```
# ============================================================
# County-level daily peaks + systemwide daily totals (EAGLE-I)
# ============================================================

suppressPackageStartupMessages({
  library(data.table)
  library(fasttime)   # fastPOSIXct
})

# ----------------------------
# Inputs / Outputs
# ----------------------------
infile  <- "eaglei_outages_2014_2024_miso_filtered.csv"   # <-- set your path if needed
out_cnty <- "county_daily_peak.csv"
out_sys  <- "daily_total_peak.csv"

# ----------------------------
# Column mapping (robust)
# ----------------------------
hdr <- fread(infile, nrows = 0)
names_raw <- names(hdr)
names_lc  <- tolower(trimws(names_raw))

find_col <- function(cands, required = TRUE) {
  i <- which(names_lc %in% tolower(cands))
  if (length(i) == 0L) {
    if (!required) return(NA_character_)
    stop(sprintf(
      "Could not find column. Tried: %s\nAvailable:\n- %s",
      paste(cands, collapse = ", "), paste(names_raw, collapse = "\n- ")
    ))
  }
  names_raw[i[1]]
}

col_fips  <- find_col(c("fips_code","geoid","county geoid","fips","county_fips","fips code"))
col_cty   <- find_col(c("county","county_name","county name"))
col_state <- find_col(c("state","state_name","state_abbr","stusps","state code"))
col_cust  <-
find_col(c("customers_out","customers_affected","customers","max_customers_out"))
col_time  <-
find_col(c("run_start_time","time","timestamp","datetime","date_time","ts","interval_start","st
art_time"))
```

```
# ----------------------------
# Read only needed columns
# ----------------------------
dt <- fread(
  infile,
  select = c(col_fips, col_cty, col_state, col_cust, col_time),
  showProgress = TRUE
)
setnames(dt, c("fips_code","county","state","customers_out","run_start_time"))

# ----------------------------
# Types / parsing
# ----------------------------
# FIPS as 5-char strings
dt[, fips_code := sprintf("%05s", as.character(fips_code))]

# Coerce customers to numeric (integer-safe)
suppressWarnings(dt[, customers_out := as.numeric(customers_out)])

# Parse timestamps (input like 2020-08-14T03:15:00Z)
# fastPOSIXct prefers "YYYY-mm-dd HH:MM:SS"; do a minimal transform
ts_str <- gsub("Z$", "", gsub("T", " ", dt$run_start_time, fixed = TRUE))
dt[, ts := fasttime::fastPOSIXct(ts_str, tz = "UTC")]
if (dt[, any(is.na(ts))]) {
  # fallback parser if any failed
  dt[, ts := as.POSIXct(ts_str, tz = "UTC")]
}
# Drop rows with missing essentials
dt <- dt[!is.na(ts) & !is.na(customers_out) & !is.na(fips_code)]

# Calendar date in UTC (adjust if you want local time)
dt[, date := as.Date(ts, tz = "UTC")]

# ----------------------------
# County-level daily peaks
# ----------------------------
# For every (date, county): max customers_out across 15-min intervals that day
county_daily <- dt[
  , .(county_daily_peak = max(customers_out, na.rm = TRUE)),
  by = .(date, fips_code, county, state)
][order(date, state, county)]

# ----------------------------
# Systemwide daily totals (sum of county peaks per day)
```

```
# ----------------------------
daily_total <- county_daily[
  , .(daily_total_customers = sum(county_daily_peak, na.rm = TRUE)),
  by = .(date)
][order(date)]

# ----------------------------
# Write results
# ----------------------------
fwrite(county_daily, out_cnty)
fwrite(daily_total, out_sys)

cat(sprintf("✅ Wrote %s  (rows: %d)\n", out_cnty, nrow(county_daily)))
cat(sprintf("✅ Wrote %s  (rows: %d)\n", out_sys,  nrow(daily_total)))
```