Project (a):
Association mining to find hotspots based on a Patient Route Data

1. What pre-processing was required on the dataset before building the association mining model? What variables did you include in the analysis? Justify your choice.

Solution: Before building the association mining model, it is required
We selected the 'location' and 'patient_id' variables in the analysis, because we need to find the connection between the patient and the location they have been. Other variables are not helpful for our analysis, so we choose to delete global_num, date, latitude and longitude. Below is the code.

```
In [40]: df.drop(['global_num'],axis=1, inplace=True)
         df.drop(['date'],axis=1, inplace=True)
         df.drop(['latitude'],axis=1, inplace=True)
         df.drop(['longitude'],axis=1, inplace=True)
```

2. Conduct association mining and answer the following:
a.What 'min_support' and `min_confidence' thresholds were set for this mining exercise? Rationale why these values were chosen?
First, we group by patient_id, then list all routes.

```
#group by locations
routes = df.groupby(['patient_id'])['location'].apply(list)
routes.count()
```

```
911
```

Totally 911 routes.
Then we want routes that at least appear 10 times.

```
10/911
```

```
0.010976948408342482
```

Therefore, we set min_support as 0.01.
Next, we found the lowest support of all single locations is 0.01977, which means that every location at least appears 10 times in all routes.

[82]:
```
result_df_min = result_df.sort_values(by='Support', ascending=True)
result_df_min[:5]
```

[82]:

	Left_side	Right_side	Support	Confidence	Lift
11		Chittorgarh_Rajasthan	0.010977	0.010977	1.0
34		Ramanagara_Karnataka	0.010977	0.010977	1.0
17		Jagadhri_Haryana	0.010977	0.010977	1.0
4		Basirhat_West Bengal	0.010977	0.010977	1.0
5		Batumi_Adjara	0.010977	0.010977	1.0

We want the min_confidence higher than all single locations' support. Therefore, we set the min_confidence as 0.01 also.

```
from apyori import apriori

transaction_list = list(transactions)
results = list(apriori(transaction_list, min_support= 0.01, min_confidence = 0.01))
print(results[:5])
```

```
[RelationRecord(items=frozenset({'Alipurduar_West Bengal'}), support=0.026344676180021953, ordered_statistics=[OrderedStatistic(items_base=f
rozenset(), items_add=frozenset({'Alipurduar_West Bengal'}), confidence=0.026344676180021953, lift=1.0)]), RelationRecord(items=frozenset
({'Anand_Gujarat'}), support=0.01646542261251372, ordered_statistics=[OrderedStatistic(items_base=frozenset(), items_add=frozenset({'Anand_G
ujarat'}), confidence=0.01646542261251372, lift=1.0)]), RelationRecord(items=frozenset({'Asansol_West Bengal'}), support=0.01866081229418221
6, ordered_statistics=[OrderedStatistic(items_base=frozenset(), items_add=frozenset({'Asansol_West Bengal'}), confidence=0.01866081229418221
6, lift=1.0)]), RelationRecord(items=frozenset({'Barh_Bihar'}), support=0.012074643249176729, ordered_statistics=[OrderedStatistic(items_bas
e=frozenset(), items_add=frozenset({'Barh_Bihar'}), confidence=0.012074643249176729, lift=1.0)]), RelationRecord(items=frozenset({'Basirhat_
West Bengal'}), support=0.010976948408342482, ordered_statistics=[OrderedStatistic(items_base=frozenset(), items_add=frozenset({'Basirhat_We
st Bengal'}), confidence=0.010976948408342482, lift=1.0)])]
```

b.Report the top-5 rules and interpret them

Because we want to find the top 5 highly correlated locations, we sorted the results by "Lift".

```
result_df_top5 = result_df.sort_values(by='Lift', ascending=False)
result_df_top5[:5]
```

	Left_side	Right_side	Support	Confidence	Lift
50	Jorapokhar_Jharkhand	Channapatna_Karnataka	0.012075	0.305556	5.458061
49	Channapatna_Karnataka	Jorapokhar_Jharkhand	0.012075	0.215686	5.458061
52	Gokak_Karnataka	Sardarshahar_Rajasthan	0.027442	0.500000	3.399254
53	Sardarshahar_Rajasthan	Gokak_Karnataka	0.027442	0.186567	3.399254
59	Sardarshahar_Rajasthan	Lucknow_Uttar Pradesh	0.018661	0.126866	1.462970

From the above tests we found three highly correlated locations rules:
1: Town Jorapokhar in Jharkhand state ≒ Town Channapatna in Karnataka state
30% of patients who visited Jorapokhar also visited Channapatna. This usually happens in 1.2% of all routines.

2: Town Gokak in Karnataka state ≒ Town Sardarshahar in Rajasthan state

50% of patients who visited Gokak also visited Sardarshahar .This usually happens in 2.7% of all routines.

3: Town Sardarshahar in Rajasthan state ⇏ Town Lucknow in Uttar Pradesh state

12% of patients who visited Sardarshahar also visited Lucknow.This usually happens in 1.8% of all routines.

3. Identify top-5 common routes that COVID-19 positive patients from the town Ranebennur in Karnataka state have travelled.

If we set min_support and min_confidence to 0.01, it shows the empty dataframe.

```
Empty DataFrame
Columns: [Left_side, Right_side, Support, Confidence, Lift]
Index: []
```

So we need to use the smaller min_support to find the top-5 common routes, and we set it as 0.001 as shown below.

```
from apyori import apriori

transaction_list = list(transactions)
results = list(apriori(transaction_list, min_support=0.001, min_confidence=0.01))
print(results[:5])
```

We sort the top-5 common routes by lift.

```
result_df_routes = result_df.loc[result_df['Left_side'] == 'Ranebennur_Karnataka']
result_df_routes = result_df_routes.sort_values(by='Lift', ascending=False)
result_df_routes[["Left_side","Right_side"]].values[:5]
```

```
array([['Ranebennur_Karnataka',
        'Jorapokhar_Jharkhand,Ratnagiri_Maharashtra'],
       ['Ranebennur_Karnataka',
        'Channapatna_Karnataka,Medinipur_West Bengal,Ratnagiri_Maharashtra'],
       ['Ranebennur_Karnataka',
        'Mahidpur_Madhya Pradesh,Channapatna_Karnataka,Ramanagara_Karnataka'],
       ['Ranebennur_Karnataka',
        'Jorapokhar_Jharkhand,Channapatna_Karnataka,Ratnagiri_Maharashtra'],
       ['Ranebennur_Karnataka',
        'Jorapokhar_Jharkhand,Channapatna_Karnataka,Mahidpur_Madhya Pradesh']],
      dtype=object)
```

4. Can you perform sequence analysis on this dataset? If yes, present your results. If not, rationalize why.

Yes, set min_sup, min_confidence as 0.01 and 0.01.

```
get_association_rules(sequences, 0.01, 0.01)
```

	Left_rule	Right_rule	Support	Confidence
0	[Sardarshahar_Rajasthan]	[Gokak_Karnataka]	0.026345	0.179104
1	[Sardarshahar_Rajasthan]	[Lucknow_Uttar Pradesh]	0.018661	0.126866
2	[Sardarshahar_Rajasthan]	[Kollam_Kerala]	0.012075	0.082090
3	[Channapatna_Karnataka]	[Jorapokhar_Jharkhand]	0.010977	0.196078

5. How can the outcome of this study be used by the relevant decision-makers?

From the association test above, we can identify three pair correlated regions:

- Town Jorapokhar in Jharkhand state ≒ Town Channapatna in Karnataka state
- Town Gokak in Karnataka state ≒ Town Sardarshahar in Rajasthan state
- Town Sardarshahar in Rajasthan state ≒ Town Lucknow in Uttar Pradesh state

Therefore, it is advised that the government should share the covid-19 information with the correlated region. This may be extremely helpful for tracking patients' routines and preventing regional outbreak of infection.

Project (b):
Clustering Diabetes data

1. What pre-processing was required on the dataset (D2.csv) before building the clustering model?

```
admission_type_id          36183 non-null int64
discharge_disposition_id   36183 non-null int64
admission_source_id        36183 non-null int64
```

From the .info() output, we found the **admission_type_id,discharge_disposition_id and admission_source_id** variable types are set incorrectly. The output lists set these variables as integers, while based on the description, these variables should be categorical variables. Thus, we changed incorrect data types.

```
df['admission_type_id'] = df['admission_type_id'].astype(str)
df['discharge_disposition_id'] = df['discharge_disposition_id'].astype(str)
df['admission_source_id'] = df['admission_source_id'].astype(str)
```

Next we applied .value_counts() to get more information, and found some empty values exist in **age, race** and **chlorpropamide** columns.

```
df['age'].value_counts()
```

```
[70-80)     8990
[60-70)     8232
[80-90)     6386
[50-60)     6159
[40-50)     3328
[30-40)     1245
[90-100)    1072
[20-30)      561
[10-20)      179
[0-10)        22
                9
Name: age, dtype: int64
```

```
df['race'].value_counts()
```

```
Caucasian          27776
AfricanAmerican     5764
                     848
Hispanic             806
Other                696
Asian                293
Name: race, dtype: int64
```

```
df['chlorpropamide'].value_counts()
```

```
No         36160
Steady        14
                9
Name: chlorpropamide, dtype: int64
```

Filling missing values with mode.

```
#data pre-processing
df['race'].fillna(df['race'].mode()[0], inplace=True)
df['chlorpropamide'].fillna(df['chlorpropamide'].mode()[0], inplace=True)
df['age'].fillna(df['age'].mode()[0], inplace=True)
```

2. Build a clustering model to profile the characteristics of diabetic patients. Answer the followings:

a. What clustering algorithm have you used?

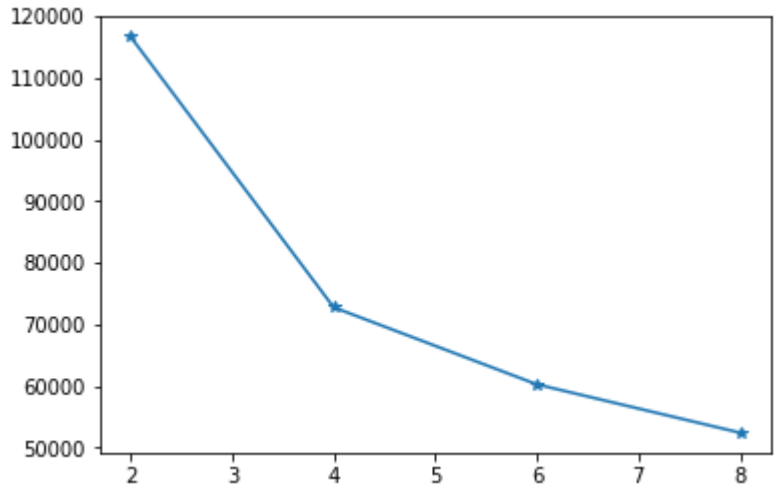
We run the K Prototypes clustering since we have mixed variables types.
Two categorical variables: **A1C result** and **age** as attributes.
Because **A1C results** reflect patients' average blood sugar level for the past two to three months which can evaluate the blood glucose control in diabetic patients. And **age** is a commonly used categorical variable
Three numerical variables:**'num_medications', 'number_diagnoses' and 'time_in_hospital'**.
These three variables are useful for assessing the severity of diabetic patients.

b. List the attributes used in this analysis.

We want to classify patients according to the severity of diabetes. Thus, we used three numerical variables including **'num_medications', 'number_diagnoses', 'time_in_hospital'**.And two categorical variables including **A1C result** and **age**.

c. What is the optimal number of clusters identified? How did you reach this optimal number?

To get the optimal number of clusters we first used the elbow plot to find the turning point.



From the above plot, we see that the turning points appear in k = 4 and k = 6. Next, we use the silhouette score to decide which is the best parameter.

```
X_num = [[row[0], row[1],row[2]] for row in X] # Variables of X with numeric datatype
X_cat = [[row[3], row[4]] for row in X] # variables of X with categorical datatype
```

```
model = clusters[1]
silScoreNums = silhouette_score(X_num, model.fit_predict(X,categorical=[1]), metric='euclidean')
silScoreCats = silhouette_score(X_cat, model.fit_predict(X,categorical=[1]), metric='hamming')
silScore = (silScoreNums + silScoreCats) / 2
print("The avg Silhouette score for k=4: " + str(silScore))

model = clusters[2]
silScoreNums = silhouette_score(X_num, model.fit_predict(X,categorical=[1]), metric='euclidean')
silScoreCats = silhouette_score(X_cat, model.fit_predict(X,categorical=[1]), metric='hamming')
silScore = (silScoreNums + silScoreCats) / 2
print("The avg Silhouette score for k=6: " + str(silScore))
```

```
The avg Silhouette score for k=4: 0.08483099759819981
The avg Silhouette score for k=6: 0.0414079224453025
```

The optimal number is 4, the silhouette score of 4 is 0.08, which is higher than the score of 6.

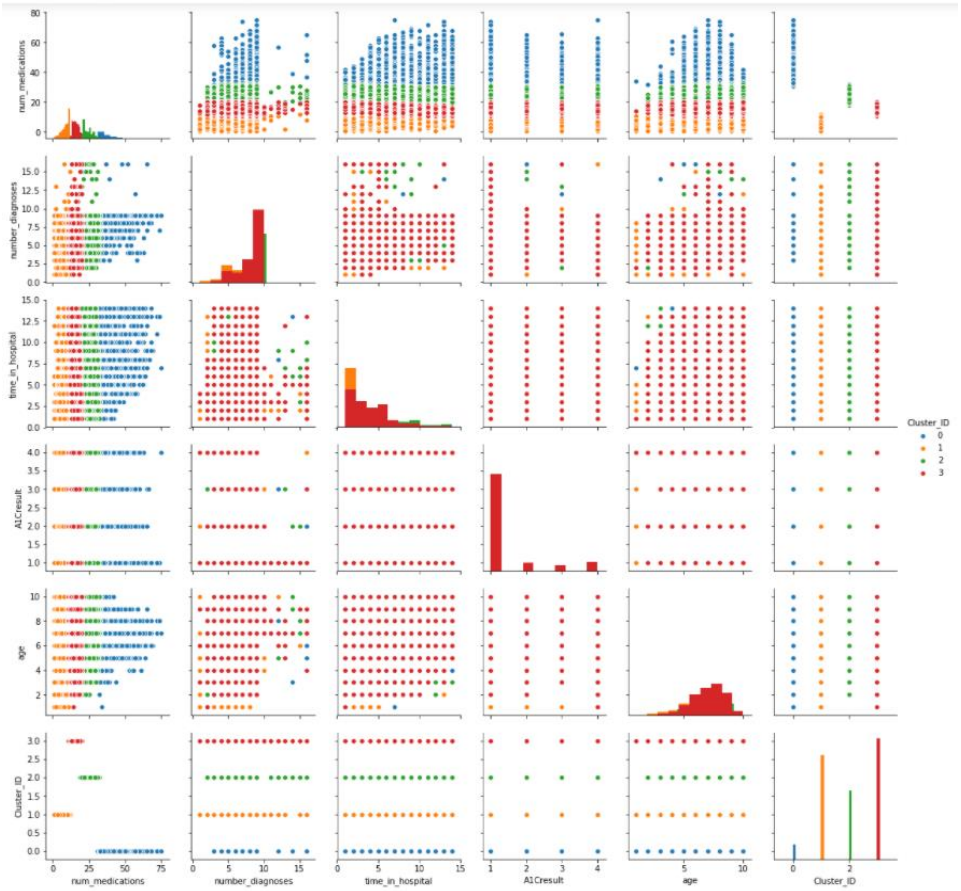
d. Did you normalise the variables? What was its effect on the model?

Yes, we use it to adjust all of the values measured on different scales to a notionally common scale.

```
df2 = df[[ 'num_medications','number_diagnoses','time_in_hospital','A1Cresult','age']]

X = df2.to_numpy()

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

For comparison we also generated clustering without normalization.

1: Not normalization:



2: Normalized

Does the variable normalization process enable a better clustering solution?

Yes, the normalization process has a great effect on the model, because clusters are determined by the distance between points in mathematical space, this importance is especially great in cluster analysis. By comparing these above pair plots we found that not normalized clustering is only sensitive in the wider range variables like **'num_medications'**, where clustering after normalization can interpret all variables well.

3. For the model with the optimal number of clusters,

a. Visualize the clusters using 'pairplot' and interpret the visualization.

The 'pairplot' shows us how different cluster members have different value distributions on different variables. Here is how to interpret the plots:

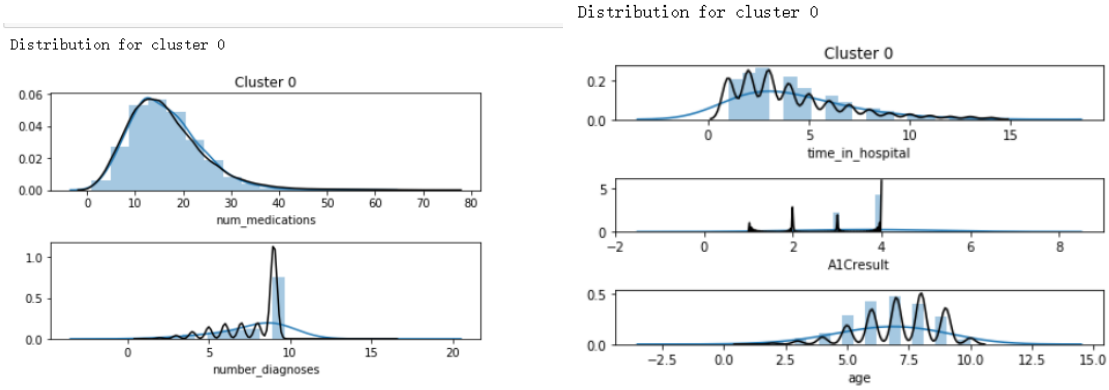
1. Looking at the **num_medications** and **time_in_hospital** (first row, third column), we could see Cluster 2 has a large number of medications and large diagnoses, while cluster 3 took less medications and had less diagnoses .
2. For **age** and **A1C result** (fifth row, fourth column), pairplot shows cluster 3 is elderly patients with lower blood sugar level, while cluster 0 is young patients with high blood sugar level.

The visualisation helps us to profile the clusters as follow:

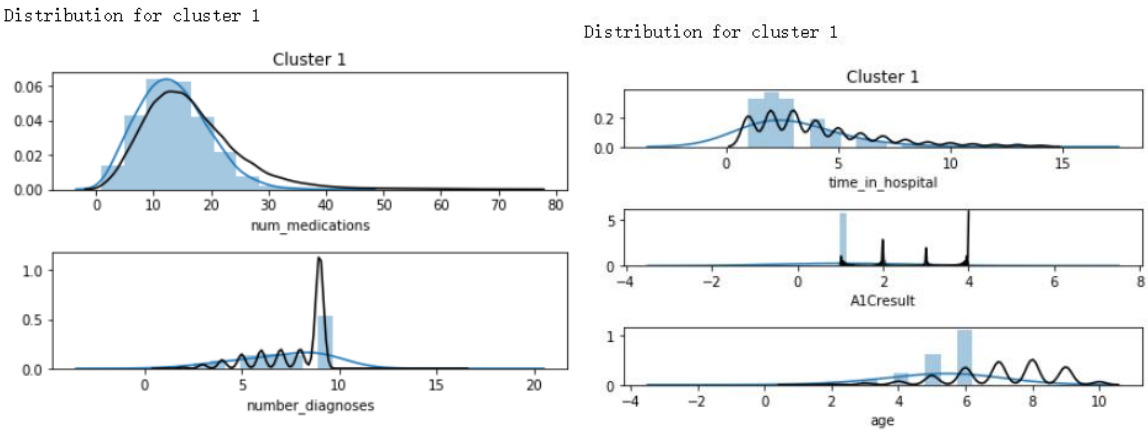
- Cluster0: high blood sugar level and low number of medications taken.
- Cluster1: youth patients with a lower number of medications taken and diagnoses.
- Cluster2: patients stay a long period in hospital.
- Cluster3: elderly patients with low blood sugar level.

b. Characterize the nature of each cluster by giving it a descriptive label and a brief description. Hint: use cluster distribution.

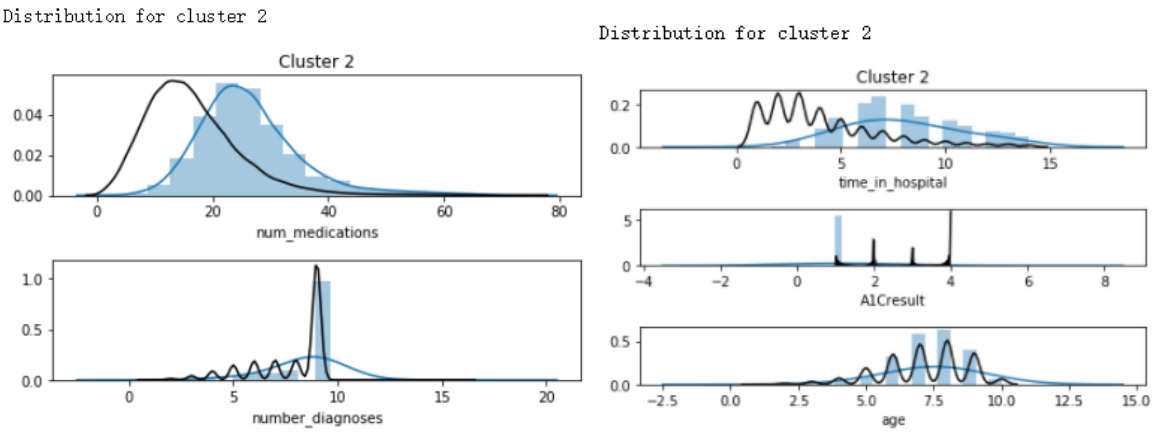
Here, we plot the distributions of cluster 0, cluster 1, cluster 2 and cluster 3 against the distributions from all data. The black lines are the distributions from all records, while light-blue lines are for a specific cluster. These plots show us the key characteristics of the clusters, as follows:



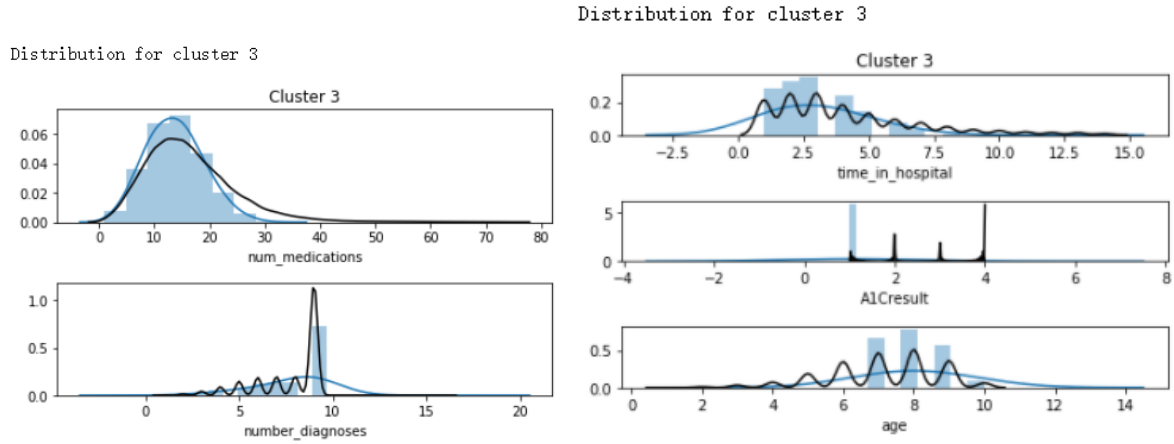
Cluster0: All attributes similar to the population.



Cluster1: Left leaning **num_medications** and **age**. Cluster 1 patients are low blood sugar level,young and taken slightly less medications.



Cluster2: Right leaning **time_in_hospital** and **num_medications.** Cluster 2 patients are low blood sugar level, stay a long period of time in hospital and take more medications.



Cluster3: Left leaning **A1C result** and **num_medications.** Cluster 3 are patients who have low blood sugar level and take less medication.

4. Build another clustering model using an algorithm that helps to profile the patients of specific races including Asian and Caucasian. Use the best setting (e.g., variable normalisations, optimal K, etc) obtained in the previous model. Answer the following:

a. List the attributes used in this analysis.

In order to profile the patients of Asian and Caucsian, we firstly fiter Asian and Caucasian. Next, we used the same attributes as the previous model including **'num_medications', 'number_diagnoses', 'time_in_hospital'**. And two categorical variables **A1C result** and **age**.

b. What difference do you see in this clustering interpretation when compared to the previous one?

We did not see any difference in these two clusterings. This may result from the majority of patients are Caucasian.

```
df['race'].value_counts()
```

```
Caucasian          28624
AfricanAmerican     5764
Hispanic             806
Other                696
Asian                293
Name: race, dtype: int64
```

Thus, although we filter other races out, we can not get a different result. If we want to profile other races we need more samples.

5. How can the outcome of this study be used by the relevant decision-makers?

Doctors can make treatment strategies for patients in different clusters based on the clustering results. In addition, hospitals can identify the type of diabetes and make improvements on this basis, which can reduce errors due to individuals.

Project (c): Predictive modelling using Decision Tree

1. What pre-processing was required on the dataset (D3.csv) before decision tree modelling?

Firstly, we change three variables from interval/integer to nominal/str. And then we filled the missing values with mode.

```
# change from interval/integer to nominal/str
df['admission_type_id'] = df['admission_type_id'].astype(str)
df['discharge_disposition_id'] = df['discharge_disposition_id'].astype(str)
df['admission_source_id'] = df['admission_source_id'].astype(str)

# denote errorneous values
df['race'].replace('', np.nan)
df['chlorpropamide'].replace('', np.nan)
df['age'].replace('', np.nan)

# using mode
df['race'].fillna(df['race'].mode(), inplace=True)
df['chlorpropamide'].fillna(df['chlorpropamide'].mode(), inplace=True)
df['age'].fillna(df['chlorpropamide'].mode(), inplace=True)
```

What distribution split between training and test datasets have you used?

We set test set as 30% and train set as 70%

```
# setting random state
rs = 10

X_mat = X.to_numpy()
X_train, X_test, y_train, y_test = train_test_split(X_mat, y, test_size=0.3, stratify=y, random_state=rs)
return df,X,y,X_train, X_test, y_train, y_test
```

2. Build a decision tree using the default setting. Answer the followings:

a. What is the classification accuracy of training and test datasets?

```
print("Train accuracy:", model.score(X_train, y_train))
```

```
Train accuracy: 1.0
```

```
print("Test accuracy:", model.score(X_test, y_test))
```

```
Test accuracy: 0.5714745653573728
```

b. What is the size of the tree (number of nodes and rules)?

In [7]:
```
n_nodes = model.tree_.node_count
print("The number of nodes are:")
print(n_nodes)
```

```
The number of nodes are:
19655
```

In [8]:
```
leaf = model.get_n_leaves()
print("The number of rules are:")
print(leaf)
```

```
The number of rules are:
9828
```

c. Which variable is used for the first split?

```
print(feature_names[1], ':', importances[1])
```

```
num_lab_procedures : 0.15855607389280404
```

d. What are the 5 important variables (in the order) in building the tree?

```
import numpy as np

# grab feature importances from the model and feature name from the original X
importances = model.feature_importances_
feature_names = X.columns

# sort them out in descending order
indices = np.argsort(importances)
indices = np.flip(indices, axis=0)

# limit to 5 features, you can leave this out to print out everything
indices = indices[:5]

for i in indices:
    print(feature_names[i], ':', importances[i])
```

```
num_lab_procedures : 0.15855607389280404
num_medications : 0.11691369607549702
number_inpatient : 0.06747650712495533
time_in_hospital : 0.066901282951523
num_procedures : 0.0444135794656724
```

e. What parameters have been used in building the tree? Detail them.

The above tree is using the default setting but we can improve the accuracy by three parameters:

1. **Criterion:** An algorithm for determining a split's quality. This model used the gini method.
2. **Max depth:** The tree's maximum depth. Deeper models are more complicated and contain a greater number of nodes. This model has no depth restrictions, which means it can perfectly fit the data (resulting in overfitting)
3. **Min samples leaf:** The minimum number of samples that must be present at a leaf node in order to limit the leaf node's minimum size. There is essentially no limitation on a node leaf in this model because it has a minimum sample leaf of one.

3. Build another decision tree tuned with GridSearchCV. Answer the followings:

a. What is the classification accuracy of training and test datasets?

```
cv_1.fit(X_train, y_train)

print("Train accuracy:", cv_1.score(X_train, y_train))
print("Test accuracy:", cv_1.score(X_test, y_test))
```

```
Train accuracy: 0.6476432277293299
Test accuracy: 0.6387636831938184
```

b. What is the size of the tree (i.e. number of nodes and rules)?

```
n_nodes = cv_1.tree_.node_count
print("The number of nodes are:")
print(n_nodes)
```

```
The number of nodes are:
221
```

```
leaf = cv_1.get_n_leaves()
print("The number of rules are:")
print(leaf)
```

```
The number of rules are:
111
```

c. Which variable is used for the first split?

```
from dm_tools import analyse_feature_importance, visualize_decision_tree

analyse_feature_importance(cv_1.best_estimator_, X.columns, 1)
visualize_decision_tree(cv_1.best_estimator_, X.columns, "optimal_tree.png")
```

```
number_inpatient : 0.5251164317425324
```

d. What are the 5 important variables (in the order) in building the tree?

```
from dm_tools import analyse_feature_importance, visualize_decision_tree

analyse_feature_importance(cv_1.best_estimator_, X.columns, 5)
visualize_decision_tree(cv_1.best_estimator_, X.columns, "optimal_tree.png")
```

```
number_inpatient : 0.5251164317425324
discharge_disposition_id_11 : 0.12595951186119314
number_emergency : 0.051277435893517485
number_outpatient : 0.04072885376284951
diabetesMed : 0.029781302282978147
```

e. Report if you see any evidence of model overfitting.

As we did not set the max depth for the first tree we can find a 100% accuracy in the train set but lower accuracy in the test set.

```
print("Train accuracy:", model.score(X_train, y_train))
```

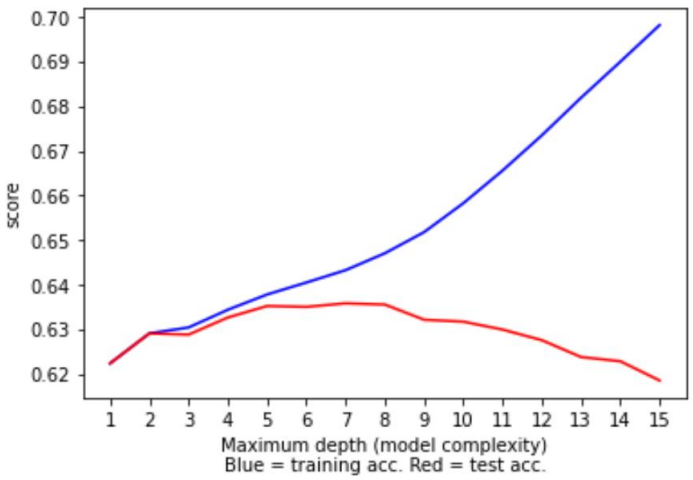
```
Train accuracy: 1.0
```

```
print("Test accuracy:", model.score(X_test, y_test))
```

```
Test accuracy: 0.5714745653573728
```

Next, we focus only on the max_depth to understand the bias and variance for the fixed criterion = gini and max_sample_leaf = 10.

```
import matplotlib.pyplot as plt
result_set['params']
dd = pd.DataFrame(result_set['params'])

index_ = list(dd.index[(dd['criterion']=='gini') & (dd['min_samples_leaf']==10)])
train_result = result_set['mean_train_score']
test_result = result_set['mean_test_score']

max_depth_train = []
max_depth_test = []
index_
for i in range(len(index_)):
    max_depth_train.append(train_result[index_[i]])
    max_depth_test.append(test_result[index_[i]])
plt.plot(range(1, len(max_depth_train)+1), max_depth_train, 'b', range(1,len(max_depth_test)+1), max_depth_test, 'r')
plt.xlabel('Maximum depth (model complexity)\nBlue = training acc. Red = test acc.')
plt.xticks(np.arange(1, len(max_depth_train)+1, 1))
plt.ylabel('score')
plt.show()
```

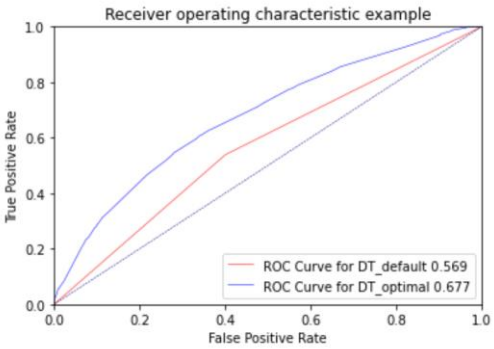


From the above plot we found overfitting appears after depth = 8.

4. What differences do you observe between these two decision tree models (with and without fine-tuning)? How do they compare performance-wise? Produce the ROC curve for both DTs. Explain why those changes may have happened.

```
from sklearn.metrics import roc_auc_score

y_pred_proba_dt = model.predict_proba(X_test)
y_pred_proba_dt_cv = dt_cv_best.predict_proba(X_test)

roc_index_dt = roc_auc_score(y_test, y_pred_proba_dt[:, 1])
roc_index_dt_cv = roc_auc_score(y_test, y_pred_proba_dt_cv[:, 1])

print("ROC index on test for DT_default:", roc_index_dt)
print("ROC index on test for DT_optimal:", roc_index_dt_cv)
```

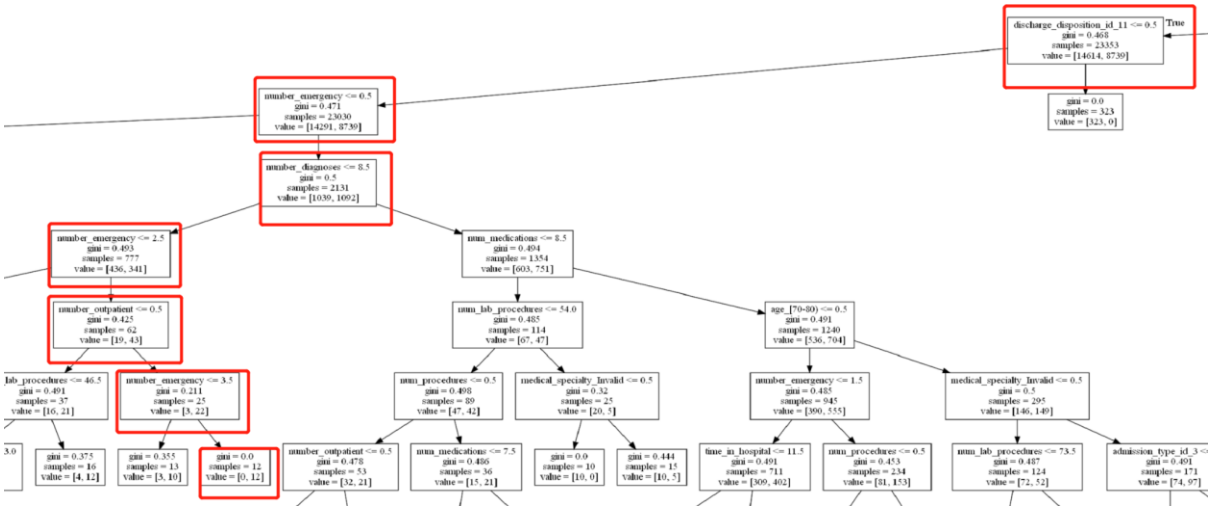
```
ROC index on test for DT_default: 0.5690761587471861
ROC index on test for DT_optimal: 0.6774909097184696
```



From the ROC score we can see the score of DT_CV with fine-tuning is the highest (0.677). And the DT without fine-tuning gets a low score (0.569).

In addition, we also need to see the performance of different model's curves. The closer the curve is to the top and left corner, the better the model is.The DT with GridSearchCV (DT_CV) shows the largest curve area compared to the DT with default settings (DT). Thus DT_CV is the best performing model.

5. From the better model, can you identify which patients could potentially be "readmitted"? Can you provide general characteristics of those patients?



From the optimal decision tree, we can identify patients whose discharge_disposition id is not 11(Expired), emergency visits of the patient in the year preceding the encounter high than 4, outpatient visits higher than 0 of the patient, and diagnoses entered to the system lower than 9 are more likely to be "readmitted".

- Discharge_disposition is not "Expired"
- emergency visits >= 4
- outpatient visits > 0
- Diagnoses < 9

Predictive modelling using Regression

1. What pre-processing was required on the dataset before regression modelling?

Firstly, we change three variables from interval/integer to nominal/str. And then we filled the missing values with mode.

```
# change from interval/integer to nominal/str
df['admission_type_id'] = df['admission_type_id'].astype(str)
df['discharge_disposition_id'] = df['discharge_disposition_id'].astype(str)
df['admission_source_id'] = df['admission_source_id'].astype(str)

# denote errorneous values
df['race'].replace('', np.nan)
df['chlorpropamide'].replace('', np.nan)
df['age'].replace('', np.nan)

# using mode
df['race'].fillna(df['race'].mode(), inplace=True)
df['chlorpropamide'].fillna(df['chlorpropamide'].mode(), inplace=True)
df['age'].fillna(df['chlorpropamide'].mode(), inplace=True)
```

What distribution split between training and test datasets have you used?

We set test set as 30% and train set as 70%

```
# setting random state
rs = 10

X_mat = X.to_numpy()
X_train, X_test, y_train, y_test = train_test_split(X_mat, y, test_size=0.3, stratify=y, random_state=rs)
return df,X,y,X_train, X_test, y_train, y_test
```

2. Build a regression model using the default regression method with all inputs. Build another regression model tuned with GridSearchCV. Now, choose a better model to answer the followings:

a. Explain why you chose that model.

The score of train data and test data in default model:

```
model.fit(X_train, y_train)

print("Train accuracy:", model.score(X_train, y_train))
print("Test accuracy:", model.score(X_test, y_test))
```

```
Train accuracy: 0.6483883430842257
Test accuracy: 0.6476497102382486
```

The score of train data and test data in the model tuned with GridSearchCV:

```
cv.fit(X_train, y_train)

print("Train accuracy:", cv.score(X_train, y_train))
print("Test accuracy:", cv.score(X_test, y_test))
```

```
Train accuracy: 0.6479191963792913
Test accuracy: 0.6475209272376047
```

Firstly, we notice that the default model's train accuracy is slightly higher than the test accuracy, which could indicate slight overfitting, but this needs to be investigated further. We used GridSearchCV to fine-tune this logistic regression model to improve its accuracy. However, by comparing the accuracy of two models we choose the default model as it provides a slightly better accuracy in the test set.

b. Name the regression function used.

Logistic regression

As we are dealing with classification predictive mining, therefore, the logistic regression model is the best regression function.

c. Did you apply standardization of variables? Why would you normalise the variables for regression mining?

```
from sklearn.preprocessing import StandardScaler

# initialise a standard scaler object
scaler = StandardScaler()

# visualise min, max, mean and standard dev of data before scaling
print("Before scaling\n-------------")
for i in range(5):
    col = X_train[:,i]
    print("Variable #{}: min {}, max {}, mean {:.2f} and std dev {:.2f}".
          format(i, min(col), max(col), np.mean(col), np.std(col)))

# learn the mean and std.dev of variables from training data
# then use the learned values to transform training data
X_train = scaler.fit_transform(X_train, y_train)

print("After scaling\n-------------")
for i in range(5):
    col = X_train[:,i]
    print("Variable #{}: min {}, max {}, mean {:.2f} and std dev {:.2f}".
          format(i, min(col), max(col), np.mean(col), np.std(col)))

# use the statistic that you learned from training to transform test data
# NEVER learn from test data, this is supposed to be a set of dataset
# that the model has never seen before
X_test = scaler.transform(X_test)
```

```
Before scaling
-------------
Variable #0: min 1, max 14, mean 4.23 and std dev 2.86
Variable #1: min 1, max 132, mean 43.76 and std dev 19.93
Variable #2: min 0, max 6, mean 1.31 and std dev 1.72
Variable #3: min 1, max 75, mean 16.70 and std dev 8.01
Variable #4: min 0, max 42, mean 0.51 and std dev 1.52
After scaling
-------------
Variable #0: min -1.1282939178870754, max 3.4191321542486945, mean 0.00 and std dev 1.00
Variable #1: min -2.1456388744174166, max 4.427803373304037, mean 0.00 and std dev 1.00
Variable #2: min -0.7597114725238956, max 2.7229762266823645, mean -0.00 and std dev 1.00
Variable #3: min -1.9607230541112892, max 7.279995173622201, mean -0.00 and std dev 1.00
Variable #4: min -0.3360252938779217, max 27.226814366375155, mean -0.00 and std dev 1.00
```

Yes. Because input variables on different scales make comparison between data points difficult.

d. Report the variables included in the regression model.

The detailed variables are: race, gender, age, admission_type_id, discharge_disposition_id, admission_source_id, time_in_hospital, medical_specialty, num_lab_procedures, num_procedures, num_medications, number_outpatient, number_emergency, number_inpatient, number_diagnoses, max_glu_serum, A1C result, metformin, repaglinide, nateglinide,

chlorpropamide, glimepiride, acetohexamide, glipizide, glyburide, tolbutamide, insulin, change, diabetesMed.
Target value is readmitted.

e. Report the top-5 important variables (in the order) in the model.

The top-5 important variables are as follows:

```
discharge_disposition_id_11 : -1.0872634195504072
number_inpatient : 0.5348585476887653
number_emergency : 0.24564818376953979
age_[70-80) : 0.21407647045229386
age_[80-90) : 0.20589738979956457
```

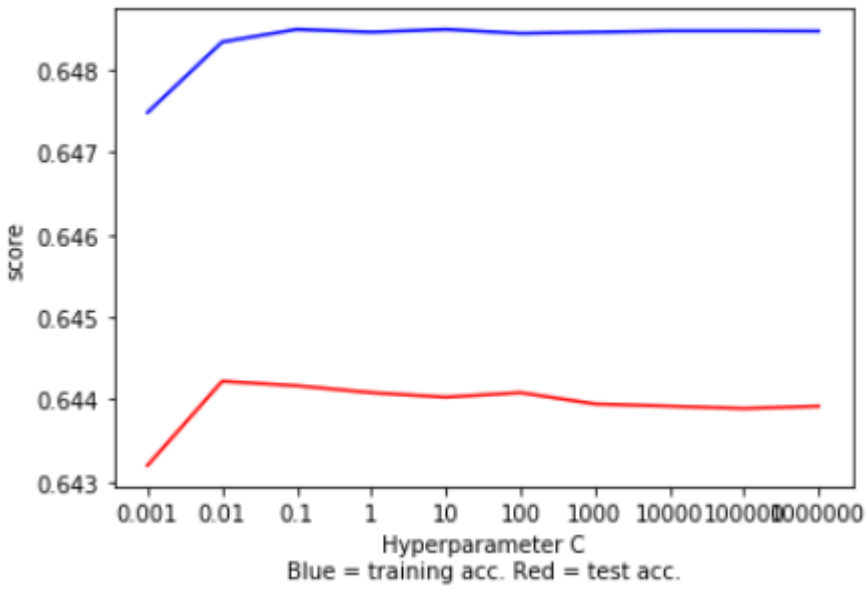
- Discharge_disposition_id_11: Negative coefficient
- Number_inpatient: Positive coefficient
- Number_emergency: Positive coefficient
- Age_[70 - 80): Positive coefficient
- Age_[80 - 90): Positive coefficient

f. What is the classification accuracy on training and test datasets?

```
print("Train accuracy:", model.score(X_train, y_train))
print("Test accuracy:", model.score(X_test, y_test))
```

```
Train accuracy: 0.6483883430842257
Test accuracy: 0.6476497102382486
```

g. Report any sign of overfitting in this model.



From the plot above, it shows that the model performs best when C = 0.01, and overfitting starts at this point. The gap between test accuracy and training accuracy gradually increases after 0.01.

3. Build another regression model on the reduced variables set. Perform dimensionality reduction with Recursive feature elimination. Tune the model with GridSearchCV to find the best parameter setting. Answer the followings:

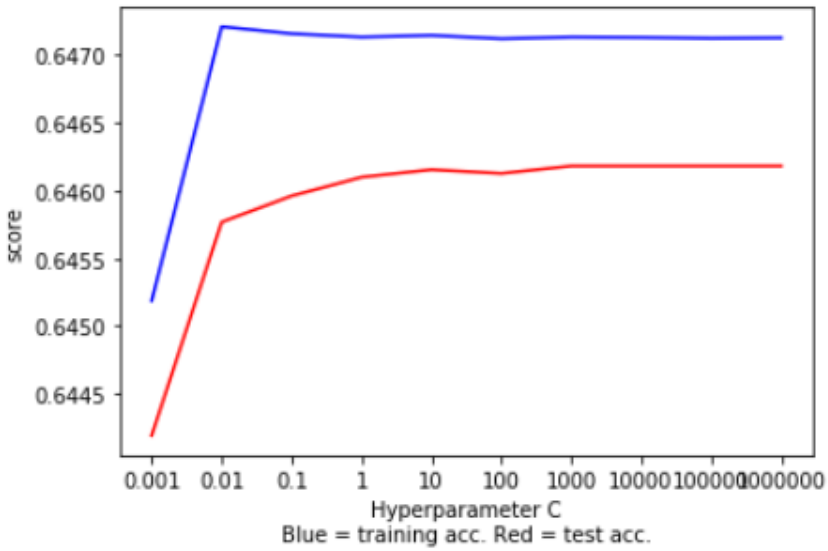
a. Was dimensionality reduction useful to identify a good feature set for building the accurate model?

RFE works by first training the model on all features. Each feature is assigned a weight. Features with small weights (less important) are eliminated, making a smaller feature set. This process is repeated a number of times until reaching the optimal performance. Therefore, it is useful to identify a good feature set for building an accurate model.

b. What is the classification accuracy on training and test datasets?

```
Train accuracy: 0.6470912904294073
Test accuracy: 0.6443013522215068
```

c. Report any sign of overfitting.



From the above plot, we could not find any sign of overfitting. The model is trained well after c=0.1.

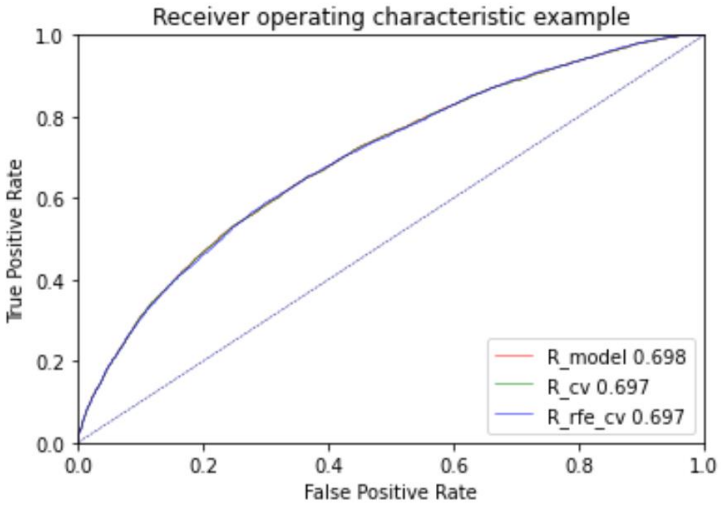
d. Report the top-3 important variables (in the order) in the model.

```
coef = rfe_cv.best_estimator_.coef_[0]
feature_names = X.columns

# sort them out in descending order
indices = np.argsort(np.absolute(coef))
indices = np.flip(indices, axis=0)

# limit to 3 features, you can leave this out to print out everything
indices = indices[:3]

for i in indices:
    print(feature_names[i], ':', coef[i])
```

```
age_[0-10) : -1.888023333718872
num_medications : 0.5421601695751735
num_procedures : 0.24523465898132044
```

4. Produce the ROC curve for all different regression models. Using the best regression model, can you identify which patients could potentially be "readmitted"? Can you provide general characteristics of those patients?

```
ROC index on test for R_model: 0.6976573013934382
ROC index on test for R_cv: 0.6974257882928443
ROC index on test for R_rfe_cv: 0.6970013503901624
```



From the ROC result, we choose the default regression model.

```
discharge_disposition_id_11 : -1.0872634195504072
number_inpatient : 0.5348585476887653
number_emergency : 0.24564818376953979
age_[70-80) : 0.21407647045229386
age_[80-90) : 0.20589738979956457
```

By the top 5 features shown above, we know patients who have elder age and a large number of inpatient visits and emergency visits will probably be readmitted.

Predictive modelling using Neural Networks

1. What pre-processing was required on the dataset before neural network modelling? What distribution split between training and test datasets have you used?

We reuse the data preparation function we builded in the previous model.

```
# load the data
df,X,y,X_train, X_test, y_train, y_test = data_prep()
```

2. Build a Neural Network model using the default setting. Answer the following:

a. Explain the parameters used in building this model, e.g., network architecture, iterations, activation function, etc.

In the first MLPClassifier, we build the model without any additional parameter except the random state.

network architecture: it is formed in three layers, called the input layer, hidden layer, and output layer.

iterations：describes the number of times a batch of data passed through the algorithm.

activation function：it is a function that converts all negative numbers to zero.

Solver: the solver for weight optimization.

The parameters of the default setting as below:

```
model_1.get_params()
```

```
{'activation': 'relu',
 'alpha': 0.0001,
 'batch_size': 'auto',
 'beta_1': 0.9,
 'beta_2': 0.999,
 'early_stopping': False,
 'epsilon': 1e-08,
 'hidden_layer_sizes': (100,),
 'learning_rate': 'constant',
 'learning_rate_init': 0.001,
 'max_fun': 15000,
 'max_iter': 200,
 'momentum': 0.9,
 'n_iter_no_change': 10,
 'nesterovs_momentum': True,
 'power_t': 0.5,
 'random_state': 10,
 'shuffle': True,
 'solver': 'adam',
 'tol': 0.0001,
 'validation_fraction': 0.1,
 'verbose': False,
 'warm_start': False}
```

b. What is the classification accuracy on training and test datasets?

```
model_1 = MLPClassifier(random_state=rs)
model_1.fit(X_train, y_train)

print("Train accuracy:", model_1.score(X_train, y_train))
print("Test accuracy:", model_1.score(X_test, y_test))

y_pred = model_1.predict(X_test)
print(classification_report(y_test, y_pred))

print(model_1)
```

```
Train accuracy: 0.7796114361408544
Test accuracy: 0.6083708950418545
```

c. Did the training process converge and result in the best model?

No, the training process did not converge and result in the best model. In sklearn, if a neural network does not achieve convergence before maximum iteration, it will raise a "convergence is not reached" warning message，like the below picture. And then,

the `max_iter` hyperparameter of the neural network should be increased. We set the `max_iter` as 700.
The training dataset shows the accuracy of 0.7796 while the testing dataset shows 0.60837 (lower), leading to overfitting to the training data. It is necessary to notice a convergence warning. Also, the neural network after setting the iteration of 700 performed with a little higher accuracy on the training dataset(0.7897), but test accuracy remains the same. This also indicates that overfitting to the training data.

```
model_1 = MLPClassifier(random_state=rs)
model_1.fit(X_train, y_train)

print("Train accuracy:", model_1.score(X_train, y_train))
print("Test accuracy:", model_1.score(X_test, y_test))

y_pred = model_1.predict(X_test)
print(classification_report(y_test, y_pred))

print(model_1)
```

```
Train accuracy: 0.7796390330058506
Test accuracy: 0.6088860270444302
              precision    recall  f1-score   support

           0       0.63      0.68      0.65      8382
           1       0.58      0.53      0.55      7148

    accuracy                           0.61     15530
   macro avg       0.61      0.60      0.60     15530
weighted avg       0.61      0.61      0.61     15530

MLPClassifier(random_state=10)
```

```
C:\Users\86186\Anaconda3\lib\site-packages\sklearn\neural_network\_multilayer_perceptron.py:617: ConvergenceWarning: Stochasti
c Optimizer: Maximum iterations (200) reached and the optimization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
```

```
model_2 = MLPClassifier(max_iter=700, random_state=rs)
model_2.fit(X_train, y_train)

print("Train accuracy:", model_2.score(X_train, y_train))
print("Test accuracy:", model_2.score(X_test, y_test))

y_pred = model_2.predict(X_test)
print(classification_report(y_test, y_pred))

print(model_2)
```

```
Train accuracy: 0.7897118887294403
Test accuracy: 0.6025112685125563
```

3. Refine this network by tuning it with GridSearchCV. Report the trained model.
a. Explain the parameters used in building this model, e.g., network architecture, iterations, activation function, etc.

network architecture: it is formed in three layers, called the input layer, hidden layer, and output layer.
Iterations: describes the number of times a batch of data passed through the algorithm.
activation function: it is a function that converts all negative numbers to zero.
Solver: the solver for weight optimization.
param_grid: is the dictionary with parameters names (str) as keys and lists of parameter settings to try as values.

The parameters of this model setting as below：

```
cv_3.get_params()
```

```
{'cv': 10,
 'error_score': nan,
 'estimator__activation': 'relu',
 'estimator__alpha': 0.0001,
 'estimator__batch_size': 'auto',
 'estimator__beta_1': 0.9,
 'estimator__beta_2': 0.999,
 'estimator__early_stopping': False,
 'estimator__epsilon': 1e-08,
 'estimator__hidden_layer_sizes': (100,),
 'estimator__learning_rate': 'constant',
 'estimator__learning_rate_init': 0.001,
 'estimator__max_fun': 15000,
 'estimator__max_iter': 200,
 'estimator__momentum': 0.9,
 'estimator__n_iter_no_change': 10,
 'estimator__nesterovs_momentum': True,
 'estimator__power_t': 0.5,
 'estimator__random_state': 10,
 'estimator__shuffle': True,
 'estimator__solver': 'adam',
 'estimator__tol': 0.0001,
 'estimator__validation_fraction': 0.1,
 'estimator__verbose': False,
 'estimator__warm_start': False,
 'estimator': MLPClassifier(random_state=10),
 'n_jobs': -1,
 'param_grid': {'hidden_layer_sizes': [(4,), (6,), (8,), (10,)],
  'alpha': [0.01, 0.001, 0.0001, 1e-05]},
 'pre_dispatch': '2*n_jobs',
 'refit': True,
 'return_train_score': False,
 'scoring': None,
 'verbose': 0}
```

We used **GridSearchCV** to find the best hidden layers size and alpha rate.
Hidden layers size: the number of neurons exist in the hidden layers.
Alpha rate: A momentum item used to update weights during training. The momentum is intended to move the weight change in a consistent direction.
We can see there are 168 input features.

```
print(X_train.shape)
```

```
(36236, 168)
```

Therefore,we will start tuning with one hidden layer of 8 to 168 neurons, increment of 20.

```
]: params = {'hidden_layer_sizes': [(x,) for x in range(8, 169, 20)]}

   cv_1 = GridSearchCV(param_grid=params, estimator=MLPClassifier(random_state=rs),return_train_score=True, cv=10, n_jobs=-1)
   cv_1.fit(X_train, y_train)
```

```
]: GridSearchCV(cv=10, estimator=MLPClassifier(random_state=10), n_jobs=-1,
               param_grid={'hidden_layer_sizes': [(8,), (28,), (48,), (68,),
                                                  (88,), (108,), (128,), (148,),
                                                  (168,)]},
               return_train_score=True)
```

```
1]: print("Train accuracy:", cv_1.score(X_train, y_train))
    print("Test accuracy:", cv_1.score(X_test, y_test))

    y_pred = cv_1.predict(X_test)
    print(classification_report(y_test, y_pred))
    print(cv_1.best_params_)
```

```
Train accuracy: 0.6688100231813666
Test accuracy: 0.6402446877012234
              precision    recall  f1-score   support

           0       0.65      0.72      0.68      8382
           1       0.63      0.55      0.58      7148

    accuracy                           0.64     15530
   macro avg       0.64      0.63      0.63     15530
weighted avg       0.64      0.64      0.64     15530

{'hidden_layer_sizes': (8,)}
```

We found the optimal number of neurons is 8. Next, we attempted to tune the model with the lower number of neurons in the hidden layer and try to find the best alpha rate.

```
: params = {'hidden_layer_sizes': [(4,), (6,), (8,), (10,)], 'alpha': [0.01,0.001, 0.0001, 0.00001]}

  cv_3 = GridSearchCV(param_grid=params, estimator=MLPClassifier(random_state=rs), cv=10, n_jobs=-1)
  cv_3.fit(X_train, y_train)

  print("Train accuracy:", cv_3.score(X_train, y_train))
  print("Test accuracy:", cv_3.score(X_test, y_test))

  y_pred = cv_3.predict(X_test)
  print(classification_report(y_test, y_pred))

  print(cv_3.best_params_)
```

```
Train accuracy: 0.6644497185119771
Test accuracy: 0.6462974887314874
              precision    recall  f1-score   support

           0       0.66      0.72      0.69      8382
           1       0.63      0.56      0.59      7148

    accuracy                           0.65     15530
   macro avg       0.64      0.64      0.64     15530
weighted avg       0.64      0.65      0.64     15530

{'alpha': 1e-05, 'hidden_layer_sizes': (6,)}
```

Finally, we found the best alpha rate is 1e-05.

b. What is the classification accuracy on training and test datasets?

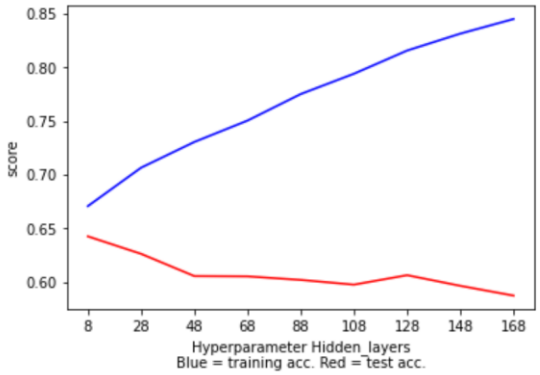
```
3]: params = {'hidden_layer_sizes': [(4,), (6,), (8,), (10,)], 'alpha': [0.01,0.001, 0.0001, 0.00001]}

  cv_3 = GridSearchCV(param_grid=params, estimator=MLPClassifier(random_state=rs), cv=10, n_jobs=-1)
  cv_3.fit(X_train, y_train)

  print("Train accuracy:", cv_3.score(X_train, y_train))
  print("Test accuracy:", cv_3.score(X_test, y_test))

  y_pred = cv_3.predict(X_test)
  print(classification_report(y_test, y_pred))

  print(cv_3.best_params_)
```

```
Train accuracy: 0.6644497185119771
Test accuracy: 0.6462974887314874
              precision    recall  f1-score   support

           0       0.66      0.72      0.69      8382
           1       0.63      0.56      0.59      7148

    accuracy                           0.65     15530
   macro avg       0.64      0.64      0.64     15530
weighted avg       0.64      0.65      0.64     15530

{'alpha': 1e-05, 'hidden_layer_sizes': (6,)}
```

c. Did the training process converge and result in the best model?

Yes, the training process converged and resulted in the best model. In sklearn, if a neural network does not achieve convergence before maximum iteration, it will raise a "convergence is not reached" warning message. We don't receive the warning message "Maximum iterations (200) reached and the optimization hasn't converged yet." It means the training process converged.

d. Do you see any sign of over-fitting?



It is obvious that the plot above shows the sign of over-fitting from the hidden_layer_sizes as 8.

4. Let us see if feature selection helps in improving the model? Build another Neural Network model with reduced features set. Perform dimensionality reduction by selecting variables with a decision tree (use the best decision tree model that you have built in the previous modelling task). Tune the model with GridSearchCV to find the best parameters setting.

Answer the following:

a. Did feature selection favour the outcome? Any change in network architecture? What inputs are being used as the network input?

Yes, the Neural Network model trained with decision tree selected variables manage to improve model performance.

The change in network architecture is the parameter 'hidden_layer_sizes' changed range from (8, 169) to (4, 6, 8, 10). We tune the parameter of the default model and find the best parameter(hidden layer size: 10).

```
params = {'hidden_layer_sizes': [(x,) for x in range(8, 169, 20)]}
```

```
params = {'hidden_layer_sizes': [(4,), (6,), (8,), (10,)]}
```

```
params = {'hidden_layer_sizes': [(4,), (6,), (8,), (10,)], 'alpha': [0.01, 0.001, 0.0001, 0.00001]}

cv_sel_model = GridSearchCV(param_grid=params, estimator=MLPClassifier(random_state=rs), cv=10, n_jobs=-1, return_train_score=True)
cv_sel_model.fit(X_train_sel_model, y_train)

print("Train accuracy:", cv_sel_model.score(X_train_sel_model, y_train))
print("Test accuracy:", cv_sel_model.score(X_test_sel_model, y_test))

y_pred = cv_sel_model.predict(X_test_sel_model)
print(classification_report(y_test, y_pred))

print(cv_sel_model.best_params_)
```

```
Train accuracy: 0.648664311734187
Test accuracy: 0.6478428847392145
              precision    recall  f1-score   support

           0       0.65      0.75      0.70      8382
           1       0.64      0.53      0.58      7148

    accuracy                           0.65     15530
   macro avg       0.65      0.64      0.64     15530
weighted avg       0.65      0.65      0.64     15530

{'alpha': 0.0001, 'hidden_layer_sizes': (10,)}
```

This model identifies the set of 18 variables as the important features.

```
from sklearn.feature_selection import SelectFromModel

selectmodel = SelectFromModel(dt_best.best_estimator_, prefit=True)
X_train_sel_model = selectmodel.transform(X_train)
X_test_sel_model = selectmodel.transform(X_test)

print(X_train_sel_model.shape)
```

```
(36236, 18)
```

b. What is the classification accuracy on training and test datasets?

```
: params = {'hidden_layer_sizes': [(4,), (6,), (8,), (10,)], 'alpha': [0.01,0.001, 0.0001, 0.00001]}

cv_sel_model = GridSearchCV(param_grid=params, estimator=MLPClassifier(random_state=rs), cv=10, n_jobs=-1)
cv_sel_model.fit(X_train_sel_model, y_train)

print("Train accuracy:", cv_sel_model.score(X_train_sel_model, y_train))
print("Test accuracy:", cv_sel_model.score(X_test_sel_model, y_test))

y_pred = cv_sel_model.predict(X_test_sel_model)
print(classification_report(y_test, y_pred))

print(cv_sel_model.best_params_)
```

```
Train accuracy: 0.648664311734187
Test accuracy: 0.6478428847392145
              precision    recall  f1-score   support

           0       0.65      0.75      0.70      8382
           1       0.64      0.53      0.58      7148

    accuracy                           0.65     15530
   macro avg       0.65      0.64      0.64     15530
weighted avg       0.65      0.65      0.64     15530

{'alpha': 0.0001, 'hidden_layer_sizes': (10,)}
```

c. How many iterations are now needed to train this network?

200 or less than 200. We use the default set of the max iteration, and we don’t receive the warning message “Maximum iterations (200) reached and the optimization hasn't converged yet.”

```
cv_sel_model.get_params()
```

```
{'cv': 10,
 'error_score': nan,
 'estimator__activation': 'relu',
 'estimator__alpha': 0.0001,
 'estimator__batch_size': 'auto',
 'estimator__beta_1': 0.9,
 'estimator__beta_2': 0.999,
 'estimator__early_stopping': False,
 'estimator__epsilon': 1e-08,
 'estimator__hidden_layer_sizes': (100,),
 'estimator__learning_rate': 'constant',
 'estimator__learning_rate_init': 0.001,
 'estimator__max_fun': 15000,
 'estimator__max_iter': 200,
 'estimator__momentum': 0.9,
 'estimator__n_iter_no_change': 10,
 'estimator__nesterovs_momentum': True,
 'estimator__power_t': 0.5,
 'estimator__random_state': 10,
 'estimator__shuffle': True,
 'estimator__solver': 'adam',
 'estimator__tol': 0.0001,
 'estimator__validation_fraction': 0.1,
 'estimator__verbose': False,
 'estimator__warm_start': False,
 'estimator': MLPClassifier(random_state=10),
 'iid': 'deprecated',
 'n_jobs': -1,
 'param_grid': {'hidden_layer_sizes': [(4,), (6,), (8,), (10,)],
  'alpha': [0.01, 0.001, 0.0001, 1e-05]},
 'pre_dispatch': '2*n_jobs',
 'refit': True,
 'return_train_score': False,
 'scoring': None,
 'verbose': 0}
```

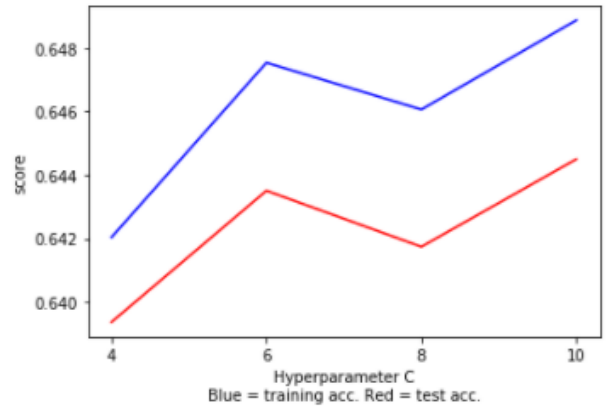
d. Do you see any sign of over-fitting? Did the training process converge and result in the best model?

```
params = {'hidden_layer_sizes': [(4,), (6,), (8,), (10,)]}

cv_sel_model = GridSearchCV(param_grid=params, estimator=MLPClassifier(random_state=rs), return_train_score=True, cv=10, n_job
cv_sel_model.fit(X_train_sel_model, y_train)

print("Train accuracy:", cv_sel_model.score(X_train_sel_model, y_train))
print("Test accuracy:", cv_sel_model.score(X_test_sel_model, y_test))

y_pred = cv_sel_model.predict(X_test_sel_model)
print(classification_report(y_test, y_pred))

print(cv_sel_model.best_params_)
```

```
Train accuracy: 0.648664311734187
Test accuracy: 0.6478428847392145
              precision    recall  f1-score   support

           0       0.65      0.75      0.70      8382
           1       0.64      0.53      0.58      7148

    accuracy                           0.65     15530
   macro avg       0.65      0.64      0.64     15530
weighted avg       0.65      0.65      0.64     15530

{'hidden_layer_sizes': (10,)}
```

We used GridSearchCV to find the best hidden_layers size is 10(4-10).

```
Total number of models:  4
```



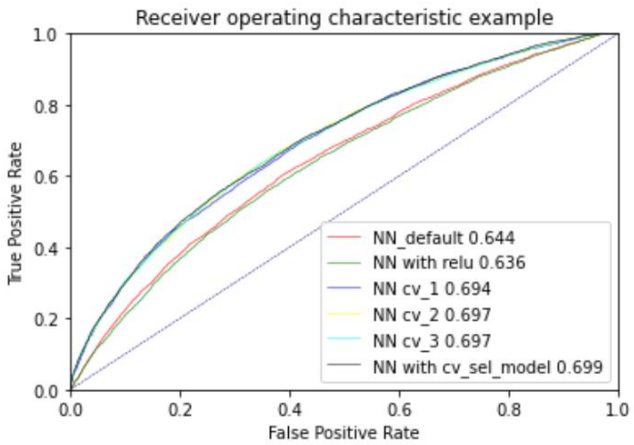
From the above plot, we did not find any sign of over-fitting.

5. Produce the ROC curve for all different NNs. Now, using the best neural network model, can you provide general characteristics of the patients identified by the model? If it is difficult (or even infeasible) to comprehend, discuss why?

According to the ROC curve below, the neural network with feature selection(a decision tree) and grid search is the best model(ROC: 0.699).

```
fpr_nn_1, tpr_nn_1, thresholds_nn_1 = roc_curve(y_test, y_pred_proba_nn_1[:,1])
fpr_nn_2, tpr_nn_2, thresholds_nn_2 = roc_curve(y_test, y_pred_proba_nn_2[:,1])
fpr_cv_1, tpr_cv_1, thresholds_cv_1 = roc_curve(y_test, y_pred_proba_cv_1[:,1])
fpr_cv_2, tpr_cv_2, thresholds_cv_2 = roc_curve(y_test, y_pred_proba_cv_2[:,1])
fpr_cv_3, tpr_cv_3, thresholds_cv_3 = roc_curve(y_test, y_pred_proba_cv_3[:,1])
fpr_cv_sel_model, tpr_cv_sel_model, thresholds_cv_sel_model = roc_curve(y_test, y_pred_proba_cv_sel_model[:,1])
```

```
ROC index on test for NN_default: 0.6444320873986239
ROC index on test for NN with relu: 0.6357697904895734
ROC index on test for NN with gridsearch 1: 0.6944341853869986
ROC index on test for NN with gridsearch 2: 0.6967134569814577
ROC index on test for NN with gridsearch 3: 0.6967561427831136
ROC index on test for NN with feature selection (model selection) and gridsearch: 0.6989094098967903
```

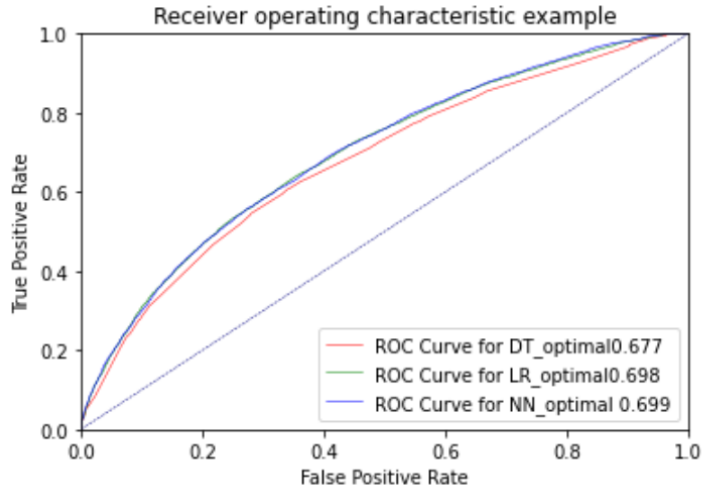


No, because neural networks can only predict readmission based on the features of patients, it is difficult to directly get general features of such patients from the neural network. There is no straightforward relationship between the weights and the estimated function.

Final remarks: Decision making

1. Finally, based on all models and analysis, is there a model you will use in decision making? Justify your choice. Draw a ROC chart and Accuracy Table to support your findings.

We will use the Neural Network model in decision making. From the ROC chart and accuracy table below, the Neural NetWork has the highest ROC score(0.699) and accuracy.



	Train accuracy	Test accuracy
Decision Tree	**0.6476432277293299**	**0.6387636831938184**
Logistic Regression	**0.6483883430842257**	**0.6476497102382486**
Neural Network	**0.648664311734187**	**0.6478428847392145**

2. Can you summarise the positives and negatives of each predictive modelling method based on this analysis?

	Positives	Negatives

Decision tree	It takes less time to run the code(less amount of computation).	Easy to be overfitting.
Logistic Regression	Easy to display each feature's impact (coefficients) to the target variable. Less likely to be overfitting	The missing values and noise need to be processed in the dataset.
Neural Network	Can build a more accurate model compared to other methods.	Large amounts of computation (take a long time to run the code) for training. Easy to be overfitting Hard to know the details of variables in the models (Black-box).