# Purple Team — Weekender: Pittsburgh Plan-O-Matic

## Executive Summary

**Goal.** Build a Python-only command-line tool that plans an optimal Pittsburgh weekend by scoring and ranking events against forecasted weather and nearby food options. The output is a ranked itinerary plus clear visuals (charts and an optional map) that look polished without heavy engineering.

**Why it matters.** People waste time juggling tabs for events, weather, and restaurants. We unify them into one repeatable data pipeline and a simple decision algorithm.

**Data plan (≥3 sources, ≥1 scraped).** - **Weather:** Open-Meteo forecast (hourly/daily; no key) - **Events:** Eventbrite Events API (with token) **and** a scraped local calendar page (e.g., VisitPittsburgh) - **Food nearby:** Yelp Fusion Business Search API (to count/rank well-rated spots close to each event)

**Deliverable.** A single runnable Python entrypoint (e.g., `weekender.py`) that prompts for date/time window, budget, and start location, then prints and saves: (1) a ranked event list with component scores; (2) charts as PNGs; (3) optional HTML map if we include a Folium view.

---

## Project Objectives

1. **Aggregate** events for a chosen weekend (Fri–Sun) from at least two distinct sources (API + one scraped page).
2. **Enrich** each event with weather features matching its time window and with nearby dining density/quality.
3. **Score** each event via a transparent composite formula (weather fit, food density/quality, travel distance, budget fit).
4. **Output** a sorted itinerary and simple visuals, with a cache option to avoid slow/fragile network calls.
5. **Document** a clean setup/run path (README), and provide a first-draft pitch deck aligned to the class template.

---

## Users & Use Cases

- **Students / residents** deciding between events when the weather is iffy.
- **Visitors** who want a quick “best bet” list for a specific date with tasty food nearby.
- **Policy/Events teams** (stretch): sense-check whether outdoor programming is robust to weather.

---

## Scope & Non-Goals

**In scope** - Weekends in the next ~2 weeks (configurable) - Simple heuristic classification of indoor/outdoor events - Straight-line travel time proxy (haversine + default walking/driving speed) - CSV/JSON cache to support reproducible runs

**Out of scope** (for v1) - True routing/traffic times, ticket purchasing, calendars integration - Advanced recommender models or user history

---

## Data Sources (Planned)

1. **Open-Meteo API** — hourly precipitation probability, temperature, wind. Used to derive a per-event **weather_fit** score.
2. **Eventbrite API** — event lists, times, venues (lat/lon where available). Used for canonical event attributes.
3. **Scraped local events page** — VisitPittsburgh (or Downtown Partnership/Cultural Trust calendar) to meet the scraping requirement and to supplement the API with hyperlocal events.
4. **Yelp Fusion API** — dining options near event venue; derive **food_score** from counts and ratings within a radius.

**Data freshness.** Weather is fetched for the specific weekend (or read from cache if `--use-cache`). Events and Yelp data are cached per run.

**Robustness.** If scraping fails, the program still runs using Eventbrite + Yelp. If an API is rate-limited, we back off and use cached JSON.

---

## System Architecture (Python-only)

```
weekender/
  weekender.py                # SINGLE entrypoint (CLI). Imports modules
below.
  config.py                   # API tokens, defaults, constants (reads
from .env if present)
  etl/
    fetch_eventbrite.py       # API pull, with pagination & basic
normalization
    scrape_visitpgh.py        # Requests + BeautifulSoup scrape → rows [title,
dt, venue, url]
    fetch_weather.py          # Open-Meteo hourly forecast retrieval
    fetch_yelp.py             # Yelp search near (lat, lon) for eateries
    cache.py                  # read/write JSON & CSV; simple TTL logic
  features/
    classify_indoor_outdoor.py  # keyword rules ("park", "trail", "museum",
etc.)
    scoring.py                  # composite score; helpers for normalization
    geo.py                      # haversine distance; coarse travel-time proxy
  viz/
    charts.py                   # matplotlib figures saved to /out/*.png
    mapview.py                  # optional Folium HTML map (stretch goal)
  data/
    raw/                        # cached API/HTML responses
    processed/                  # tidy CSVs
  out/                          # report CSV + PNG charts + optional HTML map
  README.md
  requirements.txt
```

---

## Stack & Libraries

- **Core:** `python>=3.11`, `requests`, `pandas`, `beautifulsoup4`, `matplotlib`
- **Optional:** `folium` (HTML map), `python-dotenv` (local tokens), `tqdm` (progress bars)
- **Testing/quality (light):** `pytest`, `black`, `ruff` (optional but recommended)

**Why this stack?** It's standard, lightweight, and satisfies the course emphasis (scraping + APIs + pandas + matplotlib) while keeping installation minimal.

---

## CLI Interface (User Flow)

**Run:**

```
python weekender.py --start 2025-09-12 --end 2025-09-14
  --home "5000 Forbes Ave, Pittsburgh, PA"
  --budget 60 --style outdoorsy --fresh yes
```

**Prompts (if flags omitted):** date range, budget ($), preferred style (`indoorsy|outdoorsy|mixed`).

**Outputs:** - `out/ranked_events.csv` with columns: `rank,score,weather_fit,food_score,travel_score,budget_score,title,start,end,venue,lat,lon,source,u` - `out/top10_bar.png`, `out/timeline.png` (and `out/map.html` if Folium is included) - Console printout of Top 5 with component breakdowns

---

## Feature Engineering & Scoring

### 1) Indoor/Outdoor classification (rule-based)

- **Outdoor hints:** title/venue contains {park, trail, riverfront, outdoors, plaza, market (outdoor), concert (outdoor)}
- **Indoor hints:** {museum, gallery, theater, library, hall, arena, rink, studio}
- Default: `indoors` when unknown.

### 2) Weather features (from hourly forecast)

For each event time window, aggregate: - `p_precip` = max precipitation probability in window - `t_mean` = average temperature in window (°F) - `wind_max` = max wind speed in window

### 3) Weather-fit score

Normalize each component to 0–1 and compute:

```
weather_fit = 1
if outdoor:
    weather_fit -= 0.7 * clamp01((p_precip - 0.15)/0.35)  # strong rain penalty above 15%
    weather_fit -= 0.2 * clamp01((abs(t_mean-72))/25)     # discomfort away from ~72°F
    weather_fit -= 0.1 * clamp01((wind_max-10)/20)        # breezy penalty
else:  # indoor
    weather_fit -= 0.3 * clamp01((p_precip - 0.30)/0.50)  # smaller rain penalty
    weather_fit -= 0.2 * clamp01((abs(t_mean-72))/35)
weather_fit = clamp01(weather_fit)
```

### 4) Food score (Yelp)

Within radius `R=600m` of venue: count businesses with `rating≥4.2` and `review_count≥50`.

```
food_score = clamp01( 0.7 * min(top_spots, 8)/8 + 0.3 * avg_rating/5 )
```

### 5) Travel score (coarse)

Haversine distance from `home` to venue. Assume 5 km/h walking or 25 km/h rideshare default.

```
travel_minutes = (distance_km / 25) * 60
travel_score = clamp01(1 - travel_minutes/30)  # 30+ min decays to ~0
```

### 6) Budget score (simple)

If event has price info: full points if `price <= budget`, else linear decay until `2*budget`. Unknown price → neutral (0.6) so unknowns aren't unfairly punished.

### 7) Composite score (weights)

```
score = 0.45*weather_fit + 0.25*food_score + 0.15*travel_score +
0.15*budget_score
```

Weights are easy to tweak in `config.py`.

---

## Data Model (Tidy Tables)

**events.csv** - `event_id, source (eventbrite|visitpgh), title, description, category, start, end, venue_name, lat, lon, price_low, price_high, url, indoor_outdoor`

**weather_hourly.csv** - `ts, lat, lon, temp_f, precip_prob, wind_mph`

**yelp_stats.csv** - `event_id, radius_m, top_spots, avg_rating, avg_review_count`

**ranked_events.csv** (final) - `rank, score, weather_fit, food_score, travel_score, budget_score, <event fields…>`

---

## Roles & Workstreams (Team of 4)

**Purple Team Roster** - **Noah** — Project Lead & Viz/Packaging - **Teammate B** — Events Ingestion (API + Scrape) - **Teammate C** — Weather & Scoring - **Teammate D** — Yelp & Geospatial

> Swap names as you wish. Responsibilities below are designed to interlock with minimal merge pain.

### Role 1 — Events Ingestion (API + Scrape)

**Deliverables** - `etl/fetch_eventbrite.py` with: token auth, city/lat-lon filters, pagination, venue lat/lon resolution - `etl/scrape_visitpgh.py` with: HTTP GET, polite headers, table/list parsing via BeautifulSoup, date normalization - `data/processed/events.csv` (deduped across sources)

**Key details** - Normalize time to local TZ (America/New_York). Ensure ISO-8601 strings. - Use a **source** column to track provenance. - Implement simple cleaning: strip whitespace, drop events missing title/start. - Caching: write raw responses to `data/raw/eventbrite_*.json`, `data/raw/visitpgh_*.html`.

**Tests** - Parsing unit tests on static HTML sample (saved under `tests/samples/visitpgh.html`).

### Role 2 — Weather & Scoring

**Deliverables** - `etl/fetch_weather.py` that pulls hourly weather for each unique venue (lat, lon) and the event time window - `features/scoring.py` implementing the formula above + normalization helpers - `data/processed/weather_hourly.csv`

**Key details** - Map event time windows to relevant hourly rows (left-join by hour). - Ensure temperature is Fahrenheit and wind in mph (convert if needed). - Provide a `--fresh` flag to bypass cache.

**Tests** - Deterministic scoring for crafted inputs (e.g., heavy rain → low score).

### Role 3 — Yelp & Geospatial

**Deliverables** - `etl/fetch_yelp.py` to query food near each venue, returning `top_spots`, `avg_rating`, etc. - `features/geo.py` providing haversine + travel minutes. - `data/processed/yelp_stats.csv`

**Key details** - Respect daily request caps; sleep between pages. - If venue lat/lon missing, geocode fallback is **out of scope**; skip with a warning.

**Tests** - Deterministic extraction of `top_spots` from a saved JSON fixture.

### Role 4 — Viz/Packaging & CLI

**Deliverables** - `weekender.py` (single entrypoint) invoking modules and writing outputs - `viz/charts.py` producing: `top10_bar.png`, `timeline.png` - Optional `viz/mapview.py` producing `map.html` (Folium) - `README.md` with install/run steps and screenshots of outputs

**Key details** - Argument parsing with `argparse` and sensible defaults. - Clear console output (Top 5 with component scores). - Save artifacts to `/out` with timestamped filenames.

**Tests** - Smoke test: run CLI on fixture data with `--use-cache` to produce outputs without network.

---

## Git & Collaboration

- **Repo name:** `purple-weekender`
- **Branching:** feature branches per role (`feat/events`, `feat/weather`, `feat/yelp`, `feat/viz`)
- **PRs:** small, reviewer from adjacent role; squash-merge
- **Code style:** apply `black` and `ruff` (pre-commit optional)
- **Issues/Milestones:** use GitHub Projects board (To Do / In Progress / Review / Done)

---

## Schedule & Milestones

**By Thursday (Draft deck due)** - Events ingestion MVP from **one** source (Eventbrite **or** scraper) with cache - Weather fetch for a single venue and date, working end-to-end - CLI that prints Top 5 using stubbed/default scores - One chart saved as PNG (e.g., `top10_bar.png` on dummy data) - First-draft slides (Vision → Problem → Solution → Data Sources → Demo preview)

**Week 2** - Add second/third data sources (scraped site + Yelp) - Implement full scoring and two charts + optional map - README v1 with screenshots

**Week 3 (polish)** - Robust caching and error handling, final weights tuning, UX polish - Final slide deck + short demo video

---

## Risks & Mitigations

- **Scrape breaks** → Pin a sample HTML file and support cached parse.
- **API rate limits** → Cache responses; exponential backoff; document how to add tokens.
- **Missing venue coords** → Skip event with warning; plenty of others remain.
- **Weather mismatch** → Always log the chosen hourly rows per event for auditability.

---

## Ethics, Terms & Privacy

- Respect `robots.txt` and site terms. Identify user-agent and add a polite delay for scraping.
- Only collect public event info; no PII. Keep tokens in local `.env` (not committed).

---

## Installation & Running (Draft)

**requirements.txt** (initial)

```
requests
pandas
beautifulsoup4
matplotlib
folium
python-dotenv
```

**Setup**

```
python -m venv .venv
source .venv/bin/activate  # Windows: .venv\Scripts\activate
pip install -r requirements.txt
cp .env.example .env  # add EVENTBRITE_TOKEN, YELP_API_KEY
python weekender.py --start 2025-09-12 --end 2025-09-14 --home "5000 Forbes Ave, Pittsburgh, PA"
```

`.env.example`

```
EVENTBRITE_TOKEN=xxxxxxxxxxxxxxxx
YELP_API_KEY=xxxxxxxxxxxxxxxx
```

---

## Minimal Code Skeleton (illustrative)

```python
# weekender.py
from etl.fetch_eventbrite import get_events
from etl.scrape_visitpgh import scrape_events
from etl.fetch_weather import get_weather
from etl.fetch_yelp import get_yelp_stats
from features.scoring import score_events
from viz.charts import save_top10_bar, save_timeline

# parse args → call pipeline steps → write ranked_events.csv and charts
```

---

## Slide Deck Mapping (for the Draft)

- **Vision:** “Plan the perfect Pittsburgh weekend in one go.”
- **Problem:** Tabs + gut feel → suboptimal choices.
- **Solution:** Unified Python tool, transparent scoring, pretty outputs.
- **Data:** list sources; call out which is scraped; show a small table sample.
- **Demo Preview:** screenshot of `ranked_events.csv` head + `top10_bar.png`.
- **Team:** Purple Team roles + who is spokesperson and Canvas submitter.

---

## Definition of Done (DoD)

- One command runs end-to-end with either fresh or cached data
- `ranked_events.csv` produced with all four component scores
- At least two distinct charts saved to `/out`

- README with installation/run + screenshots; tokens handled via `.env`
- Draft and final pitch decks complete and consistent with the tool’s outputs

---

## Stretch Ideas (if time allows)

- Lightweight clustering of events (time/price/distance) to suggest combos
- Simple “day planner” that avoids overlapping start times
- Add a preference profile (indoors vs outdoors weight) persisted to a small JSON file

---

**Purple Team FTW.** Simple pipeline, fancy outputs, clean code. Let’s ship it. 💜