ReadMe (analyze_loop.py - Field-Test Data Processing Report)

Distance-to-RSU, range profiles, and elevation enrichment

Primary I/O: metrics.csv + events.csv -> metrics_enhanced.csv + range profiles + RSU metadata + elevation

Overview

This document describes how analyze_loop.py is used after field data collection to convert raw monitoring outputs (metrics.csv and events.csv) into analysis-ready datasets. The script adds time features, computes distance-to-RSU columns for one or more RSUs, produces distance-binned range profiles, and can enrich samples with elevation_m from USGS EPQS using a local cache to minimize repeated API calls.

1. What the script produces

analyze_loop.py processes a single field-test loop directory and generates standardized outputs used for downstream plotting and reporting:

- Time features: t_sec, dt_sec, has_pkts, time_since_last_pkt.
- Distance-to-RSU columns for each RSU: dist_<RSU_ID>_m (e.g., dist_RSU1_m).
- Nearest-RSU selection: nearest_rsu and dist_from_rsu_m.
- Optional union distance for multi-RSU scenarios: dist_union_m (minimum distance to any RSU).
- Distance-binned range profiles (coverage fraction, mean delta_bytes, mean_time_since_last_pkt).
- Elevation_m via USGS EPQS with CSV caching.
- RSU metadata export to rsus_used.json for traceability.

2. Recommended project structure

Place analyze_loop.py under scripts/ and organize data by scenario. A typical layout used in this project is:

```
RSU_Range/
  scripts/
    analyze_loop.py
  data/
    highway/
      raw/
        <scenario_name>/
          <loop_id>/
            metrics.csv
            events.csv
      processed/
        <scenario_name>/
          <loop_id>/
            (generated files)
    county/
      raw/
        uphill/
          metrics.csv
```

```
      events.csv
    downhill/
      metrics.csv
      events.csv
  processed/
    uphill/
    downhill/
city/
  raw/
    <scenario_name>/
      <loop_id>/
        metrics.csv
        events.csv
  processed/
    <scenario_name>/
      <loop_id>/
cache/
  elevation_cache.csv
venv/
```

Important: --loop-dir must point to a folder that contains metrics.csv (required) and optionally events.csv (recommended).

3. Inputs and outputs

3.1 Input files (from field monitoring)

Required: metrics.csv must contain at least:

- timestamp
- latitude
- longitude
- delta_bytes

Optional: events.csv should contain at least:

- timestamp
- latitude
- longitude

3.2 Output files (written to --out-dir)

The script writes the following files to the output directory:

1. metrics_enhanced.csv - original metric columns plus derived time and distance features (and elevation_m if enabled).

2. range_profile_nearest.csv - distance-binned profile using nearest-RSU distance (dist_from_rsu_m).

3. events_with_distance.csv - only if events.csv exists; events table with the same distance-to-RSU columns.

4. rsus_used.json - RSU metadata used for this run (IDs + coordinates).

5. range_profile_union.csv - only if --write-union-profile and 2+ RSUs; bins by dist_union_m (min distance to any RSU).

Key derived columns added to metrics_enhanced.csv:

- t_sec, dt_sec
- has_pkts (delta_bytes > 0)
- time_since_last_pkt
- dist_<RSU_ID>_m for each RSU
- nearest_rsu, dist_from_rsu_m
- dist_union_m (if 2+ RSUs)
- elevation_m (if --add-elevation)

4. Environment setup (Windows example)

From the project root (RSU_Range):

4.1 Create and activate a virtual environment

```
python -m venv venv
venv\Scripts\activate
```

4.2 Install dependencies

```
pip install -U pip
pip install numpy pandas requests
```

5. Script usage

5.1 Help

```
python scripts\analyze_loop.py -h
```

5.2 Required arguments

- --loop-dir: folder containing metrics.csv (and optionally events.csv)
- --out-dir: folder where processed outputs will be written

5.3 RSU input options (choose one)

Option A - Provide RSUs on the command line (repeatable):

```
--rsu "RSU1:lat,lon" --rsu "RSU2:lat,lon"
```

Option B - Provide an RSU JSON file:

```
--rsu-json path\to\rsus.json
```

Expected JSON format:

```
{
  "rsus": [
    {"id": "RSU1", "lat": 0.0, "lon": 0.0},
    {"id": "RSU2", "lat": 0.0, "lon": 0.0}
  ]
}
```

5.4 Elevation options (USGS EPQS)

- --add-elevation: add elevation_m via USGS EPQS
- --elev-cache: cache CSV path (default: cache/elevation_cache.csv)
- --elev-round: rounding decimals for caching (default: 5)

5.5 Optional union profile

- --write-union-profile: writes union profile if 2+ RSUs are provided

6. Example commands (same style used in this project)

Replace lat,lon with your RSU coordinates for each scenario.

6.1 Highway - Single RSU (per loop)

```
python scripts\analyze_loop.py ^
  --loop-dir "data\highway\raw\single_rsu\170246" ^
  --out-dir  "data\highway\processed\single_rsu\170246" ^
  --bin-m 50 ^
  --rsu "RSU1:lat,lon" ^
  --add-elevation
```

6.2 Highway - Two RSU (overlap / separate)

```
python scripts\analyze_loop.py ^
  --loop-dir "data\highway\raw\two_rsu_overlap\162700" ^
  --out-dir  "data\highway\processed\two_rsu_overlap\162700" ^
  --bin-m 50 ^
  --rsu "RSU1:lat,lon" ^
  --rsu "RSU2:lat,lon" ^
  --add-elevation ^
  --write-union-profile
```

6.3 County - Uphill / Downhill (single file per direction)

```
python scripts\analyze_loop.py ^
  --loop-dir "data\county\raw\downhill" ^
  --out-dir  "data\county\processed\downhill" ^
  --bin-m 50 ^
  --rsu "RSU1:lat,lon" ^
  --add-elevation
```

6.4 City - Two RSU scenarios (per loop)

```
python scripts\analyze_loop.py ^
  --loop-dir "data\city\raw\two_rsu_same_side\171008" ^
  --out-dir  "data\city\processed\two_rsu_same_side\171008" ^
  --bin-m 50 ^
  --rsu "RSU1:lat,lon" ^
  --rsu "RSU2:lat,lon" ^
  --add-elevation ^
  --write-union-profile
```

7. Elevation enrichment (USGS EPQS) and caching

When run with --add-elevation, the script:

1. Rounds each (latitude, longitude) pair to --elev-round decimals (default 5).

2. Computes unique rounded coordinate keys for the run.

3. Checks the cache CSV (--elev-cache) and reuses any existing elevations.

4. Queries USGS EPQS only for missing coordinate keys (then appends results to the cache).

5. Merges elevation_m back into metrics_enhanced.csv.

8. Common errors and fixes

8.1 Missing required arguments

You must provide --loop-dir and --out-dir:

```
python scripts\analyze_loop.py --loop-dir "..." --out-dir "..."
```

8.2 FileNotFoundError for metrics.csv

- Confirm metrics.csv is inside the folder passed to --loop-dir.
- Confirm the path spelling and quoting are correct.

8.3 PermissionError on output CSV

- Close the output CSV if it is open in Excel or another program.
- Rerun the command after closing the file handle.

8.4 EPQS rate limiting (HTTP 429)

- Reuse the cache (recommended).
- Run fewer sessions back-to-back.
- Keep --elev-round 5 to reduce unique coordinate count.

9. Why this preprocessing stage matters

After running analyze_loop.py, each loop has a standardized set of tables that downstream plotting scripts can consume consistently across scenarios. metrics_enhanced.csv becomes the main analysis table for route-based and distance-based plots, and the range_profile CSVs provide compact binned summaries suitable for coverage curves and comparison charts.

10. Notes on scenarios

- Highway and City typically store each drive as a separate loop folder (one loop per timestamp).
- County is often organized as one uphill folder and one downhill folder rather than multiple loop IDs; in that case, --loop-dir points directly to the direction folder.