Predicting Length of Stay at Hospitals using MIMIC data

Group 11:

Jaya Sindhu Yannam
Shikhar Bharat Jain
Priyank Nilesh Gandhi
Siddharth Nandargi

Member contribution statement:

Shikhar, Jaya, Priyank and Siddharth conceived of the presented idea. Jaya conducted the initial Literature Survey of existing work related to the chosen topic. Priyank and Siddharth performed an analysis of the existing data mining methods and implemented in Python. Shikhar performed the Regression algorithm and concluded that the model performance is not so great. All team members mutually agreed to proceed with implementing classification algorithms to improve the prediction accuracy. Shikhar implemented SMOTE to balance the imbalance dataset and proceeded to implement classification algorithms. Jaya and Priyank performed the EDA analysis and created visualizations which helped to understand the important features/attributes. Shikhar and Siddharth worked on implementing the Recursive Feature Selection methodology. Jaya, Priyank and Siddharth implemented 3 classification algorithm on the sampled dataset in Weka software. Shikhar implemented 2 classification algorithms using Python. All members discussed the results and contributed to the final report development.

I. Introduction

The length of stay (LOS) is the period a patient is admitted to a hospital or other similar medical facility. Moreover, LOS is correlated with disease severity and mortality. Our goal is to provide an initial length of stay prediction of new coming patients using a numerical and categorical database as a starting point. By using data mining algorithms, this prediction is made achievable. The data mining techniques are employed to make sure the procedure is reliable and lessen the effect of certainty. Moreover, studies have found patients that, at an emergency department (ED) are associated with a longer LOS and patients who develop further complications in intensive care units (ICU) have a longer LOS beforehand at the ED. To optimize healthcare planning and resources while minimizing the impact of uncertainty, the LOS must be predicted.

One of the most crucial medical resources is hospital beds, which are frequently regarded as an important indicator of hospital performance and an accurate predictor of the state of the local healthcare system's development. Patients' length of stays and hospital costs are strongly related because most hospitals have a limited supply of beds. Therefore, cutting the length of stay can save medical costs and the social burden of medical care while simultaneously increasing the turnover of inpatients. Finding the pertinent risk variables associated with patient recovery and length of stay is crucial for the medical system. As a result, one of the biggest issues facing hospital management right now is how to better allocate hospital beds and address the bed shortage.

The issue of bed allocation has not been fully solved, and there are still a lot of hospitals that need beds due to a lack of medical resources in hospitals and a number of unpredictable events during treatment. Therefore, estimating the length of stay precisely will aid in logical bed allocation and boost bed usage. An important measure of hospital management is how long patients stay there. Its prediction calls for the use of statistical techniques to summarize, analyze, and research its distribution law and change rule as well as the use of machine learning algorithms to create models that forecast the length of hospital stay.Studies contributing to LOS have regularly appeared in the literature. One study conducted determine the factors affecting LOS in the public hospital in Iran. Demonstrated that, an increase in the age would lead to increase in average LOS, Secondly, the average LOS of men is longer than women. Data mining algorithms such as Artificial Neural Network (ANN), Support Vector Machine (SVM), and Decision trees were used for this study.

II. Methods

a. Data source

A crucial component of this project is to be able to analyze the large amount of data contained in Electronic Medical Records (EMRs). Our dataset of choice was the MIMIC III database which houses over 50,000 actual records of people that have been admitted to ICU units between 2001 and 2012. As one would expect the MIMIC database comprises of anonymized EMRs with the data stored in a relational database format. The Data tables that we have used for our analysis and predictions are as follows:

A) **PATIENTS**: This table has 42,520 number of instances, and links to ADMISSIONS table as well as ICUSTAYS on "Subject_ID"

B) **ICUSTAYS**: ICUSTAYS tables contains 61,532 rows in it and has links to PATIENTS table on "Subject_ ID", and ADMISSIONS table on "HADM_ID".

C) **ADMISSIONS**: The number of rows that the ADMISSIONS table contains is 58,976. And this table has links to PATIENTS table on "Subject_ID" .

D) **DIAGNOSES_ICD**: This table has 651,047 rows, and it has links to PATIENTS on "Subject_ID", ADMISSIONS on "HADM_ID".

b. Data preprocessing:

Data preprocessing involves various steps that need to be followed to prepare the data appropriately for the model building and training. These steps depend on the type of dataset we are working on, and this varies from data to data.

Firstly, we checked the data for any null values present. Upon checking that, we found that 3% of the data is missing so we removed those instances from the dataset. The length of stay variable (LOS) represented the number of days a patient stayed in the hospital. Upon examining this variable, it was found that few of the data values were negative, and the number of days cannot be negative, so we removed such values from the data tables, as a part of data cleaning process.

The admission type variable had 4 types, but it seems that “Emergency and “Urgent” are one and the same, so the “Urgent” values were replaced with “Emergency” and the value counts were normalized.

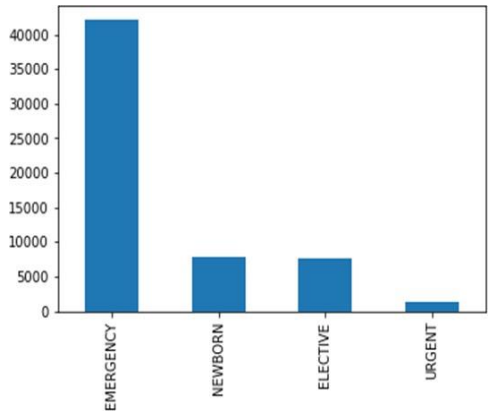


Fig. 1 Frequency of Admission type

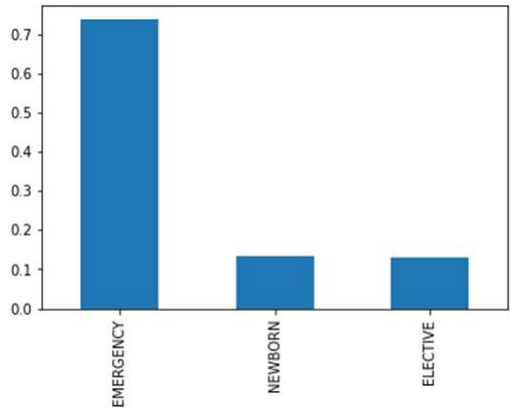


Fig. 2 After converting urgent category to emergency

As LOS is the target variable, it was important to check the distribution of this variable across the dataset. A histogram was generated, along with a frequency curve.

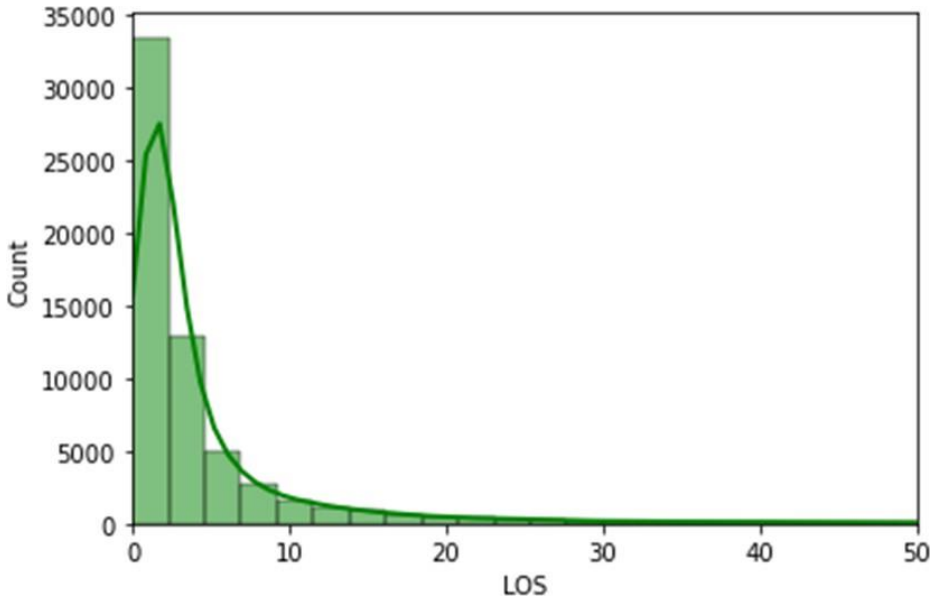


Fig. 3 Distribution of LOS variable

The above histogram suggests that the data values in LOS variable are not normally distributed, and the distribution is heavily skewed towards the right. Hence, we can conclude from this that most of the patients admitted had their length of stay in the hospital to be less than 10 days.

There was no column containing the ages of patients. So, we created a new column containing Ages of patient by using data values from 2 other columns like DOB from PATIENTS Table and ADMITTIME from ADMISSIONS table

A boxplot for Age variable was also generated, and a table containing descriptive analysis of Age was prepared.

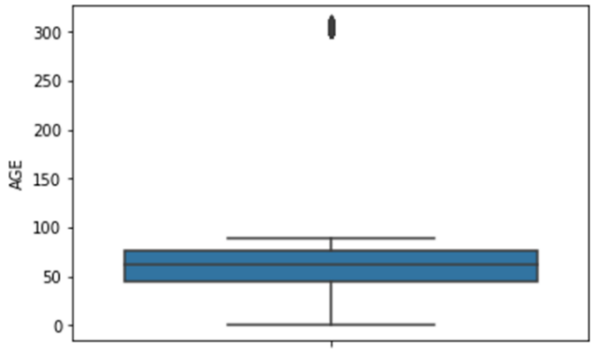


Fig. 4 Boxplot for Age variable

count	61532.000000
mean	64.925951
std	56.951788
min	0.000000
25%	44.374658
50%	62.112329
75%	76.101370
max	311.767123

Fig. 5 Descriptive analysis of Age

This boxplot helped in detecting the outliers in the age variable. And on examining these outliers, it was found that these outliers were irrelevant for the analysis and prediction but were present in a very small negligible amount across the data.

For regression analysis, min-max scaler was used to scale the data and for classification technique, we binarized the response variable where 0 represents LOS=< 10 and 1 represents LOS>10 days (about 1 and a half weeks).

Distribution of the death variable based on the genders was generated to check what gender type had most deaths and vice versa.

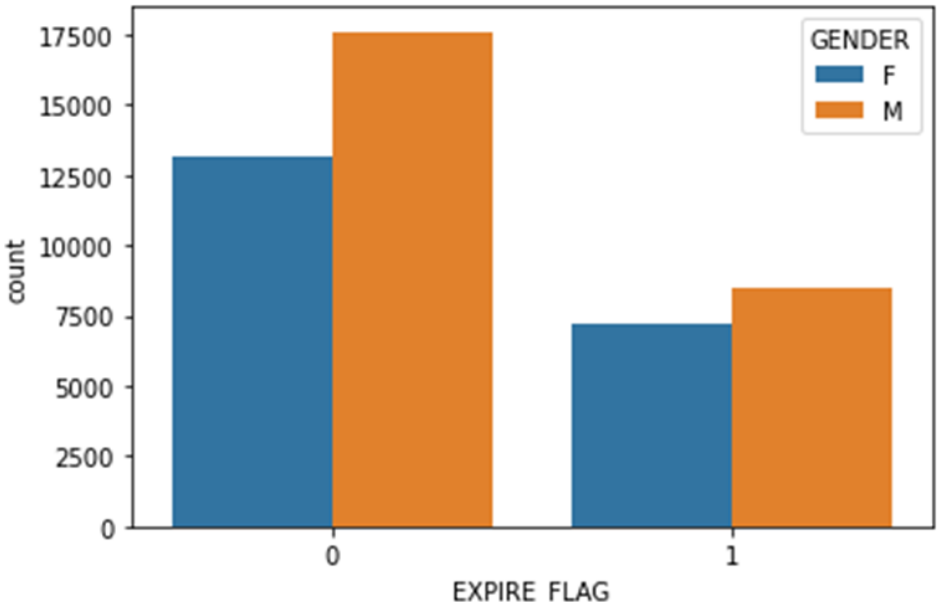


Fig. 6 Distribution of death variable based on gender

Here it can be seen that Males were the most who died and the same can be seen for alive. This indicates that the number of females admitted were less than that of male patients.

Later, to increase the interpretability for each ICD-9 code we mentioned the corresponding diseases and plotted it using a bar graph.

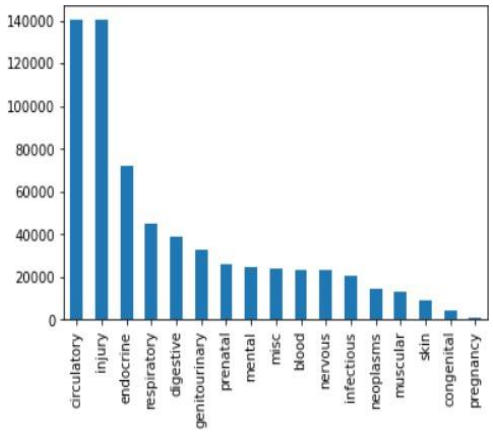


Fig. 7 ICD-9 Codes and diseases

	ROW_ID	SUBJECT_ID	HADM_ID	SEQ_NUM	ICD9_CODE	codetype
0	1297	109	172335	1.0	40301	circulatory
1	1298	109	172335	2.0	486	respiratory
2	1299	109	172335	3.0	58281	genitourinary
3	1300	109	172335	4.0	5855	genitourinary
4	1301	109	172335	5.0	4254	circulatory

Fig.8 ICD-9 Code Diseases

From the bar graph, it is evident that circulatory diseases are more prominent in ICU patients.

c. Feature selection process

There are various techniques that we can use for selecting features from the dataset to get accurate predictions after training the model. There needs to maintain a balance between model complexity and model simplicity.

Too many features make a model complex, while less features make the model very simple, which would hinder the results. So, selecting appropriate number of features is very crucial.

For this we have used “Recursive Feature Elimination” technique.
In this, the least important features are eliminated at each feature elimination step, for this feature importance metrics are used, to select the least important features for model building.

In general, feature selection techniques are used to get rid of unnecessary and redundant attributes that don't contribute much to the predictive model's accuracy or, in certain cases, even make it less accurate. The accuracy of the prediction model may increase because of feature selection, but it will also be faster and more cost-effective because the model will be less complex.

We chose all the variables such as Gender_M, nervous, neoplasams, misc, pregnancy, prenatal, respiratory, skin, age, muscular, mental, Gender_F, circulatory, blood, injury, congenital, digestive, endocrine, genitourinary, infectious, ICU, NICU etc. Removing all death related columns because we are finding length of stay and these patients were never discharged.

We also used Weka to see what attributes Weka suggests based on the Attribute Rankings. Classifier Feature Evaluator was used by using Attribute Ranking.

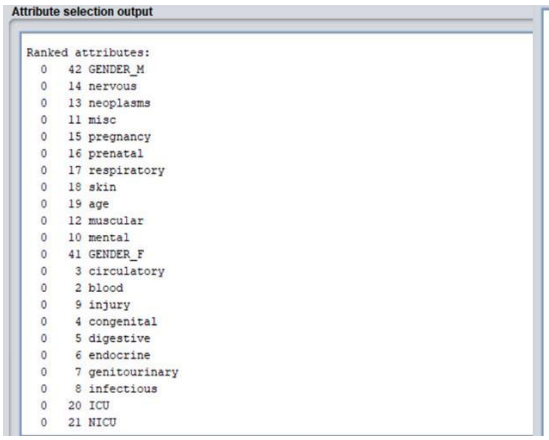


Fig.9 Attribute Selection in Weka

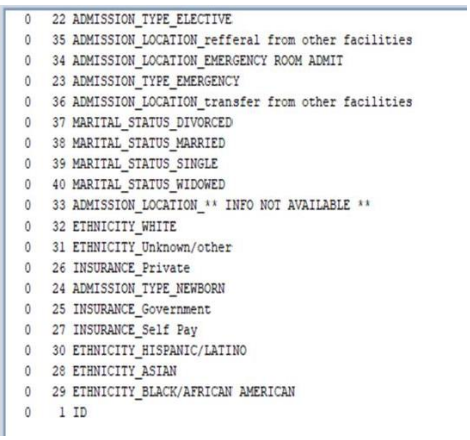


Fig.10 Attribute Selection in Weka

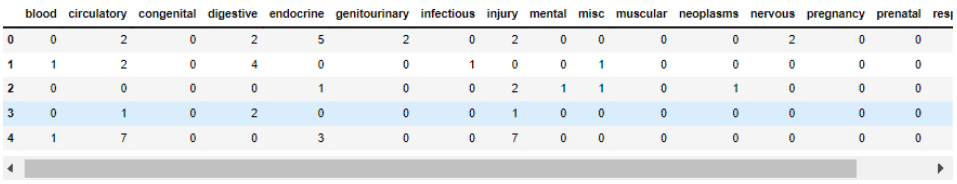
d. Description of dependent and independent variables:

A dependent variable here is Length of Stay.i.e., **LOS.** We are predicting the LOS variable of a patient in ICU. The histogram that we generated (Fig.) for LOS depicted that the LOS variable is not normally distributed and has most of its data values under 10 days (about 1 and a half weeks).

For classification problem, we have converted this variable into a binary variable, where 1 means LOS>10 days, and 0 means LOS<=10 days.

There are numerous independent variables present in the dataset which help in predicting the data for LOS variable. These variables differ based on the type of problem. For classification problems there are a different set of variables than in the regression problem.

Various dummy variables were also generated in the process, to help for better model building and prediction.



	blood	circulatory	congenital	digestive	endocrine	genitourinary	infectious	injury	mental	misc	muscular	neoplasms	nervous	pregnancy	prenatal	resp
0	0	2	0	2	5	2	0	2	0	0	0	0	2	0	0	
1	1	2	0	4	0	0	1	0	0	1	0	0	0	0	0	
2	0	0	0	0	1	0	0	2	1	1	0	1	0	0	0	
3	0	1	0	2	0	0	0	1	0	0	0	0	0	0	0	
4	1	7	0	0	3	0	0	7	0	0	0	0	0	0	0	

Fig.11 Creation of Dummy Variables

```
diag_dummy = diag_dummy.join(diag['HADM_ID'], how="outer")
diag_dummy.head()
```

	blood	circulatory	congenital	digestive	endocrine	genitourinary	infectious	injury	mental	misc	muscular	neoplasms	nervous	pregnancy	prenatal	res
0	0	2	0	2	5	2	0	2	0	0	0	0	2	0	0	
1	1	2	0	4	0	0	1	0	0	1	0	0	0	0	0	
2	0	0	0	0	1	0	0	2	1	1	0	1	0	0	0	
3	0	1	0	2	0	0	0	1	0	0	0	0	0	0	0	
4	1	7	0	0	3	0	0	7	0	0	0	0	0	0	0	

Fig.12 Creation of Dummy Variables

Descriptive statistics of all the variables is as follows.

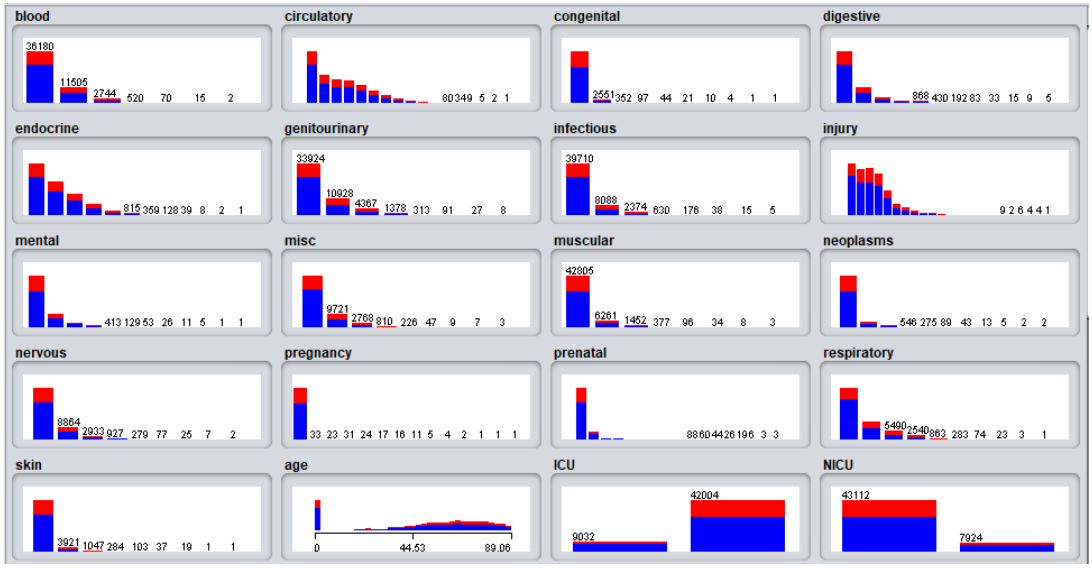


Fig.13 Statistical analysis for Independent Variables

Fig.14 Statistical analysis for Independent Variables

III. Results and Discussion

Random Forest is a versatile machine learning method capable of performing both regression and classification model. The random forest starts with a standard machine learning technique called “decision trees”. In a decision tree, an input is entered at the top and as it traverses down the tree the data gets put into smaller and smaller sets or classes. The random forest goes a step beyond by combining multiple trees to produce the mode class, in the case of classification, or, in the case of regression, a mean prediction by averaging the results of the individual trees. In addition, random forests correct for decision trees’ habit of over fitting to their training set. When a new input is entered into the system, it is run down all the trees. In the case of a numerical variable, the result may either be an average or weighted average of all the terminal nodes that are reached. However, if the input variable is categorical, then the result will be a voting majority.

Initially we implemented the Regression technique on our dataset after splitting it into 80% test and 20% train data, and the following results were obtained. Here, we have used the Decision Tree and Random Forest models for the regression tasks. Decision Trees and Random Forest models can be used for both – Classification and Regression. The metrics used here are R2 (R-square) and MSE (Mean Square Error) rate. R square explains how well the regression model explains the observed data which is also known as goodness of fit measure for linear regression models. R-square value measures the strength of relationship between your model and the dependent variable on a convenient scale of 0-100% or 0 to 1. We note the our R-square values obtained for the two models implemented yield a value close to 0.5 which is considered to be a good fit. Higher the R-square value does not necessarily mean that the model fits well, but it means the model is overfitting. It is also noted that the differences between the observed and the predicted variables are small and unbiased.

Test Set

Model	R2	MSE
Decision Tree	0.3982	0.0011
Random Forest	0.4070	0.0010

Table 2

Mean Square Error (MSE) signifies the average square residual. MSE represents the error of the predicted model created based on the given set of observations in the sample. To put it in other words, MSE shows the cost incurred while making each prediction. There is no correct value for MSE. Simply put, the lower the value the better and 0 means the model is perfect. Since there is no correct answer, the MSE's basic value is in selecting one prediction model over another. MSE is a metric to know how close you got to the perfect fit. The model is often fitted by minimizing MSE, so you'd like to know how close you got to ZERO in the end. You can use this metric later to compare models, those with smaller MSE fit better to the data.

Next, we proceeded onto implementing Classification algorithms using *sklearn* library in Python where we considered the same split of dataset as we did while implementing Regression techniques. We implemented the Decision Tree and Logistic Regression as a classifier and obtained the following results.

Model	Accuracy	Sensitivity	Specificity
Decision Tree	0.7442	0.7157	0.7728
Logistic Regression	0.6402	0.6768	0.6036

Train Set

Model	Accuracy	Sensitivity	Specificity
Decision Tree	0.7200	0.9405	0.7760
Logistic Regression	0.5643	0.4777	0.6033

Test Set

Fig 15. Accuracy Metrics

We achieved a F1score of 0.793 for Decision tree classifier algorithm while for Logistic Regression we achieved 0.394.

We considered Accuracy, Sensitivity, F1 Score and Specificity as metrics to evaluate the classification models. Our sensitivity describes how well our test catches all our positive cases. Sensitivity is calculated by dividing the number of true-positive results by the total number of positives (which include false positives).

Our specificity describes how well our test classifies negative cases as negatives. Specificity is calculated by dividing the number of true-negative results by the total number of negatives (which include false negatives).

Our study involves implementing 3 other classification models for the sake of comparison through which we could infer the best model that suits our dataset. For this, we used the Weka software and the results for the classification algorithms implemented are as follows. It is to be noted that the Weka software could not process algorithms on the complete dataset involved in our study, hence we have performed a random sampling in Weka, and considered 15% of data for implementing these algorithms. After sampling, the total instance count was 7655.

The three classification algorithms implemented in Weka were:

1. Naïve Bayes - every pair of features being classified is independent of each other
2. J48 - J48 is a machine learning decision tree classification algorithm based on Iterative Dichotomiser 3. It is very helpful in examine the data categorically and continuously
3. A random forest is a meta estimator that fits several decision tree classifiers on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting

	Classification Algorithm on Test Set			
Model	Accuracy	AUC	F measure	Software
Naïve Bayes	72%	0.757	0.721	Weka
J48	71%	0.660	0.710	Weka
Random Forest	74%	0.779	0.734	Weka

Apart from Accuracy as a metric, we have considered AUC which stands for Area under the ROC curve. AUC is used for model comparison. AUC value of 1 infers a perfect model which might be a true case of overfitting and a value under 0.5 is not so favored. Our results have AUC values greater than 0.5 and less than 1 which concludes a good fitting model. Amongst the three models,

Random Forest performs the best basis on accuracy and AUC values. Next in order is Naïve Bayes and the least performing model is J48.

Strengths and limitations of your study

1. Mimic dataset does not provide any data for patients aged between 7 to 13 years.
2. Weka software could not handle large datasets and we sampled the data to perform Classification for few of the approaches.
3. MIMIC contains a vast data set and the tables in it. A careful study of all the tables and variables needs to be conducted to join the tables to form the required dataset to build the model. Extracting and joining tables is a tedious process.
4. Classification models perform better than the Regression models which has been proved with the metrics chosen while implementing models.
5. Our data does not include any bias as we did not focus on a particular race/ethnicity, nor patients with certain diseases, nor based on the resources of the hospitals, etc.
6. We followed the best Feature selection methodology as we note the models have given good accuracy and specificity values, which confirms our model fits the data best and does not lead to overfitting.
7. Initially our dataset was imbalanced which would yield to biased and inaccurate models. To improve this, we performed SMOTE (Synthetic Minority Over-Sampling technique) and balanced our dataset to ensure the models are accurate which was done by upsampling the minority class.

IV. Conclusion

The scope of our project showcases how Classification algorithms improves the drawbacks of Regression on the MIMIC dataset. We can infer from the scores of Specificity and Sensitivity obtained from the Classification and Regression models in Python that the models we built are good for predicting the LOS. Predicting LOS is an example of how EHR data can be potentially used for good by applying data mining techniques. The MIMIC database offered surprisingly good depth and detail related to medical admissions which enabled me to create a hospital length-of-stay prediction model that considered a lot of interesting input features. The most surprising aspect of this work was how the patient ICD-9 diagnoses played a more important role than age when predicting the length-of-stay.

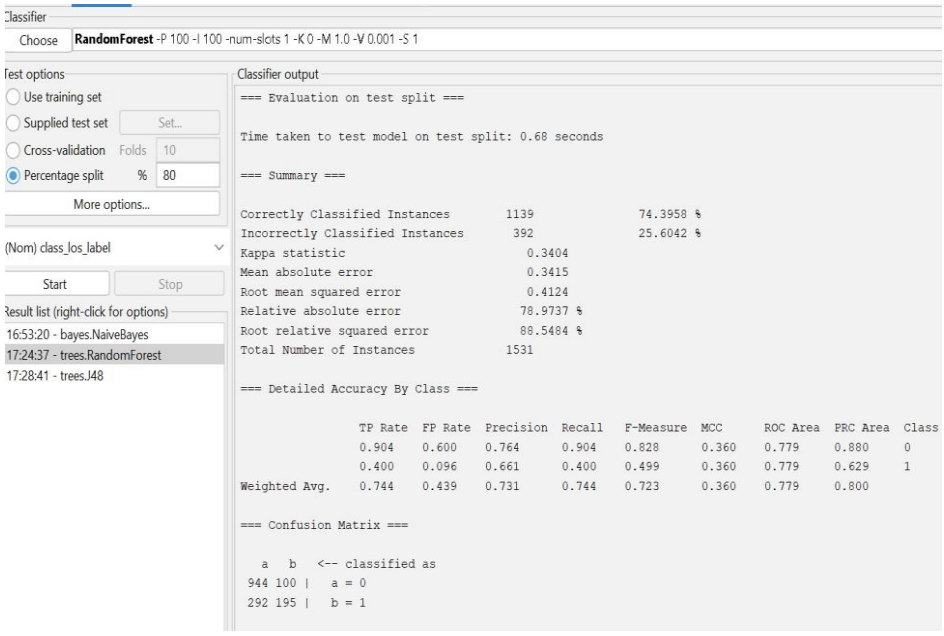
Future work:

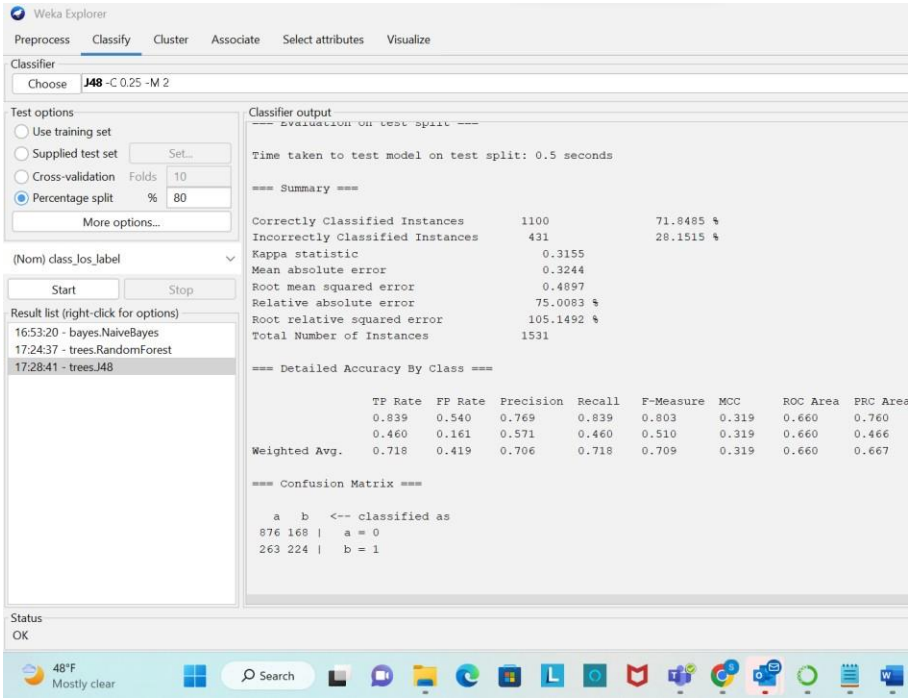
This way of predicting Length of Stay outcome in patients admitted in hospitals can be used effectively. We used random forest models to predict LOS at the ED since this model is well suited for treating data with complex interactions between variables and other non-linear effects. Models based on Deep Neural Networks are another option that could be explored in further studies.

References:

1) *Nouaouri, A. Samet and H. Allaoui, "Evidential data mining for length of stay (LOS) prediction problem," 2015 IEEE International Conference on Automation Science and Engineering (CASE), 2015, pp. 1415-1420, doi: 10.1109/CoASE.2015.7294296.*
2) *Hachesu PR, Ahmadi M, Alizadeh S, Sadoughi F. Use of data mining techniques to determine and predict length of stay of cardiac patients. Healthc Inform Res. 2013 Jun;19(2):121-9.doi: 10.4258/hir.2013.19.2.121.Epub 2013 Jun 30. PMID: 23882417; PMCID: PMC3717435*
3) *Rathor, R., & Agarkar, P. (2015). LOSH Prediction using Data Mining. International Journal of Computer Applications, 119.*
4) *Johnson AE;Pollard TJ;Shen L;Lehman LW;Feng M;Ghassemi M;Moody B;Szolovits P;Celi LA;Mark RG; (n.d.). Mimic-III, a freely accessible Critical Care Database. Scientific data. Retrieved November 29, 2022, from https://pubmed.ncbi.nlm.nih.gov/27219127/*
5) *Yong Chen, "Prediction and Analysis of Length of Stay Based on Nonlinear Weighted XGBoost Algorithm in Hospital", Journal of Healthcare Engineering, vol. 2021, Article ID 4714898, 9 pages, 2021. https://doi.org/10.1155/2021/4714898*
6) *Chrusciel, J., Girardon, F., Roquette, L. et al. The prediction of hospital length of stay using unstructured data. BMC Med Inform Decis Mak 21, 351 (2021). https://doi.org/10.1186/s12911-021-01722-4*

Appendix: SQL/STATA/Python/R codes, WEKA screen shots, etc.





```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

import warnings
warnings.filterwarnings("ignore")
```

```
#diag_data=pd.read_csv("C:\\Users\\shikh\\Downloads\\DIAGNOSES_ICD.csv")
#adm_data=pd.read_csv("C:\\Users\\shikh\\Downloads\\ADMISSIONS.csv")
#pat_data=pd.read_csv("C:\\Users\\shikh\\Downloads\\PATIENTS.csv")
#icu_data=pd.read_csv("C:\\Users\\shikh\\Downloads\\ICUSTAYS.csv")

adm_data = pd.read_csv('C:\\Users\\Siddi\\OneDrive\\Documents\\MIMIC Datasets\\mimic-iii-clinical-database-1.4\\A
diag_data = pd.read_csv('C:\\Users\\Siddi\\OneDrive\\Documents\\MIMIC Datasets\\mimic-iii-clinical-database-1.4\\
icu_data=pd.read_csv('C:\\Users\\Siddi\\OneDrive\\Documents\\MIMIC Datasets\\mimic-iii-clinical-database-1.4\\ICU
pat_data=pd.read_csv('C:\\Users\\Siddi\\OneDrive\\Documents\\MIMIC Datasets\\mimic-iii-clinical-database-1.4\\PAT
```

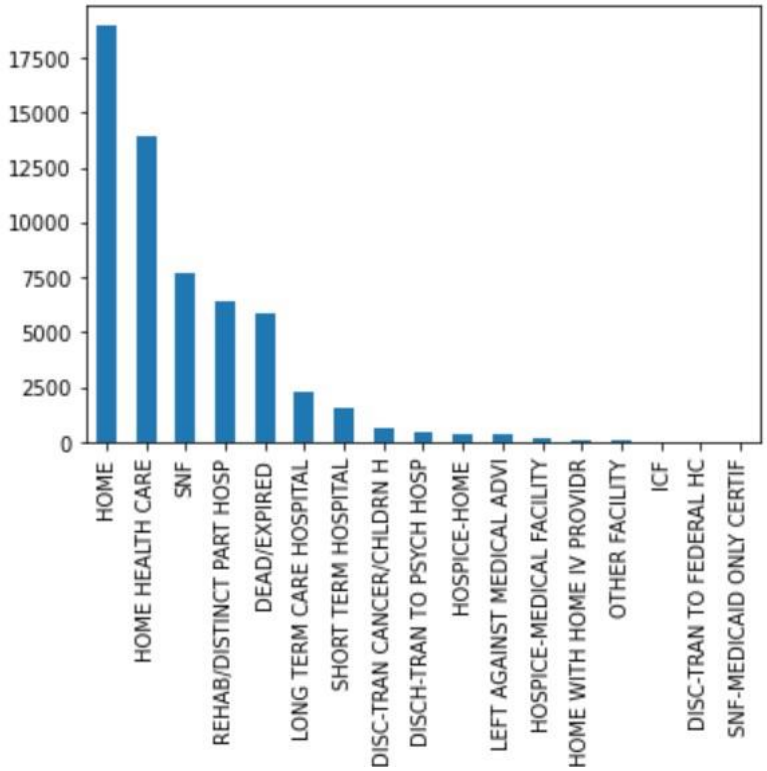
```
diag_data.head()
```

ROW_ID SUBJECT_ID HADM_ID SEQ_NUM ICD9_CODE

```
print(adm_data.info())
print("-----------------------------------------------------------------")
print(icu_data.info())
print("-----------------------------------------------------------------")
print(pat_data.info())
print("-----------------------------------------------------------------")
print(diag_data.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 58976 entries, 0 to 58975
Data columns (total 19 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   ROW_ID                58976 non-null  int64
 1   SUBJECT_ID            58976 non-null  int64
 2   HADM_ID               58976 non-null  int64
 3   ADMITTIME             58976 non-null  object
 4   DISCHTIME             58976 non-null  object
 5   DEATHTIME             5854 non-null   object
 6   ADMISSION_TYPE        58976 non-null  object
 7   ADMISSION_LOCATION    58976 non-null  object
 8   DISCHARGE_LOCATION    58976 non-null  object
```

```
adm_data.DISCHARGE_LOCATION.value_counts().plot(kind="bar")
plt.show()
```



In [30]:
```
# HOME WITH HOME IV PROVIDR and HOSPICE home is also a HOME with health care so replacing its value
```

In [31]:
```
adm_data.DISCHARGE_LOCATION=adm_data.DISCHARGE_LOCATION.apply(lambda x: str(x).replace("HOME WITH HOME IV PROVIDR","HOME HEA
```

In [32]:
```
adm_data.DISCHARGE_LOCATION=adm_data.DISCHARGE_LOCATION.apply(lambda x: str(x).replace("HOSPICE-HOME","HOME HEALTH CARE"))
```

In [33]:
```
# moreover REHAB/DISTINCT PART HOSP,LONG TERM CARE HOSPITAL, SHORT TERM HOSPITAL, DISC-TRAN CANCER/CHLDRN H ,
#DISCH-TRAN TO PSYCH HOSP etc are sent to some other facilities
```

In [34]:
```
a=["SNF","REHAB/DISTINCT PART HOSP","LONG TERM CARE HOSPITAL","SHORT TERM HOSPITAL","DISC-TRAN CANCER/CHLDRN H ","DISCH-TRAN
```

In [35]:
```
for i in a:
    adm_data.DISCHARGE_LOCATION=adm_data.DISCHARGE_LOCATION.apply(lambda x: str(x).replace(i,"sent to other facilities"))

```

In [36]:
```
adm_data.DISCHARGE_LOCATION.value_counts()
```

Out[36]:
```
sent to other facilities                19362
HOME                                    18962
HOME HEALTH CARE                        14432
DEAD/EXPIRED                             5854
LEFT AGAINST MEDICAL ADVI                 365
```

```
adm_data.INSURANCE.value_counts()
```

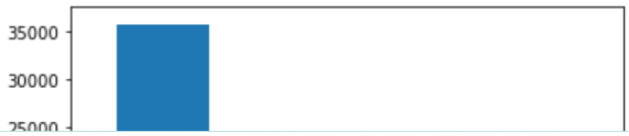
```
Medicare      28214
Private       22582
Medicaid       5785
Government     1783
Self Pay        611
Name: INSURANCE, dtype: int64
```

```
# medicare and mediaid are services offered by government hence converting them
```

```
x=["Medicare","Medicaid"]
```

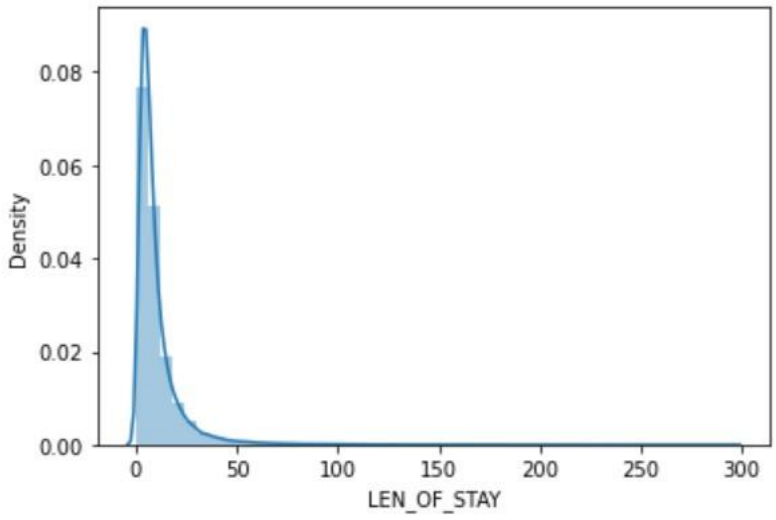
```
for i in x:
    adm_data.INSURANCE=adm_data.INSURANCE.apply(lambda x: str(x).replace(i,"Government"))

```

```
adm_data.INSURANCE.value_counts().plot(kind="bar")
plt.show()
```




```
adm__data=adm_data[adm_data.HOSPITAL_EXPIRE_FLAG==0]
```

```
sns.distplot(adm__data.LEN_OF_STAY)
plt.show()
```




```
# we can length of stay is right skewed
```

```
# list of columns no longer required

m=["ROW_ID","DISCHTIME","DEATHTIME","EDREGTIME","EDOUTTIME",'
```

```
]:  1 diag_data["codetype"]=diag_data.ICD9_CODE
    2 diag_data["codetype"] = diag_data.ICD9_CODE[~diag_data.ICD9_CODE.str.contains("[a-zA-Z]").fillna(False)]
```

```
]:  1 diag_data.head()
```

]:

	ROW_ID	SUBJECT_ID	HADM_ID	SEQ_NUM	ICD9_CODE	codetype
0	1297	109	172335	1.0	40301	40301
1	1298	109	172335	2.0	486	486
2	1299	109	172335	3.0	58281	58281
3	1300	109	172335	4.0	5855	5855
4	1301	109	172335	5.0	4254	4254

```
]:  1 diag_data["codetype"].fillna(value='999', inplace=True)
```

```
]:  1 diag_data['codetype'] = diag_data['codetype'].str.slice(start=0, stop=3, step=1)
    2 diag_data['codetype'] = diag_data['codetype'].astype(int)
```

```
]:  1 code_ranges = [(1, 140), (140, 240), (240, 280), (280, 290), (290, 320), (320, 390),(390, 460), (460, 520), (520, 580), (580
    2               (710, 740), (740, 760), (760, 780), (780, 800), (800, 1000), (1000, 2000)]
```

```
]:  1 code_ranges = [(1, 140), (140, 240), (240, 280), (280, 290), (290, 320), (320, 390),(390, 460), (460, 520), (520, 580), (580
    2               (710, 740), (740, 760), (760, 780), (780, 800), (800, 1000), (1000, 2000)]
```

```
]:  1 diag = {0: 'infectious', 1: 'neoplasms', 2: 'endocrine', 3: 'blood',4: 'mental', 5: 'nervous', 6: 'circulatory', 7: 'respira
    2         8: 'digestive', 9: 'genitourinary', 10: 'pregnancy', 11: 'skin', 12: 'muscular', 13: 'congenital', 14: 'prenatal', 1
    3              16: 'injury', 17: 'misc'}
    4
```

```
]:  1 for i, j in enumerate(code_ranges):
    2     diag_data['codetype'] = np.where(diag_data['codetype'].between(j[0],j[1]),
    3             i,diag_data['codetype'])
```

```
]:  1 diag_data.head()
```

]:

	ROW_ID	SUBJECT_ID	HADM_ID	SEQ_NUM	ICD9_CODE	codetype
0	1297	109	172335	1.0	40301	6
1	1298	109	172335	2.0	486	7
2	1299	109	172335	3.0	58281	9
3	1300	109	172335	4.0	5855	9

0]:
```
diag_data['codetype'] = diag_data['codetype'].replace(diag)
```

1]:
```
diag_data.head()
```

1]:

	ROW_ID	SUBJECT_ID	HADM_ID	SEQ_NUM	ICD9_CODE	codetype
0	1297	109	172335	1.0	40301	circulatory
1	1298	109	172335	2.0	486	respiratory
2	1299	109	172335	3.0	58281	genitourinary
3	1300	109	172335	4.0	5855	genitourinary
4	1301	109	172335	5.0	4254	circulatory

2]:
```
diag_data.codetype.value_counts().plot(kind="bar")
plt.show()
```

]:
```
diag = diag_data.groupby('HADM_ID')['codetype'].apply(list).reset_index()
diag.head()
```

]:

	HADM_ID	codetype
0	100001	[endocrine, nervous, genitourinary, digestive,...
1	100003	[digestive, blood, infectious, digestive, circ...
2	100006	[respiratory, respiratory, respiratory, neopla...
3	100007	[digestive, digestive, injury, respiratory, ci...
4	100009	[circulatory, injury, circulatory, endocrine, ...

]:
```
diag_dummy = pd.get_dummies(diag['codetype'].apply(pd.Series).stack()).sum(level=0)
diag_dummy.head()
```

]:

	blood	circulatory	congenital	digestive	endocrine	genitourinary	infectious	injury	mental	misc	muscular	neoplasms	nervous	pregnancy	prenatal	res
0	0	2	0	2	5	2	0	2	0	0	0	0	2	0	0	
1	1	2	0	4	0	0	1	0	0	1	0	0	0	0	0	
2	0	0	0	0	1	0	0	2	1	1	0	1	0	0	0	
3	0	1	0	2	0	0	0	1	0	0	0	0	0	0	0	
4	1	7	0	0	3	0	0	7	0	0	0	0	0	0	0	

```
df_age_min = model_data[['SUBJECT_ID', 'ADMITTIME']].groupby('SUBJECT_ID').min().reset_index()
df_age_min.columns = ['SUBJECT_ID', 'ADMIT_MIN']
df_age_min.head()
```

	SUBJECT_ID	ADMIT_MIN
0	2	2138-07-17 19:04:00
1	3	2101-10-20 19:08:00
2	4	2191-03-16 00:28:00
3	5	2103-02-02 04:31:00
4	6	2175-05-30 07:15:00

```
model_data= model_data.merge(df_age_min, how='outer', on='SUBJECT_ID')
```

```
model_data['DOB'] = pd.to_datetime(model_data['DOB']).dt.date
```

```
model_data['ADMIT_MIN'] = pd.to_datetime(model_data['ADMIT_MIN']).dt.date
```

```
model_data['age'] = model_data.apply(lambda x: (x['ADMIT_MIN'] - x['DOB']).days/365, axis=1)
```

```
model_data=model_data[model_data.age<100]
```


```
model_data=model_data[model_data.age<100]
```


```
model_data.head()
```

	SUBJECT_ID	HADM_ID	ADMITTIME	ADMISSION_TYPE	ADMISSION_LOCATION	DISCHARGE_LOCATION	INSURANCE	MARITAL_STATUS	ETHNICITY	
0	22	165315	2196-04-09 12:26:00	EMERGENCY	EMERGENCY ROOM ADMIT	sent to other facilities	Private	MARRIED	WHITE	BE
1	23	152223	2153-09-03 07:15:00	ELECTIVE	refferal from other facilities	HOME HEALTH CARE	Government	MARRIED	WHITE	CORC DISEA ART
2	23	124321	2157-10-18 19:34:00	EMERGENCY	transfer from other facilities	HOME HEALTH CARE	Government	MARRIED	WHITE	
3	24	161859	2139-06-06 16:14:00	EMERGENCY	transfer from other facilities	HOME	Private	SINGLE	WHITE	
4	25	129635	2160-11-02 02:06:00	EMERGENCY	EMERGENCY ROOM ADMIT	HOME	Private	MARRIED	WHITE	ACU

5 rows × 32 columns

```
icu_data.head()
```

```
model_data.ICU=model_data.ICU.astype("object")
```

```
model_data.NICU=model_data.NICU.astype("object")
```

```
drop=["SUBJECT_ID","HADM_ID","ADMITTIME","DOB","ADMIT_MIN"]
```

```
model_data.drop(drop,axis=1,inplace=True)
```

```
model_data.drop("DIAGNOSIS",axis=1, inplace=True)
```

```
model_data.drop("DISCHARGE_LOCATION",axis=1, inplace=True)
```

```
model_data=pd.get_dummies(model_data, columns=['ADMISSION_TYPE','INSURANCE','ETHNICITY', 'ADMISSION_LOCATION', 'MARITAL_STAT
```

```
model_data.isnull().sum()
```

```
LEN_OF_STAY                                    7858
blood                                          7858
circulatory                                    7858
congenital                                     7858
digestive                                      7858
endocrine                                      7858
genitourinary                                  7858
```

```
from sklearn.model_selection import train_test_split
```

```
from sklearn.tree import DecisionTreeRegressor
```

```
dt = DecisionTreeRegressor(random_state=42, max_depth=4, min_samples_leaf=10)
```

```
np.random.seed(0)
df_train, df_test = train_test_split(model_data, train_size=0.8,test_size=0.2, random_state=100,shuffle=True)
```

```
#(df_train)
#df_test
#df_train.to_csv('train2.csv')
df_test.to_csv('test2.csv')
```

```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
```

```
df_train['LEN_OF_STAY'] = scaler.fit_transform(df_train[['LEN_OF_STAY']])
df_test['LEN_OF_STAY'] = scaler.transform(df_test[['LEN_OF_STAY']])
```

```
y_train = df_train.pop("LEN_OF_STAY")
X_train = df_train

```

```
X_test.shape, X_train.shape
```

```
((10208, 41), (40828, 41))
```

```
print((df_train))
#print(len(df_test))
```

```
       blood  circulatory  congenital  digestive  endocrine  genitourinary  \
13105    0.0          0.0         0.0        0.0        0.0            0.0
46351    0.0          1.0         0.0        1.0        1.0            1.0
6283     0.0          0.0         0.0        0.0        3.0            0.0
4808     1.0          3.0         0.0        0.0        2.0            0.0
344      1.0          6.0         0.0        0.0        2.0            0.0
...      ...          ...         ...        ...        ...            ...
16304    0.0          2.0         0.0        0.0        1.0            0.0
79       0.0          3.0         0.0        1.0        0.0            1.0
12119    0.0          0.0         0.0        0.0        0.0            0.0
14147    0.0          0.0         0.0        3.0        2.0            0.0
38408    0.0          1.0         0.0        0.0        1.0            1.0

       infectious  injury  mental  misc  ...  \
13105         0.0     3.0     0.0   0.0  ...
46351         0.0     3.0     0.0   0.0  ...
6283          1.0     2.0     1.0   0.0  ...
4808          0.0     1.0     0.0   0.0  ...
344           0.0     1.0     0.0   0.0  ...
...           ...     ...     ...   ...  ...
```

In [149]:
```
dt.fit(X_train, y_train)
```

Out[149]:
```
DecisionTreeRegressor(max_depth=4, min_samples_leaf=10, random_state=42)
```

In [150]:
```
y_train_pred = dt.predict(X_train)
```

In [151]:
```
from sklearn.metrics import r2_score
```

In [152]:
```
r2_score(y_train, y_train_pred)
```

Out[152]: 0.31064417653789667

In [153]:
```
from sklearn.metrics import mean_squared_error
mean_squared_error(y_train, y_train_pred)
```

Out[153]: 0.0012233773256710109

In [154]:
```
y_test_pred = dt.predict(X_test)
```

In [155]:
```
r2_score(y_test, y_test_pred)
```

Out[155]: 0.3601967080069599

]:
```
y_test_pred = dt.predict(X_test)
```

]:
```
r2_score(y_test, y_test_pred)
```

]: 0.3601967080069599

]:
```
from sklearn.metrics import mean_squared_error
mean_squared_error(y_test, y_test_pred)
```

]: 0.0011801236024813495

]:
```
from sklearn.model_selection import GridSearchCV

model = DecisionTreeRegressor()

gs = GridSearchCV(model,
                  param_grid = {'max_depth': range(1, 11),
                                'min_samples_split': range(10, 60, 10)},
                  cv=5,
                  n_jobs=1,
                  scoring='neg_mean_squared_error')

gs.fit(X_train, y_train)

print(gs.best_params_)
```

```
                    cv=5,
                    n_jobs=1,
                    scoring='neg_mean_squared_error')

gs.fit(X_train, y_train)

print(gs.best_params_)
print(-gs.best_score_)
```

```
{'max_depth': 8, 'min_samples_split': 40}
0.001185003465366101
```

58]:
```
new_model = DecisionTreeRegressor(max_depth=8,
                                  min_samples_split=40)

new_model.fit(X_train, y_train)
```

58]:
```
DecisionTreeRegressor(max_depth=8, min_samples_split=40)
```

59]:
```
y_train_pred = new_model.predict(X_train)
```

60]:
```
r2_score(y_train, y_train_pred)
```

60]: 0.408334998845035

```
0.408334998845035
```

```
mean_squared_error(y_train, y_train_pred)
```

```
0.0010500086053830057
```

```
y_test_pred = new_model.predict(X_test)
```

```
r2_score(y_test, y_test_pred)
```

```
0.39821902012769683
```

```
mean_squared_error(y_test, y_test_pred)
```

```
0.0011099910656280029
```

```
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestRegressor
# Create the parameter grid based on the results of random search
param_grid = {
    'bootstrap': [True],
    'max_depth': [80, 90, 100, 110],
    'max_features': [2, 3],
    'min_samples_leaf': [3, 4, 5],
    'min_samples_split': [8, 10, 12]
```

```
    'min_samples_leaf' : [3, 4, 5],
    'min_samples_split': [8, 10, 12],
    'n_estimators': [100, 200, 300, 1000]
}
# Create a based model
rf = RandomForestRegressor()
# Instantiate the grid search model
grid_search = GridSearchCV(estimator = rf, param_grid = param_grid,
                           cv = 3, n_jobs = -1, verbose = 2)
```

```
#grid_search.fit(X_train, y_train)
#grid_search.best_params_
```

```
rf = RandomForestRegressor(bootstrap= True,max_depth=100,max_features= 3,
                           min_samples_leaf= 3,min_samples_split= 10,n_estimators= 300)
```

```
rf.fit(X_train, y_train)
```

RandomForestRegressor

```
RandomForestRegressor(max_depth=100, max_features=3, min_samples_leaf=3,
                      min_samples_split=10, n_estimators=300)
```

```
y_train_pred = rf.predict(X_train)
```

```
y_train_pred = rf.predict(X_train)
```

```
r2_score(y_train, y_train_pred)
```

```
0.4991058021742466
```

```
mean_squared_error(y_train, y_train_pred)
```

```
0.0008889206173709558
```

```
y_test_pred = rf.predict(X_test)
```

```
r2_score(y_test, y_test_pred)
```

```
0.4070452076206904
```

```
mean_squared_error(y_test, y_test_pred)
```

```
0.0010937110740887898
```

```
model_data.head()
```

LEN_OF_STAY	blood	circulatory	congenital	digestive	endocrine	genitourinary	infectious	injury	mental	...	ADMISSION_LOCATION_** INFO NOT AVAILABLE **	ADMISSION_LOCA

```
# we can see data data is imbalanced
```

```
X=class_data.drop(["class_los_label"],axis=1)
```

```
y=class_data["class_los_label"]
```

```
X_train,X_test,y_train,y_test=train_test_split(X, y, train_size=0.8, test_size=0.2,random_state=100, shuffle=True)
```

```
y_train.value_counts(normalize=True)
```

```
0    0.693372
1    0.306628
Name: class_los_label, dtype: float64
```

```
y_test.value_counts(normalize=True)
```

```
0    0.689753
1    0.310247
Name: class_los_label, dtype: float64
```

```
# handling imbalance problem

# SMOTE
from imblearn.over_sampling import SMOTE
```

```
# SMOTE
from imblearn.over_sampling import SMOTE
smt = SMOTE(random_state=27)
X_resampled_smt, y_resampled_smt = smt.fit_resample(X_train, y_train)
len(X_resampled_smt)
```

```
56618
```

```
# Importing decision tree classifier
from sklearn.tree import DecisionTreeClassifier
```

```
import sklearn
from sklearn.feature_selection import RFE
```

```
dtree = DecisionTreeClassifier()
```

```


# Importing RFE

# Intantiate RFE with 15 columns
rfe = RFE(20)

# Fit the rfe model with train set
rfe = rfe.fit(X_resampled_smt, y_resampled_smt)
```

```
dtree = DecisionTreeClassifier()
# Importing RFE
from sklearn.feature_selection import RFE

# Intantiate RFE with 15 columns
rfe = RFE(dtree,n_features_to_select=20)

# Fit the rfe model with train set
rfe = rfe.fit(X_resampled_smt, y_resampled_smt)
```

```
# RFE selected columns
rfe_cols = X_resampled_smt.columns[rfe.support_]
print(rfe_cols)
```

```
Index(['blood', 'circulatory', 'digestive', 'endocrine', 'genitourinary',
       'infectious', 'injury', 'mental', 'misc', 'muscular', 'neoplasms',
       'nervous', 'prenatal', 'respiratory', 'skin', 'age', 'ETHNICITY_WHITE',
       'ADMISSION_LOCATION_refferal from other facilities',
       'MARITAL_STATUS_MARRIED', 'GENDER_F'],
      dtype='object')
```

```
# Importing libraries for cross validation
```

```
# Importing libraries for cross validation
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import GridSearchCV
# Create the parameter grid
param_grid = {
    'max_depth': range(5, 15, 5),
    'min_samples_leaf': range(50, 150, 50),
    'min_samples_split': range(50, 150, 50),
}


# Instantiate the grid search model
dtree = DecisionTreeClassifier()

grid_search = GridSearchCV(estimator = dtree,
                           param_grid = param_grid,
                           scoring= 'recall',
                           cv = 5,
                           verbose = 1)

# Fit the grid search to the data
grid_search.fit(X_resampled_smt[rfe_cols],y_resampled_smt)
```

```
Fitting 5 folds for each of 8 candidates, totalling 40 fits
```

```
Fitting 5 folds for each of 8 candidates, totalling 40 fits
```

GridSearchCV
estimator: DecisionTreeClassifier
DecisionTreeClassifier

```
# cv results
cv_results = pd.DataFrame(grid_search.cv_results_)
cv_results
```

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_max_depth	param_min_samples_leaf	param_min_samples_split	params	split
0	0.155799	0.012851	0.036331	0.006940	5	50	50	{'max_depth': 5, 'min_samples_leaf': 50, 'min_...	
1	0.225724	0.047187	0.052582	0.011328	5	50	100	{'max_depth': 5, 'min_samples_leaf': 50, 'min_...	
2	0.165192	0.038695	0.036868	0.015044	5	100	50	{'max_depth': 5, 'min_samples_leaf': 100, 'min...	
3	0.134946	0.007306	0.031133	0.002559	5	100	100	{'max_depth': 5, 'min_samples_leaf':	

```
print(grid_search.best_estimator_)
```

```
DecisionTreeClassifier(max_depth=5, min_samples_leaf=50, min_samples_split=50)
```

```
dt= DecisionTreeClassifier(max_depth=5, min_samples_leaf=50, min_samples_split=50)

dt.fit(X_resampled_smt[rfe_cols], y_resampled_smt)
```

DecisionTreeClassifier
DecisionTreeClassifier(max_depth=5, min_samples_leaf=50, min_samples_split=50)

```
# Predictions on the train set
y_train_pred = dt.predict(X_resampled_smt[rfe_cols])
```

```
# Confusion matrix
from sklearn import metrics
from sklearn.metrics import confusion_matrix
confusion = metrics.confusion_matrix(y_resampled_smt, y_train_pred)
print(confusion)
```

```
[[21855  6454]
 [ 8015 20294]]
```

```
[[21855  6454]
 [ 8015 20294]]
```

```
TP = confusion[1,1] # true positive
TN = confusion[0,0] # true negatives
FP = confusion[0,1] # false positives
FN = confusion[1,0] # false negatives
```

```
# Accuracy
print("Accuracy:-",metrics.accuracy_score(y_resampled_smt, y_train_pred))

# Sensitivity
print("Sensitivity:-",TP / float(TP+FN))

# Specificity
print("Specificity:-", TN / float(TN+FP))
```

```
Accuracy:- 0.744445229432336
Sensitivity:- 0.7168744922109577
Specificity:- 0.7720159666537144
```

```
# Prediction on the test set
y_test_pred = dt.predict(X_test[rfe_cols])
```

```
# Prediction on the test set
y_test_pred = dt.predict(X_test[rfe_cols])
```

```
# Confusion matrix
confusion = metrics.confusion_matrix(y_test, y_test_pred)
print(confusion)
```

```
[[5460 1581]
 [1268 1899]]
```

```
P = confusion[1,1] # true positive
TN = confusion[0,0] # true negatives
FP = confusion[0,1] # false positives
FN = confusion[1,0] # false negatives
```

```
# Accuracy
print("Accuracy:-",metrics.accuracy_score(y_test, y_test_pred))

# Sensitivity
print("Sensitivity:-",TP / float(TP+FN))

# Specificity
print("Specificity:-", TN / float(TN+FP))

#F1 Score
from sklearn.metrics import f1_score
```

```
Accuracy:- 0.744445229432336
Sensitivity:- 0.7168744922109577
Specificity:- 0.7720159666537144
```

```
# Prediction on the test set
y_test_pred = dt.predict(X_test[rfe_cols])
```

```
# Confusion matrix
confusion = metrics.confusion_matrix(y_test, y_test_pred)
print(confusion)
```

```
[[5460 1581]
 [1268 1899]]
```

```
P = confusion[1,1] # true positive
TN = confusion[0,0] # true negatives
FP = confusion[0,1] # false positives
FN = confusion[1,0] # false negatives
```

```
# Accuracy
print("Accuracy:-",metrics.accuracy_score(y_test, y_test_pred))

# Sensitivity
```

```
# Accuracy
print("Accuracy:-",metrics.accuracy_score(y_test, y_test_pred))

# Sensitivity
print("Sensitivity:-",TP / float(TP+FN))

# Specificity
print("Specificity:-", TN / float(TN+FP))

#F1 Score
from sklearn.metrics import f1_score
print("F1_Score",f1_score(y_test, y_test_pred, pos_label="0"))

```

```
Accuracy:- 0.7209051724137931
Sensitivity:- 0.9411928392542436
Specificity:- 0.7754580315296122
F1_Score 0.7930859176410777
```

```
## Logistic regression
```

```
##### Importing stats model
import statsmodels.api as sm
```

```
X_resampled_smt.ICU=X_resampled_smt.ICU.astype("int")
```

```
X_resampled_smt.NICU=X_resampled_smt.NICU.astype("int")
```

```
X_resampled_smt.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 56618 entries, 0 to 56617
Data columns (total 41 columns):
 #   Column                                          Non-Null Count  Dtype
---  ------                                          --------------  -----
 0   blood                                           56618 non-null  float64
 1   circulatory                                     56618 non-null  float64
 2   congenital                                      56618 non-null  float64
 3   digestive                                       56618 non-null  float64
 4   endocrine                                       56618 non-null  float64
 5   genitourinary                                   56618 non-null  float64
 6   infectious                                      56618 non-null  float64
 7   injury                                          56618 non-null  float64
 8   mental                                          56618 non-null  float64
```

```
y_resampled_smt=y_resampled_smt.astype("int")
```

```
lreg=sm.GLM(y_resampled_smt,(sm.add_constant(X_resampled_smt)),family=sm.families.Binomial())
```

```
# Fit the model
lreg=lreg.fit().summary()
```

```
lreg
```

Generalized Linear Model Regression Results

Dep. Variable:	class_los_label	**No. Observations:**	56618
Model:	GLM	**Df Residuals:**	56576
Model Family:	Binomial	**Df Model:**	41
Link Function:	Logit	**Scale:**	1.0000
Method:	IRLS	**Log-Likelihood:**	-23895.
Date:	Sat, 10 Dec 2022	**Deviance:**	47791.
Time:	21:25:07	**Pearson chi2:**	4.52e+04
No. Iterations:	29	**Pseudo R-squ. (CS):**	0.4185
Covariance Type:	nonrobust		

```
lreg
```

Generalized Linear Model Regression Results

Dep. Variable:	class_los_label	**No. Observations:**	56618
Model:	GLM	**Df Residuals:**	56576
Model Family:	Binomial	**Df Model:**	41
Link Function:	Logit	**Scale:**	1.0000
Method:	IRLS	**Log-Likelihood:**	-23895.
Date:	Sat, 10 Dec 2022	**Deviance:**	47791.
Time:	21:25:07	**Pearson chi2:**	4.52e+04
No. Iterations:	29	**Pseudo R-squ. (CS):**	0.4185
Covariance Type:	nonrobust		

	coef	std err	z	P>|z|	[0.025	0.975]
const	144.6232	7.86e+04	0.002	0.999	-1.54e+05	1.54e+05
blood	0.0625	0.019	3.301	0.001	0.025	0.100
circulatory	0.1111	0.006	17.518	0.000	0.099	0.123
congenital	0.2610	0.037	7.109	0.000	0.189	0.333
digestive	0.2263	0.011	21.044	0.000	0.205	0.247

ETHNICITY_BLACK/AFRICAN AMERICAN	-29.4138	2.89e+04	-0.001	0.999	-5.66e+04	5.66e+04
ETHNICITY_HISPANIC/LATINO	-29.3425	2.89e+04	-0.001	0.999	-5.66e+04	5.66e+04
ETHNICITY_Unknown/other	-29.2404	2.89e+04	-0.001	0.999	-5.66e+04	5.66e+04
ETHNICITY_WHITE	-29.1738	2.89e+04	-0.001	0.999	-5.66e+04	5.66e+04
ADMISSION_LOCATION_ INFO NOT AVAILABLE ****	-30.5316	2.73e+04	-0.001	0.999	-5.35e+04	5.34e+04
ADMISSION_LOCATION_EMERGENCY ROOM ADMIT	-29.0338	2.73e+04	-0.001	0.999	-5.35e+04	5.34e+04
ADMISSION_LOCATION_refferal from other facilities	-29.5240	2.73e+04	-0.001	0.999	-5.35e+04	5.34e+04
ADMISSION_LOCATION_transfer from other facilities	-28.8920	2.73e+04	-0.001	0.999	-5.35e+04	5.34e+04
MARITAL_STATUS_DIVORCED	-1.3159	0.061	-21.438	0.000	-1.436	-1.196
MARITAL_STATUS_MARRIED	-1.1739	0.046	-25.601	0.000	-1.264	-1.084
MARITAL_STATUS_SINGLE	-1.2900	0.049	-26.278	0.000	-1.386	-1.194
MARITAL_STATUS_WIDOWED	-1.3066	0.057	-23.103	0.000	-1.417	-1.196
GENDER_F	-29.1795	3.12e+04	-0.001	0.999	-6.12e+04	6.12e+04
GENDER_M	-29.1409	3.12e+04	-0.001	0.999	-6.12e+04	6.12e+04

]:
```
# Importing logistic regression from sklearn
from sklearn.linear_model import LogisticRegression
# Intantiate the logistic regression
logreg = LogisticRegression()
```

```
# Importing logistic regression from sklearn
from sklearn.linear_model import LogisticRegression
# Intantiate the logistic regression
logreg = LogisticRegression()
```

```
# Intantiate RFE with 15 columns
rfe = RFE(logreg, n_features_to_select=15)

# Fit the rfe model with train set
rfe = rfe.fit(X_resampled_smt, y_resampled_smt)
```

```
rfe_cols = X_resampled_smt.columns[rfe.support_]
print(rfe_cols)
```

```
Index(['INSURANCE_Government', 'INSURANCE_Private', 'INSURANCE_Self Pay',
       'ETHNICITY_ASIAN', 'ETHNICITY_BLACK/AFRICAN AMERICAN',
       'ETHNICITY_HISPANIC/LATINO', 'ETHNICITY_Unknown/other',
       'ETHNICITY_WHITE', 'ADMISSION_LOCATION_** INFO NOT AVAILABLE **',
       'ADMISSION_LOCATION_EMERGENCY ROOM ADMIT',
       'ADMISSION_LOCATION_refferal from other facilities',
       'ADMISSION_LOCATION_transfer from other facilities',
       'MARITAL_STATUS_SINGLE', 'GENDER_F', 'GENDER_M'],
      dtype='object')
```

```
# Adding constant to X_train
```

```
# Adding constant to X_train
X_train_sm_1 = sm.add_constant(X_resampled_smt[rfe_cols])

#Instantiate the model
log_1 = sm.GLM(y_resampled_smt, X_train_sm_1, family=sm.families.Binomial())

# Fit the model
log_1 = log_1.fit()

log_1.summary()
```

Generalized Linear Model Regression Results

Dep. Variable:	class_los_label	**No. Observations:**	56618
Model:	GLM	**Df Residuals:**	56602
Model Family:	Binomial	**Df Model:**	15
Link Function:	Logit	**Scale:**	1.0000
Method:	IRLS	**Log-Likelihood:**	-30318.
Date:	Sat, 10 Dec 2022	**Deviance:**	60636.
Time:	21:25:17	**Pearson chi2:**	4.61e+04
No. Iterations:	29	**Pseudo R-squ. (CS):**	0.2705
Covariance Type:	nonrobust		

	coef	std err	z	P>|z|	[0.025	0.975]
const	117.6701	6.71e+04	0.002	0.999	-1.31e+05	1.32e+05
INSURANCE_Government	-29.2458	3.84e+04	-0.001	0.999	-7.53e+04	7.52e+04
INSURANCE_Private	-29.4039	3.84e+04	-0.001	0.999	-7.53e+04	7.52e+04
INSURANCE_Self Pay	-30.1317	3.84e+04	-0.001	0.999	-7.53e+04	7.52e+04
ETHNICITY_ASIAN	-30.0488	3.13e+04	-0.001	0.999	-6.13e+04	6.13e+04
ETHNICITY_BLACK/AFRICAN AMERICAN	-29.6896	3.13e+04	-0.001	0.999	-6.13e+04	6.13e+04
ETHNICITY_HISPANIC/LATINO	-29.7361	3.13e+04	-0.001	0.999	-6.13e+04	6.13e+04
ETHNICITY_Unknown/other	-29.6023	3.13e+04	-0.001	0.999	-6.13e+04	6.13e+04
ETHNICITY_WHITE	-29.5043	3.13e+04	-0.001	0.999	-6.13e+04	6.13e+04
ADMISSION_LOCATION_ INFO NOT AVAILABLE ****	-31.2396	3.01e+04	-0.001	0.999	-5.9e+04	5.89e+04
ADMISSION_LOCATION_EMERGENCY ROOM ADMIT	-29.5970	3.01e+04	-0.001	0.999	-5.9e+04	5.89e+04
ADMISSION_LOCATION_refferal from other facilities	-29.8224	3.01e+04	-0.001	0.999	-5.9e+04	5.89e+04
ADMISSION_LOCATION_transfer from other facilities	-29.4197	3.01e+04	-0.001	0.999	-5.9e+04	5.89e+04
MARITAL_STATUS_SINGLE	-0.3207	0.025	-13.009	0.000	-0.369	-0.272
GENDER_F	-29.5266	3.38e+04	-0.001	0.999	-6.63e+04	6.63e+04
GENDER_M	-29.4972	3.38e+04	-0.001	0.999	-6.63e+04	6.63e+04

```
# Check for the VIF values of the feature variables.
from statsmodels.stats.outliers_influence import variance_inflation_factor
```

```
vif = pd.DataFrame()
vif['Features'] = X_resampled_smt[rfe_cols].columns
vif['VIF'] = [variance_inflation_factor(X_resampled_smt[rfe_cols].values, i) for i in range(X_resampled_smt[rfe_cols].shape[
vif['VIF'] = round(vif['VIF'], 2)
vif = vif.sort_values(by = "VIF", ascending = False)
vif
```

	Features	VIF
8	ADMISSION_LOCATION_** INFO NOT AVAILABLE **	1.04
2	INSURANCE_Self Pay	0.47
3	ETHNICITY_ASIAN	0.22
5	ETHNICITY_HISPANIC/LATINO	0.19
6	ETHNICITY_Unknown/other	0.10
1	INSURANCE_Private	0.08
4	ETHNICITY_BLACK/AFRICAN AMERICAN	0.08
0	INSURANCE_Government	0.07
14	GENDER_M	0.07
13	GENDER_F	0.04
7	ETHNICITY_WHITE	0.03
12	MARITAL_STATUS_SINGLE	0.01
9	ADMISSION_LOCATION_EMERGENCY ROOM ADMIT	0.00
10	ADMISSION_LOCATION_refferal from other facilities	0.00
11	ADMISSION_LOCATION_transfer from other facilities	0.00

227]:
```
y_train_pred= log_1.predict(X_train_sm_1)
y_train_pred.head()
```

227]:
```
0    0.249350
1    0.378761
2    0.349263
3    0.394379
4    0.449278
dtype: float64
```

228]:
```
y_train_pred_final= pd.DataFrame({'class_label':y_resampled_smt.values, 'prob':y_train_pred.values})

y_train_pred_final.head(10)
```

228]:

class_label	prob

```
y_train_pred_final= pd.DataFrame({'class_label':y_resampled_smt.values, 'prob':y_train_pred.values})

y_train_pred_final.head(10)
```

	class_label	prob
0	0	0.249350
1	0	0.378761
2	0	0.349263
3	0	0.394379
4	1	0.449278
5	0	0.357313
6	0	0.364105
7	1	0.401435
8	0	0.401435
9	0	0.357313

```
## Finding Optimal Probablity Cutoff Point
```

```
# Creating columns for different probablity cutoffs
prob_cutoff = [float(p/10) for p in range(10)]
```


```
## Finding Optimal Probablity Cutoff Point
```


```
# Creating columns for different probablity cutoffs
prob_cutoff = [float(p/10) for p in range(10)]

for i in prob_cutoff:
    y_train_pred_final[i] = y_train_pred_final['prob'].map(lambda x : 1 if x > i else 0)

y_train_pred_final.head()
```

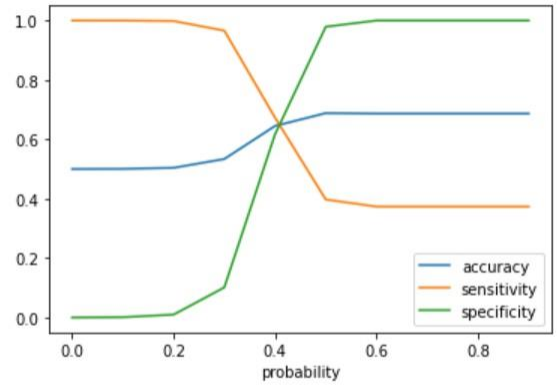
	class_label	prob	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
0	0	0.249350	1	1	1	0	0	0	0	0	0	0
1	0	0.378761	1	1	1	1	0	0	0	0	0	0
2	0	0.349263	1	1	1	1	0	0	0	0	0	0
3	0	0.394379	1	1	1	1	0	0	0	0	0	0
4	1	0.449278	1	1	1	1	1	0	0	0	0	0

```
# Creating a dataframe
from sklearn import metrics
from sklearn.metrics import confusion_matrix
cutoff_df = pd.DataFrame(columns=['probability', 'accuracy', 'sensitivity', 'specificity'])

```

```
# Creating a dataframe
from sklearn import metrics
from sklearn.metrics import confusion_matrix
cutoff_df = pd.DataFrame(columns=['probability', 'accuracy', 'sensitivity', 'specificity'])

for i in prob_cutoff:
    cm1 = metrics.confusion_matrix(y_train_pred_final['class_label'], y_train_pred_final[i] )
    total1=sum(sum(cm1))
    accuracy = (cm1[0,0]+cm1[1,1])/total1

    speci = cm1[0,0]/(cm1[0,0]+cm1[0,1])
    sensi = cm1[1,1]/(cm1[1,0]+cm1[1,1])
    cutoff_df.loc[i] =[ i ,accuracy,sensi,speci]
print(cutoff_df)
```

	probability	accuracy	sensitivity	specificity
0.0	0.0	0.500000	1.000000	0.000000
0.1	0.1	0.500512	0.999894	0.001130
0.2	0.2	0.504009	0.998304	0.009714
0.3	0.3	0.533664	0.966089	0.101240
0.4	0.4	0.645025	0.673143	0.616906
0.5	0.5	0.688191	0.397294	0.979088
0.6	0.6	0.686707	0.373415	1.000000
0.7	0.7	0.686707	0.373415	1.000000
0.8	0.8	0.686707	0.373415	1.000000
0.9	0.9	0.686707	0.373415	1.000000

```
# Plotting accuracy, sensitivity and specificity for different probabilities.
cutoff_df.plot('probability', ['accuracy','sensitivity','specificity'])
plt.show()
```




```
# Creating a column with name "predicted", which is the predicted value for 0.6 cutoff
y_train_pred_final['predicted'] = y_train_pred_final['prob'].map(lambda x: 1 if x > 0.4 else 0)
y_train_pred_final.head()
```

class_label	prob	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	predicted

```
# Creating a column with name "predicted", which is the predicted value for 0.6 cutoff
y_train_pred_final['predicted'] = y_train_pred_final['prob'].map(lambda x: 1 if x > 0.4 else 0)
y_train_pred_final.head()
```

	class_label	prob	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	predicted
0	0	0.249350	1	1	1	0	0	0	0	0	0	0	0
1	0	0.378761	1	1	1	1	0	0	0	0	0	0	0
2	0	0.349263	1	1	1	1	0	0	0	0	0	0	0
3	0	0.394379	1	1	1	1	0	0	0	0	0	0	0
4	1	0.449278	1	1	1	1	1	0	0	0	0	0	1

```
# Confusion metrics
confusion = metrics.confusion_matrix(y_train_pred_final['class_label'], y_train_pred_final['predicted'])
print(confusion)
```

```
[[17464 10845]
 [ 9253 19056]]
```

```
TP = confusion[1,1] # true positive
TN = confusion[0,0] # true negatives
FP = confusion[0,1] # false positives
FN = confusion[1,0] # false negatives
```

```
# Confusion metrics
confusion = metrics.confusion_matrix(y_train_pred_final['class_label'], y_train_pred_final['predicted'])
print(confusion)
```

```
[[17464 10845]
 [ 9253 19056]]
```

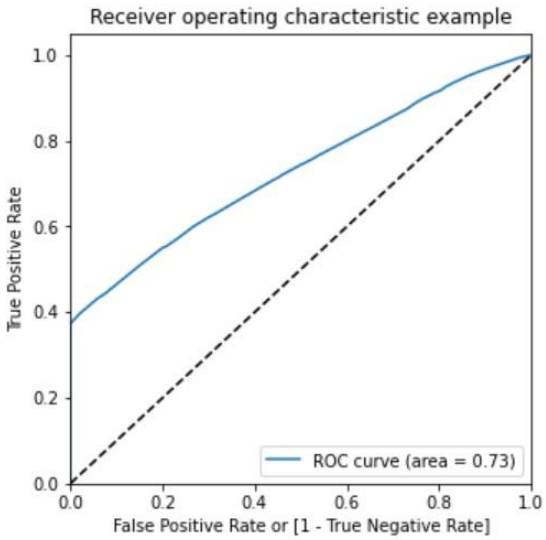
```
TP = confusion[1,1] # true positive
TN = confusion[0,0] # true negatives
FP = confusion[0,1] # false positives
FN = confusion[1,0] # false negatives
```

```
# Accuracy
print("Accuracy:-",metrics.accuracy_score(y_train_pred_final['class_label'], y_train_pred_final['predicted']))

# Sensitivity
print("Sensitivity:-",TP / float(TP+FN))

# Specificity
print("Specificity:-", TN / float(TN+FP))
```

```
Accuracy:- 0.6450245504963086
Sensitivity:- 0.6731428167720513
Specificity:- 0.6169062842205659
```

```
    return None
```

```
draw_roc(y_train_pred_final['class_label'], y_train_pred_final['prob'])
```




```
X_test_log=X_test[rfe_cols]
```

```
X_test_sm = sm.add_constant(X_test_log)
```

```
y_test_pred = log_1.predict(X_test_sm)
```

```
y_test_pred.head()
```

```
37611    0.364105
41184    0.417696
45658    0.366555
4908     0.378761
2995     0.456576
dtype: float64
```

```
y_pred_1 = pd.DataFrame(y_test_pred)
y_pred_1.head()
```

	0
37611	0.364105
41184	0.417696
45658	0.366555
4908	0.378761

```
y_test_pred_final.test_predicted=y_test_pred_final.test_predicted.astype("int64")
y_test_pred_final.class_los_label=y_test_pred_final.class_los_label.astype("int64")
```

```
confusion = metrics.confusion_matrix(y_test_pred_final['class_los_label'], y_test_pred_final['test_predicted'])
print(confusion)
```

```
[[4339 2702]
 [1728 1439]]
```

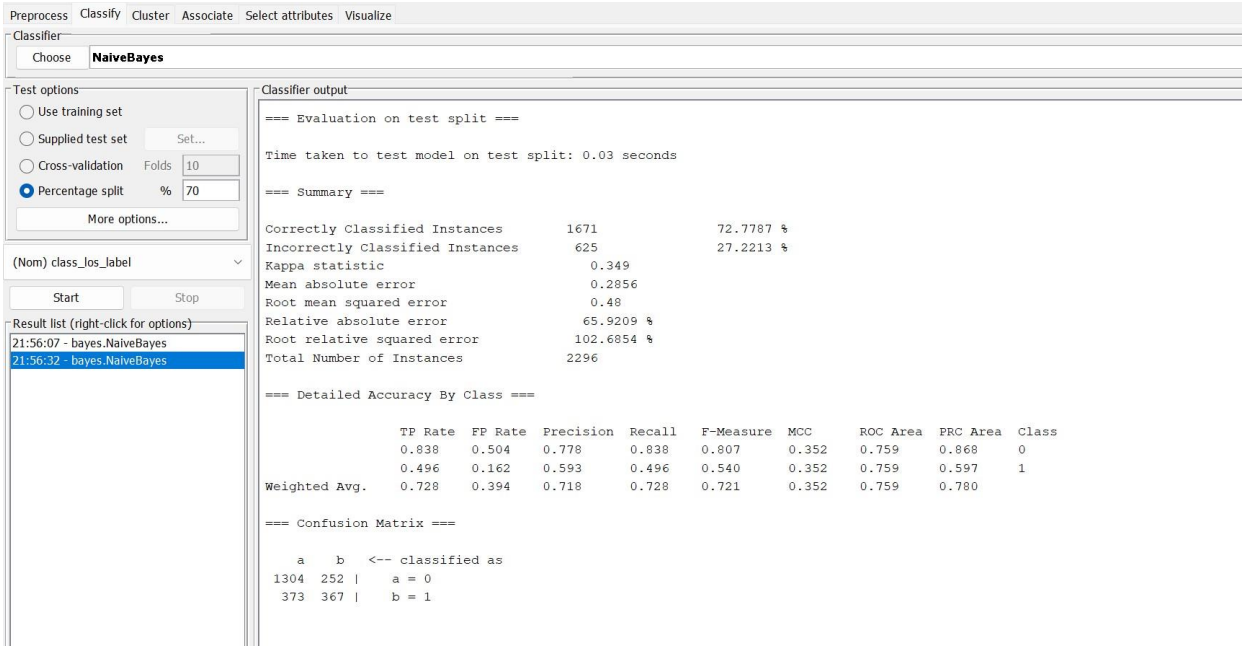
```
TP = confusion[1,1] # true positive
TN = confusion[0,0] # true negatives
FP = confusion[0,1] # false positives
FN = confusion[1,0] # false negatives
```

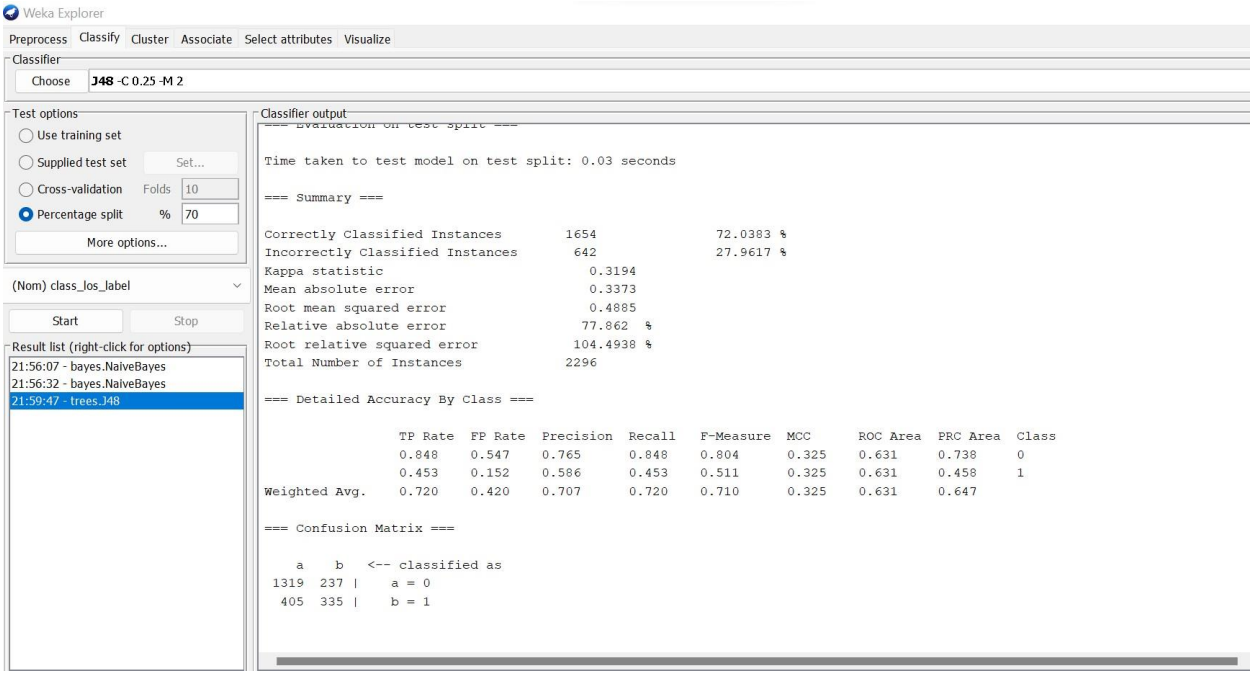
```
# Accuracy
print("Accuracy:-",metrics.accuracy_score(y_test_pred_final['class_los_label'], y_test_pred_final['test_predicted']))

# Sensitivity
print("Sensitivity:-",TP / float(TP+FN))

# Specificity
print("Specificity:-", TN / float(TN+FP))

#F1 Score
from sklearn.metrics import f1_score
print("F1_Score",f1_score(y_test_pred_final['class_los_label'], y_test_pred_final['test_predicted']))
```

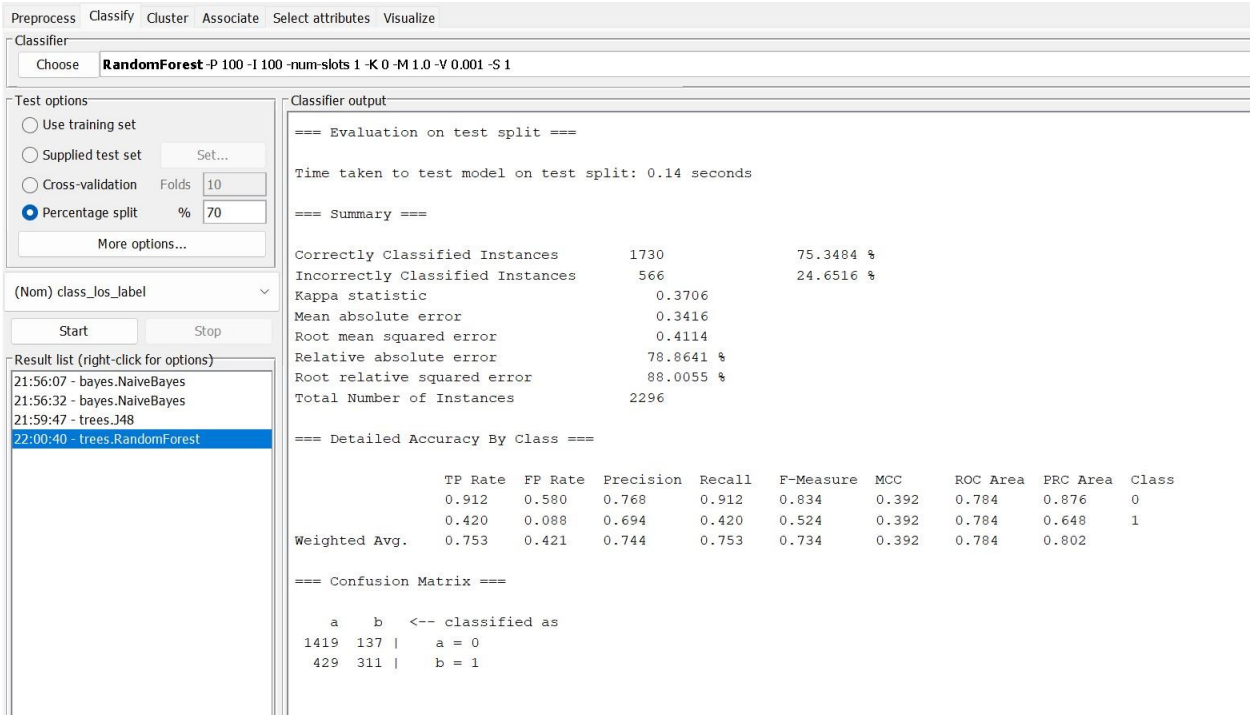
:

```
# Accuracy
print("Accuracy:-",metrics.accuracy_score(y_test_pred_final['class_los_label'], y_test_pred_final['test_predicted']))

# Sensitivity
print("Sensitivity:-",TP / float(TP+FN))

# Specificity
print("Specificity:-", TN / float(TN+FP))

#F1 Score
from sklearn.metrics import f1_score
print("F1_Score",f1_score(y_test_pred_final['class_los_label'], y_test_pred_final['test_predicted']))

```

```
Accuracy:- 0.5660266457680251
Sensitivity:- 0.45437322387117146
Specificity:- 0.6162476920891918
F1_Score 0.3938149972632732
```


Preprocess
Classify
Cluster
Associate
Select attributes
Visualize
Classifier
Choose
NaiveBayes
Test options
Use training set
Supplied test set
Set...
Cross-validation
Folds
10
Percentage split
%
70
More options...
(Nom) class_los_label
Start
Stop
Result list (right-click for options)
21:56:07 - bayes.NaiveBayes
21:56:32 - bayes.NaiveBayes
Classifier output
=== Evaluation on test split ===
Time taken to test model on test split: 0.03 seconds
=== Summary ===
Correctly Classified Instances 1671 72.7787 %
Incorrectly Classified Instances 625 27.2213 %
Kappa statistic 0.349
Mean absolute error 0.2856
Root mean squared error 0.48
Relative absolute error 65.9209 %
Root relative squared error 102.6854 %
Total Number of Instances 2296
=== Detailed Accuracy By Class ===
TP Rate FP Rate Precision Recall F-Measure MCC ROC Area PRC Area Class
0.838 0.504 0.778 0.838 0.807 0.352 0.759 0.868 0
0.496 0.162 0.593 0.496 0.540 0.352 0.759 0.597 1
Weighted Avg. 0.728 0.394 0.718 0.728 0.721 0.352 0.759 0.780
=== Confusion Matrix ===
a b <-- classified as
1304 252 | a = 0
373 367 | b = 1


Weka Explorer
Preprocess
Classify
Cluster
Associate
Select attributes
Visualize
Classifier
Choose
J48 -C 0.25 -M 2
Test options
Use training set
Supplied test set
Set...
Cross-validation
Folds
10
Percentage split
%
70
More options...
(Nom) class_los_label
Start
Stop
Result list (right-click for options)
21:56:07 - bayes.NaiveBayes
21:56:32 - bayes.NaiveBayes
21:59:47 - trees.J48
Classifier output
Time taken to test model on test split: 0.03 seconds
=== Summary ===
Correctly Classified Instances 1654 72.0383 %
Incorrectly Classified Instances 642 27.9617 %
Kappa statistic 0.3194
Mean absolute error 0.3373
Root mean squared error 0.4885
Relative absolute error 77.862 %
Root relative squared error 104.4938 %
Total Number of Instances 2296
=== Detailed Accuracy By Class ===
TP Rate FP Rate Precision Recall F-Measure MCC ROC Area PRC Area Class
0.848 0.547 0.765 0.848 0.804 0.325 0.631 0.738 0
0.453 0.152 0.586 0.453 0.511 0.325 0.631 0.458 1
Weighted Avg. 0.720 0.420 0.707 0.720 0.710 0.325 0.631 0.647
=== Confusion Matrix ===
a b <-- classified as
1319 237 | a = 0
405 335 | b = 1


Preprocess
Classify
Cluster
Associate
Select attributes
Visualize
Classifier
Choose
RandomForest -P 100 -I 100 -num-slots 1 -K 0 -M 1.0 -V 0.001 -S 1
Test options
Use training set
Supplied test set
Set...
Cross-validation
Folds
10
Percentage split
%
70
More options...
(Nom) class_los_label
Start
Stop
Result list (right-click for options)
21:56:07 - bayes.NaiveBayes
21:56:32 - bayes.NaiveBayes
21:59:47 - trees.J48
22:00:40 - trees.RandomForest
Classifier output
=== Evaluation on test split ===
Time taken to test model on test split: 0.14 seconds
=== Summary ===
Correctly Classified Instances 1730 75.3484 %
Incorrectly Classified Instances 566 24.6516 %
Kappa statistic 0.3706
Mean absolute error 0.3416
Root mean squared error 0.4114
Relative absolute error 78.8641 %
Root relative squared error 88.0055 %
Total Number of Instances 2296
=== Detailed Accuracy By Class ===
TP Rate FP Rate Precision Recall F-Measure MCC ROC Area PRC Area Class
0.912 0.580 0.768 0.912 0.834 0.392 0.784 0.876 0
0.420 0.088 0.694 0.420 0.524 0.392 0.784 0.648 1
Weighted Avg. 0.753 0.421 0.744 0.753 0.734 0.392 0.784 0.802
=== Confusion Matrix ===
a b <-- classified as
1419 137 | a = 0
429 311 | b = 1