ANSWER KEY & MARKING SCHEME · CBSE CLASS 12

Data Handling Using Pandas – I

Informatics Practices · Chapter 1 · Use this with the Board Paper · Companion to Quick Drill

HOW TO USE

Attempt the Board Paper first (closed-book, full time). Then come here. For 2-mark+ questions, compare your answer to the model. For 3-4 mark questions, also consult the **Topper Templates** below — these show the exact step-by-step structure that scores full marks per CBSE marking-scheme conventions.

MODEL ANSWERS · BOARD PAPER

Section A — Short (2 × 4 = 8 marks)

Q1. Distinguish between a Pandas Series and a Pandas DataFrame. ***[2 marks]***

> **Ans:** Series is 1-D labelled array (one column with index). DataFrame is 2-D labelled table (multiple columns, each a Series, sharing a row index).

Q2. What is the default number of rows returned by df.head() and df.tail()? ***[2 marks]***

> **Ans:** Both default to 5 rows.

Q3. Distinguish between .loc and .iloc indexers with one example each. ***[2 marks]***

> **Ans:** .loc uses label-based indexing: df.loc['Alice'] returns row labelled Alice. .iloc uses integer position: df.iloc[0] returns first row regardless of label.

Q4. Write the command to read 'students.csv' into a DataFrame called df. ***[2 marks]***

> **Ans:** df = pd.read_csv('students.csv') (assuming pandas already imported as pd)

Section B — Medium (3 × 4 = 12 marks)

Q5. Create a Pandas Series from dict {'A':10,'B':20,'C':30}. Predict the output. ***[3 marks]***

> **Ans:** import pandas as pd s = pd.Series({'A':10,'B':20,'C':30}) print(s) Output: A 10 B 20 C 30 dtype: int64

Q6. Given DataFrame df with columns Maths, Science, English. Write Pandas code to add a new column 'Total' with the row-wise sum. ***[3 marks]***

> **Ans:** df['Total'] = df['Maths'] + df['Science'] + df['English']

Q7. Write Pandas code to filter df and keep only rows where Marks > 75 AND Subject == 'Maths'. ***[3 marks]***

> **Ans:** filtered = df[(df['Marks']>75) & (df['Subject']=='Maths')] — use & not 'and'; always parenthesise each condition.

Q8. df.drop('Old', axis=1) is called but the column 'Old' still appears in df. Why? How to actually delete it? ***[3 marks]***

> **Ans:** By default drop() returns a NEW DataFrame; it does NOT modify the original. To modify in place, either pass inplace=True (df.drop('Old', axis=1, inplace=True)) or reassign (df = df.drop('Old', axis=1)).

Section C — Long (5 × 2 = 10 marks)

Q9. Given two Series — maths = pd.Series([85,90,72], index=['A','B','C']) and science = pd.Series([78,88,95], index=['A','B','C']) — write code to (i) create a DataFrame with columns 'Maths' and 'Science'; (ii) add a column 'Total'; (iii) print only the first 2 rows; (iv) print the DataFrame after sorting by Total descending. ***[5 marks]***

> **Ans:** import pandas as pd maths = pd.Series([85,90,72], index=['A','B','C']) science = pd.Series([78,88,95], index=['A','B','C']) df = pd.DataFrame({'Maths':maths, 'Science':science}) df['Total'] = df['Maths'] + df['Science'] print(df.head(2)) print(df.sort_values('Total', ascending=False))

Q10. Given 'students.csv' with columns Name, Class, Marks. Write a program to (i) read it into df; (ii) display first 5 rows; (iii) filter rows where Class==12 AND Marks>=85; (iv) write the filtered result to 'toppers.csv' WITHOUT the Pandas integer index. ***[5 marks]***

> **Ans:** import pandas as pd df = pd.read_csv('students.csv') print(df.head()) toppers = df[(df['Class']==12) & (df['Marks']>=85)] print(toppers) toppers.to_csv('toppers.csv', index=False)

★ TOPPER ANSWER TEMPLATES

3 TEMPLATES · MEMORISE THE FORMAT

★ TOPPER TEMPLATE — 3 marks: Create a Pandas Series from a Python dictionary {'Maths': 85, 'Science': 78, 'English': 92}. Display the output and explain the index.

Annual

Step 1 *[1 mark]*	**Import + create**	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.
Step 2 *[1 mark]*	**Predict the output**	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.
Step 3 *[1 mark]*	**Explain the index**	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.

COMMON LOSS OF MARKS:

- Forgetting the pandas import — board examiners deduct ½ mark.
- Writing the output with extra commas or no alignment — Pandas prints columns aligned.
- Confusing 'index' (left column) with 'value' (right column) in the explanation.

★ TOPPER TEMPLATE — 4 marks: Create a DataFrame from a dictionary of Series for three students Alice/ Bob/Carol with Maths and Science marks. Add a new column 'Total'.

Annual

Step 1 *[1 mark]*	**Import + create the two Series**	import pandas as pd maths = pd.Series([85, 90, 72], index=['Alice','Bob','Carol']) science = pd.Series([78, 88, 95], index=['Alice','Bob','Carol'])
Step 2 *[1.5 marks]*	**Create the DataFrame**	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.
Step 3 *[1.5 marks]*	**Add the Total column**	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.

COMMON LOSS OF MARKS:

- Mismatched indices between the two Series — Pandas aligns by index, NaN where missing.
- Adding the Total column with a loop instead of vectorised arithmetic — works but is bad style.
- Forgetting to print() after each step — output is what examiners mark.

★ TOPPER TEMPLATE — 5 marks: Read a CSV file 'students.csv' into a DataFrame, display the first 5 rows, filter rows where Marks > 80, and write the filtered result to a new CSV 'toppers.csv'.

Annual

Step 1 *[1 mark]*	**Import + read CSV**	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.
Step 2 *[1 mark]*	**Display first 5 rows**	print(df.head()) No argument needed — head() defaults to 5 rows. Alternative: print(df.head(5)) makes the count explicit. Both produce identical output.
Step 3 *[1.5 marks]*	**Boolean filter**	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.
Step 4 *[1.5 marks]*	**Write to new CSV**	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.

COMMON LOSS OF MARKS:

- Using df['Marks'] > 80 OUTSIDE of df[ ] — that returns just the Boolean Series, not the filtered rows.
- Forgetting index=False in to_csv — the resulting file has an extra unnamed first column.
- Spelling read_csv with capital R — Pandas is case-sensitive.

MARKING SCHEME — GENERAL NOTES

- Always show `import pandas as pd` before any Pandas code — half-mark deduction if missing.
- For Boolean filters, compound conditions MUST use & (not 'and') with parentheses around each condition.
- to_csv requires index=False for clean output — examiners check this.
- Output predictions: align columns the way Pandas prints them (dtype line at the end of Series output).
- .loc vs .iloc distinction is the most-tested distinction in this chapter — get it right every time.