Practice: Data Gathering and Cleaning

Python and SQL. Work through the tasks in order.

Part 1 — Gathering

Data comes in different formats. You need a different reader for each one. That is the whole idea of this part.

Source	Format	How you read it
World Bank API	JSON	`urllib + json.loads()`
IMF API	JSON	`urllib + json.loads()`
World Bank bulk	CSV in a ZIP	`zipfile + csv.DictReader`
Eurostat	TSV	`split on tab`
Wikipedia	HTML table	`pandas.read_html()`
Workbook	Excel .xlsx	`openpyxl.load_workbook()`
Agency database	SQL	`sqlite3 + SELECT`

Task 1.1 — Download with Python

Make a folder, then write a small function that fetches a URL and saves it under a name you choose:

```
import urllib.request, os

os.makedirs("staging_multi", exist_ok=True)
os.makedirs("sources", exist_ok=True)

UA = "DWH course exercise (student@example.edu)"

def download(url, filename):
    req = urllib.request.Request(url, headers={"User-Agent": UA})
    with urllib.request.urlopen(req, timeout=60) as r:
        data = r.read()
    with open(filename, "wb") as f:
        f.write(data)
    print("saved", filename, len(data), "bytes")
```

Call it once per source. The second argument is the file name — use these exactly, because the cleaning code opens them by name:

```
download("https://www.imf.org/external/datamapper/api/v1/NGDP_RPCH",
         "staging_multi/s2_imf_datamapper.json")

download("https://ec.europa.eu/eurostat/api/dissemination/sdmx/2.1/"
         "data/sdg_08_10?format=TSV",
         "staging_multi/s4_eurostat.tsv")

download("https://en.wikipedia.org/wiki/"
         "List_of_sovereign_states_by_central_bank_interest_rates",
         "staging_multi/s5_wiki_rates.html")

download("https://dwhdemo.vedara.net/docs/oecd_style_outlook.xlsx",
         "staging_multi/s6_oecd_outlook.xlsx")

download("https://dwhdemo.vedara.net/docs/statagency_oltp.sqlite",
         "sources/statagency_oltp.sqlite")
```

The User-Agent matters. Wikipedia refuses requests that do not say who is asking (“403 Forbidden”), and the IMF refuses short invented ones like `"student/1.0"` and browser impersonation like `"Mozilla/5.0"`. A plain sentence with a contact address, as above, is accepted by both. Put your own address in it.

Task 1.2 — The zipped CSV

One source arrives as a ZIP. Open it in memory and save only the CSV you need:

```
import urllib.request, zipfile, io, os

UA = "DWH course exercise (student@example.edu)"
os.makedirs("staging_multi", exist_ok=True)

url = ("https://api.worldbank.org/v2/en/indicator/"
       "NY.GDP.PCAP.PP.CD?downloadformat=csv")
req = urllib.request.Request(url, headers={"User-Agent": UA})
blob = urllib.request.urlopen(req, timeout=60).read()
z = zipfile.ZipFile(io.BytesIO(blob))

name = [n for n in z.namelist() if n.startswith("API_")][0]
text = z.open(name).read().decode("utf-8-sig")

with open("staging_multi/s3_worldbank_bulk.csv", "w", encoding="utf-8") as f:
    f.write(text)

print(len(text.splitlines()), "lines")  # should print 270
```

The ZIP holds three CSVs; you want the one starting with `API_`. Its first four lines are not data.

Task 1.3 — An API that sends pages

The World Bank API returns 50 rows at a time. Ask for more, and loop until you have every page:

```
import urllib.request, json, os

UA = "DWH course exercise (student@example.edu)"
os.makedirs("staging", exist_ok=True)

rows, page = [], 1
while True:
    url = ("https://api.worldbank.org/v2/country/all/indicator/"
           "NY.GDP.MKTP.CD?format=json"
           f"&per_page=1000&page={page}")
    req = urllib.request.Request(url, headers={"User-Agent": UA})
    body = json.loads(urllib.request.urlopen(req, timeout=60).read())
    rows += body[1]
    if page >= body[0]["pages"]:
        break
    page += 1

print(len(rows), "rows")
json.dump(rows, open("staging/ind_NY.GDP.MKTP.CD.json", "w"))
```

`body[0]` tells you how many pages there are; `body[1]` is the data.

Task 1.4 — The SQL source

The agency database spreads the data over four tables. Join them and save the result as a CSV:

```
import sqlite3, csv, os

os.makedirs("staging_multi", exist_ok=True)
db = sqlite3.connect("sources/statagency_oltp.sqlite")
rows = db.execute("""
    SELECT c.iso_alpha3, c.country_nm, c.continent_cd,
           m.measure_cd, m.uom_cd,
           p.period_yr, p.is_final,
           o.obs_val, o.status_cd, o.updated_ts
    FROM observation o
    JOIN country c ON c.country_id = o.country_id
    JOIN measure m ON m.measure_id = o.measure_id
    JOIN period  p ON p.period_id  = o.period_id
""").fetchall()

print(len(rows), "rows")  # should print 752

cols = ["iso3", "country_nm", "continent", "measure_cd", "uom",
        "year", "is_final", "value", "status", "updated_ts"]
with open("staging_multi/s7_statagency_extract.csv", "w", newline="") as f:
    w = csv.writer(f)
    w.writerow(cols)
    w.writerows(rows)
```

Keep `status_cd` and `updated_ts` in the extract — Task 2.3 needs them to drop deleted rows and pick the newest correction.

Each block above repeats the `UA` line so you can run any one of them on its own. In a real script you would define it once.

Get the scripts first. The ready-made ETL scripts live at `dwhdemo.vedara.net/docs/etl/` — extract.py, extract_multi.py, clean_consolidate.py, make_source_fixtures.py. Download them into a folder called `etl/` next to your data.

On Google Colab

Everything here works in a Colab notebook unchanged. Run this in the first cell to fetch the scripts, then use the same commands:

```
import urllib.request, os
os.makedirs("etl", exist_ok=True)
UA = "DWH course exercise (student@example.edu)"
for f in ["extract.py", "extract_multi.py",
          "clean_consolidate.py", "make_source_fixtures.py"]:
    req = urllib.request.Request(
        "https://dwhdemo.vedara.net/docs/etl/" + f,
        headers={"User-Agent": UA})
    open("etl/" + f, "wb").write(
        urllib.request.urlopen(req, timeout=60).read())
```

Colab already has pandas, openpyxl and lxml. Files you create are lost when the session ends, so download `consolidated_long.csv` before you close it — you need it next week.

Shortcut

All of this is already written. To skip ahead, or to check your version against it:

```
python3 etl/extract_multi.py staging_multi sources
```

All seven sources at a glance

#	Saved as	URL	Format
S1	**staging/ ind_NY.GDP.MKTP.CD.json**	api.worldbank.org/v2/country/all/ indicator/NY.GDP.MKTP.CD?format=json	JSON
S2	**staging_multi/ s2_imf_datamapper.json**	www.imf.org/external/datamapper/ api/v1/NGDP_RPCH	JSON
S3	**staging_multi/ s3_worldbank_bulk.csv**	api.worldbank.org/v2/en/indicator/ NY.GDP.PCAP.PP.CD?downloadformat=csv	CSV in ZIP
S4	**staging_multi/ s4_eurostat.tsv**	ec.europa.eu/eurostat/api/dissemination/ sdmx/2.1/data/sdg_08_10?format=TSV	TSV
S5	**staging_multi/ s5_wiki_rates.html**	en.wikipedia.org/wiki/List_of_sovereign_ states_by_central_bank_interest_rates	HTML
S6	**staging_multi/ s6_oecd_outlook.xlsx**	dwhdemo.vedara.net/docs/ oecd_style_outlook.xlsx	Excel
S7	**sources/ statagency_oltp.sqlite**	dwhdemo.vedara.net/docs/ statagency_oltp.sqlite	SQL

All URLs start with `https://`. S7 is the database itself — your SQL query writes `staging_multi/s7_statagency_extract.csv`.

Rule: save the downloaded file first, clean it afterwards. If your cleaning code has a bug, you fix the code — you do not download everything again.

Part 2 — Cleaning

Each source is messy in its own way. Fix each one separately. The goal is that every source ends up in the same simple shape:

```
(source, country, indicator, year, value)
```

The five problems you will meet

#	Problem	Looks like	What to do
1	Missing values	`: .. null (empty)`	Leave it out. Do not write 0.

#	Problem	Looks like	What to do
2	Text stuck to numbers	`45390 p 1,234.5 (e)`	Strip the letters and commas, keep the number.
3	Different units	`GDP in billions vs units`	Convert to one unit before comparing.
4	Different country codes	`USA US EL UK`	Map everything to ISO3 (EL=GRC, UK=GBR).
5	Deleted / repeated rows	`status_cd = 'D'`	Skip deleted rows. If a row repeats, keep the newest.

Task 2.1 — Clean the Excel file

Open `sources/oecd_style_outlook.xlsx` in a spreadsheet program first and look at it. The real table does not start at row 1: there is a title, a merged header, and notes at the bottom. Write code that:

1. finds the row where the header actually starts
2. stops before the notes at the bottom
3. turns "1,234.5 (e)" into the number 1234.5
4. skips cells containing ".."

Task 2.2 — Clean the Eurostat file

Open `staging_multi/s4_eurostat.tsv`. The first column holds four values joined by commas; the country code is the last one. Split it, map the code to ISO3, and skip any cell that is `:`.

Task 2.3 — Clean the SQL extract

Some rows are marked deleted, and some countries have the same year twice because the number was corrected later. Write SQL that returns one row per country/measure/year:

```
SELECT ... FROM observation o
WHERE o.status_cd <> 'D'
  AND o.updated_ts = (SELECT MAX(o2.updated_ts)
                      FROM observation o2
                      WHERE o2.country_id = o.country_id
                        AND o2.measure_id = o.measure_id
                        AND o2.period_id  = o.period_id);
```

The table has 752 rows. You should keep 715 and remove 37 (25 deleted + 12 corrected duplicates).

Task 2.4 — Run the full cleaner and check your numbers

```
python3 etl/clean_consolidate.py staging staging_multi build/consolidated
```

These are the numbers from the full script, which fetches four IMF indicators. If you only did the one in Task 1.1, your S2 row will be smaller — that is expected. The other six rows should match exactly.

Source	Clean rows	What had to be fixed
S1	39,042	some countries had no code
S2	24,083	years before 2000 removed; future years marked
S3	6,352	empty year columns skipped
S4	1,854	':' skipped; quality letters removed
S5	41	footnotes removed; names matched to codes
S6	350	footnote markers removed
S7	715	deleted rows skipped; repeats reduced to newest
Total	**72,437**	

Task 2.5 — Break it on purpose

In the Excel cleaner, remove the step that strips "(e)" and ",". Run it again. What happens? Now put that back and instead remove the step that converts billions to units. Does anything crash?

Point of this task: the first mistake makes the program fail, so you notice it. The second one gives you numbers that are a billion times too small, and the program runs fine. The second kind is more dangerous.

Part 3 — Putting the sources together

When all sources are clean, put them in one table and look for rows where two sources describe the same country, indicator and year.

Clean rows in total	Countries	Indicators	Covered by 2+ sources	Where they disagree
72,437	289	10	14,134	9,838

Task 3.1 — Find a disagreement

Pick one country and one indicator where two sources give different numbers. Write down both values and the difference. You do not have to decide which one is right — just show that they differ.

Task 3.2 — Check one number three ways

Look up US GDP for 2023 in the API data, in the Excel file, and in the SQL database. All three should give about 27.8 trillion USD. If one of them does not, your unit conversion is wrong.

Next class

Transformation. We take the clean table and reshape it for analysis: choose which source wins when two disagree, build the dimension and fact tables, and load them. Nothing in this practice needs to be redone — we continue from the clean table you produced today.

Data: World Bank, IMF, Eurostat, Wikipedia. The Excel workbook and the SQL database are practice files made for this course — download them from the addresses in Part 1. A finished version of this warehouse is at dwhdemo.vedara.net.