World Salon ML Case Study

World Salon – Machine Learning Engineer Take-Home Technical Exercise

---

0. Purpose & Logistics

We want to see **how you think, code, and communicate** when solving problems that are central to World Salon:

1. **Matching speakers to event topics** (recommendation system).

2. **Automatically enriching our speaker database** with new candidates pulled from public sources (data-pipeline automation).

You may use **any open-source libraries or cloud-free resources** you like. If you rely on paid APIs, stub or document the calls so we can run your code without extra keys.

Please keep total effort reasonable (a long weekend, not a full-time week). Show clear, working prototypes and explain any unfinished edges.

---

1. Deliverables & Submission Format

Item	Where / How
Code (two clearly-separated folders: /recommendation and /pipeline)	Public Git repo **or** ZIP file. Include a requirements.txt or environment.yml.
One-click / one-command run instructions	README.md at project root. We should be able to reproduce your demo on macOS/Linux with Python ≥ 3.9.

Short technical report (report.pdf or .md, ≤ 5 pages)	Explain approach, design choices, assumptions, metrics, and future work. Use plain language; diagrams welcome.
Architecture diagram(s)	Embed in the report or place under /docs.
Sample outputs	For each part, commit example JSON/CSV results or screenshots so we can inspect without running everything.

Push or email everything **before the 72-hour deadline**.

---

2. Part A – Speaker-to-Event Recommendation Prototype

2.1. Data

We provide two CSVs in /data:

- speakers.csv speaker_id, name, title, expertise_topics
- events.csv event_id, event_name, event_topics

Each *_topics column is a comma-separated list of topic words (e.g. **"AI, Healthcare, Privacy"**). Feel free to perform additional cleaning or feature enrichment.

2.2. Task

1. **Data audit & cleaning**

   Summarise obvious issues (duplicates, missing topics, inconsistent casing) and show how you fixed them.

2. **Feature engineering**

   Represent speakers and events so that we can find best match speakers given an

event title - any model type is ok - justify your choice.

3. **Matching algorithm**

   Implement code that, given an event_id, returns the **top 5** ranked speaker_ids with a relevance score.

   Bonus: expose the matcher through a lightweight REST endpoint or CLI.

4. **Evaluation**

   - If the dataset includes historic matches (precision@k, recall, etc.).
   - Otherwise, print the top results for **three** sample events and briefly comment on their plausibility.

     Discuss how you would measure success in production.

2.3. What We're Looking For

Aspect	Indicators of Strength
Correctness	Matching logic clearly stems from features; outputs make intuitive sense.
Clarity	Code is modular and easy to follow; report explains decisions in plain English.
Insight	Shows awareness of cold-start problems, future use of collaborative filtering, or hybrid approaches.
Pragmatism	Solution works without over-engineering; future improvements are sketched realistically.

---

3. Part B – Automated Speaker-Sourcing Data-Pipeline Test

3.1. Scenario

> *"Find highly relevant speakers for a new virtual panel on **'AI for Sustainable Agriculture'**. We want **senior data-science leaders** (Director level or above) with demonstrated expertise in agricultural technology."*

Your mission: **design and prototype an automated pipeline** that ingests such plain-English queries, discovers suitable speaker candidates on the open internet, extracts key details, deduplicates them against our existing database, and stores the new records.

3.2. Minimum Requirements

Step	Details
Input handler	Accept a free-text query (like the example above). Parse the *desired topic* and *speaker persona*.
Data discovery	Programmatically search at least **one** public source (search-engine results, conference sites, professional profiles, etc.).
Extraction	For each new speaker found, capture **name, current title/role, organization, email (if publicly available), short background/bio**, plus any topic/evidence text you scraped (URL, snippet).
Cleaning & validation	Handle missing fields, remove obvious duplicates, normalise casing, etc.
Persistence	Write the cleaned records into a simple SQLite DB **or** a CSV called new_speakers.csv (include header).

Sample run	Hard-code or pass the example query via CLI/arg. Produce at least **3** new speaker rows as proof of concept.
Design outline for scaling	In your report, describe (no need to implement) how this pipeline would be scheduled (e.g., Airflow), how you'd add new sources, and how you'd guard against scraping bans / CAPTCHAs.

3.3. Allowed Shortcuts

- You **do not** need to build a general NLP parser – simple keyword parsing or prompt-engineering an open API (e.g., OpenAI, SerpAPI) is fine.
- If a public email is not easily available, leave that field blank but still record the candidate.
- Limit API keys to free tiers; scrub any private credentials before sharing code.

3.4. What We're Looking For

Aspect	Indicators of Strength
Resourcefulness	Chooses realistic data sources; handles obstacles (rate limits, pagination).
Data hygiene	Performs basic validation, deduplication, and documents assumptions.
Automation mindset	Provides a clear roadmap to productionise the pipeline (scheduling, monitoring, retries).
Ethics & compliance	Acknowledges legal/ethical constraints of web-scraping and personal-data use.

---

5. Tips & Expectations

- **Time-box yourself.** We care more about seeing *your* best approach in a limited window than a perfect final product. If something is unfinished, note what you would do next.
- **Document assumptions.** E.g., "We assume event topics are already cleaned" or "We assume public emails are allowed to be stored."
- **Be pragmatic.** Off-the-shelf models or APIs are acceptable if you explain why they are appropriate and how you would switch to in-house solutions later.
- **Show your thinking.** Inline comments, commit messages, or a brief design doc section "Why I chose X over Y" help us understand your decision process.
- **Keep it legal & ethical.** Only scrape public pages that permit it. Mask any sensitive data in your example outputs.

---

6. How to Submit

1. Push to a **public GitHub/GitLab repo** or create a ZIP archive.
2. Email the link/ZIP to interviewer's email with subject "**[Your Name] – ML Take-Home Submission**".
3. Include any special run instructions in the email body if not already in the README.

We look forward to seeing your work. Thank you for investing the time to demonstrate your skills, and good luck!