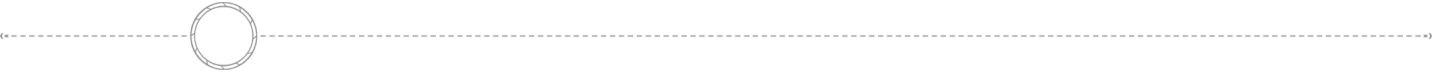
**NASA DEVELOP National Program**



City of Tempe HUE Group and Arizona State University

*Fall 2020*

Tempe Urban Development II

Establishing an Urban Heat Exposure Score for Infrastructure Prioritization in Tempe, Arizona, Using NASA Earth Observations and LiDAR

 **Code Tutorial**

Final Draft – November 12, 2020

Sydney Boogaard (Project Lead)

John Dialesandro

Blake Steiner (Data and Code Lead)

Anson Pang

Dr. David Hondula, Arizona State University (Science Advisor)

Point of Contact: Blake Steiner, blake.a.steiner@gmail.com

Please use somewhere in the subject line: DEVELOP FALL 2020.

**Brief Background**

There are over two billion people in arid regions around the world facing the brunt of increasing temperatures and climate uncertainty. About 200,000 of them call Tempe, Arizona home. Local climate records in Tempe have demonstrated the summer of 2020 shattered the previous record of days exceeding 43.4° C (110.0° F). Increasing temperatures cause multiple concerns, such as higher energy and water costs, lower quality of life, and health consequences for residents. As part of Tempe’s 2020 climate action plan, Tempe is looking to address this concern of urban heat. To do so, they partnered with NASA DEVELOP program, a program that builds capacity in applied sciences professionals to use NASA Earth Observations and other data to help address local problems. Two DEVELOP terms were completed with Tempe. This code tutorial will focus on products made in the second term.

In the second term, carried out in Fall, 2020, Tempe requested maps and data to help them prioritize areas of concern. Thus, this code tutorial will walkthrough how to collect and process the data to create these maps. The tutorial will go line by line, describing what is being done in the code and why it is being done. It will also contain suggestions for improvement and some debugging tips with some errors encountered along the way. It is strongly advised that, if it is your first time using the Google Earth Engine or RStudio platforms, please follow the tutorial in order. Without further a due, let the tutorialing commence!

# Setup

## Google Earth Engine (GEE)

### Sign up for GEE (for free!)

#### https://earthengine.google.com

#### If you have a .edu email, you may get access faster than not. It may take up to a few weeks.

#### It is easier to back up and save data with a gmail account or an edu account that uses gmail for google drive.

#### Be sure to check your spam email!

#### This will be for processing the geospatial data

#### Download Tempe Census Tract boundaries https://data.tempe.gov/datasets/select-demographics-acs-2012-2016-census-2010-tempe-tracts?geometry=-112.194%2C33.292%2C-111.662%2C33.493

## R and RStudio

### You need to install both R and the RStudio platform version 4.0.3.

#### A tutorial on how to install them can be found here: <https://rstudio-education.github.io/hopr/starting.html>

#### R will be used for getting demographic data and statistical analysis

# GEE

## Running \_SatAgg script to create multi-year composite image for UHEAT scores

### Upload the study area shape file into the assets tab on the left hand side

### Go to scripts and, if you have not already, create a new script by clicking on new then file.

### Copy and paste the \_SatAggData text file into the new script file

### Hit Save

### Click on assets, click on the study area asset file you uploaded earlier, then click import.

### At the top of your script you will see imports. Change the import name from “table” to w/e you like.

#### I went with tempe\_ct for tempe\_census tracts.

### Now fill out the parameters between the //NOTE: and the EXTRACT DATA section.

#### Start\_year is the first year you want to consider. Bear in mind that satellite platforms only go back so far. For example, Landsat 8, where most of the data comes from, only goes back to 2013. If you want to go back further, there are ways to do that in GEE.

#### End year is the last year you want to consider when making the composite.

#### Start day of year and end day of year are the days of the year you want to consider imagery for. This was added so you could make a composite of just June from, say 2015-2020 and compare that to composites of other months throughout the years for a finer analysis.

#### Save is a true or false variable in case you want to download the data. This will port to a folder in your google Drive.

#### The output\_dir is the folder you want to save the data to in your google drive.

#### Geography is part of the naming convention. In this case it was named Tempe.

#### Studyoutline is a key variable that should be made the same name as what you named the shapefile asset import at the top. Print studyoutline is right below so you can see if GEE is reading it in the console.

#### Display can be either true or false. If true, it will load up maps of the data for you to see in the map. If not, it saves time.

### The next major step is to check the file path of the var Retrieve\_ functions.

#### Let’s take var Retrieve\_LST for example. It is equal to require(‘users/\*\*\*/\*\*\*./’). This path must be corrected.

#### To do this, make a script for each of the Retrieve\_\*\*\* text files in the same folder you found the \_SatAggData text file. Name them as “Retrieve\_variable”.

#### Then, in the parantheses for the require( ), put the path where those scripts are so GEE can call them in. For example, mine would be: require(‘users/Steiner/Retrieve\_LST’)

#### Once this is done, the hard part is over!

### You are now ready to run the \_AggSatData script in which the CSV output is needed to make the UHEAT score maps. For 10, I am going to go over a bit more in depth of what each section of code does for this script.

### \_AggSatData code context

#### At line 92, this is used to filter all the Landsat 8 imagery into the year range you specified above. This helps narrow down the imagery.

#### Line 95 does the same as above, but for the day of year range you entered earlier so it knows what images to look within the given year range.

#### Lines 98 and 99 gets the current date on your computer and makes that as part of the export files’ names so you can keep track of when your outputs were made.

#### Lines 105 – 115 are where the magic happens. After calling your functions earlier with the require, it is here that those functions are put to work. When each calculation goes through, it ends a “band” to the image collection it is using. If you wanted to change which satellites the data comes from, you can do that by changing the underlying functions in their respective scripts. However, please make a copy first before changing everything! Also note that the functions calculate the median pixel value across the time ranges you specified. This was done because there was not enough images per year in that day of year range to take a proper mean.

#### Lines 124 – 130 takes the variables you just calculated and puts them into a new group called combinedBands. The print(combinedBands) on line 130 allows you to view them in the console.

#### Lines 134 to 139 is where you take the raster data and get the mean value per census tract thanks to your studyoutline import you set up earlier. If you have shapefile of census blocks, this would then output the mean per census block and so on. In other words, the reduceRegions tells GEE to summarize the data via sub-regions, in this case census-tracts, and summarize it as the mean.

#### Lines 143 – 151 then takes these summarized census tracts for each sat. variable and outputs them as columns in a CSV.

#### After that is the EXPORT IMAGES section. This is optional, but you can save GEOTIFFS of each composite variable to your google drive to then import into a GIS, Python, etc. The projection is WGS 1984 and scale units are in meters (30 = 30 m and 1000 = 1000 m).

## Yearly Satellite Datasets

### This is the same set up as the previous script. Simply make the start\_year and end\_year variables equal each other and you got yearly composites!

### Make sure to import your study area asset into this script like you did in the \_AggSatData script.

## Monthly Satellite Datasets

### Conveniently, this is also the same set up as both prior scripts. However, there are two key differences.

#### The first difference is the parameter, month. This is the month you want to look into further. This name is part of the output file name scheme to keep track of the data.

#### The second is the start\_day\_of\_year and end\_day\_of\_year variables. You must make these the first and last day of year of that month, respectively. I have put a link to a day of year calendar to make it easier. Also, make sure to keep the year variables equal to each other.

### Make sure to import your study area asset into this script like you did in the \_AggSatData script.

## Land Surface Temperature (LST) Time Series Script

### The first step is to import your study area asset again.

### Next, you must define your parameters:

#### Start is equal to the start year, month, and day in the format seen.

#### End is when you want the time series to end. NOTE: GEE does not like to process large amounts of data, so for the time series you may have to do only two years at a time and then copy and paste the resulting CSVs into a spreadsheet join data rows from CSVs in a loop in Python or R.

#### Save is equal to true if you want to save the data.

#### Output\_dir is the google drive folder you are saving to.

#### Geography is the name of your study area for file naming.

#### Studyoutline is the name of your study area shapefile you imported in step 1.

#### The variable palettes is to allow you to call a lovely custom color ramps for the LST variable. You do not need to change anything here.

#### The print(palettes) allows you to see the codes for those color ramps much like color brewer in Python or R.

### On line 86, there is a map.setCenter function. You can put the lat-lon of your location, then its zoom level to focus on where you are working in the map portion of GEE.

### After that, you may hit run!

### Once it is done, a timeseries will appear on the right.

### Click on the little arrow at the top right of the time series graph

### It will open a new tab that will allow you to download the CSV of the data.

### Now we will go through line by line of the code.

#### Lines 65 – 81 is a mask function to filter for the highest quality data. This uses the metadata found in the landsat 8 collection, LANDSAT/LC08/C01/T1\_SR, to filter out images with high amount of clouds, water, etc. If you look at the collection and then click on bands, you can see what Bitmask you can use.

#### Lines 91 – 94 goes to the specific image collection, in this case from Landsat 8, filters the date thanks to the start and end variables you defined earlier, and then applies the cloud\_mask from the cloud\_mask function defined earlier.

#### Lines 101 to 114 calculate NDVI, which is the first step towards calculating LST.

#### Lines 124 – 128 scale the thermal band of Landsat 8 and uses that, along with NDVI, to get emissivity of the surface next.

#### Lines 140 – 157 calculate the emissivity. NOTE that all the print functions help to let you know in the console where GEE is at when it comes to the calculations.

#### Lines 170 – 180 calculates LST using what we calculated earlier. Since we are creating a timeseries, we must use the .copyProperties(image, image.propertyNames()) section so GEE does not erase the time index for each image. GEE erases the time index for some reason when using the image.expression function to do calculations.

#### Line 185 provides the number of LST values per pixel.

#### Line 188 creates a histogram of LST values in the study area at 30 m resolution.

#### Line 194 calculates the median LST value per pixel across all imagery to create a composite for visualization.

#### Line 202 creates the color ramp for LST.

#### Line 204 is an example of what else you can make lst\_pal equal to using the special color ramps we loaded in earlier.

#### Line 207 - 213 is the creation of the time series plot. It reduces the images to one by taking the mean to get average LST over time.

#### Line 219 adds the data to the map to see, clipped to the study area.

# R and Rstudio

## Social Demographic Data from \_TidyCensus.R (AZ\_Tempe\_UDII\_Fall2020\_TidyCensus.R)

### Make sure to go to the link on line 35 to retrieve your personal census API key if you do not have one already!

### In the “Setup libraries” section from lines 38 to 47, highlight the code and click run.

#### This code loads in the required functions to run the code.

#### Below this section is a TIPS section to help provide context around how to use tidycensus, calculate error, and explore available data.

### Set your environments (lines 69 – 106)

#### On line 72 you will copy and paste YOUR API code you go from the request link earlier. Lines 73-76 loads this API key into your R environment so you can access the data.

#### Line 79 allows you choose between tract and block level.

#### Line 82 is the year you want to look at. If you want to look at a 5-year ACS sample, put the end year (in this case 2018 was the end year).

#### Line 85 is what type of dataset you want, which can be seen by loading up the help documentation of the ?load\_variables function.

#### Line 89 is setting your working directory of input Data. This folder will be the same folder you will call later in the second R script.

#### Line 92 is where you read in the census tract or census block shape file boundaries for the study region.

#### Lines 95 – 104 is just in case your GEOIDs do not start with zeros. If they do start with zeros, you may skip these.

### Load in the social data (lines 110 – 119)

#### On line 110, this loads in the variables with the previously defined parameters.

#### Line 114 allows you to see the large dataframe with ALL social variables. Using this in combination with the data explorer linked in line 62 will allow you to either add new variables to this script or replace some.

#### Line 118 creates an output data frame (or tibble, etc.) and has the first column as the GEOIDs you called earlier. Line 119 renames that first column to GEOID. This is your common key to link to the satellite data and boundary shape file in a GIS for mapping tasks.

### Derive Total population (lines 127 – 155)

#### Line 127 assigns the totpop variable as the total population variable found in v18 dataframe or the data explorer.

#### Lines 129 – 133 filters the total population variable to Maricopa county. If you want to collect more total population type variables, you can add their call signs to the c() part of the “variables =” parameter. For example, variables = c(totpop, totpop\_thing).

#### Line 136 filters the total population from all of Maricopa county to the matching GEOIDs of Tempe.

#### Line 139 renames the TOTALPOP data frame

#### Lines 145 and 146 calculates the standard error and coefficient of variation for each census track or block for the total population variable. This gives you a sense of how reliable that data is.

#### Line 150 takes the TOTALPOP dataframe or tibble and joins it to the output data frame by the GEOID.

#### Congratulations, you just collected total pop. data through R! Yay!

### Derive total median income (lines 162 – 185)

#### All of this is the same as the total population process except for total median income. If you want to derive specific income variables, you can copy and paste and rework the section accordingly by changing the medinc <- “ “ variable.

#### If you do add more variables or change the current ones, please be sure to change the names, such as in lines 174 and 185.

### Derive percent ethnic minority (lines 194 – 237)

#### This one gets a touch more complicated. Lines 194 – 205 are the same as before, except on line 198 for variables you can see an example of calling multiple variables.

#### In order to estimate error for these variables via the ACS assessment, you need to make the 0s into 1s. That is what lines 211-213 are for. The brackets subset the dataframe and where a cell = 0 you assign it as 1.

#### Line 216 then calculates the percent minority estimate by taking the total minority estimate divided by total ethnic population estimate multiplied by 100.

#### Line 217 calculates the proportion.

#### Lines 218-219 calculate the standard error.

#### Lines 225-232 is a for loop function to estimate the error for the ACS assessments. It says that for each cell for each census tract or block group, if the proportion of x in Qminority\_p is = 1, then calculate standard error this way. If it is not equal to 1, then calculate it this other way instead. If new information comes out that the way to calculate standard error for these variables changes, this is where you would apply those changes. If a variable requires this special way of estimating error, you can copy and paste this loop and change the parameters to fit the new variable.

#### Lines 234-235 calculates the CV and then outputs the results to the output dataframe.

### Derive percent below poverty line (lines 246 – 284)

#### This block of code operates in the same manner as the percent ethnic minority block of code above.

### Derive percent of adults (>25) without a high school diploma (lines 288 – 356)

#### This code block setup is like the previous two.

#### One notable difference is line 294 in collecting multiple variables to be able to estimate education level.

#### On line 306, it uses the apply function to add all the NOHSDip (no high school diploma) variables to together with the sum option at the end of line 306. Thus, if you pull multiple variable IDs like in lines 294 and 295, you can find the sum, mean, etc. of those variables by using the code setup on line 306.

#### What line 306 is doing is: apply(QNoHSDip[,c(paste0(QNoHSDip\_IDs,"E"))],1,sum)

##### The first part, QNoHSDip[ , c(paste0(\*\*\*,”E”))], is saying for R to look at the QNoHSDip rows with E in the title

##### The second part says that once you are looking at those variables, stay in the rows with the 1 parameter.

##### The third part says what to do with the data which, in this case, is sum because we want to find the total.

#### In code block lines 315 – 318 you are looking for NA values to make 0 for the error estimates.

### Derive percent over 65 years old (lines 361 – 427)

#### Again, this is like the previous code block but for estimating seniors.

### Derive percent over 65 years old and living alone (lines 432 – 494)

#### Again, this is like the previous code block but for estimating seniors living alone.

### Derive percent living alone (lines 505 – 566)

#### Finally, this is the same as the above code blocks.

### Quality Control and Outputs (lines 570 – 588)

#### Lines 571 – 574 looks for NAs and removes them. Specifically, line 571 creates a dataframe that is only NAs. Line 572 Looks to see if there are any unique values, which should be none. Line 573 removes NAs by GEOID. Line 574 then creates a dataframe where the output.df does not equal (the ! symbol) anything with an NA.

#### Lines 578 to 582 separates the estimates, E, and coefficient of variation, CV, to different data frames so you can have CSVs, or spreadsheets, of separate factors.

#### Lines 586 – 588 creates the CSVs and outputs them into an output folder or a place of your choice. NOTE, make sure to change the output directory where it says: paste("/AZ\_Fall2020/R\_Scripts/Output\_R/TMP\_UHEAT\_CENSUS\_VARIABLES", year, type, "CT\_ALLVariables\_ALLMetrics", paste(Sys.Date(),".csv",sep=""),sep = "\_").

#### A tip to see if something works before assigning it or running the full function is to highlight key parts, such as the paste(\*\*\* above to see if it outputs what you want in the console below.

## Principal Component Analysis from \_PrincipalComponentAnalysis.R (AZ\_Tempe\_UDII\_Fall2020\_PrincipalComponentAnalysis\_UHEAT.R)

### Set up your libraries and input variable names (lines 37 – 70)

#### Lines 39 to 53 install the libraries that you need functions from to run the code.

#### Line 57 is to set the path where your input data are, being the CSV from the TidyCensus file and CSV file from the GEE output.

#### Lines 62 and 65 are the names of the TidyCensus and GEE outputs, make sure the names match those files!

#### Line 68 is the output name for the PCA analysis that you will join to an attribute table in GIS with GEOID as the common key.

### Calculating the Heat Vulnerability Score (lines 73 – 180)

#### Line 76 is loading in the data given the corrected working directory and SEH\_filename.

#### Line 81 takes all the data EXCEPT the GEOID column (hence the [,2:ncol(SHE\_data)] to prepare z-score standardization.

#### Lines 84 and 85 rename the column variables and must be changed here if you changed or added more. Add them in the order the columns are in.

#### Line 87 scales the data you selected previously to z-scores to normalize the data.

#### Line 92 does the principal component analysis without any modifications.

#### There are then several criteria used to determine how many components you should use in the analysis. Line 96 is the first one.

#### Line 100 and 101 will show you how much variance each component explains.

#### Line 105 creates a scree plot to investigate how many components you should use.

#### Lines 109 – 127 create random data and plots the difference between the random dataset and the actual dataset regarding how many components to use and how much variance each component explains.

#### After looking at how many components explain about 80% of the variation, line 131 assigns how many components to use (nfactors parameter).

#### Line 137 displays in the console the correlations between variables in the components (RCx).

#### Lines 143-145 repeats the process but with correction factors such as a varimax rotation as per the literature.

#### Line 148 subsets the loadings by selecting the loadings data frame and the first two columns.

#### Line 152 is renaming the principal components (PCs), if you want more than two, you would add the names.

#### Lines 157 to 163 is a function to rotate the labels by 45 degrees in the coming plots or, pheatmap.

#### Lines 166 to 172 creates the correlation visualization and saves it as a .eps file (for Macs) to edit. Dev.off() is a function that clears the plots so Rstudio can continue to visualize plots effectively.

##### If you do not have a Mac, run just the postscript and pheatmap parts, then when it appears on the bottom right of the rstudio window under the plots tab, click on export and export it to an appropriate file and name it accordingly. It will export to your working directory by default.

#### Lines 175 – 179 round the scores and outputs them as a matrix to join to a dataframe later.

### Heat Exposure Score (lines 183 – 278)

#### Line 185 reads in the data.

#### Line 188 filters the data to needed variables. In my case, the GEOID in GEE was TempeTra\_3, so I renamed it in line 191 along with the other variables.

#### The rest of the code for this section is in the same format as the previous score derivation. Just be sure to check the names and the number of principal components to be used.

### Heat Priority Score (lines 281 – 379)

#### Line 285 joins both the vulnerability and environmental datasets for a combined analysis.

#### Then it continues the same as the previous two sections.

### Saving the Data (lines 383 – 416)

#### Line 384 creates a difference from the mean column for all the variables. This was not used for the difference of the mean maps that were presented. This can be skipped.

#### Lines 391 – 397 creates an output dataframe that joins the GEOID, input variables, DMu, and all the score variables.

#### Lines 401 – 413 renames all variables.

#### Line 416 writes the CSV table to the current working directory. NOTE: this is the table you will load into GIS and join to either the census tract or census block shapefile with GEOID as the common key. Great job for finishing!