CBSE Class 12 | Informatics Practices

Chapter 2: Data Handling Using Pandas

COMPREHENSIVE QUESTION BANK

Aligned with NEET | IIT | JEE Pattern

30 MCQs	**20 Fill Blanks**	**10 Matching**	**30 FAQs**
20 Short Q&A;	**10 Long Ans**	**120 Questions**	**With Answers**

★ SECTION A: Multiple Choice Questions (MCQ)

30 Questions | 1 Mark Each

Q1. Which Python library is used for tabular data analysis?

(a) NumPy

(b) Matplotlib

(c) Pandas

(d) SciPy

✔ **Answer: (c)**

Q2. What does 'pd' represent when we write 'import pandas as pd'?

(a) Package directory

(b) Alias for Pandas

(c) Python data

(d) Primary data

✔ **Answer: (b)**

Q3. A Pandas Series is a:

(a) 2D labelled array

(b) 1D labelled array

(c) 3D matrix

(d) Dictionary

✔ **Answer: (b)**

Q4. What is the default starting index of a Pandas Series?

(a) 1

(b) -1

(c) 0

(d) None

✔ **Answer: (c)**

Q5. Which of the following creates a Series from a dictionary?

(a) pd.Series([dict])

(b) pd.Series(dict)

(c) pd.array(dict)

(d) pd.DataFrame(dict)

✔ **Answer: (b)**

Q6. In Pandas, NaN stands for:

(a) Not a Node

(b) Null and None

(c) Not a Number

(d) Numeric and Null

✔ **Answer: (c)**

Q7. Which method shows the first 5 rows of a DataFrame by default?

(a) tail()

(b) show()

(c) first()

(d) head()

✔ **Answer: (d)**

Q8. What is the axis value for columns in DataFrame.drop()?

(a) axis=0

(b) axis=2

(c) axis=1

(d) axis=-1

✔ **Answer: (c)**

Q9. Which attribute gives the shape of a DataFrame?

(a) .size

(b) .shape

(c) .ndim

(d) .dim

✔ **Answer: (b)**

Q10. Which function is used to read a CSV file into a DataFrame?

(a) pd.load_csv()

(b) pd.open_csv()

(c) pd.read_csv()

(d) pd.import_csv()

✔ **Answer: (c)**

Q11. In labelled slicing of a Pandas Series, the end index is:

(a) Excluded

(b) Included

(c) Optional

(d) None of these

✔ **Answer: (b)**

Q12. In positional slicing of a Pandas Series, the end index is:

(a) Excluded

(b) Included

(c) Optional

(d) Depends on dtype

✔ **Answer: (a)**

Q13. Which operator is used for element-wise addition of two Series with NaN handling?

(a) seriesA + seriesB

(b) seriesA.add(seriesB, fill_value=0)

(c) seriesA.sum(seriesB)

(d) seriesA.combine(seriesB)

✔ **Answer: (b)**

Q14. What does .T attribute of a DataFrame do?

(a) Shows tail

(b) Converts to tuple

(c) Transposes rows and columns

(d) Returns total count

✔ **Answer: (c)**

Q15. Which method is used to rename rows/columns in a DataFrame?

(a) relabel()

(b) setname()

(c) rename()

(d) change()

✔ **Answer: (c)**

Q16. The .loc[] accessor in Pandas is used for:

(a) Position-based indexing

(b) Label-based indexing

(c) Boolean filtering only

(d) Merging DataFrames

✔ **Answer: (b)**

Q17. What does the .empty attribute return for a non-empty Series?

(a) 0

(b) None

(c) True

(d) False

✔ **Answer: (d)**

Q18. When a Dictionary of Series is used to create a DataFrame, what becomes columns?

(a) Dictionary values

(b) Dictionary keys

(c) Series index

(d) None

✔ **Answer: (b)**

Q19. Which Pandas method exports a DataFrame to a CSV file?

(a) export_csv()

(b) save_csv()

(c) to_csv()

(d) write_csv()

✔ **Answer: (c)**

Q20. In DataFrame creation from list of dictionaries, missing keys result in:

(a) Error

(b) 0

(c) NaN

(d) Empty string

✔ **Answer: (c)**

Q21. Which library is used for numerical array-based computation in Python?

(a) Pandas

(b) Matplotlib

(c) SciPy

(d) NumPy

✔ **Answer: (d)**

Q22. To access multiple elements from a Series using index list, we use:

(a) series[1,2]

(b) series[[1,2]]

(c) series{1,2}

(d) series(1,2)

✔ **Answer: (b)**

Q23. Which parameter in to_csv() prevents writing row index labels?

(a) header=False

(b) index=False

(c) row=False

(d) label=False

✔ **Answer: (b)**

Q24. What does .values attribute of a Series return?

(a) A list

(b) A Series

(c) A NumPy array

(d) A DataFrame

✔ **Answer: (c)**

Q25. Which of the following correctly creates an empty DataFrame?

(a) pd.DataFrame(None)

(b) pd.empty()

(c) pd.DataFrame()

(d) pd.Series([])

✔ **Answer: (c)**

Q26. The .count() method on a Series counts:

(a) All elements including NaN

(b) Only NaN values

(c) Non-NaN (non-missing) values

(d) Total memory usage

✔ **Answer: (c)**

Q27. Adding a new row to DataFrame using .loc[] with a new label will:

(a) Raise an error

(b) Replace first row

(c) Append that new row

(d) Ignore the command

✔ **Answer: (c)**

Q28. In a DataFrame, axis=0 refers to:

(a) Columns

(b) Diagonal

(c) Rows

(d) Index labels

✔ **Answer: (c)**

Q29. The Pandas library full form 'Panel Data' refers to:

(a) A GUI panel

(b) Multi-dimensional structured data

(c) A chart panel

(d) Python Analysis Node Environment

✔ **Answer: (b)**

Q30. What happens when two Series with different indices are added?

(a) Error is raised

(b) Smaller Series is padded with 0

(c) Unmatched indices get NaN

(d) Only common indices are retained

✔ **Answer: (c)**

⌨ SECTION B: Fill in the Blanks

20 Questions | 1 Mark Each

Q1. Pandas is used for _____ and _____ of data.

✔ **Answer: handling and analysis**

Q2. To import Pandas with alias, we write: _____

✔ **Answer: import pandas as pd**

Q3. A Pandas Series is a _____ dimensional labelled array.

✔ **Answer: one (1)**

Q4. By default, the index of a Pandas Series starts from _____.

✔ **Answer: 0**

Q5. When a dictionary is used to create a Series, the keys become _____ and values become _____.

✔ **Answer: index / labels; values of the Series**

Q6. NaN stands for _____.

✔ **Answer: Not a Number**

Q7. The .head() method by default returns the first _____ rows.

✔ **Answer: 5**

Q8. To delete a column from a DataFrame, we set axis = _____.

✔ **Answer: 1**

Q9. The .shape attribute returns a _____ with (rows, columns).

✔ **Answer: tuple**

Q10. The command to read a CSV file in Pandas is _____.

✔ **Answer: pd.read_csv('filename.csv')**

Q11. In labelled slicing, the end label is _____ in the result.

✔ **Answer: included**

Q12. In positional slicing, the end index is _____ from the result.

✔ **Answer: excluded**

Q13. The _____ attribute of a DataFrame swaps rows and columns.

✔ **Answer: .T (Transpose)**

Q14. To add a fill value for missing data during Series addition, use _____.

✔ **Answer: .add(seriesB, fill_value=0)**

Q15. The _____ method exports a DataFrame to a CSV file.

✔ **Answer: to_csv()**

Q16. A DataFrame is a _____ dimensional data structure.

✔ **Answer: two (2)**

Q17. The .loc[] accessor uses _____ based indexing.

✔ **Answer: label**

Q18. Missing data during DataFrame creation from list of dicts is filled as _____.

✔ **Answer: NaN**

Q19. The _____ attribute returns the total number of elements in a DataFrame.

✔ **Answer: .size**

Q20. The command to install pandas is: _____

✔ **Answer: pip install pandas**

↔ SECTION C: Match the Following

10 Sets | 1 Mark Each

Set 1: Library & Purpose

Column A	Column B
1. NumPy	A. Tabular data analysis
2. Pandas	B. Numerical array computation
3. Matplotlib	C. Data visualisation / graphs

✔ **Correct Match: NumPy → Numerical; Pandas → Tabular; Matplotlib → Graphs**

Set 2: Data Structure & Dimension

Column A	Column B
1. Series	A. 2-Dimensional
2. DataFrame	B. 1-Dimensional
3. NumPy ndarray	C. N-Dimensional

✔ **Correct Match: Series → 1D; DataFrame → 2D; ndarray → N-D**

Set 3: Method & Action

Column A	Column B
1. head()	A. Last n rows
2. tail()	B. Non-NaN count
3. count()	C. First n rows

✔ **Correct Match: head()→First rows; tail()→Last rows; count()→Non-NaN count**

Set 4: Attribute & Result

Column A	Column B
1. .shape	A. Total elements
2. .size	B. Row & Column count tuple
3. .T	C. Swaps rows & columns

✔ **Correct Match: .shape→(rows,cols); .size→total elements; .T→Transpose**

Set 5: Axis Value & Operation Target

Column A	Column B
1. axis=0	A. Columns
2. axis=1	B. Rows

3. axis='index'	C. Row labels in rename

✔ **Correct Match: axis=0→Rows; axis=1→Columns; axis='index'→Row labels**

Set 6: Indexing Type & Behaviour

Column A	Column B
1. Positional slicing	A. End label included
2. Label slicing	B. End position excluded
3. .loc[]	C. Label-based accessor

✔ **Correct Match: Positional→End excluded; Label→End included; .loc[]→Label-based**

Set 7: Function & CSV Operation

Column A	Column B
1. pd.read_csv()	A. Export DataFrame to CSV
2. to_csv()	B. Read CSV into DataFrame
3. sep parameter	C. Defines column separator

✔ **Correct Match: read_csv→Import; to_csv→Export; sep→Separator**

Set 8: Missing Data Concept

Column A	Column B
1. NaN	A. Fill missing during math ops
2. fill_value=0	B. Not a Number placeholder
3. .dropna()	C. Remove rows with missing data

✔ **Correct Match: NaN→placeholder; fill_value→fill during ops; dropna→remove missing**

Set 9: Creation Method & Source

Column A	Column B
1. pd.Series([1,2,3])	A. From dictionary
2. pd.Series({'a':1})	B. From list
3. pd.Series(np.array([]))	C. From NumPy array

✔ **Correct Match: List→pd.Series(list); Dict→pd.Series(dict); Array→pd.Series(np.array)**

Set 10: DataFrame Attribute & Output

Column A	Column B
1. .dtypes	A. Column header labels
2. .columns	B. Data type of each column

3. .values	C. All data as NumPy array

✔ **Correct Match: .dtypes→data types; .columns→headers; .values→NumPy array**

■ SECTION D: Frequently Asked Questions (FAQ)

30 Questions | 2-3 Marks Each

Q1. What is a Python Library?

Ans: A Python Library is a collection of pre-written functions and modules that developers can import and use in their programs to perform common tasks without writing code from scratch. Examples: NumPy, Pandas, Matplotlib.

Q2. What is Pandas and why is it used?

Ans: Pandas is an open-source Python library used for data analysis and manipulation. It provides two main data structures — Series (1D) and DataFrame (2D) — to handle structured/tabular data efficiently. It supports reading/writing CSV files, filtering, aggregation, and much more.

Q3. What is the difference between NumPy and Pandas?

Ans: NumPy works with homogeneous data (same type) and supports N-dimensional arrays. Pandas works with heterogeneous data (mixed types) in labelled structures (Series/DataFrame). Pandas allows custom index labels, while NumPy uses integer positions only.

Q4. How do you install and import Pandas?

Ans: Install: pip install pandas. Import: import pandas as pd. The alias 'pd' is a convention used universally by the community.

Q5. What is a Pandas Series?

Ans: A Pandas Series is a one-dimensional labelled array capable of holding any data type (integers, strings, floats, etc.). It has an index (labels) and corresponding values. Index can be default (0,1,2...) or custom.

Q6. How can you create a Series from a dictionary?

Ans: When pd.Series(dict) is called, the dictionary keys become the index labels and dictionary values become the Series values. Example: pd.Series({'India':'NewDelhi', 'UK':'London'})

Q7. What is the difference between positional and label-based slicing in Series?

Ans: In positional slicing (series[1:3]), the end index is EXCLUDED — like Python lists. In label-based slicing (series['USA':'France']), the end label is INCLUDED. This is an important Pandas-specific behaviour.

Q8. What is NaN in Pandas?

Ans: NaN stands for Not a Number. It represents a missing or undefined value. When two Series with different indices are added, unmatched positions get NaN. Use fill_value parameter in .add(), .sub() etc. to replace NaN with a default value.

Q9. What is a DataFrame in Pandas?

Ans: A DataFrame is a two-dimensional labelled data structure with rows and columns — similar to a spreadsheet or SQL table. Each column is essentially a Series. It can hold data of different types.

Q10. How is a DataFrame created from a Dictionary of Lists?

Ans: Each key of the dictionary becomes a column name, and the corresponding list becomes column values. Example: pd.DataFrame({'Name':['A','B'],'Marks':[90,85]})

Q11. How do you add a new column to a DataFrame?

Ans: By assignment: df['NewColumn'] = [values]. If column already exists, it gets updated. If it's new, it gets appended.

Q12. How do you add a new row to a DataFrame?

Ans: Using .loc[]: df.loc['NewRowLabel'] = [values]. Number of values must match number of columns, otherwise ValueError is raised.

Q13. How do you delete rows and columns from a DataFrame?

Ans: Using .drop(): df.drop('RowLabel', axis=0) removes a row; df.drop('ColLabel', axis=1) removes a column. For multiple, pass a list: df.drop(['col1','col2'], axis=1).

Q14. What does the .rename() method do?

Ans: .rename() allows renaming of row labels (axis='index') or column labels (axis='columns'). Example: df.rename({'OldName':'NewName'}, axis='columns')

Q15. Explain .loc[] with an example.

Ans: .loc[] is a label-based indexer. df.loc['Maths'] returns the row labelled Maths. df.loc['Maths':'Science','Arnab'] slices rows from Maths to Science for column Arnab.

Q16. What is Boolean Indexing in Pandas?

Ans: Boolean Indexing filters data based on a condition. Example: df.loc[df.loc['Maths'] > 90] returns only those columns where Maths marks > 90. Each element is evaluated as True/False and only True ones are displayed.

Q17. What are the key attributes of a Pandas DataFrame?

Ans: .index (row labels), .columns (column labels), .dtypes (data type per column), .values (numpy array), .shape (rows, cols tuple), .size (total elements), .T (transpose), .empty (True if empty).

Q18. How do you read a CSV file into a DataFrame?

Ans: Using pd.read_csv('filepath', sep=',', header=0). Parameters: sep = separator character; header = row number with column names; names = custom column names.

Q19. How do you export a DataFrame to a CSV file?

Ans: Using df.to_csv('filepath', sep=',', header=True, index=True). Set header=False to skip column names, index=False to skip row labels.

Q20. What happens when two Series with different indexes are added without fill_value?

Ans: When indexes don't match, the unmatched positions get NaN (Not a Number) in the result. To avoid NaN, use seriesA.add(seriesB, fill_value=0) — this replaces missing values with 0 before addition.

Q21. What is the difference between Series.size and DataFrame.size?

Ans: Series.size = total number of elements in the Series. DataFrame.size = total number of elements = rows × columns. Both count NaN as elements too.

Q22. What is the use of .count() method?

Ans: .count() returns the number of non-NaN (non-missing) values in a Series or each column of a DataFrame. It does NOT count NaN values, unlike .size.

Q23. What does .values attribute return?

Ans: .values returns all the data as a NumPy ndarray, stripping away the index and column labels. Useful when you need raw array data for computation.

Q24. Why is the alias 'pd' used for Pandas?

Ans: 'pd' is not mandatory but is a universal convention followed by all Pandas developers worldwide. Using a different alias makes code less readable for others. It saves typing the full word 'pandas' each time.

Q25. What is the difference between .head() and .tail()?

Ans: .head(n) displays the first n rows (default 5). .tail(n) displays the last n rows (default 5). Both are useful for quickly previewing large DataFrames.

Q26. How do you access a single element from a Series?

Ans: Using index: series[index_label] or series[position]. For Series with custom index: seriesCapCntry['India'] returns 'NewDelhi'. For positional: seriesCapCntry[0] returns first element.

Q27. What is the Transpose of a DataFrame?

Ans: Transpose (.T) swaps rows and columns — what was a row becomes a column and vice versa. Useful when data needs to be restructured for analysis or presentation.

Q28. How is a DataFrame created from List of Dictionaries?

Ans: Each dictionary in the list represents one row. Keys become column labels. If a key is missing in any dictionary, that cell becomes NaN. Example: pd.DataFrame([{'a':10,'b':20},{'a':5,'b':10,'c':20}])

Q29. What is the role of Matplotlib with Pandas?

Ans: Matplotlib is used for data visualisation — creating plots, bar charts, histograms, line graphs etc. Pandas DataFrames and Series integrate seamlessly with Matplotlib to produce visual representations of data.

Q30. What is the significance of fill_value in Series arithmetic methods?

Ans: fill_value in methods like .add(), .sub(), .mul(), .div() replaces NaN that would otherwise appear when indexes don't match. Example: seriesA.add(seriesB, fill_value=0) uses 0 where one Series has no value.

■ SECTION E: Short Answer Questions

20 Questions | 3-4 Marks Each

Q1. Write Python code to create a Series of 5 student names with custom index starting from 1.

Ans:

```
import pandas as pd
s = pd.Series(['Arnab','Samridhi','Ramit','Divyam','Kritika'], index=[1,2,3,4,5])
print(s)
# Output:
# 1 Arnab
# 2 Samridhi
# 3 Ramit
# 4 Divyam
# 5 Kritika
# dtype: object
```

Q2. Differentiate between positional and label-based indexing in Pandas Series with example.

Ans: Positional Indexing: Uses integer position (0,1,2...). series[2] → returns element at position 2 (end excluded in slicing) Label-based Indexing: Uses custom labels. series['India'] → returns value for key 'India' (end included in slicing) Key difference: In label slicing series['USA':'France'], France IS included. In positional slicing series[1:3], index 3 is NOT included.

Q3. Write code to create a DataFrame from a dictionary of lists for 3 students with Name, Maths, Science columns.

Ans:

```
import pandas as pd
data = {
'Name' : ['Arnab', 'Kritika', 'Ramit'],
'Maths' : [90, 85, 78],
'Science': [88, 92, 81]
}
df = pd.DataFrame(data)
print(df)
```

Q4. Explain the use of axis parameter in drop() with example.

Ans:

```
axis=0 → operates on rows (deletes rows)
axis=1 → operates on columns (deletes columns)

Example:
df.drop('Science', axis=0) # removes row labelled 'Science'
df.drop('Arnab', axis=1) # removes column 'Arnab'
df.drop(['Riya','Malika'], axis=1) # removes multiple columns

Mnemonic: axis=0 = rows (downward), axis=1 = columns (sideways)
```

Q5. What will be the output of the following code? seriesA = pd.Series([1,2,3], index=['a','b','c']) seriesB = pd.Series([10,20], index=['a','c']) print(seriesA + seriesB)

Ans:

```
Output:
a 11.0
b NaN ← 'b' exists only in seriesA, no match in seriesB
c 23.0
dtype: float64

Reason: Pandas matches by index. 'a' and 'c' match, so addition happens.
'b' has no corresponding value in seriesB → result is NaN.
```

Q6. Write Python code to add a new row and a new column to a DataFrame.

Ans:

```
# Adding a new column:
df['English'] = [85, 79, 76] # List must match number of rows

# Adding a new row using .loc[]:
df.loc['Student4'] = [88, 90, 76, 82] # Values must match number of columns

Note: Value count must match exactly, else ValueError is raised.
```

Q7. What is Boolean Indexing? Write code to filter students with Maths > 90.

Ans: Boolean Indexing filters rows/columns based on a condition (True/False). Code: condition = ResultDF.loc['Maths'] > 90 # Returns True for students scoring > 90 in Maths filtered = ResultDF.loc[:, condition] # OR filtered = ResultDF.loc[ResultDF.loc['Maths'] > 90] Only columns/rows where condition is True are displayed.

Q8. Explain pd.read_csv() with all important parameters.

Ans: pd.read_csv(filepath, sep=',', header=0, names=None, index_col=None) filepath → path to the CSV file (string) sep → delimiter/separator character (default: ',') header → row number to use as column names (default: 0 = first row) names → list of custom column names to use index_col → column to use as row index Example: marks = pd.read_csv('C:/data/result.csv', sep=',', header=0)

Q9. Write code demonstrating .rename() to rename both rows and columns.

Ans:

```
import pandas as pd
# Rename rows (axis='index' or axis=0):
df = df.rename({'Maths':'Sub1', 'Science':'Sub2'}, axis='index')

# Rename columns (axis='columns' or axis=1):
df = df.rename({'Arnab':'Student1', 'Ramit':'Student2'}, axis='columns')

Note: .rename() returns a new DataFrame. Use inplace=True to modify original.
```

Q10. Differentiate between Series.size and Series.count().

Ans:

```
Series.size:
→ Returns TOTAL number of elements including NaN values.
→ Example: pd.Series([1, 2, NaN]).size → 3

Series.count():
→ Returns count of NON-NaN (non-missing) values only.
→ Example: pd.Series([1, 2, NaN]).count() → 2

Key Insight: .size counts everything; .count() skips missing values.
```

Q11. Write code to create a Series from a NumPy array and display its index and values separately.

Ans:

```
import numpy as np
import pandas as pd

arr = np.array([100, 200, 300, 400])
s = pd.Series(arr, index=['Q1','Q2','Q3','Q4'])

print(s.index) # Index(['Q1','Q2','Q3','Q4'], dtype='object')
print(s.values) # [100 200 300 400] ← NumPy array
print(s.size) # 4
```

Q12. What is the .T attribute? Show its effect with an example.

Ans: The .T attribute transposes a DataFrame — rows become columns and columns become rows. Before Transpose (3 rows, 2 cols): A B 0 1 4 1 2 5 2 3 6 After df.T (2 rows, 3 cols): 0 1 2 A 1 2 3 B 4 5 6 Use case: When data format needs to be rotated for analysis/display.

Q13. How do you export a DataFrame to CSV without row index and column headers?

Ans:

```
Use to_csv() with index=False and header=False:

df.to_csv('C:/data/output.csv', sep=',', index=False, header=False)

index=False → Row labels (0,1,2...) will NOT be written to CSV
header=False → Column names will NOT be written as first row
sep=',' → Comma is used as separator (default)

Useful when CSV is consumed by systems that don't need labels.
```

Q14. Write a program to create a DataFrame from dictionary of Series and display its shape and dtypes.

Ans:

```
import pandas as pd

s1 = pd.Series([90, 85, 78], index=['Maths','Science','Hindi'])
s2 = pd.Series([88, 92, 81], index=['Maths','Science','Hindi'])

df = pd.DataFrame({'Arnab': s1, 'Kritika': s2})
print(df)
print("Shape:", df.shape) # (3, 2)
print("Dtypes:")
print(df.dtypes) # Maths→int64, Science→int64, Hindi→int64
```

Q15. What is the difference between .loc[] row slice and column slice in a DataFrame?

Ans:

```
Row slice: df.loc['Maths':'Science']
→ Returns all rows from label 'Maths' to 'Science' (inclusive), all columns.

Column slice via .loc[]:
df.loc[:, 'Arnab':'Ramit']
→ Returns all rows, columns from 'Arnab' to 'Ramit' (inclusive).

Combined: df.loc['Maths':'Science', 'Arnab':'Ramit']
→ Row slice + column slice together.

Note: Both start AND end labels are INCLUDED in .loc[] slicing.
```

Q16. Explain what happens when creating a DataFrame from two NumPy arrays of unequal length.

Ans: When creating a DataFrame from arrays of unequal length using column names, the shorter array rows get NaN for the extra columns. Example: arr1 = np.array([10, 30]) # 2 elements arr2 = np.array([-10, -20, 30, 40]) # 4 elements df = pd.DataFrame([arr1, arr2], columns=['A','B','C','D']) Row 0 (arr1): A=10, B=30, C=NaN, D=NaN Row 1 (arr2): A=-10, B=-20, C=30, D=40

Q17. What is the difference between .index, .columns, and .values attributes of a DataFrame?

Ans: .index → Returns the row labels (RangeIndex or custom Index object) Example: RangeIndex(start=0, stop=3, step=1) .columns → Returns the column labels as an Index object Example: Index(['Name', 'Maths', 'Science'], dtype='object') .values → Returns all data as a 2D NumPy ndarray (no labels) Example: array([['Arnab', 90, 88], ['Kritika', 85, 92]]) Key: .values strips away all labels — raw data only.

Q18. Write code showing how to access multiple rows and a specific column using .loc[].

Ans:

```
import pandas as pd

# Accessing multiple rows + single column:
result = ResultDF.loc[['Maths','Hindi'], 'Arnab']

# Accessing row range + multiple columns:
result = ResultDF.loc['Maths':'Science', ['Arnab','Malika']]

# Accessing single row (returns Series):
row = ResultDF.loc['Science']

# Accessing single column (returns Series):
col = ResultDF['Arnab'] # or ResultDF.loc[:,'Arnab']
```

Q19. What is the significance of the 'sep' parameter in read_csv() and to_csv()?

Ans: The 'sep' parameter defines the character used to separate values in the CSV file. In read_csv(): pd.read_csv('file.csv', sep=',') # comma-separated (default) pd.read_csv('file.tsv', sep='\t') # tab-separated In to_csv(): df.to_csv('file.txt', sep='#') # uses # as separator Why it matters: Real-world files may use different delimiters. If sep is wrong, data won't parse correctly — all text may land in one column.

Q20. Explain .append() method for joining DataFrames with a suitable example.

Ans: The .append() method joins two DataFrames vertically (row-wise). df1 = pd.DataFrame([[1,2,3],[4,5,6]], columns=['C1','C2','C3'], index=['R1','R2']) df2 = pd.DataFrame([[10,20],[40,50]], columns=['C2','C3'], index=['R4','R2']) result = df1.append(df2) Output: C1 C2 C3 R1 1.0 2.0 3.0 R2 4.0 5.0 6.0 R4 NaN 10.0 20.0 R2 NaN 40.0 50.0 Note: Missing columns fill with NaN. sort=True sorts columns alphabetically.

■ SECTION F: Long Answer Questions

10 Questions | 5-6 Marks Each

Q1. Explain all the ways to create a Pandas Series with Python code examples for each method.

```
A Pandas Series is a 1D labelled array. It can be created in 4 main ways:
```

1. From a Python List (Scalar values):

```
import pandas as pd
s1 = pd.Series([10, 20, 30]) # Default index: 0,1,2
s2 = pd.Series([2,3,4], index=['Feb','Mar','Apr']) # Custom index
```

2. From a NumPy Array:

```
import numpy as np
arr = np.array([1,2,3,4])
s3 = pd.Series(arr) # Default index used
# Note: array length must match index length, else ValueError
```

3. From a Dictionary:

```
dict1 = {'India':'NewDelhi', 'UK':'London', 'Japan':'Tokyo'}
s4 = pd.Series(dict1)
# Keys → Index labels | Values → Series values
```

4. From a Scalar value (broadcasts to all positions):

```
s5 = pd.Series(5, index=['a','b','c'])
# Output: a 5, b 5, c 5
```

```
Key Points: import pandas as pd is always required. Custom index and values must
have same length. Dictionary-based Series automatically uses keys as labels.
```

Q2. Describe all methods to create a Pandas DataFrame with examples.

A DataFrame is a 2D labelled structure. It can be created in the following ways:

```
1. Empty DataFrame: pd.DataFrame()
```

2. From NumPy Arrays:

```
import numpy as np, pandas as pd
a1 = np.array([10,30])
a2 = np.array([-10,-20,30,40])
df = pd.DataFrame([a1,a2], columns=['A','B','C','D'])
# Shorter arrays get NaN for extra columns
```

3. From List of Dictionaries:

```
listDict = [{'a':10,'b':20}, {'a':5,'b':10,'c':20}]
df = pd.DataFrame(listDict)
# Missing key → NaN; Each dict = one row
```

4. From Dictionary of Lists:

```
data = {'State':['Assam','Delhi'], 'Area':[78438,1483]}
df = pd.DataFrame(data)
# Keys=columns, List values=column data
```

5. From Dictionary of Series:

```
s1 = pd.Series([90,85], index=['Maths','Science'])
s2 = pd.Series([88,92], index=['Maths','Science'])
df = pd.DataFrame({'Arnab':s1, 'Kritika':s2})
```

Each method is suited for different data sources. Dictionary of Series is most common in exams.

Q3. Explain Series arithmetic operations in Pandas with examples including NaN handling.

Pandas Series support element-wise arithmetic: +, -, *, /

KEY RULE: Operations are performed by matching INDEX LABELS, not positions.

Setup:

```
seriesA = pd.Series([1,2,3,4,5], index=['a','b','c','d','e'])
seriesB = pd.Series([10,20,-10,-50,100], index=['a','b','y','n','c'])
```

1. Addition (+ operator):

```
result = seriesA + seriesB
# a:11, b:22, c:NaN(3+?), d:NaN(?+?), e:NaN, y:NaN, n:NaN
# Unmatched indices → NaN
```

2. Addition with fill_value (avoids NaN):

```
result = seriesA.add(seriesB, fill_value=0)
# Missing values replaced with 0 before operation
```

```
3. Subtraction: seriesA - seriesB OR seriesA.sub(seriesB, fill_value=0)
```

```
4. Multiplication: seriesA * seriesB OR seriesA.mul(seriesB, fill_value=0)
```

```
5. Division: seriesA / seriesB OR seriesA.div(seriesB, fill_value=0)
```

```
Important: Results are float64 when NaN is involved. fill_value=0 is recommended to
avoid NaN in results.
```

Q4. Explain label-based indexing using .loc[] in a DataFrame with different use cases and code.

```
.loc[] is the primary label-based accessor in Pandas. Syntax: df.loc[row_label,
col_label]
```

```
Setup: ResultDF has students as columns (Arnab,Ramit..) and subjects as rows
(Maths,Science,Hindi)
```

1. Single Row (returns Series):

```
ResultDF.loc['Science'] → Returns Science row as a Series
```

2. Single Column:

```
ResultDF['Arnab'] OR ResultDF.loc[:,'Arnab'] → Arnab's column
```

3. Multiple Rows (list):

```
ResultDF.loc[['Science','Hindi']] → Both rows
```

4. Row Slicing (both ends included):

```
ResultDF.loc['Maths':'Science'] → Maths and Science rows
```

5. Row + Column:

```
ResultDF.loc['Maths':'Science', 'Arnab'] → Row slice, single col
```

6. Row + Multiple Columns:

```
ResultDF.loc['Maths':'Science', ['Arnab','Malika']]
```

7. Boolean Indexing:

```
condition = ResultDF.loc['Maths'] > 90
filtered = ResultDF.loc[:, condition]
# Shows only students who scored >90 in Maths

Summary: .loc[] always uses labels. Start AND end are both included. Use double
brackets [[]] for multiple rows/columns.
```

Q5. Explain all operations that can be performed on a DataFrame: add column, add row, delete, rename with code.

Once a DataFrame is created, we can modify it in several ways:

Setup:

```
Arnab = pd.Series([90,92,97], index=['Maths','Science','Hindi'])
Ramit = pd.Series([82,81,90], index=['Maths','Science','Hindi'])
ResultDF = pd.DataFrame({'Arnab':Arnab, 'Ramit':Ramit})
```

1. ADD a new Column:

```
ResultDF['Preeti'] = [85, 79, 76]
# Values count must equal number of rows

2. ADD a new Row using .loc[]:

ResultDF.loc['English'] = [85, 85, 85]
# Values count must equal number of columns
```

3. DELETE Row (axis=0):

```
ResultDF = ResultDF.drop('Science', axis=0)
ResultDF = ResultDF.drop(['Science','Hindi'], axis=0)
```

4. DELETE Column (axis=1):

```
ResultDF = ResultDF.drop('Arnab', axis=1)
ResultDF = ResultDF.drop(['Arnab','Ramit'], axis=1)
```

5. RENAME Rows (axis='index'):

```
ResultDF = ResultDF.rename({'Maths':'Sub1','Science':'Sub2'}, axis='index')
```

6. RENAME Columns (axis='columns'):

```
ResultDF = ResultDF.rename({'Arnab':'Stu1','Ramit':'Stu2'}, axis='columns')

Key Rule: axis=0 → rows | axis=1 → columns. Always remember this for exams!
```

Q6. Compare Pandas Series and NumPy ndarray. When should you use each?

```
Both Series and ndarray are used for 1D/ND data, but have key differences:
```

PANDAS SERIES:

```
→ Has labelled index (can be custom: 'Jan','Feb' or integers)

→ Supports heterogeneous data per element (mixed types)

→ Index can be non-sequential or descending

→ Unaligned operations result in NaN (safe handling)

→ Uses more memory due to index overhead

→ Built on top of NumPy arrays internally

Example: pd.Series([1,2,3], index=['a','b','c'])
```

NUMPY NDARRAY:

```
→ Integer position only (0,1,2...) — no custom labels
→ Homogeneous data (all same type) — faster computation
→ Fixed integer index only
→ Mismatch in operations raises an error
→ Uses less memory — optimised for numerical computation
Example: np.array([1,2,3])
```

When to use Series: Data with meaningful labels, mixed operations, data analysis

```
When to use ndarray: Pure numerical computation, performance-critical code,
scientific computing
```

Q7. Explain the complete process of reading from and writing to CSV files using Pandas.

CSV (Comma Separated Values) is the most common data exchange format. Pandas handles it seamlessly.

```
READING a CSV file — pd.read_csv():
marks = pd.read_csv('C:/NCERT/ResultData.csv', sep=',', header=0)
# This creates a DataFrame from the CSV file
```

Parameters of read_csv():

```
filepath → path to CSV file (string, required)
sep → separator char (default: ',')
Use '\t' for tab-separated, ';' for semicolons
header → which row is column names (default: 0 = first row)
Use header=None if file has no header row
names → list of custom column names to assign
index_col → which column to use as row index
```

WRITING to CSV — to_csv():

```
ResultDF.to_csv('C:/NCERT/output.csv', sep=',')
# Creates output.csv with data, headers, and row index
```

Parameters of to_csv():

```
filepath → output file path (required)
sep → separator (default: ',')
header → True/False — write column names? (default True)
index → True/False — write row labels? (default True)
Use index=False for clean data without row numbers
```

Practical Example:

```
# Export without index/header (clean data only):
ResultDF.to_csv('clean_data.txt', sep='#', header=False, index=False)
```

CSV is universally compatible with Excel, databases, and other tools.

Q8. Describe all important attributes of a Pandas DataFrame with their use and examples.

DataFrame attributes allow you to inspect structural properties without modifying data.

1. .index — Row labels:

```
df.index → RangeIndex(start=0, stop=4, step=1) or custom Index
```

2. .columns — Column labels:

```
df.columns → Index(['State','GArea','VDF'], dtype='object')
```

3. .dtypes — Data type of each column:

```
df.dtypes
# State: object, GArea: int64, VDF: float64
```

```
4. .values – All data as NumPy array (labels stripped):
```

```
df.values → array([['Assam',78438,2797.0],['Delhi',1483,6.72]])
```

5. .shape — (rows, columns) tuple:

```
df.shape → (3, 3) # 3 rows, 3 columns
```

6. .size — Total elements = rows × columns:

```
df.size → 9 # 3 × 3
```

7. .T — Transpose (swap rows & columns):

```
df.T → columns become rows, rows become columns
```

8. .head(n) / .tail(n) — Preview data:

```
df.head(3) → first 3 rows
df.tail(2) → last 2 rows
```

9. .empty — Is DataFrame empty?

```
pd.DataFrame().empty → True
df.empty → False (if has data)
```

These attributes are read-only properties (no parentheses needed, except head/tail).

Q9. Write a complete Python program to: create a DataFrame of student marks, add/delete rows and columns, rename labels, and export to CSV.

Complete Program:

```
import pandas as pd

# Step 1: Create DataFrame
Arnab = pd.Series([90,92,97], index=['Maths','Science','Hindi'])
Kritika= pd.Series([85,88,91], index=['Maths','Science','Hindi'])
Malika = pd.Series([94,95,99], index=['Maths','Science','Hindi'])

ResultDF = pd.DataFrame({'Arnab':Arnab,'Kritika':Kritika,'Malika':Malika})
print("Original DataFrame:")
print(ResultDF)

# Step 2: Add new column
ResultDF['Preeti'] = [78, 82, 88]
print("\nAfter adding Preeti column:")
print(ResultDF)

# Step 3: Add new row using .loc[]
ResultDF.loc['English'] = [85, 90, 92, 80]
print("\nAfter adding English row:")
print(ResultDF)

# Step 4: Delete a row
ResultDF = ResultDF.drop('Science', axis=0)
print("\nAfter deleting Science row:")
print(ResultDF)

# Step 5: Delete a column
ResultDF = ResultDF.drop('Preeti', axis=1)
print("\nAfter deleting Preeti column:")
print(ResultDF)

# Step 6: Rename rows and columns
ResultDF = ResultDF.rename({'Maths':'Sub1','Hindi':'Sub2','English':'Sub3'},
axis='index')
ResultDF = ResultDF.rename({'Arnab':'Stu1','Kritika':'Stu2','Malika':'Stu3'},
axis='columns')
print("\nAfter renaming:")
print(ResultDF)

# Step 7: Export to CSV (without index)
ResultDF.to_csv('StudentResult.csv', sep=',', index=False)
print("\nData exported to StudentResult.csv")
print("Shape:", ResultDF.shape)
```

This program covers all major DataFrame operations in one flow.

Q10. Explain Python Libraries for Data Handling. Compare NumPy, Pandas and Matplotlib in detail.

Python Libraries are collections of pre-written functions that save programming time and effort.

WHY LIBRARIES?

```
→ Avoid rewriting common code (arrays, statistics, graphs)

→ Tested, optimised functions ready to use

→ Install once, import whenever needed

→ Vast community support
```

1. NumPy (Numerical Python):

```
Purpose : Numerical computation with N-dimensional arrays
Key Feature: Homogeneous data (all same type), fast computation
Use Case : Scientific computing, matrix operations, linear algebra
Import : import numpy as np
Example : np.array([1,2,3,4])
```

2. Pandas (Panel Data):

```
Purpose : Data analysis and manipulation with labelled structures
Key Feature: Heterogeneous data (mixed types), custom index labels
Use Case : Data cleaning, CSV handling, analysis, filtering
Import : import pandas as pd
Structures : Series (1D), DataFrame (2D)
Example : pd.DataFrame({'Name':['A','B'],'Marks':[90,85]})
```

3. Matplotlib:

```
Purpose : Data visualisation – plots, charts, graphs
Key Feature: Works seamlessly with NumPy and Pandas
Use Case : Bar charts, line plots, histograms, scatter plots
Import : import matplotlib.pyplot as plt
Example : plt.plot([1,2,3],[4,5,6])
```

COMPARISON TABLE:

```
NumPy: 1D-ND arrays | Same type | Numerical ops | No custom labels
```

Pandas: 1D/2D structures | Mixed types | Data analysis | Custom labels

Matplotlib: Visualisation | N/A | Graphing | N/A

```
In Data Science, all three work TOGETHER: NumPy for raw arrays, Pandas for data
handling, Matplotlib for visualisation.
```