

# AIN 413 MACHINE LEARNING FOR HEALTHCARE

## Project Report

ÇAĞRI ÇAKIROĞLU

2200765005

May 2024

# Contents

<b>1</b>	<b>Introduction</b>	<b>4</b>
<b>2</b>	<b>Dataset Information</b>	<b>4</b>
2.1	Data Types and Missing Values . . . . .	5
2.2	Data Summary . . . . .	5
<b>3</b>	<b>Exploratory Data Analysis (EDA)</b>	<b>5</b>
<b>4</b>	<b>Data Preprocessing</b>	<b>14</b>
4.1	Dropping Unnecessary Columns . . . . .	14
4.2	Handling Missing Values . . . . .	14
4.2.1	Bed Grade . . . . .	15
4.2.2	City Code for Patient . . . . .	15
4.3	Verification of Preprocessing . . . . .	15
4.4	Converting Categorical Ranges to Numerical Values . . . . .	15
4.4.1	Stay Duration . . . . .	15
4.4.2	Age . . . . .	15
4.4.3	Implementation . . . . .	16
4.5	Label Encoding of Categorical Columns . . . . .	16
4.6	Train-Test Split . . . . .	16
4.7	Feature Scaling and Importance . . . . .	17
4.7.1	Scaling the Features . . . . .	17
4.7.2	Training the Model . . . . .	17
4.7.3	Evaluating Feature Importances . . . . .	17
<b>5</b>	<b>Handling Class Imbalance and Feature Reduction</b>	<b>18</b>
5.1	Class Imbalance . . . . .	18
5.1.1	SMOTE with Minimum Class Count of 20,000 . . . . .	18
5.1.2	SMOTE with Maximum Class Count . . . . .	18
5.2	Principal Component Analysis (PCA) . . . . .	19
5.3	Datasets Prepared for Modeling . . . . .	19
5.4	Dataset Evaluation with LightGBM . . . . .	20
<b>6</b>	<b>Model Performance</b>	<b>20</b>
6.1	XGBoost . . . . .	21
6.2	Decision Tree . . . . .	21
6.3	Support Vector Machine (SVM) . . . . .	22
6.4	K-Nearest Neighbors (KNN) . . . . .	22
6.5	Logistic Regression with Polynomial Features . . . . .	23
6.6	CatBoost . . . . .	24
6.7	Random Forest . . . . .	25
<b>7</b>	<b>Results and Discussion</b>	<b>25</b>
7.1	Discussion of Model Performance . . . . .	26
7.1.1	LightGBM . . . . .	26
7.1.2	XGBoost . . . . .	26
7.1.3	Decision Trees . . . . .	26

7.1.4	Support Vector Machine (SVM)	26
7.1.5	K-Nearest Neighbors (KNN)	26
7.1.6	Logistic Regression	27
7.1.7	CatBoost	27
7.1.8	Random Forest	27
7.2	Conclusion	27
<b>8</b>	<b>Further Improvements</b>	<b>27</b>
8.1	Testing Multiple Datasets Across All Models	27
8.2	Avoiding Data Leakage	28
8.3	Hyperparameter Tuning	28
8.4	Computational Resources	28
8.5	Conclusion	28
<b>9</b>	<b>Notes</b>	<b>28</b>
<b>10</b>	<b>References</b>	<b>29</b>

# 1 Introduction

**Statement of the problem** The COVID-19 epidemic has underscored the crucial significance of effective healthcare administration. An essential factor in enhancing healthcare efficiency in hospitals is the precise prediction of the patient length of stay (LOS). Hospitals can use accurate LOS projections to identify high-risk patients, who are more likely to have longer stays, upon admission. Hospitals can enhance treatment plans, minimize length of stay (LOS), mitigate the risk of staff and visitor infection, and enhance overall resource management, including room and bed allocation, by promptly identifying these patients.

As a Data Scientist at HealthMan, a non-profit organization focused on efficient hospital management, your responsibility is to forecast the duration of hospitalization for individual patients. The duration of the stay is divided into 11 distinct categories, spanning from 0-10 days to over 100 days. Precise forecasts would empower hospitals to efficiently distribute resources and enhance operational efficacy.

**Approach** In order to accomplish this objective, an initial exploratory data analysis (EDA) was performed to gain insights into the dataset and uncover any discernible patterns and correlations within the data. Data preparation approaches, such as addressing missing values and outliers, were utilized to ensure the quality of the data. In addition, Principal Component Analysis (PCA) and Synthetic Minority Over-sampling Technique (SMOTE) were used to handle the problems of reducing dimensionality and balancing class distribution, respectively.

After completing the data preparation and feature engineering stages, many machine learning models were utilized to forecast the duration of hospitalization. The models' performance was assessed using measures such as F1 score, Matthews correlation coefficient (MCC), and accuracy. The evaluations yielded valuable insights into the efficacy of each model, which in turn informed the decision-making process for selecting the most appropriate model for deployment.

This report will include an in-depth study of the dataset, including exploratory data analysis, preprocessing methods, the machine learning models employed, and the resulting outcomes. The objective is to offer a thorough comprehension of the process and results, emphasizing the potential for enhancing healthcare management by means of precise length of stay estimates.

## 2 Dataset Information

The dataset consists of 318,438 entries and 18 columns. Each column contains specific information about the patients and their hospital stay. Below is a detailed description of each column:

- **case\_id**: Unique identifier for each case registered in the hospital.
- **Hospital\_code**: Unique code for the hospital where the case was registered.
- **Hospital\_type\_code**: Unique code for the type of hospital.
- **City\_Code\_Hospital**: City code of the hospital.
- **Hospital\_region\_code**: Region code of the hospital.
- **Available Extra Rooms in Hospital**: Number of extra rooms available in the hospital.
- **Department**: Department overseeing the case.
- **Ward\_Type**: Code for the type of ward.
- **Ward\_Facility\_Code**: Code for the ward facility.

- **Bed Grade:** Condition of the bed in the ward (note: some entries are missing).
- **patientid:** Unique identifier for each patient.
- **City\_Code\_Patient:** City code for the patient (note: some entries are missing).
- **Type of Admission:** Type of admission registered by the hospital.
- **Severity of Illness:** Severity of the illness recorded at the time of admission.
- **Visitors with Patient:** Number of visitors accompanying the patient.
- **Age:** Age of the patient.
- **Admission\_Deposit:** Deposit paid at the time of admission.
- **Stay:** Length of stay by the patient, categorized into different classes.

## 2.1 Data Types and Missing Values

- **case\_id, Hospital\_code, City\_Code\_Hospital, Available Extra Rooms in Hospital, patientid, Visitors with Patient:** These columns contain integer values with no missing data.
- **Hospital\_type\_code, Hospital\_region\_code, Department, Ward\_Type, Ward\_Facility\_Code, Type of Admission, Severity of Illness, Age, Stay:** These columns contain object (string) data types.
- **Bed Grade:** This column contains floating-point numbers, with some missing values.
- **City\_Code\_Patient:** This column contains floating-point numbers, with some missing values.
- **Admission\_Deposit:** This column contains floating-point numbers with no missing data.

## 2.2 Data Summary

- The dataset provides comprehensive information about patients and their hospital stays.
- Some columns, such as **Bed Grade** and **City\_Code\_Patient**, have missing values that need to be handled during data preprocessing.
- The target variable for the prediction task is **Stay**, which is categorized into 11 different classes representing the length of stay. The classes range from 0-10 days, 10-20 days, 20-30 days, up to more than 100 days.

This dataset will be used to predict the length of stay of patients in the hospital, helping in resource management and optimizing hospital operations.

## 3 Exploratory Data Analysis (EDA)

Exploratory Data Analysis (EDA) is a critical step in understanding the underlying patterns and relationships in the dataset. The following plots provide insights into various aspects of the dataset.

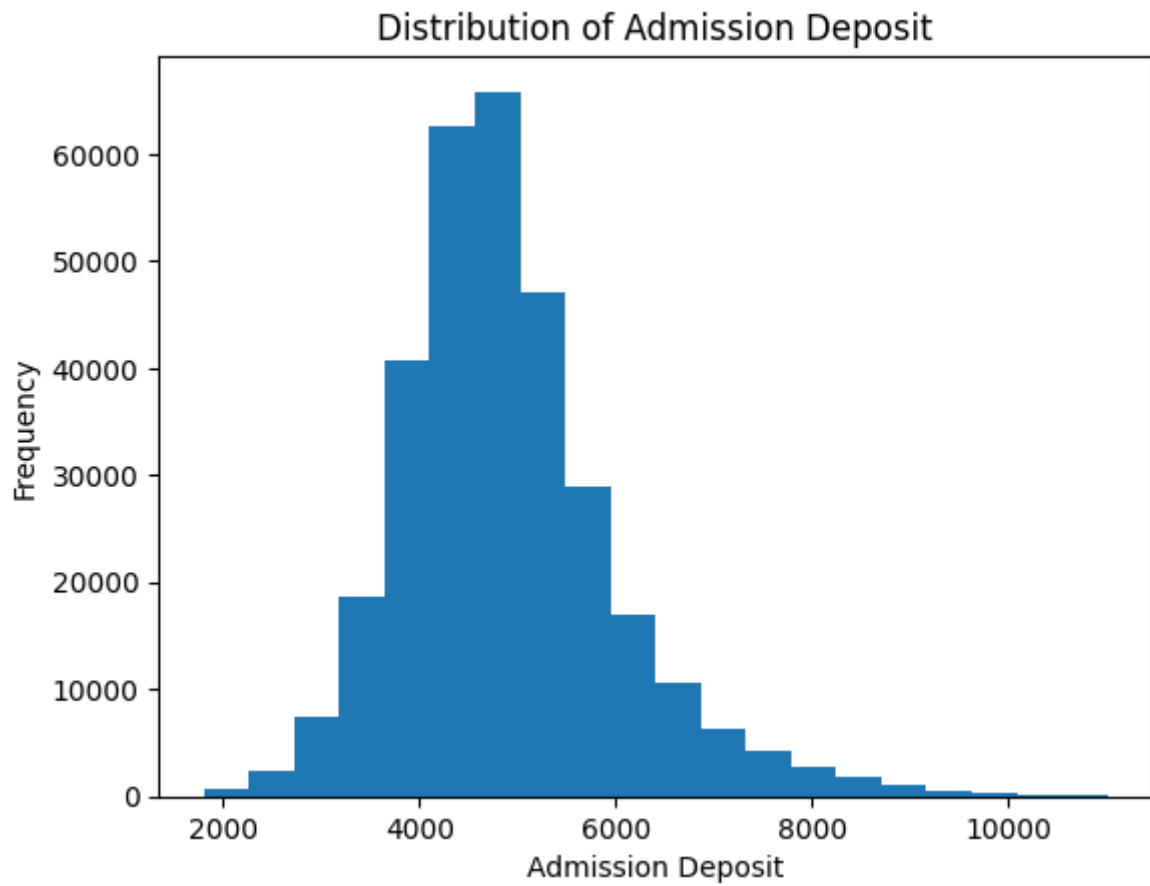


Figure 1: Distribution of Admission Deposit

This histogram shows the distribution of the admission deposit amounts. The majority of the deposits fall within the 3000 to 7000 range, indicating a somewhat normal distribution with a slight right skew.

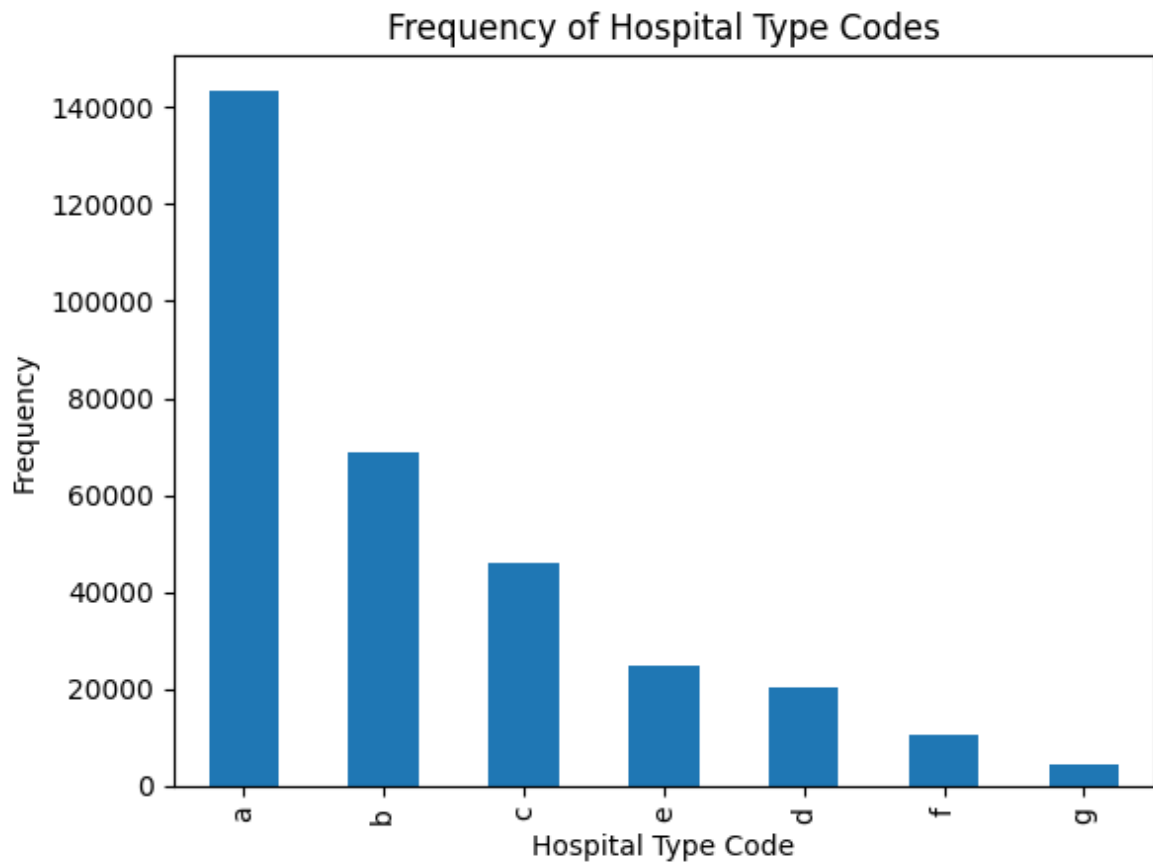


Figure 2: Frequency of Hospital Type Codes

This bar plot illustrates the frequency of different hospital type codes. Type 'a' is the most common, followed by types 'b' and 'c'.

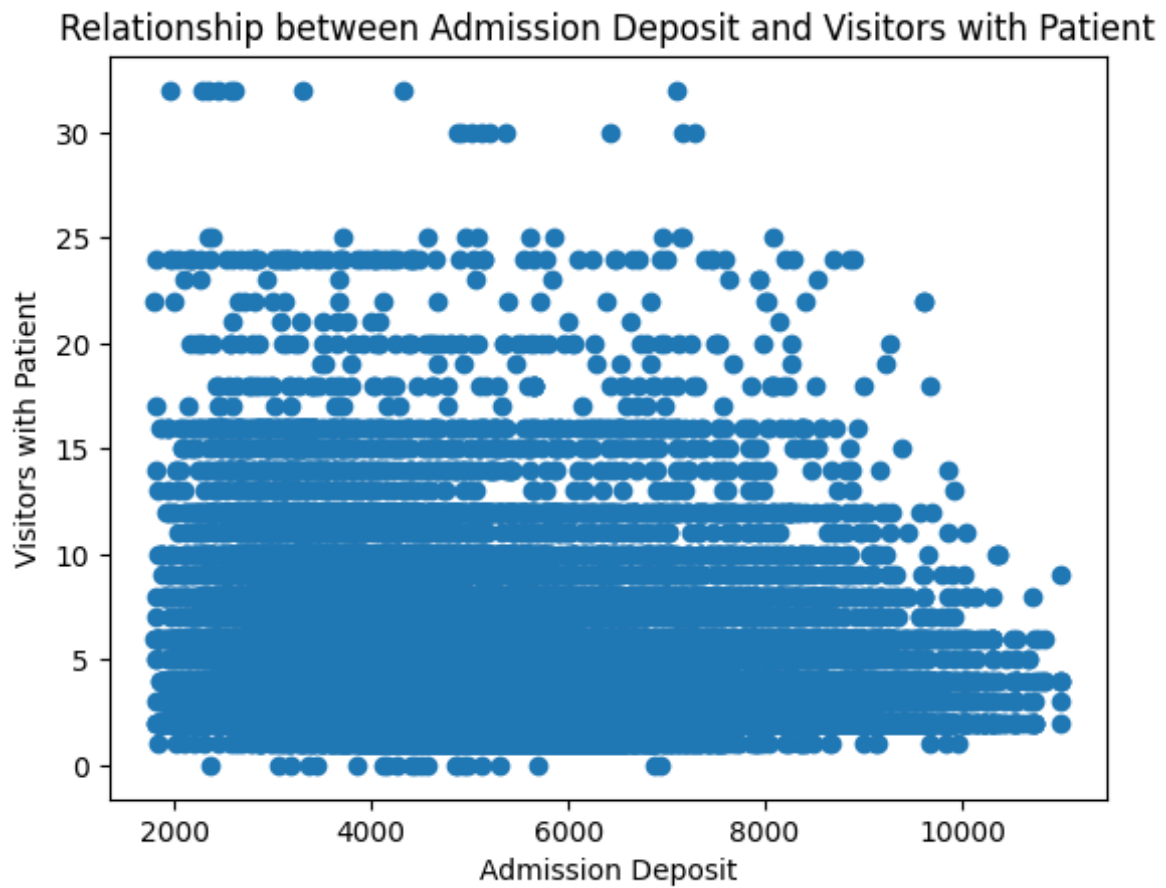


Figure 3: Relationship between Admission Deposit and Visitors with Patient

The scatter plot shows the relationship between admission deposit and the number of visitors with the patient. There is no clear linear relationship, suggesting that the number of visitors does not directly influence the admission deposit.



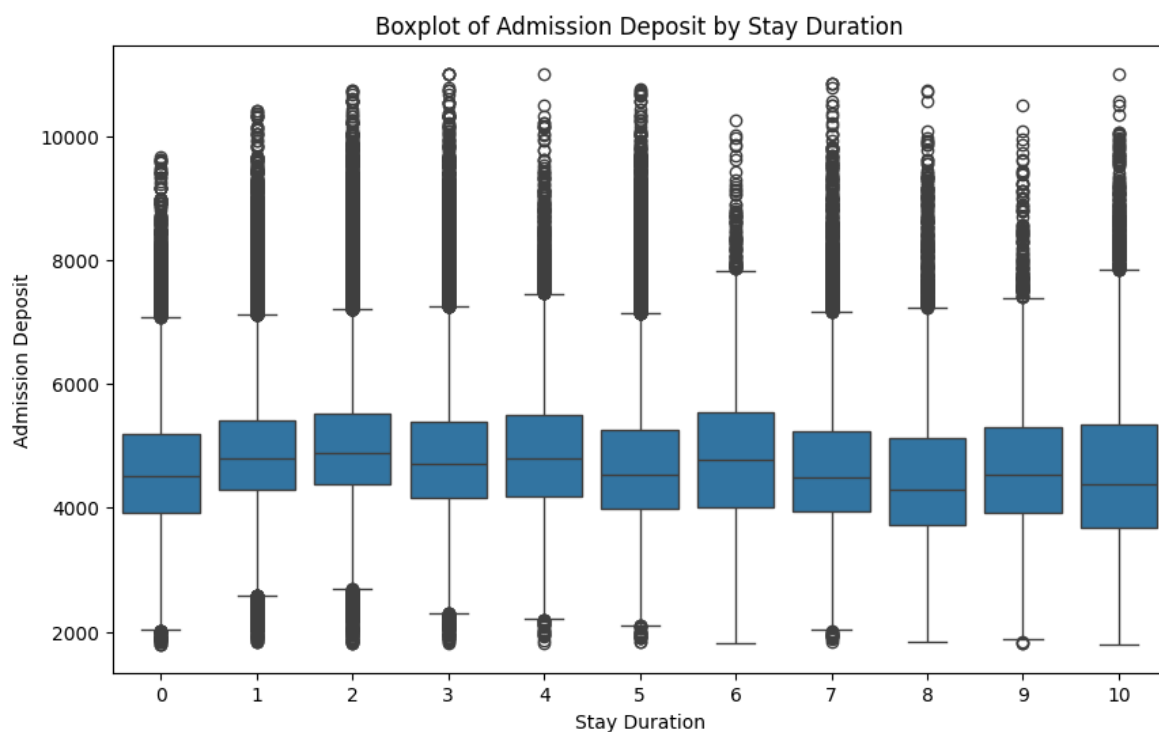


Figure 4: Boxplot of Admission Deposit by Stay Duration

The boxplot displays the distribution of admission deposits for different stay durations. It highlights the variations and outliers in deposit amounts across different stay duration classes.

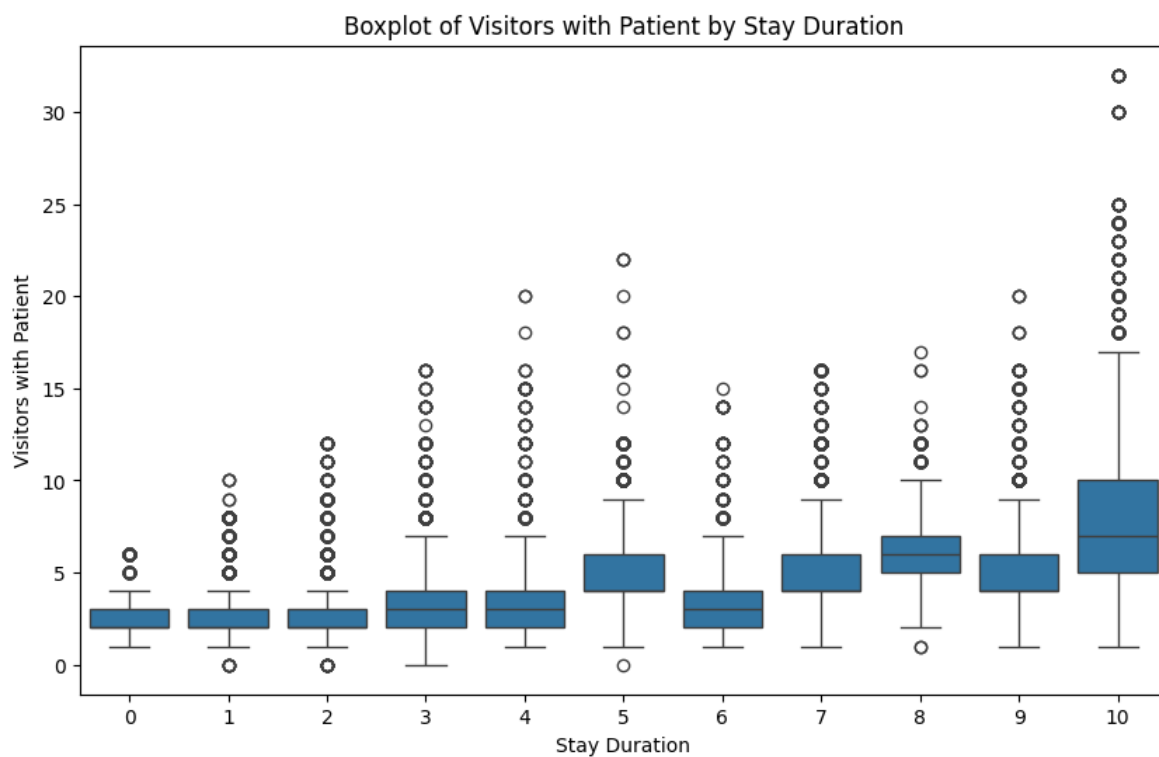


Figure 5: Boxplot of Visitors with Patient by Stay Duration

This boxplot shows the number of visitors with the patient across different stay duration classes. Longer stays tend to have higher variability in the number of visitors.

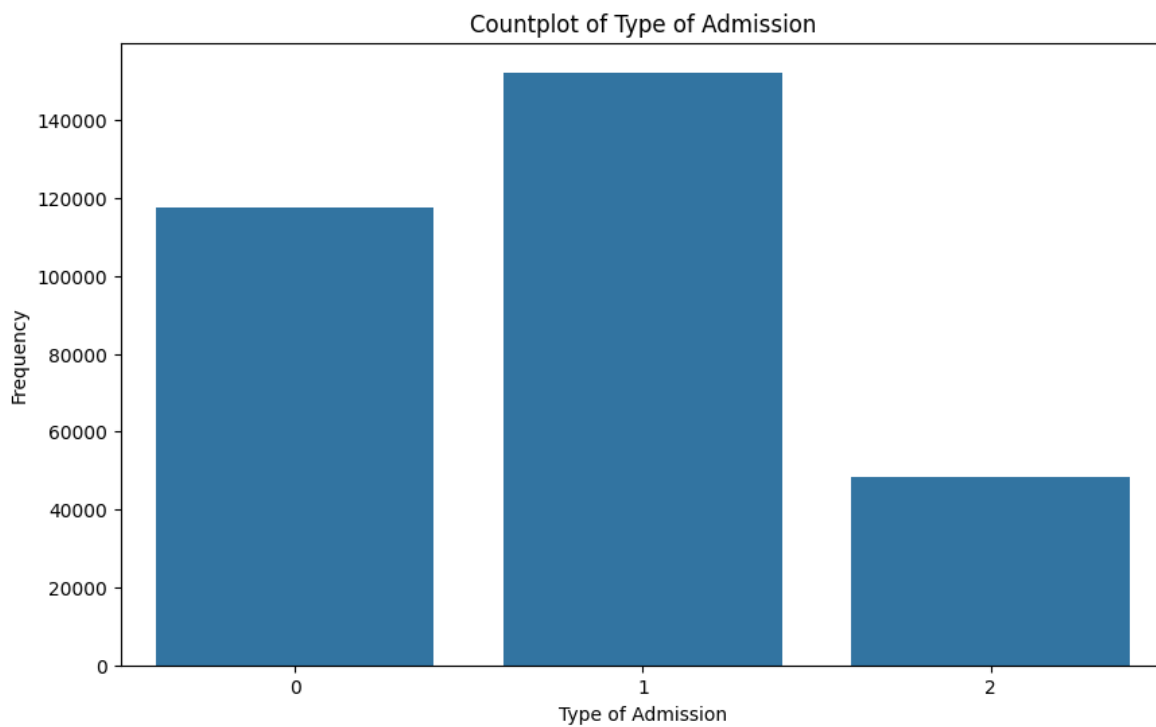


Figure 6: Countplot of Type of Admission

The count plot represents the frequency of different types of admissions. Type '1' admissions are the most frequent, followed by type '0'.

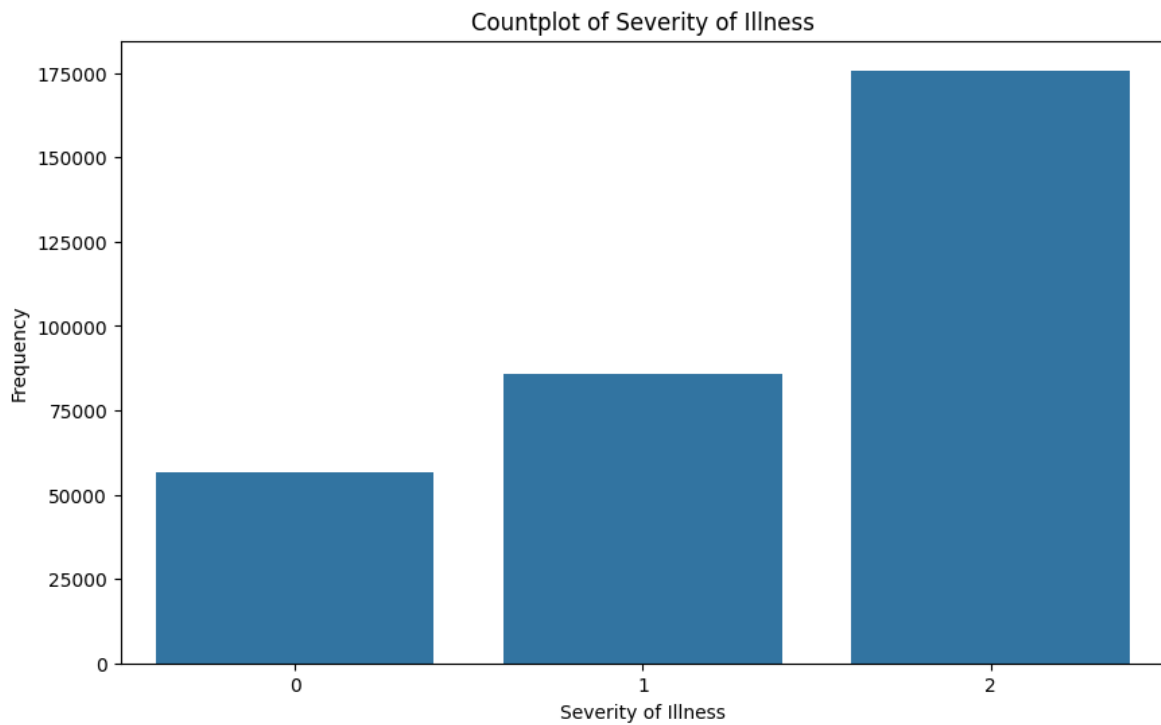


Figure 7: Countplot of Severity of Illness

This plot displays the distribution of the severity of illness among patients. Severity '2' is the most common, indicating a significant number of patients with severe conditions.

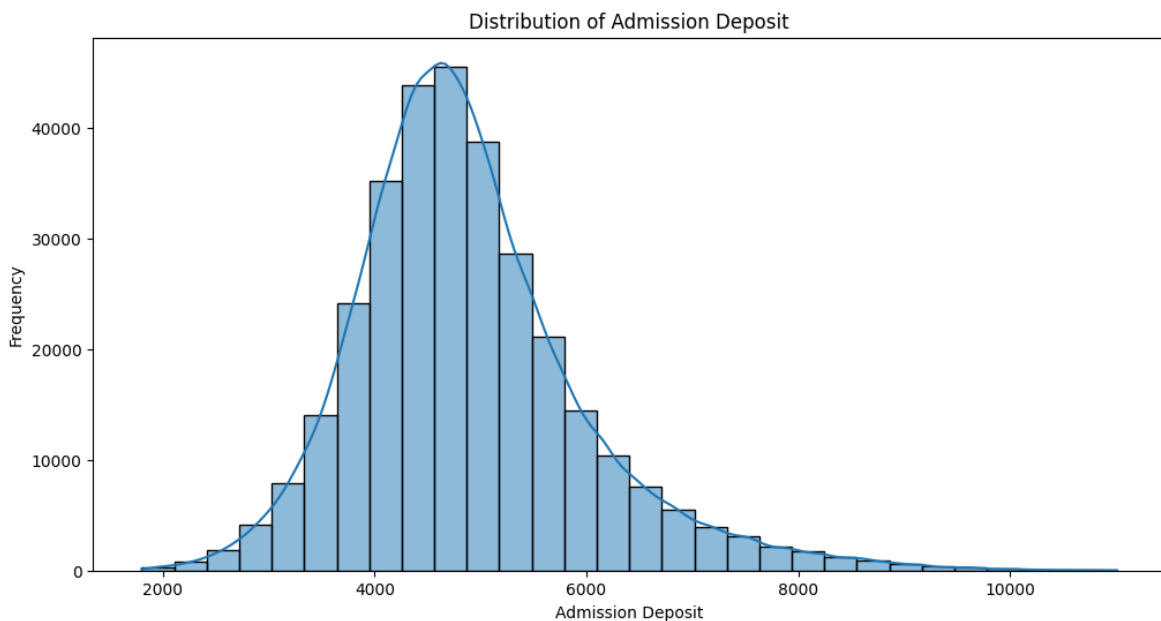


Figure 8: Distribution of Admission Deposit with Density Curve

This histogram with a density curve further emphasizes the distribution of admission deposits, showing a peak around 4000-5000 units.

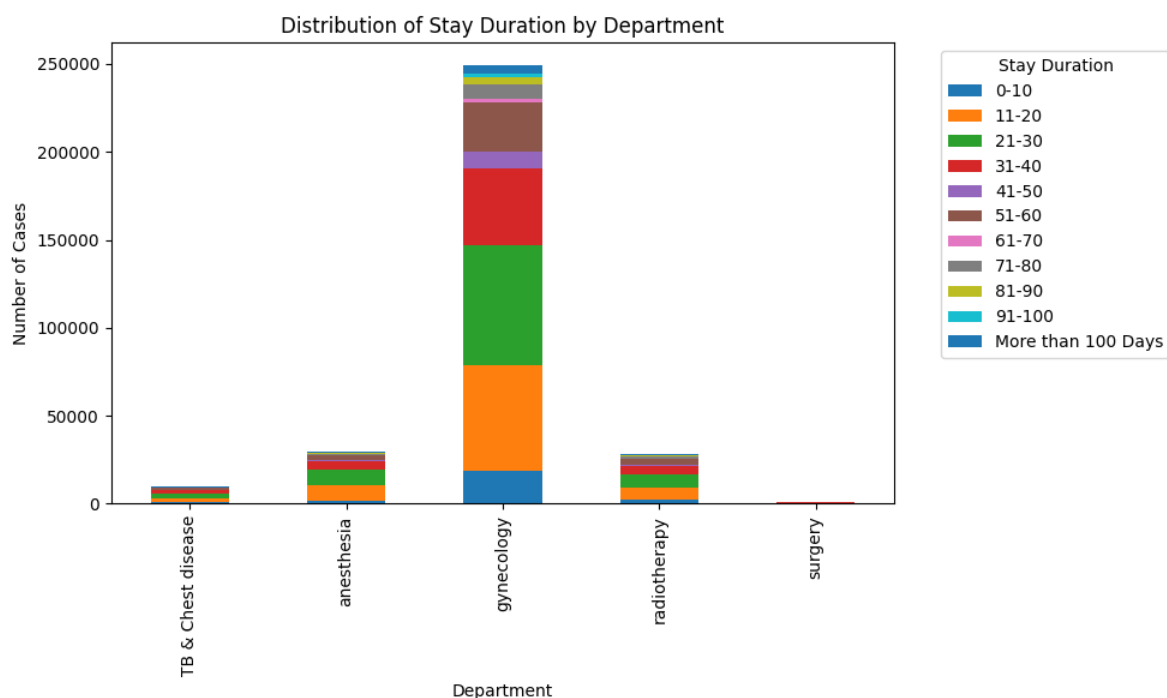


Figure 9: Distribution of Stay Duration by Department

The bar plot shows the distribution of stay durations across different departments. The gynecology department has the highest number of cases, with varying stay durations.

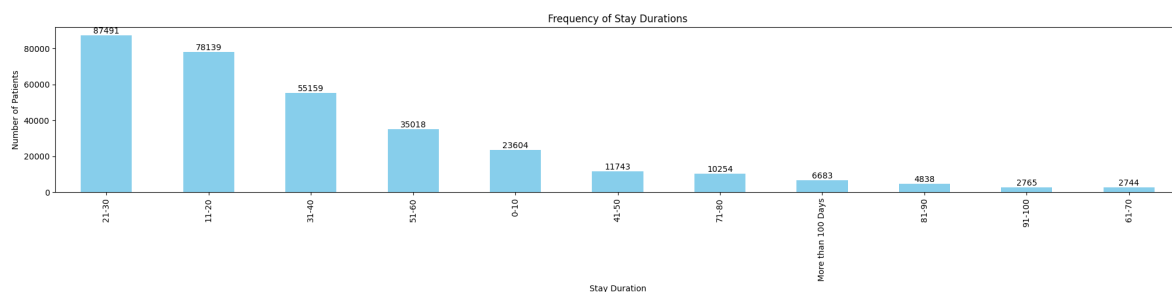


Figure 10: Frequency of Stay Durations

This bar plot shows the frequency of different stay durations among patients. The stay durations are divided into various ranges such as 0-10, 11-20, 21-30, and so on. The most frequent stay duration range is 21-30 days, followed by 11-20 days. The least frequent stay duration ranges are 61-70 days, 91-100 days, and more than 100 days. This distribution highlights that most patients tend to stay between 11 to 30 days, with a sharp decline in the number of patients staying beyond 50 days.

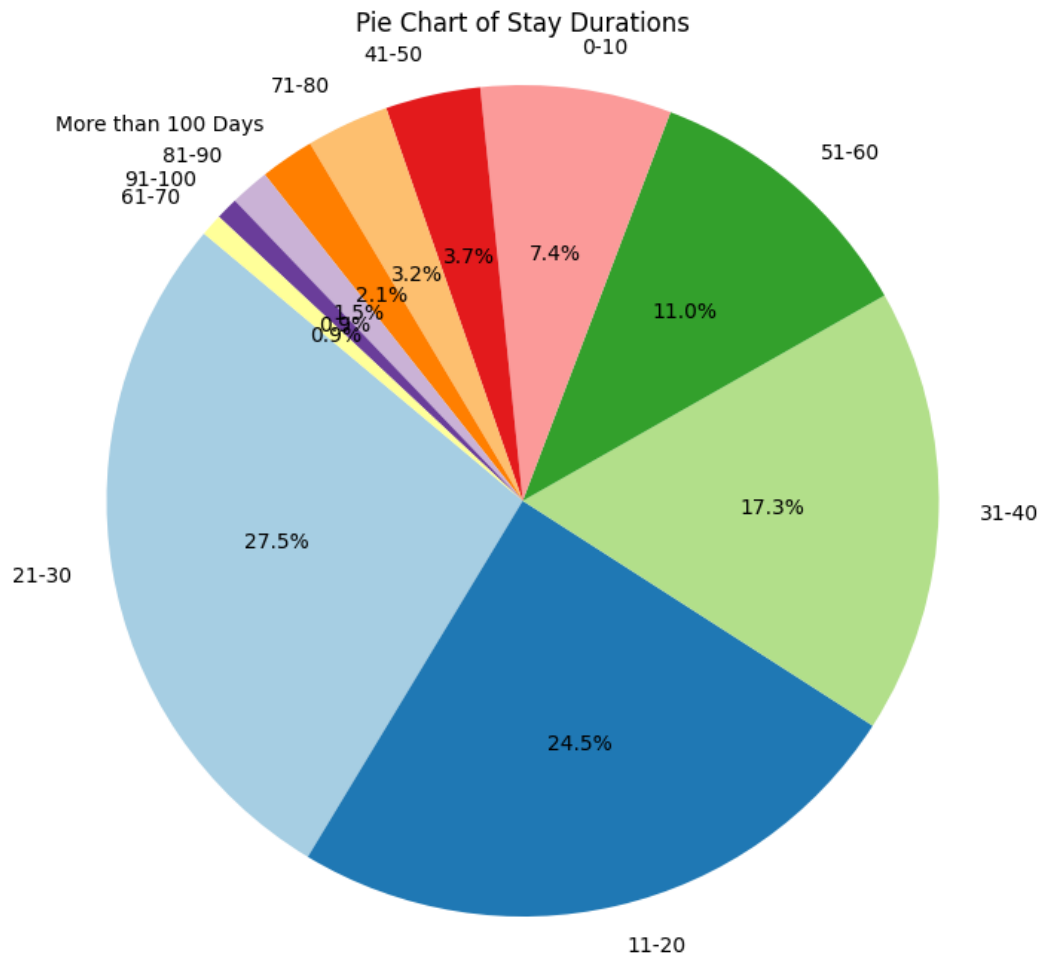


Figure 11: Pie Chart of Stay Durations

This pie chart represents the percentage distribution of various stay durations among patients. It provides a visual understanding of the proportion of patients falling into each stay duration category. The largest segment is for the 21-30 days stay duration, making up 27.5% of the total, followed by the 11-20 days duration at 24.5%. The smallest segments are for the 61-70 days (0.9%), 91-100 days (0.9%), and more than 100 days (2.1%). This visualization confirms that the majority of patients stay between 11 to 30 days, similar to the bar plot, and provides a clear visual representation of the distribution.

Sankey Diagram of Patient Flow from Departments to Stay Durations

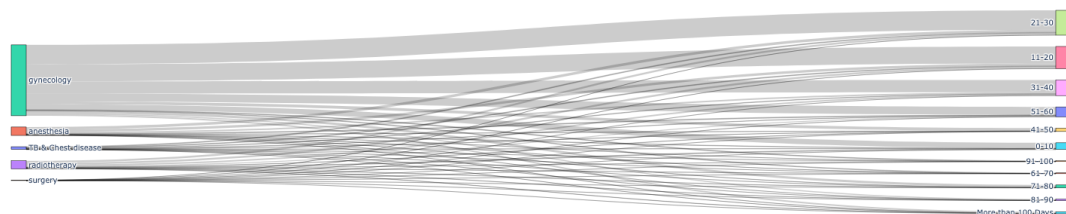


Figure 12: Sankey Diagram of Patient Flow from Departments to Stay Durations

The Sankey diagram visualizes the flow of patients from different departments to their respective stay durations. It provides an overview of how patients move through the hospital system and the distribution of their stay durations.

Sunburst Plot of Stay Duration by Various Categories

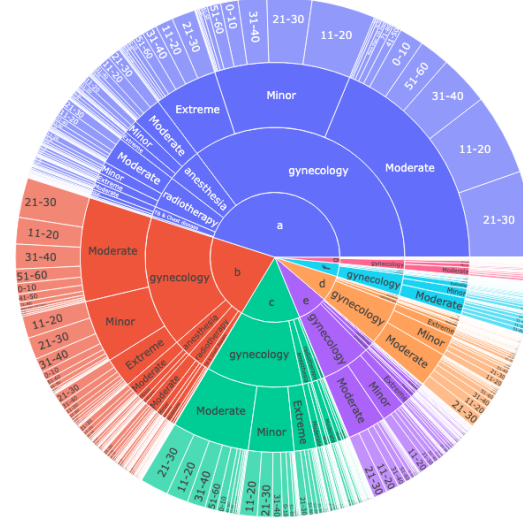


Figure 13: Sunburst diagram

This sunburst plot visualizes the stay duration of patients across various categories such as hospital type, department, and severity of illness. Each segment represents a different category and the hierarchical relationships between them, providing a comprehensive overview of how different factors contribute to the length of stay.

## 4 Data Preprocessing

Data preprocessing is an essential step to ensure the quality and integrity of the dataset before applying any machine learning algorithms. The following steps were taken to preprocess the data:

### 4.1 Dropping Unnecessary Columns

Certain columns were deemed unnecessary for the analysis and were therefore dropped from the dataset. Specifically, the `Hospital_region_code` and `City_Code_Hospital` columns were removed:

```
data = data.drop(['Hospital_region_code', 'City_Code_Hospital'], axis=1)
```

### 4.2 Handling Missing Values

Missing values in critical columns can significantly impact the performance of machine learning models. Therefore, appropriate strategies were employed to handle missing values:

### 4.2.1 Bed Grade

The `Bed Grade` column had some missing values. The mode (most frequent value) of the column was used to fill in these missing values:

```
bed_grade_mode = data['Bed Grade'].mode()[0]
# [0] is used to select the first mode in case there are multiple modes

# Fill in null values in the 'Bed Grade' column with the mode
data['Bed Grade'].fillna(bed_grade_mode, inplace=True)

# Optionally, verify that there are no more null values in the 'Bed Grade' column
print(data['Bed Grade'].isnull().sum()) # This should print 0
```

### 4.2.2 City Code for Patient

Similarly, the `City_Code_Patient` column had missing values which were filled with the mode of the column:

```
City_Code_Patient_mode = data['City_Code_Patient'].mode()[0] # [0] is used to select the first mode

# Fill in null values in the 'City_Code_Patient' column with the mode
data['City_Code_Patient'].fillna(City_Code_Patient_mode, inplace=True)

# Optionally, verify that there are no more null values in the 'City_Code_Patient' column
print(data['City_Code_Patient'].isnull().sum()) # This should print 0
```

## 4.3 Verification of Preprocessing

After filling the missing values, it was verified that there are no more null values in the `Bed Grade` and `City_Code_Patient` columns to ensure the data integrity before proceeding to the modeling phase.

## 4.4 Converting Categorical Ranges to Numerical Values

To facilitate easier prediction and improve the performance of machine learning models, categorical ranges in the `Stay` and `Age` columns were converted to numerical values. This transformation allows the models to process these columns more effectively.

### 4.4.1 Stay Duration

The `Stay` column, which represents the length of stay of patients, was originally categorized into several ranges. These ranges were mapped to numerical values.

### 4.4.2 Age

Similarly, the `Age` column, which represents the age of patients, was also categorized into several ranges. These ranges were mapped to numerical values.

### 4.4.3 Implementation

By converting these categorical ranges to numerical values, the dataset becomes more suitable for machine learning algorithms, enabling more accurate and efficient predictions.

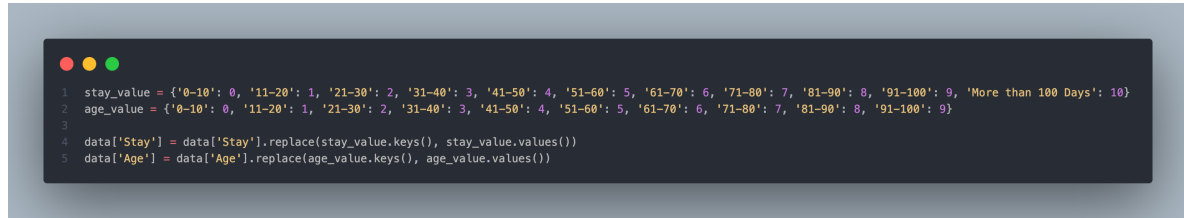


Figure 14: Implementation

## 4.5 Label Encoding of Categorical Columns

To prepare the dataset for machine learning algorithms, it was necessary to convert categorical columns into a numerical format. This process was accomplished using label encoding. The following categorical columns were identified and encoded:

- Department
- Hospital\_type\_code
- Ward\_Facility\_Code
- Ward\_Type
- Type of Admission
- Severity of Illness

A label encoder was applied to each of these categorical columns, transforming the categorical values into unique numerical labels. This step ensures that machine learning algorithms can interpret these features effectively.

Furthermore, any boolean columns in the dataset were converted to integer format (0 and 1) to standardize the data representation. After these transformations, the dataset was ready for the modeling phase.

## 4.6 Train-Test Split

To evaluate the performance of the machine learning models, the dataset was divided into training and testing sets. This ensures that the models are trained on one portion of the data and tested on an unseen portion to assess their generalization capabilities. The target variable for prediction is **Stay**, while the remaining columns serve as features.

The **case\_id** column was excluded from the feature set as it serves as a unique identifier and does not contribute to the prediction task. The following steps were taken for the train-test split:

- The target variable **Stay** was separated from the feature set.
- The feature set and target variable were then split into training and testing sets.
- A stratified split was used to ensure that the distribution of the target variable remains consistent across both the training and testing sets.
- 30% of the data was allocated to the test set, while the remaining 70% was used for training.



## 4.7 Feature Scaling and Importance

To ensure that the features are on a similar scale and improve the performance of the machine learning models, feature scaling was applied. The `StandardScaler` from scikit-learn was used to standardize the features by removing the mean and scaling to unit variance.

### 4.7.1 Scaling the Features

The scaler was fitted on the training data and then used to transform both the training and test sets:

- The training data was scaled using `fit_transform` method.
- The test data was scaled using the `transform` method, ensuring the same scaling is applied to both sets.

### 4.7.2 Training the Model

A `RandomForestRegressor` model was trained on the scaled training data to predict the length of stay. The model's feature importances were then extracted to understand which features contribute most to the predictions.

### 4.7.3 Evaluating Feature Importances

The feature importances from the trained model were plotted to visualize the significance of each feature in predicting the length of stay. Below is the bar plot showing the feature importances:

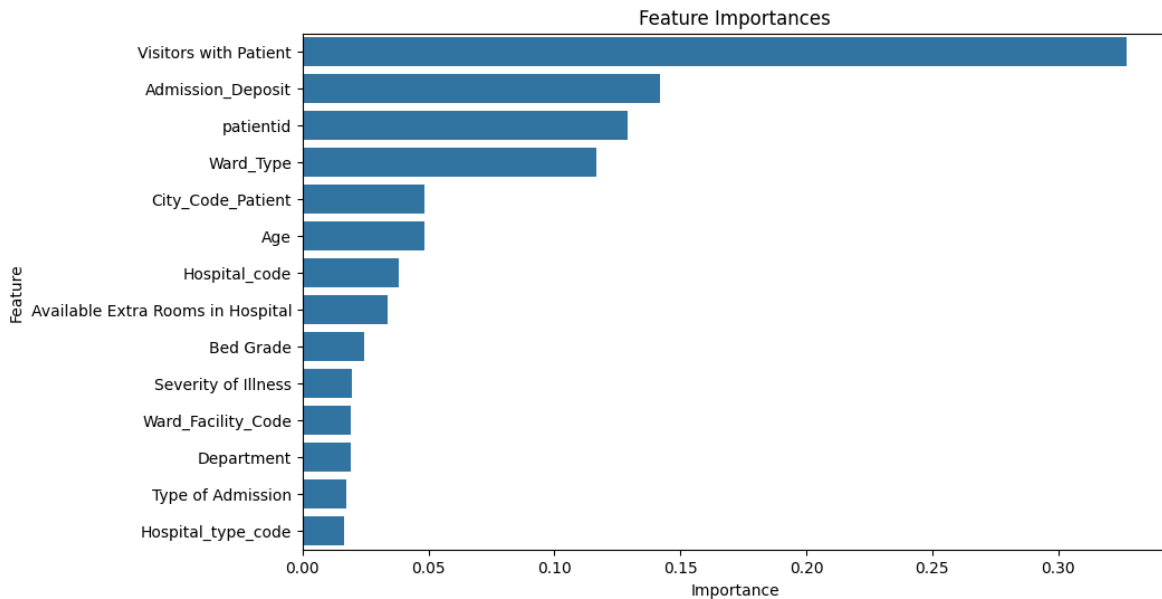


Figure 15: Feature Importances

The bar plot above shows the importance of each feature in the random forest model. The most significant feature is the number of visitors with the patient, followed by the admission deposit and patient ID. Other important features include ward type, city code of the patient, and age. Features such as hospital type code and type of admission have relatively lower importance in predicting the length of stay.

## 5 Handling Class Imbalance and Feature Reduction

In this section, we address the issue of class imbalance in the dataset by applying Synthetic Minority Over-sampling Technique (SMOTE) and subsequently use Principal Component Analysis (PCA) for feature reduction. This preprocessing ensures the data is well-prepared for machine learning models.

### 5.1 Class Imbalance

The dataset exhibited significant class imbalance, as shown by the distribution of the target variable **Stay**:

- Class 0: 23,604 instances (7.41%)
- Class 1: 78,139 instances (24.54%)
- Class 2: 87,491 instances (27.48%)
- Class 3: 55,159 instances (17.32%)
- Class 4: 11,743 instances (3.69%)
- Class 5: 35,018 instances (11.00%)
- Class 6: 2,744 instances (0.86%)
- Class 7: 10,254 instances (3.22%)
- Class 8: 4,838 instances (1.52%)
- Class 9: 2,765 instances (0.87%)
- Class 10: 6,683 instances (2.10%)

To mitigate the class imbalance, the Synthetic Minority Over-sampling Technique (SMOTE) was applied in two phases:

#### 5.1.1 SMOTE with Minimum Class Count of 20,000

First, classes with fewer than 20,000 instances were oversampled to have at least 20,000 instances. This ensured that minority classes were adequately represented without excessively increasing the dataset size.

- The target count for each class was set to a minimum of 20,000 instances.
- SMOTE was applied to the training data to achieve the desired distribution.

#### 5.1.2 SMOTE with Maximum Class Count

Next, each class was oversampled to match the size of the largest class (87,491 instances). This provided a balanced dataset for model training.


- The target count for each class was set to the maximum class count.
- SMOTE was applied again to the training data to achieve a fully balanced dataset.

## 5.2 Principal Component Analysis (PCA)

Principal Component Analysis (PCA) was applied to the original dataset to reduce the feature dimensionality while retaining 95% of the variance. This step was taken to explore if reducing the dimensionality could improve model performance by potentially reducing noise and overfitting.

- PCA was fitted on the scaled training data.
- The transformed training and test sets were obtained by applying the fitted PCA.

The PCA transformation process involved the following:



```
1
2
3  pca = PCA(n_components=0.95) # Keep 95% of the variance
4  X_train_pca = pca.fit_transform(X_train_scaled)
5  X_test_pca = pca.transform(X_test_scaled)
6
7  # Optional: Check the explained variance ratio
8  print("Explained variance ratio by each principal component:")
9  print(pca.explained_variance_ratio_)
10
11 # Display the first few rows of the PCA-transformed training data
12 print(pd.DataFrame(X_train_pca, columns=[f'PC{i+1}' for i in range(X_train_pca.shape[1])]).head())
13
14
15
16 # Display the first few rows of the PCA-transformed test data
17 print(pd.DataFrame(X_test_pca, columns=[f'PC{i+1}' for i in range(X_test_pca.shape[1])]).head())
18
```

Figure 16: PCA

## 5.3 Datasets Prepared for Modeling

After preprocessing, four different datasets were prepared for modeling:

- **Original Data:** The original dataset without any oversampling or dimensionality reduction.
- **SMOTE with Minimum Class Count of 20,000:** Dataset where classes with fewer than 20,000 instances were oversampled to have at least 20,000 instances.
- **SMOTE with Maximum Class Count:** Dataset where each class was oversampled to match the size of the largest class.
- **PCA Applied on Original Data:** The original dataset with PCA applied to retain 95% of the variance.

These different datasets allow us to evaluate the impact of handling class imbalance and feature reduction on the performance of machine learning models. The next steps involve training and evaluating models on these datasets to compare their F1, MCC, and accuracy metrics.

## 5.4 Dataset Evaluation with LightGBM

To evaluate the performance of the preprocessing techniques and their impact on the prediction of patient stay duration, a LightGBM classifier was trained and tested on each of the prepared datasets. The datasets include:

- **Original Data:** The original dataset without any oversampling or dimensionality reduction.
- **SMOTE with Minimum Class Count of 20,000:** Dataset where classes with fewer than 20,000 instances were oversampled to have at least 20,000 instances.
- **SMOTE with Maximum Class Count:** Dataset where each class was oversampled to match the size of the largest class.
- **PCA Applied on Original Data:** The original dataset with PCA applied to retain 95% of the variance.

The LightGBM classifier was evaluated using the following metrics:

- **Accuracy:** The percentage of correctly classified instances.
- **F1 Score:** The weighted average of precision and recall.
- **Matthews Correlation Coefficient (MCC):** A measure of the quality of binary and multiclass classifications.

The results for each dataset are summarized in Table 1.

Table 1: Comparison of Model Performance across Different Datasets

Dataset	Train Accuracy (%)	Test Accuracy (%)	F1 Score (%)	MCC
PCA Data	49.66	39.36	35.66	0.23
SMOTE to Max Data	46.74	39.90	37.55	0.25
SMOTE 20k Data	45.81	42.03	38.69	0.26
Scaled-Original Data	48.03	42.37	39.01	0.27

The table highlights the performance differences among the datasets. The **Scaled-Original Data** provided the best overall performance, with the highest test accuracy, F1 score, and MCC. The **SMOTE 20k Data** also performed well, demonstrating that balancing the classes to a reasonable extent can improve model performance. The **PCA Data**, while useful for reducing dimensionality, showed lower performance compared to the other datasets. Finally, the **SMOTE to Max Data** showed moderate performance improvements but was not as effective as the Scaled-Original Data or the SMOTE 20k Data.

## 6 Model Performance

This section evaluates the performance of various models trained on the preprocessed datasets. Each model's accuracy, F1 score, and Matthews Correlation Coefficient (MCC) were calculated to compare their effectiveness.

## 6.1 XGBoost

XGBoost (Extreme Gradient Boosting) was evaluated using a grid search to find the optimal hyperparameters. The parameter grid included variations in the number of estimators, learning rate, and maximum depth of the trees. Grid search was performed with cross-validation to ensure robust model selection.

- **Parameter Grid:**

- `n_estimators`: [50, 100, 150, 200]
- `learning_rate`: [0.01, 0.1, 0.005]
- `max_depth`: [3, 5, 7, 9]

- **Best Parameters Found:**

- `learning_rate` = 0.1
- `max_depth` = 7
- `n_estimators` = 100

The best model from the grid search was then evaluated on the test set. The results are summarized in Table 2.

Table 2: XGBoost Model Performance

<b>Metric</b>	<b>Value</b>
Accuracy (%)	42.31
F1 Score	38.69
MCC	0.265

The XGBoost model demonstrated an accuracy of 42.31%, an F1 score of 38.69%, and an MCC of 0.27. These metrics indicate that XGBoost performs reasonably well on the scaled-original dataset, although there is room for improvement. The hyperparameters tuned through grid search contributed to achieving these results.

## 6.2 Decision Tree

A Decision Tree classifier was evaluated using a grid search to find the optimal hyperparameters. The parameter grid included variations in the maximum depth of the tree, the minimum number of samples required to split an internal node, and the minimum number of samples required to be at a leaf node. Grid search was performed with cross-validation to ensure robust model selection.

- **Parameter Grid:**

- `max_depth`: [None, 10, 20, 30, 40, 50, 100, 200]
- `min_samples_split`: [5, 10, 20, 50]
- `min_samples_leaf`: [1, 2, 4, 7, 9]

- **Best Parameters Found:**

- `max_depth` = 10
- `min_samples_leaf` = 7

– `min_samples_split = 50`

The best model from the grid search was then evaluated on the test set. The results are summarized in Table 3.

Table 3: Decision Tree Model Performance

<b>Metric</b>	<b>Value</b>
Accuracy (%)	41.02
F1 Score	41.02
MCC	0.247

The Decision Tree model demonstrated an accuracy of 41.02%, an F1 score of 41.02%, and an MCC of 0.25. These metrics indicate that the Decision Tree classifier performs moderately well on the scaled-original dataset. The hyperparameters tuned through grid search contributed to achieving these results.

### 6.3 Support Vector Machine (SVM)

A Support Vector Machine (SVM) classifier was evaluated using a linear kernel. The model was configured with specific hyperparameters to optimize performance on the scaled dataset.

- **Model Parameters:**

- `C = 0.5`
- `kernel = 'linear'`
- `tol = 0.01`
- `cache_size = 5000`
- `max_iter = 10000`
- `random_state = 42`

The SVM classifier was trained on the scaled training data and evaluated on the scaled test data. The results are summarized in Table 4.

Table 4: SVM Model Performance

<b>Metric</b>	<b>Value</b>
Accuracy (%)	22.00
F1 Score	22.00
MCC	0.07

The SVM model demonstrated an accuracy of 22.00%, an F1 score of 22.00%, and an MCC of 0.07. These metrics indicate that the SVM classifier, with the specified parameters, performed relatively poorly on the scaled-original dataset. Further tuning of hyperparameters or alternative kernel functions might be necessary to improve the performance.

### 6.4 K-Nearest Neighbors (KNN)

A K-Nearest Neighbors (KNN) classifier was evaluated using a grid search to find the optimal hyperparameters. The parameter grid included variations in the number of neighbors and the distance metric used. Grid search was performed with cross-validation to ensure robust model selection.

- **Parameter Grid:**

- `n_neighbors`: [3, 5, 7]
- `metric`: ['euclidean', 'manhattan']

- **Best Parameters Found:**

- `metric` = 'manhattan'
- `n_neighbors` = 7

The best model from the grid search was then evaluated on the test set. The results are summarized in Table 5.

Table 5: K-Nearest Neighbors (KNN) Model Performance

<b>Metric</b>	<b>Value</b>
Accuracy (%)	34.64
F1 Score	32.44
MCC	0.17

The K-Nearest Neighbors (KNN) model demonstrated an accuracy of 34.64%, an F1 score of 32.44%, and an MCC of 0.17. These metrics indicate that the KNN classifier performs moderately well on the scaled-original dataset. The hyperparameters tuned through grid search contributed to achieving these results.

## 6.5 Logistic Regression with Polynomial Features

Logistic Regression was evaluated using polynomial and interaction features to enhance the model's ability to capture complex relationships. A grid search was conducted to find the optimal hyperparameters, including the regularization strength and penalty type.

- **Polynomial Features:**

- Degree of polynomial: 2
- Interaction only: False
- Include bias: False

- **Parameter Grid:**

- `C`: [1, 10, 100]
- `penalty`: ['l2', 'l1', 'none']

- **Best Parameters Found:**

- `C` = 1
- `penalty` = 'l1'

The best model from the grid search was then evaluated on the test set. The results are summarized in Table 6.

Table 6: Logistic Regression Model Performance

<b>Metric</b>	<b>Value</b>
Accuracy (%)	27.47
F1 Score	27.47
MCC	0.01

The Logistic Regression model with polynomial features demonstrated an accuracy of 27.47%, an F1 score of 27.47%, and an MCC of 0.01. These metrics indicate that the logistic regression classifier with polynomial features struggled to perform well on the scaled-original dataset. Further tuning or alternative preprocessing strategies might be necessary to improve the performance.

## 6.6 CatBoost

The CatBoost classifier was evaluated using a grid search to find the optimal hyperparameters. The parameter grid included variations in learning rate, depth, L2 leaf regularization, iterations, loss function, and evaluation metric. Grid search was performed with cross-validation to ensure robust model selection.

- **Parameter Grid:**

- `learning_rate`: [0.01, 0.03]
- `depth`: [6, 10]
- `l2_leaf_reg`: [3, 5]
- `iterations`: [50, 100]
- `loss_function`: ['MultiClass']
- `eval_metric`: ['MultiClass']

- **Best Parameters Found:**

- `depth` = 10
- `eval_metric` = 'MultiClass'
- `iterations` = 100
- `l2_leaf_reg` = 3
- `learning_rate` = 0.03
- `loss_function` = 'MultiClass'

The best model from the grid search was then evaluated on the test set. The results are summarized in Table 7.

Table 7: CatBoost Model Performance

<b>Metric</b>	<b>Value</b>
Accuracy (%)	42.39
F1 Score	42.39
MCC	0.27

The CatBoost model demonstrated an accuracy of 42.39%, an F1 score of 42.39%, and an MCC of 0.27. These metrics indicate that the CatBoost classifier performs well on the scaled-original dataset. The hyperparameters tuned through grid search contributed to achieving these results.



## 6.7 Random Forest

A Random Forest classifier was evaluated using a grid search to find the optimal hyperparameters. The parameter grid included variations in the number of trees, maximum depth of the trees, minimum number of samples required to split an internal node, and the minimum number of samples required to be at a leaf node. Grid search was performed with cross-validation to ensure robust model selection.

- **Parameter Grid:**

- `n_estimators`: [100, 200]
- `max_depth`: [10, 20, None]
- `min_samples_split`: [2, 5]
- `min_samples_leaf`: [1, 2]

- **Best Parameters Found:**

- `max_depth` = 20
- `min_samples_leaf` = 2
- `min_samples_split` = 5
- `n_estimators` = 200

The best model from the grid search was then evaluated on the test set. The results are summarized in Table 8.

Table 8: Random Forest Model Performance

<b>Metric</b>	<b>Value</b>
Accuracy (%)	42.09
F1 Score	42.09
MCC	0.26

The Random Forest model demonstrated an accuracy of 42.09%, an F1 score of 42.09%, and an MCC of 0.26. These metrics indicate that the Random Forest classifier performs well on the scaled-original dataset. The hyperparameters tuned through grid search contributed to achieving these results.

## 7 Results and Discussion

This section presents the results of various models trained and tested on the scaled-original dataset. The models evaluated include LightGBM, XGBoost, Decision Trees, SVM, K-Nearest Neighbors, Logistic Regression, CatBoost, and Random Forest. The performance of these models is summarized in Table 9, followed by a detailed discussion of the results.

Table 9: Comparison of Model Performance

Model	Accuracy (%)	F1 Score (%)	MCC
LightGBM	<b>42.37</b>	39.01	<b>0.27</b>
XGBoost	42.31	38.69	<b>0.27</b>
Decision Trees	41.02	<b>41.02</b>	0.25
SVM	22.00	22.00	0.07
K-Nearest Neighbors	34.64	32.44	0.17
Logistic Regression	27.47	27.47	0.01
CatBoost	42.39	42.39	<b>0.27</b>
Random Forest	42.09	42.09	0.26

## 7.1 Discussion of Model Performance

The table above provides a comparative analysis of the performance metrics for each model. The following sections discuss the strengths and weaknesses of each model based on the observed results.

### 7.1.1 LightGBM

The LightGBM model achieved an accuracy of 42.37%, an F1 score of 39.01%, and an MCC of 0.27. These results indicate that LightGBM is highly effective in handling the dataset, likely due to its ability to handle large datasets and its gradient boosting framework, which efficiently manages the variance and bias trade-off.

### 7.1.2 XGBoost

XGBoost performed comparably to LightGBM, with an accuracy of 42.31%, an F1 score of 38.69%, and an MCC of 0.27. XGBoost’s performance is attributed to its robust boosting algorithm, which improves model accuracy by sequentially correcting the errors of previous models. However, its slightly lower F1 score suggests that it may not handle some imbalances as effectively as LightGBM.

### 7.1.3 Decision Trees

The Decision Tree model achieved an accuracy of 41.02%, an F1 score of 41.02%, and an MCC of 0.25. Decision Trees are simple and interpretable but often prone to overfitting. The high F1 score indicates good performance in terms of precision and recall, but the overall accuracy and MCC suggest that it might not generalize as well as the ensemble methods.

### 7.1.4 Support Vector Machine (SVM)

The SVM model performed poorly with an accuracy of 22.00%, an F1 score of 22.00%, and an MCC of 0.07. The linear kernel used in this model might not have been suitable for capturing the complex patterns in the dataset. Additionally, SVMs typically require significant tuning of hyperparameters and are sensitive to feature scaling, which might have contributed to its underperformance.

### 7.1.5 K-Nearest Neighbors (KNN)

The KNN model showed moderate performance with an accuracy of 34.64%, an F1 score of 32.44%, and an MCC of 0.17. KNN’s performance is heavily dependent on the choice of  $k$  and the distance metric. The results indicate that while KNN could capture some of the underlying data structures, it was less effective than other more sophisticated models.

### 7.1.6 Logistic Regression

The Logistic Regression model, even with polynomial features, struggled with an accuracy of 27.47%, an F1 score of 27.47%, and an MCC of 0.01. This suggests that the linear nature of Logistic Regression, even when augmented with polynomial features, was insufficient to model the complex relationships in the dataset.

### 7.1.7 CatBoost

CatBoost demonstrated strong performance with an accuracy of 42.39%, an F1 score of 42.39%, and an MCC of 0.27. CatBoost's success can be attributed to its ability to handle categorical features natively and its robust gradient boosting algorithm, which makes it highly effective for classification tasks on complex datasets.

### 7.1.8 Random Forest

The Random Forest model achieved an accuracy of 42.09%, an F1 score of 42.09%, and an MCC of 0.26. Random Forests are robust ensemble methods that reduce overfitting by averaging multiple decision trees. The results show that Random Forests are highly competitive with other boosting methods like LightGBM and CatBoost.

## 7.2 Conclusion

Among the models evaluated, CatBoost, LightGBM, and Random Forest emerged as the top performers, demonstrating high accuracy, F1 scores, and MCC. These models' ability to handle complex data patterns and reduce overfitting contributed to their success. Conversely, simpler models like SVM and Logistic Regression struggled with the dataset, indicating the need for more sophisticated approaches in this context. Future work could explore further hyperparameter tuning and the application of ensemble techniques to enhance model performance even further.

## 8 Further Improvements

While the current study provides a comprehensive analysis and comparison of various machine learning models, there are several avenues for further improvements that could enhance the performance and robustness of the models.

### 8.1 Testing Multiple Datasets Across All Models

In this study, four different datasets were prepared:

- Original Data: The original dataset without any oversampling or dimensionality reduction.
- SMOTE with Minimum Class Count of 20,000: Dataset where classes with fewer than 20,000 instances were oversampled to have at least 20,000 instances.
- SMOTE with Maximum Class Count: Dataset where each class was oversampled to match the size of the largest class.
- PCA Applied on Original Data: The original dataset with PCA applied to retain 95% of the variance.

Currently, only the LightGBM model was tested across all these datasets due to computational limitations. To achieve a more comprehensive evaluation, it would be beneficial to test every dataset with each of the models (LightGBM, XGBoost, Decision Trees, SVM, KNN, Logistic Regression, CatBoost, and Random Forest). This extensive testing could uncover specific dataset-model combinations that perform better and provide deeper insights into the strengths and weaknesses of each approach.

## 8.2 Avoiding Data Leakage

Ensuring that there is no data leakage is critical for building robust predictive models. In future work, collaborating with domain experts could help identify and create new features while avoiding data leakage. These new features might capture more relevant information and improve the model's ability to predict patient stay durations.

## 8.3 Hyperparameter Tuning

Although grid search was employed for hyperparameter tuning in this study, more advanced techniques such as randomized search or Bayesian optimization could be explored to find even better hyperparameter configurations. These methods can be more efficient and effective in navigating large hyperparameter spaces.

## 8.4 Computational Resources

Future studies could benefit from access to more computational resources. Enhanced computational power would allow for extensive experimentation with various datasets and models, leading to more comprehensive and conclusive results.

## 8.5 Conclusion

The current study provides a solid foundation for predicting patient stay durations using machine learning models. However, the suggested improvements highlight the potential for further enhancement. By testing all datasets with each model, ensuring data integrity, exploring advanced hyperparameter tuning methods, employing ensemble techniques, and collaborating with domain experts, future work can build upon this study to achieve even better performance and insights.

# 9 Notes

This dataset is sourced from a Kaggle competition. The original Kaggle competition can be found at [AV: Healthcare Analytics II](#), and the hackathon link is available at [Analytics Vidhya: JanataHack Healthcare Analytics II](#).

Upon reviewing submissions on Kaggle and discussions on AnalyticsVidhya.com, it is observed that realistic approaches achieve accuracy in the range of 42-43%. Some approaches have reported metrics as high as 70-71% by applying SMOTE before train-test splitting, which introduces data leakage and allows the model to see values it should not, leading to inflated performance metrics. This practice is not considered a correct modeling approach.

Discussions on AnalyticsVidhya.com indicate that it is unclear whether the data is synthetic or based on real-world data, with many participants suggesting that it is synthetic. This belief contributes to the accuracy plateauing around 40-43%. Due to the synthetic nature of the data, I opted not to apply undersampling methods, as it would result in data loss.

My attempts at creating new features to improve performance without causing data leakage did not yield significant improvements and thus were not included in this study. The study focused on maintaining data integrity and realistic modeling practices to ensure the validity of the results.

## 10 References

1. [LightGBM](#)
2. [XGBoost](#)
3. [10 Techniques to Deal with Class Imbalance in Machine Learning](#)
4. [Principal Component Analysis \(PCA\)](#)
5. [CatBoost ML](#)
6. [Step-by-Step Exploratory Data Analysis \(EDA\) Using Python](#)
7. [One-Hot Encoding vs Label Encoding Using Scikit-Learn](#)