//Code for EE Project: Congestion and Crash Severity along Philadelphia Vision Zero High Injury Network Roads

//By Ian Schwarzenberg

//CPLN 670, Fall 2018

//STEP 1: LOAD IN AND CREATE DATA

//Step 1a: Load in shapefiles:

var TTIs = ee.FeatureCollection('ft:1ujHCjJFiyKIQamrVrxiUr2jyi7mlLsakPyCHGDjB'); // Loads in INRIX TTI lines shapefile containing congestion data (Sources: INRIX and Delaware Valley Regional

//Planning Commission)

var PTIs = ee.FeatureCollection('ft:13mAvrsxdLr4bSTSbbMwhAfTAg2a2DBMo0bg\_UyYP'); // Loads in INRIX PTI lines shapefile containing congestion data (Sources: INRIX and Delaware Valley Regional

//Planning Commission)

var City\_HIN = ee.FeatureCollection('ft:1edGX7-c7OBUVTyV7vEzcKqKxgTcXaQPjb1wWAT4e'); // Loads in city's HIN lines shapefile (source: Philadelphia Office of Transportation and Infrastructure

//Systems)

var Street\_Centerlines = ee.FeatureCollection('ft:1l-o1O1Lf89XoSA0TFUGi\_SB71PrNg1lOaSS0MQFM') // Loads in Philadelphia street centerlines which contain street width information (source:

//Philadelphia Streets Department)

var Census\_Tracts = ee.FeatureCollection('ft:1gZonlFMK3uPYcuCoN2hWBwn-FuDnF5YoEMBOp3Uv'); // Loads in 2010 city census tract polygons shapefile (source: Philadelphia Department of Planning and

//Development)

var CT\_Median\_HH\_Incs = ee.FeatureCollection('ft:1lvCiogCQVZlve8y99S9N13S-wzKyQARIZoCvAyqY'); // Loads in table of all city census tract's median household incomes (source: US Census Bureau

//American Community Survey 2012-16)

var CT\_Median\_HH\_Incs\_Metadata = ee.FeatureCollection('ft:12W3bePDjJLjY1kM49hHiBnvXJ7xtQuoNafpSWoQO'); // Loads in the metadata for the table of all city census tract's median household incomes

//(source: US Census Bureau American Community Survey 2012-16)

var Rail\_Stations = ee.FeatureCollection('ft:1wWXf-LjX\_CY9UQhXpM99zMRt0e4f6ItX1S9yG-Kg'); // Loads in all Philadelphia area rail stations (Sources: Southeastern Pennsylvania Transportation

//Authority and Delaware Valley Regional Planning Commission)

var Commercial\_Corridors = ee.FeatureCollection('ft:15e5-R5f6ZiE4By\_IzfeCyAQTG6VmL6gPyj7R71LQ'); // Loads in all Philadelphia commercial corridors (Source: Philadelphia Department of Planning and

//Development)

var Phila\_Boundary = ee.FeatureCollection('ft:1A2rSnITNWDIUUL0A3Z5C8JGZDCRI0QVsnkocUmEl'); // Loads in Phila. boundary polygon to clip INRIX lines and for map display (Source: Philadelphia

//Department of Planning and Development)

//Step 1b: Create Philadelphia 2013-17 crashes shapefile from PennDOT's CSVs:

var Crashes13CSV = ee.FeatureCollection('ft:1dpTKRh1MOzgiZCXz7\_Znsu1tUg4GHGUJug\_Q-57m'); // Loads in Philadelphia 2013 crashes CSV (Source: Pennsylvania Department of Transportation)

var Crashes14CSV = ee.FeatureCollection('ft:1M4\_85ksqykxxhMiG4gs3lBsA3W49U5oPyqjnRGoV'); // Loads in Philadelphia 2014 crashes CSV (Source: Pennsylvania Department of Transportation)

var Crashes15CSV = ee.FeatureCollection('ft:1TC0DJv52j2F5Y-g27sn0yPAYCSiRKt\_wi\_4D4QXG'); // Loads in Philadelphia 2015 crashes CSV (Source: Pennsylvania Department of Transportation)

var Crashes16CSV = ee.FeatureCollection('ft:1Ea4aVCOpd4tOT4SYi-2jNVYGvRkgLqWK6LGhItRX'); // Loads in Philadelphia 2016 crashes CSV (Source: Pennsylvania Department of Transportation)

var Crashes17CSV = ee.FeatureCollection('ft:14ALBJyASt4tjWEfFBByTHfODwKEXUrbGvsouomms'); // Loads in Philadelphia 2017 crashes CSV (Source: Pennsylvania Department of Transportation)

//Step 1c: Merge the crash CSVs into one big CSV and make a feature class/shapefile out of it:

var CrashesCSV = Crashes13CSV.merge(Crashes14CSV);

var CrashesCSV = CrashesCSV.merge(Crashes15CSV);

var CrashesCSV = CrashesCSV.merge(Crashes16CSV);

var CrashesCSV = CrashesCSV.merge(Crashes17CSV);

//Export.table.toDrive({

// collection: CrashesCSV,

// description:'Crashes\_All',

// fileFormat: 'CSV'

//}); //Creates a shapefile out of the final crashes CSV, would not export to SHP due to field names being longer than 10 characters, so I exported it as a CSV to my drive, found out how to do

//from https://developers.google.com/earth-engine/exporting. Made this command in comments so that it does not run again

//Created new fusion table in Google Drive from this CSV

//Made Google read the table's latitude and longitude columns as a "two-column" location in Google Sheets so Earth Engine could read this as a point shapefile

//(found out how to do from https://support.google.com/fusiontables/answer/175922?hl=en)

var Crashes\_All = ee.FeatureCollection('ft:12EFFwZsDvX2W6W7Il9k5vJFccMD0tVTxwDgj2D60'); //Now Google Earth reads the 2013-17 crash points in as a shapefile

//Step 1d: Create transparent Philadelphia boundary polygon to use on all maps:

var empty = ee.Image().byte();

var Phila\_Boundary\_Transparent = empty.paint({

featureCollection: Phila\_Boundary,

color: 1,

width: 1.25

});

//STEP 2: CREATE FIRST MAP SHOWING NEW HIN DATASET WITH INRIX CONGESTION DATA JOINED TO IT

//Step 2a: Create the map and add base layers (STEPS 4A, 7A, 8A, 9A, 10A, 11A AND 12A ALL FOLLOW THIS TEMPLATE):

var Map1 = ui.Map({

lon: -75.163617,

lat: 39.952370,

zoom: 11

}); //INSTEAD OF USING 1 DEFAULT MAP TO PUT ALL MY DATA ON, I AM SPLITTING EVERYTHING INTO 7 SEPARATE MAPS THAT THE USER CAN CHOOSE TO SEE AND SORT THROUGH. THIS CREATES MAP #1. Found out how to do from https://developers.google.com/earth-engine/ui\_widgets

Map1.addLayer(Phila\_Boundary\_Transparent, {palette: '000000'}, 'Philadelphia Boundary'); //This and next line set the base layers for the first map. NOTE HOW INSTEAD OF USING "Map.addLayer" LIKE HOW IT WAS TAUGHT IN CLASS, I DO IT AS "Map1.addLayer" FOR EXAMPLE SINCE I AM NOT USING EE'S DEFAULT MAP AND INSTEAD AM CREATING MY OWN

Map1.addLayer(City\_HIN, {color:'96979B'}, 'Original HIN');

//Step 2b: Clip the INRIX TTI and PTI line datasets using filter so that they just are for Philadelphia

var Phila\_Filter = ee.Filter.bounds(Phila\_Boundary);

var Phila\_TTI = TTIs.filter(Phila\_Filter);

var Phila\_PTI = PTIs.filter(Phila\_Filter);

//Step 2c: Create City HIN TTI variable by joining the INRIX TTI data to the City HIN based on street name:

var TheFILTER = ee.Filter.equals('stname', null, 'ROADNAME', null ); //Joins INRIX data to City HIN by their common street names

var TheJOIN = ee.Join.inner('primary','secondary'); //Establishes the join

var HIN\_with\_TTIs = TheJOIN.apply(City\_HIN, Phila\_TTI, TheFILTER); //Applies the join, city HIN is primary dataset since I want each city HIN line to have corresponding TTI data

//DVRPC defined "rush hours": 6-10 AM (morning), 3-7 PM (evening), told to Ian Schwarzenberg by DVRPC Senior Planner Tom Edinger on 9/21/18.

//DVRPC defined "peak periods": 7-8 AM (morning), 5-6 PM (evening), told to Ian Schwarzenberg by DVRPC Senior Planner Tom Edinger on 9/21/18.

var HIN\_with\_TTIs = HIN\_with\_TTIs.map(function(element){

var PRIMARY = ee.Feature(element.get('primary'));

var SECONDARY = ee.Feature(element.get('secondary'));

var stname = PRIMARY.get('stname');

var T600AM = SECONDARY.get('T600AM');

var T700AM = SECONDARY.get('T700AM');

var T800AM = SECONDARY.get('T800AM');

var T900AM = SECONDARY.get('T900AM');

var T1000AM = SECONDARY.get('T1000AM');

var T300PM = SECONDARY.get('T300PM');

var T400PM = SECONDARY.get('T400PM');

var T500PM = SECONDARY.get('T500PM');

var T600PM = SECONDARY.get('T600PM');

var T700PM = SECONDARY.get('T700PM');

var geom = ee.Feature(element.get('primary')).geometry();

return ee.Feature(geom, {'stname':stname, 'T600AM':T600AM,

'T700AM':T700AM, 'T800AM':T800AM, 'T900AM':T900AM, 'T1000AM':T1000AM, 'T300PM':T300PM,

'T400PM':T400PM, 'T500PM':T500PM, 'T600PM':T600PM, 'T700PM':T700PM

});

}); //Modifies the variable resulting from the join so that the INRIX lines corresponding with their respective city HIN lines have the TTI data attached to them. Found out how to do from

//https://piazza.com/class/jlcgdaowio41w3?cid=124

//print("HIN\_with\_TTIs", HIN\_with\_TTIs); //Briefly ran INDEPENDENTLY FROM THE BUTTON to show how the 2319 INRIX line segments that line up with the city HIN all have TTI data attached to them, showing rest of step 2b above worked

//Export.table.toDrive({

// collection: HIN\_with\_TTIs,

// description:'HIN\_with\_TTIs',

// fileFormat: 'SHP'

//}); //Creates a shapefile out of the HIN TTI dataset AT THIS POINT to save my progress

Map1.addLayer(HIN\_with\_TTIs, {color:'090FEA'}, 'HIN with TTI Data Joined'); //ADDS HIN WITH TTI DATA JOINED TO IT TO MAP TO SHOW THAT JOIN WORKED. NOTE HOW IT SAYS "Map1.addLayer" AS OPPOSED TO "Map.addLayer" LIKE WHAT WAS TAUGHT IN CLASS

var Map1Button1 = ui.Button({

label: 'Click Me to See: Structure of City HIN Dataset with Joined TTI Data', //Names button

onClick: function() {

print(HIN\_with\_TTIs); //Shows the line of code that will be executed on the back end when user clicks button

}

}); //Creates button for the user to press in the console which will show them the calculation. Found out how to do from https://developers.google.com/earth-engine/ui\_widgets

//Step 2d: Create City HIN PTI variable by joining the INRIX PTI data to the City HIN based on street name:

var TheFILTER = ee.Filter.equals('stname', null, 'ROADNAME', null );

var TheJOIN = ee.Join.inner('primary','secondary');

var HIN\_with\_PTIs = TheJOIN.apply(City\_HIN, Phila\_PTI, TheFILTER);

var HIN\_with\_PTIs = HIN\_with\_PTIs.map(function(element){

var PRIMARY = ee.Feature(element.get('primary'));

var SECONDARY = ee.Feature(element.get('secondary'));

var stname = PRIMARY.get('stname');

var T600AM = SECONDARY.get('T600AM');

var T700AM = SECONDARY.get('T700AM');

var T800AM = SECONDARY.get('T800AM');

var T900AM = SECONDARY.get('T900AM');

var T1000AM = SECONDARY.get('T1000AM');

var T300PM = SECONDARY.get('T300PM');

var T400PM = SECONDARY.get('T400PM');

var T500PM = SECONDARY.get('T500PM');

var T600PM = SECONDARY.get('T600PM');

var T700PM = SECONDARY.get('T700PM');

var geom = ee.Feature(element.get('primary')).geometry();

return ee.Feature(geom, {'stname':stname, 'T600AM':T600AM,

'T700AM':T700AM, 'T800AM':T800AM, 'T900AM':T900AM, 'T1000AM':T1000AM, 'T300PM':T300PM,

'T400PM':T400PM, 'T500PM':T500PM, 'T600PM':T600PM, 'T700PM':T700PM

});

});

//print("HIN\_with\_PTIs", HIN\_with\_PTIs); //Briefly ran OUTSIDE OF THE BUTTON to make sure this join worked properly, it did

//Export.table.toDrive({

// collection: HIN\_with\_PTIs,

// description:'HIN\_with\_PTIs',

// fileFormat: 'SHP'

//});

Map1.addLayer(HIN\_with\_PTIs, {color:'EA1709'}, 'HIN with PTI Data Joined');

var Map1Button2 = ui.Button({

label: 'Click Me to See: Structure of City HIN Dataset with Joined PTI Data',

onClick: function() {

print(HIN\_with\_PTIs);

}

});

//STEP 3: CREATE AVERAGE TTI AND PTI FOR BUSIEST HOUR COLUMN CREATION METHODS FOR STEPS 4 AND 5

//Found out how to construct column creator functions for this project from Found out how to do this from https://piazza.com/class/jlcgdaowio41w3?cid=40

//Step 3a: Create morning rush hour column creation method:

var T610AM\_AVERAGING = function(feature) {

var T600AM = ee.Number(feature.get('T600AM')); //Function first selects the T600AM, T700AM, T800AM, T900AM and T1000AM columns

var T700AM = ee.Number(feature.get('T700AM'));

var T800AM = ee.Number(feature.get('T800AM'));

var T900AM = ee.Number(feature.get('T900AM'));

var T1000AM = ee.Number(feature.get('T1000AM'));

var mean = ee.Reducer.mean(); //Function then creates mean reducer

var T610AM = ee.Number(ee.List([T600AM, T700AM, T800AM, T900AM, T1000AM]).reduce(mean)) //Function then finds mean of T600AM, T700AM, T800AM, T900AM and T1000AM columns using mean reducer

return feature.set({ 'T610AM' : T610AM}) //Function then creates new column called TTI610AM and fills it in with the means of T600AM, T700AM, T800AM, T900AM and T1000AM columns

};

//Step 3b: Create evening rush hour column creation method:

var T37PM\_AVERAGING = function(feature) {

var T300PM = ee.Number(feature.get('T300PM'));

var T400PM = ee.Number(feature.get('T400PM'));

var T500PM = ee.Number(feature.get('T500PM'));

var T600PM = ee.Number(feature.get('T600PM'));

var T700PM = ee.Number(feature.get('T700PM'));

var mean = ee.Reducer.mean();

var T37PM = ee.Number(ee.List([T300PM, T400PM, T500PM, T600PM, T700PM]).reduce(mean))

return feature.set({ 'T37PM' : T37PM})

};

//Step 3c: Create morning peak hour column creation method:

var T78AM\_AVERAGING = function(feature) {

var T700AM = ee.Number(feature.get('T700AM'));

var T800AM = ee.Number(feature.get('T800AM'));

var mean = ee.Reducer.mean();

var T78AM = ee.Number(ee.List([T700AM, T800AM]).reduce(mean))

return feature.set({ 'T78AM' : T78AM})

};

//Step 3d: Create evening peak hour column creation method:

var T56PM\_AVERAGING = function(feature) {

var T500PM = ee.Number(feature.get('T500PM'));

var T600PM = ee.Number(feature.get('T600PM'));

var mean = ee.Reducer.mean();

var T56PM = ee.Number(ee.List([T500PM, T600PM]).reduce(mean))

return feature.set({ 'T56PM' : T56PM})

};

//STEP 4: CREATE SECOND MAP SHOWING MOST CONSISTENTLY CONGESTED HIN ROADS FOR TTI AND PTI DURING BUSIEST HOURS

//Step 4a: Create the map and add base layers:

var Map2 = ui.Map({

lon: -75.163617,

lat: 39.952370,

zoom: 11

});

Map2.addLayer(Phila\_Boundary\_Transparent, {palette: '000000'}, 'Philadelphia Boundary');

Map2.addLayer(City\_HIN, {color:'96979B'}, 'Original HIN');

Map2.addLayer(HIN\_with\_TTIs, {color:'090FEA'}, 'HIN with TTI Data Joined');

Map2.addLayer(HIN\_with\_PTIs, {color:'EA1709'}, 'HIN with PTI Data Joined');

//Step 4a: Create HIN TTI morning rush hour field:

var HIN\_with\_TTIs\_WithMorningRush = HIN\_with\_TTIs.map(T610AM\_AVERAGING); //Creates a new field in HIN\_with\_TTIs called TTI610AM which averages the TTIs for 6-10 AM for each HIN road. Then it

//applies the function to ALL rows of HIN\_with\_TTIs.

//print(HIN\_with\_TTIs\_WithMorningRush); //Briefly ran to see if lines above successfully create a column showing average TTI from 6 to 10 AM, it did

//Step 4b: Create HIN TTI evening rush hour field:

var HIN\_with\_TTIs\_WithEveningRush = HIN\_with\_TTIs\_WithMorningRush.map(T37PM\_AVERAGING); //This line then applies the function to ALL rows of HIN\_with\_TTIs\_WithMorningRush because

//HIN\_with\_TTIs\_WithMorningRush has the T610AM column, I want my final HIN\_with\_TTIs\_Final dataset to have T610AM, T37PM, T78AM and T56PM columns. NOTE HOW THIS EXACT FUNCTION CAN BE USED FOR

//PTI EVENING RUSH HOUR IN STEP 3F, SO I DON'T HAVE TO TYPE THE FUNCTION ABOVE AGAIN FOR THAT

//Step 4c: Create HIN TTI morning peak period field:

var HIN\_with\_TTIs\_WithMorningPeak = HIN\_with\_TTIs\_WithEveningRush.map(T78AM\_AVERAGING); //This line then applies the function to ALL rows of HIN\_with\_TTIs\_WithEveningRush because

//HIN\_with\_TTIs\_WithEveningRush has the T37PM column, I want my final HIN\_with\_TTIs\_Final dataset to have T610AM, T37PM, T78AM and T56PM columns. NOTE HOW THIS EXACT FUNCTION CAN BE USED FOR PTI

//MORNING PEAK HOUR IN STEP 3G, SO I DON'T HAVE TO TYPE THE FUNCTION ABOVE AGAIN FOR THAT

//Step 4d: Create HIN TTI evening peak period field:

var HIN\_with\_TTIs\_Final = HIN\_with\_TTIs\_WithMorningPeak.map(T56PM\_AVERAGING); //CREATES FINAL HIN\_with\_TTIs\_Final dataset with T610AM, T37PM, T78AM and T56PM columns. NOTE HOW THIS EXACT

//FUNCTION CAN BE USED FOR PTI EVENING PEAK HOUR IN STEP 3H, SO I DON'T HAVE TO TYPE THE FUNCTION ABOVE AGAIN FOR THAT

//print(HIN\_with\_TTIs\_Final); //Briefly ran OUTSIDE OF BUTTON to make sure HIN\_with\_TTIs\_Final dataset indeed has T610AM, T37PM, T78AM and T56PM columns, it does

//Export.table.toDrive({

// collection: HIN\_with\_TTIs\_Final,

// description:'HIN\_with\_TTIs\_Final',

// fileFormat: 'SHP'

//}); //Creates a shapefile out of the final HIN TTI dataset to save my progress

Map2.addLayer(HIN\_with\_TTIs\_Final, {color:'060A86'}, 'HIN with TTI Busiest Hour Columns'); //ADDS HIN\_with\_TTIs\_Final TO MAP TO SHOW THAT FEATURE COLLECTION COULD STILL BE MAPPED AFTER NEW COLUMNS WERE ADDED TO IT

var Map2Button1 = ui.Button({

label: 'Click Me to See: Structure of City HIN Dataset with TTI Busiest Hour Columns',

onClick: function() {

print(HIN\_with\_TTIs\_Final);

}

});

//Step 4e: Create HIN PTI morning rush hour field:

var HIN\_with\_PTIs\_WithMorningRush = HIN\_with\_PTIs.map(T610AM\_AVERAGING);

//Step 4f: Create HIN PTI evening rush hour field:

var HIN\_with\_PTIs\_WithEveningRush = HIN\_with\_PTIs\_WithMorningRush.map(T37PM\_AVERAGING);

//Step 4g: Create HIN PTI morning peak period field:

var HIN\_with\_PTIs\_WithMorningPeak = HIN\_with\_PTIs\_WithEveningRush.map(T78AM\_AVERAGING);

//Step 4h: Create HIN PTI evening peak period field:

var HIN\_with\_PTIs\_Final = HIN\_with\_PTIs\_WithMorningPeak.map(T56PM\_AVERAGING); //CREATES FINAL HIN\_with\_PTIs\_Final dataset with T610AM, T37PM, T78AM and T56PM columns.

//print(HIN\_with\_PTIs\_Final); //Briefly ran OUTSIDE OF BUTTON to make sure HIN\_with\_PTIs\_Final dataset indeed has T610AM, T37PM, T78AM and T56PM columns, it does

//Export.table.toDrive({

// collection: HIN\_with\_PTIs\_Final,

// description:'HIN\_with\_PTIs\_Final',

// fileFormat: 'SHP'

//}); //Creates a shapefile out of the final HIN PTI dataset to save my progress

Map2.addLayer(HIN\_with\_PTIs\_Final, {color:'810F08'}, 'HIN with PTI Busiest Hour Columns'); //ADDS HIN\_with\_PTIs\_Final TO MAP TO SHOW THAT FEATURE COLLECTION COULD STILL BE MAPPED AFTER NEW COLUMNS WERE ADDED TO IT

var Map2Button2 = ui.Button({

label: 'Click Me to See: Structure of City HIN Dataset with PTI Busiest Hour Columns',

onClick: function() {

print(HIN\_with\_PTIs\_Final);

}

});

//STEP 5: FIND WHERE MOST CONSISTENTLY CONGESTED ROADS ARE (ANY HIN ROAD MARKED AS CONGESTED THROUGH BOTH TTI AND PTI):

//Step 5a: Create intersection method TO ALSO USE IN STEPS 9 THROUGH 12:

var intersection\_Filter = ee.Filter.withinDistance( 0, '.geo', null, '.geo' ); //Establishes filter where primary features within 0 meters of secondary features (intersecting features)

//get selected

var intersection\_Join = ee.Join.saveAll( 'JOINED', null, true,'DISTANCE' ); //Uses saveAll join to retain elements of primamry feature that intersect with secondary feature

//Step 5b: Find and map out where most consistently congested roads are for TTI:

var HIN\_TTI\_Congested\_MorningRush = HIN\_with\_TTIs\_Final.filterMetadata( 'T610AM', 'greater\_than', 1.5 ); //Filters all HIN roads by their average TTI for morning rush hour. TTI>1.5 means

//congested HIN road

var HIN\_TTI\_Congested\_MorningPeak = HIN\_with\_TTIs\_Final.filterMetadata( 'T78AM', 'greater\_than', 1.5 ); //Filters all HIN roads by their average TTI for morning peak hour

var HIN\_TTI\_Congested\_Morning = intersection\_Join.apply(HIN\_TTI\_Congested\_MorningRush, HIN\_TTI\_Congested\_MorningPeak, intersection\_Filter); //"Intersects" using my intersection method created in

//step 4a

//print(HIN\_TTI\_Congested\_MorningRush); //Briefly ran to make sure my intersection method worked so that feature collection is created instead of geometry collection so that way I can use my

//intersection method in step 4a for rest of project, it worked

var HIN\_TTI\_Congested\_EveningRush = HIN\_with\_TTIs\_Final.filterMetadata( 'T37PM', 'greater\_than', 1.5 ); //Filters all HIN roads by their average TTI for evening rush hour

var HIN\_TTI\_Congested\_EveningPeak = HIN\_with\_TTIs\_Final.filterMetadata( 'T56PM', 'greater\_than', 1.5 ); //Filters all HIN roads by their average TTI for evening peak hour

var HIN\_TTI\_Congested\_Evening = intersection\_Join.apply(HIN\_TTI\_Congested\_EveningRush, HIN\_TTI\_Congested\_EveningPeak, intersection\_Filter);

var HIN\_TTI\_Congested\_MorningandEvening = intersection\_Join.apply(HIN\_TTI\_Congested\_Morning, HIN\_TTI\_Congested\_Evening, intersection\_Filter);

//Step 5c: Find and map out where most consistently congested roads are for PTI (follows same template as step 4a, but for PTI):

var HIN\_PTI\_Congested\_MorningRush = HIN\_with\_PTIs\_Final.filterMetadata( 'T610AM', 'greater\_than', 3 );

var HIN\_PTI\_Congested\_MorningPeak = HIN\_with\_PTIs\_Final.filterMetadata( 'T78AM', 'greater\_than', 3 );

var HIN\_PTI\_Congested\_Morning = intersection\_Join.apply(HIN\_PTI\_Congested\_MorningRush, HIN\_PTI\_Congested\_MorningPeak, intersection\_Filter);

var HIN\_PTI\_Congested\_EveningRush = HIN\_with\_PTIs\_Final.filterMetadata( 'T37PM', 'greater\_than', 3 );

var HIN\_PTI\_Congested\_EveningPeak = HIN\_with\_PTIs\_Final.filterMetadata( 'T56PM', 'greater\_than', 3 );

var HIN\_PTI\_Congested\_Evening = intersection\_Join.apply(HIN\_PTI\_Congested\_EveningRush, HIN\_PTI\_Congested\_EveningPeak, intersection\_Filter);

var HIN\_PTI\_Congested\_MorningandEvening = intersection\_Join.apply(HIN\_PTI\_Congested\_Morning, HIN\_PTI\_Congested\_Evening, intersection\_Filter);

//Step 5d: Create final most consistently congested HIN roads variable in terms of both TTI and PTI:

var HIN\_Congested = intersection\_Join.apply(HIN\_TTI\_Congested\_MorningandEvening, HIN\_PTI\_Congested\_MorningandEvening, intersection\_Filter);

//print(HIN\_Congested); //Briefly ran to make sure whole process worked, it did

//Export.table.toDrive({

// collection: HIN\_Congested,

// description:'HIN\_Congested\_CSV',

// fileFormat: 'CSV'

//}); //Creates a CSV out of the HIN\_Congested dataset to save my progress, would not export to shapefile because EE said join column was too wide

//STEP 6: CREATE SPATIAL JOIN AND COLUMN CREATION METHODS FOR STEPS 7 AND 8

//Step 6a: Create spatial join method (found out how to do this spatial join from https://developers.google.com/earth-engine/joins\_spatial):

var distFilter = ee.Filter.withinDistance({

distance: 15.24,

leftField: '.geo',

rightField: '.geo',

maxError: 0

}); // Define a spatial filter, with distance 15.24 meters (15.24 meters = 50 feet, width of a typical street in either direction of a street centerline to capture all crashes happening on those

//streets).

var distSaveAll = ee.Join.saveAll({

matchesKey: 'points',

measureKey: 'distance'

}); // Define a saveAll join, since each street gets all points joined to it saved, making up the spatial join

//Step 6b: Create road length column creation method:

var Length\_Finder = function(feature) {

var length = ee.Number(feature.length().divide(1609.344)); //Calculates each feature's length. Divides the meter length value by 1609.344 to get miles equivalent Found out how to calculate

//length from https://www.earthdatascience.org/tutorials/basic-polygon-operations-google-earth-engine/

return feature.set({ 'length' : length}) //Creates the actual length column but does not apply length calculations to entire dataset until next line

};

//Step 6c: Create raw KSI crash count column creation method:

var KSICrashCount\_Finder = function(feature) {

var points = ee.List(feature.get('points')); //Part of KSI crash count column creation function which extracts the list called "points" containing the points joined to each street line segment

var KSICrshCnt = ee.Number(points.length()); //Part of KSI crash count column creation function which counts the number of items in the "points" list using "length()" ("length" means NUMBER OF

//ITEMS IN THE LIST, AS OPPOSED TO "DISTANCE" LENGTH), detailing each street segment's KSI crash count

return feature.set({ 'KSICrshCnt' : KSICrshCnt}) //Creates the actual KSI crash column

};

//Step 6d: Create KSI crash rate per mile column creation method:

var KSICrashRate\_Finder = function(feature) {

var points = ee.Number(feature.get('KSICrshCnt')); //Part of KSI crash rate per mile column creation function which extracts the KSI crash count column

var lengthmi = ee.Number(feature.get('length')); //Part of KSI crash rate per mile column creation function which extracts the road length in miles column

var KSICrshRte = ee.Number(points.divide(lengthmi)); //Part of KSI crash rate per mile column creation function which divides each road's number of KSI crashes by each road's length in miles,

//to get each road's KSI crash rate per mile

return feature.set({ 'KSICrshRte' : KSICrshRte}) //Creates the actual KSI crash rate per mile column

};

//STEP 7: CREATE THIRD MAP COMPARING KSI CRASHES TO CONGESTED HIN AND FIND EACH CONGESTED HIN SEGMENT'S KSI CRASH RATE PER MILE

//Step 7a: Create the map and add base layers:

var Map3 = ui.Map({

lon: -75.163617,

lat: 39.952370,

zoom: 11

});

Map3.addLayer(Phila\_Boundary\_Transparent, {palette: '000000'}, 'Philadelphia Boundary');

Map3.addLayer(City\_HIN, {color:'96979B'}, 'Entire HIN');

Map3.addLayer(HIN\_Congested, {color:'BB7025'}, 'Congested HIN Roads');

//Step 7b: Filter out all crash points that are KSI ("Killed Severe Injury") to prepare for spatial joining of KSI crash points to congested HIN street lines:

//"MAX\_SEVERITY\_LEVEL" column indicates severity of crash, 1=Killed and 2=Serious injury for that column

var Killed\_Crashes = Crashes\_All.filterMetadata('MAX\_SEVERITY\_LEVEL', 'equals', 1);

var SevereInjury\_Crashes = Crashes\_All.filterMetadata('MAX\_SEVERITY\_LEVEL', 'equals', 2);

var KSI\_Crashes = Killed\_Crashes.merge(SevereInjury\_Crashes); //Merges killed and severe injury crashes for CONGESTED HIN to get FINAL KSI CRASHES ON CONGESTED HIN

Map3.addLayer(KSI\_Crashes, {color:'C7C7C7'}, 'KSI Crash Locations', false); //ADDS KSI CRASH POINTS TO MAP, false makes it so the layer is not turned on automatically

//Step 7c: Add new column to congested HIN calculating each street segment's length:

var HIN\_Congested = HIN\_Congested.map(Length\_Finder); //CREATES COLUMN IN HIN\_Congested SHOWING EACH ROAD'S LENGTH IN MILES

//print(HIN\_Congested);// Ran to check to make sure length calculations worked, they did

//Step 7d: Add new field to congested HIN counting each segment's raw number of KSI crashes:

var HINCongested\_KSICrashes\_SpatialJoin = distSaveAll.apply(HIN\_Congested, KSI\_Crashes, distFilter); // Apply the spatial join for congested HIN segments

//print(HINCongested\_KSICrashes\_SpatialJoin); //Briefly ran to make sure spatial join worked, it did. However, when I first ran this command, EE originally gave an error saying "Error: 'system:index'

//is already specified". I solved this by going into the Crashes\_All fusion table where KSI\_Crashes comes from and changed the 'system:index' column name to "index", eliminating this error.

var HIN\_Congested = HINCongested\_KSICrashes\_SpatialJoin.map(KSICrashCount\_Finder); //CREATES COLUMN IN HIN\_Congested SHOWING EACH ROAD'S NUMBER OF KSI CRASHES HAPPENING ON IT

//print(HIN\_Congested); //Briefly ran to make sure KSI crash count column creation worked, it did.

//Step 7e: Find KSI crash rate per mile for each segment on congested HIN:

var HIN\_Congested = HIN\_Congested.map(KSICrashRate\_Finder); //CREATES COLUMN IN HIN\_Congested SHOWING EACH ROAD'S KSI CRASH RATE PER MILE

//print(HIN\_Congested); //Briefly ran OUTSIDE OF BUTTON to make sure KSI crash rate per mile column creation worked, it did.

var Map3Button1 = ui.Button({

label: 'Click Me to See: Congested HIN Dataset Structure showing Rd Lengths and Spatial Joined KSI Crashes',

onClick: function() {

print(HIN\_Congested);

}

});

//STEP 8: CREATE FOURTH MAP OF HIN ROADS THAT ARE SIMULTAENOUSLY MORE DANGEROUS AND CONGESTED THAN REST OF HIN

//Step 8a: Create the map and add base layers:

var Map4 = ui.Map({

lon: -75.163617,

lat: 39.952370,

zoom: 11

});

Map4.addLayer(Phila\_Boundary\_Transparent, {palette: '000000'}, 'Philadelphia Boundary');

Map4.addLayer(City\_HIN, {color:'96979B'}, 'Entire HIN');

Map4.addLayer(HIN\_Congested, {color:'BB7025'}, 'Congested HIN Roads');

Map4.addLayer(KSI\_Crashes, {color:'C7C7C7'}, 'KSI Crash Locations', false); //ADDS KSI CRASH POINTS TO MAP

//Step 8b: Add new column to entire HIN calculating each street segment's length, and find total city HIN mileage

var City\_HIN = City\_HIN.map(Length\_Finder);

var Adder = ee.Reducer.sum( ); //Reducer which sums up a column, WILL ALSO USE FOR STEPS 9 THROUGH 12

//print(ee.Number(City\_HIN.reduceColumns(Adder, ['length']))); //Shows a total of 283.839622074294 miles of HIN road

//Step 8c: Add new field to entire HIN counting each segment's raw number of KSI crashes

var CityHIN\_KSICrashes\_SpatialJoin = distSaveAll.apply(City\_HIN, KSI\_Crashes, distFilter); //Spatially joins KSI crashes to all of HIN

var City\_HIN = CityHIN\_KSICrashes\_SpatialJoin.map(KSICrashCount\_Finder); //Makes column showing each HIN street's raw KSI crash count

//Step 8d: Find KSI crash rate per mile for each segment on entire HIN

var City\_HIN = City\_HIN.map(KSICrashRate\_Finder);

//print(City\_HIN); //Briefly ran to make sure length, KSI crash count and rate per mile column creations worked, they did.

//Step 8e: Find average KSI crash rate per mile for entire HIN

var mean = ee.Reducer.mean(); //Same reducer used in Step 3

var CityHIN\_AvgKSICrashRate = ee.Number(City\_HIN.reduceColumns(mean, ['KSICrshRte'])); //Turns CityHIN\_AvgKSICrashRate from a dictionary to a number using ee.Number() to "cast" the result as a

//number

//print(CityHIN\_AvgKSICrashRate); //Average KSI crash rate for entire HIN is 3.900499074286767 KSI crashes per mile according to the output of this line

//Step 8f: Map out congested roads with higher KSI crash rates per mile compared to entire HIN

var HIN\_CongestedandDangerous\_Roads = HIN\_Congested.filterMetadata( 'KSICrshRte', 'greater\_than', 3.900499074286767 ); //Filters out the most congested roads which have crash rates higher than

//the rest of the HIN

Map4.addLayer(HIN\_CongestedandDangerous\_Roads, {color:'FE1004'}, 'Most Congested and Dangerous HIN Roads'); // MAPS FINAL CONSISTENTLY CONGESTED ROADS THAT ARE MORE DANGEROUS THAN REST OF HIN

//Step 8g: Find percentage of HIN road that is most congested and dangerous

//print('Total Most. Cong. Dang. HIN Mi.', ee.Number(HIN\_CongestedandDangerous\_Roads.reduceColumns(Adder, ['length']))); //Shows a total of 67.65080668920355 miles of HIN road that is considered the most dangerous and congested

//print('Most. Cong. Dang. % of HIN Mi.', ee.Number((67.65080668920355/283.839622074294)\*100)); //Shows 23.83416599656273% of all HIN road is simultaneously the most congested and dangerous

//print('Other % of HIN Mi.', ee.Number(100-23.83416599656273)); //Shows 76.16583400343727% of all HIN road is NOT simultaneously the most congested and dangerous

var Map4Button1 = ui.Button({

label: 'Click Me to See: Total Most Congested and Dangerous HIN Miles',

onClick: function() {

print(ee.Number(HIN\_CongestedandDangerous\_Roads.reduceColumns(Adder, ['length'])));

}

});

var Map4Button2 = ui.Button({

label: 'Click Me to See: % of Total HIN Miles that is the Most Congested and Dangerous',

onClick: function() {

print(ee.Number((67.65080668920355/283.839622074294)\*100));

}

});

var Map4Button3 = ui.Button({

label: 'Click Me to See: % of Total HIN Miles that is NOT the Most Congested and Dangerous',

onClick: function() {

print(ee.Number(100-23.83416599656273));

}

});

//Step 8h: Create chart displaying percentage of HIN road that is most congested and dangerous for user (found out how to do from https://developers.google.com/earth-engine/charts):

var HIN\_CongestedandDangerous\_Roads\_Chart\_Data = { //Creates variable containing a table of the data to be charted

cols: [{id: 'Level', label: 'Danger and Congestion Level', type: 'string'}, //Creates a string column detailing HIN road types

{id: 'Pct', label: '% of HIN Road Mileage', type: 'number'}], //Creates a numerical column detailing each HIN road type's percentage of all HIN road miles

rows: [{c: [{v: 'Least'}, {v: 76.16583400343727}]}, //Assigns value of 76.16583400343727% for the Other HIN road type for the % of HIN Road Mileage column

{c: [{v: 'Most'}, {v: 23.83416599656273}]}], //Assigns value of 23.83416599656273% for the Most Congested and Dangerous HIN road type for the % of HIN Road Mileage column

}; //Creates dataset to chart

var options = {

title: 'HIN Roads by Level of Danger and Congestion', //Creates graph title

vAxis: {title: 'Danger and Congestion Level'}, //Creates y-axis title (vAxis means "vertical axis" which is the y-axis)

legend: {position: 'none'}, //Makes it so that there is no legend

hAxis: {title: '% of HIN Road Mileage'} //Creates x-axis title (hAxis means "horizontal axis" which is the x-axis)

};

var HIN\_CongestedandDangerous\_Roads\_Chart = new ui.Chart(HIN\_CongestedandDangerous\_Roads\_Chart\_Data, 'BarChart', options); //Creates variable with the chart in it and makes it a bar chart

//print(HIN\_CongestedandDangerous\_Roads\_Chart); //Prints chart

var Map4Button4 = ui.Button({

label: 'Click Me to See: HIN Roads by Level of Danger and Congestion Graph',

onClick: function() {

print(HIN\_CongestedandDangerous\_Roads\_Chart);

}

}); //This button instead will show the graph created above

//MAIN FINDING: MINORITY OF HIN ROADS ARE CONSIDERED THE MOST CONGESTED AND DANGEROUS OUT OF ANY HIN ROAD

//STEP 9: CREATE FIFTH MAP EXAMINING STREET WIDTHS OF MOST DANGEROUS AND CONGESTED HIN ROADS:

//Step 9a: Create the map and add base layers:

var Map5 = ui.Map({

lon: -75.163617,

lat: 39.952370,

zoom: 11

});

Map5.addLayer(Phila\_Boundary\_Transparent, {palette: '000000'}, 'Philadelphia Boundary');

Map5.addLayer(City\_HIN, {color:'96979B'}, 'Entire HIN');

Map5.addLayer(HIN\_CongestedandDangerous\_Roads, {color:'FE1004'}, 'Most Congested and Dangerous HIN Roads');

//Step 9b: Find which of the most congested and dangerous HIN roads are also major or minor arterials:

//I define a wide, major HIN road as being a major or minor arterial. According to the metadata for the Philadelphia Streets Department Street Centerlines shapefile at

//http://metadata.phila.gov/#home/datasetdetails/555f8133f15fcb6c6ed44132/representationdetails/556dd4da3da941e307401194/, streets with a class of 2 are major arterial and 3 are minor arterial.

//Highways are excluded from city HIN dataset.

var Major\_Arterials = Street\_Centerlines.filterMetadata('CLASS', 'equals', 2);

var Minor\_Arterials = Street\_Centerlines.filterMetadata('CLASS', 'equals', 3);

var Wide\_Streets = Major\_Arterials.merge(Minor\_Arterials);

Map5.addLayer(Wide\_Streets, {color:'B49463'}, 'All Phila. Wide Streets', false); //MAPS ALL PHILADELPHIA WIDE STREETS REGARDLESS OF THEIR POSITION ON THE HIN

var HIN\_CongestedandDangerous\_WideRoads = intersection\_Join.apply(HIN\_CongestedandDangerous\_Roads, Wide\_Streets, intersection\_Filter); //Intersects most congested and dangerous HIN roads with

//widest streets to find which most congested and dangerous HIN roads are also wide. Done USING INTERSECTION METHOD FROM STEP 5A

Map5.addLayer(HIN\_CongestedandDangerous\_WideRoads, {color:'2703FF'}, 'Most Cong. and Dang. Wide HIN Rds'); //MAPS THE MOST DANGEROUS CONGESTED HIN ROADS WHICH ARE ALSO WIDE

//Step 9c: Find number of most congested and dangerous HIN road mileage that is wide:

//print('Wide Roads: Total Most. Cong. Dang. HIN Mi.', ee.Number(HIN\_CongestedandDangerous\_WideRoads.reduceColumns(Adder, ['length']))); //Shows a total of 48.3473459084451 miles of most congested and dangerous HIN road that is wide

var Map5Button1 = ui.Button({

label: 'Click Me to See: # of Wide Most Congested and Dangerous HIN Miles',

onClick: function() {

print(ee.Number(HIN\_CongestedandDangerous\_WideRoads.reduceColumns(Adder, ['length'])));

}

});

//Step 9d: Find percentage of most congested and dangerous HIN road mileage that is wide:

//print('Wide Roads: % of Most. Cong. Dang. HIN Mi.', ee.Number((48.3473459084451/67.65080668920355)\*100)); //Shows 71.46603015476073% of the most congested and dangerous HIN road mileage is simultaneously wide. Total mileage of HIN road

//that is most congested and dangerous taken from step 8f

var Map5Button2 = ui.Button({

label: 'Click Me to See: % of Most Congested and Dangerous HIN Miles that is Wide',

onClick: function() {

print(ee.Number((48.3473459084451/67.65080668920355)\*100));

}

});

//print('Narrow Roads: % of Most. Cong. Dang. HIN Mi.', ee.Number(100-71.46603015476073)); //Shows 28.5339698452% of the most congested and dangerous HIN road mileage is NOT wide. Total mileage of HIN road

//that is most congested and dangerous taken from step 8f

var Map5Button3 = ui.Button({

label: 'Click Me to See: % of Most Congested and Dangerous HIN Miles that is Narrow',

onClick: function() {

print(ee.Number(100-71.46603015476073));

}

});

//Step 9e: Create chart displaying percentage of most congested and dangerous HIN road mileage that is wide:

var HIN\_CongestedandDangerous\_WideRoads\_Chart\_Data = {

cols: [{id: 'Width', label: 'Road Width', type: 'string'},

{id: 'Pct', label: '% of Road Mileage', type: 'number'}],

rows: [{c: [{v: 'Wide'}, {v: 71.46603015476073}]},

{c: [{v: 'Narrow'}, {v: 28.5339698452}]}],

};

var options = {

title: 'Most Congested and Dangerous HIN Roads by Street Width',

vAxis: {title: 'Road Width'},

legend: {position: 'none'},

hAxis: {title: '% of Road Mileage', ticks: [{v: 0}, {v: 25}, {v: 50}, {v: 75}, {v: 100}]} //X axis scale only went up to 80, so this makes it go up to 100. Found out how to do from https://developers.google.com/earth-engine/charts\_feature\_by

};

var HIN\_CongestedandDangerous\_WideRoads\_Chart = new ui.Chart(HIN\_CongestedandDangerous\_WideRoads\_Chart\_Data, 'BarChart', options);

//print(HIN\_CongestedandDangerous\_WideRoads\_Chart);

var Map5Button4 = ui.Button({

label: 'Click Me to See: Most Congested and Dangerous HIN Roads by Street Width Graph',

onClick: function() {

print(HIN\_CongestedandDangerous\_WideRoads\_Chart);

}

});

//MAIN FINDING: MOST HIN ROADS THAT ARE THE MOST CONGESTED AND DANGEROUS OUT OF ANY HIN ROAD ARE ALSO CONSIDERED WIDE. THIS IS NOT SURPRISING CONSIDERING HOW WIDER ROADS CAN BE BUSIER

//STEP 10: CREATE SIXTH MAP EXAMINING WEALTH OF CENSUS TRACTS WHERE MOST DANGEROUS AND CONGESTED HIN ROADS ARE LOCATED:

//Step 10a: Create the map and add base layers:

var Map6 = ui.Map({

lon: -75.163617,

lat: 39.952370,

zoom: 11

});

Map6.addLayer(Phila\_Boundary\_Transparent, {palette: '000000'}, 'Philadelphia Boundary');

Map6.addLayer(City\_HIN, {color:'96979B'}, 'Entire HIN');

Map6.addLayer(HIN\_CongestedandDangerous\_Roads, {color:'FE1004'}, 'Most Congested and Dangerous HIN Roads');

//Step 10b: Join median household income table to Philadelphia census tracts

//First had to get rid of periods and dashes from column names in census tract polygons, median household income and median household income metadata fusion tables, and made sure their census

//tract identification number columns were numeric in Google Fusion Tables. Also had to remove all dashes from THE ATTRIBUTES on top of the titles, because this caused problems with the rest of this step

var TheFILTER = ee.Filter.equals('GEOID10', null, 'GEOid2', null ); //Joins median household income data (secondary features) to census tract polygons (primary features) based on their common

//census tract identification number columns

var TheJOIN = ee.Join.inner('primary','secondary'); //Establishes the join

var CensusTracts\_WithMedHHIncs = TheJOIN.apply(Census\_Tracts, CT\_Median\_HH\_Incs, TheFILTER); //Does the join

//print(CensusTracts\_WithMedHHIncs); //Briefly ran to see this stage of the join worked, it did

//print(CT\_Median\_HH\_Incs\_Metadata); //Shows that the "HC01\_EST\_VC13" variable name in CensusTracts\_WithMedHHIncs shows each tract's median household income

var CensusTracts\_WithMedHHIncs\_New = CensusTracts\_WithMedHHIncs.map(function(element){

var PRIMARY = ee.Feature(element.get('primary'));

var SECONDARY = ee.Feature(element.get('secondary'));

var IDNUM = PRIMARY.get('GEOID10');

var CTNAME = PRIMARY.get('NAMELSAD10');

var MEDHHINC = SECONDARY.get('HC01\_EST\_VC13');

var geom = ee.Feature(element.get('primary')).geometry();

return ee.Feature(geom, {'IDNUM':IDNUM, 'CTNAME':CTNAME, 'MEDHHINC':MEDHHINC});

}); //Final stage of the join which creates the final dataset with the Philadelphia census tracts and their median household incomes

//print(CensusTracts\_WithMedHHIncs\_New); //Briefly ran to see if join worked, it did

//Step 10c: Find the median Philadelphia census tract median household income

var median = ee.Reducer.median( ); //Median reducer, will apply to the MEDHHINC values for all Philadelphia census tracts to find median of them

//print(ee.Number(CensusTracts\_WithMedHHIncs\_New.reduceColumns(median, ['MEDHHINC']))); //Shows median Philadelphia census tract median household income to be $39370.153846153844

//Step 10d: See which of the most congested and dangerous HIN roads are by tracts that are less than the median Philadelphia census tract median household income

var PoorTracts = CensusTracts\_WithMedHHIncs\_New.filterMetadata('MEDHHINC', 'less\_than', 39370.153846153844); //Selects out tracts defined as being poorer than rest of Philadelphia

var PoorTracts\_Transparent = empty.paint({

featureCollection: PoorTracts,

color: 1,

width: 0.75

}); //Creates transparent version of the poor tracts feature collection, the variable "empty" is taken from step 1d

Map6.addLayer(PoorTracts\_Transparent, {palette: '000000'}, 'Poorer Census Tracts'); //MAPS POORER CENSUS TRACTS

var HIN\_CongestedandDangerous\_PoorAreas = intersection\_Join.apply(HIN\_CongestedandDangerous\_Roads, PoorTracts, intersection\_Filter); //Intersects most congested and dangerous HIN roads with

//poor tracts to find which most congested and dangerous HIN roads are also in poor areas. Done USING INTERSECTION METHOD FROM STEP 5A

Map6.addLayer(HIN\_CongestedandDangerous\_PoorAreas, {color:'03FF11'}, 'Most Cong. and Dang. HIN Rds in Poorer Areas'); //MAPS THE MOST DANGEROUS CONGESTED HIN ROADS WHICH ARE ALSO IN POOR

//AREAS

//Step 10e: Find number of most congested and dangerous HIN road mileage that is in poor areas

//print('Poor Areas: Total Most. Cong. Dang. HIN Mi.', ee.Number(HIN\_CongestedandDangerous\_PoorAreas.reduceColumns(Adder, ['length']))); //Shows a total of 56.443049649939596 miles of most congested and dangerous HIN road that is in poor

//areas

var Map6Button1 = ui.Button({

label: 'Click Me to See: # of Most Congested and Dangerous HIN Miles in Poorer Neighborhoods',

onClick: function() {

print(ee.Number(HIN\_CongestedandDangerous\_PoorAreas.reduceColumns(Adder, ['length'])));

}

});

//Step 10f: Find percentage of most congested and dangerous HIN road mileage that is in poor areas

//print('Poor Neighborhoods: % of Most. Cong. Dang. HIN Mi.', ee.Number((56.443049649939596/67.65080668920355)\*100)); //Shows 83.43292920252404% of the most congested and dangerous HIN road mileage is in poor areas. Total mileage of HIN road that is

//most congested and dangerous taken from step 8f

//print('Affluent Neighborhoods: % of Most. Cong. Dang. HIN Mi.', ee.Number(100-83.43292920252404)); //Shows 16.5670707975% of the most congested and dangerous HIN road mileage is NOT in poor areas. Total mileage of HIN road

//that is most congested and dangerous taken from step 8f

var Map6Button2 = ui.Button({

label: 'Click Me to See: % of Most Congested and Dangerous HIN Miles in Poorer Neighborhoods',

onClick: function() {

print(ee.Number((56.443049649939596/67.65080668920355)\*100));

}

});

var Map6Button3 = ui.Button({

label: 'Click Me to See: % of Most Congested and Dangerous HIN Miles in Affluent Neighborhoods',

onClick: function() {

print(ee.Number(100-83.43292920252404));

}

});

//Step 10g: Create chart displaying percentage of most congested and dangerous HIN road mileage that is in poor areas

var HIN\_CongestedandDangerous\_PoorAreas\_Chart\_Data = {

cols: [{id: 'Wealth', label: 'Neighborhood Wealth', type: 'string'},

{id: 'Pct', label: '% of Road Mileage', type: 'number'}],

rows: [{c: [{v: 'Poor'}, {v: 83.43292920252404}]},

{c: [{v: 'Affluent'}, {v: 16.5670707975}]}],

};

var options = {

title: 'Most Congested and Dangerous HIN Roads by Neighborhood Wealth',

vAxis: {title: 'Neighborhood Wealth'},

legend: {position: 'none'},

hAxis: {title: '% of Road Mileage'}

};

var HIN\_CongestedandDangerous\_PoorAreas\_Chart = new ui.Chart(HIN\_CongestedandDangerous\_PoorAreas\_Chart\_Data, 'BarChart', options);

//print(HIN\_CongestedandDangerous\_PoorAreas\_Chart);

var Map6Button4 = ui.Button({

label: 'Click Me to See: Most Congested and Dangerous HIN Roads by Neighborhood Wealth Graph',

onClick: function() {

print(HIN\_CongestedandDangerous\_PoorAreas\_Chart);

}

});

//MAIN FINDING: MOST HIN ROADS THAT ARE THE MOST CONGESTED AND DANGEROUS OUT OF ANY HIN ROAD ARE IN POORER NEIGHBORHOODS. ON ONE HAND, THIS IS SURPRISING BECAUSE THIS CAN MEAN THERE ARE MANY

//MOTORISTS IN POORER AREAS, BUT CARS ARE MORE EXPENSIVE THAN TAKING PUBLIC TRANSIT. ON THE OTHER HAND, THIS IS NOT SURPRISING BECAUSE POORER AREAS TRADITIONALLY HAVE LESSER QUALITY

//INFRASTRUCTURE THAN WEALTHIER AREAS, WHICH CAN CAUSE CONGESTION

//STEP 11: CREATE SEVENTH MAP EXAMINING DISTANCES OF MOST DANGEROUS AND CONGESTED HIN ROADS FROM RAIL STATIONS:

//Step 11a: Create the map and add base layers:

var Map7 = ui.Map({

lon: -75.163617,

lat: 39.952370,

zoom: 11

});

Map7.addLayer(Phila\_Boundary\_Transparent, {palette: '000000'}, 'Philadelphia Boundary');

Map7.addLayer(City\_HIN, {color:'96979B'}, 'Entire HIN');

Map7.addLayer(HIN\_CongestedandDangerous\_Roads, {color:'FE1004'}, 'Most Congested and Dangerous HIN Roads');

//Step 11b. Create walking distance buffer around rail stops

//First made Google read the table's latitude and longitude columns as a "two-column" location in Google Sheets so Earth Engine could read this as a point shapefile

//Map6.addLayer(Rail\_Stations, {color:'000000'}, 'Rail Stations'); //Briefly ran to make sure rail stations mapped properly after making the latitude and longitude columns as a "two-column"

//location in Google Sheets. It works

var Phila\_Stations = Rail\_Stations.filter(Phila\_Filter); //Clips train stations so that I am working with just ones in Philadelphia. The "Phila\_Filter" used for the clip comes from step 2a

Map7.addLayer(Phila\_Stations, {color:'000000'}, 'Rail Stations'); //ADDS RAIL STATIONS TO 4TH MAP and ran to make sure clip worked, it did

var Phila\_Stations\_Buffer = Phila\_Stations.geometry().buffer(402.336); //Creates a buffer of a quarter mile around rail stops (402.336 m = 0.25 mi). Chose 0.25 miles because this is considered

//"walking distance" in the US according to https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3377942/

//Map6.addLayer(Phila\_Stations\_Buffer, {color:'000000'}, 'Phila\_Stations\_Buffer'); //Briefly ran to see if buffering worked properly, it did

//Step 11c. Find which most congested and dangerous HIN roads travel within walking distance of a rail stop

var HIN\_CongestedandDangerous\_ByTransit = intersection\_Join.apply(HIN\_CongestedandDangerous\_Roads, Phila\_Stations\_Buffer, intersection\_Filter); //Intersects most congested and dangerous HIN roads

//with rail stop buffers to find which most congested and dangerous HIN roads are also within walking distance of rail stops. Done USING INTERSECTION METHOD FROM STEP 5A

Map7.addLayer(HIN\_CongestedandDangerous\_ByTransit, {color:'FF9403'}, 'Most Cong. and Dang. HIN Rds near Rail Stops'); //MAPS THE MOST DANGEROUS CONGESTED HIN ROADS WHICH ARE ALSO NEAR RAIL

//STOPS

//Step 11d: Find number of most congested and dangerous HIN road mileage that is near rail stops

//print('Rail Stops: Total Most. Cong. Dang. HIN Mi.', ee.Number(HIN\_CongestedandDangerous\_ByTransit.reduceColumns(Adder, ['length']))); //Shows a total of 39.751470596069 miles of most congested and dangerous HIN road that is near rail stops

var Map7Button1 = ui.Button({

label: 'Click Me to See: # of Most Congested and Dangerous HIN Miles near Rail Stops',

onClick: function() {

print(ee.Number(HIN\_CongestedandDangerous\_ByTransit.reduceColumns(Adder, ['length'])));

}

});

//Step 11e: Find percentage of most congested and dangerous HIN road mileage that is in near rail stops

//print('Close to Rail Stops: % of Most. Cong. Dang. HIN Mi.', ee.Number((39.751470596069/67.65080668920355)\*100)); //Shows 58.75978800769123% of the most congested and dangerous HIN road mileage is near rail stops. Total mileage of HIN road that is

//most congested and dangerous taken from step 8f

//print('Far from Rail Stops: % of Most. Cong. Dang. HIN Mi.', ee.Number(100-58.75978800769123)); //Shows 41.2402119923% of the most congested and dangerous HIN road mileage is NOT in poor areas. Total mileage of HIN road

//that is most congested and dangerous taken from step 8f

var Map7Button2 = ui.Button({

label: 'Click Me to See: % of Most Congested and Dangerous HIN Miles that is near Rail Stops',

onClick: function() {

print(ee.Number((39.751470596069/67.65080668920355)\*100));

}

});

var Map7Button3 = ui.Button({

label: 'Click Me to See: % of Most Congested and Dangerous HIN Miles that is far from Rail Stops',

onClick: function() {

print(ee.Number(100-58.75978800769123));

}

});

//Step 11f: Create chart displaying percentage of most congested and dangerous HIN road mileage that is in near rail stops

var HIN\_CongestedandDangerous\_ByTransit\_Chart\_Data = {

cols: [{id: 'Proximity', label: 'Proximity to Rail Stops', type: 'string'},

{id: 'Pct', label: '% of Road Mileage', type: 'number'}],

rows: [{c: [{v: 'Close'}, {v: 58.75978800769123}]},

{c: [{v: 'Far'}, {v: 41.2402119923}]}],

};

var options = {

title: 'Most Congested and Dangerous HIN Roads by Proximity to Rail Stops',

vAxis: {title: 'Proximity'},

legend: {position: 'none'},

hAxis: {title: '% of Road Mileage', ticks: [{v: 0}, {v: 25}, {v: 50}, {v: 75}, {v: 100}]}, //X axis scale only went up to 60 originally which was bad since that distorted the difference between the proportions when in reality the difference is smaller

};

var HIN\_CongestedandDangerous\_ByTransit\_Chart = new ui.Chart(HIN\_CongestedandDangerous\_ByTransit\_Chart\_Data, 'BarChart', options);

//print(HIN\_CongestedandDangerous\_ByTransit\_Chart);

var Map7Button4 = ui.Button({

label: 'Click Me to See: Most Congested and Dangerous HIN Roads by Proximity to Rail Stops Graph',

onClick: function() {

print(HIN\_CongestedandDangerous\_ByTransit\_Chart);

}

});

//MAIN FINDING: MOST HIN ROADS THAT ARE THE MOST CONGESTED AND DANGEROUS OUT OF ANY HIN ROAD ARE ALSO WITHIN WALKING DISTANCE OF RAIL STOPS. THIS MAY BE THE MOST SURPRISING FINDING, AS ROADS

//FARTHER FROM RAIL STOPS WOULD LOGICALLY BE MORE CONGESTED

//STEP 12: CREATE EIGHTH MAP EXAMINING DISTANCES OF MOST DANGEROUS AND CONGESTED HIN ROADS FROM COMMERCIAL CORRIDORS:

//Step 12a: Create the map and add base layers:

var Map8 = ui.Map({

lon: -75.163617,

lat: 39.952370,

zoom: 11

});

Map8.addLayer(Phila\_Boundary\_Transparent, {palette: '000000'}, 'Philadelphia Boundary');

Map8.addLayer(City\_HIN, {color:'96979B'}, 'Entire HIN');

Map8.addLayer(HIN\_CongestedandDangerous\_Roads, {color:'FE1004'}, 'Most Congested and Dangerous HIN Roads');

//Step 12b. Find which most congested and dangerous HIN roads are located in commercial corridors

var CommercialCorridors\_Transparent = empty.paint({

featureCollection: Commercial\_Corridors,

color: 1,

width: 0.75

}); //Creates transparent version of the commercial corridors feature collection, the variable "empty" is taken from step 1d

Map8.addLayer(CommercialCorridors\_Transparent, {palette: '000000'}, 'Commercial Corridors'); //MAPS COMMERCIAL CORRIDORS

var HIN\_CongestedandDangerous\_ByCommercial = intersection\_Join.apply(HIN\_CongestedandDangerous\_Roads, Commercial\_Corridors, intersection\_Filter); //Intersects most congested and dangerous

//HIN roads with commercial corridors to find which most congested and dangerous HIN roads are also within commercial corridors. Done USING INTERSECTION METHOD FROM STEP 5A

Map8.addLayer(HIN\_CongestedandDangerous\_ByCommercial, {color:'F403FF'}, 'Most Cong. and Dang. HIN Rds in Commercial Corridors'); //MAPS THE MOST DANGEROUS CONGESTED HIN ROADS WHICH ARE

//ALSO IN COMMERCIAL CORRIDORS

//Step 12c: Find number of most congested and dangerous HIN road mileage that is in commercial corridors

//print('Comm. Corrs.: Total Most. Cong. Dang. HIN Mi.', ee.Number(HIN\_CongestedandDangerous\_ByCommercial.reduceColumns(Adder, ['length']))); //Shows a total of 65.97505508393522 miles of most congested and dangerous HIN road that is in

//commercial corridors

var Map8Button1 = ui.Button({

label: 'Click Me to See: # of Most Congested and Dangerous HIN Miles in Commercial Corridors',

onClick: function() {

print(ee.Number(HIN\_CongestedandDangerous\_ByCommercial.reduceColumns(Adder, ['length'])));

}

});

//Step 12d: Find percentage of most congested and dangerous HIN road mileage that is in commercial corridors

//print('Inside Comm. Corrs.: % of Most. Cong. Dang. HIN Mi.', ee.Number((65.97505508393522/67.65080668920355)\*100)); //Shows 97.5229392119935% of the most congested and dangerous HIN road mileage is in commercial corridors. Total mileage of HIN road

//that is most congested and dangerous taken from step 8f

//print('Outside Comm. Corrs.: % of Most. Cong. Dang. HIN Mi.', ee.Number(100-97.5229392119935)); //Shows 2.47706078801% of the most congested and dangerous HIN road mileage is NOT in poor areas. Total mileage of HIN road

//that is most congested and dangerous taken from step 8f

var Map8Button2 = ui.Button({

label: 'Click Me to See: % of Most Congested and Dangerous HIN Miles that is in Commercial Corridors',

onClick: function() {

print(ee.Number((65.97505508393522/67.65080668920355)\*100));

}

});

var Map8Button3 = ui.Button({

label: 'Click Me to See: % of Most Congested and Dangerous HIN Miles that is outside of Commercial Corridors',

onClick: function() {

print(ee.Number(100-97.5229392119935));

}

});

//Step 12e: Create chart displaying percentage of most congested and dangerous HIN road mileage that is in commercial corridors

var HIN\_CongestedandDangerous\_ByCommercial\_Chart\_Data = {

cols: [{id: 'Proximity', label: 'Proximity to Commercial Corridors', type: 'string'},

{id: 'Pct', label: '% of Road Mileage', type: 'number'}],

rows: [{c: [{v: 'Inside'}, {v: 97.5229392119935}]},

{c: [{v: 'Outside'}, {v: 2.47706078801}]}],

};

var options = {

title: 'Most Congested and Dangerous HIN Roads by Proximity to Commercial Corridors',

vAxis: {title: 'Proximity'},

legend: {position: 'none'},

hAxis: {title: '% of Road Mileage'}

};

var HIN\_CongestedandDangerous\_ByCommercial\_Chart = new ui.Chart(HIN\_CongestedandDangerous\_ByCommercial\_Chart\_Data, 'BarChart', options);

var Map8Button4 = ui.Button({

label: 'Click Me to See: Most Congested and Dangerous HIN Roads by Proximity to Commercial Corridors Graph',

onClick: function() {

print(HIN\_CongestedandDangerous\_ByCommercial\_Chart);

}

});

//MAIN FINDING: MOST HIN ROADS THAT ARE THE MOST CONGESTED AND DANGEROUS OUT OF ANY HIN ROAD ARE ALSO WITHIN COMMERCIAL CORRIDORS. THIS IS NOT SURPRISING SINCE THE MANY COMMERCIAL PARCELS IN

//THESE CORRIDORS SERVE TO ATTRACT MANY DRIVING VISITORS

//OVERALL FINDING FROM THIS PROJECT: THE TYPICAL PHILADELPHIA HIN ROAD THAT IS SIMULTANEOUSLY MORE CONGESTED AND DANGEROUS THAN OTHER PHILADELPHIA HIN ROADS IS: WIDER THAN OTHER CITY ROADS

//REGARDLESS OF DANGER AND CONGESTION LEVELS, IN NEIGHBORHOODS POORER THAN REST OF CITY, CLOSE TO RAIL STOPS AND IN COMMERCIAL CORRIDORS.

//STEP 13: CREATE USER INTERFACE

//Step 13a: Create instructions and prompts:

var FirstPrompt = prompt("Welcome to my application which lets you view information about congested and dangerous roads along Philadelphia's High Injury Network (HIN). If this is your first time viewing this application, type 'begin' in the text box below. If this is not your first time, type 'skip' in the text box to skip to the map options."); //First prompt where user can either go through all instruction if it's their first time using this, or skip to choosing maps if it is not

if(FirstPrompt=='begin'){ //Means "if the user types 'begin' in first prompt, then everything in this if statement will happen"

alert("When in the application, buttons detailing all statistics and graphs can be seen by first viewing the code at the top of the screen, and then viewing the console tab that appears on the right side of the screen. To view full button names, widen the tab by clicking the dots on the left border of the console tab and drag that to the left. Once a button is clicked, the results of it can be seen by scrolling to the bottom of the console tab."); //Shows instructions

alert("After clicking OK below when you finish reading this, you will have to type a letter into a text box detailing which of the application's maps you would like to see. To toggle between the application's maps, first refresh the page, then type 'skip' at the first prompt, and then type in a new letter to see a new map. Some maps may be slow to load as indicated by moving dark gray bars on the right of the 'Layers' control, but please have patience as the map you view will load shortly."); //Shows instructions

var SecondPrompt = prompt("Please type the letter into the text box below that corresponds with what information about Philadelphia's High Injury Network (HIN) you would like to see:\n\na - City HIN vs. Clipped Philadelphia INRIX datasets to see how joining of congestion data to city HIN worked\nb - HIN with TTI and PTI busiest hour period data versus HIN with just TTI and PTI data joined to see how the busiest hour column creations worked\nc - KSI Crashes v. Congested HIN to see how creation of congested HIN dataset, creation of lengths in miles for each congested road segment, and spatial joining of KSI crash data to congested HIN worked\nd - The Most Congested and Dangerous HIN Roads\ne - The Most Congested and Dangerous Wide HIN Roads\nf - The Most Congested and Dangerous HIN Roads in Poor Neighborhoods\ng - The Most Congested and Dangerous HIN Roads near Rail Stops\nh - The Most Congested and Dangerous HIN Roads in Commercial Corridors");

//After viewing the two alerts with the instructions, the user then gets brought to this second prompt where they enter in the letter of the map they would like to see

}

else if(FirstPrompt=='skip'){ //This makes it so if the user types 'skip' in the first prompt, they simply get brought to the second prompt where they can choose which map to see

var SecondPrompt = prompt("Please type the letter into the text box below that corresponds with what information about Philadelphia's High Injury Network (HIN) you would like to see:\n\na - City HIN vs. Clipped Philadelphia INRIX datasets to see how joining of congestion data to city HIN worked\nb - HIN with TTI and PTI busiest hour period data versus HIN with just TTI and PTI data joined to see how the busiest hour column creations worked\nc - KSI Crashes v. Congested HIN to see how creation of congested HIN dataset, creation of lengths in miles for each congested road segment, and spatial joining of KSI crash data to congested HIN worked\nd - The Most Congested and Dangerous HIN Roads\ne - The Most Congested and Dangerous Wide HIN Roads\nf - The Most Congested and Dangerous HIN Roads in Poor Neighborhoods\ng - The Most Congested and Dangerous HIN Roads near Rail Stops\nh - The Most Congested and Dangerous HIN Roads in Commercial Corridors");

}

//Step 13b: Make it so when the user types the letter a, the map from step 2 will show up:

if(SecondPrompt=='a'){ //Means "if user writes 'a' in that text box"...

ui.root.clear(); //MEANS THE DEFAULT MAP WILL BE ELIMINATED. Found out how to do from https://developers.google.com/earth-engine/ui\_widgets

ui.root.add(Map1); //ADDS MAP1 TO THE BOTTOM HALF OF THE SCREEN TO REPLACE THE DEFAULT MAP WHICH WAS ELIMINATED IN THE LINE ABOVE. Found out how to do from https://developers.google.com/earth-engine/ui\_widgets

print(Map1Button1); //THIS MAKES THE BUTTONS ONLY FOR THAT STEP PRINT OUT WHEN THE USER SELECTS a

print(Map1Button2);

}

//Step 13c: Make it so when the user types the letter b, the map from step 4 will show up:

else if(SecondPrompt=='b'){

ui.root.clear();

ui.root.add(Map2);

print(Map2Button1);

print(Map2Button2);

}

//Step 13d: Make it so when the user types the letter c, the map from step 7 will show up:

else if(SecondPrompt=='c'){ //Means "if user writes 'b' instead of a or any other letter in that text box"...

ui.root.clear();

ui.root.add(Map3);

print(Map3Button1);

}

//Step 13e: Make it so when the user types the letter d, the map from step 8 will show up:

else if(SecondPrompt=='d'){

ui.root.clear();

ui.root.add(Map4);

print(Map4Button1);

print(Map4Button2);

print(Map4Button3);

print(Map4Button4);

}

//Step 13f: Make it so when the user types the letter e, the map from step 9 will show up:

else if(SecondPrompt=='e'){

ui.root.clear();

ui.root.add(Map5);

print(Map5Button1);

print(Map5Button2);

print(Map5Button3);

print(Map5Button4);

}

//Step 13g: Make it so when the user types the letter f, the map from step 10 will show up:

else if(SecondPrompt=='f'){

ui.root.clear();

ui.root.add(Map6);

print(Map6Button1);

print(Map6Button2);

print(Map6Button3);

print(Map6Button4);

}

//Step 13h: Make it so when the user types the letter g, the map from step 11 will show up:

else if(SecondPrompt=='g') {

ui.root.clear();

ui.root.add(Map7);

print(Map7Button1);

print(Map7Button2);

print(Map7Button3);

print(Map7Button4);

}

//Step 13i: Make it so when the user types the letter h, the map from step 12 will show up:

else if(SecondPrompt=='h') {

ui.root.clear();

ui.root.add(Map8);

print(Map8Button1);

print(Map8Button2);

print(Map8Button3);

print(Map8Button4);

}

//END OF CODE