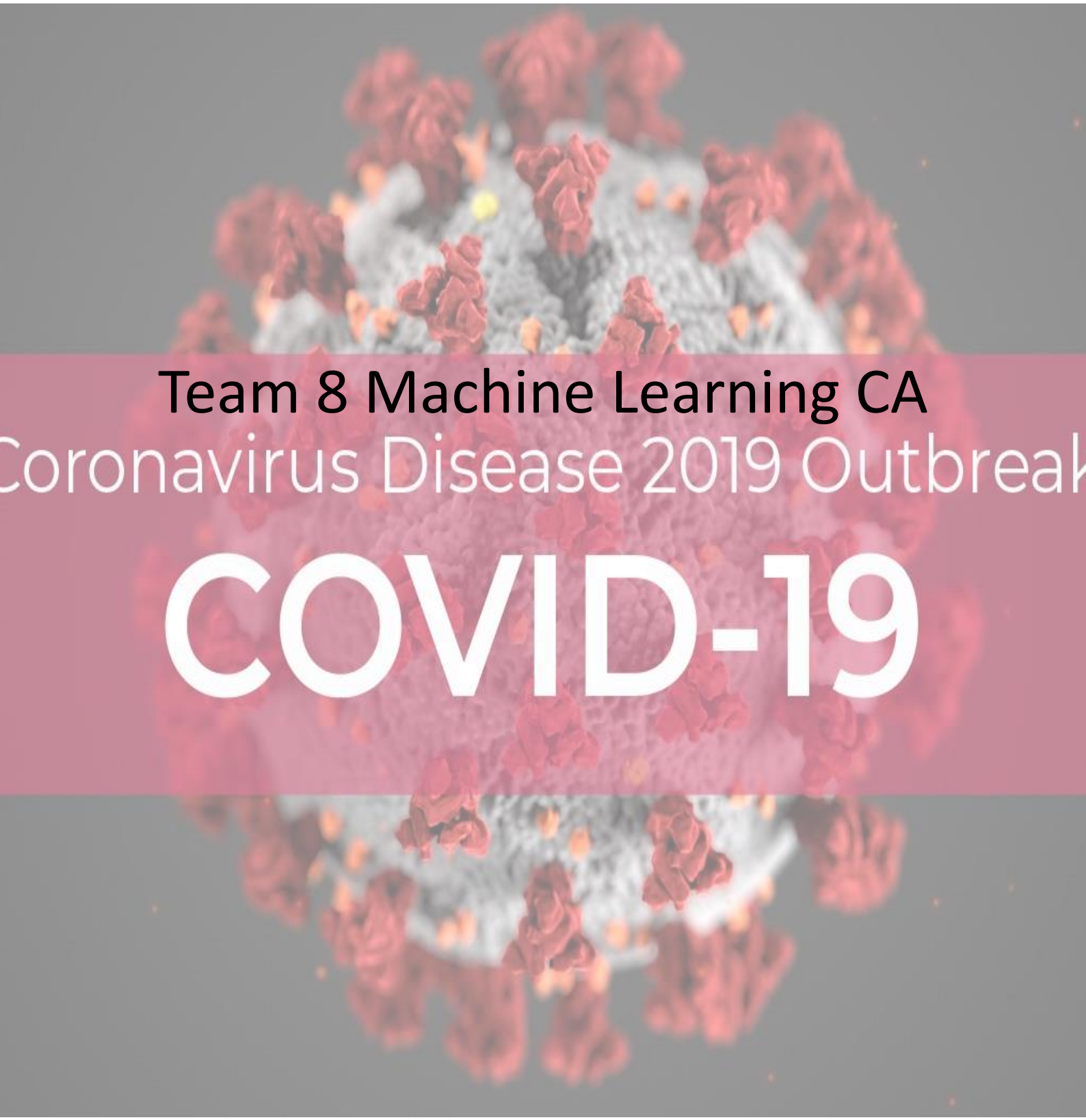

Team 8 Machine Learning CA
Coronavirus Disease 2019 Outbreak
COVID-19

Contents

1. Time Series Forecasting

1.1. Problem Statement

To create a Time Series in order to predict the number of Covid cases around the world globally and across specific countries based on the date the country recorded its 1st case.

Data Dictionary

Variable	Type	Definition
Dates	float64	Index of dates when the 1st Covid-19 case was recorded in the country
No. of Cases	float64	Number of total cases as of that day
New cases	float64	Number of additional cases that day
Percentage change in cases	float64	Number of additional cases that day as a percentage change over the previous day

1.2. Dataset Source

https://www.kaggle.com/sudalairajkumar/novel-corona-virus-2019-dataset?select=time_series_covid_19_confirmed.csv

1.3. Data processing

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

In [2]:
```
# data loading
df = pd.read_csv('time_series_covid19_confirmed_global.csv', index_col = 1,
                 parse_dates = True, infer_datetime_format = True)
# drop unnecessary cols : 'Province/State', 'Lat', 'Long
df= df.drop(columns=['Province/State', 'Lat', 'Long'], axis = 1)
# transpose data so that countries will be columns and every row in dates.
dfTrans = df.transpose()
# rename index name
dfTrans.index.name = 'Dates'
# Since one country like Australia may have more than one region, we sum up all the cases for each country.
dfSum = dfTrans.groupby(dfTrans.columns, axis=1).sum()
'''
dfSum.shape #:(117, 188)
dfSum.columns
Index(['Afghanistan', 'Albania', 'Algeria', 'Andorra', 'Angola',
       'Antigua and Barbuda', 'Argentina', 'Armenia', 'Australia', 'Austria',
       ...
       'United Kingdom', 'Uruguay', 'Uzbekistan', 'Venezuela', 'Vietnam',
       'West Bank and Gaza', 'Western Sahara', 'Yemen', 'Zambia', 'Zimbabwe'],
      dtype='object', name='Country/Region', length=188)'''
# For simplification, Only chose 10 countries to draw the trend, US is a separate plot for clearer visualisation
dfTen = dfSum[['Brazil', 'China','Germany','Italy', 'Korea, South', 'Malaysia',
               'Singapore','United Kingdom']]
dfTen.head()
```

Out[2]:

Country/Region	Brazil	Canada	China	Germany	Italy	Korea, South	Malaysia	Singapore
Dates								
1/22/20	0	0	548	0	0	1	0	0
1/23/20	0	0	643	0	0	1	0	1
1/24/20	0	0	920	0	0	2	0	3
1/25/20	0	0	1406	0	0	2	3	3
1/26/20	0	1	2075	0	0	3	4	4

Firstly, we performed some data engineering to amend the Data frame with index of dates and with each respective country as the columns with the above code(Time Series python file). As it's difficult to properly visualise and represent each individual country and its cases, we decided to filter out a select few countries which would give us enough data to plot a meaningful time series. Additionally, for ease of fitting data, we combined the provinces and states under a single country column to avoid confusion (ie. 1 column for China rather than split by the various provinces, Wuhan, Shanghai etc) and dropping the geographical attributes of longitude and latitude as we did not find it relevant to the no. of Covid-19 cases a country has.

In [48]:
```
# Get when this coutry actually got cases with not-zero values
# by counting how many zeroes in this column
countZero = dfTen.apply(lambda x : x.value_counts().get(0,0))
# run a loop in all columns
for index, row in dfTen.iteritems():
    # if this country have 3 zero then it will up-shift 3
    dfTen.loc[:,index] = dfTen.loc[:,index].shift(-countZero[index])

#reindex the dates
dfTen = dfTen.reset_index(drop=True)
dfTen.head()
```

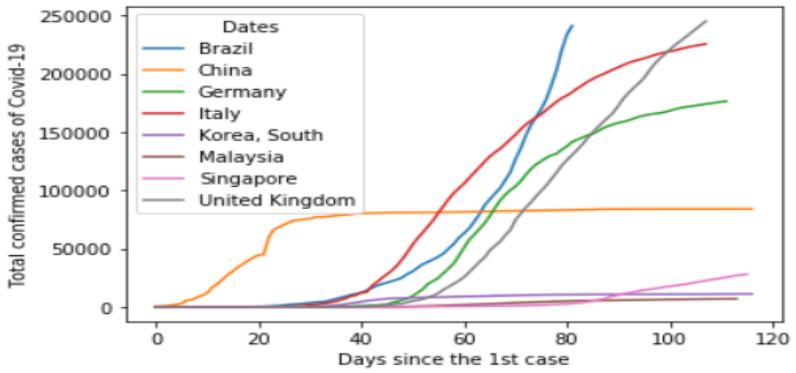
Out[48]:

Dates	Brazil	China	Germany	Italy	Korea, South	Malaysia	Singapore	United Kingdom
0	1.0	548	1.0	2.0	1	3.0	1.0	2.0
1	1.0	643	4.0	2.0	1	4.0	3.0	2.0
2	1.0	920	4.0	2.0	2	4.0	3.0	2.0
3	2.0	1406	4.0	2.0	2	4.0	4.0	2.0
4	2.0	2075	5.0	2.0	3	7.0	5.0	2.0

As the original csv file was had the started recording the dates from 22nd Jan 2020, we felt that in some cases it would be more meaningful if we used the data starting from the 1st day that a country recorded it's first covid cases as not all countries would have recorded cases yet on 22nd Jan hence this would have caused some of the lines plotted in the time series to not be useful. This involved counting all the zero values a country had, removing the zero values to fit data to start from Day of 1st recorded case and shifting the values below up accordingly.

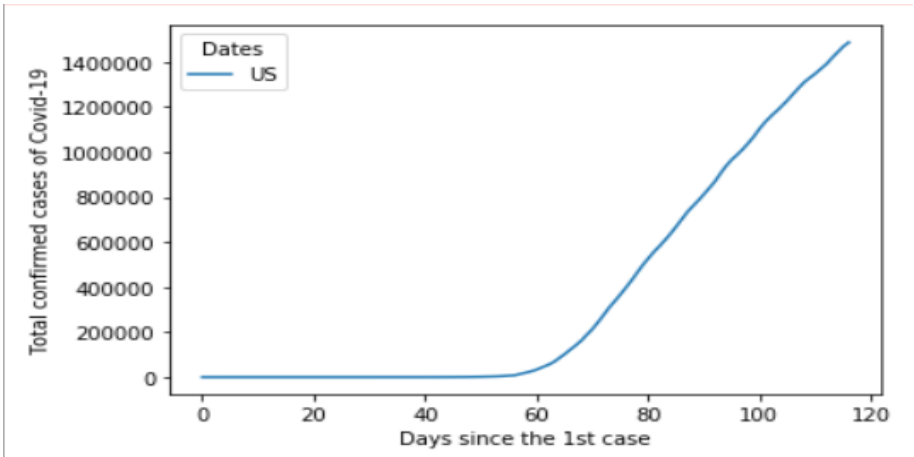
1.4. Visualisation

In [13]:
```
start_time = time.time()
dfTen.plot()
plt.xlabel('Days since the 1st case')
plt.ylabel('Total confirmed cases of Covid-19')
plt.show()
end_time = time.time()
print ((end_time - start_time))
```




```
0.3115272521972656
```

In [14]:
```
start_time = time.time()
dfus = df4[['US']]
countZero_us = dfus.apply(lambda x : x.value_counts().get(0,0))
for index, row in dfus.iteritems():
    # if this country have 3 zero then it will up-shift 3
    dfus.loc[:,index] = dfus.loc[:,index].shift(-countZero_us[index])
dfus= dfus.reset_index(drop=True)
dfus.plot()
plt.xlabel('Days since the 1st case')
plt.ylabel('Total confirmed cases of Covid-19')
plt.show()
end_time = time.time()
print ((end_time - start_time))
```

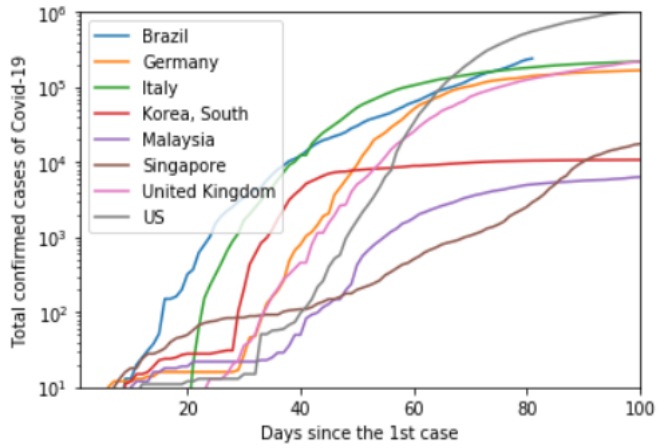



```
0.27725887298583984
```

For easier visualisation we selected 10 of the more relevant countries from the overall data set. These tended to be the countries with the higher number of cases globally, each with a different rate of case

growth for a more varied display range.
We decided to separate US from their other countries as given it's high number of cases, we felt that it would skew graph disproportionately relative to the rest of the other countries.

```
start_time = time.time()
# y-logarithmic coordinates
# Removed China because the first day is already 500 hence not good for comparison
dfLog = pd.concat([dfTen[['Brazil','Germany','Italy', 'Korea, South', 'Malaysia',
            'Singapore','United Kingdom']], dfus])
dfLog.plot()
# introduce y-logarithmic setting
plt.yscale('log')
# feel free the adjust the range
plt.xlim(1, 100)
plt.ylim(10, 1e6)
plt.xlabel('Days since the 1st case')
plt.ylabel('Total confirmed cases of Covid-19')
plt.show()

end_time = time.time()
print ((end_time - start_time))
```

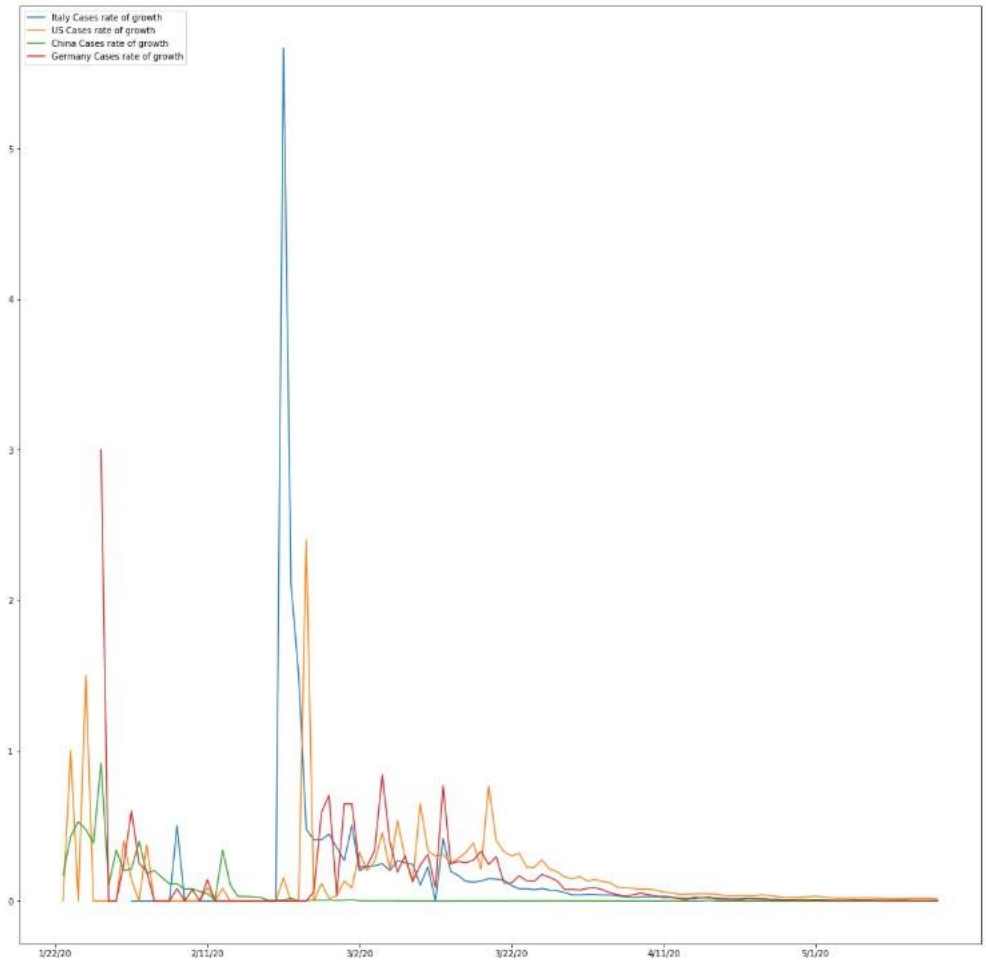



```
0.7312538623809814
```

To get around this issue to have a complete picture of all 10 countries however, we thought it might help if we changed the range of the y-axis using 'log' so that rather than representing the absolute values of the number of Covid-19 cases. Additionally, we removed China from the list for this plot as it already started with a high number of cases from Day 0 hence it would not have been ideal for comparison relative to the other countries.
This enables a better visualisation of the growth rate over time.

In addition, we could use percentage change, or additional cases to visualise the increase in cases, i.e. comparing rates of growth using pct_change()

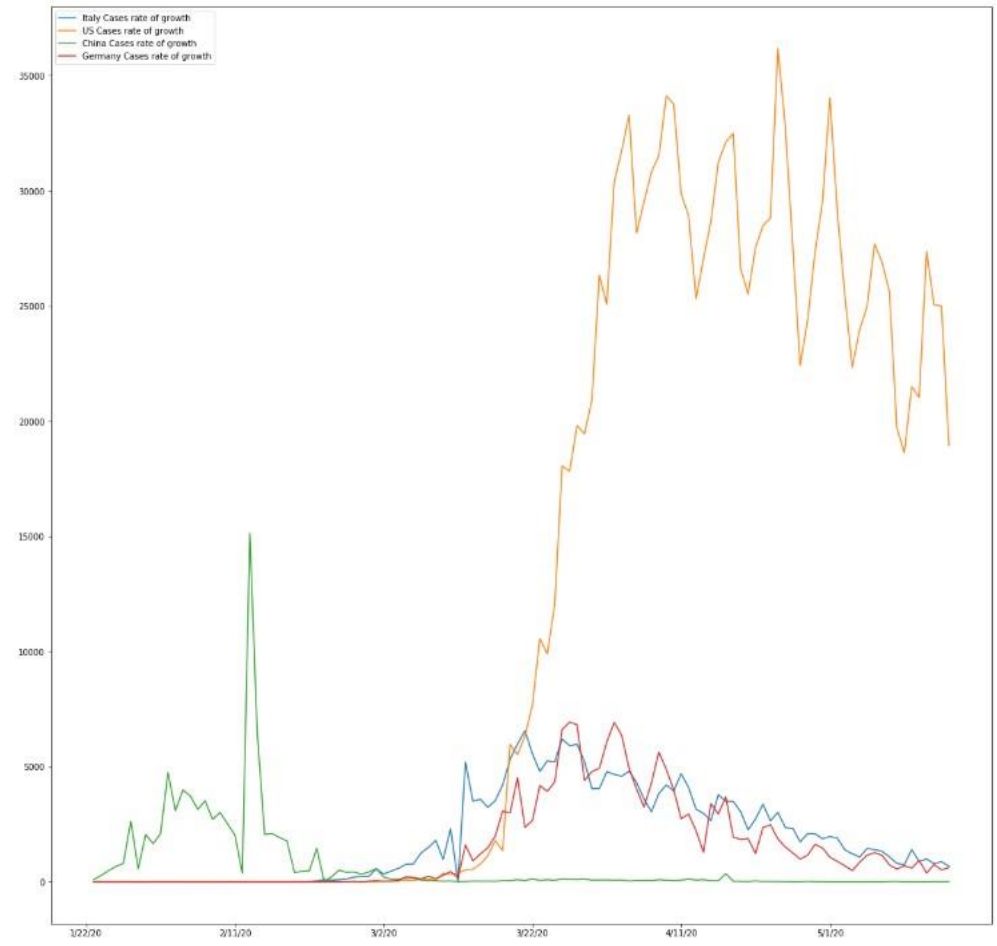
In [33]:
```
dfdiff1 = df3.pct_change()

plt.figure(figsize=(20,20))

plt.plot(dfdiff1.index,dfdiff1.Italy, label = 'Italy Cases rate of growth')
plt.plot(dfdiff1.index,dfdiff1.US, label = 'US Cases rate of growth')
plt.plot(dfdiff1.index,dfdiff1.China, label = 'China Cases rate of growth')
plt.plot(dfdiff1.index,dfdiff1.Germany, label = 'Germany Cases rate of growth')
plt.legend(loc='upper left')
plt.show()
```




```
0.8096609115600586
```

We could also use diff() to look at the number of new cases, but this figure may be highly dependent on the number of testing done and accessibility to healthcare in each country. Diff() obtains the the difference in cumulative cases between one day and the day earlier.

In [77]:
```
dfdiff1 = df4.diff()

plt.figure(figsize=(20,20))

plt.plot(dfdiff1.index,dfdiff1.Italy, label = 'Italy Cases number of new cases for each day')
plt.plot(dfdiff1.index,dfdiff1.US, label = 'US Cases number of new cases for each day')
plt.plot(dfdiff1.index,dfdiff1.China, label = 'China number of new cases for each day')
plt.plot(dfdiff1.index,dfdiff1.Germany, label = 'Germany number of new cases for each day')
plt.legend(loc='upper left')
plt.show()
```




```
0.9725379943847656
```

1.5. Average of Recent Period

In [62]:
```
RECENT_PERIOD = 3

df_recent_period = df4[['Brazil', 'China','Germany','Italy', 'Korea, South', 'Malaysia',
             'Singapore','United Kingdom', 'US']][-RECENT_PERIOD:]

df_recent_period
```

Out[62]:

Dates	Brazil	China	Germany	Italy	Korea, South	Malaysia	Singapore	United Kingdom	US
5/15/20	220291	84038	175233	223885	11037	6855	26891	238004	1442824
5/16/20	233511	84044	175752	224760	11050	6872	27356	241461	1467820
5/17/20	241080	84054	176369	225435	11065	6894	28038	244995	1486757

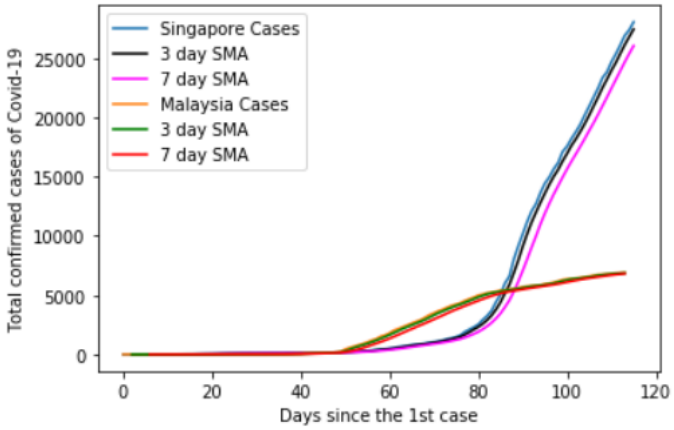
In [67]:
```
# average value of recent period
df_recent_period.mean()
```

Out[67]:
```
Dates
Brazil            2.316273e+05
China             8.404533e+04
Germany           1.757847e+05
Italy             2.246933e+05
Korea, South      1.105067e+04
Malaysia          6.873667e+03
Singapore         2.742833e+04
United Kingdom    2.414867e+05
US                1.465800e+06
dtype: float64
```

1.6. Singapore and Malaysia Moving Average

In [19]:
```
rolling_mean_sg_3 = dfTen.Singapore.rolling(window = 3).mean()
rolling_mean_sg_7 = dfTen.Singapore.rolling(window = 7).mean()

rolling_mean_my_3 = dfTen.Malaysia.rolling(window = 3).mean()
rolling_mean_my_7 = dfTen.Malaysia.rolling(window = 7).mean()
```

In [20]:
```
start_time = time.time()
plt.plot(dfTen.index,dfTen.Singapore, label = 'Singapore Cases')
plt.plot(dfTen.index, rolling_mean_sg_3, label='3 day SMA', color='black')
plt.plot(dfTen.index, rolling_mean_sg_7, label='7 day SMA', color='magenta')

plt.plot(dfTen.index,dfTen.Malaysia, label = 'Malaysia Cases')
plt.plot(dfTen.index, rolling_mean_my_3, label='3 day SMA', color='green')
plt.plot(dfTen.index, rolling_mean_my_7, label='7 day SMA', color='red')

plt.legend(loc='upper left')
plt.xlabel('Days since the 1st case')
plt.ylabel('Total confirmed cases of Covid-19')
plt.show()
end_time = time.time()
print ((end_time - start_time))
```



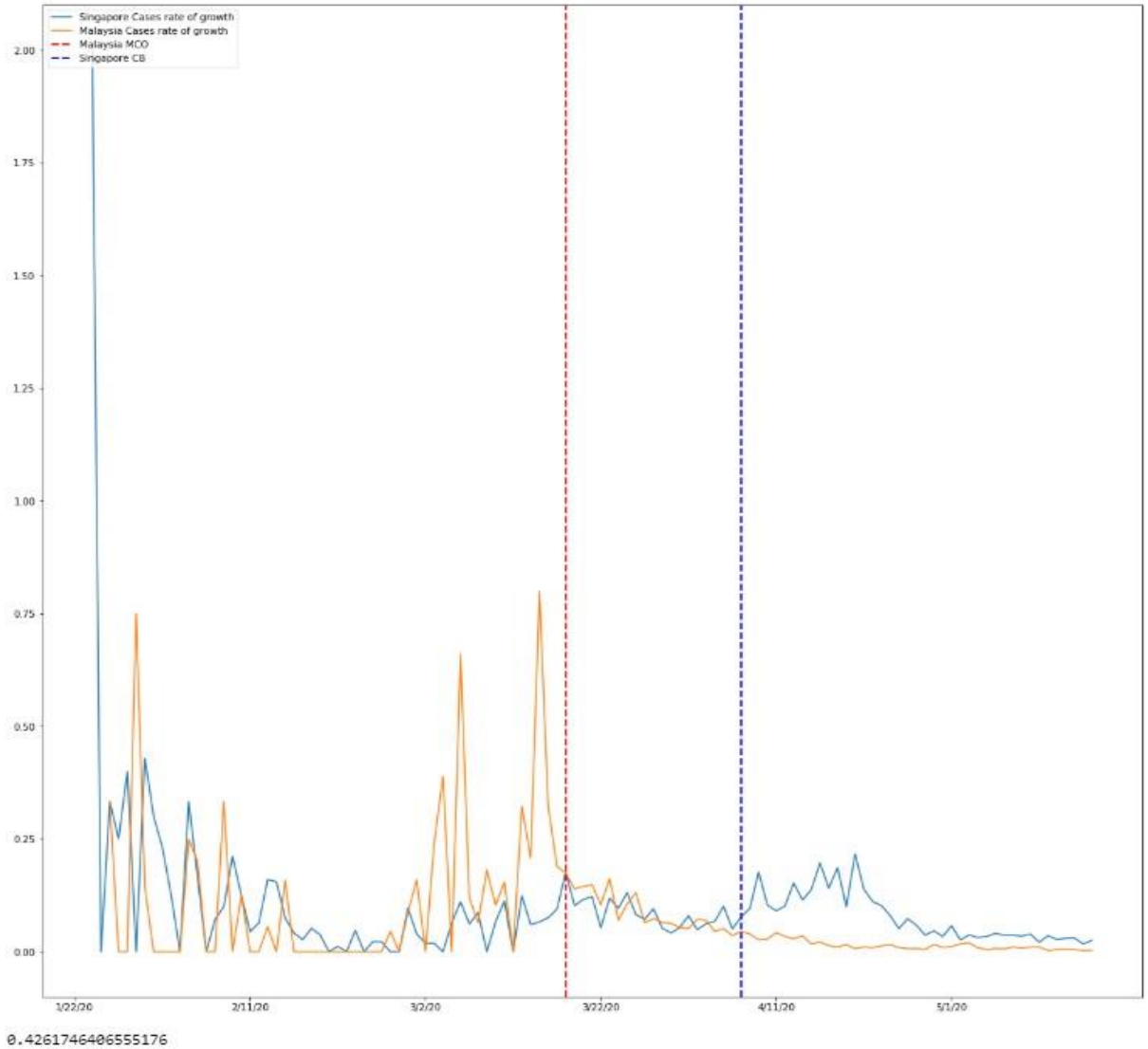

```
0.31914687156677246
```

We selected Singapore and Malaysia for the Simple Moving Average (SMA) as our 2 sample nations for comparison given their similar initial case growth. As per the graphs above, whilst the 7 day SMA do follow the actual number of cases very closely, the 3 day SMA is still by far a lot more accurate, especially in the case of Singapore where the line almost matches the actual number of cases exactly.

Using the similar method above, we could go deeper and look at country-specific growth rates pre and post the implementation of the lockdown. In this case, we look at Singapore's Circuit Breaker (CB) and Malaysia's Movement Control Order (MCO).

Rate of growth before and after Malaysia and Singapore's lockdowns

```
In [54]: plt.figure(figsize=(20,20))

         plt.plot(dfpct.index,dfpct.Singapore, label = 'Singapore Cases rate of growth')
         plt.plot(dfpct.index,dfpct.Malaysia, label = 'Malaysia Cases rate of growth')
         plt.axvline('3/18/20', linewidth=2, color='r', label = 'Malaysia MCO', linestyle='--')
         plt.axvline('4/7/20', linewidth=2, color='b', label = 'Singapore CB', linestyle='--')
         plt.legend(loc='upper left')
         plt.show()
```




```
0.4261746406555176
```

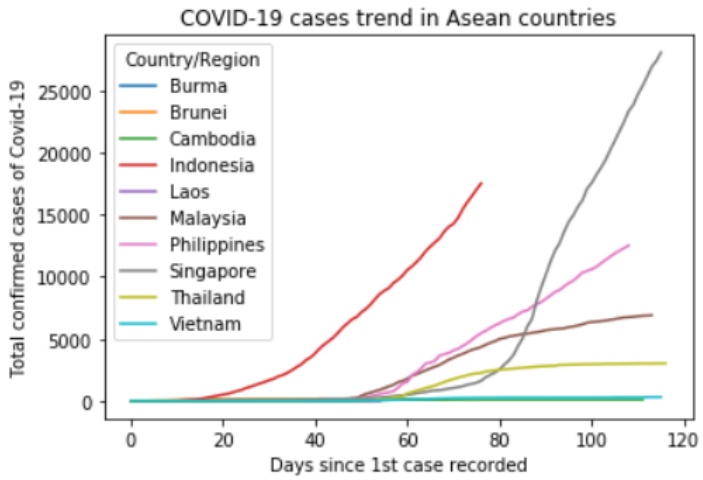
We could visualize Singapore's circuit breaker taking a significantly longer time after the CB before dropping to levels like Malaysia's. Malaysia's MCO brought about a steady decrease in the growth rates.

1.7. ASEAN Region

In [10]:
```
dfAsean = dfSum[[ 'Burma','Brunei','Cambodia','Indonesia','Laos',
                  'Malaysia','Philippines','Singapore','Thailand','Vietnam']]
```

In [15]:
```
dfAseanShift = dfAsean
# Get when this coutry actually got cases with not-zero values
# by counting how many zeros in this column totally
countZeroAsean = dfAseanShift.apply(lambda x : x.value_counts().get(0,0))
# run a loop in all columns
for index, row in dfAseanShift.iteritems():
    # if this country have 3 zero then it will up-shift 3
    dfAseanShift.loc[:,index] = dfAseanShift.loc[:,index].shift(-countZeroAsean[index])
#Testing #dfTenc.loc[:,'Brazil'].shift(-countZero['Brazil'])
#reindex the dates
dfAseanShift = dfAseanShift.reset_index(drop=True)
dfAseanShift.index.name = 'Dates'
dfAseanShift
```

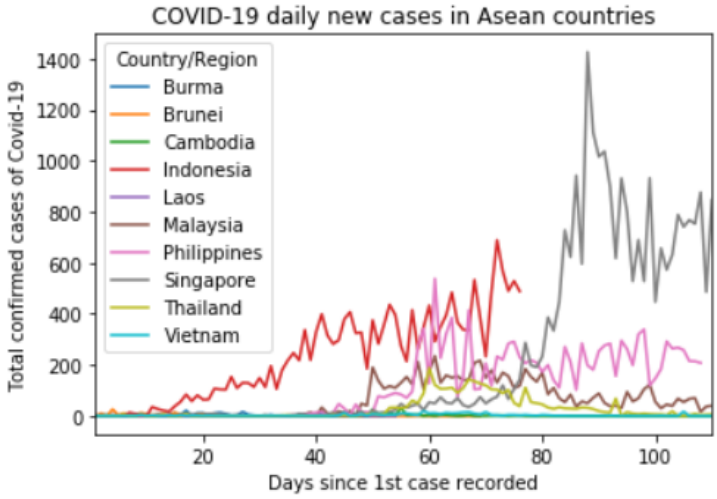
In [16]:
```
dfAseanShift.plot()
plt.title('COVID-19 cases trend in Asean countries')
plt.xlabel('Days since 1st case recorded')
plt.ylabel('Total confirmed cases of Covid-19')
plt.show()
```



We then thought to look at the countries within ASEAN to see how it fared as a region. Given that certain countries like Indonesia started recording its COVID-19 cases later than its neighbours, we then felt it might be better served if we looked the daily increase of new cases instead.

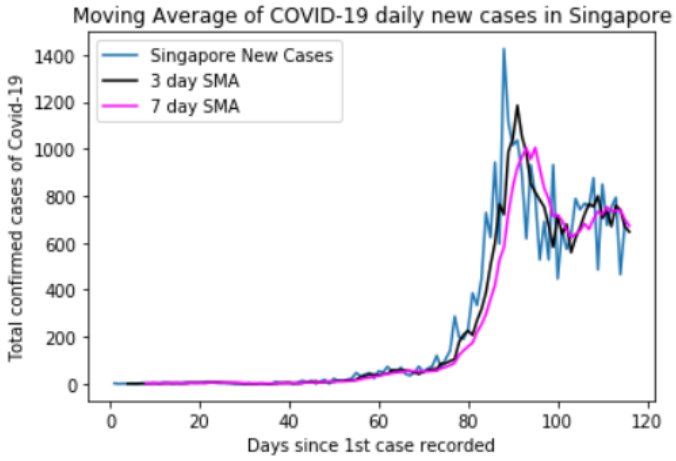
ASEAN Daily New Cases

In [17]:
```
dfAseanDiff = dfAseanShift.astype('float').diff(1).dropna(how='all')
dfAseanDiff.plot()
plt.title('COVID-19 daily new cases in Asean countries')
plt.xlim(1,110)
plt.xlabel('Days since 1st case recorded')
plt.ylabel('Total confirmed cases of Covid-19')
plt.show()
```



Singapore Daily New Cases Moving Average ¶

In [18]:
```
#Moving Average
plt.plot(dfAseanDiff.index,dfAseanDiff.Singapore, label = 'Singapore New Cases')
plt.plot(dfAseanDiff.index,dfAseanDiff.Singapore.rolling(window = 3).mean().shift(1),
         label='3 day SMA', color='black')
plt.plot(dfAseanDiff.index,dfAseanDiff.Singapore.rolling(window = 7).mean().shift(1),
         label='7 day SMA',color='magenta')
plt.title('Moving Average of COVID-19 daily new cases in Singapore')
plt.xlabel('Days since 1st case recorded')
plt.ylabel('Total confirmed cases of Covid-19')
plt.legend(loc='upper left')
plt.show()
```



Using similar techniques as we did for the initial list of countries initially, we plotted out the graph detailing the daily increase of new COVID-19 cases amongst ASEAN countries as well as the Simple Moving Average of Singapore's daily new COVID-19 cases. As seen from the graphs above, whilst there is a lot fluctuation of daily cases resulting in a non linear shape of the curve, the 3 day SMA still mirrors the actual number of daily cases quite closely as compared to the 7 day SMA.

Singapore Daily New Cases Mean Squared Error ¶

In [28]:
```
from sklearn.metrics import mean_squared_error
# ignore the first fews rows with no moving average data
rolling_mean_total_3 = dfAseanDiff.Singapore.rolling(window = 3).mean()[3:-1]
rolling_mean_total_7 = dfAseanDiff.Singapore.rolling(window = 7).mean()[7:-1]

print(mean_squared_error(dfAseanDiff.Singapore[3:-1], rolling_mean_total_3))
print(mean_squared_error(dfAseanDiff.Singapore[7:-1], rolling_mean_total_7))
```

```
6248.953373015873
14825.771164021162
```

This is further reflected when calculated the respective Mean Squared Errors where the MSE for a 3 day SMA is much smaller than the 7 day SMA, as is consistent with our other time series models.

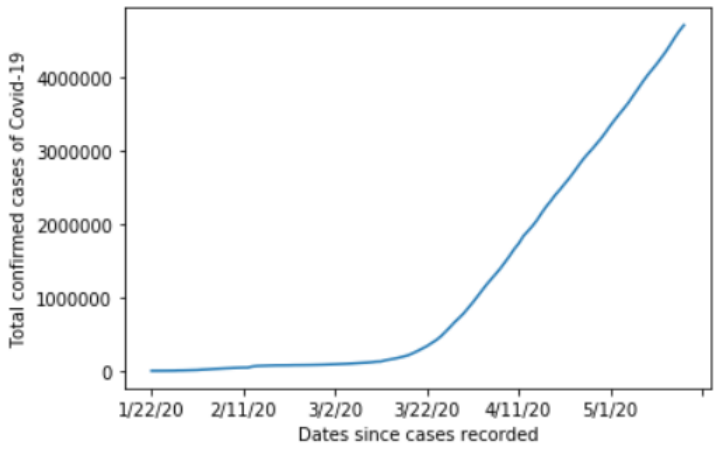
1.8. Total Global Cases

```
In [29]: df3 = df2.groupby(df2.columns, axis=1).sum()
         df_Global = df3.sum(axis = 1)
         df_Global
```

```
Out[29]: 1/22/20         555
         1/23/20         654
         1/24/20         941
         1/25/20        1434
         1/26/20        2118
                        ...
         5/13/20     4347018
         5/14/20     4442163
         5/15/20     4542347
         5/16/20     4634068
         5/17/20     4713620
         Length: 117, dtype: int64
```

We then sought to plot the Total Global Cases to identify any patterns using the similar methods as we had done for the earlier countries.

```
In [32]: start_time = time.time()
         df_Global.plot()


         plt.xlabel('Dates since cases recorded')
         plt.ylabel('Total confirmed cases of Covid-19')
         plt.show()
         end_time = time.time()
         print ((end_time - start_time))
```




```
0.15857172012329102
```

Looking at global cases combined, we did a study of the mean squared average for 3 day rolling window and 7 day rolling window. The MSE was lower for the 3 day rolling window.

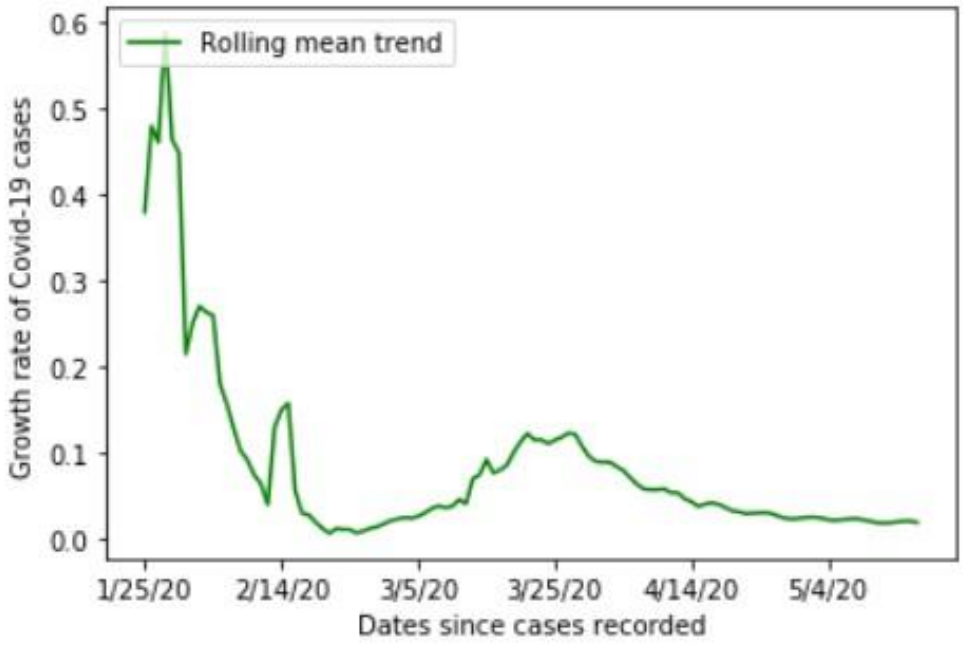
Calculation of mean squared error for global cases

```
In [56]: from sklearn.metrics import mean_squared_error
         # ignore the first fews rows with no moving average data
         rolling_mean_total_3 = df4['Total'].rolling(window = 3).mean()[3:]
         rolling_mean_total_7 = df4['Total'].rolling(window = 7).mean()[7:]

         mean_squared_error(df4['Total'][3:], rolling_mean_total_3)

Out[56]: 1795899890.2329438

In [57]:
         mean_squared_error(df4['Total'][7:], rolling_mean_total_7)

Out[57]: 16221928547.82022
```

The respective Mean Squared Errors where the MSE for a 3 day SMA is much smaller than the 7 day SMA, is consistent with our other time series models.

```
In [62]: from dateutil.relativedelta import relativedelta # working with dates with style
         from scipy.optimize import minimize              # for function minimization

         import statsmodels.formula.api as smf            # statistics and econometrics
         import statsmodels.tsa.api as smt
         import statsmodels.api as sm
         import scipy.stats as scs

         from sklearn.metrics import r2_score, median_absolute_error, mean_absolute_error
         from sklearn.metrics import median_absolute_error, mean_squared_error, mean_squared_log_error

         def mean_absolute_percentage_error(y_true, y_pred):
             return np.mean(np.abs((y_true - y_pred) / y_true)) * 100


         plt.plot(rolling_mean_total_3, "g", label="Rolling mean trend")
```

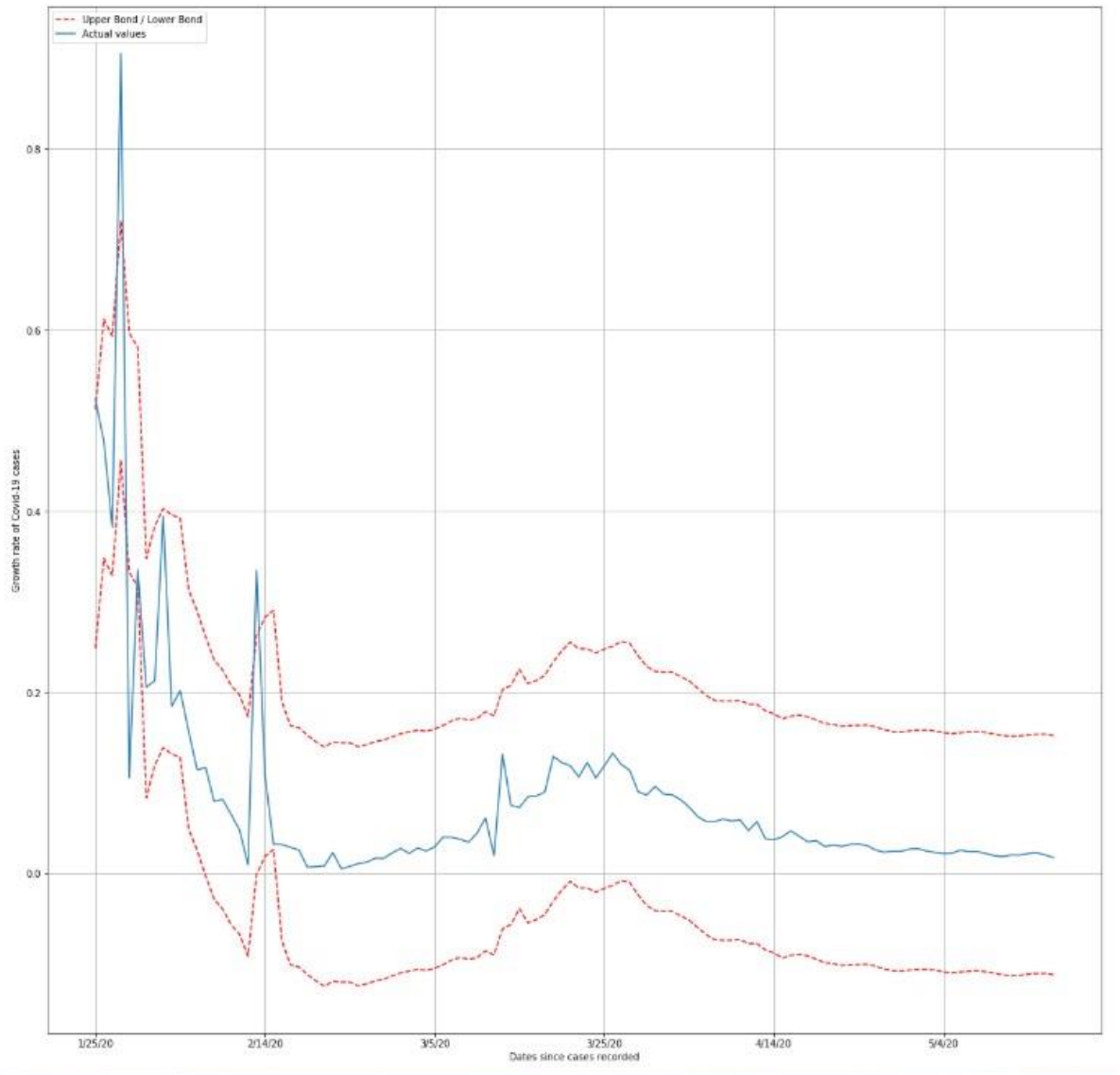
Using 3 day rolling mean of the percentage change, we plotted out the **global growth rate**.




```
0.346299409866333
```

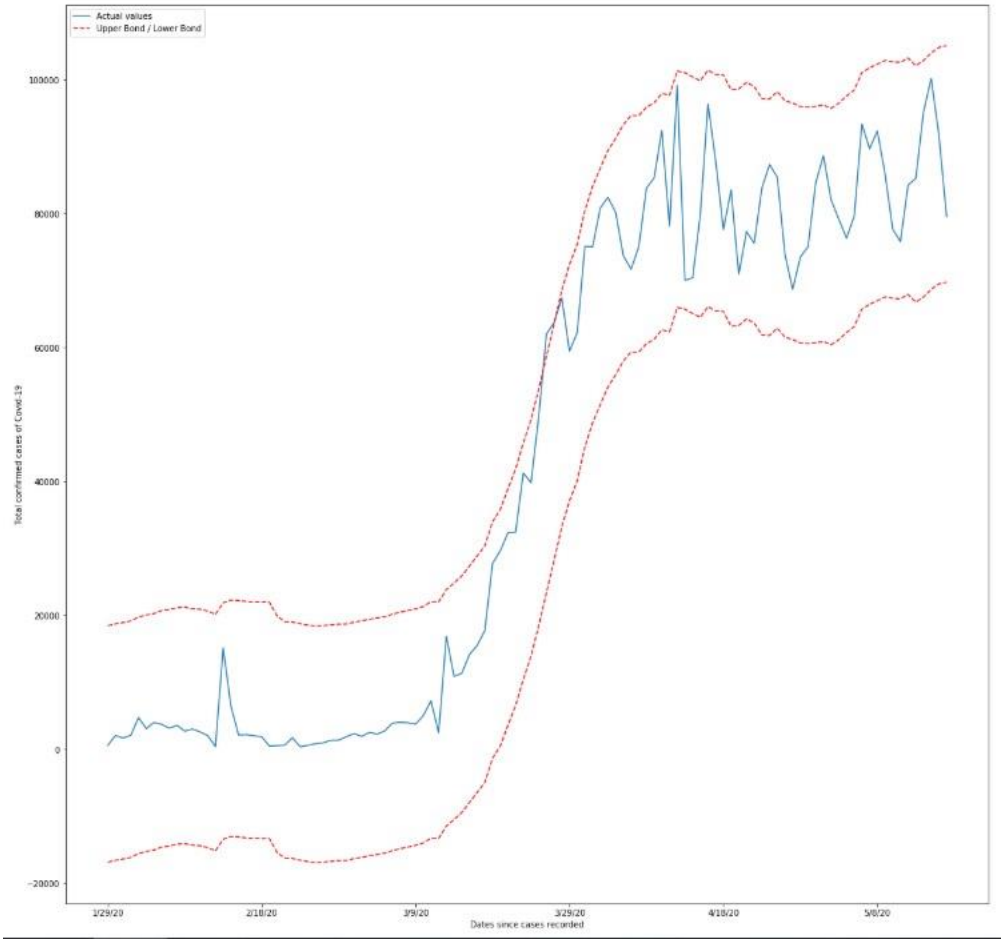
Using the 3 day rolling mean and a 95% confidence interval, we could use it to forecast the upper bond and lower bond of the **growth rate**.

```
# FORECAST PERCENTAGE GROWTH
def plotMovingAverage(dftotalpct,plot_intervals=True, scale=1.96, plot_anomalies=False) :
    plt.figure(figsize=(20,20))
    mae = mean_absolute_error(dftotalpct[3:], rolling_mean_total_3)
    deviation = np.std(dftotalpct[3:] - rolling_mean_total_3)
    lower_bond = rolling_mean_total_3 - (mae + scale * deviation)
    upper_bond = rolling_mean_total_3 + (mae + scale * deviation)
    plt.plot(upper_bond, "r--", label="Upper Bond / Lower Bond")
    plt.plot(lower_bond, "r--")

plotMovingAverage(dftotalpct,3)
plt.plot(dftotalpct[3:], label="Actual values")
plt.legend(loc="upper left")
plt.grid(True)
```



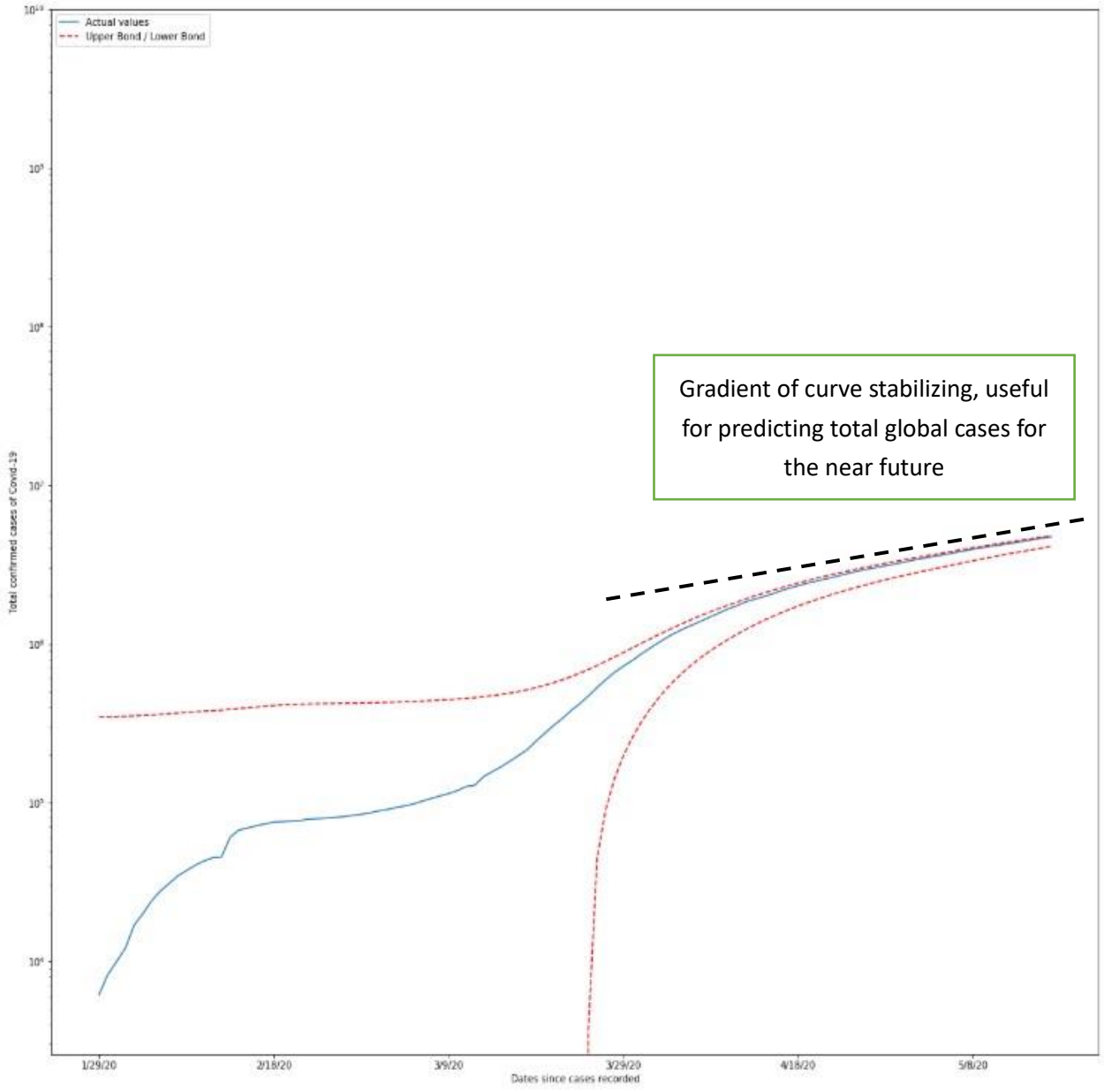
Apart from the total case percentage growth rate, we could also use the 7 day rolling mean of new cases to forecast the lower bond and upper bond of **future new cases**.

In [71]:
```
#using moving average 7 days to forecast NEW CASES

dfnew = df4['Total'].diff()
dfnew

def plotMovingAverage(dfnew,plot_intervals=True, scale=1.96, plot_anomalies=False) :
    plt.figure(figsize=(20,20))
    rolling_mean_total_3 = dfnew.rolling(window = 3).mean()[3:]
    rolling_mean_total_7 = dfnew.rolling(window = 7).mean()[7:]
    mae = mean_absolute_error(dfnew[7:], rolling_mean_total_7)
    deviation = np.std(dfnew[7:] - rolling_mean_total_7)
    lower_bond = rolling_mean_total_7 - (mae + scale * deviation)
    upper_bond = rolling_mean_total_7 + (mae + scale * deviation)
    plt.plot(dfnew[7:], label="Actual values")
    plt.plot(upper_bond, "r--", label="Upper Bond / Lower Bond")
    plt.plot(lower_bond, "r--")


plotMovingAverage(dfnew,7)
plt.legend(loc="upper left")
```



Putting **total global cases** through a logarithm with 7 day moving average, we notice that the lower bond, upper bond and actual accumulated cases seem to be converging. We can deduce that this could be the expected growth rate for future cases as the Covid19 virus continues to spread and worsen around the world.

The time series data is a useful tool to help us gain an insight to only short term future trends, as it does not take into account future changes such as more countries initiating lockdown, development of vaccine, or declining infection rate with more herd immunity.



In [66]:
```
#FORECAST TOTAL GLOBAL CASES using moving average 7 days and then LOG IT
dftotalforlog = df4['Total']

def plotMovingAverage(dftotalforlog,plot_intervals=True, scale=1.96, plot_anomalies=False) :
    plt.figure(figsize=(20,20))
    rolling_mean_total_3 = dftotalforlog.rolling(window = 3).mean()[3:]
    rolling_mean_total_7 = dftotalforlog.rolling(window = 7).mean()[7:]
    mae = mean_absolute_error(dftotalforlog[7:], rolling_mean_total_7)
    deviation = np.std(dftotalforlog[7:] - rolling_mean_total_7)
    lower_bond = rolling_mean_total_7 - (mae + scale * deviation)
    upper_bond = rolling_mean_total_7 + (mae + scale * deviation)
    plt.plot(dftotalforlog[7:], label="Actual values")
    plt.plot(upper_bond, "r--", label="Upper Bond / Lower Bond")
    plt.plot(lower_bond, "r--")


plotMovingAverage(dftotalforlog,7)
# introduce y-logarithmic setting
plt.yscale('log')
# feel free the adjust the range
plt.ylim(0, 1e10)
plt.legend(loc="upper left")
```

1.9. Conclusion

As seen across the various time series models above, in general the 3 Day Simple Moving Average(SMA) is a fairly accurate tool that one can use to forecast future values in a time series, for this particular dataset of COVID-19 cases. However, there are drawbacks to the usage of a time-series model in the first place.

There's an inherent uncertainty of other factors that may contribute to the dependent variable beyond just time. This may make the time-series appear as accurate and a good model, when it might not be so. These other factors that influence the time series may not remain constant for an extended period of time and forecasting made on this basis may become unreliable.

Additionally, given the time sensitive nature of the model, the dataset that is being fed to it is required to be constantly refreshed and updated. The time series like many other machine models should not be used as a test of data alone, and should be combined with other machine learning models to have a comprehensive and more accurate prediction.

2. Background for supervised and unsupervised learning

2.1. Problem statement

We will be investigating common laboratory test results collected from suspected Covid-19 patient to predict Covid-19 infection in patients by building Supervised Machine Learning Model and Unsupervised Learning Model based on a dataset obtained from Hospital Israelita Albert Einstein, at São Paulo, Brazil. We will be comparing the outcome of each model and exploring the impact of Data Engineering and Feature Engineering on the outcomes as well as discussing the limitations of this experiment.

2.2. Sharing of full the process

The team first come up with a basic data sets with minimal complications to feature engineering and minimal complexity of each training model. We studied each individual outcome and compare that to our hypothesis and the trying to understand why some expected trend or results were not achieved.

The training models and accuracy scores are then recalculated based on a more refined model and / or with more complications introduced to ensure the final outputs are as consistent as possible by having minimal standard deviations for discussion.

The codes for the trail run on supervised learning can be found in 'CA Supervised Learning Models Refinement Run 1'; codes for refinement runs on supervised learning can be found in 'CA Supervised Learning Models Refinement Run 1' and 'CA Supervised Learning Models Refinement Run 2'; codes for unsupervised learning can be found in 'CA Unsupervised Learning Models'

2.3. Data dictionary

Variable	Type	Definition
sars_cov_2_exam_result	Int	The 'positive' and 'negative' indicator of Covid-19
hematocrit	Float64	Hematocrit count
hemoglobin	Float64	Hemoglobin count
platelets	Float64	Platelets count
mean_platelet_volume	Float64	Average size of platelets
red_blood_cells	Float64	Red blood cells count
lymphocytes	Float64	Lymphocytes count
mean_corpuscular_hemoglobin_concentration_mchc	Float64	Average concentration of hemoglobin in a given volume of red blood cells.
leukocytes	Float64	White blood cells/Leukocytes count
basophils	Float64	Basophils count

mean_corpuscular_hemoglobin_mch	Float64	Average weight of hemoglobin in a given volume of red blood cells.
eosinophils	Float64	Eosinophils count
mean_corpuscular_volume_mcv	Float64	Average size and volume of a red blood cell
monocytes	Float64	Monocytes count
red_blood_cell_distribution_width_rdw	Float64	range of variation of red blood cell volume that is reported as part of a standard complete blood count
neutrophils	Float64	Neutrophils count
proteina_c_reativa_mg_dl	Float64	Volume of proteina_c_reativa in mg/dl

2.4. Source of dataset

https://www.kaggle.com/einsteindata4u/covid19?select=dataset.xlsx

2.5. Data cleaning

2.5.1. Selecting the proper sample size and dimensions

Our primary goal is to investigate only using the numerical data of the dataset.

```
booldict = {
    "negative" : 0,
    "positive" : 1
}

df["sars_cov_2_exam_result"] = df["sars_cov_2_exam_result"].map(booldict)
```

```
df = df.select_dtypes(exclude=['object'])
df = df.drop("patient_age_quantile", axis=1)
df.head()
```

	sars_cov_2_exam_result	hematocrit	hemoglobin	platelets	mean_platelet_volume	red_blood_cells	lymphocytes	mean_corpuscular_hemoglobin_concer
0	0	NaN	NaN	NaN	NaN	NaN	NaN	
1	0	0.236515	-0.02234	-0.517413	0.010677	0.102004	0.318366	
2	0	NaN	NaN	NaN	NaN	NaN	NaN	
3	0	NaN	NaN	NaN	NaN	NaN	NaN	
4	0	NaN	NaN	NaN	NaN	NaN	NaN	

5 rows × 70 columns

Figure 1 Encoding and Removing Object DataType

We first encoded the label, “sars_cov_2_exam_result” to numerical values and remove all non-numerical data from the dataset as shown in Figure 1.

However, the data still contain too many missing data. Our next step is to find a list of best possible sample size (rows) and dimensions (columns) so we can narrow down to the “optimal” usable sample size.

```
def cleanse():
    for i in np.arange(0.8, 1, 0.01):
        df_copy = df
        df_copy = df_copy.loc[:, df.isnull().sum() < i*df.shape[0]]
        df_copy = df_copy.dropna()
        print("Threshold: ", i, "Shape: ", df_copy.shape)

cleanse()
```

```
Threshold:  0.8 Shape:  (5644, 1)
Threshold:  0.81 Shape:  (5644, 1)
Threshold:  0.8200000000000001 Shape:  (5644, 1)
Threshold:  0.8300000000000001 Shape:  (5644, 1)
Threshold:  0.8400000000000001 Shape:  (5644, 1)
Threshold:  0.8500000000000001 Shape:  (5644, 1)
Threshold:  0.8600000000000001 Shape:  (5644, 1)
Threshold:  0.8700000000000001 Shape:  (5644, 1)
Threshold:  0.8800000000000001 Shape:  (5644, 1)
Threshold:  0.8900000000000001 Shape:  (5644, 1)
Threshold:  0.9000000000000001 Shape:  (598, 15)
Threshold:  0.9100000000000001 Shape:  (510, 16)
Threshold:  0.9200000000000002 Shape:  (420, 17)
Threshold:  0.9300000000000002 Shape:  (272, 19)
Threshold:  0.9400000000000002 Shape:  (242, 21)
Threshold:  0.9500000000000002 Shape:  (242, 21)
Threshold:  0.9600000000000002 Shape:  (122, 22)
Threshold:  0.9700000000000002 Shape:  (50, 27)
Threshold:  0.9800000000000002 Shape:  (18, 37)
Threshold:  0.9900000000000002 Shape:  (0, 47)
```

Figure 2 List of usable data shape

From the original trail run as shown in Figure 2, data shape of (420, 17) has a decent sample size of 420 while preserving a decent number of 17 dimensions for the experiment.

We later discovered based on this sample size, there were too many imbalanced data with negatives or 0 being the obvious majority so we further refine the sample selection process taking into consideration of positive/negative ratios of each sample size.

```
index  0
Number of positives: 112
Number of negatives: 1240
Positive to Negative percentage: 9.032258064516128
1352 rows x  18 columns

index  1
Number of positives: 62
Number of negatives: 758
Positive to Negative percentage: 8.179419525065963
820 rows x  3 columns

index  2
Number of positives: 59
Number of negatives: 361
Positive to Negative percentage: 16.343490304709142
420 rows x  17 columns
```

Figure 3 Example of sample data after refinement

Figure 3 shows the first 3 data sample which can be extracted from the source data based on a fair dimension and positive to negative label percentage (PNLP).The full data can be found from Appendix 1 to Appendix 3 and the details of the refinement methodology can be found in the attached code file 'CA Supervised Learning Models Refinement Run 2', Part 1.

By following our philosophy of maintaining the most number of rows and columns while minimizing imbalance, index 0 and 2 fits the criteria. Index 0 have the most number of rows and columns and a fair
PNLP of 9.03 whereas index 2 have fair amount of columns and a high PNLP of 16.34.

```
In [25]: dfs[0]
```

Out[25]:

	sars_cov_2_exam_result	respiratory_syncytial_virus	influenza_a	influenza_b	parainfluenza_1	coronavirusnl63	rhinovirus_enterovirus	coronavirus_hku1
1	0	not_detected	not_detected	not_detected	not_detected	not_detected	detected	not_detected
4	0	not_detected	not_detected	not_detected	not_detected	not_detected	detected	not_detected
8	0	not_detected	not_detected	not_detected	not_detected	not_detected	not_detected	not_detected
9	0	not_detected	not_detected	not_detected	not_detected	not_detected	not_detected	not_detected
13	0	not_detected	not_detected	not_detected	not_detected	not_detected	not_detected	not_detected
...	...	...	...	...	...	...	...	...
5602	0	not_detected	not_detected	not_detected	not_detected	not_detected	not_detected	not_detected
5607	0	detected	not_detected	not_detected	not_detected	not_detected	not_detected	not_detected
5614	0	not_detected	not_detected	not_detected	not_detected	not_detected	not_detected	not_detected
5615	0	not_detected	not_detected	not_detected	not_detected	not_detected	not_detected	not_detected
5618	0	not_detected	not_detected	not_detected	not_detected	not_detected	not_detected	not_detected

1352 rows × 18 columns

Figure 4 Index 0 sample data

```
dfs[1]
```

	sars_cov_2_exam_result	hematocrit	hemoglobin	platelets	mean_platelet_volume	red_blood_cells	lymphocytes	mean_corpuscular_hemoglobin_concentr
1	0	0.236515	-0.022340	-0.517413	0.010677	0.102004	0.318366	
8	0	-1.571682	-0.774212	1.429667	-1.672222	-0.850035	-0.005738	
18	0	0.991838	0.792188	0.072992	-0.550290	0.542763	0.045436	
28	0	1.014726	0.854844	-0.178244	0.796029	0.489872	-0.730707	
29	0	0.740064	0.854844	0.361914	-0.550290	0.436981	-0.227493	
...	...	...	...	...	...	...	...	
5602	0	0.190738	0.165628	-0.102873	0.908221	0.384090	-1.583611	
5614	0	-0.289922	-0.523588	0.663397	-0.774677	0.754327	-1.532437	
5615	0	0.717175	1.105468	-0.492289	-0.213711	0.613284	0.002791	
5618	0	-3.242548	-2.779203	-1.773594	-0.550290	-3.318285	-1.830953	
5643	1	0.694287	0.541564	-0.906829	-0.325903	0.578024	-0.295726	

420 rows × 17 columns

Figure 5 Index 2 sample data

After taking a closer look, index 0 (Figure 4) were having rows of non-numeric features thus we decided to drop it entirely. Coincidentally, index 5 (Figure 5) which is identical to the dataset used in the trail run was the perfect choice.

The source data has thus far proven changing numerical data samples with better PNLP compared to the trail data sample to further improve the results was not successful. Hence we have to use other methods such as over sampling before these existing samples are fitted into the corresponding training models.

```
# importing the stuff we need
import pandas as pd
import numpy as np
df = pd.read_csv('data\diagnosis-of-covid-19-and-its-clinical-spectrum.csv')
df
```

	patient_id	patient_age_quantile	sars_cov_2_exam_result	patient_addmited_to_regular_ward_1_yes_0_no	patient_addmited_to_semi_intensive_unit_1_
0	44477f75e8169d2	13	negative	f	
1	126e9dd13932f68	17	negative	f	
2	a46b4402a0e5696	8	negative	f	
3	f7d619a94f97c45	5	negative	f	
4	d9e41465789c2b5	15	negative	f	
...	...	...	...	...	
5639	ae66feb9e4dc3a0	3	positive	f	
5640	517c2834024f3ea	17	negative	f	
5641	5c57d6037fe266d	4	negative	f	
5642	c20c44766f28291	10	negative	f	
5643	2697fdccbfeb7f7	19	positive	f	

5644 rows × 111 columns

Figure 6 Raw source data

The raw source data had 5644 rows x 111 columns initially (Figure 6) and is filtered to be left with 420 rows x 17 columns (Figure 5). Information such as Patient's age, Ward situation, underlying diseases/ virus testing, body gas content, body metal content and urine testing was removed in data cleaning. These are some of the corresponding columns from the source data:

patient_id;	patient_addmited_to_regular_ward_1_yes_0_no;
patient_age_quantile;	patient_addmited_to_semi_intensive_unit_1_yes_0_no;
influenza_a;	patient_addmited_to_intensive_care_unit_1_yes_0_no;
influenza_b;	chlamydophila_pneumoniae;
parainfluenza_1;	urine_hemoglobin;
coronavirusnl63;	urine_bile_pigments;
coronavirus_hku1;	urine_ketone_bodies;
parainfluenza_3;	urine_nitrite;
adenovirus;	urine_urobilinogen;
parainfluenza_4;	urine_protein;
coronavirus229e;	urine_leukocytes;
coronavirusoc43;	urine_crystals;
inf_a_h1n1_2009;	urine_hyaline_cylinders;
bordetella_pertussis;	urine_granular_cylinders;
metapneumovirus;	urine_yeasts;
parainfluenza_2;	urine_color;
strepto_a;	respiratory_syncytial_virus;
myeloblasts;	rhinovirus_enterovirus;

The urine tests and underlying virus testing caught our attention initially but due to the lack of sample size we will have to remove them from the study.

```
df_filter.describe()
```

	sars_cov_2_exam_result	hematocrit	hemoglobin	platelets	mean_platelet_volume	red_blood_cells	lymphoc
count	5644.000000	6.030000e+02	6.030000e+02	6.020000e+02	5.990000e+02	6.020000e+02	6.020000e
mean	0.098866	-2.187396e-09	-1.598342e-08	-3.820598e-10	7.373957e-09	8.416943e-09	-7.863787
std	0.298509	1.000830e+00	1.000830e+00	1.000832e+00	1.000836e+00	1.000832e+00	1.000832e
min	0.000000	-4.501420e+00	-4.345603e+00	-2.552426e+00	-2.457575e+00	-3.970608e+00	-1.865070e
25%	0.000000	-5.188074e-01	-5.862439e-01	-6.053457e-01	-6.624832e-01	-5.679496e-01	-7.307069
50%	0.000000	5.340703e-02	4.031596e-02	-1.217160e-01	-1.015171e-01	1.385207e-02	-1.426696
75%	0.000000	7.171751e-01	7.295320e-01	5.314981e-01	6.838353e-01	6.661759e-01	5.976919
max	1.000000	2.662704e+00	2.671868e+00	9.532034e+00	3.713052e+00	3.645706e+00	3.764100e

Figure 7 Description of source data

Figure 7 suggests that the target range data from the source data has all been standardized as the mean near 0 and standard deviation is 1 across the columns. As the standardization is based on the whole 5644x111 dataset, so it may not remain the same after we split the source data into smaller data sets.

```
dfs[1].describe()
```

	sars_cov_2_exam_result	hematocrit	hemoglobin	platelets	mean_platelet_volume	red_blood_cells	lymphocytes
count	420.000000	420.000000	420.000000	420.000000	420.000000	420.000000	420.000000
mean	0.140476	0.119184	0.112669	-0.055109	0.016286	0.076020	-0.002428
std	0.347895	0.962373	0.955912	0.949576	0.978845	0.972173	0.923168
min	0.000000	-4.066536	-3.844355	-2.075077	-2.120995	-3.635631	-1.830953
25%	0.000000	-0.473031	-0.460932	-0.617907	-0.662483	-0.466575	-0.705120
50%	0.000000	0.167850	0.165628	-0.159401	-0.101517	0.093189	0.015585
75%	0.000000	0.808730	0.792188	0.399599	0.683835	0.736697	0.599824
max	1.000000	2.662704	2.671868	9.532034	3.713052	3.645706	3.218241

Figure 8 Description of selected dataset (index 2)

As we predicted, there are slight deviations in terms of the mean and standard deviation across the features as we have further removed rows which contain nulls entirely without introducing average data.

2.6. Feature Engineering

2.6.1. Using correlation matrix to locate the relevant variables with covid-19 test

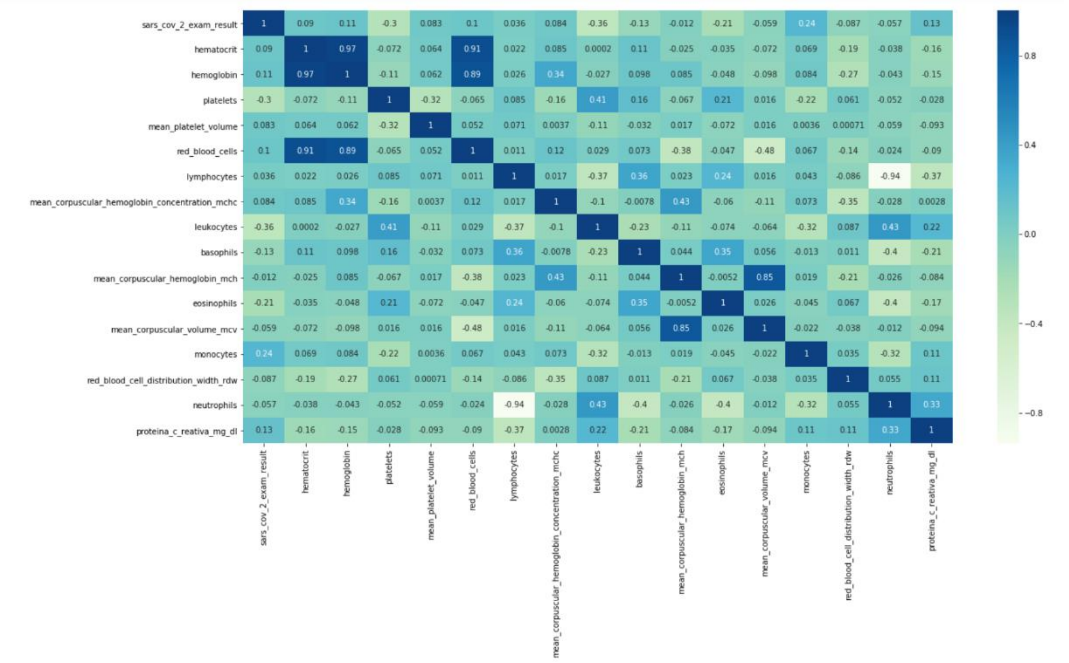


Figure 9 Pearson Correlation Matrix

None of the features is showing a high correlation with the covid-19 test result, since the number in the matrix is all between –0.5~0.5. Given weak relevance results, one single feature is not enough to tell if the patient is infected. However, chances are that after combining all the features, we can distinguish patient testing positively or negatively via Machine Learning model. Therefore, feature engineering and selection is quite significant before introducing data into training models.

```
removed = []

for c1 in candidates:
    for c2 in candidates:
        if (c1 not in removed) and (c2 not in removed):
            if c1!=c2:
                if corr_mat.loc[c1,c2] > 0.6 or corr_mat.loc[c1,c2] < -0.6:
                    removed.append(c1)
print("Removed: ", removed)
```

Figure 10 Trail run correlation matrix codes

```
#performing the pearson correlation matrix
def corrdf(df):
    import seaborn as sns
    import matplotlib.pyplot as plt

    print('#With Correlation Matrix\n')

    corr_mat = df.corr()
    plt.figure(figsize=(15,6))

    sns.heatmap(data=corr_mat,annot=True,cmap='GnBu')
    plt.show()

    target = 'sars_cov_2_exam_result'
    features = corr_mat.index.values
    features = features[features != target]

    removed = []

    for i in features:
        for j in features:
            if(j not in removed and i not in removed):
                if ((corr_mat[i].loc[[j]].values > 0.6) or (corr_mat[i].loc[[j]].values < -0.6)) and (corr_mat[i].loc[[j]].
                                                                                                      values != 1) :
                    removed.append(j)
    print('\nThe columns ',removed,'are removed\n')

    final_feature = [x for x in features if x not in removed]
    final_feature.insert(0,target)

    return df.loc[:,final_feature]
```

Figure 11 Refinement run correlation matrix codes

```
Removed:  ['hematocrit', 'hemoglobin', 'lymphocytes', 'mean_corpuscular_hemoglobin_mch']
```

Figure 12 Columns removed from trial run

```
The columns  ['hemoglobin', 'red_blood_cells', 'neutrophils', 'mean_corpuscular_volume_mcv'] are removed
```

Figure 13 Columns removed from refinement run

Interestingly, we noted during the trail run and refinement run, the columns which were removed using Pearson Correlation were not the same. Despite having the same 'filtering' logic and parameters (-0.6 and 0.6 as cut off correlation between features), the different code structure ultimately resulted in different columns to be remained in the dataset, hence changes final accuracy of the models slightly.

2.6.2. Using PCA to reduce the dimension

Principal Component Analysis (PCA) is another way streamline our model trainings. PCA turns the original number of features, K into lesser number of Principal Components (PC), N. As the explain variance increases as N gets bigger while maintaining N<K, we decided to have N number of PC shall the sum of all explained variance achieve 0.85.

```
def pcadf(df):

    from sklearn.decomposition import PCA

    print('#With PCA\n')

    best_n = {}
    for i in range(1,len(df.columns)):
        pca = PCA(n_components=i)
        pc = pca.fit_transform(df.iloc[:,1:])
        ratio_sum = pca.explained_variance_ratio_.sum()
        if ratio_sum > 0.85: #we only consider sum above 0.85
            best_n.update({i:ratio_sum})

    best_ratio_sum = min(best_n.values())

#     print(best_ratio_sum)

    for n,s in best_n.items():
#         print(n , 'components yield', s,'variance ratio sum') #can print this line if you want the details
        if s == best_ratio_sum:
            bestN=n

    pca = PCA(n_components=bestN)
    pc = pca.fit_transform(df.iloc[:,1:])

    print('The most optimal n is ',bestN,'\n')
    print('With variance ratio sum of ',pca.explained_variance_ratio_.sum(),'(minimum 0.85)\n')
    print('The variance ratios are\n', pca.explained_variance_ratio_,'\n')
    column_names = ['sars_cov_2_exam_result']
    for i in range(1,pc.shape[1]+1):
        column_names.append('PC'+str(i))

    pcadf = pd.DataFrame(np.append(np.reshape(df.iloc[:,[0]], (-1,1)), pc, axis=1))
    pcadf.columns = column_names
    return pcadf
```

Figure 14 PCA codes

Figure 14 shows the sample code from refinement run which is very similar to the trial run. Both codes generated the constant best N which is 8.

```
#With PCA

The most optimal n is  8

With variance ratio sum of  0.8504484015883913 (minimum 0.85)
```

Figure 15 PCA best N and explained variance sum from refinement run

```
Explained Variance ratio sum of n_components 7:  0.8101871191793107
Explained Variance ratio sum of n_components 8:  0.8626675153151294
Explained Variance ratio sum of n_components 9:  0.9067313771858302
```

Figure 16 PCA N(s) and explained variance sum from trial run

Figure 15 and Figure 16 shows the outcome PCA with both instances having the same data sample undergoing Pearson Correlation. The slight difference in explained variance sum is due to the Pearson Correlation giving data sample with different columns. As the Pearson correlation had proven only tightly correlated columns will have one of their counter parts removed, this should not cause significant impact to the PCA. Hence we do not redo the trail run.

2.6.3. Using Oversampling

We noted the inevitable low positive to negative label percentage (PNLP) in our ideal data set as shown in Figure 3, we have decided to introduced oversampling for the supervised learning as they are highly driven by the labels.

```
over_sampler(pcadf(corrdf(dfs[1])))
```

```
pos_y  (59,)
neg_y  (361,)
pos_y_train  (44,)
neg_y_train  (270,)
y_resampled  (540,)
Counter({0.0: 270, 1.0: 270})
```

Figure 17 Oversampling outcome

```
def over_sampler(dfs):
    import random
    from imblearn.over_sampling import RandomOverSampler
    a = random.randint(0,100000)

    ros = RandomOverSampler(random_state=a)

#splitting the original dataset into positive and negative labels
    pos_x = dfs.iloc[:,1:].loc[dfs['sars_cov_2_exam_result']==1,:]
    neg_x = dfs.iloc[:,1:].loc[dfs['sars_cov_2_exam_result']==0,:]
    pos_y = dfs.iloc[:,[0]].loc[dfs['sars_cov_2_exam_result']==1,:]
    neg_y = dfs.iloc[:,[0]].loc[dfs['sars_cov_2_exam_result']==0,:]
    pos_y = pos_y.iloc[:,0] #need to change this to a panda series as system expect a 1d array
    neg_y = neg_y.iloc[:,0] #need to change this to a panda series as system expect a 1d array
    print('pos_y ',pos_y.shape)
    print('neg_y ',neg_y.shape)

#splitting the subdivived dataset with the built in 25/75 ratio
    pos_x_train, pos_x_test, pos_y_train, pos_y_test = train_test_split(pos_x,pos_y,random_state=0)
    neg_x_train, neg_x_test, neg_y_train, neg_y_test = train_test_split(neg_x,neg_y,random_state=0)
    print('pos_y_train ',pos_y_train.shape)
    print('neg_y_train ',neg_y_train.shape)

#append all the evenly distributed labels and features back to a single array
    x_train = pos_x_train.append(neg_x_train)
    x_test = pos_x_test.append(neg_x_test)
    y_train = pos_y_train.append(neg_y_train)
    y_test = pos_y_test.append(neg_y_test)
#     print('y_train ',y_train.shape)

#removing the bias of the training dataset
    x_resampled, y_resampled = ros.fit_sample(x_train, y_train)
    print('y_resampled ', y_resampled.shape)
    print(collections.Counter(y_resampled))

#returning the over_sampled training and testing dataset
    return x_resampled,x_test, y_resampled,y_test
```

Figure 18 Oversampling codes

As Figure 17 indicates, we only have 59 positive labels vs 361 negative labels when the custom function – over_sampler from ‘Refinement Run 2’ (Figure 18) was first introduced with the data set. To add another layer of complication, most machine learning splits the dataset into training and testing subsets, so we had all features and labels pre-splat into training and testing set and only over sample the training set.

In order to address the small count of positive labels in the training set (44 positives vs 270 negatives), we will duplicate them to match the number of negative labels (270 vs 270 with 0 being negative and 1 being positive). This new data sample will then be introduced to the training models.

By doing so we avoid the same original sample being present in the training and testing set and the models would have been better trained with an equal amount of positive and negative samples. The testing set is not over sampled as we need not run the predictions with the same inputs more than once since they will provide the same prediction outcome.

Under ‘Refinement Run 1’ we have also introduced oversampling before data set is split to show the overall impact on the sensitivity and specificity.

2.7. Output

The following table summarizes the feature engineering applied to our models for discussion:

Feature Engineering mix	none	Corr(1)	PCA	Corr+PCA(1)	PCA+Corr(1)	Oversampling
Supervised learning	Yes	Yes	Yes	Yes	Yes	Yes
Unsupervised learning	Yes		Yes			

Legends - Corr: Correlation Matrix, PCA: Principal Component Analysis

(1) The outcome of Correlation Matrix may vary with the following column combination due to reasons discussed in 2.6:

Column Variation 1	Column Variation 2
sars_cov_2_exam_result	sars_cov_2_exam_result
~~hematocrit~~	hematocrit
~~hemoglobin~~	~~hemoglobin~~
platelets	platelets
mean_platelet_volume	mean_platelet_volume
red_blood_cells	~~red_blood_cells~~
~~lymphocytes~~	lymphocytes
mean_corpuscular_hemoglobin_concentration _mchc	mean_corpuscular_hemoglobin_concentration _mchc
leukocytes	leukocytes
basophils	basophils
~~mean_corpuscular_hemoglobin_mch~~	mean_corpuscular_hemoglobin_mch
eosinophils	eosinophils
mean_corpuscular_volume_mcv	~~mean_corpuscular_volume_mcv~~
monocytes	monocytes
red_blood_cell_distribution_width_rdw	red_blood_cell_distribution_width_rdw
neutrophils	~~neutrophils~~
proteina_c_reativa_mg_dl	proteina_c_reativa_mg_dl

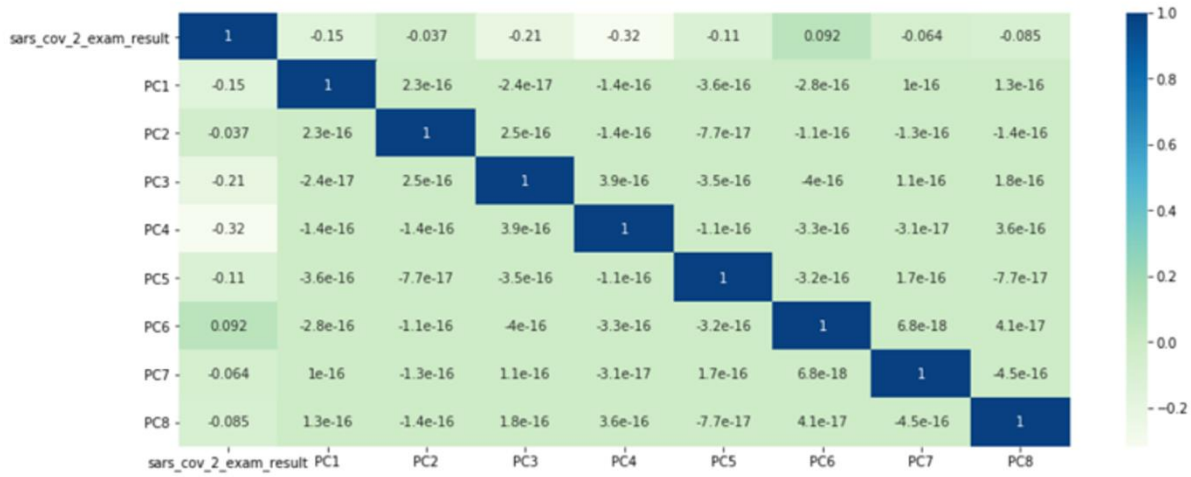
3. Supervised Machine Learning

3.1. Impact of data engineering & feature engineering (FE)

After the trail run, we have decided to introduce a series of data and feature engineering to the source and modelling dataset to see the changes in accuracy, sensitivity and specificity.

When running the FE of using principal component analysis then correlation matrix, we noted the following Pearson Correlation Matrix:

```
##Using both correlation matrix and PCA (with PCA first)

#With PCA

The most optimal n is  8

With variance ratio sum of  0.8771380365312372 (minimum 0.85)

The variance ratios are
 [0.21270762 0.18111298 0.12818513 0.1042022  0.07652559 0.0687402
 0.06036176 0.04530255]

#With Correlation Matrix
```




```
The columns  [] are removed
```

Figure 19 Pearson Correlation Matrix on a dataset with PCA applied

As PCA already transformed the dataset's dimension, by applying Correlation Matrix later has no effect at all as none of the new PCs have any tight correlation with each other. We would expect any data that has undergo PCA or PCA then correlation matrix will yield the exact same outcome after model training.

Any deviation from this assumption at the discussion of results would meant the training model had received different data due to the code structure of our program. Refer to Figure 20 for one of the FE sample codes undergoing 100 sets of data for testing. The rest can be found in the Refinement Run 2 file.

```
#the following test are for with PCA and correlation matrix (with PCA first)

print('##Using both correlation matrix and PCA (with PCA first) \n')
df=corrdf(pcadf(dfs[1]))

#reseting all parameters
LRacc,LRst,LRet,KNNacc,KNNst,KNNet,bestN,DTacc,DTst,DTet,NNacc,NNst,NNet = param_reset()

#looping all 4 training models 100 times
for i in range(100):

    #preparing the common data
    x_train,x_test,y_train,y_test = over_sampler(df)


    #using Logistic Regression
    logReg = LogisticRegression(solver = 'lbfgs')
    LRst.append(time.time())
    logReg.fit(x_train,y_train)
    LRet.append(time.time())
    LRy_pred = logReg.predict(x_test)
    LRacc.append(accuracy_score(y_test,LRy_pred))

    #using KNN
    KNNy_pred,KNNsubacc,bestn,KNNst,KNNet = bestknn(x_train,y_train,x_test,y_test)
    bestN.append(bestn)
    KNNacc.append(KNNsubacc)

    #using DT
    dt = DecisionTreeClassifier(criterion = "gini", max_depth = 5)
    DTst.append(time.time())
    dt.fit(x_train, y_train)
    DTet.append(time.time())
    DTy_pred = dt.predict(x_test)
    DTacc.append(accuracy_score(y_test,DTy_pred))

    #using NN
    y_test_ohe,NNst,NNet,NNy_pred,loss,NNindvaccuracy = bestnn(x_train,y_train,x_test,y_test)
    NNacc.append(NNindvaccuracy)

# calculation the time taken for all 4 models
LRdiff,KNNdiff,DTdiff,NNdiff = time_calculator(LRst,LRet,KNNst,KNNet,DTst,DTet,NNst,NNet)

#printing the results for all 4 models
result_printer(LRacc,LRdiff,LRy_pred,bestN,KNNacc,KNNdiff,KNNy_pred,
                   DTacc,DTdiff,DTy_pred,NNacc,NNdiff, NNy_pred)
```

Figure 20 Sample code of running 100 sets of data using PCA then Correlation Matrix

The discussion in the next section references figures of a combination of snippets from different results, the original output of the code can be found in Appendix 4 to Appendix 8.

3.1.1. Impact on Logistic Regression

No FE	#For Logistic Regresstion The average accuracy score is 0.8516981132075473 The standard deviation for accuracy score is 0.007788591487000109 The maximum accuracy is 0.8679245283018868 The minimum accuracy is 0.839622641509434 The average time taken is 0.0064955496788024905 The confusion matrix for the last iteration is [[11 4] [11 80]]
Only correlation matrix	#For Logistic Regresstion The average accuracy score is 0.852547169811321 The standard deviation for accuracy score is 0.009306210484283366 The maximum accuracy is 0.8773584905660378 The minimum accuracy is 0.8301886792452831 The average time taken is 0.004324028491973877 The confusion matrix for the last iteration is [[10 5] [12 79]]
Only PCA	#For Logistic Regresstion The average accuracy score is 0.8266981132075472 The standard deviation for accuracy score is 0.012980185902331897 The maximum accuracy is 0.8584905660377359 The minimum accuracy is 0.7830188679245284 The average time taken is 0.003482942581176758 The confusion matrix for the last iteration is [[10 5] [14 77]]
Correlation matrix then PCA	#For Logistic Regresstion The average accuracy score is 0.831509433962264 The standard deviation for accuracy score is 0.012516987389148665 The maximum accuracy is 0.8584905660377359 The minimum accuracy is 0.7924528301886793 The average time taken is 0.0034206247329711913 The confusion matrix for the last iteration is [[11 4] [14 77]]
PCA then correlation matrix	#For Logistic Regresstion The average accuracy score is 0.8287735849056603 The standard deviation for accuracy score is 0.014205868251297729 The maximum accuracy is 0.8584905660377359 The minimum accuracy is 0.7830188679245284 The average time taken is 0.003522951602935791 The confusion matrix for the last iteration is [[11 4] [14 77]]

Table 1 Summary of FE on Logistic Regression

Table 1 suggests that out of an average of 100 runs with different over sampled data, performing correlation matrix alone yield the highest accuracy score because the correlation matrix only keeps tightly correlated features and in the processes, similarly correlated features were removed which may initially have contained outliers which skewed the data.

The same FE yield a relatively higher sensitivity and specificity as shown in their individual confusion matrices as compared to those with PCAs but still lower than the non-FE, this could be due to with PCA introduced, a significant number of dimensions were reduced.

Given the time to run the training model with correlation matrix applied is shorter than without having FE while not losing too much of specificity nor sensitivity while gaining more accuracy, this seemed to be the best FE for the dataset when it comes to Logistic Regression.

3.1.2. Impact on K Nearest Neighbors

No FE	#For K Nearest Neighbours The most common n is n = 1 The average accuracy score is 0.8679245283018866 The standard deviation for accuracy score is 2.220446049250313e-16 The maximum accuracy is 0.8679245283018868 The minimum accuracy is 0.8679245283018868 The average time taken is 0.0020020008087158203 (excluding time to find the best n) The confusion matrix for the last iteration is [[7 8] [6 85]]
Only correlation matrix	#For K Nearest Neighbours The most common n is n = 1 The average accuracy score is 0.8490566037735852 The standard deviation for accuracy score is 2.220446049250313e-16 The maximum accuracy is 0.8490566037735849 The minimum accuracy is 0.8490566037735849 The average time taken is 0.0010008811950683594 (excluding time to find the best n) The confusion matrix for the last iteration is [[6 9] [7 84]]
Only PCA	#For K Nearest Neighbours The most common n is n = 1 The average accuracy score is 0.8490566037735852 The standard deviation for accuracy score is 2.220446049250313e-16 The maximum accuracy is 0.8490566037735849 The minimum accuracy is 0.8490566037735849 The average time taken is 0.0010008811950683594 (excluding time to find the best n) The confusion matrix for the last iteration is [[6 9] [7 84]]
Correlation matrix then PCA	#For K Nearest Neighbours The most common n is n = 1 The average accuracy score is 0.8490566037735852 The standard deviation for accuracy score is 2.220446049250313e-16 The maximum accuracy is 0.8490566037735849 The minimum accuracy is 0.8490566037735849 The average time taken is 0.0010008811950683594 (excluding time to find the best n) The confusion matrix for the last iteration is [[5 10] [6 85]]
PCA then correlation matrix	#For K Nearest Neighbours The most common n is n = 1 The average accuracy score is 0.8491509433962267 The standard deviation for accuracy score is 0.0009386673934968111 The maximum accuracy is 0.8584905660377359 The minimum accuracy is 0.8490566037735849 The average time taken is 0.001001596450805664 (excluding time to find the best n) The confusion matrix for the last iteration is [[6 9] [7 84]]

Table 2 Summary of FE on K Nearest Neighbors

For the K Nearest Neighbors (KNN) model, the number of neighbors, N plays a key part in the outcome of the result as shown in Table 2. All datasets regardless of FE yield the same most common N which is 1 across all 100 iterations. The overall algorithm to find the best N can be found in Figure 21.

```
#finding the best KNN model based on a input

def bestknn(x_train,y_train,x_test,y_test):
    import time
    bestacc = {}
    KNNst = []
    KNNet = []
    for i in range(1,int(x_train.shape[0]/2),2):
        #presuming best n will never be more than half of training rows to prevent underfitting
        knn = KNeighborsClassifier(n_neighbors=i)
        knn.fit(x_train,y_train)
        KNNy_pred = knn.predict(x_test)
        KNNacc = accuracy_score(y_test,KNNy_pred)
        bestacc.update({i:KNNacc})

    bestnacc = max(bestacc.values())

    for n,acc in bestacc.items():
#         print(n,acc) #only enable for single run else it still stall the loop
        if acc == bestnacc:
            bestn = n

    knn = KNeighborsClassifier(n_neighbors=bestn)
    KNNst.append(time.time())
    knn.fit(x_train,y_train)
    KNNet.append(time.time())
    KNNy_pred = knn.predict(x_test)
    KNNacc = accuracy_score(y_test,KNNy_pred)

    return KNNy_pred,KNNacc,bestn,KNNst,KNNet
```

Figure 21 Sample code for best N for KNN

As most of the results shown a low standard deviation and an identical accuracy, its safe to say for this dataset to undergo KNN, the FE doesn't really play a big part on the final accuracy, specificity nor sensitivity.

```
In [29]: bestknn(x_train,y_train,x_test,y_test)

1 0.8679245283018868
3 0.8490566037735849
5 0.7924528301886793
7 0.7830188679245284
9 0.7452830188679245
11 0.7735849056603774
13 0.7358490566037735
15 0.7547169811320755
17 0.7452830188679245
19 0.7641509433962265
21 0.7452830188679245
23 0.7358490566037735
25 0.7075471698113207
27 0.7075471698113207
29 0.7169811320754716
31 0.7075471698113207
```

Figure 22 N increases as accuracy decreases

The number of neighbors N however have a direct impact on the accuracy of the model. For this dataset when N increases the accuracy drops as shown in Figure 22.

3.1.3. Impact on Decision Tree

No FE	#For Decision Tree The average accuracy score is 0.8296226415094341 The standard deviation for accuracy score is 0.02171123475275282 The maximum accuracy is 0.8962264150943396 The minimum accuracy is 0.7924528301886793 The average time taken is 0.001950995922088623 The confusion matrix for the last iteration is [[9 6] [14 77]]
Only correlation matrix	#For Decision Tree The average accuracy score is 0.8416981132075473 The standard deviation for accuracy score is 0.02945995465787899 The maximum accuracy is 0.9245283018867925 The minimum accuracy is 0.7924528301886793 The average time taken is 0.001501767635345459 The confusion matrix for the last iteration is [[9 6] [13 78]]
Only PCA	#For Decision Tree The average accuracy score is 0.7630188679245282 The standard deviation for accuracy score is 0.0487340630356674 The maximum accuracy is 0.8773584905660378 The minimum accuracy is 0.660377358490566 The average time taken is 0.001621265411376953 The confusion matrix for the last iteration is [[9 6] [17 74]]
Correlation matrix then PCA	#For Decision Tree The average accuracy score is 0.819811320754717 The standard deviation for accuracy score is 0.016954906373218323 The maximum accuracy is 0.8679245283018868 The minimum accuracy is 0.7547169811320755 The average time taken is 0.0015216898918151856 The confusion matrix for the last iteration is [[10 5] [15 76]]
PCA then correlation matrix	#For Decision Tree The average accuracy score is 0.7634905660377358 The standard deviation for accuracy score is 0.03971810153934404 The maximum accuracy is 0.8584905660377359 The minimum accuracy is 0.6886792452830188 The average time taken is 0.0015310359001159668 The confusion matrix for the last iteration is [[10 5] [28 63]]

Table 3 Summary of FE on Decision Tree

Based on Table 3, if we were to compare the standard deviation of no FE against all the other variations, those with PCA have the greatest standard deviation and lowest accuracy after 100 runs. We believe that this is because the PCA have change the values of each and every row in the training data as compared to the no FE dataset causing the model to lose biological logic between the features and the label.

Correlation matrix however, seemed to have improved the accuracy score. One possibility is that the tightly correlated columns which were removed may have streamlined the decision making process from root to nodes while maintaining the biological relationship between the features and the label.

3.1.4. Impact on Neural Network

No FE	#For Neural Network The average accuracy score is 0.873396229147911 The standard deviation for accuracy score is 0.01486739901303666 The maximum accuracy is 0.9056603908538818 The minimum accuracy is 0.8396226167678833 The average time taken is 6.27070426940918 The confusion matrix for the last iteration is [[9 6] [8 83]]
Only correlation matrix	#For Neural Network The average accuracy score is 0.8616037750244141 The standard deviation for accuracy score is 0.010424848777302265 The maximum accuracy is 0.8867924809455872 The minimum accuracy is 0.8396226167678833 The average time taken is 6.377801418304443 The confusion matrix for the last iteration is [[10 5] [11 80]]
Only PCA	#For Neural Network The average accuracy score is 0.809811326265335 The standard deviation for accuracy score is 0.014475521315720438 The maximum accuracy is 0.849056601524353 The minimum accuracy is 0.7830188870429993 The average time taken is 5.728201866149902 The confusion matrix for the last iteration is [[10 5] [15 76]]
Correlation matrix then PCA	#For Neural Network The average accuracy score is 0.8107547253370285 The standard deviation for accuracy score is 0.014905661306811936 The maximum accuracy is 0.849056601524353 The minimum accuracy is 0.7735849022865295 The average time taken is 6.159604549407959 The confusion matrix for the last iteration is [[9 6] [16 75]]
PCA then correlation matrix	#For Neural Network The average accuracy score is 0.8082075530290603 The standard deviation for accuracy score is 0.013609452724665093 The maximum accuracy is 0.849056601524353 The minimum accuracy is 0.7830188870429993 The average time taken is 6.031477451324463 The confusion matrix for the last iteration is [[10 5] [17 74]]

Table 4 Summary of FE on Decision Tree

The Neural Network model yields the best result when no FE is applied to its dataset. This is could be because the over sampled data which is balanced (equal positives and negatives) have trained the model enough to distinguish the two possible outcomes. By applying any FE would mean to remove more dimension of the training dataset hence reducing the input layer of the model itself leading to a lower accuracy score.

This is proven true as for the case of applying correlation matrix alone, the input layer is reduced from 16 to 12 (4 feature were tightly correlated to their counterparts) where as the rest of the FE are reducing the input layer from 16 to 8 (using a PCA of n=8 for minimal sum of explained variance 0.85). This is further proven consistent as those with PCA involved have a similar average accuracy score.

3.2. Logistic Regression result and resampling

In the following parts of different supervised learning model, we only take the data after original data cleaning and feature selection based on Pearson Correlation Matrix.

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
y = data.iloc[:, 0].values.astype("int")
x = data.iloc[:, 1:]
x_train, x_test, y_train, y_test =train_test_split(x, y,random_state =42)
logReg = LogisticRegression(solver="lbfgs", random_state =0)
start_time = time.time()
logReg.fit(x_train, y_train)
end_time = time.time()
print('duration of training (Log Regression):',end_time-start_time)
#VALIDATION
y_pred = logReg.predict(x_test)
print('accuracy score: %.5f' % accuracy_score(y_test, y_pred))
from sklearn.metrics import confusion_matrix
print('confusion matrix: ')
print(confusion_matrix (y_test, y_pred, labels = [1,0]))
```

```
duration of training (Log Regression): 0.007050991058349609
accuracy score: 0.84762
confusion matrix:
[[ 7 12]
 [ 4 82]]
```

Figure 23 Logistic Regression result based on original data cleaning and feature selection

As shown in confusion matrix, *True Positive Rate* (sensitivity) is only 7/19=0.368, which is far lower than accuracy score (range between 0.8~0.9).

Since the ratio of negative and positive cases is nearly 6-to-1, we believe this may lead to biased predictions in ML model. Therefore, we introduce over-sampling to resize the positive and negative cases in order to get a balanced training data group.

```
#resample command
from imblearn.over_sampling import RandomOverSampler
ros = RandomOverSampler(random_state=0)
x_resampled, y_resampled = ros.fit_sample(x, y)
start_time = time.time()
end_time = time.time()
x_train_re, x_test_re, y_train_re, y_test_re =train_test_split(x_resampled, y_resampled,random_state =42)
start_time = time.time()
logReg.fit(x_train_re, y_train_re)
end_time = time.time()
print('duration of training (Log Regression):',end_time-start_time)
#VALIDATION
y_pred_re = logReg.predict(x_test_re)
print('accuracy score: %.5f' % accuracy_score(y_test_re, y_pred_re))
from sklearn.metrics import confusion_matrix
print('confusion matrix: ')
print(confusion_matrix (y_test_re, y_pred_re, labels = [1,0]))
```

```
duration of training (Log Regression): 0.004033803939819336
accuracy score: 0.85083
confusion matrix:
[[80  7]
 [20 74]]
```

Figure 24 Logistic Regression result after resampling based on original data cleaning and feature selection

After applying resampling, *True Positive Rate* has been improved above 0.8 which is almost as the accuracy score.

3.3. KNN result and resampling

For K-Nearest Neighbor (k-NN) classification model, we run a loop to find optimized number of nearest neighbor as the k parameter.

```
from sklearn.neighbors import KNeighborsClassifier
accuracyScoreKnn = []
truePosRateKnn = []
for i in range(1,50):
    knn_model = KNeighborsClassifier(n_neighbors = i)
    knn_model.fit(x_train, y_train)
    y_pred = knn_model.predict(x_test)
    accuracyScoreKnn.append(accuracy_score(y_test, y_pred))
    cfnMat = confusion_matrix(y_test, y_pred, labels =[1, 0])
    truePosRateKnn.append(cfnMat[0,0]/(cfnMat[0,0]+cfnMat[0,1]))
```

Figure 25 Find best k for K-NN model based on original data cleaning

Then the model is trained after selecting k as 3:

```
knn_model = KNeighborsClassifier(n_neighbors = 3)
start_time = time.time()
knn_model.fit(x_train, y_train)
end_time = time.time()
print('duration of training (KNN):',end_time-start_time)
y_pred = knn_model.predict(x_test)
print('accuracy score: %.5f' % accuracy_score(y_test, y_pred))
print('confusion matrix: ')
print(confusion_matrix(y_test, y_pred, labels =[1, 0]))
cfnMat = confusion_matrix(y_test, y_pred, labels =[1, 0])
```

```
duration of training (KNN): 0.0014028549194335938
accuracy score: 0.88571
confusion matrix:
[[10  9]
 [ 3 83]]
```

Figure 26 K-NN model result based on original data cleaning

Applying resampling also helps to improve *True Positive Rate (sensitivity).*

```
knn_model = KNeighborsClassifier(n_neighbors = 3)

start_time = time.time()
knn_model.fit(x_train_re, y_train_re)
end_time = time.time()
print('duration of training (KNN):',end_time-start_time)

y_pred_re = knn_model.predict(x_test_re)
print('accuracy score: %.5f' % accuracy_score(y_test_re, y_pred_re))
print('confusion matrix: ')
print(confusion_matrix (y_test_re, y_pred_re, labels = [1,0]))
```

```
duration of training (KNN): 0.0022187232971191406
accuracy score: 0.88398
confusion matrix:
[[87  0]
 [21 73]]
```

Figure 27 K-NN model result after resampling based on original data cleaning

Here we plot the accuracy level under different number of neighbors to show how oversampling can do to improve the overall accuracy but also can lead to overfitting since the learning model is actually memorizing certain training data.

Introduce oversampling to improve the accuracy in positivie results

```
from sklearn.neighbors import KNeighborsClassifier
accuracyScoreKnnReS = []
truePosRateKnnReS = []
for i in range(1,50):
    knn_model = KNeighborsClassifier(n_neighbors = i)
    knn_model.fit(x_train_re, y_train_re)
    y_pred_re = knn_model.predict(x_test_re)
    accuracyScoreKnnReS.append(accuracy_score(y_test_re, y_pred_re))
    cfnMat = confusion_matrix(y_test_re, y_pred_re, labels =[1, 0])
    truePosRateKnnReS.append(cfnMat[0,0]/(cfnMat[0,0]+cfnMat[0,1]))
```

```
plt.plot(range(1,50), accuracyScoreKnn,label='accuracy score')
plt.plot(range(1,50), truePosRateKnn,label='true positive rate')
plt.plot(range(1,50), accuracyScoreKnnReS,label='accuracy score after oversampling')
plt.plot(range(1,50), truePosRateKnnReS,label='true positive rate after oversampling')
plt.title('K-NN')
plt.xlabel('Number of neighbors')
plt.ylabel('Accuracy Validation')
plt.ylim(0,1)
plt.legend()
```

Figure 28 comparing accuracy level before and after resampling based on original data cleaning

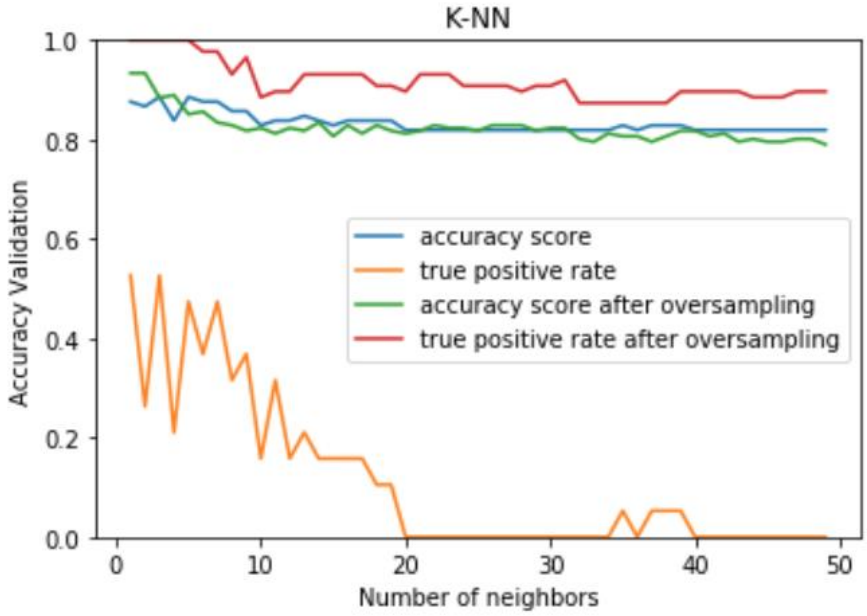


Figure 29 K-NN accuracy level of different parameter before and after resampling based on original data cleaning and feature selection

This figure shows both accuracy score and true positive rate before and after resampling. Even if oversampling doesn't improve accuracy score, it will get us better sensitivity, which is a minor class in our original dataset. Therefore, introducing oversampling is a double-edged sword: it works but also can be considered as overfitting.

3.4. Decision Tree Model Result

Find a group of parameters to get higher accuracy

```
from sklearn.tree import DecisionTreeClassifier
def decTreeScore(crit = 'gini',  maxDepth = None, minSamples = 1, minSplit = 2):
    decTree = DecisionTreeClassifier(criterion = crit, max_depth = maxDepth, min_samples_leaf = minSamples,
                                     min_samples_split = minSplit, random_state= 42)
    decTree.fit(x_train, y_train)
    accuracy = accuracy_score(y_test, decTree.predict(x_test))
    return accuracy
```

```
opt_maxDepth = 1
opt_minSamples = 1
opt_minSpilt = 1
opt_score = 0
for i in np.arange(1,10):
    for j in np.arange(1,30):
        for k in np.arange(2,30):
            if decTreeScore(maxDepth=i,minSamples = j, minSplit = k) > opt_score:
                opt_maxDepth = i
                opt_minSamples = j
                opt_minSpilt = k
                opt_score = decTreeScore(maxDepth=i,minSamples = j, minSplit = k)
print("ijk:",opt_maxDepth ," ",opt_minSamples," ",opt_minSpilt)
print(opt_score)
```

```
ijk: 5   1   2
0.8571428571428571
```

Figure 30 Parameter Selection of Decision Tree Model based on original data cleaning and feature selection

```
#Accuracy of Optimized Param
decTree = DecisionTreeClassifier(criterion = "gini", max_depth = 5, min_samples_leaf = 1,
                                 min_samples_split = 2, random_state= 42)
start_time = time.time()
decTree.fit(x_train, y_train)
end_time = time.time()
print('duration of training (Decision Tree):',end_time-start_time)

y_pred = decTree.predict(x_test)
accuracy = accuracy_score(y_test, y_pred)
print(accuracy)
```

```
duration of training (Decision Tree): 0.003131866455078125
0.8571428571428571
```

```
#-------->VISUAL RESULT
from sklearn import tree
import graphviz
from graphviz import Source
Source(tree.export_graphviz(decTree, out_file=None, class_names=None, feature_names= x_train.columns))
```

Figure 31 Decision Tree Model Training based on original data cleaning and feature selection

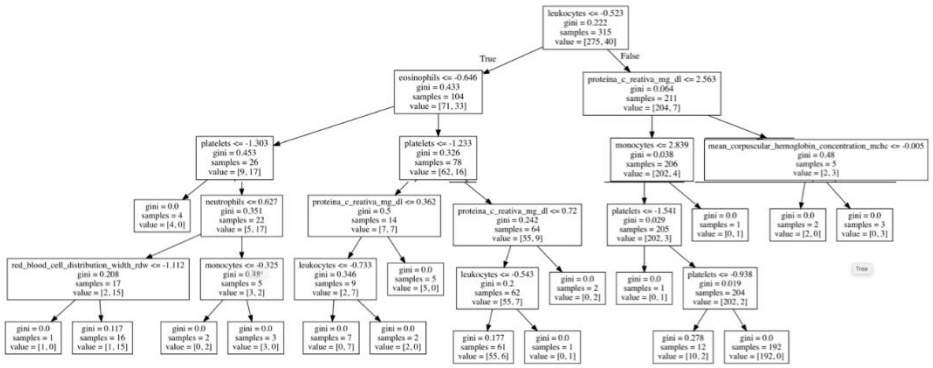


Figure 32Decision Tree Model Training based on original data cleaning and feature selection

We also generate the visualization graph of our optimal decision tree model with max depth as 5. The accuracy is still high and training time is shorter than logistic regression. It may help scientists and doctors to make the right judgement during diagnosis.

3.5. Neural Network Result

```
from keras.layers.core import Dense, Activation
from keras.models import Sequential

y_train_ohe = np.array([[1,0] if x== 1 else [0,1] for x in y_train])
y_test_ohe = np.array([[1,0] if x== 1 else [0,1] for x in y_test])

model = Sequential()
model.add(Dense(500, input_shape=(x_train.shape[1],),activation='sigmoid'))
#relu
model.add(Dense(2,activation='softmax'))
model.compile(optimizer='adam',loss='categorical_crossentropy',metrics=["accuracy"])
start_time = time.time()
model.fit(x_train, y_train_ohe,batch_size=64,epochs=500)
end_time = time.time()
print('duration of training (Neural Network):',end_time-start_time)

#evaluation
loss, accuracy = model.evaluate(x_test, y_test_ohe)
print('Loss = ', loss, ', Accuracy = ', accuracy)
```

Figure 33 Neural Network Model Training based on original data cleaning and feature selection

```
Epoch 498/500
315/315 [==============================] - 0s 40us/step - loss: 0.1834 - accuracy: 0.9079
Epoch 499/500
315/315 [==============================] - 0s 39us/step - loss: 0.1853 - accuracy: 0.9175
Epoch 500/500
315/315 [==============================] - 0s 46us/step - loss: 0.1842 - accuracy: 0.9143
duration of training (Neural Network): 7.955780982971191
105/105 [==============================] - 0s 316us/step
Loss =  0.3006168816770826 , Accuracy =  0.8571428656578064
```

Figure 34 NN Training Result based on original data cleaning and feature selection

When it comes to Neural Network, it is hard to find a group of optimal parameters setting to get higher score. After changing activation function into “relu” , it run a little bit faster but accuracy remains:

duration of training (Neural Network): 7.7241599559783936

Loss = 0.8384216183707828 , Accuracy = 0.8571428656578064

Neural network is the most time-cost model among the last more models.

3.6. Conclusion

There is no obvious gap among those models, but KNN and Decision Tree can generate results very fast with a not bad accuracy.

The results for KNN shown us that accuracy, specificity nor sensitivity wasn't affected much by the FE applied whereas number of neighbors, N being the most important criteria.

We acknowledge that different dataset (training data feature variations) does provide different results and there is no one straightforward way to determine which kind of FE best suite which kind of model. For future studies, it will be advisable to still try all FE variations on all training data and model choices in order to get the best accurate, sensitive and specific results.

4. Unsupervised Machine Learning

4.1. K-Means with PCA

The libraries and dataset are imported in the first step.

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.cluster import KMeans
import seaborn as sb
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
```

In [2]:
```
df = pd.read_csv("diagnosis-of-covid-19-and-its-clinical-spectrum.csv")
df
```

Out[2]:

	patient_id	patient_age_quantile	sars_cov_2_exam_result	patient_addmited_to_regular_ward_1_yes_0_no	patient_addmited_to_semi_intensive_unit_1_
0	44477f75e8169d2	13	negative	f	
1	126e9dd13932f68	17	negative	f	
2	a46b4402a0e5696	8	negative	f	
3	f7d619a94f97c45	5	negative	f	
4	d9e41465789c2b5	15	negative	f	
...	...	...	...	...	
5639	ae66feb9e4dc3a0	3	positive	f	
5640	517c2834024f3ea	17	negative	f	
5641	5c57d6037fe266d	4	negative	f	
5642	c20c44766f28291	10	negative	f	
5643	2697fdccbfeb7f7	19	positive	f	

5644 rows × 111 columns

From over hundreds of columns, the features are reduced by using PCA technique and got 8 dimensions to compute.

Out[15]:

	Class	PC1	PC2	PC3	PC4	PC5	PC6	PC7	PC8
0	0.0	1.108064	0.230495	0.239158	0.307115	-0.134055	0.194245	-0.282053	-1.493653
1	0.0	0.492941	0.210027	-2.438205	-2.666281	1.139606	1.626717	0.655840	1.033698
2	0.0	-0.812625	0.224659	0.083754	0.104292	0.222804	-0.910387	-0.397893	-0.020054
3	0.0	0.593251	0.166147	0.259957	0.380827	-0.976943	-0.357808	-0.216468	0.476698
4	0.0	-0.547842	-0.172084	-0.649295	-0.891485	0.253947	-0.498507	0.139637	-0.115486
...	...	...	...	...	...	...	...	...	...
415	0.0	-1.258993	-0.311066	0.204224	1.178269	1.063325	2.567273	1.146835	-1.570694
416	0.0	-2.700731	0.871426	-1.179590	2.545280	3.318637	2.257960	1.854836	-1.330977
417	0.0	-0.367396	-1.830614	-0.571186	-1.059505	0.899325	0.119782	0.357931	-0.142506
418	0.0	-1.340245	-1.108054	3.216590	-2.732091	1.711285	1.493314	-0.751024	-0.056869
419	1.0	-0.708215	-1.200388	0.469669	0.330280	0.622348	-0.740965	-0.890821	-0.378060

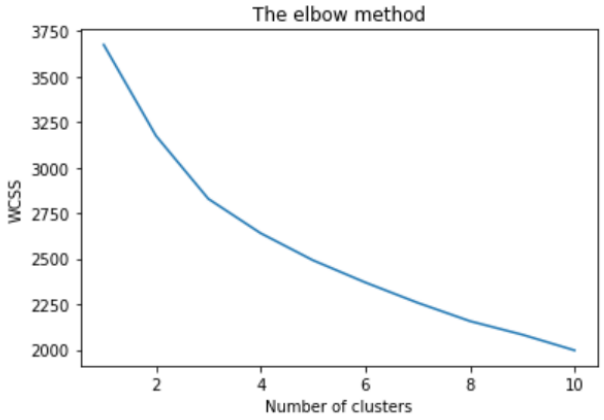
420 rows × 9 columns

```
In [22]: x_pca = PCA_data.iloc[:, 1:9].values
         x_pca
```

```
Out[22]: array([[ 1.10806364,  0.23049491,  0.23915767, ...,  0.19424473,
                  -0.28205322, -1.49365302],
                [ 0.49294129,  0.21002708, -2.43820514, ...,  1.62671712,
                   0.65584019,  1.03369774],
                [-0.81262531,  0.22465855,  0.08375385, ..., -0.91038664,
                  -0.39789269, -0.02005382],
                ...,
                [-0.36739633, -1.8306136 , -0.57118639, ...,  0.11978216,
                   0.3579306 , -0.14250618],
                [-1.34024526, -1.10805412,  3.21659004, ...,  1.49331445,
                  -0.75102395, -0.05686879],
                [-0.70821488, -1.20038755,  0.46966901, ..., -0.74096467,
                  -0.890821  , -0.37805961]])
```

From the PCA dataset, 8 out of 9 columns are selected as independent variable(x) except the class columns and 420 samples will be in the training model.

```
In [24]: #Plot and observe the elbow
         plt.plot(range(1, 11), wcss)
         plt.title('The elbow method')
         plt.xlabel('Number of clusters')
         plt.ylabel('WCSS')
         plt.show()
```



In this graph, we illustrate to determine the optimal k (number of clusters) by using the elbow method. We plot WCSS (Within Clusters Sum of Squares) against number of clusters and the line graph slightly declines so we assumed that k = 3 can be applied in the k-means model. Unfortunately, the critical point is not obvious enough in our application, so we tried k = 2,3,4 for illustration.

By using two dimensions, we plotted the data for visualization.

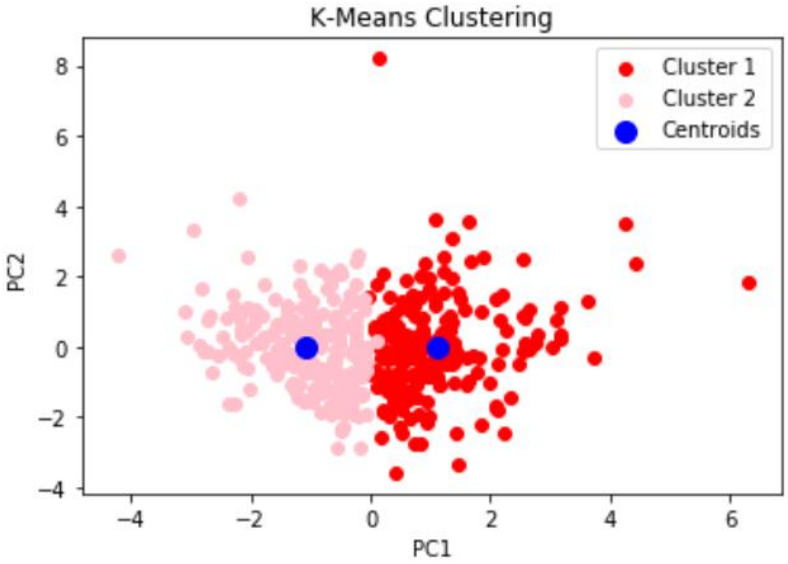


Figure 35 Scatter Plot With 2 Clusters (With PCA)

In above figure, the data trained with PCA is visualized by k=2 and we can see that the points in each cluster are separated significantly and point it out with respective centroids.

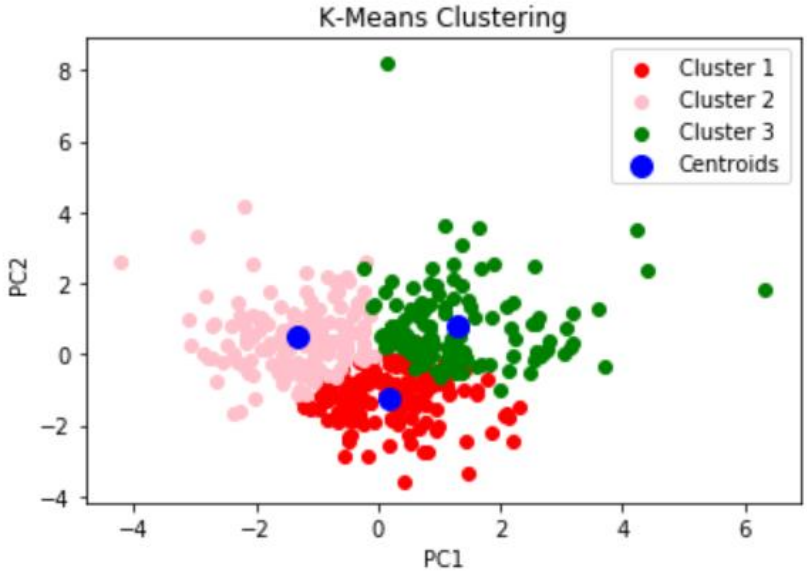


Figure 36 Scatter Plot With 3 Clusters (With PCA)

In above figure, the data trained with PCA is visualized by k=3 and we can see that the points in each cluster are fairly separated with less overlapping and classified into 3 of clusters with the corresponding centroids and the data seems to illustrate with more specific observations.

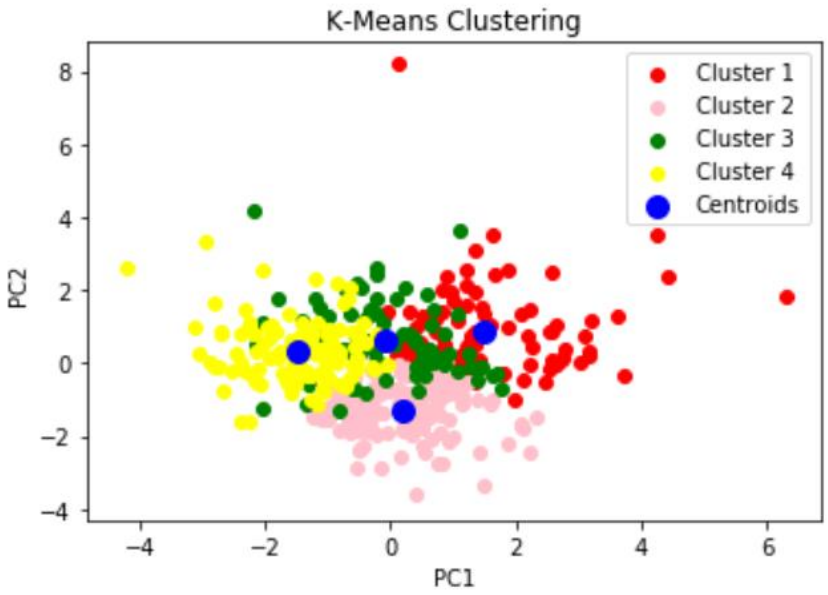


Figure 37 Scatter Plot With 4 Clusters (With PCA)

In above figure, the data trained with PCA is visualized by k=4 and we can see that the clusters are overlapping and the points seems not to be classified well and also it cannot clearly identify where the centroids belong to.

4.1.1. Duration of training k-means model

The duration time of k-means with PCA is computed based on 3 different clusters.

k-Means with PCA

Duration of training model

In [24]:
```
kmeans_model2 = KMeans(n_clusters =2)
start_time = time.time()
kmeans_model2.fit(x_pca)
end_time = time.time()
print((end_time - start_time))
```

```
0.03646039962768555
```

In [27]:
```
kmeans_model3 = KMeans(n_clusters =3)
start_time = time.time()
kmeans_model3.fit(x_pca)
end_time = time.time()
print((end_time - start_time))
```

```
0.055882930755615234
```

In [21]:
```
kmeans_model4 = KMeans(n_clusters =4)
start_time = time.time()
kmeans_model4.fit(x_pca)
end_time = time.time()
print((end_time - start_time))
```

```
0.0747683048248291
```

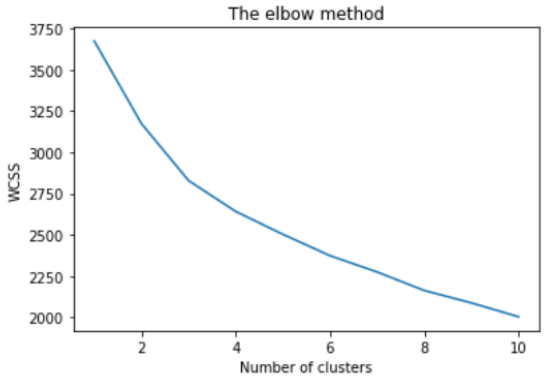
4.2. K-Means without PCA

```
In [47]: x_nopca = df.iloc[:, 1:17].values
         x_nopca
```

```
Out[47]: array([[ 0.23651545, -0.02234027, -0.51741302, ..., -0.62507266,
                  -0.61908603, -0.14789495],
                [-1.57168222, -0.774212  ,  1.42966747, ..., -0.97889912,
                  -0.12739536, -0.28698576],
                [ 0.99183822,  0.79218763,  0.07299204, ...,  0.1710353 ,
                   0.26595679, -0.48767394],
                ...,
                [ 0.71717513,  1.10546756, -0.49228939, ..., -1.15581191,
                  -0.06183668,  0.5614683 ],
                [-3.24254799, -2.77920342, -1.7735939 , ..., -0.44815987,
                   1.55254781,  0.60915661],
                [ 0.69428688,  0.54156393, -0.90682912, ..., -0.18279028,
                   0.38068476, -0.50357002]])
```

The 17 columns of clean dataset are selected as dependent variable(x) except the Class column.

```
In [48]: plt.plot(range(1, 11), wcss)
         plt.title('The elbow method')
         plt.xlabel('Number of clusters')
         plt.ylabel('WCSS')  # within cluster sum of squares
         plt.show()
```



In this above elbow graph, it shows the number of clusters to choose relevant cluster number for k and can defined k with possible 2 and 3.

By using two dimensions, we plotted the data for visualization and the model does not seem to be good enough for the model.

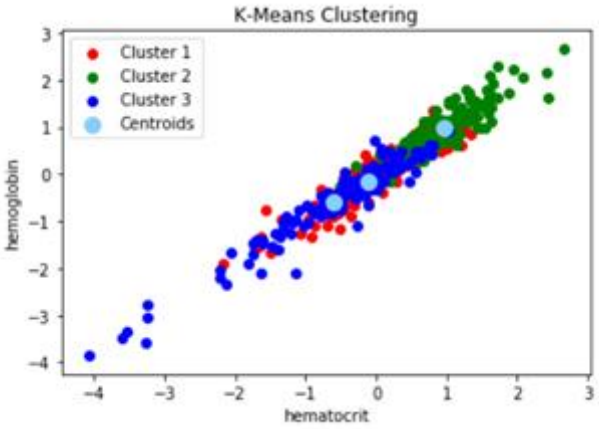


Figure 38 Scatter Plot With 3 Clusters (Without PCA)

In above Figure 38, the 3 clusters are overlapping and difficult to explain that each centroid belongs to which clusters in the image when taking only two first variable components.

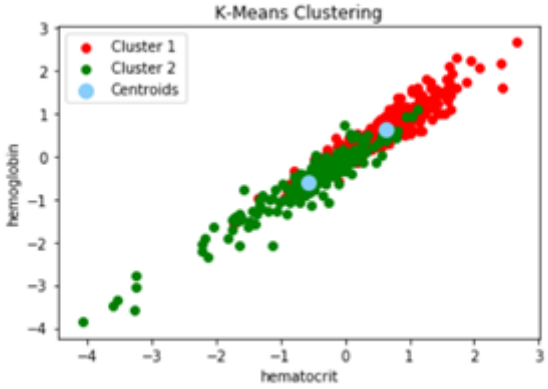


Figure 39 Scatter Plot With 2 Clusters (Without PCA)

In above Figure 39, when we compare k=2 with k=3, we can see that the centroids and clusters are decently separated and better visualization than that of 3 clusters.

4.2.1. Duration of training k-means model

The duration time of k-means without PCA is computed based on 2 different clusters.

k-Means without PCA

Duration of training Model

```
In [36]: kmeans_model_nopca2 = KMeans(n_clusters =2)
         start_time = time.time()
         kmeans_model_nopca2.fit(x_nopca)
         end_time = time.time()
         print((end_time - start_time))

         0.05285811424255371

In [37]: kmeans_model_nopca3 = KMeans(n_clusters =3)
         start_time = time.time()
         kmeans_model_nopca3.fit(x_nopca)
         end_time = time.time()
         print((end_time - start_time))

         0.07890939712524414
```

4.3. Hierarchical Clustering with PCA

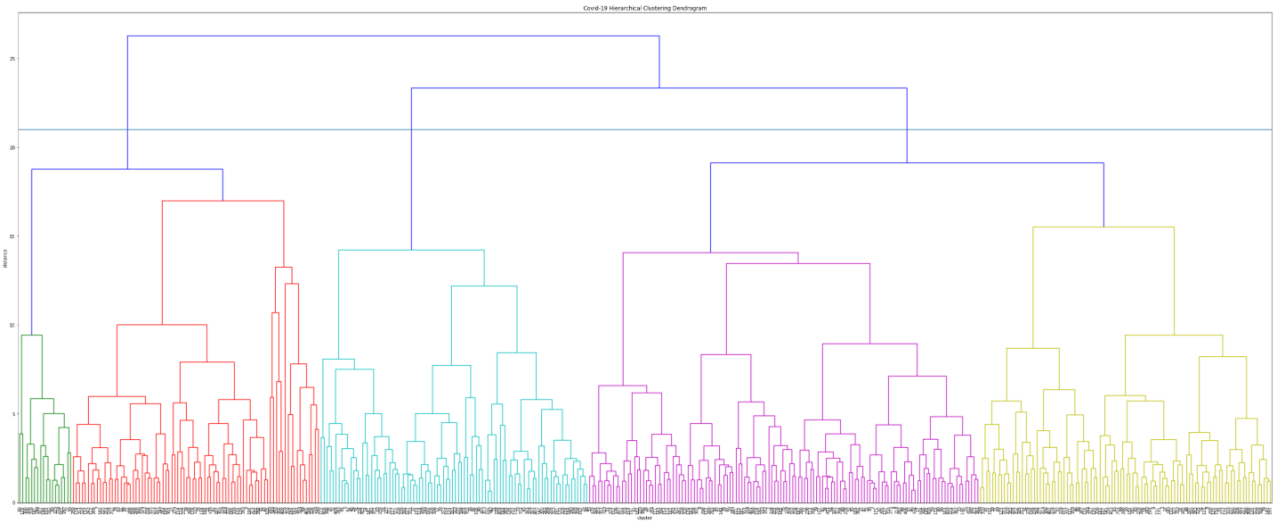


Figure 40 Dendrogram with PCA

From the figure above we can observe that 2 clusters and 3 clusters are decently spaced out from each other, we can reasonably deduce that the dataset could be divided into 2 or 3 clusters. To further justify, we plotted a scatter diagram to better visualize the clustering.

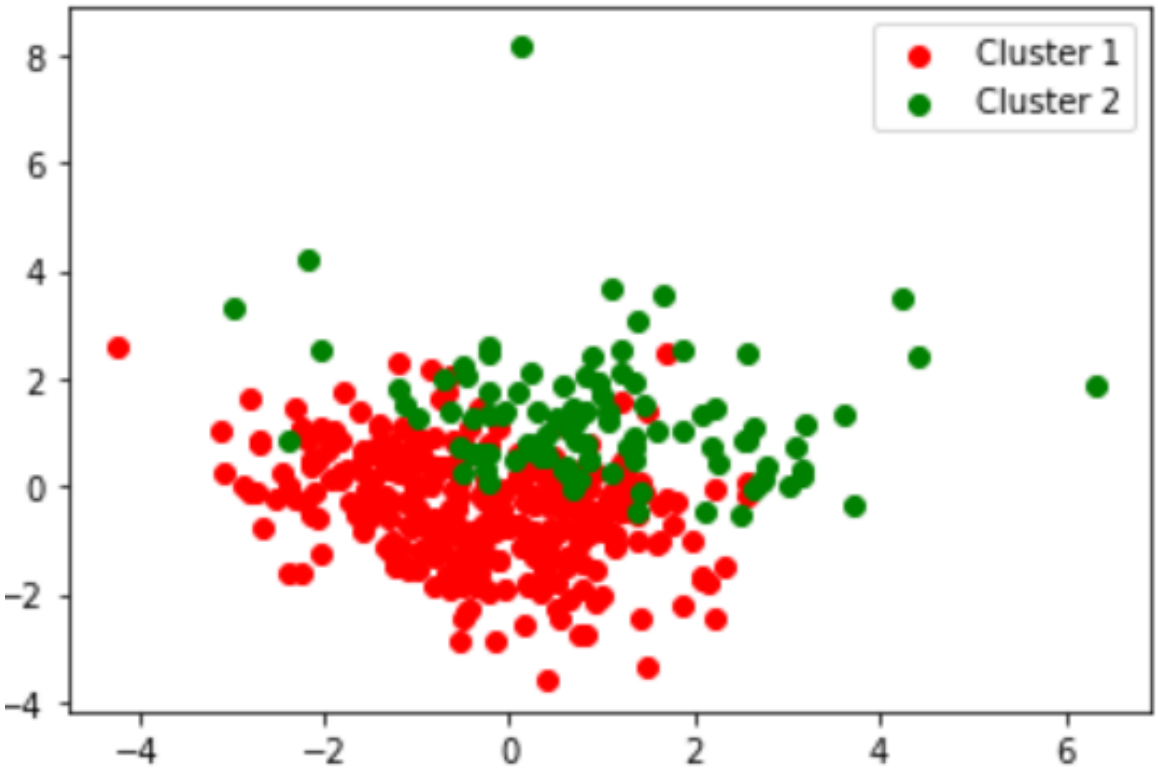


Figure 41 Scatter Plot with 2 Clusters

From the scatter diagram we can see that 2 clusters could be a fair representation of the data as they are well separated with little to no distinct overlapping of clusters.

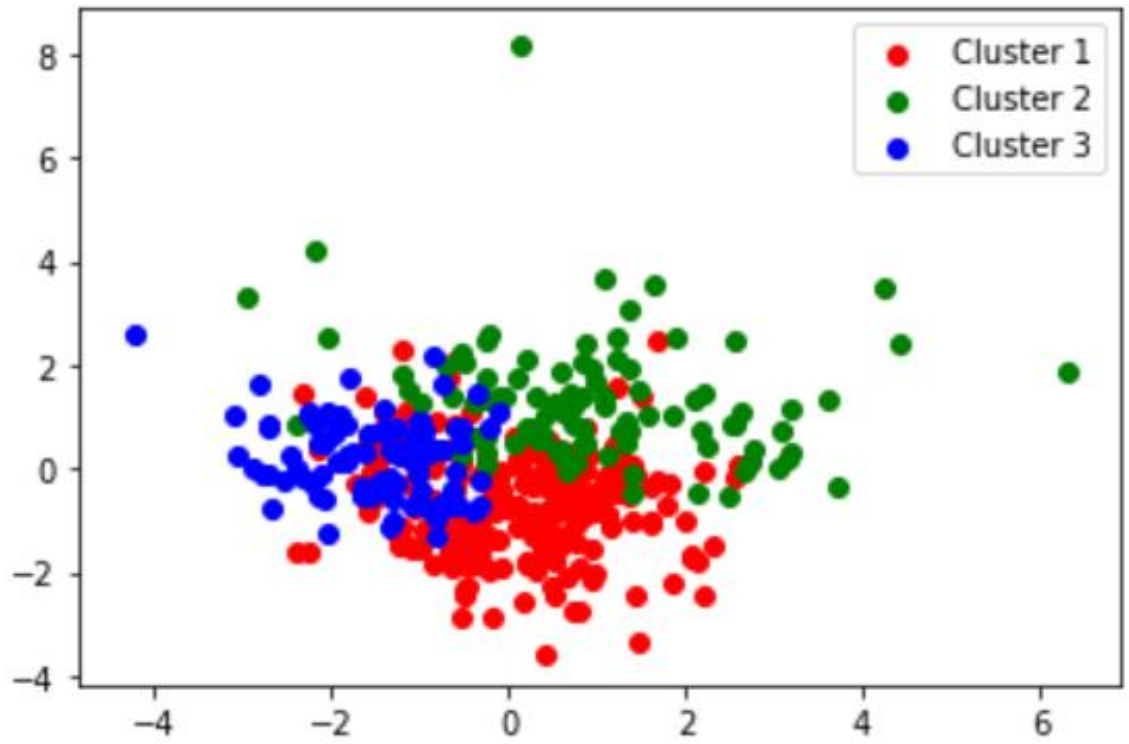


Figure 42 Scatter Plot with 3 Clusters

From the scatter diagram, we can see that the 3 clusters are decently separated with a little

overlapping between cluster 1 and cluster 3. It is also possible that the data have 3 clusters.

4.3.1. Duration of Training Hierarchical Clustering Model with PCA

```
import time
clustering = AgglomerativeClustering(linkage="ward", n_clusters=2)
start_time = time.time()
clustering.fit(x)
end_time = time.time()
print((end_time - start_time))
```

```
0.0463109016418457
```

Duration of training 2 clusters Hierarchical Clustering with PCA: 0.0463 seconds

```
import time
clustering = AgglomerativeClustering(linkage="ward", n_clusters=3)
start_time = time.time()
clustering.fit(x)
end_time = time.time()
print((end_time - start_time))
```

```
0.03896951675415039
```

Duration of training 3 clusters Hierarchical Clustering with PCA: 0.0390 seconds

4.4. Hierarchical Clustering without PCA

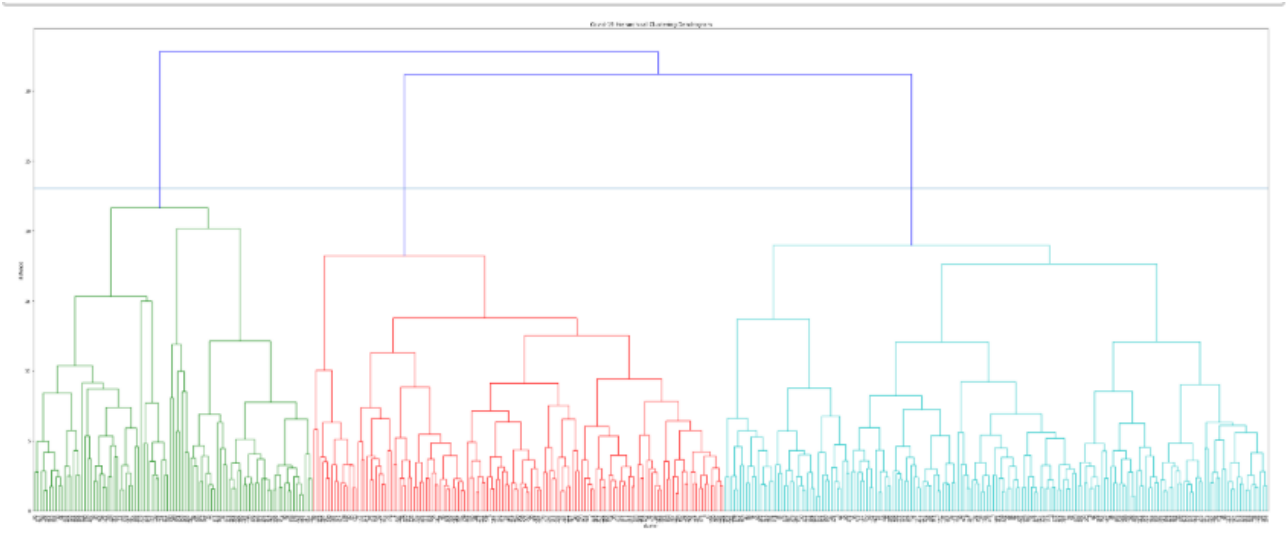


Figure 43 Dendrogram without PCA

From the Dendrogram above, we can observe that 3 clusters are decently spaced out and we will be plotting the scatter diagram to visualize it. Since the previous Hierarchical Clustering with PCA

discussed about 2 clusters, we will be plotting it for comparison.

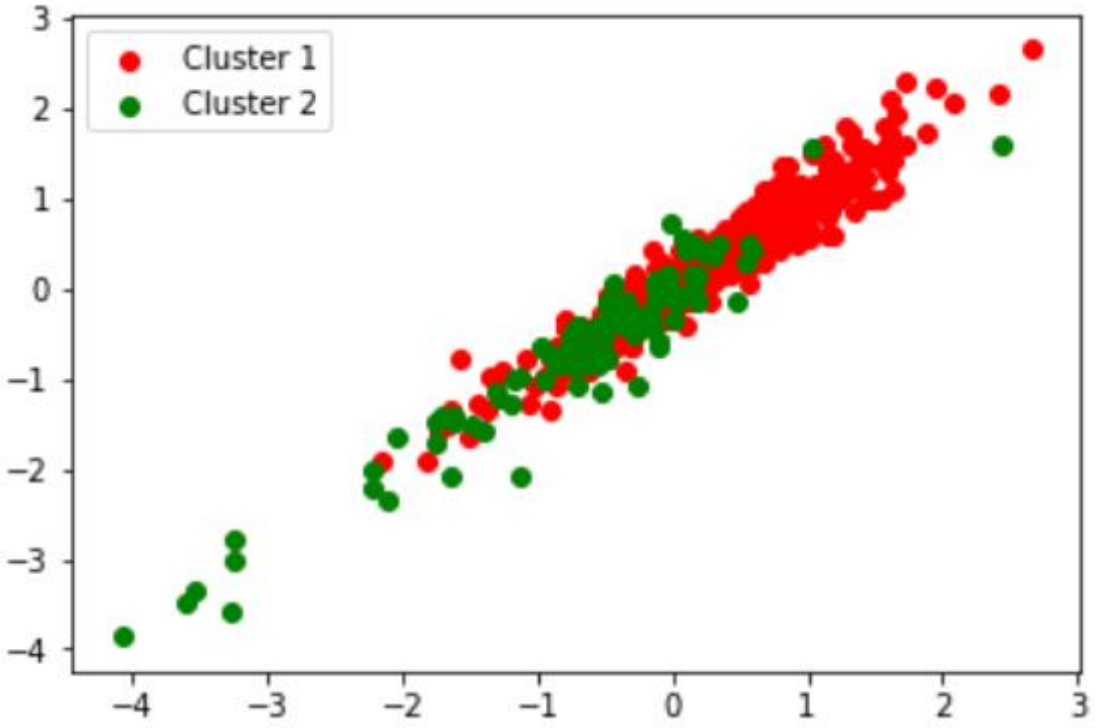


Figure 44 Scatter Plot with 2 Clusters (without PCA)

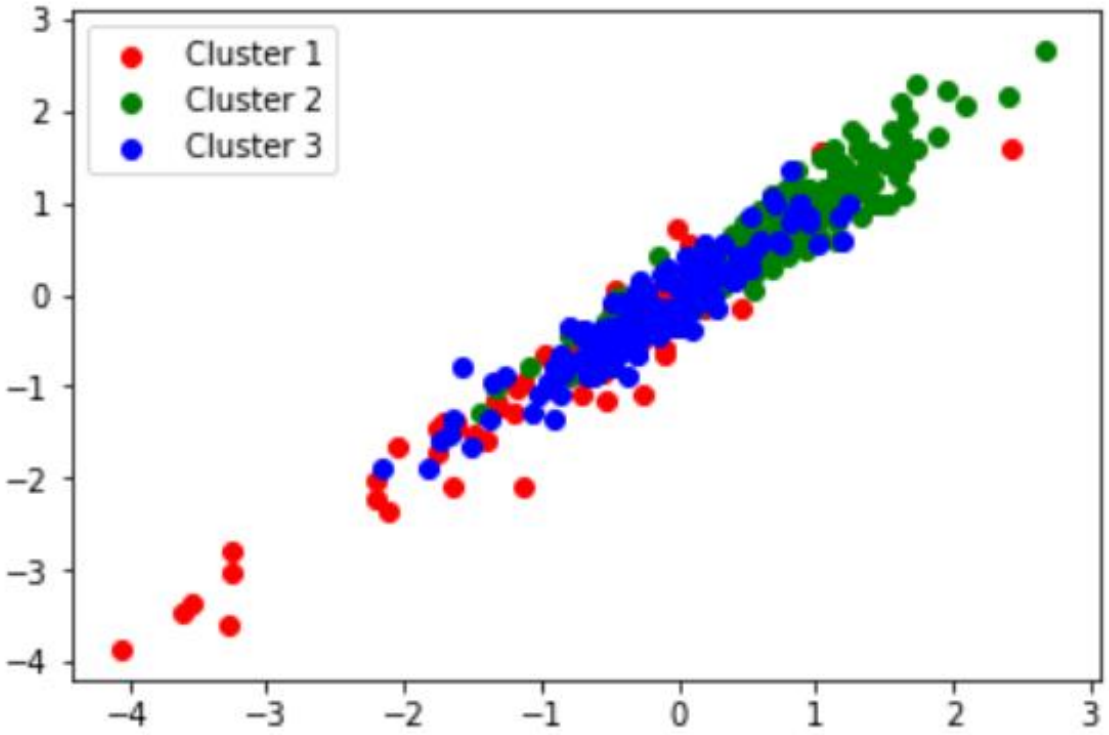


Figure 45 Scatter Plot with 3 Clusters (without PCA)

From the scatter plots above, all the clusters are poorly separated with distinct overlapping and to some extent, randomly distributed. The scatter plots provide little value in identifying patterns. Based on the significant differences in the outcome for PCA and without PCA. We can conclude that PCA data succeeded in reducing noise significantly to surface the underlying patterns in the dataset.

4.4.1. Duration of Training Hierarchical Clustering Model without PCA

```
import time
clustering = AgglomerativeClustering(linkage="ward", n_clusters=2)
start_time = time.time()
clustering.fit(x2)
end_time = time.time()
print((end_time - start_time))
```

```
0.03218197822570801
```

Duration of training 2 clusters Hierarchical Clustering without PCA: 0.0322 seconds

```
import time
clustering = AgglomerativeClustering(linkage="ward", n_clusters=3)
start_time = time.time()
clustering.fit(x2)
end_time = time.time()
print((end_time - start_time))
```

```
0.03570365905761719
```

Duration of training 3 clusters Hierarchical Clustering without PCA: 0.0357 seconds

4.5. DBSCAN

The DBSCAN machine learning model is tuned by adjusting the min_samples and the eps of the model. Base on the values set for both the parameters, the model will cluster sample points accordingly so that it matches the criterion; there should be sufficient points to form a cluster and the points in the cluster should be of a certain distance from each other. Points that are far away from the clusters will be classified as noise.

The model's behaviour is dictated by several parameters. In this analysis, we'll touch on three:
eps: Two points are considered neighbors if the distance between the two points is below the threshold epsilon.
min_samples: The minimum number of neighbors a given point should have in order to be classified as a core point. It's important to note that the point itself is included in the minimum number of samples.
metric: The metric to use when calculating distance between instances in a feature array (i.e. euclidean distance).

The algorithm works by computing the distance between every point and all other points. We then place the points into one of three categories.
Core point: A point with at least min_samples points whose distance with respect to the point is below the threshold defined by epsilon.
Border point: A point that isn't in close proximity to at least min_samples points but isclose enough to one or more core point. Border points are included in the cluster of the closest core point.
Noise point: Points that aren't close enough to core points to be considered border pointsNoise points are ignored. That is to say, they aren't part of any cluster.

In layman's terms, we find a suitable value for epsilon by calculating the distance to the nearest n points for each point, sorting and plotting the results. Then we look to see where the change is most pronounced (think of the angle between your arm and forearm) and select that as epsilon.

Algorithm 1 The pseudo code of the proposed technique DMDBSCAN to find suitable Epsi for each level of density in data set	
Purpose	*To find suitable values of Eps*
Input	*Data set of size n*
Output	*Eps for each varied density*
Procedure	*for i* *for j = 1 to n* *d(i,j) ← find distance (x_i, x_j)* *find minimum values of distances to nearest 3* *end for* *end for* *sort distances ascending and plot to find each value* *Eps corresponds to critical change in curves*

Figure 46 Pseudocode DMDBSCAN Algorithm (Elbatta 2012)

We can calculate the distance from each point to its closest neighbour using the NearestNeighbors. The point itself is included in n_neighbors. The kneighbors method returns two arrays, one which contains the distance to the closest n_neighbors points and the other which contains the index for each of those points.

```
In [12]: neigh = NearestNeighbors(n_neighbors=2)
         nbrs = neigh.fit(X2)
         distances, indices = nbrs.kneighbors(X2)
```

Figure 47 Finding distance for every 2 points

Next, we sort and plot results.

```
In [13]: distances = np.sort(distances, axis=0)
         distances = distances[:,1]
         plt.plot(distances)
Out[13]: [<matplotlib.lines.Line2D at 0x166aaadb7c8>]
```

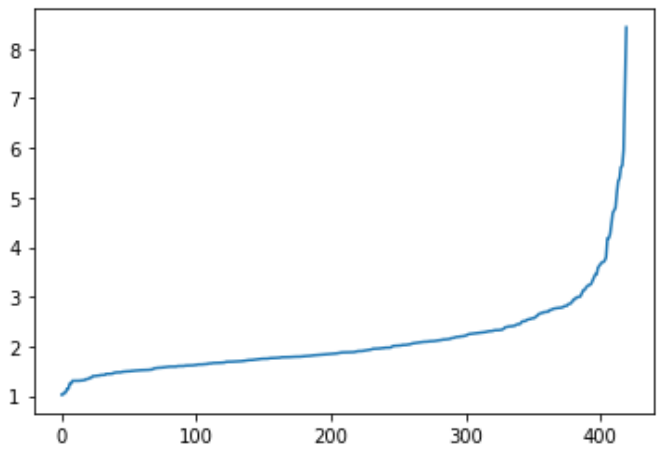


Figure 48 Line graph for optimal epsilon

The optimal value for epsilon will be found at the point of maximum curvature.
So, we firstly set 3 for eps and 3 for min_samples.

4.5.1. DBSCAN without PCA

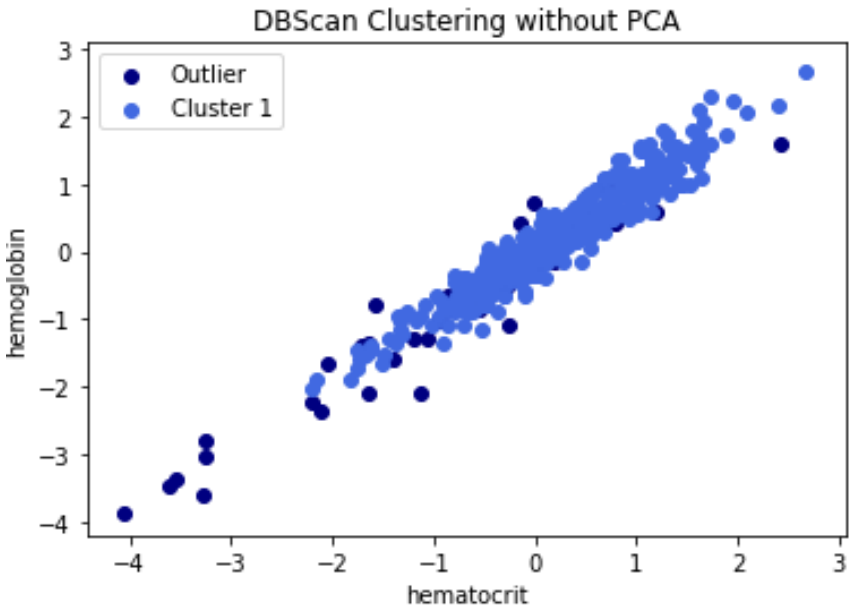


Figure 49 Scatterplot visualizations for DBSCAN (eps=3, min_samples=3) without PCA

Training Duration:0.015625 seconds

The above figure is trained on raw data without performing PCA. From the scatter diagram, we observe that the points are overlapping each other and the clusters cannot be well classified. The outcome is only 1 cluster and 35 outliers.

```
cluster_count = len(set(clusters)) - (1 if -1 in clusters else 0)
outiler_count = list(clusters).count(-1)

print("No: of Outliers:",outiler_count)
print("No: of Clusters:",cluster_count)
```

```
No: of Outliers: 35
No: of Clusters: 1
```

Figure 50 Outliers and clusters for DBSCAN (eps=3, min_samples=3) without PCA

To visualize more clusters, DBSCAN will be tested by narrowing down the min_sample to 2 according to our dataset.

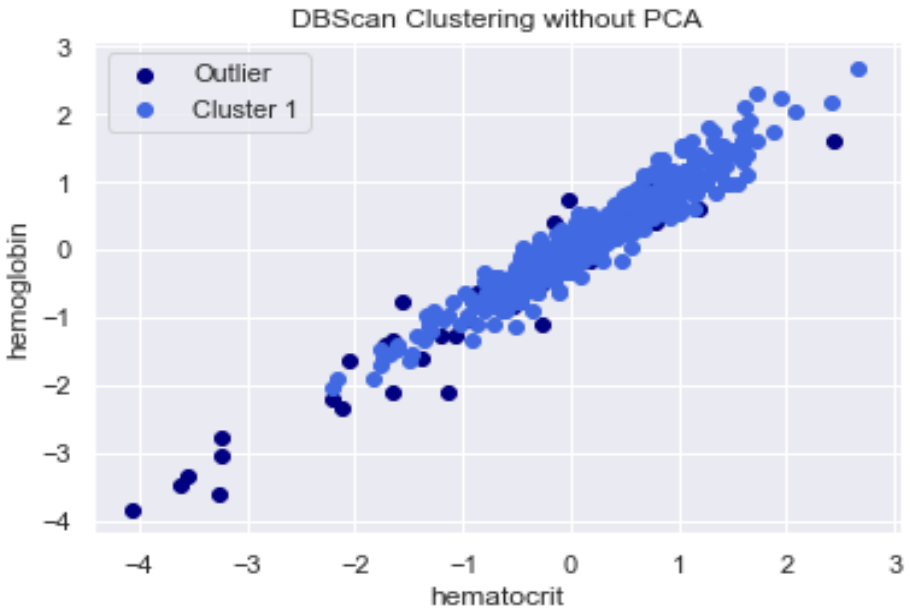


Figure 51 Scatterplot visualizations for DBSCAN (eps=3, min_samples=2) without PCA

Training Duration: 0.015625 seconds

The DBSCAN without PCA (Eps=3 , min_sample=2) found 1 clusters and 35 outliers

The outcome of cluster is still same with min_sample=3 when the model is trained on the rawdata.

4.5.2. DBSCAN with PCA

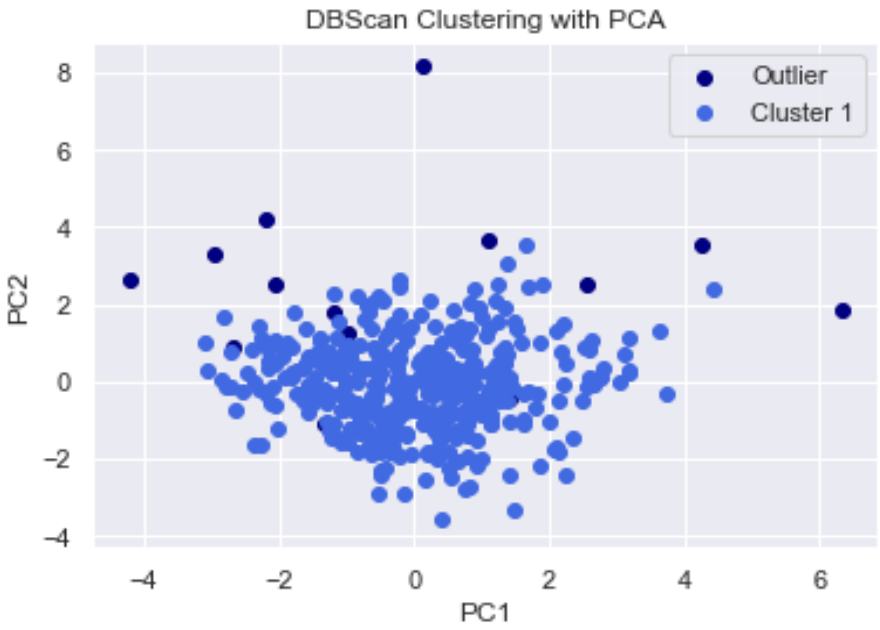


Figure 52 Scatterplot visualizations for DBSCAN (eps=3, min_samples=3) with PCA

Training Duration: 0.015625 seconds

The above figure is trained on the data which PCA is applied. From the scatter diagram, we observe that the points are overlapping each other and the clusters cannot be well classified.

The number of outliers are significantly decreasing to 14 after applying PCA on dataset. The outcome is 1 cluster and 14 outliers.

```
cluster_count = len(set(clusters)) - (1 if -1 in clusters else 0)
outiler_count = list(clusters).count(-1)

print("No: of Outliers:",outiler_count)
print("No: of Clusters:",cluster_count)
```

```
No: of Outliers: 14
No: of Clusters: 1
```

Figure 53 Outliers and clusters for DBSCAN (eps=3, min_samples=3) with PCA

To visualize more clusters, DBSCAN with PCA will be tested by narrowing down the min_sample to 2 according to our dataset.

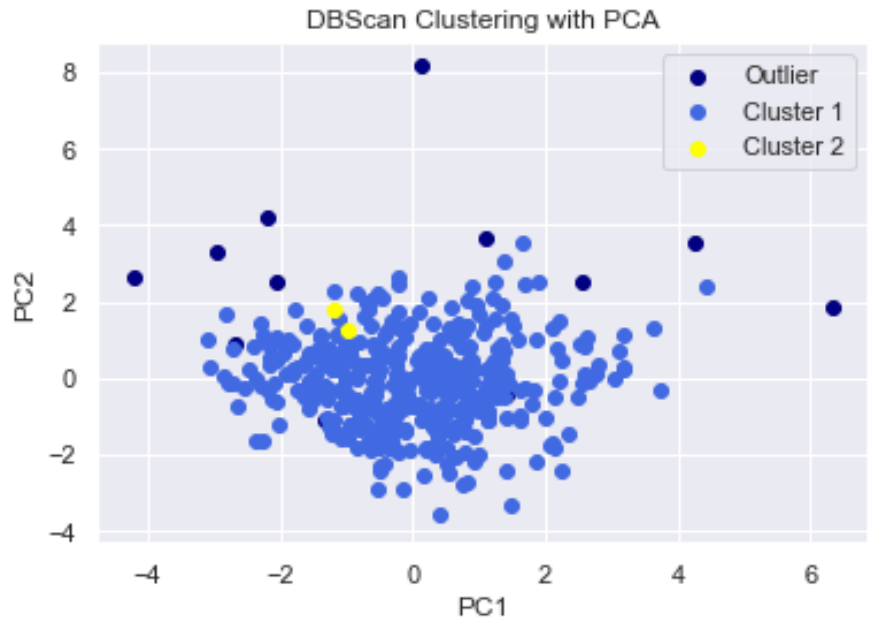


Figure 54 Scatterplot visualizations for DBSCAN (eps=3, min_samples=2) with PCA

```
Training Duration: 0.015625 seconds
```

```
cluster_count = len(set(clusters)) - (1 if -1 in clusters else 0)
outiler_count = list(clusters).count(-1)

print("No: of Outliers:",outiler_count)
print("No: of Clusters:",cluster_count)
```

```
No: of Outliers: 12
No: of Clusters: 2
```

Figure 55 Outliers and clusters for DBSCAN (eps=3, min_samples=2) with PCA

We observed that most of the points are grouped into the blue clusters which may not be an ideal predict model as it could mean that most of the prediction made in with this model will be classified as this cluster. The dark blue points are the outliers.

The outcome of the DBSCAN with PCA (Eps=3, min_sample=2) found 2 clusters and 12 outliers.

4.6. Conclusion

Based on the three unsupervised learning model, both K-Means and Hierarchical Clustering represented the data far better the DBSCAN model. Since we already know of the actual label in the in dataset, Hierarchical Clustering performs slightly better than K-Means Clustering as the scatter plot with 2 clusters(PCA) represented the actual negative to positive label ratio better than K-Means Clustering.

Interestingly, both models showed a third cluster in the data. From their scatter plots, the separation of the 3 clusters are clearly defined to discount the possibility. This third cluster could be a mutated strain of Covid-19 or a common disease across the patients.

In conclusion, the outcomes of our unsupervised learning models suggested the data could have 2 or 3 clusters. Further investigation is needed to learn and identify what the cluster represents.

4.7. Discussion

Since unsupervised learning models are mainly used to model underlying structure or distribution in the data to learn more about it, the impact of data engineering and feature engineering on unsupervised learning models are significant. We can clearly observe from the results above, the models with PCA data display more structured and distinct patterns compared to models without PCA data. The observation aligns with the main purpose of PCA technique which is to reduce noise and improve data visualization so that the underlying structures can be easily identified.

5. Limitations for all models

Missing Data (for supersived and unsupervised learning)
The Dataset contains a lot of missing data in potentially important features that have to be excluded in our models. Regardless, we are able to attain an accuracy level of at least 80% for our supervised models. However, with more complete data, larger sample size and dimensions can be used to train our models to significantly improve its prediction capabilities.

Big Assumptions
Despite the supervised learning models having decent accuracies, it will have little used in the real-world predictions. This is because by employing supervised learning models, we are theoretically assuming the distribution of the data resulting in a specific outcome to be stationary which is not true in the real-world and we are also assuming the label, Covid-19 result is dependent on the laboratory test results which may not be the case until proven.

6. Appendix

```
index  0
Number of positives: 112
Number of negatives: 1240
Positive to Negative percentage: 9.032258064516128
1352 rows x  18 columns

index  1
Number of positives: 62
Number of negatives: 758
Positive to Negative percentage: 8.179419525065963
820 rows x  3 columns

index  2
Number of positives: 59
Number of negatives: 361
Positive to Negative percentage: 16.343490304709142
420 rows x  17 columns

index  3
Number of positives: 55
Number of negatives: 290
Positive to Negative percentage: 18.96551724137931
345 rows x  5 columns

index  4
Number of positives: 14
Number of negatives: 318
Positive to Negative percentage: 4.40251572327044
332 rows x  2 columns

index  5
Number of positives: 40
Number of negatives: 185
Positive to Negative percentage: 21.62162162162162
225 rows x  3 columns

index  6
Number of positives: 33
Number of negatives: 175
Positive to Negative percentage: 18.857142857142858
208 rows x  2 columns

index  7
Number of positives: 29
Number of negatives: 100
Positive to Negative percentage: 28.999999999999996
129 rows x  6 columns
```

Appendix 1 Refined data sample 1/3

```
index  8
Number of positives: 20
Number of negatives: 116
Positive to Negative percentage: 17.24137931034483
136 rows x  8 columns

index  9
Number of positives: 23
Number of negatives: 110
Positive to Negative percentage: 20.909090909090907
133 rows x  2 columns

index  10
Number of positives: 23
Number of negatives: 81
Positive to Negative percentage: 28.39506172839506
104 rows x  2 columns

index  11
Number of positives: 26
Number of negatives: 75
Positive to Negative percentage: 34.66666666666667
101 rows x  2 columns

index  12
Number of positives: 9
Number of negatives: 88
Positive to Negative percentage: 10.227272727272728
97 rows x  7 columns

index  13
Number of positives: 15
Number of negatives: 76
Positive to Negative percentage: 19.736842105263158
91 rows x  2 columns

index  14
Number of positives: 8
Number of negatives: 43
Positive to Negative percentage: 18.6046511627907
51 rows x  16 columns

index  15
Number of positives: 9
Number of negatives: 48
Positive to Negative percentage: 18.75
57 rows x  2 columns
```

Appendix 2 Refined data sample 2/3

```
index  16
Number of positives: 5
Number of negatives: 28
Positive to Negative percentage: 17.857142857142858
33 rows x  3 columns

index  17
Number of positives: 13
Number of negatives: 14
Positive to Negative percentage: 92.85714285714286
27 rows x  10 columns

index  18
Number of positives: 2
Number of negatives: 21
Positive to Negative percentage: 9.523809523809524
23 rows x  2 columns

index  19
Number of positives: 0
Number of negatives: 2
Positive to Negative percentage: 0.0
2 rows x  3 columns

index  20
Number of positives: 0
Number of negatives: 13
Positive to Negative percentage: 0.0
13 rows x  2 columns

index  21
Number of positives: 3
Number of negatives: 5
Positive to Negative percentage: 60.0
8 rows x  2 columns

index  22
Number of positives: 0
Number of negatives: 3
Positive to Negative percentage: 0.0
3 rows x  2 columns
```

Appendix 3 Refined data sample 3/3

```
##Using no feature engineering

After a 100 runs

#For Logistic Regresstion
The average accuracy score is  0.8516981132075473
The standard deviation for accuracy score is  0.007788591487000109
The maximum accuracy is  0.8679245283018868
The minimum accuracy is  0.839622641509434
The average time taken is  0.0064955496788024905
The confusion matrix for the last iteration is
 [[11  4]
 [11 80]]

#For K Nearest Neighbours
The most common n is n =  1
The average accuracy score is  0.8679245283018866
The standard deviation for accuracy score is  2.220446049250313e-16
The maximum accuracy is  0.8679245283018868
The minimum accuracy is  0.8679245283018868
The average time taken is  0.0020020008087158203 (excluding time to find the best n)
The confusion matrix for the last iteration is
 [[ 7  8]
 [ 6 85]]

#For Decision Tree
The average accuracy score is  0.8296226415094341
The standard deviation for accuracy score is  0.02171123475275282
The maximum accuracy is  0.8962264150943396
The minimum accuracy is  0.7924528301886793
The average time taken is  0.001950995922088623
The confusion matrix for the last iteration is
 [[ 9  6]
 [14 77]]

#For Neural Network
The average accuracy score is  0.873396229147911
The standard deviation for accuracy score is  0.01486739901303666
The maximum accuracy is  0.9056603908538818
The minimum accuracy is  0.8396226167678833
The average time taken is  6.27070426940918
The confusion matrix for the last iteration is
 [[ 9  6]
 [ 8 83]]
```

Appendix 4 100 Run results with no feature engineering

```
After a 100 runs

#For Logistic Regresstion
The average accuracy score is  0.852547169811321
The standard deviation for accuracy score is  0.009306210484283366
The maximum accuracy is  0.8773584905660378
The minimum accuracy is  0.8301886792452831
The average time taken is  0.004324028491973877
The confusion matrix for the last iteration is
 [[10  5]
 [12 79]]

#For K Nearest Neighbours
The most common n is n =  1
The average accuracy score is  0.8490566037735852
The standard deviation for accuracy score is  2.220446049250313e-16
The maximum accuracy is  0.8490566037735849
The minimum accuracy is  0.8490566037735849
The average time taken is  0.0010008811950683594 (excluding time to find the best n)
The confusion matrix for the last iteration is
 [[ 6  9]
 [ 7 84]]

#For Decision Tree
The average accuracy score is  0.8416981132075473
The standard deviation for accuracy score is  0.02945995465787899
The maximum accuracy is  0.9245283018867925
The minimum accuracy is  0.7924528301886793
The average time taken is  0.001501767635345459
The confusion matrix for the last iteration is
 [[ 9  6]
 [13 78]]

#For Neural Network
The average accuracy score is  0.8616037750244141
The standard deviation for accuracy score is  0.010424848777302265
The maximum accuracy is  0.8867924809455872
The minimum accuracy is  0.8396226167678833
The average time taken is  6.377801418304443
The confusion matrix for the last iteration is
 [[10  5]
 [11 80]]
```

Appendix 5 100 Run result with only correlation matrix

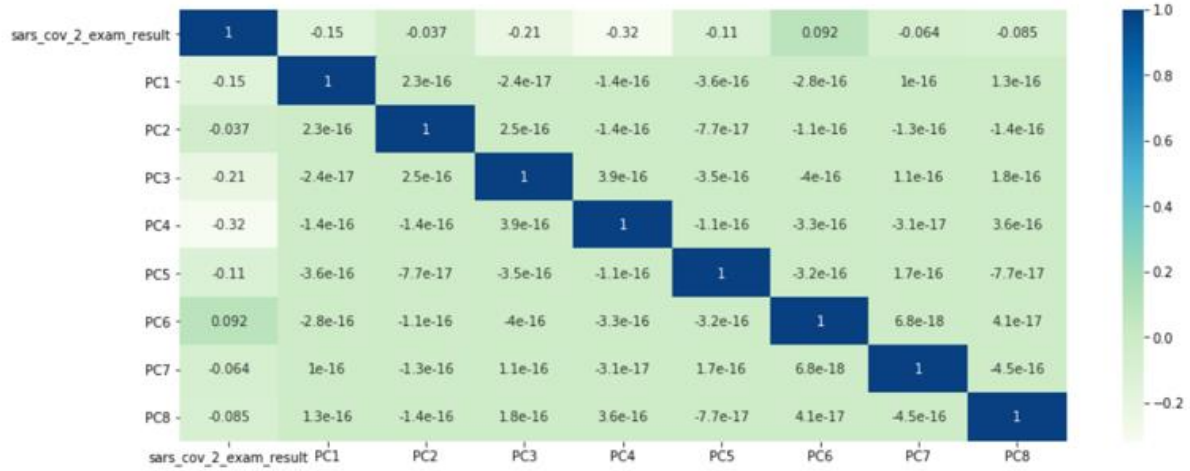
```
After a 100 runs

#For Logistic Regresstion
The average accuracy score is  0.8266981132075472
The standard deviation for accuracy score is  0.012980185902331897
The maximum accuracy is  0.8584905660377359
The minimum accuracy is  0.7830188679245284
The average time taken is  0.003482942581176758
The confusion matrix for the last iteration is
 [[10  5]
 [14 77]]

#For K Nearest Neighbours
The most common n is n =  1
The average accuracy score is  0.8490566037735852
The standard deviation for accuracy score is  2.220446049250313e-16
The maximum accuracy is  0.8490566037735849
The minimum accuracy is  0.8490566037735849
The average time taken is  0.0010008811950683594 (excluding time to find the best n)
The confusion matrix for the last iteration is
 [[ 6  9]
 [ 7 84]]

#For Decision Tree
The average accuracy score is  0.7630188679245282
The standard deviation for accuracy score is  0.0487340630356674
The maximum accuracy is  0.8773584905660378
The minimum accuracy is  0.660377358490566
The average time taken is  0.001621265411376953
The confusion matrix for the last iteration is
 [[ 9  6]
 [17 74]]

#For Neural Network
The average accuracy score is  0.809811326265335
The standard deviation for accuracy score is  0.014475521315720438
The maximum accuracy is  0.849056601524353
The minimum accuracy is  0.7830188870429993
The average time taken is  5.728201866149902
The confusion matrix for the last iteration is
 [[10  5]
 [15 76]]
```

Appendix 6 100 Run results with only PCA

```
After a 100 runs

#For Logistic Regresstion
The average accuracy score is  0.831509433962264
The standard deviation for accuracy score is  0.012516987389148665
The maximum accuracy is  0.8584905660377359
The minimum accuracy is  0.7924528301886793
The average time taken is  0.0034206247329711913
The confusion matrix for the last iteration is
 [[11  4]
 [14 77]]

#For K Nearest Neighbours
The most common n is n =  1
The average accuracy score is  0.8490566037735852
The standard deviation for accuracy score is  2.220446049250313e-16
The maximum accuracy is  0.8490566037735849
The minimum accuracy is  0.8490566037735849
The average time taken is  0.0010008811950683594 (excluding time to find the best n)
The confusion matrix for the last iteration is
 [[ 5 10]
 [ 6 85]]

#For Decision Tree
The average accuracy score is  0.819811320754717
The standard deviation for accuracy score is  0.016954906373218323
The maximum accuracy is  0.8679245283018868
The minimum accuracy is  0.7547169811320755
The average time taken is  0.0015216898918151856
The confusion matrix for the last iteration is
 [[10  5]
 [15 76]]

#For Neural Network
The average accuracy score is  0.8107547253370285
The standard deviation for accuracy score is  0.014905661306811936
The maximum accuracy is  0.849056601524353
The minimum accuracy is  0.7735849022865295
The average time taken is  6.159604549407959
The confusion matrix for the last iteration is
 [[ 9  6]
 [16 75]]
```

Appendix 7 100 Run results with correlation matrix then PCA

```
##Using both correlation matrix and PCA (with PCA first)

#With PCA

The most optimal n is  8

With variance ratio sum of  0.8771380365312372 (minimum 0.85)

The variance ratios are
 [0.21270762 0.18111298 0.12818513 0.1042022  0.07652559 0.0687402
 0.06036176 0.04530255]

#With Correlation Matrix
```




```
The columns  [] are removed

                After a 100 runs

                #For Logistic Regresstion
                The average accuracy score is  0.8287735849056603
                The standard deviation for accuracy score is  0.014205868251297729
                The maximum accuracy is  0.8584905660377359
                The minimum accuracy is  0.7830188679245284
                The average time taken is  0.003522951602935791
                The confusion matrix for the last iteration is
                 [[11  4]
                 [14 77]]

                #For K Nearest Neighbours
                The most common n is n =  1
                The average accuracy score is  0.8491509433962267
                The standard deviation for accuracy score is  0.0009386673934968111
                The maximum accuracy is  0.8584905660377359
                The minimum accuracy is  0.8490566037735849
                The average time taken is  0.001001596450805664 (excluding time to find the best n)
                The confusion matrix for the last iteration is
                 [[ 6  9]
                 [ 7 84]]

                #For Decision Tree
                The average accuracy score is  0.7634905660377358
                The standard deviation for accuracy score is  0.03971810153934404
                The maximum accuracy is  0.8584905660377359
                The minimum accuracy is  0.6886792452830188
                The average time taken is  0.0015310359001159668
                The confusion matrix for the last iteration is
                 [[10  5]
                 [28 63]]

                #For Neural Network
                The average accuracy score is  0.8082075530290603
                The standard deviation for accuracy score is  0.013609452724665093
                The maximum accuracy is  0.849056601524353
                The minimum accuracy is  0.7830188870429993
                The average time taken is  6.031477451324463
                The confusion matrix for the last iteration is
                 [[10  5]
                 [17 74]]
```

Appendix 8 100 Run result with PCA then correlation matrix