Python Pandas Filtering

Filtering data is a common operation in data analysis. Pandas allows us to filter data based on different conditions.

We can filter the data in Pandas in two main ways:

- By column names (Labels)
- By the actual data inside (Values)

Filter Data By Labels

We can use the `filter()` function to select columns by their names or labels. Let's look at an example.

```
import pandas as pd

# create a DataFrame
data = {'Name': ['Alice', 'Bob', 'Charlie', 'David'],
        'Department': ['HR', 'Marketing', 'Marketing', 'IT'],
        'Salary': [50000, 60000, 55000, 70000]}

df = pd.DataFrame(data)

# display the original DataFrame
print("Original DataFrame:")
print(df)
print("\n")

# use the filter() method to select columns based on a condition
filtered_df = df.filter(items=['Name', 'Salary'])

# display the filtered DataFrame
print("Filtered DataFrame:")
print(filtered_df)
```

```
Original DataFrame:
      Name Department  Salary
0    Alice         HR   50000
1      Bob  Marketing   60000
2  Charlie  Marketing   55000
3    David         IT   70000


Filtered DataFrame:
      Name  Salary
0    Alice   50000
1      Bob   60000
```

```
2  Charlie   55000
3    David   70000
```

In this example, we used `filter()` to select the columns `Name` and `Salary` using their columns names.

Filter Data By Values

We can also filter data by values. Some of the common ways to filter data by values are:

- Using logical operator
- The `isin()` method
- The `str` Accessor
- The `query()` method

Logical Operators

You can filter rows based on column values using logical operators. For example,

```
import pandas as pd

# create a sample DataFrame
data = {'Name': ['Alice', 'Bob', 'Charlie', 'David'],
        'Department': ['HR', 'Marketing', 'Marketing', 'IT'],
        'Salary': [50000, 60000, 55000, 70000]}

df = pd.DataFrame(data)

# display the original DataFrame
print("Original DataFrame:")
print(df)
print("\n")

# use logical operators to filter
filtered_df = df[df.Salary > 55000]

# display the filtered DataFrame
print("Filtered DataFrame:")
print(filtered_df)
```

```
Original DataFrame:
      Name Department  Salary
0    Alice         HR   50000
1      Bob  Marketing   60000
2  Charlie  Marketing   55000
3    David         IT   70000


Filtered DataFrame:
```

```
    Name Department  Salary
1    Bob  Marketing   60000
3  David         IT   70000
```

In the above example, we selected the rows based on the condition `Salary > 55000` using logical operator `>`.

The isin() Method

The `isin()` method provides another way to filter data using column values. Let's look at an example.

```
import pandas as pd

# create a sample DataFrame
data = {'Name': ['Alice', 'Bob', 'Charlie', 'David'],
        'Department': ['HR', 'Marketing', 'Marketing', 'IT'],
        'Salary': [50000, 60000, 55000, 70000]}

df = pd.DataFrame(data)

# display the original DataFrame
print("Original DataFrame:")
print(df)
print("\n")

# use isin() method
departments = ['HR', 'IT']
filtered_df = df[df.Department.isin(departments)]

# display the filtered DataFrame
print("Filtered DataFrame:")
print(filtered_df)
```

```
Original DataFrame:
      Name Department  Salary
0    Alice         HR   50000
1      Bob  Marketing   60000
2  Charlie  Marketing   55000
3    David         IT   70000


Filtered DataFrame:
    Name Department  Salary
0  Alice         HR   50000
3  David         IT   70000
```

In this example, we selected the rows whose `Department` values are present in the `departments` list.

The str Accessor

We can effectively filter rows based on string values using the `str` accessor. Let's look at an example.

```
import pandas as pd

# create a sample DataFrame
data = {'Name': ['Alice', 'Bob', 'Charlie', 'David'],
        'Department': ['HR', 'Marketing', 'Marketing', 'IT'],
        'Salary': [50000, 60000, 55000, 70000]}

df = pd.DataFrame(data)

# display the original DataFrame
print("Original DataFrame:")
print(df)
print("\n")

# use str accessor
filtered_df = df[df.Department.str.contains('Market')]

# display the filtered DataFrame
print("Filtered DataFrame:")
print(filtered_df)
```

```
Original DataFrame:
      Name Department  Salary
0    Alice         HR   50000
1      Bob  Marketing   60000
2  Charlie  Marketing   55000
3    David         IT   70000


Filtered DataFrame:
      Name Department  Salary
1      Bob  Marketing   60000
2  Charlie  Marketing   55000
```

Here, we filtered the rows based on a string value. We only selected the rows whose `Department` values contained the string `Market`.

The query() Method

This is the most flexible method for filtering a dataframe based on column values.

A query containing the filtering conditions can be passed as a string to the `query()` method.

Let's look at an example.

```
import pandas as pd

# create a sample DataFrame
data = {'Name': ['Alice', 'Bob', 'Charlie', 'David'],
        'Department': ['HR', 'Marketing', 'Marketing', 'IT'],
        'Salary': [50000, 60000, 55000, 70000]}

df = pd.DataFrame(data)

# display the original DataFrame
print("Original DataFrame:")
print(df)
print("\n")

# use query method
filtered_df = df.query('Salary > 55000 and Department == "Marketing"')

# display the filtered DataFrame
print("Filtered DataFrame:")
print(filtered_df)
```

```
Original DataFrame:
      Name Department  Salary
0    Alice         HR   50000
1      Bob  Marketing   60000
2  Charlie  Marketing   55000
3    David         IT   70000


Filtered DataFrame:
  Name Department  Salary
1  Bob  Marketing   60000
```

In this example, we selected the rows with `Salary > 55000` and `Department == "Marketing"` using the `query()` method.

Pandas Sort

Sorting is a fundamental operation in data manipulation and analysis that involves arranging data in a specific order.

Sorting is crucial for tasks such as organizing data for better readability, identifying patterns, making comparisons, and facilitating further analysis.

Sort DataFrame in Pandas

In Pandas, we can use the `sort_values()` function to sort a DataFrame. For example,

```
import pandas as pd
```

```
data = {'Name': ['Alice', 'Bob', 'Charlie'],
        'Age': [28, 22, 25]}
df = pd.DataFrame(data)

# sort DataFrame by Age in ascending order
sorted_df = df.sort_values(by='Age')

print(sorted_df.to_string(index=False))
```

```
   Name  Age
    Bob   22
Charlie   25
  Alice   28
```

In the above example, `df.sort_values(by='Age')` sorts the `df` DataFrame based on the values in the Age column in ascending order. And the result is stored in the `sorted_df` variable.

To sort values in descending order, we use the ascending parameter as:

```python sorted_df = df.sort_values(by='Age', ascending=False)

**Note:** The `.to_string(index=False)` is used to display values without the index.

## Sort Pandas DataFrame by Multiple Columns

We can also sort DataFrame by multiple columns in Pandas. When we sort a Pandas DataFrame by multiple columns, the sorting is done with a priority given to the order of the columns listed.

To sort by multiple columns in Pandas, you can pass the desired columns as a list to the `by` parameter in the `sort_values()` method. Here's how we do it.

```
import pandas as pd

data = {'Name': ['Alice', 'Bob', 'Charlie', 'David'],
        'Age': [25, 22, 30, 22],
        'Score': [85, 90, 75, 80]}

df = pd.DataFrame(data)

1. Sort DataFrame by 'Age' and then by 'Score' (Both in ascending
order)
df1 = df.sort_values(by=['Age', 'Score'])

print("Sorting by 'Age' (ascending) and then by 'Score' (ascending):\
n")
print(df1.to_string(index=False))

print()
# 2. Sort DataFrame by 'Age' in ascending order, and then by 'Score'
in descending order
df2 = df.sort_values(by=['Age', 'Score'], ascending=[True, False])
```
```

```
print("Sorting by 'Age' (ascending) and then by 'Score' (descending):\
n")
print(df2.to_string(index=False))
```

```
Sorting by 'Age' (ascending) and then by 'Score' (ascending):

   Name  Age  Score
  David   22     80
    Bob   22     90
  Alice   25     85
Charlie   30     75

Sorting by 'Age' (ascending) and then by 'Score' (descending):

   Name  Age  Score
    Bob   22     90
  David   22     80
  Alice   25     85
Charlie   30     75
```

Here,

1. `df1` shows the default sorting behavior (both columns in ascending order).
2. `df2` shows custom sorting, where `Age` is in ascending and `Score` is in descending order.

Sort Pandas Series

In Pandas, we can use the `sort_values()` function to sort a Series. For example,

```
import pandas as pd

ages = pd.Series([28, 22, 25], name='Age')

# sort Series in ascending order
sorted_ages = ages.sort_values()

print(sorted_ages.to_string(index=False))
```

```
22
25
28
```

Here, `ages.sort_values()` sorts the `ages` Series in ascending order. The sorted result is assigned to the `sorted_ages` variable.

#index Sort Pandas DataFrame Using sort_index()

We can also sort by the index of a DataFrame in Pandas using the `sort_index()` function.

The `sort_index()` function is used to sort a DataFrame or Series by its index. This is useful for organizing data in a logical order, improving query performance, and ensuring consistent data representation.

Let's look at an example.

```
import pandas as pd

data = {'Name': ['Alice', 'Bob', 'Charlie'],
        'Age': [28, 22, 25]}
# create a DataFrame with a non-sequential index
df = pd.DataFrame(data, index=[2, 0, 1])

print("Original DataFrame:")
print(df.to_string(index=True))
print("\n")

# sort DataFrame by index in ascending order
sorted_df = df.sort_index()

print("Sorted DataFrame by index:")
print(sorted_df.to_string(index=True))
```

```
Original DataFrame:
      Name  Age
2    Alice   28
0      Bob   22
1  Charlie   25


Sorted DataFrame by index:
      Name  Age
0      Bob   22
1  Charlie   25
2    Alice   28
```

In the above example, we have created the `df` DataFrame with a non-sequential index from the data dictionary.

The `index` parameter is specified as `[2, 0, 1]`, meaning that the rows will not have a default sequential index **(0, 1, 2)**, but rather the provided non-sequential index.

Then we sorted the `df` DataFrame by its index in ascending order using the `sort_index()` method.

Pandas Correlation

Correlation is a statistical concept that quantifies the degree to which two variables are related to each other.

Correlation can be calculated in Pandas using the `corr()` function.

Let's look at an example.

```
import pandas as pd

# create dataframe
data = {
    "Temperature": [22, 25, 32, 28, 30],
    "Ice_Cream_Sales": [105, 120, 135, 130, 125]
}

df = pd.DataFrame(data)

# calculate correlation matrix
print(df.corr())
```

```
                 Temperature  Ice_Cream_Sales
Temperature         1.000000         0.923401
Ice_Cream_Sales     0.923401         1.000000
```

```
In this example, we used the `corr()` method on the DataFrame `df` to
calculate the correlation coefficients between the columns.

The output is a correlation matrix that displays the correlation
coefficients between all pairs of columns in the dataframe. In this
case, there are only two columns, so the matrix is **2x2**.

Here, the correlation coefficient between `Temperature` and
`Ice_Cream_Sales` is **0.923401**, which is positive. This indicates
that as the temperature increases, the ice cream sales also increase.

The coefficient value of **1.000000** along the diagonal represents
the correlation of each column with itself.
```