The University of Melbourne

COMP90089

Machine Learning Applications for Health

Team 8

Risk Prediction Modeling for Ventilator-Associated Pneumonia (VAP) using Early Indicators in Intensive Care Unit (ICU) Patients

---

Jingwei Zhang	1067859
Jiyang XIN	1322761
Ritwik Giri	1301272
Sanskar Bhaia	1111612
Vibhuti Rajpal	1305409

Abstract

The study predicts the mortality risk of ICU patients with ventilator-associated pneumonia (VAP) from the MIMIC-IV database. Early indicators for predicting patient survival were identified using appropriate cohort selection and data preprocessing methods. The study considered early vital signs and laboratory measurements collected across all ICU stays, along with the patient's demographic information. It comprehensively describes the timeline considered for each of the indicators. The aim is to provide a practical, stepwise approach to mortality prediction and provide clusters to identify distinct patterns among VAP patients. The results show how early predictors are crucial in mortality prediction amongst VAP patients. It will empower healthcare professionals to manage and mitigate mortality risks associated with VAP in the ICU proactively, ultimately improving patient care and outcomes.

1 Introduction

Patients in the intensive care unit (ICU) are at risk of both critical illness-related mortality and comorbidities, such as hospital-acquired infections. The second most common of these is ventilator-associated pneumonia (VAP). About 25% of patients in the ICU encountered VAP-related difficulties during the COVID-19 pandemic. (Russo et al., 2022).

According to surveys, there are 250,000 to 300,000 cases in the United States each year, with 5 to 10 cases per 1,000 hospital admissions (Koenig and Truwit, 2006). VAP patients have an attributed mortality rate between 0 and 50% (Koenig and Truwit, 2006), indicating that VAP is a disease with high prevalence and high impact.

According to the clinical definition, VAP is a form of hospital-acquired pneumonia (HAP) that occurs after more than 48 hours of mechanical ventilation (Kalil et al., 2016). VAP is supported when bacterial growth exceeds a specific threshold for tracheal aspirates/ Mini-Bronchoalveolar lavage/ Protected Specimen Brush (Koenig and Truwit, 2006).

In this study, we extract, clean, and feature data from the MIMIC database to forecast the likelihood of mortality in VAP-identified patients. We selected early predictors for survival prediction based on a review of diverse research papers (Zhang et al., 2022). They not only match with clinical expertise but also capture complex insights from the VAP research.

We will exploit the power of both Supervised and Unsupervised Machine Learning (ML) approaches. Supervised ML predicts key mortality risk factors in VAP patients. We'll compare results from logistic regression, SVM, XGBoost, and Random Forest. Unsupervised ML performs clustering tasks to discern distinctions among the identified salient features.

The study empowers healthcare workers to use early indicators proactively in order to mitigate VAP-related mortality risk.

2 Methods

This section provides an overview of the methodology used for the digital phenotyping of VAP patients. It describes data preprocessing, feature selection, ML methodologies and model evaluation techniques. Finally, we discuss the ethical aspects of the dataset.

2.1 Data Source

In this study, we utilized the MIMIC-IV database (v2.2), encompassing patients from BETH Israel Deaconess Medical Centre between 2008 and 2019 (Johnson et al., 2023). Among 299,712 hospital admissions, we focused on the 16.98% (50,920 cases) necessitating ICU services, with 48.59% (24,726 cases) involving ventilation sessions.

MIMIC Modules	**Data Tables**
HOSP	admissions, d_labitems, emar, emar_detail, labevents, microbiologyevents, patients
ICU	d_items, chartevents, icustays, datetimeevents, inputevents, outputevents, procedureevents

Table 1: Data Sources by Modules

2.2 Cohort Identification

For a comprehensive analysis of VAP in the extensive MIMIC database, we first established a well-defined set of criteria to identify relevant cases. Below are the key terms that are integral to our research.

- **Ventilation Duration:** This applies to ICU patients who have been on mechanical ventilation for more than 48 hours.
- **Clinical Indicators:** For VAPs, we are considering the presence of 3 clinical indicators, Fever (Body Temperature > 38 °C), Leukocytosis (WBC $\geq$ 12,000 cells/ mm3), and Leukopenia (WBC $<$ 4,000 cells/mm3)(Kalanuria et al., 2014)
- **Microbiology Event - Pathogen Growth:** Any respiratory specimen has countable bacteria. VAP is supported as soon as bacterial growth reaches a predetermined threshold (Wimberley et al., 1979). In this research, we are considering the following thresholds:
  - High values for tracheal aspirates ($\geq$ 1,000,000 colony forming units (cfu)/mL)
  - High values for mini-BAL (mini-bronchoalveolar lavage) (10,000 cfu/mL).
  - High values for protected specimen brush (PSB) (1000 cfu/mL).

  These thresholds minimize misdiagnosing tracheobronchial colonization as VAP (i.e., low false-positive rate) (Baselski et al., 1992).

For clinical indicators and microbiology events, we considered post-48-hour ventilation session observations until the ICU end time (Figure 1).

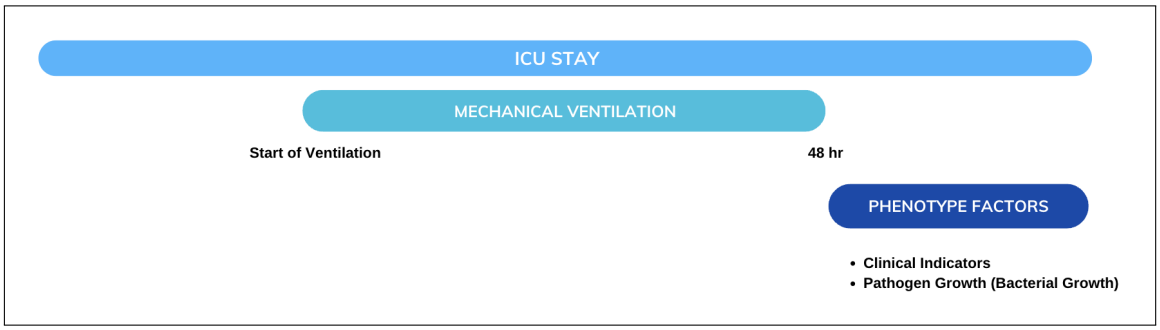


Figure 1: Phenotype Factors Timeline

A patient is diagnosed with VAP if they fall into either of the two categories.

Category 1:
ICU Patients on ventilation for over 48 hours with specified clinical indicators.

Category 2:
ICU Patients on ventilation for over 48 hours with significant microbiology events.

When considering both categories, we obtained a cohort of 3538 distinct patients (Figure 2).

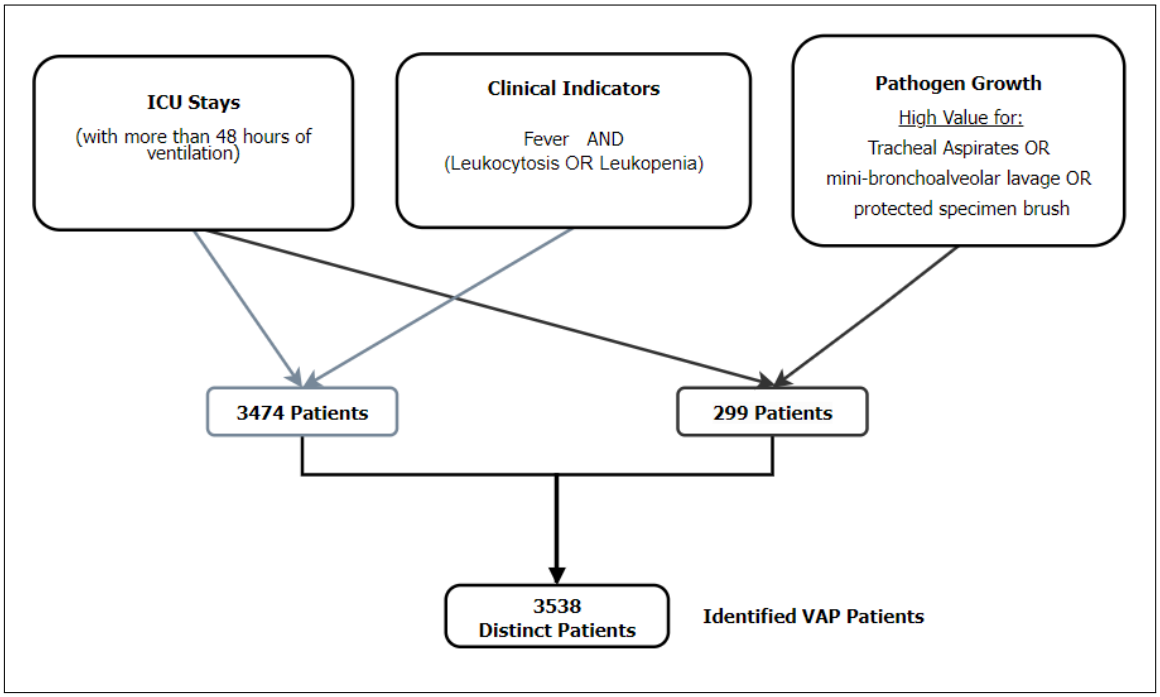


Figure 2: Cohort Identification Flowchart

2.3 Feature Selection

We analyzed all patient ICU stays and established timeframes for the features. To ensure robust data coverage, we calculated averages across all their ICU stays. Our features were categorized into 4 timeframes.

- **Hospital Admission Time:**
  - The average Comorbidity Index incorporates morbidities like diabetes, kidney/liver disease, etc., which may increase the mortality risk in VAP patients.
- **Before the start of ventilation:**
  - Average lab and vital test results from ICU stay with over 48 hours of ventilation.
- **During Ventilation:**
  - Check whether the patient received a Bronchoscopy procedure in their last ICU stay.

- **Post Ventilation:**
  - Average BMI, WBC count, and body temperature from post-ventilation to the end of the ICU stay.

Along with the above features, we have also considered the following:

- Total ICU stays
- Total ventilation stays
- Total hours of ventilation

A detailed timeline of the features is presented in the following diagram (Figure 3).

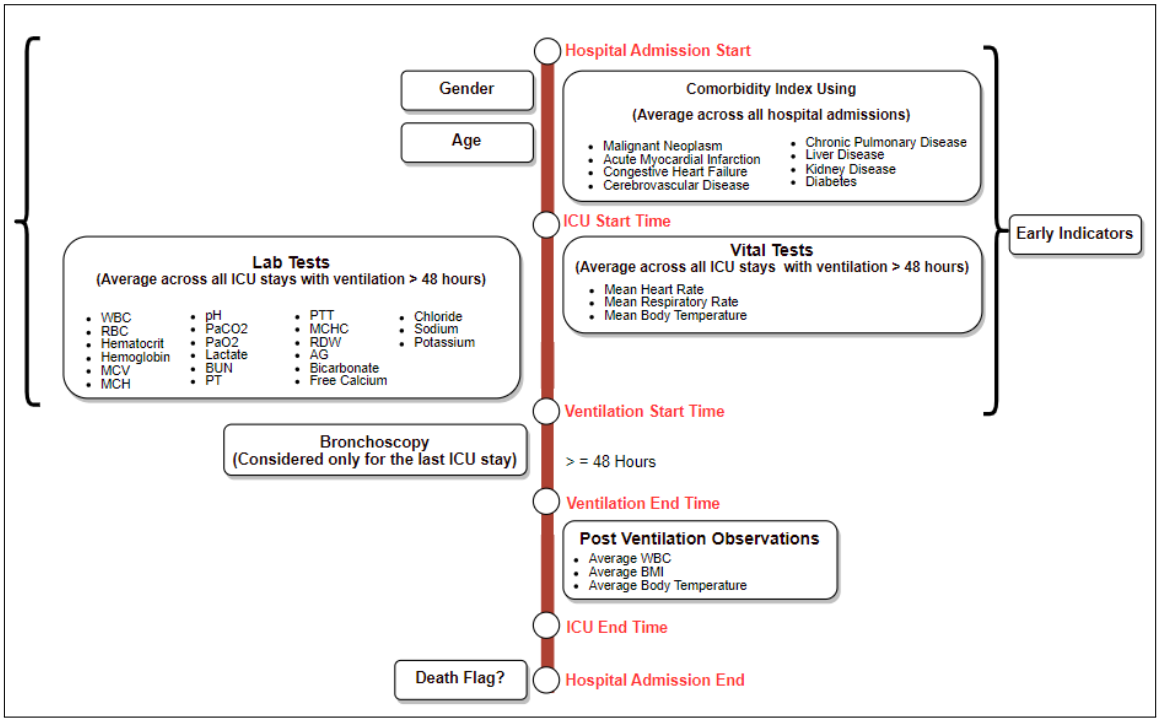


Figure 3: Feature Selection Timeline

2.4 Data Prepossessing

To ensure the completeness of the selected data, we only consider the patients with at least one record of the selected lab and vital tests (Figure 3) in between ICU in-time and ventilation start time. The cohort was reduced to 2290 distinct patients.

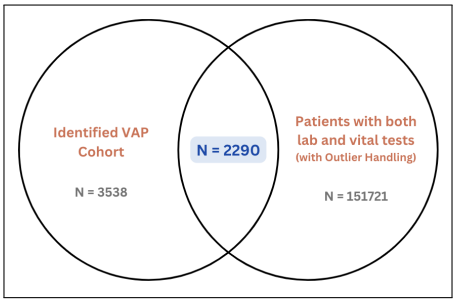


Figure 4: Reduced Cohort

2.4.1 Handling Null Values

We exclude features with over 50% null values. For the remaining features, we fill null values by imputing means to retain valuable information.

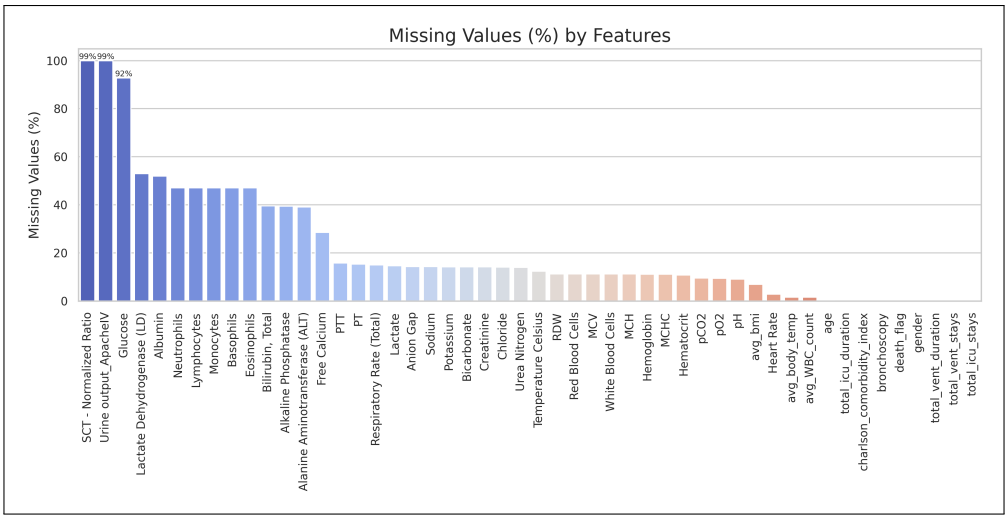


Figure 5: Missing Values

2.4.2 Column Mapping Process

Patients with known date of death are marked as 1 in the 'death_flag' column, our response variable. Gender is encoded as 0 for males and 1 for females.

2.4.3 Handling High Correlation

Highly correlated features may overfit ML models by sharing redundant information. We eliminate one column from each strongly correlated pair for our classification model (Table 2).

	Feature 1	Feature 2	Correlation
0	Hematocrit	Hemoglobin	0.97
1	Hematocrit	Red Blood Cells	0.92
2	Red Blood Cells	Hemoglobin	0.9
3	total_icu_duration	total_vent_duration	0.87
4	MCV	MCH	0.84
5	Sodium	Chloride	0.73
6	Alanine Aminotransferase (ALT)	Lactate Dehydrogenase (LD)	0.69
7	Urea Nitrogen	Creatinine	0.64
8	total_vent_stays	total_icu_stays	0.64
9	total_vent_stays	total_icu_duration	0.64

Table 2: Top Correlated Traits

2.4.4 Feature Scaling

We standardize all numeric columns using the Standard Scaler to ensure consistent scaling. It enables fair comparisons among different features with varying units and scales.

2.4.5 Feature Significance

At the end of the preprocessing stage, we identify the most significant features i.e. features that highly impact the mortality. This step will reduce the possible overfitting.

We have used the Recursive Feature Elimination (RFE) algorithm to get the top 15 scaled features identified by the logistic regression, SVM, Random Forest, and XGBoost estimators.

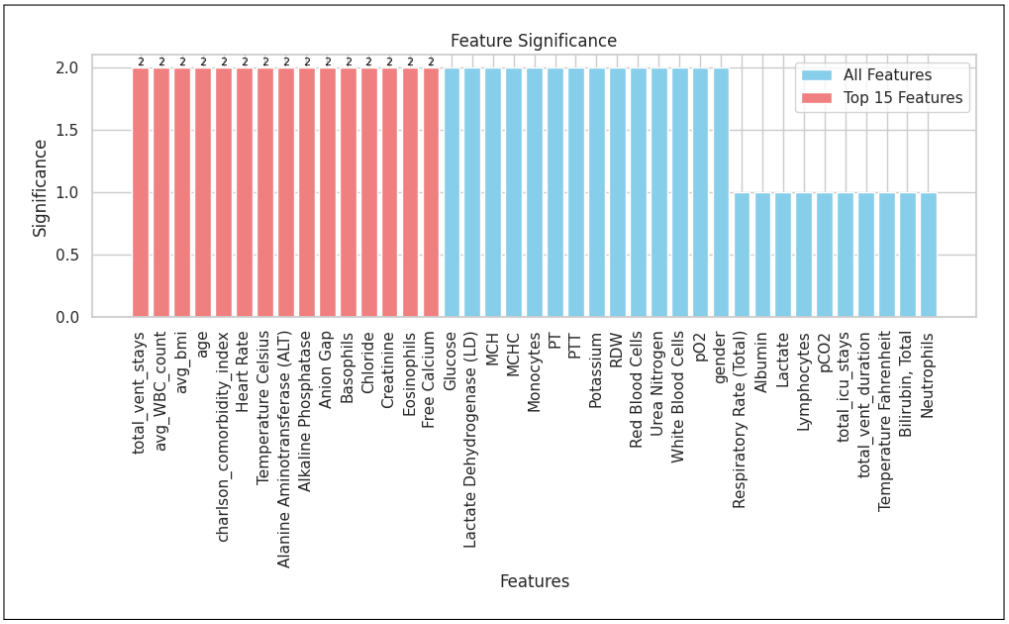


Figure 6: Feature Significance

2.5 Machine Learning Methods

We are addressing the problem of predicting mortality risk in VAP patients. We are building classification models (Logistic, SVM, Random Forest, XGBoost) to classify patients based on their survival status. These models have previously demonstrated strong recognition rates for VAP research (Liao et al., 2019). These classification models are adequate for our high-dimensional dataset (features = 15).

Additionally, we're applying K-means clustering to group patients based on early indicators and bronchoscopy procedure status. It helped us identify patterns and analyze how different features relate to mortality risk within these clusters. PCA was used for dimensionality reduction for Cluster Visualisation.

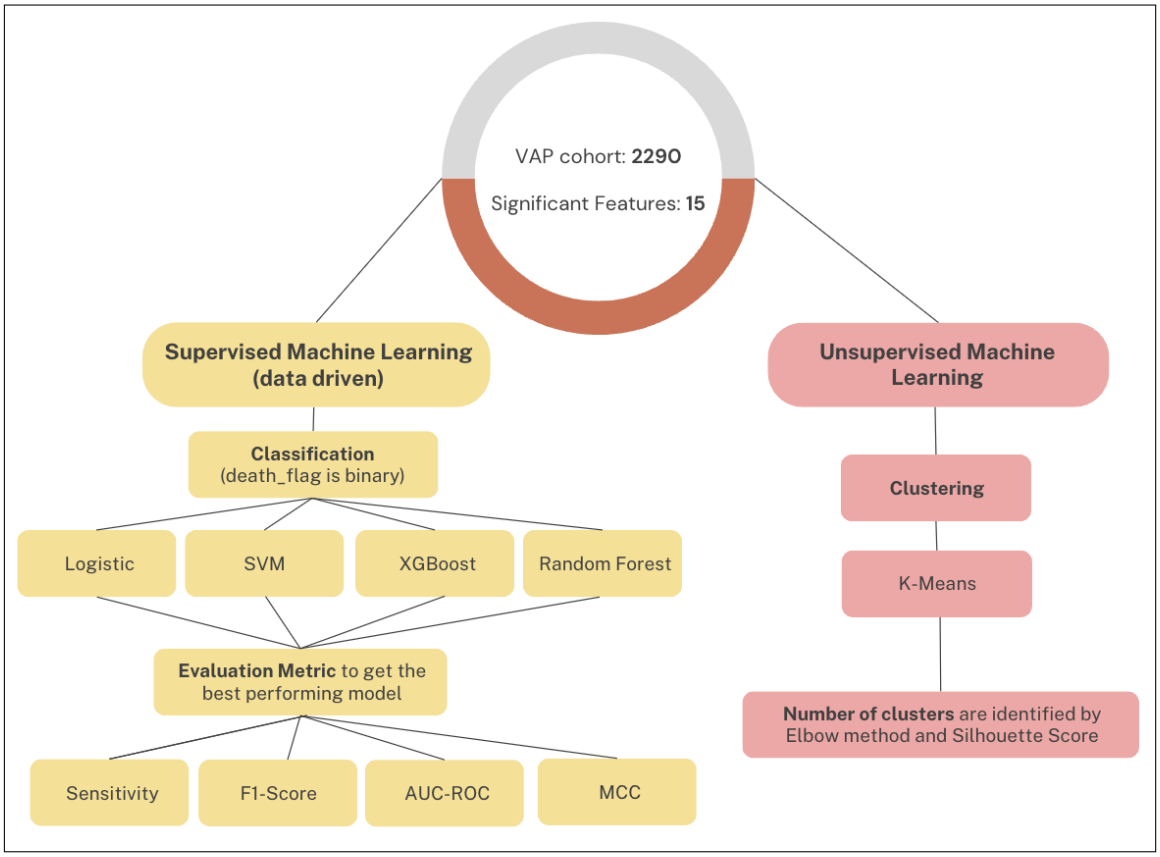


Figure 7: Machine Learning Approaches

2.6 Evaluation Methods

In our problem, we have an imbalanced distribution of the binary response variable (death_flag) (Figure 8). Hence, using accuracy as the primary performance metric may be misleading as it does not account for the class distribution.

To address this, we focus on the F1-score, AUC-ROC, and MCC. These metrics are more appropriate for evaluating the model's ability to accurately predict the 'death' category while considering the class imbalance, which is crucial in this medical context.

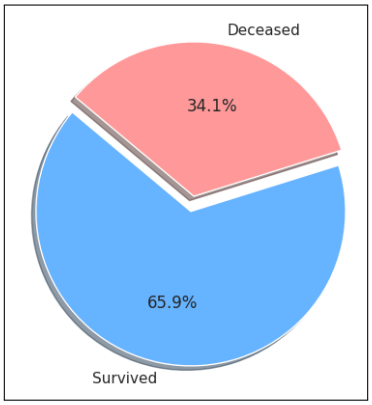


Figure 8: Class Distribution of Patient Outcomes

F1-Score is a balanced metric for imbalanced classes. It combines precision and recall and provides a harmonic mean that accounts for both false positives (FP) and false negatives (FN).

$$F1\ score = \frac{2 \cdot TP}{2 \cdot TP + FP + FN} \tag{1}$$

AUC-ROC evaluates the model's ability to distinguish between classes without being affected by class distribution, making it robust in scenarios where one class is rare.

MCC considers true and false positives and negatives, providing a reliable measure of model performance that's unaffected by class size differences.

$$MCC = \frac{(TP \times TN) - (FP \times FN)}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}} \tag{2}$$

2.7 Ethical Considerations

When using medical data, data sharing must protect patient privacy. The MIMIC database is de-identified, complying with HIPAA Safe Harbor provisions (Moosavi, 2022). The MIMIC resource was reviewed and approved by the Institutional Review Board at Beth Israel Deaconess Medical Center, waiving the need for informed consent in the data-sharing program. (Johnson et al., 2023).

3 Results

The results section comprises three key segments. We start with Exploratory Data Analysis to understand the cohort. Next, we evaluate various classification models and perform feature importance analysis. Finally, we present clustering outcomes to identify patient subgroups. Data is divided into 80% training and 20% testing sets to assess model performance.

3.1 Exploratory Data Analysis (EDA)

The cohort obtained at 2.4 was used to obtain the following EDA results.

Metric	Total	Died	Survived
Cohort Information			
Total Patients	2290	780	1510
Gender Information			
Males	1391	456	935
Females	897	324	573
Demographic Information (Average Age in years)			
All Patients	60.75	64.40	58.85
Patients with Bronchoscopy	60.26	63.48	58.67
Females	61.82	65.00	60.02
Males	60.05	63.98	58.14
Procedure Information			
Patients with Bronchoscopy	1077	355	721

Table 3: Summary of Cohort Characteristics

Mortality Information	%
Males Who Died	32.78 %
Females Who Died	36.12 %

Table 4: Gender Mortality Information

3.1.1 Cohort Overview

Within a cohort of 2290 patients, we observed a 65.9% survival rate (1510). However, 34.06% of patients (780) did not survive.

3.1.2 Bronchoscopy's Impact

Patients who underwent bronchoscopy procedures during ventilation had a promising 67% survival rate, with survivors averaging 58.8 years, notably younger than non-survivors, who had an average age of 64.4 years.

3.1.3 Gender and Age Differences

The cohort had a slight male predominance (60.8%) with comparable average ages for male and female ICU patients at 60 and 61.8 years, indicating no significant age difference between the genders.

3.2 Classification Results

In the context of our binary classification results (Table 5), it is evident that the SVM algorithm demonstrates superior performance compared to other algorithms. SVM achieves higher scores in all the key metrics. Notably, logistic regression also delivers robust performance, closely following SVM. In contrast, XGBoost and Random Forest underperformed.

Method	AUC-ROC	F1-Score	MCC	Sensitivity	Balanced Accuracy
Logistic	0.68	0.53	0.27	0.68	0.65
SVM	**0.71**	**0.56**	**0.32**	**0.70**	**0.67**
XGBoost	0.67	0.48	0.25	0.48	0.63
Random Forest	0.66	0.36	0.25	0.26	0.59

Table 5: Binary Classification Results of Different Models

The AUC value of 0.71 indicates that the SVM model has reasonably good discriminative power in distinguishing between survival and mortality outcomes (Figure 9). An F1 score of 0.56 and Sensitivity of 0.70 indicate the model's strength in identifying mortality cases. An MCC value of 0.32 can be interpreted as the model's predictions having a moderate agreement with the actual outcomes.

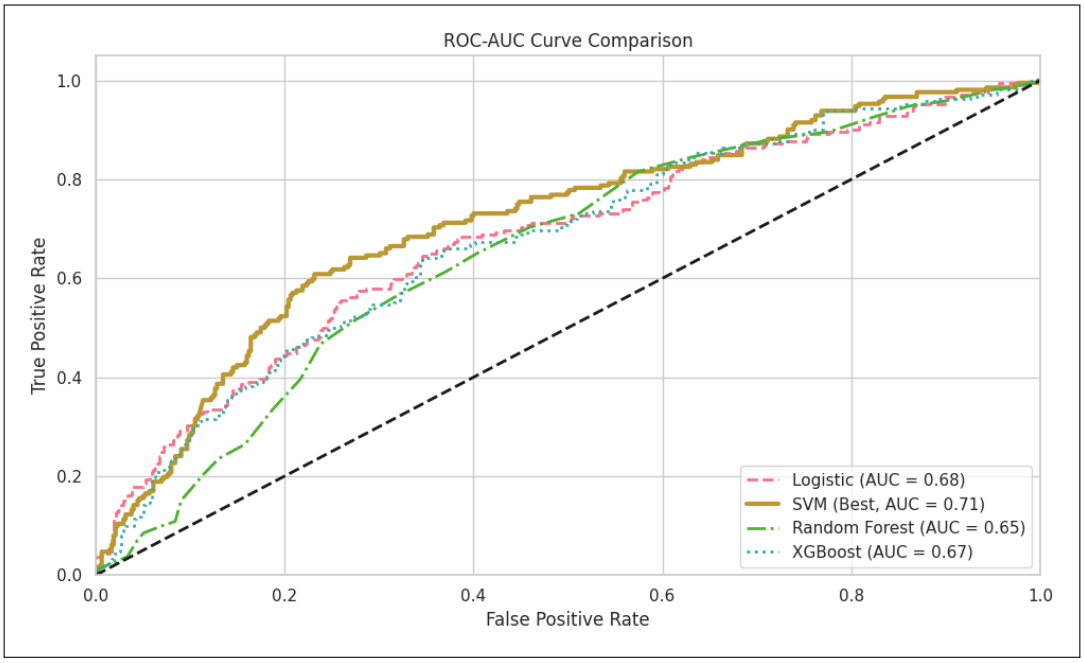


Figure 9: ROC-AUC Comparison of Different Models

3.3 Feature Importance Analysis

We have categorized the features based on their observed timeline. For each feature type, we performed feature importance analysis using the SVM and Logistic estimators (Figure 10).

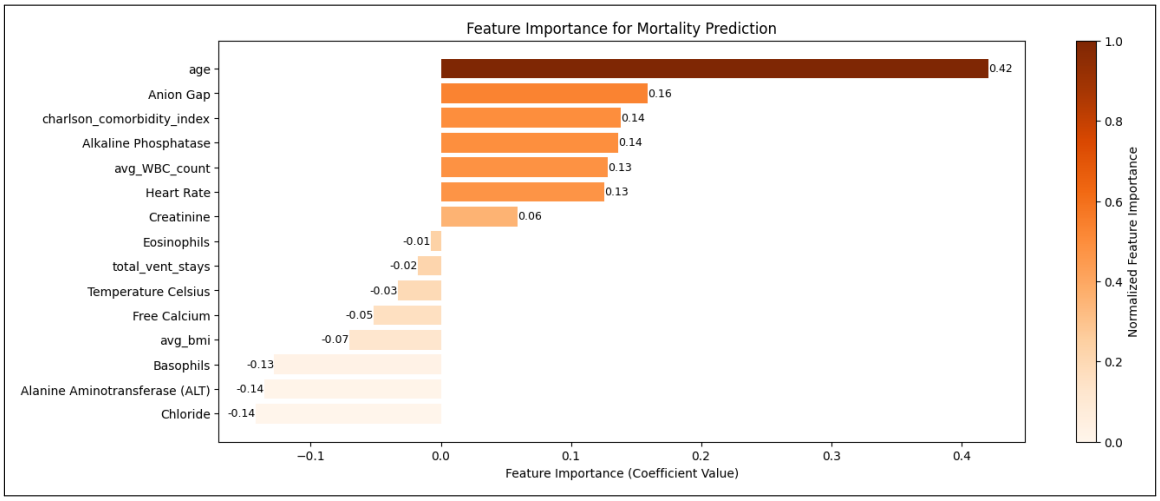


Figure 10: Feature Importance Analysis

3.3.1 Early Indicators

Elevated Anion Gap and Alkaline Phosphatase levels significantly increase VAP mortality risk, while Creatinine, Eosinophils, Basophils, and Free Calcium levels have a less pronounced impact. Lower Chloride levels also modestly contribute to increased VAP mortality risk.

3.3.2 General Information

Age, comorbidity index, and ventilator duration are key indicators. Higher age and comorbidity index, along with prolonged ventilator support, elevate VAP mortality risks.

3.3.3 Post-Ventilation Observation

Post-ventilation, a higher average WBC Count elevates VAP mortality risk. A lower average BMI slightly increases the risk, with a relatively low impact.

A detailed impact analysis of all the features is provided in the Appendix 2.

3.4 Clustering Results

K-means clustering sought distinct trends among VAP patients, particularly evaluating the impact of the Bronchoscopy procedure, as depicted in Figure 3. The influence of Early Indicators and Post-Ventilation observations was also assessed to identify high-risk patients.

Principal Component Analysis (dimensionality reduction) was used to visualise the 3 clusters identified by K-Means (Figure 11).

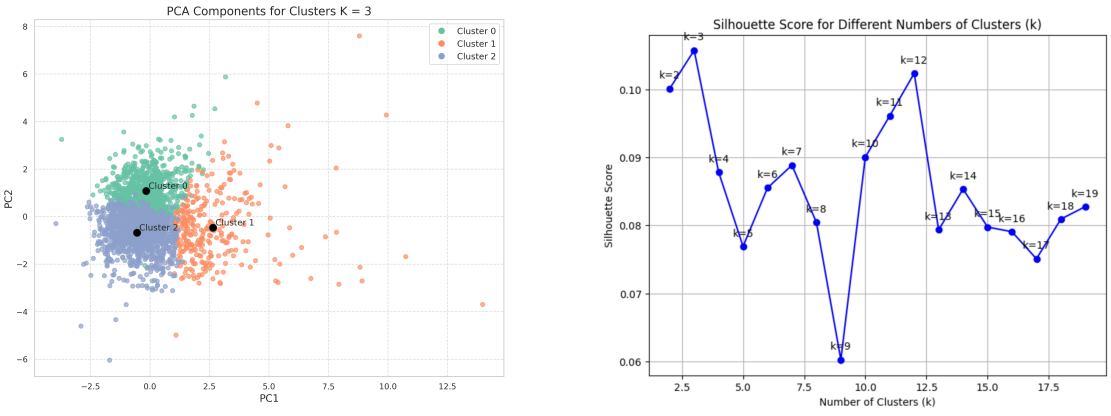


Figure 11: Cluster Visualisation for K = 3

Cluster	Cluster Summary	Post-Ventilation Observation	General Information	Early Indicator Results	Mortality Rate	Without Bronchoscopy
0	Mid-Range Low Risk Patients	Shorter ICU stays, Fewer ventilation stays, Moderate BMI	Moderately aged, Low comorbidity	Anion Gap (Within), Chloride (Within), Creatinine (Not Within)	Moderate	About Half
1	Older Adult High Risk Patients	Longer ICU stays, More ventilation stays, Similar BMI	Older age, High comorbidity	Anion Gap (Not Within), Chloride (Within), Creatinine (Not Within)	Higher	Significant Portion
2	Elderly Low-Moderate Risk Patients	Similar to Cluster 1, Slightly lower post-ventilation	Oldest age, Moderate comorbidity	Anion Gap (Within), Chloride (Within), Creatinine (Within)	Lower	Balanced

Table 6: Cluster Analysis summary

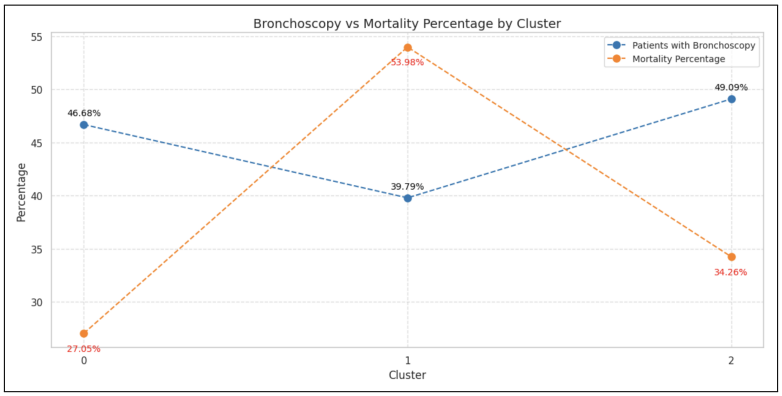


Figure 12: Relationship between Bronchoscopy and Mortality in VAP Patients

The key observations are listed in the (Table 6). It depicts how the early indicators results affect the mortality and how the bronchoscopy procedure might reduce the mortality of the VAP patients (Figure 12). Box plots are provided for further comparisons for all feature categories (Figure 13 ,14 ,15).

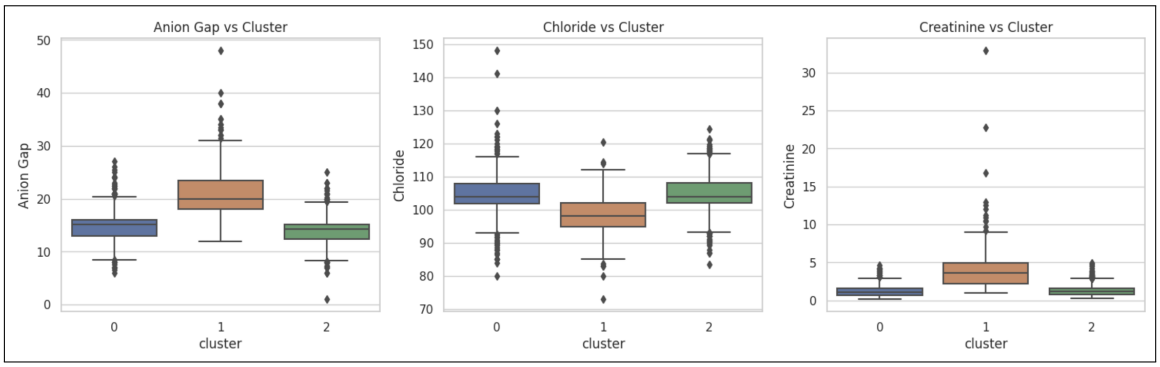


Figure 13: Box Plots for Early Predictors grouped by Clusters

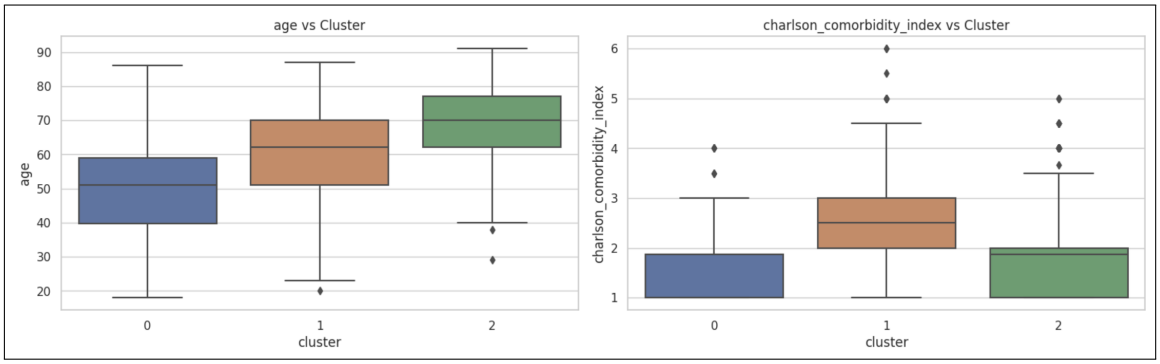


Figure 14: Box Plots for General Information grouped by Clusters

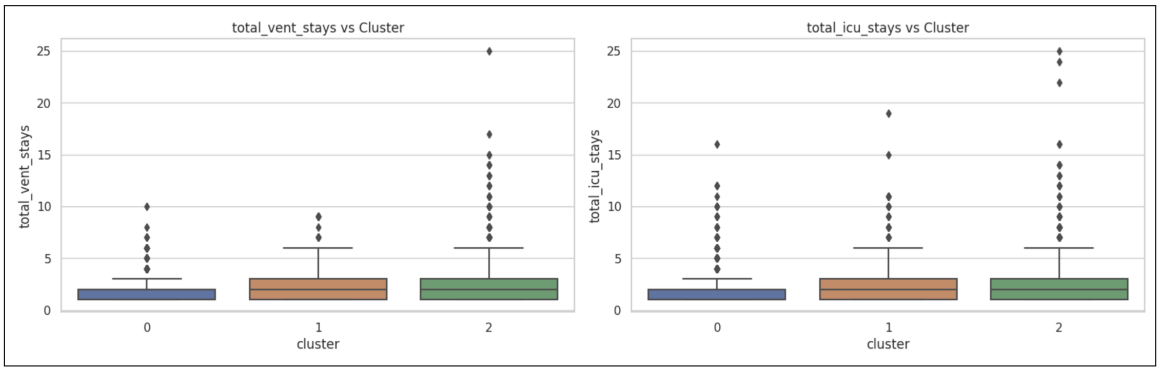


Figure 15: Box Plots for Post-Ventilation Observation grouped by Clusters

4 Discussion and Conclusion

In the absence of standardized diagnostic criteria, Ventilator-Associated Pneumonia (VAP) poses a significant diagnosis challenge. As a result of our study, we demonstrate how early indicators may be able to predict the likelihood of VAP, facilitating timely intervention and reducing complications. The study examined a range of indicators that previous research has identified as contributing factors to VAP diagnosis at different points during patients' Intensive Care Unit (ICU) stays.

Using classification models for VAP patients, we explored the influence of early indicators on mortality risk and identified the most critical early warning signs. In particular, we found that Anion Gap, Chloride, and Creatinine had a high impact on patient outcomes.

When the anion gap increases, there is likely to be an acid-base imbalance that is associated with VAP. High Creatinine levels can suggest kidney injury associated

with VAP, while high chloride levels can indicate fluid status. The relevance of these markers in VAP management has been demonstrated in numerous studies investigating their diagnostic and prognostic capabilities (Dafal et al., 2021; Pfortmueller et al., 2018; Murugan et al., 2010). Critical care can be significantly improved by applying this knowledge, potentially reducing fatality rates.

To conclude, our study utilized clustering techniques to emphasize the importance of bronchoscopy procedures at the time of ventilation. It is consistent with Zhang et al.'s observation in their study (Zhang et al., 2022) that patients with Bronchoscopy exhibit lower mortality rates.

There are, however, challenges and limitations associated with the development of machine learning models for predicting VAP patient mortality. A limitation of the MIMIC database is that it lacks data that specifies the exact time at which the patient contracted VAP. Additionally, the specific nature of the MIMIC database may restrict our ability to provide a universally applicable solution.

Prediction tasks can be performed in future studies using data collected immediately after mechanical ventilation. Access to such data would allow healthcare professionals to intervene and potentially improve patient outcomes before the 48-hour mark.

Furthermore, our study relied on one source of data. It limits the generalizability of our findings. Due to the retrospective nature of the study, it is difficult to assess how this algorithm might impact patient outcomes in a real clinical setting. Future research needs to be prospectively validated to guarantee that the findings can be applied to clinical practice in a meaningful way.

References

[1] V. S. Baselski, M. el Torky, J. J. Coalson, and J. P. Griffin. The standardization of criteria for processing and interpreting laboratory specimens in patients with suspected ventilator-associated pneumonia. *Chest*, 102:571S, 1992. doi: 10.1378/chest.102.5_supplement_1.571s.

[2] Akshay Dafal, Sunil Kumar, Sachin Agrawal, Sourya Acharya, and Apoorva Nirmal. Admission anion gap metabolic acidosis and its impact on patients in medical intensive care unit. *Journal of Laboratory Physicians*, 13(02):107–111, 2021.

[3] A. Johnson, L. Bulgarelli, T. Pollard, S. Horng, L. A. Celi, and R. Mark. Mimic-iv (version 2.2). *Circulation [Online]*, 101:e215–e220, 2023. doi: 10.13026/6mm1-ek67.

[4] Atul A Kalanuria, William Zai, and Marek Mirski. Ventilator-associated pneumonia in the icu. *Critical Care*, 18:208, 2014. doi: 10.1186/cc13775.

[5] Andre C. Kalil, Mark L. Metersky, Michael Klompas, and et al. Management of adults with hospital-acquired and ventilator-associated pneumonia: 2016 clinical practice guidelines by the infectious diseases society of america and the american thoracic society. *Clinical Infectious Diseases*, 63:e61, 2016. doi: 10.1093/cid/ciw353.

[6] Steven M Koenig and Jonathon D Truwit. Ventilator-associated pneumonia: diagnosis, treatment, and prevention. *Clinical microbiology reviews*, 19:637–657, 2006. doi: 10.1128/CMR.00051-05.

[7] Y. H. Liao, Z. C. Wang, F. G. Zhang, M. F. Abbod, C. H. Shih, and J. S. Shieh. Machine learning methods applied to predict ventilator-associated pneumonia with pseudomonas aeruginosa infection via sensor array of electronic nose in intensive care unit. *Sensors (Basel, Switzerland)*, 19:1866, 2019. doi: 10.3390/s19081866.

[8] Vahid Moosavi. Unlocking pre-trained models for natural language processing: A practical guide. *Scientific Data*, 9:1–12, 2022. doi: 10.1038/s41597-022-01899-x.

[9] Raghavan Murugan, Vijay Karajala-Subramanyam, Minjae Lee, Sachin Yende, Lan Kong, Melinda Carter, Derek C Angus, John A Kellum, et al. Acute kidney injury in non-severe pneumonia is associated with an increased immune response and lower survival. *Kidney international*, 77(6):527–535, 2010.

[10] Carmen Andrea Pfortmueller, Dominik Uehlinger, Stephan von Haehling, and Joerg Christian Schefold. Serum chloride levels in critical illness—the hidden story. *Intensive care medicine experimental*, 6:1–14, 2018.

[11] A. Russo, V. Olivadese, E. M. Trecarichi, and C. Torti. Bacterial ventilator-associated pneumonia in covid-19 patients: Data from the second and third waves of the pandemic. *Journal of Clinical Medicine*, 11:2279, 2022. doi: 10.3390/jcm11092279.

[12] Norman Wimberley, Leonard J Faling, and John G Bartlett. A fiberoptic bronchoscopy technique to obtain uncontaminated lower airway secretions for bacterial culture. *American Review of Respiratory Disease*, 119:337, 1979. doi: 10.1164/arrd.1979.119.3.337.

[13] Lu Zhang, Shuang Li, Shuai Yuan, Xiaojie Lu, Jie Li, Yang Liu, Ting Huang, Jie Lyu, and Hongjun Yin. The association between bronchoscopy and the prognoses of patients with ventilator-associated pneumonia in intensive care units: A retrospective study based on the mimic-iv database. *Frontiers in pharmacology*, 13:868920, 2022. doi: 10.3389/fphar.2022.868920.

Appendix 1

Team Contributions

Name	Student ID	Contributions
Ritwik Giri	1301272	Conceptualization Data Curation Formal Analysis Methodology Project administration Software Visualization Writing – review & editing
Vibhuti Rajpal	1305409	Conceptualization, Visualization, Writing – original draft, Writing – review & editing
Jingwei Zhang	1067859	Conceptualization Data curation Formal Analysis Software Validation Writing – review & editing
Sanskar Bhatia	1111612	Conceptualization Data curation Formal Analysis Software Visualization Validation Writing – review & editing
Jiyang Xin	1322761	Conceptualization, Visualization, Validation Writing – review & editing

Table 7: Team Contributions

Appendix 2

Feature impact on VAP Related Mortality

Feature Type	Feature	Feature Definition	Associated Mortality Risk
Early Indicators	Age	Patient's chronological age	Higher age associated with increased VAP mortality risk
Early Indicators	Anion Gap	Difference between cations and anions indicating metabolic acidosis	Elevated anion gap increases VAP mortality risk
Early Indicators	Alkaline Phosphatase	Enzyme found in the liver and bones	Elevated levels increase VAP mortality risk related to liver or bone diseases
Early Indicators	Heart Rate	Number of heartbeats per minute	Elevated rate increases VAP mortality risk due to cardiac stress
Early Indicators	Creatinine	Waste product indicating kidney function	Elevated levels increase VAP mortality risk due to kidney dysfunction
Early Indicators	Eosinophils	Type of white blood cell involved in the immune response	Lower counts increase VAP mortality risk due to immune system imbalance
Early Indicators	Temperature	Body temperature in degrees Celsius	Lower temperature increases VAP mortality risk due to health concerns
Early Indicators	Free Calcium	Calcium not bound to proteins	Lower levels increase VAP mortality risk affecting muscle and nerve function
Early Indicators	Basophils	Type of white blood cell involved in allergic reactions	Lower counts increase VAP mortality risk due to immune system imbalance
Early Indicators	ALT	Enzyme found in the liver	Higher ALT levels increase VAP mortality risk due to liver damage
Early Indicators	Chloride	Electrolyte maintaining acid-base balance	Lower levels increase VAP mortality risk due to metabolic imbalances
General Information	Comorbidity Index	Assesses comorbid conditions and their impact on mortality	Higher index increases VAP mortality risk due to comorbid conditions
General Information	Total Ventilator Stays	Number of times on a ventilator	More stays increase VAP mortality risk related to respiratory issues
Post Ventilation Observation	Average WBC Count	Average white blood cell count in the blood	Higher count increases VAP mortality risk related to infection or inflammation
Post Ventilation Observation	Average BMI	Average body mass index of a patient	Lower BMI increases VAP mortality risk due to underlying health issues

Table 8: Detailed Feature Analysis

Appendix 3

Abbreviations

- **AG:** Anion Gap
- **ALT:** Alanine Aminotransferase
- **AUC-ROC:** Area Under the Curve-Receiver Operating Characteristic
- **BMI:** Body Mass Index
- **BUN:** Blood Urea Nitrogen
- **CFU:** Colony-Forming Unit
- **FN:** False Negative
- **FP:** False Positive
- **HIPAA:** The Health Insurance Portability and Accountability Act
- **HAP:** Hospital-Acquired Pneumonia
- **ICU:** Intensive Care Unit
- **MCC:** Matthews Correlation Coefficient
- **MCH:** Mean Corpuscular Hemoglobin
- **MCHC:** Mean Corpuscular Hemoglobin Concentration
- **MCV:** Mean Corpuscular Volume
- **MIMIC:** Medical Information Mart for Intensive Care
- **ML:** Machine Learning
- **PaCO2:** Partial Pressure of Carbon Dioxide in Arterial Blood
- **PaO2:** Partial Pressure of Oxygen in Arterial Blood
- **PCA:** Principal Component Analysis
- **PSB:** Protected Specimen Brush
- **PT:** Prothrombin Time
- **PTT:** Partial Thromboplastin Time
- **RBC:** Red Blood Cells
- **RFE:** Recursive Feature Elimination
- **RDW:** Red Blood Cell Distribution Width
- **SVM:** Support Vector Machine
- **TN:** True Negative
- **TP:** True Positive
- **VAP:** Ventilator-Associated Pneumonia
- **WBC:** White Blood Cells

Appendix 4

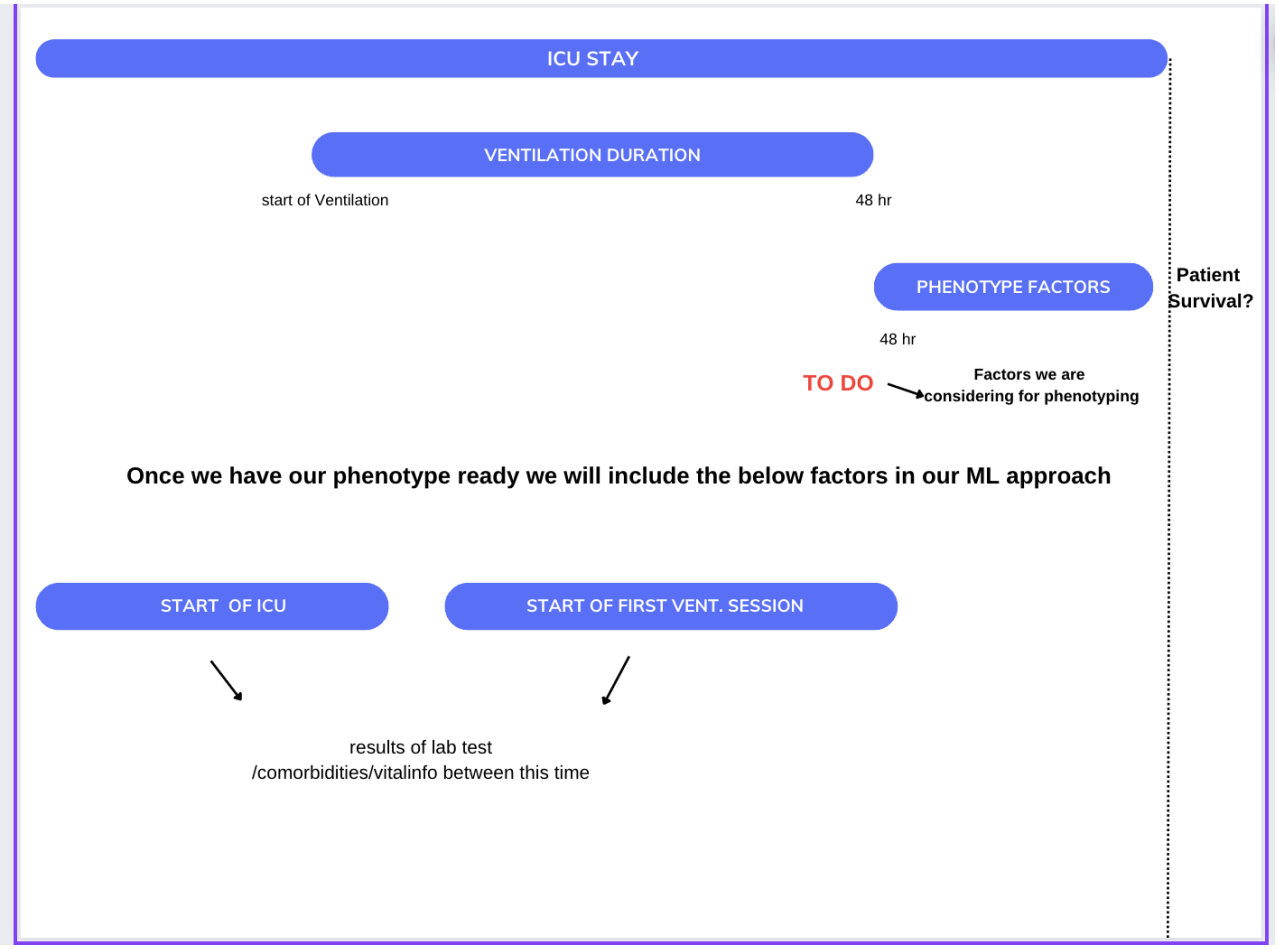
Project Notebook

Python notebook below explains the code in detail.

Importing required packages

In [3]:
```
# Importing required packages and function to connect to GCP

import pandas as pd  # Import the Pandas library for data manipulation.
import os  # Import the OS library for working with the operating system.F
from datetime import timedelta  # Import timedelta from datetime for time-related calculations.

# Access data using Google BigQuery.
from google.colab import auth  # Import the authentication module from Google Colab.
from google.cloud import bigquery  # Import the BigQuery client library.

# Authenticate the user to access Google BigQuery data.
auth.authenticate_user()

# Set up environment variables for the Google Cloud project.
project_id = 'physionet-data-395201'  # Replace with your own project ID.
if project_id == 'physionet-data':
    raise ValueError('You must change project_id to your GCP project.')

os.environ["GOOGLE_CLOUD_PROJECT"] = project_id  # Set the project ID as an environment variable.

# Read data from BigQuery into Pandas dataframes.
def run_query(query, project_id=project_id):
    return pd.io.gbq.read_gbq(
        query,
        project_id=project_id,
        dialect='standard'
    )

# Set the dataset you want to work with in BigQuery.
dataset = 'mimiciv'
```

Project Flow and stages



Digital Phenotyping

Key terms related to digital phenotyping

Ventilator-Associated Pneumonia (VAP) Definition for Phenotyping:

In the context of this analysis, Ventilator-Associated Pneumonia is defined as follows:

1. Mechanical Ventilation Duration:

- **Duration >= 48 Hours**: Patients who have been on mechanical ventilation for a duration of 48 hours or longer.

2. Clinical Indicators:

- **Leukocytosis and Fever**: Patients who exhibit leukocytosis (an elevated white blood cell count) or leukopenia (low WBC) and fever after 48 hours of ventilation and before the end of their ICU stay.

3. Microbiology Events (bacterial growth):

- **High Value for Tracheal Aspirates**: Patients with microbiology events showing high values for tracheal aspirates.
- **High Value for Mini-BAL**: Patients with microbiology events showing high values for mini-BAL (mini-bronchoalveolar lavage).
- **High Value for PSB**: Patients with microbiology events showing high values for protected specimen brush (PSB).

These criteria are used for phenotyping and identifying cases of Ventilator-Associated Pneumonia (VAP) within the dataset.

VAP diagnosis conditions

Final Phenotype Identification

In the final phenotype identification, we are looking for patients who meet one of the following sets of criteria:

1. **Mechanical Ventilation Duration** and **Clinical Indicators**: Patients who have a prolonged duration of mechanical ventilation, and they exhibit specific clinical indicators such as Leukocytosis and Fever.
    - *OR*
2. **Mechanical Ventilation Duration** and **Microbiology Events**: Patients who have a prolonged duration of mechanical ventilation and experience significant microbiology events, specifically high values in tests like tracheal aspirates, mini-BAL, or PSB (Protected Specimen Brush).

Step 1: First we find the patients with ventilation more than 48 hours

Step 1a: To Find Patients with ICU Stays > 48 Hours

Step Description: In this step, we focus on identifying patients who have spent more than 48 hours in the intensive care unit (ICU).

Methodology:

1. We begin by considering patients in the ICU setting.
2. The primary objective is to distinguish patients who have had ICU stay of more than 48 hours.

A patient undergoes ventilation inbetween the ICU stays. **Hence, to get the patients with more than 48 hrs of ventilation, we first find patients with ICU time greater than 48 hours.**

In [4]:
```
# Query to find patients with ICU stays greater than 48 hours

# Defining the SQL query
icu_admission_query = '''
SELECT
    subject_id,                          -- Patient identifier
    icu_stay.intime AS icu_intime,       -- Admission time to the ICU
    icu_stay.outtime AS icu_outtime,     -- Discharge time from the ICU
    icu_stay.los AS los,                 -- Length of stay in the ICU
    icu_stay.rn AS icu_seq,              -- ICU Stay Sequence
    icu_stay.total_icu_stays             -- Total number of ICU stays for the patient (>24hrs)
FROM (
    SELECT
        subject_id,
        hadm_id,
        first_careunit,
        last_careunit,
        intime,
        outtime,
        los,
        ROW_NUMBER() OVER (PARTITION BY subject_id ORDER BY intime ASC) AS rn,
        COUNT(*) OVER (PARTITION BY subject_id) AS total_icu_stays
    FROM `physionet-data.mimiciv_icu.icustays`
) AS icu_stay
WHERE icu_stay.los >= 2 -- Filter for ICU stays longer than 48 hours
ORDER BY subject_id;
'''

# Running the query and storing the result in a DataFrame
icu_admission_table = run_query(icu_admission_query)

# Displaying the first few rows of the resulting DataFrame
icu_admission_table.head()
```

Out[4]:

	subject_id	icu_intime	icu_outtime	los	icu_seq	total_icu_stays
0	10001884	2131-01-11 04:20:05	2131-01-20 08:27:30	9.171817	1	1
1	10002155	2129-08-04 12:45:00	2129-08-10 17:02:38	6.178912	1	3
2	10002155	2130-09-24 00:50:00	2130-09-27 22:13:41	3.891447	2	3
3	10002348	2112-11-30 23:24:00	2112-12-10 18:25:13	9.792512	1	1
4	10002428	2156-04-12 16:24:18	2156-04-17 15:57:08	4.981134	1	4

Step 1b: To Find Patients with Ventilation Stays > 48 Hours

Step Description: In this step, we focus on identifying patients who have spent more than 48 hours in the ventilation sessions.

Methodology:

1. We begin by considering patients in the "FinishedRunning" stage of ventilation.
2. The primary objective is to distinguish patients who have had ventilation sessions of more than 48 hours.

Notes:

1. We are considering all the ventilation types, i.e. Invasive, Non-Invasive and Mask.
2. In the modelling stage, we will use ventilation type as a factor to see if the **impact of the ventilation type on the mortality.**

In [5]:
```
#Create a Common Table Expression (CTE) named VentilationItemID
#This CTE selects item IDs and labels from the d_items table for items related to ventilation.
patients_with_vent_session = run_query("""

WITH VentilationSessions AS (
SELECT
    pe.subject_id,                      -- Patient identifier
    pe.starttime AS vent_starttime,      -- Start time of the procedure
    pe.endtime AS vent_endtime,          -- End time of the procedure
    vid.label AS ventilation_type,       -- Label description for the ventilation item
    CASE
        WHEN pe.value >= 2880 THEN pe.value -- 48hours has 2880 mins
        ELSE NULL
    END AS vent_duration,               -- Duration of the procedure (null for values < 2880)
    pe.valueuom AS duration_unit,        -- Unit of measurement for duration
    pe.patientweight,                   -- Patient's weight at the time of the procedure
    ROW_NUMBER() OVER (PARTITION BY pe.subject_id ORDER BY pe.starttime) AS vent_seq_no,
    COUNT(*) OVER (PARTITION BY pe.subject_id) AS total_vent_stays
FROM `physionet-data.mimiciv_icu.procedureevents` AS pe
JOIN (
    SELECT itemid, label
    FROM `physionet-data.mimiciv_icu.d_items`
    WHERE LOWER(label) LIKE '%ventilation%'
) AS vid ON vid.itemid = pe.itemid
WHERE pe.statusdescription LIKE 'FinishedRunning'
ORDER BY pe.subject_id, pe.starttime)

SELECT  subject_id,
        vent_starttime,
        vent_endtime,
        ventilation_type,
        vent_duration,
        duration_unit,
        patientweight,
        vent_seq_no,
        total_vent_stays
FROM VentilationSessions
WHERE vent_duration IS NOT NULL


""")
patients_with_vent_session.head()
```

Out[5]:

	subject_id	vent_starttime	vent_endtime	ventilation_type	vent_duration	duration_unit	patientweight	vent_seq_no	total_vent_stays
0	10001884	2131-01-13 04:00:00	2131-01-19 17:45:00	Invasive Ventilation	9465.0	min	65.0	3	4
1	10001884	2131-01-15 04:07:00	2131-01-19 17:43:00	Invasive Ventilation	6576.0	min	65.0	4	4
2	10002428	2156-04-19 20:10:00	2156-04-22 17:05:00	Invasive Ventilation	4135.0	min	43.0	1	3
3	10002428	2156-05-11 16:05:00	2156-05-20 10:45:00	Invasive Ventilation	12640.0	min	48.4	3	3
4	10003400	2137-02-25 23:37:00	2137-02-28 14:17:00	Invasive Ventilation	3760.0	min	93.0	1	4

Explanation of the code's purpose and logic

Step 1c: We combine the results obtained in step 1a and 1b.

Step Description: In this step, we focus on filtering the patients who have had ICU stays greater than 48hrs AND ventilation sessions greater than 48 hours

Methodology:

1. We combined the two dataframes obtained in step 1a and step 1b.
2. Intuitively, the ventilation stay should start between the ICU stays. But, there are some patients whose ventilation started some minutes before their ICU intime. **Hence, i have added a delta of 15 mins**

Result:

1. The resulting table combines information about patients' ICU stays and ventilation sessions. For each ICU stay lasting more than 48 hours, the table will include rows corresponding to all ventilation sessions that lasted more than 48 hours during that specific ICU stay.

In [6]:
```
#Perform inner join
result_df = pd.merge(icu_admission_table, patients_with_vent_session, on='subject_id', how='inner')
```

In [7]:
```
# filter
ventilation_and_icu_data = result_df[
    (result_df['vent_starttime'] >= result_df['icu_intime'] - pd.Timedelta(minutes=15)) & (result_df['vent_endtime'] <
```

In [8]:
```
# result data
ventilation_and_icu_data = ventilation_and_icu_data[['subject_id', 'icu_seq', 'total_icu_stays',
                                                     'icu_intime','icu_outtime', 'los',
                                                     'vent_seq_no', 'total_vent_stays',
                                                     'vent_starttime','vent_endtime', 'vent_duration', 'duration_unit'
```

End of Step 1

In this step, we have identified a total of 8034 patients who met the following criteria:

- They had ICU stays lasting more than 48 hours.
- During these extended ICU stays, these patients also had ventilation sessions that lasted more than 48 hours.

Step 2: Identifying Patients with Clinical Indicators

In this step, we aim to identify patients with clinical indicators, Leukocytosis and Fever. The criteria for this identification are as follows:

- We will consider the test results for the observations **that were charted after patients have undergone ventilation sessions lasting 48 hours or more and before the respective ICU end time associated with that ventilation session.**

Step 2a: Identifying Patients Diagnosed with Fever

In this step, we focus on the patients identified in the previous step and determine if they were diagnosed with fever, defined as a body temperature exceeding 38 degrees Celsius.

Step Description:

1. We utilize a query to retrieve all body temperature tests where the recorded temperature exceeds 38 degrees Celsius for the patients identified in Step 1.
2. Subsequently, we filter the tests to include only **those performed after 48 hours of the ventilation session and before the respective ICU end time associated with that ventilation session.**

In [10]:
```
query = f"""

WITH TempInfo AS (
  SELECT itemid, label
    FROM `physionet-data.mimiciv_icu.d_items`
    WHERE LOWER(label) LIKE '%temperature celsius%'
    OR LOWER(label) LIKE '%temperature fahrenheit%'
    OR LOWER(label) LIKE '%temperatureф_apacheiv%'
)

SELECT
    lb.subject_id,                          -- Patient identifier
    lb.charttime,                           -- Vital sign chart time
    lb.valuenum ,                          -- Vital sign value number
    lb.valueuom                             -- UoM

FROM
    `physionet-data.mimiciv_icu.chartevents` AS lb,
    TempInfo as ti

where lb.itemid = ti.itemid
AND lb.subject_id IN ({', '.join(map(str, ventilation_and_icu_data_distinct_subject_ids))})  -- Filter for specified s
AND ((lb.valuenum > 38 AND lb.valueuom LIKE '%C') OR (lb.valuenum > 100.4 AND lb.valueuom LIKE '%F'))


"""
```

```
# Execute the query and retrieve patient demographic information.
patients_with_fever = run_query(query)

# Display the first few rows of the result.
patients_with_fever.head()
```

Out[10]:

	subject_id	charttime	valuenum	valueuom
0	16780877	2124-12-22 20:39:00	104.3	°F
1	16207680	2154-08-16 09:00:00	103.9	°F
2	16215286	2181-02-04 09:00:00	103.5	°F
3	15795343	2112-03-11 23:00:00	39.9	°C
4	17399377	2172-07-02 02:00:00	39.4	°C

Filter the tests to include only those performed after 48 hours of the ventilation session and before the respective ICU end time associated with that ventilation session.

In [11]:
```
result_df = pd.merge(ventilation_and_icu_data, patients_with_fever, on='subject_id', how='inner')
```

In [12]:
```
patients_with_fever_after_48hr_ventilation = result_df[
    (result_df['charttime'] >= result_df['vent_starttime'] + timedelta(hours=48)) &
    (result_df['charttime'] <= result_df['icu_outtime'])
]
```

In [13]:
```
#renaming the column names to make it more readable
patients_with_fever_after_48hr_ventilation.rename(columns={'valuenum': 'body_temp', 'valueuom': 'temp_scale', 'chartti
                                                           'los': 'icu_los', 'duration_unit': 'vent_duration_unit'}, i
```

```
<ipython-input-13-a4ee555a8c4e>:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning
-a-view-versus-a-copy
  patients_with_fever_after_48hr_ventilation.rename(columns={'valuenum': 'body_temp', 'valueuom': 'temp_scale', 'char
ttime': 'temp_charttime',
```

In [14]:
```
patients_with_fever_after_48hr_ventilation.head()
```

Out[14]:

	subject_id	icu_seq	total_icu_stays	icu_intime	icu_outtime	icu_los	vent_seq_no	total_vent_stays	vent_starttime	vent_endtime	ven
4	10002428	2	4	2156-04-19 18:11:19	2156-04-26 18:58:41	7.032894	1	3	2156-04-19 20:10:00	2156-04-22 17:05:00	
19	10002428	4	4	2156-05-11 14:49:34	2156-05-22 14:16:46	10.977222	3	3	2156-05-11 16:05:00	2156-05-20 10:45:00	
20	10004235	1	1	2196-02-24 17:07:00	2196-02-29 15:58:02	4.952106	1	1	2196-02-24 16:52:00	2196-02-27 16:28:00	
64	10004401	5	7	2144-04-21 20:49:00	2144-05-01 13:53:03	9.711146	6	8	2144-04-21 21:23:00	2144-05-01 13:53:00	
68	10004401	5	7	2144-04-21 20:49:00	2144-05-01 13:53:03	9.711146	6	8	2144-04-21 21:23:00	2144-05-01 13:53:00	

In [16]:
```
# Distinct subject ids
patients_with_fever_after_48hr_ventilation_distinct_subject_ids = patients_with_fever_after_48hr_ventilation['subject_
patients_with_fever_after_48hr_ventilation_distinct_subject_ids = patients_with_fever_after_48hr_ventilation_distinct_
len(patients_with_fever_after_48hr_ventilation_distinct_subject_ids)
```

Out[16]: 4354

Step 2b: Identifying Patients Diagnosed with Leukocytosis

In this step, we focus on the patients identified in the previous step and determine if they were diagnosed with Leukocytosis or Leukopenia.

Leukocytosis (white blood cell count ≥ 12 K/uL) or leukopenia (white blood cell count < 4 K/uL)

Step Description:

1. We utilize a query to retrieve all White blood cell lab tests done for the patients diagnosed with fever and meets the defined Leukocytosis OR leukopenia definition.
2. Subsequently, we filter the tests to include only **those performed after 48 hours of the ventilation session and before the respective ICU end time associated with that ventilation session.**

In [17]:
```
query = f"""

WITH WBCInfo AS (
  SELECT itemid
    FROM `physionet-data.mimiciv_hosp.d_labitems`
    WHERE LOWER(label) LIKE '%white blood cells%'
```

```
)

SELECT
    lb.subject_id,                          -- Patient identifier
    lb.charttime AS WBC_charttime,                          -- Vital sign chart time
    lb.valuenum AS WBC_count ,                        -- Vital sign value number
    lb.valueuom AS WBC_scale                         -- UoM

FROM
    `physionet-data.mimiciv_hosp.labevents` AS lb,
    WBCInfo as wi

where lb.itemid = wi.itemid
AND (lb.valuenum >= 12 OR lb.valuenum < 4)
AND valueuom like 'K/uL'
AND lb.subject_id IN ({', '.join(map(str, patients_with_fever_after_48hr_ventilation_distinct_subject_ids))})  -- Filt
"""

# Execute the query and retrieve patient demographic information.
patients_with_leukocytosis = run_query(query)

# Display the first few rows of the result.
patients_with_leukocytosis.head()
```

Out[17]:

	subject_id	WBC_charttime	WBC_count	WBC_scale
0	10007818	2146-06-26 03:43:00	23.7	K/uL
1	10021927	2180-09-21 09:46:00	48.0	K/uL
2	10035168	2144-07-31 12:55:00	67.5	K/uL
3	10035168	2144-09-24 10:45:00	37.7	K/uL
4	10035168	2145-01-11 00:30:00	2.0	K/uL

Filter the tests to include only those performed after 48 hours of the ventilation session and before the respective ICU end time associated with that ventilation session.

In [18]:
```
result_df = pd.merge(ventilation_and_icu_data, patients_with_leukocytosis, on='subject_id', how='inner')
```

In [19]:
```
patients_with_leukocytosis_and_fever_after_48hr_ventilation = result_df[
    (result_df['WBC_charttime'] >= result_df['vent_starttime'] + timedelta(hours=48)) &
    (result_df['WBC_charttime'] <= result_df['icu_outtime'])
]
```

In [20]:
```
# Distinct subject ids
patients_with_leukocytosis_and_fever_after_48hr_ventilation_distinct_subject_ids = patients_with_leukocytosis_and_feve
patients_with_leukocytosis_and_fever_after_48hr_ventilation_distinct_subject_ids = patients_with_leukocytosis_and_feve
len(patients_with_leukocytosis_and_fever_after_48hr_ventilation_distinct_subject_ids)
```

Out[20]:
```
3474
```

End of Step 2

In this step, we have identified a total of 3474 patients who meet a combination of two key criteria:

1. **ICU Stays and Ventilation Sessions**: These patients had ICU stays lasting more than 48 hours, and during these extended ICU stays, they also had ventilation sessions that persisted for more than 48 hours.

AND

1. **Clinical Indicator Criteria**: We further refined our identification to focus on patients who were diagnosed with specific clinical indicators, including fever, leukocytosis, and leukopenia. These diagnostic criteria were assessed after the patients' 48-hour ventilation sessions and before the respective ICU end times for those ventilation sessions.

Step 3: Identification of patients with Microbiology Events (bacterial growth).

•Endotracheal aspirates: ≥1,000,000 colony forming units (cfu)/mL

•Bronchoscopic- or mini-BAL: 10,000 cfu/mL

•PSB: 1000 cfu/mL

In this step, we aim to count the pulmonary pathogens using the readings mentioned above. The criteria for this identification are as follows:

- We will consider the test results for the observations **that were charted after patients have undergone ventilation sessions lasting 48 hours or more and before the respective ICU end time associated with that ventilation session.**

In [21]:
```
query = f"""

SELECT
    subject_id,
```

```
    spec_type_desc as path_spec_type_desc,
    charttime AS path_charttime,
    CASE
        WHEN LOWER(spec_type_desc) LIKE '%tracheal aspirate%' THEN 'Endotracheal aspirates — ≥1,000,000 colony forming
        WHEN LOWER(spec_type_desc) LIKE '%bronchial brush%' THEN 'Bronchoscopic- or mini-BAL — 10,000 cfu/mL'
        WHEN LOWER(spec_type_desc) LIKE '%mini-bal%' THEN 'Bronchoscopic- or mini-BAL — 10,000 cfu/mL'
        ELSE NULL
    END AS path_comment_data
FROM `physionet-data.mimiciv_hosp.microbiologyevents`
WHERE (
    -- Filter spec_type_desc
    (
        LOWER(spec_type_desc) LIKE '%tracheal aspirate%'
        OR LOWER(spec_type_desc) LIKE '%bronchial brush%'
        OR LOWER(spec_type_desc) LIKE '%mini-bal%'
    )
    -- Filter negative comments
    AND comments IS NOT NULL
    AND LOWER(comments) NOT LIKE '%absent%'
    AND comments NOT LIKE '___'
    AND comments NOT LIKE '%NO POLYMORPHONUCLEAR LEUKOCYTES SEEN%'
    AND comments NOT LIKE '%NO MYCOBACTERIA%'
    AND comments NOT LIKE '%NO MICROORGANISMS SEEN.%'
    AND (
        comments NOT LIKE '%NO FUNGUS ISOLATED%'
        AND comments NOT LIKE '%NO ACID FAST BACILLI SEEN ON CONCENTRATED SMEAR%'
        AND comments NOT LIKE '%NO GROWTH%'
        AND comments NOT LIKE '%NO VIRUS ISOLATED%'
        AND comments NOT LIKE '%NO LEGIONELLA ISOLATED%'
        AND comments NOT LIKE '%NO ANAEROBES ISOLATED%'
        AND comments NOT LIKE '%NO NOCARDIA ISOLATED%'
        AND comments NOT LIKE '%No Cytomegalovirus (CMV) isolated%'
        AND comments NOT LIKE '%No Herpes simplex (HSV) virus isolated%'
        AND comments NOT LIKE '%NO FUNGAL ELEMENTS SEEN%'
    )
    -- Additional filters for invalid events
    AND LOWER(comments) NOT LIKE '%test cancelled%'
    AND LOWER(comments) NOT LIKE '%invalid%'
    AND LOWER(comments) NOT LIKE '%not detected%'
    AND LOWER(comments) NOT LIKE '%unknown amount%'
    AND LOWER(comments) NOT LIKE '%no respiratory viruses isolated%'
    AND LOWER(comments) NOT LIKE '%rare growth%'
    AND LOWER(comments) NOT LIKE '%sparse growth%'
    AND LOWER(comments) NOT LIKE '%negative%'
)
and charttime is not null;

"""

# Execute the query and retrieve patient demographic information.
pulmonary_pathogens_info = run_query(query)

# Display the first few rows of the result.
pulmonary_pathogens_info.head()
```

Out[21]:

	subject_id	path_spec_type_desc	path_charttime	path_comment_data
0	14071688	Mini-BAL	2160-03-31 16:14:00	Bronchoscopic- or mini-BAL – 10,000 cfu/mL
1	12624383	Mini-BAL	2173-12-19 17:07:00	Bronchoscopic- or mini-BAL – 10,000 cfu/mL
2	17615845	Mini-BAL	2151-02-06 08:31:00	Bronchoscopic- or mini-BAL – 10,000 cfu/mL
3	19459496	Mini-BAL	2148-07-19 16:10:00	Bronchoscopic- or mini-BAL – 10,000 cfu/mL
4	19609358	Mini-BAL	2160-11-09 16:31:00	Bronchoscopic- or mini-BAL – 10,000 cfu/mL

Filter the tests to include only those performed after 48 hours of the ventilation session and before the respective ICU end time associated with that ventilation session.

In [22]:
```
result_df = pd.merge(ventilation_and_icu_data, pulmonary_pathogens_info, on='subject_id', how='inner')
```

In [23]:
```
patients_with_abnormal_bacterial_growth_after_48hr_ventilation = result_df[
    (result_df['path_charttime'] >= result_df['vent_starttime'] + timedelta(hours=48)) &
    (result_df['path_charttime'] <= result_df['icu_outtime'])
]
```

In [24]:
```
# Distinct subject ids
patients_with_abnormal_bacterial_growth_after_48hr_ventilation_distinct_subject_ids = patients_with_abnormal_bacterial
patients_with_abnormal_bacterial_growth_after_48hr_ventilation_distinct_subject_ids = patients_with_abnormal_bacterial
len(patients_with_abnormal_bacterial_growth_after_48hr_ventilation_distinct_subject_ids)
```

Out[24]:
```
299
```

End of Step 3

In this step, we have identified a total of 299 patients who meet a combination of two key criteria:

1. **ICU Stays and Ventilation Sessions**: These patients had ICU stays lasting more than 48 hours, and during these extended ICU stays, they also had ventilation sessions that persisted for more than 48 hours.

AND

1. **Microbiology Events (bacterial growth)**: We further refined our identification to focus on patients who surpassed the threshold value of the pulmonary pathogens. These diagnostic criteria were assessed after the patients' 48-hour ventilation sessions and before the respective ICU end times for those ventilation sessions.

Final Phenotype Identification

In the final phenotype identification, we are looking for patients who meet one of the following sets of criteria:

1. **Mechanical Ventilation Duration** and **Clinical Indicators**: Patients who have a prolonged duration of mechanical ventilation, and they exhibit specific clinical indicators such as Leukocytosis and Fever.

For this, we have identified 3474 patients.

- *OR*

1. **Mechanical Ventilation Duration** and **Microbiology Events**: Patients who have a prolonged duration of mechanical ventilation and experience significant microbiology events, specifically high values in tests like tracheal aspirates, mini-BAL, or PSB (Protected Specimen Brush).

For this, we have identified 299 patients.

Next, we will combine the patients obtained from both the condition and that will make our **Final phenotype.**

In [25]:
```
final_phenotype_patients = list(set(patients_with_abnormal_bacterial_growth_after_48hr_ventilation_distinct_subject_ic
        patients_with_leukocytosis_and_fever_after_48hr_ventilation_distinct_subject_ids))
```

In [26]:
```
len(final_phenotype_patients)
```

Out[26]:
```
3538
```

OUR FINAL PHENOTYPE HAS 3538 patients

Important Note: We Are Not Utilizing the ICD Table

It's crucial to emphasize that in our current analysis, we are not making use of the ICD (International Classification of Diseases) table. Our primary objective at this stage is to verify the accuracy of our phenotyping conditions, ensuring they correctly identify the desired patient criteria.

WE ARE NOT USING ICD TABLE.

Checking how many are there in the ICD code diagnosis table (NOT USING IT FOR ANY FILTARATION)

In [27]:
```
# Selecting from ICD code diagnosis table
query = f"""
       With VAP_ICD AS (
  SELECT icd_code, long_title
  FROM `physionet-data.mimiciv_hosp.d_icd_diagnoses`
  WHERE
  --ventilator associated pneumonia
  lower(icd_code) LIKE '99731%'
  OR lower(icd_code) LIKE 'j95851%'
)
SELECT di.subject_id
FROM `physionet-data.mimiciv_hosp.diagnoses_icd` as di
JOIN VAP_ICD as ICDEvents ON ICDEvents.icd_code = di.icd_code
AND di.hadm_id IS NOT NULL
AND di.subject_id IN ({', '.join(map(str, final_phenotype_patients))})  -- Filter for specified subject IDs
"""
df_t = run_query(query)
df_t.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1091 entries, 0 to 1090
Data columns (total 1 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   subject_id  1091 non-null   Int64
dtypes: Int64(1)
memory usage: 9.7 KB
```

GETTING INFORMATION FOR THE IDENTIFIED PATIENTS

Total phenotype length: 3538

Extracting important information we got from Step 1

In [28]:
```
# Create a copy of the original DataFrame to work with
icu_admission_table_sum = icu_admission_table.copy()

# Calculate the total ICU duration for each patient (subject_id) by summing the 'los' column within each group
icu_admission_table_sum['total_icu_duration'] = icu_admission_table.groupby('subject_id')['los'].transform('sum')

# Select and retain only the columns 'subject_id', 'total_icu_stays', and 'total_icu_duration'
icu_admission_table_sum = icu_admission_table_sum[['subject_id', 'total_icu_stays', 'total_icu_duration']]

# Remove duplicate rows based on all columns, keeping one row per unique combination
icu_admission_table_sum = icu_admission_table_sum.drop_duplicates()
```

In [29]:
```
icu_admission_table_sum.head()
```

Out[29]:

	subject_id	total_icu_stays	total_icu_duration
0	10001884	1	9.171817
1	10002155	3	10.070359
3	10002348	1	9.792512
4	10002428	4	25.015093
8	10002430	1	2.922593

In [30]:
```
# Can have mutiple stays
icu_admission_table[icu_admission_table['subject_id'] == 10002155]
```

Out[30]:

	subject_id	icu_intime	icu_outtime	los	icu_seq	total_icu_stays
1	10002155	2129-08-04 12:45:00	2129-08-10 17:02:38	6.178912	1	3
2	10002155	2130-09-24 00:50:00	2130-09-27 22:13:41	3.891447	2	3

Calculating Total ICU Stays and Total ICU Duration

In our analysis, we need to consider both the total number of ICU stays and the total ICU duration.

1. Total ICU Stays (total_icu_stays):

- This metric encompasses the complete count of ICU stays for each patient.
- **It includes all ICU stays, regardless of their duration, including those that are less than 48 hours.**

2. Total ICU Duration (total_icu_duration):

- This metric focuses specifically on the cumulative duration of ICU stays that **exceeded 48 hours.
- **It excludes the durations of ICU stays that were shorter than 48 hours.**

In [31]:
```
# Create a copy of the original DataFrame to work with
patients_with_vent_session_sum = patients_with_vent_session.copy()

# Calculate the total ICU duration for each patient (subject_id) by summing the 'los' column within each group
patients_with_vent_session_sum['total_vent_duration'] = patients_with_vent_session.groupby('subject_id')['vent_duratic

# Select and retain only the columns 'subject_id', 'total_icu_stays', and 'total_icu_duration'
patients_with_vent_session_sum = patients_with_vent_session_sum[['subject_id', 'total_vent_stays', 'total_vent_duratic

# Remove duplicate rows based on all columns, keeping one row per unique combination
patients_with_vent_session_sum = patients_with_vent_session_sum.drop_duplicates()
```

In [32]:
```
patients_with_vent_session_sum.head()
```

Out[32]:

	subject_id	total_vent_stays	total_vent_duration	duration_unit
0	10001884	4	16041.0	min
2	10002428	3	16775.0	min
4	10003400	4	14479.0	min
7	10004235	1	4296.0	min
8	10004401	8	62724.0	min

In [33]:
```
# Can have multiple ventilation stays
patients_with_vent_session[patients_with_vent_session['subject_id'] == 10001884]
```

Out[33]:

	subject_id	vent_starttime	vent_endtime	ventilation_type	vent_duration	duration_unit	patientweight	vent_seq_no	total_vent_stays
0	10001884	2131-01-13 04:00:00	2131-01-19 17:45:00	Invasive Ventilation	9465.0	min	65.0	3	4
1	10001884	2131-01-15 04:07:00	2131-01-19 17:43:00	Invasive Ventilation	6576.0	min	65.0	4	4

Calculating Total Ventilation Sessions and Total ventilation Duration

In our analysis, we need to consider both the Total Ventilation Sessions and Total ventilation Duration.

1. Total Ventilation Sessions (total_vent_stays):

- This metric encompasses the complete count of Ventilation sessions for each patient.
- **It includes all ventilation sessions, regardless of their duration, including those that are less than 48 hours.**

2. Total Ventilation Duration (total_vent_duration):

- This metric focuses specifically on the cumulative duration of Ventilation sessios that **exceeded 48 hours.
- **It excludes the durations of ventilation sessions that were shorter than 48 hours.**

In [34]:
```
import warnings

# Suppress all warnings
warnings.filterwarnings("ignore")
```

In [35]:
```
result_df = pd.merge(icu_admission_table_sum, patients_with_vent_session_sum, on='subject_id', how='inner')
```

In [36]:
```
# Filter the DataFrame to select rows with subject IDs in 'final_phenotype_patients'
ventilation_and_icu_data_sum = result_df[result_df['subject_id'].isin(final_phenotype_patients)]

# Convert 'total_vent_duration' from minutes to days
ventilation_and_icu_data_sum['total_vent_duration'] = ventilation_and_icu_data_sum['total_vent_duration'] / 1440

# Round 'total_vent_duration' to three decimal places
ventilation_and_icu_data_sum['total_vent_duration'] = ventilation_and_icu_data_sum['total_vent_duration'].round(3)

# Round 'total_icu_duration' to three decimal places
ventilation_and_icu_data_sum['total_icu_duration'] = ventilation_and_icu_data_sum['total_icu_duration'].round(3)

# Drop the 'duration_unit' column
ventilation_and_icu_data_sum = ventilation_and_icu_data_sum.drop('duration_unit', axis=1)
```

In [37]:
```
ventilation_and_icu_data_sum.head()
```

Out[37]:

	subject_id	total_icu_stays	total_icu_duration	total_vent_stays	total_vent_duration
3	10004235	1	4.952	1	2.983
4	10004401	7	53.950	8	43.558
7	10005606	1	6.595	1	3.778
8	10005817	2	18.332	2	13.226
9	10007818	1	20.529	1	20.174

In [38]:
```
ventilation_and_icu_data_sum.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 3538 entries, 3 to 8359
Data columns (total 5 columns):
 #   Column               Non-Null Count  Dtype  
---  ------               --------------  -----  
 0   subject_id           3538 non-null   Int64  
 1   total_icu_stays      3538 non-null   Int64  
 2   total_icu_duration   3538 non-null   float64
 3   total_vent_stays     3538 non-null   Int64  
 4   total_vent_duration  3538 non-null   float64
dtypes: Int64(3), float64(2)
memory usage: 176.2 KB
```

Extracting important information we got from Step 2

In [39]:
```
patients_with_fever_sum = patients_with_fever_after_48hr_ventilation.copy()
```

In [40]:
```
# Define a function to convert Fahrenheit to Celsius
def fahrenheit_to_celsius(fahrenheit):
    return (fahrenheit - 32) * 5/9

# Convert 'temp_scale' to lowercase for consistency
patients_with_fever_sum['temp_scale'] = patients_with_fever_sum['temp_scale'].str.lower()

# Convert temperatures to Celsius based on 'temp_scale'
patients_with_fever_sum['body_temp'] = patients_with_fever_sum.apply(
```

```
    lambda row: fahrenheit_to_celsius(row['body_temp']) if 'f' in row['temp_scale'] else row['body_temp'], axis=1)

# Drop the 'temp_scale' and 'temp_charttime' columns
patients_with_fever_sum = patients_with_fever_sum.drop(['temp_scale', 'temp_charttime'], axis=1)

# Select only 'subject_id' and 'body_temp' columns
patients_with_fever_sum = patients_with_fever_sum[['subject_id', 'body_temp']]

# Calculate the average body temperature for each patient
patients_with_fever_sum['avg_body_temp'] = patients_with_fever_sum.groupby('subject_id')['body_temp'].transform('mean'

# Round the 'avg_body_temp' to three decimal places
patients_with_fever_sum['avg_body_temp'] = patients_with_fever_sum['avg_body_temp'].round(3)

patients_with_fever_sum = patients_with_fever_sum.drop('body_temp', axis=1)

# Drop duplicate rows
patients_with_fever_sum = patients_with_fever_sum.drop_duplicates()

# Reset the index and drop the old index column
patients_with_fever_sum.reset_index(inplace=True, drop=True)
```

In [41]:
```
patients_with_fever_sum.head()
```

Out[41]:

	subject_id	avg_body_temp
0	10002428	38.500
1	10004235	38.500
2	10004401	38.222
3	10005606	38.250
4	10005817	38.685

The avg body temp indicates the average temperature of the patient after 48 hours of the ventilation session and before the respective ICU end time associated with that ventilation session.

In [42]:
```
patients_with_leukocytosis_sum = patients_with_leukocytosis_and_fever_after_48hr_ventilation.copy()
```

In [43]:
```
# Drop the 'WBC_count' and 'temp_charttime' columns
patients_with_leukocytosis_sum = patients_with_leukocytosis_sum.drop(['WBC_scale', 'WBC_charttime'], axis=1)

# Select only 'subject_id' and 'WBC_count' columns
patients_with_leukocytosis_sum = patients_with_leukocytosis_sum[['subject_id', 'WBC_count']]

# Calculate the average WBC for each patient
patients_with_leukocytosis_sum['avg_WBC_count'] = patients_with_leukocytosis_sum.groupby('subject_id')['WBC_count'].tr

# Round the 'avg_body_temp' to three decimal places
patients_with_leukocytosis_sum['avg_WBC_count'] = patients_with_leukocytosis_sum['avg_WBC_count'].round(3)

patients_with_leukocytosis_sum = patients_with_leukocytosis_sum.drop('WBC_count', axis=1)

# Drop duplicate rows
patients_with_leukocytosis_sum = patients_with_leukocytosis_sum.drop_duplicates()

# Reset the index and drop the old index column
patients_with_leukocytosis_sum.reset_index(inplace=True, drop=True)
```

In [44]:
```
patients_with_leukocytosis_sum.head()
```

Out[44]:

	subject_id	avg_WBC_count
0	10004235	15.100
1	10004401	9.288
2	10005606	16.186
3	10005817	13.257
4	10007818	18.686

The avg white blood cell (WBC) indicates the average WBC of the patient after 48 hours of the ventilation session and before the respective ICU end time associated with that ventilation session.

In [45]:
```
result_df = pd.merge(patients_with_leukocytosis_sum, patients_with_fever_sum, on='subject_id', how='inner')
```

In [46]:
```
step1_2_patient_info = pd.merge(ventilation_and_icu_data_sum, result_df, on='subject_id', how='left')
```

In [47]:
```
step1_2_patient_info.head()
```

Out[47]:

	subject_id	total_icu_stays	total_icu_duration	total_vent_stays	total_vent_duration	avg_WBC_count	avg_body_temp
0	10004235	1	4.952	1	2.983	15.100	38.500
1	10004401	7	53.950	8	43.558	9.288	38.222
2	10005606	1	6.595	1	3.778	16.186	38.250
3	10005817	2	18.332	2	13.226	13.257	38.685
4	10007818	1	20.529	1	20.174	18.686	38.056

Extracting important information we got from Step 3

In [48]:
```
patients_with_abnormal_bacterial_growth = patients_with_abnormal_bacterial_growth_after_48hr_ventilation[['subject_id'
```

In [49]:
```
patients_with_abnormal_bacterial_growth = patients_with_abnormal_bacterial_growth.drop_duplicates()
```

In [50]:
```
step1_2_3_patient_info = pd.merge(step1_2_patient_info, patients_with_abnormal_bacterial_growth, on='subject_id', how=
```

In [51]:
```
step1_2_3_patient_info.head()
```

Out[51]:

	subject_id	total_icu_stays	total_icu_duration	total_vent_stays	total_vent_duration	avg_WBC_count	avg_body_temp	path_spec_type_desc
0	10004235	1	4.952	1	2.983	15.100	38.500	NaN
1	10004401	7	53.950	8	43.558	9.288	38.222	NaN
2	10005606	1	6.595	1	3.778	16.186	38.250	NaN
3	10005817	2	18.332	2	13.226	13.257	38.685	NaN
4	10007818	1	20.529	1	20.174	18.686	38.056	NaN

COMBINED DIGITAL PHENOTYPE INFORMATION

NOTE: THE DATA ONLY HAS 300 records for the pathogen test information.

In [52]:
```
final_digital_phenotype = step1_2_3_patient_info.copy()
```

In [53]:
```
final_digital_phenotype.head()
```

Out[53]:

	subject_id	total_icu_stays	total_icu_duration	total_vent_stays	total_vent_duration	avg_WBC_count	avg_body_temp	path_spec_type_desc
0	10004235	1	4.952	1	2.983	15.100	38.500	NaN
1	10004401	7	53.950	8	43.558	9.288	38.222	NaN
2	10005606	1	6.595	1	3.778	16.186	38.250	NaN
3	10005817	2	18.332	2	13.226	13.257	38.685	NaN
4	10007818	1	20.529	1	20.174	18.686	38.056	NaN

In [54]:
```
final_patient_cohort_subject_ids = final_digital_phenotype['subject_id'].drop_duplicates()
final_patient_cohort_subject_ids = final_patient_cohort_subject_ids.tolist()
len(final_patient_cohort_subject_ids)
```

Out[54]:
```
3538
```

In [55]:
```
set(final_patient_cohort_subject_ids ) == set(final_phenotype_patients)
```

Out[55]:
```
True
```

Feature Selection

Risk Assessment Factors

A. Patient Characteristics

1. **Average BMI (Body Mass Index):** Average BMI before the ventilation of 48 hrs started.

   1a. First we find all the height and weight measurement of the patient before the start of its 48 hrs ventilation session.

   NOTE: We are considering the average of weight that was taken before each ventilation stay

   1b. Then we find the average BMI

2. **Gender**

3. **Age**

B. Outcome Variable

1. **Date of Death:** This is the variable we aim to predict or assess, representing whether the patient passed away or not.

C. Medical Data

1. **Lab Test Results** - Average of the lab results taken between the ICU intime and before the first ventillation session for that ICD sequence.
2. **Vital Test Results** - Average of the vital results taken between the ICU intime and before the first ventillation session for that ICD sequence.

D. Patient History

1. **Co-morbidity Index:** This factor represents the patient's pre-existing medical conditions and co-morbidities when they were initially admitted to the first ICU stay. It helps us understand the patient's overall health condition.

E. Bronschopy

1. **Bronchoscopy Procedure**: Check if the patient has received Bronchoscopy Procedure during its ventilation session (that lasted for more than 48 hours).

A. Patient Characteristics

BMI Value

1. First, we will find the height of the patients
2. Next, we will use the weight we got in our `ventilation_and_icu_data` table

In [56]:
```
query = f"""

WITH ht_in AS (
    SELECT
        c.subject_id, c.stay_id, c.charttime
        -- Ensure that all heights are in centimeters
        , ROUND(CAST(c.valuenum * 2.54 AS NUMERIC), 2) AS height
        , c.valuenum AS height_orig
    FROM `physionet-data.mimiciv_icu.chartevents` c
    WHERE c.valuenum IS NOT NULL
        -- Height (measured in inches)
        AND c.itemid = 226707
)

, ht_cm AS (
    SELECT
        c.subject_id, c.stay_id, c.charttime
        -- Ensure that all heights are in centimeters
        , ROUND(CAST(c.valuenum AS NUMERIC), 2) AS height
    FROM `physionet-data.mimiciv_icu.chartevents` c
    WHERE c.valuenum IS NOT NULL
        -- Height cm
        AND c.itemid = 226730
)

-- merge cm/height, only take 1 value per charted row
, ht_stg0 AS (
    SELECT
        COALESCE(h1.subject_id, h1.subject_id) AS subject_id
        , COALESCE(h1.stay_id, h1.stay_id) AS stay_id
        , COALESCE(h1.charttime, h1.charttime) AS charttime
        , COALESCE(h1.height, h2.height) AS height
    FROM ht_cm h1
    FULL OUTER JOIN ht_in h2
        ON h1.subject_id = h2.subject_id
            AND h1.charttime = h2.charttime
)

SELECT subject_id, height As height_cm, charttime As height_charttime
FROM ht_stg0
WHERE height IS NOT NULL
AND subject_id IN ({', '.join(map(str, final_phenotype_patients))})  -- Filter for specified subject IDs
ORDER BY subject_id
"""

# Execute the query and retrieve patient demographic information.
patient_height_info = run_query(query)
```

```
# Display the first few rows of the result.
patient_height_info.head()
```

Out[56]:

	subject_id	height_cm	height_charttime
0	10004235	183.000000000	2196-02-24 14:39:00
1	10004401	170.000000000	2144-06-05 19:46:00
2	10004401	168.000000000	2144-01-26 22:28:00
3	10004401	170.000000000	2144-04-05 09:32:00
4	10004401	170.000000000	2144-05-15 23:06:00

In [57]:
```
#Perform inner join
result_df = pd.merge(ventilation_and_icu_data, patient_height_info, on='subject_id', how='inner')
```

In [58]:
```
patient_avg_bmi_before_vent = result_df[
    result_df['height_charttime'] <= result_df['vent_starttime'] ]
```

In [59]:
```
patient_avg_bmi_before_vent = patient_avg_bmi_before_vent[['subject_id', 'height_cm', 'patientweight'  ]]
```

In [60]:
```
patient_avg_bmi_before_vent_sum = patient_avg_bmi_before_vent.copy()
```

In [61]:
```
# Calculate the average height and weight for each patient
patient_avg_bmi_before_vent_sum['avg_ht_cm'] = patient_avg_bmi_before_vent.groupby('subject_id')['height_cm'].transfor
patient_avg_bmi_before_vent_sum['avg_wt_kg'] = patient_avg_bmi_before_vent.groupby('subject_id')['patientweight'].tran

patient_avg_bmi_before_vent_sum = patient_avg_bmi_before_vent_sum.drop(['height_cm', 'patientweight'], axis=1)

patient_avg_bmi_before_vent_sum['avg_ht_cm'] = patient_avg_bmi_before_vent_sum['avg_ht_cm'].round(3)

patient_avg_bmi_before_vent_sum['avg_bmi'] = patient_avg_bmi_before_vent_sum['avg_wt_kg'] / ((patient_avg_bmi_before_v

patient_avg_bmi_before_vent_sum = patient_avg_bmi_before_vent_sum.drop(['avg_wt_kg', 'avg_ht_cm'], axis=1)

patient_avg_bmi_before_vent_sum = patient_avg_bmi_before_vent_sum.drop_duplicates()
patient_avg_bmi_before_vent_sum.reset_index(inplace = True, drop = True)
```

In [62]:
```
patient_avg_bmi_before_vent_sum.head()
```

Out[62]:

	subject_id	avg_bmi
0	10004235	29.860551
1	10004401	31.149509
2	10005606	26.543366
3	10005817	29.330163
4	10007818	25.186267

END OF CALCULATION FOR AVERAGE BMI

Age, Gender and Death Information

In [63]:
```
query = f"""

SELECT
  subject_id,
  gender,
  anchor_age AS age,
  CASE
    WHEN dod IS NOT NULL THEN 1
    ELSE 0
  END AS death_flag
FROM `physionet-data.mimic_core.patients`
WHERE anchor_age <> 0
AND subject_id IN ({', '.join(map(str, final_phenotype_patients))})  -- Filter for specified subject IDs

"""

# Execute the query and retrieve patient demographic information.
patient_general_info = run_query(query)

# Display the first few rows of the result.
patient_general_info.head()
```

Out[63]:

	subject_id	gender	age	death_flag
0	15069820	F	18	0
1	13207437	M	18	0
2	17716301	M	19	0
3	11842879	F	19	1
4	17400046	F	19	0

END OF CALCULATION FOR GENERAL PATIENT INFO

Medical Data

Lab Test Results

Average of the lab results taken between the ICU intime and before the first ventillation session for that ICD sequence.

In [64]:
```
###DO NOT RUN. I  have already taken the CSV for this.
query = f"""

WITH lab_events_Hematology AS (
    SELECT itemid, label, fluid, category
    FROM `physionet-data.mimiciv_hosp.d_labitems`
    WHERE
    LOWER(label) LIKE '%white blood cell%'
    OR LOWER(label) LIKE 'neutrophils'
    OR (LOWER(label) LIKE 'lymphocytes' AND fluid LIKE 'Blood' AND category LIKE 'Hematology')
    OR (LOWER(label) LIKE 'basophils' AND fluid LIKE 'Blood' AND category LIKE 'Hematology')
    OR (LOWER(label) LIKE 'eosinophils' AND fluid LIKE 'Blood' AND category LIKE 'Hematology')
    OR (LOWER(label) LIKE 'monocytes' AND fluid LIKE 'Blood' AND category LIKE 'Hematology')
    OR LOWER(label) LIKE '%red blood cells%'
    OR (LOWER(label) LIKE 'hematocrit' AND fluid LIKE 'Blood' AND category LIKE 'Hematology')
    OR (LOWER(label) LIKE 'hemoglobin' AND fluid LIKE 'Blood' AND category LIKE 'Hematology')
    OR (LOWER(label) LIKE 'mcv' AND fluid LIKE 'Blood' AND category LIKE 'Hematology')
    OR (LOWER(label) LIKE 'mch' AND fluid LIKE 'Blood' AND category LIKE 'Hematology')
    OR (LOWER(label) LIKE 'mchc' AND fluid LIKE 'Blood' AND category LIKE 'Hematology')
    OR (LOWER(label) LIKE 'rdw' AND fluid LIKE 'Blood' AND category LIKE 'Hematology')
    OR (LOWER(label) LIKE '%platelets%' AND fluid LIKE 'Blood' AND category LIKE 'Hematology')
    -- Electrolytes and Blood Gases
    OR LOWER(label) LIKE 'anion gap'
    OR LOWER(label) LIKE 'bicarbonate'
    OR LOWER(label) LIKE 'total calcium'
    OR (LOWER(label) LIKE 'free calcium' AND fluid LIKE 'Blood' AND category LIKE 'Blood Gas')
    OR LOWER(label) LIKE 'chloride'
    OR LOWER(label) LIKE 'sodium'
    OR LOWER(label) LIKE 'potassium'
    OR (LOWER(label) LIKE 'base excess' AND fluid LIKE 'Blood' AND category LIKE 'Blood Gas')
    OR (LOWER(label) LIKE 'ph' AND fluid LIKE 'Blood' AND category LIKE 'Blood Gas')
    OR (LOWER(label) LIKE 'pco2' AND fluid LIKE 'Blood' AND category LIKE 'Blood Gas')
    OR (LOWER(label) LIKE 'po2' AND fluid LIKE 'Blood' AND category LIKE 'Blood Gas')
-- Metabolic Markers
    OR (LOWER(label) LIKE 'lactate' AND fluid LIKE 'Blood' AND category LIKE 'Blood Gas')
    OR LOWER(label) LIKE 'creatinine'
    OR LOWER(label) LIKE 'urea nitrogen'
    OR (LOWER(label) LIKE 'glucose' AND fluid LIKE 'Urine' AND category LIKE 'Hematology')
    OR LOWER(label) LIKE 'sct - normalized ratio'
    -- Coagulation
    OR LOWER(label) LIKE 'pt'
    OR LOWER(label) LIKE 'ptt'
 -- Liver Function
    OR LOWER(label) LIKE '%alanine aminotransferase (alt)'
    OR LOWER(label) LIKE 'alkaline phosphatase'
    OR LOWER(label) LIKE '%lactate dehydrogenase (ld)'
    OR LOWER(label) LIKE 'bilirubin, total'
    OR LOWER(label) LIKE 'albumin'

    )

    SELECT subject_id,
            charttime,
            label,
            valuenum,
            valueuom,
            flag

    FROM
    `physionet-data.mimiciv_hosp.labevents` AS lb,
    lab_events_Hematology AS leh
    WHERE lb.itemid = leh.itemid
    AND lb.subject_id IN ({', '.join(map(str, final_phenotype_patients))})  -- Filter for specified subject IDs
    ORDER BY label, valueuom;

    """
lab_results_all = run_query(query)
```

```
lab_results_all.head()
```

Out[64]:

	subject_id	charttime	label	valuenum	valueuom	flag
0	10413295	2174-11-27 09:09:00	Alanine Aminotransferase (ALT)	24.0	IU/L	None
1	10952678	2141-09-06 08:50:00	Alanine Aminotransferase (ALT)	161.0	IU/L	abnormal
2	11088384	2177-08-19 05:25:00	Alanine Aminotransferase (ALT)	14.0	IU/L	None
3	11885477	2176-09-11 08:20:00	Alanine Aminotransferase (ALT)	20.0	IU/L	None
4	12453404	2126-11-26 16:10:00	Alanine Aminotransferase (ALT)	18.0	IU/L	None

In [ ]:
```
# csv_filename = 'lab_results_all.csv'
# lab_results_all.to_csv(csv_filename, index=True)
```

In [ ]:
```
lab_results_all = pd.read_csv('/content/lab_results_all.csv')
```

In [65]:
```
#Perform inner join
result_df = pd.merge(ventilation_and_icu_data, lab_results_all, on='subject_id', how='inner')
```

In [66]:
```
#Average of the lab results taken between the ICU intime and before the first ventillation session for that ICD sequen
patients_lab_results_before_ventilation_session = result_df[
    (result_df['charttime'] >= result_df['icu_intime'] )&
    (result_df['charttime'] <= result_df['vent_starttime'])
]
```

In [67]:
```
patients_lab_results_before_ventilation_session = patients_lab_results_before_ventilation_session[['subject_id',
                                                                                                    'icu_seq',
                                                                                                   'icu_intime',
                                                                                                    'vent_seq_no',
                                                                                                    'vent_starttime',
                                                                                                    'label',
                                                                                                    'charttime',
                                                                                                    'valuenum',
                                                                                                    'valueuom',
                                                                                                    'flag']]

column_mapping = {
    'label': 'lab_test',
    'charttime': 'lab_charttime',
    'valuenum': 'lab_test_value',
    'valueuom': 'lab_test_uom',
    'flag': 'lab_test_flag'
}

patients_lab_results_before_ventilation_session.rename(columns=column_mapping, inplace=True)
```

In [68]:
```
patients_lab_results_before_ventilation_session = patients_lab_results_before_ventilation_session[['subject_id', 'icu_

#Taking the first ventilation session information for the ICU sequence
patients_lab_results_before_ventilation_session.sort_values(by=['subject_id', 'icu_seq', 'vent_seq_no', 'lab_test' ],

patients_lab_results_before_ventilation_session = patients_lab_results_before_ventilation_session.groupby(['subject_id

patients_lab_results_before_ventilation_session = patients_lab_results_before_ventilation_session[['subject_id', 'lab_
```

In [69]:
```
patients_lab_results_before_ventilation_session.head()
```

Out[69]:

	subject_id	lab_test	lab_test_value
0	10004401	Anion Gap	17.571429
1	10004401	Base Excess	-4.833333
2	10004401	Basophils	0.000000
3	10004401	Bicarbonate	20.000000
4	10004401	Chloride	96.142857

Get the normal range for different lab tests

In [70]:
```
distinct_lab_tests = patients_lab_results_before_ventilation_session['lab_test'].drop_duplicates()
distinct_lab_tests = distinct_lab_tests.tolist()
len(distinct_lab_tests)
```

Out[70]:
```
36
```

In [71]:
```
# get the reference range for these lab tests

query = f"""
WITH lab_events_Hematology AS (
    SELECT itemid, label, fluid, category
    FROM `physionet-data.mimiciv_hosp.d_labitems`
    WHERE
    label IN ({', '.join([f"'{test}'" for test in distinct_lab_tests])})
    )
SELECT distinct label, ref_range_lower, ref_range_upper
```

```
FROM
`physionet-data.mimiciv_hosp.labevents` AS lb,
lab_events_Hematology AS leh
WHERE lb.itemid = leh.itemid
AND ref_range_upper <> 0
AND (ref_range_upper IS NOT NULL)
AND (ref_range_lower IS NOT NULL)
"""

reference_range_info = run_query(query)
reference_range_info.head()
```

Out[71]:

	label	ref_range_lower	ref_range_upper
0	Free Calcium	1.12	1.32
1	Lactate Dehydrogenase (LD)	94.00	250.00
2	White Blood Cells	4.00	10.00
3	Hematocrit	34.00	45.00
4	Hematocrit	36.00	48.00

We take the average of the lower and upper range, if we have multiple of these rows.

In [72]:
```
reference_range_info.sort_values(by=['label'], inplace=True)
reference_range_info = reference_range_info.groupby(['label']).mean().reset_index()
reference_range_info = reference_range_info.round(3)
```

In [73]:
```
reference_range_info.head()
```

Out[73]:

	label	ref_range_lower	ref_range_upper
0	Alanine Aminotransferase (ALT)	0.000	40.000
1	Albumin	3.450	5.000
2	Alkaline Phosphatase	38.000	117.333
3	Anion Gap	8.667	18.000
4	Basophils	0.000	1.500

Now that we have our range, we append `patients_lab_results_before_ventilation_session` with the **flag value**

In [74]:
```
# Merge the DataFrames based on the 'lab_test' column
patients_lab_results_before_ventilation_session = patients_lab_results_before_ventilation_session.merge(reference_ranc

# Function to determine 'flag' based on the lab_test_value and reference range
def determine_flag(row):
    if row['lab_test_value'] >= row['ref_range_lower'] and row['lab_test_value'] <= row['ref_range_upper']:
        return 'Normal'
    else:
        return 'Abnormal'

# Apply the function to create the 'flag' column
patients_lab_results_before_ventilation_session['flag'] = patients_lab_results_before_ventilation_session.apply(determ

patients_lab_results_before_ventilation_session = patients_lab_results_before_ventilation_session.drop(
    patients_lab_results_before_ventilation_session[patients_lab_results_before_ventilation_session['ref_range_lower']
# Drop the extra 'label' column
patients_lab_results_before_ventilation_session = patients_lab_results_before_ventilation_session.drop(['label','ref_r
```

In [75]:
```
patients_lab_results_before_ventilation_session.reset_index(inplace = True, drop = True)
```

In [77]:
```
patients_lab_results_before_ventilation_session = patients_lab_results_before_ventilation_session.drop(['index'], axis
```

In [78]:
```
distinct_lab_tests = patients_lab_results_before_ventilation_session['lab_test'].drop_duplicates()
distinct_lab_tests = distinct_lab_tests.tolist()
len(distinct_lab_tests)
```

Out[78]:
```
34
```

A total of 34 labtests are considered

In [79]:
```
patients_lab_results_before_ventilation_session.head()
```

Out[79]:

	subject_id	lab_test	lab_test_value	flag
0	10004401	Anion Gap	17.571429	Normal
1	10004401	Basophils	0.000000	Normal
2	10004401	Bicarbonate	20.000000	Abnormal
3	10004401	Chloride	96.142857	Normal
4	10004401	Creatinine	2.028571	Abnormal

END OF LAB TEST RESULTS

Vital Test Results

In [80]:
```
query = f"""

WITH vital_signs_selected AS (
    SELECT itemid, label, category, unitname
    FROM `physionet-data.mimiciv_icu.d_items`
    WHERE LOWER(label) LIKE '%heart rate%'
       OR LOWER(label) LIKE '%arterial pressure%'
       OR LOWER(label) LIKE '%respiratory rate (total)%'
       OR LOWER(label) LIKE '%temperature celsius%'
       OR LOWER(label) LIKE '%temperature f%'
       OR LOWER(label) LIKE '%spo2%'
       OR LOWER(label) LIKE '%urine output%'
)

SELECT lb.subject_id,                         -- Patient identifier
    vital_signs_selected.label as Vital_Test,             -- Vital sign label
    vital_signs_selected.category as Vital_Test_Category,          -- Vital sign category
    lb.charttime As Vital_Test_charttime,                          -- Vital sign chart time
    lb.valuenum As Vital_Test_Value,                        -- Vital sign value number
    vital_signs_selected.unitname AS Vital_Test_uom,         -- Unit of measurement

FROM `physionet-data.mimiciv_icu.chartevents` AS lb
JOIN vital_signs_selected ON vital_signs_selected.itemid = lb.itemid
AND lb.subject_id IN ({', '.join(map(str, final_phenotype_patients))})  -- Filter for specified subject IDs

    """
vitals_results_all = run_query(query)

vitals_results_all.head()
```

Out[80]:

	subject_id	Vital_Test	Vital_Test_Category	Vital_Test_charttime	Vital_Test_Value	Vital_Test_uom
0	17478232	Temperature Fahrenheit	Routine Vital Signs	2171-07-22 16:00:00	99.5	°F
1	17678798	Respiratory Rate (Total)	Respiratory	2176-03-30 03:00:00	10.0	insp/min
2	18078191	Temperature Fahrenheit	Routine Vital Signs	2151-01-22 16:00:00	99.1	°F
3	19599196	Temperature Celsius	Routine Vital Signs	2148-05-10 11:00:00	38.2	°C
4	10236326	Temperature Fahrenheit	Routine Vital Signs	2122-09-27 18:00:00	97.7	°F

In [ ]:
```
csv_filename = 'vitals_results_all.csv'
vitals_results_all.to_csv(csv_filename, index=True)
```

In [ ]:
```
vitals_results_all = pd.read_csv('/content/vitals_results_all.csv')
```

In [81]:
```
#Perform inner join
result_df = pd.merge(ventilation_and_icu_data, vitals_results_all, on='subject_id', how='inner')
```

In [82]:
```
#Average of the lab results taken between the ICU intime and before the first ventillation session for that ICD sequer
patients_vital_results_before_ventilation_session = result_df[
    (result_df['Vital_Test_charttime'] >= result_df['icu_intime'] )&
    (result_df['Vital_Test_charttime'] <= result_df['vent_starttime'])
]
```

In [83]:
```
patients_vital_results_before_ventilation_session = patients_vital_results_before_ventilation_session[['subject_id', '
                                                                                                        'Vital_Test_Categor
                                                                                                        'Vital_Test_Value',
                                                                                                        'Vital_Test_uom'   ]
```

In [84]:
```
patients_vital_results_before_ventilation_session[patients_vital_results_before_ventilation_session['subject_id'] == 1
```

Out[84]:

	subject_id	icu_seq	vent_seq_no	Vital_Test	Vital_Test_Category	Vital_Test_Value	Vital_Test_uom
187	10004401	1	1	Temperature Fahrenheit	Routine Vital Signs	99.4	°F
193	10004401	1	1	Temperature Fahrenheit	Routine Vital Signs	96.5	°F
202	10004401	1	1	Temperature Fahrenheit	Routine Vital Signs	96.9	°F
262	10004401	1	1	Temperature Fahrenheit	Routine Vital Signs	99.4	°F
272	10004401	1	1	Temperature Fahrenheit	Routine Vital Signs	98.1	°F
...	...	...	...	...	...	...	...
6758	10004401	3	4	Temperature Fahrenheit	Routine Vital Signs	98.8	°F
9444	10004401	5	6	Respiratory Rate (Total)	Respiratory	20.0	insp/min
10192	10004401	5	6	Heart Rate	Routine Vital Signs	70.0	bpm
11562	10004401	6	7	Temperature Fahrenheit	Routine Vital Signs	96.7	°F
12812	10004401	6	7	Heart Rate	Routine Vital Signs	70.0	bpm

171 rows × 7 columns

In [85]:
```
group_columns = ['subject_id', 'icu_seq', 'vent_seq_no', 'Vital_Test', 'Vital_Test_Category', 'Vital_Test_uom'  ]
value_column = 'Vital_Test_Value'
result_df_1 = patients_vital_results_before_ventilation_session.groupby(group_columns)[value_column].mean().reset_inde
```

In [86]:
```
group_columns = ['subject_id', 'Vital_Test', 'Vital_Test_Category', 'Vital_Test_uom']
value_column = 'Vital_Test_Value'
result_df_2 = patients_vital_results_before_ventilation_session.groupby(group_columns)[value_column].mean().reset_inde
```

In [87]:
```
patients_vital_results_before_ventilation_session = result_df_2.copy()
```

In [88]:
```
patients_vital_results_before_ventilation_session.head(5)
```

Out[88]:

	subject_id	Vital_Test	Vital_Test_Category	Vital_Test_uom	Vital_Test_Value
0	10004401	Heart Rate	Routine Vital Signs	bpm	78.031250
1	10004401	Heart Rate Alarm - Low	Alarms	bpm	54.583333
2	10004401	Heart rate Alarm - High	Alarms	bpm	120.000000
3	10004401	Respiratory Rate (Total)	Respiratory	insp/min	21.454545
4	10004401	SpO2 Desat Limit	Alarms	%	87.000000

END OF VITAL INFO RESULTS

Bronschopy Procedure

In [89]:
```
query = f"""
-- Define bronchoscopy items
WITH bronchoscopy_info AS (
    SELECT * FROM `physionet-data.mimiciv_icu.d_items`
    WHERE LOWER(label) LIKE 'bronchoscopy'
)

SELECT
    pe.subject_id,                -- Patient identifier
    pe.starttime,                 -- Bronchoscopy start time
    pe.endtime,                   -- Bronchoscopy end time
    bi.label,                     -- Bronchoscopy label

FROM
    `physionet-data.mimiciv_icu.procedureevents` AS pe
JOIN bronchoscopy_info AS bi ON bi.itemid = pe.itemid
AND pe.statusdescription LIKE 'FinishedRunning'
AND subject_id IN ({', '.join(map(str, final_patient_cohort_subject_ids))})  -- Filter for specified subject IDs

"""
bronchoscopy_patient_info = run_query(query)

bronchoscopy_patient_info.head()
```

Out[89]:

	subject_id	starttime	endtime	label
0	10004401	2144-01-29 14:52:00	2144-01-29 14:53:00	Bronchoscopy
1	10004401	2144-01-27 19:12:00	2144-01-27 19:13:00	Bronchoscopy
2	10004401	2144-01-27 20:25:00	2144-01-27 20:26:00	Bronchoscopy
3	10004401	2144-01-28 14:36:00	2144-01-28 14:37:00	Bronchoscopy
4	10005817	2135-01-09 17:11:00	2135-01-09 17:12:00	Bronchoscopy

In [90]:
```
#Perform inner join
result_df = pd.merge(ventilation_and_icu_data, bronchoscopy_patient_info, on='subject_id', how='inner')
```

In [91]:
```
result_df.head()
```

Out[91]:

	subject_id	icu_seq	total_icu_stays	icu_intime	icu_outtime	los	vent_seq_no	total_vent_stays	vent_starttime	vent_endtime	vent
0	10004401	1	7	2144-01-26 22:28:04	2144-02-06 13:44:15	10.636238	1	8	2144-01-30 13:00:00	2144-02-03 08:01:00	
1	10004401	1	7	2144-01-26 22:28:04	2144-02-06 13:44:15	10.636238	1	8	2144-01-30 13:00:00	2144-02-03 08:01:00	
2	10004401	1	7	2144-01-26 22:28:04	2144-02-06 13:44:15	10.636238	1	8	2144-01-30 13:00:00	2144-02-03 08:01:00	
3	10004401	1	7	2144-01-26 22:28:04	2144-02-06 13:44:15	10.636238	1	8	2144-01-30 13:00:00	2144-02-03 08:01:00	
4	10004401	2	7	2144-02-12 18:27:59	2144-02-19 14:42:24	6.843345	2	8	2144-02-12 19:00:00	2144-02-16 10:35:00	

In [92]:
```
patients_with_bronchoscopy_bw_ventilation = result_df[
    (result_df['starttime'] >= result_df['vent_starttime']) &
    (result_df['endtime'] <= result_df['vent_endtime'])
]
```

In [93]:
```
# Distinct subject ids
distinct_subject_ids_1 = patients_with_bronchoscopy_bw_ventilation['subject_id'].drop_duplicates()
distinct_subject_ids_1 = distinct_subject_ids_1.tolist()
len(distinct_subject_ids_1)
```

Out[93]:
```
1485
```

In [94]:
```
data = {
    'subject_id': distinct_subject_ids_1,
    'bronchoscopy': 1
}

patients_with_bronchoscopy = pd.DataFrame(data)
```

In [95]:
```
patients_with_bronchoscopy.head()
```

Out[95]:

	subject_id	bronchoscopy
0	10005817	1
1	10007818	1
2	10011365	1
3	10021927	1
4	10032381	1

END OF Bronschopy Procedure

Charlson-Comorbidity Index

In [96]:
```
patient_comorbidity_index = """

WITH diag AS (
    SELECT
        hadm_id
        , CASE WHEN icd_version = 9 THEN icd_code ELSE NULL END AS icd9_code
        , CASE WHEN icd_version = 10 THEN icd_code ELSE NULL END AS icd10_code
    FROM `physionet-data.mimiciv_hosp.diagnoses_icd`
)
, com AS (
    SELECT
        ad.hadm_id,
        -- Malignant Neoplasm
        MAX(CASE
              WHEN (icd10_code LIKE 'V109%') THEN 2
              ELSE 0 END) AS malignancy,
        -- Acute Myocardial Infarction
        MAX(CASE
              WHEN (icd9_code LIKE '410%' OR icd10_code LIKE 'I21%') THEN 1
              ELSE 0 END) AS mi,
        -- Congestive Heart Failure
        MAX(CASE
              WHEN (icd9_code LIKE '39891%' OR icd10_code LIKE 'I50%') THEN 1
              ELSE 0 END) AS chf,
        -- Cerebrovascular Disease
```

```
        MAX(CASE
              WHEN (icd9_code LIKE '43401%' OR icd9_code LIKE '43411%'
                  OR icd9_code LIKE '43491%' OR icd10_code LIKE 'I63%') THEN 1
              ELSE 0 END) AS cerebrovascular_disease,
        -- Chronic Pulmonary Disease
        MAX(CASE
              WHEN (icd9_code LIKE '496%' OR icd10_code LIKE 'I279%'
                  OR icd10_code LIKE 'J44%') THEN 1
              ELSE 0 END) AS chronic_pulmonary_disease,
        -- Liver Disease
        MAX(CASE
              WHEN (icd9_code LIKE '571%'
                  OR (icd10_code LIKE 'K7%' AND icd10_code >= 'K70' AND icd10_code <= 'K77')) THEN 1
              ELSE 0 END) AS liver_disease,
        -- Kidney Disease
        MAX(CASE
              WHEN (icd10_code LIKE 'N18%' OR icd9_code LIKE '585%') THEN 2
              ELSE 0 END) AS kidney_disease,
        -- Diabetes
        MAX(CASE
              WHEN (icd9_code LIKE '250%'
                  OR (icd10_code LIKE 'E10%' AND icd10_code >= 'E10' AND icd10_code <= 'E11')) THEN 1
              ELSE 0 END) AS diabetes
    FROM `physionet-data.mimiciv_hosp.admissions` ad
    LEFT JOIN diag
        ON ad.hadm_id = diag.hadm_id
    GROUP BY ad.hadm_id
)
, ag AS (
    SELECT
        ad.hadm_id,
        ad.subject_id,
        anchor_age + (EXTRACT(YEAR FROM ad.admittime) - anchor_year) AS age,
        CASE
            WHEN (anchor_age + (EXTRACT(YEAR FROM ad.admittime) - anchor_year)) <= 50 THEN 0
            WHEN (anchor_age + (EXTRACT(YEAR FROM ad.admittime) - anchor_year)) <= 60 THEN 1
            WHEN (anchor_age + (EXTRACT(YEAR FROM ad.admittime) - anchor_year)) <= 70 THEN 2
            WHEN (anchor_age + (EXTRACT(YEAR FROM ad.admittime) - anchor_year)) <= 80 THEN 3
            ELSE 4
        END AS age_score
    FROM
        `physionet-data.mimiciv_hosp.admissions` ad
    JOIN
        `physionet-data.mimiciv_hosp.patients` p
    ON
        ad.subject_id = p.subject_id
)
SELECT
    ad.subject_id,
    malignancy + mi + chf + cerebrovascular_disease + chronic_pulmonary_disease
      + liver_disease + kidney_disease + diabetes
    AS charlson_comorbidity_index
FROM `physionet-data.mimiciv_hosp.admissions` ad
LEFT JOIN com
    ON ad.hadm_id = com.hadm_id
LEFT JOIN ag
    ON com.hadm_id = ag.hadm_id
;
"""
patient_comorbidity_index_info = run_query(patient_comorbidity_index)
patient_comorbidity_index_info.head()
```

Out[96]:

	subject_id	charlson_comorbidity_index
0	16648184	5
1	13375185	5
2	14271881	5
3	17373919	5
4	19265652	5

In [97]:
```
patient_comorbidity_index_info = patient_comorbidity_index_info[patient_comorbidity_index_info['charlson_comorbidity_i
```

In [98]:
```
# Distinct subject ids
distinct_subject_ids_1 = patient_comorbidity_index_info['subject_id'].drop_duplicates()
distinct_subject_ids_1 = distinct_subject_ids_1.tolist()
len(distinct_subject_ids_1)
```

Out[98]:
```
62989
```

In [99]:
```
final_digital_phenotype.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 3540 entries, 0 to 3539
Data columns (total 9 columns):
 #   Column               Non-Null Count  Dtype  
---  ------               --------------  -----  
 0   subject_id           3540 non-null   Int64  
 1   total_icu_stays      3540 non-null   Int64  
 2   total_icu_duration   3540 non-null   float64
 3   total_vent_stays     3540 non-null   Int64  
 4   total_vent_duration  3540 non-null   float64
 5   avg_WBC_count        3476 non-null   float64
 6   avg_body_temp        3476 non-null   float64
 7   path_spec_type_desc  301 non-null    object 
 8   path_comment_data    301 non-null    object 
dtypes: Int64(3), float64(4), object(2)
memory usage: 286.9+ KB
```

In [100…
```
#Perform inner join
result_df = pd.merge(final_digital_phenotype, patient_comorbidity_index_info, on='subject_id', how='inner')
result_df = result_df[['subject_id', 'charlson_comorbidity_index']]
result_df = result_df.drop_duplicates()
```

In [101…
```
result_df.head()
```

Out[101]:

	subject_id	charlson_comorbidity_index
0	10004235	2
1	10004401	2
5	10005606	1
6	10005817	3
7	10005817	4

In [102…
```
patient_comorbidity_index_info = result_df.copy()
```

In [103…
```
# Final Comorbidity Index table
patient_comorbidity_index_info.head()
```

Out[103]:

	subject_id	charlson_comorbidity_index
0	10004235	2
1	10004401	2
5	10005606	1
6	10005817	3
7	10005817	4

END OF Charlson-Comorbidity Index

FINAL PHENOTYPE

In [104…
```
final_digital_phenotype.head()
```

Out[104]:

	subject_id	total_icu_stays	total_icu_duration	total_vent_stays	total_vent_duration	avg_WBC_count	avg_body_temp	path_spec_type_des
0	10004235	1	4.952	1	2.983	15.100	38.500	Na
1	10004401	7	53.950	8	43.558	9.288	38.222	Na
2	10005606	1	6.595	1	3.778	16.186	38.250	Na
3	10005817	2	18.332	2	13.226	13.257	38.685	Na
4	10007818	1	20.529	1	20.174	18.686	38.056	Na

In [105…
```
patient_avg_bmi_before_vent_sum.head()
```

Out[105]:

	subject_id	avg_bmi
0	10004235	29.860551
1	10004401	31.149509
2	10005606	26.543366
3	10005817	29.330163
4	10007818	25.186267

In [106…
```
patient_general_info.head()
```

Out[106]:

	subject_id	gender	age	death_flag
0	15069820	F	18	0
1	13207437	M	18	0
2	17716301	M	19	0
3	11842879	F	19	1
4	17400046	F	19	0

In [107…
```
patients_lab_results_before_ventilation_session.head()
```

Out[107]:

	subject_id	lab_test	lab_test_value	flag
0	10004401	Anion Gap	17.571429	Normal
1	10004401	Basophils	0.000000	Normal
2	10004401	Bicarbonate	20.000000	Abnormal
3	10004401	Chloride	96.142857	Normal
4	10004401	Creatinine	2.028571	Abnormal

In [108…
```
patients_vital_results_before_ventilation_session.head()
```

Out[108]:

	subject_id	Vital_Test	Vital_Test_Category	Vital_Test_uom	Vital_Test_Value
0	10004401	Heart Rate	Routine Vital Signs	bpm	78.031250
1	10004401	Heart Rate Alarm - Low	Alarms	bpm	54.583333
2	10004401	Heart rate Alarm - High	Alarms	bpm	120.000000
3	10004401	Respiratory Rate (Total)	Respiratory	insp/min	21.454545
4	10004401	SpO2 Desat Limit	Alarms	%	87.000000

In [109…
```
patients_with_bronchoscopy.head()
```

Out[109]:

	subject_id	bronchoscopy
0	10005817	1
1	10007818	1
2	10011365	1
3	10021927	1
4	10032381	1

In [110…
```
patient_comorbidity_index_info.head()
```

Out[110]:

	subject_id	charlson_comorbidity_index
0	10004235	2
1	10004401	2
5	10005606	1
6	10005817	3
7	10005817	4

In [111…
```
result_df_1 = pd.merge(final_digital_phenotype, patient_avg_bmi_before_vent_sum, on='subject_id', how='left')
result_df_2 = pd.merge(result_df_1, patient_general_info, on='subject_id', how='left')
result_df_3 = pd.merge(result_df_2, patients_lab_results_before_ventilation_session, on='subject_id', how='left')
result_df_4 = pd.merge(result_df_3, patients_vital_results_before_ventilation_session, on='subject_id', how='left')
result_df_5 = pd.merge(result_df_4, patients_with_bronchoscopy, on='subject_id', how='left')
result_df_6 = pd.merge(result_df_5, patient_comorbidity_index_info, on='subject_id', how='left')
```

In [112…
```
result_df_6.drop(['path_spec_type_desc', 'path_comment_data'], axis=1, inplace=True)
```

In [113…
```
column_mapping = {
    'flag': 'lab_test_flag'
}

result_df_6.rename(columns=column_mapping, inplace=True)
```

In [114…
```
result_df_6['bronchoscopy'] = result_df_6['bronchoscopy'].fillna(0).astype(int)
```

In [116…
```
identified_final_cohort = result_df_6.copy()
```

In [117…
```
csv_filename = 'identified_final_cohort.csv'
identified_final_cohort.to_csv(csv_filename, index=True)
```

MACHINE LEARNING SECTION

In [ ]:
```
identified_final_cohort = pd.read_csv('/content/identified_final_cohort.csv')
```

Data Pre-processing

Drop Null values

In [120…
```
#dropping columns that got created while dataframe was saved as a csv file
identified_final_cohort.drop(['Unnamed: 0'], axis = 1, inplace = True)
```

In [121…
```
result_df = identified_final_cohort.copy()
```

In [122…
```
result_df.head()
```

Out[122]:

	subject_id	total_icu_stays	total_icu_duration	total_vent_stays	total_vent_duration	avg_WBC_count	avg_body_temp	avg_bmi	gender
0	10004235	1	4.952	1	2.983	15.100	38.500	29.860551	M
1	10004401	7	53.950	8	43.558	9.288	38.222	31.149509	M
2	10004401	7	53.950	8	43.558	9.288	38.222	31.149509	M
3	10004401	7	53.950	8	43.558	9.288	38.222	31.149509	M
4	10004401	7	53.950	8	43.558	9.288	38.222	31.149509	M

In [123…
```
#drop columns that have NaNs
result_df = result_df[result_df['lab_test'].notnull()]
result_df = result_df[result_df['lab_test_value'].notnull()]
result_df = result_df[result_df['Vital_Test'].notnull()]
result_df = result_df[result_df['Vital_Test_Category'] != 'Alarms' ]
result_df = result_df[result_df['Vital_Test_Value'].notnull()]
```

In [124…
```
result_df = result_df.drop(['lab_test_flag', 'Vital_Test_Category', 'Vital_Test_uom'], axis=1)
result_df.reset_index(inplace = True, drop = True)
```

In [125…
```
result_df.head()
```

Out[125]:

	subject_id	total_icu_stays	total_icu_duration	total_vent_stays	total_vent_duration	avg_WBC_count	avg_body_temp	avg_bmi	gender
0	10004401	7	53.95	8	43.558	9.288	38.222	31.149509	M
1	10004401	7	53.95	8	43.558	9.288	38.222	31.149509	M
2	10004401	7	53.95	8	43.558	9.288	38.222	31.149509	M
3	10004401	7	53.95	8	43.558	9.288	38.222	31.149509	M
4	10004401	7	53.95	8	43.558	9.288	38.222	31.149509	M

LAB TEST

In [126…
```
lab_info_df = result_df.copy()
```

In [127…
```
lab_info_df = lab_info_df[['subject_id', 'lab_test', 'lab_test_value']]
```

In [128…
```
lab_info_df.drop_duplicates(inplace = True)
lab_info_df.reset_index(inplace = True, drop = True)
```

In [129…
```
# Pivot the dataframe to create the desired structure
lab_info_df = lab_info_df.pivot_table(index='subject_id', columns='lab_test', values='lab_test_value', aggfunc='first'

# If you want to replace NaN with None, you can use the following line:
lab_info_df = lab_info_df.where(lab_info_df.notna(), None)
```

In [131…
```
lab_info_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2290 entries, 0 to 2289
Data columns (total 35 columns):
 #   Column                          Non-Null Count  Dtype
---  ------                          --------------  -----
 0   subject_id                      2290 non-null   Int64
 1   Alanine Aminotransferase (ALT)  1393 non-null   float64
 2   Albumin                         1099 non-null   float64
 3   Alkaline Phosphatase            1386 non-null   float64
 4   Anion Gap                       1961 non-null   float64
 5   Basophils                       1211 non-null   float64
 6   Bicarbonate                     1963 non-null   float64
 7   Bilirubin, Total                1383 non-null   float64
 8   Chloride                        1966 non-null   float64
 9   Creatinine                      1965 non-null   float64
 10  Eosinophils                     1211 non-null   float64
 11  Free Calcium                    1635 non-null   float64
 12  Glucose                         164 non-null    float64
 13  Hematocrit                      2041 non-null   float64
 14  Hemoglobin                      2034 non-null   float64
 15  Lactate                         1953 non-null   float64
 16  Lactate Dehydrogenase (LD)      1077 non-null   float64
 17  Lymphocytes                     1211 non-null   float64
 18  MCH                             2033 non-null   float64
 19  MCHC                            2034 non-null   float64
 20  MCV                             2033 non-null   float64
 21  Monocytes                       1211 non-null   float64
 22  Neutrophils                     1211 non-null   float64
 23  PT                              1938 non-null   float64
 24  PTT                             1928 non-null   float64
 25  Potassium                       1963 non-null   float64
 26  RDW                             2032 non-null   float64
 27  Red Blood Cells                 2033 non-null   float64
 28  SCT - Normalized Ratio          1 non-null      float64
 29  Sodium                          1962 non-null   float64
 30  Urea Nitrogen                   1970 non-null   float64
 31  White Blood Cells               2033 non-null   float64
 32  pCO2                            2072 non-null   float64
 33  pH                              2080 non-null   float64
 34  pO2                             2073 non-null   float64
dtypes: Int64(1), float64(34)
memory usage: 628.5 KB
```

VITAL INFO

In [132…
```
vital_info_df = result_df.copy()
```

In [133…
```
vital_info_df = vital_info_df[['subject_id', 'Vital_Test', 'Vital_Test_Value']]
```

In [134…
```
vital_info_df.drop_duplicates(inplace = True)
vital_info_df.reset_index(inplace = True, drop = True)
```

In [135…
```
# Pivot the dataframe to create the desired structure
vital_info_df = vital_info_df.pivot_table(index='subject_id', columns='Vital_Test', values='Vital_Test_Value', aggfunc

# If you want to replace NaN with None, you can use the following line:
vital_info_df = vital_info_df.where(vital_info_df.notna(), None)
```

In [136…
```
vital_info_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2290 entries, 0 to 2289
Data columns (total 6 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   subject_id                2290 non-null   Int64
 1   Heart Rate                2224 non-null   float64
 2   Respiratory Rate (Total)  1947 non-null   float64
 3   Temperature Celsius       303 non-null    float64
 4   Temperature Fahrenheit    1981 non-null   float64
 5   Urine output_ApacheIV     1 non-null      float64
dtypes: Int64(1), float64(5)
memory usage: 109.7 KB
```

Comobidity Index

In [137…
```
result_df.drop(['lab_test', 'lab_test_value', 'Vital_Test', 'Vital_Test_Value'], axis=1, inplace = True)
```

In [138…
```
result_df.drop_duplicates(inplace = True)
```

In [139…
```
result_df.reset_index(inplace = True, drop = True)
```

In [140…
```
result_df.head()
```

Out[140]:

	subject_id	total_icu_stays	total_icu_duration	total_vent_stays	total_vent_duration	avg_WBC_count	avg_body_temp	avg_bmi	gender
0	10004401	7	53.950	8	43.558	9.288	38.222	31.149509	M
1	10005606	1	6.595	1	3.778	16.186	38.250	26.543366	M
2	10005817	2	18.332	2	13.226	13.257	38.685	29.330163	M
3	10005817	2	18.332	2	13.226	13.257	38.685	29.330163	M
4	10007818	1	20.529	1	20.174	18.686	38.056	25.186267	M

In [141…
```
cmob_result = result_df.groupby('subject_id')['charlson_comorbidity_index'].mean().reset_index()
```

In [142…
```
cmob_result.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2290 entries, 0 to 2289
Data columns (total 2 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   subject_id                2290 non-null   Int64
 1   charlson_comorbidity_index  1628 non-null   Float64
dtypes: Float64(1), Int64(1)
memory usage: 40.4 KB
```

In [143…
```
result_df.drop(['charlson_comorbidity_index'], axis=1, inplace = True)
```

In [144…
```
result_df.drop_duplicates(inplace = True)
```

Combining all the results

In [145…
```
#Perform inner join
preprocessed_cohort = pd.merge(result_df, cmob_result, on='subject_id', how='inner')
preprocessed_cohort = pd.merge(preprocessed_cohort, vital_info_df, on='subject_id', how='inner')
preprocessed_cohort = pd.merge(preprocessed_cohort, lab_info_df, on='subject_id', how='inner')
```

In [146…
```
preprocessed_cohort.head()
```

Out[146]:

	subject_id	total_icu_stays	total_icu_duration	total_vent_stays	total_vent_duration	avg_WBC_count	avg_body_temp	avg_bmi	gender
0	10004401	7	53.950	8	43.558	9.288	38.222	31.149509	M
1	10005606	1	6.595	1	3.778	16.186	38.250	26.543366	M
2	10005817	2	18.332	2	13.226	13.257	38.685	29.330163	M
3	10007818	1	20.529	1	20.174	18.686	38.056	25.186267	M
4	10011365	1	8.703	2	7.390	12.750	38.683	18.783723	F

5 rows × 52 columns

In [ ]:
```
csv_filename = 'preprocessed_cohort.csv'
preprocessed_cohort.to_csv(csv_filename, index=True)
```

In [ ]:
```
preprocessed_cohort = pd.read_csv('preprocessed_cohort.csv')
```

In [148…
```
preprocessed_cohort['Temperature Celsius'] = preprocessed_cohort['Temperature Celsius'].combine_first((preprocessed_cc
preprocessed_cohort = preprocessed_cohort.drop(columns=['Temperature Fahrenheit'])
```

In [150…
```
preprocessed_cohort['charlson_comorbidity_index'].fillna(0, inplace=True)
preprocessed_cohort.drop(['Unnamed: 0'], axis = 1, inplace = True)

preprocessed_cohort.drop(['subject_id'], axis = 1, inplace = True)
#checked hosp patient table
preprocessed_cohort['gender'] = preprocessed_cohort['gender'].fillna('M')
preprocessed_cohort['death_flag'] = preprocessed_cohort['death_flag'].fillna(0)
preprocessed_cohort['death_flag'] = preprocessed_cohort['death_flag'].astype("int")
```

Next, we remove rows with high null values

In [151…
```
# Check for null values in each column
null_counts = preprocessed_cohort.isnull().sum()

# Calculate the percentage of null values in each column
percentage_null = (null_counts / len(preprocessed_cohort)) * 100

# List columns with more than 80% null values
columns_with_high_null = percentage_null[percentage_null > 40]

# Print null counts and columns with high null values
print("Null Counts:")
print(null_counts)
```

```
print("\nColumns with More Than 40% Null Values:")
print(columns_with_high_null)
```

```
Null Counts:
total_icu_stays                      0
total_icu_duration                   0
total_vent_stays                     0
total_vent_duration                  0
avg_WBC_count                       37
avg_body_temp                       37
avg_bmi                            160
gender                               0
age                                  2
death_flag                           0
bronchoscopy                         0
charlson_comorbidity_index           0
Heart Rate                          66
Respiratory Rate (Total)           343
Temperature Celsius                283
Urine output_ApacheIV             2289
Alanine Aminotransferase (ALT)     897
Albumin                           1191
Alkaline Phosphatase               904
Anion Gap                          329
Basophils                         1079
Bicarbonate                        327
Bilirubin, Total                   907
Chloride                           324
Creatinine                         325
Eosinophils                       1079
Free Calcium                       655
Glucose                           2126
Hematocrit                         249
Hemoglobin                         256
Lactate                            337
Lactate Dehydrogenase (LD)        1213
Lymphocytes                       1079
MCH                                257
MCHC                               256
MCV                                257
Monocytes                         1079
Neutrophils                       1079
PT                                 352
PTT                                362
Potassium                          327
RDW                                258
Red Blood Cells                    257
SCT - Normalized Ratio            2289
Sodium                             328
Urea Nitrogen                      320
White Blood Cells                  257
pCO2                               218
pH                                 210
pO2                                217
dtype: int64

Columns with More Than 40% Null Values:
Urine output_ApacheIV         99.956332
Albumin                       52.008734
Basophils                     47.117904
Eosinophils                   47.117904
Glucose                       92.838428
Lactate Dehydrogenase (LD)    52.969432
Lymphocytes                   47.117904
Monocytes                     47.117904
Neutrophils                   47.117904
SCT - Normalized Ratio        99.956332
dtype: float64
```

In [152…

```
# Drop columns that are not important
columns_to_drop = ['Urine output_ApacheIV', 'Albumin', 'Basophils',
                   'Eosinophils', 'Glucose', 'Lactate Dehydrogenase (LD)', 'Lymphocytes',
                   'Monocytes', 'Neutrophils', 'SCT - Normalized Ratio']


preprocessed_cohort.drop(columns=columns_to_drop, inplace=True)
```

Check the total number of lab tests and vital information considered

In [ ]:

```
preprocessed_cohort = preprocessed_cohort.rename(columns=lambda x: x.lower().replace(' ', '_'))
```

In [153…

```
preprocessed_cohort.head()
```

Out[153]:

	total_icu_stays	total_icu_duration	total_vent_stays	total_vent_duration	avg_WBC_count	avg_body_temp	avg_bmi	gender	age	death
0	7	53.950	8	43.558	9.288	38.222	31.149509	M	82	
1	1	6.595	1	3.778	16.186	38.250	26.543366	M	38	
2	2	18.332	2	13.226	13.257	38.685	29.330163	M	66	
3	1	20.529	1	20.174	18.686	38.056	25.186267	M	69	
4	1	8.703	2	7.390	12.750	38.683	18.783723	F	73	

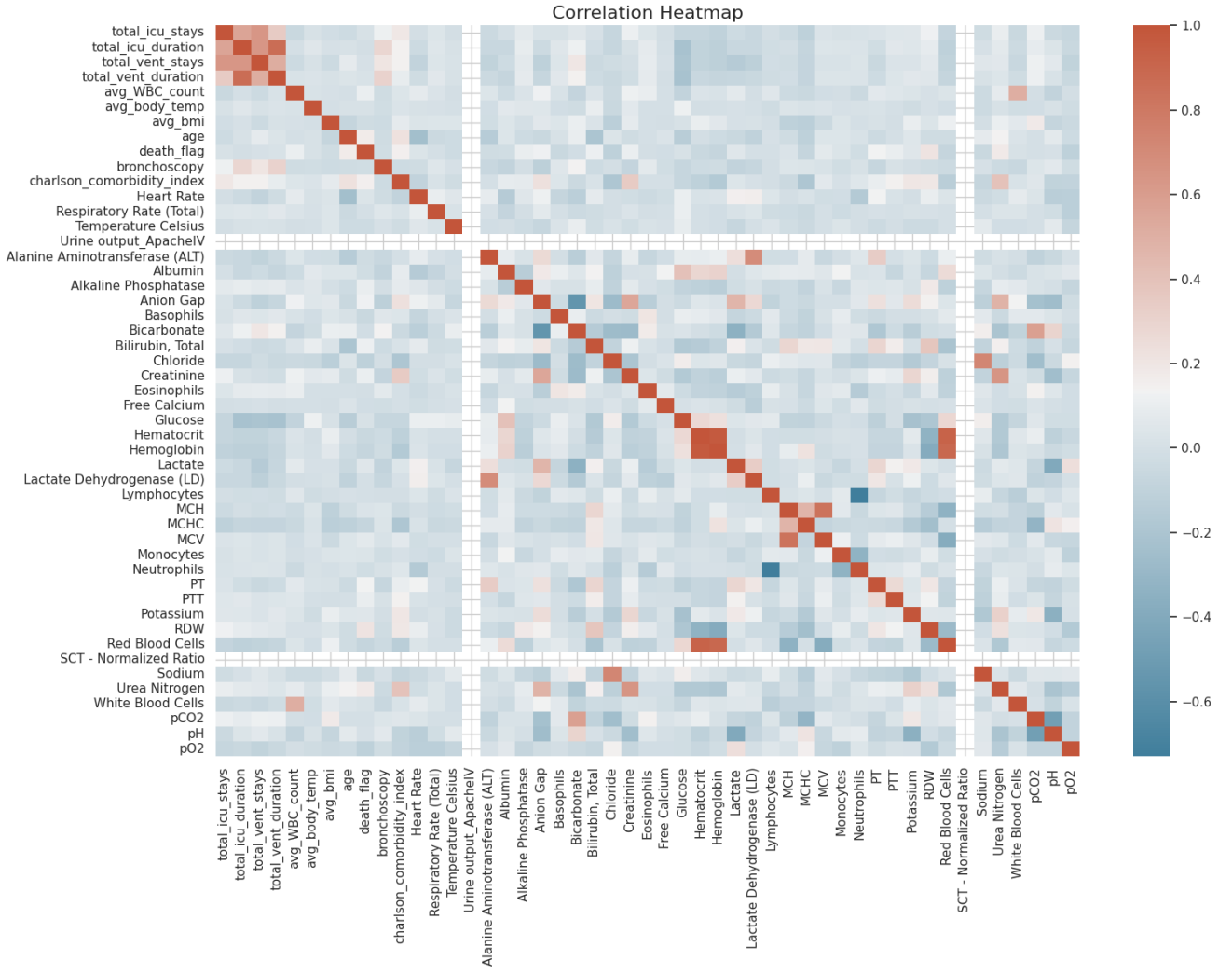
5 rows × 40 columns

In [ ]:
```
csv_filename = 'preprocessed_cohort_2290_mean_null.csv'
machine_learning_data.to_csv(csv_filename, index=True)
```

Remove the highly correlated columns

In [ ]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from pandas.plotting import table

# Sample data (replace with your actual data)
data = preprocessed_cohort.copy()

# Calculate the correlation matrix
corr_matrix = data.corr()

# Set up the matplotlib figure
plt.figure(figsize=(16, 12))

# Define a custom color palette
cmap = sns.diverging_palette(230, 20, as_cmap=True)

# Create the correlation heatmap with color intensity
sns.heatmap(corr_matrix, cmap=cmap, annot=False, fmt=".2f", cbar=True)

# Customize the plot
plt.title("Correlation Heatmap", fontsize=16)
plt.xticks(rotation=90)
plt.tight_layout()

# Calculate the top 10 correlated pairs (excluding pairs with corr = 1)
corr_pairs = (corr_matrix.unstack().sort_values(ascending=False)
              .drop_duplicates().head(11))  # Change '11' to '10' to exclude pairs with corr = 1

# Create a DataFrame for the top 10 correlated pairs with rounded correlations
corr_pairs_df = pd.DataFrame(corr_pairs, columns=["Correlation"])
corr_pairs_df.index.names = ["Feature 1", "Feature 2"]  # Adjust index names

# Exclude pairs with corr = 1 (keeping the top 10)
corr_pairs_df = corr_pairs_df[corr_pairs_df['Correlation'] < 1]

# Round the correlation values to 2 decimal points
corr_pairs_df['Correlation'] = corr_pairs_df['Correlation'].round(2)

# Create a table image with wider columns
fig, ax = plt.subplots(figsize=(10, 6))
ax.axis('tight')
ax.axis('off')
table_data = corr_pairs_df.reset_index()
tab = table(ax, table_data, loc='center', cellLoc='center', colWidths=[0.4, 0.4, 0.2])  # Adjust column widths

# Make index names bold
tab.scale(1, 1.5)
for (i, j), cell in tab._cells.items():
    if i == 0:
        cell.set_text_props(weight='bold')

tab.auto_set_font_size(False)
tab.set_fontsize(10)

# Save the table image as a PNG file
plt.title("Top 10 Correlated Pairs", fontsize=14)
plt.subplots_adjust(top=0.85)  # Reduce space between title and table
plt.savefig("correlation_pairs_table.png", bbox_inches='tight', pad_inches=0.2)

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

```
<ipython-input-9-bd8c42afd4de>:11: FutureWarning: The default value of numeric_only in DataFrame.corr is deprecated.
In a future version, it will default to False. Select only valid columns or specify the value of numeric_only to sile
nce this warning.
  corr_matrix = data.corr()
```



Top 10 Correlated Pairs

	Feature 1	Feature 2	Correlation
0	Hematocrit	Hemoglobin	0.97
1	Hematocrit	Red Blood Cells	0.92
2	Red Blood Cells	Hemoglobin	0.9
3	total_icu_duration	total_vent_duration	0.87
4	MCV	MCH	0.84
5	Sodium	Chloride	0.73
6	Alanine Aminotransferase (ALT)	Lactate Dehydrogenase (LD)	0.69
7	Urea Nitrogen	Creatinine	0.64
8	total_vent_stays	total_icu_stays	0.64
9	total_vent_stays	total_icu_duration	0.64

Removing Outliers

- **get_whisker_range:** The get_whisker_range function calculates and prints the lower and upper whisker values. It is used to eliminate the outliers.
- **box_plot:** This function creates a box plot for selected columns in a dataset. It also calculates the whisker range for the columns by calling the get_whisker_range function for the outlier removal process.
- **remove_outliers:** This function filters out outliers from the dataset based on the specified bounds
- **comparision_plot:** This function plots the comparison between different statistics for the original and outlier-removed dataset

- **feature_scaling:** This function standardizes the numeric columns of the dataset

In [ ]:
```
#Data Manipulation

import pandas as pd
import numpy as np

#Data Visualization
import matplotlib.pyplot as plt
from matplotlib.lines import Line2D
import seaborn as sns

#Clustering

from sklearn.preprocessing import StandardScaler  # For data scaling
from sklearn.cluster import KMeans  # For K-means clustering
from sklearn.decomposition import PCA  # For Principal Component Analysis
from sklearn.metrics import silhouette_score # For Silhouette score

# Function to calculate and print the lower and upper whisker values for the columns
def get_whisker_range(dataset, columns_to_exclude):
    # Select columns to analyze, excluding those in 'columns_to_exclude'
    selected_columns = [col for col in dataset.columns if col not in columns_to_exclude]

    # Initialize a dictionary to store whisker values for each column
    whisker_dict = {}

    # Calculate whisker values for each selected column
    for column in selected_columns:
        Q1 = dataset[column].quantile(0.25)
        Q3 = dataset[column].quantile(0.75)
        IQR = Q3 - Q1
        lower_bound = Q1 - 1.5 * IQR
        upper_bound = Q3 + 1.5 * IQR

        # Store the whisker values in the dictionary
        whisker_dict[column] = {'Lower Whisker': lower_bound, 'Upper Whisker': upper_bound}

        # Filter the dataset to include only data within the whisker range
        dataset = dataset[(dataset[column] >= lower_bound) & (dataset[column] <= upper_bound)]

    # Print the whisker values for each column
    for col, values in whisker_dict.items():
        print(f"Column: {col}")
        print(f"Lower Whisker: {values['Lower Whisker']}")
        print(f"Upper Whisker: {values['Upper Whisker']}\n")

    return whisker_dict


# function to visualize the box plots and return the values of the IQR range of the box plot
def box_plot(dataset, columns_to_exclude, disp):

    # Drop specified columns from the dataset
    dataset = dataset.drop(columns=columns_to_exclude)

    #get whisker range
    whisker_dict = get_whisker_range(dataset, columns_to_exclude )

    if (disp == True):
        # Create a box plot for the remaining columns
        boxplot = dataset.boxplot(figsize=(10, 7))

        # Add a title to the box plot
        plt.title('Box Plot for Selected Columns')
        # Show the plot
        plt.show()

    return whisker_dict

# Function to remove outliers from a dataset based on lower and upper bounds for each column
def remove_outliers(bounds, dataset, columns_to_exclude):

    # Get the list of columns after excluding non-numeric columns
    selected_columns = [col for col in dataset.columns if col not in columns_to_exclude]

    # Iterate through selected columns and filter out outliers
    for col in selected_columns:
        lower_bound = bounds[col]['Lower Whisker']
        upper_bound = bounds[col]['Upper Whisker']
        dataset = dataset[(dataset[col] >= lower_bound) & (dataset[col] <= upper_bound)]

    #return clean dataset with removed outliers
    return dataset

# plot the comparison graph between the summary statistics of the original and outlier-removed dataset
def comparision_plot(statistics_to_compare, dataset,summary_stats,cleaned_data_summary_stats, columns_to_exclude ):


    # Select columns to analyze, excluding those in 'columns_to_exclude'
    selected_columns = [col for col in dataset.columns if col not in columns_to_exclude]
```

```
    # Create an empty DataFrame to store the statistics for comparison
    comparison_df = pd.DataFrame({'Column': selected_columns})

    # Creating a dataframe that contains the statistics value for both the dataset (with and without outliers)
    for stat in statistics_to_compare:
        mean_original = summary_stats.loc[stat, selected_columns].values
        mean_no_outliers = cleaned_data_summary_stats.loc[stat, selected_columns].values
        comparison_df[f'{stat}_Original'] = mean_original
        comparison_df[f'{stat}_No_Outliers'] = mean_no_outliers
        print (f"Difference in {stat} statistic between original and cleaned dataset is\n")
        diff = abs(cleaned_data_summary_stats.loc[stat] - summary_stats.loc[stat])
        print(diff)
        print("\n")
        print (f"Percentage change in {stat} statistic between original and cleaned dataset is\n")
        print((diff / summary_stats.loc[stat]) * 100)
        print("\n")

    # To better visualization, we set the index
    comparison_df.set_index('Column', inplace=True)

    # Create a bar chart to compare the statistics
    ax = comparison_df.plot(kind='bar', figsize=(12, 6), colormap='viridis', width=0.7)
    ax.set_title('Comparison of Summary Statistics (Original vs. No Outliers)')
    ax.set_xlabel('Column')
    ax.set_ylabel('Value')
    ax.set_xticklabels(selected_columns, rotation=45, ha="right")
    ax.legend(title='Dataset', loc='upper right', bbox_to_anchor=(1.15, 1))

    # Annotate the bars with values
    for p in ax.patches:
        ax.annotate(f'{p.get_height():.2f}', (p.get_x() + p.get_width() / 2., p.get_height()),
                    ha='center', va='center', fontsize=8, color='black', xytext=(0, 5),
                    textcoords='offset points')

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

#Standardize selected numeric columns in the dataset to have mean=0 and standard deviation=1
def feature_scaling(dataset, columns_to_exclude):

    # initialize StandardScaler
    scaler = StandardScaler()

    selected_columns = [col for col in dataset.columns if col not in columns_to_exclude]

    # Apply the scaler to the selected columns
    dataset[selected_columns] = scaler.fit_transform(dataset[selected_columns])
    return dataset, scaler
```

In [ ]:
```
standardized_data_red_cleaned = remove_outliers(whisker_info, standardized_data_red, ['death_flag'])
```

In [ ]:
```
standardized_data_red_cleaned.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 629 entries, 1 to 2288
Data columns (total 24 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   total_vent_duration       629 non-null    float64
 1   avg_WBC_count             629 non-null    float64
 2   avg_body_temp             629 non-null    float64
 3   avg_bmi                   629 non-null    float64
 4   age                       629 non-null    float64
 5   Heart Rate                629 non-null    float64
 6   Respiratory Rate (Total)  629 non-null    float64
 7   Temperature Celsius       629 non-null    float64
 8   Chloride                  629 non-null    float64
 9   Creatinine                629 non-null    float64
 10  Lactate                   629 non-null    float64
 11  MCH                       629 non-null    float64
 12  MCHC                      629 non-null    float64
 13  PT                        629 non-null    float64
 14  PTT                       629 non-null    float64
 15  RDW                       629 non-null    float64
 16  Red Blood Cells           629 non-null    float64
 17  Urea Nitrogen             629 non-null    float64
 18  White Blood Cells         629 non-null    float64
 19  pCO2                      629 non-null    float64
 20  pO2                       629 non-null    float64
 21  bronchoscopy_0            629 non-null    uint8
 22  bronchoscopy_1            629 non-null    uint8
 23  death_flag                629 non-null    int64
dtypes: float64(21), int64(1), uint8(2)
memory usage: 114.3 KB
```

Feature Selection

Using RFE (Recursive Feature Elimination)

In [154…
```
import pandas as pd
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from xgboost import XGBClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.feature_selection import RFE

# Define the response variable and predictor variables
response_variable = 'death_flag'
predictor_variables = [col for col in standardized_data.columns if col != response_variable]

# Create the design matrix (X) and response variable (y)
X = standardized_data[predictor_variables]
y = standardized_data[response_variable]

# Split the data into a training set and a test set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Initialize estimators for RFE
estimators = [
    ('Logistic Regression', LogisticRegression()),
    ('SVM', SVC(kernel='linear'))
]

# Initialize a dictionary to store feature rankings
feature_rankings = {estimator_name: [] for estimator_name, _ in estimators}

# Perform RFE with each estimator and collect feature rankings
for estimator_name, estimator in estimators:
  rfe = RFE(estimator, n_features_to_select=20)
  rfe.fit(X_train, y_train)
  feature_rankings[estimator_name] = rfe.ranking_

# Combine the rankings based on the frequency of features being in the top 15
from collections import Counter
combined_rankings = Counter()
for estimator_name, ranking in feature_rankings.items():
    for feature_index, rank in enumerate(ranking):
        if rank <= 15:
            combined_rankings[feature_index] += 1

# Select the top 10 features based on the voting mechanism
selected_feature_indices = [feature_index for feature_index, count in combined_rankings.most_common(15)]

# Get the names of the selected features
selected_features = [predictor_variables[i] for i in selected_feature_indices]
```

In [ ]:
```
len(selected_features)
```

Out[ ]:
```
15
```

In [ ]:
```
standardized_data_reduced = standardized_data[selected_features + ['death_flag' , 'gender']]
```

Machine Learning Models

Feature Importance

In [ ]:
```
from sklearn.feature_selection import RFE
# Define the response variable and predictor variables
response_variable = 'death_flag'
predictor_variables = [col for col in standardized_data.columns if col != response_variable]

# Create the design matrix (X) and response variable (y)
X = standardized_data[predictor_variables]
y = standardized_data[response_variable].astype(int)

# Split the data into a training set and a test set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize estimators for RFE
estimators = [
    ('Logistic Regression', LogisticRegression()),
    ('SVM', SVC(kernel='linear'))
]

# Initialize a dictionary to store feature rankings
feature_rankings = {estimator_name: [] for estimator_name, _ in estimators}

# Perform RFE with each estimator and collect feature rankings
for estimator_name, estimator in estimators:
  rfe = RFE(estimator, n_features_to_select=20)
  rfe.fit(X_train, y_train)
  feature_rankings[estimator_name] = rfe.ranking_

# Combine the rankings based on the frequency of features being in the top 15
```

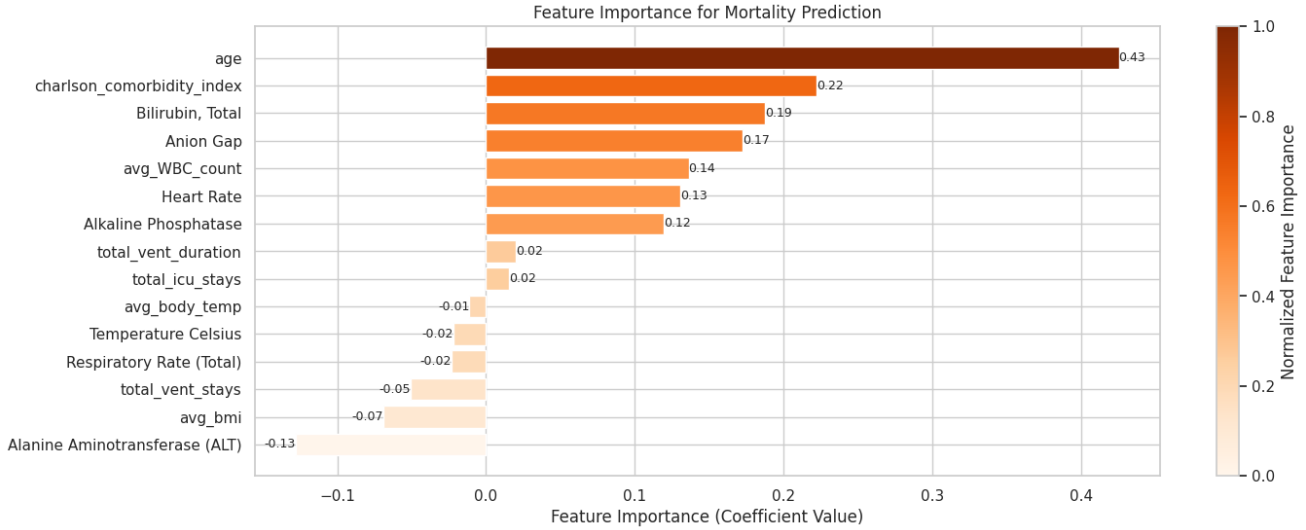
```
from collections import Counter
combined_rankings = Counter()
for estimator_name, ranking in feature_rankings.items():
    for feature_index, rank in enumerate(ranking):
        if rank <= 15:
            combined_rankings[feature_index] += 1

# Select the top 10 features based on the voting mechanism
selected_feature_indices = [feature_index for feature_index, count in combined_rankings.most_common(15)]

# Get the names of the selected features
selected_features = [predictor_variables[i] for i in selected_feature_indices]

# Now, 'selected_features' contains the names of the top 15 features selected by the majority of the estimators
print("Selected Features:", selected_features)
```

```
Selected Features: ['total_icu_stays', 'total_vent_stays', 'total_vent_duration', 'avg_WBC_count', 'avg_body_temp',
'avg_bmi', 'age', 'charlson_comorbidity_index', 'Heart Rate', 'Respiratory Rate (Total)', 'Temperature Celsius', 'Ala
nine Aminotransferase (ALT)', 'Alkaline Phosphatase', 'Anion Gap', 'Bilirubin, Total']
```

In [ ]:

```
import matplotlib.pyplot as plt

# Fit the selected estimator on the training data
selected_estimator = LogisticRegression()  # You can replace this with 'SVM' if needed
selected_estimator.fit(X_train[selected_features], y_train)

# Get the coefficients/importance of the features
feature_importance = selected_estimator.coef_[0]  # For Logistic Regression, change this for SVM if needed

# Sort the features by importance in descending order
sorted_indices = (-feature_importance).argsort()
sorted_features = [selected_features[i] for i in sorted_indices]
sorted_importance = feature_importance[sorted_indices]

# Choose a color map for the feature importance
color_map = plt.get_cmap('Oranges')  # Use 'Oranges' for positive values

# Normalize the feature importance values to the range [0, 1]
normalized_importance = (sorted_importance - sorted_importance.min()) / (sorted_importance.max() - sorted_importance.m

# Create a bar chart with varying colors based on feature importance
plt.figure(figsize=(15, 6))
bars = plt.barh(sorted_features, sorted_importance, color=color_map(normalized_importance))
plt.xlabel('Feature Importance (Coefficient Value)')
plt.title('Feature Importance for Mortality Prediction')

# Invert the y-axis to show the most important feature at the top
plt.gca().invert_yaxis()

# Add a color bar to indicate the significance of feature importance
sm = plt.cm.ScalarMappable(cmap=color_map, norm=plt.Normalize(vmin=normalized_importance.min(), vmax=normalized_import
sm.set_array([])
cbar = plt.colorbar(sm)
cbar.set_label('Normalized Feature Importance')

# Add feature importance values to the right of the bars
for feature, importance in zip(sorted_features, sorted_importance):
    if importance > 0:
        plt.text(importance, feature, f'{importance:.2f}', ha='left', va='center', fontsize=9)
    else:
        plt.text(importance, feature, f'{importance:.2f}', ha='right', va='center', fontsize=9)

plt.show()
```

```
<ipython-input-147-490735cf65f8>:33: MatplotlibDeprecationWarning:

Unable to determine Axes to steal space for Colorbar. Using gca(), but will raise in the future. Either provide the *
cax* argument to use as the Axes for the Colorbar, provide the *ax* argument to steal space from it, or add *mappable
* to an Axes.
```



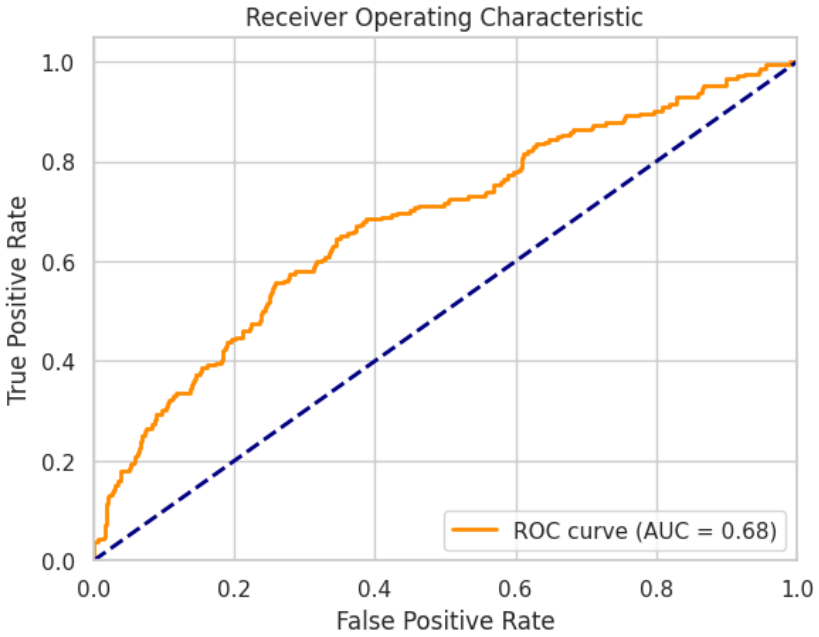
Logistic Regression

In [ ]:

```python
# Import necessary libraries
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import (
    precision_score, recall_score, f1_score, accuracy_score, confusion_matrix,
    classification_report, roc_auc_score, roc_curve
)
import matplotlib.pyplot as plt
from sklearn.metrics import matthews_corrcoef

# Define response variable and predictor variables
response_variable = 'death_flag'
predictor_variables = [col for col in standardized_data_reduced.columns if col != response_variable]

# Create the design matrix (X) and response variable (y)
X = standardized_data_reduced[predictor_variables]
y = standardized_data_reduced[response_variable]

# Split the data into a training set and a test set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Calculate class weights
class_counts = y_train.value_counts()
total_samples = len(y_train)
class_weights = {0: total_samples / (2 * class_counts[0]), 1: total_samples / (2 * class_counts[1])}

# Fit a logistic regression model
model = LogisticRegression(class_weight=class_weights, max_iter=1000)  # Increase max_iter if necessary
model.fit(X_train, y_train)

# Make predictions on the test set
y_pred = model.predict(X_test)

# Calculate evaluation metrics
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
tn, fp, fn, tp = conf_matrix.ravel()
sensitivity = tp / (tp + fn)
specificity = tn / (tn + fp)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1_score = 2 * (precision * recall) / (precision + recall)
y_prob = model.predict_proba(X_test)[:, 1]  # Probability of the positive class
roc_auc_log = roc_auc_score(y_test, y_prob)
fpr_log, tpr_log, _ = roc_curve(y_test, y_prob)
balanced_acc = (sensitivity + specificity) / 2
mcc = matthews_corrcoef(y_test, y_pred)

# Add MCC to the list of evaluation metrics
print(f'Matthews Correlation Coefficient (MCC): {mcc:.2f}')
# Plot the ROC curve
plt.figure()
plt.plot(fpr_log, tpr_log, color='darkorange', lw=2, label=f'ROC curve (AUC = {roc_auc_log:.2f})')
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic')
plt.legend(loc='lower right')
plt.show()
```

```
# Output the evaluation metrics
print("Evaluation Metrics:")
print(f'Accuracy: {accuracy:.2f}')
print(f'Sensitivity: {sensitivity:.2f}')
print(f'Specificity: {specificity:.2f}')
print(f'F1 Score: {f1_score:.2f}')
print(f'AUC-ROC: {roc_auc:.2f}')
print(f'Balanced Accuracy: {balanced_acc:.2f}')

# Output the confusion matrix
print("\nConfusion Matrix:")
labels = ["True Negative", "False Positive", "False Negative", "True Positive"]
for i in range(2):
    for j in range(2):
        print(f"{labels[i * 2 + j]:<15}: {conf_matrix[i][j]:<5}", end="\t")
    print()  # Start a new line for the next row

# Generate and display the classification report
print("\nClassification Report:")
classification_rep = classification_report(y_test, y_pred)
print(classification_rep)
```

```
Matthews Correlation Coefficient (MCC): 0.27
```




```
Evaluation Metrics:
Accuracy: 0.63
Sensitivity: 0.68
Specificity: 0.61
F1 Score: 0.53
AUC-ROC: 0.67
Balanced Accuracy: 0.65

Confusion Matrix:
True Negative  : 292    False Positive : 183
False Negative : 68     True Positive  : 144

Classification Report:
              precision    recall  f1-score   support

           0       0.81      0.61      0.70       475
           1       0.44      0.68      0.53       212

    accuracy                           0.63       687
   macro avg       0.63      0.65      0.62       687
weighted avg       0.70      0.63      0.65       687
```

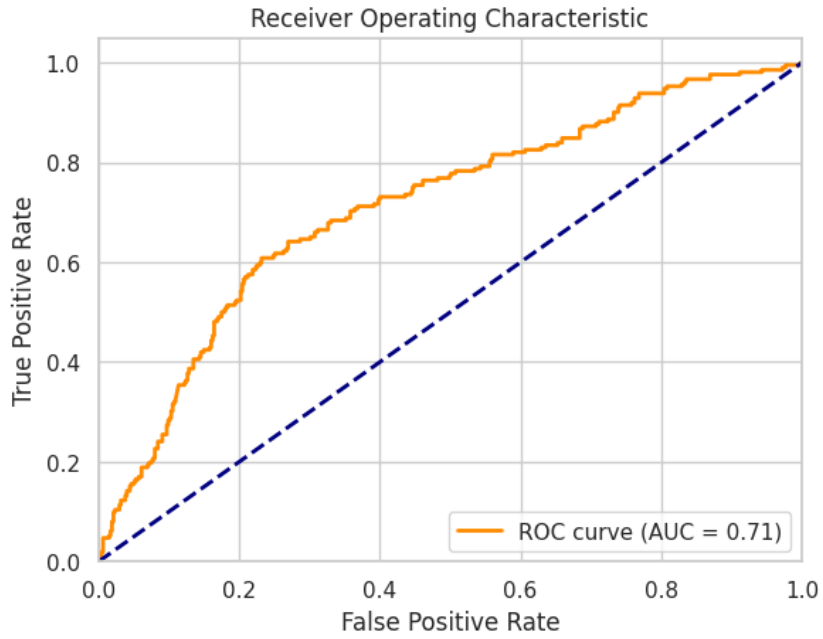
Support Vector Machines

In [ ]:
```
# Perform one-hot encoding
standardized_data_reduced_svm = pd.get_dummies(standardized_data_reduced, columns=['gender' ])
```

In [ ]:
```
# Import necessary libraries
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from sklearn.metrics import (
    precision_score, recall_score, f1_score, accuracy_score, confusion_matrix,
    classification_report, roc_auc_score, roc_curve
)
import matplotlib.pyplot as plt
```

```
response_variable = 'death_flag'
predictor_variables = [col for col in standardized_data_reduced_svm.columns if col != response_variable]

# Create the design matrix (X) and response variable (y)
X = standardized_data_reduced_svm[predictor_variables]
y = standardized_data_reduced_svm[response_variable]

# Split the data into a training set and a test set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

class_counts = y_train.value_counts()
total_samples = len(y_train)

# Tackle the class imbalance problem by using class_weight="balanced"
svm_model = SVC(class_weight="balanced", probability=True)
svm_model.fit(X_train, y_train)

# Make predictions on the test set
y_pred = svm_model.predict(X_test)

# Calculate evaluation metrics
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
tn, fp, fn, tp = conf_matrix.ravel()
sensitivity = tp / (tp + fn)
specificity = tn / (tn + fp)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1_score = f1_score(y_test, y_pred)
y_prob = svm_model.predict_proba(X_test)[:, 1]
roc_auc_svm = roc_auc_score(y_test, y_prob)
fpr_svm, tpr_svm, _ = roc_curve(y_test, y_prob)
balanced_acc = (sensitivity + specificity) / 2
mcc = matthews_corrcoef(y_test, y_pred)

# Add MCC to the list of evaluation metrics
print(f'Matthews Correlation Coefficient (MCC): {mcc:.2f}')
# Plot the ROC curve
plt.figure()
plt.plot(fpr_svm, tpr_svm, color='darkorange', lw=2, label=f'ROC curve (AUC = {roc_auc_svm:.2f})')
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title ('Receiver Operating Characteristic')
plt.legend(loc='lower right')
plt.show()

# Beautify the evaluation metrics
print("Evaluation Metrics:")
print(f'Accuracy: {accuracy:.2f}')
print(f'Sensitivity: {sensitivity:.2f}')
print(f'Specificity: {specificity:.2f}')
print(f'Precision: {precision:.2f}')
print(f'Recall: {recall:.2f}')
print(f'F1 Score: {f1_score:.2f}')
print(f'AUC-ROC: {roc_auc:.2f}')
print(f'Balanced Accuracy: {balanced_acc:.2f}')

# Output the confusion matrix
print("\nConfusion Matrix:")
labels = ["True Negative", "False Positive", "False Negative", "True Positive"]
print(conf_matrix)
for i in range(2):
    for j in range(2):
        print(f"{labels[i * 2 + j]:<15}: {conf_matrix[i][j]:<5}", end="\t")
    print()  # Start a new line for the next row

# Generate and display the classification report
print("\nClassification Report:")
classification_rep = classification_report(y_test, y_pred)
print(classification_rep)
```

```
Matthews Correlation Coefficient (MCC): 0.32
```



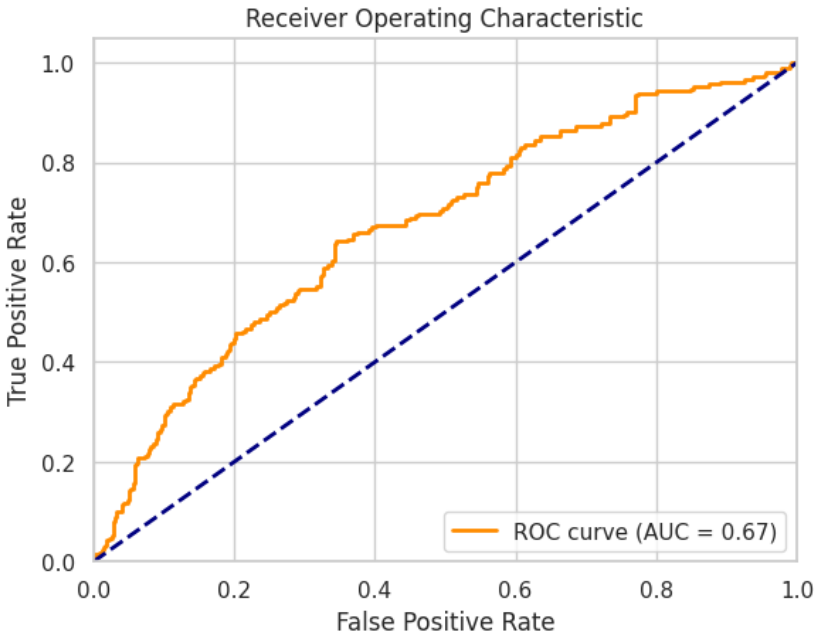

```
Evaluation Metrics:
Accuracy: 0.66
Sensitivity: 0.70
Specificity: 0.64
Precision: 0.47
Recall: 0.70
F1 Score: 0.56
AUC-ROC: 0.67
Balanced Accuracy: 0.67

Confusion Matrix:
[[304 171]
 [ 63 149]]
True Negative  : 304    False Positive : 171
False Negative : 63     True Positive  : 149

Classification Report:
              precision    recall  f1-score   support

           0       0.83      0.64      0.72       475
           1       0.47      0.70      0.56       212

    accuracy                           0.66       687
   macro avg       0.65      0.67      0.64       687
weighted avg       0.72      0.66      0.67       687
```

XGBoost

In [ ]:
```
# Import necessary libraries
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from xgboost import XGBClassifier
from sklearn.metrics import precision_score, recall_score, f1_score, accuracy_score, confusion_matrix, classification_
import matplotlib.pyplot as plt

# Load your data into a DataFrame (your_dataframe)

response_variable = 'death_flag'
predictor_variables = [col for col in standardized_data_reduced.columns if col != response_variable]

# Create the design matrix (X) and response variable (y)
X = standardized_data_reduced[predictor_variables]
y = standardized_data_reduced[response_variable]

# Split the data into a training set and a test set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

class_counts = y_train.value_counts()
total_samples = len(y_train)

# Tackle the class imbalance problem by calculating the scale_pos_weight
positive_samples = sum(y_train == 1)
negative_samples = sum(y_train == 0)
scale_pos_weight = negative_samples / positive_samples

# Create and train the XGBoost model with class imbalance handling
xgb_model = XGBClassifier(scale_pos_weight=scale_pos_weight)
xgb_model.fit(X_train, y_train)

# Make predictions on the test set
```

```
y_pred = xgb_model.predict(X_test)

# Calculate evaluation metrics
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
tn, fp, fn, tp = conf_matrix.ravel()
sensitivity = tp / (tp + fn)
specificity = tn / (tn + fp)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1_score = f1_score(y_test, y_pred)
y_prob = xgb_model.predict_proba(X_test)[:, 1]
roc_auc_xg = roc_auc_score(y_test, y_prob)
fpr_xg, tpr_xg, _ = roc_curve(y_test, y_prob)
balanced_acc = (sensitivity + specificity) / 2
mcc = matthews_corrcoef(y_test, y_pred)

# Add MCC to the list of evaluation metrics
print(f'Matthews Correlation Coefficient (MCC): {mcc:.2f}')
# Plot the ROC curve
plt.figure()
plt.plot(fpr_xg, tpr_xg, color='darkorange', lw=2, label=f'ROC curve (AUC = {roc_auc_xg:.2f})')
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title ('Receiver Operating Characteristic')
plt.legend(loc='lower right')
plt.show()

# Beautify the evaluation metrics
print("Evaluation Metrics:")
print(f'Accuracy: {accuracy:.2f}')
print(f'Sensitivity: {sensitivity:.2f}')
print(f'Specificity: {specificity:.2f}')
print(f'Precision: {precision:.2f}')
print(f'Recall: {recall:.2f}')
print(f'F1 Score: {f1_score:.2f}')
print(f'AUC-ROC: {roc_auc:.2f}')
print(f'Balanced Accuracy: {balanced_acc:.2f}')

# Output the confusion matrix
print("\nConfusion Matrix:")
labels = ["True Negative", "False Positive", "False Negative", "True Positive"]
print(conf_matrix)
for i in range(2):
    for j in range(2):
        print(f"{labels[i * 2 + j]:<15}: {conf_matrix[i][j]:<5}", end="\t")
    print()  # Start a new line for the next row

# Generate and display the classification report
print("\nClassification Report:")
classification_rep = classification_report(y_test, y_pred)
print(classification_rep)
```

```
Matthews Correlation Coefficient (MCC): 0.25
```



```
Evaluation Metrics:
Accuracy: 0.68
Sensitivity: 0.48
Specificity: 0.77
Precision: 0.48
Recall: 0.48
F1 Score: 0.48
AUC-ROC: 0.67
Balanced Accuracy: 0.63

Confusion Matrix:
[[366 109]
 [110 102]]
True Negative  : 366    False Positive : 109
False Negative : 110    True Positive  : 102

Classification Report:
              precision    recall  f1-score   support

           0       0.77      0.77      0.77       475
           1       0.48      0.48      0.48       212

    accuracy                           0.68       687
   macro avg       0.63      0.63      0.63       687
weighted avg       0.68      0.68      0.68       687
```

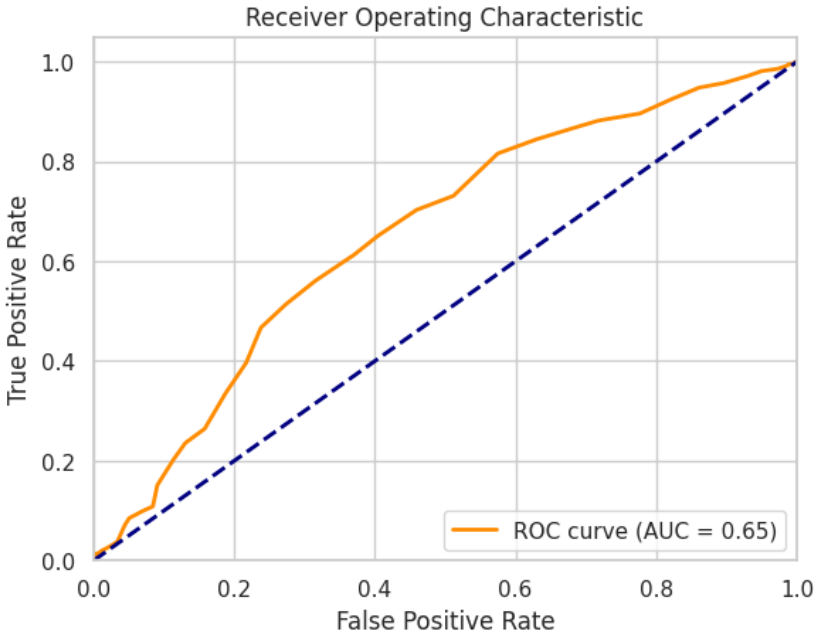
Random Forest

In [ ]:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report, roc_auc_score, confusion_matrix, precision_score, r
import matplotlib.pyplot as plt

# Load your data into a DataFrame (your_dataframe)

response_variable = 'death_flag'
predictor_variables = [col for col in standardized_data_reduced.columns if col != response_variable]

# Create the design matrix (X) and response variable (y)
X = standardized_data_reduced[predictor_variables]
y = standardized_data_reduced[response_variable]

# Split the data into a training set and a test set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Create and train the Random Forest classifier
clf = RandomForestClassifier(n_estimators=best_n_estimators, random_state=42, class_weight='balanced')
clf.fit(X_train, y_train)

# Make predictions on the test set
y_pred = clf.predict(X_test)

# Calculate evaluation metrics
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
tn, fp, fn, tp = conf_matrix.ravel()
sensitivity = tp / (tp + fn)
specificity = tn / (tn + fp)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1_score = f1_score(y_test, y_pred)
y_prob = clf.predict_proba(X_test)[:, 1]
roc_auc_rf = roc_auc_score(y_test, y_prob)
fpr_rf, tpr_rf, _ = roc_curve(y_test, y_prob)
balanced_acc = (sensitivity + specificity) / 2
mcc = matthews_corrcoef(y_test, y_pred)

# Add MCC to the list of evaluation metrics
print(f'Matthews Correlation Coefficient (MCC): {mcc:.2f}')

# Plot the ROC curve
plt.figure()
plt.plot(fpr_rf, tpr_rf, color='darkorange', lw=2, label=f'ROC curve (AUC = {roc_auc_rf:.2f})')
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic')
plt.legend(loc='lower right')
plt.show()

# Beautify the evaluation metrics
print("Evaluation Metrics:")
print(f'Accuracy: {accuracy:.2f}')
print(f'Sensitivity: {sensitivity:.2f}')
print(f'Specificity: {specificity:.2f}')
print(f'Precision: {precision:.2f}')
print(f'Recall: {recall:.2f}')
```

```
print(f'F1 Score: {f1_score:.2f}')
print(f'AUC-ROC: {roc_auc:.2f}')
print(f'Balanced Accuracy: {balanced_acc:.2f}')

# Output the confusion matrix
print("\nConfusion Matrix:")
labels = ["True Negative", "False Positive", "False Negative", "True Positive"]
print(conf_matrix)
for i in range(2):
    for j in range(2):
        print(f"{labels[i * 2 + j]:<15}: {conf_matrix[i][j]:<5}", end="\t")
    print()  # Start a new line for the next row

# Generate and display the classification report
print("\nClassification Report:")
classification_rep = classification_report(y_test, y_pred)
print(classification_rep)
```

```
Matthews Correlation Coefficient (MCC): 0.09
```



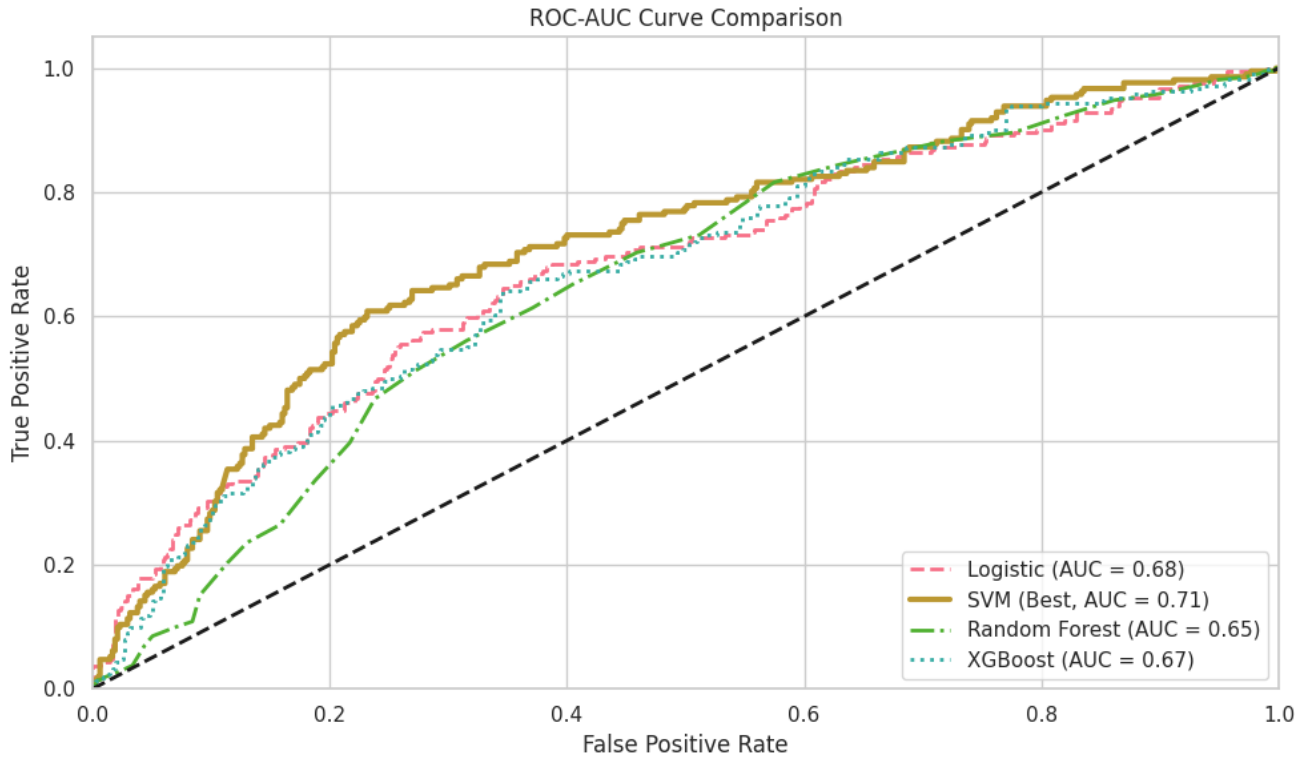

```
Evaluation Metrics:
Accuracy: 0.68
Sensitivity: 0.15
Specificity: 0.91
Precision: 0.43
Recall: 0.15
F1 Score: 0.22
AUC-ROC: 0.67
Balanced Accuracy: 0.53

Confusion Matrix:
[[432  43]
 [180  32]]
True Negative  : 432    False Positive : 43
False Negative : 180    True Positive  : 32

Classification Report:
              precision    recall  f1-score   support

           0       0.71      0.91      0.79       475
           1       0.43      0.15      0.22       212

    accuracy                           0.68       687
   macro avg       0.57      0.53      0.51       687
weighted avg       0.62      0.68      0.62       687
```

AUC Comparision

In [ ]:

```
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import roc_curve, auc

# Define the ROC curves for your models
roc_curves = {
    'Logistic': (fpr_log, tpr_log),
    'SVM': (fpr_svm, tpr_svm),
    'Random Forest': (fpr_rf, tpr_rf),
    'XGBoost': (fpr_xg, tpr_xg)
}

# Determine the best-performing model (e.g., highest AUC)
best_model = max(roc_curves, key=lambda k: auc(roc_curves[k][0], roc_curves[k][1]))
```

```
# Set Seaborn style with a bright color palette
sns.set(style="whitegrid")
sns.set_palette("husl")

# Create a figure and axis
plt.figure(figsize=(10, 6))

# Set custom line styles
line_styles = ['--', '-', '-.', ':']

# Plot ROC curves for each model
for i, (label, (fpr, tpr)) in enumerate(roc_curves.items()):
    roc_auc = auc(fpr, tpr)
    if label == best_model:
        plt.plot(fpr, tpr, linestyle=line_styles[i], lw=3, label=f'{label} (Best, AUC = {roc_auc:.2f})')
    else:
        plt.plot(fpr, tpr, linestyle=line_styles[i], lw=2, label=f'{label} (AUC = {roc_auc:.2f})')

# Plot the diagonal line (random classifier)
plt.plot([0, 1], [0, 1], 'k--', lw=2)

# Set labels and title
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC-AUC Curve Comparison')
plt.legend(loc='lower right')

# Customize the appearance of the plot
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xticks([0.0, 0.2, 0.4, 0.6, 0.8, 1.0])
plt.yticks([0.0, 0.2, 0.4, 0.6, 0.8, 1.0])

# Save or display the plot
plt.tight_layout()
plt.savefig('roc_auc_comparison_final.png')  # Save the plot as an image
plt.show()
```



Clustering Task

Selected Features

In [ ]:
```
selected_features
```

Out[ ]:
```
['total_vent_stays',
 'avg_WBC_count',
 'avg_bmi',
 'age',
 'charlson_comorbidity_index',
 'Heart Rate',
 'Temperature Celsius',
 'Alanine Aminotransferase (ALT)',
 'Alkaline Phosphatase',
 'Anion Gap',
 'Basophils',
 'Chloride',
 'Creatinine',
 'Eosinophils',
 'Free Calcium']
```

Data Prepration

In [ ]:
```
cluster_dataframe = preprocessed_cohort.copy()
```

In [ ]:
```
cluster_dataframe = cluster_dataframe[selected_features + ['death_flag','gender','bronchoscopy']]
```

In [ ]:
```
cluster_dataframe['death_flag'].value_counts()
```

Out[ ]:
```
0    1510
1     780
Name: death_flag, dtype: int64
```

Gender Mapping

In [ ]:
```
gender_mapping = {'M': 0, 'F': 1}

cluster_dataframe['gender'] = cluster_dataframe['gender'].replace(gender_mapping)
```

```
<ipython-input-176-874157035c01>:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning
-a-view-versus-a-copy
  cluster_dataframe['gender'] = cluster_dataframe['gender'].replace(gender_mapping)
```

Standardize the data

In [ ]:
```
import pandas as pd
from sklearn.preprocessing import StandardScaler

# Assuming your DataFrame is named cluster_dataframe
columns_to_exclude = ['death_flag', 'gender', 'bronchoscopy']
columns_to_standardize = [col for col in cluster_dataframe.columns if col not in columns_to_exclude]

# Standardize the selected columns and store in a new DataFrame
standardized_data = StandardScaler().fit_transform(cluster_dataframe[columns_to_standardize])
standardized_df = pd.DataFrame(standardized_data, columns=columns_to_standardize)

# Merge the standardized DataFrame with the excluded columns
for col in columns_to_exclude:
    standardized_df[col] = cluster_dataframe[col]
```

Outlier Removal

In [ ]:
```
from sklearn.ensemble import IsolationForest

# Create an Isolation Forest model with the contamination parameter
iso_forest = IsolationForest(contamination=0.05, random_state=0)

# Fit the Isolation Forest model on your data
iso_forest.fit(standardized_df)

# Perform outlier detection and add the 'outlier' column to the DataFrame
standardized_df['outlier'] = iso_forest.predict(standardized_df)

# Filter out the outliers (retain only non-outliers)
standardized_df = standardized_df[standardized_df['outlier'] != -1]

# Drop the 'outlier' column if you no longer need it
standardized_df = standardized_df.drop(columns=['outlier'])
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439: UserWarning: X does not have valid feature names, but Is
olationForest was fitted with feature names
  warnings.warn(
```

K-Means Clustering

Function definition

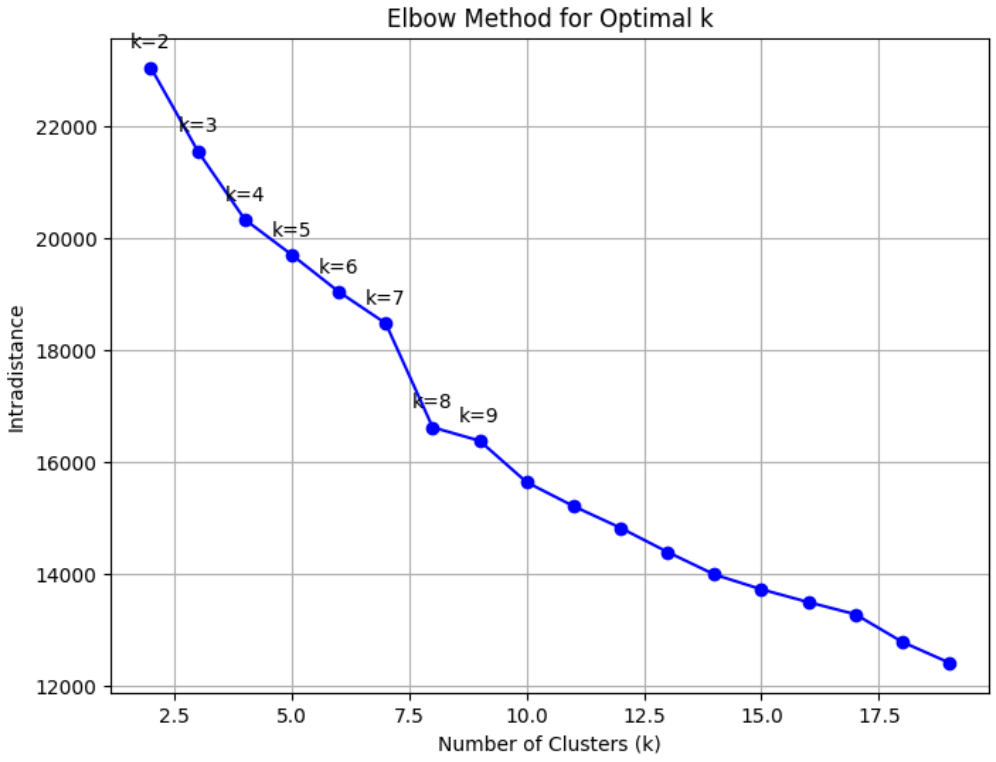
In [ ]:
```
from sklearn.cluster import KMeans  # For K-means clustering
from sklearn.decomposition import PCA  # For Principal Component Analysis
from sklearn.metrics import silhouette_score # For Silhouette score

#Defining a function that will perform k_means clustering for different k values
def k_means_clustering(tot_clusters, data, clustering_columns ):

    data = data.copy()

    kmeans = KMeans(n_clusters= tot_clusters, random_state=0, n_init=10, init = "k-means++")
    kmeans.fit(data[clustering_columns])

    #Find which cluster each data-point belongs to
    clusters = kmeans.predict(data[clustering_columns])

    #Add the cluster information as a new column to our DataFrame
    data["cluster"] = clusters

    return data

#Display the cumulative variance plot that is used to decide the number of PCA components
def cummulative_variance_plot(dataset, columns, title):

    #First we determine the number of principal components
    pca = PCA()
    pca.fit(dataset[columns])

    # Get explained variance ratio for each PC
    explained_variance_ratio = pca.explained_variance_ratio_

    # Create a Scree plot
    plt.figure(figsize=(8, 6))
    plt.plot(np.cumsum(explained_variance_ratio))
    plt.xlabel("Principal Component")
    plt.ylabel("Explained Variance")
    plt.title(f"cumulative variance plot for {title}")
    plt.grid(True)
    plt.show()


# Perform PCA analysis and visualizes first 2 Principal components (2-D visualization)
def plot_pc_analysis(data, num_clusters):

    # PCA with 2 principal components
    pca_2d = PCA(n_components=2)

    # Calculate the 2 principal components
    PCs_2d = pd.DataFrame(pca_2d.fit_transform(data.drop(["cluster"], axis=1)))
    PCs_2d.columns = ["PC1", "PC2"]

    # Combine original data with the 2 principal components
    pca_df = pd.concat([data, PCs_2d], axis=1, join='inner')

    # Colors for clusters (red, green, blue, yellow)
    cluster_colors = ['r', 'g', 'b', 'y']

    # Create a scatter plot for each cluster
    plt.figure(figsize=(8, 6))

    for cluster_id in range(num_clusters):
        # Filter data for the current cluster
        cluster_data = pca_df[pca_df['cluster'] == cluster_id]

        # Plot cluster data points
        plt.scatter(cluster_data['PC1'], cluster_data['PC2'], label=f'Cluster {cluster_id}', c=cluster_colors[cluster_

        # Calculate and plot the cluster centroid
        centroid = cluster_data[['PC1', 'PC2']].mean()
        plt.scatter(centroid[0], centroid[1], marker='o', s=50, c='black', label=f'Centroid {cluster_id}')

        # Add cluster number label near the centroid
        plt.text(centroid[0] + 0.1, centroid[1] + 0.1, f'Cluster {cluster_id}', fontsize=12)


    # Set the plot components
    plt.title(f'PCA Components for Clusters K = {num_clusters}')
    plt.xlabel('PC1')
    plt.ylabel('PC2')
    plt.grid(True)

    legend_elements = [plt.Line2D([0], [0], marker='o', color='w', markersize=10, markerfacecolor=cluster_colors[clust
                       for cluster_id in range(num_clusters)]
    plt.legend(handles=legend_elements, loc='upper right')


    plt.grid(True)
    # Display the scatter plot with centroids
    plt.show()
```

In [ ]:
```
columns_to_exclude = ['death_flag']
clustering_columns = [col for col in standardized_df.columns if col not in columns_to_exclude]
```

In [ ]:
```
kmeans = KMeans(n_clusters=3, random_state=0, n_init='auto', init="k-means++")

# Fit the KMeans model to the scaled dataset using selected clustering columns
kmeans.fit(standardized_df[clustering_columns])
```

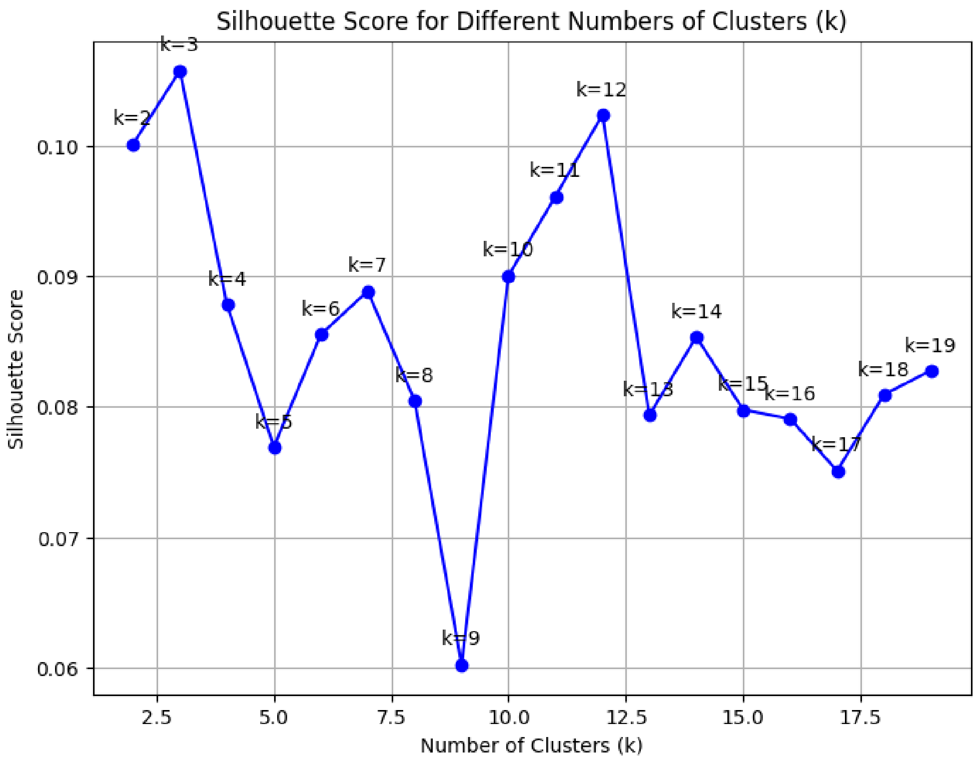
Elbow method to find the optimal k (number of clusters)

In [ ]:
```
# Calculate the intradistance for different values of k
intradistance = []

# List to store silhouette scores for different k values
silhouette_scores = []

# Loop through a range of k values from 2 to 19
for k in range(2, 20):
    # Initialize KMeans with the current k value
    kmeans = KMeans(n_clusters=k, random_state=0, n_init='auto', init="k-means++")

    # Fit the KMeans model to the scaled dataset using selected clustering columns
    kmeans.fit(standardized_df[clustering_columns])

    # Calculate and store the inertia (intradistance) for this k value
    intradistance.append(kmeans.inertia_)

    # Get cluster labels for data points
    cluster_labels = kmeans.fit_predict(standardized_df[clustering_columns])

    # Calculate the silhouette score and store it
    silhouette_avg = silhouette_score(standardized_df[clustering_columns], cluster_labels)
    silhouette_scores.append(silhouette_avg)

# Plot the elbow curve
plt.figure(figsize=(8, 6))
plt.plot(range(2, 20), intradistance, marker='o', linestyle='-', color='b')
plt.xlabel('Number of Clusters (k)')
plt.ylabel('Intradistance')
plt.title('Elbow Method for Optimal k')
plt.grid(True)

# Annotate the points on the graph with k values
for i, k in enumerate(range(2, 10)):
    plt.annotate(f'k={k}', (k, intradistance[i]), textcoords="offset points", xytext=(0, 10), ha='center')

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



In [ ]:
```
# Plot the silhouette scores
plt.figure(figsize=(8, 6))
plt.plot(range(2, 20), silhouette_scores, marker='o', linestyle='-', color='b')
plt.title('Silhouette Score for Different Numbers of Clusters (k)')
plt.xlabel('Number of Clusters (k)')
plt.ylabel('Silhouette Score')
```

```
plt.grid(True)

# Annotate the points with k values
for i, k in enumerate(range(2, 20)):
    plt.annotate(f'k={k}', (range(2, 20)[i], silhouette_scores[i]), textcoords="offset points", xytext=(0,10), ha='cer

plt.show()
```



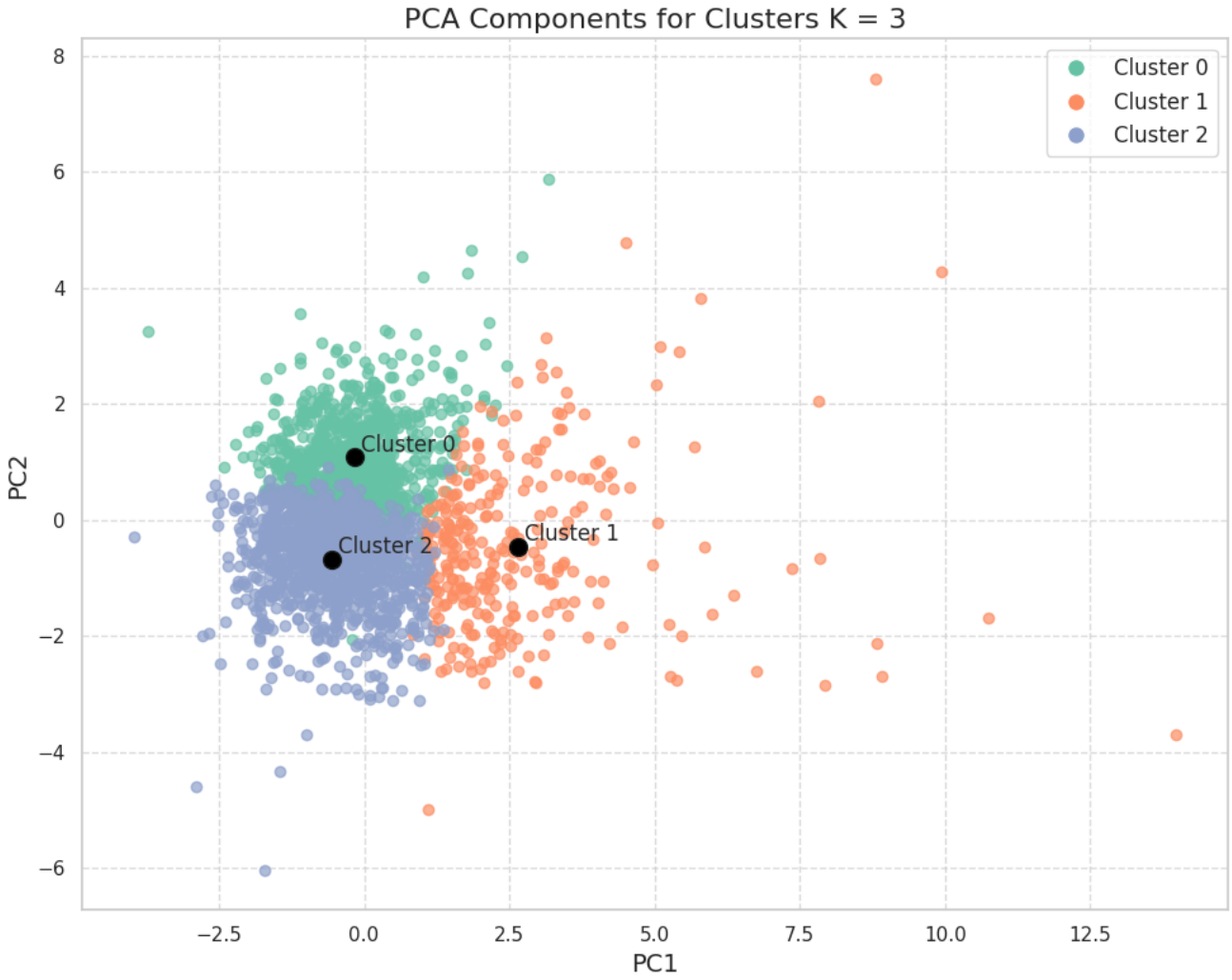
Number of Clusters = 3

Clustering Plot

In [ ]:
```
standardized_data_reduced_km_2 = k_means_clustering(3, standardized_data_reduced, clustering_columns)
```

In [ ]:
```
#PCA analysis of dataset with 3 clusters
plot_pc_analysis(standardized_data_reduced_km_2,3)
```

```
<ipython-input-49-b54e9936686e>:116: UserWarning: *c* argument looks like a single numeric RGB or RGBA sequence, whic
h should be avoided as value-mapping will have precedence in case its length matches with *x* & *y*.  Please use the
*color* keyword-argument or provide a 2D array with a single row if you intend to specify the same RGB or RGBA value
for all points.
  plt.scatter(cluster_data['PC1'], cluster_data['PC2'], label=f'Cluster {cluster_id}', c=palette[cluster_id], alpha=
0.7)
<ipython-input-49-b54e9936686e>:116: UserWarning: *c* argument looks like a single numeric RGB or RGBA sequence, whic
h should be avoided as value-mapping will have precedence in case its length matches with *x* & *y*.  Please use the
*color* keyword-argument or provide a 2D array with a single row if you intend to specify the same RGB or RGBA value
for all points.
  plt.scatter(cluster_data['PC1'], cluster_data['PC2'], label=f'Cluster {cluster_id}', c=palette[cluster_id], alpha=
0.7)
<ipython-input-49-b54e9936686e>:116: UserWarning: *c* argument looks like a single numeric RGB or RGBA sequence, whic
h should be avoided as value-mapping will have precedence in case its length matches with *x* & *y*.  Please use the
*color* keyword-argument or provide a 2D array with a single row if you intend to specify the same RGB or RGBA value
for all points.
  plt.scatter(cluster_data['PC1'], cluster_data['PC2'], label=f'Cluster {cluster_id}', c=palette[cluster_id], alpha=
0.7)
```



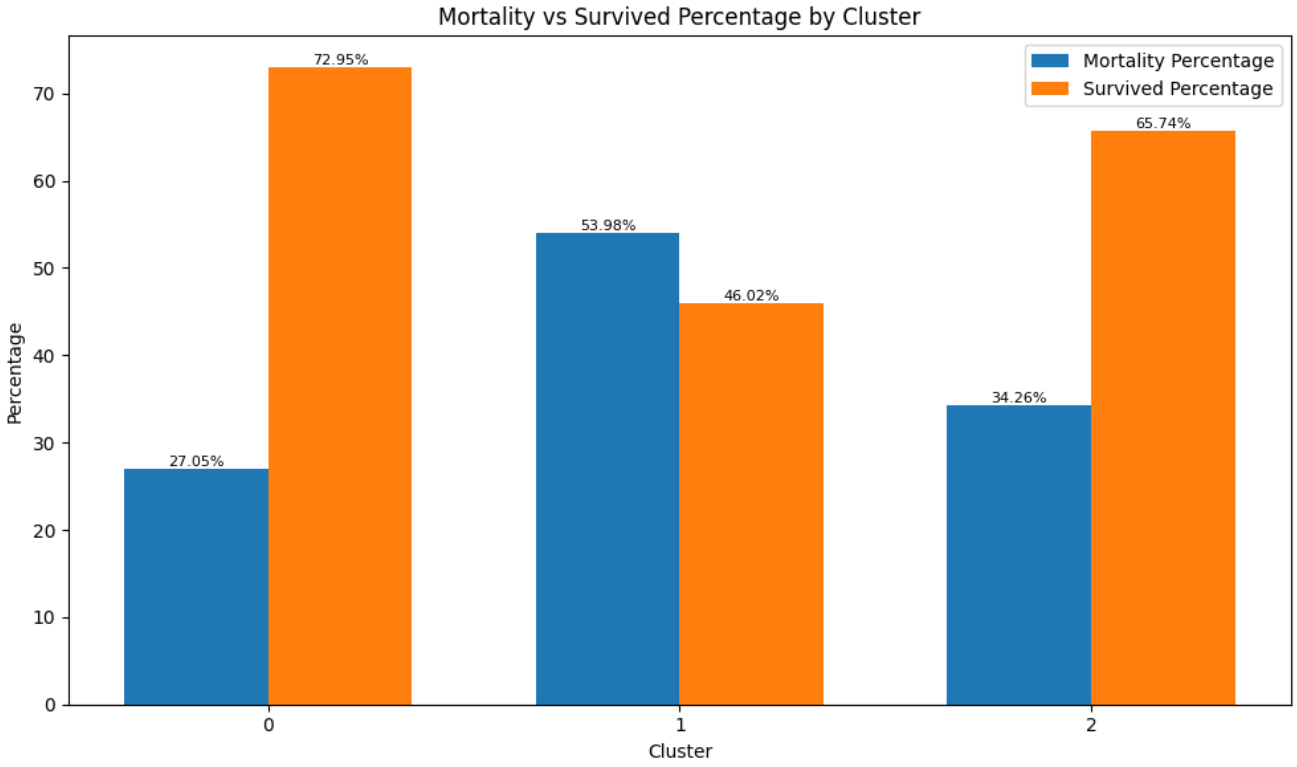
Clustering Analysis

In [ ]:
```
cluster_groups = standardized_data_reduced_km_2.groupby('cluster')

# Calculate the mortality percentage within each cluster
mortality_percentage = cluster_groups['death_flag'].mean() * 100

# Print or display the mortality percentage for each cluster
print("Mortality Percentage by Cluster:")
print(mortality_percentage)

# Calculate mortality percentages by cluster
mortality_percentage = standardized_data_reduced_km_2.groupby('cluster')['death_flag'].mean() * 100
# Calculate survived percentages by subtracting mortality percentage from 100
survived_percentage = 100 - mortality_percentage

# Beautify the output
mortality_percentage = mortality_percentage.reset_index()
mortality_percentage.rename(columns={'death_flag': 'Mortality Percentage'}, inplace=True)

# Add the "Survived Percentage" column
mortality_percentage['Survived Percentage'] = survived_percentage

print(mortality_percentage)

# Calculate gender percentages by cluster
gender_percentage = standardized_data_reduced_km_2.groupby('cluster')['gender'].mean() * 100

# Beautify the output
gender_percentage = gender_percentage.reset_index()
gender_percentage.rename(columns={'gender': 'Female Percentage'}, inplace=True)
gender_percentage['Male Percentage'] = 100 - gender_percentage['Female Percentage']

print(gender_percentage)

# Calculate gender percentages by cluster
bronchoscopy_percentage = standardized_data_reduced_km_2.groupby('cluster')['bronchoscopy'].mean() * 100

# Beautify the output
bronchoscopy_percentage = bronchoscopy_percentage.reset_index()
bronchoscopy_percentage.rename(columns={'bronchoscopy': 'With bronchoscopy'}, inplace=True)
bronchoscopy_percentage['Without bronchoscopy'] = 100 - bronchoscopy_percentage['With bronchoscopy']

print(bronchoscopy_percentage)
```

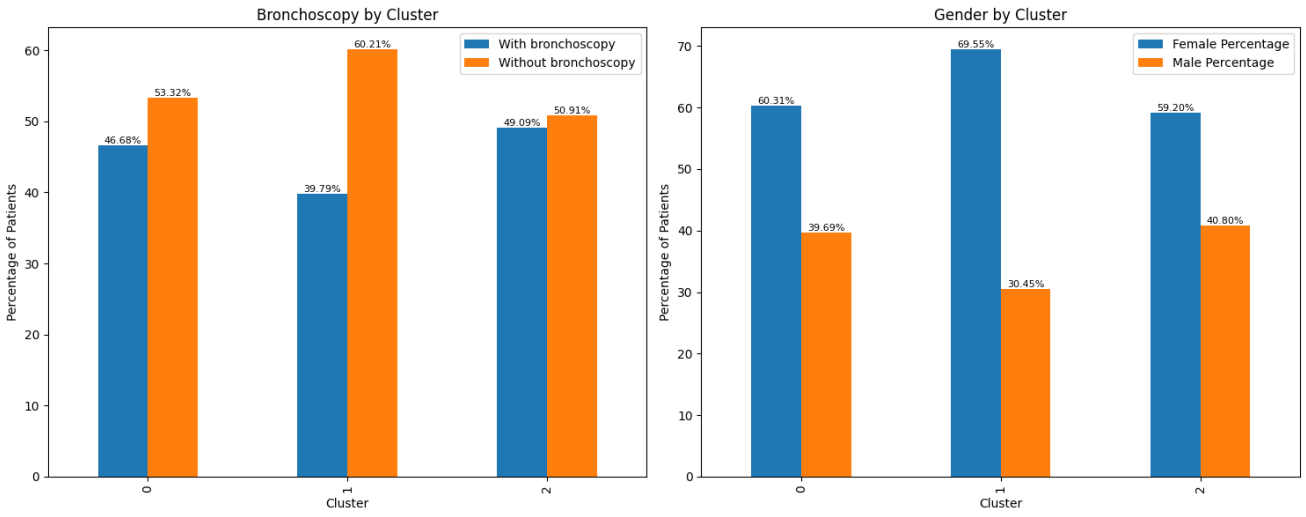
```
Mortality Percentage by Cluster:
cluster
0    27.054610
1    53.979239
2    34.255913
Name: death_flag, dtype: float64
   cluster  Mortality Percentage  Survived Percentage
0        0             27.054610            72.945390
1        1             53.979239            46.020761
2        2             34.255913            65.744087
   cluster  Female Percentage  Male Percentage
0        0          60.308057        39.691943
1        1          69.550173        30.449827
2        2          59.204840        40.795160
   cluster  With bronchoscopy  Without bronchoscopy
0        0          46.682464             53.317536
1        1          39.792388             60.207612
2        2          49.092481             50.907519
```

In [ ]:

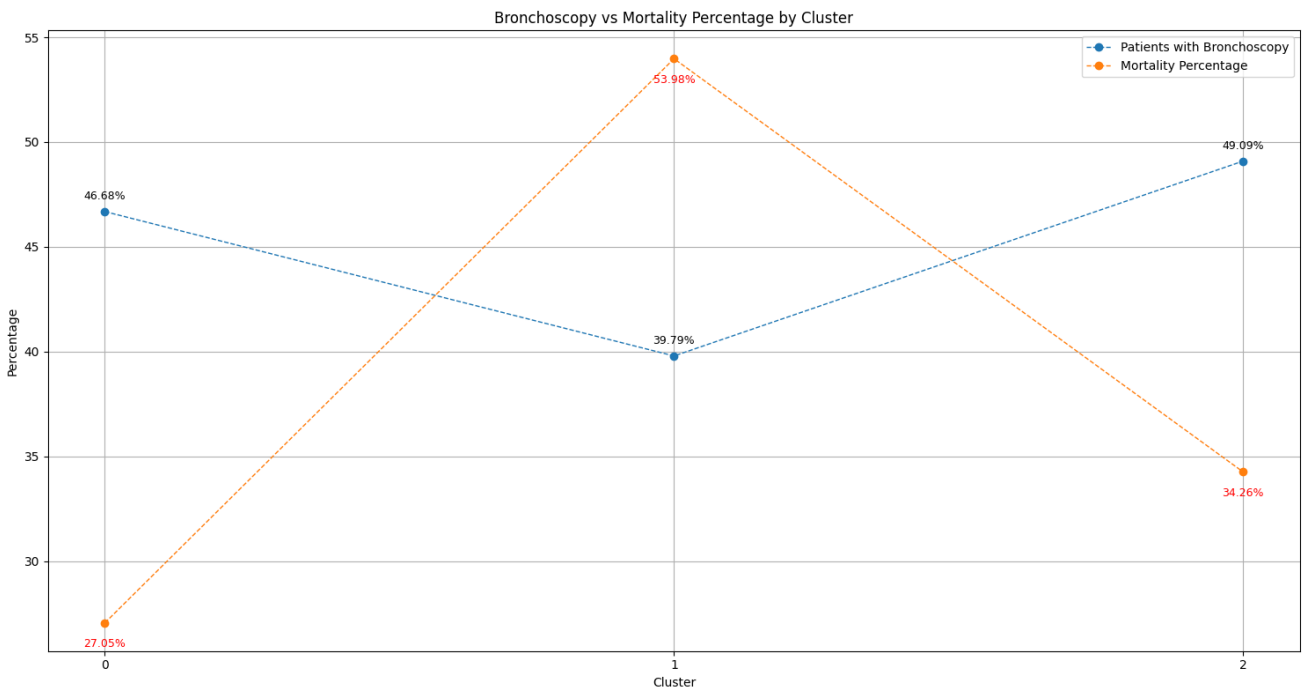
```python
import matplotlib.pyplot as plt

# Create a grouped bar plot
fig, ax = plt.subplots(figsize=(10, 6))

cluster = mortality_percentage['cluster']
mortality_percentage_values = mortality_percentage['Mortality Percentage']
survived_percentage_values = mortality_percentage['Survived Percentage']

bar_width = 0.35
bar_positions = range(len(cluster))

bar1 = plt.bar(bar_positions, mortality_percentage_values, bar_width, label='Mortality Percentage')
bar2 = plt.bar([pos + bar_width for pos in bar_positions], survived_percentage_values, bar_width, label='Survived Perc

# Set the x-axis labels
ax.set_xticks([pos + bar_width / 2 for pos in bar_positions])
ax.set_xticklabels(cluster)

# Labeling and styling
plt.xlabel('Cluster')
plt.ylabel('Percentage')
plt.title('Mortality vs Survived Percentage by Cluster')
plt.legend()

# Add percentage values above the bars
def add_percentage_labels(bars):
    for bar in bars:
        height = bar.get_height()
        ax.annotate(f'{height:.2f}%', (bar.get_x() + bar.get_width() / 2, height),
                    ha='center', va='bottom', fontsize=8)

add_percentage_labels(bar1)
add_percentage_labels(bar2)

plt.tight_layout()
plt.savefig('mortality.png', dpi=300, bbox_inches='tight')
# Show the graph
plt.show()
```



In [ ]:
```
import matplotlib.pyplot as plt

# Create subplots for the graphs
fig, axes = plt.subplots(1, 2, figsize=(15, 6))

# Plot the Bronchoscopy vs Mortality
bronchoscopy_percentage.plot(x='cluster', y=['With bronchoscopy', 'Without bronchoscopy'], kind='bar', ax=axes[0])
axes[0].set_title('Bronchoscopy by Cluster')
axes[0].set_xlabel('Cluster')
axes[0].set_ylabel('Percentage of Patients')
axes[0].legend(['With bronchoscopy', 'Without bronchoscopy'])

# Plot the Gender vs Mortality
gender_percentage.plot(x='cluster', y=['Female Percentage', 'Male Percentage'], kind='bar', ax=axes[1])
axes[1].set_title('Gender by Cluster')
axes[1].set_xlabel('Cluster')
axes[1].set_ylabel('Percentage of Patients')
axes[1].legend(['Female Percentage', 'Male Percentage'])

# Add percentage values above the bars
def add_percentage_labels(ax):
    for p in ax.patches:
        ax.annotate(f'{p.get_height():.2f}%', (p.get_x() + p.get_width() / 2, p.get_height()),
                    ha='center', va='bottom', fontsize=8)

add_percentage_labels(axes[0])
add_percentage_labels(axes[1])

# Adjust the layout
plt.tight_layout()

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



In [ ]:
```
import matplotlib.pyplot as plt
clusters = mortality_percentage['cluster'].unique()
# Create a figure
fig, axes = plt.subplots(1, 1, figsize=(15, 8))

# Colors for the lines and markers
line_colors = ['tab:blue', 'tab:orange']
marker_colors = ['blue', 'orange']

# Style the lines
line_styles = ['--', '--']

# Linewidth
line_width = 1

# Marker size
marker_size = 6

# With bronchoscopy vs Mortality Percentage
axes.plot(clusters, bronchoscopy_percentage['With bronchoscopy'], label='Patients with Bronchoscopy', marker='o', mark
axes.plot(clusters, mortality_percentage['Mortality Percentage'], label='Mortality Percentage', marker='o', markersize

# Annotate points with percentage values
for i, txt in enumerate(bronchoscopy_percentage['With bronchoscopy']):
    axes.annotate(f'{txt:.2f}%', (clusters[i], txt), textcoords="offset points", xytext=(0, 10), ha='center', fontsize

for i, txt in enumerate(mortality_percentage['Mortality Percentage']):
    axes.annotate(f'{txt:.2f}%', (clusters[i], txt), textcoords="offset points", xytext=(0, -20), color='red', ha='cer

# Labels and title
axes.set_xlabel('Cluster')
axes.set_ylabel('Percentage')
axes.set_title('Bronchoscopy vs Mortality Percentage by Cluster')

# Set the x-axis to display integer clusters only
axes.set_xticks(clusters)

# Legend
axes.legend(loc='best')

# Grid
axes.grid(True)

# Tight layout
plt.tight_layout()

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



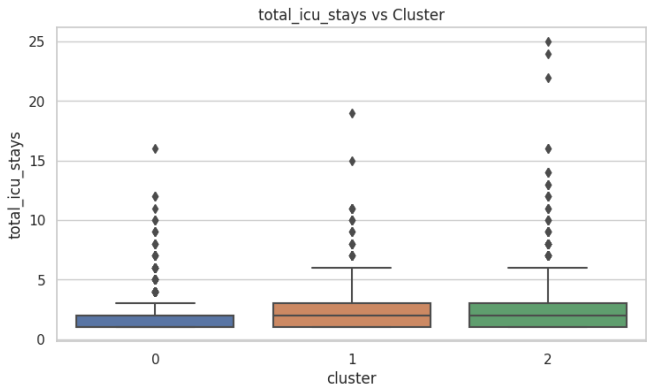
In [ ]:
```
# create 3 different datasets (one for each cluster)
cluster_0 = destandardized_data[destandardized_data["cluster"] == 0]
cluster_1 = destandardized_data[destandardized_data["cluster"] == 1]
cluster_2 = destandardized_data[destandardized_data["cluster"] == 2]
```

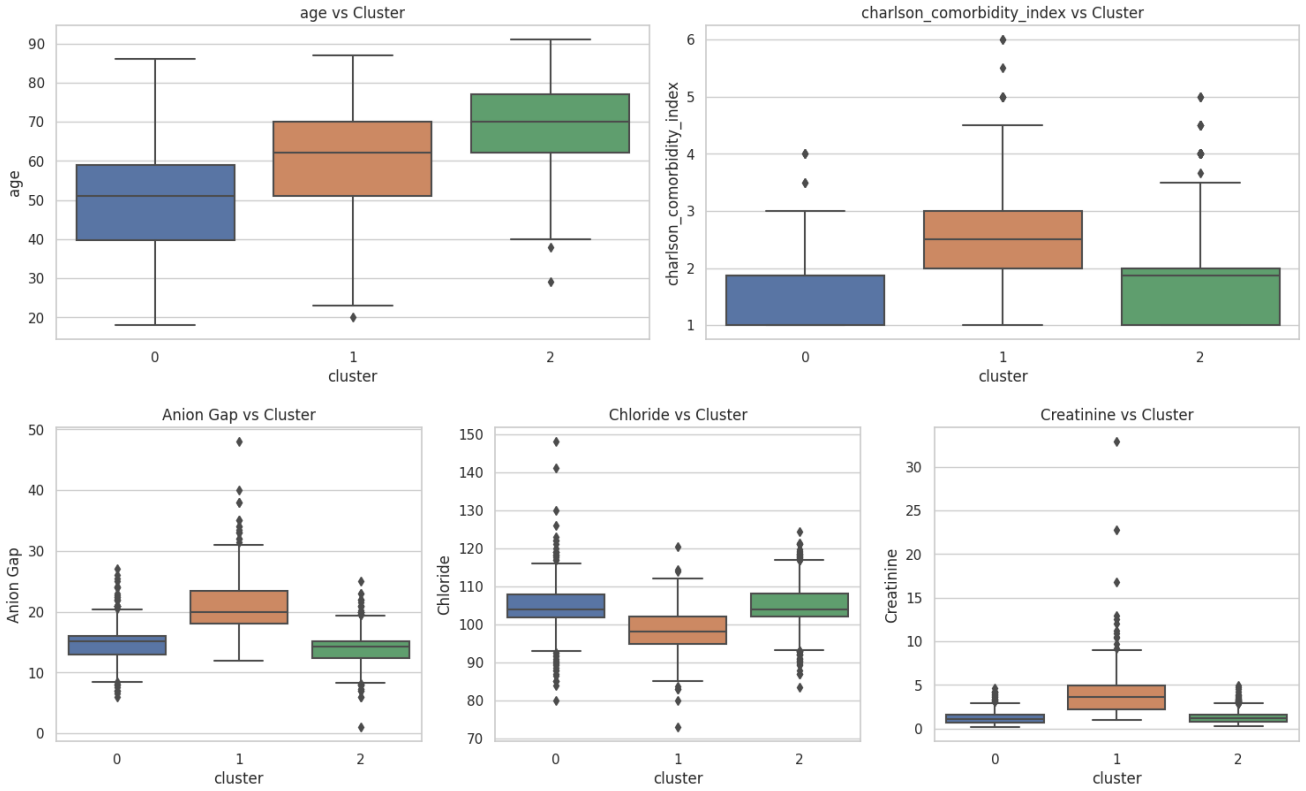
In [ ]:
```
print(cluster_2[['total_icu_stays', 'total_vent_stays', 'avg_bmi',
                 'age', 'charlson_comorbidity_index', 'Heart Rate',
                 'Temperature Celsius', 'Anion Gap', 'Chloride', 'Creatinine']].describe())
```

```
       total_icu_stays  total_vent_stays      avg_bmi          age  \
count      1157.000000       1157.000000  1157.000000  1157.000000
mean          2.420916          2.854797    28.275788    69.442304
std           2.379631          2.231344     7.103919    10.329259
min           1.000000          1.000000     0.308642    29.000000
25%           1.000000          1.000000    23.689398    62.000000
50%           2.000000          2.000000    28.061224    70.000000
75%           3.000000          3.000000    31.409788    77.000000
max          25.000000         25.000000    96.604388    91.000000

       charlson_comorbidity_index   Heart Rate  Temperature Celsius  \
count                 1157.000000  1157.000000          1157.000000
mean                     1.896469    84.808210            37.096441
std                      0.796656    12.399042             1.295236
min                      1.000000    40.556818             0.000000
25%                      1.000000    76.600000            37.121196
50%                      1.861691    84.349164            37.121196
75%                      2.000000    93.483578            37.121196
max                      5.000000   133.044776            55.270000

         Anion Gap     Chloride   Creatinine
count  1157.000000  1157.000000  1157.000000
mean     13.996107   104.895903     1.264473
std       2.523404     5.214634     0.658748
min       1.000000    83.500000     0.268421
25%      12.375000   102.062500     0.750000
50%      14.222222   104.000000     1.150000
75%      15.202186   108.000000     1.598185
max      25.000000   124.428571     4.933333
```

In [ ]:
```python
def draw_box_plot(columns_to_plot, j):
    # Group the data by the 'cluster' column
    cluster_groups = destandardized_data.groupby('cluster')

    # Create subplots for the box plots
    fig, axes = plt.subplots(1, len(columns_to_plot), figsize=(15, 5))

    # Loop through the selected columns
    for i, col in enumerate(columns_to_plot):
        ax = axes[i]
        ax.set_title(f'{col} vs Cluster')
        ax.set_ylabel(col)  # Set y-axis title to the column name
        ax.set_xlabel('Cluster')

        # Create box plots for each column within each cluster
        sns.boxplot(x='cluster', y=col, data=destandardized_data, ax=ax)

    # Adjust the layout
    #plt.suptitle(f"Box Plots {j} by Cluster")
    plt.tight_layout()
    plt.subplots_adjust(top=0.85)

    # Display the box plots
    plt.show()
```

In [ ]:
```python
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns

# Set the style for better readability (you can adjust the style to your preference)
sns.set(style="whitegrid")

# Define the combinations of features
feature_combinations = [
    ['total_vent_stays', 'total_icu_stays'],
    ['age', 'charlson_comorbidity_index'],
    ['Anion Gap', 'Chloride', 'Creatinine' ]
]
j = 1
for x in feature_combinations:
    draw_box_plot(x, j)
    j+=1
```





In [ ]:
```
# Create an empty DataFrame to store the results
result_df = pd.DataFrame(columns=['cluster'] + [f'{col}_within_range' for col in columns_to_average])

# Check if the average values are within the reference range for each cluster
for cluster in average_values_by_cluster['cluster']:
    cluster_data = average_values_by_cluster[average_values_by_cluster['cluster'] == cluster]
    within_range = {}
    for col in columns_to_average:
        if col in ref_ranges:
            col_lower, col_upper = ref_ranges[col]
            #print (ref_ranges[col])
            within_range[f'{col}_within_range'] = (cluster_data[col].values[0] > col_lower) and (cluster_data[col].val
        else:
            within_range[f'{col}_within_range'] = 'null'
    result_df.loc[len(result_df)] = [cluster, *list(within_range.values())]

result_df = result_df.drop(columns=['Heart Rate_within_range', 'Temperature Celsius_within_range'])
```

In [ ]:
```
result_df
```

Out[ ]:

	cluster	Alanine Aminotransferase (ALT)_within_range	Alkaline Phosphatase_within_range	Anion Gap_within_range	Basophils_within_range	Chloride_within_range	Creatinine_within_
0	0	False	False	True	null	True	
1	1	False	False	False	null	True	
2	2	False	False	True	null	True	

In [ ]:
```
# Merge the dataframes by the 'cluster' column
merged_df = pd.merge(average_values_by_cluster, result_df, on='cluster', how='inner')

merged_df
```

Out[ ]:

	cluster	Heart Rate	Temperature Celsius	Alanine Aminotransferase (ALT)	Alkaline Phosphatase	Anion Gap	Basophils	Chloride	Creatinine	Eosinophils	Free Calcium	Al (AL
0	0	105.173990	37.163008	179.142400	120.606253	14.706229	0.195095	104.533380	1.193389	0.919454	1.108535	
1	1	94.074211	37.098196	381.340534	140.365630	21.479079	0.188954	98.293658	4.116356	0.929007	1.088096	
2	2	84.808210	37.096441	111.705621	106.118779	13.996107	0.230852	104.895903	1.264473	1.124339	1.156892	