Western Graduate&PostdoctoralStudies

Western University
Scholarship@Western

Electronic Thesis and Dissertation Repository

12-13-2023 4:00 PM

Learning Mortality Risk for COVID-19 Using Machine Learning and Statistical Methods

Shaoshi Zhang,

Supervisor: Grace Y. Yi, *The University of Western Ontario*

A thesis submitted in partial fulfillment of the requirements for the Master of Science degree in Computer Science

© Shaoshi Zhang 2023

Follow this and additional works at: https://ir.lib.uwo.ca/etd

 Part of the Artificial Intelligence and Robotics Commons, and the Data Science Commons

Recommended Citation

Zhang, Shaoshi, "Learning Mortality Risk for COVID-19 Using Machine Learning and Statistical Methods" (2023). *Electronic Thesis and Dissertation Repository*. 9871.
https://ir.lib.uwo.ca/etd/9871

This Dissertation/Thesis is brought to you for free and open access by Scholarship@Western. It has been accepted for inclusion in Electronic Thesis and Dissertation Repository by an authorized administrator of Scholarship@Western. For more information, please contact wlswadmin@uwo.ca.

Abstract

This research investigates the mortality risk of COVID-19 patients across different variant waves, using the data from Centers for Disease Control and Prevention (CDC) websites. By analyzing the available data, including patient medical records, vaccination rates, and hospital capacities, we aim to discern patterns and factors associated with COVID-19-related deaths.

To explore features linked to COVID-19 mortality, we employ different techniques such as Filter, Wrapper, and Embedded methods for feature selection. Furthermore, we apply various machine learning methods, including support vector machines, decision trees, random forests, logistic regression, K-nearest neighbours, naïve Bayes methods, and artificial neural networks, to uncover underlying trends and correlations within the data.

The study identifies nine crucial factors significantly impacting patient survival in the context of COVID-19. These encompass patient-level factors, including pre-existing medical conditions, acute respiratory distress syndrome status, pneumonia status, age group category, headache status, and shortness of breath (dyspnea) status, as well as the three factors showing the patient's status related to hospital aspects: hospitalization status, mechanical ventilation status, and intensive care unit admission status.

Utilizing these identified features, we further conduct a detailed statistical analysis using the logistic regression model to estimate the effects of these risk factors on COVID-19 mortality. The findings of this research indicate that the majority of those identified factors are statistically significant in influencing the likelihood of mortality.

However, exceptions and variations are observed across different waves of COVID-19 variants, underscoring the dynamic nature of the pandemic. This study contributes insights into understanding the evolving landscape of COVID-19 outcomes.

Keywords: Artificial neural networks, cost-sensitive classification, COVID-19, death rates, feature selection, hospital capacities, imbalanced classification, logistic regression, machine learning, medical conditions, mortality risk, statistical analysis, vaccination rates.

Summary for Lay Audience

This research investigates the mortality risk of COVID-19 patients across different variant waves, using the data from Centers for Disease Control and Prevention (CDC) websites. By analyzing the available data, including patient medical records, vaccination rates, and hospital capacities, we aim to discern patterns and factors associated with COVID-19-related deaths.

To explore features linked to COVID-19 mortality, we employ different techniques such as Filter, Wrapper, and Embedded methods for feature selection. Furthermore, we apply various machine learning methods, including support vector machines, decision trees, random forests, logistic regression, K-nearest neighbours, naïve Bayes methods, and artificial neural networks, to uncover underlying trends and correlations within the data.

The study identifies nine crucial factors significantly impacting patient survival in the context of COVID-19. These encompass patient-level factors, including pre-existing medical conditions, acute respiratory distress syndrome status, pneumonia status, age group category, headache status, and shortness of breath (dyspnea) status, as well as the three factors showing the patient's status related to hospital aspects: hospitalization status, mechanical ventilation status, and intensive care unit admission status.

Utilizing these identified features, we further conduct a detailed statistical analysis using the logistic regression model to estimate the effects of these risk factors on COVID-19 mortality. The findings of this research indicate that the majority of those identified factors are statistically significant in influencing the likelihood of mortality. However, exceptions and variations are observed across different waves of COVID-19 variants, underscoring the

dynamic nature of the pandemic. This study contributes insights into understanding the evolving landscape of COVID-19 outcomes.

Acknowledgments

I would like to express my heartfelt gratitude to my dedicated supervisor, Dr. Grace Y. Yi, for her invaluable guidance, unwavering support, and insightful feedback throughout this research journey. Her expertise and mentorship have been instrumental in shaping this work.

I am also deeply thankful to the Centers for Disease Control and Prevention (CDC) for granting access to the COVID-19 Case Data that formed the backbone of this study.

Contents

Abstract **i**

Summary for Lay Audience **iii**

Acknowledgments **v**

List of Figures **xi**

List of Tables **xiv**

1 Introduction **1**

2 Three CDC Datasets on COVID-19 **8**

2.1 COVID-19 Case Data 9

2.1.1 Data Source and Pre-Processing 9

2.1.2 Objective and Pre-Processed Data 12

2.2 Hospital Data 15

2.2.1 Data Source 15

2.2.2 Objective and Data Pre-Processing 16

2.3 Vaccination Data 17

2.3.1 Data Source 17
2.3.2 Objective and Data Pre-Processing 19

3 Data Preparation **22**
3.1 COVID-19 Case Data 23
3.1.1 Missing Values 23
3.1.2 Correcting and Converting Datatype 23
3.2 Hospital Data 24
3.2.1 Extracting Useful Subset 24
3.2.2 Converting 29
3.2.3 Plotting 30
3.3 Vaccination Data 31
3.3.1 Subsets of Death Information 31
3.3.2 Converting and Creating 32
3.4 Summary of Variables After Pre-processing 33
3.5 Data Integration and Division 35
3.5.1 Integration 35
3.5.2 Division 39

4 An Overview of Machine Learning Methods **41**
4.1 Feature Selection 41
4.1.1 Filter Method 42
4.1.2 Wrapper Method 44
4.1.3 Embedded Method 47
4.2 Machine Learning Models 48
4.2.1 Support Vector Machines 48

4.2.2 Decision Tree 50
4.2.3 Random Forest 50
4.2.4 Logistic Regression 51
4.2.5 K-Nearest Neighbour 51
4.2.6 Gaussian Naïve Bayes 52
4.2.7 Artificial Neural Networks 53
4.3 Cost-Sensitive Classification Models 56
4.3.1 Cost-Sensitive SVM 57
4.3.2 Cost-Sensitive Decision Tree 57
4.3.3 Cost-Sensitive Logistic Regression 58

5 Data Learning **59**
5.1 Feature Selection 59
5.1.1 Encoding Categorical Variables 60
5.1.2 Learning with the Filter Method 61
5.1.3 Learning with the Wrapper Method 64
5.1.4 Learning with the Embedded Method 67
5.1.5 Results with Intersection and Union 69
5.2 Data Balancing 72
5.2.1 Over-Sampling on Training Set 73
5.2.2 Under-Sampling on Testing Set 73
5.3 Learning with Machine Learning Models 74
5.3.1 Learning with the Support Vector Machine 75
5.3.2 Learning with the Decision Tree 77
5.3.3 Learning with the Random Forest 77
5.3.4 Learning with the Logistic Regression 78

5.3.5 Learning with the K-Nearest Neighbor 79
5.3.6 Learning with the Gaussian Naïve Bayes 80
5.3.7 Learning with the Artificial Neural Networks 81
5.4 Learning with the Cost-Sensitive Classification Models 86
5.4.1 Learning with the Cost-Sensitive SVM 87
5.4.2 Learning with the Cost-Sensitive Decision Tree 88
5.4.3 Learning with the Cost-Sensitive Logistic Regression . 89
5.5 Summary 90
5.6 Statistical Analysis 92

6 COVID-19 Variant of Concerns 94
6.1 Outbreak 95
6.2 AlphaBeta 96
6.3 Gamma 97
6.4 Delta 98
6.5 Omicron 99
6.6 Summary 100
6.7 Statistical Analysis 100

7 Summary and Discussion 104

References 108

Appendix 116
Data Pre-processing 116
Feature Selection 129
Data Analysis 133

VOC 147
Hyperparameter Tuning 150

List of Figures

3.1 *Example of pre-processed Hospital Data before split by Setting* 28

3.2 *Example of pre-processed Hospital Data* 29

3.3 *The percentage of non-COVID-19 patients admitted to selected hospitals during the COVID-19 waves* 31

3.4 *Age group pairing in COVID-19 case data, primary doses, and booster doses from the Vaccination Data* 36

3.5 *Summary of the integrated dataset. The first column records the index of each covariate, the second column lists the name of covariates, the third columns shows the number of non-null records for each covariate, and the last column indicates the corresponding datatype of each covariate* 37

5.1 *Association heat-map with Cramér's V values* 63

5.2 *Macro F1 Score(vertical axis) obtained by Sequential Forward Selection (with standard error shown in light blue shaded area)* 66

5.3 *The graph of feature importance, shown by the magnitude of the coefficient versus the feature's name* 68

5.4 *A sample MLP network architecture with nine input variables and two hidden layers* 83

List of Tables

2.1 *Summary of Pre-processed three datasets* 9

2.2 *Summary of COVID-19 Case Data* 10

2.3 *Summary of Hospital Data* 15

2.4 *Summary of Vaccination Data* 18

3.1 *Summary of Covariates After Pre-processing* 34

3.2 *Descriptive Statistics for Covariates displayed in Figure 3.5* . 38

3.3 *Variant of Concerns(VOCs)* 40

5.1 *Features selected by the Filter Method and their Cramér's V value with variable Y ("death_yn")* 64

5.2 *Features selected by the Wrapper Method* 66

5.3 *Features selected by the Embedded Method and their coefficients with variable Y ("death_yn")* 69

5.4 *Features selected by the Intersection* 70

5.5 *Features selected by the Union* 71

5.6 *The results obtained from applying the Support Vector Machine (SVM) to the integrated dataset in Section 3.5.1, with the use of one of the five best feature subsets obtained in Section 5.1* . 76

5.7 *The results obtained from applying the Decision Tree models to the integrated dataset in Section 3.5.1, with the use of one of the five best feature subsets obtained in Section 5.1* 77
5.8 *The results obtained from applying the Random Forest models to the integrated dataset in Section 3.5.1, with the use of one of the five best feature subsets obtained in Section 5.1* 78
5.9 *The results obtained from applying the Logistic Regression models to the integrated dataset in Section 3.5.1, with the use of one of the five best feature subsets obtained in Section 5.1* . . 79
5.10 *The results obtained from applying the K-Nearest Neighbour(KNN) models to the integrated dataset in Section 3.5.1, with the use of one of the five best feature subsets obtained in Section 5.1* . 80
5.11 *The results obtained from applying the Naïve Bayes models to the integrated dataset in Section 3.5.1, with the use of one of the five best feature subsets obtained in Section 5.1* 81
5.12 *The results obtained from applying the Multi-Layers perceptron (MLP) models to the integrated dataset in Section 3.5.1, with the use of one of the five best feature subsets obtained in Section 5.1* 84
5.13 *The results obtained from applying the Radial Basis Function Network (RBFN) models to the integrated dataset in Section 3.5.1, with the use of one of the five best feature subsets obtained in Section 5.1* 86

5.14 *The results obtained from applying the Cost-Sensitive SVM models to the integrated dataset in Section 3.5.1, with the use of one of the five best feature subsets obtained in Section 5.1* . 88

5.15 *The results obtained from applying the Cost-Sensitive Decision Tree models to the integrated dataset in Section 3.5.1, with the use of one of the five best feature subsets obtained in Section 5.1* 89

5.16 *The results obtained from applying the Cost-Sensitive logistic regression models to the integrated dataset in Section 3.5.1, with the use of one of the five best feature subsets obtained in Section 5.1* 90

5.17 *Statistical analysis of the integrated dataset with the intersection feature subset under the logistic regression model* 93

6.1 *The results obtained from applying various models to the Outbreak dataset in Section 3.5.2 using the intersection feature subset obtained in Section 5.1.5* 96

6.2 *The results obtained from applying various models to the AlphaBeta dataset in Section 3.5.2 using the intersection feature subset obtained in Section 5.1.5* 97

6.3 *The results obtained from applying various models to the Gamma dataset in Section 3.5.2 using the intersection feature subset obtained in Section 5.1.5* 98

6.4 *The results obtained from applying various models to the Delta dataset in Section 3.5.2 using the intersection feature subset obtained in Section 5.1.5* 99

6.5 *The results obtained from applying various models to the Omicron dataset in Section 3.5.2 using the intersection feature subset obtained in Section 5.1.5* 100
6.6 *Statistical analysis of each VOC dataset with the intersection feature subset under the logistic regression model* 103

A1 *Best Hyperparameters for Some Classifiers Used in the Study* . 150

Chapter 1

Introduction

Since the discovery of the novel coronavirus (COVID-19) in Wuhan in December 2019, research on COVID-19 has received extensive attention, and scientists from various disciplines have gained a deeper understanding of this virus due to advancements in technology and the availability of increasingly collected data. Despite the emergence of new variants of concern, treatment strategies and vaccinations have been developed to treat COVID-19 patients and prevent the spread of the virus among humans.

In response to the pandemic, many studies have been conducted to provide valuable insights into the clinical characteristics and outcomes of COVID-19 patients and to develop effective tools for managing the pandemic. To name a few, Bertsimas et al. (2020) developed a COVID-19 mortality risk assessment tool using patient-level medical data from multiple international centers. The study analyzed data from over 8,000 COVID-19 patients across 12 countries, identifying key factors associated with COVID-19 mortality risk and developing a predictive model to study the mortality risk in COVID-19

patients. Considering that individuals with pre-existing conditions, such as multiple sclerosis (MS), may be at a higher risk of adverse outcomes from COVID-19, Louapre et al. (2020) conducted a retrospective study of 347 COVID-19 patients with MS to explore the clinical characteristics and outcomes of these patients. They investigated the frequency and severity of COVID-19 symptoms and risk factors associated with severe outcomes, shedding light on the unique challenges faced by COVID-19 patients with MS and providing evidence-based recommendations for managing these patients during the pandemic. These two studies took advantages of patient-level medical data and provided valuable insights for healthcare providers in making informed decisions regarding patient care and resources allocation.

To control the spread of the virus during the COVID-19 pandemic, vaccination has emerged as a critical tool and the COVID-19 vaccination program has been one of the largest public health campaigns in history, with the aim to vaccinate the majority of the population in the world. Research has been conducted to address the impact and effectiveness of vaccinations. For example, Moghadas et al. (2021) conducted a study to investigate the impact of 2-dose COVID-19 vaccination on outbreaks in the United States in 2021. By analyzing vaccination rates and COVID-19 case data from different regions in the United States, the study explored the effectiveness of vaccination in reducing COVID-19-related hospitalization and mortality rates and provided insights into the potential benefits of vaccination programs in mitigating the spread of the virus. Also, McNamara et al. (2022) conducted an ecological analysis of national surveillance data in 2022 to estimate the early impact of the US COVID-19 vaccination program on COVID-19 cases, emergency

department visits, hospital admissions, and deaths among adults aged 65 years and older. Both studies assessed the effectiveness of the vaccination program in reducing COVID-19-related morbidity and mortality among the population and provided insights into the early impact of the COVID-19 vaccination program.

The COVID-19 pandemic has posed unprecedented challenges to health-care systems worldwide. The emergency medical services (EMS) system has been particularly affected due to the fast virus transmission. In this context, Lerner, Newgard, and Mann (2020) conducted a study to assess the effect of the pandemic on the US EMS system. The study analyzed data from a national EMS database, identifying changes in EMS activations, patient acuity, and transport patterns during the pandemic. They found that early in the COVID-19 outbreak there was a significant decrease in the number of EMS responses across the United States, and simultaneously, the rate of EMS-attended death doubled, while the rate of injuries decreased. They highlighted the need for ongoing efforts to ensure the safety and well-being of EMS providers and patients during pandemics and other public health emergencies.

Despite extensive research in the literature, there does appears to be a lack of studies that comprehensively integrate the COVID-19 patient-level medical data, vaccination status, and hospital capacity together to examine risk factors concerning the COVID-19 mortality. This research aims to study the mortality risk of COVID-19 patients in different variant waves by leveraging patient-level medical data, vaccination rates, and hospital capacities.

Given that the United States has the largest infected population and

death cases in the world according to World Health Organization (2023a), and that the Centers for Disease Control and Prevention (CDC) has a great amount of dependable and public accessible datasets at the patient-level, this research primarily utilizes three CDC-released data sets to examine risk factors for COVID-19 mortality. These data sets include: (1) COVID-19 Hospital Data from the National Hospital Care Survey, (2) Rates of COVID-19 Cases or Deaths by Age Group and Vaccination Status (and Booster Dose), and (3) COVID-19 Case Surveillance Restricted Access Detailed Data.

In handling missing values within the COVID-19 Case Data, our approach involves the removal of these values, which is consistent with common practice in handling missing data (e.g., Roth, 1994; Bennett, 2001; Scheffer, 2002; Newman, 2014). The remaining individuals still contain a sizable post-removal dataset (242,772 rows/cases) relative to the 24 features, allowing us to effectively examine the relationship among the associated variables. While there is a belief that the prioritized symptom indicators in the CDC's data quality assurance may lead to dropout or skipped questions during form filling, no official explanations for the missing data are available for us to perceive what the missing data mechanisms are here. Our approach is basically called the *complete data analysis*, whose validity is ensured if missing data possess the missing completely at random (MACR) mechanism, (e.g., Yi (2017, Section 5.5.1) and Little and Rubin (2019)). It is recognized that biased results are generally expected if MCAR is not feasible, because the retained subjects do not represent a random sample of the underlying population.

Recognizing the absence of a one-size-fits-all solution for classification

tasks, we propose to first employ a range of methods and algorithms to examine risk factors for COVID-19 mortality from different perspectives, then we compare the commonality and discrepancy in the results to enhance the understanding of the underlying truth. The methods include feature selection using Filter, Wrapper, and Embedded approaches, complemented by data balancing techniques. Our analyses employ a combination of machine learning models, such as cost-sensitive classification models and artificial neural networks, and statistical inference based on logistic regression model.

Our study systematically evaluates various methods to compare their effectiveness. In terms of feature selection using the three methods (Filter, Wrapper, and Embedded approaches), our findings emphasize the effectiveness of selecting common features, i.e., those in the intersection set, revealed by all the three methods. Acknowledging the potential loss of important features by using this intersection set, we also extend our analysis to include any feature identified by each of the three methods to form a union set of those features.

Consistently, our results highlight the effectiveness of multiple models based on nine key features: age group category, hospitalization status, intensive care unit admission status, pneumonia status, acute respiratory distress syndrome status, mechanical ventilation status, headache status, shortness of breath (dyspnea) status, and pre-existing medical conditions. These variables are shortened as follows in the subsequent sections, which are further studied for their effects on COVID-19 mortality through statistical analysis based on the logistic regression model:

- "age_group" $(x_3^{(1)})$

- "hosp_yn" ($x_5^{(1)}$)
- "icu_yn" ($x_6^{(1)}$)
- "pna_yn" ($x_8^{(1)}$)
- "acuterespdistress_yn" ($x_{10}^{(1)}$)
- "mechvent_yn" ($x_{11}^{(1)}$)
- "headache_yn" ($x_{21}^{(1)}$)
- "sob_yn" ($x_{19}^{(1)}$)
- "medcond_yn" ($x_{24}^{(1)}$)

While our analyses are directed to analyze COVID-19 data, our strategy of forming the intersection and union sets of features, by examining different feature selection methods, can apply to other contexts as well. As there is no one-size-fit-all method, our study underscores the importance of delving into and contrasting various methods within the wider realm of feature selection and the selection of machine learning models. Because of the lack of a universally superior algorithm, examining and analyzing data using various approach helps reveal a comprehensive picture of the relationship among variables.

The remainder of this essay is organized as follows. In Chapter 2, we present an introduction to the three datasets used in this essay, along with their respective objectives. In Chapter 3, we present the detail for the data preparation process, including handling missing values, converting datatypes, and creating new covariates by reformatting initial variables. Chapter 4

elaborates on the methods to be employed to analyze the data, including feature selection techniques, data balancing strategies, machine learning models, cost-sensitive models, and artificial neural networks. The results of the models and the identification of the best feature subset are reported in Chapter 5. Lastly, Chapter 6 conducts an in-depth analysis of five distinct COVID-19 variants, utilizing the best feature subset and the optimal model identified in Chapter 5. Supplementary materials are included in appendices.

Chapter 2

Three CDC Datasets on COVID-19

In this chapter, we present detailed descriptions for the three datasets to be analyzed in subsequent chapters. The first dataset, called COVID-19 Case Data, contains patient-level measurements per case, including the information on demographics, geography, exposure history, disease existence indicators, and so on. The second dataset, called Vaccination Data, records the death/confirm rates for different age groups and vaccination types. The last dataset, called Hospital Data, indicates the running pressure of hospitals with the percentage of non-COVID-19 patients accepted weekly.

With the objective to explore the mortality risk of COVID-19 patients based on patient-level medical information, different vaccination rates, and hospital running capacities, we merge these three independent datasets into a single one by first pre-processing them individually, and then merging them by either date or age groups. The detailed steps of pre-processing will be

provided in Chapter 3. Here we present in Table 2.1 a summary of each dataset after being pre-processed, where for $j = \{1, 2, 3\}$, n^j represents the size of the jth dataset, p^j stands for the number of covariates obtained after the pre-processing steps from jth dataset, and the number of columns records the number of attributes before pre-processing for one of these datasets.

Table 2.1: *Summary of Pre-processed three datasets*

	COVID-19 Case Data	Hospital Data	Vaccination Data
Dataset collection period	2020-01-01 to 2022-04-04	2020-03-18 to 2021-11-30	2021-04-04 to 2022-03-19
Dataset size	$n^1 = 69,664,983$	$n^2 = 19447$	$n^3 = 997$
Number of columns before pre-processing	32	10	22
Number of covariates after pre-processing	$p^1 = 24$	$p^2 = 2$	$p^3 = 1$

Below we describe the associated variables in the same way as stated on the CDC website, where we let Y denote the common output variable for the three datasets, and let $x_j^{(k)}$ represent the jth covariate in the kth dataset for $k = 1, 2, 3$ and $j = 1, \ldots, p^k$, with p^k being the number of covariates in the kth dataset.

2.1 COVID-19 Case Data

2.1.1 Data Source and Pre-Processing

The COVID-19 Case Data, accessed through CDC with permission needed for the restricted data at `https://data.cdc.gov/Case-Surveillance/COVID-19-Case-Surveillance-Restricted-Access-Detai/mbd7-r32t`, are collected from all COVID-19 cases reported to CDC between 2020-01-01 and 2022-04-04. Table 2.2 gives a summary of the data where there are 32 columns

Table 2.2: *Summary of COVID-19 Case Data*

The data collection period	2020-01-01 to 2022-04-04
The dataset size	$n^1 = 69,664,983$
The number of columns	32
The number of covariates after pre-processing and the symbols of covariates in parentheses	$p^1 = 24$ ($\{x_1^{(1)}, \ldots, x_{24}^{(1)}\}$)
The name of outcome	Y: "death_yn" (binary variable of death status)
Source website	`https://data.cdc.gov/Case-Surveillance/COVID-19-Case-Surveillance-Restricted-Access-Detai/mbd7-r32t`

(variables) and 69,664,983 rows (cases/patients) in total, with Y representing the death indicator ("death yn"). The data include the information about demographics, geography, exposure history, disease existence indicators, underlying medical conditions indicator, and the final outcome (alive or death) for each case.

The original dataset contains some similar or repeating columns for dates and residence areas, such as "cdc_report_dt" versus "cdc_case_earliest_date", "res_county" versus "county_fips_code", and so on. Also, since the COVID-19 case surveillance data are collected by jurisdictions yet these jurisdictions may or may not share the data with CDC, the quality of input data heavily depends on the data collection performance of each jurisdiction; or uncertain measurements are present in the dataset a considerable proportion of missing values. Therefore, it is necessary to pre-process the dataset to get rid of redundant information or columns with missing or uncertain values before analyzing cleaned data.

The official CDC website mentioned in the previous paragraphs, the CDC employs several measures to maintain data quality in this dataset, as outlined on their website. For questions with three possible answers ("Yes", "No", and "Unknown"), any blank responses are reclassified as "Missing". Examinations of the accuracy are implemented for the date associated with the data. Most crucially, the CDC prioritizes maintaining data quality, especially concerning information about symptom indicators, race, ethnicity, and healthcare worker status, throughout the data assurance procedures.

A filtering process is applied to the "current_status" column which contains two possible statuses of each study subject: "Laboratory-confirmed case" or "Probable case." The filtering process retains the confirmed cases without including those probable cases for further analyses. Moreover, the following 6 columns are removed due to the presence of missing values or low priority in the data assurance control:

1. cdc_report_dt: Date for the case that was first reported to the CDC;
2. onset_dt: Date of symptom onset;
3. pos_spec_dt: Date of first positive specimen collection;
4. county_fips_code: five-digit integer which uniquely identifies geographic areas;
5. res_county: County of residence;
6. res_state: State of residence.

Then we obtain a pre-processed dataset that is taken as the primary dataset. We will use this dataset to track the detailed patient-level medical

information for subsequent analyses. The death indicator Y ("death_yn") of the laboratory-confirmed case will be taken as the outcome or the response variable.

2.1.2 Objective and Pre-Processed Data

With the pre-processed dataset, we are interested in identifying those demographic variables and medical condition indicators of COVID-19 patients that are important for predicting the mortality risk at the patient-level; such a study helps us understand what kinds of patients are at a higher risk and need more medical attention.

For the pre-processed COVID-19 Case Data, we take the output variable Y: recorded as "death_yn", to be the binary indicator for each case to be dead or not, with $Y = 0$ representing "not dead (No)" and $Y = 1$ representing "dead (Yes)". The $p^1 = 24$ covariates in this COVID-19 Case Data dataset are described as follows:

1. cdc_case_earliest_dt ($x_1^{(1)}$): "The earlier of the clinical date (date related to the illness or specimen collection) or the date received by CDC"
2. sex ($x_2^{(1)}$): "male; female; other; unknown; missing; NA"
3. age_group ($x_3^{(1)}$): "0-9; 10-19; 20-29; 30-39; 40-49; 50-59; 60-69; 70-79; 80+"
4. race_ethnicity_combined ($x_4^{(1)}$): "American Indian/Alaska native; non-Hispanic Asian; non-Hispanic black; non-Hispanic multiple/other; non-Hispanic native Hawaiian/other pacific islander; non-Hispanic white; non-Hispanic Hispanic/Latino; and unknown"

5. hosp_yn ($x_5^{(1)}$): "Was the patient hospitalized?"
6. icu_yn ($x_6^{(1)}$): "Was the patient admitted to an intensive care unit(ICU)?"
7. hc_work_yn ($x_7^{(1)}$): "Is the patient a health care worker in the United States?"
8. pna_yn ($x_8^{(1)}$): "Did the patient develop pneumonia?"
9. abxchest_yn ($x_9^{(1)}$): "Did the patient have an abnormal chest X-ray?"
10. acuterespdistress_yn ($x_{10}^{(1)}$): "Did the patient have acute respiratory distress syndrome?"
11. mechvent_yn ($x_{11}^{(1)}$): "Did the patient receive mechanical ventilation (MV) or intubation?"
12. fever_yn ($x_{12}^{(1)}$): "Did the patient have fever $>$ 100.4F(38C)?"
13. sfever_yn ($x_{13}^{(1)}$): "Did the patient have subjective fever (felt feverish)?"
14. chills_yn ($x_{14}^{(1)}$): "Did the patient have chills?"
15. myalgia_yn ($x_{15}^{(1)}$): "Did the patient have muscle aches (myalgia)?"
16. runnose_yn ($x_{16}^{(1)}$): "Did the patient have runny nose (rhinorrhea)?"
17. sthroat_yn ($x_{17}^{(1)}$): "Did the patient have sore throat?"
18. cough_yn ($x_{18}^{(1)}$): "Did the patient have cough (new onset or worsening of chronic cough)?"
19. sob_yn ($x_{19}^{(1)}$): "Did the patient have shortness of breath (dyspnea)?"

20. nauseavomit_yn ($x_{20}^{(1)}$): "Did the patient have nausea or vomiting?"

21. headache_yn ($x_{21}^{(1)}$): "Did the patient have headache ?"

22. abdom_yn ($x_{22}^{(1)}$): "Did the patient have abdominal pain?"

23. diarrhea_yn ($x_{23}^{(1)}$): "Did the patient have diarrhea ($\geq$ 3 loose/looser than normal stools/24hr period)?"

24. medcond_yn ($x_{24}^{(1)}$): "Did the patient have pre-existing medical conditions?"

Indicator variables $\{x_5^{(1)}, \ldots, x_{24}^{(1)}\}$ are denoted by the suffix "_yn" in their column names, representing the answer to the corresponding question to be either "yes" or "no", coded as "1" or "0". These variables take only one of four values: *yes, no, unknown, missing*, which are regulated by the CDC.

Except for cdc_case_earliest_dt ($x_1^{(1)}$), all covariates are categorical variables that are to be encoded as numerical values by using **OridinalEncoder** from **Scikit-learn**(Pedregosa et al., 2011). Scikit-learn is an open-source machine learning library in Python, which provides various machine learning algorithms, including classification, regression, and clustering. It also supports data pre-processing, model evaluation, and feature selection. Categorical encoding enables the representation of categorical variables in a numerical format, which is often needed for the implementation of many machine learning algorithms(Potdar, Pardawala, and Pai, 2017).

Table 2.3: *Summary of Hospital Data*

The data collection period	2020-03-18 to 2021-11-30
The dataset size	$n^2 = 19447$
The number of columns before pre-processing	10
The number of covariates after pre-processing and the symbols of covariates in parentheses	$p^2 = 2$ ("Non_COVID_Percent ED"($x_1^{(2)}$) & "Non_COVID_Percent_IP"($x_2^{(2)}$))
Source website	`https://data.cdc.gov/NCHS/COVID-19-Hospital-Data-from-the-National-Hospital-/q3t8-zr7t`

2.2 Hospital Data

2.2.1 Data Source

The Hospital Data, publicly available at `https://data.cdc.gov/NCHS/COVID-19-Hospital-Data-from-the-National-Hospital-/q3t8-zr7t`, are collected by the National Hospital Care Survey (NHCS), which contain inpatient department and emergency department measurements from 40 hospitals for the period from 2020-03-18 to 2021-11-30.

Table 2.3 gives a summary of the data, which contain 10 columns recording various indicators such as COVID-19 patient percentage, length of inpatient stay, and so on. According to the CDC official website, this dataset may provide insight into understanding the pressure brought by COVID-19 on the entire U.S. healthcare system and the impact on various types of hospitals, even though it is not necessarily nationally representative.

2.2.2 Objective and Data Pre-Processing

COVID-19 is an unprecedented challenge to healthcare systems in the world. It is critical to monitor the running pressure of hospitals in waves of different variants, and moreover, it is important to understand the induced effects on the COVID-19 death rate. As a large number of COVID-19 patients rush to hospitals, unprecedented pressure is placed on the healthcare system, which will impact both COVID and Non-COVID patients.

With the Hospital Data, we are interested in examining how the population COVID-19 death rate would be associated with by the hospital running pressure.

However, the original dataset does not contain measurements of a variable to represent hospital pressure. As a remedy, we use the information contained in the following six columns to generate two new covariates to indicate the hospital running pressure brought by COVID-19: The detailed description of this pre-processing will be presented in Section 3.2.1.

- Start_Time: Week starting date
- Setting: ED = Emergency Department, IP = Inpatient
- Indicator: Confirmed COVID-19, non-COVID-19, Suspected COVID-19, Total screenings, death, etc.
- Group: Subgroups such as age groups and sex.
- Measure: Number of encounters, average length of stay, percent, etc; measurement for each indicated patient type.

- Value: the value for corresponding measurement

The resulting two new covariates are named as "Non_COVID_Percent_ED"($x_1^{(2)}$) and "Non_COVID_Percent_IP"($x_2^{(2)}$), with the following definition:

1. Non_COVID_Percent_ED ($x_1^{(2)}$): the percentage of non-COVID-19 patients in the Emergency Department;

2. Non_COVID_Percent_IP ($x_2^{(2)}$): the percentage of non-COVID-19 patients in the Inpatient Department.

In the data analyses to be reported in Chapter 5, these new covariates $x_1^{(2)}$ and $x_2^{(2)}$ in the Hospital Data will be used as additional features to $\{x_1^{(1)}, \ldots, x_{24}^{(1)}\}$ in the COVID-19 Case Data described in Section 2.1, for which a outcome variable Y, defined in Section 2.1.2, is shared by the both data sets.

2.3 Vaccination Data

2.3.1 Data Source

The Vaccination Data, publicly available at (1) `https://data.cdc.gov/Public-Health-Surveillance/Rates-of-COVID-19-Cases-or-Deaths-by-Age-Group-and/3rge-nu2a`, and (2) `https://data.cdc.gov/Public-Health-Surveillance/Rates-of-COVID-19-Cases-or-Deaths-by-Age-Group-and/d6p8-wqjm`, are collected to monitor rates of COVID-19 cases and deaths by vaccination status and age groups. They contains two datasets: (1) Rates of COVID-19 cases or deaths by age group and vaccination status, and (2)

Table 2.4: *Summary of Vaccination Data*

The data collection period	2021-04-04 to 2022-03-19 (Dataset 1), and 2021-09-19 to 2022-03-19 (Dataset 2)
The dataset size	$n_1^3 = 997$ (Dataset 1) $n_2^3 = 364$ (Dataset 2)
The number of columns	16 (Dataset 1), and 22 (Dataset 2)
The number of covariates after pre-processing and the symbols of covariates in parentheses	$p^3 = 1$ ("death_rate"$x_1^{(3)}$)
Source website	`https://data.cdc.gov/Public-Health-Surveillance/Rates-of-COVID-19-Cases-or-Deaths-by-Age-Group-and/3rge-nu2a` `https://data.cdc.gov/Public-Health-Surveillance/Rates-of-COVID-19-Cases-or-Deaths-by-Age-Group-and/d6p8-wqjm`

Rates of COVID-19 cases or deaths by age group and vaccination status and booster dose.

Table 2.4 gives a summary of the data where the first dataset spans from 2021-04-04 to 2022-03-19, including 16 columns and 997 rows in terms of weeks and age groups; the second dataset enriches the previous dataset with Booster Dose information from 2021-09-19 by adding 6 more columns to the original dataset.

These datasets record dates, the confirmed cases, and deaths for different age groups with different vaccination status during the given time frame. Similar to the COVID-19 Case Data, the vaccination information is provided on jurisdictions-basis and updated weekly. According to the CDC website, the participating jurisdictions come from ten health and human services regions in the U.S., covering about 71% of the total population.

2.3.2 Objective and Data Pre-Processing

Since the vaccination is critical to prevent or reduce infection and death caused by COVID-19, this dataset may be used to provide meaningful insight into the effectiveness of vaccination. With this dataset, we are interested in examining the strength of protection brought by vaccinations for different age groups. Therefore, the corresponding death rate for each age group will be calculated as the population-weighted sum of the death rates of different vaccination statuses.

Because the Vaccination Data do not include specific columns indicating the death rate for different age groups and vaccination status directly, we utilize the following 9 available columns to derive a new covariate, called

"death_rate"($x_1^{(3)}$), with the detailed process of generating this covariate deferred to Section 3.3.2:

- MMWR week/mmwr_week: MMWR epidemiological year and week (YYYYWW format; e.g. 202101)
- outcome: COVID-19 case or death
- age group: "12-17; 18-29; 30-49; 50-64; 65-79; 80+" and "12-17; 18-49; 50-64; 65+" respectively
- fully vaccinated population: cumulative weekly count of the population vaccinated with at least a primary series
- unvaccinated population: cumulative weekly estimated count of the unvaccinated population
- boosted_population: cumulative weekly count of the population vaccinated with a primary series and booster dose
- crude vax IR: unadjusted incidence rate of the corresponding outcome among the population vaccinated with at least a primary series (per 100,000 population)
- crude unvax IR: unadjusted incidence rate of the corresponding outcome among the unvaccinated population (per 100,000 population)
- crude_booster_ir: unadjusted incidence rate of the corresponding outcome among the population vaccinated with a primary series and booster dose (per 100,000 population)

The new covariate, "death_rate"($x_1^{(3)}$), is defined as the population-weighted sum of the death rate that is calculated for each vaccination status within each age group. It will be used as an additional feature to $\{x_1^{(1)}, \ldots, x_{24}^{(1)}, x_1^{(2)}, x_2^{(2)}\}$, a combined features of the COVID-19 Case Data and the Hospital Data described in Section 2.2.2, for which the common outcome variable Y is shared.

Chapter 3

Data Preparation

To leverage the datasets described in Chapter 2 to examine the mortality risk of COVID-19, we consolidate those three datasets into a unified dataset chronologically. This data aggregation allows us to have a comprehensive and informative dataset for which each patient contains not only his/her medical details but also additional external factors, such as vaccination status and hospital pressures upon reception.

Even though these 3 datasets described in Chapter 2 are related, each of them covers different aspects of COVID-19 factors so they need to be cleaned and manipulated separately before being aggregated. In this Chapter, we further pre-process those 3 datasets individually and then merge them together as a single dataset to be analyzed.

3.1 COVID-19 Case Data

3.1.1 Missing Values

The COVID-19 Case Data dataset described in Section 2.1 has a huge file size of about 18GB in total. More than 99.995% of the subjects have at least one missing value in all 32 columns. As mentioned in Section 2.1.1, the first step is to only keep the confirmed cases and remove the probable cases and those entries with missing values. Specifically, for those columns related to $\{x_1^{(1)} \ldots x_{24}^{(1)}\}$, we remove any rows containing entries with *missing*, or *unknown*. The resulting subset contains only 242,744 rows and the file size is significantly reduced to 30MB. This means that the COVID-19 Case Data dataset to be analyzed includes 242,774 confirmed patients with complete COVID-19 records.

The removal of missing values, allows us to employ available methods in the literature, which commonly require complete observations for each study subject. This treatment of missing observations is called the complete data analysis, a widely used approach in applications. Its validity is ensured if data are missing completely at random (MCAR). When data are missing at random (MAR) or missing not at random (MNAR), biased results are generally expected (Little and Rubin, 2019).

3.1.2 Correcting and Converting Datatype

Correcting: This COVID-19 Case Data dataset is maintained regularly and published after careful inspection, and most of the columns are binary indicator columns (Yes/No) or categorical answers (e.g., age group, race,

state, county, etc.). There are no aberrant or unreasonable data inputs and no correction seems needed.

Converting: For the selected 24 covariates $\{x_1^{(1)} \ ... \ x_{24}^{(1)}\}$, only date variable "cdc_case_earliest_dt"$(x_1^{(1)})$ is needed to change the datatype. The "cdc_case_earliest_dt"$(x_1^{(1)})$ column is converted from "object" to "datetime64" datatype to allow more operations in further analysis such as sorting the dataframe by date, joining two or more dataframes by the date and so on.

3.2 Hospital Data

3.2.1 Extracting Useful Subset

The Hospital Data described in Section 2.2 contain a number of measurements that are not useful in this study. It is thereby necessary to pre-process the original data to obtain a relevant subset to conduct analyses. In particular, the measurements in the following six columns are pre-processed as follows:

- Start_Time: Week starting date
  - The column under the heading "Start_Time" is converted to the "datetime64" data type, a data type in the **NumPy** library that is commonly used for numerical computations in Python (Harris et al., 2020). It serves as a key to combine the Hospital Data with the COVID-19 Case Data dataset on $x_1^{(1)}$("cdc_case_earliest_dt"). Once integrated, this column will be dropped, and further details regarding this process will be provided in Section 3.5.1.

- Indicator: the single column with heading "Indicator" records the following 17 types of indicators to describe the status of the patients: *Confirmed COVID-19, Non-COVID-19, Suspected COVID-19, Confirmed Screenings, Negative Screenings, Total Screenings, Confirmed COVID-19 with pneumonia, Confirmed COVID-19 with acute respiratory failure, Confirmed COVID-19 with LRI, Confirmed COVID-19 with ARDS, Confirmed COVID-19 with bronchitis, Confirmed COVID-19 with any respiratory illness, Confirmed COVID-19 Deaths, Intubation or Ventilator Use, Deaths of confirmed cases with intubation or ventilator use, Deaths of confirmed cases without intubation or ventilator use, NULL.*

  - Among these 17 variables, there are three variables about screening, which are "Confirmed Screenings", "Negative Screenings", and "Total Screenings". Peculiarly, all these variables record the exactly same 1740 entries, which do not make sense. As a result, all entries associated with these three screening-related indicators have been removed to rectify this error. The *NULL* entries are removed as well. Thus, all the remaining 13 indicators are associated with confirmed COVID-19 cases, except "Non-COVID-19" and "Suspected COVID-19".
  - Among the 17 indicators under consideration, 16 are associated with confirmed or suspected COVID-19 cases, indicating varying levels of additional hospital resource required. In contrast, one indicator, labeled as "Non-COVID-19," represents cases that do not need any additional healthcare resources. Therefore, by examining

the proportion of "Non-COVID-19" patients within the dataset, we can gain certain insights into the overall operational burden on hospitals. To specifically reflect the hospital pressure brought by COVID-19, we keep only the rows where the "Indicator" column has the value "Non-COVID-19". Subsequently, we remove all other rows, as they pertain to confirmed or suspected COVID-19 cases that require further medical attention. This approach offers us a way to understand overall hospital running pressure by separating out non-COVID-19 cases who do not strain hospital resources from others.

- Group: Each entry in the "Group" column contains one of the 8 provided values: *Age, Sex, Total, Urban-rural, Discharge and Intubation or Ventilator Use Status, Discharge Status, Intubation or Ventilator Use Status, NULL*
  - After keeping only the "Non-COVID-19" rows from the previous step, only three values for the "Group" column remain: *Age, Sex*, and *Total*.
  - Since the age or sex of patients are not regarded as direct factors to increase the running pressure of hospitals, only the rows with "Total" are kept in the "Group" column.
- Measure:
  - The column with heading "Measure" records exactly one of three words: *Number of encounters, Average length of stay(days), Per-*

cent. The corresponding values are stored in a separate column named "Values" and will be discussed later.

  - However, neither the *average length of stay* nor the *number of encounters* has spanned the entire time frame (Mar 2020 - Nov 2021), only *percent* has sufficiently many instances and it is therefore kept to provide insight into the hospital running pressure. After only keeping the selected rows as described in the previous steps, the corresponding values for this *Percent* column now indicate the percent of non-COVID-19 patients admitted to the hospital regardless their age or sex. After eliminating other measurements with numerous missing values, only the rows with the word "*percent*" remains.

- Value: the value for corresponding measurement

  - After selecting only the *Percent* measurement and completing previous preprocessing steps, each entry in this column represents the percentages of non-COVID-19 patients admitted to the hospital in a given week, as shown in Figure 3.1.

- Setting: *ED* or *IP*

  - This column records a binary indicator that each entry must be either *ED* or *IP*, where *ED* represents the emergency department, and *IP* represents the inpatient department.

  - To have a better picture of running pressure, defined as the percent of non-COVID-19 patients, and to monitor the running pres-

	A	B	C	D	E	F
1	Setting	Indicator	Group	Start_Time	Value	Measure
2049	ED	Non-COVID-19	Total	3/18/2020	97.7	Percent
2050	ED	Non-COVID-19	Total	3/25/2020	95.4	Percent
2051	ED	Non-COVID-19	Total	4/1/2020	91.8	Percent
2052	ED	Non-COVID-19	Total	4/8/2020	91.7	Percent
2053	ED	Non-COVID-19	Total	4/15/2020	92	Percent
2054	ED	Non-COVID-19	Total	4/22/2020	92.9	Percent
2055	ED	Non-COVID-19	Total	4/29/2020	93.7	Percent
2056	ED	Non-COVID-19	Total	5/6/2020	93	Percent
2057	ED	Non-COVID-19	Total	5/13/2020	93.2	Percent
2058	ED	Non-COVID-19	Total	5/20/2020	93.5	Percent
2059	ED	Non-COVID-19	Total	5/27/2020	94	Percent
2060	ED	Non-COVID-19	Total	6/3/2020	94.5	Percent
2061	ED	Non-COVID-19	Total	6/10/2020	94.4	Percent
2062	ED	Non-COVID-19	Total	6/17/2020	93.4	Percent
2063	ED	Non-COVID-19	Total	6/24/2020	91.4	Percent
2064	ED	Non-COVID-19	Total	7/1/2020	91	Percent
2065	ED	Non-COVID-19	Total	7/8/2020	90.4	Percent
2066	ED	Non-COVID-19	Total	7/15/2020	90.6	Percent
2067	ED	Non-COVID-19	Total	7/22/2020	90.9	Percent
2068	ED	Non-COVID-19	Total	7/29/2020	90.7	Percent
2069	ED	Non-COVID-19	Total	8/5/2020	90.3	Percent
2070	ED	Non-COVID-19	Total	8/12/2020	89.9	Percent
2071	ED	Non-COVID-19	Total	8/19/2020	89	Percent
2072	ED	Non-COVID-19	Total	8/26/2020	88.8	Percent
2073	ED	Non-COVID-19	Total	9/2/2020	88.6	Percent
2074	ED	Non-COVID-19	Total	9/9/2020	88.3	Percent
2075	ED	Non-COVID-19	Total	9/16/2020	87.5	Percent
2076	ED	Non-COVID-19	Total	9/23/2020	87.7	Percent
2077	ED	Non-COVID-19	Total	9/30/2020	85.5	Percent

Figure 3.1: *Example of pre-processed Hospital Data before split by Setting*

sure on emergency department (ED) and inpatient department (IP) separately, the pre-processed "value" column from the last step is split into two distinct columns corresponding to "ED" and "IP" by this "Setting" indicator. Then we further sort these two new columns by date, given by the "Start_Time" column. To differentiate them, they are renamed to "Non_COVID_Percent_ED" and "Non_COVID_Percent_IP" correspondingly. After the splitting process, the "value" column is divided into two new columns we just created.

With the following two created new variables:

- "Non_COVID_Percent_ED",
- "Non_COVID_Percent_IP",

we form a subset of data by combining the information about those two variables with the "Start_Time".

Result: After pre-processing, the Hospital Data contain only three variables:

1. "Start_Time",
2. "Non_COVID_Percent_ED"($x_1^{(2)}$),
3. "Non_COVID_Percent_IP"($x_2^{(2)}$)

The example of this pre-processed Hospital Data is shown as below in Figure 3.2:

	Start_Time	Non_COVID_Percent_ED	Non_COVID_Percent_IP
0	2020-03-18	97.7	91.0
1	2020-03-25	95.4	78.3
2	2020-04-01	91.8	74.0
3	2020-04-08	91.7	75.1
4	2020-04-15	92.0	79.9

Figure 3.2: *Example of pre-processed Hospital Data*

3.2.2 Converting

Start_Time: Similar to the "cdc_case_earliest_dt"($x_1^{(1)}$) column mentioned in Section 3.1.2, the "Start_Time" column is converted from "object"

to "datetime64" datatype so it can be used to aggregate with other two datasets.

3.2.3 Plotting

After the pre-processing steps for the Hospital Data dataset, in Figure 3.3, we plot the measurements of "Non_COVID_Percent_ED"($x_1^{(2)}$) or "Non_COVID_Percent_IP"($x_2^{(2)}$) versus the date recorded in the "Start_Time" column for the period from April 2020 to the end of November 2021.

In Figure 3.3, the operation strain experienced by hospitals is visually represented by the two lines on the plot, denoting the percentages of non-COVID-19 patients in the Inpatient Department and the Emergency Department. When patients are admitted to a hospital, they are typically categorized as either confirmed or suspected COVID-19 cases, with the exception of non-COVID-19 cases. The proportion of non-COVID-19 patients admitted provides an indication of the current hospital workload attributed to confirmed or suspected COVID-19 cases. As the percentage of non-COVID-19 patients admitted to hospitals decreases, it signifies a higher proportion of confirmed or suspected COVID-19 cases. In other words, when a smaller portion of non-COVID-19 cases are present, the majority of incoming patients are more likely to be confirmed or suspected to have COVID-19. This observation highlights a clear trend indicating a decline in the proportions of non-COVID-19 patients in both Inpatient department and Emergency department.

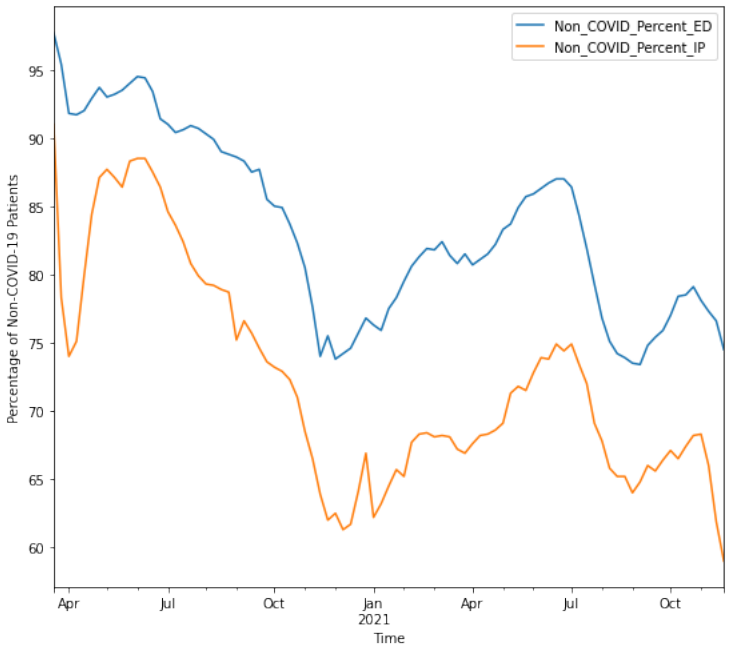


Figure 3.3: *The percentage of non-COVID-19 patients admitted to selected hospitals during the COVID-19 waves*

3.3 Vaccination Data

3.3.1 Subsets of Death Information

The two Vaccination Data datasets described in Section 2.3 can be used to study the effectiveness of vaccination on preventing or reducing death in each age group.

Three types of vaccination recorded by CDC in this dataset which are Pfizer, Moderna, and Janssen (Johnson&Johnson). According to the study conducted by Lin et al. (2022) in North Carolina, all these three Covid-19 vaccines demonstrated persistent efficacy in decreasing the probabilities of

both hospitalization and mortality. Hence, our following analyses will not focus on assessing the impact of vaccination by the vaccine type; instead, we will concentrate on determining the impact on changing mortality according to the vaccination status of the patients. In addition, our analyses will not incorporate possible gender effects because the CDC has not included gender information in these datasets.

To this end, we concentrate on those measurements for those subjects who died of COVID-19 and remove all other columns that contain duplicated or irrelevant information for this study, except for the columns mentioned in Section 2.3.2. This process gives us a subset of 222 subjects to be analyzed in the sequel.

3.3.2 Converting and Creating

- Date: the columns named "MMWR week" and "mmwr_week", stated in Section 2.3.2, in these two datasets indicate the MMWR epidemiological year and week for each death outcome, respectively. For both Vaccination Data datasets, the date column is first converted to strings from "object" datatype, and then a new date column in the "YYYY-MM-DD" format is created from the "MMWR week" column to track the variant waves, with the corresponding MMWR week column removed to avoid duplication.

- Death_rate ($x_1^{(3)}$): The death rate for each age group is calculated as the population-weighted sum of the death rate for each vaccination status,

given by:

$$x_1^{(3)} = w_1 \cdot \frac{\#(\text{vaccinated})}{\#(\text{vaccinated}) + \#(\text{unvaccinated})} + w_2 \cdot \frac{\#(\text{unvaccinated})}{\#(\text{vaccinated}) + \#(\text{unvaccinated})},$$

where w_1 represents the unadjusted incidence rate of death among the population vaccinated with at least a primary series, determined by "Crude vax IR"; w_2 represents unadjusted incidence rate of the corresponding outcome among the unvaccinated population, determined by "Crude unvax IR"; #(vaccinated) stands for the fully vaccinated population; and #(unvaccinated) is determined by unvaccinated population. These values can be obtained using the corresponding population columns and incidence rate columns described in Section 2.3.2.

Result: After preparation, this Vaccination Data subset contains three variables: date, "age_group", and "death_rate"($x_1^{(3)}$). The date and "age_group" column will be used as keys to join with the COVID-19 Case Data dataset based on the information on $x_1^{(1)}$("cdc_case_earliest_dt") and $x_3^{(1)}$("age_group").

3.4 Summary of Variables After Pre-processing

For ease of reference in subsequent analysis, we summarize the pre-processed covariates in Table 3.1, with their original dataset source names used. All these covariates will be merged into one single dataframe as described in the following section, to be used for the feature selection and building the models to be described in Chapter 4. The datatype of each covariate is shown in the last column of Figure 3.4 for reference.

Table 3.1: *Summary of Covariates After Pre-processing*

Covariate's name	**Mathematical Symbol**	**Dataset Source**
cdc_case_earliest_dt	$x_1^{(1)}$	COVID-19 Case Data
sex	$x_2^{(1)}$	COVID-19 Case Data
age_group	$x_3^{(1)}$	COVID-19 Case Data
race_ethnicity_combined	$x_4^{(1)}$	COVID-19 Case Data
hosp_yn	$x_5^{(1)}$	COVID-19 Case Data
icu_yn	$x_6^{(1)}$	COVID-19 Case Data
hc_work_yn	$x_7^{(1)}$	COVID-19 Case Data
pna_yn	$x_8^{(1)}$	COVID-19 Case Data
abxchest_yn	$x_9^{(1)}$	COVID-19 Case Data
acuterespdistress_yn	$x_{10}^{(1)}$	COVID-19 Case Data
mechvent_yn	$x_{11}^{(1)}$	COVID-19 Case Data
fever_yn	$x_{12}^{(1)}$	COVID-19 Case Data
sfever_yn	$x_{13}^{(1)}$	COVID-19 Case Data
chills_yn	$x_{14}^{(1)}$	COVID-19 Case Data
myalgia_yn	$x_{15}^{(1)}$	COVID-19 Case Data
runnose_yn	$x_{16}^{(1)}$	COVID-19 Case Data
sthroat_yn	$x_{17}^{(1)}$	COVID-19 Case Data
cough_yn	$x_{18}^{(1)}$	COVID-19 Case Data
sob_yn	$x_{19}^{(1)}$	COVID-19 Case Data
nauseavomit_yn	$x_{20}^{(1)}$	COVID-19 Case Data
headache_yn	$x_{21}^{(1)}$	COVID-19 Case Data
abdom_yn	$x_{22}^{(1)}$	COVID-19 Case Data
diarrhea_yn	$x_{23}^{(1)}$	COVID-19 Case Data
medcond_yn	$x_{24}^{(1)}$	COVID-19 Case Data
Death_rate	$x_1^{(2)}$	Vaccination Data
Non_COVID_Percent_ED	$x_1^{(3)}$	Hospital Data
Non_COVID_Percent_IP	$x_2^{(3)}$	Hospital Data

3.5 Data Integration and Division

3.5.1 Integration

The three datasets, namely COVID-19 Case Data, Hospital Data, and Vaccination Data, are combined by aligning them based on date or age groups. First, the Hospital Data are matched with COVID-19 Case Data using the date only, as the Hospital Data lack information on age groups.

Next, the Vaccination Data are integrated with the combined Hospital Data and COVID-19 Case Data, using both age groups and date as matching criteria. As CDC has different age group structures for the primary doses dataset and the booster dose dataset (Section 2.3.2) contained in the Vaccination Data dataset, the age group matching process is done by manually specifying the matched pairs. While the available information does not yield a perfect match for age groups, we pair age groups by rounding. For example, the age group "60-69 years" in the COVID-19 Case Data cannot be matched perfectly with the age groups of "50-64" and "65-79", or "50-64" and "65+" in the Vaccination Data. In this case, "65-79" or "65+" groups in Vaccination Data will be chosen as the paired age group for "60-69" in the COVID-19 Case Data. The detailed information about age group pairing can be found in Figure 3.4.

As a result, the final integrated dataset has three additional columns with 27 covariates in total, in which 24 covariates, $\{x_1^{(1)} \ ... \ x_{24}^{(1)}\}$, comes from the COVID-19 Case Data dataset, two covariates, "Non_COVID_Percent_ED"$(x_1^{(2)})$ and "Non_COVID_Percent_IP"$(x_2^{(2)})$ come from Hospital Data, and one covariate, "death_rate"$(x_1^{(3)})$, is from Vaccination Data.

COVID-19 Case Data Age Group	Primary Vaccination Age Group	Booster Vaccination Age Group
0–9	12–17	12–17
10–19	12–17	12–17
20–29	18–29	18–49
30–39	30–49	18–49
40–49	30–49	18–49
50–59	50–64	50–64
60–69	65–79	65 +
70–79	65–79	65 +
80 +	80 +	65 +

Figure 3.4: *Age group pairing in COVID-19 case data, primary doses, and booster doses from the Vaccination Data*

The information is summarized in Figure 3.5, where the first column lists the name of covariates, the second column indicates the number of non-empty observations in the corresponding column, and the last column of the figure shows the datatype for each covariate after being pre-processed, as mentioned in Section 3.4. Detailed descriptions about the covariates are included in the Table 3.2.

Remark: Since the Hospital Data start on 2020-03-18 but the COVID-19 Cases Data start on 2020-01-01, there are no hospital data between 2020-01-01 and 2020-03-18, yield missing values for "Non_COVID_Percent_ED"($x_1^{(2)}$) and "Non_COVID_Percent_IP"($x_2^{(2)}$). As CDC did not start documenting the corresponding hospital data to address the impact of COVID-19 from 2020-01-01 to 2020-03-18, all patients in this time window are non-COVID-19, so that "Non_COVID_Percent_ED"($x_1^{(2)}$) and "Non_COVID_Percent_IP"($x_2^{(2)}$) are set to 100%. As a result, the missing values are filled with number 100, assuming that the non-COVID-19 patient percentage is 100% for the period between 2020-01-01 and 2020-03-18.

```
Data columns (total 28 columns):
 #   Column                    Non-Null Count   Dtype
---  ------                    --------------   -----
 0   abdom_yn                  242774 non-null  object
 1   abxchest_yn               242774 non-null  object
 2   acuterespdistress_yn      242774 non-null  object
 3   age_group                 242774 non-null  object
 4   date                      242774 non-null  datetime64[ns]
 5   chills_yn                 242774 non-null  object
 6   cough_yn                  242774 non-null  object
 7   death_yn                  242774 non-null  object
 8   diarrhea_yn               242774 non-null  object
 9   fever_yn                  242774 non-null  object
 10  sfever_yn                 242774 non-null  object
 11  hc_work_yn                242774 non-null  object
 12  headache_yn               242774 non-null  object
 13  hosp_yn                   242774 non-null  object
 14  icu_yn                    242774 non-null  object
 15  mechvent_yn               242774 non-null  object
 16  medcond_yn                242774 non-null  object
 17  myalgia_yn                242774 non-null  object
 18  nauseavomit_yn            242774 non-null  object
 19  pna_yn                    242774 non-null  object
 20  race_ethnicity_combined   242774 non-null  object
 21  runnose_yn                242774 non-null  object
 22  sex                       242774 non-null  object
 23  sob_yn                    242774 non-null  object
 24  sthroat_yn                242774 non-null  object
 25  Non_COVID_Percent_ED      242774 non-null  float64
 26  Non_COVID_Percent_IP      242774 non-null  float64
 27  death_rate                47651 non-null   float64
dtypes: datetime64[ns](1), float64(3), object(24)
memory usage: 53.7+ MB
```

Figure 3.5: *Summary of the integrated dataset. The first column records the index of each covariate, the second column lists the name of covariates, the third columns shows the number of non-null records for each covariate, and the last column indicates the corresponding datatype of each covariate*

In Table 3.2, we present descriptive statistics for all columns within the integrated dataset, where each row displays the count of each feature, together with its mean, standard deviation, minimum and maximum values, as well as the 25th, 50th, and 75th percentile values.

Table 3.2: *Descriptive Statistics for Covariates displayed in Figure 3.5*

Variable	Count	Mean	Std	Min	25%	50%	75%	Max
abdom_yn	242774	0.116	0.320	0.000	0.000	0.000	0.000	1.000
abxchest_yn	242774	0.046	0.209	0.000	0.000	0.000	0.000	1.000
acuterespdistress_yn	242774	0.013	0.112	0.000	0.000	0.000	0.000	1.000
age_group	242774	3.692	2.057	0.000	2.000	4.000	5.000	8.000
date	242774	341.042	156.471	0.000	254.000	307.000	418.000	777.000
chills_yn	242774	0.397	0.489	0.000	0.000	0.000	1.000	1.000
cough_yn	242774	0.613	0.487	0.000	0.000	1.000	1.000	1.000
death_yn	242774	0.014	0.119	0.000	0.000	0.000	0.000	1.000
diarrhea_yn	242774	0.248	0.432	0.000	0.000	0.000	0.000	1.000
fever_yn	242774	0.344	0.475	0.000	0.000	0.000	1.000	1.000
sfever_yn	242774	0.379	0.485	0.000	0.000	0.000	1.000	1.000
hc_work_yn	242774	0.120	0.325	0.000	0.000	0.000	0.000	1.000
headache_yn	242774	0.570	0.495	0.000	0.000	1.000	1.000	1.000
hosp_yn	242774	0.079	0.269	0.000	0.000	0.000	0.000	1.000
icu_yn	242774	0.014	0.119	0.000	0.000	0.000	0.000	1.000
mechvent_yn	242774	0.005	0.073	0.000	0.000	0.000	0.000	1.000
medcond_yn	242774	0.437	0.496	0.000	0.000	0.000	1.000	1.000
myalgia_yn	242774	0.512	0.500	0.000	0.000	1.000	1.000	1.000
nauseavomit_yn	242774	0.220	0.414	0.000	0.000	0.000	0.000	1.000
pna_yn	242774	0.045	0.208	0.000	0.000	0.000	0.000	1.000
race_ethnicity_combined	242774	5.165	1.555	0.000	6.000	6.000	6.000	6.000
runnose_yn	242774	0.436	0.496	0.000	0.000	0.000	1.000	1.000
sex	242774	0.443	0.497	0.000	0.000	0.000	1.000	1.000
sob_yn	242774	0.248	0.432	0.000	0.000	0.000	0.000	1.000
sthroat_yn	242774	0.353	0.478	0.000	0.000	0.000	1.000	1.000
Non_COVID_Percent_ED	242774	81.309	6.374	73.400	75.700	80.500	85.500	100.000
Non_COVID_Percent_IP	242774	69.889	7.799	59.000	64.100	68.100	73.600	100.000
death_rate	47651	2.867	4.671	0.000	0.326	0.704	2.962	30.448

3.5.2 Division

To access the impact of the variant type of virus, we also divide the integrated dataset in Section 3.5.1 into 5 subsets based on date according to the Variant of Concerns (VOCs) published by World Health Organization (2023b), and display the information in Table 3.2. To be specific, the 5 subsets are:

- Outbreak: Cases confirmed before 2020-12-18. VOC: None.
- AlphaBeta: Cases confirmed between 2020-12-18 and 2021-01-11. VOC: Alpha & Beta variants.
- Gamma: Cases confirmed between 2021-01-11 and 2021-05-11. VOC: Gamma variant.
- Delta: Cases confirmed between 2021-05-11 and 2021-11-26. VOC: Delta variant.
- Omicron: Cases confirmed after 2021-11-26. VOC: Omicron variant.

Remark: There is the disparity in data availability. The Vaccination Data start on 2021-04-11, yet the COVID-19 Cases Data start on 2020-01-01. Also, shown in Figure 3.4, the "death_rate" ($x_1^{(3)}$) column in the integrated dataset contains only 47,651 non-null values out of a total of 242,774 rows.

Table 3.3: *Variant of Concerns(VOCs)*

Date	Subset name	Variant	Subset Size (# of patients)
2020-01-01 to 2020-12-18	Outbreak	None	135727
2020-12-18 to 2021-01-11	AlphaBeta	Alpha & Beta variants	20086
2021-01-11 to 2021-05-11	Gamma	Gamma variant	39855
2021-05-11 to 2021-11-26	Delta	Delta variant	36224
2021-11-26 to 2022-04-04	Omicron	Omicron variant	11287

Chapter 4

An Overview of Machine Learning Methods

This chapter 4 provides an overview of useful machine learning techniques that will be employed to analyze the integrated dataset described in Section 3.5.

4.1 Feature Selection

Feature selection is a useful technique in data analysis and machine learning, as it assists in identifying the most pertinent and informative features by reducing the dimension of the original dataset. The reduction in dimensionality can lead to improved computational efficiency and performance of machine learning algorithms (Guyon and Elisseeff, 2003). Furthermore, by eliminating irrelevant or redundant features, feature selection mitigates the risk of overfitting and enhancing the model's ability to generalize well to

handle future data (Dash and Liu, 1997).

Jović, Brkić, and Bogunović (2015) provided a comprehensive review on various techniques used for feature selection, and they mentioned that filter methods based on information theory and wrapper methods based on greedy stepwise approaches usually offer best results. Here we particularly describe three approaches: the Filter Method works for univariate data, and the Wrapper Method and the Embedded Method that apply to multivariate data.

4.1.1 Filter Method

The *Filter method* is a technique used to identify the most relevant features from a dataset. It measures the relevance of covariates with the target variable by using different statistical methods to calculate the correlation (Bommert et al., 2020).

For the case with two continuous variables, *Pearson's Correlation Coefficient* can be used as a measure to describe the linear dependence between two continuous variables X and Y. Let $\{\{X_i, Y_i\} : i = 1, \ldots, n\}$ denote a random sample taken from the distribution of $\{X, Y\}$, then Pearson's Correlation Coefficient for X and Y is given by

$$r_{XY} \triangleq \frac{\Sigma_{i=1}^{n}(X_i - \bar{X})(Y_i - \bar{Y})}{\sqrt{\Sigma_{i=1}^{n}(X_i - \bar{X})^2}\sqrt{\Sigma_{i=1}^{n}(Y_i - \bar{Y})^2}}$$

where $\bar{X} = n^{-1}\Sigma_{i=1}^{n} X_i$ and $\bar{Y} = n^{-1}\Sigma_{i=1}^{n} Y_i$.

As noted by Rodgers and Nicewander (1988) this coefficient is invariant to linear transformations of either variable.

The *correlation ratio* is used to calculate the association between a categorical variable, say X, and a numeric variable, say Y; the definition is provided by the **Dython** documentation(Zychlinski, n.d.). Suppose the ith observation for Y is y_{xi}, corresponding to the category $X = x$. Let n_x denote the number of observations in category $X = x$, and let $\bar{y}_x = \frac{\sum_i y_{xi}}{n_x}$ represent the mean of observations for $X = x$; and let $\bar{y} = \frac{\sum_i n_x \bar{y}_x}{\sum_x n_x}$. Then the correlation ratio for X and Y is defined as:

$$\eta_{XY} = \sqrt{\frac{\sum_x n_x(\bar{y}_x - \bar{y})^2}{\sum_{x,i} n_x(y_{xi} - \bar{y})^2}}.$$

If both X and Y are categorical variables, then the *Chi-square test* of independence can be applied to evaluate the association between X and Y. This Chi-square test relies on a contingency table, which is a tabular representation of the observed frequencies of the categorical variables' combinations (Bolboacă et al., 2011). For combination of categories from X and Y indexed by (i, j), let $O_{i,j}$ represent the observed value in the contingency table that records the counts of occurrences, and let $E_{i,j}$ denote the count that would be seen if X and Y were independent. These expected values ($E_{i,j}$) can be calculated as:

$$E_{i,j} = \frac{(\sum_j O_{i,j})(\sum_i O_{i,j})}{N},$$

where, $\sum_j O_{i,j}$ represents the sum of observed values in row i, $\sum_i O_{i,j}$ represents the sum of observed values in column j, and N is the total number of observations.

The Chi-square statistic is then calculated based on the differences between observed and expected values in each cell, quantifying how much the observed data deviates from what would be expected under the independence

assumption for X and Y. Specifically, the Chi-square statistic is defined as:

$$T = \sum_{i=1}^{r} \sum_{j=1}^{c} \frac{(O_{i,j} - E_{i,j})^2}{E_{i,j}},$$

where r represents the number of rows and c represents the number of columns in the contingency table. Under the independence assumption, the statistic T follows the Chi-Square distribution with degrees of freedom $(r-1)(c-1)$.

Concerning two categorical variables, say X and Y, one may also use Cramér's V to measure the strength of association. This measure adapts the Chi-Square statistic that is normalized to provide a value ranging from 0 to 1, inclusive. Let n denote the sample size, and let r and c represent the number of rows and columns in the contingency table, respectively. Cramér's V was initially introduced by Cramér (1999) as:

$$V = \sqrt{\frac{\chi^2}{n(z-1)}},$$

where χ^2 is the Chi-square statistic defined in Section 4.1.1, and $z = min(r, c)$.

The measure V assumes a value in the internal $[0, 1]$. The higher the Cramér's V value, the stronger ther association between X and Y. As mentioned by Akoglu (2018), a Cramér's V value greater than 0.1 is considered a moderate association, and a value greater than 0.25 is regarded a strong association.

4.1.2 Wrapper Method

The *wrapper method*(Das, 2001) is a machine learning algorithm that is a greedy search based approach. This method evaluates all possible subsets

of features to against a specific evaluation criterion.

With regression models, an evaluation criterion can be based on *P-value*s or *R-squared score*. The P-value is a measure of the statistical significance of the relationship between the predictor variables and the response variable in the regression model. Wasserstein and Lazar (2016) stated that the P-value is "probability under a specified statistical model that a statistical summary of the data (for example, the sample mean difference between two compared groups) would be equal to or more extreme than its observed value". In regression analysis, each predictor variable has its own P-value associated with its coefficient estimate. A low P-value, typically below a chosen significance level, such as 0.05, suggests that the predictor variable has a significant impact on the response variable, meaning the relationship is unlikely to be due to random chance. On the other hand, the R-squared score is a sample estimate that represents the proportion of variance in the response variable that is explained by the predictor variables in the regression model(Miles, 2005). It indicating the goodness-of-fit of the model, takes a value from 0 to 1, where 0 indicates that the predictor variables explain none of the variance and 1 indicates a perfect fit where all the variance is explained.

For binary classification tasks, the criterion can be *accuracy*, *precision*, *F1 score*, and so on. The accuracy is simply the ratio of correctly classified instances to the total number of instances in the sample, and the precision is the proportion of correctly predicted positive cases out of all positive predictions made by the model (both true positives and false positives)(Goutte and Gaussier, 2005). The F1 score is a combined measure of accuracy and precision.

To be specific, suppose there are N classes in total, let f_j be the F1 score for the jth class for $j = 1, \ldots, N$. Let true positives (TP) and false positives (FP) represent the count of correctly and incorrectly predicted positive instances, respectively. Likewise, true negatives (TN) and false negatives (FN) indicate the number of correctly and incorrectly predicted negative instances, respectively. Then the *recall*, precision, F1 Score, and *Macro F1 Score* are defined as(Goutte and Gaussier, 2005):

$$\text{Recall} = \frac{\text{TP}}{\text{TP+FN}};$$

$$\text{Precision} = \frac{\text{TP}}{\text{TP+FP}};$$

$$\text{F1 Score} = \frac{\text{Precision} * \text{Recall}}{\text{Precision} + \text{Recall}};$$

$$\text{Macro F1 Score} = \frac{\sum_{j=1}^{N} f_j}{\text{N}}.$$

To make the feature selection process of the wrapper method more explainable and straightforward, the sequential selection strategies can be used. According to El Aboudi and Benhlima (2016), there are two sequential selection strategies:

- *The Sequential Feature Selection algorithm* (Forward Selection): Starting with no feature in the model, we keep adding features to improve the performance of the resulting model in each iteration until the performance of the model cannot be improved by adding more features.

- *The Sequential Backward Selection algorithm* (Backward Elimination): Starting with all features in the model, we keep removing features to

improve the performance of the resulting model in each iteration until there is no improvement on the model when removing additional features.

4.1.3 Embedded Method

The *Embedded method* (Lal et al., 2006) is a feature selection approach which is integrated as part of a machine learning algorithm. It performs the feature selection process during the algorithm's execution, and it takes care of each iteration during the model training process and carefully extracts those features which contribute the most. In this way, the embedded method has less computation time and is less prone to over-fitting.

As commented by Jović et al. (2015), certain embedded methods utilize regularization models, such as derived from using the *Lasso method*, to assign weights to features to minimize fitting errors while simultaneously encourage small or zero coefficients corresponding to the features. These methods are commonly employed with linear classifiers and apply penalties to the features that have little contribution to the model.

For example, logistic regression can be used as a straightforward linear classifier, in combination with the Lasso or L1 penalization function to select only the predictors with non-zero coefficients, resulting in a reduced overall dimensionality. On the other hand, under the linear regression model, with the response vector $\mathbf{y}$ in $\mathbb{R}^n$ and feature matrix $\mathbf{X} = [\mathbf{x}^1, \ldots, \mathbf{x}^p]$ in $\mathbb{R}^{n \times p}$, the Lasso regularization can be formed as(Rosset and Zhu, 2007):

$$\min_{\mathbf{w} \in \mathbb{R}^p} \Big\{ \frac{1}{2} \|\mathbf{y} - \mathbf{X}\mathbf{w}\|_2^2 + \lambda \|\mathbf{w}\|_1 \Big\},$$

where the L_1-norm induces sparsity in the solution $\mathbf{w}$ and $\lambda > 0$ controls the amount of regularization.

4.2 Machine Learning Models

In this section, our primary objective is to consider robust yet effective models for data classification. considering on the balanced datasets, we employ a variety of algorithms and techniques to optimize their performances in our classification task. Specifically, we review six machine learning algorithms: *Support Vector Machine (SVM)*, *Decision Tree*, *Random Forest*, *Logistic Regression*, *K-Nearest Neighbour (KNN)*, and *Naïve Bayes*, with each algorithm explained in the subsequent subsections.

4.2.1 Support Vector Machines

The Support Vector Machine(SVM)(Noble, 2006) is a supervised learning algorithm that can be used for both regression and classification tasks. In general, a SVM classifier is aimed to find a separating hyperplane that maximizes the margin between the two classes of samples, i.e., the distance between the hyperplane and the nearest data points from each class(Mohri, Rostamizadeh, and Talwalkar, 2018).

For a training set

$$S = \{(x_i, y_i) : x_i \in \mathbb{R}^p; y_i \in \{-1, +1\}; i = 1, \ldots, m\}$$

that is linearly separable, we want to find the hyperplane satisfying:

$$y_i(\mathbf{w}^\intercal x_i + b) \geq 0 \text{ for } i = 1, \ldots, m,$$

where $\mathbf{w}$ is a non-zero $p \times 1$ vector and b is a scalar that are to be determined. The problem can be equivalently formulated as the optimization problem to find the optimal $\mathbf{w}$ and b:

$$\min_{\mathbf{w},b} \left\{ \frac{1}{2} \|\mathbf{w}\|^2 \right\}$$

subject to

$$y_i(\mathbf{w}^\intercal x_i + b) \geq 1 \text{ for } i = 1, \ldots, m,$$

where $\|\mathbf{w}\|$ is the Euclidean norm of vector $\mathbf{w}$(Mohri et al., 2018).

In the case that S is not linearly separable, one may project the features into a higher dimensional space so that the transformed features can be linearly separated. Alternatively, we keep the features in their original space but introduce slack variables to allow the SVM to accommodate some classification errors. The optimization problem is then defined as:

$$\min_{\mathbf{w},b} \left\{ \frac{1}{2} \|w\|^2 + C \sum_{i=1}^{m} \xi_i \right\}$$

subject to

$$y_i(w^T x_i + b) \geq 1 - \xi_i;$$

$$\xi_i \geq 0 \text{ for } i = 1, \ldots, m,$$

where C is the cost coefficient and the ξ_i are the slack variables which measure the degree of misclassification of each feature x_i (e.g., Jiang, Missoum, and Chen, 2014; He, Yi, and Chen, 2019).

4.2.2 Decision Tree

Decision Tree is a non-parametric supervised learning algorithm for which data are not assumed to follow on probability distribution. The construction of a decision tree is to recursively partition the feature space of the training set in order to find a set of decision rules(Myles et al., 2004). A decision tree can be viewed as a flowchart-like model, where each internal node represents a feature or attribute, each branch represents a decision rule, and each leaf node represents the outcome or class label. This method is easier to understand and the results are readily visualized.

The decision tree classifier determines the optimal splits by using various measures of impurity, such as the *Gini impurity* or *entropy*. The **DecisionTreeClassifier** in the **Scikit-Learn** Python library takes the Gini impurity as the default criterion(Pedregosa et al., 2011).

4.2.3 Random Forest

Random Forest is an ensemble method that improves the accuracy and avoids the over-fitting problem by using a collection of decision trees. Random forest is used to classify a new instance by majority vote based on the construction of a number of decision trees, where each decision tree in the random forest model randomly uses a subset of features and different bootstrap sample data for its construction(Oshiro, Perez, and Baranauskas, 2012).

Since a random forest is an integrated estimator that fits several decision tree classifiers, it uses the same criterion as the decision tree classifier does.

4.2.4 Logistic Regression

Logistic regression is a simple statistical model that characterizes the probability of an event happening based on a given dataset. This model is useful to facilitate the relationship between a binary outcome variable and a vector of predictor variables that can be categorical or continuous(Peng, Lee, and Ingersoll, 2002).

For $i = 1, \ldots, n$, let Y_i represent the binary variable for subject i, taking value 0 or 1, and let X_i denote the vector of associated covariates. Let $\mu_i = P(Y_i = 1|X_i)$. Logistic regression describes the relationship between Y_i and X_i as follows:

$$\text{logit}(\mu_i) = \theta_0 + \theta_x^T X_i$$

for $i = 1, \ldots, n$, where $\text{logit}(t) = log\frac{t}{1-t}$ for $0 < t < 1$, and θ_0 and θ_x represent regression parameters.

4.2.5 K-Nearest Neighbour

The K-Nearest Neighbours (K-NN) is another non-parametric supervised learning algorithm. The K-NN method is a simple and widely used learner, and sometimes, it is called *Lazy Learning*. It is also called *Memory-based Classification* since KNN needs to be in memory at runtime and the induction is delayed to runtime (Cunningham and Delany, 2021).

This method assigns a class label to a given data input point with the label of the nearest neighbours based on its distance to other data points. Several distance measures are commonly in use, including *Euclidean distance*, *Manhattan distance*, and *Minkowski distance*. The Minkowski distance is the

default distance metric in **Scikit-Learn** package. Minkowski distance of order p between two points, $u = (u_1, u_2, \ldots, u_n)$ and $v = (v_1, v_2, \ldots, v_n)$, is defined as(Pedregosa et al., 2011):

$$\|u - v\|_p = \left(\sum_{i=1}^{n} |u_i - v_i|^p \right)^{1/p},$$

where $p > 0$.

4.2.6 Gaussian Naïve Bayes

Gaussian Naïve Bayes algorithm is one of Naïve Bayes methods that naively assumes every pair of features is conditional independent, given the class variable(Jahromi and Taheri, 2017). This algorithm assumes that the features follow a Gaussian distribution.

Let Y denote the target class variable, and let $\{X_1, \ldots, X_n\}$ denote the dependent feature vectors. Assume that given $Y = y$, $X_1, \ldots, X_n$ independently follow a Gaussian distribution, $N(\mu_y, \sigma_y^2)$, with mean μ_y and standard error σ_y. That is, the density of X_i is given by

$$f(x_i|y) = \frac{1}{\sqrt{2\pi\sigma_y}} \exp\Big\{ -\frac{(x_i - \mu_y)^2}{2\sigma_y^2} \Big\},$$

where μ_y and σ_y can be estimated from the *maximum likelihood method*. Then the conditional probability for Y given $\{x_1, \ldots, x_n\}$ is determined by:

$$P(Y = y|X_1, \ldots, X_n) = \frac{f(y) f(x_1, \ldots, x_n|y)}{f(x_1, \ldots, x_n)} = \frac{f(y) \prod_{i=1}^{n} f(x_i|y)}{\sum_y f(y) \prod_{i=1}^{n} f(x_i|y)},$$

where $f(y)$ is the prior probability of the class.

To classify a new data instance, the classifier first calculates the likelihood of the observed features given each class using the Gaussian distribution. The

class with the highest probability becomes the predicted class for the new data point.

Because two assumptions required by the Gaussian Naive Bayes classifier often do not hold in real-world scenarios, the performance of the classifier may be compromised when confronted with features that do not adhere to a Gaussian distribution or when significant dependencies exist among the features.

4.2.7 Artificial Neural Networks

An *artificial neural network* (ANN) is a computational model inspired by the structure and function of the human brain. It consists of artificial neurons, serving as the fundamental processing units, and a complex network of connections between these neurons (Osowski, Siwek, and Markiewicz, 2004).

Mimicking the brain, an ANN processes information by receiving inputs, applying mathematical operations on those inputs, and producing output signals. The interconnections between neurons allow for the network to learn and adapt its behaviour through the training process. During training, the network adjusts the weights and biases associated with each connection based on the provided input-output pairs, allowing it to learn and make predictions or decisions on unseen data.

4.2.7.1 Multi-Layers Perceptron

Multi-layer perceptron (MLP) is a powerful supervised learning algorithm widely used for approximating non-linear functions in classification and regression tasks. MLP consists of multiple layers: the input layer, which rep-

resents the input features; hidden layers, which receive the values from the previous layer and apply non-linear transformations using various activation functions; and the output layer, which takes the transformed values from the last hidden layer and produces the final output values. By incorporating multiple hidden layers with non-linear activation functions, MLP is capable of capturing complex patterns and relationships within the data. Murtagh (1991) lists a few popular activation functions, including the sigmoid, tanh, and ReLU functions. The **Scikit-Learn** document (Pedregosa et al., 2011) provides the logistic sigmoid function:

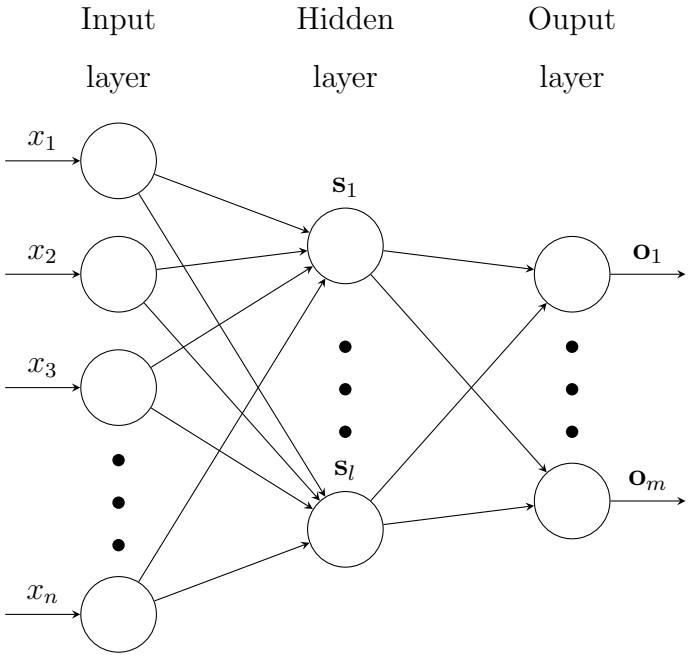
$$f(x) = \frac{1}{1 + \exp(-x)} \qquad \text{for} - \infty < x < \infty.$$

4.2.7.2 Radial Basis Function Network

The *radial basis function network (RBFN)*, introduced by Broomhead and Lowe (1988), is an alternative to the MLP neural network. Unlike the MLP, the RBFN has a fixed three-layer architecure, including a input layer, a single hidden layer that utilizes radial basis functions as activation functions, and the output layer. According to a comparative study by Xie, Yu, and Wilamowski (2011), there are four key differences between RBFN and MLP networks, despite their similar structures. Firstly, RBFN generally has faster training processes due to its simpler architecture than MLP networks. Secondly, as local approximation networks, RBFN functions rely on specific hidden units within local receptive fields to determine outputs, while MLP networks operate globally, considering outputs from all neurons. Thirdly, the initial states of RBFN are critical for its performance, whereas MLP networks initially use randomly generated parameters. Lastly, RBFN and

MLP networks employ different classification mechanisms. RBFN separates clusters using hyperspheres, while MLP networks use arbitrarily shaped hypersurfaces for separation.

Suppose there are n inputs $\mathbf{x} \triangleq \{x_1, \ldots, x_n\}$, l hidden units $\{\mathbf{s}_1, \ldots, \mathbf{s}_l\}$, and m outputs $\{\mathbf{o}_1, \ldots, \mathbf{o}_m\}$; the structure and formulas of RBFN is shown as follows (Xie et al., 2011):



For $h = 1, \ldots, l$, the input vector $\mathbf{x}$ is multiplied by input weights $\mathbf{w}^h$ at the input of hidden unit h to calculate $\mathbf{s}_h$. That is, if $w^h_{n,l}$ is the input weight between input n and hidden unit h:

$$\mathbf{s}_h = [x_1 w^h_1, x_2 w^h_2, \ldots, x_n w^h_n].$$

The output of hidden unit h is calculated with the activation function ϕ_h

where ϕ_h is usually chosen as a Gaussian function:

$$\phi_h(\mathbf{s}_h) = \exp\Big(-\frac{\|\mathbf{s}_h - \mathbf{c}_h\|^2}{\sigma_h}\Big),$$

where $\mathbf{c}_h$ is the centre of hidden unit h and σ_h is the width of hidden unit h.

Finally, the output $\mathbf{o}_k$ is calculated with the output weight $w^o_{h,k}$ between hidden unit h and output unit k, and $w^o_{0,k}$ as the bias weight of output unit k:

$$\mathbf{o}_k = \sum_{h=1}^{l} \phi_h(\mathbf{s}_h) w^o_{h,k} + w^o_{0,k}.$$

4.3 Cost-Sensitive Classification Models

Cost-sensitive classification, as described by Sun, Wong, and Kamel (2009), is an important approach in machine learning that considers the diverse costs associated with different types of misclassifications. Traditional classification problems treat all misclassifications equally and assign them the same cost, which may not reflect the real-world scenario accurately. Cost-sensitive classification tackles this limitation by incorporating varying costs into the classification process. It assigns different costs to misclassifications based on the specific classes or types of misclassifications, thus providing a more realistic and nuanced perspective. This approach enables the development of models that prioritize certain types of errors over others, aligning with the specific requirements and objectives of the problem domain.

4.3.1 Cost-Sensitive SVM

Cost-Sensitive SVM is an adapted version of the conventional SVM algorithm that introduces weights to the margin based on the significance of each class. Unlike traditional SVM, which tends to prioritize the majority class in imbalanced datasets, cost-sensitive SVM considers the importance of each class. This adjustment allows cost-sensitive SVM to enhance performance on imbalanced datasets, surpassing the capabilities of standard SVM algorithms.

As in Cao, Zhao, and Zaiane (2013), let C_+ represent the higher misclassification cost assigned to the positive class, which is of primary interest, and let C_- represent the lower misclassification cost assigned to the negative class. Then the cost-sensitive SVM (CS-SVM) is formulated as follows:

$$\min_{\mathbf{w},b}\Big\{\frac{1}{2}\|w\|^2 + C_+ \sum_{i:y_i=+1} \xi_i + C_- \sum_{j:y_j=-1} \xi_j\Big\}$$

subject to

$$y_i(w^T x_i + b) \geq 1 - \xi_i;$$

$$\xi_i \geq 0 \text{ for } i = 1, \ldots, n.$$

It is clear that the Cost-Sensitive SVM recovers the usual SVM described by (4.2.1) if C_+ and C_- are set to be identical.

4.3.2 Cost-Sensitive Decision Tree

In the context of decision trees, the selection of split points is typically aimed at minimizing overlap between different groups of data samples. However, imbalanced datasets pose a challenge when one class dominates these

groups, leading to the neglect of minority class samples in the split points. With imbalanced datasets, traditional decision trees may not have satisfactory performance. To solve this issue, it becomes crucial to consider the significance of each class during the split point selection process. By incorporating a criterion that accounts for class importance, we can overcome this limitation and enhance the performance of decision trees on imbalanced datasets. For example, Krawczyk, Woźniak, and Schaefer (2014) discussed the use of cost-sensitive decision tree ensembles to improve classification performance on imbalanced datasets, addressing the issue of class imbalance in decision tree learning.

4.3.3 Cost-Sensitive Logistic Regression

The *cost-sensitive logistic regression* technique addresses class imbalance by assigning different weights to each class. This means that the model is penalized differently, depending on the class membership of the samples. For the samples belonging to the minority class, the model incurs higher penalties for errors, and for the majority class samples, they receive relatively less penalty. This approach improves the performance of logistic regression classification.

Considering the use of logistic regression to handle imbalanced datasets, Luo, Pan, Wang, Ye, and Qian (2019) took into account class weights to adjust the decision boundary to address issues of imbalanced data. Basically, the effectiveness of logistic regression may vary depending on the unique characteristics of the dataset under consideration.

Chapter 5

Data Learning

In this chapter we analyze the integrated dataset described in Section 3.5 using the methods reviewed in Chapter 4.

5.1 Feature Selection

In the integrated dataset and all VOC subsets described in Section 3.5, there are 27 features in total, plus the outcome variable "death_yn" (Y). The primary objective of this section is to reduce the dimensionality of the dataset and select the most informative features to predict the outcome variable Y. To achieve this, we employ three feature selection methods introduced in Section 4.1: the Filter Method, Wrapper Method, and Embedded Method, each method resulting in distinct best feature subsets. Based on these results, we further examine the intersection and union sets of these subsets to assess their performance. In the following subsections, we provide detailed results derived from each of these five methods.

5.1.1 Encoding Categorical Variables

There are several popular categorical data encoding methods such as label encoding, one-hot encoding, and so on. Categorical data encoding involves the transformation of non-numeric categories or labels into numerical formats, making them suitable for analysis with machine learning algorithms. However, label encoding cannot capture the relationship/order between the classes. For example, a "temperature" column may be classified as three classes: "cold", "warm", "hot". By employing one-hot encoding, each temperature category is mapped as follows: "cold", "warm", and "hot" are represented as [1, 0, 0], [0, 1, 0], and [0, 0, 1], respectively, with two additional columns included and hence increasing computational cost. On the contrary, using label encoding, one may represent the labels "cold", "warm", and "hot" as arbitrary numerical values such as 5, 3, and 7, respectively, by ignoring any inherent ranking among them.

Several studies have been available to investigate the performance of different encoding methods. For example, Choong and Lee (2017) conducted a comparison between ordinal and one-hot encoding methods. Ordinal encoding is a technique used to represent categorical information using numerical values in a way that retains the inherent order or ranking of the categories. They discovered that ordinal encoding and one-hot encoding method yield similar performance in DNA motif prediction with convolutionary neural network (CNN), but the former encoding significantly reduces the training time. They also found that in certain evaluated datasets, the ordinal encoding proved to be the most optimal approach.

Based on these findings, we adapt ordinal encoding to assign numerical

values for categorical variables similar to Choong and Lee (2017). This treatment is also driven by that most of the categorical covariates in this study are binary indicators, and the "age_group"($x_3^{(1)}$) and "date"($x_1^{(1)}$) have its own ranked ordering. All categorical columns in this study will be converted to numbers using **OrdinalEncoder** in the **Scikit-Learn** package of Python. For example, the 9 different age groups of the covariate "age_group"($x_3^{(1)}$) mentioned in Section 2.1.3 is converted to integers of $\{0, 1, 2..., 8\}$, and the variable "date"($x_1^{(1)}$) is converted to integers of $\{0, 1, 2..., 777\}$ to reflect the temporal order.

In this study, let $X = (x_1^{(1)}, \ldots, x_{24}^{(1)}, x_1^{(2)}, x_1^{(3)}, x_2^{(3)})^T$ denote the vector of covariates, and let Y denote the outcome variable,"death_yn". The detailed definition of those variables can be found in Section 3.4.

5.1.2 Learning with the Filter Method

In the integrated dataset described in Section 3.5.1, a majority of the covariates are categorical variables, as depicted in Figure 3.4. The conversion process, particularly for binary variables, where values are typically transformed into 0s and 1s, can significantly impact variance and mean calculations, hindering the interpretability of the deviations. Consequently, computing Pearson's Correlation coefficient alone for all covariates may not provide the desired insights.

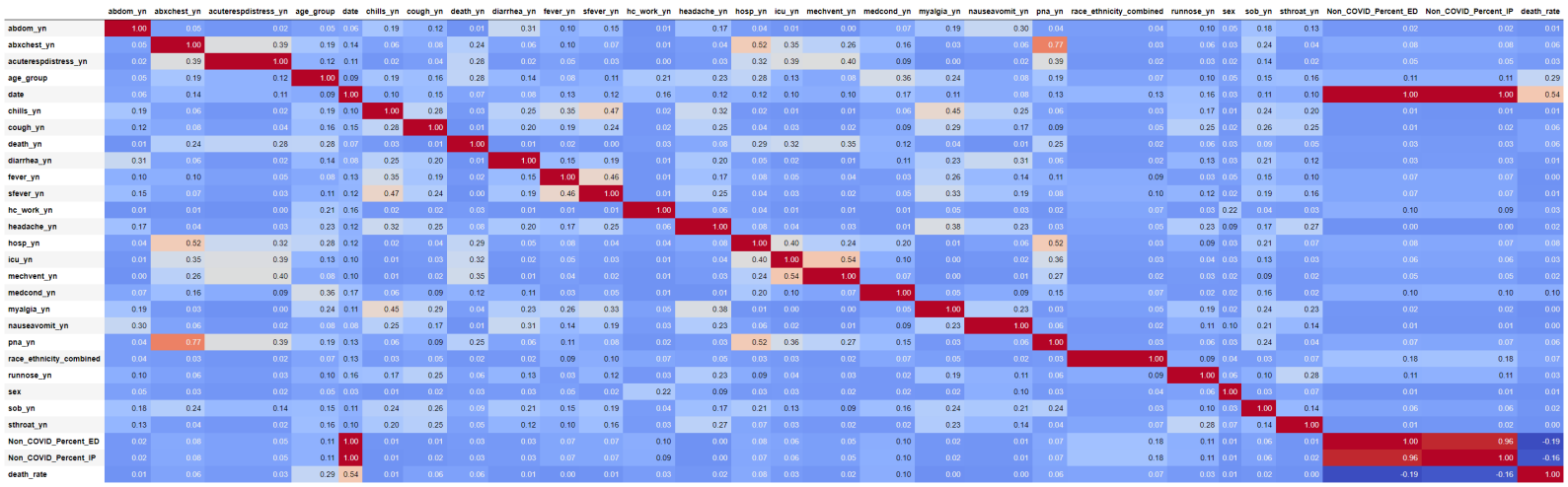
To address this limitation, the Python library named **Dython**, can be used to evaluate the associations among the covariates and plot them as a heatmap. By default, **Dython** uses Pearson's Correlation, defined in Section 4.1.1, to calculate the association between numeric features.

In our analysis, we utilize Pearson's correlation to determine the association between two numeric features, the correlation ratio to compute the associations between a categorical feature and a numeric feature, and Cramér's V to evaluate the strength of association between two categorical variables.

In Figure 5.1(a), we display 14 features whose association with the outcome variable Y, "death_yn", expressed in terms of Cramér's V values, is greater than 0.05, indicating a moderate level of association. In Figure 5.1(b) presents a comprehensive heatmap illustrating the relationships among all features. Notably, the highest association strength observed in this study is 0.35. Based on this analysis, we select the features with a Cramér's V value exceeding 0.05 to be included as the best feature subset. The feature "pna_yn" ($x_8^{(1)}$) shows a strong correlation with "abxchest_yn" ($x_9^{(1)}$), with a Cramér's V value about 0.8. This high correlation indicates that "pna_yn" and "abxchest_yn" are closely related to each other. Consequently, in order to reduce dimensionality and simplify the analysis, "abxchest_yn" ($x_9^{(1)}$) is excluded from the feature subset. This decision is based on the fact that "abxchest_yn" ($x_9^{(1)}$) has a lower Cramér's V value in relation to the outcome variable, Y.

	mechvent_yn	icu_yn	hosp_yn	age_group	acuterespdistress_yn	pna_yn	abxchest_yn	medcond_yn	sob_yn	headache_yn	date	runnose_yn	death_rate	sthroat_yn
death_yn	0.35	0.32	0.29	0.28	0.28	0.25	0.24	0.12	0.09	0.08	0.07	0.06	0.06	0.05

(a) *Features that has Cramér's V value* $>= 0.05$ *with the outcome variable* Y.



(b) *Associations between all features.*

Figure 5.1: *Association heat-map with Cramér's V values*

In this way, by keeping only the features that have a Cramér's V value with the outcome variable Y("death_yn") larger than 0.05, we obtain the 12 features provided by Filter Method, shown in Table 5.1 where the features are ordered by their the Cramér's V values with the outcome variable Y("death_yn"), in descending order.

Table 5.1: *Features selected by the Filter Method and their Cramér's V value with variable Y ("death_yn")*

Variable Name	Notation	Cramér's V value with y
mechvent_yn	$x_{11}^{(1)}$	0.35
icu_yn	$x_{6}^{(1)}$	0.32
hosp_yn	$x_{5}^{(1)}$	0.29
age_group	$x_{3}^{(1)}$	0.28
acuterespdistress_yn	$x_{10}^{(1)}$	0.28
pna_yn	$x_{8}^{(1)}$	0.25
medcond_yn	$x_{24}^{(1)}$	0.12
sob_yn	$x_{19}^{(1)}$	0.09
headache_yn	$x_{21}^{(1)}$	0.08
date	$x_{1}^{(1)}$	0.07
runnose_yn	$x_{16}^{(1)}$	0.06
sthroat_yn	$x_{17}^{(1)}$	0.05

5.1.3 Learning with the Wrapper Method

We now employ the Python library **mlxtend** to perform sequential feature selection for the data. Considering the binary classification nature of our task, we choose logistic regression to fit the data, which was driven by its inherent advantages in terms of interpretability, simplicity, and computational efficiency. Its suitability for handling large datasets enables us to efficiently identify the optimal subset of features. Moreover, the interpretability of logistic regression allows us to provide meaningful explanations for the selected features.

Concerning X and Y, defined in Section 5.1, we start with having no

feature in the logistic regression model, and then keep adding a feature to improve the model in each step until adding an additional variable cannot improve the model performance anymore. Macro F1 Score, defined in Section 4.1.2, is used as the evaluation criterion for the classification process since by Sadi et al. (2022), it "is the arithmetic mean of per-class F1 scores for a more appropriate measurement of model performance on class-imbalanced data." Macro F1 Score is the unweighted mean of the F1 Scores from each classification class (death or not in the study), and it can be viewed as a simple aggregation of F1 Score to all classes (Pedregosa et al., 2011).

Importantly, we note that this approach demands considerable computational resources, prompting us to focus exclusively on the *forward selection* algorithm within the wrapper method. A graphical representation, as depicted in Figure 5.2, showcases the evaluation criterion, the Macro F1 Score, plotted along the y-axis. This visualization demonstrates the influence of the number of selected features on the outcome. It is observed that the Macro F1 Score attains its peak when 20 features are chosen in the subset, shown in Table 5.2. These features are listed by the level of importance in the descending order, where the importance of a variable is determined by the level of its enhancement of the Macro F1 Score with its inclusion.

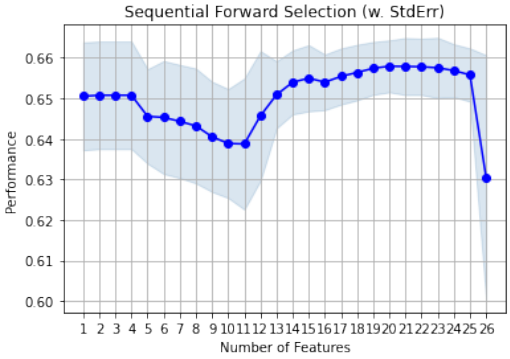


Figure 5.2: *Macro F1 Score(vertical axis) obtained by Sequential Forward Selection (with standard error shown in light blue shaded area)*

Table 5.2: *Features selected by the Wrapper Method*

Variable Name	Notation
abdom_yn	$x_{22}^{(1)}$
abxchest_yn	$x_{9}^{(1)}$
acuterespdistress_yn	$x_{10}^{(1)}$
age_group	$x_{3}^{(1)}$
chills_yn	$x_{14}^{(1)}$
cough_yn	$x_{18}^{(1)}$
fever_yn	$x_{12}^{(1)}$
headache_yn	$x_{21}^{(1)}$
hosp_yn	$x_{5}^{(1)}$
icu_yn	$x_{6}^{(1)}$
mechvent_yn	$x_{11}^{(1)}$
medcond_yn	$x_{24}^{(1)}$
myalgia_yn	$x_{15}^{(1)}$
nauseavomit_yn	$x_{20}^{(1)}$
pna_yn	$x_{8}^{(1)}$
sex	$x_{2}^{(1)}$
sob_yn	$x_{19}^{(1)}$
sthroat_yn	$x_{17}^{(1)}$
Non_COVID_Percent_ED	$x_{1}^{(2)}$
Non_COVID_Percent_IP	$x_{2}^{(2)}$

5.1.4 Learning with the Embedded Method

To implement the embedded method in our study, we use **SelectFromModel** module provided by **Scikit-Learn**, since this library can attach importance to each feature with different attributes such as "coef_", "feature_importances_" and so on(Pedregosa et al., 2011). This package enables us to evaluate the contribution of each feature. To achieve feature selection, we employ logistic regression as a linear classifier with the Lasso (L1) penalization. This approach enables us to select only the predictors with non-zero coefficients, effectively reducing the overall dimensionality of the dataset.

For better visualization and understanding, in Figure 5.3 we depict the significance of each feature. The plot showcases the absolute values of the coefficients plotted against their respective feature names. The magnitude of the coefficient reflects the contribution of the feature to the prediction, where larger coefficients indicate greater importance. If using 0.1 as a threshold, features with coefficients below 0.1 are excluded. As a result, we retain 17 predictors as the best feature subset, shown in Table 5.3. These features are ordered by the magnitude of coefficients in descending order.

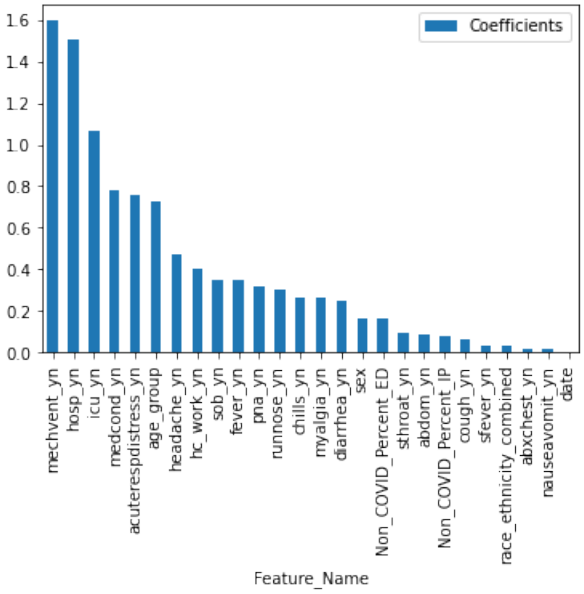


Figure 5.3: *The graph of feature importance, shown by the magnitude of the coefficient versus the feature's name*

Table 5.3: *Features selected by the Embedded Method and their coefficients with variable Y ("death_yn")*

Variable Name	Notation	Coefficients with y
mechvent_yn	$x_{11}^{(1)}$	1.5982
hosp_yn	$x_{5}^{(1)}$	1.5061
icu_yn	$x_{6}^{(1)}$	1.0668
medcond_yn	$x_{24}^{(1)}$	0.7798
acuterespdistress_yn	$x_{10}^{(1)}$	0.7601
age_group	$x_{3}^{(1)}$	0.7230
headache_yn	$x_{21}^{(1)}$	0.4693
hc_work_yn	$x_{7}^{(1)}$	0.4008
sob_yn	$x_{19}^{(1)}$	0.3501
fever_yn	$x_{12}^{(1)}$	0.3490
pna_yn	$x_{8}^{(1)}$	0.3141
runnose_yn	$x_{16}^{(1)}$	0.3031
chills_yn	$x_{14}^{(1)}$	0.2616
myalgia_yn	$x_{15}^{(1)}$	0.2611
diarrhea_yn	$x_{23}^{(1)}$	0.2459
sex	$x_{2}^{(1)}$	0.1638
Non_COVID_Percent_ED	$x_{1}^{(2)}$	0.1634

5.1.5 Results with Intersection and Union

Examining the best features selected by the Filter, Wrapper, and Embedded Methods, we consider two special sets. The first set includes 9 commonly selected features of the three methods, shown in Table 5.4 where the features are listed by the subscript order in the variables.

Table 5.4: *Features selected by the Intersection*

Variable Name	Notation
age_group	$x_3^{(1)}$
hosp_yn	$x_5^{(1)}$)
icu_yn	$x_6^{(1)}$
pna_yn	$x_8^{(1)}$
acuterespdistress_yn	$x_{10}^{(1)}$
mechvent_yn	$x_{11}^{(1)}$
sob_yn	$x_{19}^{(1)}$
headache_yn	$x_{21}^{(1)}$
medcond_yn	$x_{24}^{(1)}$

In contrast, the second set contains the union of the best feature subsets chosen by the Filter, Wrapper, and Embedded Methods. This set contains the following 24 predictors listed by the subscript order, as shown in Table 5.5.

Table 5.5: *Features selected by the Union*

Variable Name	Notation
date	$x_1^{(1)}$
sex	$x_2^{(1)}$
age_group	$x_3^{(1)}$
hosp_yn	$x_5^{(1)}$
icu_yn	$x_6^{(1)}$
hc_work_yn	$x_7^{(1)}$
pna_yn	$x_8^{(1)}$
abxchest_yn	$x_9^{(1)}$
acuterespdistress_yn	$x_{10}^{(1)}$
mechvent_yn	$x_{11}^{(1)}$
fever_yn	$x_{12}^{(1)}$
chills_yn	$x_{14}^{(1)}$
myalgia_yn	$x_{15}^{(1)}$
runnose_yn	$x_{16}^{(1)}$
sthroat_yn	$x_{17}^{(1)}$
cough_yn	$x_{18}^{(1)}$
sob_yn	$x_{19}^{(1)}$
nauseavomit_yn	$x_{20}^{(1)}$
headache_yn	$x_{21}^{(1)}$
abdom_yn	$x_{22}^{(1)}$
diarrhea_yn	$x_{23}^{(1)}$
medcond_yn	$x_{24}^{(1)}$
Non_COVID_Percent_ED	$x_1^{(2)}$
Non_COVID_Percent_IP	$x_2^{(2)}$

5.2 Data Balancing

Imbalanced data refers to datasets in which instances belonging to one class significantly outnumber those in the other class, posing a challenge in using traditional classification algorithms. As commented by Sun et al. (2009) in a review article, standard classifier learning algorithms typically assume a balanced class distribution and equal misclassification costs, and hence is hindered in the presence of imbalanced class distributions. To improve the classification performance on imbalanced dataset, Sun et al. (2009) compared four different types of strategies: resampling techniques, cost-sensitive learning, boosting, and adapting existing algorithms.

In our study here, we focus on two strategies. The first strategy involves the resampling scheme, to be discussed from Sections 5.2.1 to 5.2.3. In the second strategy, we assess the performance of cost-sensitive classification models which will be explained in detail in Section 5.4.

Before conducting any data balancing technique, we use the **Scikit-Learn** library to split the training set and a test set so that 20% of the original dataset is to be used to test the performance of models. The split is done randomly to ensure the training and testing sets to have the same class distribution as the original dataset. This procedure yields that the training set contains 2778 death cases and 191441 alive cases, presenting imbalance in the outcome variable.

5.2.1 Over-Sampling on Training Set

To address the class imbalance in the dataset, we employ the over-sampling method from the **imbalance-learn** library. In doing so, there is, however, a potential risk of overfitting associated with oversampling the minority class(Sun et al., 2009). Therefore, to simultaneously balance the training dataset and mitigate the risk of overfitting caused by random over-sampling, the *synthetic minority over-sampling technique (SMOTE)* is employed.

SMOTE is a sampling method to make the minority class be over-sampled by creating "synthetic" examples rather than by over-sampling with replacement (Chawla et al., 2002). This approach not only generates additional training data but also circumvents the issue of duplicated rows that may arise from simple random over-sampling.

This over-sampling yields 191441 observations for each class, Alive or Death, to form a balanced training set. This balanced training set will be used for both traditional machine learning models and artificial neural networks, to be reported in Sections 5.3 and 5.4.

5.2.2 Under-Sampling on Testing Set

A persistent issue arises in the evaluation of the models on the testing set, irrespective of the approach taken. The models consistently exhibit low precision scores, accompanied by either high recall or high accuracy scores, even when the "class_weight" parameter is set to be "balanced".

To overcome this issue, we employ the random under-sampling method, a useful technique to address the problem of class imbalance (X.-Y. Liu, Wu,

and Zhou, 2008). Under-sampling entails reducing the number of instances in the majority class, regarded as the over-represented class, to match the number of instances in the minority class, treated as the under-represented class. This approach is carried out by randomly selecting a subset of the majority class instances to make it equal in size to the minority class. This approach guarantees an equal representation of each class in the testing set. Consequently, in this study, all testing sets will be balanced prior to model evaluation, with the "Alive" and "Death" classes each containing 694 records.

5.3 Learning with Machine Learning Models

This section focuses on developing effective models to classify the data through the utilization of various algorithms and techniques. The study employs six traditional machine learning algorithms: Support Vector Machine (SVM), Decision Tree, Random Forest, Logistic Regression, K-Nearest Neighbor (KNN), and Naïve Bayes.

With the datasets appropriately balanced as described in Section 5.2, we construct a model using each of these six learning algorithms in subsequent subsections. All five best feature subsets identified in Section 5.1, (i.e., those displayed in Tables 5.1 -5.5), will be evaluated using these algorithms to compare their performance, where hyperparameters are set to their default values specified in the respective software package, except for logistic regression.

The classification reports generated by **Scikit-Learn** will be presented in the following sections, where "0.0" corresponds to the "Non-Death/Alive" class and "1.0" corresponds to the "Death" class; the first column of each

table shows the feature subset names and the subsequent columns display the scores for various evaluation metrics: precision, recall, and F1-score. The best feature subset with the highest score is highlighted in bold font.

5.3.1 Learning with the Support Vector Machine

As commented by Batuwita and Palade (2013), the support vector machine (SVM) algorithm is a widely used machine learning technique due to several reasons, including a robust mathematical foundation, strong generalization capability, and the capacity to discover global and nonlinear classification solutions. Despite its effectiveness in handling balanced datasets, SVMs may yield unsatisfactory results when confronted with imbalanced datasets (e.g., X. Liu et al., 2021). There are various methods to mitigate this challenge, such as pre-balancing the dataset prior to training (as mentioned in Section 5.2) and assigning distinct error costs to different classes (Cost-Sensitive SVM in Section 4.3.1).

In our study, we face the challenge of dealing with an imbalanced dataset, leading us to investigate the efficacy of these strategies described in Section 4.3.1 and 5.2. The first approach involves pre-balancing the dataset before the training phase, and the second strategy entails assigning unique error costs to different classes in order to optimize the performance of SVM.

Furthermore, in addition to implementing the Cost-Sensitive SVM, our investigation encompasses the incorporation of Cost-Sensitive Decision Tree and Cost-Sensitive Logistic Regression, as mentioned in Sections 4.3.1-4.3.3.

Support Vector Classification (SVC) is useful to perform classification for many datasets, but with over tens of thousands of samples, LinearSVC, doc-

umented in **Scikit-Learn**, is suggested to be used (Pedregosa et al., 2011). The difference between LinearSVC and SVC is that the former has more flexibility than the latter one in the choice of penalization or regularization methods to fit a large sample set better and faster.

The Support Vector Machine (SVM) models are trained using the default hyperparameter values of LinearSVC in the **Scikit-Learn** library. Table 5.6 reports the results of applying SVM models to the integrated dataset described in Section 3.5.1, in combination with the use of each of the five different feature subsets obtained from using the feature selection methods described in Section 5.1. The results indicate that applying SVM to the intersection feature subset demonstrates the best performance among the five subsets.

Table 5.6: *The results obtained from applying the Support Vector Machine (SVM) to the integrated dataset in Section 3.5.1, with the use of one of the five best feature subsets obtained in Section 5.1*

Features Selection Method in Section 5.1	Accuracy	Status	Precision	Recall	F1-score
filter method	0.8696	alive	0.8480	0.9006	0.8735
		death	0.8940	0.8386	0.8654
wrapper method	0.8235	alive	0.7657	0.9323	0.8408
		death	0.9134	0.7147	0.8019
embedded method	0.8610	alive	0.8112	0.9409	0.8712
		death	0.9297	0.7810	0.8489
intersection method	**0.8927**	alive	0.8833	0.9049	0.8940
		death	0.9025	0.8804	0.8913
union method	0.8163	alive	0.7532	0.9409	0.8366
		death	0.9213	0.6916	0.7901

5.3.2 Learning with the Decision Tree

We utilize the default hyperparameter values from the **Scikit-Learn** library to fit the decision tree models to the integrated dataset in Section 3.5.1, with each of five identified feature subsets obtained in Section 5.1. Table 5.7 below displays the performances of decision tree models applied to each of the five different feature subsets. Notably, the application to the intersection feature subset achieved the highest accuracy, indicating its superior performance to the applications to other feature subsets.

Table 5.7: *The results obtained from applying the Decision Tree models to the integrated dataset in Section 3.5.1, with the use of one of the five best feature subsets obtained in Section 5.1*

Features Selection Method in Section 5.1	Accuracy	Status	Precision	Recall	F1-score
filter method	0.7356	alive	0.6614	0.9654	0.7850
		death	0.9360	0.5058	0.6567
wrapper method	0.6938	alive	0.6237	0.9769	0.7614
		death	0.9468	0.4107	0.5729
embedded method	0.7104	alive	0.6385	0.9697	0.7700
		death	0.9371	0.4510	0.6089
intersection method	**0.8912**	alive	0.8596	0.9352	0.8958
		death	0.9289	0.8473	0.8862
union method	0.6787	alive	0.6011	0.9827	0.7536
		death	0.9559	0.3746	0.5383

5.3.3 Learning with the Random Forest

We employ the Random Forest classifier from the **Scikit-Learn** library with default hyperparameter values to train various Random Forest models

on the integrated dataset in Section 3.5.1 by using the five feature subsets described in Section 5.1. The results are presented in Table 5.8, and again, the application to the intersection feature subset exhibited the highest accuracy.

Table 5.8: *The results obtained from applying the Random Forest models to the integrated dataset in Section 3.5.1, with the use of one of the five best feature subsets obtained in Section 5.1*

Features Selection Method in Section 5.1	Accuracy	Status	Precision	Recall	F1-score
filter method	0.7341	alive	0.6607	0.9625	0.7836
		death	0.9310	0.5058	0.6555
wrapper method	0.7104	alive	0.6362	0.9827	0.7724
		death	0.9620	0.4380	0.6020
embedded method	0.7197	alive	0.6454	0.9755	0.7768
		death	0.9499	0.4640	0.6234
intersection method	**0.8912**	alive	0.8596	0.9352	0.8958
		death	0.9289	0.8473	0.8862
union method	0.6873	alive	0.6171	0.9870	0.7594
		death	0.9676	0.3876	0.5535

5.3.4 Learning with the Logistic Regression

In this study, custom parameter values are utilized for the **Scikit-Learn LogisticRegression** algorithm to address the "failed to converge" error caused by the large dataset size. The "saga" solver was employed, and the maximum number of iterations was increased to 5000. Table 5.9 summarizes the performances of applying logistic regression models to the integrated dataset in Section 3.5.1 with five different feature subsets described in Section 5.1. Among the five different feature subsets considered, the application

to the intersection feature subset emerged as the top performer, achieving the highest accuracy.

Table 5.9: *The results obtained from applying the Logistic Regression models to the integrated dataset in Section 3.5.1, with the use of one of the five best feature subsets obtained in Section 5.1*

Features Selection Method in Section 5.1	Accuracy	Status	Precision	Recall	F1-score
filter method	0.9107	alive	0.9014	0.9222	0.9117
		death	0.9204	0.8991	0.9096
wrapper method	0.9085	alive	0.8943	0.9265	0.9101
		death	0.9238	0.8905	0.9068
embedded method	0.9049	alive	0.8849	0.9308	0.9073
		death	0.9271	0.8790	0.9024
intersection method	**0.9114**	alive	0.9015	0.9236	0.9125
		death	0.9217	0.8991	0.9103
union method	0.9035	alive	0.8867	0.9251	0.9055
		death	0.9217	0.8818	0.9013

5.3.5 Learning with the K-Nearest Neighbor

We use the default hyperparameter values (such as: “metric”: “minkowski”; “n_neighbors”: 5; “weights”: “uniform”) of the K-Nearest Neighbour (KNN) classifier from the **Scikit-Learn** library to train our KNN models. Table 5.10 presents the results of applying K-Nearest Neighbour(KNN) models to the integrated dataset in Section 3.5.1 with 5 different feature subsets described in Section 5.1. Differed from the results in Sections 5.3.1-5.3.4, the application to the integrated dataset with the feature subset obtained from the Filter Method achieves the best accuracy of 0.76.

Table 5.10: *The results obtained from applying the K-Nearest Neighbour(KNN) models to the integrated dataset in Section 3.5.1, with the use of one of the five best feature subsets obtained in Section 5.1*

Features Selection Method in Section 5.1	Accuracy	Status	Precision	Recall	F1-score
filter method	**0.7572**	alive	0.6798	0.9726	0.8002
		death	0.9519	0.5418	0.6905
wrapper method	0.7399	alive)	0.6634	0.9741	0.7893
		death	0.9512	0.5058	0.6604
embedded method	0.7543	alive	0.6785	0.9669	0.7974
		death	0.9424	0.5418	0.6880
intersection method	0.6037	alive	0.5581	0.9971	0.7156
		death	0.9865	0.2104	0.3468
union method	0.7529	alive	0.6757	0.9726	0.7974
		death	0.9512	0.5331	0.6833

5.3.6 Learning with the Gaussian Naïve Bayes

We train Naïve Bayes models using the **GaussianNB** classifier from the **Scikit-Learn** library with default hyperparameter values. Table 5.11 presents the results of applying Naïve Bayes models to the integrated dataset in Section 3.5.1 with five different feature subsets described in Section 5.1. The three feature subsets, namely "filter method", "wrapper method", and "union method", achieved similar performance, with accuracy around 0.82-0.85. The "embedded method" subset performed slightly better, with an accuracy of 0.85. The "intersection method" subset also exhibited good performance, with an accuracy of 0.84. These results indicate that the Naïve Bayes models, particularly when trained with the "embedded method" sub-

set, can effectively do classification.

Table 5.11: *The results obtained from applying the Naïve Bayes models to the integrated dataset in Section 3.5.1, with the use of one of the five best feature subsets obtained in Section 5.1*

Features Selection Method in Section 5.1	Accuracy	Status	Precision	Recall	F1-score
filter method	0.8264	alive	0.7559	0.9640	0.8474
		death	0.9503	0.6888	0.1987
wrapper method	0.8249	alive	0.7554	0.9611	0.8459
		death	0.9465	0.6888	0.7973
embedded method	**0.8487**	alive	0.7834	0.9640	0.8643
		death	0.9532	0.7334	0.8290
intersection method	0.8386	alive	0.7695	0.9669	0.8570
		death	0.9554	0.7104	0.8149
union method	0.8509	alive	0.7868	0.9625	0.8658
		death	0.9518	0.7392	0.8321

5.3.7 Learning with the Artificial Neural Networks

In this section, we develop two artificial neural network models to perform binary classification, predicting whether a confirmed COVID-19 patient will survive or not. The balanced training and testing datasets discussed in Section 5.2 are utilized for training, fitting, and evaluating the model performance. We apply the five feature subsets obtained in Section 5.1 to these two networks in order to compare their accuracy rates and overall performance. By employing these models, we aim to assess their effectiveness in accurately classifying the survival outcomes of patients and determine which feature subsets contribute most significantly to this classification task.

5.3.7.1 Learning with the Multi-Layers Perceptron

We construct the network using the **MLPClassifier** from the **Scikit-Learn** library. The network architecture consists of two hidden layers, with the first layer comprising five neurons and the second layer consisting of two neurons. To perform the binary classification task, we utilize the logistic sigmoid activation function for the hidden layers. The "learning_rate" parameter was set to be "adaptive" to optimize the model's performance. To ensure convergence to be reached, we increase the maximum number of iterations to 5000. Figure 5.4 below illustrates a sample structure of our MLP network built with the nine input variables in the intersection feature set described in Section 5.1.5.

Table 5.12 presents the performances of applying MLP models to the integrated dataset in Section 3.5.1 with five feature subsets described in Section 5.1. Each subset is evaluated based on precision, recall, and F1-score. The model trained on the intersection feature subset performs the best, achieving an accuracy about 91%. The model trained on the feature subset obtained from the Filter Method performed as the second-best model, with slightly lower precision and recall scores compared to the best-performing model. The model applied to the integrated dataset with the "wrapper method" subset and "embedded method" attained an accuracy of 89% and 90% respectively. The model integrated dataset with the "union method" subset had the lowest accuracy 88%. Overall, the MLP models demonstrate consistent performance across the five feature subsets, with the intersection set showing the highest accuracy.

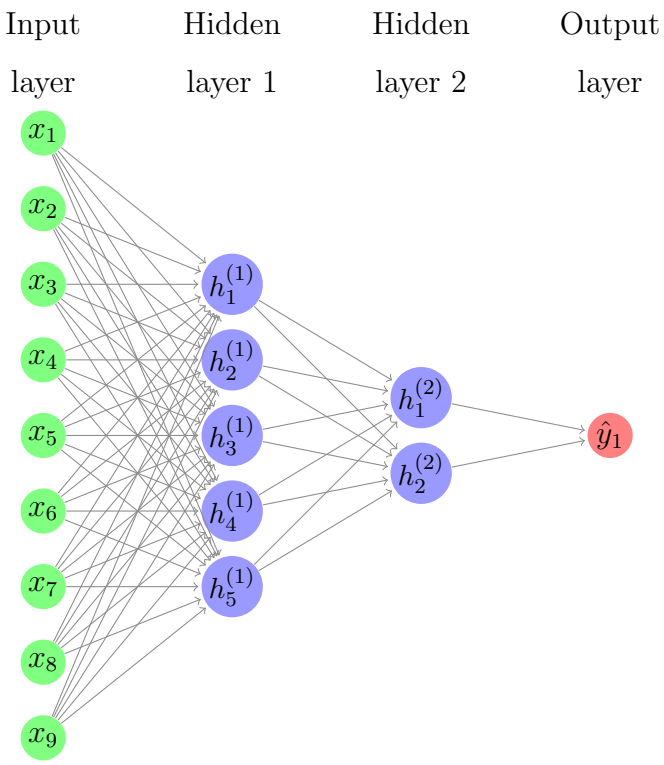


Figure 5.4: *A sample MLP network architecture with nine input variables and two hidden layers*

Table 5.12: *The results obtained from applying the Multi-Layers perceptron (MLP) models to the integrated dataset in Section 3.5.1, with the use of one of the five best feature subsets obtained in Section 5.1*

Features Selection Method in Section 5.1	Accuracy	Status	Precision	Recall	F1-score
filter method	0.9085	alive	0.8921	0.9294	0.9104
		death	0.9263	0.8876	0.9065
wrapper method	0.8847	alive	0.8532	0.9294	0.8897
		death	0.9225	0.8401	0.8793
embedded method	0.9027	alive	0.8752	0.9395	0.9062
		death	0.9347	0.8660	0.8990
intersection method	**0.9092**	alive	0.8944	0.9280	0.9109
		death	0.9251	0.8905	0.9075
union method	0.8818	alive	0.8478	0.9308	0.8874
		death	0.9233	0.8329	0.8758

5.3.7.2 Learning with the Radial Basis Function Network

In Python, there are multiple ways to construct an RBFN. For example, one option is to utilize the **Keras** library, which provides flexibility in customizing the hidden layer and integrating it into a sequential model along with input and output layers. In contrast, in this study, we use another method which only requires the **Numpy** library.

The RBFN model in this study has only three layers: an input layer, a hidden layer, and an output layer. The hidden layer has 6 neurons and we use a Gaussian function as the activation function for these neurons. Each neuron in the hidden layer represents a centre, and the centres will be determined by using K-mean clustering; therefore, the initialized model is built with

6 centres. The weights connecting the input vectors and hidden neurons depend on the space or distance between the vectors and the centres, so these weights are predetermined once the centres are set. The weights connecting hidden neurons and the output are determined to train the network. and these weights can be computed by using pseudo inverse since the RBFN can be represented as matrix multiplications (Ghosh and Nag, 2001).

Let g_{ij} denote the output of jth neuron for ith input vector, and let w_{ij} denote the weight connecting ith output neuron to jth hidden neuron. Define G and W to be a matrix with the (i, j) element g_{ij} and w_{ij}, respectively. Let T represent a column vector with the ith element recording the target variable's value for the ith training vector. Then

$$GW = T,$$

suggesting that W can be computed by using pseudo inverse:

$$W = (G^\top G)^{-1} G^\top T.$$

Once we have the weights, we can apply them on the testing set and obtain the predictions by matrix multiplications and analyze the performance.

Table 5.13 presents the performances of applying Radial Basis Function Network (RBFN) models to the integrated dataset(Section 3.5.1) with five feature subsets described in Section 5.1. The application to the features subset obtained from the "embedded method" achieved the highest accuracy of 0.87, followed closely by that for the intersection set with an accuracy of 0.86.

Table 5.13: *The results obtained from applying the Radial Basis Function Network (RBFN) models to the integrated dataset in Section 3.5.1, with the use of one of the five best feature subsets obtained in Section 5.1*

Features Selection Method in Section 5.1	Accuracy	Status	Precision	Recall	F1-score
filter method	0.8228	alive	0.7894	0.8804	0.8324
		death	0.8648	0.7651	0.8119
wrapper method	0.8365	alive	0.8186	0.8646	0.8409
		death	0.8565	0.8084	0.8317
embedded method	**0.8710**	alive	0.8503	0.9006	0.8747
		death	0.8943	0.8415	0.8671
intersection method	0.8588	alive	0.8630	0.8530	0.8580
		death	0.8547	0.8646	0.8596
union method	0.8386	alive	0.7982	0.9063	0.8489
		death	0.8917	0.7709	0.8269

5.4 Learning with the Cost-Sensitive Classification Models

To address possible imbalance in data, we now apply three cost-sensitive learning algorithms: Cost-Sensitive SVM, Cost-Sensitive Decision Tree, and Cost-Sensitive Logistic Regression for binary classification to the integrated dataset in Section 3.5.1, along with five feature subsets described in Section 5.1. These algorithms offer a distinct advantage with the inclusion of the "class_weight" parameter in the widely used **Scikit-Learn** library. The **Scikit-Learn** library utilizes the inverse class distribution as the cost matrix for these three cost-sensitive classifiers. It adjusts weights inversely propor-

tional to the frequencies of classes in the input data. In this way, we can smoothly convert an imbalanced classification challenge into a balanced one by setting the inverted class weights as the constant cost matrix for cost-sensitive classifiers. Consequently, evaluation metrics such as accuracy and F1 score can be employed to assess the performance of cost-sensitive models.

According to the **Scikit-Learn** documentations of Pedregosa et al. (2011), the "class_weight" parameter is set to "balanced" before train and fit these models. The imbalanced training set is used in this section to allow the models to assign different class weights.

The initial training set and testing set, representing the datasets before applying any over-sampling or under-sampling techniques, are utilized in this study for the Cost-Sensitive classification models.

5.4.1 Learning with the Cost-Sensitive SVM

We utilize the **Scikit-Learn** library to train a Cost-Sensitive SVM model with default hyperparameter values. Table 5.14 presents the results of applying the Cost-Sensitive SVM models on the integrated dataset in Section 3.5.1 using five different feature subsets described in Section 5.1. The application to the integrated dataset with the intersection feature subset stands out with an overall accuracy of 0.92.

Table 5.14: *The results obtained from applying the Cost-Sensitive SVM models to the integrated dataset in Section 3.5.1, with the use of one of the five best feature subsets obtained in Section 5.1*

Features Selection Method in Section 5.1	Accuracy	Status	Precision	Recall	F1-score
filter method	0.9215	alive	0.9270	0.9150	0.9210
		death	0.9161	0.9280	0.9220
wrapper method	0.8905	alive	0.8623	0.9294	0.8946
		death	0.9234	0.8516	0.8861
embedded method	0.9078	alive	0.8953	0.9236	0.9092
		death	0.9211	0.8919	0.9063
intersection method	**0.9229**	alive	0.9272	0.9179	0.9225
		death	0.9187	0.9280	0.9233
union method	0.8811	alive	0.8485	0.9280	0.8864
		death	0.9205	0.8343	0.8864

5.4.2 Learning with the Cost-Sensitive Decision Tree

We conduct experiments using the **Scikit-Learn** library to train Cost-Sensitive Decision Tree models with default hyperparameter values. The performances of these models was evaluated by their application to the integrated dataset in Section 3.5.1 with five different subsets of features described in Section 5.1. Table 5.15 compares the performances of the application of Cost-Sensitive Decision Tree models to the integrated dataset with five different feature subsets. Notably, the use of the intersection feature subset outperformed other subsets, achieving the highest accuracy of 0.91.

Table 5.15: *The results obtained from applying the Cost-Sensitive Decision Tree models to the integrated dataset in Section 3.5.1, with the use of one of the five best feature subsets obtained in Section 5.1*

Features Selection Method in Section 5.1	Accuracy	Status	Precision	Recall	F1-score
filter method	0.6571	alive	0.5951	0.9827	0.7413
		death	0.9504	0.3314	0.4915
wrapper method	0.6563	alive	0.5949	0.9798	0.7403
		death	0.9429	0.3329	0.4920
embedded method	0.6700	alive	0.6052	0.9784	0.7478
		death	0.9436	0.3617	0.5229
intersection method	**0.9078**	alive	0.9031	0.9135	0.9083
		death	0.925	0.9020	0.9072
union method	0.6326	alive	0.5769	0.9942	0.7302
		death	0.9792	0.2709	0.4244

5.4.3 Learning with the Cost-Sensitive Logistic Regression

In this study, we train a Cost-Sensitive logistic regression model using the **Scikit-Learn** library. The advantage of using this library is the availability of the "class_weight" parameter, which enhances the performance of the logistic regression algorithm on imbalanced datasets. Furthermore, we evaluate the performances of Cost-Sensitive logistic regression models by using the five different feature subsets described in Section 5.1 on the integrated dataset(Section 3.5.1). In Table 5.16, we can observe the precision, recall, and F1-score for each class within the subsets. It is worth noting that the application of the intersection feature subset excelled in terms of accuracy

achieving a remarkable value of 0.91. On the other hand, the remaining subsets exhibited diverse performance across the metrics.

Table 5.16: *The results obtained from applying the Cost-Sensitive logistic regression models to the integrated dataset in Section 3.5.1, with the use of one of the five best feature subsets obtained in Section 5.1*

Features Selection Method in Section 5.1	Accuracy	Status	Precision	Recall	F1-score
filter method	0.8948	alive	0.9463	0.8372	0.8884
		death	0.8540	0.9524	0.9005
wrapper method	0.7925	alive	0.7128	0.9798	0.8252
		death	0.9677	0.6052	0.7447
embedded method	0.9020	alive	0.9020	0.9020	0.9020
		death	0.9020	0.9020	0.9020
intersection method	**0.9164**	alive	0.9176	0.9150	0.9163
		death	0.9152	0.9179	0.9165
union method	0.9006	alive	0.8971	0.9049	0.9010
		death	0.9041	0.8963	0.9001

5.5 Summary

This chapter explores the applications of various methods or models to learn the integrated dataset in Section 3.5.1, with five feature subsets described in Section 5.1. The logistic regression model utilizing the intersection set of predictors stands out with the best performance. The intersection subset consistently proves to be the most effective set of features in four out of the six models we considered. Moreover, after balancing the training and testing sets, all models exhibit robust performance, achieving accuracy, pre-

cision, and recall levels of around 90% in most cases, except for the KNN model.

The intersection set is also the best feature subset for all three cost-sensitive models. These models exhibit comparable performance to traditional machine learning models, achieving accuracy, precision, and recall levels of 90% approximately. These findings suggest that employing the "class_weight" parameter on the imbalanced training set can yield comparable results to the approach of balancing the dataset prior to training and fitting. Hence, utilizing the class-weighting technique during model training proves to be an effective strategy for addressing class imbalance and achieving robust classification performance.

The ANNs described in Section 5.5 exhibit comparable performances to the aforementioned models, with accuracy, precision, and recall scores hovering around 90%. While the MLP model requires slightly more time for fitting and training compared to the RBFN, it demonstrates similar time duration to the cost-sensitive models.

In summary, the analyses here using different classification methods consistently demonstrate that the analyses based on the intersection set of features yield the best results, followed closely by those for the embedded set. This finding indicates that the prediction of COVID-19-related patient mortality maybe likely done well by utilizing the nine key features described in the intersection feature set in Section 5.1.5. The predictions based on this subset of features are fairly robust, highlighting their significance in clinical decision-making.

5.6 Statistical Analysis

As discussed in Section 5.6, the intersection set of features, as discussed in Section 5.1.5, has demonstrated its effectiveness in seven out of the nine machine learning models we employed. Given that the logistic model exhibited superior performance, here we further conduct statistical analysis by fitting the logistic model to the integrated dataset in Section 3.5.1, with the features determined by the intersection method in Section 5.1.5. This analysis is done by implementing the R package **stats** and using the "glm" function to fit the logistic model, which yields point estimates, standard errors, and P-values corresponding to the coefficients of each of the nine features. The results are reported in Table 5.17.

Table 5.17 reveals significant associations between various covariates and the binary outcome variable, "death_y" (Y). Notably, the intercept is negative and highly significant, shown by the P-value $< 2 \times 10^{-16}$, suggesting that in the absence of any predictor variables, the probability of death is exceedingly low. Among the predictor variables, "pna_yn" ($x_8^{(1)}$) exhibits a near-zero coefficient (Estimate = -0.02074), and is not statistically significant (P-value = 0.399). In contrast, all other predictor variables exhibit strong statistical significance, as indicated by their extremely low P-values. Variables "age_group" ($x_3^{(1)}$), "hosp_yn" ($x_5^{(1)}$), "icu_yn" ($x_6^{(1)}$), "acuterespdistress_yn" ($x_{10}^{(1)}$), "mechvent_yn" ($x_{11}^{(1)}$), "sob_yn" ($x_{19}^{(1)}$) and "medcond_yn" ($x_{24}^{(1)}$) all exhibit positive effects on affecting the outcome, suggesting that these factors contribute to a higher probability of the death. Surprisingly, binary variable "headache_yn" ($x_{21}^{(1)}$), coded as 0 if no headache and 1 otherwise, has a negative effect on influencing the outcome, showing that individuals who report

Table 5.17: *Statistical analysis of the integrated dataset with the intersection feature subset under the logistic regression model*

Variable	**Estimate**	**Std. Error**	**P-value**
(Intercept)	-10.6405	0.0501	$< 2 \times 10^{-16}$
age_group ($x_3^{(1)}$)	10.8505	0.0528	$< 2 \times 10^{-16}$
hosp_yn ($x_5^{(1)}$)	2.1537	0.0188	$< 2 \times 10^{-16}$
icu_yn ($x_6^{(1)}$)	0.7278	0.0378	$< 2 \times 10^{-16}$
pna_yn ($x_8^{(1)}$)	-0.0207	0.0246	0.399
acuterespdistress_yn ($x_{10}^{(1)}$)	0.2374	0.0411	7.88×10^{-9}
mechvent_yn ($x_{11}^{(1)}$)	2.8978	0.0662	$< 2 \times 10^{-16}$
sob_yn ($x_{19}^{(1)}$)	0.4681	0.0169	$< 2 \times 10^{-16}$
headache_yn ($x_{21}^{(1)}$)	-0.6152	0.0157	$< 2 \times 10^{-16}$
medcond_yn ($x_{24}^{(1)}$)	1.7842	0.0212	$< 2 \times 10^{-16}$

headaches is less likely to increase the probability of death compared to those without headaches.

Chapter 6

COVID-19 Variant of Concerns

In this section, we further perform an in-depth analysis of the five distinct variants of COVID-19, as classified by the World Health Organization (WHO). Utilizing the explorations in Chapter 5 which are summarized in Section 5.6, here we consider only the intersection feature subset by using the logistic regression model, cost-sensitive SVM, and MLP models for each Variant of Concerns(VOC) dataset defined in Section 3.5.2.

Given the varying sizes of each VOCs dataset and the presence of datasets with limited death records, we employ a balancing approach for both the training sets and testing sets. We balance the training sets using the Synthetic Minority Over-sampling Technique (SMOTE) to address potential overfitting. This method helps alleviate the risk of overemphasizing the majority class during model training. In addition, for the testing sets, we implement random oversampling to counteract scenarios where the test set may have an insufficient number of death samples. Specifically, we apply the simple random oversampling technique in Python, using the **Scikit-Learn**

library. This procedure aims to ensure a more representative assessment of the model's performance. By addressing potential imbalances, this approach contributes to a more accurate evaluation of the model's proficiency in predicting mortality, especially in situations with a limited number of death instances in the test set.

We present a comprehensive classification report for each section, offering a detailed evaluation of the model's performance. The model with the best performance is highlighted by presenting its name in bold font.

6.1 Outbreak

The "Outbreak" period is defined as all cases confirmed before 2020-12-18, during which no significant variant of concern was published by WHO. In this dataset, there are no records for vaccination status and corresponding death rates. Consequently, the "death_rate"($x_1^{(3)}$) column is simply omitted.

Table 6.1 presents the performances of applying logistic regression, cost-sensitive SVM, and MLP models to the Outbreak dataset described in Section 3.5.2 using the intersection feature subsets described in Section 5.1.5. All these three models demonstrate similar performance.

Table 6.1: *The results obtained from applying various models to the Outbreak dataset in Section 3.5.2 using the intersection feature subset obtained in Section 5.1.5*

Models	Accuracy	Status	Precision	Recall	F1-score
logistic regression	0.9185	alive	0.9295	0.9056	0.9174
		death	0.9080	0.9313	0.9195
cost-sensitive SVM	0.9183	alive	0.9265	0.9086	0.9175
		death	0.9103	0.9279	0.9190
MLP	0.9191	alive	0.9289	0.9077	0.9182
		death	0.9097	0.9305	0.9200

6.2 AlphaBeta

The "AlphaBeta" period is defined as all cases confirmed between 2020-12-18 and 2021-01-11, during which two WHO-published variants of concerns: Alpha and Beta, were present. In this dataset, there are no records for vaccination status and corresponding death rates. Therefore, the "death_rate"($x_1^{(3)}$) column is also excluded.

Table 6.2 shows the performances of applying logistic regression, cost-sensitive SVM, and MLP models to the AlphaBeta dataset described in Section 3.5.2 using the intersection feature subsets described in Section 5.1.5. In this case, both the cost-sensitive SVC and MLP models exhibited superior performance compared to logistic regression, with slightly better results.

Table 6.2: *The results obtained from applying various models to the Alpha-Beta dataset in Section 3.5.2 using the intersection feature subset obtained in Section 5.1.5*

Models	Accuracy	Status	Precision	Recall	F1-score
logistic regression	0.9064	alive	0.9167	0.8940	0.9052
		death	0.8966	0.9187	0.9075
cost-sensitive SVM	0.9094	alive	0.9196	0.8973	0.9083
		death	0.8997	0.9215	0.9105
MLP	0.9099	alive	0.9375	0.8784	0.9070
		death	0.8856	0.9415	0.9127

6.3 Gamma

The "Gamma" period is defined as all cases confirmed between 2021-01-11 and 2021-05-11, during which the WHO-recognized variant of concern was the Gamma variant. This dataset, as described in Section 3.5.2, contains some missing values in the "death_rate"($x_1^{(3)}$) column. To address this, the missing values are imputed using the median of this column.

Table 6.3 presents the performances of applying logistic regression, cost-sensitive SVM, and MLP models to the Gamma dataset described in Section 3.5.2 using the intersection feature subsets described in Section 5.1.5. Logistic regression outperforms the other two models with a 90% accuracy rate, surpassing the 85% accuracy achieved by the cost-sensitive SVC and the 88% accuracy achieved by the MLP.

Table 6.3: *The results obtained from applying various models to the Gamma dataset in Section 3.5.2 using the intersection feature subset obtained in Section 5.1.5*

Models	Accuracy	Status	Precision	Recall	F1-score
logistic regression	0.9047	alive	0.9056	0.9036	0.9046
		death	0.9038	0.9058	0.9048
cost-sensitive SVM	0.8533	alive	0.8105	0.9222	0.8628
		death	0.9098	0.7844	0.8425
MLP	0.8875	alive	0.8771	0.9013	0.8891
		death	0.8985	0.8738	0.8860

6.4 Delta

The "Delta" period is defined as all cases confirmed between 2021-05-11 and 2021-11-26, during which the WHO-recognized variant of concern was the Delta variant. Similar to the dataset in Section 6.4, this dataset also contains missing values in the "death_rate"($x_1^{(3)}$) column, and we use the median of this column to impute the missing values.

Table 6.4 presents the performances of applying logistic regression, cost-sensitive SVM, and MLP models to the Delta dataset described in Section 3.5.2 using the intersection feature subsets described in Section 5.1.5. During the Delta period, both the logistic regression and cost-sensitive SVC models outperformed the MLP, demonstrating slightly superior performance.

Table 6.4: *The results obtained from applying various models to the Delta dataset in Section 3.5.2 using the intersection feature subset obtained in Section 5.1.5*

Models	Accuracy	Status	Precision	Recall	F1-score
logistic regression	0.9092	alive	0.9163	0.9007	0.9084
		death	0.9023	0.9178	0.9100
cost-sensitive SVM	0.9074	alive	0.9085	0.9060	0.9073
		death	0.9063	0.9088	0.9075
MLP	0.9009	alive	0.9071	0.8932	0.9001
		death	0.8949	0.9085	0.9016

6.5 Omicron

The "Omicron" period is defined as all cases confirmed after 2021-11-26, during which the WHO-recognized variant of concern was the Omicron variant. This dataset also contains some missing values in the "death_rate"($x_1^{(3)}$) column which are imputed using the median of this column.

Table 6.5 presents the performances of applying logistic regression, cost-sensitive SVM, and MLP models to the Omicron dataset described in Section 3.5.2 using the intersection feature subsets described in Section 5.1.5. Both logistic regression and MLP models outperformed the cost-sensitive SVC model, demonstrating slightly improved performance for the data in the Omicron period.

Table 6.5: *The results obtained from applying various models to the Omicron dataset in Section 3.5.2 using the intersection feature subset obtained in Section 5.1.5*

Models	Accuracy	Status	Precision	Recall	F1-score
logistic regression	0.8938	alive	0.8605	0.9400	0.8985
		death	0.9339	0.8477	0.8887
cost-sensitive SVM	0.7764	alive	0.7077	0.9418	0.8082
		death	0.9130	0.6111	0.7322
MLP	0.8900	alive	0.8596	0.9323	0.8945
		death	0.9261	0.8477	0.8851

6.6 Summary

Based on the classification reports for each variant of concern (VOC) in the previous sections, the performance of the models using the intersection feature subset is comparable, with average performance level reaching around 90%. This underscores the significance of the selected features and their consistent impact on the accurate classification of COVID-19 outcomes across various VOCs.

6.7 Statistical Analysis

In this section, we further conduct statistical analysis on each VOC dataset, Outbreak, AlphaBeta, Gamma, Delta, and Omicron, using the intersection feature subset of predictor variables defined in Section 5.1.5 and the binary outcome variable, "death_y" (Y), for which we use the logistic

regression model to characterize their relationship. This analysis is done by implementing the R package **stats** and using the "glm" function to fit the logistic model. We report the analysis results in Table 6.6.

Across all datasets, the "Intercept" represents the baseline or reference level, the "Estimate" column provides the estimated coefficients for each predictor variable, indicating their impact on the log-odds of the outcome (in this content, death). The "Std. Error" column indicates the standard error associated with each coefficient estimate, providing a measure of the estimate's precision. The "P-value" column displays the P-value associated with each coefficient, which is used to assess the statistical significance of the predictor's contribution to the model.

Table 6.6 presents a summary of predictor significance across all datasets. In all cases, almost all predictors exhibit statistical significance, with P-values consistently below 0.05. The only exception is "acuterespdistress_yn" ($x_{10}^{(1)}$), with P-value of 0.0613 that is slightly higher than 0.05.

As shown in Table 6.6, "age_group"($x_3^{(1)}$), "hosp_yn"($x_5^{(1)}$), "icu_yn"($x_6^{(1)}$), "mechvent_yn"($x_{11}^{(1)}$), and "medcond_yn"($x_{24}^{(1)}$) all have positive estimates in all five variants of the VOC datasets, with their P-values are all smaller than 0.05 and many of them being extremely small, suggesting they are all statistically significant and contribute to an increased risk of death.

However, other variables have varying effects for different variant datasets. Variable "acuterespdistress_yn" ($x_{10}^{(1)}$) is statistically significant with positive estimates for all VOC datasets except the Alpha Beta dataset; for the Alpha Beta dataset, "acuterespdistress_yn" ($x_{10}^{(1)}$) has a negative estimate although this finding is not considered statistically significant. Variable "sob_yn" ($x_{19}^{(1)}$)

is statistically significant for all VOC datasets, but it has a negative estimate for the Gamma dataset and positive estimates for all other VOC datasets. Variable "pna_yn" ($x_8^{(1)}$) is found to be statistically significant in all VOC datasets. However, in the case of the Omicron dataset, it has a negative estimate, whereas in all other VOC datasets, it has positive estimates.

The binary variable "headache_yn" ($x_{21}^{(1)}$) is estimated to be positive for the Delta dataset but negative for all other VOC datasets. These results suggest that the variable headache has a positive influence on the outcome for the Delta variant but negative impacts on other variants. Such findings refine the discovery in Section 5.7, which indicates that people who report having headaches are less likely to see an increase in their risk of death compared to those who do not report experiencing headaches.

Table 6.6: *Statistical analysis of each VOC dataset with the intersection feature subset under the logistic regression model*

dataset	quantity	intercept	age_group $x_3^{(1)}$	hosp_yn $x_5^{(1)}$	icu_yn $x_6^{(1)}$	pna_yn $x_8^{(1)}$	acuterespdistress_yn $x_{10}^{(1)}$	mechvent_yn $x_{11}^{(1)}$	sob_yn $x_{19}^{(1)}$	headache_yn $x_{21}^{(1)}$	medcond_yn $x_{24}^{(1)}$
Outbreak	Estimate	-8.4603	9.0498	1.8504	0.8533	0.1594	0.6072	2.7182	0.4861	-0.9728	1.2884
	Std. Error	0.0508	0.0561	0.0217	0.0427	0.0272	0.6072	0.0759	0.0195	0.0185	0.0226
	P-value	$< 2 \times 10^{-16}$	$< 2 \times 10^{-16}$	$< 2 \times 10^{-16}$	$< 2 \times 10^{-16}$	4.36×10^{-9}	$< 2 \times 10^{-16}$	$< 2 \times 10^{-16}$	$< 2 \times 10^{-16}$	$< 2 \times 10^{-16}$	$< 2 \times 10^{-16}$
AlphaBeta	Estimate	-8.2709	9.4061	1.8391	1.9851	0.9815	-0.2844	3.2259	0.1402	-0.7030	0.7523
	Std. Error	0.1280	0.1442	0.0649	0.1722	0.0940	0.1520	0.3427	0.0545	0.0464	0.0469
	P-value	$< 2 \times 10^{-16}$	$< 2 \times 10^{-16}$	$< 2 \times 10^{-16}$	$< 2 \times 10^{-16}$	$< 2 \times 10^{-16}$	0.0613	$< 2 \times 10^{-16}$	0.0101	$< 2 \times 10^{-16}$	$< 2 \times 10^{-16}$
Gamma	Estimate	-7.3467	8.1602	2.0606	1.1987	0.5615	0.2605	2.6620	-0.1537	-0.2118	1.0087
	Std. Error	0.0805	0.0951	0.0445	0.0847	0.0557	0.2605	0.1790	0.0380	0.0325	0.0358
	P-value	$< 2 \times 10^{-16}$	$< 2 \times 10^{-16}$	$< 2 \times 10^{-16}$	$< 2 \times 10^{-16}$	$< 2 \times 10^{-16}$	0.0066	$< 2 \times 10^{-16}$	5.22×10^{-5}	7.25×10^{-11}	$< 2 \times 10^{-16}$
Delta	Estimate	-6.8338	7.1947	2.5648	1.1613	0.1792	0.2836	3.8208	0.4432	0.0808	0.7290
	Std. Error	0.0811	7.1947	2.5648	1.1613	0.1792	0.0859	0.2132	0.0348	0.0331	0.0368
	P-value	$< 2 \times 10^{-16}$	$< 2 \times 10^{-16}$	$< 2 \times 10^{-16}$	$< 2 \times 10^{-16}$	0.0007	0.0010	$< 2 \times 10^{-16}$	$< 2 \times 10^{-16}$	0.0146	$< 2 \times 10^{-16}$
Omicron	Estimate	-6.7238	5.2820	4.5221	0.6229	-1.0321	1.2832	2.6637	0.6026	-0.3094	0.9474
	Std. Error	0.1649	0.1922	0.0945	0.1671	0.1188	1.2832	0.4002	0.0837	0.0844	0.1026
	P-value	$< 2 \times 10^{-16}$	$< 2 \times 10^{-16}$	$< 2 \times 10^{-16}$	0.0002	$< 2 \times 10^{-16}$	1.6600×10^{-14}	2.8200×10^{-11}	5.9500×10^{-13}	0.0002	$< 2 \times 10^{-16}$

Chapter 7

Summary and Discussion

This essay explores the mortality risk of COVID-19 patients across different variant waves. It utilizes patient-level medical data, vaccination rates, and hospital capacities and employs a spectrum of methods and algorithms, including feature selection techniques, data balancing strategies, and diverse machine learning models, cost-sensitive classification models, artificial neural networks, and statistical inference through logistic regression. In recognizing the absence of a universally applicable solution for feature selection or classification tasks, we systematically evaluate a range of methods, emphasizing the necessity of tailoring approaches to specific contexts. Furthermore, we propose to particularly consider the intersection set of common features revealed by all considered methods and the union set of all features revealed by at least one considered method. These two sets offer us useful candidates for in-depth analysis.

The findings and contributions of this research are useful. The logistic regression model with the intersection set of predictors demonstrates the best

performance in most cases, followed by other models utilizing the same feature subset. The cost-sensitive models and artificial neural networks achieve comparable performance to traditional machine learning models, demonstrating accuracy, precision, and recall levels of around 90%. Our study also reveals that employing the "class_weight" parameter on the imbalanced training set can yield comparable results to balance the dataset before training.

Our study identifies nine important factors impacting patient survival in the context of COVID-19. These encompass six patient-level factors, including pre-existing medical conditions, acute respiratory distress syndrome status, pneumonia status, age group category, headache status, and shortness of breath (dyspnea) status, as well as the three factors showing the patient's status related to hospital aspects: hospitalization status, mechanical ventilation status, and intensive care unit admission status.

Our strategies applied to handle COVID-19 data can be extended to deal with other data for feature selection and classification. The interdisciplinary nature of our work highlights how computational tools and methodologies are applied to address real-world challenges, emphasizing the need for tailored approaches in the field. Critically, the selection and evaluation of machine learning models in specific contexts challenge the notion of a one-size-fits-all solution. Our findings underscore the importance of exploring and comparing various methods, not only within the domain of feature selection but also across the wider spectrum of selecting machine learning models.

Despite these promising findings, the thesis has limitations. While the selected feature subset shows strong performance, there might be additional relevant features not considered in this study. Furthermore, the research

focuses on datasets from the United States, which may limit generalizability to other regions. Further research could explore other features and datasets to enhance predictive capabilities. Additionally, applying the models to other countries or regions can be valuable to assess their performance in different contexts.

While our study uses the default hyperparameters specified by respective software packages, this does not necessary ensure the robustness of the results to different choices of hyperparameters. It is noted that different specifications of hyperparameters may generate discrepancies in results. To address this issue, one scheme is to employ the grid search algorithm, which allows us to systematically traverse a spectrum of parameter values in pursuit of the optimal combination. The optimal parameter values for certain chosen models are available in the appendix, as shown in Table A1. Unsurprisingly, the default models generally exhibit overall good performance, rendering the hyperparameter tuning process less impactful on the overall model performance. Given the substantial time associated with hyperparameter tuning yet the modest gains achieved, we opt to use the default parameters specified in software packages we use. However, it is recognized that this decision brings about certain limitations, notably the risk of overlooking a globally optimal solution and other intricacies in model optimization that cannot be fully captured by the default configuration. The balance between time efficiency and incremental performance gains thereby becomes a noteworthy and practical consideration.

Another limitation concerns the missing data mechanism. Consistent with many authors (e.g., Roth, 1994; Bennett, 2001; Scheffer, 2002; Newman,

2014), we remove those subject with missing values and essentially conduct a complete data analysis. Such a treatment of missing data is justified if data follows the Missing Completely at Random (MCAR) mechanism, because the remaining data subset can still be regarded as a random subsample from the underlying population. Without a comprehensive understanding of the mechanism behind missing values, our choice of regarding it as MCAR remains speculative, potentially diverging from the true nature of the missing data. Our MCAR assumption is grounded in the belief that missing values, particularly those associated with symptom indicators ($\{x_1^{(1)}, ..., x_{24}^{(1)}\}$) stem from instances where respondents dropped out or skipped questions during the form-filling process.

While adopting the MCAR assumption provides a pragmatic strategy for managing missing values, it is crucial to recognize and openly address the inherent limitations and induced biases when MCAR is unrealistic. Developing sensible methods to accommodate missing at random (MAR) or missing not at random (MNAR) is useful (e.g., Yi (2017, Section 5.5.1) and Little and Rubin (2019)), and this topic remains active in the literature.

References

Akoglu, H. (2018). User's guide to correlation coefficients. *Turkish Journal of Emergency Medicine*, *18*(3), 91–93.

Batuwita, R., and Palade, V. (2013). Class imbalance learning methods for support vector machines. *Imbalanced Learning: Foundations, Algorithms, and Applications*, 83–99.

Bennett, D. A. (2001). How can I deal with missing data in my study? *Australian and New Zealand Journal of Public Health*, *25*(5), 464–469.

Bertsimas, D., Lukin, G., Mingardi, L., Nohadani, O., Orfanoudaki, A., Stellato, B., Wiberg, H., Gonzalez-Garcia, S., Parra-Calderón, C. L., Robinson, K., Schneider, M., Stein, B., Estirado, A., a Beccara, L., Canino, R., Dal Bello, M., Pezzetti, F., Pan, A., and Group, T. H. C.-. S. (2020). Covid-19 mortality risk assessment: An international multi-center study. *PLOS ONE*, *15*(12), e0243262.

Bolboacă, S. D., Jäntschi, L., Sestraş, A. F., Sestraş, R. E., and Pamfil, D. C. (2011). Pearson-fisher chi-square statistic revisited. *Information*, *2*(3), 528–545.

Bommert, A., Sun, X., Bischl, B., Rahnenführer, J., and Lang, M. (2020). Benchmark for filter methods for feature selection in high-dimensional

classification data. *Computational Statistics & Data Analysis*, *143*, 106839.

Broomhead, D. S., and Lowe, D. (1988). Radial basis functions, multivariable functional interpolation and adaptive networks. royal signals and radar establishment malvern (united kingdom). *No. RSRE-MEMO-4148*, 1–34.

Cao, P., Zhao, D., and Zaiane, O. (2013). An optimized cost-sensitive svm for imbalanced data learning. In *Pacific-Asia Conference on Knowledge Discovery and Data mining* (pp. 280–292).

Chawla, N. V., Bowyer, K. W., Hall, L. O., and Kegelmeyer, W. P. (2002). Smote: Synthetic minority over-sampling technique. *Journal of Artificial Intelligence Research*, *16*, 321–357. doi: 10.1613/jair.953

Choong, A. C. H., and Lee, N. K. (2017). Evaluation of convolutionary neural networks modeling of dna sequences using ordinal versus one-hot encoding method. In *2017 International Conference on Computer and Drone Applications* (pp. 60–65).

Cramér, H. (1999). *Mathematical Methods of Statistics*. Princeton university press.

Cunningham, P., and Delany, S. J. (2021). K-nearest neighbour classifiers: a tutorial. *ACM Computing Surveys*, *54*(6), 1–25.

Das, S. (2001). Filters, wrappers and a boosting-based hybrid for feature selection. In *The International Conference on Machine Learning* (Vol. 1, pp. 74–81).

Dash, M., and Liu, H. (1997). Feature selection for classification. *Intelligent Data Analysis*, *1*(1-4), 131–156.

El Aboudi, N., and Benhlima, L. (2016). Review on wrapper feature selection approaches. In *2016 International Conference on Engineering & MIS* (pp. 1–5).

Ghosh, J., and Nag, A. (2001). An overview of radial basis function networks. *Radial Basis Function Networks 2: New Advances in Design*, 1–36.

Goutte, C., and Gaussier, E. (2005). A probabilistic interpretation of precision, recall and f-score, with implication for evaluation. In *European Conference on Information Retrieval* (pp. 345–359).

Guyon, I., and Elisseeff, A. (2003). An introduction to variable and feature selection. *Journal of Machine Learning Research*, *3*(Mar), 1157–1182.

Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., Kern, R., Picus, M., Hoyer, S., van Kerkwijk, M. H., Brett, M., Haldane, A., del Río, J. F., Wiebe, M., Peterson, P., Gérard-Marchant, P., Sheppard, K., Reddy, T., Weckesser, W., Abbasi, H., Gohlke, C., and Oliphant, T. E. (2020, September). Array programming with NumPy. *Nature*, *585*(7825), 357–362. Retrieved from `https://doi.org/10.1038/s41586-020-2649-2` doi: 10.1038/s41586-020-2649-2

He, W., Yi, G. Y., and Chen, L.-P. (2019). Support vector machine with graphical network structures in features. In *The 15th international conference on machine learning and data mining (mldm 2019)* (pp. 557–570).

Jahromi, A. H., and Taheri, M. (2017). A non-parametric mixture of gaussian naive bayes classifiers based on local independent features. In *2017 Artificial Intelligence and Signal Processing Conference* (pp. 209–212).

Jiang, P., Missoum, S., and Chen, Z. (2014). Optimal svm parameter selection for non-separable and unbalanced datasets. *Structural and Multidisciplinary Optimization*, *50*, 523–535.

Jović, A., Brkić, K., and Bogunović, N. (2015). A review of feature selection methods with applications. In *2015 38th international convention on information and communication technology, electronics and microelectronics* (pp. 1200–1205).

Krawczyk, B., Woźniak, M., and Schaefer, G. (2014). Cost-sensitive decision tree ensembles for effective imbalanced classification. *Applied Soft Computing*, *14*, 554–562.

Lal, T. N., Chapelle, O., Weston, J., and Elisseeff, A. (2006). Embedded methods. In *Feature Extraction: Foundations and Applications* (pp. 137–165). Springer.

Lerner, E. B., Newgard, C. D., and Mann, N. C. (2020). Effect of the coronavirus disease 2019 (covid-19) pandemic on the us emergency medical services system: a preliminary report. *Academic Emergency Medicine*, *27*(8), 693–699.

Lin, D.-Y., Gu, Y., Wheeler, B., Young, H., Holloway, S., Sunny, S.-K., Moore, Z., and Zeng, D. (2022). Effectiveness of covid-19 vaccines over a 9-month period in north carolina. *New England Journal of Medicine*, *386*(10), 933–941.

Little, R. J., and Rubin, D. B. (2019). *Statistical analysis with missing data* (Vol. 793). John Wiley & Sons.

Liu, X., Yi, G. Y., Bauman, G., and He, W. (2021). Ensembling imbalanced-spatial-structured support vector machine. *Econometrics and Statis-*

tics, *17*, 145-155. Retrieved from `https://www.sciencedirect.com/science/article/pii/S2452306220300344` doi: https://doi.org/10.1016/j.ecosta.2020.02.003

Liu, X.-Y., Wu, J., and Zhou, Z.-H. (2008). Exploratory undersampling for class-imbalance learning. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, *39*(2), 539–550.

Louapre, C., Collongues, N., Stankoff, B., Giannesini, C., Papeix, C., Bensa, C., Deschamps, R., Créange, A., Wahab, A., Pelletier, J., Heinzlef, O., Labauge, P., Guilloton, L., Ahle, G., Goudot, M., Bigaut, K., Laplaud, D.-A., Vukusic, S., Lubetzki, C., De Sèze, J., and for the Covisep investigators. (2020). Clinical characteristics and outcomes in patients with coronavirus disease 2019 and multiple sclerosis. *JAMA Neurology*, *77*(9), 1079–1088.

Luo, H., Pan, X., Wang, Q., Ye, S., and Qian, Y. (2019). Logistic regression and random forest for effective imbalanced classification. In *2019 IEEE 43rd Annual Computer Software and Applications Conference* (Vol. 1, pp. 916–917).

McNamara, L. A., Wiegand, R. E., Burke, R. M., Sharma, A. J., Sheppard, M., Adjemian, J., Ahmad, F. B., Anderson, R. N., Barbour, K. E., Binder, A. M., Dasgupta, S., Dee, D. L., Jones, E. S., Kriss, J. L., Lyons, B. C., McMorrow, M., Payne, D. C., Reses, H. E., Rodgers, L. E., Walker, D., Verani, J. R., and Schrag, S. J. (2022). Estimating the early impact of the us covid-19 vaccination programme on covid-19 cases, emergency department visits, hospital admissions, and deaths among adults aged 65 years and older: an ecological analysis of national

surveillance data. *The Lancet*, *399*(10320), 152–160.

Miles, J. (2005). R-squared, adjusted r-squared. *Encyclopedia of Statistics in Behavioral Science*.

Moghadas, S. M., Vilches, T. N., Zhang, K., Wells, C. R., Shoukat, A., Singer, B. H., Meyers, L. A., Neuzil, K. M., Langley, J. M., Fitzpatrick, M. C., et al. (2021). The impact of vaccination on coronavirus disease 2019 (covid-19) outbreaks in the united states. *Clinical Infectious Diseases*, *73*(12), 2257–2264.

Mohri, M., Rostamizadeh, A., and Talwalkar, A. (2018). *Foundations of Machine Learning*. MIT press.

Murtagh, F. (1991). Multilayer perceptrons for classification and regression. *Neurocomputing*, *2*(5-6), 183–197.

Myles, A. J., Feudale, R. N., Liu, Y., Woody, N. A., and Brown, S. D. (2004). An introduction to decision tree modeling. *Journal of Chemometrics*, *18*(6), 275–285.

Newman, D. A. (2014). Missing data: Five practical guidelines. *Organizational Research Methods*, *17*(4), 372–411.

Noble, W. S. (2006). What is a support vector machine? *Nature Biotechnology*, *24*(12), 1565–1567.

Oshiro, T. M., Perez, P. S., and Baranauskas, J. A. (2012). How many trees in a random forest? In *International workshop on machine learning and data mining in pattern recognition* (pp. 154–168).

Osowski, S., Siwek, K., and Markiewicz, T. (2004). MLP and SVM networks-a comparative study. In *Proceedings of the 6th nordic signal processing symposium, 2004. norsig 2004.* (pp. 37–40).

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, *12*(Oct), 2825–2830.

Peng, C.-Y. J., Lee, K. L., and Ingersoll, G. M. (2002). An introduction to logistic regression analysis and reporting. *The Journal of Educational Research*, *96*(1), 3–14.

Potdar, K., Pardawala, T. S., and Pai, C. D. (2017). A comparative study of categorical variable encoding techniques for neural network classifiers. *International Journal of Computer Applications*, *175*(4), 7–9.

Rodgers, J. L., and Nicewander, W. A. (1988). Thirteen ways to look at the correlation coefficient. *American Statistician*, *42*(1), 59–66.

Rosset, S., and Zhu, J. (2007). Piecewise linear regularized solution paths. *The Annals of Statistics*, *35*(3), 1012–1030. Retrieved 2023-11-06, from `http://www.jstor.org/stable/25463590`

Roth, P. L. (1994). Missing data: A conceptual review for applied psychologists. *Personnel Psychology*, *47*(3), 537–560.

Sadi, A. A., Chowdhury, L., Jahan, N., Rafi, M. N. S., Chowdhury, R., Khan, F. A., and Mohammed, N. (2022). Lmfloss: A hybrid loss for imbalanced medical image classification. *arXiv preprint arXiv:2212.12741*.

Scheffer, J. (2002). Dealing with missing data. *Research Letters in the Information and Mathematical Sciences*, *3*, 153–160.

Sun, Y., Wong, A. K., and Kamel, M. S. (2009). Classification of imbalanced data: A review. *International Journal of Pattern Recognition*

and Artificial Intelligence, *23*(04), 687–719.

Wasserstein, R. L., and Lazar, N. A. (2016). The ASA statement on p-values: context, process, and purpose. *The American Statistician*, *70*(2), 129–133.

World Health Organization. (2023a). *COVID-19 Data Explorer.* `https://covid19.who.int/`. (Accessed: May 18, 2023)

World Health Organization. (2023b). *Tracking SARS-CoV-2 Variants.* `https://www.who.int/en/activities/tracking-SARS-CoV-2-variants/`. (Accessed: May 18, 2023)

Xie, T., Yu, H., and Wilamowski, B. (2011). Comparison between traditional neural networks and radial basis function networks. In *2011 IEEE International Symposium on Industrial Electronics* (pp. 1194–1199).

Yi, G. Y. (2017). *Statistical Analysis with Measurement Error or Misclassification: Strategy, Method and Application.* Springer.

Zychlinski, S. (n.d.). *Dython:nominal.* Retrieved from `http://shakedzy.xyz/dython/modules/nominal/`

Appendix

In this appendix, we provide the Python code utilized in this study. The code is organized as different sections based on its functionality, corresponding to the topics discussed in the previous chapters. The included code covers various aspects such as data pre-processing, feature selection, and statistical analysis for the integrated data as well as the data related to VOCs.

Data Pre-processing

The following code is used to pre-process all the three CDC datasets described in Chapters 2 and 3.

```
####################
#Data Preprocessing#
####################
import pandas as pd
import numpy as np
from datetime import datetime, timedelta
import matplotlib.pyplot as plt
import seaborn as sns
from dython.nominal import associations
import dataframe_image as dfi
```

```
##############################################################################
#                                                                            #
#Chapter 3.1: COVID-19 Case Surveillance Restricted Use Detailed Data:       #
#                                                                            #
#Since this dataset contains different types of date,                        #
# it may be confused in later analysis,                                      #
#therefore only the earliest date is kept to better track with the variant waves. #
#                                                                            #
#For geography indicators such as state and county, only keep the county_fips_code#
#to reduce the dimension.                                                    #
#                                                                            #
#For each indicator variable/column, it has 5 different value:              #
#Yes, No, Missing, Unknown, and empty cell.                                  #
#                                                                            #
#To understand the relationship between covariates and the outcome variable, #
#all rows that contain at least one "Missing"                                #
#or "Unknown" or a empty cell will be dropped.                               #
#                                                                            #
##############################################################################

#reading selected columns only to reduce the memory requirement.
fields = ['current_status', 'cdc_case_earliest_dt', 'sex', 'age_group', 'race_ethnicity_combined',
          'county_fips_code', 'hosp_yn', 'icu_yn', 'hc_work_yn', 'pna_yn', 'abxchest_yn',
          'acuterespdistress_yn', 'mechvent_yn', 'fever_yn', 'sfever_yn', 'chills_yn', 'myalgia_yn',
          'runnose_yn', 'sthroat_yn', 'cough_yn', 'sob_yn', 'nauseavomit_yn',
         'headache_yn', 'abdom_yn', 'diarrhea_yn', 'medcond_yn', 'death_yn']

#deleting the rows with missing value or unknown value
missing_values = ['Missing', 'Unknown']


part1 = pd.read_csv('E:/WesternU_MSc/research_paper/CDC_data/restricted
_____________________/COVID_Cases_Restricted_Detailed_04042022_Part_1.csv',
                    usecols= fields)
part2 = pd.read_csv('E:/WesternU_MSc/research_paper/CDC_data/restricted
_____________________/COVID_Cases_Restricted_Detailed_04042022_Part_2.csv',
                    usecols= fields)
part3 = pd.read_csv('E:/WesternU_MSc/research_paper/CDC_data/restricted
_____________________/COVID_Cases_Restricted_Detailed_04042022_Part_3.csv',
                    usecols= fields)
part4 = pd.read_csv('E:/WesternU_MSc/research_paper/CDC_data/restricted
_____________________/COVID_Cases_Restricted_Detailed_04042022_Part_4.csv',
                    usecols= fields)
part5 = pd.read_csv('E:/WesternU_MSc/research_paper/CDC_data/restricted/
_____________________COVID_Cases_Restricted_Detailed_04042022_Part_5.csv',
                    usecols= fields)
part6 = pd.read_csv('E:/WesternU_MSc/research_paper/CDC_data/restricted
_____________________/COVID_Cases_Restricted_Detailed_04042022_Part_6.csv',
                    usecols= fields)
part7 = pd.read_csv('E:/WesternU_MSc/research_paper/CDC_data/restricted
_____________________/COVID_Cases_Restricted_Detailed_04042022_Part_7.csv',
```

```
                    usecols= fields)
part8 = pd.read_csv('E:/WesternU_MSc/research_paper/CDC_data/restricted
_____________________/COVID_Cases_Restricted_Detailed_04042022_Part_8.csv',
                    usecols= fields)
part9 = pd.read_csv('E:/WesternU_MSc/research_paper/CDC_data/restricted
_____________________/COVID_Cases_Restricted_Detailed_04042022_Part_9.csv',
                    usecols= fields)
part10 = pd.read_csv('E:/WesternU_MSc/research_paper/CDC_data/restricted
_____________________/COVID_Cases_Restricted_Detailed_04042022_Part_10.csv',
                     usecols= fields)

#for field in fields:

cases_list = []

for part in [part1, part2, part3, part4, part5, part6, part7, part8, part9, part10]:

    #drop all rows with empty cell
    part = part.dropna()

    #filter out probable case and keep confirmed cases only
    part = part[part['current_status'] == 'Laboratory-confirmed_case']

    #remove the rows with "Missing" & "Unknown"
    part = part[(~part[fields].isin(missing_values)).all(1)]
    part = part.drop(columns = ['current_status'])

    cases_list.append(part)


#combine multiple parts into one final dataframe
cases = pd.concat(cases_list)


#save to csv
cases.to_csv('E:/WesternU_MSc/research_paper/CDC_data/cleaned_confirmed_case.csv', index=False)


#read cleaned_confirmed_case dataset
cases = pd.read_csv('E:/WesternU_MSc/research_paper/CDC_data/cleaned_confirmed_case.csv')

#convert object to datetime datatype and sort by date
cases['cdc_case_earliest_dt'] = pd.to_datetime(cases['cdc_case_earliest_dt'])
cases = cases.sort_values(by = 'cdc_case_earliest_dt')

#################################################################
#modify FIPS code datatype: float -> object                     #
#final decision here: remove FIPS code since maybe some area #
#recorded the details better but some areas did not.            #
#################################################################

#cases['county_fips_code'] = cases['county_fips_code'].astype(object)
```

```
cases = cases.drop(columns = ['county_fips_code'])

#convert yes/no indicator into binary integer
#cases = cases[cases.columns].replace({'No': 0, 'Yes': 1, 'Female': 0, 'Male': 1})
cases.shape
cases.head()
cases.info()
categorical = cases.columns.to_list()
categorical.remove('cdc_case_earliest_dt')

fig, axes = plt.subplots(nrows=6, ncols=4, figsize=(50,50))

for i in range(0,6):
    for j in range(0,4):
        cases.groupby('death_yn')[categorical[4*i+j]].value_counts()
        .sort_index().unstack(0).plot(kind = 'bar', ax = axes[i,j])

plt.subplots_adjust(left=0.1,
                    bottom=0.1,
                    right=0.9,
                    top=0.9,
                    wspace=0.2,
                    hspace=0.8)


plt.show()


#cases.groupby('death_yn')['county_fips_code'].value_counts().plot(kind='bar', figsize=(300,300))
indicator_yn = categorical.copy()
indicator_yn.remove('age_group')
#indicator_yn.remove('county_fips_code')
indicator_yn.remove('race_ethnicity_combined')
indicator_yn.remove('death_yn')

dic = {}


for i in range(len(indicator_yn)):
    dic[indicator_yn[i]] = cases[cases['death_yn'] == "Yes"][indicator_yn[i]].value_counts().sort_index()



df = pd.concat(dic)
df
death = cases[cases['death_yn'] == "Yes"]

fig, axes = plt.subplots(nrows=6, ncols=4, figsize=(50,50))

for i in range(0,6):
    for j in range(0,4):
        death[categorical[4*i+j]].value_counts().sort_index().plot(kind = 'bar',
```

```
                        ax = axes[i,j], title = categorical[4*i+j], color = ['green', 'black'])

plt.subplots_adjust(left=0.1,
                    bottom=0.1,
                    right=0.9,
                    top=0.9,
                    wspace=0.2,
                    hspace=0.8)
plt.show()



complete_correlation= associations(cases, nominal_columns = 'all',
                                   numerical_columns = None, compute_only = True)
complete_corr=complete_correlation['corr']
complete_corr = complete_corr.style.background_gradient(cmap='coolwarm', axis=None).set_precision(2)
complete_corr
#dfi.export(complete_corr,'complete_corr.png')
hide_columns = cases.columns.to_list()
hide_columns.remove('death_yn')
complete_corr=complete_correlation['corr']
death_corr = complete_corr.drop(columns = hide_columns)
death_corr = death_corr.sort_values(by = 'death_yn', ascending= False)
death_corr = death_corr.iloc[1: , :].head(15).transpose()
death_corr = death_corr.style.background_gradient(cmap='coolwarm', axis=None).set_precision(2)
death_corr
#dfi.export(death_corr,'death_corr.png')
```

```
##############################################################################
#                                                                            #
#Chapter 3.2: COVID-19 Hospital Data from the National Hospital Care Survey#
#                                                                            #
#In this dataset, we are trying to monitor the running pressure              #
#or the running capacity of hospitals during COVID-19 waves.                 #
#                                                                            #
##There are 3 different corresponding measurements which are:                #
#percent, Average length of stay(days), and Number of Encounters.            #
#                                                                            #
#Since neither averge length of stay and # of encounters has spanned the     #
#entire timeframe (Mar 2020 - Nov 2021),                                     #
#only percent is used to provide the insight about running pressure.         #
#                                                                            #
#To better monitor the pressure, we use indicator columns to select          #
#only "Non-COVID-19" as the field we interested in because remaining         #
#columns are about confirmed/suspected COVID-19 cases.                       #
#The 3 screening columns are excluded as well.                               #
#                                                                            #
#To have a better overview, we igorne the effect of age or sex of patients,#
#since these two group indicators will not affect if the patient need        #
#medical treatments or not. They all need medical care.                      #
# Therefore, only "Total" is selected in the "Group" column.                 #
#                                                                            #
#Hospital settings are indicated by the "Setting" column                     #
#and it has "ED" for emergency department,                                   #
#and "IP" for Inpatient. To have a better picture of running pressure,       #
#the dataframe is splitted by setting and combined together by date.         #
##############################################################################



#read NHCS hospital data
hospital = pd.read_csv('E:/WesternU_MSc/research_paper/CDC_data
_____________________/COVID-19_Hospital_Data_from_the_National_Hospital_Care_Survey.csv')

hospital = hospital[(hospital['Indicator'] == 'Non-COVID-19') & (hospital['Group'] == 'Total')]
#drop unnecessary columns
hospital = hospital.drop(columns = ['Figure', 'Time', 'End_Time','Indicator',
                                    'Group', 'Subgroup', 'Measure'])
#split the dataframe by setting/department and combine them together by date.
ED = hospital[hospital['Setting'] == 'ED']
IP = hospital[hospital['Setting'] == 'IP']
running_cap = ED.merge(IP, on = 'Start_Time', how = 'left')
#rename columns
running_cap = running_cap.rename(columns = {'Value_x' : 'Non_COVID_Percent_ED',
                                            'Value_y' : 'Non_COVID_Percent_IP'})
running_cap = running_cap.drop(columns = ['Setting_x', 'Setting_y'])
running_cap['Start_Time'] = pd.to_datetime(running_cap['Start_Time'])
running_cap.head()
running_cap.plot(x="Start_Time", y=["Non_COVID_Percent_ED", 'Non_COVID_Percent_IP'],
```

```
                kind="line", figsize=(9, 8))
plt.xlabel('Time')
plt.ylabel('Percentage of Non-COVID-19 Patients')
plt.show()



##############################################################################################
#                                                                                            #
#Chapter 3.3: Rates of COVID-19 Cases or Deaths by Age Group                                 #
#and Vaccination Status(and Booster Dose)                                                    #
#                                                                                            #
#The difference between these two dataset,                                                   #
#(1)Rates of COVID-19 Cases or Deaths by Age Group and Vaccination Status &                  #
#(2)Rates of COVID-19 Cases or Deaths by Age Group and Vaccination Status                    #
#and Booster Dose,                                                                           #
#is that the later dataset only contain the data after Sep 2021                             #
#with information about booster dose,                                                        #
#while the previous one starts from Apr 2021 without any columns about booster dose.        #
#                                                                                            #
#For these two datasets, we only interested in the effectivenss of vaccination              #
#to prevent death on each age group ignoring the effects brought by gender                   #
#or vaccination type.                                                                        #
#Therefore, all rows with case rate are removed. Also, the MMWR week is converted           #
#to datetime variable as well to be matched with the case data.                              #
#                                                                                            #
##############################################################################################



#read vaccination datasets
vacc = pd.read_csv('E:/WesternU_MSc/research_paper/CDC_data
_____________________/Rates_of_COVID-19_Cases_or_Deaths_by_Age_Group_and_Vaccination_Status.csv')

#drop wanted columns in vaccination dataset
vacc = vacc.drop(columns = ['month','Vaccine_product', 'Vaccinated_with_outcome',
                            'Unvaccinated_with_outcome',
                           'Crude_IRR', 'Age_adjusted_vax_IR', 'Age_adjusted_unvax_IR',
                            'Age_adjusted_IRR', 'Continuity_correction'])

#convert MMWR week to string and create a date column
vacc['MMWR_week'] = vacc['MMWR_week'].astype(str)
vacc['date'] = pd.to_datetime(vacc['MMWR_week']+ '-0', format='%Y%W-%w')

#keep all age groups and only outcome == death
vacc = vacc[(vacc['outcome'] != 'case') & (vacc['Age_group'] != 'all_ages_adj')]

#sort by date and drop a few more columns
vacc = vacc.sort_values(by = 'date')
vacc = vacc.drop(columns = ['outcome', 'MMWR_week'])

#calculate the  death rate for each age group as the population-weighted sum of
```

```
# death rate of different vaccination status
vacc['death_IR'] = vacc['Fully_vaccinated_population']/(vacc['Fully_vaccinated_population']
                    +vacc['Unvaccinated_population'])*vacc['Crude_vax_IR']
                    +vacc['Unvaccinated_population']/(vacc['Fully_vaccinated_population']
                    +vacc['Unvaccinated_population'])*vacc['Crude_unvax_IR']

#drop other columns and only keep the overall death rate
vacc = vacc.drop(columns = ['Fully_vaccinated_population', 'Unvaccinated_population',
                            'Crude_vax_IR', 'Crude_unvax_IR'])

#from Sep 26, 2021, use the booster dose dataframe to indicate the death rate
# since the age groups are different
vacc = vacc[vacc['date'] < "2021-09-26"]
vacc.rename(columns = {'Age_group':"age_group"},inplace = True)

#convert age_group column to a list and make-up a new list by using the upper bound of the each age group
#for example, age group "12-17" will be converted to integer 17.
age_group = vacc['age_group'].tolist()
age = []

for i in range(len(vacc['death_IR'])):
    if age_group[i] == "12-17":
        age.append(17)
    elif age_group[i] == "18-29":
        age.append(29)
    elif age_group[i] == "30-49":
        age.append(49)
    elif age_group[i] == "50-64":
        age.append(64)
    elif age_group[i] == "65-79":
        age.append(79)
    elif age_group[i] == "80+":
        age.append(80)
    else:
        age.append(0)

vacc['age_group'] = age



#read booster dose datasets
booster = pd.read_csv('E:/WesternU_MSc/research_paper/CDC_data/
_____________________Rates_of_COVID-19_Cases_or_Deaths_by_Age_Group_and_Vaccination_Status_and_Booster_Dose.csv')

#drop wanted columns in booster dose dataset
booster = booster.drop(columns = ['month', 'vaccine_product', 'boosted_with_outcome',
                                  'primary_series_only_with_outcome',
                                 'unvaccinated_with_outcome','crude_booster_irr', 'crude_irr',
                                 'age_adj_booster_ir',
                                 'age_adj_vax_ir', 'age_adj_unvax_ir', 'age_adj_booster_irr',
                                 'age_adj_irr', 'continuity_correction'])
```

```
#keep all age groups and only outcome == death
booster = booster[(booster['outcome'] != 'case') & (booster['age_group'] != 'all_ages')]

#convert MMWR week to string and create a date column
booster['mmwr_week'] = booster['mmwr_week'].astype(str)
booster['date'] = pd.to_datetime(booster['mmwr_week']+ '-0', format='%Y%W-%w')

#sort by date and drop a few more columns
booster = booster.sort_values(by = 'date')
booster = booster.drop(columns = ['outcome', 'mmwr_week'])

#calculate the  death rate for each age group as the population-weighted sum
# of death rate of different vaccination status
booster['death_IR'] = booster['boosted_population']/(booster['boosted_population']
                        +booster['primary_series_only_population']
                        +booster['unvaccinated_population'])*booster['crude_booster_ir']
                        +booster['primary_series_only_population']/(booster['boosted_population']
                        +booster['primary_series_only_population']
                        +booster['unvaccinated_population'])*booster['crude_primary_series_only_ir']
                        +booster['unvaccinated_population']/(booster['boosted_population']
                        +booster['primary_series_only_population']
                        +booster['unvaccinated_population'])*booster['crude_unvax_ir']

#drop other columns and only keep the overall death rate
booster = booster.drop(columns = ['boosted_population', 'primary_series_only_population',
                                  'unvaccinated_population',
                                  'crude_booster_ir', 'crude_primary_series_only_ir', 'crude_unvax_ir'])



#convert age_group column to a list and make-up a new list by using the upper bound of the each age group
#for example, age group "12-17" will be converted to integer 17.
age_group = booster['age_group'].tolist()
age = []

for i in range(len(booster['death_IR'])):
    if age_group[i] == "12-17":
        age.append(17)
    elif age_group[i] == "18-49":
        age.append(49)
    elif age_group[i] == "50-64":
        age.append(64)
    elif age_group[i] == "65+":
        age.append(65)
    else:
        age.append(0)

booster['age_group'] = age
```

```
################################################################################
#                                                                              #
#Chapter 3.5.1: Data Integration                                               #
#                                                                              #
#Now we have already pre-processed all these three datasets,                   #
#and it is the time to combine them together based on date and age.            #
#                                                                              #
#convert age_group column in cases dataframe to a list and make-up a new list#
#by using the median of the each age group.                                    #
#for example, age group "10-19" will be converted to integer 15.               #
#                                                                              #
#the "age" column created by the following lines will be removed later         #
#since it is just an indicator to match with age groups                        #
#in vaccination and booster dataframe.                                         #
#                                                                              #
################################################################################

age_group = cases['age_group'].tolist()
age = []

for i in range(cases.shape[0]):
    if age_group[i] == "0 - 9 Years":
        age.append(5)
    elif age_group[i] == "10 - 19 Years":
        age.append(15)
    elif age_group[i] == "20 - 29 Years":
        age.append(25)
    elif age_group[i] == "30 - 39 Years":
        age.append(35)
    elif age_group[i] == "40 - 49 Years":
        age.append(45)
    elif age_group[i] == "50 - 59 Years":
        age.append(55)
    elif age_group[i] == "60 - 69 Years":
        age.append(65)
    elif age_group[i] == "70 - 79 Years":
        age.append(75)
    elif age_group[i] == "80+ Years":
        age.append(80)
    else:
        age.append(0)

cases['age'] = age
```

```
#########################################################################
#First, the cases dataframe is merged with running capacity dataframe. #
#This process is relatively easier since the date is the               #
#only variable to be matched and age group is excluded in the process. #
#This part is done by outer join with date as the key.                 #
#                                                                      #
#Since the running capacity is recorded weekly since 2020-03-18,       #
#I assume that the non-covid-19 patient percent is 100% before         #
#2020-03-18. For the case date that is in between of two consecutive   #
#running capacity record date,                                         #
#the running capacity is filled by the value of previous row.          #
#                                                                      #
#For example, there are two consecutive running capacity record        #
#date: 2020-03-18 and 2020-03-25,                                      #
#the running capacity on the date of 2020-03-19 will be filled         #
#by the value of 2020-03-18,                                           #
#and the value of 2020-03-20 will be filled                            #
#by the value of 2020-03-19 and so on.                                 #
#########################################################################

cases.rename(columns = {'cdc_case_earliest_dt':"date"},inplace = True)
running_cap.rename(columns = {'Start_Time':"date"},inplace = True)

df = pd.concat([cases, running_cap])
df = df.sort_values(by = 'date')

df = pd.merge(cases, running_cap, on = 'date', how = 'outer')
df['Non_COVID_Percent_ED'].fillna(method='pad', inplace=True)
df['Non_COVID_Percent_IP'].fillna(method='pad', inplace=True)
df['Non_COVID_Percent_ED'].fillna(100, inplace=True)
df['Non_COVID_Percent_IP'].fillna(100, inplace=True)
```

```
#############################################################################
#Now it time to deal with the vaccination dataframe                         #
#and the cases dataframe.                                                   #
#                                                                           #
#Since CDC has different age group structures for primary doses             #
#and the booster dose, the age group are matched mannually:                 #
#                                                                           #
#CDC case age gorup;     primary vaccination age group;     booster age group#
#0-9 with median 5       N/A                                N/A             #
#10-19 with median 15    12-17                              12-17           #
#20-29 with median 25    18-29                              18-49           #
#30-39 with median 35    30-49                              18-49           #
#40-49 with median 45    30-49                              18-49           #
#50-59 with median 55    50-64                              50-64           #
#60-69 with median 65    65-79                              65+             #
#70-79 with median 75    65-79                              65+             #
#80+                     80+                                65+             #
#                                                                           #
#                                                                           #
#The missing value for age group 0-9 will be handled later after splitted  #
#according to different variant waves.                                      #
#                                                                           #
#Since the death rate recorded in vacc and booster dataframe is on          #
#weekly-basis, when comparing the dates, we need to specify the range       #
#of each week so that the dates in cases dataframe could be matched.        #
#                                                                           #
#For example, two consecutive vaccination record date is 2021-04-11         #
#and 2021-04-18, then all rows in cases dateframe with date >= 2021-04-11   #
#and date < 2021-04-18 will be matched with the records on 2021-04-11       #
#in vacc dataframe.                                                         #
#############################################################################


integrated = df.copy()
integrated["death_rate"] = np.nan

for i in range(integrated.shape[0]):
    for j in range(vacc.shape[0]):
        if integrated.iloc[i]['date'] >= vacc.iloc[j]['date']
            and integrated.iloc[i]['date'] < vacc.iloc[j]['date']+ timedelta(days=7):
            if integrated.iloc[i]['age'] == 15 and vacc.iloc[j]['age_group'] == 17:
                integrated.iloc[i,28] = vacc.iloc[j]['death_IR']
            if integrated.iloc[i]['age'] == 25 and vacc.iloc[j]['age_group'] == 29:
                integrated.iloc[i,28] = vacc.iloc[j]['death_IR']
            if integrated.iloc[i]['age'] == 35 and vacc.iloc[j]['age_group'] == 49:
                integrated.iloc[i,28] = vacc.iloc[j]['death_IR']
            if integrated.iloc[i]['age'] == 45 and vacc.iloc[j]['age_group'] == 49:
                integrated.iloc[i,28] = vacc.iloc[j]['death_IR']
            if integrated.iloc[i]['age'] == 55 and vacc.iloc[j]['age_group'] == 64:
                integrated.iloc[i,28] = vacc.iloc[j]['death_IR']
            if integrated.iloc[i]['age'] == 65 and vacc.iloc[j]['age_group'] == 79:
                integrated.iloc[i,28] = vacc.iloc[j]['death_IR']
```

```
            if integrated.iloc[i]['age'] == 75 and vacc.iloc[j]['age_group'] == 79:
                integrated.iloc[i,28] = vacc.iloc[j]['death_IR']
            if integrated.iloc[i]['age'] == 80 and vacc.iloc[j]['age_group'] == 80:
                integrated.iloc[i,28] = vacc.iloc[j]['death_IR']


for i in range(integrated.shape[0]):
    for j in range(booster.shape[0]):
        if integrated.iloc[i]['date'] >= booster.iloc[j]['date']
            and integrated.iloc[i]['date'] < booster.iloc[j]['date']+ timedelta(days=7):
            if integrated.iloc[i]['age'] == 15 and booster.iloc[j]['age_group'] == 17:
                integrated.iloc[i,28] = booster.iloc[j]['death_IR']
            if integrated.iloc[i]['age'] == 25 and booster.iloc[j]['age_group'] == 49:
                integrated.iloc[i,28] = booster.iloc[j]['death_IR']
            if integrated.iloc[i]['age'] == 35 and booster.iloc[j]['age_group'] == 49:
                integrated.iloc[i,28] = booster.iloc[j]['death_IR']
            if integrated.iloc[i]['age'] == 45 and booster.iloc[j]['age_group'] == 49:
                integrated.iloc[i,28] = booster.iloc[j]['death_IR']
            if integrated.iloc[i]['age'] == 55 and booster.iloc[j]['age_group'] == 64:
                integrated.iloc[i,28] = booster.iloc[j]['death_IR']
            if integrated.iloc[i]['age'] == 65 and booster.iloc[j]['age_group'] == 65:
                integrated.iloc[i,28] = booster.iloc[j]['death_IR']
            if integrated.iloc[i]['age'] == 75 and booster.iloc[j]['age_group'] == 65:
                integrated.iloc[i,28] = booster.iloc[j]['death_IR']
            if integrated.iloc[i]['age'] == 80 and booster.iloc[j]['age_group'] == 65:
                integrated.iloc[i,28] = booster.iloc[j]['death_IR']


integrated = integrated.drop(columns = ['age'])
integrated.to_csv('E:/WesternU_MSc/research_paper/CDC_data/integrated.csv', index=False)
integrated.info()


#########################################################################
#                                                                       #
#Chapter 3.5.2: Division                                                #
#Split the integrated dataframe according to the Date of designation    #
#for each VOCs(variants of concern) published by WHO                    #
#                                                                       #
#For the Delta variant, there are two dates published by WHO            #
#which are VOI (variant of interest) and VOC.                           #
#Here the VOC date is used for consistency.                             #
#                                                                       #
#Alpha & Beta : 18-Dec-2020                                             #
#Gamma : 11-Jan-2021                                                    #
#Delta : 11-May-2021                                                    #
#Omicron: 26-Nov-2021                                                   #
#                                                                       #
#########################################################################

outbreak = integrated[integrated['date']< '2020-12-18']
AlphaBeta = integrated[(integrated['date']>= '2020-12-18') & (integrated['date'] < '2021-01-11')]
```

```
Gamma = integrated[(integrated['date']>= '2021-01-11') & (integrated['date'] < '2021-05-11')]
Delta = integrated[(integrated['date']>= '2021-05-11') & (integrated['date'] < '2021-11-26')]
Omicron = integrated[integrated['date']>= '2021-11-26']

outbreak.to_csv('E:/WesternU_MSc/research_paper/CDC_data/outbreak.csv', index=False)
AlphaBeta.to_csv('E:/WesternU_MSc/research_paper/CDC_data/AlphaBeta.csv', index=False)
Gamma.to_csv('E:/WesternU_MSc/research_paper/CDC_data/Gamma.csv', index=False)
Delta.to_csv('E:/WesternU_MSc/research_paper/CDC_data/Delta.csv', index=False)
Omicron.to_csv('E:/WesternU_MSc/research_paper/CDC_data/Omicron.csv', index=False)
```

Feature Selection

The following code is used to select the best feature subset that is subsequently used for applying machine learning models described in Chapter 5.1.

```
#########################
#feature selection      #
#########################

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from dython.nominal import associations
import dataframe_image as dfi

from sklearn.compose import make_column_transformer
from sklearn.preprocessing import OrdinalEncoder

from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2
```

```
#########################################################################
#                                                                       #
#Chapter 5.1.1: Filter Method                                           #
#                                                                       #
#In this part, we only keep the predictors that                         #
#has correlation with death_yn >= 0.05.                                 #
#                                                                       #
#Use Cramer's V from Dython as the indicator, and the chi-square        #
#is used to double-check.                                               #
#                                                                       #
#Also, we will only keep one of pna_ yn and abxchest_yn                 #
#since they are highly correlated (>=0.75)                              #
#                                                                       #
#Therefore, final Xs are: mechvent_yn, icu_yn, hosp_yn, age_group,      #
#acuterespdistress_yn, pna_yn, medcond_yn, sob_yn, headache_yn,         #
#date, runnose_yn, death_rate, sthroat_yn                               #
#                                                                       #
#Chi-Squared Feature Selection with SelectKBest provides the same       #
#set of features with abxchest_yn that is dropped due to                #
#high correlation with pna_yn                                           #
#                                                                       #
#########################################################################

integrated = pd.read_csv('E:/WesternU_MSc/research_paper/CDC_data/integrated.csv')
integrated.columns
categorical_columns = ['abdom_yn', 'abxchest_yn', 'acuterespdistress_yn', 'age_group', 'date',
       'chills_yn', 'cough_yn', 'diarrhea_yn', 'fever_yn',
       'sfever_yn', 'hc_work_yn', 'headache_yn', 'hosp_yn', 'icu_yn',
       'mechvent_yn', 'medcond_yn', 'myalgia_yn', 'nauseavomit_yn', 'pna_yn',
       'race_ethnicity_combined', 'runnose_yn', 'sex', 'sob_yn', 'sthroat_yn', 'death_yn']

numerical_columns = ['Non_COVID_Percent_ED', 'Non_COVID_Percent_IP', 'death_rate']
df = integrated.copy()

pre_processeing = make_column_transformer(
    (OrdinalEncoder(handle_unknown="use_encoded_value", unknown_value = np.nan), categorical_columns))


df.loc[:, categorical_columns] = pre_processeing.fit_transform(df[categorical_columns])


y = df['death_yn']

predictors = ['abdom_yn', 'abxchest_yn', 'acuterespdistress_yn', 'age_group', 'date',
       'chills_yn', 'cough_yn', 'diarrhea_yn', 'fever_yn',
       'sfever_yn', 'hc_work_yn', 'headache_yn', 'hosp_yn', 'icu_yn',
       'mechvent_yn', 'medcond_yn', 'myalgia_yn', 'nauseavomit_yn', 'pna_yn',
       'race_ethnicity_combined', 'runnose_yn', 'sex', 'sob_yn', 'sthroat_yn',
       'Non_COVID_Percent_ED', 'Non_COVID_Percent_IP', 'death_rate']

x = df[predictors]
complete_correlation= associations(df, nominal_columns = categorical_columns,
```

```
                                    numerical_columns = numerical_columns, compute_only = True)
complete_corr=complete_correlation['corr']
complete_corr = complete_corr.style.background_gradient(cmap='coolwarm', axis=None).set_precision(2)
complete_corr
#dfi.export(complete_corr,'complete_corr.png')
hide_columns = df.columns.to_list()
hide_columns.remove('death_yn')
complete_corr = complete_correlation['corr']
death_corr = complete_corr.drop(columns = hide_columns)
death_corr = death_corr[death_corr >= 0.05]
death_corr = death_corr.dropna()
death_corr = death_corr.sort_values(by = 'death_yn', ascending= False)
death_corr = death_corr.iloc[1: , :].transpose()
death_corr = death_corr.style.background_gradient(cmap='coolwarm', axis=None).set_precision(2)
death_corr
#dfi.export(death_corr,'death_corr.png')
y = df['death_yn']

filter_features = ['mechvent_yn', 'icu_yn', 'hosp_yn', 'age_group',
                   'acuterespdistress_yn', 'pna_yn', 'medcond_yn',
             'sob_yn', 'headache_yn', 'date', 'runnose_yn', 'sthroat_yn']

len(filter_features)
# Chi-Squared Feature Selection with SelectKBest provides the same set of categorical features

x = df[categorical_columns]
x = x.drop(columns = ['death_yn'])
y = df['death_yn']

fs = SelectKBest(score_func=chi2, k='all')
fs.fit(x, y)

chi2 = pd.DataFrame({'Feature_Name': x.columns, 'Chi-Square_Value': fs.scores_})
chi2 = chi2.sort_values(by='Chi-Square_Value', ascending=False)
chi2
chi2.plot.bar(x="Feature_Name", y="Chi-Square_Value")

plt.show(block=True)



######################################################################
#                                                                    #
#Chapter 5.1.2: Wrapper method with foward selection                 #
#In this part, we drop the "death_rate" since it contains            #
#so many null value. "death_rate" will be an additional feature for  #
#corresponding variant subsets.                                      #
#                                                                    #
#Also, we modify the logistic regression parameters to avoid         #
#the "non convergent error": solver = 'saga', max_iter = 5000.       #
#We use weighted F1 score since the classes are imbalanced.          #
######################################################################
```

```
import mlxtend
from mlxtend.feature_selection import SequentialFeatureSelector as SFS
from sklearn.linear_model import LogisticRegression

y = df['death_yn']
x = df[predictors]
x = x.drop(columns = ['death_rate'])

sfs = SFS(LogisticRegression(solver = 'saga', max_iter = 5000),
          k_features = 26,
          forward = True,
          floating = False,
          scoring = 'f1_macro',
          cv = 5,
          verbose = 2,
          n_jobs = 1)
#Use SFS to select the top 5 features
sfs.fit(x, y)
df_SFS_results = pd.DataFrame(sfs.subsets_).transpose()
df_SFS_results
from mlxtend.plotting import plot_sequential_feature_selection as plot_sfs
fig = plot_sfs(sfs.get_metric_dict(), kind='std_err')

plt.title('Sequential Forward Selection (w. StdErr)')
plt.grid()
plt.show()
df_SFS_results['avg_score'] = df_SFS_results['avg_score'].astype(float)
max_index = df_SFS_results['avg_score'].idxmax()
selected_features = df_SFS_results['feature_names'][max_index]
wrapper_features = list(selected_features)
wrapper_features


######################################################################
# Chapter 5.1.3: embedded method                                     #
######################################################################

from sklearn.linear_model import LogisticRegression
from sklearn.feature_selection import SelectFromModel

y = df['death_yn']
x = df[predictors]
x = x.drop(columns = ['death_rate'])

logistic = LogisticRegression(C = 1, penalty = 'l1', solver = 'saga', max_iter= 5000).fit(x, y)
model = SelectFromModel(logistic, prefit = True)

x_new = model.transform(x)

df_new = pd.DataFrame(x_new)
```

```
len(list(x.columns))
coef = pd.DataFrame({'Feature_Name': x.columns, 'Coefficients': np.abs(logistic.coef_[0])})
coef.sort_values(by='Coefficients')
coef.plot.bar(x="Feature_Name", y="Coefficients")

plt.show(block=True)
embedded_features = list(coef['Feature_Name'][coef['Coefficients'] > 0.1])
embedded_features

#########################################################################
#Chapter 5.1.5 Intersection and union                                   #
#########################################################################
def intersection(a, b, c):
    return list(set(a) & set(b) & set(c))

def union(a, b, c):
    return list(set(a) | set(b) | set(c))
intersection_set = intersection(filter_features, wrapper_features, embedded_features)
intersection_set
union_set = union(filter_features,  wrapper_features, embedded_features)
union_set
len(union_set)
```

Data Analysis

The following code implements the procedures described from Sections 5.2 to 5.6, where data balancing techniques and various machine learning models are discussed.

```
############################
#Data Analysis             #
############################


import pandas as pd
import numpy as np
import imblearn
from imblearn.over_sampling import SMOTENC
from imblearn.under_sampling import RandomUnderSampler

from sklearn.compose import make_column_transformer
from sklearn.preprocessing import OrdinalEncoder
```

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler

from sklearn import metrics

##############################################################################
#Chapter 5.2: read data and balance the dataset                              #
##############################################################################
integrated = pd.read_csv('E:/WesternU_MSc/research_paper/CDC_data/integrated.csv')

categorical_columns = ['abdom_yn', 'abxchest_yn', 'acuterespdistress_yn', 'age_group', 'date',
       'chills_yn', 'cough_yn', 'diarrhea_yn', 'fever_yn',
       'sfever_yn', 'hc_work_yn', 'headache_yn', 'hosp_yn', 'icu_yn',
       'mechvent_yn', 'medcond_yn', 'myalgia_yn', 'nauseavomit_yn', 'pna_yn',
       'race_ethnicity_combined', 'runnose_yn', 'sex', 'sob_yn', 'sthroat_yn', 'death_yn']

numerical_columns = ['Non_COVID_Percent_ED', 'Non_COVID_Percent_IP', 'death_rate']
filter_features = ['mechvent_yn', 'icu_yn', 'hosp_yn', 'age_group', 'acuterespdistress_yn',
                   'pna_yn', 'medcond_yn',
                   'sob_yn', 'headache_yn', 'date', 'runnose_yn', 'sthroat_yn']

wrapper_features = ['abdom_yn', 'abxchest_yn', 'acuterespdistress_yn', 'age_group',
                    'chills_yn', 'cough_yn', 'fever_yn',
                    'headache_yn', 'hosp_yn', 'icu_yn', 'mechvent_yn', 'medcond_yn',
                    'myalgia_yn', 'nauseavomit_yn',
                    'pna_yn', 'sex', 'sob_yn', 'sthroat_yn', 'Non_COVID_Percent_ED',
                    'Non_COVID_Percent_IP']

embedded_features = ['acuterespdistress_yn', 'age_group', 'chills_yn', 'diarrhea_yn',
                     'fever_yn', 'hc_work_yn', 'headache_yn',
                     'hosp_yn', 'icu_yn', 'mechvent_yn', 'medcond_yn', 'myalgia_yn',
                     'pna_yn', 'runnose_yn', 'sex', 'sob_yn',
                     'Non_COVID_Percent_ED']

intersection_set = ['hosp_yn', 'medcond_yn', 'acuterespdistress_yn', 'mechvent_yn',
                    'pna_yn', 'icu_yn', 'age_group',
                    'headache_yn', 'sob_yn']

union_set = ['medcond_yn', 'diarrhea_yn', 'acuterespdistress_yn', 'chills_yn',
             'hosp_yn', 'abxchest_yn', 'fever_yn',
             'mechvent_yn', 'icu_yn', 'hc_work_yn', 'age_group', 'cough_yn',
               'nauseavomit_yn', 'sex', 'pna_yn', 'date',
             'sthroat_yn', 'Non_COVID_Percent_ED', 'Non_COVID_Percent_IP',
               'abdom_yn', 'myalgia_yn', 'sob_yn', 'runnose_yn',
             'headache_yn']

feature_set_dic = {'filter_features' : filter_features,
                   'wrapper_features' : wrapper_features,
                   'embedded_features' : embedded_features,
                   'intersection_set' : intersection_set,
                   'union_set' : union_set}
df = integrated.copy()
```

```
# drop death_rate column since it contains lot of missing value
df = df.drop(columns = ['death_rate'])

# convert nominal columns from string to numerical label by OridinalEncoder

pre_processeing = make_column_transformer(
    (OrdinalEncoder(handle_unknown="use_encoded_value",
                    unknown_value = np.nan), categorical_columns))


df.loc[:, categorical_columns] = pre_processeing.fit_transform(df[categorical_columns])

# for age group, date, race, Non_COVID_Percent_ED and Non_COVID_Percent_IP, scaled them down to range(0,1)

scaler = MinMaxScaler()
df[["age_group", "date", "race_ethnicity_combined", 'Non_COVID_Percent_ED', 'Non_COVID_Percent_IP']] =
scaler.fit_transform(df[["age_group", "date", "race_ethnicity_combined",
                         'Non_COVID_Percent_ED', 'Non_COVID_Percent_IP']])
df.head()

# split training and testing subset, use "stratify" parameter to
# ensure the ratio of death cases is the same in both train&test.
df_train, df_test = train_test_split(df, test_size=0.2, stratify=df['death_yn'], random_state=100)
y_train = df_train['death_yn']
y_test = df_test['death_yn']

# use imbalance-learn library to over-sampling the death cases for the training dataset.
smote_nc = SMOTENC(categorical_features=list(range(0, 25)), random_state = 100)
df_train_over, y_train_over = smote_nc.fit_resample(df_train, y_train)

print('Training set \n')
print('after over-sampling: \n', y_train_over.value_counts())
print('before over-sampling: \n', y_train.value_counts())
print('\n')


# use imbalance-learn library to under-sampling the death cases for
# the test dataset to avoid low-precision & high-accuracy.
rus = RandomUnderSampler(random_state=200)
df_test_under, y_test_under = rus.fit_resample(df_test, y_test)

print('Testing set \n')
print('after under-sampling: \n', y_test_under.value_counts())
print('before under-sampling: \n', y_test.value_counts())
```

```
#################################################################
#Chapter 5.3: Traditional machine learning models               #
#                                                               #
#Now let's build ML Models: SVM, Decision tree, Random forest,#
#Logistic regression, KNN, and Naive bayes                      #
#                                                               #
#################################################################

# SVM
from sklearn.svm import SVC

svc = SVC()

df_list = []
keys = []

for key, value in feature_set_dic.items():
    x_train = df_train_over[value]
    x_test = df_test_under[value]
    svc.fit(x_train ,y_train_over)
    y_pred = svc.predict(x_test)

    report = metrics.classification_report(y_test_under, y_pred, output_dict=True)
    df = pd.DataFrame(report).transpose()
    keys.append(key)
    df_list.append(df)

df = pd.concat(df_list, keys = keys)
df = df.round(2)
display(df)

with open('svm.tex', 'w') as tf:
     tf.write(df.to_latex())
tf.close()



# decision tree
from sklearn.tree import DecisionTreeClassifier

decision_tree = DecisionTreeClassifier()

df_list = []
keys = []

for key, value in feature_set_dic.items():
    x_train = df_train_over[value]
    x_test = df_test_under[value]
    decision_tree.fit(x_train ,y_train_over)
    y_pred = decision_tree.predict(x_test)

    report = metrics.classification_report(y_test_under, y_pred, output_dict=True)
```

```
    df = pd.DataFrame(report).transpose()
    keys.append(key)
    df_list.append(df)

df = pd.concat(df_list, keys = keys)
df = df.round(2)
display(df)

with open('decision_tree.tex', 'w') as tf:
     tf.write(df.to_latex())
tf.close()


# random forest
from sklearn.ensemble import RandomForestClassifier

random_forest = RandomForestClassifier()

df_list = []
keys = []

for key, value in feature_set_dic.items():
    x_train = df_train_over[value]
    x_test = df_test_under[value]
    random_forest.fit(x_train ,y_train_over)
    y_pred = random_forest.predict(x_test)

    report = metrics.classification_report(y_test_under, y_pred, output_dict=True)
    df = pd.DataFrame(report).transpose()
    keys.append(key)
    df_list.append(df)

df = pd.concat(df_list, keys = keys)
df = df.round(2)
display(df)

with open('random_forest.tex', 'w') as tf:
     tf.write(df.to_latex())
tf.close()


# Logistic regression
from sklearn.linear_model import LogisticRegression

logreg = LogisticRegression(solver = 'saga', max_iter = 5000)

df_list = []
keys = []

for key, value in feature_set_dic.items():
    x_train = df_train_over[value]
    x_test = df_test_under[value]
```

```
    logreg.fit(x_train ,y_train_over)
    y_pred = logreg.predict(x_test)

    report = metrics.classification_report(y_test_under, y_pred, output_dict=True)
    df = pd.DataFrame(report).transpose()
    keys.append(key)
    df_list.append(df)

df = pd.concat(df_list, keys = keys)
df = df.round(2)
display(df)

with open('logreg.tex', 'w') as tf:
     tf.write(df.to_latex())
tf.close()


# KNN
from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier(n_neighbors = 5)

df_list = []
keys = []

for key, value in feature_set_dic.items():
    x_train = df_train_over[value]
    x_test = df_test_under[value]
    knn.fit(x_train ,y_train_over)
    y_pred = knn.predict(x_test)

    report = metrics.classification_report(y_test_under, y_pred, output_dict=True)
    df = pd.DataFrame(report).transpose()
    keys.append(key)
    df_list.append(df)

df = pd.concat(df_list, keys = keys)
df = df.round(2)
display(df)

with open('KNN.tex', 'w') as tf:
     tf.write(df.to_latex())
tf.close()


#Naive bayes
from sklearn.naive_bayes import GaussianNB

gaussian = GaussianNB()

df_list = []
keys = []
```

```
for key, value in feature_set_dic.items():
    x_train = df_train_over[value]
    x_test = df_test_under[value]
    gaussian.fit(x_train ,y_train_over)
    y_pred = gaussian.predict(x_test)

    report = metrics.classification_report(y_test_under, y_pred, output_dict=True)
    df = pd.DataFrame(report).transpose()
    keys.append(key)
    df_list.append(df)

df = pd.concat(df_list, keys = keys)
df = df.round(2)
display(df)

with open('naive_bayes.tex', 'w') as tf:
     tf.write(df.to_latex())
tf.close()

##############################################################################################
#Chapter 5.4: cost-sensitive classfication models                                            #
##############################################################################################
import pandas as pd
import numpy as np
import imblearn
from imblearn.under_sampling import RandomUnderSampler

from sklearn.compose import make_column_transformer
from sklearn.preprocessing import OrdinalEncoder

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler

from sklearn import metrics
integrated = pd.read_csv('E:/WesternU_MSc/research_paper/CDC_data/integrated.csv')

categorical_columns = ['abdom_yn', 'abxchest_yn', 'acuterespdistress_yn', 'age_group', 'date',
       'chills_yn', 'cough_yn', 'diarrhea_yn', 'fever_yn',
       'sfever_yn', 'hc_work_yn', 'headache_yn', 'hosp_yn', 'icu_yn',
       'mechvent_yn', 'medcond_yn', 'myalgia_yn', 'nauseavomit_yn', 'pna_yn',
       'race_ethnicity_combined', 'runnose_yn', 'sex', 'sob_yn', 'sthroat_yn', 'death_yn']

numerical_columns = ['Non_COVID_Percent_ED', 'Non_COVID_Percent_IP', 'death_rate']
filter_features = ['mechvent_yn', 'icu_yn', 'hosp_yn', 'age_group', 'acuterespdistress_yn',
                   'pna_yn', 'medcond_yn',
                   'sob_yn', 'headache_yn', 'date', 'runnose_yn', 'sthroat_yn']

wrapper_features = ['abdom_yn', 'abxchest_yn', 'acuterespdistress_yn', 'age_group',
                    'chills_yn', 'cough_yn', 'fever_yn',
                    'headache_yn', 'hosp_yn', 'icu_yn', 'mechvent_yn', 'medcond_yn',
                    'myalgia_yn', 'nauseavomit_yn',
```

```
                    'pna_yn', 'sex', 'sob_yn', 'sthroat_yn', 'Non_COVID_Percent_ED',
                    'Non_COVID_Percent_IP']

embedded_features = ['acuterespdistress_yn', 'age_group', 'chills_yn', 'diarrhea_yn',
                     'fever_yn', 'hc_work_yn', 'headache_yn',
                     'hosp_yn', 'icu_yn', 'mechvent_yn', 'medcond_yn', 'myalgia_yn',
                       'pna_yn', 'runnose_yn', 'sex', 'sob_yn',
                     'Non_COVID_Percent_ED']

intersection_set = ['hosp_yn', 'medcond_yn', 'acuterespdistress_yn', 'mechvent_yn',
                    'pna_yn', 'icu_yn', 'age_group',
                    'headache_yn', 'sob_yn']

union_set = ['medcond_yn', 'diarrhea_yn', 'acuterespdistress_yn', 'chills_yn',
             'hosp_yn', 'abxchest_yn', 'fever_yn',
             'mechvent_yn', 'icu_yn', 'hc_work_yn', 'age_group', 'cough_yn',
               'nauseavomit_yn', 'sex', 'pna_yn', 'date',
             'sthroat_yn', 'Non_COVID_Percent_ED', 'Non_COVID_Percent_IP',
             'abdom_yn', 'myalgia_yn', 'sob_yn', 'runnose_yn',
             'headache_yn']

feature_set_dic = {'filter_features' : filter_features,
                    'wrapper_features' : wrapper_features,
                    'embedded_features' : embedded_features,
                    'intersection_set' : intersection_set,
                    'union_set' : union_set}
df = integrated.copy()

# drop death_rate column since it contains lot of missing value
df = df.drop(columns = ['death_rate'])

# convert nominal columns from string to numerical label by OridinalEncoder

pre_processeing = make_column_transformer(
    (OrdinalEncoder(handle_unknown="use_encoded_value", unknown_value = np.nan), categorical_columns))


df.loc[:, categorical_columns] = pre_processeing.fit_transform(df[categorical_columns])

# for age group, date, race, Non_COVID_Percent_ED and Non_COVID_Percent_IP,
#  scaled them down to range(0,1)

scaler = MinMaxScaler()
df[["age_group", "date", "race_ethnicity_combined", 'Non_COVID_Percent_ED', 'Non_COVID_Percent_IP']] =
scaler.fit_transform(df[["age_group", "date", "race_ethnicity_combined",
                         'Non_COVID_Percent_ED', 'Non_COVID_Percent_IP']])
df.head()
# split training and testing subset, use "stratify" parameter
# to ensure the ratio of death cases is the same in both train&test.
df_train, df_test = train_test_split(df, test_size=0.2, stratify=df['death_yn'], random_state=100)
y_train = df_train['death_yn']
y_test = df_test['death_yn']
```

```
print(y_train.value_counts())
print(y_test.value_counts())


# use imbalance-learn library to under-sampling the death cases
# for the test dataset to avoid low-precision & high-accuracy.
rus = RandomUnderSampler(random_state=200)
df_test_under, y_test_under = rus.fit_resample(df_test, y_test)

print('Testing set \n')
print('after under-sampling: \n', y_test_under.value_counts())
print('before under-sampling: \n', y_test.value_counts())



'''
The following cost sensitive/weighted models are trained on the unbalanced training set.
But are evaluated on balanced testing set with undersampling method.
'''


# SVM
# use the parameter class_weight to achieve the cost-sensitive/weighted SVC classifier

from sklearn.svm import SVC

svc = SVC(class_weight = 'balanced')

df_list = []
keys = []

for key, value in feature_set_dic.items():
    x_train = df_train[value]
    x_test = df_test_under[value]
    svc.fit(x_train ,y_train)
    y_pred = svc.predict(x_test)

    report = metrics.classification_report(y_test_under, y_pred, output_dict=True)
    df = pd.DataFrame(report).transpose()
    keys.append(key)
    df_list.append(df)

df = pd.concat(df_list, keys = keys)
df = df.round(2)
display(df)

with open('weighted_svm.tex', 'w') as tf:
     tf.write(df.to_latex())
tf.close()
# decision tree
# use the parameter class_weight to achieve the cost-sensitive/weighted decision tree classifier
```

```
from sklearn.tree import DecisionTreeClassifier

decision_tree = DecisionTreeClassifier(class_weight = 'balanced')

df_list = []
keys = []

for key, value in feature_set_dic.items():
    x_train = df_train[value]
    x_test = df_test_under[value]
    decision_tree.fit(x_train ,y_train)
    y_pred = decision_tree.predict(x_test)

    report = metrics.classification_report(y_test_under, y_pred, output_dict=True)
    df = pd.DataFrame(report).transpose()
    keys.append(key)
    df_list.append(df)

df = pd.concat(df_list, keys = keys)
df = df.round(2)
display(df)


with open('weighted_decision_tree.tex', 'w') as tf:
     tf.write(df.to_latex())
tf.close()
# Logistic regression
# use the parameter class_weight to achieve the cost-sensitive/weighted Logistic regression

from sklearn.linear_model import LogisticRegression

logreg = LogisticRegression(solver = 'saga', max_iter = 5000, class_weight = 'balanced')

df_list = []
keys = []

for key, value in feature_set_dic.items():
    x_train = df_train[value]
    x_test = df_test_under[value]
    logreg.fit(x_train ,y_train)
    y_pred = logreg.predict(x_test)

    report = metrics.classification_report(y_test_under, y_pred, output_dict=True)
    df = pd.DataFrame(report).transpose()
    keys.append(key)
    df_list.append(df)

df = pd.concat(df_list, keys = keys)
df = df.round(2)
display(df)
```

```
with open('weighted_logreg.tex', 'w') as tf:
    tf.write(df.to_latex())
tf.close()


##############################################################################
#Chapter 5.5: ANN                                                            #
##############################################################################

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

import imblearn
from imblearn.over_sampling import SMOTENC
from imblearn.under_sampling import RandomUnderSampler

from sklearn.compose import make_column_transformer
from sklearn.preprocessing import OrdinalEncoder

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler

from sklearn.neural_network import MLPClassifier
from sklearn.cluster import KMeans
import math

from sklearn import metrics
integrated = pd.read_csv('E:/WesternU_MSc/research_paper/CDC_data/integrated.csv')

categorical_columns = ['abdom_yn', 'abxchest_yn', 'acuterespdistress_yn',
                       'age_group', 'date',
       'chills_yn', 'cough_yn', 'diarrhea_yn', 'fever_yn',
       'sfever_yn', 'hc_work_yn', 'headache_yn', 'hosp_yn', 'icu_yn',
       'mechvent_yn', 'medcond_yn', 'myalgia_yn', 'nauseavomit_yn', 'pna_yn',
       'race_ethnicity_combined', 'runnose_yn', 'sex', 'sob_yn', 'sthroat_yn', 'death_yn']

numerical_columns = ['Non_COVID_Percent_ED', 'Non_COVID_Percent_IP', 'death_rate']
filter_features = ['mechvent_yn', 'icu_yn', 'hosp_yn', 'age_group',
                   'acuterespdistress_yn', 'pna_yn', 'medcond_yn',
                   'sob_yn', 'headache_yn', 'date', 'runnose_yn', 'sthroat_yn']

wrapper_features = ['abdom_yn', 'abxchest_yn', 'acuterespdistress_yn',
                    'age_group', 'chills_yn', 'cough_yn', 'fever_yn',
                    'headache_yn', 'hosp_yn', 'icu_yn', 'mechvent_yn',
                    'medcond_yn', 'myalgia_yn', 'nauseavomit_yn',
                    'pna_yn', 'sex', 'sob_yn', 'sthroat_yn',
                    'Non_COVID_Percent_ED', 'Non_COVID_Percent_IP']

embedded_features = ['acuterespdistress_yn', 'age_group', 'chills_yn',
                     'diarrhea_yn', 'fever_yn', 'hc_work_yn', 'headache_yn',
```

```
                    'hosp_yn', 'icu_yn', 'mechvent_yn', 'medcond_yn',
                    'myalgia_yn', 'pna_yn', 'runnose_yn', 'sex', 'sob_yn',
                    'Non_COVID_Percent_ED']

intersection_set = ['hosp_yn', 'medcond_yn', 'acuterespdistress_yn',
                    'mechvent_yn', 'pna_yn', 'icu_yn', 'age_group',
                    'headache_yn', 'sob_yn']

union_set = ['medcond_yn', 'diarrhea_yn', 'acuterespdistress_yn',
             'chills_yn', 'hosp_yn', 'abxchest_yn', 'fever_yn',
             'mechvent_yn', 'icu_yn', 'hc_work_yn', 'age_group',
             'cough_yn', 'nauseavomit_yn', 'sex', 'pna_yn', 'date',
             'sthroat_yn', 'Non_COVID_Percent_ED', 'Non_COVID_Percent_IP',
             'abdom_yn', 'myalgia_yn', 'sob_yn', 'runnose_yn',
             'headache_yn']

feature_set_dic = {'filter_features' : filter_features,
                    'wrapper_features' : wrapper_features,
                    'embedded_features' : embedded_features,
                    'intersection_set' : intersection_set,
                    'union_set' : union_set}
df = integrated.copy()

# drop death_rate column since it contains lot of missing value
df = df.drop(columns = ['death_rate'])

# convert nominal columns from string to numerical label by OridinalEncoder

pre_processeing = make_column_transformer(
    (OrdinalEncoder(handle_unknown="use_encoded_value", unknown_value = np.nan), categorical_columns))


df.loc[:, categorical_columns] = pre_processeing.fit_transform(df[categorical_columns])

# for age group, date, race, Non_COVID_Percent_ED and Non_COVID_Percent_IP,
#  scaled them down to range(0,1)

scaler = MinMaxScaler()
df[["age_group", "date", "race_ethnicity_combined", 'Non_COVID_Percent_ED', 'Non_COVID_Percent_IP']] =
scaler.fit_transform(df[["age_group", "date", "race_ethnicity_combined",
                         'Non_COVID_Percent_ED', 'Non_COVID_Percent_IP']])
df.head()
# split training and testing subset, use "stratify" parameter
# to ensure the ratio of death cases is the same in both train&test.
df_train, df_test = train_test_split(df, test_size=0.2, stratify=df['death_yn'], random_state=100)
y_train = df_train['death_yn']
y_test = df_test['death_yn']

# use imbalance-learn library to over-sampling the death cases for the training dataset.
smote_nc = SMOTENC(categorical_features=list(range(0, 25)), random_state = 100)
df_train_over, y_train_over = smote_nc.fit_resample(df_train, y_train)
```

```
print('Training set \n')
print('after over-sampling: \n', y_train_over.value_counts())
print('before over-sampling: \n', y_train.value_counts())
print('\n')


# use imbalance-learn library to under-sampling the death cases
# for the test dataset to avoid low-precision & high-accuracy.
rus = RandomUnderSampler(random_state=200)
df_test_under, y_test_under = rus.fit_resample(df_test, y_test)

print('Testing set \n')
print('after under-sampling: \n', y_test_under.value_counts())
print('before under-sampling: \n', y_test.value_counts())



#Now let's build MLP classifier

mlp_clf = MLPClassifier(hidden_layer_sizes = (5, 2),
                        activation = 'logistic', learning_rate = 'adaptive', max_iter = 5000)

df_list = []
keys = []

for key, value in feature_set_dic.items():
    x_train = df_train_over[value]
    x_test = df_test_under[value]
    mlp_clf.fit(x_train ,y_train_over)
    y_pred = mlp_clf.predict(x_test)

    report = metrics.classification_report(y_test_under, y_pred, output_dict=True)
    df = pd.DataFrame(report).transpose()
    keys.append(key)
    df_list.append(df)

df = pd.concat(df_list, keys = keys)
df = df.round(2)
display(df)

with open('mlp_clf.tex', 'w') as tf:
     tf.write(df.to_latex())
tf.close()




# RBFN
# https://www.madrasresearch.org/post/radial-basis-functions-neural-networks
```

```
from sklearn.cluster import KMeans
import math

df_list = []
keys = []

for key, value in feature_set_dic.items():
    x_train = df_train_over[value]
    x_test = df_test_under[value]
    x_train = x_train.to_numpy()
    x_test = x_test.to_numpy()

    K_cent= 6
    km= KMeans(n_clusters= K_cent, max_iter= 5000)
    km.fit(x_train)
    cent= km.cluster_centers_

    max=0
    for i in range(K_cent):
        for j in range(K_cent):
            d= np.linalg.norm(cent[i]-cent[j])
            if(d> max):
                max= d
    d= max

    sigma= d/math.sqrt(2*K_cent)

    shape= x_train.shape
    row= shape[0]
    column= K_cent
    G= np.empty((row,column), dtype= float)

    for i in range(row):
        for j in range(column):
            dist= np.linalg.norm(x_train[i]-cent[j])
            G[i][j]= math.exp(-math.pow(dist,2)/math.pow(2*sigma,2))

    GTG= np.dot(G.T,G)
    GTG_inv= np.linalg.inv(GTG)
    fac= np.dot(GTG_inv,G.T)
    W= np.dot(fac,y_train_over)


    row= x_test.shape[0]
    column= K_cent
    G_test= np.empty((row,column), dtype= float)
    for i in range(row):
        for j in range(column):
            dist= np.linalg.norm(x_test[i]-cent[j])
            G_test[i][j]= math.exp(-math.pow(dist,2)/math.pow(2*sigma,2))

    y_pred= np.dot(G_test,W)
```

```
    y_pred= 0.5*(np.sign(y_pred-0.5)+1)


    report = metrics.classification_report(y_test_under, y_pred, output_dict=True)
    df = pd.DataFrame(report).transpose()
    keys.append(key)
    df_list.append(df)

df = pd.concat(df_list, keys = keys)
df = df.round(2)
display(df)


with open('RBFN.tex', 'w') as tf:
     tf.write(df.to_latex())
tf.close()
```

VOC

The following code implements the procedures described in Chapter 6, with each specific VOC are analyzed with the intersection feature subset.

```
############################################################
#Chapter 6: VOC                                            #
############################################################
import pandas as pd
import numpy as np
import imblearn
from imblearn import over_sampling
from imblearn.over_sampling import RandomOverSampler

from sklearn.compose import make_column_transformer
from sklearn.preprocessing import OrdinalEncoder

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler

from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from sklearn.neural_network import MLPClassifier

from sklearn import metrics
intersection_set = ['hosp_yn', 'medcond_yn', 'acuterespdistress_yn',
                    'mechvent_yn', 'pna_yn', 'icu_yn', 'age_group',
                    'headache_yn', 'sob_yn']
```

```
categorical_col = ['hosp_yn', 'medcond_yn', 'acuterespdistress_yn',
                   'mechvent_yn', 'pna_yn', 'icu_yn', 'age_group',
                    'headache_yn', 'sob_yn', 'death_yn']
drop_columns = ['abdom_yn', 'abxchest_yn', 'date', 'Non_COVID_Percent_ED',
       'chills_yn', 'cough_yn', 'diarrhea_yn', 'fever_yn', 'Non_COVID_Percent_IP',
       'sfever_yn', 'hc_work_yn','myalgia_yn', 'nauseavomit_yn',
       'race_ethnicity_combined', 'runnose_yn', 'sex', 'sthroat_yn']


outbreak = pd.read_csv('E:/WesternU_MSc/research_paper/CDC_data/outbreak.csv')
outbreak = outbreak.drop(columns = ['death_rate'])
outbreak = outbreak.drop(columns = drop_columns)

AlphaBeta = pd.read_csv('E:/WesternU_MSc/research_paper/CDC_data/AlphaBeta.csv')
AlphaBeta = AlphaBeta.drop(columns = ['death_rate'])
AlphaBeta = AlphaBeta.drop(columns = drop_columns)

Gamma = pd.read_csv('E:/WesternU_MSc/research_paper/CDC_data/Gamma.csv')
Gamma['death_rate'] = Gamma['death_rate'].fillna(Gamma['death_rate'].median())
Gamma = Gamma.drop(columns = drop_columns)

Delta = pd.read_csv('E:/WesternU_MSc/research_paper/CDC_data/Delta.csv')
Delta['death_rate'] = Delta['death_rate'].fillna(Delta['death_rate'].median())
Delta = Delta.drop(columns = drop_columns)

Omicron = pd.read_csv('E:/WesternU_MSc/research_paper/CDC_data/Omicron.csv')
Omicron['death_rate'] = Omicron['death_rate'].fillna(Omicron['death_rate'].median())
Omicron = Omicron.drop(columns = drop_columns)


voc_list = [outbreak, AlphaBeta, Gamma, Delta, Omicron]
voc_dic = {'outbreak' : outbreak, 'AlphaBeta' : AlphaBeta,
           'Gamma' : Gamma, 'Delta' : Delta, 'Omicron' : Omicron}
for key, value in voc_dic.items():

    df_list = []
    keys = []

    df = value.copy()

    # convert nominal columns from string to numerical label by OridinalEncoder

    pre_processeing = make_column_transformer(
        (OrdinalEncoder(handle_unknown="use_encoded_value",
                        unknown_value = np.nan), categorical_col))

    df.loc[:, categorical_col] = pre_processeing.fit_transform(df[categorical_col])

    # for numerical columns scaled them down to range(0,1)
    scaler = MinMaxScaler()

    if 'death_rate' not in df.columns:
```

```
    df[["age_group"]] = scaler.fit_transform(df[["age_group"]])
else:
    df[["age_group", 'death_rate']] = scaler.fit_transform(df[["age_group", 'death_rate']])

# split training and testing subset, use "stratify" parameter to
# ensure the ratio of death cases is the same in both train&test.
df_train, df_test = train_test_split(df, test_size=0.2, stratify=df['death_yn'], random_state=100)

x_train = df_train[df_train.columns[~df_train.columns.isin(['death_yn'])]]
x_test = df_test[df_test.columns[~df_test.columns.isin(['death_yn'])]]
y_train = df_train['death_yn'].astype('int')
y_test = df_test['death_yn'].astype('int')

# use imbalance-learn library to over-sampling the death cases for the training dataset.
smote = over_sampling.SMOTE()
x_train_over, y_train_over = smote.fit_resample(x_train, y_train)

# use imbalance-learn library to over-sampling the death cases for
# the test dataset to avoid low-precision & high-accuracy.
ros = RandomOverSampler(random_state=200)
#rus = RandomUnderSampler(random_state=200)
#x_test_under, y_test_under = rus.fit_resample(x_test, y_test)
x_test_over, y_test_over = ros.fit_resample(x_test, y_test)


logreg = LogisticRegression(solver = 'saga', max_iter = 5000)
logreg.fit(x_train_over ,y_train_over)
#y_pred_logreg = logreg.predict(x_test_under)
#report_logreg = metrics.classification_report(y_test_under, y_pred_logreg, output_dict=True)
y_pred_logreg = logreg.predict(x_test_over)
report_logreg = metrics.classification_report(y_test_over, y_pred_logreg, output_dict=True)
df_logreg = pd.DataFrame(report_logreg).transpose()
keys.append('logreg')
df_list.append(df_logreg)


svc = SVC(class_weight = 'balanced')
svc.fit(x_train ,y_train)
#y_pred_svc = svc.predict(x_test_under)
#report_svc = metrics.classification_report(y_test_under, y_pred_svc, output_dict=True)
y_pred_svc = svc.predict(x_test_over)
report_svc = metrics.classification_report(y_test_over, y_pred_svc, output_dict=True)
df_svc = pd.DataFrame(report_svc).transpose()
keys.append('cost_sensitive_svc')
df_list.append(df_svc)


mlp_clf = MLPClassifier(hidden_layer_sizes = (5, 2), activation = 'logistic',
                        learning_rate = 'adaptive', max_iter = 5000)
mlp_clf.fit(x_train_over ,y_train_over)
#y_pred_mlp = mlp_clf.predict(x_test_under)
#report_mlp = metrics.classification_report(y_test_under, y_pred_mlp, output_dict=True)
```

```
y_pred_mlp = mlp_clf.predict(x_test_over)
report_mlp = metrics.classification_report(y_test_over, y_pred_mlp, output_dict=True)
df_mlp = pd.DataFrame(report_mlp).transpose()
keys.append('mlp')
df_list.append(df_mlp)


df = pd.concat(df_list, keys = keys)
df = df.round(2)
display(df)

with open("%s.tex" % key, 'w') as tf:
    tf.write(df.to_latex())
tf.close()
```

Hyperparameter Tuning

Here, we present a table showing the optimal parameter values for selected models that are considered in Sections 5.3.2-5.3.4 in the main text.

Classifier	**Best Macro F1 Score**	**Best Hyperparameters**
DecisionTree	0.936432	'max_features': 'sqrt', 'splitter': 'random'
RandomForest	0.936463	'criterion': 'gini', 'max_features': 'log2', 'n_estimators': 10
Logistic Regression	0.929522	'C': 0.01, 'max_iter': 5000, 'penalty': 'l2', 'solver': 'newton-cg'

Table A1: *Best Hyperparameters for Some Classifiers Used in the Study*