will selected for each dataset and will be used for final prediction problem for our project. The selected classification model will predict whether patient will die or not, and if patient is likely to die then, the selected regression model will be used to count the number of days in which the patient may die. The results of these models will help doctors to give medications and other treatments according to patients' condition which will lead to fast recovery and also reduce the overall mortality rate of both disease patients.

Here, we can see the high level diagram of our project.

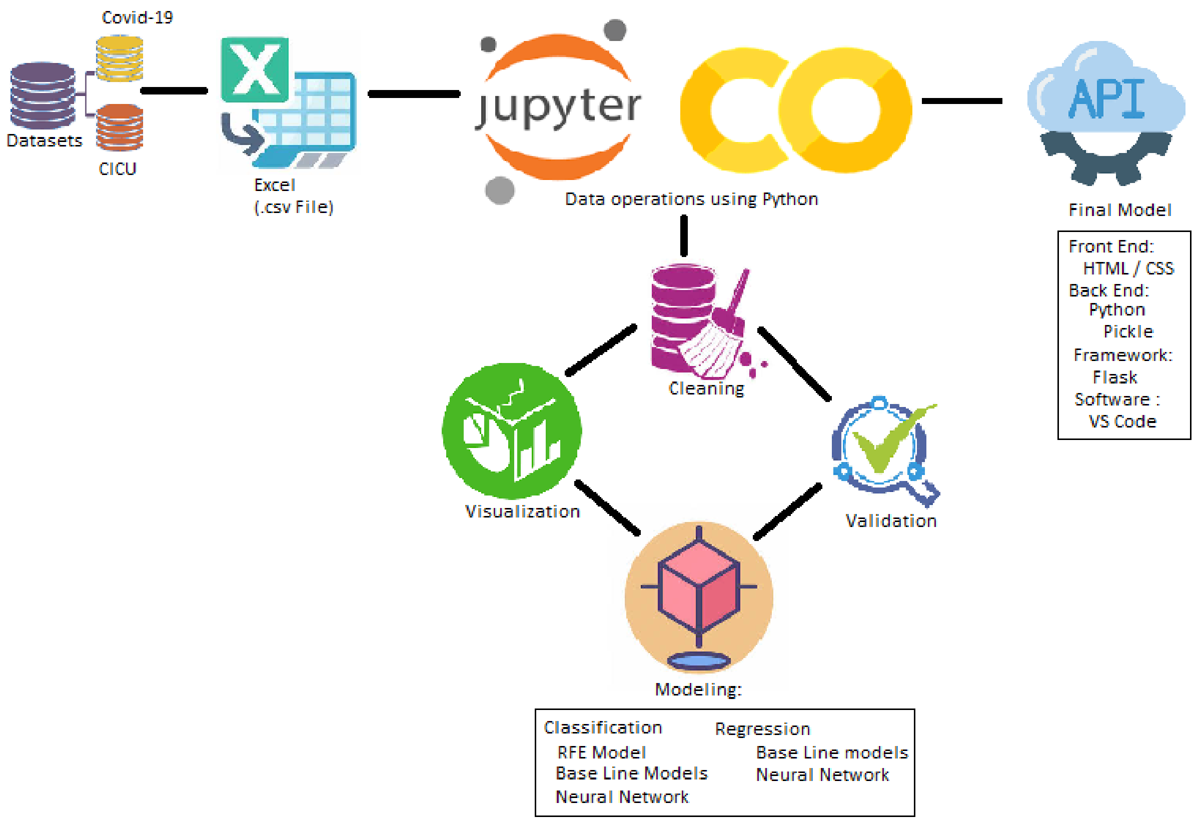


Figure 1: High Level diagram of Project

4 Tools and Technologies

Figure 2: Tools and Technologies

4.1 Excel

Microsoft Excel is the most suitable tool when it comes to working with data. Therefore, it is popular for data science. All small and big companies are using it in their routine operations as it is data analysis tool. We use Excel to access our .csv format datasets.

4.2 Jupyter Notebook

Jupyter is an open-source tool and it is developed based on IPython. It helps to develop open-source software. It is a web-application used for writing code. Jupyter Notebooks is a tool by which we will perform data cleaning, visualization and create machine learning models.

4.3 Google Colab

Google Colab is an open-source tool and allows to write and run python code on web browser. The google colab is based on the jupyter. Anyone can run and load the python files stored on the google drive and github directly in the google colab.

4.4 Python

Python is a popular programming language which is an interpreted, high-level and for general purpose.

4.5 Visual Studio Code

Visual Studio Code is an open source code editor which is available for windows, macOS and linux.

4.6 Flask

Flask is a micro-framework which is based on the python programming language. As it is a micro-framework, it does not require any tools or libraries. It is used for implementing a web applications.

4.7 HTML and CSS

HTML stands for Hypertext Markup Language and CSS stands for Cascade Styling Sheets. Both are used for the front-end development. HTML used for designing the pages or creating the content of the web pages and CSS is used to apply styles and visual effects to the pages.

5 Methodology and Approach

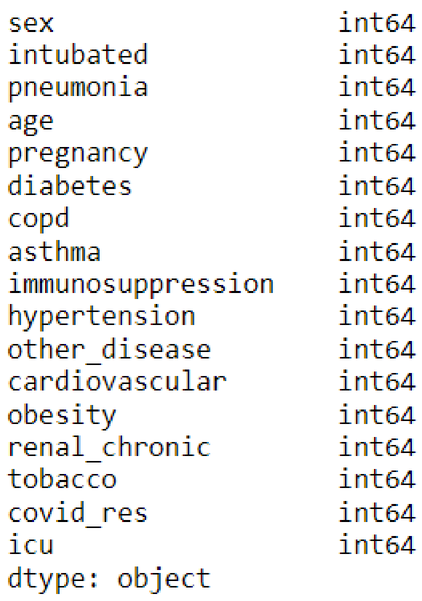
The general approach to the problem includes cleaning and preparing the dataset for applying the models on it. The prediction of COVID-19 and CICU patient's risk of fatality is a classification problem and in how many days patient can die is regression problem. We will apply selected models on clean datasets to create an API which can predict the best results for patients.

5.1 Data Collection

5.1.1 Dataset1: COVID-19 pre-condition dataset

We used a dataset1 that has been made available for the general public by the Mexican Government (updated eight months ago). It was retrieved from Kaggle. This data includes information about every person who has been tested for COVID-19 in Mexico. It includes demographic information such as, age, sex, information on pre-existing conditions, including whether the patient has: diabetes, chronic obstructive pulmonary disease (COPD), asthma, immunosuppression, hypertension, cardiovascular, obesity, pregnancy, chronic renal failure, other prior diseases, and whether he/she uses tobacco. In addition, the data reports the dates on which the patient first showed symptoms, the date when the patient arrived to a care unit, and the date when the patient died (if applicable). It also includes person's id and the type of patient, whether he/she is an inpatient or an outpatient. Apart from these, it includes details of whether a patient came in contact with other COVID-19 patient or not. Furthermore, it contains fields showing the result of the COVID-19 test, whether the patient has developed pneumonia or not, if he/she is intubated, and if she/he is/was being treated in an ICU.

The dataset1 contains 17 features (figure3) and 5,66,602 records, in which, all the attribute values are numeric. The 'covid_res' column shows the test result of people containing 1, 2, and 3 values which represent positive, negative and awaiting result respectively. All the attributes related to pre-existing conditions have 1, 2, 97, 98 or 99 values, where 1 stands for yes, 2 stands for no and 97, 98, 99 values stand for not available. The attribute Id has unique patient id (e.g., '16169f'); sex (1 – female, 2 - male); patient type (1 – outpatient, 2 – inpatient); all dates are in 'DD-MM-YYYY' format.



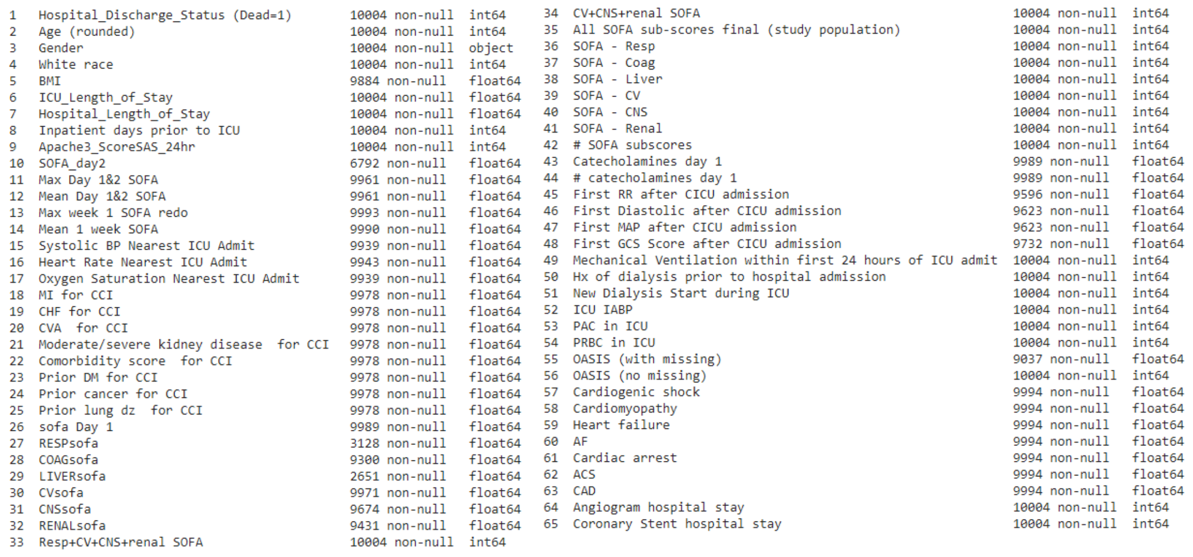
```
sex                    int64
intubated              int64
pneumonia              int64
age                    int64
pregnancy              int64
diabetes               int64
copd                   int64
asthma                 int64
immunosuppression      int64
hypertension           int64
other_disease          int64
cardiovascular         int64
obesity                int64
renal_chronic          int64
tobacco                int64
covid_res              int64
icu                    int64
dtype: object
```

Figure 3: Dataset1: Attribute List

5.1.2 Dataset2:CICU(Cardiac Intensive Care Unit) patient Dataset

We used a dataset2 that has been made available for the general public by Dr. JAcob C. El al. It was retrieved from figshare . This data includes information about every person who has been treated in CICU(Cardiac Intensive Care Unit). It includes patients' demographic information such as, age, sex, information on pre-existing conditions, including whether the patient regular health updates such as: SOFA Score, all six human body system score, Oxygen status and many more information as (Figure4). Also it includes patients length of stay in hospital and ICU.

The dataset2 contains 66 features(figure4) and 10,004 records, in which values are numeric.



```
1   Hospital_Discharge_Status (Dead=1)          10004 non-null  int64
2   Age (rounded)                               10004 non-null  int64
3   Gender                                      10004 non-null  object
4   White race                                  10004 non-null  int64
5   BMI                                         9884 non-null   float64
6   ICU_Length_of_Stay                          10004 non-null  float64
7   Hospital_Length_of_Stay                     10004 non-null  float64
8   Inpatient days prior to ICU                 10004 non-null  int64
9   Apache3_ScoreSAS_24hr                       10004 non-null  int64
10  SOFA_day2                                   6792 non-null   float64
11  Max Day 1&2 SOFA                            9961 non-null   float64
12  Mean Day 1&2 SOFA                           9961 non-null   float64
13  Max week 1 SOFA redo                        9993 non-null   float64
14  Mean 1 week SOFA                            9990 non-null   float64
15  Systolic BP Nearest ICU Admit               9939 non-null   float64
16  Heart Rate Nearest ICU Admit                9943 non-null   float64
17  Oxygen Saturation Nearest ICU Admit         9939 non-null   float64
18  MI for CCI                                  9978 non-null   float64
19  CHF for CCI                                 9978 non-null   float64
20  CVA  for CCI                                9978 non-null   float64
21  Moderate/severe kidney disease  for CCI     9978 non-null   float64
22  Comorbidity score  for CCI                  9978 non-null   float64
23  Prior DM for CCI                            9978 non-null   float64
24  Prior cancer for CCI                        9978 non-null   float64
25  Prior lung dz  for CCI                      9978 non-null   float64
26  sofa Day 1                                  9989 non-null   float64
27  RESPsofa                                    3128 non-null   float64
28  COAGsofa                                    9300 non-null   float64
29  LIVERsofa                                   2651 non-null   float64
30  CVsofa                                      9971 non-null   float64
31  CNSsofa                                     9674 non-null   float64
32  RENALsofa                                   9431 non-null   float64
33  Resp+CV+CNS+renal SOFA                      10004 non-null  int64
```

```
34  CV+CNS+renal SOFA                                         10004 non-null  int64
35  All SOFA sub-scores final (study population)              10004 non-null  int64
36  SOFA - Resp                                               10004 non-null  int64
37  SOFA - Coag                                               10004 non-null  int64
38  SOFA - Liver                                              10004 non-null  int64
39  SOFA - CV                                                 10004 non-null  int64
40  SOFA - CNS                                                10004 non-null  int64
41  SOFA - Renal                                              10004 non-null  int64
42  # SOFA subscores                                          10004 non-null  int64
43  Catecholamines day 1                                      9989 non-null   float64
44  # catecholamines day 1                                    9989 non-null   float64
45  First RR after CICU admission                             9596 non-null   float64
46  First Diastolic after CICU admission                      9623 non-null   float64
47  First MAP after CICU admission                            9623 non-null   float64
48  First GCS Score after CICU admission                      9732 non-null   float64
49  Mechanical Ventilation within first 24 hours of ICU admit 10004 non-null  int64
50  Hx of dialysis prior to hospital admission                10004 non-null  int64
51  New Dialysis Start during ICU                             10004 non-null  int64
52  ICU IABP                                                  10004 non-null  int64
53  PAC in ICU                                                10004 non-null  int64
54  PRBC in ICU                                               10004 non-null  int64
55  OASIS (with missing)                                      9037 non-null   float64
56  OASIS (no missing)                                        10004 non-null  int64
57  Cardiogenic shock                                         9994 non-null   float64
58  Cardiomyopathy                                            9994 non-null   float64
59  Heart failure                                             9994 non-null   float64
60  AF                                                        9994 non-null   float64
61  Cardiac arrest                                            9994 non-null   float64
62  ACS                                                       9994 non-null   float64
63  CAD                                                       9994 non-null   float64
64  Angiogram hospital stay                                   10004 non-null  int64
65  Coronary Stent hospital stay                              10004 non-null  int64
```

Figure 4: Dataset2: Attribute List

5.2 Data Cleaning

Data cleaning is a crucial step in data mining. The dataset being used for mining purposes might not be perfect. It might contain erroneous data, unimportant data or records with missing values. The dataset needs to be transformed in such a manner that it gives the best and accurate results. The dataset that has been chosen here needed some preprocessing, which was conducted in the steps mentioned in this section.

5.2.1 Dataset 1

5.2.1.1 Converted Values of Some attributes The dataset is in binary format. For better understanding we have converted the value of sex attribute to 0 and 1 for Female and Male, respectively.

```
#converting to female : 0, male : 1
data['sex'] = data['sex'].apply(lambda x : x-1)
data.sex.value_counts().plot(kind='bar')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f1a80a3c150>
```

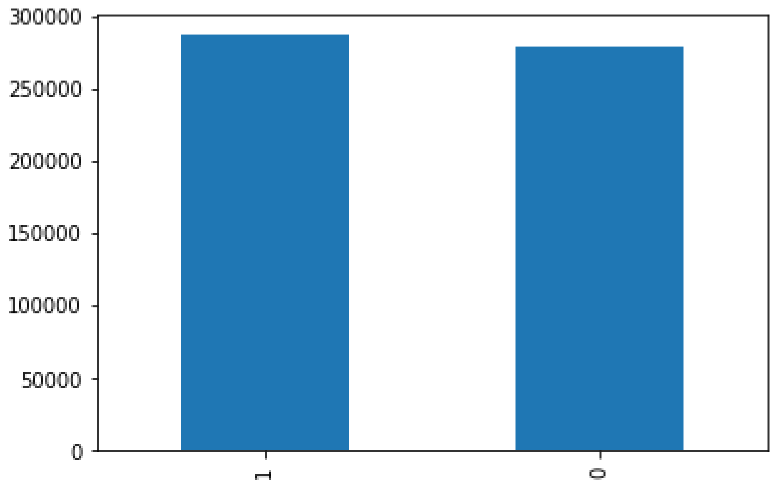


Figure 5: Dataset1: Sex attribute value Conversion

5.2.1.2 Created Some New Attributes To train our model we need to be more clear about attributes. Therefore, we have created some new features by using existed ones. For Example, Created new attribute called 'Hospitalized' if value of 'Patient_type' is 1 which means if patient is impatient.

```
# converting to Outpatient : 0, Inpatient : 1 and changing name to hospitalized
data['hospitalized'] = data['patient_type'].apply(lambda x : x-1)
data.hospitalized.value_counts().plot(kind='bar')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f1a804c4c50>
```

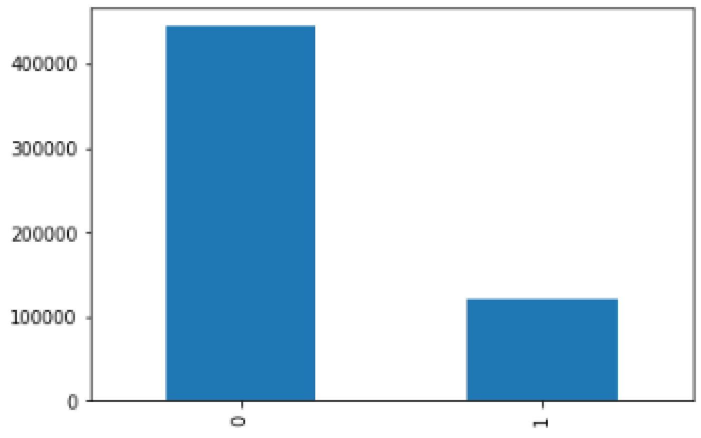
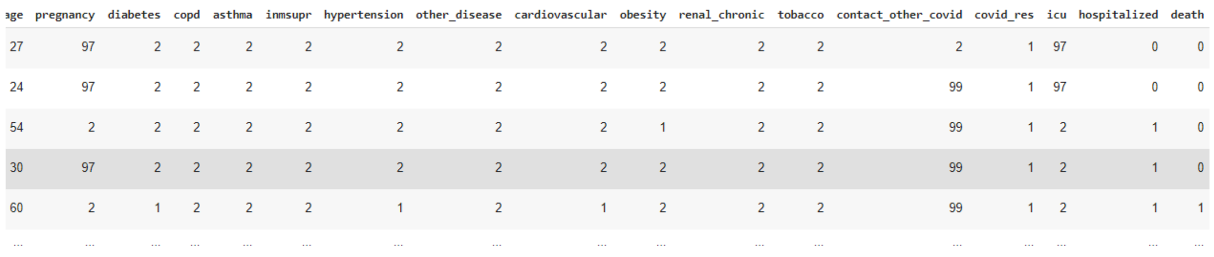


Figure 6: Dataset1: New 'Hospitalized' attribute

Likewise, 'death' class attribute has been created, if 'date_died' attribute is available for patient which means if date of patient's death is available then death status will be 1 for that patient.

```python
# converting to Outpatient : 0, Inpatient : 1 and changing name to hospitalized
data['hospitalized'] = data['patient_type'].apply(lambda x : x-1)
data.hospitalized.value_counts().plot(kind='bar')
```

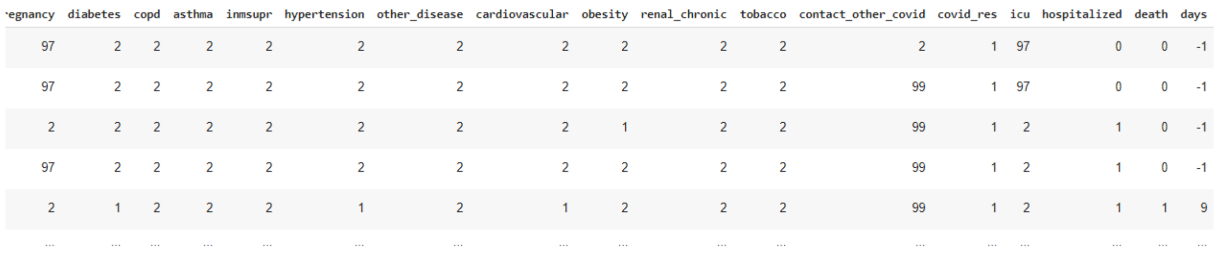


age	pregnancy	diabetes	copd	asthma	inmsupr	hypertension	other_disease	cardiovascular	obesity	renal_chronic	tobacco	contact_other_covid	covid_res	icu	hospitalized	death
27	97	2	2	2	2	2	2	2	2	2	2	2	1	97	0	0
24	97	2	2	2	2	2	2	2	2	2	2	99	1	97	0	0
54	2	2	2	2	2	2	2	2	1	2	2	99	1	2	1	0
30	97	2	2	2	2	2	2	2	2	2	2	99	1	2	1	0
60	2	1	2	2	2	1	2	1	2	2	2	99	1	2	1	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...

Figure 7: Dataset1: New 'Death' Class attribute

Moreover, We have created the 'days' class attribute by differentiating patient's 'entry_date' and 'date_died' which is a clear difference between patient's hospital date of entry and date of death.

```python
# Create new column of number of days after patient died
def calculateDays(start_date_str, end_date_str):
    start_date_obj = datetime.strptime(start_date_str, '%d-%m-%Y')
    end_date_obj = datetime.strptime(end_date_str, '%d-%m-%Y')
    days = end_date_obj - start_date_obj
    return days.days
positive_patients['days'] = positive_patients.apply(lambda x: calculateDays(x.entry_date,x.
    date_died) if x.death == 1 else -1, axis=1)
```



egnancy	diabetes	copd	asthma	inmsupr	hypertension	other_disease	cardiovascular	obesity	renal_chronic	tobacco	contact_other_covid	covid_res	icu	hospitalized	death	days
97	2	2	2	2	2	2	2	2	2	2	2	1	97	0	0	-1
97	2	2	2	2	2	2	2	2	2	2	99	1	97	0	0	-1
2	2	2	2	2	2	2	2	1	2	2	99	1	2	1	0	-1
97	2	2	2	2	2	2	2	2	2	2	99	1	2	1	0	-1
2	1	2	2	2	1	2	1	2	2	2	99	1	2	1	1	9
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...

Figure 8: Dataset1: New 'Days' class attribute

5.2.1.3 Removing records having insignificant data The dataset contained data of a bunch of people who were tested for COVID-19. The feature 'covid_res' had 3 values in which '1' represents people who tested positive, '2' represents people who tested negative and '3' represents people whose results were yet to be received. Since, the model makes predictions about the likely condition of a COVID-19 patient's health, it does not require the records of people who tested negative and whose results were not available. Hence, all those records have been removed.

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f1a80462050>
```

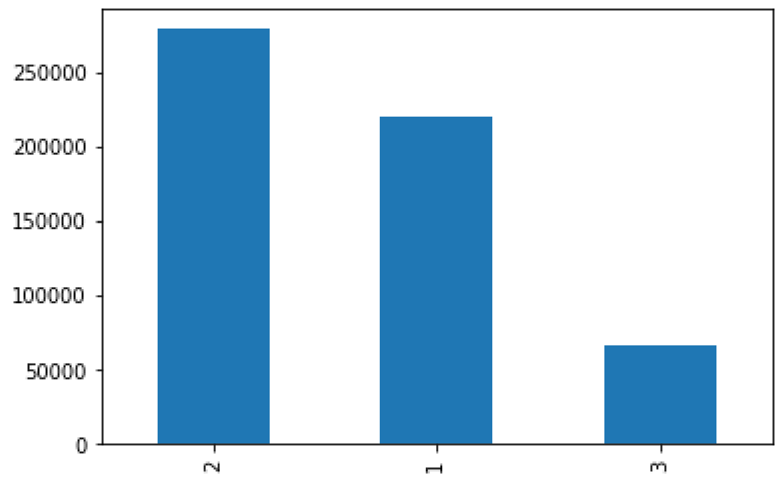
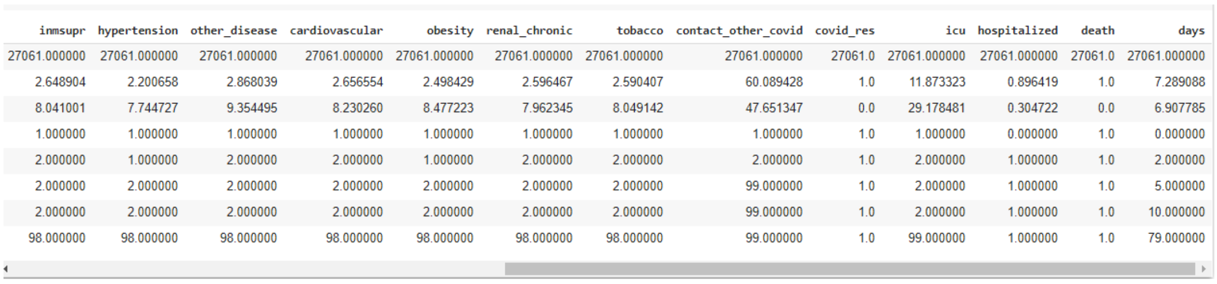


Figure 9: Dataset1: Drop records using 'Covid_res' attribute

```python
#drop patients with negative or pending results
positive_patients = data.drop(data[(data.covid_res == 2) | (data.covid_res == 3) ].index)
```

Likewise there are some negative values appeared for days feature which will affect the model training part. So, We have dropped those records with negative days count.

```python
#drop these 20 records.
positive_patients = positive_patients.drop(positive_patients[positive_patients.days < 0].index)
```

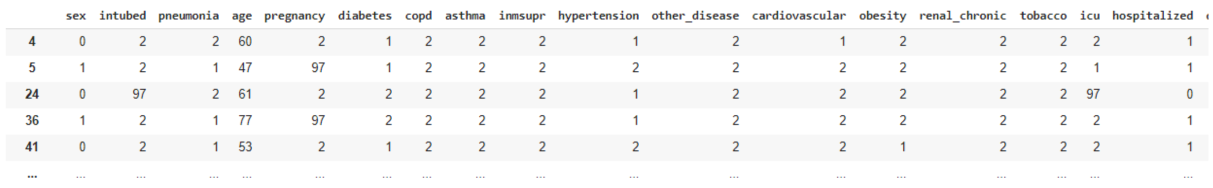


inmsupr	hypertension	other_disease	cardiovascular	obesity	renal_chronic	tobacco	contact_other_covid	covid_res	icu	hospitalized	death	days
27061.000000	27061.000000	27061.000000	27061.000000	27061.000000	27061.000000	27061.000000	27061.000000	27061.0	27061.000000	27061.000000	27061.0	27061.000000
2.648904	2.200658	2.868039	2.656554	2.498429	2.596467	2.590407	60.089428	1.0	11.873323	0.896419	1.0	7.289088
8.041001	7.744727	9.354495	8.230260	8.477223	7.962345	8.049142	47.651347	0.0	29.178481	0.304722	0.0	6.907785
1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.0	1.000000	0.000000	1.0	0.000000
2.000000	1.000000	2.000000	2.000000	1.000000	2.000000	2.000000	2.000000	1.0	2.000000	1.000000	1.0	2.000000
2.000000	2.000000	2.000000	2.000000	2.000000	2.000000	2.000000	99.000000	1.0	2.000000	1.000000	1.0	5.000000
2.000000	2.000000	2.000000	2.000000	2.000000	2.000000	2.000000	99.000000	1.0	2.000000	1.000000	1.0	10.000000
98.000000	98.000000	98.000000	98.000000	98.000000	98.000000	98.000000	99.000000	1.0	99.000000	1.000000	1.0	79.000000

Figure 10: Dataset1: Drop records using 'Days' attribute

5.2.1.4 Removing unnecessary attributes The dataset should not have any features that are not required for mining the data. The lower the dimension of the dataset, better is the model's performance. The features that were needed for predicting the risk were the pre-existing health conditions of the patients. So, the features that were not pertinent were simply dropped. The columns with names: id, entry_date, date_symptoms, patient_type, date_died, contact_other_covid were dropped from the dataset. These columns were removed using the drop method of Data Frame.

```
#dropping unneccary columns
positive_patients = positive_patients.drop(columns=['id','patient_type','entry_date','
    date_symptoms','date_died','contact_other_covid','covid_res'])
positive_patients
```



	sex	intubed	pneumonia	age	pregnancy	diabetes	copd	asthma	inmsupr	hypertension	other_disease	cardiovascular	obesity	renal_chronic	tobacco	icu	hospitalized
4	0	2	2	60	2	1	2	2	2	1	2	1	2	2	2	2	1
5	1	2	1	47	97	1	2	2	2	2	2	2	2	2	2	1	1
24	0	97	2	61	2	2	2	2	2	1	2	2	2	2	2	97	0
36	1	2	1	77	97	2	2	2	2	1	2	2	2	2	2	2	1
41	0	2	1	53	2	1	2	2	2	2	2	2	1	2	2	2	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...

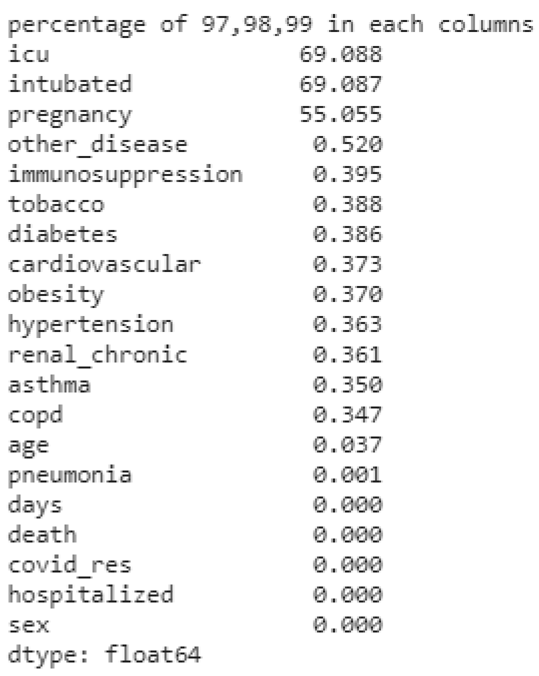
Figure 11: Dataset1: Drop unnecessary attributes

5.2.1.5 Renaming features Some features which were named incorrectly were renamed. Feature named 'intubed' was renamed to 'intubated' and 'inmsupr' was renamed to 'immunosuppression', so that it would be clear what kind of data they represent.

```
#some attributtes' name should be more specific
positive_patients = positive_patients.rename(columns={"intubed":"intubated","inmsupr":"
    immunosuppression"})
```

5.2.1.6 Dealing with duplicate values All the features related to medical conditions had some values like '97', '98', '99' which denote 'not available/not applicable'. Before any actions we checked the percentages of missing values for every column.

```
#missing values percentage
print('percentage of 97,98,99 in each columns')
print(((positive_patients.isin([97, 98, 99]).sum()/positive_patients.shape[0] * 100).round(3)).
    sort_values(ascending = False) )
```

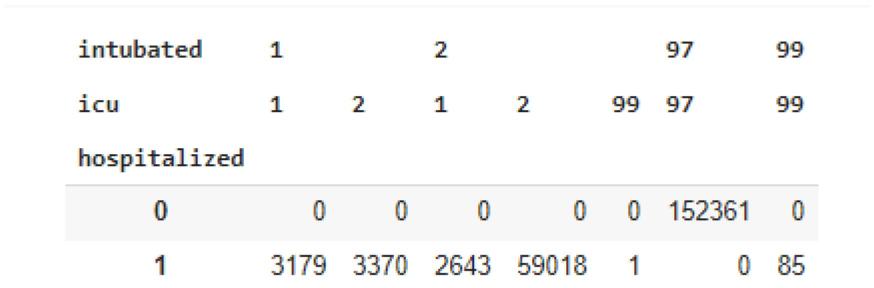


```
percentage of 97,98,99 in each columns
icu                      69.088
intubated                69.087
pregnancy                55.055
other_disease             0.520
immunosuppression         0.395
tobacco                   0.388
diabetes                  0.386
cardiovascular            0.373
obesity                   0.370
hypertension              0.363
renal_chronic             0.361
asthma                    0.350
copd                      0.347
age                       0.037
pneumonia                 0.001
days                      0.000
death                     0.000
covid_res                 0.000
hospitalized              0.000
sex                       0.000
dtype: float64
```

Figure 12: Dataset1: Percentage of missing values

According to results, it is clear that it only available in three features. To more accuracy, we have checked count of each missing value for founded columns.

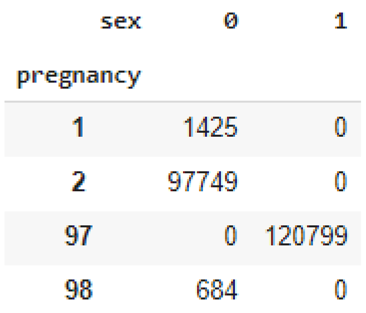
```
pd.crosstab(positive_patients['hospitalized'],[positive_patients['intubated'], positive_patients
    ['icu']])
```



intubated	1		2			97	99
icu	1	2	1	2	99	97	99
hospitalized							
0	0	0	0	0	0	152361	0
1	3179	3370	2643	59018	1	0	85

Figure 13: Dataset1: Missing value counts for 'icu' and 'intubated'

```
pd.crosstab(Dataset1: positive_patients['pregnancy'], positive_patients['sex'])
```



sex	0	1
pregnancy		
1	1425	0
2	97749	0
97	0	120799
98	684	0

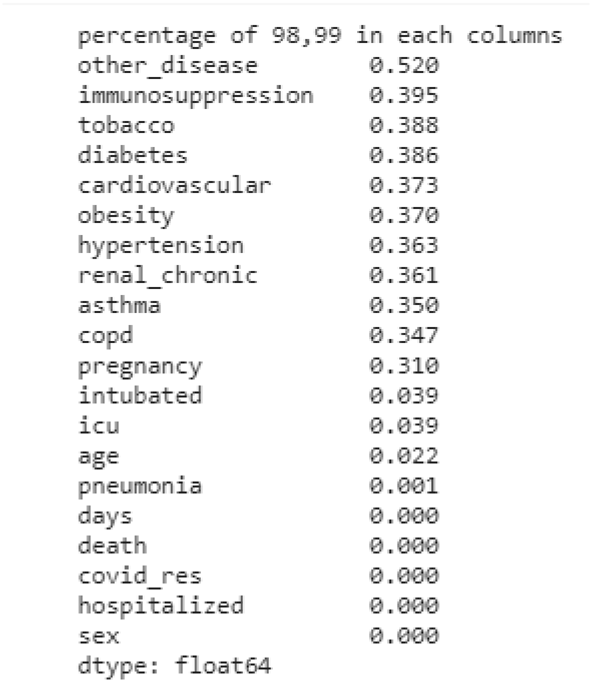
Figure 14: Dataset1: Missing value counts for 'pregnancy'

Finally we found that, the value 97 for 'icu' and 'intubated' columns can be replaced with 0 as their 'hospitalized' value is 0 i.e. if patient has not been hospitalized then how he/she can be in icu or on intubated. Likewise, it is obvious that the male patients' pregnancy status should be 0 not 97. As a result we have converted value 97 for these three columns to 0.

```
#convert 97 to 2 for icu, intubated and pregnancy
for column in ['icu','intubated','pregnancy']:
  positive_patients[column] = positive_patients[column].replace(97,2)
```

For 98 and 99 missing values we have again checked the percentages for every column.

```
#missing values percentage for 98 and 99
print('percentage of 98,99 in each columns')
print(((positive_patients.isin([98, 99]).sum()/positive_patients.shape[0] * 100).round(3)).
     sort_values(ascending = False))
```

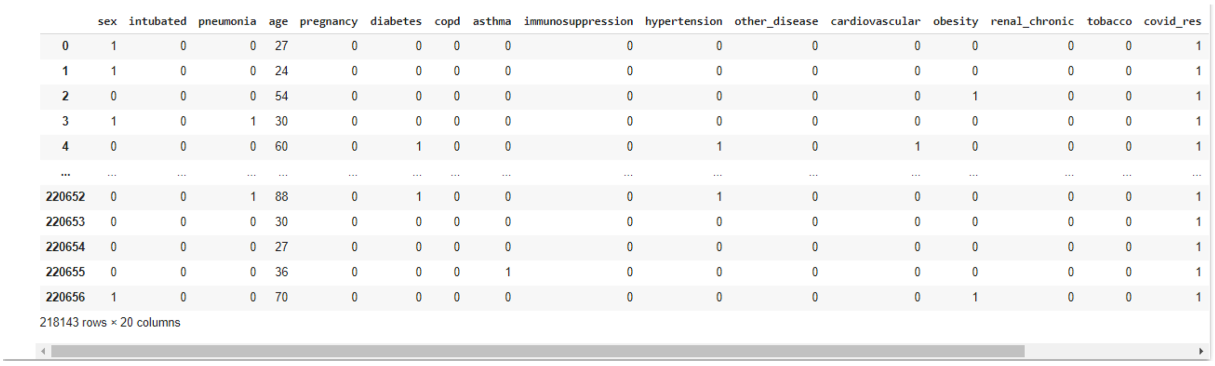


```
percentage of 98,99 in each columns
other_disease          0.520
immunosuppression      0.395
tobacco                0.388
diabetes               0.386
cardiovascular         0.373
obesity                0.370
hypertension           0.363
renal_chronic          0.361
asthma                 0.350
copd                   0.347
pregnancy              0.310
intubated              0.039
icu                    0.039
age                    0.022
pneumonia              0.001
days                   0.000
death                  0.000
covid_res              0.000
hospitalized           0.000
sex                    0.000
dtype: float64
```

Figure 15: Dataset1: Percentage of 98 and 99 values

The percentage ratio is too minor so we have dropped the records having 98 and 99 values and converted value 2 with 0 for all the columns which indicates No.

```python
#dropping rows with values 98,99  and coverting 2 to 0 for all the columns
for column in ['intubated','pneumonia','pregnancy','diabetes','copd','asthma','immunosuppression
    ','hypertension','other_disease','cardiovascular','obesity','renal_chronic','tobacco','
    covid_res','icu']:
    positive_patients[column] = positive_patients[column].replace(2,0)

for column in positive_patients:
  positive_patients.drop(positive_patients[positive_patients[column] == 98].index, inplace=True)
  positive_patients.drop(positive_patients[positive_patients[column] == 99].index, inplace=True)
positive_patients
```



	sex	intubated	pneumonia	age	pregnancy	diabetes	copd	asthma	immunosuppression	hypertension	other_disease	cardiovascular	obesity	renal_chronic	tobacco	covid_res
0	1	0	0	27	0	0	0	0	0	0	0	0	0	0	0	1
1	1	0	0	24	0	0	0	0	0	0	0	0	0	0	0	1
2	0	0	0	54	0	0	0	0	0	0	0	0	1	0	0	1
3	1	0	1	30	0	0	0	0	0	0	0	0	0	0	0	1
4	0	0	0	60	0	1	0	0	0	1	0	1	0	0	0	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
220652	0	0	1	88	0	1	0	0	0	1	0	0	0	0	0	1
220653	0	0	0	30	0	0	0	0	0	0	0	0	0	0	0	1
220654	0	0	0	27	0	0	0	0	0	0	0	0	0	0	0	1
220655	0	0	0	36	0	0	0	1	0	0	0	0	0	0	0	1
220656	1	0	0	70	0	0	0	0	0	0	0	0	1	0	0	1

218143 rows × 20 columns

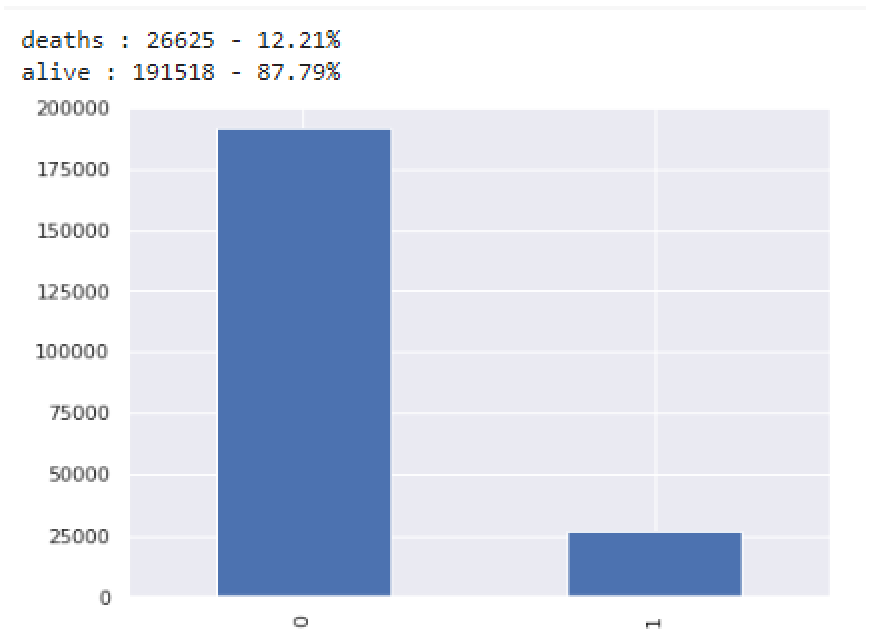
Figure 16: Dataset1: Cleaned dataset

5.2.1.7 Balancing class

The target class of the dataset for classification is 'death'. It has 2 class labels '0' and '1'. Approximately 85% of the records present in the dataset had the class label '0'. This imbalance of classes (figure 17) affects the accuracy of a model.

Without balancing the classes, the accuracy of the model was around 92%. Such high accuracy was a result of correct predictions of class label ‘0’ only, which was present in majority. Hence, the classes needed to be balanced for training the model, so that, both the classes could be predicted correctly.

```
sb.set(rc={'figure.figsize':(7,5)})
positive_patients['death'].value_counts().plot(kind='bar')
deaths = len(positive_patients[positive_patients['death'] == 1])
alive = len(positive_patients[positive_patients['death'] == 0])

print('deaths : {} - {:.2f}% '.format(deaths,deaths*100/len(positive_patients)))
print('alive : {} - {:.2f}% '.format(alive,alive*100/len(positive_patients)))
```



```
deaths : 26625 - 12.21%
alive : 191518 - 87.79%
```

Figure 17: Dataset1: Class distribution before balancing

For balancing the classes, combined Under and Over sampling has been used. The number of records with class label ’1’ were reproduced to match with the number of records of the majority class, that is, class label ‘0’. Additional records were generated by using random sampling method to equalize the number of class labels (figure 18). The y-axis of both the plots denote the number of class labels and the x-axis denotes the class labels themselves.

```
patients_majority = positive_patients[positive_patients.death==0]
patients_minority = positive_patients[positive_patients.death==1]
patients_minority_length = len(patients_minority)
patients_majority_length = len(patients_majority)


patients_minority = resample(patients_minority,
                                 replace=True,     # sample with replacement
                                 n_samples=patients_minority_length * 2,    # to match majority
    class
```

```
                                     random_state=123)

patients_majority = resample(patients_majority,
                                     replace=False,     # sample without replacement
                                     n_samples=patients_minority_length * 2,     # to match majority
     class
                                     random_state=123)

positive_patients_mixedsampled = pd.concat([patients_minority, patients_majority])
positive_patients_mixedsampled.death.value_counts().plot(kind='bar')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f1a79bcbfd0>
```

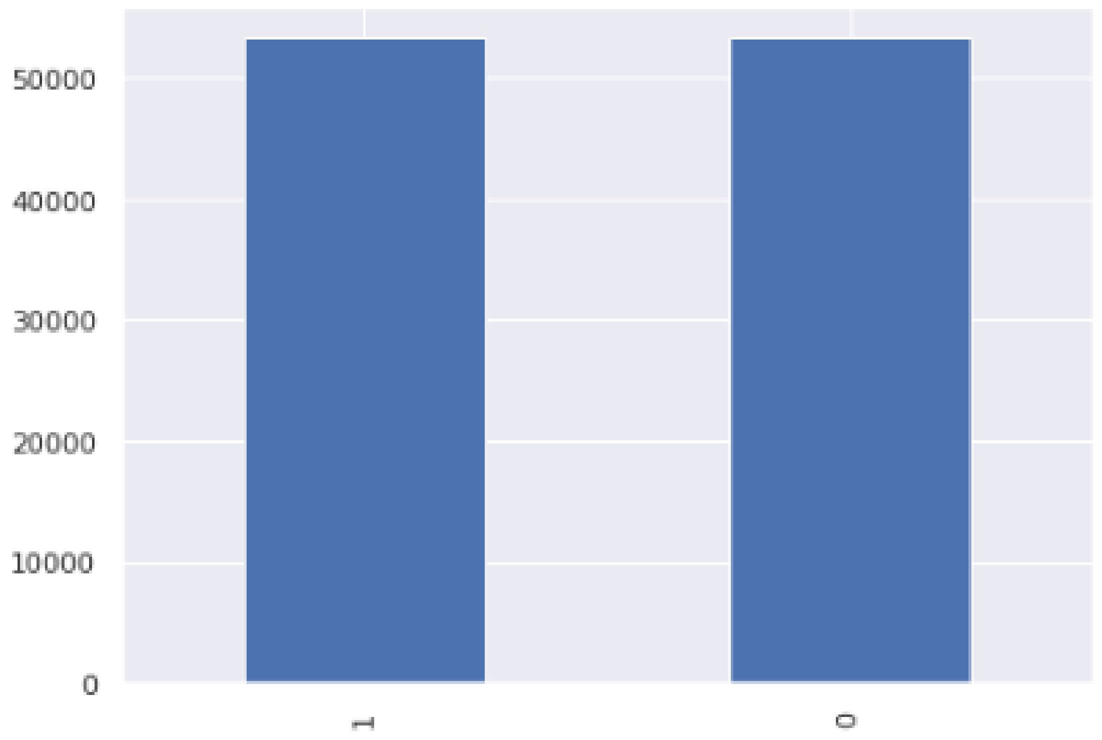


Figure 18: Dataset1: Class distribution after balancing

5.2.2 Dataset 2

5.2.2.1 Remove all the unnamed attributes The dataset had 334 attributes as there were some unnamed columns. Therefore, first we just dropped those columns.

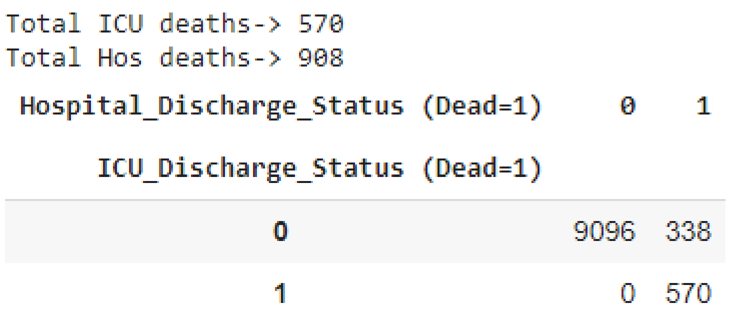
```
# removing all extra unnamed colums
patients = data.loc[:,~data.columns.str.match("Unnamed")]
```

5.2.2.2 Converted Values of Some attributes In this section, we converted the value of 'Gender' attribute to 0 and 1 for Female and Male,respectively.

```
#changing gender to female:0 male:1
patients['Gender'] = patients['Gender'].apply(lambda x: 0 if x=='F' else 1)
```

5.2.2.3 Created Some New Attributes To train our model we need to be more clear about attributes. Therefore, we have created some new features by using existed ones. For Example, Created new class attribute called 'Death' using total ICU and hospital deaths.

```
print('Total ICU deaths-> {}'.format(len(patients[(patients['ICU_Discharge_Status (Dead=1)'] ==
    1)])))
print('Total Hos deaths-> {}'.format(len(patients[(patients['Hospital_Discharge_Status (Dead=1)'
    ] == 1)])))
pd.crosstab(patients['ICU_Discharge_Status (Dead=1)'],patients['Hospital_Discharge_Status (Dead
    =1)'])
```

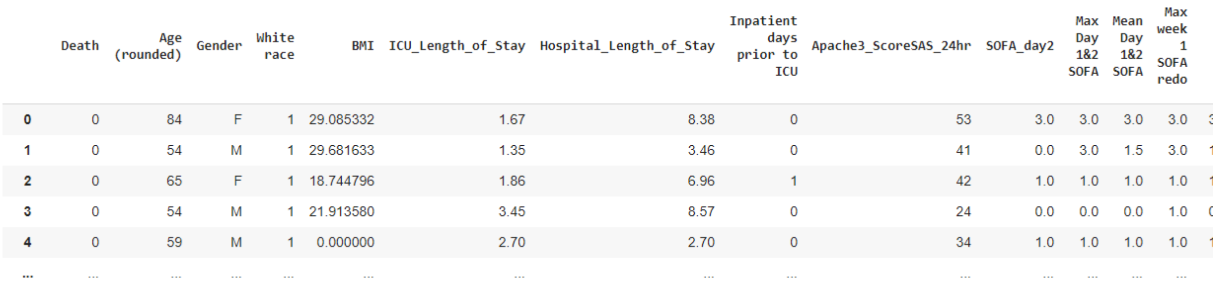


```
Total ICU deaths-> 570
Total Hos deaths-> 908
```

Hospital_Discharge_Status (Dead=1)	0	1
ICU_Discharge_Status (Dead=1)		
0	9096	338
1	0	570

Figure 19: Dataset2: Cross table for Patient death

```
#Dropping ICU discharge status as it is a subset of hospital_discharge
patients = patients.drop(columns=['ICU_Discharge_Status (Dead=1)'])
#renaming 'Hospital_discharge_status' to death
patients = patients.rename(columns={'Hospital_Discharge_Status (Dead=1)' : 'Death'})
```

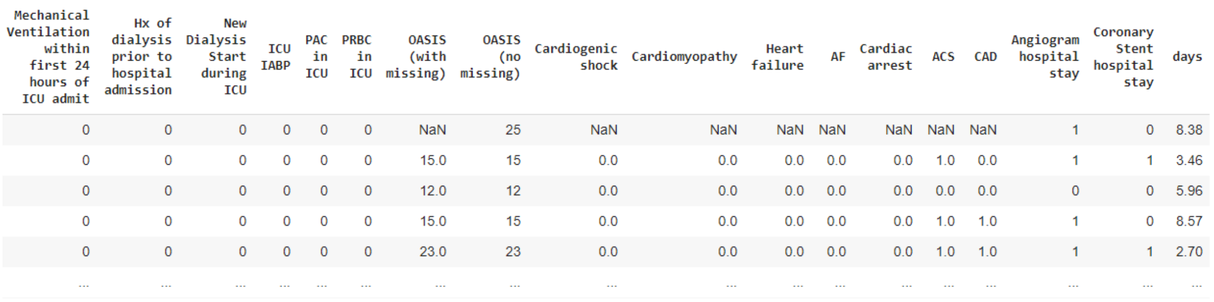


	Death	Age (rounded)	Gender	White race	BMI	ICU_Length_of_Stay	Hospital_Length_of_Stay	Inpatient days prior to ICU	Apache3_ScoreSAS_24hr	SOFA_day2	Max Day 1&2 SOFA	Mean Day 1&2 SOFA	Max week 1 SOFA redo
0	0	84	F	1	29.085332	1.67	8.38	0	53	3.0	3.0	3.0	3.0
1	0	54	M	1	29.681633	1.35	3.46	0	41	0.0	3.0	1.5	3.0
2	0	65	F	1	18.744796	1.86	6.96	1	42	1.0	1.0	1.0	1.0
3	0	54	M	1	21.913580	3.45	8.57	0	24	0.0	0.0	0.0	1.0
4	0	59	M	1	0.000000	2.70	2.70	0	34	1.0	1.0	1.0	1.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...

Figure 20: Dataset2: New 'Death' attribute

Likewise, the 'Days' class attribute has been created by differentiating patient's 'Hospital_Length_of_Stay' and 'Inpatient days prior to ICU'.

```
#calculating number of days the patients was alive
patients['days'] = patients['Hospital_Length_of_Stay'] - patients['Inpatient days prior to ICU']
```



Mechanical Ventilation within first 24 hours of ICU admit	Hx of dialysis prior to hospital admission	New Dialysis Start during ICU	ICU IABP	PAC in ICU	PRBC in ICU	OASIS (with missing)	OASIS (no missing)	Cardiogenic shock	Cardiomyopathy	Heart failure	AF	Cardiac arrest	ACS	CAD	Angiogram hospital stay	Coronary Stent hospital stay	days
0	0	0	0	0	0	NaN	25	NaN	NaN	NaN	NaN	NaN	NaN	NaN	1	0	8.38
0	0	0	0	0	0	15.0	15	0.0	0.0	0.0	0.0	0.0	1.0	0.0	1	1	3.46
0	0	0	0	0	0	12.0	12	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0	0	5.96
0	0	0	0	0	0	15.0	15	0.0	0.0	0.0	0.0	0.0	1.0	1.0	1	0	8.57
0	0	0	0	0	0	23.0	23	0.0	0.0	0.0	0.0	0.0	1.0	1.0	1	1	2.70
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...

Figure 21: Dataset2: New 'Days' attribute

5.2.2.4 Removing unnecessary attributes

The dataset should not require some fields as it contains so many missing values or it does not affects the prediction. Therefore, we have dropped some features from the dataset.

```python
#removing unnecessary columns as number of days are already extracted from these fields
patients = patients.drop(columns=['White race','ICU_Length_of_Stay','Hospital_Length_of_Stay','Inpatient days prior to ICU','SOFA_score'])
#dropping these fields as SOFA score is already extracted
patients = patients.drop(columns=['SOFA_day2','Max Day 1&2 SOFA','Mean Day 1&2 SOFA','Max week 1 SOFA redo','Mean 1 week SOFA'])
#drop unnecessary fields after some calculations
patients = patients.drop(columns=['sofa Day 1','All SOFA sub-scores final (study population)','SOFA - Resp','# SOFA subscores'])
patients
```

5.2.2.5 Removing records having insignificant data

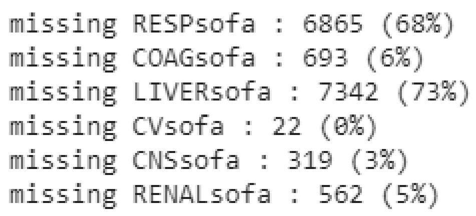
The dataset contains some missing values which can not be replaced. Therefore, we have dropped records with missing values. There are some patients' days count is negative. So, we have dropped those records.

```python
#checking if there any negative days
patients[(patients['days']<0)]
#Dropping records having negative days count
patients=drop.patients[(patients['days']<0)]
```

5.2.2.6 Dealing with duplicate values

All the features related to medical conditions had some missing values. Before any actions we checked the percentages of missing values for every column.

```python
#calculate number of missing values(percentage) in these 6 fields
print('missing RESPsofa : {} ({}%)'.format(patients.RESPsofa.isna().sum(), patients.RESPsofa.isna().sum()*100//len(patients)))
print('missing COAGsofa : {} ({}%)'.format(patients.COAGsofa.isna().sum(), patients.COAGsofa.isna().sum()*100//len(patients)))
print('missing LIVERsofa : {} ({}%)'.format(patients.LIVERsofa.isna().sum(), patients.LIVERsofa.isna().sum()*100//len(patients)))
print('missing CVsofa : {} ({}%)'.format(patients.CVsofa.isna().sum(), patients.CVsofa.isna().sum()*100//len(patients)))
print('missing CNSsofa : {} ({}%)'.format(patients.CNSsofa.isna().sum(), patients.CNSsofa.isna().sum()*100//len(patients)))
print('missing RENALsofa : {} ({}%)'.format(patients.RENALsofa.isna().sum(), patients.RENALsofa.isna().sum()*100//len(patients)))
```



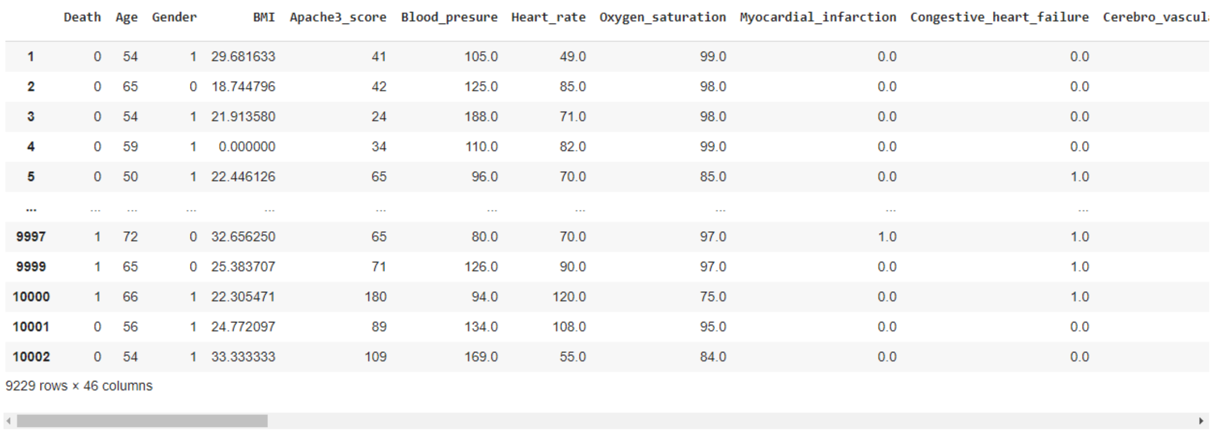
```
missing RESPsofa : 6865 (68%)
missing COAGsofa : 693 (6%)
missing LIVERsofa : 7342 (73%)
missing CVsofa : 22 (0%)
missing CNSsofa : 319 (3%)
missing RENALsofa : 562 (5%)
```

Figure 22: Dataset2: Percentage of missing values

5.2.2.7 Renaming features

At the end renaming the feature is essential for the better understanding.

```python
#renaming all Features
rename_columns = {'Age (rounded)':'Age',
                  'Apache3_ScoreSAS_24hr':'Apache3_score',
                  'Systolic BP Nearest ICU Admit':'Blood_presure',
                  'Heart Rate Nearest ICU Admit':'Heart_rate',
                  'Oxygen Saturation Nearest ICU Admit':'Oxygen_saturation',
                  'MI for CCI':'Myocardial_infarction',
                  'CHF for CCI':'Congestive_heart_failure',
                  'CVA  for CCI':'Cerebro_vascular_accident',
                  'Moderate/severe kidney disease  for CCI':'Kidney_disease_severity',
                  'Comorbidity score  for CCI':'Cormorbility_score',
                  'Prior DM for CCI':'Diabetes_mellitus',
                  'Prior cancer for CCI':'Prior_cancer',
                  'Prior lung dz  for CCI':'Prior_lung_disease',
                  'Catecholamines day 1':'Catecholamines_presence',
                  '# catecholamines day 1':'Catecholamines',
                  'First RR after CICU admission':'Respiration_rate',
                  'First Diastolic after CICU admission':'Diastolic',
                  'First MAP after CICU admission':'Mean_Arterial_pressure',
                  'First GCS Score after CICU admission':'Glasgow_coma_scale',
                  'Mechanical Ventilation within first 24 hours of ICU admit':'Ventilation',
                  'Hx of dialysis prior to hospital admission':'Previous_Dialysis',
                  'New Dialysis Start during ICU':'On_dialysis',
                  'ICU IABP':'On_intra-aortic_balloon_pump',
                  'PAC in ICU':'Pre-anesthesia_checkup',
                  'PRBC in ICU':'Packed_red_blood_cells',
                  'Cardiogenic shock':'Cardiogenic_shock',
                  'Heart failure':'Heart_failure',
                  'AF':'Irregular_heartbeat',
                  'Cardiac arrest':'Cardiac_arrest',
                  'ACS':'Acute_coronary_syndrome',
                  'CAD':'Coronary_artery_disease',
                  'Angiogram hospital stay':'Angiogram_hospital_stay',
                  'Coronary Stent hospital stay':'Coronary_stent'}
patients = patients.rename(columns=rename_columns)
```



	Death	Age	Gender	BMI	Apache3_score	Blood_presure	Heart_rate	Oxygen_saturation	Myocardial_infarction	Congestive_heart_failure	Cerebro_vascul
1	0	54	1	29.681633	41	105.0	49.0	99.0	0.0	0.0	
2	0	65	0	18.744796	42	125.0	85.0	98.0	0.0	0.0	
3	0	54	1	21.913580	24	188.0	71.0	98.0	0.0	0.0	
4	0	59	1	0.000000	34	110.0	82.0	99.0	0.0	0.0	
5	0	50	1	22.446126	65	96.0	70.0	85.0	0.0	1.0	
...	...	...	...	...	...	...	...	...	...	...	
9997	1	72	0	32.656250	65	80.0	70.0	97.0	1.0	1.0	
9999	1	65	0	25.383707	71	126.0	90.0	97.0	0.0	1.0	
10000	1	66	1	22.305471	180	94.0	120.0	75.0	0.0	1.0	
10001	0	56	1	24.772097	89	134.0	108.0	95.0	0.0	0.0	
10002	0	54	1	33.333333	109	169.0	55.0	84.0	0.0	0.0	

9229 rows × 46 columns

Figure 23: Dataset2: Cleaned dataset

5.2.2.8 Balancing class The target class of the dataset for classification is 'death'. It has 2 class labels '0' and '1'. It is unbalanced as we can see in the (Figure 24). Therefore, to get accurate prediction we have balanced the data(Figure 25).

```python
#unbalanced data
data.Death.value_counts().plot(kind='bar')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fb2adeda1d0>
```

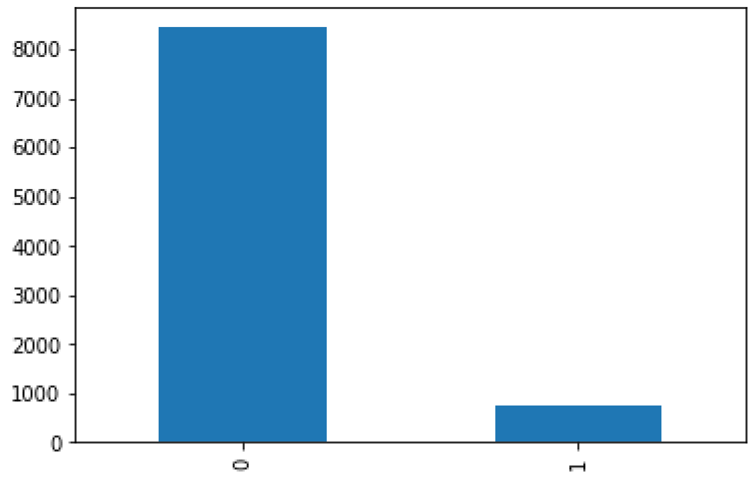


Figure 24: Dataset2: Class distribution before balancing

```python
#oversampling the data
patients_majority = data[data.Death==0]
patients_minority = data[data.Death==1]
patients_majority_length = len(patients_majority)

patients_sample = resample(patients_minority,
                                  replace=True,     # sample without replacement
                                  n_samples=patients_majority_length,    # to match majority
    class
                                  random_state=123)


data_oversampled = pd.concat([patients_majority, patients_sample])
```

```
data_oversampled.Death.value_counts().plot(kind='bar')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fb2ade3c110>
```

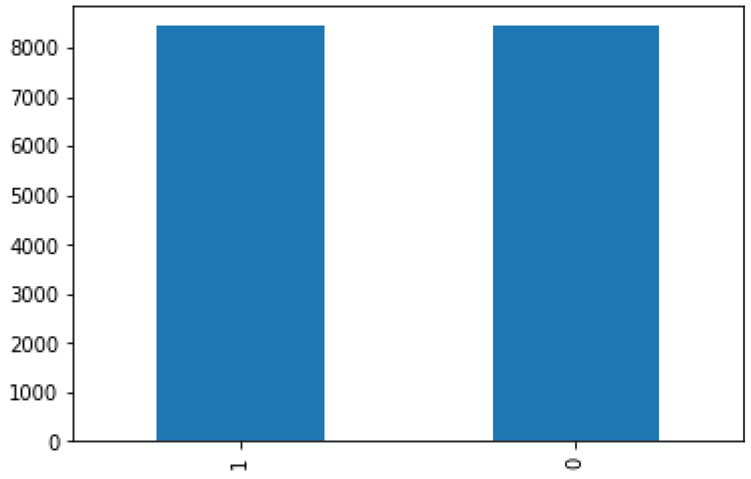


Figure 25: Dataset2: Class distribution after balancing

5.3 Data Analysis

Visualizing relationships and distribution of data gives information about the characteristics of the dataset. Graphs are a visual representation of the relationship between variables. They have been very useful for humans as they aid in understanding the data and deducing information from it, which is not evident in the dataset. So, we have provided some plots to help observe the trends in our pre-processed dataset.

5.3.1 Dataset 1

First let's start with the initial dataset(figure 26) with missing values.

```
sb.set(rc={'figure.figsize':(25,10)})
sb.heatmap(positive_patients.isin([97, 98, 99]))
```

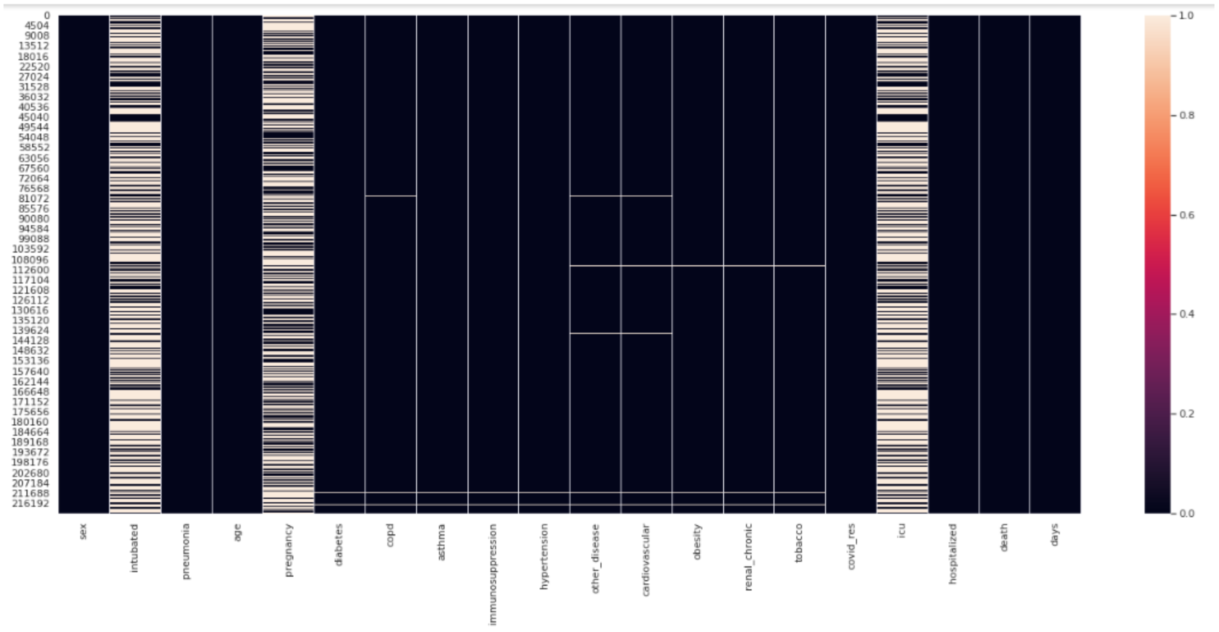


Figure 26: Dataset1: before Pre-processing

The (figure 27) shows the pre-processed dataset without missing values.

```python
sb.set(rc={'figure.figsize':(25,10)})
sb.heatmap(positive_patients.isin([98, 99]),cbar = False)
```

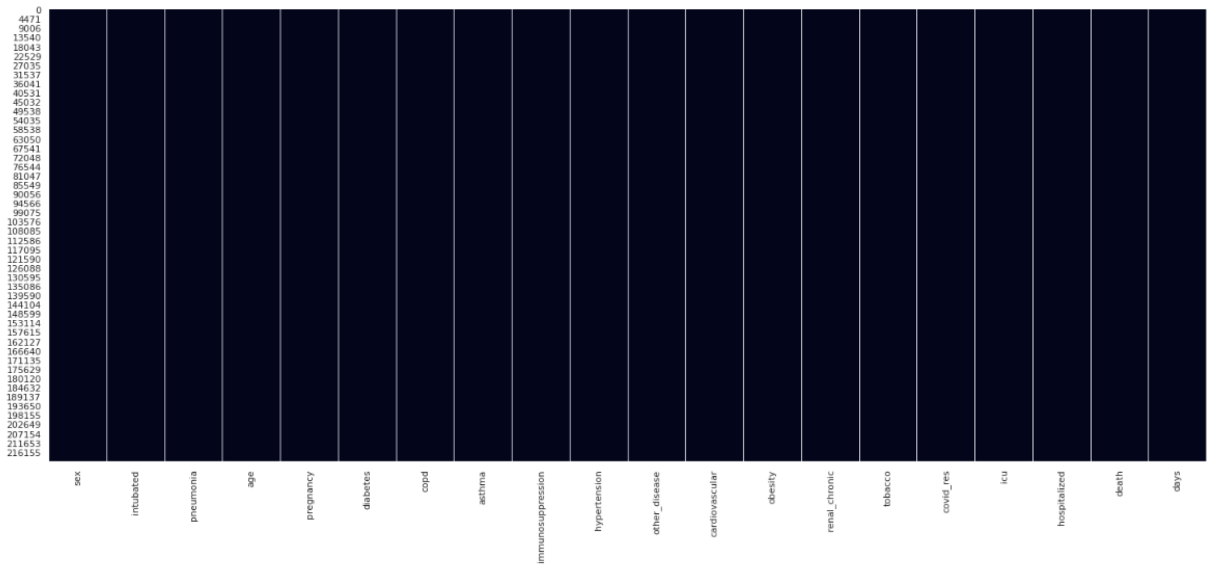


Figure 27: Dataset1: after Pre-processing

After that, we have presented the feature co-relation on our dataset. As a result(figure 28), we have not found any co-relation between features.

```python
#finding corelation coefficient by pearson's correlation
corr_matrix = positive_patients.corr().abs()
print(corr_matrix)
plt.figure(figsize=(20, 8))
sb.heatmap(corr_matrix, annot=True, cmap='Oranges')
```

```python

#selecting upper triangle
upper = corr_matrix.where(np.triu(np.ones(corr_matrix.shape), k=1).astype(np.bool))

#getting columns if it has greater than 0.8 corelation coefficiant
to_drop = [column for column in upper.columns if any(upper[column] > 0.8)]
print(to_drop)
```

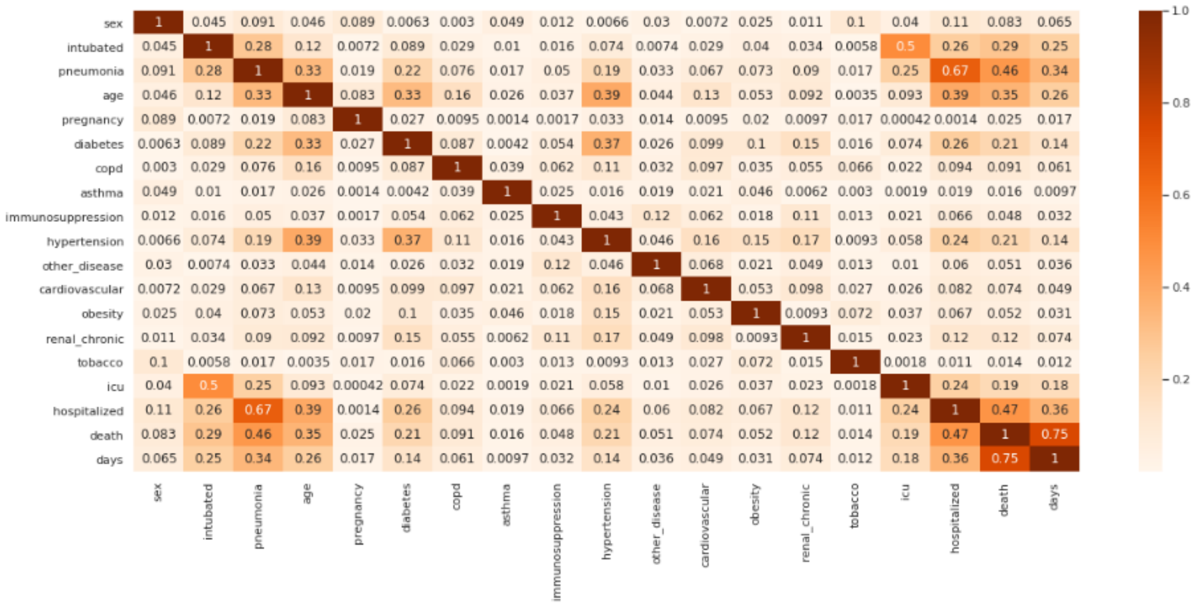


Figure 28: Dataset1: Feaure co-relation

We have also performed some operations on the dataset to check which feature and which condition affects the COVID+ patients fatality rate.

```python
neg_death = []
pos_death = []
nores_death = []
for index, patient in data.iterrows():
    if patient['covid_res'] == 2 and patient['death'] == 1:
        neg_death.append(patient)
    if patient['covid_res'] == 1 and patient['death'] == 1:
        pos_death.append(patient)
    if patient['covid_res'] == 3 and patient['death'] == 1:
        nores_death.append(patient)

fig = plt.figure()
ax = fig.add_axes([0,0,1,1])
ax.axis('equal')
result = ['positive', 'negative', 'pending result']
deaths = [len(pos_death),len(neg_death),len(nores_death)]
ax.pie(deaths, labels = result,autopct='%1.2f%%', explode = [0.05, 0.05, 0.05])
plt.show()
```

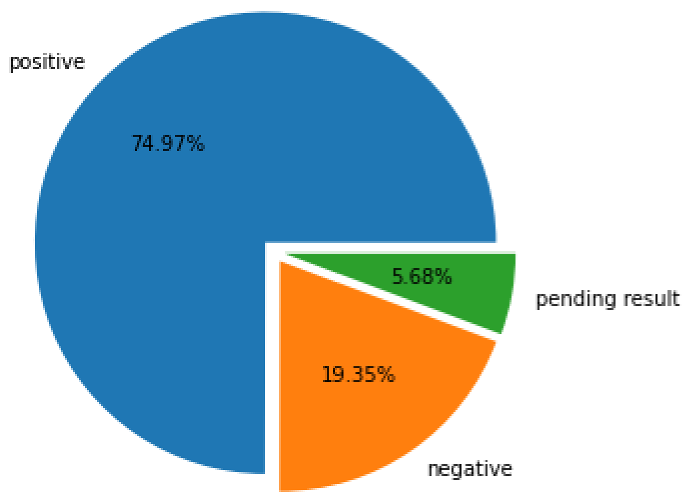


Figure 29: Dataset1: Fatality rate with respect to Covid result

We began by dis-aggregating data into covid results: Positive, Negative and Pending. Figure 29 describes the percentage of patients fatality rate with respect to their test results. It is clear that, COVID+ patients fatality rate is high as compare to patients with negative and pending results.
Similarly We have categorise the data into age groups. Figure 30 describes the total number of patients (Y - axis, count of patients) in a certain age range (X – axis, age band) who have tested COVID-19 positive. It can be seen that patients who belong to the age group 60-80 are most likely to die when compared to other age groups.

```
def age_band(age):

    if age<2:
        return 'Less than 2'
    elif (age>1) and (age<11):
        return '2-10'
    elif (age>10 and age<21):
        return '10-20'
    elif (age>20 and age<31):
        return '20-30'
    elif (age>30 and age<41):
        return '30-40'
    elif (age>40 and age<51):
        return '40-50'
    elif (age>50 and age<61):
        return '50-60'
    elif (age>60 and age<81):
        return '60-80'
    else:
        return 'Above 80'

positive_patients['Age band']=positive_patients['age'].apply(age_band)

sb.catplot('Age band',kind='count',hue='death',data=positive_patients,height=8,aspect=2,palette=
    'winter')
plt.xticks(size=15)
plt.title('COVID +ve case fatality with respect to age groups',size=20)
```

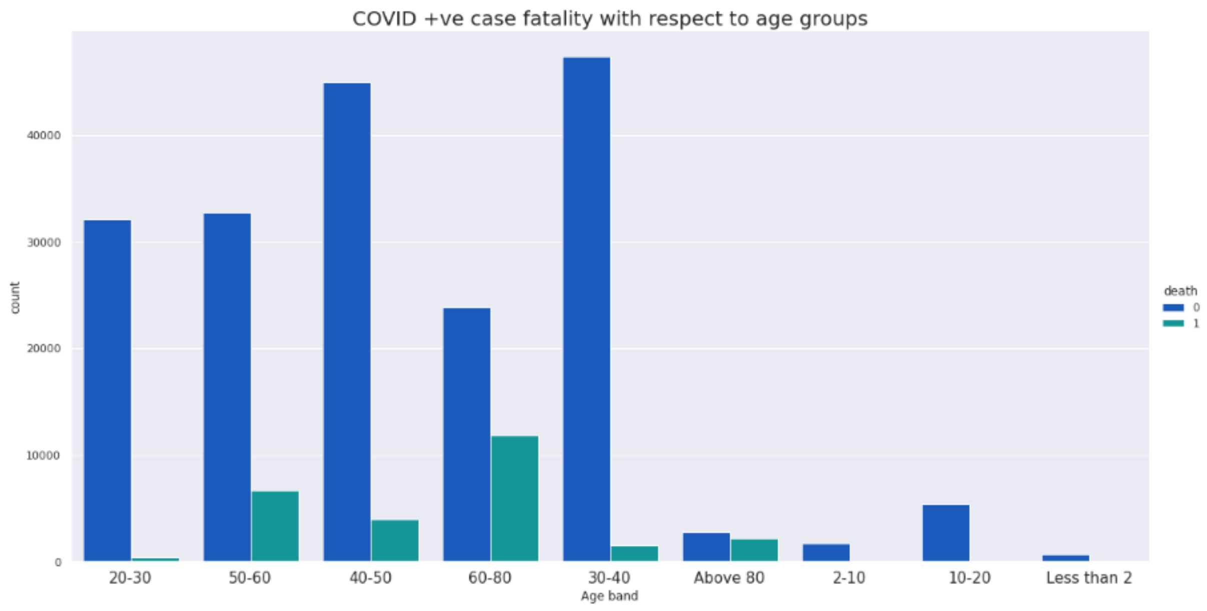


Figure 30: Dataset1: Fatality with respect to age groups

Next, we specified the fatality rate separately by considering COVID positive patients' co-morbidities: pneumonia, pregnancy, cardiovascular and obesity. According to figure 31, Y–axis denotes total number of patients and X–axis denotes patients' likely outcome (1 - yes, 0 - no) for a selected comorbidity. The result shows that the patients had a higher chance of dying with the fatality rate being 38.96%, 27.80%, 15.64% and 2.26% for patients with pneumonia, cardiovascular, obesity, and pregnancy respectively.

```
fig2=plt.figure(figsize=(15,15))
ax1=fig2.add_subplot(2,2,1)
ax2=fig2.add_subplot(2,2,2)
ax3=fig2.add_subplot(2,2,3)
ax4=fig2.add_subplot(2,2,4)
df_pneu=positive_patients[positive_patients['pneumonia']==1]
df_preg=positive_patients[positive_patients['pregnancy']==1]
df_card=positive_patients[positive_patients['cardiovascular']==1]
df_obes=positive_patients[positive_patients['obesity']==1]

sb.countplot('pneumonia',data=df_pneu,hue='death',ax=ax1,palette='gnuplot')
sb.countplot('pregnancy',data=df_preg,hue='death',ax=ax2,palette='summer')
sb.countplot('cardiovascular',data=df_card,hue='death',ax=ax3,palette='viridis')
sb.countplot('obesity',data=df_obes,hue='death',ax=ax4,palette='winter')

ax1.set_title('Pneumonia + COVID',size=20)
ax2.set_title('Pregnancy + COVID',size=20)
ax3.set_title('Cardiovascular disease + COVID',size=20)
ax4.set_title('Obesity + COVID',size=20)

ax1.set_xlabel('Fatality rate: {0:.2f} %'.format(100*df_pneu['death'].value_counts()[1]/df_pneu[
    'death'].shape[0]),size=15)
ax2.set_xlabel('Fatality rate: {0:.2f} %'.format(100*df_preg['death'].value_counts()[1]/df_preg[
    'death'].shape[0]),size=15)
```

```
ax3.set_xlabel('Fatality rate: {0:.2f} %'.format(100*df_card['death'].value_counts()[1]/df_card[
    'death'].shape[0]),size=15)
ax4.set_xlabel('Fatality rate: {0:.2f} %'.format(100*df_obes['death'].value_counts()[1]/df_obes[
    'death'].shape[0]),size=15)
```

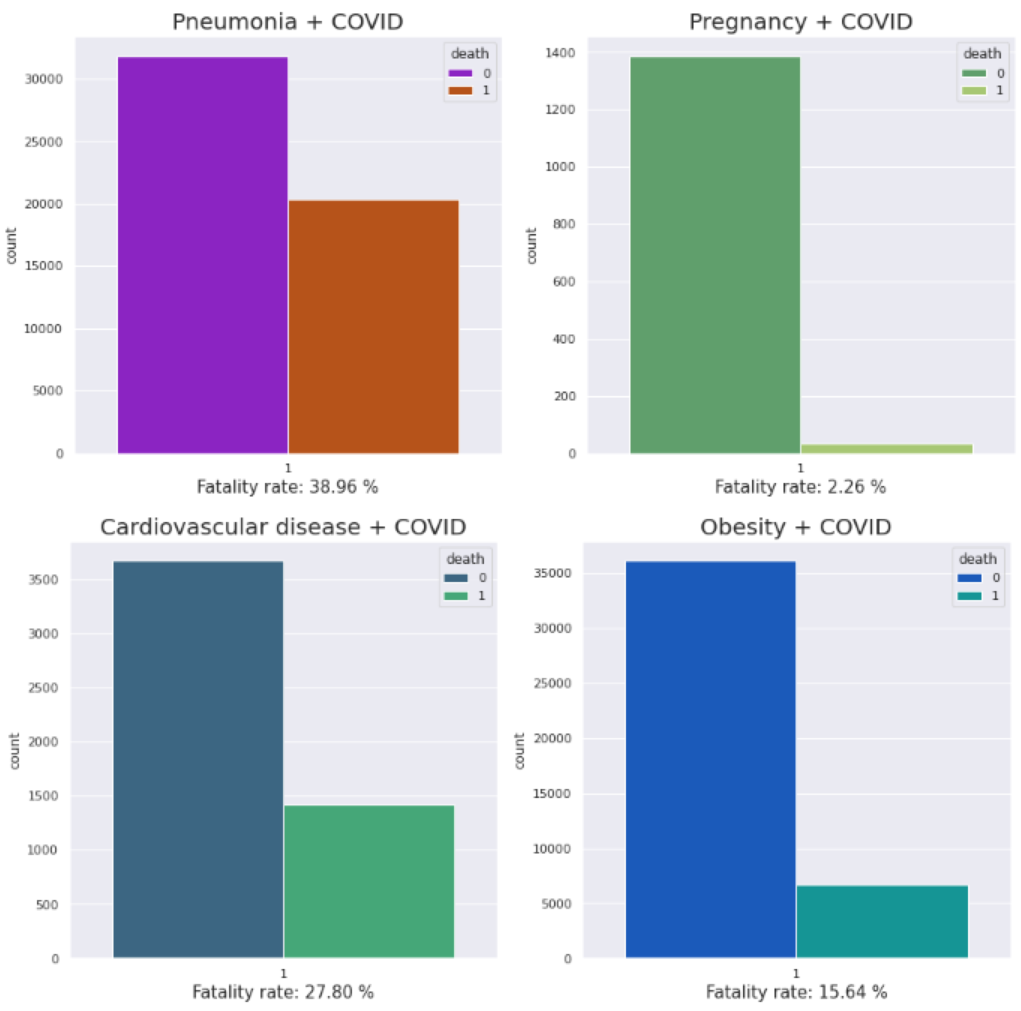


Figure 31: Dataset1: Fatality rate with respect to some co-morbidities

5.3.2 Dataset 2

First let's start with the initial dataset(figure 32) with missing values.

```
#missing vlalues visualization
msno.matrix(patients, labels=True)
```

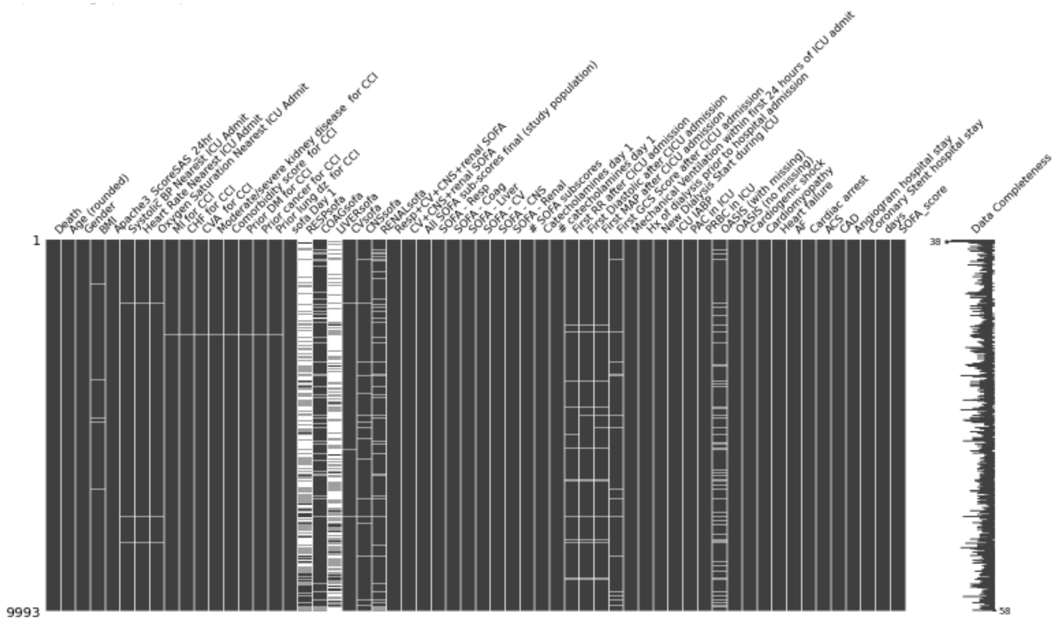


Figure 32: Dataset2: Before Pre-processing

The (figure 33) shows the pre-processed dataset without missing values

```
#missing vlalues visualization
msno.matrix(patients, labels=True)
msno.matrix(patients,labels=True)
```

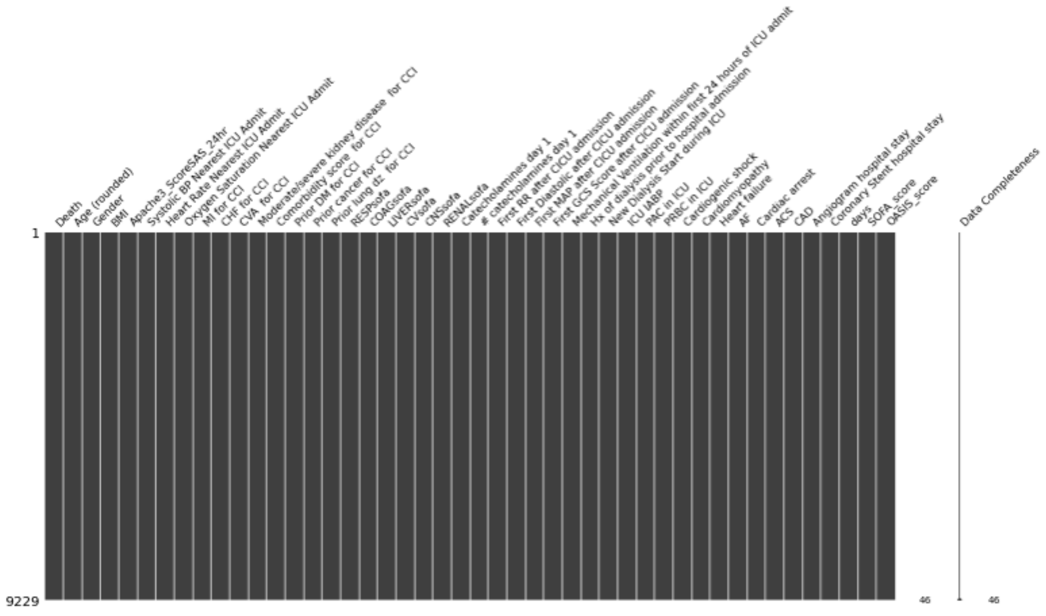
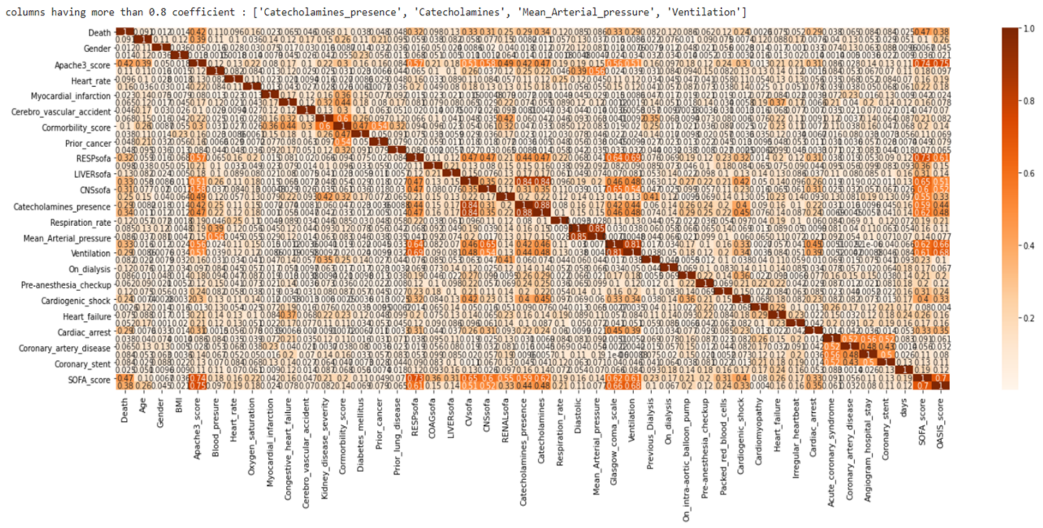


Figure 33: Dataset2: After Pre-processing

We represented the feature co-relation on our dataset. As a result(figure 34),we have found four co-related features. As a result, we have dropped those features.

```
#missing vlalues visualization
msno.matrix(patients, labels=True)
msno.matrix(patients,labels=True)
```



```
columns having more than 0.8 coefficient : ['Catecholamines_presence', 'Catecholamines', 'Mean_Arterial_pressure', 'Ventilation']
```

Figure 34: Dataset2: feature co-relation graph

5.4 Feature Selection using REF Method(Research Paper work)

We have applied the research paper to select the best features which play highest role to predict the status of patients death. Let's understand how this feature selection has been done using main part of the whole code.
After preprocessing on the dataset, It drops the feature having 99% constant values.

```python
#Remove features which have more than 99% constant values
df_features_constants = pd.DataFrame({'features':df_final.sum().index.tolist(),'values':df_final
    .sum().values.tolist()})
df_dropped_features_constants = df_final[df_features_constants[df_features_constants['values'] >
     df_final.shape[0]*0.01].features.tolist()]
X_df = df_final[df_dropped_features_constants.columns.tolist()]
```

The following code creates and processes a development set to train and evaluate candidate classifiers, and a first test set to evaluate the final classifier obtained from the whole development set.

```python
#Split pre-processed data into development and test sets in an 80:20 ratio
development_features, test_features, development_labels, test_labels = train_test_split(X_df,
    y_raw_df,test_size=0.2,stratify = y_raw_df, random_state=64)
```

Generate the traing and test dataset into csv files using below code.

```python
#Export development and test sets to csv files
development_features.to_csv(data_path+config_IO['df_development'],index=False)
development_labels.to_csv(data_path+config_IO['y_development'],index=False)
test_features.to_csv(data_path+config_IO['df_test'],index=False)
test_labels.to_csv(data_path+config_IO['y_test'],index=False)
```

During missing value imputation process, it scales all the continuous feature into z-score. and find highly co-related features to drop as they may be redundant.

```python
#Scale all continuous features into z-scores (standard scaling)
ss = StandardScaler()
```

```
development_features_ss = pd.DataFrame(ss.fit_transform(X_development_cont), columns =
    X_development_cont.columns.tolist())

# Create correlation matrix
corr_matrix = df_development_scaled.corr().abs()
# Select upper triangle of correlation matrix
upper = corr_matrix.where(np.triu(np.ones(corr_matrix.shape), k=1).astype(np.bool))
# Find index of feature columns with correlation greater than 0.90
to_drop = [column for column in upper.columns if any(upper[column] > 0.9)]
#Drop the highly correlated features from the dataframe
df_development_scaled_dropped =  df_development_scaled.drop(to_drop , axis=1 )
if len(to_drop) == 0:
    print('None of the columns is dropped due to highly correlated features/columns.')
else:
    print('Column(s) being dropped due to their high correlation(s) with other features/columns:
    ')
    print(to_drop)
```

Next, it will apply the RFE(recursive feature elimination) method, which automatically selects a subset of features with potentially better performance than all the features.

```
feature_df = df_development_imputed_dropped_high_corr
feature_np_array = feature_df.values
set_of_seed = range(11)
outcome_np_array = y_development_np
total_feature = feature_df.shape[1]
feature_name_list = list(feature_df.columns.values)
estimator_dict = {
            'LR': LogisticRegression(),
            'SVM': LinearSVC(),
            'RF': RandomForestClassifier()
}
# Container of the RFE results
all_result_df_list = []
result_df = pd.DataFrame(columns=['seed', 'classifier', 'num_features_picked', 'AUC'] +
    feature_name_list)
result_col = result_df.columns
result_container = dict()
# Iterating over classification algorithms
for est_name, est in estimator_dict.items():
    auc_mean_list = []
    auc_se_list = []
    # This for loop generates the training-holdout splits of the development set to evaluate the
     above algorithms
    # with different feature subsets
    for random_seed in set_of_seed:
        if (random_seed % 20 == 0) and (random_seed != 0):
            print('{}th round of {} training is running...'.format(random_seed, est_name))
        train_features, holdout_features, train_labels, holdout_labels = train_test_split(
            feature_df, outcome_np_array, test_size=0.25, stratify = outcome_np_array)
        est_var = est
        selector = RFE(estimator=est_var, step=1, n_features_to_select=1)
        selector = selector.fit(train_features, train_labels)
        rank_f = selector.ranking_
        for n_f in no_features_to_pick_list:
            selected_feat = rank_f <= n_f
            est_var = est
            est_var.fit(train_features.iloc[:,selected_feat], train_labels)
            if hasattr(est_var, "predict_proba"):
                prob_pos = est_var.predict_proba(holdout_features.iloc[:,selected_feat])[:,1]
            else:  # use decision function
                prob_pos = est_var.decision_function(holdout_features.iloc[:,selected_feat])
            auc = roc_auc_score(holdout_labels, prob_pos)
            dict_to_append = {'seed': random_seed, 'classifier': est_name, 'num_features_picked'
    : n_f, 'AUC': auc}
            for idx, f_name in enumerate(feature_name_list):
                dict_to_append[f_name] = selected_feat[idx]
            result_df = result_df.append(dict_to_append, ignore_index=True)
all_result_df_list.append(result_df)
```

This code block analyzes the results of the RFE process above to identify the top number of features for the three classification algorithms, and visualize them as a bar chart like in figure 22 and figure 23.

```
estimator_dict = {
            'LR': LogisticRegression,
            'SVM': LinearSVC,
            'RF': RandomForestClassifier
}
cp = sns.color_palette(n_colors=len(estimator_dict))
all_result_df = pd.read_csv(result_path+'RFE_result.csv')
feature_name_list = list(df_development_imputed_dropped_high_corr.columns.values)
estimator_full_name_dict = {
            'LR': 'Logistic Regression',
            'SVM': 'Support Vector Machine',
            'RF': 'Random Forest'}
set_of_seed = [range(11)]
# Code for bar plot
top_n_list = [top_N]
bar_color_list = ['b', 'olive', 'darkorange']
for top_n in top_n_list:
    top_n_df = all_result_df[all_result_df['num_features_picked'] == top_n]
    freq_df_list = []
    freq_df = pd.DataFrame(columns=['set_to_run', 'classifier', 'freq', 'feature'])
    feature_frequency_dict_by_clf = dict()
    fig2, axs = plt.subplots(3, figsize=(15,15))
    top_n_df_by_seed_set = top_n_df
    for est_idx, (est_name, est) in enumerate(estimator_dict.items()):
        top_n_df_by_clf = top_n_df_by_seed_set[top_n_df_by_seed_set['classifier']==est_name]
        for idx, f_name in enumerate(feature_name_list):
            freq = sum(top_n_df_by_clf[f_name].values)
            dict_to_append = {'classifier': est_name,
                              'freq': freq, 'feature': f_name}
            freq_df = freq_df.append(dict_to_append, ignore_index=True)
        freq_df_est = freq_df[freq_df['classifier']==est_name]
        freq_df_est = freq_df_est.sort_values(by=['freq'], ascending=False).iloc[:top_n]
        axs[est_idx] = sns.barplot(ax=axs[est_idx], x="freq", y="feature", data=freq_df_est,
    orient='h', color=cp[est_idx])
        axs[est_idx].set_yticks(range(top_n))
        axs[est_idx].set_title(estimator_full_name_dict[est_name], fontweight='bold',fontsize =
    20)
        axs[est_idx].set_xlim([0, set_of_seed[-1][-1]+1])
        axs[est_idx].set_xlabel('')
        new_yticks_label = []
        axs[est_idx].set_ylabel('')
        for idx, yticks in enumerate(axs[est_idx].get_yticklabels()):
            new_yticks_label.append("{} ({})".format(yticks.get_text(), freq_df_est['freq'].iloc
    [idx]))
        axs[est_idx].set_yticklabels(new_yticks_label, fontsize=18)
```

5.4.0.1 Dataset1

As we can see, after 11 iteration, RFE model chose 5 features: 'age, sex, pneumonia, intubated, copd, as a final selected features out of 23 features.

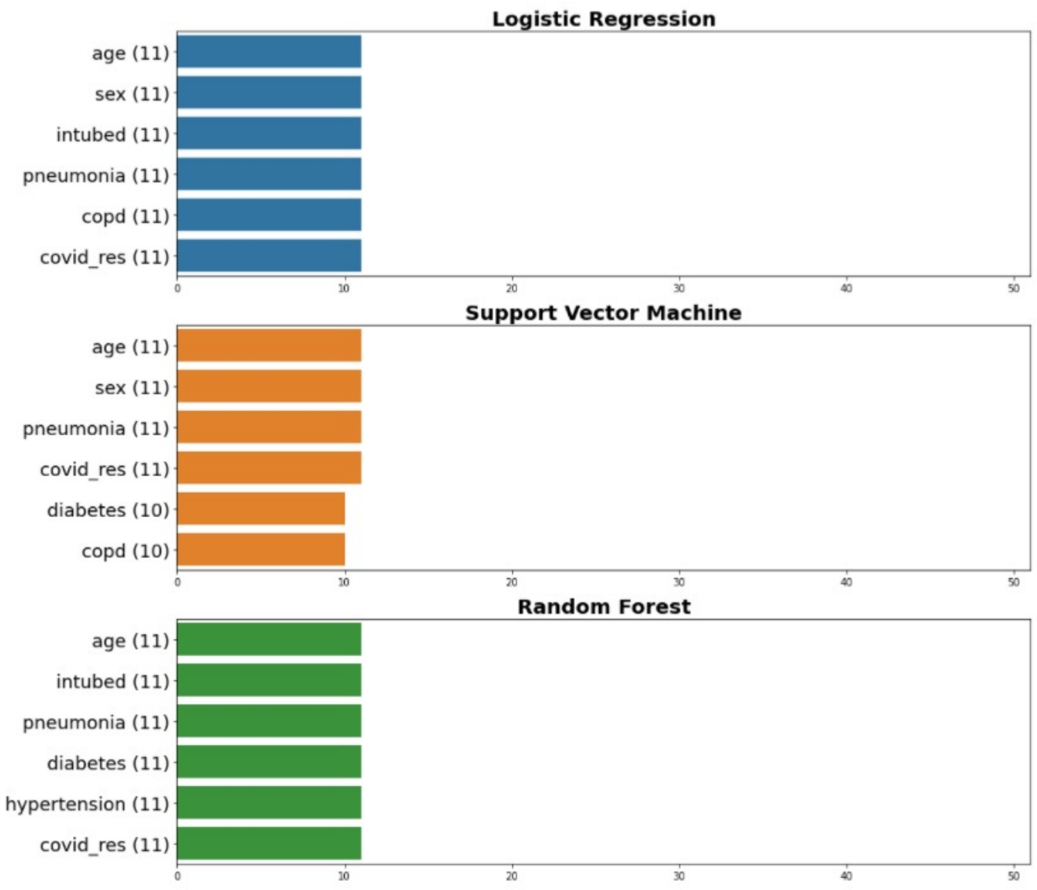
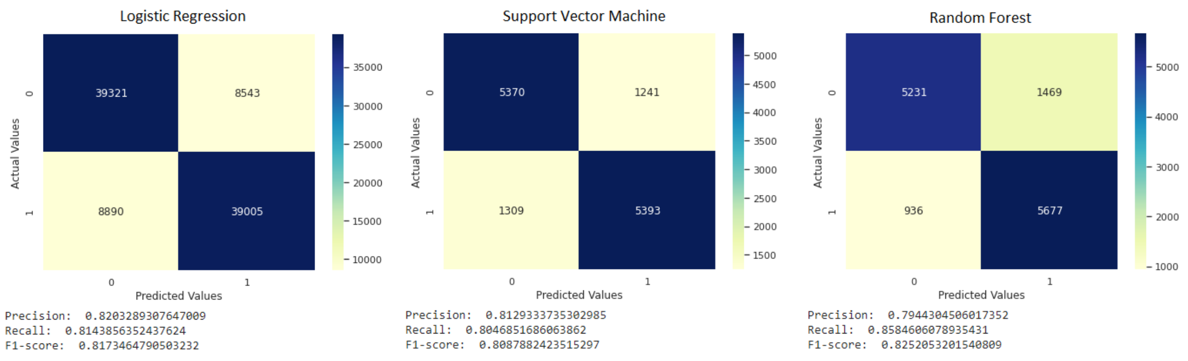


Figure 35: Dataset1: Feature Selection using RFE method

We have trained our model using these selected features by REF model.

```
target='death'
X = positive_patients_oversampled[['age','sex','intubated','pneumonia','copd']]
Y = positive_patients_oversampled[target]
X,Y = shuffle(X,Y)
X_train,X_test,Y_train,Y_test = train_test_split(X,Y,test_size=0.25,random_state=12)

#Logistic Regression
logistic_regression = LogisticRegression(max_iter=200)
logistic_regression.fit(X_train,Y_train)
Y_pred_logistic_prob=logistic_regression.decision_function(X_test)
Y_pred_logistic_prob.reshape(len(Y_pred_logistic_prob))

Y_pred_logistic = logistic_regression.predict(X_test)

confusion_matrix = pd.crosstab(Y_test, Y_pred_logistic, rownames=['Actual Values'], colnames=['
    Predicted Values'])
sb.heatmap(confusion_matrix, annot=True, fmt='d', cmap='YlGnBu')
plt.show()
```

```

print('Precision: ',metrics.precision_score(Y_test, Y_pred_logistic))
print('Recall: ',metrics.recall_score(Y_test, Y_pred_logistic))
print('F1-score: ',metrics.f1_score(Y_test, Y_pred_logistic))

fpr_logistic, tpr_logistic, thresholds_logistic = roc_curve(Y_test, Y_pred_logistic_prob)
auc_logistic = auc(fpr_logistic, tpr_logistic)

plt.figure(1)
plt.plot([0, 1], [0, 1], 'k--')
plt.plot(fpr_logistic, tpr_logistic, label='Logistic Regression (AUC = {:.3f})'.format(
    auc_logistic))
plt.xlabel('False positive rate')
plt.ylabel('True positive rate')
plt.title('ROC curve')
plt.legend(loc='best')
plt.show()
```



```
Precision:  0.8203289307647009
Recall:  0.8143856352437624
F1-score:  0.8173464790503232
```

```
Precision:  0.8129333735302985
Recall:  0.8046851686063862
F1-score:  0.8087882423515297
```

```
Precision:  0.7944304506017352
Recall:  0.8584606078935431
F1-score:  0.8252053201540809
```

Figure 36: Dataset1: Accuracy with RFE features

5.4.0.2 Dataset2 As we can see, after 51 iteration, RFE model chose 5 features: 'OASIS_Score, SOFA_Score, Blood_pressure, BMI, Ventilation' as a final selected features out of 66 features.

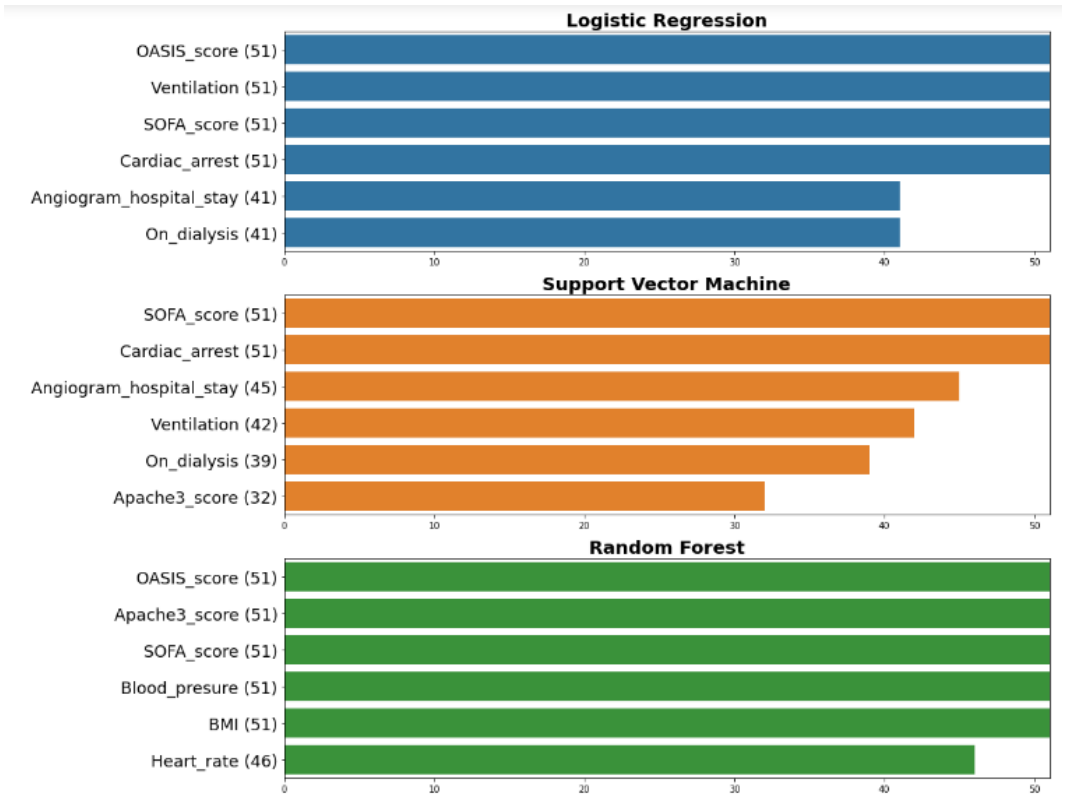
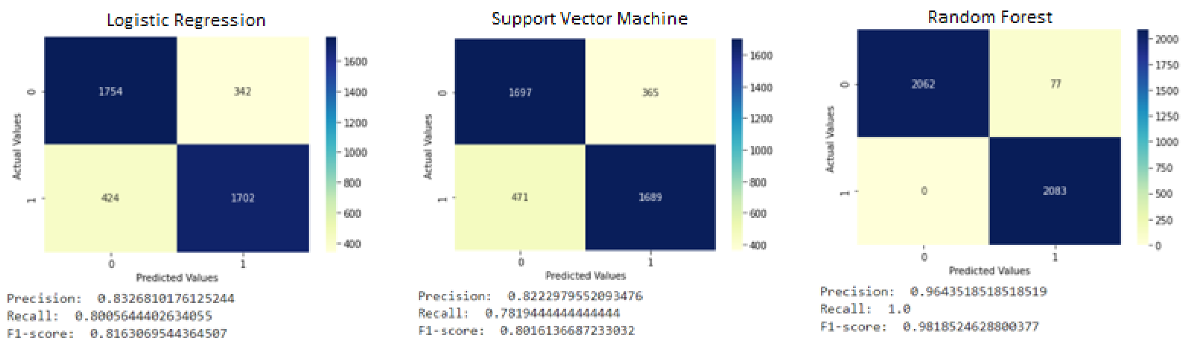


Figure 37: Dataset2: Feature Selection using RFE method



```
Precision:  0.8326810176125244
Recall:  0.8005644402634055
F1-score:  0.8163069544364507
```

```
Precision:  0.8222979552093476
Recall:  0.7819444444444444
F1-score:  0.8016136687233032
```

```
Precision:  0.9643518518518519
Recall:  1.0
F1-score:  0.9818524628800377
```

Figure 38: Dataset2: Accuracy with RFE features

We got satisfied accuracy using selected features but still We wanted to check the accuracy for predicted results using all the features. Therefore, We have applied some selected algorithms on our cleaned datasets in Data modeling section.

5.5 Data Modeling

In this section, we briefly introduce the methodologies used to build the model. We trained the model using four different supervised classification methodologies: Logistic Regression, Decision Tree, Random Forests and Neural Network with six different regression algorithms: Linear Regression, Decision Tree, Random Forest Gradient Boosting, Support Vector Machine and Neural Network. All these algorithms are widely used for developing healthcare applications and yield good results with high performing models. Our main goal is to predict a patient's likely condition by using trained classification and Regression models. The feature 'death', has been used as the target variable for classification and 'days' has been used for regression model. We used various python libraries for implementing this project. The dataset was handled using the pandas Data Frame and numpy. The plotting operations were performed using the matplotlib library. The methods available with the scikit-learn library were used for evaluating our algorithms. For the classification and regression algorithms, we have used available scikit-learn libraries for specific algorithms such as RandomForestClassifier, DecisionTreeClassifier, SVM, LogisticRegression LinearRegression, etc. Likewise to apply neural network we have used Keras libraries. Lastly, the pickle module was used for dumping the model into a file, which was used to predict an outcome for the data entered via the web application developed by us. The next section gives a brief explanation of our built models. To train and test the model we have applied above listed models on our cleaned datasets as below:

```
#Required libraries for modeling
#To data cleaning and other operations
import pandas as pd
import numpy as np
from datetime import datetime

#sklearn for model training and testing
from sklearn.linear_model import LogisticRegression, LinearRegression
from sklearn.svm import SVC
from sklearn.utils import resample,shuffle
from sklearn.ensemble import RandomForestClassifier, RandomForestRegressor
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import cross_val_predict, train_test_split, cross_val_score
from sklearn import metrics
from sklearn.metrics import roc_curve
from sklearn.metrics import auc
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import MinMaxScaler

#matplotlab for plots
import matplotlib.pyplot as plt
#keras for neural network
from keras.models import Sequential
from keras.layers import Dense
from keras.wrappers.scikit_learn import KerasRegressor
import seaborn as sb
%matplotlib inline
```

```
#spliting data into train and test set.
target='death'
X = positive_patients_oversampled.drop([target,'days'],axis=1)
Y = positive_patients_oversampled[target]
X,Y = shuffle(X,Y)
```

```
X_train,X_test,Y_train,Y_test = train_test_split(X,Y,test_size=0.25,random_state=12)
```

Classification Algorithms:

```
#Logistic Regression
logistic_regression = LogisticRegression(max_iter=200)
logistic_regression.fit(X_train,Y_train)
Y_pred_logistic_prob=logistic_regression.decision_function(X_test)
Y_pred_logistic_prob.reshape(len(Y_pred_logistic_prob))

Y_pred_logistic = logistic_regression.predict(X_test)

confusion_matrix = pd.crosstab(Y_test, Y_pred_logistic, rownames=['Actual Values'], colnames=['
    Predicted Values'])
sb.heatmap(confusion_matrix, annot=True, fmt='d', cmap='YlGnBu')
plt.show()
```

```
#Decision Tree
decisionTree_classifier = DecisionTreeClassifier()
decisionTree_classifier.fit(X_train,Y_train)
Y_pred_decision = decisionTree_classifier.predict(X_test)
Y_pred_decision_prob = decisionTree_classifier.predict_proba(X_test)

confusion_matrix = pd.crosstab(Y_test, Y_pred_decision, rownames=['Actual Values'], colnames=['
    Predicted Values'])
sb.heatmap(confusion_matrix, annot=True, fmt='d', cmap='YlGnBu')
plt.show()
```

```
#Random Forest
random_forest_classifier = RandomForestClassifier()
random_forest_classifier.fit(X_train,Y_train)
Y_pred_forest = random_forest_classifier.predict(X_test)

Y_pred_forest_prob = random_forest_classifier.predict_proba(X_test)

confusion_matrix = pd.crosstab(Y_test, Y_pred_forest, rownames=['Actual Values'], colnames=['
    Predicted Values'])
sb.heatmap(confusion_matrix, annot=True, fmt='d', cmap='YlGnBu')
plt.show()
```

```
#Neural Nework
#defining Model
classification_model = Sequential()
classification_model.add(Dense(25, input_dim=17, activation='relu'))
classification_model.add(Dense(10, activation='relu'))
classification_model.add(Dense(1, activation='sigmoid'))

classification_model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy','
    binary_crossentropy'])

history = classification_model.fit(X_train, Y_train, epochs=100, batch_size=1000)
#predicting for testing samples
Y_pred_keras = classification_model.predict(X_test)
Y_pred_keras_prob = np.array(Y_pred_keras)
#converting probabilty to 0 and 1 class at 0.5 threshold value
for i in range(len(Y_pred_keras)):
  if Y_pred_keras[i][0]>0.5:
    Y_pred_keras[i][0] = 1
  else:
```

```python
    Y_pred_keras[i][0] = 0
#reshaping predicted values
Y_pred_keras = Y_pred_keras.reshape(len(Y_pred_keras))
```

Regression Algorithms:

```python
#Linear Regressor
linear_regression = LinearRegression()
linear_regression.fit(X_train,Y_train)
Y_pred = np.array(linear_regression.predict(X_test))

print('Mean Absolute Error :',mean_absolute_error(Y_test,Y_pred))
print('Mean SquaredError :',mean_squared_error(Y_test,Y_pred))
print('r2 score :',r2_score(Y_test,Y_pred))

plt.figure(figsize=(5,5))
plt.scatter(Y_test, Y_pred, c='crimson')

p1 = max(max(Y_pred), max(Y_test))
p2 = min(min(Y_pred), min(Y_test))
plt.plot([p1, p2], [p1, p2], 'b-')
plt.xlabel('True Values', fontsize=15)
plt.ylabel('Predictions', fontsize=15)
plt.axis('equal')
plt.show()

pd.DataFrame({'Actual': np.array(Y_test),'Predicted':np.array(Y_pred)}).plot(kind='kde')

mae_scores['linear'] = mean_absolute_error(Y_test,Y_pred)
mse_scores['linear'] = mean_squared_error(Y_test,Y_pred)
r2_scores['linear'] = r2_score(Y_test,Y_pred)
```

```python
#SVM Regressor
from sklearn import svm
regr = svm.SVR()
regr.fit(X_train, Y_train)
Y_pred = regr.predict(X_test)

print('Mean Absolute Error :',mean_absolute_error(Y_test,Y_pred))
print('Mean SquaredError :',mean_squared_error(Y_test,Y_pred))
print('r2 score :',r2_score(Y_test,Y_pred))
```

```python
#Decision Tree Regressor
from sklearn.tree import DecisionTreeRegressor
clf = DecisionTreeRegressor()
clf.fit(X_train, Y_train)
Y_pred = clf.predict(X_test)

print('Mean Absolute Error :',mean_absolute_error(Y_test,Y_pred))
print('Mean SquaredError :',mean_squared_error(Y_test,Y_pred))
print('r2 score :',r2_score(Y_test,Y_pred))
```

```python
#Random Forest Regressor
rf_regr = RandomForestRegressor().fit(X_train, Y_train)
Y_pred = rf_regr.predict(X_test)

print('Mean Absolute Error :',mean_absolute_error(Y_test,Y_pred))
print('Mean SquaredError :',mean_squared_error(Y_test,Y_pred))
print('r2 score :',r2_score(Y_test,Y_pred))
```

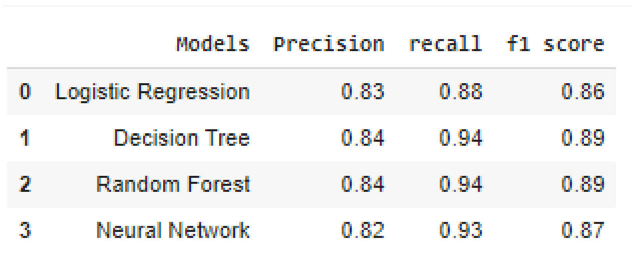
```python
#Gradient Boosting Regressor
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.metrics import mean_squared_error

params = {'n_estimators': 350,
          'max_depth': 4,
          'min_samples_split': 5,
          'learning_rate': 0.01,
          'loss': 'ls'}
reg = GradientBoostingRegressor(**params)
reg.fit(X_train, Y_train)
Y_pred = reg.predict(X_test)

print('Mean Absolute Error :',mean_absolute_error(Y_test,Y_pred))
print('Mean Squared Error :',mean_squared_error(Y_test,Y_pred))
print('r2 score :',r2_score(Y_test,Y_pred))
```

```python
model = Sequential()
model.add(Dense(30, input_dim=17, activation='relu', kernel_initializer='normal'))
model.add(Dense(10, kernel_initializer='normal'))
model.add(Dense(1, kernel_initializer='normal'))
model.compile(loss='mae', optimizer='adam' ,metrics=['mse','mean_absolute_error'])
history = model.fit(X_train, Y_train, epochs=250, batch_size=1000 )
Y_pred = model.predict(X_test)
Y_pred = Y_pred.reshape(len(Y_pred))

print('Mean Absolute Error :',mean_absolute_error(Y_test,Y_pred))
print('Mean SquaredError :',mean_squared_error(Y_test,Y_pred))
print('r2 score :',r2_score(Y_test,Y_pred))
```

We have applied these classifiers and Regressors on both the datasets.

5.6 Data Validation

5.6.0.1 Dataset1 In Classification comparison table(Figure 39) and ROC-curve graph(figure 40) we can clearly see that, Random Forest gives the best predictions as compare to other 3 classifiers.



	Models	Precision	recall	f1 score
0	Logistic Regression	0.83	0.88	0.86
1	Decision Tree	0.84	0.94	0.89
2	Random Forest	0.84	0.94	0.89
3	Neural Network	0.82	0.93	0.87

Figure 39: Dataset1:Classification model comparison Table

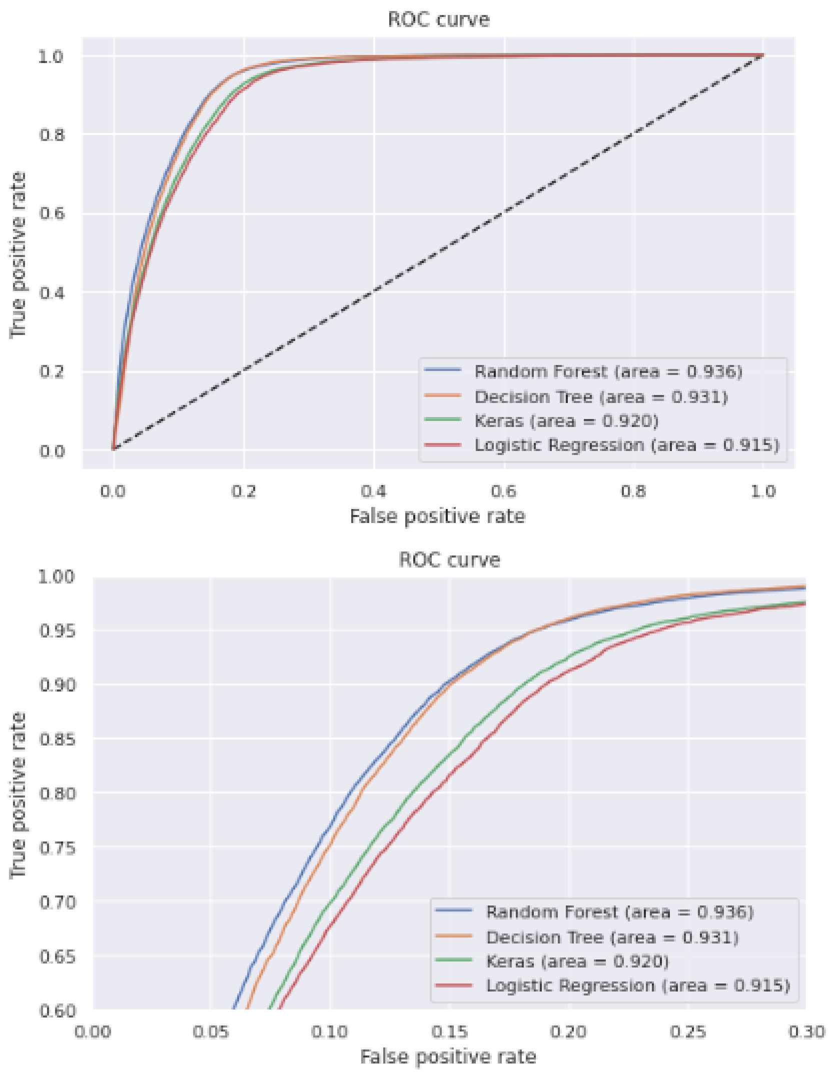
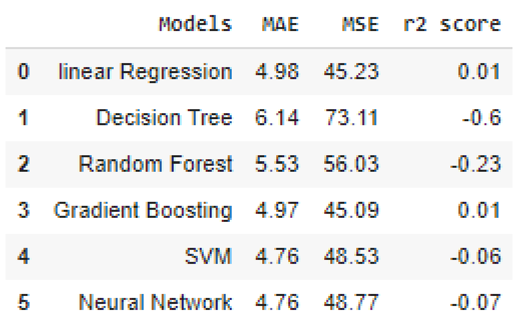


Figure 40: Dataset1:Classification models ROC Curve

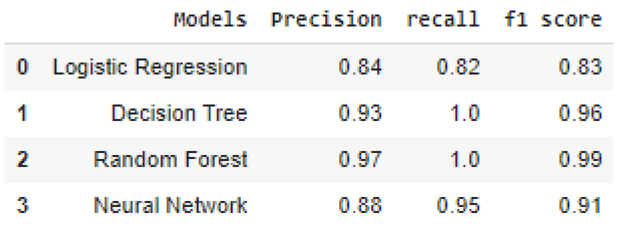
In Regression comparison table(Figure 41), the SVM and Neural network give the best predictions as compare to other 4 regressors. As a result, we chose Neural network as a final model.



	Models	MAE	MSE	r2 score
0	linear Regression	4.98	45.23	0.01
1	Decision Tree	6.14	73.11	-0.6
2	Random Forest	5.53	56.03	-0.23
3	Gradient Boosting	4.97	45.09	0.01
4	SVM	4.76	48.53	-0.06
5	Neural Network	4.76	48.77	-0.07

Figure 41: Dataset1:Regression model comparison Table

5.6.0.2 Dataset2 In Classification comparison table(Figure 42) and ROC-curve graph(figure 43) we can clearly see that, Random Forest gives the best predictions as compare to other 3 classifiers.



	Models	Precision	recall	f1 score
0	Logistic Regression	0.84	0.82	0.83
1	Decision Tree	0.93	1.0	0.96
2	Random Forest	0.97	1.0	0.99
3	Neural Network	0.88	0.95	0.91

Figure 42: Dataset2:Classification model comparison Table

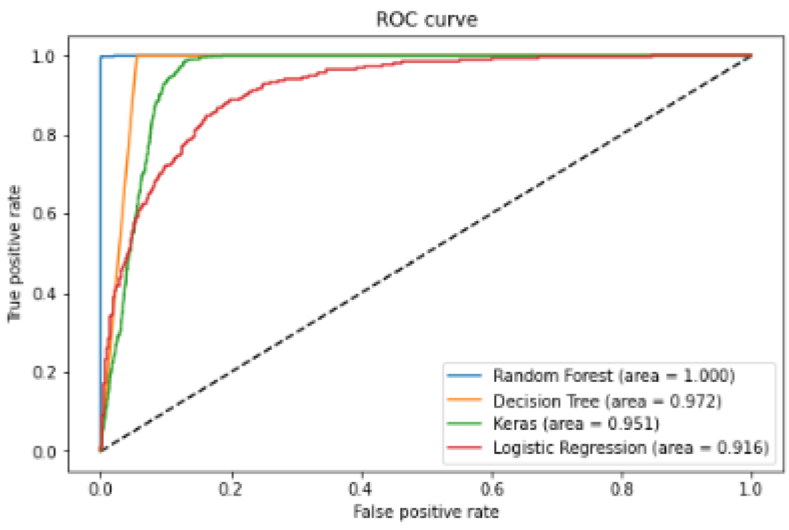
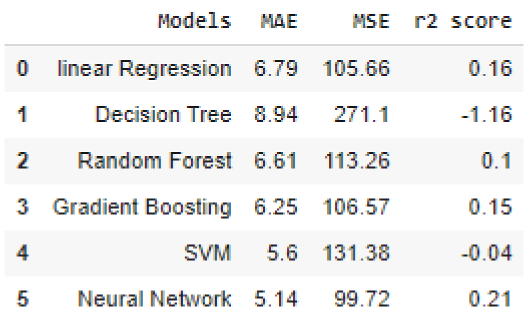


Figure 43: Dataset2:Classification models ROC Curve

In Regression comparison table(Figure 44), the Neural network gives the best predictions as compare to other 4 regressors. Therefore, we chose Neural network as a final model.



	Models	MAE	MSE	r2 score
0	linear Regression	6.79	105.66	0.16
1	Decision Tree	8.94	271.1	-1.16
2	Random Forest	6.61	113.26	0.1
3	Gradient Boosting	6.25	106.57	0.15
4	SVM	5.6	131.38	-0.04
5	Neural Network	5.14	99.72	0.21

Figure 44: Dataset2:Regression model comparison Table

5.7 Deployment

After building and evaluating the model, a user interface was developed (figure 45) to enable users to get a prediction on a COVID-19 and CICU patient's likely condition by inputting values of the patient's details, including pre-existing health conditions, age and gender. The user inputted values are submitted, a record with these values is formed in the back end to be fed to a model for a prediction to be made. We used that model in the user interface to predict the output to be displayed, using Flask framework. Our flask project consists of 2 files to work on, one is the Python file consisting of all the back end part and the other is the HTML file that contains the front end of the interface that can be seen by the user. There is also a directory in which the files containing models are stored. The import and export of the trained model was done using the pickle module available in Python. The model was written to a file using the dump() method. This file was then imported and made available in the Python file of the project. The flow of our application is like this: As soon as the user inputs the values, they are collected by the python file and stored in a Data Frame. This record is then used by the imported trained model to predict the outcome. The prediction made by the model is returned and is sent to the application's interface to be displayed to the user.

```python
#Pickle module to generate trained model file
pickle.dump(random_forest_classifier, open('sofa_classification.pkl', 'wb'))
pickle.dump(scaler, open('sofa_scaler.pkl', 'wb'))
model.save("sofa_regression.hdf5")
```

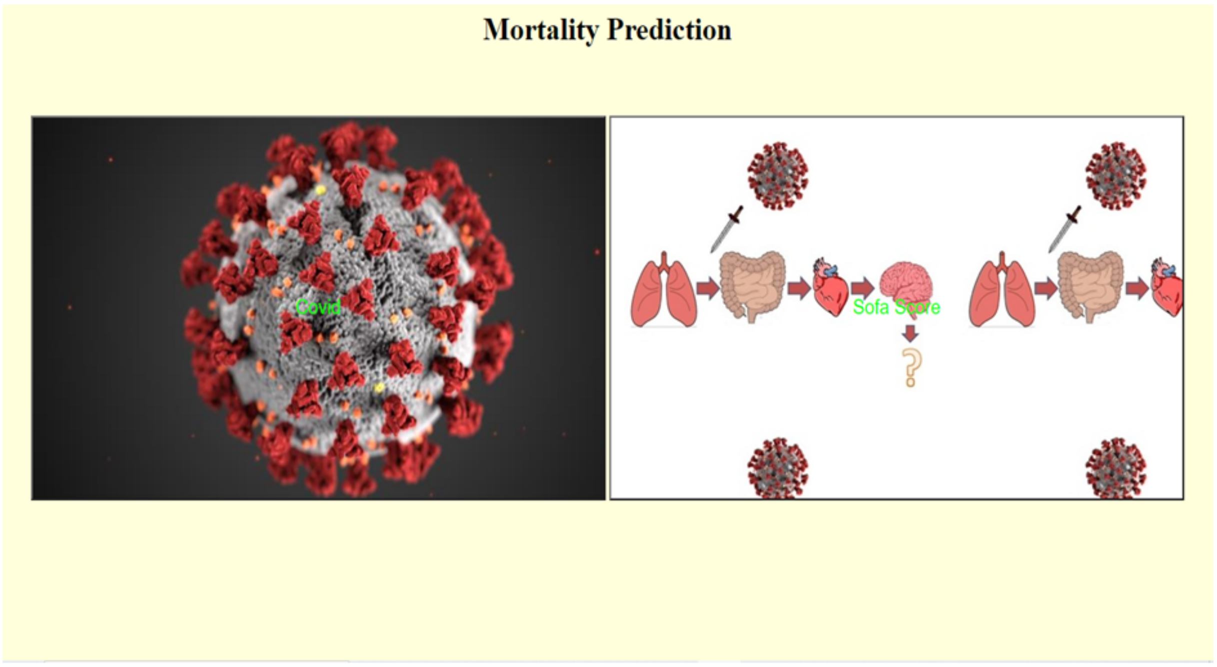
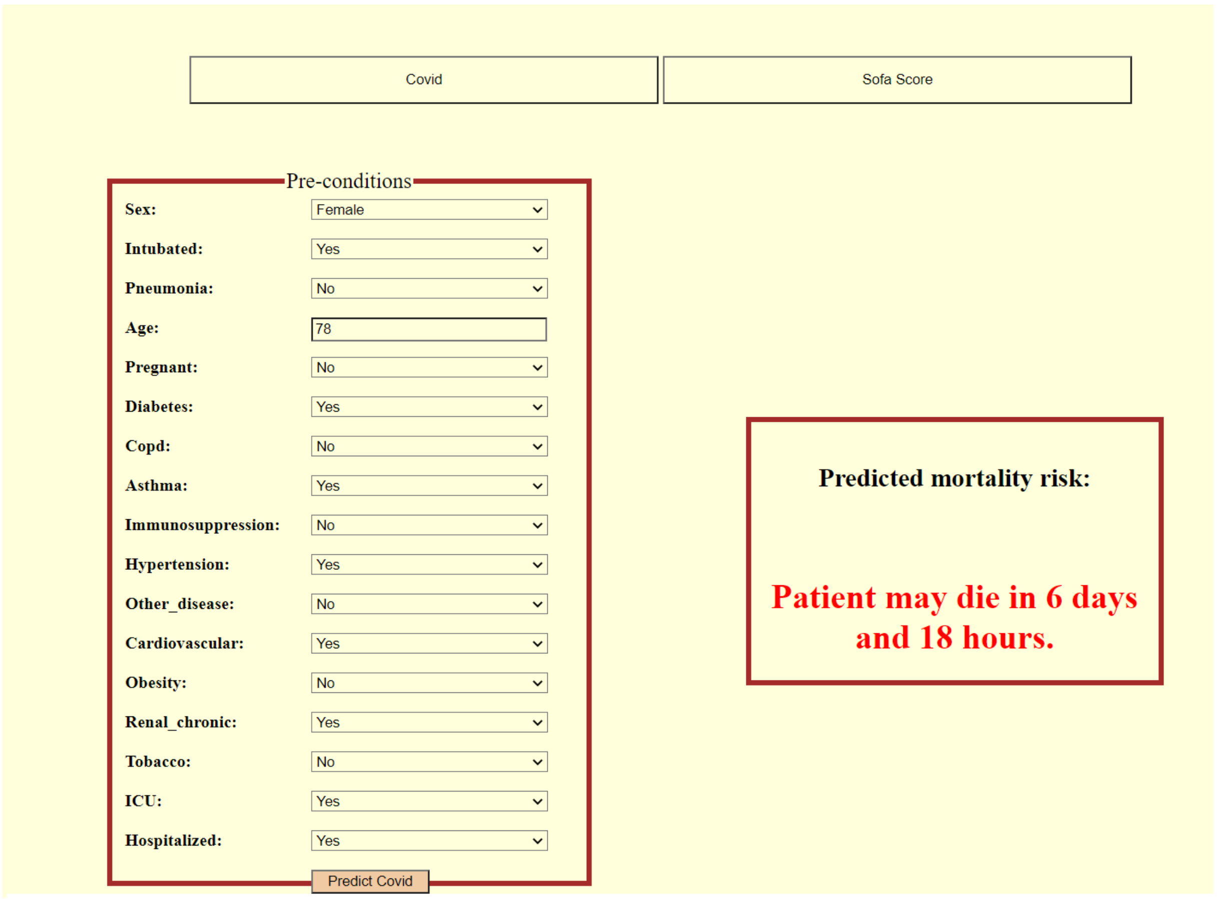


Figure 45: API:Dashboard



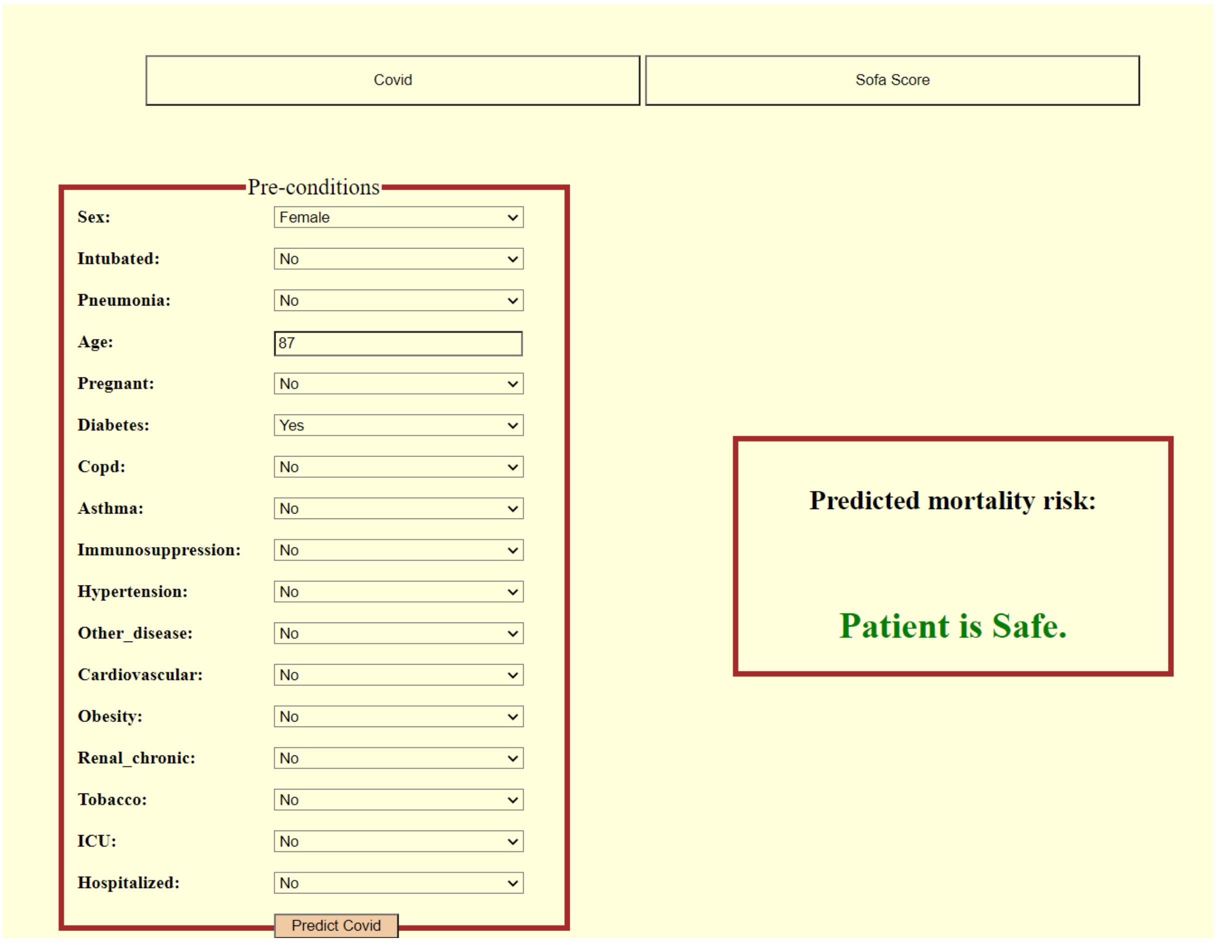
Pre-conditions

Sex:	Female
Intubated:	Yes
Pneumonia:	No
Age:	78
Pregnant:	No
Diabetes:	Yes
Copd:	No
Asthma:	Yes
Immunosuppression:	No
Hypertension:	Yes
Other_disease:	No
Cardiovascular:	Yes
Obesity:	No
Renal_chronic:	Yes
Tobacco:	No
ICU:	Yes
Hospitalized:	Yes

Predicted mortality risk:

Patient may die in 6 days and 18 hours.

Figure 46: COVID-19: Death Prediction



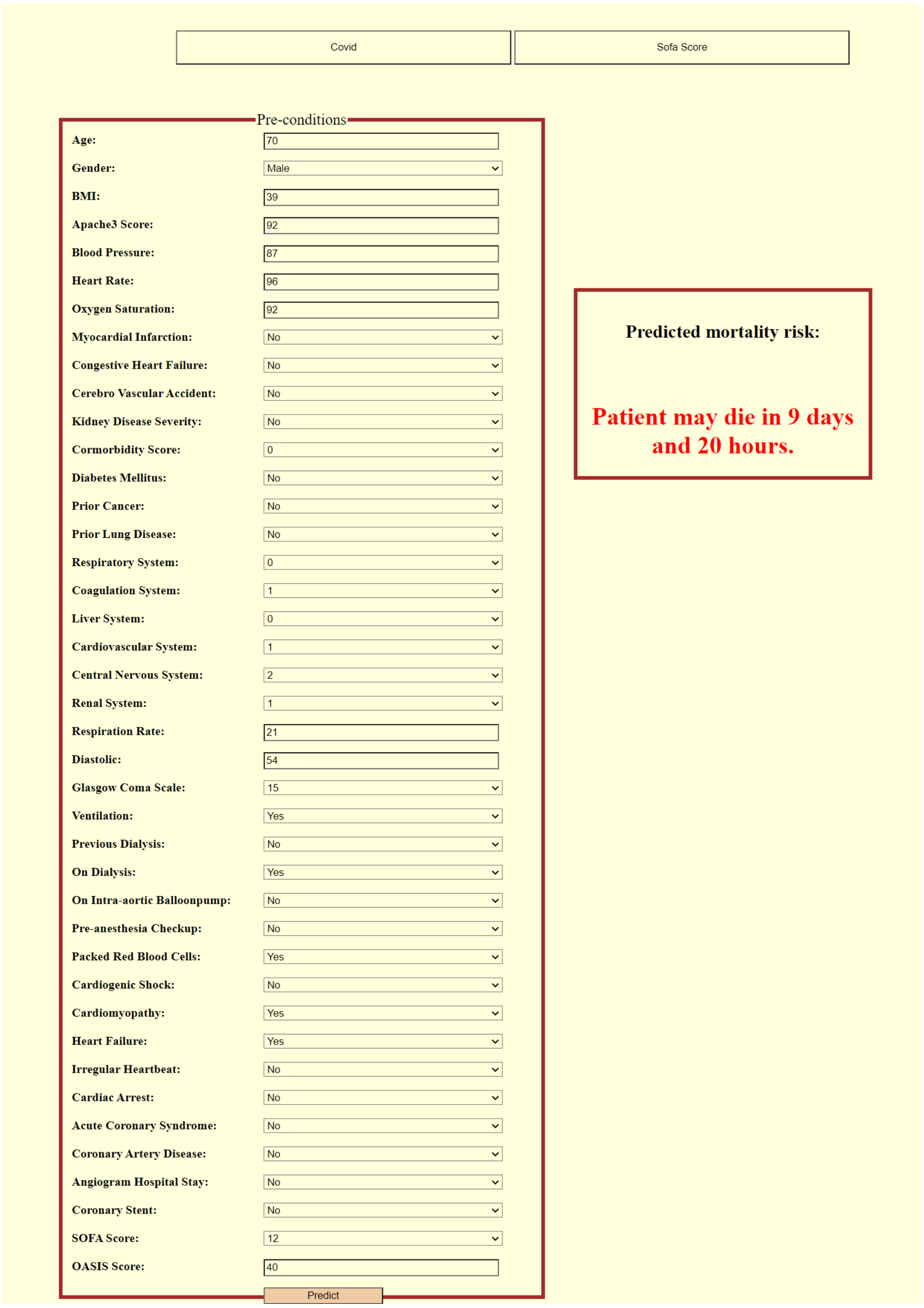
Pre-conditions

Sex:	Female
Intubated:	No
Pneumonia:	No
Age:	87
Pregnant:	No
Diabetes:	Yes
Copd:	No
Asthma:	No
Immunosuppression:	No
Hypertension:	No
Other_disease:	No
Cardiovascular:	No
Obesity:	No
Renal_chronic:	No
Tobacco:	No
ICU:	No
Hospitalized:	No

Predicted mortality risk:

Patient is Safe.

Figure 47: COVID-19: Survival Prediction



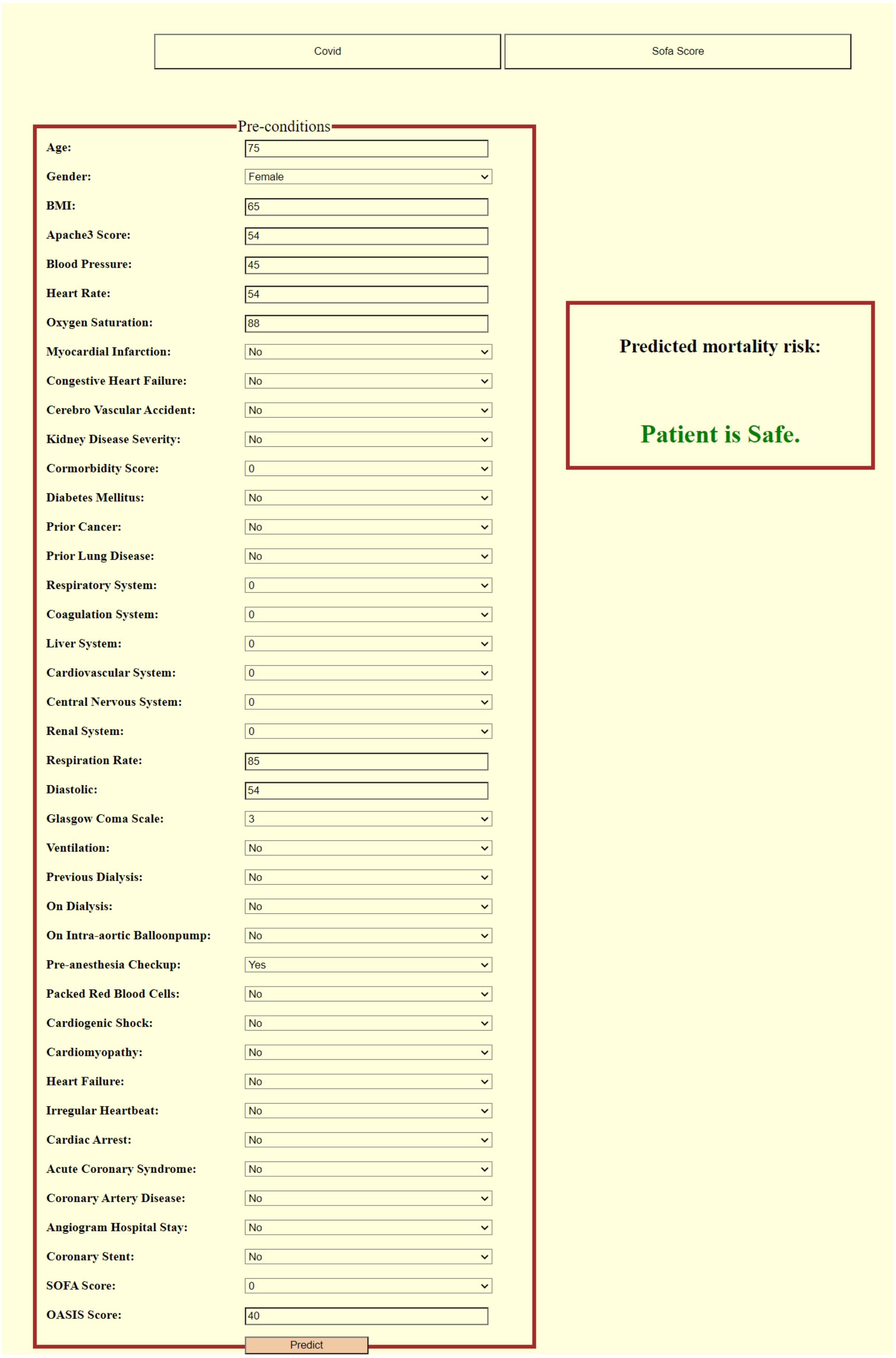
Pre-conditions

Age:	70
Gender:	Male
BMI:	39
Apache3 Score:	92
Blood Pressure:	87
Heart Rate:	96
Oxygen Saturation:	92
Myocardial Infarction:	No
Congestive Heart Failure:	No
Cerebro Vascular Accident:	No
Kidney Disease Severity:	No
Cormorbidity Score:	0
Diabetes Mellitus:	No
Prior Cancer:	No
Prior Lung Disease:	No
Respiratory System:	0
Coagulation System:	1
Liver System:	0
Cardiovascular System:	1
Central Nervous System:	2
Renal System:	1
Respiration Rate:	21
Diastolic:	54
Glasgow Coma Scale:	15
Ventilation:	Yes
Previous Dialysis:	No
On Dialysis:	Yes
On Intra-aortic Balloonpump:	No
Pre-anesthesia Checkup:	No
Packed Red Blood Cells:	Yes
Cardiogenic Shock:	No
Cardiomyopathy:	Yes
Heart Failure:	Yes
Irregular Heartbeat:	No
Cardiac Arrest:	No
Acute Coronary Syndrome:	No
Coronary Artery Disease:	No
Angiogram Hospital Stay:	No
Coronary Stent:	No
SOFA Score:	12
OASIS Score:	40

Predicted mortality risk:

Patient may die in 9 days and 20 hours.

Figure 48: CICU: Death Prediction



Pre-conditions

Age:	75
Gender:	Female
BMI:	65
Apache3 Score:	54
Blood Pressure:	45
Heart Rate:	54
Oxygen Saturation:	88
Myocardial Infarction:	No
Congestive Heart Failure:	No
Cerebro Vascular Accident:	No
Kidney Disease Severity:	No
Cormorbidity Score:	0
Diabetes Mellitus:	No
Prior Cancer:	No
Prior Lung Disease:	No
Respiratory System:	0
Coagulation System:	0
Liver System:	0
Cardiovascular System:	0
Central Nervous System:	0
Renal System:	0
Respiration Rate:	85
Diastolic:	54
Glasgow Coma Scale:	3
Ventilation:	No
Previous Dialysis:	No
On Dialysis:	No
On Intra-aortic Balloonpump:	No
Pre-anesthesia Checkup:	Yes
Packed Red Blood Cells:	No
Cardiogenic Shock:	No
Cardiomyopathy:	No
Heart Failure:	No
Irregular Heartbeat:	No
Cardiac Arrest:	No
Acute Coronary Syndrome:	No
Coronary Artery Disease:	No
Angiogram Hospital Stay:	No
Coronary Stent:	No
SOFA Score:	0
OASIS Score:	40

Predicted mortality risk:

Patient is Safe.

Figure 49: CICU: Survival Prediction

6 Conclusion

In this project, we presented Classification and Regression models, one to predict the patient's death status and another to predict the number of expected days. We have found Random Forest and Neural Network to be better than Logistic Regression, Decision Tree, Linear Regression, Gradient Boosting and Support Vector Machine algorithms.

Also performed feature selection on both the datasets and found some important features such as (age, sex, pneumonia, intubated, diabetes, hypertension and COPD) and (OASIS score, Apache3 score, SOFA score, Blood pressure, BMI and heart rate) for Covid-19 and CICU patient dataset, respectively.Random forest and Logistic regression gives theses best features as compare to Support vector machine classifier.

The data analyzed in this report revealed that the majority of affected patients are aged between 30 and 80 years. Moreover, the mortality rate for 60-80 years old patients is higher than other age group patients. Apart from that, the likeliness of death of patients with pneumonia is greater in comparison with COVID-19 patients with other co-morbidities.

7 Timeline

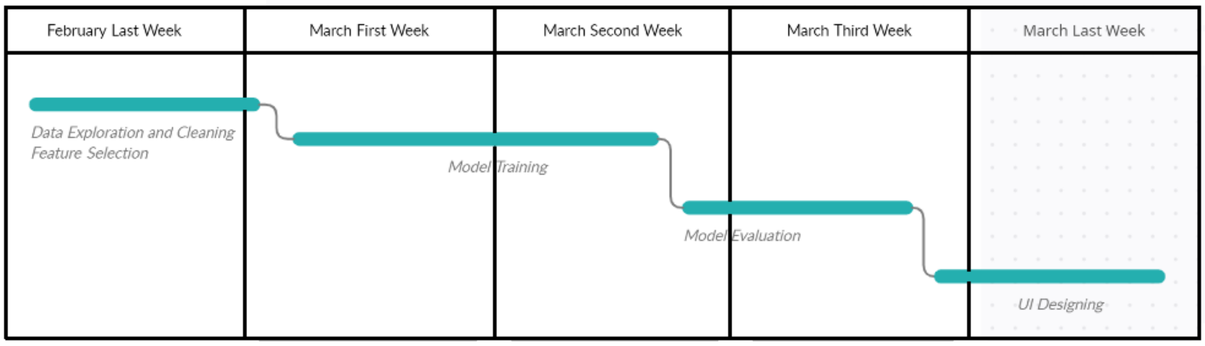


Figure 50: Time Line Chart

REFERENCES

[1] COVID-19 Coronavirus Pandemic. Retrieved from https://www.worldometers.info/coronavirus/.

[2] COVID-19 Image. Retrieved from https://phil.cdc.gov//PHIL_Images/23311/23311_lores.jpg

[3] Organ Dysfunction Image. Retrieved from https://industry.asianhhm.com/articles/images/target-organs-in-covid.jpg

[4] COVID-19 patient pre-condition dataset. Retrieved from https://www.kaggle.com/tanmoyx/covid19-patient-precondition-dataset.

[5] CICU(Cardiac Intensive Care unit) dataset. Retrieved from https://figshare.com/articles/dataset/Predictive_value_of_individual_Sequential_Organ_Failure_Assessment_sub-scores_for_mortality_in_the_cardiac_intensive_care_unit/8154995

[6] Clinical Feature selection. Retrieved from https://github.com/SBCNY/Clinical-predictors-of-COVID-19-mortality/blob/master/Clinical_predictor_notebook.ipynb

[7] Yadaw, A. S., Li, Y. C., Bose, S., Iyengar, R., Bunyavanich, S., Pandey, G. "Clinical features of COVID-19 mortality: development and validation of a clinical prediction model," The Lancet Digital Health, 2020, 2(10), 516-525.

[8] Blanca V. Et al. "RISK MARKERS BY SEX AND AGE GROUP FOR IN-HOSPITAL MORTALITY IN PATIENTS WITH STEMI OR NSTEMI: AN APPROACH BASED ON MACHINE LEARNING", 2021.

[9] Hiske O. Et al. "Interpretable Outcome Prediction with Sparse Bayesian Neural Networks in Intensive Care" , 2019.