**COVID Vaccines Analysis**

**DAC\_PHASE 3 Submission document**

Coronavirus disease 2019 (COVID-19), also known as the coronavirus, or COVID, is a contagious disease caused by severe acute respiratory syndrome coronavirus 2 (SARS-CoV-2). The first known case was identified in Wuhan, China, in December 2019. The disease has since spread worldwide, leading to an ongoing pandemic. And this data is all about the COVID 19 Vaccination Coverage across Health Unit district wise in Tamil Nadu.

1 Health Unit District

2 Achievement towards vaccination of 1st Dosage Covishield to HCW

3 Achievement towards vaccination of 2nd Dosage Covishield to HCW

4 Achievement towards vaccination of 1st Dosage Covishield to FLW

5 Achievement towards vaccination of 2nd Dosage Covishield to FLW

6 Achievement towards vaccination of 1st Dosage Covishield to beneficiaries of 18 years and less than 44 years age group

7 Achievement towards vaccination of 2nd Dosage Covishield to beneficiaries of 18 years and less than 44 years age group

8 Achievement towards vaccination of 1st Dosage Covishield to beneficiaries of 45 years and less than 60 years age group with Comorbidities

9 Achievement towards vaccination of 2nd Dosage Covishield to beneficiaries of 45 years and less than 60 years age group with Comorbidities

10 Achievement towards vaccination of 1st Dosage Covishield to 60+ years beneficiaries with Comorbidities

11 Achievement towards vaccination of 2nd Dosage Covishield to 60+ years beneficiaries with Comorbidities

12 Total Achievement of vaccination to beneficiaries under 1st Dose of Covishield

13 Total Achievement of vaccination to beneficiaries under 2nd Dose of Covishield

14 Achievement towards vaccination of 1st Dosage Covaxin to HCW

15 Achievement towards vaccination of 2nd Dosage Covaxin to HCW

16 Achievement towards vaccination of 1st Dosage Covaxin to FLW

17 Achievement towards vaccination of 2nd Dosage Covaxin to FLW

18 Achievement towards vaccination of 1st Dosage Covaxin to beneficiaries of 18 years and less than 44 years age group

19 Achievement towards vaccination of 2nd Dosage Covaxin to beneficiaries of 18 years and less than 44 years age group

20 Achievement towards vaccination of 1st Dosage Covaxin to beneficiaries of 45 years and less than 60 years age group with Comorbidities

21 Achievement towards vaccination of 2nd Dosage Covaxin to beneficiaries of 45 years and less than 60 years age group with Comorbidities

22 Achievement towards vaccination of 1st Dosage Covaxin to 60+ years beneficiaries with Comorbidities

23 Achievement towards vaccination of 2nd Dosage Covaxin to 60+ years beneficiaries with Comorbidities

24 Total Achievement of vaccination to beneficiaries under 1st Dose of Covaxin

25 Total Achievement of vaccination to beneficiaries under 2nd Dose of Covaxin

26 Total Achievement towards vaccination of 1st Dosage Covishield and Covaxin to HCW

27 Total Achievement towards vaccination of 2nd Dosage Covishield and Covaxin to HCW

28 Total Achievement towards vaccination of 1st Dosage Covishield and Covaxin to FLW

29 Total Achievement towards vaccination of 2nd Dosage Covishield and Covaxin to FLW

30 Total Achievement towards vaccination of 1st Dosage Covishield and Covaxin to beneficiaries of 18 years and less than 44 years age group 31 Total Achievement towards vaccination of 2nd Dosage Covishield and Covaxin to beneficiaries of 18 years and less than 44 years age group

32 Total Achievement towards vaccination of 1st Dosage Covishield and Covaxin to beneficiaries of 45 years and less than 60 years age group with Comorbidities

33 Total Achievement towards vaccination of 2nd Dosage Covishield and Covaxin to beneficiaries of 45 years and less than 60 years age group with Comorbidities

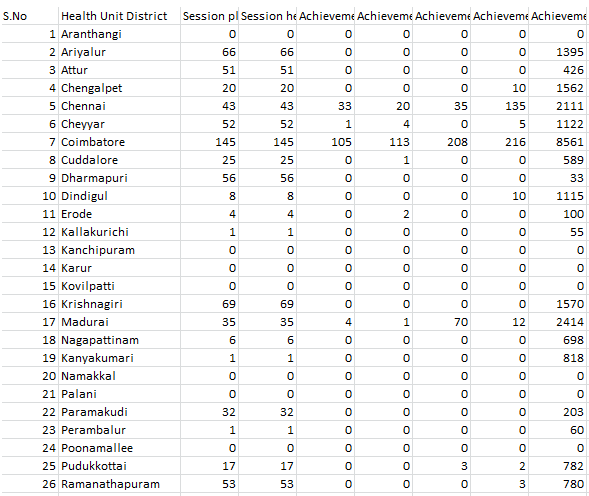
34 Total Achievement towards vaccination of 1st Dosage Covishield and Covaxin to 60+ years beneficiaries with Comorbidities

35 Total Achievement towards vaccination of 2nd Dosage Covishield and Covaxin to 60+ years beneficiaries with Comorbidities

36 Total Achievement towards vaccination to beneficiaries under 1st Dose of Covishield and Covaxin

37 Total Achievement towards vaccination to beneficiaries under 2nd Dose of Covishield and Covaxin

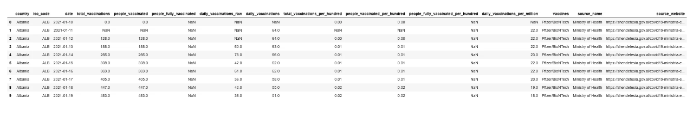
38 Total Achievement towards vaccination of Covishield and Covaxin (1st and 2nd Dose)



libraries are needed:

import pandas as pd # to import the dataset  
import numpy as np # to handle matrices  
import matplotlib.pyplot as plt # to plot

dataset = pd.read\_csv("country\_vaccinations.csv")  
dataset.head(10) # we check the first 10 rows of our dataset



dataset.columns # we read the column namesIndex(['country', 'iso\_code', 'date', 'total\_vaccinations',  
 'people\_vaccinated', 'people\_fully\_vaccinated',  
 'daily\_vaccinations\_raw', 'daily\_vaccinations',  
 'total\_vaccinations\_per\_hundred', 'people\_vaccinated\_per\_hundred',  
 'people\_fully\_vaccinated\_per\_hundred', 'daily\_vaccinations\_per\_million',  
 'vaccines', 'source\_name', 'source\_website'],  
 dtype='object')

As i’m planning to predict the **total vaccinations (y)** based on the other **features of the dataset (x)**, where x is my independent variable (in this case, more than one) and y is the dependent variable.

x\_df = dataset[['country', 'date','daily\_vaccinations\_raw', 'daily\_vaccinations','total\_vaccinations\_per\_hundred', 'people\_vaccinated\_per\_hundred',  
 'people\_fully\_vaccinated\_per\_hundred', 'daily\_vaccinations\_per\_million','vaccines']] # independent variables  
y\_df = dataset[['total\_vaccinations']] # dependent variable  
  
x = x\_df.values # as array  
y = y\_df.values # as arrayx[0:10,:] # we check the first 10 rows of our array xarray([['Albania', '2021-01-10', nan, nan, 0.0, 0.0, nan, nan,  
 'Pfizer/BioNTech'],  
 ['Albania', '2021-01-11', nan, 64.0, nan, nan, nan, 22.0,  
 'Pfizer/BioNTech'],  
 ['Albania', '2021-01-12', nan, 64.0, 0.0, 0.0, nan, 22.0,  
 'Pfizer/BioNTech'],  
 ['Albania', '2021-01-13', 60.0, 63.0, 0.01, 0.01, nan, 22.0,  
 'Pfizer/BioNTech'],  
 ['Albania', '2021-01-14', 78.0, 66.0, 0.01, 0.01, nan, 23.0,  
 'Pfizer/BioNTech'],  
 ['Albania', '2021-01-15', 42.0, 62.0, 0.01, 0.01, nan, 22.0,  
 'Pfizer/BioNTech'],  
 ['Albania', '2021-01-16', 61.0, 62.0, 0.01, 0.01, nan, 22.0,  
 'Pfizer/BioNTech'],  
 ['Albania', '2021-01-17', 36.0, 58.0, 0.01, 0.01, nan, 20.0,  
 'Pfizer/BioNTech'],  
 ['Albania', '2021-01-18', 42.0, 55.0, 0.02, 0.02, nan, 19.0,  
 'Pfizer/BioNTech'],  
 ['Albania', '2021-01-19', 36.0, 51.0, 0.02, 0.02, nan, 18.0,  
 'Pfizer/BioNTech']], dtype=object)y[0:10,:] # we check the first 10 rows of our array yarray([[ 0.],  
 [ nan],  
 [128.],  
 [188.],  
 [266.],  
 [308.],  
 [369.],  
 [405.],  
 [447.],  
 [483.]])

**Taking care of missing data**

We notice that the dataset has some ‘nan’ values, which we can not use to train and test our model, so **we need to normalize** them, although deleting them is also an option, we will replace them using the SimpleImputer class.

from sklearn.impute import SimpleImputer # importing the SimpleImputer class that let's us replace the missing values   
 # with the average of the column  
  
imputer = SimpleImputer(missing\_values= np.nan, strategy="mean")

Applied to x:

imputer.fit(X = x[:, 2:8]) # expects only the numerical values columns, not the category ones  
  
x[:,2:8] = imputer.transform(x[:,2:8]) # we replace the missing values on the original array  
x[0:10,:]array([['Albania', '2021-01-10', 74045.48985801217, 57184.43567251462,  
 0.0, 0.0, 1.7913693270735522, 2284.0280701754386,  
 'Pfizer/BioNTech'],  
 ['Albania', '2021-01-11', 74045.48985801217, 64.0,  
 6.074087996582658, 5.177969072164948, 1.7913693270735522, 22.0,  
 'Pfizer/BioNTech'],  
 ['Albania', '2021-01-12', 74045.48985801217, 64.0, 0.0, 0.0,  
 1.7913693270735522, 22.0, 'Pfizer/BioNTech'],  
 ['Albania', '2021-01-13', 60.0, 63.0, 0.01, 0.01,  
 1.7913693270735522, 22.0, 'Pfizer/BioNTech'],  
 ['Albania', '2021-01-14', 78.0, 66.0, 0.01, 0.01,  
 1.7913693270735522, 23.0, 'Pfizer/BioNTech'],  
 ['Albania', '2021-01-15', 42.0, 62.0, 0.01, 0.01,  
 1.7913693270735522, 22.0, 'Pfizer/BioNTech'],  
 ['Albania', '2021-01-16', 61.0, 62.0, 0.01, 0.01,  
 1.7913693270735522, 22.0, 'Pfizer/BioNTech'],  
 ['Albania', '2021-01-17', 36.0, 58.0, 0.01, 0.01,  
 1.7913693270735522, 20.0, 'Pfizer/BioNTech'],  
 ['Albania', '2021-01-18', 42.0, 55.0, 0.02, 0.02,  
 1.7913693270735522, 19.0, 'Pfizer/BioNTech'],  
 ['Albania', '2021-01-19', 36.0, 51.0, 0.02, 0.02,  
 1.7913693270735522, 18.0, 'Pfizer/BioNTech']], dtype=object)

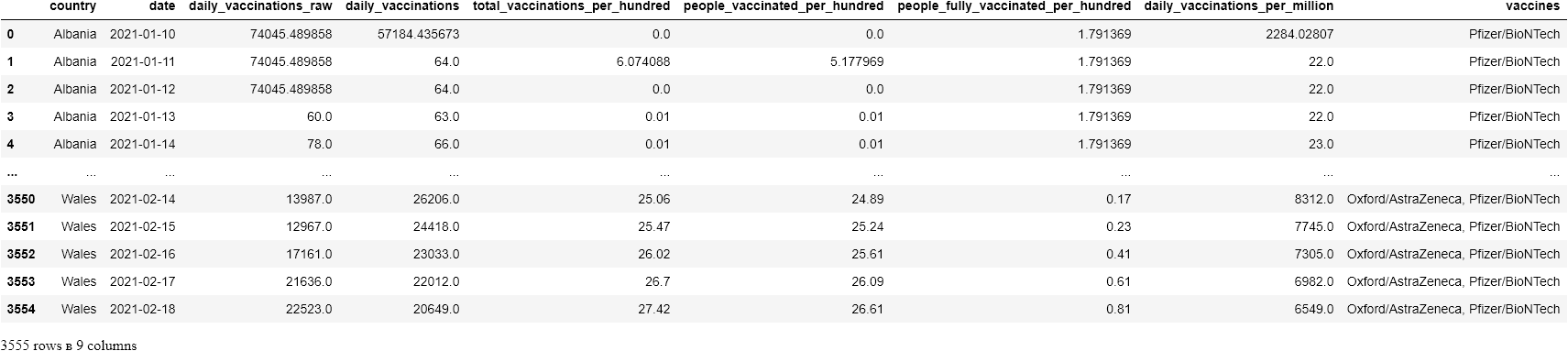
Applied to y:

imputer.fit(X = y) # expects only the numerical values columns, not the category ones  
  
y = imputer.transform(y) # we replace the missing values on the original array  
y[0:10,:]array([[0.00000000e+00],  
 [1.48631876e+06],  
 [1.28000000e+02],  
 [1.88000000e+02],  
 [2.66000000e+02],  
 [3.08000000e+02],  
 [3.69000000e+02],  
 [4.05000000e+02],  
 [4.47000000e+02],  
 [4.83000000e+02]])

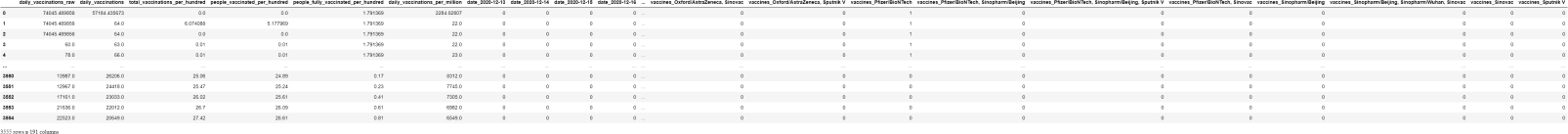
**Encoding categorical data**

In order to train our models with categorical data, we need to **encode** the categories into “dummy variables”, this needs to be done because we can not train regression models directly with strings.

x\_df = pd.DataFrame(x, columns = ['country', 'date','daily\_vaccinations\_raw', 'daily\_vaccinations','total\_vaccinations\_per\_hundred', 'people\_vaccinated\_per\_hundred','people\_fully\_vaccinated\_per\_hundred', 'daily\_vaccinations\_per\_million','vaccines'])  
x\_df



x\_df\_nodummies = x\_df  
x\_df = pd.get\_dummies(x\_df, columns=["date","country", "vaccines"], prefix=["date","country", "vaccines"]) # now, x\_df has the dummy variablesx\_df # we verify th structure with the dummy variables



x = x\_df.values # convertind the data frame into an array  
xarray([[74045.48985801217, 57184.43567251462, 0.0, ..., 0, 0, 0],  
 [74045.48985801217, 64.0, 6.074087996582658, ..., 0, 0, 0],  
 [74045.48985801217, 64.0, 0.0, ..., 0, 0, 0],  
 ...,  
 [17161.0, 23033.0, 26.02, ..., 0, 0, 0],  
 [21636.0, 22012.0, 26.7, ..., 0, 0, 0],  
 [22523.0, 20649.0, 27.42, ..., 0, 0, 0]], dtype=object)yarray([[0.00000000e+00],  
 [1.48631876e+06],  
 [1.28000000e+02],  
 ...,  
 [8.20339000e+05],  
 [8.41975000e+05],  
 [8.64498000e+05]])

**Splitting the dataset into the Training set and Test set**

Let’s create our train and test arrays with 20% of the data corresponding to the test array.

from sklearn.model\_selection import train\_test\_split  
x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size = 0.2, random\_state = 1)len(x\_train) # verifying its size2844len(x\_test) # verifying its size711

# Regression Model Application

## 1. ****Multiple Linear Regression****

## Training the Multiple Linear Regression model on the Training set

from sklearn.linear\_model import LinearRegression  
  
regressor = LinearRegression()  
regressor.fit(x\_train, y\_train)LinearRegression()

## Predicting the Test set results

We test the trained model with the test data and concatenate it in an array for easy comparison

y\_pred = regressor.predict(x\_test)  
np.set\_printoptions(precision = 0)  
print(np.concatenate((y\_pred.reshape(len(y\_pred),1), y\_test.reshape(len(y\_test),1)),1))[[ 1.e+06 3.e+04]  
 [-8.e+05 0.e+00]  
 [ 5.e+06 5.e+06]  
 ...  
 [-1.e+05 9.e+04]  
 [ 2.e+07 2.e+07]  
 [-2.e+05 2.e+04]]

## Evaluating the Model Performance

We obtain an r squared of 0.82, meaning a fit of 82%

from sklearn.metrics import r2\_score  
r2\_score(y\_test, y\_pred)0.820361971864213

## 2. Polynomial Regression

## Training the Polynomial Regression model on the Training set

from sklearn.preprocessing import PolynomialFeatures  
from sklearn.linear\_model import LinearRegression  
  
poly\_reg = PolynomialFeatures(degree = 2)  
x\_poly = poly\_reg.fit\_transform(x\_train)  
regressor = LinearRegression()  
regressor.fit(x\_poly, y\_train)LinearRegression()

## Predicting the Test set results

y\_pred = regressor.predict(poly\_reg.transform(x\_test))  
np.set\_printoptions(precision=2)  
print(np.concatenate((y\_pred.reshape(len(y\_pred),1), y\_test.reshape(len(y\_test),1)),1))[[-5.50e+03 3.20e+04]  
 [-2.63e+08 0.00e+00]  
 [ 7.68e+06 5.30e+06]  
 ...  
 [ 5.99e+04 9.13e+04]  
 [ 2.42e+07 2.27e+07]  
 [ 2.23e+04 1.93e+04]]

## Evaluating the Model Performance

from sklearn.metrics import r2\_score  
r2\_score(y\_test, y\_pred)-275.73153181051356

This result means the performance of our model was really really bad.

## 3. Decision Tree Regression

## Training the model

from sklearn.tree import DecisionTreeRegressor  
regressor = DecisionTreeRegressor(random\_state = 0)  
regressor.fit(x\_train, y\_train)DecisionTreeRegressor(random\_state=0)

## Predicting the Test set results

y\_pred = regressor.predict(x\_test)  
np.set\_printoptions(precision = 2)  
print(np.concatenate((y\_pred.reshape(len(y\_pred),1), y\_test.reshape(len(y\_test),1)),1))[[6.68e+02 3.20e+04]  
 [0.00e+00 0.00e+00]  
 [5.19e+06 5.30e+06]  
 ...  
 [8.80e+04 9.13e+04]  
 [2.35e+07 2.27e+07]  
 [2.20e+04 1.93e+04]]

## Evaluating Model Performance

from sklearn.metrics import r2\_score  
r2\_score(y\_test, y\_pred)0.9627743674208652

We notice that our Decision Tree Regression model has a fit of 96%, which is actually pretty good!

# 4. Random Forest Regression

## Training the Random Forest Regression model

from sklearn.ensemble import RandomForestRegressor  
regressor = RandomForestRegressor(n\_estimators = 10, random\_state = 0)  
regressor.fit(x\_train, y\_train)

## Predicting the Test set results

y\_pred = regressor.predict(x\_test)  
np.set\_printoptions(precision=2)  
print(np.concatenate((y\_pred.reshape(len(y\_pred),1), y\_test.reshape(len(y\_test),1)),1))[[1.53e+05 3.20e+04]  
 [0.00e+00 0.00e+00]  
 [5.29e+06 5.30e+06]  
 ...  
 [1.28e+05 9.13e+04]  
 [2.15e+07 2.27e+07]  
 [2.11e+04 1.93e+04]]

## Evaluating the Model Performance

from sklearn.metrics import r2\_score  
r2\_score(y\_test, y\_pred)0.9659467525523365plt.plot\_date(x\_df\_nodummies.date, x\_df\_nodummies.vaccines)  
plt.xticks(rotation ='vertical')  
plt.show()



Finally, we observe that the winner of our previously tested regression models is the **Random Forest Regression model with a fit of 96.59%** which is really great! Now, you can be sure that this is the model that will perform better in case you want to make a prediction of the total vaccinations based on the features selected as predictors in x. **(Stay tuned for the part 2, where we will work a little bit more on the predictions and visualizations of the data)**. By the way, don’t forget the performance of each model changes for each dataset, so you actually do need to compare them to select the best one for each case (hint: you can also change their performance by optimizing the default hyperparameters).