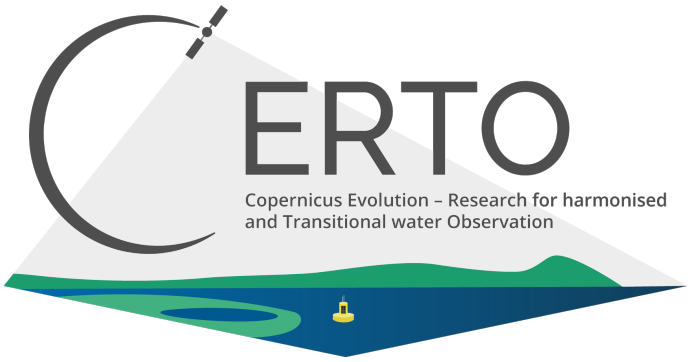


Horizon 2020 LC-SPACE-04-EO-2019-2020

Copernicus Evolution – Research for harmonised and Transitional-water Observation (CERTO)

Project Number: 870349

Deliverable No: D4.3		Work Package:	
Date:	29-APR-2022	Contract delivery due date	30-APR-2022
Title:	Classification Toolbox		
Lead Partner for Deliverable	PML Applications		
Author(s):	Angus Laurenson (PML), Liz Atwood (PML), Ben Calton (PMLA)		
Dissemination level (PU=public, RE=restricted, CO=confidential)			PU
Report Status (DR = Draft, FI = FINAL)			FI

Acknowledgements
This project has received funding from the European Union's Horizon 2020 research and innovation programme grant agreement N° 870349

Table of Contents

1 Executive Summary

Water quality is a key worldwide issue relevant to human food consumption and production, industry, nature, and recreation. The European Copernicus programme includes satellite sensors designed to observe water quality, and services to provide data and information to end-users in industry, policy, monitoring agencies, and science. However, water-quality data production is split across three services, Copernicus Marine, Copernicus Climate Change, and Copernicus Land, with different methods and approaches.

The CERTO project aims to address this lack of harmonisation by undertaking research and development necessary to produce harmonised water-quality data from each service and indicators to extend Copernicus to the large number of stakeholders operating in transitional waters. The main output of the project will be a prototype system that can be “plugged into” the existing Copernicus services, into the developing Data and Information Access Services (DIAS), or into popular open-source software used widely by the community (e.g. SNAP).

One element of the CERTO prototype is the classification toolbox, and this document describes how to access the toolbox as a stand-alone tool, and it provides a working example using Sentinel-3 OLCI data for the Curonian Lagoon, one of the CERTO case study areas.

2 Classification Toolbox code

The core of this deliverable is the software that is the classification toolbox. This is hosted in a publicly accessible git repository which can be found at https://github.com/CERTO-project/D4.3_Classification_toolbox

Within this repository is the code required to produce an optical water type class set for whichever region of interest the user is concerned with for both Sentinel-3 OLCI and Sentinel-2 MSI. Whilst the main focus of the toolbox in a CERTO project context will be on the case study regions, the toolbox itself can be used for any geographic region, providing a suitable training dataset is available. An installation guide is provided to enable users to begin using the software, and a fully worked example is provided as a demonstration of the steps required to produce a set of water classes.

3 Worked Example

Within the code repository there is a Jupyter Notebook which offers an interactive step-by-step walkthrough of how to use the toolbox. This includes a view of how to build a training dataset and run the classification toolbox to produce a set of optical water type classes. The example included allows the user to change the code as they move through the notebook, adapting it to their own needs, and for the purposes of this deliverable we have included a static PDF view of this notebook along with the output the user would expect to see.

Classification Toolbox - Sentinel-3 Example

The classification toolbox provides three main methods; production of a training dataset, generation of a model which defines the optimum number of optical water types and 'fuzziness', and the application of this model to a complete dataset.

Fuzzy clustering optical water type (OWT) models are fitted to a subset of the total data timeseries in a phase called training. In a second step, the fitted model is then applied to new data. Table of cluster centers and plots of model selection peformance are provided, together with membership histograms (ideally bimodal), example geographic membership distrubtion on a single day, membership occurrence rates over time and dominant OWT maps over whole time series as well as monthly.

1. Creation of Training Dataset

1.1 Libraries & variables setup

In [75]:

```
from datetime import datetime
import glob
import holoviews as hv
import hvplot.pandas
import hvplot.xarray
import matplotlib.pyplot as plt
import matplotlib.ticker as ticker
import numpy as np
import os
import pandas as pd
import random
import rasterio
from statsmodels.graphics.gofplots import qqplot
import tempfile
import warnings
import xarray as xr
```

In [76]:

```
# run_date is added to exported filenames for book keeping
run_date = datetime.strftime(datetime.now(), "%Y%m%d")

# The ROI to process - one of:
# "Curonian_Lagoon","Danube_Delta","German_Bight","Tagus_Estuary","Plymouth_Sound",'
region = "Curonian_Lagoon"

# The sampling method to use - one of:
# 'regular','random','stratified_random','weighted_random'
sampling_method = 'weighted_random'

features = ["Rw400_rep","Rw412_rep","Rw443_rep","Rw490_rep","Rw510_rep","Rw560_rep",
            "Rw674_rep","Rw681_rep","Rw709_rep","Rw754_rep","Rw779_rep","Rw865_rep",
# list used in first reporting 400, 412, 443, 490, 510, 560, 620, 665, 674, 681 and
# removed atm corr bands with predominantly negative Rw values
shore_dist = 20000
# in meters
```

This second set of variables match the regions full name with a shortened version of the name (`dirdict`); `winterdict` sets the months the exclude when creating the training data, mainly due to poor light conditions and/or poor coverage due to weather conditions. `tgooddict` specifies the index of the good example images to use

In [77]:

```
dirdict = {"Plymouth_Sound":"tamar", "Curonian_Lagoon":"curonian","Danube_Delta":"da
           "Venice_Lagoon":"venice","Tagus_Estuary":"tagus", "German_Bight":"elbe"}
site = dirdict[region]

# defined using NOAA solcalc (https://gml.noaa.gov/grad/solcalc/) and max incident l
winterdict = {"Plymouth_Sound":[2,11], "Curonian_Lagoon":[2,10],"Danube_Delta":[1,11
           "Venice_Lagoon":[0,12],"Tagus_Estuary":[0,12], "German_Bight":[1,11]}
# 'tamar':[2,11],'curonian':[2,10],'danube':[1,11],'venice':[0,12],'tagus':[0,12],'e
winter_months = winterdict[region]

# dates with very low cloud cover, good example image
tgooddict = {"Plymouth_Sound":1273, "Curonian_Lagoon":1398,"Danube_Delta":1263,
           "Venice_Lagoon":1210,"Tagus_Estuary":1092, "German_Bight":1600}
time_good = tgooddict[region]
```

2. Select subsampling routine, check coverage, and create training data

We used OLCI 300m data (sensors A and B) from Apr. 2016 to Mar. 2021 to build the training dataset. Data were atmospherically corrected with POLYMER, following the CALIMNOS lakes processing chain. Specifics include:

- Reflectances in Rw as opposed to Rrs
- Bands used for clustering: 400, 412, 443, 490, 510, 560, 620, 665, 674, 681, 709, 754, 779, 865, 885 nm
- Winter month exclusion from training dataset (based on incident light level <30° calculated using NOAA solcalc): October-February

In [78]:

```
data_path = f'/data/datasets/Projects/CERTO/no-backup/olci_data/L3_com/v1.4.0/1D/300
dflist = glob.glob(data_path)
dflist.sort()
ds = xr.open_dataset(dflist[time_good])

ds = ds[features]
ds
```

Out[78]:

xarray.Dataset

▸ Dimensions: (**time**: 1, **lat**: 377, **lon**: 345)

▾ Coordinates:

lat	(lat)	float64	55.73 55.73 55.73 ... 54.89 5…
lon	(lon)	float64	20.52 20.52 20.52 ... 21.28 2…
time	(time)	datetime64[ns]	2020-08-15T08:43:07.500000

▾ Data variables:

Rw400_rep	(time, lat, lon)	float32	...
Rw412_rep	(time, lat, lon)	float32	...
Rw443_rep	(time, lat, lon)	float32	...
Rw490_rep	(time, lat, lon)	float32	...
Rw510_rep	(time, lat, lon)	float32	...
Rw560_rep	(time, lat, lon)	float32	...
Rw620_rep	(time, lat, lon)	float32	...
Rw665_rep	(time, lat, lon)	float32	...
Rw674_rep	(time, lat, lon)	float32	...
Rw681_rep	(time, lat, lon)	float32	...
Rw709_rep	(time, lat, lon)	float32	...
Rw754_rep	(time, lat, lon)	float32	...
Rw779_rep	(time, lat, lon)	float32	...
Rw865_rep	(time, lat, lon)	float32	...
Rw885_rep	(time, lat, lon)	float32	...

▸ Attributes: (27)

1.3 Create sampling points

In this example we use `weighted_random` where sampling locations are random but with greater weighting towards locations nearer the coast. This cell produces a dataset, `ds_sum` , that records a supposed land mask by recording only pixels where there are no data values for any given pixel within the entire time series

In [79]:

```
ds_init = xr.open_dataset(dflist[0])
ds_sum = ds_init[[x for x in ds_init.data_vars if x.startswith("Rw4")][0]]
ds_sum = ds_sum.fillna(0)
ds_sum = ds_sum*0

for i in range(0,len(dflist)):
    temp = xr.open_dataset(dflist[i])[[x for x in ds_init.data_vars if x.startswith(
    ds_sum[0] += np.nan_to_num(temp[0])
#     print(pd.to_datetime(temp.time.item()))
    if i % 50 ==0:
        print(f'{100*np.round(i/len(dflist), 3)} % done (till date: {pd.to_datetime(
    temp.close()

ds_init.close()
```

```
0.0 % done (till date: 2016-04-26 09:36:33.750000128)
3.1 % done (till date: 2016-06-21 08:45:56.249999872)
6.2 % done (till date: 2016-08-17 09:08:26.249999872)
9.4 % done (till date: 2016-10-13 09:30:56.249999872)
12.5 % done (till date: 2016-12-08 08:37:30)
15.6 % done (till date: 2017-02-03 09:00:00)
18.7 % done (till date: 2017-03-31 09:47:48.750000128)
21.8 % done (till date: 2017-05-26 08:57:11.249999872)
24.9 % done (till date: 2017-07-21 09:45:00)
28.1 % done (till date: 2017-09-15 08:54:22.500000)
31.2 % done (till date: 2017-11-11 09:16:52.500000)
34.300000000000004 % done (till date: 2018-01-07 09:36:33.750000128)
37.4 % done (till date: 2018-03-04 08:45:56.249999872)
40.5 % done (till date: 2018-04-30 09:08:26.249999872)
43.6 % done (till date: 2018-06-26 09:30:56.249999872)
46.800000000000004 % done (till date: 2018-08-23 09:25:18.750000128)
49.9 % done (till date: 2018-10-18 08:34:41.249999872)
53.0 % done (till date: 2018-12-13 09:22:30)
56.10000000000001 % done (till date: 2019-02-07 08:31:52.500000)
59.199999999999996 % done (till date: 2019-04-04 09:19:41.249999872)
62.3 % done (till date: 2019-05-30 08:26:15)
65.5 % done (till date: 2019-07-25 09:16:52.500000)
68.60000000000001 % done (till date: 2019-09-19 08:23:26.249999872)
71.7 % done (till date: 2019-11-15 08:45:56.249999872)
74.8 % done (till date: 2020-01-12 08:43:07.500000)
77.9 % done (till date: 2020-03-09 09:05:37.500000)
81.0 % done (till date: 2020-05-05 09:28:07.500000)
84.2 % done (till date: 2020-06-28 08:48:45)
87.3 % done (till date: 2020-08-17 08:51:33.750000128)
90.4 % done (till date: 2020-10-06 08:54:22.500000)
93.5 % done (till date: 2020-11-25 09:00:00)
96.6 % done (till date: 2021-01-14 09:02:48.750000128)
99.8 % done (till date: 2021-03-05 09:05:37.500000)
```

In [80]:

```
# remove zero buffer as needed, watch for edge error in distance plots below

if site == 'curonian':
    ds_sum = xr.where(ds_sum.lat == float(ds_sum.lat.values.max()), ds_sum+1., ds_su
    ds_sum = xr.where(ds_sum.lon == float(ds_sum.lon.values.min()), ds_sum+1., ds_su
    ds_sum = ds_sum.T
    ds_sum = ds_sum.rename({'lon':'x', 'lat':'y'})

if site == 'tagus':
    ds_sum = ds_sum.isel(lon = slice(1,-1))
    ds_sum = ds_sum.isel(lat = slice(1,-1))

if site == 'tamar':
    ds_sum = ds_sum.isel(lon = slice(0,-1))
```

The resulting dataset is saved to a temp file location so that it can be accessed by gdal

In [81]:

```
temp_dir = tempfile.mkdtemp(prefix="temp_", dir=os.getcwd())
ds_sum.rio.write_crs("epsg:4326", inplace=True)
output_tiff = temp_dir + f'/{site}_summed.tif'
ds_sum.rio.to_raster(output_tiff)
```

Then saved the proximity file:

In [82]:

```
output_proximity = temp_dir + f'/{site}_proximity.tif'
!gdal_proximity.py -distunits PIXEL -values 0 -use_input_nodata YES -ot UInt16 $outp
```

```
0...10...20...30...40...50...60...70...80...90...100 - done.
```

Transform distances into percentages for weighting

In [83]:

```
ds_prox = xr.open_rasterio(output_proximity)
d = ds_prox.where(ds_prox != 0)
d = d.where(d<66) # (66*300m is 20km)
# scale to percent and invert distances
d = -d/d.max()+1
# set "offshore" weights
d = d.where(d>0.1)
d = d.fillna(0.1)
# mask land values (nan in ds_prox), this is purely for visualization
mask = d.where(np.isnan(ds_prox))
mask = mask.fillna(2)
d = d.where(mask == 2)
```

```
/tmp/ipykernel_117564/1056582937.py:1: DeprecationWarning: open_raster
io is Deprecated in favor of rioxarray. For information about transiti
oning, see: https://corteva.github.io/rioxarray/stable/getting_starte
d/getting_started.html (https://corteva.github.io/rioxarray/stable/get
ting_started/getting_started.html)
  ds_prox = xr.open_rasterio(output_proximity)
```

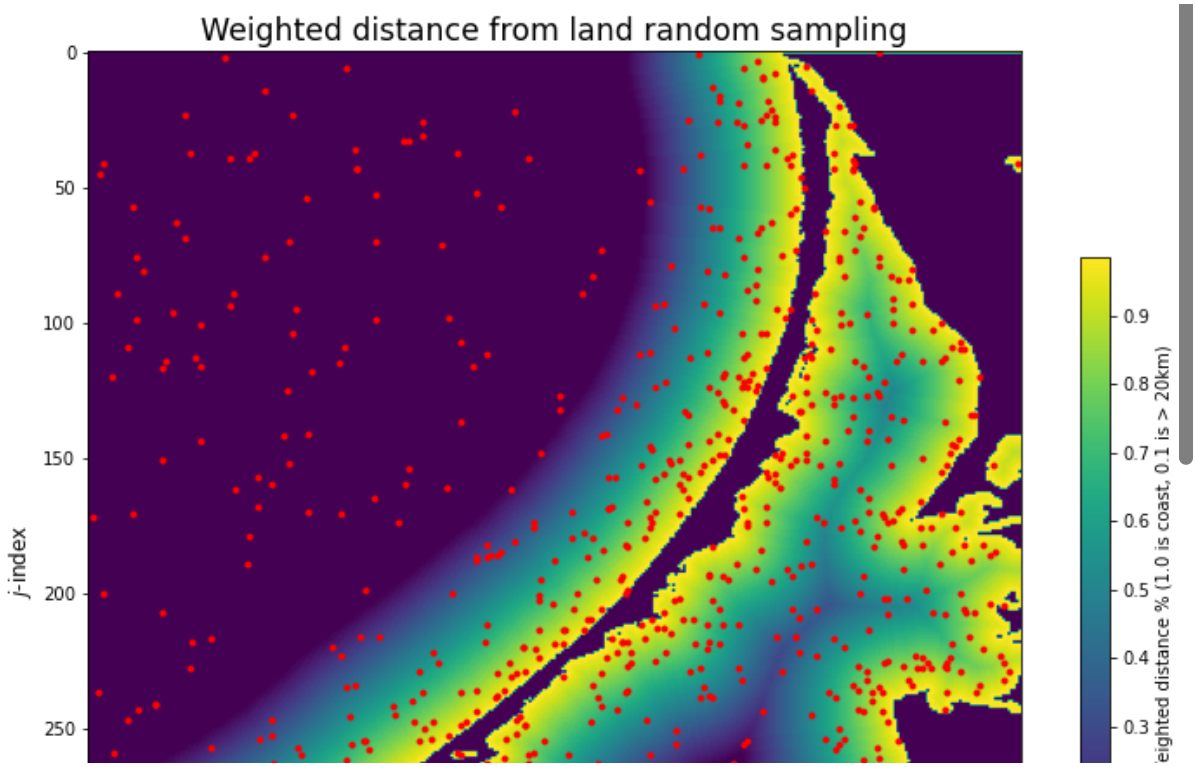
Produce a plot to demonstrate the distribution of sampling points for a single day

In [84]:

```
distances = np.clip(300.*ds_prox, 0, shore_dist) #300m per pixel and we want to have

rev_distances = ((shore_dist-(1*distances))/shore_dist)
capped_rev_distances = np.clip(rev_distances, 0.1, 0.99)

capped_masked_rev_distances_weightings = np.where(distances==0, distances, capped_re
capped_masked_rev_distances_weightings = np.nan_to_num(capped_masked_rev_distances_w
distances = distances.fillna(0)

flat_indices = np.arange(capped_masked_rev_distances_weightings.squeeze().size)
flat_weightings = capped_masked_rev_distances_weightings.squeeze().flat
test = np.unravel_index(
    random.choices(
        flat_indices,
        k=1000,
        weights=flat_weightings
    ),
    capped_masked_rev_distances_weightings.squeeze().shape
)

# indices = np.indices(capped_masked_rev_distances_weightings.squeeze().shape)
sample_x = test[1]
sample_y = test[0]

plt.figure(figsize=(10,10))
plt.imshow(d.squeeze())
plt.colorbar(shrink=0.5, label="Weighted distance % (1.0 is coast, 0.1 is > 20km)")
plt.tight_layout()
plt.title("Weighted distance from land random sampling", size='xx-large')
plt.xlabel("$\it{i}$-index", size='large'); plt.ylabel("$\it{j}$-index", size='large
plt.scatter(sample_x, sample_y, c='r', marker=".")
```

Out[84]:

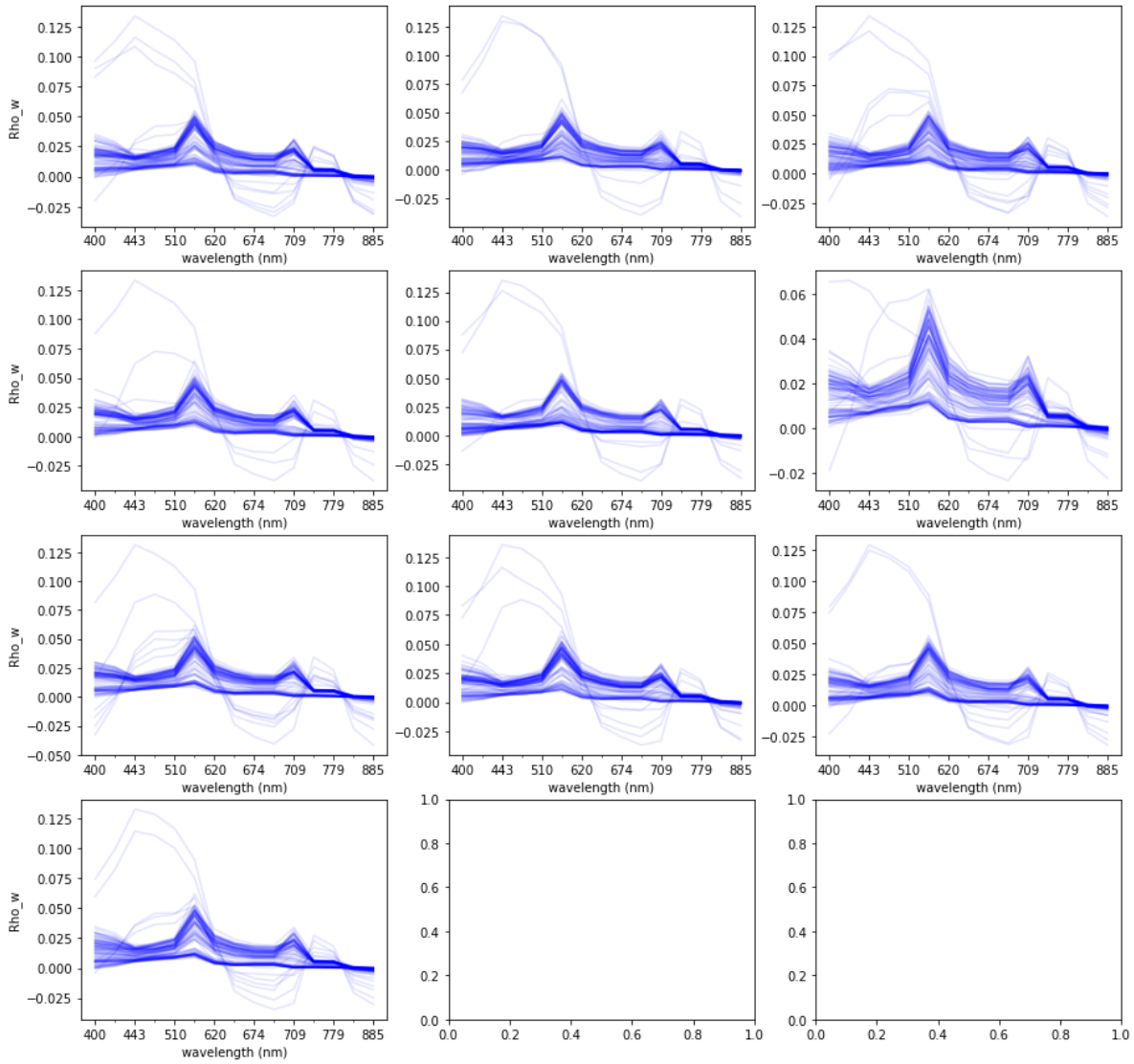
```
<matplotlib.collections.PathCollection at 0x7f218381dc70>
```



For the first ten points identified in the plot above, produce a plot of the spectra

In [85]:

```
dd2 = ({key:ds[key].values[0,sample_y,sample_x] for key in ds.data_vars})

fig, axes_set=plt.subplots(4,3, figsize=(15,15))
for i in range(0,10):
    subset=pd.DataFrame.from_dict(dd2).values.T[:,i*100:(i*100)+100]
    axes_set.flatten()[i].plot([x[2:5] for x in features], subset, color='blue', alp
    axes_set.flatten()[i].xaxis.set_major_locator(ticker.MultipleLocator(2))
    axes_set.flatten()[i].xaxis.set_minor_locator(plt.MultipleLocator(1))
    axes_set.flatten()[i].set_xlabel("wavelength (nm)")
    if i%3 == 0:
        axes_set.flatten()[i].set_ylabel("Rho_w")
```



1.4 Extract sampling point reflectance values

Provided the test was successful, move on to pull out sampling points for the entire time series.

WARNING this will take a long time and requires access to the full time series for the selected region

In [86]:

```
def subsample_OLCI_weightDist(region, months, mask, weights, indices, n):
    """Creates a subsample dataframe using weight file (e.g. distance from land)"""

    # find and sort file list
    flist = glob.glob(f"/data/datasets/Projects/CERTO/no-backup/olci_data/L3_com/v1.
    flist.sort()

    # extract dates to excluded from filenames
    dtmvec = pd.to_datetime([dstr.split("_")[5][:-4] for dstr in flist])
    tmask_season = np.full(len(flist), False)
    tmask_season[(dtmvec.month > months[0]) & (dtmvec.month < months[1])] = True
    tmask = tmask_season

    # Create subsample data for each date, update to dflist
    dflist = []
    for i, fn in enumerate(np.array(flist)[tmask]):
        ds = open_OLCI(fn, region=region, mask=mask)
        test = np.unravel_index(
            random.choices(indices, k=n, weights=weights.squeeze().flat),
            weights.squeeze().shape,
        )
        sample_i = test[0]
        sample_j = (
            weights.squeeze().shape[0] - test[0] - 1
        )  # needed due to indexes issues, j-axis is flipped without this
        dd = {key: ds[key].values[0, sample_j, sample_i] for key in ds.data_vars}
        df = pd.DataFrame(dd)
        sample_j = test[0]  # returned to normal
        df["ipos"] = sample_i.flat
        df["jpos"] = sample_j.flat
        df["date"] = pd.to_datetime(ds.time.item())
        df = df.dropna()
        ds.close()
        dflist.append(df)
        if i % 50 == 0:
            print(
                f"{100*np.round(i/len(tmask), 3)} % done (till date: {pd.to_datetime
            )
    dflist_n = pd.concat(dflist, axis=0)
    dflist_n = dflist_n.iloc[random.sample(range(dflist_n.shape[0]), k=n), :]
    return dflist_n


# samples_full = subsample_OLCI_weightDist(
#     region=region,
#     months=winter_months,
#     weights=capped_masked_rev_distances_weightings,
#     indices=flat_indices,
#     n=100000
# )
```

Instead, for the purposes of demonstration, we can load a pre-prepared example:

In [87]:

```
samples_full = pd.read_csv(f'./curonian_S3_20220428_samplesFull.csv', sep=',')
```

Format the data rady for clustering:

In [88]:

```
samples = samples_full.loc[:,features]
# samples = samples_full.iloc[:,:-3]

col_names = {}
for n in range(0, len(samples.columns)):
    col_names[str(samples.columns[n])]=str(samples.columns[n].split("_")[0].split("F
samples = samples.rename(columns=col_names)

samples
```

Out[88]:

	400	412	443	490	510	560	620	665	67
0	0.023530	0.019728	0.012376	0.016155	0.022873	0.051459	0.034511	0.027195	0.02325
1	0.006053	0.007931	0.010238	0.011270	0.012659	0.018133	0.005843	0.002309	0.00206
2	0.004672	0.005603	0.006207	0.008114	0.009371	0.011813	0.004722	0.003105	0.00330
3	0.007548	0.009193	0.007935	0.009069	0.009479	0.009612	0.002398	0.001188	0.00108
4	0.001264	0.003000	0.005062	0.006881	0.007425	0.009551	0.003916	0.002481	0.00325
...	...	...	...	...	...	...	...	...	..
99995	0.003940	0.006028	0.007448	0.009441	0.011318	0.014464	0.006071	0.003250	0.00400
99996	0.000854	0.001821	0.003437	0.004304	0.005114	0.006025	0.002184	0.002195	0.00229
99997	0.005037	0.005746	0.005939	0.008130	0.009653	0.017439	0.013926	0.008927	0.00840
99998	0.018886	0.018937	0.012623	0.010667	0.011442	0.029726	0.009995	0.005361	0.00345
99999	0.020202	0.016881	0.012270	0.013400	0.017730	0.036885	0.021047	0.016419	0.01377

100000 rows × 15 columns

At this point we have 10,000 random points, weighted by proximity to the coast; for each point (row), a reflectance value for each wavelength (column) is defined.

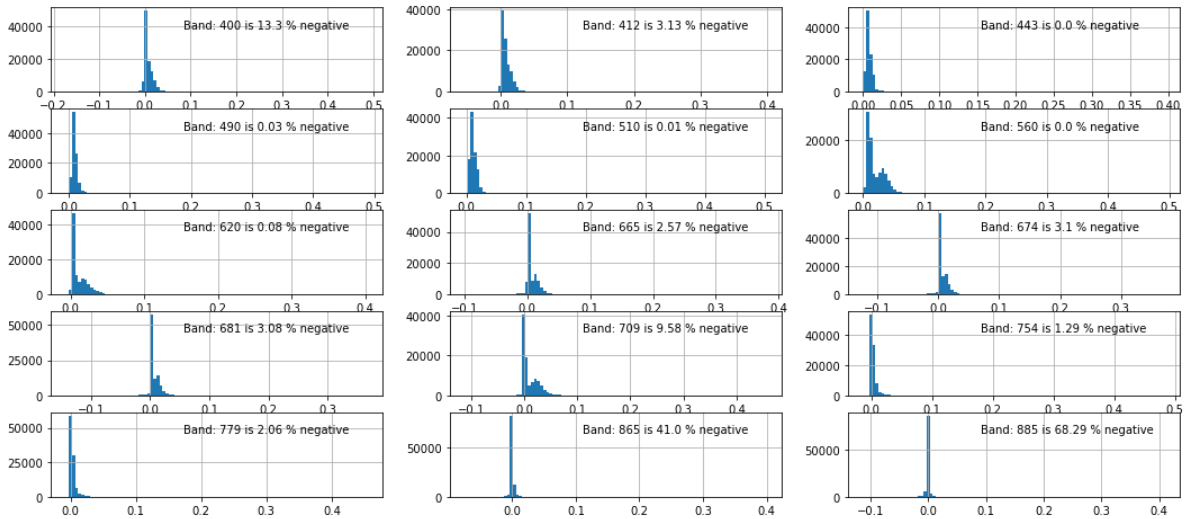
Some of the training data contains negative reflectances for particular bands, arising from errors in the atmospheric correction step. The decision has been made to retain as much of this information as possible in the cluster formation step so as to allow for exploration of clustering as a tool to identify such problematic pixels. A log transformation was chosen to reduce magnitude differences in similarly shaped spectra (plus helps fulfil the log-normal distribution expectation for reflectance data). However, log of a negative number is not usable for our purposes, thus a small additive shift was implemented prior to log transformation, chosen to balance reducing data loss with keeping the shift as small as possible. Whilst we could simply remove the negative values, the effect would be a reduction in the size of the date. Instead, we shift the reflectance value by adding `0.015`.

In this next step, we produce two series of plots; the first is a series of histograms showing the distribution of values along with a percentage of negative values at each wavelength. The second series shows the same after the shift of `0.015` has been applied

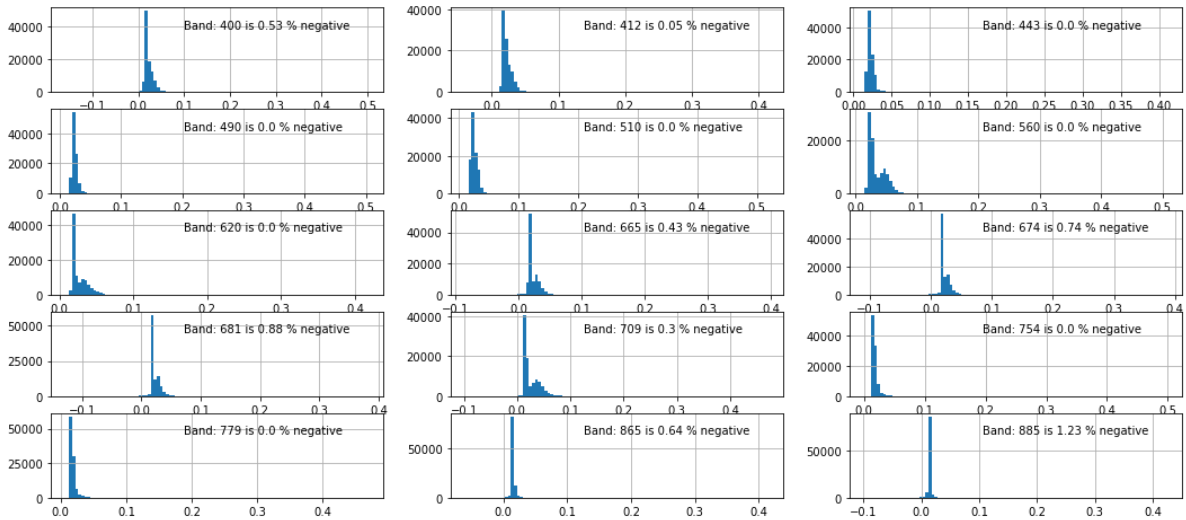
In [89]:

```
shift = 0.015

fig, axes=plt.subplots(int(np.ceil(len(samples.keys())/3)),3, figsize=(18,8))
for i,band in enumerate(samples.keys()):
    ax=axes.flatten()[i]
    hist=samples[band].hist(bins=100, ax=ax)
    plt.text(0.4, 0.75,  f"Band: {band} is {round(100*((samples[band]<0.0).sum())/le
fig.suptitle('Untransformed training data', fontsize=16, ha='left', x=fig.subplotpar

fig, axes=plt.subplots(int(np.ceil(len(samples.keys())/3)),3, figsize=(18,8))
for i,band in enumerate(samples.keys()):
    ax=axes.flatten()[i]
    hist=(samples[band]+shift).hist(bins=100, ax=ax)
    plt.text(0.4, 0.75,  f"Band: {band} is {round(100*((samples[band]+shift<0.0).sum
fig.suptitle(f'Shifted training data, (x + {shift})', fontsize=16, ha='left', x=fig.

plt.show()
```

Untransformed training data



Shifted training data, (x + 0.015)



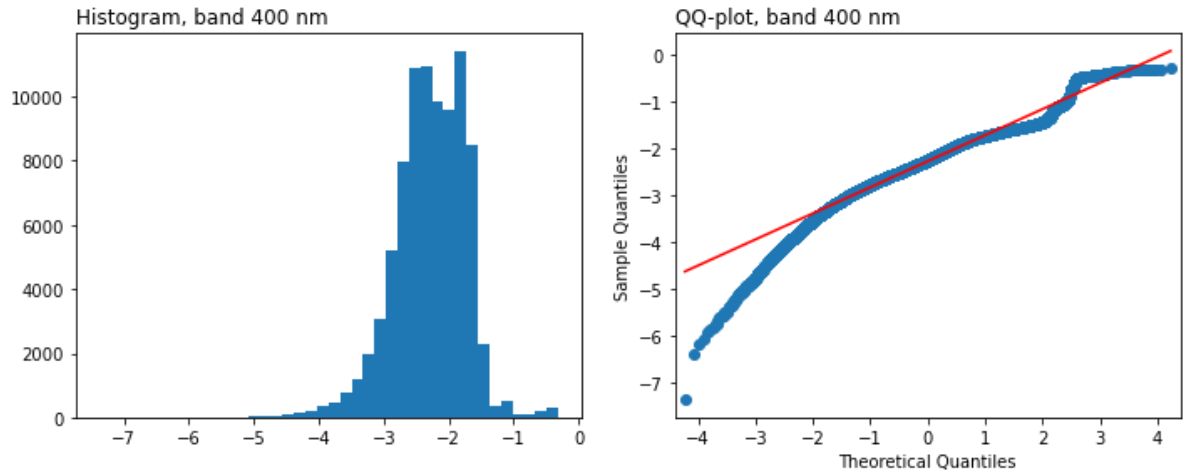
Checking distribution of the reflectances contained within the training data, we observed multimodal structure in histograms from each band, likely representing the various water types present in the time series data over both space (e.g. near coast/open ocean) and time (e.g. seasonal bloom/high river run-off period). Quantile-quantile (QQ) plots for each band appear relatively linear between marked steps from one mode to the next.

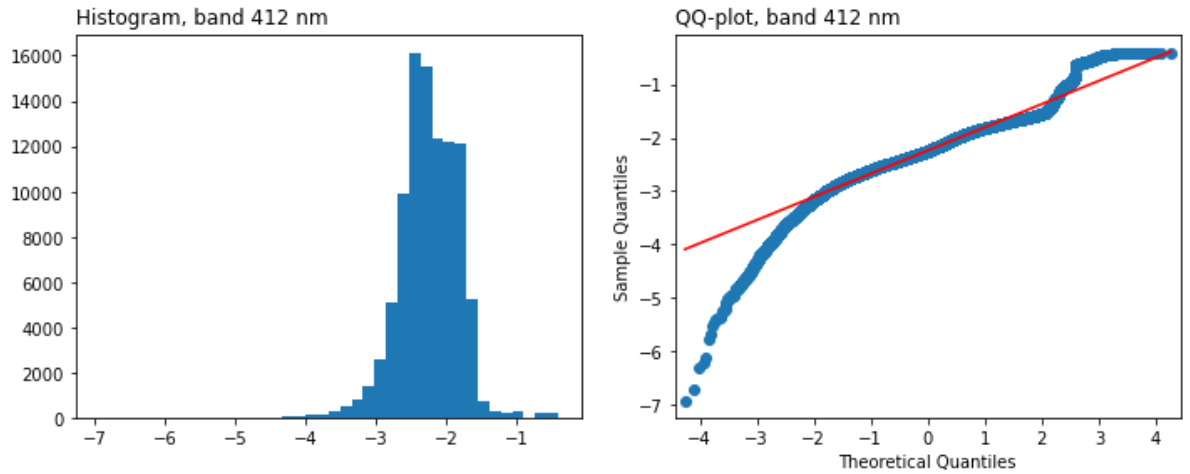
This next step produces histograms and QQ-plots (quantile-quantile) of log-transformed training data for each band used for clustering. Note clear multimodal structure, likely representing different water types occurring over space and time

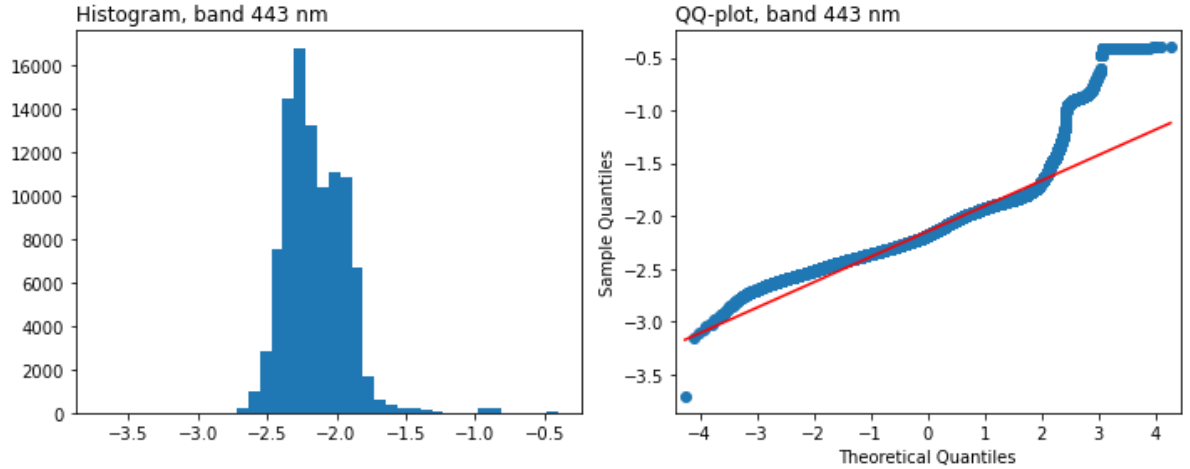
In [90]:

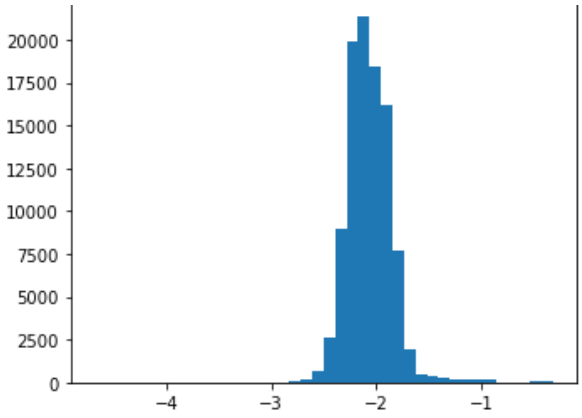
```
# check log normal distribution assumption
print('Per band Rw log10 normal plots on log transformed training data (negative val
fig, axs = plt.subplots(15, 2, figsize=(10, 60), tight_layout=True)
for n,band in enumerate(samples.columns):
    da=samples.iloc[:,n]
    da=da[da>0]
    da=np.log10(da)
    da=np.msort(da)
    axs[n,0].hist(da, bins=40)
    axs[n,0].set_title(f"Histogram, band {band} nm", loc='left')
    qqplot(da, line='s', ax=axs[n,1])
    axs[n,1].set_title(f"QQ-plot, band {band} nm", loc='left')
```

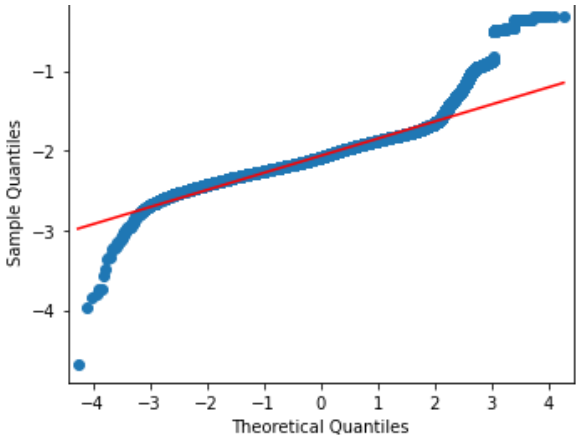
```
Per band Rw log10 normal plots on log transformed training data (negat
ive values removed)
```



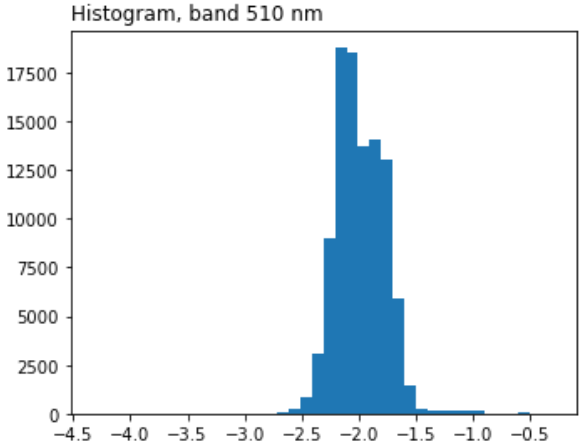




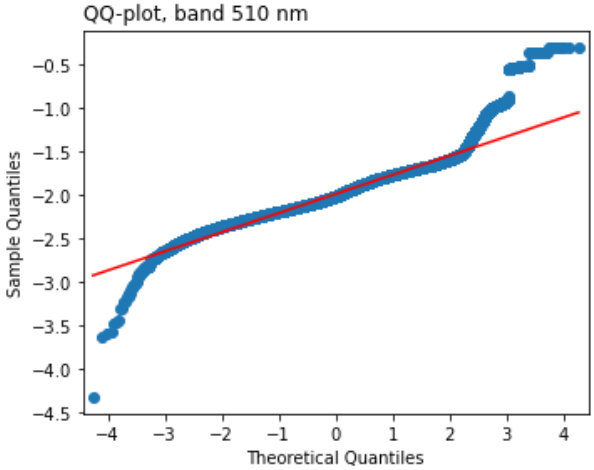




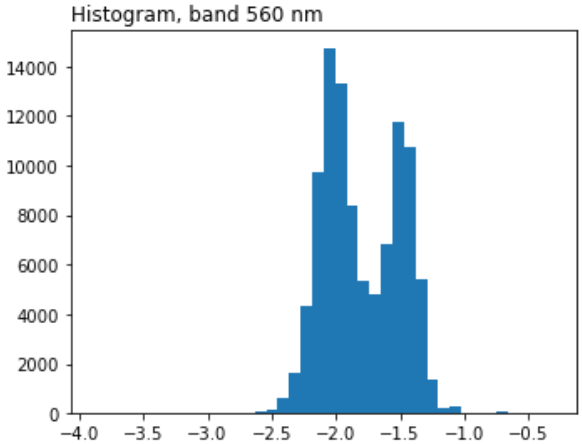
Histogram, band 510 nm



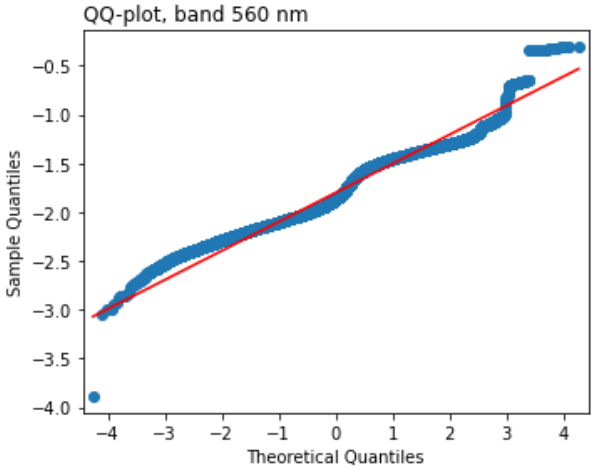
QQ-plot, band 510 nm



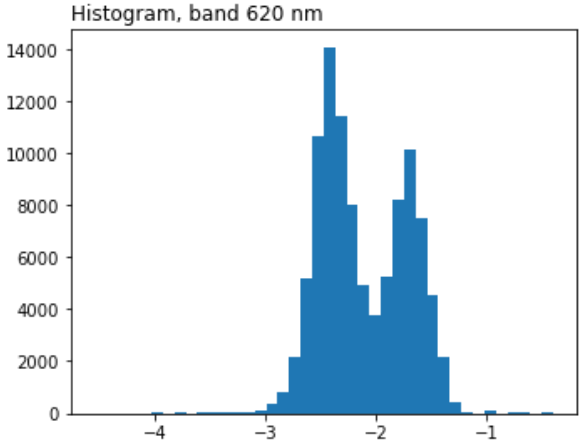
Histogram, band 560 nm



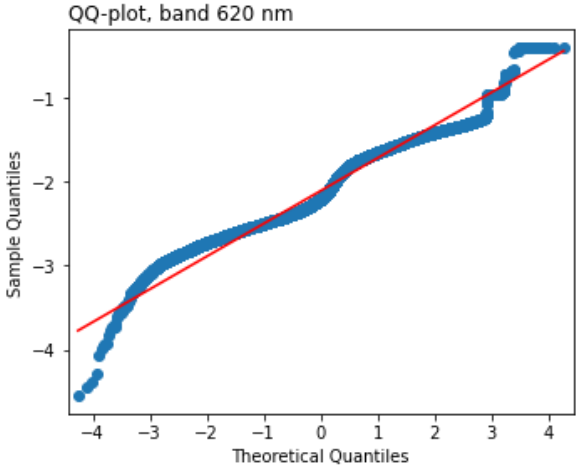
QQ-plot, band 560 nm



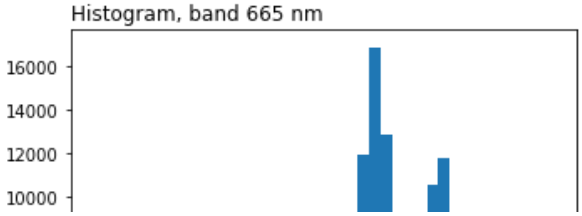
Histogram, band 620 nm



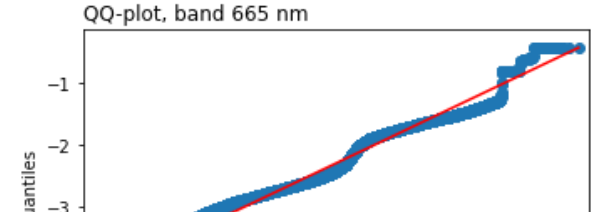
QQ-plot, band 620 nm

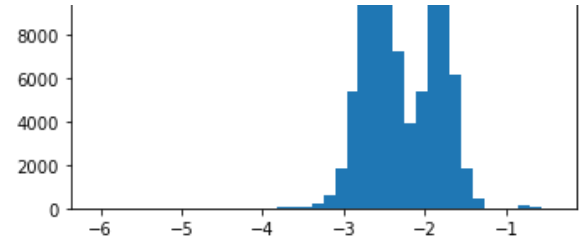


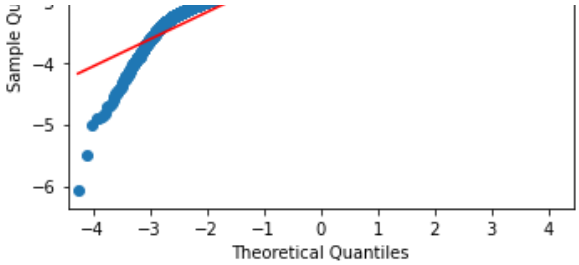
Histogram, band 665 nm



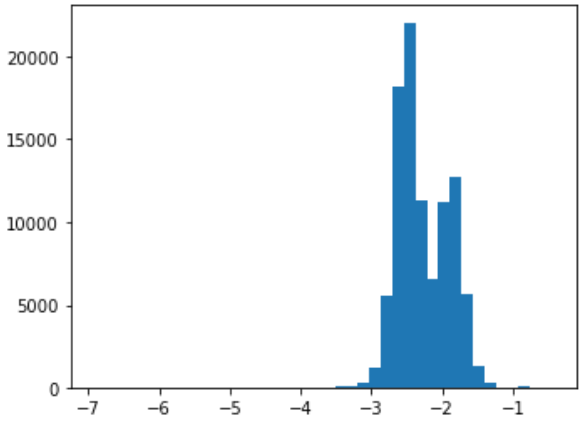
QQ-plot, band 665 nm



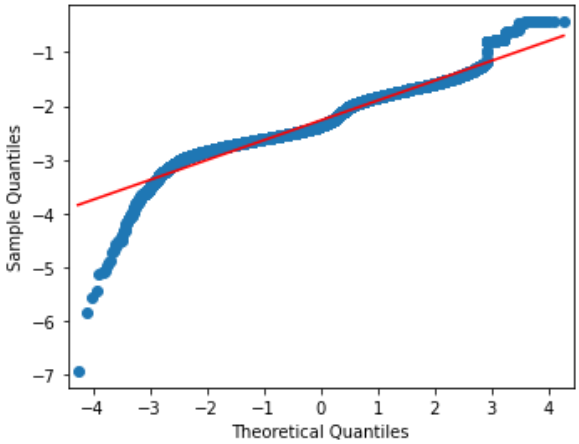




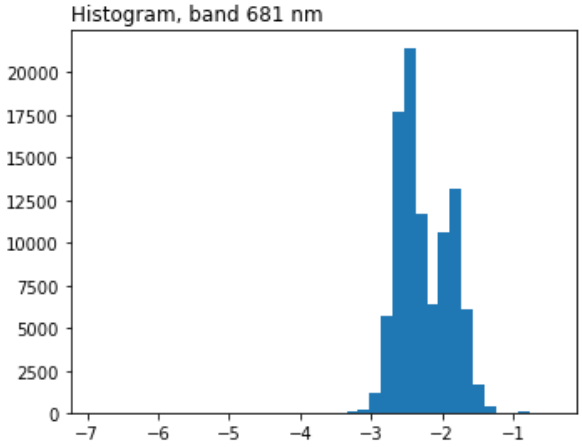
Histogram, band 674 nm



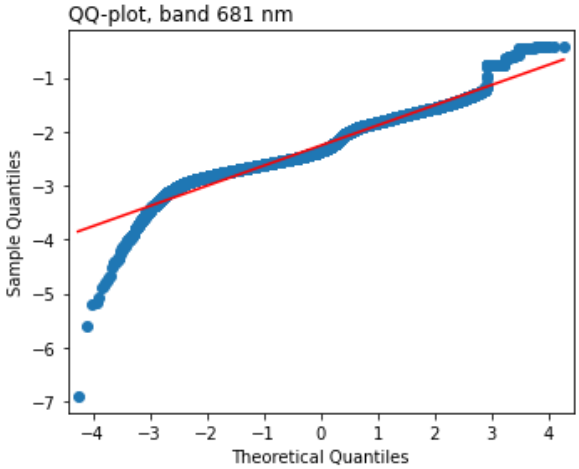
QQ-plot, band 674 nm



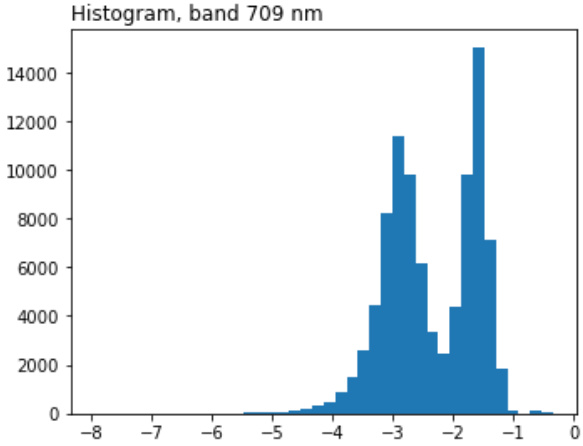
Histogram, band 681 nm



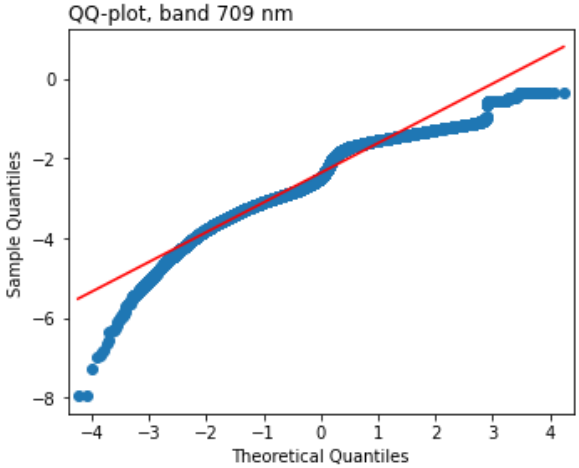
QQ-plot, band 681 nm



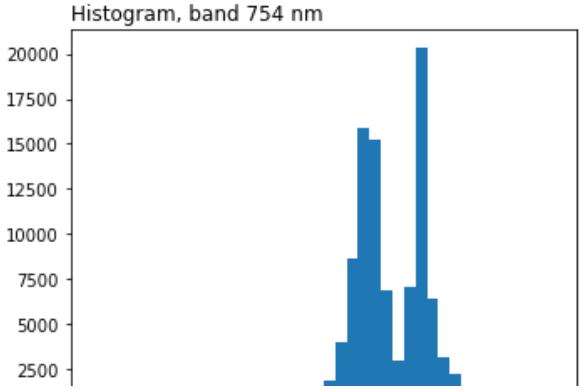
Histogram, band 709 nm



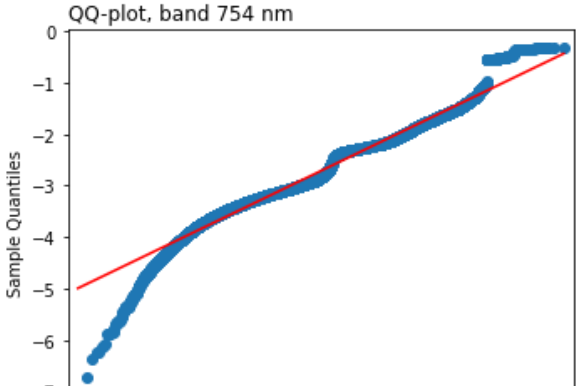
QQ-plot, band 709 nm

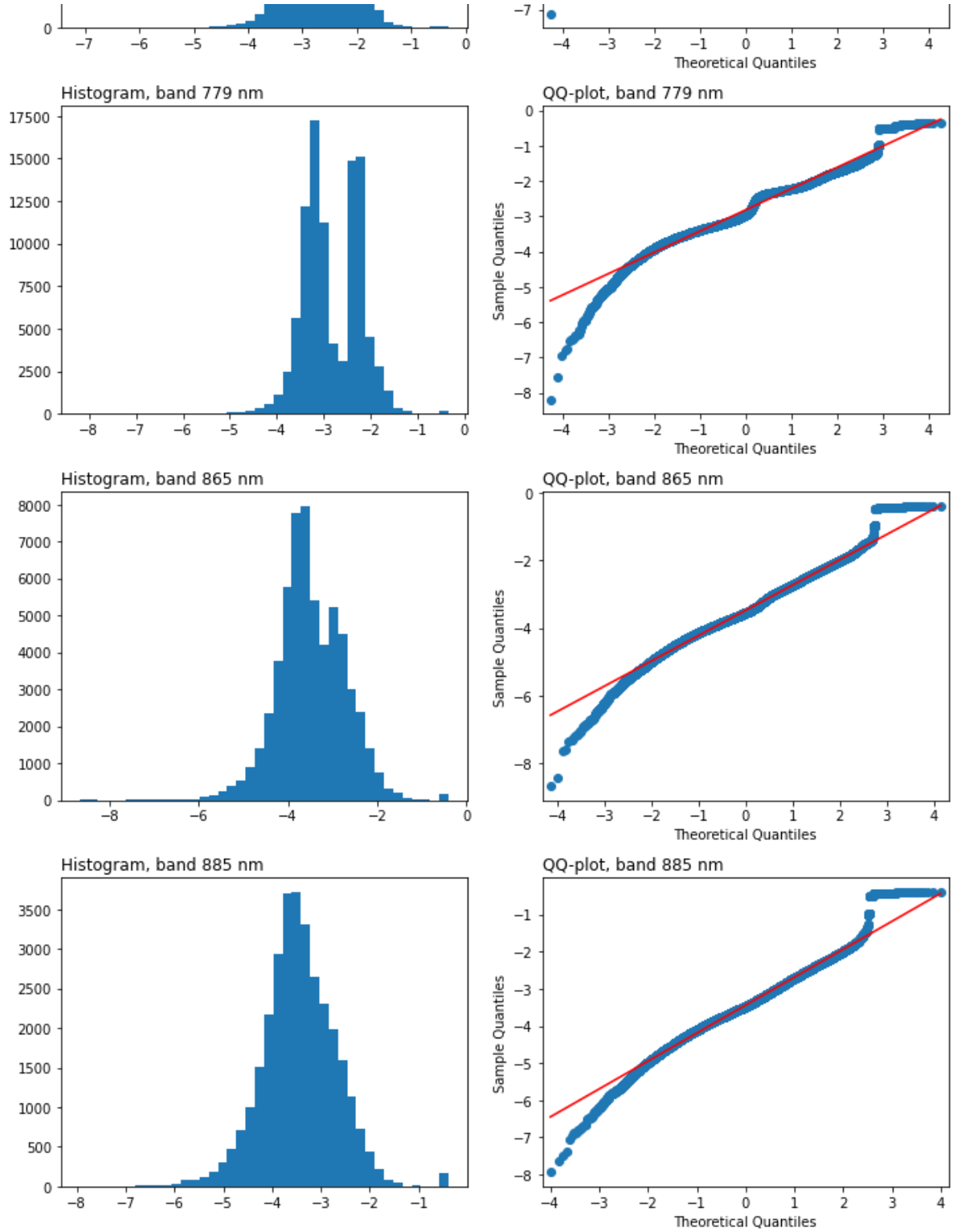


Histogram, band 754 nm



QQ-plot, band 754 nm



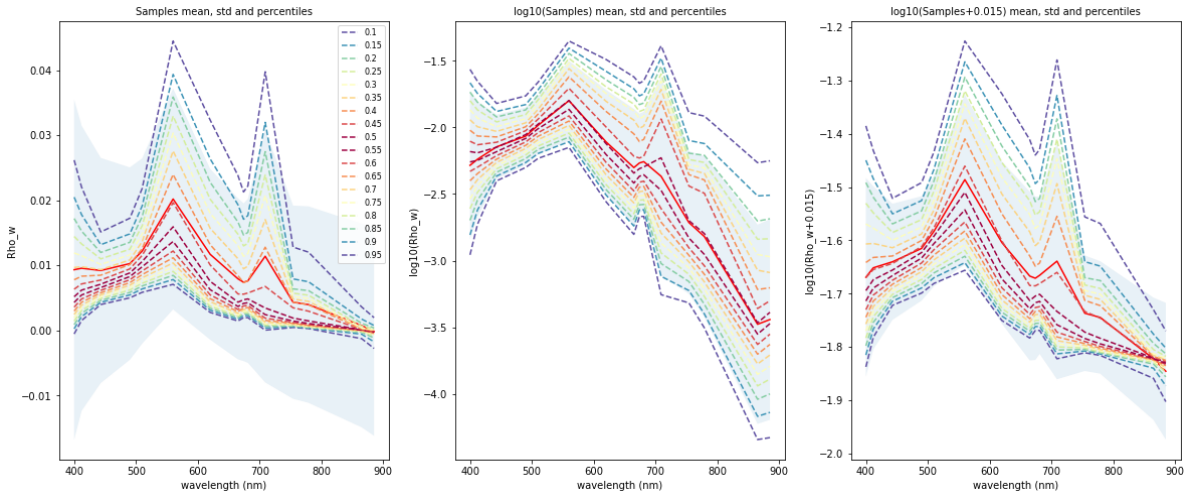


We then check that transformations are properly scaling spectra magnitude differences, without large loss of spectral shape variability captured by the training data.

Distribution of training data spectra, showing sample mean (solid red line), +/- 1 standard deviation (grey shading), and percentile distribution of all spectra (broken lines, rainbow coloured, see legend for percent). Left plot is for untransformed data, middle plot for straight log-transform data (thus losing a sizeable portion of the data), right plot shows log-transform of additive shifted data. The rightmost plot shows more even distribution around the mean and median after both transformations.

In [91]:

```
colors = hv.plotting.util.process_cmap('Spectral', 9)
colors = colors[::-1]+colors

with warnings.catch_warnings():
    warnings.simplefilter("ignore")
    fig, axes=plt.subplots(1,3,figsize=(20,8))
    ax=axes.flatten()[0]
    ax.fill_between(samples.keys().astype(int), samples.mean(axis=0)-samples.std(axi
    ax.plot(samples.keys().astype(int), samples.mean(axis=0), c='r')
    for i, perc in enumerate(np.arange(0.1,1.0,0.05)):
        ax.plot(samples.keys().astype(int), np.percentile(samples, 100*perc, axis=0)
    ax.legend(fontsize=8)
    ax.set_title('Samples mean, std and percentiles', fontsize=10)
    ax.set_xlabel("wavelength (nm)")
    ax.set_ylabel("Rho_w")
    ax=axes.flatten()[1]
    ax.fill_between(samples.keys().astype(int), np.ma.masked_invalid(np.log10(sample
    ax.plot(samples.keys().astype(int), np.ma.masked_invalid(np.log10(samples)).mean
    for i, perc in enumerate(np.arange(0.1,1.0,0.05)):
        ax.plot(samples.keys().astype(int), np.nanpercentile(np.log10(samples), 100*
#     ax.legend(fontsize=8)
    ax.set_title('log10(Samples) mean, std and percentiles', fontsize=10)
    ax.set_xlabel("wavelength (nm)")
    ax.set_ylabel("log10(Rho_w)")
    ax=axes.flatten()[2]
    ax.fill_between(samples.keys().astype(int), np.ma.masked_invalid(np.log10(sample
    ax.plot(samples.keys().astype(int), np.ma.masked_invalid(np.log10(samples+shift)
    for i, perc in enumerate(np.arange(0.1,1.0,0.05)):
        ax.plot(samples.keys().astype(int), np.nanpercentile(np.log10(samples+shift)
#     ax.legend(fontsize=8)
    ax.set_title(f'log10(Samples+{shift}) mean, std and percentiles', fontsize=10)
    ax.set_xlabel("wavelength (nm)")
    ax.set_ylabel(f"log10(Rho_w+{shift})")
```



2. Fuzzy classification

2.1 Define model parameters

In [92]:

```
import os, sys
cwd_path = os.getcwd()
fwc_path = cwd_path[0:cwd_path.rfind("/")]
sys.path.append(fwc_path)
import fuzzy_water_clustering as fwc
import fuzzy_water_clustering.sample_netcdf as sampling

from sklearn.datasets import make_blobs
from sklearn.preprocessing import StandardScaler, FunctionTransformer
from sklearn.decomposition import PCA
from sklearn.metrics import pairwise_distances, make_scorer, silhouette_score
from sklearn.metrics.cluster import calinski_harabasz_score, davies_bouldin_score, s
import sklearn.pipeline as pipeline
from sklearn.model_selection import GridSearchCV
from sklearn import set_config
```

In [100]:

```
# create functions for transformations based on analyses above
# shift = 0.015 from above

def negative_adjust(x):
    return np.log10(x+0.015, dtype='float32')

def negative_adjust_inverse(x):
    return (10**x)-0.015

transforms = FunctionTransformer(negative_adjust, inverse_func=negative_adjust_inver
transforms
```

Out[100]:

```
FunctionTransformer(check_inverse=False,
                    func=<function negative_adjust at 0x7f217d9c53a0>,
                    inverse_func=<function negative_adjust_inverse at
0x7f217d9c51f0>)
```

In [101]:

```
pl = pipeline.make_pipeline(
    transforms,
    PCA(),
    fwc.CmeansModel()
)
```

In [102]:

```
cluster_paramgrid = {
#     'cmeansmodel__n_clusters': list(np.arange(6,12,1)),
    'cmeansmodel__n_clusters': list(np.arange(6,8,1)),
    'cmeansmodel__m': list(np.arange(1.5,2.2,0.3))
}
```

In [103]:

```
scoring = {
    'XB' : fwc.cluster_scoring.xie_beni,
    'SIL': fwc.cluster_scoring.hard_silouette,
    'FPC': fwc.cluster_scoring.fuzzy_partition_coef,
    'DB' : fwc.cluster_scoring.davies_bouldin,
}
```

2.2 Cleaning steps

First perform operations to remove NaNs and unusually high reflectance values

In [104]:

```
samples_orig = samples
samples = samples.drop(samples[np.isnan(transforms.transform(samples)).any(axis=1)].
print(f'Cleaning input samples with transformation, n reduced from {len(samples_orig
samples
```

```
Cleaning input samples with transformation, n reduced from 97988 to 97
988
```

Out[104]:

	400	412	443	490	510	560	620	665	67
0	0.023530	0.019728	0.012376	0.016155	0.022873	0.051459	0.034511	0.027195	0.02325
1	0.006053	0.007931	0.010238	0.011270	0.012659	0.018133	0.005843	0.002309	0.00206
2	0.004672	0.005603	0.006207	0.008114	0.009371	0.011813	0.004722	0.003105	0.00330
3	0.007548	0.009193	0.007935	0.009069	0.009479	0.009612	0.002398	0.001188	0.00108
4	0.001264	0.003000	0.005062	0.006881	0.007425	0.009551	0.003916	0.002481	0.00325
...	...	...	...	...	...	...	...	...	...
99995	0.003940	0.006028	0.007448	0.009441	0.011318	0.014464	0.006071	0.003250	0.00400
99996	0.000854	0.001821	0.003437	0.004304	0.005114	0.006025	0.002184	0.002195	0.00229
99997	0.005037	0.005746	0.005939	0.008130	0.009653	0.017439	0.013926	0.008927	0.00840
99998	0.018886	0.018937	0.012623	0.010667	0.011442	0.029726	0.009995	0.005361	0.00345
99999	0.020202	0.016881	0.012270	0.013400	0.017730	0.036885	0.021047	0.016419	0.01377

97988 rows × 15 columns

In [105]:

```
# remove training data with reflectances over 0.2
for n,band in enumerate(samples.columns):
    if n == 0:
        exclude = np.full(len(samples[band]), False)
        exclude[np.where(samples[band] > 0.2)[0]] = True
    else:
        exclude_temp = np.full(len(samples[band]), False)
        exclude_temp[np.where(samples[band] > 0.2)[0]] = True
        exclude = exclude | exclude_temp

nan_n = len(samples)
samples = samples.iloc[~exclude, :]
print(f'Cleaning input samples with high reflectance, n further reduced from {nan_n}
samples
```

```
Cleaning input samples with high reflectance, n further reduced from 9
7988 to 97988
```

Out[105]:

	400	412	443	490	510	560	620	665	67
0	0.023530	0.019728	0.012376	0.016155	0.022873	0.051459	0.034511	0.027195	0.02325
1	0.006053	0.007931	0.010238	0.011270	0.012659	0.018133	0.005843	0.002309	0.00206
2	0.004672	0.005603	0.006207	0.008114	0.009371	0.011813	0.004722	0.003105	0.00330
3	0.007548	0.009193	0.007935	0.009069	0.009479	0.009612	0.002398	0.001188	0.00108
4	0.001264	0.003000	0.005062	0.006881	0.007425	0.009551	0.003916	0.002481	0.00325
...	...	...	...	...	...	...	...	...	...
99995	0.003940	0.006028	0.007448	0.009441	0.011318	0.014464	0.006071	0.003250	0.00400
99996	0.000854	0.001821	0.003437	0.004304	0.005114	0.006025	0.002184	0.002195	0.00229
99997	0.005037	0.005746	0.005939	0.008130	0.009653	0.017439	0.013926	0.008927	0.00840
99998	0.018886	0.018937	0.012623	0.010667	0.011442	0.029726	0.009995	0.005361	0.00345
99999	0.020202	0.016881	0.012270	0.013400	0.017730	0.036885	0.021047	0.016419	0.01377

97988 rows × 15 columns

2.3 Run gridsearch to identify best model

Run the estimator to find the best model

Warning this takes a long, please be patient. If you have lots of CPU cores the `n_jobs` parameter can be increased, or set to `-1` to use *all* cores

The best performing model is stored in `final_pipeline` and provides the optimum number of OWTs, `n_clusters` , and the best fuzziness as `m`

In [106]:

```
gs_clusters = GridSearchCV(
    pl,
    param_grid=cluster_paramgrid,
    scoring=scoring,
    refit='XB',
    n_jobs=6,
    verbose=4
)
gs_clusters.fit(samples)
print(gs_clusters.best_estimator_['cmeansmodel'])

final_pipeline = gs_clusters.best_estimator_
final_pipeline
```

```
Fitting 5 folds for each of 6 candidates, totalling 30 fits
CmeansModel(m=2.1, n_clusters=6)
```

Out[106]:

```
Pipeline(steps=[('functiontransformer',
                 FunctionTransformer(check_inverse=False,
                                     func=<function negative_adjust at
0x7f217d9c53a0>,
                                     inverse_func=<function negative_a
djust_inverse at 0x7f217d9c51f0>)),
                ('pca', PCA()),
                ('cmeansmodel', CmeansModel(m=2.1, n_clusters=6))])
```

```
[CV 5/5] END cmeansmodel__m=1.5, cmeansmodel__n_clusters=6; DB: (test=
-1.122) FPC: (test=0.766) SIL: (test=0.300) XB: (test=-1.747) total ti
me=  30.7s
[CV 2/5] END cmeansmodel__m=1.8, cmeansmodel__n_clusters=6; DB: (test=
-1.129) FPC: (test=0.602) SIL: (test=0.293) XB: (test=-1.494) total ti
me=  23.5s
[CV 1/5] END cmeansmodel__m=1.8, cmeansmodel__n_clusters=7; DB: (test=
-1.180) FPC: (test=0.535) SIL: (test=0.230) XB: (test=-1.842) total ti
me=  45.4s
[CV 1/5] END cmeansmodel__m=2.1, cmeansmodel__n_clusters=7; DB: (test=
-1.198) FPC: (test=0.404) SIL: (test=0.218) XB: (test=-1.368) total ti
me=  30.3s
[CV 1/5] END cmeansmodel__m=1.5, cmeansmodel__n_clusters=7; DB: (test=
-1.246) FPC: (test=0.758) SIL: (test=0.301) XB: (test=-1.786) total ti
me=  26.1s
[CV 4/5] END cmeansmodel__m=1.5, cmeansmodel__n_clusters=7; DB: (test=
-1.174) FPC: (test=0.713) SIL: (test=0.234) XB: (test=-2.252) total ti
me=  38.5s
[CV 3/5] END cmeansmodel__m=1.8, cmeansmodel__n_clusters=7; DB: (test=
-1.188) FPC: (test=0.535) SIL: (test=0.230) XB: (test=-1.849) total ti
me=  33.7s
[CV 4/5] END cmeansmodel__m=2.1, cmeansmodel__n_clusters=6; DB: (test=
-1.219) FPC: (test=0.431) SIL: (test=0.229) XB: (test=-1.347) total ti
me=  34.4s
[CV 1/5] END cmeansmodel__m=1.5, cmeansmodel__n_clusters=6; DB: (test=
-1.106) FPC: (test=0.764) SIL: (test=0.309) XB: (test=-1.756) total ti
me=  17.4s
[CV 2/5] END cmeansmodel__m=1.5, cmeansmodel__n_clusters=7; DB: (test=
-1.178) FPC: (test=0.713) SIL: (test=0.234) XB: (test=-2.231) total ti
me=  27.1s
[CV 3/5] END cmeansmodel__m=1.8, cmeansmodel__n_clusters=6; DB: (test=
```

```
-1.128) FPC: (test=0.602) SIL: (test=0.296) XB: (test=-1.496) total ti
me=  23.9s
[CV 5/5] END cmeansmodel__m=1.8, cmeansmodel__n_clusters=7; DB: (test=
-1.192) FPC: (test=0.536) SIL: (test=0.225) XB: (test=-1.851) total ti
me=  31.3s
[CV 5/5] END cmeansmodel__m=2.1, cmeansmodel__n_clusters=6; DB: (test=
-1.218) FPC: (test=0.431) SIL: (test=0.229) XB: (test=-1.348) total ti
me=  33.8s
[CV 3/5] END cmeansmodel__m=1.5, cmeansmodel__n_clusters=6; DB: (test=
-1.110) FPC: (test=0.764) SIL: (test=0.311) XB: (test=-1.755) total ti
me=  19.8s
[CV 3/5] END cmeansmodel__m=1.5, cmeansmodel__n_clusters=7; DB: (test=
-1.173) FPC: (test=0.712) SIL: (test=0.238) XB: (test=-2.248) total ti
me=  27.2s
[CV 4/5] END cmeansmodel__m=1.8, cmeansmodel__n_clusters=6; DB: (test=
-1.137) FPC: (test=0.603) SIL: (test=0.291) XB: (test=-1.492) total ti
me=  20.5s
[CV 4/5] END cmeansmodel__m=1.8, cmeansmodel__n_clusters=7; DB: (test=
-1.188) FPC: (test=0.535) SIL: (test=0.227) XB: (test=-1.856) total ti
me=  19.3s
[CV 3/5] END cmeansmodel__m=2.1, cmeansmodel__n_clusters=6; DB: (test=
-1.211) FPC: (test=0.431) SIL: (test=0.233) XB: (test=-1.345) total ti
me=  27.3s
[CV 3/5] END cmeansmodel__m=2.1, cmeansmodel__n_clusters=7; DB: (test=
-1.206) FPC: (test=0.404) SIL: (test=0.219) XB: (test=-1.377) total ti
me=  20.5s
[CV 4/5] END cmeansmodel__m=1.5, cmeansmodel__n_clusters=6; DB: (test=
-1.122) FPC: (test=0.765) SIL: (test=0.306) XB: (test=-1.754) total ti
me=  29.3s
[CV 1/5] END cmeansmodel__m=1.8, cmeansmodel__n_clusters=6; DB: (test=
-1.122) FPC: (test=0.602) SIL: (test=0.294) XB: (test=-1.491) total ti
me=  23.3s
[CV 5/5] END cmeansmodel__m=1.8, cmeansmodel__n_clusters=6; DB: (test=
-1.139) FPC: (test=0.604) SIL: (test=0.286) XB: (test=-1.482) total ti
me=  30.9s
[CV 1/5] END cmeansmodel__m=2.1, cmeansmodel__n_clusters=6; DB: (test=
-1.147) FPC: (test=0.471) SIL: (test=0.277) XB: (test=-1.157) total ti
me=  31.7s
[CV 4/5] END cmeansmodel__m=2.1, cmeansmodel__n_clusters=7; DB: (test=
-1.208) FPC: (test=0.404) SIL: (test=0.215) XB: (test=-1.382) total ti
me=  23.3s
[CV 2/5] END cmeansmodel__m=1.5, cmeansmodel__n_clusters=6; DB: (test=
-1.112) FPC: (test=0.764) SIL: (test=0.309) XB: (test=-1.753) total ti
me=  26.3s
[CV 5/5] END cmeansmodel__m=1.5, cmeansmodel__n_clusters=7; DB: (test=
-1.179) FPC: (test=0.712) SIL: (test=0.232) XB: (test=-2.256) total ti
me=  37.5s
[CV 2/5] END cmeansmodel__m=1.8, cmeansmodel__n_clusters=7; DB: (test=
-1.190) FPC: (test=0.536) SIL: (test=0.226) XB: (test=-1.834) total ti
me=  20.4s
[CV 2/5] END cmeansmodel__m=2.1, cmeansmodel__n_clusters=6; DB: (test=
-1.153) FPC: (test=0.471) SIL: (test=0.276) XB: (test=-1.162) total ti
me=  21.5s
[CV 2/5] END cmeansmodel__m=2.1, cmeansmodel__n_clusters=7; DB: (test=
-1.209) FPC: (test=0.404) SIL: (test=0.214) XB: (test=-1.363) total ti
me=  22.2s
[CV 5/5] END cmeansmodel__m=2.1, cmeansmodel__n_clusters=7; DB: (test=
-1.211) FPC: (test=0.405) SIL: (test=0.213) XB: (test=-1.376) total ti
me=  15.2s
```

2.4 Inspect results

Here we can see the results of the fuzzy classification model run

In [107]:

```
samples_results = pd.DataFrame(gs_clusters.cv_results_)
samples_results
```

Out[107]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_cmeansmodel__m	paran
0	17.114392	4.203982	7.577053	1.265242	1.5	
1	23.853080	4.976823	7.449444	1.497519	1.5	
2	17.127491	3.153444	7.289993	0.439566	1.8	
3	22.835234	8.560193	7.173323	1.261692	1.8	
4	22.869498	3.546126	6.849932	1.301527	2.1	
5	17.048011	3.583293	5.274250	2.346233	2.1	

6 rows × 39 columns

Focusing on the best fit model, display a table of cluster centre values. For each reflectance wavelength (row), the centre value is defined for each OWT (column)

In [108]:

```
# table cluster centers
class_spectra = final_pipeline['functiontransformer'].inverse_transform(
    final_pipeline['pca'].inverse_transform(
        final_pipeline['cmeansmodel'].cluster_centers_
    )
)

class_spectra = pd.DataFrame(class_spectra.T, columns=['Class_'+str(x) for x in np.a
class_spectra.index.names = ['Cluster']
class_spectra.index = samples.columns
print("Table of cluster centers")
class_spectra
```

```
Table of cluster centers
```

Out[108]:

	Class_1	Class_2	Class_3	Class_4	Class_5	Class_6
400	0.000499	0.003061	0.004783	0.010133	0.020180	0.014599
412	0.002254	0.004409	0.006119	0.010266	0.017555	0.013100
443	0.004404	0.005701	0.007429	0.009765	0.011999	0.010708
490	0.005542	0.007244	0.009501	0.011757	0.012036	0.013985
510	0.006449	0.008278	0.011267	0.014994	0.015522	0.019113
560	0.008067	0.010252	0.016298	0.029629	0.034080	0.040550
620	0.003603	0.004249	0.007936	0.020381	0.017424	0.032630
665	0.002223	0.002609	0.005010	0.014832	0.012993	0.024152
674	0.002882	0.003114	0.005184	0.013263	0.010523	0.021499
681	0.002936	0.003122	0.005221	0.013663	0.010946	0.022673
709	0.001004	0.001102	0.003583	0.019149	0.025958	0.038164
754	0.000820	0.000924	0.001900	0.005392	0.005541	0.011889
779	0.000557	0.000650	0.001518	0.005002	0.005406	0.011754
865	-0.000021	-0.000036	-0.000050	0.000752	-0.001129	0.003745
885	-0.000309	-0.000366	-0.000686	-0.000394	-0.002781	0.001686

Save these results to file:

In [110]:

```
class_spectra.to_csv(f'{site}_S3_{run_date}_classSpectra.csv', index=False)
```

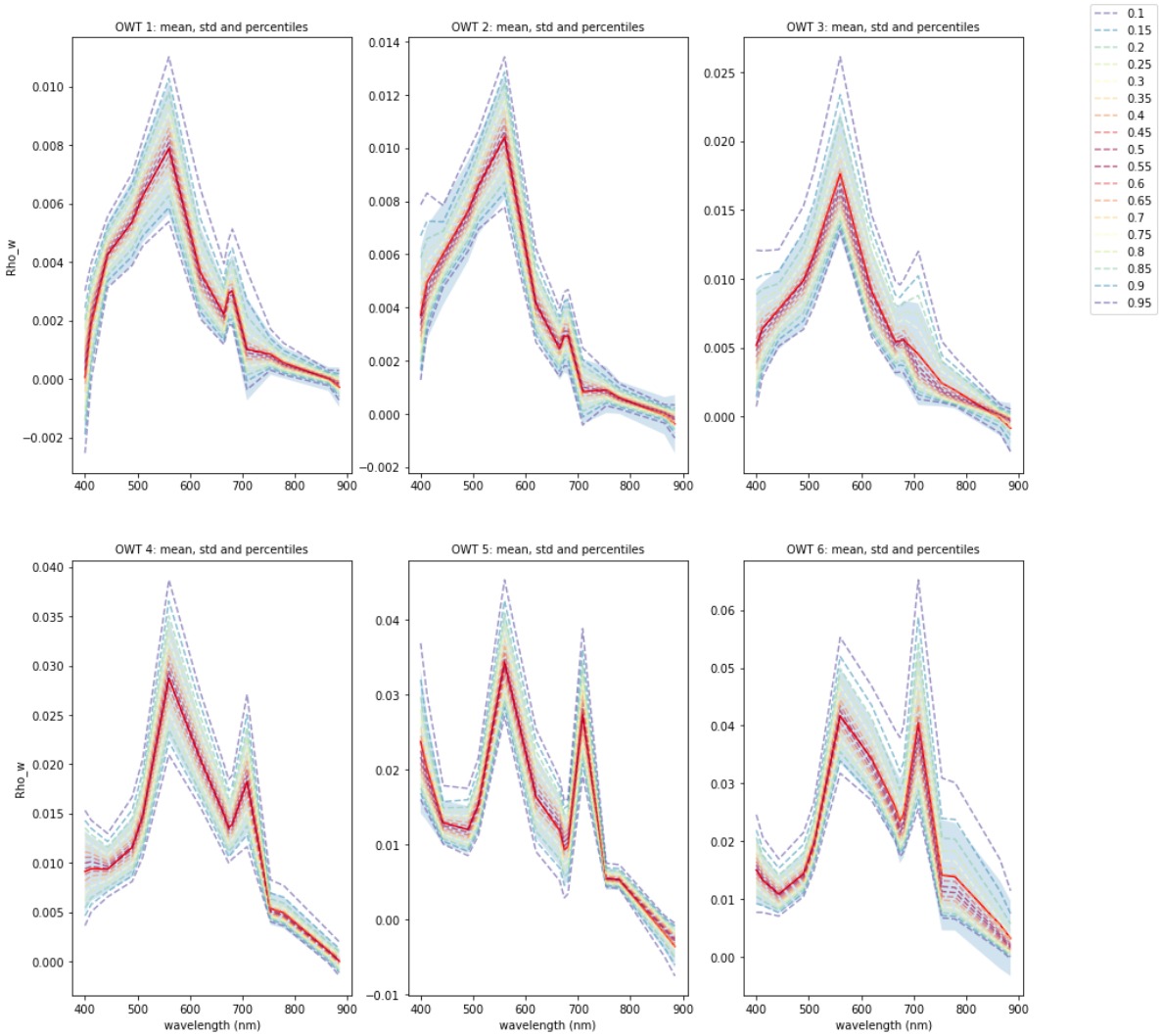
Plot the spectra for each OWT:

In [114]:

```
from copy import deepcopy

# cluster center spectra with mean, s.d. & percentiles
colors = hv.plotting.util.process_cmap('Spectral', 9)
colors = colors[::-1]+colors

point_cols = deepcopy(samples)
point_cols['dom_OWT'] = np.argmax(final_pipeline.predict(samples), axis=0)

with warnings.catch_warnings():
    warnings.simplefilter("ignore")
    nrow_base = np.ceil(class_spectra.shape[1]/3)
    if np.ceil(class_spectra.shape[1]/3) == np.floor(class_spectra.shape[1]/3):
        a = 1
    else:
        a = 2
    fig, axes_sub=plt.subplots(nrows=int(nrow_base)+a, ncols=3, figsize=(15,12*int(r
    axes_lrg=plt.subplot(3,1,3)
    axes = [axes_sub, axes_lrg]
    for clust, ax in enumerate(axes_sub.flatten()):
        owt_samples = point_cols.loc[point_cols['dom_OWT'] == clust].iloc[:,:-1]
        ax.fill_between(owt_samples.keys().astype(int),
                        owt_samples.mean(axis=0)-owt_samples.std(axis=0),
                        owt_samples.mean(axis=0)+owt_samples.std(axis=0),
                        alpha=0.2
                       )
        ax.plot(owt_samples.keys().astype(int), owt_samples.mean(axis=0), c='r')
        if owt_samples.shape[0] != 0:
            for i, perc in enumerate(np.arange(0.1,1.0,0.05)):
                if clust == 0:
                    ax.plot(owt_samples.keys().astype(int),
                    np.percentile(owt_samples, 100*perc, axis=0),
                    linestyle='dashed',
                    label=np.round(perc,2),
                    color=colors[i],
                    alpha=0.6
                    )
                else:
                    ax.plot(owt_samples.keys().astype(int),
                    np.percentile(owt_samples, 100*perc, axis=0),
                    linestyle='dashed',
                    color=colors[i],
                    alpha=0.6
                    )
                ax.set_title(f'OWT {clust+1}: mean, std and percentiles', fontsize=1
                if clust%3 == 0:
                    ax.set_ylabel("Rho_w")
                if np.ceil(class_spectra.shape[1]/3) == np.floor(class_spectra.shape
                    if clust >= int(nrow_base-1)*3:
                        ax.set_xlabel("wavelength (nm)")
                elif clust >= int(np.floor(class_spectra.shape[1]/3))*3:
                    ax.set_xlabel("wavelength (nm)")
fig.legend(bbox_to_anchor=(0.95, 0.9), loc='upper left')
```

Out[114]:

```
<matplotlib.legend.Legend at 0x7f2178fbd160>
```

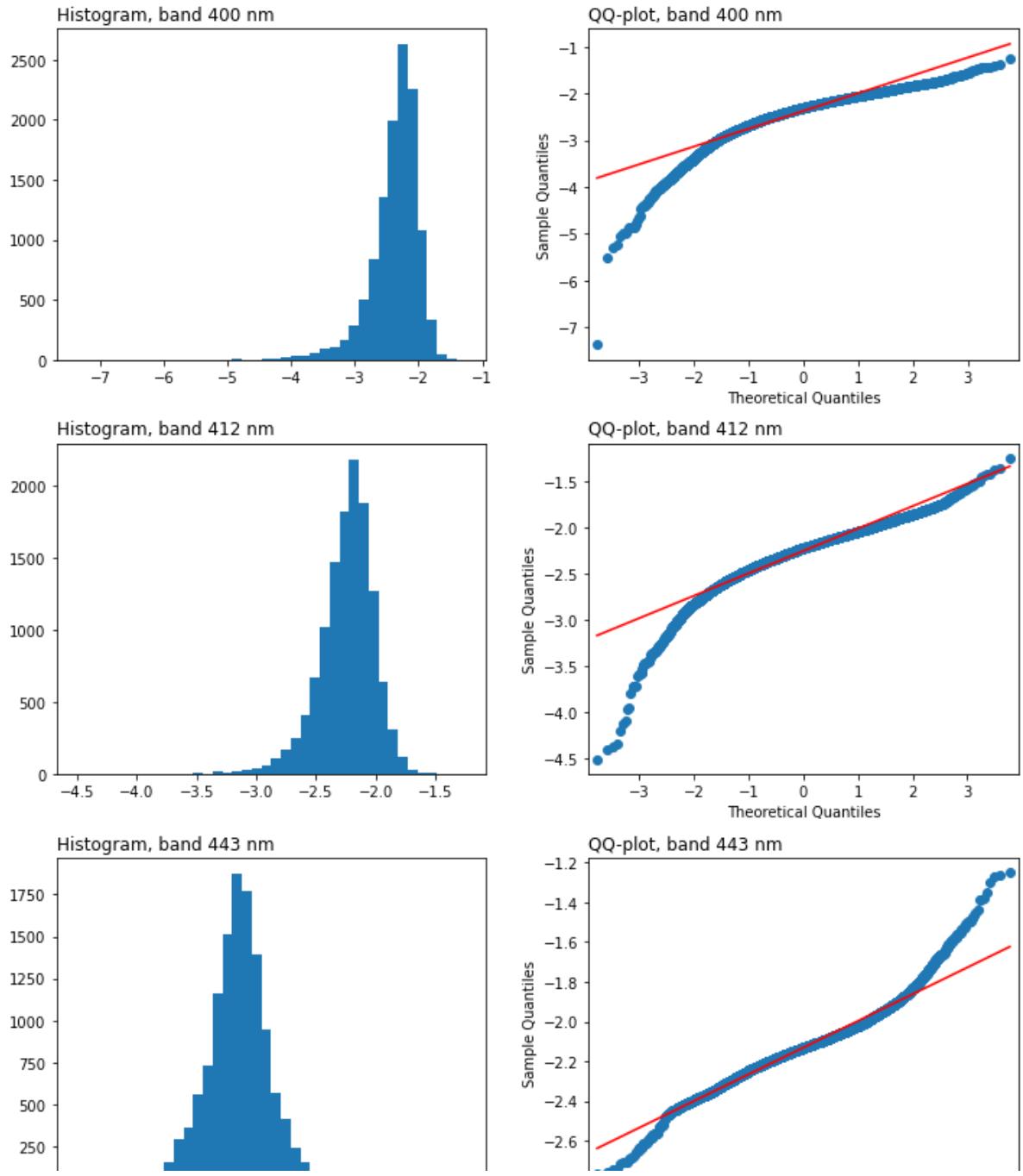


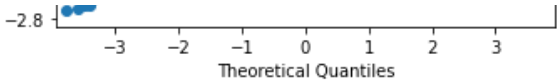
We can produce the same set of histograms and QQ-plots to check for a normal distribution

In [115]:

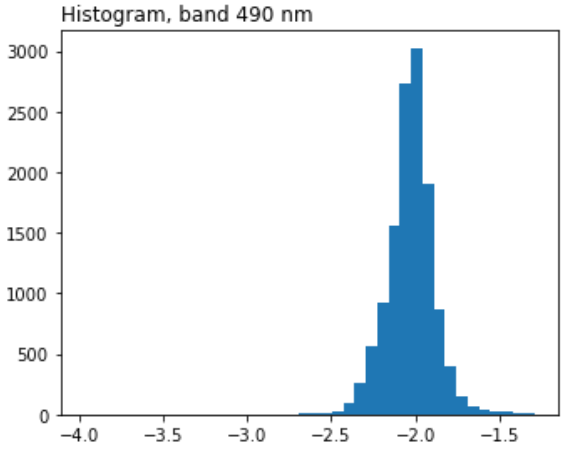
```
# histograms to check log normal distribution assumption for each OWT
# any improvement prior to clustering?
clust = 2

print(f'For OWT {clust+1}: Per band Rw log10 normal plots on log transformed trainir
fig, axs = plt.subplots(15, 2, figsize=(10, 60), tight_layout=True)
for n,band in enumerate(samples.columns):
    da=point_cols.loc[point_cols['dom_OWT'] == clust].iloc[:,n]
    da=da[da>0]
    da=np.log10(da)
    da=np.msort(da)
    axs[n,0].hist(da, bins=40)
    axs[n,0].set_title(f"Histogram, band {band} nm", loc='left')
    qqplot(da, line='s', ax=axs[n,1])
    axs[n,1].set_title(f"QQ-plot, band {band} nm", loc='left')
```

```
For OWT 3: Per band Rw log10 normal plots on log transformed training
data (negative values removed)
```

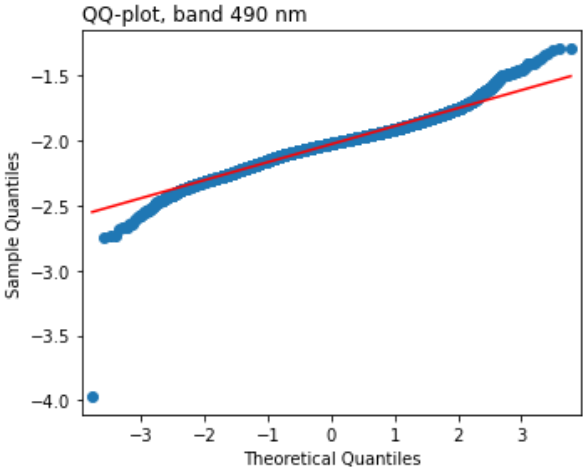




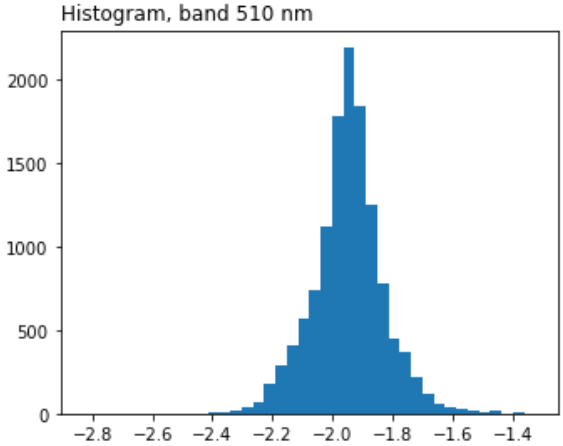
Histogram, band 490 nm



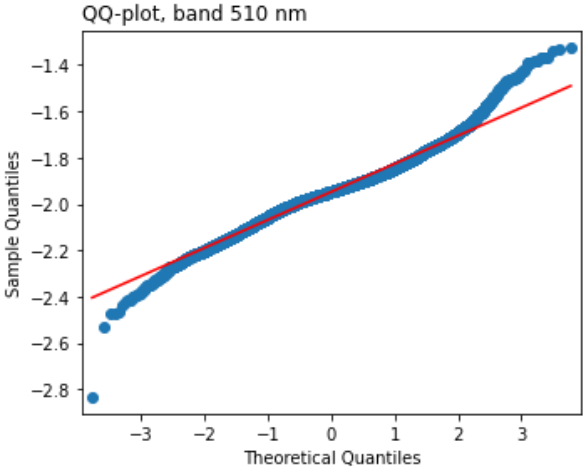
QQ-plot, band 490 nm



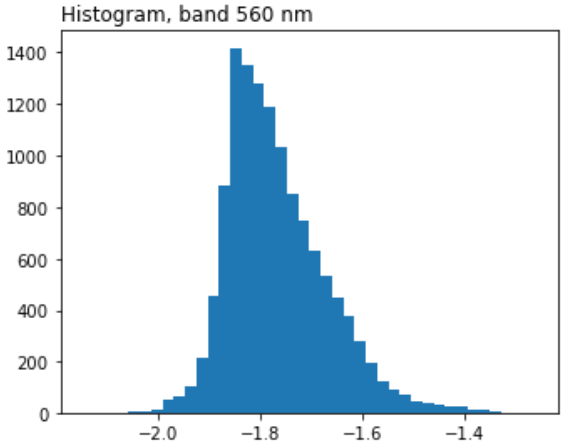
Histogram, band 510 nm



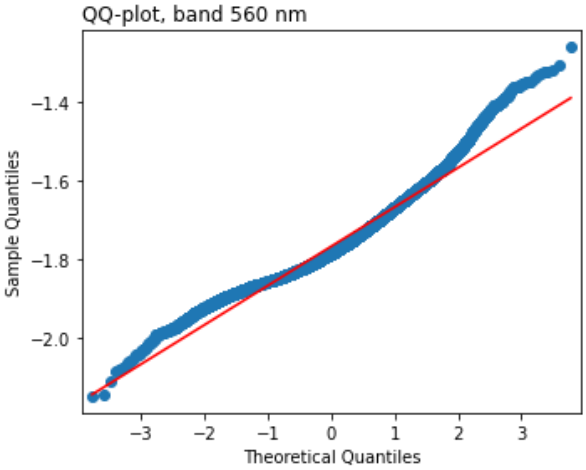
QQ-plot, band 510 nm



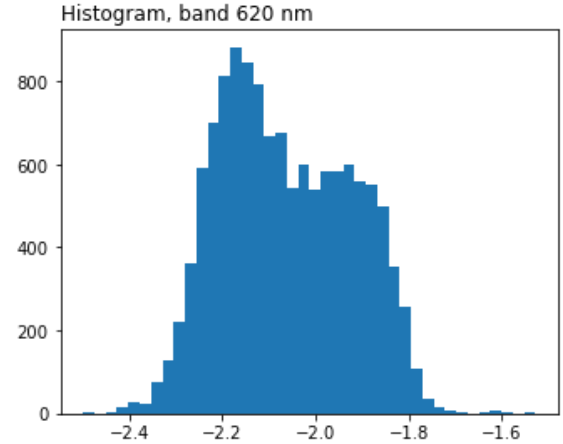
Histogram, band 560 nm



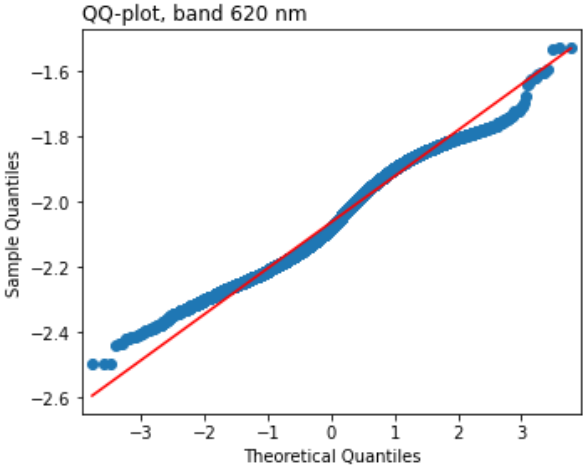
QQ-plot, band 560 nm



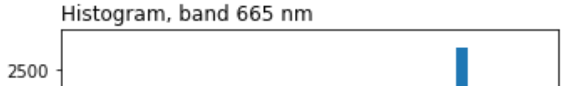
Histogram, band 620 nm



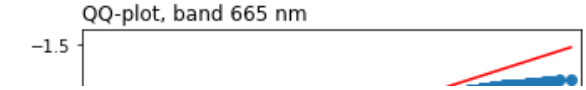
QQ-plot, band 620 nm

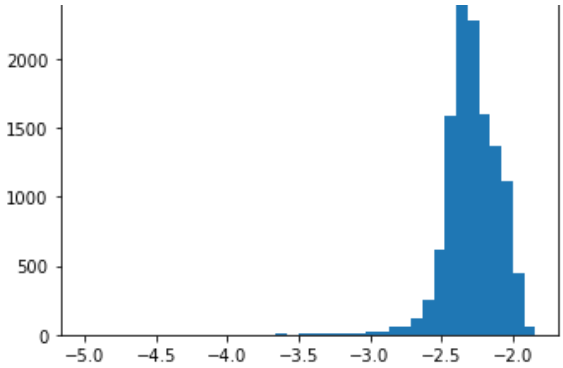


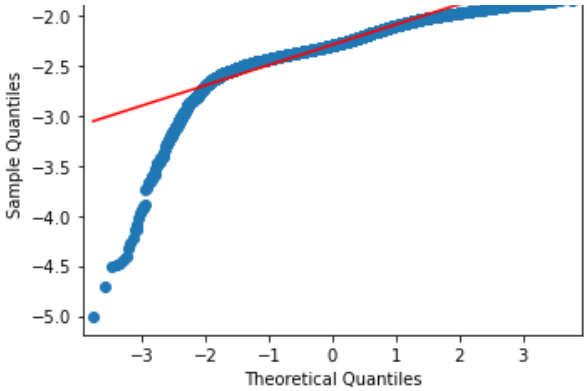
Histogram, band 665 nm



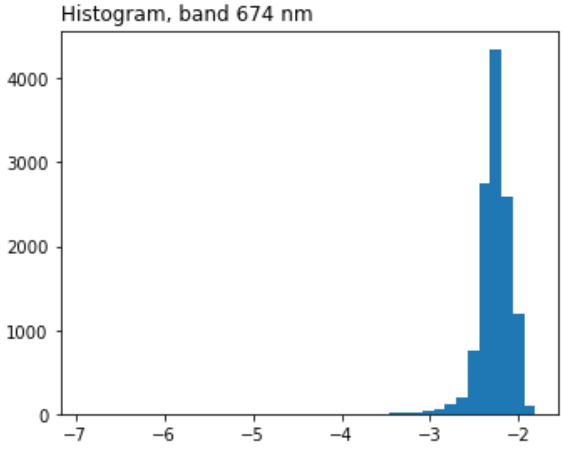
QQ-plot, band 665 nm



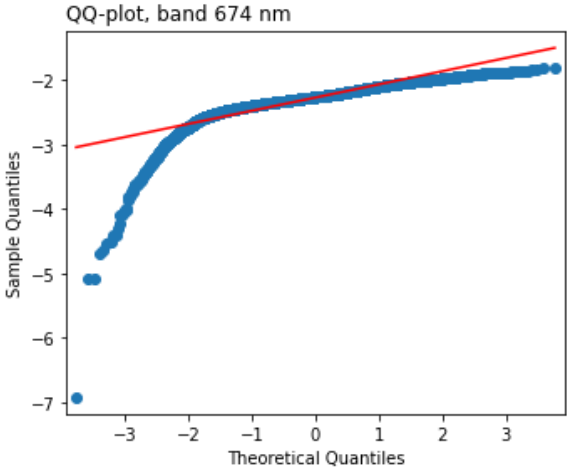




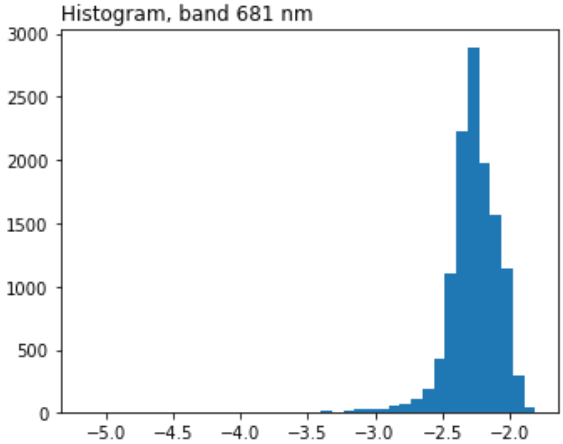
Histogram, band 674 nm



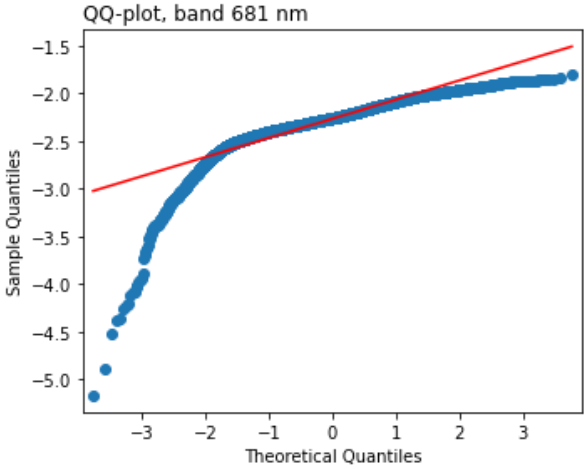
QQ-plot, band 674 nm



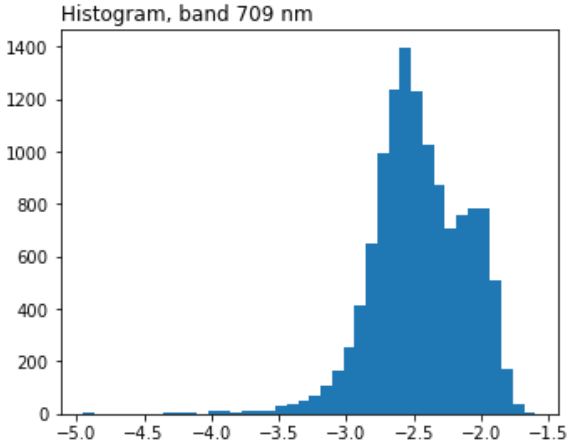
Histogram, band 681 nm



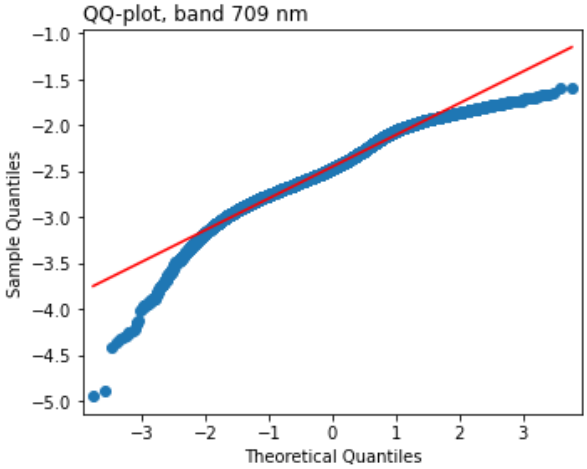
QQ-plot, band 681 nm



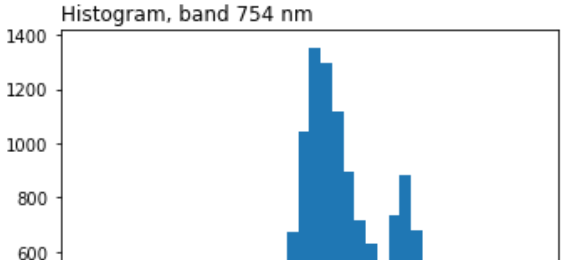
Histogram, band 709 nm



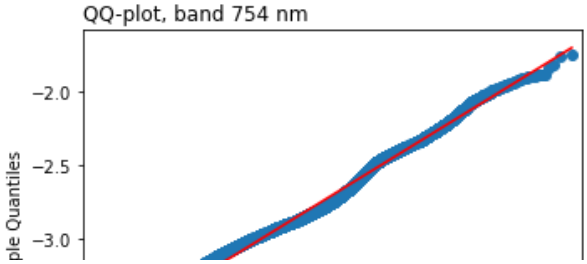
QQ-plot, band 709 nm

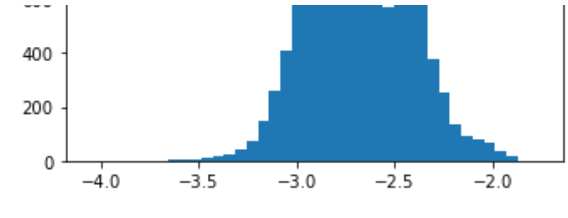
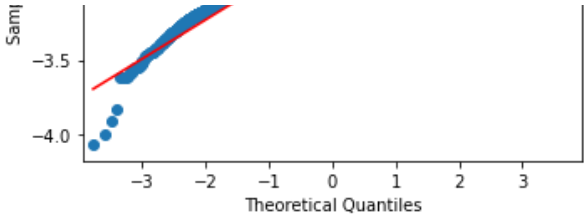


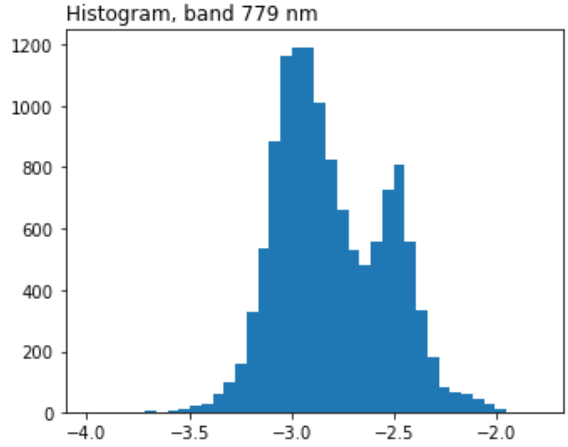
Histogram, band 754 nm

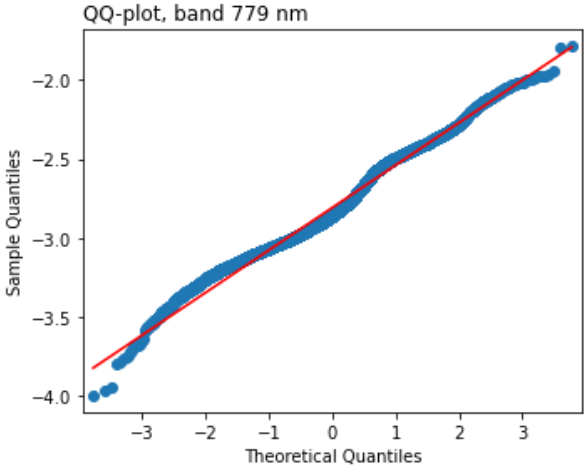


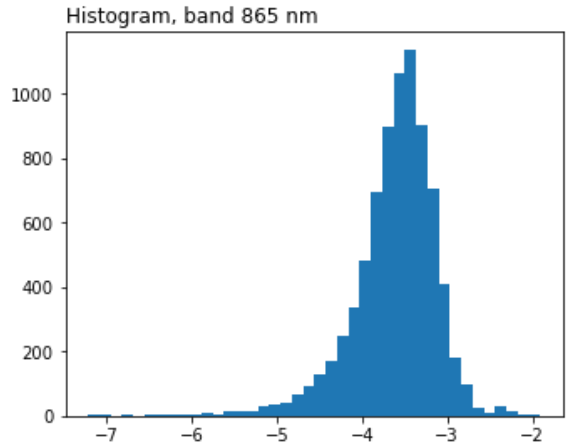
QQ-plot, band 754 nm

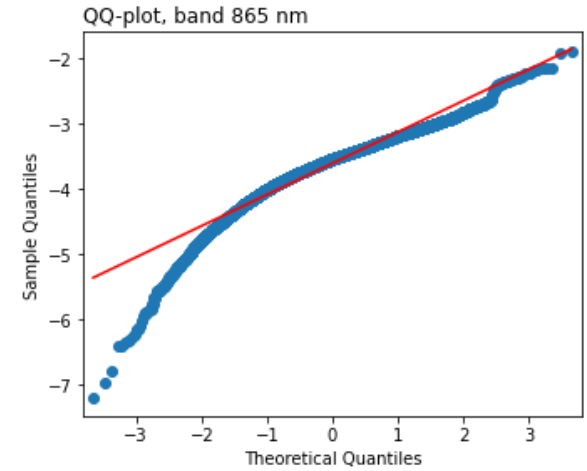


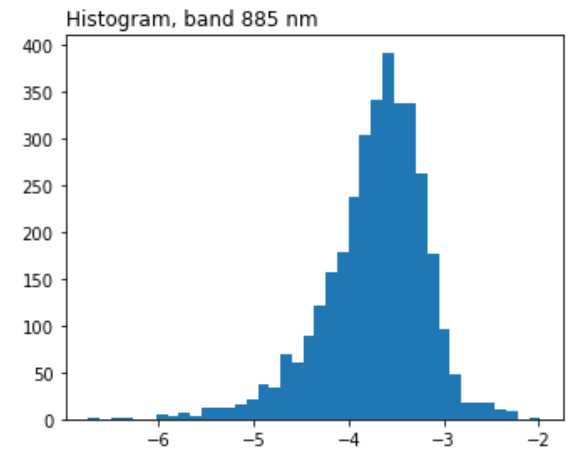



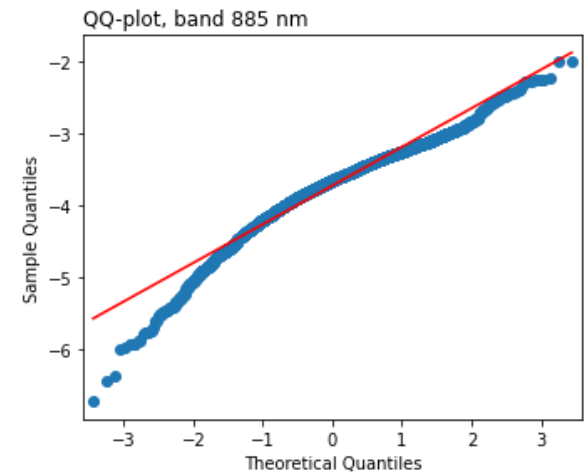












Evaluate the Principle Component Analysis:

In [116]:

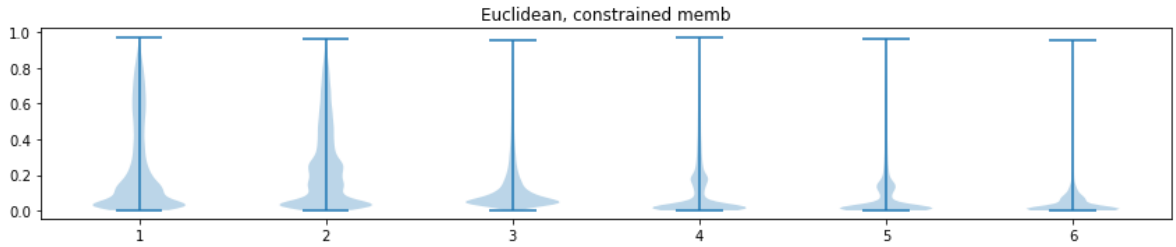
```
df_to_use = np.min(np.argwhere(final_pipeline['pca'].explained_variance_ratio_.cumsu
print(f"Greater than 95% of the variance is contained within the first {df_to_use} c
```

```
Greater than 95% of the variance is contained within the first 4 compo
nents
```

Produce violin plots to demonstrate OWT membership using Euclidean distance

In [125]:

```
# membership histograms
fig, axes = plt.subplots(2,1, figsize=(12,5))
ax1=axes.flatten()[0]

memberships=final_pipeline.predict(samples).T
bp = ax1.violinplot(memberships)
ax1.set_title('Euclidean, constrained memb')

plt.tight_layout()
```

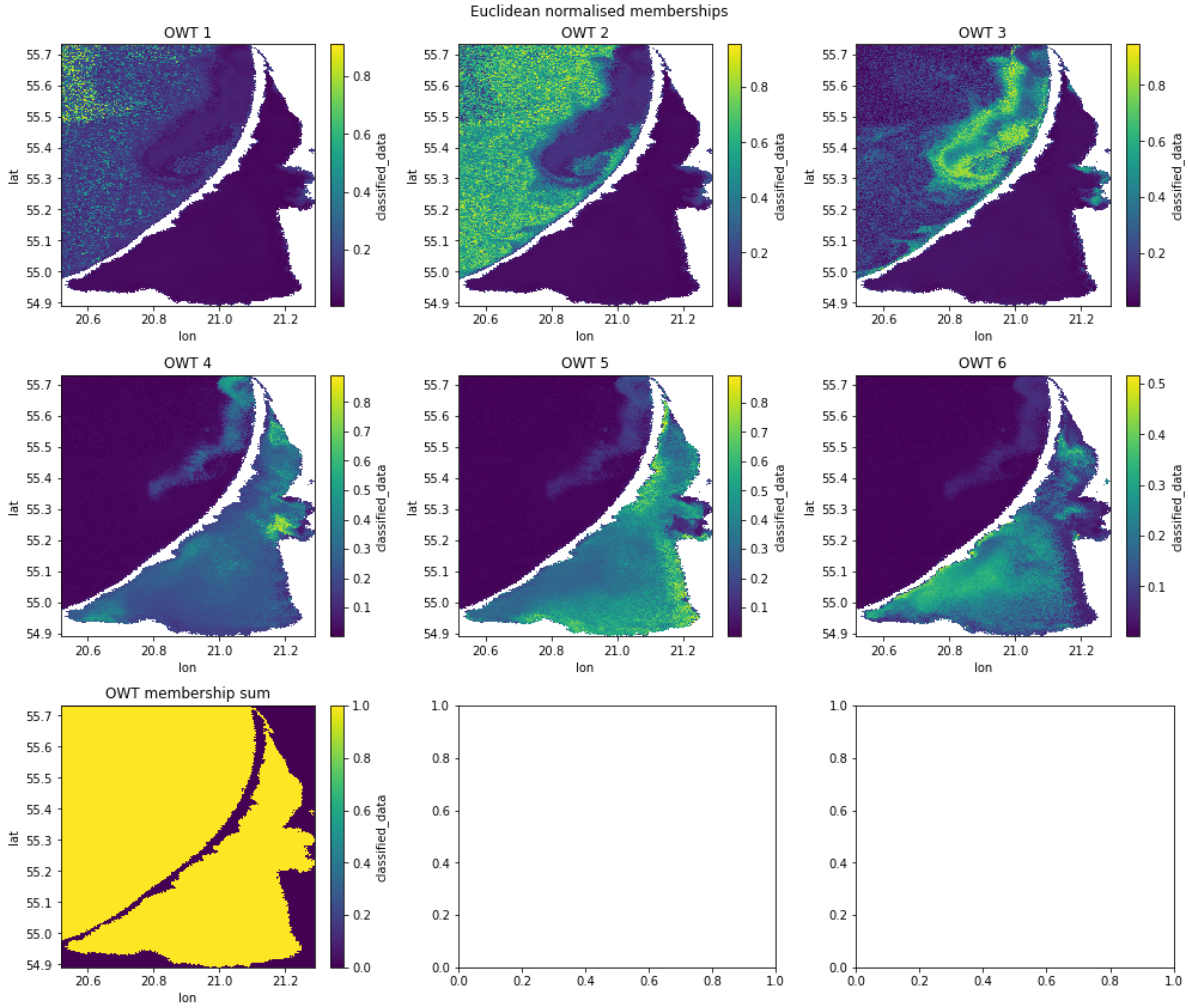


Produce a series of geopgraphic plots to demonstrate the spacial distribution of OWTs

In [128]:

```
shift = 0.015
ds_cleaned = ds.copy()[features]
for var in ds_cleaned.keys():
    ds_cleaned=xr.where(ds_cleaned[var]>-shift, ds_cleaned, np.nan)

euc_mships_file = fwc.predict_file(ds_cleaned, final_pipeline, variables=features, s

n_classes = euc_mships_file.dims['cluster']
class_sums = euc_mships_file.classified_data.sum(axis=0)
plot_rows = int(np.ceil((n_classes+1)/3))
f, axes = plt.subplots(plot_rows, 3, figsize=(14, 4*plot_rows))
for n in range(n_classes):
    ax_now=axes.flatten()[n]
#     (euc_mships_file.classified_data.isel(owt=n)/class_sums).plot(ax=ax_now)
    euc_mships_file.classified_data.isel(cluster=n).plot(ax=ax_now)
    ax_now.title.set_text(f"OWT {n+1}")
ax_now = axes.flatten()[n_classes]
class_sums.plot(ax=ax_now)
ax_now.title.set_text(f"OWT membership sum")
f.suptitle('Euclidean normalised memberships')
f.tight_layout()

plt.show()
```

```
Depreciation warning: predict_file is being replaced by XarrayWrapper
Expected variable names ending with 3 digits
            indicating integer wavelengths in nanometers
            not found. Therefore proceeding with,
            dim = 'variables' instead.
/users/rsg/bac/miniconda3/envs/certo/lib/python3.9/site-packages/sklea
rn/base.py:450: UserWarning: X does not have valid feature names, but
PCA was fitted with feature names
  warnings.warn(
/users/rsg/bac/miniconda3/envs/certo/lib/python3.9/site-packages/sklea
rn/base.py:450: UserWarning: X does not have valid feature names, but
PCA was fitted with feature names
  warnings.warn(
```



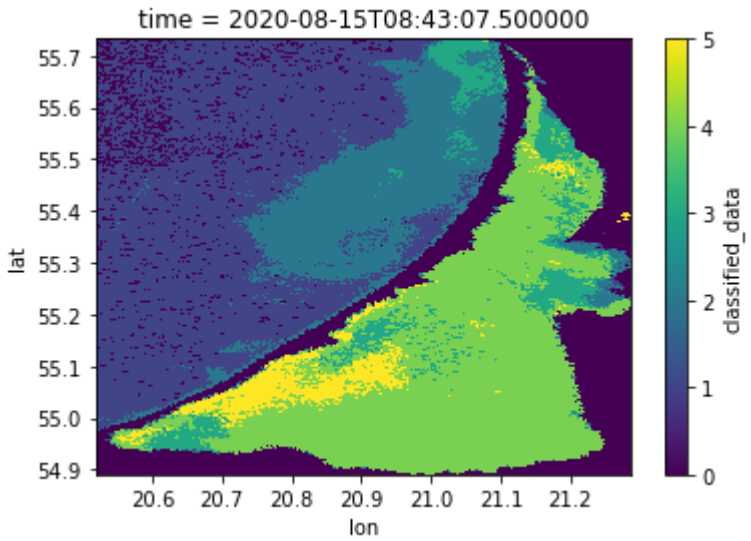
Plot the dominant OWT per pixel

In [129]:

```
euc_mships_file.argmax(dim='cluster', skipna=False)['classified_data'].plot()
```

Out[129]:

```
<matplotlib.collections.QuadMesh at 0x7f2179f1b160>
```



3. Save the results

For each input file used to generate the training dataset, `dflist` , create a new files that with `_OWT_{run_date}` suffixed which contain the OWT features

In [ ]:

```
output_path = f'./'

for fn in np.array(dflist):
    ds = xr.open_dataset(fn)
    output = fwc.predict_file(ds[features], final_pipeline, variables=features)
    output_ds = deepcopy(output)

    output_ds['dominant_OWT'] = 1 + output_ds.argmax(
                                        dim='owt',
                                        skipna=False
                                        ).classified_data.where(
                                                        output_ds.classified_data.is
                                        ).astype('float32')

    for owt in list(output_ds.owt.data):
        output_ds[f"OWT_{owt+1}"] = output_ds.classified_data.isel(owt=owt).astype('
    output_ds = output_ds.drop('owt')
    output_ds = output_ds.drop('classified_data')
    for Rw in features:
        output_ds[Rw] = ds[Rw]
    for attr in list(ds.attrs):
        output_ds.attrs[attr] = ds.attrs[attr]
    output_ds.attrs['project'] = "CERTO"
    output_ds.attrs['contacts'] = "liat@pml.ac.uk"
    output_ds.attrs['title'] = "OWT, CERTO preliminary product; Water Quality, Calim
    output_ds.attrs['credit'] = "OWT and Lake Surface Reflectance were produced by P
    output_ds.attrs['purpose'] = "This product was produced for the EU H2020 CERTO p
    output_ds.attrs['processing_time_OWT'] = str(pd.Timestamp.now())

    # write out to file
    output_netcdf = output_path + fn.split("/")[-1].split(".nc")[0]+f'_OWT_{run_date
    output_ds.to_netcdf(output_netcdf, engine="netcdf4")
    ds.close()
    print(pd.to_datetime(ds.time.item()))
```