# Pandas Tutorial: Filtering, Sorting, Indexing, and Aggregation

## 1. Display Settings with set\_option

Use `pd.set\_option()` to customize display options such as column width or number of columns displayed.  
Example:  
```python  
import pandas as pd  
pd.set\_option('display.max\_columns', None)  
pd.set\_option('display.max\_rows', 100)  
```

## 2. Filtering and Sorting

- `str.contains()`: Filters rows based on substring match in a column.  
```python  
df[df['column'].str.contains('substring', na=False)]  
```  
  
- `isin()`: Filters rows where column values are in a list.  
```python  
df[df['column'].isin(['value1', 'value2'])]  
```  
  
- `.filter(items=[])`: Select specific columns by name.  
```python  
df.filter(items=['col1', 'col2'])  
```  
  
- `.filter(like='text')`: Select columns that contain a substring.  
```python  
df.filter(like='2023', axis=1)  
```  
  
- `sort\_values()`: Sort by one or more columns.  
```python  
df.sort\_values('col1') # single column  
df.sort\_values(['col1', 'col2'], ascending=[True, False]) # multiple columns  
```

## 3. Indexing with loc and iloc

- `.loc[]`: Label-based indexing.  
```python  
df.loc[0, 'col1'] # row index label 0, column 'col1'  
df.loc[:, ['col1', 'col2']] # all rows, selected columns  
```  
  
- `.iloc[]`: Integer position-based indexing.  
```python  
df.iloc[0, 1] # first row, second column  
df.iloc[:, [0, 2]] # all rows, first and third columns  
```

## 4. Grouping and Aggregating

Group data by one or more columns and apply aggregation functions like mean, sum, count, etc.  
  
```python  
df.groupby('column')['value'].mean()  
df.groupby(['col1', 'col2'])['value'].agg(['mean', 'sum', 'count'])  
```

Grouping in pandas is done using the `.groupby()` function. It allows you to split the data into groups based on one or more column values and then apply aggregation functions like `mean()`, `sum()`, `count()`, `max()`, `min()`, etc., to analyze each group separately.  
  
🔹 Example 1: Group by a single column  
```python  
df.groupby('column')['value'].mean()  
```  
- Purpose: Groups the DataFrame by unique values in 'column' and computes the mean of the 'value' column for each group.  
- Output: A Series where the index is the unique values from 'column', and values are the group-wise mean.  
  
🔹 Example 2: Group by multiple columns  
```python  
df.groupby(['col1', 'col2'])['value'].agg(['mean', 'sum', 'count'])  
```  
- Purpose: Groups the data by every unique combination of 'col1' and 'col2', and computes mean, sum, and count for the 'value' column.  
- Output: A DataFrame with a multi-level index (col1, col2) and the aggregated statistics in columns.  
  
🔹 Real-world example  
```python  
import pandas as pd  
  
data = {  
 'Department': ['Sales', 'Sales', 'HR', 'HR', 'IT', 'IT'],  
 'Gender': ['M', 'F', 'M', 'F', 'M', 'F'],  
 'Salary': [50000, 55000, 60000, 58000, 70000, 72000]  
}  
  
df = pd.DataFrame(data)  
  
# Group by Department and Gender, then aggregate Salary  
result = df.groupby(['Department', 'Gender'])['Salary'].agg(['mean', 'sum', 'count'])  
  
print(result)  
```  
Output:  
```  
 mean sum count  
Department Gender   
HR F 58000 58000 1  
 M 60000 60000 1  
IT F 72000 72000 1  
 M 70000 70000 1  
Sales F 55000 55000 1  
 M 50000 50000 1  
```