Analyzing and Predicting Patient Length of Stay based off Admission Data in Hospitals

Completed by: Saptarshi Kundu, Paul George Parakkal, Mike Willis, Kunal Dubey, Yash Jignesh Joysher

Contents

Introduction

With the COVID-19 pandemic running rampart, healthcare has taken the forefront globally. Healthcare systems are comprised of many parts, including hospitals, and being able to accurately and efficiently allocate resources to all the systems would increase the quality of treatment available to all patients. When attending a hospital for treatment, a key metric is a patient's length of stay. The length of stay affects things such as disease transmissibility, how many beds are available, how many nurses, doctors, and staff are needed and when, and which wards are at what capacity. This report's focus is on: **Can patient length of stay in a healthcare system be reliably predicted using a machine learning model based on hospital admission data?**

Dataset

The dataset we chose is: AV: Healthcare Analytics II. It contained 17 features, the target, and 318,438 data samples. Our dataset contained 3 files: test_data.csv, train_data.csv, and train_data_dictionary.csv.

Methodology

Picking a Dataset

We spent many hours as a team looking at and discussing various datasets on Kaggle. We wanted one that would be an interesting and practical problem to solve in real life. Many datasets were interesting but beyond the scope of our report. Paul finally discovered the Healthcare Analytics II dataset and we agreed it would be an interesting challenge to tackle!

Exploratory Data Analysis

We began by exploring all the features and target within the dataset. There are 17 features total:

1. Hospital Code: It states that which hospital that we are going to select.
2. Hospital type code: This feature explains what type of hospital patient will be admitted i.e. government hospital or private hospital. Also, with this feature, we can the type of facilities that are available in the hospital for the patient.
3. City Code Hospital: The city code feature helps in determining the city where the hospital is located, helping us to understand what
4. Hospital Region code: The region code further classifies the data and helps in selecting the region of the hospital within the city.
5. Available Extra Rooms in Hospital: The extra rooms specifies the number of rooms that are available of the patients to accommodate the patients in any case of emergency without causing any problem to the existing patients.
6. Departments: The number of departments in the hospital helps to determine the what type of cases can hospital manage. The different departments such as intensive care unit, emergency department, operating room are example of department that classify how good the hospital is.
7. Ward type: The ward type determines the type of room and facilities the hospital will provide to the patient. Types of ward are general ward, private ward, special ward. General ward is where

number of patient of accommodated in single room with limited facilities whereas private ward is where patient have separate room with all facilities, The special ward have dedicated nurse and doctors for a single patient with all the facilities.

8. Ward Type: The ward type code.
9. Ward Facility Code: The ward facilities are the facilities that are provided in the ward such as food, television, dedicated doctors and nurses.
10. Bed grade: The bed grade tells the type of bed that has been assigned to the patient. This depends on the amount that is paid by the patient. Few examples of bed grades that differs are size, mattress type, electronic equipment in bed.
11. Patient ID: Unique patient ID when they checked in.
12. Case ID: Case ID registered in hospital.
13. City Code Of Patient: The city code tells the location of the patient.
14. Type of Admission: Admission Type registered by the Hospital
15. Severity of Illness: This explains critical the condition of the patient and the kind of treatment the patient requires in order sustain his/her life. This also helps in predicting the fatality rate of an individual.
16. Visitors with patient: This tells us the number of visitors that are with the patient at the time of admission and till the patient is discharged.
17. Age: The age of patient is key feature as it determines the fatality rate of the individual.
18. Admission deposit: It to the amount of money that has been deposited in the hospital when the patient of was admitted.

The x co-ordinate of the target has several features, as explained above, which are processed and used to predict the target.

The y co-ordinate of the target shows the target. In this report this is the number of days a patient stays in the hospital. As shown in Figure 1, there are 11 classes. The maximum number of stays was 21-30 days whereas the minimum was somewhere around 61-70 days.

The same has been shown below in the Output Bar graph.

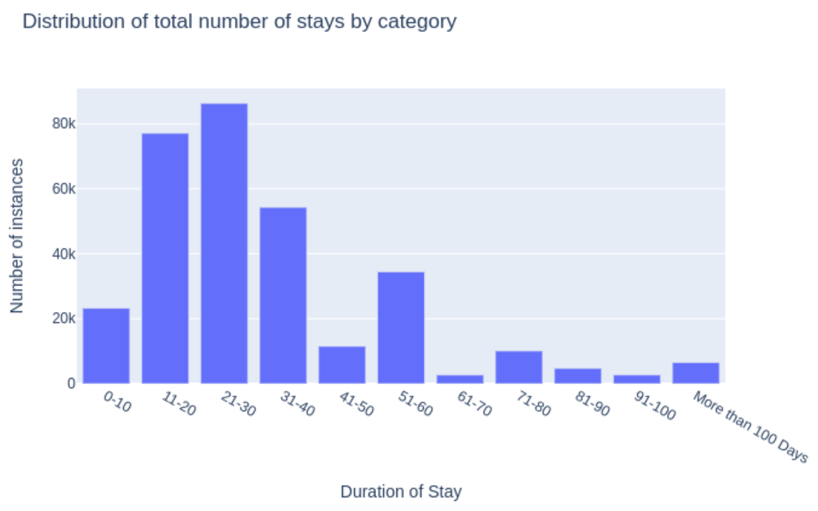
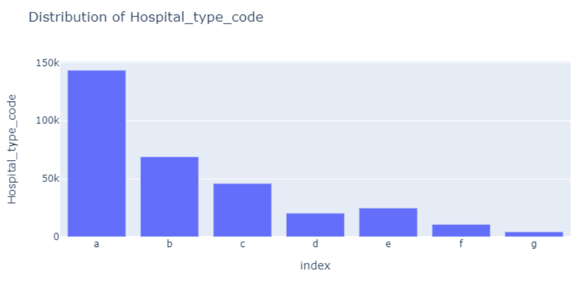


Figure 1. Distribution of total number of stays by category

The following Figures show the distribution of data by feature.



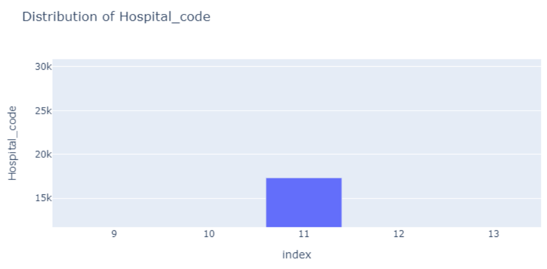
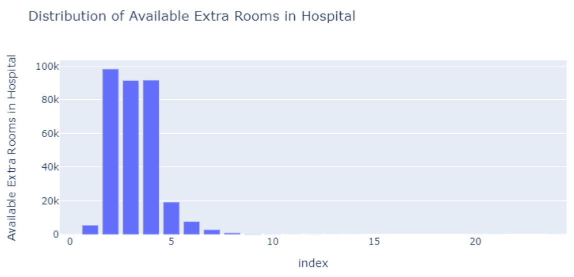


Figure 2. Hospital Type Code Distribution

Figure 3. Hospital Code Distribution



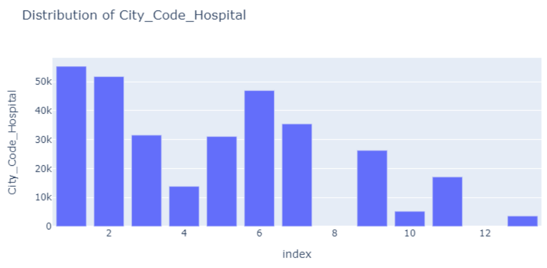
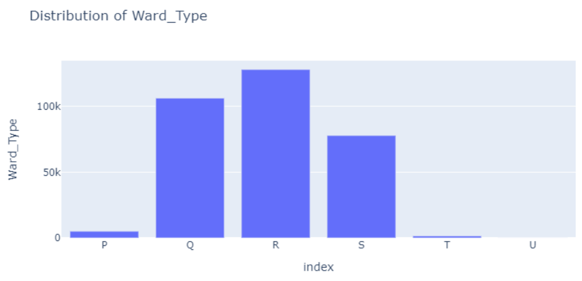


Figure 4. Available Extra Rooms in Hospital Distribution

Figure 5. City Code Hospital Distribution



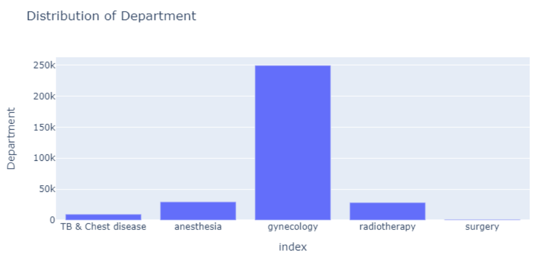
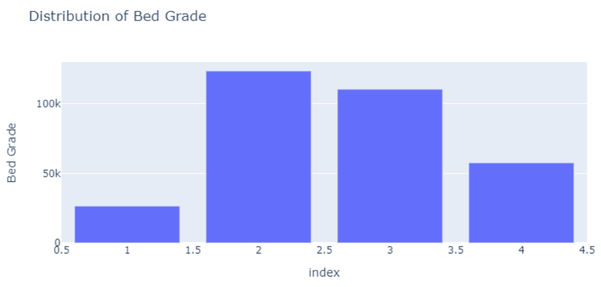


Figure 6. Ward Type Distribution

Figure 7. Department Distribution



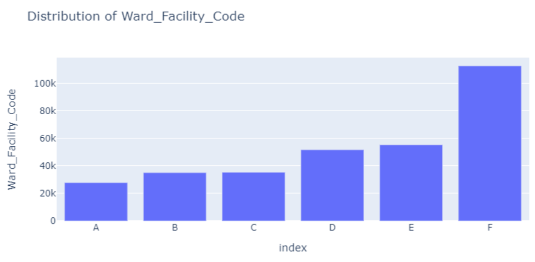


Figure 8. Bed Grade Distribution

Figure 9. Ward Facility Code Distribution

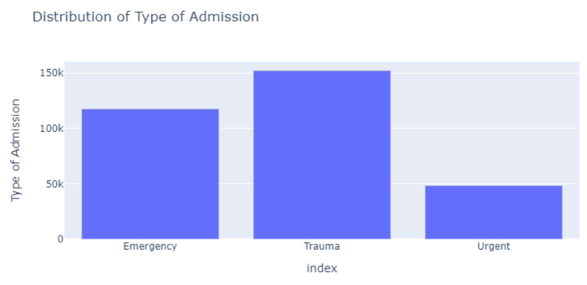


Figure 10. Type of Admission Distribution

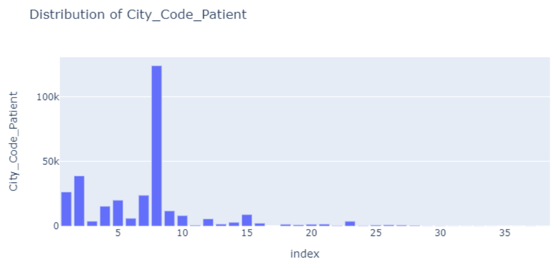


Figure 11. City Code Patient Distribution

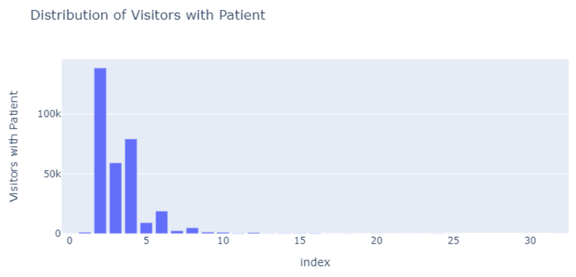


Figure 12. Visitors with Patient Distribution

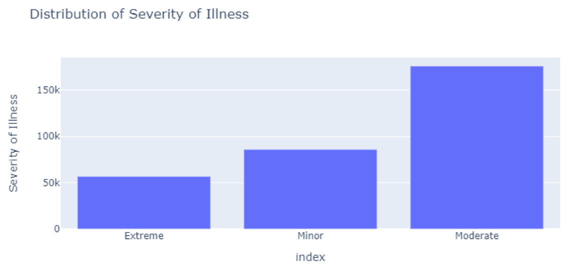


Figure 13. Severity of Illness Distribution

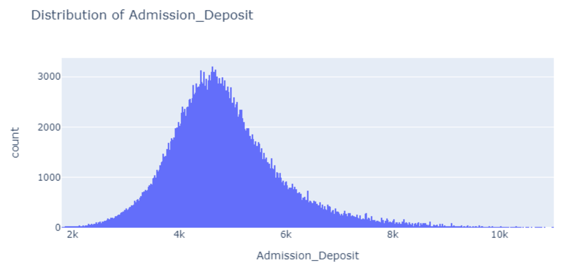


Figure 14. Admission Deposit Distribution

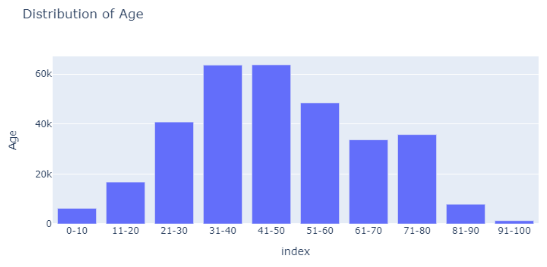


Figure 15. Age Distribution

Preprocessing

Data Preprocessing refers to the transformations applied to a dataset before using it on a machine learning model. In this process, we take the raw dataset, apply some transformation methods, and convert it to a clean dataset, something that our machine learning models can process.

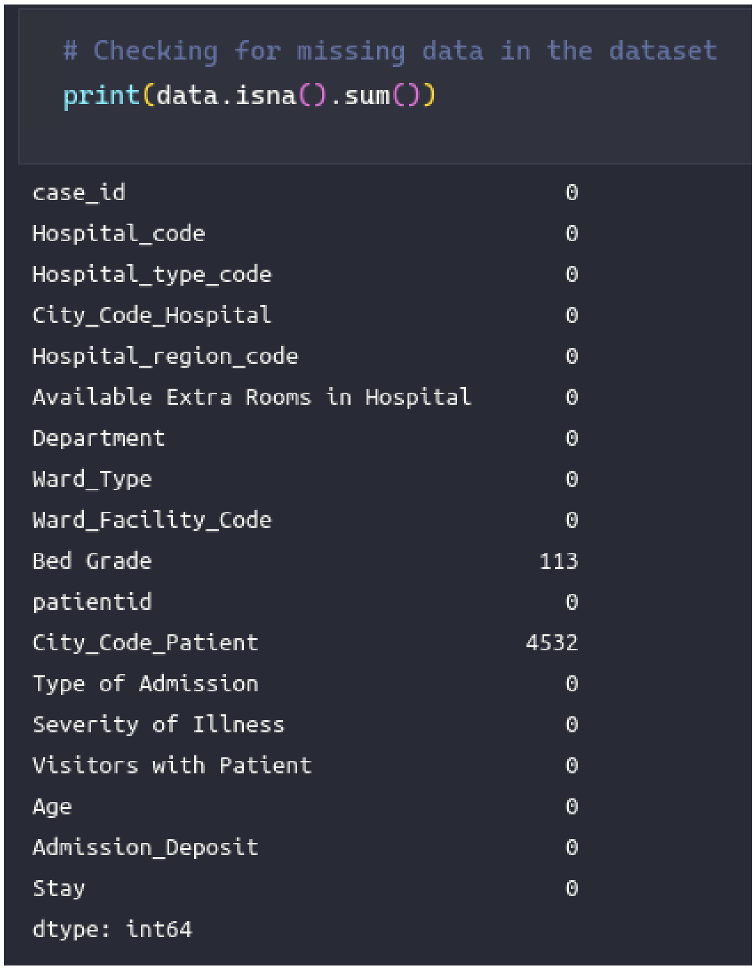
Following are the steps we used to transform our raw data into clean data, that we can eventually feed to our model:

1. Handling missing data
2. Encoding categorical & ordinal features to numerical features
3. Feature Scaling

4. Dimensionality Reduction

Handling of missing data

As we explored before, our total dataset consisted of more than 318,000 samples. The total number of missing data points was 4645, which is less than 1.5% of the total number of samples. Hence, we decided to drop all the samples with any missing values. After dropping the samples, our total number of samples was 313,793.



```
# Checking for missing data in the dataset
print(data.isna().sum())
```

```
case_id                                0
Hospital_code                          0
Hospital_type_code                     0
City_Code_Hospital                     0
Hospital_region_code                   0
Available Extra Rooms in Hospital      0
Department                             0
Ward_Type                              0
Ward_Facility_Code                     0
Bed Grade                            113
patientid                              0
City_Code_Patient                   4532
Type of Admission                      0
Severity of Illness                    0
Visitors with Patient                  0
Age                                    0
Admission_Deposit                      0
Stay                                   0
dtype: int64
```

Figure 16. Missing Data Table

Encoding categorical & ordinal features to numerical features

We had a total of 9 categorical and ordinal features, plus our target labels. Our decision to go with Label Encoding instead of One Hot Encoding was based on two factors: Overall performance of the model, and the models we chose. As we will discuss in detail in a later part of the project, we picked two models, Random Forest Classifier and Light Gradient Boosting Machine Classifier, both of which are decision tree-based models. Below is a representation of decision trees with and without One Hot Encoding:

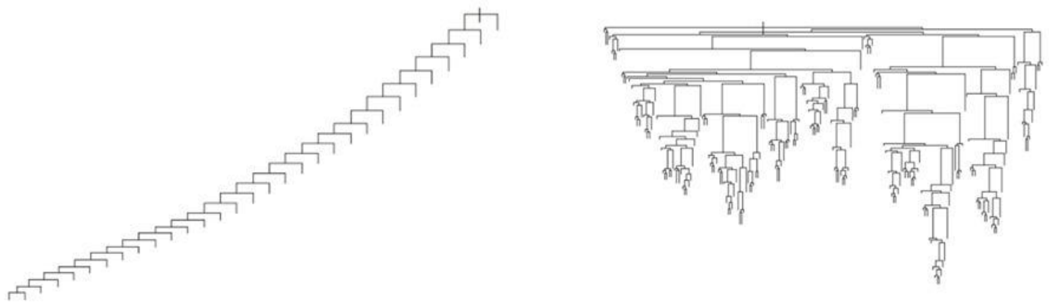


Figure 17. Decision Tree with One Hot Encoding Visualization vs Decision Tree without

It can be clearly observed from the graphical representations that in decision tree-based models, use of One Hot Encoding results in the decision trees becoming a lot sparser. A huge portion of the tree becomes binary, which in most cases hurts the performance of the model significantly. Hence, we decided to use Label Encoding to encode all our categorical and ordinal features, as well as the target classes. It should also be noted that Label Encoding does not have the same negative effects on a decision tree-based model, as it does on Regression based models.

Feature Scaling

The general consensus that tree-based models do not require feature scaling, due to their insensitivity to the variance in the data, convinced us not to go ahead with it initially. However, as a last resort to improve accuracy, we tried out feature scaling.

The Standard scaler removes the mean and scales the data to unit variance, while the MinMax scaler rescales so that all data points lie in the range [0,1]. Depending on the distribution of the features, we used both scalers. Features with distributions that resembled 'Gaussian' or a mixture of 'Gaussian' distributions were scaled using Standard scaler, and the MinMax scaler were applied on the rest of the features.

Dimensionality Reduction (PCA)

Dimensionality reduction is generally used to remove multicollinearity from the features, a situation where one feature can be closely expressed as a linear combination of a subset of features. We applied dimensionality reduction using PCA as one of our attempts to improve accuracy score. In spite of trying multiple values for number of components, like 3, 5 and 7, no significant improvement in accuracy was observed.

Feature Selection

In Machine Learning, the model is always as good as the data that it is trained with. Feature selection is one of the core concepts in machine learning which has a huge impact on the performance of the model. It should always be considered as one of the first and foremost important steps in designing the model.

Feature Selection is the process of manually or automatically selecting the features that have the largest impact on the prediction labels. Some of the key benefits of the feature selection process are:

- Less misleading data helps increase the model accuracy.
- Reducing redundant data provides less opportunity to the model to make decisions based on noise.
- Having a smaller number of data points reduces algorithm complexity, which reduces the chance of overfitting, and significantly reduces training times.

For our project, we tried a combination of both manual and automatic feature selection processes.

Manual Feature Selection

Looking at the dataset, there are two different attributes that are different forms of IDs. IDs in a dataset help in separating unique samples, but it does not have any importance in training or predicting the samples. Hence, we dropped both attributes manually from our features list. Additionally we dropped some features based on the correlation between the features. This is pictured below in Figure 16.

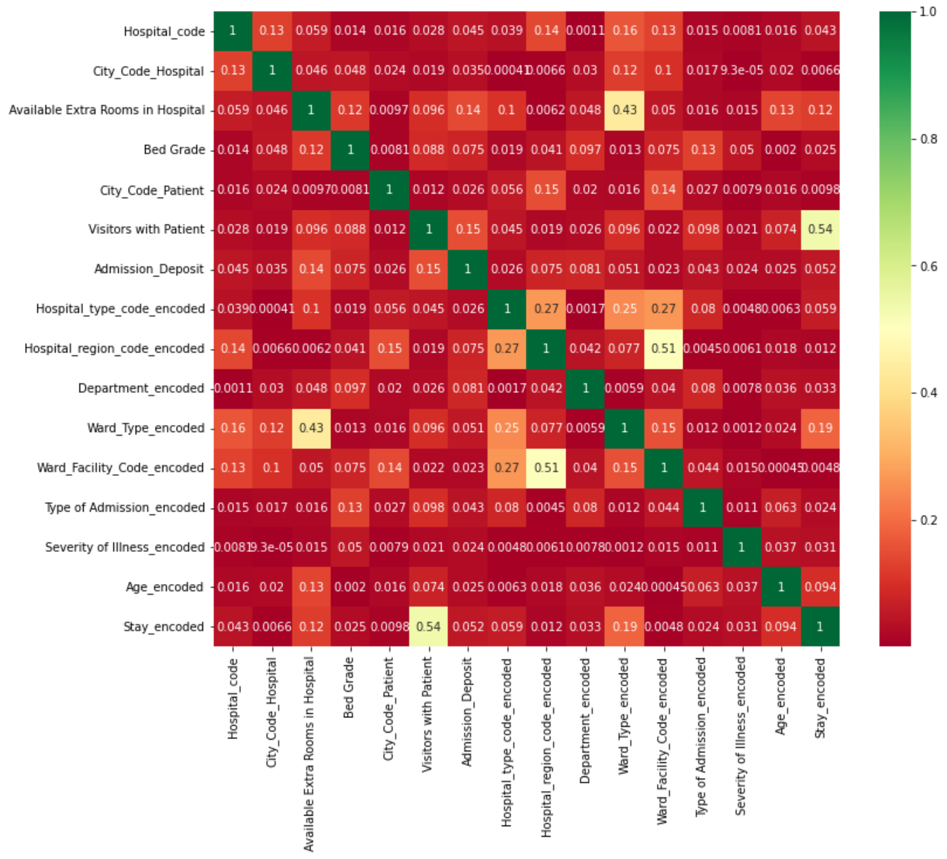


Figure 18. Correlation between Stay_encoded and all the features

Automated Feature Selection

For automated feature selection, we used two embedded feature selection methods: Random Forest Classifier and Light Gradient Boosting Machine (LGBM) Classifier. The following charts represent the feature importance ratios as represented by the two methods.

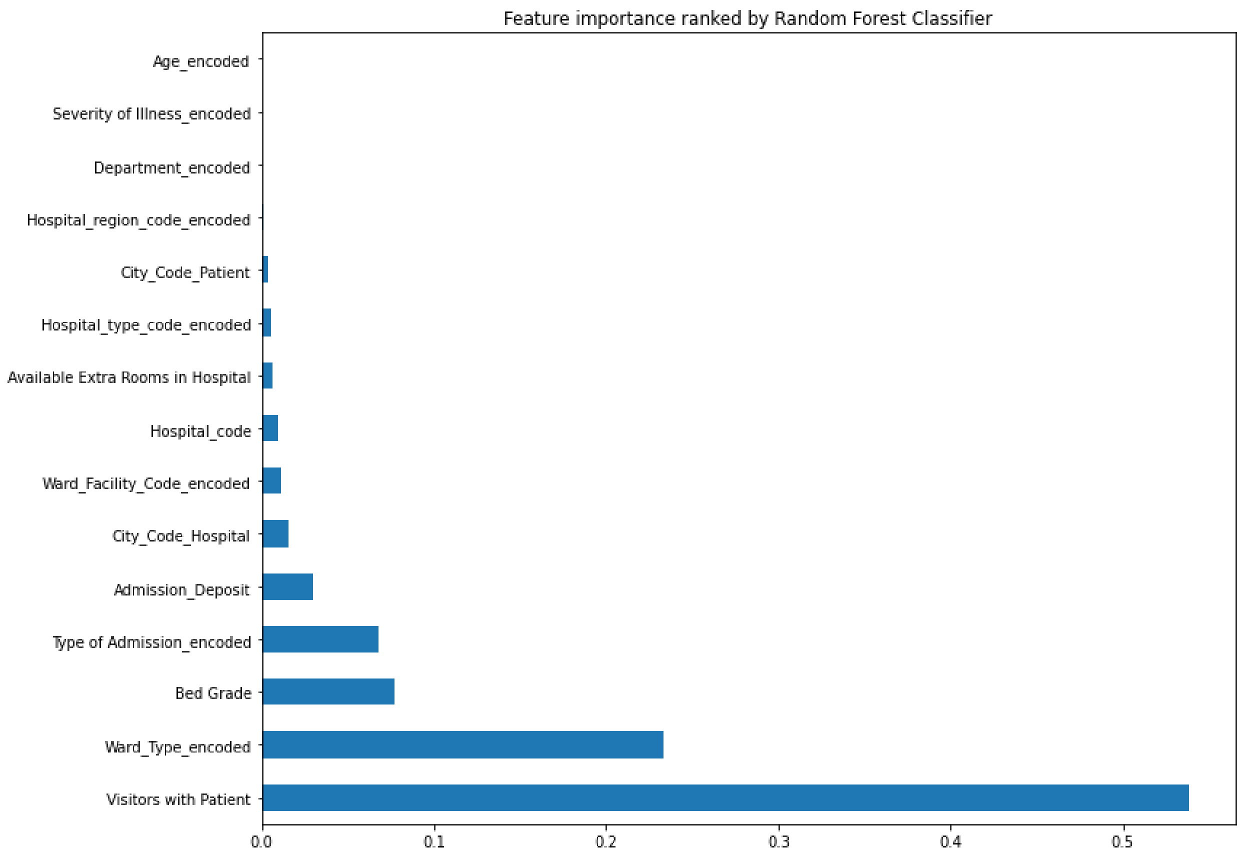


Figure 19. Feature Importance Ranked by Random Tree Classifier

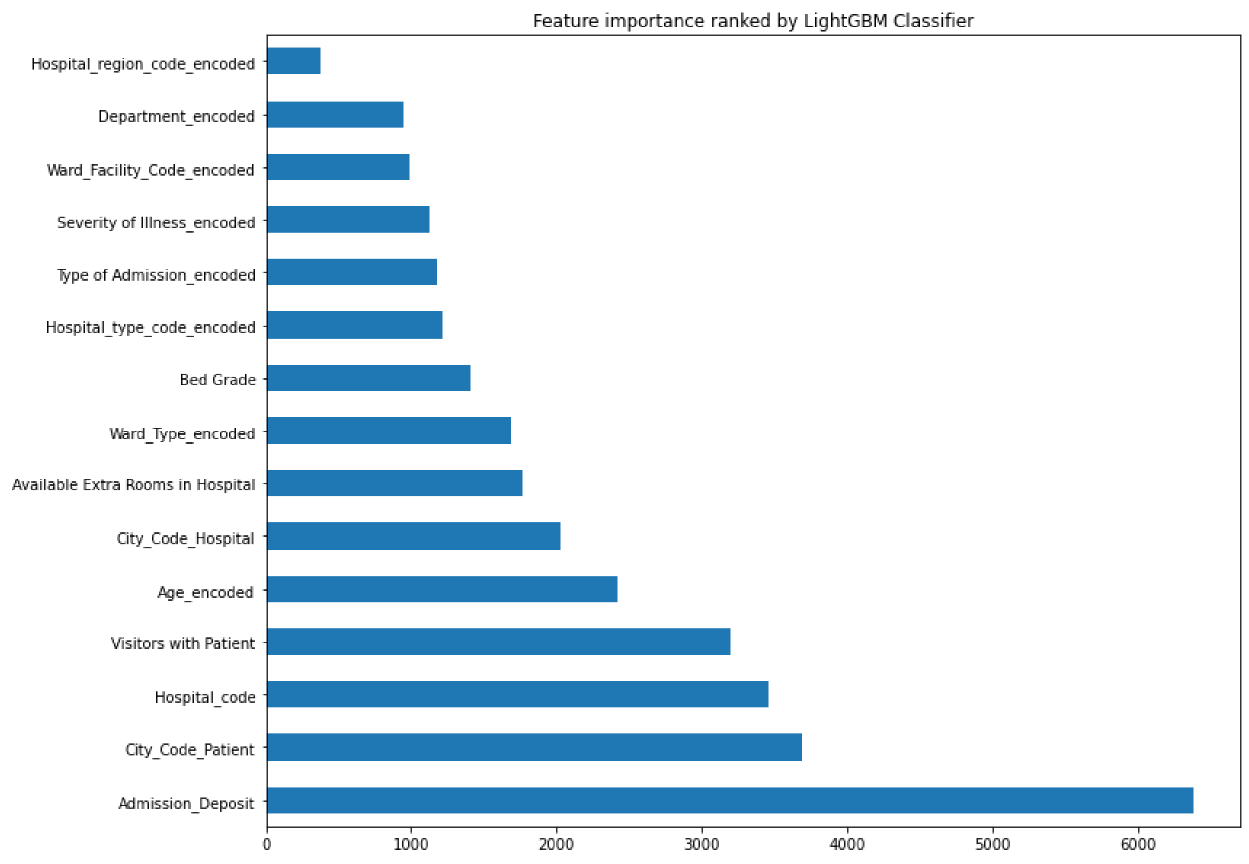


Figure 20. Feature Importance Ranked by LightGBM Classifier

From the results of feature importance picked by both the methods, if we pick the top 10 features, 8 of the features picked by both the methods are common. These features are:

- Admission Deposit
- Ward Type
- Visitors with Patient
- Available extra rooms in Hospital
- Type of Admission
- Hospital Type
- Bed Grade
- Ward Type

While some of these features did not match with our initial intuition to be the most important features, these methods gave us some meaningful insights about the relationships between the features within the dataset.

We ran a total of 3 different combinations, one with top 5 features, one with top 10 features, and one run with all the features included. The run with all the features included returned us with the highest accuracy, so we went with that for the final model. The difference in the final accuracy of the 3 different feature sets were around 2%. That made us arrive at the conclusion that even if some of the features are less important than others, it still has some impact on the final accuracy.

The Final list of features after combining manual and automatic feature selection:

- Hospital_code
- City_Code_Hospital
- Available Extra Rooms in Hospital
- Bed Grade
- City_Code_Patient
- Visitors with Patient
- Admission_Deposit
- Hospital_type_code_encoded
- Hospital_region_code_encoded
- Department_encoded
- Ward_Type_encoded
- Ward_Facility_Code_encoded
- Type of Admission_encoded
- Severity of Illness_encoded
- Age_encoded

Model Selection

Random Forest Classifier

Random Forest is an ensemble learning method where the collective knowledge of multiple individual decision trees is extracted to arrive at a classification.

The random forest implementation through scikit-learn library has multiple parameters that can be tweaked to produce better results.

Parameter	Description	Parameter Values for GridSearchCV
‘n_estimators’	the number of independent decision trees that are used to populate the ‘forest’	10 to 150, in steps of 20
‘max_depth’	the maximum depth the trees can go before their growth is terminated	5, 10
‘min_samples_split’	the minimum number of samples that is required to proceed with splitting the node	2, 5, 10
‘max_features’	the maximum number of features to consider while searching for best split	‘sqrt’ – where sqrt(num_features) are considered ‘log2’ – where log2(num_features) are considered None – where all the features are considered
‘bootstrap’	sample features with or without replacements	True / False

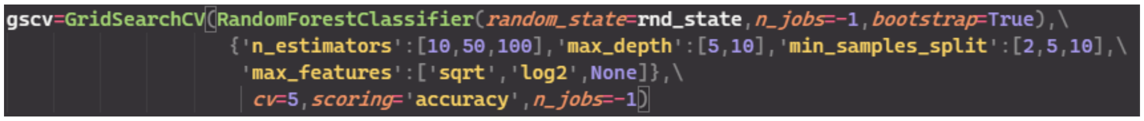
LightGBM

Light GBM is a gradient boosting framework that implements tree based learning algorithm. It differs from other tree based algorithms as it grows trees vertically instead of horizontally. This means the tree grows leaf-wise instead of level-wise. It chooses the leaf with max delta loss to grow.

Parameter	Description	Parameter Values for RandomizedSearchCV
‘n_estimators’	the number of independent decision trees that are used to populate the ‘forest’	100, 200
‘num_leaves’	the maximum number of leaves in one tree	20, 30, 40, 50
‘learning_rate’	shrinkage rate	0.05, 0.10, 0.15
‘min_split_gain’	The minimal gain to perform a split	0.01, 0.02
‘colsample_bytree’	randomly select a subset of feature on each tree	0.2, 0.4, 0.6, 0.8, 1.0
‘reg_alpha’	L1 regularization	0, 1, 2, 3
‘reg_lambda’	L2 regularization	0, 1, 2
‘min_child_weight’	minimal number of data in one leaf, used to help with over-fitting	30, 40, 50, 60, 80, 90, 100

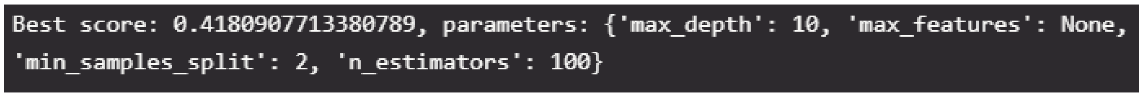
Tuning Model Hyperparameters

When evaluating a model one of the most important aspects is the parameters used in the model. We used both GridSearchCV (GSCV) and RandomizedSearchCV (RSCV) to tune the model. For the Random Forest Classifier we use both GSCV and RSCV. The dictionary parameters explored can be viewed below.



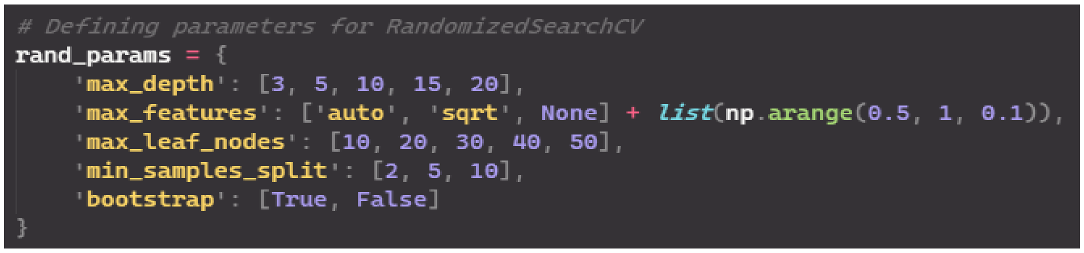
```
gscv=GridSearchCV(RandomForestClassifier(random_state=rnd_state,n_jobs=-1,bootstrap=True),\
                  {'n_estimators':[10,50,100],'max_depth':[5,10],'min_samples_split':[2,5,10],\
                   'max_features':['sqrt','log2',None]},\
                   cv=5,scoring='accuracy',n_jobs=-1)
```

Figure 21. GridSearchCV Dictionary RFC



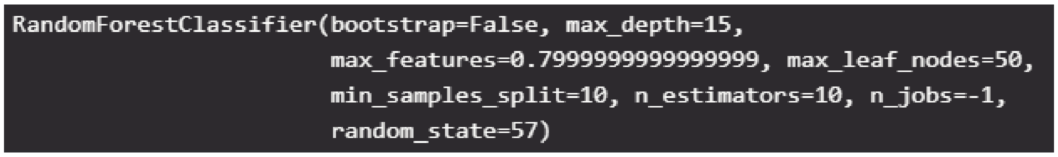
```
Best score: 0.4180907713380789, parameters: {'max_depth': 10, 'max_features': None,
'min_samples_split': 2, 'n_estimators': 100}
```

Figure 22. GridSearchCV Best Parameters RFC



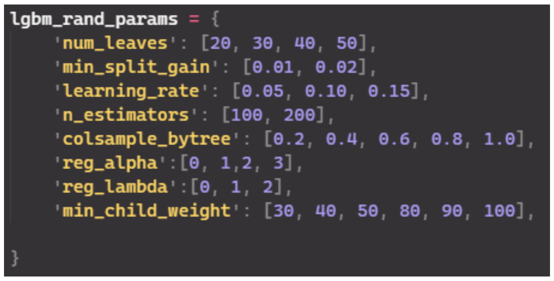
```
# Defining parameters for RandomizedSearchCV
rand_params = {
    'max_depth': [3, 5, 10, 15, 20],
    'max_features': ['auto', 'sqrt', None] + list(np.arange(0.5, 1, 0.1)),
    'max_leaf_nodes': [10, 20, 30, 40, 50],
    'min_samples_split': [2, 5, 10],
    'bootstrap': [True, False]
}
```

Figure 23. RandomizedSearchCV Dictionary RFC



```
RandomForestClassifier(bootstrap=False, max_depth=15,
                       max_features=0.7999999999999999, max_leaf_nodes=50,
                       min_samples_split=10, n_estimators=10, n_jobs=-1,
                       random_state=57)
```

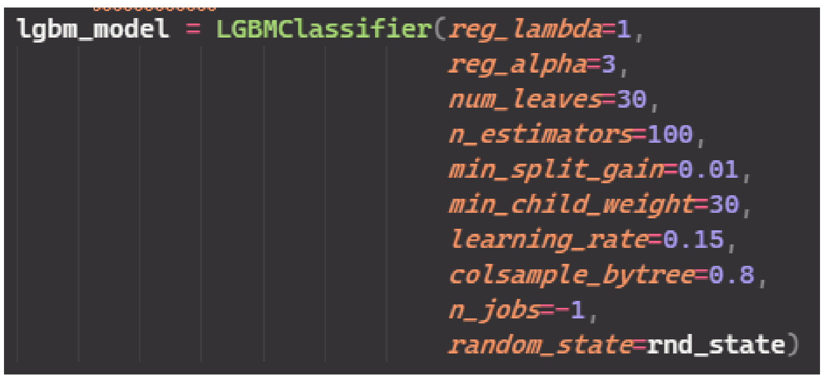
Figure 24. RandomizedSearchCV Best Parameters RFC



```
lgbm_rand_params = {
    'num_leaves': [20, 30, 40, 50],
    'min_split_gain': [0.01, 0.02],
    'learning_rate': [0.05, 0.10, 0.15],
    'n_estimators': [100, 200],
    'colsample_bytree': [0.2, 0.4, 0.6, 0.8, 1.0],
    'reg_alpha':[0, 1,2, 3],
    'reg_lambda':[0, 1, 2],
    'min_child_weight': [30, 40, 50, 80, 90, 100],

}
```

Figure 25. RandomizedSearchCV LightGBM Dictionary

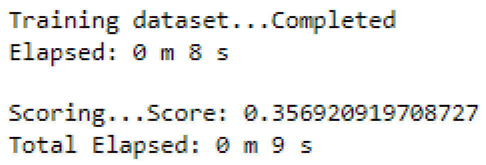


```
lgbm_model = LGBMClassifier(reg_lambda=1,
                            reg_alpha=3,
                            num_leaves=30,
                            n_estimators=100,
                            min_split_gain=0.01,
                            min_child_weight=30,
                            learning_rate=0.15,
                            colsample_bytree=0.8,
                            n_jobs=-1,
                            random_state=rnd_state)
```

Figure 26. RandomizedSearchCV LightGBM Best Parameters

Results

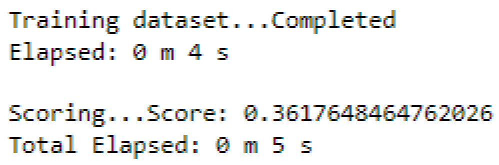
Accuracy & Benchmarks



```
Training dataset...Completed
Elapsed: 0 m 8 s

Scoring...Score: 0.356920919708727
Total Elapsed: 0 m 9 s
```

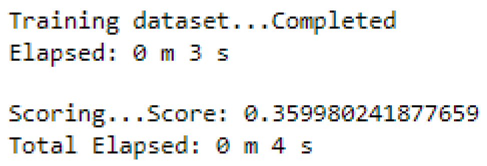
Figure 27. Accuracy obtained by dropping ['City_Code_Hospital', 'City_Code_Patient', 'Ward_Facility_Code_encoded'] Features



```
Training dataset...Completed
Elapsed: 0 m 4 s

Scoring...Score: 0.3617648464762026
Total Elapsed: 0 m 5 s
```

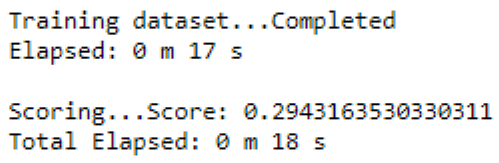
Figure 28. Accuracy obtained by dropping ['City_Code_Hospital', 'City_Code_Patient', 'Ward_Facility_Code_encoded', 'Hospital_code', 'Admission_Deposit'] Features



```
Training dataset...Completed
Elapsed: 0 m 3 s

Scoring...Score: 0.359980241877659
Total Elapsed: 0 m 4 s
```

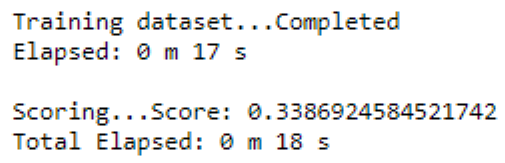
Figure 29. Accuracy obtained by dropping ['City_Code_Hospital', 'City_Code_Patient', 'Ward_Facility_Code_encoded', 'Hospital_code', 'Bed Grade', 'Admission_Deposit'] Features



```
Training dataset...Completed
Elapsed: 0 m 17 s

Scoring...Score: 0.2943163530330311
Total Elapsed: 0 m 18 s
```

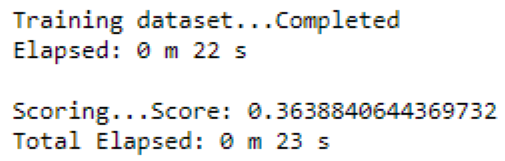
Figure 30. Accuracy obtained with PCA - 5 components



```
Training dataset...Completed
Elapsed: 0 m 17 s

Scoring...Score: 0.3386924584521742
Total Elapsed: 0 m 18 s
```

Figure 31. Accuracy obtained with PCA - 7 components

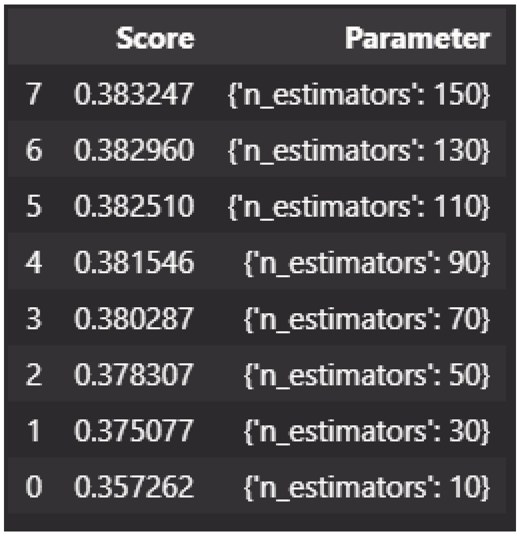


```
Training dataset...Completed
Elapsed: 0 m 22 s

Scoring...Score: 0.3638840644369732
Total Elapsed: 0 m 23 s
```

Figure 32. Accuracy obtained with PCA -10 components

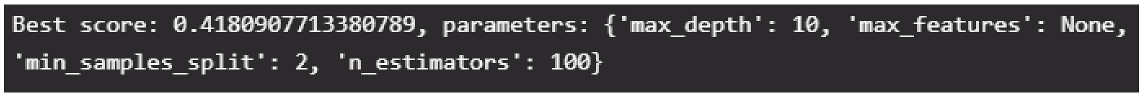
As seen in Figure 34, below, the bench mark accuracy for a Random Forest Classifier when run with all features and only estimators adjusted.



	Score	Parameter
7	0.383247	{'n_estimators': 150}
6	0.382960	{'n_estimators': 130}
5	0.382510	{'n_estimators': 110}
4	0.381546	{'n_estimators': 90}
3	0.380287	{'n_estimators': 70}
2	0.378307	{'n_estimators': 50}
1	0.375077	{'n_estimators': 30}
0	0.357262	{'n_estimators': 10}

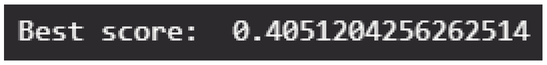
Figure 33. Accuracy Scores for different estimators for RFC

In Figure 35, below, the best parameters selected using a GridSearchCV. The parameters search criteria can be seen in Figure 22 in Methodology.



```
Best score: 0.4180907713380789, parameters: {'max_depth': 10, 'max_features': None,
'min_samples_split': 2, 'n_estimators': 100}
```

Figure 34. Score and best parameters for RFC using GridSearchCV parameters

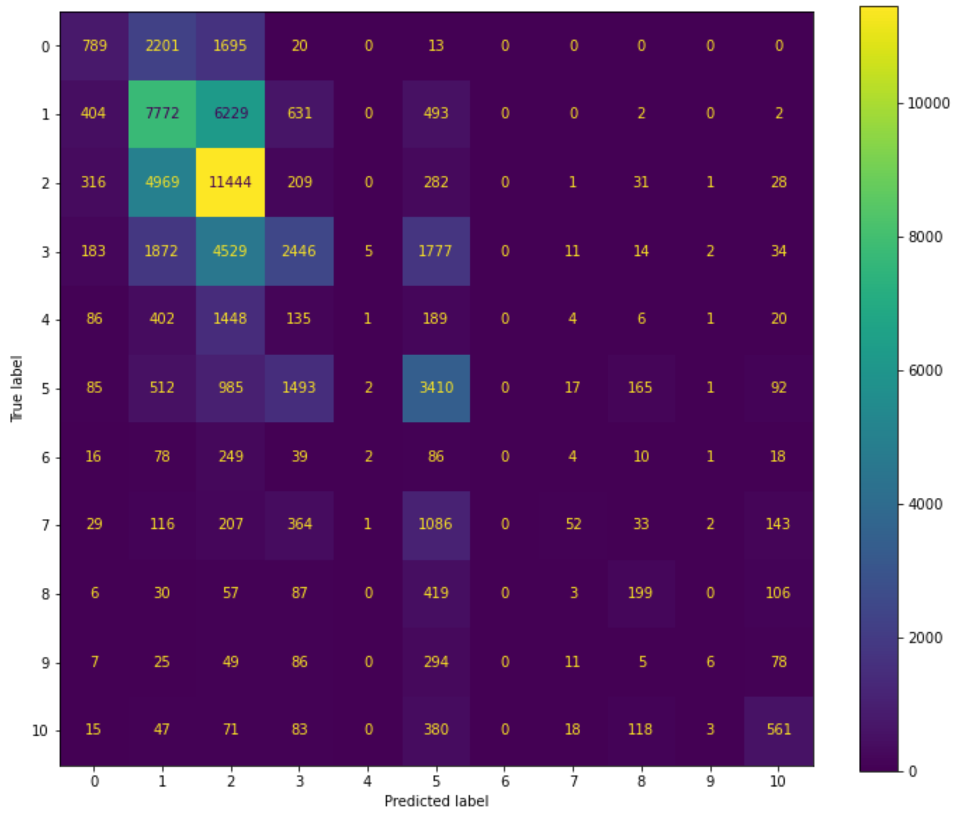


```
Best score:  0.4051204256262514
```

Figure 35. Score for using RandomizedSearchCV parameters with RFC model

Beyond that we ran the data through a LightGBM model using a RandomizedGridSearchCV to adjust the hyperparameters where we got a prediction accuracy of 42.51%.

Confusion Matrix



True label \ Predicted label	0	1	2	3	4	5	6	7	8	9	10
0	789	2201	1695	20	0	13	0	0	0	0	0
1	404	7772	6229	631	0	493	0	0	2	0	2
2	316	4969	11444	209	0	282	0	1	31	1	28
3	183	1872	4529	2446	5	1777	0	11	14	2	34
4	86	402	1448	135	1	189	0	4	6	1	20
5	85	512	985	1493	2	3410	0	17	165	1	92
6	16	78	249	39	2	86	0	4	10	1	18
7	29	116	207	364	1	1086	0	52	33	2	143
8	6	30	57	87	0	419	0	3	199	0	106
9	7	25	49	86	0	294	0	11	5	6	78
10	15	47	71	83	0	380	0	18	118	3	561

Figure 36. LightGBM Model Confusion Matrix

Discussion

We evaluated the model using accuracy score. As explained in our methodology the model was run initially using the top 4, top 8 and top 12 correlated features and then all features. We found that including all the features provided the highest accuracy. As reported in the results we noticed that a higher number of estimators resulted in a higher accuracy. Referring to Figure (results showing accuracy with estimators) you can see for the Random Forest Classifier model, the accuracy varied from 35.72% with 10 estimators to 38.32% with 150 estimators. From there the model was tuned using a combination of GridSearchCV and RandomizedSearchCV to provide the best parameters. The model was run again with the tuned parameters and 10, 50, and 100 estimators. Increasing the estimators from 10 lowered the model's accuracy. Based on this conclusion we would assume that 10 estimators would be the best when considering all the parameters.

While completing this analysis and subsequent report we noticed many challenges. The first, and largest, was time. We wanted to run more iterations and use a combination of GridSearchCV, along with RandomizedSearchCV to maximize the hyperparameters for the LightGBM model as well as calculate the ROC-AUC score but were unable to, due to time constraints.

Beyond that we ran an iterative train-test-split and included the stratify parameters. This dropped the model's accuracy significantly. This was unexpected since a stratified train-test-split should take an equal number of samples from each target class. We assumed that when we added the stratify option some of the classes didn't have sufficient samples to be accuracy predicted based on the dataset. This leads us to another observation and assumption where we believe we could increase the model's accuracy by reducing the number of target classes. This would be accomplished by combining target labels or binning the classes. It all comes back to time; we didn't have sufficient time to experiment with various combinations of classes.

Another observation was that the final accuracy scores did not have any differences running with or without Feature Scaling. This was right in-line with what our expectationswere, since decision tree-based models do not have a negative impact when trained with unscaled data.

We also believe additional features would've helped the model's accuracy. Such features as date and time of admission. On top of the features available, when you view the graphs showing the data distribution, there is clear bias towards the gynecology ward which would skew the model.

References

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

Image credits for OneHotEncoding vs Label Encoding: https://towardsdatascience.com/one-hot-encoding-is-making-your-tree-based-ensembles-worse-heres-why-d64b282b5769