

Project title: **Covid-19 Vaccines Analysis**

Phase 3: **Development**

Part 1

In this part you will begin building your project by loading and preprocessing the dataset.

Begin conducting the Covid-19 vaccines analysis by collecting and preprocessing the data.

Collect and preprocess the Covid-19 vaccine data for analysis.

Data Preprocessing:

Data preprocessing is a crucial step within the statistics analysis and gadget gaining knowledge of pipeline.

It includes a sequence of strategies and operations finished on uncooked statistics to clean, organize, and transform it right into a layout that is suitable for analysis or device mastering version schooling.

Data preprocessing goals to enhance the first-class of the records, making it greater reliable and conducive to generating accurate consequences.

Here are some common tasks and techniques involved in data preprocessing:

Data Cleaning:

Handling missing values: Deciding how to deal with missing data, whether by imputing values or removing incomplete records.

Outlier detection and treatment: Identifying and handling data points that significantly deviate from the norm.

Noise reduction:

Smoothing noisy data through techniques like filtering.

Data Transformation:

Data normalization: Scaling numerical features to a standard range (e.g., between 0 and 1) to ensure that they have similar influence in the analysis. Encoding categorical variables: Converting categorical data into numerical format, such as one-hot encoding or label encoding.

Feature engineering: Creating new features or modifying existing ones to capture more meaningful information from the data.

Dimensionality reduction: Reducing the number of features while retaining essential information, using methods like Principal Component Analysis (PCA).

Data Integration:

Merging or joining datasets: Combining data from multiple sources into a single dataset for analysis.

Aggregation: Summarizing data at a higher level of granularity, such as aggregating daily sales into monthly totals.

Data Reduction:

Sampling: Reducing the size of a large dataset by randomly selecting a representative subset.

Binning: Grouping continuous data into discrete bins to simplify analysis.

Filtering: Selecting a subset of data based on specific criteria.

Data Standardization:

Ensuring that data follows a consistent format and structure.

Date and time format conversion: Converting date and time data into a uniform format.

Currency conversion: Converting monetary values into a common currency.

Data Scaling:

Scaling numerical data to a common range to prevent some features from dominating the analysis.

Data preprocessing is an iterative process that may involve several of these steps in various orders, depending on the specific dataset and the analysis goals. Proper data preprocessing is essential for improving the accuracy and effectiveness of machine learning models, as well as for making data more accessible for traditional statistical analysis.

Here is the data preprocessing codes along with the output of the given dataset:

Importing the libraries:

Import three basic libraries which are very common in machine learning and will be used every time you train a model

NumPy: it is a library that allows us to work with arrays and as most machine learning models work on arrays NumPy makes it easier

matplotlib: this library helps in plotting graphs and charts, which are very useful while showing the result of your model

Pandas: pandas allows us to import our dataset and also creates a matrix of features containing the dependent and independent variable.

Code:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

Output:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
✓ 0.0s
```

Load the dataset: (DATASET 1)

Data sets are available in .csv format. A CSV file stores tabular data in plain text. Each line of the file is a data record. We use the read_csv method of the pandas library to read a local CSV file as a dataframe.

Load our customer data from the CSV file

Code:

```
import pandas as pd
# Try reading the file with different encodings
encodings = ['utf-8', 'latin1', 'ISO-8859-1']
for encoding in encodings:
    try:
        dataset =
pd.read_csv(r'C:\\Users\\KISHORE\\OneDrive\\Documents\\country_vaccinatio
ns.csv', encoding=encoding)
print(f'Successfully read with encoding: {encoding}')
```

```
break # If successful, no need to try other encodingsexcept
UnicodeDecodeError:
print(f'Failed to read with encoding: {encoding}')
# Now 'dataset' should contain your data
```

Output:

```
import pandas as pd

# Try reading the file with different encodings
encodings = ['utf-8', 'latin1', 'ISO-8859-1']

for encoding in encodings:
    try:
        dataset = pd.read_csv(r'C:\\Users\\KISHORE\\OneDrive\\Documents\\country_vaccinations.csv', encoding=encoding)
        print(f"Successfully read with encoding: {encoding}")
        break # If successful, no need to try other encodings
    except UnicodeDecodeError:
        print(f"Failed to read with encoding: {encoding}")

# Now 'dataset' should contain your data
✓ 0.1s
Successfully read with encoding: utf-8
```

Head() Function:

The head() function is used to get the first n rows.

This function returns the first n rows for the object based on position.

It is useful for quickly testing if your object has the right type of data in it. If the value of the n is not assigned it returns a default value of first 5 rows

Code:

dataset.head()

Output:

dataset.head						
✓ 0.0s						
<bound method NDFrame.head of				country iso_code	date	total_vaccinations \
0	Afghanistan	AFG	22-02-2021	0.0		
1	Afghanistan	AFG	23-02-2021	NaN		
2	Afghanistan	AFG	24-02-2021	NaN		
3	Afghanistan	AFG	25-02-2021	NaN		
4	Afghanistan	AFG	26-02-2021	NaN		
...		
86507	Zimbabwe	ZWE	25-03-2022	8691642.0		
86508	Zimbabwe	ZWE	26-03-2022	8791728.0		
86509	Zimbabwe	ZWE	27-03-2022	8845039.0		
86510	Zimbabwe	ZWE	28-03-2022	8934360.0		
86511	Zimbabwe	ZWE	29-03-2022	9039729.0		
				people_vaccinated	people_fully_vaccinated	daily_vaccinations_raw \
0				0.0	NaN	NaN
1				NaN	NaN	NaN
2				NaN	NaN	NaN
3				NaN	NaN	NaN
4				NaN	NaN	NaN

Info() Function:

The info() method prints information about the DataFrame.
The information contains the number of columns, column labels, column data types, memory usage, range index, and the number of cells in each column (non-nullvalues).

Code:

```
dataset.info()
```

Output:

```
dataset.info()
✓ 0.0s

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 86512 entries, 0 to 86511
Data columns (total 15 columns):
#   Column                                Non-Null Count  Dtype
---  -
0   country                             86512 non-null  object
1   iso_code                            86512 non-null  object
2   date                                86512 non-null  object
3   total_vaccinations                  43607 non-null  float64
4   people_vaccinated                   41294 non-null  float64
5   people_fully_vaccinated              38802 non-null  float64
6   daily_vaccinations_raw              35362 non-null  float64
7   daily_vaccinations                  86213 non-null  float64
8   total_vaccinations_per_hundred      43607 non-null  float64
9   people_vaccinated_per_hundred       41294 non-null  float64
10  people_fully_vaccinated_per_hundred  38802 non-null  float64
```

Df.isnull().sum() Function:

This code is used to count the number of missing (null) values in each column of a DataFrame, denoted as df.

It returns a summary of the missing data for each column, showing how many missing values are there in each column.

This information is essential in data preprocessing and analysis to identify and handle missing data appropriately.

Code:

```
dataset.isnull().sum()
```

Output:

```
dataset.isnull().sum()
```

✓ 0.0s

```
country          0
iso_code         0
date            0
total_vaccinations  42905
people_vaccinated  45218
people_fully_vaccinated  47710
daily_vaccinations_raw  51150
daily_vaccinations    299
total_vaccinations_per_hundred  42905
people_vaccinated_per_hundred  45218
people_fully_vaccinated_per_hundred  47710
daily_vaccinations_per_million    299
vaccines          0
source_name       0
source_website    0
dtype: int64
```

Describe Function:

The describe() function in pandas, a popular Python data analysis library, is used to generate summary statistics of a DataFrame or Series.

It provides a quick overview of the key statistics for numerical data in the dataset, including:

Count: The number of non-null values.

Mean: The average of the values.

Standard Deviation (std): A measure of the spread or dispersion of the data.

Minimum: The minimum value in the dataset.

25th Percentile (25%): The value below which 25% of the data falls (the first quartile).

Median (50% or the 2nd quartile): The middle value when the data is sorted.

75th Percentile (75%): The value below which 75% of the data falls (the third quartile).

Maximum: The maximum value in the dataset.

Code:

```
import matplotlib.pyplot as plt  
# Ensure your dataset contains only numerical data for box plotting  
numerical_data = dataset.select_dtypes(include='number')  
# Transpose the data to prepare for box plotting  
data_to_plot = numerical_data.values.T  
# Create subplots  
fig, axs = plt.subplots(9, 1, dpi=95, figsize=(7, 17))
```




```
# Iterate through columns and create boxplots for
i, col in enumerate(numerical_data.columns):

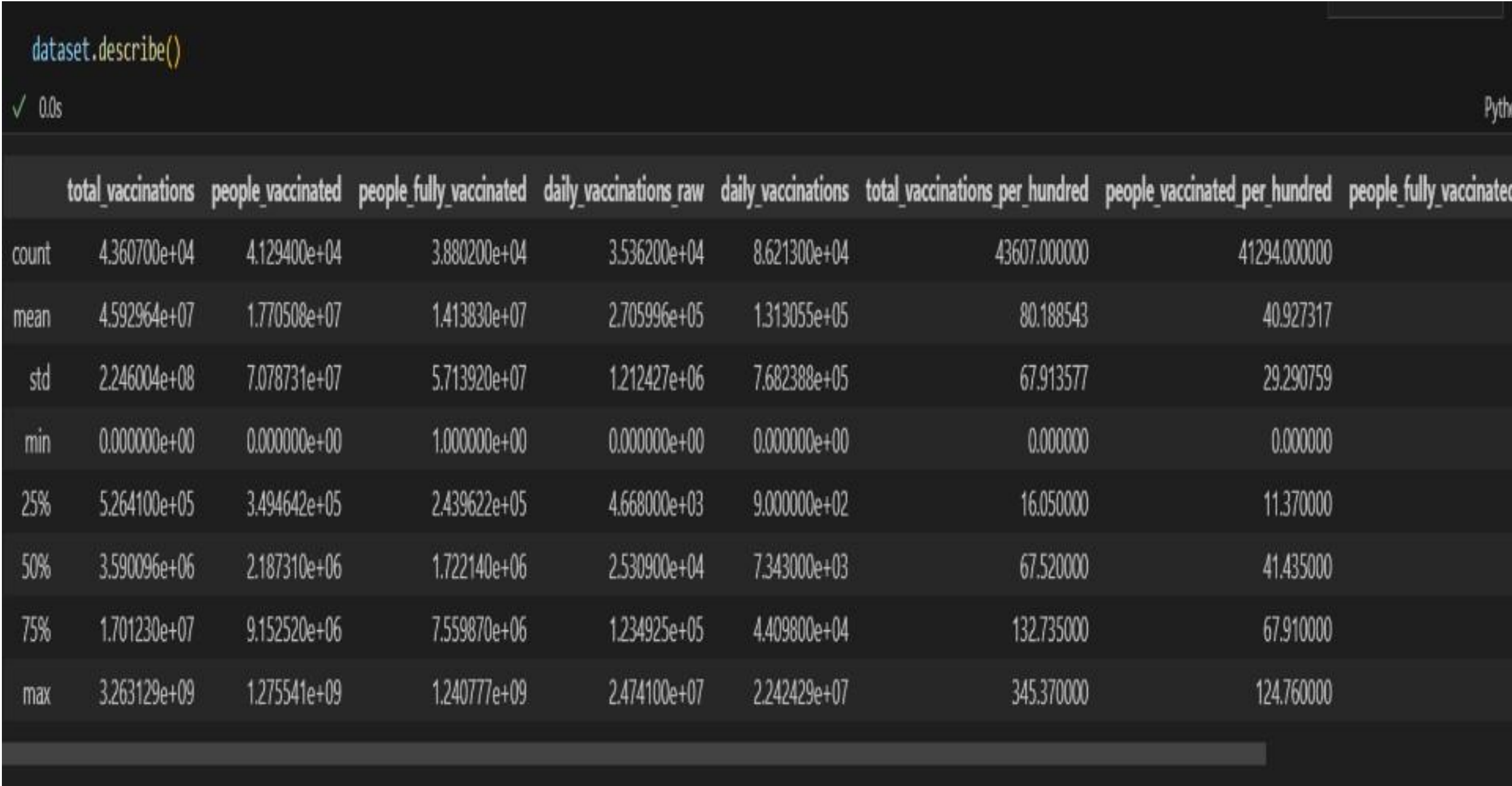
    axs[i].boxplot(data_to_plot[i], vert=False)

    axs[i].set_ylabel(col)

plt.show()
```

```
dataset.describe()
```

Output:



Outliers:

Outliers are data points that significantly deviate from the rest of the data in adataset. They can be exceptionally high or low values compared to the majority of the data.

Code:

Output:

```
import matplotlib.pyplot as plt

# Ensure your dataset contains only numerical data for box plotting
numerical_data = dataset.select_dtypes(include='number')

# Transpose the data to prepare for box plotting
data_to_plot = numerical_data.values.T

# Create subplots
fig, axs = plt.subplots(9, 1, dpi=95, figsize=(7, 17))

# Iterate through columns and create boxplots
for i, col in enumerate(numerical_data.columns):
    |   axs[i].boxplot(data_to_plot[i], vert=False)
    |   axs[i].set_ylabel(col)

plt.show()
```



Corelation:

Correlation is a statistical measure that indicates the extent to which two or more variables fluctuate in relation to each other.

Correlation describes the relationship between variables. It can be described as either strong or weak, and as either positive or negative.

Code:

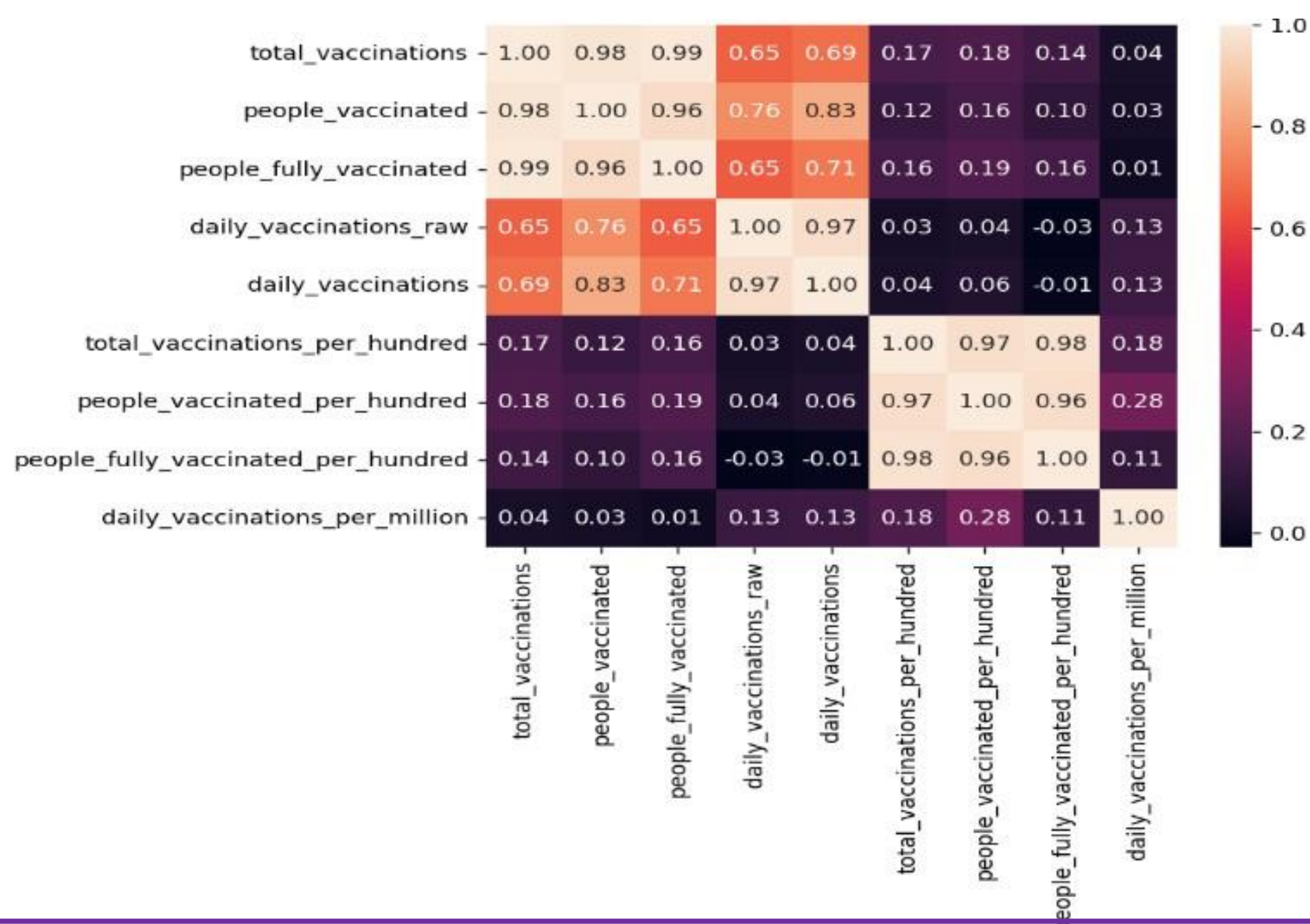
```
numeric_dataset = dataset.select_dtypes(include=['number'])
corr = numeric_dataset.corr()
import matplotlib.pyplot as plt
import seaborn as sns
plt.figure(dpi=130)
sns.heatmap(corr, annot=True, fmt='.2f')
plt.show()
```

Output:

```
numeric_dataset = dataset.select_dtypes(include=['number'])
corr = numeric_dataset.corr()
import matplotlib.pyplot as plt
import seaborn as sns

plt.figure(dpi=130)
sns.heatmap(corr, annot=True, fmt='.2f')
plt.show()
```

✓ 0.4s



Normalization

MinMaxScaler scales the data so that each feature is in the range [0, 1].

It works well when the features have different scales and the algorithm being used is sensitive to the scale of the features, such as k-nearest neighbors or neural networks.

Rescale your data using scikit-learn using the **MinMaxScaler**.

Code:

```
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline

numeric_cols = dataset.select_dtypes(include=['number']).columns
categorical_cols = dataset.select_dtypes(exclude=['number']).columns

numeric_transformer = Pipeline(steps=[
    ('scaler', MinMaxScaler(feature_range=(0, 1)))
])

categorical_transformer = Pipeline(steps=[
    ('encoder', OneHotEncoder())
])

preprocessor = ColumnTransformer(
    transformers=[
        ('num', numeric_transformer, numeric_cols),
        ('cat', categorical_transformer, categorical_cols)
    ]
)
```

```
X_transformed = preprocessor.fit_transform(dataset)
```

```
X_transformed[:5]
```

Output:

```
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
numeric_cols = dataset.select_dtypes(include=['number']).columns
categorical_cols = dataset.select_dtypes(exclude=['number']).columns
numeric_transformer = Pipeline(steps=[
    ('scaler', MinMaxScaler(feature_range=(0, 1)))
])
categorical_transformer = Pipeline(steps=[
    ('encoder', OneHotEncoder())
])
preprocessor = ColumnTransformer(
    transformers=[
        ('num', numeric_transformer, numeric_cols),
        ('cat', categorical_transformer, categorical_cols)
    ]
)
X_transformed = preprocessor.fit_transform(dataset)
```


✓ 0.1s

```
<10x1222 sparse matrix of type '<class 'numpy.float64'>'
  with 146 stored elements in Compressed Sparse Row format>
```

Standardization

Standardization is a useful technique to transform attributes with a Gaussian distribution and differing means and standard deviations to a standard Gaussian distribution with a mean of 0 and a standard deviation of 1.

We can standardize data using scikit-learn with the StandardScaler class.

It works well when the features have a normal distribution or when the algorithm being used is not sensitive to the scale of the features

Code:

```
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
numeric_cols = dataset.select_dtypes(include=['number']).columns
categorical_cols = dataset.select_dtypes(exclude=['number']).columns
numeric_transformer = Pipeline(steps=[('scaler', StandardScaler())])
categorical_transformer = Pipeline(steps=[('encoder', OneHotEncoder())])
preprocessor = ColumnTransformer(
    transformers=[
        ('num', numeric_transformer, numeric_cols), ('cat',
        categorical_transformer, categorical_cols])
X_transformed = preprocessor.fit_transform(dataset)
```


X_transformed[:5]

Output:

```
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
numeric_cols = dataset.select_dtypes(include=['number']).columns
categorical_cols = dataset.select_dtypes(exclude=['number']).columns
numeric_transformer = Pipeline(steps=[
    ('scaler', StandardScaler())
])

categorical_transformer = Pipeline(steps=[
    ('encoder', OneHotEncoder())
])
preprocessor = ColumnTransformer(
    transformers=[
        ('num', numeric_transformer, numeric_cols),
        ('cat', categorical_transformer, categorical_cols)
    ])
x_transformed = preprocessor.fit_transform(dataset)
x_transformed[:5]
```

✓ 0.0s

✓ 0.1s

```
<10x1222 sparse matrix of type '<class 'numpy.float64'>'
  with 150 stored elements in Compressed Sparse Row format>
```

K-means Clustering Function:

K-means clustering is a machine learning and data analysis technique used for grouping data points into clusters based on their similarity. It's primarily used for:

Unsupervised Learning: K-means helps identify patterns or structure in data without labelled categories.

Segmentation: It can segment data into distinct groups, making it useful for customer segmentation, image compression, and more.

Pattern Recognition: It's used in pattern recognition tasks, such as image analysis and natural language processing.

Anomaly Detection: It can identify outliers by placing data points that don't fit well into any cluster.

Data Compression: K-means can reduce the dimensionality of data while preserving important information.

Recommendation Systems: It can be applied to recommend items or services based on user preferences.

Code:

```
plt.scatter(x[y_kmeans==0,0],x[y_kmeans==0,1],s=100,c="red",label = "cluster1")
plt.scatter(x[y_kmeans==1,0],x[y_kmeans==1,1],s=100,c="blue",label = "cluster2")
plt.scatter(x[y_kmeans==2,0],x[y_kmeans==2,1],s=100,c="green",label = "cluster3")
plt.scatter(x[y_kmeans==3,0],x[y_kmeans==3,1],s=100,c="cyan",label = "cluster4")
plt.scatter(x[y_kmeans==4,0],x[y_kmeans==4,1],s=100,c="magenta",label = "cluster
5") plt.scatter(kmeans.cluster_centers[:,0],kmeans.cluster_centers[:,1],s=300,c="y
```

```
ellow",label="centroids")
```

```
plt.title('clusters of customers')
```

```
plt.xlabel('Annual Income')
```

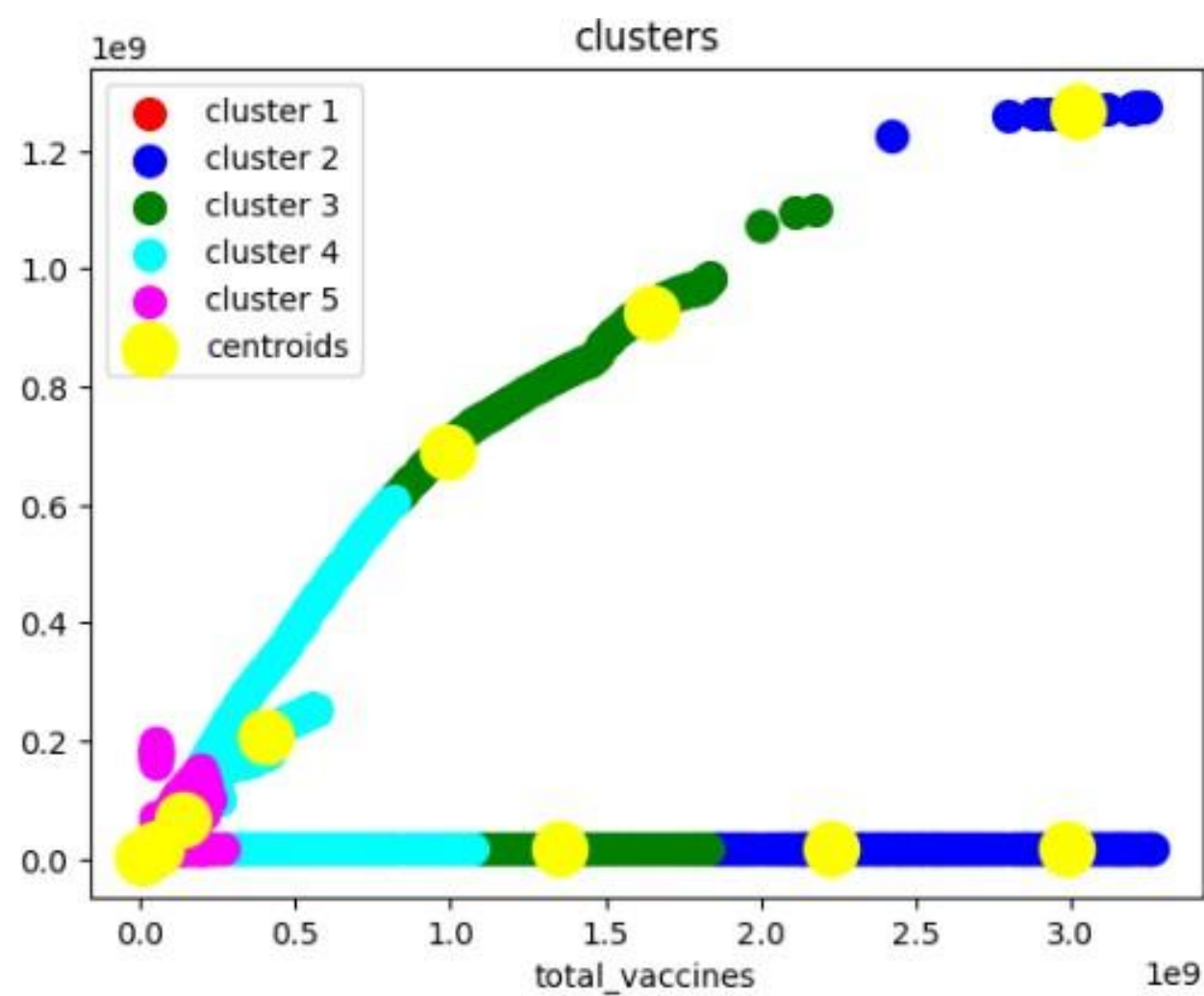
```
plt.ylabel=('Spending Score')
```

```
plt.legend()
```

```
plt.show()
```

Output:

```
plt.scatter(x[y_kmeans==0,0],x[y_kmeans==0,1],s=100,c="red",label = "cluster 1")
plt.scatter(x[y_kmeans==1,0],x[y_kmeans==1,1],s=100,c="blue",label = "cluster 2")
plt.scatter(x[y_kmeans==2,0],x[y_kmeans==2,1],s=100,c="green",label = "cluster 3")
plt.scatter(x[y_kmeans==3,0],x[y_kmeans==3,1],s=100,c="cyan",label = "cluster 4")
plt.scatter(x[y_kmeans==4,0],x[y_kmeans==4,1],s=100,c="magenta",label = "cluster 5")
plt.scatter(kmeans.cluster_centers[:,0],kmeans.cluster_centers[:,1],s=300,c="yellow",label="centroids")
plt.title("clusters of customers")
plt.xlabel("Annual Income")
plt.ylabel("Spending Score")
plt.legend()
plt.show()
```

WCSS Function:

WCSS is the sum of the squared distance between each point and the centroid in a cluster.

When we plot the WCSS with the K value, the plot looks like an Elbow.

As the number of clusters increases, the WCSS value will start to decrease.

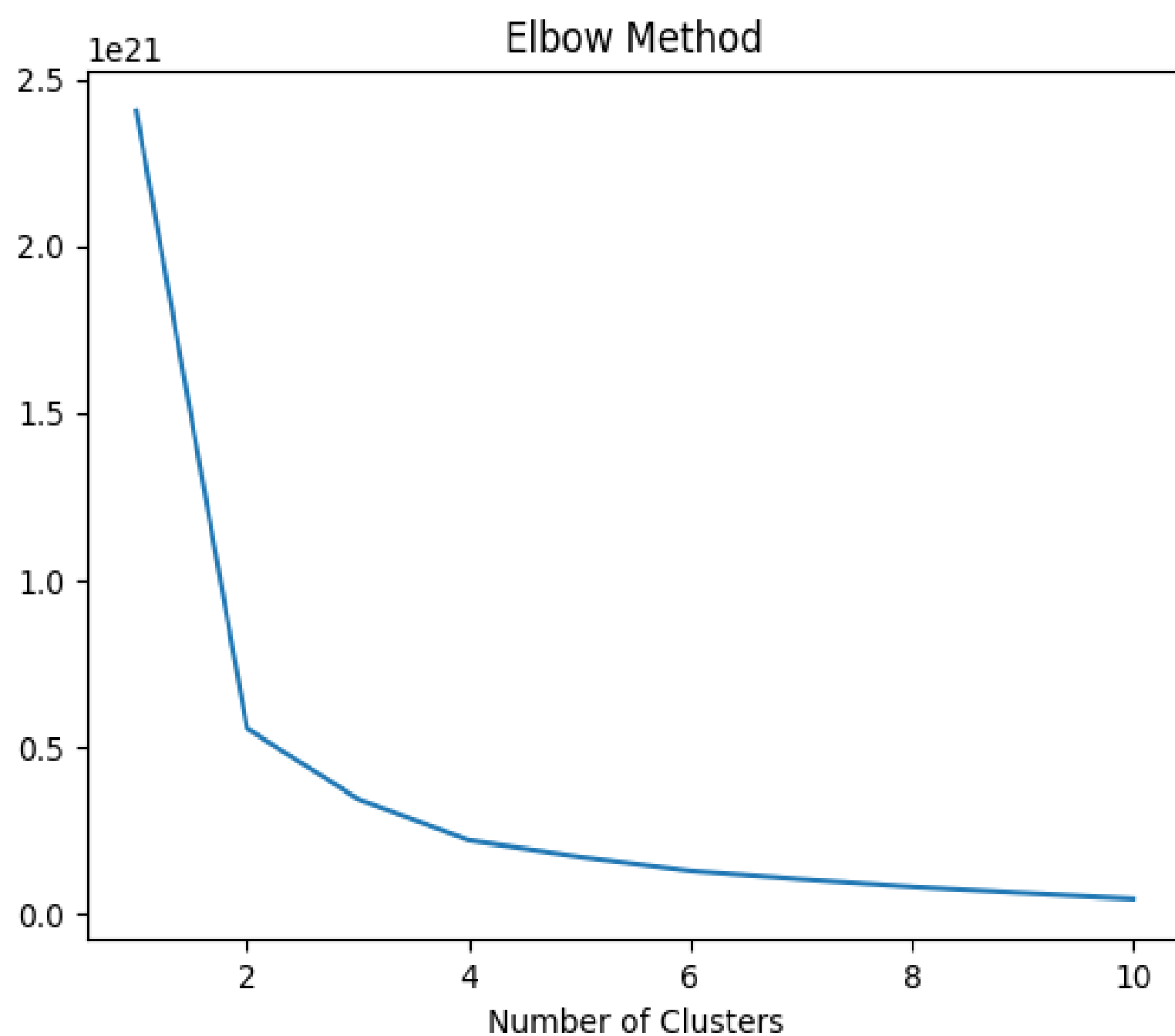
Code:

```
plt.plot(range(1,11),wcss)
plt.title("elbow method")
plt.xlabel("no of cluster")
plt.ylabel("wcss")
plt.show()
```

Output:

```
plt.plot(range(1,11),wcss)
plt.title("elbow method")
plt.xlabel("no of cluster")
plt.ylabel("wcss")
plt.show()
```

✓ 0.0s



Dataset.columns :

We can use the loc and iloc functions to access columns in a PandasDataFrame. for example the Grades column, we could simply use the loc function andspecify the name of the column in order to retrieve it.

Code:

```
dataset.columns
```

Output:

```
dataset.columns
```

✓ 0.0s

```
Index(['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'])
```

Memory Function:

Pandas dataframe.memory_usage() function return the memory usage of each column in bytes.

The memory usage can optionally include the contribution of the index and elements of object dtype. This value is displayed in DataFrame.info by default.

Code:

```
memory = dataset.memory_usage()  
print(memory)  
print('Total Memory Usage = ',sum(memory))
```

Output:


```
memory = dataset.memory_usage()
print(memory)
print("Total Memory Usage = ",sum(memory))
```

✓ 0.0s

Index	132
country	692096
iso_code	692096
date	692096
total_vaccinations	692096
people_vaccinated	692096
people_fully_vaccinated	692096
daily_vaccinations_raw	692096
daily_vaccinations	692096
total_vaccinations_per_hundred	692096
people_vaccinated_per_hundred	692096
people_fully_vaccinated_per_hundred	692096
daily_vaccinations_per_million	692096
vaccines	692096
source_name	692096
source_website	692096
dtype: int64	
Total Memory Usage =	10381572

Dropna() Function:

dropna() is a function used in data preprocessing, often in the context of data analysis and cleaning, to remove or drop rows or columns with missing (NaN or null) values from a dataset.

It's a method to eliminate incomplete or unreliable data from your dataset, which can be important to ensure the quality of your analysis or machinelearning models

Code:

```
print("Size before dropping NaN rows",dataset.shape,"\n")
nan_dropped = dataset.dropna()
```

```
print(nan_dropped.isnull().sum())
print("\nSize after dropping NaN rows",nan_dropped.shape)
```

Output:

```
print("Size before dropping NaN rows",dataset.shape,"\n")

nan_dropped = dataset.dropna()

print(nan_dropped.isnull().sum())
print("\nSize after dropping NaN rows",nan_dropped.shape)
✓ 0.0s
```

Size before dropping NaN rows (86512, 15)	
country	0
iso_code	0
date	0
total_vaccinations	0
people_vaccinated	0
people_fully_vaccinated	0
daily_vaccinations_raw	0
daily_vaccinations	0
total_vaccinations_per_hundred	0
people_vaccinated_per_hundred	0
people_fully_vaccinated_per_hundred	0
daily_vaccinations_per_million	0
vaccines	0
source_name	0
source_website	0
dtype: int64	
Size after dropping NaN rows (30847, 15)	

Iloc() Function:

The `iloc()` function is a method in pandas, a popular Python library for data manipulation and analysis. It is primarily used to select and access data in a `DataFrame` by integer-based indexing.

Select specific rows and columns from a `DataFrame` using integer-based indexing.

Provide a way to slice and filter data by row and column positions.

Code:

```
X = dataset.iloc[:, :-1].values
Y = dataset.iloc[:, 3].values
print('X: %s'%(str(X)))
print('-')
print('Y: %s'%(str(Y)))
```

Output:

```
X = dataset.iloc[:, :-1].values
Y = dataset.iloc[:, 3].values
print('X: %s'%(str(X)))
print('-----')
print('Y: %s'%(str(Y)))
✓ 0.0s
```

```
X: [['Afghanistan' 'AFG' '22-02-2021' ... nan
      'Johnson&Johnson, Oxford/AstraZeneca, Pfizer/BioNTech, Sinopharm/Beijing'
      'World Health Organization']
      ['Afghanistan' 'AFG' '23-02-2021' ... 34.0
      'Johnson&Johnson, Oxford/AstraZeneca, Pfizer/BioNTech, Sinopharm/Beijing'
      'World Health Organization']
      ['Afghanistan' 'AFG' '24-02-2021' ... 34.0
      'Johnson&Johnson, Oxford/AstraZeneca, Pfizer/BioNTech, Sinopharm/Beijing'
      'World Health Organization']
      ...
      ['Zimbabwe' 'ZWE' '27-03-2022' ... 6005.0
      'Oxford/AstraZeneca, Sinopharm/Beijing, Sinovac, Sputnik V'
      'Ministry of Health']
      ['Zimbabwe' 'ZWE' '28-03-2022' ... 6667.0
      'Oxford/AstraZeneca, Sinopharm/Beijing, Sinovac, Sputnik V'
      'Ministry of Health']
      ['Zimbabwe' 'ZWE' '29-03-2022' ... 6874.0
      'Oxford/AstraZeneca, Sinopharm/Beijing, Sinovac, Sputnik V'
      'Ministry of Health']]
-----
Y: [      0.      nan      nan ... 8845039. 8934360. 9039729.]
```

Subplots Function:

Subplots are a feature in data visualization that allow you to create multiple smaller plots within a larger figure.

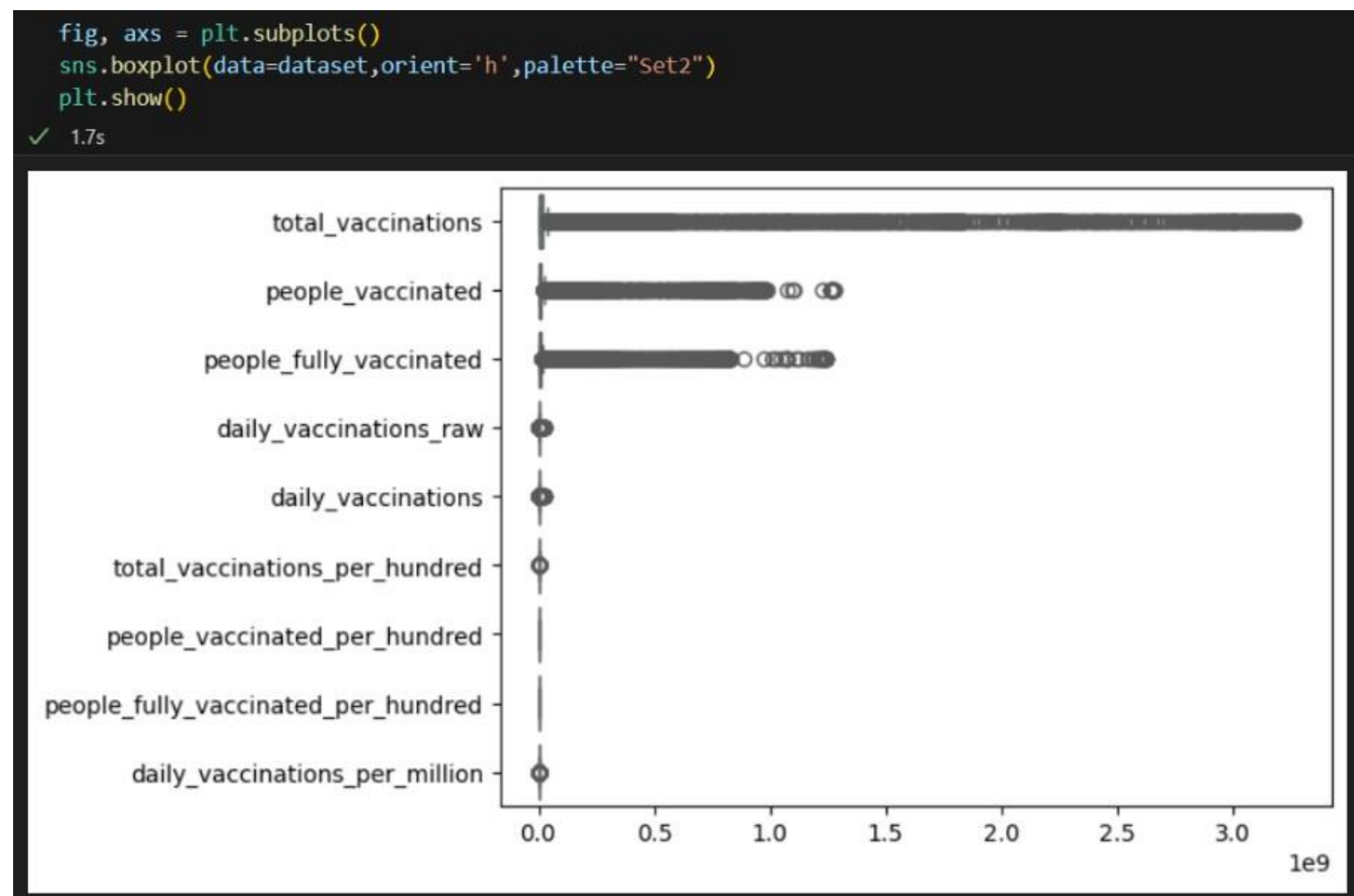
They are useful for displaying multiple related visualizations side by side, making it easier to compare and analyze data.

subplots help you arrange and present multiple charts, graphs, or plots in a single figure, improving the overall clarity and readability of your data visualizations.

Code:

```
fig, axs = plt.subplots()
sns.boxplot(data=dataset, orient='h', palette="Set2")
plt.show()
```

Output:



Missingno Function:

"missingno" is a Python library used for visualizing and analyzing missing data in a dataset.

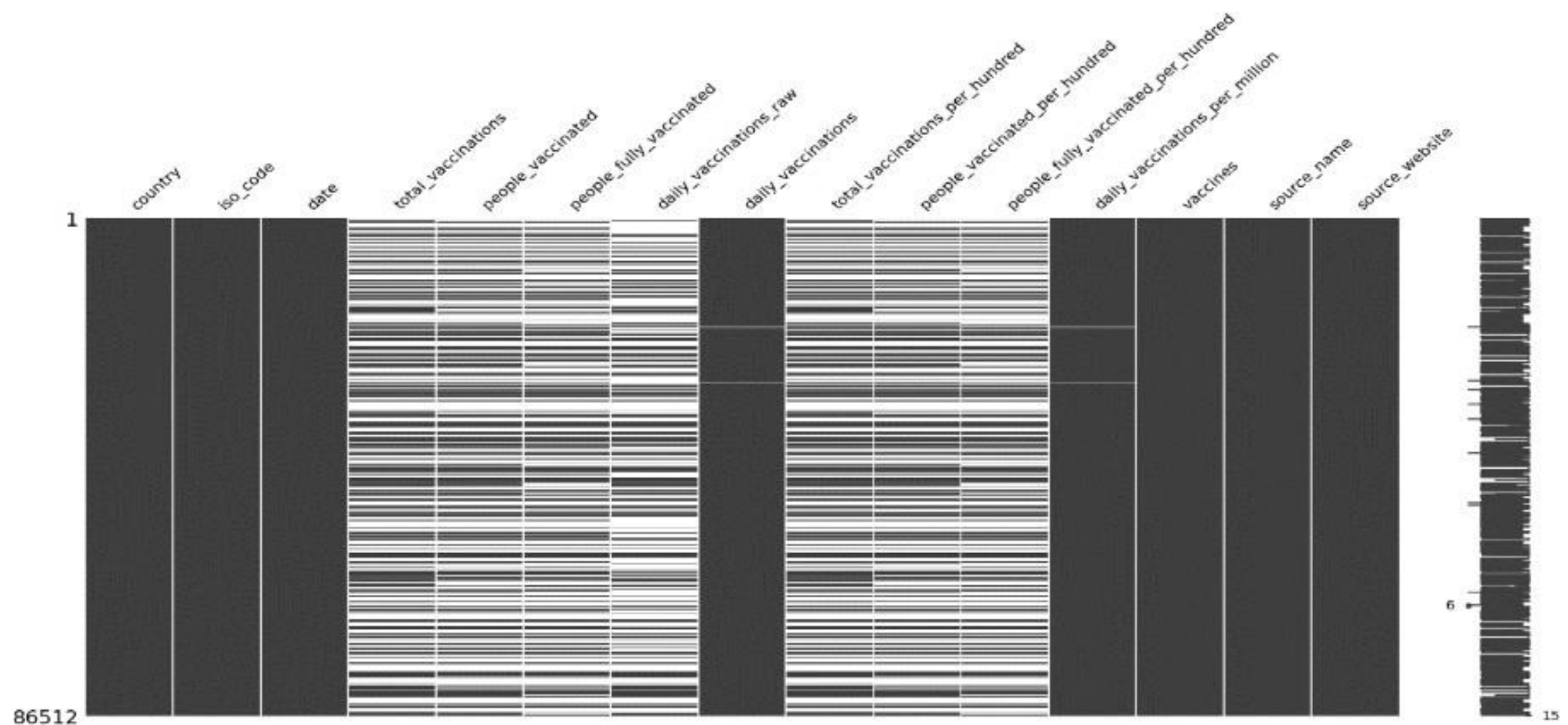
It provides various visualization tools to quickly understand and identify missing values in your data, allowing you to make informed decisions on how to handle or impute missing data.

Code:

```
import missingno as msno
msno.matrix(dataset)
plt.figure(figsize = (15,9))
plt.show()
```

Output:

```
import missingno as msno
msno.matrix(dataset)
plt.figure(figsize = (15,9))
plt.show()
✓ 0.2s
```

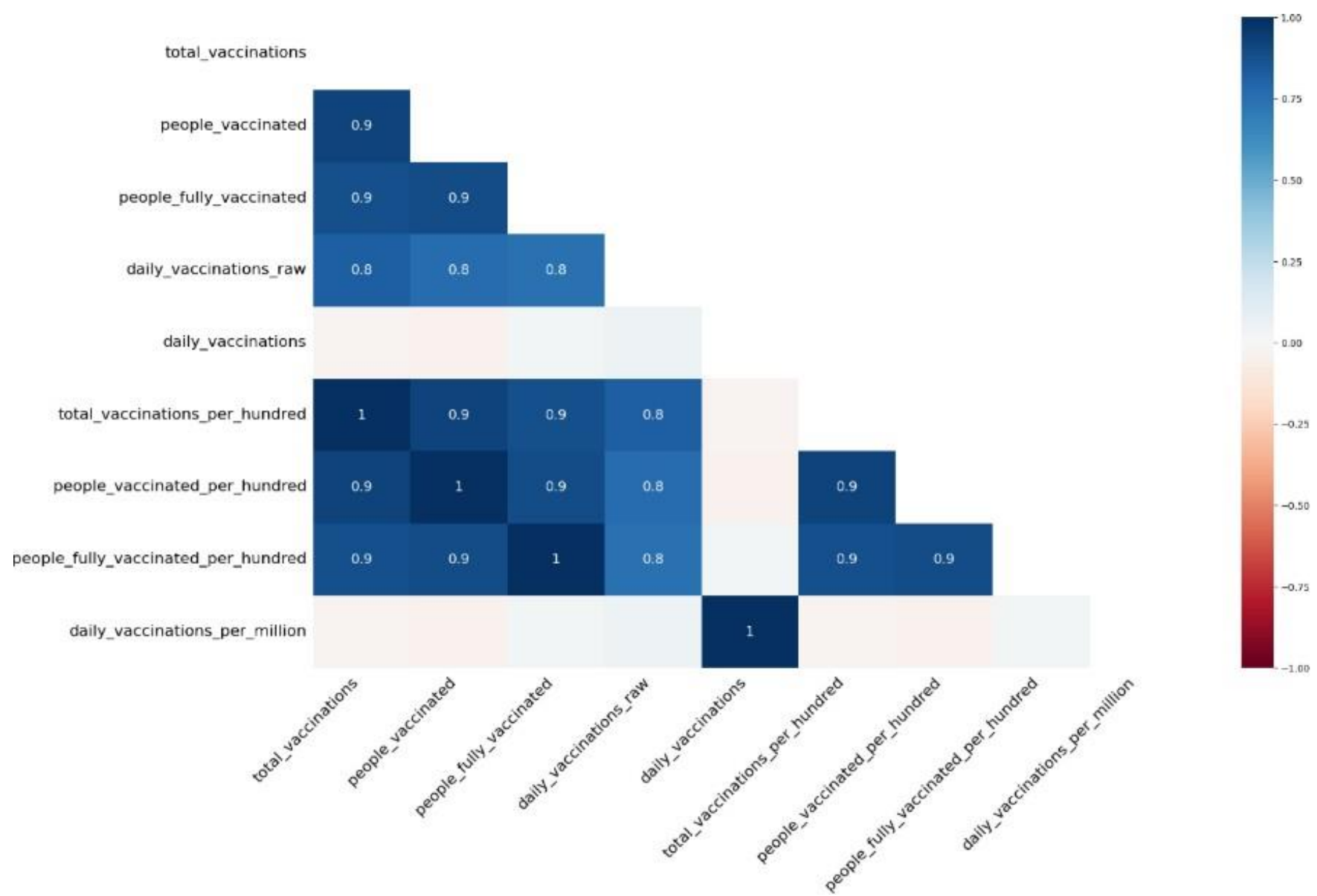


Code:

```
msno.heatmap(dataset, labels = True)
```

Output:

```
msno.heatmap(dataset, labels = True)
✓ 0.1s
```



Load the dataset: (DATASET 2):

Head() Function:

Output:

```
dataset.head
✓ 0.0s

<bound method NDFrame.head of          location      date      vaccine  total_vaccinations
0      Argentina  29-12-2020      Moderna           2
1      Argentina  29-12-2020  Oxford/AstraZeneca           3
2      Argentina  29-12-2020  Sinopharm/Beijing           1
3      Argentina  29-12-2020      Sputnik V        20481
4      Argentina  30-12-2020      Moderna           2
...          ...      ...          ...          ...
35618  European Union  29-03-2022  Oxford/AstraZeneca        67403106
35619  European Union  29-03-2022    Pfizer/BioNTech        600519998
35620  European Union  29-03-2022  Sinopharm/Beijing        2301516
35621  European Union  29-03-2022      Sinovac           1809
35622  European Union  29-03-2022      Sputnik V        1845103

[35623 rows x 4 columns]>
```

Info() Function:

Output:

```
dataset.info()
✓ 0.0s

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 35623 entries, 0 to 35622
Data columns (total 4 columns):
#   Column              Non-Null Count  Dtype
---  -
0   location            35623 non-null  object
1   date                35623 non-null  object
2   vaccine             35623 non-null  object
3   total_vaccinations  35623 non-null  int64
dtypes: int64(1), object(3)
memory usage: 1.1+ MB
```

Df.isnull().sum() Function:

Output:

```
dataset.isnull().sum()

✓ 0.0s

location      0
date           0
vaccine        0
total_vaccinations  0
dtype: int64
```

Describe Function:

Output:

```
dataset.describe()

✓ 0.0s

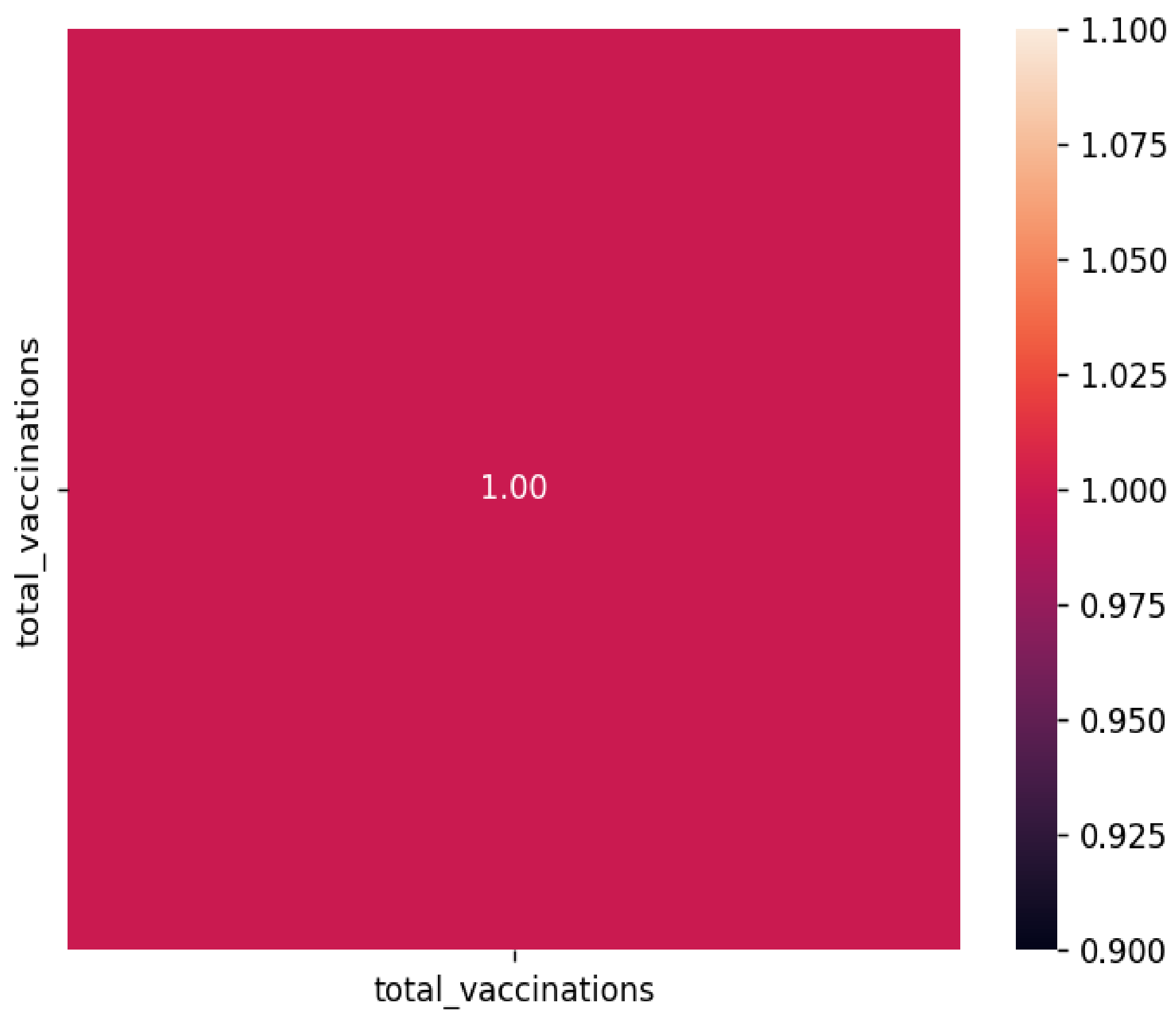
total_vaccinations
count      3.562300e+04
mean       1.508357e+07
std        5.181768e+07
min         0.000000e+00
25%         9.777600e+04
50%         1.305506e+06
75%         7.932423e+06
max         6.005200e+08
```

Corelation:

Output:

```
numeric_dataset = dataset.select_dtypes(include=['number'])  
corr = numeric_dataset.corr()  
import matplotlib.pyplot as plt  
import seaborn as sns  
plt.figure(dpi=130)  
sns.heatmap(corr, annot=True, fmt='.2f')  
plt.show()
```

✓ 0.0s



Normalization:

Output:

```
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
numeric_cols = dataset.select_dtypes(include=['number']).columns
categorical_cols = dataset.select_dtypes(exclude=['number']).columns
numeric_transformer = Pipeline(steps=[
    ('scaler', MinMaxScaler(feature_range=(0, 1)))])
categorical_transformer = Pipeline(steps=[
    ('encoder', OneHotEncoder())])
preprocessor = ColumnTransformer(
    transformers=[
        ('num', numeric_transformer, numeric_cols),
        ('cat', categorical_transformer, categorical_cols)])
X_transformed = preprocessor.fit_transform(dataset)
X_transformed[:5]
```

✓ 0.0s

```
<5x527 sparse matrix of type '<class 'numpy.float64'>'
    with 20 stored elements in Compressed Sparse Row format>
```

Standardization:

Output:

```
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline

numeric_cols = dataset.select_dtypes(include=['number']).columns
categorical_cols = dataset.select_dtypes(exclude=['number']).columns

numeric_transformer = Pipeline(steps=[('scaler', StandardScaler())])
categorical_transformer = Pipeline(steps=[('encoder', OneHotEncoder())])

preprocessor = ColumnTransformer(
    transformers=[
        ('num', numeric_transformer, numeric_cols),
        ('cat', categorical_transformer, categorical_cols)
    ])

X_transformed = preprocessor.fit_transform(dataset)
X_transformed[:10]
```

✓ 0.0s

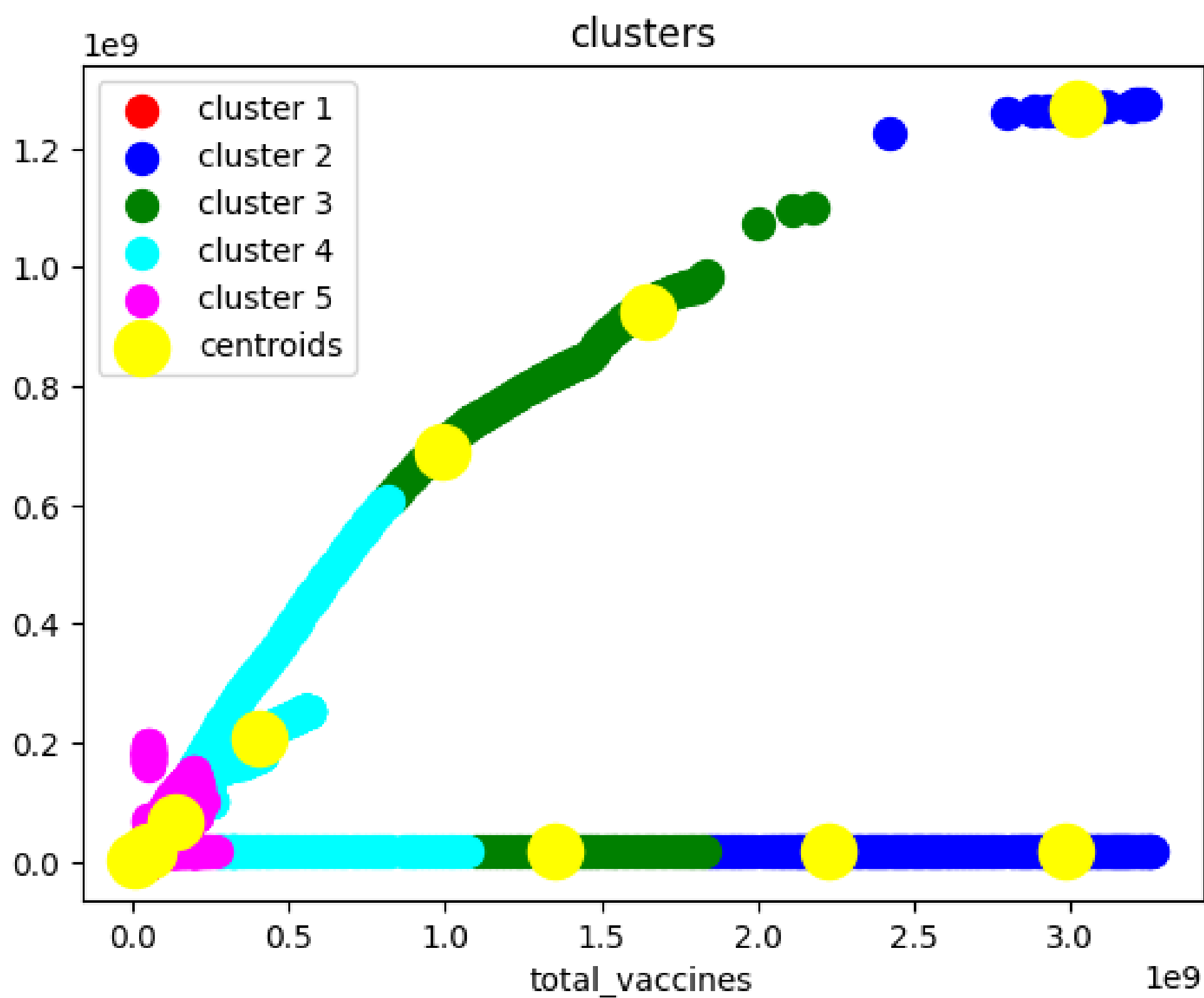
```
<10x527 sparse matrix of type '<class 'numpy.float64'>'
    with 40 stored elements in Compressed Sparse Row format>
```

K-means Clustering Function:

Output:

```
plt.scatter(x[y_kmeans==0],x[y_kmeans==0,1],s=100,c="red",label = "cluster 1")
plt.scatter(x[y_kmeans==1],x[y_kmeans==1,1],s=100,c="blue",label = "cluster 2")
plt.scatter(x[y_kmeans==2],x[y_kmeans==2,1],s=100,c="green",label = "cluster 3")
plt.scatter(x[y_kmeans==3],x[y_kmeans==3,1],s=100,c="cyan",label = "cluster 4")
plt.scatter(x[y_kmeans==4],x[y_kmeans==4,1],s=100,c="magenta",label = "cluster 5")
plt.scatter(kmeans.cluster_centers[:,0],kmeans.cluster_centers[:,1],s=300,c="yellow",label="centroids")
plt.title("clusters")
plt.xlabel("total_vaccines")
plt.ylabel="People_vaccinated")
plt.legend()
plt.show()
```

✓ 0.7s



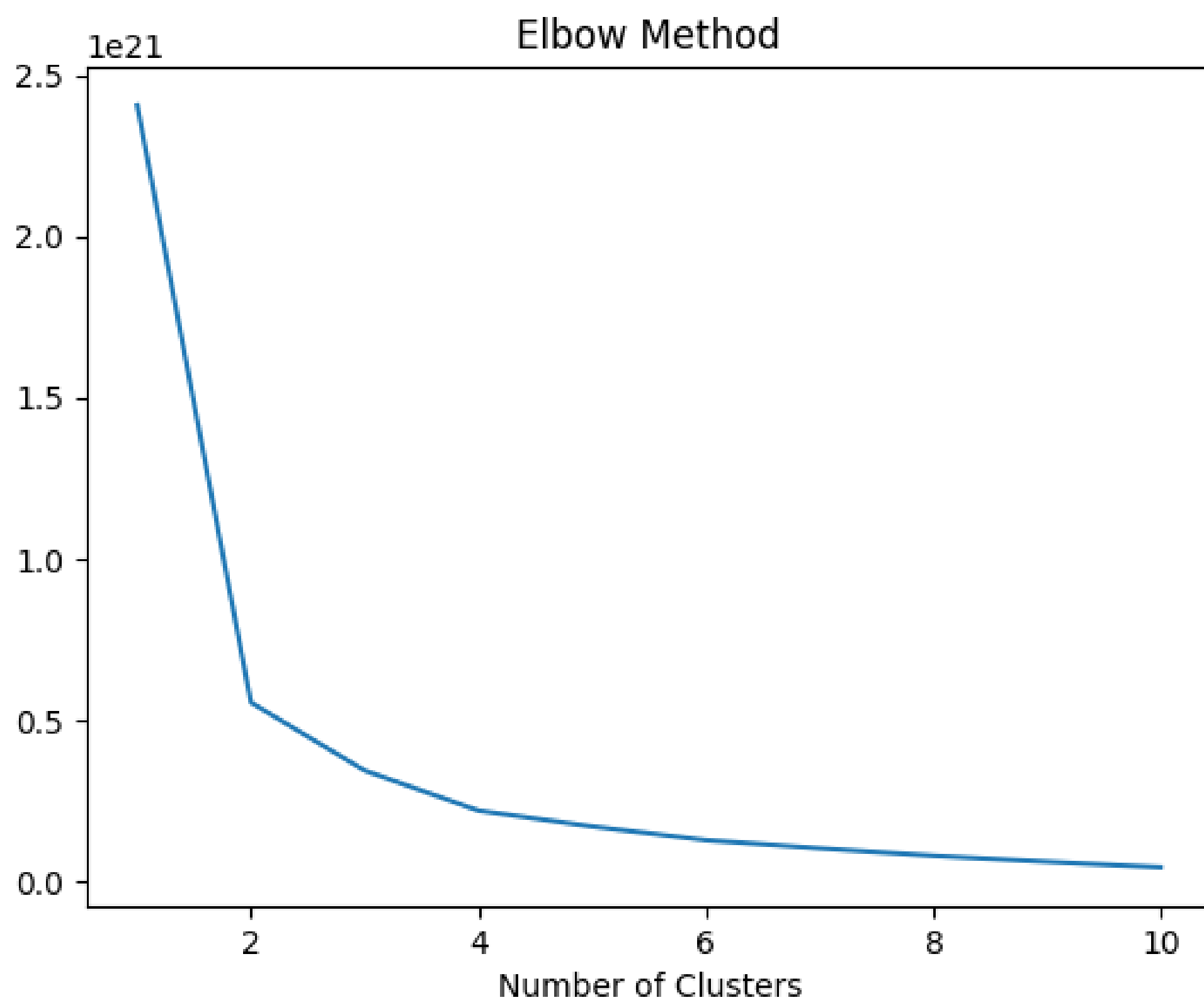
WCSS Function:

Output:

```
import matplotlib.pyplot as plt

# Your K-Means clustering code

# Create the elbow plot
plt.plot(range(1, 11), wcss)
plt.title("Elbow Method")
plt.xlabel("Number of Clusters")
plt.ylabel("WCSS") # Use plt.ylabel to set the y-axis label
plt.show()
```



Dataset.columns:

Output:

```
dataset.columns
✓ 0.0s
Index(['location', 'date', 'vaccine', 'total_vaccinations'], dtype='object')
```

Memory Function:

Output:

```
memory = dataset.memory_usage()
print(memory)
print("Total Memory Usage = ",sum(memory))
✓ 0.0s
Index          132
location      284984
date          284984
vaccine       284984
total_vaccinations 284984
dtype: int64
Total Memory Usage = 1140068
```

Dropna() Function:

Output:

```
memory = dataset.memory_usage()
print(memory)
print("Total Memory Usage = ",sum(memory))
✓ 0.0s
Index          132
location      284984
date          284984
vaccine       284984
total_vaccinations 284984
dtype: int64
Total Memory Usage = 1140068
```

Iloc() Function:

Output:

```
x = dataset.iloc[:, :-1].values
y = dataset.iloc[:, 3].values
print ('X: %s'%(str(x)))
print ('-----')
print ('Y: %s'%(str(y)))
✓ 0.0s
```

X: [['Argentina' '29-12-2020' 'Moderna']
['Argentina' '29-12-2020' 'Oxford/AstraZeneca']
['Argentina' '29-12-2020' 'Sinopharm/Beijing']
...
['European Union' '29-03-2022' 'Sinopharm/Beijing']
['European Union' '29-03-2022' 'Sinovac']
['European Union' '29-03-2022' 'Sputnik V']]

Y: [2 3 1 ... 2301516 1809 1845103]

Subplots Function:

Output:

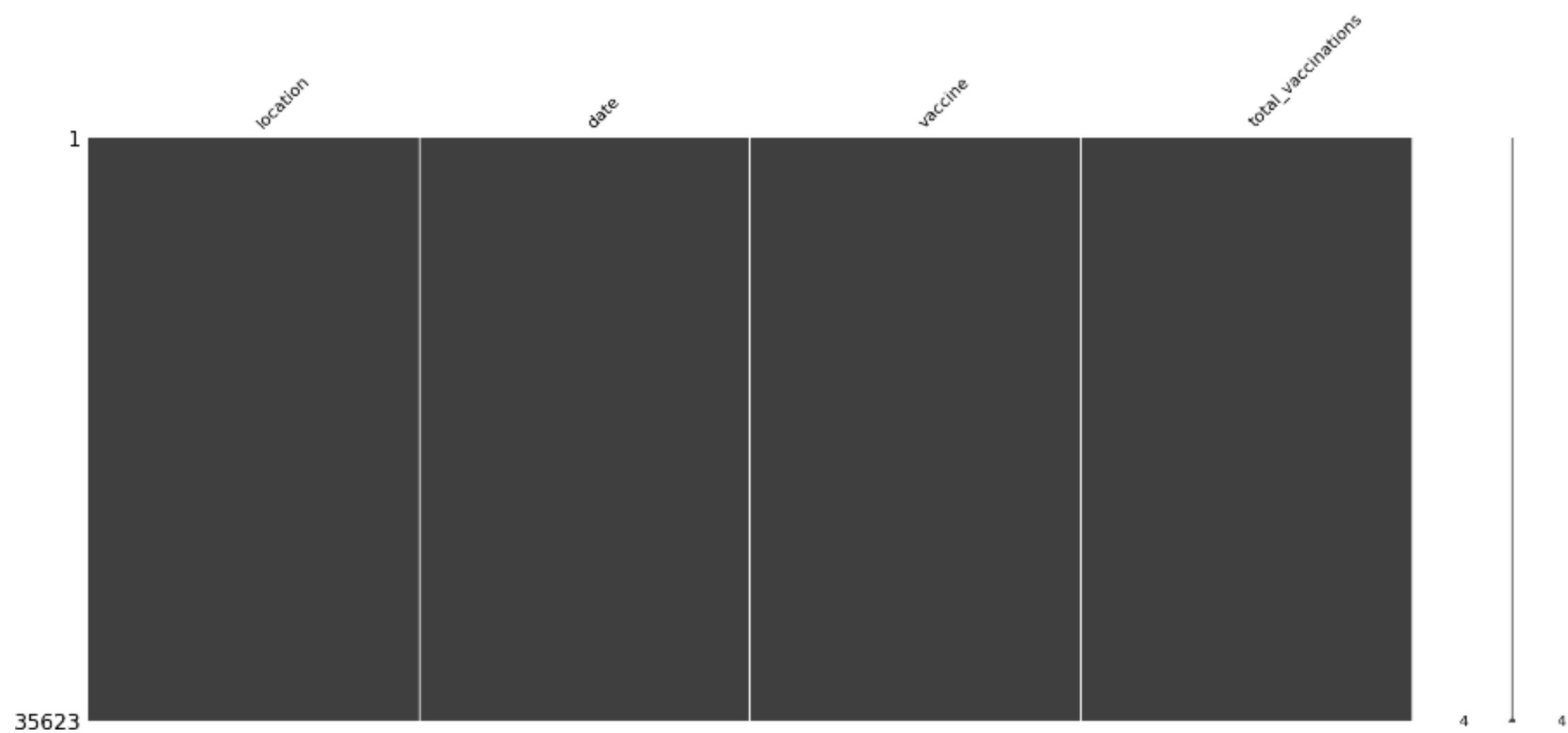


Missingno Function:

Output:

```
msno.heatmap(dataset, labels = True)
```

✓ 0.1s



S

```
import missingno as msno
msno.matrix(dataset)
plt.figure(figsize = (15,9))
plt.show()
```

✓ 0.2s

