**Covid-19 Vaccination Analysis**

**Team members:**

SS Ameesha-961721104012

Gowri Suresh-961721104009

Devichandana AS-961721104006

Nima P-961721104010

**Phase 3 submission document**

**Project Title**: Covid-19 Vaccine Analysis

**Phase 3**: Development Part 1

**Topic**: Start by loading and pre-processing the dataset.

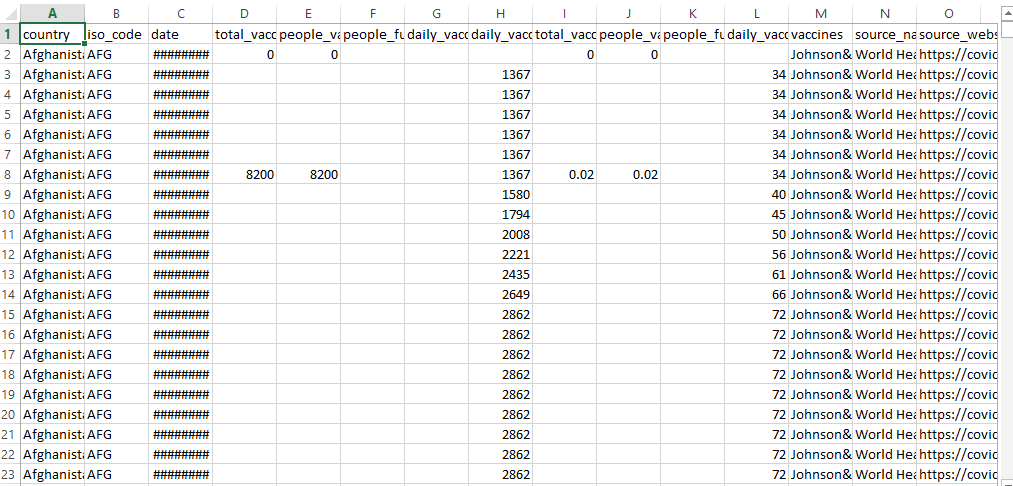
Covid-19 Vaccine Analysis



Introduction:

Vaccines are the most important instrument for bringing the pandemic to a close and saving lives and helping to reduce the risks of infection. It is important that everyone has equal access to immunizations that are both safe and effective. There is no one who is safe until everyone gets vaccinated. COVID-19 vaccinations are a game-changer in the fight against diseases. In addition to examining attitudes toward these vaccines in Africa, Asia, Oceania, Europe, North America, and South America, the purpose of this paper is to predict the acceptability of COVID-19 vaccines and study their predictors.

Given data set:



**Model building by loading the dataset**

Necessary step to follow:

1. Library Used:

* pandas
* matplotlib
* seaborn

**Start by importing the necessary libraries**:

**Program:**

Importing the Libraries

import warnings

warnings.filterwarnings('ignore')

import pandas as pd, numpy as np

import math, json, gc, random, os, sys

from matplotlib import pyplot as plt

from tqdm import tqdm

import tensorflow as tf

import tensorflow\_addons as tfa

import tensorflow.keras.backend as K

import tensorflow.keras.layers as L

from sklearn.model\_selection import train\_test\_split, KFold

Getting the data with some basic EDA

I

train = pd.read\_json('/kaggle/input/stanford-covid-vaccine’) train = pd.read\_json('/kaggle/input/stanford-covid-vaccine/train.json', lines=True)

test = pd.read\_json('/kaggle/input/stanford-covid-vaccine/test.json', lines=True)

sample\_sub = pd.read\_csv('/kaggle/input/stanford-covid-vaccine/sample\_submission.csv')

target\_cols = ['reactivity', 'deg\_Mg\_pH10', 'deg\_pH10', 'deg\_Mg\_50C', 'deg\_50C']

token2int = {x:i for i, x **in** enumerate('().ACGUBEHIMSX')}

def get\_pair\_index\_structure(structure):

structure = np.array([struc for struc **in** structure], dtype="<U4")

open\_index = np.where(structure == "(")[0]

closed\_index = np.where(structure == ")")[0]

structure[open\_index] = range(0, len(open\_index))

structure[closed\_index] = range(len(open\_index)-1, -1, -1)

structure[structure == "."] = -1

structure = structure.astype(int)

pair\_structure = np.array([-1]\*len(structure))

for i **in** range(len(open\_index)):

start, end = np.where(structure == i)[0]

pair\_structure[start] = end

pair\_structure[end] = start

return pair\_structure

def preprocess\_inputs(df, cols=['sequence', 'structure', 'predicted\_loop\_type']):

return np.transpose(

np.array(df[cols].applymap(lambda seq: [token2int[x] for x **in** seq]).values.tolist()),

(0, 2, 1))

train\_inputs = preprocess\_inputs(train[train.signal\_to\_noise > 1])

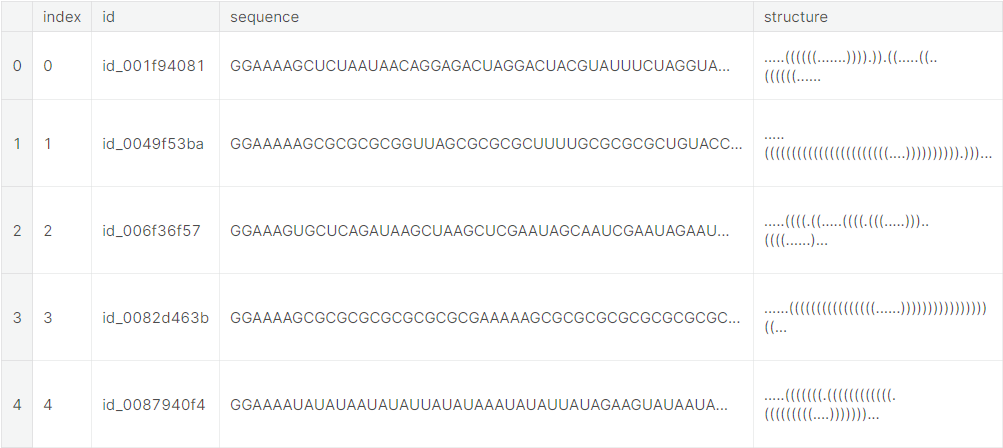
train\_labels = np.array(train[train.signal\_to\_noise > 1][target\_cols].values.tolist()).transpose((0, 2, 1))

def MCRMSE(y\_true, y\_pred):

colwise\_mse = tf.reduce\_mean(tf.square(y\_true - y\_pred), axis=1)

return tf.reduce\_mean(tf.sqrt(colwise\_mse), axis=1)

train.head()



### Defining the Neural Network Layers

def gru\_layer(hidden\_dim, dropout):

return tf.keras.layers.Bidirectional(

tf.keras.layers.GRU(hidden\_dim,

dropout=dropout,

return\_sequences=True,

kernel\_initializer = 'orthogonal'))

def lstm\_layer(hidden\_dim, dropout):

return tf.keras.layers.Bidirectional(

tf.keras.layers.LSTM(hidden\_dim,

dropout=dropout,

return\_sequences=True,

kernel\_initializer = 'orthogonal'))

def build\_model(gru=1,seq\_len=107, pred\_len=68, dropout=0.5,

embed\_dim=75, hidden\_dim=128):

inputs = tf.keras.layers.Input(shape=(seq\_len, 3))

embed = tf.keras.layers.Embedding(input\_dim=len(token2int), output\_dim=embed\_dim)(inputs)

reshaped = tf.reshape(

embed, shape=(-1, embed.shape[1], embed.shape[2] \* embed.shape[3]))

reshaped = tf.keras.layers.SpatialDropout1D(.2)(reshaped)

if gru==1:

hidden = gru\_layer(hidden\_dim, dropout)(reshaped)

hidden = gru\_layer(hidden\_dim, dropout)(hidden)

hidden = gru\_layer(hidden\_dim, dropout)(hidden)

elif gru==0:

hidden = lstm\_layer(hidden\_dim, dropout)(reshaped)

hidden = lstm\_layer(hidden\_dim, dropout)(hidden)

hidden = lstm\_layer(hidden\_dim, dropout)(hidden)

elif gru==3:

hidden = gru\_layer(hidden\_dim, dropout)(reshaped)

hidden = lstm\_layer(hidden\_dim, dropout)(hidden)

hidden = lstm\_layer(hidden\_dim, dropout)(hidden)

elif gru==4:

hidden = lstm\_layer(hidden\_dim, dropout)(reshaped)

hidden = gru\_layer(hidden\_dim, dropout)(hidden)

hidden = gru\_layer(hidden\_dim, dropout)(hidden)

elif gru==5:

hidden = lstm\_layer(hidden\_dim, dropout)(reshaped)

hidden = gru\_layer(hidden\_dim, dropout)(hidden)

hidden = lstm\_layer(hidden\_dim, dropout)(hidden)

elif gru==6:

hidden = lstm\_layer(hidden\_dim, dropout)(reshaped)

hidden = gru\_layer(hidden\_dim, dropout)(hidden)

hidden = lstm\_layer(hidden\_dim, dropout)(hidden)

truncated = hidden[:, :pred\_len]

out = tf.keras.layers.Dense(5, activation='linear')(truncated)

model = tf.keras.Model(inputs=inputs, outputs=out)

adam = tf.optimizers.Adam()

radam = tfa.optimizers.RectifiedAdam()

lookahead = tfa.optimizers.Lookahead(adam, sync\_period=6)

ranger = tfa.optimizers.Lookahead(radam, sync\_period=6)

model.compile(optimizer = adam, loss=MCRMSE)

return model

### Train-Test Split of the dataset

train\_inputs, val\_inputs, train\_labels, val\_labels = train\_test\_split(train\_inputs, train\_labels, test\_size=.1, random\_state=34)

### Training the GRU Model

lr\_callback = tf.keras.callbacks.ReduceLROnPlateau()

gru = build\_model(gru=1)

sv\_gru = tf.keras.callbacks.ModelCheckpoint('model\_gru.h5')

history\_gru = gru.fit(

train\_inputs, train\_labels,

validation\_data=(val\_inputs,val\_labels),

batch\_size=64,

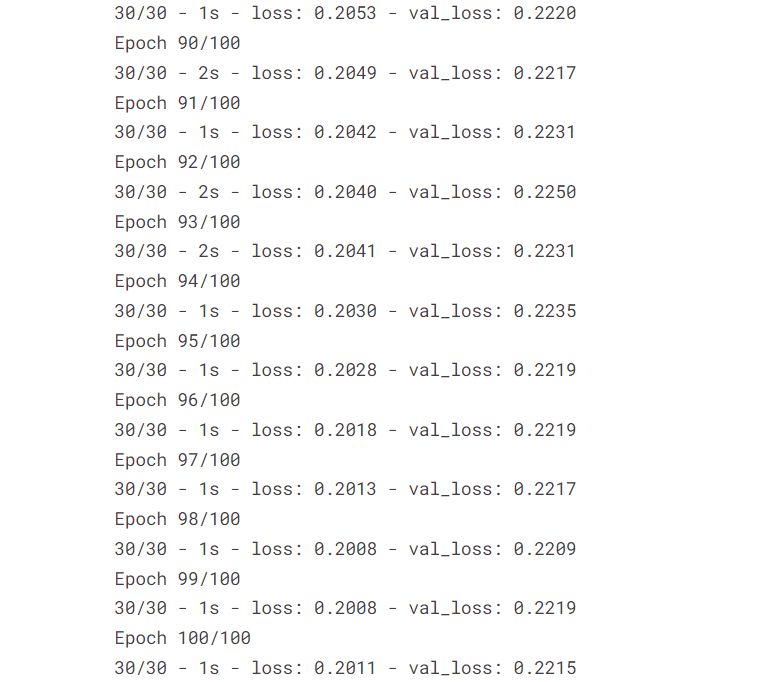
epochs=100,

callbacks=[lr\_callback,sv\_gru],

verbose = 2

)

print(f"Min training loss=**{**min(history\_gru.history['loss'])**}**, min validation loss=**{**min(history\_gru.history['val\_loss'])**}**")

Training the LSTM Model

lstm = build\_model(gru=0)

sv\_lstm = tf.keras.callbacks.ModelCheckpoint('model\_lstm.h5')

history\_lstm = lstm.fit(

train\_inputs, train\_labels,

validation\_data=(val\_inputs,val\_labels),

batch\_size=64,

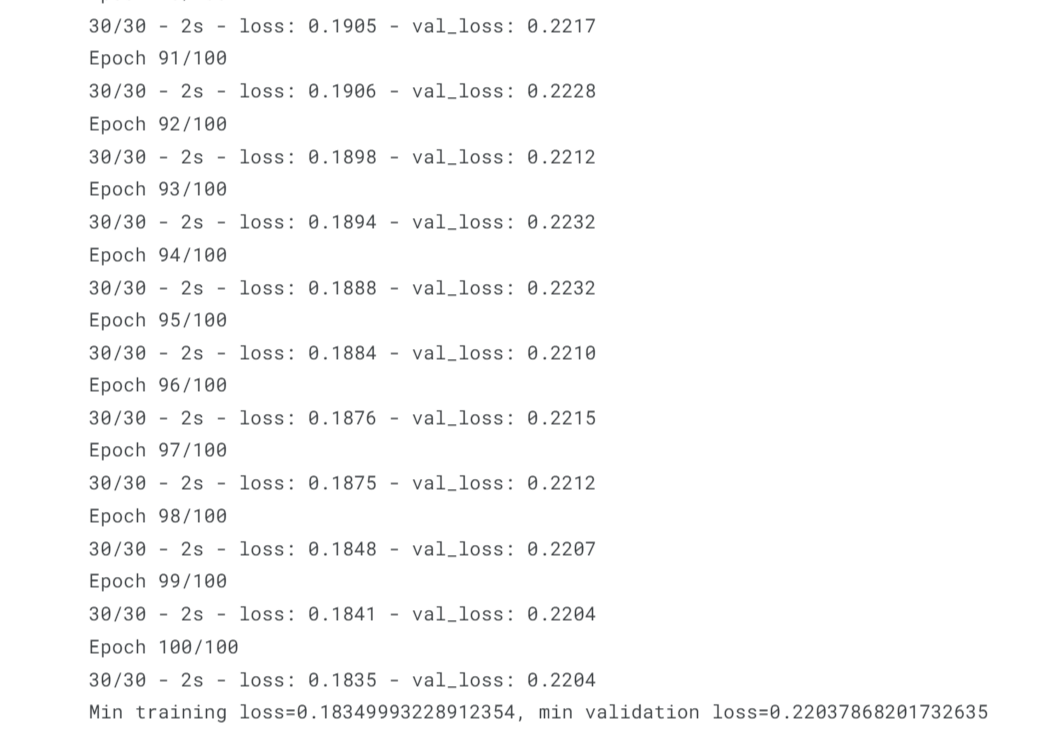
epochs=100,

callbacks=[lr\_callback,sv\_lstm],

verbose = 2

)

print(f"Min training loss=**{**min(history\_lstm.history['loss'])**}**, min validation loss=**{**min(history\_lstm.history['val\_loss'])**}**")



### Training the other models

lstm = build\_model(gru=3)

sv\_lstm = tf.keras.callbacks.ModelCheckpoint('model\_hyb1.h5')

history\_lstm = lstm.fit(

train\_inputs, train\_labels,

validation\_data=(val\_inputs,val\_labels),

batch\_size=64,

epochs=100,

callbacks=[lr\_callback,sv\_lstm],

verbose = 2

)

print(f"Min training loss=**{**min(history\_lstm.history['loss'])**}**, min validation loss=**{**min(history\_lstm.history['val\_loss'])**}**")

### 

lstm = build\_model(gru=5)

sv\_lstm = tf.keras.callbacks.ModelCheckpoint('model\_hyb3.h5')

history\_lstm = lstm.fit(

train\_inputs, train\_labels,

validation\_data=(val\_inputs,val\_labels),

batch\_size=64,

epochs=100,

callbacks=[lr\_callback,sv\_lstm],

verbose = 2

)

print(f"Min training loss=**{**min(history\_lstm.history['loss'])**}**, min validation loss=**{**min(history\_lstm.history['val\_loss'])**}**")

Epoch 91/100

30/30 - 2s - loss: 0.1915 - val\_loss: 0.2197

Epoch 92/100

30/30 - 2s - loss: 0.1909 - val\_loss: 0.2189

Epoch 93/100

30/30 - 2s - loss: 0.1904 - val\_loss: 0.2194

Epoch 94/100

30/30 - 2s - loss: 0.1899 - val\_loss: 0.2217

Epoch 95/100

30/30 - 2s - loss: 0.1898 - val\_loss: 0.2194

Epoch 96/100

30/30 - 2s - loss: 0.1896 - val\_loss: 0.2186

Epoch 97/100

30/30 - 2s - loss: 0.1869 - val\_loss: 0.2176

Epoch 98/100

30/30 - 2s - loss: 0.1856 - val\_loss: 0.2178

Epoch 99/100

30/30 - 2s - loss: 0.1855 - val\_loss: 0.2175

Epoch 100/100

30/30 - 2s - loss: 0.1853 - val\_loss: 0.2179

Min training loss=0.18529632687568665, min validation loss=0.2174813449382782

### Prediction on the test set and Combining with the Submissions File

public\_df = test.query("seq\_length == 107").copy()

private\_df = test.query("seq\_length == 130").copy()

public\_inputs = preprocess\_inputs(public\_df)

private\_inputs = preprocess\_inputs(private\_df)

gru\_short = build\_model(gru=1, seq\_len=107, pred\_len=107)

gru\_long = build\_model(gru=1, seq\_len=130, pred\_len=130)

lstm\_short = build\_model(gru=0, seq\_len=107, pred\_len=107)

lstm\_long = build\_model(gru=0, seq\_len=130, pred\_len=130)

hyb1\_short = build\_model(gru=3, seq\_len=107, pred\_len=107)

hyb1\_long = build\_model(gru=3, seq\_len=130, pred\_len=130)

hyb2\_short = build\_model(gru=4, seq\_len=107, pred\_len=107)

hyb2\_long = build\_model(gru=4, seq\_len=130, pred\_len=130)

hyb3\_short = build\_model(gru=5, seq\_len=107, pred\_len=107)

hyb3\_long = build\_model(gru=5, seq\_len=130, pred\_len=130)

hyb4\_short = build\_model(gru=6, seq\_len=107, pred\_len=107)

hyb4\_long = build\_model(gru=6, seq\_len=130, pred\_len=130)

gru\_short.load\_weights('model\_gru.h5')

gru\_long.load\_weights('model\_gru.h5')

lstm\_short.load\_weights('model\_lstm.h5')

lstm\_long.load\_weights('model\_lstm.h5')

hyb1\_short.load\_weights('model\_hyb1.h5')

hyb1\_long.load\_weights('model\_hyb1.h5')

hyb2\_short.load\_weights('model\_hyb2.h5')

hyb2\_long.load\_weights('model\_hyb2.h5')

hyb3\_short.load\_weights('model\_hyb3.h5')

hyb3\_long.load\_weights('model\_hyb3.h5')

hyb4\_short.load\_weights('model\_hyb4.h5')

hyb4\_long.load\_weights('model\_hyb4.h5')

gru\_public\_preds = gru\_short.predict(public\_inputs)

gru\_private\_preds = gru\_long.predict(private\_inputs)

lstm\_public\_preds = lstm\_short.predict(public\_inputs)

lstm\_private\_preds = lstm\_long.predict(private\_inputs)

hyb1\_public\_preds = hyb1\_short.predict(public\_inputs)

hyb1\_private\_preds = hyb1\_long.predict(private\_inputs)

hyb2\_public\_preds = hyb2\_short.predict(public\_inputs)

hyb2\_private\_preds = hyb2\_long.predict(private\_inputs)

hyb3\_public\_preds = hyb3\_short.predict(public\_inputs)

hyb3\_private\_preds = hyb3\_long.predict(private\_inputs)

hyb4\_public\_preds = hyb4\_short.predict(public\_inputs)

hyb4\_private\_preds = hyb4\_long.predict(private\_inputs)

preds\_gru = []

for df, preds **in** [(public\_df, gru\_public\_preds), (private\_df, gru\_private\_preds)]:

for i, uid **in** enumerate(df.id):

single\_pred = preds[i]

single\_df = pd.DataFrame(single\_pred, columns=target\_cols)

single\_df['id\_seqpos'] = [f'**{**uid**}**\_**{**x**}**' for x **in** range(single\_df.shape[0])]

preds\_gru.append(single\_df)

preds\_gru\_df = pd.concat(preds\_gru)

preds\_lstm = []

for df, preds **in** [(public\_df, lstm\_public\_preds), (private\_df, lstm\_private\_preds)]:

for i, uid **in** enumerate(df.id):

single\_pred = preds[i]

single\_df = pd.DataFrame(single\_pred, columns=target\_cols)

single\_df['id\_seqpos'] = [f'**{**uid**}**\_**{**x**}**' for x **in** range(single\_df.shape[0])]

preds\_lstm.append(single\_df)

preds\_lstm\_df = pd.concat(preds\_lstm)

preds\_hyb1 = []

for df, preds **in** [(public\_df, hyb1\_public\_preds), (private\_df, hyb1\_private\_preds)]:

for i, uid **in** enumerate(df.id):

single\_pred = preds[i]

single\_df = pd.DataFrame(single\_pred, columns=target\_cols)

single\_df['id\_seqpos'] = [f'**{**uid**}**\_**{**x**}**' for x **in** range(single\_df.shape[0])]

preds\_hyb1.append(single\_df)

preds\_hyb1\_df = pd.concat(preds\_hyb1)

preds\_hyb2 = []

for df, preds **in** [(public\_df, hyb2\_public\_preds), (private\_df, hyb2\_private\_preds)]:

for i, uid **in** enumerate(df.id):

single\_pred = preds[i]

single\_df = pd.DataFrame(single\_pred, columns=target\_cols)

single\_df['id\_seqpos'] = [f'**{**uid**}**\_**{**x**}**' for x **in** range(single\_df.shape[0])]

preds\_hyb2.append(single\_df)

preds\_hyb2\_df = pd.concat(preds\_hyb2)

preds\_hyb3 = []

for df, preds **in** [(public\_df, hyb3\_public\_preds), (private\_df, hyb3\_private\_preds)]:

for i, uid **in** enumerate(df.id):

single\_pred = preds[i]

single\_df = pd.DataFrame(single\_pred, columns=target\_cols)

single\_df['id\_seqpos'] = [f'**{**uid**}**\_**{**x**}**' for x **in** range(single\_df.shape[0])]

preds\_hyb3.append(single\_df)

preds\_hyb3\_df = pd.concat(preds\_hyb3)

preds\_hyb4 = []

for df, preds **in** [(public\_df, hyb4\_public\_preds), (private\_df, hyb4\_private\_preds)]:

for i, uid **in** enumerate(df.id):

single\_pred = preds[i]

single\_df = pd.DataFrame(single\_pred, columns=target\_cols)

single\_df['id\_seqpos'] = [f'**{**uid**}**\_**{**x**}**' for x **in** range(single\_df.shape[0])]

preds\_hyb4.append(single\_df)

preds\_hyb4\_df = pd.concat(preds\_hyb4)

blend\_preds\_df = pd.DataFrame()

blend\_preds\_df['id\_seqpos'] = preds\_gru\_df['id\_seqpos']

blend\_preds\_df['reactivity'] = 0.2\*preds\_gru\_df['reactivity'] + 0.2\*preds\_lstm\_df['reactivity'] + 0.2\*preds\_hyb1\_df['reactivity'] + 0.2\*preds\_hyb2\_df['reactivity'] + 0.1\*preds\_hyb3\_df['reactivity']+0.1\*preds\_hyb4\_df['reactivity']

blend\_preds\_df['deg\_Mg\_pH10'] = 0.2\*preds\_gru\_df['deg\_Mg\_pH10'] + 0.2\*preds\_lstm\_df['deg\_Mg\_pH10'] + 0.2\*preds\_hyb1\_df['deg\_Mg\_pH10'] + 0.2\*preds\_hyb2\_df['deg\_Mg\_pH10'] + 0.1\*preds\_hyb3\_df['deg\_Mg\_pH10']+ 0.1\*preds\_hyb4\_df['deg\_Mg\_pH10']

blend\_preds\_df['deg\_pH10'] = 0.2\*preds\_gru\_df['deg\_pH10'] + 0.2\*preds\_lstm\_df['deg\_pH10'] + 0.2\*preds\_hyb1\_df['deg\_pH10'] + 0.2\*preds\_hyb2\_df['deg\_pH10'] + 0.1\*preds\_hyb3\_df['deg\_pH10']+0.1\*preds\_hyb4\_df['deg\_pH10']

blend\_preds\_df['deg\_Mg\_50C'] = 0.2\*preds\_gru\_df['deg\_Mg\_50C'] + 0.2\*preds\_lstm\_df['deg\_Mg\_50C'] + 0.2\*preds\_hyb1\_df['deg\_Mg\_50C'] + 0.2\*preds\_hyb2\_df['deg\_Mg\_50C'] + 0.1\*preds\_hyb3\_df['deg\_Mg\_50C']+0.1\*preds\_hyb4\_df['deg\_Mg\_50C']

blend\_preds\_df['deg\_50C'] = 0.2\*preds\_gru\_df['deg\_50C'] + 0.2\*preds\_lstm\_df['deg\_50C'] + 0.2\*preds\_hyb1\_df['deg\_50C'] + 0.2\*preds\_hyb2\_df['deg\_Mg\_50C'] + 0.1\*preds\_hyb3\_df['deg\_Mg\_50C']+0.1\*preds\_hyb4\_df['deg\_Mg\_50C']

submission = sample\_sub[['id\_seqpos']].merge(blend\_preds\_df, on=['id\_seqpos'])

### The Final Submissions File

submission.head()

submission.to\_csv('submission.csv', index=False)

**Preprocess Dataset**

import numpy as np *# linear algebra*

import pandas as pd *# data processing, CSV file I/O (e.g. pd.read\_csv)*

import plotly.graph\_objects as go

import warnings

warnings.filterwarnings('ignore')

import matplotlib.pyplot as plt

import matplotlib.patches as patches

from ast import literal\_eval

In [2]:

!pip install python-gdcm

# Load Data

submission = pd.read\_csv('/kaggle/input/siim-covid19-detection/sample\_submission.csv', index\_col=None)

image\_df = pd.read\_csv('/kaggle/input/siim-covid19-detection/train\_image\_level.csv', index\_col=None)

study\_df = pd.read\_csv('/kaggle/input/siim-covid19-detection/train\_study\_level.csv', index\_col=None)

print(f"Train image level csv shape : **{**image\_df.shape**}**")

print(f"Train study level csv shape : **{**study\_df.shape**}**")

Train image level csv shape : (6334, 4)

Train study level csv shape : (6054, 5)

import os

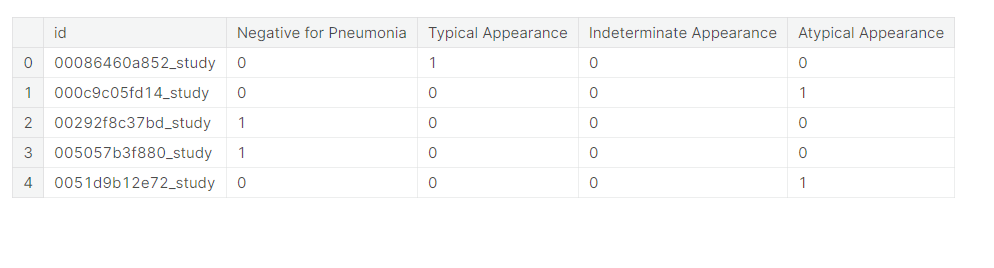
all\_files = []

for dirname, \_, filenames **in** os.walk('/kaggle/input'):

for filename **in** filenames:

all\_files.append(os.path.join(dirname, filename))

study\_df.head()

****

# Explore Study level dataset

study\_group = pd.melt(study\_df, id\_vars=list(study\_df.columns)[:1], value\_vars=list(study\_df.columns)[1:],

var\_name='label', value\_name='value')

study\_group = study\_group.loc[study\_group['value']!=0]

study\_group = study\_group.groupby('label').sum().sort\_values('value',ascending=False).reset\_index()

study\_group['percentage']= round((study\_group['value'] / study\_group['value'].sum())\*100,2)

study\_group.head()

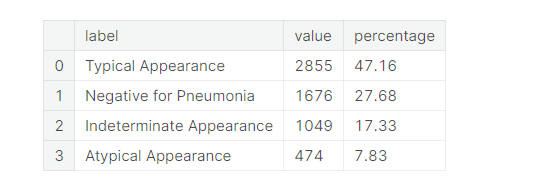
****

fig = go.Figure(data=[go.Pie(labels=study\_group['label'],

values=study\_group['percentage'],

hole=.3,

pull=[0.1, 0.1, 0.1, 0.1]

)

]

)

fig.update\_traces(hoverinfo='label+percent', textinfo='percent', textfont\_size=16,

marker=dict(colors=study\_group['color'], line=dict(color='#000000', width=2))

)

fig.update\_layout(title={'text': "**% o**f labels in training data",

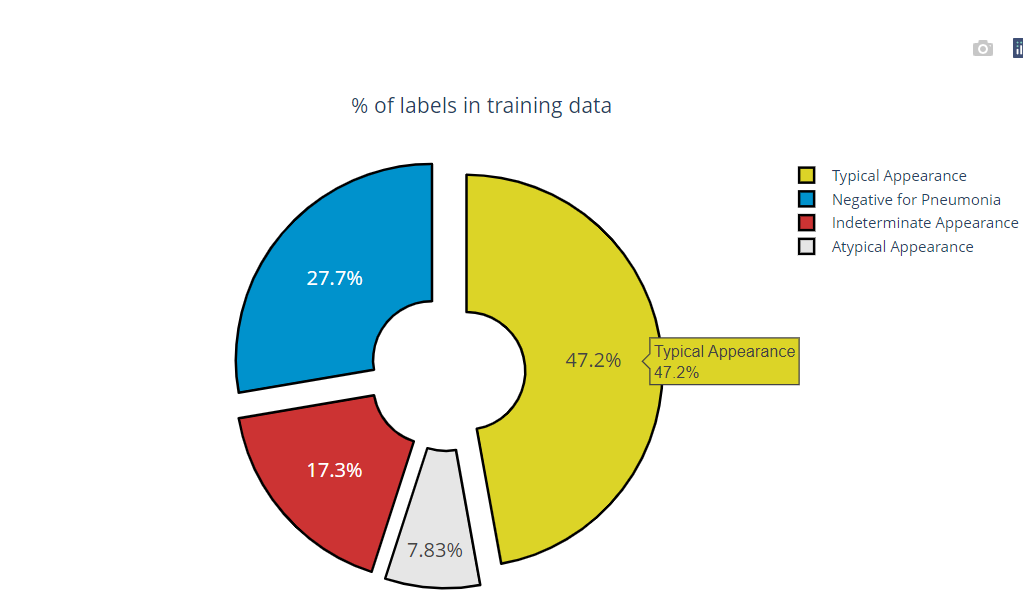
'y':0.9,

'x':0.45,

'xanchor': 'center',

'yanchor': 'top'})

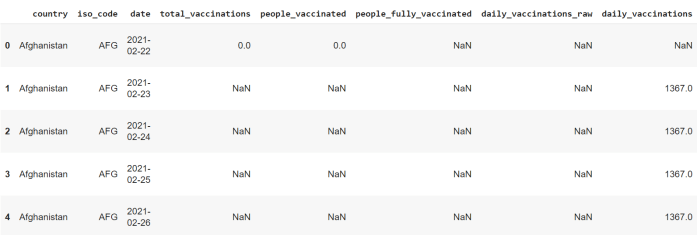
fig.show()



Preprocessing Complete

**Performing data analysis**

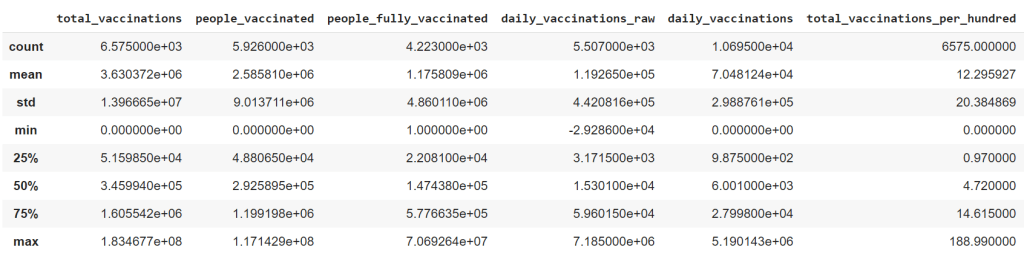
|  |
| --- |
|  |
|  | import pandas as pd |
|  | import matplotlib.pyplot as plt |
|  | import seaborn as sns |
|  | data = pd.read\_csv("country\_vaccinations.csv") |
|  | data.head() |



Now let’s explore this data before we start analyzing the vaccines taken by countries:

1

data.describe()



1

pd.to\_datetime(data.date)

2

data.country.value\_counts()

United Kingdom 118

Northern Ireland 118

Wales 118

England 118

Canada 118

...

Mali 4

Bahamas 2

Brunei 2

Laos 1

Armenia 1

Name: country, Length: 175, dtype: int64

The United Kingdom is made up of England, Scotland, Wales, and Northern Ireland. But in the above data, these countries are mentioned separately with the same values as in the United Kingdom. So this may be an error while recording this data. So let’s see how we can fix this error:

1

data = data[data.country.apply(lambda x: x not in ["England", "Scotland", "Wales", "Northern Ireland"])]

2

data.country.value\_counts()

3

​

Canada 118

United Kingdom 118

China 117

Russia 117

Israel 113

...

Mali 4

Bahamas 2

Brunei 2

Laos 1

Armenia 1

Name: country, Length: 171, dtype: int64

Now let’s explore the vaccines available in this dataset:

1

Moderna, Oxford/AstraZeneca, Pfizer/BioNTech 2587

Oxford/AstraZeneca 1673

Pfizer/BioNTech 1416

Oxford/AstraZeneca, Pfizer/BioNTech 845

Pfizer/BioNTech, Sinovac 475

Moderna, Pfizer/BioNTech 407

Sputnik V 351

Oxford/AstraZeneca, Sinovac 301

Oxford/AstraZeneca, Sinopharm/Beijing 268

Oxford/AstraZeneca, Sinopharm/Beijing, Sputnik V 235

Pfizer/BioNTech, Sinopharm/Beijing 208

Oxford/AstraZeneca, Pfizer/BioNTech, Sinopharm/Beijing, Sputnik V 202

Sinopharm/Beijing 186

Oxford/AstraZeneca, Pfizer/BioNTech, Sinovac 123

Sinopharm/Beijing, Sinopharm/Wuhan, Sinovac 117

EpiVacCorona, Sputnik V 117

Johnson&Johnson, Moderna, Pfizer/BioNTech 112

Oxford/AstraZeneca, Pfizer/BioNTech, Sinovac, Sputnik V 108

Moderna, Oxford/AstraZeneca, Pfizer/BioNTech, Sinopharm/Beijing, Sputnik V 104

Oxford/AstraZeneca, Pfizer/BioNTech, Sinopharm/Beijing, Sinopharm/Wuhan, Sputnik V 96

Covaxin, Oxford/AstraZeneca 86

Sinovac 84

Moderna, Oxford/AstraZeneca 79

Oxford/AstraZeneca, Sinopharm/Beijing, Sinovac 61

Johnson&Johnson 54

Sinopharm/Beijing, Sputnik V 51

Pfizer/BioNTech, Sputnik V 42

Pfizer/BioNTech, Sinovac, Sputnik V 29

Name: vaccines, dtype: int64

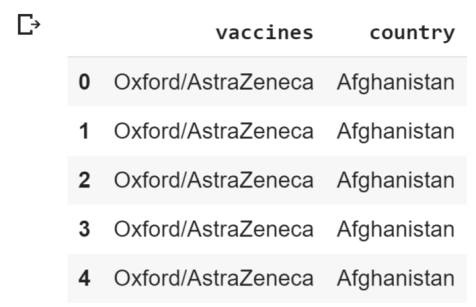
So we have almost all the Covid-19 vaccines available in this dataset. Now I will create a new DataFrame by only selecting the vaccine and the country columns to explore which vaccine is taken by which country:

1

df = data[["vaccines", "country"]]

2

df.head()



Now let’s see how many countries are taking each of the vaccines mentioned in this data:

|  |  |
| --- | --- |
|  | dict\_ = {} |
|  | for i in df.vaccines.unique(): |
|  | dict\_[i] = [df["country"][j] for j in df[df["vaccines"]==i].index] |
|  |  |
|  | vaccines = {} |
|  | for key, value in dict\_.items(): |
|  | vaccines[key] = set(value) |
|  | for i, j in vaccines.items(): |
|  | print(f"{i}:>>{j}") |

Oxford/AstraZeneca:>>{'Gambia', 'Maldives', 'Jamaica', 'Saint Lucia', 'Myanmar', 'Barbados', 'Brunei', 'Togo', 'Ghana', 'Mauritius', 'Malawi', 'Antigua and Barbuda', 'Nepal', 'Taiwan', 'Uganda', 'Bahamas', 'Vietnam', 'Eswatini', 'Saint Helena', 'Mongolia', 'Kenya', "Cote d'Ivoire", 'Moldova', 'Trinidad and Tobago', 'Uzbekistan', 'Mali', 'Botswana', 'Bangladesh', 'Falkland Islands', 'Suriname', 'Ukraine', 'Papua New Guinea', 'Afghanistan', 'Sierra Leone', 'Sudan', 'Nigeria', 'Belize', 'Grenada', 'Montserrat', 'Kosovo', 'Sri Lanka', 'Georgia', 'Bhutan', 'Saint Kitts and Nevis', 'Solomon Islands', 'Angola', 'Guyana', 'Sao Tome and Principe', 'Cape Verde', 'Dominica', 'Saint Vincent and the Grenadines', 'Anguilla'}

Pfizer/BioNTech, Sinovac:>>{'Colombia', 'Malaysia', 'Uruguay', 'Albania', 'Chile', 'Turkey', 'Hong Kong'}

Sputnik V:>>{'Guinea', 'Iran', 'Paraguay', 'Kazakhstan', 'Algeria', 'Syria', 'Belarus', 'Armenia', 'Venezuela'}

Pfizer/BioNTech:>>{'Cayman Islands', 'Greenland', 'Monaco', 'Andorra', 'Gibraltar', 'Lebanon', 'Turks and Caicos Islands', 'Panama', 'Japan', 'North Macedonia', 'Qatar', 'Cyprus', 'Bermuda', 'Slovakia', 'Ecuador', 'New Zealand', 'Costa Rica', 'Kuwait'}

Oxford/AstraZeneca, Sinopharm/Beijing, Sputnik V:>>{'Argentina', 'Pakistan', 'Bolivia'}

Oxford/AstraZeneca, Pfizer/BioNTech:>>{'Australia', 'United Kingdom', 'Jersey', 'Isle of Man', 'South Korea', 'Sweden', 'Slovenia', 'Guernsey', 'Oman', 'Saudi Arabia'}

Moderna, Oxford/AstraZeneca, Pfizer/BioNTech:>>{'Austria', 'Rwanda', 'Canada', 'Belgium', 'Lithuania', 'France', 'Germany', 'Spain', 'Latvia', 'Estonia', 'Ireland', 'Norway', 'Luxembourg', 'Netherlands', 'Italy', 'Malta', 'Czechia', 'Iceland', 'Palestine', 'Croatia', 'Denmark', 'Greece', 'Poland', 'Romania', 'Bulgaria', 'Portugal', 'Finland'}

Sinovac:>>{'Azerbaijan'}

Oxford/AstraZeneca, Pfizer/BioNTech, Sinopharm/Beijing, Sputnik V:>>{'Serbia', 'Bahrain'}

Oxford/AstraZeneca, Sinovac:>>{'Brazil', 'Dominican Republic', 'Indonesia', 'Philippines', 'Thailand'}

Oxford/AstraZeneca, Sinopharm/Beijing, Sinovac:>>{'Cambodia'}

Sinopharm/Beijing, Sinopharm/Wuhan, Sinovac:>>{'China'}

Oxford/AstraZeneca, Sinopharm/Beijing:>>{'Iraq', 'Morocco', 'Seychelles', 'Egypt'}

Oxford/AstraZeneca, Pfizer/BioNTech, Sinovac:>>{'Northern Cyprus', 'El Salvador'}

Sinopharm/Beijing:>>{'Equatorial Guinea', 'Mauritania', 'Senegal', 'Zimbabwe', 'Gabon', 'Kyrgyzstan', 'Namibia', 'Mozambique'}

Moderna, Pfizer/BioNTech:>>{'Liechtenstein', 'Faeroe Islands', 'Israel', 'Switzerland', 'Singapore'}

Moderna, Oxford/AstraZeneca:>>{'Honduras', 'Guatemala'}

Moderna, Oxford/AstraZeneca, Pfizer/BioNTech, Sinopharm/Beijing, Sputnik V:>>{'Hungary'}

Covaxin, Oxford/AstraZeneca:>>{'India'}

Pfizer/BioNTech, Sinopharm/Beijing:>>{'Peru', 'Macao', 'Jordan'}

Sinopharm/Beijing, Sputnik V:>>{'Montenegro', 'Laos'}

Oxford/AstraZeneca, Pfizer/BioNTech, Sinovac, Sputnik V:>>{'Mexico'}

EpiVacCorona, Sputnik V:>>{'Russia'}

Pfizer/BioNTech, Sputnik V:>>{'San Marino'}

Johnson&Johnson:>>{'South Africa'}

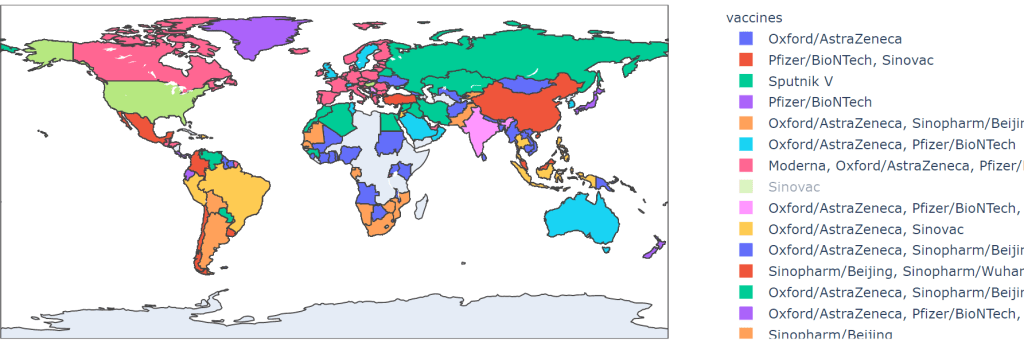
Pfizer/BioNTech, Sinovac, Sputnik V:>>{'Tunisia'}

Oxford/AstraZeneca, Pfizer/BioNTech, Sinopharm/Beijing, Sinopharm/Wuhan, Sputnik V:>>{'United Arab Emirates'}

Johnson&Johnson, Moderna, Pfizer/BioNTech:>>{'United States'}

Now let’s visualize this data to have a look at what combination of vaccines every country is using:

|  |  |
| --- | --- |
|  | import plotly.express as px |
|  | import plotly.offline as py |
|  |  |
|  | vaccine\_map = px.choropleth(data, locations = 'iso\_code', color = 'vaccines') |
|  | vaccine\_map.update\_layout(height=300, margin={"r":0,"t":0,"l":0,"b":0}) |
|  | vaccine\_map.show() |



**Conclusion**:

In this study, visualize and debate the current state of COVID-19 vaccination in terms of the proportion of top 10 vaccines in the race to combat COVID-19, the number of cumulative vaccinations and every day vaccinations as per the countries, cumulative vaccinations per country grouped by vaccines, daily vaccinations per countries, and the relationship among cumulative vaccinations and cumulative vaccinations per hundred of the top five countries seriously affected by the COVID-19 globally as of May 24, 2021, including the United States, India, Brazil, France, and Turkey. The statistics reveal that Oxford/AstraZeneca is the top vaccine used across the globe with 26.54%, the United States is the top in vaccination, with 277,290,173, India is the top in number of daily vaccinations with 3.659357M, and in total vaccinations per hundred people, the United States has the highest count with 82.91, among the top five countries. It is also anticipated that the vaccination rate in the United States will reach almost 60%, while India, Brazil, France, and Turkey will reach about 15%, 28%, 60%, and 23%, respectively, in the following 50 days beginning 20 May 2021. However, this will not be enough to save the public, and policymakers in India, Brazil, and Turkey should take the necessary steps.