**Phase 1: Problem Definition and Design Thinking**

**Project Title:** covid 19 vaccine analaysis

**Problem Definition:**

The problem is to conduct an in-depth analysis of Covid-19 vaccine data, focusing on vaccine efficacy, distribution, and adverse effects. The goal is to provide insights that aid policymakers and health organizations in optimizing vaccine deployment strategies. This project involves data collection, data preprocessing, exploratory data analysis, statistical analysis, and visualization.

**Design Thinking:**

Design thinking is a problem-solving approach that prioritizes human-centered solutions through a structured and iterative process. It encourages creativity, empathy, and collaboration to address complex problems. When applied to a project like conducting an in-depth analysis of COVID-19 vaccine data, design thinking can help ensure that the analysis is not only technically sound but also relevant and actionable for policymakers and health organizations. Here's a detailed breakdown of how design thinking can be applied to this project:

* **Empathize:**

Begin by understanding the needs and perspectives of the stakeholders involved in the COVID-19 vaccine deployment, such as policymakers, healthcare professionals, and the public.

Conduct interviews, surveys, and research to gather insights into their concerns, goals, and pain points related to vaccine efficacy, distribution, and adverse effects.

Develop empathy for the end-users of your analysis to ensure that the results address their real-world challenges.

* **Define:**

Based on the information collected in the empathy phase, define a clear problem statement that encapsulates the key issues and challenges related to COVID-19 vaccine deployment.

Identify specific goals and objectives for the analysis, such as optimizing vaccine distribution, assessing vaccine efficacy, and understanding adverse effects.

* **Ideate:**

Brainstorm potential approaches and strategies for conducting the analysis. Encourage creative thinking and generate a wide range of ideas.

Collaborate with a multidisciplinary team to bring diverse perspectives and expertise to the table. Consider different data sources, analysis methods, and visualization techniques.

* **Prototype:**

Create a preliminary plan for the analysis, including data collection methods, data preprocessing steps, and analytical models.

Develop prototypes or mock-ups of data visualizations and reports to explore how the insights could be presented effectively to stakeholders.

Test and refine these prototypes based on feedback from team members and stakeholders.

* **Test:**

Conduct a pilot analysis using a subset of the data to validate the chosen analytical approach and to identify any potential issues or limitations.

Gather feedback from stakeholders to ensure that the analysis aligns with their needs and expectations.

Iterate on the analysis and visualization based on the feedback received**.**

* **Implement:**

Once the analysis and visualization approaches have been refined and validated, proceed with the full-scale analysis using the complete dataset.

Implement the data collection, preprocessing, statistical analysis, and visualization steps as planned.

* **Evaluate:**

Continuously assess the results and insights generated throughout the analysis process.

Measure the impact of the analysis on decision-making and policy formulation related to COVID-19 vaccine deployment.

Gather feedback from stakeholders and end-users to identify areas for improvement.

* **Iterate:**

Use the feedback and lessons learned from the evaluation phase to make refinements and enhancements to the analysis and its outputs.

Repeat the design thinking process as necessary to address evolving challenges and questions related to COVID-19 vaccine deployment.

**Data Collection:**

* In this step, you gather relevant data related to Covid-19 vaccines. You can obtain data from sources such as government health departments, research institutions, or publicly available datasets like those provided by Kaggle.
* Code Example: Depending on the source, you may use libraries like Pandas to read data from CSV files or APIs to fetch data.

**Data Preprocessing:**

* Clean and prepare the data for analysis. This includes handling missing values, removing duplicates, and converting data types if necessary.
* Code Example: Using Pandas for data cleaning and preprocessing operations.

**Exploratory Data Analysis (EDA):**

* Conduct a preliminary analysis to understand the dataset's characteristics. This involves generating summary statistics, creating visualizations, and identifying patterns or outliers.
* Code Example: Utilize libraries like Matplotlib, Seaborn, or Plotly for creating visualizations to explore the data.

**Statistical Analysis:**

* Perform statistical tests to analyze vaccine efficacy, adverse effects, and distribution. You can use hypothesis testing, regression analysis, or other statistical methods to derive meaningful insights.
* Code Example: Implement statistical tests using libraries like SciPy or StatsModels.

**Visualization:**

* Visualize your findings to communicate insights effectively. This may include creating plots, graphs, and interactive dashboards to illustrate key points.
* Code Example: Continue using visualization libraries to create meaningful charts or dashboards.

**Insights and Recommendations:**

* Summarize the insights you've gained from the analysis. Provide actionable recommendations for policymakers and health organizations based on your findings.
* Code Example: Document your insights and recommendations in a report or presentation.

**Phase 2:** **Innovation**

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

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

Exploring advanced machine learning techniques like clustering and time series forecasting can be valuable in uncovering hidden patterns in vaccine distribution and adverse effects data. Let's dive into each of these techniques with detailed explanations:

**1. Clustering for Vaccine Distribution Analysis:**

Objective: Clustering aims to group similar data points together based on certain features or characteristics. In the context of vaccine distribution, clustering can help identify regions or population groups with similar vaccination patterns, which can be useful for targeted allocation and resource planning.

**Process:**

* **Data Preprocessing:** Start by means of amassing and cleansing your vaccine distribution data. This might also consist of information at the quantity of vaccine doses introduced, vaccination costs, geographical data, and extra.
* **Feature Selection:** Choose relevant features consisting of area, populace density, vaccination charges, and time-related facts.
* **Clustering Algorithm:** Apply clustering algorithms like K-Means, DBSCAN, or hierarchical clustering to group areas or populations with similar vaccination characteristics.
* **Evaluation:** Evaluate the high-quality of clusters the use of metrics like silhouette score, Davies-Bouldin index, or visual inspection of cluster brotherly love and separation.
* **Interpretation:** Examine the clusters to uncover styles, together with regions with consistently high or low vaccination rates, urban vs. Rural clusters, or clusters with precise demographic characteristics.
* **Decision Making:** Use the insights received from clustering to make informed decisions approximately resource allocation, vaccine distribution strategies, and focused outreach efforts.

**2. Time Series Forecasting for Adverse Effects Analysis:**

Objective: Time series forecasting is used to predict future values based on historical data. In the context of adverse effects of vaccines, this technique can help in predicting the occurrence of adverse events over time.

**Process:**

* **Data Collection:** Gather historical statistics on unfavourable effects of vaccines, such as the form of vaccine administered, the date of administration, and said adverse events.
* **Data Preprocessing:** Clean and put together the information, handling lacking values and outliers, and convert it into a time collection layout.
* **Time Series Model Selection:** Choose the perfect time series forecasting version which includes ARIMA (Auto Regressive Integrated Moving Average), LSTM (Long Short-Term Memory), or Prophet.
* **Training and Validation:** Split the facts into training and validation sets. Train the version on the historic information, music hyperparameters, and validate its performance the usage of validation information.
* **Forecasting:** Use the trained model to make predictions on future damaging events based at the historical records.
* **Evaluation:** Evaluate the forecasting model the usage of metrics like Mean Absolute Error (MAE), Mean Squared Error (MSE), or Root Mean Squared Error (RMSE).
* **Interpretation:** Analyse the forecasting effects to become aware of tendencies, seasonality, and potential spikes in unfavourable events related to precise vaccines or time periods.
* **HRisk Assessment:** Use the forecasts to evaluate the potential chance of detrimental consequences, permitting healthcare authorities to take proactive measures such as monitoring, communication, or changes to vaccination schedules.

Both clustering and time series forecasting can provide valuable insights into vaccine distribution and adverse effects data, helping healthcare organizations and policymakers make data-driven decisions to optimize vaccination campaigns and ensure public safety. These techniques should be applied by data scientists and analysts with expertise in machine learning and domain knowledge in healthcare and epidemiology for the most accurate and meaningful results.

**Clustering for Vaccine Distribution Data:**

* Clustering can help identify patterns in vaccine distribution data, such as grouping regions or locations with similar distribution characteristics. In this example, we'll use K-Means clustering as it's a widely used technique for this purpose.
* Clustering can help identify patterns in vaccine distribution data, such as grouping regions or populations with similar distribution characteristics.

**Explanation:**

We'll use K-means clustering, a popular technique for grouping data points into clusters. In this context, each data point might represent a region or location, and the features could include distribution rates, vaccine types, and demographics.

**Code (Python with scikit-learn):**

import pandas as pd

from sklearn.cluster import KMeans

import matplotlib.pyplot as plt

# Load your vaccine distribution data into a DataFrame

data = pd.read\_csv('vaccine\_distribution\_data.csv')

# Select relevant features for clustering, e.g., number of doses, location, etc.

features = data[['Doses', 'Location\_X', 'Location\_Y']]

# Choose the number of clusters (you may need to experiment with this)

num\_clusters = 4

# Apply K-Means clustering

kmeans = KMeans(n\_clusters=num\_clusters)

data['Cluster'] = kmeans.fit\_predict(features)

# Visualize the clusters

for cluster in range(num\_clusters):

cluster\_data = data[data['Cluster'] == cluster]

plt.scatter(cluster\_data['Location\_X'], cluster\_data['Location\_Y'], label=f'Cluster {cluster}')

plt.xlabel('Location\_X')

plt.ylabel('Location\_Y')

plt.legend()

plt.show()

**Time Series Forecasting for Adverse Effects Data:**

* Time series forecasting can be used to predict adverse effects over time, helping healthcare professionals and authorities make informed decisions. Let's use an example of ARIMA (Autoregressive Integrated Moving Average) for time series forecasting.
* Time series forecasting can help predict adverse effects of vaccines over time, allowing healthcare organizations to prepare for potential surges in cases.

**Explanation:**

We'll use an autoregressive integrated moving average (ARIMA) model, a common technique for time series forecasting. In this scenario, the time series data would represent the number of adverse effects reported over time.

**Code (Python with statsmodels):**

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from statsmodels.tsa.arima\_model import ARIMA

# Load your adverse effects time series data into a DataFrame

data = pd.read\_csv('adverse\_effects\_data.csv')

data['Date'] = pd.to\_datetime(data['Date'])

data.set\_index('Date', inplace=True)

# Visualize the time series data

plt.plot(data)

plt.xlabel('Date')

plt.ylabel('Adverse Effects Count')

plt.show()

# Split the data into training and testing sets

train\_size = int(len(data) \* 0.8)

train\_data, test\_data = data[:train\_size], data[train\_size:]

# Fit an ARIMA model

model = ARIMA(train\_data, order=(5,1,0))

model\_fit = model.fit(disp=0)

# Make predictions on the test set

predictions = model\_fit.forecast(steps=len(test\_data))[0]

# Visualize the predictions vs. actual data

plt.plot(test\_data, label='Actual')

plt.plot(test\_data.index, predictions, color='red', label='Predicted')

plt.xlabel('Date')

plt.ylabel('Adverse Effects Count')

plt.legend()

plt.show()

**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:**

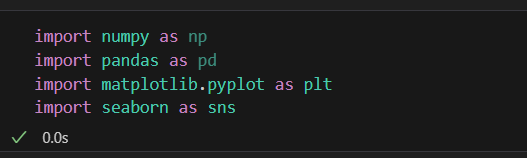
import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

**Output:**



**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\_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

**Output:**

****

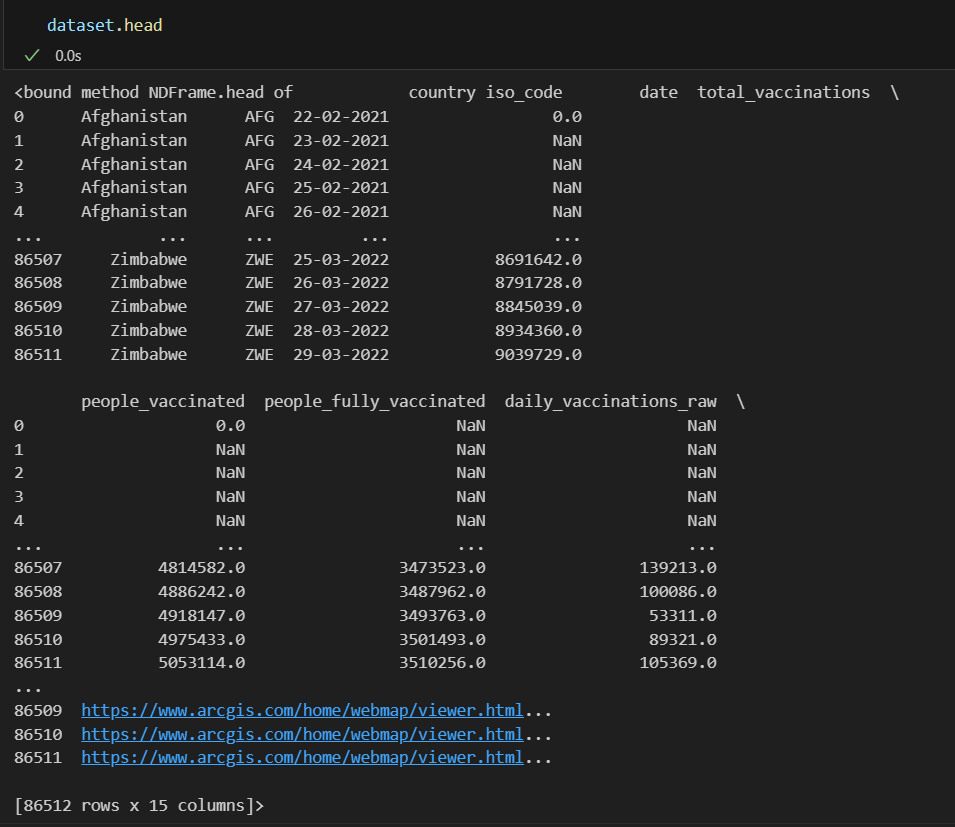
**Head() Function:**

* 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:**



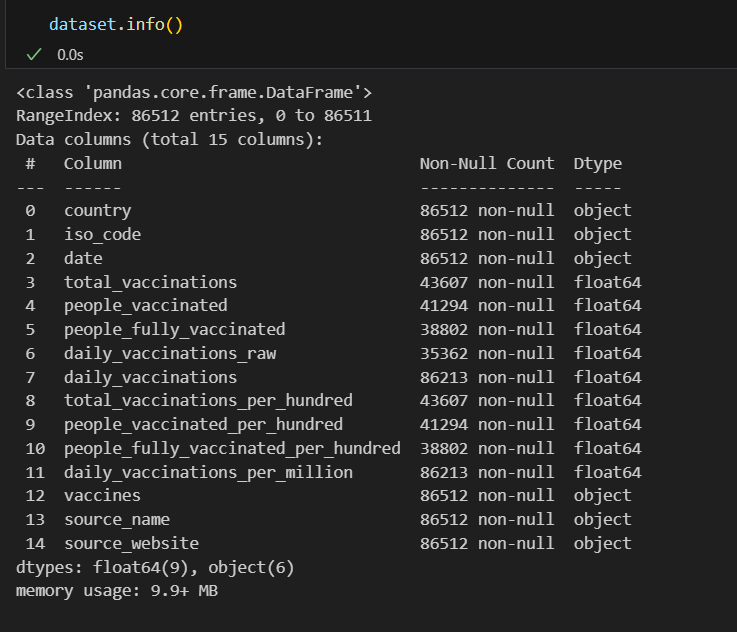
**Info() Function:**

* 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:**



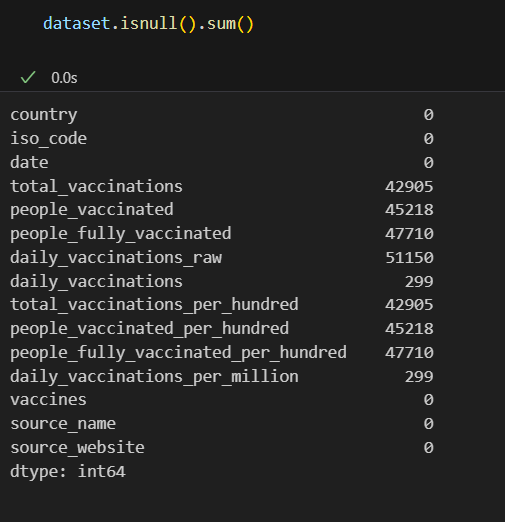
**Df.isnull().sum() Function:**

* 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.Top of Form

**Code:**

dataset.isnull().sum()

**Output:**



**Describe Function:**

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

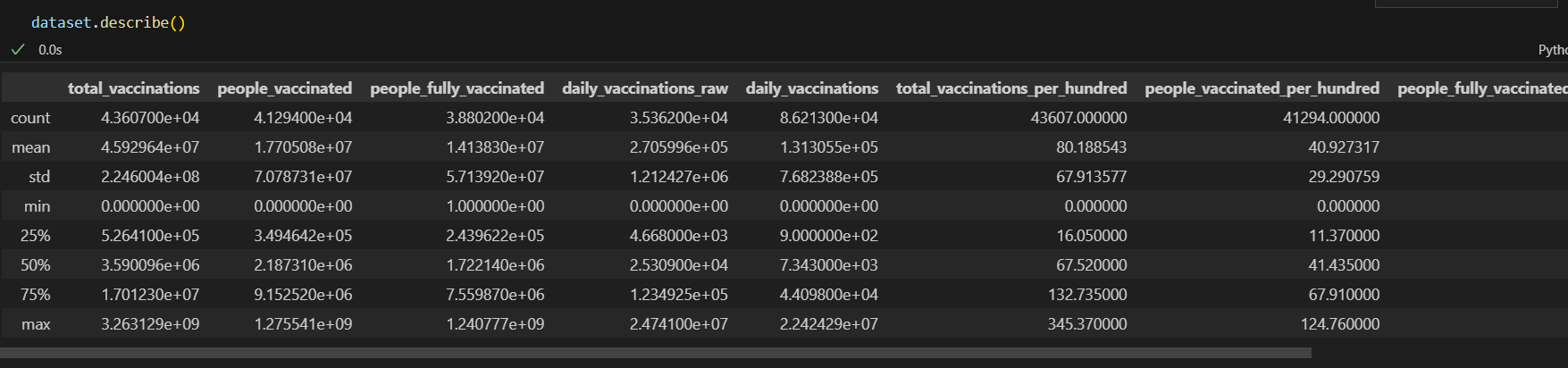
It provides a quick overview of the key statistics for numerical data in the dataset, including:

* **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:**

dataset.describe()

**Output:**

****

**Outliers:**

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

**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

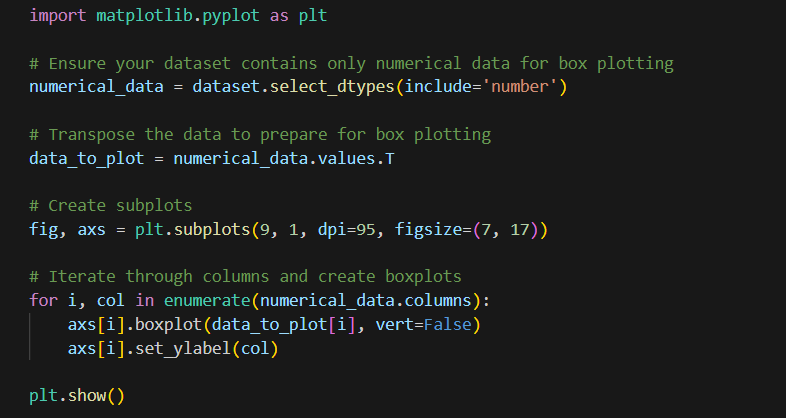
for i, col in enumerate(numerical\_data.columns):

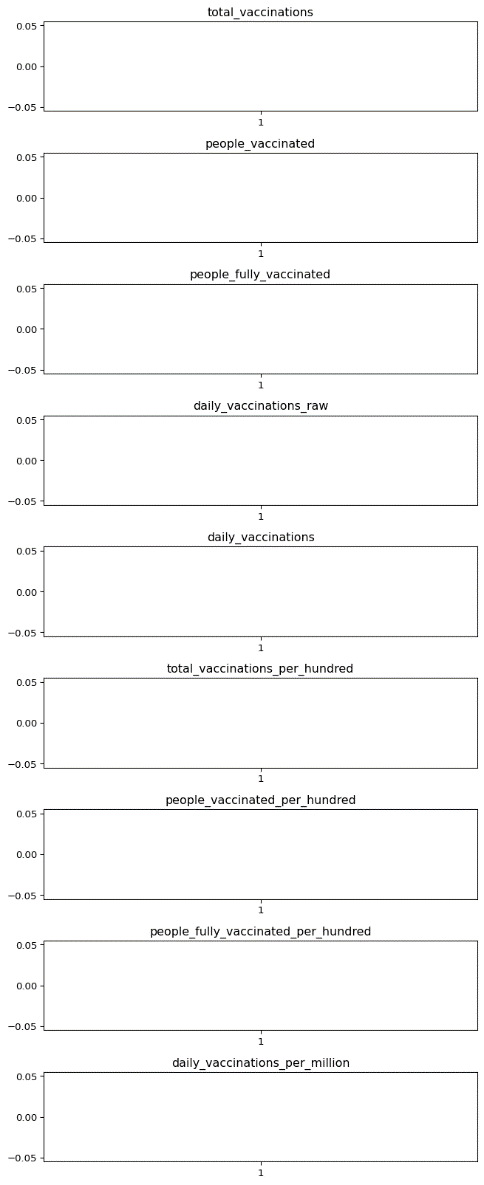
    axs[i].boxplot(data\_to\_plot[i], vert=False)

    axs[i].set\_ylabel(col)

plt.show()

**Output:**

****

****

**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

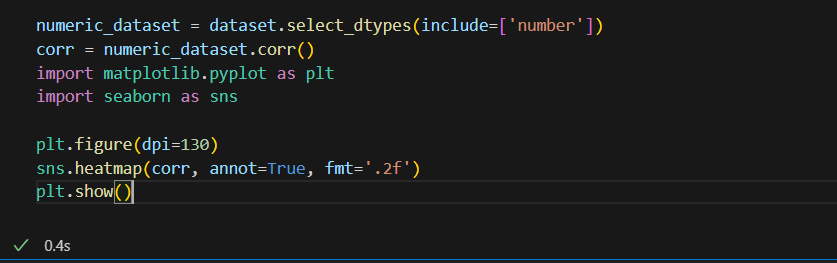
import seaborn as sns

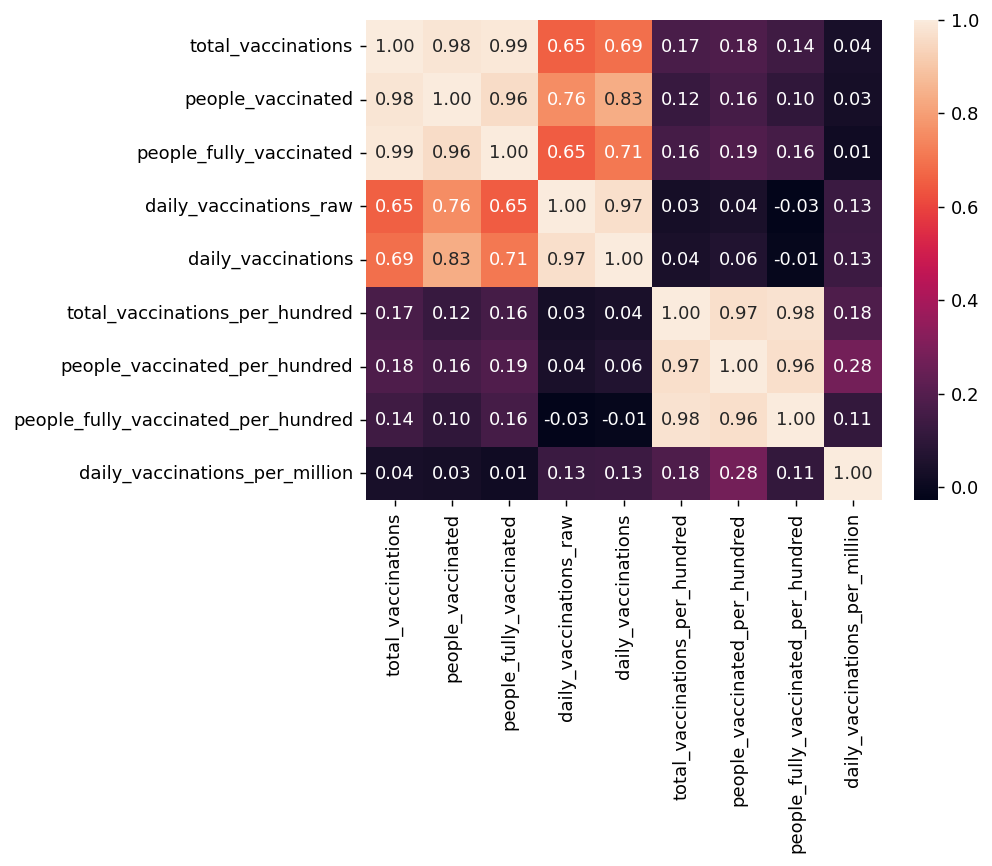
plt.figure(dpi=130)

sns.heatmap(corr, annot=True, fmt='.2f')

plt.show()

**Output:**

****

****

**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 MinMaxScalar.

**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=[

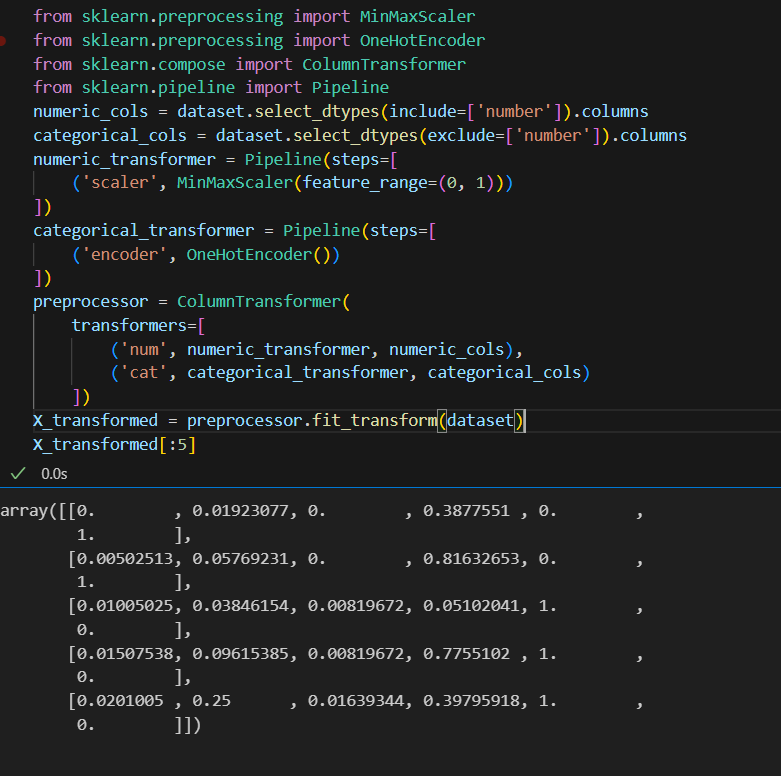
('num', numeric\_transformer, numeric\_cols),

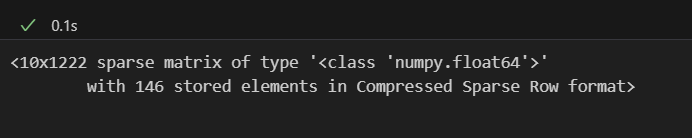
('cat', categorical\_transformer, categorical\_cols)])

X\_transformed = preprocessor.fit\_transform(dataset)

X\_transformed[:5]

**Output:**



****

**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 StandardScalar 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=[

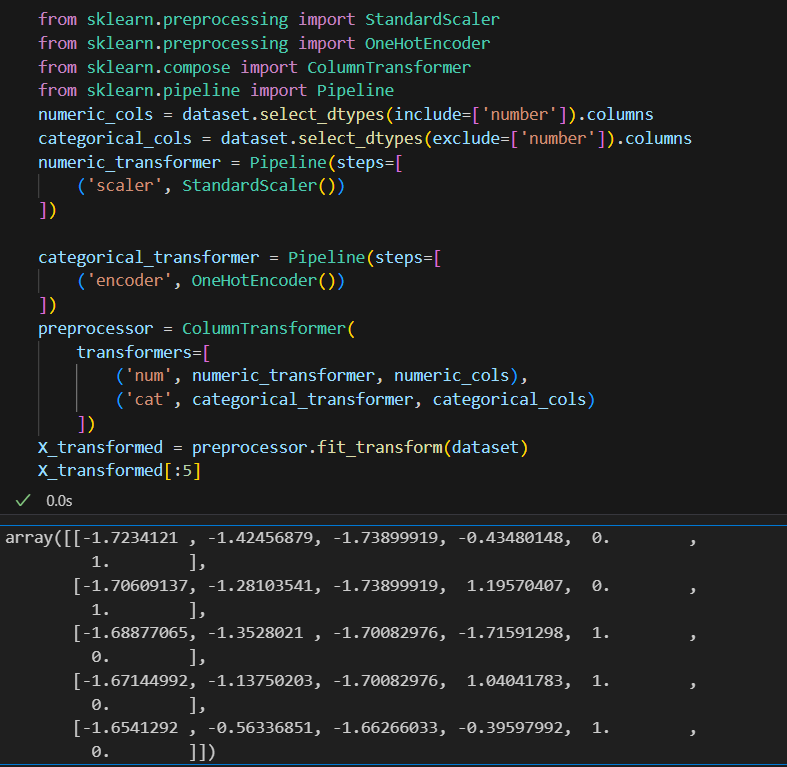
('num', numeric\_transformer, numeric\_cols),

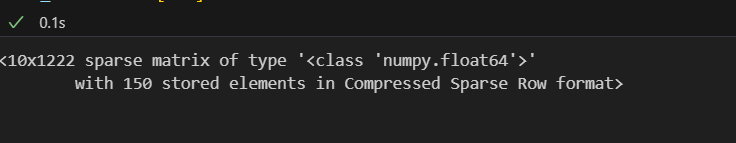
('cat', categorical\_transformer, categorical\_cols])

X\_transformed = preprocessor.fit\_transform(dataset)

X\_transformed[:5]

**Output:**

****



**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 = "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")

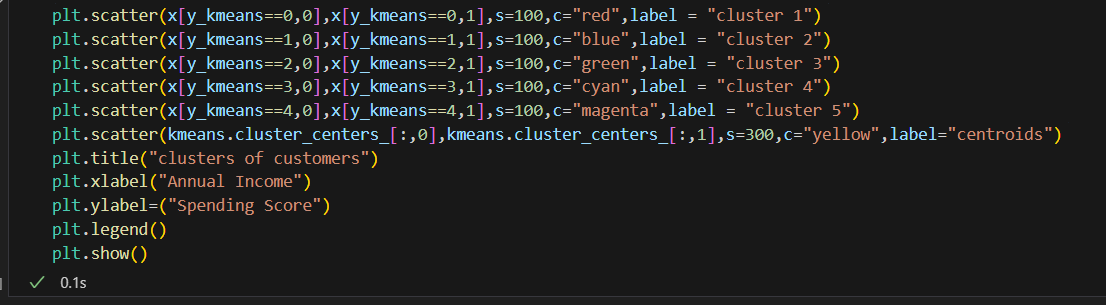
plt.xlabel("Annual Income")

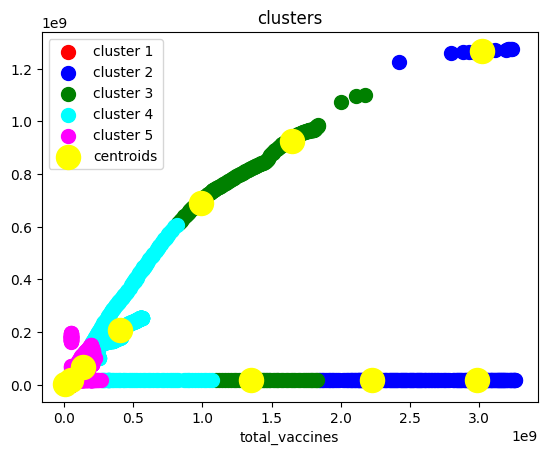
plt.ylabel=("Spending Score")

plt.legend()

plt.show()

**Output:**





**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)

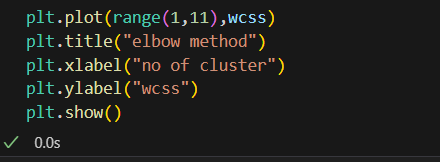
plt.title("elbow method")

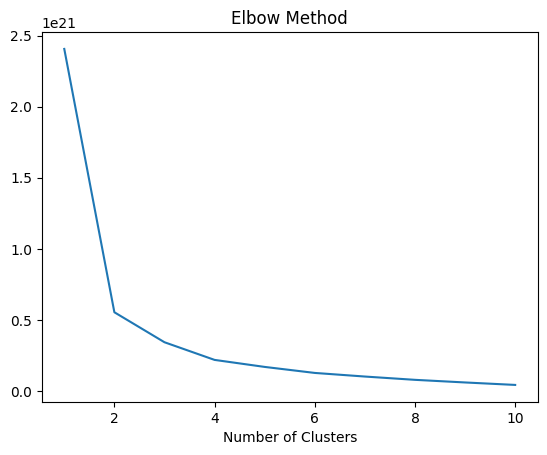
plt.xlabel("no of cluster")

plt.ylabel("wcss")

plt.show()

**Output:**

****



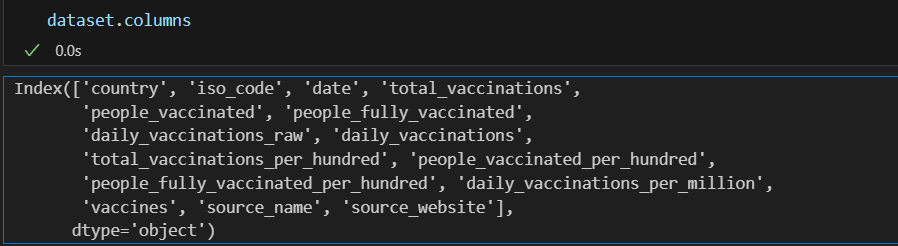
**Dataset columns:**

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

**Code:**

dataset.columns

**Output:**

****

**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.

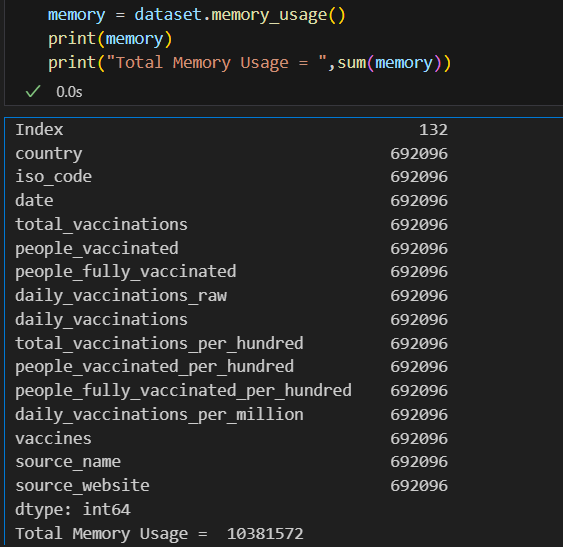
**Code:**

memory = dataset.memory\_usage()

print(memory)

print("Total Memory Usage = ",sum(memory))

**Output:**



**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 machine learning models

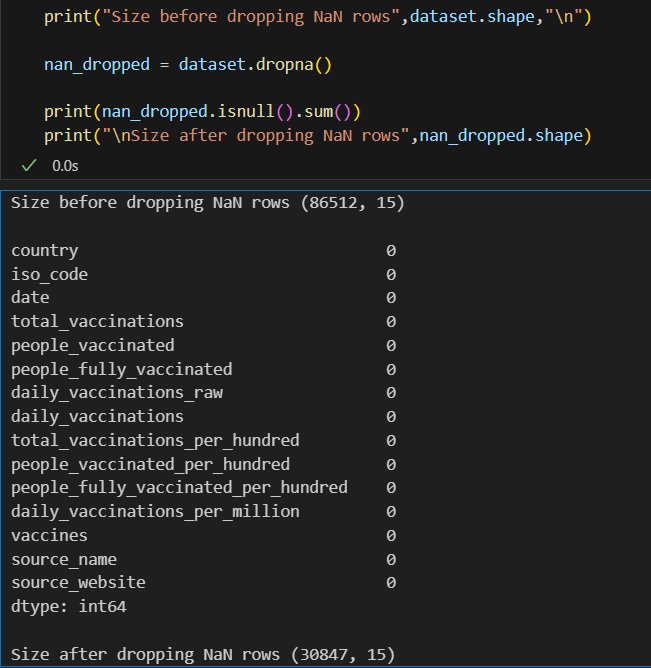
**Code:**

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:**

**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:**

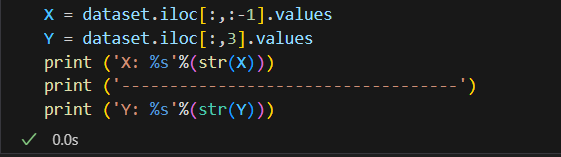
X = dataset.iloc[:,:-1].values

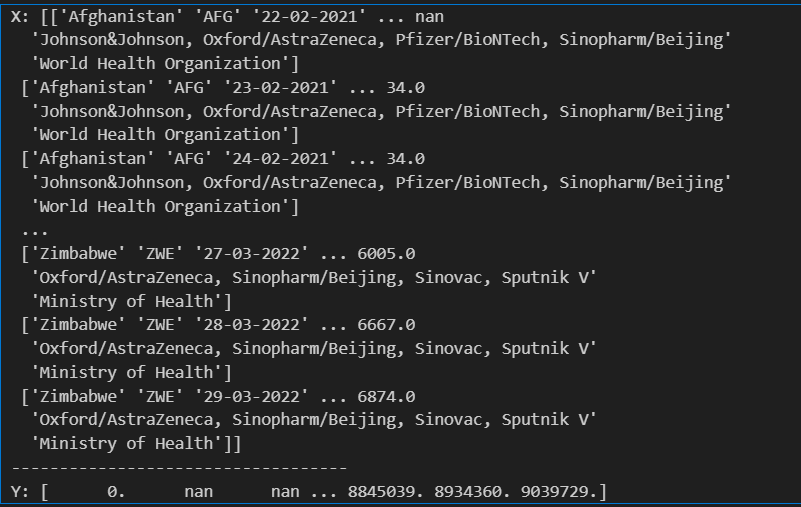
Y = dataset.iloc[:,3].values

print ('X: %s'%(str(X)))

print ('-----------------------------------')

print ('Y: %s'%(str(Y)))

**Output:**



**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.

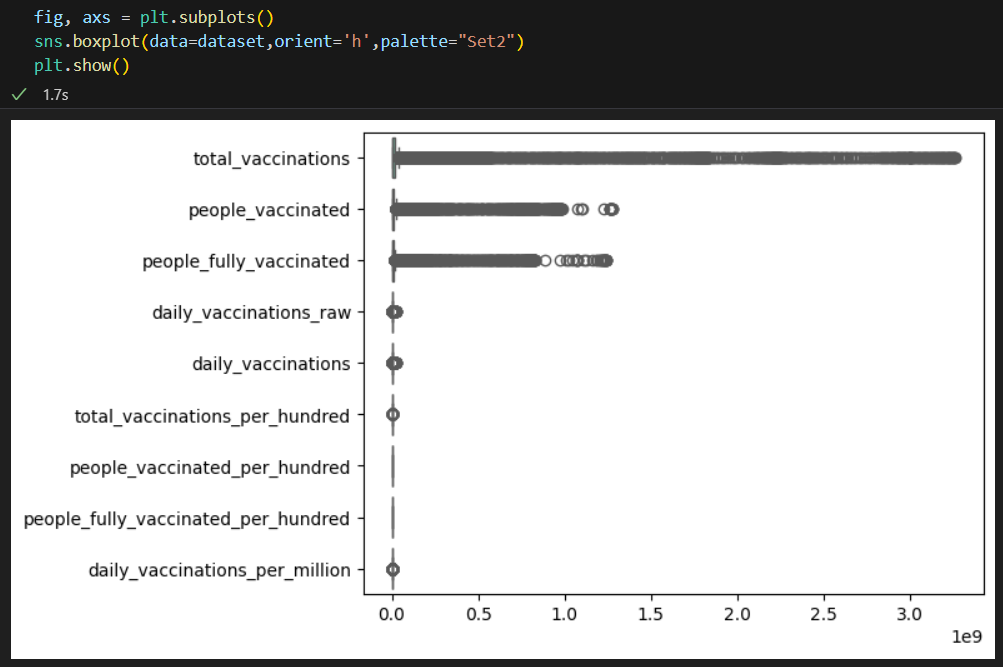
**Code:**

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.

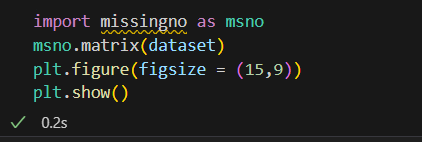
**Code:**

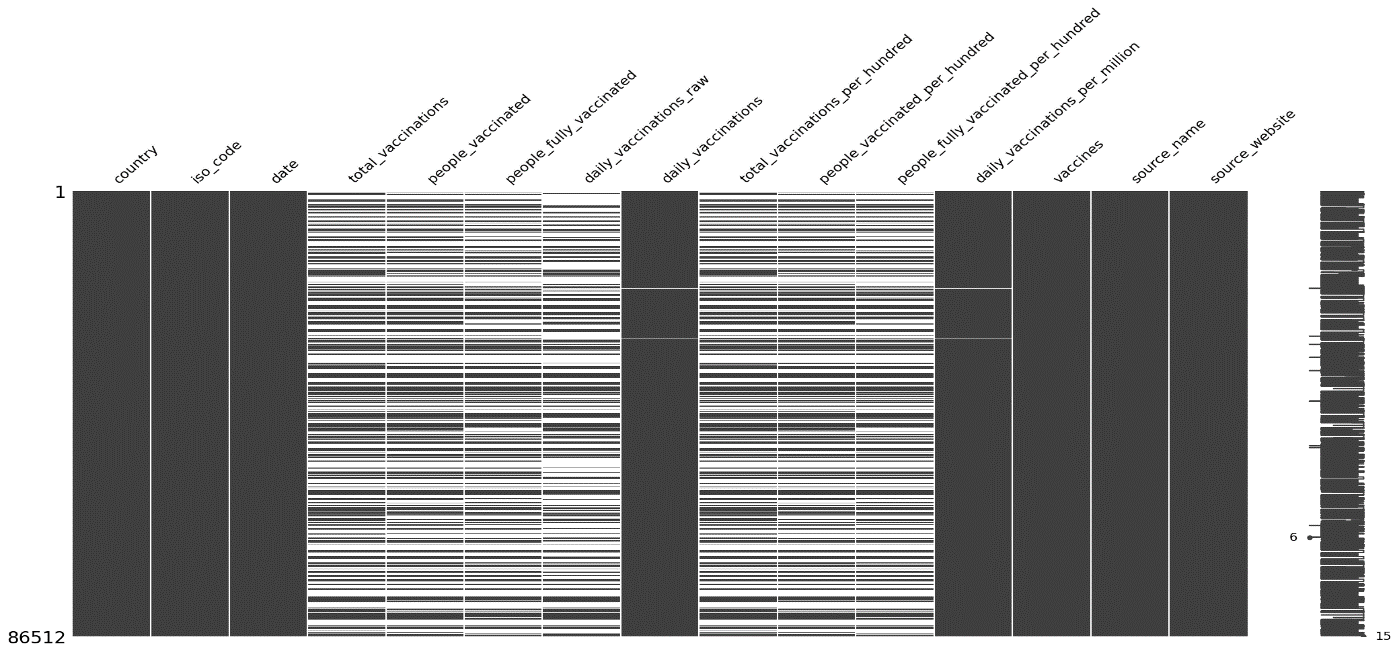
import missingno as msno

msno.matrix(dataset)

plt.figure(figsize = (15,9))

plt.show()

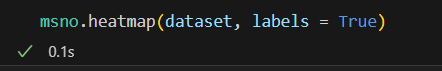
**Output:**

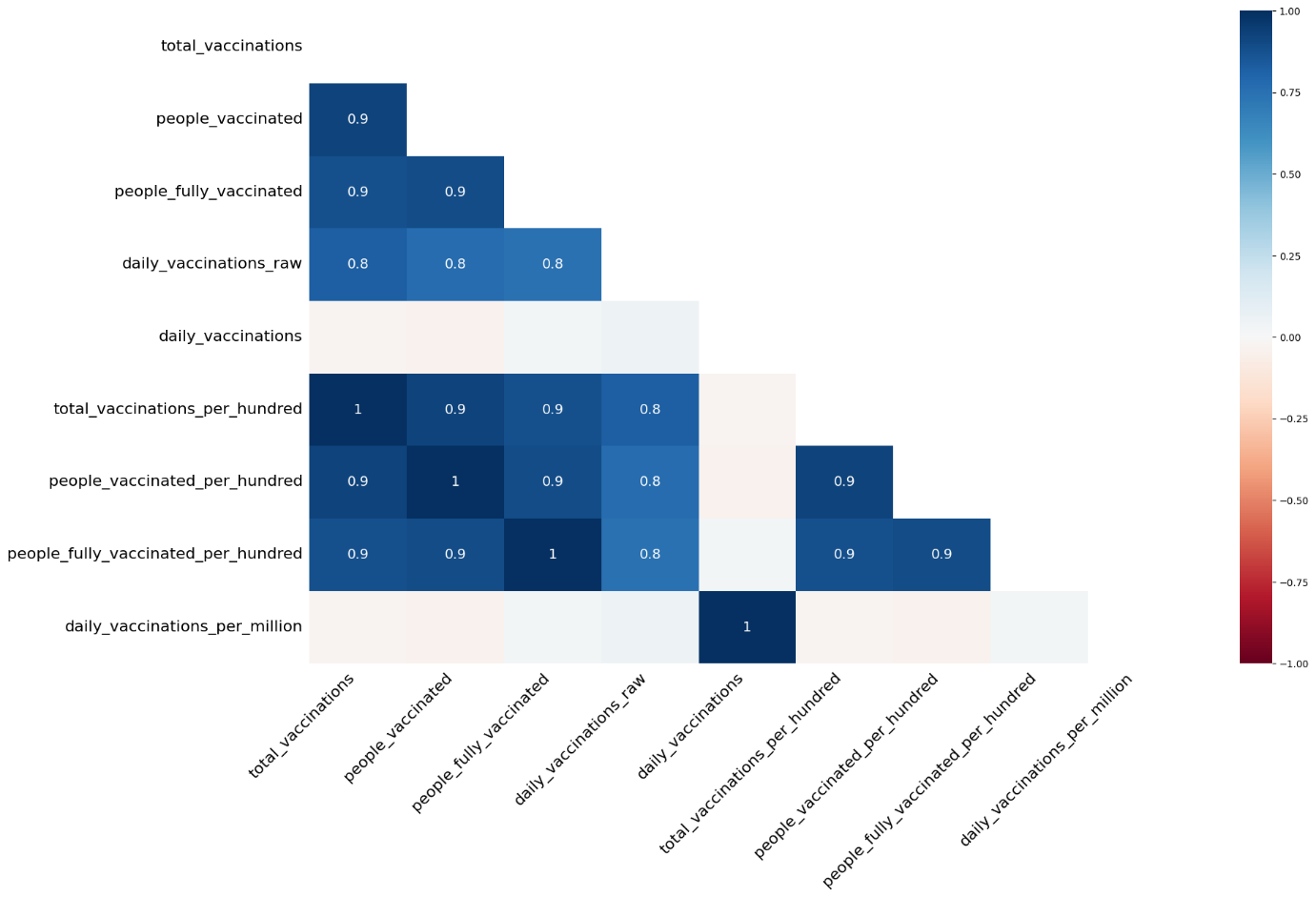
****

**Code:**

msno.heatmap(dataset, labels = True)

**Output:**

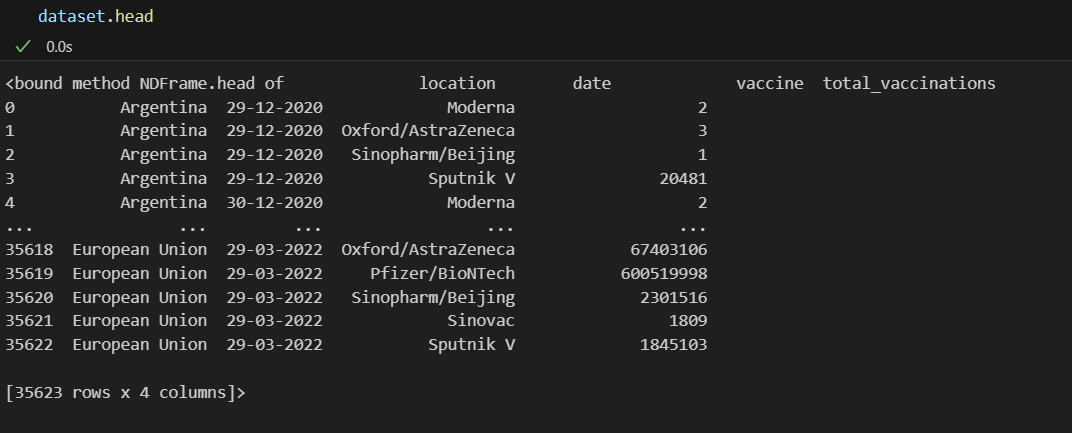




**Load the dataset: (DATASET 2):**

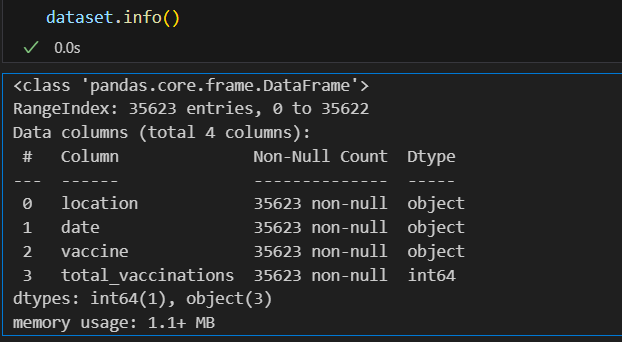
**Head() Function:**

**Output:**

****

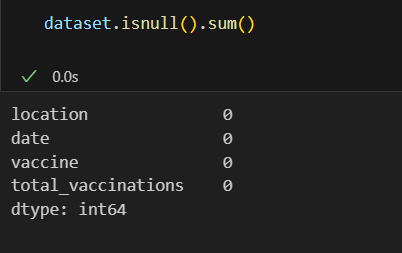
**Info() Function:**

**Output:**

****

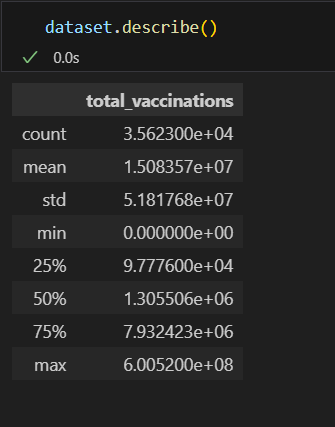
**Df.isnull().sum() Function:**

**Output:**



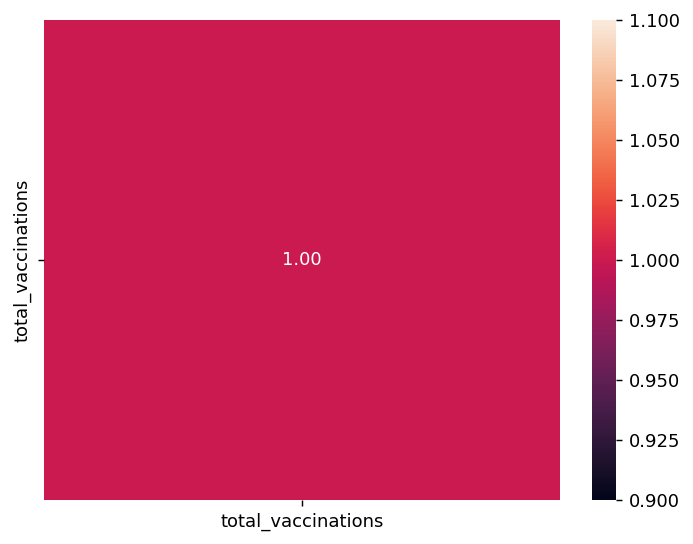
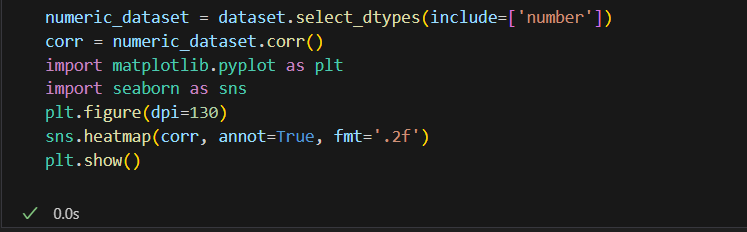
**Describe Function:**

**Output:**

****

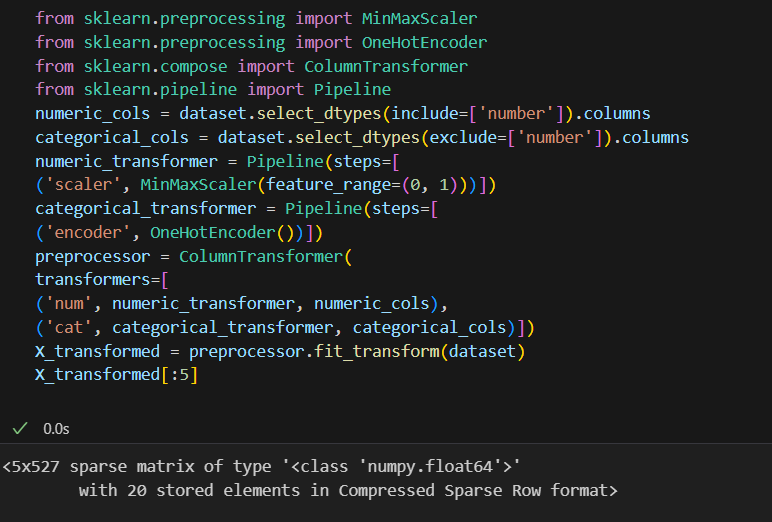
**Corelation:**

**Output:**

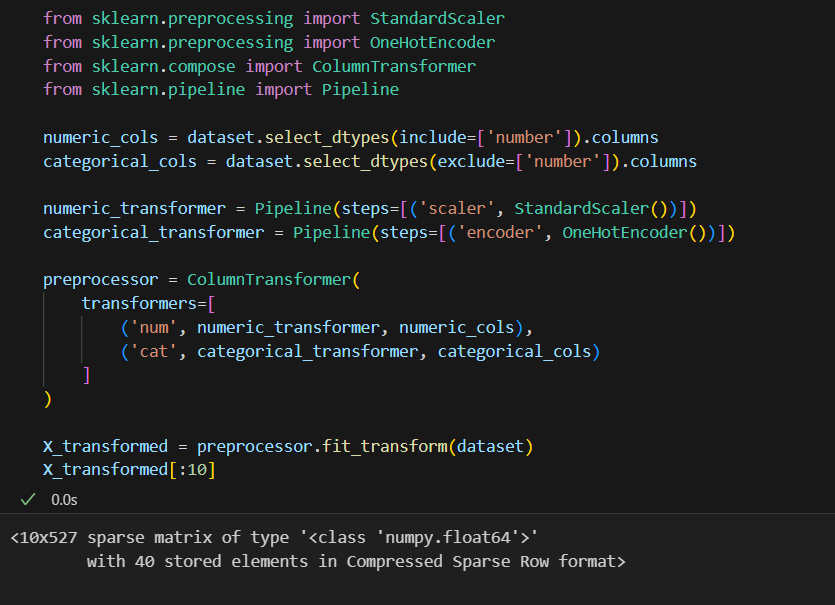
****

**Normalization:**

**Output:**

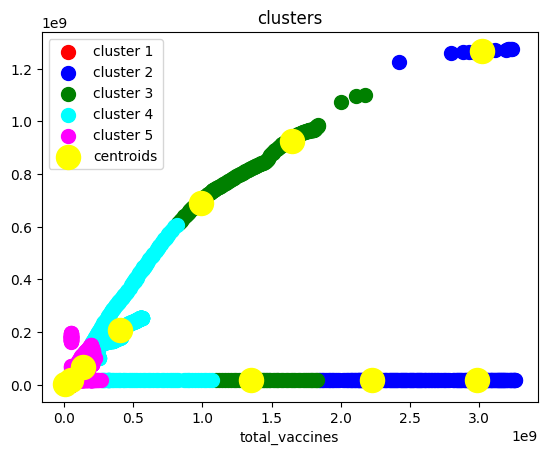
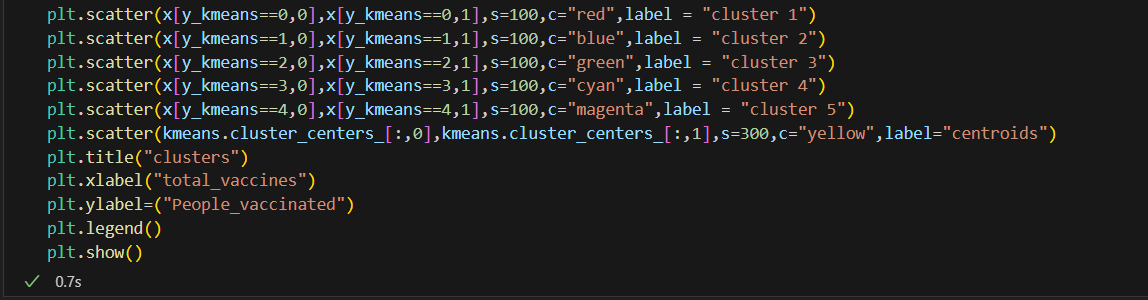
****

**Standardization:**

**Output:**

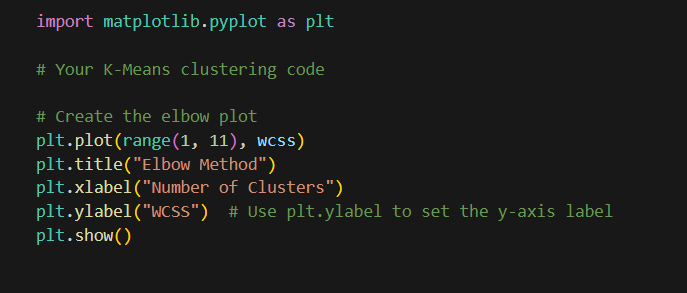
**K-means Clustering Function:**

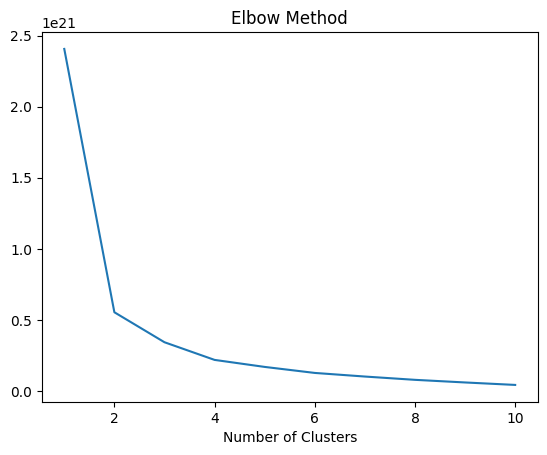
**Output:**

****

**WCSS Function:**

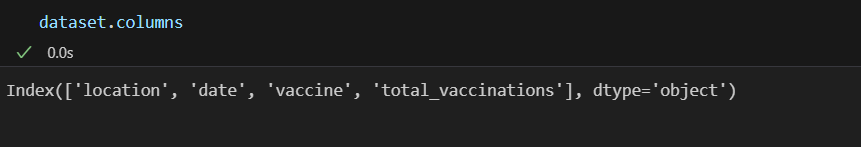
**Output:**

****

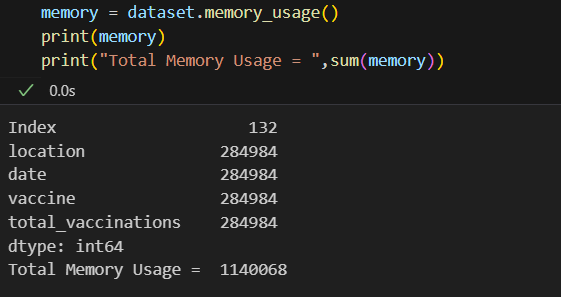
****

**Dataset.columns:**

**Output:**

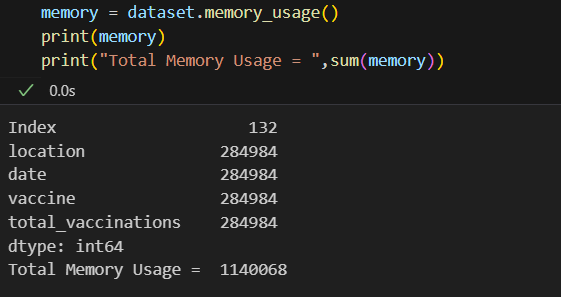
****

**Memory Function:**

**Output:**

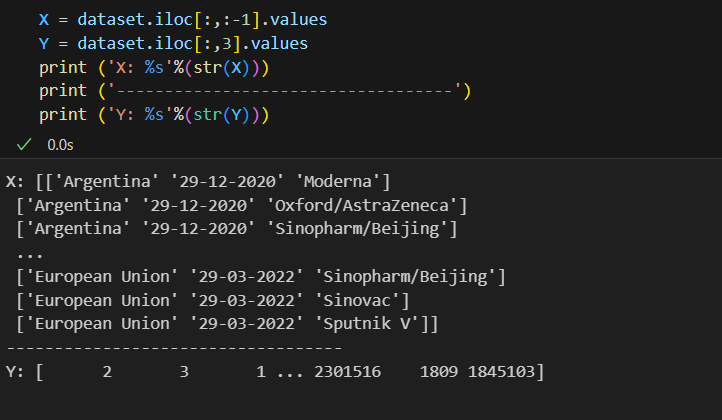
**Dropna() Function:**

**Output:**

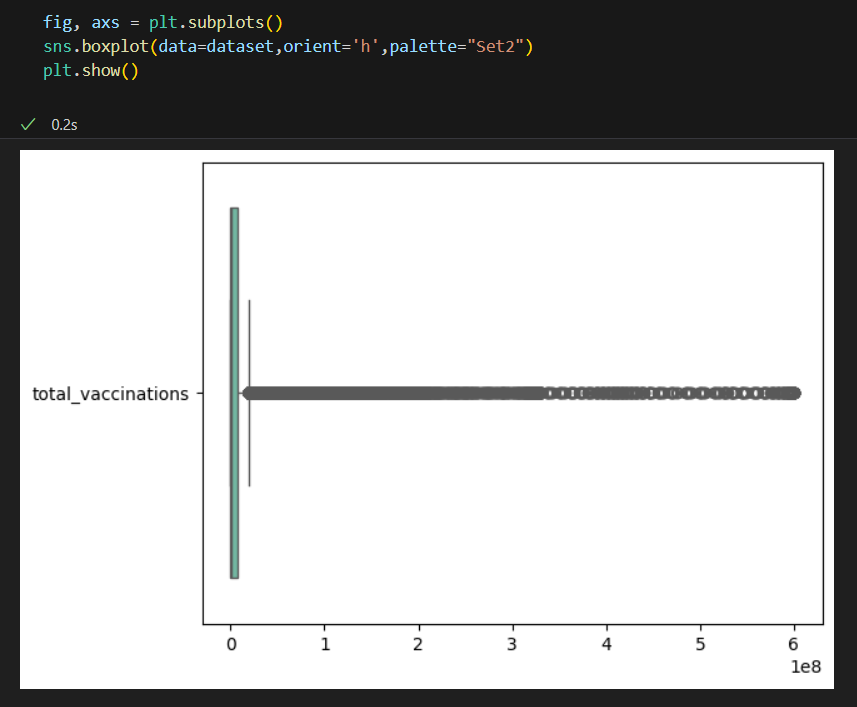
****

**Iloc() Function:**

**Output:**

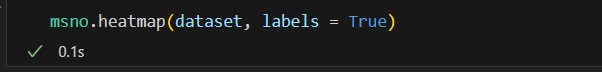
****

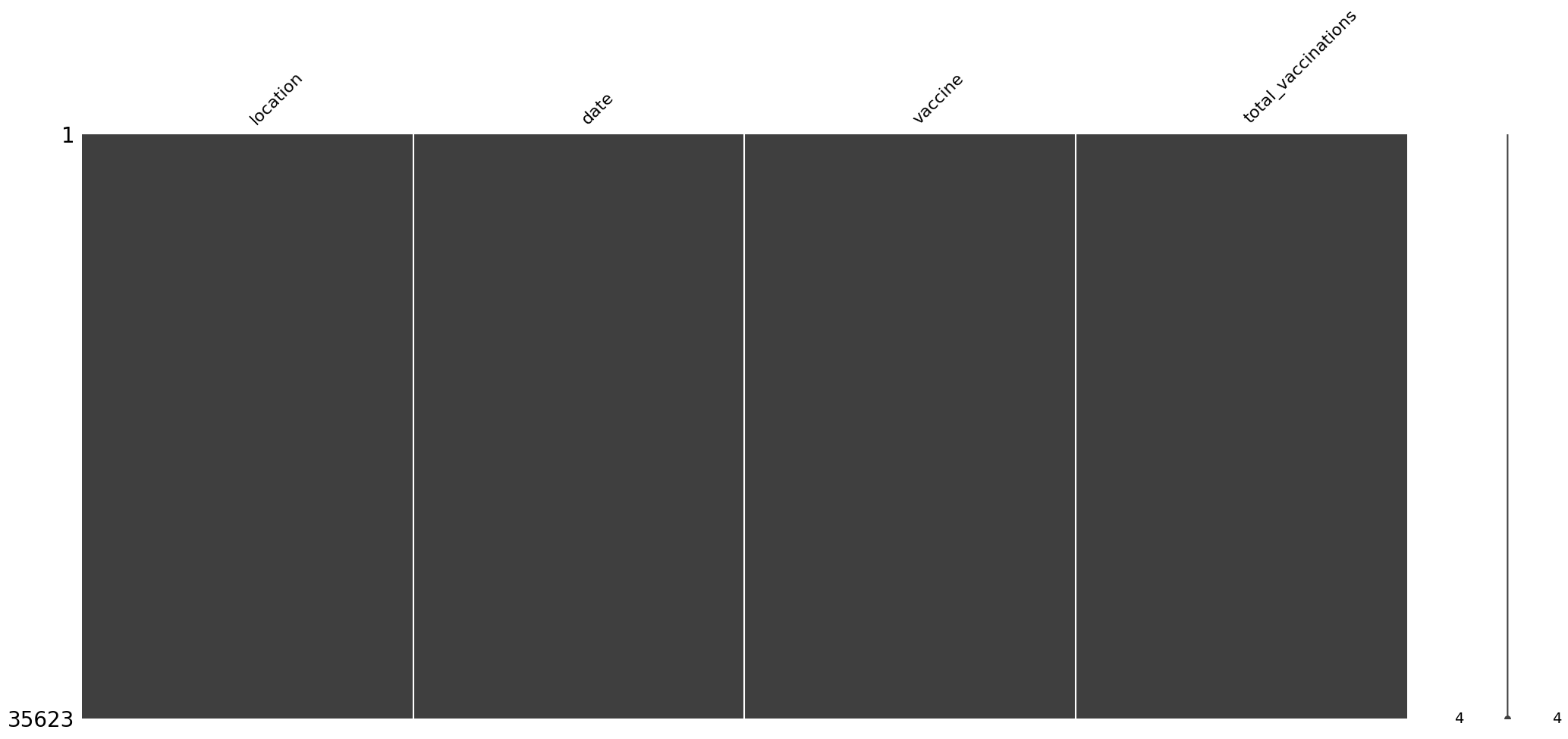
**Subplots Function:**

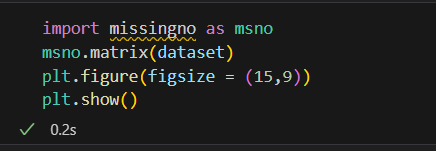
**Output:**

**Missingno Function:**

**Output:**

****

****



**Phase 4:** **Development**

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

**Part 2**

In this part you will begin building your project by

Begin conducting the Covid-19 vaccines analysis by Performing exploratory data analysis Statistical analysis

Visualization.

**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:**

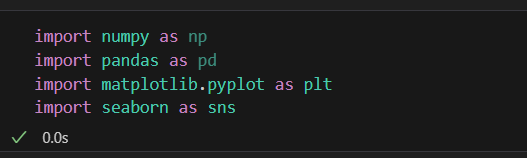
import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

**Output:**



**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\_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

**Output:**

****

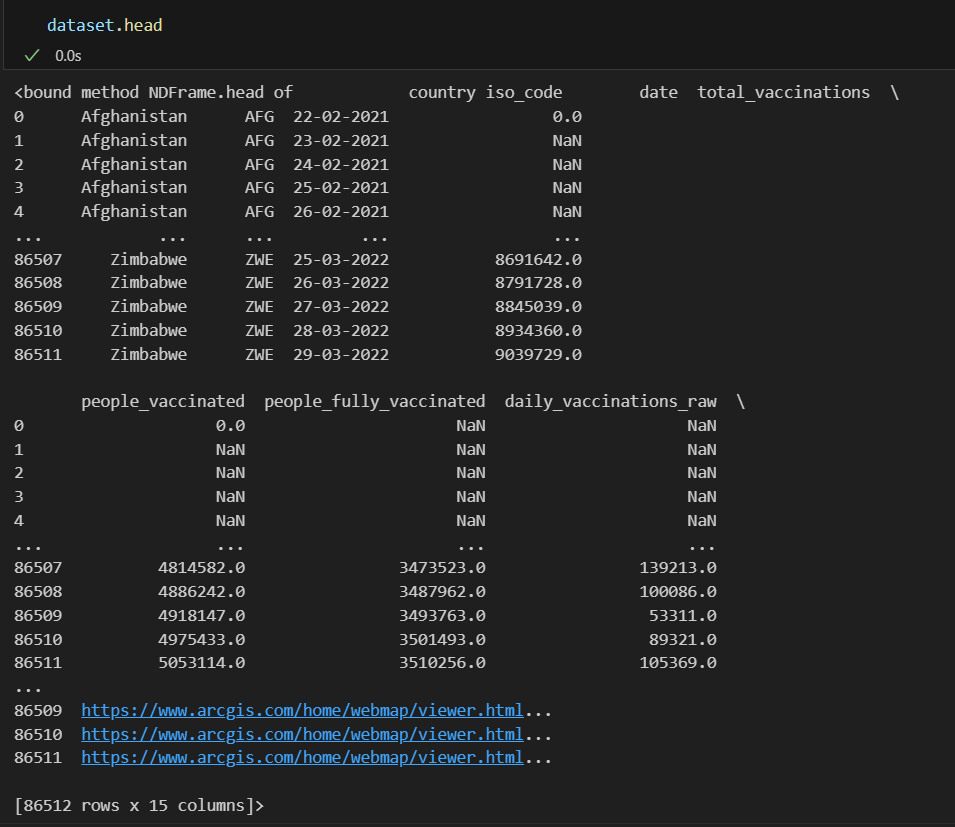
**Head() Function:**

* 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:**



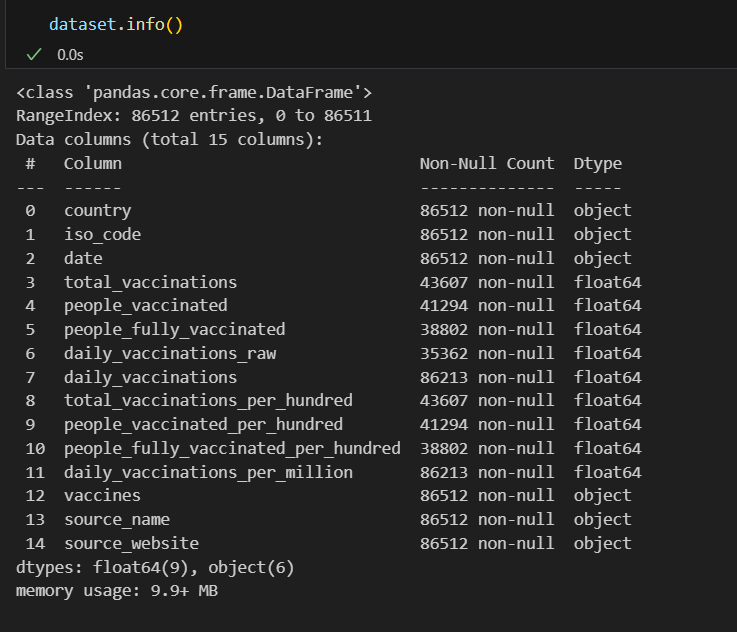
**Info() Function:**

* 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:**



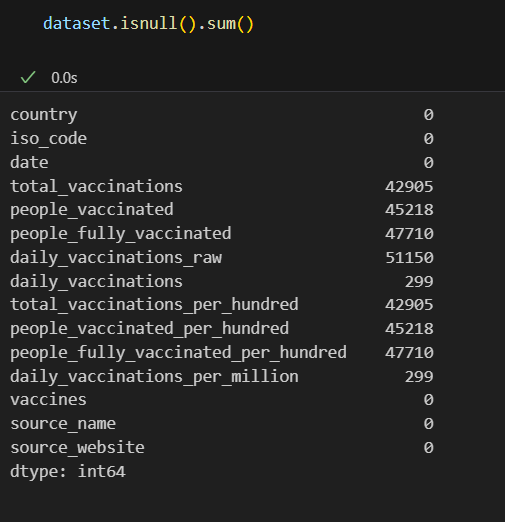
**Df.isnull().sum() Function:**

* 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.Top of Form

**Code:**

dataset.isnull().sum()

**Output:**



**Describe Function:**

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

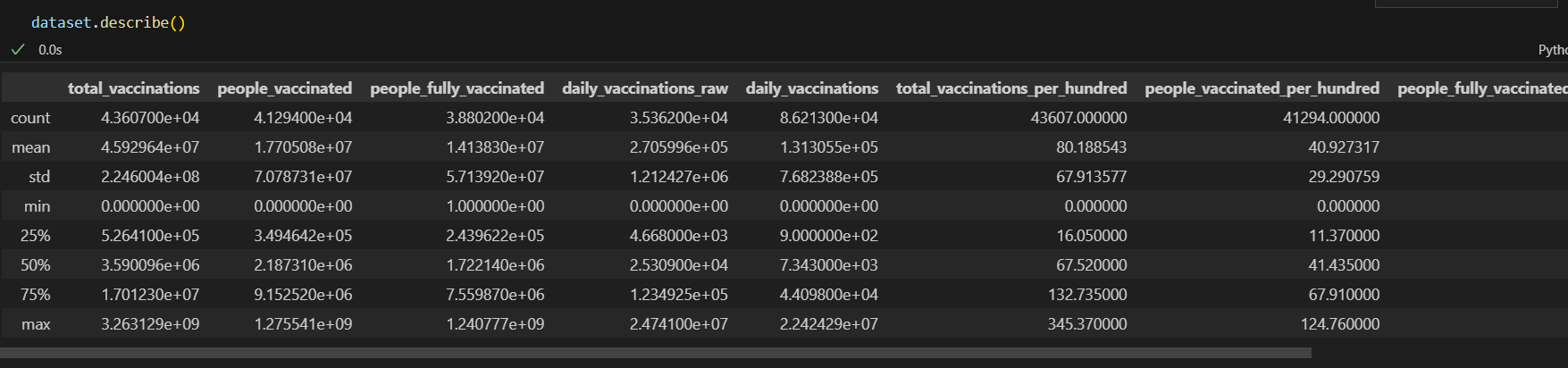
It provides a quick overview of the key statistics for numerical data in the dataset, including:

* **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:**

dataset.describe()

**Output:**

****

**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.

**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.

**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).

Here are some common tasks and techniques involved in data preprocessing

**Outliers:**

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

**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

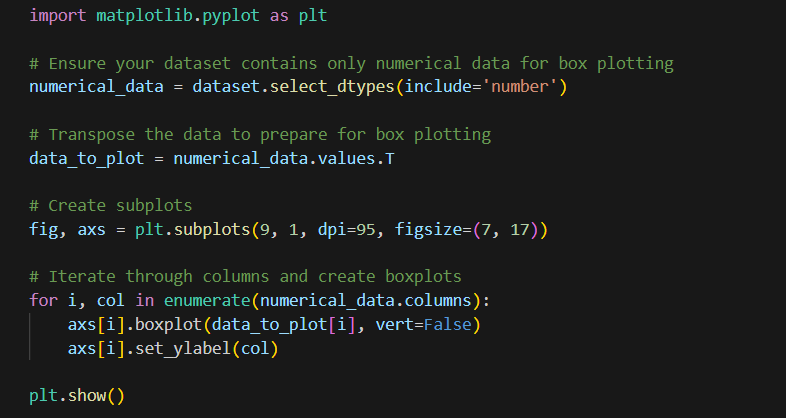
for i, col in enumerate(numerical\_data.columns):

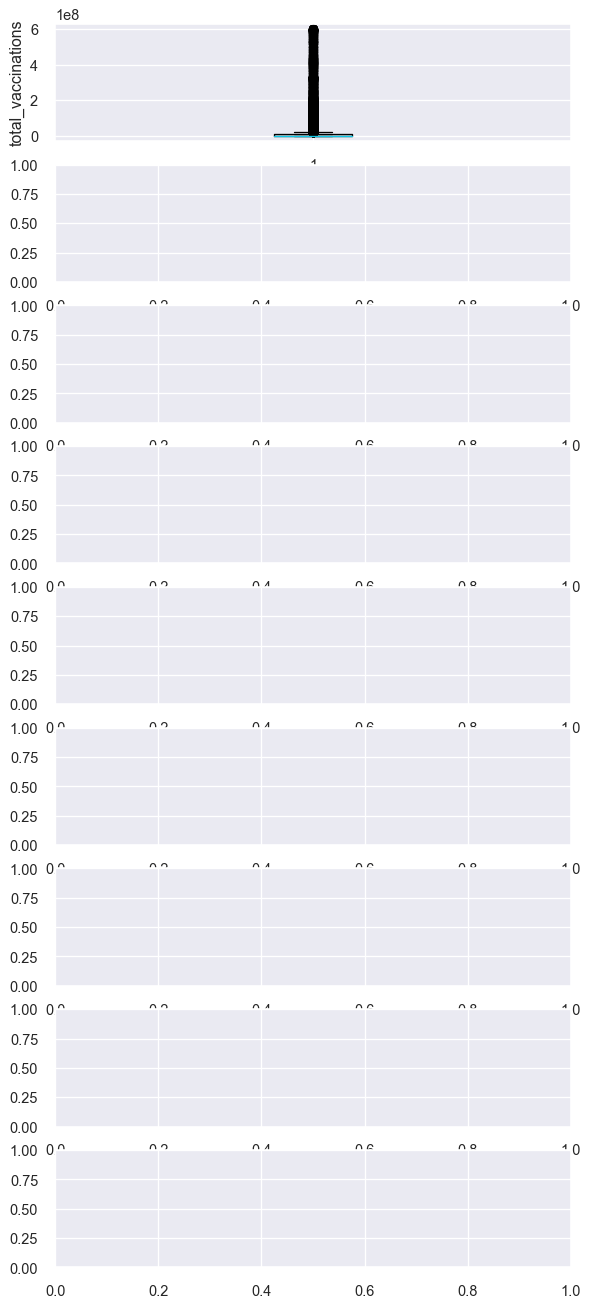
    axs[i].boxplot(data\_to\_plot[i], vert=False)

    axs[i].set\_ylabel(col)

plt.show()

**Output:**

****

**SCorelation:**

* 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

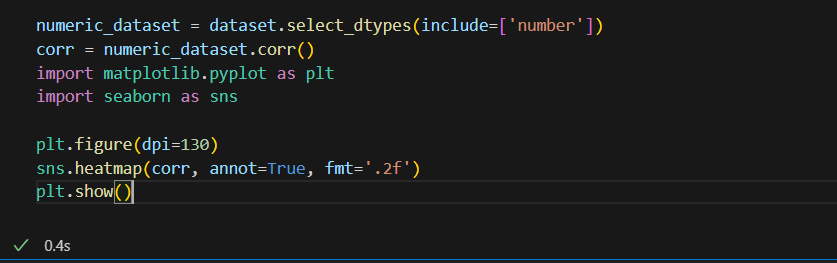
import seaborn as sns

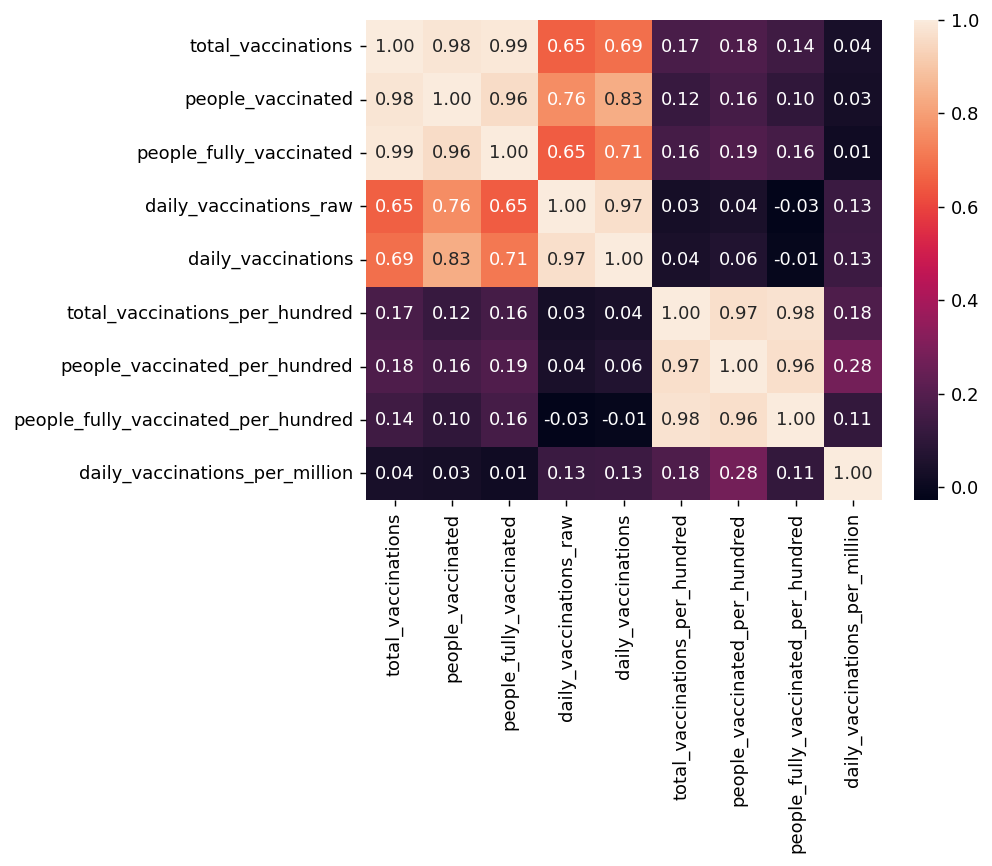
plt.figure(dpi=130)

sns.heatmap(corr, annot=True, fmt='.2f')

plt.show()

**Output:**

****

****

**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 MinMaxScalar.

**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=[

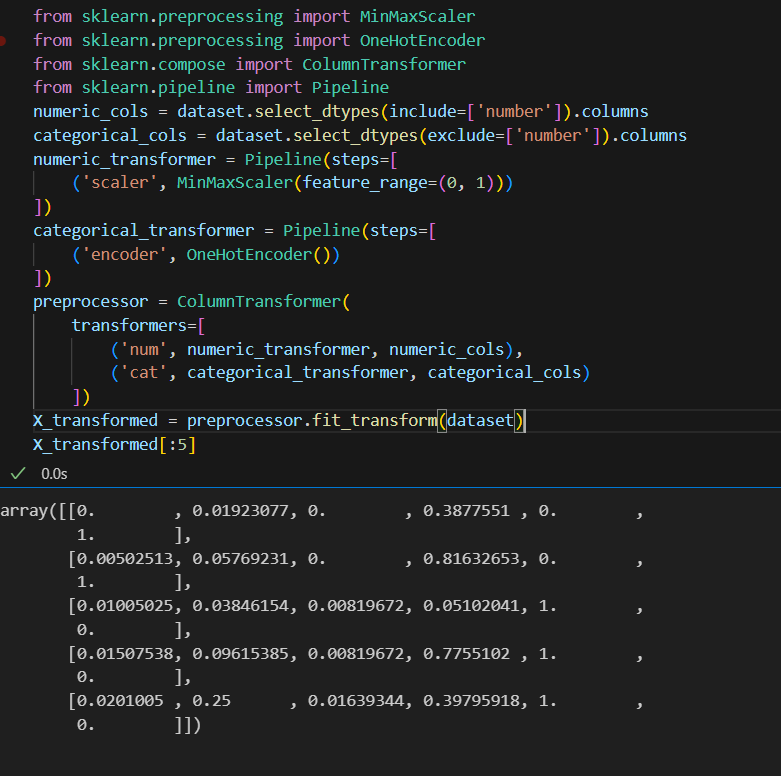
('num', numeric\_transformer, numeric\_cols),

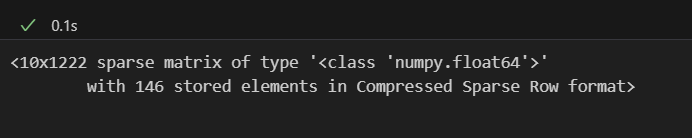
('cat', categorical\_transformer, categorical\_cols)])

X\_transformed = preprocessor.fit\_transform(dataset)

X\_transformed[:5]

**Output:**



****

**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.

**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 StandardScalar 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=[

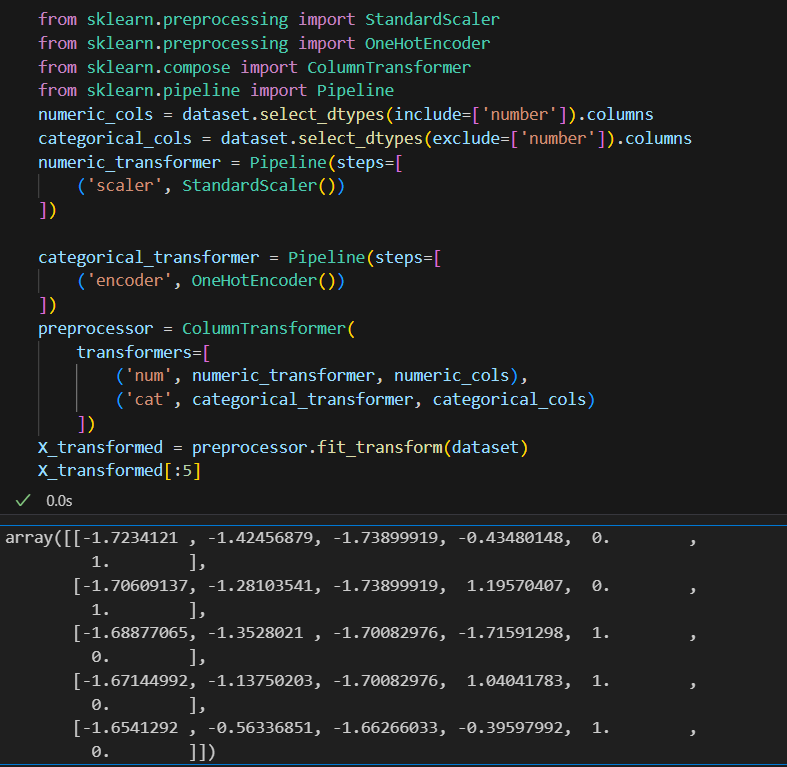
('num', numeric\_transformer, numeric\_cols),

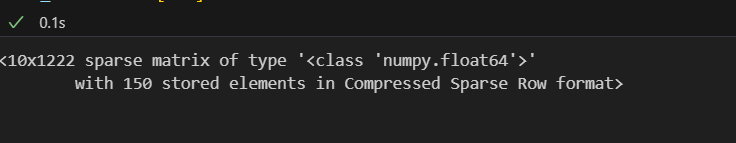
('cat', categorical\_transformer, categorical\_cols])

X\_transformed = preprocessor.fit\_transform(dataset)

X\_transformed[:5]

**Output:**

****



**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 = "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")

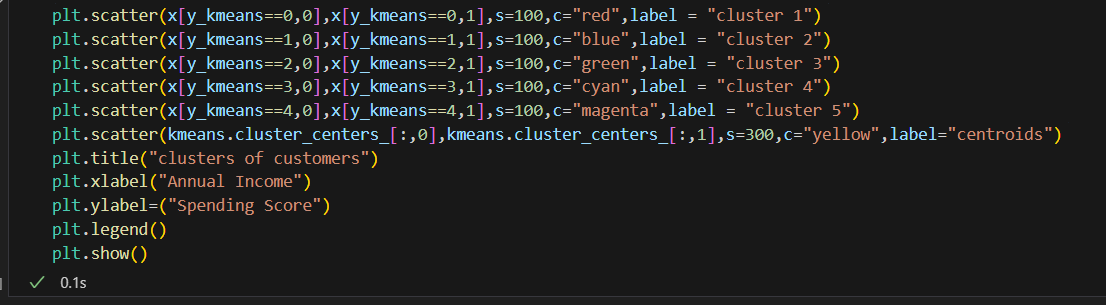
plt.xlabel("Annual Income")

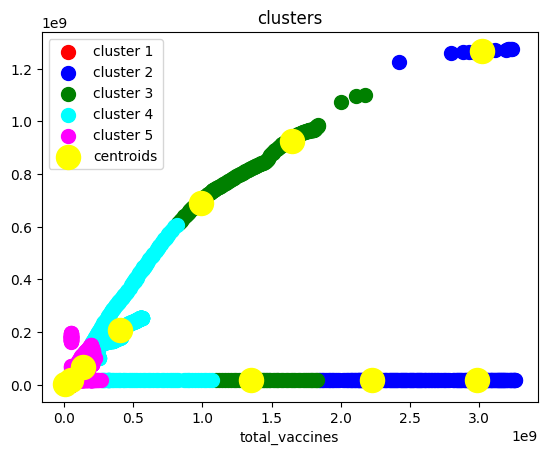
plt.ylabel=("Spending Score")

plt.legend()

plt.show()

**Output:**





**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)

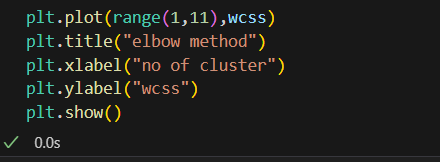
plt.title("elbow method")

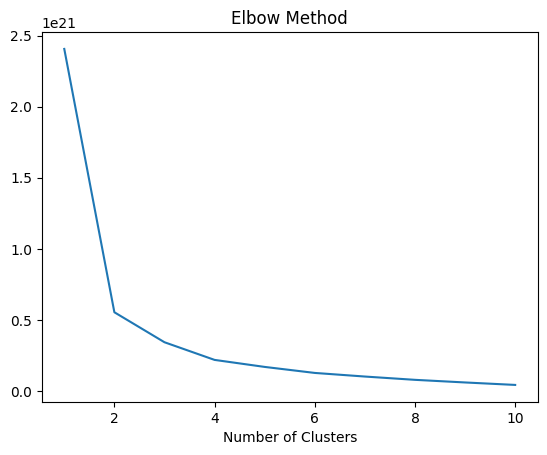
plt.xlabel("no of cluster")

plt.ylabel("wcss")

plt.show()

**Output:**

****



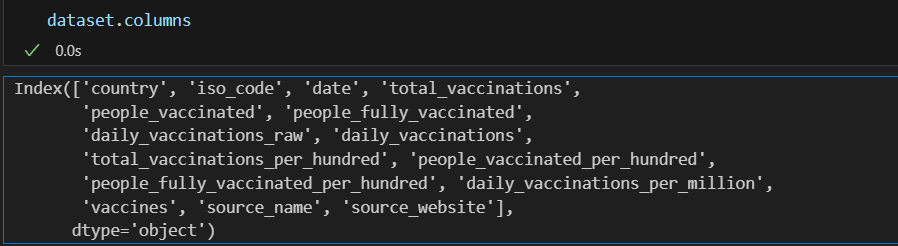
**Dataset.columns :**

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

**Code:**

dataset.columns

**Output:**

****

**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.

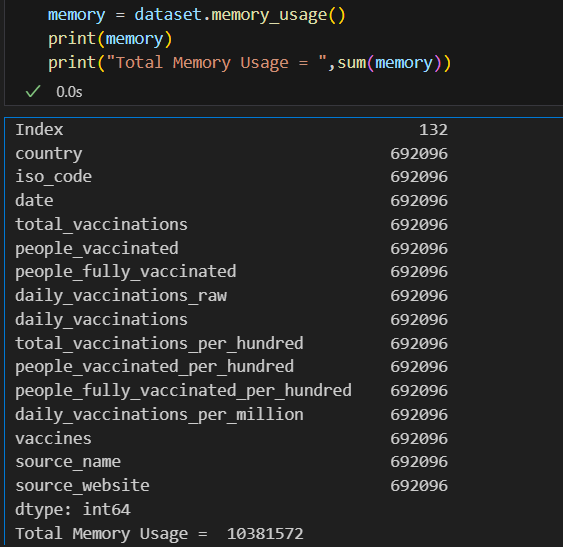
**Code:**

memory = dataset.memory\_usage()

print(memory)

print("Total Memory Usage = ",sum(memory))

**Output:**



**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 machine learning models

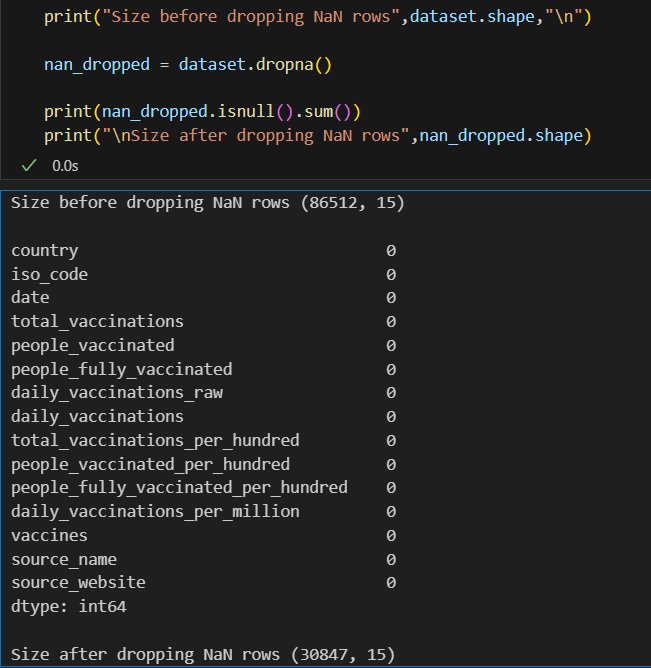
**Code:**

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:**

**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:**

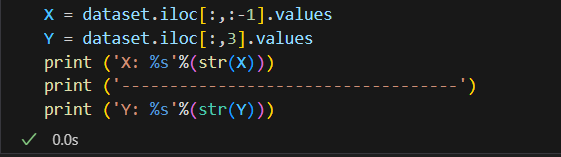
X = dataset.iloc[:,:-1].values

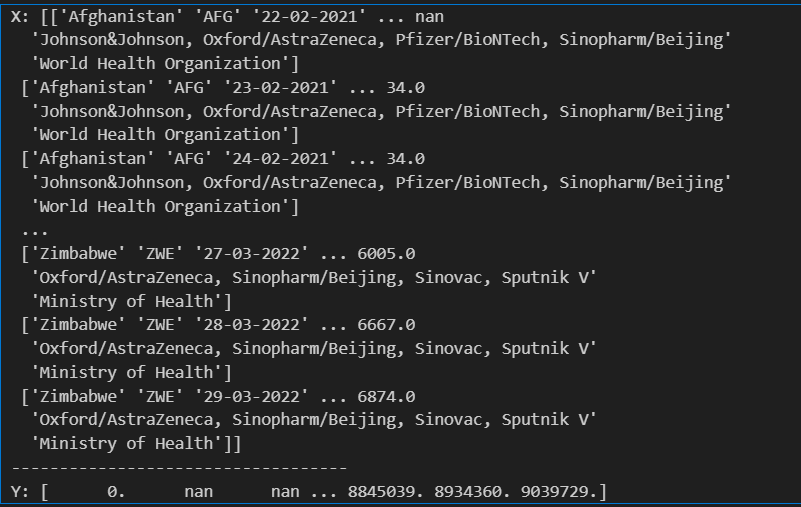
Y = dataset.iloc[:,3].values

print ('X: %s'%(str(X)))

print ('-----------------------------------')

print ('Y: %s'%(str(Y)))

**Output:**



**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.

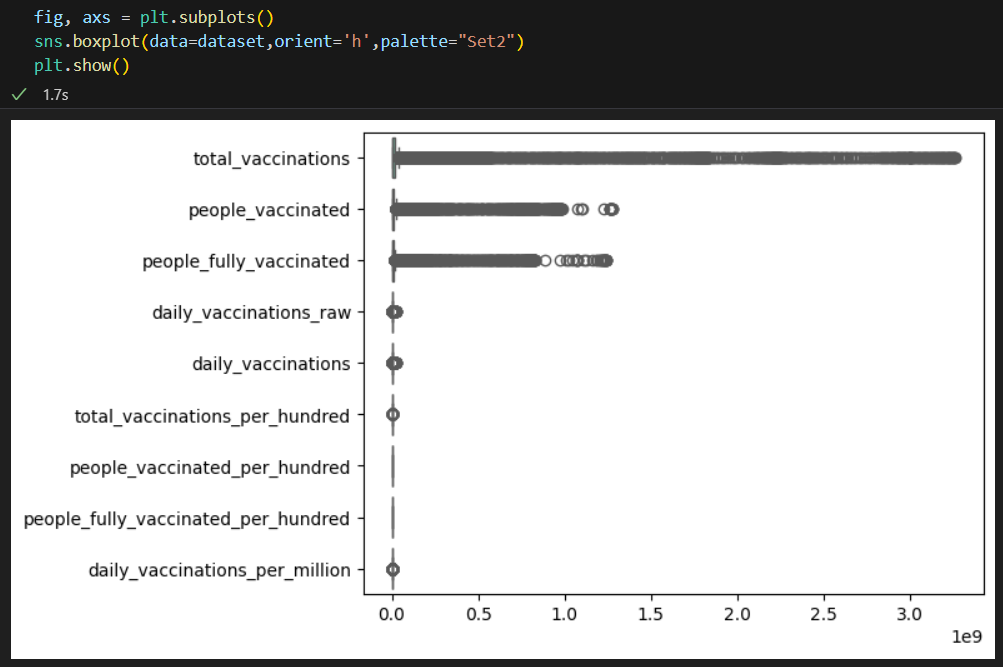
**Code:**

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.

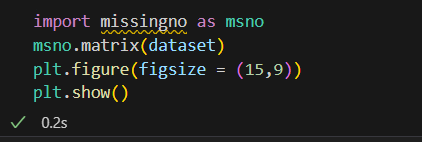
**Code:**

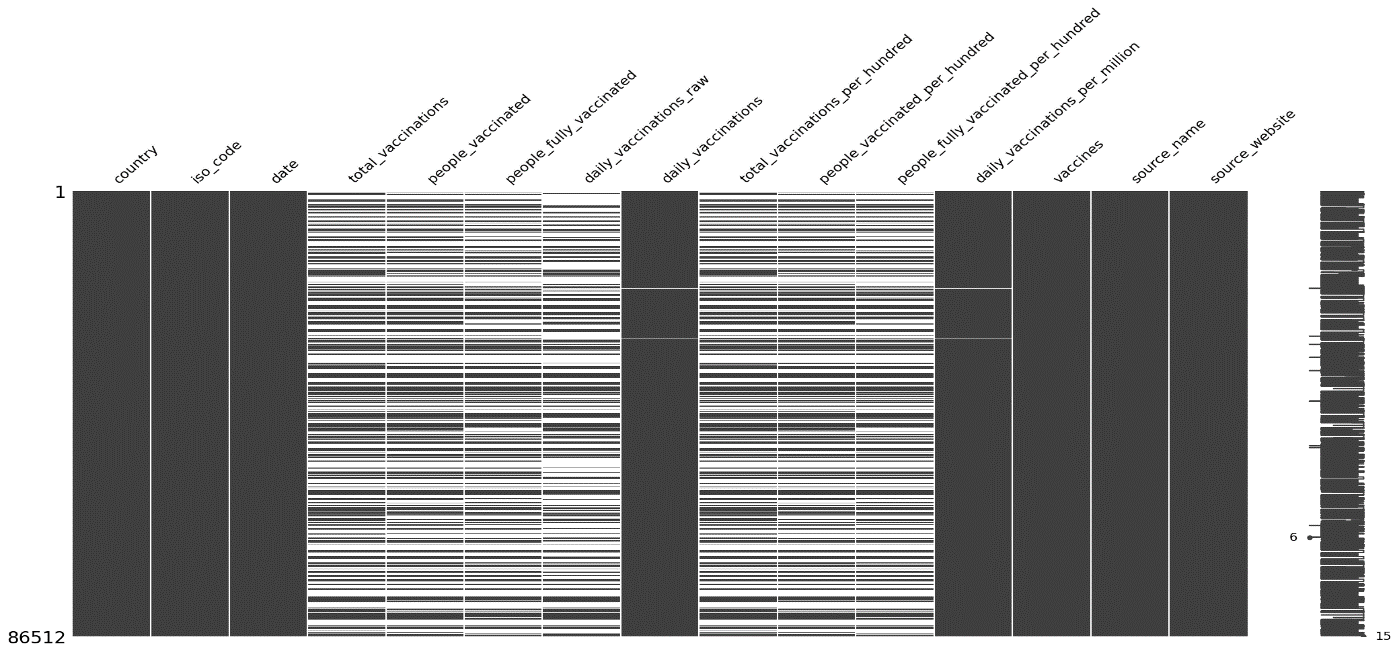
import missingno as msno

msno.matrix(dataset)

plt.figure(figsize = (15,9))

plt.show()

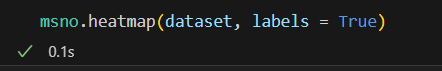
**Output:**

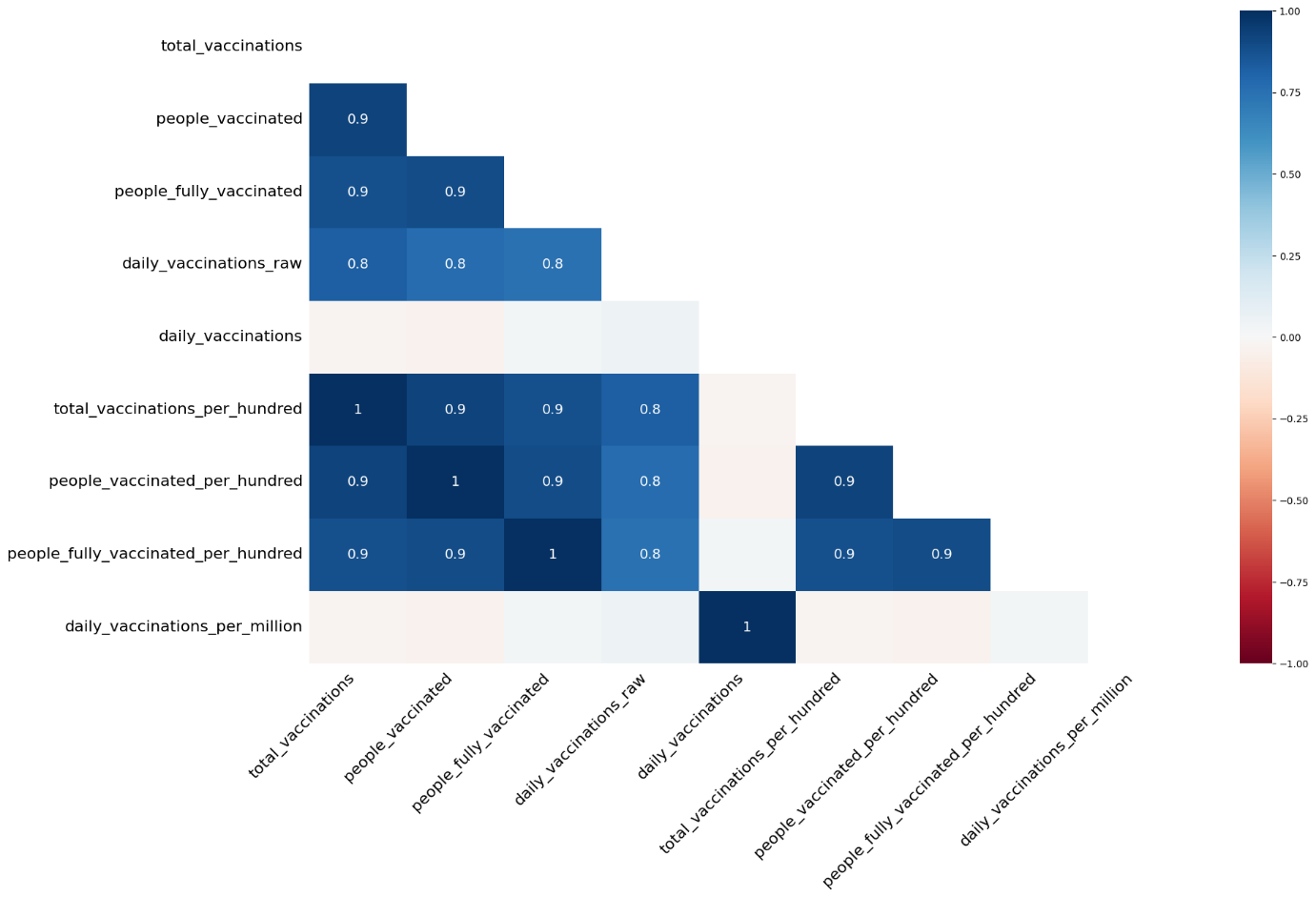
****

**Code:**

msno.heatmap(dataset, labels = True)

**Output:**

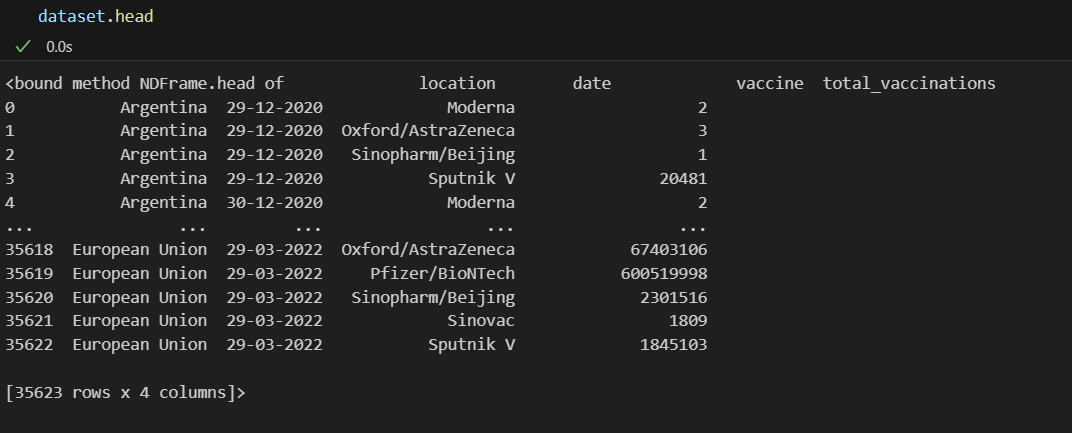




**Load the dataset: (DATASET 2):**

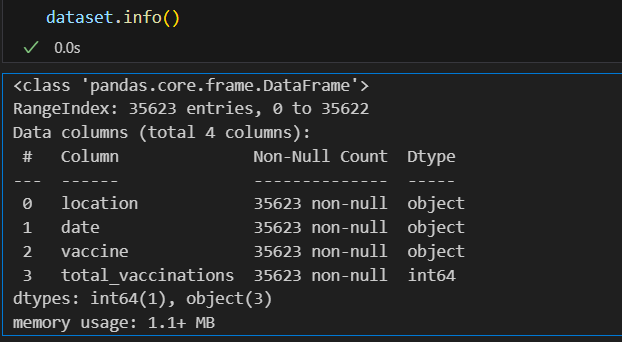
**Head() Function:**

**Output:**

****

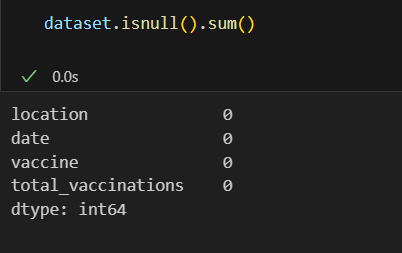
**Info() Function:**

**Output:**

****

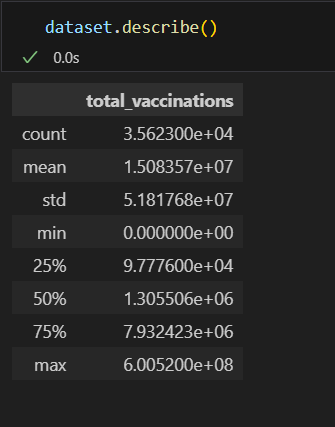
**Df.isnull().sum() Function:**

**Output:**



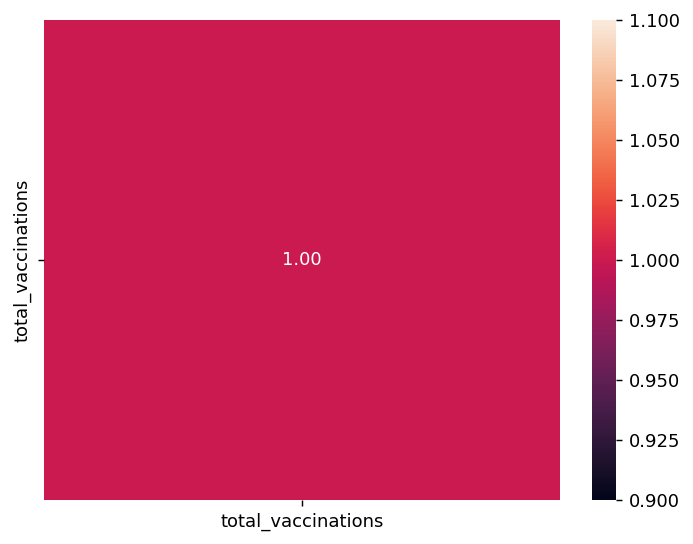
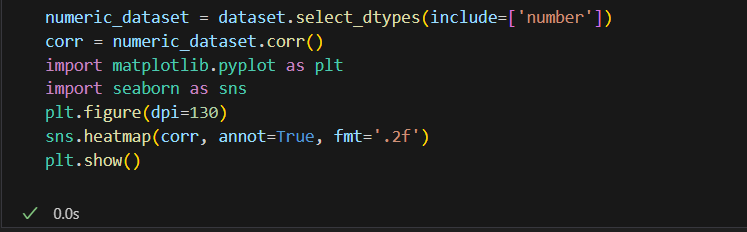
**Describe Function:**

**Output:**

****

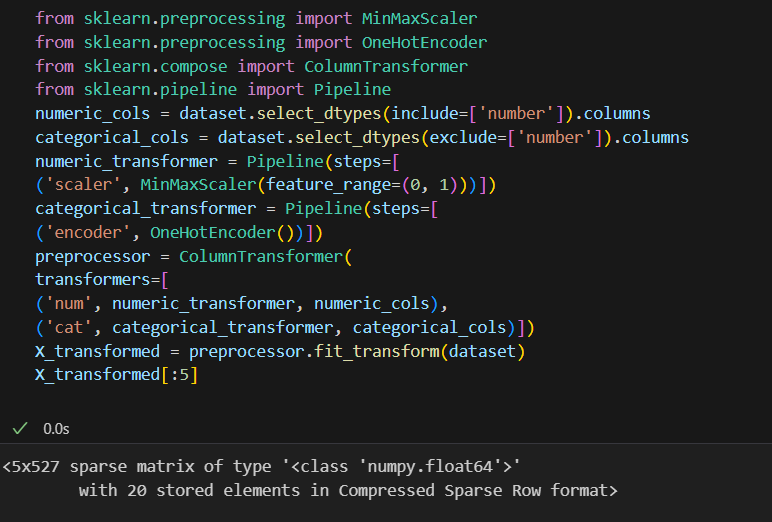
**Corelation:**

**Output:**

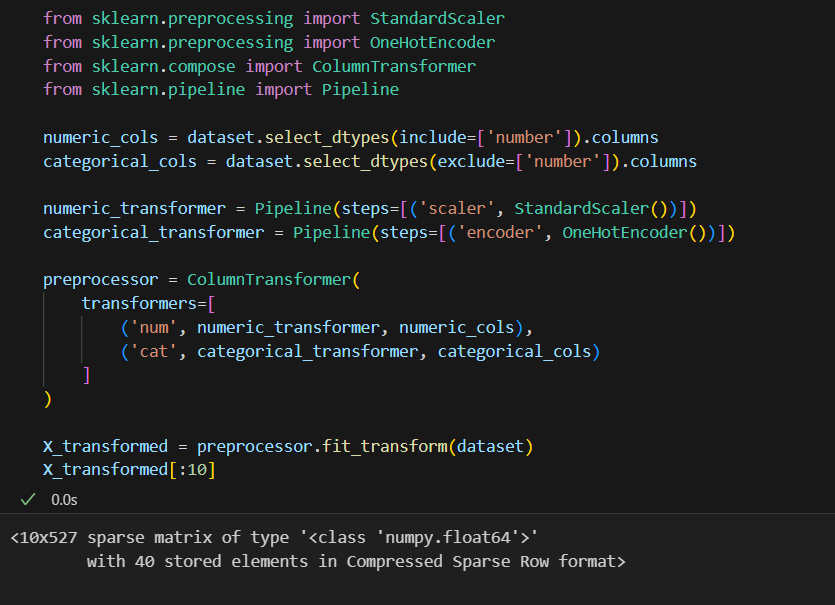
****

**Normalization:**

**Output:**

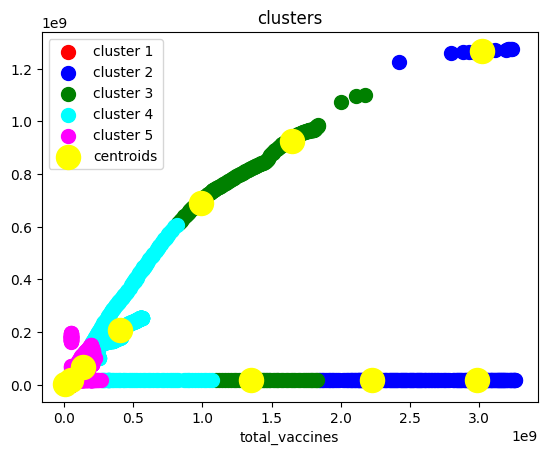
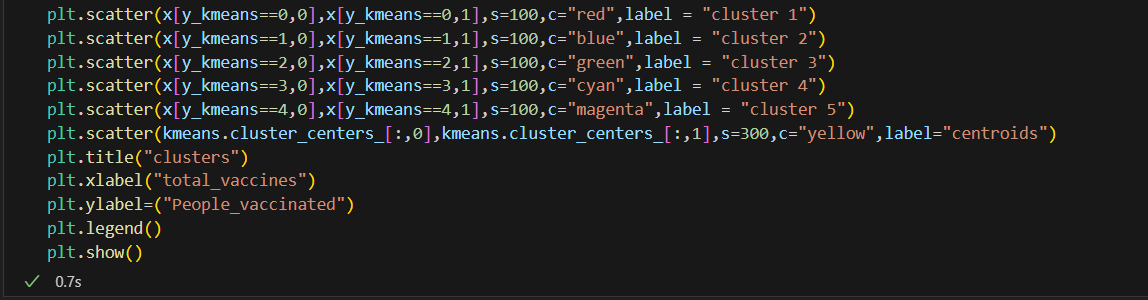
****

**Standardization:**

**Output:**

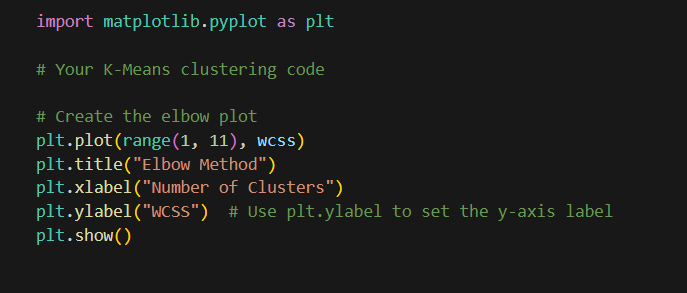
**K-means Clustering Function:**

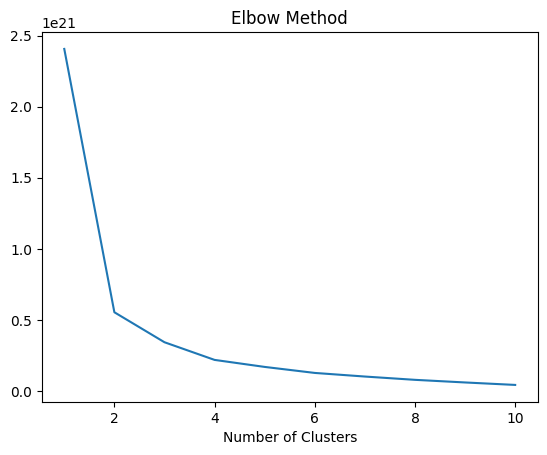
**Output:**

****

**WCSS Function:**

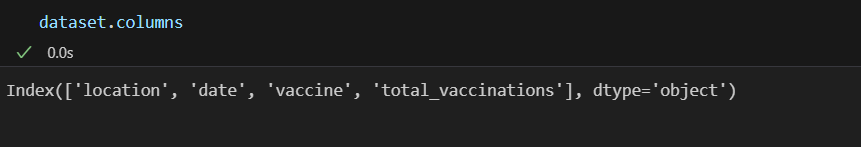
**Output:**

****

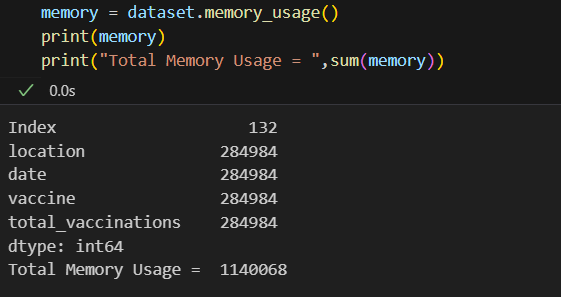
****

**Dataset.columns:**

**Output:**

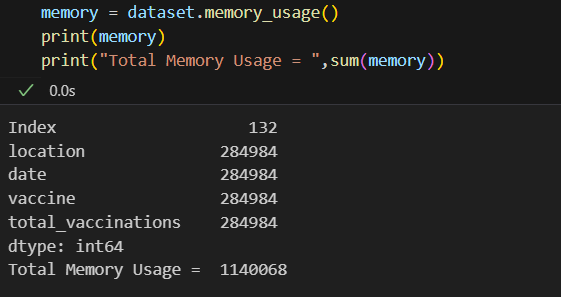
****

**Memory Function:**

**Output:**

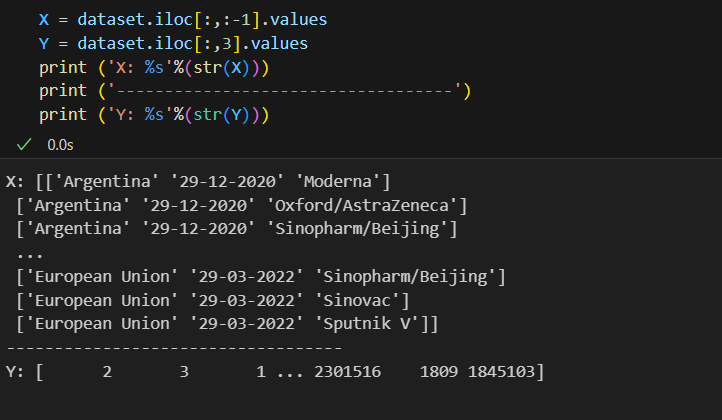
**Dropna() Function:**

**Output:**

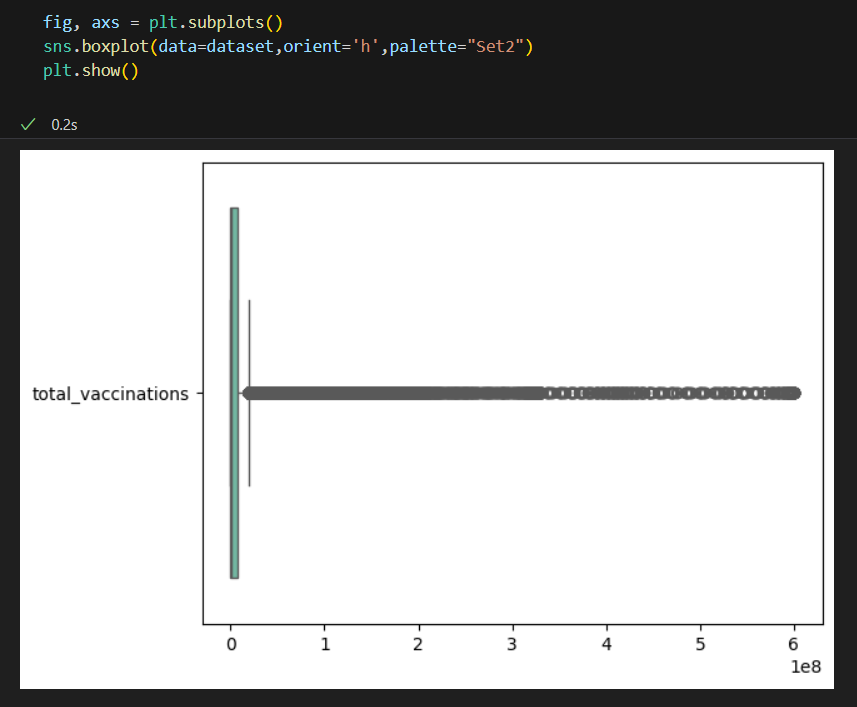
****

**Iloc() Function:**

**Output:**

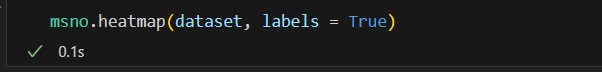
****

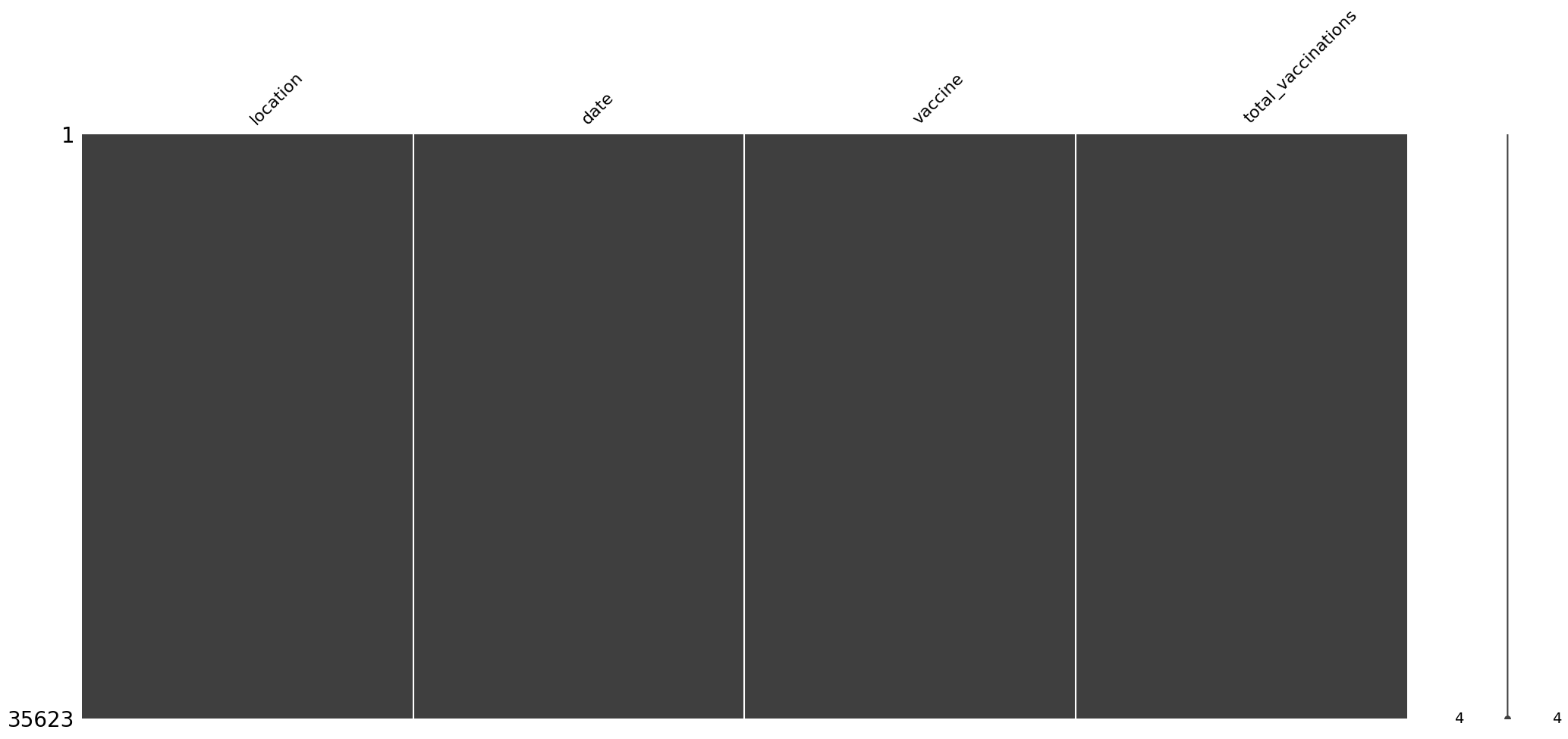
**Subplots Function:**

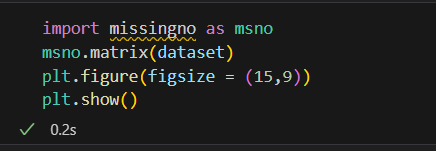
**Output:**

**Missingno Function:**

**Output:**

****

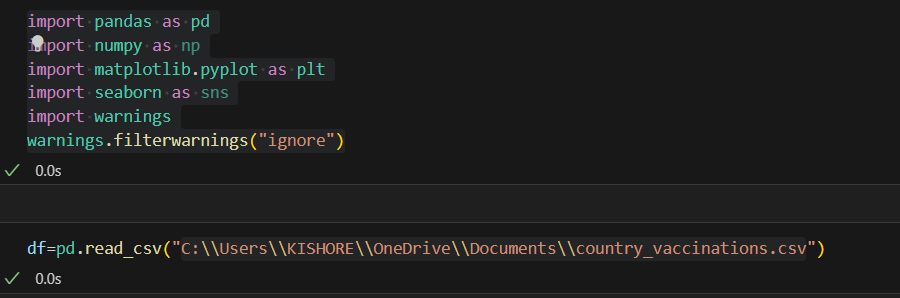
****

S

**Data Scaling:**

* Data scaling, also known as feature scaling or data normalization, is a crucial preprocessing step in data analysis and machine learning.
* It involves transforming the values of different features (variables or columns) in your dataset to a common scale. Here are some important notes about data scaling:
* Data scaling is important because many machine learning algorithms are sensitive to the scale of input features. Rescaling the data helps these algorithms perform better.
* It can help prevent features with larger scales from dominating or biasing the learning process.

**Importing the required libraries:**



**Code:**

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

import warnings

warnings.filterwarnings("ignore")

df=pd.read\_csv("C:\\Users\\KISHORE\\OneDrive\\Documents\\country\_vaccinations.csv")

**Standardization**

* Standardization is a scaling technique where the values are centered around the mean with a unit standard deviation.
* This means that the mean of the attribute becomes zero and the resultant distribution has a unit standard deviation.

**Code**:

import pandas as pd

from sklearn.preprocessing import MaxAbsScaler

# Read the data

df = pd.read\_csv("C:\\Users\\KISHORE\\OneDrive\\Documents\\country\_vaccinations.csv")

# Drop the 'Age' column

df.drop("date", axis=1, inplace=True)

# Select only the numeric columns for scaling

numeric\_columns = [ 'daily\_vaccinations\_per\_million', 'daily\_vaccinations']

numeric\_data = df[numeric\_columns]

# Apply MaxAbsScaler to the numeric data

maxabsscaler = MaxAbsScaler()

scaled\_data = pd.DataFrame(maxabsscaler.fit\_transform(numeric\_data), columns=numeric\_columns)

# Combine the scaled numeric data with the non-numeric data

df = pd.concat([df.drop(numeric\_columns, axis=1), scaled\_data], axis=1)

ax=df.plot.scatter(x="daily\_vaccinations",y="daily\_vaccinations\_per\_million",marker="\*",label="Before-Scaling",color="red")

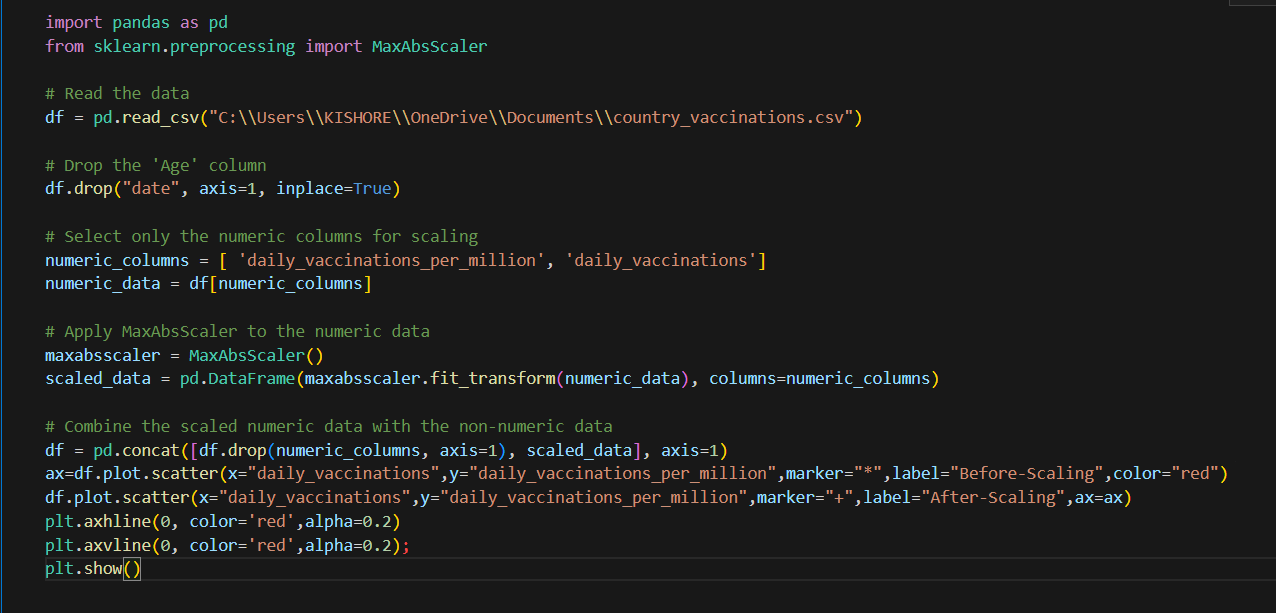
df.plot.scatter(x="daily\_vaccinations",y="daily\_vaccinations\_per\_million",marker="+",label="After-Scaling",ax=ax)

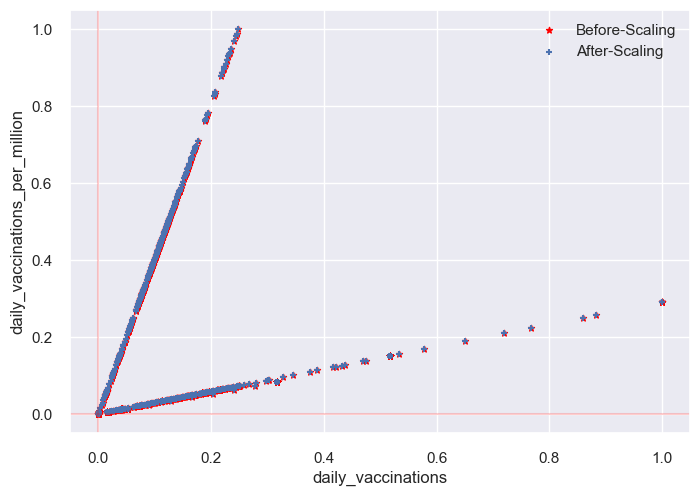
plt.axhline(0, color='red',alpha=0.2)

plt.axvline(0, color='red',alpha=0.2);

plt.show()

**Output:**

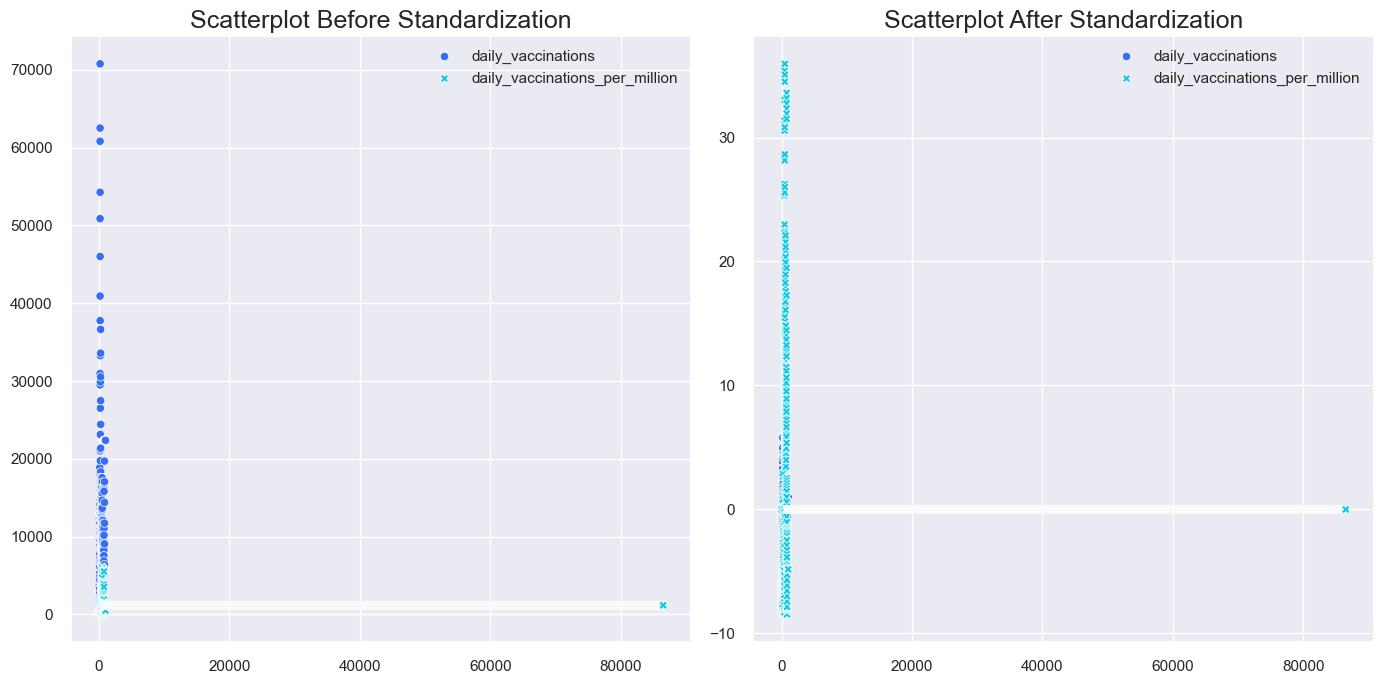




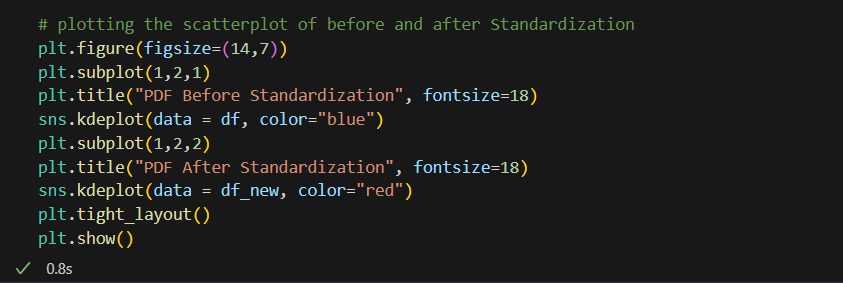
**Scatter plot standardization:**

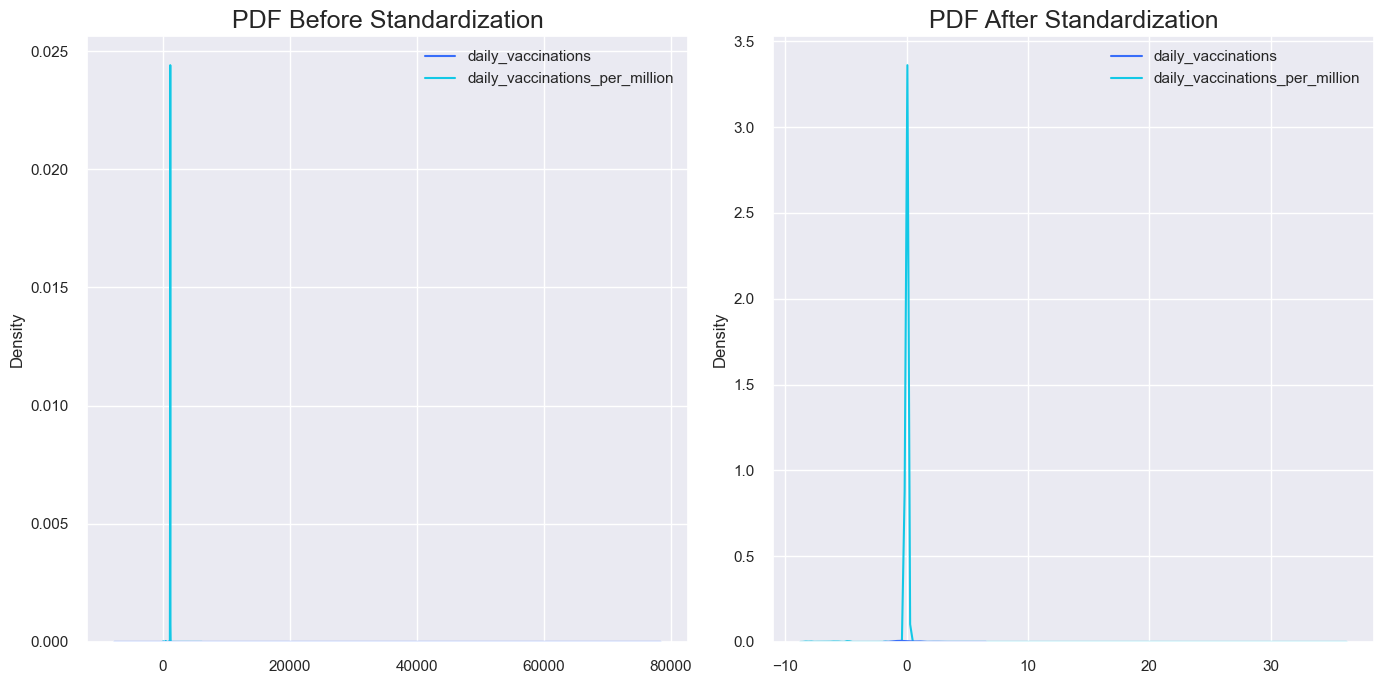


**Output:**



**PDF standardization:**



**Output:**

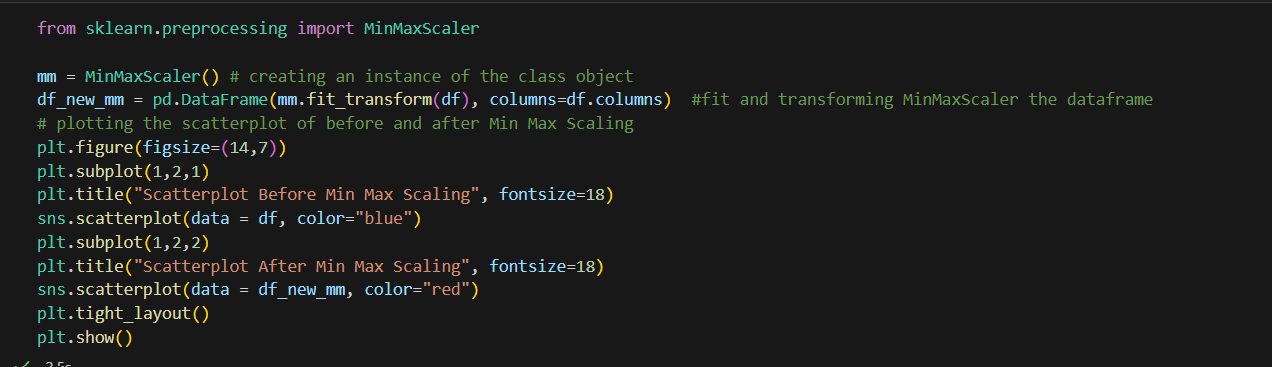
**Normalization**

* Normalization is a technique often applied as part of data preparation for machine learning.
* The goal of normalization is to change the values of numeric columns in the dataset to use a common scale, without distorting differences in the ranges of values or losing information.

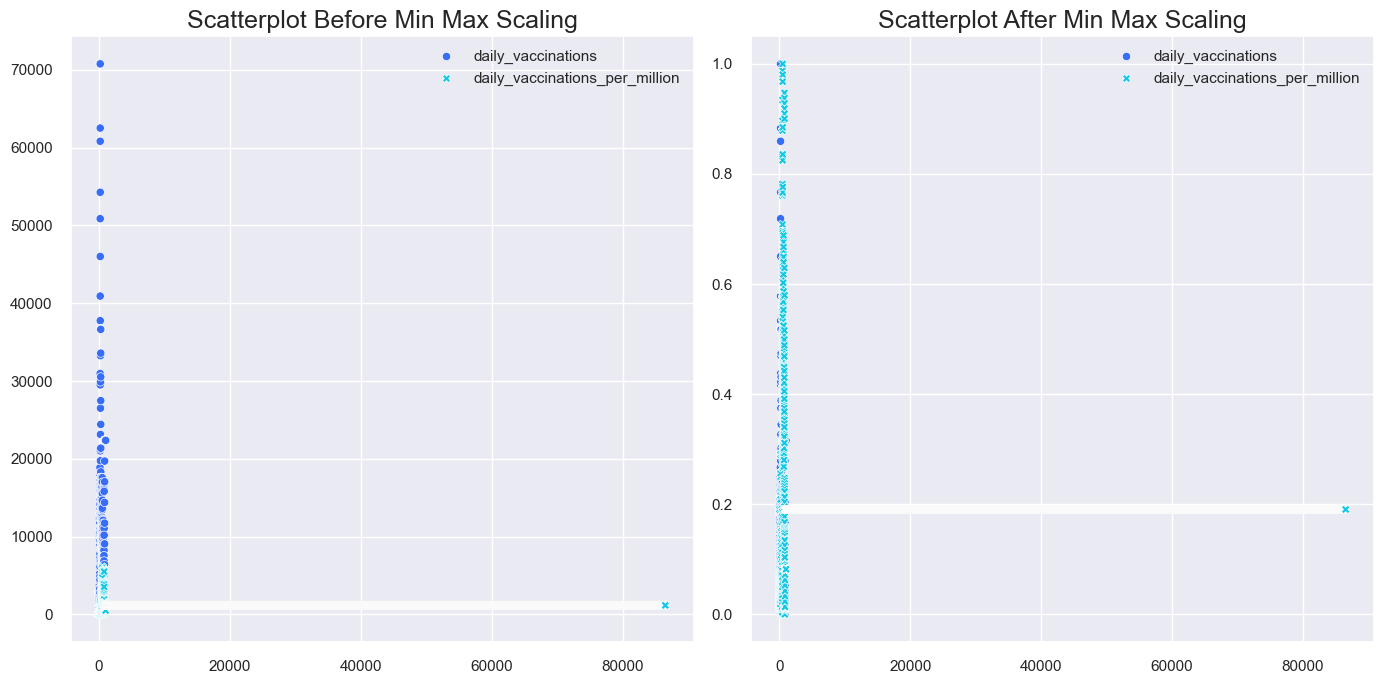
**Min Max Scaling**

* Min-max normalization is one of the most common ways to normalize data.
* For every feature, the minimum value of that feature gets transformed into a 0, the maximum value gets transformed into a 1, and every other value gets transformed into a decimal between 0 and 1.

**Scatter plot Min Max Scaling**

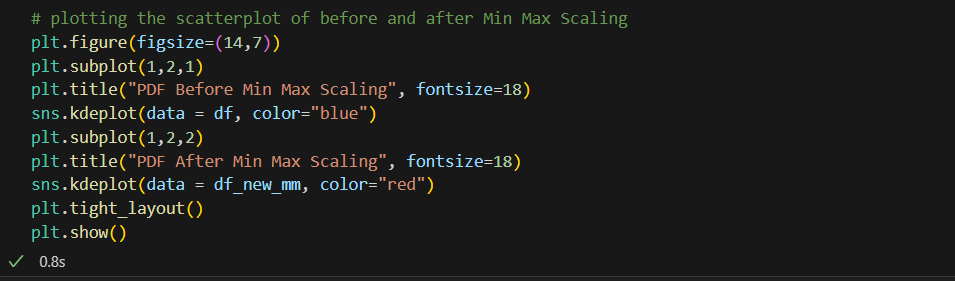
**Code:**

f

**Output:**

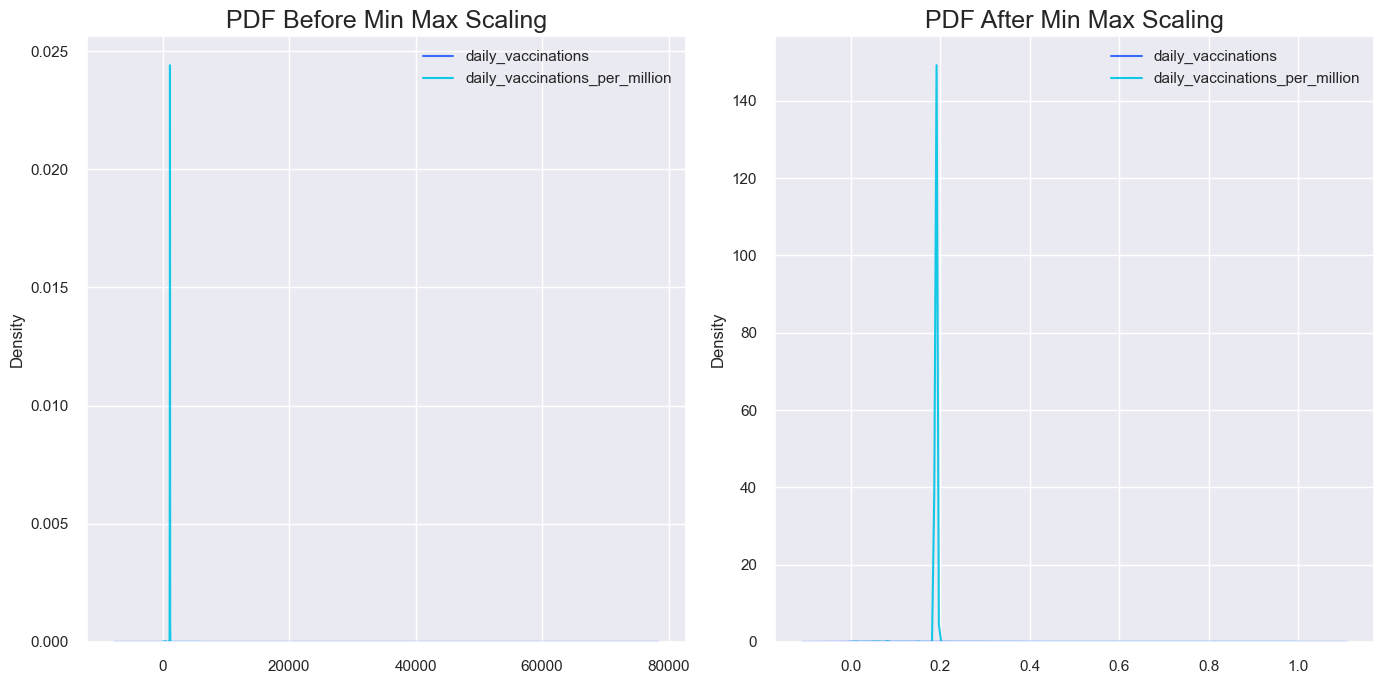
**PDF Min Max Scaling:**

**Code:**



D

**Output:**

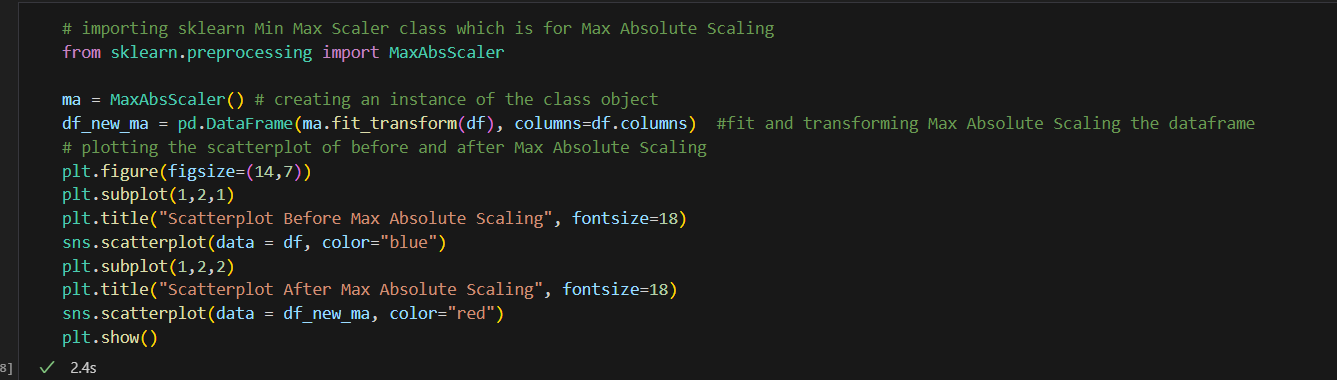


**Max Absolute Scaling**

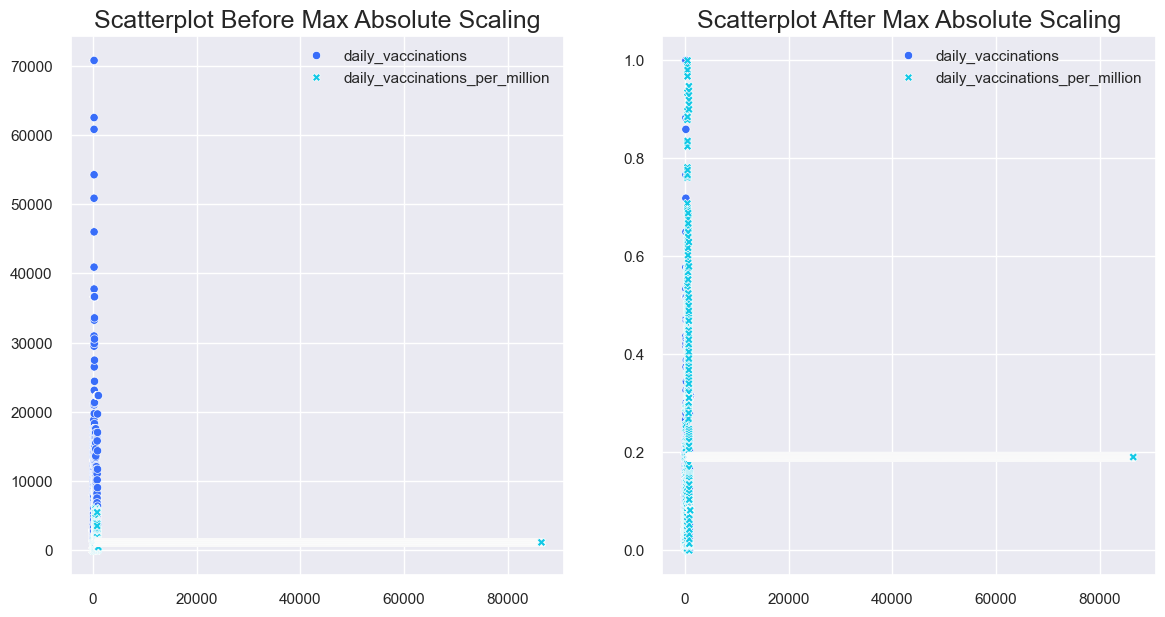
* Scale each feature by its maximum absolute value.
* This estimator scales and translates each feature individually such that the maximal absolute value of each feature in the training set will be 1.0.
* It does not shift/center the data, and thus does not destroy any sparsity.
* This scaler can also be applied to sparse CSR or CSC matrices.

**Scatter Plot Max Absolute Scaling:**

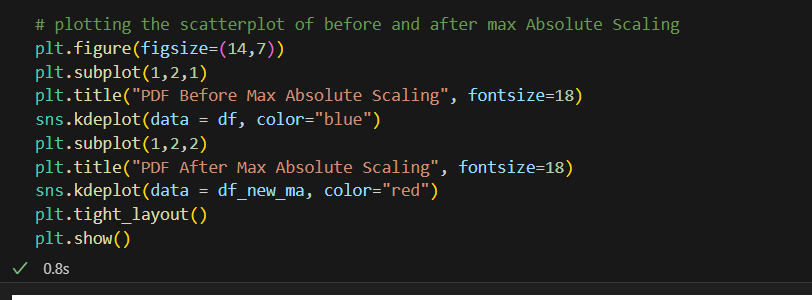
**Code:**

a

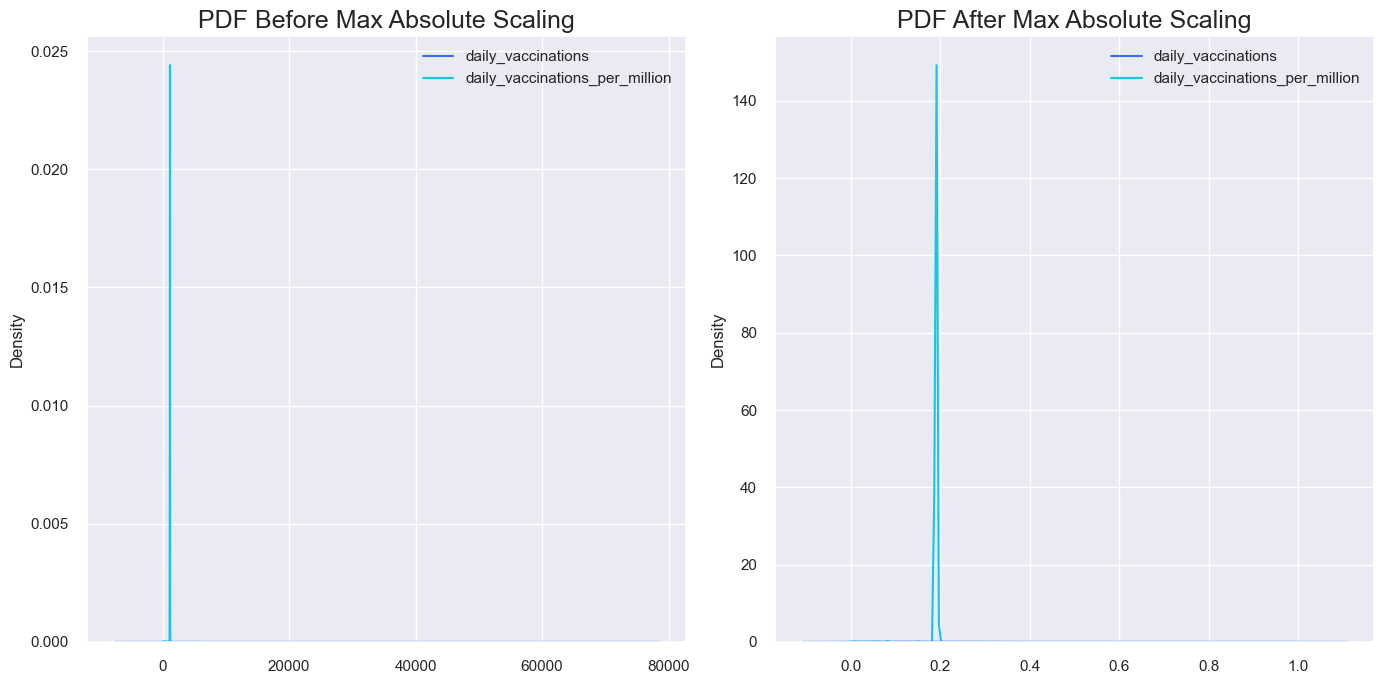
**Output:**



**PDF Max Absolute Scaling:**



**Output:**

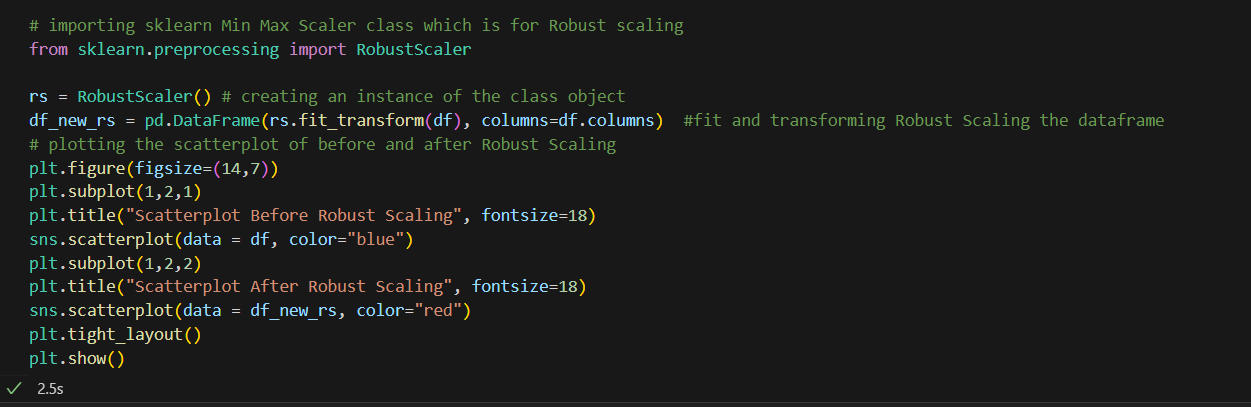


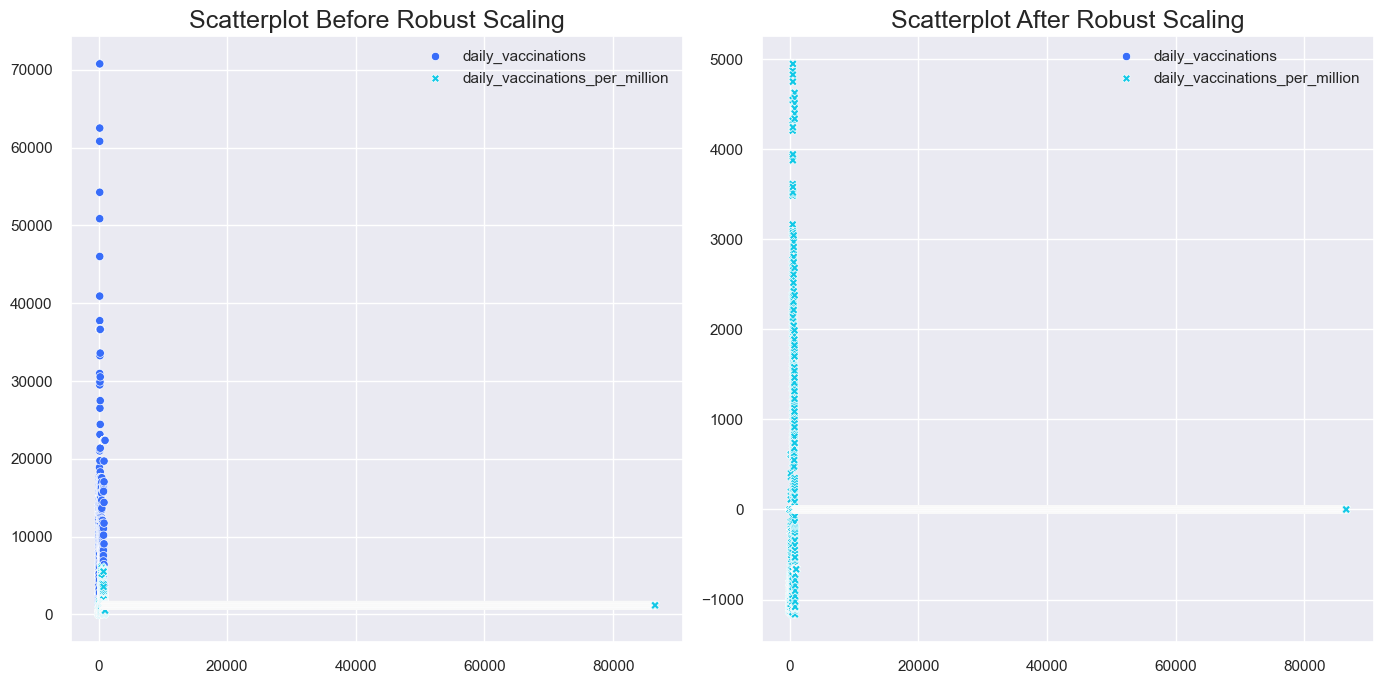
**Robust Scaling**

* This Scaler removes the median and scales the data according to the quantile range (defaults to IQR: Interquartile Range).
* The IQR is the range between the 1st quartile (25th quantile) and the 3rd quartile (75th quantile).

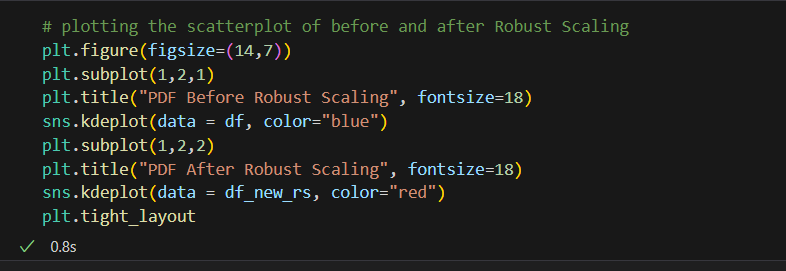
**Scatter Plot Robust Scaling**

**Code**:

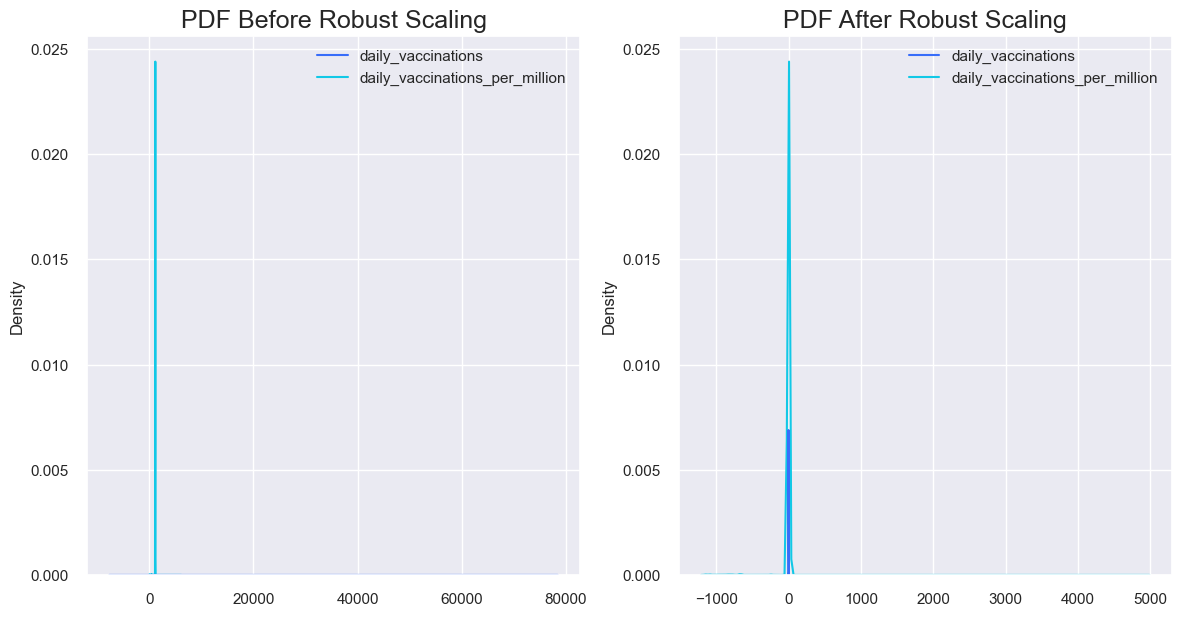


**Output:**

**PDF Robust Scaling**



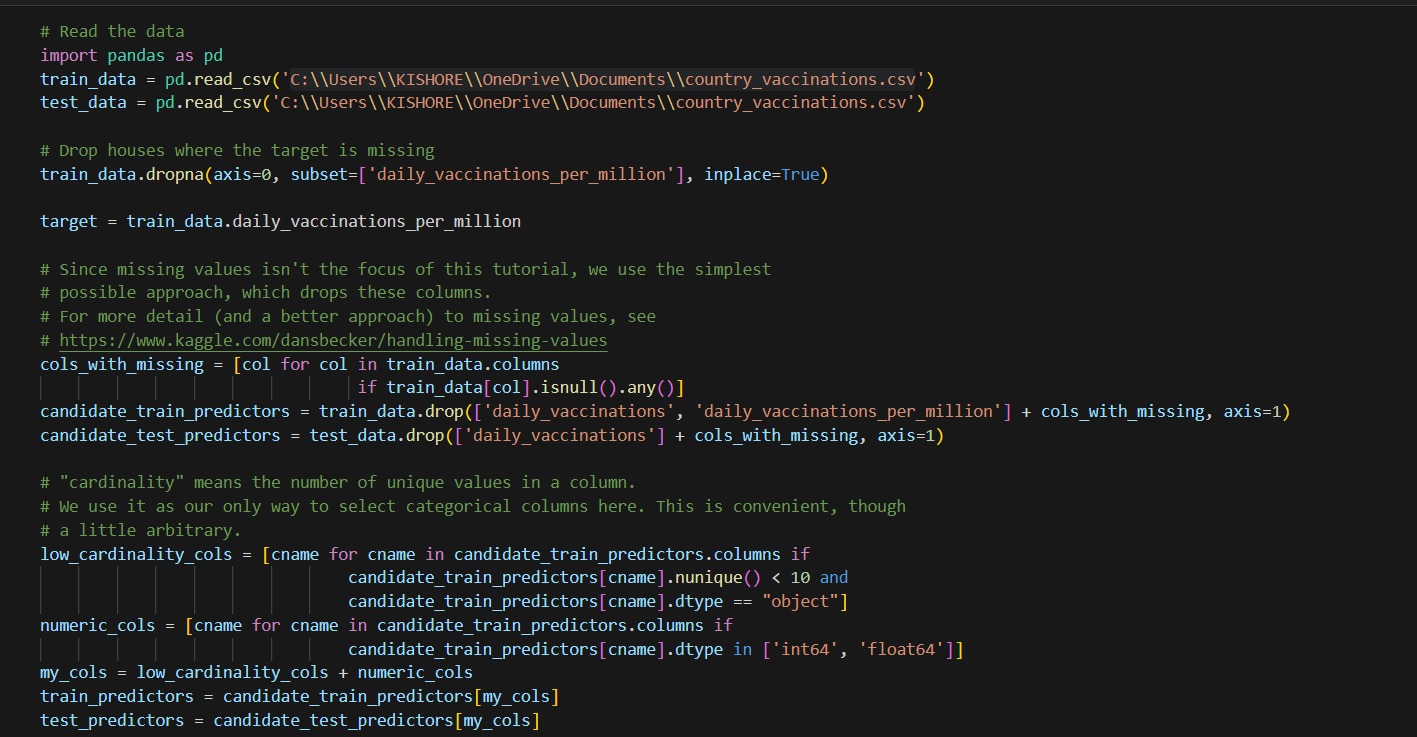
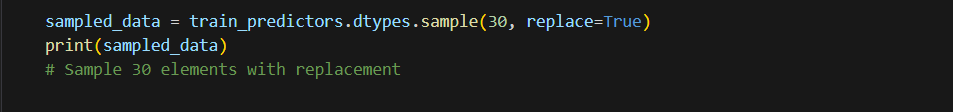
**Output:**



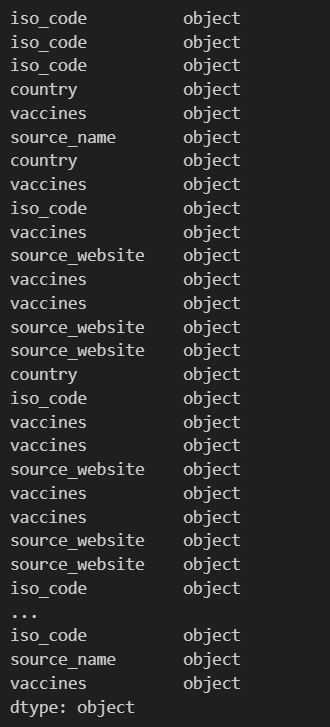
**Splitting the Data set:**

* A train test split is when you split your data into a training set and a testing set.
* The training set is used for training the model, and the testing set is used to test your model.
* This allows you to train your models on the training set, and then test their accuracy on the unseen testing set.
* There are a few different ways to do a train test split, but the most common is to simply split your data into two sets.
* For example 80% for training and 20% for testing. This ensures that both sets are representative of the entire dataset, and gives you a good way to measure the accuracy of your models.

**Code:**



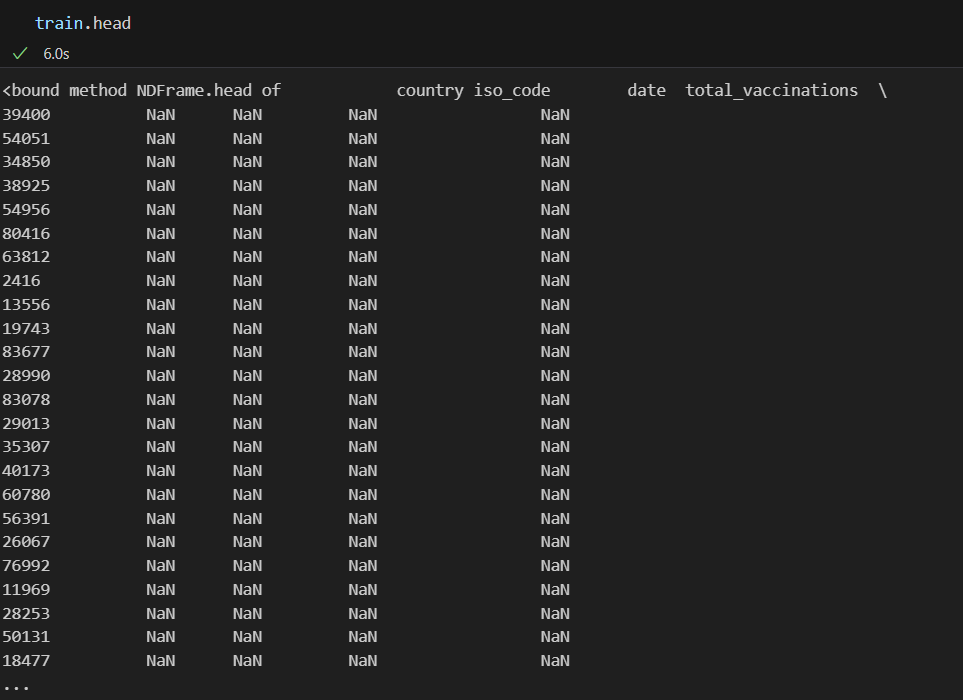
**Output:**

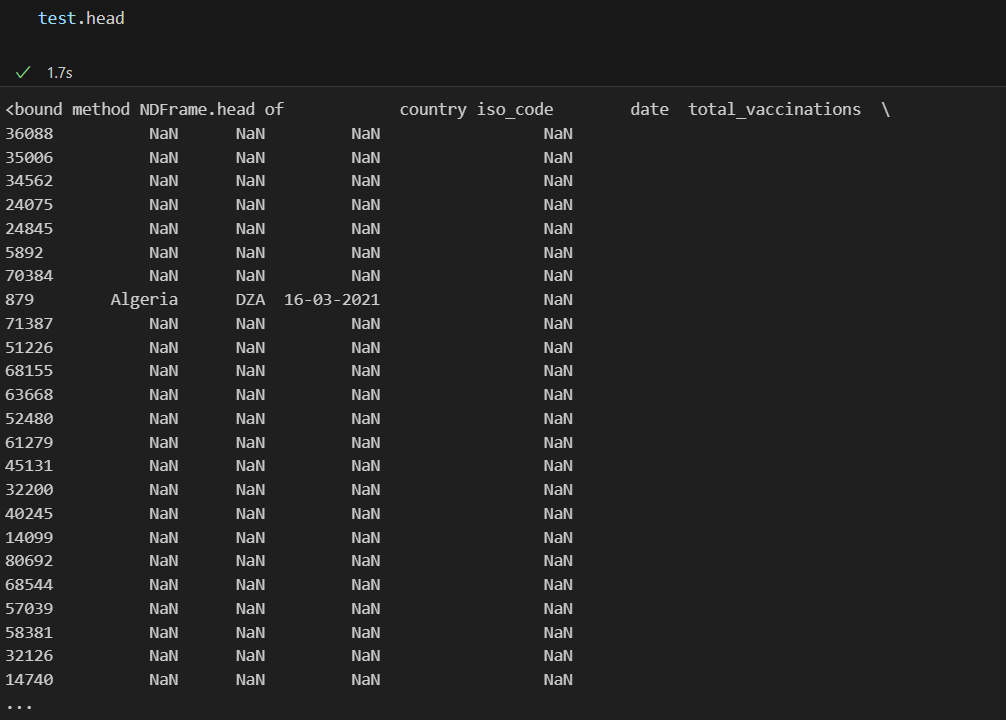


**Reading the Trained dataset:**

* The head() function in R is used to display the first n rows present in the input data frame.

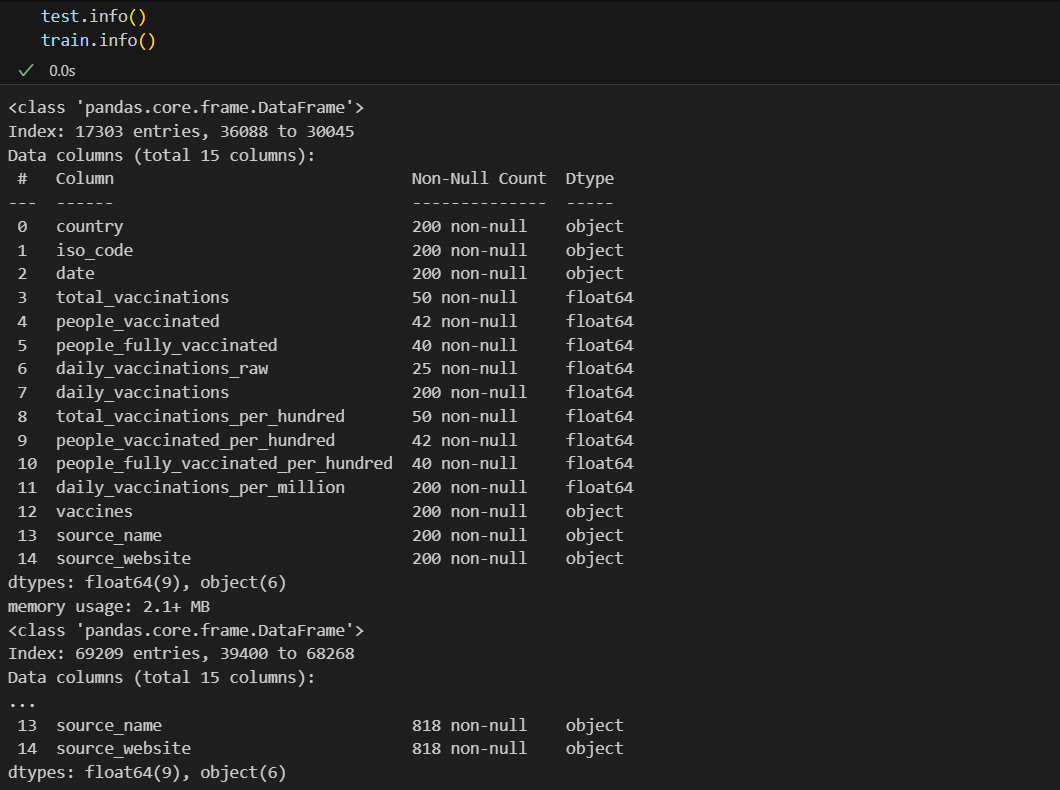
In this section, we are going to get the first n rows using head() function

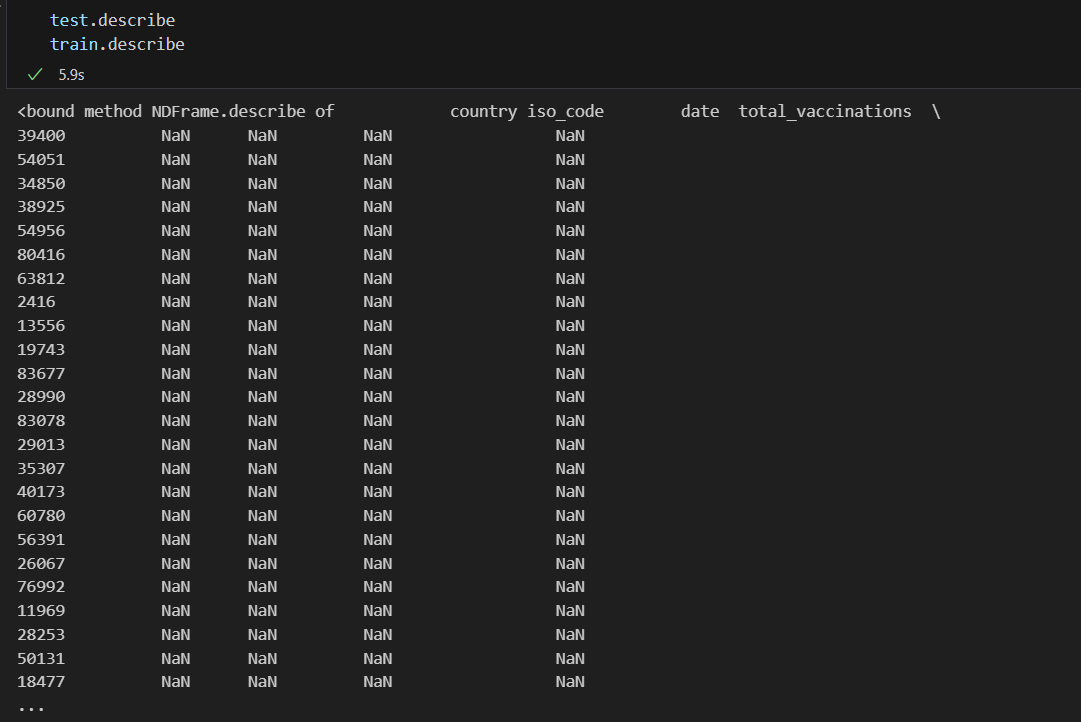




**Info()**

* To get the Information of the trained dataset

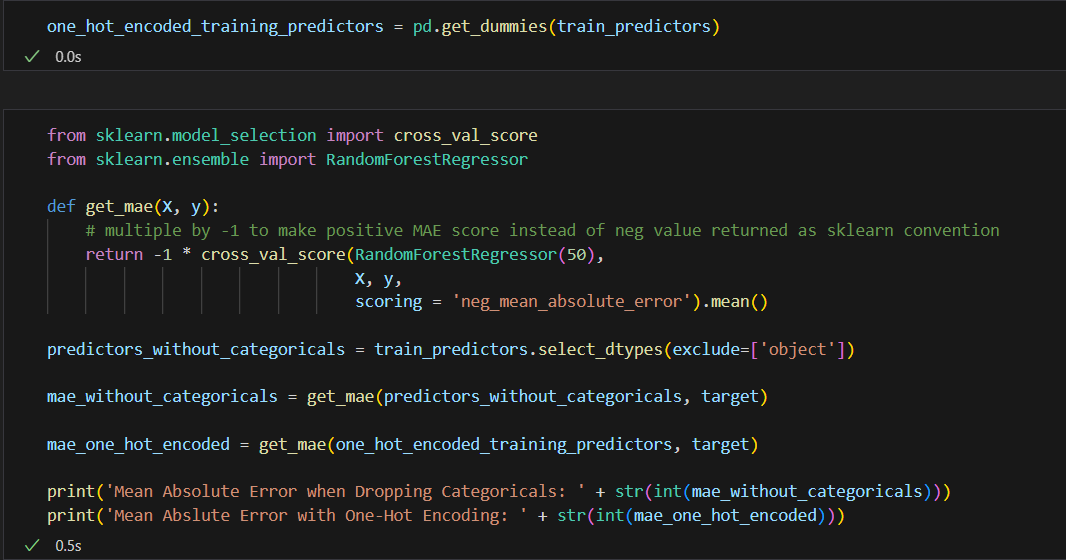


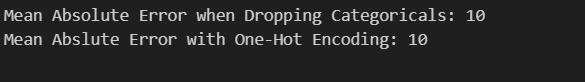


**One-hot encoding**

* One-hot encoding can be used to transform one or more categorical features into numerical dummy features useful for training machine learning model.
* One-hot encoding is also called dummy encoding due to the fact that the transformation

of categorical features results into dummy features. OneHotEncoder class of sklearn



o

**Output**:

