Project: Covid-19 Vaccine Analysis

Phase3: Development (part 1)

Question:

Begin conducting the Covid-19 vaccines analysis by collecting and preprocessing the dataset. Collect and preprocess the Covid-19 vaccine data for analysis.

Data set and its details:

The dataset "COVID-19 World Vaccination Progress" on Kaggle is a collection of data related to the COVID-19 vaccination efforts worldwide. It provides information about the progress of COVID-19 vaccinations in various countries and regions. This dataset is designed to help researchers, data scientists, and analysts understand and analyze the progress of COVID-19 vaccination campaigns across different countries. A second file, with manufacturers information, is included. Below is a detailed overview of the dataset:

Title: COVID-19 World Vaccination Progress

Dataset ID: gpreda/covid-world-vaccination-progress

Source: The dataset was created by a Kaggle user named Gabriel Preda, collected from various sources, including government health agencies, international organizations, and research institutions.

Description:

1. The dataset provides information about the COVID-19 vaccination progress from various countries around the world.
2. It includes data on vaccine distribution, vaccination coverage, and other related statistics.
3. The dataset may include information about the types of vaccines used, vaccination rates over time, and population demographics.

Columns/Attributes:

1. The dataset typically contains columns such as country, iso_code, date, total_vaccinations, people_vaccinated, people_fully_vaccinated, daily_vaccinations_raw, daily_vaccinations, and more.

2.These columns provide information about the total number of vaccinations, daily vaccination rates, and other vaccination-related metrics for each country.

Usage:

1. Analyzing vaccination progress over time for different countries.
2. Identifying countries with high vaccination rates or disparities.
3. Forecasting future vaccination trends.
4. Studying the impact of different vaccines on vaccination rates.
5. Correlating vaccination progress with COVID-19 infection and mortality rates.

Data Format:

The data is usually structured as a CSV (Comma-Separated Values) file, with rows representing different countries or regions and columns representing various attributes related to vaccination progress and population.

Updates:

The dataset may be updated regularly to reflect the latest vaccination data, making it useful for tracking changes and trends over time.

Columns:

- Country- this is the country for which the vaccination information is provided.
- Country ISO Code - ISO code for the country.
- Date - date for the data entry; for some of the dates we have only the daily vaccinations, for others, only the (cumulative) total.
- Total number of vaccinations - this is the absolute number of total immunizations in the country. Total number of people vaccinated - a person, depending on the immunization scheme, will receive one or more (typically 2) vaccines; at a certain moment, the number of vaccinations might be larger than the number of people.
- Total number of people fully vaccinated - this is the number of people that received the entire set of immunization according to the immunization scheme (typically 2); at a certain moment in time, there might be a certain number of people that received one vaccine and another number (smaller) of people that received all vaccines in the scheme.
- Daily vaccinations (raw) - for a certain data entry, the number of vaccinations for that date/country.
- Daily vaccinations - for a certain data entry, the number of vaccinations for that date/country.
- Total vaccinations per hundred - ratio (in percent) between vaccination number and total population up to the date in the country.
- Total number of people vaccinated per hour- ratio (in percent) between population immunized and total population up to the date in the country.
- Total number of people fully vaccinated per hundred - ratio (in percent) between population fully immunized and total population up to the date in the country.
- Number of vaccinations per day - number of daily vaccinations for that day and country.
- Daily vaccinations per million - ratio (in ppm) between vaccination number and total population for the current date in the country.
- Vaccines used in the country - total number of vaccines used in the country (up to date).
- Source name - source of the information (national authority, international organization, local organization etc.).

- Source website - website of the source of information.

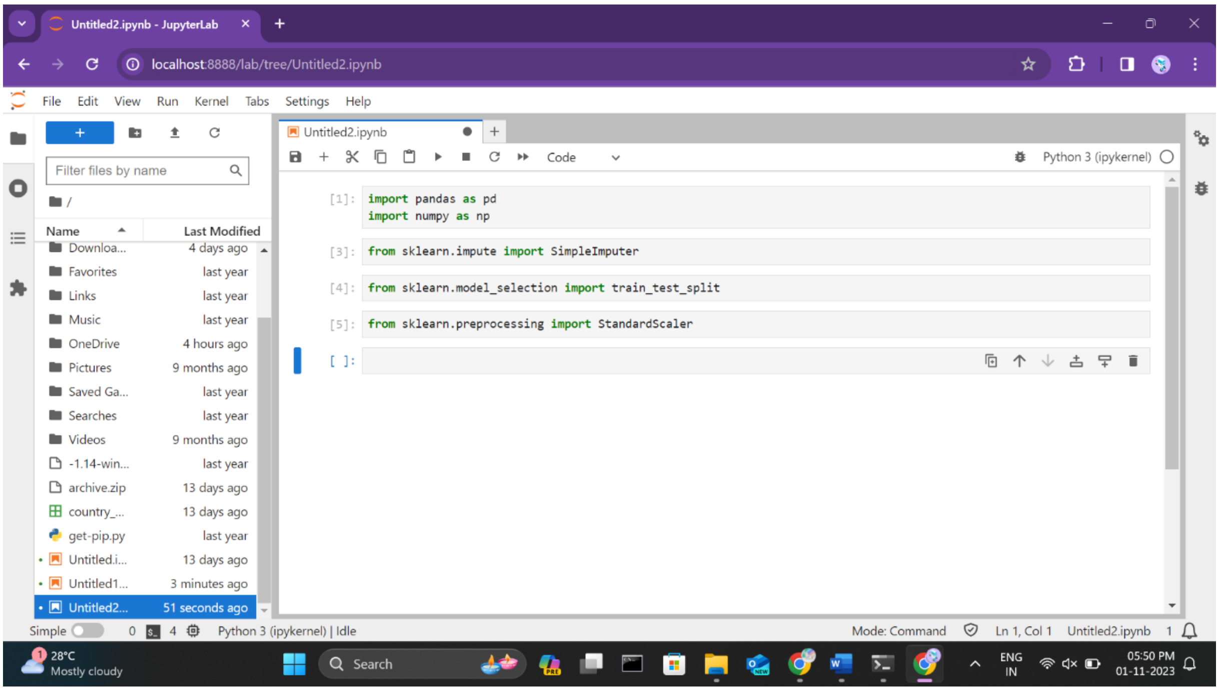
There is a second file added (country vaccinations by manufacturer), with the following columns:

- Location - country.
- Date - date.
- Vaccine - vaccine type.
- Total number of vaccinations - total number of vaccinations / current time and vaccine type.

Importing the required libraries:

To perform the data preprocessing, splitting, scaling, and other tasks as described, several libraries in Python are needed to be imported. Here are the required libraries for the code:

1. For loading and preprocessing the dataset:
   import pandas as pd import numpy as np
2. For handling missing data:
   from sklearn.impute import SimpleImputer
3. For splitting the dataset into training and test sets:
   from sklearn.model_selection import train_test_split
4. For feature scaling: from sklearn.preprocessing import StandardScaler



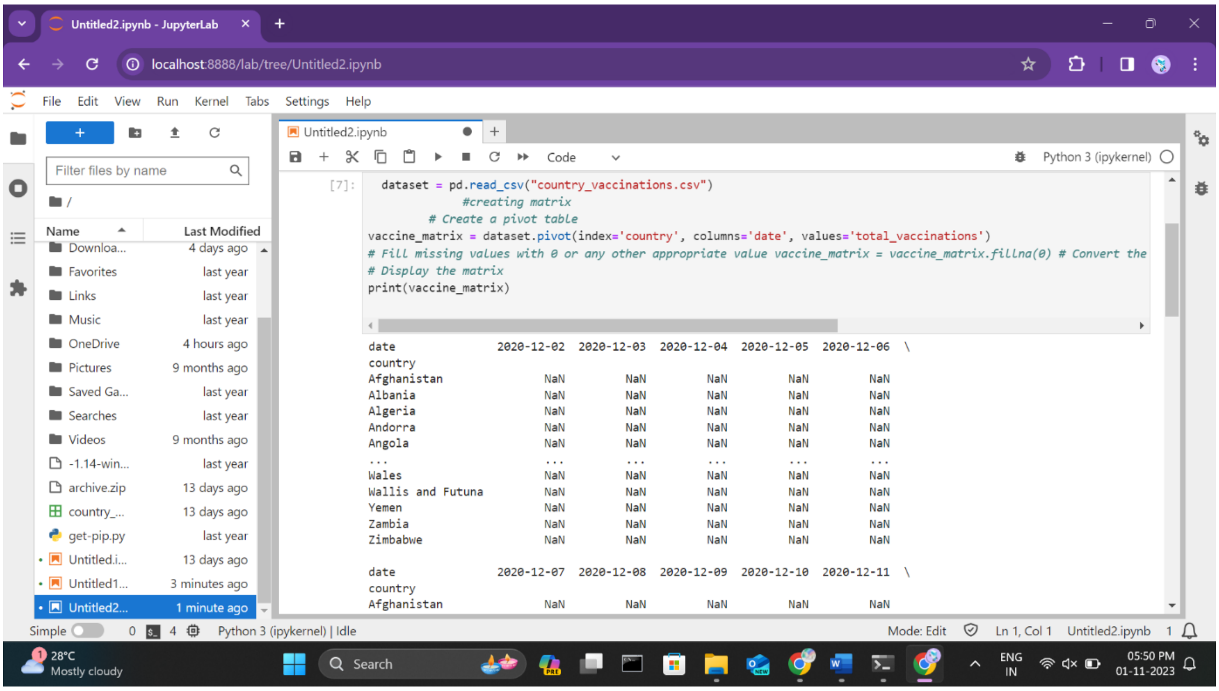
Untitled2.ipynb - JupyterLab
localhost:8888/lab/tree/Untitled2.ipynb
File Edit View Run Kernel Tabs Settings Help
Filter files by name
Code
Python 3 (ipykernel)
Simple 0 4 Python 3 (ipykernel) | Idle
Mode: Command Ln 1, Col 1 Untitled2.ipynb 1
28°C Mostly cloudy
Search
ENG IN 05:50 PM 01-11-2023

Name	Last Modified
Downloa...	4 days ago
Favorites	last year
Links	last year
Music	last year
OneDrive	4 hours ago
Pictures	9 months ago
Saved Ga...	last year
Searches	last year
Videos	9 months ago
-1.14-win...	last year
archive.zip	13 days ago
country_...	13 days ago
get-pip.py	last year
Untitled.i...	13 days ago
Untitled1...	3 minutes ago
Untitled2...	51 seconds ago

```
[1]: import pandas as pd
     import numpy as np

[3]: from sklearn.impute import SimpleImputer

[4]: from sklearn.model_selection import train_test_split

[5]: from sklearn.preprocessing import StandardScaler

[ ]:
```

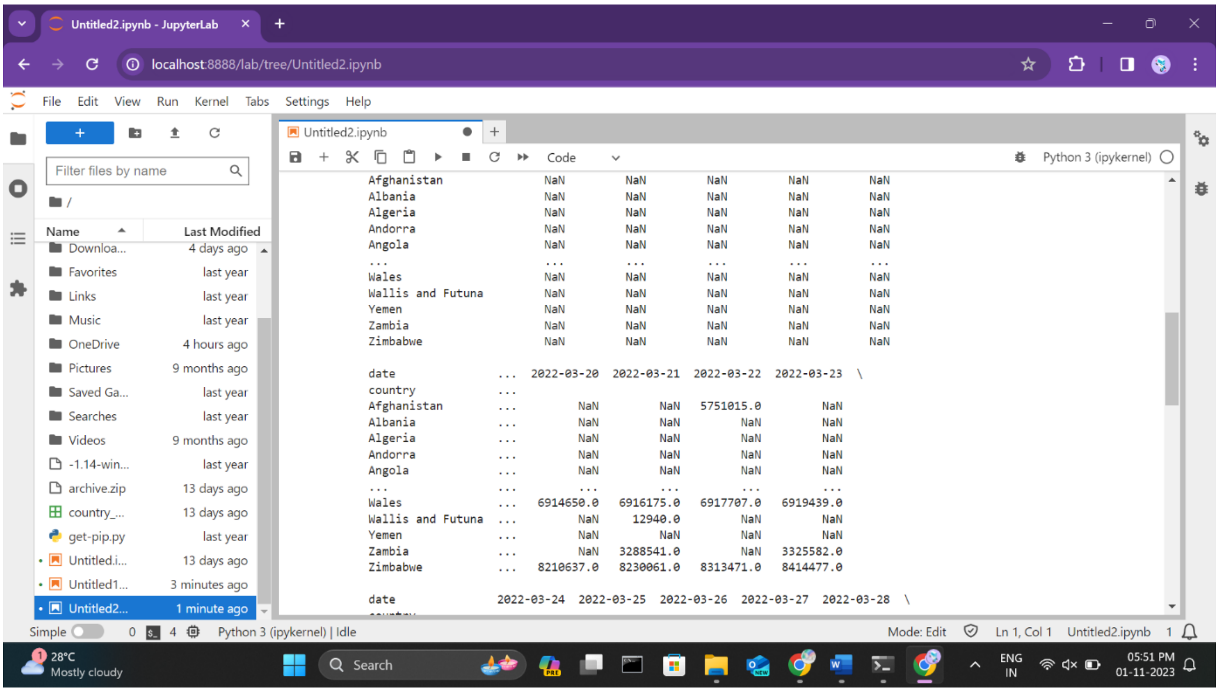
Importing the dataset:

Use Pandas to read the dataset file you downloaded into a DataFrame:

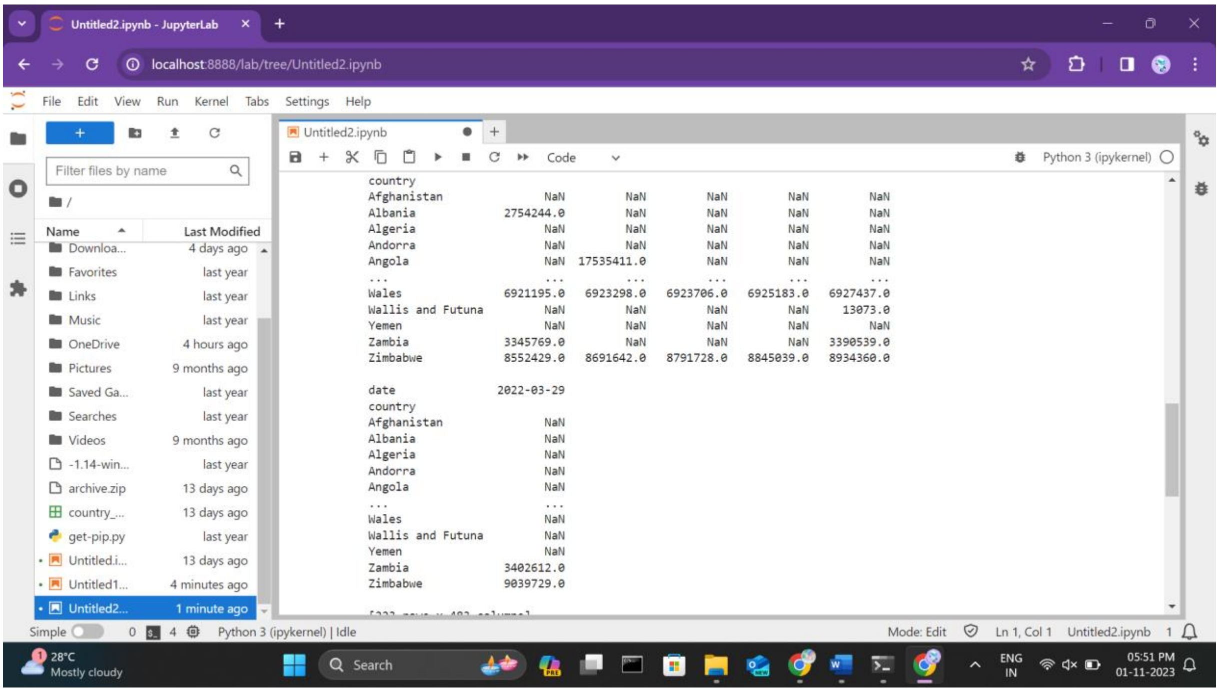


```
[7]:  dataset = pd.read_csv("country_vaccinations.csv")
            #creating matrix
        # Create a pivot table
vaccine_matrix = dataset.pivot(index='country', columns='date', values='total_vaccinations')
# Fill missing values with 0 or any other appropriate value vaccine_matrix = vaccine_matrix.fillna(0) # Convert the
# Display the matrix
print(vaccine_matrix)
```

```
date               2020-12-02  2020-12-03  2020-12-04  2020-12-05  2020-12-06  \
country
Afghanistan               NaN         NaN         NaN         NaN         NaN
Albania                   NaN         NaN         NaN         NaN         NaN
Algeria                   NaN         NaN         NaN         NaN         NaN
Andorra                   NaN         NaN         NaN         NaN         NaN
Angola                    NaN         NaN         NaN         NaN         NaN
...                       ...         ...         ...         ...         ...
Wales                     NaN         NaN         NaN         NaN         NaN
Wallis and Futuna         NaN         NaN         NaN         NaN         NaN
Yemen                     NaN         NaN         NaN         NaN         NaN
Zambia                    NaN         NaN         NaN         NaN         NaN
Zimbabwe                  NaN         NaN         NaN         NaN         NaN

date               2020-12-07  2020-12-08  2020-12-09  2020-12-10  2020-12-11  \
country
Afghanistan               NaN         NaN         NaN         NaN         NaN
```



```
Afghanistan               NaN         NaN         NaN         NaN         NaN
Albania                   NaN         NaN         NaN         NaN         NaN
Algeria                   NaN         NaN         NaN         NaN         NaN
Andorra                   NaN         NaN         NaN         NaN         NaN
Angola                    NaN         NaN         NaN         NaN         NaN
...                       ...         ...         ...         ...         ...
Wales                     NaN         NaN         NaN         NaN         NaN
Wallis and Futuna         NaN         NaN         NaN         NaN         NaN
Yemen                     NaN         NaN         NaN         NaN         NaN
Zambia                    NaN         NaN         NaN         NaN         NaN
Zimbabwe                  NaN         NaN         NaN         NaN         NaN

date               ...  2022-03-20  2022-03-21  2022-03-22  2022-03-23  \
country            ...
Afghanistan        ...         NaN         NaN   5751015.0         NaN
Albania            ...         NaN         NaN         NaN         NaN
Algeria            ...         NaN         NaN         NaN         NaN
Andorra            ...         NaN         NaN         NaN         NaN
Angola             ...         NaN         NaN         NaN         NaN
...                ...         ...         ...         ...         ...
Wales              ...   6914650.0   6916175.0   6917707.0   6919439.0
Wallis and Futuna  ...         NaN     12940.0         NaN         NaN
Yemen              ...         NaN         NaN         NaN         NaN
Zambia             ...         NaN   3288541.0         NaN   3325582.0
Zimbabwe           ...   8210637.0   8230061.0   8313471.0   8414477.0

date               2022-03-24  2022-03-25  2022-03-26  2022-03-27  2022-03-28  \
```

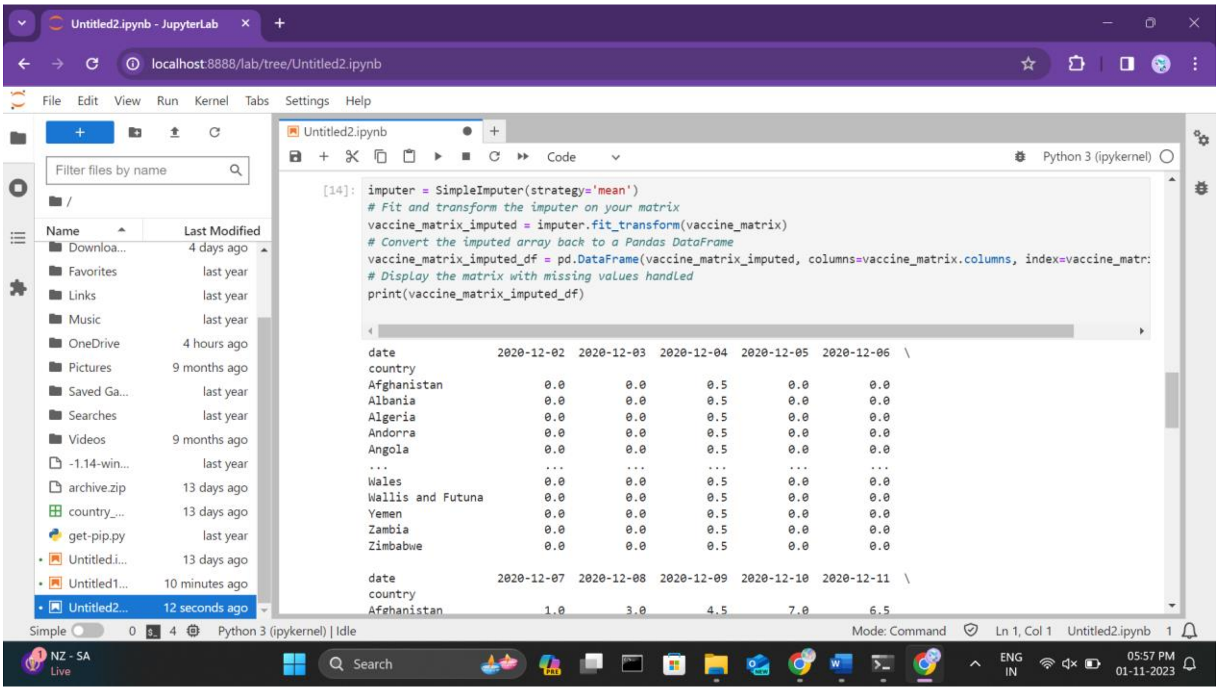


```
             country
Afghanistan              NaN        NaN        NaN        NaN        NaN
Albania            2754244.0        NaN        NaN        NaN        NaN
Algeria                  NaN        NaN        NaN        NaN        NaN
Andorra                  NaN        NaN        NaN        NaN        NaN
Angola                   NaN 17535411.0        NaN        NaN        NaN
...                      ...        ...        ...        ...        ...
Wales              6921195.0  6923298.0  6923706.0  6925183.0  6927437.0
Wallis and Futuna        NaN        NaN        NaN        NaN    13073.0
Yemen                    NaN        NaN        NaN        NaN        NaN
Zambia             3345769.0        NaN        NaN        NaN  3390539.0
Zimbabwe           8552429.0  8691642.0  8791728.0  8845039.0  8934360.0

date              2022-03-29
country
Afghanistan              NaN
Albania                  NaN
Algeria                  NaN
Andorra                  NaN
Angola                   NaN
...                      ...
Wales                    NaN
Wallis and Futuna        NaN
Yemen                    NaN
Zambia             3402612.0
Zimbabwe           9039729.0
```

In this code, we first pivot the DataFrame to transform it into a matrix where rows represent countries, columns represent dates, and the values are the total vaccination counts. We fill a ny missing values with 0.

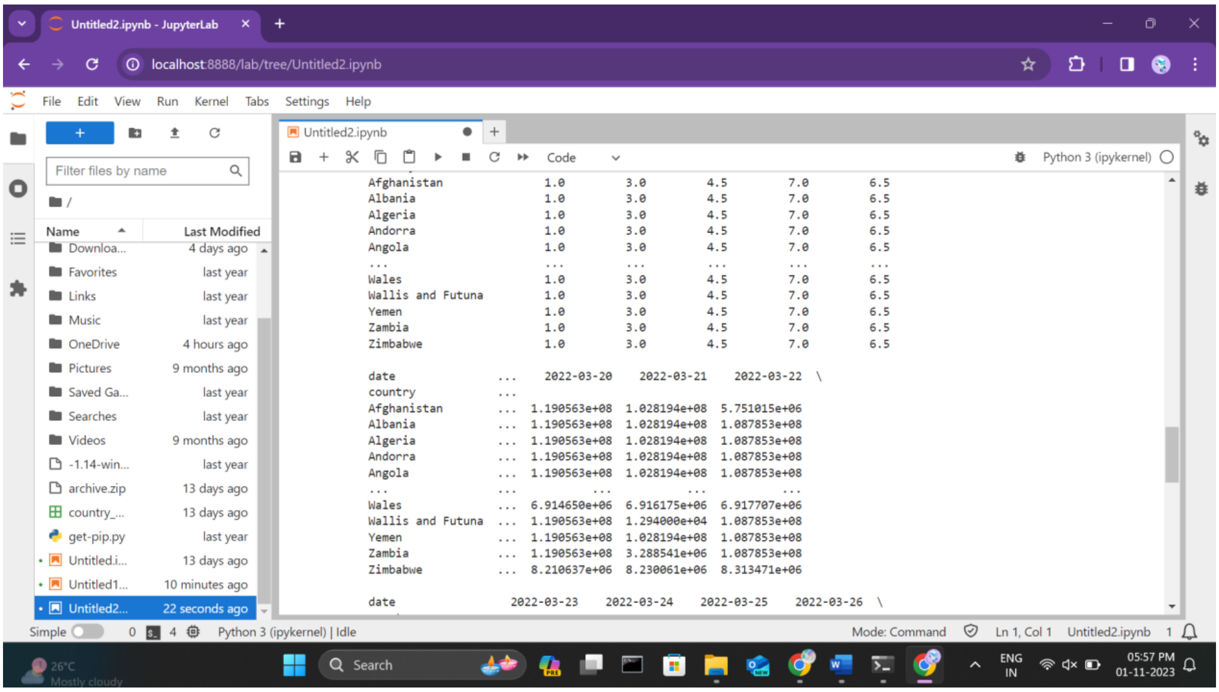
Handling the missing data:

Scikit-learn library provides the SimpleImputer class, which is a handy tool for handling missing data.

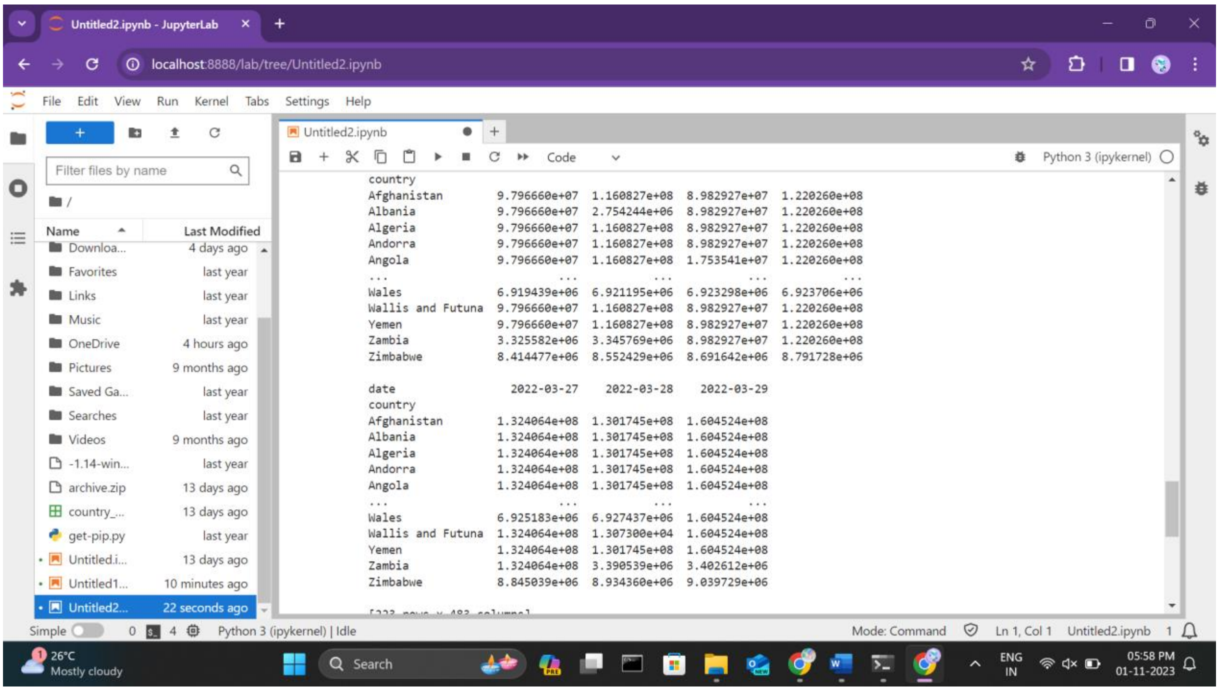


```
imputer = SimpleImputer(strategy='mean')
# Fit and transform the imputer on your matrix
vaccine_matrix_imputed = imputer.fit_transform(vaccine_matrix)
# Convert the imputed array back to a Pandas DataFrame
vaccine_matrix_imputed_df = pd.DataFrame(vaccine_matrix_imputed, columns=vaccine_matrix.columns, index=vaccine_matr:
# Display the matrix with missing values handled
print(vaccine_matrix_imputed_df)
```

```
date               2020-12-02  2020-12-03  2020-12-04  2020-12-05  2020-12-06  \
country
Afghanistan               0.0         0.0         0.5         0.0         0.0
Albania                   0.0         0.0         0.5         0.0         0.0
Algeria                   0.0         0.0         0.5         0.0         0.0
Andorra                   0.0         0.0         0.5         0.0         0.0
Angola                    0.0         0.0         0.5         0.0         0.0
...                       ...         ...         ...         ...         ...
Wales                     0.0         0.0         0.5         0.0         0.0
Wallis and Futuna         0.0         0.0         0.5         0.0         0.0
Yemen                     0.0         0.0         0.5         0.0         0.0
Zambia                    0.0         0.0         0.5         0.0         0.0
Zimbabwe                  0.0         0.0         0.5         0.0         0.0

date               2020-12-07  2020-12-08  2020-12-09  2020-12-10  2020-12-11  \
country
Afghanistan               1.0         3.0         4.5         7.0         6.5
```



```
Afghanistan          1.0       3.0       4.5       7.0       6.5
Albania              1.0       3.0       4.5       7.0       6.5
Algeria              1.0       3.0       4.5       7.0       6.5
Andorra              1.0       3.0       4.5       7.0       6.5
Angola               1.0       3.0       4.5       7.0       6.5
...                  ...       ...       ...       ...       ...
Wales                1.0       3.0       4.5       7.0       6.5
Wallis and Futuna    1.0       3.0       4.5       7.0       6.5
Yemen                1.0       3.0       4.5       7.0       6.5
Zambia               1.0       3.0       4.5       7.0       6.5
Zimbabwe             1.0       3.0       4.5       7.0       6.5

date               ...    2022-03-20    2022-03-21    2022-03-22  \
country            ...
Afghanistan        ...  1.190563e+08  1.028194e+08  5.751015e+06
Albania            ...  1.190563e+08  1.028194e+08  1.087853e+08
Algeria            ...  1.190563e+08  1.028194e+08  1.087853e+08
Andorra            ...  1.190563e+08  1.028194e+08  1.087853e+08
Angola             ...  1.190563e+08  1.028194e+08  1.087853e+08
...                ...           ...           ...           ...
Wales              ...  6.914650e+06  6.916175e+06  6.917707e+06
Wallis and Futuna  ...  1.190563e+08  1.294000e+04  1.087853e+08
Yemen              ...  1.190563e+08  1.028194e+08  1.087853e+08
Zambia             ...  1.190563e+08  3.288541e+06  1.087853e+08
Zimbabwe           ...  8.210637e+06  8.230061e+06  8.313471e+06

date                 2022-03-23    2022-03-24    2022-03-25    2022-03-26  \
```

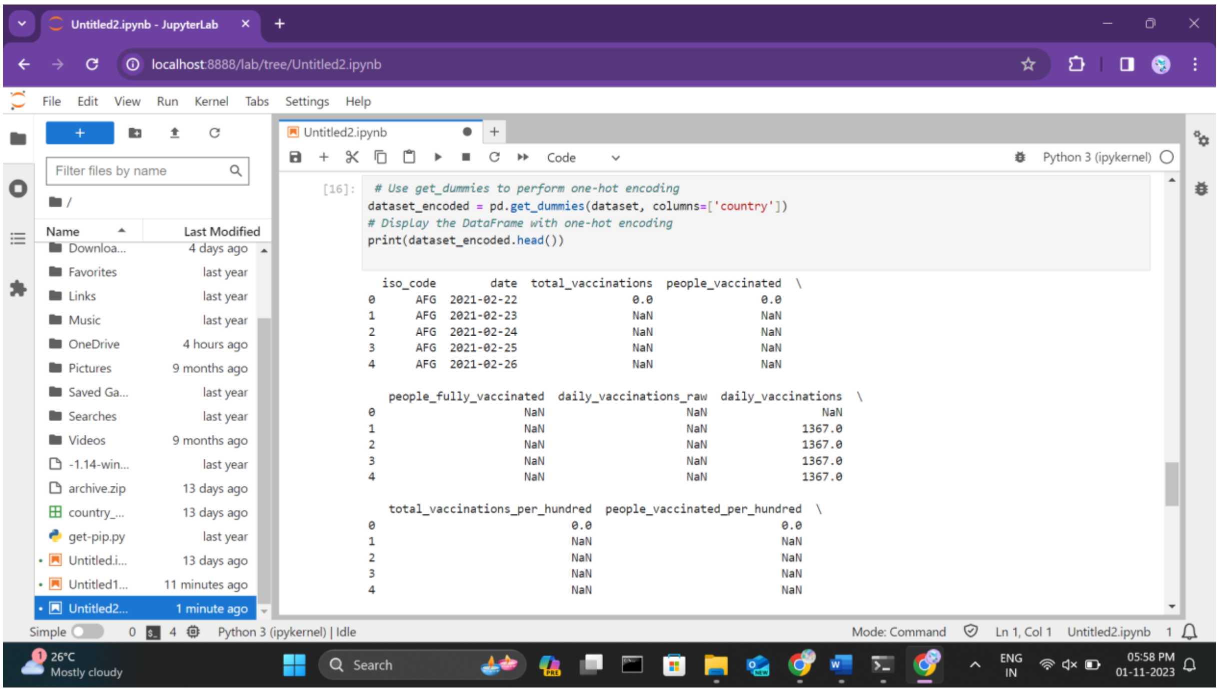


```
country
Afghanistan        9.796660e+07  1.160827e+08  8.982927e+07  1.220260e+08
Albania            9.796660e+07  2.754244e+06  8.982927e+07  1.220260e+08
Algeria            9.796660e+07  1.160827e+08  8.982927e+07  1.220260e+08
Andorra            9.796660e+07  1.160827e+08  8.982927e+07  1.220260e+08
Angola             9.796660e+07  1.160827e+08  1.753541e+07  1.220260e+08
...                         ...           ...           ...           ...
Wales              6.919439e+06  6.921195e+06  6.923298e+06  6.923706e+06
Wallis and Futuna  9.796660e+07  1.160827e+08  8.982927e+07  1.220260e+08
Yemen              9.796660e+07  1.160827e+08  8.982927e+07  1.220260e+08
Zambia             3.325582e+06  3.345769e+06  8.982927e+07  1.220260e+08
Zimbabwe           8.414477e+06  8.552429e+06  8.691642e+06  8.791728e+06

date                 2022-03-27    2022-03-28    2022-03-29
country
Afghanistan        1.324064e+08  1.301745e+08  1.604524e+08
Albania            1.324064e+08  1.301745e+08  1.604524e+08
Algeria            1.324064e+08  1.301745e+08  1.604524e+08
Andorra            1.324064e+08  1.301745e+08  1.604524e+08
Angola             1.324064e+08  1.301745e+08  1.604524e+08
...                         ...           ...           ...
Wales              6.925183e+06  6.927437e+06  1.604524e+08
Wallis and Futuna  1.324064e+08  1.307300e+04  1.604524e+08
Yemen              1.324064e+08  1.301745e+08  1.604524e+08
Zambia             1.324064e+08  3.390539e+06  3.402612e+06
Zimbabwe           8.845039e+06  8.934360e+06  9.039729e+06

[223 rows x 483 columns]
```

The SimpleImputer is used to replace missing values in the vaccine_matrix with the mean of the non-missing values.

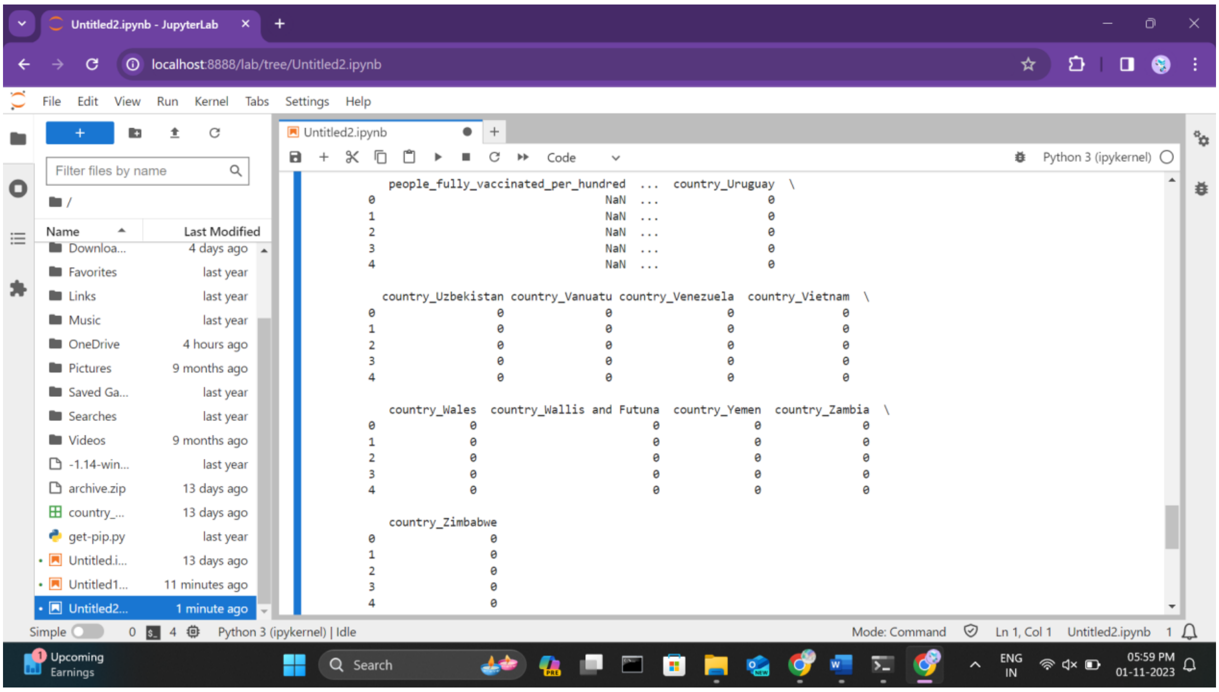
Encoding Categorical Data:

To encode categorical data using one-hot encoding in Python, you can use the pd.get_dummies function in the Pandas library. One-hot encoding converts categorical variables into binary (0/1) format, making them suitable for machine learning algorithms.



```
[16]: # Use get_dummies to perform one-hot encoding
      dataset_encoded = pd.get_dummies(dataset, columns=['country'])
      # Display the DataFrame with one-hot encoding
      print(dataset_encoded.head())
```

```
  iso_code        date  total_vaccinations  people_vaccinated  \
0      AFG  2021-02-22                 0.0                0.0
1      AFG  2021-02-23                 NaN                NaN
2      AFG  2021-02-24                 NaN                NaN
3      AFG  2021-02-25                 NaN                NaN
4      AFG  2021-02-26                 NaN                NaN

   people_fully_vaccinated  daily_vaccinations_raw  daily_vaccinations  \
0                      NaN                     NaN                 NaN
1                      NaN                     NaN              1367.0
2                      NaN                     NaN              1367.0
3                      NaN                     NaN              1367.0
4                      NaN                     NaN              1367.0

   total_vaccinations_per_hundred  people_vaccinated_per_hundred  \
0                             0.0                            0.0
1                             NaN                            NaN
2                             NaN                            NaN
3                             NaN                            NaN
4                             NaN                            NaN
```

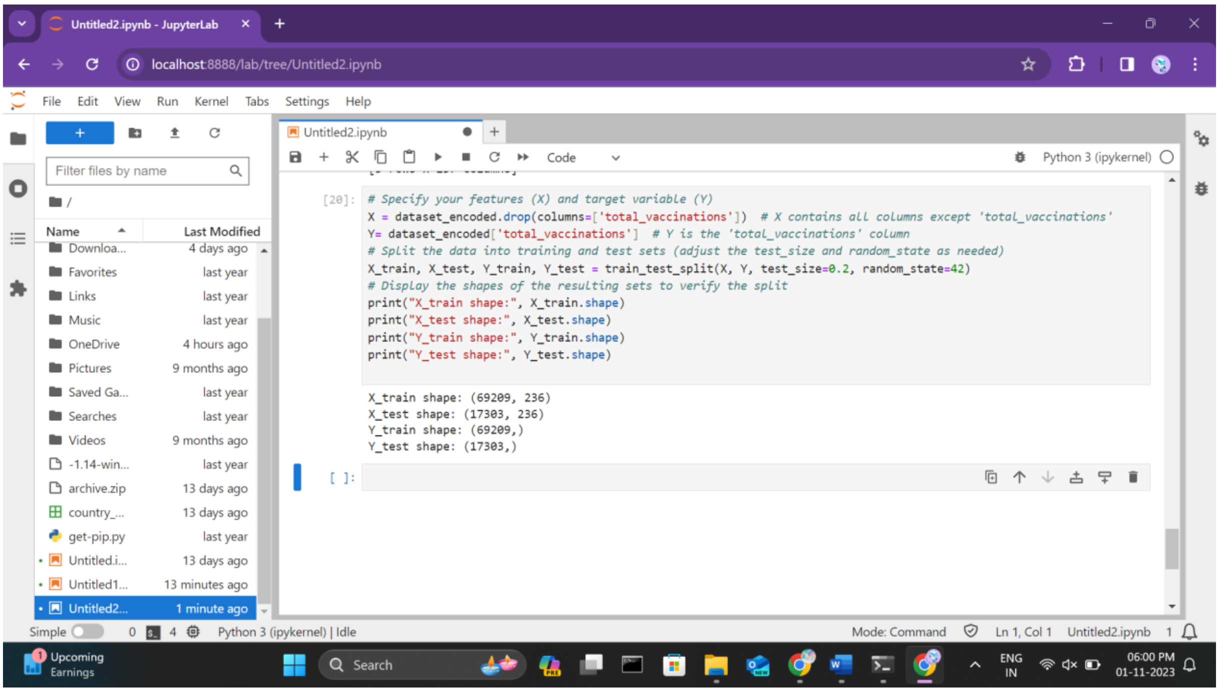


```
   people_fully_vaccinated_per_hundred  ...  country_Uruguay  \
0                                  NaN  ...                0
1                                  NaN  ...                0
2                                  NaN  ...                0
3                                  NaN  ...                0
4                                  NaN  ...                0

  country_Uzbekistan country_Vanuatu country_Venezuela  country_Vietnam  \
0                  0               0                 0                0
1                  0               0                 0                0
2                  0               0                 0                0
3                  0               0                 0                0
4                  0               0                 0                0

   country_Wales  country_Wallis and Futuna  country_Yemen  country_Zambia  \
0              0                          0              0               0
1              0                          0              0               0
2              0                          0              0               0
3              0                          0              0               0
4              0                          0              0               0

   country_Zimbabwe
0                 0
1                 0
2                 0
3                 0
4                 0
```

The get_dummies function will create binary (0/1) columns for each unique category in the ' country' column. This process effectively converts the categorical data into a numerical form at suitable for analysis or machine learning.

Splitting the dataset into test set and training set:

To split dataset into training and test sets using the train_test_split function from scikitlearn,input features (X) and target variable (Y) needed to be specified first.



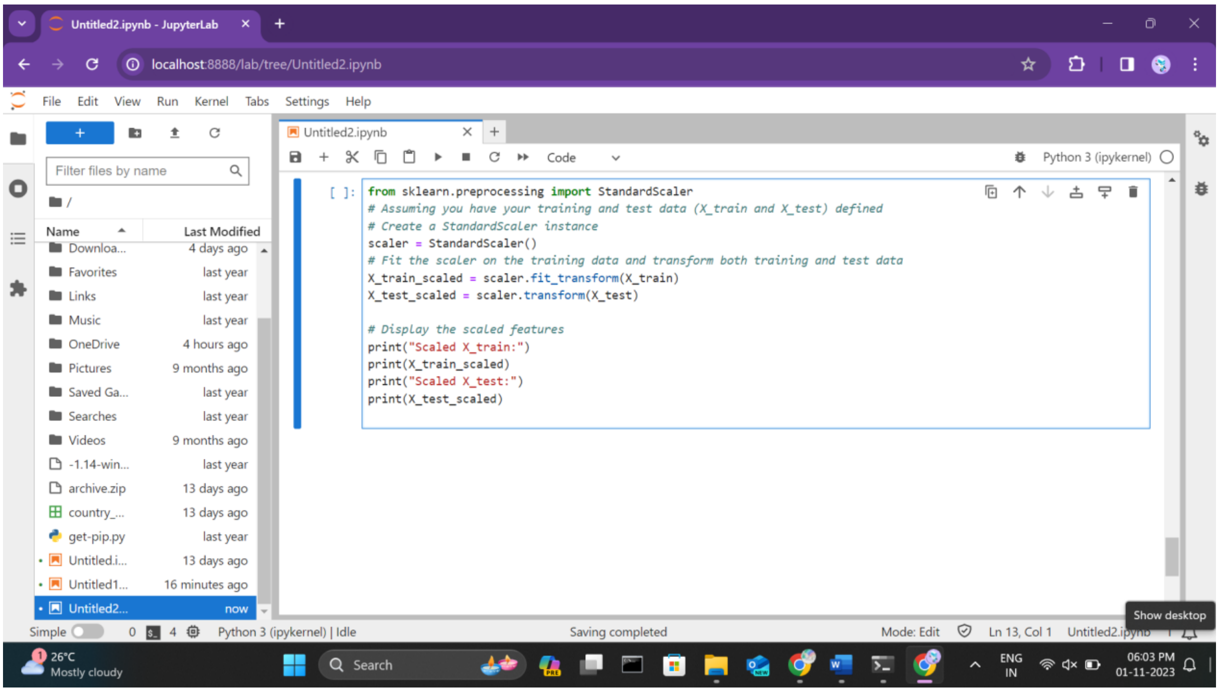
```
[20]: # Specify your features (X) and target variable (Y)
X = dataset_encoded.drop(columns=['total_vaccinations'])  # X contains all columns except 'total_vaccinations'
Y= dataset_encoded['total_vaccinations']  # Y is the 'total_vaccinations' column
# Split the data into training and test sets (adjust the test_size and random_state as needed)
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state=42)
# Display the shapes of the resulting sets to verify the split
print("X_train shape:", X_train.shape)
print("X_test shape:", X_test.shape)
print("Y_train shape:", Y_train.shape)
print("Y_test shape:", Y_test.shape)
```

```
X_train shape: (69209, 236)
X_test shape: (17303, 236)
Y_train shape: (69209,)
Y_test shape: (17303,)
```

In this code, we first separate the features (X) and the target variable (Y) from the dataset. Then, we use train_test_split to split the data into training and test sets. The test_size parameter determines the proportion of the data that will be allocated to the test set, and random_state is set to a specific value (e.g., 42) to ensure reproducibility.

Feature Scaling:

Feature scaling is an important preprocessing step in many machine learning algorithms. You can use the StandardScaler from scikit-learn to scale your features so that they have a mean of 0 and a standard deviation of 1.



```
[ ]: from sklearn.preprocessing import StandardScaler
     # Assuming you have your training and test data (X_train and X_test) defined
     # Create a StandardScaler instance
     scaler = StandardScaler()
     # Fit the scaler on the training data and transform both training and test data
     X_train_scaled = scaler.fit_transform(X_train)
     X_test_scaled = scaler.transform(X_test)

     # Display the scaled features
     print("Scaled X_train:")
     print(X_train_scaled)
     print("Scaled X_test:")
     print(X_test_scaled)
```

In this code, we first create a StandardScaler instance. We then fit the scaler on the training data using the fit_transform method, and apply the same transformation to both the training and test data using the transform method. This ensures that the scaling is consistent between the two sets.

TEAM-MATES: RITHIKA B

SOWMIYA G