Final Report

Optimizing Machine Learning Models for Diabetic Data Analysis

[Block Y1B-2024-25]

Introduction

Healthcare systems face constant challenges in improving patient care and managing resources effectively. This report explores how machine learning can help solve some of these challenges by analyzing patient data. It focuses on three main tasks: predicting how long a patient will stay in the hospital, identifying patients at risk of being readmitted, and grouping patients with similar characteristics. Using various machine learning models, feature engineering, and optimization techniques, this study aims to provide insights and build tools that can support better decision-making in healthcare.

Problem Statement

Hospitals need to balance providing high-quality care with managing their resources efficiently. Predicting hospital stay duration, identifying patients at risk of readmission, and finding patterns in patient data are crucial to meeting these goals. This project uses machine learning to tackle these challenges by creating predictive models and finding meaningful groupings in the data, which can help hospitals make smarter, data-driven decisions.

1. Exploratory Data Analysis (EDA) with Python and SQL

The first step in this study was to become familiar with the dataset through Exploratory Data Analysis (EDA). This involved gaining a preliminary understanding of the data's structure, contents, and potential issues. Initially, I conducted a brief overview of the data using SQL. Following this, I carried out a more in-depth EDA using Python, leveraging visualization libraries and statistical tools to uncover patterns, relationships, and trends in the data.

1A. Exploratory Data Analysis with SQL

1A.1 General Overview of the Data

To initiate the analysis, I conducted an exploration to understand the dataset's structure and key attributes.

Using the *COUNT* function on the *encounter* table, I identified a total of **101,766 records** in the dataset. This is a sufficiently large sample size for machine learning applications but raises the need to address potential data quality issues.

By joining the *admission_type* and *encounter* tables, I analyzed the distribution of admission types. The most frequent categories were:

- **Emergency**
- **Elective**
- **Urgent**

Additionally, several records had unknown or unclassified admission types such as *NULL*, *Not Available*, and *Not Mapped*. Combining these categories into a unified "Unknown" group could enhance clarity for subsequent analysis.

To further understand patient outcomes, I examined the most common discharge dispositions:

- **Discharged to home:** 60,234 cases.
- **Discharged/transferred to SNF (Skilled Nursing Facility):** 13,954 cases.
- **Discharged/transferred to home with home health service:** 12,902 cases.

Interestingly, the fourth most frequent discharge category was unknown (*NULL*), comprising **3,691 records.**

1A.2 Identifying Missing or Anomalous Data

During my analysis of the *race* column, I found two distinct entries representing missing data:

- **?:** 1,948 records.
- **Other:** 1,178 records.

In the *weight* column, **68,665 records** had unknown values, a significant proportion of the dataset. This suggests that weight was either inconsistently recorded or frequently omitted. Given its sparse coverage, I recommend removing the *weight* feature from further analysis.

1A.3 Understanding Age Distribution

To explore the age distribution, I analyzed the *age* column while excluding unknown values (*?*). The age groups are divided into 10-year intervals (e.g., 0–9, 10–19, 20–29, etc.).

The majority of records belonged to patients aged 50–89, with few records for individuals aged 29 and below. This distribution suggests that the dataset primarily focuses on middle-aged and elderly populations.

1A.4 Admission Trends by Source and Type

I analyzed the contribution of different admission sources using the *admission_source* table. The most frequent sources were:

- **Emergency Room:** 57,494 records
- **Physician Referral:** 29,565 records
- **Unknown:** 6,781 records

Some sources had minimal contributions, such as:

- **Court/Law Enforcement:** 16 records
- **Transfer from Hospital:** 12 records
- **Normal Delivery:** 2 records

Next, I joined the *encounter*, *admission_source*, and *admission_type* tables to analyze which admission types correspond to specific sources. The most common pairs were:

- **Emergency Room – Emergency**
- **Physician Referral – Elective**
- **Physician Referral – Urgent**

Several less frequent pairs (e.g., *Transfer from CAH – Elective*) had fewer than **500 records**, **some even as low as 1 record only**.

1A.5 Hospital Stay and Readmission Patterns

The average hospital stay varied by admission type:

- **Trauma Center:** 4.86 days
- **Urgent:** 4.61 days
- **Emergency:** 4.38 days
- **Elective:** 4.32 days
- **Newborn:** 3.20 days

Three readmission categories are present in the dataset:

- **NO:** No readmission
- **<30:** Readmission within 30 days
- **>30:** Readmission after 30 days

Emergency admissions were the most frequent for both **non-readmissions (28,460 records)** and **readmissions >30 days (19,309 records)**. This indicates a high rate of readmissions among emergency cases.

1A.6 Comparing Admission Types and Outcomes

To compare discharge dispositions across admission types, I joined the *encounter*, *admission_type*, and *discharge_disposition* tables, grouping by type and disposition descriptions while excluding *NULL* values.

Similarly, to analyze the relationship between readmissions and discharge dispositions, I joined the *encounter* and *discharge_disposition* tables. Excluding *NULL* values for discharge dispositions and *NO* values for readmissions allowed me to focus on cases with actual readmissions.

The results revealed patterns linking specific admission types to discharge outcomes and readmission frequencies, providing insights into the quality of care and patient trajectories.

1B. Exploratory Data Analysis with Python

To gain a deeper understanding of the dataset and its features, I used Python's visualization libraries to explore trends, distributions, and anomalies. Each visualization served a specific purpose in addressing a research question or hypothesis.

1B.1 Loading and Analyzing the Dataset Shape

To begin the Python-based analysis, I loaded the dataset using the *pandas* library's *read_csv* function. Upon examining its shape, I identified that the dataset consists of **101,766 records** and **50 features**.

Reviewing the feature names revealed that a significant number are related to medications. This indicates that the dataset provides detailed pharmaceutical information, which could be leveraged for understanding medication usage patterns among patients.

1B.2 Load and Explore a Dataset Using NumPy

Continuing with the Python-based analysis, I performed the same steps as in the subtask 1B.2 using the package NumPy to further confirm my findings. Looking at the dataset's shape, I confirmed that the data contains 101, 766 records and 50 features.

1B.3 Analyzing Data Types

I assessed the data types of all features to understand their structure. Most features were categorized as objects, indicating they are categorical. This step is critical for preparing the data, as many machine learning algorithms require numerical encoding for categorical variables.

To gain further insights, I analyzed the unique values for each feature, looking at the distributions and frequencies of their entries. This helped identify features with sparse data, possible missing values, or unusual entries requiring further exploration.

1B.4 Exploratory Data Analysis (EDA) with Visualizations

To explore trends, distributions, and potential anomalies, I made use of Python visualization libraries Matplotlib and Seaborn. Each visualization addressed specific questions about the dataset's characteristics.

Outlier Detection in Numerical Features

I created boxplots for all numerical features to identify outliers. These visualizations showed that many columns, such as *num_outpatient* and *num_inpatient*, contained a significant proportion of outliers.

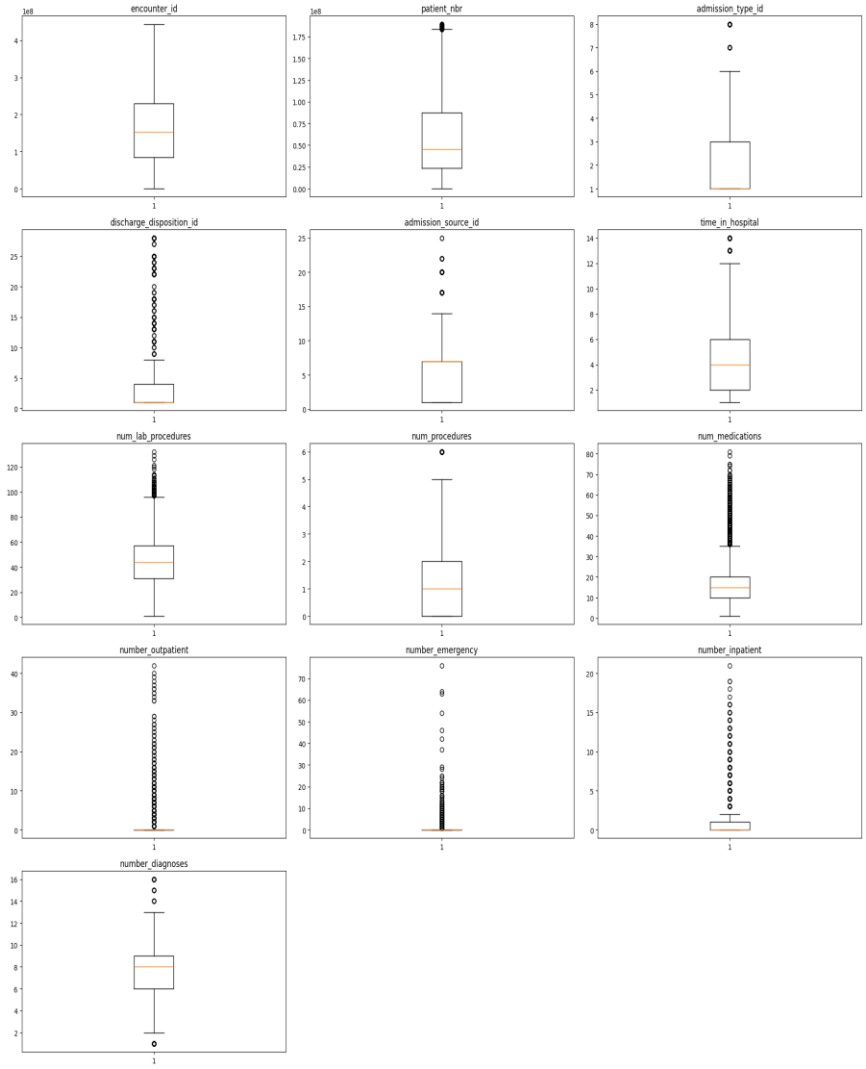

Image 1 Boxplot visualizations of numerical features.

Detecting these anomalies is crucial, as they may affect the reliability of statistical analysis and machine learning outcomes.

Age Distribution

Using a bar chart, I visualized the age distribution of patients. The dataset primarily represents middle-aged and elderly individuals, with most records falling in the 50–90 age range.

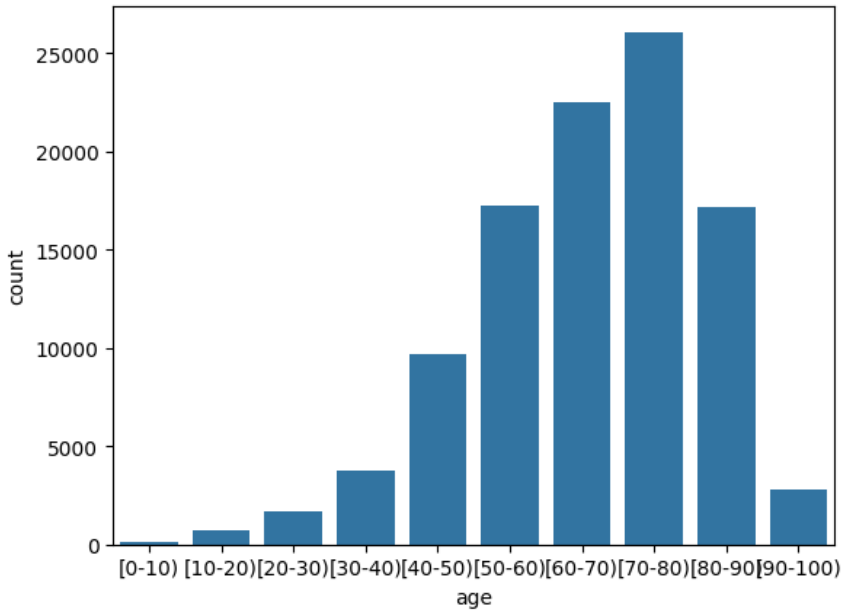

Image 2 Bar chart of age distribution.

Gender Distribution

A countplot of gender distribution indicated a relatively equal representation of male and female patients. This balance minimizes the risk of gender bias in subsequent analyses related to patient outcomes or treatment patterns.

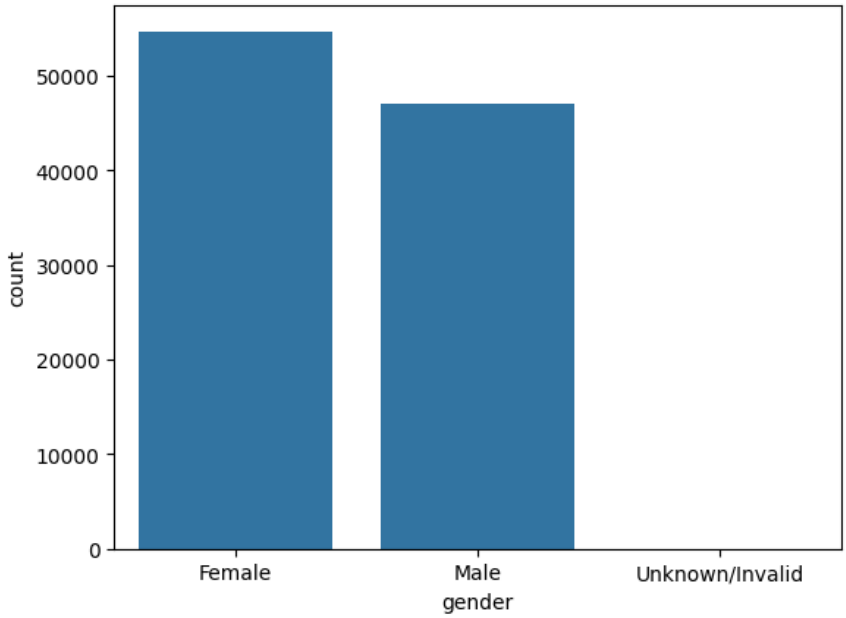

Image 3 Count plot of gender distribution.

Length of Stay in Hospitals

I used a histogram to examine the distribution of hospital stays. The visualization revealed a right-skewed pattern, with most stays lasting between **1–6 days**. Understanding the typical length of stay is crucial for analyzing resource utilization and patient care efficiency.

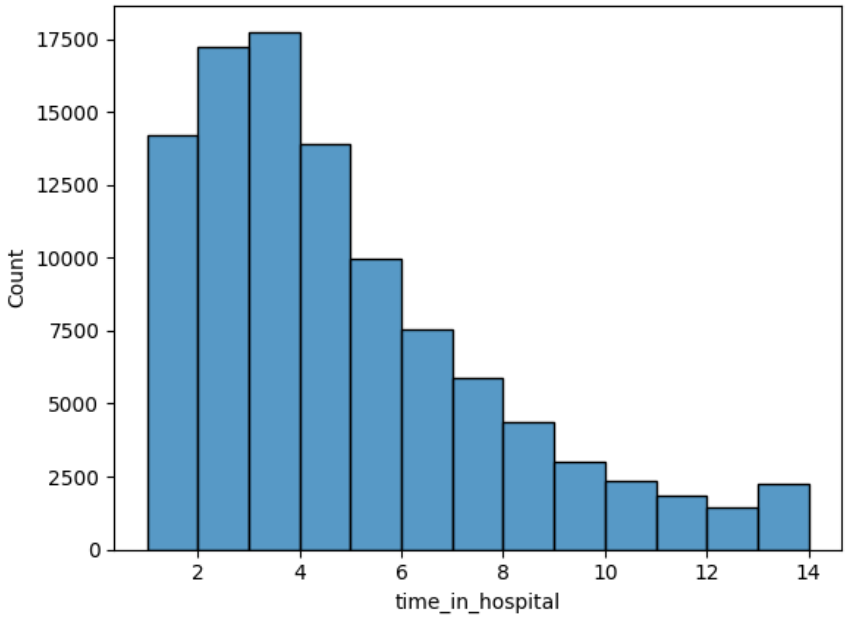

Image 4 Histogram of time in hospital.

Laboratory Procedures

A boxplot of the number of laboratory procedures (*num_lab_procedures*) highlighted the presence of extreme outliers, with some patients undergoing over 100 procedures. These cases may indicate specialized or intensive care needs and warrant further investigation.

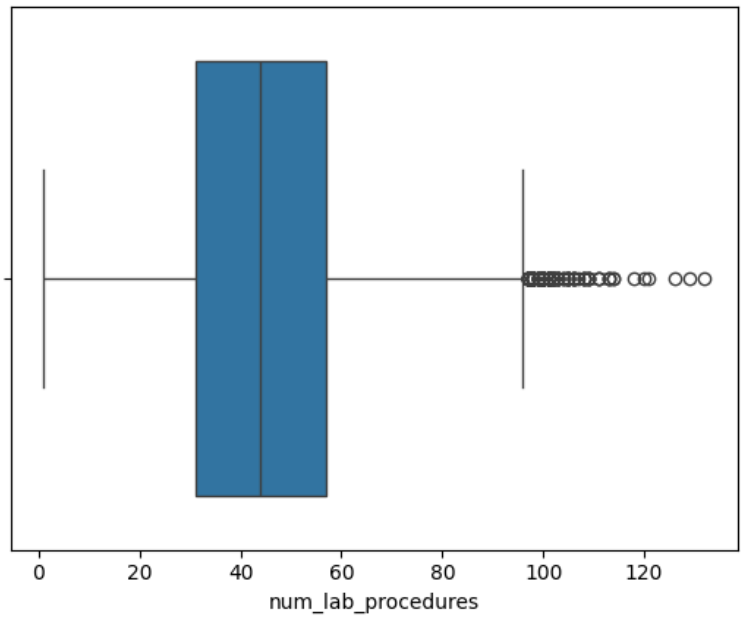

Image 5 Boxplot of laboratory procedures.

Medication Features

Bar charts displaying medication value counts showed that some features exclusively contained the value **No**, while others included additional values such as **Up**, **Steady**, and **Down**. To simplify future analysis, I propose encoding these features into binary categories (**No** and **Yes**), consolidating values like Steady and Down under Yes.

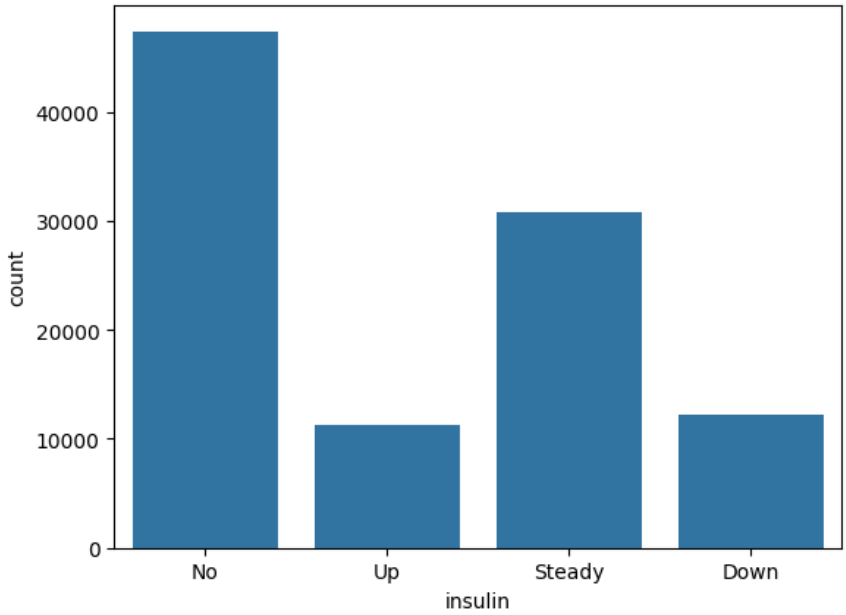

Image 6 Bar chart of insulin feature.

Readmission Status

A pie chart of readmission status distribution revealed that over 50% of patients were not readmitted (*No*). Among the remaining records, more patients were readmitted after 30 days (*>30*) than within 30 days (*<30*). This insight highlights potential areas for improving post-discharge care.

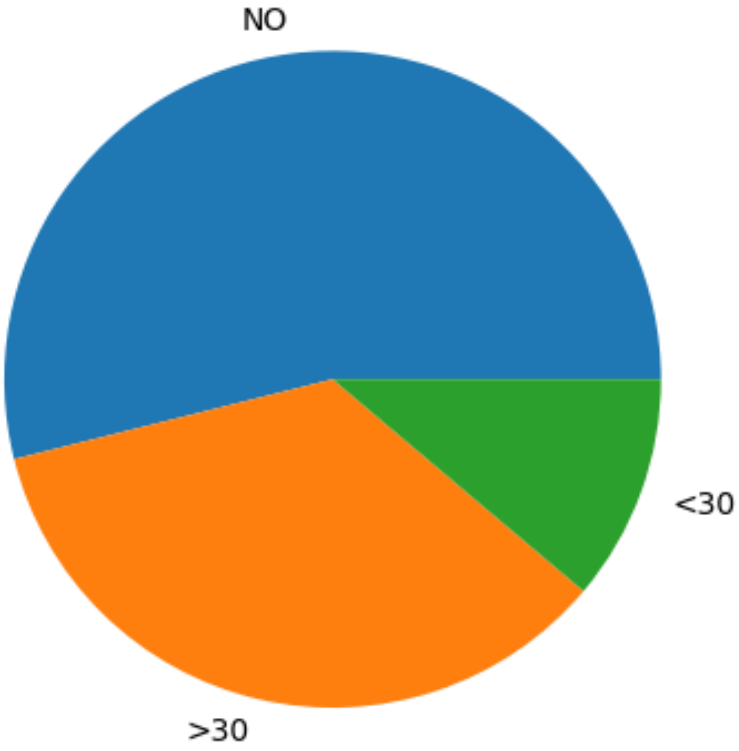

Image 7 Pie chart of readmission status distribution.

Patient Race

To analyze racial demographics, I created a bar chart showing patient counts by race. The majority of records belong to Caucasians, followed by African Americans. A relatively small proportion of entries were categorized as unknown or other races.

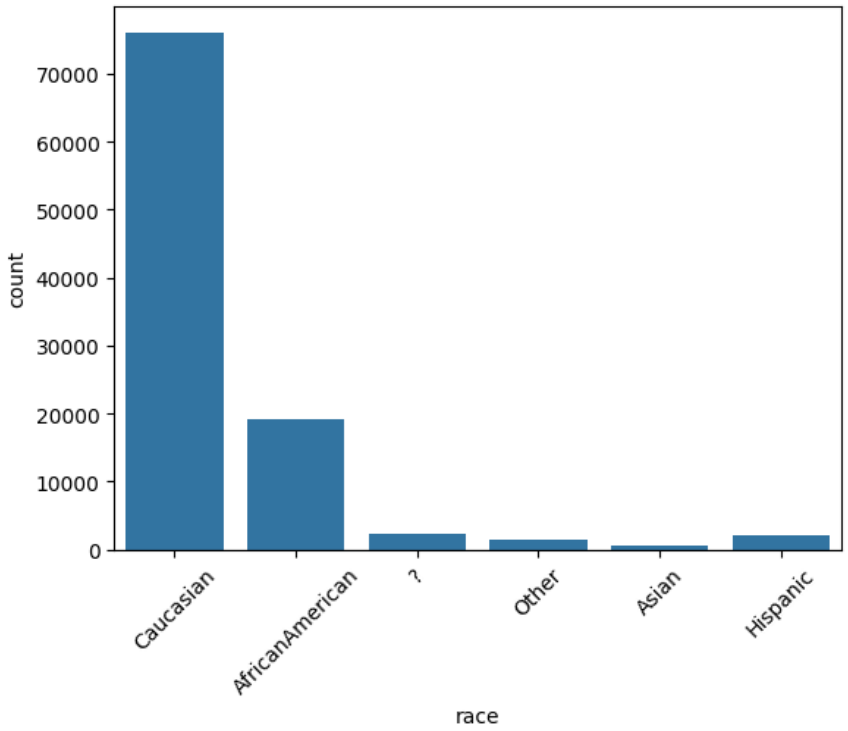

Image 8 Bar chart of patient counts by race.

Admission Sources

Next, I created a stacked bar chart to explore the relationship between race and admission source, scaling the y-axis logarithmically for clarity. Notably, the Hispanic race exhibited the highest proportion of missing values (id *17*) in admission sources, highlighting potential gaps in data collection for this demographic.

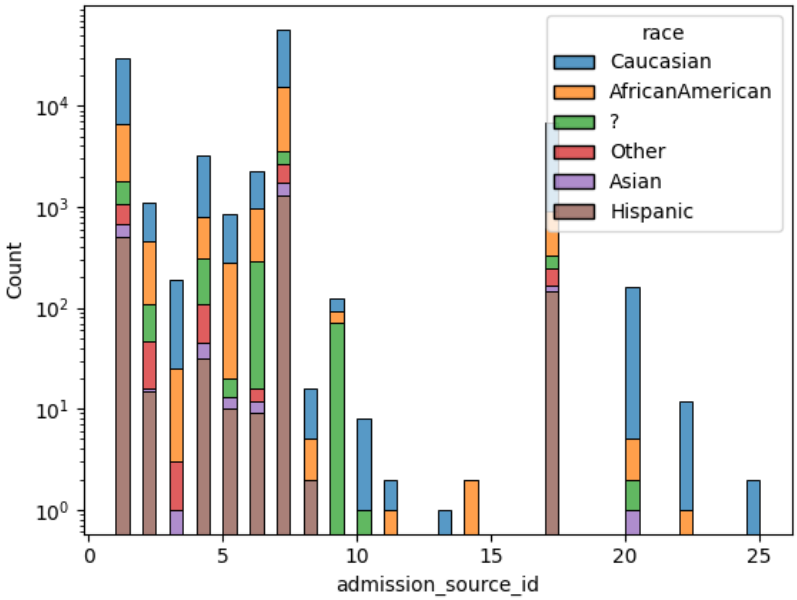

Image 9 Stacked bar chart of race and admission source.

Medical Specialty Distribution

A bar chart of the top 10 most frequent medical specialties revealed that *?* (unknown specialty) accounts for a significant proportion of the data. The next most common specialties were *Internal Medicine, Emergency/Trauma,* and *Family/General Practice*, each with approximately **10,000 records**. The high number of unknown values suggests a critical data quality issue.

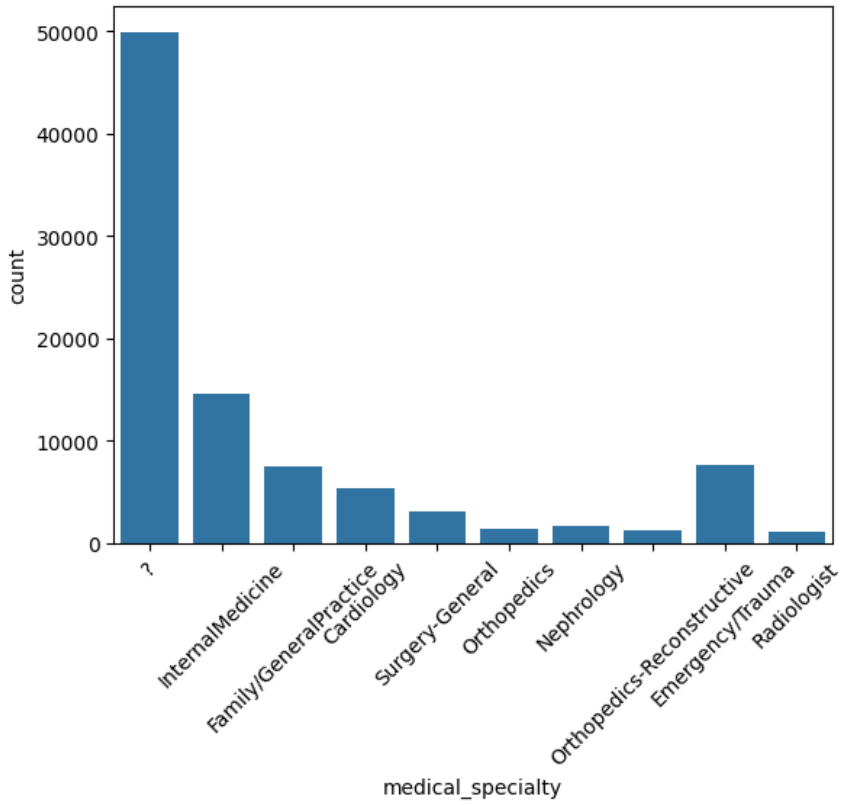

Image 10 Bar chart of the top 10 most frequent medical specialties.

To further investigate, I created a grouped bar chart comparing readmission statuses across these top specialties. This revealed that a large proportion of readmissions, both *<30* and *>30*, were associated with unknown specialties. This underscores the importance of addressing missing data in this feature to improve the dataset's usability.

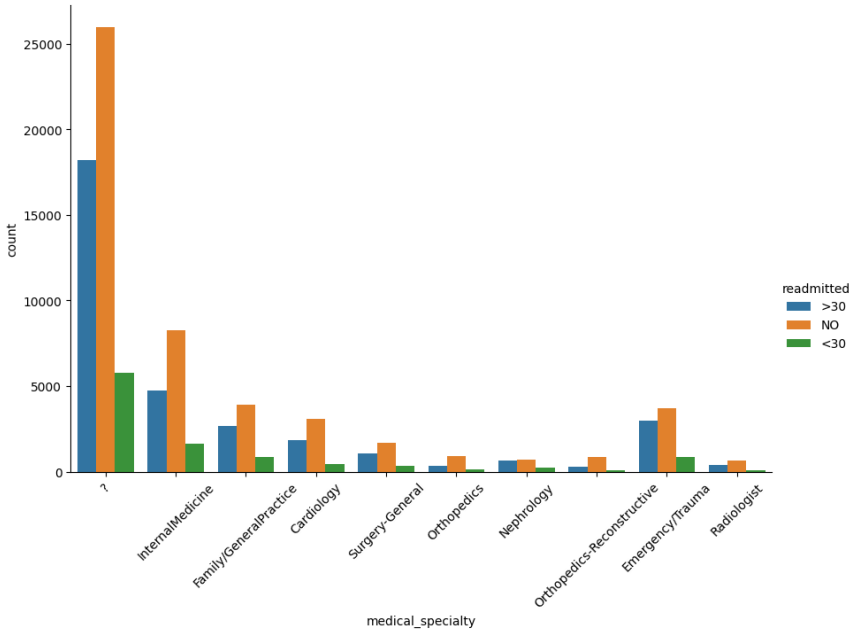

Image 11 Grouped bar chart comparing readmission statuses and medical specialties.

1B.5 Identify Metadata from the Dataset

To further examine the main dataset, I analyzed the additional file, *IPS_mapping.csv*, which contains metadata mappings. Upon inspecting the raw content of this file, I observed that it did not follow the standard CSV formatting, as it included multiple tables within a single file.

To address this, I developed a helper function, *split_csv_into_tables*, which reads the file and extracts each table as a separate DataFrame. This approach allows for more efficient handling of the non-standard file format.

Upon extracting the data, I identified three tables:

- **Admission Type**
- **Discharge Disposition**
- **Admission Source**

Examining these tables, I gained insight into which descriptions correspond to which IDs in the main dataset. This information enabled me to interpret the categorical IDs more effectively, providing valuable context.

A key insight from this analysis was the presence of *NULL* descriptions in some mappings, indicating unknown or missing data. Understanding these metadata mappings was critical for improving the clarity and interpretability of the main dataset.

2. Data Processing

Preparing the dataset is a critical step before applying machine learning models. The way data is processed can greatly influence how well the models perform. In this section, I explain how the dataset was cleaned and prepared, including steps like handling missing data, simplifying categories, addressing outliers, and selecting relevant features.

2.1 Dataset and Features Inspection

To begin, I inspected the dataset in detail to gather insights and establish a clear understanding of its structure. This preparation ensured that I could apply appropriate transformations during the preprocessing phase.

2.1.1 Load and Check for Incorrect Types

I loaded the dataset into a pandas DataFrame named *diabetic_data*, which served as the main working variable throughout the project. Inspecting the first few rows provided an overview of the data and revealed patterns that needed attention. Using the *info* method, I reviewed the structure and data types of all features. This step was critical for identifying issues, such as numerical features incorrectly stored as strings. For convenience, I also created a reference list of column names which I could refer to in the further steps of the workflow.

2.2 Preprocessing the Dataset

Once the dataset's structure was clear, I started cleaning and transforming the data to make it suitable for machine learning.

2.2.1 Identify Missing Values

Handling missing data was one of the first challenges. Using the *missingno* module, I visualized the missing values, which highlighted gaps in several features. To simplify processing, I standardized the representation of missing entries by replacing values like *?*, *Unknown/Invalid*, and *Other* with *np.nan* value.

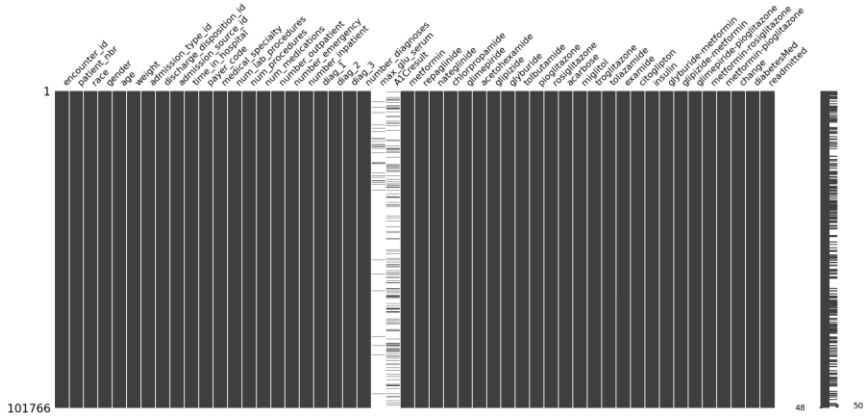

Image 12 Visualization of missing values.

Next, I addressed incomplete data. Rows with missing values in **less than 5%** of the dataset were removed, reducing the dataset to **96,569 records**. For features with excessive missing

data (**more than 50%**), such as *A1Cresult*, *max_glu_serum*, and *weight*, I dropped the columns entirely, as they offered little value. I filled the remaining missing values in **numerical features using the median**, while **categorical features were imputed with their mode**. This approach preserved the dataset's integrity without introducing unnecessary noise.

2.2.2 Handle Irrelevant or Redundant Data

Some features in the dataset were irrelevant or offered little value for analysis. For example, the *patient_nbr* column had 67,507 unique values, making it unlikely to contribute meaningful insights. Similarly, *medical_specialty* contained 49% unknown values, and *payer_code* was unrelated to the goals of the project. Removing these columns simplified the dataset and **improved its focus on relevant information**.

2.2.3 Standardize, Transform, and Encode Data

Since machine learning models typically require numerical inputs, I converted all categorical features into numerical formats. For the *medications* columns, such as metformin and repaglinide, I simplified the entries by treating *Steady*, *Up*, and *Down* as *Yes*, while *No* remained unchanged. These features were then encoded using **ordinal encoding**.

The *readmitted* column, which indicated whether a patient was readmitted within 30 days, was converted to a binary format: **0 for no readmission** and **1 for any readmission**. Similarly, the *age* column, which originally contained ranges (e.g., *[20–30)*), was **mapped to numerical values** for consistency. Other features, like *diabetesMed* and *change*, were encoded **ordinally**, while the rest of the categorical features were **one-hot encoded**. This process ensured the dataset was **entirely numerical**, making it compatible with machine learning algorithms.

2.2.4 Handle Outliers

Outliers can skew analyses and reduce model accuracy. I used boxplots to reveal the presence of extreme values in several features. To address this, **I removed the top and bottom 15% of records** for all of the features. After this adjustment, the distributions appeared more balanced, as confirmed by revisiting the boxplots.

2.2.5 Feature Selection

To identify the most relevant features, I used a **correlation matrix** and focused on **features with correlations above 25% or below -25%**. This analysis highlighted **24 features with stronger relationships**, which will play a central role in the modeling phase. Selecting relevant features helps simplify models, reduces computational complexity, and helps prevent overfitting.

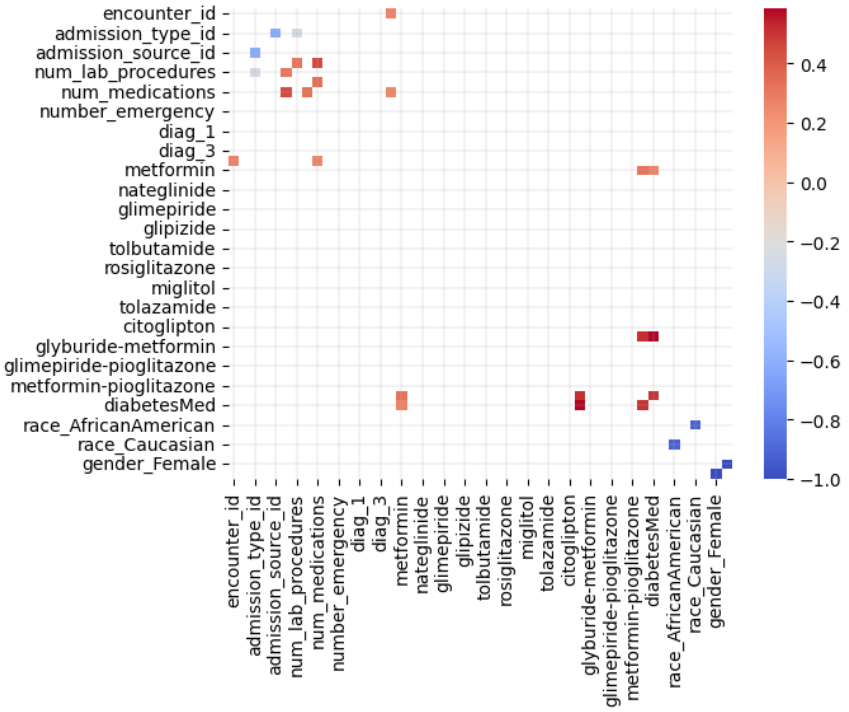

Image 13 Correlation matrix visualization.

Summary

Starting with 101,766 records and 50 features, the preprocessing steps **reduced the dataset to 96,569 records** and **48 features**. These adjustments resulted in a cleaner, more reliable dataset, ready for machine learning implementation. By addressing missing values, standardizing formats, and selecting meaningful features, I ensured the data accurately reflects underlying patterns and is optimized for modeling.

3. Machine Learning

In this part I implemented various machine learning models for regression, classification, and clustering, together with analyzing their performances and subsequent evaluation.

3.1 Implementing a Regression Model

In this part of the task, I implemented various regression models to **predict numerical output**. The goal was **to predict how long a patient will stay in the hospital**, which is represented by the *time_in_hospital* feature. To evaluate the performance of these models, I used two common metrics for regression tasks:

- **MAE (Mean Absolute Error):** The average of the absolute differences between predictions and actual values. This metric represents how much on average my predictions are off.
- **MSE (Mean Squared Error):** The average of the squared differences between predictions and actual values. This metric emphasizes larger errors more because of squaring.

3.1.1 Split the data

I began by splitting the dataset, allocating **20% of the data for testing purposes** and using the remaining **80% for training**. This split ensures that the model is evaluated on a separate, unseen dataset, allowing for a more realistic measure of its performance.

3.1.2 Implement a Linear Regression Model

The first model I implemented was the *LinearRegression* model. Linear regression is an approach that **finds the best-fitting straight line to predict the target variable**. The model works by establishing a relationship between the features and the target variable, aiming to minimize the error between the predicted and actual values.

For predicting *time_in_hospital*, the **MAE came out to be 1.46** and the **MSE was 3.21**. This suggests that, on average, the model's predictions were **off by about 1.46 days**. The MSE of 3.21 indicates that while most predictions are fairly close, **there are some larger errors** that impact the overall performance.

3.1.3 Visualise the Output for Linear Regression

To better understand how well the Linear Regression model performed, I created **a scatter plot comparing the actual target values** (x-axis) **against the predicted values** (y-axis). The closer the points are to the red dashed line (which represents perfect predictions), the better the model is performing. The plot revealed that, while the points were generally close to the red dashed line, they were stacked vertically in some places, **indicating that the model may**

be making similar predictions for certain input features. This could be due to either poor data scaling or the model failing to capture the complexity of some relationships in the data.

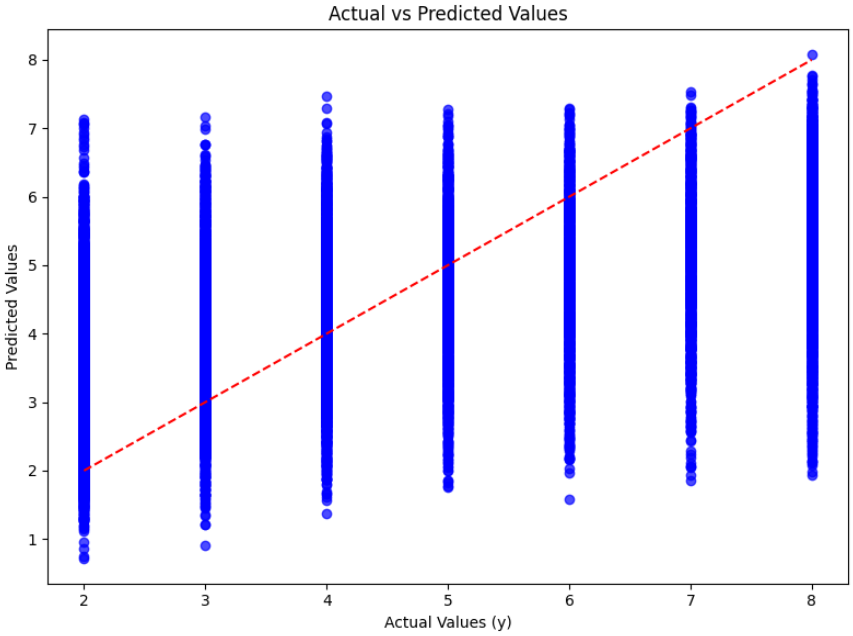

Image 14 Scatter plot comparing the actual target values against the predicted values.

Additionally, I created **a residual plot to examine the errors of the model**. A good regression model should have residuals that are randomly scattered around zero, with no apparent pattern. However, the residuals in this case **showed a clear pattern**, suggesting that **the model might not be the best fit for the data** and that the assumption of constant variance was violated.

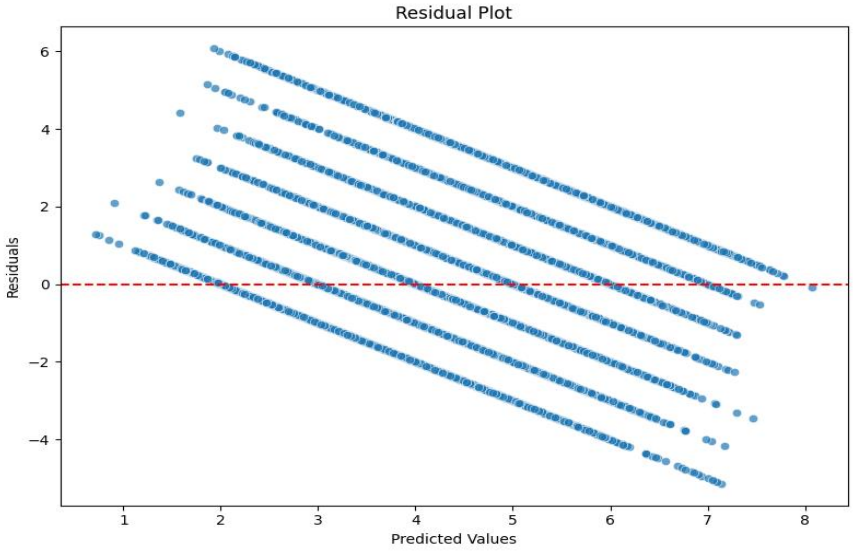

Image 15 Plot of residuals.

3.1.4 Implement Decision Tree for Non-linear Relationships

Next, I used the *DecisionTreeRegressor* model. A decision tree makes predictions by **recursively splitting the data based on the feature values**, creating a tree-like structure. Each branch in the tree represents a decision rule, and each leaf represents a predicted value for the target variable.

For predicting *time_in_hospital*, the **MAE came out to be 1.69**, and the **MSE was 5.53**. The higher MSE compared to the Linear Regression model indicates that the decision tree model was **more sensitive to larger errors**, likely due to overfitting, which is common in decision trees.

3.1.5 Implement Gradient Boosting Regression

The next model I tried was the *GradientBoostingRegressor*. This model builds a series of decision trees, with **each new tree attempting to correct the errors made by the previous ones**. Over time, this method boosts performance by focusing more on the data points where previous models made mistakes.

The **MAE for this model was 1.35**, and the **MSE was 2.82**. The lower MAE and MSE compared to both Linear Regression and Decision Tree models suggest that Gradient Boosting Regression performed better at predicting *time_in_hospital*, with **fewer large errors and better overall accuracy**.

3.1.6 Implement XGBoost Regression

Finally, I implemented *XGBRegressor*, an advanced version of gradient boosting. XGBoost **uses optimizations to improve the performance and efficiency** of traditional gradient boosting. By adjusting the learning process, it is designed to achieve better generalization to unseen data.

The **MAE came out to be 1.27**, and the **MSE was 2.64**. Compared to the other models, XGBoost **showed the best performance**, with the lowest MAE and MSE.

3.1.7 Visualise the Output for Non-linear Regression

To further understand the contribution of different features, I visualized **the feature importance for the XGBoost regression model**. This plot showed that the most important features for predicting *time_in_hospital* were *num_medications*, *discharge_disposition_id*, *num_lab_procedures*, *num_procedures*, and *diag_1*. All of these features had an importance score **greater than 0.05**, indicating their strong influence on the model's predictions.

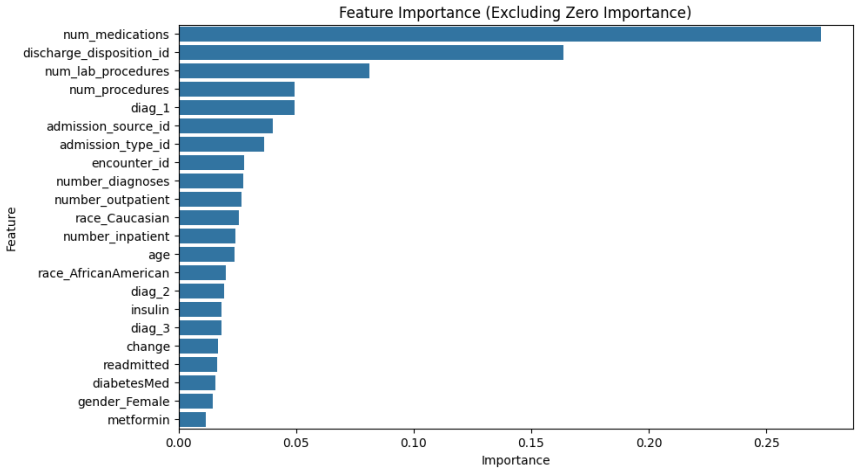

Image 16 Plot of XGBoost's feature importances.

3.1.8 Evaluation and Model Comparison

When comparing all the models, **XGBoost performed the best**, with the lowest MAE and MSE, followed by Gradient Boosting, Linear Regression, and Decision Tree in that order. The performance of XGBoost suggests that **more complex models with tree-based methods can better capture the patterns in the data**, especially when there are non-linear relationships between the features and the target variable. This is important for predicting hospital stay durations accurately, which can help hospitals plan their resources, like beds and staff, more effectively.

While Linear Regression was the simplest and provided decent results, it did not handle the complexities of the data as effectively as the other models. The Decision Tree, although intuitive, was impacted by large errors. Gradient Boosting was an improvement, but XGBoost showed the highest accuracy, likely due to its more advanced optimization techniques.

In conclusion, **XGBoost would be the best choice for this regression task**, as it balances accuracy, efficiency, and reliability. It handles non-linear relationships and overfitting better than simpler models, making it the most robust model for predicting *time_in_hospital*.

3.2 Implementing Classification Baseline

In this section, I focused on implementing various classification models **to predict a categorical feature**. For evaluation, I used four commonly used metrics in classification tasks

- **Accuracy:** The percentage of predictions that are correct.
- **Precision:** How many of the predicted positives are actually correct.
- **Recall:** How many of the actual positives were correctly identified.
- **F1 Score:** Combines precision and recall into a single metric.

I also used *correlation_matrix*, which helps to visualize the number of **true positives (TP)**, **true negatives (TN)**, **false positives (FP)**, and **false negatives (FN)**. This matrix helps me understand where the model is making errors.

3.2.1 Prepare Data for Classification

I began by preparing the data, focusing on predicting the *readmitted* feature. I used the dataset from Task 2 (stored in the variable *diabetic_data*) and split it into **training (80%)** and **testing (20%)** sets.

3.2.2 Implement Logistic Regression Model

The first model I tried was *LogisticRegression*, a standard model for **binary classification** tasks which I already used in Task 3.1.2. Initially, **the results were as follows**:

- **Accuracy:** 0.51
- **Precision:** 0.48
- **Recall:** 0.57
- **F1 Score:** 0.52

I used the ROC curve to visualize the output of the model. **ROC curve is a graph that shows how well a classification model distinguishes between two classes.** The plot contains TPR on the y-axis and FPR on the x-axis. It shows the trade-off between sensitivity and specificity at different thresholds of classification.

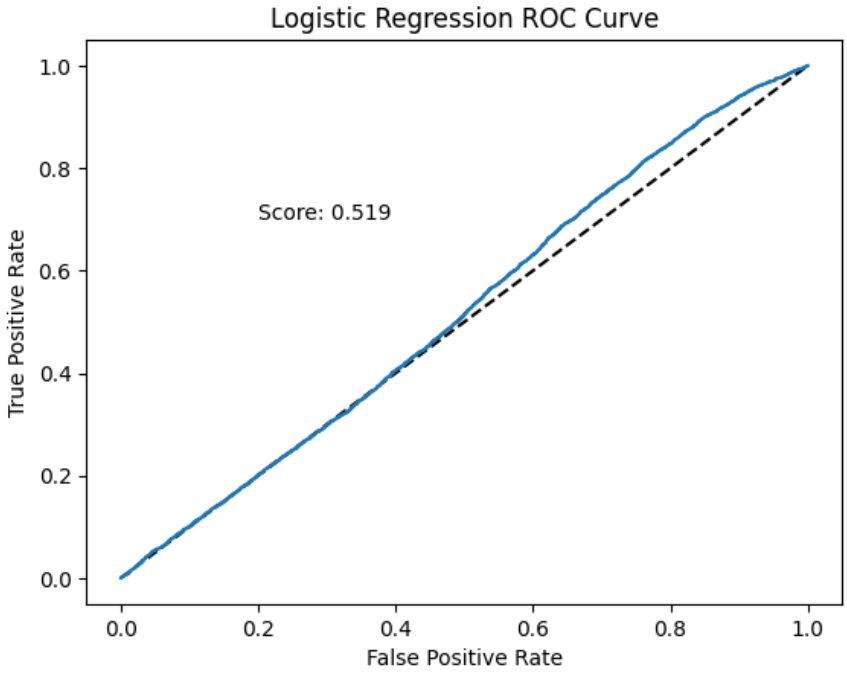

Image 17 ROC curve visualization.

This ROC curve suggests that the Logistic Regression **model is performing poorly**. The ROC curve is almost identical to the dashed diagonal line, which represents random guessing. An **AUC score of 0.519** (close to 0.5) confirms that the model's predictive power is no better than a random classifier. A good classifier would have an AUC much closer to 1.0.

After this, I decided to scale the data for this particular model. I used *StandardScaler*, which transforms the data so that each feature has a **mean of 0** and a **standard deviation of 1**. After scaling the data, the results improved:

- **Accuracy:** 0.62

- **Precision:** 0.60
- **Recall:** 0.56
- **F1 Score:** 0.58

THE ROC curve has **moved further from the diagonal line**, and the **AUC score is 0.655**. The score indicates that the model is doing better than random guessing. The curve moving further away from the diagonal line shows that the model has some ability to distinguish between the positive and negative classes.

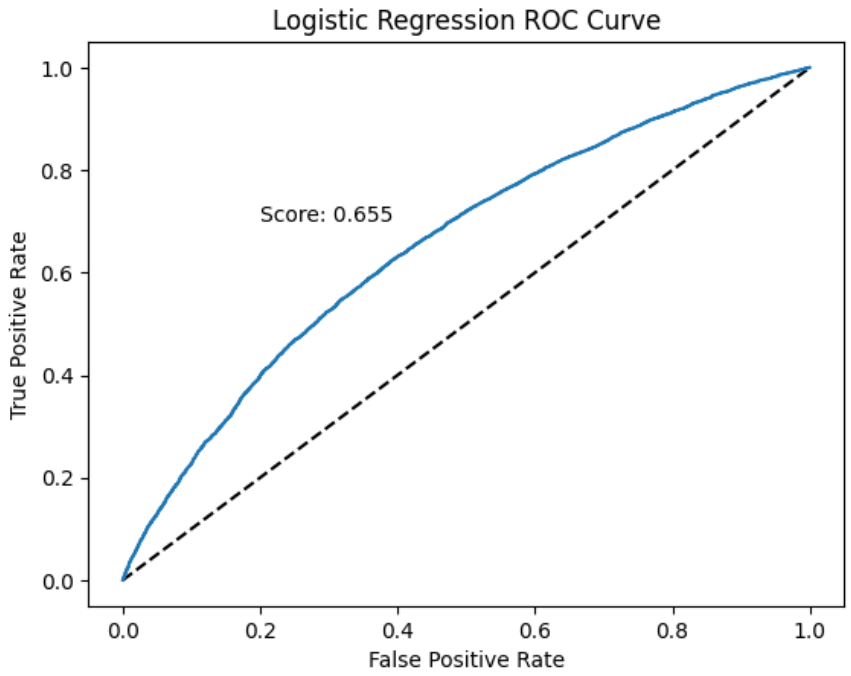

Image 18 ROC curve visualization after data scaling.

While the model has some predictive capability, it **isn't very reliable** yet and there's room for improvement. A robust classifier would typically achieve an AUC closer to 0.8 or higher.

3.2.3 Implement Random Forest Classifier as a Non-linear Baseline

Next I implemented *RandomForestClassifier*, a machine learning model that **uses an ensemble of decision trees to classify data**. It creates many decision trees that are trained on random subsets of the data and features. Next, it gathers predictions from all the trees and combines them to produce the final class prediction.

The performance of the Random Forest Classifier was:

- **Accuracy:** 0.63
- **Precision:** 0.62
- **Recall:** 0.52
- **F1 Score:** 0.57

Compared to the Logistic Regression model, Random Forest showed improvements in accuracy and precision. However, recall decreased slightly. Overall, **Random Forest provides a**

more reliable model for classification tasks, particularly when dealing with non-linear relationships in the data.

3.2.4 Implement K-Nearest Neighbors (KNN) Classifier

Lastly, I implemented the K-Nearest Neighbors (KNN) algorithm. *KNeighborsClassifier* **classifies data points based on the classes of their nearest neighbors**. Firstly, it finds the k closest points in the training set using Euclidean distance, then it looks at the classes of these neighbors and assigns the majority class to the new point. I experimented with different values for k and found that **k = 15 gave the best results**.

The results for the KNN model were:

- **Accuracy:** 0.54
- **Precision:** 0.50
- **Recall:** 0.45
- **F1 Score:** 0.48

While KNN performed relatively well, its accuracy was slightly lower than Random Forest, and the precision was similar. The recall was lower compared to the other models, suggesting that **KNN struggles more with identifying positive cases**. KNN is sensitive to the distribution and structure of the data, and its performance can suffer in high-dimensional datasets.

3.2.5 Evaluation and Model Comparison

Out of all the classification models, Random Forest Classifier performed the best, with the highest accuracy and precision. It handled the complexities of the data well, making it a reliable choice for identifying patients at risk of readmission. This is critical for hospitals to improve patient care and reduce unnecessary readmissions, which can save both time and money.

Logistic Regression improved after scaling, but it was still not as effective as Random Forest. KNN, while useful in some scenarios, had trouble identifying readmitted patients, shown by its low recall score. In summary, **Random Forest is the most suitable** model for this task, as it provides accurate and balanced predictions.

3.3 Implementing Clustering Baseline

In this section, I implemented various clustering algorithms to **group data points into clusters** based on similarities. The goal was to **identify natural groupings within the data** to better understand patterns. To evaluate the models, I used the **silhouette score**, which measures how well each data point fits within its assigned cluster compared to other clusters. The silhouette score ranges from -1 to 1, with values closer to 1 indicating that the data points are well-matched to their own clusters, and values closer to -1 suggesting that they may be misclassified.

3.3.1 Load and Preprocess the Data

For clustering, I used the dataset from Task 2 stored in the variable *diabetic_data*, but only selected a subset of features: *time_in_hospital*, *num_lab_procedures*, *num_procedures*, *readmitted*, and *num_medications*. The reason for selecting only a few features was to

reduce noise, focusing on attributes that are likely to show clear patterns or relationships between patients. Clustering can be sensitive to irrelevant or noisy features, so by narrowing down the features, I aimed to improve the quality of the clusters.

Since clustering algorithms generally perform better when features are on a similar scale, I standardized the dataset using *StandardScaler*. This step is crucial because clustering algorithms like K-Means use distances between points to group them, and features with different scales could dominate the results if not scaled appropriately. After scaling the data, I was ready to implement the clustering models.

3.3.2 Implement K-Means Clustering

The first clustering model I implemented was *KMeans*. K-Means algorithm is a method that **groups data into a certain number (k) of clusters**. It firstly forms clusters by assigning each data point to the nearest centroid, after which it recalculates the centroids as the mean of the points in each group, repeating this process until the centroids stop changing significantly (or a maximum number of iterations is reached). The number of clusters, **k**, is a key parameter in K-Means, and selecting the optimal value is essential for obtaining meaningful clusters.

To determine the best value for k, I used multiple methods. First, I applied the **Elbow Method**, which involves plotting the **inertia** (sum of squared distances to the closest centroid) for different values of k and looking for an "elbow" point on the graph, where the inertia starts to decrease more slowly. The Elbow Method suggested **k = 3 as the optimal number of clusters.**

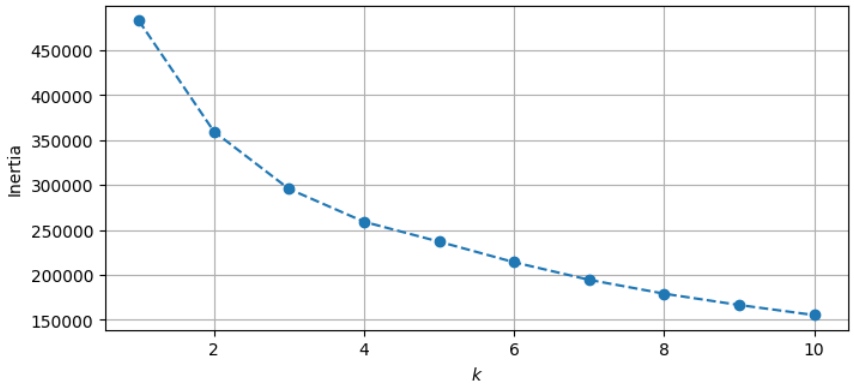

Image 19 Elbow method for determining the optimal k value.

Next, I plotted the **silhouette score for various values of k**. This method measures how similar each point is to other points in the same cluster compared to those in other clusters. The silhouette score showed that **k = 3** or **k = 4 were reasonable choices**. Choosing from higher values of k would not be optimal according to the other methods I used, and k = 2 would not capture the diversity of the dataset.

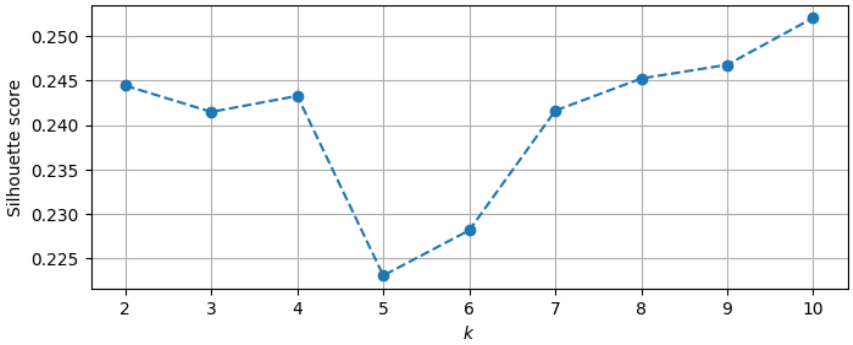

Image 20 Silhouette scores method for determining the optimal k value.

Finally, trying the **Silhouette Analysis** method, which measures how similar each point in a cluster is to the points in the same clusters compared to the points in others, I determined that **k = 3 is the optimal value**.

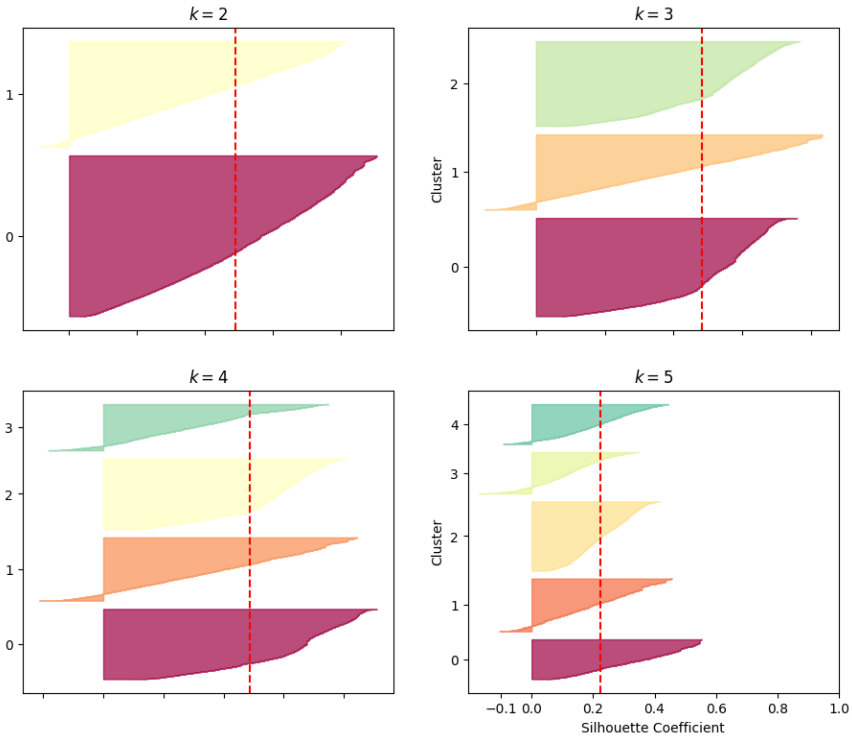

Image 21 Silhouette Analysis method for determining the optimal k value.

After determining k = 3, I implemented K-Means and obtained a **silhouette score of 0.24**. While this indicates that the clusters are somewhat well-formed, there is room for improvement, as the silhouette score is relatively low. However, this score shows that the clusters are better than random assignments, which is a good starting point.

3.3.3 Apply PCA for Dimensionality Reduction

To visualize the results of K-Means, I implemented **Principal Component Analysis (PCA)**, a dimensionality reduction technique that reduces the data to fewer dimensions while retaining the most important information. PCA works by identifying the directions in which

the data varies the most and projecting the data onto those directions. This is helpful in clustering because it **allows us to visualize high-dimensional data in a lower-dimensional space** while preserving the structure of the clusters.

The visualization showed that, while the data points were generally divided into three clusters, **there was some overlap**, particularly in cluster 2, where many points were misclassified or spread across the other clusters. This suggests that the K-Means algorithm might not have fully captured the complexity of the data, and some points are not as clearly separable as others.

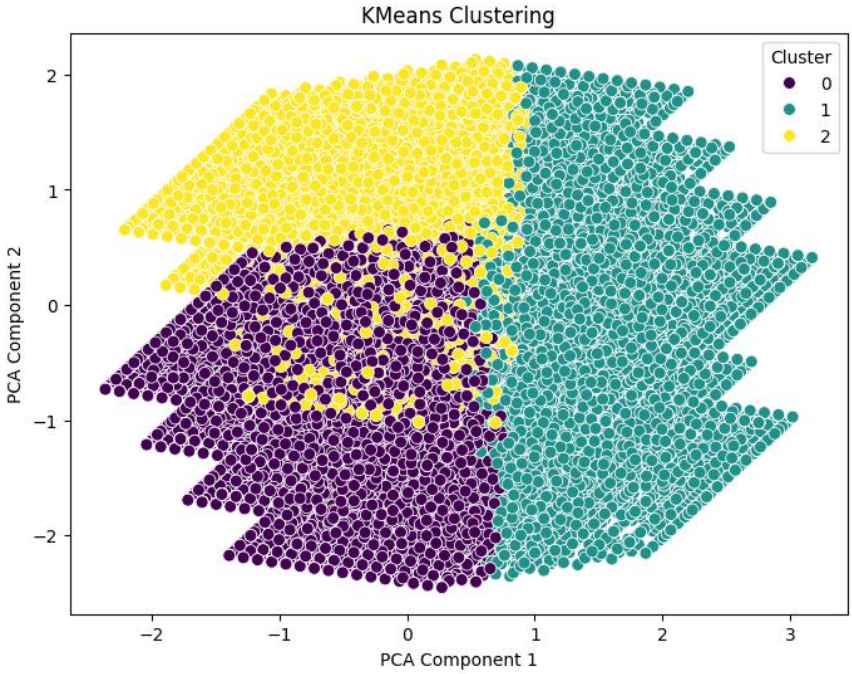

Image 22 K-Means clusters visualization using PCA.

3.3.4 Implement Random Hierarchical Clustering

Next, I implemented *AgglomerativeClustering*, which is a hierarchical clustering method that **builds clusters step by step by merging smaller clusters**. It starts with each point as its own cluster, gradually merging closest clusters together until all points are combined into a specified number of clusters.

Given the computational constraints, I had to **implement this method in chunks**, as the dataset was too large to process all at once. I visualized the results using a **dendrogram**, a tree-like diagram that shows how clusters are merged. By inspecting the dendrogram, I determined that **a threshold around three would result in 4 clusters**, which seemed to balance cluster cohesion and separation well. **The silhouette score for Agglomerative Clustering was 0.21**, which is slightly lower than K-Means.

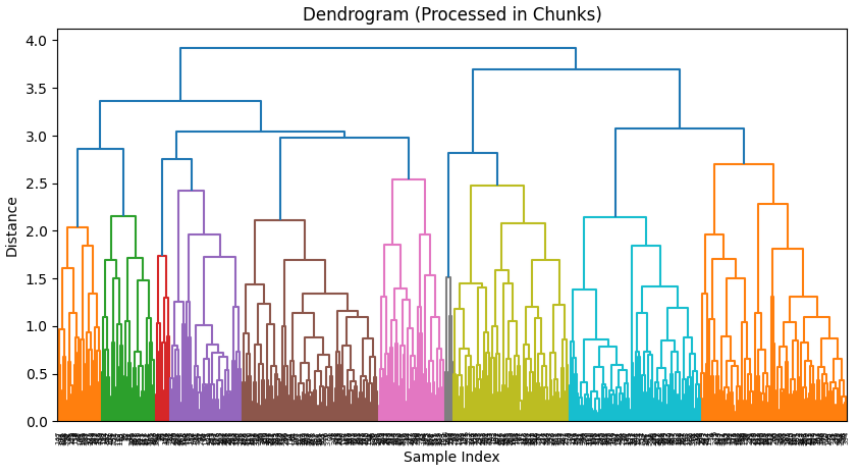

Image 23 Dendrogram visualization.

3.3.5 Implement DBSCAN for Density-Based Clustering

Finally, I implemented **DBSCAN (Density-Based Spatial Clustering of Applications with Noise)**, which is a density-based clustering algorithm that **groups together data points that are close to each other based on a density criterion**. One of DBSCAN's advantages is its ability to identify noise or outliers, which are points that do not belong to any cluster.

For DBSCAN, I set **the distance threshold (ε) to 0.5** and **the minimum number of samples to 5**. However, the resulting clusters **did not identify any points as noise**, suggesting that with these parameters, the algorithm is more likely to form clusters than to detect noise points. The **silhouette score for DBSCAN was 0.09**, which is the lowest of all the clustering models. This suggests that DBSCAN struggled to identify well-separated clusters in this particular dataset, possibly due to its sensitivity to the density parameter and the distribution of the data.

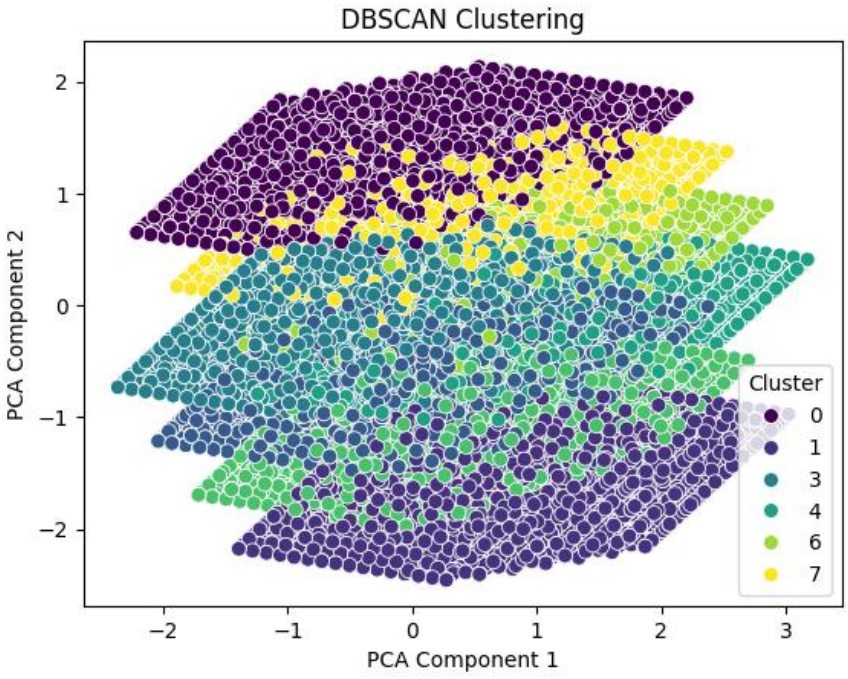

Image 24 DBSCAN clusters visualization using PCA.

3.3.6 Evaluation and Model Comparison

For clustering tasks, **K-Means had the best performance**, with a **silhouette score of 0.24**. It grouped the data into the most meaningful clusters, which is helpful for understanding patterns in patient care and creating targeted treatment plans. Clustering can help hospitals identify groups of patients with similar needs, making care delivery more efficient.

Agglomerative Clustering also performed well but **had a slightly lower silhouette score**, making its clusters less distinct. **DBSCAN**, which is good for identifying outliers, **did not work well with this dataset** because it couldn't form clear clusters. K-Means, especially after the data improvements through feature engineering, gave the best results for identifying useful patient groupings.

3.4 Feature Engineering and Feature Selection

In this section, I focused on enhancing my dataset and models by applying additional feature engineering and feature selection techniques. These steps are crucial for improving model performance by reducing noise, creating more meaningful features, and ensuring that only the most relevant data is used.

3.4.1 Feature Engineering

Building on the extensive feature engineering completed in Task 2, I conducted additional transformations to improve data consistency and eliminate unnecessary complexity. For example, I closely examined the ID columns (*admission_type_id*, *discharge_disposition_id*, and *admission_source_id*) and identified opportunities to simplify their representation. For *admission_type_id*, three IDs (*5*, *6*, and *8*) referred to "unknown" situations, so I replaced these with a generalized *np.nan* value to ensure consistency in handling missing data.

Similarly, the *discharge_disposition_id* column, which initially contained **29 distinct values**, was condensed into 9 general categories, such as "Home-Based Care," "Short-Term Transfers," and "Hospice Care." This simplification retained the essential information while making the data easier to interpret and use.

A similar approach was applied to the *admission_source_id* column, **reducing 26 unique values to 5 categories**, including "Referral," "Transfers," and "Emergency or Legal." These transformations help the models focus on meaningful distinctions without being distracted by unnecessary noise.

Another critical step was identifying and removing features that contained only one unique value. These features, **24 in total**, offered no variability and thus no predictive value for the models. While some of these features were originally constant, others lost their variability due to outlier removal and trimming in Task 2. Eliminating such features ensures that computational resources are not wasted on irrelevant data.

3.4.2 Generate Features

In addition to refining existing features, I created new ones to capture previously overlooked aspects of the data. Diagnoses represented by ICD9 codes in the *diag_1*, *diag_2*, and *diag_3* columns were revisited, and I applied a risk categorization to identify **high-risk conditions**. This process resulted in three binary features (*diag_1_risk*, *diag_2_risk*, and *diag_3_risk*) and a combined *risk_score* feature that quantifies the overall risk level for each patient, with scores ranging from 0 to 3. This transformation helps the models assess patient risk in a straightforward and interpretable manner.

Beyond risk categorization, I developed several new metrics to capture patient and resource activity. For instance, I created a *lab_intensity* feature that aggregates the number of lab procedures and diagnostic tests performed, providing a single measure of diagnostic activity. Similarly, I introduced *resource_intensity* to quantify overall hospital resource usage, aggregating various indicators of hospital activity. Another useful addition was the *total_visits* feature, which sums outpatient and inpatient visits to capture a patient's overall interaction with the healthcare system. Finally, I included clustering labels from KMeans and DBSCAN in the dataset to provide additional group-based context, though I remained cautious of potential data leakage when these labels were used in predictive models.

These new features expanded the dataset's richness, allowing models to better capture relationships and patterns.

3.4.3 Feature Selection

To ensure that only the most relevant features were included, I applied three feature selection techniques. First, I used the **feature importance scores** provided by a Random Forest Classifier to identify which features contributed most to the model's predictions. This method highlights features that have a significant impact on model performance, helping to prioritize them for inclusion. Next, I employed **Recursive Feature Elimination (RFE)**, which iteratively removes the least important features, retraining the model with each iteration to refine the feature set further. Finally, I analyzed a **correlation matrix** using a heatmap visualization to identify features with high correlations (above 0.5 or below -0.5).

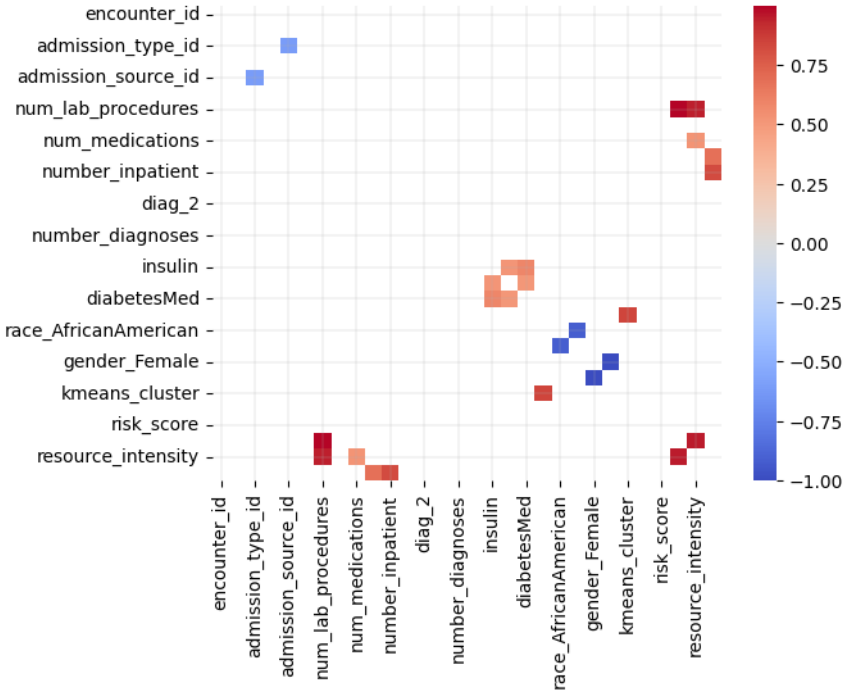

Image 25 Heatmap visualization of the correlation matrix.

Despite these efforts, feature selection **did not lead to performance improvements** in this case. Testing various subsets of features consistently worsened the models' results, suggesting that the full feature set was already optimized. This outcome emphasizes that while feature selection is a valuable tool, its benefits depend on the dataset and model context.

3.4.4 Model Evaluation After Feature Engineering

To evaluate the impact of these feature engineering efforts, I revisited the best-performing models from previous tasks. The results demonstrated clear improvements, particularly for **XGBoost**. After incorporating the new features, the mean absolute error (MAE) **decreased from 1.27 to 1.06**, and the mean squared error (MSE) **dropped from 2.64 to 1.87**. These reductions indicate that the additional features effectively captured more predictive information, enabling the model to make more accurate predictions.

For the **Random Forest Classifier**, I removed cluster labels to prevent data leakage when predicting the *readmitted* feature. Despite this adjustment, the model's performance remained stable, with **only minor changes in metrics** such as precision and recall. This consistency suggests that the refined features preserved the model's effectiveness.

Version	Accuracy	Precision	Recall	F1
Old	0.62	0.62	0.52	0.56
New	0.62	0.61	0.53	0.57

Clustering also showed marked improvement. For **KMeans**, the silhouette score **increased significantly from 0.24 to 0.32** after feature engineering. This improvement was achieved by performing feature selection specifically for KMeans, where I selected the columns *time_in_hospital*, *num_medications*, *risk_score*, *resource_intensity*, *discharge_disposition_id* as the most relevant features for clustering. These selected features effectively captured key aspects of patient behavior and hospital resource usage, enabling the algorithm to create more cohesive and well-separated clusters, validating their relevance to the clustering task.

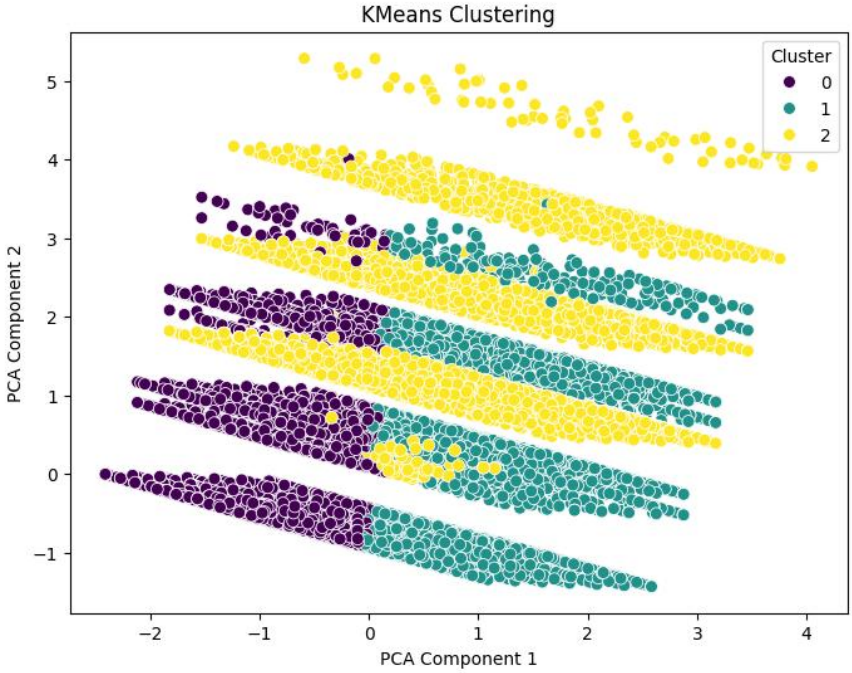

Image 26 K-Means clusters visualization using PCA (after Feature Engineering).

Through feature engineering, I was able to enhance the dataset by introducing new metrics like *risk_score*, *lab_intensity*, and *resource_intensity* while refining existing features for consistency and clarity. These changes **improved the performance** of both predictive and clustering models, demonstrating the value of thoughtful feature design. While feature selection did not yield additional benefits in this case, it confirmed that the dataset was already well-optimized, highlighting the importance of thorough preprocessing and careful evaluation in building machine learning models.

3.5 Hyperparameter Tuning

In this section, I focus on improving my models' performances by performing hyperparameter optimization using a technique called Grid Search. Grid Search systematically tests various combinations of model hyperparameters to identify the best configuration that maximizes performance based on a chosen scoring metric. This method helps fine-tune models to their optimal settings, ensuring better predictions and generalization. For this task, I worked with the best-performing models from earlier stages: **XGBoost**, **Random Forest Classifier**, and **K-Means**.

3.5.1 GridSearchCV for Systematic Hyperparameter Tuning

To perform hyperparameter tuning, I defined parameter grids for each model, considering computational efficiency by limiting the number of values per parameter. Below are the parameters I tuned for each model, with brief explanations:

- **XGBoost**:
  - *n_estimators*: Number of trees in the model.
  - *max_depth*: Maximum depth of the trees, controlling model complexity.
  - *learning_rate*: Step size for updating weights during training.
  - *subsample*: Fraction of the data used for training each tree.
  - *colsample_bytree*: Fraction of features used when building each tree.
  - *gamma*: Minimum loss reduction required to split a leaf node.
- **RandomForestClassifier**:
  - *n_estimators*: Number of trees in the forest.
  - *max_depth*: Maximum depth of the trees.
  - *min_samples_split*: Minimum number of samples required to split an internal node.
  - *min_samples_leaf*: Minimum number of samples required to form a leaf node.
  - *bootstrap*: Whether to sample data with replacement.
- **KMeans**:
  - *n_clusters*: Number of clusters to form.
  - *init*: Method for initializing cluster centroids (e.g., k-means++).
  - *n_init*: Number of times the algorithm will run with different initializations.
  - *max_iter*: Maximum number of iterations to run the algorithm.

3.5.2 Apply GridSearchCV with Cross-Validation

Using the parameter grids, I applied *GridSearchCV*, a method that combines grid search with cross-validation, to each model. The scoring metrics used were **NMSE (Negative Mean Squared Error)** for *XGBoost*, **accuracy** for *RandomForestClassifier*, and **silhouette score** for *KMeans*.

- **XGBoost:** After evaluating 324 candidates across 5 folds, the best parameters were:

{'colsample_bytree': 0.8, 'gamma': 0.2, 'learning_rate': 0.1, 'max_depth': 7, 'n_estimators': 200, 'subsample': 1.0}

- **RandomForestClassifier**: After evaluating 162 candidates across 5 folds, the best parameters were:

{'bootstrap': True, 'max_depth': 20, 'min_samples_leaf': 4, 'min_samples_split': 10, 'n_estimators': 200}

- **KMeans**: After evaluating 72 candidates across 5 folds, the best parameters were:

{'init': 'k-means++', 'max_iter': 300, 'n_clusters': 3, 'n_init': 10}

These configurations should ensure that the models are tuned to capture the patterns in the data more effectively.

3.5.3 Nested Cross-Validation to Validate Stability of Hyperparameters

To confirm the stability of the selected hyperparameters, I performed **nested cross-validation**. This method involves an inner loop for hyperparameter tuning and an outer loop for model evaluation, providing an unbiased estimate of performance. I used *StratifiedKFold*, which ensures class balance within each fold, splitting the data into **five outer folds** and **three inner folds**.

Due to computational constraints, nested cross-validation was only applied to *XGBoost*, resulting in an **NMSE of -2.122** (approximately an average MSE of 2.122), indicating stable performance across folds. This process validates that the selected parameters are not overfitting to any particular data subset.

3.5.4 Evaluate Tuned Models with Multiple Metrics

Finally, I re-evaluated each model using the optimized parameters to compare their performance with the earlier configurations. Surprisingly, **the performance of *XGBoost* slightly worsened** after tuning, increasing MAE from 1.06 to 1.127 and MSE from 1.865 to 2.098. This could be due to over-regularization or the tuned parameters not generalizing as expected. ***RandomForestClassifier*'s results improved** across all metrics, with accuracy increasing from 0.628 to 0.633, precision from 0.617 to 0.622, recall from 0.529 to 0.544, and F1 score from 0.57 to 0.581, indicating a clear benefit from the tuning. Finally, ***KMeans*'s silhouette score remained stable** at 0.315 as the suggested hyperparameter tuning was already in place.

Hyperparameter tuning significantly improved the **RandomForestClassifier**, while results for **XGBoost** showed a slight decline, warranting further investigation into the choice of hyperparameters. For **KMeans**, the initial parameters were already optimal. These results showcase the importance of exploring and validating hyperparameters to achieve the best balance between complexity and generalization.

3.6 Model Selection

In this final part of the project, I evaluated and compared all machine learning models implemented throughout, focusing on regression, classification, and clustering tasks. This comprehensive comparison highlights the strengths and weaknesses of each approach, providing insights into their effectiveness for different aspects of the problem.

3.6.1 Compare and Choose the Best Model

Regression Models

The task of predicting hospital stay duration was evaluated using two metrics: **Mean Absolute Error (MAE)** and **Mean Squared Error (MSE)**.

Model	MAE	MSE
Linear Regression	1.461	3.209
Decision Tree	1.686	5.53
Gradient Boosting	1.349	2.818

XGBoost	1.062	1.869

Conclusions:

- XGBoost demonstrated the best performance with the lowest MAE and MSE, making it the most accurate and reliable regression model for this task.
- Gradient Boosting also performed well but slightly lagged behind XGBoost.
- Linear Regression was a simple but effective baseline model, while the Decision Tree model showed the poorest performance.

Classification Models

The task of predicting whether a patient would be readmitted was evaluated using four metrics: **Accuracy**, **Precision**, **Recall**, and **F1 Score**.

Model	Accuracy	Precision	Recall	F1
Logistic Regression	0.618	0.595	0.563	0.578
Random Forest	0.632	0.621	0.541	0.579
K-Nearest Neighbors	0.537	0.503	0.454	0.478

Conclusions:

- Random Forest achieved the best overall performance, with the highest accuracy, precision, and F1 score, making it the most suitable model for this task.
- Logistic Regression was a close second, offering consistent results across metrics with a simpler implementation.
- K-Nearest Neighbors struggled to perform well, indicating that it may not be well-suited to the characteristics of this dataset.

Clustering Models

The clustering task aimed to group similar patients, evaluated using the **silhouette score**, which measures how well-defined clusters are.

Model	Silhouette Score
K-Means	0.315
Hierarchical	0.206
DBSCAN	0.091

Conclusions:

- K-Means outperformed the other clustering algorithms, producing moderately well-defined clusters. Feature engineering and selection were crucial in achieving this result.
- Hierarchical clustering showed a weaker performance, and DBSCAN struggled to create meaningful clusters, potentially due to parameter sensitivity or the structure of the data.

3.6.2 Finalize and Save the Best Model

After identifying the best models for each task—**XGBoost** for regression, **Random Forest** for classification, and **K-Means** for clustering—I retrained them using the entire dataset and exported them for future use with Python's *joblib* library. These models could be integrated into hospital systems, helping with decisions like predicting hospital stays, identifying patients at risk of readmission, and finding patterns in patient data.

4. Conclusion and Reflection

This project demonstrates the potential of machine learning in addressing key challenges in healthcare, such as predicting hospital stays, identifying readmission risks, and discovering patterns in patient data. By building and optimizing machine learning models, this study offers practical tools and insights for improving hospital resource management and patient care.

XGBoost stood out as the best model for predicting hospital stays. It handled complex relationships in the data better than simpler models like Linear Regression or Decision Trees. For identifying patients at risk of readmission, Random Forest was the most reliable, with the highest accuracy and precision. In clustering tasks, K-Means successfully grouped patients into meaningful clusters, which could help with planning care strategies.

Feature engineering was a cornerstone of this project's success, significantly improving model performance. Moreover, the exploratory data analysis (EDA) conducted at the beginning of the project provided a solid foundation for understanding the dataset, identifying trends, and preparing for feature engineering and model development.

While the models performed well in this study, several areas for future work could enhance their usefulness. First, there is room for additional feature engineering and selection. Exploring deeper relationships in the data and creating interaction features could yield even better results. Second, the use of ensemble learning methods—combining predictions from multiple models—could improve overall performance and robustness, especially for regression and classification tasks. Investigating more advanced ensemble techniques like stacking or boosting could be particularly promising.

Further, it is essential to assess whether the performance metrics accurately reflect real-world effectiveness. This step could uncover any potential biases or limitations in the models that were not evident in this study.

Finally, refining hyperparameter tuning methods, such as leveraging advanced techniques like Bayesian optimization or randomized search, could significantly enhance model performance while reducing computational costs. Incorporating parallel processing during tuning could further streamline the optimization process, especially for larger datasets.

In summary, this project highlights the importance of thoughtful data preparation, feature engineering, and iterative modeling in achieving meaningful results. It also outlines clear steps to build on this foundation, paving the way for more advanced machine learning applications in healthcare. These tools have the potential to help hospitals make smarter, data-driven decisions, improving both patient care and resource management.