[Final Report Template]

[Block Y1B-2024-25]

Please, use this template to write down your final report for the DataLabTasks. If you have any questions, please, contact your mentor or the content responsible.

Important Notes:

- Please, rename the file to "FinalReport_<your_name>_<studentnumber>.pdf" before submitting it.
- Upload this template to the 'Deliverables' folder in your BUas GitHub repository.
- You are allowed to add as many sub-sections as you need.
- This report will be used for your final presentation in Block Y1B-2024-25 in week 8.
- The final report should only be submitted in "**PDF"** format.

Introduction

For this project, I will be using the **Diabetes 130-US Hospitals for Years 1999-2008** dataset. It represents collection of patient data for 10 years. It's main goal is to correctly provide care and prevention for patients with diabetes, many of which don't get this in time.

Problem Statement

Even with modern healthcare, many patients still do not get the care they need in time. This can lead to worsening health conditions and, in some cases, even preventable complications. Addressing these gaps requires a concerted effort from healthcare providers, policymakers, and communities to ensure timely access to necessary treatments. With a well-trained model, I aim to predict whether the patient will be readmitted or not based on patients' data. By analyzing various factors such as previous admissions, treatment history, and social determinants of health, I can develop insights that may help healthcare systems implement targeted interventions. Ultimately, this proactive approach could significantly improve patient outcomes and reduce strain on hospital resources.

Tasks:

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

1A. Exploratory Data Analysis with SQL

1A.1: # Count the total number of records in the encounter table

CODE:

```
SELECT COUNT(*) AS encounter_count
FROM encounter;
```

After executing code above, I get that the total number of records is 101766. So far, no empty or null values have been removed, so I can assume there are data which provide no value for me.

1A.1: # Check the distribution of different admission types

CODE:

```
SELECT description, count(*) AS admission_count
FROM encounter as e
LEFT JOIN admission_type as adt
ON e.admission_type_id = adt.admission_type_id
GROUP BY description
ORDER BY admission_count DESC;
```

To check distribution of admission types I counted the total number of all specific admission types. To show me the name of admission types I needed to join tables „encounter“ and „admission_type“ on Primary and Foreign keys. After this I was able to see that the most common admission type was „Emergency“ with total of 53990 records. On the opposite side with the least, was “Newborn” with 10 records. There were also unknown descriptions such as “Not Available” with 4785 records, “Not Mapped” with 320 records and finally also a “NULL” with 5291 records. This means we would need to take care of these data before working with them.

1A1: # Explore the top discharge dispositions

CODE:

```
SELECT description, count(*) AS discharge_count
FROM encounter AS
LEFT JOIN discharge_disposition AS dd
ON e.discharge_disposition_id = dd.discharge_disposition_id
GROUP BY description
ORDER BY discharge_count DESC;
```

With the code above, I get a list of all discharge types and their total number. Most common discharge description being „Discharged to home“ with 60234 records. On the fourth most common discharge type we have again “NULL” and on 9th place we have “Not Mapped”. The total number of these two missing values is around 4000, so there should be no issues when deleting these records.

1A2 : #Check for missing values in the race column (Hint: Count the occurance of each unique value in this column)

CODE:

```
SELECT count(*) AS race_count, race
```

FROM patient

GROUP BY race

Looking at race data, we have a solid result. Out of all records, only 1948 were marked as "?" and 1178 as "Other". All other types have their own description and number of records.

1A2: #Check for missing or unusual values in the weight column

CODE:

SELECT weight, count(DISTINCT e.patient_nbr) AS weight_count

FROM encounter AS e

LEFT JOIN patient AS p

ON e.patient_nbr = p.patient_nbr

GROUP BY weight

ORDER BY weight_count DESC

Here we explored the weight data categories and their own respective records. By ordering them, I notice that 68665 rows have „?" as weight. All the other 9 categories barely make over 1000. This column will have no use for us for project because of the quantity of missing data.

1A3 : # Explore the age distribution of the patients

CODE:

SELECT age, count(DISTINCT e.patient_nbr) AS age_count

FROM encounter AS e

LEFT JOIN patient AS p

ON e.patient_nbr = p.patient_nbr

GROUP BY age

ORDER BY age DESC

By running this code and looking at the age distribution, I notice that the biggest part of records, are patients above the age of 50. This could be an issue if we want to explore something related younger ages, because we will only have reliable data for old patients, but if we want to diagnose old patients, then this is going to be useful for us.

1A4 : # Analyze how different admission sources contribute to hospital admissions

CODE:

SELECT description,count(*) AS count_of_admissions_per_source

FROM encounter AS e

LEFT JOIN admission_source AS ads

ON e.admission_source_id = ads.admission_source_id

GROUP BY description

ORDER BY count_of_admissions_per_source DESC;

Looking at this I noticed „NULL“ being the third most recorded admission with 6781 records, but there is plenty of data for the first and second admissions, so we could drop those missing values and work with data without having too much of an impact. There are also admissions with very low records (2 or less), such as: Sick Baby, Transfer from Ambulatory Surgery Center , Extramural Birth and Normal Delivery.

1A4 : #Investigate which admission types correspond to specific admission sources

CODE:

SELECT ads.description, adt.description, count(*) AS count_of_admissions_per_source

FROM encounter AS e

LEFT JOIN admission_source AS ads

ON e.admission_source_id = ads.admission_source_id

LEFT JOIN admission_type AS adt

ON e.admission_type_id = adt.admission_type_id

GROUP BY ads.description, adt.description

ORDER BY count_of_admissions_per_source DESC;

With this code above, I can explore all the combinations of admission types and admission sources and see their total count and how they are connected. Most of data is reasonably related, but there are also instances where either admission type is null and source having a label, or the other way around, or both being null. I ran additional query to check the total number of these instances where NULL occured and the total number of these was 9112, which is not that much considering there is 101766 records, so we can delete them if we decide to without having an issues.

1A5: # Find the average time in hospital for each admission type

CODE:

SELECT DISTINCT description, ROUND(AVG(time_in_hospital),2) AS average_time_per_type

FROM encounter AS e

LEFT JOIN admission_type AS adt

ON e.admission_type_id = adt.admission_type_id

GROUP BY description

ORDER BY average_time_per_type DESC;

Using ROUND and AVG I got the average time spent for each admission. The third logest time was for NULL with 4.58. I personally would not rely on these data at first, so I checked with additional query the total sum of hours per each admission type and saw that altough Trauma Center with the highest average time has only 102

total hours and 21 records, which I don't consider reliable due to lack of records. Other types such as: Urgent, Emergency, Elective had enough records to make a realistic time estimate.

1A5 : #Investigate readmission rates by admission type

CODE:

```
SELECT description, COUNT(readmitted) AS rate_of_readmissions
FROM encounter AS e
LEFT JOIN admission_type AS adt
ON e.admission_type_id = adt.admission_type_id
WHERE readmitted NOT LIKE 'NO'
GROUP BY  description
ORDER BY rate_of_readmissions DESC;
```

At first, my query did not include condition where I removed „NO", which did not make sense since I want to look at readmission rates, so I fixed that, and then I was able to see which admission type had readmissions. Then I noticed that the readmissions are either <30 or >30, so there was no point in dividing them since both are more than 0. With that I was able to see how many times people got readmitted. These data seem useful with top three results being labeled admission types and fourth being NULL with 2817. There were also types such as Not Available with around 2000 and Not Mapped with around 100 and Newborn being the least readmissions with only 3 records.

1A6 : # Compare discharge dispositions across different admission types

CODE:

```
SELECT DISTINCT adt.description AS admission_type_desc,  dd.description AS outcome
FROM encounter AS e
LEFT JOIN discharge_disposition AS dd
ON e.discharge_disposition_id = dd.discharge_disposition_id
LEFT JOIN admission_type AS adt
ON e.admission_type_id = adt.admission_type_id
```

Running this query, I got 117 different combinations of discharge dispositions across different admission types. I made a quick addition to that query so that I see the number of records for each of them and sorted them. Most of the outcomes are Discharged home, but there are still missing and incomplete data with NULL and Not Mapped as labels. It would be prefered to somehow handle these before using the data.

1A6 : # Compare readmission count by discharge disposition

CODE:

```
SELECT DISTINCT dd.description AS Discharge_Disposition, count(readmitted)
readmission_count
FROM encounter AS e
LEFT JOIN discharge_disposition AS dd
ON e.discharge_disposition_id = dd.discharge_disposition_id
WHERE readmitted NOT LIKE 'NO'
GROUP BY dd.description
ORDER BY readmission_count DESC
```

Lastly this using this query I was able to see if patients were readmitted after discharge disposition. For this I had to make sure to remove all instances where readmission was „NO“. After that I was able to see the real count of readmission. Again there are instances of NULL and Not Mapped, but these only make up about 2000 records, which could be handled to improve data quality

1B. Exploratory Data Analysis with Python

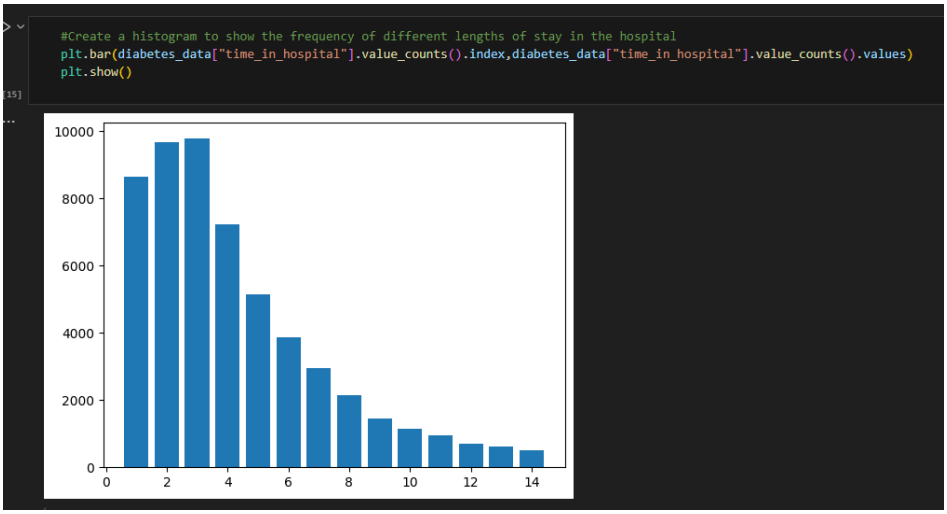
First, I loaded my data and looked at it using .head(), .shape, and .columns to get an overview of the dataset. The entire dataset contains 101 766 rows and 50 columns. Columns include numerical but also categorical features. Some columns were clear to understand, but some needed the use of domain knowledge, like diagnoses.

After that I used .value_counts() on all columns to see all the unique values for each column and their number. With this I notice there are “?” in the data that represent missing values. Although they are present only in a few columns, these columns may be important.

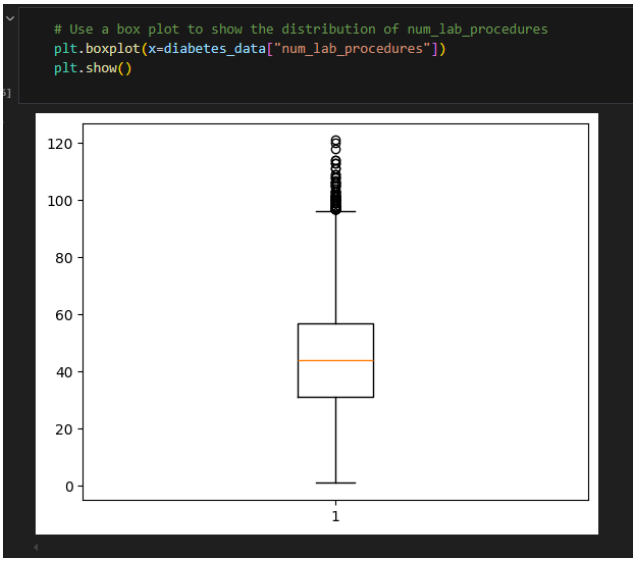
Then I went ahead and did visualisations of this data. The first visualisation was about age, where I noticed that the majority of ages are above 50.

The distribution of gender is nearly equal, with sufficient representation for both genders, yet a minor proportion of Unknown/Invalid appears.

Column “time in hospital” was right-skewed with a range of time going from 0 to 14. I also find out that the time is in days.

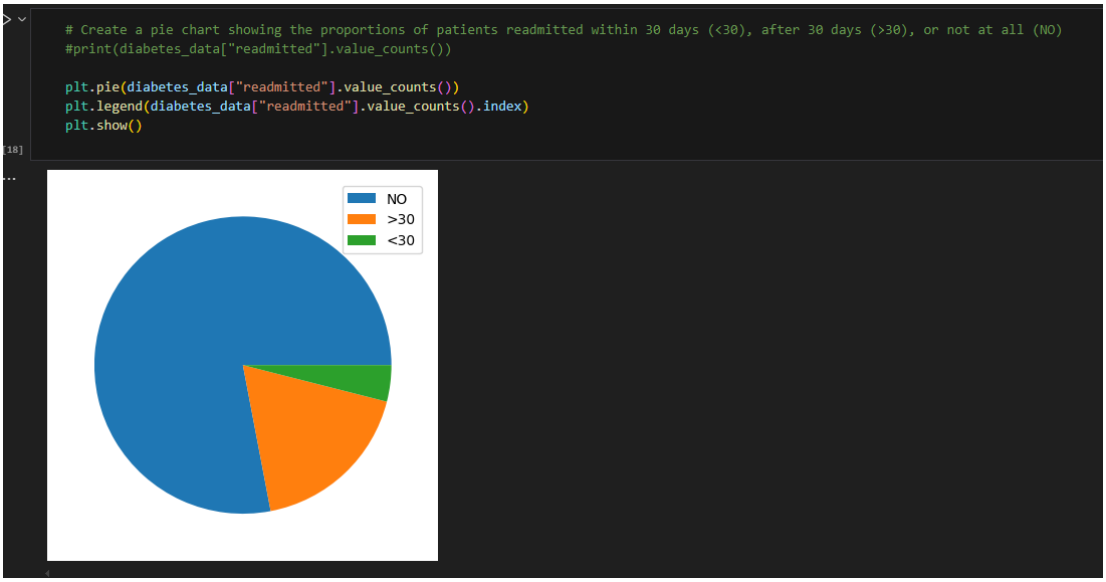


Using a boxplot on “num lab procedures,” I see that the average amount of lab procedures is around 40 and that there are outliers present in this column.



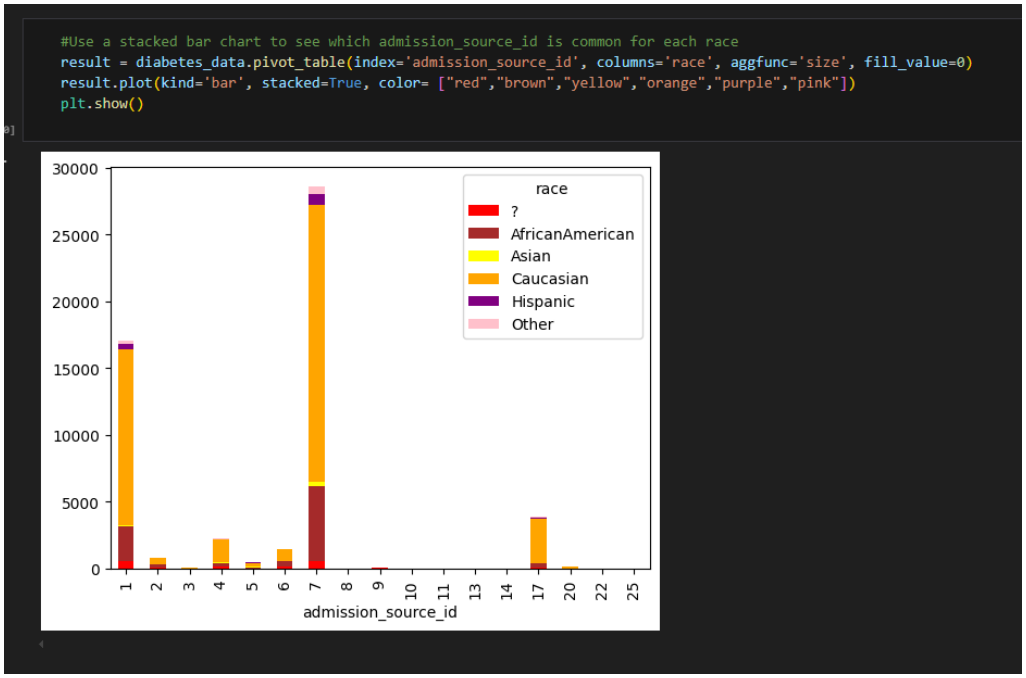
Then I wanted to see the values in medicine, so using a for loop, I plotted all the medications. Here I notice that most of them have only one dominant value, and only a few of them have all 4 instances (“No,” “Steady,” “Up,” ”Down”).

I then used a pie chart on readmissions. I could easily see that more than half of the values are “No,” but with combining “>30” and “<30,” it’s almost evenly distributed.



Then I moved onto the race column. With a bar plot, I quickly saw that Caucasian was the most common race in the data set, with other races having less than 5000 records.

I continued using races and displayed them on a stacked bar chart to see which admissions are common for which race. Admission source 7 was the most common, which in mappings is an “emergency room,” and the second most common is “physician referral“. Both of these have „Causasian“ as the most common race, with „African/American“ being the second most common.



Next on the list was medical speciality. Using a bar plot, I notice that „?“ had almost 50 thousand entries, which is over half the total. For this column I am considering dropping it, because there is no way to precisely input correct values into them.

Then I plotted the top 10 medical specialities by readmissions. If I exclude the „?“ entry, most of them had „NO“ as the most common out of the tree. Looking at „num lab procedures“ again with histplot, I now see where the outliers

are and that the data is right-skewed. Then I made boxplots of all numeric columns to detect outliers, and there were many that had these outliers (examples: number_diagnoses, number_emergency, num_medications).

With this I had a rough idea of what to do with the data and the steps I needed to take to clean it.

2. Data Processing

2.1 Initial Cleaning and Pre-processing

My first step was removing duplicated rows based on a **“patient_nbr”** column. In total, **30248** duplicated rows were dropped. At first I did not want to remove these patients, because if I want to predict whether the patient gets readmitted or not, they need to stay in the dataset in order to get more **“readmitted”** cases. But in many papers I’ve read, they removed them. So I decided on removing it too, but in later stages, I will maybe experiment and keep them in to see how it affects the results. (Liu VB, 2024)

The next step was changing categorical values to numerical so that the model can work with the data. Multiple columns had only two values, so these were **changed to 1 and 0 (“Gender,Change,DiabetesMed”).**

For readmissions, if the patient wasn't readmitted, then the value was set to 0, and all the others were set to 1.

Columns for diagnoses had letters in them, so I removed them to convert the string to int. Thankfully they contained only two letters and a symbol. So, I only had to **remove "V," "E," and ">.".** For the age column, instead of mapping the values, I replaced them with the mean of the interval. So, if the age was marked as **"[20-30]," it is converted to 25.**

After this there were still diagnoses mapped as **"?"** but their total amount was around **1500**. So I **dropped** them because replacing them incorrectly may influence the model. While dropping these rows I also dropped row with gender that was **"Unknown/Invalid"**. After that, I was able to change the data type of these four features. As for other columns I didn't mention, they got dropped, because most of them didn't really correlate that much at first. Then my last step was removing outliers from certain columns.

2.2 Handling Metadata and Missing Values

Here I check for metadata. Column names consist of basic patient information like gender, age, diagnoses, medications. Then columns have IDs which are related to hospital, admission type, discharge disposition or time in hospital. I removed missing values in preprocessing, so there should not be any missing data left. So far I am left with **68529** entries and **19** columns. I will use this for training models in the following sections. All columns are already transformed into numerical; more specifically, diagnoses are as floats, and all the others are as ints.

3. Machine Learning Baselines

3.1 Implementing Regression Baseline

For the regression task, I will be trying to predict **time in hospital**, since it is a numeric value for which regression models are optimal. After some research, I have also learnt that the time in the hospital is in **days**, not in hours.

For my first regression model, I have chosen **the Ridge** linear regression model. It is a type of linear regression that adds a **regularisation** term to the loss function. This penalty **shrinks** the **coefficients** of the model, **reducing overfitting** by discouraging large weights while keeping all features in the model. It's useful when there's **multicollinearity** (several independent variables are correlated) or when the model is **overfitting** the training data.

First I import the evaluation metrics and Ridge model, and then I instantiate it. I fit in the training data, then predict on the test set and calculate MSE and MAE.

The second model for regression was **Decision Tree**. It builds a tree structure where each node represents a decision based on a feature, and the leaves contain values. the predicted Without setting the **max_depth** parameter, the results were very off. **Max_depth** is a **hyperparameter** in decision trees that limits the maximum number of levels the tree can have. It controls the depth of the tree, preventing it from growing too large. With trial and error, I was able to get the best results with **max_depth 9.**

The next model was the Gradient **Boosting Regressor**. It is an **ensemble** machine learning **model** that builds multiple decision trees sequentially. Each tree corrects the errors of the previous one by minimizing a loss function using gradient descent. It combines all the trees' predictions to produce a strong, accurate model. It's powerful for handling complex, non-linear data but can be prone to overfitting if not properly tuned.

For the last regression model, I implemented **XGBoost Regressor**. It's an advanced implementation of gradient boosting designed for speed and performance. It improves on standard gradient boosting by using techniques like regularisation, tree pruning, and parallel processing. Among all other models, XGBoost Regressor is supposed to be the most robust one of all these models.

Here are the overall results from regression models:

Model	MAE	MSE
Linear Regression	1.861866	5.996177
Decision Tree	1.741247	5.511803
Gradient Boosting	1.663609	4.938055
XGBoost	1.590066	4.609218

Honestly, these are the results I expected. Although I wished that linear regression would be higher, I think due to the nature of the dataset, it's more difficult for it to calculate the results. But as with the others, I expected the **XGBoost** to be the most efficient one, which was true according to my results. It has a **1.59 MAE,** which means it is off by one and a half days, and for MSE it's off by 4.6. I only used the models without tuning or further feature engineering, so maybe the results will improve.

3.2 Implementing Classification Baseline

For **classification** I have decided to predict **readmissions**, since I think it's more suited for these models. I started making a copy of the dataset and then splitting it.

My first model, which I implemented for classification, is **LogisticRegression**. It's a linear model used for binary classification tasks. It predicts the probability of an outcome belonging to one of two classes using the sigmoid function to map values to a range of 0 to 1. I started by importing it along with performance metrics such as **accuracy, precision, recall, and F1 score**. I instantiated the model with a max_iter of **5000**.

The next model for classification is the Random **Forest Classifier**. It is an **ensemble** learning model that combines multiple **decision trees** to perform classification tasks. Each tree is trained on a random subset of the data and features (using bootstrapping and feature sampling). The final prediction is made by aggregating the predictions from all trees, typically using majority voting. I personally think this model will provide the best results considering the state of the dataset.

Lastly, for classification, I have tried **K-nearest neighbours.** This model works by finding the **k closest data points** (neighbours) to a given input based on a distance metric. It can be used for regression or classification. So for classification, it assigns the class that is most common among the neighbours.

When choosing which metrics to look at and improve, I need to consider my **objective** and the **dataset**.

Accuracy is best when the dataset is balanced, and all errors have equal importance.

Precision is best when false positives are critical to avoid.

Recall is best when minimizing false negatives is important.

F1-score is best when we want to balance recall and precision.

After working and looking at the data, I know that the dataset is **not really balanced**, so **accuracy** is **not** going to be the best metric. For **precision,** predicting that someone has to come back to the hospital (the model **predicts 1**) even though they didn't need to (the true **label is 0**) as prevention doesn't sound that bad; rather be safe than sorry. With **recall,** I want to get the **best** score because I want to have the **least** amount of people getting predicted as not readmitted (**model predicted 0**) while they indeed are and will be needed to get readmitted (**true label 1**). As for **F1,** it is a good **balance**, but as I mentioned I would rather have a higher number of people getting preventive care than to neglect those who really need it.

With this theory I will be focusing on **minimising false negatives**. So, **recall** is my metric I want to improve.

Here are the results of their performance:

Model	Accuracy	Precision	Recall	F1-score
Logistic Regression	0.621042	0.573780	0.173105	0.265969
Random Forest	0.631183	0.563437	0.311258	0.400995
K-Nearest Neighbors	0.579746	0.438168	0.211185	0.285005

Based on the results, I was yielding the **best recall** score on **Random Forest**. This model was doing overall very well, except in **precision,** where **logistic regression** was better, but as I previously mentioned, this metric has a **secondary importance**. In the next sections I will be further working on this model.

3.3 Implementing Clustering Baseline

The next couple of models will be for clustering. I will not be choosing any target, because clustering is supposed to help with finding some **underlying** similarities between data.

Before implementing models, I must **scale** the data, because clustering works best with scaled features. For this I used the **StandardScaler** provided within **sklearn**.

The first baseline model I used was **the K-means** clustering model. This model is grouping data points into k clusters based on their features. It works by assigning each data point to the nearest cluster centre and then updating the centre to the mean of all points assigned to it. For my k, I will use 3, and for evaluation, I will be using the silhouette **score**.

After this I used **PCA** for dimension reduction to **visualise** the output of the K-means cluster labels. Although it looks as if the three clusters are well separated, the silhouette score says otherwise.

The next model for clustering is **Agglomerative Clustering**. It is a type of hierarchical clustering that builds clusters in a bottom-up approach. It starts with each data point as its own cluster and iteratively merges the closest clusters based on a distance metric until all points are in one cluster or a desired number of clusters is reached.

The process is visualised using a **dendrogram**, which shows the hierarchy of cluster merges. It's effective for discovering nested structures but can be computationally expensive for large datasets.

At first I tried to run the model on the entire dataset at once because I wanted to see for myself if it really is **computationally expensive**. After consuming my entire memory and crushing the VSCode 2 times, I have decided to believe this and try another approach.

I decided to take a **chunk-based** approach for hierarchical clustering. Instead of processing the entire dataset at once, I broke it down into smaller, more manageable chunks. First, the data was split into chunks based on a defined **chunk_size**. By creating indices for these chunks, I sliced the data into smaller subsets and processed each one individually. For each chunk, I computed a linkage matrix using the linkage function, which represents the hierarchy of clusters within that subset of data. Then, I applied agglomerative clustering to assign labels to the data points in the chunk. To summarise the chunk, I calculated the centroids for each cluster by averaging the data points belonging to that cluster.

Once all the chunks were processed, I combined their **centroids** into a single dataset. This significantly reduced the size of the data and made it easier to apply hierarchical clustering on the entire dataset. Using the combined centroids, I performed hierarchical clustering again and assigned final cluster labels.

To evaluate how well the clustering performed, I calculated the silhouette score, which measures how well each data point fits into its assigned cluster. This score gave me an idea of the overall quality of the clustering. Finally, I plotted a **dendrogram** based on the final clustering results. To keep it clear and readable, I truncated it to show only the top levels of the hierarchy.

This **dendrogram** shows that the data splits into **four main groups**, represented by the orange, green, purple, and brown clusters. The orange and green groups stand out because they merge at lower distances, meaning the data points within these groups are quite similar to each other. On the other hand, the purple and brown groups are a bit more diverse, as their branches merge at higher distances.

Lastly, I used **DBSCAN**. It is a clustering algorithm that groups data points based on their density. Unlike other clustering methods, it doesn't require specifying the number of clusters beforehand. Instead, it identifies clusters as areas of high point density, separating them from sparse regions, which are labelled as noise. I have set the **esp,** which is a parameter that decides the **maximum distance between two samples**. The second parameter I set was **min_samples,** which is responsible for the number of points in neighborhoods to be considered as a core point.

After comparing the results, I was surprised to see K-means with such a low score. Maybe I have chosen the wrong parameters, but other models had okay scores, with DBSCAN having the highest silhouette score. I will continue with this model after performing feature engineering and parameter tuning, to see its results.

	Clustering Method	Silhouette Score
0	K-Means	0.100670
1	Hierarchical	0.398958
2	DBSCAN	0.419849

3.4 Feature Engineering and Feature Selection

My first step was **replacing** some mapped IDs to null based on the "**IDS_mapping.csv**" file. In the mapping some numbers were labeled as "**Not available**" or "**NULL**". This was the case for all three ID columns. Then I checked how many nulls were now present in the dataset. In total **15000**. I decided to **drop** them, but maybe for future It would be better to replace them with something else.

After removing the null values, it was time to re-map the IDs. There were many unnecessary mappings, which could be grouped together. These mappings were based on work I had found on the internet (Zeglam, 2020). Then I change the type of features to int and float so I can work with them.

Next I changed the diagnosis numbers to according to **IDS-9 codes**. Again this was **inspired** by the **work** I have found, but here it became clear that **domain knowledge** was also important for this project.

That concluded my **feature engineering**. I've done most of stuff in the **pre-processing** part of this project, so here I just implemented what I have found in other works and projects.

I didn't see the need to **generate statistical features**, because my main objective is to **predict categorical** label (**readmitted**). Although there may be some crazy statistical feature which would improve my model, for now it is beyond my skill level. As for **feature transformation**, although it's not that bad to use it, I will rather go without it for my project. Maybe in later stages for the sake of experimenting I will scale the data and train the model on it to compare the results.

After doing **feature engineering** I performed **feature selection**. For this I used **XGBClassifier** and I wanted to see feature importance based on "**readmitted**" column. By looking at the results, I have decided to use features which have a feature importance more than **0.05**. I was then left with **7 out of 18** features. I then created a new dataframe which contained these features and I appended the target one.

Then I re-trained the Random Forest Classifier on the new features. When comparing the baseline model results and the results from newly trained model after feature engineering and feature selection, I noticed a performance approvement in accuracy of 1%, but all the other metrics got way worse. My theory is that the feature engineering part was helpful, but when doing feature selection I have reduced the dimensions way too much and now the model can't find any connection. I will test this in later parts.

After that I also re-trained the DBSCAN clustering model. With just feature engineering, I was able to improve my results to have a silhouette score of 0.67.

I did the same process for regression too, so my target feature was "**time_in_hospital**". I checked the feature importance and chosen only **7 most important features**, but these were different from the classification one. Then I ran the **XGBoostRegresso**r again. I was expecting some sort of improvement, but instead I got worse results. Honestly I don't know what to think of this. Maybe although some features were not important for "**time_in_hospital**" but were important between each other which resulted in better results? For now I will keep this and try some hyper tuning to see if it helps.

3.5 Hyperparameter Tuning

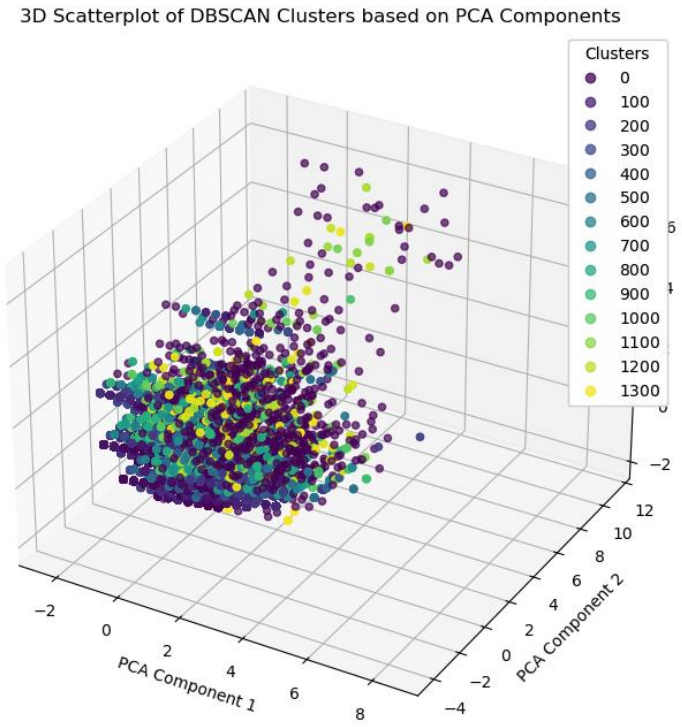
First I prepared parameter grid. I have prepared **7 parameters** I wanted to tune. For each there were different values but in total it was **3600 fits** and since I was doing 3 cross validations inside the Grid Search it resulted in **10 800 fits**. This took **4 hours** to complete, so when I got the results, I marked them down so I don't need to run the same parameters again next time. I also reduced the parameters in the grid, because it is computationally expensive and I want to be able to run the file as fast as possible, but I am keeping the resulting parameters in the grid.

I then used these parameters to re-train my Random Forest Classifier again. I expected to get better results across all metrics, but I couldn't be more wrong. There were really small changes in the metrics. All of them changed approximately by **0.004**. I am still using the data from feature selection, so at least there is some sort of improvement, because from previous observations it did not help classification to reduce the already reduced features.

For regression I made a grid too, but it only had **6 parameters** and in total it would be **2160 fits** not including cross validation. So, for this I will not be using **GridSearch** but **RandomSearch**. It will not result in the best parameters, but It should be able to find good or decent parameters while not being computationally expensive. When initializing the RandomSearch, I set the parameter "**n_iter" to 300**, so it randomly tries **300 combinations** of parameters and chooses the best one. With cross validation it performs **900 fits**. After getting the results, I used them to retrain the model and even though I am using the previously mentioned feature selected dataset, the results did get **better** for MAE and for MSE.

Then I did hyper parameter tuning on **DBSCAN**, because I felt like it could have the most potential. One problem is that there is **no** such thing as **Grid Search** for **DBSCAN**, so it had to be done manually. I used a for loop to just iterate though different eps and different **min_sample**. Every combination is then evaluated and the best one with the best parameters is saved. It took two and a half hours to complete, but based on the best silhouette score the parameters are **eps: 0.5 and min_sampes: 2 with a silhouette score of 0.97**. This indicates that the data points belong to correct clusters.

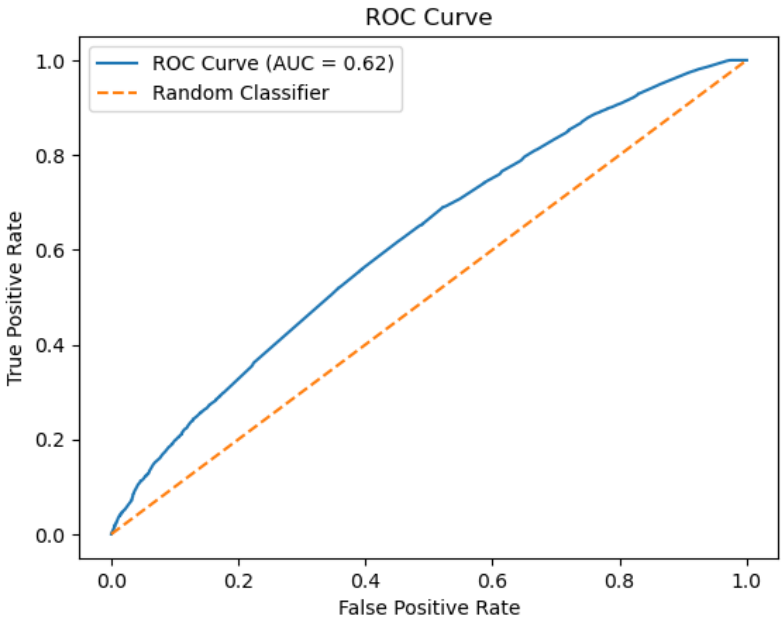
I then used **PCA** to try and visualize these clusters into 3D. Since there were **14 clusters**, it was still difficult to distinguish between them, but at least I was able to visualize them.



After this I was tasked to perform **nested cross-validation** to validate stability of my hyperparameters. For this I used **Random Forest Classifier**. I have split the data, made a new parameter grid but this time only using the values from previously used grid search to make the process faster. I only did **3 cross validations** inside the **GridSearch** because I wanted to make the process a bit faster again. After this I instantiated **K-fold with 3 splits** and using **cross_val_score** I calculated the performance of my model on unseen data. Having this implemented my nested cross validation for recall score was **0.18.**

Last part of this was Error Analysis. I will be focusing only on Classification error analysis, since it's the objective of this project.

First I ran the model again with the hyperparameters tuned and I calculated the mean recall score, which was 0.20. Looking at the confusion matrix, I can clearly see that there are many false negatives being predicted. Plotting the precision_recall_curve, I can observe which threshold could improve my score and model. Threshold of 0.3 has much higher recall but lower precision, which is something that would work for my objective. But rather than setting threshold, some other methods are appropriate to try, like oversampling or undersampling, class weighing and other.



3.6 Model Selection

For this last section I will be choosing only a classification model for my classification problem specifically. In section 3.2, I have decided to use Random Forest Classifier, because of its performance in all metrics without any feature engineering or hyperparameter tuning.

My first step for this last model is to clean the data again but with some changes than in previous sections. I drop duplicated columns based on **"patient_nbr",** make changes to columns to make them into numeric, but for gender difference I will not remap them, but use **OneHotEncoder** to make separate columns. I think it makes more sense to have a separate columns for both genders, rather than assigning 1 and 0 to them and making the model think that they actually have a weight. Next steps remained the same, removing outliers, remapping IDs and removing null values. Reducing the number of diagnosis based on **IDS-9 grouping**. Dropping columns which are not related or useless and changing the columns to integers.

I then split the data and first I train the model without hyperparameter tuning. Reminding that my focus should be on improving Recall to reduce the number of people missing treatment. This first model has a recall score of **0.40**. After plotting confusion matrix, I can see that it performs well on prediction not readmitted, but still has a high number of people who get put into not readmitted while in fact being readmitted. My next goal will be to reduce this with parameters.

My main issue is that the class is imbalanced. After researching different techniques for solving this, I have decided to go with class weighting. I make a dictionary with weights for 1:1.5 (readmitted) and 0:0.6 (not readmitted). I also put in the parameters from grid search in section 3.5. I will be scoring this on recall and performing 3 cross validations. Results from this were promising. My goal to reduce Recall was a success. I had a 0.75 on recall which reduced false

negatives from 2034 to 824. But this came with a cost. There was a big increase in people getting predicted as readmitted while in fact not needing to be readmitted.

Next I tried RandomizedSearch for best parameters with scoring on recall. I also included weights in the parameter grid, because I that will play big role in getting the model to work properly. Running this took around 50 minutes. I was able to get 92% recall, but other evaluation metrics went down, this got reflected when I plotted a confusion matrix of this model. Although I had only 242 false negatives, the number of false positives was at 3545. If the treatment or the medical process for these readmissions is not expensive, then it could work, but I think it's better if I try another approach.

I will run this RandomizedSearch again, but now I will focus on F1-score, which is the balance between false negatives and false positives. This way I can prevent people from not getting proper care, while also minimizing people who do not need to spend money and time in hospital.

After getting the parameters and training a new model, I was able to get F1 score of 0.62, which was apparently the highest. Plotting the confusion matrix of this model tells me that it got better at predicting true negative class while predicting less false positives. But when It is still not the result I was expecting.

Looking at the last models I trained, the one that I think is currently the best performing is the first model which had 0.75 recall score. Comparing all three confusion matrixes. This one had relatively low number of false negatives, while also being able to predict true negatives and true positives. It did have a higher number of false positives, but among the tree models it was the lowest number.

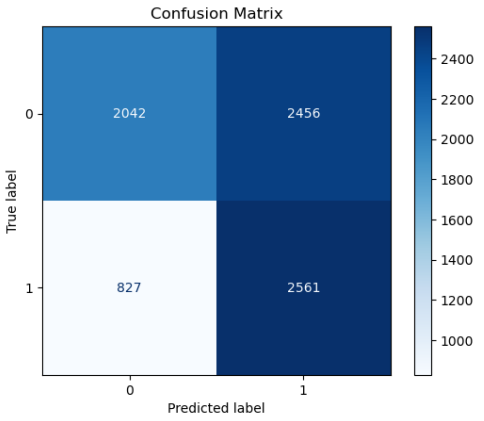


Figure 1 First model (Recall: 0.75)

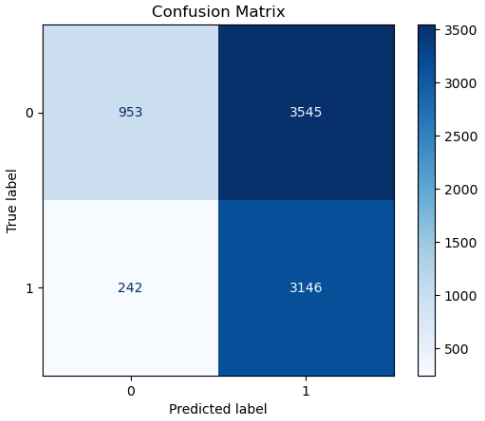


Figure 2 Second model (Recall: 0.92)

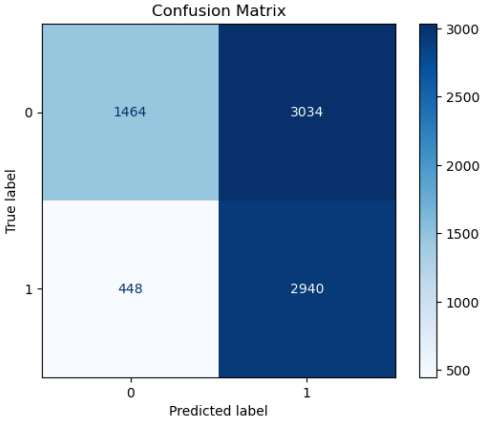


Figure 3 Third model (Recall: 0.86)

4. Conclusion and Reflection

Working for the first time on making machine learning models with this dataset was a challenge. I encountered various obstacles, from data preprocessing to model selection, but each hurdle provided valuable learning experiences. Ultimately, I gained a deeper understanding of the intricacies involved in developing effective machine learning solutions. I was able to understand how much effort it takes to make a decent dataset that can be used for training models. I experimented many times and understood that sometimes, even when something seems logical and should provide better results, it is not always the case. I managed to deal with an unbalanced target class and make a model that is somewhat usable in predicting whether patients will need to be readmitted or not. The model can still be further improved, but due to the time limit and my capabilities, this is as far as I could go. For the future, it would be wise to not only look at recall but also find a balance between all metrics and carefully analyse the impact of different techniques for handling the class imbalance. Additionally, exploring advanced algorithms or ensemble methods could enhance the model's performance and provide a more comprehensive evaluation of patient readmission risks. By taking these steps, we can

ensure a more robust model that not only predicts outcomes more accurately but also contributes to better decision-making in clinical settings. Ultimately, the goal is to implement solutions that are not only effective but also sustainable in real-world applications.

5. References

Liu VB, S. L. (2024, September 30). *Comparison of machine learning models for predicting 30-day readmission rates for patients with diabetes*. Retrieved from jmai.amegroups: https://jmai.amegroups.org/article/view/9179/html#table1

Zeglam. (2020). *Diabetes Patient Readmitted Prediction*. Retrieved from github: https://github.com/zeglam/Diabetes-Patient-Readmission-Classification/blob/master/Diabetes%20Patient%20Readmitted%20Prediction.ipynb