pandas-dsp

August 31, 2024

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

1. Getting Familiar with Pandas:

[2]:
```
# Creating a Series from a list
data_series = pd.Series([10, 20, 30, 40, 50])
print("Series from a list:\n", data_series)
```

```
Series from a list:
 0    10
1    20
2    30
3    40
4    50
dtype: int64
```

[3]:
```
# Creating a DataFrame from a dictionary
data_dict = {'Name': ['John', 'Arun', 'Peter', 'Rama'],
             'Age': [28, 24, 35, 32],
             'City': ['New York', 'Paris', 'Berlin', 'London']}
data_frame = pd.DataFrame(data_dict)
print("\nDataFrame from a dictionary:\n", data_frame)
```

```
DataFrame from a dictionary:
     Name  Age      City
0   John   28  New York
1   Arun   24     Paris
2  Peter   35    Berlin
3   Rama   32    London
```

Common Operations

[5]:
```
# Selecting data
print("\nSelecting the 'Name' column:\n", data_frame['Name'])
```

```
Selecting the 'Name' column:
 0     John
```

```
1     Arun
2    Peter
3     Rama
Name: Name, dtype: object
```

[6]:
```
# Filtering rows
print("\nFiltering rows where Age > 30:\n", data_frame[data_frame['Age'] > 30])
```

```
Filtering rows where Age > 30:
     Name  Age    City
2  Peter   35  Berlin
3   Rama   32  London
```

[7]:
```
# Modifying data
data_frame['Age'] += 1
print("\nDataFrame after modifying 'Age' column:\n", data_frame)
```

```
DataFrame after modifying 'Age' column:
     Name  Age      City
0    John   29  New York
1    Arun   25     Paris
2  Peter   36    Berlin
3   Rama   33    London
```

2. Data Handling with Pandas

[9]:
```
# Reading data from a CSV file
data = pd.read_csv('simple data.csv')
```

[10]:
```
# Handling missing data
print("\nData with missing values:\n", data.isnull().sum())
```

```
Data with missing values:
 Name      0
Age       2
City      0
Gender    0
Salary    3
dtype: int64
```

[11]:
```
# Filling missing values with a specific value
data_filled = data.fillna(0)
print("\nData after filling missing values:\n", data_filled)
```

```
Data after filling missing values:
```

```
     Name   Age      City  Gender   Salary
0    John  28.0  New York    Male  55000.0
1   Aruna   0.0     Paris  Female  60000.0
2   Peter  35.0    Berlin    Male      0.0
3   indhu  32.0    London  Female  70000.0
4   Vinay  30.0  New York    Male      0.0
5    Emma  27.0    Berlin  Female  48000.0
6    Rama   0.0    London    Male  62000.0
7   Lucky  31.0     Paris  Female      0.0
```

[12]:
```
# Removing duplicates
data_no_duplicates = data.drop_duplicates()
print("\nData after removing duplicates:\n", data_no_duplicates)
```

```
Data after removing duplicates:
     Name   Age      City  Gender   Salary
0    John  28.0  New York    Male  55000.0
1   Aruna   NaN     Paris  Female  60000.0
2   Peter  35.0    Berlin    Male      NaN
3   indhu  32.0    London  Female  70000.0
4   Vinay  30.0  New York    Male      NaN
5    Emma  27.0    Berlin  Female  48000.0
6    Rama   NaN    London    Male  62000.0
7   Lucky  31.0     Paris  Female      NaN
```

[13]:
```
# Converting data types
data['Age'] = data['Age'].astype(float)
print("\nData after converting 'Age' to float:\n", data.dtypes)
```

```
Data after converting 'Age' to float:
 Name       object
Age        float64
City        object
Gender      object
Salary     float64
dtype: object
```

3. Data Analysis with Pandas

[14]:
```
# Generating summary statistics
print("\nSummary statistics:\n", data.describe())
```

```
Summary statistics:
              Age        Salary
count   6.000000      5.000000
mean   30.500000  59000.000000
```

```
std     2.880972   8185.352772
min    27.000000  48000.000000
25%    28.500000  55000.000000
50%    30.500000  60000.000000
75%    31.750000  62000.000000
max    35.000000  70000.000000
```

[15]:
```python
# Grouping data and calculating the mean only for numeric columns
grouped_data = data.groupby('City').mean(numeric_only=True)
print("\nAverage Age and Salary by City:\n", grouped_data)
```

```
Average Age and Salary by City:
            Age   Salary
City
Berlin    31.0  48000.0
London    32.0  66000.0
New York  29.0  55000.0
Paris     31.0  60000.0
```

[16]:
```python
# Merging DataFrames
more_data = pd.DataFrame({'City': ['New York', 'Paris', 'Berlin'],
                          'Population': [8500000, 2141000, 3615000]})
merged_data = pd.merge(data, more_data, on='City')
print("\nMerged DataFrame:\n", merged_data)
```

```
Merged DataFrame:
     Name   Age      City  Gender   Salary  Population
0    John  28.0  New York    Male  55000.0     8500000
1  Aruna   NaN     Paris  Female  60000.0     2141000
2   Peter  35.0    Berlin    Male      NaN     3615000
3   Vinay  30.0  New York    Male      NaN     8500000
4    Emma  27.0    Berlin  Female  48000.0     3615000
5   Lucky  31.0     Paris  Female      NaN     2141000
```

[18]:
```python
# Concatenating DataFrames
additional_data = pd.DataFrame({'Name': ['Tom', 'Jerry'],
                                'Age': [22, 19],
                                'City': ['Tokyo', 'Osaka']})
concatenated_data = pd.concat([data, additional_data])
print("\nConcatenated DataFrame:\n", concatenated_data)
```

```
Concatenated DataFrame:
     Name   Age      City  Gender   Salary
0    John  28.0  New York    Male  55000.0
1  Aruna   NaN     Paris  Female  60000.0
```

```
2  Peter  35.0    Berlin    Male      NaN
3  indhu  32.0    London  Female  70000.0
4  Vinay  30.0  New York    Male      NaN
5   Emma  27.0    Berlin  Female  48000.0
6   Rama   NaN    London    Male  62000.0
7  Lucky  31.0     Paris  Female      NaN
0    Tom  22.0     Tokyo     NaN      NaN
1  Jerry  19.0     Osaka     NaN      NaN
```

[19]:
```python
# Selecting only numeric columns
numeric_data = data.select_dtypes(include=['number'])

# Grouping by 'City' and calculating the mean
grouped_data = numeric_data.groupby(data['City']).mean()
print("\nAverage Age and Salary by City:\n", grouped_data)
```

```
Average Age and Salary by City:
            Age   Salary
City
Berlin    31.0  48000.0
London    32.0  66000.0
New York  29.0  55000.0
Paris     31.0  60000.0
```

4. Application in Data Science

Advantages

Efficient Data Handling: Pandas can handle large datasets efficiently using its powerful data structures.

Rich Functionality: It offers a vast array of functions for data cleaning, transformation, and analysis.

Seamless Integration: Pandas integrates well with other Python libraries such as NumPy and Matplotlib, which are often used in data science.

Real-world Examples

Data Cleaning: Handling missing data, removing duplicates, and transforming data types.

Exploratory Data Analysis (EDA): Generating summary statistics, visualizing data, and identifying trends.

Data Merging: Combining data from multiple sources to create a comprehensive dataset for analysis.

Pandas' ability to streamline these tasks makes it invaluable in data science, allowing professionals to focus on insights rather than the mechanics of data manipulation.