FREE EDITION · NOTES + 3 SAMPLE MCQS

CUET · COMPUTER SCIENCE · CLASS XII · CODE 308

Data Handling Using Pandas - I

CUET unit: Data Handling Using Pandas - I

By **UniDrill** · NCERT-grounded study material

WWW.UNIDRILL.IN

📌 Snapshot

- The Pandas library is a high-level data manipulation tool built on NumPy and Matplotlib, with two core data structures: Series and DataFrame.
- Python-based data handling is a central skill in Informatics Practices; CUET questions probe creation, indexing, slicing, operations, and CSV import/export, so it is tested heavily.
- Series is covered in depth (creation from scalars, NumPy arrays, and dictionaries; indexing and slicing; attributes and methods; mathematical operations), then DataFrame (creation from multiple sources; row/column operations; label-based and Boolean indexing; joining; attributes).
- Section 2.4 covers practical data exchange via CSV files using `read_csv()` and `to_csv()`, which CUET links to real-world data analysis tasks.
- Section 2.5 contrasts Pandas Series with NumPy ndarray — a favourite NTA trap area requiring careful distinction of behaviours around index alignment, NaN handling, and memory.

📖 Detailed Notes

2.1 Core concepts

- **Python libraries (§2.1, p. 27–28):** NumPy, Pandas, and Matplotlib are three well-established Python libraries for scientific and analytical use. NumPy (Numerical Python) supports multidimensional arrays and numeric computing; Matplotlib is used for plotting graphs and visualisation; Pandas (PANel DAta) is a high-level data manipulation tool that is easy to use for import and export of data, and provides a rich set of functions. Pandas is built on top of NumPy and Matplotlib. (NCERT §2.1, p. 27–28)
- **Installing Pandas (§2.1.1, p. 28):** Pandas is installed from the command line using `pip install pandas`. NumPy and Pandas can be installed only when Python is already installed on the system. Pandas is conventionally imported with the alias `pd`: `import pandas as pd`. (NCERT §2.1.1, p. 28)
- **Data structures in Pandas (§2.1.2, p. 29):** A data structure is a collection of data values and the operations that can be applied to that data, enabling efficient storage,

retrieval, and modification. The two commonly used Pandas data structures are Series and DataFrame. (NCERT §2.1.2, p. 29)

- **Pandas vs NumPy key differences (§2.1, p. 28):** (1) A NumPy array requires homogeneous data, while a Pandas DataFrame can have different data types per column. (2) Pandas has a simpler interface for file loading, plotting, selection, joining, and GROUP BY. (3) Pandas DataFrames use named column labels, making it easy to track data. (4) Pandas is suited for tabular data; NumPy for numeric array-based computation. (NCERT §2.1, p. 28)
- **Series — definition (§2.2, p. 29):** A Series is a one-dimensional array containing a sequence of values of any data type (int, float, list, string, etc.) which by default have numeric data labels (index) starting from zero. The data label associated with a particular value is called its index. A Series can be imagined as a column in a spreadsheet. (NCERT §2.2, p. 29)
- **Creation of Series (§2.2.1, p. 29–31):** A Series can be created from: (A) scalar/list values using `pd.Series([10,20,30])`; (B) a one-dimensional NumPy ndarray using `pd.Series(array1)`; (C) a Python dictionary using `pd.Series(dict1)` — dictionary keys become the index. When creating a Series with an explicit `index` parameter, the length of the index list must match the length of the data; a mismatch raises a `ValueError`. (NCERT §2.2.1, p. 29–31)
- **Accessing elements of a Series — Indexing (§2.2.2, p. 31–32):** Two types of indexing: (i) Positional index — integer position starting from 0; (ii) Labelled index — any user-defined label as index. More than one element can be accessed using a list of positions or labels. Index values of a Series can be altered by assigning new index values (e.g., `series.index = [10, 20, 30, 40]`). (NCERT §2.2.2, p. 31–32)
- **Accessing elements of a Series — Slicing (§2.2.2, p. 33–34):** Slicing extracts a part of a series using `[start:end]`. When positional indices are used, the value at the end index position is **excluded**. When labelled indices are used for slicing, the value at the end index label is **included**. Slicing can also be used to modify values. (NCERT §2.2.2, p. 33–34)
- **Attributes of Series (§2.2.3, p. 34–35):** Key attributes of a Pandas Series include: `name` (assigns a name to the Series), `index.name` (assigns a name to the index), `values` (returns list of values), `size` (number of values), `empty` (True if the series is empty, False otherwise). (NCERT §2.2.3, p. 34–35, Table 2.1)
- **Methods of Series (§2.2.4, p. 35–36):** Key methods include: `head(n)` — returns first n members (default 5); `tail(n)` — returns last n members (default 5); `count()` — returns the number of non-NaN values. (NCERT §2.2.4, p. 35–36)
- **Mathematical operations on Series (§2.2.5, p. 36–39):** Basic operations (+, -, *, /) can be done using operators or explicit methods (`add()`, `sub()`, `mul()`, `div()`). Index matching is implemented: if indices do not match, the result for that position is NaN. The explicit methods accept a `fill_value` parameter to replace

missing values with a specified number before the operation, avoiding NaN in the result. (NCERT §2.2.5, p. 36–39)

- **DataFrame — definition (§2.3, p. 40):** A DataFrame is a two-dimensional labelled data structure like a table of MySQL. It contains rows and columns and therefore has both a row index and a column index. Each column can have a different data type (numeric, string, boolean, etc.). (NCERT §2.3, p. 40)
- **Creation of DataFrame (§2.3.1, p. 40–44):** A DataFrame can be created from: (A) an empty DataFrame: `pd.DataFrame()` ; (B) NumPy ndarrays — a single ndarray or multiple ndarrays with `columns` parameter; (C) a list of dictionaries — dictionary keys become column labels, each dict is a row; (D) a dictionary of lists — keys become column labels, lists become rows; (E) one or more Series objects — Series labels become column names, each Series becomes a row; (F) a dictionary of Series — dictionary keys become column labels, and the resulting index is the union of all series indexes. NaN is inserted where a value is missing. (NCERT §2.3.1, p. 40–44)
- **Operations on rows and columns in DataFrames (§2.3.2, p. 44–48):** (A) Adding a new column: assign a list to `df['NewColumn']` ; if the column already exists it is updated. (B) Adding a new row: use `df.loc['label'] = [values]` ; if the label already exists the row is updated. Adding a row or column with mismatched length raises a `ValueError` . (C) Deleting rows or columns: use `df.drop('label', axis=0)` for rows (axis=0) and `df.drop('label', axis=1)` for columns; if multiple rows share the same label, all are deleted. (D) Renaming row labels: use `df.rename({'old':'new'}, axis='index')` ; (E) Renaming column labels: use `df.rename({'old':'new'}, axis='columns')` . (NCERT §2.3.2, p. 44–48)
- **Accessing DataFrame elements through indexing (§2.3.3, p. 49–51):** Two types: (A) Label-based indexing using `DataFrame.loc[]` — a single row label returns the row as a Series; a single column label using `df.loc[:,'col']` or `df['col']` returns the column as a Series; a list of row labels using `[[]]` returns a DataFrame; (B) Boolean indexing — conditions on column values return a Series of True/False; used to filter data. (NCERT §2.3.3, p. 49–51)
- **Accessing DataFrame elements through slicing (§2.3.4, p. 50–51):** Slicing with `df.loc['start':'end']` retrieves rows inclusively (label-based slicing in DataFrames is inclusive of the end label). Can combine a slice of row labels with a list of column names or a slice of column names. Boolean lists can be used with `loc[]` to filter rows. (NCERT §2.3.4, p. 50–51)
- **Joining, merging and concatenation of DataFrames (§2.3.5, p. 52–53):** The `DataFrame.append()` method appends rows of the second DataFrame to the end of the first. Columns not present in the first DataFrame are added as new columns with NaN for existing rows. Parameters: `sort` (True/False for sorting column labels), `verify_integrity` (True raises error if duplicate row labels exist; default False), `ignore_index` (True ignores row index labels; default False). (NCERT §2.3.5, p. 52–53)

- **Attributes of DataFrames (§2.3.6, p. 53–55):** Key attributes include: `index` (row labels), `columns` (column labels), `dtypes` (data type of each column), `values` (NumPy ndarray of all values), `shape` (tuple of rows × columns), `size` (total number of values), `T` (transpose — rows and columns swap positions), `head(n)` (first n rows, default 5), `tail(n)` (last n rows, default 5), `empty` (True if DataFrame is empty). (NCERT §2.3.6, p. 53–55, Table 2.4)
- **Importing a CSV file to a DataFrame (§2.4.1, p. 55–56):** Use `pd.read_csv(filepath, sep=',', header=0)`. The `sep` parameter specifies the separator character (default is space, but comma is standard for CSV). The `header` parameter specifies which row to use as column names (default header=0 means first line). The `names` parameter allows explicit specification of column labels. (NCERT §2.4.1, p. 55–56)
- **Exporting a DataFrame to a CSV file (§2.4.2, p. 56–57):** Use `df.to_csv('filepath', sep=',')`. The `header=False` parameter omits column names; `index=False` omits row labels from the file. (NCERT §2.4.2, p. 56–57)
- **Pandas Series vs NumPy ndarray (§2.5, p. 57–58):** Key differences (Table 2.5): Pandas Series supports user-defined labelled index (numbers or letters); NumPy arrays are accessed only by integer position. Series elements can be indexed in descending order; NumPy index is fixed starting from 0. If two Series are not aligned, NaN values are generated; NumPy has no concept of NaN for misaligned arrays (alignment fails). Series require more memory than NumPy arrays. Operations between Series automatically align data based on labels, enabling computation without checking whether Series share the same labels. (NCERT §2.5, p. 57–58)

2.2 Definitions to memorise

Term	Definition	Page
Pandas	PANel DAta — a high-level Python data manipulation library built on NumPy and Matplotlib, used for analysing, transforming, and visualising data	28
Series	A one-dimensional labelled array in Pandas that can contain values of any data type; each value has a data label called its index	29
DataFrame	A two-dimensional labelled data structure in Pandas with both row and column indexes; analogous to a spreadsheet or SQL table	40
Positional index	An integer index starting from 0 corresponding to the position of an element in a Series or DataFrame	31
Labelled index	A user-defined label (number or string) assigned as the index of a Series element	31
NaN		37

Term	Definition	Page
	Not a Number — the value Pandas inserts when an index is present in one Series but missing in the other during mathematical operations	
fill_value	A parameter used in explicit Pandas methods (add, sub, mul, div) to replace missing values with a specified number before the operation	37
head(n)	Series/DataFrame method that returns the first n members (default n=5)	36
tail(n)	Series/DataFrame method that returns the last n members (default n=5)	36
count()	Series method that returns the number of non-NaN values in the Series	36
DataFrame.loc[]	Pandas method used for label-based indexing of rows and columns in a DataFrame	45
Boolean indexing	A method of selecting subsets of data in a DataFrame based on conditions that evaluate to True or False	50
read_csv()	Pandas function to load data from a CSV file into a DataFrame	56
to_csv()	Pandas DataFrame method to save data from a DataFrame to a CSV (text) file	56
DataFrame.drop()	Method used to delete rows (axis=0) or columns (axis=1) from a DataFrame by specifying label names	47
DataFrame.rename()	Method used to change row index labels (axis='index') or column labels (axis='columns') in a DataFrame	48
DataFrame.append()	Method used to merge two DataFrames by appending rows of the second DataFrame to the first	52
DataFrame.T	Attribute that returns the transpose of a DataFrame (rows and columns swap positions)	55
pip install pandas	Command-line instruction to install the Pandas library	28
CSV file	A Comma-Separated Value text file where each line is a record and fields within a record are separated by commas	57
`pd.Series(data)`	Constructor that creates a Series from a list, ndarray, scalar, or dict	29-31
`pd.DataFrame(data)`	Constructor that creates a DataFrame from various sources	40-44
`Series.values`	Attribute that returns the underlying ndarray of Series values	35
`Series.size`	Attribute returning the number of values in the Series	35
`Series.empty`	Attribute returning True if the Series has zero elements	35

Term	Definition	Page
`DataFrame.shape`	Tuple `(rows, cols)` describing DataFrame dimensions	54
`DataFrame.size`	Total number of values = rows × cols	54
`DataFrame.columns`	Index object containing the column labels	53
`DataFrame.index`	Index object containing the row labels	53
`DataFrame.dtypes`	Series listing the data type of each column	53
`iloc[]`	Integer-position-based indexer	49
Alignment	Pandas behaviour of matching elements by index label before operations	37

2.3 Diagrams / processes to remember

- **Table 2.1 — Attributes of Pandas Series** (p. 35): A tabular reference showing `name`, `index.name`, `values`, `size`, and `empty` with purpose and code examples. Flip to p. 35 to review the exact syntax for each attribute.
- **Table 2.2 / 2.3 — Addition of two Series with and without fill_value** (p. 37–38): These two tables visually demonstrate how index matching works and how NaN arises when indices do not overlap; also shows the effect of `fill_value=0` replacing NaN. Critical for understanding mathematical operations questions.
- **Table 2.4 — Attributes of Pandas DataFrame** (p. 54–55): Covers `index`, `columns`, `dtypes`, `values`, `shape`, `size`, `T`, `head(n)`, `tail(n)`, `empty` with examples using a ForestAreaDF. Refer to p. 54–55 for code examples.
- **Table 2.5 — Difference between Pandas Series and NumPy Arrays** (p. 58): A four-row comparison covering index types, descending order access, NaN handling, and memory usage. Flip to p. 58 for the exact wording used by NTA distractors.
- **Figure: DataFrame structure** (p. 40): The diagram shows Row Indexes on the left axis and Column Indexes across the top, illustrating State, Geographical Area, and Area under Very Dense Forests. Helps visualise two-axis labelling.

2.4 Common confusions / NTA trap points

- **Slicing end-index inclusion rule:** With positional indices, slicing a Series or DataFrame **excludes** the end index; with labelled indices, slicing **includes** the end label. NTA frequently presents code and asks for the correct output, exploiting this asymmetry (NCERT §2.2.2, p. 33).
- **NaN vs fill_value:** When two Series are added using `+`, unmatched indices produce NaN. Using `seriesA.add(seriesB, fill_value=0)` replaces missing values with 0 **before** addition, so no NaN appears in the result. Confusing the two methods is a classic NTA trap (NCERT §2.2.5, p. 37–38).

- **drop() axis parameter:** `axis=0` deletes a **row**, `axis=1` deletes a **column**. Students often swap these. If a label appears multiple times (duplicate rows), `drop()` removes **all** matching rows (NCERT §2.3.2, p. 47).
- **Creating a DataFrame from Series:** When multiple Series with different index labels are combined, columns equal the union of all indexes and missing values become NaN. When creating a DataFrame from a dictionary of Series, the **column** names are the dictionary keys and the **row** labels are the union of Series indexes — the reverse of what students intuitively expect (NCERT §2.3.1, p. 44).
- **read_csv() default parameters (NCERT §2.4.1, p. 56).** The default value of `header` is 0 (first row used as column names), and the default value of `sep` is a comma per the standard library default; NCERT's exposition explicitly recommends specifying `sep=','` for standard CSV files.
- **Positional vs labelled slicing (NCERT §2.2.2, p. 33-34).** Positional excludes the end index; labelled includes it. Critical for output-prediction questions.
- **DataFrame from dict of Series (NCERT §2.3.1, p. 44).** Keys become column labels; union of indexes becomes row labels.
- `head()` / `tail()` **defaults (NCERT §2.2.4, p. 36).** Default n=5; pass n explicitly to override.
- `drop()` **axis parameter (NCERT §2.3.2, p. 47).** axis=0 → row; axis=1 → column.
- `to_csv()` **index=False (NCERT §2.4.2, p. 56-57).** Without this, the saved CSV has an extra column for the DataFrame's row labels.
- `name` **attribute applies to both Series and its index (NCERT Table 2.1, p. 35).** Two separate name properties: `Series.name` and `Series.index.name`.

🎯 Practice MCQs

Q1. Which of the following statements correctly describes a Pandas Series?

A. It is a two-dimensional labelled data structure with both row and column indexes.

B. It is a one-dimensional array that can only hold integer values.

C. It is a one-dimensional labelled array that can hold values of any data type, with each value associated with a data label called its index.

D. It is a Python dictionary where keys must be strings and values must be numeric.

Q2. Consider the following code: ```python import pandas as pd seriesA = pd.Series([10, 20, 30], index=['x', 'y', 'z']) seriesB = pd.Series([1, 2, 3], index=['y', 'z', 'w']) result = seriesA + seriesB ``` How many NaN values will be present in `result`?

A. 0

B. 1

C. 2

D. 3

Q3. Which of the following is the correct command to install the Pandas library from the command line?

A. `install pandas`

B. `python install pandas`

C. `pip install pandas`

D. `import pandas`

📊 PYQ Alignment

Pandas appears with moderate-to-high frequency in CUET Computer Science (subject code 308), typically contributing 2–4 questions per year on Pandas data structures, with questions focused on Series creation and indexing, DataFrame operations (add/delete/rename rows and columns), and import/export of CSV files; assertion-reason and output-based questions on the slicing inclusion/exclusion rule and NaN behaviour in mathematical operations are recurring favourites. See PYQ archive for Computer Science.