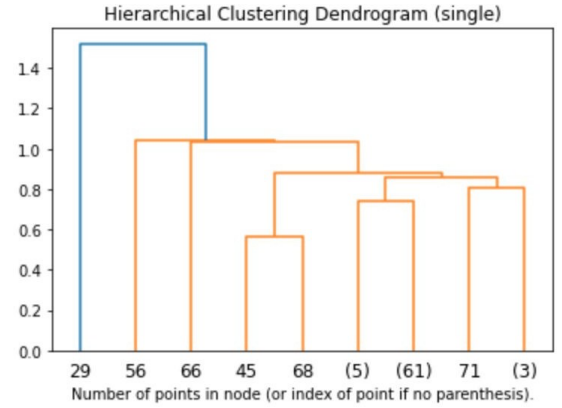
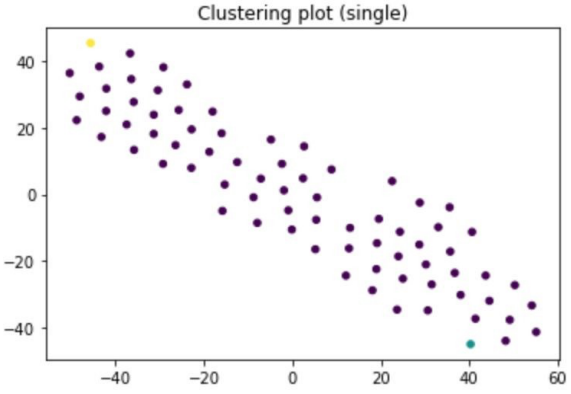
COMP4331 Project

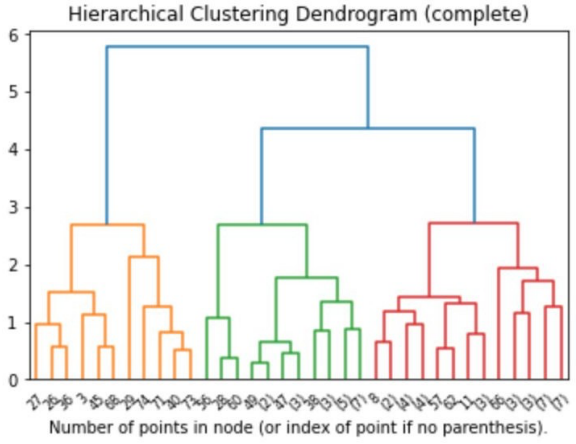
Q1b

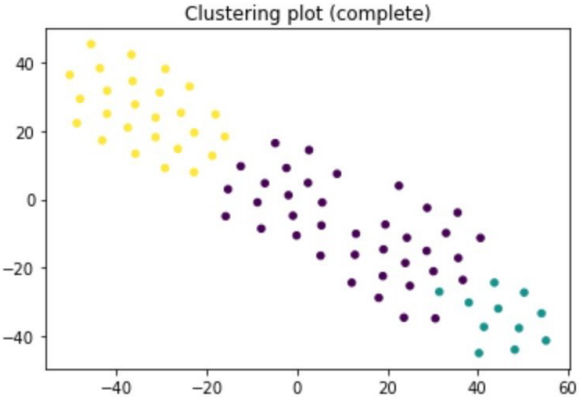
Single-link distance



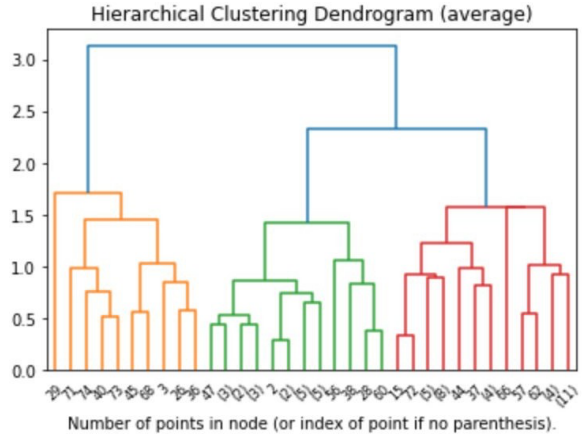


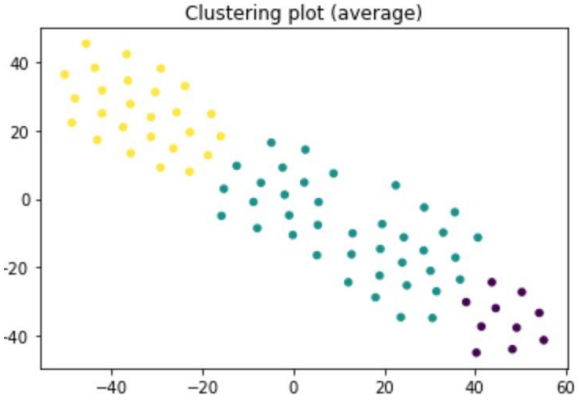
Complete-link distance





Group average distance





Q1c

The following are the Davies-Bouldin score and Silhouette score for the cluster using different distance measures.

```
Davies-Bouldin score (single): 0.5137742665128471
Davies-Bouldin score (complete): 0.819269481106398
Davies-Bouldin score (average): 0.7733178860340705

Silhouette score (single): 0.1020194433430128
Silhouette score (complete): 0.41386704795264717
Silhouette score (average): 0.42968049502721345
```

Davies-Bouldin score

This score is a ratio between "within-cluster" and "between-cluster" distances which evaluates the intra-cluster similarity and inter-cluster differences. If the spread in the cluster is small while the difference between the two cluster groups is large, that means the clusters are distinct. Thus, in general, the smaller the Davies-Bouldin score, the better the clustering performance.

However, we also have to take into account the number of data points in each cluster. From the result above, the clustering solution obtained through single-link distance has the lowest ratio. Such a low score has resulted from the fact it contains single-element clusters. This reduced their intra-cluster difference to zero which might lead to a good but misleading result.

For the clustering solution obtained by complete-link distance and group average distance, they have a more balanced data point in each cluster group. It is valid to compare the two using the Davies-Bouldin score. Thus we can conclude that the clustering solution obtained using group average distance appears to be the optimal clustering which has the lower score.

Silhouette score

The score measures the average similarity of the objects within a cluster [a(i)] and their distance to the other objects in the other clusters [b(i)]. The score is measured as follows.

$$S(i) = \frac{b(i) - a(i)}{\max\left(a(i),\ b(i)\right)}$$

If the score is close to 1, that means the points are correctly clustered as it is closer to its own cluster than to its neighbour. Vice versa for the case of score near -1. Silhouette score does take into account the occurrence of the single-element clusters and will assign a 0 score for them. Therefore, we can directly compare the three scores resulting in the three models. The clustering solution performed using group average distance has the highest Silhouette score and thus it is the optimal clustering.

Q1di.

Countries in the 1st cluster:

Bolivia, Guatemala, Lao People's Democratic Republic, Peru, Cambodia, Myanmar, Nicaragua, Nepal, Senegal, Mali

Countries in the 2nd cluster:

Kyrgyzstan, Ukraine, Libyan Arab Jamahiriya, Dominican Republic, Egypt, Oman, Costa Rica, Colombia, Uzbekistan, Morocco, South Africa, El Salvador, Vietnam, Azerbaijan, Algeria, Moldova, Republic of, Paraguay, Thailand, Bosnia and Herzegovina, Ecuador, Jordan, Venezuela, Sri Lanka, Serbia, Albania, Tunisia, Croatia, Turkey, Honduras, Panama, Uruguay, Kazakhstan, Romania, Bulgaria, Chile, Puerto Rico, Argentina, Belarus, Iraq

Countries in the 3rd cluster:

Italy, Canada, Austria, Czech Republic, Malaysia, New Zealand, United Kingdom, Slovakia, Kuwait, Poland, Switzerland, Greece, Finland, Portugal, Sweden, Norway, Germany, Hungary, Slovenia, Saudi Arabia, Australia, Ireland, France, United Arab Emirates, Spain, Denmark

Q1dii.

Summary statistics

	Attribute	mean	std	min	25%	50%	75%	max
0	pop_total(label_0)	2.094376e+07	1.428360e+07	6.545502e+06	1.270892e+07	1.654528e+07	2.637104e+07	5.404542e+07
1	pop_density(label_0)	7.405583e+01	6.113768e+01	1.048015e+01	2.639265e+01	6.798283e+01	8.962918e+01	1.959391e+02
2	GDP(label_0)	6.468768e+03	3.379900e+03	2.423829e+03	3.811473e+03	5.493236e+03	8.784342e+03	1.338036e+04
3	basic_water(label_0)	8.497895e+01	6.105321e+00	7.826083e+01	8.088958e+01	8.191797e+01	9.054897e+01	9.419058e+01
4	safe_water(label_0)	3.892318e+01	1.419163e+01	1.608187e+01	2.704056e+01	4.157917e+01	5.128805e+01	5.599078e+01
5	basic_san(label_0)	6.254398e+01	1.109500e+01	3.933542e+01	5.959946e+01	6.319314e+01	7.202300e+01	7.445941e+01
6	safe_san(label_0)	3.205950e+01	1.259460e+01	1.870940e+01	2.218478e+01	2.903868e+01	3.896935e+01	5.805359e+01
7	pop_total(label_1)	2.464838e+07	2.666615e+07	2.657637e+06	6.455226e+06	1.073896e+07	3.789078e+07	1.003881e+08
8	pop_density(label_1)	9.532275e+01	9.108407e+01	3.795632e+00	3.885715e+01	7.702967e+01	1.017429e+02	3.600174e+02
9	GDP(label_1)	1.748943e+04	8.003124e+03	5.470812e+03	1.183343e+04	1.564373e+04	2.270090e+04	3.594819e+04
10	basic_water(label_1)	9.557285e+01	3.982459e+00	8.552212e+01	9.367335e+01	9.648397e+01	9.900780e+01	1.000000e+02
11	safe_water(label_1)	7.847559e+01	1.143773e+01	5.883333e+01	7.014479e+01	7.506801e+01	8.973583e+01	9.863917e+01
12	basic_san(label_1)	9.234720e+01	6.657334e+00	7.574710e+01	8.778764e+01	9.425851e+01	9.745348e+01	1.000000e+02
13	safe_san(label_1)	5.208711e+01	1.790631e+01	1.698649e+01	4.153926e+01	5.253579e+01	6.463559e+01	8.055492e+01
14	pop_total(label_2)	2.341429e+07	2.378461e+07	2.087946e+06	5.594874e+06	1.027744e+07	3.675908e+07	8.313280e+07
15	pop_density(label_2)	1.080339e+02	7.705248e+01	3.247871e+00	3.639949e+01	1.075547e+02	1.371456e+02	2.747090e+02
16	GDP(label_2)	4.981106e+04	1.390027e+04	2.952558e+04	4.104595e+04	4.917183e+04	5.599294e+04	8.824090e+04
17	basic_water(label_2)	9.960564e+01	8.596815e-01	9.669594e+01	9.974045e+01	1.000000e+02	1.000000e+02	1.000000e+02
18	safe_water(label_2)	9.780508e+01	2.866048e+00	8.957276e+01	9.577522e+01	9.839223e+01	9.979693e+01	1.039525e+02
19	basic_san(label_2)	9.892931e+01	1.691049e+00	9.124518e+01	9.877848e+01	9.925553e+01	9.982001e+01	1.000000e+02
20	safe_san(label_2)	9.046083e+01	7.606282e+00	7.563987e+01	8.345785e+01	9.331592e+01	9.653514e+01	1.000000e+02

How do we select the two attributes that best illustrated the difference?

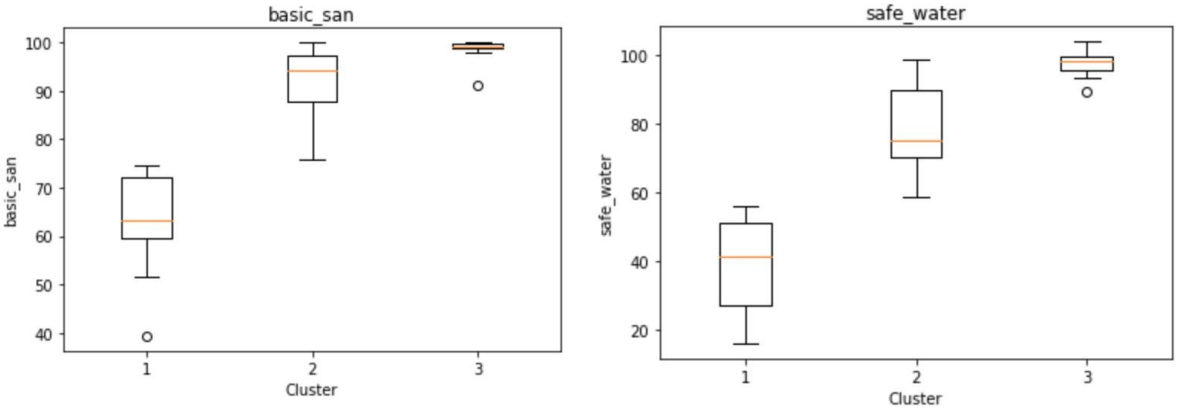
We used the z-score difference to measure the difference. We started by calculating the column mean and standard deviation for each attribute using the unnormalized data. Then, for each attribute, we will calculate the z-score for each cluster group (labeled as z_1, z_2,

z_3). We summed up the absolute difference among the three z-score using “diff = abs(z_3-z_1) + abs(z_3-z_2) + abs(z_2-z_1)”.

From the following table, we can see “safe_water” and “basic_san” are the two attributes that illustrate the most difference.

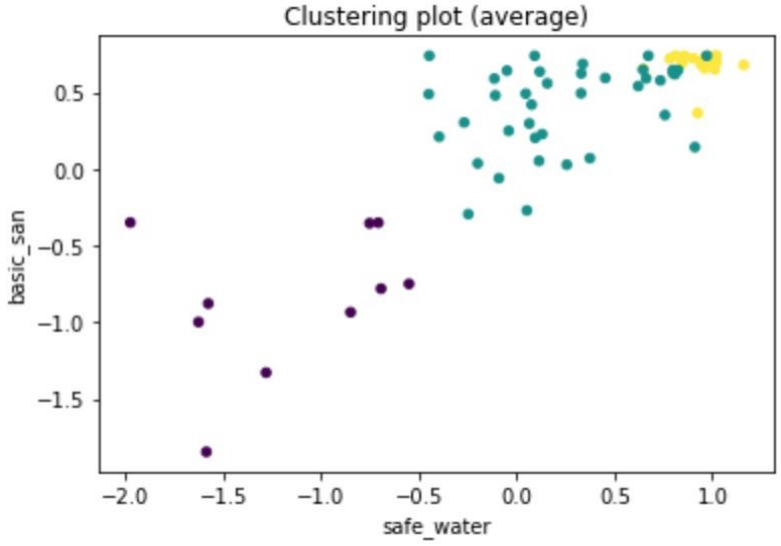
	Attribute	Cluster1	Cluster2	Cluster3	Z-score difference
4	safe_water	-1.964680	-0.068426	0.858285	5.645929
5	basic_san	-2.149417	0.129369	0.632645	5.564122
3	basic_water	-1.819507	0.002491	0.696073	5.031161
6	safe_san	-1.192999	-0.413716	1.079421	4.544840
2	GDP	-1.055142	-0.494903	1.148178	4.406639
1	pop_density	-0.276219	-0.019000	0.134738	0.821915
0	pop_total	-0.115293	0.038188	-0.012939	0.306962

Boxplots



Q1diii.

Scatter plot



Q2a

Policy combinations in the 1st cluster:

	support	itemsets	length
11	0.573290	(H2_Testing policy_1)	1
13	0.397394	(H3_Contact tracing_1)	1
38	0.322476	(H2_Testing policy_1, C2_Workplace closing_2)	2
39	0.351792	(H3_Contact tracing_1, C2_Workplace closing_2)	2
90	0.560261	(H2_Testing policy_1, H1_Public information campaigns_2)	2
92	0.397394	(H3_Contact tracing_1, H1_Public information campaigns_2)	2
104	0.309446	(C2_Workplace closing_2, H3_Contact tracing_1, C1_School closing_3)	3
179	0.322476	(H1_Public information campaigns_2, H2_Testing policy_1, C2_Workplace closing_2)	3
180	0.351792	(H1_Public information campaigns_2, H3_Contact tracing_1, C2_Workplace closing_2)	3
317	0.309446	(C2_Workplace closing_2, H1_Public information campaigns_2, H3_Contact tracing_1, C1_School closing_3)	4

Policy combinations in the 2nd cluster:

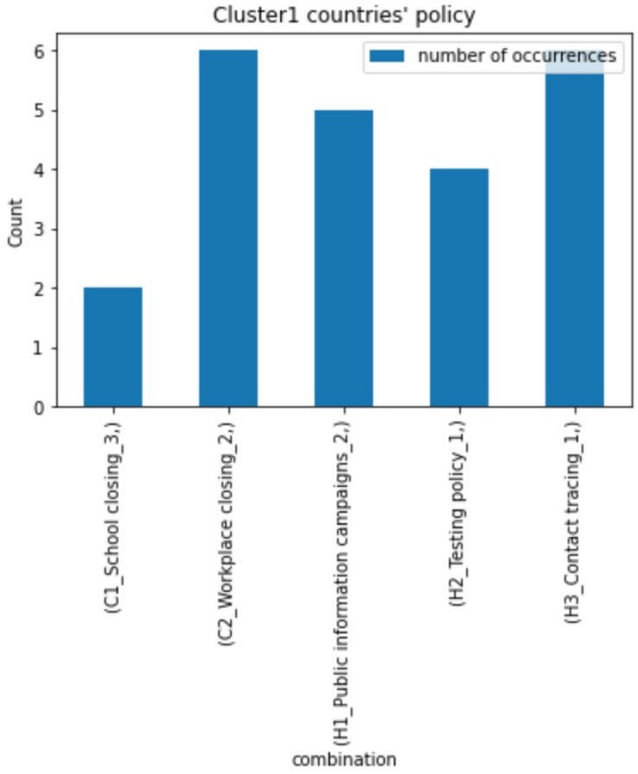
No policy combinations for this cluster is found.

Policy combinations in the 3rd cluster:

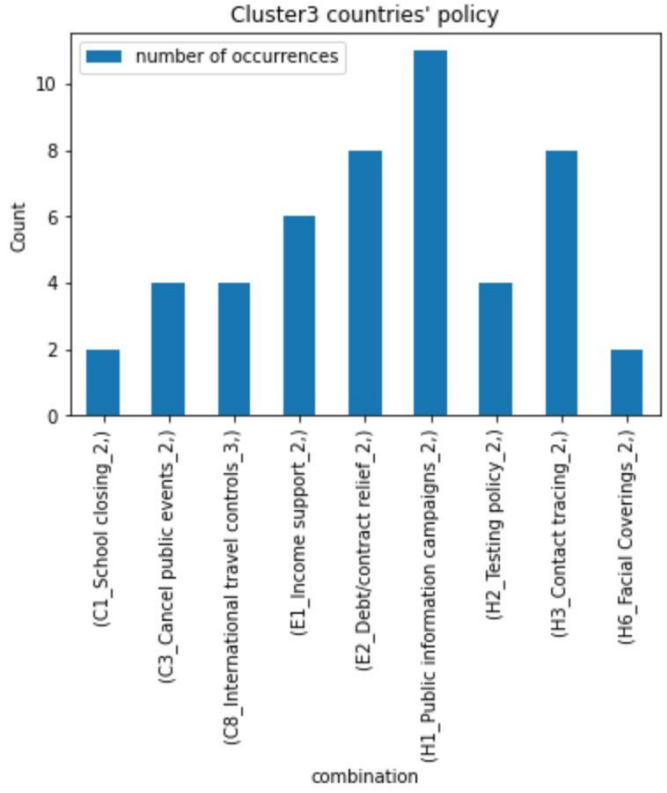
	support	itemsets	length
0	0.325203	(C1_School closing_2)	1
12	0.521341	(E2_Debt/contract relief_2)	1
17	0.347561	(H6_Facial Coverings_2)	1
18	0.324187	(H1_Public information campaigns_2, C1_School closing_2)	2
24	0.329268	(C8_International travel controls_3, C3_Cancel public events_2)	2
26	0.328252	(C3_Cancel public events_2, E2_Debt/contract relief_2)	2
37	0.394309	(C8_International travel controls_3, H3_Contact tracing_2)	2
38	0.341463	(E1_Income support_2, E2_Debt/contract relief_2)	2
40	0.315041	(H2_Testing policy_2, E1_Income support_2)	2
41	0.489837	(E1_Income support_2, H3_Contact tracing_2)	2
43	0.521341	(E2_Debt/contract relief_2, H1_Public information campaigns_2)	2
44	0.410569	(H3_Contact tracing_2, E2_Debt/contract relief_2)	2
48	0.347561	(H6_Facial Coverings_2, H1_Public information campaigns_2)	2
49	0.365854	(H2_Testing policy_2, H3_Contact tracing_2)	2
54	0.329268	(C8_International travel controls_3, C3_Cancel public events_2, H1_Public information campaigns_2)	3
56	0.328252	(C3_Cancel public events_2, E2_Debt/contract relief_2, H1_Public information campaigns_2)	3
59	0.394309	(C8_International travel controls_3, H3_Contact tracing_2, H1_Public information campaigns_2)	3
60	0.341463	(E1_Income support_2, E2_Debt/contract relief_2, H1_Public information campaigns_2)	3
61	0.315041	(H2_Testing policy_2, E1_Income support_2, H1_Public information campaigns_2)	3
62	0.489837	(E1_Income support_2, H3_Contact tracing_2, H1_Public information campaigns_2)	3
63	0.410569	(H3_Contact tracing_2, E2_Debt/contract relief_2, H1_Public information campaigns_2)	3
64	0.365854	(H2_Testing policy_2, H3_Contact tracing_2, H1_Public information campaigns_2)	3

Q2b

Histogram for the 1st cluster:



Histogram for the 3rd cluster:



Q3a

All symptom combinations that appear in at least 20% of all the records

	support	itemsets	length
0	0.499224	(pct_fever_weighted)	1
1	0.500000	(pct_cough_weighted)	1
2	0.499224	(pct_difficulty_breathing_weighted)	1
3	0.499482	(pct_fatigue_weighted)	1
4	0.499741	(pct_stuffy_runny_nose_weighted)	1
...	...	...	...
2300	0.216615	(pct_nausea_weighted, pct_headache_weighted, pct_eye_pain_weighted, pct_chills_weighted, pct_anosmia_ageusia_weighted, pct_sore_throat_weighted, pct_aches_muscle_pain_weighted, pct_chest_pain_weighted)	8
2301	0.200052	(pct_nausea_weighted, pct_headache_weighted, pct_eye_pain_weighted, pct_chills_weighted, pct_anosmia_ageusia_weighted, pct_sore_throat_weighted, pct_cough_weighted, pct_chest_pain_weighted, pct_fever_weighted)	9
2302	0.202381	(pct_nausea_weighted, pct_eye_pain_weighted, pct_chills_weighted, pct_anosmia_ageusia_weighted, pct_sore_throat_weighted, pct_aches_muscle_pain_weighted, pct_chest_pain_weighted, pct_difficulty_breathing_weighted, pct_fever_weighted)	9
2303	0.204193	(pct_nausea_weighted, pct_headache_weighted, pct_eye_pain_weighted, pct_chills_weighted, pct_anosmia_ageusia_weighted, pct_sore_throat_weighted, pct_chest_pain_weighted, pct_difficulty_breathing_weighted, pct_fever_weighted)	9
2304	0.205745	(pct_nausea_weighted, pct_headache_weighted, pct_eye_pain_weighted, pct_chills_weighted, pct_anosmia_ageusia_weighted, pct_sore_throat_weighted, pct_aches_muscle_pain_weighted, pct_chest_pain_weighted, pct_fever_weighted)	9

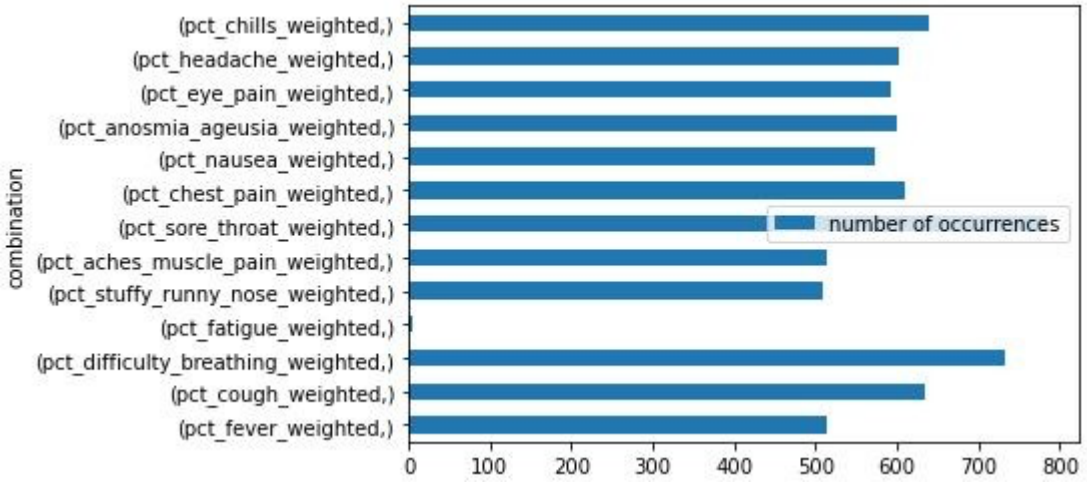
Q3b

Symptom combinations that appear in at least 60% of all records and have high total number of cases in percentage of population

	support	itemsets	length	percentage_of_high_population	total_frequent
2	0.499224	(pct_difficulty_breathing_weighted)	1	0.606013	1929
4	0.499741	(pct_stuffy_runny_nose_weighted)	1	0.656137	1931
16	0.270186	(pct_stuffy_runny_nose_weighted, pct_fever_weighted)	2	0.607280	1044
25	0.361801	(pct_difficulty_breathing_weighted, pct_cough_weighted)	2	0.665236	1398
27	0.386646	(pct_cough_weighted, pct_stuffy_runny_nose_weighted)	2	0.682731	1494
...	...	...	...	...	...
2300	0.216615	(pct_nausea_weighted, pct_headache_weighted, pct_eye_pain_weighted, pct_chills_weighted, pct_anosmia_ageusia_weighted, pct_sore_throat_weighted, pct_aches_muscle_pain_weighted, pct_chest_pain_weighted)	8	0.612903	837
2301	0.200052	(pct_nausea_weighted, pct_headache_weighted, pct_eye_pain_weighted, pct_chills_weighted, pct_anosmia_ageusia_weighted, pct_sore_throat_weighted, pct_cough_weighted, pct_chest_pain_weighted, pct_fever_weighted)	9	0.618370	773
2302	0.202381	(pct_nausea_weighted, pct_eye_pain_weighted, pct_chills_weighted, pct_anosmia_ageusia_weighted, pct_sore_throat_weighted, pct_aches_muscle_pain_weighted, pct_chest_pain_weighted, pct_difficulty_breathing_weighted, pct_fever_weighted)	9	0.624041	782
2303	0.204193	(pct_nausea_weighted, pct_headache_weighted, pct_eye_pain_weighted, pct_chills_weighted, pct_anosmia_ageusia_weighted, pct_sore_throat_weighted, pct_chest_pain_weighted, pct_difficulty_breathing_weighted, pct_fever_weighted)	9	0.626109	789
2304	0.205745	(pct_nausea_weighted, pct_headache_weighted, pct_eye_pain_weighted, pct_chills_weighted, pct_anosmia_ageusia_weighted, pct_sore_throat_weighted, pct_aches_muscle_pain_weighted, pct_chest_pain_weighted, pct_fever_weighted)	9	0.602516	795

Q3c

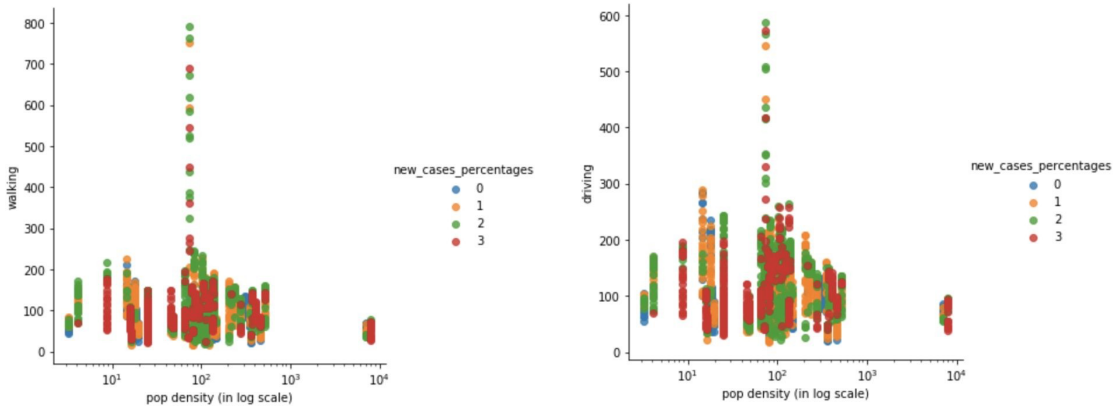
Histogram for the number of symptom combinations each symptom:

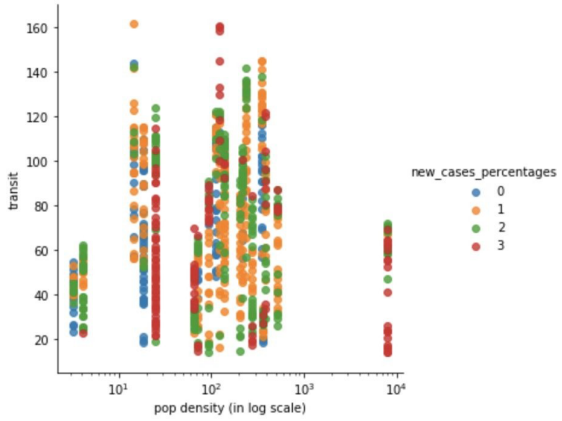


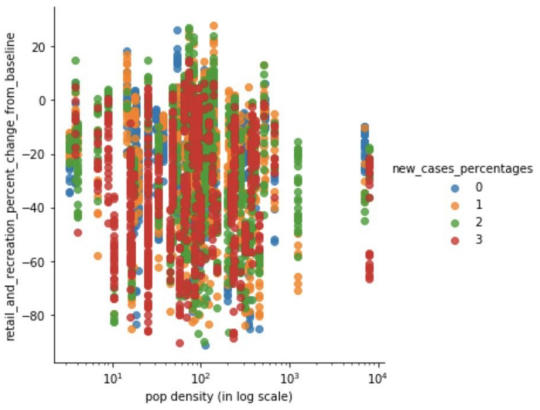
Q4a

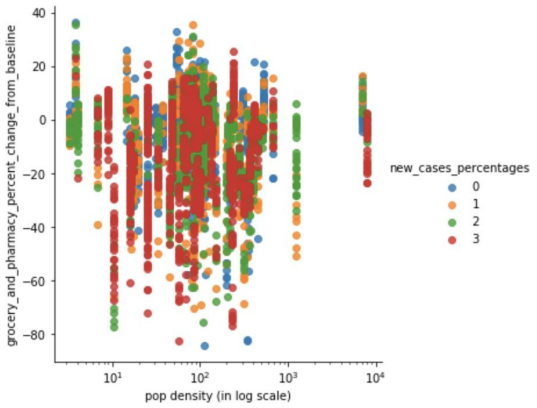
Attributes in Mobility data:

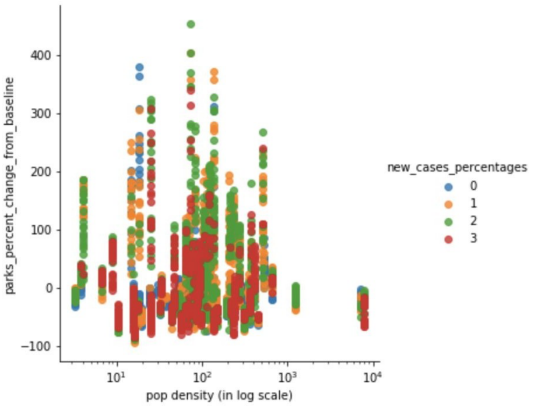
driving, walking, transit, retail_and_recreation_percent_change_from_baseline, grocery_and_pharmacy_percent_change_from_baseline, parks_percent_change_from_baseline, transit_stations_percent_change_from_baseline, workplaces_percent_change_from_baseline, residential_percent_change_from_baseline

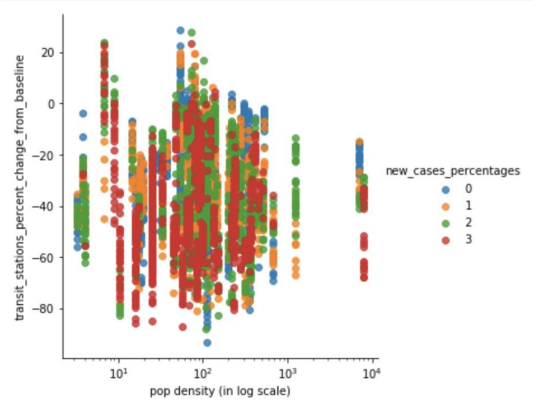


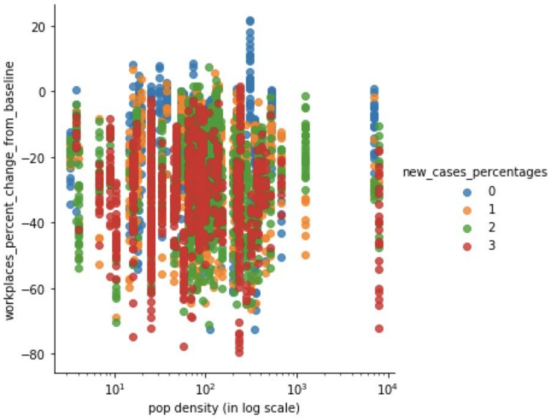

transit
pop density (in log scale)
new_cases_percentages
0
1
2
3

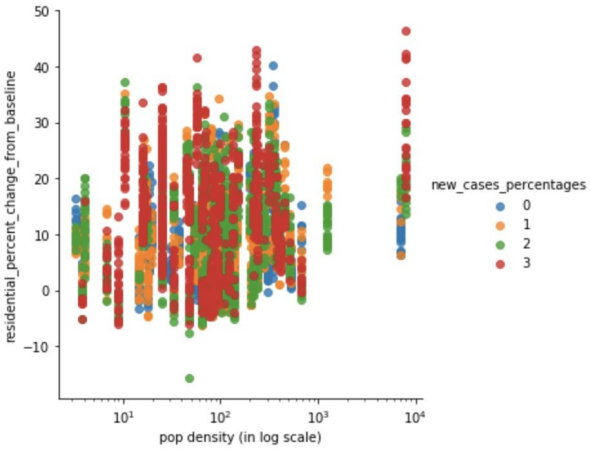

retail_and_recreation_percent_change_from_baseline
pop density (in log scale)
new_cases_percentages
0
1
2
3


grocery_and_pharmacy_percent_change_from_baseline
pop density (in log scale)
new_cases_percentages
0
1
2
3


parks_percent_change_from_baseline
pop density (in log scale)
new_cases_percentages
0
1
2
3


transit_stations_percent_change_from_baseline
pop density (in log scale)
new_cases_percentages
0
1
2
3


workplaces_percent_change_from_baseline
pop density (in log scale)
new_cases_percentages
0
1
2
3



Q4b

Quantiles of pop density:

Quantile	Pop density (people per sq km of land area)
0.0	3.24787091
0.2	30.59578423
0.4	77.47085096
0.6	107.9814868
0.8	235.25351426000185
1.0	7952.998418000001

Top 5 attributes that are most correlated with new cases percentages with quantiles 0.0 to 0.2

```
retail_and_recreation_percent_change_from_baseline   -0.399610
workplaces_percent_change_from_baseline              -0.399349
grocery_and_pharmacy_percent_change_from_baseline    -0.339602
residential_percent_change_from_baseline              0.316531
driving                                              -0.216457
Name: new_cases_percentages, dtype: float64
```

Top 5 attributes that are most correlated with new cases percentages with quantiles 0.2 to 0.4

```
workplaces_percent_change_from_baseline              -0.476512
transit_stations_percent_change_from_baseline        -0.355472
retail_and_recreation_percent_change_from_baseline   -0.314820
grocery_and_pharmacy_percent_change_from_baseline    -0.299190
residential_percent_change_from_baseline              0.273031
Name: new_cases_percentages, dtype: float64
```

Top 5 attributes that are most correlated with new cases percentages with quantiles 0.4 to 0.6

```
driving                                              0.391080
walking                                              0.283278
workplaces_percent_change_from_baseline             -0.254134
transit                                              0.234568
transit_stations_percent_change_from_baseline       -0.081535
Name: new_cases_percentages, dtype: float64
```

Top 5 attributes that are most correlated with new cases percentages with quantiles 0.6 to 0.8

```
transit                                              0.292131
residential_percent_change_from_baseline             0.281994
workplaces_percent_change_from_baseline             -0.279136
grocery_and_pharmacy_percent_change_from_baseline    -0.263481
retail_and_recreation_percent_change_from_baseline   -0.238293
Name: new_cases_percentages, dtype: float64
```

Top 5 attributes that are most correlated with new cases percentages with quantiles 0.8 to 1.0

```
transit_stations_percent_change_from_baseline        -0.359921
workplaces_percent_change_from_baseline              -0.357643
residential_percent_change_from_baseline              0.292798
transit                                              -0.277342
retail_and_recreation_percent_change_from_baseline   -0.207211
Name: new_cases_percentages, dtype: float64
```

Q5 - Additional analysis

Suggestion 1: Increase implementation of H2_Testing policy_1 in country group 1 / developing countries

From Q2 we can see the policy combinations that consist of {H1_Public information campaigns_2, H3_Contact tracing_1, C2_Workplace closing_2, C1_School closing_3, H2_Testing policy_1} are effective for countries in cluster 1 (Bolivia, Guatemala, Lao People's Democratic Republic, Peru, Cambodia, Myanmar, Nicaragua, Nepal, Senegal, Mali).

We then further investigate the "support" and "lift" value for effective policies in this country group using association rules. In particular, we extracted the policy combinations that have a lift value smaller than 1 to see what improvement can be made. The results are shown below.

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction	antecedent_len	consequents_len
21	(C1_School closing_3)	(H2_Testing policy_1)	0.755700	0.57329	0.394137	0.521552	0.909752	-0.039099	0.891863	1	1
44	(C2_Workplace closing_2)	(H2_Testing policy_1)	0.583062	0.57329	0.322476	0.553073	0.964735	-0.011788	0.954764	1	1
149	(H1_Public information campaigns_2)	(H2_Testing policy_1)	0.986971	0.57329	0.560261	0.567657	0.990174	-0.005560	0.986971	1	1

We can see that all antecedents' support are greater than H2_Testing policy_1's support. The "support" of H2_Testing policy_1 limited the confidence of {effective policies → H2_Testing policy_1}. It then capped the "lift" value. Thus, increasing the use of H2_Testing policy_1 can effectively increase the proportion of effective policy combination usage within this country group. This can be proved by the following "lift" equation.

$$lift(policy \rightarrow H2\_1) = \frac{Support(policy \cup H2\_1)}{Support(policy)\ Support(H2\_1)}$$

When we increase the number of H2_Testing policy_1, the increase in the numerator will be larger than the denominator. This implies implementing this policy will increase the "lift" value. When the likeliness of the simultaneous occurrence of effective combination increases, this ultimately helps to stop the spread of the pandemic.

Such policy adjustment might also be effective in developing countries. We can see that countries inside country group 1 have the lowest GDP and the least supporting facilities in water and sanitation service. These parameters are closely related to the country's economic health and development level. With all the parameters in "country_data" supporting cluster 1's inferiority relative to the other two clusters. We classify cluster 1 as a cluster that contains mostly developing countries. Countries that shared similar economic status and development levels with country group 1 could also consider the implementation of H2_Testing policy_1 to stop the pandemic.

Suggestion 2: Improving Immigration screening

An effective way to find out those suspicious patients who are likely to be infected with COVID-19 would be our top priority to combat the pandemic. The faster we could differentiate the infected patients from the community, the more effective we can control the spread of the virus. The current screening method is based on a few symptoms, such as fever, cough, which are also common in patients who suffer from the ordinary cold. After considering the common symptoms that appear in COVID-19 patients worldwide, apart from the most typical symptoms, like difficulty in breathing and cough, we found two other frequently occurring symptoms to help differentiate COVID-19 patients from the ordinary cold patients, which are anosmia ageusia and eye pain as shown below. Therefore, there are two recommendations with the finding.

	combination	number of occurrences
6	(pct_sore_throat_weighted,)	789.0
2	(pct_difficulty_breathing_weighted,)	742.0
1	(pct_cough_weighted,)	647.0
12	(pct_chills_weighted,)	645.0
7	(pct_chest_pain_weighted,)	616.0
11	(pct_headache_weighted,)	610.0
9	(pct_anosmia_ageusia_weighted,)	609.0
10	(pct_eye_pain_weighted,)	599.0
8	(pct_nausea_weighted,)	583.0
0	(pct_fever_weighted,)	519.0
4	(pct_stuffy_runny_nose_weighted,)	516.0
5	(pct_aches_muscle_pain_weighted,)	514.0
3	(pct_fatigue_weighted,)	5.0

Based on the finding, we could strengthen the effectiveness of screening during the immigration procedure. At the immigration counter, apart from the standard questions on the common respiratory symptoms that are using now, questions regarding whether the passengers have anosmia ageusia or eye pain should be asked. This could help remind people about their health situation if they have focused only on respiratory symptoms and overlooked the others. And this can raise the effectiveness of screening during the immigration procedure.

Suggestion 3: Prioritize patient in community check

Other than that, the finding can also help promote the efficiency in the community testing center in Hong Kong. For now, the daily capacity of the community testing in Hong Kong is around 80000 tests. Hindered by the logistic and collection arrangement, it requires 2 to 3 days for a test to be completed. There will be a potential risk for infected patients spreading the virus in the community during the time gap. Since we know that anosmia ageusia and eye pain could be a key criterion to differentiate ordinary cold from COVID-19. We can add a step in the community testing center to ask if the testees have symptoms of anosmia ageusia or eye pain. Those with symptoms should be prioritized in the testing procedure. This could help raise the efficiency of the community testing scheme

Suggestion 4: Increase implementation of C3_Cancel public events_2 in country group 3

From Q2 the policy combinations of {E1_Income support_2, H1_Public information campaigns_2, H3_Contact tracing_2} are effective for countries in cluster 3 (Italy, Canada, Austria, Czech Republic, Malaysia, New Zealand, United Kingdom, Slovakia, Kuwait, Poland, Switzerland, Greece, Finland, Portugal, Sweden, Norway, Germany, Hungary, Slovenia, Saudi Arabia, Australia, Ireland, France, United Arab Emirates, Spain, Denmark)

We further investigate the "support" and "lift" value for effective policies in this country group using association rules. We extracted the policy combinations with lift value smaller than 1 to study what improvement can be made. The results are shown below.

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction	antecedent_len	consequents_len
15	(E1_Income support_2)	(C3_Cancel public events_2)	0.686992	0.636179	0.376016	0.547337	0.860351	-0.061033	0.803736	1	1
96	(E1_Income support_2, H1_Public information campaigns_2)	(C3_Cancel public events_2)	0.686992	0.636179	0.376016	0.547337	0.860351	-0.061033	0.803736	2	1
108	(H3_Contact tracing_2, H1_Public information campaigns_2)	(C3_Cancel public events_2)	0.730691	0.636179	0.434959	0.595271	0.935698	-0.029891	0.898926	2	1
21	(H3_Contact tracing_2)	(C3_Cancel public events_2)	0.730691	0.636179	0.434959	0.595271	0.935698	-0.029891	0.898926	1	1

We can see that all antecedents' support are greater than C3_Cancel public events_2's support. The "support" of C3_Cancel public events_2 limited the confidence of {effective policies -> C3_Cancel public events_2}. It capped the "lift" value. As a result, increasing the use of C3_Cancel public events_2 policy can increase the proportion of effective policy combination usage effectively within this country group. The following "lift equation" can prove this fact.

$$\text{Lift}(\text{policy} -> \text{C3\_2}) = \frac{\text{Support}(\text{policy U C3\_2})}{Support(policy)Support(C3\_2)}$$

When we increase the number of C3_Cancel public events_2, the increase in the numerator will be greater than the denominator. This implies that the implementation of this policy will increase the “lift” value. When the likeliness of the simultaneous occurrence of effective combination increases, this ultimately helps to stop the spread of the pandemic.

This policy adjustment might be effective in developed countries. Most of the countries in this group are developed countries. In conclusion, the implementation of C3_Cancel public events_2 policy should be considered in developed countries so that the effectiveness of stopping the spread of pandemic can be increased.

Q6 - Model Training

1. Import libraries
2. Import Data
3. Data Exploration
4. Model Criterion Selection
    a. Data Pre-processing
        i. Standardization and Principal Component Analysis
    b. Model Selection
        i. Decision Tree
        ii. Random Forest
        iii. XgBoost
        iv. KNN/LMKNN
    c. Parameter Selection
5. Model Training
6. Model Validation

6.1 Step 1 - Import libraries

```
import pandas as pd
pd.set_option('display.max_columns',300)
pd.set_option('display.max_colwidth',300)
pd.set_option('display.max_rows',300)
import numpy as np
import warnings
warnings.filterwarnings('ignore')
import matplotlib.pyplot as plt
import seaborn as sns
import scipy

#Data Cleansing Package
from sklearn.experimental import enable_iterative_imputer
from sklearn.impute import IterativeImputer
from sklearn.preprocessing import StandardScaler
from sklearn import decomposition
from sklearn.model_selection import train_test_split

#Model Selection
from sklearn import model_selection
from sklearn.model_selection import StratifiedKFold
from sklearn.pipeline import Pipeline
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import RandomizedSearchCV


#Models
from sklearn import tree
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier
from sklearn.neighbors import KNeighborsClassifier

#Model Validation
from sklearn.model_selection import cross_validate
from sklearn.metrics import classification_report
```

For dataframe building and Array Processing

For graphs visualization for dataset

For filling null data

For data cleansing and feature engineering (standardization and PCA)

Package for splitting data into train set and validate set in the model validation stage

For model criterion selection, including cross validation, pipeline preparation and grid search validation

Importing decision tree model, random forest classifier model and Xgboost model

Implementing the k-nearest neighbours vote

Evaluate metric(s) by cross-validation

Build a text report showing the main classification metrics.

6.2 Step 2 - Import Data

In this stage, data will be imported. To have a brief understanding of the data, we checked for the data size, number of null values in the dataset and the classifiers' distribution.

a. Importing the data and observe shape

```
data =  pd.read_csv('covid_train.csv')
print (data.shape)
data.head()
```

```
(3864, 92)
```

There are 3864 rows and 92 columns. Since there's only around four thousands, we can conclude that the dataset is a small dataset.

b. Drop irrelavent columns

```
data = data.drop(columns=['total_cases'])
```

```
data = data.drop(columns=['country_name','country_code'])
```

- Drop the column of 'total_cases' as the testset will not provide this column
- Drop the column of 'country_name' and 'country_code' as the country name and country code are not a valid predictor for the model building.

c. Check for missing data

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

```
11373
```

```
data.isnull().sum().sort_values(ascending = False)
```

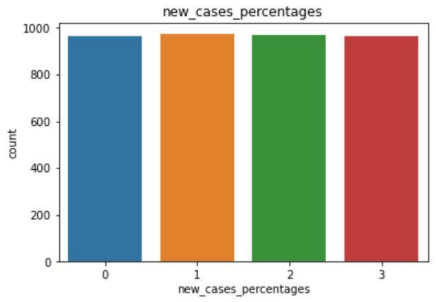
```
transit                                                3015
walking                                                1893
driving                                                1893
safe_san                                               1357
safe_water                                             1255
transit_stations_percent_change_from_baseline           335
parks_percent_change_from_baseline                      331
grocery_and_pharmacy_percent_change_from_baseline       321
retail_and_recreation_percent_change_from_baseline      317
residential_percent_change_from_baseline                306
workplaces_percent_change_from_baseline                 306
pop_density                                              36
ContainmentHealthIndex                                    3
GovernmentResponseIndex                                   3
EconomicSupportIndex                                      2
```

There are a total of 11373 null values across the dataset. The 15 listed columns above are the columns that include null value.

d. Checking for classifiers' distribution

```
sns.countplot('new_cases_percentages', data=data)
plt.title('new_cases_percentages')
```

```
Text(0.5, 1.0, 'new_cases_percentages')
```




```
data['new_cases_percentages'].value_counts()
```

```
1    972
2    967
3    963
0    962
```

There are 962 "0", 972 "1", 967 "2" and 963 "3" in the classifier's column "new_cases_percentages". The data can be regarded as a balanced data.

So, using accuracy is a suitable way to evaluate the performance of our models.

Step 2 conclusion:

- In the data set, there's 3864 attributes and 92records. 11373 missing data is found across 15 attributes. For the classifiers, we have 962 "0", 972 "1", 967 "2" and 963 "3".
- For the null value, we will use imputer to fill in the missing value for model training.
- As the data set is not large enough. We will adopt cross validation (5-fold) to do testing for the data to search for the best data pre-processing and model's construction parameters.

6.3 Step 3 - Feature engineering & handling missing values

Our first step after a brief data exploration is to handle the missing value and do the data segmentation. We followed two objectives when handling the data:

1. No overfitting of data
2. The fill-in value should show relevance to its attribute

We notice that the above two objectives might contradict thus we need to strike a balance between the two. We started by exploring the attributes correlation to evaluate appropriate feature engineering methods and identify the most suitable way to fill the missing data.

a. Correlation investigation - feature engineering methods

Here we want to evaluate whether it is valid to combine attributes into subgroups when before entering the model building process. In the following, we divided attributes into subsets and evaluated their interdependence.

```
def get_top_correlations(df, n):
    corr = df.corr().abs().unstack()
    pairs_to_drop = set()
    cols = df.columns
    for i in range(0, df.shape[1]):
        for j in range(0, i+1):
            pairs_to_drop.add((cols[i], cols[j]))
    corr = corr.drop(labels=pairs_to_drop).sort_values(ascending=False)
    return corr[0:n]
```

```
for dataset in dataset_col:
    print("Top Correlations")
    print(get_top_correlations(data[dataset], data[dataset].shape[1]))
    print()
```

```
Top Correlations
basic_water  basic_san     0.868939
safe_water   basic_san     0.808403
basic_water  safe_water    0.762894
safe_water   safe_san      0.732144
GDP          safe_san      0.728028
             safe_water    0.696726
basic_water  safe_san      0.586368
dtype: float64

Top Correlations
C3_Cancel public events_1               C3_Cancel public events_2                0.679764
C1_School closing_2                     C1_School closing_3                      0.666302
C8_International travel controls_3      C8_International travel controls_4       0.642002
C4_Restrictions on gatherings_3         C4_Restrictions on gatherings_4          0.567669
C2_Workplace closing_1                  C2_Workplace closing_2                   0.508520
C1_School closing_3                     C3_Cancel public events_2                0.506805
C7_Restrictions on internal movement_1  C7_Restrictions on internal movement_2   0.499181
C6_Stay at home requirements_2          C7_Restrictions on internal movement_2   0.490836
C6_Stay at home requirements_1          C6_Stay at home requirements_2           0.483970
C3_Cancel public events_2               C7_Restrictions on internal movement_2   0.474566
C1_School closing_3                     C7_Restrictions on internal movement_2   0.463277
C5_Close public transport_2             C7_Restrictions on internal movement_2   0.451709
                                        C6_Stay at home requirements_2           0.429199
C1_School closing_3                     C5_Close public transport_2              0.426774
C2_Workplace closing_2                  C2_Workplace closing_3                   0.423595
C3_Cancel public events_2               C4_Restrictions on gatherings_4          0.417000
C1_School closing_3                     C8_International travel controls_4       0.412824
C4_Restrictions on gatherings_4         C5_Close public transport_2              0.412246
C5_Close public transport_1             C5_Close public transport_2              0.409268
C4_Restrictions on gatherings_4         C7_Restrictions on internal movement_2   0.389640
C2_Workplace closing_1                  C3_Cancel public events_1                0.388233
C1_School closing_3                     C6_Stay at home requirements_2           0.382222
C3_Cancel public events_2               C6_Stay at home requirements_2           0.375452
dtype: float64
```

```
Top Correlations
C3_Cancel public events_1               C3_Cancel public events_2                0.679764
C1_School closing_2                     C1_School closing_3                      0.666302
C8_International travel controls_3      C8_International travel controls_4       0.642002
C4_Restrictions on gatherings_3         C4_Restrictions on gatherings_4          0.567669
C2_Workplace closing_1                  C2_Workplace closing_2                   0.508520
C1_School closing_3                     C3_Cancel public events_2                0.506805
C7_Restrictions on internal movement_1  C7_Restrictions on internal movement_2   0.499181
C6_Stay at home requirements_2          C7_Restrictions on internal movement_2   0.490836
C6_Stay at home requirements_1          C6_Stay at home requirements_2           0.483970
C3_Cancel public events_2               C7_Restrictions on internal movement_2   0.474566
C1_School closing_3                     C7_Restrictions on internal movement_2   0.463277
C5_Close public transport_2             C7_Restrictions on internal movement_2   0.451709
                                        C6_Stay at home requirements_2           0.429199
C1_School closing_3                     C5_Close public transport_2              0.426774
C2_Workplace closing_2                  C2_Workplace closing_3                   0.423595
C3_Cancel public events_2               C4_Restrictions on gatherings_4          0.417000
C1_School closing_3                     C8_International travel controls_4       0.412824
C4_Restrictions on gatherings_4         C5_Close public transport_2              0.412246
C5_Close public transport_1             C5_Close public transport_2              0.409268
C4_Restrictions on gatherings_4         C7_Restrictions on internal movement_2   0.389640
C2_Workplace closing_1                  C3_Cancel public events_1                0.388233
C1_School closing_3                     C6_Stay at home requirements_2           0.382222
C3_Cancel public events_2               C6_Stay at home requirements_2           0.375452
dtype: float64
```

```
Top Correlations
E2_Debt/contract relief_1  E2_Debt/contract relief_2    0.670285
E1_Income support_1        E1_Income support_2          0.633168
                           E2_Debt/contract relief_1    0.095534
                           E2_Debt/contract relief_2    0.069526
dtype: float64
```

```
Top Correlations
H1_Public information campaigns_1  H1_Public information campaigns_2   0.973012
H3_Contact tracing_1               H3_Contact tracing_2                0.850471
H2_Testing policy_1                H2_Testing policy_2                 0.581813
H2_Testing policy_2                H2_Testing policy_3                 0.505627
H6_Facial Coverings_2              H6_Facial Coverings_3               0.354071
H2_Testing policy_1                H2_Testing policy_3                 0.345861
H6_Facial Coverings_3              H6_Facial Coverings_4               0.331752
H6_Facial Coverings_2              H6_Facial Coverings_4               0.218246
H3_Contact tracing_2               H6_Facial Coverings_2               0.216095
H6_Facial Coverings_1              H6_Facial Coverings_3               0.203473
H3_Contact tracing_1               H6_Facial Coverings_2               0.176171
dtype: float64
```

Please refer to the notebook for the full version of the correlation analysis

Analysis on correlation:

In the correlation analysis of different subset of the training data, we found that in Policies data, including “Containment and closure policies”, “Economic policies” and “Health system policies”, the attributes which represent different ordinal scale position of the policies are always highly correlated with each other. For example, H1_Public information campaigns_1 and H1_Public information campaigns_2 are having a high correlation of 0.9730. This also implies the similarity in attribute patterns, and provided us with an option of combining attributes into groups to reduce the complexity of handling data in the latter part.

In the policy data, the extent of enacting the policy is in ordinal scale, with the higher value as tighter policies. In order to reduce the dimensionality and multilinearity of the dataset to prevent overfitting when handling missing value, we would combine the attribute of different ordinal scales of each policy in policilies data.

```
data['C1_School closing_1'] = data['C1_School closing_1']*1
data['C1_School closing_2'] = data['C1_School closing_2']*2
data['C1_School closing_3'] = data['C1_School closing_3']*3
data['C1_School closing'] = data['C1_School closing_1'] + data['C1_School closing_2']+ data['C1_School closing_3']


data['C2_Workplace closing_1'] = data['C2_Workplace closing_1']*1
data['C2_Workplace closing_2'] = data['C2_Workplace closing_2']*2
data['C2_Workplace closing_3'] = data['C2_Workplace closing_3']*3
data['C2_Workplace closing'] = data['C2_Workplace closing_1'] + data['C2_Workplace closing_2']+ data['C2_Workplace closing_3']


data['C3_Cancel public events_1'] = data['C3_Cancel public events_1']*1
data['C3_Cancel public events_2'] = data['C3_Cancel public events_2']*2
data['C3_Cancel public events'] = data['C3_Cancel public events_1'] + data['C3_Cancel public events_2']


data['C4_Restrictions on gatherings_1'] = data['C4_Restrictions on gatherings_1']*1
data['C4_Restrictions on gatherings_2'] = data['C4_Restrictions on gatherings_2']*2
data['C4_Restrictions on gatherings_3'] = data['C4_Restrictions on gatherings_3']*3
data['C4_Restrictions on gatherings_4'] = data['C4_Restrictions on gatherings_4']*4
data['C4_Restrictions on gatherings'] = data['C4_Restrictions on gatherings_1']+ data['C4_Restrictions on gatherings_2']+data['C4


data['C5_Close public transport_1'] = data['C5_Close public transport_1']*1
data['C5_Close public transport_2'] = data['C5_Close public transport_2']*2
data['C5_Close public transport'] = data['C5_Close public transport_1'] + data['C5_Close public transport_2']



data['C6_Stay at home requirements_1'] = data['C6_Stay at home requirements_1']*1
data['C6_Stay at home requirements_2'] = data['C6_Stay at home requirements_2']*2
data['C6_Stay at home requirements_3'] = data['C6_Stay at home requirements_3']*3
data['C6_Stay at home requirements'] = data['C6_Stay at home requirements_1'] +data['C6_Stay at home requirements_2']+ data['C6_S



data['C7_Restrictions on internal movement_1'] = data['C7_Restrictions on internal movement_1']*1
data['C7_Restrictions on internal movement_2'] = data['C7_Restrictions on internal movement_2']*2
data['C7_Restrictions on internal movement'] = data['C7_Restrictions on internal movement_1'] + data['C7_Restrictions on internal
```

Please refer to the notebook for the full version of the code.

b. Correlation investigation - Suitability of IterativeImputer

We would like to use IterativeImputer to deal with the missing values. It imputes missing data by modeling each feature with missing values as a function of other features in a round-robin fashion. Compared to min-max imputer and constant value impiter, IterativeImputer can help to give a better estimation of the missing values' with other features in the dataset. In order to prevent outputting an overfitted / irrelevant imputed value, we again, would investigate the attribute correlation.

Analysis on IterativeImputer

We run the code again to retrieve the correlation of attributes in different subset data, and the correlation in different subsets of data against Countries Data and Related Indexes data. If attributes are correlated, it implies the feasibility in using IterativeImputer to generate relevant data to fill in the missing values.

```
Top Correlations
ContainmentHealthIndex  GovernmentResponseIndex    0.961966
                        StringencyIndex            0.953756
StringencyIndex         GovernmentResponseIndex    0.903673
basic_water             basic_san                  0.868939
safe_water              basic_san                  0.808403
                        basic_water                0.762894
                        safe_san                   0.732144
safe_san                GDP                        0.728028
safe_water              GDP                        0.696726
safe_san                basic_water                0.586368
GDP                     basic_san                  0.577430
safe_san                basic_san                  0.555405
GDP                     basic_water                0.534389
EconomicSupportIndex    basic_water                0.489548
                        GDP                        0.446310
safe_water              pop_total                  0.433707
                        EconomicSupportIndex       0.424987
EconomicSupportIndex    basic_san                  0.415669
safe_san                EconomicSupportIndex       0.405731
                        StringencyIndex            0.383160
```

In the above, we can conclude that the attribute in countries data and related index are correlated, IterativeImputer can be applied to help to impute the missing data

```
Top Correlations
ContainmentHealthIndex   GovernmentResponseIndex    0.961966
                         StringencyIndex            0.953756
driving                  walking                    0.905180
StringencyIndex          GovernmentResponseIndex    0.903673
basic_water              basic_san                  0.868939
safe_water               basic_san                  0.808403
walking                  transit                    0.788436
driving                  transit                    0.788169
safe_water               basic_water                0.762894
StringencyIndex          transit                    0.742559
safe_water               safe_san                   0.732144
safe_san                 GDP                        0.728028
safe_water               GDP                        0.696726
ContainmentHealthIndex   transit                    0.651260
GovernmentResponseIndex  transit                    0.627822
driving                  StringencyIndex            0.599119
safe_san                 basic_water                0.586368
GDP                      basic_san                  0.577430
safe_san                 basic_san                  0.555405
GDP                      basic_water                0.534389
dtype: float64
```

In the above, we can conclude that the attribute in Traveling data and countries data with related index data are correlated, IterativeImputer can be applied to help to impute the missing data

```
Top Correlations
ContainmentHealthIndex                          GovernmentResponseIndex                          0.961966
                                                StringencyIndex                                  0.953756
StringencyIndex                                 GovernmentResponseIndex                          0.903673
basic_water                                     basic_san                                        0.868939
retail_and_recreation_percent_change_from_baseline  residential_percent_change_from_baseline     0.863751
grocery_and_pharmacy_percent_change_from_baseline   retail_and_recreation_percent_change_from_baseline  0.841604
transit_stations_percent_change_from_baseline   retail_and_recreation_percent_change_from_baseline  0.833591
safe_water                                      basic_san                                        0.808403
transit_stations_percent_change_from_baseline   residential_percent_change_from_baseline         0.807970
grocery_and_pharmacy_percent_change_from_baseline   residential_percent_change_from_baseline     0.772384
                                                transit_stations_percent_change_from_baseline    0.765775
safe_water                                      basic_water                                      0.762894
                                                safe_san                                         0.732144
safe_san                                        GDP                                              0.728028
safe_water                                      GDP                                              0.696726
workplaces_percent_change_from_baseline         residential_percent_change_from_baseline         0.684381
StringencyIndex                                 retail_and_recreation_percent_change_from_baseline  0.671115
transit_stations_percent_change_from_baseline   workplaces_percent_change_from_baseline          0.664331
parks_percent_change_from_baseline              retail_and_recreation_percent_change_from_baseline  0.664212
StringencyIndex                                 residential_percent_change_from_baseline         0.642775
```

In the above, we can conclude that the attribute in activity data and countries data with related index data are correlated, IterativeImputer can be applied to help to impute the missing data.

Handling missing value

After proofing the feasibility in using IterativeImputer on the above correlation analysis we decided to handle the missing value as follow:

```
imp = IterativeImputer(random_state=0)

imp.fit(data[Countries_Data_col+Related_Indexes_col])
data[Countries_Data_col+Related_Indexes_col] = np.round(imp.transform(data[Countries_Data_col+Related_Indexes_col]))

imp.fit(data[Countries_Data_col+Related_Indexes_col+Traveling_col])
data[Countries_Data_col+Related_Indexes_col+Traveling_col] = np.round(imp.transform(data[Countries_Data_col+Related_Indexes_col+T

imp.fit(data[Countries_Data_col+Related_Indexes_col+activity_data])
data[Countries_Data_col+Related_Indexes_col+activity_data] = np.round(imp.transform(data[Countries_Data_col+Related_Indexes_col+a
```

Step 3 Conclusion:

- High correlation means two attributes are dependent on others. Having dependent attributes in the dataset for training will lead to overfitting and multicollinearity issues.
- As there are many fields that are highly correlated. Feature engineering on policies data are implemented in the stage.
- With the correlation of the data, IterativeImputer is applied to help to impute the missing data.

6.4 Model selection

There are in total two decisions we would like to make. First, we would like to evaluate whether additional preprocessing of data (eg: PCA) is beneficial to model performance. Second, we would like to identify the optimal classifier among decision tree, random forest and xgboost.

6.4a Step 4 - Model Selection (Additional processing of data)

From session 6.3, we have had insights on the dataset and proceeded feature engineering and filling missing values in the dataset. However, the predictors still show dependency and we would like to investigate whether further preprocessing of data could benefit the model's accuracy.

a. Standard scaler and PCA

Both are the choices to reduce the dimensionality and multilinearity of the predictors. However, PCA will eliminate the meaning of the attributes and sacrifice part of the variance of the dataset.

In this stage, we will first search for the best level of cumulative variance for the PCA for the model building. A comparison between a decision tree model ,random forest classifier and Xgboost will be conducted to search for the best model. The model accuracy of the best level of cumulative variance for the PCA will be compared with the model accuracy without conducting PCA.

For the searching of the best level of cumulative variance, grid search with pipeline will be used to standardise each trial to have fair comparison between parameters combinations.

b. 5-fold cross validation

StratifiedKFold will be used to preserve the percentage of samples for each class. 5-fold will also be applied to check overfitting. The random state will be set as "42" to ensure a controlled comparison between different parameters combinations. The common practice for cross validation is to choose a split among 3, 5, 10. I eliminated the 3-fold, as it is more ideal to be used when handling small dataset. For 10-fold cross validation, it seems to be a better option for handling our current large dataset. However, only 1 of the 10 splits will be the validation set. The validation set appears to be too insignificant in 10-kold options. I would like to maintain a higher proportion of data to be split as the validation set so as to make sure the accuracy reflected in the cross validation will be more convincing. Therefore, I decided to use 5-fold cross validation in which the validation set will take up 20% of the data.

Before the criterion selection, we are going to split off the attribute and classifier for steps in the section.

Spliting of data for model selection

```
X = data.drop(columns='new_cases_percentages')
y = data['new_cases_percentages']
```

6.4 b Step 4 - Model Selection

We will then evaluate the prediction performance using three different models and pick the ideal one. The following will be the supplementary information for the three classifiers.

Model 1: Decision Tree (Supplement information)

A decision tree is a decision support tool that uses a tree-like model of decisions and their possible consequences, including chance event outcomes, resource costs, and utility. It is one way to display an algorithm that only contains conditional control statements.

Decision trees are commonly used in operations research, specifically in decision analysis, to help identify a strategy most likely to reach a goal, but are also a popular tool in machine learning.

Model 2: Random Forest (Supplement information)

In machine learning, using multiple learning algorithms can obtain better predictive performance than single learning algorithms alone. The fundamental concept behind random forest is simple— the wisdom of crowds. Many uncorrelated decision trees operating as a committee for the prediction. It is assumed that a random forest will outperform any of the trees alone. After training, predictions can be made by taking the majority vote in the case of classification trees.

Model 3: Xgboost (Supplement information)

XGBoost stands for eXtreme Gradient Boosting. It is a decision-tree-based ensemble Machine Learning algorithm that uses a gradient boosting framework. Gradient boosting employs a gradient descent algorithm to minimize errors in sequential models. It builds the model in a stage-wise fashion, and generalizes them by allowing optimization of an arbitrary differentiable loss function. In prediction problems involving unstructured data (images, text, etc.) artificial neural networks tend to outperform other algorithms or frameworks. However, when it comes to small-to-medium structured/tabular data, decision tree-based algorithms are considered best-in-class right now. In particular, XGBoost holds the advantages of its fast execution speed and great model performance.

Model 4: KNN/LMKNN (Supplement information)

KNN(k nearest neighbors) is a simple non-parametric method for both regression and classification. It chooses k nearest points from input and labels the input with the highest number of votes among the k points. There are two key parts in this model: k and the similarity/distance function. In this report, k is chosen using grid search and we will use cosine similarity. The reason we use cosine similarity is that it could find the relationship/similarity between two data points/vectors even if they are far apart with each other. Since we have a very high dimensional input, we decide to use cosine similarity.

LMKNN(local mean based k-nearest neighbor) is an improved version of KNN, proposed in 1968 by Cover and Hart. Instead of choosing k nearest points from input, it chooses k nearest points from input for each class. If we have 4 classes, we would choose 4k points in total. Then, we find the mean of the k points for each class and compare the mean point with the input. The input will be labeled with the class with the shortest distance between the input and the mean of k points in that class.

Model 1: Decision Tree classifier

1. First, create objects for Standard Scaler, PCA and Decision Tree classifier.

   Then, create a pipeline and a dictionary of all the parameter options for grid search to standardise each trial and prevent data loss

   Note that "random_state" is set as 42 for the Decision Tree classifier to guarantee the same output is generated and fair comparison of models.

```
%%time
# Creating an standardscaler object
std_slc = StandardScaler()
# Creating a pca object
pca = decomposition.PCA()
# Creating a Decision Tree Classifier
dec_tree = tree.DecisionTreeClassifier()
    # Creating a pipeline of three steps. First, standardizing the data.
    # Second, tranforming the data with PCA.
    # Third, training a Random Forest Classifier on the data.
pipe = Pipeline(steps=[('std_slc', std_slc),
                       ('pca', pca),
                       ('dec_tree', dec_tree)])
    # Creating Parameter Space
n_components = [0.80, 0.81, 0.82, 0.83, 0.84, 0.85, 0.86, 0.87, 0.88, 0.89,
                0.9, 0.91, 0.92, 0.93, 0.94, 0.95, 0.96, 0.97,0.98, 0.99]
random_state = [42]

    # Creating a dictionary of all the parameter options
parameters = dict(pca__n_components=n_components,
                  dec_tree__random_state=random_state)

cv = StratifiedKFold(n_splits = 5)
```

2. Execute grid search cv with 5-fold to search for the best level of explained variance

```
%%time
    # Creating a grid search object
model = GridSearchCV(pipe, parameters,scoring= 'accuracy', cv=cv, n_jobs = -1, verbose = 30)
    # Fitting the grid search
model.fit(X, y)
```

3. Receive the best level of variance and score

```
%%time
    # Viewing The Best Parameters
print('Best Number Of Components:', model.best_estimator_.get_params()['pca__n_components'])
print(); print(model.best_estimator_.get_params()['pca'])
```

```
Best Number Of Components: 0.9

PCA(n_components=0.9)
Wall time: 0 ns
```

```
model.best_score_
```

```
0.6658885708731878
```

4. Now, perform the same analysis without PCA.

```
%%time

dec_tree = tree.DecisionTreeClassifier()

pipe = Pipeline(steps=[('dec_tree', dec_tree)])
    # Creating Parameter Space

random_state = [42]

    # Creating a dictionary of all the parameter options
parameters = dict(dec_tree__random_state=random_state)

cv = StratifiedKFold(n_splits = 5)
```

```
%%time
    # Creating a grid search object
model = GridSearchCV(pipe, parameters,scoring= 'accuracy', cv=cv, n_jobs = -1, verbose = 30)
    # Fitting the grid search
model.fit(X, y)
```

5. Receive the score

```
model.best_score_
```

```
0.7248945967866263
```

Model 2: Random forest classifier

1. First, create objects for Standard Scaler, PCA and Random Forest classifier.

   Then, create a pipeline and a dictionary of all the parameter options for grid search to standardise each trial and prevent data loss

   Note that "random_state" is set as 42 for the Decision Tree classifier to guarantee the same output is generated and fair comparison of models.

```
%%time
# Creating an standardscaler object
std_slc = StandardScaler()
# Creating a pca object
pca = decomposition.PCA()
# Creating a Random Forest Classifier
RandomForest = RandomForestClassifier()
    # Creating a pipeline of three steps
pipe = Pipeline(steps=[('std_slc', std_slc),
                       ('pca', pca),
                       ('RandomForest', RandomForest)])

n_components = [0.80, 0.81, 0.82, 0.83, 0.84, 0.85, 0.86, 0.87, 0.88, 0.89,
                0.9, 0.91, 0.92, 0.93, 0.94, 0.95, 0.96, 0.97,0.98, 0.99]
random_state = [42]

    # Creating a dictionary of all the parameter options
parameters = dict(pca__n_components=n_components,
                  RandomForest__random_state=random_state)

cv = StratifiedKFold(n_splits = 5, random_state=42, shuffle=True)
```

2. Execute grid search cv with 5-fold to search for the best level of explained variance

```
%%time
    # Creating a grid search object
model = GridSearchCV(pipe, parameters,scoring= 'accuracy', cv=cv, n_jobs = -1, verbose = 30)
    # Fitting the grid search
model.fit(X, y)
```

3. Receive the best level of variance and score

```
%%time
    # Viewing The Best Parameters
print('Best Number Of Components:', model.best_estimator_.get_params()['pca__n_components'])
print(); print(model.best_estimator_.get_params()['pca'])
```

```
Best Number Of Components: 0.94

PCA(n_components=0.94)
Wall time: 999 µs
```

```
model.best_score_
```

```
0.7976218085783804
```

4. Now, perform the same analysis without PCA.

```
%%time

RandomForest = RandomForestClassifier()

pipe = Pipeline(steps=[('RandomForest', RandomForest)])

random_state = [42]

    # Creating a dictionary of all the parameter options
parameters = dict(RandomForest__random_state=random_state)

cv = StratifiedKFold(n_splits = 5, random_state=42, shuffle=True)
```

5. Execute grid search cv with 5-fold to search for the best level of explained variance

```
%%time
    # Creating a grid search object
model = GridSearchCV(pipe, parameters,scoring= 'accuracy', cv=cv, n_jobs = -1, verbose = 30)
    # Fitting the grid search
model.fit(X, y)
```

6. Receive the score

```
model.best_score_
```

```
0.8131480873254731
```

Model 3: Xgboost

1. First, create objects for Standard Scaler, PCA and Xgboost.

   Then, create a pipeline and a dictionary of all the parameter options for grid search to standardise each trial and prevent data loss

   Note that "random_state" is set as 42 for the Decision Tree classifier to guarantee the same output is generated and fair comparison of models.

```
%%time
# # Creating an standardscaler object
std_slc = StandardScaler()
# # Creating a pca object
pca = decomposition.PCA()
# Creating a Random Forest Classifier
xgb = XGBClassifier()
    # Creating a pipeline of three steps
pipe = Pipeline(steps=[('std_slc', std_slc),
                       ('pca', pca),
                       ('xgb', xgb)])

n_components = [0.80, 0.81, 0.82, 0.83, 0.84, 0.85, 0.86, 0.87, 0.88, 0.89,
                0.9, 0.91, 0.92, 0.93, 0.94, 0.95, 0.96, 0.97,0.98, 0.99]
random_state = [42]

    # Creating a dictionary of all the parameter options
parameters = dict(pca__n_components=n_components,
                  xgb__random_state=random_state)

cv = StratifiedKFold(n_splits = 5, random_state=42, shuffle=True)
```

2. Execute grid search cv with 5-fold to search for the best level of explained variance

```
%%time
    # Creating a grid search object
model = GridSearchCV(pipe, parameters,scoring= 'accuracy', cv=cv, n_jobs = -1, verbose = 30)
    # Fitting the grid search
model.fit(X, y)
```

3. Receive the best level of variance and score

```
%%time
    # Viewing The Best Parameters
print('Best Number Of Components:', model.best_estimator_.get_params()['pca__n_components'])
print(); print(model.best_estimator_.get_params()['pca'])
```

```
Best Number Of Components: 0.93

PCA(n_components=0.93)
Wall time: 6.98 ms
```

```
model.best_score_
```

```
0.7885608188271254
```

4. Now, perform the same analysis without PCA.

```
%%time
# Creating a Random Forest Classifier
xgb = XGBClassifier()
    # Creating a pipeline of three steps
pipe = Pipeline(steps=[('xgb', xgb)])

random_state = [42]

    # Creating a dictionary of all the parameter options
parameters = dict(xgb__random_state=random_state)

cv = StratifiedKFold(n_splits = 5, random_state=42, shuffle=True)
```

5. Execute grid search cv with 5-fold to search for the best level of explained variance

```
%%time
    # Creating a grid search object
model = GridSearchCV(pipe, parameters,scoring= 'accuracy', cv=cv, n_jobs = -1, verbose = 30)
    # Fitting the grid search
model.fit(X, y)
```

6. Receive the score

```
model.best_score_
```

```
0.8131490927615307
```

Model 4: KNN/LMKNN Classifier

1. First, create objects for Standard Scaler, PCA and KNN classifier.

   Then, create a pipeline and a dictionary of all the parameter options for grid search to standardise each trial and prevent data loss

   Note that "random_state" is set as 42 for the Decision Tree classifier to guarantee the same output is generated and fair comparison of models.

```
%%time
# # Creating an standardscaler object
std_slc = StandardScaler()
# # Creating a pca object
pca = decomposition.PCA()
# Creating a Random Forest Classifier
knn = KNeighborsClassifier()
    # Creating a pipeline of three steps
pipe = Pipeline(steps=[('std_slc', std_slc),
                       ('pca', pca),
                       ('knn', knn)])

n_components = [0.80, 0.81, 0.82, 0.83, 0.84, 0.85, 0.86, 0.87, 0.88, 0.89,
                0.9, 0.91, 0.92, 0.93, 0.94, 0.95, 0.96, 0.97,0.98, 0.99]

leaf_size = list(range(1,25))
n_neighbors = list(range(1,15))
p=[1,2]
metric = ['cosine']

    # Creating a dictionary of all the parameter options
parameters = dict(pca__n_components=n_components,
                  knn__leaf_size=leaf_size,
                  knn__n_neighbors=n_neighbors,
                  knn__p=p,
                  knn__metric=metric)

cv = StratifiedKFold(n_splits = 5, random_state=42, shuffle=True)
```

2. Execute grid search cv with 5-fold to search for the best level of explained variance

```
%%time
    # Creating a grid search object
model = GridSearchCV(pipe, parameters,scoring= 'accuracy', cv=cv, n_jobs = -1, v
    # Fitting the grid search
model.fit(X, y)
```

3. Receive the best level of variance and score

```
%%time
    # Viewing The Best Parameters
print('Best Number Of Components:', model.best_estimator_.get_params()['pca__n_c
print(); print(model.best_estimator_.get_params()['pca'])
```

```
Best Number Of Components: 0.99

PCA(n_components=0.99)
Wall time: 2.5 ms
```

```
model.best_score_
```

```
0.7924434777362943
```

4. Now, perform the same analysis without PCA.

```
%%time

knn = KNeighborsClassifier()
    # Creating a pipeline of three steps
pipe = Pipeline(steps=[('knn', knn)])

leaf_size = list(range(1,25))
n_neighbors = list(range(1,15))
p=[1,2]
metric = ['cosine']

    # Creating a dictionary of all the parameter options
parameters = dict(knn__leaf_size=leaf_size,
                  knn__n_neighbors=n_neighbors,
                  knn__p=p,
                  knn__metric=metric)

cv = StratifiedKFold(n_splits = 5, random_state=42, shuffle=True)
```

5. Execute grid search cv with 5-fold to search for the best level of explained variance

```
%%time
    # Creating a grid search object
model = GridSearchCV(pipe, parameters,scoring= 'accuracy', cv=cv, n_jobs = -1, v
    # Fitting the grid search
model.fit(X, y)
```

6. Receive the score

```
model.best_score_
```

```
0.7158372936342492
```

7. In order to improve the performance of the KNN, we try to apply the local mean method.

   During the 'fit' process, we split the training sets into 4 classes.

```
def fit(self, x, y):
    classes = [[] for i in range(self.num_classes)]
    m = x.shape[0]
    for i in range(m):
        for j in range(self.num_classes):
            if y[i] == j:
                classes[j].append(x[i])
                break
    self.classes=classes
    return self
```

   Note that we define the distance function as follow:

```
def distance_cosine(self, x, y):
    return 1 - np.inner(x, y) / (scipy.linalg.norm(x) * scipy.linalg.norm(y))
```

   This is the cosine similarity function. Since we want to maintain the idea of 'closest distance', we use 1-cosine similarity. So, the higher the cosine similarity, the lower the value it gives, hence the 'shorter' the distance between two vectors.

   In the 'predict' process, we loop all the data points in the dataset. For each data point, we find k nearest points from the data point for each class. Then, find the mean of these data points and store it in mean_vectors. Iterating all the classes, we could find the mean vector with the smallest distance from the data point and label

the input with that class where the mean vector is representing.

```
def predict(self, x_test):
    pred = []
    for i in range(x_test.shape[0]):
        # find k nearest neighbors for each class
        min_dist = 1.1
        label = 0
        for j in range(self.num_classes):
            distances = []
            for l in range(len(self.classes[j])):
                dis = self.distance_cosine(x_test[i], self.classes[j][l])
                distances.append((dis, self.classes[j][l]))
            distances = sorted(distances, key=lambda x: x[0])
            distances = distances[:self.k]
            # Get jth mean vectors
            mean_vectors = distances[0][1]/self.k
            for l in range(1, self.k):
                 mean_vectors= mean_vectors+distances[l][1]/self.k
            # Check if distance is smaller than others
            dis = self.distance_cosine(x_test[i], mean_vectors)
            if dis < min_dist:
                min_dist = dis
                label = j
        pred.append(label)
    return pred
```

8. After performing grid search, the accuracy score obtain:

```
clf.best_score_
```

```
0.7153209109730848
```

Conclusion

We identified that using XGboost classifier without undergoing PCA process would attain the highest accuracy, thus will be the optimal model.

Model	Model best accuracy with PCA	**Model best accuracy without PCA**
Decision Tree Classifier	0.6658885708731878	0.7248945967866263
Random Forest Classifier	0.7976218085783804	0.8131480873254731
XGboost Classifier	0.7885608188271254	**0.8131490927615307**
KNN Classifier	0.7158372936342492	0.7924434777362943

6.5 Step 5 - ParameterSelection

After choosing the optimal classifier, we also have to identify the best parameter to pass into the model. There are many parameters that we can select in the Xgboost classifier model. Parameter tuning can help to raise the accuracy of the model. There is a wide range of selection for each parameter, and the combination of the possible parameters will be a large number. To save computing power and time, the parameter selection will first be started with random search cv to search for an approximate range of the best parameter. After that, detailed grid search cv will be done in rounds to narrow down the range of the parameters to figure out the best parameters.

Parameter for tuning

Parameter	**Explanation**	**Range**
booster	Booster to use in the model. gbtree and dart use tree based models while gblinear uses linear functions.	'gbtree' 'gblinear' 'dart'
learning_rate	Gradient boosting involves creating and adding trees to the model sequentially. New trees are created to correct the residual errors in the predictions from the existing sequence of trees. learning_rate is set to slow down the learning in the gradient boosting model which applies a weighting factor for the corrections by new trees when added to the model.	[0,1]
max_depth	The maximum depth of each tree in the random forest classifier model.	[0,∞]

n_estimators	The number of trees in the model.	[0,∞]
min_child_weight	Minimum sum of instance weight (hessian) needed in a child. If the tree partition step results in a leaf node with the sum of instance weight less than min_child_weight, then the building process will give up further partitioning. In a linear regression task, this simply corresponds to the minimum number of instances needed to be in each node. The larger min_child_weight is, the more conservative the algorithm will be.	[0,∞]
gamma	Minimum loss reduction required to make a further partition on a leaf node of the tree. The larger gamma is, the more conservative the algorithm will be.	[0,∞]
colsample_bytree	the subsample ratio of columns when constructing each tree. Subsampling occurs once for every tree constructed.	[0,1]

Parameter for fixing

parameter	**Explanation**	**Reason for fixing**
random_state = 42		
tree_method = auto	The tree construction algorithm used in XGBoost	To let the xgboost use heuristic to choose the fastest method based on data size. For our data size with around 4000 rows of data. Xgboost will consider it as a small data set. Thus, Exact greedy algorithm will be selected. Enumerates all split candidates for accurate results for small dataset.

6.5.1 Step 5.1 - Randomized Search CV to design for the range for detailed grid search

Parameter	Range
booster	'gbtree', 'gblinear', 'dart'
learning_rate	[0.05, 0.10, 0.15, 0.20, 0.25, 0.30]
max_depth	[10,15,20,25]
n_estimators	[100,500,1000]
min_child_weight	[1, 3, 5, 7]
random_state	[42]
gamma	[0.0, 0.1, 0.2 , 0.3, 0.4]
colsample_bytree	[0.3, 0.4, 0.5 , 0.7]

1. Setting parameter space

```
classifier=XGBClassifier()
cv = StratifiedKFold(n_splits = 5, random_state=42, shuffle=True)
```

```
random_search=RandomizedSearchCV(classifier,param_distributions=params,n_iter=50,scoring='accuracy',n_jobs=-1,cv=cv,verbose=50)
```

2. Running of the random search validation

```
from datetime import datetime
# Here we go
start_time = timer(None) # timing starts from this point for "start_time" variable
random_search.fit(X,y)
timer(start_time) # timing ends here for "start_time" variable
```

3. Getting the best parameter and the best score of the input parameters.

```
random_search.best_estimator_
```

```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=0.7, gamma=0.2, gpu_id=-1,
              importance_type='gain', interaction_constraints='',
              learning_rate=0.05, max_delta_step=0, max_depth=10,
              min_child_weight=1, missing=nan, monotone_constraints='()',
              n_estimators=100, n_jobs=0, num_parallel_tree=1,
              objective='multi:softprob', random_state=42, reg_alpha=0,
              reg_lambda=1, scale_pos_weight=None, subsample=1,
              tree_method='exact', validate_parameters=1, verbosity=None)
```

```
random_search.best_params_
```

```
{'random_state': 42,
 'n_estimators': 100,
 'min_child_weight': 1,
 'max_depth': 10,
 'learning_rate': 0.05,
 'gamma': 0.2,
 'colsample_bytree': 0.7,
 'booster': 'gbtree'}
```

In this round, we found that the best parameters are:

Parameter	**Range**
booster	'gbtree',
learning_rate	0.05
max_depth	10
n_estimators	100
min_child_weight	1
random_state	[42]
gamma	0.2
colsample_bytree	0.7

We will narrow the parameter for the model with a detailed grid search CV in the next stage.

6.5.2 Step 5.2 - Grid search CV to search for the best parameter for the model

First Round:

Parameter	Range
booster	'gbtree',
learning_rate	[0.04,0.05,0.06]
max_depth	[8,10,12]
n_estimators	[50,100,200]
min_child_weight	[1,2]
random_state	[42]
gamma	[0.1,0.2,0.3]
colsample_bytree	[0.6,0.7,0.8]

1. Setting parameter space

```
%%time
# Creating a XgBoost Classifier Object
xgb = XGBClassifier()
    # Creating a pipeline of three steps
pipe = Pipeline(steps=[('xgb', xgb)])

    # Creating Parameter Space
booster = ['gbtree']
learning_rate = [0.04,0.05,0.06]
max_depth = [8,10,12]
n_estimators = [50,100,200]
min_child_weight = [1,2]
random_state = [42]
gamma = [0.1,0.2,0.3]
colsample_bytree = [0.6,0.7,0.8]

    # Creating a dictionary of all the parameter options
parameters = dict(xgb__booster=booster,
                  xgb__learning_rate=learning_rate,
                  xgb__max_depth=max_depth,
                  xgb__n_estimators=n_estimators,
                  xgb__min_child_weight = min_child_weight,
                  xgb__random_state=random_state,
                  xgb__gamma= gamma,
                  xgb__colsample_bytree= colsample_bytree)

cv = StratifiedKFold(n_splits = 5, random_state=42, shuffle=True)
```

2. Running of the grid search validation

```
%%time
    # Conducting Parameter Optmization With Pipeline
    # Creating a grid search object
model = GridSearchCV(pipe, parameters,cv=cv, n_jobs = -1, verbose = 30)
    # Fitting the grid search
model.fit(X, y)
```

3. Getting the best parameter and the best score of the input parameters.

```
%%time
print('Best xgb__booster:', model.best_estimator_.get_params()['xgb__booster'])
print('Best xgb__learning_rate:', model.best_estimator_.get_params()['xgb__learning_rate'])
print('Best max_depth:', model.best_estimator_.get_params()['xgb__max_depth'])
print('Best n_estimators:', model.best_estimator_.get_params()['xgb__n_estimators'])
print('Best xgb__min_child_weight:', model.best_estimator_.get_params()['xgb__min_child_weight'])
print('Best xgb__gamma:', model.best_estimator_.get_params()['xgb__gamma'])
print('Best colsample_bytree:', model.best_estimator_.get_params()['xgb__colsample_bytree'])
print(); print(model.best_estimator_.get_params()['xgb'])
```

```
Best xgb__booster: gbtree
Best xgb__learning_rate: 0.04
Best max_depth: 12
Best n_estimators: 200
Best xgb__min_child_weight: 2
Best xgb__gamma: 0.2
Best colsample_bytree: 0.6

XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=0.6, gamma=0.2, gpu_id=-1,
              importance_type='gain', interaction_constraints='',
              learning_rate=0.04, max_delta_step=0, max_depth=12,
              min_child_weight=2, missing=nan, monotone_constraints='()',
              n_estimators=200, n_jobs=0, num_parallel_tree=1,
              objective='multi:softprob', random_state=42, reg_alpha=0,
              reg_lambda=1, scale_pos_weight=None, subsample=1,
              tree_method='exact', validate_parameters=1, verbosity=None)
Wall time: 20 ms
```

```
model.best_score_
```

```
0.8237564431694026
```

In this round, we found that the best parameters are :

Parameter	**Range**	
booster	'gbtree'	The accuracy best score from best_estimator is 0.8237.
learning_rate	0.04	
max_depth	12	
n_estimators	200	
min_child_weight	2	
random_state	[42]	
gamma	0.2	
colsample_bytree	0.6	

We will run through the above process for several rounds and each time we will narrow the testing parameter. The results of each round is shown below (with arrow → representing the best parameter):

Second Round:

Parameter	**Range**	
booster	'gbtree'	The accuracy best score from best_estimator is 0.8263.
learning_rate	[0.03,0.04,0.05] → 0.04	
max_depth	[11,12,13] → 13	
n_estimators	[100,150,200,250,300] → 150	
min_child_weight	[1,2,3] → 1	
random_state	[42]	
gamma	[0.1,0.2,0.3] → 0.1	
colsample_bytree	[0.5,0.6,0.7] → 0.5	

Third Round:

parameter	**Range**	
booster	'gbtree'	The accuracy best score from best_estimator is 0.8263447707270644.
learning_rate	[0.03,0.04,0.05] → 0.04	
max_depth	[13,14,15] → 13	
n_estimators	[130,140,150,160,170] → 160	
min_child_weight	[1,2] → 1	
random_state	[42]	
gamma	[0.1,0.2] → 0.1	
colsample_bytree	[0.5]	

Forth Round:

Parameter	**Range**	
booster	'gbtree'	The accuracy best score from best_estimator is 0.8268622351513851.
learning_rate	[0.04]	
max_depth	[11,12,13,14,15] → 13	
n_estimators	[155,160,165] → 155	
min_child_weight	[1]	
random_state	[42]	
gamma	[0.1]	
colsample_bytree	[0.5]	

Fifth Round:

Parameter	**Range**	
booster	'gbtree'	The accuracy best score from best_estimator is 0.8268622351513851. We have run the number of estimators one by one. Therefore, we can conclude that this will be the best parameter for the modelling.
learning_rate	[0.04]	
max_depth	[12,13,14] → 13	
n_estimators	151-160, step 1 → 155	
min_child_weight	[1]	
random_state	[42]	
gamma	[0.1]	
colsample_bytree	[0.5]	

Conclusion:

6.6 Step 6 - Model Validation

In this stage, we will check the accuracy of the model built to validate the model.

1. Split the data into train and test set

```
X_train, X_test, y_train, y_test = train_test_split(X, y,test_size=0.2, random_state=42)
```

2. Training of the model

```
xgb = XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=0.5, gamma=0.1, gpu_id=-1,
              importance_type='gain', interaction_constraints='',
              learning_rate=0.04, max_delta_step=0, max_depth=13,
              min_child_weight=1, missing=np.nan, monotone_constraints='()',
              n_estimators=155, n_jobs=0, num_parallel_tree=1,
              objective='multi:softprob', random_state=42, reg_alpha=0,
              reg_lambda=1, scale_pos_weight=None, subsample=1,
              tree_method='exact', validate_parameters=1, verbosity=None)
```

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

3. Check the accuracy of the model built
    a. Using contingency table
        i. Put the features to the model for prediction

```
predictedOutcome = xgb.predict(X_test)
```

        ii. Compare the result with the real classifier

new_cases_percentages	0	1	2	3	All
row_0					
0	157	21	5	0	183
1	25	135	30	0	190
2	1	27	154	24	206
3	2	0	10	182	194
All	185	183	199	206	773

    b. Using cross validation
        i. Put the features to the cross-validation model and obtain the score

```
scores = model_selection.cross_val_score(xgb, X, y, cv = 5)
```

```
scores
```

```
array([0.81112549, 0.81112549, 0.79301423, 0.8240621 , 0.81735751])
```

c. Using classification report with f1-score

   i. Put the features to the classification report function and obtain the score

	precision	recall	f1-score	support
0	0.85	0.85	0.85	185
1	0.70	0.70	0.70	183
2	0.74	0.78	0.76	199
3	0.94	0.90	0.92	206
accuracy			0.81	773
macro avg	0.81	0.81	0.81	773
weighted avg	0.81	0.81	0.81	773

From the score, most of the score is higher than 80%, we can conclude that the problem of overfitting is not significant.

Step 5 Conclusion:

From the contingency table, we can see that most of the value can be matched. To check for overfitting, cross validation has been applied and the score is satisfying. Thus, the model is satisfying.