Data Preparation — Topper-Style Notes

- **Why**: Raw data is messy; prep ensures correctness before analysis.

Excel Data Cleansing

- **Find and Replace**: Use Ctrl+H to remove unwanted text (e.g., “[more]” from entries)
- **Data Format Conversion**: Change columns from general to numerical format
- **Text Cleaning**: TRIM function to remove extra spaces
- **Blank Cell Removal**: “Go To Special” function to identify and delete empty rows
- **Duplicate Removal**: Built-in “Remove Duplicates” feature
- **Functions**: `TRIM()`, `CLEAN()`, Data Validation for consistency
- **Pitfalls**: Non-breaking spaces, inconsistent casing, hidden characters
- **Checklist**: Apply TRIM/CLEAN systematically; set validation rules; audit for duplicates

General Cleaning Principles

- Handle missing (NA/NULL/blank), outliers, inconsistent categories.
- Convert types (string → date/number); unify encodings (UTF-8).
- Tools: OpenRefine, pandas, DuckDB, shell (`sed`, `awk`, `grep`).

OpenRefine (Entity Resolution)

- **Purpose**: Interactive data cleaning with clustering algorithms
- **Why**: Resolve entity discrepancies and merge similar entries
- **Core Process**:
    - **Data Upload**: Import data and create new project
    - **Text Faceting**: Group similar entries and analyze frequency
    - **Clustering**: Apply algorithms to merge entries with minor differences
    - **Resolution Options**: Manual clustering vs automated batch processing

- **Key Features**: Punctuation normalization, case standardization, entity deduplication
- **Workflow**: Upload → Facet → Cluster → Merge → Export
- **Use Cases**: Address standardization, company name normalization, product categorization
- **Advantages**: Visual interface, undo/redo, clustering algorithms
- **Pitfalls**: Manual intensive for large datasets; requires domain knowledge
- **Checklist**: Review clustering suggestions; validate merges; document decisions; export clean data

Excel Data Transformation

- **Ratio Calculations**: Compute derived metrics (metro/city ratios)
- **Pivot Tables**: Aggregate data and identify outliers
- **Filtering**: Apply filters to analyze specific data subsets
- **Counting Occurrences**: Use pivot tables for frequency analysis
- **Chart Creation**: Generate visualizations from pivot table data
- **Functions**: Calculated fields, SUMIFS, XLOOKUP over VLOOKUP
- **Examples**:

```
 =TEXTSPLIT(A2, ", ")  # Split delimited text
=XLOOKUP(E2, customers[Email], customers[Name], "Not found")
=SUMIFS(Sales[Amount], Sales[Region], "US", Sales[Month], H2)
```

Shell Transformations

- **UNIX Tools**: `curl` / `wget` for downloads, `gzip` for compression, `wc` for counting
- **Text Processing**: `head` / `tail` , `cut` , `uniq` , `sort` , `grep` , `sed` , `awk`
- **Advantages**: Agile (quick exploration), Fast (C-based, parallelizable), Popular (universal support)
- Shell quick wins:

```
 # Extract column 1 (space-delimited), sort, count unique top 10
cut -d' ' -f1 access.log | sort | uniq -c | sort -n | tail
# Replace [datetime] with "datetime"
sed 's/\[\([^]]*\)\]/"\1"/' access.log > log.csv
```

```
# Download with resume capability
curl --continue-at - --location --output file.gz URL
```

- Pandas:

```
 import pandas as pd
df[['first','last']] = df['full_name'].str.split(' ', 1, expand=True)
```

Excel Data Aggregation

- **Data Cleanup**: Remove empty columns and rows with missing values
- **Excel Tables**: Convert raw data to tables for easier manipulation
- **Date Functions**: Extract week/month/year using WEEKNUM, TEXT functions
- **Visualization**: Color scales, sparklines, data bars for trend analysis
- **Pivot Tables**: Aggregate by location and date, weekly/monthly summaries
- **Examples**:

```
 =WEEKNUM(A2)  # Extract week number
=TEXT(A2, "yyyy-mm")  # Format date as year-month
```

SQL Aggregating

- GroupBy + aggregate; window functions (SQL) for moving averages.
- DuckDB patterns:

```
 -- Skip bad rows, then summarize
SELECT * FROM read_csv_auto('messy.csv', ignore_errors=true);
SELECT region, COUNT(*) n, SUM(amount) total FROM orders GROUP BY region;
```

Profiling

- Head/tail, describe, value_counts; null and duplicates report.
- ydata-profiling, Great Expectations for data tests.

DuckDB Data Preparation

- **Purpose**: High-performance SQL for large-scale data cleaning
- **Why**: Memory-efficient processing of files too large for traditional tools
- **File Format Support**: `read_csv_auto`, `read_json_auto`, `read_parquet`; export to JSON/Parquet
- **Error Handling**: `ignore_errors=true` to skip malformed rows
- **Missing Values**: `COALESCE(customer, 'Unknown')` for NULL replacement
- **String Operations**:

```
 SELECT DISTINCT TRIM(LOWER(product)) AS clean_product FROM orders;
SELECT REGEXP_REPLACE(product, '\\s+', ' ', 'g') AS tidy_product FROM order
```

- **Date Processing**:

```
 SELECT order_id, STRFTIME(order_date, '%Y-%m') AS order_month FROM orders;
```

- **Conditional Logic**: CASE statements for data binning and categorization
- **Chunking Large Files**:

```
SELECT * FROM read_csv_auto('big.csv') LIMIT 1000 OFFSET 0;
```

- **Multi-format Integration**: Combine CSV, JSON, and Parquet in single queries
- **Derived Columns and Pivots**:

```
 SELECT *, amount * 0.1 AS tax, UPPER(region) AS region_code FROM orders;
SELECT * FROM orders PIVOT(COUNT(*) FOR region IN ('US','EU'));
```

- **Business Applications**: E-commerce data cleaning, transaction processing, inventory analysis

Comparison Table

Tool	Strengths	Weaknesses	Best for
Shell	Fast, composable	Regex quirks, portability	Logs, quick fixes

Tool	Strengths	Weaknesses	Best for
OpenRefine	Interactive clustering	Manual steps	Entity resolution
DuckDB	Big-file SQL, formats	SQL learning curve	Batch cleaning, joins
Pandas	Pythonic transforms	Memory-bound	Notebooks, small-mid data

Video takeaways

- Shell: pipelines are powerful (`cut|sort|uniq|sort`).
- DuckDB: skip bad rows, handle large files, export Parquet/JSON.
- OpenRefine: facets + clustering for near-duplicate cleanup.

Checks and tips

- Explicitly parse dates and numeric types early; fail fast on mixed formats.
- Treat NULLs intentionally with `COALESCE`; document imputations.
- Validate CSVs against RFC 4180 if vendors are inconsistent.
- Prefer window functions for rolling stats over manual loops.
- Sample with stratification when class imbalance exists.

Advanced theory and tricky exam asks

- **Type systems**: Strings vs numeric vs datetime; implicit casts hide errors; prefer explicit parsing with formats.
- **NULL semantics**: `NULL != 0` and `NULL != ''`; aggregations skip NULLs; use `COALESCE` intentionally.
- **CSV quoting**: RFC 4180 requires doubling quotes inside cells; watch for vendors using backslash-escaping; ensure consistent delimiters.
- **Regex pitfalls**: Greedy vs non-greedy; multiline flags; escaping in shell vs SQL vs Python differs.
- **Window functions**: Moving averages, ranks, partitions—prefer for time-series prep.
- **Sampling bias**: Downstream models inherit biases from non-representative samples; stratify and document filters.

Likely exam asks:

- Why `NULL` handling changes aggregation outputs and how to fix.
- Show a robust date parser across mixed formats.
- Convert malformed CSV with embedded quotes into valid RFC 4180 CSV.

Deep dive details

Date parsing variants:

```
import pandas as pd
# Mixed formats: day-first and ISO
s = pd.Series(['2024-12-03', '03/12/2024', '12/03/24'])
dates = pd.to_datetime(s, dayfirst=True, errors='coerce')
```

RFC 4180 quoting example:

```
id,comment
1,"He said ""hello"" to her"
```

Regex greedy vs non-greedy:

```
import re
text = '<p>one</p><p>two</p>'
re.findall(r'<p>.*</p>', text)     # ['<p>one</p><p>two</p>'] (greedy)
re.findall(r'<p>.*?</p>', text)    # ['<p>one</p>', '<p>two</p>'] (non-gree
```

Window function example:

```
SELECT date,
       amount,
       AVG(amount) OVER (ORDER BY date ROWS BETWEEN 6 PRECEDING AND CURRENT
FROM sales;
```

Sampling strategies:

- Simple random: unbiased but high variance.
- Stratified: maintain proportions across groups.
- Systematic: every k-th record; beware periodicity artifacts.
- Weighted: oversample rare classes; reweight during analysis.

Excel transformations

- Purpose: fast cleaning without code.
- Why: common, collaborative, visual.
- Core: TEXTSPLIT/TEXTAFTER, XLOOKUP over VLOOKUP, SUMIFS, Tables.
- Examples:

```
 =TEXTSPLIT(A2, ", ")
=XLOOKUP(E2, customers[Email], customers[Name], "Not found")
=SUMIFS(Sales[Amount], Sales[Region], "US", Sales[Month], H2)
```

- Pitfalls: mixed types; hidden whitespace; volatile arrays.
- Checklist: convert ranges → Tables; consistent number/date formats; document formulas.

Spreadsheet cleansing

- Purpose: remove noise and duplicates.
- Why: prevent downstream errors.
- Core: TRIM, CLEAN, Remove Duplicates, Data Validation.
- Pitfalls: non-breaking spaces; inconsistent casing.
- Checklist: apply TRIM/CLEAN; set validation; audit duplicates.

SQL pipelines (dbt-style)

- Purpose: declarative, testable transforms.
- Why: versioned, repeatable, documented.
- Core: models (SELECTs), tests, snapshots, materializations.
- Pitfalls: silent type changes; unintended cross-joins.
- Checklist: tests on nulls/unique/relationships; incremental where possible; document lineage.

Python profiling/validation

- Purpose: rapid EDA and guardrails.
- Why: catch schema drift early.

- Core: automated profiles and declarative checks (types, ranges, nulls, uniques).
- Pitfalls: profiling on PII; long runtimes on big data.
- Checklist: sample first; redact sensitive columns; persist reports.

Editor-driven prep (VS Code)

- **Purpose**: Speed repetitive edits with visual feedback
- **Why**: Quick feedback loops for data cleaning tasks
- **Core Features**:
    - **JSON Formatting**: Auto-format JSON files for readability
    - **Find All + Multiple Cursors**: Extract specific fields simultaneously
    - **Line Operations**: Sort lines, delete duplicates
    - **Text Replacement**: Multi-cursor text replacement
- **Workflow**: Format → Find All → Multi-cursor edit → Sort → Deduplicate
- **Use Cases**: JSON field extraction, text normalization, batch edits
- **Pitfalls**: Overbroad regex; accidental global changes; no undo for bulk operations
- **Checklist**: Preview diffs; narrow scope; backup before bulk edits; test on sample first

JSON Processing

- Purpose: handle nested/large JSON efficiently.
- Why: APIs, logs, config files use JSON extensively.
- Tools:
    - **jq**: command-line JSON processor for quick exploration
    - **JMESPath**: declarative queries in Python
    - **ijson**: streaming for large files
    - **Pandas**: normalize JSON columns
    - **DuckDB**: SQL queries on JSON files
- Examples:

```
 # jq: extract specific fields
cat data.jsonl | jq -c 'select(.type == "user") | {id, name}'
```

```
# JMESPath: filter nested data
import jmespath
cities = jmespath.search("locations[?info.population > `700000`].name", dat
```

```
-- DuckDB: analyze JSON directly
SELECT json_extract_string(data, '$.user.name') as name,
       avg(json_extract_float(data, '$.metrics.value')) as avg_value
FROM read_json_auto('data/*.jsonl')
GROUP BY 1
```

- Pitfalls: memory issues with large files; complex nested structures.
- Checklist: use streaming for large files; validate JSON structure; handle missing keys.

Image Processing

- Purpose: transform images for analysis/ML.
- Why: computer vision, data visualization, preprocessing.
- Tools:
  - **PIL/Pillow**: Python image manipulation
  - **ImageMagick**: command-line batch processing
- Core operations:

```
# Pillow: basic transformations
from PIL import Image, ImageEnhance
img = Image.open('input.jpg').convert('RGB')
img.thumbnail((800, 800))  # Resize maintaining aspect
img = ImageEnhance.Contrast(img).enhance(1.2)
```

```
# ImageMagick: batch operations
convert input.jpg -resize 800x600 -quality 85 output.jpg
mogrify -resize 800x600 -path output/ *.jpg  # Batch resize
```

- Advanced: filters, watermarks, format conversion, metadata extraction.
- Pitfalls: memory usage with large images; format compatibility.
- Checklist: use thumbnail() for memory efficiency; validate formats; batch process systematically.

Text Splitting (Excel)

- Purpose: split single column into multiple organized columns.
- Why: common data cleaning task in spreadsheets.
- Core: Text-to-Columns feature with delimiters.
- Functions: `TEXTSPLIT()`, `TEXTAFTER()`, `TEXTBEFORE()`.
- Examples:

```
 =TEXTSPLIT(A2, ", ")  # Split by comma-space
=TEXTAFTER(A2, "@")   # Extract domain from email
```

- Pitfalls: inconsistent delimiters; embedded quotes.
- Checklist: preview results; handle edge cases; validate split accuracy.

Audio/Video Processing

- Purpose: extract audio, transcripts from media files.
- Why: speech analysis, content extraction, ML training data.
- Tools:
  - **FFmpeg**: media format conversion, audio extraction
  - **yt-dlp**: download audio/video from web
  - **Whisper**: speech-to-text transcription
  - **Gemini**: intelligent transcription with translation/summarization
- Core operations:

```
 # FFmpeg: extract audio for speech recognition
ffmpeg -i input.mp4 -ar 16000 -ac 1 audio.wav
# Extract frames for computer vision
ffmpeg -i input.mp4 -vf "fps=1" frames_%04d.png

# yt-dlp: download audio at speech quality
yt-dlp -f "ba[abr<50]/worstaudio" --extract-audio --audio-format mp3 URL

# Whisper: transcribe to multiple formats
faster-whisper-xxl --model medium --output_format json srt audio.wav
```

- Advanced: batch processing, quality optimization, format conversion.
- Pitfalls: large file sizes; processing time; copyright restrictions.

- Checklist: optimize quality for use case; respect terms of service; validate transcription accuracy.

dbt (Data Build Tool)

- **Purpose**: Transform data in warehouses using SQL with software engineering practices
- **Why**: Version-controlled, testable, documented data transformations
- **Core Concepts**:
  - **Models**: SQL SELECT statements that create tables/views
  - **Tests**: Data quality assertions (not null, unique, relationships)
  - **Snapshots**: Track slowly changing dimensions
  - **Materializations**: Table, view, incremental, ephemeral
- **Workflow**: Raw data → Staging → Intermediate → Marts
- **Features**: Lineage tracking, documentation generation, dependency management
- **Best Practices**:

```
 -- Model example with tests
{{ config(materialized='table') }}
SELECT customer_id, SUM(amount) as total_spent
FROM {{ ref('orders') }}
GROUP BY customer_id
```

- **Testing**: Schema tests, data tests, custom tests for business rules
- **Pitfalls**: Silent type changes; unintended cross-joins; complex dependency chains
- **Checklist**: Test nulls/unique/relationships; use incremental where possible; document lineage; version control all changes