Experiment 8

[ ]:
```python
## Import libraries
import ee
import geemap
```

[ ]:
```python
ee.Authenticate()
ee.Initialize(project='ee-nandiinii')
```

```
<IPython.core.display.HTML object>
```

[ ]:
```python
## Create an interactive map
Map = geemap.Map()
Map
```

```
<IPython.core.display.HTML object>
```

```
Map(center=[0, 0], controls=(WidgetControl(options=['position',␣
 ↪'transparent_bg'], widget=SearchDataGUI(childr…
```

[ ]:
```python
## Add data to the map
point = ee.Geometry.Point([-122.4439, 37.7538])

image = (
    ee.ImageCollection("LANDSAT/LC08/C02/T1_L2")
    .filterBounds(point)
    .filterDate("2023-01-01", "2023-12-31")
    .sort("CLOUD_COVER")
    .first()
    .select(['SR_B1', 'SR_B2', 'SR_B3', 'SR_B4', 'SR_B5', 'SR_B6', 'SR_B7'])
)

vis_params = {"min": 0, "max": 3000, "bands": ["SR_B5", "SR_B4", "SR_B3"]}

Map.centerObject(point, 8)
Map.addLayer(image, vis_params, "Landsat-8")
```

```
<IPython.core.display.HTML object>
```

0.0.1 Make training dataset

There are several ways you can create a region for generating the training dataset.

- Define a geometry, such as `region = ee.Geometry.Rectangle([-122.6003, 37.4831, -121.8036, 37.8288])`
- Create a buffer zone around a point, such as `region = ee.Geometry.Point([-122.4439, 37.7538]).buffer(10000)`
- If you don't define a region, it will use the image footprint by default

[ ]:
```
# region = ee.Geometry.Rectangle([-122.6003, 37.4831, -121.8036, 37.8288]) OR
region = ee.Geometry.Point([-122.4439, 37.7538]).buffer(10000)
```

```
<IPython.core.display.HTML object>
```

[ ]:
```
# Make the training dataset.
training = image.sample(
    **{
        #'region': region,
        "scale": 30,
        "numPixels": 5000,
        "seed": 0,
        "geometries": True,  # Set this to False to ignore geometries
    }
)

Map.addLayer(training, {}, "training", False)
Map
```

```
<IPython.core.display.HTML object>
```

```
Map(bottom=12954.0, center=[37.87485339352928, -121.13452736427544],␣
↪controls=(WidgetControl(options=['positio…
```

0.0.2 Train the clusterer

[ ]:
```
# Instantiate the clusterer and train it.
n_clusters = 5
clusterer = ee.Clusterer.wekaKMeans(n_clusters).train(training)
```

```
<IPython.core.display.HTML object>
```

0.0.3 Classify the image

[ ]:
```
# Cluster the input using the trained clusterer.
result = image.cluster(clusterer)

# # Display the clusters with random colors.
Map.addLayer(result.randomVisualizer(), {}, "clusters")
Map
```

```
<IPython.core.display.HTML object>
```

```
Map(bottom=6487.0, center=[40.25437660372652, -119.37574198403479],␣
↪controls=(WidgetControl(options=['position…
```

0.0.4 Label the clusters

```
[ ]: legend_keys = ["One", "Two", "Three", "Four", "etc"]
     legend_colors = ["#8DD3C7", "#FFFFB3", "#BEBADA", "#FB8072", "#80B1D3"]

     # Reclassify the map
     result = result.remap([0, 1, 2, 3, 4], [1, 2, 3, 4, 5])

     Map.addLayer(
         result, {"min": 1, "max": 5, "palette": legend_colors}, "Labelled clusters"
     )
     Map.add_legend(
         legend_keys=legend_keys, legend_colors=legend_colors, position="bottomright"
     )
     Map
```

```
<IPython.core.display.HTML object>
```

```
Map(bottom=25766.0, center=[37.18657859524883, -121.32461215611129],␣
↪controls=(WidgetControl(options=['positio…
```

0.0.5 Export the result to your computer

```
[ ]: import os

     # Define a directory that exists
     out_dir = os.path.join(os.path.expanduser("~"), "Downloads")
     # If the directory doesn't exist, create it
     os.makedirs(out_dir, exist_ok=True)
     out_file = os.path.join(out_dir, "cluster1.tif")

     geemap.ee_export_image(result, filename=out_file, scale=90)
```

```
<IPython.core.display.HTML object>
```

```
Generating URL …
Downloading data from https://earthengine.googleapis.com/v1/projects/ee-nandiini
i/thumbnails/fec5373a7dcafbef14ee85759ec79ee4-
f910dc1d69a6ec7c0b54a1a9fa95912f:getPixels
Please wait …
Data downloaded to /root/Downloads/cluster1.tif
```

1 Here a link will be generated, click this link to download cluster1.zip folder

[ ]:
```
pip install rasterio
```

```
<IPython.core.display.HTML object>
```

```
Collecting rasterio
  Downloading rasterio-1.4.3-cp311-cp311-
manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (9.1 kB)
Collecting affine (from rasterio)
  Downloading affine-2.4.0-py3-none-any.whl.metadata (4.0 kB)
Requirement already satisfied: attrs in /usr/local/lib/python3.11/dist-packages
(from rasterio) (25.3.0)
Requirement already satisfied: certifi in /usr/local/lib/python3.11/dist-
packages (from rasterio) (2025.1.31)
Requirement already satisfied: click>=4.0 in /usr/local/lib/python3.11/dist-
packages (from rasterio) (8.1.8)
Collecting cligj>=0.5 (from rasterio)
  Downloading cligj-0.7.2-py3-none-any.whl.metadata (5.0 kB)
Requirement already satisfied: numpy>=1.24 in /usr/local/lib/python3.11/dist-
packages (from rasterio) (2.0.2)
Collecting click-plugins (from rasterio)
  Downloading click_plugins-1.1.1-py2.py3-none-any.whl.metadata (6.4 kB)
Requirement already satisfied: pyparsing in /usr/local/lib/python3.11/dist-
packages (from rasterio) (3.2.3)
Downloading
rasterio-1.4.3-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (22.2
MB)
                         22.2/22.2 MB
101.8 MB/s eta 0:00:00
Downloading cligj-0.7.2-py3-none-any.whl (7.1 kB)
Downloading affine-2.4.0-py3-none-any.whl (15 kB)
Downloading click_plugins-1.1.1-py2.py3-none-any.whl (7.5 kB)
Installing collected packages: cligj, click-plugins, affine, rasterio
Successfully installed affine-2.4.0 click-plugins-1.1.1 cligj-0.7.2
rasterio-1.4.3
```

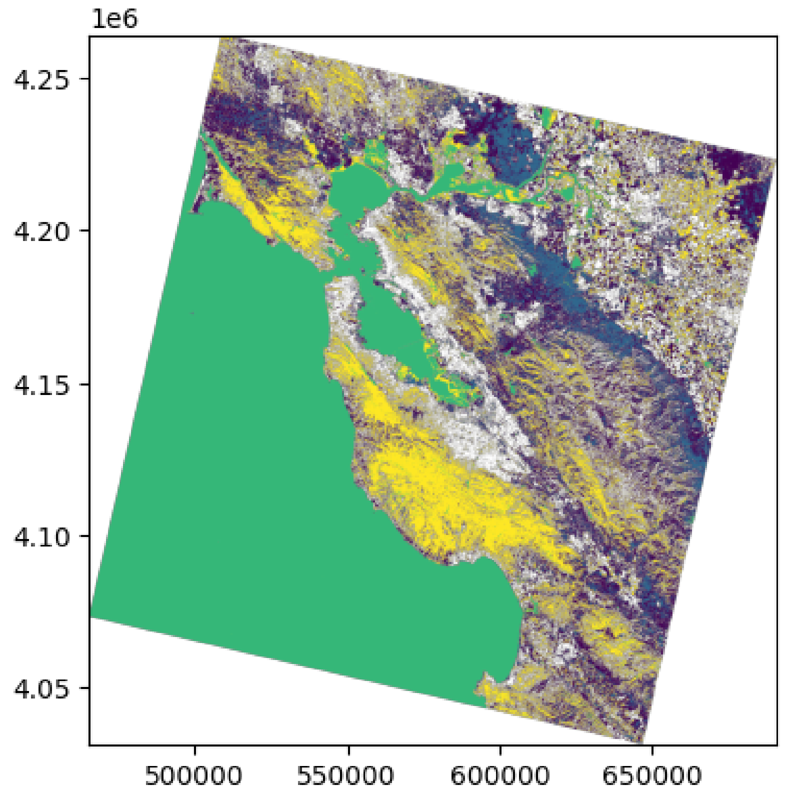
[ ]:
```python
print("Cluster1")
import rasterio as rio
clusters1 = rio.open("/content/cluster1.tif")

from rasterio.plot import show

show(clusters1)
```

```
<IPython.core.display.HTML object>
```

```
Cluster1
```



```
[ ]: <Axes: >
```

```
[3]: !jupyter nbconvert --to pdf /content/221_Expt_8.ipynb
```

```
<IPython.core.display.HTML object>

[NbConvertApp] WARNING | pattern '/content/221_Expt_8.ipynb' matched no files
This application is used to convert notebook files (*.ipynb)
        to various other formats.

        WARNING: THE COMMANDLINE INTERFACE MAY CHANGE IN FUTURE RELEASES.

Options
=======
The options below are convenience aliases to configurable class-options,
as listed in the "Equivalent to" description-line of the aliases.
To see all configurable class-options for some <cmd>, use:
    <cmd> --help-all

--debug
```

```
    set log level to logging.DEBUG (maximize logging output)
    Equivalent to: [--Application.log_level=10]
--show-config
    Show the application's configuration (human-readable format)
    Equivalent to: [--Application.show_config=True]
--show-config-json
    Show the application's configuration (json format)
    Equivalent to: [--Application.show_config_json=True]
--generate-config
    generate default config file
    Equivalent to: [--JupyterApp.generate_config=True]
-y
    Answer yes to any questions instead of prompting.
    Equivalent to: [--JupyterApp.answer_yes=True]
--execute
    Execute the notebook prior to export.
    Equivalent to: [--ExecutePreprocessor.enabled=True]
--allow-errors
    Continue notebook execution even if one of the cells throws an error and
include the error message in the cell output (the default behaviour is to abort
conversion). This flag is only relevant if '--execute' was specified, too.
    Equivalent to: [--ExecutePreprocessor.allow_errors=True]
--stdin
    read a single notebook file from stdin. Write the resulting notebook with
default basename 'notebook.*'
    Equivalent to: [--NbConvertApp.from_stdin=True]
--stdout
    Write notebook output to stdout instead of files.
    Equivalent to: [--NbConvertApp.writer_class=StdoutWriter]
--inplace
    Run nbconvert in place, overwriting the existing notebook (only
            relevant when converting to notebook format)
    Equivalent to: [--NbConvertApp.use_output_suffix=False
--NbConvertApp.export_format=notebook --FilesWriter.build_directory=]
--clear-output
    Clear output of current file and save in place,
            overwriting the existing notebook.
    Equivalent to: [--NbConvertApp.use_output_suffix=False
--NbConvertApp.export_format=notebook --FilesWriter.build_directory=
--ClearOutputPreprocessor.enabled=True]
--coalesce-streams
    Coalesce consecutive stdout and stderr outputs into one stream (within each
cell).
    Equivalent to: [--NbConvertApp.use_output_suffix=False
--NbConvertApp.export_format=notebook --FilesWriter.build_directory=
--CoalesceStreamsPreprocessor.enabled=True]
--no-prompt
    Exclude input and output prompts from converted document.
    Equivalent to: [--TemplateExporter.exclude_input_prompt=True
--TemplateExporter.exclude_output_prompt=True]
--no-input
    Exclude input cells and output prompts from converted document.
            This mode is ideal for generating code-free reports.
    Equivalent to: [--TemplateExporter.exclude_output_prompt=True
--TemplateExporter.exclude_input=True
--TemplateExporter.exclude_input_prompt=True]
--allow-chromium-download
    Whether to allow downloading chromium if no suitable version is found on the
system.
    Equivalent to: [--WebPDFExporter.allow_chromium_download=True]
--disable-chromium-sandbox
    Disable chromium security sandbox when converting to PDF..
    Equivalent to: [--WebPDFExporter.disable_sandbox=True]
--show-input
    Shows code input. This flag is only useful for dejavu users.
    Equivalent to: [--TemplateExporter.exclude_input=False]
--embed-images
    Embed the images as base64 dataurls in the output. This flag is only useful
for the HTML/WebPDF/Slides exports.
    Equivalent to: [--HTMLExporter.embed_images=True]
--sanitize-html
    Whether the HTML in Markdown cells and cell outputs should be sanitized..
    Equivalent to: [--HTMLExporter.sanitize_html=True]
--log-level=<Enum>
    Set the log level by value or name.
    Choices: any of [0, 10, 20, 30, 40, 50, 'DEBUG', 'INFO', 'WARN', 'ERROR',
'CRITICAL']
    Default: 30
    Equivalent to: [--Application.log_level]
--config=<Unicode>
    Full path of a config file.
    Default: ''
    Equivalent to: [--JupyterApp.config_file]
--to=<Unicode>
    The export format to be used, either one of the built-in formats
            ['asciidoc', 'custom', 'html', 'latex', 'markdown', 'notebook',
'pdf', 'python', 'qtpdf', 'qtpng', 'rst', 'script', 'slides', 'webpdf']
            or a dotted object name that represents the import path for an
            ``Exporter`` class
    Default: ''
    Equivalent to: [--NbConvertApp.export_format]
--template=<Unicode>
    Name of the template to use
    Default: ''
    Equivalent to: [--TemplateExporter.template_name]
--template-file=<Unicode>
```

```
    Name of the template file to use
    Default: None
    Equivalent to: [--TemplateExporter.template_file]
--theme=<Unicode>
    Template specific theme(e.g. the name of a JupyterLab CSS theme distributed
    as prebuilt extension for the lab template)
    Default: 'light'
    Equivalent to: [--HTMLExporter.theme]
--sanitize_html=<Bool>
    Whether the HTML in Markdown cells and cell outputs should be sanitized.This
    should be set to True by nbviewer or similar tools.
    Default: False
    Equivalent to: [--HTMLExporter.sanitize_html]
--writer=<DottedObjectName>
    Writer class used to write the
                                    results of the conversion
    Default: 'FilesWriter'
    Equivalent to: [--NbConvertApp.writer_class]
--post=<DottedOrNone>
    PostProcessor class used to write the
                                    results of the conversion
    Default: ''
    Equivalent to: [--NbConvertApp.postprocessor_class]
--output=<Unicode>
    Overwrite base name use for output files.
                Supports pattern replacements '{notebook_name}'.
    Default: '{notebook_name}'
    Equivalent to: [--NbConvertApp.output_base]
--output-dir=<Unicode>
    Directory to write output(s) to. Defaults
                                  to output to the directory of each notebook.
To recover
                                  previous default behaviour (outputting to the
current
                                  working directory) use . as the flag value.
    Default: ''
    Equivalent to: [--FilesWriter.build_directory]
--reveal-prefix=<Unicode>
    The URL prefix for reveal.js (version 3.x).
            This defaults to the reveal CDN, but can be any url pointing to a
copy
            of reveal.js.
            For speaker notes to work, this must be a relative path to a local
            copy of reveal.js: e.g., "reveal.js".
            If a relative path is given, it must be a subdirectory of the
            current directory (from which the server is run).
            See the usage documentation
            (https://nbconvert.readthedocs.io/en/latest/usage.html#reveal-js-
html-slideshow)
            for more details.
    Default: ''
    Equivalent to: [--SlidesExporter.reveal_url_prefix]
--nbformat=<Enum>
    The nbformat version to write.
            Use this to downgrade notebooks.
    Choices: any of [1, 2, 3, 4]
    Default: 4
    Equivalent to: [--NotebookExporter.nbformat_version]

Examples
--------

    The simplest way to use nbconvert is

            > jupyter nbconvert mynotebook.ipynb --to html

            Options include ['asciidoc', 'custom', 'html', 'latex', 'markdown',
'notebook', 'pdf', 'python', 'qtpdf', 'qtpng', 'rst', 'script', 'slides',
'webpdf'].

            > jupyter nbconvert --to latex mynotebook.ipynb

            Both HTML and LaTeX support multiple output templates. LaTeX
includes
            'base', 'article' and 'report'.  HTML includes 'basic', 'lab' and
            'classic'. You can specify the flavor of the format used.

            > jupyter nbconvert --to html --template lab mynotebook.ipynb

            You can also pipe the output to stdout, rather than a file

            > jupyter nbconvert mynotebook.ipynb --stdout

            PDF is generated via latex

            > jupyter nbconvert mynotebook.ipynb --to pdf

            You can get (and serve) a Reveal.js-powered slideshow

            > jupyter nbconvert myslides.ipynb --to slides --post serve

            Multiple notebooks can be given at the command line in a couple of
            different ways:

            > jupyter nbconvert notebook*.ipynb
            > jupyter nbconvert notebook1.ipynb notebook2.ipynb
            or you can specify the notebooks list in a config file, containing::

                c.NbConvertApp.notebooks = ["my_notebook.ipynb"]

            > jupyter nbconvert --config mycfg.py

To see all available configurables, use `--help-all`.
```