**How was apparent temperature calculated for this analysis?**

Data:

1. Gridded Surface Meteorological Dataset (GridMET): <https://developers.google.com/earth-engine/datasets/catalog/IDAHO_EPSCOR_GRIDMET>

Step 1: Determine a point for each zipcode from which to extract GridMET data.

1. Data Sources:
   1. Population weighted centroid point shapefile: <https://hudgis-hud.opendata.arcgis.com/datasets/HUD::zip-code-population-weighted-centroids/about>
   2. ZCTA polygon shapefile: <https://www.census.gov/cgi-bin/geo/shapefiles/index.php?year=2020&layergroup=ZIP+Code+Tabulation+Areas>
   3. GEE Code to download raster of GridMET grid:

var test\_image=

ee.Image('IDAHO\_EPSCOR/GRIDMET/20160101').select(['tmmx']);

Export.image.toDrive({

image:test\_image,

description:'export\_1',

folder:'gridmet',

fileNamePrefix:'export\_1',

scale:4000

});

* 1. Coastline polyline shapefile: <https://searchworks.stanford.edu/view/xv279yj9196>
  2. Ocean polygon shapefile: <https://geodata.library.ucsb.edu/catalog/3853-s3_2002_s3_reg_pacific_ocean>

1. Upload population weighted centroid file and ZCTA polygon file into ArcGIS Pro. For zipcodes with a population weighted centroid (Data source 1), use the population weighted centroid.
2. For zipcodes without a population weighted centroid, generate a geometric centroid (center of the area) in ArcGIS using Data source 2.
3. For zipcodes with a population weighted centroid or geometric centroid in a GridMET grid cell (Data source 3) that overlaps >20% with the ocean (Data sources 3&4), move the zipcode point to the nearest inland grid cell.
   1. After consultation with Dr. Francesca de’Donato, these points were moved because since ocean temperatures are lower, temperatures taken from GridMET grid cells (4km resolution) that overlap with the ocean may be skewed lower and not reflect the actual temperatures experienced in those zipcodes.
4. Create a final file of one point for each zipcode from which to extract the GridMET temperature data – this includes 1670 population weighted centroids (76 of which were moved) and 131 geometric centroids (5 of which were moved).

Step 2: Extract GridMET data (daily AT) by zipcode for all study years in Google Earth Engine.

1. Data Sources:
   1. Apparent Temperature calculation: <https://ehp.niehs.nih.gov/action/downloadSupplement?doi=10.1289%2Fehp.1409119&file=ehp.1409119.s001.acco.pdf> (S1, page 9)
      1. <https://www.wpc.ncep.noaa.gov/html/heatindex_equation.shtml>
   2. Zipcode point file: “all\_points.shp” (calculated in Step 1)
2. GEE code:

/\*\*\*\* Start of imports. If edited, may not auto-convert in the playground. \*\*\*\*/

var table2 = ee.FeatureCollection("projects/ee-sydneymontesano/assets/all\_points");

var CAboundary = ee.FeatureCollection("projects/ee-sydneymontesano/assets/ca-state-boundary");

/\*\*\*\*\* End of imports. If edited, may not auto-convert in the playground. \*\*\*\*\*/

//Temp (Raster)

var temps = ee.ImageCollection("IDAHO\_EPSCOR/GRIDMET").select("tmmx","rmax","tmmn","rmin")

.filter(ee.Filter.date('2008-01-01', '2009-01-01'));

//Map.addLayer(temps, null, 'gridmet temp and rh'); //review in inspector but easier with more limited dates

//Add heat index variable - https://ehp.niehs.nih.gov/action/downloadSupplement?doi=10.1289%2Fehp.1409119&file=ehp.1409119.s001.acco.pdf

var to\_heatindex = function(img){

img = img.set({"date":img.get("system:index")})

//convert kelvin to fahrenheit

img = img.addBands(img.select("tmmx").expression("(b(0) - 273.15) \* (9/5) + 32"))

img = img.addBands(img.select("tmmn").expression("(b(0) - 273.15) \* (9/5) + 32"))

//heat index for max temp and rh

img = img.select(["tmmx","rmax","tmmn","rmin","tmmx\_1","tmmn\_1"],["tmmx","rmax","tmmn","rmin","tmmx\_fahrenheit","tmmn\_fahrenheit"])

img = img.addBands(img.expression(

'(0.5\*(T+61.0+((T-68)\*1.2)+RH\*0.094)+T)/2 < 80 ? 0.5\*(T+61.0+((T-68)\*1.2)+RH\*0.094) : RH>85 && T<87 && T>80 ? -42.379 + 2.04901523\*T + 10.14333127\*RH - .22475541\*T\*RH - .00683783\*T\*T - .05481717\*RH\*RH + .00122874\*T\*T\*RH + .00085282\*T\*RH\*RH - .00000199\*T\*T\*RH\*RH + (((RH-85)/10) \* (87-T)/5) : RH<13 && T>80 && T<112 ? -42.379 + 2.04901523\*T + 10.14333127\*RH - .22475541\*T\*RH - .00683783\*T\*T - .05481717\*RH\*RH + .00122874\*T\*T\*RH + .00085282\*T\*RH\*RH - .00000199\*T\*T\*RH\*RH - (((13-RH)/4)\*((17-(((T-95.)\*\*2)\*\*(1/2)))/17)\*\*(1/2)) : -42.379 + 2.04901523\*T + 10.14333127\*RH - .22475541\*T\*RH - .00683783\*T\*T - .05481717\*RH\*RH + .00122874\*T\*T\*RH + .00085282\*T\*RH\*RH - .00000199\*T\*T\*RH\*RH',

{'T': img.select('tmmx\_fahrenheit'),

'RH': img.select('rmin')}

))

//repeat for min

img = img.addBands(img.expression(

'(0.5\*(T+61.0+((T-68)\*1.2)+RH\*0.094)+T)/2 < 80 ? 0.5\*(T+61.0+((T-68)\*1.2)+RH\*0.094) : RH>85 && T<87 && T>80 ? -42.379 + 2.04901523\*T + 10.14333127\*RH - .22475541\*T\*RH - .00683783\*T\*T - .05481717\*RH\*RH + .00122874\*T\*T\*RH + .00085282\*T\*RH\*RH - .00000199\*T\*T\*RH\*RH + (((RH-85)/10) \* (87-T)/5) : RH<13 && T>80 && T<112 ? -42.379 + 2.04901523\*T + 10.14333127\*RH - .22475541\*T\*RH - .00683783\*T\*T - .05481717\*RH\*RH + .00122874\*T\*T\*RH + .00085282\*T\*RH\*RH - .00000199\*T\*T\*RH\*RH - (((13-RH)/4)\*((17-(((T-95.)\*\*2)\*\*(1/2)))/17)\*\*(1/2)) : -42.379 + 2.04901523\*T + 10.14333127\*RH - .22475541\*T\*RH - .00683783\*T\*T - .05481717\*RH\*RH + .00122874\*T\*T\*RH + .00085282\*T\*RH\*RH - .00000199\*T\*T\*RH\*RH',

{'T': img.select('tmmn\_fahrenheit'),

'RH': img.select('rmax')}

))

//rename bands

return img.select(["tmmx","rmax","tmmn","rmin","tmmx\_fahrenheit","tmmn\_fahrenheit","constant","constant\_1"],

["tmmx","rmax","tmmn","rmin","tmmx\_fahrenheit","tmmn\_fahrenheit","heat\_index\_max","heat\_index\_min"])

}

temps = temps.map(to\_heatindex) //.map applies function over each image

print(temps);

//Compute average heat index

var temps = temps.map(function(img){

var avg = img.select('heat\_index\_max').add(img.select("heat\_index\_min")).divide(2);

return img.addBands(avg)

})

//Apply name "heat\_index\_avg" to band and remove superfluous bands to save memory

temps = temps.select(["heat\_index\_max","heat\_index\_min","heat\_index\_max\_1"],

["heat\_index\_max","heat\_index\_min","heat\_index\_avg"])

print(temps);

// Set point shapefile

var point = table2.filterBounds(CAboundary);

print('point', point)

//Loading study area

Map.centerObject(point);

Map.addLayer(point,{},"point");

var img\_col = temps

.filterDate('2008-1-1', '2009-1-1')

.filterBounds(point)

.select("heat\_index\_max")

var tar\_img = img\_col.toBands()

// Get the temperature at the point

var new\_point = tar\_img.sampleRegions({

collection: point,

properties: ["ZIP", "heat\_index\_max"],

scale: 4638.3,

tileScale:16,

geometries: true

});

print('new\_point',new\_point)

//Export to CSV

Export.table.toDrive({

collection:new\_point,

description: "new\_point",

fileNamePrefix: "new\_point",

fileFormat: "CSV"

});

/\*\*/

//Function

function L\_clip (img){return ee.Image(img).clip(point).subtract(273.15)}

* 1. This code outputs a CSV file with daily max AT for each zipcode for 1 year
  2. Repeat for each year in the study period
  3. Repeat using daily min AT for each year in the study period

Step 3: Compile daily max/min AT for all study years into clean, usable files.

1. Data Sources:
   1. Folder with 1 CSV per year of daily max AT for each zipcode: “GridMET\_MaxDailyHI\_1801ZIPS” (generated in Step 2)
   2. Folder with 1 CSV per year of daily min AT for each zipcode: “GridMET\_MinDailyHI\_1801ZIPS” (generated in Step 2)
2. R code:

```{r setup, include=FALSE}

knitr::opts\_chunk$set(echo = TRUE)

install.packages("readr")

install.packages("dplyr")

install.packages("vctrs")

install.packages("knitr")

install.packages("writexl")

install.packages("lubridate")

install.packages("openair")

library(writexl)

library(readxl)

library(dplyr)

library(readr)

library(vctrs)

library(openair)

library(plyr)

library(data.table)

library(tidyr)

# Temp data pathway

MaxHI <- read\_excel("/Users/sydneymontesano/Desktop/GridMET\_MaxDailyHI\_1801ZIPS/2019.xlsx")

# This code formats column 157 to lat/long

items\_to\_delete <- c('\\{"geodesic":false,"type":"Point","coordinates":\\[', '\\]}')

MaxHI$.geo <- gsub(paste(items\_to\_delete, collapse = "|"), "", MaxHI$.geo)

# Split the ".geo" column by comma and create "latitude" and "longitude" columns

MaxHI <- separate(MaxHI, .geo, into = c("longitude", "latitude"), sep = ",")

head(subset(MaxHI[,c(366:369)]))

# All dates into a single column

MaxHI\_long <-

pivot\_longer(

MaxHI,

cols = '20190101\_heat\_index\_max':'20191231\_heat\_index\_max',

names\_to = "DailyHI",

values\_to = "Max\_HI\_Value"

)

#Reformat the dates

MaxHI\_long$date <- as.POSIXct(strptime(MaxHI\_long$DailyHI, format='%Y%m%d', tz='GMT'))

MaxHI\_long <- selectByDate(MaxHI\_long, start = "2019-05-01", end = "2019-10-31")

# Output file

write.csv(MaxHI\_long, file = '/Users/sydneymontesano/Desktop/MaxDailyHI\_reformatted\_20240402/2019.csv')

# Combine all the year files into one

folder\_path4 <- "/Users/sydneymontesano/Desktop/MaxDailyHI\_reformatted\_20240402"

# List all files in the folder

file\_paths4 <- list.files(path = folder\_path4, pattern = "\\.csv$", full.names = TRUE)

# Function to read and process each file

process\_file <- function(folder\_path4) {

df <- read\_csv(folder\_path4, show\_col\_types = FALSE)

}

# Process all files and store the results in a list

processed\_dfs <- lapply(file\_paths4, process\_file)

# If you want to combine all processed dataframes into one dataframe

MaxHI\_2008\_2019 <- do.call(rbind, processed\_dfs)

# Add column for month-zip

MaxHI\_2008\_2019$month\_zip <- paste(format(as.Date(MaxHI\_2008\_2019$date), "%m"), MaxHI\_2008\_2019$ZIP, sep = "\_")

write.csv(MaxHI\_2008\_2019, file = '/Users/sydneymontesano/Desktop/MaxHI\_2008\_2019.csv')

```

* 1. This code outputs a CSV file with the maximum daily AT for every zipcode on every day in the study period.
  2. Repeat for min AT.