Pandas

Pandas is a powerful open-source Python library for data manipulation and analysis. It provides easy-to-use data structures, such as DataFrame, for efficient handling of structured data. Pandas is widely used in data science, finance, economics, and other fields for tasks like data cleaning, exploration, transformation, and visualization. It offers a wide range of functions and methods for data manipulation, aggregation, filtering, merging, and more, making it a valuable tool for working with tabular data in Python.

Key features of pandas

- **Data Structures**: Pandas provides powerful data structures, such as DataFrame and Series, for efficient handling and manipulation of structured data.
- **Data Manipulation**: Pandas offers a wide range of functions and methods for data cleaning, transformation, aggregation, filtering, merging, and more.
- **Integration with Other Libraries**: Pandas integrates well with other Python libraries, such as NumPy, Matplotlib, and scikit-learn, enabling seamless data analysis and visualization workflows.
- **Missing Data Handling**: Pandas provides methods for handling missing data, including filling in missing values, dropping rows or columns with missing values, and identifying missing data patterns.
- **Data Alignment**: Pandas automatically aligns data based on labels, making it easy to perform operations on data with different indexes or labels.
- **Data Visualization**: Pandas can be used in conjunction with Matplotlib to create various types of plots and visualizations for data exploration and analysis.
- **Efficiency**: Pandas is designed to be efficient and performant, with optimized algorithms and data structures for fast data processing.
- **Flexibility**: Pandas offers flexibility in data manipulation, allowing users to select, filter, and transform data in various ways to meet their specific needs.
  These features make Pandas a popular choice for data analysis and manipulation tasks in Python.

Installation

pip install pandas

Data Structures

1. `pd.DataFrame(data, index, columns)`: Creates a DataFrame from various inputs like arrays, lists, or dictionaries.
2. `pd.Series(data, index)`: Creates a Series, a one-dimensional labeled array, from various inputs like arrays, lists, or dictionaries.

Input/Output

1. `pd.read_csv(filepath_or_buffer)`: Reads a CSV file into a DataFrame.
2. `pd.read_excel(filepath_or_buffer)`: Reads an Excel file into a DataFrame.
3. `pd.read_sql(sql, con)`: Reads a SQL query or database table into a DataFrame.
4. `pd.to_csv(filepath_or_buffer)`: Writes a DataFrame to a CSV file.
5. `pd.to_excel(filepath_or_buffer)`: Writes a DataFrame to an Excel file.

Viewing/Inspecting Data

1. `df.head(n)`: Returns the first `n` rows of a DataFrame.
2. `df.tail(n)`: Returns the last `n` rows of a DataFrame.
3. `df.info()`: Provides a concise summary of a DataFrame.
4. `df.describe()`: Generates descriptive statistics of a DataFrame.

Indexing and Selecting Data

1. `df.loc[]`: Accesses a group of rows and columns by label(s) or a boolean array.
2. `df.iloc[]`: Accesses a group of rows and columns by integer position(s).
3. `df.at[]`: Accesses a single value for a row/column label pair.
4. `df.iat[]`: Accesses a single value for a row/column pair by integer position.

Data Manipulation

1. `df.drop(labels, axis)`: Drops specified labels from rows or columns.
2. `df.dropna(axis)`: Drops rows/columns with missing values.
3. `df.fillna(value)`: Fills missing values with specified values.
4. `df.rename(columns)`: Renames columns.
5. `df.sort_values(by)`: Sorts DataFrame by specified column(s).
6. `df.groupby(by)`: Groups DataFrame using a mapper or by a Series of columns.
7. `df.merge(right)`: Merges DataFrame or named Series objects with a database-style join.

Statistical Functions

1. `df.mean()`: Computes mean of each column.
2. `df.median()`: Computes median of each column.
3. `df.std()`: Computes standard deviation of each column.
4. `df.count()`: Counts non-NA/null values for each column.

Plotting

1. `df.plot()`: Plots the DataFrame.

Time Series

1. `pd.to_datetime(arg)`: Converts argument to datetime.
2. `df.resample(rule)`: Conform DataFrame to new index with optional filling logic.
3. `df.rolling(window)`: Provides rolling window calculations.

Data Cleaning and Preprocessing

1. `df.drop_duplicates(subset)`: Removes duplicate rows from the DataFrame.
2. `df.replace(to_replace, value)`: Replaces values in the DataFrame.
3. `df.astype(dtype)`: Converts the data types of columns in the DataFrame.
4. `df.dropna(axis, thresh)`: Drops rows/columns with a certain number of missing values.

Combining and Concatenating DataFrames

1. `pd.concat(objs, axis)`: Concatenates pandas objects along a particular axis.
2. `df.append(other)`: Appends rows of other DataFrame to the end of the caller DataFrame.

Reshaping and Pivoting

1. `df.pivot_table(values, index, columns)`: Creates a pivot table from DataFrame columns.
2. `pd.melt(frame, id_vars, value_vars)`: Unpivots a DataFrame from wide format to long format.

String Operations

1. `df.str.upper()`: Converts strings in the DataFrame to uppercase.
2. `df.str.lower()`: Converts strings in the DataFrame to lowercase.
3. `df.str.contains(pat)`: Checks whether each string contains a substring.

Categorical Data

1. `pd.Categorical(values)`: Constructs a categorical variable.
2. `df.astype('category')`: Converts columns to categorical data type.

Handling Time Series Data

1. `pd.date_range(start, end, freq)`: Generates a sequence of fixed-frequency dates.
2. `df.shift(periods)`: Shifts index by desired number of periods.
3. `df.diff(periods)`: Computes difference of DataFrame elements over periods.

Missing Data Handling

1. `df.interpolate(method)`: Interpolates missing values in the DataFrame.

Statistical Operations

1. `df.corr()`: Computes pairwise correlation of columns.
2. `df.cov()`: Computes covariance matrix of columns.
3. `df.quantile(q)`: Computes sample quantiles of columns.

Aggregation and Grouping

1. `df.aggregate(func)`: Aggregate using one or more operations over the specified axis.
2. `df.transform(func)`: Call function producing a like-indexed DataFrame.

Time Zone Handling

1. `df.tz_localize(tz)`: Localizes time zone of DataFrame.
2. `df.tz_convert(tz)`: Converts index to new time zone.