IFN 509: Data Exploration and Mining
Assessment 2

Team Name: Flyers
Group No. 71

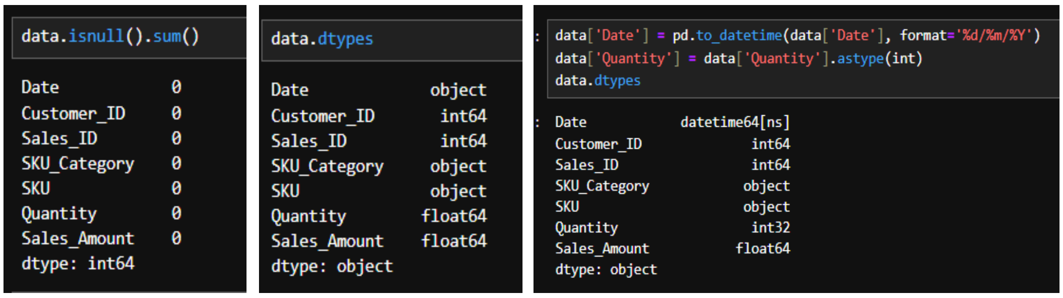
Student Name	**Student Id**
Chenghao Hou	N11567210
ChinLam Yuen	N10696393
Yonten Loday	N11828773

	Student 1	Student 2	Student 3
Student 1	60%	60%	60%
Student 2	70%	70%	70%
Student 3	100%	100%	100%

Project (a): Association mining to find common purchase patterns in a retail store sales data

TASK 1

For pre-processing, null values were checked first, and the data had no null values in all variables. We also ensured that all variables matched the data types described in the description, especially ensuring that the data column was in a consistent date-time format for accurate grouping (in the later analysis if needed). The Quantity column was changed to ***int*** from ***float*** as quantities are whole numbers and cannot be in decimals. This preprocessing is shown in Figure 1.

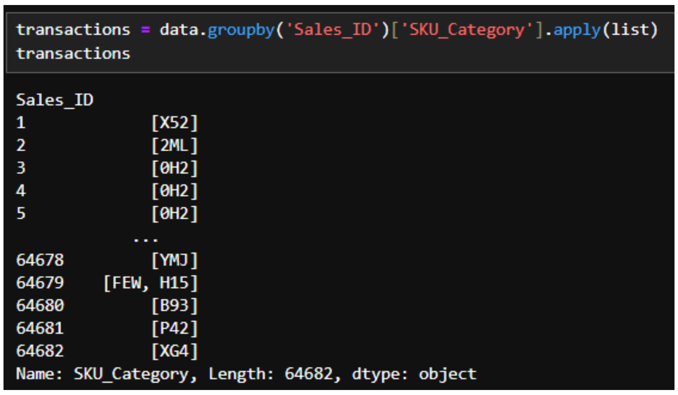


```
data.isnull().sum()

Date            0
Customer_ID     0
Sales_ID        0
SKU_Category    0
SKU             0
Quantity        0
Sales_Amount    0
dtype: int64
```

```
data.dtypes

Date             object
Customer_ID       int64
Sales_ID          int64
SKU_Category     object
SKU              object
Quantity        float64
Sales_Amount    float64
dtype: object
```

```
data['Date'] = pd.to_datetime(data['Date'], format='%d/%m/%Y')
data['Quantity'] = data['Quantity'].astype(int)
data.dtypes

Date            datetime64[ns]
Customer_ID              int64
Sales_ID                 int64
SKU_Category            object
SKU                     object
Quantity                 int32
Sales_Amount           float64
dtype: object
```

Figure 1 Preprocessing



```
transactions = data.groupby('Sales_ID')['SKU_Category'].apply(list)
transactions

Sales_ID
1              [X52]
2              [2ML]
3              [0H2]
4              [0H2]
5              [0H2]
            ...
64678          [YMJ]
64679     [FEW, H15]
64680          [B93]
64681          [P42]
64682          [XG4]
Name: SKU_Category, Length: 64682, dtype: object
```

Figure 2 Creating transaction list

Before building the association mining, transaction lists must be created. For that, the variables included, and the justifications are as follows:

1. Sales_ID: Used to group products into transactions. This variable is important to define transactions since each sale (transaction) can include multiple product categories. By grouping products by Sales_ID, we can analyze which categories often appear together in a single transaction.
2. SKU_Category: Represents the categories of products bought in each transaction. This is the main focus of the analysis to identify common purchase patterns among different product categories.

With the selected variables, a list of transactions was created for SKU_Category grouped by Sales_ID as shown in Figure 2.

TASK 2

a. Choosing min_support and min_confidence.

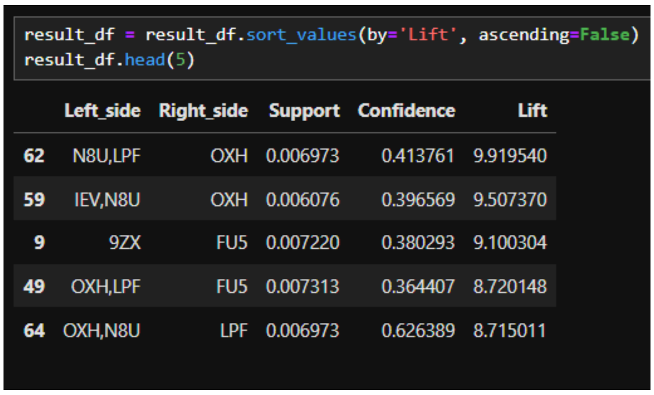
min_support: 0.005 (0.5%).

Setting minimum support to 0.005 ensured that we captured itemsets appearing in at least 0.5% of all transactions, which helped strike a balance between identifying frequent enough patterns and avoiding too many rare combinations. Initial minimum support was 5% and 1%, both generated only a few itemsets, limiting from meaningful patterns. Adjusting it to 0.005%, we identified 318 item sets.

min_confidence: 0.25 (25%).

This choice is a strategic decision to ensure we generate enough association rules for analysis. When set at 0.5 and 0.4, only 9 itemsets were generated, indicating that very few product combinations have a high enough confidence. So, we lowered the threshold to 0.25 which allowed us to uncover 65 itemsets while still maintaining a reasonable level of predictive power.

b. The itemsets in the data frame named result_df were sorted based on Lift values in descending order to get the top 5 rules as shown in Figure 3. Each rule is reported and interpreted as follows:



```
result_df = result_df.sort_values(by='Lift', ascending=False)
result_df.head(5)
```

	Left_side	Right_side	Support	Confidence	Lift
62	N8U,LPF	OXH	0.006973	0.413761	9.919540
59	IEV,N8U	OXH	0.006076	0.396569	9.507370
9	9ZX	FU5	0.007220	0.380293	9.100304
49	OXH,LPF	FU5	0.007313	0.364407	8.720148
64	OXH,N8U	LPF	0.006973	0.626389	8.715011

Figure 3 Top 5 rules based on Lift Values

1. Rule 1: N8U, LPF => OXH About 0.6973% (Support) of transactions contain this rule, and there are 41.4% (Confidence) chances that OXH will also be bought if N8U and LPF were already bought. The combination of ‘N8U’ and ‘LPF’ significantly increases the likelihood of ‘OXH’ being bought by approximately 9.92 (Lift) times, compared to buying OXH independently.

2. Rule 2: IEV, N8U => OXH
   About 0.6076% (Support) of transactions contain this rule, and there are 39.7% (Confidence) chances that OXH will also be bought if IEV and N8U were already bought. The combination of IEV and N8U significantly increases the likelihood of OXH being bought by approximately 9.51 (Lift) times, compared to buying OXH independently.

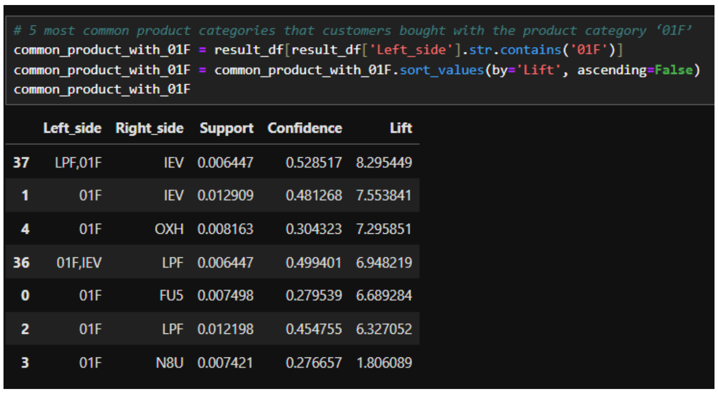
3. 9ZX => FU5
   About 0.722% (Support) of transactions contain this rule, and there are 38.0% (Confidence) chances that FU5 will also be bought if 9ZX was already bought. The purchase of 9ZX significantly increases the likelihood of FU5 being bought by approximately 9.1 (Lift) times, compared to buying FU5 independently.

4. Rule 4: OXH, LPF => FU5
   About 0.7313% (Support) of transactions contain this rule, and there are 36.4% (Confidence) chances that FU5 will also be bought if OXH and LPF were already bought. The combination of OXH and LPF significantly increases the likelihood of FU5 being bought by approximately 8.72 (Lift) times, compared to buying FU5 independently.

5. Rule 5: OXH, N8U => LPF
   About 0.6973% (Support) of transactions contain this rule, and there are 62.6% (Confidence) chances that LPF will also be bought if OXH and N8U were already bought. The combination of OXH and N8U significantly increases the likelihood of LPF being bought by approximately 8.72 (Lift) times, compared to buying LPF independently.

TASK 3

The top 5 common product categories that customers bought with product category 01F are *IEV, OXH, LPF, FU5, and N8U* as shown in Figure 5. We got these results by investigating transactions that contain '01F' on the Left_side.



```
# 5 most common product categories that customers bought with the product category '01F'
common_product_with_01F = result_df[result_df['Left_side'].str.contains('01F')]
common_product_with_01F = common_product_with_01F.sort_values(by='Lift', ascending=False)
common_product_with_01F
```

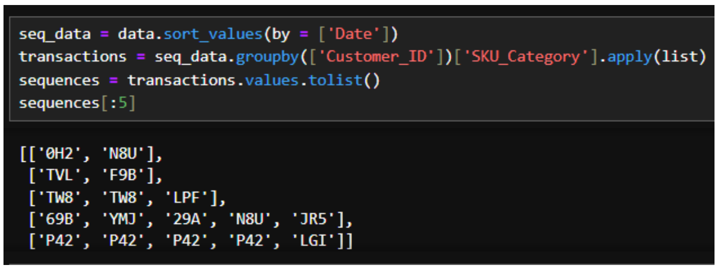
```
    Left_side  Right_side  Support   Confidence  Lift
37  LPF,01F    IEV         0.006447  0.528517    8.295449
1   01F        IEV         0.012909  0.481268    7.553841
4   01F        OXH         0.008163  0.304323    7.295851
36  01F,IEV    LPF         0.006447  0.499401    6.948219
0   01F        FU5         0.007498  0.279539    6.689284
2   01F        LPF         0.012198  0.454755    6.327052
3   01F        N8U         0.007421  0.276657    1.806089
```

Figure 4 Top 5 common product categories with product category 01F

TASK 4

Yes, we can perform sequence analysis on this dataset. We have date information and customer IDs. These two variables can be used together to track customer behaviour over time and identify the sequence of purchases across multiple transactions.

We sorted the dataset by date to make the purchase transaction in sequential order and then grouped the customers by Customer_ID to create sequential purchase transactions of SKU_Category for each customer as shown in Figure 5.

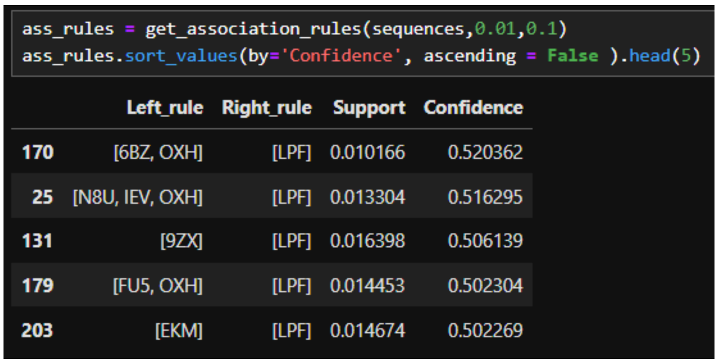


```
seq_data = data.sort_values(by = ['Date'])
transactions = seq_data.groupby(['Customer_ID'])['SKU_Category'].apply(list)
sequences = transactions.values.tolist()
sequences[:5]
```

```
[['0H2', 'N8U'],
 ['TVL', 'F9B'],
 ['TW8', 'TW8', 'LPF'],
 ['69B', 'YMJ', '29A', 'N8U', 'JR5'],
 ['P42', 'P42', 'P42', 'P42', 'LGI']]
```

Figure 5 Sequential transactions of SKU_Category for each customer

Figure 6 shows the sequential rules based on confidence (due to space constraints, the code snippet for get_association_rules() is omitted).



```
ass_rules = get_association_rules(sequences,0.01,0.1)
ass_rules.sort_values(by='Confidence', ascending = False ).head(5)
```

```
           Left_rule  Right_rule  Support   Confidence
170       [6BZ, OXH]       [LPF]  0.010166    0.520362
25   [N8U, IEV, OXH]       [LPF]  0.013304    0.516295
131            [9ZX]       [LPF]  0.016398    0.506139
179       [FU5, OXH]       [LPF]  0.014453    0.502304
203            [EKM]       [LPF]  0.014674    0.502269
```

Figure 6 Top 5 Sequential Rules Based on Confidence

TASK 5

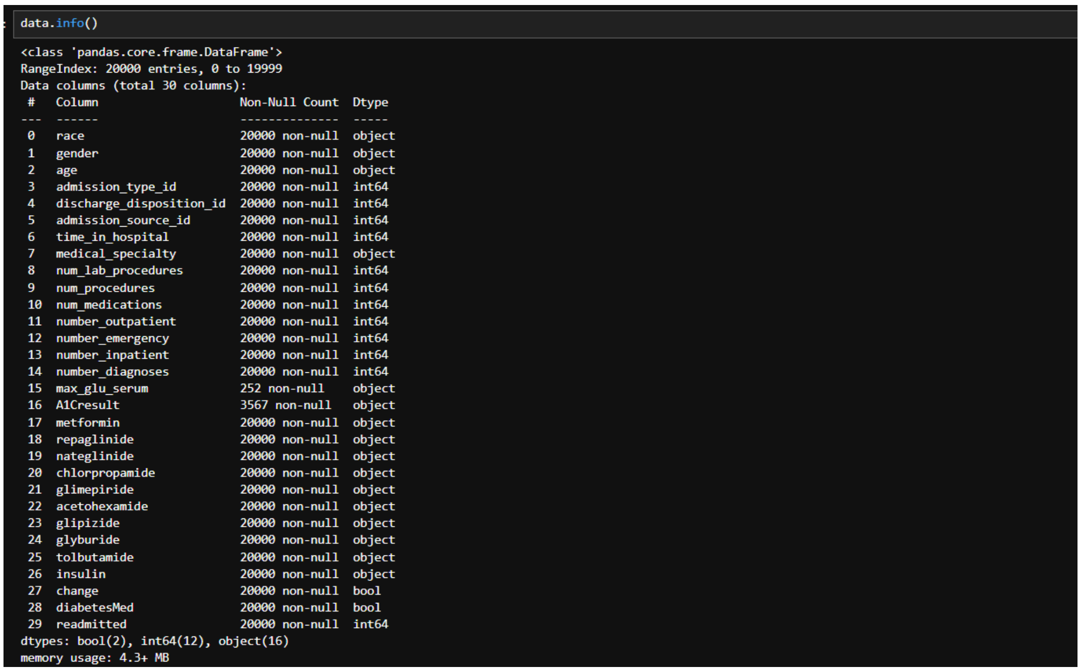
The outcomes of the association mining and sequence analysis can provide valuable insights for decision-makers in the retail store:

1. Product Bundling and Promotions: Decision-makers can use the identified product associations to create targeted product bundles and promotional offers, increasing sales by encouraging customers to purchase related items together.
2. Inventory Management: Insights from this study can guide inventory planning by ensuring that commonly co-purchased product categories are adequately stocked, reducing the chances of stockouts and improving customer satisfaction.
3. Personalized Marketing: The findings can be utilized to develop personalized marketing campaigns, such as recommending related products to customers based on their purchasing behaviour, enhancing the customer shopping experience and driving repeat business.

Project (b): Clustering Diabetes data

TASK 1

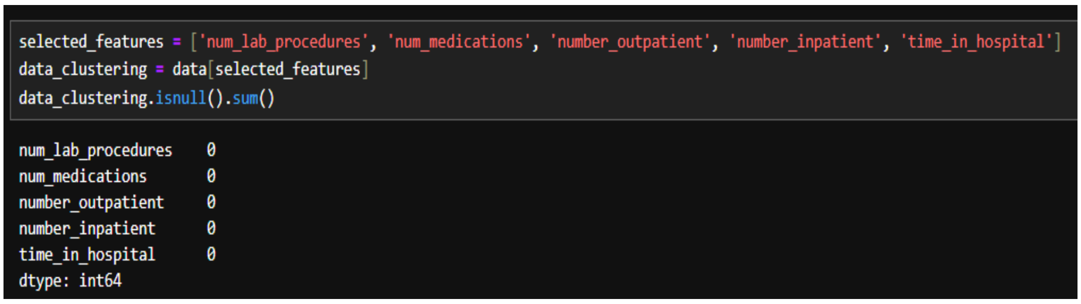
Before the clustering model, we checked the data types, missing values, multicollinearity, standardizing/scaling, and feature selection. For the second clustering model, we encoded the age variables. This preprocessing ensured the dataset was clean, consistent, and suitable for the clustering algorithm. The preprocessing processes are shown in Figure 7-11.



```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20000 entries, 0 to 19999
Data columns (total 30 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   race                      20000 non-null  object
 1   gender                    20000 non-null  object
 2   age                       20000 non-null  object
 3   admission_type_id         20000 non-null  int64
 4   discharge_disposition_id  20000 non-null  int64
 5   admission_source_id       20000 non-null  int64
 6   time_in_hospital          20000 non-null  int64
 7   medical_specialty         20000 non-null  object
 8   num_lab_procedures        20000 non-null  int64
 9   num_procedures            20000 non-null  int64
 10  num_medications           20000 non-null  int64
 11  number_outpatient         20000 non-null  int64
 12  number_emergency          20000 non-null  int64
 13  number_inpatient          20000 non-null  int64
 14  number_diagnoses          20000 non-null  int64
 15  max_glu_serum             252 non-null    object
 16  A1Cresult                 3567 non-null   object
 17  metformin                 20000 non-null  object
 18  repaglinide               20000 non-null  object
 19  nateglinide               20000 non-null  object
 20  chlorpropamide            20000 non-null  object
 21  glimepiride               20000 non-null  object
 22  acetohexamide             20000 non-null  object
 23  glipizide                 20000 non-null  object
 24  glyburide                 20000 non-null  object
 25  tolbutamide               20000 non-null  object
 26  insulin                   20000 non-null  object
 27  change                    20000 non-null  bool
 28  diabetesMed               20000 non-null  bool
 29  readmitted                20000 non-null  int64
dtypes: bool(2), int64(12), object(16)
memory usage: 4.3+ MB
```

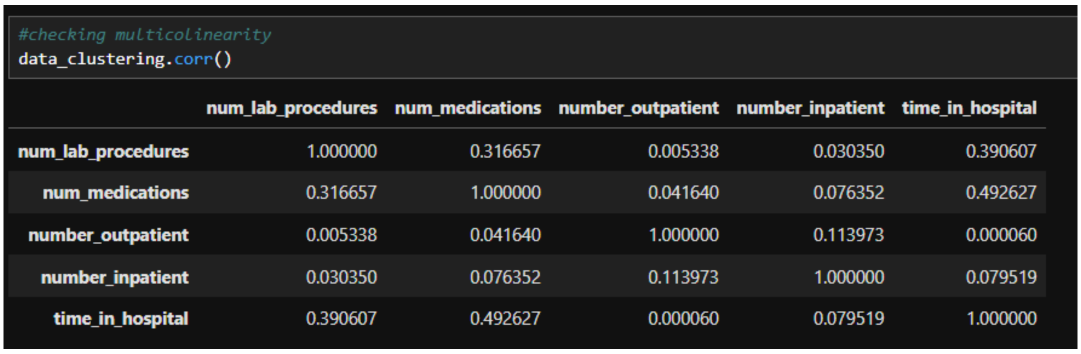
Figure 7 Overview of the dataset



```
selected_features = ['num_lab_procedures', 'num_medications', 'number_outpatient', 'number_inpatient', 'time_in_hospital']
data_clustering = data[selected_features]
data_clustering.isnull().sum()
```

```
num_lab_procedures    0
num_medications       0
number_outpatient     0
number_inpatient      0
time_in_hospital      0
dtype: int64
```

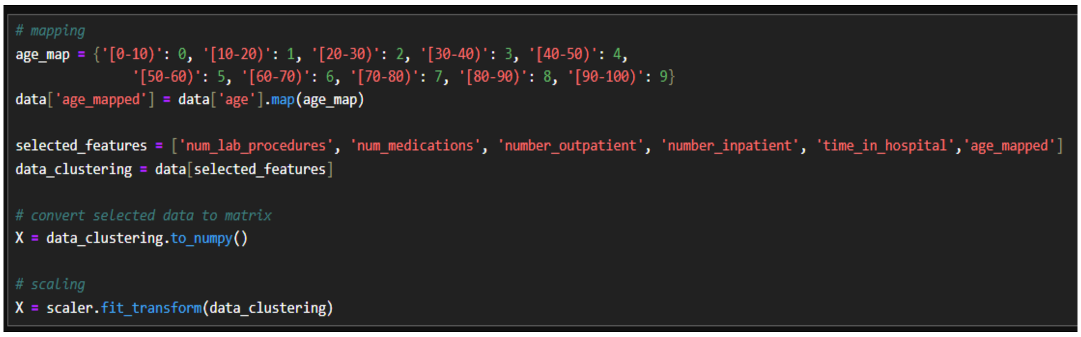
Figure 8 Feature selection and null values



```
#checking multicolinearity
data_clustering.corr()
```

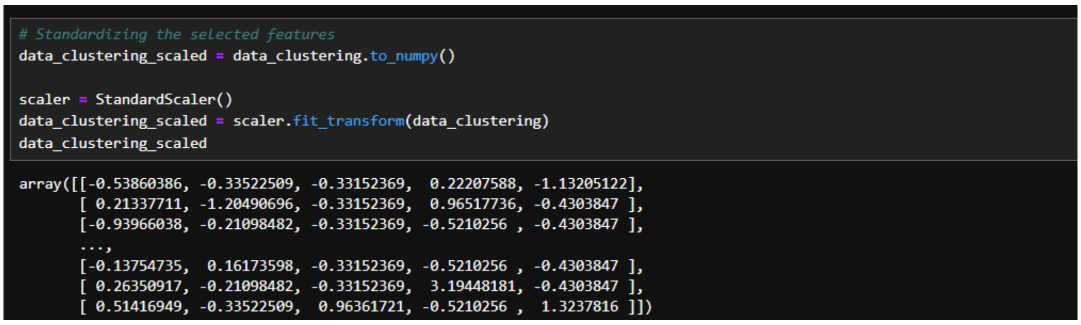
	num_lab_procedures	num_medications	number_outpatient	number_inpatient	time_in_hospital
num_lab_procedures	1.000000	0.316657	0.005338	0.030350	0.390607
num_medications	0.316657	1.000000	0.041640	0.076352	0.492627
number_outpatient	0.005338	0.041640	1.000000	0.113973	0.000060
number_inpatient	0.030350	0.076352	0.113973	1.000000	0.079519
time_in_hospital	0.390607	0.492627	0.000060	0.079519	1.000000

Figure 9 Checking multicollinearity



```
# mapping
age_map = {'[0-10)': 0, '[10-20)': 1, '[20-30)': 2, '[30-40)': 3, '[40-50)': 4,
               '[50-60)': 5, '[60-70)': 6, '[70-80)': 7, '[80-90)': 8, '[90-100)': 9}
data['age_mapped'] = data['age'].map(age_map)

selected_features = ['num_lab_procedures', 'num_medications', 'number_outpatient', 'number_inpatient', 'time_in_hospital','age_mapped']
data_clustering = data[selected_features]

# convert selected data to matrix
X = data_clustering.to_numpy()

# scaling
X = scaler.fit_transform(data_clustering)
```

Figure 10 Age encoding



```
# Standardizing the selected features
data_clustering_scaled = data_clustering.to_numpy()

scaler = StandardScaler()
data_clustering_scaled = scaler.fit_transform(data_clustering)
data_clustering_scaled
```

```
array([[-0.53860386, -0.33522509, -0.33152369,  0.22207588, -1.13205122],
       [ 0.21337711, -1.20490696, -0.33152369,  0.96517736, -0.4303847 ],
       [-0.93966038, -0.21098482, -0.33152369, -0.5210256 , -0.4303847 ],
       ...,
       [-0.13754735,  0.16173598, -0.33152369, -0.5210256 , -0.4303847 ],
       [ 0.26350917, -0.21098482, -0.33152369,  3.19448181, -0.4303847 ],
       [ 0.51416949, -0.33522509,  0.96361721, -0.5210256 ,  1.3237816 ]])
```

Figure 11 Data scaling

TASK 2

a. The k-means clustering algorithm is used for this task because K-means is a fast, scalable, and easy-to-interpret algorithm that works well with large, numeric datasets, making it a better fit than other methods for this task.

b. The attributes that are important in profiling the diabetes patients in this analysis are: 'num_lab_procedures', 'num_medications', 'number_outpatient', 'number_inpatient', 'time_in_hospital'.

c. To get an optimal number of clusters, we first tried to make a random guess which was not a good approach as there is no business clue to guess appropriate clusters. We skipped that approach and implemented the "elbow method" shown in Figure 12.

From the plot, the elbow appears around K = 3. This is where the rate of reduction in inertia starts to slow down. Beyond this point (K > 3), adding more clusters doesn't significantly reduce the inertia, suggesting that three clusters provide a good balance between simplicity and capturing the underlying structure of the data. However, 4 and 5 also seem to be a good representation of an optimal number of clusters too. Therefore, silhouette_score was performed as shown in Figure 13. Using the silhouette score, we confirmed that the optimal K value (number of clusters) is 3 because it best fits the data with the *highest silhouette score*.

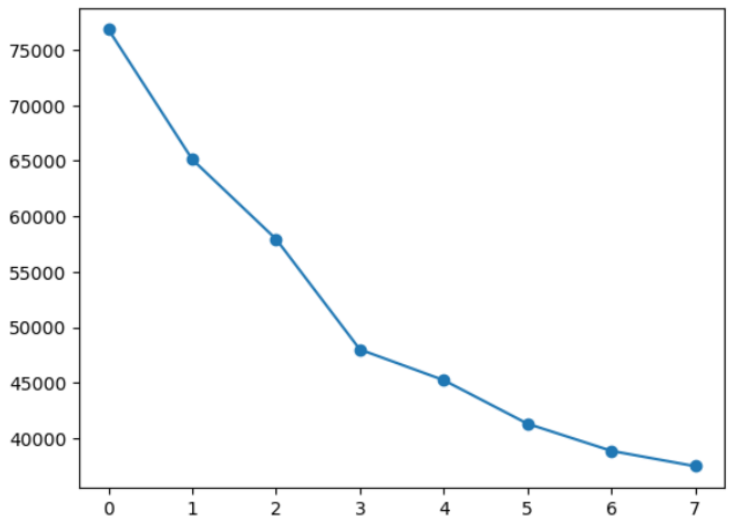
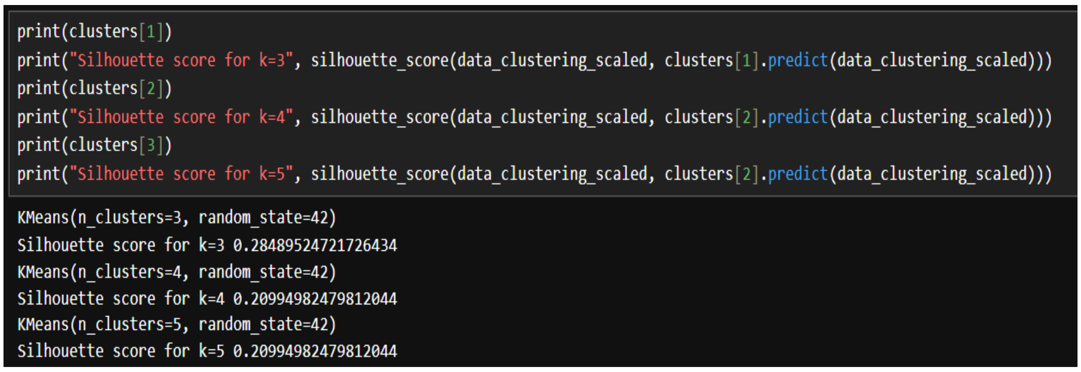


Figure 12 Elbow method to determine no. of clusters



```
print(clusters[1])
print("Silhouette score for k=3", silhouette_score(data_clustering_scaled, clusters[1].predict(data_clustering_scaled)))
print(clusters[2])
print("Silhouette score for k=4", silhouette_score(data_clustering_scaled, clusters[2].predict(data_clustering_scaled)))
print(clusters[3])
print("Silhouette score for k=5", silhouette_score(data_clustering_scaled, clusters[2].predict(data_clustering_scaled)))
```

```
KMeans(n_clusters=3, random_state=42)
Silhouette score for k=3 0.28489524721726434
KMeans(n_clusters=4, random_state=42)
Silhouette score for k=4 0.20994982479812044
KMeans(n_clusters=5, random_state=42)
Silhouette score for k=5 0.20994982479812044
```

Figure 13 Silhouette Score

d.

Centroid 0: [0.71072392, 0.85765468, -0.13186751, -0.14712594, 1.02026448]

- num_lab_procedures: 0.71 (above average)
- num_medications: 0.86 (above average)
- number_outpatient: -0.13 (slightly below average)
- number_inpatient: -0.15 (slightly below average)
- time_in_hospital: 1.02 (above average)

Interpretation: Cluster 0 represents patients who have above-average lab procedures, above-average medications, and above-average time in the hospital, but with slightly fewer outpatient and inpatient visits. The patients in this cluster need more extensive monitoring and treatment during a single hospital stay but tend to have fewer outpatient or inpatient follow-ups.

Centroid 1: [-0.38038856, -0.46561967, -0.14447548, -0.25558336, -0.53166282]

- num_lab_procedures: -0.38 (below average)
- num_medications: -0.47 (below average)
- number_outpatient: -0.14 (slightly below average)
- number_inpatient: -0.26 (below average)
- time_in_hospital: -0.53 (below average)

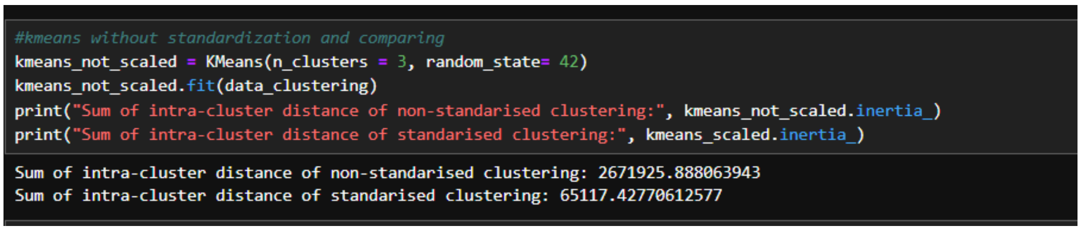
Interpretation: This cluster represents patients who are below average in all aspects: fewer lab procedures, fewer medications, fewer hospital visits (both outpatient and inpatient), and shorter hospital stays. This group possibly represents the healthier patients or those with less severe cases, requiring fewer treatments, and procedures, and overall, less interaction with the healthcare system.

Centroid 2: [0.08594033, 0.14561118, 1.35112612, 2.10749826, 0.03185678]

- num_lab_procedures: 0.09 (slightly above average)
- num_medications: 0.15 (slightly above average)
- number_outpatient: 1.35 (well above average)
- number_inpatient: 2.11 (well above average)
- time_in_hospital: 0.03 (average)

Interpretation: This cluster represents patients with very high outpatient and inpatient visits, with slightly above-average lab procedures and medications, and a typical hospital stay. This group consists of patients requiring frequent follow-up care outside of hospital stays, possibly those with chronic conditions or patients needing regular check-ups and inpatient services.

e. Yes, we did standardise the variables with *StandardScaler()* as shown in Figure 11. The sum of intra-cluster distances (inertia) for the standardized clustering is significantly lower (65117.43) compared to the non-standardized clustering (2671925.89) as shown in Figure 14, indicating that standardizing the data greatly improves the compactness of clusters in K-Means. By standardizing the variables, the clustering algorithm could better form cohesive clusters, avoiding distortion caused by differences in the scale of features. Thus, standardization was essential for achieving more compact and meaningful clusters.



```
#kmeans without standardization and comparing
kmeans_not_scaled = KMeans(n_clusters = 3, random_state= 42)
kmeans_not_scaled.fit(data_clustering)
print("Sum of intra-cluster distance of non-standarised clustering:", kmeans_not_scaled.inertia_)
print("Sum of intra-cluster distance of standarised clustering:", kmeans_scaled.inertia_)
```

```
Sum of intra-cluster distance of non-standarised clustering: 2671925.888063943
Sum of intra-cluster distance of standarised clustering: 65117.42770612577
```

Figure 14 Intra-cluster distance Comparison

TASK 3

a. Interpretation using the K-means cluster pair plot shown in Figure 15:

- Cluster 0 (green) tends to dominate the higher ranges for most medical features, which can indicate that these patients are more medically intensive in terms of treatments and hospital resources.
- Cluster 1 (orange) stays in the lower ranges, indicating less medical intervention and shorter hospital stays.
- Cluster 2 (blue) occupies the middle ground and shows moderate medical needs, neither as resource-intensive as Cluster 0 nor as low as Cluster 1.

This plot provides an overview of the medical usage patterns for each cluster. Cluster 0 represents the high-resource group (more procedures, medications, and hospital time), Cluster 1 shows the low-resource group (less medical intervention), and Cluster 2 falls in the middle, with moderate medical usage. These insights can be useful for designing patient-specific care pathways based on their medical resource requirements.

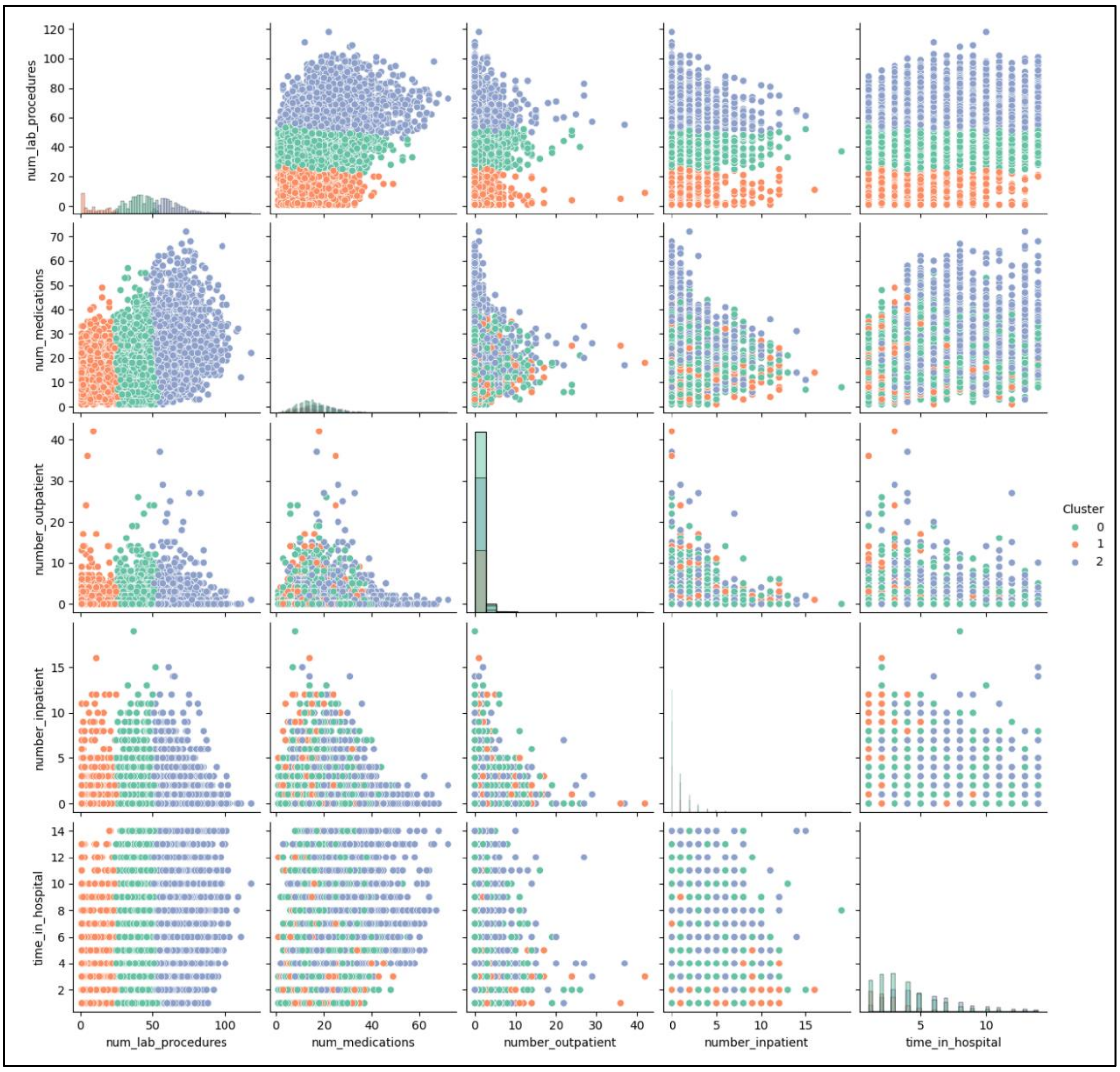


Figure 15 K-means clusters Pair plot

b.
Cluster 0: “Low Resource Users”
This cluster is characterized by patients who require a high number of medical procedures and medications. These patients typically have longer hospital stays and more inpatient visits. They likely have complex or severe health conditions, requiring extensive medical intervention.

Cluster 1: “Low Resource Users”
Patients in this cluster require minimal medical interventions. They have fewer lab procedures and medications, with shorter hospital stays and fewer inpatient and outpatient visits. These patients likely have less severe health conditions or are well-managed diabetics requiring routine care.

Cluster 2: “Moderate Resource Users”
This cluster is characterized by patients who fall between the other two clusters. They require a moderate number of medical procedures and medications, with moderate hospital stays and outpatient visits. These patients likely have moderately severe health conditions that require regular medical attention, but are not as intensive as Cluster 0.

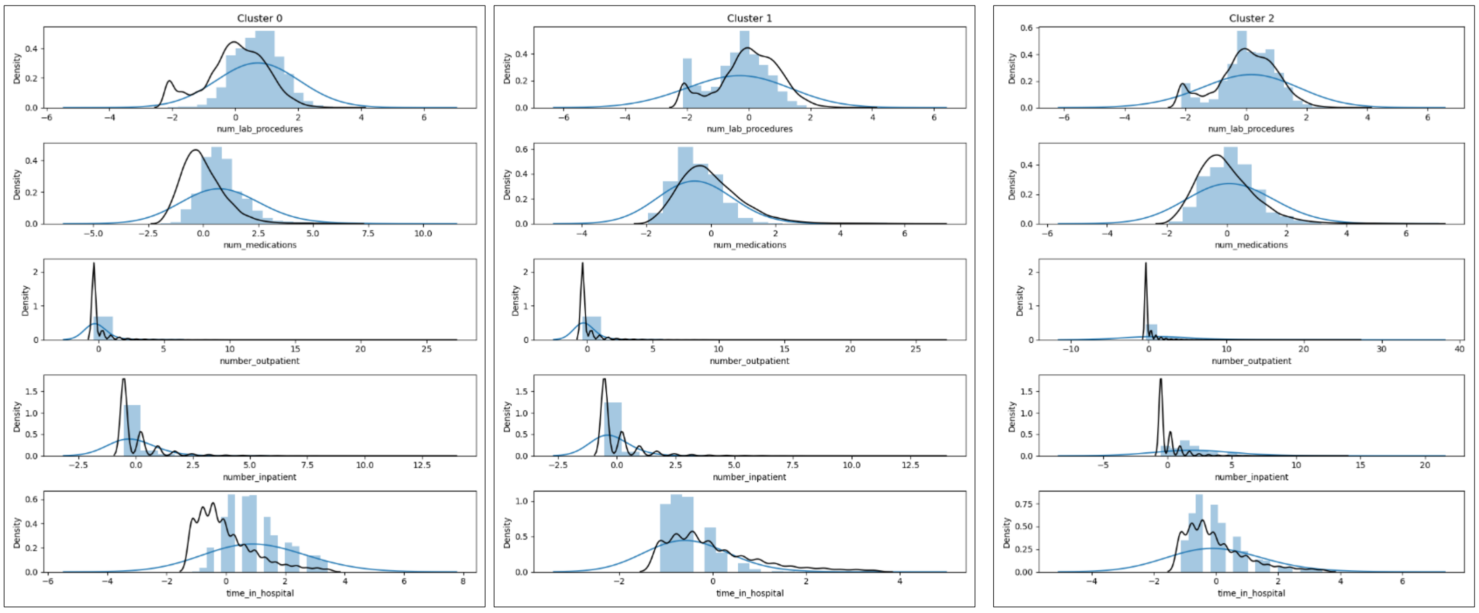
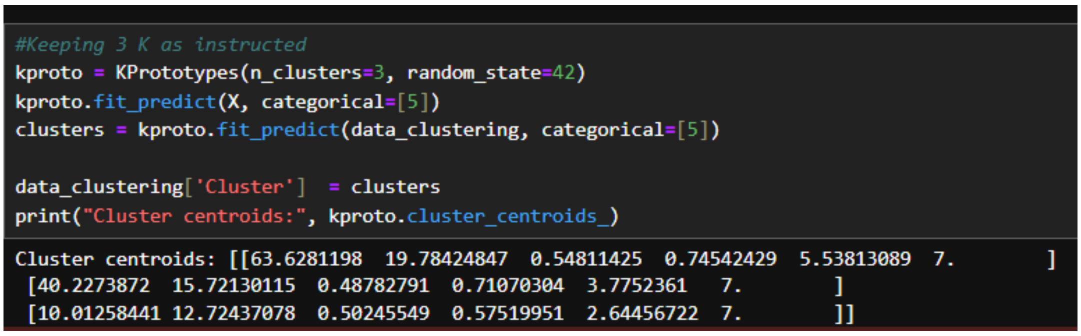


Figure 16 Comparison between K-means clusters

TASK 4

a. The K-prototypes algorithm is chosen for this task because it is well-suited when you have mixed data types, i.e., both numerical and categorical variables. We are including age, which is in categorical format in this dataset, we can not use K means.

Keeping the best settings from K- means clustering (standardised variables and optimal K value of 3), we built K-prototypes clustering on the dataset with the inclusion of age variable as shown in Figure 17.



```
#Keeping 3 K as instructed
kproto = KPrototypes(n_clusters=3, random_state=42)
kproto.fit_predict(X, categorical=[5])
clusters = kproto.fit_predict(data_clustering, categorical=[5])

data_clustering['Cluster']  = clusters
print("Cluster centroids:", kproto.cluster_centroids_)
```

```
Cluster centroids: [[63.6281198  19.78424847  0.54811425  0.74542429  5.53813089  7.        ]
 [40.2273872  15.72130115  0.48782791  0.71070304  3.7752361   7.        ]
 [10.01258441 12.72437078  0.50245549  0.57519951  2.64456722  7.        ]]
```

Figure 17 K-Prototypes clustering

b. In the K-Means clustering, patients were grouped solely based on medical intensity and in K-Prototypes, the inclusion of age *changed the composition of clusters, showing clearer clusters with age in it* as shown in Figure 18.

- **Cluster 0 ("Older, High-Resource Users"):** Older patients with high medical needs (frequent procedures, medications, and long hospital stays). Represents more complex cases requiring intensive intervention.
- **Cluster 1 ("Younger, Low-Resource Users"):** Younger patients with minimal medical needs (fewer procedures, medications, and shorter hospital stays). Represents healthier or well-managed diabetic patients.
- **Cluster 2 ("Middle-Aged, Moderate-Resource Users"):** Middle-aged patients with moderate medical needs (some procedures, medications, and hospital visits). Represents patients with manageable conditions requiring regular but not extensive medical care.

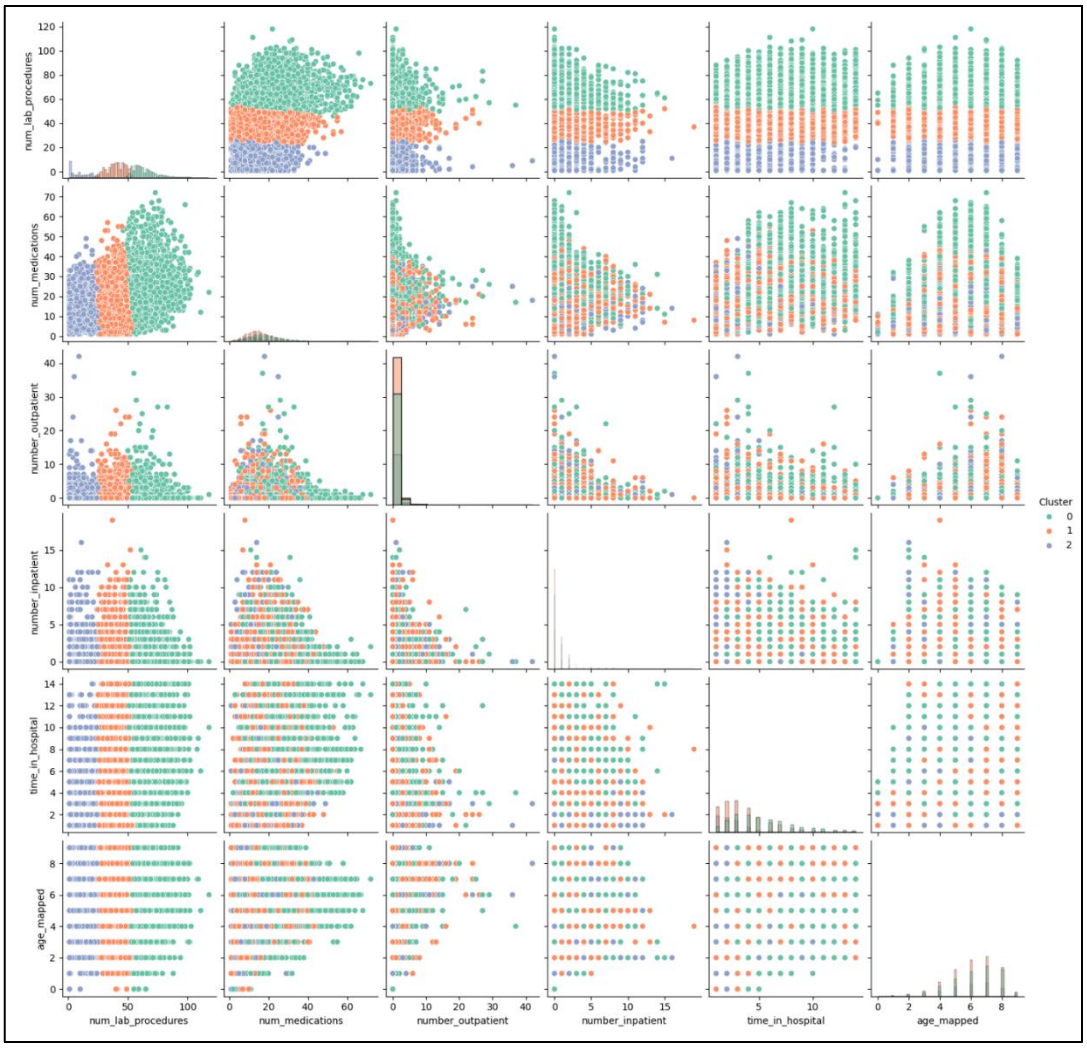


Figure 18 K-prototype clusters Pair plot

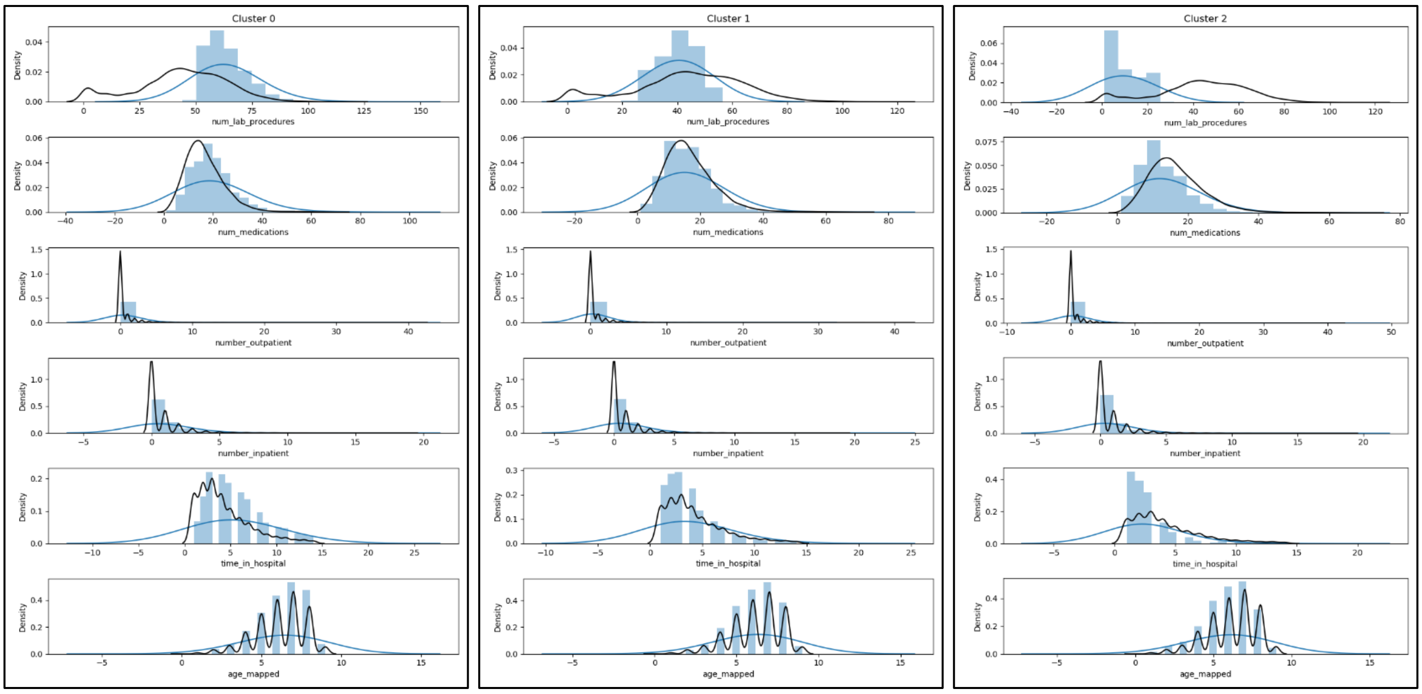


Figure 19 Comparison between K-prototype clusters

TASK 5

This study provides several actionable insights for the relevant decision-makers. We listed the three major outcomes the decision-makers can use:

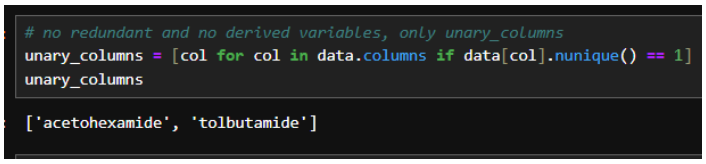
1. Efficient Resource Allocation
   Healthcare facilities can use these cluster profiles to optimize the allocation of medical resources. For example, clusters with high-resource users can prompt healthcare providers to prioritize staffing and resource availability for more intensive care units. Higher-resource clusters will require more funding for inpatient care and procedures, while lower-resource clusters may focus on outpatient services and routine checkups.
2. Improving Healthcare Efficiency
   Understanding the high-risk clusters (patients with frequent inpatient visits) can help hospitals design strategies to reduce hospital readmissions, improving both patient outcomes and healthcare efficiency.
   By identifying patient clusters with varying needs, healthcare providers can streamline operations, ensuring that the right care is delivered to the right group at the right time, reducing unnecessary costs and improving efficiency.
3. Strategic Planning and Capacity Management
   Facilities can forecast the demand for inpatient services, outpatient visits, and laboratory resources, adjusting their capacity accordingly.
   This can also guide the hiring and training of healthcare professionals, such as diabetologists, nurse specialists, and nutritionists, based on the needs of the different patient groups.

Project (c): Building and Evaluating Predictive Models

Predictive modelling using Decision Tree

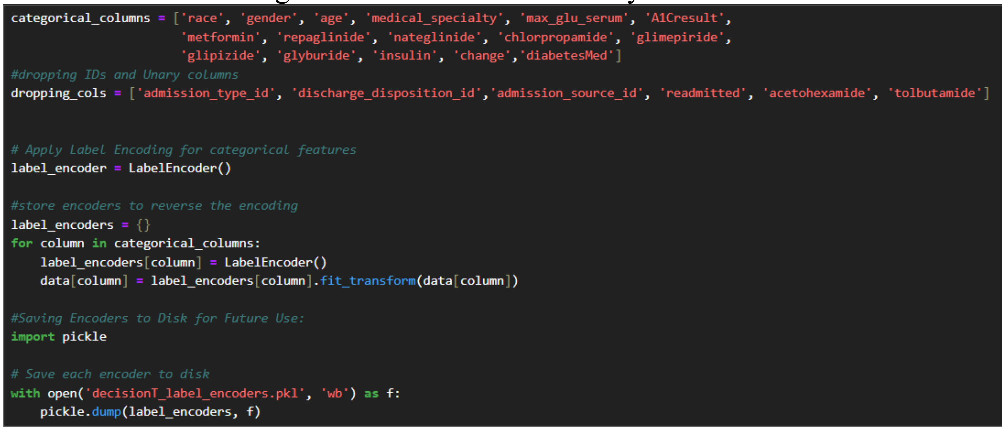
TASK 1

Before the decision tree modelling:
1. We checked the data types and missing values of the variables shown in Figure 7.
2. There are no redundant variables and derived variables in this dataset. Looked for unary variables shown in Figure 20. Encoded the categorical variables and saved those encodings using the pickle library to inverse the encodings to the original form to determine the characteristics of patients shown in Figure 21. Dropped ID variables and unary variables from the modelling dataset.



```
# no redundant and no derived variables, only unary_columns
unary_columns = [col for col in data.columns if data[col].nunique() == 1]
unary_columns

['acetohexamide', 'tolbutamide']
```

Figure 20 Identification of Unary variables



```
categorical_columns = ['race', 'gender', 'age', 'medical_specialty', 'max_glu_serum', 'A1Cresult',
                       'metformin', 'repaglinide', 'nateglinide', 'chlorpropamide', 'glimepiride',
                       'glipizide', 'glyburide', 'insulin', 'change','diabetesMed']
#dropping IDs and Unary columns
dropping_cols = ['admission_type_id', 'discharge_disposition_id','admission_source_id', 'readmitted', 'acetohexamide', 'tolbutamide']


# Apply Label Encoding for categorical features
label_encoder = LabelEncoder()

#store encoders to reverse the encoding
label_encoders = {}
for column in categorical_columns:
    label_encoders[column] = LabelEncoder()
    data[column] = label_encoders[column].fit_transform(data[column])

#Saving Encoders to Disk for Future Use:
import pickle

# Save each encoder to disk
with open('decisionT_label_encoders.pkl', 'wb') as f:
    pickle.dump(label_encoders, f)
```

Figure 21 Encoding and saving with Pickle

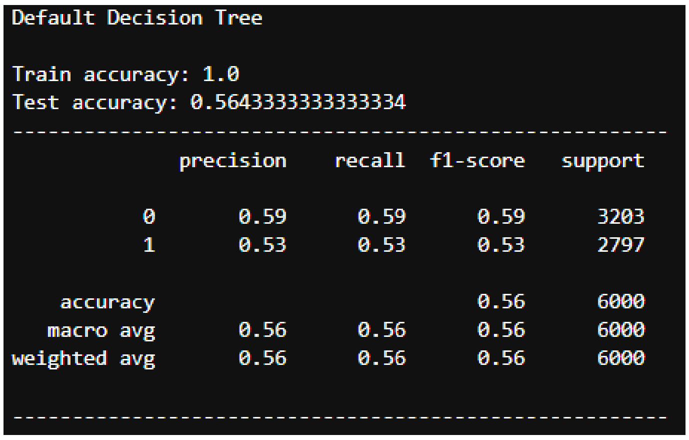
TASK 2

a. Classification accuracy of training and test datasets:

Train Accuracy: 100%

Test Accuracy: 56.43%

The model perfectly fits the training data, and this indicates *overfitting*.

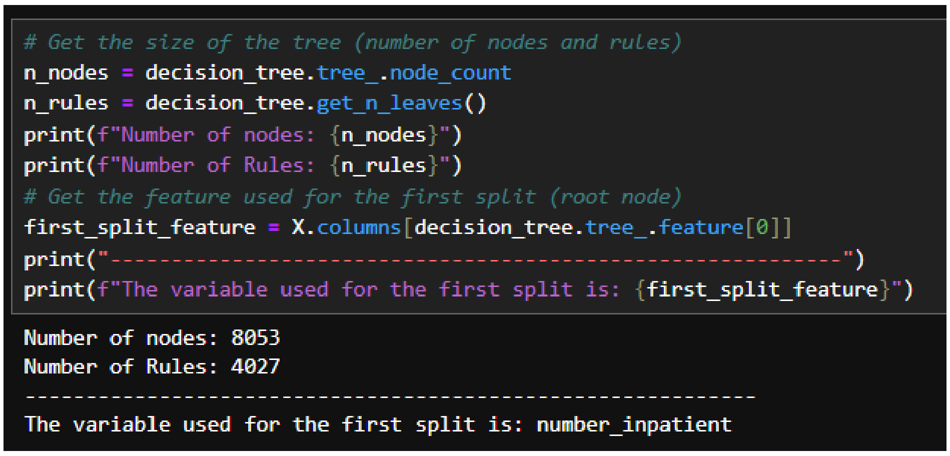


```
Default Decision Tree

Train accuracy: 1.0
Test accuracy: 0.5643333333333334
-------------------------------------------------------
              precision    recall  f1-score   support

           0       0.59      0.59      0.59      3203
           1       0.53      0.53      0.53      2797

    accuracy                           0.56      6000
   macro avg       0.56      0.56      0.56      6000
weighted avg       0.56      0.56      0.56      6000

-------------------------------------------------------
```

Figure 22 Default Decision Tree and its Accuracy

b. Number of nodes: 8053

Number of Rules: 4027



```python
# Get the size of the tree (number of nodes and rules)
n_nodes = decision_tree.tree_.node_count
n_rules = decision_tree.get_n_leaves()
print(f"Number of nodes: {n_nodes}")
print(f"Number of Rules: {n_rules}")
# Get the feature used for the first split (root node)
first_split_feature = X.columns[decision_tree.tree_.feature[0]]
print("--------------------------------------------------------------")
print(f"The variable used for the first split is: {first_split_feature}")
```

```
Number of nodes: 8053
Number of Rules: 4027
--------------------------------------------------------------
The variable used for the first split is: number_inpatient
```

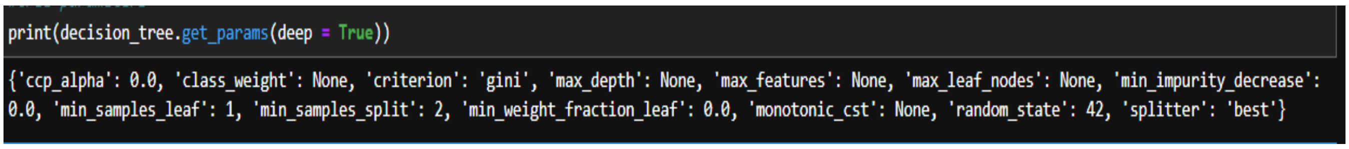
Figure 23 Size and first variable to split of default tree

c. First variable used for splitting: number_inpatient.

d. Five important variables in building decision tree, in order:

num_lab_procedures : 0.1919
num_medications : 0.148
time_in_hospital : 0.095
age : 0.0696
number_inpatient : 0.0675

e.



```python
print(decision_tree.get_params(deep = True))
```

```
{'ccp_alpha': 0.0, 'class_weight': None, 'criterion': 'gini', 'max_depth': None, 'max_features': None, 'max_leaf_nodes': None, 'min_impurity_decrease':
0.0, 'min_samples_leaf': 1, 'min_samples_split': 2, 'min_weight_fraction_leaf': 0.0, 'monotonic_cst': None, 'random_state': 42, 'splitter': 'best'}
```

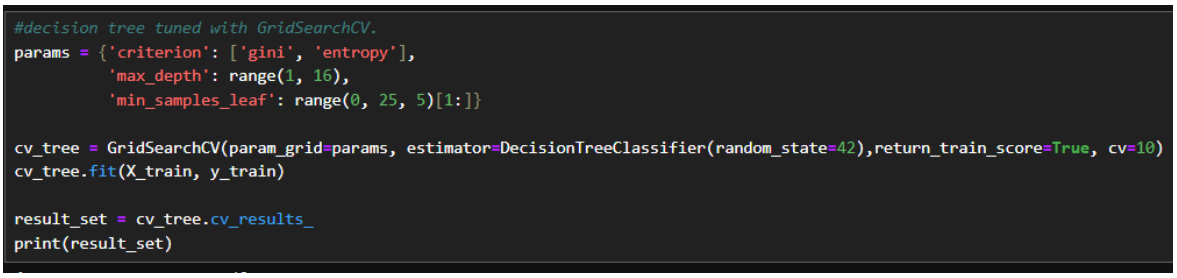
Figure 24 Decision Tree Parameters

Details:

- ccp_alpha: 0.0 – No Minimal Cost-Complexity Pruning applied.
- class_weight: None – No class weights applied; Imbalanced classes not handled.
- criterion: 'gini' – Gini impurity is used to measure the quality of splits.
- max_depth: None – No maximum depth is set, allowing the tree to grow until all leaves are pure or contain fewer than the minimum samples.
- max_features: None – All features are considered when searching for the best split.
- max_leaf_nodes: None – No maximum number of leaf nodes is set; the tree can grow without this restriction.
- min_impurity_decrease: 0.0 – No minimum impurity decrease is required to split a node.
- min_samples_leaf: 1 – A leaf node can be created with as few as one sample.
- min_samples_split: 2 – A node can be split if it contains at least two samples.
- min_weight_fraction_leaf: 0.0 – No minimum weighted fraction of the sum of weights is required at a leaf node.
- monotonic_cst: None – No monotonic constraints applied to the splits.
- random_state: 42 – Controls randomness for reproducibility across different runs.
- splitter: 'best' – The best split is chosen based on the Gini criterion.

TASK 3

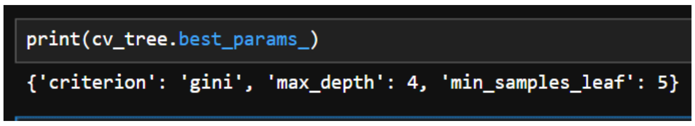
Figure 25 shows the tuning of the Decision Tree with GridSearchCV, focusing on three hyperparameters - criterion, max_length, and min_samples_leaf.



```
#decision tree tuned with GridSearchCV.
params = {'criterion': ['gini', 'entropy'],
          'max_depth': range(1, 16),
          'min_samples_leaf': range(0, 25, 5)[1:]}

cv_tree = GridSearchCV(param_grid=params, estimator=DecisionTreeClassifier(random_state=42),return_train_score=True, cv=10)
cv_tree.fit(X_train, y_train)

result_set = cv_tree.cv_results_
print(result_set)
```

Figure 25 Tuning Decision tree with GridSearchCV

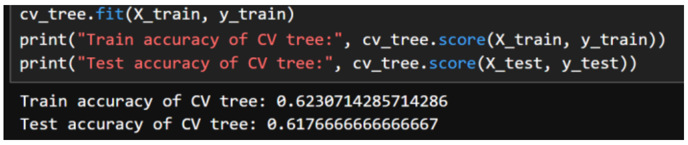
We have identified the best values for the three hyperparameters for the model. Figure 26 shows how the model overfits when the depth of the tree is increased because increasing the size of the tree depth will increase the variance. The model fits the training data too well, including noise, and thus performs poorly for the test data.



```
print(cv_tree.best_params_)

{'criterion': 'gini', 'max_depth': 4, 'min_samples_leaf': 5}
```

Figure 26 Best values for the hyperparameters

a. As shown in Figure 27, the training accuracy of the decision tree tuned with GridSearchCV is 62.31% and the test accuracy is 61.77%. Looking at the tree, the issue of overfitting in the default decision tree without any tuning has been solved with improvement in test accuracy and the minimal difference between the two accuracies.



```
cv_tree.fit(X_train, y_train)
print("Train accuracy of CV tree:", cv_tree.score(X_train, y_train))
print("Test accuracy of CV tree:", cv_tree.score(X_test, y_test))

Train accuracy of CV tree: 0.6230714285714286
Test accuracy of CV tree: 0.6176666666666667
```

Figure 27 Classification Accuracies of GridSearchCV

b. The number of nodes and rules has decreased significantly with the tuning from GridSearchCV. The number of nodes and rules is 31 and 16, respectively.

c. The variable used for the first split is still the number_inpatient. This indicates that the number of inpatient encounters is the most important feature for distinguishing between patients who are likely to be readmitted and those who are not.

d. The feature importance for each variable is retrieved from the decision tree (tuned with GridSearchCV) using the feature_importances_ attribute. The top 5 important variables in the tuned decision tree are:

1. number_inpatient: 0.7663
2. number_outpatient: 0.0628
3. num_medications: 0.0515
4. number_emergency: 0.0513
5. age: 0.0289

This shows that the model focuses heavily on the number of inpatient encounters, followed by outpatient encounters and num_medications.

e. Up until the tree depth of 4, there is no sign of overfitting. However, after depths of 5, as the complexity of the model increases, the gap between the training accuracy and test accuracy becomes wider, which is a sign of overfitting. We can see this in Figure 28.

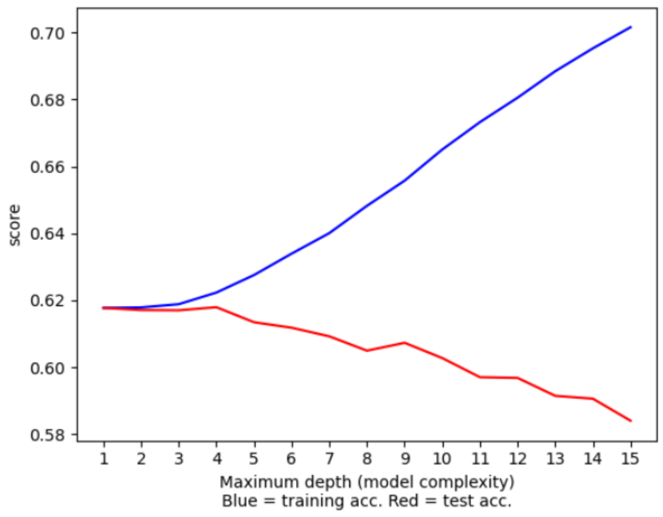


Figure 28 Decision tree model complexity

TASK 4

To understand the difference between the default decision tree and the tuned decision tree, we used the ROC curve as shown in Figure 29**.** This graph also gives insights into the ROC index and model overfitting. The ROC curves show that the tuned model (blue curve) performs better than the default model (red curve) across all thresholds. The ROC index for the tuned decision tree is 0.645, while the default tree is 0.562. The default tree's ROC index suggests that it is overfitting to the training data and performing poorly on the test data. The tuned tree's ROC index indicates improved performance due to the optimization of hyperparameters, which helps the model generalize better to unseen data.

This improvement is due to hyperparameter tuning and the complexity of the tree. Adjusting parameters like max_depth and min_samples_leaf through GridSearchCV reduces overfitting and increases generalization. Moreover, the fine-tuned model is less complex, with better control over the tree's depth, preventing it from memorizing the training data and improving performance on the test set.

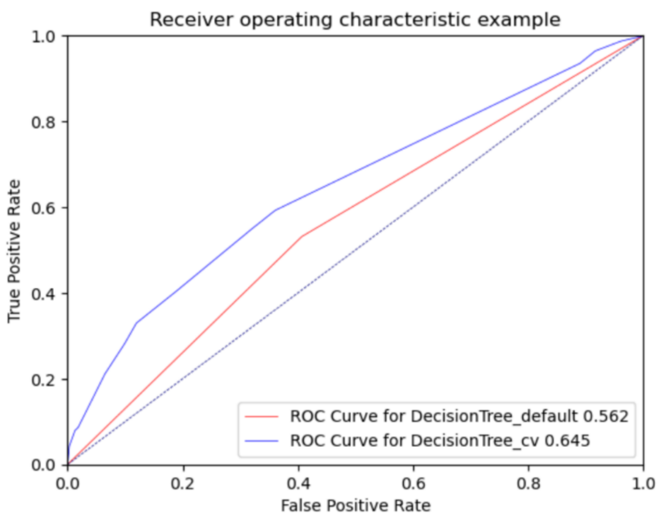
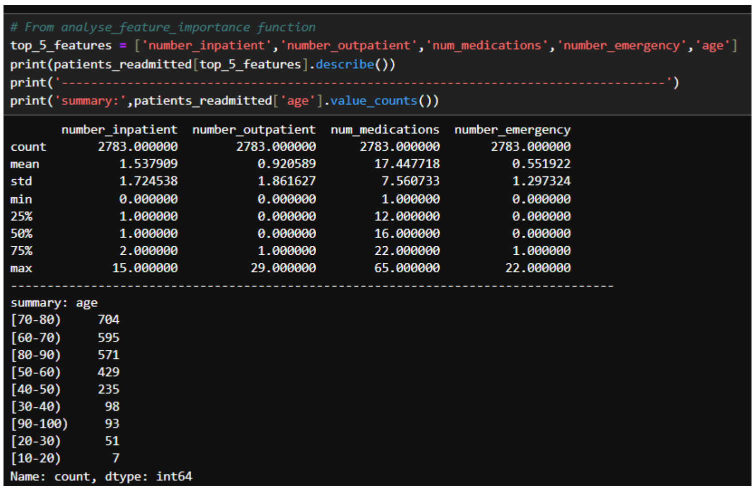


Figure 29 ROC curve for two Decision Trees

TASK 5

As discussed in task 4, the best decision tree model is the one tuned with GridSearchCV and using that model and to answer the general characteristics of patients who could potentially be "readmitted", we used the top 5 important variables from Task 3.d, 'number_inpatient', 'number_outpatient', 'num_medications', 'number_emergency', 'age'.



```
# From analyse_feature_importance function
top_5_features = ['number_inpatient','number_outpatient','num_medications','number_emergency','age']
print(patients_readmitted[top_5_features].describe())
print('-----------------------------------------------------------------------------------')
print('summary:',patients_readmitted['age'].value_counts())
```

```
       number_inpatient  number_outpatient  num_medications  number_emergency
count       2783.000000        2783.000000      2783.000000       2783.000000
mean           1.537909           0.920589        17.447718          0.551922
std            1.724538           1.861627         7.560733          1.297324
min            0.000000           0.000000         1.000000          0.000000
25%            1.000000           0.000000        12.000000          0.000000
50%            1.000000           0.000000        16.000000          0.000000
75%            2.000000           1.000000        22.000000          1.000000
max           15.000000          29.000000        65.000000         22.000000
-----------------------------------------------------------------------------------
summary: age
[70-80)     704
[60-70)     595
[80-90)     571
[50-60)     429
[40-50)     235
[30-40)      98
[90-100)     93
[20-30)      51
[10-20)       7
Name: count, dtype: int64
```

Figure 30 Summary to derive characteristics of patients who would get "readmitted"

From the summary, we can draw insights:

1. Inpatient Visits: Most patients had 1-2 inpatient visits, with some reaching up to 15 visits.
2. Moderate Outpatient Visits: The majority of patients had 0 or 1 outpatient visit, though a few had up to 29 visits.
3. High Medication Usage: Patients typically took 12-22 medications, with some as high as 65 medications.
4. Low to Moderate Emergency Visits: Most patients had 0 or 1 emergency visit, with some outliers having up to 22 visits.
5. Older Age: The majority of patients fall within the 70-80 age group, followed by those aged 60-70, and 80-90. Few patients are younger than 30 years.

In general, patients with **frequent hospital interactions, high medication consumption, and older age** have a higher risk of readmission**.** Improving post-discharge care, medication management, and follow-up visits, particularly for high-risk, older patients with frequent inpatient and emergency visits, can help reduce readmission rates.

Predictive modelling using Regression

TASK 1

The pre-processing was like what we had applied before modelling the decision tree. We checked into data types and null values as shown in Figure 7. As shown in Figure 20, we also considered unary variables. Encoding the categorical variables was also done as shown in Figure 21. The ID and unary variables were dropped before modelling the neural network.

We partitioned the data into training and testing with 70:30 to keep it consistent across every predictive model. Additionally, we also did *standardization on our splits* using StandardScaler to reduce the effect of differences in the scale of input variables.

TASK 2

a. Even though the accuracy and classification metrics are identical to the default model, the GridSearchCV model is preferred because it:

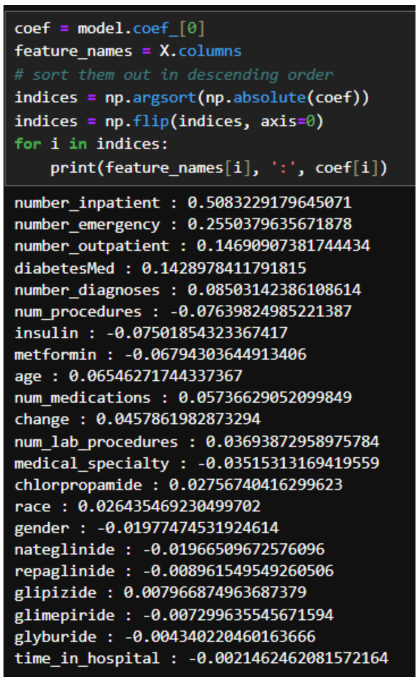
- Optimizes the hyperparameters and confirms that no better configuration is available.
- Validates the model's robustness using cross-validation, making it more reliable for future predictions.

Thus, GridSearchCV provides a well-validated, optimized model, ensuring that the logistic regression model is performing at its best under the current data and setup.

b. The regression function employed is Logistic Regression because it is simple, interpretable, and efficient for binary classification tasks.

c. Yes, standardization was applied to the features using StandardScaler. Standardization is necessary for Logistic Regression because it ensures that all variables are on the same scale. Without it, features with larger numerical ranges could dominate the model, leading to skewed results. Standardization allows the model to focus on the relationships between variables, rather than being influenced by the magnitude of certain features.

d. Variables included in the GridSearchCV regression model are listed in Figure 31.



```python
coef = model.coef_[0]
feature_names = X.columns
# sort them out in descending order
indices = np.argsort(np.absolute(coef))
indices = np.flip(indices, axis=0)
for i in indices:
    print(feature_names[i], ':', coef[i])
```

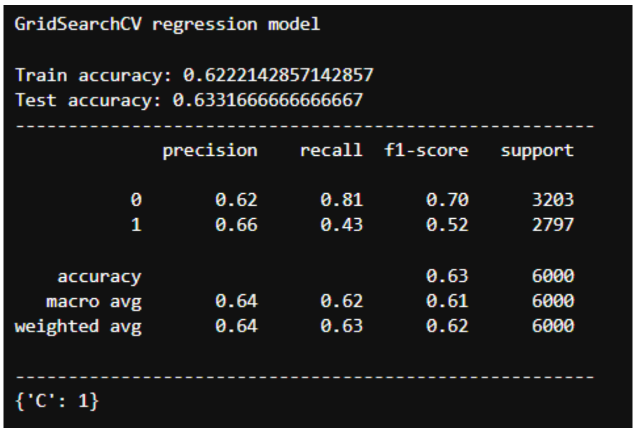
```
number_inpatient : 0.5083229179645071
number_emergency : 0.2550379635671878
number_outpatient : 0.14690907381744434
diabetesMed : 0.1428978411791815
number_diagnoses : 0.08503142386108614
num_procedures : -0.07639824985221387
insulin : -0.07501854323367417
metformin : -0.06794303644913406
age : 0.06546271744337367
num_medications : 0.05736629052099849
change : 0.0457861982873294
num_lab_procedures : 0.03693872958975784
medical_specialty : -0.03515313169419559
chlorpropamide : 0.02756740416299623
race : 0.026435469230499702
gender : -0.01977474531924614
nateglinide : -0.01966509672576096
repaglinide : -0.008961549549260506
glipizide : 0.007966874963687379
glimepiride : -0.007299635545671594
glyburide : -0.004340220460163666
time_in_hospital : -0.0021462462081572164
```

Figure 31 Variables included in GridSearchCV regression model

e. Top five important variables based on coefficients are the following:

number_inpatient : 0.5083229179645071
number_emergency : 0.2550379635671878
number_outpatient : 0.14690907381744434
diabetesMed : 0.1428978411791815
number_diagnoses : 0.08503142386108614

f. Train Accuracy: 62.22%
Test Accuracy: 63.32%



```
GridSearchCV regression model

Train accuracy: 0.6222142857142857
Test accuracy: 0.6331666666666667
-------------------------------------------------------
              precision    recall  f1-score   support

           0       0.62      0.81      0.70      3203
           1       0.66      0.43      0.52      2797

    accuracy                           0.63      6000
   macro avg       0.64      0.62      0.61      6000
weighted avg       0.64      0.63      0.62      6000

-------------------------------------------------------
{'C': 1}
```

Figure 32 GridSearchCV regression model classification

g. Based on the classification accuracy shown in 3. f and the graph shown in Figure 33:
There is no sign of overfitting in this model because:

- The train and test accuracies are very close across different values of hyperparameter C.
- The model performs similarly on training and unseen test data, indicating good generalisation.
- The chosen value of {'C': 1} strikes a balance between fitting the data well and avoiding overfitting.

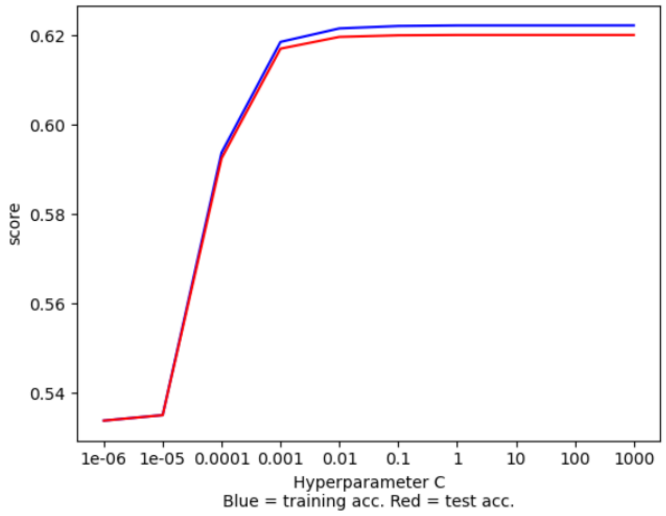


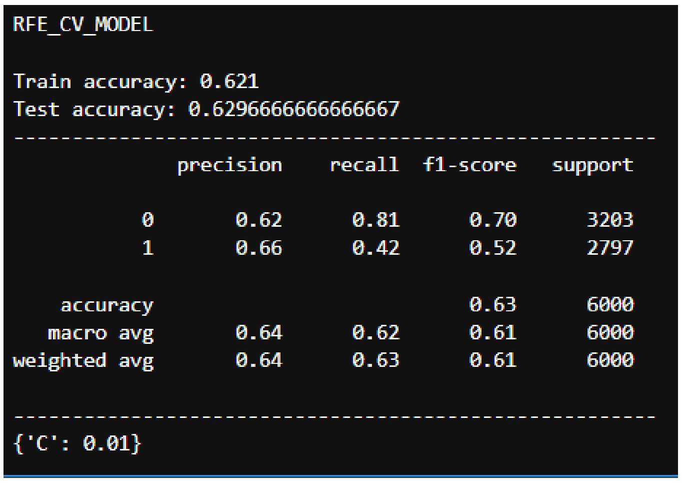
Figure 33 Train and Test Accuracy vs Hyperparameter C

TASK 3

a. Yes, dimensionality reduction using Recursive Feature Elimination (RFE) was useful for identifying a more streamlined set of important features, reducing the complexity of the model without a significant drop in accuracy. The performance of the model with the reduced feature set was very similar to the GridSearchCV model from task 2 which used the full feature set. This suggests that the model's performance was already near-optimal, but reducing the number of features helped maintain a simpler model.

The two models differ primarily in the number of features and the regularization strength. The model with reduced features (RFE) achieved almost the same test accuracy as the full-feature model but with stronger regularization (C=0.01), indicating a more parsimonious model.

b. Train accuracy: 0.62
   Test accuracy: 0.63



```
RFE_CV_MODEL

Train accuracy: 0.621
Test accuracy: 0.6296666666666667
-------------------------------------------------------
              precision    recall  f1-score   support

           0       0.62      0.81      0.70      3203
           1       0.66      0.42      0.52      2797

    accuracy                           0.63      6000
   macro avg       0.64      0.62      0.61      6000
weighted avg       0.64      0.63      0.61      6000

-------------------------------------------------------
{'C': 0.01}
```

Figure 34 Train & Test accuracy of RFE

c. There is no sign of overfitting in the RFE dimensionality-reduced regression model tuned with GridSearchCV:

- The training and test accuracies are very close as shown in 3. b.
- The performance on different values of hyperparameter C shows no significant gap between the training and test accuracy curves as shown in Figure 35**.**
- The model is well-regularized with C=0.01, providing good generalization performance without overfitting the training data.

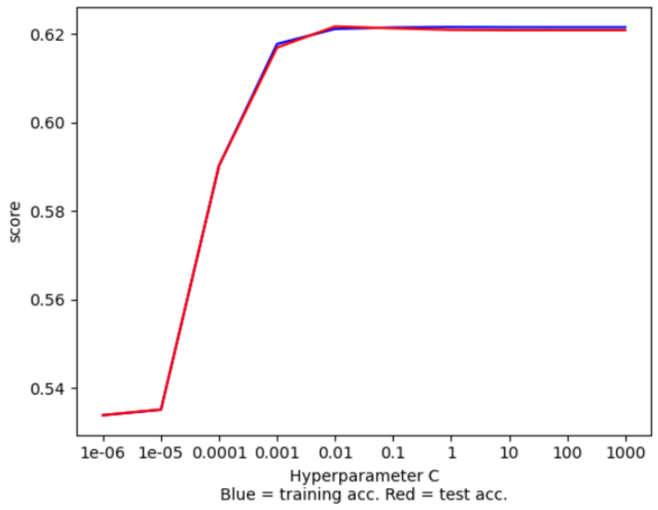
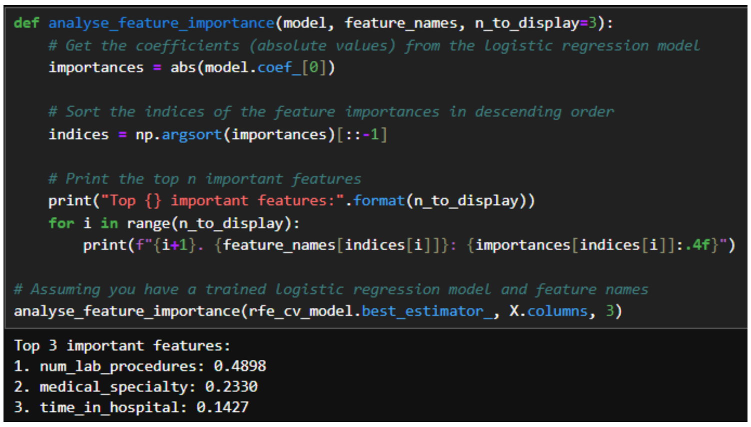


Figure 35 Train and Test Accuracy vs Hyperparameter C

d. The top three important features from RFE dimensionality-reduced regression model tuned with GridSearchCV are:

1. num_lab_procedures: 0.4898
2. medical_specialty: 0.2330
3. time_in_hospital: 0.1427



```python
def analyse_feature_importance(model, feature_names, n_to_display=3):
    # Get the coefficients (absolute values) from the logistic regression model
    importances = abs(model.coef_[0])

    # Sort the indices of the feature importances in descending order
    indices = np.argsort(importances)[::-1]

    # Print the top n important features
    print("Top {} important features:".format(n_to_display))
    for i in range(n_to_display):
        print(f"{i+1}. {feature_names[indices[i]]}: {importances[indices[i]]:.4f}")

# Assuming you have a trained logistic regression model and feature names
analyse_feature_importance(rfe_cv_model.best_estimator_, X.columns, 3)
```

```
Top 3 important features:
1. num_lab_procedures: 0.4898
2. medical_specialty: 0.2330
3. time_in_hospital: 0.1427
```

Figure 36 Top 3 important features

TASK 4

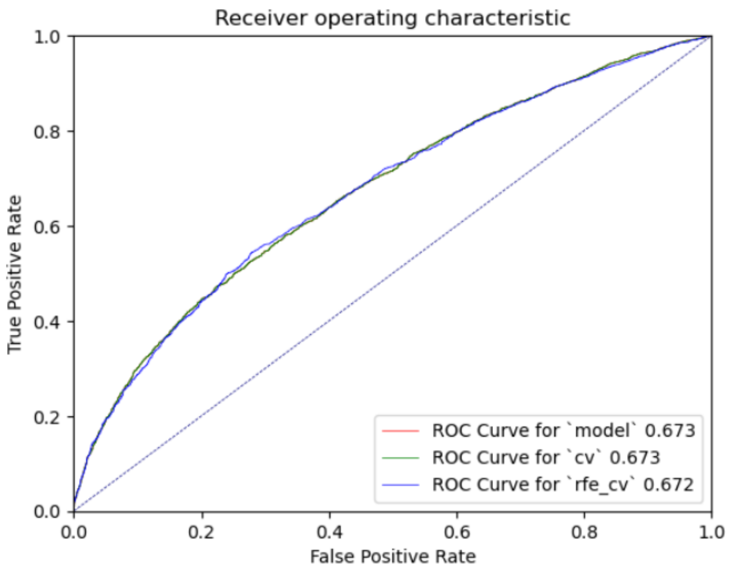


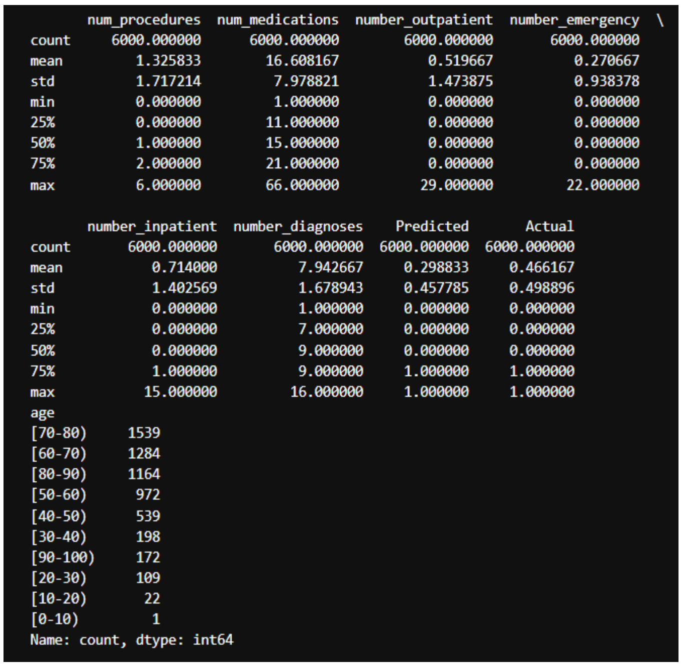
Figure 37 ROC Curve Comparison

Task 2 showed that the regression model tuned with GridSerachCV (the 'cv' in the ROC curve) is better than default regression (the 'model' in the ROC curve) and in task 3 we have concluded that rfe_cv is better than cv. So, based on the ROC curves shown in Figure 37 and from the previous tasks, we select a regression model from the RFE dimensionality-reduced regression model tuned with GridSearchCV (the 'rfe_cv' in the ROC curve).

Using rfe_cv model we determined the patients who could potentially be readmitted and the following are the general characteristics of those patients based on Figure 38:

1. Age: Most patients are between the ages of 70-80, followed by 60-70 and 80-90 age groups.
2. Number of Procedures: On average, around 1.33 procedures, with the majority having 0-2 procedures.
3. Number of Medications: The patients are on an average of 16.6 medications, with a range from 1 to 66 medications.
4. Number of Outpatient Visits: Most patients had around 0.52 outpatient visits, but a few had up to 29 visits.
5. Number of Emergency Visits: These patients had 0.27 emergency visits on average, but the maximum observed was 22 visits.
6. Number of Inpatient Visits: The average number of inpatient visits was 0.71, with a maximum of 15 inpatient visits.
7. Number of Diagnoses: On average, patients had around 7.94 diagnoses, with a range from 1 to 16 diagnoses.

These characteristics show that older patients with a higher number of medications and a moderate number of diagnoses are more likely to be readmitted.



```
       num_procedures  num_medications  number_outpatient  number_emergency  \
count     6000.000000      6000.000000        6000.000000       6000.000000
mean         1.325833        16.608167           0.519667          0.270667
std          1.717214         7.978821           1.473875          0.938378
min          0.000000         1.000000           0.000000          0.000000
25%          0.000000        11.000000           0.000000          0.000000
50%          1.000000        15.000000           0.000000          0.000000
75%          2.000000        21.000000           0.000000          0.000000
max          6.000000        66.000000          29.000000         22.000000

       number_inpatient  number_diagnoses    Predicted       Actual
count       6000.000000       6000.000000  6000.000000  6000.000000
mean           0.714000          7.942667     0.298833     0.466167
std            1.402569          1.678943     0.457785     0.498896
min            0.000000          1.000000     0.000000     0.000000
25%            0.000000          7.000000     0.000000     0.000000
50%            0.000000          9.000000     0.000000     0.000000
75%            1.000000          9.000000     1.000000     1.000000
max           15.000000         16.000000     1.000000     1.000000
age
[70-80)     1539
[60-70)     1284
[80-90)     1164
[50-60)      972
[40-50)      539
[30-40)      198
[90-100)     172
[20-30)      109
[10-20)       22
[0-10)         1
Name: count, dtype: int64
```

Figure 38 General Characteristics of Patients Predicted to be Readmitted

Predictive modelling using Neural Networks

TASK 1

The pre-processing was the same as what we had applied before modelling the decision tree and regression. We checked into data types and null values as shown in Figure 7. As shown in Figure 20, we also considered unary variables. Encoding the categorical variables was also done as shown in Figure 21. The ID and unary variables were dropped before modelling the neural network.

We partitioned the data into training and testing with 70:30 to keep it consistent across every predictive model. Additionally, we also did standardization on our splits using StandardScaler to reduce the effect of differences in the scale of input variables.

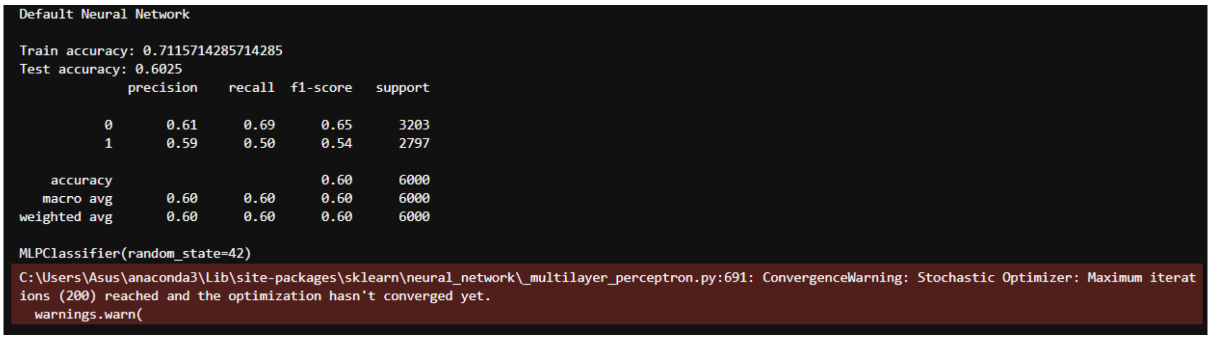
TASK 2

a. The default neural network was developed using MLPClassifier, and using get_params() method, The following are the parameters with short explanations each:

- activation: 'relu' – The activation function for the hidden layer neurons.
- alpha: 0.0001 – L2 regularization term to prevent overfitting.
- batch_size: 'auto' – Number of training samples used in each update of weights, determined automatically if not set.
- beta_1: 0.9 – Exponential decay rate for the first moment estimates in the Adam optimizer.
- beta_2: 0.999 – Exponential decay rate for the second moment estimates in the Adam optimizer.
- early_stopping: False – Whether to stop training early if validation score doesn't improve.
- epsilon: 1e-08 – A small constant to prevent division by zero in Adam optimizer.
- hidden_layer_sizes: (100,) – The number of neurons in the hidden layers (100 neurons in one hidden layer).
- learning_rate: 'constant' – Learning rate schedule, constant throughout training.
- learning_rate_init: 0.001 – Initial learning rate for weight updates.
- max_fun: 15000 – Maximum number of loss function calls.

- max_iter: 200 – Maximum number of iterations (epochs) for training.
- momentum: 0.9 – Momentum for gradient updates to accelerate convergence.
- n_iter_no_change: 10 – Number of iterations with no improvement in validation score to trigger early stopping (if enabled).
- nesterovs_momentum: True – Whether to use Nesterov momentum, which improves the convergence speed.
- power_t: 0.5 – Power of the inverse scaling of learning rate when learning_rate is set to 'invscaling'.
- random_state: 42 – Seed for random number generation to ensure reproducibility.
- shuffle: True – Whether to shuffle training data before each epoch.
- solver: 'adam' – The optimization algorithm used for weight updates (Adam optimizer).
- tol: 0.0001 – Tolerance for optimization convergence.
- validation_fraction: 0.1 – Fraction of training data used for validation when early stopping is enabled.
- verbose: False – Whether to print progress messages during training.
- warm_start: False – Whether to reuse the solution from a previous training run for further training

b. Train accuracy: 71.16%
   Test accuracy: 60.25%



```
Default Neural Network

Train accuracy: 0.7115714285714285
Test accuracy: 0.6025
              precision    recall  f1-score   support

           0       0.61      0.69      0.65      3203
           1       0.59      0.50      0.54      2797

    accuracy                           0.60      6000
   macro avg       0.60      0.60      0.60      6000
weighted avg       0.60      0.60      0.60      6000

MLPClassifier(random_state=42)
C:\Users\Asus\anaconda3\Lib\site-packages\sklearn\neural_network\_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic Optimizer: Maximum iterat
ions (200) reached and the optimization hasn't converged yet.
  warnings.warn(
```

Figure 39 Accuracy of training and test datasets

c. No, as we can see in "ConvergenceWarnging" shown in Figure 39, the neural network did not achieve convergence before the maximum iteration. This reminds us that max_iter hyperparameter must be increased. It is not the best model as we can see the difference between training and testing accuracies, showing the sign of overfitting to the training data.

TASK 3

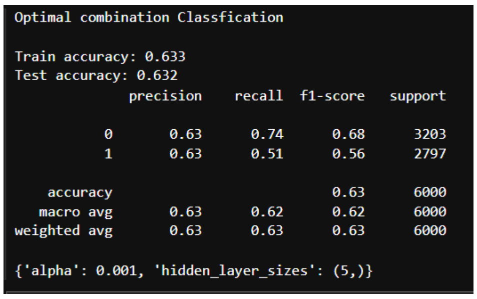
a. We modified three key parameters from the default model using GridSearchCV: max_iter, alpha, and hidden_layer_sizes. Through this systematic search, we optimized the neural network's architecture and regularization to achieve the best model performance.

- max_iter: We initially set max_iter to 800 to ensure convergence during the training process, as the default value of 200 was insufficient for achieving the best model performance.
- hidden_layer_sizes: To identify the optimal size for the hidden layer, we compared various sizes. Since the size of the hidden layer must be less than the number of input variables (which we identified as 24 using the shape method on X_train), we tested hidden layer sizes ranging from 5 to 24 in increments of 5. Through this process, we discovered that a hidden layer size of 5 was optimal for this model.
- alpha and hidden_layer_sizes combination: In the final step, we refined our search by varying the hidden layer size further (between 1 and 7) and testing different values for the regularization parameter alpha (ranging from 0.00001 to 0.01). This allowed us to identify the best combination of parameters, which are alpha: 0.001 and hidden_layer_sizes: 5

b. Classification accuracy of train and test datasets:

Train Accuracy: 0.633

Test accuracy: 0.632



```
Optimal combination Classfication

Train accuracy: 0.633
Test accuracy: 0.632
              precision    recall  f1-score   support

           0       0.63      0.74      0.68      3203
           1       0.63      0.51      0.56      2797

    accuracy                           0.63      6000
   macro avg       0.63      0.62      0.62      6000
weighted avg       0.63      0.63      0.63      6000

{'alpha': 0.001, 'hidden_layer_sizes': (5,)}
```

Figure 40 Accuracy of training and test datasets

c. Yes it did converge as there is no warning and compared to the earlier models; the default and the one with max_iter of 800, this model with optimal combination of 5 hidden layers and 0.001 alpha seems to be better.

d. No. There is no overfitting in model with a hidden layer size of 5 and alpha of 0.001 and this is also backed by the very close train and test accuracy. However, other hidden layers can cause overfitting, especially after 5 as shown in Figure 41.

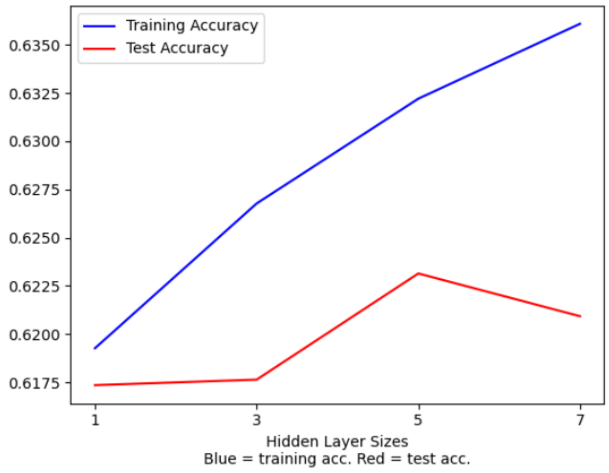
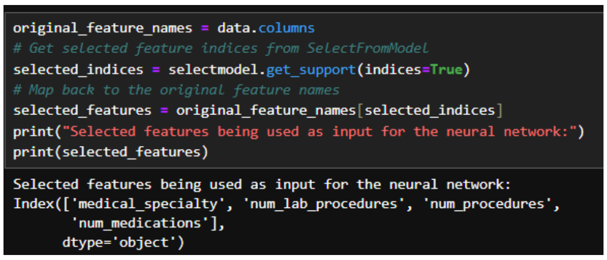


Figure 41 Train and Test Accuracy vs Hidden Layers

TASK 4

e. Yes, fine-tuning the network with dimensionality reduction by selecting variables from the best decision tree did improve the network accuracy. Although the training accuracy dropped compared to the network tuned with GridSearchCV, the testing accuracy improved slightly. Moreover, the network architecture has significantly reduced features used to predict the same response variable. Inputs used as the network input are medical_speciality, num_lab_procedures, num_procedures, and num_medications as shown in Figure 42. This change in input variables will change the network architecture.

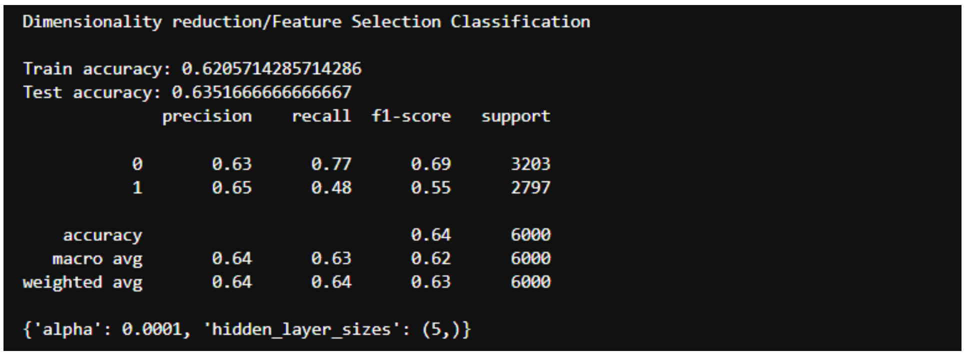


```
original_feature_names = data.columns
# Get selected feature indices from SelectFromModel
selected_indices = selectmodel.get_support(indices=True)
# Map back to the original feature names
selected_features = original_feature_names[selected_indices]
print("Selected features being used as input for the neural network:")
print(selected_features)
```

```
Selected features being used as input for the neural network:
Index(['medical_specialty', 'num_lab_procedures', 'num_procedures',
       'num_medications'],
      dtype='object')
```

Figure 42 Features being used as input for the neural network

f. Train accuracy: 0.623 and test accuracy: 0.634



```
Dimensionality reduction/Feature Selection Classification

Train accuracy: 0.6205714285714286
Test accuracy: 0.6351666666666667
              precision    recall  f1-score   support

           0       0.63      0.77      0.69      3203
           1       0.65      0.48      0.55      2797

    accuracy                           0.64      6000
   macro avg       0.64      0.63      0.62      6000
weighted avg       0.64      0.64      0.63      6000

{'alpha': 0.0001, 'hidden_layer_sizes': (5,)}
```

Figure 43 Accuracy of training and test datasets

g. To find the number of iterations, we used 'cv_sel_model.best_estimator_.n_iter_'. The number of iterations has dropped to 29 from 87 before the dimensionality reduction. This indicates that the network processing and training time has been reduced.

h. There is no significant sign of overfitting based on the training and testing accuracy as shown in Figure 44. Both training and testing accuracy increase with more hidden layer sizes except 6-7 hidden layers, and there's no significant divergence between the two, which indicates the model is generalizing well to the test data. In summary, the model network to perform consistently across training and testing, so there's no significant sign of overfitting.

The training process has also converged as there is no warning about the convergence.

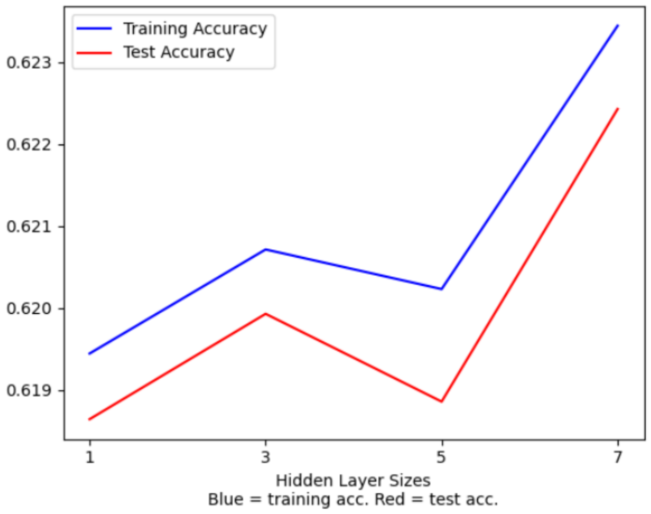


Figure 44 Train and Test Accuracy vs Hidden Layers

TASK 5

In comparing the performance of five different networks constructed, cv_1, the model in which we tried to find the optimal number of hidden layers and cv_2, the model in which we tried to find the optimal combination of the number of hidden layers and alpha have the *same highest ROC index*. To select one, we compared their accuracies, classification report, parameters, and overfitting check.

1. Train Accuracy: cv_1: 0.6322 and cv_2: 0.633
2. Test Accuracy: cv_1: 0.6317 and cv_2: 0.632

Checking the accuracies, cv_2 shows slight improvement with very minimal differences.

3. Both models have the same classification report metrics, drawing no difference.
4. cv_1: {'hidden_layer_sizes': (5,)}
   cv_2: {'alpha': 0.001, 'hidden_layer_sizes': (5,)}

Both models have selected 5 neurons in the hidden layer as optimal, but cv_2 also tunes the regularisation parameter (alpha = 0.001), which could provide better generalisation.

5. Both models show similar performance between training and testing, with very close accuracies. This indicates no significant overfitting in either model.

Therefore, cv_2 is slightly better, and we have selected cv_2.

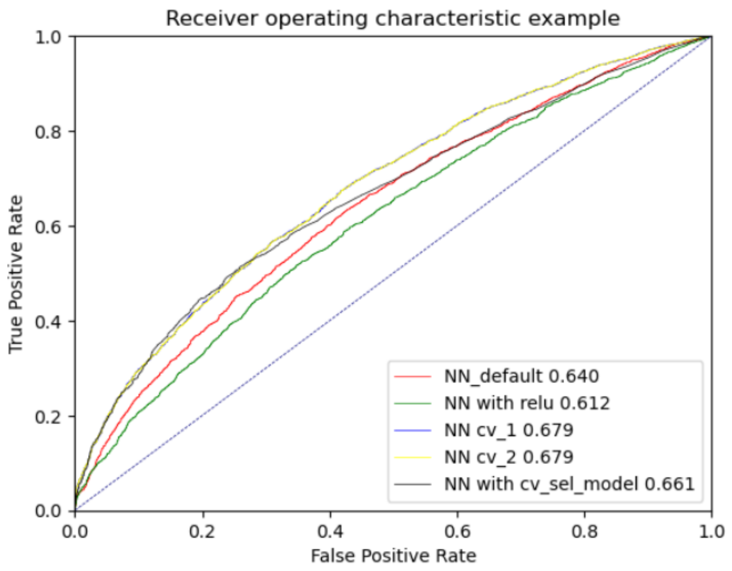
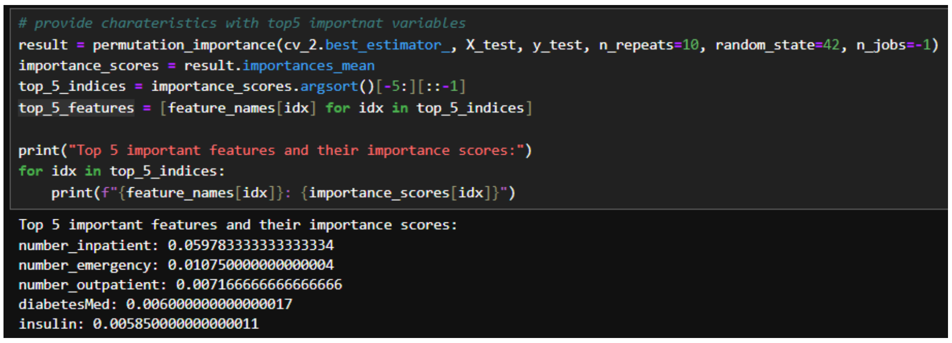


Figure 45 ROC curve for all different neural networks

Before generating the characteristics of the patients who would potentially get “readmitted” using cv_2, we noticed that there were too many variables to draw characteristics from and some variables did not affect much on the prediction, so we used permutation_importance from sklearn inspection to get top 5 important variables from cv_2 model.



```
# provide charateristics with top5 importnat variables
result = permutation_importance(cv_2.best_estimator_, X_test, y_test, n_repeats=10, random_state=42, n_jobs=-1)
importance_scores = result.importances_mean
top_5_indices = importance_scores.argsort()[-5:][::-1]
top_5_features = [feature_names[idx] for idx in top_5_indices]

print("Top 5 important features and their importance scores:")
for idx in top_5_indices:
    print(f"{feature_names[idx]}: {importance_scores[idx]}")
```

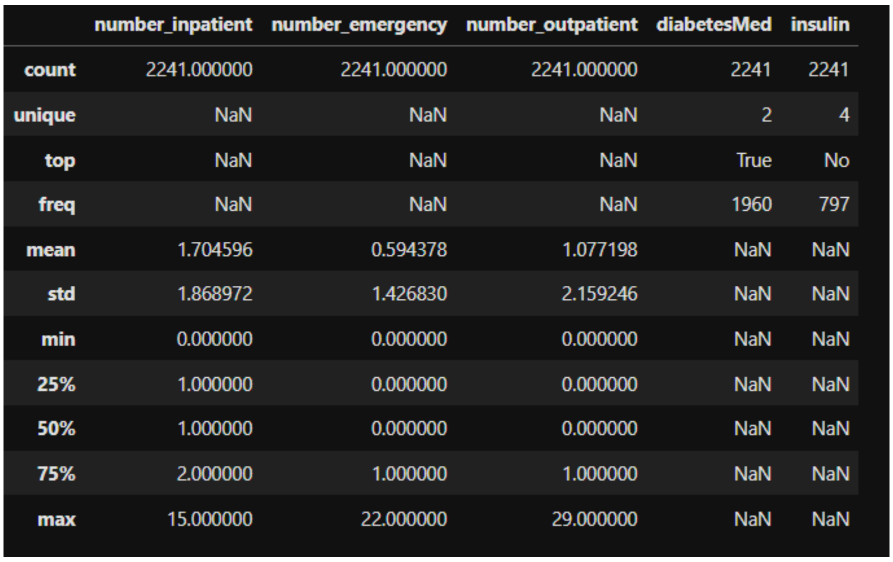
```
Top 5 important features and their importance scores:
number_inpatient: 0.059783333333333334
number_emergency: 0.010750000000000004
number_outpatient: 0.007166666666666666
diabetesMed: 0.006000000000000017
insulin: 0.005850000000000011
```

Figure 46 Top 5 important features and their importance score

Using those top 5 important variables, we get the following characteristics:

1. Number_inpatient: Patients had a mean of 1-2 inpatient visits, indicating they often required hospital stays before being readmitted.
2. Number_emergency: Most patients had no emergency visits, but some had as many as 22, reflecting varying levels of acute care needs.
3. Number_outpatient: Few outpatient visits, with a median of 0, though some patients had frequent outpatient care.
4. diabetesMed: 87.5% of patients are on diabetes medications, indicating that diabetes is a significant factor in readmissions.
5. Insulin: 35.6% of patients use insulin, suggesting that severe diabetes cases contribute to readmission risk.

In summary, patients with **frequent inpatient visits, those managing chronic conditions like diabetes, and some with high outpatient or emergency care needs** are more likely to be readmitted.



	number_inpatient	number_emergency	number_outpatient	diabetesMed	insulin
count	2241.000000	2241.000000	2241.000000	2241	2241
unique	NaN	NaN	NaN	2	4
top	NaN	NaN	NaN	True	No
freq	NaN	NaN	NaN	1960	797
mean	1.704596	0.594378	1.077198	NaN	NaN
std	1.868972	1.426830	2.159246	NaN	NaN
min	0.000000	0.000000	0.000000	NaN	NaN
25%	1.000000	0.000000	0.000000	NaN	NaN
50%	1.000000	0.000000	0.000000	NaN	NaN
75%	2.000000	1.000000	1.000000	NaN	NaN
max	15.000000	22.000000	29.000000	NaN	NaN

Figure 47 Characteristics of Patients Predicted to be Readmitted

Decision Making

TASK 1

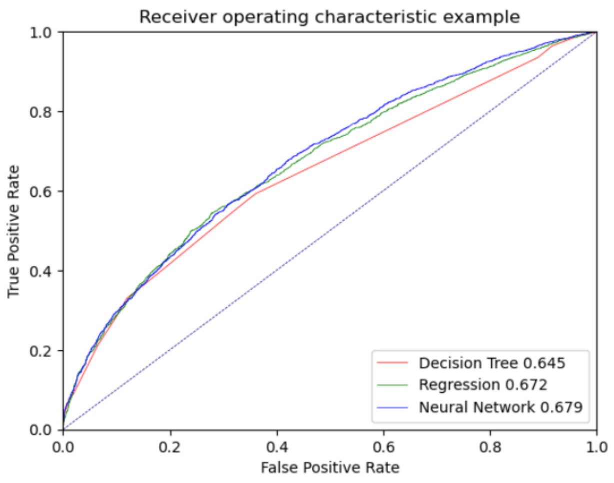
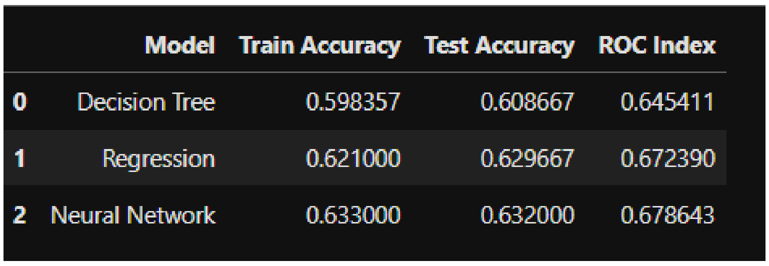


Figure 48 ROC Curve Comparison



	Model	Train Accuracy	Test Accuracy	ROC Index
0	Decision Tree	0.598357	0.608667	0.645411
1	Regression	0.621000	0.629667	0.672390
2	Neural Network	0.633000	0.632000	0.678643

Figure 49 Comparison of model's performance

Based on the comparison of the models using the ROC curve shown in Figure 48 and the accuracy table shown in Figure 49, the best model for decision-making would be the Neural Network tune with GridSearchCV due to the following reasons:

1. The Neural Network has the highest test accuracy at 0.632, indicating it performs slightly better than the other models on unseen data.
2. The ROC Index for the Neural Network is the highest at 0.679, meaning it performs better at distinguishing between readmitted and not readmitted classes.
3. Neural network has minimal overfitting compared to Decision Tree.

4. The ROC curve for the Neural Network (blue line) consistently performs better across all thresholds compared to the Decision Tree (red) and Regression (green), indicating it has better predictive power across the entire spectrum of false positive and true positive rates.

TASK 2

1. Decision tree
    a. Positives
        i. Interpretability: Decision trees are easy to understand and visualize.
        ii. Handling Non-linear Relationships: Decision trees can capture complex, non-linear relationships
        iii. No Feature Scaling Required: Decision trees do not require feature scaling
    b. Negatives
        i. Overfitting: Decision trees are prone to overfitting, especially when not properly pruned or when the tree becomes too deep
        ii. Lower Predictive Power: In this analysis, the Decision Tree had the lowest ROC Index and the lowest test accuracy, indicating that it doesn't perform as well as the other models for this dataset.
2. Regression
    a. Positives
        i. Simplicity and Efficiency: Regression is computationally efficient, particularly for large datasets, and works well when the relationship between the features and the target variable is linear.
        ii. Well-understood and Interpretable: The model provides clear coefficients for each feature, which can be easily interpreted to understand the contribution of each variable to the prediction.
        iii. Less Prone to Overfitting: With proper regularization, regression models are less prone to overfitting compared to complex models like decision trees or neural networks.
    b. Negatives
        i. Linear Assumptions: Regression assumes a linear relationship between the independent variables and the dependent variable. In cases with complex non-linear relationships, it may underperform.
        ii. Moderate Performance: In this case, regression showed decent performance with a test accuracy of 0.630 and ROC AUC of 0.672, but it was outperformed by the Neural Network.

3. Neural Network
    a. Positives
        i. Best Predictive Performance: The neural network had the highest test accuracy (0.632) and the best ROC AUC score (0.679), showing superior performance in predicting patient readmission.
        ii. Captures Complex Relationships: Neural networks can model complex, non-linear relationships between features, making them suitable for datasets where such relationships exist.
        iii. Flexible Model: Neural networks can be adjusted by changing the number of layers, neurons, and other hyperparameters to fine-tune the performance.
    b. Negatives
        i. Interpretability: Neural networks are difficult to interpret how specific predictions are made, especially compared to decision trees or logistic regression.
        ii. Longer Training Time: Neural networks typically require more computational resources and longer training times compared to simpler models like decision trees or logistic regression.