

Supplementary Material for “A Geographically Validated Administrative Wildfire Inventory for Turkey (2013–2025)”, Author: Bikem Ekberzade

```
# =====  
# SUPPLEMENTARY CODE:  
# A Geographically Validated Administrative Wildfire Inventory for Turkey (2013–2025)  
# This script is the companion spatial validation framework for the manuscript submitted to  
# Earth System Science Data (ESSD). It processes the raw OGM wildfire archive by normalizing  
# administrative text, strictly verifying coordinates against civil boundaries, rescuing major  
# spatial anomalies via NASA VIIRS satellite data, and performing guaranteed terrestrial imputation.  
#  
# CITATION INFO: If you use or adapt this code, please cite the corresponding dataset and  
# manuscript: Ekberzade, (2026). A Geographically Validated Administrative Wildfire Inventory for Turkey  
# (2013–2025). Earth System Science Data.  
#  
# Dataset available at: [10.5281/zenodo.21887740].  
#  
# For all inquiries: Bikem Ekberzade; ekberzade19@itu.edu.tr  
# =====
```

```
library(dplyr)  
library(janitor)  
library(lubridate)  
library(stringr)  
library(sf)  
library(purrr)
```

```
# Function to convert Turkish/Special characters to base English (ASCII)
```

```
normalize_tr_chars <- function(text) {  
  text %>%  
    str_to_upper(locale = "tr") %>%          # Safely uppercase first  
    str_replace_all(c(  
      "Ç" = "C", "Ğ" = "G", "İ" = "I", "ı" = "i",  
      "Ö" = "O", "Ş" = "S", "Ü" = "U",  
      "Â" = "A", "Î" = "I", "Û" = "U" # Catches different spellings e.g. LÂPSEKİ -> LAPSEKI  
    )) %>%  
    str_trim() # Removes leading/trailing spaces  
}
```

```
# ----- VIIRS -----
```

```
viirs <- read.csv("D:/Documents/VIIRS/viirs.csv")  
viirs_year <- viirs %>% filter(acq_date >= "2024-01-01" & acq_date < "2026-01-01")  
viirs_year <- viirs_year %>%          #Change the timestamp to POSIXct  
  mutate(acq_date = as.Date(acq_date),  
         acq_datetime = as.POSIXct(acq_datetime))  
viirs_sf <- st_as_sf(viirs_year, coords = c("longitude","latitude"), crs = 4326)
```

```
# ----- ADMINISTRATIVE BOUNDARIES -----
```

```
# Load administrative boundaries - ADM2 district boundaries instead of ADM1 (make sure CRS matches  
# raw OGM data)
```

```
districts_sf <- st_read("D:/Documents/Shapefiles/tur_adm2.shp") %>%  
  st_transform(4326) %>%  
  mutate(  
    # Create the normalized matching columns  
    adm1_sf_match = normalize_tr_chars(adm1_tr),  
    adm2_sf_match = normalize_tr_chars(adm2_tr),
```

Supplementary Material for “A Geographically Validated Administrative Wildfire Inventory for Turkey (2013–2025)”, Author: Bikem Ekberzade

```
# Fix the known shapefile typo for Eskişehir just in case
ad_shp_match = if_else(ad_shp_match == "ESKISEHIR", "ESKISEHIR", ad_shp_match)
)
unique(districts_sf$adm1_en)

# ----- OGM -----

ogm <- read.csv("D:/Documents/OGM/OGM_2024_2025.csv")

#This cleans the name (gets rid of Turkish characters for easy data wrangling)

ogm <- ogm %>% clean_names()
ogm <- ogm %>%
  mutate(
    universal_id = row_number(), # Creates a guaranteed 1, 2, 3... ID

    # This part below has to be manually adjusted to the column formatting of each OGM year/file
    # For 2013-2018: This will successfully read BOTH "02.07.2013 14:05:00" AND "2015-05-20 15:30:00"
    # fire_datetime = parse_date_time(yangin_cikisi, orders = c("dmy_HMS", "ymd_HMS"))
    # For 2019-2020: Use the new column name, and add "dmy" and "ymd" to the fallback list!
    # fire_datetime = parse_date_time(yangin_baslama_zamani, orders = c("dmy_HMS", "ymd_HMS", "dmy",
"ymd"))
    # POST-2021:
    # 1. Paste the two columns (date and time) together with a space in the middle
    # This turns "05.06.2021" and "13:15" into "05.06.2021 13:15"
    temp_datetime_str = paste(baslama_tarihi, baslama_saati),
    # 2. Parse the new combined string!
    # Notice "dmy_HM" is added to handle the missing seconds.
    fire_datetime = parse_date_time(
      temp_datetime_str,
      orders = c("dmy_HM", "ymd_HM", "dmy_HMS", "dmy", "ymd")
    ),
    date = as.Date(fire_datetime)
  )
summary(ogm$date)

# Coordinate converter
dms_to_dd_safe <- function(dms) {
  # match strict DMS pattern
  # The \\D+ acts as a universal separator.
  # It will cleanly extract the numbers from "37°05'27\"", "37:05:27", or even "37 05 27"
  parts <- str_match(dms, "^\\s*(\\d{1,3})\\D+(\\d{1,2})\\D+(\\d{1,2})")
  deg <- as.numeric(parts[,2])
  min <- as.numeric(parts[,3])
  sec <- as.numeric(parts[,4])
  dd <- deg + min/60 + sec/3600
  return(dd)
}

# Convert coordinates
ogm <- ogm %>%
  mutate(lat = dms_to_dd_safe(kuzey), lon = dms_to_dd_safe(dogu)) # Kuzey = North, Dogu (doğu) = East
summary(ogm$lat)
summary(ogm$lon)

# This part is necessary otherwise the sf conversion will not work
```

Supplementary Material for “A Geographically Validated Administrative Wildfire Inventory for Turkey (2013–2025)”, Author: Bikem Ekberzade

```
ogm <- ogm %>%
  mutate(flag_invalid_coord = is.na(lat) | is.na(lon))
ogm_clean <- ogm %>% filter(!flag_invalid_coord)
# only the clean file gets converted
ogm_sf <- st_as_sf(ogm_clean, coords = c("lon","lat"), crs = 4326)

# ----- Townships & Districts -----

ogm_town <- ogm_sf %>%
  mutate(
# For 2013-2018:
    # ili_match = normalize_tr_chars(ili),
    # ilce_match = normalize_tr_chars(ilce),
# For 2019-2025:
    # 1. Base normalization to English characters
    ili_match = normalize_tr_chars(il_adi), #il_adi=province_name, ilce_adi=district_name
    ilce_match = normalize_tr_chars(ilce_adi),
# Additional for 2022-2025:
    # koy_en = normalize_tr_chars(koy),
    # mevkii_en = normalize_tr_chars(mevkii),
    yangin_nedeni_en = normalize_tr_chars(yangin_cikis_nedeni), #yangin_cikis_nedeni=cause_of_fire
    # yangin_tali_en = normalize_tr_chars(yangin_tali_nedeni),
# Normalize the Forestry Hierarchy columns
    isletme_m_match = normalize_tr_chars(isletme_mudurlugu),
#isletme_mudurlugu=operations_directorate
    isletme_s_match = normalize_tr_chars(isletme_sefliqi), #isletme_sefliqi=operations_chiefdom (local)
    # 2. Province Typos (in English alphabet format!)
    ili_match = case_when(
      ili_match == "USAK" ~ "USAK",
      ili_match == "GUMUSHANE" ~ "GUMUSHANE",
      ili_match == "AFYON" ~ "AFYONKARAHISAR",
      ili_match == "K.MARAS" ~ "KAHRAMANMARAS",
      TRUE ~ ili_match
    ),
    # 3. District Typos & The MERKEZ (town center) Fix: ili=province, ilce=district
    ilce_match = case_when(
      # The Universal Merkez Fix
      ilce_match == "MERKEZ" ~ ili_match,
      # The specific spacing/spelling differences (using normalized letters)
      ili_match == "KAHRAMANMARAS" & ilce_match == "ONIKISUBAT" ~ "ONIKI SUBAT",
      ili_match == "HATAY" & ilce_match == "SAMANDAGI" ~ "SAMANDAG",
      ili_match == "ZONGULDAK" & ilce_match == "K.DENIZ EREGLI" ~ "EREGLI",
      ili_match == "BURSA" & ilce_match == "M.KEMALPASA" ~ "MUSTAFAKEMALPASA",
      ili_match == "GAZIANTEP" & ilce_match == "NURDAG" ~ "NURDAGI",
      ili_match == "ISTANBUL" & ilce_match == "GAZIOSMANPASA" ~ "GAZI OSMANPASA",
      ili_match == "ISTANBUL" & ilce_match == "EYUP" ~ "EYUPSULTAN",
      ili_match == "KAHRAMANMARAS" & ilce_match == "ANDIRAN" ~ "ANDIRIN",
      ili_match == "KASTAMONU" & ilce_match == "AGILI" ~ "AGLI",
      ili_match == "KUTAHYA" & ilce_match == "SEPHANE" ~ "SAPHANE",
      ili_match == "MANISA" & ilce_match == "YUNUS EMRE" ~ "YUNUSEMRE",
      ili_match == "YALOVA" & ilce_match == "CIFLIKKOY" ~ "CIFTLIKKOY",
      ili_match == "AFYONKARAHISAR" & ilce_match == "SINCANLI" ~ "SINANPASA",
      ili_match == "ANKARA" & ilce_match == "KAZAN" ~ "KAHRAMANKAZAN",
      ili_match == "AFYONKARAHISAR" & ilce_match == "INCEHISAR" ~ "ISCEHISAR",
      ili_match == "ANTALYA" & ilce_match == "KALE" ~ "DEMRE",
      ili_match == "GIRESUN" & ilce_match == "-" ~ "GIRESUN",
```

Supplementary Material for “A Geographically Validated Administrative Wildfire Inventory for Turkey (2013–2025)”, Author: Bikem Ekberzade

```
ili_match == "HATAY" & ilce_match == "-" ~ "HATAY",
ili_match == "SAMSUN" & ilce_match == "ONDOKUZMAYIS" ~ "19 MAYIS",
ili_match == "TOKAT" & ilce_match == "BAHCECIFTLIK" ~ "BASCIFTLIK",
ili_match == "SIRNAK" & ilce_match == "BEYTISSEBAP" ~ "BEYTUSSEBAP",
ili_match == "EDIRNE" & ilce_match == "SULEOGLU" ~ "SULOGLU",
ili_match == "DENIZLI" & ilce_match == "AKKOY" ~ "PAMUKKALE",
ili_match == "MALATYA" & ilce_match == "ARAPKIR" ~ "ARAPGIR",
ili_match == "SIIRT" & ilce_match == "AYDINLAR" ~ "TILLO",
# Add any other hard-coded spacing typos here...
TRUE ~ ilce_match
)
)

# ----- PRE-JOIN AUDIT: THE DICTIONARY BUILDER -----
# Run this BEFORE the spatial join to catch new typos in each year's data!
# 1. Get unique Province + District from the normalized OGM data

ogm_unique_check <- ogm_town %>%
  st_drop_geometry() %>%
  distinct(ili_match, ilce_match)
# 2. Get unique Province + District from the Shapefile
shp_unique_check <- districts_sf %>%
  st_drop_geometry() %>%
  distinct(ad_shp_match, ilce_shp_match)

# 3. Find the spelling mismatches (In OGM, but missing in the Shapefile)

new_typos_to_fix <- ogm_unique_check %>%
  anti_join(shp_unique_check, by = c("ili_match" = "ad_shp_match", "ilce_match" = "ilce_shp_match"))
%>%
# CRITICAL FILTER: it ignores the Metropolitan "Merkez" conversions (e.g., Hatay-Hatay)
# because we already know those don't exist in the shapefile and are handled by the fallback!
filter(ilce_match != ili_match) %>%
  arrange(ili_match, ilce_match)

# 4. Lists what needs to be added to the dictionary!
# Afterwards return and manually add the corrections to the universal fix above before continuing below
# this requires a double back, but it is safer this way to make sure the algorithm does what it intends to.

print(paste("AUDIT ALERT: Found", nrow(new_typos_to_fix), "unrecognized district spellings!"))
print(new_typos_to_fix)

# ----- Verify the fix
# 1. Get the unique list of Province + District from cleaned OGM data
ogm_unique <- ogm_town %>%
  st_drop_geometry() %>%
  distinct(ili_match, ilce_match)
# 2. Get the unique list of Province + District from Shapefile
shp_unique <- districts_sf %>%
  st_drop_geometry() %>%
  distinct(ad_shp_match, ilce_shp_match)
# 3. Find the Mismatches! (What is in OGM but missing in the Shapefile?)
problem_districts <- ogm_unique %>%
  anti_join(shp_unique, by = c("ili_match" = "ad_shp_match", "ilce_match" = "ilce_shp_match")) %>%
  arrange(ili_match, ilce_match)
# 4. View the culprits
```

Supplementary Material for “A Geographically Validated Administrative Wildfire Inventory for Turkey (2013–2025)”, Author: Bikem Ekberzade

```
print(problem_districts)
```

```
# ----- IDENTIFY THE FAULTY ENTRIES!!!!!!
```

```
# 4. Perform the Spatial Join pulling the uniquely named columns
```

```
ogm_town <- st_join(ogm_town, districts_sf[c("ad_shp_match", "ilce_shp_match")])
```

```
# 5. Calculate the TRUE mismatch rate with the "Lazy Clerk" exception
```

```
ogm_town <- ogm_town %>%
```

```
  mutate(
```

```
    geom_mismatch = case_when(
```

```
      # RULE 1: If the point lands in a completely different province (or the ocean)
```

```
      # This is a major error. Flag as TRUE (Mismatch).
```

```
      is.na(ad_shp_match) | ili_match != ad_shp_match ~ TRUE,
```

```
      # RULE 2: The Metropolitan Exception (The Lazy Clerk)
```

```
      # If the point lands inside the correct province, AND the OGM data just lists
```

```
      # the province name as the district (e.g., Hatay-Hatay), TRUST THE COORDINATE.
```

```
      # Flag as FALSE (Not a mismatch).
```

```
      ili_match == ad_shp_match & ilce_match == ili_match ~ FALSE,
```

```
      # RULE 3: Explicit District Contradictions
```

```
      # If the point is in the right province, but the OGM data explicitly names a DIFFERENT district
```

```
      # (e.g., OGM says Köyceğiz, point is in Milas, both districts in the province of Muğla).
```

```
      # Flag as TRUE (Mismatch).
```

```
      ilce_match != ilce_shp_match ~ TRUE,
```

```
      # RULE 4: Everything matches perfectly
```

```
      TRUE ~ FALSE
```

```
    )
```

```
  )
```

```
# Moment of truth: Check how many mismatches are left now!
```

```
table(ogm_town$geom_mismatch)
```

```
# ----- FIX THE FAULTY ENTRIES
```

```
# 1. Isolate the bad records
```

```
mismatched_fires <- ogm_town %>% filter(geom_mismatch == TRUE)
```

```
print(paste("Attempting to rescue", nrow(mismatched_fires), "fires using VIIRS..."))
```

```
# 2. Run the Expanding Net Rescue Loop
```

```
rescued_locations <- map_dfr(1:nrow(mismatched_fires), function(i) {
```

```
  bad_fire <- mismatched_fires[i, ]
```

```
  target_date <- bad_fire$date
```

```
  target_province <- bad_fire$ili_match
```

```
  target_district <- bad_fire$ilce_match
```

```
  # Step A: Find the strict District polygon
```

```
  dist_poly <- districts_sf %>%
```

```
    filter(ad_shp_match == target_province & ilce_shp_match == target_district)
```

```
  # Step B: Find the broad Province polygon
```

```
  prov_poly <- districts_sf %>%
```

```
    filter(ad_shp_match == target_province) %>%
```

```
    st_union() %>%
```

```
    st_sf()
```

```
  # If the province doesn't exist at all, skip it
```

```
  if(nrow(prov_poly) == 0) return(NULL)
```

```
  # Filter VIIRS by time first (±3 days)
```

```
  viirs_time_filtered <- viirs_sf %>%
```

```
    filter(acq_date >= target_date - 3 & acq_date <= target_date + 3)
```

```
  viirs_candidates <- NULL
```

```
  # ATTEMPT 1: The High-Fidelity District Search
```

```
  if(nrow(dist_poly) > 0) {
```

```
    viirs_candidates <- viirs_time_filtered %>% st_intersection(dist_poly)
```

Supplementary Material for “A Geographically Validated Administrative Wildfire Inventory for Turkey (2013–2025)”, Author: Bikem Ekberzade

```
}
# ATTEMPT 2: The 10km Buffer Fallback (PREVENTS 24KM JUMPS!)
# If Attempt 1 failed, but we DO have a district polygon, buffer it by 10km (10,000 meters)
if((is.null(viirs_candidates) || nrow(viirs_candidates) == 0) && nrow(dist_poly) > 0) {
  buffered_poly <- st_buffer(dist_poly, dist = 10000)
  viirs_candidates <- viirs_time_filtered %>% st_intersection(buffered_poly)
}
# ATTEMPT 3: The Metropolitan Fallback
# ONLY if the specific district polygon doesn't exist (e.g. "HATAY - HATAY") do we search the whole province
if((is.null(viirs_candidates) || nrow(viirs_candidates) == 0) && nrow(dist_poly) == 0) {
  viirs_candidates <- viirs_time_filtered %>% st_intersection(prov_poly)
}
# If STILL no fires found anywhere within those limits, we cannot rescue it
if(is.null(viirs_candidates) || nrow(viirs_candidates) == 0) return(NULL)
# If we found a match, grab the strongest one
best_viirs_match <- viirs_candidates %>%
  arrange(desc(frp)) %>%
  slice(1)
# Return the newly discovered coordinates!
data.frame(
  universal_id = bad_fire$universal_id,
  original_ili = target_province,
  original_ilce = target_district,
  fire_date = target_date,
  rescued_lon = st_coordinates(best_viirs_match)[,1],
  rescued_lat = st_coordinates(best_viirs_match)[,2],
  viirs_frp = best_viirs_match$frp,
  viirs_date = best_viirs_match$acq_date
)
})
# 3. See how many we successfully found!
print(paste("Successfully rescued", nrow(rescued_locations), "fires!"))
head(rescued_locations)
# ----- STICH UP THE CORRECTED ENTRIES BACK INTO THE DATASET
# 1. Safely extract the original coordinates from the geometry column first!
ogm_town <- ogm_town %>%
  mutate(
    lon = st_coordinates(geometry)[,1],
    lat = st_coordinates(geometry)[,2]
  )
# 2. Drop the geometry to make merging easy
ogm_df <- st_drop_geometry(ogm_town)
# 3. Join the rescued data based on the unique ID (s_no)
ogm_merged <- ogm_df %>%
  left_join(
    rescued_locations %>% select(universal_id, rescued_lon, rescued_lat, viirs_frp),
    by = "universal_id"
  )
# 4. Overwrite the bad coordinates with the rescued ones
ogm_clean_final <- ogm_merged %>%
  mutate(
    # If we have a rescued longitude, use it. Otherwise, keep the original.
    final_lon = if_else(!is.na(rescued_lon), rescued_lon, lon),
    final_lat = if_else(!is.na(rescued_lat), rescued_lat, lat),
    # Create the flag so you always know which ones VIIRS saved
    data_source = if_else(!is.na(rescued_lon), "VIIRS_Rescued", "OGM_Original")
  )
```

Supplementary Material for “A Geographically Validated Administrative Wildfire Inventory for Turkey (2013–2025)”, Author: Bikem Ekberzade

```
)
# 5. Separate the unrescued fires for manual checking later
unrescued_orphans <- ogm_clean_final %>%
  filter(geom_mismatch == TRUE & is.na(rescued_lon))
# 6. Convert the clean dataset back into a spatial shapefile
ogm_final_sf <- st_as_sf(
  ogm_clean_final %>% filter(!(geom_mismatch == TRUE & is.na(rescued_lon))),
  coords = c("final_lon", "final_lat"),
  crs = 4326
)
print(paste("Final clean dataset has", nrow(ogm_final_sf), "records ready for analysis!"))
print(paste("There are", nrow(unrescued_orphans), "records completely lost/unverifiable."))

# ----- SAVE THE CLEAN RECORDS
# 1. Export as a modern GeoPackage (Highly recommended for ArcGIS)
st_write(ogm_final_sf, "OGM_2024_2025_Cleaned.gpkg", append = FALSE)
# OR 2. Export as a classic Shapefile (if using an older version of ArcMap)
st_write(ogm_final_sf, "OGM_2024_2025_Cleaned.shp", append = FALSE)
# ----- FOR CSV
# 1. Create a dataframe that exactly matches the shapefile (dropping the orphans)
ogm_csv_ready <- ogm_clean_final %>%
  filter(!(geom_mismatch == TRUE & is.na(rescued_lon)))
# 2. Export the clean CSV
write.csv(ogm_csv_ready, "OGM_2024_2025_Cleaned.csv", row.names = FALSE)
# 3. BONUS: Export the lost records so there is now a record of what was dropped
write.csv(unrescued_orphans, "OGM_2024_2025_Orphans.csv", row.names = FALSE)

# ----- THE ORPHAN DISTANCE ANALYZER
# 1. Ensure the orphans dataset is a spatial object using the original coordinates
orphans_sf <- st_as_sf(unrescued_orphans, coords = c("lon", "lat"), crs = 4326, remove = FALSE)
print(paste("Analyzing distance for", nrow(orphans_sf), "orphan records..."))

# 2. Run the Distance Loop with Coastal Snapping

orphan_analysis <- map_dfr(1:nrow(orphans_sf), function(i) {

  bad_fire <- orphans_sf[i, ]
  target_prov <- bad_fire$ili_match
  target_dist <- bad_fire$ilce_match
  # Find the intended district polygon
  target_poly <- districts_sf %>%
    filter(ad_shp_match == target_prov & ilce_shp_match == target_dist)
  # Fallback: If it's a metropolitan "Merkez" that didn't match, use the whole province
  if(nrow(target_poly) == 0) {
    target_poly <- districts_sf %>%
      filter(ad_shp_match == target_prov) %>%
      st_union() %>%
      st_sf()
  }

  # If the polygon STILL doesn't exist (e.g., total typo in the OGM data), flag it
  if(nrow(target_poly) == 0) {
    return(data.frame(
      universal_id = bad_fire$universal_id,
      error_dist_km = NA,
      orphan_status = "Polygon Not Found - Unverifiable",
```

Supplementary Material for “A Geographically Validated Administrative Wildfire Inventory for Turkey (2013–2025)”, Author: Bikem Ekberzade

```
  snapped_lon = NA, snapped_lat = NA, coastal_adjustment = FALSE
))
}

# 3. Calculate the shortest distance from the point to the intended polygon boundary
# st_distance returns meters. We divide by 1000 to get kilometers.

dist_km <- as.numeric(st_distance(bad_fire, target_poly)) / 1000
is_sea <- is.na(bad_fire$ad_shp_match)

# 4. Correct for coastlines and prevent fire landing in sea

status <- case_when(
  # TRUE BORDER DISPUTE: Within 10km AND it landed on valid land (another district)
  dist_km <= 10 & !is_sea ~ "Border Dispute (Acceptable)",
  # MARINE DRIFT: Within 10km BUT it landed in the water
  dist_km <= 10 & is_sea ~ "Coastal Drift (Snapped)",
  # MAJOR ERROR: Too far away regardless of where it landed
  TRUE ~ "Major Spatial Error (Reject)"
)

# THE SNAP: If it's in the sea, pull it to the nearest shoreline
snap_lon <- NA
snap_lat <- NA
coast_adj <- FALSE

if(status == "Coastal Drift (Snapped)") {
  nearest_geom <- st_nearest_points(bad_fire, target_poly)
  snapped_coords <- st_coordinates(st_cast(nearest_geom, "POINT"))[2, ]
  snap_lon <- snapped_coords[1]
  snap_lat <- snapped_coords[2]
  coast_adj <- TRUE # Flag it for the new column
}

# Return the results
data.frame(
  universal_id = bad_fire$universal_id,
  error_dist_km = round(dist_km, 2),
  orphan_status = status,
  snapped_lon = snap_lon,
  snapped_lat = snap_lat,
  coastal_adjustment = coast_adj
)
})

# 3. Join the analysis back to your orphans dataframe
unrescued_orphans <- unrescued_orphans %>%
  left_join(orphan_analysis, by = "universal_id")

# 4. See the results!
table(unrescued_orphans$orphan_status)

# Add the FALSE flag to the already clean records so columns match perfectly
ogm_csv_ready <- ogm_csv_ready %>%
  mutate(coastal_adjustment = FALSE)
```

Supplementary Material for “A Geographically Validated Administrative Wildfire Inventory for Turkey (2013–2025)”, Author: Bikem Ekberzade

Append the final file with the rescued records, but flag the rescued ones to be able to distinguish

1. Isolate Acceptable Border Disputes (On Land)

```
acceptable_orphans <- unrescued_orphans %>%
  filter(orphan_status == "Border Dispute (Acceptable)") %>%
  mutate(
    data_source = "OGM_Border_Accepted",
    final_lon = lon,
    final_lat = lat) %>%
  select(-error_dist_km, -orphan_status, -snapped_lon, -snapped_lat)
```

2. Isolate and apply the Snapped Coastal Points

```
snapped_orphans <- unrescued_orphans %>%
  filter(orphan_status == "Coastal Drift (Snapped)") %>%
  mutate(
    # Keep identical to your existing category to protect the legend!
    data_source = "OGM_Border_Accepted",
    final_lon = snapped_lon,
    final_lat = snapped_lat
  ) %>% select(-error_dist_km, -orphan_status, -snapped_lon, -snapped_lat)
```

3. Isolate the TRUE Major Errors (The trash pile)

```
rejected_orphans <- unrescued_orphans %>%
  filter(orphan_status == "Major Spatial Error (Reject)")
```

4. Append BOTH the acceptable and snapped ones back into your main clean dataset

```
final_master_dataset <- bind_rows(ogm_csv_ready, acceptable_orphans, snapped_orphans) %>%
  # Rename the original coordinate columns so ArcGIS doesn't auto-map them by mistake!
  rename(original_lon = lon, original_lat = lat)
```

4. Verify the row counts to ensure everything adds up

```
print(paste("Final Master Dataset has", nrow(final_master_dataset), "records.))
print(paste("Rejected / Lost records:", nrow(rejected_orphans)))
```

5. Save the ultimate files!

The Master File (Use this for all your future mapping and clustering)

```
write.csv(final_master_dataset, "OGM_2024_2025_Master_Clean.csv", row.names = FALSE)
```

The Audit File (Keep this just to know what you threw away and why)

```
write.csv(rejected_orphans, "OGM_2024_2025_Rejected_Errors.csv", row.names = FALSE)
```

=====

THE FINAL RESCUE: ON-LAND-SURFACE IMPUTATION FOR MAJOR ERRORS

** THIS IS THE LAST RESORT! **

Only genuinely rejected spatial errors are imputed. OGM_Border_Accepted records retain their
original coordinates. Each imputed record receives its own on-land coordinate within its intended
district polygon.

=====

```
print(paste("Calculating on-land-surface Imputation points for", nrow(rejected_orphans), "Major Errors..."))
```

1. Join the rejected orphans with their original fire IDs to their intended district polygons

```
imputed_orphans <- rejected_orphans %>%
  inner_join(districts_sf %>% select(ad_shp_match, ilce_shp_match, geometry),
    by = c("ili_match" = "ad_shp_match", "ilce_match" = "ilce_shp_match")) %>%
  st_as_sf()
```

2. Generate exactly one random point INSIDE the polygon for each record to prevent stacking

```
random_points <- st_sample(imputed_orphans, size = rep(1, nrow(imputed_orphans)),
```

Supplementary Material for “A Geographically Validated Administrative Wildfire Inventory for Turkey (2013–2025)”, Author: Bikem Ekberzade

```
type = "random", exact = TRUE)
```

2. Assign the new random geometries back to the dataset

```
safe_imputed_points <- imputed_orphans %>% st_set_geometry(random_points)
```

3. Extract the new coordinates and apply the safety flag!

```
final_imputed_records <- safe_imputed_points %>%  
  mutate(  
    final_lon = st_coordinates(geometry)[,1],  
    final_lat = st_coordinates(geometry)[,2],  
    data_source = "Imputed_On_Land_Surface", # Original category preserved!  
    coastal_adjustment = FALSE # Add the new column flag  
  ) %>%  
  rename(  
    original_lon = lon,  
    original_lat = lat  
  ) %>%  
  st_drop_geometry() %>% select(-error_dist_km, -orphan_status, -snapped_lon, -snapped_lat)
```

4. Append these finalized imputed records to the Master Dataset

```
ultimate_master_dataset <- bind_rows(final_master_dataset, final_imputed_records)
```

5. Save the ULTIMATE Master File

```
write.csv(ultimate_master_dataset, "OGM_2024_2025_Master_Clean_fin.csv", row.names = FALSE)  
print(paste("Imputation Complete with land correction! Ultimate Master Dataset has",  
  nrow(ultimate_master_dataset), "records."))
```

#----- BONUS: QC – FINAL COUNT

```
cat("FINAL QUALITY CONTROL\n")
```

1. Provenance counts

```
print(table(ultimate_master_dataset$data_source, useNA = "ifany"))
```

2. Check that no rejected records remain

```
print(paste("Rejected records:", nrow(rejected_orphans)))
```

3. Check that all final records have coordinates

```
print(paste("Records with missing final longitude:", sum(is.na(ultimate_master_dataset$final_lon))))  
print(paste("Records with missing final latitude:", sum(is.na(ultimate_master_dataset$final_lat))))
```

```
cat("\nEND OF QUALITY CONTROL")
```