```
#########################################################
# Step 1: Forecast SNF Demand
# Reads in and interpolates training features
# Trains a random forest regression between training features and mal prevalence
# A train-test split validates the model
# Predicts mal prevalence to 2021
#########################################################

from __future__ import division
import csv
from math import sqrt
from sys import exit
import pickle
import matplotlib.pyplot as plt
import numpy as np
import matplotlib.mlab as mlab
#from mpl_toolkits.basemap import Basemap
import os
from scipy.stats import norm, mode
import matplotlib as mpl
from matplotlib.patches import Polygon
import random
import matplotlib.cm as cm
import matplotlib.colors as colors
from PIL import Image
from osgeo import gdal
#import ogr
from IPython import embed
import shapefile
#from shapely.geometry import shape, Point
import matplotlib.patches as patches
from math import sin, cos, sqrt, atan2, radians, pi, degrees
from geopy.geocoders import Nominatim
geolocator = Nominatim()
import geopy.distance
from scipy import ndimage
from scipy.signal import convolve2d
from sklearn import linear_model
from scipy.interpolate import RectSphereBivariateSpline
from scipy.interpolate import griddata
from sklearn.ensemble import RandomForestRegressor
from scipy.interpolate import RegularGridInterpolator
import sklearn
#import googlemaps

#################################################
# Functions
#################################################
def Variance(x):
        '''function to compute the variance (std dev squared)'''
        xAvg=np.mean(x)
        xOut=0.
        for k in range(len(x)):
                xOut=xOut+(x[k]-xAvg)**2
        xOut=xOut/(k+1)
        return xOut

def SumOfSquares(x):
        '''function to compute the sum of squares'''
        xOut=0.
        for k in range(len(x)):
                xOut=xOut+x[k]**2
        return xOut

def corr(x,y):
```

```
        ''' function to find the correlation of two arrays'''
        xAvg=np.mean(x)
        Avgy=np.mean(y)
        rxy=0.
        n=min(len(x),len(y))
        for k in range(n):
                rxy=rxy+(x[k]-xAvg)*(y[k]-Avgy)
        rxy=rxy/(k+1)
        stdDevx=np.std(x)
        stdDevy=np.std(y)
        rxy=rxy/(stdDevx*stdDevy)
        return rxy

def scale(var):
        varScaled=(var-np.amin(var))/(np.amax(var)-np.amin(var))
        return varScaled

from itertools import compress
class MaskableList(list):
        def __getitem__(self, index):
                try: return super(MaskableList, self).__getitem__(index)
                except TypeError: return MaskableList(compress(self, index))

def checkIfInPolygon(lon, lat,polygon):
        point = Point(lon, lat)
        return polygon.contains(point)

def findGridDataLays(shapePoints,gridMid):
        '''Convert Vector To Raster'''
        griddedHits=np.zeros(shape=(len(gridMid[:,0]),len(gridMid[0,:])))
        stepsize=np.zeros(shape=(len(shapePoints)))
        for i in range(len(shapePoints)-1):
                x1,y1=shapePoints[i]
                x2,y2=shapePoints[i+1]
                if geopy.distance.distance([y1,x1],[y2,x2]).km<2:
                        ##### Find Grid Cell of Midpoint #####
                        linePoints=np.array([(x1,y1),((x1+x2)/2.,(y1+y2)/2.),(x2,y2)])
                else:
                        dist=geopy.distance.distance([y1,x1],[y2,x2]).km+1e-6
                        if dist>50:
                                continue
                        if x2<x1:
                                step=abs(x2-x1)/(2*dist)
                                if step==0:
                                        step==0.004
                                x=np.arange(x2,x1,step)
                        else:
                                if x2==x1:
                                        x2=x1+1e-4
                                step=abs(x2-x1)/(2*dist)
                                if step==0:
                                        step==0.004
                                x=np.arange(x1,x2,step)
                        stepsize[i]=abs(x2-x1)/(2*dist)
                        slope,bInt=np.polyfit([x1,x2],[y1,y2],1)
                        yfit=slope*np.array(x)+bInt
                        linePoints=np.zeros(shape=(len(yfit),2))
                        for j in range(len(yfit)):
                                linePoints[j,:]=x[j],yfit[j]
                for j in range(len(linePoints)):
                        midPoint=linePoints[j]
                        yClosestL=min(gridMid[0,:,1], key=lambda i:abs(i-midPoint[1]))
                        xClosestL=min(gridMid[:,0,0], key=lambda i:abs(i-midPoint[0]))
                        xClosestGrid=np.where(gridMid[:,0,0]==xClosestL)[0][0]
                        yClosestGrid=np.where(gridMid[0,:,1]==yClosestL)[0][0]
```

```
                griddedHits[xClosestGrid,yClosestGrid]=1
        return griddedHits # lon lat


def findGridAndDistance(shapePoints,gridMid):
        ''' Give it a list of points in a road, will return pixel of center of each segment'''
        midpointHits=np.zeros(shape=(len(shapePoints)-1,2))
        dist=np.zeros(shape=(len(shapePoints)-1))
        for i in range(len(shapePoints)-1):
                x1,y1=shapePoints[i]
                x2,y2=shapePoints[i+1]
                ##### Find Grid Cell of Midpoint #####
                midPoint=(x1+x2)/2.,(y1+y2)/2.

                yClosestL=min(gridMid[0,:,1], key=lambda i:abs(i-midPoint[1]))
                xClosestL=min(gridMid[:,0,0], key=lambda i:abs(i-midPoint[0]))

                xClosestGrid=np.where(gridMid[:,0,0]==xClosestL)[0][0]
                yClosestGrid=np.where(gridMid[0,:,1]==yClosestL)[0][0]
                midpointHits[i]=xClosestGrid,yClosestGrid

                ### Convert radians to meters ###
                R = 6373.0 # Earth's radius
                coord1=x1,y1
                coord2=x2,y2
                dist[i]=geopy.distance.distance(coord1,coord2).km

        return midpointHits,dist

def createMidpointGrid(grid,pixelsize):
        gridMid=np.zeros(shape=(grid.shape))
        for ilon in range(len(grid[:,0,0])):
                gridMid[ilon,:,0]=np.mean([grid[ilon,0,0],grid[ilon,0,0]+pixelsize])
        for ilat in range(len(grid[0,:,1])):
                gridMid[:,ilat,1]=np.mean([grid[:,ilat,1],grid[:,ilat,1]+pixelsize])
        return gridMid

def findNearest(cityLonLat,gridMid,imageMask1):
        '''Give it:
        1. one dimentional lon lat data set
        2. grid of the mid points of cells (aquire from createMidpointGrid)
        3. mask for the grid in lon lat
        Will return:
        1. an array of the closest distance to one of the original lat lons (cityLonLat)
        2. which index of cityLonLat is closest
        '''
        lenLon,lenLat=len(gridMid[:,0,0]),len(gridMid[0,:,0])
        closestDist=10000*np.ones(shape=(lenLon,lenLat))
        iclosestDist=np.zeros(shape=(lenLon,lenLat),dtype=int)
        R = 6373.0 #earth's radius
        for ilon in range(lenLon):
                for ilat in range(lenLat):
                        if imageMask1[ilon,ilat]==True:
                                continue
                        distances=np.sqrt((gridMid[ilon,ilat,0]-cityLonLat[:,0])**2+(gridMid[ilon,ilat,1]
                        leastDist=np.amin(distances)
                        iclosestDist[ilon,ilat]=np.where(distances==leastDist)[0][0]
                        closestDist[ilon,ilat]=geopy.distance.distance([gridMid[ilon,ilat,1],gridMid[ilon
                print np.round(100*ilon/float(lenLon),2),'%'

        closestDistM=np.ma.masked_array(closestDist,imageMask1)
        iclosestDistM=np.ma.masked_array(iclosestDist,imageMask1)
        return closestDistM,iclosestDistM

def findNearestThree(cityLonLat,gridMid,imageMask1):
```

```
        lenLon,lenLat=len(gridMid[:,0,0]),len(gridMid[0,:,0])
        closestDist=10000*np.ones(shape=(lenLon,lenLat))
        closestDistDeg=100*np.ones(shape=(lenLon,lenLat,3))
        iclosestDist=np.zeros(shape=(lenLon,lenLat,3))
        imageMask4=np.zeros(shape=(lenLon,lenLat,3))
        for i in range(3):
                imageMask4[:,:,i]=imageMask1
        R = 6373.0 #earth's radius
        for ilon in range(lenLon):
                for ilat in range(lenLat):
                        if imageMask1[ilon,ilat]==True:
                                continue
                        lon,lat=gridMid[ilon,0,0],gridMid[0,ilat,1]
                        for city in range(len(cityLonLat[:,0])):
                                clon,clat=cityLonLat[city,0],cityLonLat[city,1]
                                dist=sqrt((clon-lon)**2+(clat-lat)**2)
                                if dist<closestDistDeg[ilon,ilat,0]:
                                        closestDistDeg[ilon,ilat,2]=closestDistDeg[ilon,ilat,1]
                                        iclosestDist[ilon,ilat,2]=iclosestDist[ilon,ilat,1]
                                        closestDistDeg[ilon,ilat,1]=closestDistDeg[ilon,ilat,0]
                                        iclosestDist[ilon,ilat,1]=iclosestDist[ilon,ilat,0]
                                        closestDistDeg[ilon,ilat,0]=dist
                                        iclosestDist[ilon,ilat,0]=city
                                        clonBest,clatBest=clon,clat
                                elif dist<closestDistDeg[ilon,ilat,1]:
                                        closestDistDeg[ilon,ilat,2]=closestDistDeg[ilon,ilat,1]
                                        iclosestDist[ilon,ilat,2]=iclosestDist[ilon,ilat,1]
                                        closestDistDeg[ilon,ilat,1]=dist
                                        iclosestDist[ilon,ilat,1]=dist
                                elif dist<closestDistDeg[ilon,ilat,2]:
                                        closestDistDeg[ilon,ilat,2]=dist
                                        iclosestDist[ilon,ilat,2]=dist

                        coord1=[lat,lon]
                        coord2=[clatBest,clonBest]
                        closestDist[ilon,ilat]=geopy.distance.distance(coord1,coord2).km
                print np.round(100*ilon/float(len(gridMid[:,0,0])),2),'%'
        closestDistM=np.ma.masked_array(closestDist,imageMask1)
        closestDistDegM=np.ma.masked_array(closestDistDeg,imageMask4)
        iclosestDistM=np.ma.masked_array(iclosestDist,imageMask4)
        closestDistM=np.swapaxes(closestDistM,0,1)
        closestDistDegM=np.swapaxes(closestDistDegM,0,1)
        iclosestDistM=np.swapaxes(iclosestDistM,0,1)
        return closestDistM,iclosestDistM
DTYPEf = np.float64
#ctypedef np.float64_t DTYPEf_t

DTYPEi = np.int32
#ctypedef np.int32_t DTYPEi_t

#@cython.boundscheck(False) # turn of bounds-checking for entire function
#@cython.wraparound(False) # turn of bounds-checking for entire function
def replace_nans(array, max_iter, tol,kernel_size=1,method='localmean'):
        """Replace NaN elements in an array using an iterative image inpainting algorithm.
The algorithm is the following:
1) For each element in the input array, replace it by a weighted average
of the neighbouring elements which are not NaN themselves. The weights depends
of the method type. If ``method=localmean`` weight are equal to 1/( (2*kernel_size+1)**2 -1 )
2) Several iterations are needed if there are adjacent NaN elements.
If this is the case, information is "spread" from the edges of the missing
regions iteratively, until the variation is below a certain threshold.
Parameters
----------
array : 2d np.ndarray
an array containing NaN elements that have to be replaced
```

```
max_iter : int
the number of iterations
kernel_size : int
the size of the kernel, default is 1
method : str
the method used to replace invalid values. Valid options are
`localmean`, 'idw'.
Returns
-------
filled : 2d np.ndarray
a copy of the input array, where NaN elements have been replaced.
"""

        filled = np.empty( [array.shape[0], array.shape[1]], dtype=DTYPEf)
        kernel = np.empty( (2*kernel_size+1, 2*kernel_size+1), dtype=DTYPEf )

        # indices where array is NaN
        inans, jnans = np.nonzero( np.isnan(array) )

        # number of NaN elements
        n_nans = len(inans)

        # arrays which contain replaced values to check for convergence
        replaced_new = np.zeros( n_nans, dtype=DTYPEf)
        replaced_old = np.zeros( n_nans, dtype=DTYPEf)

        # depending on kernel type, fill kernel array
        if method == 'localmean':

                print 'kernel_size', kernel_size
                for i in range(2*kernel_size+1):
                        for j in range(2*kernel_size+1):
                                kernel[i,j] = 1
                print kernel, 'kernel'

        elif method == 'idw':
                kernel = np.array([[0, 0.5, 0.5, 0.5,0],
                                   [0.5,0.75,0.75,0.75,0.5],
                                   [0.5,0.75,1,0.75,0.5],
                                   [0.5,0.75,0.75,0.5,1],
                                   [0, 0.5, 0.5 ,0.5 ,0]])
                print kernel, 'kernel'
        else:
                raise ValueError( 'method not valid. Should be one of `localmean`.')

        # fill new array with input elements
        for i in range(array.shape[0]):
                for j in range(array.shape[1]):
                        filled[i,j] = array[i,j]

        # make several passes
        # until we reach convergence
        for it in range(max_iter):
                print 'iteration', it
                # for each NaN element
                for k in range(n_nans):
                        i = inans[k]
                        j = jnans[k]

                        # initialize to zero
                        filled[i,j] = 0.0
                        n = 0

                        # loop over the kernel
                        for I in range(2*kernel_size+1):
```

```
                                for J in range(2*kernel_size+1):

                                        # if we are not out of the boundaries
                                        if i+I-kernel_size < array.shape[0] and i+I-kernel_size >= 0:
                                                if j+J-kernel_size < array.shape[1] and j+J-kernel_size >=

                                                        # if the neighbour element is not NaN itself.
                                                        if filled[i+I-kernel_size, j+J-kernel_size] == fi

                                                                # do not sum itself
                                                                if I-kernel_size != 0 and J-kernel_size !=

                                                                        # convolve kernel with original a
                                                                        filled[i,j] = filled[i,j] + fille
                                                                        n = n + 1*kernel[I,J]
                                                                        #print n

                                # divide value by effective number of added elements
                                if n != 0:
                                        filled[i,j] = filled[i,j] / n
                                        replaced_new[k] = filled[i,j]
                                else:
                                        filled[i,j] = np.nan

                        # check if mean square difference between values of replaced
                        #elements is below a certain tolerance
                        print 'tolerance', np.mean( (replaced_new-replaced_old)**2 )
                        if np.mean( (replaced_new-replaced_old)**2 ) < tol:
                                break
                        else:
                                for l in range(n_nans):
                                        replaced_old[l] = replaced_new[l]

        return filled


def sincinterp(image, x,  y, kernel_size=3 ):
        """Re-sample an image at intermediate positions between pixels.
        This function uses a cardinal interpolation formula which limits the loss of information in the re
        The new image :math:`im^+` at fractional locations :math:`x` and :math:`y` is computed as:
        .. math::
        im^+(x,y) = \sum_{i=-\mathtt{kernel\_size}}^{i=\mathtt{kernel\_size}} \sum_{j=-\mathtt{kernel\_si
        Parameters
        ----------
        image : np.ndarray, dtype np.int32 the image array.
        x : two dimensions np.ndarray of floats an array containing fractional pixel row positions at whi
        y : two dimensions np.ndarray of floats an array containing fractional pixel column positions at
        kernel_size : int
        interpolation is performed over a ``(2*kernel_size+1)*(2*kernel_size+1)`` submatrix in the neighb
        Returns
        -------
        im : np.ndarray, dtype np.float64
        the interpolated value of ``image`` at the points specified
        by ``x`` and ``y``
        """

        # the output array
        r = np.zeros( [x.shape[0], x.shape[1]], dtype=DTYPEf)

        # fast pi
        pi = 3.1419

        # for each point of the output array
        for I in range(x.shape[0]):
                for J in range(x.shape[1]):
```

```
                        #loop over all neighbouring grid points
                        for i in range( int(x[I,J])-kernel_size, int(x[I,J])+kernel_size+1 ):
                                for j in range( int(y[I,J])-kernel_size, int(y[I,J])+kernel_size+1 ):
                                        # check that we are in the boundaries
                                        if i >= 0 and i <= image.shape[0] and j >= 0 and j <= image.shape[
                                                if (i-x[I,J]) == 0.0 and (j-y[I,J]) == 0.0:
                                                        r[I,J] = r[I,J] + image[i,j]
                                                elif (i-x[I,J]) == 0.0:
                                                        r[I,J] = r[I,J] + image[i,j] * np.sin( pi*(j-y[I,J
                                                elif (j-y[I,J]) == 0.0:
                                                        r[I,J] = r[I,J] + image[i,j] * np.sin( pi*(i-x[I,J
                                                else:
                                                        r[I,J] = r[I,J] + image[i,j] * np.sin( pi*(i-x[I,J
        return r

def moving_average_2d(data, window):
        """Moving average on two-dimensional data.
        """
        # Makes sure that the window function is normalized.
        window /= window.sum()
        # Makes sure data array is a numpy array or masked array.
        if type(data).__name__ not in ['ndarray', 'MaskedArray']:
                data = numpy.asarray(data)

        # The output array has the same dimensions as the input data
        # (mode='same') and symmetrical boundary conditions are assumed
        # (boundary='symm').
        return convolve2d(data, window, mode='same', boundary='symm')

def marketPotentials(pop,popCoord,gridMid,imageMask2):
        lenLat=len(gridMid[:,0,0])
        lenLon=len(gridMid[0,:,0])
        MP=np.zeros(shape=(gridMid[:,:,0].shape))
        for ilat in range(lenLat):
                print np.round(100*ilat/float(lenLat),2),'%'
                for ilon in range(lenLon):
                        if imageMask2[ilat,ilon]==True:
                                continue
                        dists=np.sqrt((gridMid[ilat,ilon,0]-popCoord[:,0])**2+(gridMid[ilat,ilon,1]-popCo
                        MP[ilat,ilon]=np.sum(pop/(dists**1.2))
        return MP

def make_cmap(colors, position=None, bit=False):
    '''
    make_cmap takes a list of tuples which contain RGB values. The RGB
    values may either be in 8-bit [0 to 255] (in which bit must be set to
    True when called) or arithmetic [0 to 1] (default). make_cmap returns
    a cmap with equally spaced colors.
    Arrange your tuples so that the first color is the lowest value for the
    colorbar and the last is the highest.
    position contains values from 0 to 1 to dictate the location of each color.
    '''
    import matplotlib as mpl
    import numpy as np
    bit_rgb = np.linspace(0,1,256)
    if position == None:
        position = np.linspace(0,1,len(colors))
    else:
        if len(position) != len(colors):
            sys.exit("position length must be the same as colors")
        elif position[0] != 0 or position[-1] != 1:
            sys.exit("position must start with 0 and end with 1")
    if bit:
        for i in range(len(colors)):
```

```
            colors[i] = (bit_rgb[colors[i][0]],
                         bit_rgb[colors[i][1]],
                         bit_rgb[colors[i][2]])
    cdict = {'red':[], 'green':[], 'blue':[]}
    for pos, color in zip(position, colors):
        cdict['red'].append((pos, color[0], color[0]))
        cdict['green'].append((pos, color[1], color[1]))
        cdict['blue'].append((pos, color[2], color[2]))

    cmap = mpl.colors.LinearSegmentedColormap('my_colormap',cdict,256)
    return cmap
#, (50,205,50)(173,255,47) ,
#colors = [(0,128,0) , (50,205,50) , (173,255,47) , (255,255,0) , (255,179,25) , (255,69,0) , (139,0,0)]
colors = [(255,255,255) , (50,205,50) ,  (255,255,0) ,(255,213,0) , (255,179,25) ,  (255,69,0) , (255,0,0
my_cmap = make_cmap(colors,bit=True)

################################################
try:
        wddata='/Users/lilllianpetersen/iiasa/data/'
        wdfigs='/Users/lilllianpetersen/iiasa/figs/'
        wdvars='/Users/lilllianpetersen/iiasa/saved_vars/'
except:
        wddata='C:/Users/garyk/Documents/python_code/riskAssessmentFromPovertyEstimations/data/'
        wdfigs='C:/Users/garyk/Documents/python_code/riskAssessmentFromPovertyEstimations/figs/'
        wdvars='C:/Users/garyk/Documents/python_code/riskAssessmentFromPovertyEstimations/vars/'

MakePlots=False

latsubsaharan=np.load(wdvars+'latsubsaharan.npy')
lonsubsaharan=np.load(wdvars+'lonsubsaharan.npy')
africaMask1=np.load(wdvars+'africaMasksubsaharan.npy')

nyearsT=16
nyears=22

################################################
# Gridded Malnutrition
################################################
print 'Malnutrition'
try:
        mal=np.load(wdvars+'malnutrition_prevalence_2000-2015.npy')
        latm=latsubsaharan
        lonm=lonsubsaharan
        pixelsize=0.04166666666650246

        gridm = np.zeros(shape=(len(latm),len(lonm),2))
        for x in range(len(latm)):
                gridm[x,:,0]=latm[x]
        for x in range(len(lonm)):
                gridm[:,x,1]=lonm[x]

        gridMid=createMidpointGrid(gridm,pixelsize) # lon lat

        imageMask2=np.load(wdvars+'imageMask2.npy')
        imageMask2_1year=imageMask2[0]
        imageMask2=np.zeros(shape=(nyears,len(latm),len(lonm)))
        for y in range(nyears):
                imageMask2[y]=imageMask2_1year

        imageMask3=np.zeros(shape=(nyears,10,imageMask2.shape[1],imageMask2.shape[2]),dtype=bool)
        for y in range(nyears):
                for i in range(10):
                        imageMask3[y,i,:,:]=imageMask2[y] # imageMask2=(lat,lon,3)

except:
```

```
print 'try command failed: retrieving malnutrition'

mal=np.zeros(shape=(16,len(latsubsaharan),len(lonsubsaharan)))
imageMask2=np.zeros(shape=(mal.shape),dtype=bool)
imageMask3=np.zeros(shape=(nyearsT,10,imageMask2.shape[1],imageMask2.shape[2]),dtype=bool)
nationsM=np.load(wdvars+'nations')
nationsMask=np.ma.getmask(nationsM)
nationsMask[nationsM==818]=True

year=1999
for y in range(16):
        year+=1
        ds=gdal.Open(wddata+'malnutrition/IHME_AFRICA_CGF_2000_2015_WASTING_MEAN_'+str(year)+'_PRE
        width1 = ds.RasterXSize
        height1 = ds.RasterYSize
        gt = ds.GetGeoTransform()
        minx = gt[0]
        miny = gt[3] + width1*gt[4] + height1*gt[5]
        maxx = gt[0] + width1*gt[1] + height1*gt[2]
        maxy = gt[3]
        pixelsize=abs(gt[-1])

        # lat and lon
        latm=np.ones(shape=(height1))
        lonm=np.ones(shape=(width1))
        for w in range(width1):
                lonm[w]=minx+w*pixelsize
        for h in range(height1):
                latm[h]=miny+h*pixelsize
        latm=latm[::-1] # reverse the order

        maltmp=ds.ReadAsArray()
        maltmp[maltmp>1]=0.0308212 # The average of this weird pixel's neighbors

        ##### Scale to Africa #####
        maltmp=maltmp[latm<np.amax(latsubsaharan)+1e-2]
        latm=latm[latm<np.amax(latsubsaharan)+1e-2]
        maltmp=maltmp[latm>np.amin(latsubsaharan)-1e-2]
        latm=latm[latm>np.amin(latsubsaharan)-1e-2]

        maltmp=maltmp[:,lonm<np.amax(lonsubsaharan)+1e-2]
        lonm=lonm[lonm<np.amax(lonsubsaharan)+1e-2]
        maltmp=maltmp[:,lonm>np.amin(lonsubsaharan)-1e-2]
        lonm=lonm[lonm>np.amin(lonsubsaharan)-1e-2]
        ########## Mask ###########
        latm=latsubsaharan
        lonm=lonsubsaharan

        mal[y,:,:]=maltmp

        latm=latm[::-1]
        gridm = np.zeros(shape=(len(latm),len(lonm),2))
        for x in range(len(latm)):
                gridm[x,:,0]=latm[x]
        for x in range(len(lonm)):
                gridm[:,x,1]=lonm[x]
        latm=latm[::-1]

        gridMid=createMidpointGrid(gridm,pixelsize) # lon lat

        imageMask2[y,mal[y]<0]=1
        imageMask2[y,africaMask1==1]=1
        imageMask2[y,900:1300,1450:]=1
        imageMask2[y,nationsMask==1]=1
```

```
                for y in range(16):
                        for i in range(10):
                                imageMask3[y,i,:,:]=imageMask2[y] # imageMask2=(lat,lon,3)

                mal[y][mal[y]<0]=0
        mal=np.ma.masked_array(mal,imageMask2)

        mal.dump(wdvars+'malnutrition_prevalence_2000-2015y')
        imageMask2.dump(wdvars+'imageMask2')

if MakePlots:
        for y in range(nyearsT):
                plt.clf()
                plt.imshow(mal[y]*100,cmap=cm.jet,vmax=30)
                plt.title(str(year)+': % Population with Acute Malnutrition')
                plt.xticks([])
                plt.yticks([])
                plt.colorbar()
                plt.savefig(wdfigs+'malnutrition_'+str(year),dpi=500)

        plt.clf()
        plt.imshow(np.mean(mal,axis=0),cmap=cm.jet,vmax=0.3)
        plt.title('Average % Population with Severe Malnutrition')
        plt.xticks([])
        plt.yticks([])
        plt.colorbar()
        plt.savefig(wdfigs+'malnutrition_avg',dpi=700)
exit()

################################################
# Gridded Population
################################################
print 'Population'
try:
        pop5km1year=np.load(wdvars+'pop5km_2010.npy')

        pop5km=np.ma.zeros(shape=(nyears,len(latm),len(lonm)))
        for y in range(nyears):
                pop5km[y]=pop5km1year
except:
        print 'try command failed: retrieveing population'

        ds=gdal.Open(wddata+'population/gpw_v4_basic_demographic_characteristics_rev10_a000_004bt_2010_cnt
        width1 = ds.RasterXSize
        height1 = ds.RasterYSize
        gt = ds.GetGeoTransform()
        minx = gt[0]
        miny = gt[3] + width1*gt[4] + height1*gt[5]
        maxx = gt[0] + width1*gt[1] + height1*gt[2]
        maxy = gt[3]
        pixelsizep=abs(gt[-1])

        # lat and lon
        latp=np.ones(shape=(height1))
        lonp=np.ones(shape=(width1))
        for w in range(width1):
                lonp[w]=minx+w*pixelsizep
        for h in range(height1):
                latp[h]=miny+h*pixelsizep
        latp=latp[::-1] # reverse the order

        worldPop=ds.ReadAsArray()

        ##### Scale to Africa #####
        pop=worldPop[latp<np.amax(latm)+pixelsize]
```

```
        latp=latp[latp<np.amax(latm)+pixelsize]
        pop=pop[latp>np.amin(latm)]
        latp=latp[latp>np.amin(latm)]

        pop=pop[:,lonp<np.amax(lonm)+pixelsize]
        lonp=lonp[lonp<np.amax(lonm)+pixelsize]
        pop=pop[:,lonp>np.amin(lonm)]
        lonp=lonp[lonp>np.amin(lonm)]
        ############################

        pop[pop<0]=0

        plt.clf()
        plt.imshow(pop,cmap=cm.gist_ncar_r,vmax=2000)
        plt.title('gridded population')
        plt.colorbar()
        plt.savefig(wdfigs+'pop5km',dpi=900)

        gridp=np.zeros(shape=(len(lonp),len(latp),2))
        for x in range(len(lonp)):
                gridp[x,:,0]=lonp[x]
        for x in range(len(latp)):
                gridp[:,x,1]=latp[y]

        latl=np.radians(latp[::-1])+1.2
        lonl=np.radians(lonp)+1.2
        lutpop=RectSphereBivariateSpline(latl, lonl, pop)

        newLats,newLons=np.meshgrid(np.radians(latm[::-1])+1.2,np.radians(lonm)+1.2)
        pop1=lutpop.ev(newLats.ravel(),newLons.ravel()).reshape((len(lonm),len(latm))).T
        pop1[pop1<0]=0

        R=6371 #km
        latdiff1=abs(np.sin(np.radians(latm[1:]))-np.sin(np.radians(latm[:-1])))
        latdiff=np.zeros(shape=(len(latm)))
        latdiff[:len(latdiff1)]=latdiff1
        latdiff[-1]=latdiff[-2]

        londiff1=abs(np.radians(lonm[1:])-np.radians(lonm[:-1]))
        londiff=np.zeros(shape=(len(lonm)))
        londiff[:len(londiff1)]=londiff1
        londiff[-1]=londiff[-2]-(londiff[-3]-londiff[-2])

        area=np.zeros(shape=(pop1.shape))
        for ilon in range(len(londiff)):
                area[:,ilon]=(R**2)*latdiff*londiff[ilon]  #radians

        pop5km=pop1*area
        np.save(wdvars+'pop5km_2010',pop5km)

pop5km=np.ma.masked_array(pop5km,imageMask2)

malnumber=mal*pop5km[:nyearsT]
malnumber=np.ma.masked_array(malnumber,imageMask2[:nyearsT])

#if MakePlots:
fig = plt.figure(figsize=(9, 6))
plt.clf()
plt.imshow(pop5km[15]/100,cmap=cm.gist_ncar_r,vmax=20)
plt.title('2015 Population (100 per square 5km)')
plt.colorbar(ticks=[0,5,10,15,20])
plt.xticks([])
plt.yticks([])
plt.savefig(wdfigs+'pop5km',dpi=500)
```

```
plt.clf()
plt.imshow(malnumber[15],cmap=cm.nipy_spectral_r,vmax=500)
plt.title('malnutrition number')
plt.colorbar()
plt.savefig(wdfigs+'malnumber',dpi=900)

########################################
# Under 5 year old pop
########################################
print '5-year-old Population'
try:
        popUnder5 = np.load(wdvars+'age_structures/popUnder5.npy')
        popUnder5 = np.ma.masked_array(popUnder5,imageMask2)
except:
        print 'try command failed: retrieveing population under 5'
        popUnder5=np.zeros(shape=(nyears,len(latm),len(lonm)))
        sex=['M','F']

        y=-5
        for year in range(2000,2021,5):
                y+=5
                print y,year
                for g in range(2):
                        ds=gdal.Open(wddata+'age_structures/AFR_PPP_A0004_'+sex[g]+'_'+str(year)+'_adj_v5
                        width1 = ds.RasterXSize
                        height1 = ds.RasterYSize
                        gt = ds.GetGeoTransform()
                        minx = gt[0]
                        miny = gt[3] + width1*gt[4] + height1*gt[5]
                        maxx = gt[0] + width1*gt[1] + height1*gt[2]
                        maxy = gt[3]
                        pixelsizep=abs(gt[-1])

                        # lat and lon
                        latp=np.ones(shape=(height1))
                        lonp=np.ones(shape=(width1))
                        for w in range(width1):
                                lonp[w]=minx+w*pixelsizep
                        for h in range(height1):
                                latp[h]=miny+h*pixelsizep
                        latp=latp[::-1] # reverse the order

                        worldPop=ds.ReadAsArray()

                        ##### Scale to Africa #####
                        pop=worldPop[latp<np.amax(latm)+pixelsize]
                        latp=latp[latp<np.amax(latm)+pixelsize]
                        pop=pop[latp>np.amin(latm)]
                        latp=latp[latp>np.amin(latm)]

                        pop=pop[:,lonp<np.amax(lonm)+pixelsize]
                        lonp=lonp[lonp<np.amax(lonm)+pixelsize]
                        pop=pop[:,lonp>np.amin(lonm)]
                        lonp=lonp[lonp>np.amin(lonm)]
                        ###########################

                        pop[pop<0]=0

                        plt.clf()
                        plt.imshow(pop,cmap=cm.gist_ncar_r,vmax=2000)
                        plt.title('gridded population under 5')
                        plt.colorbar()
                        plt.savefig(wdfigs+'popUnder5',dpi=900)

                        ##### rect sphere bivariate spline #####
```

```
                        latl=np.radians(latp[::-1])+1.2
                        lonl=np.radians(lonp)+1.2
                        lutpop=RectSphereBivariateSpline(latl, lonl, pop)

                        newLats,newLons=np.meshgrid(np.radians(latm[::-1])+1.2,np.radians(lonm)+1.2)
                        pop1=lutpop.ev(newLats.ravel(),newLons.ravel()).reshape((len(lonm),len(latm))).T
                        pop1[pop1<0]=0
                        ######################################

                        ##### scale up the population #####
                        R=6371 #km
                        latdiff1=abs(np.sin(np.radians(latm[1:]))-np.sin(np.radians(latm[:-1])))
                        latdiff=np.zeros(shape=(len(latm)))
                        latdiff[:len(latdiff1)]=latdiff1
                        latdiff[-1]=latdiff[-2]

                        londiff1=abs(np.radians(lonm[1:])-np.radians(lonm[:-1]))
                        londiff=np.zeros(shape=(len(lonm)))
                        londiff[:len(londiff1)]=londiff1
                        londiff[-1]=londiff[-2]-(londiff[-3]-londiff[-2])

                        area=np.zeros(shape=(pop1.shape))
                        for ilon in range(len(londiff)):
                                area[:,ilon]=(R**2)*latdiff*londiff[ilon]  #radians

                        vars()['pop5_'+sex[g]+'_'+str(year)]=pop1*area
                        ###################################

                        np.save(wdvars+'age_structures/pop5_'+str(year),vars()['pop5_'+sex[g]+'_'+str(yea

                        if y==0:
                                popUnder5[y:y+3,:,:] += vars()['pop5_'+sex[g]+'_'+str(year)]
                        elif y==20:
                                popUnder5[y-2:y+2,:,:] += vars()['pop5_'+sex[g]+'_'+str(year)]
                        else:
                                popUnder5[y-2:y+3,:,:] += vars()['pop5_'+sex[g]+'_'+str(year)]
        np.save(wdvars+'age_structures/popUnder5.npy',popUnder5)
        popUnder5 = np.ma.masked_array(popUnder5,imageMask2)
exit()

#if MakePlots:
fig = plt.figure(figsize=(9, 6))
for y in range(nyears):
        plt.clf()
        plt.imshow(popUnder5[y]/100,cmap=cm.gist_ncar_r,vmax=5)
        plt.title('2015 Population (100 per square 5km)')
        plt.colorbar() #ticks=[0,5,10,15,20])
        plt.xticks([])
        plt.yticks([])
        plt.savefig(wdfigs+'popUnder5'+str(y),dpi=500)

plt.clf()
plt.imshow(malnumber[15],cmap=cm.nipy_spectral_r,vmax=500)
plt.title('malnutrition number')
plt.colorbar()
plt.savefig(wdfigs+'malnumber',dpi=900)

#####################################
# Countries
#####################################
print 'Nations'
try:
        nations=np.load(wdvars+'nations_unmasked')
        nationsM=np.load(wdvars+'nations')
except:
```

```
        print 'try command failed: retrieveing nations'

        ds=gdal.Open(wddata+'boundaries/gpw-v4-national-identifier-grid-rev10_30_sec_tif/gpw_v4_national_:
        width = ds.RasterXSize
        height = ds.RasterYSize
        gt = ds.GetGeoTransform()
        minx = gt[0]
        miny = gt[3] + width*gt[4] + height*gt[5]
        maxx = gt[0] + width*gt[1] + height*gt[2]
        maxy = gt[3]
        pixelsizeNI=abs(gt[-1])

        latc=np.ones(shape=(height))
        lonc=np.ones(shape=(width))
        for w in range(width):
                lonc[w]=minx+w*pixelsizeNI
        for h in range(height):
                latc[h]=miny+h*pixelsizeNI

        latc=latc[::-1]

        nations=ds.ReadAsArray()
        ##### Scale to Africa #####
        nations=nations[latc<np.amax(latm)+pixelsize]
        latc=latc[latc<np.amax(latm)+pixelsize]
        nations=nations[latc>np.amin(latm)]
        latc=latc[latc>np.amin(latm)]

        nations=nations[:,lonc<np.amax(lonm)+pixelsize]
        lonc=lonc[lonc<np.amax(lonm)+pixelsize]
        nations=nations[:,lonc>np.amin(lonm)]
        lonc=lonc[lonc>np.amin(lonm)]

        for ilat in range(len(nations[:,0])):
                print 100*np.round(ilat/float(len(nations[:,0])),3),'%'
                for ilon in range(len(nations[0,:])):
                        if nations[ilat,ilon]==32767:
                                nations[ilat,ilon]=nations[ilat,ilon-1]

        for ilat in range(len(nations[:,0])):
                print 100*np.round(ilat/float(len(nations[:,0])),3),'%'
                for ilon in range(len(nations[0,:]))[::-1]:
                        if nations[ilat,ilon]==32767:
                                nations[ilat,ilon]=nations[ilat,ilon+1]

        ##### Scale to 5km #####
        latl=np.radians(latc[::-1])+1.2
        lonl=np.radians(lonc)+1.2
        lut=RectSphereBivariateSpline(latl, lonl, nations)

        newLats,newLons=np.meshgrid(np.radians(latm[::-1])+1.2,np.radians(lonm)+1.2)
        nations=lut.ev(newLats.ravel(),newLons.ravel()).reshape((len(lonm),len(latm))).T
        nations=np.round(nations,0)
        nations.dump(wdvars+'nations_unmasked')

        nationsM=np.ma.masked_array(nations,imageMask2[0])
        nationsM=np.round(nationsM,0)
        nationsMask=np.ma.getmask(nationsM)
        nationsMask[nationsM==818]=True #Egypt
        nationsMask[nationsM==32767]=True #Bad
        nationsM=np.ma.masked_array(nationsM,nationsMask)

        nationsM.dump(wdvars+'nations')

plt.clf()
```

```
plt.imshow(nationsM,cmap=cm.nipy_spectral)
plt.yticks([])
plt.xticks([])
plt.title('Countries')
plt.colorbar()
plt.savefig(wdfigs +'nations_masked',dpi=700)

plt.clf()
plt.imshow(nations,cmap=cm.nipy_spectral,vmin=24,vmax=895)
#plt.yticks([])
#plt.xticks([])
plt.title('Countries')
plt.colorbar()
plt.savefig(wdfigs +'nations',dpi=700)

f=open(wddata+'boundaries/countries_countryCodes.csv')
code=np.zeros(shape=(247),dtype=int)
countryNames=[]
i=-1
for line in f:
        i+=1
        tmp=line.split(',')
        code[i]=int(tmp[3])
        countryNames.append(tmp[0])

f=open(wddata+'boundaries/africanCountries.csv','r')
africanCountries=[]
for line in f:
        africanCountries.append(line[:-1])

#indexing the country codes
countryToIndex={}
indexToCountry={}
indexedcodes=np.zeros(shape=len(africanCountries))
for i in range(len(africanCountries)):
        j=np.where(africanCountries[i]==np.array(countryNames))[0][0]
        indexedcodes[i]=code[j]
        countryToIndex[africanCountries[i]]=indexedcodes[i]
        indexToCountry[indexedcodes[i]]=africanCountries[i]
        if len(np.where(nationsM==indexedcodes[i])[0])==0:
                print africanCountries[i],'has a bad mask'

countryToi={}
iToCountry={}
for i in range(len(indexedcodes)):
        index=indexedcodes[i]
        country=indexToCountry[index]
        countryToi[country]=i
        iToCountry[i]=country

#####################################
# Sum each country
#####################################
try:
        malCountry = np.load(wdvars+'malCountry_2000-2015.npy')
except:
        print 'try command failed: calculating malCountry'
        malCountry=np.zeros(shape=(len(africanCountries),nyearsT))
        j=-1
        for i in indexedcodes:
                j+=1
                for y in range(nyearsT):
                        popWhole=np.sum(population[y,nations==i])
                        poptmp=np.ma.masked_array(population[y],nations!=i)
                        maltmp=np.ma.masked_array(mal[y],nations!=i)
```

```
                        malnumtmp=maltmp*poptmp
                        malCountry[j,y]=np.sum(malnumtmp)/popWhole
        malCountry=malCountry*100

        malCountryGrid=np.zeros(shape=(nyearsT,len(latm),len(lonm)))
        j=-1
        for i in indexedcodes:
                j+=1
                for y in range(nyearsT):
                        malCountryGrid[y,nations==i]=malCountry[j,y]

        malCountryGrid=np.ma.masked_array(malCountryGrid,imageMask2[:nyearsT])
        np.save(wdvars+'malCountry_2000-2015',malCountry)

if MakePlots:
        year=1998
        for y in range(nyearsT):
                year+=1
                plt.clf()
                plt.imshow(malCountryGrid[y],cmap=cm.jet,vmax=20)
                plt.colorbar()
                plt.yticks([])
                plt.xticks([])
                plt.title(str(year)+': Average Malnutrition Prevalence')
                plt.savefig(wdfigs+'malCountryGrid_'+str(year)+'.png',dpi=700)

lats=np.zeros(shape=(43,16))
f = open(wddata+'boundaries/countryLats.csv')
for line in f:
        line=line[:-1]
        tmp=line.split(',')
        if np.amax(tmp[3]==np.array(africanCountries))==0:
                continue
        i=countryToi[tmp[3]]
        lats[i,:]=float(tmp[1])
latsS=scale(lats)

#####################################
# SAM vs MAM from UNICEF numbers
#####################################
print 'SAM and MAM from UNICEF'

#### MAM ####
MAMcaseload=np.zeros(shape=(43,nyearsT))

with open(wddata+'unicef_caseload/UNICEF_Wasting_2018_May/Trend-Table1.csv', 'rb') as csvfile:
        k=0
        reader = csv.reader(csvfile, delimiter=',', quotechar='"')
        for tmp in reader:
                k+=1

                country=tmp[2]
                if np.amax(country==np.array(africanCountries))==0:
                        continue
                year=int(tmp[5])
                y=year-2000
                if year<2000 or year>2015:
                        continue
                i=countryToi[country]
                MAMcaseload[i,y]=tmp[9]

MAMcaseload=np.ma.masked_array(MAMcaseload,MAMcaseload==0)
MAMmask=np.ma.getmask(MAMcaseload)
malCountryM=np.ma.masked_array(malCountry,MAMmask)
CorrM=corr(np.ma.compressed(MAMcaseload),np.ma.compressed(malCountryM))
```

```
#### SAM ####
SAMcaseload=np.zeros(shape=(43,nyearsT))

with open(wddata+'unicef_caseload/UNICEF_Severe_Wasting_2018_May/Trend-Table1.csv', 'rb') as csvfile:
        k=0
        reader = csv.reader(csvfile, delimiter=',', quotechar='"')
        for tmp in reader:
                k+=1

                country=tmp[2]
                if np.amax(country==np.array(africanCountries))==0:
                        continue
                year=int(tmp[5])
                y=year-2000
                if year<2000 or year>2015:
                        continue
                i=countryToi[country]
                SAMcaseload[i,y]=tmp[9]

SAMcaseload=np.ma.masked_array(SAMcaseload,SAMcaseload==0)
SAMmask=np.ma.getmask(SAMcaseload)
malCountryM=np.ma.masked_array(malCountry,SAMmask)
CorrS=corr(np.ma.compressed(SAMcaseload),np.ma.compressed(malCountryM))

#### MAM + SAM ####
ydata=np.ma.compressed(MAMcaseload)
x=np.ma.compressed(np.ma.masked_array(malCountry,MAMmask))
colors=np.ma.compressed(np.ma.masked_array(lats,MAMmask))

slope,b=np.polyfit(x,ydata,1)
yfit=slope*x+b
stdDev=np.std(ydata-x)
error = abs(yfit - ydata)

fig = plt.figure(figsize=(9, 5))
plt.clf()
plt.scatter(x,ydata,c=colors,marker='*',cmap=cm.jet)
plt.colorbar()
plt.plot(x,yfit,'-g')
plt.plot(x,yfit-stdDev,'--g',linewidth=0.5)
plt.plot(x,yfit+stdDev,'--g',linewidth=0.5)
plt.title('UNICEF MAM + SAM vs Paper, Corr = '+str(round(CorrM,2))+', Avg Error = '+str(round(np.mean(erro
plt.ylabel('MAM + SAM %')
plt.xlabel('Interpolated %')
plt.savefig(wdfigs+'AMvsInterpolated.pdf')

#### SAM ####
ydata=np.ma.compressed(SAMcaseload)
x=np.ma.compressed(np.ma.masked_array(malCountry,SAMmask))
colors=np.ma.compressed(np.ma.masked_array(lats,SAMmask))

SAMa,SAMb,SAMc=np.polyfit(x,ydata,2)
yfit=SAMa*x**2+SAMb*x+SAMc
SAMstdDev=np.std(ydata-x)
s=np.argsort(yfit)
yfit2=np.array(yfit[s])
x2=x[s]
error = abs(yfit-ydata)

plt.clf()
plt.scatter(x,ydata,c=colors,marker='*',cmap=cm.jet)
plt.colorbar()
plt.plot(x2,yfit2,'-g')
plt.plot(x2,yfit2-SAMstdDev,'--g',linewidth=0.5)
```

```
plt.plot(x2,yfit2+SAMstdDev,'--g',linewidth=0.5)
plt.title('UNICEF SAM vs Paper, Corr = '+str(round(CorrS,2))+', Avg Error = '+str(round(np.mean(error),1))
plt.ylabel('SAM %')
plt.xlabel('Paper %')
plt.savefig(wdfigs+'SAMvsInterpolated.pdf')

#### MAM ####
ydata=np.ma.compressed(MAMcaseload-SAMcaseload)
Mmask=MAMmask+SAMmask
x=np.ma.compressed(np.ma.masked_array(malCountry,Mmask))
colors=np.ma.compressed(np.ma.masked_array(lats,Mmask))

MAMa,MAMb,MAMc=np.polyfit(x,ydata,2)
yfit=MAMa*x**2+MAMb*x+MAMc
MAMstdDev=np.std(ydata-x)
s=np.argsort(yfit)
yfit2=np.array(yfit[s])
x2=x[s]
error = abs(yfit-ydata)

fig = plt.figure(figsize=(9, 5))
plt.clf()
plt.scatter(x,ydata,c=colors,marker='*',cmap=cm.jet)
plt.colorbar()
plt.plot(x2,yfit2,'-g')
plt.plot(x2,yfit2-MAMstdDev,'--g',linewidth=0.5)
plt.plot(x2,yfit2+MAMstdDev,'--g',linewidth=0.5)
plt.title('UNICEF MAM vs Paper, Corr = '+str(round(CorrM,2))+', Avg Error = '+str(round(np.mean(error),1))
plt.ylabel('MAM %')
plt.xlabel('Interpolated %')
plt.savefig(wdfigs+'MAMvsInterpolated.pdf')

#############

#####################################
# Country Indices
#####################################

################
index=0

f=open(wddata+'country_indices/gdp_per_cap.csv','r')
i=-2
countries=[]
GDPperCap=np.zeros(shape=(len(africanCountries),nyears))
for line in f:
        i+=1
        if i==-1:
                continue
        line=line[:-2]
        line=line.replace('"','')
        tmp=np.array(line.split(','))
        country=tmp[0]
        if np.amax(country==np.array(africanCountries[:]))==0:
                continue
        countries.append(country)
        icountry=int(countryToi[country])
        j=42 # 1998
        for y in range(19):
                j+=1
                try:
                        GDPperCap[icountry,y]=float(tmp[j])
                except:
                        exit()
```

```
f=open(wddata+'country_indices/gdp_per_cap2018.csv','r')
for line in f:
        line=line[:-1]
        line=line.replace('"','')
        tmp=np.array(line.split(','))
        country=tmp[0]
        if np.amax(country==np.array(africanCountries[:]))==0:
                continue
        icountry=int(countryToi[country])
        GDPperCap[icountry,19]=float(tmp[1])

f=open(wddata+'country_indices/gdp_per_cap2019.csv','r')
for line in f:
        line=line[:-1]
        line=line.replace('"','')
        tmp=np.array(line.split(','))
        country=tmp[0]
        if np.amax(country==np.array(africanCountries[:]))==0:
                continue
        icountry=int(countryToi[country])
        GDPperCap[icountry,20]=float(tmp[1])

f=open(wddata+'country_indices/gdp_per_cap2020.csv','r')
for line in f:
        line=line[:-1]
        line=line.replace('"','')
        tmp=np.array(line.split(','))
        country=tmp[0]
        if np.amax(country==np.array(africanCountries[:]))==0:
                continue
        icountry=int(countryToi[country])
        GDPperCap[icountry,21]=float(tmp[1])


################
index+=1

f=open(wddata+'country_indices/percent_girls_educated.csv','r')

i=-2
g=-1
countries=[]
girlEd=np.zeros(shape=(len(africanCountries),nyears))
girlEdMask=np.zeros(shape=(len(africanCountries),nyears),dtype=bool)
for line in f:
        i+=1
        if i==-1:
                continue
        line=line[:-2]
        line=line.replace('"','')
        tmp=np.array(line.split(','))
        country=tmp[0]
        if np.amax(country==np.array(africanCountries[:]))==0:
                continue
        countries.append(country)
        icountry=int(countryToi[country])
        g+=1
        j=42 # 1998
        for y in range(nyears):
                j+=1
                try:
                        girlEd[icountry,y]=float(tmp[j])
                except:
                        girlEdMask[icountry,y]=1
```

```
for i in range(len(girlEd[:,0])):
        edtmp=np.ma.masked_array(girlEd[i],girlEdMask[i])
        for y in range(nyears):
                ed1=0
                ed2=0
                if girlEd[i,y]==0:
                        j1=y-1
                        if girlEdMask[i,j1]==0:
                                ed1=girlEd[i,j1]
                        for j2 in range(y,nyears):
                                if girlEdMask[i,j2]==0:
                                        ed2=girlEd[i,j2]
                                        break

                        if ed1!=0 and ed2!=0:
                                diff=ed2-ed1
                                missingyears=j2-j1
                                for k in range(y,j2):
                                        girlEd[i,k]=diff/missingyears+girlEd[i,k-1]
                                        girlEdMask[i,k]=0
                        #elif ed1!=0:
                        #       girlEd[i,y]=girlEd[i,j1]
                        #       girlEdMask[i,y]=0
                        elif ed2!=0:
                                girlEd[i,y]=girlEd[i,j2]
                                girlEdMask[i,y]=0
        ed1=girlEd[i,0]
        for y in range(nyears):
                if girlEdMask[i,y+1]==1:
                        ed2=girlEd[i,y]
                        j=y
                        break
        diff=ed2-ed1
        years=j-0
        for k in range(j+1,nyears):
                girlEd[i,k]=diff/years+girlEd[i,k-1]

#################
f=open(wddata+'country_indices/electricity.csv','r')

i=-2
g=-1
countries=[]
electricity=np.zeros(shape=(len(africanCountries),nyears))
electricityMask=np.zeros(shape=(len(africanCountries),nyears),dtype=bool)
for line in f:
        i+=1
        if i==-1:
                continue
        line=line[:-2]
        line=line.replace('"','')
        tmp=np.array(line.split(','))
        country=tmp[0]
        if np.amax(country==np.array(africanCountries[:]))==0:
                continue
        countries.append(country)
        icountry=int(countryToi[country])
        g+=1
        j=42 # 1998
        for y in range(nyears):
                j+=1
                try:
                        electricity[icountry,y]=float(tmp[j])
                except:
                        electricityMask[icountry,y]=1
```

```
        if electricityMask[icountry,0]==0:
                e1=electricity[icountry,0]
        else:
                print 'hmmmm'
                exit()
        for y in range(nyears):
                if electricityMask[icountry,y+1]==1:
                        e2=electricity[icountry,y]
                        j=y
                        break
        diff=e2-e1
        years=j-0
        for k in range(j+1,nyears):
                electricity[icountry,k]=diff/years+electricity[icountry,k-1]

################
f=open(wddata+'country_indices/cereal_yield.csv','r')

i=-2
g=-1
countries=[]
Yield=np.zeros(shape=(len(africanCountries),nyears))
yieldMask=np.zeros(shape=(len(africanCountries),nyears),dtype=bool)
for line in f:
        i+=1
        if i==-1:
                continue
        line=line[:-2]
        line=line.replace('"','')
        tmp=np.array(line.split(','))
        country=tmp[0]
        if np.amax(country==np.array(africanCountries[:]))==0:
                continue
        countries.append(country)
        icountry=int(countryToi[country])
        g+=1
        j=42 # 1998
        for y in range(nyears):
                j+=1
                try:
                        Yield[icountry,y]=float(tmp[j])
                except:
                        yieldMask[icountry,y]=1

        if yieldMask[icountry,11]==0: # yield in 2010
                yield1=Yield[icountry,11]
        else:
                print 'hmmmm'
                exit()
        for y in range(nyears):
                if yieldMask[icountry,y+1]==1:
                        yield2=Yield[icountry,y]
                        j=y
                        break
        diff=yield2-yield1
        years=j-11
        for k in range(j+1,nyears):
                Yield[icountry,k]=diff/years+Yield[icountry,k-1]

################
f=open(wddata+'country_indices/iq.csv','r')

i=-2
g=-1
```

```
countries=[]
iq=np.zeros(shape=(len(africanCountries),nyears))
iqMask=np.zeros(shape=(len(africanCountries),nyears),dtype=bool)
for line in f:
        i+=1
        if i==-1:
                continue
        tmp=np.array(line.split(','))
        country=tmp[0]
        if np.amax(country==np.array(africanCountries[:]))==0:
                continue
        countries.append(country)
        icountry=int(countryToi[country])
        g+=1
        j=42 # 1998
        for y in range(nyears):
                j+=1
                try:
                        iq[icountry,y]=float(tmp[2])
                except:
                        iqMask[icountry,y]=1

#################
#f=open(wddata+'country_indices/fertility.csv','r')
#
#i=-2
#g=-1
#countries=[]
#fertility=np.zeros(shape=(len(africanCountries),nyears))
#fertilityMask=np.zeros(shape=(len(africanCountries),nyears),dtype=bool)
#for line in f:
#       i+=1
#       if i==-1:
#               continue
#       line=line[:-2]
#       line=line.replace('"','')
#       tmp=np.array(line.split(','))
#       country=tmp[0]
#       if np.amax(country==np.array(africanCountries[:]))==0:
#               continue
#       countries.append(country)
#       icountry=int(countryToi[country])
#       g+=1
#       j=42 # 1998
#       for y in range(nyears):
#               j+=1
#               try:
#                       fertility[icountry,y]=float(tmp[j])
#               except:
#                       fertilityMask[icountry,y]=1
#
#Corr[index] = corr(np.ma.compressed(fertility),np.ma.compressed(malCountry))
#
#################
#index+=1
#
#f=open(wddata+'country_indices/under5mortality.csv','r')
#
#i=-2
#g=-1
#countries=[]
#mortality5=np.zeros(shape=(len(africanCountries),nyears))
#mortality5Mask=np.zeros(shape=(len(africanCountries),nyears),dtype=bool)
#for line in f:
#       i+=1
```

```
#       if i==-1:
#               continue
#       line=line[:-2]
#       line=line.replace('"','')
#       tmp=np.array(line.split(','))
#       country=tmp[0]
#       if np.amax(country==np.array(africanCountries[:]))==0:
#               continue
#       countries.append(country)
#       icountry=int(countryToi[country])
#       g+=1
#       j=42 # 1998
#       for y in range(nyears):
#               j+=1
#               try:
#                       mortality5[icountry,y]=float(tmp[j])
#               except:
#                       mortality5Mask[icountry,y]=1
#
#Corr[index] = corr(np.ma.compressed(mortality5),np.ma.compressed(malCountry))
#
#################
#index+=1
#
#f=open(wddata+'country_indices/corruption.csv','r')
#
#i=-2
#g=-1
#countries=[]
#corruption=np.zeros(shape=(len(africanCountries),nyears))
#corruptionMask=np.zeros(shape=(len(africanCountries),nyears),dtype=bool)
#for line in f:
#       i+=1
#       if i==-1:
#               continue
#       line=line[:-2]
#       line=line.replace('"','')
#       tmp=np.array(line.split(','))
#       country=tmp[0]
#       if np.amax(country==np.array(africanCountries[:]))==0:
#               continue
#       countries.append(country)
#       icountry=int(countryToi[country])
#       g+=1
#       j=42 # 1998
#       for y in range(nyears):
#               j+=1
#               try:
#                       corruption[icountry,y]=float(tmp[6])
#               except:
#                       corruptionMask[icountry,y]=1
#
#Corr[index] = corr(np.ma.compressed(corruption),np.ma.compressed(malCountry))
#
################

#indices=[GDPperCap[:nyearsT],girlEd[:nyearsT],electricity[:nyearsT],Yield[:nyearsT],iq[:nyearsT]]
#indexNames=['GDPperCap','girlEd','electricity','Yield','iq']
#xMulti=np.ma.zeros(shape=(len(africanCountries),nyears,len(indices)))
#
#Mask = yieldMask + girlEdMask
#ydata=np.ma.compressed(np.ma.masked_array(malCountry,Mask[:,:nyearsT]))
#
#xMultiC=np.zeros(shape=(len(ydata),len(indices)))
#for i in range(len(indices)):
```

```
#       xMultiC[:,i]=np.ma.compressed(np.ma.masked_array(indices[i],Mask[:,:nyearsT]))
#
#malPred=np.zeros(shape=(10,141))
#malActual=np.zeros(shape=(10,141))
#for i in range(10):
#       Xtrain,Xtest,Ytrain,Ytest=sklearn.model_selection.train_test_split(xMultiC,ydata,test_size=.2,tra
#
#       clf=linear_model.LinearRegression()
#       clf.fit(Xtrain,Ytrain)
#       #clf=RandomForestRegressor()
#       #clf.fit(Xtrain,Ytrain)
#
#       malPred[i]=clf.predict(Xtest)
#       malActual[i]=Ytest
#
#MultiCorr=corr(np.ma.compressed(malPred),np.ma.compressed(malActual))
#print MultiCorr
#
#### Yield Correlations
#i=13 #ethiopia for now
#
## detrend
#YieldD=np.zeros(shape=(Yield.shape))
#
#x=np.arange(2000,2016)
#slope,b=np.polyfit(x,Yield[i],1)
#yfit=slope*x+b
#
#YieldD[i]=Yield[i]-yfit
#
#######################

#plt.clf()
#plt.plot(np.ma.compressed(malActual),np.ma.compressed(malPred),'b*')
#plt.title('Acuracy of Malnutrition Predictions, Corr = '+str(round(MultiCorr,2)))
#plt.xlabel('Actual Prevalence')
#plt.ylabel('Predicted Prevalence')
#plt.gca().set_aspect('equal', adjustable='box')
#plt.grid(True)
#plt.xlim([0,27])
#plt.ylim([0,27])
#plt.savefig(wdfigs+'world_bank_malnutrition_predictions.pdf')

indices=[GDPperCap,girlEd,electricity,Yield,iq]
indexNames=['GDPperCap','girlEd','electricity','Yield','iq']
titles=['GDP per cap','% Females with Secondary School Education','% Population with Access to Electricity

for j in range(len(indices)):
        try:
                vars()[indexNames[j]+'Grid']=np.load(wdvars+indexNames[j]+'Grid_1999-2020')
        except:
                print 'try command failed: calculating',indexNames[j]+'Grid'

                vars()[indexNames[j]+'Grid']=np.zeros(shape=(nyears,len(latm),len(lonm)))
                k=-1
                for i in indexedcodes:
                        k+=1
                        for y in range(nyears):
                                vars()[indexNames[j]+'Grid'][y,nations==i]=indices[j][k,y]

                vars()[indexNames[j]+'Grid'] = np.ma.masked_array(vars()[indexNames[j]+'Grid'], imageMask
                vars()[indexNames[j]+'Grid'][vars()[indexNames[j]+'Grid'] == 0] = np.mean(vars()[indexName

                #if MakePlots:
```

```
                vars()[indexNames[j]+'Grid'].dump(wdvars+indexNames[j]+'Grid_1999-2020')

j=3
plt.clf()
plt.imshow(vars()[indexNames[j]+'Grid'][21],cmap=cm.jet_r)
plt.colorbar()
plt.yticks([])
plt.xticks([])
plt.title(titles[j]+', 2020')
plt.savefig(wdfigs+indexNames[j]+'2020.png',dpi=700)

##########################################

####################################
# Data from Matt
####################################
print 'data from Matt'

indices = ['ag_pct_gdp','assistance','bare','builtup','crop_prod','elevation','enrollment','fieldsize','f

try:
        for i in range(len(indices)):
                fromMatt=indices[i]
                vars()[fromMatt]=np.load(wdvars+fromMatt)
except:
        print 'try command failed: retrieveing dataFromMatt'

        for i in range(len(indices)):
                vars()[indices[i]]=np.zeros(shape=(nyears,len(latm),len(lonm)))

        year=1998
        for y in range(nyears):
                year+=1
                print year
                for i in range(len(indices)):
                        fromMatt=indices[i]

                        ds=gdal.Open(wddata+'data_from_Matt/Data/'+str(year)+'/'+fromMatt+'.tif')
                        width = ds.RasterXSize
                        height = ds.RasterYSize
                        gt = ds.GetGeoTransform()
                        minx = gt[0]
                        miny = gt[3] + width*gt[4] + height*gt[5]
                        maxx = gt[0] + width*gt[1] + height*gt[2]
                        maxy = gt[3]
                        pixelsizeM=abs(gt[-1])

                        latc=np.ones(shape=(height))
                        lonc=np.ones(shape=(width))
                        for w in range(width):
                                lonc[w]=minx+w*pixelsizeM
                        for h in range(height):
                                latc[h]=miny+h*pixelsizeM

                        latc=latc[::-1]

                        datatmp=ds.ReadAsArray()
                        ##### Scale to Africa #####
                        datatmp=datatmp[latc<np.amax(latm)+pixelsize]
                        latc=latc[latc<np.amax(latm)+pixelsize]
                        datatmp=datatmp[latc>np.amin(latm)]
                        latc=latc[latc>np.amin(latm)]

                        datatmp=datatmp[:,lonc<np.amax(lonm)+pixelsize]
                        lonc=lonc[lonc<np.amax(lonm)+pixelsize]
```

```
                        datatmp=datatmp[:,lonc>np.amin(lonm)]
                        lonc=lonc[lonc>np.amin(lonm)]

                        ##### Scale to 5km #####
                        datatmp[datatmp<-100]=0
                        latl=np.radians(latc[::-1])+1.2
                        lonl=np.radians(lonc)+1.2
                        lut=RectSphereBivariateSpline(latl, lonl, datatmp)

                        newLats,newLons=np.meshgrid(np.radians(latm[::-1])+1.2,np.radians(lonm)+1.2)
                        vars()[fromMatt][y]=lut.ev(newLats.ravel(),newLons.ravel()).reshape((len(lonm),le

        for i in range(len(indices)):
                fromMatt=indices[i]
                vars()[fromMatt]=np.ma.masked_array(vars()[fromMatt],imageMask2)
                vars()[fromMatt].dump(wdvars+fromMatt)

for i in range(len(indices)):
        fromMatt=indices[i]

        year=2014
        for y in range(1):
                year+=1

                plt.clf()
                plt.imshow(vars()[fromMatt][y],cmap=cm.nipy_spectral)
                plt.yticks([])
                plt.xticks([])
                plt.title(fromMatt)
                plt.colorbar()
                plt.savefig(wdfigs +fromMatt+str(year),dpi=700)

plt.clf()
plt.imshow(female_education[16],cmap=cm.nipy_spectral)
plt.colorbar()
plt.xticks([])
plt.yticks([])
plt.title('2015 Female Education, Years of Attainment')
plt.savefig(wdfigs+'female_education_2015.png',dpi=500)

plt.clf()
plt.imshow(mean_annual_precip[16]/1000.,cmap=cm.nipy_spectral)
plt.colorbar()
plt.xticks([])
plt.yticks([])
plt.title('2015 Mean Precipitation (m)')
plt.savefig(wdfigs+'mean_annual_precip_2015.png',dpi=500)

plt.clf()
plt.imshow(forest[16],cmap=cm.rainbow)
plt.colorbar()
plt.xticks([])
plt.yticks([])
plt.title('Forest Cover (%)')
plt.savefig(wdfigs+'forest_2015.png',dpi=500)
exit()


##########################################
# Conflicts
##########################################
print 'Conflicts'
try:
        MPconflicts=np.load(wdvars+'MPconficts.npy')
except:
```

```
        conflictsAll=np.load(wdvars+'africa_fatalities')
        conflictCoordAll=np.load(wdvars+'conflictCoord')
        MPconflicts=np.zeros(shape=(nyears,len(latm),len(lonm)))

        year=1998
        k=-1
        for y in range(99,118):
                k+=1
                year+=1
                print '\n\n',year,'\n\n\n'
                conflicts=np.ma.compressed(conflictsAll[y:y+1])
                conflictCoord=np.zeros(shape=(len(conflicts),2))
                conflictCoord[:,0]=np.ma.compressed(conflictCoordAll[y:y+1,:,:,0])
                conflictCoord[:,1]=np.ma.compressed(conflictCoordAll[y:y+1,:,:,1])

                MPconflicts[k]=marketPotentials(conflicts,conflictCoord,gridMid,imageMask2[0,:,:])

        for k in range(19,nyears):
                MPconflicts[k]=np.mean(MPconflicts[15:19],axis=0)

        np.save(wdvars+'MPconficts',MPconflicts)

year=2014
k=14
for y in range(1):
        k+=1
        year+=1

        plt.clf()
        plt.imshow(MPconflicts[k],vmax=12000,cmap=cm.gist_heat_r)
        plt.colorbar()
        plt.xticks([])
        plt.yticks([])
        plt.title(str(year)+' Conflicts Index')
        plt.savefig(wdfigs+'MPconflicts_'+str(year)+'.png',dpi=500)
exit()

MPconflicts=np.ma.masked_array(MPconflicts,MPconflicts==0)

'''
#######################################################
# Travel Time
#######################################################
print 'Travel Time'
try:
        travel1year=np.load(wdvars+'travelAfrica')

        travel=np.ma.zeros(shape=(nyears,len(latm),len(lonm)))
        for y in range(nyears):
                travel[y]=travel1year
except:
        ds=gdal.Open(wddata+'travel_time/accessibility_to_cities_2015_v1.0.tif')
        width = ds.RasterXSize
        height = ds.RasterYSize
        gt = ds.GetGeoTransform()
        minx = gt[0]
        miny = gt[3] + width*gt[4] + height*gt[5]
        maxx = gt[0] + width*gt[1] + height*gt[2]
        maxy = gt[3]
        pixelsizel=abs(gt[-1])

        latt=np.ones(shape=(height))
        lnt=np.ones(shape=(width))
        for w in range(width):
                lont[w]=minx+w*pixelsizel
```

```
        for h in range(height):
                latt[h]=miny+h*pixelsizel

        latt=latt[::-1]
        if latm[0]<latm[-1]:
                latm=latm[::-1]

        travelWorld=ds.ReadAsArray()
        ##### Scale to Africa #####
        travel=travelWorld[latt<np.amax(latm)+pixelsize]
        latt=latt[latt<np.amax(latm)+pixelsize]
        travel=travel[latt>np.amin(latm)]
        latt=latt[latt>np.amin(latm)]

        travel=travel[:,lont<np.amax(lonm)+pixelsize]
        lont=lont[lont<np.amax(lonm)+pixelsize]
        travel=travel[:,lont>np.amin(lonm)]
        lont=lont[lont>np.amin(lonm)]

        travel[travel<0]=-10
        travelbi=np.array(travel)
        travelbi[travel!=0]=1
        travelbi=1-travelbi

        ### 5km ###
        latl=np.radians(latt[::-1])+1.2
        lonl=np.radians(lont)+1.2
        lut=RectSphereBivariateSpline(latl, lonl, travel)

        newLats,newLons=np.meshgrid(np.radians(latm[::-1])+1.2,np.radians(lonm)+1.2)
        travel=lut.ev(newLats.ravel(),newLons.ravel()).reshape((len(lonm),len(latm))).T

        travel=np.ma.masked_array(travel,imageMask2)
        travel.dump(wdvars+'travelAfrica')

plt.clf()
plt.imshow(travel[0],cmap=cm.gist_earth_r,vmin=0,vmax=800)
plt.title('Travel Time To Nearest City, Hours')
plt.colorbar()
plt.xticks([])
plt.yticks([])
plt.savefig(wdfigs+'travel',dpi=700)
'''

##########################################
# Coast Lines
##########################################
print 'Coasts'
try:
        distToCoasts1year=np.load(wdvars+'Africa_distToCoasts.npy')
        coastLines=np.load(wdvars+'Africa_coastLines.npy')

        distToCoasts=np.ma.zeros(shape=(nyears,2,len(latm),len(lonm)))
        for y in range(nyears):
                for i in range(2):
                        distToCoasts[y,i]=distToCoasts1year[:,:,i]
except:
        print 'calculating distToCoasts: try command failed'
        rCoasts0=shapefile.Reader(wddata+'nigeria_landtype/nigeria_coastline/noaa_gshhg/GSHHS_shp/f/GSHHS_
        rCoasts1=shapefile.Reader(wddata+'nigeria_landtype/nigeria_coastline/noaa_gshhg/GSHHS_shp/f/GSHHS_
        shapes0=rCoasts0.shapes
        shapes1=rCoasts1.shapes
        shapes0=shapes0()
        shapes1=shapes1()
```

```
        coastLines=np.zeros(shape=(len(latm),len(lonm),2))
        distToCoasts=np.zeros(shape=(len(latm),len(lonm),2))
        for coastType in range(2):
                print '\n Coast',coastType
                for i in range(len(vars()['shapes'+str(coastType)])):
                        print round(100*i/float(len(vars()['shapes'+str(coastType)])),2),'%'
                        pointsLonLat=np.array(vars()['shapes'+str(coastType)][i].points)
                        points=np.array(pointsLonLat)
                        points[:,0]=pointsLonLat[:,1]
                        points[:,1]=pointsLonLat[:,0]

                        if (points[:,1]>lonm[0]).any() and (points[:,1]<lonm[-1]).any():
                                if (latm[-1]<points[:,1]).any() and (points[:,1]<latm[0]).any():
                                        # lon mask
                                        moreThanLon=lonm[0]<points[:,1]
                                        lessThanLon=points[:,1]<lonm[-1]
                                        lonMask=moreThanLon==lessThanLon
                                        # lat mask
                                        moreThanLat=latm[-1]<points[:,0]
                                        lessThanLat=points[:,0]<latm[0]
                                        latMask=moreThanLat==lessThanLat
                                        # total mask
                                        nigeriaMask=np.zeros(shape=(len(latMask)),dtype=bool)
                                        for i in range(len(latMask)):
                                                if latMask[i]==True and lonMask[i]==True:
                                                        nigeriaMask[i]=True
                                        if np.sum(nigeriaMask)==0:
                                                continue

                                        pointsM=points[nigeriaMask]
                                        coastLines[:,:,coastType]+=findGridDataLays(pointsM,gridMid)
                coastLines[coastLines>1]=1

                coastsMidPoints=gridMid[coastLines[:,:,coastType]==1,:]
                distToCoasts[:,:,coastType],iclosest=findNearest(coastsMidPoints,gridMid,imageMask2)

        np.save(wdvars+'Africa_coastLines.npy',coastLines)
        np.save(wdvars+'Africa_distToCoasts.npy',distToCoasts)

distToCoasts=np.ma.masked_array(distToCoasts,imageMask3[:,:2,:,:])
coastLines=np.ma.masked_array(coastLines,imageMask3[0,:2,:,:])

plt.clf()
plt.imshow(np.clip(distToCoasts[0,1,:,:],0,700),cmap=cm.YlGnBu_r)
plt.colorbar()
plt.xticks([])
plt.yticks([])
plt.title('Distance to Coastlines, km')
plt.savefig(wdfigs+'distToCoasts',dpi=700)

##########################################
# Machine learning
##########################################
print 'Machine Learning'

indices = ['ag_pct_gdp','assistance','bare','builtup','crop_prod','elevation','enrollment','fieldsize','fo

indices = [ag_pct_gdp,assistance,bare,builtup,crop_prod,elevation,enrollment,fieldsize,forest,government_e

xMulti=np.zeros(shape=(nyears,len(latm),len(lonm),len(indices)))

exit()
for i in range(len(indices)):
        xMulti[:,:,:,i]=indices[i]
        print i
```

```
if MakePlots:
        for i in range(len(indices)):
                plt.clf()
                plt.imshow(xMulti[10,:,:,i])
                plt.title(indexNames[i])
                plt.savefig(wdfigs+indexNames[i]+'2010',dpi=700)

#np.save(wdvars+'xMulti.npy',xMulti)

#if nyears==22:
xMultiMask=np.zeros(shape=(nyears,len(latm),len(lonm),len(indices)),dtype=bool)
for i in range(len(indices)):
        xMultiMask[:,:,:,i]=imageMask2
        print i
xMultiMask=np.array(xMultiMask,dtype=bool)

#np.save(wdvars+'xMultiMask.npy',xMultiMask)

xMulti = np.load(wdvars+'xMulti.npy')
xMultiMask = np.load(wdvars+'xMultiMask.npy')

xMultiTrain=np.ma.masked_array(xMulti[:nyearsT], xMultiMask[:nyearsT])
xMultiTest=np.ma.masked_array(xMulti[nyearsT:], xMultiMask[nyearsT:])
ydata=np.ma.compressed(mal)

tmp=np.ma.compressed(xMultiTrain[:,:,:,0])
xMultiTrainC=np.zeros(shape=(len(tmp),len(indices)))
tmp=np.ma.compressed(xMultiTest[:,:,:,0])
xMultiTestC=np.zeros(shape=(len(tmp),len(indices)))
for i in range(len(indices)):
        xMultiTrainC[:,i]=np.ma.compressed(xMultiTrain[:,:,:,i])
        #xMultiTestC[:,i]=np.ma.compressed(xMultiTest[:,:,:,i])
        print i

tmp=np.ma.compressed(xMultiTest[0,:,:,0])
xMultiTestC2016=np.zeros(shape=(len(tmp),len(indices)))
xMultiTestC2017=np.zeros(shape=(len(tmp),len(indices)))
xMultiTestC2018=np.zeros(shape=(len(tmp),len(indices)))
xMultiTestC2019=np.zeros(shape=(len(tmp),len(indices)))
xMultiTestC2020=np.zeros(shape=(len(tmp),len(indices)))
xMultiTestC2021=np.zeros(shape=(len(tmp),len(indices)))
for i in range(len(indices)):
        xMultiTestC2016[:,i]=np.ma.compressed(xMultiTest[0,:,:,i])
        xMultiTestC2017[:,i]=np.ma.compressed(xMultiTest[1,:,:,i])
        xMultiTestC2018[:,i]=np.ma.compressed(xMultiTest[2,:,:,i])
        xMultiTestC2019[:,i]=np.ma.compressed(xMultiTest[3,:,:,i])
        xMultiTestC2020[:,i]=np.ma.compressed(xMultiTest[4,:,:,i])
        xMultiTestC2021[:,i]=np.ma.compressed(xMultiTest[5,:,:,i])
        print i

print 'fitting'
##############################
clf=RandomForestRegressor()
clf.fit(xMultiTrainC,ydata)

pickle.dump(clf, open(wdvars+'randomForest_trained1999-2014all', 'wb'))

#malPred2016=clf.predict(xMultiTestC2016)
#malPred2017=clf.predict(xMultiTestC2017)
#malPred2018=clf.predict(xMultiTestC2018)
#malPred2019=clf.predict(xMultiTestC2019)
#malPred2020=clf.predict(xMultiTestC2020)
#malPred2021=clf.predict(xMultiTestC2021)
```

```
### Plot
imageMask2=np.load(wdvars+'imageMask2.npy')
imageMask2_1year=imageMask2[0]
imageMask2=np.zeros(shape=(nyears,len(latm),len(lonm)))
for y in range(nyears):
        imageMask2[y]=imageMask2_1year

malAll = np.zeros(shape=(nyears,len(latm),len(lonm)))
malAll[:nyearsT]=mal

year = 2015
for y in range(6):
        year+=1
        #xMultiPlot=np.ma.array(xMultiTest[y])
        #
        #malPred=np.zeros(shape=(len(latm),len(lonm)))
        #malPred=np.ma.masked_array(malPred,imageMask2[y])
        #malPredMask=np.zeros(shape=(len(latm),len(lonm)),dtype=bool)
        #
        #for i in range(len(latm)):
        #       if np.amin(imageMask2[y,i,:])!=1:
        #               xplot=np.zeros(shape=(len(np.ma.compressed(xMultiPlot[i,:,0])),len(indices)))
        #               for k in range(len(indices)):
        #                       xplot[:,k]=np.ma.compressed(xMultiPlot[i,:,k])
        #               malPred[i][imageMask2[y,i]==0]=clf.predict(xplot)

        #vars()['malPred'+str(year)] = malPred
        #malAll[y+16]=malPred
        malPred=np.load(wdvars+'stuffForFinalPrediction/malPred'+str(year))
        malAll[y+16]=malPred
        ##vars()['malPred'+str(year)].dump(wdvars+'stuffForFinalPrediction/malPred'+str(year))

        #plt.clf()
        #plt.imshow(malPred,cmap=cm.jet,vmin=0,vmax=0.3)
        #plt.colorbar()
        #plt.yticks([])
        #plt.xticks([])
        #plt.title(str(year)+' Predicted Malnutrition, % of Population')
        #plt.savefig(wdfigs+'malPred_Africa'+str(year),dpi=700)

malAll=np.ma.masked_array(malAll,imageMask2)

############################
# Sum for Country
############################
malCountryAll=np.zeros(shape=(len(indexedcodes),nyears))
j=-1
for i in indexedcodes:
        j+=1
        for y in range(nyears):
                popWhole=np.sum(population[y,nations==i])
                poptmp=np.ma.masked_array(population[y],nations!=i)
                maltmp=np.ma.masked_array(malAll[y],nations!=i)
                malnumtmp=maltmp*poptmp
                malCountryAll[j,y]=np.sum(malnumtmp)/popWhole
malCountryAll=malCountryAll*100

SAMcountry = np.zeros(shape=(3,len(malCountryAll),nyears))
MAMcountry = np.zeros(shape=(3,len(malCountryAll),nyears))
for i in range(len(malCountryAll)):
        for y in range(nyears):
                x=malCountryAll[i,y]
                SAMcountry[1,i,y]=SAMa*x**2 + SAMb*x + SAMc
                MAMcountry[1,i,y]=MAMa*x**2 + MAMb*x + MAMc
```

```
                SAMcountry[0,i,y]=np.amax([SAMcountry[1,i,y]-SAMstdDev,0.0])
                MAMcountry[0,i,y]=np.amax([MAMcountry[1,i,y]-MAMstdDev,0.0])

                SAMcountry[2,i,y]=SAMcountry[1,i,y]+SAMstdDev
                MAMcountry[2,i,y]=MAMcountry[1,i,y]+MAMstdDev

f=open(wddata+'country_indices/country_population.csv','r')
i=-2
countries=[]
countryPop=np.zeros(shape=(len(africanCountries),nyears))
for line in f:
        i+=1
        if i==-1:
                continue
        line=line[:-2]
        line=line.replace('"','')
        tmp=np.array(line.split(','))
        country=tmp[0]
        if np.amax(country==np.array(africanCountries[:]))==0:
                continue
        countries.append(country)
        icountry=int(countryToi[country])
        j=43 # 1998
        for y in range(19):
                j+=1
                try:
                        countryPop[icountry,y]=float(tmp[j])
                except:
                        print country
                        continue

        p1=countryPop[icountry,0]
        for y in range(nyears):
                if countryPop[icountry,y+1]==0.:
                        p2=countryPop[icountry,y]
                        j=y
                        break
        diff=p2-p1
        years=j-0
        for k in range(j+1,nyears):
                countryPop[icountry,k]=diff/years+countryPop[icountry,k-1]

MAMburden=np.zeros(shape=(MAMcountry.shape))
SAMburden=np.zeros(shape=(SAMcountry.shape))
for i in range(3):
        MAMburden[i,:,:]=(MAMcountry[i]/100.)*countryPop
        SAMburden[i,:,:]=(SAMcountry[i]/100.)*countryPop

np.save(wdvars+'MAMcountry.npy',MAMcountry)
np.save(wdvars+'SAMcountry.npy',SAMcountry)
np.save(wdvars+'MAMburden.npy',MAMburden)
np.save(wdvars+'SAMburden.npy',SAMburden)

############################
# Split into boxes
############################
if latm[0]>latm[-1]:
        latm=latm[::-1]
ysplit=len(latm)/20.
xsplit=len(lonm)/20.

yboxes=np.arange(0,len(latm)+1,ysplit)
xboxes=np.arange(0,len(lonm)+1,xsplit)

boxesGrid=np.zeros(shape=(gridm.shape))
```

```
for i in range(len(yboxes)-1):
        boxesGrid[:,:,0][gridm[:,:,0]>latm[int(np.round(yboxes[i],0))]]=i
for i in range(len(xboxes)-1):
        boxesGrid[:,:,1][gridm[:,:,1]>lonm[int(np.round(xboxes[i],0))]]=i

plt.clf()
plt.imshow(boxesGrid[:,:,0])
plt.title('Latitude Boxes')
plt.colorbar()
plt.savefig(wdfigs+'latboxes.pdf')
plt.clf()
plt.imshow(boxesGrid[:,:,1])
plt.title('Longitude Boxes')
plt.colorbar()
plt.savefig(wdfigs+'lonboxes.pdf')

boxes=np.zeros(shape=(len(latm),len(lonm)))
k=-1
for i in range(len(yboxes)):
        for j in range(len(xboxes)):
                k+=1
                ytrue=boxesGrid[:,:,0]==i
                xtrue=boxesGrid[:,:,1]==j
                true=ytrue*xtrue
                boxes[true]=k

boxes=np.array(boxes)
boxNums=np.unique(boxes)
boxNums=np.array(boxNums,dtype=int)

boxMask=np.zeros(shape=(boxes.shape))
for i in boxNums:
        masktmp=np.zeros(shape=(boxes.shape),dtype=bool)
        masktmp[boxes==i]=1
        if np.sum(1-imageMask2[0,masktmp])<100:
                boxMask[masktmp==1]=1
boxes=np.ma.masked_array(boxes,boxMask)

boxNums=np.unique(boxes)
boxNums=boxNums[np.ma.getmask(boxNums)==False]
boxNums=np.array(boxNums)

trainNums,testNums=sklearn.model_selection.train_test_split(boxNums,test_size=.2,train_size=.8)
testPixelsAll=[boxes==trainNums[i] for i in range(len(trainNums))]
testPixels1year=np.sum(testPixelsAll,axis=0)
trainPixelsAll=[boxes==testNums[i] for i in range(len(testNums))]
trainPixels1year=np.sum(trainPixelsAll,axis=0)

trainPixels=np.zeros(shape=(nyears,len(latm),len(lonm)))
testPixels=np.zeros(shape=(nyears,len(latm),len(lonm)))
for y in range(nyears):
        trainPixels[y]=trainPixels1year
        testPixels[y]=testPixels1year

maltrain=np.ma.array(mal)
maltrain=np.ma.masked_array(maltrain,trainPixels)
maltest=np.ma.array(mal)
maltest=np.ma.masked_array(maltest,testPixels)

plt.clf()
plt.imshow(maltrain[4],cmap=cm.jet,vmax=0.3)
plt.title('Train Pixels')
plt.colorbar()
plt.savefig(wdfigs+'maltrain.png',dpi=700)
plt.clf()
```

```
plt.imshow(maltest[4],cmap=cm.jet,vmax=0.3)
plt.title('Test Pixels')
plt.colorbar()
plt.savefig(wdfigs+'maltest.png',dpi=700)

if MakePlots:
        boxMask=np.ma.masked_array(boxMask,testPixels)
        plt.clf()
        plt.imshow(boxMask,cmap=cm.seismic)
        plt.title('Boxes')
        plt.colorbar()
        plt.savefig(wdfigs+'trainboxes.png',dpi=700)

        plt.clf()
        plt.imshow(boxes,cmap=cm.seismic)
        plt.title('Boxes')
        plt.colorbar()
        plt.savefig(wdfigs+'boxes.png',dpi=700)

trainMask=np.ma.getmask(maltrain)
testMask=np.ma.getmask(maltest)
trainMask3=np.zeros(shape=(nyears,len(latm),len(lonm),len(indices)),dtype=bool)
testMask3=np.zeros(shape=(nyears,len(latm),len(lonm),len(indices)),dtype=bool)
for i in range(len(indices)):
        trainMask3[:,:,:,i]=trainMask
        testMask3[:,:,:,i]=testMask
        print i

xMultiTest=np.ma.masked_array(xMulti, testMask3)
xMultiTrain=np.ma.masked_array(xMulti, trainMask3)

xMultiTrainC=np.zeros(shape=(len(np.ma.compressed(xMultiTrain[:,:,:,0])),len(indices)))
xMultiTestC=np.zeros(shape=(len(np.ma.compressed(xMultiTest[:,:,:,0])),len(indices)))
for i in range(len(indices)):
        xMultiTrainC[:,i]=np.ma.compressed(xMultiTrain[:,:,:,i])
        xMultiTestC[:,i]=np.ma.compressed(xMultiTest[:,:,:,i])
        print i
ydataTrain=np.ma.compressed(maltrain)
ydataTest=np.ma.compressed(maltest)

exit()
print 'fitting'
################################
clf=RandomForestRegressor()
clf.fit(xMultiTrainC,ydataTrain)
malPred=clf.predict(xMultiTestC)

#clf=linear_model.LinearRegression()
#clf.fit(xMultiTrainC,ydataTrain)
#malPred=clf.predict(xMultiTestC)
################################

Corr=corr(malPred,ydataTest)
print Corr

x,y=ydataTest*100,malPred*100
slope,b=np.polyfit(x,y,1)
yfit=slope*x+b
error=np.mean((y-x)/x)*100
heatmap, xedges, yedges = np.histogram2d(x, y, bins=(200,200),normed=True)
extent = [xedges[0], xedges[-1], yedges[0], yedges[-1]]
heatmap = heatmap.T

plt.clf()
fig = plt.figure(figsize=(7, 5))
```

```
ax = fig.add_subplot(111, title='Accuracy of Predicted Malnutrition, Corr = '+str(round(Corr,2))+', Avg Er
X, Y = np.meshgrid(xedges, yedges)
ax.pcolormesh(X, Y, heatmap,cmap=cm.terrain_r)
plt.xlabel('Actual Malnutrition, % population')
plt.ylabel('Predicted Malnutrition, % population')
ax.set_xlim([0,30])
ax.set_ylim([0,30])
plt.savefig(wdfigs+'accuracy_predictions_heatmap_multivariate.pdf')
exit()

importances=clf.feature_importances_
importanceOrder=np.argsort(importances)
indexNames=np.array(indexNames)
importanceVars=indexNames[importanceOrder][::-1]
exit()

y=15

imageMask2=np.load(wdvars+'imageMask2.npy')
traintmp=imageMask2[y]==testMask[y]
testtmp=imageMask2[y]==trainMask[y]
ydata=maltrain[y]
ydata=ydata.filled(0)
ydata=np.ma.masked_array(ydata,imageMask2[y])

xMultiPlot=np.ma.array(xMultiTest[y])
for i in range(len(indices)):
        xMultiPlot[:,:,i]=np.ma.masked_array(xMultiPlot[:,:,i],testtmp)

malPred=np.zeros(shape=(len(latm),len(lonm)))
malPred=np.ma.masked_array(malPred,imageMask2[y])
malPredMask=np.zeros(shape=(len(latm),len(lonm)),dtype=bool)

x1test=0
x1train=0
for i in range(len(latm)):
        malPred[i]=ydata[i]
        if len(np.where(ydata[i]==0)[0])!=0:
                xplot=np.zeros(shape=(len(np.ma.compressed(xMultiPlot[i,:,0])),len(indices)))
                for k in range(len(indices)):
                        xplot[:,k]=np.ma.compressed(xMultiPlot[i,:,k])
                malPred[i][ydata[i]==0]=clf.predict(xplot)
                malPredMask[i][ydata[i]==0][0]=0
                malPredMask[i][ydata[i]==0][-1]=0

plt.clf()
plt.imshow(malPred,cmap=cm.jet,vmin=0,vmax=0.3)
plt.colorbar()
plt.yticks([])
plt.xticks([])
plt.title('Predicted Malnutrition, Percent of Population')
plt.savefig(wdfigs+'malPred_Africa',dpi=700)

plt.clf()
plt.imshow(maltrain[y],cmap=cm.jet,vmin=0,vmax=0.3)
plt.colorbar()
plt.yticks([])
plt.xticks([])
plt.title('Malnutrition Prevalence, Percent of Population')
plt.savefig(wdfigs+'maltrain_Africa',dpi=700)

Corr=corr(np.ma.compressed(malPred),np.ma.compressed(mal))

##########################################
# Read in saved variables for plots
```

```
##########################################

caseload=np.zeros(shape=(nyears,len(latm),len(lonm)))
avgMal=np.zeros(shape=(nyears))
for y in range(22):
        if y<=15:
                vars()['mal'+str(y)] = mal[y]
        else:
                vars()['mal'+str(y)] = np.load(wdvars+'stuffForFinalPrediction/malPred20'+str(y))
        caseload[y]=vars()['mal'+str(y)] * popUnder5[y]
        avgMal[y]=np.mean(vars()['mal'+str(y)])

caseload=np.ma.masked_array(caseload,imageMask2)
caseloadsum=np.zeros(shape=(nyears))
for y in range(22):
        print np.sum(caseload[y])/1e6
        caseloadsum[y]=np.sum(caseload[y])/1e6

x=np.arange(0,21.1,5)
ydata = [caseloadsum[0],caseloadsum[5],caseloadsum[10],caseloadsum[15],caseloadsum[20]]
plt.clf()
plt.plot(x,ydata,'r*-')
plt.title('Total Acute Malnutrition Caseload')
plt.savefig(wdfigs+'caseload_2000-2021.pdf')

x=np.arange(0,21.1)
plt.clf()
plt.plot(x,avgMal,'r*-')
plt.title('Avg Acute Malnutrition Prevalence')
plt.savefig(wdfigs+'avgmal_2000-2021.pdf')

plt.clf()
plt.imshow(caseload[21],cmap=my_cmap,vmax=200)
plt.title('2021 Caseload of Acute Malnutrition \nin Children Under 5',fontsize=18)
plt.colorbar(label='Children per 5km^2')
plt.xticks([])
plt.yticks([])
plt.text(350,1000,'13.7 Million Total Cases',horizontalalignment='center')
plt.savefig(wdfigs+'caseload2021',dpi=500)
```