Ready For Boards
10th & 12th Exam Prep

CHAPTER 1

Data Handling Using Pandas – I

CBSE Class 12 Informatics Practices · Unit 1

CBSE · Informatics Practices · Class 12

WHAT THIS CHAPTER DOES

A Memorise the import convention (`import pandas as pd`) and the three Series-construction forms.

B Create a DataFrame from a dict of Series and predict its output.

Boards prep that builds confidence, not anxiety.

TODAY'S MISSION

Today's mission

1. Memorise the import convention (`import pandas as pd`) and the three Series-construction forms.
2. Create a DataFrame from a dict of Series and predict its output.
3. Use Boolean indexing to filter rows on a condition.
4. Read a CSV with read_csv and write a filtered subset with to_csv(index=False).

WHY THIS MATTERS

Why this chapter matters

1. Pandas alone accounts for ~1/3 of the IP theory paper. Mastery here = ~25 of 70 marks.
2. Pandas is the de-facto data-analysis library in industry — IT, finance, analytics, research.
3. Every later chapter (matplotlib, MySQL connectivity) consumes DataFrames produced here.

TOPIC

A

Series — the 1-D labelled array

THEOREM · LOAD-BEARING RESULT

What a Pandas Series is

> *A Pandas SERIES is a one-dimensional labelled array — it holds a sequence of values with a parallel sequence of index LABELS, so each value can be addressed by its label as well as by position.*

STATEMENT

Constructed as pd.Series(data, index=[...], dtype=...). The INDEX is a separate object (a pd.Index instance) parallel to the values; default index is 0, 1, 2, ... when not specified. Operations: arithmetic (s + 5, s * 2, s1 + s2 with index

WHY THIS MATTERS

- Series is the building block of DataFrame — every column of a DataFrame is itself a Series
- Understanding Series perfectly = understanding half of Pandas
- Every board paper has at least one Series-creation or Series-output question.

WATCH OUT FOR

NOTE Do NOT confuse INDEX (the labels, left column of output) with VALUES (the data, right column). In s = pd.Series([10,20,30], index=['a','b','c']), the index is ['a','b','c'] and the values are [10,20,30]. Examiners mark this distinction strictly.

TOPIC

Three ways to create a Series

FROM A PYTHON LIST

pd.Series([10, 20, 30]) creates a 3-element Series with default INTEGER index 0, 1, 2. The values 10, 20, 30 are stored as int64 by default (Pandas auto-detects dtype). If you want custom labels, pass index=['a','b','c']. From a list is the simplest construction form and the one most students see first.

FROM A PYTHON DICTIONARY

pd.Series({'Maths': 85, 'Science': 78}) creates a Series where the dictionary KEYS become the INDEX and the dictionary VALUES become the Series VALUES. Order is preserved (since Python 3.7 dicts are ordered). This is the most COMMON construction form in real Pandas code because dict naturally pairs labels with values

FROM A NUMPY NDARRAY

pd.Series(np.array([1,2,3])) creates a Series from a NumPy array. Useful when the data is already in NumPy form (from a computation, sensor reading, image pixel array). The dtype is preserved from the ndarray. Without an explicit index, default integer index 0, 1, 2, ... is used as with the list constructor

FROM A SCALAR

pd.Series(7, index=['a','b','c']) creates a Series where the single value 7 is BROADCAST across all three labels — final Series is a=7, b=7, c=7. Requires an explicit index (otherwise Pandas cannot infer the length). Useful for initialising a Series with a default value before populating cells one by one.

TOPIC

B

DataFrame — the 2-D labelled table

THEOREM · LOAD-BEARING RESULT

What a DataFrame is

> *A Pandas DATAFRAME is a two-dimensional labelled tabular data structure — a collection of columns where each column is a Series, all sharing a common row index. It is the closest Python analogue to a SQL table or a spreadsheet.*

STATEMENT

Constructed in several forms: from a dict where keys are column names and values are lists or Series (most common); from a list of dicts where each dict is a row; from a NumPy 2-D array (with index and columns specified);

WHY THIS MATTERS

- DataFrame is the workhorse of all data analysis
- The board paper allots roughly 70% of Pandas marks to DataFrame operations (vs ~30% for Series alone)
- Mastering DataFrame is the highest-leverage hour of the IP course.

WATCH OUT FOR

NOTE Most DataFrame operations RETURN A NEW DataFrame and do NOT modify the original. df.drop('col', axis=1) gives you a new DataFrame without the column; the original df is unchanged. To modify in place, use inplace=True or reassign df = df.drop(...).

TOPIC

DataFrame operations you must memorise

SELECT A COLUMN

df['Marks'] returns the 'Marks' column as a Series. Equivalent dot-form: df.Marks (works only when the column name is a valid Python identifier — no spaces, no special chars). To select multiple columns, pass a LIST: df[['Marks','Name']] returns a 2-column DataFrame (note the

ADD A COLUMN

df['Total'] = df['Maths'] + df['Science'] — assignment creates the new column. Arithmetic is VECTORISED (no loop needed). Pandas adds element-wise across rows; the new column inherits the same index. To add a constant column: df['Status'] = 'Active' broadcasts the string to every row. New

DELETE + RENAME

DELETE: df.drop('Total', axis=1) returns a new DataFrame WITHOUT the 'Total' column. axis=1 means 'column'; axis=0 (default) drops rows. Use inplace=True to modify the original. RENAME: df.rename(columns={'Maths':'Math Score'}) returns a new DataFrame with the column renamed. To rename rows pass index={...} instead of columns={...}.

BOOLEAN INDEXING

df[df['Marks'] > 80] returns ONLY the rows where Marks > 80. The inner expression df['Marks'] > 80 returns a BOOLEAN Series; passing it inside df[ ] selects the True rows. This is the most powerful Pandas pattern — replaces what would be a for-loop with an if-condition in plain Python

WORKED EXAMPLE

End-to-end workflow — read CSV, filter, write

1. PROBLEM: Read students.csv into a DataFrame, show first 5 rows, filter those with Marks > 80, save the result to toppers.csv.
2. STEP 1 — Import + read: import pandas as pd; df = pd.read_csv('students.csv'). read_csv auto-detects column types and uses the first row as the header by default. If your CSV uses a different separator pass sep=';' (or whatever).
3. STEP 2 — Show first 5 rows: print(df.head()). The default count is 5; pass head(3) for a custom number. Equivalent reverse: df.tail() for last 5.
4. STEP 3 — Filter: toppers = df[df['Marks'] > 80]. The condition produces a Boolean Series; passing it to df[] selects matching rows. Always use parentheses if combining conditions with & or |.
5. STEP 4 — Save: toppers.to_csv('toppers.csv', index=False). index=False prevents Pandas from writing its integer row labels as a separate column in the CSV — produces a clean file matching the input structure.

TOPIC

Series index vs values

TRAP → TRUTH

× **MISTAKE** A Pandas Series is just a list of numbers.

✓ **CORRECT** A Pandas Series has TWO components: a sequence of VALUES (the data) and a parallel sequence of INDEX LABELS (the row identifiers). When you create Series([10, 20, 30]) the values are 10, 20, 30 and the default INDEX is 0, 1, 2 — Pandas auto-generates integer labels. You can explicitly pass index=['a','b','c'] to get string labels. The index is what makes Series different from a list — it lets you access values by LABEL, not just by position. Confusing index with values is the single biggest source of board paper errors.

TOPIC

.loc vs .iloc

TRAP → TRUTH

× MISTAKE .loc and .iloc are interchangeable.

✓ CORRECT They are NOT interchangeable. .loc uses LABEL-based indexing — df.loc['Alice'] looks up the row whose index label is 'Alice'. .iloc uses INTEGER POSITION — df.iloc[0] returns the first row regardless of its label. Using the wrong one gives wrong rows or KeyError. Rule: if you wrote out the label as a string, use .loc; if you are counting positions (0,1,2,...), use .iloc.

TOPIC

Slicing endpoints — Series vs lists

TRAP → TRUTH

✗ **MISTAKE** Pandas slicing follows Python list rules — endpoint excluded.

✓ **CORRECT** Pandas slicing with LABEL-based .loc INCLUDES the endpoint: s.loc['a':'c'] returns rows with labels 'a', 'b', AND 'c'. Position-based .iloc EXCLUDES the endpoint (like Python lists): s.iloc[0:3] returns positions 0, 1, 2 (NOT 3). This inconsistency confuses students; the rule is: labels are inclusive, positions are exclusive. Examiners deliberately mix these to test whether the student noticed.

TOPIC

Default behaviour of df.head() and df.tail()

TRAP → TRUTH

× **MISTAKE** df.head() returns 1 row.

✓ **CORRECT** Both df.head() and df.tail() return the FIRST or LAST 5 rows by default — NOT 1 row. You can pass an integer argument: df.head(3) returns the first 3, df.tail(10) returns the last 10. The default 5 was chosen because it usually fits on a screen without scrolling. Mixing this up costs a half-mark on every output-prediction question involving .head() or .tail().

TOPIC

Modifying a DataFrame in place vs producing a new one

TRAP → TRUTH

✗ **MISTAKE** All DataFrame operations modify the DataFrame in place.

✓ **CORRECT** Most DataFrame operations RETURN A NEW DataFrame and do NOT modify the original. df.drop('A', axis=1) returns a new DataFrame without column A — but the original df is unchanged. To modify in place: use inplace=True (df.drop('A', axis=1, inplace=True)) OR reassign (df = df.drop('A', axis=1)). Beginners who do df.drop(...) and then expect df to have changed are confused when it has not. Always check the documentation for inplace=True availability.

TOPPER TEMPLATE · MARK-BY-MARK

3 marks: Create a Pandas Series from a Python dictionary {'Maths': 85, 'Science': 78, 'English': 92}. Display the output

1	**IMPORT + CREATE** 1 m	import pandas as pd marks_dict = {'Maths': 85, 'Science': 78, 'English': 92} s = pd.Series(marks_dict) print(s) The import is mandatory; without it the code will not run.
2	**PREDICT THE OUTPUT** 1 m	Output: Maths 85 Science 78 English 92 dtype: int64 The dictionary KEYS became the Series INDEX (left column); the dictionary VALUES became the Series VALUES (right column). dtype: int64 is auto-detected from the integer values.
3	**EXPLAIN THE INDEX** 1 m	When a Series is created from a dictionary, Pandas automatically uses the dictionary keys as the INDEX of the Series — preserving the natural label–value pairing. This is one of three ways to create a Series; the others are from a Python list (with integer-default index 0,1,2...) and from a NumPy ndarray.

TOPPER TEMPLATE · MARK-BY-MARK

4 marks: Create a DataFrame from a dictionary of Series for three students Alice/Bob/Carol with Maths and Science

1	**IMPORT + CREATE THE TWO SERIES** 1 m	import pandas as pd maths = pd.Series([85, 90, 72], index=['Alice','Bob','Carol']) science = pd.Series([78, 88, 95], index=['Alice','Bob','Carol'])
2	**CREATE THE DATAFRAME** 1.5 m	df = pd.DataFrame({'Maths': maths, 'Science': science}) print(df) Output: Maths Science Alice 85 78 Bob 90 88 Carol 72 95 The dictionary KEYS became the COLUMN names; the Series INDICES aligned to create the row index.
3	**ADD THE TOTAL COLUMN** 1.5 m	df['Total'] = df['Maths'] + df['Science'] print(df) Output: Maths Science Total Alice 85 78 163 Bob 90 88 178 Carol 72 95 167 Vectorised addition — Pandas adds element-wise without any explicit loop.

TOPPER TEMPLATE · MARK-BY-MARK

5 marks: Read a CSV file 'students.csv' into a DataFrame, display the first 5 rows, filter rows where Marks > 80, and

1	**IMPORT + READ CSV**	1 m	import pandas as pd df = pd.read_csv('students.csv') The read_csv() function auto-detects column types and uses the first row as the column header by default.
2	**DISPLAY FIRST 5 ROWS**	1 m	print(df.head()) No argument needed — head() defaults to 5 rows. Alternative: print(df.head(5)) makes the count explicit. Both produce identical output.
3	**BOOLEAN FILTER**	1.5 m	toppers = df[df['Marks'] > 80] print(toppers) This is BOOLEAN INDEXING — the inner expression df['Marks'] > 80 produces a Boolean Series; passing that to df[ ] selects the rows where True. No explicit loop needed.
4	**WRITE TO NEW CSV**	1.5 m	toppers.to_csv('toppers.csv', index=False) The index=False suppresses Pandas' integer row labels from being written as a column. With index=False the resulting CSV has the same column structure as the input — clean for downstream consumption.

PYQ PATTERNS

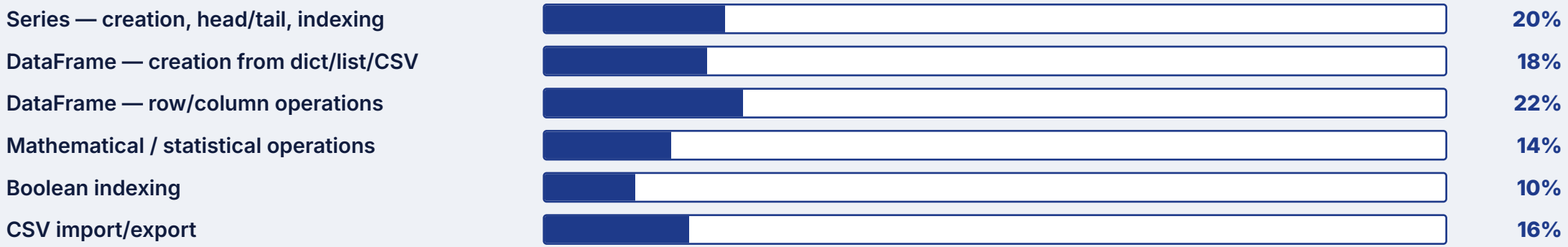
Top PYQ patterns to drill

#1	Create a Pandas Series from a Python dictionary and predict its output. (3 marks)	Annual
#2	Create a DataFrame from a dict of Series; perform .head(2) / .tail(2); predict output. (3 marks)	Annual
#3	Add a new column, delete an existing one, rename an index	give the Pandas code. (4 marks) — Annual
#4	Use Boolean indexing to filter rows of a DataFrame on a condition (e.g. Marks > 80). (3 marks)	Most years
#5	Read a CSV into a DataFrame, display first 5 rows, export to a new CSV. (5 marks)	2021, 2023

MARKS DISTRIBUTION

10-year marks distribution

10-YEAR PYQ MARKS DISTRIBUTION



RECAP · MEMORISE THESE

Recap — what you must know cold

1 Import — Always begin with `import pandas as pd`. The `pd` alias is universal Pandas convention.

2 Series — 1-D labelled array. Construct from list, dict, ndarray, or scalar+index. Has INDEX (labels) + VALUES separately.

3 DataFrame — 2-D labelled table. Each column is a Series. Construct from dict of Series, list of dicts, ndarray + index/ columns, or read_csv.

4 Access — df['col'] for a column. df.loc['label'] for a row by label. df.iloc[0] for a row by position.

5 Mutating columns — df['new'] = expr (add). df.drop('col', axis=1) (delete).

6 Boolean indexing — df[df['col'] > value] returns rows matching the condition. Combine

WHAT'S NEXT

What's next

→ Chapter 2 — Data Handling Using Pandas – II (aggregation, groupby, joining/merging).

→ Sit the 15-MCQ Quick Drill for this chapter.

→ Then the full Board-Pattern Paper — 30 marks.

Ready For Boards
10th & 12th Exam Prep

You've started Pandas.

Series · DataFrame · filtering · CSV — now prove it on the board paper.

readyforboards.com

Helpline: +91 70330 05444

Boards prep that builds confidence, not anxiety.