```
import pandas as pd
import numpy as np


arr = np.array([1,2,3,4])
print(arr)
```

```
[1 2 3 4]
```

Advanced Indexing and Selection

> **Multi-level indexing with hierarchical indexing**

- Creating DataFrames with multiple levels of indexes to work with multi-dimensional data.

```
import pandas as pd

data = {
    'A': [1, 2, 3, 4, 5, 6],
    'B': [7, 8, 9, 10, 11, 12],
    'C': [13, 14, 15, 16, 17, 18]
}
index = pd.MultiIndex.from_tuples([('X', 2020), ('X', 2021), ('Y', 2020), ('Y', 2021), ('Z', 2020),
df = pd.DataFrame(data, index=index)

print(df['A']['X'])
```

```
Year
2020    1
2021    2
Name: A, dtype: int64
```

> **Indexing and slicing with loc[] and iloc[]:**

- Accessing DataFrame elements using labeled and integer-based indexing.

```
# Using loc[] for labeled indexing
print(df.loc[('X', 2020), 'A'])

# Using iloc[] for integer-based indexing
print(df.iloc[0, 1])
```

```
1
7
```

> **Boolean indexing and filtering:**

- Selecting data from a DataFrame based on specified conditions.

```
# Boolean indexing to filter rows with 'B' values greater than 9
filtered_df = df[df['B'] > 9]
print(filtered_df)
```

```
            A   B   C
City Year
Y    2021   4  10  16
Z    2020   5  11  17
     2021   6  12  18
```

Combining DataFrames

> **Merging and joining DataFrames with merge() and join():**

- Combining DataFrames based on common columns.

```
df1 = pd.DataFrame({'ID': [1, 2, 3], 'Name': ['Alice', 'Bob', 'Charlie']})
df2 = pd.DataFrame({'ID': [2, 3, 4], 'Age': [25, 30, 22]})

# Merging based on 'ID'
merged_df = pd.merge(df1, df2, on='ID', how='inner')
print(merged_df)
```

```
   ID     Name  Age
0   2      Bob   25
1   3  Charlie   30
```

> **Concatenating DataFrames using concat():**

- Combining DataFrames along a specified axis (rows or columns).

```
df1 = pd.DataFrame({'A': [1, 2], 'B': [3, 4]})
df2 = pd.DataFrame({'A': [5, 6], 'B': [7, 8]})

# Concatenating along rows
concatenated_df = pd.concat([df1, df2])
print(concatenated_df)
```

```
   A  B
0  1  3
1  2  4
0  5  7
1  6  8
```

Data Manipulation

> **Filtering and subsetting data based on conditions**

- Extracting specific subsets of data using conditional statements.

```
d = {'id':[1,2,3],'name':['A','B','C'],'Age':[22,26,38],'City':['Kolhapur','Pune','Mumbai']}
df = pd.DataFrame(d)

# Filter rows where column 'Age' is greater than 25
filtered_data = df[df['Age'] > 25]
print(filtered_data)
```

```
   id name  Age    City
1   2    B   26    Pune
2   3    C   38  Mumbai
```

> **Sorting and ranking data:**

- Ordering data based on column values and assigning ranks to data elements.

```
# Sorting DataFrame based on 'Age' in descending order
sorted_df = df.sort_values(by='Age', ascending=False)

# Ranking 'Age' within the DataFrame
df['Rank'] = df['Age'].rank(ascending=False)

print(df)
```

```
   id name  Age      City  Rank
0   1    A   22  Kolhapur   3.0
1   2    B   26      Pune   2.0
2   3    C   38    Mumbai   1.0
```

> **Aggregating and summarizing data using groupby()**

- Grouping data based on one or more columns and applying aggregation functions.

```
# Grouping data by 'City' and calculating mean 'Age'
grouped_df = df.groupby('City')['Age'].mean()
print(grouped_df)
```

```
City
Kolhapur    22.0
Mumbai      38.0
Pune        26.0
Name: Age, dtype: float64
```

> **Merging, joining, and concatenating DataFrames:**

- Combining data from different DataFrames.

```
# Merging based on 'ID'
merged_df = pd.merge(df1, df2, on='ID', how='inner')
print(merged_df)
```

```
   ID     Name  Age
0   2      Bob   25
1   3  Charlie   30
```

> **Pivoting and melting data for reshaping:**

- Changing the layout of the DataFrame to perform analysis efficiently.

```
data = {
    'Id': [1,2,3,4,5],
    'Name': ['A', 'B', 'C', 'D', 'E'],
    'City': ['Kolhapur', 'Pune', 'Sangli', 'Satara', 'Mumbai'],
    'Age': [21, 23, 56, 34, 68],
    'Salary': [23000, 45000, 35000, 78000, 56000],
    'Year': [2021, 2018, 2023, 2015, 2019]
}

df = pd.DataFrame(data)

# Pivoting DataFrame to show 'Age' for each 'City'
pivoted_df = df.pivot(index='City', columns='Year', values='Age')

print(pivoted_df)
```

```
Year      2015  2018  2019  2021  2023
City
Kolhapur   NaN   NaN   NaN  21.0   NaN
Mumbai     NaN   NaN  68.0   NaN   NaN
Pune       NaN  23.0   NaN   NaN   NaN
Sangli     NaN   NaN   NaN   NaN  56.0
Satara    34.0   NaN   NaN   NaN   NaN
```

```
# Melting DataFrame to convert columns into rows
melted_df = pd.melt(df, id_vars=['City', 'Year'], value_vars=['Age', 'Salary'], var_name='Attribute'

print(melted_df)
```

```
        City  Year Attribute  Value
0  Kolhapur  2021       Age     21
1      Pune  2018       Age     23
2    Sangli  2023       Age     56
3    Satara  2015       Age     34
4    Mumbai  2019       Age     68
5  Kolhapur  2021    Salary  23000
6      Pune  2018    Salary  45000
7    Sangli  2023    Salary  35000
8    Satara  2015    Salary  78000
9    Mumbai  2019    Salary  56000
```

Advanced Data Manipulation

> **Multi-level indexing and hierarchical data**

- Creating DataFrames with multiple levels of indexes to handle complex datasets.

```
import pandas as pd

# Creating a DataFrame with multi-level index
data = {
    'A': [1, 2, 3, 4, 5, 6],
    'B': [7, 8, 9, 10, 11, 12],
    'C': [13, 14, 15, 16, 17, 18]
}
index = pd.MultiIndex.from_tuples([('X', 2020), ('X', 2021), ('Y', 2020), ('Y', 2021), ('Z', 2020),
df = pd.DataFrame(data, index=index)

print(df['A']['X'])
```

```
     Year
     2020    1
     2021    2
     Name: A, dtype: int64
```

> **Pivot tables and cross-tabulations:**

- Transforming data and summarizing it using pivot tables and cross-tabulations.

```
# Creating a DataFrame
data = {
    'City': ['A', 'A', 'B', 'B', 'A', 'B'],
    'Year': [2020, 2021, 2020, 2021, 2020, 2021],
    'Sales': [100, 150, 120, 200, 80, 250]
}
df = pd.DataFrame(data)

# Creating a pivot table to summarize 'Sales' based on 'City' and 'Year'
pivot_table = df.pivot_table(values='Sales', index='City', columns='Year', aggfunc='sum')
print(pivot_table)
```

```
     Year  2020  2021
     City
     A      180   150
     B      120   450
```

> **Handling text data and regular expressions:**

- Dealing with text data and applying regular expressions for pattern matching and extraction.

```
# Creating a DataFrame with text data
data = {
    'Text': ['apple', 'orange', 'banana', 'grape', 'peach']
}
df = pd.DataFrame(data)

# Using str.contains() to filter rows with text containing 'a'
filtered_df = df[df['Text'].str.contains('a')]
print(filtered_df)
```

```
          Text
     0   apple
```

```
1  orange
2  banana
3   grape
4   peach
```

> **Working with JSON and other data:**

- Reading, manipulating, and analyzing data in JSON format and other formats like XML, HTML, etc.

```
# Reading JSON data into a DataFrame
import json

json_data = '{"name": "John", "age": 30, "city": "New York"}'
df = pd.DataFrame(json.loads(json_data), index=[0])
print(df)
```

```
   name  age      city
0  John   30  New York
```

Data Aggregation and Grouping

> **Grouping data using groupby()**

- Splitting data into groups based on one or more categorical variables.

```
# Creating a DataFrame
data = {
    'City': ['A', 'B', 'A', 'B', 'A', 'B'],
    'Sales': [100, 120, 80, 150, 200, 250]
}
df = pd.DataFrame(data)

# Grouping data by 'City'
grouped_df = df.groupby('City')

for i in grouped_df:
    print(i)
```

```
('A',   City  Sales
0    A    100
2    A     80
4    A    200)
('B',   City  Sales
1    B    120
3    B    150
5    B    250)
```

> **Applying aggregation functions to groups:**

- Calculating summary statistics for each group.

```
# Calculating the total sales for each city
grouped_df = df.groupby('City')['Sales'].sum()
print(grouped_df)
```

```
City
A    380
B    520
Name: Sales, dtype: int64
```

> **Performing multi-level aggregation:**

- Aggregating data at multiple levels of grouping.

```
data = {
    'Id': [1,2,3,4,5,6],
    'City': ['Kolhapur', 'Pune', 'Sangli', 'Mumbai', 'Satara', 'Kolhapur'],
    'Year': [2012, 2016, 2013, 2015, 2017, 2018],
    'Sales': [23000, 43000, 30000, 40000, 65000, 34000]
}

df = pd.DataFrame(data)

# Grouping data by 'City' and 'Year', and calculating the total sales for each group
grouped_df = df.groupby(['City', 'Year'])['Sales'].sum()
print(grouped_df)
```

```
City      Year
Kolhapur  2012    23000
          2018    34000
Mumbai    2015    40000
Pune      2016    43000
Sangli    2013    30000
Satara    2017    65000
Name: Sales, dtype: int64
```

> **Grouping data with groupby() and split-apply-combine operations**

- Applying transformations to groups and combining the results.

```
# Applying multiple aggregation functions to 'Sales' for each city
grouped_df = df.groupby('City')['Sales'].agg(['sum', 'mean', 'max'])
print(grouped_df)
```

```
      sum        mean  max
City
A     380  126.666667  200
B     520  173.333333  250
```

> **Aggregation functions (e.g., mean, sum, count, min, max):**

- Using various aggregation functions to calculate statistics on grouped data.

```
# Calculating the average and total sales for each city
grouped_df = df.groupby('City')['Sales'].agg(['mean', 'sum'])
print(grouped_df)
```

```
             mean  sum
City
A     126.666667  380
B     173.333333  520
```