PANDAS

Pandas is a fundamental Python library for data manipulation and analysis. Since we're already proficient in Python and data science, We will keep the explanations concise and focused on practical examples.

1. Introduction to Pandas

Pandas provides two main data structures:

- **Series**: A one-dimensional labeled array.
- **DataFrame**: A two-dimensional labeled data structure (like a table).

Installation

pip install pandas

Importing Pandas

import pandas as pd

2. Pandas Series

A **Series** is a one-dimensional labeled array.

Creating a Series

```
import pandas as pd
data = [10, 20, 30, 40]
s = pd.Series(data)
print(s)
```

Custom Index

```
s = pd.Series(data, index=['a', 'b', 'c', 'd'])
print(s)
```

Accessing Elements

```
print(s['b'])   # Access using label
print(s[1])     # Access using index
```

3. Pandas DataFrame

A **DataFrame** is a two-dimensional labeled data structure.

Creating a DataFrame

```
data = {
    'Name': ['Alice', 'Bob', 'Charlie'],
    'Age': [25, 30, 35],
    'Salary': [50000, 60000, 70000]
}
df = pd.DataFrame(data)
print(df)
```

Reading Data

```
df = pd.read_csv('data.csv')   # Read CSV file
df = pd.read_excel('data.xlsx') # Read Excel file
df = pd.read_json('data.json')  # Read JSON file
```

Writing Data

```
df.to_csv('output.csv', index=False)
df.to_excel('output.xlsx', index=False)
df.to_json('output.json')
```

4. Basic DataFrame Operations

```
print(df.head())   # First 5 rows
print(df.tail())   # Last 5 rows
print(df.shape)    # Dimensions (rows, columns)
print(df.columns)  # Column names
```

```
print(df.dtypes)   # Data types of columns
print(df.info())   # Summary of DataFrame
print(df.describe())  # Statistical summary
```

Selecting Columns

```
print(df['Age'])    # Single column
print(df[['Name', 'Salary']])  # Multiple columns
```

Selecting Rows

```
print(df.loc[1])   # Select row by label
print(df.iloc[1])  # Select row by index
```

Filtering Data

```
df_filtered = df[df['Age'] > 30]  # Filter rows
print(df_filtered)
```

5. Modifying Data

Adding Columns

```
df['Bonus'] = df['Salary'] * 0.1
```

Updating Values

```
df.loc[df['Name'] == 'Alice', 'Salary'] = 55000
```

Removing Columns

```
df.drop(columns=['Bonus'], inplace=True)
```

Removing Rows

```
df.drop(index=1, inplace=True)
```

6. Handling Missing Data

```
df.dropna()       # Remove missing values
df.fillna(0)      # Replace NaN with 0
df.fillna(df.mean())  # Replace NaN with column mean
```

7. Grouping and Aggregation

```
df.groupby('Age')['Salary'].mean()
df.groupby('Age').agg({'Salary': 'sum', 'Bonus': 'mean'})
```

8. Sorting Data

```
df.sort_values('Age', ascending=False, inplace=True)
```

9. Merging & Joining Data

```
df1 = pd.DataFrame({'ID': [1, 2, 3], 'Name': ['A', 'B', 'C']})
df2 = pd.DataFrame({'ID': [1, 2, 3], 'Salary': [5000, 6000, 7000]})
df_merged = pd.merge(df1, df2, on='ID', how='inner')  # Inner, Left, Right,
Outer
```

10. Pivot Tables & Crosstabs

```
df.pivot_table(index='Age', values='Salary', aggfunc='sum')
pd.crosstab(df['Age'], df['Salary'])
```

11. Working with Dates

```
df['Date'] = pd.to_datetime(df['Date'])
df['Year'] = df['Date'].dt.year
df['Month'] = df['Date'].dt.month
```

12. Visualization with Pandas

```
import matplotlib.pyplot as plt
df['Salary'].plot(kind='bar')
plt.show()
```

13. Advanced Data Cleaning

- **Handling Duplicates**

```
df.drop_duplicates(inplace=True)  # Remove duplicate rows
df.duplicated()  # Check for duplicates
```

Handling Outliers

```
import numpy as np

Q1 = df['Salary'].quantile(0.25)
Q3 = df['Salary'].quantile(0.75)
IQR = Q3 - Q1
df_filtered = df[(df['Salary'] >= Q1 - 1.5 * IQR) & (df['Salary'] <= Q3 + 1.5 *
IQR)]
```

Handling Categorical Data

```
df['Category'] = df['Category'].astype('category')  # Convert to category
df['Category'] = df['Category'].cat.codes  # Convert categorical to numerical
```

14. MultiIndexing (Hierarchical Indexing)

- MultiIndexing allows you to work with multiple levels of row/column labels.

Creating a MultiIndex DataFrame

```
arrays = [['A', 'A', 'B', 'B'], [1, 2, 1, 2]]
index = pd.MultiIndex.from_arrays(arrays, names=('Group', 'Subgroup'))
df = pd.DataFrame({'Values': [10, 20, 30, 40]}, index=index)
print(df)
```

Accessing MultiIndex Data

```
print(df.loc['A'])    # Get all data from Group A
print(df.loc[('A', 1)])  # Get specific subgroup
```

15. Time Series Analysis

Creating a DateTime Index

```
df['Date'] = pd.to_datetime(df['Date'])
df.set_index('Date', inplace=True)
```

Resampling Data

```
df.resample('M').mean()  # Aggregate monthly data
df.resample('D').sum()   # Aggregate daily data
```

Shifting & Rolling Windows

```
df['Shifted'] = df['Values'].shift(1)  # Shift previous row values
df['Rolling_Avg'] = df['Values'].rolling(window=3).mean()  # Rolling average
```

16. Performance Optimization

Using Vectorization (Avoid Loops)

```
df['New_Col'] = df['Salary'] * 1.1  # Faster than looping
```

Using `apply()` Efficiently

```
df['Bonus'] = df['Salary'].apply(lambda x: x * 0.1)  # Apply function to a column
```

Using `.eval()` for Faster Computation

```
df.eval("New_Col = Salary * 1.1", inplace=True)
```

17. Custom Functions & Apply

- `apply()` is useful for applying row-wise or column-wise functions.

```
def categorize_age(age):
    if age < 30:
        return "Young"
    elif age < 50:
        return "Middle-Aged"
    else:
        return "Senior"

df['Age_Group'] = df['Age'].apply(categorize_age)
```

Using `map()` for Column-Wise Mapping

```
df['Category'] = df['Category'].map({'A': 1, 'B': 2, 'C': 3})
```

Using `applymap()` for Element-Wise Mapping

```
df[['Col1', 'Col2']] = df[['Col1', 'Col2']].applymap(lambda x: x * 2)
```

18. Working with Large Datasets

- For large datasets, read data in chunks instead of loading everything into memory.

Reading Large CSV in Chunks

```
chunk_size = 1000
for chunk in pd.read_csv('large_file.csv', chunksize=chunk_size):
    process(chunk)  # Process each chunk separately
```

19. Visualization with Seaborn & Pandas

```
import seaborn as sns
import matplotlib.pyplot as plt

sns.histplot(df['Salary'])  # Histogram
sns.boxplot(x=df['Salary'])  # Boxplot to detect outliers
sns.scatterplot(x=df['Age'], y=df['Salary'])  # Scatter plot
plt.show()
```

Next Steps?

Now you've mastered **100% of Pandas!** 🏆🔥
To practice:

✅ Work with real datasets (CSV, JSON, Excel)
✅ Build data analysis projects
✅ Try Pandas in **machine learning**
✅ Try working with real datasets like Kaggle datasets.
✅ Practice using Pandas on different file formats.
✅ Experiment with Pandas built-in functions.

Let me know if you want **real-world exercises or projects!** 🚀