Distributed Data Analysis Report

1. Data description and research question

The dataset is about admitted patients when a patient is hospitalized to predict if a patient will survive or not with the help of adopting Machine learning algorithms implemented in High Performance Computational language- PYSPARK. The dataset is 31.41 MB of size and stored as a CSV file. It is retrieved from Kaggle website with URL mentioned below. It consists of different parameters collected when a patient is admitted like medical factors w.r.t blood pressure, heartrate, respiratory rate, temperature etc., demographic factors like age, gender, ethnicity etc., identification factors like patient ID, icu ID, hospital ID, etc. In this report we aim to provide a prediction if a patient is going to survive or not depending on the various parameters/variables in the dataset.

Research Question for dataset:

Using the medical parameters in the dataset, predict whether the patient will survive or not?

2. Data cleaning and preparation

Data cleaning is the most important and crucial part of the modelling for prediction since feeding in the right data into any prediction algorithm is most essential (Bowne-Anderson, 2020). In this section of the report, we clean and pre-process the data before using it for prediction.

```
patinet_data <- read.csv("out_pat_data.csv",stringsAsFactors = TRUE, na.strings = c("",NA))
str(patinet_data)
summary(patinet_data)
```

By using the *'str()'* function, we see the dataset is a mix of numeric, binary and categorical variables but also includes identification columns like 'X.1, X, encounter_id, etc.,' which need to be dropped. Using the ever-efficient *'summary()'* function, we get the summary statistics of all the objects in dataframe and also the NA's count. The dataset cleaning consists of dealing with missing values (NAs), duplicate instances, data type conversions and outliers which is achieved by the help of various exploratory data analysis. These are segregated into subsections of the report below –

2.1 Dealing with missing values

A. Finding the number and percentage of Missing values (NAs)

On a preliminary note, we first look for the number and percentage of NAs in each variable in the dataframe by using 'is.na()' function inside the 'apply()' function.

```
patinet_data_NA_count <- apply(is.na(patinet_data), 2, sum)
patinet_data_NA_count
patinet_data_NA_perc <- patinet_data_NA_count / dim(patinet_data)[1] * 100
patinet_data_NA_perc
```

B. Dropping the columns which have more than 5% NAs

As our group agreed to have a 5% threshold to drop the columns with NAs, we now further move on to drop the columns above the threshold (Rengaraju, 2020). Using the 'sapply' function on 'patient_data' dataset with a conditional statement to remove the columns which have more than 0.05% mean NAs reducing our dataset to 70 variables, meaning that 15 variables were dropped which had more than 5% NAs.

```
pat_data <- patient_data[,!sapply(patient_data, function(x) mean(is.na(x)))> 0.05]
```

The *'diagnose()'* function from the 'flextable' package is a powerful tool to look at the summary statistics of numerical and categorical variables w.r.t missing value and uniqueness of variables. These functions are not sensitive to variables and hence feeding any wrong variables in the categorical and numerical functions will be ignored automatically.

```
diagnose(pat_data)%>% flextable()
diagnose_category(pat_data)%>% flextable()
diagnose_numeric(pat_data)%>% flextable()
```

Unique variables (unique count = 1) are bound to be eliminated from data analysis. If the data type isn't numeric (integer, numeric), and the number of unique values equals the number of observations (unique rate = 1), then the variable is probably an identifier. In this table, 'encounter_id' & 'patient_id' are such variables which are inappropriate for our analytical model. Although there are other ID variables like *'icu_id'* and *'hospital_id'* which don't have 1 as their unique_rate, we should omit them in our analysis as well (choonghyunryu.github.io, n.d.). Comparing the previous bland *'diagnose()'* function's results with the *'diagnose_category()'* results, we can relate that the unique count of categorical/factor variables are equal to the number of levels in the categorical/factors. Since we do not have to include identifier variables in our dataset, we can confirm this by linking both the results. But we do not have to remove all the variables which match but only the ones with identification factors like 'icu_admit_source', 'icu_stay_type', 'icu_type'. All the rest categorical variables are necessary for our analysis. From the *'diagnose_numeric()'* statistics, if we look closely at *'gcs_eyes_apache', 'gcs_motor_apache'* and *'gcs_verbal_apache'* appear to be as factors since their range is from 0 to 1,4,6 respectively. Hence, we convert these variables into factors. The other binary variables are converted in the later stage. If we look at the numeric results more closely, Zero, minus and outliers' columns show that 'pre_icu_los_days' has 778 negative values which cannot be possible. Days in the ICU cannot be negative and hence this needs to be imputed with median.

```
pat_data$ gcs_eyes_apache = as.factor(pat_data$gcs_eyes_apache)
pat_data$ gcs_motor_apache = as.factor(pat_data$gcs_motor_apache)
pat_data$gcs_verbal_apache = as.factor(pat_data$gcs_verbal_apache)
#replacing the negative values with median.
out_pat_data$pre_icu_los_days[out_pat_data$pre_icu_los_days<0]<-median(out_pat_data$pre_icu_los_days)
```

2.2 Imputation of Missing values

To impute the missing values w.r.t numerical, categorical and binary variables, different approaches are carried out. The numerical columns are imputed with the median whereas the categorical columns are imputed with mode (Harshitha Mekala, 2018). However, the binary variables are imputed using logistic regression from the 'mice' package (Bhalla, n.d.).

Imputing categorical variables

```
Mode <- function(x) {
  ux<- unique(x)
  ux[which.max(tabulate(match(x,ux)))]
  }
pat_data <- pat_data %>% mutate_if(is.factor,funs(replace(.,is.na(.), Mode(na.omit(.)))))
```

Imputing the categorical variables requires a generic custom mode function to be initially defined. Using the *'is.factor()'* function inside the *'mutate_if()'* function from the *'dplyr'* package is used to select only the factors variables. The *'replace()'* function restores all values which have missing values (NAs) with the mode which was defined earlier.

Imputing numerical variables

The numerical variables are replaced with the median by applying a conditional IF ELSE statement using the *'lapply'* function. Since the binary variables have blank (NA), 0 and 1 as unique lengths, the IF ELSE statement is passed w.r.t to the unique lengths greater than 3. The condition of statement replaces NAs with unique length more than 3 but remains unchanged for equal or less than 3.

```
data.frame(lapply(pat_data,function(x) {
    if(is.numeric(x) && length(unique(x))>3)ifelse(is.na(x),median(x,na.rm=T),x) else x}))-> pat_data
```

Imputing binary variables

Replacing the binary variables is a slightly more complex task compared to the categorical and numerical ones since they consist of only 2 factors, ruling out the option of imputing them mean or median. This can be done through the use of logistic regression in the 'mice' package. However, If the method option is left blank, it checks the variable type by default and applies the missing imputation method based on the variable type. To impute the binary variables, we may use *'filter()'* and *'diagnose()'* function to refine the variables having equal or less than 3 unique count and missing count above 1. Even though the binary variables have only 2 factors, the unique count is set as 3 since there is another blank/missing (NA) variables in the categorical columns.

```
pat_data %>%
diagnose() %>%
filter(unique_count<=3 & missing_count>0)
```

The list of 12 variables having missing values with their unique count and rate is returned in a table with their data types. Although these are still integer values, all of them will be converted to factors later on after we remove the duplicate instances from the variables and missing values. We may now progress to use *'mice()'* to impute the binary variables using logistic regression. In the above code for imputation, 'm' is defined to the number of imputed data sets assigned during imputation while the 'maxit' parameter is defined to the number of iterations taken to impute missing values. The method used defined here is 'logreg' which is short for logistic regression. The *'mice()'* function produces 5 different copies of the dataset ran on 5 iterations. After the imputations are finished, we now move on to use the 'complete()' function to assign on our dataset. We can confirm this by seeing the summary statistics and also using the 'is.na()' function on the sum of the columns *'colSums()'*.

```
my_imp_1 = mice(pat_data, m=5, method = "logreg",maxit = 5)
pat_data = complete(my_imp_1,5)
summary(pat_data)
colSums(is.na(pat_data))
```

2.3 Removing duplicate instances

Duplicates might be an indication of a data problem, which could affect subsequent analysis in the report. Duplicate instances also create bias in our prediction when repetitive observations are fed in the training data. As a result, checking for data collection for duplicates and omitting them is an ideal approach (Banghart, n.d.). The 'unique()' function in R is used to delete the rows which have duplicate or repetitive rows in the dataset. This code clears the issue of missing values in the dataset preventing any biasing towards our analysis and prediction in further stages.

```
pat_data<- unique(pat_data)
```

2.4 Data type conversion

From the numerical diagnose statistics, we also observed that many of the numeric variables have to be changed to factors. We can change these variables to factor by seeing which of them have unique count of 2. Changing the data type of binary variables from numeric to factor is done by the help of 'as.factor()' function.

```
pat_data %>%
diagnose() %>%
filter(unique_count<= 2)
pat_data$elective_surgery <- as.factor(pat_data$elective_surgery)
pat_data$ apache_post_operative = as.factor(pat_data$apache_post_operative)
pat_data$ arf_apache = as.factor(pat_data$arf_apache)
pat_data$ gcs_unable_apache = as.factor(pat_data$gcs_unable_apache)
pat_data$intubated_apache = as.factor(pat_data$intubated_apache)
pat_data$ventilated_apache = as.factor(pat_data$ventilated_apache)
pat_data$aids = as.factor(pat_data$aids)
pat_data$ cirrhosis = as.factor(pat_data$cirrhosis)
pat_data$diabetes_mellitus = as.factor(pat_data$diabetes_mellitus)
pat_data$ hepatic_failure = as.factor(pat_data$hepatic_failure)
pat_data$ immunosuppression = as.factor(pat_data$immunosuppression)
pat_data$leukemia = as.factor(pat_data$leukemia)
pat_data$lymphoma = as.factor(pat_data$lymphoma)
pat_data$solid_tumor_with_metastasis = as.factor(pat_data$solid_tumor_with_metastasis)
```

2.5 Outliers

Outliers are data points that are far off distant from the rest of the datapoints, disrupting the dataset's general distribution. Outliers generally have 3 effects on the model generation which are skewing the data, changing the overall statistical distribution of data w.r.t mean, variance, etc, and leading to a bias in the accuracy level of our model and data interpretation (Kwak and Kim, 2017). Treating outliers in our analysis/model is completely on the analyst's choice because it mainly depends on the domain/context of the analyses and research question. In some datasets it is an efficient approach to remove the outliers to squeeze better results out of the analysis or prediction while some analysis require to consider them as they contain critical and valuable information needed (Stats and R, n.d.). However, working as a group in this project, we decided to remove the outliers from the analysis for a more generic and safer approach towards biasing our prediction accuracy. Another reason to not include outliers is due to the usage of K-Means clustering for exploratory data analysis for the prediction model because K-means clustering is sensitive to outliers and therefore removing of outliers for this was more suitable (Franklin, 2019). From the table we see that, most of the variables contain outliers which we can interpret from the 'outlier_cnt'. The 'outliers_ratio' gives the percentages of outliers in each variable while the 'outliers_mean' gives the average of outliers in the variable. More insights can be drawn by plotting the outliers of the dataset using the 'plot_outlier()' function. The correction of skewed variables can be seen by comparing the with and without outlier graphs of boxplot and density distribution of the variables.
A sample of removing the outliers is shown above. By locating the outliers in the variable using the 'which()' function inside the created boxplot objects of the variables, we are able to remove the outliers. This is repeated for all the variables having outliers.

```
# Identifying the outliers.
diagnose_outlier(pat_data)%>% flextable()
# Plotting with or without outliers boxplots and density graphs w.r.t each variable
plot_outlier(pat_data)
out_pat_data_boxplot_age <- boxplot(out_pat_data$age)

out_pat_data_boxplot_age$out

out_pat_data$age[which(out_pat_data$age %in% out_pat_data_boxplot_age$out)]<-"NA"
```

3. Exploratory data analysis

In this section of the report, we see how we explore the data in depth regarding the patient's survival with the help of exploratory tools like normality, correlation, principal component analysis and K-means clustering.

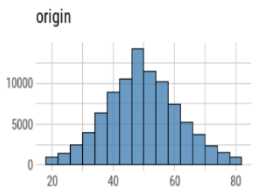
3.1 Normality

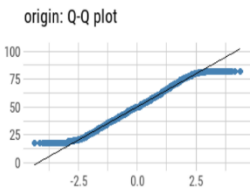
The normality of the data w.r.t the patient's survival is explored using normality statistics, density and Q-Q plots of the numeric variables in the dataset.

```
# Normality statistics
normality(out_pat_data[numeric]) %>% flextable()
# Checking the normality of numerical variables by plotting the q-q plot and histogram.
out_pat_data[numeric] %>%S
  plot_normality(all_of(numeric))
```

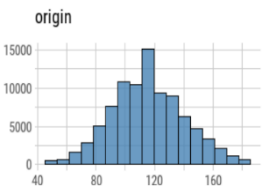
The normality regarding the statistics can be yielded through looking closely which variables have statistics closer to 1 and later reconfirm by looking at their density and Q-Q plots. The variables recording minimum parameters have more normality meaning they are less skewed and have relation w.r.t the patients survival. We can see that parameters like *d1_diasbp_min, d1_diasbp_noninvasive_min, d1_heartrate_min, d1_mbp_min, d1_mbp_noninvasive_min, d1_resprate_min, d1_spo2_min, d1_sysbp_min, d1_sysbp_noninvasive_min, d1_temp_min, h1_diasbp_min, h1_heartrate_min, h1_resprate_min, h1_spo2_min* and *h1_sysbp_min* are more normally distributed than the maximum parameters concluding that as the parameters tend to drop towards the minimum, there is some relation with the patients survival i.e. eventually leading to the patient's death. We can confirm this by also seeing the normality plots and Q-Q plots that these minimum parameters are more normally distributed than the maximum ones. Some examples of such normally distributed minimum parameters are given below. The non-inclusive usage of normality tests is a very critical thing to reflect here. Normality tests like Shapiro-Wilk tests have best use cases in terms of theoretical normality. In large datasets, these tests will discover even very minute and modest deviations from theoretical normality. This arises a question of doubt in data scientists mind in the practical concerns of using these normality tests. Therefore, we have chosen to look for normality in the density and Q-Q plots. (Greener, 2020)

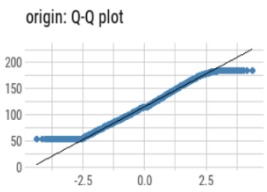
Normality Diagnosis Plot (d1_diasbp_min)





Normality Diagnosis Plot (h1_sysbp_min)

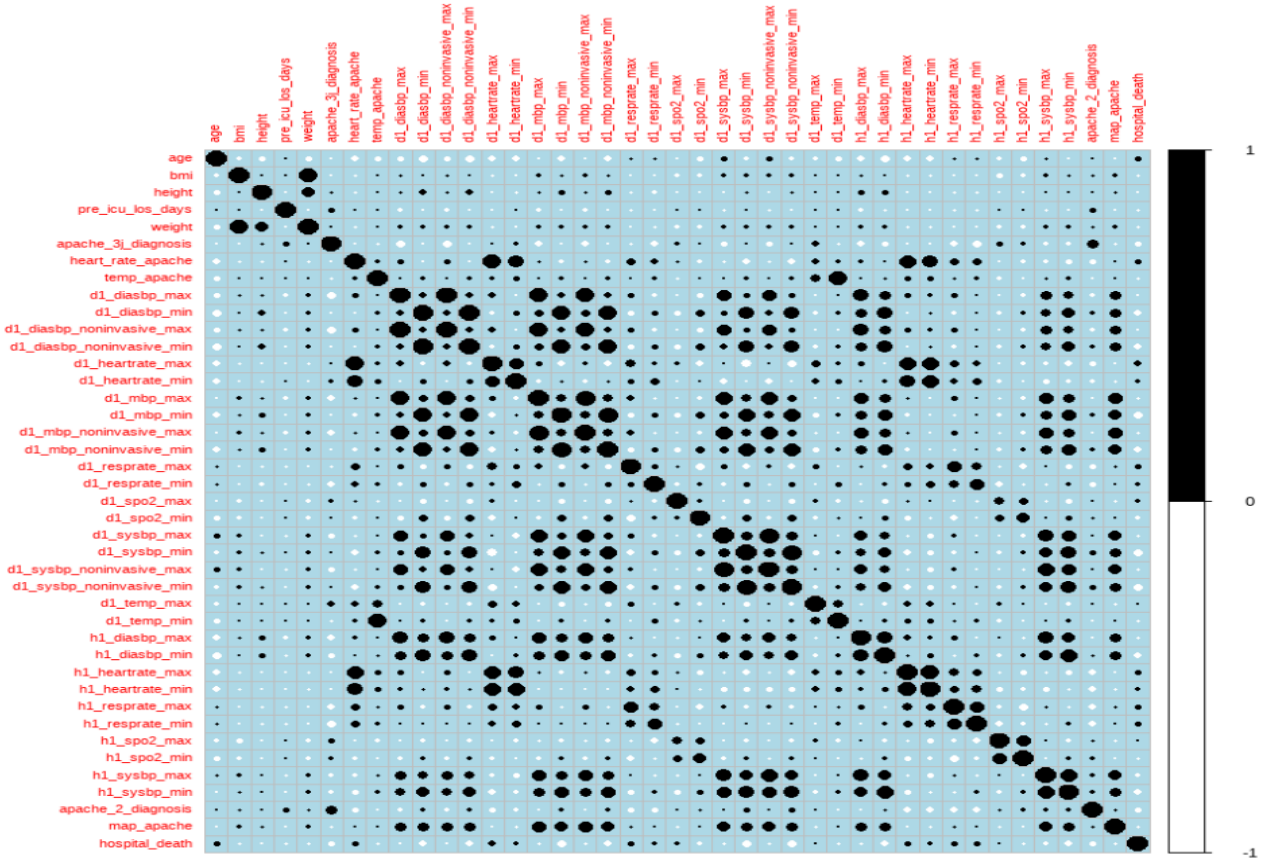




3.2 Correlation

Correlation is a very insightful aspect in terms of interpreting how the numerical variables interact with each other. To see the interactions between these medical parameters, we first create a correlation matrix and then plot it using the 'corrplot()' function.

```
#correlation matrix
cor.mat <- cor(out_pat_data[numeric])
cor.mat
#plot the correlation matrix
corrplot(cor.mat, tl.cex = 0.7)
```



By having a close look at the correlation plot, we can note that the *height, weight* and *bmi* are correlated to each other which is evidences at how *bmi* is calculated. *height* has positive correlation w.r.t minimum parameters of blood pressure indicating that taller patients have a lower resting heart rate compared to shorter patients. w*eight* on the other hand, has slightly negative correlation with *sp02* parameters indicating that obese patients tend have more oxygen consumption fluctuations. The most visibly highlighted correlations of heart rate, blood pressure, respirate and spO2 are with their hour 1 and day 1 parameters or with the same type of parameters recorded i.e., *heart_rate_apache* is highly correlated to *d1_heartrate_min*. For another example, *h1_diasbp_max* and *h1_diasbp_max* are highly correlated to *d1_diasbp_max* & *d1_diasbp_min* which is very obvious. Moving away from the common correlations and looking closer for patterns we can see that *h1_heartrate_max* & *h1_heartrate_min* have positive correlations with *heart_rate_apache, temp_apache, h1_diasbp_max, h1_diasbp_min d1_diasbp_max, d1_resprate_max, d1_resprate_min, d1_temp_max & d1_temp_min*. This ceases an intuition that heartrate is dependently correlated with the patient's blood pressure, temperature and respiratory rate. However, for predicting the patient's survival, we need to look into the interactions of *hospital_death* with other parameters to find patterns with which it is correlated with. Relating from the intuition above, when we look at the correlations of *hospital_death,* we can discover that it is correlated to *age, heart_rate_apache, d1_heartrate_max, d1_heartrate_min, d1_resprate_max, d1_resprate_min, h1_heartrate_max, h1_heartrate_min, h1_resprate_max* and *h1_resprate_min.* This is true from the biological aspect of human body too, since the heart rate depends on the body's blood flow, temperature and oxygen intake.

3.3 Principal Component Analysis

In principal component analysis, we create new superficial linear combinations of original variables in order to have maximum variance spread of the new variables across is high (Ashwini Kumar Pal, 2017). Having a lot of data can be a curse in machine learning because too many features or dimensions can reduce our model's accuracy. Since our dataset is large with too many noise

columns/features, we can use dimensionality reduction in PCA to feature extract our desired variables which are most important for prediction (Li, 2019). Initially, we start off by standardising the data in order to get all the numerical columns to generalised scale.

```
In [8]: ##Standardising the numerical data in order to get a normal scale
        df_st =  StandardScaler().fit_transform(df1)
```

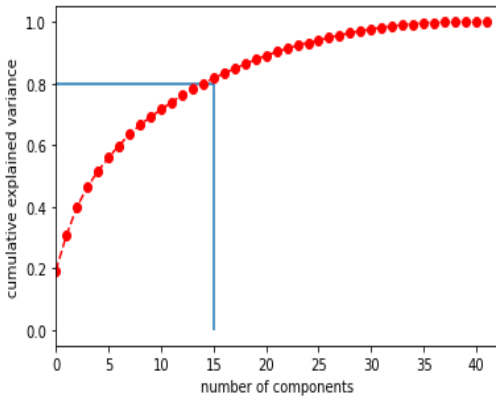
Now we may apply the PCA to find the components and calculate the component variance. Components are directions of maximum variance in the data. The cumulative proportion of variance explained can be derived and plotted on a graph. The threshold for selecting the number of PC's is agreed upon to be 80% by the group. (Jr, 2022)

```
In [9]: ##Applying PCA to find PCA's components
        pca_n = PCA().fit(df_n)
        components = pca_n.transform(df_n)
```

```
In [10]: ##Calculating the component variance
         pca_n.explained_variance_ratio_
```

```
In [11]: ## Caluclation of cumulative proportion of variance
         np.cumsum(pca_n.explained_variance_ratio_)
```

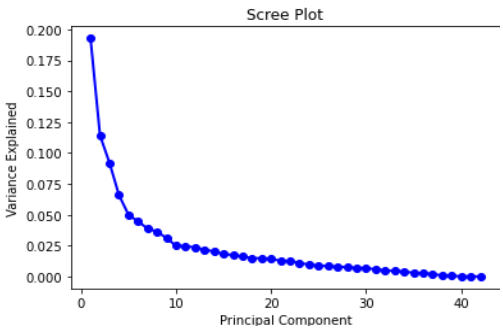
```
In [16]: ##Plotting Cumalative values of PCs and adding a 80% threshold line.
         plt.plot(np.cumsum(pca_n.explained_variance_ratio_), marker='o', linestyle='--', color='r
         plt.xlabel('number of components')
         plt.ylabel('cumulative explained variance')

         plt.xlim(0,42)
         plt.hlines(y=0.8, xmin=0, xmax=15)
         plt.vlines(x=15, ymin=0, ymax=0.8)
```



Since PCA replaces original variables with new variables called principal components, explained variance is the amount of variance explained by each component. As shown in the plot, having an 80% threshold gives us the cut off for choosing the number of components as 15. This fulfils the dimensionality reduction from 42 components to just 15.

Similarly, a scree plot is drawn to show the respective variance explained by each PCs. The first PC contributes to the highest variance. There is a point on the graph where the difference between the PCs gets stabilized, which is at after 15 PCs. This is also where 80% of the explained ratio is attained. Hence, the first 15 PCs are enough to know the most about our dataset and prediction.

```
In [24]: ##Screeplot of PCs
         PC_values = np.arange(pca_n.n_components_) + 1
         plt.plot(PC_values, pca_n.explained_variance_ratio_, 'o-', linewidth=2, color='blue
         plt.title('Scree Plot')
         plt.xlabel('Principal Component')
         plt.ylabel('Variance Explained')
         plt.show()
```



3.4 Clustering

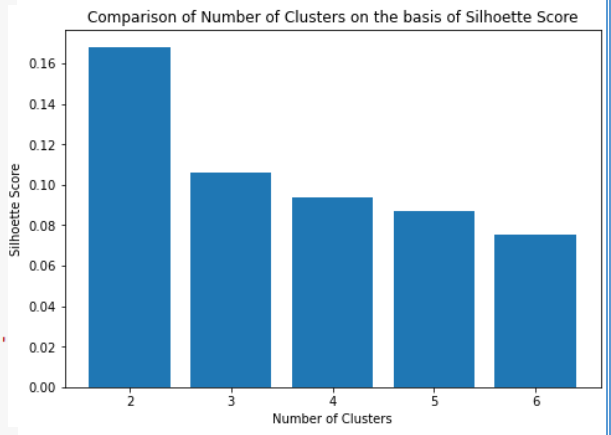
Clustering is an unsupervised learning method to segregate data points into groups for better understandability of the data using similarity/dissimilarity and distinctiveness metrics. The main idea is to put each data point into a cluster of similar data points which are dissimilar to data points belonging to a different cluster. This is done through various similarity measures like vectors (cosine distance), sets (jaccard distance) and points (Euclidean Distance) (Sanatan Mishra, 2017). In this dataset, K-Means clustering is used to determine the clusters where a pre-defined level of clusters *'n'* and *'k'* is set to use as a preliminary centroid datapoint. This centroid starts a repetitive jump to the subsequent datapoint which is similar to it and becomes the centroid until all the similar datapoints are captured inside the cluster. The main advantage of using K-means is because it's easy

to implement, faster and flexible on large datasets but the drawback is that its sensitive to outliers which is why we removed the outliers initially. Another drawback is to specify the number of centroids before clustering is done. Although as a group we also considered other methods like Hierarchical clustering and DBSCAN, but scalability and time-consuming issues for our large dataset did not make them efficient enough (Blogs & Updates on Data Science, AI Machine Learning, 2020).

```
In [28]: #Normalising the numerical data between 0 to 1 in order to get a normal scale
         std = MinMaxScaler()
         arr1 = std.fit_transform(new_df)
```

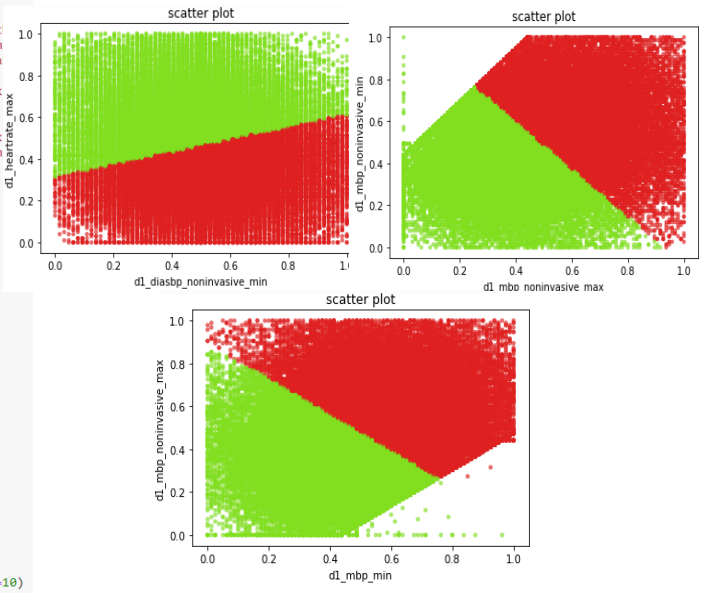
```
In [30]: #Using For Loop to get the Silhoette Score at different values of n_clusters
         n = [2,3,4,5,6]
         silhouette_sco = []
         max_score = 0
         max_n = 0
         for i in n:
             kmeans = KMeans(n_clusters = i, random_state = 45)
             kmeans.fit(df_normal)
             y_kmeans = kmeans.predict(df_normal)
             val = silhouette_score(df_normal, y_kmeans)
             silhouette_sco.append(val)
```

```
In [9]: #Plotting the n_clusters value against Silhoette Score
        fig=plt.figure()
        ax = fig.add_axes([0,0,1,1])
        ax.bar(n,silhouette_sco)
        plt.title('Comparison of Number of Clusters on the basis of Silhoette Score')
        plt.xlabel('Number of Clusters')
        plt.ylabel('Silhoette Score')
        plt.show()
```



Primarily, we feed in only the numerical data into the K-means algorithm since categorical data is not suitable for K-Means (Kumar, 2021). We kick-start the clustering by normalising the data from 0 to 1 since all the variables need to be on a generalized scale. Applying the KMeans inside the FOR loop defining i = 2 to 6, we can obtain the silhouette scores for 6 clusters. We plot the silhouette scores of each of these clusters on the bar graph to realise that the scores have significantly dropped after the 2nd cluster. The 2nd cluster returned a silhouette score of 0.1875 and we can see from the graph that there is a drastic drop in the score and hence this points out to the fact to choose k=2 as the number of clusters to explore more about our clusters (Kumar, 2020). Therefore, we choose the number of clusters to be 2 for our analysis.

```
In [*]: #Using KMeans for Number of Clusters value = 2
        #Consideration of all the variables for clustering
        arr =['age','bmi','pre_icu_los_days','apache_2_diagnosis','apache_3j_diagnosis',
              'gcs_eyes_apache','gcs_motor_apache','gcs_verbal_apache','heart_rate_apac
              'map_apache','resprate_apache','temp_apache','d1_diasbp_max','d1_diasbp_m
              'd1_diasbp_noninvasive_max', 'd1_diasbp_noninvasive_min','d1_heartrate_ma
              'd1_heartrate_min','d1_mbp_max','d1_mbp_min','d1_mbp_noninvasive_max',
              'd1_mbp_noninvasive_min','d1_resprate_max','d1_resprate_min','d1_spo2_max
              'd1_spo2_min','d1_sysbp_max','d1_sysbp_min','d1_sysbp_noninvasive_max',
              'd1_sysbp_noninvasive_min','d1_temp_max','d1_temp_min','h1_diasbp_max',
              'h1_diasbp_min', 'h1_heartrate_max', 'h1_heartrate_min', 'h1_resprate_max
              'h1_resprate_min','h1_spo2_max', 'h1_spo2_min','h1_sysbp_max','h1_sysbp_m

        for i in range(len(arr)-1):
            kmeans = KMeans(n_clusters = 2, random_state = 45)
            df_normal['cluster'] = kmeans.fit_predict(df_normal[[arr[i],arr[i+1]]])
        #Get the centroids
            centroids = kmeans.cluster_centers_
            cen_x = [i[0] for i in centroids]
            cen_y = [i[1] for i in centroids]

            #Plotting the total numbers of elements in each cluster
            count = df_normal['cluster'].value_counts()
            fig = plt.figure()
            ax = fig.add_axes([1,1,1,1])
            ax.bar(count.index,count)
            plt.xticks(count.index, color='black', fontweight='bold', fontsize='10',
                       horizontalalignment='right')
            plt.xlabel('Clusters')
            plt.ylabel('Counts')
            plt.title('Cluster Counts')
            plt.show()
         #Adding the centroids to the dataframe
            df_normal['cen_x'] = df_normal.cluster.map({0:cen_x[0], 1:cen_x[1]})
            df_normal['cen_y'] = df_normal.cluster.map({0:cen_y[0], 1:cen_y[1]})
            #Defining the colur map
            colors = ['#DF2020', '#81DF20']
            df_normal['c'] = df_normal.cluster.map({0:colors[0], 1:colors[1]})
            #Plotting the scatter plot
            plt.scatter(df_normal[arr[i]], df_normal[arr[i+1]], c=df_normal.c, alpha = 0.6, s=10)
            plt.title("scatter plot")
            plt.xlabel(arr[i])
            plt.ylabel(arr[i+1])
```



The FOR loop is used again for 2 clusters and fitting the Kmeans to the numerical dataframe. The

centroids of the clusters can be reached by using 'kmeans.cluster_centers_' from the sklearn library in the Python. The scatter plot for all the variables can be plotted using the centroid and plot all the elements inside the cluster adding the respective colours to it. From the scatter plot, we can interpret that though our silhouette score is low, the clusters are still formed correctly. Though the cluster data points are closely packed but the colour differentiation of green and red shows that they are distinct in nature. It was not a better choice to choose a higher number of clusters to get a better silhouette score. Therefore, from the scatter plots we can say that although choosing the right number of clusters and getting a low silhouette score, the clusters are properly formed and distinct from one another. This intuits that the difference between the clusters is significant even though there were low scores and right choice in number of clusters (Bhardwaj, 2020).

4. Machine Learning Prediction

Machine learning prediction is the computer's ability to recognise pattern recognition with the help of various algorithms to produce predictive information about the data. The output of prediction mainly depends on what training data is fed primitively inside the algorithm. In this report, we see how Decision trees and Random forests are used to predict the patient's survival by using the various medical parameters recorded when admitted. Random forests are originally an ensemble of decision trees which combines the output of multiple randomly generated Decision Trees while Decision trees are a series of sequential decisions to just form 1 tree. We can also relate Random Forests to the concept of bagging, where several subsets of data are derived from base dataset. Similarly, in random forests the division of features along with the data are used to grow the separate individual trees (Paperspace Blog, 2020). The repetitive nature of random forests makes it less overfitted than decision trees producing less deeper trees also boosting the accuracy at a cost of increased storage & computational power (Team, 2020). Since this is an ensemble of Decision trees, Random forests cannot be implemented without creating a decision tree at first. In this dataset, we start by using stratified random sampling on the data w.r.t *'hospital_death'* to balance the distribution since the imbalance in data can create bias to our prediction. Sampling also help is in run time, balancing the majority class which is '0' in our target variable by using proportioning/fractioning of the distribution (Analytics Vidhya, 2021). The main motive to do stratified sampling is to retain the overall population characteristics of the dataset similar in our sample. This removes the concept of bias in the way sampling is done (Investopedia, n.d.). Now we move on to use vector assembler to transform the numerical columns into a single vector called features and string indexer to encode *'hospital_death'* into label indices (spark.apache.org, n.d.). Encoding the categorical variables is not needed in random forest or decision trees since doing so will induce sparsity into the dataset, causing inefficiency to the model. Using One-Hot-Encoding on the categorical, will only create new levels in the tree which is not good for tree interpretation (Ravi, 2019). Both the assembler and indexer are defined as features and labels first and then transformed on the dataset. Next step is to split the dataset into train and test for prediction purposes. The concept of splitting the data is to use the train set to feed in data into the Random Forest algorithm to learn about the data and its interactions and then use the test set to evaluate how good the algorithm has learnt about the data to predict about it. The dataset is divided on a general 70/30 split giving 10190 and 4315 records respectively. The algorithm for decision tree and random forest is almost the same where

we assign the features and labels into a classifier and then fit this into the train set. This creates a classification tree through which the predictions measures are obtained. *'prediction'* is the predicted label of *'hospital_death'*, *'rawPrediction'* is the measure of train labels at the node of the tree assisting in the prediction, whereas *'probablity'* is the normalized raw prediction on a scale of 0-1. Although, the decision trees are easy to build and interpret, random forests are a combination of randomly generated multiple trees where each tree is a predictor making it slightly more difficult to interpret. Thus, using random forest gives a boost in the accuracy from the basic level decision tree. To accomplish further more prediction accuracies, we can use Gradient-Boosted Tree algorithm which uses concept of boosting the weak predictors by using information about previous built trees whereas Random Forests depend on the idea of bagging by averaging the prediction of all individual trees. The depth of trees is comparatively standardized to 3 as unlike Random Forests because building individual trees can soon end in overfitting the data. In conclusion, the high accuracy rate of GBT is achieved at the cost of small tree depth which are difficult to interpret and harder to tune also resulting in larger bias than the other 2 algorithms (Kraan, n.d.).

5. High Performance Computational Implementation

Apache Spark is a data processing framework that can perform processing tasks on very large data sets quickly, as well as distribute data processing tasks across multiple computers. The main advantage to use spark instead of Hadoop is speed of the batch processing. Spark consists of cluster manager which works with the driver program and worker nodes. The driver schedules tasks on the worker nodes for the assigned job through the help of cluster managers chosen like Standalone, Mesos and Hadoop YARN which help in resource allocation and tracking of jobs/tasks. There are many APIs connected to spark but PySpark was the most efficient and flexible to me to apply ML on this dataset. The collaboration of Python and Spark integrates the simplicity with speed to do ML prediction. Given below with screenshots are steps to do ML prediction in Pyspark.

In [8]:
```
# Creating master nodes
spark = SparkSession.builder\
 .master("local[*]")\
 .appName("ML Coursework")\
 .getOrCreate()
sc = spark.sparkContext
```

In [11]:
```
# Finding the distribution w.r.t target variable
df.groupBy("hospital_death").count().toPandas()
```

In [17]:
```
# Using stratified sampling to balance the target variables distribution
from pyspark.sql import Row
from pyspark.sql import SparkSession

df_s = df.sampleBy("hospital_death", fractions={1: 0.9125, 0: 0.0875}, seed=0)

df_s.groupBy("hospital_death").count().toPandas()
```

To start the spark session in PySpark, we primarily have to create a spark session which creates master nodes for the jobs to be assigned. After reading in the dataset and using the *'printSchema()'*, we can print the structure of the dataframe. Having a look at the dataframe, we can observe that the dataset contains some unnecessary identifier columns for our prediction, which may be dropped using the *'df.drop()'* function. We can know more about the distribution of the dataset w.r.t our target variable "hospital_death" using the *'df.groupBy()'* function which returns that it is largely imbalanced. The simplest and most efficient way to sort this imbalance is do stratified sampling using fractioning of target variables in the *'df.sampleBy'* function. After the sampling, we can see

In [18]:
```
# Using String and Vector Assembler to obtain features and labels
from pyspark.ml.feature import VectorAssembler
from pyspark.ml.feature import StringIndexer

numericCols = ["age", "bmi", "height","pre_icu_los_days","weight","apache_3j_diagnosis","heart_rat
assembler = VectorAssembler(inputCols=numericCols, outputCol="features")
df1 = assembler.transform(df_s)
label_stringIdx = StringIndexer(inputCol = 'hospital_death', outputCol = 'label')
df2 = label_stringIdx.fit(df1).transform(df1)
df2.printSchema()
```

```
# Splitting the data into Train & Test
train, test = df2.randomSplit([0.7, 0.3], seed = 2018)
print("Training Dataset Count: " + str(train.count()))
print("Test Dataset Count: " + str(test.count()))
train.printSchema()
```

that the counts of *'hospital_death'* having 1 and 0 has been balanced to 7257 & 7248 respectively. Now using the vector assembler and string indexer, we may fit and transform to obtain the features and labels needed for prediction. The vector assembler is used to put all the numerical features of the row into a single feature array called features while the string indexer encodes *'hospital_death'* into a label column. After assembling and encoding, we may now move to split the dataset in a 70/30 split to use it to use as train and test set for prediction. Below are the screenshots of codes of decision tree, random forest and gradient-boosted algorithms. The thing in common in all the algorithms is that initially a classifier of the respective algorithm is used to assign the features and labels inside and later fitted on the train set into the respective algorithm to obtain the model. Later, these models are transformed on the test to get the predictions on how well the algorithm has learnt about the data fed in. The evaluation of these algorithms is done in the next part of the report.

```
In [35]: # Decision Tree
         # import the Decision Tree classifier
         from pyspark.ml.classification import DecisionTreeClassifier

         # configuring and training the Decision Tree classifier using the training data
         dt = DecisionTreeClassifier(featuresCol = 'features', labelCol = 'label', maxDepth = 3)
         dtModel = dt.fit(train)
         # Testing the performance of the Decision Tree classifier using the testing data
         predictions = dtModel.transform(test)
         predictions.select('label', 'prediction', 'rawPrediction', 'probability').show(10)
```

```
# Random Forest
from pyspark.ml.classification import RandomForestClassifier

# configuring and training the Random Forrest classifier using the training data
rf = RandomForestClassifier(featuresCol = 'features', labelCol = 'label')
rfModel = rf.fit(train)

# Testing the performance of the Random Forest classifier using the testing data
predictions = rfModel.transform(test)
predictions.select('label', 'prediction', 'rawPrediction', 'probability').show(10)
```

```
In [41]: # Gradient Boosted Tree
         # import the Gradient-Boosted Tree classifier
         from pyspark.ml.classification import GBTClassifier
         # configuring and training the Gradient-Boosted Tree classifier using the training data
         gbt = GBTClassifier(maxIter=10)
         gbtModel = gbt.fit(train)
         # Testing the performance of the Gradient-Boosted Tree classifier using the testing data
         predictions = gbtModel.transform(test)
         predictions.select('label', 'prediction', 'rawPrediction', 'probability').show(10)
```

6. Performance Evaluation and Comparison of methods

The prediction accuracy of the algorithms is done by using a *'BinaryClassificationEvaluator()'* on the predictions obtained from the algorithm's trees. The accuracy used in the evaluation in these algorithms is Receiver Operating Characteristic (ROC) value of the predictions which ranges from 0 to 1 which is a plot of the false positive versus true positive rate of the predictions. Given below are the accuracy results of Decision Tree, Random Forests and Gradient Boosted Trees.

	DECISION TREE	Random Forest	Gradient-Boosted
	+----------+-----+	+----------+-----+	+----------+-----+
	|prediction|count|	|prediction|count|	|prediction|count|
	+----------+-----+	+----------+-----+	+----------+-----+
	| 0.0| 1614|	| 0.0| 1986|	| 0.0| 2082|
	| 1.0| 2701|	| 1.0| 2329|	| 1.0| 2233|
	+----------+-----+	+----------+-----+	+----------+-----+
Test under ROC:	0.71832	0.79483	0.80697

The basic level decision tree gives the lowest accuracy of 0.7183 but they are easy to interpret since they produce only 1 tree. An improved accuracy can be achieved by tuning the model with bagging (Random Forests) gives 0.7984 and boosting (GBT) gives 0.8069. In thirst of a higher accuracy rate, the level of interpretation of trees becomes more complex, difficult and time consuming taking up more computational power and as also the number of trees produced (RF and GBT) and features increase. As we can also see that the number in predicting 0.0s increases with increasing accuracy in the algorithms. This means that if we want to know how the algorithm is working and want to interpret the model w.r.t the patient's survival, then we can use decision trees. But if a boost is needed compromising this interpretability, then we can use Random Forest or GBT. On a comparative note, Random forests are the most efficient for prediction if there is less need for interpretation and want better performance since Gradient-boosted takes more time, harder to tune and easily overfits. Hence, random forests sit at the perfect intersection. Also, the removal of outliers might have lost some critical information required to train the algorithms. Therefore, there is a possibility that there could be a boost in all algorithms considering the inclusion of outliers

because the trees are not sensitive to outliers (Kho, 2018). The ROC value of all the group members prediction was shared for comparison. Mr. Uzoma's neural network prediction had an ROC value of 0.922 which is very appealing. Although, critical to note that neural networks are very hard to interpret as they work on a black box mechanism. Ms. Ferdows shared her result of 0.604 ROC value of decision trees, wheres after one-hot-encoding it turned out to be 1.0 which is too good to be true for decision trees. This must be because of overfitting of the labels when encoding the data. Mr. Avi shared his K-NN results of 0.912 which is quite impressive for this dataset too. In my opinion, Mr. Avi's K-NN algorithm is the best or efficient one because it has high accuracy and interpretation of the prediction could be made even though it is not easy. Mr. Uzoma's neural network algorithm has high accuracy too but because of the hidden layers in neural networks, they become rigid for interpretation. More could be said if more information about the method or technique they used (i.e. sampling, tuning parameters) were shared.

7. Discussion and findings

In this report, we first explored the dataset seeing how the different parameters recorded when a patient is admitted interact with each other and *'hospital_death'* to predict if the patient would survive or not. The exploratory data analysis brought light on how different parameters like blood pressure, heartrate, respiration and temperature affect the patient's body and also saw how they are correlated with each other. They also contribute to predicting the patient's survival. In the data preparation stage, there are many identifiers like icu_id, hospital_id, etc., which are recognised as numeric variables. The removal of outliers might have lost some critical information for prediction, thus including them intuits the possibility of getting a higher prediction accuracy in all algorithms. The correlation in the parameters affecting *'hospital_death'* are also an important thing to notice because it tells us about the predictors for patient's survival. While doing cluster analysis, it's the analyst's choice to choose the number of 'k's for cluster formation. Another suggestion that could be done towards sampling of data is to include SMOTE undersampling or oversampling of classes which synthetically generates data accordingly using 'k-NN'. Using SMOTE, we can increase the recall of prediction at the cost of precision. Although generation of synthetic data is synthetic, SMOTE balances the majority and minority classes. Using such predictive analysis in the healthcare industry, the patient's hour 1 and day 1 records are fed into algorithms with the kind of disease they are infected with. These algorithms can predict the patient's survival or how worse the patient's condition is. This helps in assisting the doctors in treating the patients more closely and create a plan to treat the patients more effectively.

8. Data Management Plan

The dataset is 31.41 MB CSV file accessed from public-source website Kaggle which is contains data about patients admitted. This data is used in the report to predict whether a patient will survive or not. No new data is collected as it is static. The data is backed up on group members laptop and shared OneDrive folder. The data is accessible to all group members and further accessibility would be given to the Brunel University which could be saved up to 10 years.

9. Author Contribution Statement

The data cleaning in R was done by myself and Ms. Ferdoos. In python, PCA was done by me and Ms. Ferdoos while K-Means Clustering was done by Mr. Avi Sharma. Very little or mere effort and contribution was seen towards group work from Mr. Uzoma and Mr. Abeiku in Data cleaning and Exploratory data analysis with poor communication from their side.

References:

- Bowne-Anderson, H. (2020). The unreasonable importance of data preparation. [online] O'Reilly Media. Available at: https://www.oreilly.com/radar/the-unreasonable-importance-of-data-preparation/ [Accessed 17 Apr. 202
- Rengaraju, U. (2020). Handling Missing Values. [online] Medium. Available at: https://medium.com/wids-mysore/handling-missing-values-82ce096c0cef.2].
- choonghyunryu.github.io. (n.d.). Data quality diagnosis. [online] Available at: https://choonghyunryu.github.io/dlookr/articles/diagonosis.html [Accessed 17 Apr. 2022].
- Harshitha Mekala (2018). Dealing with Missing Data using R. [online] Medium. Available at: https://medium.com/coinmonks/dealing-with-missing-data-using-r-3ae428da2d17.
- Bhalla, D. (n.d.). Missing Imputation with MICE Package in R. [online] ListenData. Available at: https://www.listendata.com/2015/08/missing-imputation-with-mice-package-in.html [Accessed 18 Apr. 2022].
- Banghart, M. (n.d.). 4.9 Duplicate observations | Data Wrangling Essentials. [online] sscc.wisc.edu. Available at: https://sscc.wisc.edu/sscc/pubs/DWE/book/4-9-duplicate-observations.html [Accessed 18 Apr. 2022].
- Kwak, S.K. and Kim, J.H. (2017). Statistical data preparation: management of missing values and outliers. Korean Journal of Anesthesiology, [online] 70(4), p.407. Available at: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5548942/.
- Stats and R. (n.d.). Outliers detection in R. [online] Available at: https://statsandr.com/blog/outliers-detection-in-r/.
- Franklin, S.J. (2019). Effect of outliers on K-Means algorithm using Python. [online] Analytics Vidhya. Available at: https://medium.com/analytics-vidhya/effect-of-outliers-on-k-means-algorithm-using-python-7ba85821ea23#:~:text=Since%20K%2DMeans%20algorithm%20is [Accessed 19 Apr. 2022].
- Jr, T.T. (2022). *PCA 102: Should you use PCA? How many components to use? How to interpret them?* [online] Medium. Available at:

https://towardsdatascience.com/pca-102-should-you-use-pca-how-many-components-to-use-how-to-interpret-them-da0c8e3b11f0.

- Greener, R. (2020). *Stop testing for normality*. [online] Medium. Available at: https://towardsdatascience.com/stop-testing-for-normality-dba96bb73f90 [Accessed 20 Apr. 2022].
- Ashwini Kumar Pal (2017). *Understanding Dimension Reduction with Principal Component Analysis (PCA)*. [online] Hello Paperspace. Available at: https://blog.paperspace.com/dimension-reduction-with-principal-component-analysis/
- Li, L. (2019). Principal Component Analysis for Dimensionality Reduction. [online] Medium. Available at: https://towardsdatascience.com/principal-component-analysis-for-dimensionality-reduction-115a3d157bad.
- Sanatan Mishra (2017). *Unsupervised Learning and Data Clustering*. [online] Medium. Available at: https://towardsdatascience.com/unsupervised-learning-and-data-clustering-eeecb78b422a.
- Blogs & Updates on Data Science, Business Analytics, AI Machine Learning. (2020). *Types of Clustering Algorithms in Machine Learning With Examples*. [online] Available at: https://www.analytixlabs.co.in/blog/types-of-clustering-algorithms/.
- Kumar, S. (2021). Clustering Algorithm for data with mixed Categorical and Numerical features. [online] Medium. Available at: https://towardsdatascience.com/clustering-algorithm-for-data-with-mixed-categorical-and-numerical-features-d4e3a48066a0.
- https://towardsdatascience.com/silhouette-method-better-than-elbow-method-to-find-optimal-clusters-378d62ff6891
- Bhardwaj, A. (2020). Silhouette Coefficient: Validating clustering techniques. [online] Medium. Available at: https://towardsdatascience.com/silhouette-coefficient-validating-clustering-techniques-e976bb81d10c.
- Team, G.L. (2020). Random Forest Algorithm- An Overview | Understanding Random Forest. [online] GreatLearning. Available at: https://www.mygreatlearning.com/blog/random-forest-algorithm/.
- Analytics Vidhya. (2021). *What is Imbalanced Data | Techniques to Handle Imbalanced Data*. [online] Available at: https://www.analyticsvidhya.com/blog/2021/06/5-techniques-to-handle-imbalanced-data-for-a-classification-problem/.

- Investopedia. (n.d.). *Reading Into Stratified Random Sampling*. [online] Available at: https://www.investopedia.com/terms/stratified_random_sampling.asp#:~:text=The%20main%20advantage%20of%20stratified.
- Ravi, R. (2019). One-Hot Encoding is making your Tree-Based Ensembles worse, here's why? [online] Medium. Available at: https://towardsdatascience.com/one-hot-encoding-is-making-your-tree-based-ensembles-worse-heres-why-d64b282b5769#:~:text=In%20general%2C%20one%20hot%20encoding [Accessed 22 Apr. 2022].
- Kraan, R. (n.d.). Demystifying decision trees, random forests & gradient boosting. [online] www.vantage-ai.com. Available at: https://www.vantage-ai.com/en/blog/demystifying-decision-trees-random-forests-gradient-boosting [Accessed 22 Apr. 2022].
- Kho, J. (2018). Why Random Forest is My Favorite Machine Learning Model. [online] Medium. Available at: https://towardsdatascience.com/why-random-forest-is-my-favorite-machine-learning-model-b97651fa3706.