Department of Computer Science

MSc Data Science and Analytics

CS5811_Distributed Data Analysis
2020/21
Dr. Stasha Lauria

Data Analysis Report

"Does temperature affect the number of Covid infections in each country? Using Machine Learning techniques to predict infection status."

Student ID: 1043795

Acknowledgement

As someone who is immensely curious about the field of Data Science and always keen to participate in data analysis projects, the CS5811 Distributed Data Analysis module has been most invaluable to successfully understanding how to generate value and insight from heterogeneous datasets using various Machine Learning analytic algorithms, as well as high performance computational infrastructures techniques to manage and manipulate Big Data effectively.(I.e. MapReduce)
I wish to express my sincere appreciation to the module leader, Dr. Stasha Lauria. This module has provided me with comprehensive guidance in how to successfully execute a data analysis project as well as all the crucial 'what not to dos!'. Through this process, I have gained life-long skill that I will treasure for many years to come throughout my career. I would also like to give special thanks to my tutor and lecturers Dr. Martin Shepperd, Dr. Alessandro Pandini, & Dr. Alaa Marshan for their guidance. This project would not have been possible without the support of these very important individuals.

Table of Contents

1. Data description and research question

1.1 Introduction

The aim of this report is to satisfy two main learning outcomes for the module CS5811 Distributed data analysis. This is achieved by implementing analytical methods and algorithms introduced in study blocks CS5706 Machine Learning and CS5710 High Performance Computational Infrastructure.
For this purpose, Covid19 data was collated as the team were most interested in better understanding this data. With the current coronavirus situation in the world, there is an even greater need to understand the Covid-19 virus and how different variables may affect its infection rate. As a result, weather metrics were discussed. Recent studies from China also concluded both positive (Xie, J. & Zhu, Y 2020) and negative (Iqbal et al. 2020) association with environmental temperature. Similarly, in Spain researchers reported no effect of temperature and Covid transmissions (Briz-Redon A. & Serrano-Aroca, A., 2020).
The group is keen to apply new skills gained through the study blocks and have proposed the research question **'Does temperature affect the number of Covid infections in each country? Using Machine Learning techniques to predict infection status.'** This report catalogue the analytical steps taken to answer the research question above by generating valuable insight from the processing of heterogenous datasets collected, as well as any limitations encountered when working with such datasets.

1.2 Data description

The Covid19 dataset was downloaded from the John Hopkin University (JHU) repository, it contains 68,533 rows of daily Covid deaths reported by each country with 58 other demographic information.
The daily temperature data was downloaded from the National Oceanic and Atmospheric Administration (NOAA) repository, and it contains the daily precipitation, minimum, maximum, and average temperature observations for each weather station in the world. The raw data contains 37,865,898 rows in long format. Additional files required to link the temperature data to Covid data are the CountryID file and the StationID file.

Covid Data

iso_code <chr>	continent <chr>	location <chr>	date <chr>	total_cases <int>	new_cases <int>	new_cases_smoothed <dbl>	total_deaths <int>	new_deaths <int>
ZWE	Africa	Zimbabwe	07/02/2021	34552	65	166.286	1326	10
ZWE	Africa	Zimbabwe	08/02/2021	34658	106	158.571	1339	13
ZWE	Africa	Zimbabwe	09/02/2021	34781	123	138.143	1353	14
ZWE	Africa	Zimbabwe	10/02/2021	34864	83	128.571	1364	11
ZWE	Africa	Zimbabwe	11/02/2021	34864	0	99.000	1364	0
ZWE	Africa	Zimbabwe	12/02/2021	35045	181	102.000	1393	29

Country.Codes <chr>	ID <chr>	LATITUDE <dbl>	LONGITUDE <dbl>
AC	ACW00011604	17.1167	-61.7833
AC	ACW00011647	17.1333	-61.7833
AE	AE000041196	25.3330	55.5170
AE	AEM00041194	25.2550	55.3640
AE	AEM00041217	24.4330	54.6510
AE	AEM00041218	24.2620	55.6090

StationID

Country.Codes <chr>	Country <chr>
AC	Antigua and Barbuda
AE	United Arab Emirates
AF	Afghanistan
AG	Algeria
AJ	Azerbaijan
AL	Albania

CountryID

V1 <chr>	V2 <int>	V3 <chr>	V4 <int>
AE000041196	20200101	TMIN	168
AE000041196	20200101	PRCP	0
AE000041196	20200101	TAVG	211
AEM00041194	20200101	PRCP	0
AEM00041194	20200101	TAVG	217
AEM00041218	20200101	TMIN	148

Temperature Data

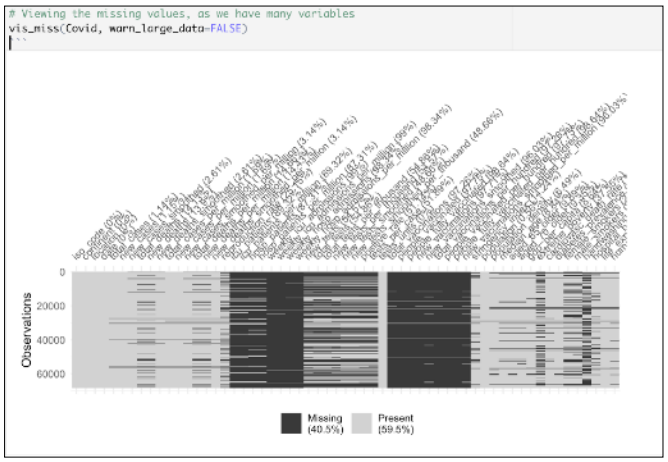
2. Data preparation and cleaning

Most of the cleaning and preprocessing was performed using R notebook.

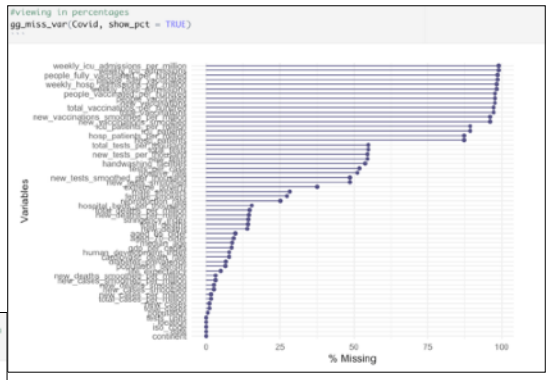
2.1 Data visualisation

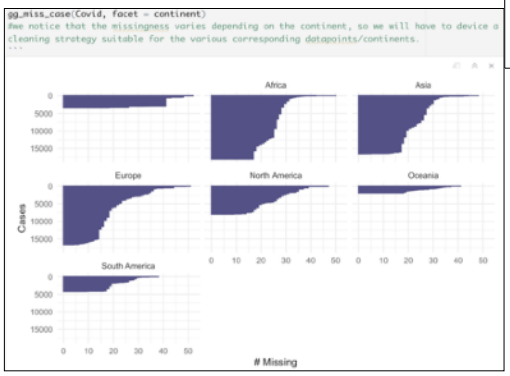
2.1.1 MISSINGNESS

As we have many variables for the Covid data especially, we will make use of graph to visualise this. We can see that most of the missing data are from Covid patients admissions and vaccinations attributes. Overall there are 40.5% missing data. The temperature data are comprehensive and few to none missing observations.



Looking at the missingness by continent. Another view of the missing-ness, give better idea of how to deal with it.





2.2 Data pre-processing

2.2.1 DATA FORMATTING

Converting the data type of $ iso_code, $ continent and $ location into factor variables.

```
#Convert to factor
Covid$iso_code <- as.factor(Covid$iso_code)
Covid$continent <- as.factor(Covid$continent)
```

```
#converting date to date format to join with countryID and station
date <- MasterWeather$date
MasterWeather <- transform(MasterWeather, date = as.Date(as.character(date), "%Y%m%d"))
MasterWeather <- as.data.frame(MasterWeather)

head(MasterWeather)
```

	ID <chr>	date <date>	AvgTemp <dbl>
1	AE000041196	2020-01-01	9.94
2	AEM00041194	2020-01-01	10.28
3	AEM00041218	2020-01-01	9.28
4	AEM00041217	2020-01-01	9.61
5	AFM00040938	2020-01-01	1.22
6	AFM00040990	2020-01-01	4.72

Converting date to date format to join with countryID and station

2.2.2 DATA CLEANING

```
#Weather Data clean & tidying
```{r}
#we will take the average temperature only from column 3, TAVG
Weather2020 <- filter(Weather2020, V3 == "TAVG")
Weather2021 <- filter(Weather2021, V3 == "TAVG")
```

```
#Remove missing values
table(Weather2020$V4==c(9999, 999, 0))
table(Weather2021$V4==c(9999, 999, 0))
dim(Weather2020)
dim(Weather2021)
```

Filtering by TAVG:

Data documentation from NOAA specifies that 9's in a field (e.g. 9999) indicate missing data or data that has not been received. We will check for these in our numerical values and remove if necessary. If the number of 9999 present in data is significant, we need to remove before performing calculations.

```
#Renaming Columns
MasterWeather <- MasterWeather %>%
  dplyr::rename(ID = V1, date = V2)
#Checking the changes took place:
head(MasterWeather)
str(MasterWeather)
#as.data.frame
MasterWeather <- as.data.frame(MasterWeather)
```

Renaming columns for easier representation, then check that the changes have taken place.

```
weathercombined <- merge(CombinedID, MasterWeather, all.x=TRUE)
head(weathercombined, n=50)
```

	ID <chr>	Country.Codes <chr>	Country <chr>	date <date>	AvgTemp <dbl>
3	AE000041196	AE	United Arab Emirates	2020-01-01	9.94
4	AE000041196	AE	United Arab Emirates	2021-01-18	9.50
5	AE000041196	AE	United Arab Emirates	2020-04-27	16.83
6	AE000041196	AE	United Arab Emirates	2020-02-14	7.78
7	AE000041196	AE	United Arab Emirates	2020-09-07	17.89
8	AE000041196	AE	United Arab Emirates	2020-10-16	13.89
9	AE000041196	AE	United Arab Emirates	2020-09-11	17.56
10	AE000041196	AE	United Arab Emirates	2021-01-04	9.00
11	AE000041196	AE	United Arab Emirates	2020-05-03	16.11
12	AE000041196	AE	United Arab Emirates	2020-03-07	10.22

Joining the Covid and AvgTemp dataset by date and location = country, making use of the full_join function.
Merging weather dataframe with combinedID, the default is FALSE, so that only rows with data from both x and y are included in the output. Setting argument all.x=TRUE, then extra rows will be added to the output, one for each row in x(CombinedID) that has no matching row in y(MasterWeather). Note that these rows will have NAs in those columns that are usually filled with values from y.

```
```{r}
#Combining datasets for weather from 2 tables
MasterWeather <- rbind(Weather2020, Weather2021)
#Checking that combined as it should:
nrow(Weather2020) + nrow(Weather2021)
str(MasterWeather)
```

```
[1] 1966182
'data.frame':   1966182 obs. of  3 variables:
 $ ID     : chr  "AE000041196" "AEM00041194" "AEM00041218"
"AEM00041217" ...
 $ date   : int  20200101 20200101 20200101 20200101 20200101
20200101 20200101 20200101 20200101 20200101 ...
 $ AvgTemp: num  9.94 10.28 9.28 9.61 1.22 ...
```

The temperature data was downloaded by years, so have separate files for 2020 and 2021 and so have to combine this into MasterWeather data file.

```
```{r}
#Merge Country data frame and Station data frame
CombinedID <- merge(CountryID, Stations, all.x=TRUE)
head(CombinedID)
str(CombinedID)
summary(CombinedID)
```

	Country.Codes <chr>	Country <chr>	ID <chr>
1	AC	Antigua and Barbuda	ACW00011604
2	AC	Antigua and Barbuda	ACW00011647
3	AE	United Arab Emirates	AE000041196
4	AE	United Arab Emirates	AEM00041194
5	AE	United Arab Emirates	AEM00041217
6	AE	United Arab Emirates	AEM00041218

Merging the CountryID file and the StationsID file into one dataframe, retaining all observations for each weather station.
```

The average temperature column was converted from the measure of fahrenheit to celsius using the formula (x - 32)*5/9. Restructuring the temperature data was necessary for easier analysis.

```
head(Weather2021)
```

V1 <chr>	V2 <int>	V3 <chr>	V4 <int>
AE000041196	20210101	TMAX	278
AE000041196	20210101	PRCP	0
AE000041196	20210101	TAVG	214
AEM00041194	20210101	TMAX	266
AEM00041194	20210101	TMIN	178
AEM00041194	20210101	PRCP	0

```
head(MasterWeather)
```

	ID <chr>	date <date>	AvgTemp <dbl>
1	AE000041196	2020-01-01	9.94
2	AEM00041194	2020-01-01	10.28
3	AEM00041218	2020-01-01	9.28
4	AEM00041217	2020-01-01	9.61
5	AFM00040938	2020-01-01	1.22
6	AFM00040990	2020-01-01	4.72

2.2.3 DATA SAMPLING

For the purpose of this project, where it involves a large volume of data it is necessary to come up with a random sample of a smaller size for processing. A random sample is one which each instance in the original dataset has an equal chance of being included. (Witten et al., 2017). Sampling with replacement is also considered, given a batch of N instances, a sample of any desired size is created by generating uniform random integers between 1 and N and retrieve the corresponding instances until the appropriate number has been collected. In this method, the same instance might be selected more than once. For sampling without replacement, simply note the instances selected and whether it has already been chosen, if so, discard the second copy. (Witten et al., 2017)

There are few other options to choose from including Systematic Sampling, Stratified sampling and Cluster Sampling. It is decide that the Stratified sampling is the most suitable for this dataset, where samples are taken as a random proportion of the locations within the dataset. Using this method, all countries are included within each of the samples. This type of sampling is applied when representation from all the subgroups of the population is desired. (Gangwal, 2019). In this case, all countries in the dataset.

```
## Stratified random split into the Training set and Test set
```{r}
## Using the createDataPartition function from caret package, does a stratified random split
of the data.
## Using the createDataPartition function from caret package, does a stratified random split
of the data. We need the same amount of data from all countries/locations.
library(caret)
train.index <- createDataPartition(Covid.SVM2$location, p = .7, list = FALSE)
training_set <- Covid.SVM2[ train.index,]
test_set  <- Covid.SVM2[-train.index,]
```

2.3 Data transformation

2.3.1 SCALING

Scaling and normalisation is carefully considered throughout the analysis, for the Support Vector Machine prediction preparation, both the training dataset and test dataset was normalised using the mean and standard deviation from the training dataset. As demonstrated below:

```
## Feature Scaling
```{r}
# Scaling the dataset without row 1 and row 58 the response variable
training_set[-c(1,58)] = scale(training_set[-c(1,58)])
test_set[-c(1,58)] = scale(test_set[-c(1,58)])
str(training_set)
str(test_set)
# remove location column
training_set <- training_set[-c(1)]
test_set <- test_set[-c(1)]

```
```
```

```
# Normalise training and test dataset
# transform the test set using training set parameters (mean, sd)

# create new dataframe without the target variables

X = training_set.iloc[:, :-56]
Y = test_set.iloc[:, :-56]
print(X)
scaler = preprocessing.StandardScaler().fit(X)
scaled_X = scaler.transform(X)
scaled_Y = scaler.transform(Y)
```

2.3.2 AGGREGATION

```
```{r}
AvgTemp <- aggregate(AvgTemp~Country+date, weathercombined, mean)
summary(AvgTemp)
str(AvgTemp)
#so we have 73,210/72533 observations of temperature
```

   Country   date   AvgTemp
 Length:72533   Min.   :2020-01-01   Min.   :-1.720
 Class :character   1st Qu.:2020-04-09   1st Qu.: 6.080
 Mode :character   Median :2020-07-18   Median :11.480
                    Mean   :2020-07-18   Mean   : 9.618
                    3rd Qu.:2020-10-27   3rd Qu.:13.402
                    Max.   :2021-02-11   Max.   :22.280
```

Aggregating the weathercombined data frame by country and date, taking the mean for aggregation. The AvgTemp variable is the average temperature readings of all weather stations in each country for a particular corresponding date.

2.3.3 TRANSFORM ATTRIBUTE TYPE

For preparation of Support Vector classifier, the discrete variable “new_cases_per_million” was transformed to ordinal to include three categories: Green, Orange & Red. In accordance to the European Centre for Disease Prevention and Control’s traffic warning system, where countries are classified into the above three categories based on their infection rates.

```
```{r}
# Transform attribute type to three levels:
Covid.SVM <- Covid.Temp %>% mutate(Traffic_warning =
                     case_when(new_cases_per_million <= 250 ~ "Green",
                               new_cases_per_million <= 500 ~ "Orange",
                               new_cases_per_million > 500 ~ "Red")
)
```

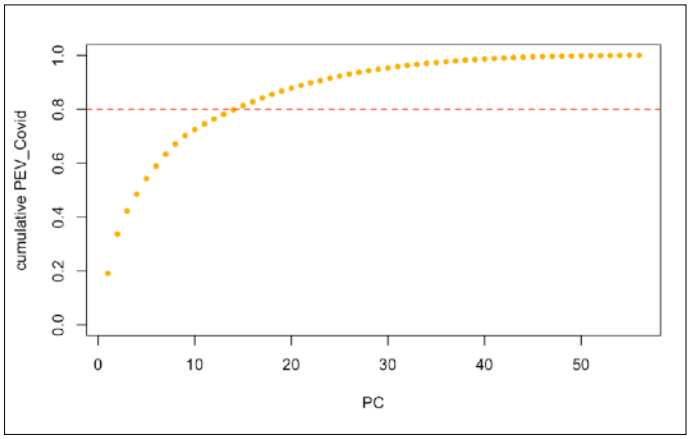
3. Exploratory data analysis

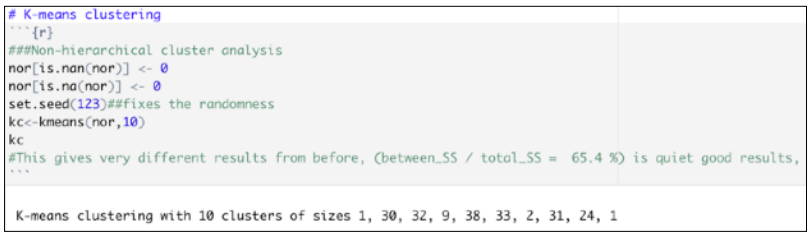
3.1 Data selection

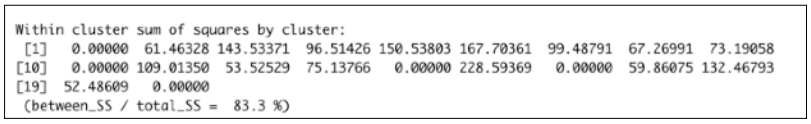
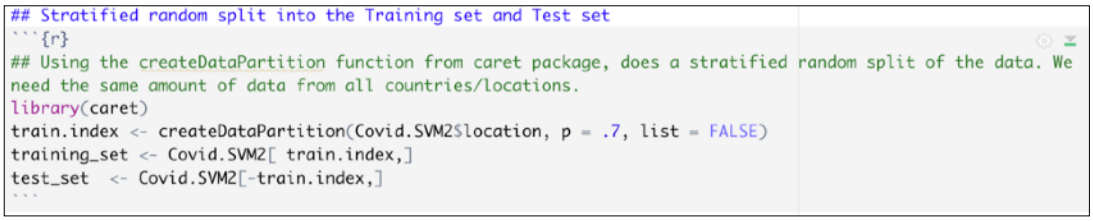
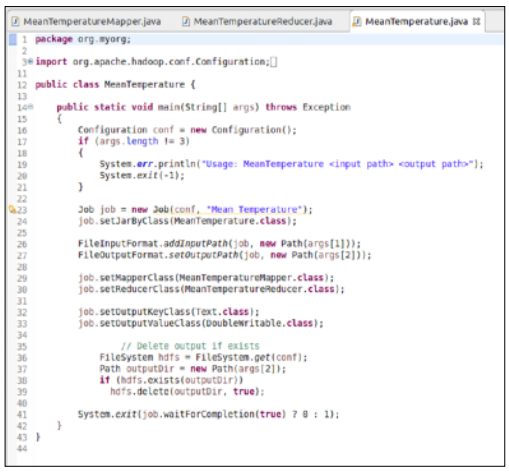
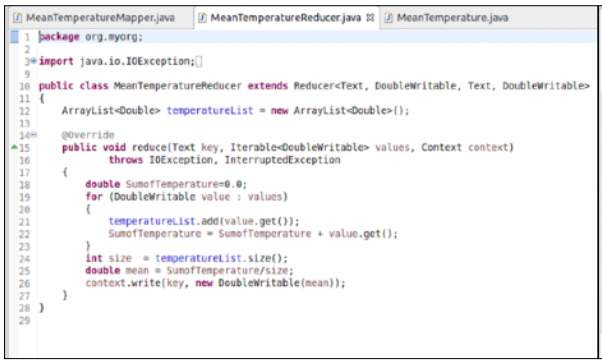
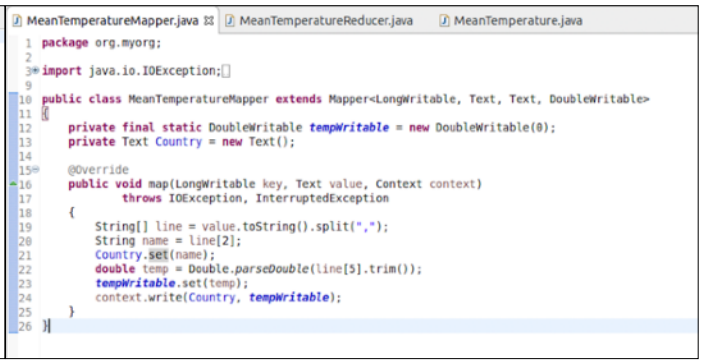
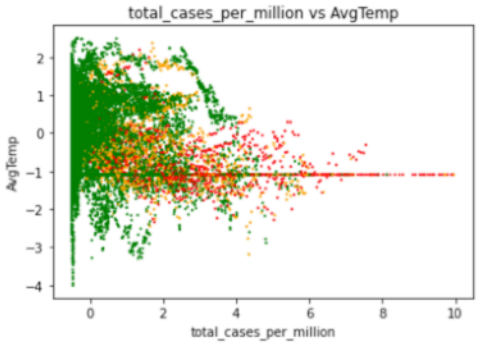
For features extraction and reduction, we will make use of one of the most popular technique to examine the high dimensional combined dataset.

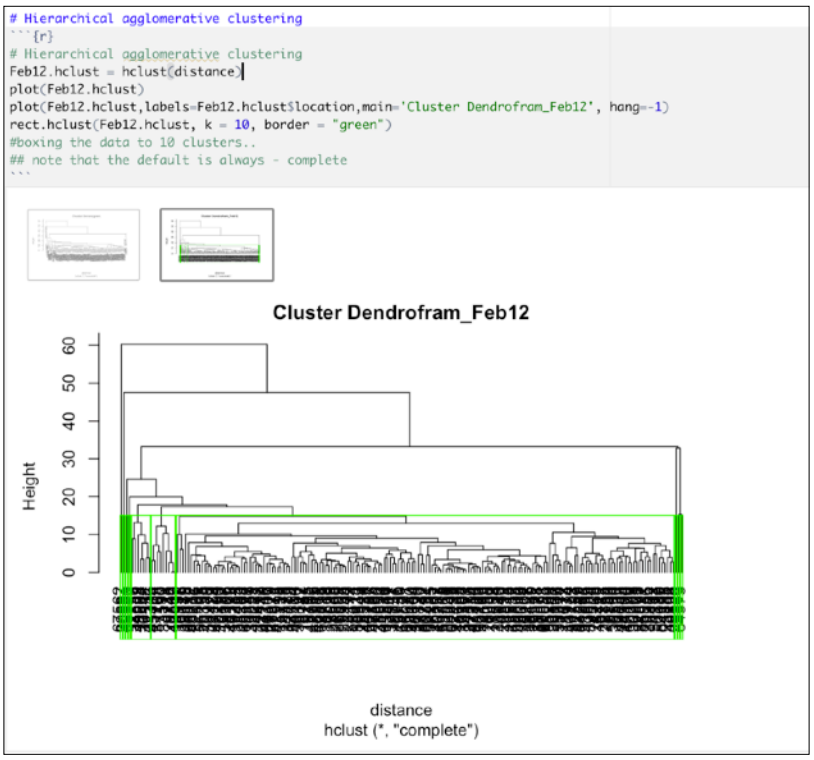
3.1.1 PRINCIPAL COMPONENT ANALYSIS

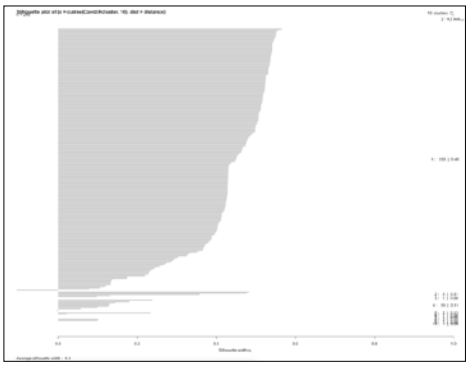
PCA on all countries:

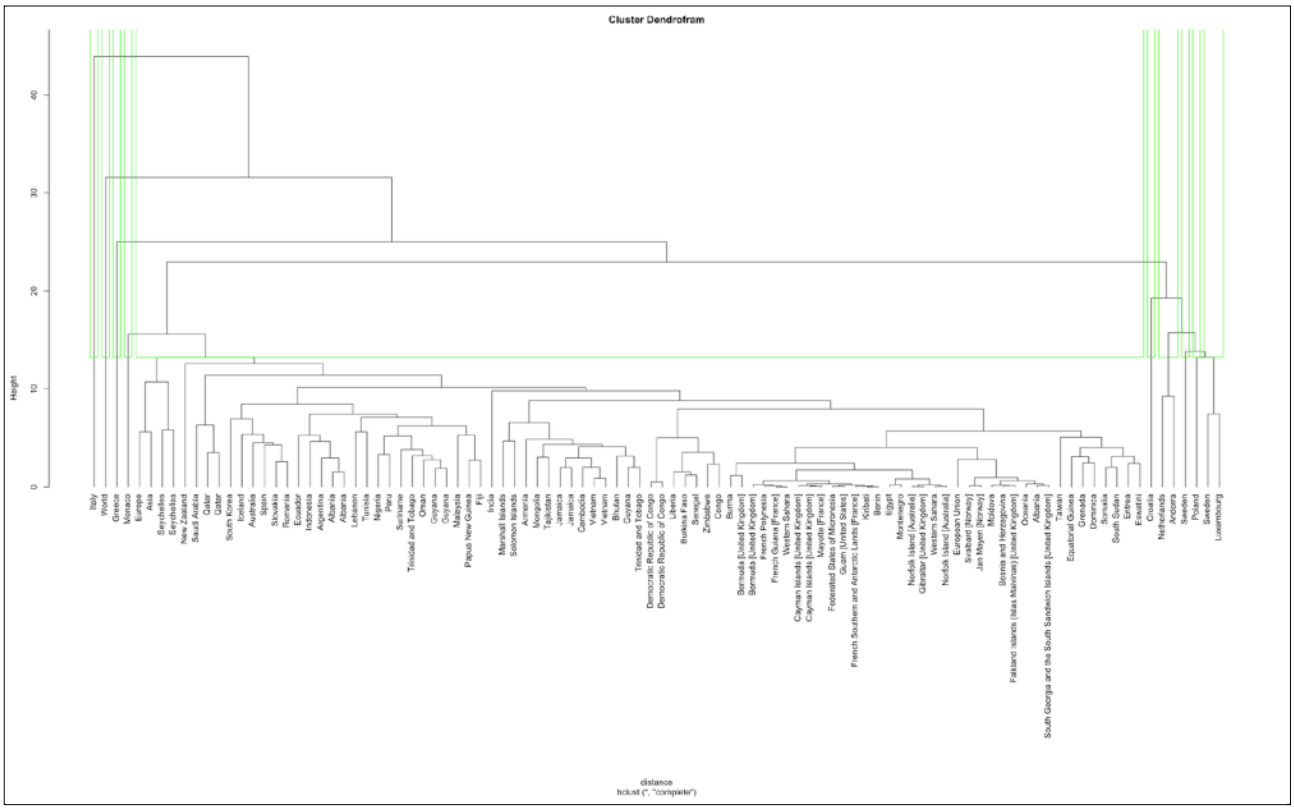


The aim is for features extraction. See Appendix A for Importance of components. We can see that 80% of the variations are explained in PC1:PC14 in the scree plot, where we can visual the ‘elbow’ around the 80% margin. From summary(pca) we can note that up to PC14, variability is covered by 80%. To cover 95% variability we will need PC1:PC29. See Appendix A for PC loading for the first 14 PCs. PC1 explains 19% of the total variance, which means that nearly one-fifth of the information in the dataset (53 variables) can be encapsulated by just that one Principal
```

Component. PC1:PC14 explains 80% of the variance. So, by knowing the position of a sample in relation to just PC1 and PC14, you can get a very accurate view on where it stands in relation to other samples, as PC1:PC14 can explain 80% of the variance.

3.1.2 HIERARCHICAL AGGLOMERATIVE CLUSTERING

We have very uneven clusters, cluster 1 contains 178 members which is most of the countries. Thus this is not very effective in clustering the countries. It can be noted that new deaths on Feb12 in cluster 10 is 14,320 whereas for cluster 1 for the same day was 47. It would be interesting to see why the massive difference between the two clusters. See Appendix A for Dendrogram.

3.1.3 K-MEANS CLUSTERING



```
# K-means clustering
```{r}
###Non-hierarchical cluster analysis
nor[is.nan(nor)] <- 0
nor[is.na(nor)] <- 0
set.seed(123)##fixes the randomness
kc<-kmeans(nor,10)
kc
#This gives very different results from before, (between_SS / total_SS =  65.4 %) is quiet good results,
```

K-means clustering with 10 clusters of sizes 1, 30, 32, 9, 38, 33, 2, 31, 24, 1
```



```
Within cluster sum of squares by cluster:
 [1]   0.00000  61.46328 143.53371  96.51426 150.53803 167.70361  99.48791  67.26991  73.19058
[10]   0.00000 109.01350  53.52529  75.13766   0.00000 228.59369   0.00000  59.86075 132.46793
[19]  52.48609   0.00000
 (between_SS / total_SS =  83.3 %)
```

Initially specify 10 clusters given quiet good results, this means that the clusters are performing well, however when specify 20clusters, it performed much better to (between_SS / total_SS = 83.3 %).

With K-mean clustering, we can see a more distributed clusters compared to agglomerative clustering. Thus a more effective clustering solution for our data. See Appendix A for 20 cluster memberships.

3.2 Dataset splitting

Observations were split into 70% for training and 30% for testing. Data clean was necessary to get data ready for splitting and ready for Support Vector Machine analysis. See Appendix A for R code of split.

3.2.1 TRAINING SET & TEST SET

The data was split using the createDataPartition function from caret package, for a stratified random split of the data. As the same proportion of data from all countries/locations was desired for both the raining and test set. K-fold Cross validation were also applied to the models, where k = 10. See Appendix A.



```
## Stratified random split into the Training set and Test set
```{r}
## Using the createDataPartition function from caret package, does a stratified random split of the data. We
need the same amount of data from all countries/locations.
library(caret)
train.index <- createDataPartition(Covid.SVM2$location, p = .7, list = FALSE)
training_set <- Covid.SVM2[ train.index,]
test_set  <- Covid.SVM2[-train.index,]
```
```

4. Machine learning predictions

4.1 Model training

The first Support Vector Machine Classifier model was build using the 'radial basis Kernel' function on all variables in the dataset:

classifier = svm(formula = Traffic_warning ~ ., data = training_set, type = 'C-classification', kernel = 'radial')

SVM Models	Accuracy	Improved		
Model 1: Radial Kernel: svm(Traffic_warning ~ all variables)	0.9871801			
Model 1: Radial Kernel: svm(Traffic_warning ~ all variables) K-fold cross validation	0.9882540	yes	0.0010739	
Model 2: Linear Kernel: svm(Traffic_warning ~ all variables)	0.9980621			
Model 2: Linear Kernel: svm(Traffic_warning ~ all variables) K-fold cross validation	0.9982222	yes	0.0001601	
Model 3: Linear Kernel: svm(Traffic_warning ~ AvgTemp)	0.9322236			
Model 3: Linear Kernel: svm(Traffic_warning ~ AvgTemp) K-fold cross validation	0.9326138	yes	0.0003902	
Model 4: Linear Kernel: svm(Traffic_warning ~ all variables) - UK data only	0.9026549			
Model 4: Linear Kernel: svm(Traffic_warning ~ all variables) - UK data only - K fold cross validation	0.4435083	no	-0.4591466	**reached max iterations
Model 4: Linear Kernel: svm(Traffic_warning ~ all variables) - UK data only - chose Best model	0.9	yes	0.4564917	

Accuracy rate achieved was 0.9871801. Then with k-fold cross validation, where k = 10 the average accuracy rate achieved was 0.9882540. This is a small improvement of 0.0010739. See Appendix A for the accuracy rate for for the 10 folds.

Model2: Support Vector Machine Classifier model was build using the 'Linear basis Kernel' function on all variables in the dataset :
classifier = svm(formula = Traffic_warning ~ ., data = training_set, type = 'C-classification', kernel = 'linear')
Accuracy rate achieved was 0.9980621. Then with k-fold cross validation, where k = 10 the average accuracy rate achieved was 0.9982222. This is a small improvement of 0.0001601. See Appendix A for the accuracy rate for for the 10 folds.
The linear model showed a higher accuracy, although not by much. Taking the linear model as the better one of the Model1 and Model2.

Model3: Support Vector Machine Classifier model was build using the 'Linear basis Kernel' function with only the AvgTemp variable:
classifier = svm(formula = Traffic_warning ~ AvgTemp, data = training_set, type = 'C-classification', kernel = 'linear')
Accuracy rate achieved was 0.9322236. Then with k-fold cross validation, where k = 10 the average accuracy rate achieved was 0.9326138. This is a small improvement of 0.0003902. See Appendix A for the accuracy rate for for the 10 folds.
The previous two models showed a higher accuracy rates which included all other variables.

Model3: Support Vector Machine Classifier model was then build using the 'Linear basis Kernel' function on all variables in the dataset, but filtered to the United Kingdom only:
classifier = svm(formula = Traffic_warning ~ ., data = UK.training_set, type = 'C-classification', kernel = 'linear', scale = TRUE)
Accuracy rate achieved was 0.9026549. Then with k-fold cross validation, where k = 10 the average accuracy rate achieved was 0.4435083. Accuracy rates for the folds fluctuated from 0.1851852 to 0.8888889, please see Appendix A for screenshot. R notebook also showed an error message indicating that iterations have reached maximum.

Model 4 was built on a random sample of model3, in order to tune the model using the smaller sample to select the optimal model.
tune_classifier <- tune(svm, Traffic_warning~., data = UK.training_set1, type = 'C-classification', kernel = 'linear', ranges = list(epsilon = seq(0,1,0.1), cost = 2 ^(2:7)))
And the accuracy rate achieved was 0.900000 using the best model selected. This did not improve the previous Model3. See Appendix A for screenshot.

4.2 Model evaluation and testing

The choice of model selection was influenced by some advantages of Support Vector Machine include high accuracy in prediction, it allows compact model representation, it's not prone to overfitting and is insensitive to noise as compared to other similar methods of classification and regression. (Pandini, 2021). Given the high dimensionality of the Covid data with 50+ attributes, it was one of the preferred methods to explore given the characteristics of the dataset collected. Table of results displayed as below.

All of the models produced fairly high accuracy rates in predicting the correct Covid status for the observations, with the lowest being 0.90 and highest 0.9982222. The best model was the linear model after the 10-fold cross validation.

The 10-fold cross validation always improved the accuracy rate for the models, except for the model where maximum iterations was reached as the dataset was too large for my computer to handle. And this was more of a technical issue to be solved.

The best model using average temperature variable was:
Model 3: Linear Kernel: svm(Traffic_warning ~ AvgTemp) K-fold cross validation
When the model was evaluated on the test set, this is to extrapolate predictions on unseen data cases. It achieved a 0.9326138 accuracy rate, which means 93.26138% of the time perform accurately in predicting the Covid status of observations based on the given average temperature.

This implies that given the daily average temperature, the model correctly predicts the Covid status 93.26138% of the time. Covid status can be Green, Orange or Red. Where green status means a country has 250 or less daily new Covid infections per million of the population. Orange status, more than 250 and up to 500 new cases per million, and Red status represents 500 new infections or more per million of the population.

However, it was also noted that Model2 achieved the highest accuracy rate of all the models, at 0.9982222. This is 99.82222% of the time correctly predicting the Covid status of a given observation. It is to note that this model utilised all 56 variables within the dataset, this includes all the variables from the covid data as well as the weather metrics of average temperature and average precipitation.

During execution, one of the disadvantages of SVM was that it was fairly slow in applying the k-fold cross validation to the dataset.The training dataset was 50MB and 21.3KB respectively, training the dataset took approximately 5mins using R notebook on a 16GB MacOS Big Sur Version 11.4. Sub-sampling a smaller dataset was then considered for tuning the model to avoid such technical issues.

4.2.1 SAMPLING AND CROSS-VALIDATION

Stratified sampling was utilised through the analysis. Given the nature of the dataset, the researchers wanted to include samples from all location equally for the prediction. Therefore, applying stratified sampling, this ensures the even sampling of all locations presented within the dataset. As the same proportion of data from all countries/locations is presented in each of the strata when using this technique.

This was also applied to ensure that each country in the dataset is properly represented in both the training and test sets. This is one main advantage of the stratified method, allowing primitive safeguard against uneven representation in the training and test datasets. (Witten, 2017).

K-fold cross validation was also used to further mitigate any bias by repeating the whole process k times with different random sample observations and exploring the results using k-fold validation where both the training and test data is partitioned to 10 fixed number of folds. Perform individual assessment and mean analysis on the 10 folds and and cross validate with previous achieved accuracy rates. K- fold cross validation improved the accuracy rate for all the models in this analysis, with the exception of a fault caused by technical error in one model.

5. High performance computational implementation

Given the circumstances and the limited time available for this analysis, it was applied in mapreduce to achieve the average temperature for each country for the training and test datasets. This was implemented as a way of validating the results of methods applied in Jupiter notebook and R markdown notebook. The average temperature should be equal across all instances and we validate this using mapreduce method, as indicated below. With more time, much of this component can be further explored to help to solve big data issues and improve processing/ running time.

Feb21 Average temperature Output check			
Mapreduce Output		R Output	
Afghanistan	2.59	Afghanistan	2.59
Albania	3.89	Albania	3.89
Algeria	6.78	Algeria	6.78
American Sa	14.17	American Sa	14.17
Angola	12.50	Angola	12.50
Antarctica	-0.74	Antarctica	-0.74
Argentina	8.91	Argentina	8.91
Armenia	0.74	Armenia	0.74
Australia	11.10	Australia	11.10
Austria	-0.48	Austria	-0.48
Azerbaijan	1.61	Azerbaijan	1.61
Bahamas	1.00	Bahamas	1.00
Bahrain	8.28	Bahrain	8.28
Belgium	0.28	Belgium	0.28
Belize	12.22	Belize	12.22
Benin	13.50	Benin	13.50
Bermuda [Ur	6.83	Bermuda [Ur	6.83
Bolivia	12.45	Bolivia	12.45
Bosnia and H	1.13	Bosnia and H	1.13
Botswana	11.83	Botswana	11.83
Brazil	13.02	Brazil	13.02
Brunei	13.39	Brunei	13.39







See Appendix A for screenshots

6. Performance Evaluation and comparison of methods

To reflect on the research question, ***'Does temperature affect the number of Covid infections in each country? Using Machine Learning techniques to predict infection status.'*** Support Vector Machine models were utilised for this analysis and prediction, achieving 93% accuracy rate when using average temperature only to predict the Covid status of the observations. This algorithm was chosen because it is one of the most used robust prediction methods that can be applied to many use cases involving classifications. SVM works by finding an optimal separation line called a hyperplane to accurately separate the classes. But this process was lengthy and took a lot of processing power. Sub-sampling to a smaller training set had to be considered when tuning the models. An intermediate step in SVM is to build an *nxn* matrix by calculating n-square dot product, in order to find the maximising margin. This increased the computational complexity compared to other classification methods.

Moreover, a Random Forest model achieved 95% accuracy rate prior to pruning. And 98.5% accuracy after pruning. This model gives the probability of belonging to a class. This algorithm was simpler to train and faster in performing the predictions, the only parameters to tune were the number of trees. The results from this model was easier to interpret than that of the SVM model. Besides the confusion matrix, quantitative values for the important variables can be easily identified using the *importance()* function within the randomForest package. The *varImpPlot()* function will display a very useful plot of the important variables. (See Appendix A for plot). Other metrics such as predictor variables used in the random forest are also easily displayed with another useful function called *varUsed().*

In conclusion, with similar achieved accuracy rate in both methods, the Random forest algorithm was easier to implement with less processing time and computational power required. See Appendix A for more comprehensive results table.

SVM Models	Accuracy achieved
Model 2: Linear Kernel: svm(Traffic_warning ~ all variables)	0.9982222

Random Forest Models	Accuracy achieved
Model 2: Pruned Random Forest model	0.9848000

7. Conclusion

7.1 Discussion of findings

Support Vector Machine and Random forest algorithm was used for the classification models to predict the Covid infection status of countries. It was concluded that both methods achieved similar accuracy rate, above 98% achieved in predicting the correct classification of observation to “Green”, “Orange”, or “Red” classes. However, one disadvantage of SVM noted was the requirement of more processing power than that needed to run similar models. The SVM results were also not easily interpreted, whereas the Random Forest model was easier to understand.

Where possible, stratified sampling was applied in proportion to the location of the dataset, so as to ensure equal amount of observations from all location are sampled for the analysis. And the results indeed conclude to better accuracy rate where this is applied. In addition, k-fold cross validation increased the model accuracy 100% of the time when applied to the training algorithms.

Moreover, using R notebook for SVM classification have produced fairly high accuracy levels for predicting the Covid status of observations. The best linear SVM model achieved 93.26138% accuracy rate. Moreover, the same classification executed in Jupiter notebook also achieved 93% accuracy rate.
This means that the model correctly predicts the Covid status for a country given the average temperature ~93% of time using either R or Jupiter notebook.
Where Model2 achieved highest accuracy rate of 99.82222% which includes all variables in the dataset. The equivalent model executed in Python achieved 98% accuracy rate, 1.82222% less than that of the model produced by R. However, it must be noted that k-fold cross validation was omitted during the analysis in Jupiter notebook. See Appendix A for comparisons.

	R	Python
Model 3: Linear Kernel: svm(Traffic_warning ~ AvgTemp) K-fold cross validation	0.9326138	0.93
Model 2: Linear Kernel: svm(Traffic_warning ~ all variables) K-fold cross validation	0.9982222	0.98

It was noted that the analysis in R required more local memory than that of when using Jupiter notebook to do similar job. Processing time was also noted to be faster using Python while running the support vector classifications.

8. Data management plan and author contribution statement

Data Management Plan

As a collective team effort, it was decided that the dataset are large enough to satisfy the course requirements but can also be stored locally on individual computers of the students. Options of sharing work-in-progress through GitHub was also considered however never materialised within the group. For sharing, most of team exchange was done via the Whatapps group chat, but the actual work file was stored and shared through OneDrive. Individual work have since moved to Github: https://github.com/1043795/CS5811_DistributedDataAnalysis_GroupProject

“Authorship Contribution" statement (ACS):
Suzy Zhu (1043795) collected the temperature data. Hamad Othman (1505526) collected the Covid dataset. Data clean and pre-processing jointly contributed by: Suzy Zhu (1043795), Hamad Othman (1505526), Aleksy Herominski (1122327), Karmjeet Sandhu (1422758); however EDA was mainly individual contribution. Machine Learning and HPCI techniques, individual work by Suzy Zhu (1043795).

References

Gangwal, R. (2019). A Data Scientist's guide to 8 Types of Sampling Techniques. [Blog} Analytics Vidhya. Available at: https://www.analyticsvidhya.com/blog/2019/09/data-scientists-guide-8-types-of-sampling-techniques/ [Accessed 02 April, 2021].
Lin, J. & Dyer, C. (2013). Data Intensive Text Processing with Mapreduce. pp31
Marshan, A. (2021) Python Lecture 11.
Pandini, A. (2021) CS5706 Machine Learning, Lecture8: Predictive Methods II: SVM and k-NN.
Pandini, A. (2021) CS5706 Machine Learning, Lecture9: Performance Evaluation.
Tucker, A. (2021) CS5607: High Performance Computational Infrastructures: Algorithm Design 1
Tucker, A. (2021) CS5607: High Performance Computational Infrastructures: MapReduce Introduction
Witten, I., Frank, E., Hall, M., Pal, C. (2017). Data Mining. Cambridge: Elsevier, pp. 167.
Witten, I., Frank, E., Hall, M., Pal, C. (2017). Data Mining. Cambridge: Elsevier, pp. 315.
Xie, J., Zhu, Y.(2020). Association between ambient temperature and COVID-19 infection in 122 cities from China. Science Direct Journal, [online] Volume 724, 138201. Available at: https://doi.org/10.1016/j.scitotenv.2020.138201 [Accessed 20 June, 2021].
Iqbal, N., Fareed, Z., Shadzed, F., Xin, H., Shahzad, U., Ma, L. (2020) The nexus between COVID-19, temperature and exchange rate in Wuhan city: New findings from partial and multiple wavelet coherence. Science Direct Journal, [online] Volume 729, 138916. Available at: https://doi.org/10.1016/j.scitotenv.2020.138916 [Accessed 20 June, 2021].
Briz-Redon, A., Serrano-Aroca, A., (2020) A spatio-temporal analysis for exploring the effect of temperature on COVID-19 early evolution in Spain, Science Direct Journal, [online] Volume 728, 138811. Available at: https://doi.org/10.1016/j.scitotenv.2020.138811 [Accessed 20 June, 2021].

Appendix

2.1 Data Visualisation

```
#Scatter Plot of two features of the training set against each other
training_set.plot.scatter(x = 'total_cases_per_million', y = 'AvgTemp', s = 1, c = 'Traffic_warning')
plt.xlabel('total_cases_per_million')
plt.ylabel('AvgTemp')
plt.title('total_cases_per_million vs AvgTemp')
plt.show()
```



3.1.1 Principal Component Analysis

PC loading for the first 14 PCs:

	PC1	PC2	PC3	PC4	PC5	PC6	PC7	PC8	PC9	PC10	PC11	PC12	PC13	PC14
total_cases	-0.10377453	0.298929954	-0.089066279	0.0383986463	-0.076412588	0.094683039	-0.0246458419	0.033709327	-0.025642623	0.0080895970	-0.0140246564	0.001102322	-0.0060107080	4.071835e-03
new_cases	-0.10077795	0.272185926	-0.061600871	0.1026323392	-0.129588316	0.210680875	-0.0519001906	0.045538324	-0.017131429	-0.0061467028	-0.0227911848	0.029673300	-0.0091078790	-1.112896e-02
new_cases_smoothed	-0.10216800	0.276539496	-0.062982336	0.0993193428	-0.125229512	0.202127257	-0.0541975304	0.045605856	-0.018392672	-0.0044606669	-0.0220072711	0.026531669	-0.0083957757	-1.021732e-02
total_deaths	-0.10067638	0.294705380	-0.091582199	0.0456381999	-0.110251791	0.125028720	-0.0374671886	0.042972709	-0.018676964	-0.0001841271	-0.0197437389	0.003773205	-0.0036572288	-8.376452e-03
new_deaths	-0.09897271	0.279467726	-0.079088629	0.0609655036	-0.137893006	0.157923506	-0.0445307638	0.035021330	-0.001305808	-0.0221121151	-0.0249923951	0.041555387	0.0008272505	-2.720555e-02
new_deaths_smoothed	-0.10039017	0.284247175	-0.080395393	0.0618051865	-0.135824861	0.156221871	-0.0533508524	0.039519528	-0.004871002	-0.0194436374	-0.0232418666	0.035552860	-0.0001532657	-2.639337e-02
total_cases_per_million	-0.18711137	-0.084781172	0.168739285	-0.0819224697	-0.171913286	-0.020775421	0.0644540975	-0.123550581	-0.088841053	0.1940071712	0.0253642106	-0.206655302	0.0292974724	1.358779e-01
new_cases_per_million	-0.16497454	-0.016528901	0.158712707	-0.0561327358	-0.201551213	-0.035470957	0.0594721573	-0.165144396	-0.034570263	0.1042246223	0.0085745853	-0.054497713	-0.0145237027	7.144789e-02
new_cases_smoothed_per_million	-0.19022736	-0.018252141	0.180536854	-0.0614872991	-0.214478401	-0.046807314	0.0342410840	-0.163078970	-0.045215697	0.1014495365	0.0107070089	-0.042516769	-0.0116887805	6.667260e-02
total_deaths_per_million	-0.18037796	0.005601971	0.143904414	-0.0183038258	-0.183622774	-0.041847096	0.0425979175	-0.137773939	0.053466110	0.1295350379	0.0173424240	-0.206750419	0.0543074378	8.083773e-02
new_deaths_per_million	-0.14540549	-0.007323614	0.137580262	-0.0229189310	-0.220296855	-0.076183065	0.0328192657	-0.205101769	0.091383658	-0.0034228904	-0.0359369146	0.054588567	0.0665950166	1.740051e-02
new_deaths_smoothed_per_million	-0.17926118	-0.010106719	0.163048965	-0.0265380036	-0.240950521	-0.089703808	0.0113627316	-0.205625542	0.088850346	-0.0014992681	-0.0230703856	0.028143175	0.0568085712	1.477096e-02
reproduction_rate	-0.15653227	-0.105074301	-0.168505036	0.0381365029	0.014689155	0.037684892	-0.0311143206	-0.112281066	-0.069388695	-0.0632615697	0.0047179172	0.150803860	-0.0587902234	-1.750615e-01
icu_patients	-0.13121384	0.065681278	0.182613146	0.2439846494	0.241077039	-0.005587659	0.0126806526	-0.053704113	0.130343094	-0.0148283865	-0.0165361584	0.078534107	0.0211081346	-7.543194e-02
icu_patients_per_million	-0.16553043	-0.016473137	0.190569511	0.0381653402	-0.046382606	-0.105673407	-0.0407951952	0.006937163	0.120726969	-0.2268279106	-0.0331288375	0.267351577	0.0036107338	-1.150257e-01
hosp_patients	-0.14021857	0.062039550	0.195574278	0.2383178212	0.223671867	-0.012788103	0.0196391248	-0.062009547	0.130397921	-0.0402251505	-0.0209732579	0.098319322	0.0222235541	-7.922655e-02
hosp_patients_per_million	-0.16579118	-0.027457726	0.186518361	0.0100790531	-0.103888809	-0.114244563	-0.0020188293	-0.014885217	0.096785393	-0.2628131512	-0.0177417215	0.266547376	-0.0014799387	-7.173718e-02

Importance of PC components:

```
summary(Covid.pca)
# PC1 explains 19% of the total variance, which means that nearly one-fifth of the information in the dataset (53 variables) can be encapsulated by just
that one Principal Component. PC1:PC14 explains 80% of the variance. So, by knowing the position of a sample in relation to just PC1 and PC14, you can
get a very accurate view on where it stands in relation to other samples, as PC1:PC14 can explain 80% of the variance.
```

```
Importance of components:
                          PC1    PC2     PC3     PC4     PC5     PC6     PC7     PC8     PC9    PC10    PC11    PC12    PC13    PC14    PC15
Standard deviation     3.2737 2.8521 2.19234 1.86693 1.79538 1.61954 1.56398 1.4664 1.32310 1.11366 1.08094 1.01645 0.98524 0.95952 0.93895
Proportion of Variance 0.1914 0.1453 0.08583 0.06224 0.05756 0.04684 0.04368 0.0384 0.03126 0.02215 0.02086 0.01845 0.01733 0.01644 0.01574
Cumulative Proportion  0.1914 0.3366 0.42247 0.48471 0.54227 0.58910 0.63278 0.6712 0.70244 0.72459 0.74545 0.76390 0.78124 0.79768 0.81342
                          PC16    PC17    PC18    PC19    PC20    PC21    PC22    PC23    PC24    PC25    PC26    PC27    PC28    PC29    PC30
Standard deviation     0.90027 0.89469 0.85919 0.80729 0.79028 0.75504 0.73559 0.69517 0.68677 0.6734 0.64407 0.6078 0.58313 0.5450 0.53862
Proportion of Variance 0.01447 0.01429 0.01318 0.01164 0.01115 0.01018 0.00966 0.00863 0.00842 0.0081 0.00741 0.0066 0.00607 0.0053 0.00518
Cumulative Proportion  0.82790 0.84219 0.85537 0.86701 0.87816 0.88834 0.89801 0.90663 0.91506 0.9232 0.93056 0.9372 0.94323 0.9485 0.95372
                          PC31    PC32    PC33    PC34    PC35    PC36    PC37    PC38    PC39    PC40    PC41    PC42    PC43    PC44    PC45
Standard deviation     0.53029 0.4905 0.45402 0.45262 0.43172 0.40364 0.39272 0.38949 0.37247 0.33933 0.3266 0.31589 0.30593 0.28784 0.27436
Proportion of Variance 0.00502 0.0043 0.00368 0.00366 0.00333 0.00291 0.00275 0.00271 0.00248 0.00206 0.0019 0.00178 0.00167 0.00148 0.00134
Cumulative Proportion  0.95874 0.9630 0.96672 0.97038 0.97370 0.97661 0.97937 0.98208 0.98455 0.98661 0.9885 0.99030 0.99197 0.99345 0.99479
                          PC46    PC47    PC48    PC49    PC50    PC51    PC52    PC53    PC54    PC55    PC56
Standard deviation     0.2482 0.21348 0.18540 0.17679 0.16554 0.15133 0.13787 0.12843 0.12411 0.11619 0.06458
Proportion of Variance 0.0011 0.00081 0.00061 0.00056 0.00049 0.00041 0.00034 0.00029 0.00028 0.00024 0.00007
Cumulative Proportion  0.9959 0.99671 0.99732 0.99788 0.99837 0.99878 0.99911 0.99941 0.99968 0.99993 1.00000
```

3.1.2 Hierarchical agglomerative clustering

```
# Hierarchical agglomerative clustering
```{r}
# Hierarchical agglomerative clustering
Feb12.hclust = hclust(distance)
plot(Feb12.hclust)
plot(Feb12.hclust,labels=Feb12.hclust$location,main='Cluster Dendrofram_Feb12', hang=-1)
rect.hclust(Feb12.hclust, k = 10, border = "green")
#boxing the data to 10 clusters..
## note that the default is always - complete
```
```







3.1.3 K-means clustering

```r
#try 20 clusters
set.seed(123)
kc<-kmeans(nor,20)
kc
# then 20 clusters produced even better results, (between_SS / total_SS =  83.3 %)
```

```
K-means clustering with 20 clusters of sizes 1, 10, 21, 5, 7, 15, 2, 10, 24, 1, 5, 10, 14, 1, 17, 1, 18,
33, 5, 1
```

3.2 Dataset splitting

```r
# Data Split
```{r}
# Subset data ready for split into training data and test data
Feb01 <- subset(Covid.Temp, Covid.Temp$date == "2021-02-01")
Feb01 <- as.data.frame(Feb01)
str(Feb01)
# removing Nas
Feb01[is.na(Feb01)] <- 0
# converting HDI_Rank to factor
Feb01$HDI_Rank <- as.factor(Feb01$HDI_Rank)
# removing character and date columns
Feb01_svm <- Feb01[, -c(1:4, 34)]
str(Feb01_svm)

# set random seed
set.seed(1999)
# create a 70/30 training/test set split
n_rows <- nrow(Feb01_svm)
# sample 70% (n_rows * 0.7) indices in the ranges 1:nrows
training_idx <- sample(n_rows, n_rows * 0.7)
# filter the data frame with the training indices (and the complement)
training_svm_data <- Feb01_svm[training_idx,]
test_svm_data <- Feb01_svm[-training_idx,]
nrow(training_svm_data)
nrow(test_svm_data)
```
```

```
[1] 168
[1] 73
```

3.2.1 Training set & Test set

```
# Using Pandas to load in training csv file and the test set
import pandas as pd
training_set = pd.read_csv('/Users/suzie/Desktop/MSc_DSA_2020/CS5811_Distributed Data Analysis_Martin Shepherd/PCA_S
test_set = pd.read_csv('/Users/suzie/Desktop/MSc_DSA_2020/CS5811_Distributed Data Analysis_Martin Shepherd/PCA_SVM/t
```

```
# Divide the training dataset into a smaller training set (II) and a validation set using
# the train test split function, using the 70/30 rule making test set 30%
from sklearn.model_selection import train_test_split
y = training_set.HDI_Rank
SXtrain, Xvalid, Sytrain, vtest = train_test_split(training_set.iloc[:, :-56], y, test_size = 0.3, random_state=4)
# Check that we have split this properly
print(SXtrain.shape, Sytrain.shape)
print(Xvalid.shape, vtest.shape)
```

```
(117, 1) (117,)
(51, 1) (51,)
```

```
# Divide the training dataset into a smaller training set (II) and a validation set using
# the train test split function, using the 80/20 rule making test set 30%
from sklearn.model_selection import train_test_split
y = training_set.HDI_Rank
SXtrain, Xvalid, Sytrain, vtest = train_test_split(training_set.iloc[:, :-56], y, test_size = 0.3, random_state=4)
# Check that we have split this properly
print(SXtrain.shape, Sytrain.shape)
print(Xvalid.shape, vtest.shape)
```

```
(97, 1) (97,)
(42, 1) (42,)
```

```
#Look at headers for both the training and test dataset to ensure
#we have 57 dimensions and 1 target value which is categorical
print(training_set.columns)
print(test_set.columns)
```

```
Index(['total_cases', 'new_cases', 'new_cases_smoothed', 'total_deaths',
       'new_deaths', 'new_deaths_smoothed', 'total_cases_per_million',
       'new_cases_per_million', 'new_cases_smoothed_per_million',
       'total_deaths_per_million', 'new_deaths_per_million',
       'new_deaths_smoothed_per_million', 'reproduction_rate', 'icu_patients',
       'icu_patients_per_million', 'hosp_patients',
       'hosp_patients_per_million', 'weekly_icu_admissions',
       'weekly_icu_admissions_per_million', 'weekly_hosp_admissions',
       'weekly_hosp_admissions_per_million', 'new_tests', 'total_tests',
       'total_tests_per_thousand', 'new_tests_per_thousand',
       'new_tests_smoothed', 'new_tests_smoothed_per_thousand',
       'positive_rate', 'tests_per_case', 'total_vaccinations',
       'people_vaccinated', 'people_fully_vaccinated', 'new_vaccinations',
       'new_vaccinations_smoothed', 'total_vaccinations_per_hundred',
       'people_vaccinated_per_hundred', 'people_fully_vaccinated_per_hundred',
       'new_vaccinations_smoothed_per_million', 'stringency_index',
       'population', 'population_density', 'median_age', 'aged_65_older',
       'aged_70_older', 'gdp_per_capita', 'extreme_poverty',
       'cardiovasc_death_rate', 'diabetes_prevalence', 'female_smokers',
       'male_smokers', 'handwashing_facilities', 'hospital_beds_per_thousand',
       'life_expectancy', 'human_development_index', 'AvgTemp', 'Prcp',
       'Traffic_warning'],
      dtype='object')
Index(['total_cases', 'new_cases', 'new_cases_smoothed', 'total_deaths',
       'new_deaths', 'new_deaths_smoothed', 'total_cases_per_million',
       'new_cases_per_million', 'new_cases_smoothed_per_million',
       'total_deaths_per_million', 'new_deaths_per_million',
       'new_deaths_smoothed_per_million', 'reproduction_rate', 'icu_patients',
       'icu_patients_per_million', 'hosp_patients',
       'hosp_patients_per_million', 'weekly_icu_admissions',
       'weekly_icu_admissions_per_million', 'weekly_hosp_admissions',
       'weekly_hosp_admissions_per_million', 'new_tests', 'total_tests',
       'total_tests_per_thousand', 'new_tests_per_thousand',
       'new_tests_smoothed', 'new_tests_smoothed_per_thousand',
       'positive_rate', 'tests_per_case', 'total_vaccinations',
       'people_vaccinated', 'people_fully_vaccinated', 'new_vaccinations',
       'new_vaccinations_smoothed', 'total_vaccinations_per_hundred',
       'people_vaccinated_per_hundred', 'people_fully_vaccinated_per_hundred',
       'new_vaccinations_smoothed_per_million', 'stringency_index',
       'population', 'population_density', 'median_age', 'aged_65_older',
       'aged_70_older', 'gdp_per_capita', 'extreme_poverty',
       'cardiovasc_death_rate', 'diabetes_prevalence', 'female_smokers',
       'male_smokers', 'handwashing_facilities', 'hospital_beds_per_thousand',
       'life_expectancy', 'human_development_index', 'AvgTemp', 'Prcp',
       'Traffic_warning'],
      dtype='object')
```

```
str(training_svm_data)
```

```
'data.frame':   168 obs. of  51 variables:
 $ total_cases                          : num  26927 0 0 22842 64983 ...
 $ new_cases                            : num  484 0 0 71 826 ...
 $ new_cases_smoothed                   : num  276 0 0 127 831 ...
 $ total_deaths                         : num  638 0 0 673 323 ...
 $ new_deaths                           : num  10 0 0 2 7 58 0 5 0 0 ...
 $ new_deaths_smoothed                  : num  8 0 0 1.71 5.14 ...
 $ total_cases_per_million              : num  1608 0 0 255 3035 ...
 $ new_cases_per_million                : num  24.128 0 0 8.793 38.574 ...
 $ new_cases_smoothed_per_million       : num  16.5 0 0 1.42 38.8 ...
 $ total_deaths_per_million             : num  38.1 0 0 7.51 15.08 ...
 $ new_deaths_per_million               : num  0.597 0 0 0.022 0.327 0.529 0 0.286 0 0 ...
 $ new_deaths_smoothed_per_million      : num  0.478 0 0 0.019 0.24 0.671 0 0.294 0 0 ...
 $ icu_patients                         : num  0 0 0 0 0 0 0 0 0 0 ...
 $ icu_patients_per_million             : num  0 0 0 0 0 0 0 0 0 0 ...
 $ hosp_patients                        : num  0 0 0 0 0 0 0 0 0 0 ...
 $ hosp_patients_per_million            : num  0 0 0 0 0 0 0 0 0 0 ...
 $ new_tests                            : num  2411 0 0 671 12340 ...
 $ total_tests                          : num  339674 0 0 0 1734276 ...
 $ total_tests_per_thousand             : num  20.3 0 0 0 81 ...
 $ new_tests_per_thousand               : num  0.144 0 0 0.007 0.576 0.233 0 0 0 0 ...
 $ new_tests_smoothed                   : num  2036 0 0 636 15604 ...
 $ new_tests_smoothed_per_thousand      : num  0.122 0 0 0.007 0.729 0.323 0 0 0 0 ...
 $ positive_rate                        : num  0.136 0 0 0.199 0.053 0.85 0 0 0 0 ...
 $ tests_per_case                       : num  7.4 0 0 5 18.8 20.2 0 0 0 0 ...
 $ total_vaccinations                   : num  0 0 0 0 95550 ...
 $ people_vaccinated                    : num  0 0 0 0 0 0 0 0 0 0 ...
 $ people_fully_vaccinated              : num  0 0 0 0 0 0 0 0 0 0 ...
 $ new_vaccinations                     : num  0 0 0 0 36396 ...
 $ new_vaccinations_smoothed            : num  0 533 275 0 23888 ...
 $ total_vaccinations_per_hundred       : num  0 0 0 0 0.45 0 0 0 0 0 ...
 $ people_vaccinated_per_hundred        : num  0 0 0 0 0 0 0 0 0 0 ...
 $ people_fully_vaccinated_per_hundred  : num  0 0 0 0 0 0 0 0 0 0 ...
 $ new_vaccinations_smoothed_per_million: num  0 6268 4184 0 1116 ...
 $ stringency_index                     : num  50.5 0 0 32.4 81.9 ...
 $ population                           : num  16743930 85032 65720 89561404 21413250 ...
 $ population_density                   : num  82.3 147.9 256.5 35.9 342 ...
 $ median_age                           : num  18.7 0 0 17 34.1 25.2 0 21.7 0 0 ...
 $ aged_65_older                        : num  3.01 0 0 3.82 10.07 ...
 $ aged_70_older                        : num  1.8 0 0 1.75 5.33 ...
 $ gdp_per_capita                       : num  2471 0 49903 808 11669 ...
 $ extreme_poverty                      : num  38 0 0 77.1 0.7 0 0 0 0 0 ...
 $ cardiovasc_death_rate                : num  241 0 0 319 197 ...
 $ diabetes_prevalence                  : num  2.42 0 13.22 6.1 10.68 ...
 $ female_smokers                       : num  0.4 0 0 0 0.3 7.8 0 0 0 0 ...
 $ male_smokers                         : num  16.6 0 0 0 27 40.8 0 0 0 0 ...
 $ handwashing_facilities               : num  28.86 0 0 4.47 0 ...
 $ hospital_beds_per_thousand           : num  0 0 0 3.6 3 0 1.5 2.3 0 ...
 $ life_expectancy                      : num  67.9 81.4 83.9 60.7 77 ...
 $ human_development_index              : num  0.512 0 0 0.48 0.782 0.718 0 0.567 0.779 0 ...
 $ HDI_Rank                             : num  4 0 0 0 2 2 0 3 0 4 ...
 $ AvgTemp                              : num  13.2 0 0 0 12 ...
```

```
str(test_svm_data)
```

```
'data.frame':   73 obs. of  51 variables:
 $ total_cases                          : num  55059 78992 9972 19829 28823 ...
 $ new_cases                            : num  36 865 35 33 5 ...
 $ new_cases_smoothed                   : num  55.29 882.86 53.71 50.43 6.14 ...
 $ total_deaths                         : num  2404 1393 101 486 909 ...
 $ new_deaths                           : num  4 13 0 0 0 57 4 10 1 0 ...
 $ new_deaths_smoothed                  : num  2.714 9.857 0.571 0.714 0 ...
 $ total_cases_per_million              : num  1414 27449 129062 603 1130 ...
 $ new_cases_per_million                : num  0.925 300.577 452.986 1.004 0.196 ...
 $ new_cases_smoothed_per_million       : num  1.42 306.782 695.196 1.534 0.241 ...
 $ total_deaths_per_million             : num  61.8 484.1 1307.2 14.2 35.6 ...
 $ new_deaths_per_million               : num  0.103 4.517 0 0 0 ...
 $ new_deaths_smoothed_per_million      : num  0.07 3.425 7.396 0.022 0 ...
 $ icu_patients                         : num  0 0 0 0 0 294 0 0 0 0 ...
 $ icu_patients_per_million             : num  0 0 0 0 0 ...
 $ hosp_patients                        : num  0 0 0 0 0 ...
 $ hosp_patients_per_million            : num  0 0 0 0 0 ...
 $ new_tests                            : num  0 2960 0 0 38319 ...
 $ total_tests                          : num  0 346069 144933 0 12982095 ...
 $ total_tests_per_thousand             : num  0 120 1876 0 509 ...
 $ new_tests_per_thousand               : num  0 1.03 0 0 1.19 ...
 $ new_tests_smoothed                   : num  0 3534 605 0 33746 ...
 $ new_tests_smoothed_per_thousand      : num  0 1.23 7.83 0 1.32 ...
 $ positive_rate                        : num  0 0.25 0.089 0 0 0 0 0 0.034 0 0 ...
 $ tests_per_case                       : num  0 4 11.3 0 5493.4 ...
 $ total_vaccinations                   : num  0 0 1036 0 0 ...
 $ people_vaccinated                    : num  0 0 1036 0 0 ...
 $ people_fully_vaccinated              : num  0 0 0 0 0 ...
 $ new_vaccinations                     : num  0 0 0 0 0 3010 0 0 0 0 ...
 $ new_vaccinations_smoothed            : num  0 0 66 0 0 ...
 $ total_vaccinations_per_hundred       : num  0 0 1 34 0 0 2.53 0 0 0 ...
 $ people_vaccinated_per_hundred        : num  0 0 1 34 0 0 2.07 0 0 0 0 ...
 $ people_fully_vaccinated_per_hundred  : num  0 0 0 0 0 0 0.18 0 0 0 0 ...
 $ new_vaccinations_smoothed_per_million: num  0 0 854 0 0 ...
 $ stringency_index                     : num  12 60.2 55.6 63 78.2 ...
 $ population                           : num  38928341 2877800 77265 32866268 25499881 ...
 $ population_density                   : num  54.4 104.9 163.8 23.9 3.2 ...
 $ median_age                           : num  18.6 38 0 16.8 37.9 44.4 32.4 27.5 25 0 ...
 $ aged_65_older                        : num  2.58 13.19 0 2.4 15.5 ...
 $ aged_70_older                        : num  1.34 8.64 0 1.36 10.13 ...
 $ gdp_per_capita                       : num  1804 11803 0 5819 44649 ...
 $ extreme_poverty                      : num  0 1.1 0 0 0.5 0.7 0 14.8 0 0 ...
 $ cardiovasc_death_rate                : num  597 304 109 276 108 ...
 $ diabetes_prevalence                  : num  9.59 10.08 7.97 3.94 5.07 ...
 $ female_smokers                       : num  0 7.1 29 0 13 28.4 0.3 1 0 0 ...
 $ male_smokers                         : num  0 51.2 57.8 0 16.5 30.9 42.5 44.7 0 0 ...
 $ handwashing_facilities               : num  37.7 0 0 26.7 0 ...
 $ hospital_beds_per_thousand           : num  0.5 2.89 0 0 3.84 7.37 4.7 0.8 1.3 0 ...
 $ life_expectancy                      : num  64.8 78.6 83.7 61.1 83.4 ...
 $ human_development_index              : num  0.511 0.795 0.868 0.581 0.944 0.922 0.756 0.632 0.716 0 ...
 $ HDI_Rank                             : num  4 2 0 3 1 1 2 0 2 0 ...
 $ AvgTemp                              : num  2.58 3.89 0 12.5 11.1 ...
```

```{r}
###For March 01, 2020 data extraction into csv file for SVM in Python: this will allow us for the comparison of these two dates, beginning of pandemic and date of data collection.
Mar01 <- subset(Covid.Temp, Covid.Temp$date == "2020-03-01")
summary(Mar01)
#Removing character columns
Mar01 <- Mar01[, -c(1,2,4)]
gg_miss_var(Mar01, show_pct = TRUE)
Mar01[is.na(Mar01)] <- 0
table(Mar01$HDI_Rank)
str(Mar01)
# removing variable with mean of 0/characters, PCA does not allow
Mar01_svm <- Mar01[, -c(31)]
summary(Mar01_svm)
str(Mar01_svm)
#remove the countrycode too from dataframe
Mar01_svm <- Mar01[, -c(1)]
# set random seed
set.seed(1999)
# create a 70/30 training/test set split
n_rows <- nrow(Mar01_svm)
# sample 70% (n_rows * 0.7) indices in the ranges 1:nrows
training_idx <- sample(n_rows, n_rows * 0.7)
# filter the data frame with the training indices (and the complement)
training_svm_data <- Mar01_svm[training_idx,]
test_svm_data <- Mar01_svm[-training_idx,]
nrow(training_svm_data)
nrow(test_svm_data)

str(training_svm_data)
str(test_svm_data)
#write the training_svm_data and test_svm_data into two text files
write.table(training_svm_data, file = "training_svm_data_01Mar20_noloc.csv", sep = ",", row.names = FALSE)
write.table(test_svm_data, file = "test_svm_data_01Mar20_noloc.csv", sep = ",", row.names = FALSE)
# for execute the svm in python
```

```
[1] 139
[1] 60
```

```
# Saving to text files_for_prediction_python_notebook
```{r}
#prepare data for extraction with HDI groups 0-5
#removing character columns
#making the HDI_Rank column into factor
Feb01 <- subset(Covid.Temp, Covid.Temp$date == "2021-02-01")
summary(Feb01)
#Removing character columns prior to PCA
Feb01 <- Feb01[, -c(1,2,4)]
#Check that NAs are removed correctly and no other issues
summary(Feb01)
#Change the NAs to 0 as data is precious
Feb01 <- as.data.frame(Feb01)
gg_miss_var(Feb01, show_pct = TRUE)

table(Feb01$HDI_Rank)
Feb01[is.na(Feb01)] <- 0
#Check that NAs are gone
summary(Feb01)
str(Feb01)
# removing variable with mean of 0/characters, PCA does not allow
Feb01_svm <- Feb01[, -c(31)]
summary(Feb01_svm)
#remove the countrycode too from dataframe
Feb01_svm <- Feb01[, -c(1)]
# set random seed
set.seed(1999)
# create a 70/30 training/test set split
n_rows <- nrow(Feb01_svm)
# sample 70% (n_rows * 0.7) indices in the ranges 1:nrows
training_idx <- sample(n_rows, n_rows * 0.7)
# filter the data frame with the training indices (and the complement)
training_svm_data <- Feb01_svm[training_idx,]
test_svm_data <- Feb01_svm[-training_idx,]
nrow(training_svm_data)
nrow(test_svm_data)

str(training_svm_data)
str(test_svm_data)

#we do not need this to be a dataframe otherwise, it will read in the first row with index
#training_svm_data <- as.data.frame(training_svm_data)
#test_svm_data <- as.data.frame(test_svm_data)

#write the training_svm_data and test_svm_data into two text files
write.table(training_svm_data, file = "training_svm_data_01Feb21_noloc.csv", sep = ",", row.names = FALSE)
write.table(test_svm_data, file = "test_svm_data_01Feb21_noloc.csv", sep = ",", row.names = FALSE)
# for execute the svm in python
```

```
[1] 168
[1] 73
```
```

4.1 Model training

Model 1: Radial Kernel: svm(Traffic_warning ~ all variables)

```
# Fitting Radial Kernel: Traffic_warning ~ .
```{r}
library(e1071)
classifier = svm(formula = Traffic_warning ~ .,
                 data = training_set,
                 type = 'C-classification',
                 kernel = 'radial')
summary(classifier)
```
```

```
Call:
svm(formula = Traffic_warning ~ ., data = training_set, type = "C-classification", kernel = "radial")


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  radial
       cost:  1

Number of Support Vectors:  2919

 ( 1138 1281 500 )


Number of Classes:  3

Levels:
 Green Orange Red
```

```
## Ploting the Confusion Matrix
```{r}
cm = table(test_set[, 57], y_pred)
cm
```
```

```
        y_pred
         Green Orange   Red
  Green  18730     41     0
  Orange   131    696    15
  Red        9     62   441
```

```
```{r}
accuracy = (cm[1,1] + cm[2,2]+ cm[3,3]) / (cm[1,
cm[3,2])
accuracy
```
```

```
[1] 0.9871801
```

```
## Applying k-Fold Cross Validation_Radial Kernel
```{r}
library(caret)
library(lattice)
library(ggplot2)
# in creating the folds we specify the target feature (dependent variable) and # of folds
folds = createFolds(training_set$Traffic_warning, k = 10)
# in cv we are going to applying a created function to our 'folds'
cv = lapply(folds, function(x) { # start of function
  # in the next two lines we will separate the Training set into it's 10 pieces
  training_fold = training_set[-x, ] # training fold =  training set minus (-) it's sub test fold
  test_fold = training_set[x, ] # here we describe the test fold individually
  # now apply (train) the classifer on the training_fold
  classifier = svm(formula = Traffic_warning ~ .,
                   data = training_fold,
                   type = 'C-classification',
                   kernel = 'radial')
  # next step in the loop, we calculate the predictions and cm and we equate the accuracy
  # note we are training on training_fold and testing its accuracy on the test_fold
  y_pred = predict(classifier, newdata = test_fold[-57])
  cm = table(test_fold[, 57], y_pred)
  accuracy = (cm[1,1] + cm[2,2]+ cm[3,3]) / (cm[1,1] + cm[2,2] + cm[3,3] + cm[1,2]+ cm[1,3]+ cm[2,1]+ cm[2,3]+ cm[3,1]+
cm[3,2])
  return(accuracy)
})
```
```

```
```{r}
# For CV we can see we have 10 folds/iterations each with slight variations of accuracy.
knitr::include_graphics("cv.png")
cv
```
```

```
$Fold01
[1] 0.9862463

$Fold02
[1] 0.9894157

$Fold03
[1] 0.9885714

$Fold04
[1] 0.9881481

$Fold05
[1] 0.9885714

$Fold06
[1] 0.9879391

$Fold07
[1] 0.988997

$Fold08
[1] 0.9887807

$Fold09
[1] 0.9881481

$Fold10
[1] 0.9877223
```

```
## Mean of accuracy
```{r}
accuracy = mean(as.numeric(cv))
accuracy
```
```

```
[1] 0.988254
```

Model 2: Linear Kernel: svm(Traffic_warning ~ all variables)

```
# Fitting Linear Kernel: Traffic_warning ~ .
```{r}
library(e1071)
classifier = svm(formula = Traffic_warning ~ .,
                 data = training_set,
                 type = 'C-classification',
                 kernel = 'linear')

summary(classifier)
```
```

```
Call:
svm(formula = Traffic_warning ~ ., data = training_set, type = "C-classification", kernel = "linear")


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  linear
       cost:  1

Number of Support Vectors:  530

 ( 186 268 76 )


Number of Classes:  3

Levels:
 Green Orange Red
```

```
## Ploting the Confusion Matrix
```{r}
cm = table(test_set[, 57], y_pred)
cm
```
```

```
        y_pred
         Green Orange   Red
  Green  18743     18     0
  Orange     5    845    14
  Red        0      2   498
```

```
```{r}
# Printing accuracy rate
accuracy = (cm[1,1] + cm[2,2]+ cm[3,3]) / (cm[1,1] + cm[2,2] +
cm[3,2])
accuracy
```
```

```
[1] 0.9980621
```

```
## Applying k-Fold Cross Validation_Linear Kernel
```{r}
library(caret)
library(lattice)
library(ggplot2)
# in creating the folds we specify the target feature (dependent variable) and # of folds
folds = createFolds(training_set$Traffic_warning, k = 10)
# in cv we are going to applying a created function to our 'folds'
cv = lapply(folds, function(x) { # start of function
  # in the next two lines we will separate the Training set into it's 10 pieces
  training_fold = training_set[-x, ] # training fold =  training set minus (-) it's sub test fold
  test_fold = training_set[x, ] # here we describe the test fold individually
  # now apply (train) the classifer on the training_fold
  classifier = svm(formula = Traffic_warning ~ .,
                   data = training_fold,
                   type = 'C-classification',
                   kernel = 'linear')
  # next step in the loop, we calculate the predictions and cm and we equate the accuracy
  # note we are training on training_fold and testing its accuracy on the test_fold
  y_pred = predict(classifier, newdata = test_fold[-57])
  cm = table(test_fold[, 57], y_pred)
  accuracy = (cm[1,1] + cm[2,2]+ cm[3,3]) / (cm[1,1] + cm[2,2] + cm[3,3] + cm[1,2]+ cm[1,3]+ cm[2,1]+ cm[2,3]+ cm[3,1]+
cm[3,2])
  return(accuracy)
})
```
```

```
```{r}
# For CV we can see we have 10 folds/iterations each with slightest variations of accuracy.
cv
```
```

```
$Fold01
[1] 0.9976715

$Fold02
[1] 0.997884

$Fold03
[1] 0.9980952

$Fold04
[1] 0.9989418

$Fold05
[1] 0.9980952

$Fold06
[1] 0.9980952

$Fold07
[1] 0.9987299

$Fold08
[1] 0.9983072

$Fold09
[1] 0.9985185

$Fold10
[1] 0.9978836
```

```
## Mean of accuracy
```{r}
accuracy = mean(as.numeric(cv))
accuracy
```
```

```
[1] 0.9982222
```

Model 3: Linear Kernel: svm(Traffic_warning ~ AvgTemp)

```
# Fitting Linear SVM: Traffic_warning~AvgTemp
```{r}
summary(training_set)
library(e1071)
classifier = svm(formula = Traffic_warning ~ AvgTemp,
                 data = training_set,
                 type = 'C-classification',
                 kernel = 'linear')

summary(classifier)
```


Call:
svm(formula = Traffic_warning ~ AvgTemp, data = training_set, type = "C-classification", kernel = "linear")

Parameters:
   SVM-Type: C-classification
 SVM-Kernel: linear
       cost: 1

Number of Support Vectors: 5200

 (2016 2015 1169)

Number of Classes: 3

Levels:
 Green Orange Red
```

```
## Ploting the Confusion Matrix
```{r}
cm = table(test_set[, 57], y_pred)
cm
```
```

        y_pred
         Green Orange   Red
  Green 18761   0   0
  Orange   864   0   0
  Red   500   0   0
```

```
```{r}
# Printing accuracy rate
accuracy = (cm[1,1] + cm[2,2]+ cm[3,3]) / (cm[1,1] + cm[
cm[3,2])
accuracy
```
```

```
[1] 0.9322236
```

```
## Applying k-Fold Cross Validation_Linear Kernel
```{r}
library(caret)
library(lattice)
library(ggplot2)
# in creating the folds we specify the target feature (dependent variable) and # of folds
folds = createFolds(training_set$Traffic_warning, k = 10)
# in cv we are going to applying a created function to our 'folds'
cv = lapply(folds, function(x) { # start of function
  # in the next two lines we will separate the Training set into it's 10 pieces
  training_fold = training_set[-x, ] # training fold =  training set minus (-) it's sub test fold
  test_fold = training_set[x, ] # here we describe the test fold individually
  # now apply (train) the classifer on the training_fold
  classifier = svm(formula = Traffic_warning ~ AvgTemp,
                   data = training_fold,
                   type = 'C-classification',
                   kernel = 'linear')
  # next step in the loop, we calculate the predictions and cm and we equate the accuracy
  # note we are training on training_fold and testing its accuracy on the test_fold
  y_pred = predict(classifier, newdata = test_fold[-57])
  cm = table(test_fold[, 57], y_pred)
  accuracy = (cm[1,1] + cm[2,2]+ cm[3,3]) / (cm[1,1] + cm[2,2] + cm[3,3] + cm[1,2]+ cm[1,3]+ cm[2,1]+ cm[2,3]+ cm[3,1]+
cm[3,2])
  return(accuracy)
})
```

```{r}
# For CV we can see we have 10 folds/iterations each with slight variations of accuracy.
cv
```

$Fold01
[1] 0.9325011

$Fold02
[1] 0.9328959

$Fold03
[1] 0.9324868

$Fold04
[1] 0.9326984

$Fold05
[1] 0.9326984

$Fold06
[1] 0.9324868

$Fold07
[1] 0.9324868

$Fold08
[1] 0.9326984

$Fold09
[1] 0.9325011

$Fold10
[1] 0.9326842

```
## Mean of accuracy
```{r}
accuracy = mean(as.numeric(cv))
accuracy
```
```

```
[1] 0.9326138
```

Model 4: Linear Kernel: svm(Traffic_warning ~ all variables) - UK data only

```r
summary(UK.training_set)
# remove location column
UK.training_set <- UK.training_set[-c(1)]
UK.test_set <- UK.test_set[-c(1)]

library(e1071)
classifier = svm(formula = Traffic_warning ~ .,
                 data = UK.training_set,
                 type = 'C-classification',
                 kernel = 'linear', scale = TRUE)

summary(classifier)
```

```
Call:
svm(formula = Traffic_warning ~ ., data = UK.training_set, type = "C-classification", kernel = "linear",
    scale = TRUE)


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  linear
       cost:  1

Number of Support Vectors:  34

 ( 10 19 5 )


Number of Classes:  3

Levels:
 Green Orange Red
```

Ploting the Confusion Matrix

```r
cm = table(UK.test_set[, 57], y_pred)
cm
```

```
        y_pred
         Green Orange Red
  Green     80      1   0
  Orange     6     20   3
  Red        0      1   2
```

```r
# Printing accuracy rate
accuracy = (cm[1,1] + cm[2,2]+ cm[3,3]) / (cm[1,1] + cm[2,2] + cm[3,3] + cm[1,2]+ cm[1,3]+ cm[2,1]+ cm[2,3]+ cm[3,1]+
cm[3,2])
accuracy
```

```
[1] 0.9026549
```

Applying k-Fold Cross Validation_Linear Kernel

```r
library(caret)
library(lattice)
library(ggplot2)
# in creating the folds we specify the target feature (dependent variable) and # of folds
folds = createFolds(UK.training_set$Traffic_warning, k = 10)
# in cv we are going to applying a created function to our 'folds'
cv = lapply(folds, function(x) { # start of function
  # in the next two lines we will separate the Training set into it's 10 pieces
  training_fold = UK.training_set[-x, ] # training fold =  training set minus (-) it's sub test fold
  test_fold = UK.training_set[x, ] # here we describe the test fold individually
  # now apply (train) the classifer on the training_fold
  classifier = svm(formula = Traffic_warning ~ .,
                   data = training_fold,
                   type = 'C-classification',
                   kernel = 'linear')
  # next step in the loop, we calculate the predictions and cm and we equate the accuracy
  # note we are training on training_fold and testing its accuracy on the test_fold
  y_pred = predict(classifier, newdata = test_fold[-57])
  cm = table(test_fold[, 57], y_pred)
  accuracy = (cm[1,1] + cm[2,2]+ cm[3,3]) / (cm[1,1] + cm[2,2] + cm[3,3] + cm[1,2]+ cm[1,3]+ cm[2,1]+ cm[2,3]+ cm[3,1]+
cm[3,2])
  return(accuracy)
})
```

```r
# For CV we can see we have 10 folds/iterations each with slight variations of accuracy.
cv
```

```
$Fold01
[1] 0.1851852

$Fold02
[1] 0.2692308

$Fold03
[1] 0.8888889

$Fold04
[1] 0.3333333

$Fold05
[1] 0.2142857

$Fold06
[1] 0.3076923

$Fold07
[1] 0.2692308

$Fold08
[1] 0.8518519

$Fold09
[1] 0.8076923

$Fold10
[1] 0.3076923
```

Mean of accuracy

```r
accuracy = mean(as.numeric(cv))
accuracy
```

```
[1] 0.4435083
```

```
Variable(s) 'weekly_icu_admissions' and 'weekly_icu_admissions_per_million' and 'population' and 'population_density'
and 'median_age' and 'aged_65_older' and 'aged_70_older' and 'gdp_per_capita' and 'extreme_poverty' and
'cardiovasc_death_rate' and 'diabetes_prevalence' and 'female_smokers' and 'male_smokers' and 'handwashing_facilities'
and 'hospital_beds_per_thousand' and 'life_expectancy' and 'human_development_index' constant. Cannot scale data.
WARNING: reaching max number of iterations
```

Tune model: smaller UK only dataset

```
##tune using smaller dataset UK = 10 rows
```{r}
#tune usin smaller dataset
UK.training_set1 <- sample_n(UK.training_set, 10)
UK.test_set1 <- sample_n(UK.test_set, 10)
library(e1071)
classifier = svm(formula = Traffic_warning ~ .,
                 data = UK.training_set1,
                 type = 'C-classification',
                 kernel = 'linear', scale = TRUE)
summary(classifier)
set.seed(123)
tune_classifier <- tune(svm, Traffic_warning~., data = UK.training_set1, type = 'C-classification',
                 kernel = 'linear',
     ranges = list(epsilon = seq(0,1,0.1), cost = 2 ^(2:7)))
plot(tune_classifier)

```
```

Call:
svm(formula = Traffic_warning ~ ., data = UK.training_set1, type = "C-classification", kernel = "linear",
    scale = TRUE)

Parameters:
   SVM-Type: C-classification
 SVM-Kernel: linear
       cost: 1

Number of Support Vectors: 6

 (2 2 2)

Number of Classes: 3

Levels:
 Green Orange Red
```

```
summary(tune_classifier)
```

R Console | data.frame 1 x 2 | data.frame 66 x 4

```
Parameter tuning of 'svm':

- sampling method: leave-one-out

- best parameters:

- best performance: 0.1

- Detailed performance results:
```

```
summary(tune_classifier)
```

R Console | data.frame 1 x 2 | data.frame 66 x 4

epsilon <dbl>	cost <dbl>	error <dbl>	dispersion <dbl>
0.0	4	0.1	0.3162278
0.1	4	0.1	0.3162278
0.2	4	0.1	0.3162278
0.3	4	0.1	0.3162278
0.4	4	0.1	0.3162278
0.5	4	0.1	0.3162278
0.6	4	0.1	0.3162278
0.7	4	0.1	0.3162278
0.8	4	0.1	0.3162278
0.9	4	0.1	0.3162278

```
summary(tune_classifier)
```

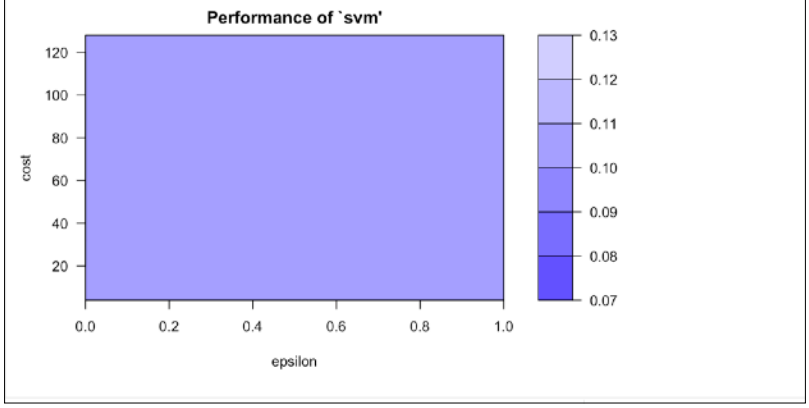
R Console | data.frame 1 x 2 | data.frame 66 x 4

epsilon <dbl>	cost <dbl>
0	4

1 row

```
summary(classifier)
set.seed(123)
tune_classifier <- tune(svm, Traffic_warning~., data = UK.training_set1, type = 'C-classification',
                 kernel = 'linear',
     ranges = list(epsilon = seq(0,1,0.1), cost = 2 ^(2:7)))
plot(tune_classifier)

```


```

choose the best model

```{r}
# choose the best model
final_svm_model <- tune_classifier$best.model
summary(final_svm_model)
```

```
Call:
best.tune(method = svm, train.x = Traffic_warning ~ ., data = UK.training_set1, ranges = list(epsilon = seq(0,
    1, 0.1), cost = 2^(2:7)), type = "C-classification", kernel = "linear")


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  linear
       cost:  4

Number of Support Vectors:  6

 ( 2 2 2 )


Number of Classes:  3

Levels:
 Green Orange Red
```

Ploting the Confusion Matrix

```{r}
cm = table(UK.test_set1[, 57], y_pred)
cm
```

```
        y_pred
         Green Orange Red
  Green      6      0   0
  Orange     0      3   1
  Red        0      0   0
```

```{r}
# Printing accuracy rate
accuracy = (cm[1,1] + cm[2,2]+ cm[3,3]) / (cm[1,1] +
cm[3,2])
accuracy
```

```
[1] 0.9
```

Applying k-Fold Cross Validation

```{r}
library(caret)
library(lattice)
library(ggplot2)
# in creating the folds we specify the target feature (dependent variable) and # of folds
folds = createFolds(UK.training_set1$Traffic_warning, k = 10)
# in cv we are going to applying a created function to our 'folds'
cv = lapply(folds, function(x) { # start of function
  # in the next two lines we will separate the Training set into it's 10 pieces
  training_fold = UK.training_set1[-x, ] # training fold =  training set minus (-) it's sub test fold
  test_fold = UK.training_set1[x, ] # here we describe the test fold individually
  # now apply (train) the classifer on the training_fold
  classifier = svm(formula = Traffic_warning ~ .,
                   data = training_fold,
                   type = 'C-classification',
                   kernel = 'linear', scale = TRUE)
  # next step in the loop, we calculate the predictions and cm and we equate the accuracy
  # note we are training on training_fold and testing its accuracy on the test_fold
  y_pred = predict(classifier, newdata = test_fold[-57])
  cm = table(test_fold[, 57], y_pred)
  accuracy = (cm[1,1] + cm[2,2]+ cm[3,3]) / (cm[1,1] + cm[2,2] + cm[3,3] + cm[1,2]+ cm[1,3]+ cm[2,1]+ cm[2,3]+ cm[3,1]+
cm[3,2])
  return(accuracy)
})
```

```{r}
# For CV we can see we have 10 folds/iterations each with slight variations of accuracy.
cv
```

```
$Fold01
[1] 1

$Fold02
[1] 1

$Fold03
[1] 1

$Fold04
[1] 1

$Fold05
[1] 1

$Fold06
[1] 1

$Fold07
[1] 0

$Fold08
[1] 1

$Fold09
[1] 1

$Fold10
[1] 1
```

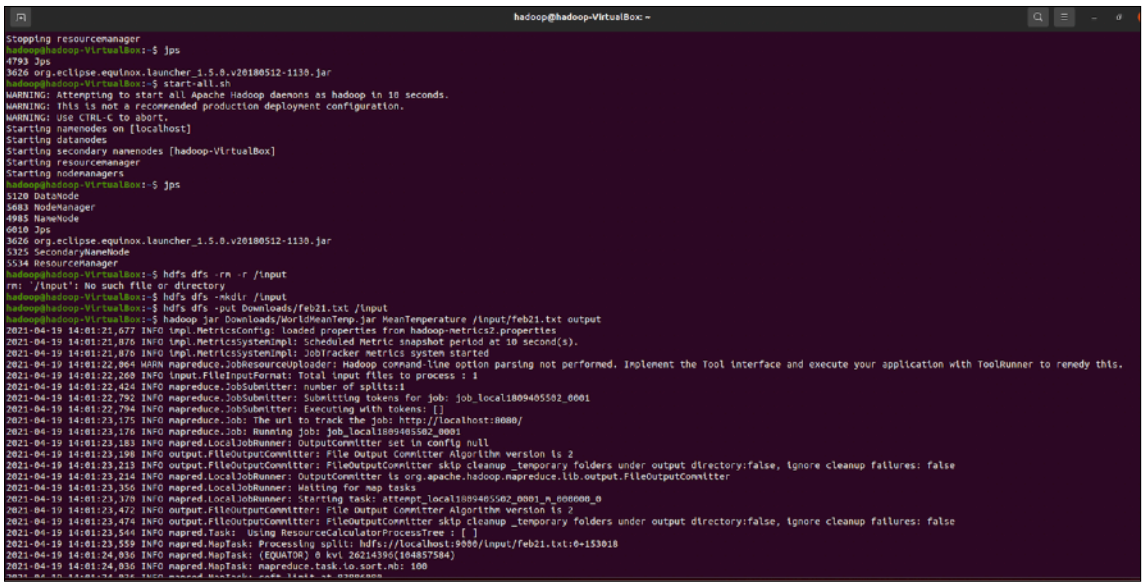
Mean of accuracy

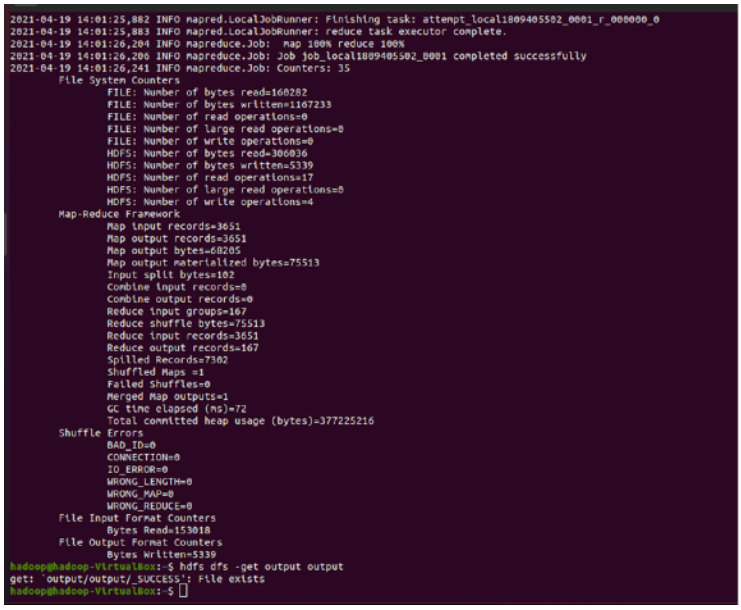
```{r}
accuracy = mean(as.numeric(cv))
accuracy
```

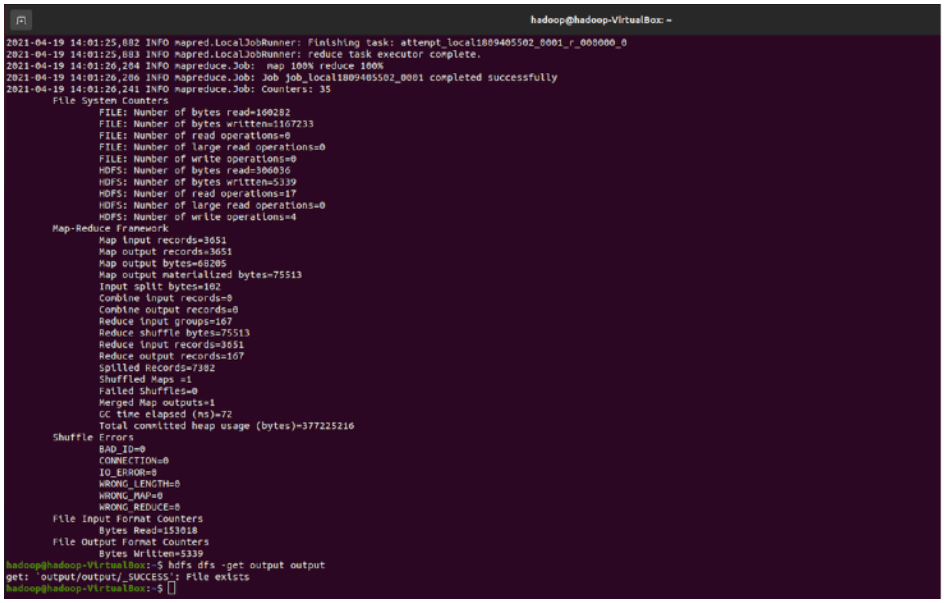
```
[1] 0.9
```

4.2 Model evaluation and testing

5. High Performance Computational implementation






hadoop@hadoop-VirtualBox: ~
2021-04-19 14:01:25,882 INFO mapred.LocalJobRunner: Finishing task: attempt_local1809405502_0001_r_000000_0
2021-04-19 14:01:25,883 INFO mapred.LocalJobRunner: reduce task executor complete.
2021-04-19 14:01:26,204 INFO mapreduce.Job: map 100% reduce 100%
2021-04-19 14:01:26,206 INFO mapreduce.Job: Job job_local1809405502_0001 completed successfully
2021-04-19 14:01:26,241 INFO mapreduce.Job: Counters: 35
File System Counters
FILE: Number of bytes read=160282
FILE: Number of bytes written=1167233
FILE: Number of read operations=0
FILE: Number of large read operations=0
FILE: Number of write operations=0
HDFS: Number of bytes read=306036
HDFS: Number of bytes written=5339
HDFS: Number of read operations=17
HDFS: Number of large read operations=0
HDFS: Number of write operations=4
Map-Reduce Framework
Map input records=3651
Map output records=3651
Map output bytes=68205
Map output materialized bytes=75513
Input split bytes=102
Combine input records=0
Combine output records=0
Reduce input groups=167
Reduce shuffle bytes=75513
Reduce input records=3651
Reduce output records=167
Spilled Records=7302
Shuffled Maps =1
Failed Shuffles=0
Merged Map outputs=1
GC time elapsed (ms)=72
Total committed heap usage (bytes)=377225216
Shuffle Errors
BAD_ID=0
CONNECTION=0
IO_ERROR=0
WRONG_LENGTH=0
WRONG_MAP=0
WRONG_REDUCE=0
File Input Format Counters
Bytes Read=153018
File Output Format Counters
Bytes Written=5339
hadoop@hadoop-VirtualBox:~$ hdfs dfs -get output output
get: `output/output/_SUCCESS': File exists
hadoop@hadoop-VirtualBox:~$

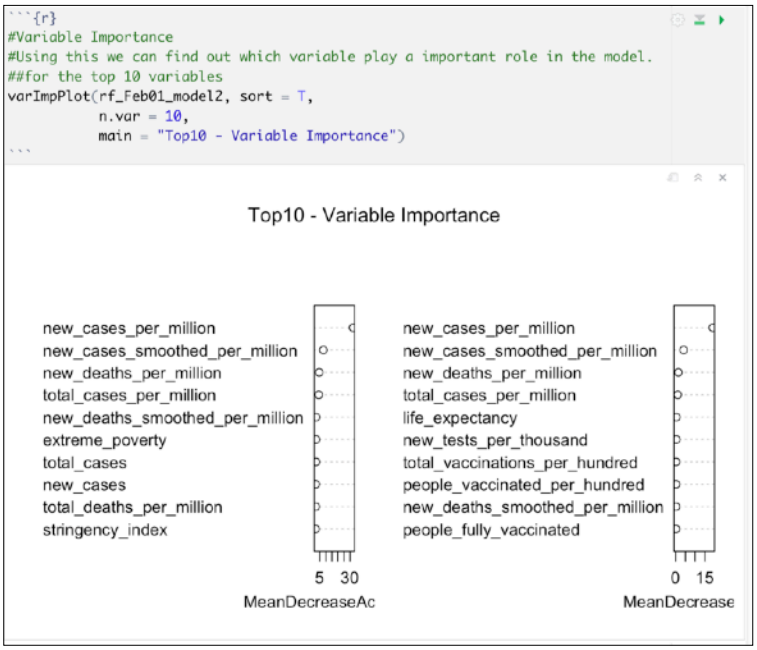
6. Performance Evaluation and comparison of methods

SVM Models	Accuracy achieved
Model 1: Radial Kernel: svm(Traffic_warning ~ all variables) K-fold cross validation	0.9882540
Model 2: Linear Kernel: svm(Traffic_warning ~ all variables) K-fold cross validation	0.9982222
Model 3: Linear Kernel: svm(Traffic_warning ~ AvgTemp) K-fold cross validation	0.9326138
Model 4: Linear Kernel: svm(Traffic_warning ~ all variables) - UK data only - chose Best model	0.9000000
Random Forest Models	**Accuracy achieved**
Model 1: Random Forest model	0.9545455
Model 2: Pruned Random Forest model	0.9848000

```{r}
#can also get the quantitative values for the important variables
importance(rf_Feb01_model2)
```

	1	2	3	MeanDecreaseAccuracy	MeanDecreaseGini
total_cases	1.40794824	0.4281313	0.00000000	1.873545113	0.08359885
new_deaths	1.00167084	0.0000000	0.00000000	1.001670845	0.00534632
new_cases_smoothed_per_million	7.41491441	2.6323921	3.24799177	7.964886312	4.11367196
reproduction_rate	0.00000000	0.0000000	0.00000000	0.000000000	0.00000000
hosp_patients_per_million	1.36432378	-1.4102149	0.00000000	1.001670845	0.00743013
weekly_hosp_admissions_per_million	0.00000000	0.0000000	0.00000000	0.000000000	0.00000000
new_tests_per_thousand	1.42001876	-0.2698035	-0.34194835	0.989055942	0.39177549
tests_per_case	1.41890410	-1.0016708	-1.00167084	0.061429898	0.07126707
new_vaccinations	-1.00167084	1.0016708	0.00000000	0.000000000	0.00470792
people_fully_vaccinated_per_hundred	0.15971967	-0.6769980	0.00000000	-0.298631313	0.17954119
population_density	-0.04095297	-0.7586635	-1.00167084	-1.121240671	0.19143159
gdp_per_capita	1.60426038	0.3371846	-1.00167084	1.517836178	0.20291604
female_smokers	0.02719142	0.2732469	0.00000000	-0.053940643	0.08731815
life_expectancy	-1.16374083	-3.0149459	2.10631307	-1.999883249	0.41309308
new_cases	1.59285298	0.7392213	-0.44736274	1.663436117	0.11957792
new_deaths_smoothed	1.00167084	0.5816460	0.63287759	1.253764744	0.07140863
total_deaths_per_million	2.17441922	1.8673853	-2.09197432	1.652856196	0.17228206
icu_patients	-1.00167084	0.0000000	0.00000000	-1.001670845	0.03361111
weekly_icu_admissions	0.00000000	0.0000000	0.00000000	0.000000000	0.00000000
new_tests	1.41877139	-1.4189513	0.00000000	0.031419263	0.06358128
new_tests_smoothed	-1.00167084	-1.3456839	-1.00167084	-1.913813655	0.08034273
total_vaccinations	0.00000000	0.0000000	0.00000000	0.000000000	0.04058859
new_vaccinations_smoothed	0.00000000	0.0000000	0.00000000	0.000000000	0.06335277
new_vaccinations_smoothed_per_million	-1.00167084	1.0016708	0.00000000	-1.001670845	0.09704291
median_age	-1.00167084	1.0016708	1.41895131	0.990277720	0.05078510
extreme_poverty	1.62226442	1.4130851	-1.00167084	1.895533703	0.07917236
male_smokers	-0.49113534	1.3804229	-0.27738566	0.626312505	0.14555354
human_development_index	-1.00167084	-1.0016708	0.00000000	-1.325851111	0.05584314
new_cases_smoothed	0.02318074	1.6626396	0.00000000		

```{r}
#Variable Importance
#Using this we can find out which variable play a important role in the model.
##for the top 10 variables
varImpPlot(rf_Feb01_model2, sort = T,
           n.var = 10,
           main = "Top10 - Variable Importance")
```



Applying k-Fold Cross Validation_Linear Kernel

```{r}
library(caret)
library(lattice)
library(ggplot2)
# in creating the folds we specify the target feature (dependent variable) and # of folds
folds = createFolds(training_set$Traffic_warning, k = 10)
# in cv we are going to applying a created function to our 'folds'
cv = lapply(folds, function(x) { # start of function
  # in the next two lines we will separate the Training set into it's 10 pieces
  training_fold = training_set[-x, ] # training fold =  training set minus (-) it's sub test fold
  test_fold = training_set[x, ] # here we describe the test fold individually
  # now apply (train) the classifer on the training_fold
  classifier = svm(formula = Traffic_warning ~ AvgTemp,
                   data = training_fold,
                   type = 'C-classification',
                   kernel = 'linear')
  # next step in the loop, we calculate the predictions and cm and we equate the accuracy
  # note we are training on training_fold and testing its accuracy on the test_fold
  y_pred = predict(classifier, newdata = test_fold[-57])
  cm = table(test_fold[, 57], y_pred)
  accuracy = (cm[1,1] + cm[2,2]+ cm[3,3]) / (cm[1,1] + cm[2,2] + cm[3,3] + cm[1,2]+ cm[1,3]+ cm[2,1]+ cm[2,3]+ cm[3,1]+ cm[3,2])
  return(accuracy)
})
```

```{r}
# For CV we can see we have 10 folds/iterations each with slight variations of accuracy.
cv
```

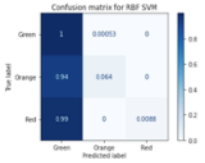
```
$Fold01
[1] 0.9325011

$Fold02
[1] 0.9328959

$Fold03
[1] 0.9324868

$Fold04
[1] 0.9326984

$Fold05
[1] 0.9326984

$Fold06
[1] 0.9324868

$Fold07
[1] 0.9324868

$Fold08
[1] 0.9326984

$Fold09
[1] 0.9325011

$Fold10
[1] 0.9326842
```

Mean of accuracy

```{r}
accuracy = mean(as.numeric(cv))
accuracy
```

```
[1] 0.9326138
```

```
# Choosing the most suitable parameters

# Fit a support vector machine by using Scikit-Learn's support vector classifier to train an SVM model on the data

from sklearn.svm import SVC # "Support vector classifier"
from sklearn.gaussian_process import GaussianProcessClassifier
from sklearn.gaussian_process.kernels import RBF
from sklearn.metrics import plot_confusion_matrix, accuracy_score
from sklearn.datasets import make_gaussian_quantiles
from mpl_toolkits.mplot3d import Axes3D

model = SVC(kernel='rbf', C=50, gamma = 10, random_state=42)
svm = model.fit(SXtrain, Sytrain)

# Evaluate by means of a confusion matrix
matrix = plot_confusion_matrix(svm, Xvalid, vtest,
                               cmap=plt.cm.Blues,
                               normalize='true')
plt.title('Confusion matrix for RBF SVM')
plt.show(matrix)
plt.show()
```




```
# Generate predictions
y_pred = svm.predict(Xvalid)

# Evaluate by means of accuracy
from sklearn import metrics
accuracy = metrics.accuracy_score(vtest, y_pred)
print(f'Model accuracy: {accuracy}')

from sklearn.metrics import classification_report
print(classification_report(vtest, y_pred))
```

```
Model accuracy: 0.9356613756613756
              precision    recall  f1-score   support

       Green       0.94      1.00      0.97     13228
      Orange       0.85      0.06      0.12       607
         Red       1.00      0.01      0.02       340

    accuracy                           0.94     14175
   macro avg       0.93      0.36      0.37     14175
weighted avg       0.93      0.94      0.91     14175
```


Applying k-Fold Cross Validation_Linear Kernel

```{r}
library(caret)
library(lattice)
library(ggplot2)
# in creating the folds we specify the target feature (dependent variable) and # of folds
folds = createFolds(training_set$Traffic_warning, k = 10)
# in cv we are going to applying a created function to our 'folds'
cv = lapply(folds, function(x) { # start of function
  # in the next two lines we will separate the Training set into it's 10 pieces
  training_fold = training_set[-x, ] # training fold =  training set minus (-) it's sub test fold
  test_fold = training_set[x, ] # here we describe the test fold individually
  # now apply (train) the classifer on the training_fold
  classifier = svm(formula = Traffic_warning ~ AvgTemp,
                   data = training_fold,
                   type = 'C-classification',
                   kernel = 'linear')
  # next step in the loop, we calculate the predictions and cm and we equate the accuracy
  # note we are training on training_fold and testing its accuracy on the test_fold
  y_pred = predict(classifier, newdata = test_fold[-57])
  cm = table(test_fold[, 57], y_pred)
  accuracy = (cm[1,1] + cm[2,2]+ cm[3,3]) / (cm[1,1] + cm[2,2] + cm[3,3] + cm[1,2]+ cm[1,3]+ cm[2,1]+ cm[2,3]+ cm[3,1]+ cm[3,2])
  return(accuracy)
})
```

```{r}
# For CV we can see we have 10 folds/iterations each with slight variations of accuracy.
cv
```

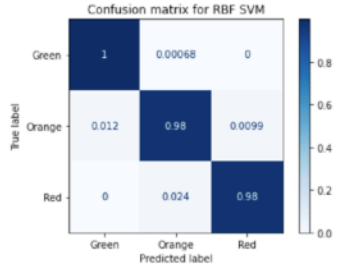
```
$Fold01
[1] 0.9325011

$Fold02
[1] 0.9328959

$Fold03
[1] 0.9324868

$Fold04
[1] 0.9326984

$Fold05
[1] 0.9326984

$Fold06
[1] 0.9324868

$Fold07
[1] 0.9324868

$Fold08
[1] 0.9326984

$Fold09
[1] 0.9325011

$Fold10
[1] 0.9326842
```

Mean of accuracy

```{r}
accuracy = mean(as.numeric(cv))
accuracy
```

```
[1] 0.9982222
```


```
model2 = SVC(kernel='linear', C=1, gamma = 4, random_state=42)
svm = model2.fit(SXtrain, Sytrain)

# Evaluate by means of a confusion matrix
matrix = plot_confusion_matrix(svm, Xvalid, vtest,
                               cmap=plt.cm.Blues,
                               normalize='true')
plt.title('Confusion matrix for linear SVM')
plt.show(matrix)
plt.show()

# Generate predictions
y_pred = svm.predict(Xvalid)

# Evaluate by means of accuracy
accuracy = accuracy_score(vtest, y_pred)
print(f'Model2 accuracy: {accuracy}')
print(classification_report(vtest, y_pred))
```




```
Model2 accuracy: 0.9978835978835979
              precision    recall  f1-score   support

       Green       1.00      1.00      1.00     13228
      Orange       0.97      0.98      0.98       607
         Red       0.98      0.98      0.98       340

    accuracy                           1.00     14175
   macro avg       0.98      0.98      0.98     14175
weighted avg       1.00      1.00      1.00     14175
```