Master calculation details

This document explains in detail how the cafe/POI scoring and aggregation are computed by `master.py`.

Overview

- The generator reads a cafes CSV and multiple POI CSVs (banks, education, health, temples, other).
- For each POI category it computes per-POI combined weights (from rank/weight, rating, reviews, weekly hours), annotates and writes a small final CSV for that category, then computes per-cafe counts and weight sums within a radius.
- It aggregates per-category components, cafe-level properties, and nearby-cafes contributions into a final `poi_composite_score`.
- It also computes an individual per-cafe score and normalized `cafe_weight`, writes a `cafe_final.csv`, and produces a minimal master CSV.

Constants & Defaults

- `MASTER_RADIUS_M`: 1500 (meters) — radius used for master aggregation.
- `DEFAULT_WEEKLY`: 72 (12 * 6) — default weekly open hours used for normalization.
- `CAFE_PROPS_WEIGHT`, `CAFE_NEIGHBOR_WEIGHT`: 1.0 — multipliers applied to cafe-level props and nearby-cafes component.
- Haversine earth radius: $R = 6{,}371{,}000$ m.

Coordinate detection

- `detect_latlon(df)` scans the dataframe for common latitude/longitude column name pairs (for example, `lat` / `lon`, `latitude` / `longitude`, `y` / `x`) and returns the first matching pair. That pair is used as coordinates for distance calculations.

Per-POI weight calculation (function `compute_weights_and_annotate`)

1. Detect available columns:
   - weight/rank column via `detect_weight_col` (candidates include `weight`, `rank`, `score`, etc.).
   - rating column candidates: `rating`, `stars`.
   - reviews column candidates: `reviewsCount`, `reviews`, etc.
   - weekly hours candidates from `WEEKLY_HOURS_COL_CANDS`.
2. Base score from weight/rank column:
   - If the column is rank-like (`"rank" in wc.lower()`), missing ranks are filled with `max_rank + 1`, then the inverse is taken and normalized:
     - ranks_i = filled rank for row i
     - inv_i = 1 / (ranks_i + 1e-9)
     - base_i = inv_i / max_j(inv_j)
       Lower (better) ranks therefore become higher base scores after inversion and normalization.
   - Else (numeric weight): base_i = value_i / max(value).
   - If no weight column present: base_i = 0 for all rows.
3. Rating normalization:
   - rating_norm_i = (rating_i / 5.0) clipped to [0,1].
4. Reviews normalization:
   - reviews_norm_i = log1p(reviews_i) / max(log1p(reviews)) if any reviews present, else 0.
5. Weekly hours normalization:
   - weekly_hours_norm_i = (weekly_hours_i / DEFAULT_WEEKLY) clipped to [0,1]. If weekly missing, defaults to 1.0.
6. Combined per-POI weight:
   - The per-row combined score is the mean of available components: base, rating_norm (if present), reviews_norm (if present), weekly_hours_norm.
   - Formally for POI j with k available components:

$$\text{combined}_j = \frac{1}{k}\sum_{m=1}^{k} c_{jm}$$

- Saved as `_computed_weight` and `combined_score` in the annotated POI DataFrame.

7. Dynamic category weight:
   - `dyn_cat_w = mean(combined)` (the average combined score across POIs in the category).
   - This value is used to override the static category weight for the master composite.

Haversine distance (function `haversine_m`)

- Vectorized calculation giving distances in meters between a single point and arrays of points. Using lat/lon in radians:

$$\Delta\varphi = \varphi_2 - \varphi_1, \quad \Delta\lambda = \lambda_2 - \lambda_1$$

$$a = \sin^2\left(\frac{\Delta\varphi}{2}\right) + \cos\varphi_1 \cos\varphi_2 \sin^2\left(\frac{\Delta\lambda}{2}\right)$$

$$d = 2R \arctan 2(\sqrt{a}, \sqrt{1-a})$$

where $R = 6{,}371{,}000\,\mathrm{m}$.

Per-cafe POI metrics (function `compute_poi_metrics_for_cafes`)

For each cafe and each POI category (e.g., `banks`, `education`):

- Compute distances from the cafe to all POIs using haversine.
- Within-radius mask: `dists <= radius_m` (radius default used per-call; master uses `MASTER_RADIUS_M`).
- Outputs per-cafe (suffix `_{K}km`, e.g. `_1km` for 1000m):
  - `{category}_count_{K}km`: count of POIs within radius.
  - `{category}_weight_{K}km`: sum of `_computed_weight` of POIs within radius (i.e., sum of per-POI combined scores).
  - `{category}_min_dist_m`: minimum distance to a POI in that category within radius (NaN if none).
  - Also stores `{category}_category_weight = category_weight` for downstream composition.

Master composite score (in `generate_master_metrics`)

1. Per-category components:
   - For category `pname`, take the per-cafe weight column `w = {pname}_weight_{K}km`.
   - Normalize: if `max(w) > 0` then `norm_w_i = w_i / max(w)` else `norm_w_i = 0`.

- Multiply by the category weight (static or dynamic):
  `component_{pname}_i = norm_w_i * category_weights[pname]` .

2. Cafe-level property components:
   - Rating: `r_i / 5.0` (if `rating` present).
   - Reviews: `log1p(reviews_i) / max(log1p(reviews))` (if reviews present).
   - Weekly hours: `(weekly_hours_i / DEFAULT_WEEKLY).clip(0,1)` (defaults to 1.0 if missing).
   - Rank: inverse-of-rank normalized to 0..1 (lower rank → higher value).
   - These components are averaged to produce `cafe_props_i` :

$$\text{cafe\_props}_i = \frac{1}{k}\sum_{m=1}^{k} p_{im}$$

- The cafe-props contribution is scaled by `CAFE_PROPS_WEIGHT` .

3. Nearby-cafes component:
   - For each cafe i compute the sum of `cafe_props_j` for other cafes j where distance <= `MASTER_RADIUS_M` (exclude i). Call this `nearby_sum_i` .
   - Normalize: `nearby_norm_i = nearby_sum_i / max(nearby_sum)` if max>0 else 0.
   - Scale by `CAFE_NEIGHBOR_WEIGHT` and include as a component. Also saves raw `cafes_nearby_weight_{K}km` .
4. Final composite:
   - Sum all components element-wise:

$$\text{poi\_composite\_score}_i = \sum_{\text{components}} \text{component}_i$$

- Saved as `poi_composite_score` .

Individual cafe weight & related fields

- `cafe_individual_score = cafe_props` (mean of the cafe-level normalized components).
- `cafe_weight` = normalized `cafe_individual_score` scaled to 0..1 by dividing by `max(cafe_individual_score)` (or 0 if max is 0).
- `cafes_count_1km` : number of *other* cafes within 1000 meters (excludes the cafe itself). Calculated by pairwise haversine distances between cafes and counting neighbors with `d <= 1000` .

Column name conventions

- Per-category counts and weights: `{category}_count_{K}km`, `{category}_weight_{K}km`, `{category}_min_dist_m`, `{category}_category_weight`.
- Cafe-level fields added: `cafe_individual_score`, `cafe_weight`, `cafes_nearby_weight_{K}km`, `cafes_count_1km`.
- Master composite: `poi_composite_score`.

Files produced and contents

- `backend/Data/CSV/master_cafes_metrics.csv` : full output with original cafe columns plus per-category counts/weights/min distances, `cafe_individual_score`, `cafe_weight`, `poi_composite_score`, `cafes_nearby_weight_{K}km`, `cafes_count_1km`.
- `backend/Data/CSV/final/cafe_final.csv` : curated per-cafe CSV with name, lat/lon, key cafe fields (rating/reviews/weekly/rank if present), `cafe_individual_score`, `cafe_weight`, per-category counts and weights, and `cafes_count_1km`.
- `backend/Data/CSV/final/master_cafes_minimal.csv` : minimal master with name, lat/lon, per-category counts/weights, `cafe_weight`, and `poi_composite_score`.

Notes & edge-case handling

- Missing numeric columns are treated as zeros or sensible defaults (e.g., missing weekly hours → `DEFAULT_WEEKLY`, missing ranks → `max_rank + 1`).
- When a normalization denominator is zero (no values present or all zeros), that term yields zero for all rows rather than causing divide-by-zero.
- If a POI CSV lacks coordinates, the script writes zeros/NaNs for that category in the cafes dataset.
- Dynamic category weights are computed as the mean per-POI combined score and may override static `CATEGORY_WEIGHTS`.

How to run and inspect outputs

Run the generator:

```
python backend/Data/master.py
```

Preview the cafe final CSV (example):

```
python - <<'PY'
import pandas as pd
df = pd.read_csv("backend/Data/CSV/final/cafe_final.csv")
print(df.head(10).to_markdown())
PY
```

If you want a specific explanation of any column or the math behind a particular normalization, tell me which one and I'll expand it here.