GEOVIS - Process Crime Data

Consolidate crime data calculating monthly averages for each crime variable, for each county in TX and NY

0. Libraries and Variables

In [1]:
```
#  variables

import arcpy
import arcgis
import os
import numpy as np
import pandas as pd

from arcgis import GIS
from arcgis.features import GeoAccessor, GeoSeriesAccessor
```

In [3]:
```
_gis = GIS('Pro')


_inGdb = r'C:\Users\Ameri\OneDrive - The University of Texas at Dallas\UT Dalla

_outFolder = r'C:\Users\Ameri\OneDrive - The University of Texas at Dallas\UT D
_outGdb = 'Crime1.gdb'
_outGdb = os.path.join(_outFolder, _outGdb)

d_CrimeTypes = {'aggravated_assault': 'agga',
                'arson': 'arsn',
                'burglary': 'burg',
                'homicide': 'homi',
                'larceny': 'larc',
                'motor_vehicle_theft': 'mvt',
                'rape': 'rape',
                'robbery': 'rob'}
l_CrimeTypes = list(sorted(d_CrimeTypes.keys()))

# county polygons come from processed weather fc
_fc_ProcessedWeather = r'C:\Users\Ameri\OneDrive - The University of Texas at D
# city polygons
_fc_Cities = r'C:\Users\Ameri\OneDrive - The University of Texas at Dallas\UT D

print(_inGdb, os.path.exists(_inGdb))
print(_outFolder, os.path.exists(_outFolder))
print(_fc_ProcessedWeather, arcpy.Exists(_fc_ProcessedWeather))
print(_fc_Cities, arcpy.Exists(_fc_Cities))
```

```
C:\Users\Ameri\OneDrive - The University of Texas at Dallas\UT Dallas\2024 03
Fall\EPPS 6356 Data Visualization\GeoVis\RawData\Crime\States_all\monthly_198
5to2023_PD19178_point.gdb True
C:\Users\Ameri\OneDrive - The University of Texas at Dallas\UT Dallas\2024 03
Fall\EPPS 6356 Data Visualization\GeoVis\DataProcess True
C:\Users\Ameri\OneDrive - The University of Texas at Dallas\UT Dallas\2024 03
Fall\EPPS 6356 Data Visualization\GeoVis\DataProcess\Weather3.gdb\Weather_NY_
TX_v3 True
C:\Users\Ameri\OneDrive - The University of Texas at Dallas\UT Dallas\2024 03
Fall\EPPS 6356 Data Visualization\GeoVis\RawData\AdministrativeBoundaries\Cit
y_Boundary\city_TX_NY.gdb\City_NY_TX True
```

In [4]:
```
#  Verify all crime feature classes exist

for tag in l_CrimeTypes:
    fc = f'{tag}_monthly_1985to2023'
    print(fc, arcpy.Exists(os.path.join(_inGdb, fc)))
```

```
aggravated_assault_monthly_1985to2023 True
arson_monthly_1985to2023 True
burglary_monthly_1985to2023 True
homicide_monthly_1985to2023 True
larceny_monthly_1985to2023 True
motor_vehicle_theft_monthly_1985to2023 True
rape_monthly_1985to2023 True
robbery_monthly_1985to2023 True
```

1. Filter all crime feature classes in Texas & New York

In [5]:
```
#  Filter all crime feature classes

d_DFs_Unprocessed = {}  #  dictionary of crime types dataframes

#  loop all crime types,
#     filter NY & TX data,
#     save filtered dataframe in dictionary

for tag in l_CrimeTypes:
    #  get feature class name
    fc = f'{tag}_monthly_1985to2023'
    print(f"~   ... filtering feature class --> {fc}")
    fc = os.path.join(_inGdb, fc)

    #  Read feature class
    sdf = pd.DataFrame.spatial.from_featureclass(fc)

    #  filter the data  # NY & TX
    #  state_name IN ('New York', 'Texas')
    filtered_df = sdf[sdf['state_name'].isin(['New York', 'Texas'])].copy()

    #  save filtered data in dictionary d_DFs for later processing
    d_DFs_Unprocessed[tag] = filtered_df
```

```
~   ... filtering feature class --> aggravated_assault_monthly_1985to2023
~   ... filtering feature class --> arson_monthly_1985to2023
~   ... filtering feature class --> burglary_monthly_1985to2023
~   ... filtering feature class --> homicide_monthly_1985to2023
~   ... filtering feature class --> larceny_monthly_1985to2023
~   ... filtering feature class --> motor_vehicle_theft_monthly_1985to2023
~   ... filtering feature class --> rape_monthly_1985to2023
~   ... filtering feature class --> robbery_monthly_1985to2023
```

In [6]:
```
#  read weather feature class (processed)
#  crime data will be added to weather fc

sdf_Weather = pd.DataFrame.spatial.from_featureclass(_fc_ProcessedWeather)
sdf_Weather.head()
```

Out[6]:

	OBJECTID	statefp	countyfp	countyns	geoid	name	namelsad	lsad	classfp	mtfcc	cs
0	1	48	327	01383949	48327	Menard	Menard County	06	H1	G4020	
1	2	48	189	01383880	48189	Hale	Hale County	06	H1	G4020	
2	3	48	011	01383791	48011	Armstrong	Armstrong County	06	H1	G4020	
3	4	48	057	01383814	48057	Calhoun	Calhoun County	06	H1	G4020	
4	5	48	077	01383824	48077	Clay	Clay County	06	H1	G4020	

5 rows × 115 columns

In [7]:
```
#  Create a temp Weather df with county name as index
#  to allow for a quick data retrieval

df_Weather_Temp = sdf_Weather.copy()
df_Weather_Temp['COUNTYID'] = df_Weather_Temp['name'].str.strip().str.upper()
df_Weather_Temp.set_index('COUNTYID', inplace=True)
l_counties = df_Weather_Temp.index.tolist()

print(len(l_counties))
print(len(df_Weather_Temp))
df_Weather_Temp.head()
```

```
316
316
```

Out[7]:

COUNTYID	OBJECTID	statefp	countyfp	countyns	geoid	name	namelsad	lsad	classfp
MENARD	1	48	327	01383949	48327	Menard	Menard County	06	H1
HALE	2	48	189	01383880	48189	Hale	Hale County	06	H1
ARMSTRONG	3	48	011	01383791	48011	Armstrong	Armstrong County	06	H1
CALHOUN	4	48	057	01383814	48057	Calhoun	Calhoun County	06	H1
CLAY	5	48	077	01383824	48077	Clay	Clay County	06	H1

5 rows × 115 columns

In [9]:
```
#  read cities feature class
sdf_Cities = pd.DataFrame.spatial.from_featureclass(_fc_Cities)
sdf_Cities.head()
```

Out[9]:

	OBJECTID	NAME	MUNI_TYPE	MUNITYCODE	COUNTY	GNIS_ID	FIPS_CODE	SWIS
0	1	Albany	city	1.0	Albany	978659	3600101000	010100
1	2	Amsterdam	city	1.0	Montgomery	978677	3605702066	270100
2	3	Auburn	city	1.0	Cayuga	978695	3601103078	050100
3	4	Batavia	city	1.0	Genesee	978713	3603704715	180200
4	5	Beacon	city	1.0	Dutchess	978716	3602705100	130200

In [11]:
```
#  Create a temp Cities df with county name as index
#  to allow for a quick data retrieval

df_City_Temp = sdf_Cities[['OBJECTID', 'NAME', 'COUNTY', 'SHAPE']].copy()
df_City_Temp['COUNTYID'] = df_City_Temp['NAME'].str.strip().str.upper()
df_City_Temp.set_index('COUNTYID', inplace=True)

print(len(df_City_Temp))
df_City_Temp.head()
```

```
1285
```

Out[11]:

COUNTYID	OBJECTID	NAME	COUNTY	SHAPE
ALBANY	1	Albany	Albany	{"rings": [[[-73.85154879699996, 42.7115402990...
AMSTERDAM	2	Amsterdam	Montgomery	{"rings": [[[-74.20012254099998, 42.9659517100...
AUBURN	3	Auburn	Cayuga	{"rings": [[[-76.59473494199995, 42.9560956210...
BATAVIA	4	Batavia	Genesee	{"rings": [[[-78.19013013599994, 43.0164674340...
BEACON	5	Beacon	Dutchess	{"rings": [[[-73.94261272799997, 41.5150065580...

In [12]:
```
#  create dictionary for months that will be used to process data

d_month = {}
d_month[1] = 'Jan'
d_month[2] = 'Feb'
d_month[3] = 'Mar'
d_month[4] = 'Apr'
d_month[5] = 'May'
d_month[6] = 'Jun'
d_month[7] = 'Jul'
d_month[8] = 'Aug'
d_month[9] = 'Sep'
d_month[10] = 'Oct'
d_month[11] = 'Nov'
d_month[12] = 'Dec'

l_months = list(d_month.keys())

print(d_month)
print(l_months)
```

```
{1: 'Jan', 2: 'Feb', 3: 'Mar', 4: 'Apr', 5: 'May', 6: 'Jun', 7: 'Jul', 8: 'Au
g', 9: 'Sep', 10: 'Oct', 11: 'Nov', 12: 'Dec'}
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]
```

In [13]:
```
#  loop through crime datasets
#

#  ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
def PreProcessCrimeDataset(df):

    # Create an empty DataFrame with the same columns
    l_columns = list(df.columns)

    # remove all 2024 columns
    l_cols2remove = [x for x in l_columns if x.startswith(f'y2024')]
    l_columns = [x for x in l_columns if not x in l_cols2remove]

    # 1st get all crime data that do NOT need spliting values
    df = df[l_columns].copy()
    new_df = df[~df['counties'].str.contains(',')].copy()

    # Loop through each row in the original DataFrame that needs spliting value
    df_to_duplicate = df[df['counties'].str.contains(',')].copy()
    for index, row in df_to_duplicate.iterrows():

        l_counties = row['counties'].split(',')
        # Trim whitespace from each county
        l_counties = [cty.strip() for cty in l_counties]
        pct = int(100 / len(l_counties))

        for cty in l_counties:

            # Create a new row for each county
            new_row = row.copy()
            new_row['counties'] = cty

            for m in l_months:
                l_cols2Process = [x for x in l_columns if x.endswith(f'_{m}')]
                for col2Process in l_cols2Process:
                    if not pd.isnull(new_row[col2Process]):
                        newValue = new_row[col2Process]
                        if newValue >= len(l_counties):
                            newValue = int(newValue / len(l_counties))
                        new_row[col2Process] = newValue

            # Append the new row to the new DataFrame
            new_df = new_df.append(new_row, ignore_index=True)

    return new_df


for tag in l_CrimeTypes:
    print(f"~      ... Pre-processing dataset: {tag}")
    df = d_DFs_Unprocessed[tag]
    newdf = PreProcessCrimeDataset(df)
    d_DFs_Unprocessed[tag] = newdf


# df = d_DFs_Unprocessed['arson'].copy()
# # newdf = df[df['counties'].str.contains(',')]
# newdf = ProcessCrimeDataset(df)
```

```
# print(len(newdf))
# newdf.head()
```

```
[13]:43: FutureWarning: The frame.append method is deprecated and will be r
emoved from pandas in a future version. Use pandas.concat instead.
[13]:43: FutureWarning: The frame.append method is deprecated and will be r
emoved from pandas in a future version. Use pandas.concat instead.
[13]:43: FutureWarning: The frame.append method is deprecated and will be r
emoved from pandas in a future version. Use pandas.concat instead.
[13]:43: FutureWarning: The frame.append method is deprecated and will be r
emoved from pandas in a future version. Use pandas.concat instead.
[13]:43: FutureWarning: The frame.append method is deprecated and will be r
emoved from pandas in a future version. Use pandas.concat instead.
[13]:43: FutureWarning: The frame.append method is deprecated and will be r
emoved from pandas in a future version. Use pandas.concat instead.
[13]:43: FutureWarning: The frame.append method is deprecated and will be r
emoved from pandas in a future version. Use pandas.concat instead.
[13]:43: FutureWarning: The frame.append method is deprecated and will be r
emoved from pandas in a future version. Use pandas.concat instead.
[13]:43: FutureWarning: The frame.append method is deprecated and will be r
emoved from pandas in a future version. Use pandas.concat instead.
[13]:43: FutureWarning: The frame.append method is deprecated and will be r
emoved from pandas in a future version. Use pandas.concat instead.
```

2. Calculate monthly averages

In [19]:
```
#  function to calculate monthly averages in dataframe
def ProcessCrimeDF(df, crimeType):

    sdf = df.copy()  # make a copy to preserve original df

    l_columns = list(sdf.columns)
    l_colsProcessed = []
    l_newColumns = []

    #  calculate monthly averages
    for m in l_months:

        #  get columns ending in m (month)
        l_cols2Process = [x for x in l_columns if x.endswith(f'_{m}')]

        #  add cols2process to list of processed cols
        l_colsProcessed.extend(l_cols2Process)

        #  create new monthly avg column
        #colName = f'{weatherType}_avg_{m}_{d_month[m]}'
        colName = f'c_{d_CrimeTypes[crimeType]}_{d_month[m]}'

        #  calculate mean value in monthly column
        print(f"~      ... Processing column: {colName}")
        sdf[colName] = sdf[l_cols2Process].mean(axis=1)

        #  add to list of new columns
        l_newColumns.append(colName)

    #  get updated columns list
    #l_columns = list(sdf.columns)

    #  columns excluding processed ones
    #l_columns = [x for x in l_columns if x not in l_colsProcessed]

    l_desiredCols = [x for x in l_newColumns]
    l_desiredCols.insert(0, 'counties')
    sdf = sdf[l_desiredCols].copy()

    # group by county and calculate averages
    sdf['COUNTYID'] = sdf['counties'].str.strip().str.upper()
    grouped_df = sdf.groupby('COUNTYID')[l_newColumns].mean()
    grouped_df = grouped_df.reset_index()

    #  return updated/grouped sdf
    return grouped_df
```

In [20]:
```
#  process all crime dataframes

d_DFs_Processed = {}  #  dictionary of processed crime dataframes

for tag in l_CrimeTypes:

    #  get dataframe to process from dict
    df = d_DFs_Unprocessed[tag]

    print(f"~   ... calculating monthly averages for crime variable --> {tag}")
    sdf = ProcessCrimeDF(df, tag)

    # add processed df to processed dict
    d_DFs_Processed[tag] = sdf
```

```
~      ... Processing column: c_arsn_Jul
~      ... Processing column: c_arsn_Aug
~      ... Processing column: c_arsn_Sep
~      ... Processing column: c_arsn_Oct
~      ... Processing column: c_arsn_Nov
~      ... Processing column: c_arsn_Dec
~   ... calculating monthly averages for crime variable --> burglary
~      ... Processing column: c_burg_Jan
~      ... Processing column: c_burg_Feb
~      ... Processing column: c_burg_Mar
~      ... Processing column: c_burg_Apr
~      ... Processing column: c_burg_May
~      ... Processing column: c_burg_Jun
~      ... Processing column: c_burg_Jul
~      ... Processing column: c_burg_Aug
~      ... Processing column: c_burg_Sep
~      ... Processing column: c_burg_Oct
~      ... Processing column: c_burg_Nov
~      ... Processing column: c_burg_Dec
~   ... calculating monthly averages for crime variable --> homicide
```

3. Consolidate Crime Variables Monthly Averages into a single Dataset

In [21]:
```
#  merge processed columns into a single dataframe

df_Final = None  # this will hold the final dataframe

for tag in l_CrimeTypes:

    #  get processed dataframe
    df = d_DFs_Processed[tag]

    if df_Final is None:
        df_Final = df.copy() # initialize with first dataframe
    else:
        # append dataframe df to df_Final

        #  get only processed columns
        l_columns = list(df.columns)
        l_cols_desired = [x for x in l_columns if x.startswith(f'c_{d_CrimeType
        l_cols_desired.insert(0, 'COUNTYID') # keep id for merging purposes

        #  merge to df final
        df = df[l_cols_desired].copy()
        df_Final = pd.merge(df_Final, df, on='COUNTYID')


df_Final.head()
```

Out[21]:

	COUNTYID	c_agga_Jan	c_agga_Feb	c_agga_Mar	c_agga_Apr	c_agga_May	c_agga_Jun	c_
0	ALBANY	4.228518	3.859037	4.952757	5.110903	5.718672	5.813904	
1	ALLEGANY	0.576763	0.432766	0.544778	0.558405	0.537479	0.466439	
2	ANDERSON	4.405271	4.469373	4.777778	4.839744	5.288462	5.248575	
3	ANDREWS	2.076923	1.250337	2.048920	1.987179	2.162281	1.753711	
4	ANGELINA	1.820441	1.708778	2.013495	1.987916	2.600978	2.271838	

5 rows × 97 columns

In [22]:
```
#  Merge crime variables to weather variables

df_Weather_Crime = df_Weather_Temp.reset_index() # weather dataframe

df_Weather_Crime = pd.merge(df_Weather_Crime, df_Final, on='COUNTYID') # merge

df_Weather_Crime.head()
```

Out[22]:

	COUNTYID	OBJECTID	statefp	countyfp	countyns	geoid	name	namelsad	lsad	clas
0	MENARD	1	48	327	01383949	48327	Menard	Menard County	06	
1	HALE	2	48	189	01383880	48189	Hale	Hale County	06	
2	ARMSTRONG	3	48	011	01383791	48011	Armstrong	Armstrong County	06	
3	CALHOUN	4	48	057	01383814	48057	Calhoun	Calhoun County	06	
4	CLAY	5	48	077	01383824	48077	Clay	Clay County	06	

5 rows × 212 columns

In [23]:
```
# Create filegdb
if (not os.path.exists(_outGdb)):
    _outGdb = 'Weather_Crime_1.gdb'
    arcpy.management.CreateFileGDB(_outFolder, _outGdb)
    _outGdb = os.path.join(_outFolder, _outGdb)

print(_outGdb, os.path.exists(_outGdb))
```

```
C:\Users\Ameri\OneDrive - The University of Texas at Dallas\UT Dallas\2024 03 Fall\EPPS 6356 Data Visualization\GeoVis\DataProcess\Weather_Crime_1.gdb True
```

In [24]:
```
#  convert to spatial dataframe

sdf_Weather_Crime = GeoAccessor.from_df(df_Weather_Crime, geometry_column='SHAP
sdf_Weather_Crime.head()
```

Out[24]:

	COUNTYID	OBJECTID	statefp	countyfp	countyns	geoid	name	namelsad	lsad	clas
0	MENARD	1	48	327	01383949	48327	Menard	Menard County	06	
1	HALE	2	48	189	01383880	48189	Hale	Hale County	06	
2	ARMSTRONG	3	48	011	01383791	48011	Armstrong	Armstrong County	06	
3	CALHOUN	4	48	057	01383814	48057	Calhoun	Calhoun County	06	
4	CLAY	5	48	077	01383824	48077	Clay	Clay County	06	

5 rows × 212 columns

In [26]:
```
#  save consolidated data to feature class
outWeatherCrimeFC = 'WeatherCrime_NYTX_v1'
outWeatherCrimeFC = os.path.join(_outGdb, outWeatherCrimeFC)

sdf_Weather_Crime.spatial.to_featureclass(outWeatherCrimeFC)
```

Out[26]:
```
'C:\\Users\\Ameri\\OneDrive - The University of Texas at Dallas\\UT Dallas\\2
024 03 Fall\\EPPS 6356 Data Visualization\\GeoVis\\DataProcess\\Weather_Crime
_1.gdb\\WeatherCrime_NYTX_v1'
```

In [27]:
```
print(len(sdf_Weather_Crime))
```

```
313
```

In [ ]:
```

```