In [1]:

```
%matplotlib inline

from __future__ import print_function # For py 2.7 compat

import ee
import os

from IPython.html import widgets
from IPython.display import display
from IPython.utils import traitlets
from IPython.core.display import Javascript
```

In [2]:

```
## requires modules ee, os, ipython, shapely, geopandas, statsmodels (scipy),
matplotlib & pyplot (numpy)
```

In [3]:

```
# This script assumes your authentification credentials are stored as operatori
ng system
# environment variables.
MY_SERVICE_ACCOUNT = os.environ.get('MY_SERVICE_ACCOUNT')
MY_PRIVATE_KEY_FILE = os.environ.get('MY_PRIVATE_KEY_FILE')

# Initialize the Earth Engine object, using your authentication credentials.
ee.Initialize(ee.ServiceAccountCredentials(MY_SERVICE_ACCOUNT,
MY_PRIVATE_KEY_FILE))
```

Tyler Erickson's Google Maps widget.

In [4]:

```
class GoogleMapsWidget(widgets.DOMWidget):
    _view_name = traitlets.Unicode('GoogleMapsView', sync=True)
    value = traitlets.Unicode(sync=True)
    description = traitlets.Unicode(sync=True)
    lat = traitlets.CFloat(0, help="Center latitude, -90 to 90", sync=True)
    lng = traitlets.CFloat(0, help="Center longitude, -180 to 180", sync=True)
    zoom = traitlets.CInt(0, help="Zoom level, 0 to ~25", sync=True)
    bounds = traitlets.List([], help="Visible bounds, [W, S, E, N]", sync=True)

    def __init__(self, lng=0.0, lat=0.0, zoom=2):
        self.lng = lng
        self.lat = lat
        self.zoom = zoom

    def addLayer(self, image, vis_params=None, name=None, visible=True):
        mapid = image.getMapId(vis_params)
        self.send({'command':'addLayer', 'mapid': mapid['mapid'], 'token':
mapid['token'], 'name': name, 'visible': visible})

    def center(self, lng, lat, zoom=None):
        self.send({'command': 'center', 'lng': lng, 'lat': lat, 'zoom': zoom})
```

Tyler's Javascript code for the widget.

In [5]:

```
%%javascript

require(["widgets/js/widget"], function(WidgetManager){
    var maps = [];

    // Define the GoogleMapsView
    var GoogleMapsView = IPython.DOMWidgetView.extend({

        render: function() {
            // Resize widget element to be 100% wide
            this.$el.css('width', '100%');

            // iframe source;  just enough to load Google Maps and let us poll
whether initialization is complete
            var src='<html style="height:100%"><head>' +
                '<scr'+'ipt src="http://maps.googleapis.com/maps/api/js?sensor=f
alse"></scr'+'ipt>' +

'<scr'+'ipt>google.maps.event.addDomListener(window,"load",function(){ready=tru
e});</scr'+'ipt>' +
                '</head>' +
                '<body style="height:100%; margin:0px; padding:0px"></body></htm
l>';

            // Create the Google Maps container element.
            this.$iframe = $('<iframe />')
                .css('width', '100%')
                .css('height', '500px')
                .attr('srcdoc', src)
                .appendTo(this.$el);

            var self = this; // hold onto this for initMapWhenReady

            // Wait until maps library has finished loading in iframe, then
create map
            function initMapWhenReady() {
                // Iframe document and window
                var doc = self.$iframe[0].contentDocument;
                var win = self.$iframe[0].contentWindow;
                if (!win || !win.ready) {
                    // Maps library not yet loaded;  try again soon
                    setTimeout(initMapWhenReady, 20);
                    return;
                }

                // Maps library finished loading.  Build map now.
                var mapOptions = {
                    center: new win.google.maps.LatLng(self.model.get('lat'), se
lf.model.get('lng')),
                    zoom: self.model.get('zoom')
                };
                var mapDiv = $('<div />')
                    .css('width', '100%')
                    .css('height', '100%')
                    .appendTo($(doc.body));
                self.map = new win.google.maps.Map(mapDiv[0], mapOptions);
```

```
                // Add an event listeners for user panning, zooming, and
resizing map
                // TODO(rsargent): Bind self across all methods, and save some
plumbing here
                win.google.maps.event.addListener(self.map, 'bounds_changed', fu
nction () {
                    self.handleBoundsChanged();
                });

                self.initializeLayersControl();
            }
            initMapWhenReady();
        },

        LayersControl: function(widget, controlDiv, map) {
            var win = widget.$iframe[0].contentWindow;
            var chicago = new win.google.maps.LatLng(41.850033, -87.6500523);

            // Set CSS styles for the DIV containing the control
            // Setting padding to 5 px will offset the control
            // from the edge of the map.
            controlDiv.style.padding = '5px';

            // Set CSS for the control border.
            var $controlUI = $('<div />')
                .css('backgroundColor', 'white')
                .css('borderStyle', 'solid')
                .css('borderWidth', '1px')
                .css('cursor', 'pointer')
                .css('textAlign', 'center')
                .appendTo($(controlDiv));

            // Set CSS for the control interior.
            var $controlContents = $('<div />')
                .css('fontFamily', 'Arial,sans-serif')
                .css('fontSize', '12px')
                .css('paddingLeft', '4px')
                .css('paddingRight', '4px')
                .css('paddingTop', '0px')
                .css('paddingBottom', '0px')
                .appendTo($controlUI);

            this.$controlTable = $('<table />')
                .append($('<tr><td colspan=2>Layers</td></tr>'))
                .appendTo($controlContents);
        },

        initializeLayersControl: function() {
            var doc = this.$iframe[0].contentDocument;
            var win = this.$iframe[0].contentWindow;

            // Create the DIV to hold the control and call the LayersControl()
constructor
            // passing in this DIV.

            var layersControlDiv = document.createElement('div');
            this.layersControl = new this.LayersControl(this, layersControlDiv,
this.map);
```

```
            layersControlDiv.index = 1;
            this.map.controls[win.google.maps.ControlPosition.TOP_RIGHT].push(la
yersControlDiv);
        },

        // Map geometry changed (pan, zoom, resize)
        handleBoundsChanged: function() {
            this.model.set('lng', this.map.getCenter().lng());
            this.model.set('lat', this.map.getCenter().lat());
            this.model.set('zoom', this.map.getZoom());
            var bounds = this.map.getBounds();
            var playgroundCompatible = [bounds.getSouthWest().lng(), bounds.getS
outhWest().lat(),
                                        bounds.getNorthEast().lng(), bounds.getNo
rthEast().lat()];
            this.model.set('bounds', playgroundCompatible);
            this.touch();
        },

        // Receive custom messages from Python backend
        on_msg: function(msg) {
            var win = this.$iframe[0].contentWindow;
            if (msg.command == 'addLayer') {
                this.addLayer(msg.mapid, msg.token, msg.name, msg.visible);
            } else if (msg.command == 'center') {
                this.map.setCenter(new win.google.maps.LatLng(msg.lat, msg.lng))
;
                if (msg.zoom !== null) {
                    this.map.setZoom(msg.zoom);
                }
            }
        },

        // Add an Earth Engine layer
        addLayer: function(mapid, token, name, visible) {
            var win = this.$iframe[0].contentWindow;
            var eeMapOptions = {
                getTileUrl: function(tile, zoom) {
                    var url = ['https://earthengine.googleapis.com/map',
                               mapid, zoom, tile.x, tile.y].join("/");
                    url += '?token=' + token
                    return url;
                },
                tileSize: new win.google.maps.Size(256, 256),
                opacity: visible ? 1.0 : 0.0,
            };

            // Create the overlay map type
            var mapType = new win.google.maps.ImageMapType(eeMapOptions);

            // Overlay the Earth Engine generated layer
            this.map.overlayMapTypes.push(mapType);

            // Update layer visibility control
            var maxSlider = 100;

            function updateOpacity() {
                mapType.setOpacity($checkbox[0].checked ? $slider[0].value / 100
.0 : 0);
```

```
            }

            var $checkbox = $('<input type="checkbox">')
                .prop('checked', visible)
                .change(updateOpacity);

            var $slider = $('<input type="range" />')
                .prop('min', 0)
                .prop('max', maxSlider)
                .prop('value', maxSlider)
                .css('width', '60px')
                .on('input', updateOpacity);

            // If user doesn't specify a layer name, create a default
            if (name === null) {
                name = 'Layer ' + this.map.overlayMapTypes.length;
            }

            var $row = $('<tr />');
            $('<td align="left"
/>').append($checkbox).append(name).appendTo($row);
            $('<td />').append($slider).appendTo($row);

            this.layersControl.$controlTable.append($row);
        }
    });

    // Register the GoogleMapsView with the widget manager.
    WidgetManager.register_widget_view('GoogleMapsView', GoogleMapsView);
});
```

Guido Lemoine's Feature Collection to Pandas Dataframe Function

In [6]:

```
from geopandas import GeoDataFrame
from shapely.geometry import shape


def fc2df(fc):
    # Convert a FeatureCollection into a pandas DataFrame

    # Features is a list of dict with the output
    features = fc.getInfo()['features']

    dictarr = []

    for f in features:
        # Store all attributes in a dict
        attr = f['properties']
        # and treat geometry separately
        attr['geometry'] = f['geometry']  # GeoJSON Feature!
        # attr['geometrytype'] = f['geometry']['type']
        dictarr.append(attr)

    df = GeoDataFrame(dictarr)
    # Convert GeoJSON features to shape
    #df['geometry'] = map(lambda s: shape(s), df.geometry)
    return df
```

```
# End fc2df
```

Retrieve coordinates of CT state polygon:

In [7]:

```
filename = "/Users/Kit/CTpolygon.txt"
vertex_list = []

with open(filename, 'r') as o_file:
    line = o_file.readline()[1:-3]
    while line:
        line = line.split(',')
        coords = [float(line[0]), float(line[1])]

        vertex_list.append(coords)
        line = o_file.readline()[2:-3]

print(vertex_list[0:10])
```

```
[[-72.3977171022447, 42.03309189931522], [-72.3885943398962,
42.03292197771742], [-72.3745266657436, 42.03280694244558], [-72.3681473358252,
42.03274819850335], [-72.3670431790677, 42.03273632331485], [-72.3580406358245,
42.03263917018669], [-72.3450571810366, 42.03235477126519], [-72.3368185442856,
42.032173740333064], [-72.3251543767583, 42.0319533775531 16], [-
72.3249740439011, 42.03194997230511]]
```

Get images of elevation and night lights; clip to CT polygon:

In [8]:

```
map = GoogleMapsWidget(lat=41.5, lng=-73, zoom=9) # lat, lng and zoom are
optional
display(map)
image1 = ee.Image('CGIAR/SRTM90_V4')
image2 = ee.Image('NOAA/DMSP-OLS/NIGHTTIME_LIGHTS/F152005')
ct_geom = ee.Feature.Polygon(vertex_list, {'name':'Connecticut'})
ct = ee.Feature(ct_geom)
image1 = image1.clip(ct)
image2 = image2.clip(ct)
map.addLayer(image=image1, name='Elevation',vis_params={'min':50, 'max':500})
map.addLayer(image=image2, name='Lights',vis_params={'min':0, 'max':100})
```

Make 1000 random points and use them to sample both layers:

In [9]:

```
region = ee.Feature.Rectangle(-73, 41, -72,  42)
rand_points = ee.FeatureCollection.randomPoints(region, 1000, 1000, 1)

sample1 = image1.addBands(image1).reduceToVectors(reducer="mean",
geometry=rand_points, geometryType="centroid", scale=30, crs="EPSG:4326")
sample2 = image2.reduceToVectors(reducer="mean", geometry=rand_points,
geometryType="centroid", scale=30, crs="EPSG:4326")
```

Convert samples (ee Feature Collections) to joined (geo)pandas

dataframe

In [10]:

```
sample1df = fc2df(sample1)
sample2df = fc2df(sample2)

## big assumption is that order was preserved
k = sample1df.join(sample2df, lsuffix='l', rsuffix='r')
del k['labell']
del k['geometryr']
del k['cf_cvg']
del k['avg_lights_x_pct']
del k['labelr']
k[0:10]
## option to export to csv, shp, xls, many other formats
```

Out[10]:

	geometryl	mean	stable_lights
0	{u'type': u'Point', u'geodesic': False, u'coor...	230	11
1	{u'type': u'Point', u'geodesic': False, u'coor...	198	8
2	{u'type': u'Point', u'geodesic': False, u'coor...	24	15
3	{u'type': u'Point', u'geodesic': False, u'coor...	187	21
4	{u'type': u'Point', u'geodesic': False, u'coor...	192	6
5	{u'type': u'Point', u'geodesic': False, u'coor...	166	6
6	{u'type': u'Point', u'geodesic': False, u'coor...	133	6
7	{u'type': u'Point', u'geodesic': False, u'coor...	66	8
8	{u'type': u'Point', u'geodesic': False, u'coor...	5	15
9	{u'type': u'Point', u'geodesic': False, u'coor...	157	29

Perform OLS regression on data columns:

In [11]:

```
%%bash
r --no-save
print("Hi I'm printing this in R.")
q()
```

```
R version 3.0.2 (2013-09-25) -- "Frisbee Sailing"
Copyright (C) 2013 The R Foundation for Statistical Computing
Platform: x86_64-apple-darwin10.8.0 (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

  Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
```

```
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

> print("Hi I'm printing this in R.")
[1] "Hi I'm printing this in R."
> q()
```

In [12]:

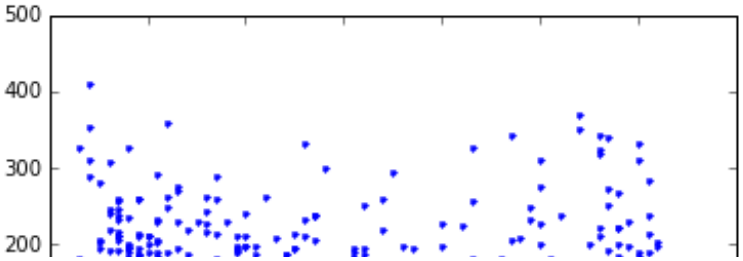
```
import statsmodels.formula.api as sm
Y = k['mean']
X = k['stable_lights']
result = sm.ols(formula="Y ~ X", data=k).fit()
print (result.summary())
intercept = result.params[0]
slope = result.params[1]
```

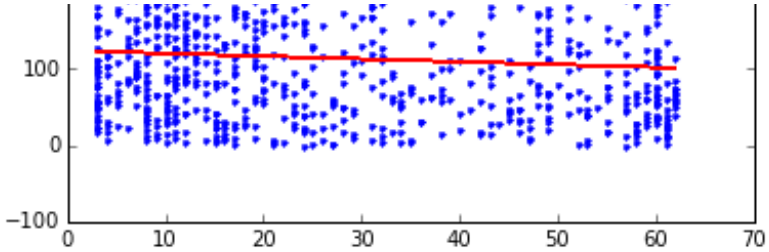
```
                            OLS Regression Results
==============================================================================
Dep. Variable:                      Y   R-squared:                       0.008
Model:                            OLS   Adj. R-squared:                  0.007
Method:                 Least Squares   F-statistic:                     5.873
Date:                Thu, 26 Mar 2015   Prob (F-statistic):             0.0156
Time:                        11:20:54   Log-Likelihood:                -4286.4
No. Observations:                 740   AIC:                             8577.
Df Residuals:                     738   BIC:                             8586.
Df Model:                           1
==============================================================================
                 coef    std err          t      P>|t|      [95.0% Conf. Int.]
------------------------------------------------------------------------------
Intercept    124.0935      5.127     24.205      0.000       114.029   134.158
X             -0.3656      0.151     -2.424      0.016        -0.662    -0.069
==============================================================================
Omnibus:                       59.949   Durbin-Watson:                   1.496
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               72.985
Skew:                           0.764   Prob(JB):                     1.42e-16
Kurtosis:                       3.183   Cond. No.                         59.7
==============================================================================
```

Take a look at the random sample and regression line:

In [13]:

```
from matplotlib import pyplot as plt
plt.plot(X, Y, '.')                    # plots the points
plt.plot(X, slope*X + intercept, 'r')
plt.show()
```





Return coefficients to make predicted image layer in ee:

In [14]:

```
slope_image = ee.Image.constant(slope)
int_image = ee.Image.constant(intercept)

temp = image1.multiply(slope_image)
predicted_image = temp.add(int_image)
error_image = image2.subtract(predicted_image)
error_image = error_image.abs()
```

Display:

In [15]:

```
map = GoogleMapsWidget(lat=41.5, lng=-73, zoom=9) # lat, lng and zoom are
optional
display(map)
PALETTE = ['DC143C','EA728A','F8DOD8','EA728a','DC143C']
map.addLayer(image=image2, name='Measured',vis_params={'min':0, 'max':100})
map.addLayer(image=predicted_image, name='Predicted',vis_params={'min':0, 'max'
:200})
map.addLayer(image=error_image, name='Error',vis_params={'min':0, 'max':150},
visible=False)
```

In [122]:

```

```

In [31]:

```

```

In []:

```

```