Analysis of Patient's stay in the hospital

Saranya Chintalapati, Yeswanth Chamarthy,
Hareesh Prathipati

Contents

Introduction

Reducing the length of stay (LOS) at hospitals decreases the cost of care for patients, thereby improving financial, operational, and clinical outcomes. Additionally, hospital-acquired conditions can be minimized, improving outcomes.

Analyzing industry data is a critical component of healthcare analytics, which helps predict trends, improve outreach, and even control disease spread. Healthcare management uses different metrics for measuring performance, but the length of stay of a patient is one of the most important. When LOS is decreased, hospitals can match the demand for elective and emergent admissions, intensive care unit (ICU) care, and interhospital transfers with capacity.

Project Goal

This project aims to predict how long patients will stay in a hospital, allowing hospitals to optimize their resources and function more efficiently.

Assumptions from the data set

Patient-Level:
- Type of Admission – Hospitals usually admit patients based on three levels: urgent, emergency, and trauma. Patients who are admitted to urgent care stay for a shorter period. Contrarily, trauma patients typically stay longer because they need to be carefully monitored before they can be discharged.
- The severity of Illness – According to severity, there are three levels: Minor, Moderate, and Extreme. A minor patient will stay for a shorter period than an extreme patient.
- Visitors with Patients – Patients with more visitors will likely stay in the hospital longer.
- Age – Patients who are infants or elderly generally recover more slowly, so they are usually hospitalized for a more extended period.
- Admission Deposit – It is likely that patients who deposit a high amount of money at the time of admission will usually have severe conditions and will need to stay longer.

Hospital-Level:
- Ward Type – As the ICU patients' conditions are more severe, they may stay longer than patients in the general ward.
- Department – Patients undergoing surgery will likely stay longer than other

Data Cleaning and Preparation

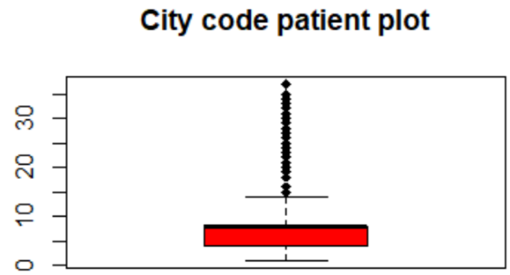
Data scrubbing is fixing duplicate or erroneous data in a data set. This helps in improving quality and helps in providing more consistent and accurate information for decision-making.

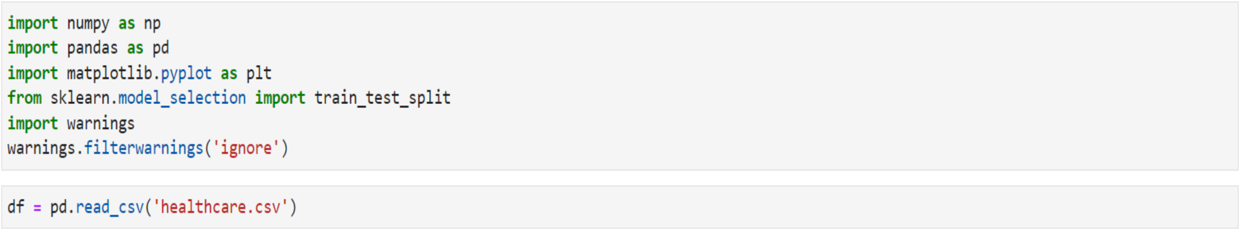
In our dataset, variables like “City_code_patient” and “Bed Grade” have null values. These missing values must be treated before feeding to the algorithm as they distort the model performance. To clean the data, we need to either replace the nulls with meaningful values or remove the records with nulls. So, the missing values are returned using the “mode” imputation technique.

Theory for mode imputation

We can note that data is right-skewed, so if we apply mean imputation, Outliers data points will significantly impact the mean. Hence it is not recommended to replace mean values in place of missing values. Since our data is a categorical variable, we can prefer to use mode imputation.

Since most of the variables in the dataset are categorical, we transformed them into numerical data.





```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
import warnings
warnings.filterwarnings('ignore')
```

```
df = pd.read_csv('healthcare.csv')
```

```
#replacing NA values in "bed grade" column
df['Bed Grade'].fillna(df['Bed Grade'].mode()[0], inplace = True)
```

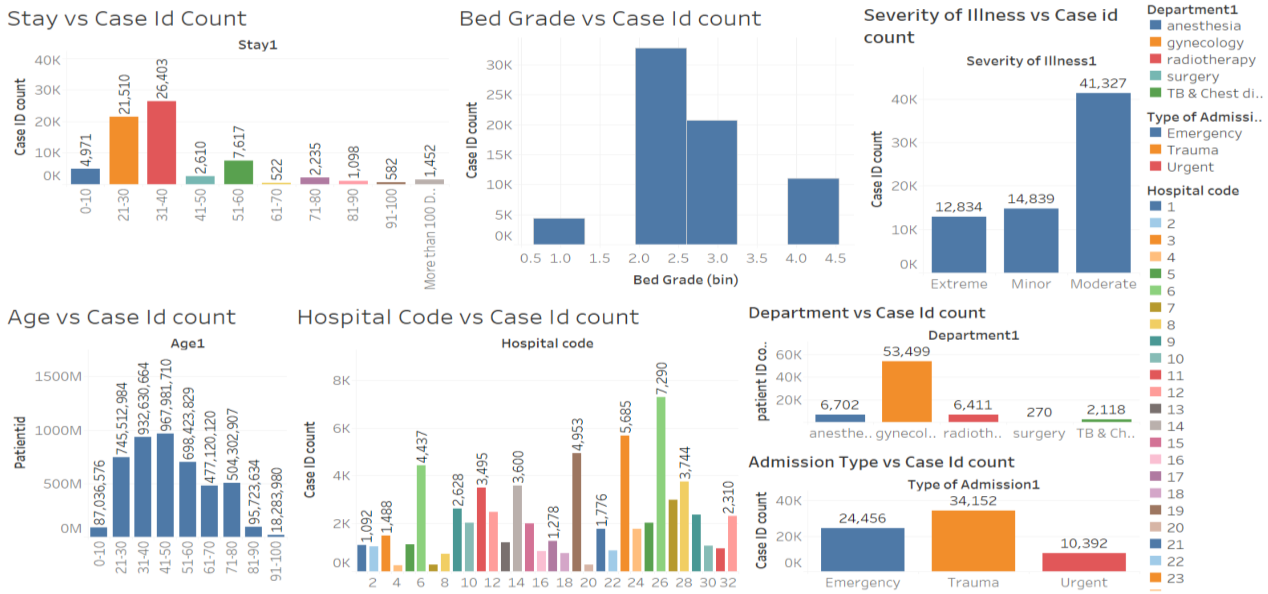
```
#replacing NA values in "City_Code_Patient" column
df['City_Code_Patient'].fillna(df['City_Code_Patient'].mode()[0], inplace = True)
```

```
#NA values in the dataset
df.isnull().sum().sort_values(ascending = False)
```

```
case_id                              0
Hospital_code                        0
Admission_Deposit                    0
Age                                  0
Visitors with Patient                0
Severity of Illness                  0
Type of Admission                    0
City_Code_Patient                    0
patientid                            0
Bed Grade                            0
Ward_Facility_Code                   0
Ward_Type                            0
Department                           0
Available Extra Rooms in Hospital    0
Hospital_region_code                 0
City_Code_Hospital                   0
Hospital_type_code                   0
Stay                                 0
dtype: int64
```

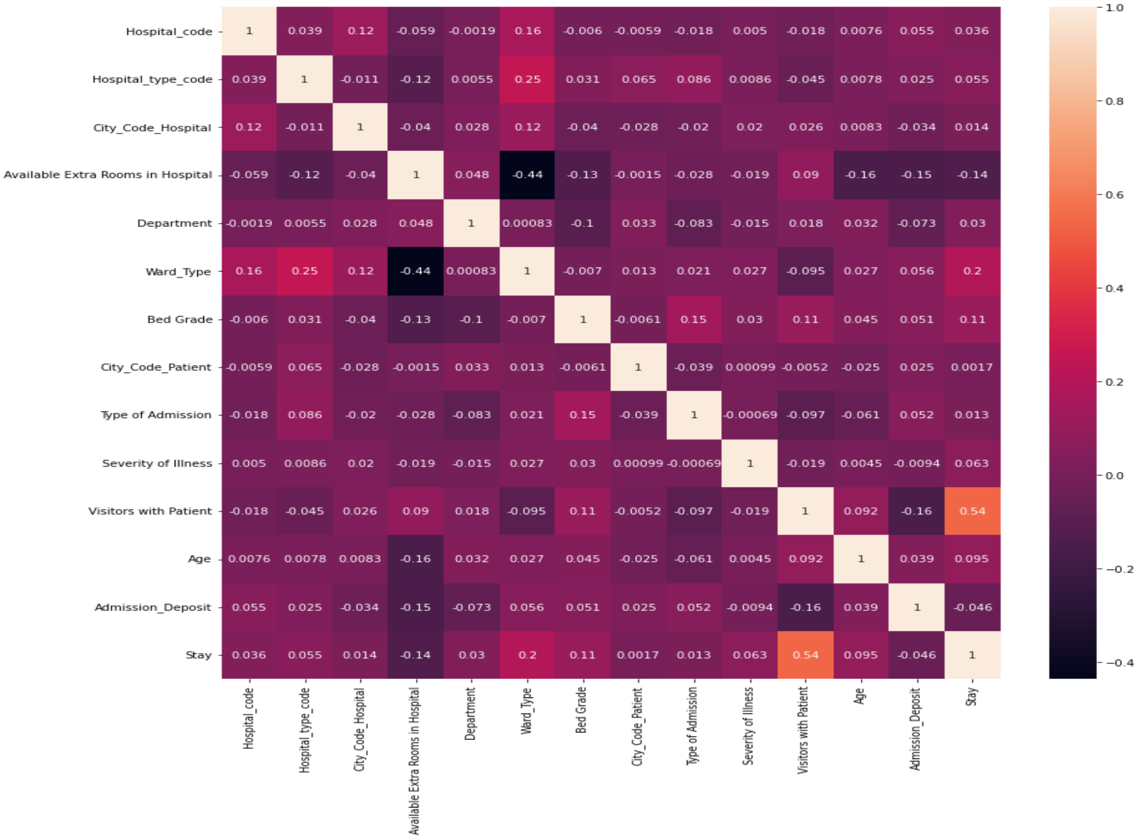
Data Exploratory Analysis

The below picture depicts the variation between different variables.



Correlation Matrix

The correlation matrix helps to check for any correlation between the variables so those correlated variables can be dropped when performing a regression analysis. However, in our case, we do not have any variables with a high correlation.



	Hospital_code	Hospital_type_code	City_Code_Hospital	Available Extra Rooms in Hospital	Department	Ward_Type	Bed Grade	City_Code_Patient	Type of Admission	Severity of Illness	Visitors with Patient	Age	Admission_Deposit	Stay
Hospital_code	1	0.039	0.12	-0.059	-0.0019	0.16	-0.006	-0.0059	-0.018	0.005	-0.018	0.0076	0.055	0.036
Hospital_type_code	0.039	1	-0.011	-0.12	0.0055	0.25	0.031	0.065	0.086	0.0086	-0.045	0.0078	0.025	0.055
City_Code_Hospital	0.12	-0.011	1	-0.04	0.028	0.12	-0.04	-0.028	-0.02	0.02	0.026	0.0083	-0.034	0.014
Available Extra Rooms in Hospital	-0.059	-0.12	-0.04	1	0.048	-0.44	-0.13	-0.0015	-0.028	-0.019	0.09	-0.16	-0.15	-0.14
Department	-0.0019	0.0055	0.028	0.048	1	0.00083	-0.1	0.033	-0.083	-0.015	0.018	0.032	-0.073	0.03
Ward_Type	0.16	0.25	0.12	-0.44	0.00083	1	-0.007	0.013	0.021	0.027	-0.095	0.027	0.056	0.2
Bed Grade	-0.006	0.031	-0.04	-0.13	-0.1	-0.007	1	-0.0061	0.15	0.03	0.11	0.045	0.051	0.11
City_Code_Patient	-0.0059	0.065	-0.028	-0.0015	0.033	0.013	-0.0061	1	-0.039	0.00099	-0.0052	-0.025	0.025	0.0017
Type of Admission	-0.018	0.086	-0.02	-0.028	-0.083	0.021	0.15	-0.039	1	-0.00069	-0.097	-0.061	0.052	0.013
Severity of Illness	0.005	0.0086	0.02	-0.019	-0.015	0.027	0.03	0.00099	-0.00069	1	-0.019	0.0045	-0.0094	0.063
Visitors with Patient	-0.018	-0.045	0.026	0.09	0.018	-0.095	0.11	-0.0052	-0.097	-0.019	1	0.092	-0.16	0.54
Age	0.0076	0.0078	0.0083	-0.16	0.032	0.027	0.045	-0.025	-0.061	0.0045	0.092	1	0.039	0.095
Admission_Deposit	0.055	0.025	-0.034	-0.15	-0.073	0.056	0.051	0.025	0.052	-0.0094	-0.16	0.039	1	-0.046
Stay	0.036	0.055	0.014	-0.14	0.03	0.2	0.11	0.0017	0.013	0.063	0.54	0.095	-0.046	1

Label Encoding

Label Encoding refers to converting the labels into a numeric form to convert them into a machine-readable format. Label encoding converts the data into machine-readable form, but it assigns a unique number (starting from 0) to each data class.

```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
df['Stay'] = le.fit_transform(df['Stay'].astype('str'))
```

```
#Label Encoding for all the Categorical columns
for i in ['Hospital_type_code', 'Department',
          'Ward_Type', 'Type of Admission', 'Severity of Illness', 'Age']:
    df[i] = LabelEncoder().fit_transform(df[i].astype('str'))
```

```
#Drop redundant columns "Hospital_region_code" and "Ward_facility_code" and duplicate columns
df = df.drop(['case_id', 'patientid', 'Hospital_region_code', 'Ward_Facility_Code'], axis =1)
```

```
df.shape
```

```
(69000, 14)
```

Data Partitioning

Splitting data into training and test partitions is essential to improve our predictions. Therefore, the data is divided into 70/30, where 70% of the data is used for training the model while the rest 30% is utilized for testing.

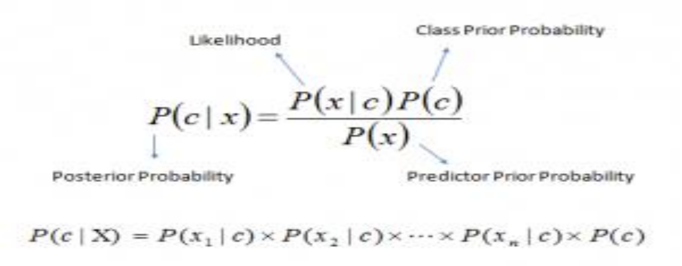
```
features = ['Hospital_code','Hospital_type_code','City_Code_Hospital','Available Extra Rooms in Hospital',
'Department','Ward_Type','Bed Grade','City_Code_Patient','Type of Admission','Severity of Illness',
'Visitors with Patient','Age','Admission_Deposit']
X = df.loc[:,features] #feature variable
y = df.loc[:,['Stay']] #target variable
X_train,X_test,y_train,y_test = train_test_split(X,y,random_state = 45,train_size = 0.70)
```

Models

Naïve Bayes Classification:

It is a classification technique based on Bayes' Theorem with an assumption of independence among predictors. In simple terms, a Naive Bayes classifier assumes that a particular feature in a class is unrelated to the presence of any other feature.

Bayes's theorem provides a way of calculating posterior probability P(c|x) from P(c), P(x), and P(x|c).

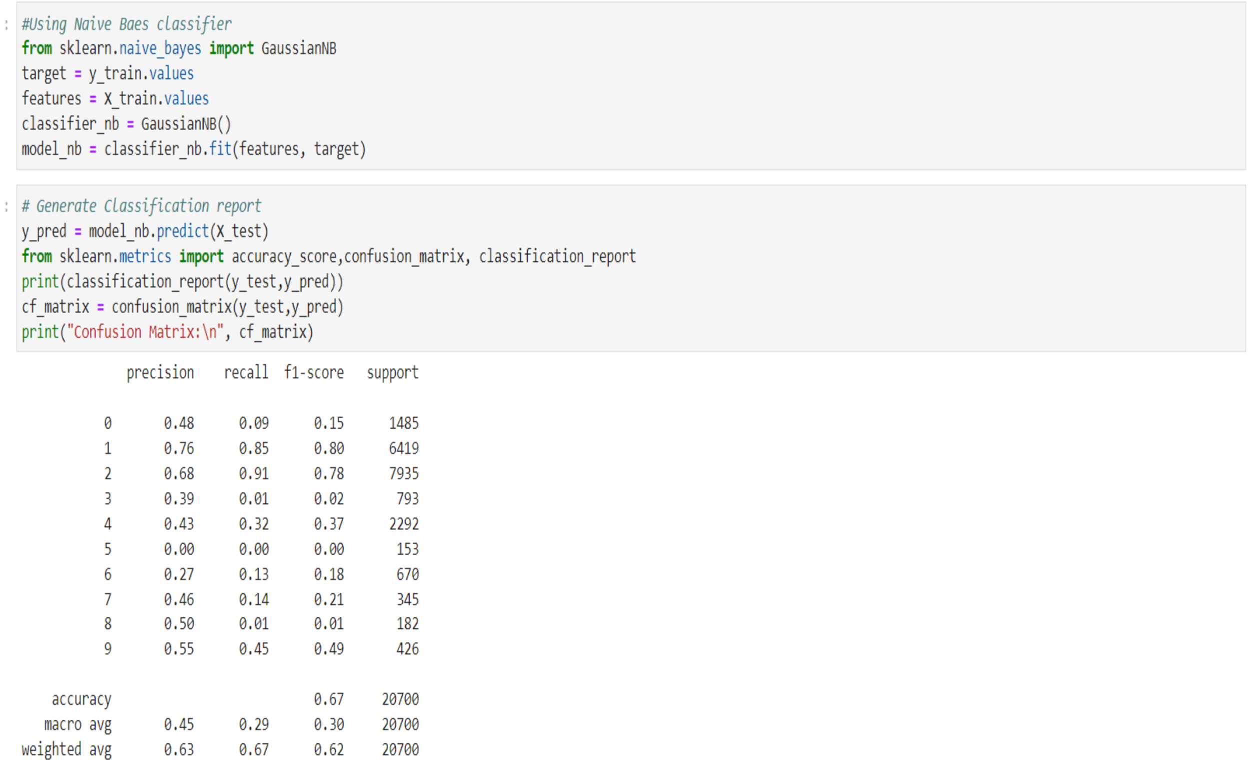


$$P(c \mid x) = \frac{P(x \mid c) P(c)}{P(x)}$$

$$P(c \mid X) = P(x_1 \mid c) \times P(x_2 \mid c) \times \cdots \times P(x_n \mid c) \times P(c)$$

Above,

- *P*(*c*|*x*) is the posterior probability of *class* (c, *target*) given *predictor* (x, *attributes*).
- *P*(*c*) is the prior probability of *class*.
- *P*(*x*|*c*) is the likelihood which is the probability of the *predictor* given *class*.
- *P*(*x*) is the prior probability of the *predictor*.



```
#Using Naive Baes classifier
from sklearn.naive_bayes import GaussianNB
target = y_train.values
features = X_train.values
classifier_nb = GaussianNB()
model_nb = classifier_nb.fit(features, target)
```

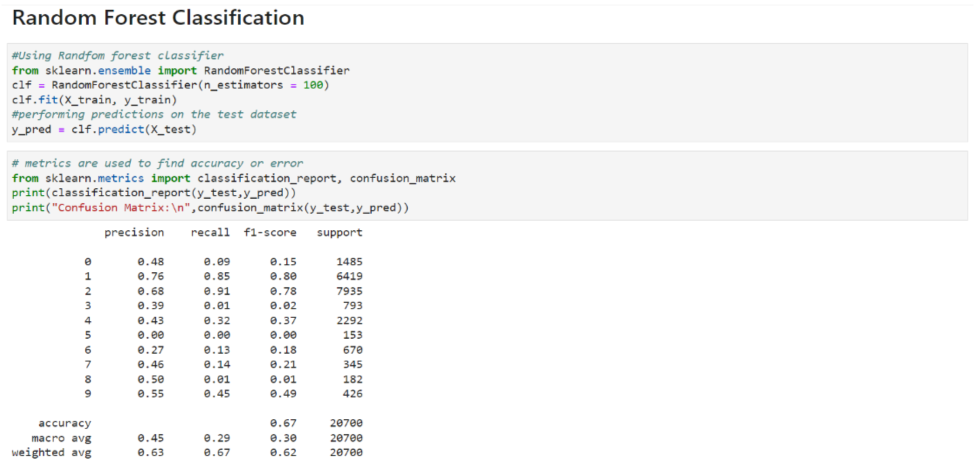
```
# Generate Classification report
y_pred = model_nb.predict(X_test)
from sklearn.metrics import accuracy_score,confusion_matrix, classification_report
print(classification_report(y_test,y_pred))
cf_matrix = confusion_matrix(y_test,y_pred)
print("Confusion Matrix:\n", cf_matrix)
```

```
              precision    recall  f1-score   support

           0       0.48      0.09      0.15      1485
           1       0.76      0.85      0.80      6419
           2       0.68      0.91      0.78      7935
           3       0.39      0.01      0.02       793
           4       0.43      0.32      0.37      2292
           5       0.00      0.00      0.00       153
           6       0.27      0.13      0.18       670
           7       0.46      0.14      0.21       345
           8       0.50      0.01      0.01       182
           9       0.55      0.45      0.49       426

    accuracy                           0.67     20700
   macro avg       0.45      0.29      0.30     20700
weighted avg       0.63      0.67      0.62     20700
```

Random Forest:

Random forest is a meta-estimator that uses averaging to improve predictive accuracy and prevent over-fitting by fitting several decision tree classifiers to diverse subsamples of the dataset.

The random forest consists of many individual decision trees, called stumps that operate as an ensemble. Each tree in the random forest generates a class prediction, and the class with the most votes become the model’s prediction.



Random Forest Classification

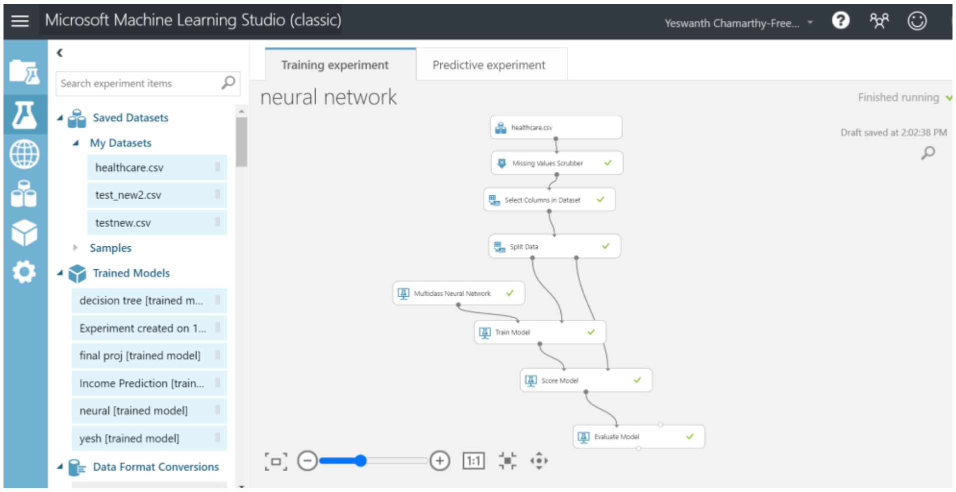
```
#Using Randfom forest classifier
from sklearn.ensemble import RandomForestClassifier
clf = RandomForestClassifier(n_estimators = 100)
clf.fit(X_train, y_train)
#performing predictions on the test dataset
y_pred = clf.predict(X_test)
```

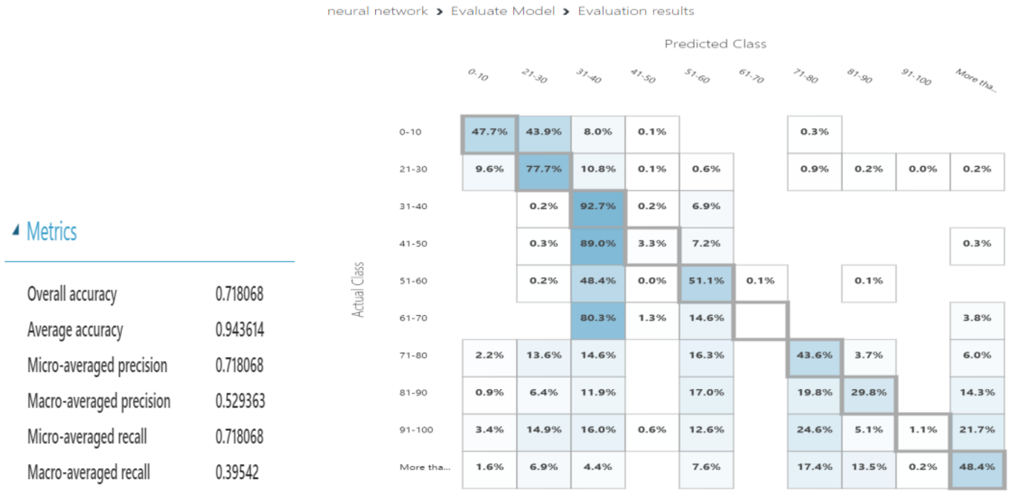
```
# metrics are used to find accuracy or error
from sklearn.metrics import classification_report, confusion_matrix
print(classification_report(y_test,y_pred))
print("Confusion Matrix:\n",confusion_matrix(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.48      0.09      0.15      1485
           1       0.76      0.85      0.80      6419
           2       0.68      0.91      0.78      7935
           3       0.39      0.01      0.02       793
           4       0.43      0.32      0.37      2292
           5       0.00      0.00      0.00       153
           6       0.27      0.13      0.18       670
           7       0.46      0.14      0.21       345
           8       0.50      0.01      0.01       182
           9       0.55      0.45      0.49       426

    accuracy                           0.67     20700
   macro avg       0.45      0.29      0.30     20700
weighted avg       0.63      0.67      0.62     20700
```

Neural Network:

Neural networks are based on a collection of connected units (neurons), which, like the brain's synapses, can transmit a signal to other neurons so that, acting like interconnected brain cells, they can learn and make decisions in a more human-like manner.





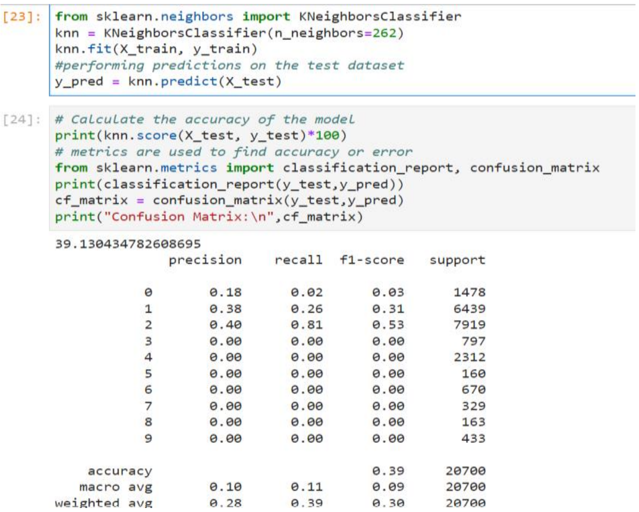
◢ Metrics

Overall accuracy	0.718068
Average accuracy	0.943614
Micro-averaged precision	0.718068
Macro-averaged precision	0.529363
Micro-averaged recall	0.718068
Macro-averaged recall	0.39542

	0-10	21-30	31-40	41-50	51-60	61-70	71-80	81-90	91-100	More tha...
0-10	47.7%	43.9%	8.0%	0.1%			0.3%			
21-30	9.6%	77.7%	10.8%	0.1%	0.6%		0.9%	0.2%	0.0%	0.2%
31-40		0.2%	92.7%	0.2%	6.9%					
41-50		0.3%	89.0%	3.3%	7.2%					0.3%
51-60		0.2%	48.4%	0.0%	51.1%	0.1%		0.1%		
61-70			80.3%	1.3%	14.6%					3.8%
71-80	2.2%	13.6%	14.6%		16.3%		43.6%	3.7%		6.0%
81-90	0.9%	6.4%	11.9%		17.0%		19.8%	29.8%		14.3%
91-100	3.4%	14.9%	16.0%	0.6%	12.6%		24.6%	5.1%	1.1%	21.7%
More tha...	1.6%	6.9%	4.4%		7.6%		17.4%	13.5%	0.2%	48.4%

KNN

K-nearest neighbors (KNN) is a supervised machine learning technique. The KNN classifier identifies a data point's class using the majority voting principle.



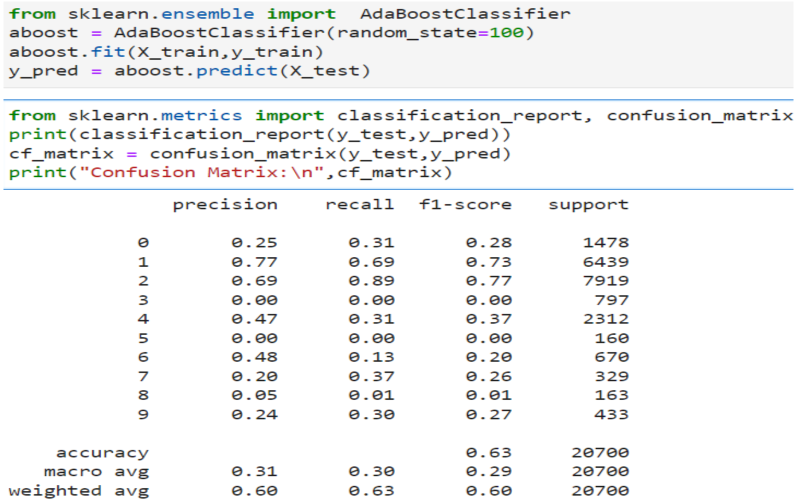
```
[23]: from sklearn.neighbors import KNeighborsClassifier
      knn = KNeighborsClassifier(n_neighbors=262)
      knn.fit(X_train, y_train)
      #performing predictions on the test dataset
      y_pred = knn.predict(X_test)
```

```
[24]: # Calculate the accuracy of the model
      print(knn.score(X_test, y_test)*100)
      # metrics are used to find accuracy or error
      from sklearn.metrics import classification_report, confusion_matrix
      print(classification_report(y_test,y_pred))
      cf_matrix = confusion_matrix(y_test,y_pred)
      print("Confusion Matrix:\n",cf_matrix)
```

```
39.130434782608695
              precision    recall  f1-score   support

           0       0.18      0.02      0.03      1478
           1       0.38      0.26      0.31      6439
           2       0.40      0.81      0.53      7919
           3       0.00      0.00      0.00       797
           4       0.00      0.00      0.00      2312
           5       0.00      0.00      0.00       160
           6       0.00      0.00      0.00       670
           7       0.00      0.00      0.00       329
           8       0.00      0.00      0.00       163
           9       0.00      0.00      0.00       433

    accuracy                           0.39     20700
   macro avg       0.10      0.11      0.09     20700
weighted avg       0.28      0.39      0.30     20700
```

Adaboost

Ada-boost is one ensemble boosting classifier. It combines multiple poorly performing classifiers to increase the accuracy of classifiers. This is done by increasing the weights of misclassified instances and forcing the model to choose a different prediction. This ensemble model can be implemented on any model, like a decision tree or logistic regression.



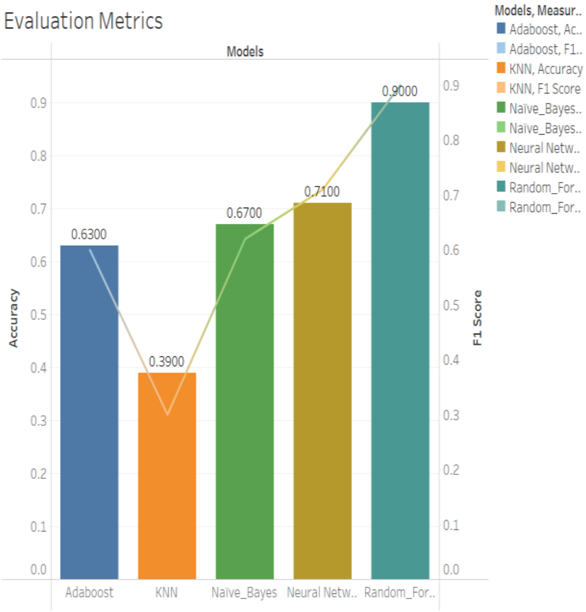
```
from sklearn.ensemble import  AdaBoostClassifier
aboost = AdaBoostClassifier(random_state=100)
aboost.fit(X_train,y_train)
y_pred = aboost.predict(X_test)
```

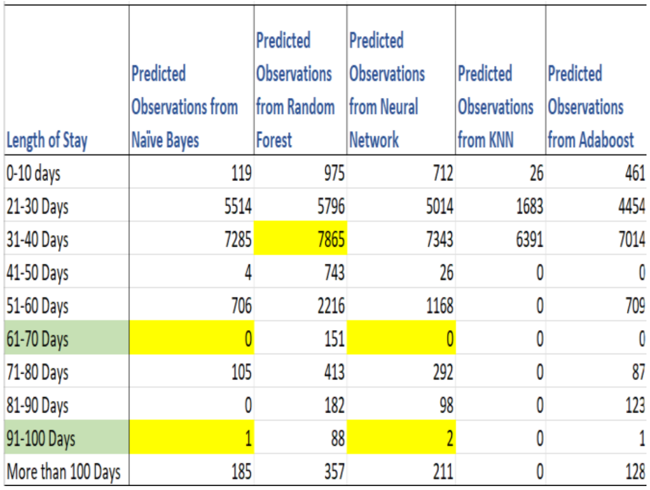
```
from sklearn.metrics import classification_report, confusion_matrix
print(classification_report(y_test,y_pred))
cf_matrix = confusion_matrix(y_test,y_pred)
print("Confusion Matrix:\n",cf_matrix)
```

```
              precision    recall  f1-score   support

           0       0.25      0.31      0.28      1478
           1       0.77      0.69      0.73      6439
           2       0.69      0.89      0.77      7919
           3       0.00      0.00      0.00       797
           4       0.47      0.31      0.37      2312
           5       0.00      0.00      0.00       160
           6       0.48      0.13      0.20       670
           7       0.20      0.37      0.26       329
           8       0.05      0.01      0.01       163
           9       0.24      0.30      0.27       433

    accuracy                           0.63     20700
   macro avg       0.31      0.30      0.29     20700
weighted avg       0.60      0.63      0.60     20700
```

Prediction and Results

From all the models we implemented, Random forest gives us the best fit for the data with a 0.9 accuracy. Although, Neural Network and Naïve Bayes tend to perform well but they fail to classify few levels like 61-70 days, and 91-100 Days. Unlike these models, Random forest was able to classify all levels with good accuracy and F1-score.





Length of Stay	Predicted Observations from Naïve Bayes	Predicted Observations from Random Forest	Predicted Observations from Neural Network	Predicted Observations from KNN	Predicted Observations from Adaboost
0-10 days	119	975	712	26	461
21-30 Days	5514	5796	5014	1683	4454
31-40 Days	7285	7865	7343	6391	7014
41-50 Days	4	743	26	0	0
51-60 Days	706	2216	1168	0	709
61-70 Days	0	151	0	0	0
71-80 Days	105	413	292	0	87
81-90 Days	0	182	98	0	123
91-100 Days	1	88	2	0	1
More than 100 Days	185	357	211	0	128

Future Insights

- From the data point of view, if we have the data required for estimating the cost of equipment and resources in the hospitals, we can analyze the estimated budget required for effective management of resources, like smart staffing, in the hospitals.
- Reducing patient queues by gathering patient information from their smartwatches data.
- Using smartwatch data, we can track sleeping habits, heart rates, step count, etc., to identify potential health risks and enhance patient engagement by improving the care for a given patient.

Conclusion

The hospital can better allocate resources and manage patients if they can predict the length of their stay at the time of admission. Identifying factors associated with LOS so that hospitals can manage resources and develop new treatment plans could help hospitals manage resources and develop new treatment plans. Utilizing hospital resources efficiently and limiting hospital stays can reduce national medical spending.

Reference Links

https://www.kaggle.com/datasets/nehaprabhavalkar/av-healthcare-analytics-ii

https://www.sisense.com/glossary/healthcare-analytics-basics/

https://www.techtarget.com/searchdatamanagement/definition/data-scrubbing

https://www.analyticsvidhya.com/blog/2017/09/naive-bayes-explained/

https://www.geeksforgeeks.org/hyperparameter-tuning/