

fixing_road

February 17, 2026

```
[1]: import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import os
import seaborn as sns
from pathlib import Path
```

1 EDA

```
[2]: RAW_PATH = Path("../data/raw/_roads.tsv")
OUT_PATH = Path("../data/processed/_roads.tsv")
```

```
[3]: road=pd.read_csv(RAW_PATH, sep='\t', low_memory=False)
road
```

```
[3]:      road    lrp1      lat1      lon1      lrp2      lat2      lon2 \
0       N1    LRPS  23.706028  90.443333    LRPSa  23.702917  90.450417
1     N101    LRPS  23.454139  91.212861    LRPSa  23.461889  91.212000
2     N102    LRPS  23.478972  91.118194    LRPSa  23.481583  91.116777
3     N103    LRPS  23.957028  91.115528  LRP001  23.961917  91.113611
4     N104    LRPS  23.009667  91.399416    LRPSa  23.009278  91.395250
..      ...
880   Z8910    LRPS  22.674722  90.408889    LRPSa  22.675916  90.412556
881   Z8913    LRPS  22.396083  90.688666    LRPSa  22.393027  90.688944
882   Z8915    LRPS  22.589389  90.619472  LRP001  22.589694  90.623360
883   Z8916    LRPS  22.625499  90.661722    LRPSa  22.623888  90.664167
884   Z8943    LRPS  22.426444  90.849472    LRPSa  22.426416  90.846833

      Unnamed: 7  Unnamed: 8  Unnamed: 9  ...  Unnamed: 4035  Unnamed: 4036 \
0       LRPSb  23.702778  90.450472  ...        92.29825      LRP466c
1       LRP001  23.462944  91.211806  ...        NaN        NaN
2       LRPSb  23.486666  91.113361  ...        NaN        NaN
3      LRP001a  23.967666  91.111889  ...        NaN        NaN
4       LRP001  23.009306  91.389805  ...        NaN        NaN
..      ...
880     LRP001  22.675583  90.417166  ...        NaN        ...
881     LRPSb  22.392666  90.689083  ...        NaN        NaN
```

882	LRP001a	22.590027	90.631360	...	NaN	NaN
883	LRP001	22.620305	90.668999	...	NaN	NaN
884	LRP001	22.425444	90.839861	...	NaN	NaN
		Unnamed: 4037	Unnamed: 4038	Unnamed: 4039	Unnamed: 4040	Unnamed: 4041 \
0		20.864667	92.298194	LRP467	20.862972	92.298083
1		NaN	NaN	NaN	NaN	NaN
2		NaN	NaN	NaN	NaN	NaN
3		NaN	NaN	NaN	NaN	NaN
4		NaN	NaN	NaN	NaN	NaN
..	
880		NaN	NaN	NaN	NaN	NaN
881		NaN	NaN	NaN	NaN	NaN
882		NaN	NaN	NaN	NaN	NaN
883		NaN	NaN	NaN	NaN	NaN
884		NaN	NaN	NaN	NaN	NaN
		Unnamed: 4042	Unnamed: 4043	Unnamed: 4044		
0		LRPE	20.862917	92.298083		
1		NaN	NaN	NaN		
2		NaN	NaN	NaN		
3		NaN	NaN	NaN		
4		NaN	NaN	NaN		
..			
880		NaN	NaN	NaN		
881		NaN	NaN	NaN		
882		NaN	NaN	NaN		
883		NaN	NaN	NaN		
884		NaN	NaN	NaN		

[885 rows x 4045 columns]

```
[4]: # ---- Load wide file (like you already did) ----
road = road.copy()

# Make sure first column is named 'road'
road = road.rename(columns={road.columns[0]: "road"})

tidy_rows = []

for _, row in road.iterrows():
    road_name = row["road"]

    # Everything after first column are repeating triples
    values = row.iloc[1:].values

    # Loop through in steps of 3 (lrp, lat, lon)
```

```

for i in range(0, len(values), 3):
    try:
        lrp = values[i]
        lat = values[i+1]
        lon = values[i+2]
    except IndexError:
        break # incomplete triple at end

    # Skip completely empty triples
    if pd.isna(lrp) and pd.isna(lat) and pd.isna(lon):
        continue

    tidy_rows.append({
        "road": road_name,
        "lrp": lrp,
        "lat": pd.to_numeric(lat, errors="coerce"),
        "lon": pd.to_numeric(lon, errors="coerce")
    })

# Create tidy dataframe
tidy = pd.DataFrame(tidy_rows)

# Drop rows where everything is missing
tidy = tidy.dropna(subset=["lrp", "lat", "lon"], how="all")

# Reset index
tidy = tidy.reset_index(drop=True)

print("Tidy shape:", tidy.shape)
tidy

```

Tidy shape: (52210, 4)

	road	lrp	lat	lon
0	N1	LRPS	23.706028	90.443333
1	N1	LRPSa	23.702917	90.450417
2	N1	LRPSb	23.702778	90.450472
3	N1	LRP001	23.702139	90.451972
4	N1	LRP002	23.697889	90.460583
...
52205	Z8943	LRP007	22.429499	90.785722
52206	Z8943	LRP008	22.430110	90.776333
52207	Z8943	LRP008a	22.430249	90.774888
52208	Z8943	LRP008b	22.430249	90.774860
52209	Z8943	LRPE	22.430166	90.768916

[52210 rows x 4 columns]

```
[5]: print("Rows before:", len(tidy))

tidy = tidy.drop_duplicates(subset=["road", "lrp"])

print("Rows after:", len(tidy))
dup_check = tidy.duplicated(subset=["road", "lrp"]).sum()
print("Remaining duplicates:", dup_check)
```

```
Rows before: 52210
Rows after: 51926
Remaining duplicates: 0
```

```
[6]: print("Number of roads:", tidy["road"].nunique())
print("Total points:", len(tidy))

print("\nLatitude summary:")
print(tidy["lat"].describe())

print("\nLongitude summary:")
print(tidy["lon"].describe())
```

```
Number of roads: 877
Total points: 51926

Latitude summary:
count      51926.000000
mean       23.825781
std        1.145005
min        2.643639
25%        22.934527
50%        23.828458
75%        24.718277
max        32.561222
Name: lat, dtype: float64
```

```
Longitude summary:
count      51926.000000
mean       90.275652
std        1.087424
min        82.460278
25%        89.361868
50%        90.171833
75%        91.088770
max        93.298416
Name: lon, dtype: float64
```

```
[7]: # Bangladesh rough bounding box
invalid_bd = tidy[
```

```

        (tidy["lat"] < 20) | (tidy["lat"] > 27) |
        (tidy["lon"] < 88) | (tidy["lon"] > 93)
    ]

print("Outside Bangladesh bounds:", len(invalid_bd))
invalid_bd

```

Outside Bangladesh bounds: 21

[7]:

	road	lrp	lat	lon
2690	N2	LRP113b	27.126944	91.319110
27617	Z1813	LRP013c	22.197833	93.298416
33190	Z4021	LRP026a	25.066777	87.809444
36445	Z5058	LRP008	27.287777	89.647472
38324	Z5463	LRP021a	24.797722	87.706333
41679	Z6031	LRP015	24.110500	85.327805
43343	Z7023	LRP039	23.255972	87.797250
43778	Z7045	LRP017a	27.944111	89.790694
44833	Z7461	LRP003b	23.950278	87.963556
45366	Z7504	LRP010	23.234583	82.460278
45542	Z7551	LRP010a	23.066167	87.162278
45620	Z7552	LRP019b	27.869888	89.062444
45752	Z7602	LRP012	2.643639	88.990166
46051	Z7606	LRP002a	27.786139	89.536916
46054	Z7606	LRP003a	29.777083	89.534000
46347	Z7618	LRP006b	27.491944	89.159556
46354	Z7618	LRP008c	27.473722	89.155611
46358	Z7618	LRP009c	27.468888	89.156250
46559	Z7702	LRP006a	32.561222	89.860361
47606	Z7801	LRP018a	22.796916	87.772110
49143	Z8056	LRP005a	27.645666	90.304944

[8]:

```

roads3 = pd.read_csv("../data/raw/_roads3.csv")

# Keep only geometry columns for comparison
roads3_tidy = roads3[["road", "lrp", "lat", "lon"]].copy()

# Ensure numeric
roads3_tidy["lat"] = pd.to_numeric(roads3_tidy["lat"], errors="coerce")
roads3_tidy["lon"] = pd.to_numeric(roads3_tidy["lon"], errors="coerce")

roads3_tidy = roads3_tidy.dropna(subset=["lat", "lon"]).reset_index(drop=True)

print("roads3_tidy shape:", roads3_tidy.shape)
roads3_tidy

```

roads3_tidy shape: (51348, 4)

```
[8]:      road      lrp       lat       lon
 0        N1    LRPS  23.706028  90.443333
 1        N1  LRPSa  23.702917  90.450417
 2        N1  LRPSb  23.702778  90.450472
 3        N1  LRP001  23.702139  90.451972
 4        N1  LRP002  23.697889  90.460583
 ...
51343    Z8943  LRP007  22.429499  90.785722
51344    Z8943  LRP008  22.430110  90.776333
51345    Z8943  LRP008a 22.430249  90.774888
51346    Z8943  LRP008b 22.430249  90.774860
51347    Z8943    LRPE  22.430166  90.768916
```

[51348 rows x 4 columns]

```
[9]: tidy.shape, roads3_tidy.shape
```

```
[9]: ((51926, 4), (51348, 4))
```

```
[10]: print("Duplicates in raw tidy:", tidy.duplicated(["road", "lrp"]).sum())

roads_raw = set(tidy["road"].unique())
roads_prof = set(roads3_tidy["road"].unique())

extra_roads = roads_raw - roads_prof
print("Extra roads:", len(extra_roads))

rows_from_extra_roads = tidy[tidy["road"].isin(extra_roads)]
print("Rows from extra roads:", len(rows_from_extra_roads))

print("Total row difference:", len(tidy) - len(roads3_tidy))
```

Duplicates in raw tidy: 0

Extra roads: 30

Rows from extra roads: 191

Total row difference: 578

```
[11]: # Count points per road
pts_per_road = tidy.groupby("road").size()

# Keep roads with more than 3 points
roads_to_keep = pts_per_road[pts_per_road > 3].index

print("Rows before:", len(tidy))
print("Roads before:", tidy["road"].nunique())

# Overwrite tidy
tidy = tidy[tidy["road"].isin(roads_to_keep)].copy()
```

```
print("Rows after:", len(tidy))
print("Roads after:", tidy["road"].nunique())
```

```
Rows before: 51926
Roads before: 877
Rows after: 51861
Roads after: 852
```

```
[12]: print("Your rows:", len(tidy))
print("Professor rows:", len(roads3_tidy))
print("Difference:", len(tidy) - len(roads3_tidy))
```

```
Your rows: 51861
Professor rows: 51348
Difference: 513
```

```
[13]: roads_tidy = set(tidy["road"].dropna().unique())
roads_prof = set(roads3_tidy["road"].dropna().unique())

only_in_tidy = sorted(roads_tidy - roads_prof)
only_in_prof = sorted(roads_prof - roads_tidy)

print("Roads only in tidy:", len(only_in_tidy))
print("Roads only in professor:", len(only_in_prof))
only_in_tidy
```

```
Roads only in tidy: 5
Roads only in professor: 0
```

```
[13]: ['N211', 'R822', 'Z1025', 'Z1447', 'Z1605']
```

2 Fixing the Road

```
[14]: import re

_lrp_re = re.compile(r"^\LRP(\d+)([A-Za-z]*)$")

def lrp_sort_key(lrp: str):
    """
    Returns a tuple used for sorting LRP in chainage-like order:
    LRPS, LRPSa, LRPSb ... first
    then LRP001, LRP001a, LRP001b ...
    then ...
    LRPE last
    Unknown formats go to the end (but before LRPE).
    """
    if pd.isna(lrp):
```

```

    return (10**9, "zz") # push NA to end

s = str(lrp).strip()

# Start markers
if s == "LRPS":
    return (-1, "")
if s.startswith("LRPS") and len(s) > 4:
    # LRPSa, LRPSb ...
    return (-1, s[4:])

# End marker
if s == "LRPE":
    return (10**12, "zzzz")

# Standard LRP###suffix
m = _lrp_re.match(s)
if m:
    num = int(m.group(1))
    suf = m.group(2) or ""
    return (num, suf)

# fallback for weird labels
return (10**9, s)

def sort_road_by_lrp(df_points):
    df = df_points.copy()
    df["_k"] = df["lrp"].apply(lrp_sort_key)
    df = df.sort_values("_k").drop(columns="_k").reset_index(drop=True)
    return df

```

```

[15]: # -----
# Distance
# -----
def haversine(lat1, lon1, lat2, lon2):
    R = 6371.0
    lat1, lon1, lat2, lon2 = map(np.radians, [lat1, lon1, lat2, lon2])
    dlat = lat2 - lat1
    dlon = lon2 - lon1
    a = np.sin(dlat/2)**2 + np.cos(lat1)*np.cos(lat2)*np.sin(dlon/2)**2
    c = 2 * np.arcsin(np.sqrt(a))
    return R * c

def compute_segments(df):
    """Adds next_lat/next_lon + seg distance. Expects ordered points."""
    out = df.copy()
    out["next_lat"] = out["lat"].shift(-1)

```

```

        out["next_lon"] = out["lon"].shift(-1)
        out["seg"] = haversine(out["lat"], out["lon"], out["next_lat"], □
        ↵out["next_lon"]))
    return out

# -----
# Stage A: Block repair
# -----
def block_repair(df_points, K=15, min_run_len=2):
    """
    Detect consecutive jump segments and linearly interpolate across the block.
    df_points: columns ['lrp', 'lat', 'lon'] in correct order.
    Returns: fixed_df, actions(list)
    """
    df = compute_segments(df_points).reset_index(drop=True)
    median_seg = df["seg"].median()
    thr = K * median_seg

    df["is_jump_seg"] = df["seg"] > thr
    df["run_id"] = (df["is_jump_seg"] != df["is_jump_seg"].shift()).cumsum()

    jump_runs = df[df["is_jump_seg"]].groupby("run_id").indices
    fixed = df_points.copy().reset_index(drop=True)
    actions = []

    for run_id, idxs in jump_runs.items():
        idxs = list(idxs)
        if len(idxs) < min_run_len:
            continue

        # segments idxs[0]..idxs[-1] correspond to points [left_idx ..□
        ↵right_idx]
        left_idx = idxs[0]
        right_idx = idxs[-1] + 1

        # bounds + must have endpoints
        if left_idx <= 0 or right_idx >= len(fixed):
            continue
        if fixed.loc[left_idx, ["lat", "lon"]].isna().any() or fixed.□
        ↵loc[right_idx, ["lat", "lon"]].isna().any():
            continue

        L = right_idx - left_idx
        for k in range(1, L):
            new_lat = fixed.loc[left_idx, "lat"] + (fixed.loc[right_idx, "lat"] □
            ↵- fixed.loc[left_idx, "lat"]) * (k / L)

```

```

        new_lon = fixed.loc[left_idx, "lon"] + (fixed.loc[right_idx, "lon"] -  

        ← fixed.loc[left_idx, "lon"]) * (k / L)

        old_lat, old_lon = fixed.loc[left_idx+k, "lat"], fixed.  

        ← loc[left_idx+k, "lon"]
        fixed.loc[left_idx+k, "lat"] = new_lat
        fixed.loc[left_idx+k, "lon"] = new_lon

        actions.append({
            "idx": left_idx+k,
            "lrp": fixed.loc[left_idx+k, "lrp"],
            "action": "block_interp",
            "old_lat": old_lat, "old_lon": old_lon,
            "new_lat": new_lat, "new_lon": new_lon,
            "run_len": len(idxs),
            "thr_km": thr
        })

    return fixed, actions

# -----
# Stage B: Smart single-segment repair
# -----
def smart_single_seg_repair(df_points, K=15, max_iters=5):
    """
    Fix remaining isolated jump segments by deciding whether to adjust point i or i+1.
    df_points: columns ['lrp', 'lat', 'lon'] ordered.
    Returns: fixed_df, actions(list)
    """
    df = df_points.copy().reset_index(drop=True)
    actions = []

    for _ in range(max_iters):
        tmp = compute_segments(df)
        med = tmp["seg"].median()
        thr = K * med

        bad_idxs = tmp.index[tmp["seg"] > thr].tolist()
        if not bad_idxs:
            break

        for i in bad_idxs:
            # need neighborhood i-1, i, i+1, i+2
            if i <= 0 or i >= len(df)-2:
                continue

```

```

p_im1 = (df.loc[i-1, "lat"], df.loc[i-1, "lon"])
p_i   = (df.loc[i,   "lat"], df.loc[i,   "lon"])
p_ip1 = (df.loc[i+1, "lat"], df.loc[i+1, "lon"])
p_ip2 = (df.loc[i+2, "lat"], df.loc[i+2, "lon"])

if any(pd.isna([*p_im1, *p_i, *p_ip1, *p_ip2])):
    continue

cur = (
    haversine(p_im1[0], p_im1[1], p_i[0], p_i[1]) +
    haversine(p_i[0], p_i[1], p_ip1[0], p_ip1[1]) +
    haversine(p_ip1[0], p_ip1[1], p_ip2[0], p_ip2[1])
)

# Fix A: adjust point i
fixA = ((p_im1[0] + p_ip1[0]) / 2, (p_im1[1] + p_ip1[1]) / 2)
scoreA = (
    haversine(p_im1[0], p_im1[1], fixA[0], fixA[1]) +
    haversine(fixA[0], fixA[1], p_ip1[0], p_ip1[1]) +
    haversine(p_ip1[0], p_ip1[1], p_ip2[0], p_ip2[1])
)

# Fix B: adjust point i+1
fixB = ((p_i[0] + p_ip2[0]) / 2, (p_i[1] + p_ip2[1]) / 2)
scoreB = (
    haversine(p_im1[0], p_im1[1], p_i[0], p_i[1]) +
    haversine(p_i[0], p_i[1], fixB[0], fixB[1]) +
    haversine(fixB[0], fixB[1], p_ip2[0], p_ip2[1])
)

best = min(cur, scoreA, scoreB)
if best >= cur:
    continue

if best == scoreA:
    old_lat, old_lon = df.loc[i, "lat"], df.loc[i, "lon"]
    df.loc[i, "lat"], df.loc[i, "lon"] = fixA
    actions.append({
        "idx": i,
        "lrp": df.loc[i, "lrp"],
        "action": "smart_fix_i_mid(im1,ip1)",
        "old_lat": old_lat, "old_lon": old_lon,
        "new_lat": fixA[0], "new_lon": fixA[1],
        "thr_km": thr
    })
else:
    old_lat, old_lon = df.loc[i+1, "lat"], df.loc[i+1, "lon"]

```

```

        df.loc[i+1, "lat"], df.loc[i+1, "lon"] = fixB
        actions.append({
            "idx": i+1,
            "lrp": df.loc[i+1, "lrp"],
            "action": "smart_fix_i+1_mid(i,ip2)",
            "old_lat": old_lat, "old_lon": old_lon,
            "new_lat": fixB[0], "new_lon": fixB[1],
            "thr_km": thr
        })
    }

    return df, actions
}

# -----
# Full repair for one road
# -----
def repair_road_points(df_road_points, K=15, min_run_len=2):
    # NEW: sort by LRP before any distance work
    df_road_points = sort_road_by_lrp(df_road_points)

    before = compute_segments(df_road_points)
    before_med = before["seg"].median()
    before_thr = K * before_med
    before_jumps = int((before["seg"] > before_thr).sum())
    before_max = float(before["seg"].max())

    fixedA, actionsA = block_repair(df_road_points, K=K,
                                     min_run_len=min_run_len)
    fixedB, actionsB = smart_single_seg_repair(fixedA, K=K)

    after = compute_segments(fixedB)
    after_med = after["seg"].median()
    after_thr = K * after_med
    after_jumps = int((after["seg"] > after_thr).sum())
    after_max = float(after["seg"].max())

    metrics = {
        "median_seg_before": before_med,
        "thr_before": before_thr,
        "jumps_before": before_jumps,
        "max_seg_before": before_max,
        "median_seg_after": after_med,
        "thr_after": after_thr,
        "jumps_after": after_jumps,
        "max_seg_after": after_max,
        "actions_block": len(actionsA),
        "actions_smart": len(actionsB),
        "actions_total": len(actionsA) + len(actionsB),
    }

```

```

    }

    actions = actionsA + actionsB
    return fixedB, actions, metrics

# -----
# Plot before/after for a road
# -----
def plot_road_before_after(tidy_df, road_name, fixed_df=None, K=15, ↴
    ↪title_suffix=""):

    """
    tidy_df: full tidy dataframe (road, lrp, lat, lon) in original order
    fixed_df: optional fixed points (lrp, lat, lon) for the road; if None we ↴
    ↪repair on the fly.
    """
    road_pts = tidy_df[tidy_df["road"] == road_name][["lrp", "lat", "lon"]]. ↴
    ↪copy().reset_index(drop=True)

    if fixed_df is None:
        fixed_df, _, metrics = repair_road_points(road_pts, K=K)
    else:
        metrics = None

    # plot side-by-side
    fig, axes = plt.subplots(1, 2, figsize=(14, 6))

    axes[0].plot(road_pts["lon"], road_pts["lat"], marker="o", markersize=2, ↴
    ↪linewidth=1)
    axes[0].set_title(f"{road_name} BEFORE {title_suffix}")
    axes[0].set_xlabel("Longitude")
    axes[0].set_ylabel("Latitude")

    axes[1].plot(fixed_df["lon"], fixed_df["lat"], marker="o", markersize=2, ↴
    ↪linewidth=1)
    axes[1].set_title(f"{road_name} AFTER {title_suffix}")
    axes[1].set_xlabel("Longitude")
    axes[1].set_ylabel("Latitude")

    plt.tight_layout()
    plt.show()

    if metrics is not None:
        print(metrics)

# -----
# Repair ALL roads

```

```

# -----
def repair_all_roads(tidy_df, K=15, min_run_len=2, max_roads=None):
    """
    tidy_df: columns ['road', 'lrp', 'lat', 'lon'] in original order
    Returns: cleaned_tidy_df, actions_df, metrics_df
    """
    cleaned_parts = []
    all_actions = []
    all_metrics = []

    roads = tidy_df["road"].dropna().unique().tolist()
    if max_roads is not None:
        roads = roads[:max_roads]

    for r in roads:
        road_pts = tidy_df[tidy_df["road"] == r][["lrp", "lat", "lon"]].copy()
        road_pts.reset_index(drop=True)
        fixed_pts, actions, metrics = repair_road_points(road_pts, K=K,
                                                       min_run_len=min_run_len)

        fixed_out = fixed_pts.copy()
        fixed_out.insert(0, "road", r)
        cleaned_parts.append(fixed_out)

        for a in actions:
            a["road"] = r
        all_actions.extend(actions)

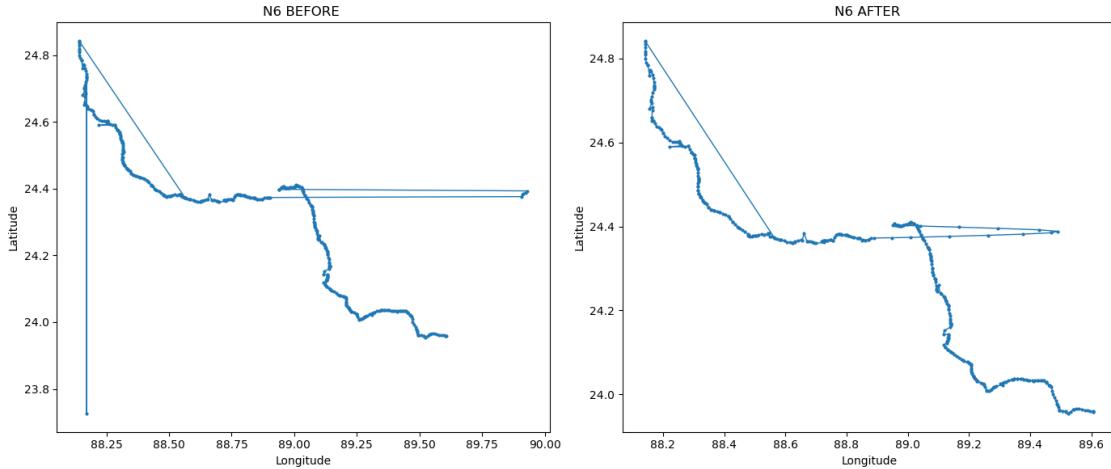
        metrics["road"] = r
        all_metrics.append(metrics)

    cleaned = pd.concat(cleaned_parts, ignore_index=True)
    actions_df = pd.DataFrame(all_actions)
    metrics_df = pd.DataFrame(all_metrics)

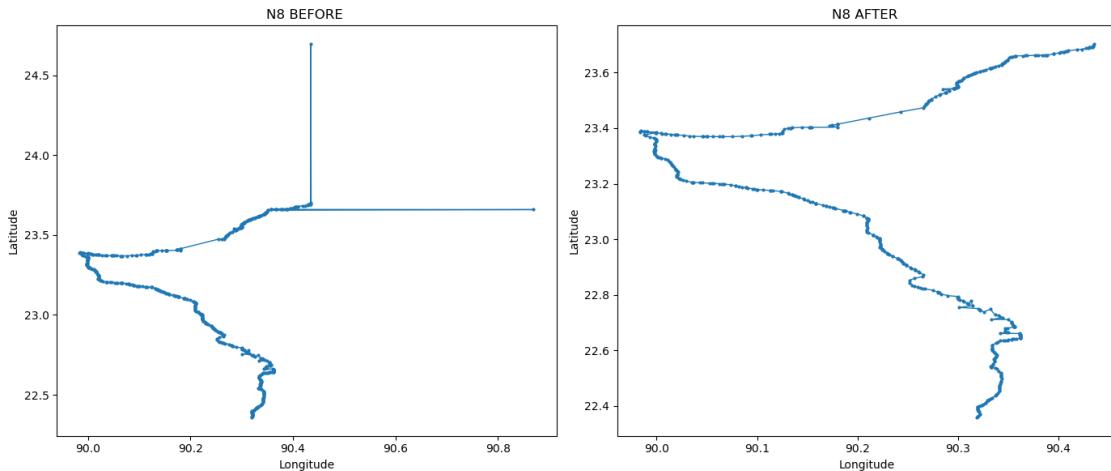
    return cleaned, actions_df, metrics_df

```

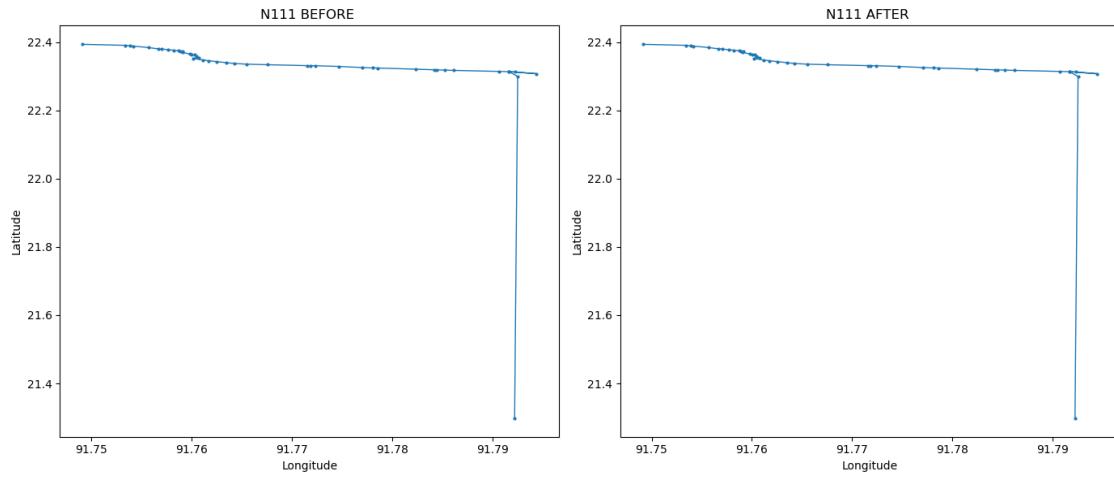
[16]: plot_road_before_after(tidy, "N6", K=15)
plot_road_before_after(tidy, "N8", K=15)
plot_road_before_after(tidy, "N111", K=15) # pick any road you know is messy



```
{'median_seg_before': np.float64(0.539430031216072), 'thr_before':  
np.float64(8.09145046824108), 'jumps_before': 5, 'max_seg_before':  
111.93011058173765, 'median_seg_after': np.float64(0.5472850147985902),  
'thr_after': np.float64(8.209275221978853), 'jumps_after': 9, 'max_seg_after':  
67.33889773877009, 'actions_block': 1, 'actions_smart': 33, 'actions_total': 34}
```



```
{'median_seg_before': np.float64(0.3186359366109248), 'thr_before':  
np.float64(4.779539049163873), 'jumps_before': 5, 'max_seg_before':  
111.21037161986999, 'median_seg_after': np.float64(0.3167099459429208),  
'thr_after': np.float64(4.750649189143812), 'jumps_after': 0, 'max_seg_after':  
4.054485533659262, 'actions_block': 1, 'actions_smart': 6, 'actions_total': 7}
```



```
{'median_seg_before': np.float64(0.2708457043236544), 'thr_before': np.float64(4.062685564854816), 'jumps_before': 1, 'max_seg_before': 111.27832730939268, 'median_seg_after': np.float64(0.2708457043236544), 'thr_after': np.float64(4.062685564854816), 'jumps_after': 1, 'max_seg_after': 111.27832730939268, 'actions_block': 0, 'actions_smart': 0, 'actions_total': 0}
```

```
[17]: def plot_road_raw_vs_sorted(tidy_df, road_name):
    raw = tidy_df[tidy_df["road"] == road_name][["lrp", "lat", "lon"]].copy()
    ↪reset_index(drop=True)
    srt = sort_road_by_lrp(raw)

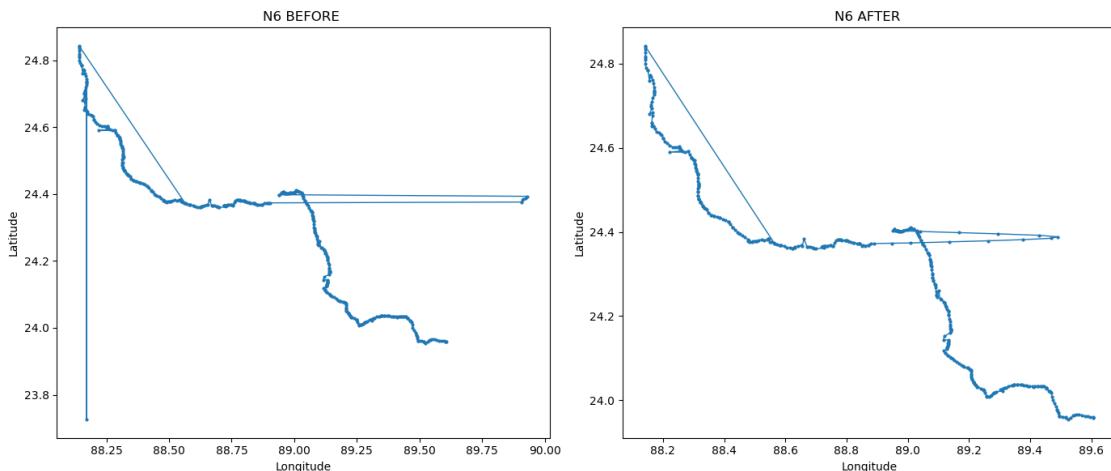
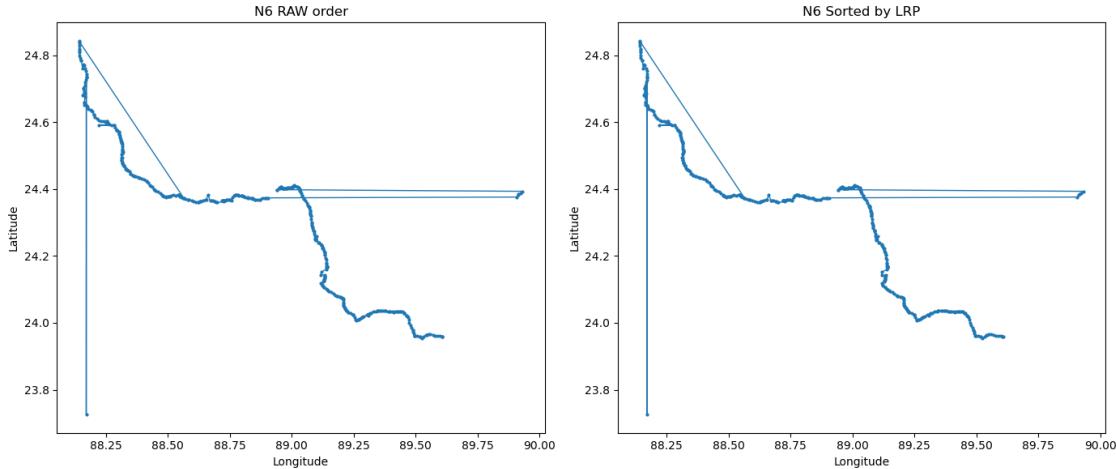
    fig, axes = plt.subplots(1, 2, figsize=(14, 6))
    axes[0].plot(raw["lon"], raw["lat"], marker="o", markersize=2, linewidth=1)
    axes[0].set_title(f"{road_name} RAW order")

    axes[1].plot(srt["lon"], srt["lat"], marker="o", markersize=2, linewidth=1)
    axes[1].set_title(f"{road_name} Sorted by LRP")

    for ax in axes:
        ax.set_xlabel("Longitude")
        ax.set_ylabel("Latitude")

    plt.tight_layout()
    plt.show()

plot_road_raw_vs_sorted(tidy, "N6")
plot_road_before_after(tidy, "N6", K=15)
```



```
{'median_seg_before': np.float64(0.539430031216072), 'thr_before': np.float64(8.09145046824108), 'jumps_before': 5, 'max_seg_before': 111.93011058173765, 'median_seg_after': np.float64(0.5472850147985902), 'thr_after': np.float64(8.209275221978853), 'jumps_after': 9, 'max_seg_after': 67.33889773877009, 'actions_block': 1, 'actions_smart': 33, 'actions_total': 34}
```

```
[18]: def detect_spikes(df_points, factor=5):
    """
    Detect spike points using local triangle rule.
    Returns list of indices to fix.
    """
    df = compute_segments(df_points)
    med = df[{"seg"}].median()

    spike_idxs = []
```

```

for i in range(1, len(df_points)-1):
    d1 = haversine(df_points.loc[i-1,"lat"], df_points.loc[i-1,"lon"],
                    df_points.loc[i,"lat"], df_points.loc[i,"lon"])
    d2 = haversine(df_points.loc[i,"lat"], df_points.loc[i,"lon"],
                    df_points.loc[i+1,"lat"], df_points.loc[i+1,"lon"])
    d3 = haversine(df_points.loc[i-1,"lat"], df_points.loc[i-1,"lon"],
                    df_points.loc[i+1,"lat"], df_points.loc[i+1,"lon"])

    if d1 > factor*med and d2 > factor*med and d3 < factor*med:
        spike_idxs.append(i)

return spike_idxs

def repair_spikes(df_points, factor=5):
    df = df_points.copy().reset_index(drop=True)
    spike_idxs = detect_spikes(df, factor=factor)

    for i in spike_idxs:
        df.loc[i,"lat"] = (df.loc[i-1,"lat"] + df.loc[i+1,"lat"]) / 2
        df.loc[i,"lon"] = (df.loc[i-1,"lon"] + df.loc[i+1,"lon"]) / 2

    return df, spike_idxs

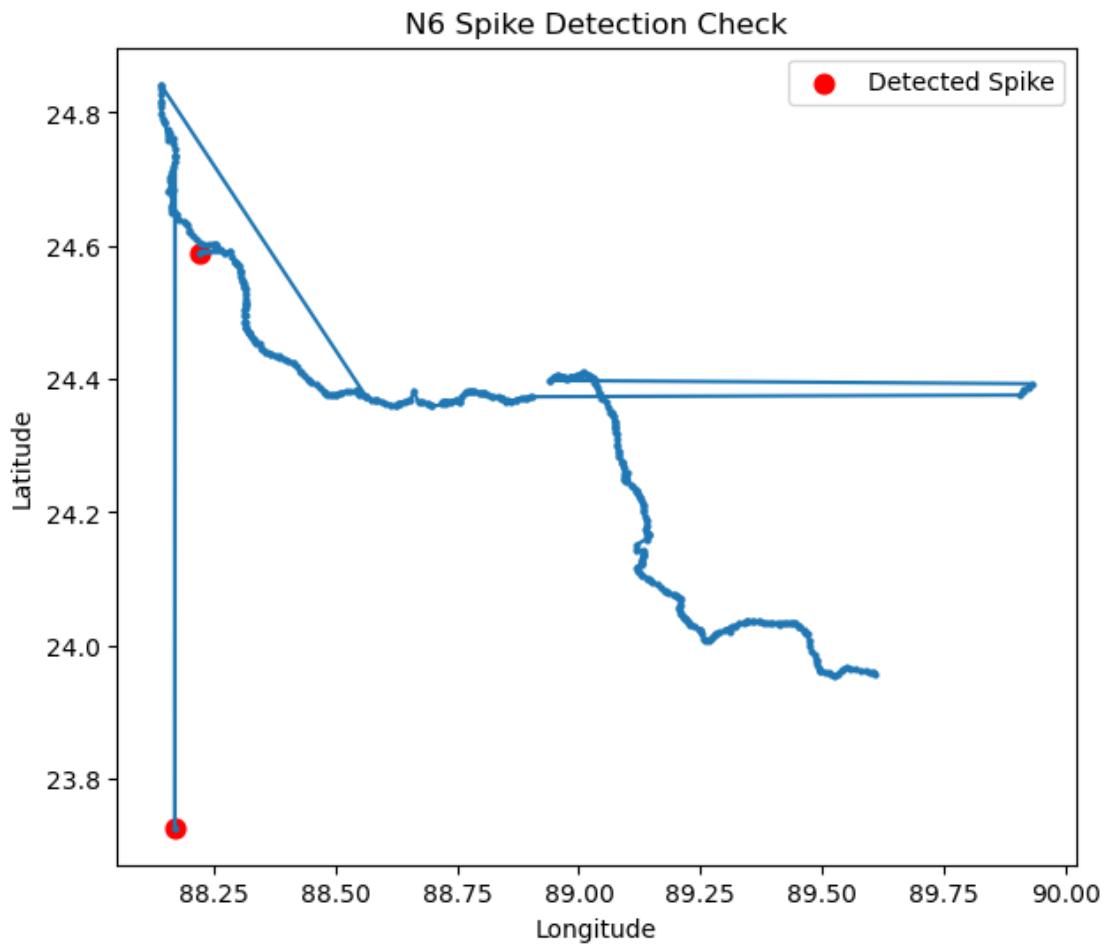
n6 = tidy[tidy["road"] == "N6"][["lrp","lat","lon"]].copy().
    ↪reset_index(drop=True)
spike_idxs = detect_spikes(n6, factor=5)
print("Spike indices:", spike_idxs)
print("Number of spikes:", len(spike_idxs))
plt.figure(figsize=(7,6))
plt.plot(n6["lon"], n6["lat"], marker="o", markersize=2)

if spike_idxs:
    plt.scatter(n6.loc[spike_idxs,"lon"],
                n6.loc[spike_idxs,"lat"],
                color="red", s=60, label="Detected Spike")

plt.title("N6 Spike Detection Check")
plt.xlabel("Longitude")
plt.ylabel("Latitude")
plt.legend()
plt.show()

```

Spike indices: [369, 404]
Number of spikes: 2

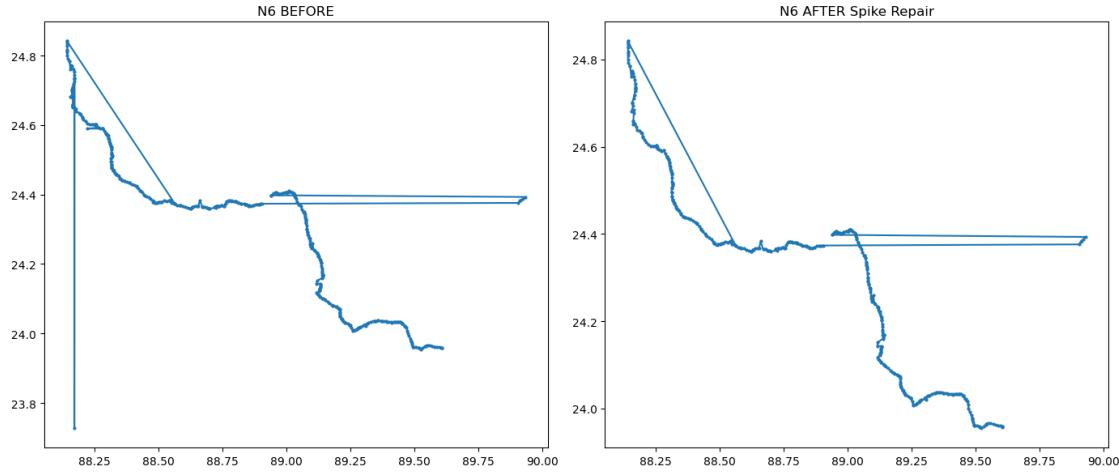


```
[19]: n6_fixed, spike_idxs = repair_spikes(n6, factor=5)
fig, axes = plt.subplots(1,2, figsize=(14,6))

axes[0].plot(n6["lon"], n6["lat"], marker="o", markersize=2)
axes[0].set_title("N6 BEFORE")

axes[1].plot(n6_fixed["lon"], n6_fixed["lat"], marker="o", markersize=2)
axes[1].set_title("N6 AFTER Spike Repair")

plt.tight_layout()
plt.show()
```



```
[20]: def plot_road_before_after_prof(tidy_df, road_name, prof_df, fixed_df=None, ↴
    ↴K=15, title_suffix=""):

    """
    tidy_df: your full tidy dataframe (road, lrp, lat, lon)
    prof_df: professor tidy dataframe (road, lrp, lat, lon)
    fixed_df: optional fixed points df for the road (lrp, lat, lon). If None -> ↴
    ↴repair on the fly.
    """

    # BEFORE (yours raw)
    road_pts = (
        tidy_df[tidy_df["road"] == road_name][["lrp", "lat", "lon"]]
        .copy()
        .reset_index(drop=True)
    )

    if road_pts.empty:
        print(f"Road {road_name} not found in tidy_df.")
        return

    # AFTER (yours fixed)
    if fixed_df is None:
        fixed_df, actions, metrics = repair_road_points(road_pts, K=K)
    else:
        actions, metrics = None, None

    # PROFESSOR
    prof_pts = (
        prof_df[prof_df["road"] == road_name][["lrp", "lat", "lon"]]
        .copy()
        .reset_index(drop=True)
    )
```

```

)

# 3-panel plot
fig, axes = plt.subplots(1, 3, figsize=(20, 6))

axes[0].plot(road_pts["lon"], road_pts["lat"], marker="o", markersize=2, linewidth=1)
axes[0].set_title(f"{road_name} BEFORE {title_suffix}")
axes[0].set_xlabel("Longitude")
axes[0].set_ylabel("Latitude")

axes[1].plot(fixed_df["lon"], fixed_df["lat"], marker="o", markersize=2, linewidth=1)
axes[1].set_title(f"{road_name} AFTER {title_suffix}")
axes[1].set_xlabel("Longitude")
axes[1].set_ylabel("Latitude")

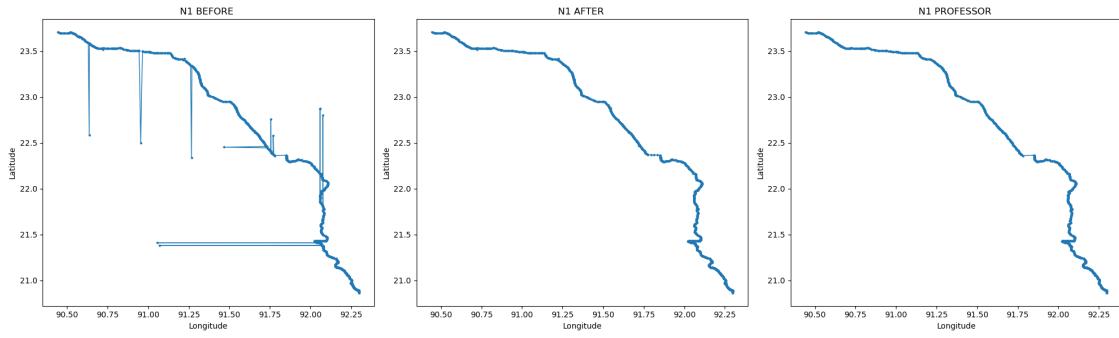
if prof_pts.empty:
    axes[2].text(0.5, 0.5, "Road not in professor data", ha="center", va="center")
    axes[2].set_axis_off()
else:
    axes[2].plot(prof_pts["lon"], prof_pts["lat"], marker="o", markersize=2, linewidth=1)
    axes[2].set_title(f"{road_name} PROFESSOR {title_suffix}")
    axes[2].set_xlabel("Longitude")
    axes[2].set_ylabel("Latitude")

plt.tight_layout()
plt.show()

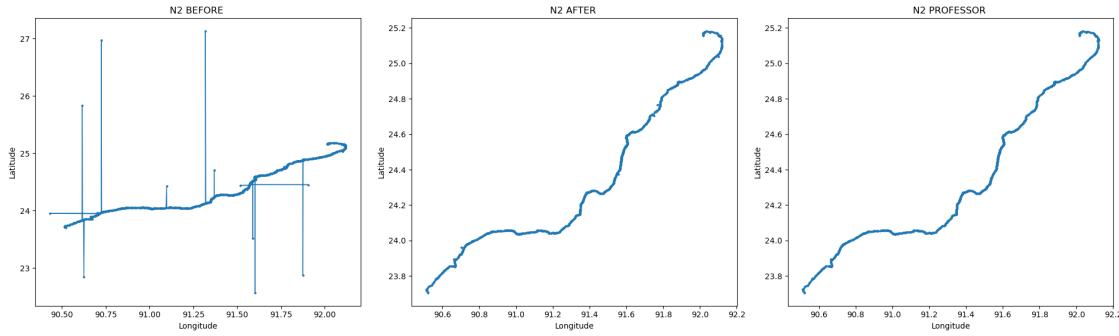
if metrics is not None:
    print(metrics)

# optionally also return objects if you want to reuse
return fixed_df, actions, metrics
plot_road_before_after_prof(tidy, "N1", roads3_tidy, K=15)
plot_road_before_after_prof(tidy, "N2", roads3_tidy, K=15)
plot_road_before_after_prof(tidy, "N11", roads3_tidy, K=15)    # pick any road
    ↪you know is messy

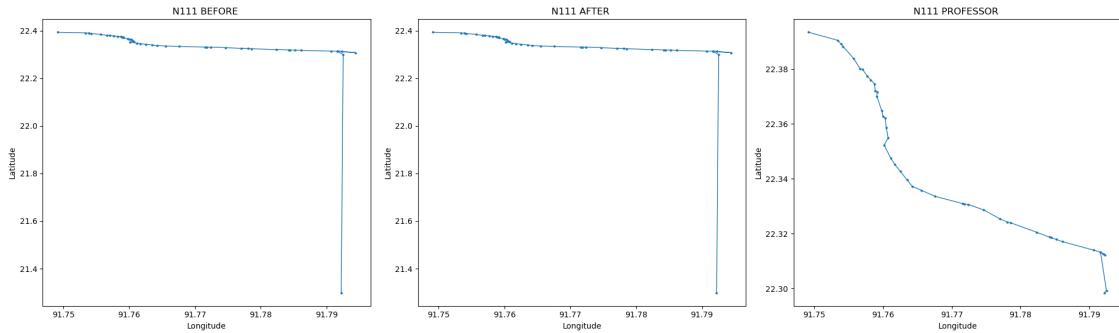
```



```
{'median_seg_before': np.float64(0.24966877898556183), 'thr_before': np.float64(3.7450316847834273), 'jumps_before': 23, 'max_seg_before': 112.16489375714208, 'median_seg_after': np.float64(0.25243396534927465), 'thr_after': np.float64(3.7865094802391197), 'jumps_after': 0, 'max_seg_after': 2.8560856083097277, 'actions_block': 9, 'actions_smart': 32, 'actions_total': 41}
```



```
{'median_seg_before': np.float64(0.27002073788171577), 'thr_before': np.float64(4.050311068225737), 'jumps_before': 24, 'max_seg_before': 333.61567045487607, 'median_seg_after': np.float64(0.2650569699659913), 'thr_after': np.float64(3.9758545494898696), 'jumps_after': 0, 'max_seg_after': 2.2702737979702183, 'actions_block': 11, 'actions_smart': 24, 'actions_total': 35}
```



```
{'median_seg_before': np.float64(0.2708457043236544), 'thr_before':  
np.float64(4.062685564854816), 'jumps_before': 1, 'max_seg_before':  
111.27832730939268, 'median_seg_after': np.float64(0.2708457043236544),  
'thr_after': np.float64(4.062685564854816), 'jumps_after': 1, 'max_seg_after':  
111.27832730939268, 'actions_block': 0, 'actions_smart': 0, 'actions_total': 0}
```

	lrp	lat	lon
0	LRPS	22.393444	91.749139
1	LRPSa	22.390500	91.753389
2	LRP001	22.389166	91.753889
3	LRP001a	22.388222	91.754194
4	LRP001b	22.383889	91.755722
5	LRP002	22.380194	91.756667
6	LRP002a	22.379833	91.757028
7	LRP002b	22.377499	91.757694
8	LRP002c	22.375999	91.758222
9	LRP002d	22.374638	91.758750
10	LRP003	22.371999	91.758889
11	LRP003a	22.371694	91.759139
12	LRP003b	22.370083	91.759111
13	LRP003c	22.364889	91.759806
14	LRP004	22.362778	91.760028
15	LRP004a	22.362222	91.760306
16	LRP004b	22.358639	91.760472
17	LRP005	22.354944	91.760750
18	LRP005a	22.352333	91.760167
19	LRP005b	22.347472	91.761139
20	LRP006	22.345222	91.761722
21	LRP006a	22.342750	91.762528
22	LRP006b	22.339611	91.763500
23	LRP007	22.337277	91.764250
24	LRP007a	22.335305	91.765528
25	LRP007b	22.333639	91.767583
26	LRP007c	22.330999	91.771555
27	LRP008	22.330833	91.771833
28	LRP008a	22.330694	91.772388
29	LRP008b	22.328666	91.774666
30	LRP008c	22.325388	91.777055
31	LRP009	22.324277	91.778083
32	LRP009a	22.323999	91.778583
33	LRP009b	22.320527	91.782388
34	LRP009c	22.318777	91.784222
35	LRP009d	22.318583	91.784472
36	LRP010	22.317916	91.785222
37	LRP010a	22.317083	91.786166

```

38 LRP010b 22.314000 91.790694
39 LRP010c 22.313278 91.791666
40 LRP011 22.312306 91.792305
41 LRP011a 22.307278 91.794416
42 LRP012 22.313278 91.791666
43 LRP012a 22.299222 91.792527
44 LRPE 21.298472 91.792222,
[],
{'median_seg_before': np.float64(0.2708457043236544),
 'thr_before': np.float64(4.062685564854816),
 'jumps_before': 1,
 'max_seg_before': 111.27832730939268,
 'median_seg_after': np.float64(0.2708457043236544),
 'thr_after': np.float64(4.062685564854816),
 'jumps_after': 1,
 'max_seg_after': 111.27832730939268,
 'actions_block': 0,
 'actions_smart': 0,
 'actions_total': 0})

```

```

[ ]: cleaned_tidy, actions_df, metrics_df = repair_all_roads(tidy, K=10, ↵
    ↵min_run_len=2, max_roads=None)
print("Cleaned tidy shape:", cleaned_tidy.shape)
cleaned_tidy2, actions_df, metrics_df = repair_all_roads(cleaned_tidy, K=10, ↵
    ↵min_run_len=2, max_roads=None)

def plot_all_roads_before_after(tidy_df, cleaned_df):
    fig, axes = plt.subplots(1, 2, figsize=(16, 7))

    # BEFORE
    axes[0].scatter(
        tidy_df["lon"], tidy_df["lat"],
        s=1, alpha=0.5
    )
    axes[0].set_title("All Roads BEFORE Repair")
    axes[0].set_xlabel("Longitude")
    axes[0].set_ylabel("Latitude")

    # AFTER
    axes[1].scatter(
        cleaned_df["lon"], cleaned_df["lat"],
        s=1, alpha=0.5
    )
    axes[1].set_title("All Roads AFTER Repair")
    axes[1].set_xlabel("Longitude")
    axes[1].set_ylabel("Latitude")

```

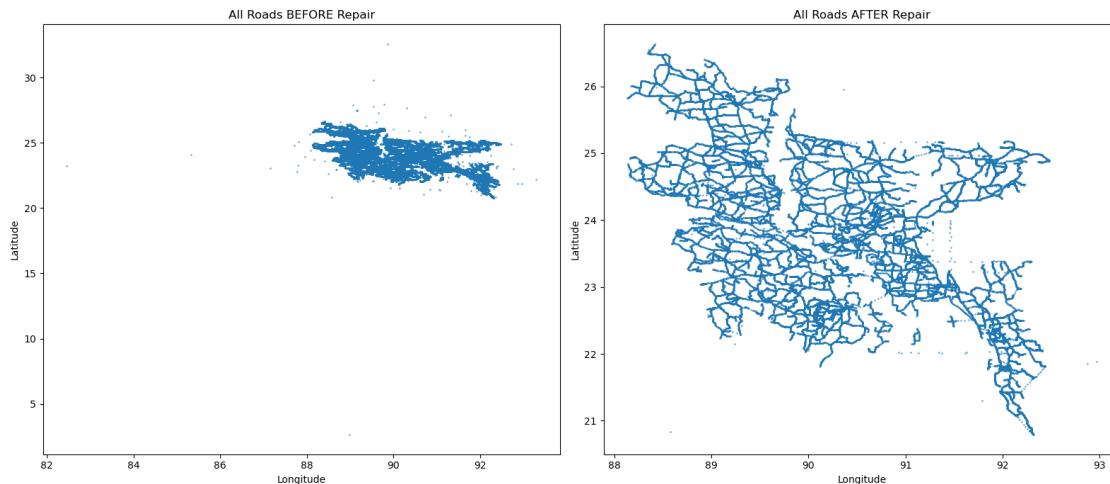
```

plt.tight_layout()
plt.show()

# plot_all_roads_before_after(tidy, cleaned_tidy)

```

Cleaned tidy shape: (51861, 4)



```

[27]: def plot_three_all_roads(before_df, after_df, ref_df):
    fig, axes = plt.subplots(1, 3, figsize=(21, 7))

    axes[0].scatter(before_df["lon"], before_df["lat"], s=1, alpha=0.3)
    axes[0].set_title("Original (_roads.tsv)")
    axes[0].set_xlabel("Longitude")
    axes[0].set_ylabel("Latitude")

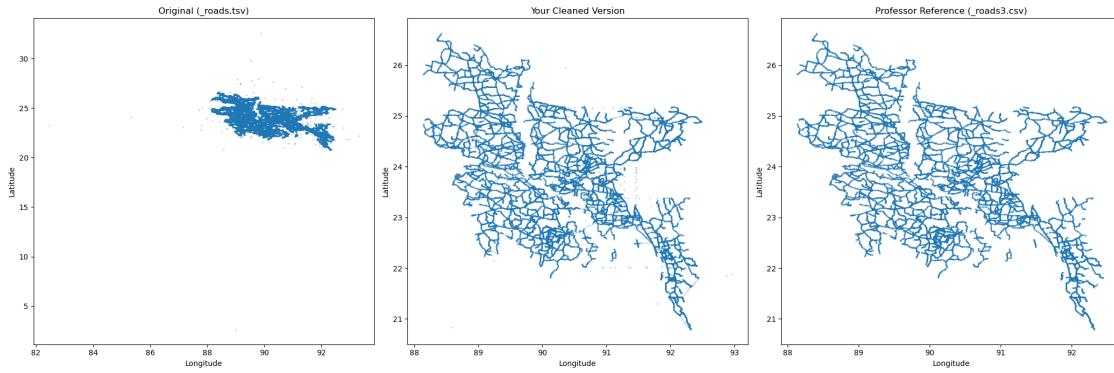
    axes[1].scatter(after_df["lon"], after_df["lat"], s=1, alpha=0.3)
    axes[1].set_title("Your Cleaned Version")
    axes[1].set_xlabel("Longitude")
    axes[1].set_ylabel("Latitude")

    axes[2].scatter(ref_df["lon"], ref_df["lat"], s=1, alpha=0.3)
    axes[2].set_title("Professor Reference (_roads3.csv)")
    axes[2].set_xlabel("Longitude")
    axes[2].set_ylabel("Latitude")

    plt.tight_layout()
    plt.show()

plot_three_all_roads(tidy, cleaned_tidy, roads3_tidy)

```



```
[ ]: def plot_three_all_roads_lines(before_df, after_df, ref_df):
    fig, axes = plt.subplots(1, 3, figsize=(21, 7))

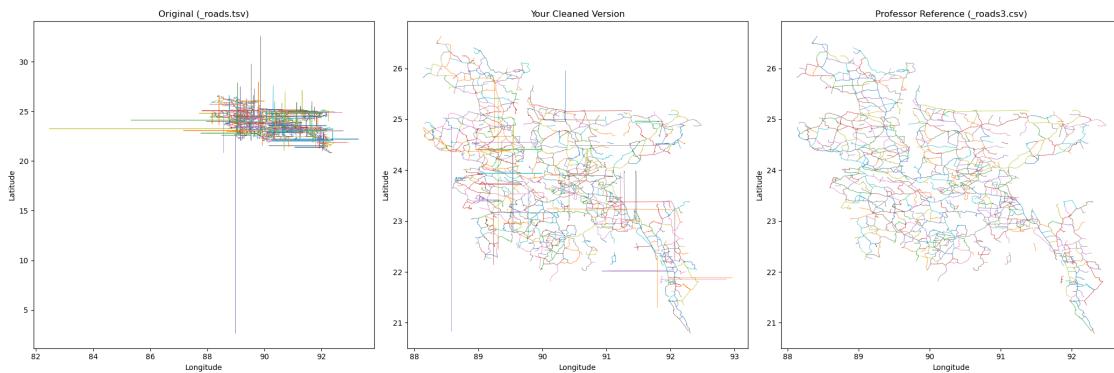
    datasets = [
        (before_df, "Original (_roads.tsv)"),
        (after_df, "Your Cleaned Version"),
        (ref_df, "Professor Reference (_roads3.csv)")]
    ]

    for ax, (df, title) in zip(axes, datasets):
        for road, group in df.groupby("road"):
            ax.plot(group["lon"], group["lat"], linewidth=0.5)

        ax.set_title(title)
        ax.set_xlabel("Longitude")
        ax.set_ylabel("Latitude")

    plt.tight_layout()
    plt.show()

plot_three_all_roads_lines(tidy, cleaned_tidy, roads3_tidy)
```





```
[37]: def plot_road_threeway(road_name):
    b = tidy[tidy["road"] == road_name]
    a = cleaned_tidy[cleaned_tidy["road"] == road_name]
    r = roads3_tidy[roads3_tidy["road"] == road_name]

    fig, axes = plt.subplots(1, 3, figsize=(18,6))

    axes[0].plot(b["lon"], b["lat"], marker="o", markersize=2)
    axes[0].set_title(f"{road_name} Original")

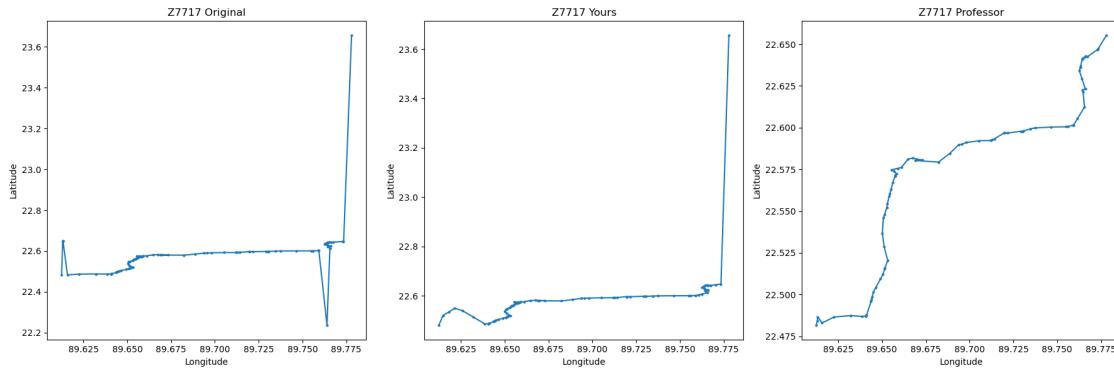
    axes[1].plot(a["lon"], a["lat"], marker="o", markersize=2)
    axes[1].set_title(f"{road_name} Yours")

    axes[2].plot(r["lon"], r["lat"], marker="o", markersize=2)
    axes[2].set_title(f"{road_name} Professor")

    for ax in axes:
        ax.set_xlabel("Longitude")
        ax.set_ylabel("Latitude")

    plt.tight_layout()
    plt.show()

plot_road_threeway("Z7717")
```



```
[35]: def road_jump_summary(df, K=15):
    rows = []
    for r, g in df.groupby("road"):
        pts = g[["lrp", "lat", "lon"]].copy().reset_index(drop=True)
        pts = sort_road_by_lrp(pts)
        seg = compute_segments(pts)[ "seg" ]
        med = seg.median()
        thr = K * med
        rows.append({
            "road": r,
            "n_points": len(pts),
            "median_seg": med,
            "thr": thr,
            "max_seg": float(seg.max()),
            "n_jumps": int((seg > thr).sum())
        })
    return pd.DataFrame(rows).sort_values(["max_seg", "n_jumps"], ↵
                                         ascending=False)

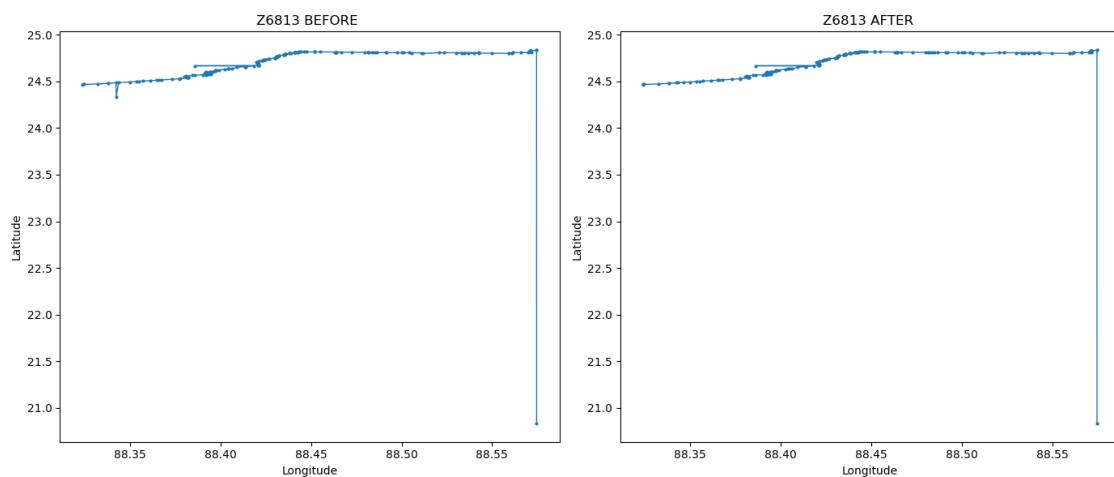
summary_after = road_jump_summary(cleaned_tidy, K=15)
print(summary_after.shape)
summary_after.head(20)
```

(852, 6)

```
[35]:   road n_points median_seg      thr    max_seg  n_jumps
664  Z6813      152  0.373761  5.606408  444.785889      1
558  Z5074       5  0.087344  1.310165  332.667495      1
759  Z7717      82  0.357206  5.358097 112.066939      1
73    N707      20  0.184816  2.772237 111.959663      1
355  Z1611       6  0.398709  5.980628 111.429670      1
521  Z5019      13  0.277046  4.155692 111.318490      1
11    N111      45  0.270846  4.062686 111.278327      1
470  Z3711      60  0.382292  5.734380 110.150946      1
```

244	Z1047	17	0.435002	6.525032	109.213546	1
286	Z1129	10	0.799441	11.991618	103.919081	1
281	Z1124	22	0.283893	4.258393	102.885986	1
820	Z8604	55	0.292797	4.391950	102.850793	1
564	Z5210	64	0.548830	8.232445	101.085729	1
27	N211	4	67.758113	1016.371698	67.758196	0
61	N6	426	0.547285	8.209275	67.338898	9
48	N507	159	0.300839	4.512592	55.446792	6
351	Z1605	78	0.490569	7.358535	51.105264	6
709	Z7451	9	25.726795	385.901928	50.777302	0
234	Z1037	44	0.273308	4.099619	45.093956	10
737	Z7606	58	0.682590	10.238851	42.094823	13

```
[33]: bad_road = summary_after.iloc[0][["road"]]
plot_road_before_after(tidy, bad_road, K=15)
```



```
{'median_seg_before': np.float64(0.38175477867618446), 'thr_before': np.float64(5.7263216801427665), 'jumps_before': 3, 'max_seg_before': 444.7858890252744, 'median_seg_after': np.float64(0.3737605388362736), 'thr_after': np.float64(5.606408082544104), 'jumps_after': 1, 'max_seg_after': 444.7858890252744, 'actions_block': 0, 'actions_smart': 2, 'actions_total': 2}
```