Patient Survival Prediction

Guided By: Dr. Ben Kim

Authors
Sugandha Baruah (sbaruah@seattleu.edu),
Pratiksha Chhajed (pchhajed@seattleu.edu),
Niki Gandhi (ngandhi@seattleu.edu),
Anusha Kumar (amakamdileepkumar@seattleu.edu)

BUAN 5510 01: Capstone (Business Analytics)
December 6, 2022

Table of Contents

Abstract

To predict the in-patient survival rate, this paper examines the U.S. mortality data collected from the Kaggle data source and has conducted data analysis through the use of different predictive models to address the in-patient mortality problem. As a first step, we have performed a literature review on mortality and/or survival among patients admitted due to various diseases. Literature review and exploratory data analysis including the data types, correlation analysis, dimensionality reduction, and data balancing led us to use Gradient Boosting and Neural Network algorithms for our final prediction.

As one of the important steps in data preprocessing, we performed dimensionality reduction technique feature selection in addition to primary component analysis. This helped us to determine the relation with influencing predictor variables based on their importance. As a next important step, we evaluate the performance of the models we built to determine which of the six models yielded the most accurate results in terms of accuracy, precision, and recall to add value to our research question. We validated the performance of these models using classification metrics and concluded that the Gradient Boosting algorithm was more accurate for predicting the in-patients' survival. Whereas, the Neural Network algorithm performed best in terms of recall by capturing approx. 95% of samples in the data related to patients' death. This research can prove to be an aid for non-technical stakeholders in the Healthcare domain including clinicians, healthcare professionals, health insurance companies, and pharmaceutical companies for better patient care and hospital management. These models can be improved by including geographic information like city names, and accurate and complete health data (no missing values). Furthermore, analyzing mortality data for a longer period would not only enhance the predictive powers of the models but could also reveal insights and patterns that could be used to recognize the factors and perhaps for better patient care.

Introduction

Background

Healthcare plays a crucial role in human life. Especially after COVID, the healthcare system has taken a toll in the US. As of today's record, US national healthcare expenditure reached $4.1 trillion in 2020 which was 19.7 percent of GDP and is estimated to reach $6.2 trillion by 2028. Analytics can enhance healthcare by improving patient outcomes and providing relief to impact people’s lives positively. Implementation of the latest technologies generates terabytes of data related to patients and hospitals through lab results, inpatient monitoring systems, and examination reports in the form of real-time structured and unstructured data. Analysis of this Big data collection enables doctors to draw samples to identify the warning signs of a serious illness before it arises to save a patient’s life. Therefore, we aim to develop a supervised learning model for survival prediction among in-patients to raise awareness to determine the treatment accuracy and relation with influencing predictor variables.

Problem Statement

The predictors of in-hospital mortality for admitted patients remain poorly characterized. Knowledge about chronic conditions can inform clinical decisions about patient care and improve patient survival outcomes. Thus, our problem statement is to **'Develop and validate a predictive model for all-cause in-hospital mortality among admitted patients and to detect and visualize significant indicators of mortality rate among patients.'**

Contributions

Traditional Methodologies vs Robust Machine Learning Models:

In previous studies, clinicians have used basic software programs, such as Excel, SPSS, STATA, and other traditional models for predicting survival rates among patients for different diseases. Some of these conventional statistical methods are not adaptable to identifying new variables and generating creative and integrative visualizations for the analysis of predictor variables and understanding key features. Hence, the goal here is to enhance our study around the research question by implementing robust machine learning models to predict and provide accurate results critical for decision-making activities for the hospital staff.

Limited Dataset vs Diverse Large Data Sources:

Apart from traditional methodologies, in the past, numerous kinds of research have been conducted around patient survival but they were restricted to patient cohorts with the same ailment. In our study, we collected complete health and demographic data for in-patients admitted due to different illnesses. We are contributing by building ML models that can help guide hospitals to ascertain patient survival irrespective of their illness based on predictive variables.

Literature Reviews

Early Detection of In-Patient Deterioration: One Prediction Model Does Not Fit All (Jacob N. Blackwell, 2020)[2]

Catastrophic illnesses can occur based on numerous factors and often cannot be determined by a single predictor attribute. In this study, the agenda is to verify the diversity of reasons which lead to clinical deterioration in patients which could also possibly lead to ICU transfers, and are determined using the predictive modeling approach. The dataset consists of 8111 adult patients, 457 of whom were transferred to an ICU for clinical deterioration. The study tests three methods to predict analytics monitoring. The first approach was to represent the class of untrained models with proper thresholds. The second approach was to use measured vital sign values, laboratory results, and continuous cardiorespiratory monitoring to train a universal prediction model on all ICU-admitted patients. The third approach was for patients who had a specific set of reasons for transferring to the ICU identified by clinician review.

The analysis showcased that having a single predictive model for clinical deterioration does not ensure correct predictions because every illness has its own specific set of symptoms and reasons. Thereby, multiple models must be trained for each clinical illness and its' predictions.

Improving the Prediction of Heart Failure Patients' Survival Using SMOTE and Effective Data Mining Techniques (Ishaq, et al., 2021)[3]

The research focuses on designing an effective decision support system that accurately diagnoses the survival of patients with cardiac failure. The main objective was to use machine learning-based expert systems which effectively diagnose a cardiovascular disease that lowers the fatality rate. The dataset consists of medical data for 299 patients, previously affected by left ventricular systolic dysfunction. Nine ML models were employed on reduced feature data: Tree-based ensemble models, tree-based boosting models, regression models, and statistical-based models. SMOTE technique was applied to handle the class-imbalance problems.

According to the experimental findings, supervised machine learning models efficiently predict the survival of patients with cardiovascular failure. SMOTE technique significantly improved the performance of tree-based classifiers in the unbalanced datasets to predict heart patient survival.

Machine learning-based early warning system enables accurate mortality risk prediction for COVID-19 (Yue Gao, 2020) [4]

This article was an observational cohort analysis of the clinical data for COVID-19 patients. The aim was to develop machine learning models to predict the mortality risk and stratify the patients accordingly at the time of admission. The research enabled the prognosis of physiological deterioration and death of admitted patients up to 20 days in advance.

The dataset consisted of 2520 consecutive COVID-19 patients from two affiliated hospitals between January 27, 2020, and March 21, 2020. Post-processing and feature selection, data were used for models. This article focuses on the use of an ensemble model approach fitted with four machine learning algorithms with tenfold cross-validation by fine-tuning the model parameters. The predictive performance of the models was evaluated by calibration curve, and evaluation metrics included area under the ROC curve (AUC), accuracy, sensitivity, and F1 score.

Early detection of type 2 diabetes mellitus using machine learning-based prediction models (Leon Kopitar, 2020) [5]

The goal of this study was to determine whether the early prediction of impaired fasting glucose and fasting plasma glucose level values were improved by new-age machine learning-based approaches over standard regression techniques.

The dataset included an Electronic Health Record collection of 27,050 adult patients without a history of type 2 diabetes who were enrolled between December 2014 and September 2017. After pre-processing, three different families of prediction models were used for prediction: boosting, bagging, and linear regression. Predictive models were validated using the following performance metrics: root means square error (RMSE) for prediction of the numerical value of FPG level and AUC for prediction of unbalanced discrete outcome for imbalanced datasets. The results of the study show that the XGBoost model had the best overall RMSE across all models. Glmnet beat all compared approaches on datasets in terms of the area under curve (AUC) measure.

Predicting mortality among patients with liver cirrhosis in electronic health records with machine learning (Aixia Guo, 2021) [6]

The current method for predicting mortality in sick patients relies on the Model for End-Stage Sodium (MELD-Na) score. Unfortunately, the MELD-Na score is not as predictive at lower scores and longer periods. In this study, Deep learning, and machine learning algorithms were employed to study the associations between baseline features such as laboratory measurements and diagnoses for each time window by a 5-fold cross-validation method. In all cases, these models consistently outperformed the MELD-Na model. Among the linear regression, random forest, and deep learning machine learning models, the Deep learning model had the best performance.

The analysis concludes that Deep learning models can be used to predict longer-term mortality among patients with liver cirrhosis more reliably than the MELD-Na variables alone. Future work should validate this methodology by incorporating the competing risk of a liver transplant.

Machine Learning Algorithms for Predicting the Recurrence of Stage IV Colorectal Cancer After Tumor Resection (Yuan Xu, 2020) [7]

In this study, four basic ML algorithms: logistic regression, decision tree, gradient boosting, and LightGBM were used for predicting the survival of stage 4 colorectal cancer patients. The four machine learning algorithms can each predict the risk of tumor recurrence in patients with stage IV colorectal cancer after surgery. Among them, Gradient Boosting and GBM performed best. Moreover, the Gradient Boosting weight matrix shows that the five most influential variables accounting for postoperative tumor recurrence are chemotherapy, age, LogCEA, CEA, and anesthesia time.

Hence, the paper concluded that Gradient Boosting and GBM are more likely to improve the accuracy of predicting the postoperative cancer progression of patients with stage IV colorectal cancer. Additional multicenter clinical studies are needed in the future.

Dataset Description

This dataset is sourced from Kaggle[1] which is around 31 MB in file size. Three tables with a total of 89 attributes and 91,713 rows. Collectively, the dataset consists of several factors involved in patient demography, hospitalization, and intensive care unit treatment. Based on these features we predict the target variable ‘Hospital_death’ i.e., whether the patient will survive or not using supervised learning models. We have identified each variable into a nominal, ordinal, and numeric category for in-depth analysis of the dataset.

The first table consists of diagnosis data related to in-patients. It has 72 attributes in total and is the largest table of all. After finding unique value counts for all attributes, we observed that ‘Patient_Id’ is the unique primary key for this table. The table has 70 numeric, 2 object data-type attributes. Further, we observed the ‘Unnamed:70’ column with all NaN values needs to be removed. The second table in our dataset consists of ICU (intensive care unit) information about the patients. It states the details about the ICU type, ICU admits source, whether the patient underwent surgery or not, etc. It has a total of 8 attributes. After in-depth analysis, we found that patient_encounter_id and hospital_id are composite primary keys. Lastly, the third table consists of patient demographic data with a total of 9 features. It states the patient’s age, weight, height, ethnicity, gender, etc. It has patient_id and encounter_id as unique row identifiers.

For the final dataset, we merged all three tables using primary key and foreign key relation. We merged the patient demographic table with the ICU table using encounter_id and hospital_id. We then merged this new table with the patient diagnosis dataset using the patient_id column. Finally, we have our final dataset ready for exploratory analysis with 91713 rows and 86 attributes.

Exploratory Data Analysis

The first step after merging the datasets was to explore the data in depth. This involved retaining the attributes that add value to the analytics and getting rid of all the columns irrelevant to our analysis. Hence, we dropped all the IDs such as 'hospital_id', 'icu_id', 'patient_id', and 'encounter_id' which are of no value in our predictive modeling. We also dropped 'Unnamed: 70' consisting of only null values. We also discarded 'apache_post_operative_y' which was a duplicate column present in two datasets.

Now, we determined the number of unique values/ categories per attribute to get an overall understanding. This was particularly important for categorical variables. 'Ethnicity' was seen to have the highest number of six different categories, which adds scope for future analysis. 18.82% of the variables were nominal and included only 2 categories. For instance – Gender has only two distinct categories - Male or Female as per the data source. Most attributes describing the patient's medical condition were represented as nominal with two categories indicating if the patient has a specific medical condition (1) or not (0). E.g., in the case of diabetes_mellitus, '0' indicates the absence of diabetes_mellitus for the patient and '1' indicates otherwise.

As a next step was to verify all the attributes had the right data types. We observed that many attributes which had '*float64*' as the datatype were nominal and had to be changed to '*object*' type. To ensure the appropriate datatypes for all the variables, we correctly changed 17 variables to the '*object'* type. We then checked for the count of non-nulls per attribute and observed that 'h1_mbp_noninvasive_max' and 'h1_mbp_noninvasive_min' have the highest missing values of approx 9.9%.

Our next objective was to gain an understanding of the nature of the distribution of the values for the numeric attributes. Hence, from Figure 1, we examined the histogram plots and noticed that many numerical attributes such as 'd1_mbp_min', 'd1_mbp_non-invasive_min', 'h1_sysbp_min', etc. were normally distributed around the mean. Other attributes such as 'age', 'd1_sp02_min', and 'd1_temp_min' were left-skewed. While attributes such as 'BMI', 'd1_diaspb_noninvasive_max', and 'h1_heartbeat_min' were all right-skewed. We observed that most numeric variables are skewed. Though tree-based classification models are non-parametric methods that do not require the data set to follow a normal distribution, we will normalize the data for other deep-learning predictive models in our study.

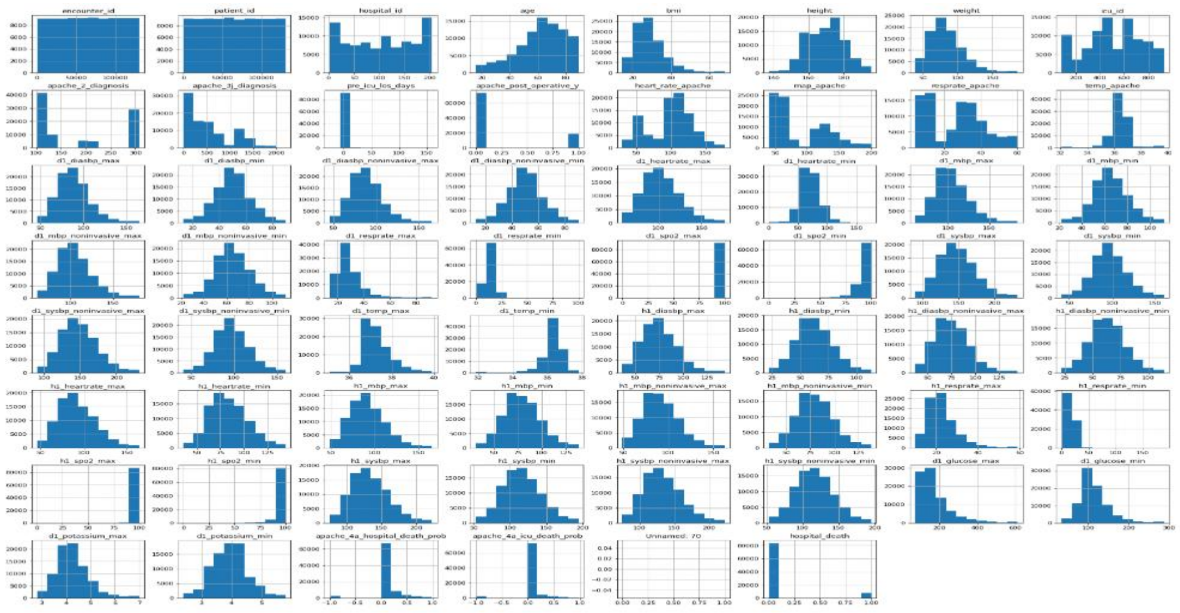


Figure 1: Frequency Distribution for Numeric Attributes

Further, we plotted bar graphs to showcase categorical variables and their respective data range distribution. For instance, from Figure 2, we can observe that the distribution of 'ethnicity' for the patients in the data is uneven. The highest number (~77%) of admitted patients belong to the 'Caucasian' ethnic group.

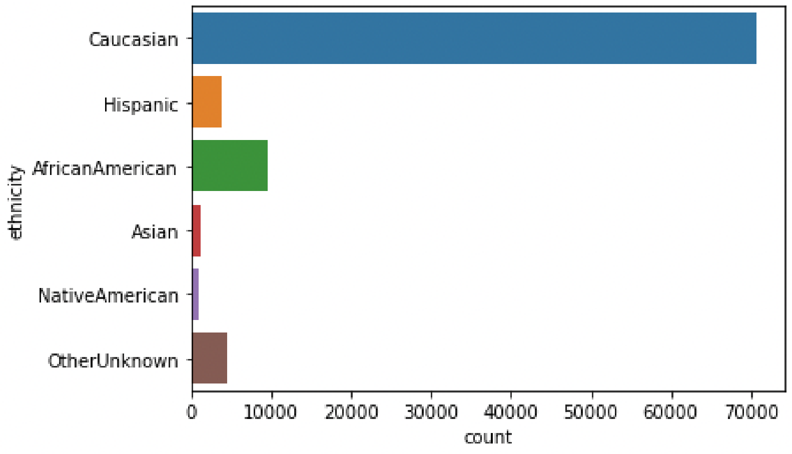


Figure 2: Distribution of Patients based on Ethnicity

Based on further analysis of the data, we realized that maximum Intensive Care Unit admits (~60% admits) happen because of Accidents and Emergencies. Another interesting insight from the data was that the in-patients were most affected by 'Cardiovascular' diseases than any other categories of illnesses, as seen in Figure 3. Cardiovascular cases account for approximately 41% of admits, whereas the next prominent reason 'Neurological' accounts for only around 14% of the admits.

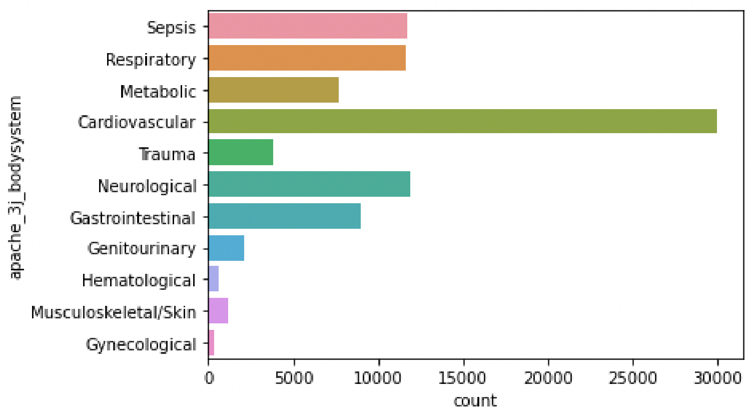


Figure 3: Distribution of Patients based on Categories of diseases

We also observed that the patient data was approximately equally distributed for 'Gender'. Another interesting observation was that the data relating to medical conditions such as 'lymphoma', 'aids', 'solid_tumor_with_metastasis', 'leukemia', etc. were all highly skewed and indicated the absence of these diseases in most patients.

Finally, we analyzed the target variable 'hospital_death'. According to the dataset, 91.37 % of the patients survived and only 8.63% of the patients died. This indicates the high skewness in data leading to an imbalanced dataset as indicated in Figure 4.

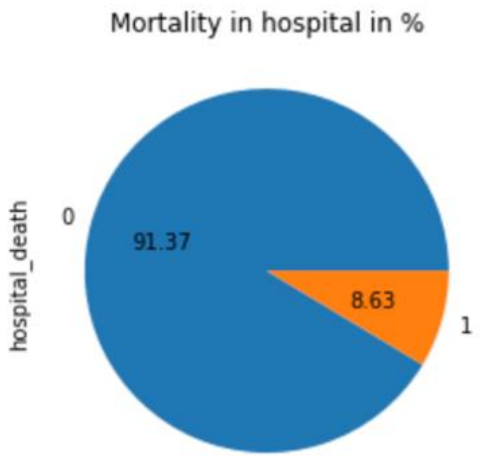


Figure 4: Mortality rate among in-patients

We further used aggregation functions to determine and analyze the mortality rate by various factors. An interesting insight was that the death rate was slightly higher for female in-patients (~8.8%) compared to male in-patients (~8.3%). The age of the admitted patients varied between 16 years to 89 years. While the average age of admitted patients was seen to be around 62 which is intuitive. Further, we plotted Probability Distribution based on Age and Gender (Figure 5) to understand the average in-hospital mortality of patients. The death probability for patients with age between 20 years to 60 years ranges between 0.02 and 0.1. It was interesting to notice that for patients above the age of 60, the average mortality rate peaks at 0.16.

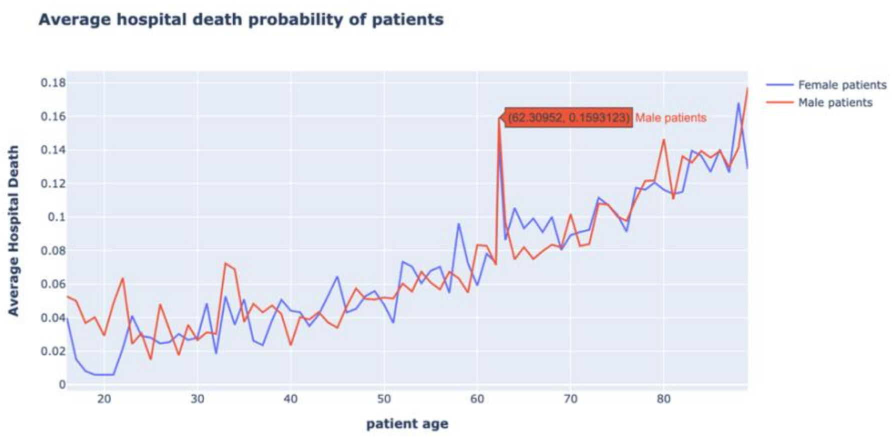


Figure 5: Average Hospital Death probability of patients

A few interesting observations were around patient ailment and health data. The Body Mass Index for in-patients ranges between 14.8 to 67.8. The average BMI for male admits is 28.7, while BMI for female in-patients averages 29.6. Both fall beyond the normal BMI range and indicate the existence of some illness. Lastly when observed from an Ethnicity perspective, Hispanic have the highest death rate (~9.9). We also observed that 'Diabetes_milletus' has the highest number of deaths compared to other medical illnesses/diseases (~1.7%). The death among in-patients with other medical conditions is lesser than 1%.

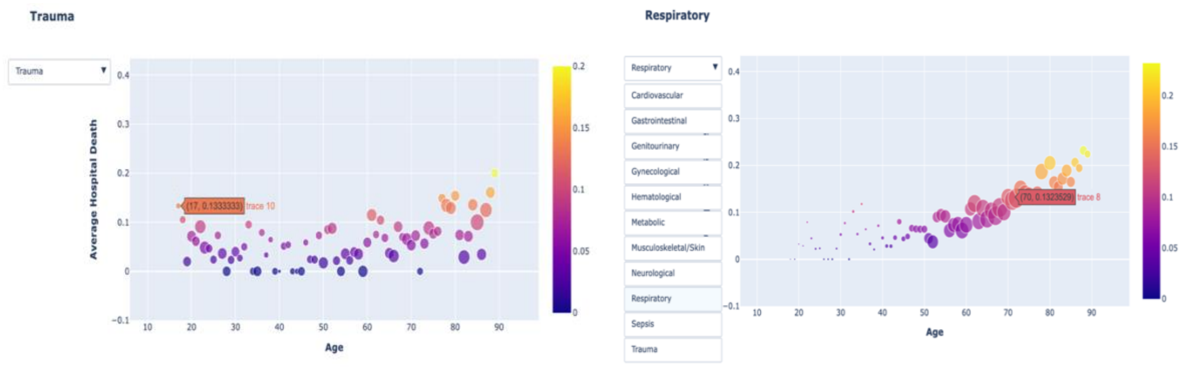


Figure 6: Scatter Plot - Average Hospital Death vs Age based on Disease/Ailments

The above graphs in Figure 6 visualize the distribution of the average in-hospital mortality across various age groups for a specific illness. It is interesting to see how young people between the age group of 20 to 50 years have a lesser average death probability. However, patients around 60 years of age suffering from respiratory diseases have an average in-hospital death probability of ~13%. It is seen to further increase to ~20% as the patient's age reaches 90 years. On comparing the above two charts in Figure 6, we can depict that the average in-hospital mortality due to Trauma is more in patients less than 50 years compared to the average in-hospital mortality due to Respiratory problems seen often in patients between the age groups of 50-90 years.

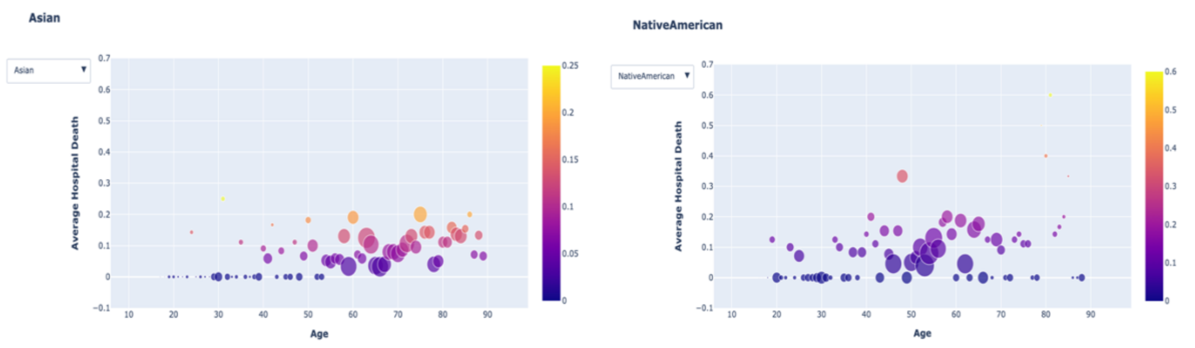


Figure 7: Scatter Plot - Average Hospital Death vs Age based on Ethnicity

Similarly, we have leveraged the aid of scatter plots for comparing average in-hospital mortality across multiple age groups and ethnicities. As seen in Figure 7, the average in-hospital mortality for Asian patients is prominently observed above the age of 50 whereas, in the case of Native Americans, the in-hospital mortality seems to be distributed across all age groups with a pick between the age of 50 to 60 years.

Data Pre-processing

Missing Values Treatment

Our dataset consisted of missing/null values. As we know, null value treatment is an important step in data pre-processing, and thus, we have calculated the percentage null values for each of the features. d1_potassium_min was seen to have the highest missing values ~10.45%. The dataset did not contain any features with missing data entries greater than ~10%. The features such as d1_potassium_max, h1_mbp_noninvasive_max, h1_mbp_noninvasive_min, and d1_potassium_min contained the highest percentage missing values. However, most features had ~99% non-null data. The target attribute, hospital_death had 0% null entries. We kept a 5% threshold and removed the features with more than 5% missing values.

The numerical features with less than 5% null values were treated using KNN Imputer. Since KNN imputer does not work for the categorical variables, the *fillna()* method was used to fill the nulls using mode values. We visualized the outliers using a box plot. To treat the outliers, a threshold of 99 percentile was set on the upper end and a 1 percentile threshold was set on the lower end. The outlier treatment resulted in a reduction of ~50% of data. So, we decided to drop the idea of outlier treatment for the dataset. Correlation Analysis was then performed to test the relationship between the quantitative variables.

Correlation Analysis

The highest correlation was discovered between features d1_diasbp_noninvasive_min, d1_sysbp_noninvasive_min, d1_mbp_noninvasive_min, d1_mbp_min, weight, h1_heartrate_max, d1_sysbp_max, d1_diasbp_max. Hence, we decided to keep a threshold of 80% and removed the features with multicollinearity greater than 0.80 from the dataset. This resolved the multicollinearity issues and the final dataset was ready for dimensionality reduction.

One-Hot Encoding

Before we could apply Principal Component Analysis (PCA) or feature selection, we created dummies for categorical variables. We then used them along with numerical variables to analyze and retrieve feature importances. We obtained 58 attributes out of which 34 attributes were numerical(int/float) and the remaining 24 attributes were categorical in nature. The end process resulted in a total of 136 attributes.

Dimensionality Reduction

1. Primary Component Analysis (PCA)

After performing data pre-processing, we decided to perform dimensionality reduction using Principal Component Analysis (PCA). We generated a Scree Plot using a cumulative explained variance. And located an elbow point to determine the number of primary components to use. As seen in below Figure 8, we visualized a Scree plot using PCA that talks about the explained variance in the dataset or how much variation in the dataset can be attributed to each of the principal components. When K (number of attributes) is greater than or equal to 20, we observed that the attributes did not have much variance and thus the K-components are not able to explain much of the variance in the dataset. Thus, to optimize the resources and to perform analytics more efficiently, we incorporated dimensionality reduction as this would still provide the same or slightly better performance using a lesser number of attributes.

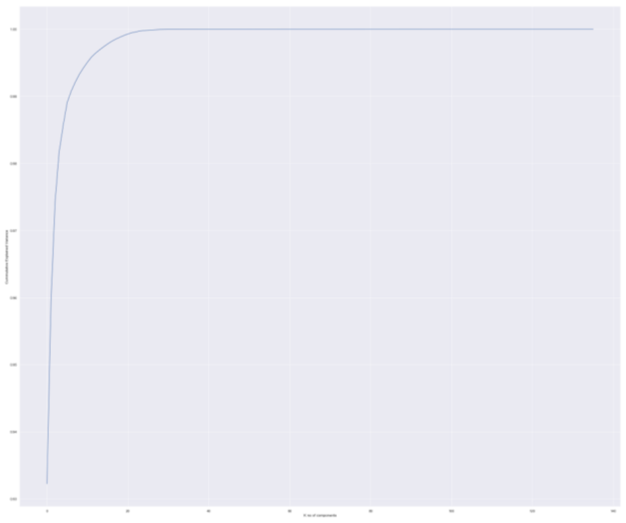


Figure 8: Scree Plot

2. Feature Selection

We further leveraged feature selection to choose the important features to be included in the reduced dataset based on their importance. We utilized the random forest classifier estimator, using mean (0.00735) as a minimum threshold for this purpose. The algorithm selected 39 features which are ordered as per their respective importance in below Figure 9.

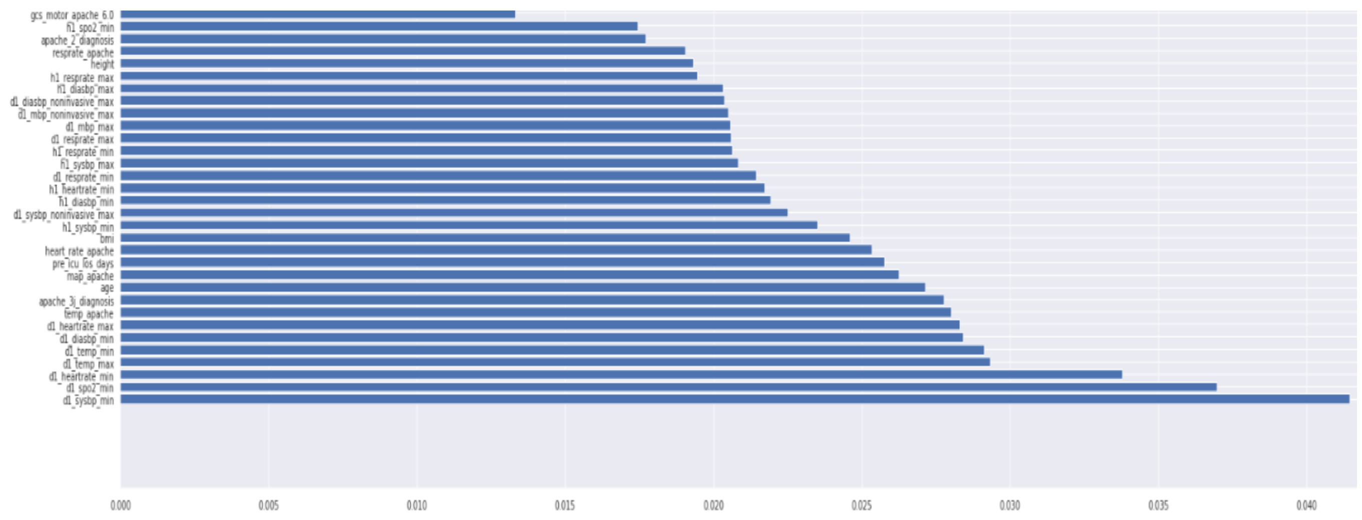


Figure 9: Feature Selection

Post feature selection, we performed a random classifier to see whether the accuracy result has improved. A slight improvement has been seen in all the matrices in the reduced data.

Data Balancing

1. Naive Random Over-Sampling

As the last step in data preprocessing, we balanced the imbalanced dataset as this might skew the class distribution. If the majority class is causing bias in the training dataset, it can then influence machine learning algorithms, leading some to ignore the minority class entirely. Hence, we addressed the problem of class imbalance by randomly oversampling the training dataset. We first did random sampling on the original 136 features, split the data into the train (70%) and test dataset (30%) and fit them in the Random Forest Classifier to determine the accuracy, precision, and recall for the attribute where death = 1.

Next, we did random sampling on the reduced features dataset. We then split the dataset into the train (70%) and test (30%) datasets and fit the random forest classifier to determine the accuracy, precision, and recall. We observed that the predictions have improved the most for the reduced dataset after the random over-sampling technique.

2. Synthetic Minority Oversampling Technique (SMOTE)

Another approach to address the imbalance dataset was by creating new artificial training examples based on the original training examples. It causes an increase in the variety of training examples that can at times be preferred over random sampling because random oversampling just increases the size of the training data set through repetition of the original examples. As we use SMOTE on the original features and the reduced features dataset, we see that SMOTE gives better results on the reduced features dataset.

Based on the illustrative results of the random forest classifier, we concluded that SMOTE is more effective than Random Over Sampling for both the original and reduced dataset. And we decided to utilize the balanced dataset generated using SMOTE.

Data Mining Models and Evaluations

As discussed earlier, we performed SMOTE on the original and reduced dataset and applied six different algorithms on four different datasets - original unbalanced, original balanced, reduced unbalanced, and reduced balanced data. Before applying algorithms, we randomly split these datasets into 70% training and 30% test sample to validate the performance of these algorithms. Merely training the raw models on datasets cannot ensure better performance. Thus, hyperparameter tuning plays a vital role in setting a tune before running a training job to control the behavior of machine learning algorithms.

Hyperparameter tuning

Model parameters are learned as part of the training process. The values fed in the hyperparameters in different algorithms act as specific instructions before running the training job. We selected different hyperparameters in six different classification algorithms – Random Forest Classifier, XG Boost, Gradient Boosting, Neural Network, Decision Tree, and K-Nearest Neighbours (KNN). We then trained models using tuned parameters and applied them to the test data. We performed two approaches of hyperparameter tuning-

1. Grid Search CV
2. Randomized Seach CV

While we performed both approaches, we found that Grid Search CV performs an extensive sweep on all possible combinations that could be inefficient in training time and from a cost perspective. On the other hand, Randomized Search CV executes a random hyperparameter combination that accelerates the training time of each model. After applying these approaches, we found the optimal parameter values for all the above algorithms resolving the underfitting or overfitting issues. We then applied these tuned models to the respective datasets to evaluate the final performance.

Data Modelling

After obtaining the optimal parameters and fitting them into each model, we observed Random Forest Classifier, XG Boost, Gradient Boosting, and Neural Network performed well among all six predictive models for specific performance metrics. For detailed information, please refer to *Table 1* on model performance evaluations.

Out of the four best-performed classification models, we obtained the best parameters for the XG Boost classifier with a *learning rate* of 0.1 which represents the speed at which the model learns. With a *maximum depth* of 5, it suggests that this tree algorithm can train and explicitly explain each node at the depth of 5, capturing the influential pattern that might lead to an increase in the test error rate. *N-estimator* as 400 indicates the number of trees inside the classifier which increases the model accuracy before the drop in accuracy. Finally, with *optimal booster* as gbtree, we achieved ~94% accuracy and precision ranging between 69% and 74% for all four datasets. Secondly, we applied a tuned Random Forest Classifier to all four datasets. Since the optimal *max depth* for this model was 15, the dataset might be explaining each node and split, which resulted in a slight decrease in accuracy. Also, it is noteworthy that Random Forest Classifier Precision is comparatively lower for balanced datasets (50% to 66%) than for unbalanced datasets (~78%). Next, the tuned Gradient Boosting Classifier provided better accuracy for original and reduced balanced datasets at 97% but dropped to 93% for the unbalanced datasets. We observed a similar trend for precision as well, where it dropped from 85% for balanced data to 80% in unbalanced data models.

Lastly, we analyzed results from Multi-Layer Perceptron (Neural Network) Classifier model. Here, we tuned the *hidden layers* (90, 80, 40) and *activation* parameters. Activation is used to introduce nonlinearity to the model, which allows the deep learning model to learn nonlinear prediction boundaries and relu turned out to be best. As a result, accuracy variedly ranged from 17% to 90% making it the lowest-performing model for this metric among the earlier three models. However, accuracy and precision are not always the ultimate performance metrics. Our dataset deals with improving the diagnosis and better prediction of patient survival. In that case, recall is a superior measure to determine what proportion of patient mortality cases were captured correctly. In the context of diagnostics and medicine, it is important to improve recall for the streamlined course of treatment as misclassification of patient survival can have serious consequences. Hence, Neural Network plays a crucial role even though it has low accuracy and precision. We observed that Neural Network has the highest recall of 99% in the original balanced data set and 95% in the reduced balanced dataset compared to 17% recall for Decision Tree, 82% for Gradient Boosting, and 43% in the case of XG Boost Classifier. Finally, we validated all our classification model results using K-fold cross-validation.

Model Performance Metrics for Reduced Balanced Dataset			
Models	**Accuracy**	**Precision**	**Recall**
Random Forest Classifier	91%	50%	82%
XG Boost Classifier	93%	69%	43%
Gradient Boosting Classifier	97%	81%	82%
Neural Network	68%	20%	95%

Table 1: Performance Metrics

Further, we explored an unsupervised learning algorithm K-means clustering for the original and reduced balanced dataset to evaluate the validity of the dataset for clustering. We tried to determine the optimal number of clusters by using the elbow method on the inertia values and observed no points or peaks where an elbow could be formed (Figure 10). Hence we concluded that clustering was not a good approach for our dataset.

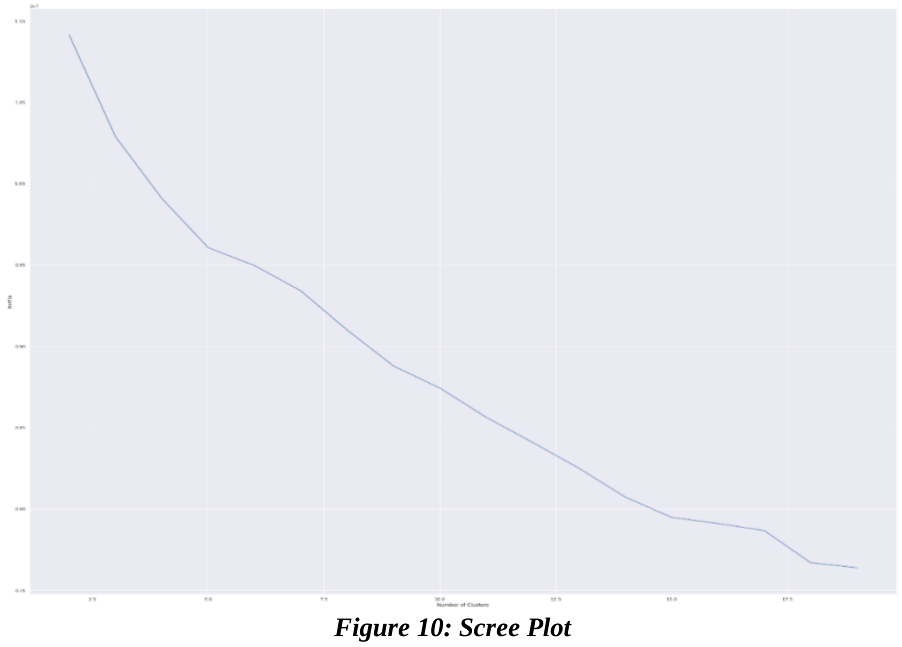


Figure 10: Scree Plot

Discussion

Methodological Contributions

We have developed prediction models with more apparent case mix differences that would outperform the sample and the validation experiments. Our research will generate better results across diverse target populations and, eventually, be more useful in routine treatment rather than treating patients with specific illnesses.

We used a variety of different machine learning models for evaluation to reduce the possible issues caused by technique variance. We also used Feature Selection to list the key influential attributes of patient mortality. Additionally, we leveraged Synthetic Minority Oversampling Technique (SMOTE) to balance the highly skewed data for obtaining an accurate model prediction. Gradient Boosting Classifier and XGBoost models provided the best results than the Decision Tree model alone because Gradient Boosting and XGBoost models are made up of several decision trees with some self-regulation to prevent the overfitting of the data.

For better comparison and recommendations, we executed our models on different operating systems including Windows, Mac OS, and Cloud Systems like Google Collab. The step of hyperparameter tuning took the maximum runtime in all the systems, and we had to minimize some test parameters while using the Grid Search CV approach because of the time constraints. But it was crucial since it helped us optimize the algorithms we were using. Finally, it was noteworthy to see the model runtime reduction for Mac OS and cloud systems saving cost and time for companies.

Reduced Balanced Model Runtimes for Various Systems (in Seconds)			
Models	**Windows** RAM: 12GB Chip: Intel Core i7 Storage: 512GB SSD	**Mac Book Pro** RAM: 8 GB Chip: M1 Storage: 512GB SSD	**Google Collab** RAM: 12 GB Chip: Intel Xeon Storage: 25 GB
Random Forest Classifier	65.58	43.74	89.85
XG Boost Classifier	62.13	Not Supported	151.26
Gradient Boosting Classifier	1010.58	1106.99	1106.99
Neural Network	241.08	91.63	232.87
Decision Tree Classifier	3.93	1.13	4.57
K-Nearest Neighbour	89.57	62.02	72.02

Table 2: System Runtime

Technical Stakeholders

Inpatient admitting teams often care for patients for multiple days, and our clinical partners and operational stakeholders wanted to improve the rate at which goals of care conversations occur shortly after admission. Moreover, this is a commonly used point to predict mortality endpoints. Hence it is imperative to understand and assess the efficacy of the classification models built to implement the objective of aptly predicting patient survival among in-patients (Gad, 2020). The accuracy of a machine learning classification algorithm is one way to assess how often the model classifies a data point correctly by considering all the true cases. As we see in our classification models, after tuning the hyperparameter using Grid Search CV for the balanced reduced dataset, the Gradient Boosting model gave the highest accuracy of 97% with a runtime of 878 seconds. The implication of such a result interprets that ~97% of the mortality and survival cases are correctly classified.

Accuracy, although a great metric, is very limited in its scope and can be deceiving. For the healthcare domain focusing on improving false positive scores is more important than improving false negative scores. Hence, we must calculate the precision and recall. However, there is always a trade-off between precision and recall. The precision determines what proportion of mortality cases were predicted correctly. From our predictive model results, it is observed that Gradient Boosting also performs the best in terms of a precision rate at 81%. In other words, the model can classify mortality correctly 81% of the time. Whereas Recall[8] describes the sensitivity of the data. It cares only about how the positive samples are classified. When the model classifies all the positive (morality) samples as Positive, then the recall will be 100%. As discussed earlier, though Neural Network has the lowest accuracy and precision, with a recall rate of 95% it performs best among all the models. This is an important metric for our research objective because the healthcare unit can predict mortality among patients 95% of the time, correctly.

Considering these important differences, hospitals wishing to implement these models may need to set goals that best align with performance. Overall, taking into account differences in in-hospital mortality rate performance metrics among accuracy, precision, and recall between the four models, these results suggest that the model needs to be chosen based on the ultimate goal.

Domain Knowledge

Non-technical Stakeholders

In addition to performing well on technical grounds, machine learning models must also change behavior to improve patient care. Behavioral change necessitates bettering choices, which are often made by physicians and hospital management. During our efforts to develop a model we also kept our non-tech stakeholders in mind. We used a framework to identify the pain points of our stakeholders and address them through our research.

Successful applications of machine learning in healthcare often necessitate significant integration efforts. Integrating our model's output into intricate human workflows can translate modeling advancements into improvements in clinical treatment. The predictive analytics model presented by us will play a crucial role in advancing care and enhancing results in the healthcare industry by helping physicians to analyze the vast amount of patient health and demographic data effectively. Further, clinicians frequently experience prognostic uncertainty and may find it challenging to provide families with an informed assessment of the expected outcomes of treatment decisions. Clinicians, patients, and families might all be vulnerable to unconscious but significant cognitive biases when making judgments under pressure, further adding to the complexity of these situations (Mark P. Sendak, 2020). With the aid of our prediction models, these difficulties will now be easier to overcome and support medical decision-making for healthcare professionals.

Additionally, accurate survival prediction is highly valued in the clinical practice of end-of-life care. Our prediction models enable better communication and preparation for impending death, helping old age people avoid futile medical treatment, and facilitating optimal palliative care quality for patients, families, and physicians altogether. Our exploratory data analysis leverages the feature importance technique to select the prime attributes influencing patient mortality.

This will enable hospitals in gathering crucial medical and demographic details from patients to make more informed decisions about treatment based on mortality prediction using the above-collected data. Further, these key features from our analysis will help researchers in the pharmaceutical industry to utilize this data for important drug developments for diseases having a significant impact on patient death. Ultimately, this will help improve in-hospital survival among patients suffering from those illnesses.

Our model will also come as an aid for hospital management who can integrate this model into the system for better in-hospital patient care and resource management for themselves and their patients. Our predictive models will prove to be a great resort for hospital management at all levels to use available data as an asset to make investment decisions in the health technology sector for ICU beds and patient infrastructure (Nathan Brajer, 2020). Lastly, the above patient survival prediction models can also be utilized by Health Insurance corporations in evaluating prospective customers' survival or mortality rate during the decision-making process for insurance plan offers and respective premium charges.

Conclusion

Summary

In conclusion, we would like to successfully present a machine learning-based solution that predicts the survival status of admitted patients by accounting for a wide spectrum of diseases/ailments and patient demography. Based on the above results and analysis, we recommend adopting the model most aligned with the hospital's needs. If our aim remains focused on improving accuracy and precision, then the Gradient Boosting model will get the priority. Whereas, if we are targeting the highest recall, then Neural Network is the most recommended model for the purpose.

Lastly, we recommend using a cloud desktop for cost-efficient results. With the increase in clinical data, it is expensive and tedious to maintain data on local offline systems. It is effortless and cost-efficient to utilize the available economical cloud computing infrastructure for faster processing when implementing predictive models.

Limitations

Our research is limited by a couple of factors and has a scope for improvement. One of our main constraints was that the dataset failed to include crucial information relating to the socio-demographics of in-hospital patients. Information such as spatial (State/Region of residence), marital status, lifestyle, education, job type, salaries, and working conditions will turn out to be an informative value addition. These factors could have helped analyze a commonality between patients affected by a similar illness/disease and suggest a cure to prevent the occurrence of these illnesses. Secondly, the expert models might not work best when certain extreme/odd cases are considered. For example, this study does not elucidate to what extent the model has learned treatment effects, and without careful instruction on how to interpret model output, clinicians may underestimate in-hospital mortality risk for patients with dangerous conditions that would usually receive intensive treatment (Nathan Brajer, Prospective and External Evaluation of a Machine Learning Model to Predict In-Hospital Mortality of Adults at Time of Admission, 2020).

Another pitfall that we encountered during the research was that the dataset was not suitable for clustering. Clustering results could have provided descriptive insights that would have proven to be beneficial in understanding the mortality among patients with similar medical conditions. Additionally, this study demonstrated that using only a limited, standardized data set in-hospital mortality can be predicted satisfactorily at the time point of hospital admission. More parameters describing patients' health are likely needed to improve our model for long-term care. Lastly, our model may not be suitable for infant mortality prediction due to the data availability limitation, where our dataset only consisted of patients between the age group of 16 to 90 years of age.[9]

Future Scope and Enhancements

We further plan on improving our work by diving deeper to analyze and predict the percentage survival rate for in-hospital patients based on the severity of the patient's condition during the time of admission. Also, based on the outcomes of the prediction, the hospitals can manage their resources, efficiently. We can enhance the result in the future by including more patient data with in-hospital deaths. Our data currently constitutes ~9% of patients who died after being admitted to the hospital. Thus we had to synthetically impute the samples for precise and accurate mortality prediction. If the model is trained on real-world mortality cases, we expect an improvement in the model's prediction performance.

Since the machine learning model is flexible and adaptable, we can add new attributes around the number of healthcare professionals per patient, hospital location, etc. for enhanced prediction of patient survival. This will prove to be valuable in analyzing healthcare professionals' demand for specific geographies where patient mortality is higher due to a lack of clinicians, doctors, and healthcare professionals. Additionally, with the advanced technology infrastructure, the in-patient mortality prediction model can have functionalities to run in a real-time environment. Finally, the benefit-to-cost ratio of developing and deploying models in clinical settings will continue to increase as commonly available data elements are more effectively used, and opportunities to scale our current models are identified.[10]

References

1. Dataset: Kaggle <https://www.kaggle.com/datasets/mitishaagarwal/patient>

2. Jacob N. Blackwell, MD,1 Jessica Keim-Malpass, Ph.D., RN,2,3 Matthew T. Clark, Ph.D.,4 Rebecca L. Kowalski, BS,1 Salim N. Najjar,1 Jamieson M. Bourque, MD,1,,2 Douglas E. Lake, Ph.D.,1,,3 and J. Randall Moorman, May 11., 2020. "Early Detection of In-Patient Deterioration: One Prediction Model Does Not Fit All"
< https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7259568/>

3. Abid Ishaq; Saima Sadiq; Muhammad Umer; Saleem Ullah; Seyedali Mirjalili; Vaibhav Rupapara, Michele Nappi, March 04., 2021., "Improving the Prediction of Heart Failure Patients' Survival Using SMOTE and Effective Data Mining Techniques"
< https://ieeexplore.ieee.org/abstract/document/9370099/authors#authors>

4. Yue Gao, Guang-Yao Cai, Wei Fang, Hua-Yi Li, Si-Yuan Wang, Lingxi Chen, Yang Yu, Dan Liu, Sen Xu, Oct 06., 2020, "Machine learning based early warning system enables accurate mortality risk prediction for COVID-19"
< https://www.nature.com/articles/s41467-020-18684-2#Sec20>

5. Leon Kopitar, Primoz Kocbek, Leona Cilar, Aziz Sheikh & Gregor Stiglic July 20., 2020., "Early detection of type 2 diabetes mellitus using machine learning-based prediction models"
< https://www.nature.com/articles/s41598-020-68771-z>

6. Aixia Guo, Nikhilesh R. Mazumder, Daniela P. Ladner, Randi E. Foraker, August 31, 2021, "Predicting mortality among patients with liver cirrhosis in electronic health records with machine learning"
< https://pubmed.ncbi.nlm.nih.gov/34464403/>

7. Yuan Xu, Lingsha Ju, Jianhua Tong, Cheng-Mao Zhou & Jian-Jun Yang, February 13, 2020, "Machine Learning Algorithms for Predicting the Recurrence of Stage IV Colorectal Cancer After Tumor Resection"
< https://www.nature.com/articles/s41598-020-59115-y#auth-Yucan-Xu >

8. Ahmed Fawzy Gad, December 1, 2020, "Evaluating Deep Learning Models: The Confusion Matrix, Accuracy, Precision, and Recall"
< https://blog.paperspace.com/deep-learning-metrics-precision-recall-accuracy/#:~:text=The%20recall%20is%20calculated%20as,the%20more%20positive%20samples%20detected>

9. Nathan Brajer, Brian Cozzi, Michael Gao, February 7, 2020, "Prospective and External Evaluation of a Machine Learning Model to Predict In-Hospital Mortality of Adults at Time of Admission"
<https://jamanetwork.com/journals/jamanetworkopen/fullarticle/2760438>

10. Mark P. Sendak, Michael Gao, Nathan Brajer, Suresh Balu, March 23, 2020, "Presenting machine learning model information to clinical end users with model facts labels"
< https://www.nature.com/articles/s41746-020-0253-3>

Appendices

<u>Code</u>

Importing all necessary libraries

In [ ]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

from pandas.plotting import hist_frame
from imblearn.over_sampling import SMOTE
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import GridSearchCV
from sklearn.preprocessing import StandardScaler
import time
from sklearn.decomposition import PCA
from sklearn.model_selection import train_test_split
from sklearn.feature_selection import SelectFromModel
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn import metrics
from imblearn.over_sampling import RandomOverSampler
from sklearn.model_selection import RandomizedSearchCV
import xgboost as xgb
from xgboost import XGBClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.cluster import Kmeans
from sklearn.impute import KNNImputer
from sklearn.neural_network import MLPClassifier

import plotly.express as px
import plotly.offline as py
import plotly.graph_objs as go
import plotly.tools as tls
from plotly.subplots import make_subplots
import plotly.figure_factory as ff

pd.set_option('display.max_rows', 500, 'display.max_columns', 100)

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

Out[ ]:
```
Mounted at /content/drive
```

Read all data files and general description

In [ ]:
```
diagnosis = pd.read_csv("/content/drive/MyDrive/Colab
Notebooks/patient_diagnosis.csv", sep= ',', header= 0)
```

Data Shape

In [ ]:
```
diagnosis.shape
```

Out[ ]:
```
(91713, 72)
```

Unique Value Count

In [ ]:
```
diagnosis.nunique()
```

Out[ ]:
```
patient_id                      91713
apache_2_diagnosis                 44
apache_3j_diagnosis               399
pre_icu_los_days                 9757
apache_post_operative               2
arf_apache                          2
gcs_eyes_apache                     4
gcs_motor_apache                    6
gcs_unable_apache                   2
gcs_verbal_apache                   5
heart_rate_apache                 149
intubated_apache                    2
map_apache                        161
resprate_apache                    74
temp_apache                       191
ventilated_apache                   2
d1_diasbp_max                     120
d1_diasbp_min                      78
d1_diasbp_noninvasive_max         120
d1_diasbp_noninvasive_min          78
d1_heartrate_max                  120
d1_heartrate_min                  154
d1_mbp_max                        125
d1_mbp_min                         91
d1_mbp_noninvasive_max            122
d1_mbp_noninvasive_min             91
d1_resprate_max                    79
d1_resprate_min                    55
d1_spo2_max                        43
d1_spo2_min                       101
```

```
d1_sysbp_max                      143
d1_sysbp_min                      120
d1_sysbp_noninvasive_max          143
d1_sysbp_noninvasive_min          120
d1_temp_max                       186
d1_temp_min                       209
h1_diasbp_max                     107
h1_diasbp_min                      92
h1_diasbp_noninvasive_max         108
h1_diasbp_noninvasive_min          93
h1_heartrate_max                  119
h1_heartrate_min                  109
h1_mbp_max                        117
h1_mbp_min                        107
h1_mbp_noninvasive_max            115
h1_mbp_noninvasive_min            107
h1_resprate_max                    50
h1_resprate_min                    91
h1_spo2_max                        72
h1_spo2_min                       100
h1_sysbp_max                      149
h1_sysbp_min                      142
h1_sysbp_noninvasive_max          149
h1_sysbp_noninvasive_min          143
d1_glucose_max                    538
d1_glucose_min                    256
d1_potassium_max                  100
d1_potassium_min                  116
apache_4a_hospital_death_prob     101
apache_4a_icu_death_prob           99
aids                                2
cirrhosis                           2
diabetes_mellitus                   2
hepatic_failure                     2
immunosuppression                   2
leukemia                            2
lymphoma                            2
solid_tumor_with_metastasis         2
apache_3j_bodysystem               11
apache_2_bodysystem                10
Unnamed: 70                         0
hospital_death                      2
dtype: int64
```

Intenise Care Unit Data

In [ ]:

```
icu = pd.read_csv("/content/drive/MyDrive/Colab 
Notebooks/patient_icu_data.csv", sep= ",", header=0)
```

Data Shape

In [ ]:
```
icu.shape
```
Out[ ]:
```
(91713, 8)
```

Unique Attributes

In [ ]:
```
icu.nunique()
```

Out[ ]:
```
encounter_id             91713
hospital_id                147
elective_surgery             2
icu_admit_source             5
icu_id                     241
icu_stay_type                3
icu_type                     8
apache_post_operative        2
dtype: int64
```

Patient Demographic Data

In [ ]:
```
patient = pd.read_csv("/content/drive/MyDrive/Colab
Notebooks/patient_info.csv", sep= ",", header= 0)
```

Data Shape

In [ ]:
```
patient.shape
```

Out[ ]:
```
(91713, 9)
```

Unique Values

In [ ]:
```
patient.nunique()
```

Out[ ]:
```
encounter_id    91713
patient_id      91713
hospital_id       147
age                74
bmi             34888
ethnicity           6
gender              2
```

```
height            401
weight           3409
dtype: int64
```

Merging all three datasets based on the primary and foreign key (common column) for Exploratory Data Analysis

In [ ]:
```
patient_data = pd.merge(patient, icu, on = ["encounter_id", "hospital_id"])
```

Data Shape

In [ ]:
```
patient_data.shape
```

Out[ ]:
```
(91713, 15)
```

Merge Third Dataset

In [ ]:
```
data = pd.merge(patient_data, diagnosis, on = ["patient_id"])
```

Data Description

Final Data Shape

In [ ]:
```
data.shape
```

Out[ ]:
```
(91713, 86)
```

Unique Value Counts

In [ ]:
```
data.nunique()
```

Out[ ]:
```
encounter_id                     91713
patient_id                       91713
hospital_id                        147
age                                 74
bmi                              34888
ethnicity                            6
gender                               2
height                             401
weight                            3409
elective_surgery                     2
icu_admit_source                     5
```

```
icu_id                             241
icu_stay_type                        3
icu_type                             8
apache_post_operative_x              2
apache_2_diagnosis                  44
apache_3j_diagnosis                399
pre_icu_los_days                  9757
apache_post_operative_y              2
arf_apache                           2
gcs_eyes_apache                      4
gcs_motor_apache                     6
gcs_unable_apache                    2
gcs_verbal_apache                    5
heart_rate_apache                  149
intubated_apache                     2
map_apache                         161
resprate_apache                     74
temp_apache                        191
ventilated_apache                    2
d1_diasbp_max                      120
d1_diasbp_min                       78
d1_diasbp_noninvasive_max          120
d1_diasbp_noninvasive_min           78
d1_heartrate_max                   120
d1_heartrate_min                   154
d1_mbp_max                         125
d1_mbp_min                          91
d1_mbp_noninvasive_max             122
d1_mbp_noninvasive_min              91
d1_resprate_max                     79
d1_resprate_min                     55
d1_spo2_max                         43
d1_spo2_min                        101
d1_sysbp_max                       143
d1_sysbp_min                       120
d1_sysbp_noninvasive_max           143
d1_sysbp_noninvasive_min           120
d1_temp_max                        186
d1_temp_min                        209
h1_diasbp_max                      107
h1_diasbp_min                       92
h1_diasbp_noninvasive_max          108
h1_diasbp_noninvasive_min           93
h1_heartrate_max                   119
h1_heartrate_min                   109
h1_mbp_max                         117
h1_mbp_min                         107
h1_mbp_noninvasive_max             115
h1_mbp_noninvasive_min             107
```

```
h1_resprate_max                    50
h1_resprate_min                    91
h1_spo2_max                        72
h1_spo2_min                       100
h1_sysbp_max                      149
h1_sysbp_min                      142
h1_sysbp_noninvasive_max          149
h1_sysbp_noninvasive_min          143
d1_glucose_max                    538
d1_glucose_min                    256
d1_potassium_max                  100
d1_potassium_min                  116
apache_4a_hospital_death_prob     101
apache_4a_icu_death_prob           99
aids                                2
cirrhosis                           2
diabetes_mellitus                   2
hepatic_failure                     2
immunosuppression                   2
leukemia                            2
lymphoma                            2
solid_tumor_with_metastasis         2
apache_3j_bodysystem               11
apache_2_bodysystem                10
Unnamed: 70                         0
hospital_death                      2
dtype: int64
```

Data Info

In [ ]:

```
data.info()
```

Out[ ]:

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 91713 entries, 0 to 91712
Data columns (total 86 columns):
 #   Column                         Non-Null Count  Dtype
---  ------                         --------------  -----
 0   encounter_id                   91713 non-null  int64
 1   patient_id                     91713 non-null  int64
 2   hospital_id                    91713 non-null  int64
 3   age                            87485 non-null  float64
 4   bmi                            88284 non-null  float64
 5   ethnicity                      90318 non-null  object
 6   gender                         91688 non-null  object
 7   height                         90379 non-null  float64
 8   weight                         88993 non-null  float64
 9   elective_surgery               91713 non-null  int64
```

```
10  icu_admit_source             91601 non-null  object
11  icu_id                       91713 non-null  int64
12  icu_stay_type                91713 non-null  object
13  icu_type                     91713 non-null  object
14  apache_post_operative_x      91713 non-null  int64
15  apache_2_diagnosis           90051 non-null  float64
16  apache_3j_diagnosis          90612 non-null  float64
17  pre_icu_los_days             91713 non-null  float64
18  apache_post_operative_y      91713 non-null  int64
19  arf_apache                   90998 non-null  float64
20  gcs_eyes_apache              89812 non-null  float64
21  gcs_motor_apache             89812 non-null  float64
22  gcs_unable_apache            90676 non-null  float64
23  gcs_verbal_apache            89812 non-null  float64
24  heart_rate_apache            90835 non-null  float64
25  intubated_apache             90998 non-null  float64
26  map_apache                   90719 non-null  float64
27  resprate_apache              90479 non-null  float64
28  temp_apache                  87605 non-null  float64
29  ventilated_apache            90998 non-null  float64
30  d1_diasbp_max                91548 non-null  float64
31  d1_diasbp_min                91548 non-null  float64
32  d1_diasbp_noninvasive_max    90673 non-null  float64
33  d1_diasbp_noninvasive_min    90673 non-null  float64
34  d1_heartrate_max             91568 non-null  float64
35  d1_heartrate_min             91568 non-null  float64
36  d1_mbp_max                   91493 non-null  float64
37  d1_mbp_min                   91493 non-null  float64
38  d1_mbp_noninvasive_max       90234 non-null  float64
39  d1_mbp_noninvasive_min       90234 non-null  float64
40  d1_resprate_max              91328 non-null  float64
41  d1_resprate_min              91328 non-null  float64
42  d1_spo2_max                  91380 non-null  float64
43  d1_spo2_min                  91380 non-null  float64
44  d1_sysbp_max                 91554 non-null  float64
45  d1_sysbp_min                 91554 non-null  float64
46  d1_sysbp_noninvasive_max     90686 non-null  float64
47  d1_sysbp_noninvasive_min     90686 non-null  float64
48  d1_temp_max                  89389 non-null  float64
49  d1_temp_min                  89389 non-null  float64
50  h1_diasbp_max                88094 non-null  float64
51  h1_diasbp_min                88094 non-null  float64
52  h1_diasbp_noninvasive_max    84363 non-null  float64
53  h1_diasbp_noninvasive_min    84363 non-null  float64
54  h1_heartrate_max             88923 non-null  float64
55  h1_heartrate_min             88923 non-null  float64
56  h1_mbp_max                   87074 non-null  float64
57  h1_mbp_min                   87074 non-null  float64
58  h1_mbp_noninvasive_max       82629 non-null  float64
```

```
 59  h1_mbp_noninvasive_min         82629 non-null  float64
 60  h1_resprate_max                87356 non-null  float64
 61  h1_resprate_min                87356 non-null  float64
 62  h1_spo2_max                    87528 non-null  float64
 63  h1_spo2_min                    87528 non-null  float64
 64  h1_sysbp_max                   88102 non-null  float64
 65  h1_sysbp_min                   88102 non-null  float64
 66  h1_sysbp_noninvasive_max       84372 non-null  float64
 67  h1_sysbp_noninvasive_min       84372 non-null  float64
 68  d1_glucose_max                 85906 non-null  float64
 69  d1_glucose_min                 85906 non-null  float64
 70  d1_potassium_max               82128 non-null  float64
 71  d1_potassium_min               82128 non-null  float64
 72  apache_4a_hospital_death_prob  83766 non-null  float64
 73  apache_4a_icu_death_prob       83766 non-null  float64
 74  aids                           90998 non-null  float64
 75  cirrhosis                      90998 non-null  float64
 76  diabetes_mellitus              90998 non-null  float64
 77  hepatic_failure                90998 non-null  float64
 78  immunosuppression              90998 non-null  float64
 79  leukemia                       90998 non-null  float64
 80  lymphoma                       90998 non-null  float64
 81  solid_tumor_with_metastasis    90998 non-null  float64
 82  apache_3j_bodysystem           90051 non-null  object
 83  apache_2_bodysystem            90051 non-null  object
 84  Unnamed: 70                    0 non-null      float64
 85  hospital_death                 91713 non-null  int64
dtypes: float64(71), int64(8), object(7)
memory usage: 60.9+ MB
```

Remove unnecessary values: all id columns and unnamed:70 column which has all null values

'encounter_id', 'patient_id', 'hospital_id', 'icu_id', 'Unnamed: 70', 'apache_post_operative_y'

In [ ]:
```
data.drop(['encounter_id', 'patient_id', 'hospital_id', 'icu_id', 'Unnamed: 70', 'apache_post_operative_y'], axis= 1, inplace= True)
```

Data Shape - 80 attributes

In [ ]:
```
data.shape
```

Out[ ]:
```
(91713, 80)
```

Changing Data Type for categorical Variables

In [ ]:

```
data[['elective_surgery', 'apache_post_operative_x',
      'arf_apache','gcs_eyes_apache', 'gcs_motor_apache',
      'gcs_unable_apache','gcs_verbal_apache', 'intubated_apache',
      'ventilated_apache','aids','cirrhosis',
      'diabetes_mellitus','hepatic_failure', 'immunosuppression',
      'leukemia', 'lymphoma','solid_tumor_with_metastasis']] =
data[['elective_surgery', 'apache_post_operative_x', 'arf_apache',
      'gcs_eyes_apache', 'gcs_motor_apache','gcs_unable_apache',
      'gcs_verbal_apache', 'intubated_apache', 'ventilated_apache','aids',
      'cirrhosis','diabetes_mellitus','hepatic_failure',
      'immunosuppression', 'leukemia','lymphoma',
      'solid_tumor_with_metastasis']].astype(str)
```

Data Type change to Object

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

Out[ ]:
```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 91713 entries, 0 to 91712
Data columns (total 80 columns):
 #   Column                         Non-Null Count  Dtype  
---  ------                         --------------  -----  
 0   age                            87485 non-null  float64
 1   bmi                            88284 non-null  float64
 2   ethnicity                      90318 non-null  object 
 3   gender                         91688 non-null  object 
 4   height                         90379 non-null  float64
 5   weight                         88993 non-null  float64
 6   elective_surgery               91713 non-null  object 
 7   icu_admit_source               91601 non-null  object 
 8   icu_stay_type                  91713 non-null  object 
 9   icu_type                       91713 non-null  object 
 10  apache_post_operative_x        91713 non-null  object 
 11  apache_2_diagnosis             90051 non-null  float64
 12  apache_3j_diagnosis            90612 non-null  float64
 13  pre_icu_los_days               91713 non-null  float64
 14  arf_apache                     91713 non-null  object 
 15  gcs_eyes_apache                91713 non-null  object 
 16  gcs_motor_apache               91713 non-null  object 
 17  gcs_unable_apache              91713 non-null  object 
 18  gcs_verbal_apache              91713 non-null  object 
 19  heart_rate_apache              90835 non-null  float64
 20  intubated_apache               91713 non-null  object 
 21  map_apache                     90719 non-null  float64
 22  resprate_apache                90479 non-null  float64
 23  temp_apache                    87605 non-null  float64
 24  ventilated_apache              91713 non-null  object 
```

```
25  d1_diasbp_max                 91548 non-null  float64
26  d1_diasbp_min                 91548 non-null  float64
27  d1_diasbp_noninvasive_max     90673 non-null  float64
28  d1_diasbp_noninvasive_min     90673 non-null  float64
29  d1_heartrate_max              91568 non-null  float64
30  d1_heartrate_min              91568 non-null  float64
31  d1_mbp_max                    91493 non-null  float64
32  d1_mbp_min                    91493 non-null  float64
33  d1_mbp_noninvasive_max        90234 non-null  float64
34  d1_mbp_noninvasive_min        90234 non-null  float64
35  d1_resprate_max               91328 non-null  float64
36  d1_resprate_min               91328 non-null  float64
37  d1_spo2_max                   91380 non-null  float64
38  d1_spo2_min                   91380 non-null  float64
39  d1_sysbp_max                  91554 non-null  float64
40  d1_sysbp_min                  91554 non-null  float64
41  d1_sysbp_noninvasive_max      90686 non-null  float64
42  d1_sysbp_noninvasive_min      90686 non-null  float64
43  d1_temp_max                   89389 non-null  float64
44  d1_temp_min                   89389 non-null  float64
45  h1_diasbp_max                 88094 non-null  float64
46  h1_diasbp_min                 88094 non-null  float64
47  h1_diasbp_noninvasive_max     84363 non-null  float64
48  h1_diasbp_noninvasive_min     84363 non-null  float64
49  h1_heartrate_max              88923 non-null  float64
50  h1_heartrate_min              88923 non-null  float64
51  h1_mbp_max                    87074 non-null  float64
52  h1_mbp_min                    87074 non-null  float64
53  h1_mbp_noninvasive_max        82629 non-null  float64
54  h1_mbp_noninvasive_min        82629 non-null  float64
55  h1_resprate_max               87356 non-null  float64
56  h1_resprate_min               87356 non-null  float64
57  h1_spo2_max                   87528 non-null  float64
58  h1_spo2_min                   87528 non-null  float64
59  h1_sysbp_max                  88102 non-null  float64
60  h1_sysbp_min                  88102 non-null  float64
61  h1_sysbp_noninvasive_max      84372 non-null  float64
62  h1_sysbp_noninvasive_min      84372 non-null  float64
63  d1_glucose_max                85906 non-null  float64
64  d1_glucose_min                85906 non-null  float64
65  d1_potassium_max              82128 non-null  float64
66  d1_potassium_min              82128 non-null  float64
67  apache_4a_hospital_death_prob 83766 non-null  float64
68  apache_4a_icu_death_prob      83766 non-null  float64
69  aids                          91713 non-null  object
70  cirrhosis                     91713 non-null  object
71  diabetes_mellitus             91713 non-null  object
72  hepatic_failure               91713 non-null  object
73  immunosuppression             91713 non-null  object
```

```
 74   leukemia                        91713 non-null  object
 75   lymphoma                        91713 non-null  object
 76   solid_tumor_with_metastasis     91713 non-null  object
 77   apache_3j_bodysystem            90051 non-null  object
 78   apache_2_bodysystem             90051 non-null  object
 79   hospital_death                  91713 non-null  int64
dtypes: float64(55), int64(1), object(24)
memory usage: 56.7+ MB
```

Data Summary

In [ ]:
```
data.describe().T
```

Out[ ]:

	count	mean	std	min	25%	50%	75%	max
age	87485.0	62.309516	16.775119	16.000000	52.000000	65.000000	75.000000	89.000000
bmi	88284.0	29.185818	8.275142	14.844926	23.641975	27.654655	32.930206	67.814990
height	90379.0	169.641588	10.795378	137.200000	162.500000	170.100000	177.800000	195.590000
weight	88993.0	84.028340	25.011497	38.600000	66.800000	80.300000	97.100000	186.000000
apache_2_diagnosis	90051.0	185.401739	86.050882	101.000000	113.000000	122.000000	301.000000	308.000000
apache_3j_diagnosis	90612.0	558.216377	463.266985	0.010000	203.010000	409.020000	703.030000	2201.050000
pre_icu_los_days	91713.0	0.835766	2.487756	-24.947222	0.035417	0.138889	0.409028	159.090972
heart_rate_apache	90835.0	99.707932	30.870502	30.000000	86.000000	104.000000	120.000000	178.000000
map_apache	90719.0	88.015873	42.032412	40.000000	54.000000	67.000000	125.000000	200.000000
resprate_apache	90479.0	25.811007	15.106312	4.000000	11.000000	28.000000	36.000000	60.000000
temp_apache	87605.0	36.414472	0.833496	32.100000	36.200000	36.500000	36.700000	39.700000
d1_diasbp_max	91548.0	88.491873	19.798379	46.000000	75.000000	86.000000	99.000000	165.000000
d1_diasbp_min	91548.0	50.161314	13.317586	13.000000	42.000000	50.000000	58.000000	90.000000
d1_diasbp_noninvasive_max	90673.0	88.610513	19.793743	46.000000	75.000000	87.000000	99.000000	165.000000
d1_diasbp_noninvasive_min	90673.0	50.242597	13.341521	13.000000	42.000000	50.000000	58.000000	90.000000
d1_heartrate_max	91568.0	103.000568	22.017346	58.000000	87.000000	101.000000	116.000000	177.000000
d1_heartrate_min	91568.0	70.321848	17.115903	0.000000	60.000000	69.000000	81.000000	175.000000

d1_mbp_max	91493.0	104.651339	20.808358	60.000000	90.000000	102.000000	116.000000	184.000000
d1_mbp_min	91493.0	64.871859	15.679680	22.000000	55.000000	64.000000	75.000000	112.000000
d1_mbp_noninvasive_max	90234.0	104.590454	20.701171	60.000000	90.000000	102.000000	116.000000	181.000000
d1_mbp_noninvasive_min	90234.0	64.941541	15.701305	22.000000	55.000000	64.000000	75.000000	112.000000
d1_resprate_max	91328.0	28.882774	10.701973	14.000000	22.000000	26.000000	32.000000	92.000000
d1_resprate_min	91328.0	12.846279	5.064943	0.000000	10.000000	13.000000	16.000000	100.000000
d1_spo2_max	91380.0	99.241836	1.794181	0.000000	99.000000	100.000000	100.000000	100.000000
d1_spo2_min	91380.0	90.454826	10.030069	0.000000	89.000000	92.000000	95.000000	100.000000
d1_sysbp_max	91554.0	148.339745	25.733259	90.000000	130.000000	146.000000	164.000000	232.000000
d1_sysbp_min	91554.0	96.923870	20.677930	41.000000	83.000000	96.000000	110.000000	160.000000
d1_sysbp_noninvasive_max	90686.0	148.235549	25.792453	90.000000	130.000000	146.000000	164.000000	232.000000
d1_sysbp_noninvasive_min	90686.0	96.993313	20.705016	41.030000	84.000000	96.000000	110.000000	160.000000
d1_temp_max	89389.0	37.284201	0.693287	35.100000	36.900000	37.110000	37.600000	39.900000
d1_temp_min	89389.0	36.268391	0.745147	31.889000	36.100000	36.400000	36.660000	37.800000
h1_diasbp_max	88094.0	75.354508	18.409190	37.000000	62.000000	74.000000	86.000000	143.000000
h1_diasbp_min	88094.0	62.838150	16.363229	22.000000	52.000000	62.000000	73.000000	113.000000
h1_diasbp_noninvasive_max	84363.0	75.805934	18.481826	37.000000	63.000000	74.000000	87.000000	144.000000
h1_diasbp_noninvasive_min	84363.0	63.270616	16.422063	22.000000	52.000000	62.000000	74.000000	114.000000
h1_heartrate_max	88923.0	92.229198	21.823704	46.000000	77.000000	90.000000	106.000000	164.000000
h1_heartrate_min	88923.0	83.663720	20.279869	36.000000	69.000000	82.000000	97.000000	144.000000
h1_mbp_max	87074.0	91.612950	20.533174	49.000000	77.000000	90.000000	104.000000	165.000000
h1_mbp_min	87074.0	79.400028	19.130590	32.000000	66.000000	78.000000	92.000000	138.000000
h1_mbp_noninvasive_max	82629.0	91.594126	20.552018	49.000000	77.000000	90.000000	104.000000	163.000000
h1_mbp_noninvasive_min	82629.0	79.709315	19.236507	32.000000	66.000000	79.000000	92.000000	138.000000
h1_resprate_max	87356.0	22.633614	7.515043	10.000000	18.000000	21.000000	26.000000	59.000000
h1_resprate_min	87356.0	17.211525	6.072588	0.000000	14.000000	16.000000	20.000000	189.000000
h1_spo2_max	87528.0	98.044637	3.212934	0.000000	97.000000	99.000000	100.000000	100.000000

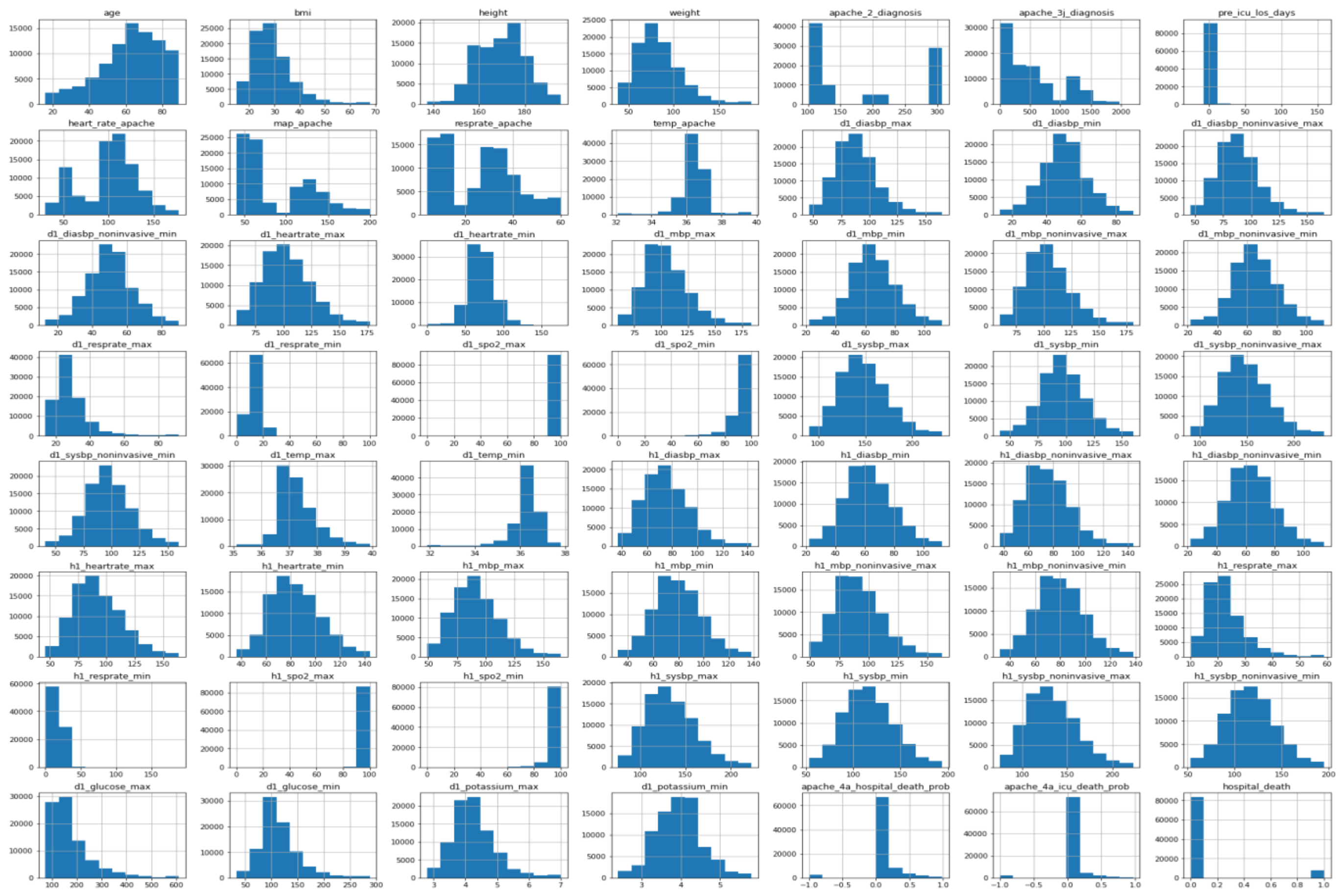
h1_spo2_min	87528.0	95.174310	6.625227	0.000000	94.000000	96.000000	99.000000	100.000000
h1_sysbp_max	88102.0	133.247395	27.556986	75.000000	113.000000	131.000000	150.000000	223.000000
h1_sysbp_min	88102.0	116.362296	26.510637	53.000000	98.000000	115.000000	134.000000	194.000000
h1_sysbp_noninvasive_max	84372.0	133.054686	27.679751	75.000000	113.000000	130.000000	150.000000	223.000000
h1_sysbp_noninvasive_min	84372.0	116.549625	26.623528	53.000000	98.000000	115.000000	134.000000	195.000000
d1_glucose_max	85906.0	174.638023	86.687955	73.000000	117.000000	150.000000	201.000000	611.000000
d1_glucose_min	85906.0	114.380940	38.273013	33.000000	91.000000	107.000000	131.000000	288.000000
d1_potassium_max	82128.0	4.251594	0.667355	2.800000	3.800000	4.200000	4.600000	7.000000
d1_potassium_min	82128.0	3.934658	0.579610	2.400000	3.600000	3.900000	4.300000	5.800000
apache_4a_hospital_death_prob	83766.0	0.086787	0.247569	-1.000000	0.020000	0.050000	0.130000	0.990000
apache_4a_icu_death_prob	83766.0	0.043955	0.217341	-1.000000	0.010000	0.020000	0.060000	0.970000
hospital_death	91713.0	0.086302	0.280811	0.000000	0.000000	0.000000	0.000000	1.000000

Frequency Distribution

Numeric Data

In [ ]:
```
df1 = data.select_dtypes([np.int64, np.float64])
hist_frame(df1, figsize= (30, 25))
```

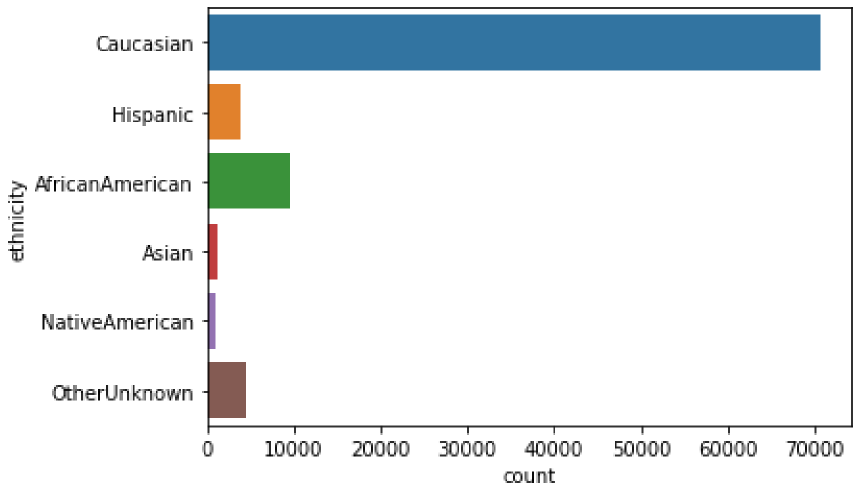
Out[ ]:
```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x7ffb97427fa0>,
         <matplotlib.axes._subplots.AxesSubplot object at 0x7ffb9d580e50>,
         <matplotlib.axes._subplots.AxesSubplot object at 0x7ffb97436880>,
         <matplotlib.axes._subplots.AxesSubplot object at 0x7ffb973bd1c0>,
         <matplotlib.axes._subplots.AxesSubplot object at 0x7ffb973655b0>,
         <matplotlib.axes._subplots.AxesSubplot object at 0x7ffb973918e0>,
         <matplotlib.axes._subplots.AxesSubplot object at 0x7ffb973919d0>],
        [<matplotlib.axes._subplots.AxesSubplot object at 0x7ffb9733fe50>,
         <matplotlib.axes._subplots.AxesSubplot object at 0x7ffb972a6610>,
         <matplotlib.axes._subplots.AxesSubplot object at 0x7ffb972d4a00>,
         <matplotlib.axes._subplots.AxesSubplot object at 0x7ffb97281e20>,
         <matplotlib.axes._subplots.AxesSubplot object at 0x7ffb9723b250>,
         <matplotlib.axes._subplots.AxesSubplot object at 0x7ffb971e7640>,
         <matplotlib.axes._subplots.AxesSubplot object at 0x7ffb97211a30>],
        [<matplotlib.axes._subplots.AxesSubplot object at 0x7ffb971c1e20>,
         <matplotlib.axes._subplots.AxesSubplot object at 0x7ffb9717b280>,
         <matplotlib.axes._subplots.AxesSubplot object at 0x7ffb971286a0>,
         <matplotlib.axes._subplots.AxesSubplot object at 0x7ffb97154a90>,
         <matplotlib.axes._subplots.AxesSubplot object at 0x7ffb97102e80>,
         <matplotlib.axes._subplots.AxesSubplot object at 0x7ffb970bd2b0>,
         <matplotlib.axes._subplots.AxesSubplot object at 0x7ffb970686a0>],
        [<matplotlib.axes._subplots.AxesSubplot object at 0x7ffb97095af0>,
         <matplotlib.axes._subplots.AxesSubplot object at 0x7ffb97044ee0>,
         <matplotlib.axes._subplots.AxesSubplot object at 0x7ffb96ffb370>,
         <matplotlib.axes._subplots.AxesSubplot object at 0x7ffb96faa760>,
         <matplotlib.axes._subplots.AxesSubplot object at 0x7ffb96fd5ac0>,
         <matplotlib.axes._subplots.AxesSubplot object at 0x7ffb96f8e220>,
         <matplotlib.axes._subplots.AxesSubplot object at 0x7ffb96f38940>],
        [<matplotlib.axes._subplots.AxesSubplot object at 0x7ffb96eed160>,
         <matplotlib.axes._subplots.AxesSubplot object at 0x7ffb96f17a60>,
         <matplotlib.axes._subplots.AxesSubplot object at 0x7ffb96ed7250>,
         <matplotlib.axes._subplots.AxesSubplot object at 0x7ffb96e7e670>,
         <matplotlib.axes._subplots.AxesSubplot object at 0x7ffb96ea1df0>,
         <matplotlib.axes._subplots.AxesSubplot object at 0x7ffb96e59520>,
         <matplotlib.axes._subplots.AxesSubplot object at 0x7ffb96e01c40>],
        [<matplotlib.axes._subplots.AxesSubplot object at 0x7ffb96db83a0>,
         <matplotlib.axes._subplots.AxesSubplot object at 0x7ffb96ddfaf0>,
         <matplotlib.axes._subplots.AxesSubplot object at 0x7ffb96d992b0>,
         <matplotlib.axes._subplots.AxesSubplot object at 0x7ffb96d419a0>,
         <matplotlib.axes._subplots.AxesSubplot object at 0x7ffb96cea160>,
         <matplotlib.axes._subplots.AxesSubplot object at 0x7ffb96d20820>,
         <matplotlib.axes._subplots.AxesSubplot object at 0x7ffb96ccaf40>],
        [<matplotlib.axes._subplots.AxesSubplot object at 0x7ffb96c806a0>,
         <matplotlib.axes._subplots.AxesSubplot object at 0x7ffb96c2adc0>,
         <matplotlib.axes._subplots.AxesSubplot object at 0x7ffb96c60520>,
         <matplotlib.axes._subplots.AxesSubplot object at 0x7ffb96c08c40>,
         <matplotlib.axes._subplots.AxesSubplot object at 0x7ffb96bbf3d0>,
         <matplotlib.axes._subplots.AxesSubplot object at 0x7ffb96b6aaf0>,
         <matplotlib.axes._subplots.AxesSubplot object at 0x7ffb96b22250>],
        [<matplotlib.axes._subplots.AxesSubplot object at 0x7ffb96b4d970>,
         <matplotlib.axes._subplots.AxesSubplot object at 0x7ffb96af6130>,
         <matplotlib.axes._subplots.AxesSubplot object at 0x7ffb96aaa7f0>,
         <matplotlib.axes._subplots.AxesSubplot object at 0x7ffbbe7762b0>,
         <matplotlib.axes._subplots.AxesSubplot object at 0x7ffb97371cd0>,
         <matplotlib.axes._subplots.AxesSubplot object at 0x7ffb9723bb80>,
         <matplotlib.axes._subplots.AxesSubplot object at 0x7ffb970e7280>]],
       dtype=object)
```

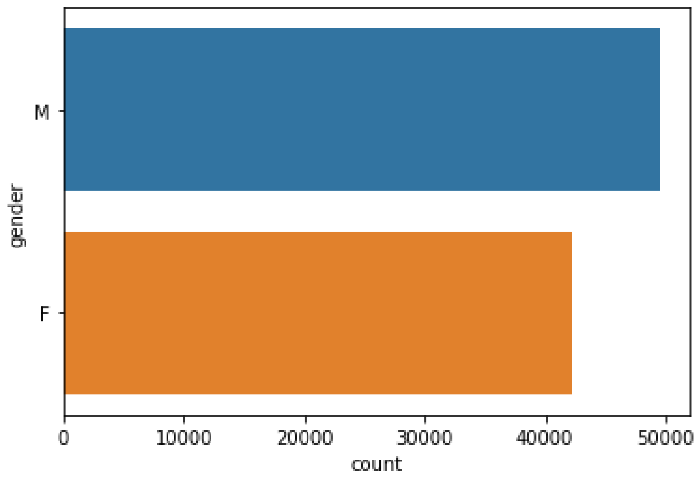


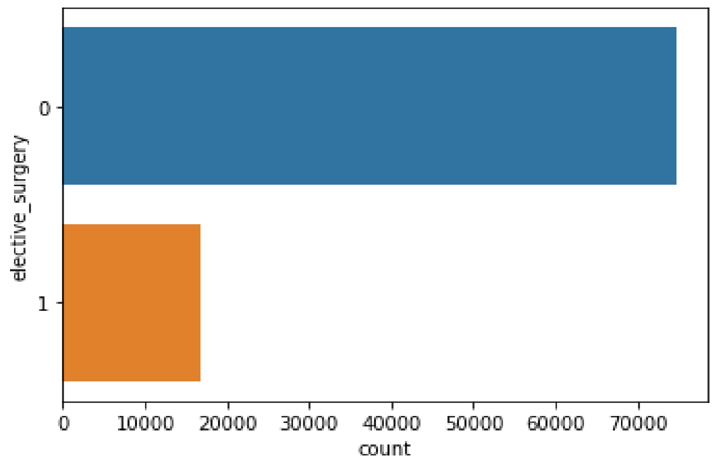
Categorical Data

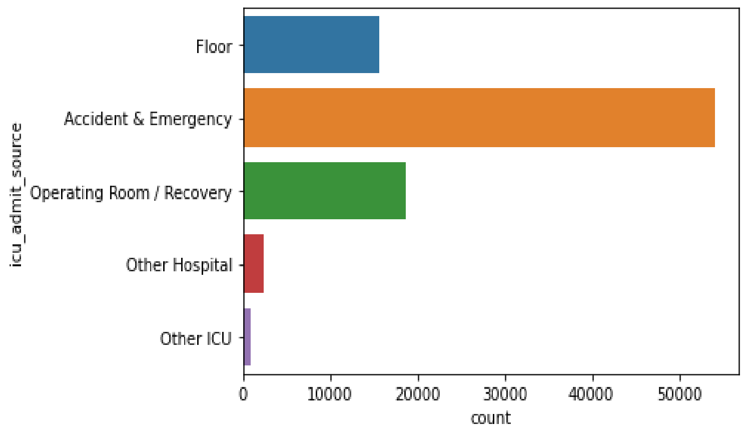
In [ ]:
```
for col in data.select_dtypes(include='object'):
    if data[col].nunique() <= 20:
        sns.countplot(y=col, data=data)
        plt.show()
```

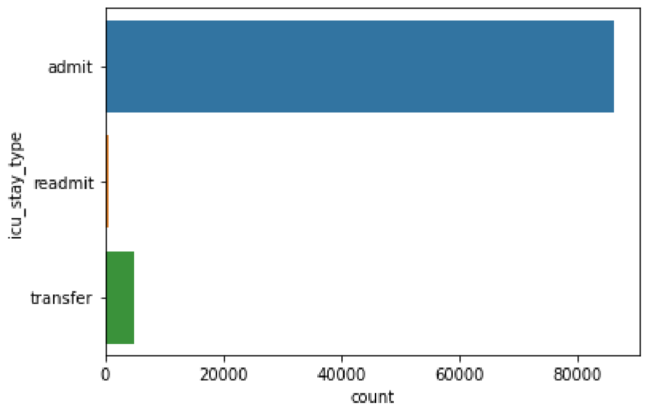
Out [ ]:

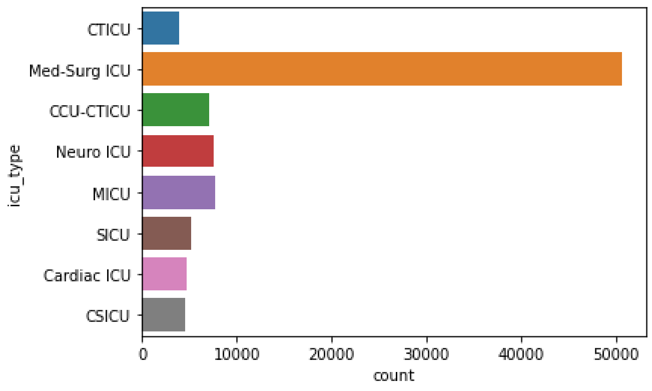


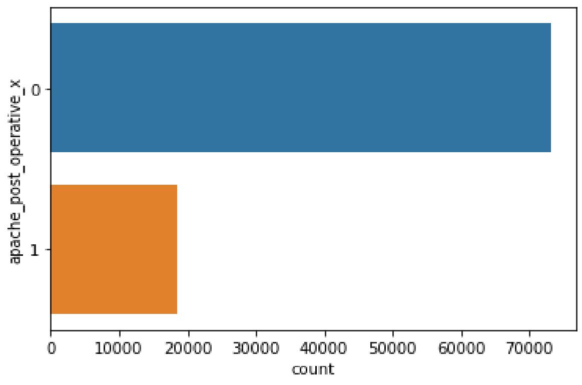

M
F
gender
0
10000
20000
30000
40000
50000
count

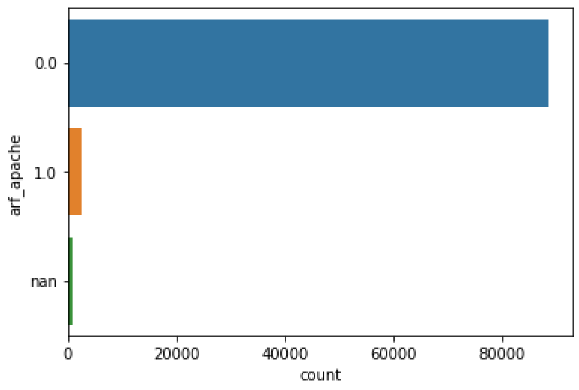

0
1
elective_surgery
0
10000
20000
30000
40000
50000
60000
70000
count

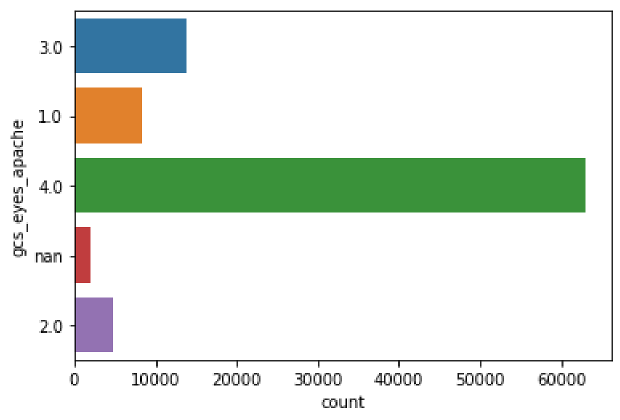

Floor
Accident & Emergency
Operating Room / Recovery
Other Hospital
Other ICU
icu_admit_source
0
10000
20000
30000
40000
50000
count

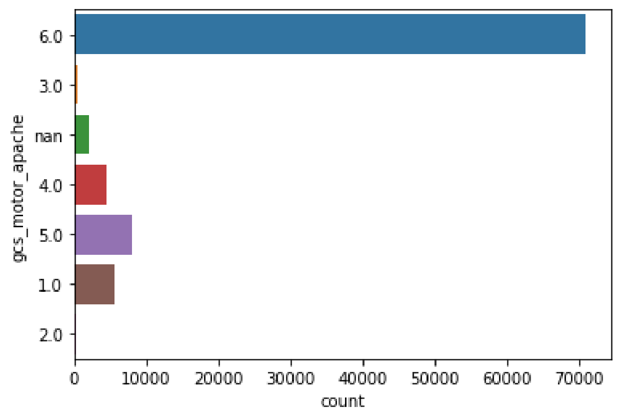

icu_stay_type
admit
readmit
transfer
0
20000
40000
60000
80000
count

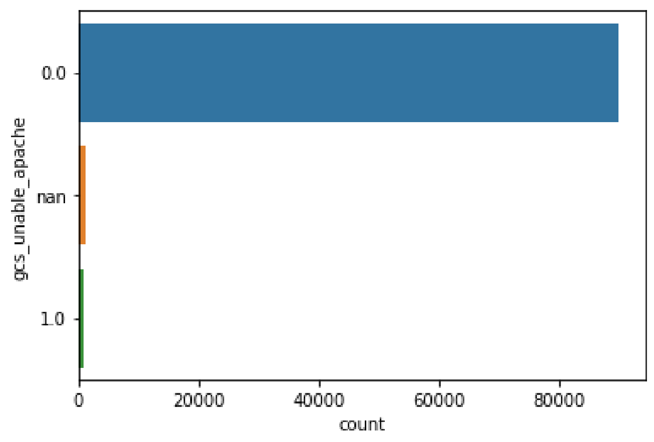

icu_type
CTICU
Med-Surg ICU
CCU-CTICU
Neuro ICU
MICU
SICU
Cardiac ICU
CSICU
0
10000
20000
30000
40000
50000
count

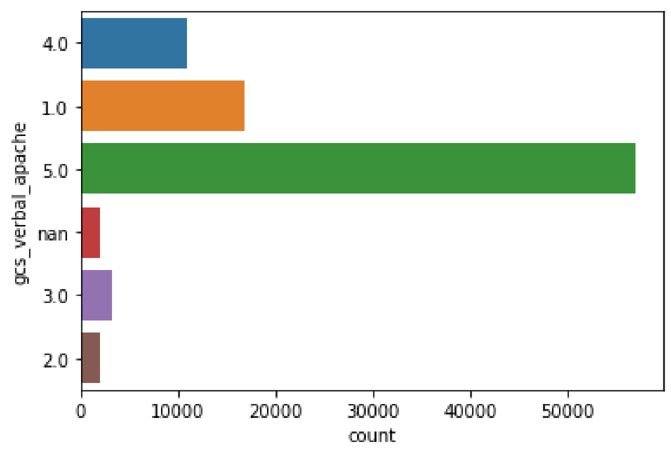

apache_post_operative_x
0
1
0
10000
20000
30000
40000
50000
60000
70000
count

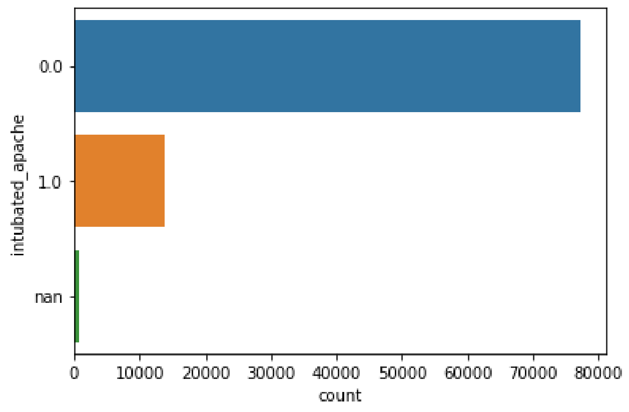

arf_apache
0.0
1.0
nan
0
20000
40000
60000
80000
count

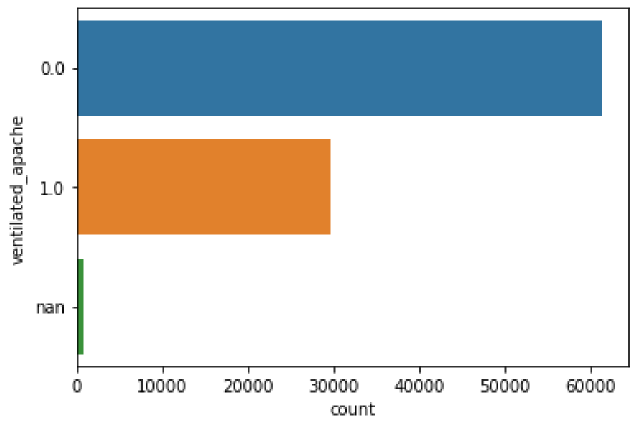

3.0
1.0
4.0
nan
2.0
gcs_eyes_apache
0
10000
20000
30000
40000
50000
60000
count

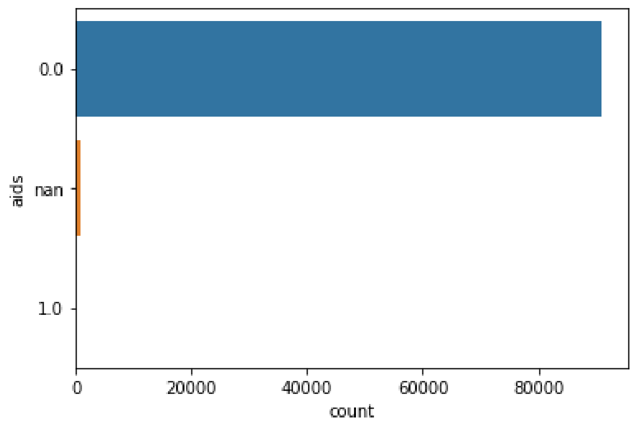

6.0
3.0
nan
4.0
5.0
1.0
2.0
gcs_motor_apache
0
10000
20000
30000
40000
50000
60000
70000
count

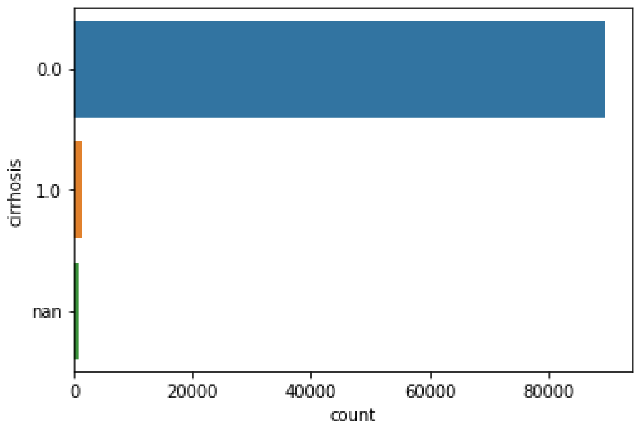

0.0
nan
1.0
gcs_unable_apache
0
20000
40000
60000
80000
count

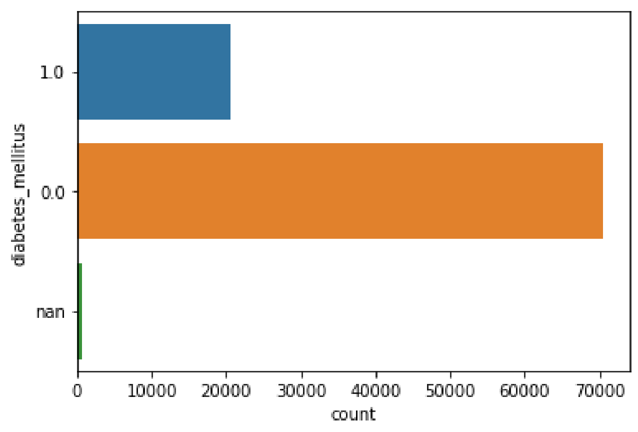

4.0
1.0
5.0
nan
3.0
2.0
gcs_verbal_apache
0
10000
20000
30000
40000
50000
count

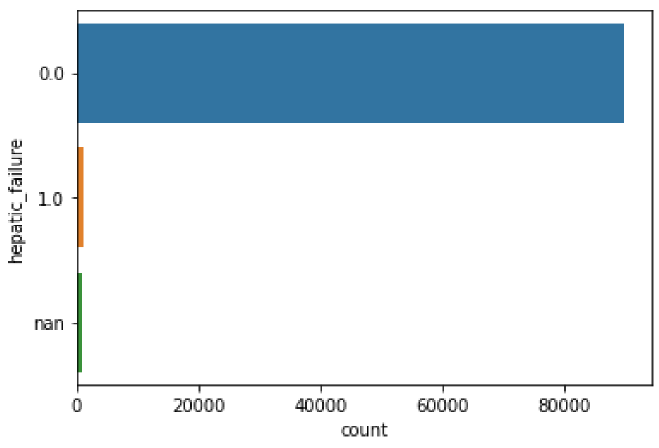

intubated_apache
0.0
1.0
nan
0
10000
20000
30000
40000
50000
60000
70000
80000
count

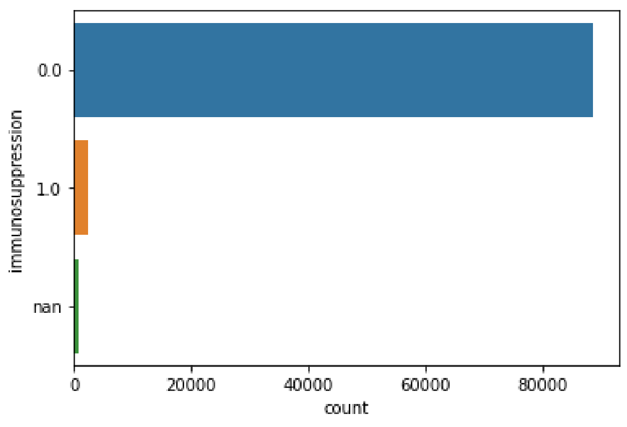

ventilated_apache
0.0
1.0
nan
0
10000
20000
30000
40000
50000
60000
count

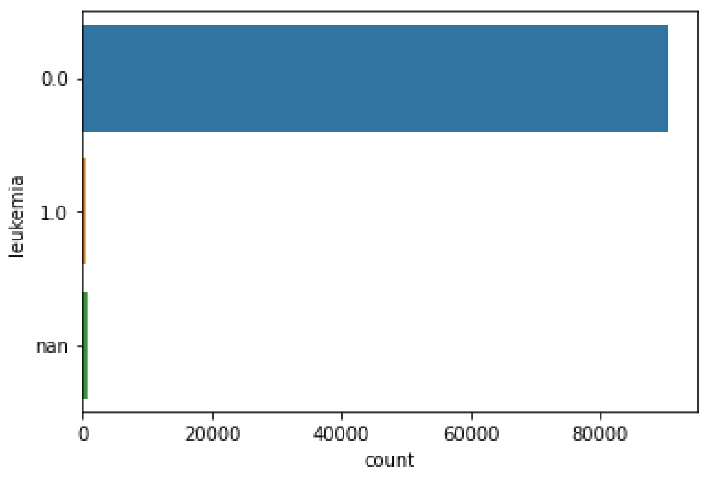

aids
0.0
nan
1.0
0
20000
40000
60000
80000
count

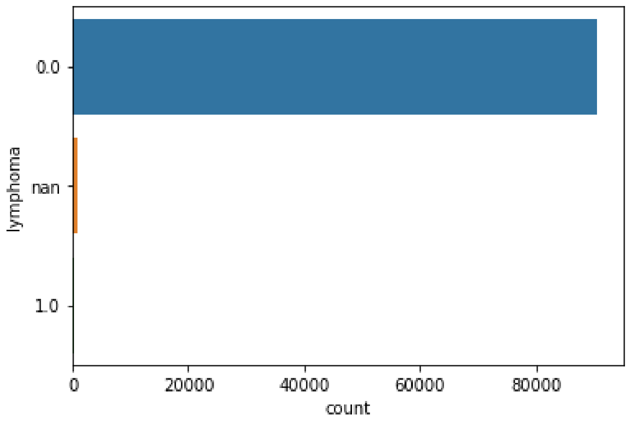

cirrhosis
0.0
1.0
nan
0
20000
40000
60000
80000
count

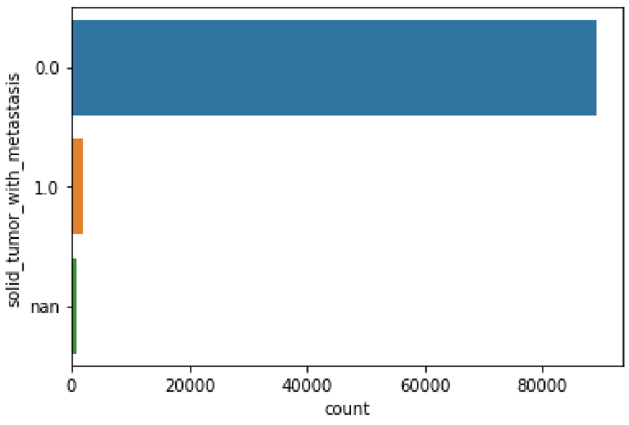

1.0
0.0
nan
diabetes_mellitus
0
10000
20000
30000
40000
50000
60000
70000
count

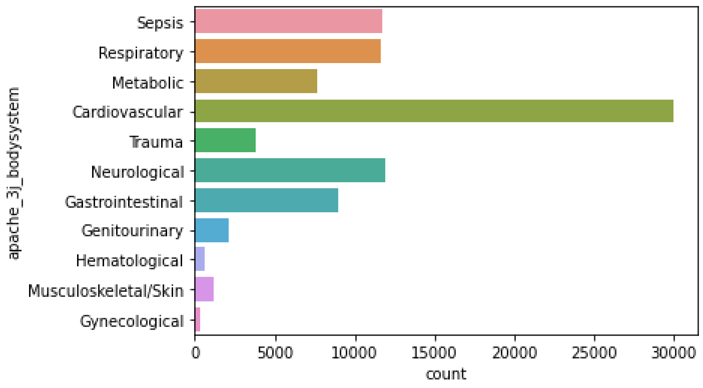

0.0
1.0
nan
hepatic_failure
0
20000
40000
60000
80000
count

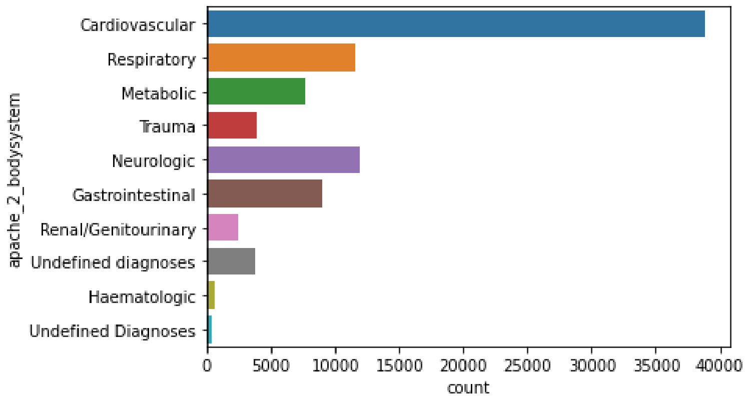

0.0
1.0
nan
immunosuppression
0
20000
40000
60000
80000
count


0.0
1.0
nan
leukemia
0
20000
40000
60000
80000
count


0.0
nan
1.0
lymphoma
0
20000
40000
60000
80000
count


0.0
1.0
nan
solid_tumor_with_metastasis
0
20000
40000
60000
80000
count


Sepsis
Respiratory
Metabolic
Cardiovascular
Trauma
Neurological
Gastrointestinal
Genitourinary
Hematological
Musculoskeletal/Skin
Gynecological
apache_3j_bodysystem
0
5000
10000
15000
20000
25000
30000
count


Cardiovascular
Respiratory
Metabolic
Trauma
Neurologic
Gastrointestinal
Renal/Genitourinary
Undefined diagnoses
Haematologic
Undefined Diagnoses
apache_2_bodysystem
0
5000
10000
15000
20000
25000
30000
35000
40000
count

Exploratory Analysis of dependent Variable

Value Counts - Hospital Death

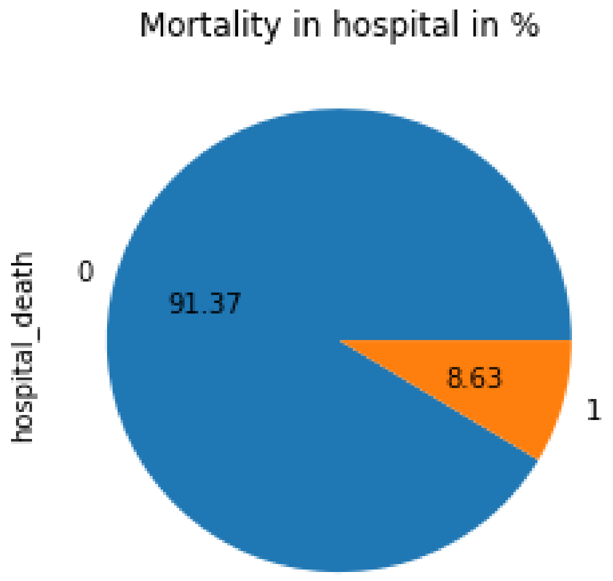
In [ ]:
```
data.hospital_death.value_counts()
```

Out[ ]:
```
0    83798
1     7915
Name: hospital_death, dtype: int64
```

From the above analysis: we can see the data is imbalanced and skewed towards more patient survival

In [ ]:
```
data.hospital_death.value_counts().plot(kind='pie', autopct="%.2f", title
                                        ='Mortality in hospital in %')
```

Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7ffb93b9ebe0>
```



Grouped aggregation distribution of hospital deaths

In [ ]:
```
print(data.groupby(['gender', 'hospital_death'])['hospital_death'].count())
print("\n",data.groupby(['ethnicity',
'hospital_death'])['hospital_death'].count())
print("\n", data.groupby(['aids',
'hospital_death'])['hospital_death'].count())
print("\n", data.groupby(['cirrhosis',
'hospital_death'])['hospital_death'].count())
```

```
print("\n", data.groupby(['diabetes_mellitus',
'hospital_death'])['hospital_death'].count())
print("\n", data.groupby(['hepatic_failure',
'hospital_death'])['hospital_death'].count())
print("\n", data.groupby(['immunosuppression',
'hospital_death'])['hospital_death'].count())
print("\n", data.groupby(['hepatic_failure',
'hospital_death'])['hospital_death'].count())
print("\n", data.groupby(['leukemia',
'hospital_death'])['hospital_death'].count())
print("\n", data.groupby(['lymphoma',
'hospital_death'])['hospital_death'].count())
print("\n", data.groupby(['solid_tumor_with_metastasis',
'hospital_death'])['hospital_death'].count())
```

Out [ ]:

```
gender  hospital_death
F       0                 38488
        1                  3731
M       0                 45293
        1                  4176
Name: hospital_death, dtype: int64

ethnicity        hospital_death
AfricanAmerican  0                  8797
                 1                   750
Asian            0                  1036
                 1                    93
Caucasian        0                 64516
                 1                  6168
Hispanic         0                  3420
                 1                   376
NativeAmerican   0                   718
                 1                    70
OtherUnknown     0                  4021
                 1                   353
Name: hospital_death, dtype: int64

aids  hospital_death
0.0   0                 83100
      1                  7820
1.0   0                    68
      1                    10
nan   0                   630
      1                    85
Name: hospital_death, dtype: int64

cirrhosis  hospital_death
0.0        0                 81988
```

```
          1                 7582
1.0       0                 1180
          1                  248
nan       0                  630
          1                   85
Name: hospital_death, dtype: int64

diabetes_mellitus  hospital_death
0.0                0                 64271
                   1                  6235
1.0                0                 18897
                   1                  1595
nan                0                   630
                   1                    85
Name: hospital_death, dtype: int64

hepatic_failure  hospital_death
0.0              0                 82200
                 1                  7616
1.0              0                   968
                 1                   214
nan              0                   630
                 1                    85
Name: hospital_death, dtype: int64

immunosuppression  hospital_death
0.0                0                 81171
                   1                  7446
1.0                0                  1997
                   1                   384
nan                0                   630
                   1                    85
Name: hospital_death, dtype: int64

hepatic_failure  hospital_death
0.0              0                 82200
                 1                  7616
1.0              0                   968
                 1                   214
nan              0                   630
                 1                    85
Name: hospital_death, dtype: int64

leukemia  hospital_death
0.0       0                 82644
          1                  7711
1.0       0                   524
          1                   119
nan       0                   630
```

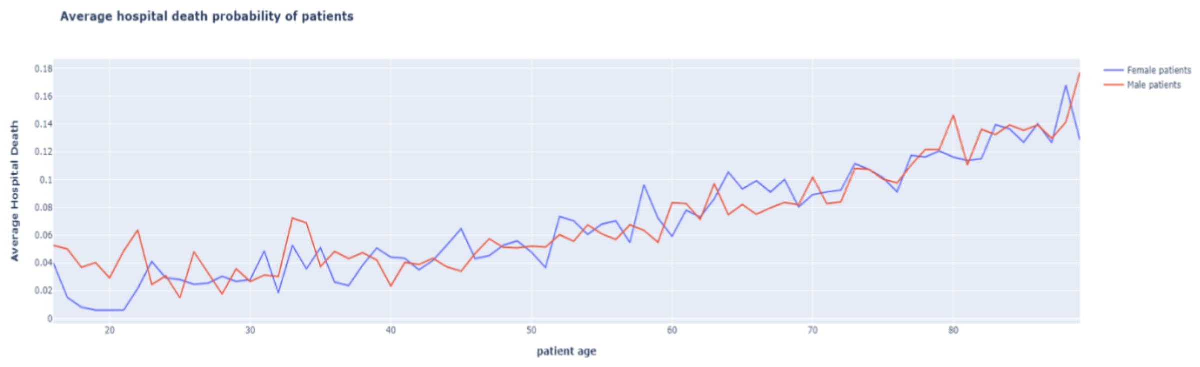
```
          1                    85
Name: hospital_death, dtype: int64

lymphoma  hospital_death
0.0       0                 82855
          1                  7767
1.0       0                   313
          1                    63
nan       0                   630
          1                    85
Name: hospital_death, dtype: int64

solid_tumor_with_metastasis  hospital_death
0.0                          0                 81637
                             1                  7483
1.0                          0                  1531
                             1                   347
nan                          0                   630
                             1                    85
Name: hospital_death, dtype: int64
```

Probability distribution based on Age and Gender

In [ ]:

```
age_death_F=
data[data['gender']=='F'][['age','hospital_death']].groupby('age').mean().r
eset_index()

age_death_M=
data[data['gender']=='M'][['age','hospital_death']].groupby('age').mean().r
eset_index()

fig = make_subplots()
fig.add_trace(go.Scatter(x=age_death_F['age'],
y=age_death_F['hospital_death'], name="Female patients"))
fig.add_trace(go.Scatter(x=age_death_M['age'],
y=age_death_M['hospital_death'],name="Male patients"))
fig.update_layout(title_text="<b>Average hospital death probability of
patients<b>")
fig.update_xaxes(title_text="<b>patient age<b>")
fig.update_yaxes(title_text="<b>Average Hospital Death</b>",
secondary_y=False)
fig.show()
```

Out [ ]:

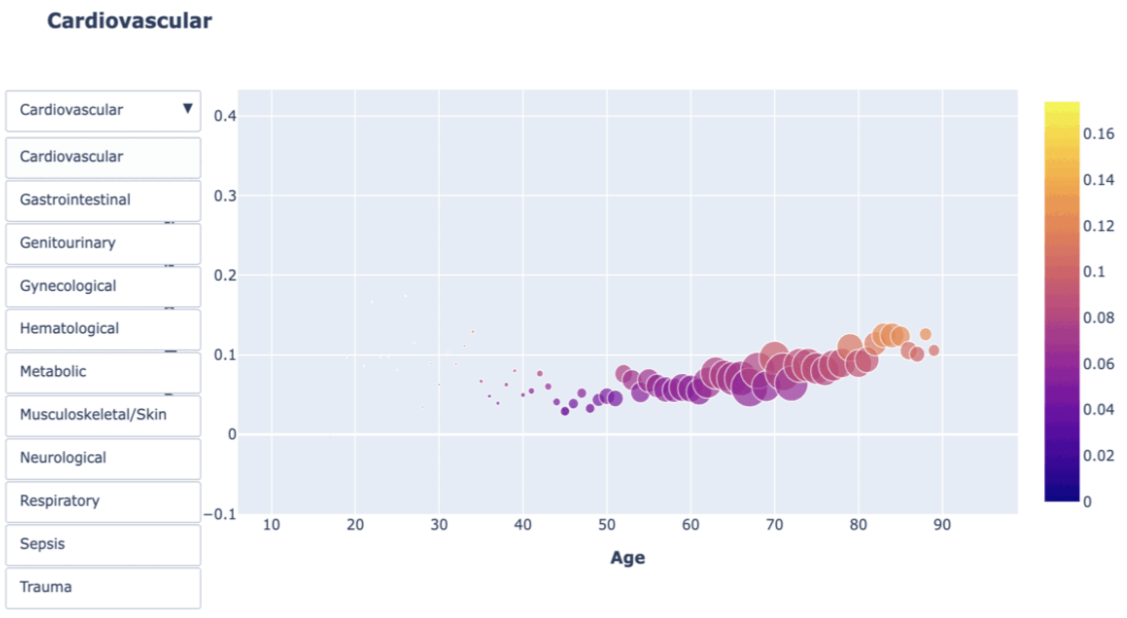


Scatter Plot between Survival rate and Age by diseases

In [ ]:
```
apache3= data[['age','apache_3j_bodysystem','hospital_death']]
apache3=apache3.groupby(['apache_3j_bodysystem','age']).agg(['size','mean']
).reset_index()

apache3['size']=apache3['hospital_death']['size']
apache3['mean']=apache3['hospital_death']['mean']

apache3.drop('hospital_death',axis=1,inplace=True)

systems =list(apache3['apache_3j_bodysystem'].unique())
data1 = []
list_updatemenus = []
for n, s in enumerate(systems):
    visible = [False] * len(systems)
    visible[n] = True
    temp_dict = dict(label = str(s),
                 method = 'update',
                 args = [{'visible': visible},
                         {'title': '<b>'+s+'<b>'}])
    list_updatemenus.append(temp_dict)

for s in systems:
    mask = (apache3['apache_3j_bodysystem'].values == s)
    trace = (dict(visible = False,
        x = apache3.loc[mask, 'age'],
        y = apache3.loc[mask, 'mean'],
        mode = 'markers',
        marker = {'size':apache3.loc[mask,
'size']/apache3.loc[mask,'size'].sum()*1000,
                 'color':apache3.loc[mask, 'mean'],
                 'showscale': True})
                   )
    data1.append(trace)
```

```
data1[0]['visible'] = True

layout = dict(updatemenus=list([dict(buttons= list_updatemenus)]),
              xaxis=dict(title = '<b>Age<b>', range=[min(apache3.loc[:,
'age'])-10, max(apache3.loc[:, 'age']) + 10]),
              yaxis=dict(title = '<b>Average Hospital Death<b>',
range=[min(apache3.loc[:, 'mean'])-0.1, max(apache3.loc[:, 'mean'])+0.1]),
              title='<b>Survival Rate<b>' )
fig = dict(data=data1, layout=layout)
```

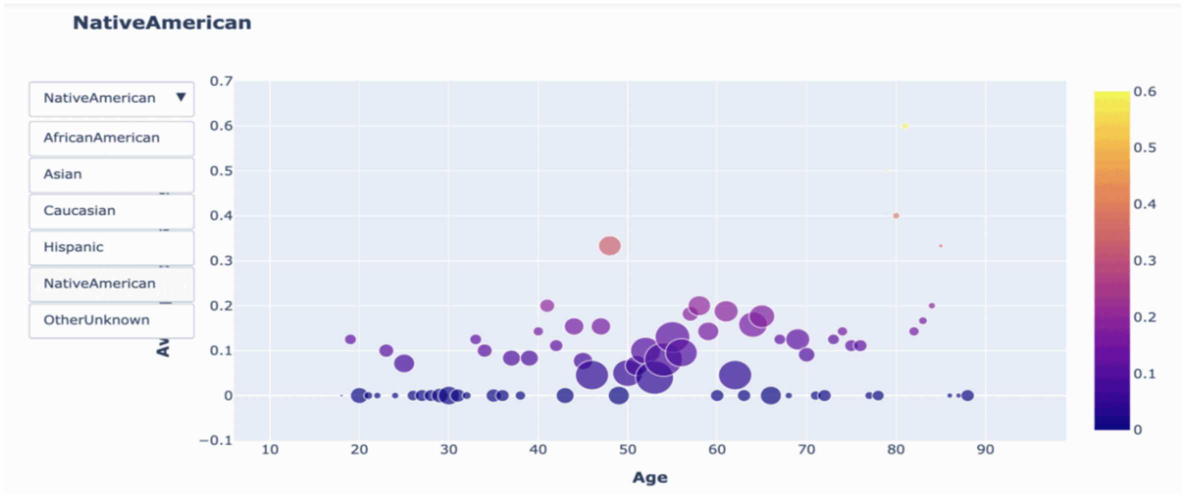
Out [ ]:



Scatter Plot between Survival rate and age by ethnicity

In [ ]:
```
apache3= data[['age','ethnicity','hospital_death']]
apache3=apache3.groupby(['ethnicity','age']).agg(['size','mean']).reset_ind
ex()

apache3['size']=apache3['hospital_death']['size']
apache3['mean']=apache3['hospital_death']['mean']

apache3.drop('hospital_death',axis=1,inplace=True)

systems =list(apache3['ethnicity'].unique())
data1 = []
list_updatemenus = []
for n, s in enumerate(systems):
    visible = [False] * len(systems)
    visible[n] = True
    temp_dict = dict(label = str(s),
                    method = 'update',
                    args = [{'visible': visible},
                            {'title': '<b>'+s+'<b>'}])
    list_updatemenus.append(temp_dict)

for s in systems:
    mask = (apache3['ethnicity'].values == s)
    trace = (dict(visible = False,
        x = apache3.loc[mask, 'age'],
        y = apache3.loc[mask, 'mean'],
        mode = 'markers',
        marker = {'size':apache3.loc[mask,
'size']/apache3.loc[mask,'size'].sum()*1000,
                 'color':apache3.loc[mask, 'mean'],
                 'showscale': True})
                   )
    data1.append(trace)

data1[0]['visible'] = True

layout = dict(updatemenus=list([dict(buttons= list_updatemenus)]),
              xaxis=dict(title = '<b>Age<b>', range=[min(apache3.loc[:,
'age'])-10, max(apache3.loc[:, 'age']) + 10]),
              yaxis=dict(title = '<b>Average Hospital Death<b>',
range=[min(apache3.loc[:, 'mean'])-0.1, max(apache3.loc[:, 'mean'])+0.1]),
              title='<b>Survival Rate<b>' )
fig = dict(data=data1, layout=layout)
py.iplot(fig, filename='update_dropdown')
```

Out [ ]:



Null Values Treatment

Null Values count

In [ ]:
```
data.isnull().sum()
```

Out[ ]:
```
age                              4228
bmi                              3429
ethnicity                        1395
gender                             25
height                           1334
weight                           2720
elective_surgery                    0
icu_admit_source                  112
icu_stay_type                       0
icu_type                            0
apache_post_operative_x             0
apache_2_diagnosis               1662
apache_3j_diagnosis              1101
pre_icu_los_days                    0
arf_apache                          0
gcs_eyes_apache                     0
gcs_motor_apache                    0
gcs_unable_apache                   0
gcs_verbal_apache                   0
heart_rate_apache                 878
intubated_apache                    0
map_apache                        994
resprate_apache                  1234
temp_apache                      4108
ventilated_apache                   0
d1_diasbp_max                     165
d1_diasbp_min                     165
d1_diasbp_noninvasive_max        1040
d1_diasbp_noninvasive_min        1040
d1_heartrate_max                  145
d1_heartrate_min                  145
d1_mbp_max                        220
d1_mbp_min                        220
d1_mbp_noninvasive_max           1479
d1_mbp_noninvasive_min           1479
d1_resprate_max                   385
d1_resprate_min                   385
d1_spo2_max                       333
d1_spo2_min                       333
d1_sysbp_max                      159
d1_sysbp_min                      159
```

```
d1_sysbp_noninvasive_max        1027
d1_sysbp_noninvasive_min        1027
d1_temp_max                     2324
d1_temp_min                     2324
h1_diasbp_max                   3619
h1_diasbp_min                   3619
h1_diasbp_noninvasive_max       7350
h1_diasbp_noninvasive_min       7350
h1_heartrate_max                2790
h1_heartrate_min                2790
h1_mbp_max                      4639
h1_mbp_min                      4639
h1_mbp_noninvasive_max          9084
h1_mbp_noninvasive_min          9084
h1_resprate_max                 4357
h1_resprate_min                 4357
h1_spo2_max                     4185
h1_spo2_min                     4185
h1_sysbp_max                    3611
h1_sysbp_min                    3611
h1_sysbp_noninvasive_max        7341
h1_sysbp_noninvasive_min        7341
d1_glucose_max                  5807
d1_glucose_min                  5807
d1_potassium_max                9585
d1_potassium_min                9585
apache_4a_hospital_death_prob   7947
apache_4a_icu_death_prob        7947
aids                               0
cirrhosis                          0
diabetes_mellitus                  0
hepatic_failure                    0
immunosuppression                  0
leukemia                           0
lymphoma                           0
solid_tumor_with_metastasis        0
apache_3j_bodysystem            1662
apache_2_bodysystem             1662
hospital_death                     0
dtype: int64
```

Null Value Percentage

In [ ]:
```
percent_missing = data.isnull().sum() * 100 / len(data)
missing_value_data = pd.DataFrame({'percent_missing%': percent_missing})
missing_value_data.sort_values(by= 'percent_missing%',ascending =
False).round(2)
```

Out[ ]:

	percent_missing%
d1_potassium_min	10.45
d1_potassium_max	10.45
h1_mbp_noninvasive_min	9.90
h1_mbp_noninvasive_max	9.90
apache_4a_icu_death_prob	8.67
apache_4a_hospital_death_prob	8.67
h1_diasbp_noninvasive_min	8.01
h1_diasbp_noninvasive_max	8.01
h1_sysbp_noninvasive_max	8.00
h1_sysbp_noninvasive_min	8.00
d1_glucose_min	6.33
d1_glucose_max	6.33
h1_mbp_min	5.06
h1_mbp_max	5.06
h1_resprate_max	4.75
h1_resprate_min	4.75
age	4.61
h1_spo2_min	4.56
h1_spo2_max	4.56
temp_apache	4.48
h1_diasbp_max	3.95
h1_diasbp_min	3.95
h1_sysbp_min	3.94
h1_sysbp_max	3.94
bmi	3.74
h1_heartrate_max	3.04
h1_heartrate_min	3.04
weight	2.97
d1_temp_min	2.53
d1_temp_max	2.53
apache_2_bodysystem	1.81
apache_3j_bodysystem	1.81
apache_2_diagnosis	1.81
d1_mbp_noninvasive_max	1.61
d1_mbp_noninvasive_min	1.61
ethnicity	1.52
height	1.45
resprate_apache	1.35

	percent_missing%
apache_3j_diagnosis	1.20
d1_diasbp_noninvasive_min	1.13
d1_diasbp_noninvasive_max	1.13
d1_sysbp_noninvasive_max	1.12
d1_sysbp_noninvasive_min	1.12
map_apache	1.08
heart_rate_apache	0.96
d1_resprate_max	0.42
d1_resprate_min	0.42
d1_spo2_min	0.36
d1_spo2_max	0.36
d1_mbp_min	0.24
d1_mbp_max	0.24
d1_diasbp_max	0.18
d1_diasbp_min	0.18
d1_sysbp_min	0.17
d1_sysbp_max	0.17
d1_heartrate_max	0.16
d1_heartrate_min	0.16
icu_admit_source	0.12
gender	0.03
elective_surgery	0.00
solid_tumor_with_metastasis	0.00
lymphoma	0.00
leukemia	0.00
immunosuppression	0.00
hepatic_failure	0.00
diabetes_mellitus	0.00
cirrhosis	0.00
aids	0.00
icu_stay_type	0.00
icu_type	0.00
apache_post_operative_x	0.00
pre_icu_los_days	0.00
arf_apache	0.00
gcs_eyes_apache	0.00
gcs_motor_apache	0.00
gcs_unable_apache	0.00
gcs_verbal_apache	0.00
intubated_apache	0.00

	percent_missing%
ventilated_apache	0.00
hospital_death	0.00

Drop Null Value Columns - 5% threshold (14 features)

In [ ]:
```
for column in data.columns:
    if ((data[column].isnull().sum() / data[column].shape[0]) > (5/100)):
        print("Dropping column = ", column,
              " as it has more than 5% of data missing")
        data.drop(column, axis = 1, inplace = True)
```

Out[ ]:
```
Dropping column =  h1_diasbp_noninvasive_max  as it has more than 5% of
data missing
Dropping column =  h1_diasbp_noninvasive_min  as it has more than 5% of
data missing
Dropping column =  h1_mbp_max  as it has more than 5% of data missing
Dropping column =  h1_mbp_min  as it has more than 5% of data missing
Dropping column =  h1_mbp_noninvasive_max  as it has more than 5% of data
missing
Dropping column =  h1_mbp_noninvasive_min  as it has more than 5% of data
missing
Dropping column =  h1_sysbp_noninvasive_max  as it has more than 5% of data
missing
Dropping column =  h1_sysbp_noninvasive_min  as it has more than 5% of data
missing
Dropping column =  d1_glucose_max  as it has more than 5% of data missing
Dropping column =  d1_glucose_min  as it has more than 5% of data missing
Dropping column =  d1_potassium_max  as it has more than 5% of data missing
Dropping column =  d1_potassium_min  as it has more than 5% of data missing
Dropping column =  apache_4a_hospital_death_prob  as it has more than 5% of
data missing
Dropping column =  apache_4a_icu_death_prob  as it has more than 5% of data
missing
```

Imputing Null Values - Numrical Data - KNN Imputer

In [ ]:
```
for column in data.columns:
    if data[column].dtype != 'object' and data[column].isnull().sum() > 0:
        print("Imputing column = ", column)
        knn_imputer = KNNImputer(n_neighbors = 3, weights = "uniform")
        data[column] = knn_imputer.fit_transform(data[[column]])
print("Finished imputing data")
```

Out [ ]:
```
Imputing column =  age
Imputing column =  bmi
Imputing column =  height
Imputing column =  weight
Imputing column =  apache_2_diagnosis
Imputing column =  apache_3j_diagnosis
Imputing column =  heart_rate_apache
Imputing column =  map_apache
Imputing column =  resprate_apache
Imputing column =  temp_apache
Imputing column =  d1_diasbp_max
Imputing column =  d1_diasbp_min
Imputing column =  d1_diasbp_noninvasive_max
Imputing column =  d1_diasbp_noninvasive_min
Imputing column =  d1_heartrate_max
Imputing column =  d1_heartrate_min
Imputing column =  d1_mbp_max
Imputing column =  d1_mbp_min
Imputing column =  d1_mbp_noninvasive_max
Imputing column =  d1_mbp_noninvasive_min
Imputing column =  d1_resprate_max
Imputing column =  d1_resprate_min
Imputing column =  d1_spo2_max
Imputing column =  d1_spo2_min
Imputing column =  d1_sysbp_max
Imputing column =  d1_sysbp_min
Imputing column =  d1_sysbp_noninvasive_max
Imputing column =  d1_sysbp_noninvasive_min
Imputing column =  d1_temp_max
Imputing column =  d1_temp_min
Imputing column =  h1_diasbp_max
Imputing column =  h1_diasbp_min
Imputing column =  h1_heartrate_max
Imputing column =  h1_heartrate_min
Imputing column =  h1_resprate_max
Imputing column =  h1_resprate_min
Imputing column =  h1_spo2_max
Imputing column =  h1_spo2_min
Imputing column =  h1_sysbp_max
Imputing column =  h1_sysbp_min
Finished imputing data
```

Fill Null Values – Categorical Attributes

In [ ]:
```
for column in data.columns:
    if data[column].dtype == 'object' and data[column].isnull().sum() > 0:
        print("Imputing column = ", column)
        data[column] = data[column].fillna(data[column].mode()[0])

print("Finished imputing data")
```

Out [ ]:
```
Imputing column =  ethnicity
Imputing column =  gender
Imputing column =  icu_admit_source
Imputing column =  apache_3j_bodysystem
Imputing column =  apache_2_bodysystem
Finished imputing data
```

Final Data after Null Value removal

In [ ]:
```
data.isnull().sum()
```

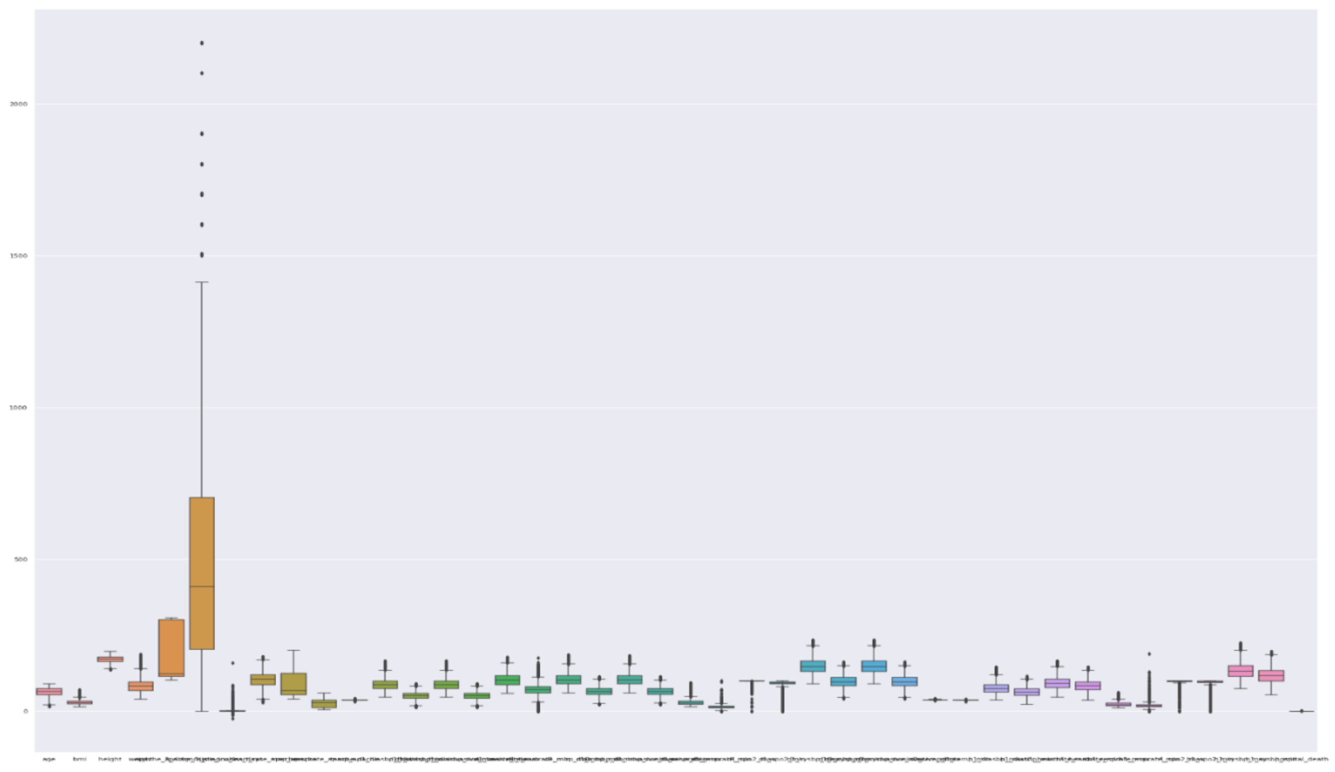
Out[ ]:
```
age                            0
bmi                            0
ethnicity                      0
gender                         0
height                         0
weight                         0
elective_surgery               0
icu_admit_source               0
icu_stay_type                  0
icu_type                       0
apache_post_operative_x        0
apache_2_diagnosis             0
apache_3j_diagnosis            0
pre_icu_los_days               0
arf_apache                     0
gcs_eyes_apache                0
gcs_motor_apache               0
gcs_unable_apache              0
gcs_verbal_apache              0
heart_rate_apache              0
intubated_apache               0
map_apache                     0
resprate_apache                0
temp_apache                    0
ventilated_apache              0
d1_diasbp_max                  0
d1_diasbp_min                  0
d1_diasbp_noninvasive_max      0
d1_diasbp_noninvasive_min      0
d1_heartrate_max               0
d1_heartrate_min               0
```

```
d1_mbp_max                      0
d1_mbp_min                      0
d1_mbp_noninvasive_max          0
d1_mbp_noninvasive_min          0
d1_resprate_max                 0
d1_resprate_min                 0
d1_spo2_max                     0
d1_spo2_min                     0
d1_sysbp_max                    0
d1_sysbp_min                    0
d1_sysbp_noninvasive_max        0
d1_sysbp_noninvasive_min        0
d1_temp_max                     0
d1_temp_min                     0
h1_diasbp_max                   0
h1_diasbp_min                   0
h1_heartrate_max                0
h1_heartrate_min                0
h1_resprate_max                 0
h1_resprate_min                 0
h1_spo2_max                     0
h1_spo2_min                     0
h1_sysbp_max                    0
h1_sysbp_min                    0
aids                            0
cirrhosis                       0
diabetes_mellitus               0
hepatic_failure                 0
immunosuppression               0
leukemia                        0
lymphoma                        0
solid_tumor_with_metastasis     0
apache_3j_bodysystem            0
apache_2_bodysystem             0
hospital_death                  0
dtype: int64
```

Outlier Treatment

Outlier - Box Plot

In [ ]:
```
sns.set(rc={'figure.figsize':(35, 30)})
sns.boxplot(data= data)
```
Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7ffb93200a60>
```



Remove Outliers - Threshold (99 percentile and 1 percentile)

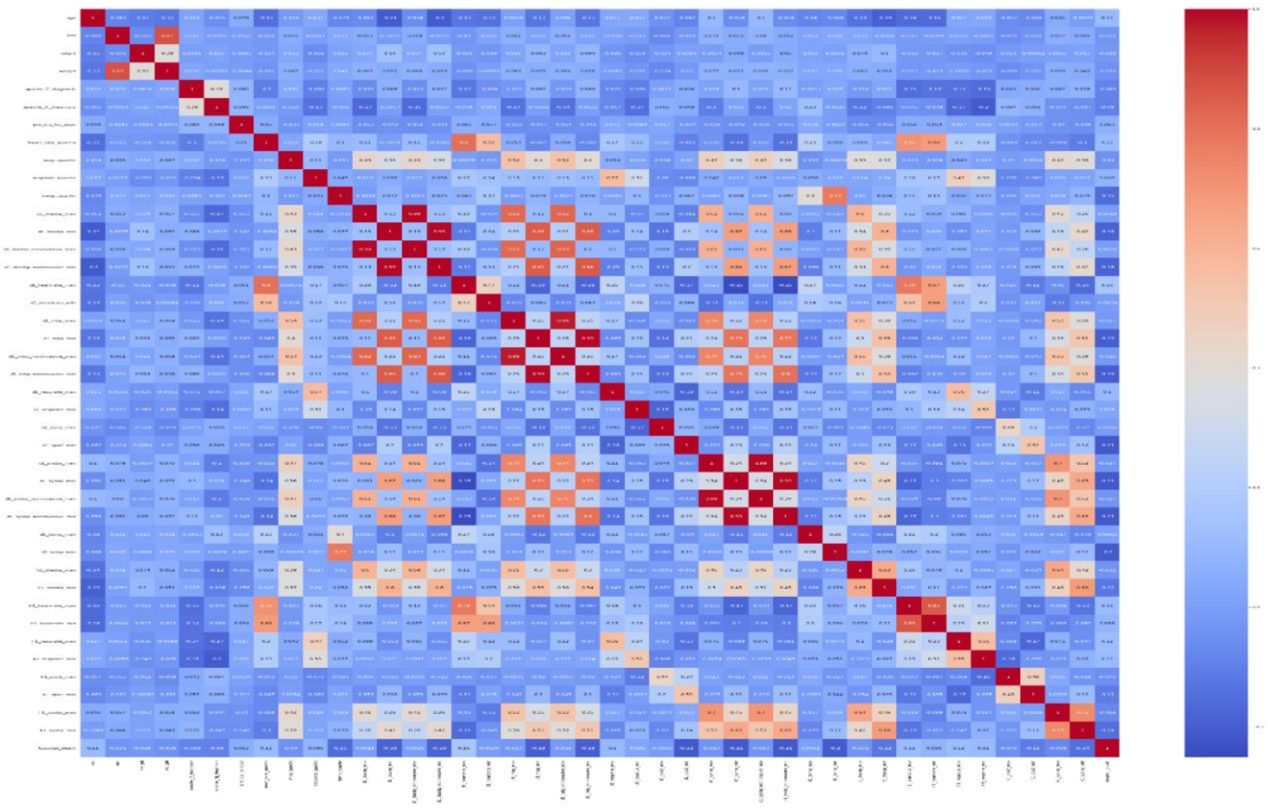
In [ ]:
```
for column in data.columns:
    if data[column].dtype != 'object':
        df1 = data.drop(data.loc[data[column] >
data[column].quantile(0.99)].index, inplace = False)

for column in data.columns:
    if data[column].dtype != 'object':
        df1 = data.drop(data.loc[data[column] <
data[column].quantile(0.01)].index, inplace = False)
```

Data Shape - 50% removal – Hence No outlier Treatment

Correlation Analysis

In [ ]:
```
plt.figure(figsize=(50,50))
sns.heatmap(data.corr(), annot = True, cmap= 'coolwarm')
```

Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7ffb9d5de040>
```



Pearson Correlation Matrix

Multi-collinearity

In [ ]:
```
corr_data = data.corr().unstack().sort_values().reset_index()
corr_data.rename(columns =  {"level_0" : 'column1', "level_1": 'column2',
                    0: 'corr_value'}, inplace = True)
corr_data[(corr_data['column1'] != corr_data['column2']) &
          (abs(corr_data['corr_value']) > 0.75)][::2]
```

Out[ ]:

	column1	column2	corr_value
1676	temp_apache	d1_temp_min	0.771412
1678	d1_heartrate_max	h1_heartrate_max	0.775461
1680	d1_mbp_min	d1_sysbp_noninvasive_min	0.789585
1682	d1_mbp_noninvasive_min	d1_sysbp_min	0.791863
1684	d1_mbp_min	d1_sysbp_min	0.793155
1686	d1_mbp_noninvasive_min	d1_sysbp_noninvasive_min	0.797673
1688	heart_rate_apache	d1_heartrate_max	0.801775
1690	d1_diasbp_noninvasive_max	d1_mbp_max	0.820365
1692	d1_diasbp_max	d1_mbp_max	0.822738
1694	d1_diasbp_max	d1_mbp_noninvasive_max	0.824858
1696	d1_diasbp_noninvasive_max	d1_mbp_noninvasive_max	0.831335

	column1	column2	corr_value
1698	d1_mbp_min	d1_diasbp_noninvasive_min	0.848739
1700	d1_mbp_noninvasive_min	d1_diasbp_min	0.849778
1702	d1_diasbp_min	d1_mbp_min	0.852638
1704	h1_heartrate_max	h1_heartrate_min	0.853792
1706	d1_diasbp_noninvasive_min	d1_mbp_noninvasive_min	0.855693
1708	bmi	weight	0.873688
1710	d1_mbp_max	d1_mbp_noninvasive_max	0.975136
1712	d1_mbp_noninvasive_min	d1_mbp_min	0.990407
1714	d1_diasbp_max	d1_diasbp_noninvasive_max	0.992332
1716	d1_sysbp_max	d1_sysbp_noninvasive_max	0.992371
1718	d1_sysbp_noninvasive_min	d1_sysbp_min	0.992402
1720	d1_diasbp_min	d1_diasbp_noninvasive_min	0.992998

Drop multi-collinear Attributes

In [ ]:
```
data.drop(["d1_diasbp_noninvasive_min", "d1_sysbp_noninvasive_min",
"d1_diasbp_max", "d1_mbp_noninvasive_min", "d1_mbp_min", "weight",
"h1_heartrate_max", "d1_sysbp_max"], axis= 1, inplace= True)
```

Converting dataset into X and Y

In [ ]:
```
x = data.iloc[:, :-1]
y = data["hospital_death"]
```

One-hot encoding

dummy variable creation for X

In [ ]:
```
x = pd.get_dummies(x, drop_first= False)
```

Principle Component Analysis

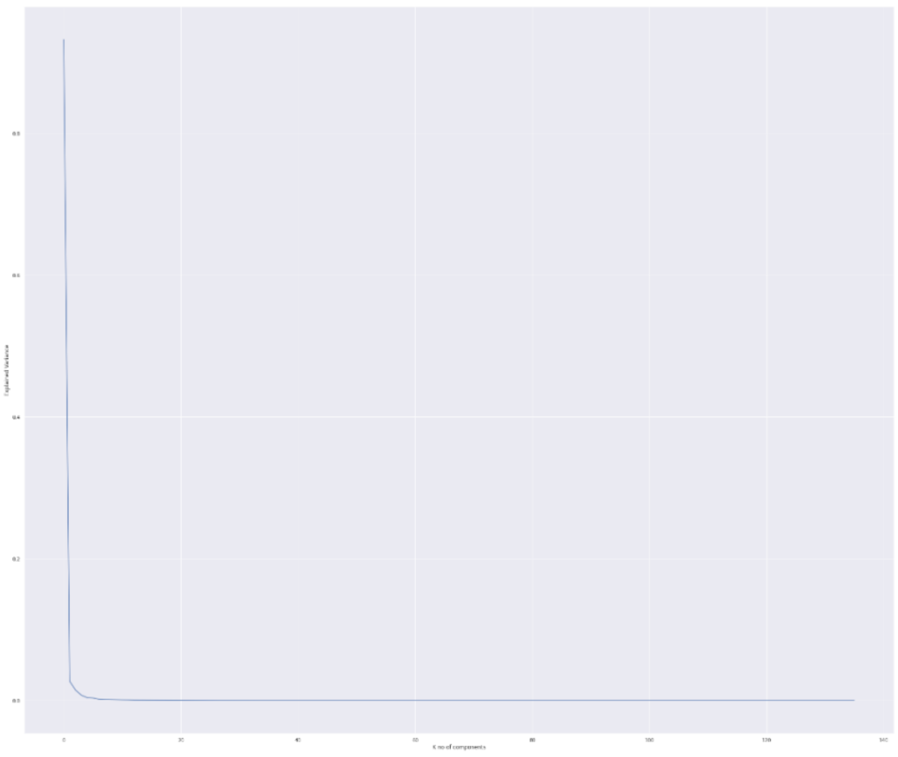
In [ ]:
```
print(f'\nThe original dataset has {x.shape[1]} features.')
```

Out [ ]:
```
The original dataset has 136 features.
```

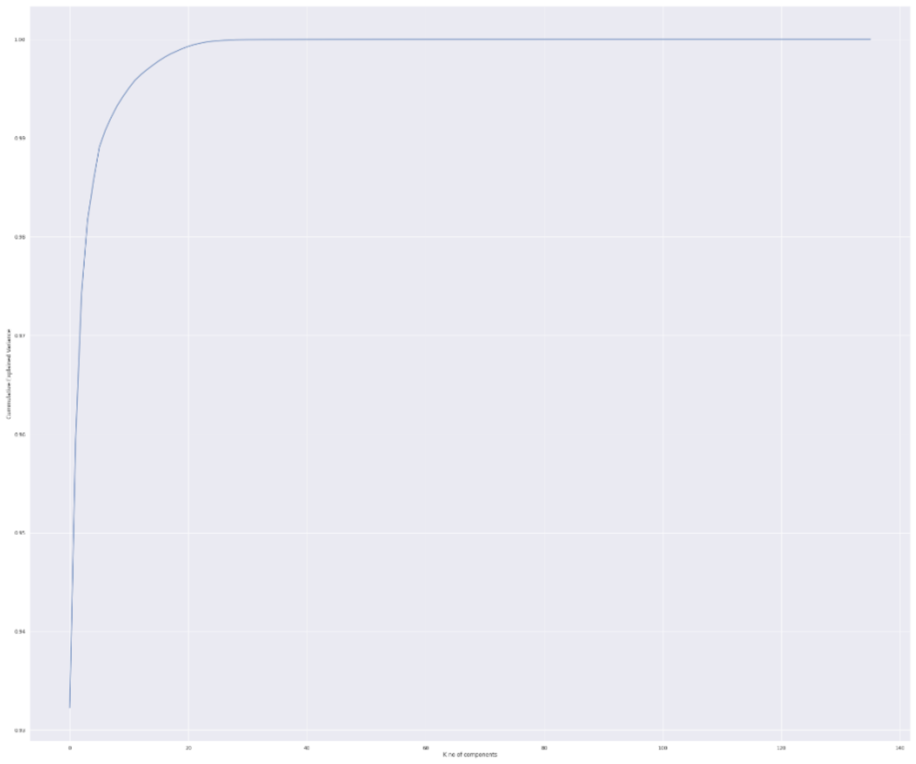
In [ ]:
```
Xn = scale(x)
pca_prep = PCA().fit(Xn)
pca_prep.n_components_
```

Out[ ]:
```
136
```

In [ ]:
```
# Find an "elbow" or an inflection point on the plot.
plt.plot(pca_prep.explained_variance_ratio_)
plt.xlabel("K no of components")
plt.ylabel("Explained Variance")
plt.grid(True)
plt.show()
```



In [ ]:
```
plt.plot(np.cumsum(pca_prep.explained_variance_ratio_))
plt.xlabel("K no of components")
plt.ylabel("Cummulative Explained Variance")
plt.grid(True)
plt.show()
```



Feature Importance

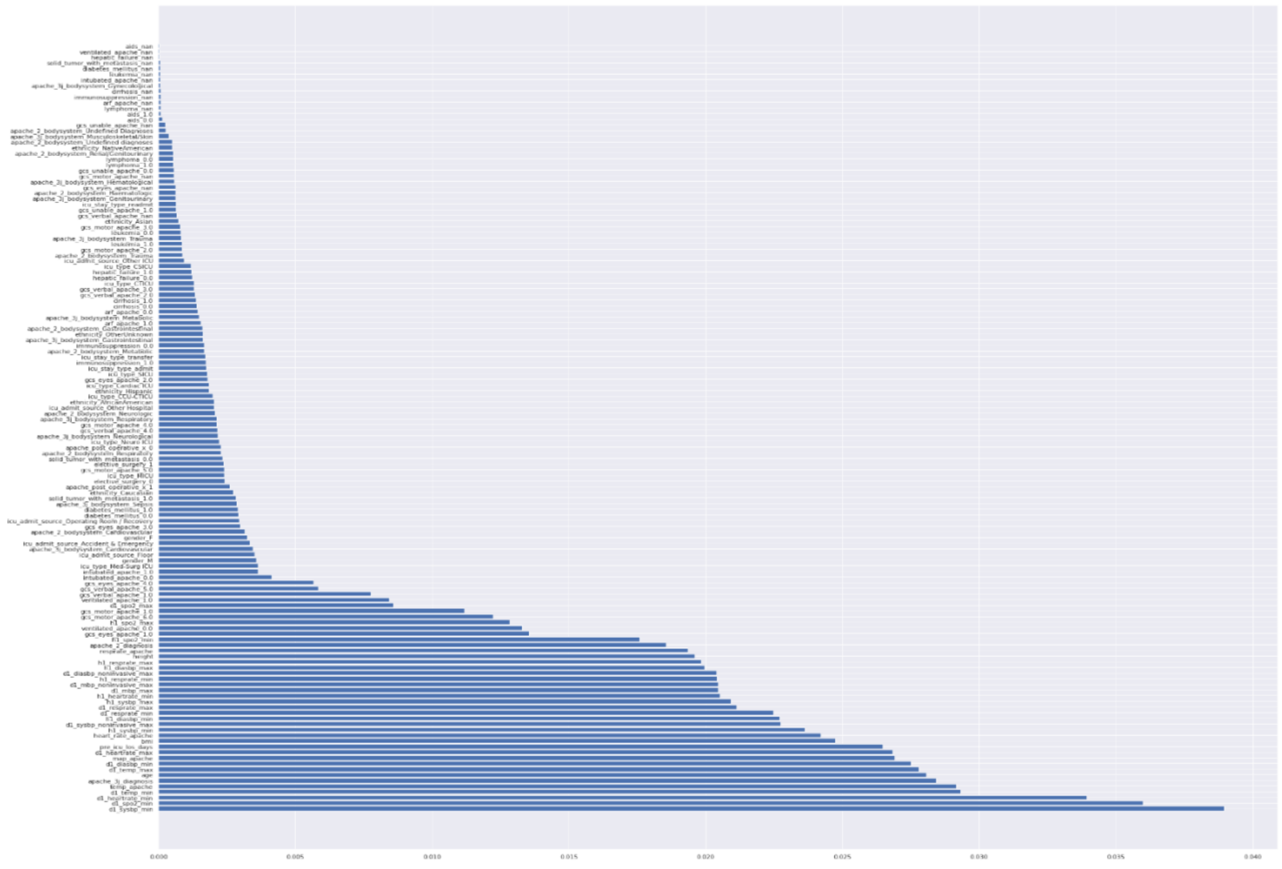
In [ ]:
```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size =.3,
stratify= y)

# Create and apply a model (object) for classification
rfm = RandomForestClassifier(random_state=123)
rfm.fit(x_train, y_train)
y_pred = rfm.predict(x_test)

# Build a confusion matrix and show the Classification Report
print(f'Confusion Matrix:\n {metrics.confusion_matrix(y_test,y_pred)}\n\n')
print(f'Classification Report for original imbalanced dataset:\n
{metrics.classification_report(y_test,y_pred)}')

importances = rfm.feature_importances_
feature_names = x.columns

# Draw a bar chart to see the sorted importance values with feature names.
df_importances = pd.DataFrame(data=importances, index=feature_names,
columns=['importance_value'])
df_importances.sort_values(by = 'importance_value', ascending=False,
inplace=True)
plt.barh(df_importances.index,df_importances.importance_value)
```

```
# Build a model with a subset of important features using mean as a
#threshold
selector = SelectFromModel(estimator= RandomForestClassifier(), threshold=
'mean')
x_reduced = selector.fit_transform(x, y)
print(f'\nThreshold mean value to be used for feature selection:
{selector.threshold_}')

#This shows how many features are selected and the list of selected
#features.
selected_features = selector.get_support()
print(f'\n {selected_features.sum()} features are selected.\n')

selected_features_names = []
for i,j in zip(selected_features, feature_names):
    if i: selected_features_names.append(j)
print(f'Selected Features:\n {selected_features_names}')

# Now, we are ready to build a model using those reduced number of
#features.
x_reduced_train, x_reduced_test, y_reduced_train, y_reduced_test =
train_test_split(x_reduced, y, test_size =.3, stratify=y)

# Build a model with the reduced number of features.
rfm_reduced = RandomForestClassifier().fit(x_reduced_train,
y_reduced_train)
y_reduced_pred = rfm_reduced.predict(x_reduced_test)

print(f'\n\nClassification Report for reduced imbalanced dataset:\n
{metrics.classification_report(y_reduced_test, y_reduced_pred)}')
```

Out [ ]:

```
Confusion Matrix:
 [[24988   151]
 [ 1908   467]]

Classification Report for original imbalanced dataset:
               precision    recall  f1-score   support

           0       0.93      0.99      0.96     25139
           1       0.76      0.20      0.31      2375

    accuracy                           0.93     27514
   macro avg       0.84      0.60      0.64     27514
weighted avg       0.91      0.93      0.90     27514


Threshold mean value to be used for feature selection: 0.00735294117647059
```

```
 40 features are selected.

Selected Features:
 ['age', 'bmi', 'height', 'apache_2_diagnosis', 'apache_3j_diagnosis',
'pre_icu_los_days', 'heart_rate_apache', 'map_apache', 'resprate_apache',
'temp_apache', 'd1_diasbp_min', 'd1_diasbp_noninvasive_max',
'd1_heartrate_max', 'd1_heartrate_min', 'd1_mbp_max',
'd1_mbp_noninvasive_max', 'd1_resprate_max', 'd1_resprate_min',
'd1_spo2_max', 'd1_spo2_min', 'd1_sysbp_min', 'd1_sysbp_noninvasive_max',
'd1_temp_max', 'd1_temp_min', 'h1_diasbp_max', 'h1_diasbp_min',
'h1_heartrate_min', 'h1_resprate_max', 'h1_resprate_min', 'h1_spo2_max',
'h1_spo2_min', 'h1_sysbp_max', 'h1_sysbp_min', 'gcs_eyes_apache_1.0',
'gcs_motor_apache_1.0', 'gcs_motor_apache_6.0', 'gcs_verbal_apache_1.0',
'gcs_verbal_apache_5.0', 'ventilated_apache_0.0', 'ventilated_apache_1.0']

Classification Report for reduced imbalanced dataset:
              precision    recall  f1-score   support

           0       0.93      0.99      0.96     25139
           1       0.72      0.22      0.34      2375

    accuracy                           0.93     27514
   macro avg       0.82      0.61      0.65     27514
weighted avg       0.91      0.93      0.91     27514
```



Create reduced dataset based on feature importance

In [ ]:
```
x_reduced = x[['age', 'bmi', 'height', 'apache_2_diagnosis',
               'apache_3j_diagnosis', 'pre_icu_los_days',
               'heart_rate_apache', 'map_apache', 'resprate_apache',
               'temp_apache', 'd1_diasbp_min','d1_diasbp_noninvasive_max',
               'd1_heartrate_max', 'd1_heartrate_min', 'd1_mbp_max',
               'd1_mbp_noninvasive_max', 'd1_resprate_max',
               'd1_resprate_min', 'd1_spo2_max', 'd1_spo2_min',
               'd1_sysbp_min', 'd1_sysbp_noninvasive_max', 'd1_temp_max',
               'd1_temp_min', 'h1_diasbp_max','h1_diasbp_min',
               'h1_heartrate_min', 'h1_resprate_max', 'h1_resprate_min',
               'h1_spo2_max', 'h1_spo2_min', 'h1_sysbp_max',
               'h1_sysbp_min', 'gcs_eyes_apache_1.0',
               'gcs_motor_apache_1.0', 'gcs_motor_apache_6.0',
               'gcs_verbal_apache_1.0', 'gcs_verbal_apache_5.0',
               'ventilated_apache_0.0', 'ventilated_apache_1.0']]
```

Data Balancing

Naive Random Over-Sampling

In [ ]:
```
ros = RandomOverSampler(random_state=0)
x_rs, y_rs = ros.fit_resample(x, y)
print(x_rs.shape)
y_rs.shape
```

Out[ ]:
```
(167596, 136)
(167596,)
```

In [ ]:
```
print(f'Oversampled Data: {np.unique(y_rs, return_counts= 1)}')
```

Out[ ]:
```
Oversampled Data: (array([0, 1]), array([83798, 83798]))
```

In [ ]:
```
ros = RandomOverSampler(random_state=0)
x_rs_reduced, y_rs_reduced = ros.fit_resample(x_reduced, y)
print(x_rs_reduced.shape)
y_rs_reduced.shape
```

Out[ ]:
```
(167596, 40)
(167596,)
```

In [ ]:
```
print(f'Oversampled Data: {np.unique(y_rs_reduced, return_counts= 1)}')
```

Out[ ]:
```
Oversampled Data: (array([0, 1]), array([83798, 83798]))
```

Synthetic Minority Over Sampling Technique (SMOTE)

In [ ]:
```
sm = SMOTE(random_state=0)
x_sm, y_sm = sm.fit_resample(x, y)
print(x_sm.shape)
y_sm.shape
```

Out[ ]:
```
(167596, 136)
(167596,)
```

In [ ]:
```
sm = SMOTE(random_state=0)
x_sm_reduced, y_sm_reduced = sm.fit_resample(x_reduced, y)
print(x_sm_reduced.shape)
y_sm_reduced.shape
```

Out[ ]:
```
(167596, 40)
(167596,)
```

Training and testing data for all models

In [ ]:
```
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size =.3, 
random_state=1234, stratify=y)

X_sm_train, X_sm_test, y_sm_train, y_sm_test = train_test_split(x_sm, y_sm, 
test_size =.3, random_state=1234, stratify=y_sm)

X_reduced_train, X_reduced_test, y_reduced_train, y_reduced_test = 
train_test_split(x_reduced, y, test_size =.3, random_state=1234, 
stratify=y)

X_smreduced_train, X_smreduced_test, y_smreduced_train, y_smreduced_test = 
train_test_split(x_sm_reduced, y_sm_reduced, test_size =.3, 
random_state=1234, stratify=y_sm_reduced)
```

Random Forest Classifier

Grid Search CV and Randomized Search CV

In [ ]:
```
### Grid Search CV
rfc = RandomForestClassifier()
param_grid = {
            'n_estimators': [100, 120, 150],
            'max_features': ['auto', 'sqrt', 'log2'],
            'criterion': ['gini', 'entropy'],
            'max_depth': [5, 10, 15]
        }

start = time.time()
cv_rfc = GridSearchCV(estimator= rfc, param_grid = param_grid, cv = 3)
cv_rfc.fit(X_smreduced_train, y_smreduced_train)
end = time.time()

print(f'The best estimator: {cv_rfc.best_estimator_}')
print(f'The best parameters: {cv_rfc.best_params_}')
print(f'The best score: {cv_rfc.best_score_:.4f}')
print(f'Total run time for GridsearchCV: {(end - start): .2f} seconds')

### Randomized Search CV
rfc = RandomForestClassifier()
param_grid = {
            'n_estimators': [100, 120, 150],
            'max_features': ['auto', 'sqrt', 'log2'],
            'criterion': ['gini', 'entropy'],
            'max_depth': [5, 10, 15]
}

start_time = time.time()
rand_src = RandomizedSearchCV(estimator= rfc, param_distributions=
param_grid, n_iter= 3)
rand_src.fit(X_smreduced_train, y_smreduced_train)
end_time = time.time()

print(f'The best estimator: {rand_src.best_estimator_}')
print(f'The best parameters:{rand_src.best_params_}')
print(f'The best score: {rand_src.best_score_: .4f}')
print(f'Total run time for RandomSearchCV: {(end_time -
start_time):.2f}seconds')
```

Out[ ]:

```
The best estimator: RandomForestClassifier(criterion='entropy',
max_depth=15, max_features='sqrt', n_estimators=120)
```

```
The best parameters: {'criterion': 'entropy', 'max_depth': 15,
                      'max_features': 'sqrt', 'n_estimators': 120}
The best score: 0.9239
Total run time for GridsearchCV:  5963.56 seconds
The best estimator: RandomForestClassifier(criterion='entropy',
max_depth=15, max_features='sqrt', n_estimators=120)
The best parameters:{'n_estimators': 120, 'max_features': 'sqrt',
                     'max_depth': 15, 'criterion': 'entropy'}
The best score:  0.9257
Total run time for RandomSearchCV: 901.48seconds
```

Radnom Forest Classifier Result Comparison

In [ ]:

```
rfc = RandomForestClassifier(n_estimators= 120, max_features= "sqrt",
max_depth= 15, criterion= 'entropy')

## Model with original unbalanced data
start = time.time()
rfc.fit(X_train, y_train)
y_pred = rfc.predict(X_test)
end = time.time()

cm = metrics.confusion_matrix(y_test, y_pred)
print('\nConfusion Matrix using the classifier using the Original
Unbalanced Data\n', cm)
print('\nClassification Report\n')
print(metrics.classification_report(y_test, y_pred))
print(f'Total run time for model: {(end - start): .2f} seconds')

## Model with original balanced data
start = time.time()
rfc.fit(X_sm_train, y_sm_train)
y_pred_sm = rfc.predict(X_test)
end = time.time()

cm_sm = metrics.confusion_matrix(y_test, y_pred_sm)
print('\nConfusion Matrix using the classifier using the Original Balanced
Data\n',cm_sm)
print('\nClassification Report\n')
print(metrics.classification_report(y_test, y_pred_sm))
print(f'Total run time for model: {(end - start): .2f} seconds')

## Model with reduced unbalanced data
start = time.time()
rfc.fit(X_reduced_train, y_reduced_train)
y_reduced_pred = rfc.predict(X_reduced_test)
end = time.time()
```

```
cm_reduced = metrics.confusion_matrix(y_reduced_test, y_reduced_pred)
print('\nConfusion Matrix using the classifier using the Reduced unbalanced
Data\n',cm_reduced)
print('\nClassification Report\n')
print(metrics.classification_report(y_reduced_test, y_reduced_pred))
print(f'Total run time for model: {(end - start): .2f} seconds')

## Model with reduced balanced data
start = time.time()
rfc.fit(X_smreduced_train, y_smreduced_train)
y_smreduced_pred = rfc.predict(X_reduced_test)
end = time.time()

cm_sm_reduced = metrics.confusion_matrix(y_reduced_test, y_smreduced_pred)
print('\nConfusion Matrix using the classifier using the reduced balanced
Data\n',cm_sm_reduced)
print('\nClassification Report\n')
print(metrics.classification_report(y_reduced_test, y_smreduced_pred))
print(f'Total run time for model: {(end - start): .2f} seconds')
```

Out[ ]:

```
Confusion Matrix using the classifier using the Original Unbalanced Data
 [[25014   125]
 [ 1931   444]]

Classification Report

              precision    recall  f1-score   support

           0       0.93      1.00      0.96     25139
           1       0.78      0.19      0.30      2375

    accuracy                           0.93     27514
   macro avg       0.85      0.59      0.63     27514
weighted avg       0.92      0.93      0.90     27514

Total run time for model:  20.92 seconds

Confusion Matrix using the classifier using the Original Balanced Data
 [[24299   840]
 [  727  1648]]

Classification Report

              precision    recall  f1-score   support

           0       0.97      0.97      0.97     25139
           1       0.66      0.69      0.68      2375

    accuracy                           0.94     27514
   macro avg       0.82      0.83      0.82     27514
weighted avg       0.94      0.94      0.94     27514

Total run time for model:  64.91 seconds

Confusion Matrix using the classifier using the Reduced unbalanced Data
 [[25003   136]
 [ 1886   489]]

Classification Report

              precision    recall  f1-score   support

           0       0.93      0.99      0.96     25139
           1       0.78      0.21      0.33      2375

    accuracy                           0.93     27514
   macro avg       0.86      0.60      0.64     27514
weighted avg       0.92      0.93      0.91     27514

Total run time for model:  25.03 seconds

Confusion Matrix using the classifier using the reduced balanced Data
 [[23188  1951]
 [  439  1936]]

Classification Report

              precision    recall  f1-score   support

           0       0.98      0.92      0.95     25139
           1       0.50      0.82      0.62      2375

    accuracy                           0.91     27514
   macro avg       0.74      0.87      0.78     27514
weighted avg       0.94      0.91      0.92     27514

Total run time for model:  89.85 seconds
```

K fold Cross Validation

In [ ]:

```
rfc_mean_score = np.mean(cross_val_score(rfc, x_sm_reduced, y_sm_reduced, 
cv=5))
rfc_mean_score
```

XG Boost

Grid search CV

In [ ]:
```
fit_params_of_xgb = {
    "early_stopping_rounds":100,
    "eval_metric" : 'auc',
    "eval_set" : [(X_smreduced_test, y_smreduced_test)],
    'verbose': 100,
}

# A parameter grid for XGBoost
params = {
    'booster': ["gbtree"],
    'learning_rate': [0.1],
    'n_estimators': range(100, 500, 100),
    'min_child_weight': [1],
    'gamma': [0],
    'subsample': [0.8],
    'colsample_bytree': [0.8],
    'max_depth': [5],
    "scale_pos_weight": [1]
}

start_time = time.time()
xgb_estimator = XGBClassifier(objective='binary:logistic')
gsearch = GridSearchCV(
    estimator=xgb_estimator,
    param_grid=params,
    scoring='roc_auc',
    n_jobs=-1,
    cv=3)

xgb_model = gsearch.fit(X_smreduced_train, y_smreduced_train,
**fit_params_of_xgb)
end_time = time.time()

print(f'The best estimator: {gsearch.best_estimator_}')
print(f'The best parameters:{gsearch.best_params_}')
print(f'The best score: {gsearch.best_score_: .4f}')
print(f'Total run time for GridSearchCV: {(end_time -
start_time):.2f}seconds')
```

Out[ ]:

```
[0]     validation_0-auc:0.909095
Will train until validation_0-auc hasn't improved in 100 rounds.
[100]   validation_0-auc:0.985806
```

```
[200]   validation_0-auc:0.988507
[300]   validation_0-auc:0.989265
[399]   validation_0-auc:0.989503
The best estimator: XGBClassifier(colsample_bytree=0.8, max_depth=5,
                                  n_estimators=400, subsample=0.8)
The best parameters:{'booster': 'gbtree', 'colsample_bytree': 0.8,
                     'gamma': 0, 'learning_rate': 0.1, 'max_depth': 5,
                     'min_child_weight': 1, 'n_estimators': 400,
                     'scale_pos_weight': 1, 'subsample': 0.8}
The best score:  0.9888
Total run time for GridSearchCV: 831.22seconds
```

XGBoost Classifier: Result comparison

In [ ]:
```
xgb_tuned = XGBClassifier(n_estimators=400,
    objective='binary:logistic',
    booster="gbtree",
    learning_rate=0.1,
    scale_pos_weight=1,
    max_depth=5,
    min_child_weight=1,
    gamma=0,
    subsample=0.8,
    colsample_bytree=0.8,
    n_jobs=-1)

## Model with original unbalanced data
start_time = time.time()
xgb_tuned.fit(X_train._get_numeric_data(), np.ravel(y_train, order='C'))
y_pred = xgb_tuned.predict(X_test._get_numeric_data())
end_time = time.time()

cm = metrics.confusion_matrix(y_test, y_pred)
print('\nConfusion Matrix using the classifier using the original
unbalanced Data\n',cm)
print('\nClassification Report\n')
print(metrics.classification_report(y_test, y_pred))
print(f'Total run time for model: {(end_time - start_time):.2f}seconds')

## Model with original balanced data
start_time = time.time()
xgb_tuned.fit(X_sm_train._get_numeric_data(), np.ravel(y_sm_train,
order='C'))
y_pred_sm = xgb_tuned.predict(X_test._get_numeric_data())
end_time = time.time()

cm_sm = metrics.confusion_matrix(y_test, y_pred_sm)
```

```
print('\nConfusion Matrix using the classifier using the original balanced
Data\n', cm_sm)
print('\nClassification Report\n')
print(metrics.classification_report(y_test, y_pred_sm))
print(f'Total run time for model: {(end_time - start_time):.2f}seconds')

## Model with reduced unbalanced data
start_time = time.time()
xgb_tuned.fit(X_reduced_train._get_numeric_data(),
np.ravel(y_reduced_train, order='C'))
y_pred_reduced = xgb_tuned.predict(X_reduced_test._get_numeric_data())
end_time = time.time()

cm_reduced = metrics.confusion_matrix(y_reduced_test, y_pred_reduced)
print('\nConfusion Matrix using the classifier using the reduced unbalanced
Data\n', cm_reduced)
print('\nClassification Report\n')
print(metrics.classification_report(y_reduced_test, y_pred_reduced))
print(f'Total run time for model: {(end_time - start_time):.2f}seconds')

## Model with reduced balanced data
start_time = time.time()
xgb_tuned.fit(X_smreduced_train._get_numeric_data(),
np.ravel(y_smreduced_train, order='C'))
y_pred_smreduced = xgb_tuned.predict(X_reduced_test._get_numeric_data())
end_time = time.time()

cm_smreduced = metrics.confusion_matrix(y_reduced_test, y_pred_smreduced)
print('\nConfusion Matrix using the classifier using the reduced balanced
Data\n', cm_smreduced)
print('\nClassification Report\n')
print(metrics.classification_report(y_reduced_test, y_pred_smreduced))
print(f'Total run time for model: {(end_time - start_time):.2f}seconds')
```

Out[ ]:

```
Confusion Matrix using the classifier using the original unbalanced Data
 [[24821   318]
 [ 1630   745]]

Classification Report

              precision    recall  f1-score   support

           0       0.94      0.99      0.96     25139
           1       0.70      0.31      0.43      2375

    accuracy                           0.93     27514
   macro avg       0.82      0.65      0.70     27514
```

```
weighted avg       0.92      0.93      0.92     27514

Total run time for model: 131.05seconds

Confusion Matrix using the classifier using the original balanced Data
 [[24770   369]
 [ 1348  1027]]

Classification Report

              precision    recall  f1-score   support

           0       0.95      0.99      0.97     25139
           1       0.74      0.43      0.54      2375

    accuracy                           0.94     27514
   macro avg       0.84      0.71      0.76     27514
weighted avg       0.93      0.94      0.93     27514

Total run time for model: 305.19seconds

Confusion Matrix using the classifier using the reduced unbalanced Data
 [[24843   296]
 [ 1640   735]]

Classification Report

              precision    recall  f1-score   support

           0       0.94      0.99      0.96     25139
           1       0.71      0.31      0.43      2375

    accuracy                           0.93     27514
   macro avg       0.83      0.65      0.70     27514
weighted avg       0.92      0.93      0.92     27514

Total run time for model: 55.01seconds

Confusion Matrix using the classifier using the reduced balanced Data
 [[24672   467]
 [ 1353  1022]]

Classification Report

              precision    recall  f1-score   support

           0       0.95      0.98      0.96     25139
           1       0.69      0.43      0.53      2375
    accuracy                           0.93     27514
   macro avg       0.82      0.71      0.75     27514
weighted avg       0.93      0.93      0.93     27514

Total run time for model: 151.26seconds
```

K Fold Cross Validation

In [ ]:
```
xgb_mean_score = np.mean(cross_val_score(xgb_tuned, x_sm_reduced,
y_sm_reduced, cv=5))
xgb_mean_score
```

Out[ ]:
```
0.938370667802773
```

Gradient Boosting

Grid Search and Randomized Search CV

In [ ]:
```
param_grid = {
            'n_estimators': [100, 120, 150],
            'max_features': ['auto', 'sqrt', 'log2'],
            'max_depth': [5, 10, 15]
        }

gbc = GradientBoostingClassifier()

## Grid Search CV
start = time.time()
cv_gbc = GridSearchCV(estimator= gbc, param_grid = param_grid, cv = 3)
cv_gbc.fit(X_smreduced_train, y_smreduced_train)
end = time.time()

print(f'The best estimator: {cv_gbc.best_estimator_}')
print(f'The best parameters: {cv_gbc.best_params_}')
print(f'The best score: {cv_gbc.best_score_:.4f}')
print(f'Total run time for GridsearchCV: {(end - start): .2f} seconds')

### Randomized Search CV
start = time.time()
rand_src = RandomizedSearchCV(estimator= gbc, param_distributions=
param_grid, n_iter= 3)
rand_src.fit(X_smreduced_train, y_smreduced_train)
end_time = time.time()

print(f'The best estimator: {rand_src.best_estimator_}')
print(f'The best parameters:{rand_src.best_params_}')
print(f'The best score: {rand_src.best_score_: .4f}')
print(f'Total run time for RandomSearchCV: {(end_time -
start_time):.2f}seconds')
```

Out[ ]:

```
The best estimator: GradientBoostingClassifier(max_depth=15,
max_features='auto', n_estimators=150)
The best parameters: {'max_depth': 15, 'max_features': 'auto'
                       'n_estimators': 150}
The best score: 0.9591
Total run time for GridsearchCV:  13855.86 seconds
The best estimator: GradientBoostingClassifier(max_depth=15,
                                               max_features='log2',
                                                n_estimators=150)
The best parameters:{'n_estimators': 150, 'max_features': 'log2',
                       'max_depth': 15}
The best score:  0.9594
Total run time for RandomSearchCV: 17679.10seconds
```

Gradient Boosting Classifier Result Comparison

In [ ]:

```
gbc = GradientBoostingClassifier(max_depth=15, max_features='auto',
n_estimators=150)

##Original Unbalanced
start = time.time()
gbc.fit(X_train, y_train)
y_pred = gbc.predict(X_test)
end = time.time()

print('Gradient Boosting original unbalanced data')
print(metrics.classification_report(y_test, y_pred))
print(f'Total run time for model: {(end - start): .2f} seconds')

##Original Balanced
start = time.time()
gbc.fit(X_sm_train, y_sm_train)
y_sm_pred = gbc.predict(X_test)
end = time.time()

print('Gradient Boosting Original Balanced Dataset')
print(metrics.classification_report(y_test, y_sm_pred))
print(f'Total run time for model: {(end - start): .2f} seconds')

## Reduced Unbalanced
start = time.time()
```

```
gbc.fit(X_reduced_train, y_reduced_train)
y_reduced_pred = gbc.predict(X_reduced_test)
end = time.time()

print('Gradient Boosting Reduced Unbalanced Dataset')
print(metrics.classification_report(y_reduced_test, y_reduced_pred))
print(f'Total run time for model: {(end - start): .2f} seconds')

##Balanced Reduced
start = time.time()
gbc.fit(X_smreduced_train, y_smreduced_train)
y_smreduced_pred = gbc.predict(X_reduced_test)
end = time.time()

print('Gradient Boosting Reduced Balanced Dataset')
print(metrics.classification_report(y_reduced_test, y_smreduced_pred))
print(f'Total run time for model: {(end - start): .2f} seconds')
```

Out[ ]:

```
Gradient Boosting original unbalanced data
              precision    recall  f1-score   support

           0       0.93      0.99      0.96     25139
           1       0.73      0.24      0.37      2375

    accuracy                           0.93     27514
   macro avg       0.83      0.62      0.66     27514
weighted avg       0.91      0.93      0.91     27514

Total run time for model:  402.58 seconds

Gradient Boosting Original Balanced Dataset
              precision    recall  f1-score   support

           0       0.98      0.99      0.98     25139
           1       0.84      0.79      0.82      2375

    accuracy                           0.97     27514
   macro avg       0.91      0.89      0.90     27514
weighted avg       0.97      0.97      0.97     27514

Total run time for model:  1074.78 seconds

Gradient Boosting Reduced Unbalanced Dataset
              precision    recall  f1-score   support

           0       0.93      0.99      0.96     25139
           1       0.71      0.24      0.36      2375
    accuracy                           0.93     27514
   macro avg       0.82      0.62      0.66     27514
weighted avg       0.91      0.93      0.91     27514

Total run time for model:  300.08 seconds

Gradient Boosting Reduced Balanced Dataset
              precision    recall  f1-score   support

           0       0.98      0.98      0.98     25139
           1       0.81      0.82      0.81      2375

    accuracy                           0.97     27514
   macro avg       0.89      0.90      0.90     27514
weighted avg       0.97      0.97      0.97     27514

Total run time for model:  878.25 seconds
```

K fold Cross Validation

In [ ]:
```
gbc_mean_score = np.mean(cross_val_score(gbc, x_sm_reduced, y_sm_reduced, 
cv=5))
gbc_mean_score
```

Decision Tree

Grid Search CV and Randomized Search CV

In [ ]:
```
dtm = DecisionTreeClassifier()

params = {'criterion': ['gini','entropy'],
                       'max_depth' : range(1,10),
                       "min_samples_leaf": range(1,5),
                       'min_samples_split' : range(1,10)
            }

## Grid Search CV
start_time = time.time()
Grid_dtm = GridSearchCV(cv=5, estimator= dtm, n_jobs=-1, param_grid = 
params, verbose=1)
Grid_dtm.fit(X_smreduced_train, y_smreduced_train)
end_time = time.time()

print(f'The best estimator: {Grid_dtm.best_estimator_}')
print(f'The best parameters:{Grid_dtm.best_params_}')
print(f'The best score: {Grid_dtm.best_score_: .4f}')
print(f'Total run time for GridSearchCV: {(end_time -
start_time):.2f}seconds')

## Randomized Search CV
start_time = time.time()
rand_dtm = RandomizedSearchCV(cv=5, estimator= dtm, n_jobs=-1,
param_distributions = params, verbose=1)
rand_dtm.fit(X_sm_train, y_sm_train)
end_time = time.time()

print(f'The best estimator: {rand_dtm.best_estimator_}')
print(f'The best parameters:{rand_dtm.best_params_}')
print(f'The best score: {rand_dtm.best_score_: .4f}')
print(f'Total run time for RandomSearchCV: {(end_time -
start_time):.2f}seconds')
```

Out[ ]:
```
The best estimator: DecisionTreeClassifier(max_depth=9)
The best parameters:{'criterion': 'gini', 'max_depth': 9,
                     'min_samples_leaf': 1, 'min_samples_split': 2}
The best score:  0.8716
Total run time for GridSearchCV: 4444.91seconds
The best estimator: DecisionTreeClassifier(criterion='entropy',
                                           max_depth=9, min_samples_leaf=3,
                                           min_samples_split=6)
The best parameters:{'min_samples_split': 6, 'min_samples_leaf': 3,
                     'max_depth': 9, 'criterion': 'entropy'}
The best score:  0.9100
Total run time for RandomSearchCV: 84.24seconds
```

Decision Tree Classifier Result Comparison

In [ ]:
```
from sklearn.tree import DecisionTreeClassifier

dtm = DecisionTreeClassifier(class_weight=None, criterion='entropy',
max_depth=9, max_features=None, max_leaf_nodes=None, min_samples_leaf=3,
                             min_samples_split=6,
min_weight_fraction_leaf=0.0, random_state= 1234, splitter='best')

## Original Unbalanced Dataset
start_time = time.time()
dtm.fit(X_train, y_train)
y_pred = dtm.predict(X_test)
end_time = time.time()

cm = metrics.confusion_matrix(y_test, y_pred)
print('\nConfusion Matrix using the classifier using the Original
Data\n',cm)
```

```
print('\nClassification Report\n')
print(metrics.classification_report(y_test,y_pred))
print(f'Total run time for model: {(end_time - start_time):.2f}seconds')

## Original Balanced Dataset
start_time = time.time()
dtm.fit(X_sm_train, y_sm_train)
y_pred_sm = dtm.predict(X_test)
end_time = time.time()

cm_sm = metrics.confusion_matrix(y_test, y_pred_sm)
print('\nConfusion Matrix using the classifier using the Original
Data\n',cm_sm)
print('\nClassification Report\n')
print(metrics.classification_report(y_test, y_pred_sm))
print(f'Total run time for model: {(end_time - start_time):.2f}seconds')

## Reduced Unbalanced Dataset
start_time = time.time()
dtm.fit(X_reduced_train, y_reduced_train)
y_reduced_pred = dtm.predict(X_reduced_test)
end_time = time.time()

cm_reduced = metrics.confusion_matrix(y_reduced_test, y_reduced_pred)
print('\nConfusion Matrix using the classifier using the reduced Data\n',
cm_reduced)
print('\nClassification Report\n')
print(metrics.classification_report(y_reduced_test, y_reduced_pred))
print(f'Total run time for model: {(end_time - start_time):.2f}seconds')

## Reduced Balanced Dataset
start_time = time.time()
dtm.fit(X_smreduced_train, y_smreduced_train)
y_smreduced_pred = dtm.predict(X_reduced_test)
end_time = time.time()

cm_sm_reduced = metrics.confusion_matrix(y_reduced_test, y_smreduced_pred)
print('\nConfusion Matrix using the classifier using the reduced balance
Data\n',cm_sm_reduced,'\n')
print('\nClassification Report\n')
print(metrics.classification_report(y_reduced_test, y_smreduced_pred))
print(f'Total run time for model: {(end_time - start_time):.2f}seconds')
```

Out[ ]:

```
Confusion Matrix using the classifier using the Original Data
 [[24841   298]
 [ 1970   405]]
```

```
Classification Report

              precision    recall  f1-score   support

           0       0.93      0.99      0.96     25139
           1       0.58      0.17      0.26      2375

    accuracy                           0.92     27514
   macro avg       0.75      0.58      0.61     27514
weighted avg       0.90      0.92      0.90     27514

Total run time for model: 3.99seconds

Confusion Matrix using the classifier using the Original Data
 [[24180   959]
 [ 1546   829]]

Classification Report

              precision    recall  f1-score   support

           0       0.94      0.96      0.95     25139
           1       0.46      0.35      0.40      2375

    accuracy                           0.91     27514
   macro avg       0.70      0.66      0.67     27514
weighted avg       0.90      0.91      0.90     27514

Total run time for model: 5.78seconds

Confusion Matrix using the classifier using the reduced Data
 [[24818   321]
 [ 1973   402]]

Classification Report

              precision    recall  f1-score   support

           0       0.93      0.99      0.96     25139
           1       0.56      0.17      0.26      2375

    accuracy                           0.92     27514
   macro avg       0.74      0.58      0.61     27514
weighted avg       0.89      0.92      0.90     27514

Total run time for model: 1.32seconds

Confusion Matrix using the classifier using the reduced balance Data
 [[22271  2868]
 [ 1072  1303]]

Classification Report

              precision    recall  f1-score   support

           0       0.95      0.89      0.92     25139
           1       0.31      0.55      0.40      2375

    accuracy                           0.86     27514
   macro avg       0.63      0.72      0.66     27514
weighted avg       0.90      0.86      0.87     27514

Total run time for model: 4.57seconds
```

K fold cross Validation

In [ ]:
```
dtm_mean_score = np.mean(cross_val_score(dtm, x_sm_reduced, y_sm_reduced, 
cv=5))
dtm_mean_score
```

Out[ ]:
```
0.8554921956421111
```

Scaling training and testing datasets

In [ ]:
```
scaler = StandardScaler()
scaler.fit(X_train)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)

scaler.fit(X_sm_train)
X_sm_train = scaler.transform(X_sm_train)
X_sm_test = scaler.transform(X_sm_test)

scaler.fit(X_reduced_train)
X_reduced_train = scaler.transform(X_reduced_train)
X_reduced_test = scaler.transform(X_reduced_test)

scaler.fit(X_smreduced_train)
X_smreduced_train = scaler.transform(X_smreduced_train)
X_smreduced_test = scaler.transform(X_smreduced_test)
```

K-Nearest Neighbours (KNN)

In [ ]:
```
##KNN model, k=350
classifier = KNeighborsClassifier(n_neighbors=350)

##Original Unbalanced
start = time.time()
classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)
end = time.time()

print(metrics.confusion_matrix(y_test, y_pred))
print(metrics.classification_report(y_test, y_pred))
print("Accuracy of Original Unbalanced
model:",metrics.accuracy_score(y_test,y_pred))
print(f'Total run time for model: {(end - start):.2f}seconds')

##Original Balanced
start = time.time()
classifier.fit(X_sm_train, y_sm_train)
y_sm_pred = classifier.predict(X_test)
end = time.time()

print(metrics.confusion_matrix(y_test, y_sm_pred))
print(metrics.classification_report(y_test, y_sm_pred))
print("Accuracy of Original Balanced
model:",metrics.accuracy_score(y_test,y_sm_pred))
print(f'Total run time for model: {(end - start): .2f} seconds')

## Reduced Unbalanced
start = time.time()
classifier.fit(X_reduced_train, y_reduced_train)
y_reduced_pred = classifier.predict(X_reduced_test)
end = time.time()

print(metrics.confusion_matrix(y_reduced_test, y_reduced_pred))
print(metrics.classification_report(y_reduced_test, y_reduced_pred))
print("Accuracy of Reduced Unbalanced
model:",metrics.accuracy_score(y_reduced_test, y_reduced_pred))
print(f'Total run time for model: {(end - start): .2f} seconds')

##Balanced Reduced
start = time.time()
classifier.fit(X_smreduced_train, y_smreduced_train)
y_smreduced_pred = classifier.predict(X_reduced_test)
end = time.time()

print(metrics.confusion_matrix(y_reduced_test, y_smreduced_pred))
print(metrics.classification_report(y_reduced_test, y_smreduced_pred))
print("Accuracy of Reduced Balanced
model:",metrics.accuracy_score(y_reduced_test, y_smreduced_pred))
print(f'Total run time for model: {(end - start): .2f} seconds')
```

Out[ ]:

```
[[25084    55]
 [ 2240   135]]
              precision    recall  f1-score   support

           0       0.92      1.00      0.96     25139
           1       0.71      0.06      0.11      2375

    accuracy                           0.92     27514
   macro avg       0.81      0.53      0.53     27514
weighted avg       0.90      0.92      0.88     27514

Accuracy of Original Unbalanced model: 0.9165879188776623
Total run time for model: 52.14seconds

[[18453  6686]
 [  547  1828]]
              precision    recall  f1-score   support

           0       0.97      0.73      0.84     25139
           1       0.21      0.77      0.34      2375

    accuracy                           0.74     27514
   macro avg       0.59      0.75      0.59     27514
weighted avg       0.91      0.74      0.79     27514

Accuracy of Original Balanced model: 0.7371156502144363
Total run time for model:  74.99 seconds

[[25088    51]
 [ 2228   147]]
              precision    recall  f1-score   support

           0       0.92      1.00      0.96     25139
           1       0.74      0.06      0.11      2375

    accuracy                           0.92     27514
   macro avg       0.83      0.53      0.54     27514
weighted avg       0.90      0.92      0.88     27514

Accuracy of Reduced Unbalanced model: 0.9171694410118485
Total run time for model:  42.74 seconds
```

```
[[16318  8821]
 [  337  2038]]
              precision    recall  f1-score   support

           0       0.98      0.65      0.78     25139
           1       0.19      0.86      0.31      2375

    accuracy                           0.67     27514
   macro avg       0.58      0.75      0.54     27514
weighted avg       0.91      0.67      0.74     27514

Accuracy of Reduced Balanced model: 0.6671512684451552
Total run time for model:  72.02 seconds
```

K-Fold Cross Validation

In [ ]:
```
knn_mean_score = np.mean(cross_val_score(classifier, x_sm_reduced,
y_sm_reduced, cv=5))
knn_mean_score
```

Out[ ]:
```
0.7110074076534514
```

Neural Network

Grid Search CV and Randomized Search CV

In [ ]:
```
mlp = MLPClassifier()

param_grid = {
    'hidden_layer_sizes': [(100,70,50), (90,80,40), (75,50,25)],
    'max_iter': [50, 75, 100],
    'activation': ['logistic','tanh', 'relu'],
    'solver': ['sgd', 'adam'],
    'alpha': [0.0001, 0.05],
    'learning_rate': ['constant','adaptive'],
}

### Grid Search CV
start = time.time()
grid = GridSearchCV(mlp, param_grid, n_jobs= -1, cv=5)
grid.fit(X_smreduced_train, y_smreduced_train)
end = time.time()

print(f'The best estimator: {grid.best_activation_}')
print(f'The best parameters: {grid.best_params_}')
print(f'The best score: {grid.best_score_:.4f}')
print(f'Total run time for GridsearchCV: {(end - start): .2f} seconds')

### Randomized Search CV
start = time.time()
random = RandomizedSearchCV(cv=5, estimator= mlp, n_jobs=-1,
param_distributions = param_grid, verbose=1)
random.fit(X_smreduced_train, y_smreduced_train)
end = time.time()

print(f'The best estimator: {random.best_activation_}')
print(f'The best parameters: {random.best_params_}')
print(f'The best score: {random.best_score_:.4f}')
print(f'Total run time for RandomizedseachCV: {(end - start): .2f}
seconds')
```

Neural Network Result Comparison

In [ ]:

```
mlp = MLPClassifier(hidden_layer_sizes=(90, 80, 40), activation='relu',
max_iter=100)

## original unbalanced dataset
start = time.time()
mlp.fit(X_train, y_train)
y_pred = mlp.predict(X_test)
end = time.time()

cm = metrics.confusion_matrix(y_test, y_pred)
print('\nConfusion Matrix using the classifier using the Original
Unbalanced Data\n',cm)
print('\nClassification Report\n')
print(metrics.classification_report(y_test, y_pred))
print(f'Total run time for model: {(end - start): .2f} seconds')

## original balanced dataset
start = time.time()
mlp.fit(X_sm_train, y_sm_train)
y_sm_pred = mlp.predict(X_test)
end = time.time()

cm_sm = metrics.confusion_matrix(y_test, y_sm_pred)
print('\nConfusion Matrix using the classifier using the Original
Unbalanced Data\n',cm_sm)
print('\nClassification Report\n')
print(metrics.classification_report(y_test, y_sm_pred))
print(f'Total run time for model: {(end - start): .2f} seconds')

## Reduced Unbalanced
```

```
start = time.time()
mlp.fit(X_reduced_train, y_reduced_train)
y_reduced_pred = mlp.predict(X_reduced_test)
end = time.time()

print(metrics.confusion_matrix(y_reduced_test, y_reduced_pred))
print(metrics.classification_report(y_reduced_test, y_reduced_pred))
print("Accuracy of Reduced Unbalanced
model:",metrics.accuracy_score(y_reduced_test, y_reduced_pred))
print(f'Total run time for model: {(end - start): .2f} seconds')

##Balanced Reduced
start = time.time()
mlp.fit(X_smreduced_train, y_smreduced_train)
y_smreduced_pred = mlp.predict(X_reduced_test)
end = time.time()

print(metrics.confusion_matrix(y_reduced_test, y_smreduced_pred))
print(metrics.classification_report(y_reduced_test, y_smreduced_pred))
print("Accuracy of Reduced Balanced
model:",metrics.accuracy_score(y_reduced_test, y_smreduced_pred))
print(f'Total run time for model: {(end - start): .2f} seconds')
```

Out[ ]:

```
Confusion Matrix using the classifier using the Original Unbalanced Data
 [[23802  1337]
 [ 1527   848]]

Classification Report

              precision    recall  f1-score   support

           0       0.94      0.95      0.94     25139
           1       0.39      0.36      0.37      2375

    accuracy                           0.90     27514
   macro avg       0.66      0.65      0.66     27514
weighted avg       0.89      0.90      0.89     27514

Total run time for model:  148.44 seconds

Confusion Matrix using the classifier using the Original Unbalanced Data
 [[ 2400 22739]
 [   18  2357]]

Classification Report

              precision    recall  f1-score   support
           0       0.99      0.10      0.17     25139
           1       0.09      0.99      0.17      2375

    accuracy                           0.17     27514
   macro avg       0.54      0.54      0.17     27514
weighted avg       0.91      0.17      0.17     27514

Total run time for model:  298.92 seconds

[[23789  1350]
 [ 1569   806]]
              precision    recall  f1-score   support

           0       0.94      0.95      0.94     25139
           1       0.37      0.34      0.36      2375

    accuracy                           0.89     27514
   macro avg       0.66      0.64      0.65     27514
weighted avg       0.89      0.89      0.89     27514

Accuracy of Reduced Unbalanced model: 0.8939085556443992
Total run time for model:  109.18 seconds

[[16405  8734]
 [  130  2245]]
              precision    recall  f1-score   support

           0       0.99      0.65      0.79     25139
           1       0.20      0.95      0.34      2375

    accuracy                           0.68     27514
   macro avg       0.60      0.80      0.56     27514
weighted avg       0.92      0.68      0.75     27514

Accuracy of Reduced Balanced model: 0.6778367376608272
Total run time for model:  232.87 seconds
```

K-fold cross validation

In [ ]:
```
nn_mean_score = np.mean(cross_val_score(mlp, x_sm_reduced, y_sm_reduced,
                         cv=5))
nn_mean_score
```

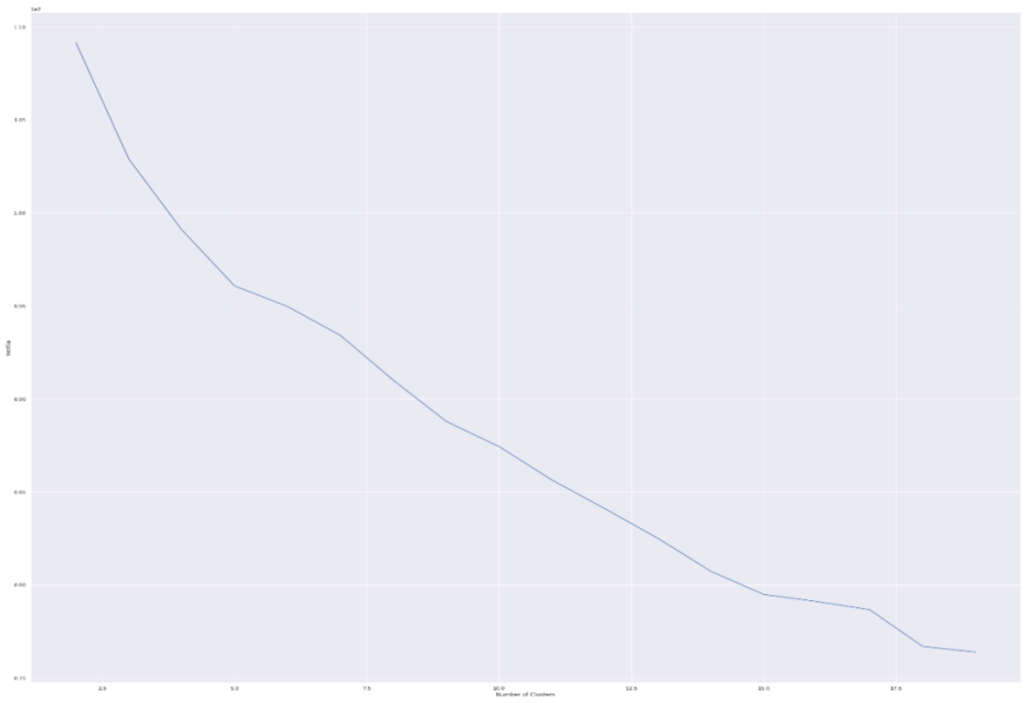
Out[ ]:
```
0.8302107234659986
```

K means clustering

In [ ]:
```
scaler = StandardScaler()
Xn = scaler.fit_transform(x)

# Initialize the list for inertia values - sum of squared distances
inertia_list = []

# Calculate the inertia for the number of clusters.
for i in range(2,20):
    km = KMeans(n_clusters=i, random_state=1234)
    km.fit(Xn)
    inertia_list.append(km.inertia_)

# Check the inertia values.
for i in range(len(inertia_list)):
    print('{0}: {1:.2f}'.format(i+2, inertia_list[i]))

#Draw the plot to find the elbow
plt.plot(range(2,20), inertia_list)
plt.grid(True)
plt.xlabel('Number of Clusters')
plt.ylabel('Inertia')
plt.show()
```

Out[ ]:

```
2: 10914507.21
3: 10289037.71
4: 9908665.37
5: 9606492.07
6: 9496354.51
7: 9340600.90
8: 9099546.47
9: 8878290.57
10: 8742755.72
11: 8562831.93
12: 8408769.22
13: 8249700.32
14: 8072707.51
15: 7947938.26
16: 7910149.90
17: 7866486.96
18: 7668828.57
19: 7638529.31
```



No Elbow was found. Hence no clustering

Data Dictionary

Attributes	**Description**	**Data Type**	**Data Classification**
encounter_id	Unique identifier associated with a patient unit stay	int64	Nominal
patient_id	Unique identifier associated with a patient	int64	Nominal
hospital_id	Unique identifier associated with a hospital	int64	Nominal
age	The age of the patient on unit admission	float64	Numeric
bmi	The body mass index of the person on unit admission	float64	Numeric
elective_surgery	Whether the patient was admitted to the hospital for an elective surgical operation	int64	Nominal
ethnicity	The common national or cultural tradition which the person belongs to	object	Nominal
gender	Sex of the patient	object	Nominal
height	The height of the person on unit admission	float64	Numeric
icu_admit_source	The location of the patient prior to being admitted to the unit	object	Nominal

icu_id	A unique identifier for the unit to which the patient was admitted	int64	Numeric
icu_stay_type	string	object	Nominal
icu_type	A classification which indicates the type of care the unit is capable of providing	object	Nominal
pre_icu_los_days	The length of stay of the patient between hospital admission and unit admission	float64	Numeric
weight	The weight (body mass) of the person on unit admission	float64	Numeric
apache_2_diagnosis	The APACHE II diagnosis for the ICU admission	float64	Numeric
apache_3j_diagnosis	The APACHE III-J sub-diagnosis code which best describes the reason for the ICU admission	float64	Numeric
apache_post_operative	The APACHE operative status; 1 for post-operative, 0 for non-operative	int64	Nominal
arf_apache	Whether the patient had acute renal failure during the first 24 hours of their unit stay, defined as a 24 hour urine output <410ml, creatinine >=133 micromol/L and no chronic dialysis	float64	Nominal
gcs_eyes_apache	The eye opening component of the Glasgow Coma Scale measured during the first 24 hours which results in the highest APACHE III score	float64	Ordinal
gcs_motor_apache	The motor component of the Glasgow Coma Scale measured during the first 24 hours which results in the highest APACHE III score	float64	Ordinal
gcs_unable_apache	Whether the Glasgow Coma Scale was unable to be assessed due to patient sedation	float64	Nominal
gcs_verbal_apache	The verbal component of the Glasgow Coma Scale measured during the first 24 hours which results in the highest APACHE III score	float64	Ordinal
heart_rate_apache	The heart rate measured during the first 24 hours which results in the highest APACHE III score	float64	Numeric
intubated_apache	Whether the patient was intubated at the time of the highest scoring arterial blood gas used in the oxygenation score	float64	Nominal
map_apache	The mean arterial pressure measured during the first 24 hours which results in the highest APACHE III score	float64	Numeric
resprate_apache	The respiratory rate measured during the first 24 hours which results in the highest APACHE III score	float64	Numeric
temp_apache	The temperature measured during the first 24 hours which results in the highest APACHE III score	float64	Numeric

ventilated_apache	Whether the patient was invasively ventilated at the time of the highest scoring arterial blood gas using the oxygenation scoring algorithm, including any mode of positive pressure ventilation delivered through a circuit attached to an endo-tracheal tube or tracheostomy	float64	Nominal
d1_diasbp_max	The patient's highest diastolic blood pressure during the first 24 hours of their unit stay, either non-invasively or invasively measured	float64	Numeric
d1_diasbp_min	The patient's lowest diastolic blood pressure during the first 24 hours of their unit stay, either non-invasively or invasively measured	float64	Numeric
d1_diasbp_noninvasive_max	The patient's highest diastolic blood pressure during the first 24 hours of their unit stay, non-invasively measured	float64	Numeric
d1_diasbp_noninvasive_min	The patient's lowest diastolic blood pressure during the first 24 hours of their unit stay, non-invasively measured	float64	Numeric
d1_heartrate_max	The patient's highest heart rate during the first 24 hours of their unit stay	float64	Numeric
d1_heartrate_min	The patient's lowest heart rate during the first 24 hours of their unit stay	float64	Numeric
d1_mbp_max	The patient's highest mean blood pressure during the first 24 hours of their unit stay, either non-invasively or invasively measured	float64	Numeric
d1_mbp_min	The patient's lowest mean blood pressure during the first 24 hours of their unit stay, either non-invasively or invasively measured	float64	Numeric
d1_mbp_noninvasive_max	The patient's highest mean blood pressure during the first 24 hours of their unit stay, non-invasively measured	float64	Numeric
d1_mbp_noninvasive_min	The patient's lowest mean blood pressure during the first 24 hours of their unit stay, non-invasively measured	float64	Numeric
d1_resprate_max	The patient's highest respiratory rate during the first 24 hours of their unit stay	float64	Numeric
d1_resprate_min	The patient's lowest respiratory rate during the first 24 hours of their unit stay	float64	Numeric
d1_spo2_max	The patient's highest peripheral oxygen saturation during the first 24 hours of their unit stay	float64	Numeric
d1_spo2_min	The patient's lowest peripheral oxygen saturation during the first 24 hours of their unit stay	float64	Numeric
d1_sysbp_max	The patient's highest systolic blood pressure during the first 24 hours of their unit stay, either non-invasively or invasively measured	float64	Numeric

d1_sysbp_min	The patient's lowest systolic blood pressure during the first 24 hours of their unit stay, either non-invasively or invasively measured	float64	Numeric
d1_sysbp_noninvasive_max	The patient's highest systolic blood pressure during the first 24 hours of their unit stay, invasively measured	float64	Numeric
d1_sysbp_noninvasive_min	The patient's lowest systolic blood pressure during the first 24 hours of their unit stay, invasively measured	float64	Numeric
d1_temp_max	The patient's highest core temperature during the first 24 hours of their unit stay, invasively measured	float64	Numeric
d1_temp_min	The patient's lowest core temperature during the first 24 hours of their unit stay	float64	Numeric
h1_diasbp_max	The patient's highest diastolic blood pressure during the first hour of their unit stay, either non-invasively or invasively measured	float64	Numeric
h1_diasbp_min	The patient's lowest diastolic blood pressure during the first hour of their unit stay, either non-invasively or invasively measured	float64	Numeric
h1_diasbp_noninvasive_max	The patient's highest diastolic blood pressure during the first hour of their unit stay, invasively measured	float64	Numeric
h1_diasbp_noninvasive_min	The patient's lowest diastolic blood pressure during the first hour of their unit stay, invasively measured	float64	Numeric
h1_heartrate_max	The patient's highest heart rate during the first hour of their unit stay	float64	Numeric
h1_heartrate_min	The patient's lowest heart rate during the first hour of their unit stay	float64	Numeric
h1_mbp_max	The patient's highest mean blood pressure during the first hour of their unit stay, either non-invasively or invasively measured	float64	Numeric
h1_mbp_min	The patient's lowest mean blood pressure during the first hour of their unit stay, either non-invasively or invasively measured	float64	Numeric
h1_mbp_noninvasive_max	The patient's highest mean blood pressure during the first hour of their unit stay, non-invasively measured	float64	Numeric
h1_mbp_noninvasive_min	The patient's lowest mean blood pressure during the first hour of their unit stay, non-invasively measured	float64	Numeric
h1_resprate_max	The patient's highest respiratory rate during the first hour of their unit stay	float64	Numeric
h1_resprate_min	The patient's lowest respiratory rate during the first hour of their unit stay	float64	Numeric
h1_spo2_max	The patient's highest peripheral oxygen saturation during the first hour of their unit stay	float64	Numeric
h1_spo2_min	The patient's lowest peripheral oxygen saturation during the first hour of their unit stay	float64	Numeric

h1_sysbp_max	The patient's highest systolic blood pressure during the first hour of their unit stay, either non-invasively or invasively measured	float64	Numeric
h1_sysbp_min	The patient's lowest systolic blood pressure during the first hour of their unit stay, either non-invasively or invasively measured	float64	Numeric
h1_sysbp_noninvasive_max	The patient's highest systolic blood pressure during the first hour of their unit stay, non-invasively measured	float64	Numeric
h1_sysbp_noninvasive_min	The patient's lowest systolic blood pressure during the first hour of their unit stay, non-invasively measured	float64	Numeric
d1_glucose_max	The highest glucose concentration of the patient in their serum or plasma during the first 24 hours of their unit stay	float64	Numeric
d1_glucose_min	The lowest glucose concentration of the patient in their serum or plasma during the first 24 hours of their unit stay	float64	Numeric
d1_potassium_max	The highest potassium concentration for the patient in their serum or plasma during the first 24 hours of their unit stay	float64	Numeric
d1_potassium_min	The lowest potassium concentration for the patient in their serum or plasma during the first 24 hours of their unit stay	float64	Numeric
apache_4a_hospital_death_prob	The APACHE IVa probabilistic prediction of in-hospital mortality for the patient which utilizes the APACHE III score and other covariates, including diagnosis.	float64	Numeric
apache_4a_icu_death_prob	The APACHE IVa probabilistic prediction of in ICU mortality for the patient which utilizes the APACHE III score and other covariates, including diagnosis	float64	Numeric
aids	Whether the patient has a definitive diagnosis of acquired immune deficiency syndrome (AIDS) (not HIV positive alone)	float64	Nominal
cirrhosis	Whether the patient has a history of heavy alcohol use with portal hypertension and varices, other causes of cirrhosis with evidence of portal hypertension and varices, or biopsy proven cirrhosis. This comorbidity does not apply to patients with a functioning liver transplant.	float64	Nominal
diabetes_mellitus	Whether the patient has been diagnosed with diabetes, either juvenile or adult onset, which requires medication.	float64	Nominal
hepatic_failure	Whether the patient has cirrhosis and additional complications including jaundice and ascites, upper GI bleeding, hepatic encephalopathy, or coma.	float64	Nominal

immunosuppression	Whether the patient has their immune system suppressed within six months prior to ICU admission for any of the following reasons; radiation therapy, chemotherapy, use of non-cytotoxic immunosuppressive drugs, high dose steroids (at least 0.3 mg/kg/day of methylprednisolone or equivalent for at least 6 months).	float64	Nominal
leukemia	Whether the patient has been diagnosed with acute or chronic myelogenous leukemia, acute or chronic lymphocytic leukemia, or multiple myeloma.	float64	Nominal
lymphoma	Whether the patient has been diagnosed with non-Hodgkin lymphoma.	float64	Nominal
solid_tumor_with_metastasis	Whether the patient has been diagnosed with any solid tumor carcinoma (including malignant melanoma) which has evidence of metastasis.	float64	Nominal
apache_3j_body system	Admission diagnosis group for APACHE III	object	Nominal
apache_2_body system	Admission diagnosis group for APACHE II	object	Nominal
hospital_death	Whether the patient died during this hospitalization	int64	Nominal