PREMIUM EDITION

B.PHARMA · IST YEAR · 2026-27

Data Handling with Pandas

UNIT-IV · BASICS OF PYTHON IN PHARMACEUTICAL SCIENCES

PK studies and ADR reports as Series and DataFrames – read, clean, filter and summarise real pharma tables.

Core Concepts | Worked Examples | Infographics | Case Study | Smart Revision | Exam Points

Inside Unit IV

4.1 Series + DataFrame anatomy

4.2 read_csv / read_excel

4.3 head, info, describe

4.4 Cleaning: isna, dropna, fillna

4.5 Filtering + groupby

Levels: Remember to Analyze

Pills and Python · PCI-aligned · Remember to Analyze

§ UNIT OVERVIEW

4 Unit-4 at a Glance

4.1 Pandas intro

- *Why pandas* for pharma tables
- `Series` = one labelled column
- `DataFrame` = full table

4.1 Series/DataFrame

- Series from a *dose list*
- Frame from a *dict of ADR lists*
- index, columns, values, shape

4.2 Reading files

- `read_csv()` for PK study CSV
- `read_excel()` for ADR workbook
- sheet_name picks the sheet

4.3 First look

- `head()` / `tail()` preview rows
- `info()` types + missing
- `describe()` stats

4.4 Cleaning

- `isna().sum()` counts gaps
- `dropna()` removes them
- `fillna()` fills them

4.5 Filter + groupby

- Boolean mask *filters rows*
- `groupby()` + mean / count
- ADR summary pipeline

Learning Roadmap



§ 4.1 WHY PANDAS + SERIES

1 Why Pandas? Meet the Series

Definition

pandas is Python's labelled-table toolkit : each column carries patient IDs or time points as labels, so selections, missing-value handling and group summaries stay readable. A *Series* is one labelled column; a *DataFrame* is a bundle of Series sharing one index.

Lists vs Series - the ladder

Python list: values only, position access `doses[2]`. NumPy: fast math, still no labels. *Series: values + index labels* - `doses["P03"]` reads as patient P03. That label is what makes PK time-courses and ADR severity counts self-documenting.

series_dose.py

```
# Series: one labelled column of doses (mg)
import pandas as pd
doses = pd.Series([100, 250, 500, 500, 250],
    index=["P01", "P02", "P03", "P04", "P05"], name="dose_mg")
print(doses.mean(), doses.max())
```

Output P01 100 P02 250 P03 500 P04 500 P05 250 | mean 320.0 max 500

Pharma Read

Dose Series answer viva questions fast: `doses["P03"]` gives one patient, `doses.mean()` gives the batch average. Labels beat positions every time.

§ 4.1 DATAFRAME ANATOMY

DataFrame – columns, index, values

Synthetic teaching data – generated with seeded numpy for classroom practice.

frame_mini.py

```
# DataFrame from a dict: each key becomes a column
adr_mini = pd.DataFrame({
    "patient": ["P01", "P02", "P03"],
    "drug": ["PCM", "IBU", "AMX"],
    "severity": ["mild", "severe", "moderate"]})
print(adr_mini.shape, list(adr_mini.columns))
```

Output patient drug severity | P01 PCM mild P02 IBU severe P03 AMX moderate | (3, 3)

Part	What it is	Example	Access
index	row labels	0, 1, 2	`df.index`
columns	column labels	drug, severity	`df.columns`
values	raw 2-D array	3 x 3 block	`df.values`
shape	(rows, cols)	(3, 3)	`df.shape`

Professor's Note

Dict keys become columns, dict values align row by row. Unequal list lengths raise ValueError – the examiner's favourite one-mark trap.

PHARMACY • DATA • AI • HEALTHCARE

§ 4.2 SYNTHETIC PK DATASET

Dataset 1 - plasma concentration vs time

Synthetic teaching data - generated with seeded numpy for classroom practice.

Design: 5 patients x 4 timepoints

Columns: `patient` `time_hr` `conc_ng_ml`. *Seeded numpy (seed 42)* so every run reproduces the same table. Exponential decay + small noise mimics a real single-dose PK curve.

build_pk.py

```
# Seeded synthetic PK study (20 rows)
import numpy as np, pandas as pd
np.random.seed(42)
rows = []
for p in ["P01", "P02", "P03", "P04", "P05"]:
    cmax = np.random.uniform(700, 950)
    for t in [0.5, 1.0, 2.0, 4.0]:
        conc = round(cmax*np.exp(-0.35*t)+np.random.normal(0,18), 1)
        rows.append((p, t, conc))
pk = pd.DataFrame(rows, columns=["patient", "time_hr", "conc_ng_ml"])
```

Output 20 rows | P01 0.5h 646.2 P01 1.0h 565.0 P02 0.5h 727.8 ... | conc falls with time

Why Seeded?

Without `seed(42)` every student gets different numbers and answers stop matching. Seeded data = one answer key for the whole class.

§ 4.2 SYNTHETIC ADR DATASET

Dataset 2 – ADR reports with missing outcomes

Synthetic teaching data – generated with seeded numpy for classroom practice.

build_adr.py

```
# 12 ADR reports; 2 outcomes left blank (NaN) on purpose
adr = pd.DataFrame({
    "patient": ["P01".."P12"], # 12 IDs
    "drug": ["Paracetamol", "Ibuprofen", ...],
    "severity": ["mild", "moderate", "severe", ...],
    "outcome": ["recovered", ...,None, ...,None]}) # 2 NaN
pk.to_csv("u4_pk.csv", index=False)
adr.to_csv("u4_adr.csv", index=False)
with pd.ExcelWriter("u4_data.xlsx") as w:
    pk.to_excel(w, sheet_name="PK", index=False)
    adr.to_excel(w, sheet_name="ADR", index=False)
```

Output u4_pk.csv (20 rows) u4_adr.csv (12 rows, 2 NaN) u4_data.xlsx sheets PK + ADR

File layout

`u4_pk.csv` holds the concentration table. `u4_adr.csv` holds ADR reports. `u4_data.xlsx` bundles both as sheets *PK* and *ADR* via openpyxl.

Remember

Missing outcomes are deliberate teaching gaps, not mistakes – pages 10–11 clean them step by step.

§ 4.2 READ_CSV + READ_EXCEL

2 Reading CSV and Excel

Synthetic teaching data – generated with seeded numpy for classroom practice.

read_files.py

```
# Read the files we just saved
pk2 = pd.read_csv("u4_pk.csv")
adr2 = pd.read_excel("u4_data.xlsx", sheet_name="ADR")
print(pk2.shape, adr2.shape)
```

Output (20, 3) (12, 4) | PK: patient, time_hr, conc_ng_ml ADR: + drug, severity, outcome

The two readers side by side

`read_csv(path)` parses comma text – fastest for PK exports from instruments. `read_excel(path, sheet_name=...)` opens workbook sheets – standard for ADR registers. Both return a *DataFrame*; check `.shape` immediately.

Exam Trap

read_excel needs the openpyxl engine installed. Forgetting sheet_name silently reads only the first sheet – always name it explicitly.

§ 4.3 FIRST LOOK - head tail info

Preview rows, check structure

Synthetic teaching data - generated with seeded numpy for classroom practice.

first_look.py

```
# Peek at the top, bottom and structure
print(pk.head(3)) # first 3 rows
print(pk.tail(3)) # last 3 rows
pk.info() # dtype + non-null counts
```

Output head: P01 0.5h 646.2, 1.0h 565.0, 2.0h 399.1 | tail: P05 1.0h 535.6, 2.0h 374.6, 4.0h 175.6

Output info PK: 20 entries, 3 cols, float64 x2 + object x1, 612 bytes | ADR: 12 entries, outcome 10 non-null

Reading info

Non-Null Count below the total = missing values. outcome shows 10 of 12 - the 2 NaN we planted. dtypes tell you which columns can do math.

§ 4.3 DESCRIBE STATISTICS

describe - one-line summary statistics

Synthetic teaching data - generated with seeded numpy for classroom practice.

describe_pk.py

```
print(pk.describe().round(2))
print(adr["severity"].value_counts())
```

Output conc_ng_ml: count 20, mean 448.51, std 183.63, min 163.90, 50% 459.80, max 727.80

Output severity value_counts: mild 4, moderate 4, severe 4 (balanced teaching set)

How to read describe

mean centres the curve, *std* spreads it, *min / max* bound it, *25% / 50% / 75%* split it in quartiles. count below 20 would itself flag missing concentrations.

Pharma Read

Mean conc 448 with max 728 fits a decaying PK profile; median 460 near the mean means no wild outlier batch. Examiners ask: what does 50% mean? Answer: the median.

§ 4.4 FIND – isna

3 Finding Missing Values

Synthetic teaching data – generated with seeded numpy for classroom practice.

Definition

Missing values appear as `NaN` (Not a Number). `isna()` maps every cell to True/False; `.sum()` collapses that map into a per-column count of gaps. Find first, fix second.

find_na.py

```
# BEFORE: count the gaps
print(adr.isna().sum())
```

Output patient 0 drug 0 severity 0 outcome 2 | only outcome has gaps

Professor's Note

Always run `isna().sum()` BEFORE cleaning and print it. In the exam it is your evidence that you diagnosed before treating – one line, one mark.

§ 4.4 FIX - dropna fillna

Two fixes - drop rows or fill values

Synthetic teaching data - generated with seeded numpy for classroom practice.

fix_na.py

```
# AFTER option A: drop gappy rows (12 -> 10 rows)
clean_drop = adr.dropna()
# AFTER option B: keep rows, label the gap
clean_fill = adr.copy()
clean_fill["outcome"] = clean_fill["outcome"].fillna("under review")
```

Output dropna: 10 rows survive (P07, P12 removed) | fillna: 12 rows, P07 + P12 outcome = under review

Which fix when?

dropna when gaps are few and rows expendable - simplest for analysis. *fillna* when every report counts - ADR registers must not lose cases. Never fill concentrations with 0; it fakes a measurement.

Warning

dropna without assignment changes nothing - it returns a new frame. Either assign it or pass inplace=True deliberately.

§ 4.5 FILTERING ROWS

4 Filtering with Conditions

Synthetic teaching data – generated with seeded numpy for classroom practice.

filter_pharma.py

```
# Boolean mask keeps only matching rows
severe = adr[adr.severity == "severe"] # 4 rows
high = pk[pk.conc_ng_ml > 500] # 9 rows above threshold
```

Output severe: P03 PCM, P05 IBU, P07 AMX, P12 PCM (4 rows)

Output conc > 500: 9 rows, all at 0.5h–1.0h (early timepoints only)

Mask anatomy

`adr.severity == 'severe'` builds a True/False mask; `adr[mask]` keeps True rows. Combine with *& (and) / | (or)*, each test in brackets: `(a) & (b)`.

Pharma Read

Severe-only frames feed safety reviews; conc-above-threshold frames flag the absorption window. Same one-line pattern, two different decisions.

PHARMACY • DATA • AI • HEALTHCARE

§ 4.5 GROUPBY AGGREGATION

5 Groupby: Split-Apply-Combine

Synthetic teaching data – generated with seeded numpy for classroom practice.

group_pharma.py

```
# Mean conc per timepoint + ADR counts per drug
print(pk.groupby("time_hr")["conc_ng_ml"].mean().round(1))
print(adr.groupby("drug").size())
```

Output mean conc: 0.5h 650.9, 1.0h 557.0, 2.0h 398.5, 4.0h 187.7 (decay curve)

Output ADR counts: Amoxicillin 3, Ibuprofen 4, Paracetamol 5

Split - Apply - Combine

Split rows by key (time, drug). *Apply* one function (mean, size, count). *Combine* results into a small summary frame. Two keys give two-level summaries: `groupby(["drug","severity"])`.

Remember

groupby + mean needs a numeric column; groupby + size just counts rows and works on any column.

§ 4.6 MASTER DEMO PIPELINE

ADR summary pipeline - one function, full workflow

Synthetic teaching data - generated with seeded numpy for classroom practice.

adr_pipeline.py

```
def adr_summary(df):
    # clean -> group -> sort, all inside one function
    df = df.copy()
    df["outcome"] = df["outcome"].fillna("under review")
    out = df.groupby(["drug", "severity"]).size().reset_index(name="n")
    return out.sort_values(["drug", "n"], ascending=[True, False])
print(adr_summary(adr).to_string(index=False))
```

Output IBU mild 2 on top of its group PCM moderate 2, severe 2 | 9-row ranked summary

Trace the pipeline

1 *copy* protects the original. 2 *fillna* labels 2 gaps. 3 *groupby + size* counts each drug-severity cell. 4 *reset_index* turns the result back into a table. 5 *sort* ranks worst-first per drug.

Exam Gold

This function chains every Unit-IV skill: clean, filter-ready, group, sort. Reproduce it from memory and the 5-mark pipeline question is yours.

§ 4.7 ERRORS TO AVOID

6 Two Errors, Zero Panic

err_key.py

```
# KeyError: wrong column name
try:
    adr["dosage"]
except KeyError as e:
    print("KeyError:", e) # KeyError: 'dosage'
```

Output KeyError: 'dosage' | fix: print(df.columns) first, copy the exact name

KeyError - the typo tax

Column names are *case- and spelling-sensitive*: `"Severity"` is not `"severity"`. Diagnosis: `print(adr.columns.tolist())` and copy-paste. Prevention: never retype a column name from memory.

SettingWithCopyWarning

Chained assignment like df[df.x > 0]['y'] = 1 may edit a temporary copy, not your frame. Fix with .loc: df.loc[df.x > 0, 'y'] = 1 - one explicit step, no ambiguity, no warning.

§ 4.8 METHOD REFERENCE

Every Unit-IV method on one card

Method	What it does	Pharma example
Series(list, index)	one labelled column	`Series(doses, P-ids)`
DataFrame(dict)	dict of lists to table	`DataFrame(adr_dict)`
read_csv / read_excel	load file to frame	`read_csv('u4_pk.csv')`
head(n) / tail(n)	preview ends	`pk.head(3)`
info() / describe()	structure / stats	`pk.describe()`
isna().sum()	count gaps per col	`adr.isna().sum()`
dropna / fillna(v)	drop / fill gaps	`fillna('under review')`
df[mask]	filter rows	`adr[sev=='severe']`
groupby + agg	split-apply-combine	`groupby('drug').size()`

How to Memorise

Read each row as verb-object-target: Series labels, DataFrame tables, readers load, head previews, info diagnoses, cleaners fix, masks filter, groupby summarises.

§ DIVE DEEP-DIVE EXAMPLES

Three Demos, Real Output

Synthetic teaching data – generated with seeded numpy for classroom practice.

D1 · describe row guide

dive_describe.py

```
# D1: what each describe() row tells - real PK frame
print(pk["conc_ng_ml"].describe().round(2))
```

Output count 20 = no gaps | mean 448.51 centre | std 183.63 spread | 50% 459.80 median | min 163.90 max 727.80 bounds

D2 · median vs mean fill

dive_fillna.py

```
# D2: knock out P03 2.0h (was 400.1), then choose the fill
pk2 = pk.copy(); pk2.loc[10, "conc_ng_ml"] = float("nan")
med = pk2.conc_ng_ml.median() # 492.4
avg = pk2.conc_ng_ml.mean() # 451.06
clean = pk2.conc_ng_ml.fillna(med) # 0 gaps left
```

Output BEFORE isna 1 (P03 2.0h knocked out) | median fill 492.4 | mean fill 451.06 | AFTER 0 gaps

D3 · multi-aggregation

dive_multiagg.py

```
# D3: centre AND peak per timepoint in one pass
print(pk.groupby("time_hr")["conc_ng_ml"].agg(["mean", "max"]).round(1))
```

Output 0.5h mean 650.9 max 727.8 | 1.0h 557.0 627.8 | 2.0h 398.5 427.2 | 4.0h 187.7 213.9

Dive Rules

count = coverage, mean = centre, 50% = median, min/max = bounds. Low 4.0h troughs drag the mean (451.06) below the median (492.4) – skewed tails mean fill with median, never 0. `agg(['mean','max'])` gives centre + peak together.

§ EXAM LIKELY QUESTIONS

7 Exam Points

Q1 – Series vs DataFrame (2m)

Series is 1-D labelled `pd.Series(doses)`; DataFrame is 2-D `pd.DataFrame(d)`. Quote dose list vs ADR table.

Q2 – read_csv vs read_excel (2m)

`read_csv('u4_pk.csv')` for text CSV; `read_excel(x, sheet_name='ADR')` for workbooks. Always verify with `.shape`.

Q3 – head/info/describe (3m)

`head()` previews, `info()` shows dtypes + non-null counts (outcome 10/12), `describe()` gives mean 448.51, max 727.80.

Q4 – isna/dropna/fillna (3m)

`isna().sum()` found outcome 2; `dropna()` 12->10 rows; `fillna('under review')` keeps all 12.

Q5 – filter + groupby pipeline (5m)

Filter `adr[sev=='severe']` (4 rows); group `groupby('time_hr').mean()` decay 650.9->187.7; reproduce `adr_summary()`.

§ REVISE REVISION GRID

One grid – the whole unit in 60 seconds

Stage	Key call	One-line recall
Build	`Series / DataFrame`	labels first, values second
Load	`read_csv/excel`	shape-check every load
Explore	`head/info/describe`	preview, diagnose, summarise
Clean	`isna/dropna/fillna`	count, then drop or fill
Filter	`df[mask]`	mask True keeps the row
Summarise	`groupby.agg`	split, apply, combine
Deliver	`adr_summary()`	clean-group-sort pipeline

60-second recall drill

Cover the right column. Say each recall line aloud: *labels first*, *shape-check*, *preview-diagnose-summarise*, *count then fix*, *True keeps*, *split-apply-combine*. Seven sentences = full unit.

Professor's Note

Revise in pipeline order, not alphabetical order. The exam tells the same story: build, load, explore, clean, filter, summarise.

§ CO-PO CO4 MAPPING

Where Unit-IV sits in your course outcomes

CO	Statement	Unit-IV evidence
CO4	Handle experimental and clinical data with Python	PK + ADR pipelines end-to-end
CO4.1	Build Series/DataFrame objects	dose Series, ADR mini-frame
CO4.2	Read CSV and Excel sources	u4_pk.csv, u4_data.xlsx
CO4.3	Clean missing values correctly	isna 2 gaps, drop/fill fixes
CO4.4	Filter and aggregate for decisions	severe filter, groupby means

PO linkage

CO4 feeds **PO1 (pharmaceutical knowledge)** via real PK/ADR reasoning, **PO2 (analysis)** via describe + groupby interpretation, and **PO9 (lifelong data skills)** via reproducible seeded workflows.

Bloom levels in this unit

Remember: method names and signatures. **Understand**: Series vs DataFrame, mask logic. **Apply**: clean + filter real tables. **Analyze**: groupby summaries that drive safety calls.

File This

Attach the adr_summary pipeline printout as CO4 evidence in your lab record – it demonstrates every CO4 sub-outcome on one page.

§ CHECK SELF-CHECK + SUMMARY

Five questions – honest answers only

Question (try first)	Answer (check after)
1. Series([100,250]) with index [P01,P02]: what does s['P02'] return?	250 – labels select values.
2. adr has 12 rows, outcome 2 NaN: what does dropna leave? What does fillna keep?	dropna: 10 rows; fillna: all 12 with labels.
3. Write the one-liner that keeps only severe ADR rows.	adr[adr.severity == 'severe'] – mask then select.
4. groupby('time_hr').mean() on PK gives 650.9, 557.0, 398.5, 187.7: rising or falling?	Falling – classic decay curve.
5. adr['dosage'] raises KeyError: what is the first debugging command?	print(adr.columns.tolist()) – copy the exact name.

Unit-IV in One Line

Pandas turns PK tables and ADR reports into labelled DataFrames you read, explore, clean, filter and group – one reproducible pipeline from raw file to safety summary.

§ FINAL PINPOINTS - MUST REMEMBER

Pinpoints - Must Remember

#	Hook	Recall in one line
1	`head/info/describe`	preview rows, diagnose types + gaps, summarise stats
2	`isna().sum()`	count gaps per column BEFORE any fix - outcome 2
3	`dropna vs fillna`	drop 12->10 rows; fill keeps 12 as 'under review'
4	`boolean mask`	df[mask] keeps True rows; combine with (a) & (b)
5	`loc vs iloc`	loc takes labels like P03; iloc takes positions 0,1,2
6	`groupby`	split-apply-combine: mean per time, size per drug
7	`read_csv/excel`	CSV is fastest; Excel needs openpyxl + sheet_name
8	`dtypes`	numeric columns do math; object columns do not
9	`.shape check`	verify every load: PK (20,3), ADR (12,4)
10	`value_counts`	severity audit 4-4-4: a balanced teaching set

3 exam predictions

P1 - 5-mark pipeline: reproduce `adr_summary()` from memory - clean, group, sort.
P2 - 3-mark describe: quote mean 448.51, median 459.80, max 727.80; say what 50% means.
P3 - 2-mark debug: KeyError on `'dosage'` - first command `print(adr.columns.tolist())`.

Mnemonic - the 5 L's

Load, Look, Lose-gaps, Limit, Lump: Load files, Look with head/info/describe, Lose gaps via dropna/fillna, Limit rows with a mask, Lump rows with groupby. Five L's, five calls, full unit.