DATA EXPLORATION AND MINING (IFN509)

Team Name: [Assess 2]
Group No. [8]

Student Name	Student Id
Xu Han (Rymin)	n10306986
Xing Ming	n10380515
Tong Xu	n10428917

As for Decision Tree, Logistic Regression and Neuron Network models, it may generate various results, which may be associated with the development environment. Our experiment is conducted on the QUT computer lab.

Tutor Name: Sharon Pingi
Due Date: 25/10/2020

Contents

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

1. What variables did you include in the analysis? Justify your choice

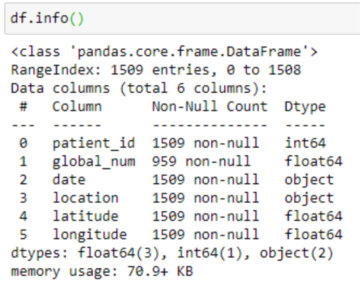
In this question, it requires to perform data analysis by establishing an association mining model. The objective is to identify the common routes (frequent routes) that positive patients have travelled. From D1.csv dataset, we aim to select some of the attributes for this purpose. A table will be donated to illustrate what attributes we selected and why they were selected

Attribute	Description	Status	Reason
patient_id	Unique identifier of the positive patient	Selected	It is a crucial variable used for indicating a particular patient, which is a unique identifier. The combination of this variable and location can contribute to the routes that a positive patient has visited.
location	Location of the patient's visit. It is a combination of province and city. For example, if 'Seoul' is the province and 'Jung-gu' is the city, the location will be recorded as 'Seoul_Jung-gu'		The attribute can specify a clear route for a patient, which is key information that conveys where the patient has travelled. The information includes provide and city. We aim combinate this attribute with the patient identifier to track the entire routes. This attribute will be donated to the transaction table for calculating association rules, i.e. Support, Confidence and Lift.
date	Date of patient's visit at the location		The variable can clearly present the date that a positive patient has travelled to a specific city. We aim to keep the attribute for finding temporal sequential patterns (rules). In reality, we may need a sequential database, sorted by the date, to discover a series of frequent subsequences (i.e., satisfying the minimum support threshold), where each subsequence presents several cities that a patient has frequently visited.
global_num	Unique identifier of the positive patient given by the Korea Centers for Control and Prevention	Not selected	The attribute is identical to patient_id, but it is compelling to convince that a patient is recognised by the Korea Cneters for Control Prevention. In this experiment, we do not consider data reliability, which means the attribute is not required for data analysis. It can be defined as a **redundant** attribute, requiring to drop away to reduce the data dimensionality.
latitude	Latitude coordinate of the location		Both attributes were not useful to contribute to solving the purpose. In reality, we may not need to consider a specific position travelled by a patient, in the current stage. The information, i.e. the province or city, is sufficiently truth-telling patients' routes. More importantly, the coordinate is not efficient to explore the associative rules, as it contains numerous data, that indicates the same location. However, in terms of extensive exploration, we can consider the use of coordinates to rank the risk level for a particular area in a city. In reality, we may say that a high-risk area may be visited by many patients; otherwise, the area may be placed on a low-risk degree. The map is a decent tool to virtualise the purpose, it is an extensive design, which will not implement in this assignment.
longitude	Longitude coordinate of the location		

2. What pre-process was required on the dataset before building the association mining model?

Data pre-processing is a crucial step that can ensure the data quality by the noise clearance and data amendment, the aim of which is to lay the ground for data analysis. In this dataset, we employed **Human Inspection** (HI) and **automatic validation by Pandas** (AVP) methods to identify the data problem. A table will be donated to illustrate our findings, where the below table will present what data problems we have found, what variable(s) involves this problem, and what method(s) we utilise to find out. Besides, elaborative problem description and solution will be appended to this table.

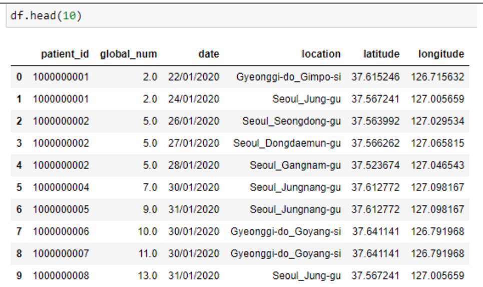
Dataset Basic Information



```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1509 entries, 0 to 1508
Data columns (total 6 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   patient_id  1509 non-null   int64
 1   global_num  959 non-null    float64
 2   date        1509 non-null   object
 3   location    1509 non-null   object
 4   latitude    1509 non-null   float64
 5   longitude   1509 non-null   float64
dtypes: float64(3), int64(1), object(2)
memory usage: 70.9+ KB
```

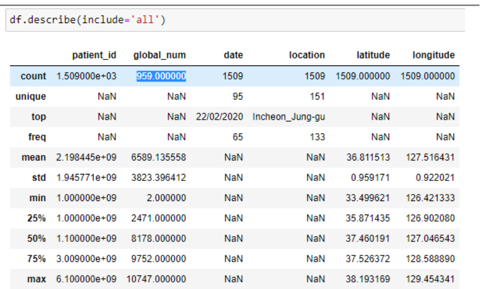
Top-10 records



```
df.head(10)
```

	patient_id	global_num	date	location	latitude	longitude
0	1000000001	2.0	22/01/2020	Gyeonggi-do_Gimpo-si	37.615246	126.715632
1	1000000001	2.0	24/01/2020	Seoul_Jung-gu	37.567241	127.005659
2	1000000002	5.0	26/01/2020	Seoul_Seongdong-gu	37.563992	127.029534
3	1000000002	5.0	27/01/2020	Seoul_Dongdaemun-gu	37.566262	127.065815
4	1000000002	5.0	28/01/2020	Seoul_Gangnam-gu	37.523674	127.046543
5	1000000004	7.0	30/01/2020	Seoul_Jungnang-gu	37.612772	127.098167
6	1000000005	9.0	31/01/2020	Seoul_Jungnang-gu	37.612772	127.098167
7	1000000006	10.0	30/01/2020	Gyeonggi-do_Goyang-si	37.641141	126.791968
8	1000000007	11.0	30/01/2020	Gyeonggi-do_Goyang-si	37.641141	126.791968
9	1000000008	13.0	31/01/2020	Seoul_Jung-gu	37.567241	127.005659

Dataset Statistic Measures

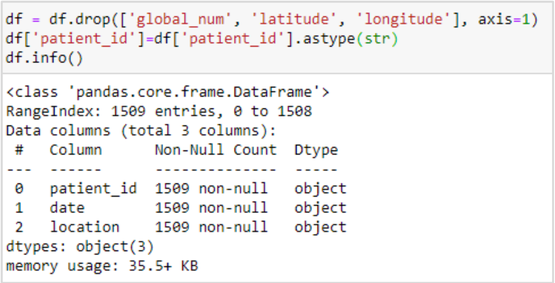


```
df.describe(include='all')
```

	patient_id	global_num	date	location	latitude	longitude
count	1.509000e+03	959.000000	1509	1509	1509.000000	1509.000000
unique	NaN	NaN	95	151	NaN	NaN
top	NaN	NaN	22/02/2020	Incheon_Jung-gu	NaN	NaN
freq	NaN	NaN	65	133	NaN	NaN
mean	2.198445e+09	6589.135558	NaN	NaN	36.811513	127.516431
std	1.945771e+09	3823.396412	NaN	NaN	0.959171	0.922021
min	1.000000e+09	2.000000	NaN	NaN	33.499621	126.421333
25%	1.000000e+09	2471.000000	NaN	NaN	35.871435	126.902080
50%	1.100000e+09	8178.000000	NaN	NaN	37.460191	127.046543
75%	3.009000e+09	9752.000000	NaN	NaN	37.526372	128.588890
max	6.100000e+09	10747.000000	NaN	NaN	38.193169	129.454341

Data Problem	Variables	Method	Problem Description	Amendment Strategy
Data Type	Patient_id	HI & AVP	The Patient_id is a unique identifier of a positive patient. It is suitably described as a nominal type, which is beneficial to the current mining task, so that we aim to convert the type of this attribute to string.	change the type of patiend_id to the string by using AVP.
Data Type	Global_num	HI & AVP	The Global_num is a unique and credible identifier of the positive patient, assigned by the Korea Centers for Disease Control and Prevention. The nominal type (string) is suitable for the attribute.	Its function is similar to Parent_ID, but it is with many data problems. For reducing the dimensionality of data and improving the efficiency of the data process, we aim to drop out the attributes by the use of AVP.
Data consistencies Missing Value	Global_num		It accounts for 550 Missing Value records, with rough 36.4% of the total records (1509).	
Redundant Attributes			The functionality of this attribute is similar to Parent_id in the current task, so that we recognise the feature may be redundant one, being necessary to drop out.	
Irrelevant Attributes	Latitude Longitude	AVP	The attributes are not the candidates for the current association mining task, so that they can be identified as irrelevant attributes. Also, both attributes can be replaced by the location variable, thus they can be defined as the redundant features.	For reducing the dimensionality of data and improving the efficiency of data processing, we aim to drop out the attributes by the use of AVP.

After Pre-process



```
df = df.drop(['global_num', 'latitude', 'longitude'], axis=1)
df['patient_id']=df['patient_id'].astype(str)
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1509 entries, 0 to 1508
Data columns (total 3 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   patient_id  1509 non-null   object
 1   date        1509 non-null   object
 2   location    1509 non-null   object
dtypes: object(3)
memory usage: 35.5+ KB
```

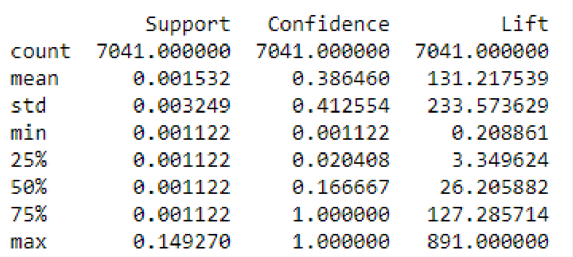
3. Conduct association mining and answer the following:

a. What is the 'min_support' threshold set and discuss why it is chosen?

In general, the definition of a sensible threshold is reliant on **experimentation**. Here, we donate the Apriori algorithm to discover frequent patterns. A parameter for minimum support is mandatory to be given ahead. This support refers to how the if/then statement (relationship) frequently occurs in the dataset. In the below experiment, we aim to **combine minimum support and confidence** for the discovery of frequent routes that patients have travelled. This confidence refers to the number of times the association rule is true in a transactional dataset.

Threshold: min_sup = 0.001 & min_con = 0

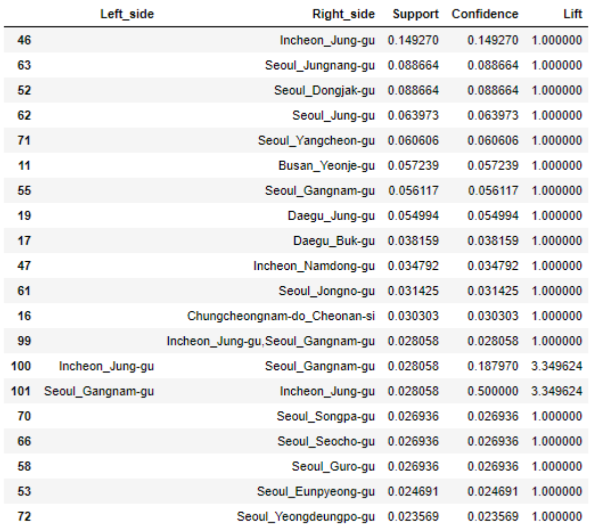
Initially, we define a **small threshold** to explore as much as possible association rules. 7041 transactional records are generated, and by the statistic that the actual maximum and minimum values for Support (0.001122 and 0.14927 respectively) contribute to the upper and lower bounds to Support threshold. Here, we do not analyse the boundary to Confidence, as its range is too broad [0.001122 to 1.0].



```
          Support   Confidence         Lift
count  7041.000000  7041.000000  7041.000000
mean      0.001532     0.386460   131.217539
std       0.003249     0.412554   233.573629
min       0.001122     0.001122     0.208861
25%       0.001122     0.020408     3.349624
50%       0.001122     0.166667    26.205882
75%       0.001122     1.000000   127.285714
max       0.149270     1.000000   891.000000
```

Threshold: min_sup = 0.005 & min_con = 0

We start by slightly adjusting the initial threshold [0.001 to 0.005] and keeping minimum confidence to 0. A transactional dataset with 135 association rules is created, and we aim to sort it by Support in decreasing order. On the **top-20 records**, numerous transactional records with empty left side indicate no correlation, as their Lift values equal 1. This data may not be practically useful to the current data mining purpose, because we are desirable to discover the potential routes that a patient has visited. For example, the association rule (No. 100) is easily explainable that a patient went to Incheon_Jung-gu, and then who was likely to visit Seoul_Gangnam-gu. The similar transactional records can be found in No.101. We have noticed that the non-correlated records (Lift = 1) hold the Confidence that is less than 0.1 (inclusive). Thus, next stage, we will adjust Confidence to 0.1 for exploration.

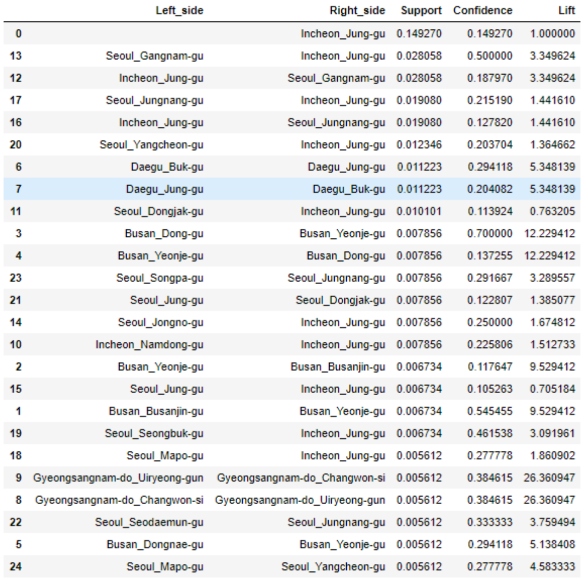


	Left_side	Right_side	Support	Confidence	Lift
46		Incheon_Jung-gu	0.149270	0.149270	1.000000
63		Seoul_Jungnang-gu	0.088664	0.088664	1.000000
52		Seoul_Dongjak-gu	0.088664	0.088664	1.000000
62		Seoul_Jung-gu	0.063973	0.063973	1.000000
71		Seoul_Yangcheon-gu	0.060606	0.060606	1.000000
11		Busan_Yeonje-gu	0.057239	0.057239	1.000000
55		Seoul_Gangnam-gu	0.056117	0.056117	1.000000
19		Daegu_Jung-gu	0.054994	0.054994	1.000000
17		Daegu_Buk-gu	0.038159	0.038159	1.000000
47		Incheon_Namdong-gu	0.034792	0.034792	1.000000
61		Seoul_Jongno-gu	0.031425	0.031425	1.000000
16		Chungcheongnam-do_Cheonan-si	0.030303	0.030303	1.000000
99		Incheon_Jung-gu,Seoul_Gangnam-gu	0.028058	0.028058	1.000000
100	Incheon_Jung-gu	Seoul_Gangnam-gu	0.028058	0.187970	3.349624
101	Seoul_Gangnam-gu	Incheon_Jung-gu	0.028058	0.500000	3.349624
70		Seoul_Songpa-gu	0.026936	0.026936	1.000000
66		Seoul_Seocho-gu	0.026936	0.026936	1.000000
58		Seoul_Guro-gu	0.026936	0.026936	1.000000
53		Seoul_Eunpyeong-gu	0.024691	0.024691	1.000000
72		Seoul_Yeongdeungpo-gu	0.023569	0.023569	1.000000

Threshold: min_sup = 0.005 & min_con = 0.1

By this threshold, we have 25 records made up to a transactional table. The result is desirable, because we have 24 records that contribute to various routes that patients have visited. Hence, we say that minimum support (0.005) and confidence (0.1) are suitable to exploit the potential association rules in the dataset.

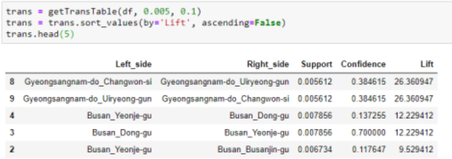
In summary, we initially discover the threshold range and dataset characteristics, then we slightly modify experimental parameters to explore the change, and finally we propose the ideal parameter, which is supported by experimental data. Note that we only retain the final experimental parameters, which is encouraged to perform the experiment individually, by the use of the code provided.



	Left_side	Right_side	Support	Confidence	Lift
0		Incheon_Jung-gu	0.149270	0.149270	1.000000
13	Seoul_Gangnam-gu	Incheon_Jung-gu	0.028058	0.500000	3.349624
12	Incheon_Jung-gu	Seoul_Gangnam-gu	0.028058	0.187970	3.349624
17	Seoul_Jungnang-gu	Incheon_Jung-gu	0.019080	0.215190	1.441610
16	Incheon_Jung-gu	Seoul_Jungnang-gu	0.019080	0.127820	1.441610
20	Seoul_Yangcheon-gu	Incheon_Jung-gu	0.012346	0.203704	1.364662
6	Daegu_Buk-gu	Daegu_Jung-gu	0.011223	0.294118	5.348139
7	Daegu_Jung-gu	Daegu_Buk-gu	0.011223	0.204082	5.348139
11	Seoul_Dongjak-gu	Incheon_Jung-gu	0.010101	0.113924	0.763205
3	Busan_Dong-gu	Busan_Yeonje-gu	0.007856	0.700000	12.229412
4	Busan_Yeonje-gu	Busan_Dong-gu	0.007856	0.137255	12.229412
23	Seoul_Songpa-gu	Seoul_Jungnang-gu	0.007856	0.291667	3.289557
21	Seoul_Jung-gu	Seoul_Dongjak-gu	0.007856	0.122807	1.385077
14	Seoul_Jongno-gu	Incheon_Jung-gu	0.007856	0.250000	1.674812
10	Incheon_Namdong-gu	Incheon_Jung-gu	0.007856	0.225806	1.512733
2	Busan_Yeonje-gu	Busan_Busanjin-gu	0.006734	0.117647	9.529412
15	Seoul_Jung-gu	Incheon_Jung-gu	0.006734	0.105263	0.705184
1	Busan_Busanjin-gu	Busan_Yeonje-gu	0.006734	0.545455	9.529412
19	Seoul_Seongbuk-gu	Incheon_Jung-gu	0.006734	0.461538	3.091961
18	Seoul_Mapo-gu	Incheon_Jung-gu	0.005612	0.277778	1.860902
9	Gyeongsangnam-do_Uiryeong-gun	Gyeongsangnam-do_Changwon-si	0.005612	0.384615	26.360947
8	Gyeongsangnam-do_Changwon-si	Gyeongsangnam-do_Uiryeong-gun	0.005612	0.384615	26.360947
22	Seoul_Seodaemun-gu	Seoul_Jungnang-gu	0.005612	0.333333	3.759494
5	Busan_Dongnae-gu	Busan_Yeonje-gu	0.005612	0.294118	5.138408
24	Seoul_Mapo-gu	Seoul_Yangcheon-gu	0.005612	0.277778	4.583333

b. Report the top 5 frequently occurring rules and interpret them.

By the aforementioned threshold, we have discovered a set of frequent association rules, but in this question, we aim to know the top-5 frequently occurring ones. The Lift measure can help to achieve this purpose, which usually the strength of an association rule, indicating **the predictive power to the chance**. We may say high-frequency rules are associated with high Lift.



```
trans = getTransTable(df, 0.005, 0.1)
trans = trans.sort_values(by='Lift', ascending=False)
trans.head(5)
```

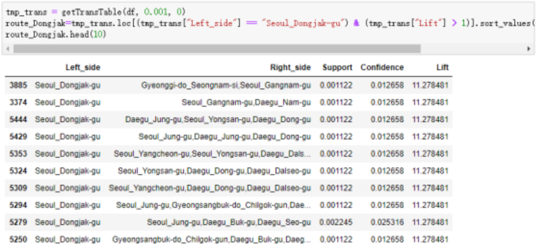
	Left_side	Right_side	Support	Confidence	Lift
8	Gyeongsangnam-do_Changwon-si	Gyeongsangnam-do_Uiryeong-gun	0.005612	0.384615	26.360947
9	Gyeongsangnam-do_Uiryeong-gun	Gyeongsangnam-do_Changwon-si	0.005612	0.384615	26.360947
4	Busan_Yeonje-gu	Busan_Dong-gu	0.007856	0.137255	12.229412
3	Busan_Dong-gu	Busan_Yeonje-gu	0.007856	0.700000	12.229412
2	Busan_Yeonje-gu	Busan_Busanjin-gu	0.006734	0.117647	9.529412

Result Analysis

In term of the five records, they hold the relatively lower values of Support, fluctuating by 0.005 to 0.008, which means the most frequent rules indicate the lower probability of occurrence of the route that a patient visits Left_side and Right_side locations (0.5% to 0.8% of all patients travelling to these routes). As for Confidence, in No.3, we can say that if a patient has visited Busan_Dong-gu, he/she is most likely to visit Busan_Yeonje-gu, as it holds 0.7 Confidence. However, in this dataset, only 0.8% of all patients get a trip to the route. Compared with No.3 rule, No.8 and No.9 indicate the lower likelihood of occurrence of the event (Support = 0.005612), but they hold 38% Confidence. It indicates that if patients have travelled to Gyeongsangnam-do_Changwon-si, they are possible to Gyeongsangnam-do_Uiryeong-gun, and vice versa. No.4 and No.2 rules present relative lower confidence, with just around 14% and 12% respectively. We may say that if a patient has visited Busan_Yeonje-gu, the person would go to Busan_Dong-gu or Busan_Busanjin-gu. In other words, the rules may be less likely to occur. When it comes to Lift measure, they hold the values that are always more than 1, so that we say the top-5 rules are reliable (strong association).

4. Identify at least 10 common routes that positive patients from the Seoul province in the Dongjak-gu city have travelled.

We employ the pre-defined threshold (see Q3.a) to find the common routes, where it starts by Seoul province in the Dongjak-gu City, but no record has been found out. By adjusting the parameters, the 10 common routes can be fetched out when the min_sup = 0.001 and min_con = 0. In reality, **the 10 routes are fairly rare to be travelled by patients in the dataset**, so that we need to reduce the threshold to detect the 10 special association rules. Regardless of Support or Confidence, they are supportive of this opinion. Although their Lift values are far more than 1, we only say the strong relationship between each rule, instead of saying these rules way frequently occur. Each rule indicates that only approximate 0.1% of all patients have visited the route (started from Seoul_Dongjak-gu to other cities). In reality, the association rules may not be beneficial to data analysis, due to the lack of sufficient data to prove that the probability of occurring the event.

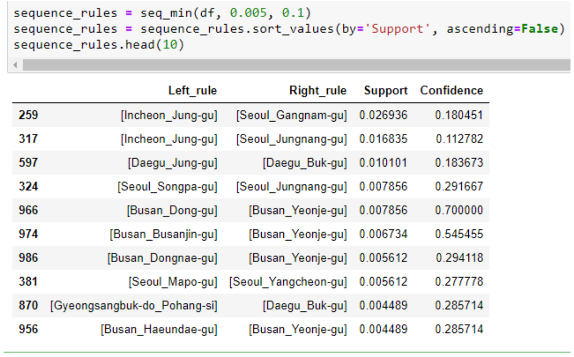


```
tmp_trans = getTransTable(df, 0.001, 0)
route_Dongjak=tmp_trans.loc[(tmp_trans["Left_side"] == "Seoul_Dongjak-gu") & (tmp_trans["Lift"] > 1)].sort_values(
route_Dongjak.head(10)
```

	Left_side	Right_side	Support	Confidence	Lift
3885	Seoul_Dongjak-gu	Gyeonggi-do_Seongnam-si,Seoul_Gangnam-gu	0.001122	0.012658	11.278481
3374	Seoul_Dongjak-gu	Seoul_Gangnam-gu,Daegu_Nam-gu	0.001122	0.012658	11.278481
5444	Seoul_Dongjak-gu	Daegu_Jung-gu,Seoul_Yongsan-gu,Daegu_Dong-gu	0.001122	0.012658	11.278481
5429	Seoul_Dongjak-gu	Seoul_Jung-gu,Daegu_Jung-gu,Daegu_Dong-gu	0.001122	0.012658	11.278481
5353	Seoul_Dongjak-gu	Seoul_Yangcheon-gu,Seoul_Yongsan-gu,Daegu_Dals...	0.001122	0.012658	11.278481
5324	Seoul_Dongjak-gu	Seoul_Yongsan-gu,Daegu_Dong-gu,Daegu_Dalseo-gu	0.001122	0.012658	11.278481
5309	Seoul_Dongjak-gu	Seoul_Yangcheon-gu,Daegu_Dong-gu,Daegu_Dalseo-gu	0.001122	0.012658	11.278481
5294	Seoul_Dongjak-gu	Seoul_Jung-gu,Gyeongsangbuk-do_Chilgok-gun,Dae...	0.001122	0.012658	11.278481
5279	Seoul_Dongjak-gu	Seoul_Jung-gu,Daegu_Buk-gu,Daegu_Seo-gu	0.002245	0.025316	11.278481
5250	Seoul_Dongjak-gu	Gyeongsangbuk-do_Chilgok-gun,Daegu_Buk-gu,Daeg...	0.001122	0.012658	11.278481

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

The dataset is suitable for discovering frequent sequential patterns, where this pattern is a sub-sequence that occurs in many sequences of a dataset. For instance, after a given threshold, we have a sequential pattern [Incheon_Jung-gu, Busan_Dong-gu] in the dataset and it appears many times in the sequence table, which means a patient visits Incheon_Jung-gu and then he/she will travel to Busan_Dong-gu based on a probability. In reality, we not only focus on the sequential patterns, but also primarily concentrate on sequential rules (X -> Y), which is beneficial to sequence analysis as it allows decision-makers to get insight into the dataset, e.g. finding frequent sequential routes that a patient usually travels. In this dataset, the **'Date' is the crucial attribute that can contribute to a sequence table**, where the routes to a patient will be sorted by the temporal ascending order. The purpose to perform sequence analysis is to explore the probability of sequential events, e.g. during a period, if a patient gets trip to city A, we may explore what possibility the person could sequentially travel to city B, then city C, etc.



```
sequence_rules = seq_min(df, 0.005, 0.1)
sequence_rules = sequence_rules.sort_values(by='Support', ascending=False)
sequence_rules.head(10)
```

	Left_rule	Right_rule	Support	Confidence
259	[Incheon_Jung-gu]	[Seoul_Gangnam-gu]	0.026936	0.180451
317	[Incheon_Jung-gu]	[Seoul_Jungnang-gu]	0.016835	0.112782
597	[Daegu_Jung-gu]	[Daegu_Buk-gu]	0.010101	0.183673
324	[Seoul_Songpa-gu]	[Seoul_Jungnang-gu]	0.007856	0.291667
966	[Busan_Dong-gu]	[Busan_Yeonje-gu]	0.007856	0.700000
974	[Busan_Busanjin-gu]	[Busan_Yeonje-gu]	0.006734	0.545455
986	[Busan_Dongnae-gu]	[Busan_Yeonje-gu]	0.005612	0.294118
381	[Seoul_Mapo-gu]	[Seoul_Yangcheon-gu]	0.005612	0.277778
870	[Gyeongsangbuk-do_Pohang-si]	[Daegu_Buk-gu]	0.004489	0.285714
956	[Busan_Haeundae-gu]	[Busan_Yeonje-gu]	0.004489	0.285714

We utilise the previous configurations (Support = 0.005 and Confidence = 0.1) to generate the sequential table. The result can be reflected contributes to top-10 interesting patterns, sorted by Support in decreasing order. The highest Support only constitutes 0.027 approximately, which indicates the percentage of patients who travelled to Incheon_Jung-gu and then go to Seoul_Gangnam-gu accounts for 2.7% (less chance to occur). Also, the confidence implies that if a patient has visited Incheon_Jung-gu, the probability of visiting Seoul_Gangnam-gu subsequently is 18%. In fact, this may not be a strong rule, but it is a frequent sequent rule truth-telling us a possible route that a patient travelled. We have noticed that No.966 holds the highest Confidence, but its Support is relatively lower. It indicates that If a patient goes to Busan_Dong-gu, the event of person who go to Busan_Yeonje-gu is more likely to occurs, because of its Confidence (70%). However, this rule tells that only 0.7% of all patients will travel to this route.

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

In the descriptive strategy, association mining task aims to observe a historical dataset, exploring patterns frequently occurred, the pattern of which is similar to a statement with an antecedent (if) and a consequent (then). The relationship is hidden in various attributes of a dataset. We are desirable to detect this relationship for clarifying the goal to improve.

During the course of the experiment, our outcomes are be summarised as:

1. A simple methodology to explore a threshold (Minimum Support and Confidence) in the dataset
2. Top-5 frequent association rules (sorted by Lift)

3. Top-10 common routes that positive patients from the Seoul province in the Dongjak-gu city have travelled (sorted by Lift)
4. Top-10 sequential rules (sorted by Support)

We assume that decision-makers are governments or the Korea Centers for Control and Prevention. We do not discuss the 1st outcome, as the decision-makers may not be benefited from this methodology. In terms of outcomes 2nd, 3rd and 4th, the contributions can be concluded as three bullet-point:

- **2nd result** -> only approximate 0.8% of patients visit Busan_Yeonje-gu after travelling to Busan_Dong-gu, but it could say that if patients **have** visited Busan_Dong-gu, they are most likely to travel to **Busan_Yeonje-gu**, on account of 70% Confidence. If patients have travelled to Gyeongsangnam-go_Changwon-si, they are also likely to **Gyeongsangnam-do_Uiryeong-gun**, because of 38% Confidence. Decision-makers could refer the result, taking measures to prevent Covid-19 outbreak on both routes. If requires, the quarantine or medicines may be prepared in advance.

- **3rd result** -> in this dataset, travelling started from **Seoul_Dongjak-gu** are special routes, as it holds the minimum Support. Also, their Confidence only accounts for just over 1%. It could say that Seoul_Dongjak-gu has positive patients, but they rarely go out of the city. It seems that the spread of coronavirus is restricted by local. As for decision-makers, they may need to allocate social resources (e.g. doctors or medicines) to the city for controlling Covid19, without need for considering other adjacent cities.

- **4th result** -> the sequence table contributes to various frequent sequential rules to decision-makers. At the top-10 sequential rules, although a few patients get trip to the two routes (No.966 and No.974), both routes hold more than 50% Confidence. It means that Busan_Yeonje-gu may have many patients who come from other cities, so that decision-makers may need to take steps (e.g. setting quarantines to people who come from Busan_Dong-gu or Busan_Busanjin-gu) to control the risk for Busan_Yeonje-gu.

The benefits are not restricted to the above-mentioned usage. The ultimate goal to provide the result with decision-makers is to ensure that they can identify the risk level for a route, and then perform resource allocation to control Covid-19 outbreak, recovering positive patients.

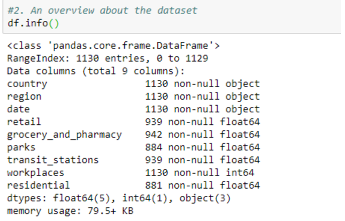
Project (b): Clustering Mobility Data

1. Identify the data quality problems in this dataset such as unusual data types, missing values, etc, and explain how to fix them?

Human Inspection (HI) and Automatic Validation by Pandas (AVP) are donated to discover data problems (shows in Table b1) for dataset D2.csv, the aim of which is to enhance data quality for improving the classification performance. The table b2 will illustrate what data problems we have found, what variable(s) involves this problem, and what method(s) we utilise to find out. Besides, elaborative problem description and solution will be appended to this table. At the last row will present other findings of data problems, but it does not require to tinker in the current clustering purpose.

Table b1

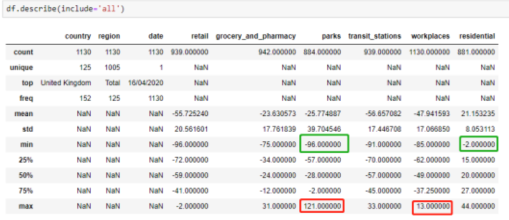
Dataset Basic Information



```
#2. An overview about the dataset
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1130 entries, 0 to 1129
Data columns (total 9 columns):
country                 1130 non-null object
region                  1130 non-null object
date                    1130 non-null object
retail                  939 non-null float64
grocery_and_pharmacy    942 non-null float64
parks                   884 non-null float64
transit_stations        939 non-null float64
workplaces              1130 non-null int64
residential             881 non-null float64
dtypes: float64(5), int64(1), object(3)
memory usage: 79.5+ KB
```

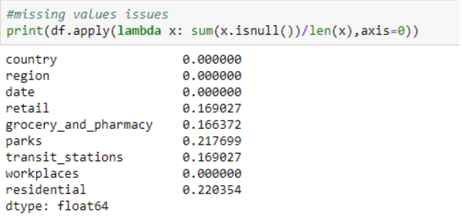
Dataset Statistic Measures



```
df.describe(include='all')
```

	country	region	date	retail	grocery_and_pharmacy	parks	transit_stations	workplaces	residential
count	1130	1130	1130	939.000000	942.000000	884.000000	939.000000	1130.000000	881.000000
unique	125	1005	1	NaN	NaN	NaN	NaN	NaN	NaN
top	United Kingdom	Total	16/04/2020	NaN	NaN	NaN	NaN	NaN	NaN
freq	152	125	1130	NaN	NaN	NaN	NaN	NaN	NaN
mean	NaN	NaN	NaN	-55.725240	-23.630573	-25.774887	-56.657082	-47.941593	21.153235
std	NaN	NaN	NaN	20.561601	17.761839	39.704546	17.446708	17.066850	8.053113
min	NaN	NaN	NaN	-96.000000	-75.000000	-96.000000	-91.000000	-85.000000	-2.000000
25%	NaN	NaN	NaN	-72.000000	-34.000000	-57.000000	-70.000000	-62.000000	15.000000
50%	NaN	NaN	NaN	-59.000000	-24.000000	-28.000000	-57.000000	-49.000000	20.000000
75%	NaN	NaN	NaN	-41.000000	-12.000000	-2.000000	-45.000000	-37.250000	27.000000
max	NaN	NaN	NaN	-2.000000	31.000000	121.000000	33.000000	13.000000	44.000000

Missing Value Detection



```
#missing values issues
print(df.apply(lambda x: sum(x.isnull())/len(x),axis=0))
```

```
country                 0.000000
region                  0.000000
date                    0.000000
retail                  0.169027
grocery_and_pharmacy    0.166372
parks                   0.217699
transit_stations        0.169027
workplaces              0.000000
residential             0.220354
dtype: float64
```

Displot Distribution

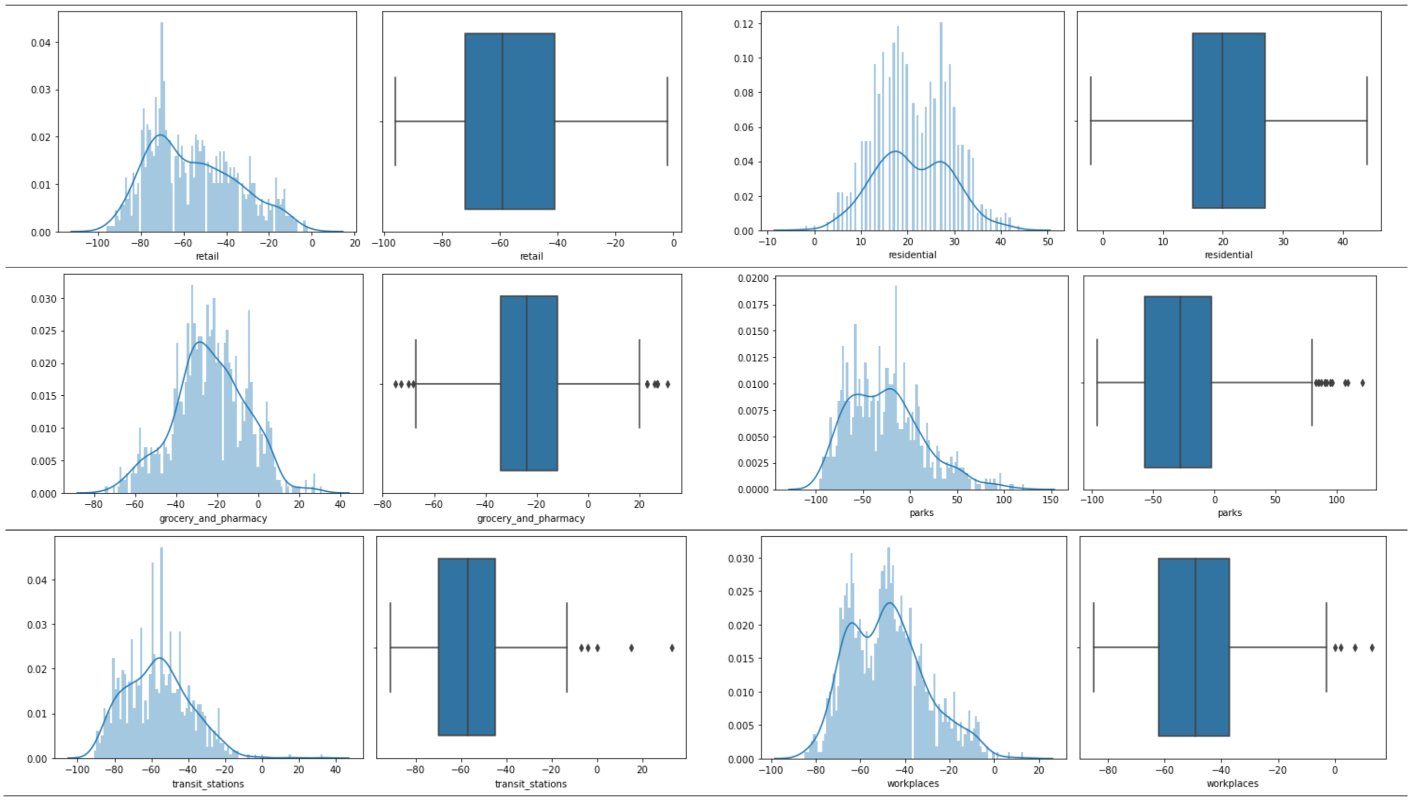


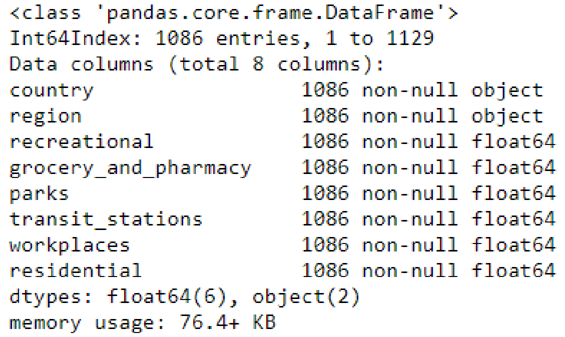
Table b2

Data Problem	Variables	Method	Problem Description	Amendment Strategy
Inappropriate Data Type	workplaces	AVP	In reality, floating-point representation is beneficial to clustering technique. Based on the variable type assigned by Pandas, it is noticed that workplaces are the attribute with an integer type. Hence, we aim to transform it to float type.	Employ Pandas (AVP) to correct the int64 to float64 type.
Incorrect Name for Attributes	retail	AVP	As the tutor's requirement, the tile needs to be corrected as Recreational.	Modify 'Retail' to 'Recreational' on code-based.
Data consistencies Missing Value	retail grocery_and_pharmacy parks transit_stations residential	AVP	According to statistic, in gross 1130 instances, we employ Pandas to discover the five numeric attributes that contain the Missing Value problem (Dataset Missing Value Detection), with more than leaking > 16%. The attributes are important to mobility analysis so that we aim to impute the Mean of each feature to this issue, enabling the K-Means and Hierarchical Agglomerative Clustering (HAC) to work properly.	Employ Pandas to impute the Mean to each feature
Unary Attributes	date	HI	The value to this column is for 14/04/2020, which is a unary attribute and not beneficial to a clustering task. For reducing the dimensionality of data and improving the efficiency of data processing, it is required to be dropped.	Employ Pandas (AVP) to drop the column.
Data over-lapping	region	AVP	There is a value "total" in the attribute region, which is aggregated by various unknown regions. As no description about the value, we aim to separately handle the record based on two situations: one is if a country only contains the record with "total" value, we retain it for clustering; another one is when a country contains a set of records that include "total" value, we will remove the instance. The optimal method may	Employ Pandas (AVP) to delete the rows based the two conditions.

			be slightly beneficial to model performance in classification.	
Attribute Value with Significantly Different Scale	retail grocery_and_pharmacy parks transit_stations workplaces residential	AVP	From the Dataset Statistic Measures in Table b1, it is noticed that attributes are with different scale, e.g. min (park = -96) vs min (residential = -2) and max (parks = 121) vs max (workplaces = 13). As clustering techniques (e.g. K-Means & HAC) are reliant on proximity measures and sensitive to attribute scale, we consider tap standardisation method to re-scope the value range.	Apply standard scaling technique when inputting variables (***The result can refer to Task d***)
Outliers	grocery_and_pharmacy parks transit_stations workplaces	AVP	We boxplot all the attributes to identify the outlier's issues. From the graphs, we notice that the outliers are near the Maximum and Minimum values, which cannot tell these outliers as errors.	No operations on handling outliers.
Note: other findings in relation to non-ASCII characters in the region column, e.g. **Ä€daÅ¾i Municipality** and **CÃ´te d'Ivoire**, but it is out scope of thinking the issue, thus we will keep it.				

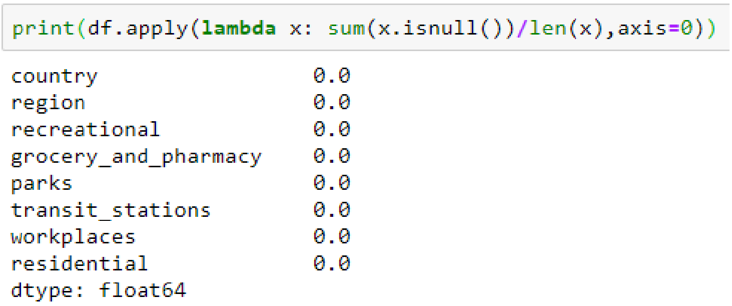
After Pre-process

Dataset Basic Informations



```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1086 entries, 1 to 1129
Data columns (total 8 columns):
country                 1086 non-null object
region                  1086 non-null object
recreational            1086 non-null float64
grocery_and_pharmacy    1086 non-null float64
parks                   1086 non-null float64
transit_stations        1086 non-null float64
workplaces              1086 non-null float64
residential             1086 non-null float64
dtypes: float64(6), object(2)
memory usage: 76.4+ KB
```

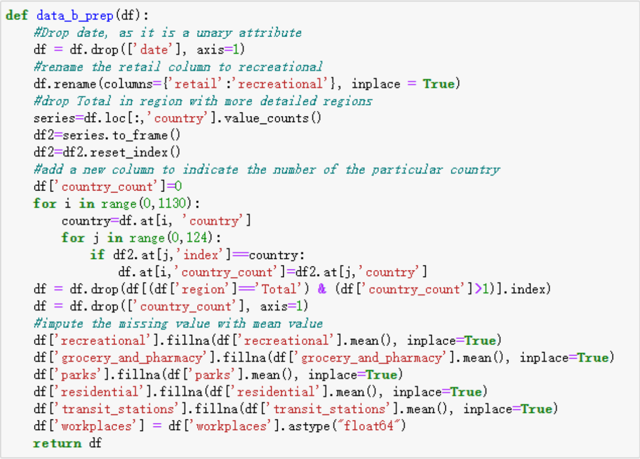
Missing Value Detection



```
print(df.apply(lambda x: sum(x.isnull())/len(x),axis=0))
```

```
country                 0.0
region                  0.0
recreational            0.0
grocery_and_pharmacy    0.0
parks                   0.0
transit_stations        0.0
workplaces              0.0
residential             0.0
dtype: float64
```

Executable Code

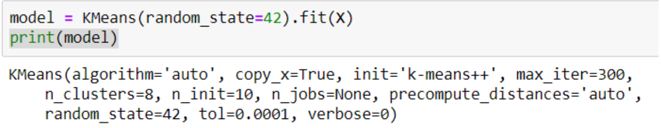


```
def data_b_prep(df):
    #Drop date, as it is a unary attribute
    df = df.drop(['date'], axis=1)
    #rename the retail column to recreational
    df.rename(columns={'retail':'recreational'}, inplace = True)
    #drop Total in region with more detailed regions
    series=df.loc[:,'country'].value_counts()
    df2=series.to_frame()
    df2=df2.reset_index()
    #add a new column to indicate the number of the particular country
    df['country_count']=0
    for i in range(0,1130):
        country=df.at[i, 'country']
        for j in range(0,124):
            if df2.at[j,'index']==country:
                df.at[i,'country_count']=df2.at[j,'country']
    df = df.drop(df[(df['region']=='Total') & (df['country_count']>1)].index)
    df = df.drop(['country_count'], axis=1)
    #impute the missing value with mean value
    df['recreational'].fillna(df['recreational'].mean(), inplace=True)
    df['grocery_and_pharmacy'].fillna(df['grocery_and_pharmacy'].mean(), inplace=True)
    df['parks'].fillna(df['parks'].mean(), inplace=True)
    df['residential'].fillna(df['residential'].mean(), inplace=True)
    df['transit_stations'].fillna(df['transit_stations'].mean(), inplace=True)
    df['workplaces'] = df['workplaces'].astype("float64")
    return df
```

2. Build a clustering model to profile the category of locations visited by the people.

a. What clustering algorithm have you used?

- **K-Means Clustering Approach**



```
model = KMeans(random_state=42).fit(X)
print(model)
```

```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
    n_clusters=8, n_init=10, n_jobs=None, precompute_distances='auto',
    random_state=42, tol=0.0001, verbose=0)
```

We aim to utilise K-Means to experiment for profiling the category of locations, e.g. recreational places, parks or residential. This state-of-the-art clustering technique pertains one of the partitioning approaches, the rationale of which is reliant on adjusting centroids to define various clusters. More specifically, the algorithm starts by randomly picking K points as the initial cluster centres. By proximity measures (e.g. Euclidean or Manhattan Distance Formula), objects will be automatically grouped a cluster when they are closest to the centre point centroid of this cluster. Subsequently, the centroid will be underlying the mean of a formed new cluster to iteratively change the position. This is a repetitive process until all objects find their optimal clusters (termination). Note that the algorithm requires a hyperparameter K to scope the number of clusters. Also, the clusters are usually Flat Structure, which is beneficial to data analysis.

- **Reason for the choice of K-Means**

Two aspects enable us to consider the use of K-Means to the current dataset (D2.csv). **Efficiency** is as the strength of K-Means, as it contributes to O(tkn) time complexity during clustering process, where *t* represents the gross number of iterations in this process, *k* symbolises the number of clusters given, and *n* refers to the number of objects in the dataset. Note that *k* and t are usually constants, which may not pose a significant impact on efficiency. If the dataset is small, the efficiency of this algorithm may hold O(n). In this experiment, we will employ another clustering technique (Hierarchical Agglomerative Clustering) to categorise the dataset, but its time complexity is at least O(n2), which is not better than K-Means. This new technique will elaborate on **Question 4.** Thus, we prioritise K-Means to perform classification on the dataset. **The experimental objective** is another consideration. According to the analysis in Question 1, we only need some of the numeric attributes, e.g. parks or residentials, to achieve the mining purpose. Compared with other variations of K-Means, i.e. **K-Mode (Categorical Data Handle)** or **K-prototype (Mixed Data Handle)**, K-Means could be the suitable candidate, as there are no categorical attributes provided to this experiment.

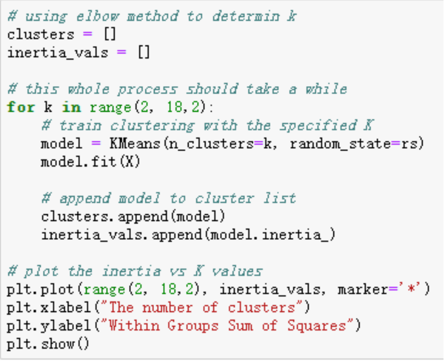
b. List the attributes used in this analysis.

The purpose of this experiment is to group the geographic regions that experience a similar changing tendency, in terms of the public areas, e.g. parks and recreations. The change is caused by Covid-19 outbreak. In this dataset, we only focus on the change for a single day (16/04/2020), with 1130 instances provided. As for unsupervised clustering technique (K-Means), it is only supportive to operate vectors. Hence, in this dataset, we select the 6 numeric attributes [**Recreational (Retail)**, **Grocery_and_pharmacy**, **Parks**, **Transit_station**, **Workplaces**, **Residential**] to form vectors, where each observation contributes to a vector. The representation for an instance is exampled as [-63, -32, -74, -79, -55, 34] during clustering. Each vector will be donated to K-Means for proximity measure calculated by Similarity Algorithm. All the vectors will be clustered by the results, ensuring each vector to be close to its cluster (high intra-class) and excluded to neighbour cluster (low inter-class). Based on the rationale, the 6 attributes are suitable to be candidates forming the vectors, thereby analysing the changing tendency in terms of various regions due to Covid-19.

Here, **Date** will not be selected, as it is a **unary** attribute, and it will be dropped out. In predictive mining, the attributes '**Country'** and '**Region'** are not choosing as the experiment, the reason of which is that **ID-like variables** are usually represented for a particular record and their numeric values are unique and non-meaningful. In other words, ID-like attributes will not be worth when clustering an unlabelled dataset.

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

The optimal number of clusters is 4 (K value) to the dataset, which is defined by experimentation with Elbow method and K-Means clustering technique (Silhouette Coefficient). In this experiment, we also employ Silhouette Coefficient to define how good performance to each cluster. More specifically, we initiate a K-Means that loads instances with vectorised representation [Recreational (Retail), Grocery_and_pharmacy, Parks, Transit_station, Workplace, and Residential] in advance. Here, to ensure the results generated by K-Means to be consistent along with multiple runs, Random Seed should be embedded during performing clustering, the suitable number of which can be 42 (defined by experiments). The selection of the number of clusters is based on 2n, e.g. the first run is started by 2 clusters and then 4 until to 18, which is referred from the IFN509 tutorial in Week 7 and 8 Computer Tutorial. In this stage, we can test the performance by donating a bag of numbers (2, 4, 6 … 18) to K-Means.



```
# using elbow method to determin k
clusters = []
inertia_vals = []

# this whole process should take a while
for k in range(2, 18,2):
    # train clustering with the specified K
    model = KMeans(n_clusters=k, random_state=rs)
    model.fit(X)

    # append model to cluster list
    clusters.append(model)
    inertia_vals.append(model.inertia_)

# plot the inertia vs K values
plt.plot(range(2, 18,2), inertia_vals, marker='*')
plt.xlabel("The number of clusters")
plt.ylabel("Within Groups Sum of Squares")
plt.show()
```

Figure b.1

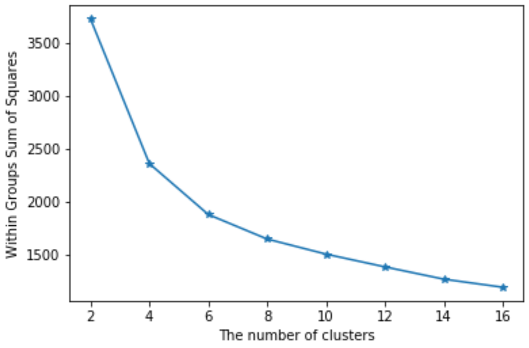
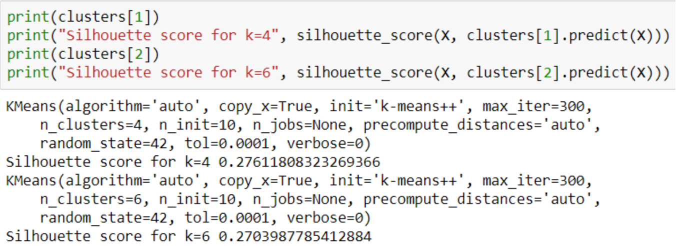


Figure b.2 (processed by standardisation)

The result shows in Figure b.2. In the Y-axis, it indicates the clustering error (also called inertia or cost in sklearn library), while X-axis represents the number of clusters fed into K-Means for the experiment. It is noticed that the larger number of clustering error, the small number of clusters is donated. The reason is when the larger number of clusters could contribute to the reduction of the intra-cluster distances, so that the clustering error could be reduced. However, it is encouraged to select the max number of clusters, although it contributes to the lower number of clustering error. This is because it is not beneficial to efficiency, and in reality, we aim to find a minimum K for training model in an efficient way. Here, Elbow will be a helpful method to define the optimal K. In terms of the rationale of Elbow, it is encouraged to find the K range that reflects an abrupt decrease (called Elbow Effect). In Figure b.2, the bend range is targeted by 4 to 6.



```
print(clusters[1])
print("Silhouette score for k=4", silhouette_score(X, clusters[1].predict(X)))
print(clusters[2])
print("Silhouette score for k=6", silhouette_score(X, clusters[2].predict(X)))
```

```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
    n_clusters=4, n_init=10, n_jobs=None, precompute_distances='auto',
    random_state=42, tol=0.0001, verbose=0)
Silhouette score for k=4 0.27611808323269366
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
    n_clusters=6, n_init=10, n_jobs=None, precompute_distances='auto',
    random_state=42, tol=0.0001, verbose=0)
Silhouette score for k=6 0.2703987785412884
```

By defining the K range, we employ Silhouette Coefficient to evaluate K-Means performance, the coefficient of which is scoped by -1 to 1. <u>If the coefficient is close to -1 that symbolises samples in a cluster are out of scope to the neighbouring clusters, while it is close to 1 that indicates the configuration to the cluster many are inappropriate and samples may be assigned to an improper cluster</u>. Here, we only examine the K values [4 and 6]. The highest number of Silhouette score is generated when K = 4, with approximate 0.28 (2 Decimals Keep), thus we are desirable to select the optimal K value as 4.

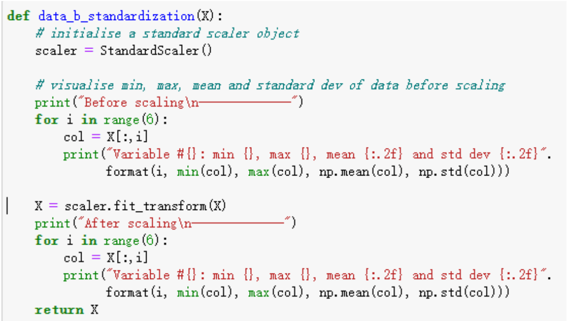
Extensive Knowledge

Two Reasons are compensative to explain why we select the techniques to define K = 4 for the current dataset.

- Silhouette Evaluation may be infeasible to the large size of the dataset, as it may cause an extra process for evaluation. However, the dataset is small, as it only contains 1130 observations.
- Elbow Method is encouraged to define the K value at the first valley (called local minima). Usually, we may have multiple peak/valley in an Elbow plot. It is not recommended to choose another peak/valley, as it may bring to the overfitting issue.

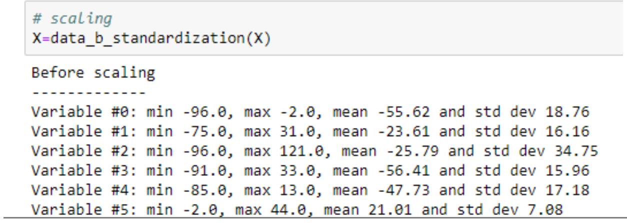
d. Did you normalise the variables? What was its effect on the model–Does the variable normalization process enable a better clustering solution?

We did standardization on the variables, i.e. recreational (retail), grocery_and_pharmacy, parks, transit_stations, workplaces, and residential, in the pre-processing step. The general-purpose to clustering techniques is that they are usually sensitive to inputs on a different scale. This is because these techniques are reliant on the proximity (distance-based) measure, where the most common approach is Euclidian distance, that is a distance formula favoured features on a larger scale. In clustering mining, the attribute with a wider range of values tends to weight more on what defines clusters, because it likely has greater distance. According to observation, from Figure b.4, we notice that the Variable #2 range from -96 to 121, while Variable #5 ranges from 3 to 44. To narrow the gap between variables, we apply standardization to enables the weighting scale of each feature equal by transforming it into a relative distance, shows in Figure b.4.1, which is reflected that the value ranges for each feature are relatively equal and scoped into a small range.



```
def data_b_standardization(X):
    # 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(6):
        col = X[:,i]
        print("Variable #{}: min {}, max {}, mean {:.2f} and std dev {:.2f}".
              format(i, min(col), max(col), np.mean(col), np.std(col)))

    X = scaler.fit_transform(X)
    print("After scaling\n————————————")
    for i in range(6):
        col = X[:,i]
        print("Variable #{}: min {}, max {}, mean {:.2f} and std dev {:.2f}".
              format(i, min(col), max(col), np.mean(col), np.std(col)))
    return X
```

Before Standardisation

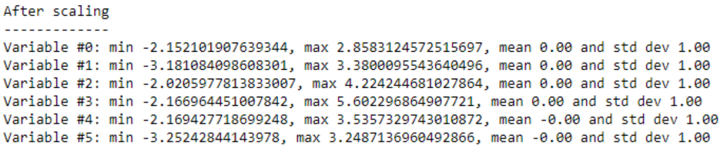


```
# scaling
X=data_b_standardization(X)
```

```
Before scaling
-------------
Variable #0: min -96.0, max -2.0, mean -55.62 and std dev 18.76
Variable #1: min -75.0, max 31.0, mean -23.61 and std dev 16.16
Variable #2: min -96.0, max 121.0, mean -25.79 and std dev 34.75
Variable #3: min -91.0, max 33.0, mean -56.41 and std dev 15.96
Variable #4: min -85.0, max 13.0, mean -47.73 and std dev 17.18
Variable #5: min -2.0, max 44.0, mean 21.01 and std dev 7.08
```

Figure b.4

After standardisation

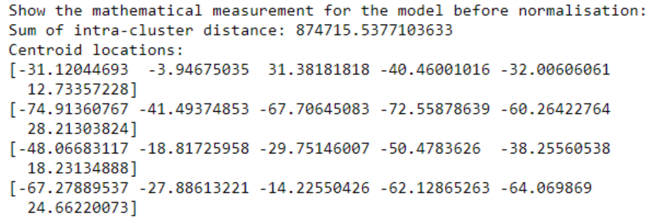


```
After scaling
-------------
Variable #0: min -2.152101907639344, max 2.8583124572515697, mean 0.00 and std dev 1.00
Variable #1: min -3.181084098608301, max 3.3800095543640496, mean 0.00 and std dev 1.00
Variable #2: min -2.0205977813833007, max 4.224244681027864, mean 0.00 and std dev 1.00
Variable #3: min -2.166964451007842, max 5.602296864907721, mean 0.00 and std dev 1.00
Variable #4: min -2.169427718699248, max 3.5357329743010872, mean -0.00 and std dev 1.00
Variable #5: min -3.25242844143978, max 3.2487136960492866, mean -0.00 and std dev 1.00
```

Figure b.4.1

K-Means relies on Euclidian Distance (default configuration) to perform instances classification. We aim to design a simple experiment to standardise the variables [Recreational (Retail), Grocery_and_pharmacy, Parks, Transit_station, Workplace, and Residential] for exploring how the result could be improved. A comparative conclusion to illustrate the influence is generated by the standardisation before and after. Note that the standard scaling may be helpful to handle outlier issues. We create a K-means model, performing classification twice (before and after standard scaling) for the comparison, the results of which are shown in Figure b.5 and Figure b.5.1.

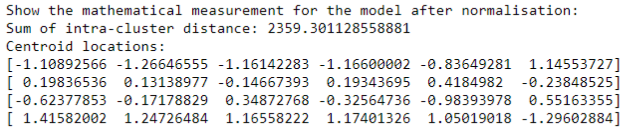
Before Standardisation



```
Show the mathematical measurement for the model before normalisation:
Sum of intra-cluster distance: 874715.5377103633
Centroid locations:
[-31.12044693  -3.94675035  31.38181818 -40.46001016 -32.00606061
  12.73357228]
[-74.91360767 -41.49374853 -67.70645083 -72.55878639 -60.26422764
  28.21303824]
[-48.06683117 -18.81725958 -29.75146007 -50.4783626  -38.25560538
  18.23134888]
[-67.27889537 -27.88613221 -14.22550426 -62.12865263 -64.069869
  24.66220073]
```

Figure b.5

After standardisation



```
Show the mathematical measurement for the model after normalisation:
Sum of intra-cluster distance: 2359.301128558881
Centroid locations:
[-1.10892566 -1.26646555 -1.16142283 -1.16600002 -0.83649281  1.14553727]
[ 0.19836536  0.13138977 -0.14667393  0.19343695  0.4184982  -0.23848525]
[-0.62377853 -0.17178829  0.34872768 -0.32564736 -0.98393978  0.55163355]
[ 1.41582002  1.24726484  1.16558222  1.17401326  1.05019018 -1.29602884]
```

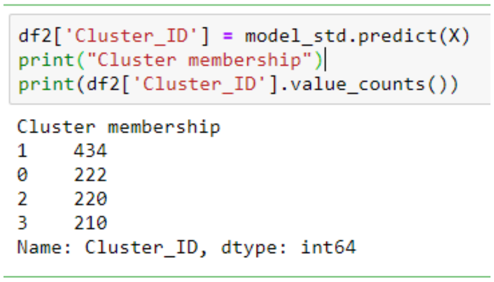
Figure b.5.1

The summation of intra-cluster distance to Figure b.5 is significantly larger than the number in Figure b.5.1, with 874715.54 and 2359.30 respectively. Another effect is on the actual value in the vectors of 4 centroids, where the values in Figure b.5 are substantially greater than the ones in Figure b.5.1. **The significant effect is to say that the K-Means model gets improvement and is well-performed after standardisation. This is because the approach contributes to the reduction of the intra-cluster distance, which is beneficial to form high intra-cluster class**. In conclusion, the outcome of our experiment is successful to discover that the dataset is suitable for clustering to 4 clusters. To improve the clustering result (minimise the intra-cluster distance), the experimental 6 attributes are encouraged to be standardised.

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

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

We aim to use 'pairplot' to virtualise the dataset after K-Means clustering. The 'pairplot' is a common approach used for presenting the distribution of variables in a given cluster, which is informative than text-based results, enabling to profile the overall clusters. We will continually employ the previous experimental configurations, e.g. the optimal K value (4) and standard scaling. The generated clusters (4) will be assigned to a unique cluster ID. After K-Means clustering, 4 clusters are generated with various instances. The cluster 1 is the largest group, with 434 instances, while the smallest group is cluster 2, with accounted for 210 observations.



```
df2['Cluster_ID'] = model_std.predict(X)
print("Cluster membership")
print(df2['Cluster_ID'].value_counts())
```

```
Cluster membership
1    434
0    222
2    220
3    210
Name: Cluster_ID, dtype: int64
```

Graph Analysis

The pairplot describes how different clusters distributes differently based on the 6 attributes. We do not intend to illustrate all cells, instead we focus on some of the distinct cells for illustration. We have defined 6 levels to clearly describe mobility tendency:

- High Positive Mobility
- Medium Positive Mobility
- Low Positive Mobility
- High Negative Mobility
- Medium Negative Mobility
- Low Negative Mobility

Positive and negative indicate the increase or decrease tendency, that is compared with the day before and after, when Covid-19 pandemic outbreak. High, low and medium reflect the various levels of change tendency on the single day (16/04/2020).

- **Row 1 & Column 6: Recreational (Retail) vs Residential**

  The cell indicates that the cluster 3 covers some regions with negative high mobility in terms of recreational places, e.g. shopping centres or museums, which means the number of people slightly decrease after the outbreak of Covid-19, but still perform high mobility compared with other regions due to Covid-19 analysed by a single day (16/04/2020). In the regions, it also with positive low mobility for staying at residences. Cluster 0 and 2 reflect the almost similar trend in terms of people's mobility. The cluster 2 implies negative low mobility in recreational places, with positive high mobility for residences.

- **Row 5 & Column 4: Transit_Stations & Workplaces**

  In this cell, it is noticed some points arranged in a line, which is caused by Mean imputation for Missing Value, but it does not curb data analysis. The cell indicates that the regions in cluster 1 and 3 contribute to high mobility in terms of workplaces and transport stations. By contrast, the cluster 0 and 2 reflect some regions that experience **low negative mobility** for the number of individuals.

- **Row 2 & Column 3: Grocery_and_pharmacy & Parks**

As for the cell, it indicates that the cluster 3 has high mobility in Grocery_and_pharmacy, with approximate range [-20 to +20], but it contributes high mobility in parks, with around range [0 to +100]. By contrast, cluster 1 and 2 are with medium negative mobility, in terms of the two places. The cluster 0 indicates that people may not go to Grocery, Pharmacy and Parks, due to the negative low mobility.

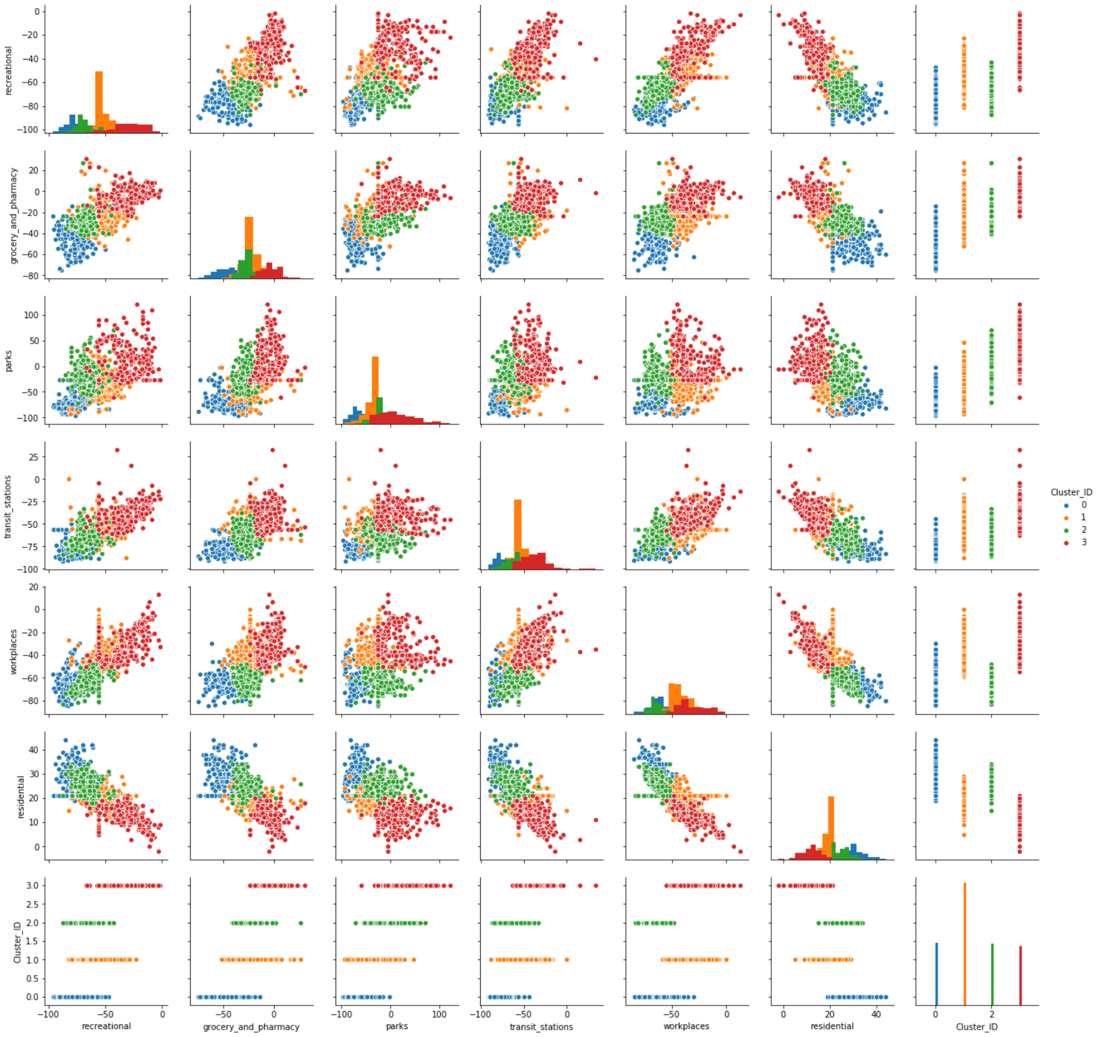
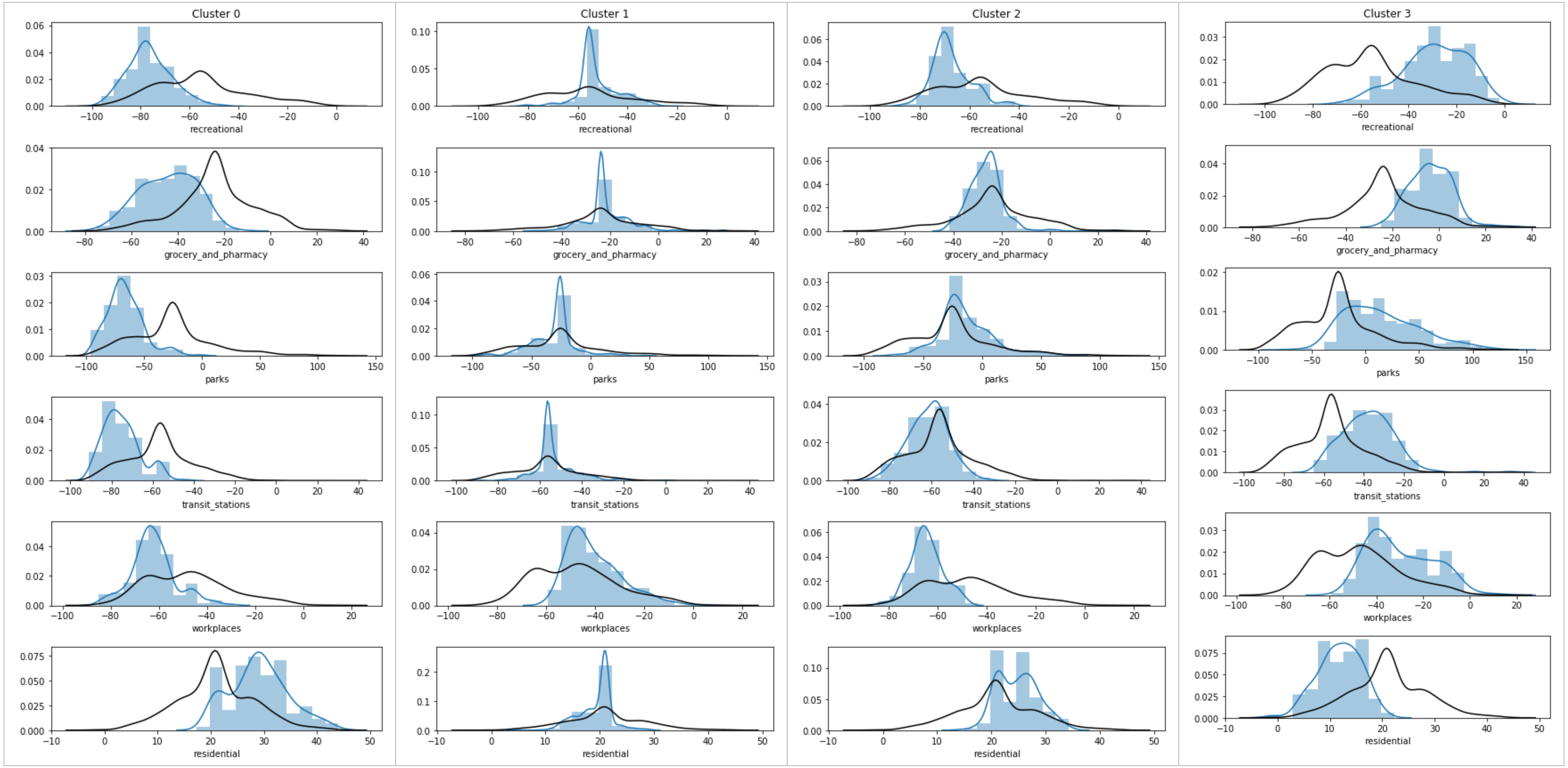


Figure b.7

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

From the "pairplot", we get an overview of the 4 clusters. Next, we utilize cluster distribution against the distribution from all data, to understand each of the specific clusters. We continually utilise the previous level configuration for the description.


Cluster 0
recreational
grocery_and_pharmacy
parks
transit_stations
workplaces
residential
Cluster 1
recreational
grocery_and_pharmacy
parks
transit_stations
workplaces
residential
Cluster 2
recreational
grocery_and_pharmacy
parks
transit_stations
workplaces
residential
Cluster 3
recreational
grocery_and_pharmacy
parks
transit_stations
workplaces
residential

Cluster 0: Higher Mobility residential (right-leaning), left-leaning grocery_and_pharmacy, transit_stations, parks, workplaces, and recreational. Regions in cluster 0 are with high people's mobility in residential (move-in), and less visiting the public places during Covid19 outbreak, e.g. Low Mobility among all regions to parks and transit stations, groceries, pharmacies and workplaces, with decreased volume compared with a baseline before the pandemic outbreak.

Cluster 1: Medium Mobility residential, right-leaning workplaces and the normal distributions in grocery_and_pharmacy, transit_stations, parks, and recreational. Among all regions, regions in cluster 1 are with medium and increased people's mobility in residential (move-in), and approximate medium and decreased people's mobility (move out) in the public places.

Cluster 2: Slightly higher Mobility residential (right-leaning), left-leaning recreational and workplaces, slight right leaning parks and grocery_and_pharmacy. Among all regions, regions in cluster 2 are with medium and increased people's mobility (move-in), medium and decreased mobility to recreational, transit_stations, and workplaces, and low and decreased mobility to groceries, pharmacies and parks.

Cluster 3: Low Mobility residential (left-leaning), right-leaning grocery_and_pharmacy, transit_stations, parks, workplaces, and residential. Among all regions, regions in cluster 3 are with low but slightly increased people's mobility (move-in) to residentials on the day. Regions in cluster 3 are with high mobility to public places. People tend to visit parks more than the day 16/04, grocery_and_pharmacy similar with baseline, and slightly decreased for recreational, transit_stations and workplaces.

4.Build another clustering model using an algorithm that helps to profile the people's mobility only in the following countries:

[Australia, United Kingdom, Japan, Latvia, Romania, Slovenia, Kenya]

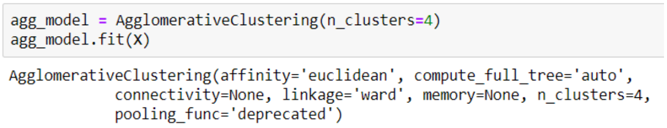
Use the best setting (e.g., variable normalisations, optimal K, etc) obtained in the previous model.

a. What clustering algorithm have you used?

In this task, we will keep the experimental settings (pre-processing, standard scaling, K = 4, and Euclidean Distance) and propose an innovative clustering algorithm for the current mining purpose, thereby profiling the individuals' mobility in terms of the 7 countries – Australia, United Kingdom, Japan, Latvia, Romania, Slovenia and Kenya.

Hierarchical Agglomerative Clustering (Agnes)

Hierarchical Agglomerative Clustering is the most popular technique applied for a small and unlabeled dataset. Agglomerative indicates a kind of categorial hierarchical approach, which is called a "bottom-up" method. More specifically, the algorithm initially treats each object (vector, that is formed Task 2B) as a singleton cluster. In term of every iteration, the pairs of sub-clusters (singleton clusters) are recursively merged as one, which is based on proximity measure, until an ultimate cluster is aggregated into all sub-clusters. In this stage, the algorithm will be terminated and a tree-based representation will be generated. Here, we name this representation as Dendrogram. Note that we aim to find the suitable number of clusters, so that the Dendrogram will be cutting as 4 sub-trees (also called 4 sub-clusters).



```
agg_model = AgglomerativeClustering(n_clusters=4)
agg_model.fit(X)
```

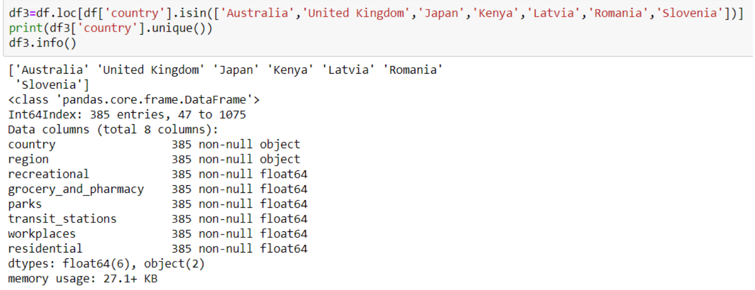
```
AgglomerativeClustering(affinity='euclidean', compute_full_tree='auto',
                        connectivity=None, linkage='ward', memory=None, n_clusters=4,
                        pooling_func='deprecated')
```

Reason for the choice of Hierarchical Agglomerative Clustering (Agnes)

Compared with K-Means, it could be identified as a non-parametric algorithm, without the need for pre-defining K value. However, in the experiment, we tap K-means to define an optimal K, which will use feeding into the cluster. It is useful to generate **cutting-tree** based on the given clusters. The cutting-tree refers that the Dendrogram can be sensibly separated into the number of clusters. Another reason is that the algorithm is advantageous to cluster a small dataset, where the current dataset only contains 1130 observations.

b. List the attributes used in this analysis

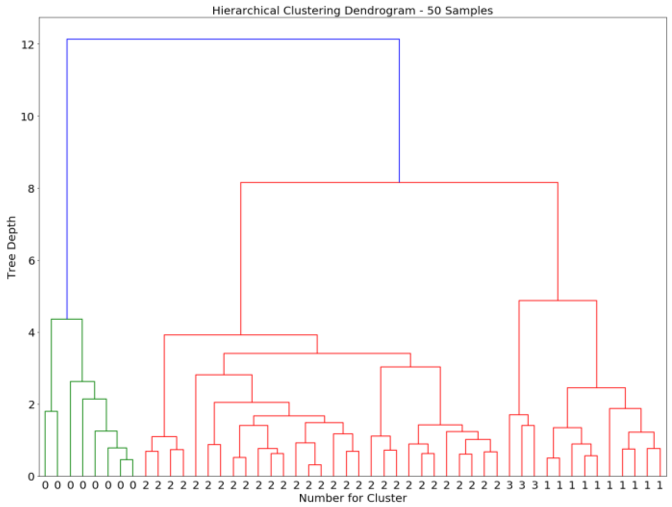
We aim to utilise the attributes that are similar to Question 2, including [*Recreational (Retail), Grocery_and_pharmacy, Parks, Transit_station, Workplace,* and *Residential*], to perform the experiment. However, the instances should be filtered by the 7 countries in the pre-processing stage. Thus, a subset extracted from the original dataset will be donated to implement the mining task. From Figure b.12, we get 385 instances after pre-process.



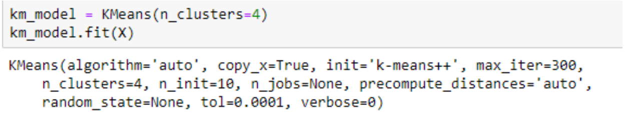
```
df3=df.loc[df['country'].isin(['Australia','United Kingdom','Japan','Kenya','Latvia','Romania','Slovenia'])]
print(df3['country'].unique())
df3.info()
```

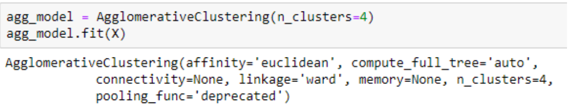
```
['Australia' 'United Kingdom' 'Japan' 'Kenya' 'Latvia' 'Romania'
 'Slovenia']
<class 'pandas.core.frame.DataFrame'>
Int64Index: 385 entries, 47 to 1075
Data columns (total 8 columns):
country                  385 non-null object
region                   385 non-null object
recreational             385 non-null float64
grocery_and_pharmacy     385 non-null float64
parks                    385 non-null float64
transit_stations         385 non-null float64
workplaces               385 non-null float64
residential              385 non-null float64
dtypes: float64(6), object(2)
memory usage: 27.1+ KB
```

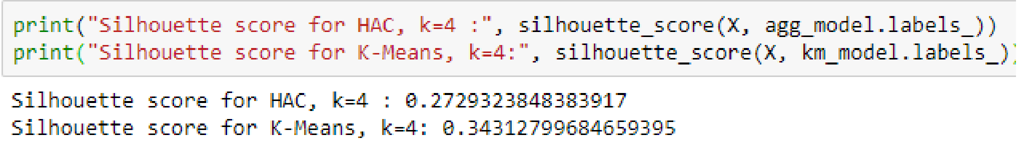
c. What difference do you see in this clustering interpretation when compared to the previous one?



The dendrogram is one of tree-based virtualization that is used to represent the classification result to Hierarchical Agglomerative Result. 50 samples are made up a tree with depth = 12, where the number for CLUSTER is clearly represented at the bottom of the tree. In this graph, the cluster is the larger one compared with others. Here, we apply the two clusters on the same dataset for the comparison. Also, the two models are donated to the optimal K value. A table is donated to compare with the required hyperparameters to classify, efficiency, approaches to virtualization, and silhouette scores.

<table>
<tr><th></th><th>K-Means
<pre>km_model = KMeans(n_clusters=4)
km_model.fit(X)

KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
    n_clusters=4, n_init=10, n_jobs=None, precompute_distances='auto',
    random_state=None, tol=0.0001, verbose=0)</pre></th><th>Hierarchical Agglomerative clustering (HAC)
<pre>agg_model = AgglomerativeClustering(n_clusters=4)
agg_model.fit(X)

AgglomerativeClustering(affinity='euclidean', compute_full_tree='auto',
            connectivity=None, linkage='ward', memory=None, n_clusters=4,
            pooling_func='deprecated')</pre></th></tr>
<tr><td><b>Hyperparameters Required</b></td><td>Require to define the number of clusters K</td><td>Non-parametric algorithm</td></tr>
<tr><td><b>Efficiency</b></td><td>O(knl)</td><td>O(n³)</td></tr>
<tr><td><b>Virtualization</b></td><td>Pairplot</td><td>Dendrogram, Pairplot</td></tr>
<tr><td rowspan="2"><b>Silhouette Score</b></td><td colspan="2"><pre>print("Silhouette score for HAC, k=4 :", silhouette_score(X, agg_model.labels_))
print("Silhouette score for K-Means, k=4:", silhouette_score(X, km_model.labels_))

Silhouette score for HAC, k=4 : 0.2729323848383917
Silhouette score for K-Means, k=4: 0.34312799684659395</pre></td></tr>
<tr><td>0.28</td><td>0.34</td></tr>
</table>

It is noticed that HAC is advantageous for performing categorization on the dataset, as it contributes 0.34 Silhouette Score, that is higher than K-Means. The dendrogram has clearly represented the distribution of data points in each cluster. However, this graph is not suitable for virtualizing the large dataset, as it may be complicated and uninterpretable.

Project (c): Building and Evaluating Predictive models

Predictive modelling using Decision Tree

1. Build a decision tree using the default setting.

1.1 Load Dataset and One-hot Transformation

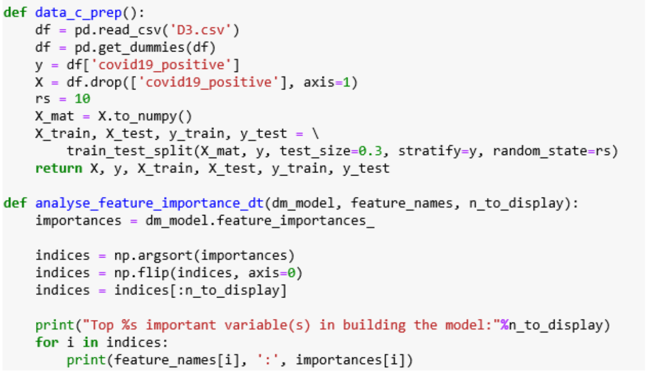
1.2 Variable Role Assignment

- Target: Covid19_Positive
- Input: the rest of input attributes (except the 'covid19_positive' attribute)

1.3 Dataset Partition (7:3)

Stratification can ensure the equal ratio of positive and negative classes in both datasets

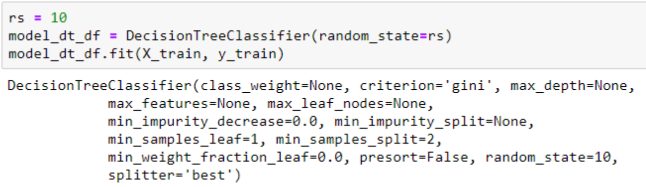
1.4 Declare Feature Importance



```
def data_c_prep():
    df = pd.read_csv('D3.csv')
    df = pd.get_dummies(df)
    y = df['covid19_positive']
    X = df.drop(['covid19_positive'], axis=1)
    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 X, y, X_train, X_test, y_train, y_test

def analyse_feature_importance_dt(dm_model, feature_names, n_to_display):
    importances = dm_model.feature_importances_

    indices = np.argsort(importances)
    indices = np.flip(indices, axis=0)
    indices = indices[:n_to_display]

    print("Top %s important variable(s) in building the model:"%n_to_display)
    for i in indices:
        print(feature_names[i], ':', importances[i])
```

1.5 DT_df: Establish Decision Tree (with default settings)

The random seed (rs = 10) ensures a set of results generated by the Decision Tree model to be consistent if it is in multiple runs.



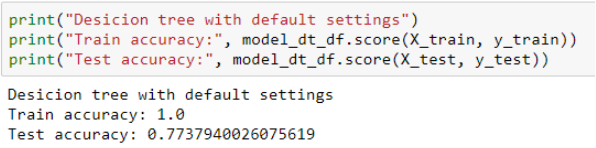
```
rs = 10
model_dt_df = DecisionTreeClassifier(random_state=rs)
model_dt_df.fit(X_train, y_train)
```

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

1.6 Decision Tree Evaluation (Accuracy, Precision, Recall and F-1 score)

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

The accuracies for training and testing datasets are 100% and approximate 77%. It is noticed that the model performed on the training set seemingly contributes to a perfect result (100% Accuracy), but the testing accuracy is significantly lower than it. In reality, the result may not be reliable, as it may indicate the occurrence of the **over-fitting** issue, where this issue implies that the model may not generalise unseen data, but it is well-performed on the training set. This will be assessed in the next stage.

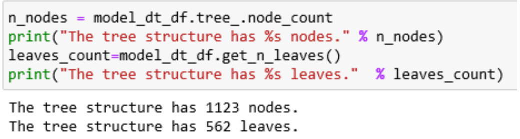


```
print("Desicion tree with default settings")
print("Train accuracy:", model_dt_df.score(X_train, y_train))
print("Test accuracy:", model_dt_df.score(X_test, y_test))
```

```
Desicion tree with default settings
Train accuracy: 1.0
Test accuracy: 0.7737940026075619
```

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

The model fits the dataset, generating 1123 nodes and 562 leaves, where each leaf is corresponding to a rule generated. As numerous nodes and leaves generated, in terms of the original model, it contributes to a quite complex tree, which is not beneficial to interpret and analyse. A **complicated** tree may indicate the potential over-fitting problem, as **noisy data** or **outliers** may be incorporated into the tree. A general-purpose solution could consider pre-pruning the tree (it is along with the tree generation), by limiting the depth of the tree, the number of leaves, or etc. We aim to configure hyperparameters to achieve the purpose, which will be elaborate in Question 2 by using **GridSearchCV** for defining suitable hyperparameters to prune an optimal tree.

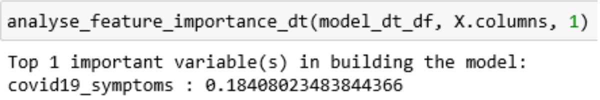


```
n_nodes = model_dt_df.tree_.node_count
print("The tree structure has %s nodes." % n_nodes)
leaves_count=model_dt_df.get_n_leaves()
print("The tree structure has %s leaves."  % leaves_count)
```

```
The tree structure has 1123 nodes.
The tree structure has 562 leaves.
```

c. Which variable is used for the first split?

The 'covid19_symtoms' is used for the first split. As for Decision Tree (default settings), Gini is the default criterion used for measuring the quality of splits. **The first split is decided by the highest number of Gini score**, where the score reflects how the importance of a feature analysed by Decision Tree in the dataset. The root node of a tree is the first split. Feature importance is either helpful to understand Decision Tree Model, or to use to get insight more information to stakeholders.

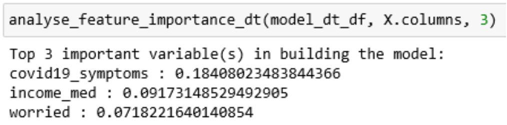


```
analyse_feature_importance_dt(model_dt_df, X.columns, 1)
```

```
Top 1 important variable(s) in building the model:
covid19_symptoms : 0.18408023483844366
```

d. What are the 3 important variables in building the tree?

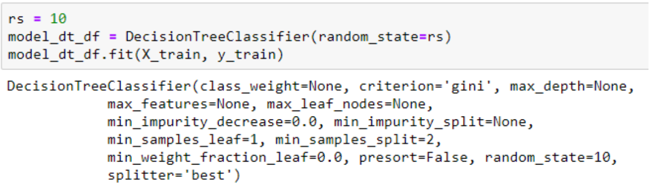
The feature ‘covid19_symtoms’ is selected as the root node for starting splitting (First Split), following by children nodes ‘income_med’ and ‘worried’, so that the top-3 important features are recognised the three nodes. Feature Importance is either useful to understand the model, or learned more from the data and getting insight into information to stakeholders.



```
analyse_feature_importance_dt(model_dt_df, X.columns, 3)

Top 3 important variable(s) in building the model:
covid19_symptoms : 0.18408023483844366
income_med : 0.09173148529492905
worried : 0.0718221640140854
```

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

In the original model, we only employ default hyperparameters to experiment. Below are all the hyperparameters with the default setting in the DecisionTreeClassifier, but we mainly focus on four hyperparameters: **criterion**, **max_depth** and **min_sample_leaf**, **random_state.**

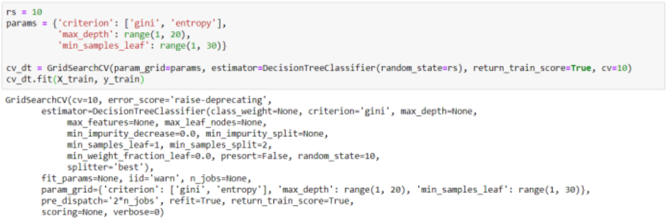


```
rs = 10
model_dt_df = DecisionTreeClassifier(random_state=rs)
model_dt_df.fit(X_train, y_train)

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

Hyperparameters	Value	Description
criterion	**'gini'**	it is applied to measure the quality of the split. There are two values: "gini" and "entropy". The “gini”, as the default value, measures the impurity while “entropy” measures the information gain to decide the split.
max_depth (not use, but explains)	**None**	max_depth represents how deep the tree will grow, as the default value **None** implies there is no limitation on the depth of the tree, that the tree will grow until all leaf nodes are pure. As a result, the model learned all the samples in training set thoroughly, i.e. **overfitting**.
min_samples_leaf	**1**	min_samples_leaf requires the minimum number of samples at a leaf node. With the default value is 1, the tree will learn every single sample to generate the tree, possibly leading to **overfitting**.
random_state	10	The random seed ensures a set of results generated to be consistent along with multiple runs.

2. Build another decision tree tuned with GridSearchCV.

2.1 Models Evaluation by GridSearchCV

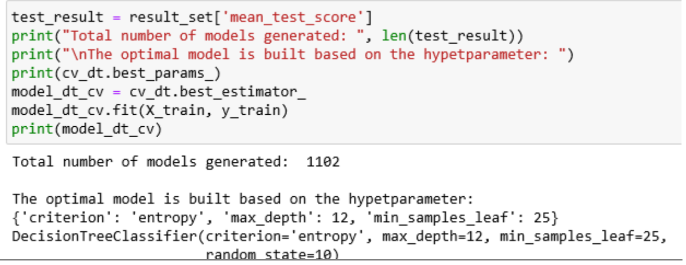


```
rs = 10
params = {'criterion': ['gini', 'entropy'],
          'max_depth': range(1, 20),
          'min_samples_leaf': range(1, 30)}

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

GridSearchCV(cv=10, error_score='raise-deprecating',
       estimator=DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,
            max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, presort=False, random_state=10,
            splitter='best'),
       fit_params=None, iid='warn', n_jobs=None,
       param_grid={'criterion': ['gini', 'entropy'], 'max_depth': range(1, 20), 'min_samples_leaf': range(1, 30)},
       pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
       scoring=None, verbose=0)
```

GridSearchCV is a third-party library enabling to perform an exhaustive search by using a bag of pre-defined hyperparameters. we only focus on discovering the optimal values for the three hyperparameters (Criterion, Max_depth, and min_samples_leaf). The drawback of GridSearchCV requires extensive computations. We set ‘max_depth’ range (1, 20), and ‘min_samp les_leaf’ (1, 30). Note that 10-Folds Cross Validation is donated to the experiment, which is beneficial to reduce the probability of overfitting issues and minimise the standard deviation.

2.2 DT_cv: Establish Decision Tree 2 (tuned by GridSearchCV)

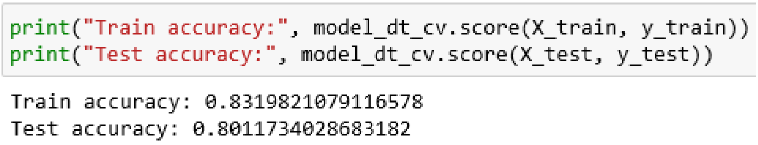
As a result, the best parameters are criterion (entropy), ‘max_depth’ (12), ‘min_samples_leaf’ (25), where the best estimator is evaluated from 1102 estimators (2*19*29) generated by GridSearchCV.



```
test_result = result_set['mean_test_score']
print("Total number of models generated: ", len(test_result))
print("\nThe optimal model is built based on the hypetparameter: ")
print(cv_dt.best_params_)
model_dt_cv = cv_dt.best_estimator_
model_dt_cv.fit(X_train, y_train)
print(model_dt_cv)

Total number of models generated:  1102

The optimal model is built based on the hypetparameter:
{'criterion': 'entropy', 'max_depth': 12, 'min_samples_leaf': 25}
DecisionTreeClassifier(criterion='entropy', max_depth=12, min_samples_leaf=25,
                       random_state=10)
```

2.3 Decision Tree 2 Evaluation (Accuracy, Precision, Recall and F-1 score)

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

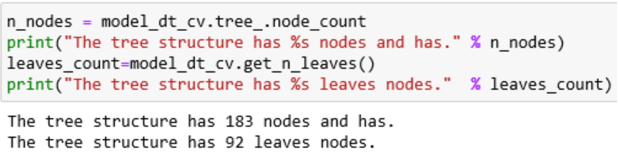


```
print("Train accuracy:", model_dt_cv.score(X_train, y_train))
print("Test accuracy:", model_dt_cv.score(X_test, y_test))

Train accuracy: 0.8319821079116578
Test accuracy: 0.8011734028683182
```

We select the best model to evaluate its accuracy in terms of training and testing results, which is approximate 0.83 and 0.80 respectively. Compared with the previous model, the testing accuracy is improved, which increases from 0.77 (model without tuning) to 0.80. Although the training accuracy decreases from 100% to 83%, it cannot be in saying that it harms model performance, which will prove in **Question 2** by virtualising entire epochs to illustrate the avoidance of overfitting phenomenon, by setting the hyperparameters.

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

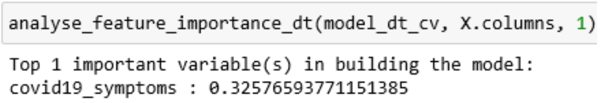
The numbers of nodes and rules are 183 and 92 respectively. In terms of the optimal Decision Tree, it contributes to relatively simple tree compared with the previous one. The *max_depth* and *min_samples_leaf* hyperparameters are helpful to prune the complicated tree, enabling it to be simple and easy to analyse and interpret. During the process, the model may efficiently remove the potential noisy data or outliers.



```
n_nodes = model_dt_cv.tree_.node_count
print("The tree structure has %s nodes and has." % n_nodes)
leaves_count=model_dt_cv.get_n_leaves()
print("The tree structure has %s leaves nodes."  % leaves_count)

The tree structure has 183 nodes and has.
The tree structure has 92 leaves nodes.
```

c. Which variable is used for the first split?

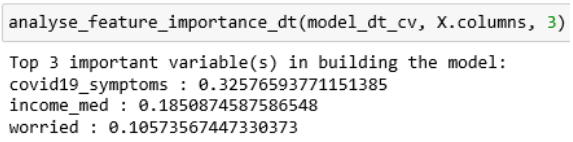
We employ a similar approach to **Question 1** for extracting the top-1 important variable, i.e. *covid19_symtoms* (around 0.33), generated by the optimal model. Note that the weight to this variable is different between the two models, because the criterion is compared by Gini and Entropy.



```
analyse_feature_importance_dt(model_dt_cv, X.columns, 1)

Top 1 important variable(s) in building the model:
covid19_symptoms : 0.325765937711513851
```

d. What are the 3 important variables in building the tree?

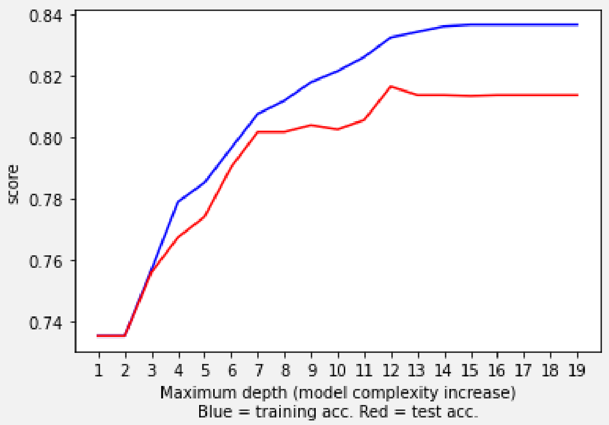
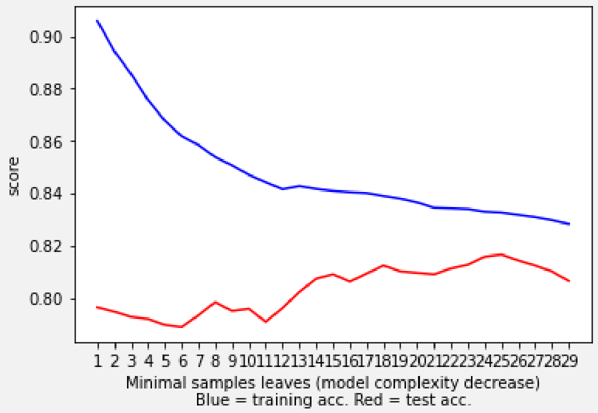
The top-3 important variables generated by the optimal model, i.e. *covid19_symtoms*, *income_med*, and *worried*. The most significant attribute 'covid19_symptoms' is selected as the first split, where the attribute indicates that the Covid-19 symptoms, is an essential indicator for identifying whether a patient is Covid-19 Positive or Negative. The 3 most important variables are the same as the definition by the default model, thus we can identify that they can be the **general characteristics** of a positive Covid-19 patient.



```
analyse_feature_importance_dt(model_dt_cv, X.columns, 3)

Top 3 important variable(s) in building the model:
covid19_symptoms : 0.325765937711513851
income_med : 0.1850874587586548
worried : 0.105735674473303731
```

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

From Question 2, the training accuracy is slightly higher than the testing accuracy, with 0.83 and 0.80 respectively. It may imply the overfitting phenomenon in the optimal model. We aim to get insight into the choice of the model, by plotting each epoch for training and testing accuracy. The below graphs present overfitting and underfitting issues.

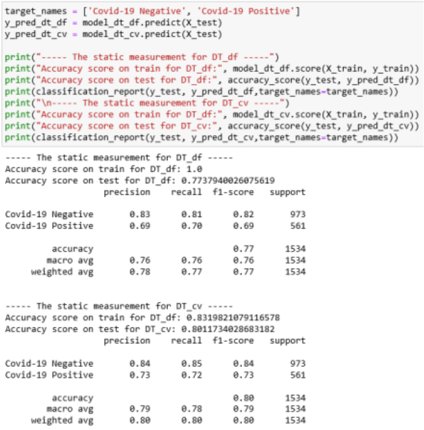
Decision Tree Depth Evaluation with Fixed Criterion (entropy) and Min_samples_leaf (25)	Decision Tree Min Samples Evaluation with Fixed Criterion (entropy) and Max_depth (12)
	
The graph is plotted for exploring the model performance when it is fed to various values [1 to 19] of Max_depth, Criterion and Min_samples_leaf is fixed, with 'entropy' and 25 respectively. Initially, the depth of the tree is small, which	The graph is plotted when [Criterion = 'entropy'] and [Max_Depth = 12], the model is fed with different Min_samples_leaf values [1, 29]. Min_samples_leaf describes the minimal number of samples that a leaf can have. Initially, when Min_samples_leaf=1, the train

indicates the high bias (a simple model) and low training and testing accuracy. As the increase of the size of tree depth, the model is ever-increasing complicated, as well as the variance, gets increase and bias gets decrease. In Max_Depth = 7, we can see the testing accuracy experiences a slight decrease, where the gap between training and the testing accuracy is gradually widened. Until the Max_Depth = 12, the ratio of testing accuracy reaches its peak point, but it experiences a slight decrease and then keeps stable. In the meantime, the training accuracy is more sable as well. When Max_Depth = 12, the overfitting occurs and leads the testing accuracy to overcome. Hence, [Max_Depth = 12] is the best choice for avoiding overfitting.	fits the training set very well, which indicates high variance and low bias. As this increase, model variability increases, which reflects on the increase of testing accuracy. At a certain point (25), the Min_samples_leaf gets increase too large, which leads the tree underfits (training accuracy decrease in this point). It indicates that the model fails to learn the important pattern in the data. This is a sign of underfitting. Thus, we can conclude [Min_samples_leaf = 25] is the best option for avoiding underfitting

3. What is the difference do you see 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.

- **Models Comparison with Evaluation Metrics (Precision, Recall, and F1)**

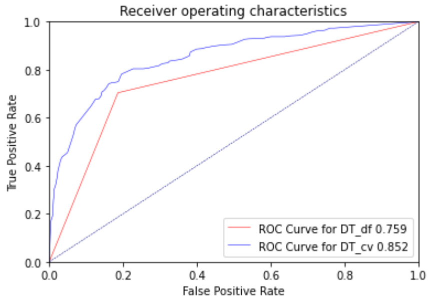
By the comparison of two DT models (with and without fine-tuning), the testing accuracy to the optimal model is more than that of the original model, with just around **80%** and **77**% respectively. In terms of Precision (breaking down to Positive or Negative classes), Recall, and F1-score, the performance of the optimal model is better than the default model. Intuitively, a general-purpose evaluation to a model performance is reliant on the measurements, i.e. Accuracy, Precision, Recall or F1-score. The simple approach enables to explore how a model could be well-performed on a dataset. However, the method may be efficient to a dataset that is with the balanced ratio of target classes, e.g. 50% Covid-19 Positive or Covid-19 Negative. In terms of the current dataset, the class is unbalanced, which is proved different Support in terms of two classes (Covid-19 Positive and Negative), with 973 and 561 respectively. Although we have employed GridSearchCV to discover three suitable hyperparameters, it is only defined by testing accuracy, which is not compelling to convince that the model is adjusted to an ideal status that can well fit the current dataset.



```
target_names = ['Covid-19 Negative', 'Covid-19 Positive']
y_pred_dt_df = model_dt_df.predict(X_test)
y_pred_dt_cv = model_dt_cv.predict(X_test)

print("----- The static measurement for DT_df -----")
print("Accuracy score on train for DT_df:", model_dt_df.score(X_train, y_train))
print("Accuracy score on test for DT_df:", accuracy_score(y_test, y_pred_dt_df))
print(classification_report(y_test, y_pred_dt_df,target_names=target_names))
print("\n----- The static measurement for DT_cv -----")
print("Accuracy score on train for DT_df:", model_dt_cv.score(X_train, y_train))
print("Accuracy score on test for DT_cv:", accuracy_score(y_test, y_pred_dt_cv))
print(classification_report(y_test, y_pred_dt_cv,target_names=target_names))
```

```
----- The static measurement for DT_df -----
Accuracy score on train for DT_df: 1.0
Accuracy score on test for DT_df: 0.7737940026075619
                   precision    recall  f1-score   support

Covid-19 Negative       0.83      0.81      0.82       973
Covid-19 Positive       0.69      0.70      0.69       561

         accuracy                           0.77      1534
        macro avg       0.76      0.76      0.76      1534
     weighted avg       0.78      0.77      0.77      1534


----- The static measurement for DT_cv -----
Accuracy score on train for DT_df: 0.8319821079116578
Accuracy score on test for DT_cv: 0.8011734028683182
                   precision    recall  f1-score   support

Covid-19 Negative       0.84      0.85      0.84       973
Covid-19 Positive       0.73      0.72      0.73       561

         accuracy                           0.80      1534
        macro avg       0.79      0.78      0.79      1534
     weighted avg       0.80      0.80      0.80      1534
```

- **Models Comparison with ROC Curse**



For the fine-grained investigation, we propose a wisdom method for comparing the two models (with and without fine-tuning). ROC is an abbreviation 'Receiver Operating Characteristic', which is calculated with AUC (Area Under Curve). ROC Curves is used for visualising the comparison of classification models, while AUC is a measure used for assessing the accuracy of a model. In a ROC curve, the x-axis represents **False Positive Rate** (FPR = FP / (FP + TN)), which represents the radio that actual negative observations are predicted incorrectly (Predict to True, but the actual class is False). The y-axis is **Ture Positive Rate** *(TPR = TP / (TP + FN)),* also called Recall, which represents the ratio that actual positive observations are predicted correctly (Predict to True, but the actual class is True). The AUC score is the area under the curve. The higher the AUC score has, the better performance the model has. If a model can predict perfectly, the AUC score is 1, but 0.5 indicates the worst model (TPR = FRP).

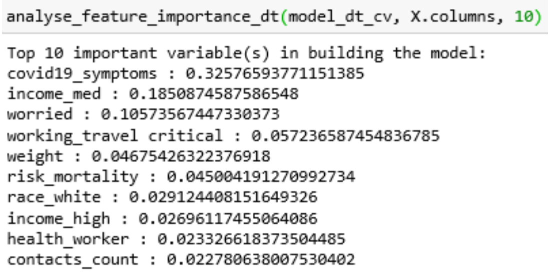
- **DT_df**: baseline model with **criterion = 'gini', max_depth = None, min_samples_leaf = 1**
- **DT_cv**: model tuned by GridSearchCV with **criterion = entropy, max_depth = 12, min_samples_leaf = 25**

The AUC score of DT_cv is greater than the DT_df, with 0.852 and 0.759 respectively, which indicates that DT_cv model is better than DT_df in terms of performance. The reasons for this change can be summarised in a comparative table.

DT_df	DT_cv
The model is suffered from **overfitting issues**, as the training accuracy = 100% and testing accuracy = 77%, which indicates that its prediction is unreliable.	There is **no overfitting** existed, it is with desirable accuracies for training and testing (83% and 80% respectively).
The tree is complicated, as there is no limitation to construct the tree, which may be impurity (contains noise and outliers).	The tree is simply pruned by a predictable depth (12) and limited samples in a leaf (25), which is helpful to remove potential noise and outliers, curbing the tree to arbitrarily expand.

4. From the better model, can you identify which patients could potentially be "COVID-19 Positive"? Can you provide general characteristics of those patients?

DT_cv is defined as the best model. Feature importance is useful to offer important features. The model is useful to generate the order of important features, but it may not be efficient to enable us to get insight into the data. Here, we aim to identify the general characteristics of Covid-19 patients, by combining the Feature Importance and Decision Tree (see in the Appendix).



```
analyse_feature_importance_dt(model_dt_cv, X.columns, 10)
```

```
Top 10 important variable(s) in building the model:
covid19_symptoms : 0.32576593771151385
income_med : 0.1850874587586548
worried : 0.10573567447330373
working_travel critical : 0.057236587454836785
weight : 0.04675426322376918
risk_mortality : 0.045004191270992734
race_white : 0.029124408151649326
income_high : 0.02696117455064086
health_worker : 0.023326618373504485
contacts_count : 0.022780638007530402
```

Covid19_symptoms are the most important feature, which is also the first split variable in the decision tree. From the decision tree (in Appendix), when covid19_symptoms<=0.5 is False (which indicates covid19_symptoms are positive), all the nodes from the second depth belong to positive classes. There are 388 (in 414 samples) covid-19 patients having covid-19 symptoms out of 1309 positive patients. In other words, if a patient has covid-19 symptoms, he/she is very likely to be a covid-19 patient. When covid19_symptoms<=0.5 is True (covid19_symptoms is negative), we find the next big partition of positive classes in the fifth depth, with 206 covid-19 patients (out of 269 samples), follow the routine of income_med<=0.5 is False, worried<=3.5 is False, working_travel_critical<=0.5 is False, which is corresponding with important features. Thus, we could say, if there is no covid-19 symptom for the patient, but she/he has the following characteristics: income_med, worried>3.5, and working_travel_critical, she/he is likely to be a potential covid-19 patient. There are many other positive classes in the decision tree. However, the routines cannot be claimed as general characteristics. For example, there is a node class in ninth depth with entropy=0.917, which indicates the high difference among the class. In summary, covid19_symptoms are a general character. Meanwhile, income_med, worried>3.5 and working _travel_critical are also the general characteristics when the patient doesn't from the covid19_symptoms.

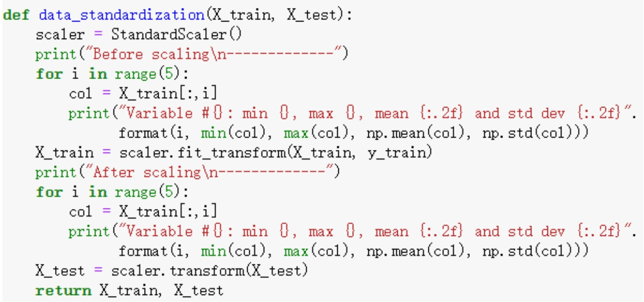
Predictive modelling using Regression

Before we get started the establishment of a model for prediction, we need to clarify the current mining task that pertains to the classification problem. The experimental purpose is to employ historical observations to train the model for enabling it to generalise unseen data. In other words, the model should equip the capability to extract general characteristics to identify whether a patient is Covid-19 Positive or Negative. In reality, we need to employ a regression algorithm to solve the classification matter. The attribute 'Covid19_Positive' is required to be the label variable (target), which is a categorical type (Binary).

In terms of several regression algorithms, Logistic Regression (LR) is a suitable candidate to achieve our objective. **The algorithm usually predicts whether something is True or False (Binary/Boolean), instead of predicting continuous data.** As for the pre-defined label (Covid19_Positive), it only contains 2 numerical values, where the value 1 indicates that a patient is Covid-19 Positive, while 0 implies a patient who is Covid-19 Negative. LR can be assistive to analyse the general characteristics of a patient for predicting whether a new patient is Positive or Negative. LR is reliant on calculating the probability of a particular level of a class variable, based on a set of input attributes, to classify instances.

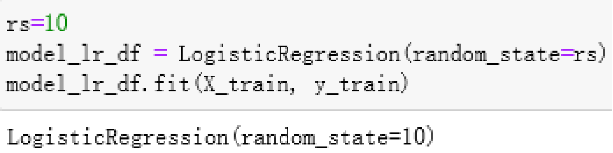
1. Build a regression model using the default regression method with all inputs. Build another regression model tuned with GridSearchCV.

1. Data Pre-process (*refer to decision tree: 1.1 – 1.3*)
2. Declare Functions (*refer to decision tree: 1.4*)
3. **Perform** Standardization



```
def data_standardization(X_train, X_test):
    scaler = StandardScaler()
    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)))
    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)
    return X_train, X_test
```

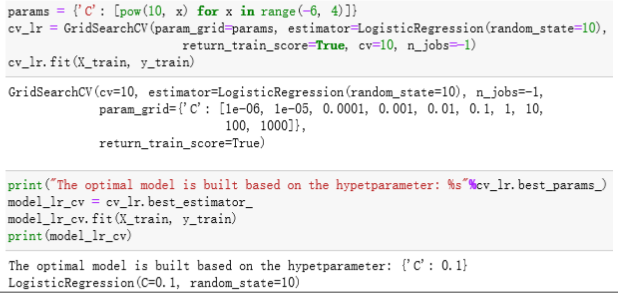
4. **LR1: Establish Logistic Regression Model (with default settings) and Train**
   <u>C = 1 & Random State = 10</u>



```
rs=10
model_lr_df = LogisticRegression(random_state=rs)
model_lr_df.fit(X_train, y_train)
```

```
LogisticRegression(random_state=10)
```

5. **LR2: Establish Regression Model Tuning by GridSearchCV and Train**
   <u>C = 0.1 & Random State = 10</u>



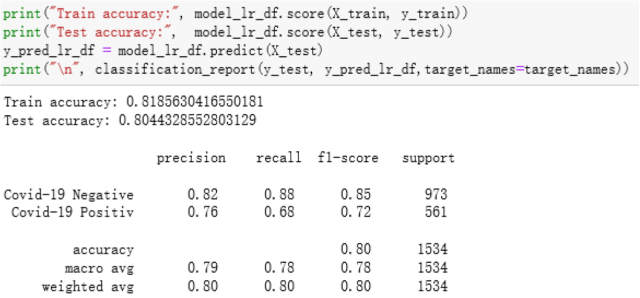
```
params = {'C': [pow(10, x) for x in range(-6, 4)]}
cv_lr = GridSearchCV(param_grid=params, estimator=LogisticRegression(random_state=10),
                     return_train_score=True, cv=10, n_jobs=-1)
cv_lr.fit(X_train, y_train)
```

```
GridSearchCV(cv=10, estimator=LogisticRegression(random_state=10), n_jobs=-1,
             param_grid={'C': [1e-06, 1e-05, 0.0001, 0.001, 0.01, 0.1, 1, 10,
                               100, 1000]},
             return_train_score=True)
```

```
print("The optimal model is built based on the hypetparameter: %s"%cv_lr.best_params_)
model_lr_cv = cv_lr.best_estimator_
model_lr_cv.fit(X_train, y_train)
print(model_lr_cv)
```

```
The optimal model is built based on the hypetparameter: {'C': 0.1}
LogisticRegression(C=0.1, random_state=10)
```

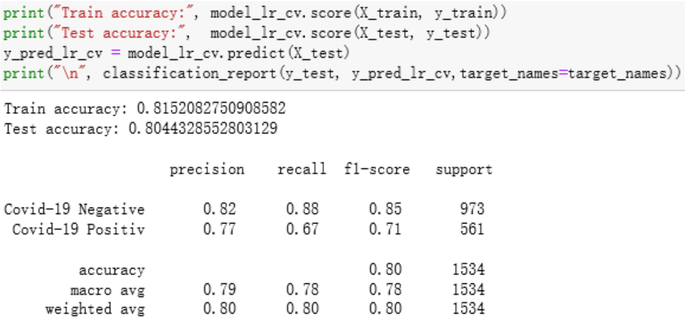
6. LR1 Evaluation



```
print("Train accuracy:", model_lr_df.score(X_train, y_train))
print("Test accuracy:",  model_lr_df.score(X_test, y_test))
y_pred_lr_df = model_lr_df.predict(X_test)
print("\n", classification_report(y_test, y_pred_lr_df, target_names=target_names))
```

```
Train accuracy: 0.8185630416550181
Test accuracy: 0.8044328552803129

                   precision    recall  f1-score   support

Covid-19 Negative       0.82      0.88      0.85       973
 Covid-19 Positiv       0.76      0.68      0.72       561

         accuracy                           0.80      1534
        macro avg       0.79      0.78      0.78      1534
     weighted avg       0.80      0.80      0.80      1534
```

7. LR2 Evaluation



```
print("Train accuracy:", model_lr_cv.score(X_train, y_train))
print("Test accuracy:",  model_lr_cv.score(X_test, y_test))
y_pred_lr_cv = model_lr_cv.predict(X_test)
print("\n", classification_report(y_test, y_pred_lr_cv, target_names=target_names))
```

```
Train accuracy: 0.8152082750908582
Test accuracy: 0.8044328552803129

                   precision    recall  f1-score   support

Covid-19 Negative       0.82      0.88      0.85       973
 Covid-19 Positiv       0.77      0.67      0.71       561

         accuracy                           0.80      1534
        macro avg       0.79      0.78      0.78      1534
     weighted avg       0.80      0.80      0.80      1534
```

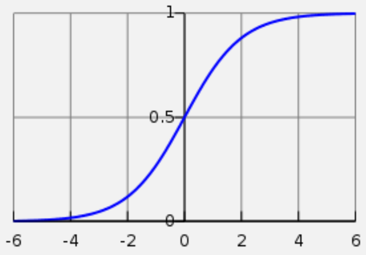
Now, choose a better model to answer the followings: a. Explain why you chose that model.

We choose the optimal model tuned by GridSearchCV, with C = 0.1 (it contributes to the inverse of the regularization strength, being beneficial to reduce the probability of occurring the overfitting issue). The method is capable of extracting the optimal model based on the given testing hyperparameters. Here, we only donate a bag of C values to discover an optimal C that enables the model to be well-performed than the previous one. The evaluative results are the convictive compelling evidence to prove our choice.

Accuracy	Precision, Recall and F1-Score to Covid-19 Positive
Baseline model (default setting with C = 1) contributes to 0.82 and 0.80 to training and testing accuracy. As the training accuracy is slightly more than testing accuracy, it may imply overfitting, which will be examined in Task g. On the contrary, the optimal model achieves 0.81 testing accuracy, although the training accuracy is slightly lower than the baseline model. The optimal model can effectively handle unseen data than the baseline model.	We are desirable to predict the patients who are Covid-19 Positive. Obviously, the baseline model is less effective than the optimal one, because the Precision to predict Covid-19 Positive is slightly less than the optimal mode, with 0.76 and 0.77 respectively. Similarly, the numbers of Recall and F1-Score to baseline model are slightly less than the optimal one. Hence, we employ C = 0.1 to feed our Logistic Regression Model for further experiment.

b. Name the regression function used.

The algorithm taps logistic function (Sigmoid Function) to calculate the probability of each instance for categorisation. Sigmoid is commonly used for logistic regression.

SIGMOID FUNCTION	FUNCTION GRAPH
$p(X) = \frac{1}{1 + \exp(-X\beta)}$	

The formula enables the Logistic Regression Model to map any input value on the line (-Xβ) to a probability into the range [0, 1]. As the Function Graph shows, the threshold is 0.5, where if the probability of an input value is more than 0.5, it is a Positive instance, while if its probability is less than 0.5, it can be classified to a Negative group.

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

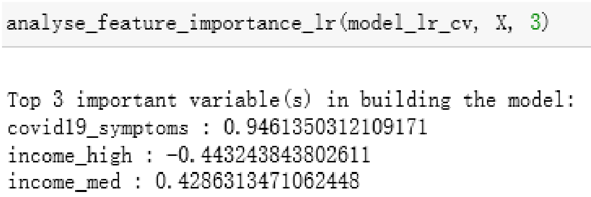
In terms of Logistic Regression Model, it is sensitive to input attributes with significantly different scales, where if the variables are measured at different scales, it may contribute **unequally to the analysis**, creating a **bias**. Although D3.csv dataset is relatively clean and pre-processed, its numerical variables reflect an obviously **different scale**, e.g. **Weight (44 - 180) and contacts_count(0 - 21),** which may cause the difficulty of comparison between data points. For example, 10kg weight increase to a person may be less important, while increase 10 unit increase in contacts_count is significantly crucial. If the patient has contacted 10 more people, the influence will be significant then a patient is 10kg weight increase. Also, this kind of scale difference may contribute an adverse impact on **gradient descent** (an optimisation algorithm embedded into Logistic Regression), resulting in suboptimal performance. Gradient descent is used for minimising the cost function for a regression algorithm, which will iteratively update the weights for this purpose. A substantial different scale to features (input variables) contributes to **extra time expenses on updating weights**. In other words, the weight update is efficient when input features are standard scaled to a small range. Thus, all input variables (except the label attribute) are required to standardise.

d. Report on which variables are included in the regression model.

In terms of the requirement mentioned, all inputs of the dataset are needed to be fed to train and test Logistic Regression Model, so that we aim to use all attributes (except the label attribute), i.e. *region, country, sex, age, height, weight, blood_type, insurance, income, race, immigrant, smoking, contacts_count, house_count, public_transport_count, working, worried, covid19_symptoms, covid19_contact, asthma, kidney_disease, liver_disease, compromised_immune, heart_disease, lung_disease, diabetes, hiv_positive, hypertension, other_chronic, nursing_home, health_worker, and risk_mortality*, to be training or testing set. The attribute '*covid19_positive*' will be selected as the target variable. The features may contain noise ones, which will be filtered in the next experiment. Note that the dataset **originally** contains categorical and numeric data types. To feed the model properly, we applied one-hot encoding, converting inputting attributes to one-hot representation for clustering.

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

The top-3 important variables are *covid19_symptoms*, *income_high* and *income_med*. The process is calculating coefficients and ordered by their absolute values about the corresponding variable. The logistic regression model assumes the linear correlation between each input variable and the target variable. This relationship can be represented by its coefficient, which indicates the weight of the corresponding input variable. There are two ways for coefficients to impact the target variable. The absolute value of the coefficient indicates the importance of the input attribute, which means the larger the absolute value, the more influence contributed by this attribute to the target variable. Meanwhile, the coefficient can be positive or negative. A positive coefficient indicates that the variable has a positive influence on the target variable, which will gain the



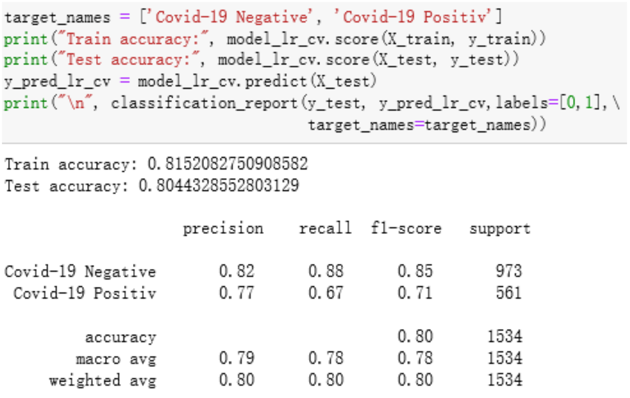
```
analyse_feature_importance_lr(model_lr_cv, X, 3)
```

```
Top 3 important variable(s) in building the model:
covid19_symptoms : 0.9461350312109171
income_high : -0.443243843802611
income_med : 0.4286313471062448
```

possibility of the target variable. On the contrary, a negative coefficient indicates it is less likely to predict the target variable. In this logistic regression tuned by GridSearchCV, the most crucial variable is *covid19_symptoms*, with a weight of 0.71, which implies people with *covid19_symptoms* are predicted very likely to be COVID-19 patients. The second important variable is *income_high*, with the weight of -0.37, which indicates that people with high income are predicted more likely not to be COVID-19 patients. The third important variable is income_med, with a weight of 0.36, which means people with medium income are predicted more likely to be COVID-19 patients.

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

The accuracy is around 0.815 for training and 0.804 for testing datasets. However, the accuracy may not convince that the model is well-performed on classification for predicting patients with Covid-19 Positive. When we analyse the confusion matrix about COVID-19 Positive class, we have found [Recall=67%] and [Precision=77%]. In the experiment, we are expected to be sensitive Covid-19 patients, but the 67% of Recall indicates that only 387 (Support 561 * 0.69) patients are fetched out, which means a number of patients (174) are not able to detect. Here, the Recall measure indicates the proportion of all real positive observations that have been predicted correctly. 77% of Precision reflects the model performance in predicting Covid-19 patients, which is less than the prediction of Covid-19 Negative cases. In summary, even the model is improved by the comparison to the previous model (with default setting), it may not be suitable for the prediction of Covid-19 Positive patients.



```
target_names = ['Covid-19 Negative', 'Covid-19 Positiv']
print("Train accuracy:", model_lr_cv.score(X_train, y_train))
print("Test accuracy:", model_lr_cv.score(X_test, y_test))
y_pred_lr_cv = model_lr_cv.predict(X_test)
print("\n", classification_report(y_test, y_pred_lr_cv,labels=[0,1],\
                                    target_names=target_names))
```

```
Train accuracy: 0.8152082750908582
Test accuracy: 0.8044328552803129

                   precision    recall  f1-score   support

Covid-19 Negative       0.82      0.88      0.85       973
 Covid-19 Positiv       0.77      0.67      0.71       561

         accuracy                           0.80      1534
        macro avg       0.79      0.78      0.78      1534
     weighted avg       0.80      0.80      0.80      1534
```

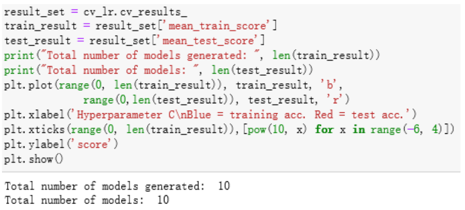
g. Report any sign of overfitting in this model.

Overfitting Definition

The overfitting is a common phenomenon in machine learning, that occurs when a model corresponds a set of training data too tightly and exactly, which may lead that the model fails to generalise or predict new instances reliably. In reality, data with some degree of noise and errors are not evitable. An overfitted model could tap unknown residual noisy data to be the model structure for prediction, which may adversely affect its predictive capability. A common judgement is that the accuracy of a model experiences an upward trend, but the testing accuracy gets decrease at a point. After the point, the testing accuracy cannot be improved or roll back to its peak point. We may say the model is overfitted after the training point.

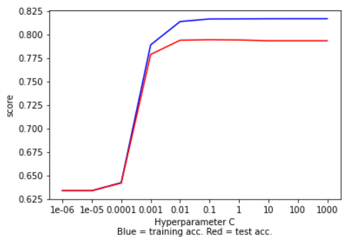
L2 Regularization

As for the optimal logistic regression model, we only focus on one hyperparameter (C), where C donates the inverse of regularization strength. A small value for **C** contributes to more robust regularization. L2 Regularization is efficient to prevent generating a large and complex model, reducing the possibility of overfitting phenomenon. We aim to plot a set results of GridSearchCV, thereby identifying the sign of overfitting phenomenon.



```
result_set = cv_lr.cv_results_
train_result = result_set['mean_train_score']
test_result = result_set['mean_test_score']
print("Total number of models generated: ", len(train_result))
print("Total number of models: ", len(test_result))
plt.plot(range(0, len(train_result)), train_result, 'b',
            range(0,len(test_result)), test_result, 'r')
plt.xlabel('Hyperparameter C\nBlue = training acc. Red = test acc.')
plt.xticks(range(0, len(train_result)),[pow(10, x) for x in range(-6, 4)])
plt.ylabel('score')
plt.show()
```

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



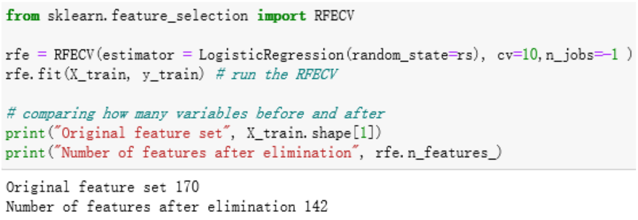
The graph is plotted for exploring the model performance when it is fed to a bag of C values **[**10^{-6}, 10^{-5}, 10^{-4}, … 10^{3}**]**. With the increase of C, accuracy scores for training and testing datasets increase to 0.01. After this point, with the increase of C, the training accuracy is growing gently, while the testing accuracy gets started to decrease. According to overfitting definition, it indicates that the overfitting occurred after this point. Hence, [C=0.1] is optimal hyperparameter to avoid overfitting. However, it is noticed that when C is set to 1, the model (with default setting) is overfitted, which proves that its results (training and testing accuracy) may not be reliable. In this stage, it proves that our choice to the optimal model (C = 0.1) is sensible to perform classification on the dataset.

2. 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.

Here, we will introduce an efficient approach to improve model performance – Feature Selection & Extraction. Feature Selection is a the-state-of-the-art method used for selecting a set of relevant features from the original datasets, based on the feature importance (feature weights). At the same time, Feature Extraction aims to transform the high-dimensional feature space into a lower data dimensionality. Both combinative methods are useful to improve efficiency (e.g. reduce time expense on classification), as only the principal components are fed to train and test. The filtering process is reliant on the number of feature weight, where a threshold is set for filtering the features that their weights are less than this threshold. All features are required to be traversed until the optimal features are selected and extracted. The threshold is usually reliant on experience to be configured. In this experiment, we aim to discover how the approach could affect model performance in terms of Accuracy, Precision, Recall and other evaluative measures.

1. Feature Selection

We employ REF with 10-Folds Cross-Validation to perform Feature Selection. The model is tapped by the original model (with default configurations). 28 (170-142) features are eliminated by the below code, and we have retained 142 features.

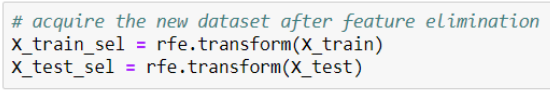


```
from sklearn.feature_selection import RFECV

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 170
Number of features after elimination 142
```

2. Establish New Dataset

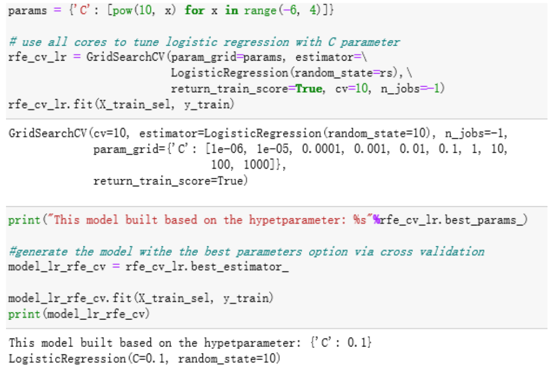
Perform feature extraction to create a new dataset with lower dimensionality.



```
# acquire the new dataset after feature elimination
X_train_sel = rfe.transform(X_train)
X_test_sel = rfe.transform(X_test)
```

3. LR3: Establish Logistic Regression Model combined with Feature Selection and GridSearchCV

We employed GridSearchCV to fine-tuning the C for a better logistic regression model, where the experiment donates C = 0.1 to the optimal model.



```
params = {'C': [pow(10, x) for x in range(-6, 4)]}

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

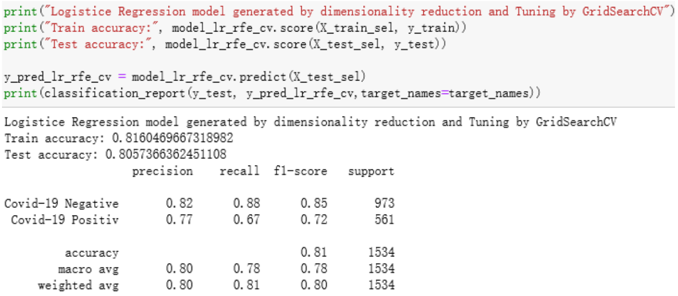
```
GridSearchCV(cv=10, estimator=LogisticRegression(random_state=10), n_jobs=-1,
             param_grid={'C': [1e-06, 1e-05, 0.0001, 0.001, 0.01, 0.1, 1, 10,
                               100, 1000]},
             return_train_score=True)
```

```
print("This model built based on the hypetparameter: %s"%rfe_cv_lr.best_params_)

#generate the model withe the best parameters option via cross validation
model_lr_rfe_cv = rfe_cv_lr.best_estimator_

model_lr_rfe_cv.fit(X_train_sel, y_train)
print(model_lr_rfe_cv)
```

```
This model built based on the hypetparameter: {'C': 0.1}
LogisticRegression(C=0.1, random_state=10)
```

a. Was dimensionality reduction useful to identify a good feature set for building the accurate model?
b. What is classification accuracy on training and test datasets?

We donate a table to summarise accuracies to two optimal models, i.e. LR1 (tuned by GridSearchCV with the original dataset), and LR2 (tuned by GridSearchCV with the dataset filtered out irrelevant features). Here, we only focus on accuracy, and Precision (positive), Recall (positive), and F1-score (Positive).



```
print("Logistice Regression model generated by dimensionality reduction and Tuning by GridSearchCV")
print("Train accuracy:", model_lr_rfe_cv.score(X_train_sel, y_train))
print("Test accuracy:", model_lr_rfe_cv.score(X_test_sel, y_test))

y_pred_lr_rfe_cv = model_lr_rfe_cv.predict(X_test_sel)
print(classification_report(y_test, y_pred_lr_rfe_cv,target_names=target_names))
```

```
Logistice Regression model generated by dimensionality reduction and Tuning by GridSearchCV
Train accuracy: 0.8160469667318982
Test accuracy: 0.8057366362451108
                   precision    recall  f1-score   support

Covid-19 Negative       0.82      0.88      0.85       973
 Covid-19 Positiv       0.77      0.67      0.72       561

         accuracy                           0.81      1534
        macro avg       0.80      0.78      0.78      1534
     weighted avg       0.80      0.81      0.80      1534
```

	LR1	LR2
Training Accuracy	0.815	0.816
Testing Accuracy	0.804	0.806
Precision (Positive)	0.77	0.77
Recall (Positive)	0.67	0.67
F1-Score (Positive)	0.71	0.72

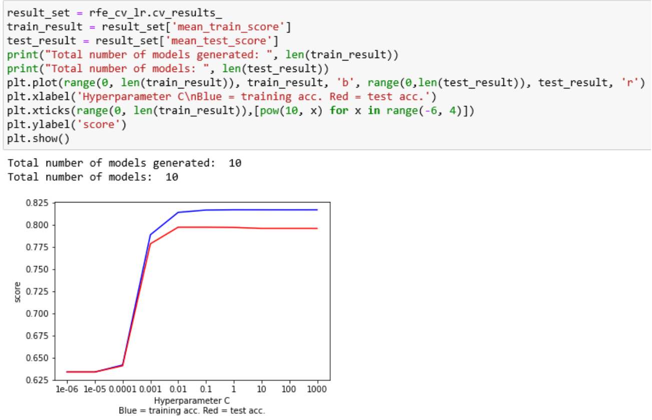
Obviously, although both models contribute to almost the same training accuracy, the testing accuracy of LR2 is slightly more than that of LR1. It proves that the Feature Selection helps to improve model performance slightly. Meanwhile, the advantages of Feature Selection can be summarised:

- Avoid the curse of dimensionality
- Improve computations as the required time and memory are reduced (efficiency)
- Be visualised easily on account of fewer attributes provided
- Eliminate irrelevant features or remove noise

In the real world, balancing efficiency and effectiveness is a trade-off. Although performance to LR2 is only slightly better than LR1, we will consider use LR2 to a real data mining task, because the LR2's results are close to LR1, i.e. Precision (0.77 vs 0.77), Recall (0.67 vs 0.67), and F1-Score (0.72 vs 0.71). Note that F1-score is the harmonic mean of the Precision and Recall. The time expense is far less than LR1. In summary, we say that a set of good features are beneficial to build a perfect model slightly than the previous one, it is also useful to reduce the dimensionality of data and improve efficiency, which is widely used data mining tasks.

c. Report any sign of overfitting.

The accuracy for testing dataset is slightly lower than training data (0.816 vs 0.806), which may indicate overfitting. We aim to get insight into the model via the plot of the GridSearchCV results. The graph experiences the same trend as the previous model (LR1). With the increase of C, the accuracies for testing and training datasets increase until the certain point [C=0.1], the testing accuracy drops slightly after this point. Note that, the accuracy for training dataset keeps slight increase. We can say when C is more than 0.1 (exclusive), the overfitting phenomenon happens. In summary, [C=0.1] is the optimal choice to generate the model, avoiding overfitting.

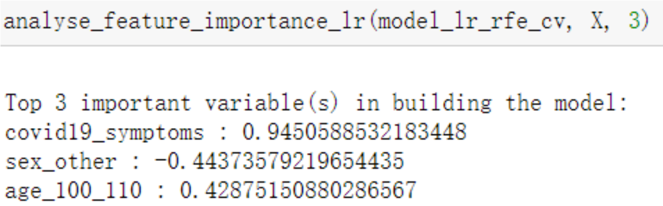


```
result_set = rfe_cv_lr.cv_results_
train_result = result_set['mean_train_score']
test_result = result_set['mean_test_score']
print("Total number of models generated: ", len(train_result))
print("Total number of models: ", len(test_result))
plt.plot(range(0, len(train_result)), train_result, 'b', range(0,len(test_result)), test_result, 'r')
plt.xlabel('Hyperparameter C\nBlue = training acc. Red = test acc.')
plt.xticks(range(0, len(train_result)),[pow(10, x) for x in range(-6, 4)])
plt.ylabel('score')
plt.show()
```

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

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

The top-3 essential variables in the model are *covid19_symptoms, sex_other* and *age_100_110*. The most important variable (*covid19_symptoms*) is the same as the previous model. However, sex_other and age_100_110 are a substitute for income_high and income_med. As for the LR2 model, the most critical variable is *covid19_symptoms*, with a weight of 0.95, which implies that people with covid19_symptoms are likely to be predicted as COVID-19 patients. The second important variable is sex_other, with the weight of -0.44, which indicates other genders may not be predicted as COVID-19 patients. The third important variable is age_100_110, with a weight of 0.43, which means people with age from 100 to 110 are likely to be predicted as COVID-19 patients. In this stage, we have two lists of feature importance generated by LR1 and LR2. The variable '*covid19_symptoms*' is most important to distinguish whether the patients are Covid-19 or not, which is a general characteristic of a patient.



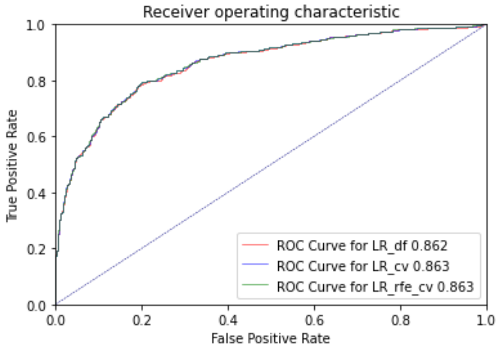
```
analyse_feature_importance_lr(model_lr_rfe_cv, X, 3)
```

```
Top 3 important variable(s) in building the model:
covid19_symptoms : 0.9450588532183448
sex_other : -0.44373579219654435
age_100_110 : 0.42875150880286567
```

3. Produce the ROC curve for all different regression models. Using the best regression model, can you identify which patients could potentially be "COVID-19 Positive"? Can you provide general characteristics of those patients?

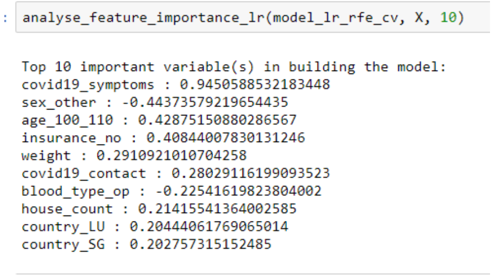
The detailed description to ROC AUC can refer to Decision Tree Question 3.

- **LR_df**: baseline model with default setting (C=1)
- **LR_cv**: the model tuned by GridSearchCV with the original dataset (C=0.1)
- **LR_rfe_cv**: the model tuned by GridSearchCV with the dataset filtered out irrelevant features (C=0.1)



The general-purpose to ROC AUC is that when the score is close to 1, the model performance is satisfactory to perform data mining on the dataset. It is noticed that LR_cv and LR_rfe_cv hold the same AUC score, accounting for 0.863. Compared with the two models, LR_df is the worst one, but it is slightly less than 0.863. Based on the previous analysis, Feature Selection is beneficial to improve efficiency, reducing the dimensionality of data, so that we identify LR_rfe_cv is the best model, that is performed on the sub-dataset with the elimination of irrelevant features from the original dataset (D3.csv).

Based on the model, we tap the Feature Importance List, generated by the best model and dataset, to extract the top-10 features, thereby identifying the general characteristics of a potential Covid-19 Positive patient. From the explanation in Question 1, the coefficient with positive value affects the target variable positive, and verse vice. Covid19_symptoms, age_100_110, insurance_no, covid19_contact, country_LU and country_SG can positively affect the target variable. In other words, when a person has covid-19 symptoms, within the age from 100 to 110, has no insurance, has contacted with covid-19 patients, from the country LU or from the country SE, can be summarized as the general characteristics. Sex_other and blood_type_on indicate less likely of being the potential patient, which means people with other gender or with the blood type of on are less to be potential patients. Other variables like house_contact and weight affect the target variables positive. However, we cannot tell how these variables affect the target variable, which means we cannot generate clear general characteristics from them. The general characteristics can be summarized as covid-19 symptoms, age from 100 to 110, without insurance, has contacted with covid-19 patients, from country LU or from the country SG.

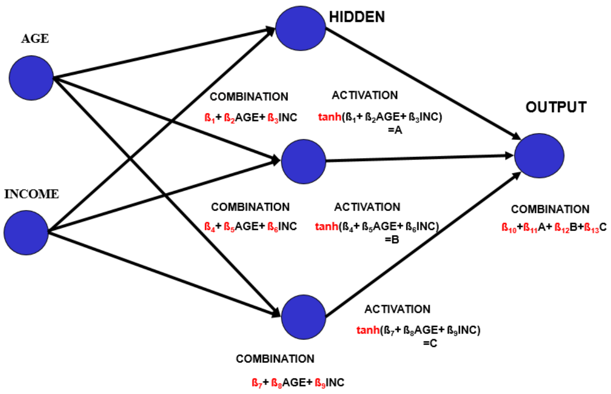


```
analyse_feature_importance_lr(model_lr_rfe_cv, X, 10)
```

```
Top 10 important variable(s) in building the model:
covid19_symptoms : 0.9450588532183448
sex_other : -0.44373579219654435
age_100_110 : 0.42875150880286567
insurance_no : 0.40844007830131246
weight : 0.2910921010704258
covid19_contact : 0.28029116199093523
blood_type_op : -0.22541619823804002
house_count : 0.21415541364002585
country_LU : 0.20444061769065014
country_SG : 0.202757315152485
```

Predictive modelling using Neural Networks

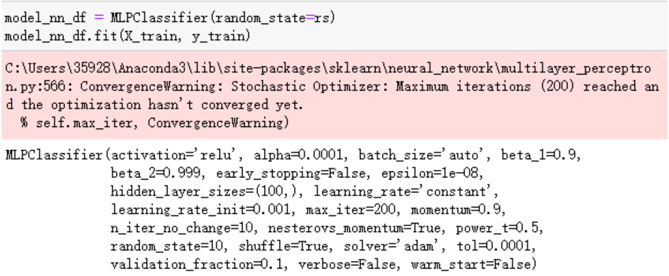
Neural Network is a natural extension of the regression model. A Neural Network model is in paralleling to process input data to prediction. The data is presented in Input Layer, where the input data will pass to Hidden Layer randomly, and then combine each other to calculate various weights. The weigh calculation is reliant on Activation Function (e.g. ReLu or Sigmoid), where the function will generate a predictive result used for comparing with the desired result (the value of target attribute). The difference between both results is called Error (Cost). In general, a desirable model is expected to generate a lower error and higher accuracy. Hence, it is noticed that the weight combination is crucial to affect model performance. In reality, seeking for an optimal combination of weights can be a substantial searching problem. In our experiment, we aim to adopt solver in sklearn library to solve the concern. Gradient Descent is the most common method pertained one of the solver algorithms. The rationale of this method initially starts with a bag of weights generated randomly. As for each iteration (Epoch), the algorithm will adjust the weight for ensuring a less cost generated, until an optimal combination has discovered.


HIDDEN
AGE
COMBINATION
ß$_1$+ ß$_2$AGE+ ß$_3$INC
ACTIVATION
tanh(ß$_1$+ ß$_2$AGE+ ß$_3$INC)
=A
OUTPUT
INCOME
COMBINATION
ß$_4$+ ß$_5$AGE+ ß$_6$INC
ACTIVATION
tanh(ß$_4$+ ß$_5$AGE+ ß$_6$INC)
=B
COMBINATION
ß$_{10}$+ß$_{11}$A+ ß$_{12}$B+ß$_{13}$C
ACTIVATION
tanh(ß$_7$+ ß$_8$AGE+ ß$_9$INC)
=C
COMBINATION
ß$_7$+ ß$_8$AGE+ ß$_9$INC

1. Build a Neural Network model using the default setting.

Data pre-processing steps are same as Logistic Regression.

1. **NN_df: Establish Neural Network Model (with default settings) and Train**
2. **Model Evaluation**
3. **Max Iteration Detection**



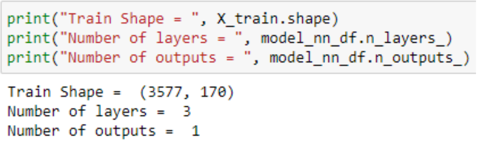
```
model_nn_df = MLPClassifier(random_state=rs)
model_nn_df.fit(X_train, y_train)
```

```
C:\Users\35928\Anaconda3\lib\site-packages\sklearn\neural_network\multilayer_perceptro
n.py:566: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached an
d the optimization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
```

```
MLPClassifier(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_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)
```

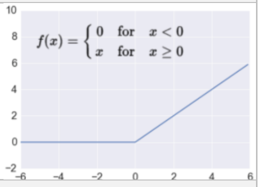
a. Explain the parameters used in building this model, e.g., network architecture, iterations, activation function, etc.

Network architecture (5 layers): Input Layer (1) -> Hidden Layer (3) -> Output Layer (1)
Input Layer: data are presented on this layer. We have 3577 records, where each record is with 170 features
Hidden Layer (300 neurons): 3 layers, with each layer contained 100 neurons.
Output Layer: 1 node output (a binary variable “Covid-19 Positive” or “Covid-19 Negative”)



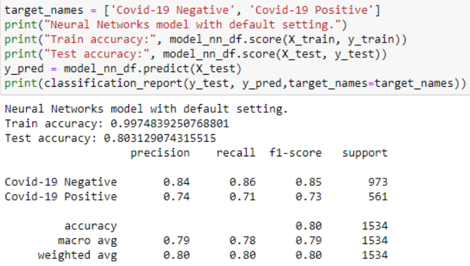
```
print("Train Shape = ", X_train.shape)
print("Number of layers = ", model_nn_df.n_layers_)
print("Number of outputs = ", model_nn_df.n_outputs_)
```

```
Train Shape =  (3577, 170)
Number of layers =  3
Number of outputs =  1
```

Hyperparameters	Value	Description
random_seed	10	The random seed ensures a set of results generated to be consistent along with multiple runs.
hidden_layer_sizes	100	It has values of tuples, and within each tuple, the element i-th represents the number of neurons contained in each hidden layer.
activation	Relu	A rectified linear unit function, returns f(x) = max (0, x), enabling the model train quickly. When the weight of an input is less than 0 (inclusive), it will be treated as 0 by the activation function. 
solver	adam	‘adam’ refers to a stochastic gradient-based optimizer.
alpha	0.0001	L2 regularization parameter is usually used for reducing the probability of occurring overfitting. The value of this hyperparameter is desirable to be relatively smaller. Although it requires extra time in terms of the training process, a smaller Alpha could be helpful to detect the local minimum.
max_iter	200	The maximum number of iterations. The stochastic solver ‘adam’ is determined by the number of epochs, which indicates the number of times every data point will be utilised, instead of the number of gradient steps.

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

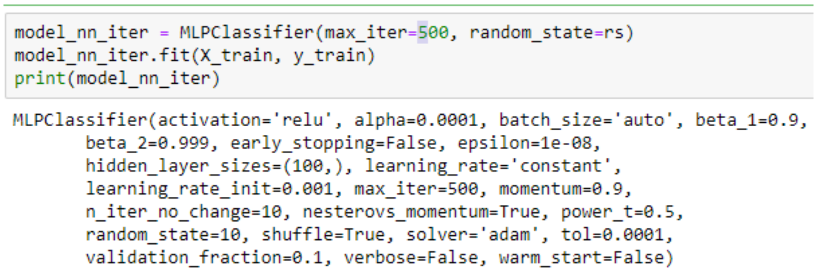
The accuracy is around 0.997 for training and 0.803 for testing datasets. The result indicates that the model is overfitted, as it is tightly fitted with the training dataset. The evaluative metrics (Precision, Recall, or F1-Score) are currently unreliable to be analysed and interpreted.



```
target_names = ['Covid-19 Negative', 'Covid-19 Positive']
print("Neural Networks model with default setting.")
print("Train accuracy:", model_nn_df.score(X_train, y_train))
print("Test accuracy:", model_nn_df.score(X_test, y_test))
y_pred = model_nn_df.predict(X_test)
print(classification_report(y_test, y_pred,target_names=target_names))
```

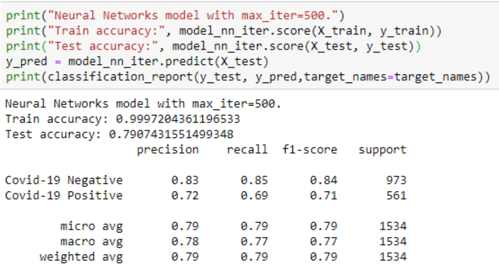
```
Neural Networks model with default setting.
Train accuracy: 0.9974839250768801
Test accuracy: 0.803129074315515
                   precision    recall  f1-score   support

Covid-19 Negative       0.84      0.86      0.85       973
Covid-19 Positive       0.74      0.71      0.73       561

         accuracy                           0.80      1534
        macro avg       0.79      0.78      0.79      1534
     weighted avg       0.80      0.80      0.80      1534
```

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



```
model_nn_iter = MLPClassifier(max_iter=500, random_state=rs)
model_nn_iter.fit(X_train, y_train)
print(model_nn_iter)
```

```
MLPClassifier(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_iter=500, 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)
```



```
print("Neural Networks model with max_iter=500.")
print("Train accuracy:", model_nn_iter.score(X_train, y_train))
print("Test accuracy:", model_nn_iter.score(X_test, y_test))
y_pred = model_nn_iter.predict(X_test)
print(classification_report(y_test, y_pred,target_names=target_names))
```

```
Neural Networks model with max_iter=500.
Train accuracy: 0.9997204361196533
Test accuracy: 0.7907431551499348
                   precision    recall  f1-score   support

Covid-19 Negative       0.83      0.85      0.84       973
Covid-19 Positive       0.72      0.69      0.71       561

        micro avg       0.79      0.79      0.79      1534
        macro avg       0.78      0.77      0.77      1534
     weighted avg       0.79      0.79      0.79      1534
```

The model with max_iter = 200 cannot enable the training process to be reachable convergence, as the Convergence Warning raised by the Virtual Machine (VM), which means the model might experience an issue with error computations. The increase of iterations is a solvable method. By the exploration, when max_iter =500, the issue is no longer prompted. Even no warning raises for the issue, the model is overfitted after evaluation (approximate 100% training accuracy & 79% testing accuracy), which may be solved by the adjustment of hyperparameters, e.g. hidden_layer_sizes and alpha.

2. Refine this network by tuning it with GridSearchCV. Report the trained model.

The range for tunning hidden layer sizes

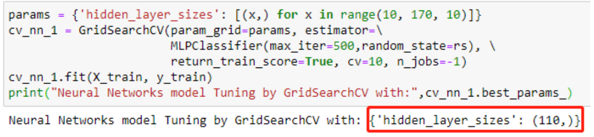
the size of input variables (170) > the number of neurons in a hidden layer > output neurons (1)

1. Tune hidden layer sizes by GridSearchCV

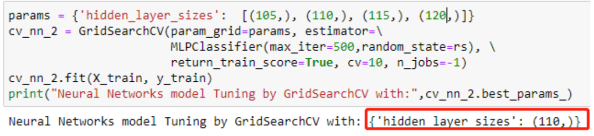
Ex1: tune 10 to 170 neurons, with each increment of 10.

Ex2: tune 4 sets of neurons for the precise detection of the minimum number of neurons in a hidden layer.

Result: 110 neurons in a single hidden layer



```
params = {'hidden_layer_sizes': [(x,) for x in range(10, 170, 10)]}
cv_nn_1 = GridSearchCV(param_grid=params, estimator=\
                       MLPClassifier(max_iter=500,random_state=rs), \
                       return_train_score=True, cv=10, n_jobs=-1)
cv_nn_1.fit(X_train, y_train)
print("Neural Networks model Tuning by GridSearchCV with:",cv_nn_1.best_params_)

Neural Networks model Tuning by GridSearchCV with: {'hidden_layer_sizes': (110,)}
```

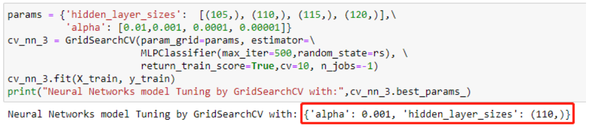


```
params = {'hidden_layer_sizes':  [(105,), (110,), (115,), (120,)]}
cv_nn_2 = GridSearchCV(param_grid=params, estimator=\
                       MLPClassifier(max_iter=500,random_state=rs), \
                       return_train_score=True, cv=10, n_jobs=-1)
cv_nn_2.fit(X_train, y_train)
print("Neural Networks model Tuning by GridSearchCV with:",cv_nn_2.best_params_)

Neural Networks model Tuning by GridSearchCV with: {'hidden_layer_sizes': (110,)}
```

2. Tune alpha by GridSearchCV

Ex3: tune 4 sets of alpha values combined with 4 sets of hidden layer sizes to detect the optimal alpha value.

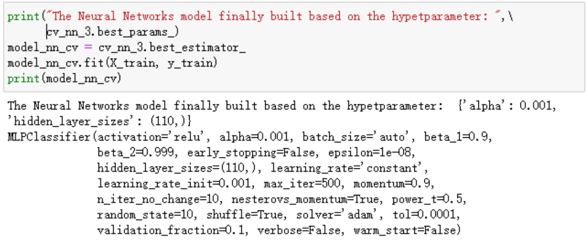
Result: alpha (0.001) & hidden_layer_sizes (110)



```
params = {'hidden_layer_sizes':  [(105,), (110,), (115,), (120,)],\
         'alpha': [0.01,0.001, 0.0001, 0.00001]}
cv_nn_3 = GridSearchCV(param_grid=params, estimator=\
                       MLPClassifier(max_iter=500,random_state=rs), \
                       return_train_score=True,cv=10, n_jobs=-1)
cv_nn_3.fit(X_train, y_train)
print("Neural Networks model Tuning by GridSearchCV with:",cv_nn_3.best_params_)

Neural Networks model Tuning by GridSearchCV with: {'alpha': 0.001, 'hidden_layer_sizes': (110,)}
```

4. NN_cv: Establish Neural Network Model

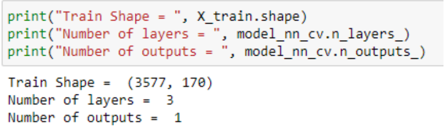
5. Model Evaluation



```
print("The Neural Networks model finally built based on the hypetparameter: ",\
      cv_nn_3.best_params_)
model_nn_cv = cv_nn_3.best_estimator_
model_nn_cv.fit(X_train, y_train)
print(model_nn_cv)

The Neural Networks model finally built based on the hypetparameter:  {'alpha': 0.001,
'hidden_layer_sizes': (110,)}
MLPClassifier(activation='relu', alpha=0.001, batch_size='auto', beta_1=0.9,
              beta_2=0.999, early_stopping=False, epsilon=1e-08,
              hidden_layer_sizes=(110,), learning_rate='constant',
              learning_rate_init=0.001, max_iter=500, 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)
```

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

The experiment only tunes two hyperparameters i.e. Hidden_Layer_Sizes (100 -> 110) and Alpha (0.0001 -> 0.001), and others are the same as NN_df model (shown in the screenshot in Step 2).



```
print("Train Shape = ", X_train.shape)
print("Number of layers = ", model_nn_cv.n_layers_)
print("Number of outputs = ", model_nn_cv.n_outputs_)

Train Shape =  (3577, 170)
Number of layers =  3
Number of outputs =  1
```

Network architecture (5 layers): Input Layer (1) -> Hidden Layer (3) -> Output Layer (1)

Input Layer: 3577 records, where each record is with 170 features

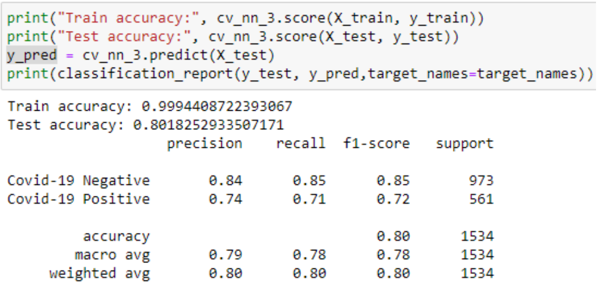
Hidden Layer (330 neurons): 3 layers, with each layer contained 110 neurons.

Output Layer: 1 node output (a binary variable "Covid-19 Positive" or "Covid-19 Negative")

It is noticed that the number of neurons in Hidden Layer changes from 300 to 330, on account of the change of Hidden_Layer_Sizes.

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

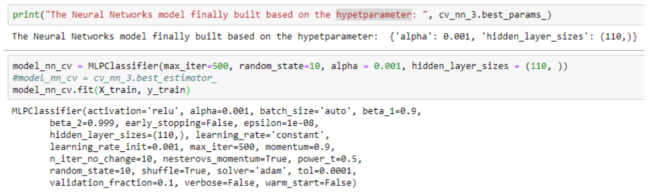
The accuracy is around 0.999 for training and 0.802 for testing datasets. Compared with the NN_df model, the training accuracy increases 0.002 from **0.997 to 0.999**, while the testing decreases 0.001 from **0.803 to 0.802**. The result indicates that the model is overfitted, as it is tightly fitted with the training dataset. Also, the model performance is harmed on account of the decrease of testing accuracy. The evaluative metrics (Precision, Recall, or F1-Score) are currently unreliable to be analysed and interpreted. We will iteratively tune the model in the next stage.



```
print("Train accuracy:", cv_nn_3.score(X_train, y_train))
print("Test accuracy:", cv_nn_3.score(X_test, y_test))
y_pred = cv_nn_3.predict(X_test)
print(classification_report(y_test, y_pred,target_names=target_names))

Train accuracy: 0.9994408722393067
Test accuracy: 0.8018252933507171
                   precision    recall  f1-score   support

Covid-19 Negative       0.84      0.85      0.85       973
Covid-19 Positive       0.74      0.71      0.72       561

         accuracy                           0.80      1534
        macro avg       0.79      0.78      0.78      1534
     weighted avg       0.80      0.80      0.80      1534
```

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

The model with random_state = 10, max_iter = 500, alpha = 0.001, and hidden_layer_sizes = (110,) is successfully converged, and there is no warning prompt. However, it may be not the best model, as the training accuracy implies the potential overfitting issue existed. Also, the testing accuracy experiences a slight decrease compared with the original model.



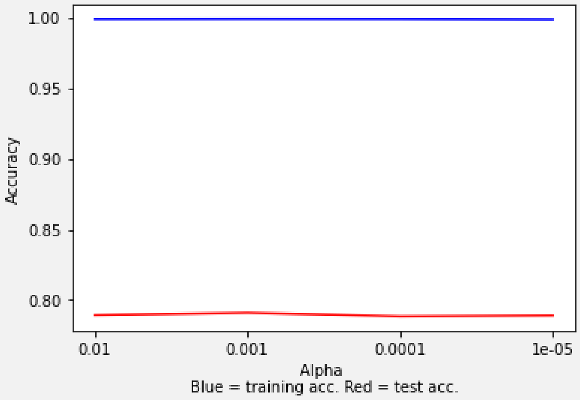
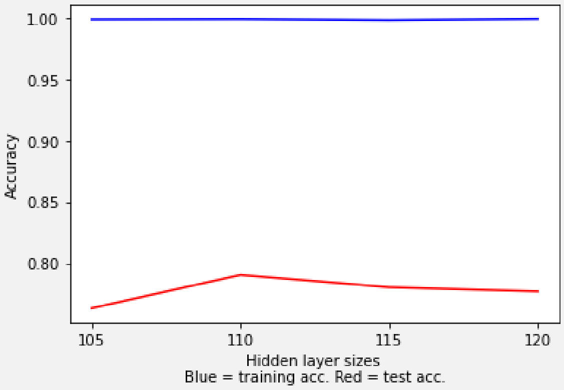
```
print("The Neural Networks model finally built based on the hypetparameter: ", cv_nn_3.best_params_)
```

```
The Neural Networks model finally built based on the hypetparameter:  {'alpha': 0.001, 'hidden_layer_sizes': (110,)}
```

```
model_nn_cv = MLPClassifier(max_iter=500, random_state=10, alpha = 0.001, hidden_layer_sizes = (110, ))
#model_nn_cv = cv_nn_3.best_estimator_
model_nn_cv.fit(X_train, y_train)
```

```
MLPClassifier(activation='relu', alpha=0.001, batch_size='auto', beta_1=0.9,
       beta_2=0.999, early_stopping=False, epsilon=1e-08,
       hidden_layer_sizes=(110,), learning_rate='constant',
       learning_rate_init=0.001, max_iter=500, 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)
```

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

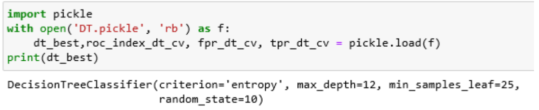
<table>
<tr><td colspan="2"><b>Tabular</b>: the model has significant the overfitting issue as the training accuracy is nearly 100%, which implies the model fits the train set very well. Let’s get an insight into this model.</td></tr>
<tr><td><b>Alpha Evaluation with Fixed Hidden Layer Sizes (110)</b></td><td><b>Hidden Layer Sizes Evaluation with Fixed Alpha (0.001)</b></td></tr>
<tr><td>
</td><td>
</td></tr>
<tr><td>The graph is plotted for exploring the model performance when it is fed to various alpha values [0.01, 0.001, 0.0001, 0.00001]. The gap is between training and testing is approximate 20%. The red line for testing accuracy slightly increases from 0.01 to 0.001, and then decreases. The training accuracy is stable at 100% approximately. Although it seemingly indicates the model is successful to avoid overfitting issue after 0.001 Alpha, it is under overfitting, even the hyperparameters are tuned by GridSearchCV. The feature is that testing accuracy is unable to improve, lacking the capability to generalise unseen data. Another overfitting sign is that the training data is tightly fitted by the model. Hence, despite the model is tuned, it is still suffering from then overfitting problem.</td><td>The graph is plotted when and [alpha = 0.001], the model is fed with different Hidden Layer Sizes values [105, 110, 115, 120]. The overfitting sign apparently may appear after [Hidden Layer Sizes = 110], because the testing accuracy starts by dropping at the point. However, even the model is tuned by GridSearchCV, overfitting issue is not be mitigated, as the model is extremely fitted with the training dataset. It lacks the capability of learning unseen patterns.</td></tr>
</table>

3. Would feature selection help in improving the model? Build another Neural Network model with reduced features set. Perform dimensionality reduction by selecting variables with the decision tree (use the best decision tree model). Tune the model with GridSearchCV to find the best parameters setting.

Dimensionality reduction is either helpful to reduce the time computations in terms of training and testing process, or expected to be improvable to model performance, because a new dataset formed by important features is easy to be efficiently processed than the original dataset. However, Neuron Network Model is not supportive to perform Recursive Feature Elimination (RFE). Here, we adopt the Decision Tree with RFE (the optimal model in Task 1), for discovering crucial features and then donate to our Neuron Network Model.

1. Establish the optimal Decision Tree

criterion = entropy, max_depth = 12,
min_samples_leaf = 25, random seed = 10.

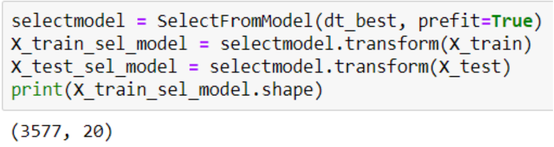


```
import pickle
with open('DT.pickle', 'rb') as f:
    dt_best,roc_index_dt_cv, fpr_dt_cv, tpr_dt_cv = pickle.load(f)
print(dt_best)
```

```
DecisionTreeClassifier(criterion='entropy', max_depth=12, min_samples_leaf=25,
                       random_state=10)
```

2. Execute RFE to select important variables

3577 records and each record are with 20 features



```
selectmodel = SelectFromModel(dt_best, prefit=True)
X_train_sel_model = selectmodel.transform(X_train)
X_test_sel_model = selectmodel.transform(X_test)
print(X_train_sel_model.shape)
```

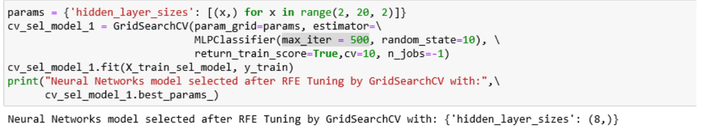
```
(3577, 20)
```

3. Tune hidden layer sizes by GridSearchCV

Ex1: tune 2 to 20 neurons, with each increment of 2.

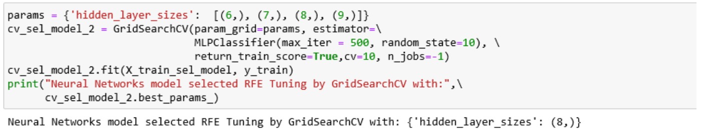
Ex2: tune 4 sets of neurons for the precise detection of the minimum number of neurons in a hidden layer.

Result: 8 neurons in a single hidden layer



```
params = {'hidden_layer_sizes': [(x,) for x in range(2, 20, 2)]}
cv_sel_model_1 = GridSearchCV(param_grid=params, estimator=\
                              MLPClassifier(max_iter = 500, random_state=10), \
                              return_train_score=True,cv=10, n_jobs=-1)
cv_sel_model_1.fit(X_train_sel_model, y_train)
print("Neural Networks model selected after RFE Tuning by GridSearchCV with:",\
      cv_sel_model_1.best_params_)
```

```
Neural Networks model selected after RFE Tuning by GridSearchCV with: {'hidden_layer_sizes': (8,)}
```



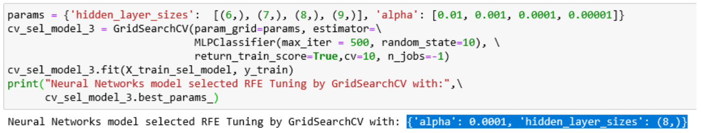
```
params = {'hidden_layer_sizes':  [(6,), (7,), (8,), (9,)]}
cv_sel_model_2 = GridSearchCV(param_grid=params, estimator=\
                              MLPClassifier(max_iter = 500, random_state=10), \
                              return_train_score=True,cv=10, n_jobs=-1)
cv_sel_model_2.fit(X_train_sel_model, y_train)
print("Neural Networks model selected RFE Tuning by GridSearchCV with:",\
      cv_sel_model_2.best_params_)
```

```
Neural Networks model selected RFE Tuning by GridSearchCV with: {'hidden_layer_sizes': (8,)}
```

4. Tune alpha by GridSearchCV

Ex3: tune 4 sets of alpha values combined with 4 sets of hidden layer sizes to detect the optimal alpha value.

Result: **alpha (0.0001) & hidden_layer_sizes (8)**

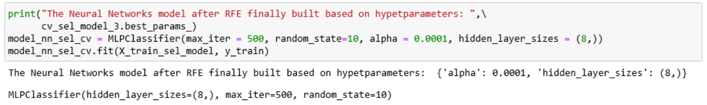


```
params = {'hidden_layer_sizes':  [(6,), (7,), (8,), (9,)], 'alpha': [0.01, 0.001, 0.0001, 0.00001]}
cv_sel_model_3 = GridSearchCV(param_grid=params, estimator=\
                              MLPClassifier(max_iter = 500, random_state=10), \
                              return_train_score=True,cv=10, n_jobs=-1)
cv_sel_model_3.fit(X_train_sel_model, y_train)
print("Neural Networks model selected RFE Tuning by GridSearchCV with:",\
      cv_sel_model_3.best_params_)
```

```
Neural Networks model selected RFE Tuning by GridSearchCV with: {'alpha': 0.0001, 'hidden_layer_sizes': (8,)}
```

4. NN_rfe_cv: Establish Neural Network Model

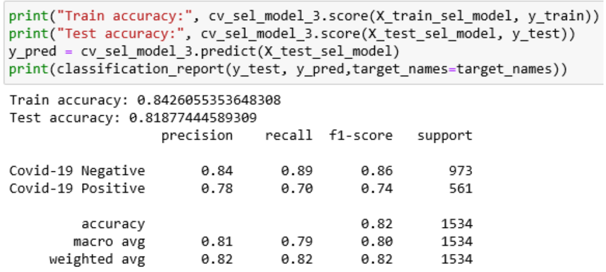
5. Model Evaluation



```
print("The Neural Networks model after RFE finally built based on hypetparameters: ",\
      cv_sel_model_3.best_params_)
model_nn_sel_cv = MLPClassifier(max_iter = 500, random_state=10, alpha = 0.0001, hidden_layer_sizes = (8,))
model_nn_sel_cv.fit(X_train_sel_model, y_train)
```

```
The Neural Networks model after RFE finally built based on hypetparameters:  {'alpha': 0.0001, 'hidden_layer_sizes': (8,)}

MLPClassifier(hidden_layer_sizes=(8,), max_iter=500, random_state=10)
```

a. Did feature selection favour the outcome? Any change in network architecture? What inputs are being used as the network input? b. What is classification accuracy on training and test datasets?

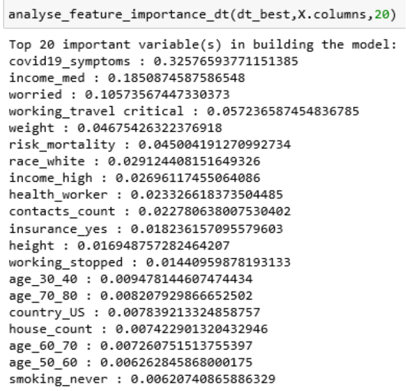


```
print("Train accuracy:", cv_sel_model_3.score(X_train_sel_model, y_train))
print("Test accuracy:", cv_sel_model_3.score(X_test_sel_model, y_test))
y_pred = cv_sel_model_3.predict(X_test_sel_model)
print(classification_report(y_test, y_pred,target_names=target_names))
```

```
Train accuracy: 0.8426055353648308
Test accuracy: 0.81877444589309
                   precision    recall  f1-score   support

Covid-19 Negative       0.84      0.89      0.86       973
Covid-19 Positive       0.78      0.70      0.74       561

         accuracy                           0.82      1534
        macro avg       0.81      0.79      0.80      1534
     weighted avg       0.82      0.82      0.82      1534
```

The feature selection approach is beneficial to the outcome. The training accuracy to the previous two models is approximately reachable to 100%, which explicitly reflects the overfitting issue existed. After dimensionality reduction, the accuracy ratios of training and testing are around **84% (decrease) and 82% (increase)** respectively. It may indicate that the model is avoidable to the overfitting issue, which will be verified in Question d. Also, the running time is decreased significantly on account of features reduced (improve efficiency). As the screenshot shows in Step 2, there have only 20 features fed to the model, which is listed as the left-side screenshot. The evaluative metrics reflects the Precision to Covid-19 patients is 78% and Recall is 70%, which may be a desirable result. Another benefit can reflect on the change of Network Architecture, which enables the model to be simple than the previous models, as the number of neurons in Hidden Layers decreases from 330 to 42, summarised as:

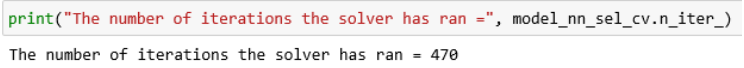
- **Network architecture (5 layers):** Input Layer (1) -> Hidden Layer (3) -> Output Layer (1)
- **Input Layer:** 3577 records, where each record is with 20 features
- **Hidden Layer (24 neurons):** 3 layers, with each layer contains 8 neurons.
- **Output Layer:** 1 node output (a binary variable “Covid-19 Positive” or “Covid-19 Negative”)



```
analyse_feature_importance_dt(dt_best,X.columns,20)
```

```
Top 20 important variable(s) in building the model:
covid19_symptoms : 0.32576593771151385
income_med : 0.1850874587586548
worried : 0.10573567447330373
working_travel critical : 0.057236587454836785
weight : 0.046754263223769l8
risk_mortality : 0.045004191270992734
race_white : 0.029124408151649326
income_high : 0.02696117455064086
health_worker : 0.023326618373504485
contacts_count : 0.022780638007530402
insurance_yes : 0.018236157095579603
height : 0.016948757282464207
working_stopped : 0.01440959878193133
age_30_40 : 0.009478144607474434
age_70_80 : 0.008207929866652502
country_US : 0.007839213324858757
house_count : 0.007422901320432946
age_60_70 : 0.007260751513755397
age_50_60 : 0.006262845868000175
smoking_never : 0.006207408658886329
```

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

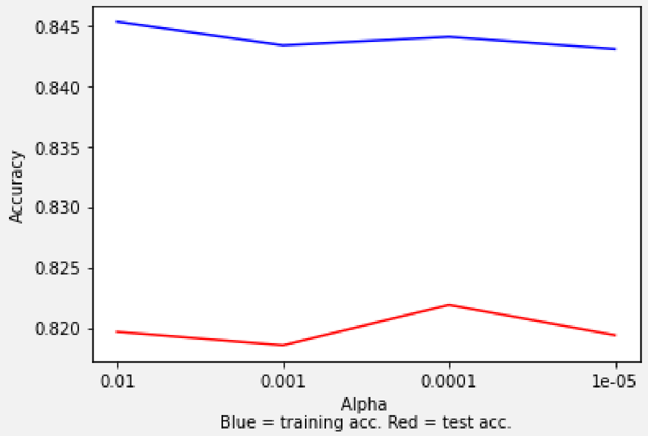
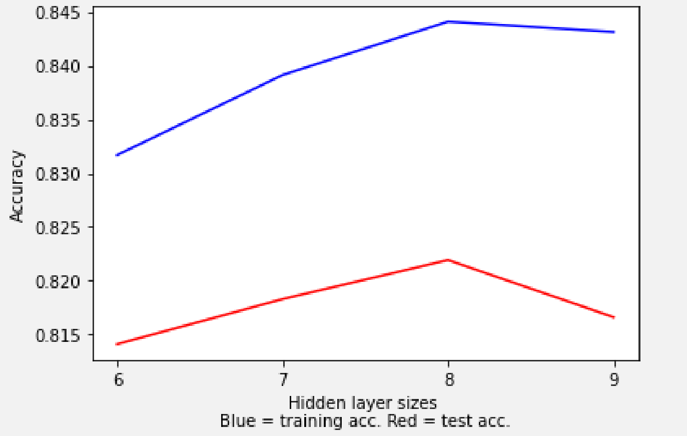


```
print("The number of iterations the solver has ran =", model_nn_sel_cv.n_iter_)
```

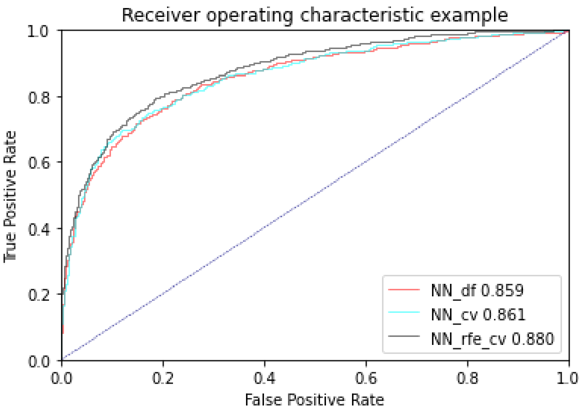
```
The number of iterations the solver has ran = 470
```

We utilise the max_iter (500) and there is no warning popped. But we find the model only required 470 iterations to be completely trained on the new dataset currently, with 3577 records and 20 features.

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

<table>
<tr><th>Alpha Evaluation with Fixed Hidden Layer Sizes (8)</th><th>Hidden Layer Sizes Evaluation with Fixed Alpha (0.0001)</th></tr>
<tr><td>
</td><td>
</td></tr>
<tr><td>The graph is plotted for exploring the model performance when it is fed to various alpha values [0.01, 0.001, 0.0001, 0.00001]. Initially, training and testing accuracies experience decrease from 0.01 to 0.001 (underfitting occurred). After they are significantly increased until reaching to 0.0001. Both accuracies start by decreasing after this point. It is a sign of overfitting phenomenon. Hence, 0.001 is an optimal Alpha value, enabling the model to avoid overfitting. It proves that Feature Selection combined with GridSearchCV is beneficial to improve model performance.</td><td>The graph is plotted when and [alpha = 0.0001], the model is fed with different Hidden Layer Sizes values [6, 7, 8, 9]. Obviously, the lines for both accuracies reflect an upward trend until they are reached to the point [Hidden Layer Size = 8] and then substantially dropped, which is an overfitting sign. This experiment proves that neurons 8 is an optimal value in each Hidden Layer, enabling the model to avoid an overfitting issue.</td></tr>
<tr><td colspan="2"><b>Summary</b>: the mode with max_iter (500) is successfully converged, because there is no Convergence Warning Prompt, which is the best model currently, as it avoids the overfitting issue and contributes to a decent result, with Hidden Layer Sizes (8) and Alpha (0.0001).</td></tr>
</table>

4. Produce the ROC curve for all different NNs. Using the best neural network model, can you provide general characteristics of the "COVID-19 Positive" patients identified by the model? If it is hard to comprehend, discuss why?

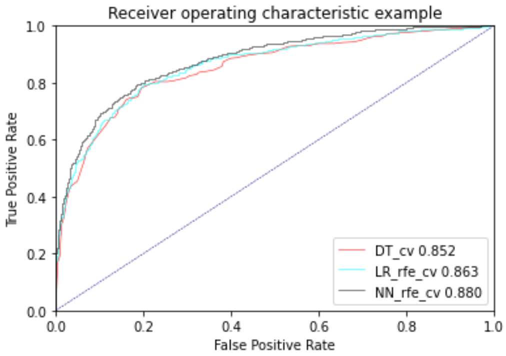


The right-side ROC curve indicates that **NN_rfe_cv is well-performed on classification on D3.csv dataset, as it contributes the highest number of AUC score (0.88).** By the analysis before, NN_rfe_cv is successful to mitigate the probability of the occurrence of overfitting issues. The resultant evaluation of this model is more reliable than NN_df and NN_cv, as the two models suffer from the overfitting issues. Hence, we say the best choice to perform mining task is NN_rfe_cv model. However, Neuron Network is an uninterpretable model. The general-purpose to the model is that it usually works in a Black box, where the process is non-transparent. The model is able to deal with complicated classification task, and adaptive to cope with data problems (e.g. Missing Values and a large number of features), but it does not provide any extra information (e.g. Feature Importance) enabling to get insight the dataset, which is totally different from Decision Tree and Logistic Regression. **As for identifying the "COVID-19 Positive" patients, it may be difficult to achieve the purpose with NN_rfe_cv model**, but it can generate a desirable result to the dataset.

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.

Evaluative Measures		Decision Tree (DT_cv)	Logistic Regression (LR_rfe_cv)	Neuron Network (NN_rfe_cv)
Accuracy	Train	0.8319	0.8160	0.8426
	Test	0.8011	0.8057	0.8187
Precision	**Positive**	**0.73**	**0.77**	**0.78**
	Negative	0.84	0.82	0.84
Recall	**Positive**	**0.70**	**0.67**	**0.70**
	Negative	0.81	0.88	0.89
F1	**Positive**	**0.73**	**0.72**	**0.74**
	Negative	0.84	0.85	0.86
ROC AUC	-	0.852	0.863	0.88



We recognise that Logistic Regression is well-performed by comparison of the three optimal models (excepting the ensemble model), which is analysed the table. However, regardless of Precision, Recall or F1-score to Covid-19 Positive patients, it seems that Neuron Network (NN) is slightly decent than Logistic Regression. NN model is **usually complicated and time-consuming during** classification. We are desirable to generate a simple model with decent efficiency and effectiveness, as well as it is expected to be easily interpreted. It is a trade-off choice. Although NN contributes 0.880 AUC score, 0.70 Recall, and 0.78 Precision, that are higher than Logistic Regression, **Logistic Regression only requires less time to on training and then contributes to the result that is close the NN model** (e.g. Precision to Covid-19 Positive patients). In the comparison of performance with Decision Tree and Logistic Regression, they are close to each other. However, we are expected to enable the model with the capability in accurately predicting Covid-19 Positive patients, so that 77% of Positive Precision is more suitable to perform prediction. Also, Logistic Regression contributes to 0.863 AUC score, that is higher than Decision Tree. Hence, we acknowledge that Logistic Regression is a decent model in our experiment.

Extensive Knowledge

As the environment is stochastic, the collection of data for mining tasks is usually unpredictable, e.g. noise and outliers. An individual model may not be well-fitted to tackle the problem. An innovative approach – Ensemble Modelling, is more robust to cope with this problem. The method is capable of combining different models for a particular mining task. In the future task, we plan to design an experiment to utilise the method and combine with the three optimal models proposed before, thereby evaluating its performance in the dataset. Ensemble Modelling is supportive to three major techniques, including Bagging, Boosting and Stacking. Here, we only focus on a simple technical method – Bagging, which performs prediction based on the average/voting of processes of our three well-calibrated models, i.e. the optimal Decision Tree, Logistic Regression and Neuron Network. We donate soft voting to be a suggestive hyperparameter the Ensemble Model, where it performs predictions underlying on the softmax value of predicted probabilities. This ideal design will be donated for further investigation and experiment.

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

	Merits	Demerits
Decision Tree	• Light Model to easily establish (no need for tedious steps to configure hyperparameters) • A predictable training time, based on the completion of constructing the ultimate tree. • Easily interpret to Model and Results • Be capable of processing a number of features and various categories (e.g. nominal, ordinal, or interval) • Robustness in handling data with different distributions and formats. • Robustness in tacking Missing Value issue (but the issue is not in D3.csv dataset) • Regression tree (allow input variables with continuous and categorical to perform prediction)	• Be ill-performed to process features with a complicated relationship. • Be rough to define the tree boundaries • Be problematic to handle numerous missing data • Be unstable, because it heavily relies on the decision of the optimal first split. • Be tedious to Tree Virtualisation if the tree is more complicated (Our result contributes to a tree with depth = 12, which is not an easy task to interpret and virtualise)
Logistic Regression Note that LR model is required to be fed to a relatively small and clean dataset, e.g. D3.csv dataset.	• Light Model to easily establish (no need for tedious steps to configure hyperparameters) • The algorithm is widely available and scientifically acceptable to explore many data mining problems, so that it is usually the first choice for the purpose. • Be simple and interpretable, as the *feature importance* offers the informative result enabling to get insight the data and stakeholders.	• Problems in the real world are usually stochastic, which cannot be summarised or interpreted as a linear problem. This a limitation to Logistic Regression, that it may not generate a desirable and reliable result. • Be ill-performed to the dataset with numerous features. • Be unable to process the Missing Value issue.
Neuron Network The model is applied when **Decision Tree fails** to perform the data mining task, or the **model is not required to be interpreted**.	• Fast Application (no need for tedious steps to configure, but the model is supportive to set various hyperparameters to tune the model). • Be capable of learning complicated class boundaries. • Be adaptive to handle a dataset with the Missing Value problem numerous features.	• Tuning Model is challenging work, as its process to predict is unknown and difficult to interpret (work in Black-Box). • Unpredicted Training Time (slow) • The number of nodes (e.g. hidden layers) defined to the model is associated with trials and error testing, which is a time-consuming process.

Appendix

Decision Tree