The Stata Journal (*yyyy*) *vv*, Number *ii*, pp. 1–29

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
Hefei, China
ss0706082021@163.com

Ruipeng Tan
School of Economics
Hefei University of Technology
Hefei, China
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. 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). Crucially, since our Stata commands rely on this integration to execute external Java libraries for raster processing, users must ensure they have Java Development Kit (JDK) version 17 or higher installed and properly configured within their Stata environment.

There are seven new Stata commands are introduced: `gtiffdisp`, `gtiffread`, `ncdisp`, `ncread`, `crsconvert`, `zonalstats` and `matchgeop`. The detailed functional description of these commands and their relationships are presented in Section 3. 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 to 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.

3 The workflow of the package

Seven new Stata commands establish a comprehensive, three-stage workflow for processing GeoTIFF and NetCDF raster data entirely within the statistical environment. As illustrated in Figure 1, the package allows Stata users to transition seamlessly from raw geospatial files to integrated, analysis-ready datasets, specifically designed to quantify environmental or geographic exposure for Areas of Interest (AOIs). This quantification relies on overlaying a zone layer (e.g., administrative boundaries or grid cells) onto a value layer (e.g., nighttime light intensity, temperature, or pollution data). The core operation involves extracting cell values falling within each zone and computing the required statistics (e.g., mean, sum) to quantify exposure per AOI.

The workflow begins with metadata inspection. `gtiffdisp` and `ncdisp` are used to display essential metadata from GeoTIFF and NetCDF files, respectively. This is followed by the Data Import stage, where `gtiffread` and `ncread` read the raster data and vectorize it into Stata's memory. The final Spatial Operation stage facilitates advanced geospatial analysis and manipulation of the imported data. `zonalstats` computes summary statistics by aggregating value raster cells intersecting polygonal zones. This operates directly on the input files, bypassing the need for prior vectorization, and includes internal logic to manage and correct spatial misalignment between the zone layer and the value raster. For estimating localized exposure at discrete point locations (e.g., monitoring stations) instead of polygon aggregation, the `matchgeop` command is used. Data imported from either format can first be processed using `crsconvert` to ensures data consistency by harmonizing Coordinate Reference Systems (CRS). And then it is followed by `matchgeop`, which facilitates geographic matching, such as finding nearest neighbors between point data and raster cells. These steps ultimately lead to advanced point exposure estimation, such as calculating an Inverse Distance Weighted (IDW) average of nearby raster cells to reflect a localized exposure measure.

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.

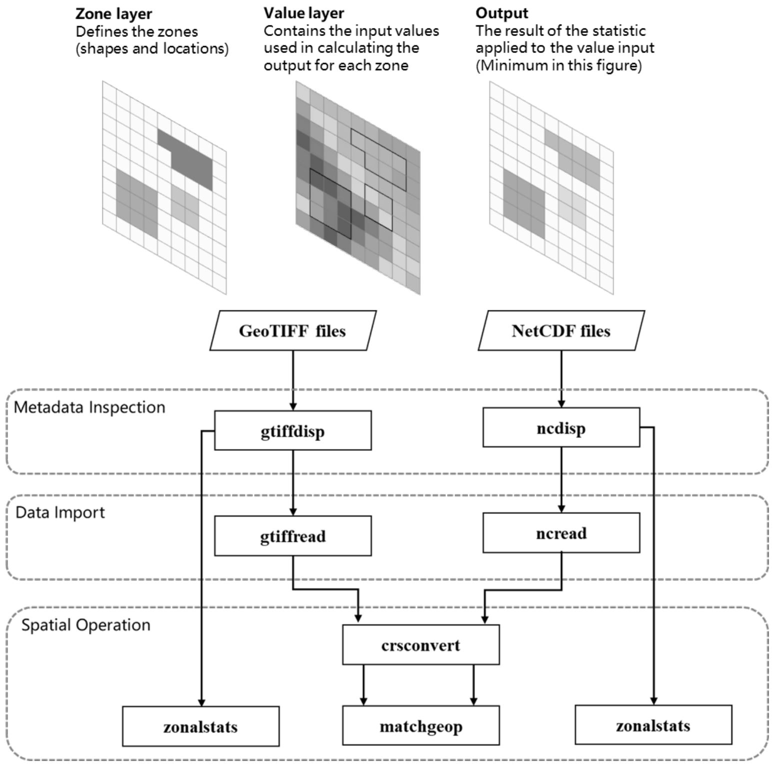
Zone layer
Defines the zones (shapes and locations)
Value layer
Contains the input values used in calculating the output for each zone
Output
The result of the statistic applied to the value input (Minimum in this figure)
GeoTIFF files
NetCDF files
Metadata Inspection
gtiffdisp
ncdisp
Data Import
gtiffread
ncread
Spatial Operation
crsconvert
zonalstats
matchgeop
zonalstats

Figure 1: The workflow for the developed command set

`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 variable in the NetCDF file.

path_to_ncfile specifies the NetCDF 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 comprehensive metadata for NetCDF files. Without a variable name, it shows global attributes (e.g., data source, coordinate system). With a variable name, it displays variable-specific details including dimensions, coordinates, and data type.

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)*). 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 NetCDF file. If not specified, the command would display the metadata in the nc file.

path_to_ncfile specifies the NetCDF file to be read.

Options

`clear` specifies clearing the current dataset.

`csv(`*filename*`[,`*replace*`])` specifies reading the data from the NetCDF 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 NetCDF 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 zonalstats command

Syntax

For GeoTiff files:

`zonalstats` *rasterfilename* `using` *shpfile*`,` [`stats(`*string*`)` `band(`*#*`)` `crs(`*string*`)` `clear`]

For NetCDF files:

`zonalstats` *rasterfilename* `using` *shpfile* `var` *string*`,` [`stats(`*string*`)` `crs(`*string*`)` `origin(`*numlist*`)` `size(`*numlist*`)` `clear`]

rasterfilename can be a GeoTIFF file (*.tif* or *.tiff*) or NetCDF file (*.nc*). The command automatically detects the file type and uses the appropriate processing method.

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.

`stats(`*string*`)` specifies which statistics to calculate. Default is *avg* 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

`crs(`*string*`)` specifies coordinate reference system for the raster data(e.g., "EPSG:4326"). If the raster file contains CRS information, this option is ignored.

`band(`#`)` specifies which raster band to use in multi-band raster files. Default is 1 (first band). This option is only applicable when performing zonal statistics on GeoTIFF files.

`var(`*string*`)` specifies variable name in the NetCDF file to process. It is required for NetCDF files

`origin(`*numlist*`)` specifies the starting indices of each dimension for rslicing the NetCDF variable. This option is only applicable when performing zonal statistics on NetCDF files.

`size(`*numlist*`)` specifies the number of elements to read along each dimension when slicing the NetCDF variable. Only up to two dimensions can have a size greater than 1. The requirement that the data must be a 2D grid is enforced because zonal statistics operations are defined for spatial data distributed over two dimensions—typically latitude and longitude (or x and y). This option is only applicable when performing zonal statistics on NetCDF files.

Description

The `zonalstats` command calculates zonal statistics for each polygon in a shapefile based on the values of raster cell values 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.

6.2 The crsconvert command

Syntax

`crsconvert` *varlist* [, `gen(`*string*`)` *from(string) to(string)*] *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 can specify a GeoTIFF (*.tif/.tiff*), Shapefile (*.shp*), or NetCDF (*.nc*) file to automatically extract the coordinate reference system from the file. This is particularly useful when the coordinate system does not correspond to a standard EPSG code but includes custom projection parameters. If a file is provided, the command will automatically extract the CRS information from it.

`to(`*string* `[,` *tif shp*`])` specifies the source coordinate reference system. It might be provided in EPSG format(e.g. EPSG:4326). Alternatively, users can specify a GeoTIFF (*.tif/.tiff*), Shapefile (*.shp*), or NetCDF (*.nc*) file 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.

6.3 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

To utilize the GeoTools-based commands (`gtiffdisp`, `gtiffread`, `crsconvert`, and `zonalstats`) as well as the NetCDF-related commands (`ncdisp`, `ncread`, `crsconvert`, and `zonalstats`) in Stata, users must first configure the necessary Java dependencies. Note that certain commands, such as `crsconvert` and `zonalstats`, rely on both GeoTools and NetCDF environments. The initialization is required only once unless the library files are moved or deleted.

For GeoTools setup, begin by downloading the GeoTools Version 32.0 Java library. After downloading, place it within Stata's adopath or add its file path manually. For a simplified setup, we provide a dedicated command: `geotools_init`. To configure the environment automatically, simply run the following line in Stata:`geotools_init, download plus(geotools)` . Note that this process may take dozens of minutes—Stata's speed for copying large files from the internet is relatively slow. As a faster alternative, we recommend manually downloading the GeoTools 32.0 package from Sourceforge [3] and unzipping the downloaded file. After doing so, initialize the environment by running: `geotools_init path_to_geotools-32.0/lib, plus(geotools)`. Note that you should replace `path_to_geotools-32.0/lib` with the actual file path to your unzipped GeoTools 32.0 lib folder.

For NetCDF setup, user can use the `netcdf_init` command. Running `netcdf_init, download plus(netcdf)` automatically downloads the necessary netcdfAll-5.9.1.jar file and copies it to the jar subdirectory within Stata's sysdir PLUS.

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

7.1 GeoTIFF Data Processing

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). (Wu 2021)

```
. 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.

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. As a major province in China, it exhibits diverse socioeconomic and environmental characteristics. And Hunan's moderate size offers a manageable spatial scale that balances data complexity and operational clarity, ensuring the example remains both illustrative and computationally efficient.

The first step is to determine the latitude and longitude range of Hunan Province. As shown in Example 5.1, the coordinate system of DMSP-like2020.tif 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.

```
. spshape2dta hunan.shp, replace
  (importing .shp file)
  (importing .dbf file)
  (creating _ID spatial-unit id)
  (creating _CX coordinate)
  (creating _CY coordinate)

  file hunan_shp.dta created
  file hunan.dta     created

. use "hunan.dta",clear

. crsconvert _CX _CY, 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_CX
. local maxX = r(max)+2000
. local minX = r(min)-2000
.
. qui sum alber_CY
. 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 2 presents the spatial distribution of the nighttime light brightness across

Hunan in 2020. The image reflects a bounding rectangle coverage rather than a precisely clipped boundary, resulting from the pixel-level slicing mechanism of the `gtiffread` command. This approach preserves original data structure and avoids interpolation, maintaining analytical integrity.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(Greys, reverse) level(6) xlabel(, angle(45))
```

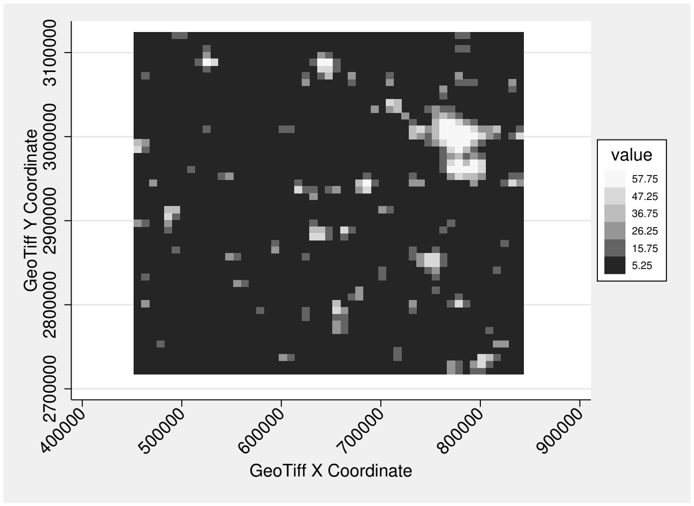

Figure 2: Spatial Distribution of Nighttime Light Brightness

Calculate average nighttime light intensity for Hunan

This part illustrates how to calculate the average nighttime light intensity for Hunan from DSMP-like2020 with `zonalstats`.

```
. zonalstats DMSP-like2020.tif using hunan.shp, stats("avg") clear
GeoTIFF CRS detected: PCS Name = WGS_1984_Albers. User-provided CRS is ignored.
Shapefile CRS: GCS_WGS_1984
Reprojecting shapefile from GCS_WGS_1984 to PCS Name = WGS_1984_Albers
Shapefile bounds for raster reading: ReferencedEnvelope[373271.4002242747 :
>  918728.259838356, 2596264.786161866 : 3202467.2929103803] DefaultProjected
>  CRS[PCS Name = WGS_1984_Albers] AXIS["Easting", EAST] AXIS["Northing", NORTH]
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 : 918728.2598383
> 56, 2596264.786161866 : 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
Data successfully exported to Stata dataset.
. list z_Name avg

     +--------------------------+
     |      z_Name          avg |
     |--------------------------|
  1. |     Changde    3.2560185 |
  2. |    Chenzhou    2.0607766 |
  3. |    Hengyang    3.2285136 |
  4. |     Huaihua    1.2616182 |
  5. |       Loudi    3.7707743 |
     |--------------------------|
  6. |    Shaoyang    1.7991827 |
  7. |    Xiangtan    7.2288051 |
  8. |     Xiangxi     1.458301 |
  9. |      Yiyang    2.4555944 |
 10. |   Yongzhou    2.0996052 |
     |--------------------------|
 11. |     Yueyang    3.8674934 |
 12. | Zhangjiajie    1.7823899 |
 13. |    Changsha    11.616603 |
 14. |     Zhuzhou    3.8699787 |
     +--------------------------+

. 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 average statistics are computed for each city in the Shape file. Figure 3 presents the 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.dta, clear

.
. rename Name z_Name

. merge 1:1 z_Name using hunan_light.dta,nogen
(variable z_Name was str11, now str16 to accommodate using
      data´s values)

    Result                      Number of obs
    -----------------------------------------
    Not matched                             0
    Matched                                14
    -----------------------------------------

. save hunan_light.dta, replace
```

```
file hunan_light.dta saved
.
. geoframe create region ///
>  "hunan_light.dta", id(_ID) centroids(_CX _CY) ///
>  shp(hunan_shp.dta) ///
>  replace
(creating frame region from hunan_light.dta)
(creating frame region_shp from hunan_shp.dta)
           Frame name: region [make current]
           Frame type: attribute
         Feature type: <none>
        Number of obs: 14
              Unit ID: _ID
          Coordinates: _CX _CY
   Linked shape frame: region_shp
.
. geoplot ///
>  (area region avg, color(Greys, reverse) ///
>         level(6, quantile weight(avg))) ///
>  (line region, lwidth(vthin)), ///
>  legend(position(sw))
```

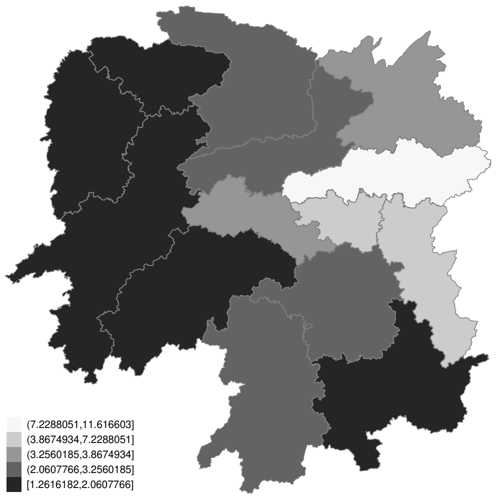

Figure 3: Average nighttime light intensity in Hunan

Calculate 80km-radius IDW nighttime light for cities

While `zonalstats` performed area-based aggregation using SHP files, this subsequent step employs the `matchgeop` command to calculate Inverse Distance Weighting (IDW), demonstrating estimation from discrete points. It provides a perspective focused on the urban core instead of aggregate administrative regions. The calculation utilizes the nighttime light data contained in the previously created `DMSP-like2020.dta` file, with the geographic information for the central points of all cities in Hunan Province sourced from hunan_city.dta.

```
. use "DMSP-like2020.dta", clear

. crsconvert x y, gen(wsg84_) from(DMSP-like2020.tif) to(hunan.shp)
Converting coordinates from CRS:
Source 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]]
Target 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]]

. gen n=_n

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

.
. use "hunan_city.dta", clear

. matchgeop ORIG_FID lat lon using light_china.dta, neighbors(n wsg84_y wsg84_x)
>  within(80) gen(distance)

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

    Result                      Number of obs
    ─────────────────────────────────────────
    Not matched                             0
    Matched                           206,615  (_merge==3)
    ─────────────────────────────────────────

. drop _merge

.
. drop if value==.
(0 observations deleted)

. gen weight  = 1/distance

. gen weighted_light = value * weight

. bysort city: egen sum_weighted_light = total(weighted_light)

. bysort city: egen total_weight = total(weight)

. gen idw_light = sum_weighted_light / total_weight
```

```
.
. duplicates drop city, force
Duplicates in terms of city
(206,601 observations deleted)
```

Figure 4 presents the resulting IDW nighttime light distributions in 2020. Changsha remains the area with the highest light intensity. However, the IDW approach (Figure 4) captures peak values nearly double those of the Zonal Average (Figure 3). This disparity is expected, as the Zonal Average calculates intensity across the entire administrative area, resulting in a dilution effect by including surrounding rural areas.

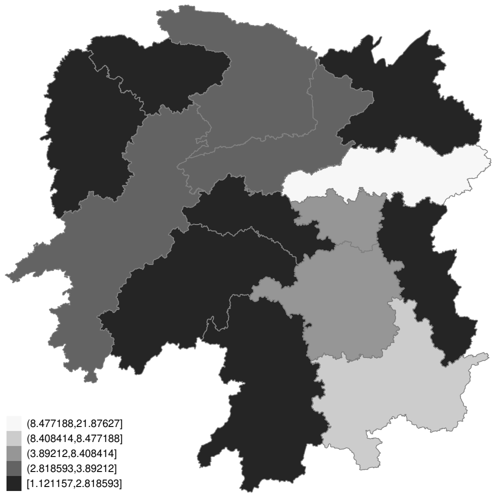

Figure 4: IDW nighttime light intensity at the geocentric center of cities in Hunan

7.2 NetCDF Data Processing

Display the Metadata of the NetCDF File

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

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).[4] 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).

4. NASA Earth Exchange Global Daily Downscaled Projections (NEX-GDDP-CMIP6) was accessed on October 5, 2025 from https://registry.opendata.aws/nex-gddp-cmip6. NEX-GDDP-CMIP6 data was accessed on October 5, 2025 from https://registry.opendata.aws/nex-gddp-cmip6.

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
```

Read the NetCDF file

Next, here is how to read the NetCDF file.

The coordinate system of tas_day_BCC-CSM2-MR_ssp245_r1i1p1f1_gn_2050.nc is WSG84, 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 NetCDF file into Stata. Then 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 NetCDF 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
```

Calculate Average temperature for Hunan

This part illustrates how to calculate the average temperature for Hunan with `zonalstats` command. In this example, we perform zonal statistics on temperature data for January 1, 2025.

```
. 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)
.
. zonalstats `url´ using "hunan.shp", var(tas) stats(avg) origin(1 `lat_start´
>  `lon_start´) size(1 `lat_count´ `lon_count´) crs(EPSG:4326) clear
NetCDF variable ´tas´ type: float
NetCDF variable ´tas´ missing value attribute: 1.0E20 (type: float)
NetCDF CRS not detected. Using user-provided CRS: EPSG:4326
Coordinate systems are compatible, no reprojection needed
Created numeric variable: avg
Data successfully exported to Stata dataset.
.
. replace avg = avg - 273.15
(14 real changes made)
.
. save "hunan_temp.dta", replace
```

```
file hunan_temp.dta saved
```

Figure 5 displays the results of the `zonalstats` command above presents, showing the temperature distribution in Hunan on January 1, 2025.

```
. use hunan.dta, clear
.
. rename Name z_Name
. merge 1:1 z_Name using hunan_temp.dta,nogen
(variable z_Name was str11, now str16 to accommodate using
       data´s values)
    Result                      Number of obs
    ─────────────────────────────────────────
    Not matched                             0
    Matched                                14
    ─────────────────────────────────────────

. save hunan_temp.dta, replace
file hunan_temp.dta saved
.
. geoframe create region ///
>  "hunan_temp.dta", id(_ID) centroids(_CX _CY) ///
>  shp(hunan_shp.dta) ///
>  replace
(creating frame region from hunan_temp.dta)
(creating frame region_shp from hunan_shp.dta)
            Frame name: region [make current]
            Frame type: attribute
          Feature type: <none>
         Number of obs: 14
               Unit ID: _ID
           Coordinates: _CX _CY
    Linked shape frame: region_shp
.
. geoplot ///
>  (area region avg, color(Greys) ///
>         level(6, quantile weight(avg))) ///
>  (line region, lwidth(vthin)), ///
>  legend(position(sw))
```

Calculate 80km-radius IDW temperature for cities

Moving beyond the area-based aggregation using SHP files, this step utilizes the `matchgeop` command to perform Inverse Distance Weighting (IDW) interpolation. We perform calculations for grid points within an 80 km radius of each city and the IDW temperatures. By estimating values based on proximity to the urban core, we gain a perspective on climate conditions most relevant to city centers.

```
. use "grid_all.dta", clear
. rename lon ulon
. rename lat ulat
. gen n=_n
. save "grid_all_2.dta", replace
```

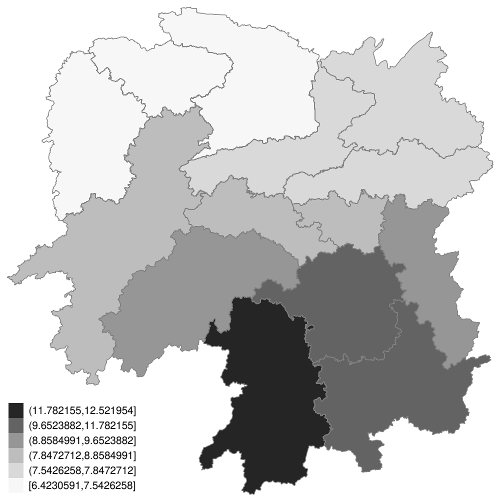

Figure 5: Average temperature in Hunan

```
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) within(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

. 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   13.184405   01jan2050    7.65365 │
  2. │ Changde   58.544605   02jan2050   7.220788 │
  3. │ Changde   74.443283   03jan2050    8.44021 │
  4. │ Changde   72.578667   04jan2050   6.136468 │
  5. │ Changde   38.741573   05jan2050   7.520319 │
     ├──────────────────────────────────────────┤
  6. │ Changde   78.467232   06jan2050   7.391534 │
  7. │ Changde   13.184405   07jan2050   6.704889 │
  8. │ Changde   16.299124   08jan2050   3.407739 │
  9. │ Changde   48.368313   09jan2050   2.457117 │
 10. │ Changde   39.140995   10jan2050   2.555414 │
     └──────────────────────────────────────────┘

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

Figure 6 presents the resulting IDW 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
. rename city Name
. merge m:1 Name using hunan.dta
    Result                      Number of obs
    ─────────────────────────────────────────
    Not matched                             0
    Matched                             5,110  (_merge==3)
    ─────────────────────────────────────────

. local dates "01jan2050 01jul2050"
. local suffixes "202500101 202500701"
.
. local i = 1
. foreach d of local dates {
  2.     local s : word `i´ of `suffixes´
  3.     preserve
  4.     keep if date == date("`d´", "DMY")
  5.     save hunan_IDW_`s´.dta, replace
  6.
.     geoframe create region ///
>      "hunan_IDW_`s´.dta", id(_ID) centroids(_CX _CY) ///
>      shp(hunan_shp.dta) ///
>      replace
  7.
.     geoplot ///
>      (area region temp_c , color(Greys) ///
>         level(6, quantile weight(temp_c))) ///
>      (line region, lwidth(vthin)), ///
>      legend(position(sw)) ///
>      title("Temperature(`s´)")
  8.
.     restore
  9.
.     graph save Temperature(`s´), replace
 10.
.     local ++i
 11. }
(5,096 observations deleted)
file hunan_IDW_202500101.dta saved
(creating frame region from hunan_IDW_202500101.dta)
(creating frame region_shp from hunan_shp.dta)
            Frame name: region [make current]
            Frame type: attribute
          Feature type: <none>
         Number of obs: 14
               Unit ID: _ID
           Coordinates: _CX _CY
    Linked shape frame: region_shp
file Temperature(202500101).gph saved
(5,096 observations deleted)
file hunan_IDW_202500701.dta saved
(creating frame region from hunan_IDW_202500701.dta)
(creating frame region_shp from hunan_shp.dta)
            Frame name: region [make current]
            Frame type: attribute
          Feature type: <none>
         Number of obs: 14
               Unit ID: _ID
           Coordinates: _CX _CY
    Linked shape frame: region_shp
file Temperature(202500701).gph saved
.
. graph combine Temperature(202500101).gph Temperature(202500701).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 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).

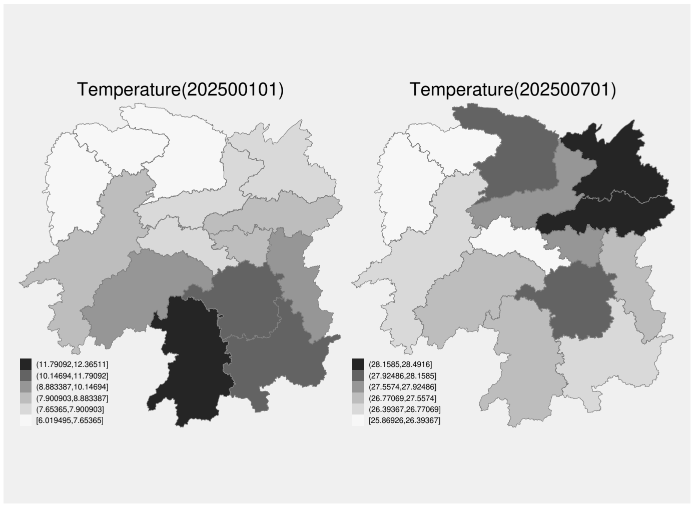

Figure 6: IDW Temperature at the geocentric center of cities in Hunan

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.

Wu, K. C. Z. . L. S. C. Z., Yizhen; Shi. 2021. An improved time-series DMSP-OLS-like data (1992-2024) in China by integrating DMSP-OLS and SNPP-VIIRS. https://doi.org/10.7910/DVN/GIYGJU.

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 , China. 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.