Week-7

Indian Air Quality Index - Dasboard

By Arijit Dhali [Linkedin]

Since industrialization, there has been an increasing concern about environmental pollution. As mentioned in the WHO report 7 million premature deaths annually are linked to air pollution, air pollution is the world's largest single environmental risk. Moreover as reported in the NY Times article, India's Air Pollution Rivals China's as World's Deadliest it has been found that India's air pollution is deadlier than China's. We will explore India's air pollution levels more granularly using this dataset.

This data is combined(across the years and states) and is largely a clean version of the Historical Daily Ambient Air Quality Data released by the Ministry of Environment and Forests and Central Pollution Control Board of India under the National Data Sharing and Accessibility Policy (NDSAP).

Importing Libraries & Inspection

Here we import the necessary libraries for array operations (`numpy`) and working with datasets (`pandas`). We also import the `warnings module` to suppress warning messages, providing cleaner output. Now we proceed to ignore warnings in the code.

In [1]:
```
import numpy as np                          # Import numpy for array operations
import pandas as pd                         # Import pandas for working with datasets
import warnings                             # Import warnings module to handle warnings
warnings.filterwarnings('ignore')           # Ignore warning messages
```

Here we set the file path for the CSV file containing air quality data. Now, we use the `pd.read_csv` function from pandas to read the CSV file into a DataFrame named ' `air` ', specifying the encoding as ' `unicode_escape` '.

In [2]:
```
url = '/content/drive/MyDrive/Prepinsta Winter Internship/Week 7/Air Quality.csv'  # Set the file path
air = pd.read_csv(url, encoding='unicode_escape')                                   # Read the CSV file into a DataFrame
using pandas
```

Here we use the sample method to randomly display 5 rows from the ' `air` ' DataFrame for a quick overview of the data.

In [3]:
```
air.sample(5)  # Display a random sample of 5 rows from the 'air' DataFrame
```

Out[3]:

	stn_code	sampling_date	state	location	agency	type	so2	no2	rspm	spm	location_monitoring_station	pm2_5	date
148649	684.0	15-01-14	Karnataka	Bidar	Karnataka State Pollution Control Board	Residential, Rural and other Areas	7.0	14.0	63.0	NaN	KSPCB Office Premises, Bidar	NaN	2014-01-15
120382	563	23-05-13	Himachal Pradesh	Baddi	Himachal Pradesh State Environment Proection &...	Residential, Rural and other Areas	2.0	24.0	167.0	NaN	Housing Board	NaN	2013-05-23
106135	35	July - M072001	Himachal Pradesh	Shimla	Himachal Pradesh Pollution Control Boar	Residential, Rural and other Areas	2.0	6.4	NaN	NaN	NaN	NaN	2001-07-01
271686	NaN	7/6/2005	Odisha	Bhubaneshwar	NaN	Residential and others	NaN	11.0	51.0	80.0	IRC	NaN	2005-06-07
343154	409.0	04-05-15	Rajasthan	Jaipur	Rajasthan State Pollution Control Board	Residential, Rural and other Areas	8.0	43.0	180.0	NaN	Regional Office (North), RSPCB, Vidyadhar Nag...	NaN	2015-05-04

Data Manipulation

Here we retrieve the unique values in the ' `type` ' column of the ' `air` ' DataFrame using the unique method, showing the different types of air quality data available in the dataset.

In [4]:
```
air['type'].unique()  # Get unique values in the 'type' column of the 'air' DataFrame
```

Out[4]:
```
array(['Residential, Rural and other Areas', 'Industrial Area', nan,
       'Sensitive Area', 'Industrial Areas', 'Residential and others',
       'Sensitive Areas', 'Industrial', 'Residential', 'RIRUO',
       'Sensitive'], dtype=object)
```

Here we replace multiple values in the ' `type` ' column of the ' `air` ' DataFrame to achieve consistency and simplify the categories.

In [5]:
```
# Replace values in the 'type' column for consistency
air['type'].replace('Residential, Rural and other Areas','Residential',inplace = True)
air['type'].replace('Residential and others','Residential',inplace = True)
air['type'].replace('Industrial Areas','Industrial',inplace = True)
air['type'].replace('Industrial Area','Industrial',inplace = True)
air['type'].replace('Sensitive Area','Sensitive',inplace = True)
air['type'].replace('Sensitive Areas','Sensitive',inplace = True)
```

Now we check the unique values in the ' `type` ' column of the ' `air` ' DataFrame to confirm that the specified replacements have been successfully applied, ensuring consistency in the categories.

In [6]:
```
air['type'].unique()  # Verify unique values in the 'type' column after replacements
```

Out[6]:
```
array(['Residential', 'Industrial', nan, 'Sensitive', 'RIRUO'],
      dtype=object)
```

Here we retrieve the unique values in the ' `state` ' column of the ' `air` ' DataFrame using the unique method, showing the different states represented in the dataset.

In [7]:
```
air['state'].unique()  # Get unique values in the 'state' column of the 'air' DataFrame
```

Out[7]:
```
array(['Andhra Pradesh', 'Arunachal Pradesh', 'Assam', 'Bihar',
       'Chandigarh', 'Chhattisgarh', 'Dadra & Nagar Haveli',
       'Daman & Diu', 'Delhi', 'Goa', 'Gujarat', 'Haryana',
       'Himachal Pradesh', 'Jammu & Kashmir', 'Jharkhand', 'Karnataka',
       'Kerala', 'Madhya Pradesh', 'Maharashtra', 'Manipur', 'Meghalaya',
       'Mizoram', 'Nagaland', 'Odisha', 'Puducherry', 'Punjab',
       'Rajasthan', 'Sikkim', 'Tamil Nadu', 'Telangana', 'Uttar Pradesh',
       'Uttarakhand', 'Uttaranchal', 'West Bengal',
       'andaman-and-nicobar-islands', 'Lakshadweep', 'Tripura'],
      dtype=object)
```

Here we replace a specific value in the ' `state` ' column of the ' `air` ' DataFrame to achieve consistency. After the replacement, we check the unique values in the ' `state` ' column to confirm the change.

In [8]:
```
# Replace a specific value in the 'state' column for consistency
air['state'].replace('andaman-and-nicobar-islands', 'Andaman and Nicobar Islands', inplace=True)
air['state'].unique()  # Verify unique values in the 'state' column after replacement
```

Out[8]:
```
array(['Andhra Pradesh', 'Arunachal Pradesh', 'Assam', 'Bihar',
       'Chandigarh', 'Chhattisgarh', 'Dadra & Nagar Haveli',
       'Daman & Diu', 'Delhi', 'Goa', 'Gujarat', 'Haryana',
       'Himachal Pradesh', 'Jammu & Kashmir', 'Jharkhand', 'Karnataka',
       'Kerala', 'Madhya Pradesh', 'Maharashtra', 'Manipur', 'Meghalaya',
       'Mizoram', 'Nagaland', 'Odisha', 'Puducherry', 'Punjab',
       'Rajasthan', 'Sikkim', 'Tamil Nadu', 'Telangana', 'Uttar Pradesh',
       'Uttarakhand', 'Uttaranchal', 'West Bengal',
       'Andaman and Nicobar Islands', 'Lakshadweep', 'Tripura'],
      dtype=object)
```

Here we process the ' `date` ' column by converting it to datetime format and extracting the ' `year` ' component. Missing ' `year` ' values are filled using forward fill, and the column is then converted to the integer type. Finally, we check for any remaining null values in the ' `year` ' column.

In [9]:
```
# Convert the 'date' column to datetime format and extract the 'year' column
air['date'] = pd.to_datetime(air['date'])
air['year'] = air['date'].dt.year

# Fill missing 'year' values using forward fill and convert to integer type
air['year'].fillna(method='ffill', inplace=True)
air['year'] = air['year'].astype(int)

air['year'].isnull().sum()  # Check for any remaining null values in the 'year' column
```

Out[9]:
```
0
```

Here we create a DataFrame named ' `missing` ' to show the proportion of missing values in each column of the ' `air` ' DataFrame. The columns are then displayed in descending order based on the proportion of missing values.

In [10]:
```
# Create a DataFrame to show the proportion of missing values in each column
missing = pd.DataFrame(air.isna().sum() / len(air))
missing.columns = ['Proportion']

# Display the columns sorted by the proportion of missing values in descending order
print(missing.sort_values(by='Proportion', ascending=False))
```

```
                             Proportion
pm2_5                          0.978625
spm                            0.544788
agency                         0.343049
stn_code                       0.330647
rspm                           0.092307
so2                            0.079510
location_monitoring_station    0.063090
no2                            0.037254
type                           0.012377
date                           0.000016
sampling_date                  0.000007
location                       0.000007
state                          0.000000
year                           0.000000
```

Here we define a function `state_wise` that takes a state as an argument and calculates and prints the median values for Industrial, Residential, and Sensitive types for that state using the 'air' DataFrame. The function returns these median values.

In [11]:
```
def state_wise(states):
    # Group the 'air' DataFrame by 'state' and 'type'
    grouped = air.groupby(['state', 'type'])

    # Create a dictionary from the grouped data
    data_dict = dict(list(grouped))

    # Extract median values for Industrial, Residential, and Sensitive types for the specified state
    kar_ind = data_dict[(states, 'Industrial')].median()
    kar_res = data_dict[(states, 'Residential')].median()
    kar_sen = data_dict[(states, 'Sensitive')].median()

    # Print and return the median values for each type
    print(kar_ind, kar_res, kar_sen)
    return kar_ind, kar_res, kar_sen
```

Here we call the `state_wise` function with the argument ' `Andhra Pradesh` ' and store the returned median values for Industrial, Residential, and Sensitive types in respective variables (`kar_ind` , `kar_res` , `kar_sen`).

In [12]:
```
# Call the state_wise function for 'Andhra Pradesh' and store the results in variables
kar_ind, kar_res, kar_sen = state_wise('Andhra Pradesh')
```

```
stn_code     584.0
so2            5.4
no2           22.2
rspm          76.0
spm          214.0
pm2_5          NaN
year        2011.0
dtype: float64 stn_code     467.0
so2            5.0
no2           20.0
rspm          78.0
spm          192.0
pm2_5          NaN
year        2010.0
dtype: float64 stn_code     389.0
so2            4.6
no2           13.0
rspm          51.0
spm          130.0
pm2_5          NaN
year        2011.0
dtype: float64
```

Here we use the ' `loc` ' method to fill missing ' `no2` ' and ' `so2` ' values in the ' `Andhra Pradesh` ' state for Industrial, Residential, and Sensitive types using the respective median values

In [13]:
```
# Fill missing 'so2' values in 'Andhra Pradesh' for Industrial, Residential, and Sensitive types
air.loc[(air['state'] == 'Andhra Pradesh') & (air['type'] == 'Industrial') & (air['so2'].isnull()), 'so2'] = kar_ind
['so2']
air.loc[(air['state'] == 'Andhra Pradesh') & (air['type'] == 'Residential') & (air['so2'].isnull()), 'so2'] = kar_re
s['so2']
air.loc[(air['state'] == 'Andhra Pradesh') & (air['type'] == 'Sensitive') & (air['so2'].isnull()), 'so2'] = kar_sen
['so2']
```

In [14]:
```
# Fill missing 'no2' values in 'Andhra Pradesh' for Industrial, Residential, and Sensitive types
air.loc[(air['state'] == 'Andhra Pradesh') & (air['type'] == 'Industrial') & (air['no2'].isnull()), 'no2'] = kar_ind
['no2']
air.loc[(air['state'] == 'Andhra Pradesh') & (air['type'] == 'Residential') & (air['no2'].isnull()), 'no2'] = kar_re
s['no2']
air.loc[(air['state'] == 'Andhra Pradesh') & (air['type'] == 'Sensitive') & (air['no2'].isnull()), 'no2'] = kar_sen
['no2']
```

Here we print the number of missing values in the ' `rspm` ' and ' `spm` ' columns of the ' `air` ' DataFrame.

In [15]:
```
# Print the number of missing values in the 'rspm' and 'spm' columns
print(air['rspm'].isnull().sum())
print(air['spm'].isnull().sum())
```

```
40222
237387
```

Here, we group the ' `air` ' DataFrame by ' `location` ' and ' `type` ', then iterate through the groups. Within each group, we sort the values by 'date' and forward-fill missing values in the 'rspm' and 'spm' columns. The results are concatenated into a new DataFrame named ' `data` '.

In [16]:
```
# Group 'air' DataFrame by 'location' and 'type' and create a new DataFrame with forward-filled 'rspm' and 'spm' val
ues
df1 = dict(list(air.groupby(['location', 'type'])))
data = pd.DataFrame()

# Iterate through groups, sort by 'date', and forward-fill 'rspm' and 'spm' values
for key in df1:
    df2 = df1[key].sort_values('date')
    df2['rspm'].fillna(method='ffill', inplace=True)
    df2['spm'].fillna(method='ffill', inplace=True)
    data = pd.concat([data, df2])
```

Here, we group the ' `data` ' DataFrame by ' `location` ' and ' `type` ', then iterate through the groups. Within each group, we sort the values by 'date' and backward-fill missing values in the 'rspm' and 'spm' columns. The results are concatenated into a new DataFrame named ' `data1` '.

In [17]:
```
# Group 'data' DataFrame by 'location' and 'type' and create a new DataFrame with backward-filled 'rspm' and 'spm' v
alues
df1 = dict(list(data.groupby(['location', 'type'])))
data1 = pd.DataFrame()

# Iterate through groups, sort by 'date', and backward-fill 'rspm' and 'spm' values
for key in df1:
    df2 = df1[key].sort_values('date')
    df2['rspm'].fillna(method='bfill', inplace=True)
    df2['spm'].fillna(method='bfill', inplace=True)
    data1 = pd.concat([data1, df2])
```

Here we display the first few rows of the ' `data1` ' DataFrame to inspect the changes made, including the backward-filled ' `rspm` ' and ' `spm` ' values.

In [18]:
```
data1.head()  # Display the first few rows of the 'data1' DataFrame
```

Out[18]:

	stn_code	sampling_date	state	location	agency	type	so2	no2	rspm	spm	location_monitoring_station	pm2_5	date	year
101624	SAMP	05-01-15	Gujarat	ANKLESHWAR	Gujarat State Pollution Control Board	RIRUO	13.0	20.0	82.0	NaN	Panoli Ind.Asso. & Emergency Response Centre,P...	26.0	2015-01-05	2015
101541	SAMP	06-01-15	Gujarat	ANKLESHWAR	Gujarat State Pollution Control Board	RIRUO	13.0	20.0	91.0	NaN	GIDC OFFICE TERRACE, GIDC ESTATE JHAGADIA,ANKL...	37.0	2015-01-06	2015
101625	SAMP	08-01-15	Gujarat	ANKLESHWAR	Gujarat State Pollution Control Board	RIRUO	14.0	21.0	70.0	NaN	Panoli Ind.Asso. & Emergency Response Centre,P...	29.0	2015-01-08	2015
101542	SAMP	09-01-15	Gujarat	ANKLESHWAR	Gujarat State Pollution Control Board	RIRUO	14.0	21.0	78.0	NaN	GIDC OFFICE TERRACE, GIDC ESTATE JHAGADIA,ANKL...	33.0	2015-01-09	2015
101626	SAMP	12-01-15	Gujarat	ANKLESHWAR	Gujarat State Pollution Control Board	RIRUO	14.0	21.0	82.0	NaN	Panoli Ind.Asso. & Emergency Response Centre,P...	25.0	2015-01-12	2015

Here we print the number of missing values in the ' `rspm` ' and ' `spm` ' columns of the ' `data1` ' DataFrame after the backward-fill operations.

In [19]:
```
# Print the number of missing values in the 'rspm' and 'spm' columns of the 'data1' DataFrame
print(data1['rspm'].isnull().sum())
print(data1['spm'].isnull().sum())
```

```
4102
47909
```

Here, we group the ' `data1` ' DataFrame by ' `state` ' and ' `type` ', then iterate through the groups. Within each group, missing values in ' `rspm` ' and ' `spm` ' columns are filled with the group-wise medians. The results are concatenated into a new DataFrame named ' `data2` '.

In [20]:
```
# Group 'data1' DataFrame by 'state' and 'type' and create a new DataFrame with median-filled 'rspm' and 'spm' value
s
df1 = dict(list(data1.groupby(['state', 'type'])))
data2 = pd.DataFrame()

# Iterate through groups and fill missing 'rspm' and 'spm' values with group-wise medians
for key in df1:
    df2 = df1[key]
    df2['rspm'].fillna(df2['rspm'].median(), inplace=True)
    df2['spm'].fillna(df2['spm'].median(), inplace=True)
    data2 = pd.concat([data2, df2])
```

Here we print the number of missing values in the ' `rspm` ' and ' `spm` ' columns of the ' `data2` ' DataFrame after filling missing values with group-wise medians.

In [21]:
```
# Print the number of missing values in the 'rspm' and 'spm' columns of the 'data2' DataFrame
print(data2['rspm'].isnull().sum())
print(data2['spm'].isnull().sum())
```

```
182
1972
```

Here we display the entire ' `data2` ' DataFrame to inspect the final dataset after filling missing values with group-wise medians.

In [22]:
```
data2  # Display the 'data2' DataFrame
```

Out[22]:

	stn_code	sampling_date	state	location	agency	type	so2	no2	rspm	spm	location_monitoring_station	pm2_5	date	year
1	151.0	February - M021990	Andhra Pradesh	Hyderabad	NaN	Industrial	3.1	7.0	130.3	82.000000	NaN	NaN	1990-02-01	1990
4	151.0	March - M031990	Andhra Pradesh	Hyderabad	NaN	Industrial	4.7	7.5	130.3	82.000000	NaN	NaN	1990-03-01	1990
7	151.0	April - M041990	Andhra Pradesh	Hyderabad	NaN	Industrial	4.7	8.7	130.3	82.000000	NaN	NaN	1990-04-01	1990
9	151.0	May - M051990	Andhra Pradesh	Hyderabad	NaN	Industrial	4.0	8.9	130.3	82.000000	NaN	NaN	1990-05-01	1990
12	151.0	June - M061990	Andhra Pradesh	Hyderabad	NaN	Industrial	5.6	11.8	130.3	82.000000	NaN	NaN	1990-06-01	1990
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
434695	650	12-12-15	West Bengal	South Suburban	West Bengal State Pollution Control Board	Residential	2.0	44.0	93.0	577.666667	Baruipur, South Suburban	NaN	2015-12-12	2015
434696	650	14-12-15	West Bengal	South Suburban	West Bengal State Pollution Control Board	Residential	2.0	47.0	145.0	577.666667	Baruipur, South Suburban	NaN	2015-12-14	2015
434697	650	18-12-15	West Bengal	South Suburban	West Bengal State Pollution Control Board	Residential	4.0	55.0	208.0	577.666667	Baruipur, South Suburban	NaN	2015-12-18	2015
434698	650	20-12-15	West Bengal	South Suburban	West Bengal State Pollution Control Board	Residential	3.0	49.0	206.0	577.666667	Baruipur, South Suburban	NaN	2015-12-20	2015
434699	650	26-12-15	West Bengal	South Suburban	West Bengal State Pollution Control Board	Residential	3.0	50.0	173.0	577.666667	Baruipur, South Suburban	NaN	2015-12-26	2015

430349 rows × 14 columns

Here, we group the ' `data2` ' DataFrame by ' `type` ', then iterate through the groups. Within each group, missing values in ' `rspm` ' and ' `spm` ' columns are filled with the group-wise medians. The results are concatenated into a new DataFrame named ' `data3` '.

In [23]:
```
# Group 'data2' DataFrame by 'type' and create a new DataFrame with median-filled 'rspm' and 'spm' values
df1 = dict(list(data2.groupby('type')))
data3 = pd.DataFrame()

# Iterate through groups and fill missing 'rspm' and 'spm' values with group-wise medians
for key in df1:
    df2 = df1[key]
    df2['rspm'].fillna(df2['rspm'].median(), inplace=True)
    df2['spm'].fillna(df2['spm'].median(), inplace=True)
    data3 = pd.concat([data3, df2])
```

In [24]:
```
data3
```

Out[24]:

	stn_code	sampling_date	state	location	agency	type	so2	no2	rspm	spm	location_monitoring_station	pm2_5	date	year
1	151.0	February - M021990	Andhra Pradesh	Hyderabad	NaN	Industrial	3.1	7.0	130.3	82.0	NaN	NaN	1990-02-01	1990
4	151.0	March - M031990	Andhra Pradesh	Hyderabad	NaN	Industrial	4.7	7.5	130.3	82.0	NaN	NaN	1990-03-01	1990
7	151.0	April - M041990	Andhra Pradesh	Hyderabad	NaN	Industrial	4.7	8.7	130.3	82.0	NaN	NaN	1990-04-01	1990
9	151.0	May - M051990	Andhra Pradesh	Hyderabad	NaN	Industrial	4.0	8.9	130.3	82.0	NaN	NaN	1990-05-01	1990
12	151.0	June - M061990	Andhra Pradesh	Hyderabad	NaN	Industrial	5.6	11.8	130.3	82.0	NaN	NaN	1990-06-01	1990
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
406659	1	29-12-15	Uttar Pradesh	Agra	Central Pollution Control Board	Sensitive	2.0	27.0	211.0	448.0	Taj Mahal, Agra	NaN	2015-12-29	2015
406660	1	30-12-15	Uttar Pradesh	Agra	Central Pollution Control Board	Sensitive	2.0	34.0	352.0	448.0	Taj Mahal, Agra	NaN	2015-12-30	2015
408838	415	30-12-15	Uttar Pradesh	Agra	Central Pollution Control Board	Sensitive	4.0	36.0	427.0	448.0	DIC Nunhai, Agra	NaN	2015-12-30	2015
406661	1	31-12-15	Uttar Pradesh	Agra	Central Pollution Control Board	Sensitive	2.0	31.0	363.0	448.0	Taj Mahal, Agra	NaN	2015-12-31	2015
409070	417	31-12-15	Uttar Pradesh	Agra	Central Pollution Control Board	Sensitive	2.0	26.0	338.0	448.0	Rambagh, Agra	NaN	2015-12-31	2015

430349 rows × 14 columns

Here we print the number of missing values in the ' `rspm` ' and ' `spm` ' columns of the ' `data3` ' DataFrame after filling missing values with group-wise medians.

In [25]:
```
# Print the number of missing values in the 'rspm' and 'spm' columns of the 'data3' DataFrame
print(data3['rspm'].isnull().sum())
print(data3['spm'].isnull().sum())
```

```
0
1304
```

Here we display the count of each type in the ' `data3` ' DataFrame using the `value_counts` method. This provides an overview of the distribution of types in the final processed dataset.

In [26]:
```
data3['type'].value_counts()  # Display the count of each type in the 'data3' DataFrame
```

Out[26]:
```
Residential    265963
Industrial     148071
Sensitive       15011
RIRUO            1304
Name: type, dtype: int64
```

Data Saving

Here we reset the index of the ' `data3` ' DataFrame and drop some unnecessary columns to obtain a cleaner and more concise dataset. The modified DataFrame is displayed using `head()` .

In [27]:
```
# Reset index and drop unnecessary columns from the 'data3' DataFrame
data3.reset_index(inplace=True)
data3.drop(columns=['index', 'stn_code', 'sampling_date', 'agency', 'location_monitoring_station'], inplace=True)
data3.head()
```

Out[27]:

	state	location	type	so2	no2	rspm	spm	pm2_5	date	year
0	Andhra Pradesh	Hyderabad	Industrial	3.1	7.0	130.3	82.0	NaN	1990-02-01	1990
1	Andhra Pradesh	Hyderabad	Industrial	4.7	7.5	130.3	82.0	NaN	1990-03-01	1990
2	Andhra Pradesh	Hyderabad	Industrial	4.7	8.7	130.3	82.0	NaN	1990-04-01	1990
3	Andhra Pradesh	Hyderabad	Industrial	4.0	8.9	130.3	82.0	NaN	1990-05-01	1990
4	Andhra Pradesh	Hyderabad	Industrial	5.6	11.8	130.3	82.0	NaN	1990-06-01	1990

Here we check for missing values in the ' `data3` ' DataFrame to ensure that the dataset is free of any remaining null values after the preprocessing steps.

In [28]:
```
data3.isnull().sum()  # Check for missing values in the 'data3' DataFrame
```

Out[28]:
```
state            0
location         0
type             0
so2          33516
no2          15312
rspm             0
spm           1304
pm2_5       421035
date             4
year             0
dtype: int64
```

Here we use the to_csv method to save a copy of the final cleaned data stored in the ' `data3` ' DataFrame to a CSV file named ' `air_quality_cleaned_data.csv` '. The index=False parameter ensures that the index is not included in the saved file.

In [29]:
```
data3.to_csv('air_quality_cleaned_data.csv', index=False)  # Save a copy of the final cleaned data to a CSV file
```