Lecture 4: Software development II

CEE 690

Intro to “structural encapsulation”

```
 99     return
100
101 def main():
102
103     """
104     The director of the orchestra. When this function is called, it runs the defined
105     sequence of functions. However, it also ensures that other parts of the script can
106     be accessed without running this.
107     """
108
109     # Configuration variables
110     INPUT_FILE = 'era_interim_monthly_197901_201512_upscaled_annual.nc'
111     OUTPUT_FILE = 'out.nc'
112     PLOT_FILE = 'plot.png'
113     VAR_NAME = 't2m'
114     LAT_MIN = 5
115     LAT_MAX = 50
116     LON_MIN = 10
117     LON_MAX = 100
118     TIME_MIN = 0
119     TIME_MAX = 10
120
121     # Load dataset
122     print("Loading the dataset")
123     t2m_data = load_dataset(INPUT_FILE,VAR_NAME)
124
125     # Compute temporal series of spatial mean and spatial standard deviation
126     print("Computing the statistics")
127     temporal_spatial_mean = calculate_spatial_mean(t2m_data,TIME_MIN,TIME_MAX,
128                                                    LAT_MIN,LAT_MAX,LON_MIN,LON_MAX)
```

```
143 if __name__ == "__main__":
144
145     main()
```

__name__

In spatialstats.py →

```
141     return
142
143 print('__name__=%s' % __name__)
144 if __name__ == "__main__":
145     main()
```

Opt 1 - Run script that contains all the functions (spatialstats.py)

```
(cee690) [20:56:37] nc153@dcc-login-02:~/CEE690/Miscellanous/refactor $ python spatialstats.py
__name__=__main__
Loading the dataset
Computing the statistics
Visualizing the data
Saving the computed statistics to netcdf
```

Opt 2 - Run script that imports the functions from spatialstats.py (driver.py)

driver.py →

```
1 import spatialstats
2
```

```
(cee690) [20:58:21] nc153@dcc-login-02:~/CEE690/Miscellanous/refactor $ python driver.py
__name__=spatialstats
```

Double underscore (dunder) variables: “Hidden plumbing of Python”

Under variables are predefined for each script/module

```
144 if __name__ == "__main__":
145     print(dir())
146     main()
```

```
(cee690) [21:12:00] nc153@dcc-login-02:~/CEE690/Miscellanous/refactor $ python spatialstats.py
__name__=__main__
['__builtins__', '__cached__', '__doc__', '__file__', '__loader__', '__name__', '__package__', '__sp
ec__', 'calculate_spatial_mean', 'calculate_spatial_variance', 'load_dataset', 'main', 'matplotlib',
 'nc', 'np', 'output_data_to_netcdf', 'plt', 'visualize_data']
```

Double underscore (dunder) variables: “Hidden plumbing of Python”

Under variables are predefined for each script/module

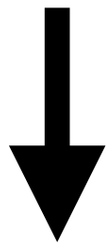
```
144 if __name__ == "__main__":
145     print(dir())
146     main()
```

```
(cee690) [21:12:00] nc153@dcc-login-02:~/CEE690/Miscellanous/refactor $ python spatialstats.py
__name__=__main__
['__builtins__', '__cached__', '__doc__', '__file__', '__loader__', '__name__', '__package__', '__sp
ec__', 'calculate_spatial_mean', 'calculate_spatial_variance', 'load_dataset', 'main', 'matplotlib',
 'nc', 'np', 'output_data_to_netcdf', 'plt', 'visualize_data']
```

Dunder variable: __name__

Name of the scope in which the code is currently executing

```
141     return
142
143 print('__name__=%s' % __name__)
144 if __name__ == "__main__":
145     main()
```

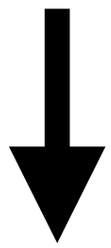


```
(cee690) [20:56:37] nc153@dcc-login-02:~/CEE690/Miscellanous/refactor $ python spatialstats.py
__name__=__main__
Loading the dataset
Computing the statistics
Visualizing the data
Saving the computed statistics to netcdf
```

Dunder variable: __file__

Absolute file path of the current script

```
143 print('__file__=%s' % __file__)
144 if __name__ == "__main__":
145     print(dir())
146     main()
```



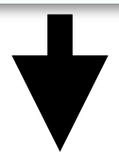
```
(cee690) nc153 $ python spatialstats.py
__file__=/hpc/home/nc153/CEE690/Miscellanous/refactor/spatialstats.py
```

Dunder variable: __doc__

Stores the doctoring defined per module, class, function, etc…

```
1 """
2 This script provides some basic spatial stats to compute on
3 some input data and spatial and temporal coordinates.
4 """
5
6 import netCDF4 as nc
7 import numpy as np
```

```
148 print('__doc__=%s' % __doc__)
149 if __name__ == "__main__":
150     print(dir())
151     main()
```

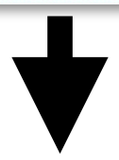


```
(cee690) nc153 $ python spatialstats.py
__doc__=
This script provides some basic spatial stats to compute on
some input data and spatial and temporal coordinates.
```

Dunder variable: __all__

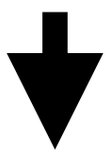
The list you provide what is imported during a wildcard import

```
148 __all__ = ['calculate_spatial_mean','calculate_spatial_variance']
149 print('__all__=%s' % __all__)
```



```
(cee690) nc153 $ python spatialstats.py
__all__=['calculate_spatial_mean', 'calculate_spatial_variance']
```

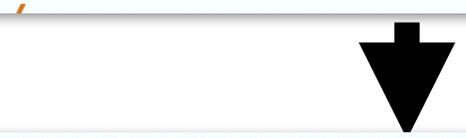
```
1 from spatialstats import *
2 print(dir())
```



```
(cee690) nc153 $ python driver.py
__all__=['calculate_spatial_mean', 'calculate_spatial_variance']
['__builtins__', '__cached__', '__doc__', '__file__', '__loader__', '_
ec__', 'calculate_spatial_mean', 'calculate_spatial_variance']
```

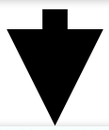
Detour: Does how we import make a difference?

```
1 import spatialstats
2 print(dir(spatialstats))
3
```



```
(cee690) nc153 $ python driver.py
__all__=['calculate_spatial_mean', 'calculate_spatial_variance']
['__all__', '__builtins__', '__cached__', '__doc__', '__file__', '__loader__', '__name__',
e__', '__spec__', 'calculate_spatial_mean', 'calculate_spatial_variance', 'load_dataset', '
atplotlib', 'nc', 'np', 'output_data_to_netcdf', 'plt', 'visualize_data']
```

```
1 from spatialstats import *
2 print(dir())
3
```



```
(cee690) nc153 $ python driver.py
__all__=['calculate_spatial_mean', 'calculate_spatial_variance']
['__builtins__', '__cached__', '__doc__', '__file__', '__loader__
ec__', 'calculate_spatial_mean', 'calculate_spatial_variance']
```

Note: The entire module is still loaded into memory in the second case; what changes is that is “visible”.

Detour: Does how we import make a difference?

```
1 import spatialstats
2 print(dir(spatialstats))
3
```

We will come back to dunder variables/methods when we cover classes in a few slides. However, let's first fix something else from our maturing software

```
3
```

```
(cee690) nc153 $ python driver.py
__all__=['calculate_spatial_mean', 'calculate_spatial_variance']
['__builtins__', '__cached__', '__doc__', '__file__', '__loader__
ec__', 'calculate_spatial_mean', 'calculate_spatial_variance']
```

Note: The entire module is still loaded into memory in the second case; what changes is that is "visible".

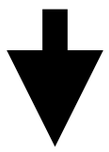
The general configuration variables are still hardcoded

```
106 def main():
107 
108     """
109     The director of the orchestra. When this function is called, it runs the defined
110     sequence of functions. However, it also ensures that other parts of the script can
111     be accessed without running this.
112     """
113 
114     # Configuration variables
115     INPUT_FILE = 'era_interim_monthly_197901_201512_upscaled_annual.nc'
116     OUTPUT_FILE = 'out.nc'
117     PLOT_FILE = 'plot.png'
118     VAR_NAME = 't2m'
119     LAT_MIN = 5
120     LAT_MAX = 50
121     LON_MIN = 10
122     LON_MAX = 100
123     TIME_MIN = 0
124     TIME_MAX = 10
```

Any ideas on how to fix this?

One option: sys.argv

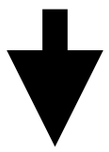
```
1 import sys
2 print('argv 0:',sys.argv[0])
3 print('argv 1:',sys.argv[1])
4 print('argv 2:',sys.argv[2])
5 print('argv 3:',sys.argv[3])
```



```
nc153 $ python test.py input1 input2 input3
argv 0: test.py
argv 1: input1
argv 2: input2
argv 3: input3
```

One option: sys.argv

```python
import sys
print('argv 0:',sys.argv[0])
print('argv 1:',sys.argv[1])
print('argv 2:',sys.argv[2])
print('argv 3:',sys.argv[3])
```



```
nc153 $ python test.py input1 input2 input3
argv 0: test.py
argv 1: input1
argv 2: input2
argv 3: input3
```

It is too basic. Use argparse instead.

Command line arguments: Argparse library

```
2 This script provides some basic spatial stats to compute on
3 some input data and spatial and temporal coordinates.
4 """
5 import argparse
6 import netCDF4 as nc
7 import numpy as np
```

Define function to read arguments

```
141 def get_args():
142     """Defines and collects command line arguments."""
143     parser = argparse.ArgumentParser(
144         description="Process spatial statistics from netCDF4 data."
145     )
146 
147     # Input/Output paths
148     parser.add_argument('--INPUT_FILE', type=str, default='era_interim_monthly_197901_201512_upscal
    ual.nc',
149                         help='Path to the input NetCDF file')
150     parser.add_argument('--OUTPUT_FILE', type=str, default='out.nc',
151                         help='Name of the output NetCDF file')
152     parser.add_argument('--PLOT_FILE', type=str, default='plot.png',
153                         help='Name of the output diagnostic plot')
154     parser.add_argument('--VAR_NAME', type=str, default='t2m',
155                         help='The variable name to extract from the NetCDF')
156 
157     # Bounding Box Arguments
158     parser.add_argument('--LAT_MIN', type=int, default=5)
159     parser.add_argument('--LAT_MAX', type=int, default=50)
160     parser.add_argument('--LON_MIN', type=int, default=10)
161     parser.add_argument('--LON_MAX', type=int, default=100)
162     parser.add_argument('--TIME_MIN', type=int, default=0)
163     parser.add_argument('--TIME_MAX', type=int, default=10)
164 
165     # Optional JSON config file path
166     parser.add_argument('--JSON_FILE', type=str, default=None)
167 
168     return parser.parse_args()
```

Include in the “main” function

```
106 def main():
107 
108     """
109     The director of the orchestra. When this function is called, it runs the defined
110     sequence of functions. However, it also ensures that other parts of the script can
111     be accessed without running this.
112     """
113 
114     # Gather arguments from the terminal and convert to dictionary
115     config = vars(get_args())
116 
117     # Load dataset
118     print("Loading the dataset")
119     t2m_data = load_dataset(config['INPUT_FILE'],config['VAR_NAME'])
120 
121     # Compute temporal series of spatial mean and spatial standard deviation
```

Include in the “main” function

```
106 def main():
107 
108     """
109     The director of the orchestra. When this function is called, it runs the defined
110     sequence of functions. However, it also ensures that other parts of the script can
111     be accessed without running this.
112     """
113 
114     # Gather arguments from the terminal and convert to dictionary
115     config = vars(get_args())
116 
117     # Load dataset
118     print("Loading the dataset")
119     t2m_data = load_dataset(config['INPUT_FILE'],config['VAR_NAME'])
120 
121     # Compute temporal series of spatial mean and spatial standard deviation
```

I did something else here as well. What is it?

Include in the “main” function

```
106 def main():
107 
108     """
109     The director of the orchestra. When this function is called, it runs the defined
110     sequence of functions. However, it also ensures that other parts of the script can
111     be accessed without running this.
112     """
113 
114     # Gather arguments from the terminal and convert to dictionary
115     config = vars(get_args())
116 
117     # Load dataset
118     print("Loading the dataset")
119     t2m_data = load_dataset(config['INPUT_FILE'],config['VAR_NAME'])
120 
121     # Compute temporal series of spatial mean and spatial standard deviation
```

I did something else here as well. What is it?

Config variables should be in a dictionary or something like it to not have to carry them around

And magic!

```
(cee690) nc153 $ python spatialstats.py --help
usage: spatialstats.py [-h] [--INPUT_FILE INPUT_FILE] [--OUTPUT_FILE OUTPUT_FILE] [--PLOT_FILE PLOT_FILE]
                       [--VAR_NAME VAR_NAME] [--LAT_MIN LAT_MIN] [--LAT_MAX LAT_MAX] [--LON_MIN LON_MIN]
                       [--LON_MAX LON_MAX] [--TIME_MIN TIME_MIN] [--TIME_MAX TIME_MAX]
                       [--JSON_FILE JSON_FILE]

Process spatial statistics from netCDF4 data.

options:
  -h, --help            show this help message and exit
  --INPUT_FILE INPUT_FILE
                        Path to the input NetCDF file
  --OUTPUT_FILE OUTPUT_FILE
                        Name of the output NetCDF file
  --PLOT_FILE PLOT_FILE
                        Name of the output diagnostic plot
  --VAR_NAME VAR_NAME   The variable name to extract from the NetCDF
  --LAT_MIN LAT_MIN
  --LAT_MAX LAT_MAX
  --LON_MIN LON_MIN
  --LON_MAX LON_MAX
  --TIME_MIN TIME_MIN
  --TIME_MAX TIME_MAX
  --JSON_FILE JSON_FILE
(cee690) nc153 $
```

Examples

1. Changing the bounding box (slightly)

```
(cee690) nc153 $ python spatialstats.py --lat_max 30 --lat_min 20
Loading the dataset
Computing the statistics
Visualizing the data
Saving the computed statistics to netcdf
```

2. Defining a non-existent dataset

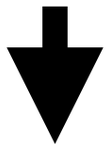
```
(cee690) nc153 $ python spatialstats.py --input_file input.nc
Loading the dataset
Traceback (most recent call last):
  File "/hpc/home/nc153/CEE690/Miscellanous/refactor/spatialstats.py", line 178, in <module>
    main()
    ~~~~^^
  File "/hpc/home/nc153/CEE690/Miscellanous/refactor/spatialstats.py", line 131, in main
    t2m_data = load_dataset(INPUT_FILE,VAR_NAME)
  File "/hpc/home/nc153/CEE690/Miscellanous/refactor/spatialstats.py", line 73, in load_dataset
    file_pointer_input = nc.Dataset(input_file,'r')
  File "src/netCDF4/_netCDF4.pyx", line 2521, in netCDF4._netCDF4.Dataset.__init__
  File "src/netCDF4/_netCDF4.pyx", line 2158, in netCDF4._netCDF4._ensure_nc_success
FileNotFoundError: [Errno 2] No such file or directory: 'input.nc'
```

Another option... JSONs

```
 1 {
 2   "INPUT_FILE": "era_interim_monthly_197901_201512_upscaled_annual.nc",
 3   "OUTPUT_FILE": "output_temporal_stats.nc",
 4   "PLOT_FILE": "output_plot.png",
 5   "VAR_NAME": "t2m",
 6   "LAT_MIN": 5,
 7   "LAT_MAX": 50,
 8   "LON_MIN": 10,
 9   "LON_MAX": 100,
10   "TIME_MIN": 0,
11   "TIME_MAX": 10
12 }
~
~
~
~
~
~
"example.json" 12L, 278B
```

JSONs

```
115     # Gather arguments from the terminal and convert to dictionary
116     config = vars(get_args())
117
118     # Override with JSON info if present
119     if config.get('JSON_FILE'):
120         with open(config['JSON_FILE'], 'r') as f:
121             json_config = json.load(f)
122             # This one line replaces all the manual overwriting
123             config.update(json_config)
124
125     # Load dataset
126     print("Loading the dataset")
127     t2m_data = load_dataset(config['INPUT_FILE'],config['VAR_NAME'])
```



```
(cee690) nc153 $ python spatialstats.py --JSON_FILE=example.json
Loading the dataset
Computing the statistics
Visualizing the data
Saving the computed statistics to netcdf
```

Too many variables as args

```
126     print("Loading the dataset")
127     t2m_data = load_dataset(config['INPUT_FILE'],config['VAR_NAME'])
128
129     # Compute temporal series of spatial mean and spatial standard deviation
130     print("Computing the statistics")
131     temporal_spatial_mean = calculate_spatial_mean(t2m_data,config['TIME_MIN'],config['TIME_MAX'],
132                                                    config['LAT_MIN'],config['LAT_MAX'],
133                                                    config['LON_MIN'],config['LON_MAX'])
134     temporal_spatial_variance = calculate_spatial_variance(t2m_data,config['TIME_MIN'],
135                                                            config['TIME_MAX'],config['LAT_MIN'],
136                                                            config['LAT_MAX'],config['LON_MIN'],
137                                                            config['LON_MAX'],temporal_spatial_mean)
138
139     #Visualize the data
140     print("Visualizing the data")
141     visualize_data(temporal_spatial_mean,temporal_spatial_variance,config['PLOT_FILE'])
142
143     #Output the data to netcdf
144     print("Saving the computed statistics to netcdf")
145     output_data_to_netcdf(config['OUTPUT_FILE'],temporal_spatial_mean,temporal_spatial_variance)
```

```
125     # Load dataset
126     print("Loading the dataset")
127     t2m_data = load_dataset(config)
128
129     # Compute temporal series of spatial mean and spatial standard deviation
130     print("Computing the statistics")
131     temporal_spatial_mean = calculate_spatial_mean(t2m_data, config)
132     temporal_spatial_variance = calculate_spatial_variance(t2m_data, config, temporal_spatial_mean)
133
134     #Visualize the data
135     print("Visualizing the data")
136     visualize_data(temporal_spatial_mean, temporal_spatial_variance, config)
137
138     #Output the data to netcdf
139     print("Saving the computed statistics to netcdf")
140     output_data_to_netcdf(config['OUTPUT_FILE'], temporal_spatial_mean, temporal_spatial_variance)
```

Defensive programming

Writing programs expecting things to go wrong

Defensive programming: Pre-emption

```
80      # Check if file exists
81      if not os.path.exists(config['INPUT_FILE']):
82          print(f"Error: The file {config['INPUT_FILE']} does not exist.")
83          sys.exit(1)
84
```

Defensive programming: Bounds check

```
21      # Validate indices against data shape
22      if config['TIME_MAX'] > data.shape[0] or config['LAT_MAX'] > data.shape[1] or conf
23          print("Error: Configuration indices are out of data bounds.")
24          sys.exit(1)
```

Defensive programming: Variable existence

```
# Check if variable exists in file
if config['VAR_NAME'] not in file_pointer_input.variables:
    print(f"Error: Variable '{config['VAR_NAME']}' not found in {config['INPUT_FILE']}.")
    file_pointer_input.close()
    sys.exit(1)
```

Catch-all: Try/except

```
105     try:
106         # Plot and save the time series
107         plt.plot(temporal_spatial_mean, label="Mean")
108         plt.plot(temporal_spatial_variance, label="Variance")
109         plt.legend()
110         plt.savefig(config['PLOT_FILE'])  # Saves directly to disk
111         plt.close()
112     except Exception as e:
113         print(f"Error during visualization: {e}")
```

Move to vectorization

When possible, drop the nested for loops

```python
def calculate_spatial_mean(data, config):
    """
    Computes spatial mean using NumPy slicing (Vectorization).
    """
    # 1. Define the time range
    t_start, t_end = config['TIME_MIN'], config['TIME_MAX']

    # 2. Extract the 3D 'cube' of interest all at once
    # Syntax: data[time_slice, lat_slice, lon_slice]
    subset = data[t_start:t_end,
                  config['LAT_MIN']:config['LAT_MAX'],
                  config['LON_MIN']:config['LON_MAX']]

    # 3. Collapse the spatial dimensions (axis 1 and 2)
    # This calculates the mean across Lat and Lon for every time step
    temporal_spatial_mean = np.mean(subset, axis=(1, 2))

    return temporal_spatial_mean
```

Reminder that Numba actually relaxes this requirement a bit

Moving to object-oriented programming

```
130     # Load dataset
131     print("Loading the dataset")
132     t2m_data = load_dataset(config)
133
134     # Compute temporal series of spatial mean and spatial standard deviation
135     print("Computing the statistics")
136     temporal_spatial_mean = calculate_spatial_mean(t2m_data, config)
137     temporal_spatial_variance = calculate_spatial_variance(t2m_data, config, temporal_spatial_mean)
138
139     #Visualize the data
140     print("Visualizing the data")
141     visualize_data(temporal_spatial_mean, temporal_spatial_variance, config)
142
143     #Output the data to netcdf
144     print("Saving the computed statistics to netcdf")
145     output_data_to_netcdf(config['OUTPUT_FILE'], temporal_spatial_mean, temporal_spatial_variance)
```

Let’s talk about the inefficiencies here

Moving to object-oriented programming

```
130     # Load dataset
131     print("Loading the dataset")
132     t2m_data = load_dataset(config)
133
134
135
136
137                                                                    an)
138
139
140
141
142
143     #Output the data to netcdf
144     print("Saving the computed statistics to netcdf")
145     output_data_to_netcdf(config['OUTPUT_FILE'], temporal_spatial_mean, temporal_spatial_variance)
```

In object-oriented programming (OOP), we bind the data and logic together into one unit. This minimizes the “passing around”.

Let’s talk about the inefficiencies here

Implementing in the code

```
class SpatialAnalyzer:
    def __init__(self, config):
        """
        Initializes the analyzer with configuration and loads the dataset.
        In OOP, __init__ is the 'constructor' that sets up the object's state.
        """
        self.config = config
        self.data = self._load_dataset()
        self.means = None
        self.variances = None

    def _load_dataset(self):
        """Internal helper to load and validate data."""
        if not os.path.exists(self.config['INPUT_FILE']):
            print(f"Error: The file {self.config['INPUT_FILE']} does not exist.")
            sys.exit(1)

        try:
            file_pointer = nc.Dataset(self.config['INPUT_FILE'], 'r')
            if self.config['VAR_NAME'] not in file_pointer.variables:
                print(f"Error: Variable '{self.config['VAR_NAME']}' not found.")
                file_pointer.close()
                sys.exit(1)

            data = file_pointer.variables[self.config['VAR_NAME']][:]
            file_pointer.close()
```

Implementing in the code

```
class SpatialAnalyzer:
    def __init__(self, config):
        """
            return data
        except Exception as e:
            print(f"An unexpected error occurred while loading: {e}")
            sys.exit(1)

    def run_analysis(self):
        """Orchestrates the computation of statistics."""
        print("Computing the statistics")
        t_start, t_end = self.config['TIME_MIN'], self.config['TIME_MAX']

        # Slicing the subset once to be used by both mean and variance
        subset = self.data[t_start:t_end,
                           self.config['LAT_MIN']:self.config['LAT_MAX'],
                           self.config['LON_MIN']:self.config['LON_MAX']]

        self.means = np.mean(subset, axis=(1, 2))
        self.variances = np.var(subset, axis=(1, 2))

    def visualize(self):
        """Generates the diagnostic plot."""
        if self.means is None:
            print("Error: No analysis results to visualize.")
            return

        print("Visualizing the data")
        try:
```

Implementing in the code

```python
class SpatialAnalyzer:
    def __init__(self, config):
        """
```

```python
            return data
        except Exception as e:
```

```python
def main():
    # 1. Setup Configuration
    config = vars(get_args())
    if config.get('JSON_FILE'):
        try:
            with open(config['JSON_FILE'], 'r') as f:
                config.update(json.load(f))
        except Exception as e:
            print(f"Error loading JSON config: {e}")
            sys.exit(1)

    # 2. OOP Execution
    # Create the object (this loads the data)
    analyzer = SpatialAnalyzer(config)

    # Run the methods
    analyzer.run_analysis()
    analyzer.visualize()
    analyzer.save_netcdf()

    return

if __name__ == "__main__":
    main()
```

Dunder methods for classes

```
15 class SpatialAnalyzer:
16     def __init__(self, config):
17         """
18         Initializes the analyzer with configuration and loads the dataset.
19         In OOP, __init__ is the 'constructor' that sets up the object's state.
20         """
21         self.config = config
22         self.data = self._load_dataset()
23         self.means = None
24         self.variances = None
25
```

The __init__ statement is basically saying “do this when instantiating this class”

The methods can now permanently “see” the config variables

The methods can now permanently “see” the config variables

```
            return data
        except Exception as e:
            print(f"An unexpected error occurred while loading: {e}")
            sys.exit(1)

    def run_analysis(self):
        """Orchestrates the computation of statistics."""
        print("Computing the statistics")
        t_start, t_end = self.config['TIME_MIN'], self.config['TIME_MAX']

        # Slicing the subset once to be used by both mean and variance
        subset = self.data[t_start:t_end,
                           self.config['LAT_MIN']:self.config['LAT_MAX'],
                           self.config['LON_MIN']:self.config['LON_MAX']]

        self.means = np.mean(subset, axis=(1, 2))
        self.variances = np.var(subset, axis=(1, 2))

    def visualize(self):
        """Generates the diagnostic plot."""
        if self.means is None:
            print("Error: No analysis results to visualize.")
            return

        print("Visualizing the data")
        try:
```

self._load_dataset()

```
15 class SpatialAnalyzer:
16     def __init__(self, config):
17         """
18         Initializes the analyzer with configuration and loads the dataset.
19         In OOP, __init__ is the 'constructor' that sets up the object's state.
20         """
21         self.config = config
22         self.data = self._load_dataset()
23         self.means = None
24         self.variances = None
25
```

The underscore is indicating to the user or future self that this method is “internal” and should not be used outside its intended purpose