1. **Loading Data**:
   * Reading data from various sources like CSV, Excel, SQL databases, JSON, etc., using functions like **pd.read\_csv()**, **pd.read\_excel()**, **pd.read\_sql()**, etc.
2. **Exploring Data**:
   * Viewing the first few rows of the DataFrame using **df.head()**.
   * Viewing the last few rows of the DataFrame using **df.tail()**.
   * Displaying the summary information of the DataFrame using **df.info()** and **df.describe()**.
3. **Selecting Data**:
   * Accessing specific columns using square brackets (**df['column\_name']**) or dot notation (**df.column\_name**).
   * Accessing specific rows and columns using **df.iloc[]** (integer-location based indexing) or **df.loc[]** (label-based indexing).
   * Filtering rows based on conditions using boolean indexing (**df[df['column'] > value]**).
4. **Cleaning Data**:
   * Handling missing values using methods like **df.dropna()**, **df.fillna()**, or **df.interpolate()**.
   * Handling duplicate rows using **df.drop\_duplicates()**.
   * Converting data types using **df.astype()** or **pd.to\_datetime()**.
5. **Manipulating Data**:
   * Applying functions to columns or rows using **df.apply()** or **df.applymap()**.
   * Grouping data using **df.groupby()** and performing aggregation operations.
   * Merging or concatenating DataFrames using **pd.merge()** or **pd.concat()**.
6. **Visualization**:
   * Visualizing data using built-in plotting functions like **df.plot()** for simple plots.
   * Using libraries like Matplotlib or Seaborn for more advanced visualizations.
7. **Statistical Analysis**:
   * Computing descriptive statistics using methods like **df.mean()**, **df.median()**, **df.std()**, etc.
   * Calculating correlations between variables using **df.corr()**.
8. **Data Transformation**:
   * Reshaping data using methods like **df.pivot\_table()**, **df.stack()**, **df.unstack()**.
   * Applying string operations to text data using **str** accessor, e.g., **df['column'].str.contains()**, **df['column'].str.lower()**, etc.
9. **Parentheses ()**:
   * Parentheses are primarily used to call functions or methods in Pandas.
   * They are used to enclose the arguments passed to a function or method.

df.head() df.groupby('column\_name').mean()

1. **Square Brackets []**:
   * Square brackets are used for indexing and selecting data in Pandas.
   * They can be used to access columns, rows, or specific elements of a DataFrame or Series.
   * They can also be used for boolean indexing, where you pass a condition inside square brackets to filter rows based on that condition.
   * For example:
     + Accessing a column: **df['column\_name']**
     + Accessing multiple columns: **df[['column1', 'column2']]**
     + Accessing rows by index: **df.iloc[0]**
     + Accessing rows by label: **df.loc['label']**
     + Boolean indexing: **df[df['column'] > value]**

In summary, parentheses **()** are primarily used to call functions or methods, while square brackets **[]** are used for indexing, selecting data, and boolean indexing in Pandas.

# Remove column name 'A'

df.drop(['A'], axis**=**1)

# Remove two columns name is 'C' and 'D'

df.drop(['C', 'D'], axis**=**1)

# Remove three columns as index base

df.drop(df.columns[[0, 4, 2]], axis**=**1, inplace**=**True)

# Rename column 'A' to 'X'

df.rename(columns={'A': 'X'}, inplace=True)

# Rename the index

df = df.rename(index={0: 'row1', 1: 'row2', 2: 'row3'}) or df.index = ['X', 'Y', 'Z', 'W']

if entire df has int type . then if we want to see the element that has > 0 else NAN

a = df[df>0]

a

# Adding a new column with values ['Engg','UDC','LDC']

df['desig'] = ['Engg','UDC','LDC']

df.columns = show all the col if the col. Name has not contain any space.

df[‘col1’].max()

to select specific columns:

df[[ 'GP Name','Activity Name','Estimated Cost' ]]

df.iloc[:,[3,4,7,10,12]]

to change the datas of a existing column:

df['Address'] = ['malda','kolkata','bongaon']

df

or

from numpy.random import rand

df['Column4'] = pd.Series(rand(3),["A","B","C"])

# Add a new row

df.loc[3] = [13, 14, 15, 16]

This code uses **pd.set\_option()** to set the maximum number of rows and columns displayed in the DataFrame to **None**, which means unlimited

pd.set\_option('display.max\_rows', None)

pd.set\_option('display.max\_columns', None)

print(df)

to retrieve specific col. By index

column\_indices = [0, 2] # Example: indices of columns you want to retrieve

specific\_columns = df.iloc[:, column\_indices]

* **df.iloc[:, column\_indices]** selects all rows (**:**) and the columns specified by their index positions (**column\_indices**).

rows\_list = df.index.tolist()

columns\_list = df.columns.tolist()