Paper SAS4293-2020

Practical Geospatial Analysis of Open and Public-Use Data

Pradeep Mohan, SAS Institute Inc.

Abstract

As a geospatial data scientist, you might want to analyze publicly available data sets from several sources, including kaggle.com, data.gov, and other academic or scientific organizations. Analyzing open and public-use data sets such as these poses many challenges, including disparate data formats (ESRI shapefile, NetCDF, GeoJSON, and so on), disparate geographic references, and the use of street address data instead of actual spatial units. This paper presents a variety of real-world examples of public-use geospatial data to show you how to transform the data into a prespecified geographic reference, import the data into SAS® Viya® via a Python interface, and select suitable subsets that can be convenient for data analysis. These subsets are then analyzed by using SAS/STAT® procedures KRIGE2D, SPP, and GLIMMIX. The paper also demonstrates how to map the estimates from these procedures by using the Base SAS® mapping procedure SGMAP invoked from Python. Examples in the paper show you how to use Python as a common programming environment to perform data management functions by using the SAS Scripting Wrapper for Analytics Transfer (SWAT) interface to SAS Viya and how to perform statistical analysis by submitting procedure calls via the SASPy interface to SAS software.

Introduction

The past decade has witnessed a rapid growth in collection velocity, archival volume, and the number of organizations that observe, process, and disseminate various forms of geospatial data. Such data are collected about several geospatial phenomena, including global climate, weather, pollution, crime, traffic, disease, and real estate. Geospatial data that are collected about these phenomena belong to two broad categories: (1) raster data that are collected by sensors, satellites, or drones and (2) vector data that are produced by digitization and simplification of topographic survey maps. Such geospatial data are disseminated by government, industry, and academic organizations via open-data platforms such as data.gov, kaggle.com, ArcGIS Hub, and open-data initiatives of several local city and state governments.

As a geospatial data scientist, you often encounter many such open and public-use data in a geospatial data science life cycle. Depending on the nature of this life cycle, you might be faced with a situation where you need to combine data from a variety of sources about the same study area (such as world, country, or city). These sources usually have the following defining characteristics:

- They all involve data about the same geographical region, but in a variety of file formats.
- Different data sets about the same region might be represented in disparate coordinate systems.
- Because of privacy concerns, some data sets might include an approximate street address instead of the actual location.

All these characteristics pose roadblocks that need to be overcome before you can perform geospatial analysis that includes spatial statistical modeling and mapping. This paper shows you a practical approach that uses open-source geospatial data processing facilities in conjunction with your investments in SAS® software.

Practical Geospatial Analysis Using Python and SAS

A practical approach to geospatial analysis requires a seamless system for geospatial data processing, geospatial analysis, modeling, and mapping of results. This paper shows you how to implement this system in a common programming framework that uses Python as a geospatial data science language in conjunction with SAS Viya and SAS 9.4 procedures. More precisely, the paper shows you how to accomplish the following:

- Read raster data that are stored as NetCDF files by using Python and convert those data into a Pandas data frame that you can upload to SAS Viya using the SAS Scripting Wrapper for Analytics Transfer (SWAT).
- Read vector data that are in spatial file formats such as ESRI shapefiles and GeoJSON and that can be converted to a GeoPandas data frame and uploaded as a SAS® Cloud Analytics Services (CAS) table to your SAS Viya server.
- Run SQL queries to subset spatial data in SAS Viya and export those data to a SAS 9.4 data set.
- Fit spatial statistical models by using SAS/STAT procedures VARIOGRAM, KRIGE2D, SPP, and GLIMMIX and the SASPy package that bridges Python and SAS 9.4.
- Wherever possible, produce maps of statistical estimates and residuals by using both SAS and Python.

Overview and Scope

The next section describes the two Python interfaces to SAS: the SAS Viya Python client and the SASPy package, which serves as a bridge to SAS 9.4 procedures. The next three sections describe case studies of global precipitation data, US county homicides data, and tax-delinquent property data from the city of Philadelphia.

All examples in this paper are implemented in Python 3.7 by using Jupyter Notebook as the development environment. This paper assumes that you are familiar with Python programming and development environments such as Jupyter Notebook for data science. It also assumes you are familiar with installing and configuring Python via platforms such as Anaconda. For more introductory information, you can read Phillips (2019) and Smith and Meng (2017).

Python Interfaces to SAS

Python is an object-oriented and interpreted programming language that has an extensive array of open-source packages. Popular Python packages relevant to geospatial analysis include Pandas (Pandas developers 2019), GeoPandas (GeoPandas developers 2019), netCDF4 (Unidata 2019), xarray (xarray developers 2019), and Cartopy (Met Office 2010–2015). These packages are primarily used to read geospatial data from different file formats and transform coordinate systems to produce a Pandas data frame. The data frame can be added as a CAS table or a SAS data set. With Python and Jupyter Notebook running, you can import different packages as follows:

```
import geopandas as geopd
import pandas as pd
import cartopy as carto
import matplotlib.pyplot as plt
from cartopy import crs as ccrs
import xarray as xr
```

Before you can start reading different types of spatial data and converting them to data frames that are suitable for analysis with SAS, you must set up a connection to the SAS Viya server and a SAS session.

SAS Viya Python Client

The SAS Viya Python client is a package named SAS Scripting Wrapper for Analytics Transfer (SWAT). SWAT is available as a public-domain Python package from SAS. For more information about downloading and installing SWAT, see Smith and Meng (2017). Once you have SWAT installed, you can connect to a SAS Viya server, as shown in the following steps:

```
from swat import *
import swat.cas.datamsghandlers as dmh
s = CAS(server,port)
s.serverstatus()
```

Connecting to the SAS Viya server requires authentication information. The preceding method assumes that you have an authinfo file in the root directory. For more information about the authinfo method of authentication to connect to the SAS Viya server, see Chapter 3 of Smith and Meng (2017).

SASPy Python Package Interface to SAS 9.4

To submit SAS procedure calls to perform statistical analysis or other tasks such as mapping, you must create a connection to a SAS 9.4 session. The Python package SASPy serves as the interface between Python and SAS 9.4. For more information about installing and configuring SASPy, see Phillips (2019). Once you have SASPy installed and configured, you can establish a connection to SAS 9.4 by running the following steps:

```
import saspy
sas = saspy.SASsession()
```

Case Study 1: Geostatistical Smoothing of Global Precipitation Data

Global precipitation data are measured from a variety of sources such as satellites, sensors deployed in oceans, and other mobile platforms. Such data are usually accumulated and distributed in raster data formats such as NetCDF files. Because such data can be noisy and have many missing values, they require geostatistical smoothing, which can smooth the noisy observations and impute the missing values.

This section shows you how to use Python to import NetCDF files, upload the data to SAS Viya, select a suitable subset for geostatistical smoothing, and finally perform the smoothing using the SAS/STAT procedures VARIOGRAM and KRIGE2D.

Global Precipitation Data

The precipitation data are originally from the Global Precipitation Climatology Project (GPCP) from NOAA (Adler et al. 2003).[1] The data contain monthly precipitation data from 1979 to the present, combining observations and satellite precipitation data onto a 2.5 degree latitute $\times$ 2.5 degree longitude global grid whose dimensions are 144 $\times$ 72. The data are available as NetCDF files that can be downloaded from the following URL:

```
https://www.esrl.noaa.gov/psd/data/gridded/data.gpcp.html
```

Once you have downloaded the NetCDF file that corresponds to the monthly mean precipitation (**precip.mon.mean.nc**), you can place the file in a directory named **Precipitation** and rename it to **PRECIP-MonthlyMeans.nc**. Next, you can declare a Python string variable that contains the path to the NetCDF file. To read this file and perform operations such as selecting subsets of its data, you use the Python package `xarray`. The following steps enable you to read the NetCDF file from a file system location, select the time slice for the current century, convert the file to a Pandas series object, and then finally convert it to a Pandas data frame:

[1] GPCP Precipitation data provided by the NOAA/OAR/ESRL PSD, Boulder, Colorado, USA, from their Web site at https://www.esrl.noaa.gov/psd/.

```
filename = 'Precipitation/PRECIPMonthlyMeans.nc'
precpmonthly = xr.open_dataset(filename)
ds21 = precpmonthly.sel(time=slice('2000-01-01',
                         '2010-12-31'))
s21 = ds21['precip'].to_series()
df = s21.to_frame()
df.info()
```

When you use the **xarray** package to read the NetCDF file, you get an xarray DataArray type that is specifically designed to store geospatial data about a study area over a period of time. Such data are usually called a spatial time series. The original data from NOAA contain timestamps for 31 years. This example is concerned only with data for the first decade of this century, January 2000 to December 2010. For the sake of simplicity, you can just focus on the year 2010 to perform geostatistical smoothing by using BY groups that correspond to each month in the year. Running the preceding code in your Jupyter notebook produces the following result:

```
<class 'pandas.core.frame.DataFrame'>
MultiIndex: 1368576 entries, (2000-01-01 00:00:00, -88.75, 1.25) to (2010-12-01
00:00:00, 88.75, 358.75)
Data columns (total 1 columns):
precip    1368576 non-null float32
dtypes: float32(1)
memory usage: 11.8 MB
                        precip
time       lat    lon
2000-01-01 -88.75 1.25      0.0
                  3.75      0.0
                  6.25      0.0
                  8.75      0.0
                  11.25     0.0
```

This result shows information about the Pandas data frame that was created. By default, the data frame has a MultiIndex of the variables **time**, **lat**, and **lon**. However, the format of this data frame has a couple of issues:

- The data frame has a MultiIndex that requires additional processing before the data frame can be uploaded to SAS Viya.
- The data are in a geodetic coordinate system that uses longitude and latitude, so you need to transform the coordinate system to projected X and Y.

For the first issue, you can execute the following steps:

```
df.to_csv('datalonlat.csv',header=True)
df21 = pd.read_csv('datalonlat.csv')
```

The preceding steps write the Pandas data frame to a CSV file and read it back, during which the MultiIndex-based representation is lost. The **df21** data frame is in a format that can be easily uploaded as a new CAS table in your SAS Viya server.

For the second issue, you can transform the data to projected coordinates by using the following steps:

```
gdf21 = geopd.GeoDataFrame(df21,
             geometry=geopd.points_from_xy(df21.lon,df21.lat))
if gdf21.crs == None:
    gdf21.crs={'init':'epsg:4326'}
gdf21 = gdf21.to_crs(epsg=32662)
gdf21['X'] = gdf21.geometry.x
gdf21['Y'] = gdf21.geometry.y
df21upload = gdf21[['time','lat','lon','precip','X','Y']]
```

The preceding coordinate transformation is a two-step process. First, you use the GeoPandas package's GeoDataFrame constructor to turn your Pandas data frame into a GeoPandas data frame. The GeoPandas data frame contains the coordinates as a point **geometry** column. Then, you use the GeoPandas coordinate transformation function **to_crs()** with the **geometry** column to perform the coordinate transformation. Since the data have global coverage, you use the Plate Caree projection whose European Petroleum Survey Group (EPSG) identifier is 32662. The coordinate transformation produces a final data frame that is suitable to be uploaded as a CAS table to the SAS Viya server. More precisely, the output looks as follows:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1368576 entries, 0 to 1368575
Data columns (total 6 columns):
time      1368576 non-null object
lat       1368576 non-null float64
lon       1368576 non-null float64
precip    1368576 non-null float64
X         1368576 non-null float64
Y         1368576 non-null float64
dtypes: float64(5), object(1)
memory usage: 62.6+ MB

         time    lat    lon  precip             X             Y
0  2000-01-01 -88.75   1.25     0.0  1.391494e+05 -9.879605e+06
1  2000-01-01 -88.75   3.75     0.0  4.174481e+05 -9.879605e+06
2  2000-01-01 -88.75   6.25     0.0  6.957468e+05 -9.879605e+06
3  2000-01-01 -88.75   8.75     0.0  9.740455e+05 -9.879605e+06
4  2000-01-01 -88.75  11.25     0.0  1.252344e+06 -9.879605e+06
```

The preceding output shows information about a Pandas data frame that can be uploaded to the SAS Viya server. The output shows that the Pandas data frame has 1.36 million rows and 6 columns. The variables **lon** and **lat** represent geodetic coordinates, and the variables **X** and **Y** represent projected coordinates in the Plate Carree projection. The variable **precip** is the value of the precipitation rate at the given time and location.

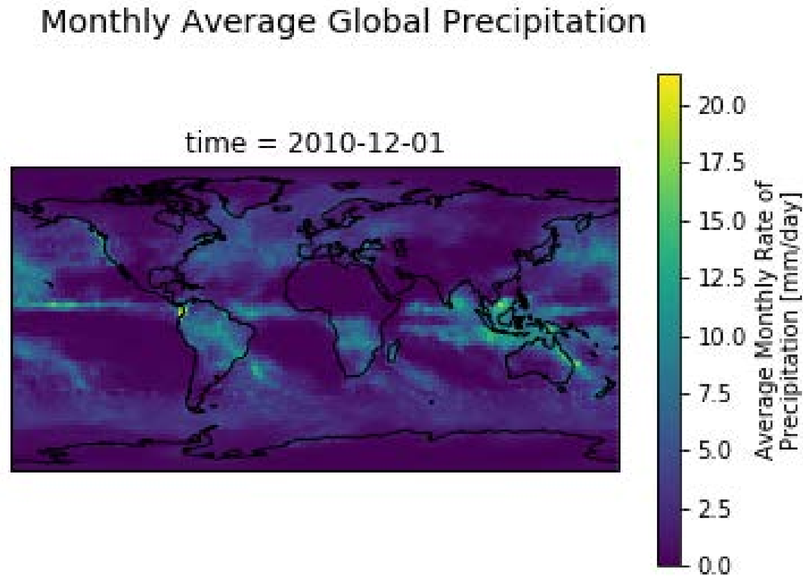
Mapping the Precipitation Data

As is common exploratory practice in geospatial analysis, you can produce a map of the input data. You use the following steps to produce a map of the precipitation data for the last timestamp, which is December 2010 and corresponds to an index value of 131:

```
ds21precip = ds21['precip']
i = 131
ax = plt.axes(projection=ccrs.PlateCarree())
ax.coastlines()
figname = "Precipitation/images/Precipitation%d.png"%(i+1)
tas = ds21precip.isel(time=i)
tas.plot.pcolormesh(ax=ax,transform=ccrs.PlateCarree())
plt.suptitle('Monthly Average Global Precipitation',
               ha='center',x=0.465,y=0.98,size='x-large')
plt.savefig(figname)
plt.clf()
```

An image like Figure 1 is written to your file system, showing the precipitation levels for December 2010.

Figure 1 Monthly Average Global Precipitation, December 2010 (Adler et al. 2003)



Managing the Precipitation Data in SAS Viya

Before the global precipitation data can be geostatistically smoothed, it is recommended that you upload the data to the SAS Viya server in order to perform important data management tasks such as subsetting, creating views, and creating BY groups. Also, SAS Viya gives you the capability to analyze such a large amount of data; these data contain 1.36 million rows and 132 timestamps. Before communicating with the SAS Viya server, you must set up the connection by using the steps outlined in the section "SAS Viya Python Client" on page 3. Then, you upload the **df21upload** Pandas data frame to SAS Viya by using the following steps, which use the `addtable` action to create a new CAS table called **Precipitation21Century**:

```
handler = dmh.PandasDataFrame(df21upload)
out = s.addtable(table='Precipitation21Century',
                 replace=True,
                 **handler.args.addtable)
Precipitation21 = s.CASTable("Precipitation21Century")
```

To focus on the year 2010 alone, you can subset the data by using the `fedSQL` action set. The `fedSQL` action set supports SQL queries and can be used for data subset selection, joins, and aggregation. To create a new CAS table that contains the information for the year 2010 alone, you can use the following code:

```
s.builtins.loadActionSet("fedSql")
PrecipResults = s.fedSql.execDirect(query=''' create table
                                 Precipitation2010{options replace=true} as
                                 select * from Precipitation21Century
                                 where time like '2010%' ''')
Precip2010 = s.CASTable("Precipitation2010")
print(Precip2010.table.tableInfo())
```

The preceding `fedSQL` query creates a new CAS table called **Precipitation2010** and produces output that gives more information about the newly created table, including the number of rows, the number of columns, and the encoding. The table contains 124,416 rows and has all six columns of the original CAS table. This table contains data for 12 timestamps which correspond to the 12 months of the year. Going further, you use the smaller CAS table as input to SAS/STAT procedures to perform geostatistical smoothing. Before you do this, the CAS table **Precipitation2010** needs to be written to a SAS data set with a LIBNAME.

Writing the CAS Table to a SAS Data Set

To write the CAS table **Precipitation2010** to a SAS data set, you must obtain a SAS data frame from the CAS table. Then the SAS data frame can be converted to a SAS data set by using the SASPy package. To create a SAS data frame, you can use the **fetch** action and request all rows in the table, as in the following step:

```
Precipitation2010DF = s.fetch(table=table('Precipitation2010'),
                              maxRows=124416,to=124416)['Fetch']
```

Finally, you can use the SASPy package to create a SAS data set from **Precipitation2010DF**. Before you use the SASPy package, make sure that you have followed the steps to connect to a SAS 9.4 session, as described in the section "SASPy Python Package Interface to SAS 9.4" on page 3. The following steps can be used to write the **Precipitation2010DF** data frame to a SAS data set named **PRECIPKRIGE**.SAS in a library named **sgf2020**:

```
sas.saslib('sgf2020',path='U:/sgf2020')
precip2010sd = sas.df2sd(Precipitation2010DF,"PRECIPKRIGE","sgf2020")
```

In the preceding steps, the variable **sas** is SASPy's SAS 9.4 session object that is used to invoke functions to access SAS 9.4 facilities. The step uses the **sas.saslib** function to create a SAS library and uses the **sas.df2sd** function to create a data set. Once the data set is created within a particular library, you can use it like a normal SAS data set as input to any SAS/STAT procedure.

The next step is to perform geostatistical smoothing of the global precipitation data by using the KRIGE2D procedure in SAS/STAT. Using the SASPy session connection object, **sas**, you can submit SAS/STAT procedure calls and obtain outputs such as logs and listings.

Geostatistical Smoothing of Global Precipitation

Before you perform geostatistical smoothing of the precipitation data, you should view the contents of the **sgf2020.PRECIPKRIGE** data set. To see the contents, you can run the following code:

```
cont= sas.submit(''' proc contents
                        data=sgf2020.PRECIPKRIGE;
                     run; ''')
sas.HTML(cont["LST"])
```

This code uses the **sas.submit()** function of the SASPy session object to submit the CONTENTS procedure call. The output listing that contains all the tables and figures that are produced by the procedure is stored in the results object, **cont**. You can access the HTML listing of the procedure by using the function **sas.HTML()** and directly reference the **LST** column of the results object **cont**. The **sgf2020.PRECIPKRIGE** contains all the variables in the SAS data frame **Precipitation2010DF**.

The variable **time** in the **sgf2020.PRECIPKRIGE** data is appropriate as a BY-group variable for geostatistical smoothing. To perform geostatistical smoothing, you use the SAS/STAT procedure KRIGE2D, which requires a suitable spatial covariance structure as input. To select an appropriate spatial covariance structure, you use the VARIOGRAM procedure. For more information about using PROC VARIOGRAM and PROC KRIGE2D, see SAS Institute Inc. (2018b). You can submit the following steps, which call PROC VARIOGRAM by using the **sas.submit()** function of the SASPy session object:

```
variog = sas.submit(''' ods select FitPlot;
                        proc variogram data=sgf2020.precipkrige
                                       plots(only)=Fit;
                           by time;
                           store out=SemivPrecipStore /
                                 label= 'Precipitation Level Model';
                           compute lagd=100000 maxlag=100;
                           coord xc=X yc=Y;
```

```
                              model form=auto(mlist=(cub,exp,gau,
                                          pen,pow,she,sph) nest=1);
                              var precip;
                           run;''')
print(variog["LOG"])
sas.HTML(variog["LST"])
```

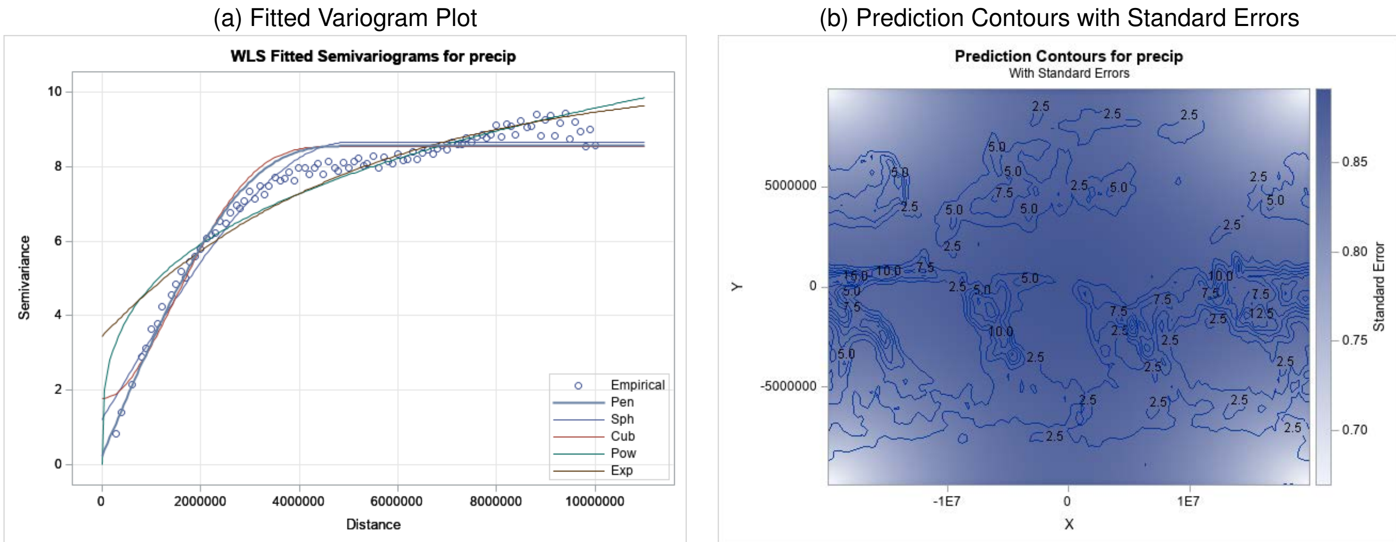
This code selects the most suitable spatial covariance model for the global precipitation data and stores the covariance parameters in the SAS item store **SemivPrecipStore**. The item store is reused by the KRIGE2D procedure to obtain a kriged estimate of precipitation. This code selects only the fitted variogram plot, which shows the best-fitting variograms for each BY group. The preceding code generates an HTML listing that shows the fitted variogram plot for each BY group. Figure 2(a) shows the fitted variogram plot for the first BY group (January 2010). PROC VARIOGRAM chooses the pentaspherical spatial covariance function as the best-fitting spatial covariance function and stores the covariance parameters in the SAS item store **SemivPrecipStore**.

The final step in geostatistical smoothing is using the KRIGE2D procedure to perform spatial interpolation by using the BY-group variable time. In effect, the spatial interpolation smooths the precipitation data.

```
krig = sas.submit(''' ods select PredictionPlot;
                      proc krige2d data=sgf2020.precipkrige
                                outest=sgf2020.PredPrecip
                                plots(only)=(pred);
                         by time;
                         restore in=SemivPrecipStore;
                         coordinates xc=X yc=Y;
                         predict var=precip numpoints=8;
                         model storeselect;
                         grid x=-19898358 to 19898358 by 276366
                              y=-9879604  to 9879604  by 274433;
                      run;''')
print(krig["LOG"])
sas.HTML(krig["LST"])
```

The preceding code performs ordinary kriging of global precipitation levels and make predictions on a user-defined grid. The grid is typically defined on the basis of the underlying data's coordinate system and the extent of the study area. The preceding statements show you how to reuse the fitted covariance parameters from the **SemivPrecipStore** item store. The output predictions and a measure of uncertainty that is associated with the predictions are stored in the **sgf2020.PredPrecip** data set. Finally, the preceding code selects only the kriging prediction plot. The listing produced by the procedure call will contain only the prediction plot for each BY group. Figure 2(b) shows the prediction contours of the global precipitation levels in January 2010, which is the first BY group. The listing produced by the call to PROC KRIGE2D produces a prediction plot for each BY group whose spatial covariance did not have any singularities and resulted in predicted values for all the output grid locations. The contours in Figure 2(b) show the precipitation levels, and the background color is the uncertainty associated with that prediction, usually quantified by the standard error.

Figure 2 Geostatistical Smoothing of Global Precipitation in 2010 (January)



Case Study 2: Generalized Linear Geostatistical Modeling of County Homicides

The example in the section "Case Study 1: Geostatistical Smoothing of Global Precipitation Data" on page 3 showed you how to import raster geospatial data such as NetCDF files and provided an illustrative example of how to smooth those data using a geostatistical process model. This section applies a similar approach to vector geospatial data.

County-Level Homicides in the 1990s

This case study considers the county level homicide data that are distributed by the GeoDa Center at the University of Chicago (GeoDa Data and Lab 2019).[2] County-level homicide data were originally published by Messner et al. (2000) in the National Consortium of Violence Research (NCOVR). The data contain information about homicides for four decades: the 1960s, '70s, '80s, and '90s. For each decade, the data contain information about the homicide counts, rates, and other additional socioeconomic covariates, including population density and unemployment rate. Baller et al. (2001) consider a spatial error model for the homicide rates that is based on the covariates.

The county homicide data are available in vector spatial data format, which are usually distributed as ESRI shapefiles. To read an ESRI shapefile in Python, you use the Geopandas package. The following code shows you how to read a shapefile and transform the coordinates to a projected coordinate system:

```
countycrimes = geopd.read_file('CountyHomicides/NAT.shp')
countycrimesUS = countycrimes.to_crs("+proj=aea +lat_1=29.5 +lat_2=45.5 \
                                  +lat_0=37.5 +lon_0=-96 +x_0=0 +y_0=0 \
                                  +ellps=GRS80 +datum=NAD83 +units=m +no_defs")
```

This code transforms the coordinate reference system of the data to the Albers equal-area projection by using its corresponding PROJ.4 string format (PROJ contributors 2020). The PROJ.4 string format requires you to specify different parameters that are associated with the coordinate reference system definition and allows you to define custom coordinate references for a geospatial data set. For more information about the PROJ.4 format and different parameters that are used to define coordinate reference systems, see PROJ contributors (2020). The following steps produce a map of the county-level homicide rates:

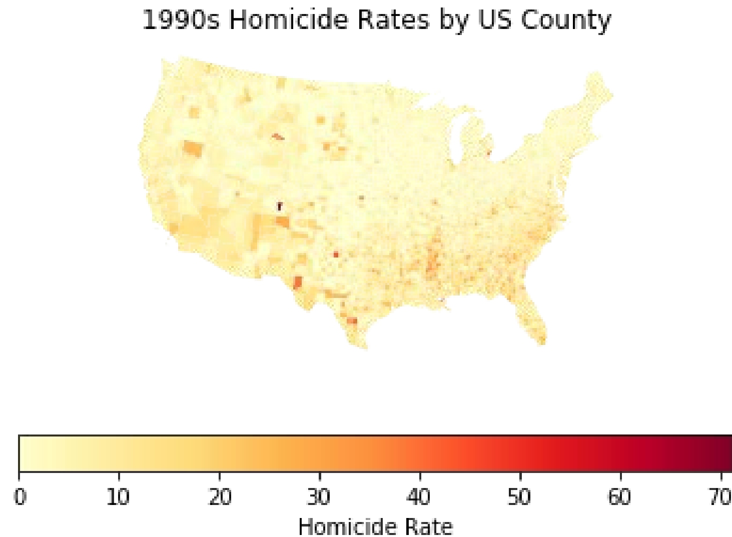
```
f,ax1 = plt.subplots(1)
ax1 = countycrimesUS.plot(column='HR90',cmap='YlOrRd',
                          linewidth=0,legend=True,
                          legend_kwds={'label':"Homicide Rate",
```

[2]The county-level homicide data from the GeoDa Center at the University of Chicago are available at the following URL: https://geodacenter.github.io/data-and-lab/ncovr/

```
                              'orientation':"horizontal"},
                              ax=ax1)
ax1.set_axis_off()
ax1.set_title("1990s Homicide Rates by US County")
plt.show()
```

Figure 3 shows the county-level homicide rates (per 100,000 people) in the 1990s.

Figure 3 Homicide Rates by US County in the 1990s



To perform generalized linear geostatistical modeling, you must also produce centroids for the US county polygons. Also, you need only a few of the 70 original columns, as shown in the following code:

```
countycrimesUS['cenX']=countycrimesUS['geometry'].centroid.x
countycrimesUS['cenY']=countycrimesUS['geometry'].centroid.y
countycrimesUSDF = countycrimesUS[['FIPS','geometry','HC90','HR90','PO90',
                                  'RD90','PS90','UE90','DV90','MA90',
                                  'POL90','DNL90','MFIL89','FP89','BLK90',
                                  'GI89','FH90','cenX','cenY']]
countycrimesUSDF['geometry'].head()
```

The preceding code produces a Geopandas data frame that has 19 columns, which include the **geometry** of each county, the **FIPS** identifier for the county, and other variables. In addition, the centroids of the counties are stored as **cenX** and **cenY**. The **geometry** is stored as a special type called geometry. The following output shows a few entries of the **geometry** column:

```
0    POLYGON ((49050.366 1231770.124, 49053.866 125{...}
1    POLYGON ((-1704187.878 1371703.990, -1690089.8{...}
2    POLYGON ((-1598348.983 1357299.562, -1605943.2{...}
3    POLYGON ((-1713271.481 1372754.888, -1712880.8{...}
4    POLYGON ((-1574802.355 1459813.690, -1545432.8{...}
Name: geometry, dtype: geometry
```

The county polygons are stored as a collection of vertices in a format called the well-known text (WKT) format, which is represented by an abstract data type called geometry. To learn more about the WKT format, see Herring (2011). To upload the data into SAS Viya, you must convert the abstract data type to a variable-length character column that can be saved in a CAS table, as in the following step:

```
countycrimesUSDF['geometry'] = countycrimesUSDF.geometry.astype(str)
```

To upload the **countycrimesUSDF** data frame to the SAS Viya server, you can run the following steps, which create a CAS table named **USCountyCrimes**:

```
handler = dmh.PandasDataFrame(countycrimesUSupload)
s.addtable(table='USCountyCrimes', **handler.args.addtable)
```

Once you have the table **USCountyCrimes** created in SAS Viya, there is one issue to overcome before you can use the table in SAS 9.4. The **geometry** cannot be stored in a SAS data set because it is a variable-length character column whose length can exceed the maximum length that SAS data sets allow for character variables. However, you can run the following step in SAS Viya to use only centroids for the polygons:

```
res = s.dataStep.runCode(code= '''data USCountyCrimesXY;
                                     set USCountyCrimes;
                                     drop geometry;
                                   run;''',
                         nThreads="MAX")
```

The preceding step produces the table **USCountyCrimesXY**, which can be written to a SAS data set that can be used as input to SAS/STAT procedures. You store the SAS data set **USCountyCrimeXY** in the `sgf2020` library exactly as in the previous section, "Case Study 1: Geostatistical Smoothing of Global Precipitation Data" on page 3.

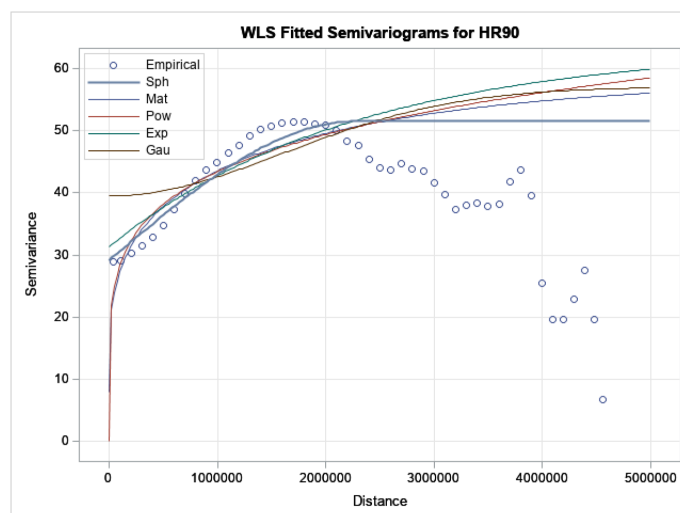
Generalized Linear Geostatistical Modeling

As described in the section "Case Study 1: Geostatistical Smoothing of Global Precipitation Data" on page 3, it is often a good practice to use PROC VARIOGRAM to get a sense of the spatial covariance structure. The spatial covariance model selection facility in PROC VARIOGRAM can help you identify the best-fitting spatial covariance function according to Akaike's information criterion (AIC) or sum of squared errors (SSE). You do this by using the following steps:

```
covarmod = sas.submit('''ods select FitPlot;
                         proc variogram data=sgf2020.USCountyCrimesXY
                                        plots(only)=Fit;
                            compute lagd=100000 maxlag=100;
                            coord xc=cenX yc=cenY;
                            model form=auto(mlist=(exp,gau,pow,mat,sph) nest=1);
                            var HR90;
                         run;''')
```

Figure 4 shows that PROC VARIOGRAM selects the spherical spatial covariance function as the best fit.

Figure 4 Fitted Semivariogram for 1990s Homicide Rate



Next, you fit a generalized linear geostatistical model for 1990 homicide rates. In this model, you treat the homicide rates as Poisson and include a spatial random effect as spatially autocorrelated residuals. You can fit such a model by using the GLIMMIX procedure in SAS/STAT. For more information about the GLIMMIX procedure, see SAS Institute Inc. (2018b). The following code uses PROC GLIMMIX to fit a Poisson model with spatial random effects for the homicide rates:

```
geoglm90 = sas.submit('''proc glimmix data=sgf2020.USCountyCrimesXY
                                      plots=none;
                            id FIPS HR90 HC90 BLK90 RD90 PS90 UE90
                               DV90 FP89 POL90 DNL90 MFIL89 GI89
                               cenX cenY;
                            model HR90 = BLK90 RD90 PS90 UE90 DV90 FP89
                                         GI89 MFIL89 /
                                         dist=poisson link=log solution;
                            random _residual_ /
                            subject=intercept type=sp(pow)(cenX cenY);
                            output out=sgf2020.uscountyHR90Pred
                            pred(ilink)=HR90Pred student=stdResidual;
                         run;''')
```

The GLIMMIX procedure stores the output estimates for each county in the data set **sgf2020.uscountyHR90Pred**. The predicted mean is the variable **HR90Pred**. You can also request studentized residuals to assess prediction quality. Finally, you can map these estimates by using the SGMAP procedure in Base SAS.

Mapping the Estimates in SAS

Before mapping, you import the original shapefiles into SAS by using the MAPIMPORT procedure to read the ESRI shapefile and output the data to **sgf2020.USCountyMap**. The following code makes a call to the MAPIMPORT procedure via the `sas.submit()` function in SASPy:

```
mapio = sas.submit('''proc mapimport out=sgf2020.USCountyMap
                         datafile="U:/sgf2020/CountyHomicides/NAT.shp";
                      run;''')
```

The following steps use the SGMAP procedure to produce one map for the actual homicide rates and one map the estimated homicide rates:

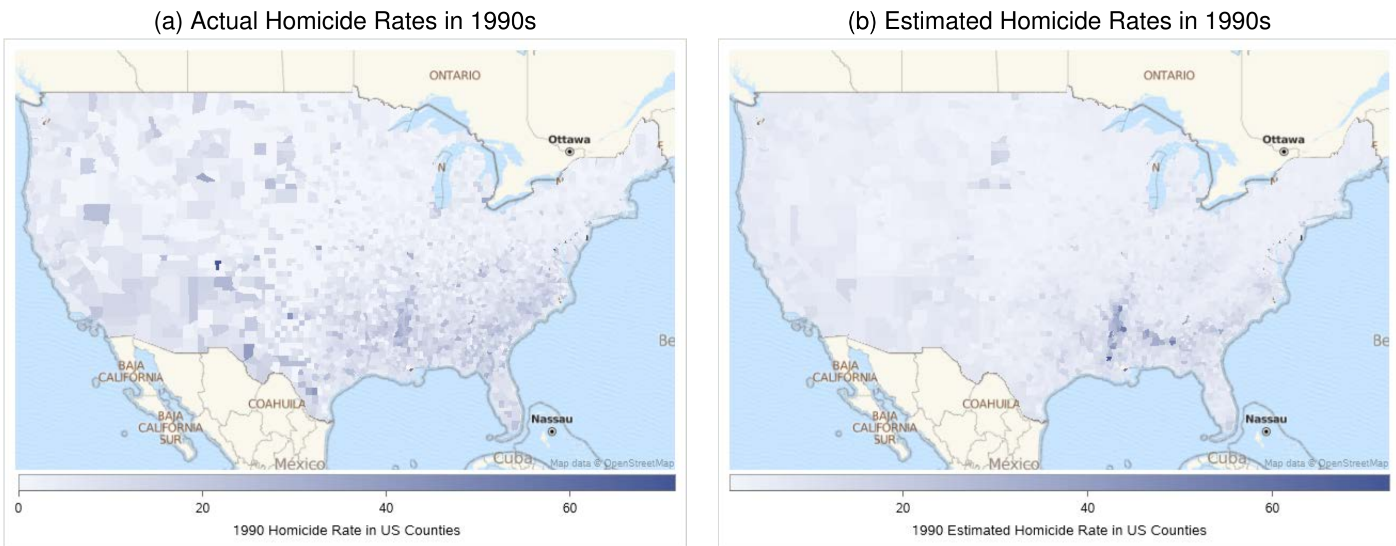
```
sgmap = sas.submit('''
                    %macro mapestm(var,title);;
                         proc sgmap mapdata=sgf2020.uscountymap
                                    maprespdata=sgf2020.uscountyhr90pred;
                             openstreetmap;
                             choromap &var / mapid=fips name='choro'
                                           lineattrs=(thickness=0px);
                             gradlegend 'choro' / title=&title
                             extractscale;
                         run;
                     %mend mapestm;
                     %mapestm(var=HR90,title='1990 Homicide Rate
                               in US Counties');
                     %mapestm(var=HR90Pred,title='1990 Estimated
                              Homicide Rate in US Counties'); ''')
```

Each of the preceding `%mapestm` macro calls produces a choropleth map of the specified quantity. In this macro, you also identify the variable you are mapping. The map data that are created by the MAPIMPORT procedure are used as the data to map the county polygons. The variable being mapped is present in the map response data. In addition to

mapping the results with SAS, you can also map the results in Python by using a few additional steps that involve using the **geometry** column in the **USCountyCrimes** table.

The preceding calls produce a map that shows a base map from OpenStreetMap that provides context for the estimates. Figure 5(a) shows the actual homicide rate in the 1990s. Figure 5(b) shows the estimated homicide rate from the preceding Poisson model that includes spatial random effects. The model appears to find estimates that are reasonably close to the actual homicide rate in the Southeastern United States. However, it underfits the actual rate in the mountainous western areas in the South, suggesting that a local or regional spatial model that distinguishes between different subregions would perhaps be more appropriate. Exploring these is beyond the scope of this paper.

Figure 5 Generalized Linear Geostatistical Modeling of US County Homicide Rates in the 1990s



Case Study 3: Property Tax Delinquency Modeling and Mapping in Philadelphia

Spatial Tax-Delinquency Process Modeling

This section shows you how to handle geospatial data available in Geospatial Javascript Object Notation GeoJSON (Butler et al. 2016). Suppose you are interested in understanding the level of property tax delinquency at any arbitrary location within a large city and are provided with GeoJSON data that contain locations of tax-delinquent properties with both street addresses and actual point locations. This case study uses the Philadelphia tax-delinquent property data available from opendataphilly.org (OpenDataPhilly 2019).[3]

Philadelphia Tax-Delinquent Property Data

The Philadelphia tax-delinquent property data are available from opendataphilly.org in three formats: ESRI shapefile, CSV, and GeoJSON. Because JSON is a popular data interchange format in situations that involve communication between machines via REST APIs, this section considers GeoJSON. You can read GeoJSON files in Python using the following steps:

```
PropTaxDelinq = geopd.read_file("PhillyData/real_estate_tax_delinquencies.geojson")
PropTaxDelinq = PropTaxDelinq.to_crs(epsg=26918)
PropTaxDelinq = PropTaxDelinq[['objectid','street_address',
                               'zip_code','zip_4',
                               'geometry','total_due',
                               'building_category',
                               'general_building_description',
                               'year_of_last_assessment']]
```

[3]The Philadelphia tax-delinquent property data can be downloaded from https://www.opendataphilly.org/dataset/property-tax-delinquencies.

This code uses the GeoPandas package to read and process the GeoJSON data. You read the GeoJSON file from the file system by using the **geopd.read_file()** function of the GeoPandas package, where **geopd** is the GeoPandas package object. The function reads the GeoJSON file and creates a GeoPandas data frame.

The GeoJSON file contains individual spatial locations as point geometries that are originally in the geodetic coordinate system. For spatial process modeling, you first reproject the data to a coordinate system suitable for Philadelphia. You can reproject the data by using the **to_crs** function in which you specify the EPSG coordinate reference identifier for Philadelphia, which is 26918. This identifier corresponds to North American Datum (NAD) 1983 and the Universal Transverse Mercator (UTM) projection zone 18N, which is suitable for the city of Philadelphia.

Next, you select only the attributes that are necessary for modeling the tax delinquency. In this subset, the first five variables correspond to identifiers of each property location; they include the street address, ZIP code, additional ZIP+4 code, and actual point geometry of each property. The next four variables contain information about individual properties such as the total tax amount due, the category of the building, a general description of the building (house, vacant land, and so on), and the year of last assessment.

Suppose you are interested only in tax-delinquent houses that were assessed in the year 2019 or later. You upload the data frame **PropertyTaxDelinquent** to SAS Viya and select only the observations that are required. Because SAS Viya does not support the **geometry** type directly, the type of the **geometry** column needs be converted to variable-length character in order to upload to SAS Viya. To produce a GeoDataFrame that contains 55,806 observations and includes all the variables you requested, you can prepare the data to be uploaded to SAS Viya by using the following steps:

```
TaxDelinqProp = PropTaxDelinq
TaxDelinqProp['X'] = PropTaxDelinq["geometry"].x
TaxDelinqProp['Y'] = PropTaxDelinq["geometry"].y
TaxDelinqProp["geometry"] = TaxDelinqProp["geometry"].astype(str)
```

The preceding steps extract the X and Y coordinates of each property and add them to the data frame that needs to be uploaded to SAS Viya. Also, the type of the **geometry** column is converted to variable-length character by using the **astype(str)** function as in the section "Case Study 2: Generalized Linear Geostatistical Modeling of County Homicides" on page 9. As in previous sections, Pandas data frames can be uploaded to SAS Viya by using the following steps:

```
hand2 = dmh.PandasDataFrame(TaxDelinqProp)
s.addtable(table='PhillyTaxDelinqProp',
           **hand2.args.addtable, replace=True)
```

The **addtable** action uploads the **TaxDelinqProp** data frame to SAS Viya and creates a new CAS table **Philly-TaxDelinqProp**. Once the data are uploaded to SAS Viya, you can use the **fedSQL** action to select a subset of the data to include only the residential properties that are houses whose latest assessment was in 2019. The following steps submit a query that creates a new CAS table named **ResTaxDelinq**:

```
s.builtins.loadActionSet("fedSql")
ResTaxDelinq = s.fedSql.execDirect(query='''create table
                                        ResTaxDelinq{options replace=true} as
                                        mean(total_due) as AvgTaxDue,
                                        street_address as address,
                                        zip_code as zip,
                                        X as orgX, Y as orgY
                                        from PhillyTaxDelinqProp where
                                        building_category like 'residential'
                                        AND general_building_description like 'house'
                                        AND year_of_last_assessment = 2019
                                        group by geometry, X, Y,
                                         street_address, zip_code ''')
ResTaxDelinq = s.CASTable("ResTaxDelinq")
```

The preceding query also groups properties by their location because it is quite likely that multiple properties have been geocoded to the same physical location. Also, while the grouping is performed, total tax due from such properties is summarized by averaging them. Now, before you analyze these data to model the spatial tax-delinquency rate and to assess how a smoothed process estimate of the average tax due in Philadephia affects it, you must first export the CAS table **ResTaxDelinq** to a SAS data set, as in the following steps:

```
ResTaxDelinqDF = s.fetch(table=table('ResTaxDelinq'),
                 maxRows=36901,to=36901)['Fetch']
ResTaxDelinqSD = sas.df2sd(ResTaxDelinqDF,
                           "PhillyResTaxDelinq","sgf2020")
ResTaxDelinqSDOrg = sas.df2sd(ResTaxDelinqDF,
                           "PhillyResTaxDelinqOrg","sgf2020")
```

These steps fetch all rows in the CAS table **ResTaxDelinq** and store them in the Pandas data frame **ResTaxDelinqSD**. Using the SASPy function `sas.df2sd`, you can then create the SAS data set **PhillyResTaxDelinq** in the library `sgf2020`. The preceding steps create two copies of the tax-delinquent homes: one is intended to use the **X** and **Y** locations directly, and the other is intended to use the street addresses directly, and derive the locations via geocoding.

Modeling the Spatial Rate of Tax Delinquency

Suppose you want a spatial point process model for the spatial rate at which properties become tax-delinquent at any arbitary location in Philadelphia. In SAS 9.4, you can use the SPP procedure to perform spatial point process modeling. For this example, the model uses only the average tax due from different properties as a predictor.

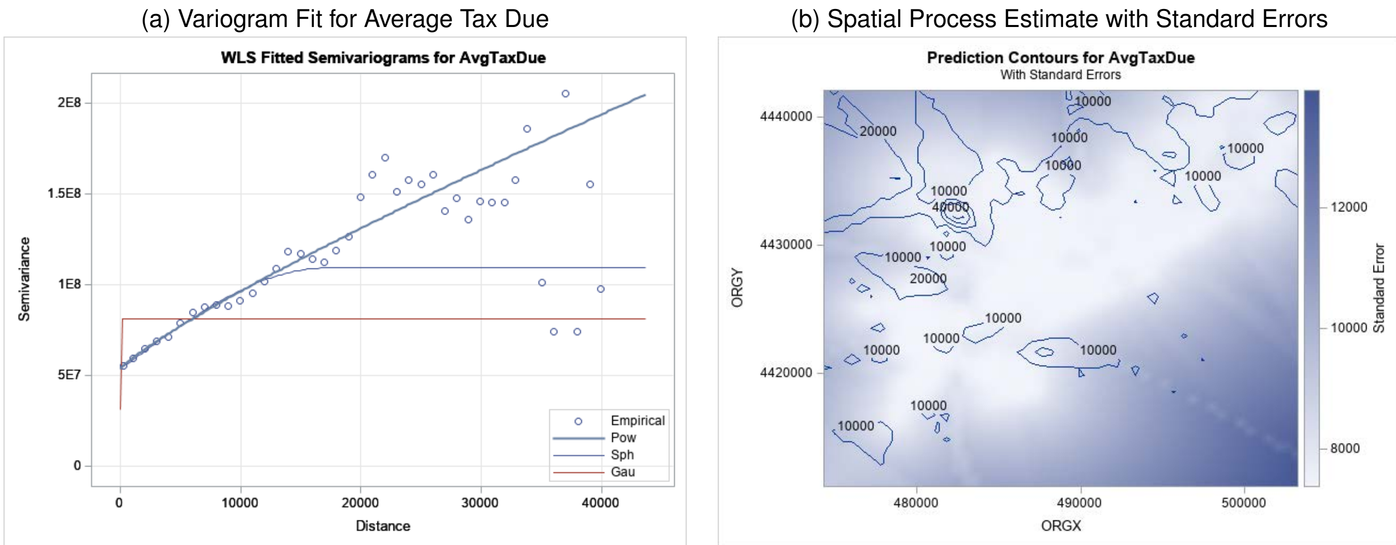
In order to use the average tax due as a spatial covariate to assess its influence on the spatial tax-delinquency rate, you must obtain an estimate of the tax due at any arbitary location in Philadelphia. You do this by fitting a geostatistical process model for the average tax due, using the steps described in the section "Geostatistical Smoothing of Global Precipitation" on page 7. The following steps combine variogram analysis with kriging to produce a smooth spatial process estimate of the average tax due in Philadelphia:

```
krigetax = sas.submit(''' ods select FitPlot;
                          proc variogram data=sgf2020.PhillyResTaxDelinqAggOrg
                                         plots=FitPlot;
                             store out=SemivDueStore / label= 'Average Tax Due Model';
                             coordinates xc = orgX yc=orgY;
                             compute lagd=1000 maxlag= 500;
                             var AvgTaxDue;
                             model form=auto(mlist=(gau,pow,sph) nest=1);
                          run;
                          ods select PredictionPlot;
                          proc krige2d
                                   data=sgf2020.PhillyResTaxDelinqAggOrg
                                   outest=sgf2020.PhillyTaxDuePred
                                   plots(only)=(pred);
                             restore in=SemivDueStore;
                             coordinates xc=orgX yc=orgY;
                             predict var=AvgTaxDue r=1000;
                             model storeselect;
                             grid x=474326.79 to 503246.54 by 578.395
                                  y=4411150.21 to 4442131.06 by 619.617;
                          run;
                          data sgf2020.PhillyTaxDuePred;
                             set sgf2020.PhillyTaxDuePred;
                             rename GXC=orgX GYC=orgY ESTIMATE=AvgTaxDue;
                             drop label npoints varname stderr;
                             Event = 0;
                          run;
                          proc sort data=sgf2020.PhillyTaxDuePred;
```

```
                        by orgX orgY;
                    run;''')
```

You first select the most appropriate spatial covariance structure for the variable **AvgTaxDue**. The call to PROC VARIOGRAM produces the fit plot shown in Figure 6(a) and also stores the covariance parameters in an item store. The next step performs ordinary kriging by using PROC KRIGE2D and the covariance parameters in the item store.

Figure 6 Spatial Process Estimate of Tax Due in Philadelphia



The fit plot in Figure 6(a) shows that PROC VARIOGRAM selects the Pow spatial covariance function. Figure 6(b) shows a contour plot of the estimated tax due in Philadelphia, which was obtained using ordinary kriging.

At this point, you have all the necessary inputs to model the spatial tax-delinquency rate in Philadelphia. You need to merge the smoothed process estimate of tax due with tax-delinquent property locations before you can use PROC SPP to fit a model for the spatial tax-delinquency rate, as in the following steps:

```
   procmerge = sas.submit('''data sgf2020.PhillyDelinqProcess;
                        merge sgf2020.PhillyResTaxDelinqAggOrg
                           sgf2020.PhillyTaxDuePred;
                        by orgX orgY;
                    run; ''')
```

The final step in modeling the spatial tax-delinquency rate is to fit a spatial point process model by using PROC SPP. To fit a nonhomogeneous spatial Poisson process model, include the process estimate of average tax due as a spatial trend covariate. In addition, you can also request that PROC SPP fit a first-order polynomial function of the coordinates X and Y, as in the following steps:

```
spptd = sas.submit(''' proc spp data=sgf2020.PhillyDelinqProcess
                           plots(unpack)=all;
                     process ptax = (orgX,orgY / Event = Event) /
                                              kernel(b=1000);
                     trend taxdue = field(orgX, orgY, AvgTaxDue);
                     model ptax = taxdue/ poly(1) residual(b=1000);
              run;''')
```

When you run the preceding steps, PROC SPP produces the Poisson parameter estimates table, a plot for the fitted spatial tax-delinquency rate, and a smoothed spatial residuals plot. Figure 7 shows the Poisson parameter estimates. The model seems to indicate that the rate of delinquency is lower for properties that have high values of tax due.

Figure 7 Poisson Parameter Estimates for Spatial Tax-Delinquency Rate

Poisson Parameter Estimates				
Parameter	**Estimate**	**Standard Error**	**z Value**	**Approx Pr > |z|**
Intercept	-96.0834	2.8805	-33.36	<.0001
ORGX	-0.00007	7.272E-7	-96.48	<.0001
ORGY	0.000027	6.624E-7	41.26	<.0001
AvgTaxDue	-0.00012	1.193E-6	-102.26	<.0001

Figure 8 Estimate of Spatial Tax-Delinquency Rate in Philadelphia

(a) Fitted Spatial Tax-Delinquency Rate (b) Spatial Residuals for Tax-Delinquency Rate

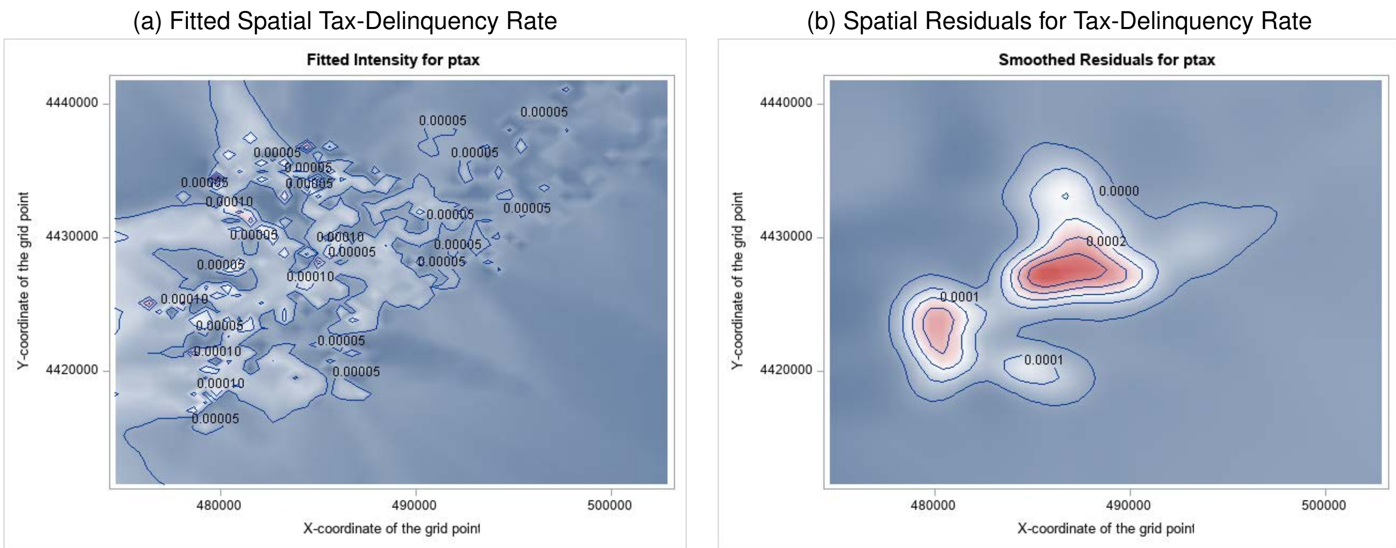


Figure 8(a) shows the fitted intensity rate that is based on the model that uses the process estimate of tax due and first-order polynomial trend surface as covariates.

Finally, Figure 8(b) shows the smoothed spatial residuals of the tax delinquency rate. Positive values of the spatial residuals indicate that the model is underestimating the actual spatial rate of tax delinquency. Areas where the value of the residual is 0 indicate that the model is able to capture the spatial tax-delinquency rate.

Figure 8(b) indicates that the model is clearly missing the high concentration of tax-delinquent properties at the center of Philadelphia. It is possible that the spatial process that relates delinquency to tax due is different across Philadelphia.

The analysis in this section used the original locations available with the Philadelphia data. However, it is possible that the data contain only street addresses instead of the actual locations. The next section discusses the impact of using street addresses over the actual locations.

Is Geocoding Street Addresses Helpful ?

The information about delinquent properties is shared and distributed as open data. Sometimes, the data-disseminating agencies provide only street addresses instead of the actual locations themselves.

To geocode the street addresses in **PhillyResTaxDelinq**, you can use the GEOCODE procedure in SAS 9.4 (SAS Institute Inc. 2018a). The following steps perform geocoding for the **PhillyResTaxDelinq** data set:

```
geocd = sas.submit('''  proc geocode
                           method = STREET
                           data=sgf2020.PhillyResTaxDelinq
                           out=sgf2020.PhillyResTaxDelinqGeocoded
                           lookupstreet=sashelp.GEOEXM;
                        run;''')
```

These steps perform street geocoding and output the geocoded street addresses to **PhillyResTaxDelinqGeocoded**. However, the geocoded data are obtained as longitude and latitude instead of projected **X** and **Y**. To perform spatial process modeling, you must reproject the data into projected coordinates. You can do this by using the GPROJECT procedure in SAS 9.4. However, geocoding can sometimes produce an over-summarization of the actual property locations. To assess this, you must sort the projected data by the coordinates and then perform an aggregation query by using the SQL procedure (SAS Institute Inc. 2018a). Creating a scatter plot of the aggregated locations via the SGPLOT procedure or the SGMAP procedure can reveal whether geocoding summarized the properties to a fewer set of locations. You can see the effect of gecoding by using the following steps:

```
gproj = sas.submit(''' proc gproject data=sgf2020.PhillyResTaxDelinqGeocoded
                         out=sgf2020.PhillyResTaxDelinqProj
                         from="EPSG:4326" to="EPSG:26918" dupok;
                         id;
                       run;
                       proc sort data=sgf2020.PhillyResTaxDelinqProj;
                         by X Y;
                       run;
                       proc sql;
                         create table sgf2020.PhillyResTaxDelinqAgg as
                         select mean(AvgTaxDue) as AvgTaxDue, X,Y
                         from sgf2020.PhillyResTaxDelinqProj
                         group by X,Y;
                       quit;
                       data sgf2020.PhillyResTaxDelinqAgg;
                         set sgf2020.PhillyResTaxDelinqAgg;
                         Event = 1;
                       run;
                       proc sgplot data=sgf2020.PhillyResTaxDelinqAgg;
                           title "Average Tax Due from Residential
                                  Properties (Geocoded, STREET)";
                           scatter X=X Y=Y/ colorresponse=AvgTaxDue
                                        colormodel=(yellow orange red)
                                        markerattrs=(symbol=circlefilled size=10px);
                           gradlegend /position=right;
                       run;''')
```

The preceding steps produce a map that shows the locations of residential properties and their average tax due, as shown in Figure 9(a). Figure 9(a) shows 46 locations as opposed to the original 36,901 locations shown in Figure 9(b), clearly indicating that geocoding has over-summarized the property location data. This is possible on the basis of the combination of the street address, ZIP code, and geocoding method that is provided to PROC GEOCODE.

When street addresses are used to produce the coordinates, **X** and **Y** are greatly influenced by the accuracy of the geocoder that was used. Using street address and ZIP code to geocode the locations explicitly before analysis might be appropriate if you have access to an accurate geocoding service. However, it is also possible that the geocoding performs poorly for the restricted street address data and reduces the number of property locations. Hence, it might not lead to accurate spatial delinquency rate estimates.

Figure 9 Comparing Geocoded and Original Property Locations

If your goal is to obtain a really accurate spatial delinquency rate estimate, comparing Figure 9(a) and Figure 9(b) reveals that using the original locations instead of the gecoded locations is more appropriate for spatial delinquency rate modeling.

Summary

Geospatial analysis of open and public-use data requires a practical approach that involves a combination of Python-based open-source software, SAS Viya, and SAS 9.4. When all these resources are available to a geospatial data scientist, they can provide great flexibility in data acess, transformation, analysis, modeling, and mapping. Handling geospatial data with street addresses requires caution and depends greatly on the amount of data available and the methodology used to geocode the addresses.

REFERENCES

Adler, R. F., Huffman, G. J., Chang, A., Ferraro, R., Xie, P.-P., Janowiak, J., Rudolf, B., et al. (2003). “The Version-2 Global Precipitation Climatology Project (GPCP) Monthly Precipitation Analysis (1979–Present).” *Journal of Hydrometeorology* 4:1147–1167.

Baller, R. D., Anselin, L., Messner, S. F., Deane, G., and Hawkins, D. F. (2001). “Structural Covariates of U.S. County Homicide Rates: Incorporating Spatial Effects.” *Criminology* 39:561–588.

Butler, H., Daly, M., Doyle, A., Gillies, S., Hagen, S., and Schaub, T. (2016). “The GeoJSON Format.” Published by the Internet Engineering Task Force (IETF). Accessed January 2020. https://tools.ietf.org/html/rfc7946.

GeoDa Data and Lab (2019). “NCOVR: US County Homicides 1960–1990.” Accessed January 2020. https://geodacenter.github.io/data-and-lab/.

GeoPandas developers (2019). “GeoPandas 0.6.0.” Accessed January 2020. http://geopandas.org/index.html.

Herring, J. R., ed. (2011). *OpenGIS Implementation Standard for Geographic Information—Simple Feature Access—Part 1: Common Architecture, Version 1.2.1*. Wayland, MA: Open Geospatial Consortium. http://dx.doi.org/10.25607/OBP-630.

Messner, S., Anselin, L., Hawkins, D., Deane, G., Tolnay, S., and Baller, R. (2000). *An Atlas of the Spatial Patterning of County-Level Homicide, 1960–1990*. Pittsburgh: National Consortium on Violence Research (NCOVR).

Met Office (2010–2015). “Cartopy: A Cartographic Python Library with a Matplotlib Interface.” Accessed January 2020. http://scitools.org.uk/cartopy.

OpenDataPhilly (2019). “Property Tax Delinquencies.” Accessed January 2020. https://www.opendataphilly.org/dataset/property-tax-delinquencies.

Pandas developers (2019). “The Pandas Project.” Accessed January 2020. https://pandas.pydata.org/about.html.

Phillips, J. (2019). “A Complete Introduction to SASPy and Jupyter Notebooks.” In *Proceedings of the SAS Global Forum 2019 Conference*, 1–9. Cary, NC: SAS Institute Inc.

PROJ contributors (2020). *PROJ coordinate transformation software library*. Open Source Geospatial Foundation.

SAS Institute Inc. (2018a). *Base SAS 9.4 Procedures Guide*. 7th ed. Cary, NC: SAS Institute Inc. https://go.documentation.sas.com/?docsetId=proc&docsetTarget=titlepage.htm&docsetVersion=9.4&locale=en.

SAS Institute Inc. (2018b). *SAS/STAT 15.1 User's Guide*. Cary, NC: SAS Institute Inc. http://go.documentation.sas.com/?docsetId=statug&docsetTarget=titlepage.htm&docsetVersion=15.1&locale=en.

Smith, K. D., and Meng, X. (2017). *SAS Viya: The Python Perspective*. Cary, NC: SAS Institute Inc.

Unidata (2019). “netCDF4 Module, Version 1.5.3.” Accessed January 2020. https://unidata.github.io/netcdf4-python/netCDF4/index.html.

xarray developers (2019). “xarray: N-D Labeled Arrays and Datasets in Python.” Accessed January 2020. http://xarray.pydata.org/en/stable/.

ACKNOWLEDGMENTS

The author thanks Randy Tobias, Bucky Ransdell, and Fang Chen for their technical comments and is grateful to Anne Baxter for her editorial assistance.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author:

Pradeep Mohan
SAS Institute Inc.
SAS Campus Drive
Cary, NC 27513
Pradeep.Mohan@sas.com

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

Other brand and product names are trademarks of their respective companies.