Table of Contents

Introduction

Welcome to the training guide for the **`wdc`** library. This guide will walk you through how to set up and start using wdc for working with coverage data in your projects. The wdc library is designed to facilitate easy interaction with coverage datasets, often used in geospatial data processing and analysis.

Classes

In this library, we utilize two primary classes:

- **`dbc` class (Database Connection):** This class is designed to establish a connection between your application and the server where the data coverages are stored. When you create an instance of the **`dbc`** class, it acts as a gateway, enabling the **`dco`** instances to communicate with the server to access the data.
- **`dco` class (Data Coverage Operations):** This class is used to define and collect the operations you wish to perform on the data coverages. After specifying your operations in a **`dco`** instance, it interacts with an instance of the **`dbc`** class to execute these operations and retrieve the necessary data from the server.

Methods Available in the dco Class

To work with an instance from the `dco` class, you can use the following methods:

initialize_var()

Description: This method initializes a variable for the datacube object. It is used to prepare variables for data operations by extracting and verifying their names and associated coverages. You can initialize as many variables as you need. This method must come first after you create a `dco` instance.
Parameters:

- **s (str):** A string that combines the variable name and its associated datacube, formatted as: `$variable_name in (coverage_name)` Keep in mind that you have to have no extra spaces. Otherwise, you will get a ValueError.
  Example:

```
datacube.initialize_var("$c in (AvgLandTemp)")
```

where()

Description:
The `where()` method sets a filter condition for querying the datacube, akin to the SQL `WHERE` clause. It allows you to specify conditions to filter data based on defined criteria. The condition is applied during the execution of the query to selectively filter results. It is applicable to scalar arguments only.

Parameters:

- **filter_condition (str)**: A string that defines the filter condition. This string must adhere to the WCPS syntax rules. The string must contain existing variable name.

Example: Applying a filter condition to a datacube query

```
datacube.where("$c > 20 and $c < 50")
```

subset()

Description:
The `subset()` method is used to specify a subset for a particular variable within the datacube object. This method involves selecting a segment of the data by defining conditions such as ranges or specific filters. It associates these conditions with a named variable, effectively updating how the data will be retrieved when queries are executed.

Parameters:

- **subset (str)**: A string that defines the subset condition, such as a range or a specific filter that determines how data is accessed. Example formats can include geographic coordinates or time ranges.
- **var_name (str)**: The name of the variable to which the subset conditions will be applied. This variable must have been previously initialized within the dco instance.

Exceptions:

- **ValueError**: Raises an error if the specified variable name does not exist in the initialized variable list of the dco instance.

Example Usage:

```
# Applying a geographic and temporal subset to a variable within th
e datacube
datacube.subset(var_name='$c', subset='Lat(53.08), Long(8.80), ansi
("2014-01":"2014-12")')
```

set_format()

Description:
The `set_format()` method configures the output format of the data returned from queries executed on the datacube. This flexibility allows you to specify how the data should be formatted, facilitating compatibility with various data processing requirements or visualization tools.

Parameters:

- **output_format (str)**: Specifies the desired format for the output data. Acceptable values are:
  - `"CSV"` for comma-separated values.
  - `"PNG"` for Portable Network Graphics format.
  - `"JPEG"` for Joint Photographic Experts Group format.

Example Usage:

transform_data()

Description:
This method configures a transformation operation that will be applied to the data within the datacube when the query is executed. This functionality allows for dynamic data manipulation based on specified mathematical or conditional operations directly within the query.

Parameters:

- **operation (str):** A string that describes the transformation operation to apply to the data. The operation should be formulated in a way that conforms to the expected syntax and functionality of the datacube's processing capabilities.

Example Usage:

```
# Applying a transformation to adjust data values within the datacu
be
datacube.transform_data("abs($c - 3.6 * $c)")
```

encode()

Description:
The `encode()` method specifies the encoding operation that will be applied to the output of the query. This method is particularly useful for defining how data should be visualized or transformed before it is presented or further processed.

Parameters:

- **operation (str):** A string representing the encoding function. This should include detailed encoding specifications, such as conditions for data representation or color mapping based on data values. The string must contain existing variable name.

Example Usage:

```
# Applying conditional encoding to style the output based on data v
alues
datacube.encode("""
    switch
        case $c = 99999
            return {red: 255; green: 255; blue: 255}
        case 18 > $c
            return {red: 0; green: 0; blue: 255}
        case 23 > $c
            return {red: 255; green: 255; blue: 0}
        case 30 > $c
            return {red: 255; green: 140; blue: 0}
        default return {red: 255; green: 0; blue: 0}
""")
```

execute()

Description:
Executes the constructed WCPS (Web Coverage Processing Service) query and processes the response according to the specified output format. This method handles the entire lifecycle of data retrieval from the server, from sending the query to processing and returning the formatted data.

Example Usage:

```
# Executing a query and receiving the formatted output
output = datacube.execute()
print(output)
```

Aggregation methods

The following methods allow you to apply aggregation functions to the coverage data. Only one aggregation function may be used at a time. Additionally, you cannot specify the output format when using any of these aggregation methods.

min()

Description:
Configures the datacube to compute the minimum value of a specified data subset, with an optional condition to narrow down the data selection.

Parameters:

- **condition (str, optional):** Defines the subset of data for which the minimum is calculated. The condition should be a string formatted according to the expected WCPS syntax. The string must contain existing variable name.

Example Usage:

```
datacube.min("$c > 20")
```

max()

Description:
Enables the datacube to compute the maximum value of the defined data subset. Users can specify a condition to filter the data.

Parameters:

- **condition (str, optional):** Specifies the criteria for selecting the subset of data for which the maximum is calculated. The string must contain existing variable name.

Example Usage:

```
datacube.max("$c > 20")
```

avg()

Description:
Sets the datacube to calculate the average value of a specified data subset. An optional condition can refine the data aggregation criteria.

Parameters:

- **condition (str, optional):** Condition that defines the data subset for which the average is computed. The string must contain existing variable name.

Example Usage:

```
datacube.avg("$c > 20")
```

sum()

Description:
Configures the datacube to sum values across a specified data subset. The operation can include an optional condition to specify which data to aggregate.

Parameters:

- **condition (str, optional):** Condition that defines the data subset for which the sum is computed. The string must contain existing variable name.

Example Usage:

```
datacube.sum("$c > 20")
```

count()

Description:
Allows the datacube to count the number of data points that meet a specified condition during execution.

Parameters:

- **condition (str, optional):** Specifies the criteria that data points must meet to be included in the count. The string must contain existing variable name.

Example Usage:

```
datacube.count("$c > 20")
```

Setup and Installation

To use the **`wdc`** library effectively, you need to import several components. Here's what you need and how to do it:

In [1]:
```
# Importing the Image module to display images in the notebook
from IPython.display import Image

# Importing the main wdc library
import wdc

# Optional: importing warnings to suppress warnings if needed
import warnings
warnings.filterwarnings("ignore")
```

Why These Imports?

- **`Image` from `IPython.display`:** Needed to convert data received from the wdc library into an image for visualization in the notebook. This is particularly useful when the format of the data is PNG or JPEG.
- **`wdc`:** The core library required to work with coverages.
- **(optional) `warnings`:** Useful for suppressing warnings during runtime, which helps in keeping the notebook output clean and focused on relevant information.

Getting started

First dco() instance

To begin working with the `dco` class, you first need to set up a connection with the database by creating an instance of the `dbc` class. Here's a simple example:

In [2]:
```
# 1st way to create a dbc() instance
# creating a dbc instance
my_dbc = wdc.dbc("https://ows.rasdaman.org/rasdaman/ows")
```

In [3]:
```
# 2nd way to create a dbc() instance
from wdc import dbc

my_dbc = dbc("https://ows.rasdaman.org/rasdaman/ows")
```

In this example `"https://ows.rasdaman.org/rasdaman/ows"` is an url of the server from which we can get a coverage.
After creating a `dbc()` instance, we can create our first `dco()` instance. We can create `dco()` instance in 2 ways.

In [4]:
```
# 1st way
# my_dco - is a dco() instance, which we created previously
my_dco = wdc.dco(my_dbc)
```

In [5]:
```
# 2nd way
from wdc import dco
my_dco = dco(my_dbc)
```

The order of the methods

In the initialize_var() section, it is said that we always need to use this method first. Let's do that. Our variable's name will be `$c` and the coverage's name that we will use is `AvgTemperatureColorScaled`. Note that variables always start with '$' sign.

In [6]:
```
my_dco.initialize_var('$c in ( AvgTemperatureColorScaled )')
```

Out[6]:
```
<wdc.dco at 0x1e8f1f096d0>
```

What if we try to initialize variable, but in the wrong format?

In [7]:
```
my_dco.initialize_var('$cin(AvgLandTemp)')
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[7], line 1
----> 1 my_dco.initialize_var('$cin(AvgLandTemp)')

File ~\Desktop\Projects\project_wcps\wdc.py:169, in dco.initialize_var(self, s)
    150 """
    151 Initializes a variable for the datacube object. This method extracts the variable name from
    152     the input,checks if it's successfully defined, and then appends the variable along with
   (...)
    166     >>> datacube.initialize_var("$c in (AvgLandTemp)")
    167 """
    168 if not(s.startswith('$') and " in (" in s and s.endswith(')')):
--> 169     raise ValueError("The format of variable initialization wasn't correct")
    171 var_name = self.get_all_var_names(s)[0]
    172 self.vars.append(s)

ValueError: The format of variable initialization wasn't correct
```

As we can see, we will get a ValueError.
After the variable initialization, we can use almost any method and in any order. However, keep in mind that variable names that you use must be defined previously. Additionally, `execute()` is always the last method that you need to use.
For now, let's take a subset: `'ansi("2014-07")'` and choose a format for our output to be `'PNG'`.

In [27]:
```
my_dco.subset(var_name='$c', subset='ansi("2014-07")')
my_dco.set_format('PNG')
```

Out[27]:
```
<wdc.dco at 0x2825381cd90>
```

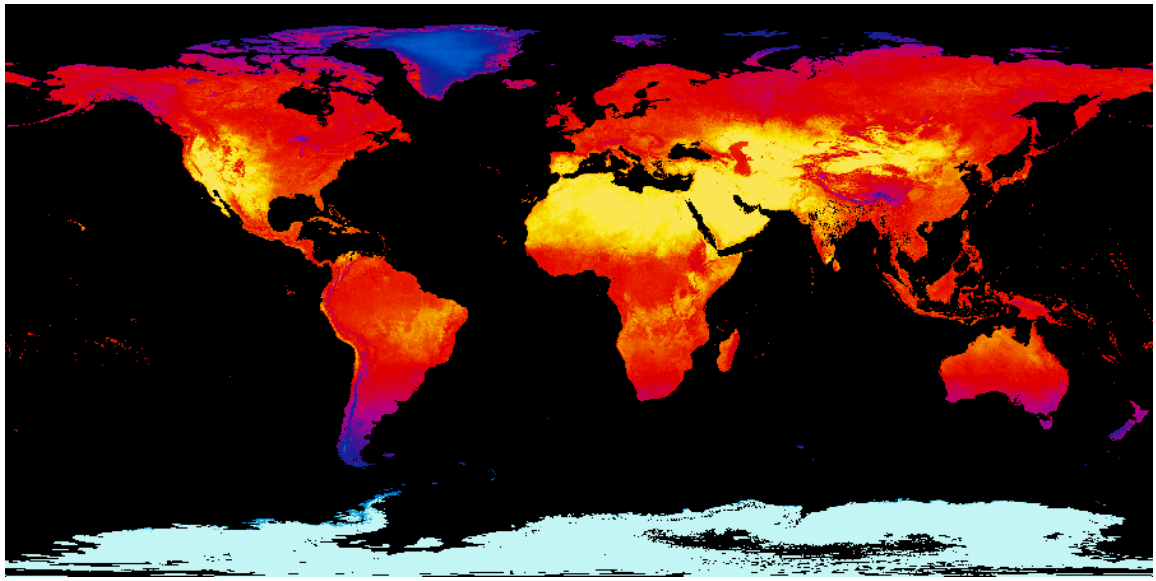
In [28]:
```
# we can also use operations in a chain like this:
my_dco.subset(var_name='$c', subset='ansi("2014-07")').set_format('PNG')
```

Out[28]:
```
<wdc.dco at 0x2825381cd90>
```

Now, let's execute `my_dco`. Because the format of the output will be `PNG`, let's use `Image()` from `IPython.display`.

In [29]:
```
output = my_dco.execute()
Image(output)
```

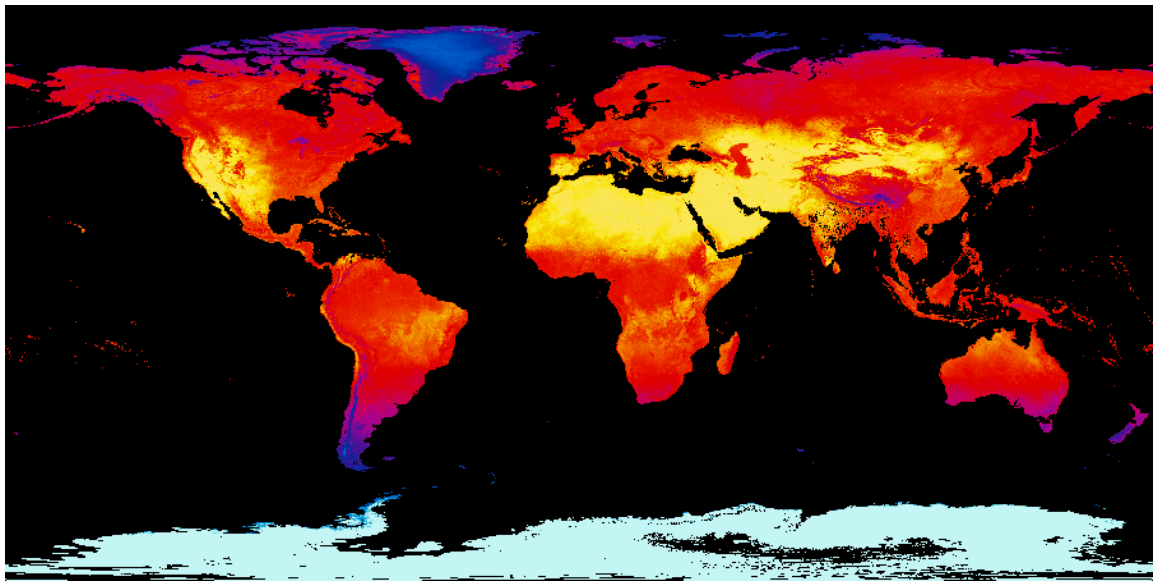
Out[29]:



After the execution, all attributes of our `dco()` instance will return to their defaults, except the `DBC` attribute, which stores the `dbc()` instance. So, if we want to use our instance again, we need to specify methods again. This time, let's swap the order of `subset()` and `set_format()`.

In [30]:
```
my_dco.initialize_var('$c in ( AvgTemperatureColorScaled )')
my_dco.set_format('PNG')
my_dco.subset(var_name='$c', subset='ansi("2014-07")')
output = my_dco.execute()
Image(output)
```

Out[30]:



As you can see, the order of methods, except `initialize_var()` and `execute`, doesn't matter.

The benefits of using `wdc` library instead of writing `WCPS` queries.

1. The order of methods almost doesn't matter.

It was shown already in the The order of the methods section that we can use methods almost in any order.

When we write a WCPS query, we must follow a specific structure and sometimes it might be confusing, especially if you are not familiar with the WCPS query's structure. Using `wdc` is simpler and you don't have to memorize much.

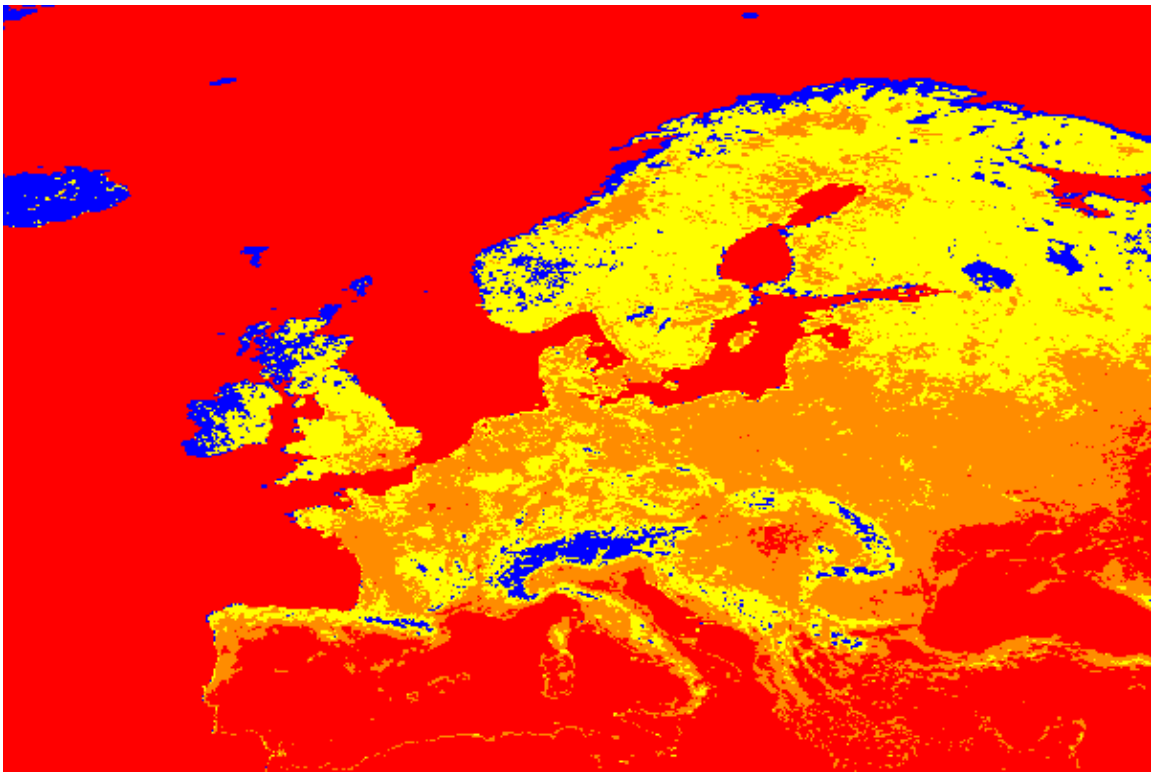
2. No need to specify a subset all the time.

Let's take a look at the following WCPS query:

```
for $c in ( AvgLandTemp )
return
encode(
    switch
            case $c[ansi("2014-07"), Lat(35:75), Long(-20:40)] = 99
999
                return {red: 255; green: 255; blue: 255}
            case 18 > $c[ansi("2014-07"), Lat(35:75), Long(-20:40)]
                return {red: 0; green: 0; blue: 255}
```

In [31]:
```
my_dco.initialize_var('$c in ( AvgLandTemp )')
my_dco.encode('''
    switch
            case $c = 99999
                return {red: 255; green: 255; blue: 255}
            case 18 > $c
                return {red: 0; green: 0; blue: 255}
            case 23 > $c
                return {red: 255; green: 255; blue: 0}
            case 30 > $c
                return {red: 255; green: 140; blue: 0}
            default return {red: 255; green: 0; blue: 0}
''')
my_dco.subset(var_name = '$c', subset = 'ansi("2014-07"), Lat(35:75), Long(
my_dco.set_format('PNG')
Image(my_dco.execute())
```

Out[31]:



Here subset is specified only **once** and the program replaces `'$c'` with `$c[ansi("2014-07"), Lat(35:75), Long(-20:40)]` . Which can save some time.

3. Byte strings are converted to the list of numbers automatically.

In general, when you try to use a simple `requests.post(server_url, data = {'query': wcps_query}, verify = False)` and then access `requests.content`, you get a byte string, but not the list of numbers. By using `wdc`, you get a list of numbers automatically in case the output is not expected to be an image.

Some more examples

3D -> 1D

In [34]:
```
my_dco.initialize_var('$c in (AvgLandTemp)')
my_dco.subset(var_name = '$c', subset = 'Lat(53.08), Long(8.80), ansi("2014
my_dco.set_format('CSV')
output = my_dco.execute()
print(output)
```

```
[2.834646, 4.488189, 11.10236, 20.19685, 21.02362, 21.29921, 25.98425, 24.
33071, 22.12598, 16.06299, 8.897637, 2.283465]
```

- 1. We initialized the variable `'$c'` and the coverage `'AvgLandTemp'`
- 2. We set a subset for the variable `'$c'` by using subset() function.
- 3. We chose a format for our output to be 'CSV'.
- 4. We executed and got a list of numbers.

Celcius to Kelvin

In [37]:
```
my_dco.initialize_var('$c in (AvgLandTemp)')
my_dco.subset(var_name = '$c', subset = 'Lat(53.08), Long(8.80), ansi("2014
my_dco.transform_data('$c + 273.15')
my_dco.set_format('CSV')
output = my_dco.execute()
print(output)
```

```
[275.9846457481384, 277.6381887435913, 284.2523626327514, 293.34684982299
8, 294.1736225128174, 294.4492134094238, 299.1342510223388, 297.4807094573
974, 295.2759841918945, 289.2129920959472, 282.0476373672485, 275.43346467
01813]
```

- 1. We initialized the variable `'$c'` and the coverage `'AvgLandTemp'`
- 2. We set a subset for the variable `'$c'` by using subset() function.
- 3. We transformed the variable by adding 273.15.
- 4. We chose a format for our output to be 'CSV'.
- 5. We executed and got a list of numbers.

Min

In [42]:
```
my_dco = dco(my_dbc)
my_dco.initialize_var('$c in (AvgLandTemp)')
my_dco.min()
my_dco.subset(var_name = '$c', subset = 'Lat(53.08), Long(8.80), ansi("2014
print(my_dco.execute())
```

```
[2.2834647]
```

- 1. We initialized the variable `'$c'` and the coverage `'AvgLandTemp'`
- 2. We indicated that we want to use an aggregation function to find minimum.
- 3. We set a subset for the variable `'$c'` by using subset() function.
- 4. We executed and got a list of numbers.

When is temp more than 15

In [40]:
```
my_dco = dco(my_dbc)
my_dco.initialize_var('$c in (AvgLandTemp)')
my_dco.subset(var_name = '$c', subset = 'Lat(53.08), Long(8.80), ansi("2014
my_dco.count('$c > 15')
my_dco.execute()
```

Out[40]:
```
[7.0]
```

- 1. We initialized the variable `'$c'` and the coverage `'AvgLandTemp'`
- 2. We set a subset for the variable `'$c'` by using subset() function.
- 3. We indicated that we want to count data points, which are greater than 15.
- 4. We executed and got a list of numbers.

FAQ

Why do I get a value error, when I use initialize_var()?

Is the format of your string correct? It must be in the format: `$var_name in ($coverage_name)` , where you must have no spaces in the beginning nor in the end.

Why do I get an error, when I use where()?

`where()` is applicable to scalar arguments only. Try to create a query and test it in DEMO_WCPS (https://standards.rasdaman.com/demo_wcps.html).

How many aggregation functions can I use?

You can use only 1 aggregation function at a time. The last aggregation function that you used is counted.

How many formats can I set?

You can set only 1 format. The last format you set is counted.