1. **Introduction to pandas**

**Objectives:**

* **Use Python libraries** to handle and analyze data.
* **Learn to perform basic data manipulation** with pandas.
* Understand **data analysis techniques** such as filtering, sorting, and aggregating data.
* Gain insight into **data visualization** using matplotlib.

**Introduction to Pandas: Creating, Reading, and Manipulating DataFrames**

**What is pandas?**

**pandas** is an open-source Python library that provides high-performance, easy-to-use data structures and data analysis tools. It is primarily used for working with tabular data in the form of DataFrames. It allows you to efficiently manipulate, clean, and analyze structured data.

* **Series**: A one-dimensional labeled array that can hold any data type (integers, strings, etc.).
* **DataFrame**: A two-dimensional, size-mutable, potentially heterogeneous tabular data structure with labeled axes (rows and columns).

**Key Operations in pandas:**

* **Creating DataFrames**: You can create a DataFrame by loading data from various formats like CSV, Excel, or SQL databases. For example:

import pandas as pd

# Creating DataFrame from a dictionary

data = {'Name': ['Alice', 'Bob', 'Charlie'],

'Age': [25, 30, 35],

'City': ['New York', 'Los Angeles', 'Chicago']}

df = pd.DataFrame(data)

print(df)

**Reading Data**: pandas allows you to load data from multiple sources, such as:

# Read data from a CSV file

df = pd.read\_csv('data.csv')

**Manipulating DataFrames**:

* **Selecting Columns**: You can select a specific column like this:

df['Age'] # Selects the Age column

* **Filtering Rows**: You can filter rows based on a condition:

df[df['Age'] > 30] # Returns rows where Age > 30

* **Adding/Removing Columns**:
* df['Country'] = ['USA', 'USA', 'USA'] # Adds a new column

df.drop('Country', axis=1, inplace=True) # Drops the 'Country' column

1. Data Analysis Techniques

**Data Analysis Techniques: Filtering, Sorting, and Aggregating Data (1.5 hours)**

**Filtering Data**

* **Basic Filtering**: To filter data based on conditions, you can use boolean indexing.
* # Get rows where Age is greater than 30

df\_filtered = df[df['Age'] > 30]

* **Multiple Conditions**: You can combine multiple conditions using logical operators:
* # Get rows where Age is greater than 30 and City is 'New York'

df\_filtered = df[(df['Age'] > 30) & (df['City'] == 'New York')]

* **Sorting Data**
  + **Sorting by Columns**: You can sort the DataFrame by a column in ascending or descending order:

df\_sorted = df.sort\_values(by='Age', ascending=False) # Sort by Age in descending order

**Sorting by Multiple Columns**:

df\_sorted = df.sort\_values(by=['Age', 'City'], ascending=[True, False])

**Aggregating Data**

* **Groupby**: pandas groupby method is essential for aggregating data based on one or more columns. For example:

grouped = df.groupby('City').agg({'Age': 'mean', 'Name': 'count'})

print(grouped)

* **Summary** **Statistics**: pandas provide several built-in functions to calculate summary statistics:

df['Age'].mean() # Mean of 'Age' column

df['Age'].sum() # Sum of 'Age' column

df['Age'].max() # Maximum of 'Age' column

1. **Data Visualization Basics with matplotlib**

**Introduction to matplotlib**

**matplotlib** is a comprehensive library for creating static, animated, and interactive visualizations in Python. It's often used for visualizing the results of data analysis. The most common form of visualization is through **line plots**, **scatter plots**, **bar plots**, and **histograms**.

**Basic Plotting with matplotlib**

* **Simple Line Plot**:
* import matplotlib.pyplot as plt
* # Plotting Age vs. Name
* plt.plot(df['Name'], df['Age'])
* plt.xlabel('Name')
* plt.ylabel('Age')
* plt.title('Name vs Age')

plt.show()

* **Bar Plot**:
* # Plotting a bar chart for Age by City
* df.groupby('City')['Age'].mean().plot(kind='bar')
* plt.xlabel('City')
* plt.ylabel('Average Age')
* plt.title('Average Age by City')

plt.show()

* **Histogram**:
* # Plotting a histogram for Age
* df['Age'].plot(kind='hist', bins=10)
* plt.xlabel('Age')
* plt.title('Age Distribution')

plt.show()

* **Scatter Plot**:
* # Plotting a scatter plot of Age vs. City
* df.plot(kind='scatter', x='City', y='Age')
* plt.title('City vs Age')

plt.show()

**Customizing Plots**

* **Adding Labels and Title**:
* plt.xlabel('City')
* plt.ylabel('Average Age')

plt.title('Average Age by City')

* **Color and Style**:

df['Age'].plot(kind='line', color='green', linestyle='--', linewidth=2)

**Best Practices for Working with Data and Basic Data Analysis:**

* 1. **Data Cleaning**: Before analysis, always ensure the data is clean. This includes handling missing values (df.fillna()), removing duplicates (df.drop\_duplicates()), and dealing with outliers.
  2. **Efficient Data Access**: pandas provides several ways to read large datasets efficiently. For example, use chunksize to read large CSV files in chunks.
  3. **Handling Data Types**: Ensure the data types of your columns are correct (e.g., using df['Age'] = df['Age'].astype(int)).
  4. **Documentation**: When working on analysis, make sure to document your code and the reasoning behind each transformation or computation.

**Additional Resources:**

* **pandas Documentation**: <https://pandas.pydata.org/pandas-docs/stable/>
* **matplotlib Documentation**: <https://matplotlib.org/stable/contents.html>

By completing these exercises and concepts, learners will gain a solid foundation in handling and analyzing data using pandas and **matplotlib**. These skills will be essential for performing data-driven tasks like cleaning, analyzing, and visualizing data.

4. **Introduction to Data Analysis with Pandas recording**

<https://powerlearnproject-org.zoom.us/rec/share/vRfmKMEuczLrqS1MhVjo5p43sARLHnRuyAgItjZUBFWa_lEn9lptkzbqx4gNVoNF.r-smLBIZf-x_9thj>