replication

November 14, 2023

Machine Learning and Feature Selection: Applications in Economics and Climate Change

Berkay Akyapi

Replication of the Application presented in Section 4

This Jupyter notebook replicates the application presented in Section 4 of the paper. It consists of the following steps:

1. Downloading US county-level daily temperatures between 1979 - 2019 from ERA5.
2. Generating county-level limits for different definitions of heatwaves.
3. Generating heatwave information.
4. Performing feature selection using Group LASSO, Sparse Group LASSO, and LASSO.
5. Presenting regression results and generating Figure 4.

Note on Dependencies and Environments

- Some required packages are imported as needed during the execution.
- To prevent potential inconsistencies with the "asgl" package and its dependencies (especially with linearmodels), it is suggested to download that package and its dependencies in a separate Python environment. Run the relevant code for Group LASSO and Sparse Group LASSO in that environment.
- Files are saved after each section, so there is no need to repeat those sections again.
- The initial part until heatwave generation can take a while to complete.
- Adjust paths to save/read files as necessary.

1 Downloading Daily Temperature Data for US Counties

[ ]:
```
import ee
import geemap
import os

print('### Geemap version: ' + geemap.__version__) # 0.22.1
```

[ ]:
```
# used to print in bold text (or other colors if needed)
class myColor:
   PURPLE = '\033[95m'
   CYAN = '\033[96m'
   DARKCYAN = '\033[36m'
   BLUE = '\033[94m'
```

```
    GREEN = '\033[92m'
    YELLOW = '\033[93m'
    RED = '\033[91m'
    BOLD = '\033[1m'
    UNDERLINE = '\033[4m'
    END = '\033[0m'
```

[ ]:
```
parentPath = os.path.split(os.path.abspath(os.getcwd()))[0]
print(parentPath)
```

[ ]:
```
#ee.Authenticate()
ee.Initialize()
```

[ ]:
```
# county boundaries
counties = ee.FeatureCollection("TIGER/2018/Counties")
```

[ ]:
```
out_dir = os.path.join(parentPath, 'Data', 'US_County_T') # Adjust the address␣
 ↪accordingly
year = 1979
for i in range(2021-year):
    for j in range(1,13):
        if j ==12:
            theDateBegin = str(year) + '-' + str(j) + '-01'
            theDateEnd = str(year+1) + '-01-01'
        else:
            theDateBegin = str(year) + '-' + str(j) + '-01'
            theDateEnd = str(year) + '-' + str(j+1) + '-01'
        print('')
        print('Dates are ', theDateBegin, 'to', theDateEnd)
        print('')

        # Temperature Data (Mean)
        docName = 'meanDailyTemp' + str(year) + '_' + str(j) + '.csv'
        print('Document Name is ', myColor.BOLD + docName + myColor.END)
        print('')
        era5_2mt = ee.ImageCollection('ECMWF/ERA5/DAILY').filter(ee.Filter.
 ↪date(theDateBegin, theDateEnd)).select('mean_2m_air_temperature')
        out_landstat_stats = os.path.join(out_dir, docName)
        geemap.zonal_statistics(era5_2mt, counties, out_landstat_stats,␣
 ↪statistics_type='MEAN', scale = 5000)

        # Temperature Data (Min)
        docName = 'minDailyTemp' + str(year) + '_' + str(j) + '.csv'
        print('')
        print('Document Name is ', myColor.BOLD + docName + myColor.END)
        print('')
```

```
        era5_2mt = ee.ImageCollection('ECMWF/ERA5/DAILY').filter(ee.Filter.
↪date(theDateBegin, theDateEnd)).select('minimum_2m_air_temperature')
        out_landstat_stats = os.path.join(out_dir, docName)
        geemap.zonal_statistics(era5_2mt, counties, out_landstat_stats,␣
↪statistics_type='MEAN', scale = 5000)

        # Temperature Data (Max)
        docName = 'maxDailyTemp' + str(year) + '_' + str(j) + '.csv'
        print('')
        print('Document Name is ', myColor.BOLD + docName + myColor.END)
        print('')
        era5_2mt = ee.ImageCollection('ECMWF/ERA5/DAILY').filter(ee.Filter.
↪date(theDateBegin, theDateEnd)).select('maximum_2m_air_temperature')
        out_landstat_stats = os.path.join(out_dir, docName)
        geemap.zonal_statistics(era5_2mt, counties, out_landstat_stats,␣
↪statistics_type='MEAN', scale = 5000)

    year = year + 1
```

2 Reading and Merging Daily Temperature Data and Calculation of Heatwaves

[ ]:
```
import sys
import os

import numpy as np
import pandas as pd
import numpy.matlib

import statsmodels.api as sm

import time

from joblib import Parallel, delayed
import multiprocessing

# from linearmodels.panel import FirstDifferenceOLS
# from linearmodels.panel import PanelOLS

# Check software versions
print('### Python version: ' + __import__('sys').version) # 3.9.12
print('### NumPy version: ' + np.__version__) # 1.22.3
print('------------')
```

Function to Read the Files

```
[ ]: def genData(dummyData, months, theDir, docName, theTemp, colInd, varname):

    theData = pd.DataFrame(index=None)
    theDataFebruary29 = pd.DataFrame()

    year = 1979
    for m in range(2020-year):
        # More than 35C
        myData2 = pd.DataFrame(index = None)
        myData2['County'] = dummyData['NAME'].values
        myData2['CountyFP'] = dummyData['COUNTYFP'].values
        myData2['StateFP'] = dummyData['STATEFP'].values


        for i, myMo in enumerate(months):

            if year < 2020 or i < 6: # No Data after June for 2020
                docNameMore = docName + str(year) + '_' + str(i+1) + '.csv'
                myData1 = pd.read_csv(theDir + docNameMore)
                myData1 = myData1.drop(['GEOID', 'LSAD', 'CBSAFP', 'CSAFP',␣
↪'FUNCSTAT', 'INTPTLAT',
                                        'INTPTLON', 'COUNTYNS', 'MTFCC', 'system:
↪index', 'METDIVFP', 'NAMELSAD',
                                        'METDIVFP', 'AWATER',␣
↪'ALAND','COUNTYFP', 'STATEFP', 'CLASSFP', 'NAME'], axis=1)

                if m == 0 and i == 0: # No observation for January 1, 1979:␣
↪Replace with mean
                    myData1.insert(0, (str(year) + '0101_' + theTemp + varname),
                                   myData1.copy().mean(axis=1))

                if i+1 == 2:
                    colNamesMore = [myMo + str(j) + colInd for j in range(1,29)]
                    if year == 1980 or year == 1984 or year == 1988 or year ==␣
↪1992 or year == 1996 or year == 2000 or year == 2004 or year == 2008 or year␣
↪== 2012 or year == 2016 or year == 2020:

                        myString = str(year) + '0229_' + theTemp + varname
                        theMore = pd.DataFrame()
                        theMore['Feb29' + colInd] = myData1[myString]
                        theMore['Year'] = year
                        theMore['County'] = dummyData['NAME'].values
                        theDataFebruary29 = pd.concat([theDataFebruary29,␣
↪theMore], axis = 0)
                        myData1 = myData1.drop(myString, axis=1)

                elif i+1 == 4 or i+1 == 6 or i+1 == 9 or i+1 == 11:
```

```
                    colNamesMore = [myMo + str(j) + colInd for j in range(1,31)]
                else:
                    colNamesMore = [myMo + str(j) + colInd for j in range(1,32)]


                #colNamesMore.append('NAME')
                myData1.columns = colNamesMore
                myData2 = pd.concat([myData2, myData1], axis=1)

            else:
                docNameMore = docName + str(year-1) + '_' + str(i+1) + '.csv'
                myData1 = pd.read_csv(theDir + docNameMore)
                myData1 = myData1.drop(['GEOID', 'LSAD', 'CBSAFP', 'CSAFP',␣
↪'FUNCSTAT', 'INTPTLAT',
                                        'INTPTLON', 'COUNTYNS', 'MTFCC', 'system:
↪index', 'METDIVFP', 'NAMELSAD',
                                        'METDIVFP', 'AWATER',␣
↪'ALAND','COUNTYFP', 'STATEFP', 'CLASSFP', 'NAME'], axis=1)


                if i+1 == 4 or i+1 == 6 or i+1 == 9 or i+1 == 11:
                    colNamesMore = [myMo + str(j) + colInd for j in range(1,31)]
                else:
                    colNamesMore = [myMo + str(j) + colInd for j in range(1,32)]


                #colNamesMore.append('NAME')
                myData1.columns = colNamesMore
                myData2 = pd.concat([myData2, myData1], axis=1)

        myData2['Year'] = year
        year = year + 1

        theData = pd.concat([theData, myData2], axis = 0)

    return theData
```

Function to calcualte heatwaves given the limits

[ ]:
```
def heatwaves(limits, temperatures, consecutiveDays, data_loc, tracker = None,␣
↪verbose = 2):

    HWN = np.zeros([temperatures.shape[0], 1]) # Yearly Number of heat waves
    HWN_JFM = np.zeros([temperatures.shape[0], 1]) # Number of heat waves␣
↪January, February, March
    HWN_AMJ = np.zeros([temperatures.shape[0], 1]) # Number of heat waves␣
↪April, May, June
```

```
   HWN_JAS = np.zeros([temperatures.shape[0], 1]) # Number of heat waves␣
↪July,August, September

   HWD = np.zeros([temperatures.shape[0], 1]) # Length of the longest yearly␣
↪event

   HWF = np.zeros([temperatures.shape[0], 1]) # Yearly sum of participating␣
↪heat waves
   HWF_JFM = np.zeros([temperatures.shape[0], 1]) # Sum of participating heat␣
↪waves January, February, March
   HWF_AMJ = np.zeros([temperatures.shape[0], 1]) # Sum of participating heat␣
↪waves April, May, June
   HWF_JAS = np.zeros([temperatures.shape[0], 1]) # Sum of participating heat␣
↪waves July,August, September

   keepDays = np.zeros([temperatures.shape[0], temperatures.shape[1]]) # To␣
↪keep track of HWM
   keepDays_JFM = np.zeros([temperatures.shape[0], temperatures.shape[1]]) #␣
↪To keep track of HWM_JFM
   keepDays_AMJ = np.zeros([temperatures.shape[0], temperatures.shape[1]]) #␣
↪To keep track of HWM_AMJ
   keepDays_JAS = np.zeros([temperatures.shape[0], temperatures.shape[1]]) #␣
↪To keep track of HWM_JAS

   for i in range(0, temperatures.shape[0]):
       count = 0
       if tracker is not None:
           if verbose > 0:
               print(tracker.iloc[i, -1], tracker.iloc[i, 1])
              ␣
↪print('------------------------------------------------------------')
           if i == 0:
               myfile = open(os.path.join(data_loc, 'Heatwaves', 'trackFile_'␣
↪+ str(consecutiveDays) +'.txt'), 'w')
               myfile.write(tracker.iloc[i, -1] + ' ' + str(tracker.iloc[i,␣
↪1]) + ': ' + str(temperatures.shape[0]-i)  + ' iterations left\n')
               myfile.close()
           else:
               myfile = open(os.path.join(data_loc, 'Heatwaves', 'trackFile_'␣
↪+ str(consecutiveDays) +'.txt'), 'a')
               myfile.write(tracker.iloc[i, -1] + ' ' + str(tracker.iloc[i,␣
↪1]) + ': ' + str(temperatures.shape[0]-i)  + ' iterations left\n')
               myfile.close()


       for j in range(0,temperatures.shape[1]-1):
```

```
            if temperatures.iloc[i,j]>limits.iloc[i,j] and temperatures.
↪iloc[i,j+1]>limits.iloc[i,j+1]:
                count = count + 1
                if temperatures.columns[j+1] == 'Dec31TMax':
                    if consecutiveDays < (count+1):
                        HWF[i,0] =  HWF[i,0] + count + 1 # For example, if␣
↪count = 1 then there were 2 consecutive days
                        HWN[i,0] = HWN[i,0] + 1
                        keepDays[i,j-count:j+2] = 1
                        if verbose > 1:
                            print('Last day is', temperatures.columns[j+1], ';␣
↪consecutive days are:', (count+1),
                                  '; temperature is: {: .2f}'.
↪format(temperatures.iloc[i,j+1]),
                                  '; Limit is: {: .2f}'.format(limits.
↪iloc[i,j+1]),
                                 "({})".format(limits.columns[j+1]), "index", i)

                        if HWD[i,0] < (count+1):
                            HWD[i,0] = count + 1 # For example, if count = 1␣
↪then there were 2 consecutive days
                        count = 0
            elif count > 0 and temperatures.iloc[i,j]>limits.iloc[i,j]:
                if consecutiveDays < (count+1):
                    HWF[i,0] = HWF[i,0] + count + 1 # For example, if count = 1␣
↪then there were 2 consecutive days
                    HWN[i,0] = HWN[i,0] + 1
                    keepDays[i,j-count:j+1] = 1
                    if verbose > 1:
                        print('Last day is', temperatures.columns[j], ';␣
↪consecutive days are:', (count+1),
                                  '; temperature is: {: .2f}'.
↪format(temperatures.iloc[i,j]),
                                '; Limit is: {: .2f}'.format(limits.iloc[i,j]),
                                  "({})".format(limits.columns[j]), "index", i)

                    # Assign heatwave to season according to end date

                    if temperatures.columns[j][0:3] == "Jun" or temperatures.
↪columns[j][0:3] == "Apr" or temperatures.columns[j][0:3] == "May":
                            HWN_AMJ[i,0] = HWN_AMJ[i,0] + 1
                            HWF_AMJ[i,0] = HWF_AMJ[i,0] + count + 1
                            keepDays_AMJ[i,j-count:j+2] = 1
                    elif temperatures.columns[j][0:3] == "Sep" or temperatures.
↪columns[j][0:3] == "Jul" or temperatures.columns[j][0:3] == "Aug":
                            HWN_JAS[i,0] = HWN_JAS[i,0] + 1
```

```
                            HWF_JAS[i,0] = HWF_JAS[i,0] + count + 1
                            keepDays_JAS[i,j-count:j+2] = 1
                    elif temperatures.columns[j][0:3] == "Jan" or temperatures.
↪columns[j][0:3] == "Feb" or temperatures.columns[j][0:3] == "Mar":
                            HWN_JFM[i,0] = HWN_JFM[i,0] + 1
                            HWF_JFM[i,0] = HWF_JFM[i,0] + count + 1
                            keepDays_JFM[i,j-count:j+2] = 1

                    if HWD[i,0] < (count+1):
                        HWD[i,0] = count + 1 # For example, if count = 1 then␣
↪there were 2 consecutive days
                count = 0

        if verbose > 1:
            ␣
↪print('-----------------------------------------------------------------')
            print('Yearly Number of heat waves are ', HWN[i,0])
            print('Number of Heatwaves in JFM ', HWN_JFM[i,0])
            print('Number of Heatwaves in AMJ ', HWN_AMJ[i,0])
            print('Number of Heatwaves in JAS ', HWN_JAS[i,0])
            print('Length of the longest yearly event is ', HWD[i,0])
            print('Yearly sum of participating heat waves are ', HWF[i,0])
            ␣
↪print('-----------------------------------------------------------------')
            print('')

    a = np.sum(temperatures.values*keepDays, axis=1).reshape(-1,1) # keepdays␣
↪is 1 only for the days with participating heatwaves
    b = np.zeros([temperatures.shape[0], 1])
    for i in range(np.shape(a)[0]):
        if HWF[i,0] != 0:
            b[i,0] = a[i,0]/HWF[i,0]
        else: # if no heatwave observed, set the average to the average of max/
↪min temperature of that year, because "0" means something else
            b[i,0] = np.mean(temperatures.iloc[i, :].values)
    TDHW = b

    a = np.sum(temperatures.values*keepDays_JFM, axis=1).reshape(-1,1) #␣
↪keepdays is 1 only for the days with participating heatwaves
    b = np.zeros([temperatures.shape[0], 1])
    for i in range(np.shape(a)[0]):
        if HWF_JFM[i,0] != 0:
            b[i,0] = a[i,0]/HWF_JFM[i,0]
        else: # if no heatwave observed, set the average to the average of max/
↪min temperature of that year, because "0" means something else
```

```
            b[i,0] = np.mean(temperatures.iloc[i, 0:90].values) # January 1st␣
↪is the 1st observation, March 31st is 89th observation
    TDHW_JFM = b

    a = np.sum(temperatures.values*keepDays_AMJ, axis=1).reshape(-1,1) #␣
↪keepdays is 1 only for the days with participating heatwaves
    b = np.zeros([temperatures.shape[0], 1])
    for i in range(np.shape(a)[0]):
        if HWF_AMJ[i,0] != 0:
            b[i,0] = a[i,0]/HWF_AMJ[i,0]
        else: # if no heatwave observed, set the average to the average of max/
↪min temperature of that year, because "0" means something else
            b[i,0] = np.mean(temperatures.iloc[i, 90:181].values) # April 1st␣
↪is the 90th observation, June 30th is 180th observation
    TDHW_AMJ = b

    a = np.sum(temperatures.values*keepDays_JAS, axis=1).reshape(-1,1) #␣
↪keepdays is 1 only for the days with participating heatwaves
    b = np.zeros([temperatures.shape[0], 1])
    for i in range(np.shape(a)[0]):
        if HWF_JAS[i,0] != 0:
            b[i,0] = a[i,0]/HWF_JAS[i,0]
        else: # if no heatwave observed, set the average to the average of max/
↪min temperature of that year, because "0" means something else
            b[i,0] = np.mean(temperatures.iloc[i, 181:273].values) # July 1st␣
↪is the 181st observation, September 30th is 272nd observation
    TDHW_JAS = b


    return HWN, HWN_JFM, HWN_AMJ, HWN_JAS, HWF, HWF_JFM, HWF_AMJ, HWF_JAS,␣
↪TDHW, TDHW_JFM, TDHW_AMJ, TDHW_JAS, HWD
```

[3]:
```
data_loc =  os.path.join(os.path.split(os.path.abspath(os.getcwd()))[0],␣
↪'Data') # Adjust the address accordingly
print(data_loc)
data_loc_temp = os.path.join(data_loc, "US_County_T") # Adjust the address␣
↪accordingly
print(data_loc_temp)
```

[ ]:
```
months = ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep', 'Oct',␣
↪'Nov', 'Dec']
dummyData = pd.read_csv(os.path.join(data_loc_temp, "meanDailyTemp1980_1.csv"))

theTemp = 'mean'

docName = '/meanDailyTemp'
```

```
myDataTempMean = genData(dummyData, months, data_loc_temp, docName, theTemp,␣
 ↪'TMean', '_2m_air_temperature')

dummyData = pd.read_csv(os.path.join(data_loc_temp, "maxDailyTemp1980_1.csv"))

theTemp = 'maximum'

docName = '/maxDailyTemp'

myDataTempMax = genData(dummyData, months, data_loc_temp, docName, theTemp,␣
 ↪'TMax', '_2m_air_temperature')

dummyData = pd.read_csv(os.path.join(data_loc_temp, "minDailyTemp1980_1.csv"))

theTemp = 'minimum'

docName = '/minDailyTemp'

myDataTempMin = genData(dummyData, months, data_loc_temp, docName, theTemp,␣
 ↪'TMin', '_2m_air_temperature')
```

[ ]:
```
theTempMean = myDataTempMean.copy()
theTempMean = theTempMean.drop(['County','Year', 'StateFP', 'CountyFP'], axis =␣
 ↪1)
theTempMean = theTempMean.iloc[0:,0:] - 273.15 # Conver to celsius
theTempMean.insert(0,'CountyFP', myDataTempMean['CountyFP'])
theTempMean.insert(0, 'StateFP', myDataTempMean['StateFP'])
theTempMean.insert(0,'Year', myDataTempMean['Year'])
theTempMean.insert(0, 'County', myDataTempMean['County'])
theTempMean = theTempMean[theTempMean['Year']<2020]
theTempMean = theTempMean[theTempMean['Year']>1978]
theTempMean['County_StateFP'] = theTempMean['County'] + ' ' +␣
 ↪theTempMean['StateFP'].astype(str)

theTempMax = myDataTempMax.copy()
theTempMax = theTempMax.drop(['County','Year', 'StateFP', 'CountyFP'], axis = 1)
theTempMax = theTempMax.iloc[0:,0:] - 273.15 # Conver to celsius
theTempMax.insert(0,'CountyFP', myDataTempMax['CountyFP'])
theTempMax.insert(0, 'StateFP', myDataTempMax['StateFP'])
theTempMax.insert(0,'Year', myDataTempMax['Year'])
theTempMax.insert(0, 'County', myDataTempMax['County'])
theTempMax = theTempMax[theTempMax['Year']<2020]
theTempMax = theTempMax[theTempMax['Year']>1978]
theTempMax['County_StateFP'] = theTempMax['County'] + ' ' +␣
 ↪theTempMax['StateFP'].astype(str)
```

```
theTempMin = myDataTempMin.copy()
theTempMin = theTempMin.drop(['County','Year', 'StateFP', 'CountyFP'], axis = 1)
theTempMin = theTempMin.iloc[0:,0:] - 273.15 # Conver to celsius
theTempMin.insert(0,'CountyFP', myDataTempMin['CountyFP'])
theTempMin.insert(0, 'StateFP', myDataTempMin['StateFP'])
theTempMin.insert(0,'Year', myDataTempMin['Year'])
theTempMin.insert(0, 'County', myDataTempMin['County'])
theTempMin = theTempMin[theTempMin['Year']<2020]
theTempMin = theTempMin[theTempMin['Year']>1978]
theTempMin['County_StateFP'] = theTempMin['County'] + ' ' +␣
 ↪theTempMin['StateFP'].astype(str)

theTempMax = theTempMax.sort_values(by=['County_StateFP', 'Year'])
theTempMin = theTempMin.sort_values(by=['County_StateFP', 'Year'])
theTempMean = theTempMean.sort_values(by=['County_StateFP', 'Year'])

TempMax = theTempMax.drop(['County','Year', 'StateFP', 'CountyFP',␣
 ↪'County_StateFP'], axis = 1)
TempMin = theTempMin.drop(['County','Year', 'StateFP', 'CountyFP',␣
 ↪'County_StateFP'], axis = 1)
TempMean = theTempMean.drop(['County','Year', 'StateFP', 'CountyFP',␣
 ↪'County_StateFP'], axis = 1)
```

[ ]:
```
theTempMax_dum = theTempMax.drop(['County','Year', 'StateFP', 'CountyFP',␣
 ↪'County_StateFP'], axis = 1)
theTempMax_dum.insert(0, 'PreviousYearDec31', theTempMax_dum['Dec31TMax'].
 ↪shift())
theTempMax_dum.insert(0, 'PreviousYearDec30', theTempMax_dum['Dec30TMax'].
 ↪shift())
theTempMax_dum.insert(0, 'PreviousYearDec29', theTempMax_dum['Dec29TMax'].
 ↪shift())
theTempMax_dum.insert(0, 'PreviousYearDec28', theTempMax_dum['Dec28TMax'].
 ↪shift())
theTempMax_dum.insert(0, 'PreviousYearDec27', theTempMax_dum['Dec27TMax'].
 ↪shift())
theTempMax_dum.insert(0, 'PreviousYearDec26', theTempMax_dum['Dec26TMax'].
 ↪shift())
theTempMax_dum.insert(0, 'PreviousYearDec25', theTempMax_dum['Dec25TMax'].
 ↪shift())
theTempMax_dum['NextYearJan1'] = theTempMax_dum['Jan1TMax'].shift(-1)
theTempMax_dum['NextYearJan2'] = theTempMax_dum['Jan2TMax'].shift(-1)
theTempMax_dum['NextYearJan3'] = theTempMax_dum['Jan3TMax'].shift(-1)
theTempMax_dum['NextYearJan4'] = theTempMax_dum['Jan4TMax'].shift(-1)
theTempMax_dum['NextYearJan5'] = theTempMax_dum['Jan5TMax'].shift(-1)
theTempMax_dum['NextYearJan6'] = theTempMax_dum['Jan6TMax'].shift(-1)
theTempMax_dum['NextYearJan7'] = theTempMax_dum['Jan7TMax'].shift(-1)
```

```
theTempMax_dum.insert(0,'CountyFP', theTempMax['CountyFP'])
theTempMax_dum.insert(0, 'StateFP', theTempMax['StateFP'])
theTempMax_dum.insert(0,'Year', theTempMax['Year'])
theTempMax_dum.insert(0, 'County', theTempMax['County'])
theTempMax_dum['County_StateFP'] = theTempMax_dum['County'] + ' ' +␣
 ↪theTempMax_dum['StateFP'].astype(str)
theTempMax_dum = theTempMax_dum[theTempMax_dum['Year']<2020]
theTempMax_dum = theTempMax_dum[theTempMax_dum['Year']>1979]

theTempMin_dum = theTempMin.drop(['County','Year', 'StateFP', 'CountyFP',␣
 ↪'County_StateFP'], axis = 1)
theTempMin_dum.insert(0, 'PreviousYearDec31', theTempMin_dum['Dec31TMin'].
 ↪shift())
theTempMin_dum.insert(0, 'PreviousYearDec30', theTempMin_dum['Dec30TMin'].
 ↪shift())
theTempMin_dum.insert(0, 'PreviousYearDec29', theTempMin_dum['Dec29TMin'].
 ↪shift())
theTempMin_dum.insert(0, 'PreviousYearDec28', theTempMin_dum['Dec28TMin'].
 ↪shift())
theTempMin_dum.insert(0, 'PreviousYearDec27', theTempMin_dum['Dec27TMin'].
 ↪shift())
theTempMin_dum.insert(0, 'PreviousYearDec26', theTempMin_dum['Dec26TMin'].
 ↪shift())
theTempMin_dum.insert(0, 'PreviousYearDec25', theTempMin_dum['Dec25TMin'].
 ↪shift())
theTempMin_dum['NextYearJan1'] = theTempMin_dum['Jan1TMin'].shift(-1)
theTempMin_dum['NextYearJan2'] = theTempMin_dum['Jan2TMin'].shift(-1)
theTempMin_dum['NextYearJan3'] = theTempMin_dum['Jan3TMin'].shift(-1)
theTempMin_dum['NextYearJan4'] = theTempMin_dum['Jan4TMin'].shift(-1)
theTempMin_dum['NextYearJan5'] = theTempMin_dum['Jan5TMin'].shift(-1)
theTempMin_dum['NextYearJan6'] = theTempMin_dum['Jan6TMin'].shift(-1)
theTempMin_dum['NextYearJan7'] = theTempMin_dum['Jan7TMin'].shift(-1)
theTempMin_dum.insert(0,'CountyFP', theTempMin['CountyFP'])
theTempMin_dum.insert(0, 'StateFP', theTempMin['StateFP'])
theTempMin_dum.insert(0,'Year', theTempMin['Year'])
theTempMin_dum.insert(0, 'County', theTempMin['County'])
theTempMin_dum['County_StateFP'] = theTempMin_dum['County'] + ' ' +␣
 ↪theTempMin_dum['StateFP'].astype(str)
theTempMin_dum = theTempMin_dum[theTempMin_dum['Year']<2020]
theTempMin_dum = theTempMin_dum[theTempMin_dum['Year']>1979]

print(theTempMin_dum.shape)
iteratorTempMin = theTempMin_dum.drop(['County','Year', 'StateFP', 'CountyFP',␣
 ↪'County_StateFP'],axis=1)
print(iteratorTempMin.shape)
print(theTempMax_dum.shape)
```

```
iteratorTempMax = theTempMax_dum.drop(['County','Year', 'StateFP', 'CountyFP',␣
 ↪'County_StateFP'],axis=1)
print(iteratorTempMax.shape)
```

[ ]:
```
############### 90th percentile ####################
# 15 day window moving 20-year average tracker
Tmin90Perc_15 = pd.DataFrame()
Tmax90Perc_15 = pd.DataFrame()
# 5 day window moving 20-year average tracker
Tmin90Perc_5 = pd.DataFrame()
Tmax90Perc_5 = pd.DataFrame()
# 15 day window 20-year no moving average tracker
Tmin90Perc_15_noMov = pd.DataFrame()
Tmax90Perc_15_noMov = pd.DataFrame()
# 5 day window 20-year no moving average tracker
Tmin90Perc_5_noMov = pd.DataFrame()
Tmax90Perc_5_noMov = pd.DataFrame()
############### 95th percentile ####################
# 15 day window moving 20-year average tracker
Tmin95Perc_15 = pd.DataFrame()
Tmax95Perc_15 = pd.DataFrame()
# 5 day window moving 20-year average tracker
Tmin95Perc_5 = pd.DataFrame()
Tmax95Perc_5 = pd.DataFrame()
# 15 day window 20-year no moving average tracker
Tmin95Perc_15_noMov = pd.DataFrame()
Tmax95Perc_15_noMov = pd.DataFrame()
# 5 day window 20-year no moving average tracker
Tmin95Perc_5_noMov = pd.DataFrame()
Tmax95Perc_5_noMov = pd.DataFrame()


CountyNames = theTempMin_dum['County_StateFP'].unique()

num_iter = CountyNames.shape[0]

for index, c in enumerate(CountyNames):
    print(c, num_iter, 'iterations left')
    if index == 0:
        myfile = open(os.path.join(data_loc, 'Heatwave_Limits', 'myFile.txt'),␣
 ↪'w')
        myfile.write(c + ' ' + str(num_iter) +  ' iterations left\n')
        myfile.close()
    else:
        myfile = open(os.path.join(data_loc, 'Heatwave_Limits', 'myFile.txt'),␣
 ↪"a")  # append mode
        myfile.write(c + ' ' + str(num_iter) +  ' iterations left\n')
```

```
        myfile.close()

    num_iter -= 1
    ################ 90th percentile ####################
    # 15 day window moving 20-year
    temporary90min_15 = iteratorTempMin[theTempMin_dum['County_StateFP']== c].
↪copy()
    temporary90max_15 = iteratorTempMax[theTempMax_dum['County_StateFP']== c].
↪copy()
    # 5 day window moving 20-year
    temporary90min_5 = iteratorTempMin[theTempMin_dum['County_StateFP']== c].
↪copy()
    temporary90max_5 = iteratorTempMax[theTempMax_dum['County_StateFP']== c].
↪copy()
    # 15 day window not moving 20-year
    temporary90min_15_noMov =␣
↪iteratorTempMin[theTempMin_dum['County_StateFP']== c].copy()
    temporary90max_15_noMov =␣
↪iteratorTempMax[theTempMax_dum['County_StateFP']== c].copy()
    # 5 day window not moving 20-year
    temporary90min_5_noMov = iteratorTempMin[theTempMin_dum['County_StateFP']==␣
↪c].copy()
    temporary90max_5_noMov = iteratorTempMax[theTempMax_dum['County_StateFP']==␣
↪c].copy()
    ################ 95th percentile ####################
    # 15 day window moving 20-year
    temporary95min_15 = iteratorTempMin[theTempMin_dum['County_StateFP']== c].
↪copy()
    temporary95max_15 = iteratorTempMax[theTempMax_dum['County_StateFP']== c].
↪copy()
    # 5 day window moving 20-year
    temporary95min_5 = iteratorTempMin[theTempMin_dum['County_StateFP']== c].
↪copy()
    temporary95max_5 = iteratorTempMax[theTempMax_dum['County_StateFP']== c].
↪copy()
    # 15 day window not moving 20-year
    temporary95min_15_noMov =␣
↪iteratorTempMin[theTempMin_dum['County_StateFP']== c].copy()
    temporary95max_15_noMov =␣
↪iteratorTempMax[theTempMax_dum['County_StateFP']== c].copy()
    # 5 day window not moving 20-year
    temporary95min_5_noMov = iteratorTempMin[theTempMin_dum['County_StateFP']==␣
↪c].copy()
    temporary95max_5_noMov = iteratorTempMax[theTempMax_dum['County_StateFP']==␣
↪c].copy()
```

```
  uniqueCountyTmin = iteratorTempMin[theTempMin_dum['County_StateFP']==c].
↪copy()
  uniqueCountyTmax = iteratorTempMax[theTempMax_dum['County_StateFP']==c].
↪copy()
  for i in range(7, uniqueCountyTmin.shape[1]-7):
      for j in range(20,uniqueCountyTmin.shape[0]):
          # 90th percentile, 15-day window, 20-year moving average
          temporary90min_15.iloc[j,i] = np.percentile(np.
↪array(uniqueCountyTmin.iloc[(j-20):j,i-7:i+8]),90)
          temporary90max_15.iloc[j,i] = np.percentile(np.
↪array(uniqueCountyTmax.iloc[(j-20):j,i-7:i+8]),90)
          # 95th percentile, 15-day window, 20-year moving average
          temporary95min_15.iloc[j,i] = np.percentile(np.
↪array(uniqueCountyTmin.iloc[(j-20):j,i-7:i+8]),95)
          temporary95max_15.iloc[j,i] = np.percentile(np.
↪array(uniqueCountyTmax.iloc[(j-20):j,i-7:i+8]),95)
          # 90th percentile, 15-day window, not moving 20-year
          temporary90min_15_noMov.iloc[j,i] = np.percentile(np.
↪array(uniqueCountyTmin.iloc[0:20,i-7:i+8]),90)
          temporary90max_15_noMov.iloc[j,i] = np.percentile(np.
↪array(uniqueCountyTmax.iloc[0:20,i-7:i+8]),90)
          # 95th percentile, 15-day window, not moving 20-year
          temporary95min_15_noMov.iloc[j,i] = np.percentile(np.
↪array(uniqueCountyTmin.iloc[0:20,i-7:i+8]),95)
          temporary95max_15_noMov.iloc[j,i] = np.percentile(np.
↪array(uniqueCountyTmax.iloc[0:20,i-7:i+8]),95)

          # 90th percentile, 5-day window, 20-year moving average
          temporary90min_5.iloc[j,i] = np.percentile(np.
↪array(uniqueCountyTmin.iloc[(j-20):j,i-7+5:i+8-5]),90)
          temporary90max_5.iloc[j,i] = np.percentile(np.
↪array(uniqueCountyTmax.iloc[(j-20):j,i-7+5:i+8-5]),90)
          # 95th percentile, 5-day window, 20-year moving average
          temporary95min_5.iloc[j,i] = np.percentile(np.
↪array(uniqueCountyTmin.iloc[(j-20):j,i-7+5:i+8-5]),95)
          temporary95max_5.iloc[j,i] = np.percentile(np.
↪array(uniqueCountyTmax.iloc[(j-20):j,i-7+5:i+8-5]),95)
          # 90th percentile, 5-day window, not moving 20-year
          temporary90min_5_noMov.iloc[j,i] = np.percentile(np.
↪array(uniqueCountyTmin.iloc[0:20,i-7+5:i+8-5]),90)
          temporary90max_5_noMov.iloc[j,i] = np.percentile(np.
↪array(uniqueCountyTmax.iloc[0:20,i-7+5:i+8-5]),90)
          # 95th percentile, 5-day window, not moving 20-year
          temporary95min_5_noMov.iloc[j,i] = np.percentile(np.
↪array(uniqueCountyTmin.iloc[0:20,i-7+5:i+8-5]),95)
```

```
            temporary95max_5_noMov.iloc[j,i] = np.percentile(np.
 ↪array(uniqueCountyTmax.iloc[0:20,i-7+5:i+8-5]),95)

    ################ 90th percentile ####################
    Tmin90Perc_15 = pd.concat([Tmin90Perc_15, temporary90min_15], axis = 0)
    Tmax90Perc_15 = pd.concat([Tmax90Perc_15, temporary90max_15], axis = 0)
    Tmin90Perc_5 = pd.concat([Tmin90Perc_5, temporary90min_5], axis = 0)
    Tmax90Perc_5 = pd.concat([Tmax90Perc_5, temporary90max_5], axis = 0)
    Tmin90Perc_15_noMov = pd.concat([Tmin90Perc_15_noMov,␣
 ↪temporary90min_15_noMov], axis = 0)
    Tmax90Perc_15_noMov = pd.concat([Tmax90Perc_15_noMov,␣
 ↪temporary90max_15_noMov], axis = 0)
    Tmin90Perc_5_noMov = pd.concat([Tmin90Perc_5_noMov,␣
 ↪temporary90min_5_noMov], axis = 0)
    Tmax90Perc_5_noMov = pd.concat([Tmax90Perc_5_noMov,␣
 ↪temporary90max_5_noMov], axis = 0)
    ################ 95th percentile ####################
    Tmin95Perc_15 = pd.concat([Tmin95Perc_15, temporary95min_15], axis = 0)
    Tmax95Perc_15 = pd.concat([Tmax95Perc_15, temporary95max_15], axis = 0)
    Tmin95Perc_5 = pd.concat([Tmin95Perc_5, temporary95min_5], axis = 0)
    Tmax95Perc_5 = pd.concat([Tmax95Perc_5, temporary95max_5], axis = 0)
    Tmin95Perc_15_noMov = pd.concat([Tmin95Perc_15_noMov,␣
 ↪temporary95min_15_noMov], axis = 0)
    Tmax95Perc_15_noMov = pd.concat([Tmax95Perc_15_noMov,␣
 ↪temporary95max_15_noMov], axis = 0)
    Tmin95Perc_5_noMov = pd.concat([Tmin95Perc_5_noMov,␣
 ↪temporary95min_5_noMov], axis = 0)
    Tmax95Perc_5_noMov = pd.concat([Tmax95Perc_5_noMov,␣
 ↪temporary95max_5_noMov], axis = 0)
```

[ ]:
```
dumCols = ['PreviousYearDec25', 'PreviousYearDec26',␣
 ↪'PreviousYearDec27','PreviousYearDec28',
           'PreviousYearDec29', 'PreviousYearDec30',␣
 ↪'PreviousYearDec31','NextYearJan1', 'NextYearJan2',
           'NextYearJan3','NextYearJan4','NextYearJan5','NextYearJan6',␣
 ↪'NextYearJan7']

Tmin90Perc_15 = Tmin90Perc_15.drop(dumCols, axis=1)
Tmax90Perc_15 = Tmax90Perc_15.drop(dumCols, axis=1)
Tmin90Perc_5 = Tmin90Perc_5.drop(dumCols, axis=1)
Tmax90Perc_5 = Tmax90Perc_5.drop(dumCols, axis=1)
Tmin90Perc_15_noMov = Tmin90Perc_15_noMov.drop(dumCols, axis=1)
Tmax90Perc_15_noMov = Tmax90Perc_15_noMov.drop(dumCols, axis=1)
Tmin90Perc_5_noMov = Tmin90Perc_5_noMov.drop(dumCols, axis=1)
Tmax90Perc_5_noMov = Tmax90Perc_5_noMov.drop(dumCols, axis=1)
```

```
Tmin95Perc_15 = Tmin95Perc_15.drop(dumCols, axis=1)
Tmax95Perc_15 = Tmax95Perc_15.drop(dumCols, axis=1)
Tmin95Perc_5 = Tmin95Perc_5.drop(dumCols, axis=1)
Tmax95Perc_5 = Tmax95Perc_5.drop(dumCols, axis=1)
Tmin95Perc_15_noMov = Tmin95Perc_15_noMov.drop(dumCols, axis=1)
Tmax95Perc_15_noMov = Tmax95Perc_15_noMov.drop(dumCols, axis=1)
Tmin95Perc_5_noMov = Tmin95Perc_5_noMov.drop(dumCols, axis=1)
Tmax95Perc_5_noMov = Tmax95Perc_5_noMov.drop(dumCols, axis=1)

myfile = open(os.path.join(data_loc, 'Heatwave_Limits', 'myFile.txt'), "a")  #␣
 ↪append mode
myfile.write("Dummy Columns Dropped \n")
myfile.close()

datasets = [Tmin90Perc_15, Tmax90Perc_15, Tmin90Perc_5, Tmax90Perc_5,
           Tmin90Perc_15_noMov, Tmax90Perc_15_noMov, Tmin90Perc_5_noMov,␣
 ↪Tmax90Perc_5_noMov,
           Tmin95Perc_15, Tmax95Perc_15, Tmin95Perc_5, Tmax95Perc_5,
           Tmin95Perc_15_noMov, Tmax95Perc_15_noMov, Tmin95Perc_5_noMov,␣
 ↪Tmax95Perc_5_noMov]

docNames = ['CTN90pct_15', 'CTX90pct_15', 'CTN90pct_5', 'CTX90pct_5',
           'CTN90pct_15_noMov', 'CTX90pct_15_noMov', 'CTN90pct_5_noMov',␣
 ↪'CTX90pct_5_noMov',
           'CTN95pct_15', 'CTX95pct_15', 'CTN95pct_5', 'CTX95pct_5',
           'CTN95pct_15_noMov', 'CTX95pct_15_noMov', 'CTN95pct_5_noMov',␣
 ↪'CTX95pct_5_noMov']

for index, i in enumerate(datasets):
    myfile = open(os.path.join(data_loc, 'Heatwave_Limits', 'myFile.txt'), "a")␣
 ↪ # append mode
    myfile.write(docNames[index] + " going now.\n")
    myfile.close()
    i.insert(0,'County_StateFP',theTempMin_dum['County_StateFP'])
    i.insert(0,'CountyFP', theTempMin_dum['CountyFP'])
    i.insert(0,'StateFP', theTempMin_dum['StateFP'])
    i.insert(0,'Year', theTempMin_dum['Year'])
    i.insert(0,'County', theTempMin_dum['County'])
    # The first 20 years are used as a baseline
    i[i['Year']>1999].to_csv(os.path.join(data_loc, 'Heatwave_Limits',␣
 ↪docNames[index] + '.csv'), index=False)

myfile = open(os.path.join(data_loc, 'Heatwave_Limits', 'myFile.txt'), "a")  #␣
 ↪append mode
myfile.write("\n DONE \n")
myfile.close()
```

```
[ ]: CTN90pct_15 = pd.read_csv(os.path.join(data_loc, 'Heatwave_Limits',␣
      ↪'CTN90pct_15.csv')).sort_values(by=['County_StateFP', 'Year'])
     CTX90pct_15 = pd.read_csv(os.path.join(data_loc, 'Heatwave_Limits',␣
      ↪'CTX90pct_15.csv')).sort_values(by=['County_StateFP', 'Year'])
     CTN90pct_5 = pd.read_csv(os.path.join(data_loc, 'Heatwave_Limits', 'CTN90pct_5.
      ↪csv')).sort_values(by=['County_StateFP', 'Year'])
     CTX90pct_5 = pd.read_csv(os.path.join(data_loc, 'Heatwave_Limits', 'CTX90pct_5.
      ↪csv')).sort_values(by=['County_StateFP', 'Year'])
     CTN90pct_15_noMov = pd.read_csv(os.path.join(data_loc, 'Heatwave_Limits',␣
      ↪'CTN90pct_15_noMov.csv')).sort_values(by=['County_StateFP', 'Year'])
     CTX90pct_15_noMov = pd.read_csv(os.path.join(data_loc, 'Heatwave_Limits',␣
      ↪'CTX90pct_15_noMov.csv')).sort_values(by=['County_StateFP', 'Year'])
     CTN90pct_5_noMov = pd.read_csv(os.path.join(data_loc, 'Heatwave_Limits',␣
      ↪'CTN90pct_5_noMov.csv')).sort_values(by=['County_StateFP', 'Year'])
     CTX90pct_5_noMov = pd.read_csv(os.path.join(data_loc, 'Heatwave_Limits',␣
      ↪'CTX90pct_5_noMov.csv')).sort_values(by=['County_StateFP', 'Year'])

     CTN95pct_15 = pd.read_csv(os.path.join(data_loc, 'Heatwave_Limits',␣
      ↪'CTN95pct_15.csv')).sort_values(by=['County_StateFP', 'Year'])
     CTX95pct_15 = pd.read_csv(os.path.join(data_loc, 'Heatwave_Limits',␣
      ↪'CTX95pct_15.csv')).sort_values(by=['County_StateFP', 'Year'])
     CTN95pct_5 = pd.read_csv(os.path.join(data_loc, 'Heatwave_Limits', 'CTN95pct_5.
      ↪csv')).sort_values(by=['County_StateFP', 'Year'])
     CTX95pct_5 = pd.read_csv(os.path.join(data_loc, 'Heatwave_Limits', 'CTX95pct_5.
      ↪csv')).sort_values(by=['County_StateFP', 'Year'])
     CTN95pct_15_noMov = pd.read_csv(os.path.join(data_loc, 'Heatwave_Limits',␣
      ↪'CTN95pct_15_noMov.csv')).sort_values(by=['County_StateFP', 'Year'])
     CTX95pct_15_noMov = pd.read_csv(os.path.join(data_loc, 'Heatwave_Limits',␣
      ↪'CTX95pct_15_noMov.csv')).sort_values(by=['County_StateFP', 'Year'])
     CTN95pct_5_noMov = pd.read_csv(os.path.join(data_loc, 'Heatwave_Limits',␣
      ↪'CTN95pct_5_noMov.csv')).sort_values(by=['County_StateFP', 'Year'])
     CTX95pct_5_noMov = pd.read_csv(os.path.join(data_loc, 'Heatwave_Limits',␣
      ↪'CTX95pct_5_noMov.csv')).sort_values(by=['County_StateFP', 'Year'])
```

```
[ ]: datNames = [CTN90pct_15, CTX90pct_15, CTN90pct_5, CTX90pct_5,
                CTN90pct_15_noMov, CTX90pct_15_noMov, CTN90pct_5_noMov,␣
      ↪CTX90pct_5_noMov,
                CTN95pct_15, CTX95pct_15, CTN95pct_5, CTX95pct_5,
                CTN95pct_15_noMov, CTX95pct_15_noMov, CTN95pct_5_noMov,␣
      ↪CTX95pct_5_noMov]

     docNames = ['CTN90pct_15', 'CTX90pct_15', 'CTN90pct_5', 'CTX90pct_5',
                'CTN90pct_15_noMov', 'CTX90pct_15_noMov', 'CTN90pct_5_noMov',␣
      ↪'CTX90pct_5_noMov',
                'CTN95pct_15', 'CTX95pct_15', 'CTN95pct_5', 'CTX95pct_5',
```

```
          'CTN95pct_15_noMov', 'CTX95pct_15_noMov', 'CTN95pct_5_noMov',␣
 ↪'CTX95pct_5_noMov']

print(theTempMin_dum.shape)
TempMin = theTempMin_dum[(theTempMin_dum['Year']>1999) &␣
 ↪(theTempMin_dum['Year']<2020)]
print(TempMin.shape)
print(CTN90pct_15.shape)
print(theTempMax_dum.shape)
TempMax = theTempMax_dum[(theTempMax_dum['Year']>1999) &␣
 ↪(theTempMax_dum['Year']<2020)]
print(TempMax.shape)
print(CTX90pct_15.shape)
```

[ ]:
```
dumCols = ['PreviousYearDec25', 'PreviousYearDec26',␣
 ↪'PreviousYearDec27','PreviousYearDec28',
          'PreviousYearDec29', 'PreviousYearDec30',␣
 ↪'PreviousYearDec31','NextYearJan1', 'NextYearJan2',
          'NextYearJan3','NextYearJan4','NextYearJan5','NextYearJan6',␣
 ↪'NextYearJan7']

TempMin = TempMin.drop(dumCols, axis=1)
TempMax = TempMax.drop(dumCols, axis=1)

TempMin = TempMin.drop(['County','Year', 'StateFP', 'CountyFP',␣
 ↪'County_StateFP'], axis=1)
TempMax = TempMax.drop(['County','Year', 'StateFP', 'CountyFP',␣
 ↪'County_StateFP'], axis=1)

CTN90pct_15 = CTN90pct_15.drop(['County','Year', 'StateFP', 'CountyFP',␣
 ↪'County_StateFP'], axis=1)
CTX90pct_15 = CTX90pct_15.drop(['County','Year', 'StateFP', 'CountyFP',␣
 ↪'County_StateFP'], axis=1)
CTN90pct_5 = CTN90pct_5.drop(['County','Year', 'StateFP', 'CountyFP',␣
 ↪'County_StateFP'], axis=1)
CTX90pct_5 = CTX90pct_5.drop(['County','Year', 'StateFP', 'CountyFP',␣
 ↪'County_StateFP'], axis=1)
CTN90pct_15_noMov = CTN90pct_15_noMov.drop(['County','Year', 'StateFP',␣
 ↪'CountyFP', 'County_StateFP'], axis=1)
CTX90pct_15_noMov = CTX90pct_15_noMov.drop(['County','Year', 'StateFP',␣
 ↪'CountyFP', 'County_StateFP'], axis=1)
CTN90pct_5_noMov = CTN90pct_5_noMov.drop(['County','Year', 'StateFP',␣
 ↪'CountyFP', 'County_StateFP'], axis=1)
CTX90pct_5_noMov = CTX90pct_5_noMov.drop(['County','Year', 'StateFP',␣
 ↪'CountyFP', 'County_StateFP'], axis=1)
```

```
CTN95pct_15 = CTN95pct_15.drop(['County','Year', 'StateFP', 'CountyFP',␣
 ↪'County_StateFP'], axis=1)
CTX95pct_15 = CTX95pct_15.drop(['County','Year', 'StateFP', 'CountyFP',␣
 ↪'County_StateFP'], axis=1)
CTN95pct_5 = CTN95pct_5.drop(['County','Year', 'StateFP', 'CountyFP',␣
 ↪'County_StateFP'], axis=1)
CTX95pct_5 = CTX95pct_5.drop(['County','Year', 'StateFP', 'CountyFP',␣
 ↪'County_StateFP'], axis=1)
CTN95pct_15_noMov = CTN95pct_15_noMov.drop(['County','Year', 'StateFP',␣
 ↪'CountyFP', 'County_StateFP'], axis=1)
CTX95pct_15_noMov = CTX95pct_15_noMov.drop(['County','Year', 'StateFP',␣
 ↪'CountyFP', 'County_StateFP'], axis=1)
CTN95pct_5_noMov = CTN95pct_5_noMov.drop(['County','Year', 'StateFP',␣
 ↪'CountyFP', 'County_StateFP'], axis=1)
CTX95pct_5_noMov = CTX95pct_5_noMov.drop(['County','Year', 'StateFP',␣
 ↪'CountyFP', 'County_StateFP'], axis=1)
```

[ ]:
```
def savehws(limits, temperatures, consecutiveDays, data_loc, docName, tracker =␣
 ↪None, verbose = 0):
    hws = pd.DataFrame()
    hws['County'] = theTracker['County'].copy()
    hws['Year'] = theTracker['Year'].copy()
    hws['StateFP'] = theTracker['StateFP'].copy()
    hws['CountyFP'] = theTracker['CountyFP'].copy()
    hws['County_StateFP'] = theTracker['County_StateFP'].copy()

    HWN, HWN_JFM, HWN_AMJ, HWN_JAS, HWF, HWF_JFM, HWF_AMJ, HWF_JAS, TDHW,␣
 ↪TDHW_JFM, TDHW_AMJ, TDHW_JAS, HWD = heatwaves(limits, temperatures,
                                                                          ␣
 ↪                                        consecutiveDays, data_loc,
                                                                          ␣
 ↪                                        tracker, verbose)
    hws['# of HW'] = HWN
    hws['# of HW JFM'] = HWN_JFM
    hws['# of HW AMJ'] = HWN_AMJ
    hws['# of HW JAS'] = HWN_JAS
    hws['# of HW days'] = HWF
    hws['# of HW days JFM'] = HWF_JFM
    hws['# of HW days AMJ'] = HWF_AMJ
    hws['# of HW days JAS'] = HWF_JAS
    hws['T of HW'] = TDHW
    hws['T of HW JFM'] = TDHW_JFM
    hws['T of HW AMJ'] = TDHW_AMJ
    hws['T of HW JAS'] = TDHW_JAS
    hws['Longest HW'] = HWD
```

```
    try:
        hws.to_csv(os.path.join(data_loc, 'Heatwaves', 'HW_from_' + docName +␣
↪'_' + str(consecutiveDays) + '.csv'), index = False)
    except Exception as e:
        myfile = open(os.path.join(data_loc, 'Heatwaves', 'errorFile_' +␣
↪str(consecutiveDays) +'.txt'), 'w')
        myfile.write(e)
        myfile.close()
```

[ ]:
```
theTracker = theTempMax_dum[theTempMax_dum['Year']>1999] # To keep track of the␣
↪county - year pair in the loop
```

[ ]:
```
results = Parallel(n_jobs=2)(delayed(savehws)(CTN90pct_15, TempMin, days,␣
↪data_loc, 'CTN90pct_15', theTracker) for days in np.array([3,6]))
results = Parallel(n_jobs=2)(delayed(savehws)(CTX90pct_15, TempMax, days,␣
↪data_loc, 'CTX90pct_15', theTracker) for days in np.array([3,6]))
results = Parallel(n_jobs=2)(delayed(savehws)(CTN90pct_15_noMov, TempMin, days,␣
↪data_loc, 'CTN90pct_15_noMov', theTracker) for days in np.array([3,6]))
results = Parallel(n_jobs=2)(delayed(savehws)(CTX90pct_15_noMov, TempMax, days,␣
↪data_loc, 'CTX90pct_15_noMov', theTracker) for days in np.array([3,6]))
results = Parallel(n_jobs=2)(delayed(savehws)(CTN90pct_5, TempMin, days,␣
↪data_loc, 'CTN90pct_5', theTracker) for days in np.array([3,6]))
results = Parallel(n_jobs=2)(delayed(savehws)(CTX90pct_5, TempMax, days,␣
↪data_loc, 'CTX90pct_5', theTracker) for days in np.array([3,6]))
results = Parallel(n_jobs=2)(delayed(savehws)(CTN90pct_5_noMov, TempMin, days,␣
↪data_loc, 'CTN90pct_5_noMov', theTracker) for days in np.array([3,6]))
results = Parallel(n_jobs=2)(delayed(savehws)(CTX90pct_5_noMov, TempMax, days,␣
↪data_loc, 'CTX90pct_5_noMov', theTracker) for days in np.array([3,6]))

results = Parallel(n_jobs=2)(delayed(savehws)(CTN95pct_15, TempMin, days,␣
↪data_loc, 'CTN95pct_15', theTracker) for days in np.array([3,6]))
results = Parallel(n_jobs=2)(delayed(savehws)(CTX95pct_15, TempMax, days,␣
↪data_loc, 'CTX95pct_15', theTracker) for days in np.array([3,6]))
results = Parallel(n_jobs=2)(delayed(savehws)(CTN95pct_15_noMov, TempMin, days,␣
↪data_loc, 'CTN95pct_15_noMov', theTracker) for days in np.array([3,6]))
results = Parallel(n_jobs=2)(delayed(savehws)(CTX95pct_15_noMov, TempMax, days,␣
↪data_loc, 'CTX95pct_15_noMov', theTracker) for days in np.array([3,6]))
results = Parallel(n_jobs=2)(delayed(savehws)(CTN95pct_5, TempMin, days,␣
↪data_loc, 'CTN95pct_5', theTracker) for days in np.array([3,6]))
results = Parallel(n_jobs=2)(delayed(savehws)(CTX95pct_5, TempMax, days,␣
↪data_loc, 'CTX95pct_5', theTracker) for days in np.array([3,6]))
results = Parallel(n_jobs=2)(delayed(savehws)(CTN95pct_5_noMov, TempMin, days,␣
↪data_loc, 'CTN95pct_5_noMov', theTracker) for days in np.array([3,6]))
```

```
results = Parallel(n_jobs=2)(delayed(savehws)(CTX95pct_5_noMov, TempMax, days,␣
 ↪data_loc, 'CTX95pct_5_noMov', theTracker) for days in np.array([3,6]))
```

[ ]:
```
index = 0
for index_i, i in enumerate(docNames):
    for index_j, j in enumerate(np.array([3,6])):
        if index_i == 0 and index_j == 0:
            weather = pd.read_csv(os.path.join(data_loc, 'Heatwaves',
                                               'HW_from_' + i + '_' + str(j) +␣
 ↪'.csv'))

            ## We will use average temperatures of heatwaves for summary␣
 ↪statistics, and we only need the selected group
            ## hence they will be dopped at this stage

            weather = weather[weather.columns.drop(list(weather.filter(regex='T␣
 ↪of HW')))]
            #specify columns to add suffix to
            cols = weather.columns[5:]
            #add suffix to specific columns
            weather = weather.rename(columns={c: c+ ' (' + i[2:5] + '-' + i[-2:
 ↪] + '-' + str(j) + ')' for c in weather.columns if c in cols})
            group_index = np.ones(weather.columns[5:].shape).astype(int) #␣
 ↪needed for Sparse Group LASSO
        else:
            temp = pd.read_csv(os.path.join(data_loc, 'Heatwaves',
                                               'HW_from_' + i + '_' + str(j) +␣
 ↪'.csv'))

            temp = temp[temp.columns.drop(list(temp.filter(regex='T of HW')))]

            cols = temp.columns[5:]

            if "noMov" in i and "15" in i:
                temp = temp.rename(columns={c: c+ ' (' + i[2:5] + '-' + i[9:11]␣
 ↪+ '-' + str(j) + '-NM)' for c in temp.columns if c in cols})

            elif "noMov" in i and "15" not in i:
                temp = temp.rename(columns={c: c+ ' (' + i[2:5] + '-' + i[9:10]␣
 ↪+ '-' + str(j) + '-NM)' for c in temp.columns if c in cols})
            elif "noMov" not in i and "15" not in i:
                temp = temp.rename(columns={c: c+ ' (' + i[2:5] + '-' + i[-1] +␣
 ↪'-' + str(j) + ')' for c in temp.columns if c in cols})

            else:
```

```
                temp = temp.rename(columns={c: c+ ' (' + i[2:5] + '-' + i[-2:]␣
↪+ '-' + str(j) + ')' for c in temp.columns if c in cols})

            weather = pd.merge(weather, temp, left_on = ['County', 'Year',␣
↪'StateFP', 'CountyFP', 'County_StateFP'],
                               right_on = ['County', 'Year', 'StateFP',␣
↪'CountyFP', 'County_StateFP'])

            group_index = np.concatenate((group_index, np.ones(weather.
↪columns[5:].shape).astype(int) + index))

        index = index + 1



weather.head()
```

[ ]:
```
weather.to_csv(os.path.join(data_loc, 'WEATHER.csv'), index = False)
```

3 MERGING WEATHER DATA WITH ECON DATA AND CONDUCTING LASSO - GROUP LASSO - SPARSE GROUP LASSO

[ ]:
```
import sys
import os

import numpy as np
import pandas as pd
import numpy.matlib

import warnings

warnings.simplefilter(action='ignore', category=pd.errors.PerformanceWarning)

import sklearn
from sklearn import linear_model
from sklearn.metrics import r2_score
from sklearn import preprocessing
from sklearn.linear_model import Lasso, LinearRegression, Lars
from sklearn.model_selection import RandomizedSearchCV

plt.rcParams['figure.figsize'] = [16, 8]
import time

import asgl
```

```
# Check software versions
print('### Python version: ' + __import__('sys').version) # 3.9.12
print('### NumPy version: ' + np.__version__) # 1.22.3
print('------------')
```

[ ]:
```
parentPath = os.path.split(os.path.abspath(os.getcwd()))[0]
print(parentPath)
weather = pd.read_csv(os.path.join(parentPath, 'Data', 'WEATHER.csv'))
weather.head() # Each group has 9 variables
```

[ ]:
```
data = weather[['County', 'StateFP', 'CountyFP']]
data = data.drop_duplicates(subset = ['County', 'StateFP', 'CountyFP'])
data['GeoFIPS'] = ''
for index, row in data.iterrows():
    if row['StateFP'] < 10:
        if row['CountyFP'] < 10:
            data.loc[index, 'GeoFIPS'] = '0' + str(row['StateFP']) + '00' +␣
 ↪str(row['CountyFP'])
        elif row['CountyFP'] > 9 and row['CountyFP'] < 100:
            data.loc[index, 'GeoFIPS'] = '0' + str(row['StateFP']) + '0' +␣
 ↪str(row['CountyFP'])
        elif row['CountyFP'] > 99:
            data.loc[index, 'GeoFIPS'] = '0' + str(row['StateFP']) +␣
 ↪str(row['CountyFP'])
    if row['StateFP'] > 9:
        if row['CountyFP'] < 10:
            data.loc[index, 'GeoFIPS'] = str(row['StateFP']) + '00' +␣
 ↪str(row['CountyFP'])
        elif row['CountyFP'] > 9 and row['CountyFP'] < 100:
            data.loc[index, 'GeoFIPS'] = str(row['StateFP']) + '0' +␣
 ↪str(row['CountyFP'])
        elif row['CountyFP'] > 99:
            data.loc[index, 'GeoFIPS'] = str(row['StateFP']) +␣
 ↪str(row['CountyFP'])

data.head(14)
```

[ ]:
```
print(weather.shape)
weather = pd.merge(weather, data)
print(weather.shape)
```

3.1 Heatwave Figures

[ ]:
```
warnings.filterwarnings("ignore")

from matplotlib import pyplot as plt

# map coloring
from matplotlib import cm
import cmasher as cmr


plt.rcParams['figure.figsize'] = [16, 8]
import plotly.figure_factory as ff


cols = np.array(['# of HW (X95-15-6)', '# of HW (X95-15-6-NM)', '# of HW␣
 ↪(X95-5-6)', '# of HW (X90-15-6)',
                  '# of HW (X95-5-3)', '# of HW (X95-5-3-NM)', '# of HW␣
 ↪(X95-15-3)', '# of HW (X90-5-3)'])



for col in cols:
    df_sample = theData[theData['Year']==2012]
    df_sample = df_sample[['GeoFIPS', col]]

    colorscale = cmr.take_cmap_colors('Reds', 12, return_fmt='hex')

    endpts = list(np.linspace(0, np.max(theData['# of HW (X90-5-3)']),␣
 ↪len(colorscale) - 1))
    fips = df_sample['GeoFIPS'].tolist()
    values = df_sample[col].tolist()


    fig = ff.create_choropleth(
        fips=fips, values=values, scope=['usa'],
        binning_endpoints=endpts, colorscale=colorscale,
        show_state_data=True,
        state_outline = {'color': 'gray', 'width': .5},
        show_hover=True,
        round_legend_values = True,
        asp = 2.9,
       # title_text = col.replace('#', 'Number') + " in " + str(2012),
        legend_title = col
    )
    fig.layout.template = None
```

```
    fig.write_image(os.path.join(parentPath, 'Figures', col[9:-1] + ".png"))

fig.show()
```

3.1.1 Download county level gdp from https://apps.bea.gov/regional/downloadzip.cfm

[ ]:
```
data = pd.read_csv(os.path.join(parentPath, 'Data',␣
 ↪'CAINC1__ALL_AREAS_1969_2021.csv'))
data = data[data['Description'] != 'Personal income (thousands of dollars) ']
data.loc[data['Description'] == 'Population (persons) 1/', 'Description'] =␣
 ↪'Population'
data.loc[data['Description'] == 'Per capita personal income (dollars) 2/',␣
 ↪'Description'] = 'Per capita personal income'
data.loc[:,'GeoFIPS'] = data.loc[:,'GeoFIPS'].apply(lambda s: s[2:-1]) # Erase␣
 ↪" "
data = data[data['Region'] != ' '] # No need for US level data now
# No information for IndustryClassification, only '...'
data = data.drop(['TableName', 'LineCode', 'IndustryClassification', 'Unit'],␣
 ↪axis=1)
data.head()
```

[ ]:
```
population_temp = data[data['Description'] == 'Population']
population_temp = population_temp.drop('Description', axis =1)
income_temp = data[data['Description'] == 'Per capita personal income']
income_temp = income_temp.drop('Description', axis =1)

cols = population_temp.columns[3:]
for index, i in enumerate(cols):
    if index == 0:
        # Population
        population = population_temp[['GeoFIPS', 'GeoName', 'Region', i]].copy()
        population['Year'] = int(i)
        population = population.rename(columns = {i: 'Population'})
        # Income
        income = income_temp[['GeoFIPS', 'GeoName', i]].copy()
        income['Year'] = int(i)
        income = income.rename(columns = {i: 'Personalincome_pc'})
    else:
        # Population
        temp = population_temp[['GeoFIPS', 'GeoName', 'Region', i]].copy()
        temp['Year'] = int(i)
        temp = temp.rename(columns = {i: 'Population'})
        population = pd.concat((population, temp), axis=0)
        # income
        temp = income_temp[['GeoFIPS', 'GeoName', i]].copy()
        temp['Year'] = int(i)
        temp = temp.rename(columns = {i: 'Personalincome_pc'})
```

```
        income = pd.concat((income, temp), axis=0)
```

[ ]:
```
bea = pd.merge(population, income)
print('Shape of population data: {}, shape of income data: {}, shape of merged␣
 ↪data: {}'.format(population.shape[0],
                                                                                 ␣
 ↪                 income.shape[0],
                                                                                 ␣
 ↪                 bea.shape[0]))
bea = bea[bea['Population'] != "(NA)"]
bea['Population'] = bea['Population'].astype(float)
bea['Personalincome_pc'] = bea['Personalincome_pc'].astype(float)
```

[ ]:
```
data_cpi = pd.read_csv(os.path.join(parentPath, "Data", "USACPIALLAINMEI.csv"))
data_cpi['Year'] = np.arange(1960, 2023, 1)
data_cpi = data_cpi.drop('DATE', axis=1)
data_cpi = data_cpi.rename(columns = {'USACPIALLAINMEI': 'CPI'})
data_cpi.head()
```

[ ]:
```
print(bea.shape)
bea = pd.merge(bea, data_cpi)
print(bea.shape)
bea.head()
```

[ ]:
```
for i in bea.columns[5:-1]:
    bea[i] = 100*bea[i]/bea['CPI']
```

[ ]:
```
# Take logs
bea[bea.columns[5:-1]] =  np.log(1000*bea[bea.columns[5:-1]]) # Data is in␣
 ↪thousands
bea = bea.sort_values(by = ['GeoFIPS', 'Year'])

potential_LHS = []
for i in bea.columns[5:-1]:
    bea['L_' + i] = bea.groupby('GeoFIPS')[i].shift()
    bea['D_' + i] = bea[i] - bea['L_' + i]
    potential_LHS.append('D_' + i)
    bea = bea.drop([i, 'L_' + i], axis = 1)
bea = bea.sort_values(by = ['GeoFIPS', 'Year'])
potential_LHS = np.array(potential_LHS)

# Add lags for the dependent variable
lags = 2
lags_RHS = []
for j in potential_LHS:
    for i in range(1,lags+1):
        bea['L' + str(i) + '_' + j] = bea.groupby('GeoFIPS')[j].shift(i)
```

```
        lags_RHS.append('L' + str(i) + '_' + j)
bea = bea.dropna(subset = lags_RHS)
bea.head()
```

[ ]:
```
# Drop States outside of US mainland
weather = weather[weather['StateFP']<57]
# Drop Hawaii
weather = weather[weather['StateFP']!=15]
# Drop Alaska
weather = weather[weather['StateFP']!=2]
# Drop DC
weather = weather[weather['StateFP']!=11]
```

[ ]:
```
mergedData = pd.merge(bea, weather, left_on = ['GeoFIPS', 'Year'], right_on =␣
 ↪['GeoFIPS', 'Year'])
mergedData.insert(3, 'year', mergedData['Year'])
mergedData = mergedData.drop('Year', axis = 1)
mergedData = mergedData.rename(columns = {'year': 'Year'})
print('Shape of bea data: {}, shape of weather data: {}, shape of merged data:␣
 ↪{}'.format(bea.shape[0],
                                                                              ␣
 ↪                weather.shape[0],
                                                                              ␣
 ↪                mergedData.shape[0]))
mergedData = mergedData.drop(['CPI'], axis = 1)
mergedData.head()
```

[ ]:
```
mask = np.isin(np.unique(weather['GeoFIPS']), np.unique(mergedData['GeoFIPS']),␣
 ↪invert = True)
print(np.unique(weather['GeoFIPS'])[mask])
# Virginia is 51
```

[ ]:
```
allData = mergedData.drop(['Region', 'Population'], axis=1)
allData = allData.sort_values(by = ['GeoFIPS', 'Year'])
weatherCols = allData.columns[19:]

weather_events = []
for i in weatherCols:
    allData['D_' + i] = allData.groupby('GeoFIPS')[i].diff()

    allData = allData.drop([i], axis=1)

    weather_events.append('D_' + i)


allData = allData.sort_values(by = ['GeoFIPS', 'Year'])
allData = allData.dropna(subset = np.array(weather_events))
```

```
allData.head()
```

[ ]:
```
potentialLHS = np.array(potential_LHS)
lagsRHS = np.array(lags_RHS)
weatherEvents = np.array(weather_events)
```

[ ]:
```
dependent = 'Personalincome_pc'
lags_dependent = lagsRHS[np.flatnonzero(np.core.defchararray.find(lagsRHS, 'D_'␣
 ↪+ dependent)!=-1)]
#allRHS = np.concatenate((lags_dependent, np.array(['Government_Farm_State'])))
allRHS = np.concatenate((lags_dependent, weatherEvents))

allData_ = allData.copy()
allData_[np.concatenate((np.array(['D_' + dependent]), lags_dependent))].
 ↪describe()
```

[ ]:
```
warnings.filterwarnings("ignore")

from matplotlib import pyplot as plt

plt.rcParams['figure.figsize'] = [16, 8]
import plotly.figure_factory as ff

i = 2019
col = 'D_Personalincome_pc'
df_sample = allData_[allData_['Year']==i]
df_sample = df_sample[['GeoFIPS', col]]

colorscale = ["#750e13", "#a2191f", "#da1e28", "#fa4d56", "#ff8389", "#ffb3b8",␣
 ↪"#ffd7d9",
        "#deebf7", "#c6dbef", "#9ecae1", "#6baed6", "#4292c6",
        "#2171b5", "#08519c", "#08306b"
    ]

endpts = list(np.linspace(np.min(df_sample[col]), np.max(df_sample[col]),␣
 ↪len(colorscale) - 1))
fips = df_sample['GeoFIPS'].tolist()
values = df_sample[col].tolist()


fig = ff.create_choropleth(
    fips=fips, values=values, scope=['usa'],
    binning_endpoints=endpts, colorscale=colorscale,
    show_state_data=True,
    state_outline = {'color': 'gray', 'width': .5},
    show_hover=True,
    round_legend_values = False,
```

```
    asp = 2.9,
    #title_text = 'Personal Income Growth ' + str(i-1) + '-' + str(i),
    #legend_title = r'$\Delta$' + 'log(personal income per capita)'
)
fig.layout.template = None

fig.write_image(col + '_' + str(i) + ".png")

fig.write_image(os.path.join(parentPath, 'Figures', col + '_' + str(i) + ".
 ↪png"))


fig.show()
```

[ ]:
```
dependent_limit_up = allData_['D_' + dependent].quantile(0.99)
dependent_limit_down = allData_['D_' + dependent].quantile(0.01)


allData_ = allData_[(allData_['D_' + dependent]>=dependent_limit_down) &␣
 ↪(allData_['D_' + dependent]<=dependent_limit_up) &
                    (allData_[lags_dependent[0]] >= dependent_limit_down) &␣
 ↪(allData_[lags_dependent[0]] <= dependent_limit_up) &
                    (allData_[lags_dependent[1]] >= dependent_limit_down) &␣
 ↪(allData_[lags_dependent[1]] <= dependent_limit_up)]

allData_[np.concatenate((np.array(['D_' + dependent]), lags_dependent))].
 ↪describe()
```

[ ]:
```
warnings.filterwarnings("ignore")

from matplotlib import pyplot as plt

plt.rcParams['figure.figsize'] = [16, 8]
import plotly.figure_factory as ff

i = 2019
col = 'D_Personalincome_pc'
df_sample = allData_[allData_['Year']==i]
df_sample = df_sample[['GeoFIPS', col]]

colorscale = ["#750e13", "#a2191f", "#da1e28", "#fa4d56", "#ff8389", "#ffb3b8",␣
 ↪"#ffd7d9",
        "#deebf7", "#c6dbef", "#9ecae1", "#6baed6", "#4292c6",
        "#2171b5", "#08519c", "#08306b"
    ]
```

```
endpts = list(np.linspace(np.min(df_sample[col]), np.max(df_sample[col]),␣
 ↪len(colorscale) - 1))
fips = df_sample['GeoFIPS'].tolist()
values = df_sample[col].tolist()


fig = ff.create_choropleth(
    fips=fips, values=values, scope=['usa'],
    binning_endpoints=endpts, colorscale=colorscale,
    show_state_data=True,
    state_outline = {'color': 'gray', 'width': .5},
    show_hover=True,
    round_legend_values = False,
    asp = 2.9,
    #title_text = 'log() ' + str(i),
    # legend_title = col
)
fig.layout.template = None

fig.write_image(os.path.join(parentPath, 'Figures', col + '_trimmed_' + str(i)␣
 ↪+ ".png"))

fig.show()
```

[ ]:
```
yOriginal = allData_['D_' + dependent].values
XOriginal = allData_[allRHS].values
X = preprocessing.StandardScaler().fit_transform(XOriginal)
y = preprocessing.StandardScaler().fit_transform(yOriginal.reshape(-1,1))
y = y.reshape(-1,)
regData = pd.DataFrame()
regData['GeoFIPS']= allData_['GeoFIPS'].copy()
regData['StateName']= allData_['StateName'].copy()
regData['Year']= allData_['Year'].copy()
regData['ClimateRegion'] = allData_['ClimateRegion'].copy()
regData[allRHS] = X.copy()
regData['D_' + dependent] = y.copy()

regData.head()
```

[ ]:
```
# Year Dummies
yearDummies = pd.get_dummies(regData['Year'], prefix='y', dtype = 'float32')
# County Dummies
countyDummies = pd.get_dummies(regData['GeoFIPS'], prefix = 's', dtype =␣
 ↪'float32')
# Generate Matrix with both Year and Country Dummies
theDummies = pd.concat((yearDummies, countyDummies), axis = 1)
print('Year Fixed Effects and County Fixed Effects')
```

```
theDummies.head()
```

[ ]:
```
forcedRHS = theDummies.drop(['s_56045'], axis=1).values # drop one of the␣
 ↪columns because of the intercept
squareMatrix = np.transpose(forcedRHS)@forcedRHS

y_noTDummy = y - forcedRHS@(np.linalg.inv(squareMatrix) @ (np.
 ↪transpose(forcedRHS)@y))
X_noTDummy = X - forcedRHS@(np.linalg.inv(squareMatrix) @ (np.
 ↪transpose(forcedRHS)@X))
```

[ ]:
```
group_index = np.ones(2)
for i in range(1, int((X_noTDummy.shape[1]-2)/9)+1): # the first two columns␣
 ↪are lags, there are 9 features by group
    temp = np.ones(9) + i
    group_index = np.concatenate((group_index, temp))
```

[ ]:
```
gl_model = asgl.ASGL(model='lm', penalization='gl', lambda1=.034)
gl_model.fit(x=X_noTDummy,y=y_noTDummy, group_index=group_index)
nonZero_gl = np.nonzero(gl_model.coef_[0])[0]-1 # the first one is the intercept
print('The variables selected by Group LASSO are: ')
print(str(allRHS[nonZero_gl]))
```

[ ]:
```
sgl_model = asgl.ASGL(model='lm', penalization='sgl', lambda1=.0355, alpha=.465)
sgl_model.fit(x=X_noTDummy,y=y_noTDummy, group_index=group_index)
nonZero_sgl = np.nonzero(sgl_model.coef_[0])[0]-1 # the first one is the␣
 ↪intercept
print('The variables selected by Sparse Group LASSO are: ')
print(str(allRHS[nonZero_sgl]))
```

4 Regression

[ ]:
```
from linearmodels.panel import PanelOLS
```

[ ]:
```
regData_= allData_.copy()
regData_[np.concatenate((np.array('D_' + dependent).reshape(1,),
                         lags_dependent))] = regData_[np.concatenate((np.
 ↪array('D_' + dependent).reshape(1,),
                                                                      ␣
 ↪lags_dependent))]*100 # convert to percentages
regData_ = regData_.set_index(['GeoFIPS','Year'])

# exog_vars = np.concatenate((lags_dependent, np.
 ↪array(['D_PDSI_moreThan4','D_PDSI_lessThan_4',
#  'D_DTR', 'D_TNHW'])))
exog_vars = allRHS[nonZero_sgl]
```

```
exog = regData_[exog_vars]
# exog = exog.rename()
indep = regData_['D_' + dependent]

# Run the regressions
mod = PanelOLS(indep, exog, entity_effects = True, time_effects = True)
res = mod.fit(cov_type='clustered', cluster_entity=True)
print(res)
```