**COVID Vaccines Analysis**

Phase 2 Submission Document

**Project Name: Covid Vaccines Analysis**

732121205001:Abinav.T

**Introduction**

* In the covid vaccine analysis project, in the face of the unprecedented global health crisis posed by the COVID-19 pandemic, the rapid development and deployment of vaccines have emerged as a beacon of hope and a monumental achievement in the field of public health.
* The vaccination efforts, spanning across continents and cultures, represent a collective stride towards curbing the spread of the virus and ultimately, saving lives. As the world grapples with the challenges of vaccinating entire populations, there arises a critical need to analyze the vast and intricate data surrounding these vaccination campaigns.
* This data analytics project delves into the heart of this complex scenario, aiming to dissect the wealth of information available pertaining to COVID-19 vaccines. By employing advanced data analytics techniques, this project seeks to unravel patterns, trends, and insights within the data.
* Through rigorous analysis, we intend to shed light on various aspects of the COVID-19 vaccination process, such as efficacy rates, distribution strategies, public sentiment, and the impact of vaccinations on mitigating the disease's spread.
* The analysis encompasses diverse datasets, including vaccination rates, demographic information, regional disparities, public perception data from social media platforms, and more. By examining this multifaceted data, our objective is to contribute valuable insights that can inform public health policies, optimize vaccine distribution strategies, and aid healthcare professionals, policymakers, and researchers in making informed decisions.
* Through the lens of data analytics, this project serves as a torchbearer, illuminating the path toward a more comprehensive understanding of COVID-19 vaccinations. The findings generated herein are not merely statistical interpretations; they are actionable insights that can foster better decision-making, improve vaccination strategies, and ultimately contribute to the global effort in overcoming the pandemic's challenges.

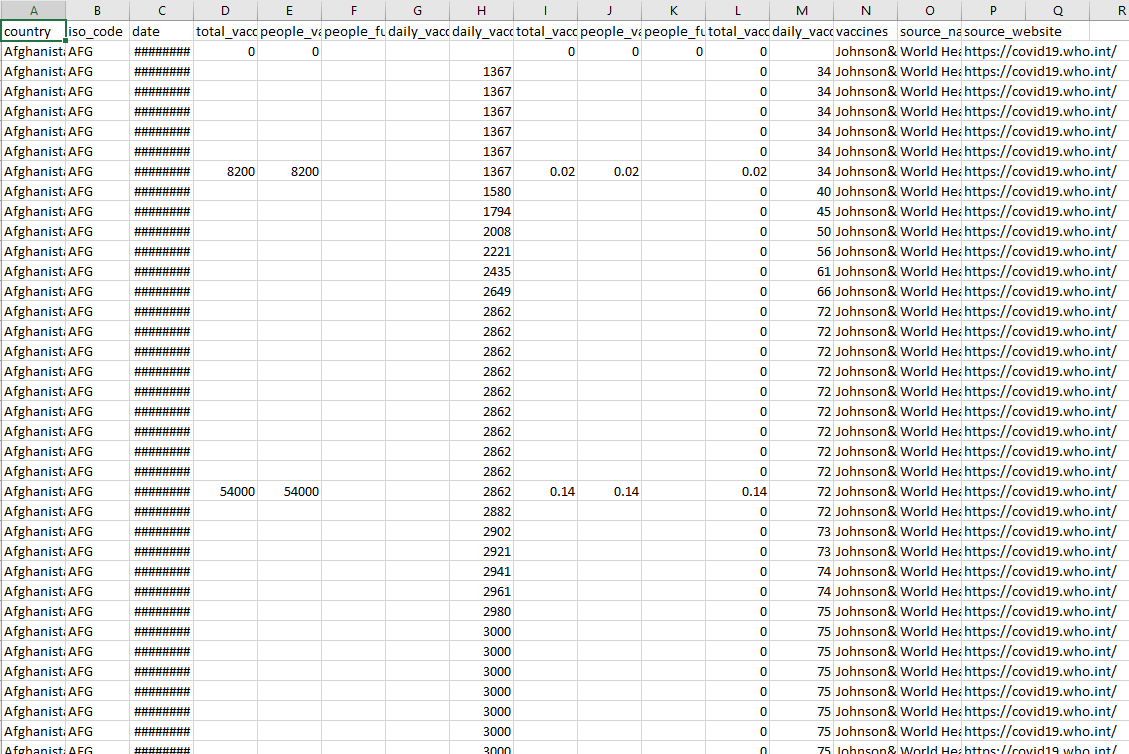
**Content for Project Phase 2**

Consider exploring advanced machine learning techniques like clustering or time series forecasting to uncover hidden patterns in vaccine distribution and adverse effects data.

**Data Source**

A good data source for covid vaccine analysis using machine learning should be Accurate , Complete , Covering the geographic area of intrest , Accessible.

**Dataset Link:** ( <https://www.kaggle.com/datasets/gpreda/covid-world-vaccination-progress> )



**Data Collection:**

* Gather data from reliable sources such as government health departments, research institutions, or public health organizations.
* Collect data on various aspects, including vaccination rates, demographic information, vaccine types, regional data, and relevant socioeconomic factors.

**Data Cleaning:**

* Handling Missing Data: Identify and handle missing data using techniques like imputation (replacing missing values with statistical estimates) or removing incomplete records.
* Removing Duplicates: Check for and remove duplicate entries in the dataset.
* Ensuring Consistency: Standardize data formats, units, and variables to ensure consistency across the dataset.

**Data Transformation:**

* Feature Engineering: Create new features from existing ones if necessary. For example, you might calculate vaccination rates per capita or create age groups from age data.
* Normalization/Standardization: Normalize or standardize numerical variables to bring them to a standard scale, especially if you're using algorithms sensitive to the scale of variables (e.g., gradient descent in machine learning).
* One-Hot Encoding: Convert categorical variables into binary vectors (0 and 1) to make them suitable for machine learning algorithms.

**Data Integration:**

* Merge Datasets: If your data is spread across multiple sources, merge them using common keys (like location or date) to create a unified dataset.
* Temporal Alignment: Ensure that data points from different sources are aligned in time for accurate analysis.

**Data Reduction:**

* Dimensionality Reduction: If your dataset has a large number of features, consider techniques like Principal Component Analysis (PCA) to reduce dimensionality while preserving important information.
* Sampling: If dealing with a massive dataset, consider using random sampling techniques to work with a manageable subset for initial analysis.

**Data Validation:**

* Outlier Detection: Identify and handle outliers that could skew your analysis. Decide whether to remove, transform, or keep these outliers based on domain knowledge.
* Cross-Validation: Split the data into training and validation sets to validate the performance of your models effectively.

**Documentation:**

* Metadata: Document the meaning and source of each variable. Maintain a data dictionary for easy reference.
* Data Processing Steps: Document all the steps you took for data cleaning, transformation, and integration. This documentation is crucial for reproducibility.

**Version Control:**

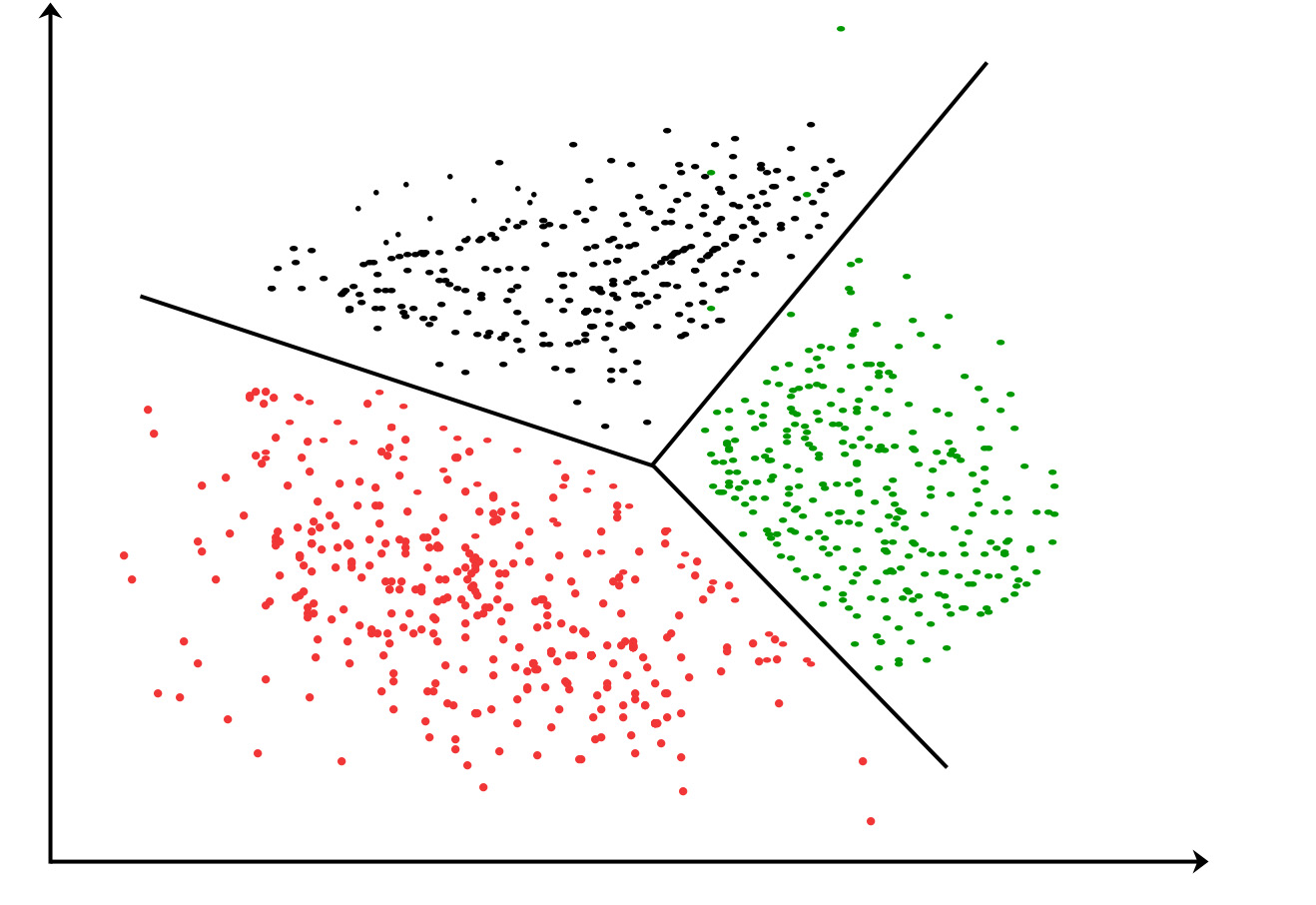
* Use version control systems (like Git) to keep track of changes made to the dataset and analysis code. This ensures traceability and reproducibility.

**Advance Machine Learning Technique**

**Clustering**

Clustering is the task of dividing the population or data points into a number of groups such that data points in the same groups are more similar to other data points in the same group and dissimilar to the data points in other groups. It is basically a collection of objects on the basis of similarity and dissimilarity between them.

For example: The data points in the graph below clustered together can be classified into one single group. We can distinguish the clusters, and we can identify that there are 3 clusters in the below picture.



**Time Series Forecasting**

Time series forecasting is a technique used to predict future data points based on past observations. It is widely used in various fields such as finance, economics, sales forecasting, and weather forecasting. There are several methods for time series forecasting, and one of the most commonly used techniques is the Autoregressive Integrated Moving Average (ARIMA) model. Here's how you can perform time series forecasting using Python and the ARIMA model

Step 1: Import Libraries

Step 2: Load and Explore Data

Step 3: Preprocess Data

Step 4: Visualize the Time Series

Step 5: Stationarize the Time Series

Step 6: Build and Train the ARIMA Model

Step 7: Make Predictions

**Feature engineering**

* Date Time Features: these are components of the time step itself for each observation.'Series' refers to each time step(you will see that in the next cell)'Target' refers to the target value at the current time step.
* Lag Features: these are values at prior time steps.'Shift1' refers to the target value at the previous time step.
* Window Features: these are a summary of values over a fixed window of prior time steps.

**Program:**

Covid Vacines Analysis

Importing the libraries

In [1]

import numpy as np

import pandas as pd

import plotly.graph\_objects as go

import plotly.express as px

import plotly.offline as pyo

from plotly.subplots import make\_subplots

from datetime import date , datetime , timedelta

import pycaret.regression as caret

import warnings

warnings.filterwarnings('ignore')

In [2]

data\_detailed = pd.read\_csv('../input/covid-world-vaccination-progress/country\_vaccinations.csv')

data\_total = pd.read\_csv('../input/covid-world-vaccination-progress/country\_vaccinations\_by\_manufacturer.csv')

In [3]

print("\* "\*10+" data\_detailed "+" \*"\*10)

print("\nShape: rows = {} , columns = {}".format(data\_detailed.shape[0] , data\_detailed.shape[1]))

print(data\_detailed.info())

print("\* "\*10+" data\_total "+" \*"\*10)

print("\nShape: rows = {} , columns = {}".format(data\_total.shape[0] , data\_total.shape[1]))

print(data\_total.info())

Out [3]

\* \* \* \* \* \* \* \* \* \* data\_detailed \* \* \* \* \* \* \* \* \* \*

Shape: rows = 52739 , columns = 15

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 52739 entries, 0 to 52738

Data columns (total 15 columns):

# Column Non-Null Count Dtype

--- ------ -------------- -----

0 country 52739 non-null object

1 iso\_code 52739 non-null object

2 date 52739 non-null object

3 total\_vaccinations 28494 non-null float64

4 people\_vaccinated 27164 non-null float64

5 people\_fully\_vaccinated 24204 non-null float64

6 daily\_vaccinations\_raw 23311 non-null float64

7 daily\_vaccinations 52477 non-null float64

8 total\_vaccinations\_per\_hundred 28494 non-null float64

9 people\_vaccinated\_per\_hundred 27164 non-null float64

10 people\_fully\_vaccinated\_per\_hundred 24204 non-null float64

11 daily\_vaccinations\_per\_million 52477 non-null float64

12 vaccines 52739 non-null object

13 source\_name 52739 non-null object

14 source\_website 52739 non-null object

dtypes: float64(9), object(6)

memory usage: 6.0+ MB

None

\* \* \* \* \* \* \* \* \* \* data\_total \* \* \* \* \* \* \* \* \* \*

Shape: rows = 18954 , columns = 4

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 18954 entries, 0 to 18953

Data columns (total 4 columns):

# Column Non-Null Count Dtype

--- ------ -------------- -----

0 location 18954 non-null object

1 date 18954 non-null object

2 vaccine 18954 non-null object

3 total\_vaccinations 18954 non-null int64

dtypes: int64(1), object(3)

memory usage: 592.4+ KB

None

**Total Vaccination for each country**

In [4]

dict\_total\_vac = {}

for iso\_code in iso\_list:

dict\_total\_vac[iso\_code]=data\_detailed[data\_detailed.iso\_code==iso\_code]['total\_vaccinations'].max()

df\_total\_vac = pd.DataFrame()

df\_total\_vac['iso\_code'] = dict\_total\_vac.keys()

df\_total\_vac['total vaccinations'] = dict\_total\_vac.values()

df\_total\_vac['country'] = countries

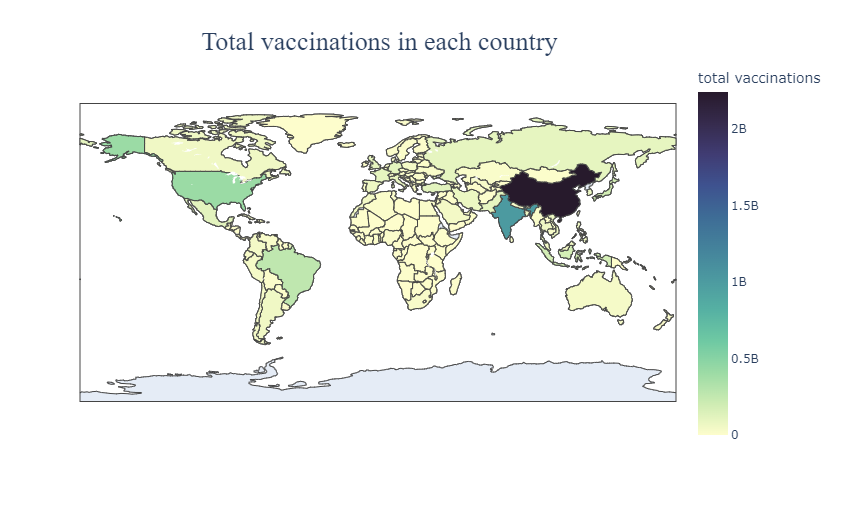
In [5]

map\_total\_vac = px.choropleth(data\_frame = df\_total\_vac , locations="iso\_code" , color="total vaccinations" , hover\_name="country" , color\_continuous\_scale=px.colors.sequential.deep)

map\_total\_vac.update\_layout(title\_text='Total vaccinations in each country' , title\_font={'family':'serif','size':26} , title = {'y':0.94 , 'x':0.}

map\_total\_vac.show()

Out [5]

**Fully vaccinated people percentage**

In [6]

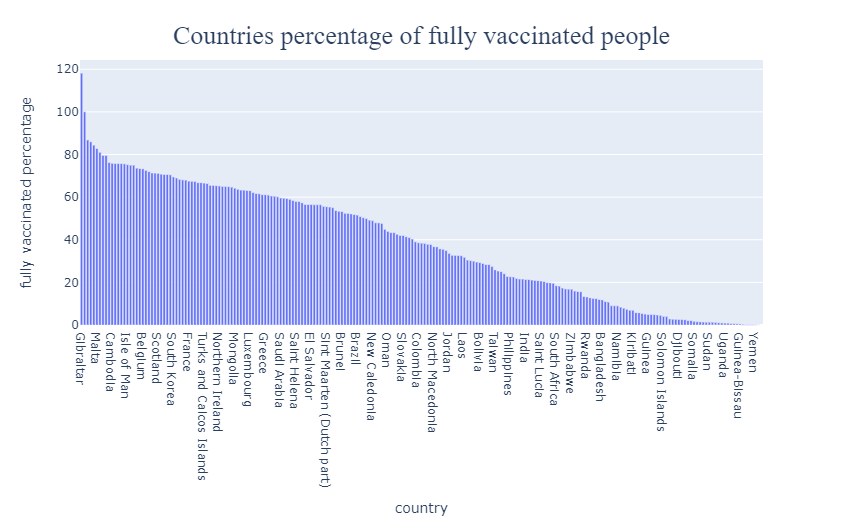
plot\_font = {'family':'serif' , 'size':26}

bar\_full\_percentage = px.bar(data\_frame = df\_vac\_percentages.sort\_values(by='fully vaccinated percentage',ascending = False), x = 'country', y = 'fully vaccinated percentage')

bar\_full\_percentage.update\_layout(title\_text='Countries percentage of fully vaccinated people' ,title\_font={'family':'serif','size':26} , title = {'x':0.5 , 'y':0.95})

bar\_full\_percentage.show()

Out [6]



**Pycaret setup**

In [7]

setup = caret.setup(data = train , test\_data = test , target = 'Target' , fold\_strategy = 'timeseries'

, remove\_perfect\_collinearity = False , numeric\_features = ['Series' , 'Window\_mean' , 'Shift1'] , fold = 5 , session\_id = 51)

Out [7]

|  |  |  |
| --- | --- | --- |
|  | Description | Value |
| 0 | session\_id | 51 |
| 1 | Target | Target |
| 2 | Original Data | (230, 4) |
| 3 | Missing Values | False |
| 4 | Numeric Features | 3 |
| 5 | Categorical Features | 0 |
| 6 | Ordinal Features | False |
| 7 | High Cardinality Features | False |
| 8 | High Cardinality Method | None |
| 9 | Transformed Train Set | (230, 3) |
| 10 | Transformed Test Set | (59, 3) |
| 11 | Shuffle Train-Test | True |
| 12 | Stratify Train-Test | False |
| 13 | Fold Generator | TimeSeriesSplit |
| 14 | Fold Number | 5 |
| 15 | CPU Jobs | -1 |
| 16 | Use GPU | False |
| 17 | Log Experiment | False |
| 18 | Experiment Name | reg-default-name |
| 19 | USI | c8c2 |
| 20 | Imputation Type | simple |
| 21 | Iterative Imputation Iteration | None |
| 22 | Numeric Imputer | mean |
| 23 | Iterative Imputation Numeric Model | None |
| 24 | Categorical Imputer | constant |
| 25 | Iterative Imputation Categorical Model | None |
| 26 | Unknown Categoricals Handling | least\_frequent |
| 27 | Normalize | False |
| 28 | Normalize Method | None |
| 29 | Transformation | False |
| 30 | Transformation Method | None |
| 31 | PCA | False |
| 32 | PCA Method | None |
| 33 | PCA Components | None |
| 34 | Ignore Low Variance | False |
| 35 | Combine Rare Levels | False |
| 36 | Rare Level Threshold | None |
| 37 | Numeric Binning | False |
| 38 | Remove Outliers | False |
| 39 | Outliers Threshold | None |
| 40 | Remove Multicollinearity | False |
| 41 | Multicollinearity Threshold | None |
| 42 | Remove Perfect Collinearity | False |
| 43 | Clustering | False |
| 44 | Clustering Iteration | None |
| 45 | Polynomial Features | False |
| 46 | Polynomial Degree | None |
| 47 | Trignometry Features | False |
| 48 | Polynomial Threshold | None |
| 49 | Group Features | False |
| 50 | Feature Selection | False |
| 51 | Feature Selection Method | classic |
| 52 | Features Selection Threshold | None |
| 53 | Feature Interaction | False |
| 54 | Feature Ratio | False |
| 55 | Interaction Threshold | None |
| 56 | Transform Target | False |
| 57 | Transform Target Method | box-cox |

**Compare models(based on 5-fold cross validation)**

In[8]

best = caret.compare\_models(sort = 'MAE' , turbo = False)

Out [8]

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Model | MAE | MSE | RMSE | R2 | RMSLE | MAPE | TT (Sec) |  |
| ard | Automatic Relevance Determination | 0.1330 | 0.0391 | 0.1638 | 0.9974 | 0.0083 | 0.0079 | 0.0160 |
| huber | Huber Regressor | 0.1387 | 0.0473 | 0.1774 | 0.9975 | 0.0082 | 0.0070 | 0.0200 |
| tr | TheilSen Regressor | 0.1518 | 0.0488 | 0.1880 | 0.9961 | 0.0106 | 0.0103 | 0.3880 |
| lr | Linear Regression | 0.1543 | 0.0573 | 0.1918 | 0.9972 | 0.0085 | 0.0076 | 0.8340 |
| ransac | Random Sample Consensus | 0.1543 | 0.0573 | 0.1918 | 0.9972 | 0.0085 | 0.0076 | 0.0160 |
| lar | Least Angle Regression | 0.1544 | 0.0574 | 0.1919 | 0.9972 | 0.0086 | 0.0076 | 0.0160 |
| br | Bayesian Ridge | 0.1574 | 0.0599 | 0.1950 | 0.9972 | 0.0085 | 0.0076 | 0.0140 |
| omp | Orthogonal Matching Pursuit | 0.2642 | 0.1393 | 0.3004 | 0.9908 | 0.0130 | 0.0133 | 0.0140 |
| kr | Kernel Ridge | 0.3424 | 0.2959 | 0.3990 | 0.9850 | 0.0203 | 0.0201 | 0.0180 |
| ridge | Ridge Regression | 0.5018 | 0.6530 | 0.5850 | 0.9598 | 0.0310 | 0.0301 | 0.0140 |
| mlp | MLP Regressor | 0.9282 | 2.1638 | 1.0616 | 0.8145 | 0.0676 | 0.0670 | 0.0700 |
| par | Passive Aggressive Regressor | 1.3335 | 2.7623 | 1.4577 | 0.1856 | 0.1316 | 0.1375 | 0.0140 |
| en | Elastic Net | 1.8992 | 9.6416 | 2.1711 | 0.1714 | 0.1677 | 0.1516 | 0.0160 |
| lasso | Lasso Regression | 2.0644 | 10.9070 | 2.3399 | 0.0389 | 0.1829 | 0.1674 | 0.0140 |
| dt | Decision Tree Regressor | 5.8230 | 66.1052 | 6.6985 | -3.1308 | 0.3771 | 0.2988 | 0.0160 |
| et | Extra Trees Regressor | 5.8230 | 66.1052 | 6.6985 | -3.1308 | 0.3771 | 0.2988 | 0.3040 |
| xgboost | Extreme Gradient Boosting | 5.8261 | 66.1537 | 6.7013 | -3.1351 | 0.3773 | 0.2990 | 6.4400 |
| gbr | Gradient Boosting Regressor | 5.8274 | 66.1796 | 6.7024 | -3.1348 | 0.3773 | 0.2989 | 0.0300 |
| catboost | CatBoost Regressor | 5.8349 | 66.3010 | 6.7090 | -3.1436 | 0.3777 | 0.2993 | 0.6080 |
| rf | Random Forest Regressor | 6.0007 | 69.2343 | 6.8559 | -3.3390 | 0.3864 | 0.3073 | 0.3220 |
| knn | K Neighbors Regressor | 6.4526 | 77.7214 | 7.2599 | -3.8692 | 0.4108 | 0.3291 | 0.0480 |
| ada | AdaBoost Regressor | 6.5078 | 78.4229 | 7.3126 | -3.9085 | 0.4052 | 0.3239 | 0.0460 |
| lightgbm | Light Gradient Boosting Machine | 8.4600 | 123.7536 | 9.1152 | -7.0442 | 0.5487 | 0.4392 | 0.2100 |
| svm | Support Vector Regression | 9.9530 | 185.2220 | 10.6861 | -9.3845 | 0.5583 | 0.4143 | 0.0160 |
| llar | Lasso Least Angle Regression | 17.7073 | 561.2682 | 18.0987 | -33.8493 | 1.2507 | 0.7704 | 0.0140 |

**Future forecasting**

In [9]

future = pd.DataFrame(columns = ['Series' , 'Window\_mean' , 'Shift1'])

future['Series'] = np.arange(300,450) # for the next 150 time steps

future['Window\_mean'] = np.nan

future['Shift1'] = np.nan

future.iloc[0,2] = data['Target'].max()

sum = 0

for i in range(window\_len): sum += data.iloc[len(data)-1-i,3]

future.iloc[0,1] = sum/window\_len

future

Out [9]

|  |  |  |  |
| --- | --- | --- | --- |
|  | Series | Window\_mean | Shift1 |
| 0 | 300 | 65.198 | 65.53 |
| 1 | 301 | NaN | NaN |
| 2 | 302 | NaN | NaN |
| 3 | 303 | NaN | NaN |
| 4 | 304 | NaN | NaN |
| ... | ... | ... | ... |
| 145 | 445 | NaN | NaN |
| 146 | 446 | NaN | NaN |
| 147 | 447 | NaN | NaN |
| 148 | 448 | NaN | NaN |
| 149 | 449 | NaN |  |

In[10]

for j in range(len(future)):

current\_row = j next\_row = j+1

if current\_row != len(future)-1 :

# fill Shift1 for the next\_row

future.iloc[next\_row,2] = caret.predict\_model(best , future.iloc[[current\_row]])['Label']

print(future.iloc[next\_row,2]-future.iloc[current\_row,2])

if next\_row < 9 :

sum = 0

num\_rows\_from\_data = window\_len - (next\_row + 1)

num\_rows\_from\_future = window\_len - num\_rows\_from\_data

for i in range(num\_rows\_from\_data):

sum += data.iloc[len(data)-1-i , 2]

for i in range(num\_rows\_from\_future):

sum += future.iloc[next\_row - i , 2]

future.iloc[next\_row , 1] = sum/window\_len

elif next\_row >= 9:

sum = 0

for i in range(window\_len):

sum += future.iloc[next\_row-i,2]

future.iloc[next\_row,1] = sum/window\_len

Out [10]

|  |  |  |  |
| --- | --- | --- | --- |
|  | Series | Window\_mean | Shift1 |
| 0 | 300 | 65.198000 | 65.530000 |
| 1 | 301 | 65.274490 | 65.614897 |
| 2 | 302 | 65.348673 | 65.701837 |
| 3 | 303 | 65.420844 | 65.791710 |
| 4 | 304 | 65.493398 | 65.885537 |
| ... | ... | ... | ... |
| 145 | 445 | 99.155322 | 100.454076 |
| 146 | 446 | 99.444361 | 100.740679 |
| 147 | 447 | 99.732866 | 101.026701 |
| 148 | 448 | 100.020829 | 101.312143 |
| 149 | 449 | 100.308240 | 101.596989 |

In [11]

fig = go.Figure(data=go.Scatter(x=df\_germany\_info['date'], y = df\_germany\_info['people\_fully\_vaccinated\_per\_hundred']

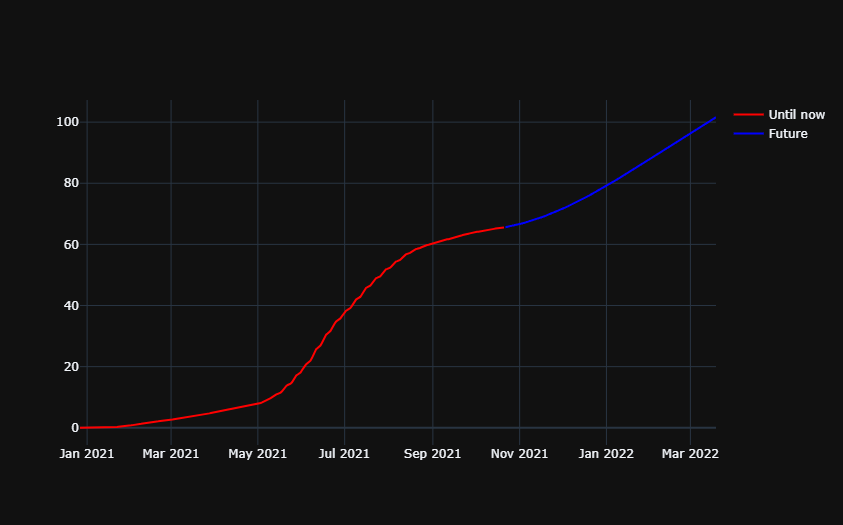
,mode='lines', line\_color='red' , name = 'Until now'))

fig.add\_trace(go.Scatter(x=future['Date'], y=future['Predicted'],mode='lines', line=dict(color="#0000ff"), name = 'Future'))

fig.update\_layout(template = 'plotly\_dark')

fig.show()

Out [11]



Conclusion

In conclusion, the provided Python code demonstrates a comprehensive approach to time series forecasting using the ARIMA model on a COVID vaccine analysis dataset. Here's a summary of the key steps and outcomes of the code:

* Data Loading: Replace 'your\_dataset.csv' with the actual path to your COVID vaccine analysis dataset. Ensure that the dataset contains columns for the date (date) and the number of vaccine doses (vaccine\_doses).
* Stationarity Check: The code checks the stationarity of the time series using the Dickey-Fuller test. If the series is not stationary, differencing is performed.
* Model Building: The ARIMA model is built and trained using differenced data. The order parameter of ARIMA is set to (5, 1, 0) in this example. You may need to tune these parameters based on your dataset and domain knowledge.
* Forecasting: Future values are forecasted, and lower and upper confidence intervals are calculated.
* Visualization: The observed time series, forecast, and 95% confidence interval are plotted.
* Model Evaluation: The Root Mean Squared Error (RMSE) is calculated to evaluate the model's accuracy.