filterpandas dataframemultiple condition

In [2]:
```
import pandas as pd
# assign data
dataFrame = pd.DataFrame({'Name': [' SMRUTI  ', ' MONICA  ', ' RATNA  ',
                                   '  ROSS    ', 'CHANDLER', ' JOEY    '],

                          'Age': [30, 35, 37, 33, 34, 30],

                          'Salary': [100000, 93000, 88000, 120000, 94000, 95000],

                          'JOB': ['DESIGNER', 'CHEF', 'MASUS', 'PALENTOLOGY',
                                  'IT', 'ARTIST']})

# display dataframe
display(dataFrame)
```

	Name	Age	Salary	JOB
0	SMRUTI	30	100000	DESIGNER
1	MONICA	35	93000	CHEF
2	RATNA	37	88000	MASUS
3	ROSS	33	120000	PALENTOLOGY
4	CHANDLER	34	94000	IT
5	JOEY	30	95000	ARTIST

In [4]:
```
# filter dataframe
display(dataFrame.loc[(dataFrame['Salary']>=100000) & (dataFrame['Age']< 40) & (dat
                    ['Name','JOB']])
```

	Name	JOB
0	SMRUTI	DESIGNER

In [8]:
```
# filter dataframe
display(dataFrame.query('Salary  <= 100000 & Age < 40 & JOB.str.startswith("C").val
```

	Name	Age	Salary	JOB
1	MONICA	35	93000	CHEF

Pandas Boolean indexing multiple conditions standard way ("Boolean indexing" works with values in a column only)

In [11]:

```
# filter dataframe
display(dataFrame[(dataFrame['Salary']>=100000) & (dataFrame['Age']<40) & dataFrame
```

	Name	Age	Salary
3	ROSS	33	120000

Pandas Merging, Joining and Concatenating

In [18]:

```
data1 = {'Name': ['Jai', 'Princi', 'Gaurav', 'Anuj'],
         'Age': [27, 24, 22, 32],
         'Address': ['Nagpur', 'Kanpur', 'Allahabad', 'Kannuaj'],
         'Qualification': ['Msc', 'MA', 'MCA', 'Phd']}

data2 = {'Name': ['Abhi', 'Ayushi', 'Dhiraj', 'Hitesh'],
         'Age': [17, 14, 12, 52],
         'Address': ['Nagpur', 'Kanpur', 'Allahabad', 'Kannuaj'],
         'Qualification': ['Btech', 'B.A', 'Bcom', 'B.hons']}

df = pd.DataFrame(data1, index=[0, 1, 2, 3])

df1 = pd.DataFrame(data2, index=[4, 5, 6, 7])

print(df, "\n\n",df1)
```

```
     Name  Age    Address Qualification
0     Jai   27     Nagpur           Msc
1  Princi   24     Kanpur            MA
2  Gaurav   22  Allahabad           MCA
3    Anuj   32    Kannuaj           Phd

      Name  Age    Address Qualification
4     Abhi   17     Nagpur         Btech
5   Ayushi   14     Kanpur           B.A
6   Dhiraj   12  Allahabad          Bcom
7   Hitesh   52    Kannuaj        B.hons
```

concat--> data adding

frames = [df, df1]

res1 = pd.concat(frames) res1

Concatenating DataFrames by Setting Logic on Axes

In [23]:

```
res2 = pd.concat([df, df1], axis=1, join='inner')
```

```
res2
```

Out[23]:

Name	Age	Address	Qualification	Name	Age	Address	Qualification

In [25]:

```
df1 = pd.DataFrame({
    "id": [1, 2, 3],
    "name": ["A", "B", "C"]
})

df2 = pd.DataFrame({
    "id": [2, 3, 4],
    "marks": [80, 85, 90]
})

result = pd.merge(df1, df2, on="id", how="inner")
print(result)
```

```
   id name  marks
0   2    B     80
1   3    C     85
```

In [27]:

```
result = pd.merge(df1, df2, on="id", how="left")
print(result)
```

```
   id name  marks
0   1    A    NaN
1   2    B   80.0
2   3    C   85.0
```

In [29]:

```
result = pd.merge(df1, df2, on="id", how="right")
print(result)
```

```
   id name  marks
0   2    B     80
1   3    C     85
2   4  NaN     90
```

Full Outer Join (Outer Join) 📘 Definition

A Full Outer Join returns all rows from both DataFrames. If no match → NaN.

👉 Keeps everything from both sides.

🧠 Easy Example

All students + all exam records, no data loss.

In [31]:

```
result = pd.merge(df1, df2, on="id", how="outer")
print(result)
```

```
   id name  marks
0   1    A    NaN
1   2    B   80.0
2   3    C   85.0
3   4  NaN   90.0
```

Index Join in Pandas 📘 Definition

An Index Join joins DataFrames using their index values instead of column values.

👉 Uses row index to match data.

🧠 Easy Example

Matching data based on roll number as index.

In [35]:

```
df1 = pd.DataFrame({
    "name": ["A", "B", "C"]
}, index=[1, 2, 3])

df2 = pd.DataFrame({
    "marks": [80, 85, 90]
}, index=[2, 3, 4])

result = df1.join(df2, how="inner")
print(result)
```

```
  name  marks
2    B     80
3    C     85
```

Sorting a data

In [38]:

```
import pandas as pd
data = {'Name': ['Smruti', 'Ratna', 'Puja', 'Rakhi'],
        'Age': [20, 21, 35, 40],
        'Score': [85, 90, 95, 80]}
df = pd.DataFrame(data)

sorted_df = df.sort_values(by='Age')
print(sorted_df)
```

```
     Name  Age  Score
0  Smruti   20     85
1   Ratna   21     90
2    Puja   35     95
3   Rakhi   40     80
```

In [40]:

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

```
     Name  Age  Score
3   Rakhi   40     80
2    Puja   35     95
1   Ratna   21     90
0  Smruti   20     85
```

In [42]:

```
sorted_df = df.sort_values(by=['Age', 'Score'])
print(sorted_df)
```

```
      Name  Age  Score
0  Smruti   20     85
1   Ratna   21     90
2    Puja   35     95
3   Rakhi   40     80
```

In [46]:
```python
data_with_nan = {"Name": ["Smruti", "Puja", "rojaline", "Dibya"],"Age": [28, 22, No
df_nan = pd.DataFrame(data_with_nan)

sorted_df = df_nan.sort_values(by="Age", na_position="first")
print(sorted_df)
```

```
       Name   Age
2  rojaline   NaN
1      Puja  22.0
3     Dibya  22.0
0    Smruti  28.0
```

In [48]:
```python
# index sorting
data = {'Name': ['Alice', 'Bob', 'Charlie', 'David'],
        'Age': [25, 30, 35, 40],
        'Score': [85, 90, 95, 80]}
df = pd.DataFrame(data)

df_sorted_by_index = df.sort_index()
print(df_sorted_by_index)
```

```
      Name  Age  Score
0    Alice   25     85
1      Bob   30     90
2  Charlie   35     95
3    David   40     80
```

In [50]:
```python
df_sorted_by_index_desc = df.sort_index(ascending=False)
print(df_sorted_by_index_desc)
```

```
      Name  Age  Score
3    David   40     80
2  Charlie   35     95
1      Bob   30     90
0    Alice   25     85
```

Quicksort is a divide-and-conquer sorting algorithm that selects a pivot and arranges data into smaller and larger values around it, making it efficient for sorting columns in a DataFrame.

In [53]:
```python
data = {
    "Name": ["Alice", "Bob", "Charlie", "David", "Eve"],
    "Age": [28, 22, 25, 22, 28],
    "Score": [85, 90, 95, 80, 88]
}
df = pd.DataFrame(data)

sorted_df = df.sort_values(by='Age', kind='quicksort')
print(sorted_df)
```

```
      Name  Age  Score
1      Bob   22     90
3    David   22     80
2  Charlie   25     95
0    Alice   28     85
4      Eve   28     88
```

MergeSort (kind='mergesort'): Divides the dataset into smaller subarrays, sorts them and then merges them back together in sorted order.

sorted_df = df.sort_values(by='Age', kind='mergesort') print(sorted_df)

HeapSort (kind= 'heapsort'): It is another comparison-based sorting algorithm that builds a heap data structure to systematically extract the largest or smallest element and reorder the dataset.

In [59]:
```
sorted_df = df.sort_values(by='Age', kind='heapsort')
print(sorted_df)
```

```
      Name  Age  Score
1      Bob   22     90
3    David   22     80
2  Charlie   25     95
4      Eve   28     88
0    Alice   28     85
```

In [61]:
```
sorted_df = df.sort_values(by='Name', key=lambda col: col.str.lower())
print(sorted_df)
```

```
      Name  Age  Score
0    Alice   28     85
1      Bob   22     90
2  Charlie   25     95
3    David   22     80
4      Eve   28     88
```

pandas pivot table

In [66]:

```
# creating dataframe
df = pd.DataFrame({'Product': ['Carrots', 'Broccoli', 'Banana', 'Banana',
                               'Beans', 'Orange', 'Broccoli', 'Banana'],
                   'Category': ['Vegetable', 'Vegetable', 'Fruit', 'Fruit',
                                'Vegetable', 'Fruit', 'Vegetable', 'Fruit'],
                   'Quantity': [8, 5, 3, 4, 5, 9, 11, 8],
                   'Amount': [270, 239, 617, 384, 626, 610, 62, 90]})
df
```

Out[66]:

	Product	Category	Quantity	Amount
0	Carrots	Vegetable	8	270
1	Broccoli	Vegetable	5	239
2	Banana	Fruit	3	617
3	Banana	Fruit	4	384
4	Beans	Vegetable	5	626
5	Orange	Fruit	9	610
6	Broccoli	Vegetable	11	62
7	Banana	Fruit	8	90

In [68]:

```
# Get the Total Sales of Each Product
pivot = df.pivot_table(index=['Product'],
                       values=['Amount'],
                       aggfunc='sum')
print(pivot)
```

```
          Amount
Product
Banana      1091
Beans        626
Broccoli     301
Carrots      270
Orange       610
```

In [70]:

```
# : Get the Total Sales of Each Category
# creating pivot table of total
# sales category-wise aggfunc = 'sum'
pivot = df.pivot_table(index=['Category'],
                       values=['Amount'],
                       aggfunc='sum')
print(pivot)
```

```
           Amount
Category
Fruit        1701
Vegetable    1197
```

In [72]:

```
#  Get Total Sales by Category and Product Both
pivot = df.pivot_table(index=['Product', 'Category'],
                       values=['Amount'], aggfunc='sum')
print(pivot)
```

```
                    Amount
Product  Category
Banana   Fruit        1091
Beans    Vegetable     626
Broccoli Vegetable     301
Carrots  Vegetable     270
Orange   Fruit         610
```

In [74]:

```
# Get the Mean, Median, Minimum Sale by Category
# 'mean', 'min'} will get median, mean and
# minimum of sales respectively
pivot = df.pivot_table(index=['Category'], values=['Amount'],
                       aggfunc={'median', 'mean', 'min'})
print(pivot)
```

```
           Amount
             mean median min
Category
Fruit      425.25  497.0  90
Vegetable  299.25  254.5  62
```

In [76]:

```
#Get the Mean, Median, Minimum Sale by Product
pivot = df.pivot_table(index=['Product'], values=['Amount'],
                       aggfunc={'median', 'mean', 'min'})
print(pivot)
```

```
              Amount
                mean median  min
Product
Banana    363.666667  384.0   90
Beans     626.000000  626.0  626
Broccoli  150.500000  150.5   62
Carrots   270.000000  270.0  270
Orange    610.000000  610.0  610
```

In [ ]: