IFN 509: Data Exploration and Mining
Assessment 2

Team Name: Dream Team

Group No.3

Student Name	Student Id
Jackson Calvert-Lane	n10787003
Bella Qian	n10661450
Darren Ross	n10788557
Brendan Wallace-Nash	n9993304

	Jackson	Bella	Darren	Brendan
Jackson	<100 %>	<100 %>	< 100%>	<100 %>
Bella	<100 %>	<100 %>	<100 %>	<100 %>
Darren	<100 %>	<100 %>	<100 %>	<100 %>
Brendan	<100 %>	<100 %>	<100 %>	<100 %>

Project A)

Association Mining to Find Hotspots Based on a Patient Rout Data

Prepossessing Patient Route Data

There was very minimal pre-processing needed for the Patient Route Data. Data in the global_num was dropped as 569 of its 575 inputs were null. It was also noted that there were only 911 unique patient id's instead of the 913 that were stated in the report. This could mean that 2 patients were excluded from the data or that there are two patients that share the same patient ID's with existing patients. This anomaly in in the data while small, should be considered in interoperating the results. Latitude and longitude were also drop from the data, as location data was deemed the more suitable metric for mapping patients travel routes and longitude had appeared to be missing a value.

Table 1: Prepossessing Analysis

Variable	Expected Unique Values	Actual Unique Values	Nun Values in data
Patient ID	913	911	0
Global num	913	575	596
Date	n/a	93	0
Location	151	151	0
Latitude	151	151	0
Longitude	151	150	0

Figure 1: Unique Variables of Data

```
for col in df:
    print(df[col].nunique())
```

```
911
575
93
151
151
150
```

Figure 2: Sum of Null Values in Data

```
df.isnull().sum()
#so there is 569 nan values in global which is more then half
#so im going to drop it
```

```
patient_id      0
global_num    569
date            0
location        0
latitude        0
longitude       0
dtype: int64
```

```
df = df.drop(columns=['global_num', 'latitude', 'longitude'])
df
```

	patient_id	date	location
0	P1000000501	22/04/2020	Chittorgarh_Rajasthan
1	P1000000501	24/04/2020	Ratnagiri_Maharashtra
2	P1000000502	26/04/2020	Pindwara_Rajasthan
3	P1000000502	27/04/2020	Raipur_Chhattisgarh
4	P1000000502	28/04/2020	Gokak_Karnataka
...	...	...	...
1539	P9011000510	19/05/2020	Sagar_Karnataka
1540	P9011000510	19/05/2020	Jorapokhar_Jharkhand
1541	P9014000503	27/05/2020	Rewa_Madhya Pradesh
1542	P9014000504	27/05/2020	Dharmavaram_Andhra Pradesh
1543	P9015000503	2/12/2020	Asansol_West Bengal

Figure 3: Dropped Data and Output

Parameters for Associative Mining on Patient Travel Route

Initially the support and confidence used were both set at 10%. This produced only one rule, which we felt was not sufficient for the large data we were using.

	Left_side	Right_side	Support	Confidence	Lift
0		Sardarshahar_Rajasthan	0.147091	0.147091	1.0

Figure 4: Association Rule at MIn Support of 10% and Min Confidence at 10%

Association mining was conducted again but this time at a minimum support of 5% and a minimum confidence of 10%. Conducting association mining at these thresholds still only produced the one rule.

	Left_side	Right_side	Support	Confidence	Lift
0		Sardarshahar_Rajasthan	0.147091	0.147091	1.0

Figure 5: Association Rule at min support of 5% and confidence at 10%.

On are third attempt thresholds were lowered again with minimum support set to a threshold of 1% and minimum confidence set to 1% also. Using these thresholds, we were able to produce 59 associative rules. While 59 associative rules were produced it should be noted that due to the thresholds set at these parameters, rules that only have a support of 1% have only occurred a minimum of 1% of the times in the data (meaning they only occurred 15 times). With the confidence set at 1% this also means that travel route including “X” location, only have “Y” location present at a minimum of 1% of the time. While this may seem like we have produced low probability associations it should be noted that with the dataset used there are 22, 500 possible travel routes and identifying

travel routes only used by 1% COVID-19 positive patients still represents a significant percentage of route usage. While associations at a threshold this low may not be useful in consumer insight association mining they are valuable insights when trying to build an understanding of possible areas were COVID-19 might be spreading to and from.

```
rom apyori import apriori
 type cast the transactions from pandas into normal list format and run apriori
ravel_list = list(travel)
esults = list(apriori(travel_list, min_support=0.01, min_confidence = 0.01))
 print first 5 rules
rint(results[:5])

 I ended up using a min_support of 0.01 and min_confidence at 0.1 because these parameters produced the least redundent rules
 with all having a lift above > 1 while still representing a percentage of the data that is meaningful.
```

```
#reverse rules are still considered valid as they represent a cluster being created and then traveling, creating a cluster ar
#going home

def convert_apriori_results_to_pandas_df(results):
    rules = []
    for rule_set in results:
        for rule in rule_set.ordered_statistics:
            # items_base = left side of rules, items_add = right side
            # support, confidence and lift for respective rules
            rules.append([','.join(rule.items_base), ','.join(rule.items_add),
                rule_set.support, rule.confidence, rule.lift])
    # typecast it to pandas df
    return pd.DataFrame(rules, columns=['Left_side', 'Right_side', 'Support',
    'Confidence', 'Lift'])
result_df = convert_apriori_results_to_pandas_df(results)
result_df
```

46		Vasco da Gama_Goa	0.014270	0.014270	1.000000
47		Viramgam_Gujarat	0.015368	0.015368	1.000000
48		Jorapokhar_Jharkhand,Channapatna_Karnataka	0.012075	0.012075	1.000000
49	Channapatna_Karnataka	Jorapokhar_Jharkhand	0.012075	0.215686	5.458061
50	Jorapokhar_Jharkhand	Channapatna_Karnataka	0.012075	0.305556	5.458061
51		Sardarshahar_Rajasthan,Gokak_Karnataka	0.027442	0.027442	1.000000
52	Gokak_Karnataka	Sardarshahar_Rajasthan	0.027442	0.500000	3.399254
53	Sardarshahar_Rajasthan	Gokak_Karnataka	0.027442	0.186567	3.399254
54		Sardarshahar_Rajasthan,Kollam_Kerala	0.012075	0.012075	1.000000
55	Kollam_Kerala	Sardarshahar_Rajasthan	0.012075	0.203704	1.384881
56	Sardarshahar_Rajasthan	Kollam_Kerala	0.012075	0.082090	1.384881
57		Sardarshahar_Rajasthan,Lucknow_Uttar Pradesh	0.018661	0.018661	1.000000
58	Lucknow_Uttar Pradesh	Sardarshahar_Rajasthan	0.018661	0.215190	1.462970
59	Sardarshahar_Rajasthan	Lucknow_Uttar Pradesh	0.018661	0.126866	1.462970

Figure 6: Association Mining at Support and Confidence of 1%

Top 5 Travel Route

With the minimum support and confidence set to 1% the top 5 rules were identified by identifying the 5 rules with the highest lift.

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

Figure 7: Top 5 Rules

Table 2: Top 5 Routes Identified by Association Mining

		Support	Confidence	Lift
Jorapokhar Jharkhand	Channapatna Karnataka	0.012	0.306	5.458
Channapatna Karnataka	Jorapokhar Jharkhand	0.012	0.216	5.458
Gokak Karnataka	Sardarshahar Rajasthan	0.027	0.5	3.399
Sardarshahar Rajasthan	Gokak Karnataka	0.027	0.187	3.399
Sardarshahar Rajasthan	Lucknow Uttar Pradesh	0.019	0.127	1.463

The route the produced the highest lift was Jorapokhar city in the state of Jharkhand to Channapatna city in the state of Karnataka. The inverse of this rule was the same but the confidence was lower in this case. These two routes being the top among all other routes indicate that Jorapokhar and Channapatna not only share a common travel route but they are also likely COVID-19 hotspots. Similar to the top two rules, Gokak city in the state of Karnataka and Sardarshahar city in the state of Rajasthan and the inverse of this rule occupied but the third and fourth spot on this list. This represent a similar analysis as to the first and second rule but it should be noted that Sardarshahar city in the state of Rajasthan is the only place to appear three times in the top five travel routes indicating that while it may not be involved in a rule that produces the highest lift if could potentially be an area of more concern as it seems to be present in more travel routes used by COVID-19 patients then other locations listed in the top 5.

Top 5 Common routes Out of Ranebennur Town in Karnataka

To identify the top travel routes out of Ranebennur the support and confidence levels had to be reduced again to allow for to be more rules to be produced for Ranebennur town, as previous parameters produced none. Support and confidence levels were reduced to 0.1%. At these settings the Apriori algorithm produced 7,040.

```
#confidence Levels have to lowered to produce 5 reslts for Ranebennur_Karnataka

from apyori import apriori

travel_list = list(travel)
results = list(apriori(travel_list, min_support=0.001, min_confidence = 0.001))

print(results[:5])
```

```
def convert_apriori_results_to_pandas_df(results):
    rules = []
    for rule_set in results:
        for rule in rule_set.ordered_statistics:

            rules.append([','.join(rule.items_base), ','.join(rule.items_add),
                rule_set.support, rule.confidence, rule.lift])

    return pd.DataFrame(rules, columns=['Left_side', 'Right_side', 'Support',
    'Confidence', 'Lift'])
result_df = convert_apriori_results_to_pandas_df(results)
result_df
```

	Left_side	Right_side	Support	Confidence	Lift
0		Alipurduar_West Bengal	0.026345	0.026345	1.000000
1		Amalner_Maharashtra	0.004391	0.004391	1.000000
2		Ambernath_Maharashtra	0.006586	0.006586	1.000000
3		Anand_Gujarat	0.016465	0.016465	1.000000
4		Anantnag_Jammu and Kashmir	0.009879	0.009879	1.000000
...	...	...	...	...	...
7036	Jorapokhar_Jharkhand,Ramanagara_Karnataka,Mahi...	Ranebennur_Karnataka	0.001098	1.000000	11.531646
7037	Channapatna_Karnataka,Jorapokhar_Jharkhand,Mah...	Ramanagara_Karnataka	0.001098	1.000000	91.100000
7038	Channapatna_Karnataka,Jorapokhar_Jharkhand,Ram...	Mahidpur_Madhya Pradesh	0.001098	1.000000	101.222222
7039	Channapatna_Karnataka,Ramanagara_Karnataka,Mah...	Jorapokhar_Jharkhand	0.001098	1.000000	25.305556
7040	Jorapokhar_Jharkhand,Ramanagara_Karnataka,Mahi...	Channapatna_Karnataka	0.001098	1.000000	17.862745

Figure 8: Association Mining at Support and Confidence of 0.1%

Once we had the rules, a data frame consisting of having just Ranebennur in the left side of the rule was created. With this we produced the top 5 rules by ordering them in descending order by lift. The output represents the top 5 travel routes out of Ranebennur town in Karnataka state.

```
result_df_Karbataka = result_df[result_df['Left_side'] == 'Ranebennur_Karnataka']
```

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

	Left_side	Right_side	Support	Confidence	Lift
3885	Ranebennur_Karnataka	Jorapokhar_Jharkhand,Ratnagiri_Maharashtra	0.002195	0.025316	11.531646
5534	Ranebennur_Karnataka	Ratnagiri_Maharashtra,Medinipur_West Bengal,Ch...	0.001098	0.012658	11.531646
5490	Ranebennur_Karnataka	Mahidpur_Madhya Pradesh,Ramanagara_Karnataka,C...	0.001098	0.012658	11.531646
5444	Ranebennur_Karnataka	Jorapokhar_Jharkhand,Ratnagiri_Maharashtra,Cha...	0.001098	0.012658	11.531646
5385	Ranebennur_Karnataka	Jorapokhar_Jharkhand,Mahidpur_Madhya Pradesh,C...	0.001098	0.012658	11.531646

Figure 9: Top 5 Rules for Ranebennur

The results showed that the most common routes involving Ranebennur also involve Jorapokhar and Ratnagiri, as there was a 0.22% chance that someone would have been to all three places and a further 0.25% chance that a COVID-19 patient who has been to Jorapokhar and Ratnagiri has travelled from

Ranebennur. This rule is further strengthened by its high lift of 11.53, meaning a COVID-Patient who is from Ranebennur is 11 times more likely to have been to Jorapokhar and Ratnagiri then a COVID-19 patient form he samples chosen at random.

Sequential Mining on COVID-19 Patient Route Data

As the date of travel routes by COVID-19 positive travellers were recorded and put in order by date sequential mining could be conducted. While sequence mining was performed on the data, it should be noted that the assumption was made that it was recoded which destination came first for travellers who travelled to two places in the same day, as specific hourly timestamps were not given just dates. Using the same support and confidence percentages as association mining produced less then 10 rules so support was dropped to 0.1% while confidence was kept at 1%. This produced 1,755 rules.

```
#sequential association mining can be done because the data is ordered by date in accending order, so each travel of
# patient is sequential. The data does not have exact time but it is assumebed that patients with travel on the same day
#is still ordered sequentially

transactions = df.groupby(['patient_id'])['location'].apply(list)
sequences = transactions.values.tolist()

# show the first 5 sequences
print(sequences[:5])
```

```
[['Chittorgarh_Rajasthan', 'Ratnagiri_Maharashtra'], ['Pindwara_Rajasthan', 'Raipur_Chhattisgarh', 'Gokak_Karnataka'], ['Lu
cknow_Uttar Pradesh'], ['Lucknow_Uttar Pradesh'], ['Delhi_Delhi']]
```

```
from collections import defaultdict
import subprocess
import re

''' Uses SPMF to find association rules in supplied transactions '''
def get_association_rules(sequences, min_sup, min_conf):
    # step 1: create required input for SPMF

    # prepare a dict to uniquely assign each item in the transactions to an int ID
    item_dict = defaultdict(int)
    output_dict = defaultdict(str)
    item_id = 1

    # write your sequences in SPMF format
    with open('seq_rule_input.txt', 'w+') as f:
        for sequence in sequences:
            z = []
            for itemset in sequence:
                # if there are multiple items in one itemset
                if isinstance(itemset, list):
                    for item in itemset:
                        if item not in item_dict:
                            item_dict[item] = item_id
                            item_id += 1

                        z.append(item_dict[item])
                else:
                    if itemset not in item_dict:
                        item_dict[itemset] = item_id
                        output_dict[str(item_id)] = itemset
                        item_id += 1
                    z.append(item_dict[itemset])

                # end of itemset
                z.append(-1)

            # end of a sequence
            z.append(-2)
            f.write(' '.join([str(x) for x in z]))
            f.write('\n')

    # run SPMF with supplied parameters
    supp_param = '{}%'.format(int(min_sup * 100))
    conf_param = '{}%'.format(int(min_conf * 100))
    subprocess.call(['java', '-jar', 'spmf.jar', 'run', 'RuleGrowth',
                     'seq_rule_input.txt', 'seq_rule_output.txt',
                     supp_param, conf_param], shell=True)

    # read back the output rules
    outputs = open('seq_rule_output.txt', 'r').read().strip().split('\n')
    output_rules = []
    for rule in outputs:
        left, right, sup, conf = re.search(pattern=r'([0-9\,]+) ==> ([0-9\,]+) #SUP: ([0-9]+) #CONF: ([0-9\.]+)', string=rule
        sup = int(sup) / len(sequences)
        conf = float(conf)
        output_rules.append([[output_dict[x] for x in left.split(',')], [output_dict[x] for x in right.split(',')], sup, conf

    # return pandas DataFrame
    return pd.DataFrame(output_rules, columns = ['Left_rule', 'Right_rule', 'Support', 'Confidence'])
```

```
get_association_rules(sequences, 0.001, 0.01)
```

	Left_rule	Right_rule	Support	Confidence
0	[Chittorgarh_Rajasthan]	[Ratnagiri_Maharashtra]	0.002195	0.200000
1	[Chittorgarh_Rajasthan]	[Ratnagiri_Maharashtra, Gola Gokrannath Kheri_...	0.001098	0.100000
2	[Chittorgarh_Rajasthan]	[Delhi_Delhi]	0.001098	0.100000
3	[Chittorgarh_Rajasthan, Channapatna_Karnataka]	[Delhi_Delhi]	0.001098	0.500000
4	[Chittorgarh_Rajasthan]	[Sagar_Karnataka]	0.001098	0.100000
...	...	...	...	...
1751	[Imphal_Manipur]	[Cooch Behar_West Bengal]	0.001098	0.250000
1752	[Cooch Behar_West Bengal]	[Viramgam_Gujarat]	0.001098	0.125000
1753	[Viramgam_Gujarat]	[Cooch Behar_West Bengal]	0.002195	0.142857
1754	[Panaji_Goa]	[Anand_Gujarat]	0.001098	0.125000
1755	[Bodhgaya_Bihar]	[Chabua_Assam]	0.001098	0.500000

1756 rows × 4 columns

Figure 10: Sequence Analysis at Support of 0.1% and Confidence at 1%

Relevance of Analysis to Decision Makers

This association mining process has demonstrated areas in which identified COVID-19 infected people travel with in India. The association rule mining demonstrates areas in which COVID-19 patients travel between. This not only identifies travel routes to monitor but also areas where COVID-19 is very prominent. This research could guide decision makers into high risk areas and travel routes that may need more testing and treatment facilities and possibly monitoring of travel or restriction. The sequential analysis proves a more precious path of where the initial travel routes are starting and were they are headed to. This data could again be very useful in trying to minimise the spread of COVID-19 between cities and states.

Part B)

Clustering Diabetes Data

Pre-processing of Diabetes Data

When evaluating the diabetes patient data, we first looked for any null values and found some present in race, age and chlorpropamide. These null values were replaced with the mode value for all three of the categorical features.

```
for col in df:
    print(df[col].unique())
```

```
['Caucasian' 'AfricanAmerican' 'Hispanic' nan 'Other' 'Asian']
['Female' 'Male' 'Unknown/Invalid']
['[60-70)' '[80-90)' '[70-80)' '[40-50)' '[50-60)' '[90-100)' '[30-40)'
 '[20-30)' '[10-20)' '[0-10)' nan]
[2 3 1 5 6 8 7 4]
[ 1  2  6  3 22 11  7 23 13  4 15  9 28  5 19  8 25 14 24 18 27]
[ 1  7  5 17  4  2  3  6  9 10 22  8 14 11 25 13]
[ 4  8  2  7  3  6  5  1 13 12 11  9 10 14]
['InternalMedicine' 'Invalid' 'Cardiology' 'Emergency/Trauma' 'Neurology'
 'Surgery-General' 'Family/GeneralPractice'
 'Surgery-Cardiovascular/Thoracic' 'Surgery-Vascular' 'Oncology'
 'Osteopath' 'Surgery-Neuro' 'Radiologist' 'Nephrology' 'Pulmonology'
 'Orthopedics-Reconstructive' 'Gastroenterology' 'Surgery-Plastic'
 'Surgery-Pediatric' 'Psychiatry' 'Orthopedics' 'Urology'
 'Surgery-Cardiovascular' 'Gynecology' 'PhysicalMedicineandRehabilitation'
 'ObstetricsandGynecology' 'Hematology/Oncology' 'Podiatry'
 'Otolaryngology' 'Surgeon' 'Pediatrics' 'Psychology' 'Hematology'
 'OutreachServices' 'Endocrinology' 'Pathology' 'Obstetrics'
 'Surgery-Thoracic' 'Cardiology-Pediatric' 'Ophthalmology' 'Rheumatology'
 'Surgery-Maxillofacial' 'Hospitalist' 'InfectiousDiseases' 'Perinatology'
 'PhysicianNotFound' 'SurgicalSpecialty' 'Neurophysiology'
 'Anesthesiology' 'Radiology' 'Endocrinology-Metabolism' 'DCPTEAM'
 'Resident']
[ 43  48  39  54  70   1  28  42  27  26  51  41  50  37  46  71  55  35
  56   4  52  31  16  45  57  64  59  33  60  53  62  30   9  38  49  29
  58   5  10   3  65  68  32  44  34  23  36  40  75  66  61  67  11  47
  22  80  12  87  76   2  69  72  63  79  77  25  14  74  81   8  21  17
  24  19  83   6  78  73  93  18  82  20 113  15  85   7  86  88  84  13
  99  95  89  91 100  94 111  92  98  96 109 114  90  97 104 105 102 101
 108 103 132 106 121 118]
[0 6 1 4 2 3 5]
[13 26  9 11 20  7 18 12  5 15 17 10 62 16 37  4  2 14  8 21 22 65 25 31
 24 19 27  6 34 33  1 28 23 53 32 64 35 55 30 29  3 47 44 69 39 38 61 42
 51 40 60 41 36 57 63 52 43 50 45 46 66 49 56 59 54 58 72 48 68 74 75 67]
[ 1  0  2  5  4  3 15  6  7 11 12 10 14  9  8 13 18 19 27 22 24 16 42 21
 36 20 26 33 17 25 23]
[ 0  2  1  3  7  5  4  8  9  6 25 10 13 11 16 37 20]
[ 0  1  2  3  5  4  9  6 10  7 12  8 11 14 19]
[ 9  3  6  4  8  7  5  2  1 16 12 13 15 10 11 14]
['None' 'Norm' '>200' '>300']
['None' 'Norm' '>8' '>7']
['No' 'Steady' 'Up' 'Down']
['No' 'Steady' 'Up' 'Down']
['No' 'Steady' 'Up' 'Down']
['No' nan 'Steady']
['No' 'Steady' 'Up' 'Down']
['No']
['No' 'Steady' 'Up' 'Down']
['No' 'Steady' 'Up' 'Down']
['No']
['Steady' 'Down' 'No' 'Up']
[False  True]
[ True False]
```

Figure 11: Identified Nan Values in Diabetic Data

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

```
array(['No', nan, 'Steady'], dtype=object)
```

```
df['chlorpropamide'] = df['chlorpropamide'].fillna('No')
df['chlorpropamide'].unique()
```

```
array(['No', 'Steady'], dtype=object)
```

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

```
array(['No', 'Steady'], dtype=object)
```

Figure 12: Procedure for Removing Nan Values

Categorical data was also converted to numeric data as apart of the prepossessing to conduct clustering. Mapping was used on all Categorical data that was going to be used in the clustering model.

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

```
array(['[60-70)', '[80-90)', '[70-80)', '[40-50)', '[50-60)', '[90-100)',
       '[30-40)', '[20-30)', '[10-20)', '[0-10)', nan], dtype=object)
```

```
df['age'] = df['age'].fillna('[70-80)')
df['age'].unique()
```

```
array(['[60-70)', '[80-90)', '[70-80)', '[40-50)', '[50-60)', '[90-100)',
       '[30-40)', '[20-30)', '[10-20)', '[0-10)'], dtype=object)
```

```
age_map = {'[0-10)':0, '[10-20)':1, '[20-30)':2, '[30-40)':3, '[40-50)':4, '[50-60)':5, '[60-70)':6
df['age'] = df['age'].map(age_map)
```

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

```
array([6, 8, 7, 4, 5, 9, 3, 2, 1, 0], dtype=int64)
```

Figure 13: Categorical Mapping Process

A loop was conducted to show the value counts of all variables in the data to examine if there were any abnormal distributions of values among the variables. Analysis of this did not show any variables with unusual distributions of data. While there some variables in data that would be considered outliers, for example here was one patient that had 37 emergency visits while close to 90% of patients had none, it was decided that these potential outliers should be left in as medical anomalies in a data of 35, 335 still represent valuable data when characterising a medical condition that is as prevalent as diabetes.

```
for col in df:
    print(df[col].value_counts())
```

```
Caucasian          27776
AfricanAmerican     5764
Hispanic             806
Other                696
Asian                293
Name: race, dtype: int64
Female             19196
Male               16984
Unknown/Invalid        3
Name: gender, dtype: int64
[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
Name: age, dtype: int64
1    20126
3     7704
2     6045
5     1103
6     1024
8      160
7       19
4        2
Name: admission_type_id, dtype: int64
```

Figure 14: Cropped Image for Value Count Output

Clustering Model Used

While K-means and K-modes are very common methods of clustering they are limited to only being able to cluster for numeric and categorical data respectively. As the data we were using for the clustering was a mix of numeric and categorical K-prototypes was used as it can handle both numeric and categorical. While we chose to leave patients with outlier data in as they represented what we believe to still be a valuable diabetic population characteristic, this could affect the clustering as K-prototypes is a mix of K-means and K-modes and K-means is very susceptible to outliers.

Attributes Chosen for Clustering

Attributes were chosen that were deemed important to representing the characteristics of diabetic patients. Attributes that we deemed to either be unrepresented of diabetic patients or that were made redundant by other variables were dropped. Variables that could also be skewed by patients not having pre-existing medical records or other factors that are not related to diabetes were also dropped.

Dropped Variables

- **Gender** – Gender was dropped as previous studies found that gender was not a relevant factor to diabetes and thus was deemed unnecessary in creating a profile of characteristics for diabetic patients.
- **Admission type** ID – Admission type was excluded from the data as majority of the admission types were emergency and urgent and it was decided that the variable Number of emergency better represented the presence of emergency visits characterised in diabetic patients.

- **Discharge Disposition Id and Admission Source ID** – these identifiers were deemed unnecessary as they mostly referred to hospital administration information.
- **Medical Speciality** – This again was a variable deemed more relevant to hospital administration then building a profile for diabetic characteristics.
- **Number of Procedures** - this variable included all procedures but lab procedures. We decided this could be redundant as it is probably measuring procedures such as checking weight, height and other characteristics that would not be measured if the patient had an up-to-date medical profile. For this reason, we decided it did not represent anything important.
- **Number of Inpatient** – Number of inpatient visits is not necessarily an important identifier of diabetic patients as inpatient visits could be more frequent dependent on the patient or doctor's preference. Inpatient visits may not also be of the upmost medical important, as they could be in reference checking in on weekly weight goals and dietary recording. The frequency of these visits could also be affected by factors such as proximity to hospital and socioeconomic status.
- **Number of Diagnose** – there was not enough information provided to number of diagnosis represents. While it would be an interesting characteristic to include a metric comorbidity, it is not stated if false-positive diagnosis is counted in this variable. For this reason, it was dropped.
- **Max Glu Serum and A1C Result** – these to test results were excluded as they were not preformed on a large majority of the patient's we had data for and for this reason we felt it was important to exclude them and might misrepresent their importance.
- **All Medication Except Insulin** – Insulin was the only medication that had a large enough proportion of a sample who were proscribed it to be deemed relevant to include as an attribute. As in most cases for other medications only had around 10% of the sample prescribed medication it was not included in the data.
- **Change -** Change was a variable that represented if there was a change or not in medication. This could have been a valuable variable for building a profile of characteristics of diabetic patients if the change listed the medication previously taken and the new medication, as this was not the case we did not include it.

Variables Chosen to be Clustering Attributes

- **Race**
- **Age**
- **Time in hospital**
- **Number of lab procedures**
- **Number of medications –** Distinct generic medication given at encounter**.**
- **Number of outpatients –** number of outpatient visits preceding the year after first encounter.
- **Number of emergencies –** number of emergency visits in the first year proceeding first encounter.
- **Insulin**
- **Diabetes Medicine –** a measure of if diabetes medicine was prescribed, deemed important as it shows patients with no prescriptions.

Optimal Number of Clusters Used

To identify the optimal K-value to use for clustering first the Elbow method was used. The elbow method did not produce a clear precise optimal number of clusters as can be seen in *figure 15*.

```
from kmodes.kmodes import KModes
from kmodes.kprototypes import KPrototypes

# list to save the clusters and cost
clusters = []
cost_vals = []
rs = 10


# this whole process should take a while
for k in range(2, 10, 2):
    # train clustering with the specified K
    model = KPrototypes(n_clusters=k, random_state=rs, n_jobs=10)
    model.fit_predict(X, categorical=[1])

    # append model to cluster list
    clusters.append(model)
    cost_vals.append(model.cost_)
```

```
# plot the cost vs K values
plt.plot(range(2,10,2), cost_vals, marker='*')
plt.show()
```

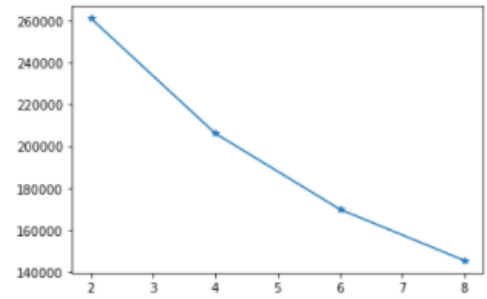


Figure 15: Elbow Method used for Optimal-K

As the elbow method did not produce a clear number of clusters average Silhouette scores were used to demonstrate the optimal number of clusters.

Number of Clusters	The Average Silhouette Score
4	0.042
6	-0.005
8	-0.045

Figure 16: Avg Silhouette Score

```
from sklearn.metrics import silhouette_score

# Calculate the Silhouette Score for the numeric and categorical variables seperately
silScoreNums = silhouette_score(X_num, model.fit_predict(X,categorical=[1]), metric='euclidean')
print("Silscore for numeric variables: " + str(silScoreNums))
silScoreCats = silhouette_score(X_cat, model.fit_predict(X,categorical=[1]), metric='hamming') # nc
print("Silscore for categorical variables: " + str(silScoreCats))

# Average the silhouette scores
silScore = (silScoreNums + silScoreCats) / 2
print("The avg silhouette score for k=4: " + str(silScore))
```

```
Silscore for numeric variables: 0.10523544598039714
Silscore for categorical variables: -0.02123804076504562
The avg silhouette score for k=4: 0.041998702607675756
```

```
from sklearn.metrics import silhouette_score
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))

model = clusters[3]
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=8: " + str(silScore))
```

```
The avg Silhouette score for k=4: 0.041998702607675756
The avg Silhouette score for k=6: -0.004756909125729121
The avg Silhouette score for k=8: -0.044518456360931706
```

Figure 17: Coding for Silhouette Score

Ideally you want your Silhouette to be as close to 1 as possible as that represents that objects in your cluster match your cluster and are dissimilar to the neighbouring clusters. The results of the Silhouette scores show that 4 clusters are the only number of clusters that produces a silhouette score close to 1 as the other two produce negative scores that represent objects that have potentially been matched to the wrong clusters. While having 4 clusters does not produce the best Silhouette score it is much better then then using 6 and 8.

Normalisation of Variables

The same clustering process was conducted without using scaling as our normalisation. We attempted to preform clustering without scaling to see if we could get more defined clusters. While the clusters were more defined visually, the produced very low silhouette scores that indicated the objects in the cluster likely belonged to the wrong cluster.

Number of Clusters	The Average Silhouette Score
4	-0.124
6	-0.014
8	-0.069

Figure 18: Silhouette Scores Without Scaling

```
from sklearn.metrics import silhouette_score

# Calculate the Silhouette Score for the numeric and categorical variables seperately
silScoreNums = silhouette_score(X_num, model.fit_predict(X,categorical=[1]), metric='euclidean')
print("Silscore for numeric variables: " + str(silScoreNums))
silScoreCats = silhouette_score(X_cat, model.fit_predict(X,categorical=[1]), metric='hamming') # nc
print("Silscore for categorical variables: " + str(silScoreCats))

# Average the silhouette scores
silScore = (silScoreNums + silScoreCats) / 2
print("The avg silhouette score for k=4: " + str(silScore))
```

```
Silscore for numeric variables: -0.12357942879509688
Silscore for categorical variables: -0.013892007227537287
The avg silhouette score for k=4: -0.06873571801131709
```

```
from sklearn.metrics import silhouette_score
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))

model = clusters[3]
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=8: " + str(silScore))
```

```
The avg Silhouette score for k=4: -0.06873571801131709
The avg Silhouette score for k=6: -0.08839123311606865
The avg Silhouette score for k=8: -0.10313701714351983
```

Figure 19: Coding Silhouette Scores Without Scaling

Optimal Clustering Model

Figure 19 shows the output of the optimal clustering model that uses scaling and the optimal number of clusters, 4. The clusters are not clearly defined when we look at certain attributes and this is likely a result of our low Silhouette score and including outliers in the model. Distribution of membership between clusters is also not evenly distributed as cluster 0 has more then double that of the next biggest cluster, this is not necessarily a negative to the model.

Cluster	Membership
0	18609
1	8744
2	7256
3	1571

Figure 20: Model 1 Cluster Membership

Interpretation from Cluster Visualisation

From the visualisation of the clustering it can be seen that cluster 3 typically has more lab procedures and more medication prescribed on first encounter then the other clusters. Cluster 0 has large representation among all age groups and is mainly represented of Caucasian and African-American patients. Cluster 2 represents a large amount of the sampled population that have not been prescribed medicine and have had under 10 emergency visits.

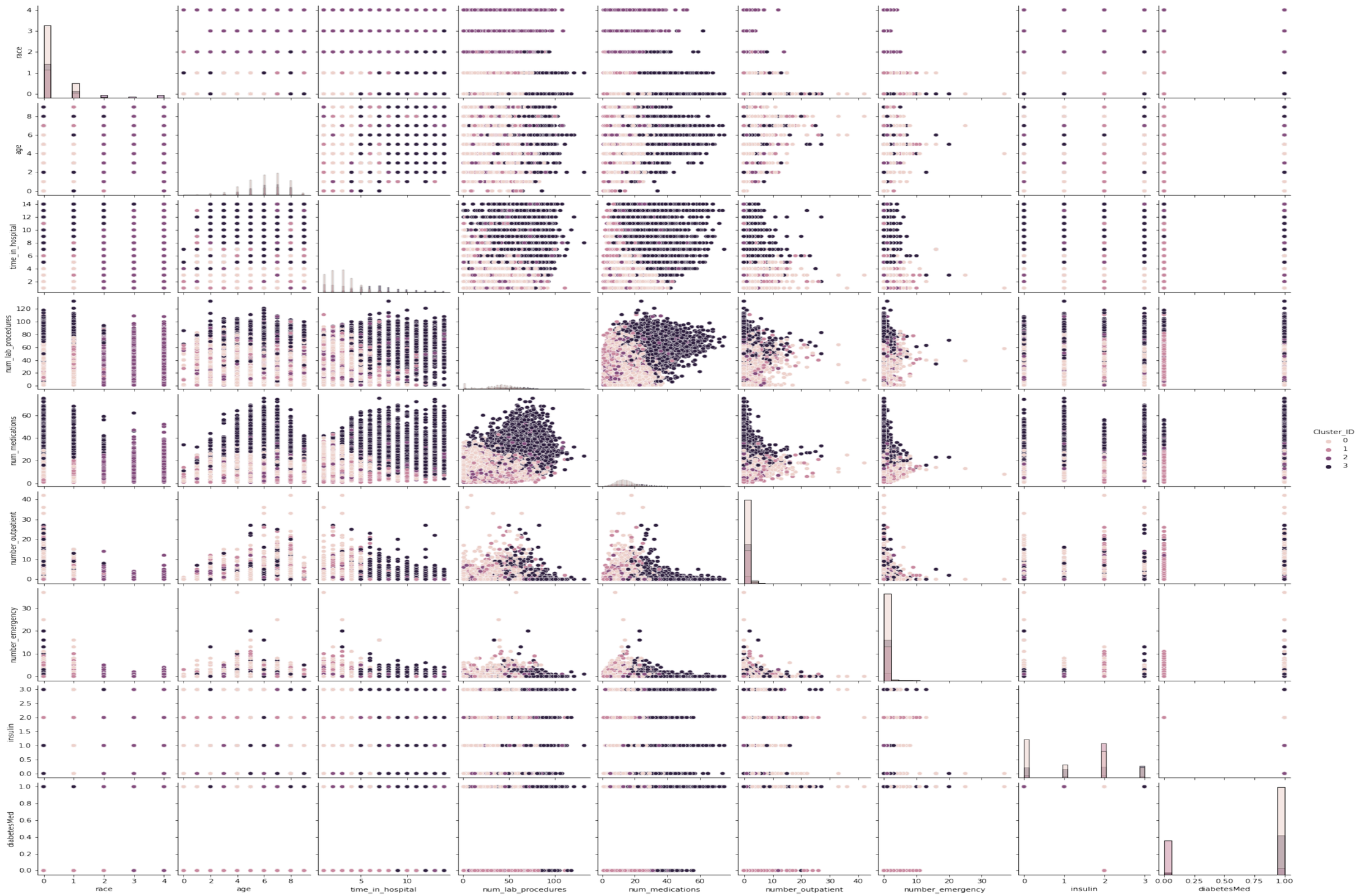


Figure 21: Optimal Clustering Model Output

Interpretation from Cluster Distribution Plot

As there was a large overlap in many of the clusters, distribution plots for each cluster were used to identify and characterise the nature of each cluster and to give it a descriptive label.

```
import pandas as pd
import numpy as np
# prepare the column and bin size. Increase bin size to be more specific, but 20 is more than enoug
cols = ['race', 'age','time_in_hospital','num_lab_procedures', 'num_medications', 'number_outpatier
n_bins = 20


clusters_to_inspect = [0,1,2,3]

for cluster in clusters_to_inspect:
    print("Distribution for cluster {}".format(cluster))
    fig, ax = plt.subplots(nrows=9)
    ax[0].set_title("Cluster {}".format(cluster))

    for j, col in enumerate(cols):
        bins = np.linspace(min(df2[col]), max(df2[col]), 20)
        sns.distplot(df2[df2['Cluster_ID'] == cluster][col], bins=bins, ax=ax[j], norm_hist=True, k
        sns.distplot(df2[col], bins=bins, ax=ax[j], hist=False, color="k")

    plt.tight_layout()
    fig.subplots_adjust(top= 10)
    plt.show()
```

Figure 22: Coding for Clustering Distribution Plot

Cluster 0: Only Diabetes Medicine Prescribed and Increased Dosage of Insulin

Cluster 0 represents a cluster that only had patients who were prescribed diabetes medication and who had their insulin dosage increased higher to that of the distribution of all sampled participants.

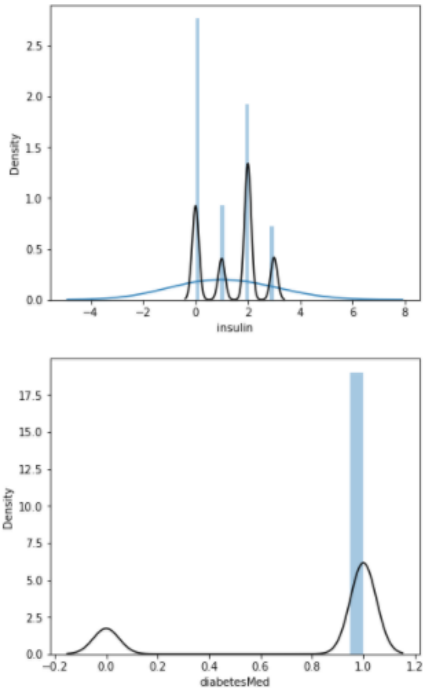


Figure 23: Distribution plot for Cluster 0

Cluster 1: Medicated but Not Insulin

Cluster 1 represents a cluster of patients that were prescribed diabetes medication but were not prescribed insulin.

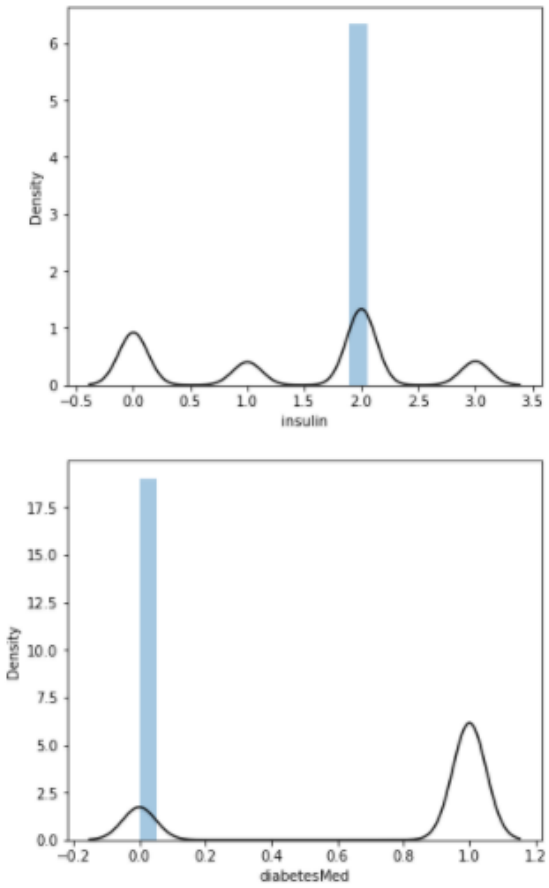
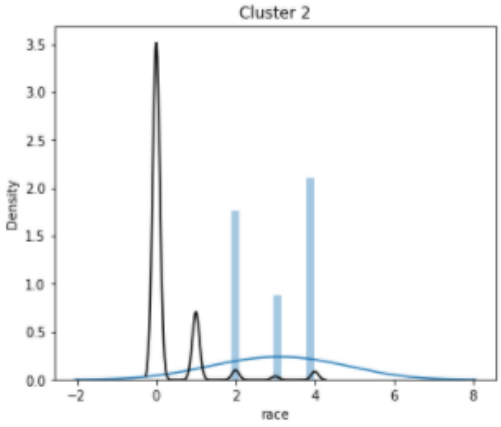


Figure 24: Distribution Plot for Cluster 1

Cluster 2: Hispanic, Asian and Other races

Cluster 2 represents a cluster that is only made up of Hispanic, Asian and other races and also is represents a cluster with slightly more outpatient visits.



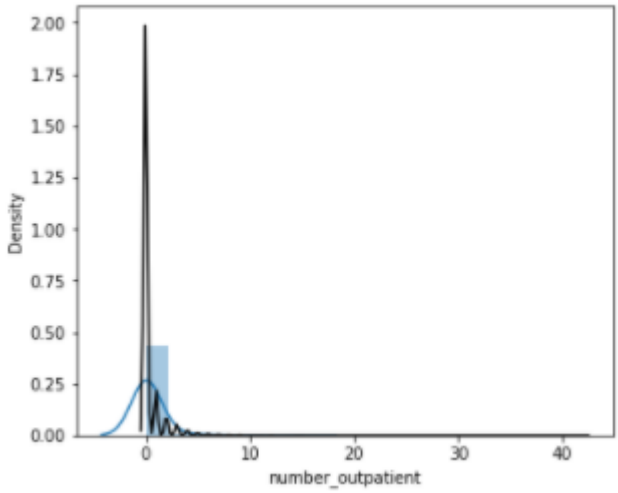


Figure 25: distribution Plot for Cluster 2

Cluster 3: More Time in Hospital, More laboratory Procedures, More Medication

Cluster 3 has a larger distribution of patients with more time spent in hospital, more laboratory procedures conducted and a higher number of medications prescribed on first encounter.

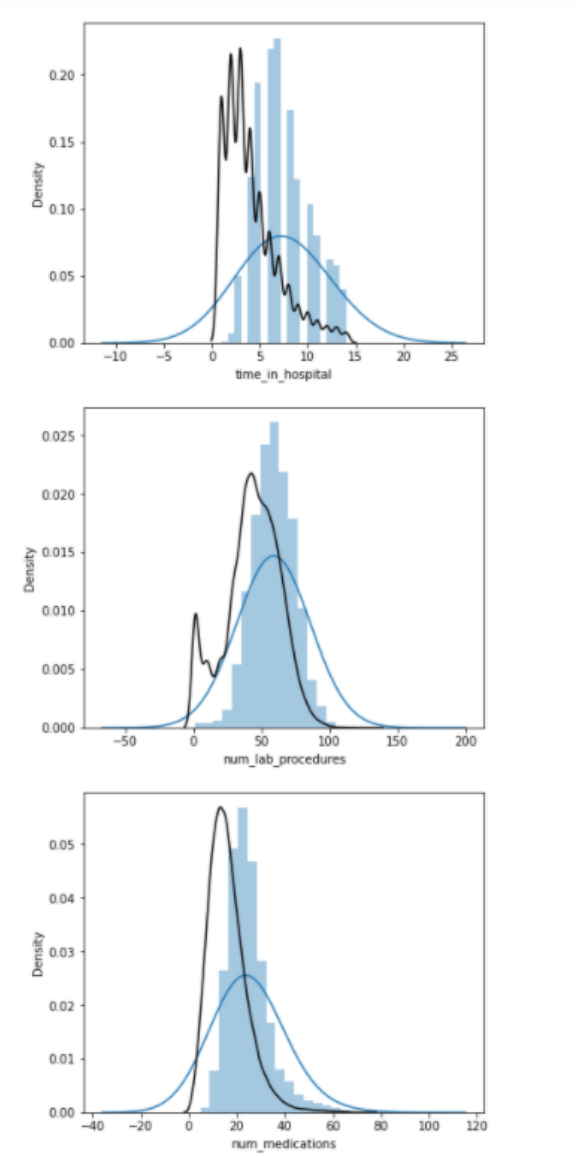


Figure 26: Cluster 3 Distribution Plots

Optimal Clustering Model on Caucasian and Asian Population

The same clustering model was used again this time only including the two races Caucasian and Asian. This was conducted by removing all other races from the data frame

```
df3 = df[['race', 'age','time_in_hospital','num_lab_procedures', 'num_medications', 'number_outpatient','number_emergency',

indexAfrice = df3[df3['race'] == 1 ].index
df3.drop(indexAfrice, inplace=True)

df3['race'].unique()

2]: array([0., 2., 4., 3.])

indexHispanic = df3[ df3['race'] == 2 ].index
df3.drop(indexHispanic , inplace=True)

df3['race'].unique()

4]: array([0., 4., 3.])

indexOther = df3[ df3['race'] == 4 ].index
df3.drop(indexOther , inplace=True)

df3['race'].unique()

6]: array([0., 3.])
```

Figure 27: Coding for Selecting Only Caucasian and Asian

We decided to keep the same attributes from the last model in this model to allow for a fair comparison between the two models.

	race	age	time_in_hospital	num_lab_procedures	num_medications	number_outpatient	number_emergency	insulin	diabetesMed
0	Caucasian	6	4	43	13	1	0	Steady	True
1	Caucasian	8	8	48	26	0	0	Down	True
2	Caucasian	8	2	39	9	0	0	No	False
3	Caucasian	6	2	54	11	0	0	No	False
4	Caucasian	6	7	70	20	1	0	Down	True
...	...	...	...	...	...	...	...	...	...
36178	Caucasian	7	9	50	33	0	0	Steady	True
36179	Other	7	14	73	26	0	1	Up	True
36180	Other	6	2	46	17	1	1	Steady	True
36181	Caucasian	8	5	76	22	0	1	Up	True
36182	Caucasian	7	6	13	3	0	0	No	False

Figure 28: Attributes used in Model 2

When comparing the two models the first notable difference is model 2 (shown below in *figure 29*) is that it has a less evenly distributed cluster membership then model 1.

Cluster	Membership
0	16,488
1	6,341
2	5,792
3	293

While the cluster shapes look relatively similar to model 1 the clusters that cover the area have changed. While cluster 3 was the cluster with most lab procedures and the greatest number of medicines in model 1, it is now cluster 2 that demonstrates that description in model 2. This is likely due to model 2 only feature Asian patients in cluster 3 which was not the case for cluster 3 in model 1

as it only included Caucasian and African American patients. This demonstrates a difference between the two races.

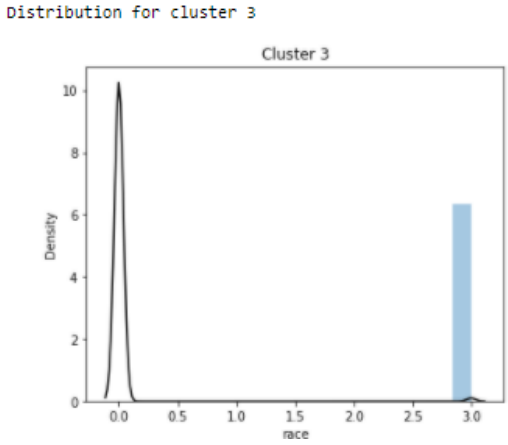


Figure 29: Race for Cluster 3 in Model 2

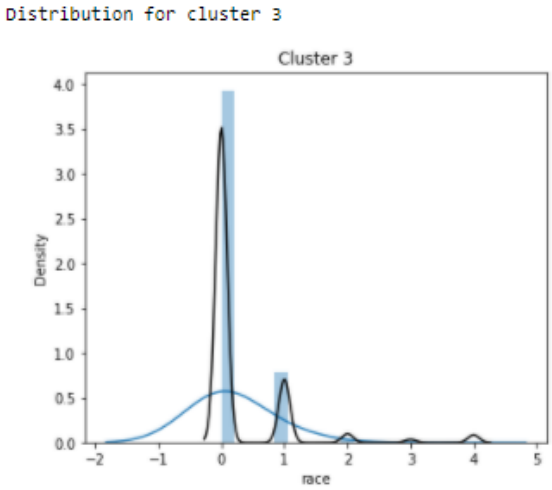


Figure 30: Race Distribution for Cluster 3 in module 1

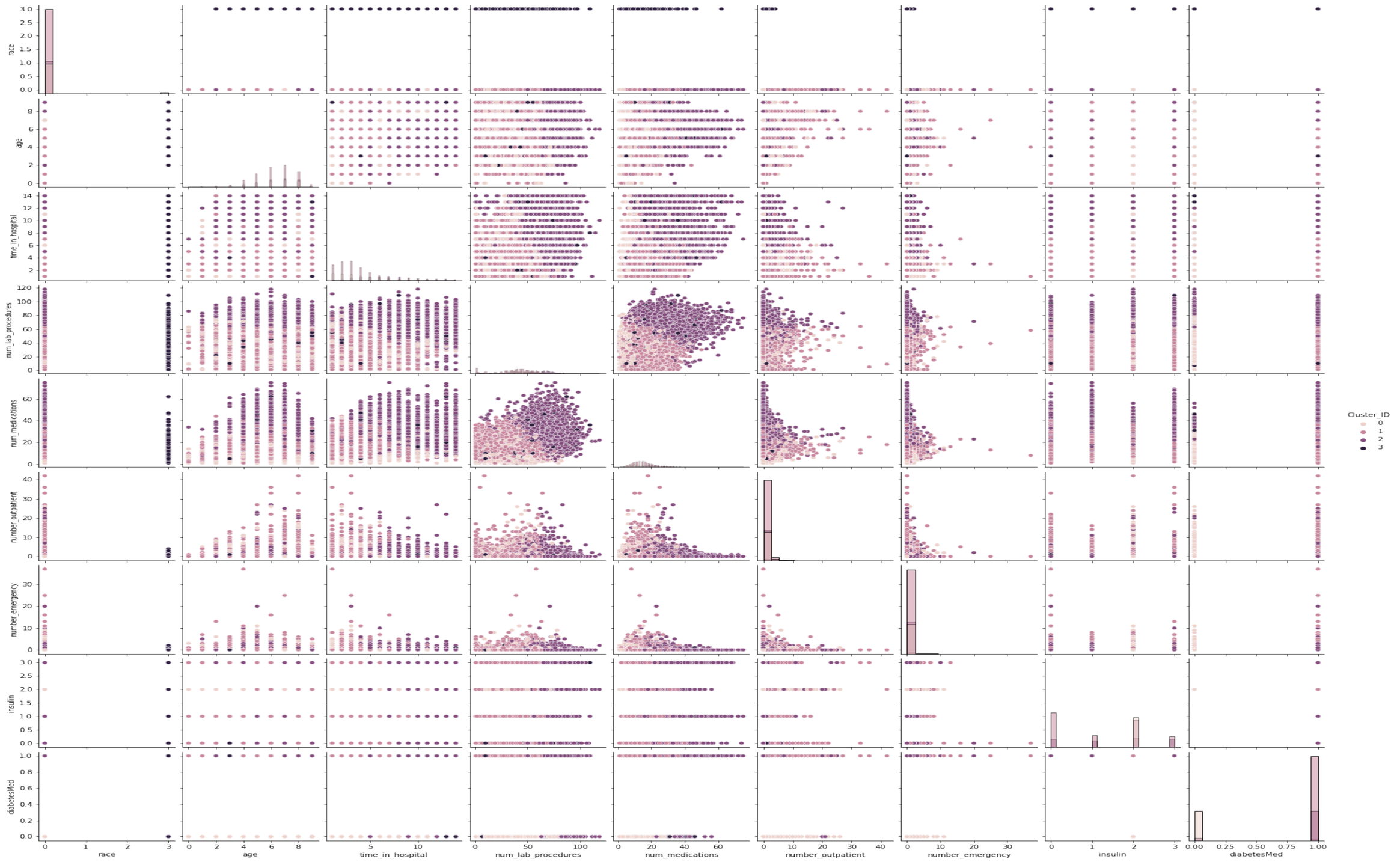


Figure 31: Clustering Model for Caucasian and Asia

Implications for Decision Makers from this Study

Using the clustering technique on the attributes provided by the diabetic patient data allowed us to create 4 unique clusters that allows us to visualise the characteristics of diabetic patients. By creating 4 unique clusters for eHealth they can identify the characteristics of the people in these 4 clusters and identify how some attributes may share a relationship and also build diabetic characteristic profiles around these 4 clusters. Building profiles around these clusters could allow for a more tailored approach to medical treatment. Furthermore modifying the model to only include specific races can demonstrate possible differences in characteristic profiles between races.

Part C

Decision Tree

Q1. Data was pre-processed; only a few minor changes were necessary. Age and Race both had missing values 'nan' ; these were replaced with the average value (Caucasian for Race & 70-80 for age). Both small enough to not impact the data. Some values were dropped completely;

Max_glu_serum mostly not recorded listed as 'none' in the data. Therefore it was dropped.

Acetohexamide & tolbutamide values only showed 'no' in the data, not useful for this type of analysis.

Medical_specialty showed mostly 'invalid', so this was dropped (assuming this means the data was not recorded as 'Family/General Practice' would account for non-specialised care).

Change & diabetes Med both account for diabetes medication, we don't want double count within the model so change was dropped.

70/30 Training to Test split was used, this is to ensure there is enough data to both train the model then test & verify it is working as expected.

Q2. A) Train accuracy shows 99.89%, Test Accuracy shows 57.08%. Large discrepancy between the two & close to 100% training accuracy shows this model is overfitted. Does not react the same way between the train set & test set

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

```
Train accuracy: 0.9998896125400154
Test accuracy: 0.5708306503541533
```

B) Number of nodes used; 20799. Number of rules used; 56. Large amount of nodes shows messy/suboptimal; there may be a lot of overlap to completely map the training data 100%. Would require pruning to be more effective, secondly is completely unreadable at this stage due to complexity.

```
x = model.fit(X_train, y_train)
n_nodes = x.tree_.node_count
print(n_nodes)
```

```
20799
```

C) Print out of following tree structure shows variable X[9] as first split, Corresponds to number_inpatient

```
The binary tree structure has 20799 nodes and has the following tree structure:

node=0 is a split node: go to node 1 if X[:, 9] <= 0.5 else to node 13796.
        node=1 is a split node: go to node 2 if X[:, 8] <= 0.5 else to node 12547.
                node=2 is a split node: go to node 3 if X[:, 7] <= 0.5 else to node 10580.
                        node=3 is a split node: go to node 4 if X[:, 3] <= 2.5 else to node 3529.
                                node=4 is a split node: go to node 5 if X[:, 10] <= 5.5 else to node 708.
                                        node=5 is a split node: go to node 6 if X[:, 1] <= 18.0 else to node 701.
                                                node=6 is a split node: go to node 7 if X[:, 5] <= 0.5 else to node 400.
```

D) Number of lab procedures, number of medications, time in hospital, the number inpatient & discharge disposition ID are the most important variables.

```
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 20 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 : 0.17465889488851036
num_medications : 0.12895241313160807
time_in_hospital : 0.07924117600387594
number_inpatient : 0.06770983992251624
discharge_disposition_id : 0.05674829215590216
```

E) This was a base model with only the random state parameter being adjusted (to Random state = 10).

```
model.fit(X_train, y_train)
```

```
DecisionTreeClassifier(random_state=10)
```

```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                       max_depth=None, max_features=None, max_leaf_nodes=Non
e,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=10, splitter='best')
```

Q3. A) Train accuracy shows 64.33%, Test Accuracy shows 64.01%. Performs similar on the test data & the training data, showing better performance than the base model.

```
CV1.fit(X_train, y_train)

print("Train accuracy: ", CV1.score(X_train, y_train))
print("Test accuracy: ", CV1.score(X_test, y_test))
```

```
Train accuracy:  0.6433381167899327
Test accuracy:  0.640888602704443
```

B) Number of nodes used; 125. Number of features used is 56. Much fewer nodes than the previous model. This again shows more optimization than the previous base model. With less complexity printing the decision tree we would be able to follow the decisions made (which is among the main benefits of decision trees compared to other ML models).

```
x2 = model2.fit(X_train, y_train)

n_nodes2 = x2.tree_.node_count
print(n_nodes2)
```

```
125
```

C) As per below first split in the model is again number_inpatient

```
The binary tree structure has 125 nodes and has the following tree structure:

node=0 is a split node: go to node 1 if X[:, 9] <= 0.5 else to node 64.
```

D) Number of inpatient visits, discharge disposition ID, number of emergency visits, number of outpatient visits & admission type ID.

```
importances = CV1.best_estimator_.feature_importances_
feature_names = X.columns

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

# limit to 20 features, you can leave this out to print out everything
indices = indices[:5]

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

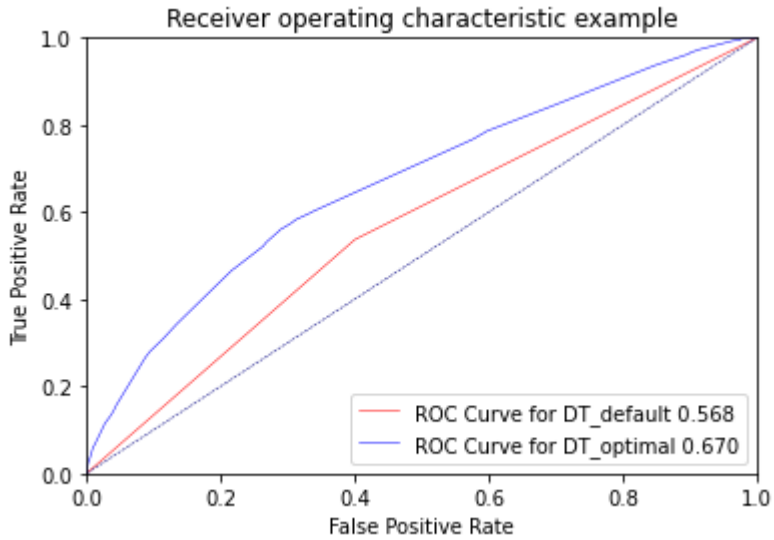
```
number_inpatient : 0.588571331787028
discharge_disposition_id : 0.1836370445558134
number_emergency : 0.06035826157463841
number_outpatient : 0.0489866964561794
admission_type_id : 0.0245273778209375
```

E) For our first model (Based on default features) there is strong evidence of overfitting. This is shown by the training data fitting the model at close to 100% & test accuracy 57% (The large disparity here confirms it does not perform as expected).

For our model optimized with GridsearchCV the training data fit the model 64% and test accuracy at 64% as well. This shows it performed within the test data set in the same way as the training data set meaning we can reliably depend on output. Further, this means the model could potentially be exported to similar data sets and we should see it react similarly. See below for full comparison.

	Nodes Total	Training Accuracy	Test Accuracy	First Variable Split	Top Feature Importance
Base Decision Tree	20799	99.89%	57.01%	number_inpatient	Num_lab_proceedures Num_medications Time_in_hospital Number_inpatient Discharge_disposition_ID
GridSearchCV Optimized	125	64.33%	64.01%	number_inpatient	Number_inpatient Discharge_disposition_ID Number_emergency Number_outpatient Admission_type_ID

4) The first model had over 20,000 nodes with second model using only 125 meaning there were over 100x more 'tests' the model had to complete. Second difference was the second model performed much better receiving the same on the test dataset and training dataset, the first model was overfitted to the training data & performed much differently on the test data. Without optimization cannot take any outcomes from this. Further, the optimized model showed a much stronger weight towards certain variables (58% for number of inpatient visits).



5) The most likely predictor of patients being readmitted is number of inpatient visits which shows how frequently in the previous year the patient has visited hospital. To explain simply, if a patient has been to hospital more times they are more likely to go to hospital again. This was the strongest variable by a large margin meaning it has the most predictive power. The other variable that could potentially have use is discharge Disposition ID, IE how they are discharged (It can be to short term/long term care eg) affects likelihood of readmission. Although a strong predictor within the context of healthcare number of inpatients isn't overly helpful, we cannot just tell patients to come to hospital less (then they are less likely to be readmitted). If it was a certain type of medication or a certain number of medications for example we could recommend the doctors review how necessary lots of medications are or if the patient is being given too high a dosage.

Linear Regression

Pre-processing

A) What pre-processing was required on the dataset before regression modelling? What distribution split between training and test datasets have you used? For this dataset, the pre-processing includes replacing 'nan' in the categorical variables with the mode values; dropping 'invalid' values in 'gender' and standardisation of variables since regression models are sensitive to input variables on different scales.

```
# impute missing values in race with 'Caucasian', which is the mode
df['race'] = df['race'].fillna('Caucasian')
# replace nan values in age with '[70-80)', which is the mode
df['age'] = df['age'].fillna('[70-80)')
# drop 'invalid' in gender
indexNames = df[ df['gender'] == 'Unknown/Invalid' ].index
df.drop(indexNames , inplace=True)
# impute nan values in chlorpropamide with 'No', which is the mode
df['chlorpropamide'] = df['chlorpropamide'].fillna('No')
# one-hot encoding
df = pd.get_dummies(df)
# target/input split
y = df['readmitted']
X = df.drop(['readmitted'], axis=1)
# setting random state
rs = 10
```

```
#Standardisation
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)))

X_test = scaler.transform(X_test)
```

We split the data into 70% training data and 30% test data.

```
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
```

Build default model ‘model’ and GridSearchCV tuned model ‘cv’

Build a regression model using the default regression method with all inputs.

```
# Build the first regression model using the default regression method
model = LogisticRegression(random_state=rs)

# fit it to training data
model.fit(X_train, y_train)
```

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='warn',
          n_jobs=None, penalty='l2', random_state=10, solver='warn',
          tol=0.0001, verbose=0, warm_start=False)
```

Build another regression model tuned with GridSearchCV.

```
#Building another model tuned with GridSearchCV
params = {'C': [pow(10, x) for x in range(-6, 4)]}

# use all cores to tune logistic regression with C parameter
cv = GridSearchCV(param_grid=params, estimator=LogisticRegression(random_state=rs),
                  return_train_score=True, cv=10, n_jobs=-1)
cv.fit(X_train, y_train)
```

```
GridSearchCV(cv=10, error_score='raise-deprecating',
       estimator=LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='warn',
          n_jobs=None, penalty='l2', random_state=10, solver='warn',
          tol=0.0001, verbose=0, warm_start=False),
       fit_params=None, iid='warn', n_jobs=-1,
       param_grid={'C': [1e-06, 1e-05, 0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000]},
       pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
       scoring=None, verbose=0)
```

Now, choose a better model to answer the followings:

A)

We choose the regression model tuned with Grid Search CV because this one has better training accuracy.

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

# classification report on test data
y_pred = model.predict(X_test)
print(classification_report(y_test, y_pred))
```

```
Train accuracy: 0.6356084232605635
Test accuracy: 0.6318500869341233
```

```
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: 0.6356360224105098
Test accuracy: 0.6318500869341233
```

B) Default linear regression with Grid Search CV.

C) Yes. Standardization was applied in the preprocessing phase because regression models are sensitive to input variables on different scales.

```
#Standardisation
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)))

X_test = scaler.transform(X_test)
```

d) All inputs were used in the regression.

e) The top inputs used in this model were as follows;

```
cv_best = cv.best_estimator_
cv_best.coef_ #
coef2 = cv_best.coef_[0]
feature_names = X.columns
# sort them out in descending order
indices = np.argsort(np.absolute(coef2))
indices = np.flip(indices, axis=0)
# limit to 5 features
indices = indices[:5]
for i in indices:
    print(feature_names[i], ':', coef2[i])
```

```
number_inpatient : 0.5192022958161517
number_emergency : 0.3114841568408783
diabetesMed : 0.15431224186529618
number_outpatient : 0.11440629583283536
medical_specialty_Cardiology-Pediatric : 0.09524684082536003
```

F) Classification of model on training and test data was 63.56% & 63.19%

```
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: 0.6356360224105098
Test accuracy: 0.6318500869341233
```

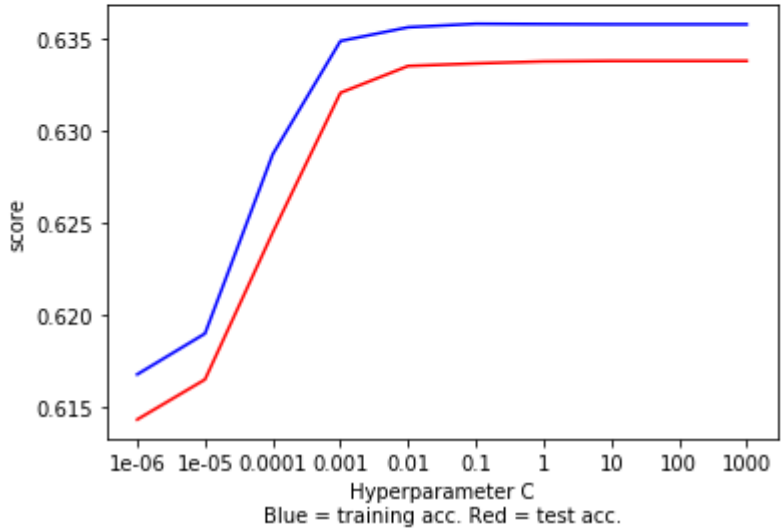
```
# classification report on test data
y_pred = cv.predict(X_test)
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.62      0.80      0.70      8381
           1       0.65      0.43      0.52      7148

   micro avg       0.63      0.63      0.63     15529
   macro avg       0.64      0.62      0.61     15529
weighted avg       0.64      0.63      0.62     15529
```

G) There is no situation where the training accuracy is significantly better than test accuracy, however we can see that from c= 0.01 and beyond, both accuracies plateaued.

```
import matplotlib.pyplot as plt
train_result = result_set['mean_train_score']
test_result = result_set['mean_test_score']
print("Total number of models: ", len(test_result))
# plot Hyperparameter C values vs training and test ac
plt.plot(range(0, len(train_result)), train_result, 'b
plt.xlabel('Hyperparameter C\nBlue = training acc. Rec
plt.xticks(range(0, len(train_result)),[pow(10, x) for
plt.ylabel('score')
plt.show()
```

```
Total number of models:  10
```



Build model ‘‘rfe_cv’

3. Build another regression model on the reduced variables set. Perform dimensionality reduction with Recursive feature elimination.

```
from sklearn.feature_selection import RFECV
rs=10
rfe = RFECV(estimator = LogisticRegression(random_state=rs), cv=10, n_jobs= -1)
rfe.fit(X_train, y_train) # run the RFECV

# comparing how many variables before and after
print("Original feature set", X_train.shape[1])
print("Number of features after elimination", rfe.n_features_)
```

```
Original feature set 124
Number of features after elimination 113
```

Tune the model with GridSearchCV to find the best parameter setting.

A) Dimension reduction was not useful to identify a good feature set for building the accurate model in this case. The accuracy is not improved but ever so slightly worsened.

B) Training accuracy was 63.56% & Test accuracy 63.18%

```
# grid search CV
params = {'C': [pow(10, x) for x in range(-6, 4)]}
rfe_cv = GridSearchCV(param_grid=params,
                      estimator=LogisticRegression(random_state=rs),
                      cv=10, n_jobs=-1)
rfe_cv.fit(X_train_rfe, y_train)
# test the best model
print("Train accuracy:", rfe_cv.score(X_train_rfe, y_train))
print("Test accuracy:", rfe_cv.score(X_test_rfe, y_test))
y_pred = rfe_cv.predict(X_test_rfe)
print(classification_report(y_test, y_pred))
# print parameters of the best model
print(rfe_cv.best_params_)
```

```
Train accuracy: 0.6355808241106174
Test accuracy: 0.631785691287269
              precision    recall  f1-score   support

           0       0.62      0.80      0.70      8381
           1       0.65      0.43      0.52      7148

   micro avg       0.63      0.63      0.63     15529
   macro avg       0.64      0.62      0.61     15529
weighted avg       0.64      0.63      0.62     15529

{'C': 1}
```

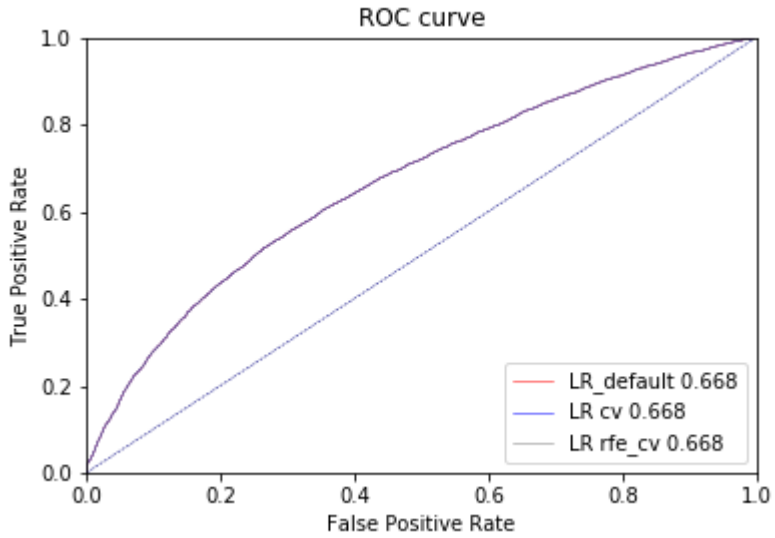
C)Similar to the previous model, there are no significant gaps where training accuracy is much higher than test accuracy. Hence no signs of overfitting.

D) The top 3 most important variables are;

```
# grab feature importances from the model and feature name from the model
coef3 = rfe_cv.coef_[0]
feature_names = X.columns
# sort them out in descending order
indices = np.argsort(np.absolute(coef3))
indices = np.flip(indices, axis=0)
# limit to 3 features
indices = indices[:3]
for i in indices:
    print(feature_names[i], ':', coef3[i])
```

```
number_inpatient : 0.5191144391032018
number_emergency : 0.311372690519885
diabetesMed : 0.1542734255982301
```

ROC curves;




```
ROC index on test for logestic regression_default: 0.668464522272278
ROC index on test for lr with gridsearch : 0.6684648394284859
ROC index on test for lr with feature selection and gridsearch: 0.6684444746614557
```

Using the best regression model, can you identify which patients could potentially be “readmitted”? Can you provide general characteristics of those patients?

Default model	ROC: 0.668464522	Very close to the highest
Grid Search CV	ROC: 0.668464839	Highest ROC index
Rfe + Grid Search CV	ROC: 0.668444474	Lowest ROC index

The default model tuned with GridSearchCV is the best model. According to the top-5 important variables in the model, we can see that if a patient has a large number of inpatient visits, more frequent emergency visits, if there was diabetic medication prescribed and more outpatient visits, then the patient is more likely to be readmitted.

```
number_inpatient : 0.5192022958161517
number_emergency : 0.3114841568408783
diabetesMed : 0.15431224186529618
number_outpatient : 0.11440629583283536
medical_specialty_Cardiology-Pediatric : 0.09524684082536003
```

Project c)

Neural Networks

1. Pre-Processing

For this dataset, the pre-processing involved filling 'na' values in the dataset with the modes for each variable where there was a nan. Modes were used as each variable was a categorical variable of some kind.

```
def data_processing(df):
    df.race.fillna(df.race.mode()[0], inplace=True)
    df.age.fillna(df.age.mode()[0], inplace=True)
    df.chlorpropamide.fillna(df.chlorpropamide.mode()[0], inplace=True)
    df.drop(['max_glu_serum', 'acetohexamide',  'change', 'tolbutamide', 'medical_specialty'], axis=1, inplace=True)
    df = pd.get_dummies(df)
    Y = df.readmitted
    df.drop(['readmitted'], axis=1, inplace=True)
    X = df
    return X, Y
```

For the same reason mentioned in the Decision tree task, 5 variables were dropped immediately. All data was then passed through a 'oneHotEncoder' in the pandas library so we could use the categorical variables in a neural network. This resulted in 65 features for the model to learn.

The data was then split into X and y variables, and then split into training and testing datasets. 30% of the data was allocated to the testing dataset. Leaving 70% for training purposes.

2. Training the Model

A)

The model was trained using the default parameters for the Sklearn MLPClassifier. According to the documentation, the default settings for this neural network are the following.

class sklearn.neural_network.**MLPClassifier**(*hidden_layer_sizes=100, activation='relu', *, solver='adam', alpha=0.0001, batch_size='auto', learning_rate='constant', learning_rate_init=0.001, power_t=0.5, max_iter=200, shuffle=True, random_state=None, tol=0.0001, verbose=False, warm_start=False, momentum=0.9, nesterovs_momentum=True, early_stopping=False, validation_fraction=0.1, beta_1=0.9, beta_2=0.999, epsilon=1e-08, n_iter_no_change=10, max_fun=15000*) [source]

a single hidden layer with 100 nodes, relu activation, alpha as 0.000, constant learning rate,

We can see that this model includes:

- a 100 nodes per hidden layer
- Relu activation
- Alpha = 0.0001
- Constant learning rate of 0.001
- Max interactions of 200 etc.

B) What is the classification accuracy on training and test datasets?

```
Train accuracy: 0.694778673142731
Test accuracy: 0.6254990341274952
              precision    recall  f1-score   support

           0       0.63      0.72      0.67      8343
           1       0.61      0.52      0.56      7187

    accuracy                           0.63     15530
   macro avg       0.62      0.62      0.62     15530
weighted avg       0.62      0.63      0.62     15530
```

The Neural Network with the default parameters was able to train to a training accuracy of 0.695 and a testing accuracy of 0.625.

C)

The model trained for 200 iterations. This could show that there was more training to be done on this model. This would make sense as the default layer size of 100 nodes. To be sure, the default model was trained separately with a max iterations count of 700. This model stopped early at iteration 227, but had worse training and testing accuracies (just). This shows that default parameters can converge at the best model.

3. Gridsearch on the Model

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

After performing gridsearch on a number of variables, (Hidden_layer_size, activation and alpha), the Gridsearch resulted in the following model parameters.

```
Train accuracy: 0.6716525002759687
Test accuracy: 0.6343206696716034
              precision    recall  f1-score   support

           0       0.65      0.69      0.67      8343
           1       0.61      0.57      0.59      7187

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

{'activation': 'logistic', 'alpha': 0.001, 'hidden_layer_sizes': (45,)}
```

This model was able to converge on an logistic activation function, alpha of 0.001 and a hidden layer size of 45 nodes.

B) What is the classification accuracy on training and test datasets?

This model resulted in a training accuracy of 0.672 and a testing accuracy of 0.634. This model had a lower training accuracy than the default model, however a higher test accuracy.

C) Did the training process converge and result in the best model?

Yes. This model was able to find the most appropriate model through a gridsearch and converge to the best

C) Do you see any sign of over-fitting?

With training accuracy and testing accuracy so close to each other, there is no sign of overfitting.

4. Feature Reduced NN

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

The top features (in terms of importance) were chosen from the decision tree. These variables were:

```
number_inpatient : 0.5847160154150457
discharge_disposition_id : 0.1599015974380108
number_emergency : 0.05631480009124843
number_outpatient : 0.0478839279662871
diabetesMed : 0.03494192905425346
number_diagnoses : 0.019512440604883895
num_medications : 0.01826039761953301
admission_type_id : 0.0171481831994704
num_lab_procedures : 0.014292740023506138
time_in_hospital : 0.013005944162757712
```

and Age. This reduced the feature size from 65 to 20.

Since alpha was found in the previous gridsearch to be best at 0.001, that was used.

When performing a gridsearch on fewer features, the model produced resulted in less nodes in the hidden layer, along with a higher testing and training accuracy than the non-reduced gridsearch model.

```
Train accuracy: 0.6505133016889282
Test accuracy: 0.6376690276883451
              precision    recall  f1-score   support

           0       0.63      0.78      0.70      8343
           1       0.65      0.48      0.55      7187

    accuracy                           0.64     15530
   macro avg       0.64      0.63      0.62     15530
weighted avg       0.64      0.64      0.63     15530

{'activation': 'logistic', 'alpha': 0.001, 'hidden_layer_sizes': (25,)}
```

The reduction in hidden layer size allowed for faster computation and for each node to learn features better.

b)

This model was able to reach a training accuracy of 0.651 and a test accuracy of 0.638, which is the closest between the training and testing accuracy so far.

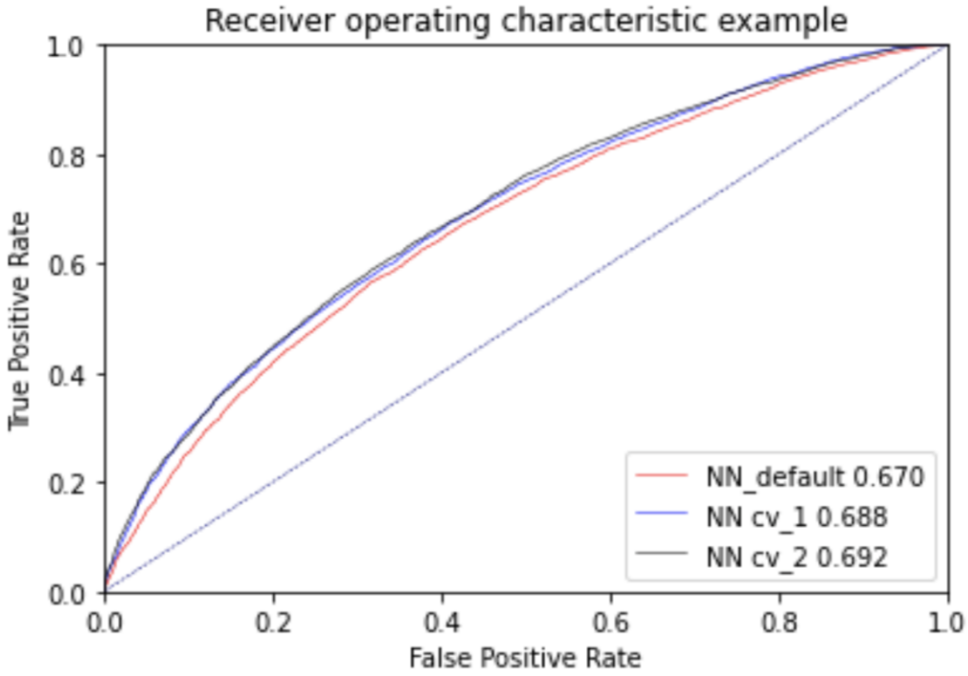
c. How many iterations are now needed to train this network?

```
Iteration 165, loss = 0.62211221
Training loss did not improve more than tol=0.000100 for 10 consecutive
epochs. Stopping.
Train accuracy: 0.6505133016889282
Test accuracy: 0.6376690276883451
              precision    recall  f1-score   support

           0       0.63      0.78      0.70      8343
           1       0.65      0.48      0.55      7187

    accuracy                           0.64     15530
   macro avg       0.64      0.63      0.62     15530
weighted avg       0.64      0.64      0.63     15530
```

Training a new network with the parameters from the gridsearch resulted in training the model in 165 iterations before early exiting. This is less than the default value for max_iter, showing we do not need to change it for training.

D) This model did not show signs of overfitting, and resulted in the best model out of the 3.

5. ROC Comparison



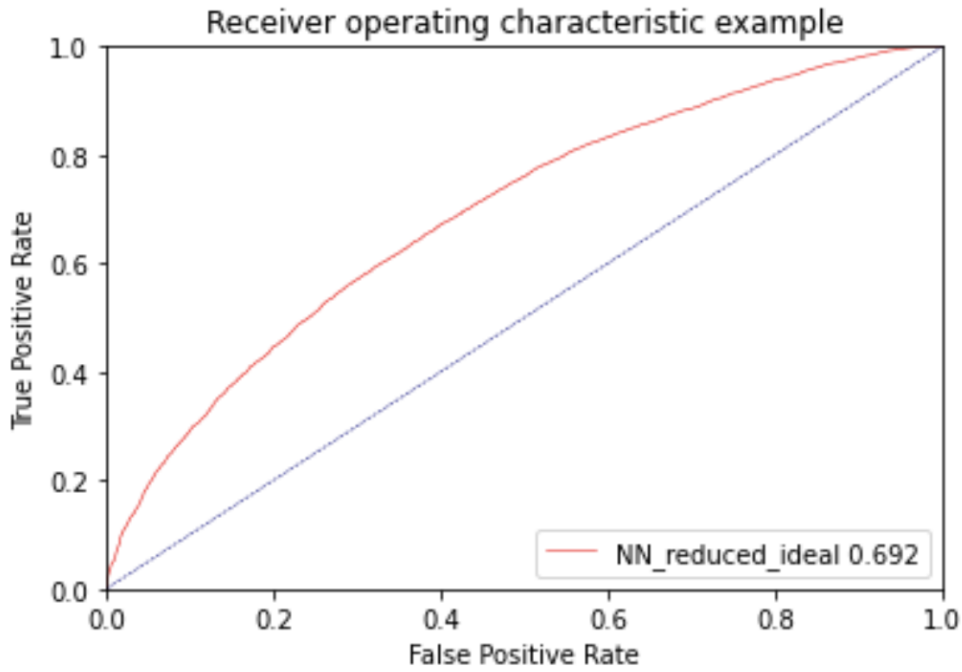
From the ROC curve for the NN's we can see that even though the default model had the best prediction score for the training data, it had the worst ROC Index value of 0.67.

Using Gridsearch for layer size, alpha and activation on the non reduced dataset proved to improve the ROC curve, resulting in an index of 0.688. This model was found to be most effective with a node count of 45, and a logistic activation function. This model used an alpha of 0.001.

Reducing the features to the most useful once (found by the decision tree) increased the ROC again up to 0.692. This model was found to be most effective with a node count of 25, alpha of 0.001, and a logistic activation function.

We were able to reduce the number of nodes in the hidden layer as the features were dramatically reduced. This allowed us to train the nodes more effectively as the feature per node was much lower in this mode (from 65 to 20 features)nn. This reduces computation time and also creates a more useful model.

Completing the ROC curve again for the reduced feature model , we can produce an ROC of the optimised model.



Unfortunately we cannot look at which feature was the most valuable in terms of predicting the model. This is the shortcoming of a neural network. The black box nature means that we cannot know which variable is more important. We can, however, look at the coefficients of the model to see which node is the most important though.

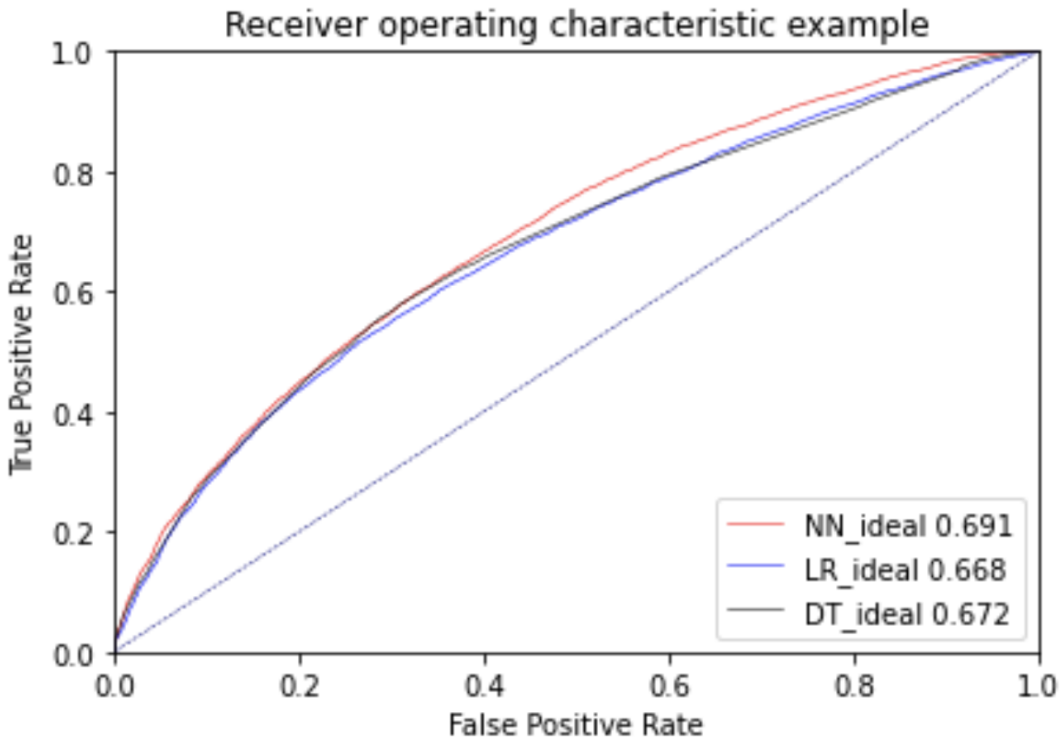
```
values = nn_model_2.coefs_[1]
max_v = 0
max_2 = 0
idx = 0
idx_2 = 0
for i in range(0,len(values)):
    if values[i] > max_v:
        max_2 = max_v
        idx_2 = idx
        max_v = values[i]
        idx = i

print(f'max val:{max_v} at idx:{idx}')
print(f'end max val:{max_2} at idx:{idx_2}')
```

```
max val:[1.99170518] at idx:4
end max val:[0.32915492] at idx:3
```

From here, we can see that the most important node happens at index 4 with a value of 1.99170518. This means the 5th node adds the most value to the model. The second most important is node 4 (index of 3) with a value of 0.32915492.

Final Comparison

1) Using the data from the 3 model types, we can produce an ROC curve for each.



From this ROC curve, we can see that the Neural Network had the best ROC score, but only just. This shows that while the neural network can produce a better performing model, it might not be worth using due to computational complexity. There are other methods that can be used, which are quicker to train and to predict, which can produce good results, on par with the best model possible.

For each model used, the training and testing accuracies for the best model were,

Model	Training Accuracy	Testing Accuracy
Decision Tree	64.3%	64.1%
Logistic Regression	63.6%	63.2%
Neural Network	65.1%	63.8%

2) From this, we can see that all models performed, more or less, equally. The best performing model was the neural network for training data, and the decision tree for testing accuracy. This shows that the decision tree learned the features the best out of all the models.

We were also able to get the most information out of the decision tree about the features used, and their level of importance. This is very useful as we are able to determine the biggest contributors correctly identifying readmitted patients.

The neural network performed very well and was able to learn the feature set to a good accuracy. However, the model itself can be computationally more expensive to make (especially going further into the deep neural networks) than a decision tree. You're also unable to get the importance of each variable in detail like the decision tree.

The logistic regression model is the middle ground between the decision tree and the neural network. We are able to learn the importance of each variable through the model's coefficients; however it was out performed by both the decision tree and the neural network in training and testing accuracies.

For the reasons above, the best model to choose for this problem would be the decision tree. It has the best testing accuracy, and allows for more in depth analysis of the features than the neural network does.