Data Processing: Shapefile Processing

Team 42: Elise Penn, Manish Vuyyuru, Victor Sheng, Yajaira Gonzalez

Processes shapefiles to get population overlap for use in imputing data.

Needs to be in the folder /shapefiles in our github repo, where it has access to the shapefile data (too big to upload here.)

Takes ~5 hours to run all the way through.

In [ ]:
```
import numpy as np
import pickle
import cartopy.io.shapereader as shpreader
from cartopy.feature import ShapelyFeature
from shapely.prepared import prep
import matplotlib.pyplot as plt
import cartopy.crs as ccrs
import pandas as pd
```

In [ ]:
```
# Projection is lat/lon (unprojected/cylindrical equidistant)
# The proj.4 string:
# +proj=longlat +ellps=GRS80 +towgs84=0,0,0,0,0,0,0 +no_defs
```

Functions

In [ ]:
```
# some useful arrays to translate between naming conventions

election_year_list = np.array([1992, 1994, 1996, 1998, 2000, 2002, 2004, 2006, 20
                               2014, 2016, 2018])
congress_ID_list = np.array([103, 104, 105, 106, 107, 108, 109, 110, 111, 112, 11

state_names = np.array(['ALABAMA', 'ALASKA', 'ARIZONA', 'ARKANSAS', 'CALIFORNIA',
                'COLORADO', 'CONNECTICUT', 'DELAWARE', 'FLORIDA', 'GEORGIA',
                'HAWAII', 'IDAHO', 'ILLINOIS', 'INDIANA', 'IOWA', 'KANSAS',
                'KENTUCKY', 'LOUISIANA', 'MAINE', 'MARYLAND', 'MASSACHUSETTS',
                'MICHIGAN', 'MINNESOTA', 'MISSISSIPPI', 'MISSOURI', 'MONTANA',
                'NEBRASKA', 'NEVADA', 'NEW HAMPSHIRE', 'NEW JERSEY', 'NEW MEXICO',
                'NEW YORK', 'NORTH CAROLINA', 'NORTH DAKOTA', 'OHIO', 'OKLAHOMA',
                'OREGON', 'PENNSYLVANIA', 'RHODE ISLAND', 'SOUTH CAROLINA',
                'SOUTH DAKOTA', 'TENNESSEE', 'TEXAS', 'UTAH', 'VERMONT',
                'VIRGINIA', 'WASHINGTON', 'WEST VIRGINIA', 'WISCONSIN', 'WYOMING']

state_abbrs = np.array(['AL','AK','AZ','AR','CA','CO','CT','DE','FL','GA','HI','I
              'IN','IA','KS','KY','LA','ME','MD','MA','MI','MN','MS','MO','MT',
              'NE','NV','NH','NJ','NM','NY','NC','ND','OH','OK','OR','PA','RI',
              'SC','SD','TN','TX','UT','VT','VA','WA','WV','WI','WY'])

state_fips = np.array([1, 2, 4, 5, 6, 8, 9, 10, 12, 13, 15, 16, 17, 18, 19, 20, 2
                       24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38
                       41, 42, 44, 45, 46, 47, 48, 49, 50, 51, 53, 54, 55, 56])
```

In [ ]:

```
def read_shapefiles(election_years,verbose=True):
    """
    Reads in shapefiles from UCLA database (pre-2016) US Census TigerLine files
    data sources: http://cdmaps.polisci.ucla.edu/
                  https://www.census.gov/geo/maps-data/data/cbf/cbf_cds.html
    Note that file names must be in the same folder as the code with the followi
        -pre-2016: districtShapes-NNN/districtsNNN.shp
        -2016 on: must be named t1_YYYY_us_cdNNN/tl_YYYY_us_cdNNN.shp
    ...where NNN is congress ID and YYYY is election year

    input:
        election_years -- (list) Elections years of the shape files you want to
        verbose -- (bool) set to True if you want it to print every file as it r
    output:
        district_df -- pandas data frame with index in format ST_00_YYYY (e.g. A
                       shapefiles stored in a column named 'shape'.
    """
    # read in the standard dictionary
    district_df = pickle.load(open('../Datasets/master_index.p','rb'))
    district_df['shape'] = [np.nan]*district_df.shape[0] # make a blank column
    district_df['shape'] = district_df['shape'].astype(object) # reassign to obj

    for election_year in election_years:
        # convert election year to "Nth Congress"
        congress_ID = congress_ID_list[election_year_list==election_year][0]

        # pre-2016 files come from http://cdmaps.polisci.ucla.edu/
        # they are a bit cleaner
        if election_year < 2016:
            # read in the shapefile (must be named 'districtsN.shp' in a folder
            shpfilename = 'districtShapes-{0}/districts{0}.shp'.format(congress_
            reader = shpreader.Reader(shpfilename)
            districts = reader.records() # get full records
            geometries = reader.geometries() # get just the shape

            # put the shapefiles into the standard dictionary
            for record in reader.records(): # loop over districts
                attr = record.attributes # dictionary of information about the d
                poly = record.geometry # coordinates of the district as a shapel

                # 1) get the state abbr of the district
                if any(state_names==attr['STATENAME'].upper()): # filter out dis
                    ST = state_abbrs[state_names==attr['STATENAME'].upper()][0]
                else: # pretty much just Washington, DC
                    print('{} is not a state.'.format(attr['STATENAME'].upper())
                    continue
                # 2) get the id of the district
                id_int = int(attr['DISTRICT'])
                if id_int == 0: # change at-large district ID from 0 to 1 to pla
                    id_int = 1 # todo: do we want to change this back? Looks lik
                ID = '{0:02d}'.format(id_int)
                # 3) reformat the polygon if it intersects with itself
                if not poly.is_valid:
                    # if it does, use buffer to correct this
                    if verbose:
                        print('The following polygons intersected with themselve
                        print(ind)
                    poly = poly.buffer(0)
                # 4) put it all together into an index
                ind = '{}_{}_{}'.format(ST, ID, election_year)
                if verbose:
                    print('{} was read in.'.format(ind))
                # put the polygon in the dictionary
                district_df.at[ind,'shape'] = poly
```

In [ ]:

```
# compute overlap percent between this district and last year's districts
def district_overlap(this_year, district_df, threshold_for_change=0.1):
    """
    Finds the fractional overlap between this year's district and the previous y
    i.e., if a Florida's 1st district has changed its borders from 2014 to 2016,
        20% of its area may come from district 2 in 2014
        40% of its area may come from district 3 in 2014
        10% of its area may come from district 4 in 2014

    input:
        this_year -- (int) Elections years of the districts you want to check. E
        district_df -- (pd dataframe) dataframe with default indices and shapefi
                       a column named 'shape'
    output:
        district_df -- pandas data frame with index in format ST_00_YYYY (e.g. A

                       fractional overlap stored in a column called 'fractional_
                       Each district has a dictionary where the keys are the ind
                       districts which overlap with our district, and the values

                       change from last year stored in a column called 'border_c
                       Statuses:
                           0 - this district has not changed at all since the pr
                           1 - this district has either changed area > threshold
                               the previous year, or it is new.
    """
    # add a column
    if 'overlap_frac' not in district_df.columns:
        district_df['overlap_frac'] = [np.nan]*district_df.shape[0] # make a bla
        district_df['overlap_frac'] = district_df['overlap_frac'].astype(object)

    if 'border_change' not in district_df.columns:
        district_df['border_change'] = [np.nan]*district_df.shape[0] # make a bl

    # loop over states so you only have to compare districts in-state
    # otherwise, comparing each district to 434 other districts would be super s
    for ST in state_abbrs:
        prev_year = this_year-2
        # get the relevant districts
        districts = district_df.loc[np.logical_and(district_df['state']==ST,
                                                   district_df['year']==this_year)
        districts_prev = district_df.loc[np.logical_and(district_df['state']==ST
                                                   district_df['year']==prev_year)

        for ind,district in districts.iterrows(): # loop over districts in your
            overlap_dict = {}
            shape = district['shape']
            area = shape.area # area of this district

            # check if shapes intersect with themselves
            if not shape.is_valid:
                # if they do, use buffer to correct this
                print('The following polygons intersected with themselves. Attem
                print(ind)
                shape = shape.buffer(0)

            for ind_prev,district_prev in districts_prev.iterrows(): # loop over
                shape_prev = district_prev['shape']

                # check if shapes intersect with themselves
                if not shape_prev.is_valid:
                    # if they do, use buffer to correct this
                    print('The following polygons intersected with themselves. A
                    print(ind_prev)
```

In [ ]:

```
def get_centroid(district_df):
    """
    Finds the centroid of a district in lon,lat.

    input:
        district_df -- (pd dataframe) dataframe with default indices and shapefil
                       a column named 'shape'
    output:
        district_df -- pandas data frame with index in format ST_00_YYYY (e.g. AI
                       with centroid stored in a column called 'centroid'.
                       The centroid is calculated as a lon,lat on a Cartesian pla
                       It ignores spherical geometry.
                       The centroid is stored as a tuple in the form (lon,lat)
    """
    # add column
    district_df['centroid'] = [np.nan]*district_df.shape[0] # make a blank columr
    district_df['centroid'] = district_df['centroid'].astype(object) # reassign t

    for ind, district in district_df.iterrows():
        shape = district['shape']
        if pd.isnull(district['shape']): # if there's no shape, fill it with a na
            district_df.at[ind, 'centroid'] = np.nan
        else:
            centroid = shape.centroid.coords
            district_df.at[ind, 'centroid'] = centroid # in units of lat/lon

    return district_df
```

In [ ]:

```
# compute overlap percent between this district and last year's districts
def population_overlap(this_year, district_df, threshold_for_change=0.1):
    """
    Estimates percent of population coming from previous district boundaries usir
    the equation:

    population_overlap = (overlap_area / area) * (1 / area_prev)

    Where overlap_area is the area of overlap between the new and old districts,
    area is the area of the new district, and area_prev is the area of the previc
    district.

    This equation assumes that the voting population in each district is constant
    but area changes. Although each district has roughly 711,000 people, variatic
    voter turnout between districts means that this assumption is false.

    density_weighted_overlap is calculated for each previous district which inter
    the new district. density_weigted_overlap is then scaled so that the sum of a
    previous districts which intersect with the new district is 1.

    input:
        this_year -- (int) Elections years of the districts you want to check. Ev
        district_df -- (pd dataframe) dataframe with default indices and shapefil
                       a column named 'shape'
    output:
        district_df -- pandas data frame with index in format ST_00_YYYY (e.g. AI

                       population_overlap is stored in a column called 'populatic
                       Each district has a dictionary where the keys are the indi
                       districts which overlap with our district, and the values

    """

    # add a column
    if 'population_overlap' not in district_df.columns:
        district_df['population_overlap'] = [np.nan]*district_df.shape[0] # make
        district_df['population_overlap'] = district_df['population_overlap'].ast

    # loop over states so you only have to compare districts in-state
    # otherwise, comparing each district to 434 other districts would be super sl
    for ST in state_abbrs:
        prev_year = this_year-2
        # get the relevant districts
        districts = district_df.loc[np.logical_and(district_df['state']==ST,
                                                   district_df['year']==this_year)]
        districts_prev = district_df.loc[np.logical_and(district_df['state']==ST,
                                                   district_df['year']==prev_year)]

        for ind,district in districts.iterrows(): # loop over districts in your c
            overlap_dict = {}
            shape = district['shape']
            area = shape.area # area of this district

            # check if shapes intersect with themselves
            if not shape.is_valid:
                # if they do, use buffer to correct this
                print('The following polygons intersected with themselves. Attemp
                print(ind)
                shape = shape.buffer(0)

            for ind_prev,district_prev in districts_prev.iterrows(): # loop over
                shape_prev = district_prev['shape']

                # check if shapes intersect with themselves
```

In [ ]:

```
def inverse_population_overlap(this_year, district_df):
    """
    Exactly the same as population_overlap, except it gives each district_prev a
    where the keys are current year's districts, and the values are the estimated
    population overlap of district_prev and district.

    This allows you to impute values from (year) into (year-2) in your dataset.

    """

    # add a column
    if 'inverse_population_overlap' not in district_df.columns:
        district_df['inverse_population_overlap'] = [np.nan]*district_df.shape[0]
        district_df['inverse_population_overlap'] = district_df['inverse_populati

    # loop over states so you only have to compare districts in-state
    # otherwise, comparing each district to 434 other districts would be super sl
    for ST in state_abbrs:
        prev_year = this_year-2
        # get the relevant districts
        districts = district_df.loc[np.logical_and(district_df['state']==ST,
                                                   district_df['year']==this_year)]
        districts_prev = district_df.loc[np.logical_and(district_df['state']==ST,
                                                   district_df['year']==prev_year)]

        for ind_prev,district_prev in districts_prev.iterrows(): # loop over dist
            overlap_dict = {}
            shape_prev = district_prev['shape']
            area_prev = shape_prev.area # area of this district

            # check if shapes intersect with themselves
            if not shape_prev.is_valid:
                # if they do, use buffer to correct this
                print('The following polygons intersected with themselves. Attemp
                print(ind_prev)
                shape_prev = shape_prev.buffer(0)

            for ind,district in districts.iterrows(): # loop over districts in yo
                shape = district['shape']

                # check if shapes intersect with themselves
                if not shape.is_valid:
                    # if they do, use buffer to correct this
                    print('The following polygons intersected with themselves. At
                    print(ind)
                    shape = shape.buffer(0)

                # calculate frac overlap
                area = shape.area # area of current district
                density = 1./area # assume population roughly the same, but area
                overlap_area = shape.intersection(shape_prev).area # area of over
                frac_overlap = overlap_area/area_prev # fractional overlap betwee
                                                                    # round to

                if frac_overlap > 10**-3: # use threshold of 0.1% to avoid trivia
                    population_overlap = frac_overlap*density
                    overlap_dict[ind] = population_overlap

            # make sure areas add up to 1.
            dict_sum = np.sum(list(overlap_dict.values()))
            # rescale so they add up to 1.
            overlap_dict = {key : np.around(val/dict_sum, decimals=3)
                            for key,val in overlap_dict.items() if val/dict_sum >

            print(ind_prev)
```

In [ ]:

```
# find if a district has changed between last year and this year
def check_if_districts_changed(this_year, district_df, threshold_for_change=0.1):
    """
    *** DEPRECIATED ***
    might have some bugs.
    use district_overlap to perform this function instead.

    Checks if each district changed more than the set threshold since the last el
    Change is a fraction of the district which overlaps with the previous year of
    If the total area of the district increases, the overlap area is divided by t
        (this prevents counting a district as unchanged if its area has increasec

    input:
        this_year -- (int) Elections years of the districts you want to check. Ev
        district_df -- (pd dataframe) dataframe with default indices and shapefil
                       a column named 'shape'
    output:
        district_df -- pandas data frame with index in format ST_00_YYYY (e.g. AI
                       with change from last year stored in a column called 'borc
                       Statuses:
                           'same' - this district has not changed at all since th
                           'new' - indicates a district with this ID was not in t
                           'changed' - indicates the borders have changed from la
                                       this district was present in its state las
    """
    # loop over states so you only have to compare districts in-state
    # otherwise, comparing each district to 434 other districts would be super sl
    for ST in state_abbrs:
        prev_year = this_year-2
        # get the relevant districts
        districts = district_df.loc[np.logical_and(district_df['state']==ST,
                                                   district_df['year']==this_year)]
        districts_prev = district_df.loc[np.logical_and(district_df['state']==ST,
                                                   district_df['year']==prev_year)]
        # loop over districts in your current year
        for ind,district in districts.iterrows():
            # find previous year's district
            district_prev = districts_prev.loc[districts_prev['district']==distri
            ind_prev = '{}{}'.format(ind[:6],prev_year)

            # determine whether district is new or borders have changed
            if ind_prev in districts_prev.index: # if the district didn't exist l
                # then this district is new this year
                district_df.loc[ind,'border_change'] = 'new'
            else:
                # check if the borders have changed
                shape_prev = district_prev['shape'].values[0]
                shape = district['shape']

                # check if shapes intersect with themselves
                if not (shape.is_valid and shape_prev.is_valid):
                    # if they do, use buffer to correct this
                    print('The following polygons intersected with themselves. At
                    if not shape.is_valid:
                        print(ind)
                        shape = shape.buffer(0)
                    if not shape_prev.is_valid:
                        print(ind_prev)
                        shape_prev = shape_prev.buffer(0)

                # calculate overlap percent
                area = shape.area # area of this district
                area_prev = shape_prev.area
                overlap_area = shape.intersection(shape_prev).area # area of over
```

Run code here

In [ ]:
```
# get centroid coords for everything
years = [2002, 2004, 2006, 2008, 2010, 2012, 2014, 2016, 2018]

centroid_df = read_shapefiles(years, verbose=False)
for year in years:
     centroid_df = get_centroid(centroid_df)
pickle.dump(centroid_df.drop('shape',axis=1), open('centroid.p', 'wb'))
```

In [ ]:
```
# find out the area overlap and border change between this year's and last year's
# takes ~2hrs to run for all 8 years.

# you can only check a district if you have last year's data, so you read one mor
years_to_read = [2002, 2004, 2006, 2008, 2010, 2012, 2014, 2016, 2018]
years_to_check = [2004, 2006, 2008, 2010, 2012, 2014, 2016, 2018]

# read in the data (make a fresh df)
#overlap_df = read_shapefiles(years_to_read, verbose=False)
for year in years_to_check:
    overlap_df = district_overlap(year, overlap_df)
    pickle.dump(overlap_df.drop('shape', axis=1), open('overlap_frac_{}.p'.format
pickle.dump(overlap_df.drop('shape', axis=1), open('overlap_frac.p', 'wb'))
```

In [ ]:
```
# estimate % population which derives from each previous district

# you can only check a district if you have last year's data, so you read one mor
years_to_read = [2002, 2004, 2006, 2008, 2010, 2012, 2014, 2016, 2018]
years_to_check = [2004, 2006, 2008, 2010, 2012, 2014, 2016, 2018]

# read in the data (make a fresh df)
pop_df = read_shapefiles(years_to_read, verbose=False)
for year in years_to_check:
    pop_df = population_overlap(year, pop_df)
    pickle.dump(pop_df.drop('shape', axis=1), open('pop_frac_{}.p'.format(year),
pickle.dump(pop_df.drop('shape', axis=1), open('pop_frac.p', 'wb'))
```

In [ ]:
```
# Read in the files written here and add them to a master overlap set
all_overlap_data_df = pickle.load(open('overlap_frac.p','rb'))
all_overlap_data_df['centroid'] = pickle.load(open('centroid.p','rb'))['centroid'
all_overlap_data_df['population_overlap'] = \
    pickle.load(open('pop_frac.p','rb'))['population_overlap']
all_overlap_data_df['inverse_population_overlap'] = \
    pickle.load(open('inv_pop_frac.p','rb'))['inverse_population_overlap']
pickle.dump(all_overlap_data_df, open('all_overlap_data.p', 'wb'))
```

In [ ]:
```

```