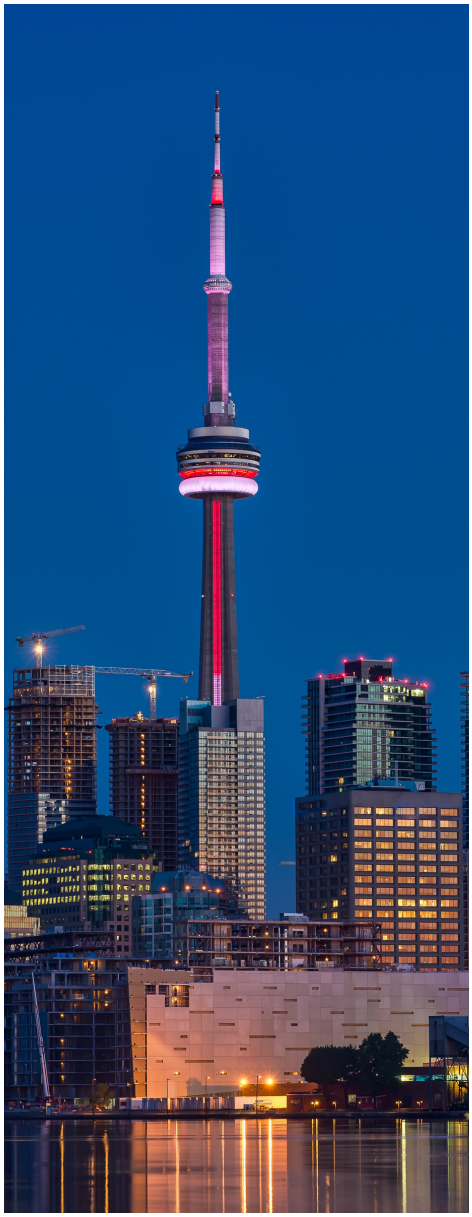
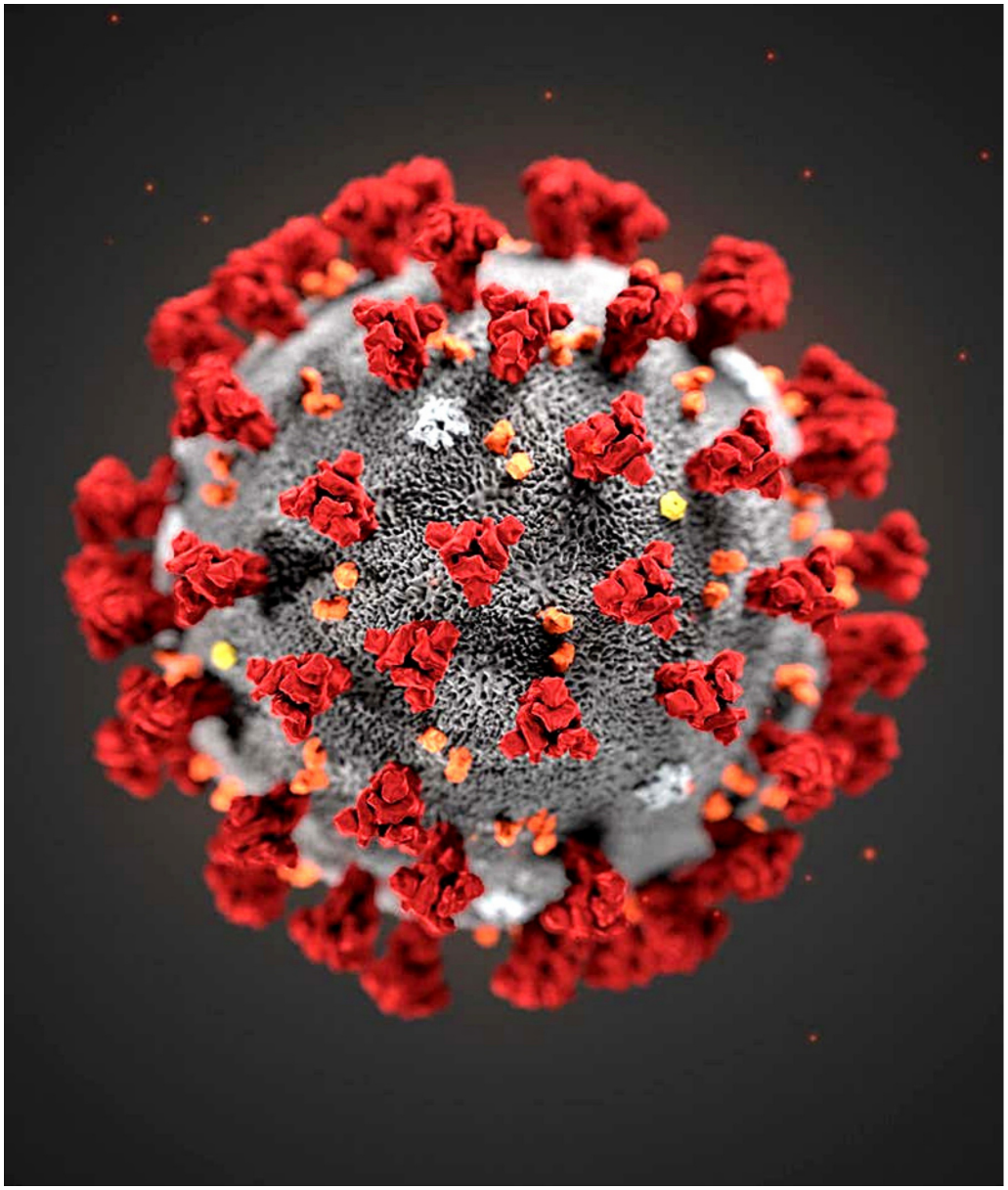
COVID-19 Mortality Rate Report

Predictive Model using SparkML





Kishan Kanaiyalal Patel
(8781642)

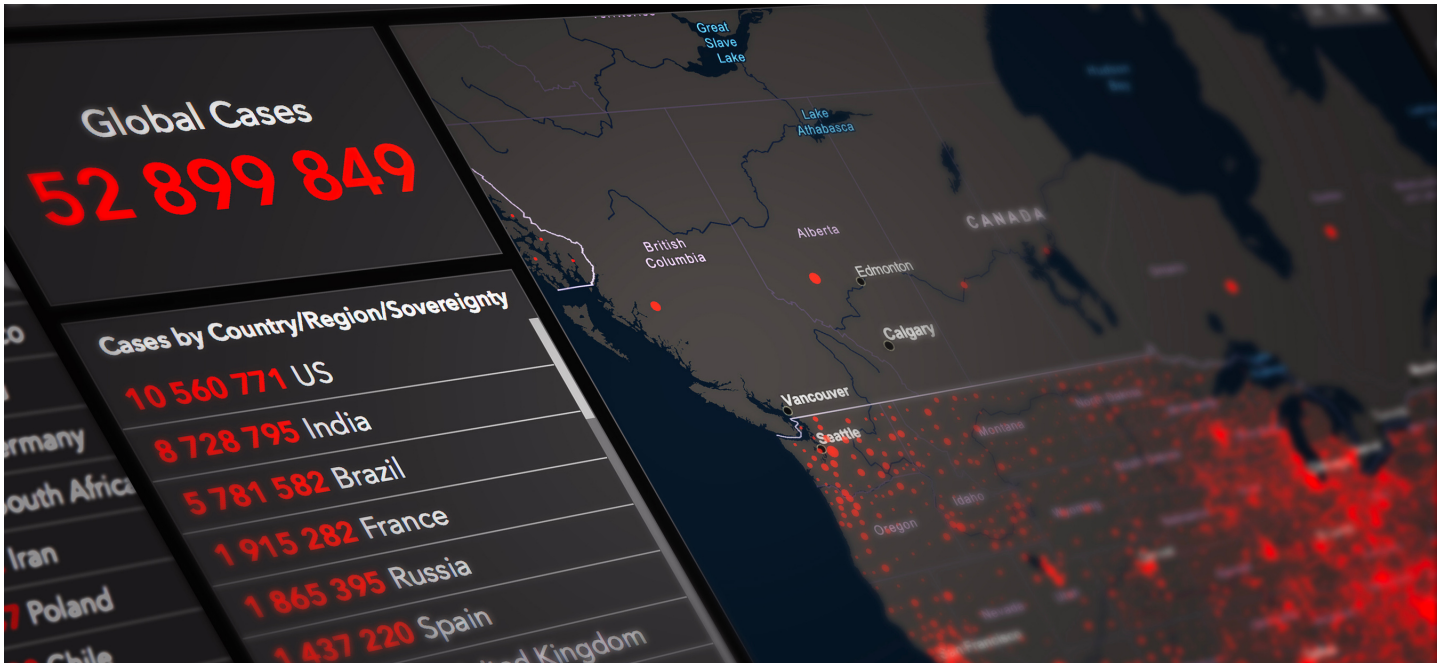


Table of Contents

Abstract

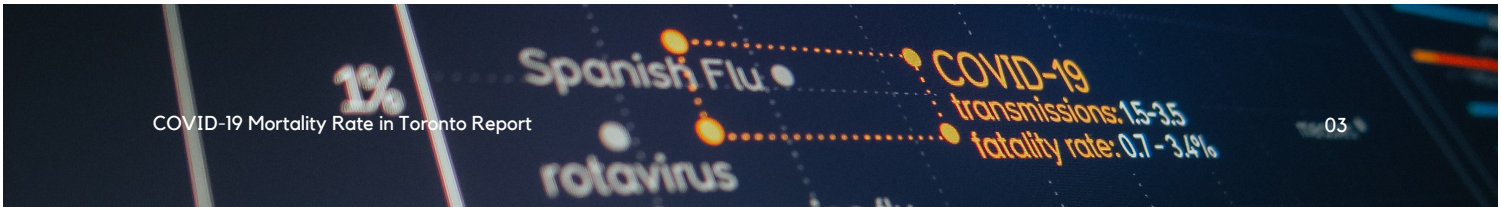
Coronavirus (COVID-19) is a worldwide pandemic that has never been seen before. It is critical to accurately anticipate the mortality risk among infected people in order to prioritize medical care and mitigate the burden on the healthcare system. The purpose of this report is to see how accurate machine learning algorithms are in predicting COVID-19 mortality using Apache Spark Machine Learning Algorithms.

Introduction

Coronaviruses are a broad family of viruses that may cause everything from the common cold to more serious illnesses like Middle East Respiratory Syndrome (MERS) and Severe Acute Respiratory Syndrome (SARS). In 2019, a new coronavirus (COVID-19) was discovered in Wuhan, China.

It has had a terrible impact on the globe's demography, resulting in the deaths of more than 5.3 million people throughout the world. Since the 1918 influenza pandemic, it has emerged as the most significant worldwide health concern.

Toronto Public Health contributed the information needed to create this prediction model, which largely updates the records retrieved from the provincial Case & Contact Management System (CCM) on a weekly basis. On April 20, 2022, the dataset was last updated.



About the Dataset

Since the first case was reported in January 2020, this data collection includes demographic, geographic, and severity data for all confirmed and probable cases reported to and handled by Toronto Public Health.

This dataset consists of 18 columns and 3,13,961 rows, meaning 3,13,961 number of positive cases have been reported in Toronto from January 2020 to April 20, 2022, considering the dataset is uncleaned.

The feature of columns can be depicted in following way.

- **_id:** Unique row identifier for Open Data database
- **Assigned_ID:** A unique ID provided by Toronto Public Health to cases for the purpose of uploading to Open Data and allowing for case monitoring.
- **Outbreak Associated:** COVID-19 outbreaks in Toronto healthcare facilities and settings, as well as other Toronto congregate settings, have been linked to outbreak-associated cases.
- **Age Group:** Age at time of illness.
- **Neighbourhood Name:** Toronto's 140 geographically distinct neighbourhood.
- **FSA:** First three characters of Postal Code
- **Source of Infection:** Infection spread through close contact with case, Outbreaks, Household contact, Travel, Community and other.
- **Classification:** Category of cases as confirmed, and probable
- **Episode Date:** Derived variable that best indicates when the disease was acquired, and it relates to the earliest accessible date from the following: symptom onset, laboratory specimen collection date, or reported date.
- **Reported Date:** The date on which Toronto Public Health was notified of the case.
- **Client Gender:** Biological sex of patient.
- **Outcome:** Result of diagnosis - Fatal, Resolved, Active.
- **Currently Hospitalized:** Cases that are currently admitted to the hospital.
- **Currently in ICU:** Cases that are currently admitted to the ICU.

- **Currently Intubated:** Cases are currently intubated related to their COVID-19 infection.
- **Ever Hospitalized:** Cases that were admitted to the hospital.
- **Ever in ICU:** Cases that were admitted to the ICU.
- **Ever Intubated:** Cases that were intubated related to their COVID-19 infection.

Methodology

First, acknowledging the dataset being unbalanced, it is downloaded from Open Toronto website in CSV format and then loaded in Hadoop Distributed File System. Then basic operations like eliminating null values and some primary feature engineering is being performed to balance the dataset. Undersampling algorithms can be used to perform balancing onto the dataset. The prediction of mortality rate can be forecasted by implementing various supervised machine learning algorithms like decision trees, Random Forest, and Support Vector Machines. In order to acquire meaningful outcomes via data visualization, Tableau is used.

Objective

Data analysis and machine learning play crucial role in health sector by detecting frauds, helping increase the engagement with patients and predicting some life-saving information. The objective of this report is to build the predictive model which helps predict the accuracy of mortality ratio using big data technology like Apache SparkML. After this research, an individual can gain some insights about mortality ratio caused by COVID-19 starting from its eruption to the current date of Toronto region. How different attributes of this dataset can contribute in developing this model is further described in forthcoming paragraphs.

Feature Engineering & Data Visualization

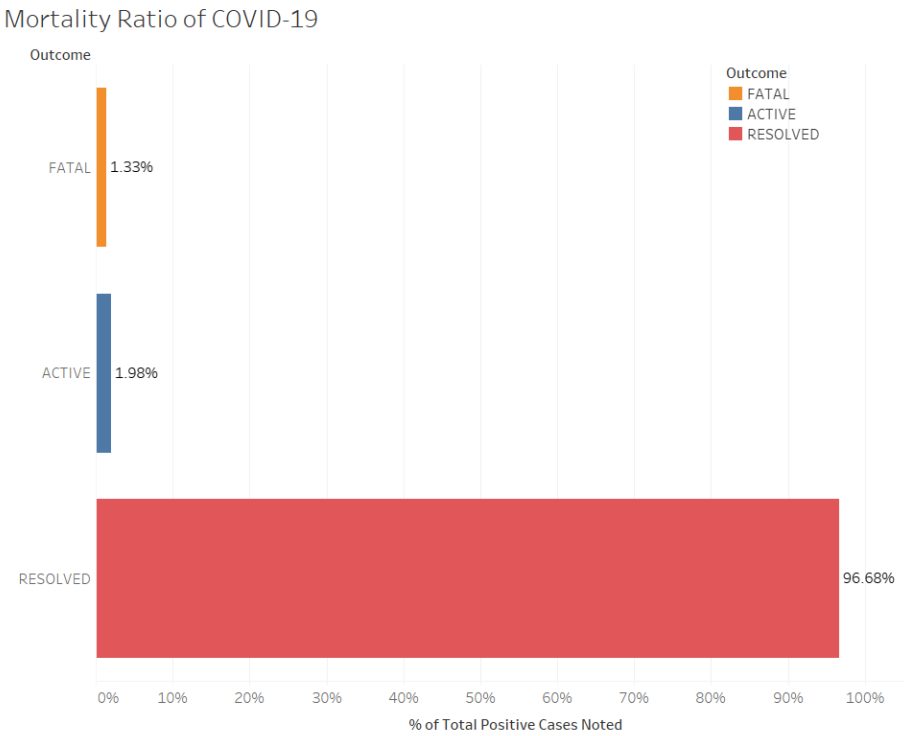
Feature Engineering is the most crucial part for building the machine learning model. The first step of performing feature engineering on this dataset is to remove null values. Then every attributes are not required for the prediction, therefore below-mentioned attributes are selected and renamed where needed.

- **Age Group:** Age Group
- **Client Gender:** Sex
- **Currently in ICU:** Currently in ICU
- **Ever Hospitalized:** Ever Hospitalized
- **Ever in ICU:** Ever in ICU
- **Outcome:** Result

To better understand the data, different visualization modes are performed.

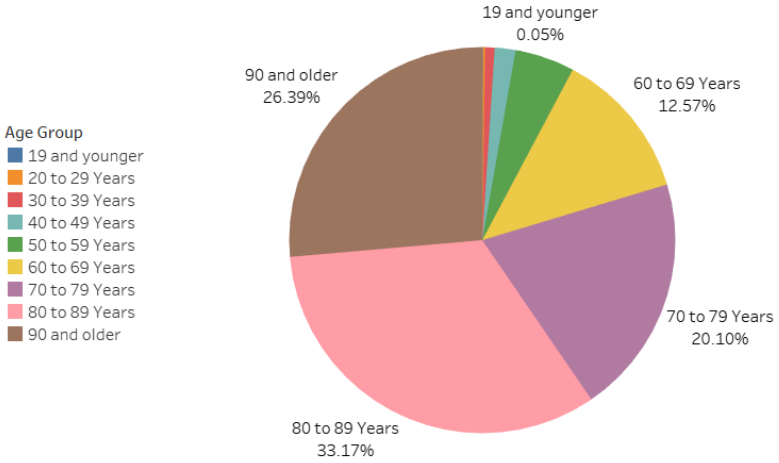
1. **Mortality Ratio of COVID-19 in Toronto**

Total mortality ratio in Toronto caused by COVID-19 starting from January 2020 to April 2022 is 1.33%. This ratio will help acquiring the insights about the prediction accuracy.



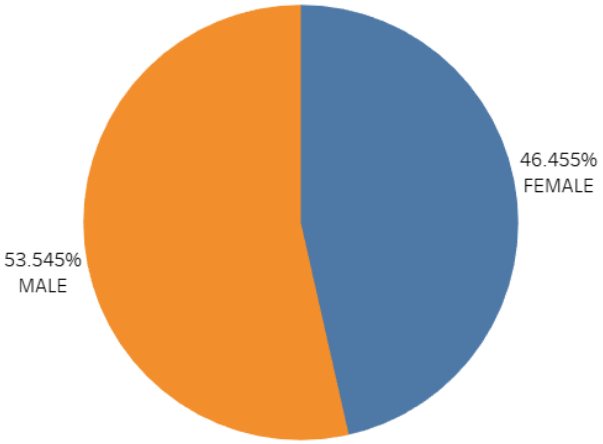
2. Mortality Ratio by Age-Group

The significant fatality is seen in the age group of 80 to 89 years' people. And the least fatality can be seen for the age group under 19 years. These results are because of the low immune system of older people and higher immune system of younger ones.



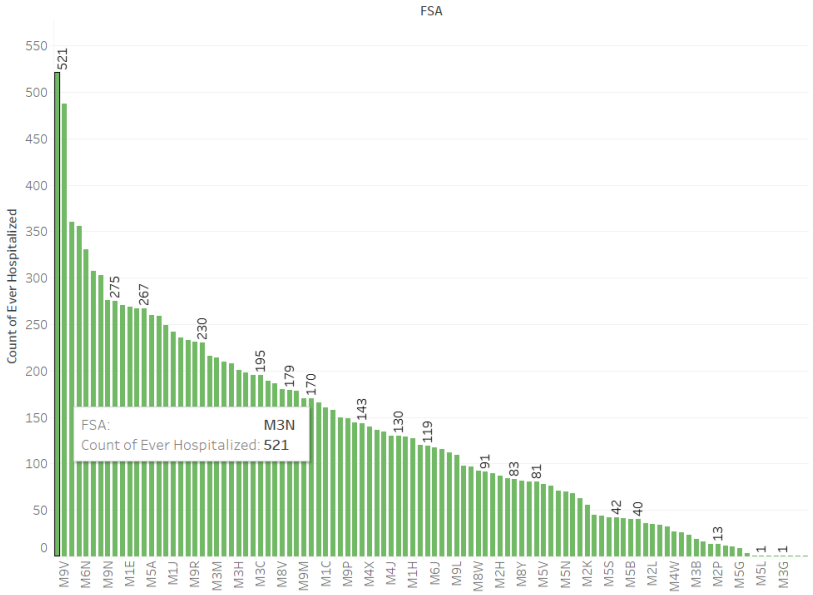
3. Mortality Ratio by Gender

The immune system may differ for males and females so does the effect of COVID-19 on them. The results depicts that females survive more than males.



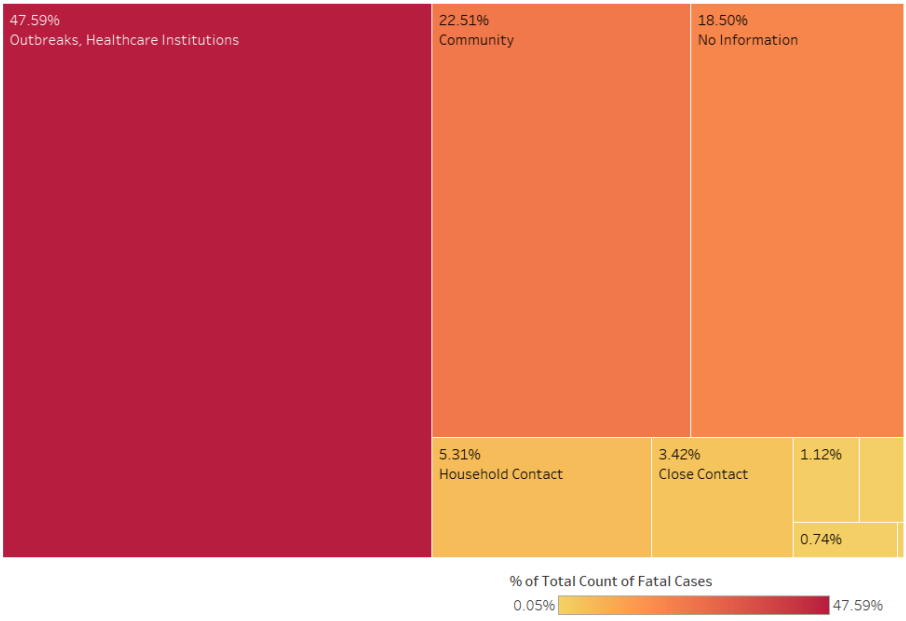
4. Number of people hospitalized per region

COVID-19 is the disease which spreads by being in the contact with infected people. The results may depend upon the population density of particular region. Higher the density, higher the chances of infection.



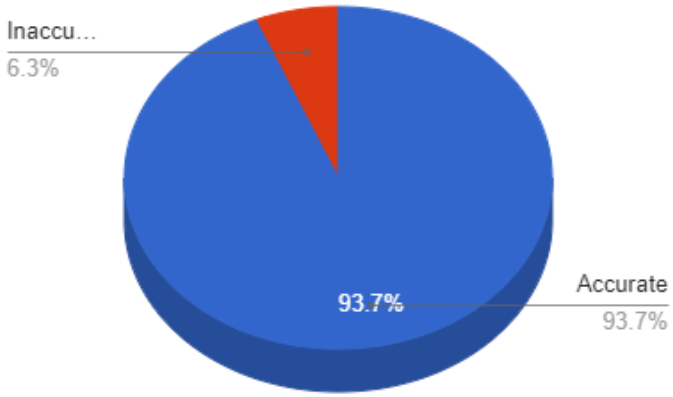
5. Mortality Ratio by Source of Infection

The Source of Infection may affect the predictive model, healthcare institution being the significant contributing factor.



Accuracy of Predictive Model

Based on the features of the dataset, the predictive model can be derived that is 93.68% accurate using Random Forest Algorithm, rest 6.32% is not accurately predicted.

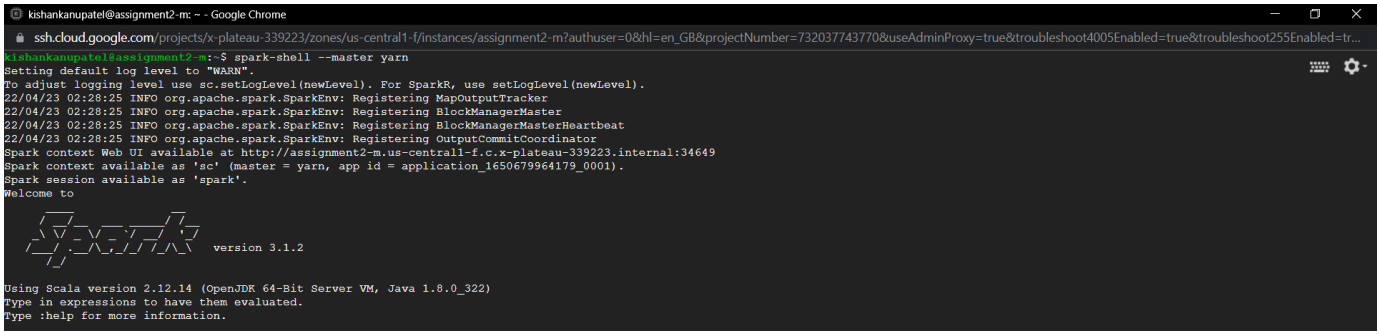


Conclusion

This report is a brief overview of how a machine learning model can be implemented to predict the mortality. Since, it gives almost 94% of accuracy in the prediction, the model can be used to make long-term predictions as well. However, an alternative of this Random Forest algorithm could be Support Machine Vector algorithm.

Appendix

```
spark-shell --master yarn
```



- Importing libraries and the dataset from hadoop directory. Eliminating null values from the dataset.

```
import org.apache.spark.sql.functions._
import org.apache.spark.sql.expressions.Window
import org.apache.spark.ml.feature.OneHotEncoder
import org.apache.spark.ml.feature.{VectorAssembler, StringIndexer}
import org.apache.spark.ml.Pipeline
import org.apache.spark.ml.classification.{RandomForestClassificationModel, RandomForestClassifier}
import org.apache.spark.ml.tuning.{CrossValidator, CrossValidatorModel, ParamGridBuilder}
import org.apache.spark.ml.evaluation.{MulticlassClassificationEvaluator}
import org.apache.spark.ml.param.ParamMap
import org.apache.spark.sql.types.{IntegerType, DoubleType}

val dataset=spark.read
 .format("csv")
 .option("header","true")
 .load("hdfs://10.128.0.24:8020/BigData/CovidData/Toronto_COVID19.csv")

val CleanedData= dataset.na.drop()
```

```
scala> :paste
// Entering paste mode (ctrl-D to finish)

import org.apache.spark.sql.functions._
import org.apache.spark.sql.expressions.Window
import org.apache.spark.ml.feature.{VectorAssembler, StringIndexer}
import org.apache.spark.ml.Pipeline
import org.apache.spark.ml.classification.{RandomForestClassificationModel, RandomForestClassifier}
import org.apache.spark.ml.tuning.{CrossValidator, CrossValidatorModel, ParamGridBuilder}
import org.apache.spark.ml.evaluation.{MulticlassClassificationEvaluator}
import org.apache.spark.ml.param.ParamMap
import org.apache.spark.sql.types.{IntegerType, DoubleType}


val dataset=spark.read
 .format("csv")
 .option("header","true")
 .load("hdfs://10.128.0.24:8020/BigData/CovidData/Toronto_COVID19.csv")


// Exiting paste mode, now interpreting.

import org.apache.spark.sql.functions._
import org.apache.spark.sql.expressions.Window
import org.apache.spark.ml.feature.{VectorAssembler, StringIndexer}
import org.apache.spark.ml.Pipeline
import org.apache.spark.ml.classification.{RandomForestClassificationModel, RandomForestClassifier}
import org.apache.spark.ml.tuning.{CrossValidator, CrossValidatorModel, ParamGridBuilder}
import org.apache.spark.ml.evaluation.MulticlassClassificationEvaluator
import org.apache.spark.ml.param.ParamMap
import org.apache.spark.sql.types.{IntegerType, DoubleType}
dataset: org.apache.spark.sql.DataFrame = [_id: string, Assigned_ID: string ... 16 more fields]

scala> val CleanedData= dataset.na.drop()
CleanedData: org.apache.spark.sql.DataFrame = [_id: string, Assigned_ID: string ... 16 more fields]
```

- Previewing the Dataset

```
CleanedData.show(20)
```

```
scala> CleanedData.show(20)
+---+----------+------------------+--------------+--------------------+---+--------------------+--------------+-----------+------------+------------+--------+----------------------+----------
------+------------------+----------------+-----------+-------------+
|_id|Assigned_ID|Outbreak Associated|     Age Group|  Neighbourhood Name|FSA| Source of Infection|Classification|Episode Date|Reported Date|Client Gender| Outcome|Currently Hospitalized|Currently
in ICU|Currently Intubated|Ever Hospitalized|Ever in ICU|Ever Intubated|
+---+----------+------------------+--------------+--------------------+---+--------------------+--------------+-----------+------------+------------+--------+----------------------+----------
------+------------------+----------------+-----------+-------------+
|  1|         1|           Sporadic|50 to 59 Years|     Willowdale East|M2N|              Travel|     CONFIRMED|  2020-01-22|  2020-01-23|      FEMALE|RESOLVED|                    No|
    No|                No|              No|         No|           No|
|  2|         2|           Sporadic|50 to 59 Years|     Willowdale East|M2N|              Travel|     CONFIRMED|  2020-01-21|  2020-01-23|        MALE|RESOLVED|                    No|
    No|                No|             Yes|         No|           No|
|  3|         3|           Sporadic|20 to 29 Years|   Parkwoods-Donalda|M3A|              Travel|     CONFIRMED|  2020-02-05|  2020-02-21|      FEMALE|RESOLVED|                    No|
    No|                No|              No|         No|           No|
|  4|         4|           Sporadic|60 to 69 Years|Church-Yonge Corr...|M4W|              Travel|     CONFIRMED|  2020-02-16|  2020-02-25|      FEMALE|RESOLVED|                    No|
    No|                No|              No|         No|           No|
|  5|         5|           Sporadic|60 to 69 Years|Church-Yonge Corr...|M4W|              Travel|     CONFIRMED|  2020-02-20|  2020-02-26|        MALE|RESOLVED|                    No|
    No|                No|              No|         No|           No|
|  6|         6|           Sporadic|50 to 59 Years|    Newtonbrook West|M2R|              Travel|     CONFIRMED|  2020-02-24|  2020-02-27|        MALE|RESOLVED|                    No|
    No|                No|              No|         No|           No|
|  7|         7|           Sporadic|80 to 89 Years|            Milliken|M1V|              Travel|     CONFIRMED|  2020-02-20|  2020-02-28|        MALE|RESOLVED|                    No|
    No|                No|              No|         No|           No|
|  8|         8|           Sporadic|60 to 69 Years|     Willowdale West|M2N|              Travel|     CONFIRMED|  2020-02-21|  2020-03-04|        MALE|RESOLVED|                    No|
    No|                No|             Yes|         No|           No|
|  9|         9|           Sporadic|50 to 59 Years|     Willowdale East|M2N|              Travel|     CONFIRMED|  2020-02-29|  2020-02-29|        MALE|RESOLVED|                    No|
    No|                No|              No|         No|           No|
| 10|        10|           Sporadic|60 to 69 Years|         Henry Farm|M2J|              Travel|     CONFIRMED|  2020-02-26|  2020-03-01|        MALE|RESOLVED|                    No|
    No|                No|              No|         No|           No|
| 11|        11|           Sporadic|70 to 79 Years|  Don Valley Village|M2J|              Travel|     CONFIRMED|  2020-02-14|  2020-03-01|      FEMALE|RESOLVED|                    No|
    No|                No|              No|         No|           No|
| 12|        12|           Sporadic|50 to 59 Years| Lawrence Park South|M4R|              Travel|      PROBABLE|  2020-03-01|  2020-03-02|        MALE|RESOLVED|                    No|
    No|                No|              No|         No|           No|
| 13|        13|           Sporadic|60 to 69 Years|Bridle Path-Sunny...|M2L|              Travel|     CONFIRMED|  2020-03-02|  2020-03-03|        MALE|RESOLVED|                    No|
    No|                No|              No|         No|           No|
| 14|        14|           Sporadic|30 to 39 Years|           Moss Park|M5A|           Community|      PROBABLE|  2020-03-03|  2020-03-04|        MALE|RESOLVED|                    No|
    No|                No|              No|         No|           No|
| 15|        15|           Sporadic|40 to 49 Years|              Annex|M6G|              Travel|     CONFIRMED|  2020-03-02|  2020-03-05|        MALE|RESOLVED|                    No|
    No|                No|              No|         No|           No|
| 16|        16|           Sporadic|50 to 59 Years|     Willowdale East|M2N|              Travel|     CONFIRMED|  2020-03-03|  2020-03-05|        MALE|RESOLVED|                    No|
    No|                No|              No|         No|           No|
| 17|        18|           Sporadic|40 to 49 Years|  Leaside-Bennington|M4G|              Travel|     CONFIRMED|  2020-03-04|  2020-03-07|      FEMALE|RESOLVED|                    No|
    No|                No|              No|         No|           No|
| 18|        19|Outbreak Associated|40 to 49 Years|           Moss Park|M5A|Outbreaks, Congre...|     CONFIRMED|  2020-04-14|  2020-03-06|        MALE|RESOLVED|                    No|
    No|                No|             Yes|         No|           No|
| 19|        20|           Sporadic|60 to 69 Years|St.Andrew-Windfields|M2P|              Travel|     CONFIRMED|  2020-03-05|  2020-03-07|        MALE|RESOLVED|                    No|
    No|                No|              No|         No|           No|
| 20|        21|           Sporadic|80 to 89 Years|     Willowdale East|M2N|              Travel|     CONFIRMED|  2020-03-03|  2020-03-08|        MALE|RESOLVED|                    No|
    No|                No|              No|         No|           No|
+---+----------+------------------+--------------+--------------------+---+--------------------+--------------+-----------+------------+------------+--------+----------------------+----------
------+------------------+----------------+-----------+-------------+
```

- Previewing the schema of dataset

```
CleanedData.printSchema()
```

```
scala> CleanedData.printSchema()
root
 |-- _id: string (nullable = true)
 |-- Assigned_ID: string (nullable = true)
 |-- Outbreak Associated: string (nullable = true)
 |-- Age Group: string (nullable = true)
 |-- Neighbourhood Name: string (nullable = true)
 |-- FSA: string (nullable = true)
 |-- Source of Infection: string (nullable = true)
 |-- Classification: string (nullable = true)
 |-- Episode Date: string (nullable = true)
 |-- Reported Date: string (nullable = true)
 |-- Client Gender: string (nullable = true)
 |-- Outcome: string (nullable = true)
 |-- Currently Hospitalized: string (nullable = true)
 |-- Currently in ICU: string (nullable = true)
 |-- Currently Intubated: string (nullable = true)
 |-- Ever Hospitalized: string (nullable = true)
 |-- Ever in ICU: string (nullable = true)
 |-- Ever Intubated: string (nullable = true)
```

- Selecting the required attributes as features to build the predictive model

```
val SelectedData= CleanedData.select(col("Age Group"),
 col("Client Gender").alias("Sex"),
 col("Currently in ICU"),
 col("Ever Hospitalized"),
 col("Ever in ICU"),
 col("Outcome").alias("Result"))
 .filter(CleanedData("Classification")==="CONFIRMED")

SelectedData.show(20)
```

```
scala> :paste
// Entering paste mode (ctrl-D to finish)

val SelectedData= CleanedData.select(col("Age Group"),
 col("Client Gender").alias("Sex"),
 col("Currently in ICU"),
 col("Ever Hospitalized"),
 col("Ever in ICU"),
 col("Outcome").alias("Result"))
 .filter(CleanedData("Classification")==="CONFIRMED")

// Exiting paste mode, now interpreting.

SelectedData: org.apache.spark.sql.Dataset[org.apache.spark.sql.Row] = [Age Group: string, Sex: string ... 4 more fields]

scala> SelectedData.show(20)
+--------------+------+----------------+-----------------+-----------+--------+
|     Age Group|   Sex|Currently in ICU|Ever Hospitalized|Ever in ICU|  Result|
+--------------+------+----------------+-----------------+-----------+--------+
|50 to 59 Years|FEMALE|              No|               No|         No|RESOLVED|
|50 to 59 Years|  MALE|              No|              Yes|         No|RESOLVED|
|20 to 29 Years|FEMALE|              No|               No|         No|RESOLVED|
|60 to 69 Years|FEMALE|              No|               No|         No|RESOLVED|
|60 to 69 Years|  MALE|              No|               No|         No|RESOLVED|
|50 to 59 Years|  MALE|              No|               No|         No|RESOLVED|
|80 to 89 Years|  MALE|              No|               No|         No|RESOLVED|
|60 to 69 Years|  MALE|              No|              Yes|         No|RESOLVED|
|50 to 59 Years|  MALE|              No|               No|         No|RESOLVED|
|60 to 69 Years|  MALE|              No|               No|         No|RESOLVED|
|70 to 79 Years|FEMALE|              No|               No|         No|RESOLVED|
|60 to 69 Years|  MALE|              No|               No|         No|RESOLVED|
|40 to 49 Years|  MALE|              No|               No|         No|RESOLVED|
|50 to 59 Years|  MALE|              No|               No|         No|RESOLVED|
|40 to 49 Years|FEMALE|              No|               No|         No|RESOLVED|
|40 to 49 Years|  MALE|              No|              Yes|         No|RESOLVED|
|60 to 69 Years|  MALE|              No|               No|         No|RESOLVED|
|80 to 89 Years|  MALE|              No|               No|         No|RESOLVED|
|70 to 79 Years|FEMALE|              No|               No|         No|RESOLVED|
|60 to 69 Years|FEMALE|              No|              Yes|        Yes|RESOLVED|
+--------------+------+----------------+-----------------+-----------+--------+
only showing top 20 rows
```

- Balancing the dataset using Random Undersampling method

```
val FatalCases= SelectedData.filter(SelectedData("Result")==="FATAL")
val ResolvedCases= SelectedData.filter(SelectedData("Result")==="RESOLVED")

val FatalRatio=FatalCases.count().toDouble/SelectedData.count().toDouble
val ResolvedSampleRatio= ResolvedCases.sample(false,FatalRatio)
val BalancedData= FatalCases.unionAll(ResolvedSampleRatio)

BalancedData.show(20)
```

```
scala> :paste
// Entering paste mode (ctrl-D to finish)

val FatalCases= SelectedData.filter(SelectedData("Result")==="FATAL")
val ResolvedCases= SelectedData.filter(SelectedData("Result")==="RESOLVED")

val FatalRatio=FatalCases.count().toDouble/SelectedData.count().toDouble
val ResolvedSampleRatio= ResolvedCases.sample(false,FatalRatio)
val BalancedData= FatalCases.unionAll(ResolvedSampleRatio)


// Exiting paste mode, now interpreting.

FatalCases: org.apache.spark.sql.Dataset[org.apache.spark.sql.Row] = [Age Group: string, Sex: string ... 4 more fields]
ResolvedCases: org.apache.spark.sql.Dataset[org.apache.spark.sql.Row] = [Age Group: string, Sex: string ... 4 more fields]
FatalRatio: Double = 0.013837129404915431
ResolvedSampleRatio: org.apache.spark.sql.Dataset[org.apache.spark.sql.Row] = [Age Group: string, Sex: string ... 4 more fields]
BalancedData: org.apache.spark.sql.Dataset[org.apache.spark.sql.Row] = [Age Group: string, Sex: string ... 4 more fields]

scala> BalancedData.show(20)
+--------------+------+---------------+-----------------+-----------+------+
|     Age Group|   Sex|Currently in ICU|Ever Hospitalized|Ever in ICU|Result|
+--------------+------+---------------+-----------------+-----------+------+
|70 to 79 Years|  MALE|             No|              Yes|        Yes| FATAL|
|60 to 69 Years|  MALE|             No|              Yes|        Yes| FATAL|
|  90 and older|  MALE|             No|               No|         No| FATAL|
|  90 and older|  MALE|             No|               No|         No| FATAL|
|70 to 79 Years|  MALE|             No|               No|         No| FATAL|
|  90 and older|  MALE|             No|              Yes|         No| FATAL|
|  90 and older|FEMALE|             No|               No|         No| FATAL|
|  90 and older|FEMALE|             No|               No|         No| FATAL|
|70 to 79 Years|FEMALE|             No|               No|         No| FATAL|
|40 to 49 Years|  MALE|             No|              Yes|        Yes| FATAL|
|80 to 89 Years|  MALE|             No|               No|         No| FATAL|
|60 to 69 Years|FEMALE|             No|              Yes|         No| FATAL|
|80 to 89 Years|  MALE|             No|              Yes|         No| FATAL|
|80 to 89 Years|FEMALE|             No|              Yes|         No| FATAL|
|60 to 69 Years|  MALE|             No|              Yes|        Yes| FATAL|
|50 to 59 Years|FEMALE|             No|              Yes|        Yes| FATAL|
|70 to 79 Years|  MALE|             No|              Yes|        Yes| FATAL|
|80 to 89 Years|  MALE|             No|              Yes|         No| FATAL|
|70 to 79 Years|  MALE|             No|              Yes|        Yes| FATAL|
|40 to 49 Years|  MALE|             No|              Yes|        Yes| FATAL|
+--------------+------+---------------+-----------------+-----------+------+
only showing top 20 rows
```

- Indexing the details of every column

```
val inputIndex= Array("Age Group", "Sex", "Currently in ICU", "Ever Hospitalized", "Ever in ICU")
val outputIndex= Array("Index_AgeGroup","Index_Sex","Index_CurrentICU","Index_EverHospital",
"Index_EverICU")

val StringToIndex = new StringIndexer()
StringToIndex.setInputCols(inputIndex)
StringToIndex.setOutputCols(outputIndex)

val StringToIndex2= new StringIndexer()
.setInputCol("Result")
.setOutputCol("Index_Result")

val IndexedData= StringToIndex.fit(BalancedData).transform(BalancedData)
val IndexedData_2= StringToIndex2.fit(IndexedData).transform(IndexedData)

val rankData= IndexedData_2.select(col("Index_Result").cast(IntegerType),
 col("Index_AgeGroup").cast(IntegerType),
 col("Index_Sex").cast(IntegerType),
 col("Index_CurrentICU").cast(IntegerType),
 col("Index_EverHospital").cast(IntegerType),
 col("Index_EverICU").cast(IntegerType))

rankData.show(20)
```

```
scala> :paste
// Entering paste mode (ctrl-D to finish)

val inputIndex= Array("Age Group", "Sex", "Currently in ICU", "Ever Hospitalized", "Ever in ICU")
val outputIndex= Array("Index_AgeGroup","Index_Sex","Index_CurrentICU","Index_EverHospital", "Index_EverICU")

val StringToIndex = new StringIndexer()
StringToIndex.setInputCols(inputIndex)
StringToIndex.setOutputCols(outputIndex)

val StringToIndex2= new StringIndexer()
.setInputCol("Result")
.setOutputCol("Index_Result")

val IndexedData= StringToIndex.fit(BalancedData).transform(BalancedData)
val IndexedData_2= StringToIndex2.fit(IndexedData).transform(IndexedData)

val rankData= IndexedData_2.select(col("Index_Result").cast(IntegerType),
 col("Index_AgeGroup").cast(IntegerType),
 col("Index_Sex").cast(IntegerType),
 col("Index_CurrentICU").cast(IntegerType),
 col("Index_EverHospital").cast(IntegerType),
 col("Index_EverICU").cast(IntegerType))


// Exiting paste mode, now interpreting.

inputIndex: Array[String] = Array(Age Group, Sex, Currently in ICU, Ever Hospitalized, Ever in ICU)
outputIndex: Array[String] = Array(Index_AgeGroup, Index_Sex, Index_CurrentICU, Index_EverHospital, Index_EverICU)
StringToIndex: org.apache.spark.ml.feature.StringIndexer = strIdx_991c838fc092
StringToIndex2: org.apache.spark.ml.feature.StringIndexer = strIdx_ea33b28f6a8e
IndexedData: org.apache.spark.sql.DataFrame = [Age Group: string, Sex: string ... 9 more fields]
IndexedData_2: org.apache.spark.sql.DataFrame = [Age Group: string, Sex: string ... 10 more fields]
scala> rankData.show(20)
+------------+--------------+---------+---------------+------------------+-------------+
|Index_Result|Index_AgeGroup|Index_Sex|Index_CurrentICU|Index_EverHospital|Index_EverICU|
+------------+--------------+---------+---------------+------------------+-------------+
|           0|             2|        0|              0|                 1|            1|
|           0|             3|        0|              0|                 1|            1|
|           0|             1|        0|              0|                 0|            0|
|           0|             1|        0|              0|                 0|            0|
|           0|             2|        0|              0|                 0|            0|
|           0|             1|        0|              0|                 1|            0|
|           0|             1|        1|              0|                 0|            0|
|           0|             1|        1|              0|                 0|            0|
|           0|             2|        1|              0|                 0|            0|
|           0|             7|        0|              0|                 1|            1|
|           0|             0|        0|              0|                 0|            0|
|           0|             3|        1|              0|                 1|            0|
|           0|             0|        0|              0|                 1|            0|
|           0|             0|        1|              0|                 1|            0|
|           0|             3|        0|              0|                 1|            1|
|           0|             6|        1|              0|                 1|            1|
|           0|             2|        0|              0|                 1|            1|
|           0|             0|        0|              0|                 1|            0|
|           0|             2|        0|              0|                 1|            1|
|           0|             7|        0|              0|                 1|            1|
+------------+--------------+---------+---------------+------------------+-------------+
only showing top 20 rows
```

- Converting the categorical data variables to be provided to machine learning algorithms which in turn improve predictions as well as classification accuracy of a model

```
val HotEncoder= new OneHotEncoder()
.setInputCols(Array("Index_AgeGroup","Index_Sex","Index_CurrentICU","Index_EverHospital",
"Index_EverICU"))
.setOutputCols(Array("Encode_AgeGroup","Encode_Sex","Encode_CurrentICU","Encode_EverHospital",
"Encode_EverICU"))

val DataEncoder= HotEncoder.fit(rankData).transform(rankData)
DataEncoder.show(20)
```

```
scala> :paste
// Entering paste mode (ctrl-D to finish)

val HotEncoder= new OneHotEncoder()
.setInputCols(Array("Index_AgeGroup","Index_Sex","Index_CurrentICU","Index_EverHospital", "Index_EverICU"))
.setOutputCols(Array("Encode_AgeGroup","Encode_Sex","Encode_CurrentICU","Encode_EverHospital", "Encode_EverICU"))

val DataEncoder= HotEncoder.fit(rankData).transform(rankData)

// Exiting paste mode, now interpreting.

HotEncoder: org.apache.spark.ml.feature.OneHotEncoder = oneHotEncoder_fcc37cbf3575
DataEncoder: org.apache.spark.sql.DataFrame = [Index_Result: int, Index_AgeGroup: int ... 9 more fields]
```

```
scala> DataEncoder.show(20)
+------------+--------------+---------+----------------+-----------------+-------------+-----------------+---------------+-------------+-----------------+-------------+
|Index_Result|Index_AgeGroup|Index_Sex|Index_CurrentICU|Index_EverHospital|Index_EverICU|Encode_CurrentICU|Encode_AgeGroup|   Encode_Sex|Encode_EverHospital|Encode_EverICU|
+------------+--------------+---------+----------------+-----------------+-------------+-----------------+---------------+-------------+-----------------+-------------+
|           0|             2|        0|               0|                1|            1|        (0,[],[])|  (8,[2],[1.0])|(3,[0],[1.0])|          (1,[],[])|     (1,[],[])|
|           0|             3|        0|               0|                1|            1|        (0,[],[])|  (8,[3],[1.0])|(3,[0],[1.0])|          (1,[],[])|     (1,[],[])|
|           0|             1|        0|               0|                0|            0|        (0,[],[])|  (8,[1],[1.0])|(3,[0],[1.0])|      (1,[0],[1.0])| (1,[0],[1.0])|
|           0|             1|        0|               0|                0|            0|        (0,[],[])|  (8,[1],[1.0])|(3,[0],[1.0])|      (1,[0],[1.0])| (1,[0],[1.0])|
|           0|             2|        0|               0|                0|            0|        (0,[],[])|  (8,[2],[1.0])|(3,[0],[1.0])|      (1,[0],[1.0])| (1,[0],[1.0])|
|           0|             1|        0|               0|                1|            0|        (0,[],[])|  (8,[1],[1.0])|(3,[0],[1.0])|          (1,[],[])| (1,[0],[1.0])|
|           0|             1|        1|               0|                0|            0|        (0,[],[])|  (8,[1],[1.0])|(3,[1],[1.0])|      (1,[0],[1.0])| (1,[0],[1.0])|
|           0|             1|        1|               0|                0|            0|        (0,[],[])|  (8,[1],[1.0])|(3,[1],[1.0])|      (1,[0],[1.0])| (1,[0],[1.0])|
|           0|             2|        1|               0|                0|            0|        (0,[],[])|  (8,[2],[1.0])|(3,[1],[1.0])|      (1,[0],[1.0])| (1,[0],[1.0])|
|           0|             7|        0|               0|                1|            1|        (0,[],[])|  (8,[7],[1.0])|(3,[0],[1.0])|          (1,[],[])|     (1,[],[])|
|           0|             0|        0|               0|                0|            0|        (0,[],[])|  (8,[0],[1.0])|(3,[0],[1.0])|      (1,[0],[1.0])| (1,[0],[1.0])|
|           0|             3|        1|               0|                1|            0|        (0,[],[])|  (8,[3],[1.0])|(3,[1],[1.0])|          (1,[],[])| (1,[0],[1.0])|
|           0|             0|        0|               0|                1|            0|        (0,[],[])|  (8,[0],[1.0])|(3,[0],[1.0])|          (1,[],[])| (1,[0],[1.0])|
|           0|             0|        1|               0|                1|            0|        (0,[],[])|  (8,[0],[1.0])|(3,[1],[1.0])|          (1,[],[])| (1,[0],[1.0])|
|           0|             3|        0|               0|                1|            1|        (0,[],[])|  (8,[3],[1.0])|(3,[0],[1.0])|          (1,[],[])|     (1,[],[])|
|           0|             6|        1|               0|                1|            1|        (0,[],[])|  (8,[6],[1.0])|(3,[1],[1.0])|          (1,[],[])|     (1,[],[])|
|           0|             2|        0|               0|                1|            1|        (0,[],[])|  (8,[2],[1.0])|(3,[0],[1.0])|          (1,[],[])|     (1,[],[])|
|           0|             0|        0|               0|                1|            0|        (0,[],[])|  (8,[0],[1.0])|(3,[0],[1.0])|          (1,[],[])| (1,[0],[1.0])|
|           0|             2|        0|               0|                1|            1|        (0,[],[])|  (8,[2],[1.0])|(3,[0],[1.0])|          (1,[],[])|     (1,[],[])|
|           0|             7|        0|               0|                1|            1|        (0,[],[])|  (8,[7],[1.0])|(3,[0],[1.0])|          (1,[],[])|     (1,[],[])|
+------------+--------------+---------+----------------+-----------------+-------------+-----------------+---------------+-------------+-----------------+-------------+
only showing top 20 rows
```

- Applying the Random Forest Supervised Machine Learning Algorithm

```
val Array(trainingData,testData)= DataEncoder.randomSplit(Array(0.7,0.3),754)

val DataAssembler = new VectorAssembler()
.setInputCols(Array("Encode_AgeGroup","Encode_Sex","Encode_CurrentICU","Encode_EverHospital",
"Encode_EverICU",
"Index_AgeGroup","Index_Sex","Index_CurrentICU","Index_EverHospital", "Index_EverICU"))
.setOutputCol("assembled-features")

val Random_Forest =new RandomForestClassifier()
.setFeaturesCol("assembled-features")
.setLabelCol("Index_Result")
.setSeed(1234)

val DataPipeline = new Pipeline()
.setStages(Array(DataAssembler,Random_Forest))

val DataEvaluator = new MulticlassClassificationEvaluator()
.setLabelCol("Index_Result")
.setPredictionCol("prediction")
.setMetricName("accuracy")

val DataParamGrid = new ParamGridBuilder()
.addGrid(Random_Forest.maxDepth,Array(3,5))
.addGrid(Random_Forest.impurity,Array("entropy","gini")).build()

val cross_validator= new CrossValidator()
.setEstimator(DataPipeline)
.setEvaluator(DataEvaluator)
.setEstimatorParamMaps(DataParamGrid)
.setNumFolds(3)
```

```
scala> :paste
// Entering paste mode (ctrl-D to finish)

val Array(trainingData,testData)= DataEncoder.randomSplit(Array(0.7,0.3),754)

val DataAssembler = new VectorAssembler()
.setInputCols(Array("Encode_AgeGroup","Encode_Sex","Encode_CurrentICU","Encode_EverHospital", "Encode_EverICU",
"Index_AgeGroup","Index_Sex","Index_CurrentICU","Index_EverHospital", "Index_EverICU"))
.setOutputCol("assembled-features")

// Exiting paste mode, now interpreting.

trainingData: org.apache.spark.sql.Dataset[org.apache.spark.sql.Row] = [Index_Result: int, Index_AgeGroup: int ... 9 more fields]
testData: org.apache.spark.sql.Dataset[org.apache.spark.sql.Row] = [Index_Result: int, Index_AgeGroup: int ... 9 more fields]
DataAssembler: org.apache.spark.ml.feature.VectorAssembler = VectorAssembler: uid=vecAssembler_337e7c156dc8, handleInvalid=error, numInputCols=10

scala> :paste
// Entering paste mode (ctrl-D to finish)

val Random_Forest =new RandomForestClassifier()
.setFeaturesCol("assembled-features")
.setLabelCol("Index_Result")
.setSeed(1234)

val DataPipeline = new Pipeline()
.setStages(Array(DataAssembler,Random_Forest))

// Exiting paste mode, now interpreting.

Random_Forest: org.apache.spark.ml.classification.RandomForestClassifier = rfc_010df82a931a
DataPipeline: org.apache.spark.ml.Pipeline = pipeline_bfdc1c8a4b0a

scala> :paste
// Entering paste mode (ctrl-D to finish)

val DataEvaluator = new MulticlassClassificationEvaluator()
.setLabelCol("Index_Result")
.setPredictionCol("Prediction")
.setMetricName("Accuracy")

// Exiting paste mode, now interpreting.

java.lang.IllegalArgumentException: mcEval_91a40fc17b3c parameter metricName given invalid value Accuracy.
  at org.apache.spark.ml.param.Param.validate(params.scala:76)
  at org.apache.spark.ml.param.ParamPair.<init>(params.scala:634)
  at org.apache.spark.ml.param.Param.$minus$greater(params.scala:85)
  at org.apache.spark.ml.param.Params.set(params.scala:713)
  at org.apache.spark.ml.param.Params.set$(params.scala:712)
  at org.apache.spark.ml.evaluation.Evaluator.set(Evaluator.scala:28)
  at org.apache.spark.ml.evaluation.MulticlassClassificationEvaluator.setMetricName(MulticlassClassificationEvaluator.scala:65)
  ... 57 elided

scala> :paste
// Entering paste mode (ctrl-D to finish)

val DataEvaluator = new MulticlassClassificationEvaluator()
.setLabelCol("Index_Result")
.setPredictionCol("prediction")
.setMetricName("accuracy")

// Exiting paste mode, now interpreting.

DataEvaluator: org.apache.spark.ml.evaluation.MulticlassClassificationEvaluator = MulticlassClassificationEvaluator: uid=mcEval_0b2f407d72f3, metricName=accuracy, metricLabel=0.0, beta=1.0, eps=1.
scala> :paste
// Entering paste mode (ctrl-D to finish)

val DataParamGrid = new ParamGridBuilder()
.addGrid(Random_Forest.maxDepth,Array(3,5))
.addGrid(Random_Forest.impurity,Array("entropy","gini")).build()

val cross_validator= new CrossValidator()
.setEstimator(DataPipeline)
.setEvaluator(DataEvaluator)
.setEstimatorParamMaps(DataParamGrid)
.setNumFolds(3)


// Exiting paste mode, now interpreting.

DataParamGrid: Array[org.apache.spark.ml.param.ParamMap] =
Array({
        rfc_010df82a931a-impurity: entropy,
        rfc_010df82a931a-maxDepth: 3
}, {
        rfc_010df82a931a-impurity: entropy,
        rfc_010df82a931a-maxDepth: 5
}, {
        rfc_010df82a931a-impurity: gini,
        rfc_010df82a931a-maxDepth: 3
}, {
        rfc_010df82a931a-impurity: gini,
        rfc_010df82a931a-maxDepth: 5
})
cross_validator: org.apache.spark.ml.tuning.CrossValidator = cv_19b737fe7136
```

- Training the model on training data and predicting it with the test data

```
val cvModel = cross_validator.fit(trainingData)
val Predictions = cvModel.transform(testData)
```

```
scala> val cvModel = cross_validator.fit(trainingData)
cvModel: org.apache.spark.ml.tuning.CrossValidatorModel = CrossValidatorModel: uid=cv_19b737fe7136, bestModel=pipeline_bfdc1c8a4b0a, numFolds=3

scala> val Predictions = cvModel.transform(testData)
Predictions: org.apache.spark.sql.DataFrame = [Index_Result: int, Index_AgeGroup: int ... 13 more fields]
```

- Evaluating the model and calculating the accuracy

```
val Accuracy = DataEvaluator.evaluate(Predictions)
println("Accuracy on Test Data =" + Accuracy)
```

```
scala> :paste
// Entering paste mode (ctrl-D to finish)

val Accuracy = DataEvaluator.evaluate(Predictions)
println("Accuracy on Test Data =" + Accuracy)

// Exiting paste mode, now interpreting.

Accuracy on Test Data =0.9368376430690971
Accuracy: Double = 0.9368376430690971
```

- Final Output File

```
Predictions.select(col("Index_Result"),
 col("Index_AgeGroup"),
 col("Index_Sex"),
 col("Index_CurrentICU"),
 col("Index_EverHospital"),
 col("Index_EverICU"),
 col("Prediction"))
  .write
  .format("csv")
  .save("hdfs://10.128.0.24:8020/BigData/CovidData/predictive_model.csv")
```

```
scala> :paste
// Entering paste mode (ctrl-D to finish)

Predictions.select(col("Index_Result"),
 col("Index_AgeGroup"),
 col("Index_Sex"),
 col("Index_CurrentICU"),
 col("Index_EverHospital"),
 col("Index_EverICU"),
 col("Prediction"))
  .write
  .format("csv")
  .save("hdfs://10.128.0.24:8020/BigData/CovidData/predictive_model/")

// Exiting paste mode, now interpreting.
```

References

- **Arnold YS Yeung, F. R.-D. (2021). Machine Learning–Based Prediction of Growth in Confirmed COVID-19 Infection Cases in 114 Countries Using Metrics of Nonpharmaceutical Interventions and Cultural Dimensions: Model Development and Validation. Journal of Medical Internet Research.**
- **Cindy Feng, G. K.-C. (2021). Predicting COVID-19 mortality risk in Toronto, Canada: a comparison of tree-based and regression-based machine learning methods. BMC Medical Research Methodology.**
- **Phatak, M. (2017, December 18). spark-ml-kaggle. Retrieved from GitHub: https://github.com/phatak-dev/spark-ml-kaggle/blob/master/src/main/scala/com/madhukaraphatak/spark/ml/UnderSampling.scala**
- **https://www.rapidtables.com/tools/pie-chart.html**
- **https://open.toronto.ca/dataset/covid-19-cases-in-toronto/**