The Stata Journal (*yyyy*) *vv*, Number *ii*, pp. 1–30

Reading and processing geographical raster data in Stata

Kerui Du
School of Management
Xiamen University
Xiamen, China
kerrydu@xmu.edu.cn

Chunxia Chen
School of Management
Xiamen University
Xiamen, China
chenchunxia@stu.xmu.edu.cn

Shuo Hu
School of Economics
Southwestern University of Finance and Economics
Chengdu, China
advancehs@163.com

Yang Song
School of Economics
Hefei University of Technology
ss0706082021@163.com

Ruipeng Tan
School of Economics
Hefei University of Technology
tanruipeng@hfut.edu.cn

Abstract. We integrate the Java GeoTools and NetCDF libraries into Stata and develop a new suite of Stata commands to read and process geographical raster data. These new functions enable Stata users to seamlessly extract data from GeoTIFF and NetCDF files, reproject geographic coordinates, match raster data with geographical locations, and compute zonal statistics. This integration supercharges Stata's data analysis capabilities, allowing for more in-depth exploration and understanding of geographic information. The syntax for these commands, along with practical examples, is detailed in the paper, helping users quickly grasp how to leverage these powerful tools to their full potential in real-world scenarios.

Keywords: GeoTIFF, NetCDF, Java, Raster, Geospatial

1 Introduction

Geospatial data has become indispensable in a variety of fields, including environmental science, economics, and public health. Increasingly, research relies on the combination of spatial information (e.g. Raster, Shapefile) with statistical modeling. To name a few, Burke et al. (2015) uses satellite remote sensing data to develop a global agricultural climate risk index, quantifying the nonlinear effect of extreme temperatures on crop yields and providing crucial evidence for climate adaptation policies. Henderson et al. (2012) develop a statistical framework for using satellite data on night lights to supplement official income growth measures. Using data from the Defense Meteorological Satellite Program (DMSP), they aggregated light intensity at various spatial scales to analyze economic activity. To explore the impacts of agricultural straw burning on air pollution and health in China, He et al. (2020) uses satellite data to detect straw burning and match it with air quality data and death records, providing critical evidence to evaluate the effectiveness of straw recycling policies. To estimate the costs and benefits of the national pollution monitoring and disclosure program, Barwick et al. (2024) overcome the challenge of limited availability of air pollution data by obtaining ambient air quality indicators from aerosol optical depth (AOD) using the Moderate Resolution Imaging Spectroradiometer (MODIS) algorithm.

Although Stata excels in traditional data processing and statistical inference, its native capabilities for handling geographic data remain limited. Stata is currently unable to import commonly used geodata files such as GeoTIFF and NetCDF formats. Statistical analysis with geospatial data requires users to preprocess geospatial data in external software (e.g., ArcGIS, QGIS, R, or Python) before importing the results into Stata, resulting in a fragmented and inefficient workflow. This limitation not only affects the reproducibility of the study, but also increases the complexity of the workflow and reduces the potential of Stata for geospatial data analysis.

This paper aims to overcome these limitations by leveraging the Java integration feature introduced in Stata 17 [1]. Taking advantage of this functionality, we integrate the Java GeoTools and NetCDF libraries [2] into Stata. Subsequently, we develop a suite of Stata commands to read and process raster data. To be specific, we focus on the two widely used geospatial data formats: GeoTiff and NetCDF. Section 2 provides essential background on these data formats and the technical approach that underlies our packages. It should be noted that, apart from GeoTiff and NetCDF, there are other data formats in the geospatial domain, such as ASCII, RINEX, and OGC KML, etc. For more technical details, readers can refer to the ESDS Standards (https://www.earthdata.nasa.gov/about/standards).

There are eight new Stata commands are introduced: `geotools_init`, `gtiffdisp`, `gtiffread`, `netcdf_init`, `ncdisp`, `ncread`, `crsconvert`, `gzonalstats` and `matchgeop`. `geotools_init` and `netcdf_init` are used to configure the necessary Java dependencies, making the underlying GeoTools and NetCDF functionalities available within Stata. `gtiffdisp` and `ncdisp` are used to display metainformation of the raster data. `gtiffread` and `ncread` are used to read the raster data and vectorize it into Stata. Furthermore, the packages provide commands for common spatial operations: `crsconvert` converts projected coordinates to ensure consistency between different spatial datasets, `gzonalstats` calculates raster data statistics (e.g., regional mean, standard deviation, minimum, maximum, etc.) for specified administrative or geographic regions, and `matchgeop` finds nearest neighbors based on geographic location. These commands allow Stata users to work directly with GeoTIFF or NetCDF raster files to obtain the data information they need. The subsequent sections of this paper will elaborate on the syntax of the developed commands and present specific application cases through real-world research scenarios.

1. The Java integration offers a JShell-like environment where developers can run Java code directly within Stata.
2. GeoTools is an open-source Java library for geospatial data processing, providing tools to manipulate, analyze, and visualize spatial datasets. NetCDF is an open source toolkit for scientific data processing, enabling cross-platform reading, writing, and analysis of NetCDF/HDF5 datasets

2 A brief introduction of GeoTIFF and NetCDF

2.1 The GeoTIFF Format

GeoTIFF is a geospatial image format based on the TIFF (Tagged Image File Format) standard, and extends the TIFF format by embedding georeferencing information (CRS, resolution, etc.). It represents spatial data as pixels in a grid and supports multiple bands, typically organized by band sequence (BSQ).

The GeoTIFF format has become the industry image standard for GIS and satellite remote sensing applications, and there is strong software support in both the open source and commercial GIS and spatial data analysis software products.It is ideal for 2D or 3D (multiband) raster data like high-resolution imagery and Digital Elevation Models (DEMs).

However, GeoTIFF is primarily image-centric and less flexible for complex multidimensional scientific datasets (e.g., time series, depth). It can be inefficient for sparse data and its structure is mainly limited to 2D layers per band. So when applied to store sparse data in meteorological and oceanographic fields, particularly the thin strip data generated by orbiting satellites, the format will store a large number of invalid and repetitive values, wasting storage space. Furthermore, meteorological and oceanographic data exist primarily in floating-point form. When recording high-resolution, large-scale, multiband floating-point data, GeoTIFF files significantly increase in size, consuming substantial storage space.

2.2 The NetCDF Format

NetCDF (Network Common Data Form) is a machine-independent, array-oriented format that uses dimensions, variables (multidimensional arrays) and attributes to structure data.

Different from GeoTIFF, NetCDF is highly effective for storing and managing complex, multidimensional scientific datasets like climate model outputs and time series. Its flexible structure accommodates various dimensions (spatial, temporal, etc.) and rich metadata.In addition, it is highly interoperable, supporting many libraries and tools such as Python (netCDF4, xarray), MATLAB and GIS software (e.g., QGIS and ArcGIS).

However, there are some drawbacks.First,the data content itself and spatial information are divided, which is not conducive to real-time data computation and sharing. Second, data organization can vary between providers, so that researchers encountering new data often need to study the structure of the data. Third, the data recorded in the file may not be the original data, which will take additional processing time when used for presentation and computation.

2.3 Comparison of the two formats

Individual time slices of the same type of data, such as digital elevation models (DEMs) or climate data (e.g., average temperature for a given year), are usually available in both GeoTIFF and NetCDF formats. The choice of which one to use depends greatly on the intended purpose and workflow. If the primary use involves time series analysis, the use of multivariate scientific models, or the need for large amounts of flexible metadata, NetCDF is usually the more appropriate and efficient format. If the data are primarily a single spatial layer or image and efficient image compression is required, GeoTIFF is preferred. netCDF is used for complex scientific modeling and analysis, while GeoTIFF is used for spatially oriented visualization and mapping.

The diversity of these formats presents challenges for statistical software like Stata, which may not have native built-in support for all of them. To enable Stata users to work with NetCDF and GeoTIFF data, external libraries are necessary. This is typically achieved by leveraging Stata's plugin framework (e.g., the Java plugin) to interact with powerful third-party libraries, such as Unidata's NetCDF-Java library for NetCDF or libraries like GeoTools Java bindings for GeoTIFF. These libraries provide the necessary functionalities for file parsing, data reading, and geospatial computations that can then be accessed using Stata commands.

3 Environment Setup Commands

3.1 The geotools_init command

Syntax

`geotools_init` [*pathtojar*]`,` [`download` `dir(`*path*`)` `plus(`*foldername*`)`]

pathtojar indicates is the file path of the GeoTools 32.0 JAR files. It will be added to Stata's adopath once it is specified.

Options

`download` specifies downloading GeoTools 32.0 library from SourceForge.

`dir(`*path*`)` specifies the directory where GeoTools should be downloaded (defaults to current directory)

`plus(`*foldername*`)` copies GeoTools 32.0 JAR files to the specified folder in Stata's sysdir PLUS adopath

Description

The developed package relies on Java Geotools Version 32.0 library. The `geotools_init` command is designed to set up the environment of Java dependencies. Stata users can manually download the Geotools Version 32.0 library from the website and specify the directory containing the Geotools version 32.0 library followed by geotools_init. Alternatively, the download option indicates downloading the Geotools Version 32.0 library within Stata and adding the corresponding directory to the Stata adopath. Stata users only need to perform one initialization and then Stata will be permanently set up.

3.2 The netcdf_init command

Syntax

`netcdf_init [`*path*`],` [`download dir(`*path*`) plus(`*foldername*`)`]

path indicates the directory that houses the Java library netcdfAll-5.6.0.jar. It will be added to Stata's adopath once it is specified.

Options

`download` specifies downloading netcdfALL-5.6.0.jar within Stata.

`dir(`*path*`)` specifies the target directory for downloading netcdfALL-5.6.0.jar. By default, netcdfALL-5.6.0.jar is downloaded to the current directory.

`plus(`*foldername*`)` specifies copying netcdfALL-5.6.0.jar to the folder in Stata's sysdir PLUS adopath.

Description

The developed package relies on the Java NetCDF library version 5.6.0. The `netcdf_init` command serves to configure the environment for these Java dependencies. Stata users have two ways to handle the setup. The first method involves manual download. Users can go to the Unidata Home website (https://www.unidata.ucar.edu/software/netcdf) and manually fetch the `netcdfALL-5.6.0.jar` file. Once downloaded, they can use the `netcdf_init` command to specify the directory where this `netcdfALL-5.6.0.jar` file is stored. The second option is the automated download. By choosing this, the `netcdfALL-5.6.0.jar` file will be downloaded directly within Stata and the corresponding directory will automatically be added to the Stata adopath. Stata users only need to perform this initialization process once. Once done, Stata will be configured permanently, ensuring smooth and continuous usage of the package.

4 Metadata Inspection Commands

4.1 The gtiffdisp command

Syntax

`gtiffdisp` *filename*

filename specifies the GeoTIFF file to be read.

Store results

`gtiffdisp` stores the following in `r()`:

`scalar`

`r(nband)` returns the number of bands.

`r(ncol)` returns the number of columns (width).

`r(nrow)` returns the number of rows (height).

`r(minX)` returns the minimum X coordinate.

`r(minY)` returns the minimum Y coordinate.

`r(maxX)` returns the maximum X coordinate.

`r(maxY)` returns the maximum Y coordinate.

`r(Xcellsize)` returns the X resolution (pixel size).

`r(Ycellsize)` returns the Y resolution (pixel size).

Description

`gtiffdisp` command is used to display the information from GeoTIFF files including dimensions,bands, spatial characteristics, coordinate systems, and other metadata.

4.2 The ncdisp command

Syntax

To view metadata for an entire NetCDF file:

`ncdisp using` *path_to_ncfile*

To view metadata for a specific variable in a NetCDF file:

`ncdisp` *varname* `using` *path_to_ncfile* *varname* specifes varable in the nc file.

path_to_ncfile specifies the nc file to be read.

Store results

`ncdisp` stores the following in `r()`:

`local`

`r(`*varname)* returns the name of the variable.

`r(`*dimensions)* returns the dimensions of the variable.

`r(`*coordinates)* returns the coordinate axes of the variable.

`r(`*datatype)* returns the datatype of the variable.

Description

`ncdisp` is used to display information about a specific variable in a NetCDF file. It reads the file path provided as an argument.

5 Data Import Commands

5.1 The gtiffread command

Syntax

`gtiffread` *filename* [`, clear band(`*#*`) origin(`*numlist*`) size(`*numlist*`)`
`crscode(`*string*`)`]

filename specifies the GeoTIFF file to be read.

Options

`clear` specifies clearing the current dataset to import data.

`band(`*#*`)` specifies which band to read from the GeoTIFF, Default is 1.

`origin(`*numlist*`)` specifies the starting position for reading ((row col)). And it must be used together with the size option. `origin(2 3)` begin the reading process from the cell located at the intersection of the second row and the third column. By default, it starts from the beginning (1,1).

`size(`*numlist*`)` determines the number of rows and columns to read. When a value of -1 is used, it means reading the data until reaching the end of the corresponding dimension. `size(3 -1)` means reading 3 elements in the row and reading elements in the column from the starting index until the end. If not specified, the entire raster from the origin to the end will be read.

`crscode(`*string)* specifies the target coordinate reference system (CRS) code. Can be:

- "EPSG:code" (e.g., "EPSG:4326" for WGS84)

- Path to a .tif file with the desired CRS

- Path to a .shp file with the desired CRS

If not specified, it uses the GeoTIFF's native CRS.

Description

The `gtiffread` command is used to extract pixel values with their corresponding coordinates.The command supports reading specific bands, subsetting by origin and size, and optional coordinate system conversion.

5.2 The ncread command

Syntax

`ncread [`*varname*`] using` *path_to_ncfile*`,` [`clear csv(`*filename* `[,`*replace*`])` `origin(`*numlist*`) size(`*numlist*`)`]

varname indicates reading the data of the specified varname in the nc file. If not specified, the command would display the metadata in the nc file.

path_to_ncfile specifies the nc file to be read.

Options

`clear` specifies clearing the current dataset.

`csv(`*filename*`[,`*replace*`])` specifies reading the data from the nc file into the specified CSV file. If *replace* is specified, the existing CSV file will be replaced.

`origin(`*numlist*`)` specifies the starting indices of each dimension for reading a section of raster data from the nc file. `origin(3 2 1)` means beginning with the third element in the first dimension, the second element in the second dimension, and the first element in the third dimension.

`size(`*numlist*`)` determines the reading count for each dimension. When a value of -1 is used, it means reading the data until reaching the end of the corresponding dimension. `size(1 3 -1)` means reading one and three elements in Dimensions 1 and 2, respectively, and reading the elements in Dimension 3 from the starting index until the end.

Description

The `ncread` command is used to read data for a specified variable from a NetCDF file into Stata. It can read the entire variable or a specified section of data. If varname is not specified, ncread displays the meta-information of the NetCDF.

6 Spatial Operation Commands

6.1 The crsconvert command

Syntax

`crsconvert` *varlist* $\Big[$, `gen(`*string) from(string) to(string)*$\Big]$

varlist must contain exactly two numeric variables representing x and y coordinates in the source coordinate reference system.

Options

`gen(`*perfix_)* specifies the prefix for the two new variables that will contain the transformed coordinates. The new variables will be named $prefix\_x$ and $prefix\_y$.

`from(`*string* `[,` *tif shp*`])` specifies the source coordinate reference system. It might be provided in EPSG format(e.g. ”EPSG:4326”). The coordinate reference system can be provided in EPSG format (e.g., ”EPSG:4326”). Alternatively, users have the option to specify a GeoTIFF file (SHP file) using the *tif* (*shp*) option to denote the source of the coordinate reference system. This approach is highly practical, considering that the coordinate reference system sometimes may not conform to a standard EPSG format and instead contains detailed projection parameters. If a file is specified, the command automatically identify its coordinate reference system.

`to(`*string* `[,` *tif shp*`])` specifies the source coordinate reference system. It might be provided in EPSG format(e.g. ”EPSG:4326”). Alternatively, users have the option to specify a GeoTIFF file (SHP file) using the *tif* (*shp*) option to denote the source of the coordinate reference system.

Description

The `crsconvert` command converts coordinates from one coordinate reference system to another. It creates two new variables storing the transformed coordinates.

6.2 The gzonalstats command

Syntax

`gzonalstats` *tif_file* `using` *shpfile*`,` [`stats(`*string*`)` `band(`#`)` `clear`]

tif_file specifies the GeoTIFF file to be read.

shpfile specifies a shpfile to read. It requires a complete shapefile set (.shp, .shx, .dbf and .prj files) in the same folder.

Options

`clear` specifies clearing the current data set to import data.

`band(`#`)` specifies which raster band to use in multi-band raster files. Default is 1 (first band).

`stats(`*string*`)` specifies which statistics to calculate. Default is "count avg min max" if not specified. Valid options are:

count: the number of pixels in the zone

avg: the average pixel value

min: the minimum pixel value

max: the maximum pixel value

std: the standard deviation of pixel values

sum: the sum of pixel values

Description

The `gzonalstats` command calculates zonal statistics for each polygon in a shapefile based on the values of a raster dataset (GeoTIFF) that fall within each polygon. This command supports automatic coordinate system transformation when the shapefile and raster have different projections, and optimizes raster reading by only processing the portion that intersects with the shapefile extent.

6.3 The matchgeop command

Syntax

`matchgeop` *baseid baselat basselon* `using` *nborfile*`,` `neighbors(`*nborid nborlat nborlon*`)` [`uframe` `within(`#`)` `mile` `gen(`*newvar*`)` `nearcount(`#`)` `bearing(`*newvar*`)` `nsplit(`#`)`]

Options

`neighbors(`*varlist*`)` specifies the neighbor variables to consider for matching. Must contain 3 variables in the order: id, latitude, and longitude.

`uframe` specifies the using data is in a frame. By default, the using data is Stata data format in the disk.

`within(`*#*`)` specifies the range within which neighbors are considered.

`mile` specifies that distances should be measured in miles. Default is kilometers.

`nearcount(`*#*`)` specifies the number of nearest neighbors to consider. Must be a positive number.

`gen(`*newvar*`)` specifies the name of the new variable to store the distance.

`bearing(`*newvar*`)` specifies the name of the new variable to store the bearing.

`nsplit(`*#*`)` specifies the number of splits for looping. nsplit = 1 does one loop with all observations (requires large memory), nsplit = 2 does two loops with half of the observations, and so on.

Description

`matchgeop` merges datasets based on a specified variable list and neighbors. It finds the nearest neighbors based on geographic location and generates a linked dataset.

7 Examples

7.1 Set up the Java dependence

When using the `gtiffdisp`, `gtiffread`, `crsconvert`, and `gzonalstats` commands in Stata, users need to initialize the Java dependencies upon their first use. The `geotools_init` command enables Stata to download GeoTools version 32.0.

However, due to the substantial size of the GeoTools package, the download process may be rather time - consuming. To save time, we suggest that users manually download the GeoTools 32.0 package from Sourceforge [3] and unzip the zipped file. Then, use the `geotools_init` command to specify the path to "geotools-32.0/lib", as demonstrated in the following code. This will generate the `path_geotoolsjar` command to record the specified path.

It's important to note that if the files in "geotools-32.0/lib" are moved, the setup process will need to be repeated.

```
. geotools_init "C:/Users/17286/Documents/geotools-32.0/lib/"
(file C:\Users\17286\ado\plus/p/path_geotoolsjar.ado not found)
```

3. https://sourceforge.net/projects/geotools/files/GeoTools%2032%20Releases/32.0/

Similarly, to use the `ncread` and `ncdisp` commands, Stata users need to initialize the Java dependencies upon their first-time use. Using the `netcdf_init`, we downlaod netcdfAll-5.6.0.jar within Stata and copy it to the folder jar in Stata sysdir PlUS.

```
. netcdf_init, download plus(jar)
Downloading netcdfAll-5.6.0.jar into C:\Users\17286\Documents...
Please wait...
Copying netcdfAll-5.6.0.jar to \Users\17286\ado\plus//jar ...
```

7.2 Display the Metadata

This section demonstrates how to display the metadata of a GeoTiff file and a NetCDF file.

Display the Metadata of the GeoTIFF File

For the GeoTiff file, the metadata can be viewed using the `gtifdisp` command, which provides detailed information containing Band Information, Spatial Characteristics, Coordinate System, Units, and Filtered Metadata.Take DMSP-like2020.tif as an example to show the usage. This dataset contains improved DMSP-OLS-like data in China by integrating DMSP-OLS (Defense Meteorological Satellite Program - Operational Linescan System) and SNPP-VIIRS (Suomi National Polar - orbiting Partnership - Visible Infrared Imaging Radiometer Suite).

```
. gtiffdisp DMSP-like2020.tif
=== Band Information ===
Number of bands: 1
Band 1 : GRAY_INDEX           | NoData: Not defined
=== Spatial Characteristics ===
X range: [-2643772.8565 ~ 2212227.1435]
Y range: [1871896.5263 ~ 5926896.5263]
Resolution: X=1000.0000 units/pixel, Y=1000.0000 units/pixel
=== Coordinate System ===
CRS Name: PCS Name = WGS_1984_Albers
CRS WKT: PROJCS["PCS Name = WGS_1984_Albers",
  GEOGCS["WGS 84",
    DATUM["World Geodetic System 1984",
      SPHEROID["WGS 84", 6378137.0, 298.257223563, AUTHORITY["EPSG","7030"]
> ],
      AUTHORITY["EPSG","6326"]],
    PRIMEM["Greenwich", 0.0, AUTHORITY["EPSG","8901"]],
    UNIT["degree", 0.017453292519943295],
    AXIS["Geodetic longitude", EAST],
    AXIS["Geodetic latitude", NORTH],
    AUTHORITY["EPSG","4326"]],
  PROJECTION["Albers_Conic_Equal_Area"],
  PARAMETER["central_meridian", 105.0],
  PARAMETER["latitude_of_origin", 0.0],
  PARAMETER["standard_parallel_1", 25.0],
  PARAMETER["false_easting", 0.0],
  PARAMETER["false_northing", 0.0],
  PARAMETER["standard_parallel_2", 47.0],
    UNIT["m", 1.0],
    AXIS["Easting", EAST],
    AXIS["Northing", NORTH]]
=== Units ===
X unit: m
Y unit: m
=== Filtered Metadata ===
image_height               : 4055
image_min_x_coord          : 0
image_width                : 4856
image_min_y_coord          : 0
```

The output provides detailed information of the specified GeoTIFF file DMSP-like2020.tif. First, it includes one band raster named "GRAY_INDEX" and the nodata value in this band is not defined. Second, the coordinate system points out that the CRS is identified as "WGS_1984_Albers", which utilizes the Albers Equal Area Conic projection. The Well-Known Text (WKT) representation provides detailed parameters of the projection. Third, the spatial information indicates that the X range of the dataset extends from approximately -2,643,772.86 to 2,212,227.14 meters, while the Y range spans from 1,871,896.53 to 5,926,896.53 meters. The resolution is set to 1000 meters per pixel for both the X and Y directions, providing a detailed representation of the area.

Display the Metadata of the NetCDF File

For the NetCDF file, we can use the `ncread` or `ncdisp` command to inspect the metadata of the file, and `ncdisp` to inspect the metadata of the variables.

We provide the following example to show how to read the metadata of a nc file of The NEX-GDDP-CMIP6 dataset located in Amazon web service (AWS).The NEX-GDDP-CMIP6 dataset is comprised of global downscaled climate scenarios. These scenarios are derived from the General Circulation Model (GCM) runs conducted under the Coupled Model Intercomparison Project Phase 6 (CMIP6) and across two of the four "Tier 1" greenhouse gas emissions scenarios known as Shared Socioeconomic Pathways (SSPs). For this example, we use the file tas_day_BCC-CSM2-MR_ssp245_r1i1p1f1_gn_2050.nc. The output provides details about global attributes, dimensions, variables, and so on.

```
. local url = "https://nex-gddp-cmip6.s3-us-west-2.amazonaws.com/" + ///
> "NEX-GDDP-CMIP6/BCC-CSM2-MR/ssp245/r1i1p1f1/tas/" + ///
> "tas_day_BCC-CSM2-MR_ssp245_r1i1p1f1_gn_2050.nc"
. ncdisp using `"`url´"´
=== File Structure ===
[Global Attributes]
_NCProperties            : version=1|netcdflibversion=4.4.1.1|hdf5libversi...
activity                 : NEX-GDDP-CMIP6
contact                  : Dr. Rama Nemani: rama.nemani@nasa.gov, Dr. Brid...
Conventions              : CF-1.7
creation_date            : 2021-10-04T14:25:12.081609+00:00
frequency                : day
institution              : NASA Earth Exchange, NASA Ames Research Center,...
```

```
variant_label           : r1i1p1f1
product                 : output
realm                   : atmos
source                  : BCSD
scenario                : ssp245
references              : BCSD method: Thrasher et al., 2012, Hydrol. Ear...
version                 : 1.0
tracking_id             : 6ab2a874-d3fe-484a-a7e8-82df448e364f
title                   : BCC-CSM2-MR, r1i1p1f1, ssp245, global downscale...
resolution_id           : 0.25 degree
history                 : 2021-10-04T14:25:12.081609+00:00: install globa...
disclaimer              : This data is considered provisional and subject...
external_variables      : areacella
cmip6_source_id         : BCC-CSM2-MR
cmip6_institution_id    : BCC
cmip6_license           : CC-BY-SA 4.0
_CoordSysBuilder        : ucar.nc2.dataset.conv.CF1Convention
[Dimensions]
Name                Length   Attribute
time                365      Coordinate
lat                 600      Coordinate
lon                 1440     Coordinate
[Variables]
Name                     Dimensions                     Type
tas                      time lat lon                   float
time                     time                           double
lat                      lat                            double
lon                      lon                            double
[Groups]
```

Next, we employ `ncdisp` for inspecting the attributes of variable tas included in tas_day_BCC-CSM2-MR_ssp245_r1i1p1f1_gn_2050.nc.The generated outputs indicate that the variable tas is the daily near-surface air temperature in units of Kelvin (K). It has three dimensions, time, lat, and lon with shape $(365, 400, 1440)$. The missing value is filled with a value of $1.0E20$.

```
. local url = "https://nex-gddp-cmip6.s3-us-west-2.amazonaws.com/" + ///
> "NEX-GDDP-CMIP6/BCC-CSM2-MR/ssp245/r1i1p1f1/tas/" + ///
> "tas_day_BCC-CSM2-MR_ssp245_r1i1p1f1_gn_2050.nc"
. ncdisp tas using `"`url´"´

=== Variable Structure ===
Name: tas                      Type: float
=== Dimensions ===
Dimension       Length   Coordinate
time            365      [Yes]
lat             600      [Yes]
lon             1440     [Yes]
=== Scale/Offset Parameters ===
missing value  : 100000002004087730000.000000 (Type: float)
 FillValue     : 100000002004087730000.000000 (Type: float)
=== Attributes ===
missing_value       : 1.0E20
cell_measures       : area: areacella
cell_methods        : area: mean time: maximum (interval: 5 minutes)
```

```
original_name       : TREFHTMX
comment             : near-surface (usually, 2 meter) air temperature; derived from d
> ownscaled tasmax & tasmin
units               : K
long_name           : Daily Near-Surface Air Temperature
standard_name       : air_temperature
_FillValue          : 1.0E20
_ChunkSizes         : 1
=== Metadata ===
Units          : K (original: K)
Shape          : [365, 600, 1440]
Data Type      : float
```

We also display the meta data of variable *time*. Note that this variable has units defined as "days since 2040-01-01". This indicates that the *time* dimension is measured in days elapsed from the reference date of January 1, 2040, and each value represents the number of days (and fractions thereof) since this origin point.

```
. local url = "https://nex-gddp-cmip6.s3-us-west-2.amazonaws.com/" + ///
> "NEX-GDDP-CMIP6/BCC-CSM2-MR/ssp245/r1i1p1f1/tas/" + ///
> "tas_day_BCC-CSM2-MR_ssp245_r1i1p1f1_gn_2050.nc"
. ncdisp time using `"`url´"´

=== Variable Structure ===
Name: time                      Type: double
=== Dimensions ===
Dimension       Length   Coordinate
time            365      [Yes]
=== Scale/Offset Parameters ===
=== Attributes ===
units               : days since 2040-01-01
standard_name       : time
long_name           : time
calendar            : 365_day
axis                : T
_ChunkSizes         : 365
_CoordinateAxisType : Time
=== Metadata ===
Units          : days since 2040-01-01 (original: days since 2040-01-01)
Shape          : [365]
Data Type      : double
```

7.3 Import Raster Data into Stata

Importing raster data into stata uses two main commands—— `ncread` for netCDF files and `gtiffread` for GeoTIFF files.

Read the GeoTIFF file for a specific region

The following example demonstrates how to extract nightlight data for the year 2020 in a region encompassing Hunan from the GeoTIFF file DMSP-like2020.tif with `gtiffread` command. The first step is to determine the latitude and longitude range of Hunan Province. As shown in Example 7.2, the coordinate system of the DMSP-like2020.tif file is "WGS_1984_Albers". Therefore, we need convert the coordinate system of the hunan.shp to "WGS_1984_Albers" with the `crsconvert` command. This allows us to obtain the coordinate range of Hunan Province in the "WGS_1984_Albers" coordinate system.

To guarantee the extracted raster data fully covers the area of Hunan Province, the boundary is expanded by ±2000 meters. Using this adjusted coordinate range, we calculate the starting index and dimensions of the relevant region in the DMSP-like2020.tif. We read the first column of the raster with the option `origin(1 1) size(-1 1)` to identify the vertical range (y-axis) of the data. Similarly, we read the first row of the raster with the option `origin(1 1) size(1 -1)` to identify the horizontal range (x-axis) of the data.

```
. shp2dta using "hunan.shp", database(hunan_db) coordinates(hunan_coord) genid(id)
type: 5

. use "hunan_coord.dta",clear

. drop if missing(_X, _Y)
(14 observations deleted)

. crsconvert _X _Y, gen(alber_) from(hunan.shp) to(DMSP-like2020.tif)
Converting coordinates from CRS:
Source CRS: GEOGCS["GCS_WGS_1984",
  DATUM["D_WGS_1984",
    SPHEROID["WGS_1984", 6378137.0, 298.257223563]],
  PRIMEM["Greenwich", 0.0],
  UNIT["degree", 0.017453292519943295],
  AXIS["Longitude", EAST],
  AXIS["Latitude", NORTH]]
Target CRS: PROJCS["PCS Name = WGS_1984_Albers",
  GEOGCS["WGS 84",
    DATUM["World Geodetic System 1984",
      SPHEROID["WGS 84", 6378137.0, 298.257223563, AUTHORITY["EPSG","7030"]],
      AUTHORITY["EPSG","6326"]],
    PRIMEM["Greenwich", 0.0, AUTHORITY["EPSG","8901"]],
    UNIT["degree", 0.017453292519943295],
    AXIS["Geodetic longitude", EAST],
    AXIS["Geodetic latitude", NORTH],
    AUTHORITY["EPSG","4326"]],
  PROJECTION["Albers_Conic_Equal_Area"],
  PARAMETER["central_meridian", 105.0],
  PARAMETER["latitude_of_origin", 0.0],
  PARAMETER["standard_parallel_1", 25.0],
  PARAMETER["false_easting", 0.0],
  PARAMETER["false_northing", 0.0],
  PARAMETER["standard_parallel_2", 47.0],
  UNIT["m", 1.0],
  AXIS["Easting", EAST],
  AXIS["Northing", NORTH]]

.
. qui sum alber__X

. local maxX = r(max)+2000

. local minX = r(min)-2000

.
```

```
. qui sum alber__Y
. local maxY = r(max)+2000
. local minY = r(min)-2000
.
. gtiffread DMSP-like2020.tif, origin(1 1) size(-1 1) clear
. gen n=_n
. sum n if y>`minY´ & y<`maxY´
    Variable │        Obs        Mean    Std. dev.       Min        Max
─────────────┼─────────────────────────────────────────────────────────
           n │        611        3028    176.5248       2723       3333
. local start_row = r(min)
. local n_rows = r(N)
.
. gtiffread DMSP-like2020.tif, origin(1 1) size(1 -1) clear
. gen n=_n
. sum n if x>`minX´ & x<`maxX´
    Variable │        Obs        Mean    Std. dev.       Min        Max
─────────────┼─────────────────────────────────────────────────────────
           n │        550      3290.5    158.9156       3016       3565
. local start_col = r(min)
. local n_cols = r(N)
.
. gtiffread DMSP-like2020.tif, origin(`start_row´ `start_col´) size(`n_rows´ `n_cols´) clear
.
. save DMSP-like2020.dta,replace
file DMSP-like2020.dta saved
```

Figure 1 presents the spatial distribution of the nighttime light brightness across Hunan in 2020. It can be seen that the spatial distribution of nighttime light brightness in Hunan Province is relatively uneven, and there are obvious core areas which may be the main cities in the province, such as Changsha.

```
. use DMSP-like2020.dta, clear
.
. heatplot value y x, color(plasma) ///
>     keylabels(, format(%4.2f))
.
```

Read the NetCDF file

Next, Here's how to read the ncfile.

We know that coordinate system of tas_day_BCC-CSM2-MR_ssp245_r1i1p1f1_gn_2050.nc is WSG84 from Example 7.2, so we can directly slice and read based on the latitude and longitude range of Hunan Province.

Hunan Province lies between 24°38'N and 30°08'N latitude and 108°47'E and 114°15'E longitude. The following example involves loading grid data that encompasses Hunan Province, China, within specific latitude longitude (108°–115°) and (24°–31°) ranges. To serve this purpose, we first read the entire variable *lon* from the nc file into Stata. Then

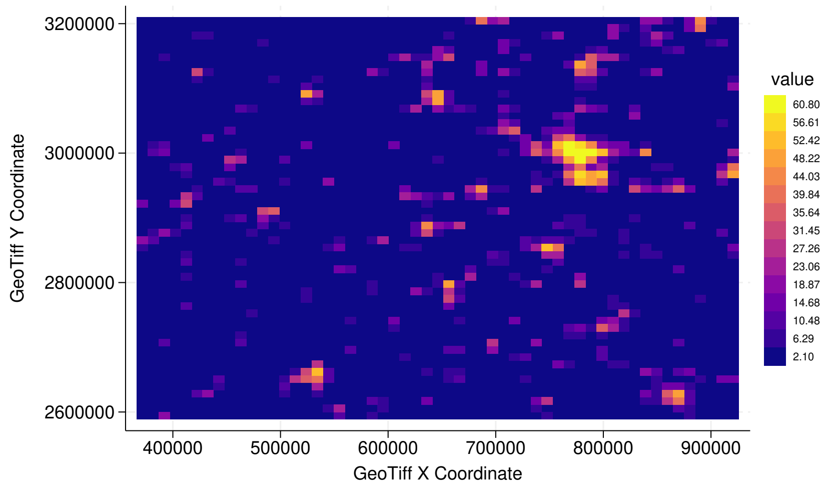

Figure 1: Spatial Distribution of Nighttime Light Brightness

we generate a variable n recording the row number and find the origin index for 108° and the size for the range (108°–115°). Similarly, we read the entire variable *lat* from the nc file into Stata and record the corresponding origin and size.

```
. local url = "https://nex-gddp-cmip6.s3-us-west-2.amazonaws.com/" + ///
> "NEX-GDDP-CMIP6/BCC-CSM2-MR/ssp245/r1i1p1f1/tas/" + ///
> "tas_day_BCC-CSM2-MR_ssp245_r1i1p1f1_gn_2050.nc"

.
. ncread lon using `"`url´"´, clear

Sucessfully import 1440 Obs into Stata.

. gen n=_n

. qui sum n if lon>=108 & lon<=115

. local lon_start = r(min)

. local lon_count = r(N)

.
. ncread lat using `"`url´"´, clear

Sucessfully import 600 Obs into Stata.

. gen n=_n

. qui sum n if lat>=24 & lat<=31

. local lat_start = r(min)

. local lat_count = r(N)

.
. ncread tas using `"`url´"´, clear origin(1 `lat_start´ `lon_start´) ///
>  size(-1 `lat_count´ `lon_count´)

Sucessfully import 286160 Obs into Stata.

.
. gen date = time - 3650.5  + date("2050-01-01", "YMD")
. format date %td
.
. list in 1/10

     +------------------------------------------------------+
     |   time      lat       lon         tas         date   |
     |------------------------------------------------------|
  1. | 3650.5   24.125   108.125   288.02673    01jan2050   |
  2. | 3650.5   24.125   108.375   289.05478    01jan2050   |
  3. | 3650.5   24.125   108.625   288.97476    01jan2050   |
  4. | 3650.5   24.125   108.875   288.53751    01jan2050   |
  5. | 3650.5   24.125   109.125   289.27686    01jan2050   |
     |------------------------------------------------------|
  6. | 3650.5   24.125   109.375   290.04214    01jan2050   |
  7. | 3650.5   24.125   109.625   289.90146    01jan2050   |
  8. | 3650.5   24.125   109.875   290.08304    01jan2050   |
  9. | 3650.5   24.125   110.125   286.86743    01jan2050   |
 10. | 3650.5   24.125   110.375   287.87402    01jan2050   |
     +------------------------------------------------------+

.
. save "grid_all.dta", replace
file grid_all.dta saved
.
```

7.4 Calculate Average and Total Nighttime Light Intensity for Hunan

`gzonalstats` command allows us to aggregate raster data based on administrative regions and compute specified statistics, such as sum and average. This example illustrates how to calculate the average and total nighttime light intensity for Hunan from DSMP-like2020 with `gzonalstats`.

```
. gzonalstats DMSP-like2020.tif using hunan.shp, stats("sum avg") clear
Raster CRS: PCS Name = WGS_1984_Albers
Raster CRS WKT: PROJCS["PCS Name = WGS_1984_Albers",
  GEOGCS["WGS 84",
    DATUM["World Geodetic System 1984",
      SPHEROID["WGS 84", 6378137.0, 298.257223563, AUTHORITY["EPSG","7030"]],
      AUTHORITY["EPSG","6326"]],
    PRIMEM["Greenwich", 0.0, AUTHORITY["EPSG","8901"]],
    UNIT["degree", 0.017453292519943295],
    AXIS["Geodetic longitude", EAST],
    AXIS["Geodetic latitude", NORTH],
    AUTHORITY["EPSG","4326"]],
  PROJECTION["Albers_Conic_Equal_Area"],
  PARAMETER["central_meridian", 105.0],
  PARAMETER["latitude_of_origin", 0.0],
  PARAMETER["standard_parallel_1", 25.0],
  PARAMETER["false_easting", 0.0],
  PARAMETER["false_northing", 0.0],
  PARAMETER["standard_parallel_2", 47.0],
  UNIT["m", 1.0],
  AXIS["Easting", EAST],
  AXIS["Northing", NORTH]]
Shapefile CRS: GCS_WGS_1984
Vector CRS WKT: GEOGCS["GCS_WGS_1984",
  DATUM["D_WGS_1984",
    SPHEROID["WGS_1984", 6378137.0, 298.257223563]],
  PRIMEM["Greenwich", 0.0],
  UNIT["degree", 0.017453292519943295],
  AXIS["Longitude", EAST],
  AXIS["Latitude", NORTH]]
Reprojecting shapefile from GCS_WGS_1984 to PCS Name = WGS_1984_Albers
Shapefile bounds for raster reading: ReferencedEnvelope[373271.4002242747 : 918728.259
> 838356, 2596264.786161866 : 3202467.2929103803] DefaultProjectedCRS[PCS Name = WGS_1
> 984_Albers] AXIS["Easting", EAST] AXIS["Northing", NORTH]
Optimizing raster read to only cover shapefile extent
Raster envelope: ReferencedEnvelope[-2643772.8565207715 : 2212227.1434792285, 1871896.
> 5262559392 : 5926896.526255939] DefaultProjectedCRS[PCS Name = WGS_1984_Albers] AXIS
> ["Easting", EAST] AXIS["Northing", NORTH]
Using intersection bounds: ReferencedEnvelope[373271.4002242747 : 918728.259838356, 25
> 96264.786161866 : 3202467.2929103803] DefaultProjectedCRS[PCS Name = WGS_1984_Albers
> ] AXIS["Easting", EAST] AXIS["Northing", NORTH]
Successfully created optimized read parameters
Successfully read raster data with optimization
Total features: 14
Created string variable: z_ShiName (length 16)
Created numeric variable: z_Shape_Leng
Created numeric variable: z_Shape_Area
Created string variable: z_Name (length 16)
Created numeric variable: z_lon
Created numeric variable: z_lat
Created numeric variable: avg
Created numeric variable: sum
Data successfully exported to Stata dataset.

. list z_Name avg sum

     +-----------------------------------+
     |      z_Name         avg       sum |
     |-----------------------------------|
  1. |     Changde   3.2560185     59240 |
  2. |    Chenzhou   2.0607766     39909 |
  3. |    Hengyang   3.2285136     49435 |
  4. |     Huaihua   1.2616182     34803 |
  5. |       Loudi   3.7707743     30630 |
     |-----------------------------------|
  6. |    Shaoyang   1.7991827     37423 |
  7. |    Xiangtan   7.2288051     36238 |
  8. |    Xiangxi     1.458301     22557 |
  9. |      Yiyang   2.4555944     30221 |
 10. |   Yongzhou    2.0996052     46796 |
     |-----------------------------------|
 11. |     Yueyang   3.8674934     57382 |
 12. | Zhangjiajie   1.7823899     17004 |
 13. |    Changsha   11.616603    137134 |
 14. |     Zhuzhou   3.8699787     43545 |
     +-----------------------------------+

. save "hunan_light.dta", replace
file hunan_light.dta saved
```

The output indicates that the GeoTiff and Shape files have different projection coordinate systems. The command first converts the coordinate system of the Shape file to match the raster data. Then, the raster data covering the Shape file extent are read. Finally, the sum and average statistics are computed for each city in the Shape file. Figure 2 presents the total night light index (TNLI) and average night light index (ANLI) in Hunan. It presents a high value agglomeration of lights in ChangZhuTan area as the core, indicating that the region is the economic, cultural and demographic center of Hunan Province, with strong links between cities, forming a more mature pattern of urban agglomeration development.

```
. use "hunan_light.dta" ,clear
. rename z_Name city
. merge 1:1 city using hunan_city.dta,nogen
    Result                      Number of obs
    ─────────────────────────────────────────
    Not matched                             0
    Matched                                14
    ─────────────────────────────────────────

.
. shp2dta using "hunan.shp", database(hunan_db) coordinates(hunan_coord)
>  genid(id)
type: 5

.
. spmap sum using "hunan_coord.dta", id(OBJECTID) clmethod(q) cln(6) fco
> lor(Heat) title("Total Night Light Index")
. graph save graph1, replace
file graph1.gph saved

.
. spmap avg using "hunan_coord.dta", id(OBJECTID) clmethod(q) cln(6) fco
> lor(Heat) title("Average Night Light Index")
. graph save graph2, replace
file graph2.gph saved

.
. graph combine graph1.gph graph2.gph
```

7.5 Match cities to nearest four grid cells using matchgeop

The `matchgeop` command is a powerful tool for geospatial analysis, enabling the matching of point data (e.g., cities) with gridded data (e.g., climate variables). In this example, we demonstrate how to use `matchgeop` to match each city in Hunan Province with the nearest four grid points from the NetCDF file
tas_day_BCC-CSM2-MR_ssp245_r1i1p1f1_gn_2050.nc.

This example utilizes the `grid_all.dta` file from Example 7.3, containing temperature data for Hunan Province.First we take a single day's data as a sample and use the matchgeop command to find the nearest four grid points (with coordinates ulat and ulon) for each city in Hunan Province. Then link the grid points' coordinates to the complete temporal dataset to obtain the full time series of temperature data (*tas*) for these points.

```
. use "grid_all.dta", clear
.
. rename lon ulon
. rename lat ulat
```

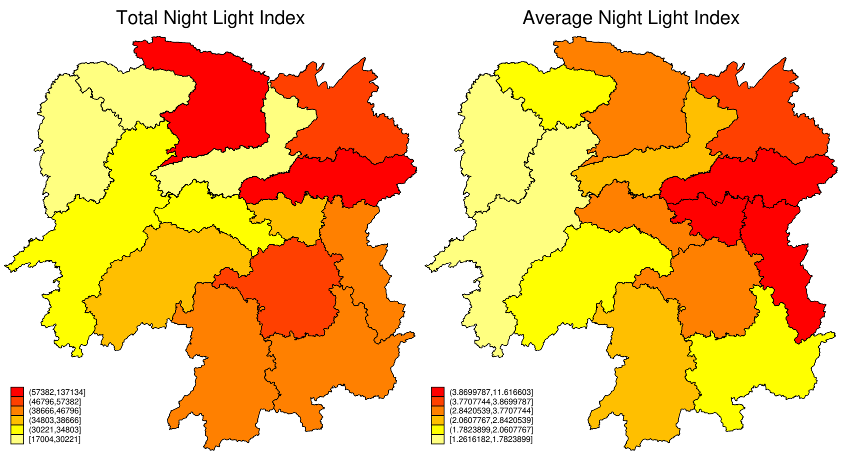

Figure 2: Night Light Index in Hunan

```
. save "grid_all_1.dta", replace
file grid_all_1.dta saved

.
. keep if time==3650.5
(285,376 observations deleted)

. gen n=_n

. save "hunan_grid.dta", replace
file hunan_grid.dta saved

.
. use "hunan_city.dta", clear

. matchgeop ORIG_FID lat lon using hunan_grid.dta, neighbors(n ulat ulon)
> nearcount(4) gen(distance) bearing(angle)

.
. merge m:1 n using hunan_grid.dta, keep(3)

    Result                      Number of obs
    -----------------------------------------

    Not matched                             0
    Matched                                56  (_merge==3)
    -----------------------------------------

. drop _merge

. save "hunan_origin.dta", replace
file hunan_origin.dta saved

.
. drop time date

. joinby ulat ulon using grid_all_1.dta

.
. sort ORIG_FID date

. list ORIG_FID distance angle date tas in 1/10
```

```
     +-------------------------------------------------------------+
     |    city    distance       angle        date           tas   |
     |-------------------------------------------------------------|
  1. | Changde   16.299124   300.75839   01jan2050   280.97092     |
  2. | Changde   21.997551   152.24742   01jan2050   281.17828     |
  3. | Changde   23.998957   215.80666   01jan2050   281.43893     |
  4. | Changde   13.184405   50.807281   01jan2050    280.9436     |
  5. | Changde   13.184405   50.807281   02jan2050    280.9436     |
     |-------------------------------------------------------------|
  6. | Changde   16.299124   300.75839   02jan2050   280.97092     |
  7. | Changde   21.997551   152.24742   02jan2050   281.17828     |
  8. | Changde   23.998957   215.80666   02jan2050   281.43893     |
  9. | Changde   13.184405   50.807281   03jan2050    280.9436     |
 10. | Changde   23.998957   215.80666   03jan2050   281.43893     |
     +-------------------------------------------------------------+
```

We also calculate the bearing (azimuth) between cities and their matched grid points. A bearing diagram using the dataset above hunan_origin.dta is provided to illustrate the relative positions of the matched points.The results are shown in Figure 3.

```
. use "hunan_origin.dta", clear
. keep if city == "Changsha"
(52 observations deleted)
.
. sort angle
. gen id=_n
.
. local R = 6371
. gen lat_rad = lat * (_pi/180)
.
. gen delta_lat = (distance / `R´) * (180/_pi)
. gen delta_lon = (distance / (`R´ * cos(lat_rad))) * (180/_pi)
.
. expand 90
(356 observations created)
. bysort n: gen t = _n - 1
. gen theta = (angle * t/89) * (_pi/180)
.
. gen arc_lat = lat + delta_lat * cos(theta)
. gen arc_lon = lon + delta_lon * sin(theta)
.
. bysort n: gen label_theta = (angle/2) * (_pi/180)
. gen label_lat = lat + delta_lat * cos(label_theta)
. gen label_lon = lon + delta_lon * sin(label_theta)
. replace label_lat = lat + delta_lat * 0.7 * cos(label_theta) if id == 3
(90 real changes made)
. replace label_lon = lon + delta_lon * 0.7 * sin(label_theta) if id == 3
(90 real changes made)
.
. gen latlon_label = "(" + string(lat, "%8.2f") + "°N, " + string(lon, "%8.2
> f") + "°E)"
. gen ulatlon_label = "(" + string(ulat, "%8.2f") + "°N, " + string(ulon, "%
> 8.2f") + "°E)"
. gen angle_label = string(angle, "%8.2f") + "{&degree}"
.
. twoway pcarrowi 28.15 113.15307 28.52 113.15307, color(black) ///
>     || pcarrow lat lon ulat ulon, color(blue) ///
>     || scatter lat lon if t == 1, mcolor(red) mlabel(latlon_label) mlabcol
> or(black) mlabpos(9) mlabgap(0.8) mlabsize(medium) ///
>     || scatter ulat ulon if t == 1 & (ulon == 113.125), mcolor(red) mlabel
> (ulatlon_label) mlabcolor(black) mlabpos(9) mlabgap(0.8) mlabsize(medium)
> ///
>     || scatter ulat ulon if t == 1 & (ulon == 113.375), mcolor(red) mlabel
> (ulatlon_label) mlabcolor(black) mlabpos(3) mlabgap(0.8) mlabsize(medium)
> ///
>     || line arc_lat arc_lon if id == 1 , lcolor(green) ///
>     || line arc_lat arc_lon if id == 2, lcolor(green) ///
>     || line arc_lat arc_lon if id == 3, lcolor(green) ///
>     || line arc_lat arc_lon if id == 4, lcolor(green) ///
>     || scatter label_lat label_lon, mlabel(angle_label) msymbol(i) mlabcol
> or(black) mlabsize(medium) ///
>     xscale(off noline) yscale(off noline) xlabel(, nogrid noticks) ylabel(
> , nogrid noticks) ///
>     aspect(1) legend(off)
.
```

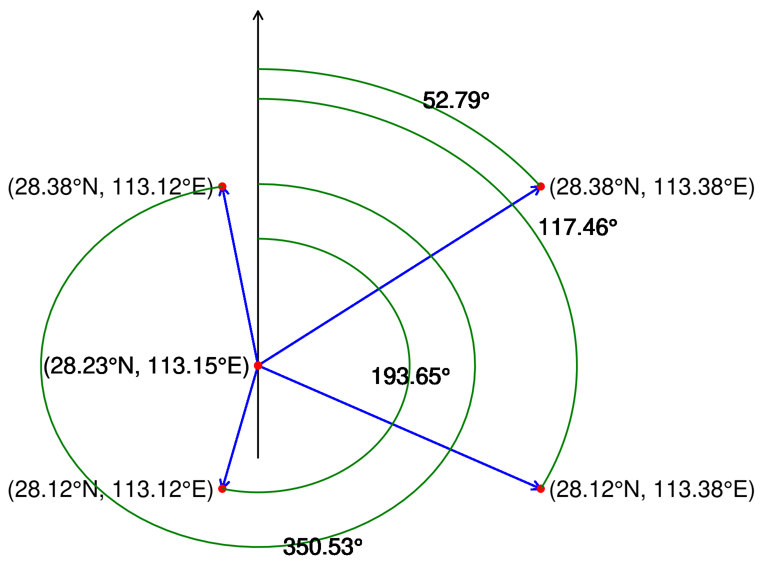

Figure 3: Diagram of azimuthal angle

7.6 Calculate 80km-radius IDW temperatures for cities

This example highlights using `matchgeop` to calculate IDW averages of temperature within a specified radius. We use the dataset `grid_all.dta` from example 7.3, which includes comprehensive temperature data for Hunan Province, to perform calculations for grid points within an 80 km radius of each city and the IDW temperatures.

```
. use "grid_all.dta", clear
. rename lon ulon
. rename lat ulat
. gen n=_n
. save "grid_all_2.dta", replace
file grid_all_2.dta saved
.
. use "hunan_city.dta", clear
. matchgeop ORIG_FID lat lon using grid_all_2.dta, neighbors(n ulat ulon) w
> ithin(80) gen(distance)
.
. merge m:1 n using grid_all_2.dta, keep(3)
    Result                      Number of obs
    ─────────────────────────────────────────
    Not matched                             0
    Matched                           148,555  (_merge==3)
    ─────────────────────────────────────────

. drop _merge
.
. list city ulat ulon distance date tas in 1/10

     ┌───────────────────────────────────────────────────────────────────┐
     │     city     ulat      ulon    distance        date          tas │
     ├───────────────────────────────────────────────────────────────────┤
  1. │  Zhuzhou   27.375   114.125   66.515572   29dec2050   267.03802 │
  2. │  Zhuzhou   27.375   113.875    45.40358   29dec2050    267.5249 │
  3. │  Zhuzhou   27.125   113.625   10.759812   29dec2050   267.89655 │
  4. │  Zhuzhou   26.875   113.875   44.757698   29dec2050   268.23151 │
  5. │  Zhuzhou   27.125   113.875    35.49094   29dec2050   268.30505 │
     ├───────────────────────────────────────────────────────────────────┤
  6. │  Yueyang   29.125   113.875   60.655354   29dec2050   268.51117 │
  7. │ Changsha   28.375   113.625   49.050537   29dec2050   268.53442 │
  8. │  Yueyang   28.625   113.875   77.779968   29dec2050   268.57874 │
  9. │ Changsha   27.875   113.625   60.629818   29dec2050   268.60806 │
 10. │  Yueyang   28.875   113.875   63.966019   29dec2050   268.64868 │
     └───────────────────────────────────────────────────────────────────┘

.
. save "hunan_80km.dta", replace
file hunan_80km.dta saved
.
. drop if tas==.
(0 observations deleted)
. gen weight  = 1/distance
. gen weighted_tas = tas * weight
. bysort city date : egen sum_weighted_tas = total(weighted_tas)
. bysort city date : egen total_weight = total(weight)
. gen idw_tas = sum_weighted_tas / total_weight
. gen temp_c = idw_tas - 273.15
.
. duplicates drop city date , force
Duplicates in terms of city date
(143,445 observations deleted)
.
. list city  distance date  temp_c in 1/10

     +----------------------------------------------+
     |    city    distance        date      temp_c |
     |----------------------------------------------|
  1. | Changde   35.447414   01jan2050     7.65365 |
  2. | Changde    72.10421   02jan2050    7.220788 |
  3. | Changde   65.515297   03jan2050     8.44021 |
  4. | Changde    72.10421   04jan2050    6.136468 |
  5. | Changde   59.276711   05jan2050    7.520319 |
     |----------------------------------------------|
  6. | Changde   58.544605   06jan2050    7.391534 |
  7. | Changde   68.854805   07jan2050    6.704889 |
  8. | Changde   65.515297   08jan2050    3.407739 |
  9. | Changde    52.57518   09jan2050    2.457117 |
 10. | Changde   39.613548   10jan2050    2.555414 |
     +----------------------------------------------+

.
. save "hunan_IDW.dta", replace
file hunan_IDW.dta saved
.
```

Figure 4 demonstrates the spatial extent of the 80 km radius around Changsha city.

```
.
. use "hunan_80km.dta" ,clear
. keep if time == 3650.5
(148,148 observations deleted)
. keep if city == "Changsha"
(377 observations deleted)
.
. summarize lon
    Variable |        Obs        Mean    Std. dev.       Min        Max
-------------+---------------------------------------------------------
         lon |         30    113.1531           0   113.1531   113.1531
. local clon = r(mean)
. summarize lat
    Variable |        Obs        Mean    Std. dev.       Min        Max
-------------+---------------------------------------------------------
         lat |         30    28.22691           0   28.22691   28.22691
. local clat = r(mean)
.
. preserve
. clear
. set obs 720
Number of observations (_N) was 0, now 720.
. gen theta = (_n-1) * 2 * _pi/720
```

```
.
. local R = 6371
. local d = 80
. local delta = `d´/`R´
.
. gen lat_c = asin(sin(`clat´*_pi/180)*cos(`delta´) + ///
>              cos(`clat´*_pi/180)*sin(`delta´)*cos(theta)) * (180/_pi)
.
. gen lon_c = `clon´ + ///
>             atan2(sin(theta)*sin(`delta´)*cos(`clat´*_pi/180), ///
>                   cos(`delta´) - sin(`clat´*_pi/180)*sin(lat_c*_pi/180))
> * (180/_pi)
.
. expand 2 if _n == _N
(1 observation created)
. replace theta = 0 if _n == _N
(1 real change made)
.
. save circle.dta ,replace
file circle.dta saved
. restore
.
. merge 1:1 _n using circle.dta, nogen
    Result                      Number of obs
    ─────────────────────────────────────────
    Not matched                           691
        from master                         0
        from using                        691
    Matched                                30
    ─────────────────────────────────────────
.
. twoway (line lat_c lon_c, sort lcolor(dimgray) lwidth(0.6) lpattern(solid
> )) ///
>     (pcarrowi  28.22691 113.1531 28.22691 113.9696, color(black)) ///
>         (scatter lat lon, mcolor(red) msymbol(D) msize(small)) ///
>     (scatter ulat ulon, mcolor(blue) msymbol(O) msize(small)), ///
>     xlabel(minmax) ylabel(minmax) ///
>         aspect(1)  legend(off) ///
>     xscale(off noline) yscale(off noline) xlabel(, nogrid noticks) ylabel
> (, nogrid noticks) ///
>         text(28.26 113.71 "80km", size(medsmall) justification(center))
.
```

Figure 5 presents the resulting IDW interpolated temperature distributions on January 1, 2050, and July 1, 2050.

```
. //January 1
. shp2dta using "hunan.shp", database(hunan_db) coordinates(hunan_coord) genid(id)
type: 5
.
. use "hunan_IDW.dta" ,clear
. keep if date == date("01jan2050", "DMY")
(5,096 observations deleted)
.
```

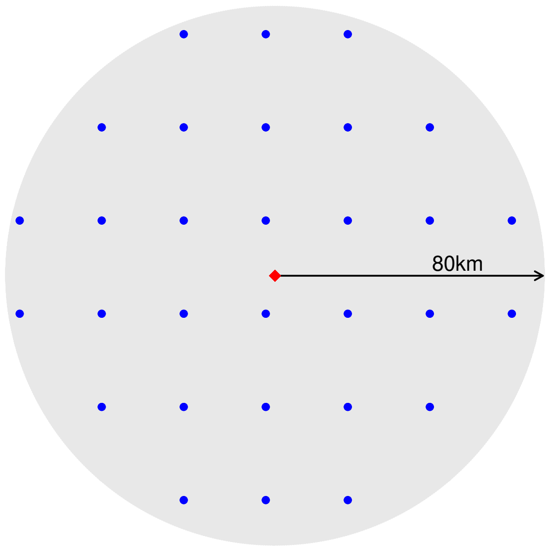

Figure 4: Distribution of raster points within 80 kilometers of Changsha City

```
. spmap temp_c using "hunan_coord.dta", id(OBJECTID) clmethod(q) cln(6) fco
> lor(Heat) legtitle("Temperature (°C)") title("Temperature(20500101)") sub
> title("Within 80km Radius")

. graph save graph1, replace
file graph1.gph saved

.
. //July 1
. use "hunan_IDW.dta" ,clear

. keep if date == date("01jul2050", "DMY")
(5,096 observations deleted)

.
. spmap temp_c using "hunan_coord.dta", id(OBJECTID) clmethod(q) cln(6) fco
> lor(Heat) legtitle("Temperature (°C)") title("Temperature(20500701)") sub
> title("Within 80km Radius")

. graph save graph2, replace
file graph2.gph saved

.
. graph combine graph1.gph graph2.gph
```

8 Conclusions

Geographical remote sensing data, with their unique advantages, play an irreplaceable role in social and economic research. To enhance Stata in handling geographical data, we have developed a set of Stata commands. These commands enable users to read

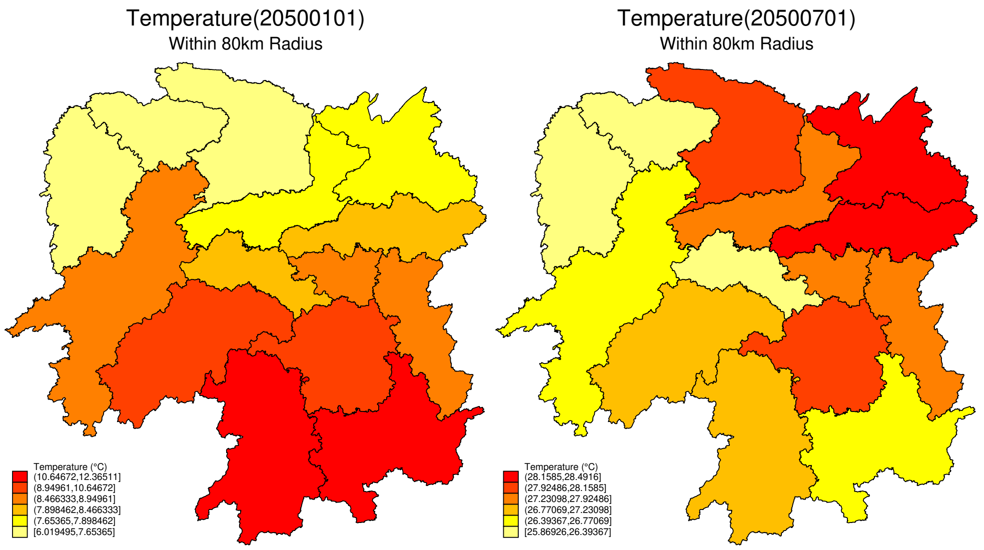

Figure 5: IDW interpolated temperature distributions in Hunan

and process data from Geotiff and NetCDF files. Although the paper highlights the strengths of our work, there are some limitations. First, the implemented commands require downloading the entire NetCDF and GeoTools libraries, which occupies more than 100 MB of disk space. Second, many advanced functionalities of GeoTools, such as creating buffers, cropping rasters, and resampling, remain unutilized. Third, compared to ArcGIS, data operations lack visualization and are less intuitive.

9 Acknowledgments

Kerui Du thanks the financial support from the National Natural Science Foundation of China (Grant nos. 72473119 and 72074184).

10 References

Barwick, P. J., S. Li, L. Lin, and E. Y. Zou. 2024. From fog to smog: The value of pollution information. *American Economic Review* 114(5): 1338–1381.

Burke, M., S. M. Hsiang, and E. Miguel. 2015. Global non-linear effect of temperature on economic production. *Nature* 527(7577): 235–239.

He, G., T. Liu, and M. Zhou. 2020. Straw burning, PM2. 5, and death: Evidence from China. *Journal of Development Economics* 145: 102468.

Henderson, J. V., A. Storeygard, and D. N. Weil. 2012. Measuring economic growth from outer space. *American economic review* 102(2): 994–1028.

About the authors

Kerui Du is an associate professor at the School of Management, Xiamen University, China. His primary research interests include applied econometrics, energy, and environmental economics.

Chunxia Chen is a master's student in Technological Economics and Management at the School of Management, Xiamen University, China.

Shuo Hu is a Ph.D. student at the School of Economics, Southwestern University of Finance and Economics. His research interest is international environmental economics.

Yang Song is master student in Applied Economics at the School of Economics, Hefei University of Technology, China

Ruipeng Tang (corresponding author) is a professor at the School of Economics, Hefei University of Technology, China. His primary research interests include energy and environmental economics, and innovation economics.