Introduction to Pandas

What is Pandas?

Pandas is an open-source Python library built on NumPy, providing essential tools for working with **structured data**, particularly through its powerful **tabular data structure** called DataFrame. It's built on top of the NumPy library and is particularly useful for working with structured data.

Pandas is a powerful Python library for data manipulation and analysis. It provides data structures like Series and DataFrame, along with a variety of tools for working with structured data.

Let us learn the following:

1. Installing Pandas
2. Importing Pandas
3. Series
4. DataFrames

1. **Installing Pandas:** Write the following command in in Jupyter Notebook cell to install Pandas:

```
pip install pandas
```

2. **Importing Pandas:** Once installed, you need to import Pandas into your Python script.

```
import pandas as pd
```

After successfully installing Pandas, let's delve into the process of creating Series and DataFrames to harness the capabilities of this powerful library.

3. **Series:** A Pandas Series is a one-dimensional labeled array that can hold any data type. *Series* can be created using the pd.Series() constructor.

Series is analogous to a column in a spreadsheet or a single column in a database table.

Example Code:

```
import pandas as pd
data = [1, 2, 3, 4, 5]
series = pd.Series(data)
print(series)

Output
0    1
1    2
2    3
3    4
4    5
dtype: int64
```

The code creates a Python list named data, containing a sequence of integers from 1 to 5. The **pd.Series()** constructor is used to convert this list **data** into a Pandas Series named **series**. This Pandas Series is a one-dimensional labelled array that can hold various data types.

4. **DataFrame:** A DataFrame is a two-dimensional tabular data structure with labelled axes (rows and columns). DataFrames can be created from dictionaries or other data structures using the pd.DataFrame() constructor. It is similar to a spreadsheet or an **SQL** table.

```
import pandas as pd
data = {'Name': ['John', 'Alice', 'Bob'], 'Age': [25, 28,
22]}
df = pd.DataFrame(data)
print(df)

Output
  Name  Age
0   John   25
1  Alice   28
2    Bob   22
```

Here, a Python dictionary named **data** is created. The keys ('Name' and 'Age') represent column names, and the corresponding values are lists containing data for each column. The pd.DataFrame() constructor is used to convert the Python dictionary data into a Pandas DataFrame named df.

Creating and Accessing DataFrames

Creating a DataFrame is often done by providing a dictionary where keys represent column names and values represent the data in each column.

Accessing elements in a DataFrame involves using various methods such as column indexing or **integer-location based indexing (iloc)**.

Example Code:

```
import pandas as pd
data = {'Name': ['John', 'Alice', 'Bob'], 'Age': [25, 28,
22]}
df = pd.DataFrame(data)
# Accessing a column
df['Name']

# Accessing a row
df.iloc[0]

Output
Name    John
Age       25
Name: 0, dtype: object
```

In this code- A Python dictionary named **data** is created, representing two columns - 'Name' and 'Age'. Each column is associated with a list containing corresponding data. The **pd.DataFrame()** constructor is used to convert the Python dictionary **data** into a Pandas DataFrame named **df**.

- The line df['Name'] demonstrates how to access a specific column ('Name' in this case) in the DataFrame.
- The iloc method is used to access a specific row (in this case, the row with index 0) in the DataFrame.

Data Cleaning and Preparation using Pandas

Data cleaning and preparation are crucial steps in the data analysis process. Pandas provides methods for handling missing values, such as dropping rows with missing values or filling missing values with specific values.

Example Code:

```
import pandas as pd
data = {'Name': ['John', 'Alice', 'Bob'], 'Age': [25, 28, 22]}
df = pd.DataFrame(data)
# Drop rows with missing values
df.dropna()

# Fill missing values with a specific value
df.fillna(0)
```

Output

	Name	Age
0	John	25
1	Alice	28
2	Bob	22

A Python dictionary named data is created, representing two columns - 'Name' and 'Age'. Each column is associated with a list containing corresponding data. The **pd.DataFrame()** constructor converts the Python dictionary **data** into a Pandas DataFrame named **df**.

- The **dropna()** method is called on the DataFrame **df**. This method removes any row containing at least one missing (NaN) value. However, it's important to note that the operation is not performed in-place by default, meaning the original DataFrame is not modified. If you want to modify the original DataFrame, you can use **df.dropna(inplace=True)**.
- The **fillna()** method is used to fill missing values in the DataFrame with a specific value, in this case, 0. Similar to **dropna()**, this operation is not performed in-place by default. If you want to modify the original DataFrame, you can use **df.fillna(0, inplace=True)**.

It's important to note that these operations return modified versions of the DataFrame, and if you want to keep the changes, you should assign the result back to the DataFrame or use the **inplace** parameter.