ocean-model-skill-assessor

Release 1.1.0

Axiom Data Science

Nov 27, 2023

USER GUIDE

CHAPTER

ONE

INSTALLATION

To install from conda-forge:

```
>>> conda install -c conda-forge ocean-model-skill-assessor
```

To install from PyPI:

```
>>> pip install ocean-model-skill-assessor
```

```
import ocean_model_skill_assessor as omsa
import cf_pandas as cfp
import xroms
```

1.1 How to use `ocean-model-skill-assessor`

... as a Python package. Other notebooks describe its command line interface uses.

But, this is written in parallel to the *CLI demo*, but will be more brief.

There are three steps to follow for a set of model-data validation, which is for one variable:

1. Make a catalog for your model output.
2. Make a catalog for your data.
3. Run the comparison.

These steps will save files into a user application directory cache, along with a log. A project directory can be checked on the command line with `omsa proj_path --project_name PROJECT_NAME`.

```
project_name = "demo_local_package"
```

1.1.1 Make model catalog

We're using example ROMS model output that is available through `xroms` for our model.

```
url = xroms.datasets.CLOVER.fetch("ROMS_example_full_grid.nc")
kwargs = {
    "filenames": [url],
    "skip_entry_metadata": True,
}
```

(continues on next page)

(continued from previous page)

```
cat_model = omsa.main.make_catalog(
                        catalog_type="local",
                        project_name=project_name,
                        catalog_name="model",
                        kwargs=kwargs,
                        return_cat=True,
)
```

```
Downloading file 'ROMS_example_full_grid.nc' from 'https://github.com/xoceanmodel/xroms/
↪raw/main/xroms/data/ROMS_example_full_grid.nc' to '/home/docs/.cache/xroms'.
```

```
cat_model
```

```
model:
  args:
    description: Catalog of type local.
    name: model
  description: Catalog of type local.
  driver: intake.catalog.base.Catalog
  metadata: {}
```

1.1.2 Make data catalog

Set up a catalog of the datasets with which you want to compare your model output. In this example, we use only known data file locations to create our catalog.

Note that we need to include the "featuretype" and "maptype" in the metadata for the data sources. More information can be found on these items in the docs.

```
filenames = ["https://erddap.sensors.axds.co/erddap/tabledap/gov_ornl_cdiac_coastalms_
↪88w_30n.csvp?time%2Clatitude%2Clongitude%2Cz%2Csea_water_temperature&time%3E=2009-11-
↪19T012%3A00%3A00Z&time%3C=2009-11-19T16%3A00%3A00Z",]

cat_data = omsa.make_catalog(project_name="demo_local_package",
                             catalog_type="local",
                             catalog_name="local",
                             kwargs=dict(filenames=filenames),
                             metadata={"featuretype": "timeSeries", "maptype": "point"})
```

```
[2023-11-27 22:03:59,878] {/home/docs/checkouts/readthedocs.org/user_builds/ocean-model-
↪skill-assessor/checkouts/latest/ocean_model_skill_assessor/main.py:216}
WARNING - Dataset gov_ornl_cdiac_coastalms_88w_30n had a timezone UTC which is being␣
↪removed. Make sure the timezone matches the model output.
```

```
cat_data
```

```
local:
  args:
    description: Catalog of type local.
    name: local
```

(continues on next page)

(continued from previous page)

```
  description: Catalog of type local.
  driver: intake.catalog.base.Catalog
  metadata: {}
```

1.1.3 Run comparison

Now that the model output and dataset catalogs are prepared, we can run the comparison of the two.

At this point we need to select a single variable to compare between the model and datasets, and this requires a little extra input. Because we don't know specifics about the format of any given input data file, variables will be interpreted with some flexibility in the form of a set of regular expressions. In the present case, we will compare the water temperature between the model and the datasets (the model output and datasets selected for our catalogs should contain the variable we want to compare). Several sets of regular expressions, called "vocabularies", are available with the package to be used for this purpose, and in this case we will use one called "general" which should match many commonly-used variable names. "general" is selected under `vocab_names`, and the particular key from the general vocabulary that we are comparing is selected with `key`.

See the vocabulary here.

```
paths = omsa.paths.Paths()
cfp.Vocab(paths.VOCAB_PATH("general"))
```

```
{'temp': {'name': '(?i)^(?!.*(air|qc|status|atmospheric|bottom|dew)).*(temp|sst).*'},
↪'salt': {'name': '(?i)^(?!.*(soil|qc|status|bottom)).*(sal|sss).*'}, 'ssh': {'name':
↪'(?i)^(?!.*(qc|status)).*(sea_surface_height|surface_elevation|zeta).*'}, 'u': {'name
↪': 'u$|(?i)(?=.*east)(?=.*vel)'}, 'v': {'name': 'v$|(?i)(?=.*north)(?=.*vel)'}, 'w': {
↪'name': 'w$|(?i)(?=.*up)(?=.*vel)'}, 'water_dir': {'name': '(?i)^(?!.
↪*(qc|status|air|wind))(?=.*dir)(?=.*water)'}, 'water_speed': {'name': '(?i)^(?!.
↪*(qc|status|air|wind))(?=.*speed)(?=.*water)'}, 'wind_dir': {'name': '(?i)^(?!.
↪*(qc|status|water))(?=.*dir)(?=.*wind)'}, 'wind_speed': {'name': '(?i)^(?!.
↪*(qc|status|water))(?=.*speed)(?=.*wind)'}, 'sea_ice_u': {'name': '(?i)^(?!.
↪*(qc|status))(?=.*sea)(?=.*ice)(?=.*u)|(?i)^(?!.*(qc|status))(?=.*sea)(?=.*ice)(?=.
↪*x)(?=.*vel)|(?i)^(?!.*(qc|status))(?=.*sea)(?=.*ice)(?=.*east)(?=.*vel)'}, 'sea_ice_v
↪': {'name': '(?i)^(?!.*(qc|status))(?=.*sea)(?=.*ice)(?=.*v)|(?i)^(?!.*(qc|status))(?=.
↪*sea)(?=.*ice)(?=.*y)(?=.*vel)|(?i)^(?!.*(qc|status))(?=.*sea)(?=.*ice)(?=.*north)(?=.
↪*vel)'}, 'sea_ice_area_fraction': {'name': '(?i)^(?!.*(qc|status))(?=.*sea)(?=.*ice)(?
↪=.*area)(?=.*fraction)'}}
```

Now we run the model-data comparison. Check the API docs for details about the keyword inputs. Also note that the data has filler numbers for this time period which is why the comparison is so far off.

```
omsa.run(project_name="demo_local_package", catalogs=cat_data, model_name=cat_model,
         vocabs="general", key_variable="temp", interpolate_horizontal=False,
         check_in_boundary=False, plot_map=True, dd=5, alpha=20)
```

```
[2023-11-27 22:03:59,917] {/home/docs/checkouts/readthedocs.org/user_builds/ocean-model-
↪skill-assessor/checkouts/latest/ocean_model_skill_assessor/main.py:1803}
INFO - Input parameters: {'catalogs': <Intake catalog: local>, 'project_name': 'demo_
↪local_package', 'key_variable': 'temp', 'model_name': <Intake catalog: model>, 'vocabs
↪': 'general', 'vocab_labels': None, 'ndatasets': None, 'kwargs_map': None, 'verbose':␣
↪True, 'mode': 'w', 'testing': False, 'alpha': 20, 'dd': 5, 'preprocess': False, 'need_
```

(continues on next page)

(continued from previous page)

```
↪xgcm_grid': False, 'xcmocean_options': None, 'kwargs_xroms': None, 'locstream': True,
↪'interpolate_horizontal': False, 'horizontal_interp_code': 'delaunay', 'save_
↪horizontal_interp_weights': True, 'want_vertical_interp': False, 'extrap': False,
↪'model_source_name': None, 'catalog_source_names': None, 'user_min_time': None, 'user_
↪max_time': None, 'check_in_boundary': False, 'tidal_filtering': None, 'ts_mods': None,
↪'model_only': False, 'plot_map': True, 'no_Z': False, 'skip_mask': False, 'wetdry':␣
↪False, 'plot_count_title': True, 'cache_dir': None, 'return_fig': False, 'override_
↪model': False, 'override_processed': False, 'override_stats': False, 'override_plot':␣
↪False, 'plot_description': None, 'kwargs_plot': None, 'skip_key_variable_check': False,
↪ 'kwargs': {}, 'paths': <ocean_model_skill_assessor.paths.Paths object at␣
↪0x7ff9cd41e9a0>, 'logger': <Logger ocean_model_skill_assessor.utils (INFO)>}
```

```
[2023-11-27 22:03:59,925] {/home/docs/checkouts/readthedocs.org/user_builds/ocean-model-
↪skill-assessor/checkouts/latest/ocean_model_skill_assessor/main.py:1838}
INFO - Note that there are 1 datasets to use. This might take awhile.
```

```
[2023-11-27 22:03:59,925] {/home/docs/checkouts/readthedocs.org/user_builds/ocean-model-
↪skill-assessor/checkouts/latest/ocean_model_skill_assessor/main.py:1855}
INFO - Catalog <Intake catalog: local>.
```

```
[2023-11-27 22:03:59,926] {/home/docs/checkouts/readthedocs.org/user_builds/ocean-model-
↪skill-assessor/checkouts/latest/ocean_model_skill_assessor/main.py:1870}
INFO -
source name: gov_ornl_cdiac_coastalms_88w_30n (1 of 1 for catalog <Intake catalog: local>
↪.
```

```
[2023-11-27 22:04:00,080] {/home/docs/checkouts/readthedocs.org/user_builds/ocean-model-
↪skill-assessor/checkouts/latest/ocean_model_skill_assessor/main.py:1069}
INFO -
                        User time range: NaT to NaT.
                        Model time range: 2009-11-19 12:00:00 to 2009-11-19 16:00:00.
                        Data time range: 2009-11-19 12:17:00 to 2009-11-19 15:17:00.
                        Data lon range: -88.6 to -88.6.
                        Data lat range: 30.0 to 30.0.
```

```
[2023-11-27 22:04:00,081] {/home/docs/checkouts/readthedocs.org/user_builds/ocean-model-
↪skill-assessor/checkouts/latest/ocean_model_skill_assessor/main.py:1946}
INFO - running gov_ornl_cdiac_coastalms_88w_30n for key_variable(s) temp from key_
↪variable_list ['temp']
```

```
[2023-11-27 22:04:00,600] {/home/docs/checkouts/readthedocs.org/user_builds/ocean-model-
↪skill-assessor/checkouts/latest/ocean_model_skill_assessor/main.py:878}
INFO - Processed data file name is /home/docs/.cache/ocean-model-skill-assessor/demo_
↪local_package/processed/local_gov_ornl_cdiac_coastalms_88w_30n_temp_data.csv.
```

```
[2023-11-27 22:04:00,600] {/home/docs/checkouts/readthedocs.org/user_builds/ocean-model-
↪skill-assessor/checkouts/latest/ocean_model_skill_assessor/main.py:879}
INFO - Processed model file name is /home/docs/.cache/ocean-model-skill-assessor/demo_
↪local_package/processed/local_gov_ornl_cdiac_coastalms_88w_30n_temp_model.nc.
```

```
[2023-11-27 22:04:00,601] {/home/docs/checkouts/readthedocs.org/user_builds/ocean-model-
↪skill-assessor/checkouts/latest/ocean_model_skill_assessor/main.py:880}
INFO - model file name is /home/docs/.cache/ocean-model-skill-assessor/demo_local_
↪package/model_output/local_gov_ornl_cdiac_coastalms_88w_30n_temp.nc.
```

```
[2023-11-27 22:04:00,602] {/home/docs/checkouts/readthedocs.org/user_builds/ocean-model-
↪skill-assessor/checkouts/latest/ocean_model_skill_assessor/main.py:2004}
INFO - Figure name is /home/docs/.cache/ocean-model-skill-assessor/demo_local_package/
↪out/local_gov_ornl_cdiac_coastalms_88w_30n_temp.png.
```

```
[2023-11-27 22:04:00,603] {/home/docs/checkouts/readthedocs.org/user_builds/ocean-model-
↪skill-assessor/checkouts/latest/ocean_model_skill_assessor/main.py:2025}
INFO - No previously processed model output and data available for gov_ornl_cdiac_
↪coastalms_88w_30n, so setting up now.
```

```
[2023-11-27 22:04:00,608] {/home/docs/checkouts/readthedocs.org/user_builds/ocean-model-
↪skill-assessor/checkouts/latest/ocean_model_skill_assessor/main.py:1363}
INFO - Finding and saving mask to cache to /home/docs/.cache/ocean-model-skill-assessor/
↪demo_local_package/mask_temp.nc.
```

```
[2023-11-27 22:04:00,608] {/home/docs/checkouts/readthedocs.org/user_builds/ocean-model-
↪skill-assessor/checkouts/latest/ocean_model_skill_assessor/utils.py:570}
INFO - Retrieving mask
```

```
[2023-11-27 22:04:04,437] {/home/docs/checkouts/readthedocs.org/user_builds/ocean-model-
↪skill-assessor/checkouts/latest/ocean_model_skill_assessor/main.py:1158}
INFO - Calculating numerical domain boundary.
```

```
[2023-11-27 22:04:04,445] {/home/docs/checkouts/readthedocs.org/user_builds/ocean-model-
↪skill-assessor/checkouts/latest/ocean_model_skill_assessor/main.py:969}
WARNING - Dataset gov_ornl_cdiac_coastalms_88w_30n had a timezone UTC which is being␣
↪removed. Make sure the timezone matches the model output.
```

```
[2023-11-27 22:04:04,528] {/home/docs/checkouts/readthedocs.org/user_builds/ocean-model-
↪skill-assessor/checkouts/latest/ocean_model_skill_assessor/main.py:793}
WARNING - the 'vertical' key cannot be identified in dam by cf-xarray. Maybe you need to␣
↪include the xgcm grid and vertical metrics for xgcm grid, but maybe your variable does␣
↪not have a vertical axis.
```

```
[2023-11-27 22:04:04,537] {/home/docs/checkouts/readthedocs.org/user_builds/ocean-model-
↪skill-assessor/checkouts/latest/ocean_model_skill_assessor/main.py:667}
INFO - Will not perform vertical interpolation and will find nearest depth to 0.0.
```

```
[2023-11-27 22:04:04,539] {/home/docs/checkouts/readthedocs.org/user_builds/ocean-model-
↪skill-assessor/checkouts/latest/ocean_model_skill_assessor/main.py:1452}
INFO - Selecting model output at locations to match dataset gov_ornl_cdiac_coastalms_88w_
↪30n.
```

```
[2023-11-27 22:04:04,596] {/home/docs/checkouts/readthedocs.org/user_builds/ocean-model-
↪skill-assessor/checkouts/latest/ocean_model_skill_assessor/main.py:1484}
```

(continues on next page)

(continued from previous page)

```
INFO -
    Model coordinates found are Coordinates:
    xi_rho      int64 299
    eta_rho     int64 92
    lon_rho     float64 -88.6
    lat_rho     float64 29.97
    s_rho       float64 -0.01667
    npts        int64 0
  * ocean_time  (ocean_time) datetime64[ns] 2009-11-19T12:17:00 2009-11-19T15....

    Output information from finding nearest neighbors to requested points are {'distances
↪': array([0.12069942]), 'eta_rho': array([92]), 'xi_rho': array([299])}.
```

```
[2023-11-27 22:04:04,600] {/home/docs/checkouts/readthedocs.org/user_builds/ocean-model-
↪skill-assessor/checkouts/latest/ocean_model_skill_assessor/main.py:1524}
INFO - Trying to drop vertical coordinates time series
```

```
[2023-11-27 22:04:04,602] {/home/docs/checkouts/readthedocs.org/user_builds/ocean-model-
↪skill-assessor/checkouts/latest/ocean_model_skill_assessor/main.py:1537}
INFO - Loading model output...
```

```
[2023-11-27 22:04:04,805] {/home/docs/checkouts/readthedocs.org/user_builds/ocean-model-
↪skill-assessor/checkouts/latest/ocean_model_skill_assessor/main.py:1644}
INFO - Saving model output to file...
```

```
[2023-11-27 22:04:04,969] {/home/docs/checkouts/readthedocs.org/user_builds/ocean-model-
↪skill-assessor/checkouts/latest/ocean_model_skill_assessor/main.py:2337}
INFO - model file name is /home/docs/.cache/ocean-model-skill-assessor/demo_local_
↪package/model_output/local_gov_ornl_cdiac_coastalms_88w_30n_temp.nc.
```

```
[2023-11-27 22:04:04,970] {/home/docs/checkouts/readthedocs.org/user_builds/ocean-model-
↪skill-assessor/checkouts/latest/ocean_model_skill_assessor/main.py:2339}
INFO - Reading model output from file.
```

```
[2023-11-27 22:04:05,097] {/home/docs/checkouts/readthedocs.org/user_builds/ocean-model-
↪skill-assessor/checkouts/latest/ocean_model_skill_assessor/main.py:2362}
INFO - Calculating stats for temp.
```

```
[2023-11-27 22:04:05,112] {/home/docs/checkouts/readthedocs.org/user_builds/ocean-model-
↪skill-assessor/conda/latest/lib/python3.9/warnings.py:109}
WARNING - /home/docs/checkouts/readthedocs.org/user_builds/ocean-model-skill-assessor/
↪conda/latest/lib/python3.9/site-packages/numpy/lib/function_base.py:2853:␣
↪RuntimeWarning: invalid value encountered in divide
  c /= stddev[:, None]
```

```
[2023-11-27 22:04:05,112] {/home/docs/checkouts/readthedocs.org/user_builds/ocean-model-
↪skill-assessor/conda/latest/lib/python3.9/warnings.py:109}
WARNING - /home/docs/checkouts/readthedocs.org/user_builds/ocean-model-skill-assessor/
↪conda/latest/lib/python3.9/site-packages/numpy/lib/function_base.py:2854:␣
↪RuntimeWarning: invalid value encountered in divide
```

(continues on next page)

(continued from previous page)

```
  c /= stddev[None, :]
```

```
[2023-11-27 22:04:05,118] {/home/docs/checkouts/readthedocs.org/user_builds/ocean-model-
↪skill-assessor/conda/latest/lib/python3.9/warnings.py:109}
WARNING - /home/docs/checkouts/readthedocs.org/user_builds/ocean-model-skill-assessor/
↪checkouts/latest/ocean_model_skill_assessor/stats.py:100: RuntimeWarning: divide by␣
↪zero encountered in double_scalars
  return float(1 - ((obs - model) ** 2).sum() / ((obs - obs_model) ** 2).sum())
```

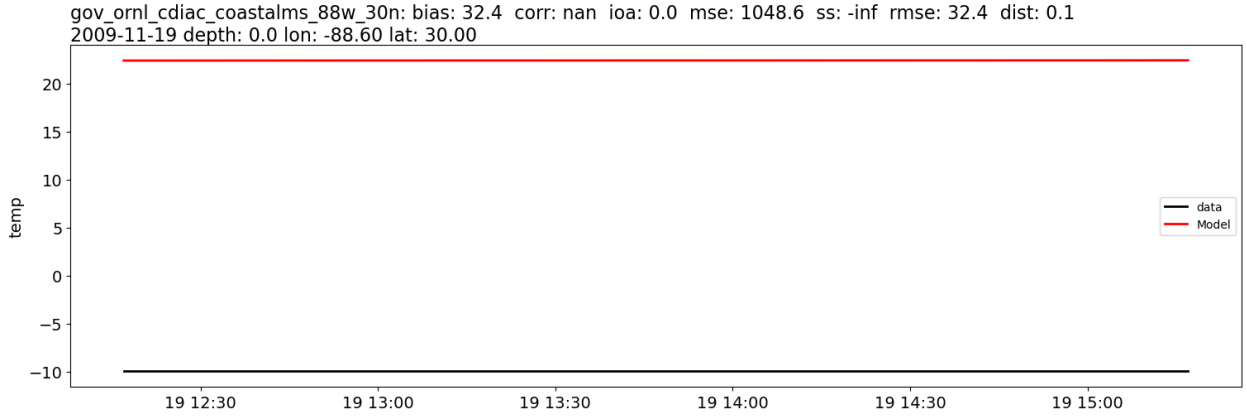
```
[2023-11-27 22:04:05,124] {/home/docs/checkouts/readthedocs.org/user_builds/ocean-model-
↪skill-assessor/checkouts/latest/ocean_model_skill_assessor/main.py:2380}
INFO - Saved stats file.
```

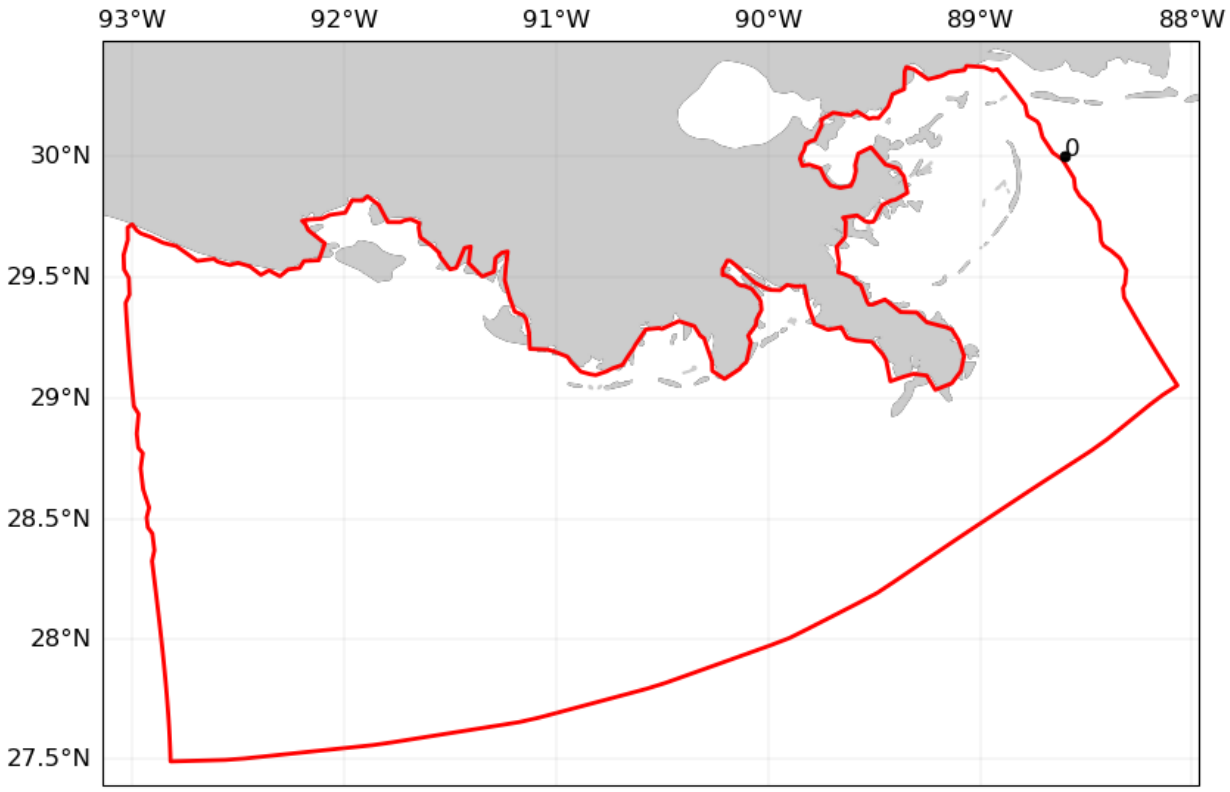
```
[2023-11-27 22:04:06,217] {/home/docs/checkouts/readthedocs.org/user_builds/ocean-model-
↪skill-assessor/checkouts/latest/ocean_model_skill_assessor/main.py:2437}
INFO - Made plot for gov_ornl_cdiac_coastalms_88w_30n
.
```

```
[2023-11-27 22:04:06,372] {/home/docs/checkouts/readthedocs.org/user_builds/ocean-model-
↪skill-assessor/conda/latest/lib/python3.9/warnings.py:109}
WARNING - /home/docs/checkouts/readthedocs.org/user_builds/ocean-model-skill-assessor/
↪conda/latest/lib/python3.9/site-packages/cartopy/io/__init__.py:241: DownloadWarning:␣
↪Downloading: https://naturalearth.s3.amazonaws.com/10m_physical/ne_10m_coastline.zip
  warnings.warn(f'Downloading: {url}', DownloadWarning)
```

```
[2023-11-27 22:04:08,003] {/home/docs/checkouts/readthedocs.org/user_builds/ocean-model-
↪skill-assessor/conda/latest/lib/python3.9/warnings.py:109}
WARNING - /home/docs/checkouts/readthedocs.org/user_builds/ocean-model-skill-assessor/
↪conda/latest/lib/python3.9/site-packages/cartopy/io/__init__.py:241: DownloadWarning:␣
↪Downloading: https://naturalearth.s3.amazonaws.com/10m_physical/ne_10m_land.zip
  warnings.warn(f'Downloading: {url}', DownloadWarning)
```

```
[2023-11-27 22:04:30,682] {/home/docs/checkouts/readthedocs.org/user_builds/ocean-model-
↪skill-assessor/checkouts/latest/ocean_model_skill_assessor/main.py:2451}
INFO - Finished analysis. Find plots, stats summaries, and log in /home/docs/.cache/
↪ocean-model-skill-assessor/demo_local_package.
```





The plots show the time series comparisons for sea water temperatures of the model output and data at one location. Also shown is a map of the Mississippi river delta region where the model is located. An approximation of the numerical domain is shown along with the data location. Note that the comparison is poor because the data is missing for this time period.

1.2 Catalog and dataset set up, NCEI feature type explainer

`ocean-model-skill-assessor` (OMSA) reads datasets from input `Intake` catalogs in order to abstract away the read in process. However, there are a few requirements of and suggestions for these catalogs, which are presented here.

1.2.1 NCEI feature types

The NCEI netCDF feature types are useful because they describe what does and does not fit various definitions of oceanography data types. This defines types of dataset. More information is available in general and for the current NCEI NetCDF Templates 2.0. The following information may be useful for thinking about this and the necessary information below:

1.2.2 Requirements for datasets

Requirements: pandas DataFrames

- `cf-pandas` must be able to identify a single column for each of the following keys:
  - T
  - Z
  - latitude
  - longitude

You can check a Catalog object with `omsa.utils.check_dataframe(df, no_Z)`.

Additionally, the variable you want to compare between model and data must be identifiable in both the dataset and model output using the custom vocabulary and a key in the vocabulary.

1.2.3 Requirements and suggestions for Intake catalogs

Requirements

- Metadata for a dataset must include:
  - an entry for "featuretype" that is a string of the NCEI-defined feature type that describes the dataset. Currently supported are `timeSeries`, `profile`, `trajectoryProfile`, `timeSeriesProfile` (`trajectory` and `grid` still to come).
  - an entry for "maptype" that is how to plot the dataset on a map. Currently supported are "point", "line", and "box".
  - "minLongitude", "maxLongitude", "minLatitude", "maxLatitude"
  - "minTime", "maxTime"

You can check a Catalog object with `omsa.utils.check_catalog(cat)`.

Suggestions

- Do not encode indices for pandas DataFrames. If you do, though, they will be reset in OMSA.
- Note that DataFrames with a column that can be identified by `cf-pandas` as "T" will be parsed as datetimes.

1.2.4 How to make an Intake catalog

- Use an Intake driver that supports direct catalog creation such as `intake-erddap`.
- Use `omsa.main.make_catalog()` or `omsa.main.make_local_catalog()`

1.2.5 How to modify an Intake catalog

- coming soon, to add metadata to existing catalog

1.3 How to make and work with vocabularies and vocab labels

This page demonstrates the workflow of making a new vocabulary, saving it to the user application cache, and reading it back in to use it. The vocabulary created is the exact same as the "general" vocabulary that is saved with the OMSA package, though here it is given another name to demonstrate that you could be making any new vocabulary you want.

Here is the list of variables of interest (with "nickname"), aimed at a physical oceanographer, which are built into the vocabulary:

- water temperature "temp"
- salinity "salt"
- sea surface height "ssh"
- u velocity "u"
- v velocity "v"
- w upward velocity "w"
- direction of water velocity "water_dir"
- magnitude of water velocity "water_speed"
- wind direction "wind_dir"
- wind speed "wind_speed"
- sea ice velocity u "sea_ice_u"
- sea ice velocity v "sea_ice_v"
- sea ice area fraction "sea_ice_area_fraction"

Vocab labels are used in model-data comparison plots to support nice labeling. They are a dictionary with the same keys as the vocabularies being used and the value is the string you want to use for that variable key's label in a plot.

```
import cf_pandas as cfp
import ocean_model_skill_assessor as omsa
import pandas as pd
```

1.3.1 Vocabulary workflow

Make vocabulary

Here we show making the "general" vocabulary that is saved into the repository. This is a more general vocabulary to identify variables from sources that don't use exact CF standard_names.

```
nickname = "temp"
vocab = cfp.Vocab()

# define a regular expression to represent your variable
reg = cfp.Reg(include_or=["temp","sst"], exclude=["air","qc","status","atmospheric",
```

(continues on next page)

(continued from previous page)

```
↪"bottom"])

# Make an entry to add to your vocabulary
vocab.make_entry(nickname, reg.pattern(), attr="name")

vocab.make_entry("salt", cfp.Reg(include_or=["sal","sss"], exclude=["soil","qc","status",
↪"bottom"]).pattern(), attr="name")
vocab.make_entry("ssh", cfp.Reg(include_or=["sea_surface_height","surface_elevation"],␣
↪exclude=["qc","status"]).pattern(), attr="name")

reg = cfp.Reg(include=["east", "vel"])
vocab.make_entry("u", "u$", attr="name")
vocab.make_entry("u", reg.pattern(), attr="name")

reg = cfp.Reg(include=["north", "vel"])
vocab.make_entry("v", "v$", attr="name")
vocab.make_entry("v", reg.pattern(), attr="name")

reg = cfp.Reg(include=["up", "vel"])
vocab.make_entry("w", "w$", attr="name")
vocab.make_entry("w", reg.pattern(), attr="name")

vocab.make_entry("water_dir", cfp.Reg(include=["dir","water"], exclude=["qc","status",
↪"air","wind"]).pattern(), attr="name")

vocab.make_entry("water_speed", cfp.Reg(include=["speed","water"], exclude=["qc","status
↪","air","wind"]).pattern(), attr="name")

vocab.make_entry("wind_dir", cfp.Reg(include=["dir","wind"], exclude=["qc","status",
↪"water"]).pattern(), attr="name")

vocab.make_entry("wind_speed", cfp.Reg(include=["speed","wind"], exclude=["qc","status",
↪"water"]).pattern(), attr="name")

reg1 = cfp.Reg(include=["sea","ice","u"], exclude=["qc","status"])
reg2 = cfp.Reg(include=["sea","ice","x","vel"], exclude=["qc","status"])
reg3 = cfp.Reg(include=["sea","ice","east","vel"], exclude=["qc","status"])
vocab.make_entry("sea_ice_u", reg1.pattern(), attr="name")
vocab.make_entry("sea_ice_u", reg2.pattern(), attr="name")
vocab.make_entry("sea_ice_u", reg3.pattern(), attr="name")

reg1 = cfp.Reg(include=["sea","ice","v"], exclude=["qc","status"])
reg2 = cfp.Reg(include=["sea","ice","y","vel"], exclude=["qc","status"])
reg3 = cfp.Reg(include=["sea","ice","north","vel"], exclude=["qc","status"])
vocab.make_entry("sea_ice_v", reg1.pattern(), attr="name")
vocab.make_entry("sea_ice_v", reg2.pattern(), attr="name")
vocab.make_entry("sea_ice_v", reg3.pattern(), attr="name")

vocab.make_entry("sea_ice_area_fraction", cfp.Reg(include=["sea","ice","area","fraction
↪"], exclude=["qc","status"]).pattern(), attr="name")

vocab
```

```
{'temp': {'name': '(?i)^(?!.*(air|qc|status|atmospheric|bottom)).*(temp|sst).*'}, 'salt
↪': {'name': '(?i)^(?!.*(soil|qc|status|bottom)).*(sal|sss).*'}, 'ssh': {'name': '(?i)^
↪(?!.*(qc|status)).*(sea_surface_height|surface_elevation).*'}, 'u': {'name': 'u$|(?i)(?
↪=.*east)(?=.*vel)'}, 'v': {'name': 'v$|(?i)(?=.*north)(?=.*vel)'}, 'w': {'name': 'w$|(?
↪i)(?=.*up)(?=.*vel)'}, 'water_dir': {'name': '(?i)^(?!.*(qc|status|air|wind))(?=.
↪*dir)(?=.*water)'}, 'water_speed': {'name': '(?i)^(?!.*(qc|status|air|wind))(?=.
↪*speed)(?=.*water)'}, 'wind_dir': {'name': '(?i)^(?!.*(qc|status|water))(?=.*dir)(?=.
↪*wind)'}, 'wind_speed': {'name': '(?i)^(?!.*(qc|status|water))(?=.*speed)(?=.*wind)'},
↪'sea_ice_u': {'name': '(?i)^(?!.*(qc|status))(?=.*sea)(?=.*ice)(?=.*u)|(?i)^(?!.
↪*(qc|status))(?=.*sea)(?=.*ice)(?=.*x)(?=.*vel)|(?i)^(?!.*(qc|status))(?=.*sea)(?=.
↪*ice)(?=.*east)(?=.*vel)'}, 'sea_ice_v': {'name': '(?i)^(?!.*(qc|status))(?=.*sea)(?=.
↪*ice)(?=.*v)|(?i)^(?!.*(qc|status))(?=.*sea)(?=.*ice)(?=.*y)(?=.*vel)|(?i)^(?!.
↪*(qc|status))(?=.*sea)(?=.*ice)(?=.*north)(?=.*vel)'}, 'sea_ice_area_fraction': {'name
↪': '(?i)^(?!.*(qc|status))(?=.*sea)(?=.*ice)(?=.*area)(?=.*fraction)'}}
```

Save it

This exact vocabulary was previously saved as "general" and is available under that name, but this page demonstrates saving a new vocabulary and so we use the name "general2" to differentiate.

```
paths = omsa.paths.Paths()
vocab.save(paths.VOCAB_PATH("general2"))
```

```
paths.VOCAB_PATH("general2")
```

```
PosixPath('/home/docs/.cache/ocean-model-skill-assessor/vocab/general2.json')
```

Use it later

Read the saved vocabulary back in to use it:

```
vocab = cfp.Vocab(paths.VOCAB_PATH("general2"))

df = pd.DataFrame(columns=["sst", "time", "lon", "lat"], data={"sst": [1,2,3]})
with cfp.set_options(custom_criteria=vocab.vocab):
    print(df.cf["temp"])
```

```
0    1
1    2
2    3
Name: sst, dtype: int64
```

1.3.2 Combine vocabularies

A user can add together vocabularies. For example, here we combine the built-in "standard_names" and "general" vocabularies.

```
v1 = cfp.Vocab(paths.VOCAB_PATH("standard_names"))
v2 = cfp.Vocab(paths.VOCAB_PATH("general"))

v = v1 + v2
v
```

```
{'sea_ice_area_fraction': {'name': '(?i)^(?!.*(qc|status))(?=.*sea)(?=.*ice)(?=.*area)(?
↪=.*fraction)', 'standard_name': 'sea_ice_area_fraction$'}, 'water_speed': {'name': '(?
↪i)^(?!.*(qc|status|air|wind))(?=.*speed)(?=.*water)', 'standard_name': 'sea_water_speed
↪$'}, 'salt': {'name': '(?i)^(?!.*(soil|qc|status|bottom)).*(sal|sss).*', 'standard_name
↪': 'sea_surface_salinity$|sea_water_absolute_salinity$|sea_water_practical_salinity
↪$|sea_water_salinity$'}, 'sea_ice_u': {'name': '(?i)^(?!.*(qc|status))(?=.*sea)(?=.
↪*ice)(?=.*u)|(?i)^(?!.*(qc|status))(?=.*sea)(?=.*ice)(?=.*x)(?=.*vel)|(?i)^(?!.
↪*(qc|status))(?=.*sea)(?=.*ice)(?=.*east)(?=.*vel)', 'standard_name': 'eastward_sea_
↪ice_velocity$|sea_ice_x_velocity$'}, 'sea_ice_v': {'name': '(?i)^(?!.*(qc|status))(?=.
↪*sea)(?=.*ice)(?=.*v)|(?i)^(?!.*(qc|status))(?=.*sea)(?=.*ice)(?=.*y)(?=.*vel)|(?i)^(?!
↪.*(qc|status))(?=.*sea)(?=.*ice)(?=.*north)(?=.*vel)', 'standard_name': 'northward_sea_
↪ice_velocity$|sea_ice_y_velocity$'}, 'ssh': {'name': '(?i)^(?!.*(qc|status)).*(sea_
↪surface_height|surface_elevation|zeta).*', 'standard_name': 'sea_surface_elevation|sea_
↪surface_height_above_geoid$|sea_surface_height_above_geopotential_datum$|sea_surface_
↪height_above_mean_sea_level$|sea_surface_height_above_reference_ellipsoid$|surface_
↪height_above_geopotential_datum$|tidal_sea_surface_height_above_lowest_astronomical_
↪tide$|tidal_sea_surface_height_above_mean_higher_high_water$|tidal_sea_surface_height_
↪above_mean_lower_low_water$|tidal_sea_surface_height_above_mean_low_water_springs
↪$|tidal_sea_surface_height_above_mean_sea_level$|water_surface_height_above_reference_
↪datum$|water_surface_reference_datum_altitude$'}, 'wind_speed': {'name': '(?i)^(?!.
↪*(qc|status|water))(?=.*speed)(?=.*wind)', 'standard_name': 'wind_speed$'}, 'water_dir
↪': {'name': '(?i)^(?!.*(qc|status|air|wind))(?=.*dir)(?=.*water)', 'standard_name':
↪'sea_water_velocity_from_direction$|sea_water_velocity_to_direction$'}, 'u': {'name':
↪'u$|(?i)(?=.*east)(?=.*vel)', 'standard_name': 'baroclinic_eastward_sea_water_velocity
↪$|barotropic_eastward_sea_water_velocity$|barotropic_sea_water_x_velocity$|eastward_
↪sea_water_velocity$|eastward_sea_water_velocity_assuming_no_tide$|geostrophic_eastward_
↪sea_water_velocity$|sea_water_x_velocity$|surface_eastward_sea_water_velocity$|surface_
↪geostrophic_eastward_sea_water_velocity$|surface_geostrophic_sea_water_x_velocity$'},
↪'temp': {'name': '(?i)^(?!.*(air|qc|status|atmospheric|bottom|dew)).*(temp|sst).*',
↪'standard_name': 'sea_surface_temperature$|sea_water_potential_temperature$|sea_water_
↪temperature$'}, 'w': {'name': 'w$|(?i)(?=.*up)(?=.*vel)', 'standard_name': 'upward_sea_
↪water_velocity$'}, 'v': {'name': 'v$|(?i)(?=.*north)(?=.*vel)', 'standard_name':
↪'baroclinic_northward_sea_water_velocity$|barotropic_northward_sea_water_velocity
↪$|barotropic_sea_water_y_velocity$|northward_sea_water_velocity$|northward_sea_water_
↪velocity_assuming_no_tide$|northward_sea_water_velocity_due_to_tides$|sea_water_y_
↪velocity$|surface_northward_sea_water_velocity$'}, 'wind_dir': {'name': '(?i)^(?!.
↪*(qc|status|water))(?=.*dir)(?=.*wind)', 'standard_name': 'wind_from_direction$|wind_
↪to_direction$'}}
```

1.3.3 Using the cf-pandas widget

.. raw:: html

1.3.4 Vocab labels

There is a default set of labels in the repository available alongside the default vocabs, called "vocab_labels.json".

You can use `cf-pandas` to open up and look at `vocal_labels` like a vocabulary since they are both just dictionaries stored as json.

```
vocab_labels = cfp.Vocab(paths.VOCAB_PATH("vocab_labels"))
vocab_labels
```

```
{'temp': 'Sea water temperature [C]', 'salt': 'Sea water salinity [psu]', 'ssh': 'Sea␣
↪surface height [m]', 'u': 'x-axis velocity [m/s]', 'v': 'y-axis velocity [m/s]', 'w':
↪'z velocity [m/s]', 'along': 'Along-channel velocity [m/s]', 'across': 'Across-channel␣
↪velocity [m/s]', 'speed': 'Horizontal speed [m/s]', 'east': 'Eastward velocity [m/s]',
↪'north': 'Northward velicity [m/s]', 'water_dir': 'Sea water direction [degrees]',
↪'water_speed': 'Sea water speed [m/s]', 'wind_dir': 'Wind direction [degrees]', 'wind_
↪speed': 'Wind speed [m/s]', 'sea_ice_u': 'Sea ice x-axis velocity [m/s]', 'sea_ice_v':
↪'Sea ice y-axis velocity [m/s]', 'sea_ice_area_fraction': 'Sea ice area fraction []'}
```

1.4 API

main	Main run functions.
utils	Utility functions.
paths	Create paths, and handle file locations and vocabulary files.
stats	Statistics functions.
plot.map	Plot map.
plot.line	Time series plots.
plot.surface	Surface plot.

1.4.1 ocean_model_skill_assessor.main

Main run functions.

Functions

make_catalog(catalog_type, project_name[, ...])	Make a catalog given input selections.
make_local_catalog(filenames[, filetype, ...])	Make an intake catalog from specified data files, including model output locations.
run(catalogs, project_name, key_variable, ...)	Run the model-data comparison.

ocean_model_skill_assessor.main.**_check_prep_narrow_data**(*dd*, *key_variable_data*, *source_name*, *maps*, *vocab*, *user_min_time*, *user_max_time*, *data_min_time*, *data_max_time*, *logger=None*)

Check, prep, and narrow the data in time range.

Parameters

- **dd** (*Union[pd.DataFrame, xr.Dataset]*) – Dataset.
- **key_variable_data** (*str*) – Name of variable to access from dataset.
- **source_name** (*str*) – Name of dataset we are accessing from the catalog.
- **maps** (*list*) – Each entry is a list of information about a dataset; the last entry is for the present source_name or dataset. Each entry contains [min_lon, max_lon, min_lat, max_lat, source_name] and possibly an additional element containing "maptype".
- **vocab** (*Vocab*) – Way to find the criteria to use to map from variable to attributes describing the variable. This is to be used with a key representing what variable to search for.
- **user_min_time** (*pd.Timestamp*) – If this is input, it will be used as the min time for the model. At this point in the code, it will be a pandas Timestamp though could be "NaT" (a null time value).
- **user_max_time** (*pd.Timestamp*) – If this is input, it will be used as the max time for the model. At this point in the code, it will be a pandas Timestamp though could be "NaT" (a null time value).
- **data_min_time** (*pd.Timestamp*) – The min time in the dataset catalog metadata, or if there is a constraint in the metadata such as an ERDDAP catalog allows, and it is more constrained than data_min_time, then the constraint time.
- **data_max_time** (*pd.Timestamp*) – The max time in the dataset catalog metadata, or if there is a constraint in the metadata such as an ERDDAP catalog allows, and it is more constrained than data_max_time, then the constraint time.
- **logger** (*optional*) – logger, by default None

Returns

- dd: data container that has been checked and processed. Will be None if a problem has been detected.
- maps: list of data information. If there was a problem with this dataset, the final entry in *maps* representing the dataset will have been deleted.

Return type
tuple

ocean_model_skill_assessor.main.**_check_time_ranges**(*source_name*, *data_min_time*, *data_max_time*, *model_min_time*, *model_max_time*, *user_min_time*, *user_max_time*, *maps*, *logger=None*)

Compare time ranges in case should skip dataset source_name.

Parameters

- **source_name** (*str*) – Name of dataset we are accessing from the catalog.
- **data_min_time** (*pd.Timestamp*) – The min time in the dataset catalog metadata, or if there is a constraint in the metadata such as an ERDDAP catalog allows, and it is more constrained than data_min_time, then the constraint time.

- **data_max_time** (*pd.Timestamp*) – The max time in the dataset catalog metadata, or if there is a constraint in the metadata such as an ERDDAP catalog allows, and it is more constrained than data_max_time, then the constraint time.
- **user_min_time** (*pd.Timestamp*) – If this is input, it will be used as the min time for the model. At this point in the code, it will be a pandas Timestamp though could be "NaT" (a null time value).
- **user_max_time** (*pd.Timestamp*) – If this is input, it will be used as the max time for the model. At this point in the code, it will be a pandas Timestamp though could be "NaT" (a null time value).
- **model_min_time** (*pd.Timestamp*) – Min model time step
- **model_max_time** (*pd.Timestamp*) – Max model time step
- **maps** (*list*) – Each entry is a list of information about a dataset; the last entry is for the present source_name or dataset. Each entry contains [min_lon, max_lon, min_lat, max_lat, source_name] and possibly an additional element containing "maptype".
- **logger** (*logger, optional*) – Logger for messages.

Returns

- skip_dataset: bool that is True if this dataset should be skipped
- maps: list of dataset information with the final entry (representing the present dataset) removed if skip_dataset is True.

Return type
tuple

ocean_model_skill_assessor.main.**_choose_depths**(*dd*, *model_depth_attr_positive*, *no_Z*, *want_vertical_interp*, *logger=None*)

Determine depths to interpolate to, if any.

This assumes the data container does not have indices, or at least no depth indices.

Parameters

- **dd** (*DataFrame or Dataset*) – Data container
- **model_depth_attr_positive** (*str*) – result of model.cf["Z"].attrs["positive"]: "up" or "down", from model
- **no_Z** (*bool*) – If True, set Z=None so no vertical interpolation or selection occurs. Do this if your variable has no concept of depth, like the sea surface height.
- **want_vertical_interp** (*bool*) – This is False unless the user wants to specify that vertical interpolation should happen. This is used in only certain cases but in those cases it is important so that it is known to interpolate instead of try to figure out a vertical level index (which is not possible currently).
- **logger** (*logger, optional*) – Logger for messages.

Returns

- *dd* – Possibly modified Dataset with sign of depths to match model
- *Z* – Depths to interpolate to with sign that matches the model depths.
- *vertical_interp* – Flag, True if we should interpolate vertically, False if not.

ocean_model_skill_assessor.main.**_dam_from_dsm**(*dsm2*, *key_variable*, *key_variable_data*, *source_metadata*, *no_Z*, *logger=None*)

Select or calculate variable from Dataset.

cf-xarray needs to work for Z, T, longitude, latitude after this

Parameters

- **dsm2** (*Dataset*) – Dataset containing model output. If this is being run from *main*, the model output has already been narrowed to the relevant time range.
- **key_variable** (*str, dict*) – Information to select variable from Dataset. Will be a dict if something needs to be calculated or accessed. In the more simple case will be a string containing the key variable name that can be interpreted with cf-xarray to access the variable of interest from the Dataset.
- **key_variable_data** (*str*) – A string containing the key variable name that can be interpreted with cf-xarray to access the variable of interest from the Dataset.
- **source_metadata** (*dict*) – Metadata for dataset source. Accessed by *cat[source_name].metadata*.
- **no_Z** (*bool*) – If True, set Z=None so no vertical interpolation or selection occurs. Do this if your variable has no concept of depth, like the sea surface height.
- **logger** (*logger, optional*) – Logger for messages.

Returns
Single variable DataArray from Dataset.

Return type
DataArray

ocean_model_skill_assessor.main.**_find_data_time_range**(*cat*, *source_name*)

Determine min and max data times.

Parameters

- **cat** (*Catalog*) – Catalog that contains dataset source_name from which to find data time range.
- **source_name** (*str*) – Name of dataset within cat to examine.

Returns

- **data_min_time** (*pd.Timestamp*) – The min time in the dataset catalog metadata, or if there is a constraint in the metadata such as an ERDDAP catalog allows, and it is more constrained than data_min_time, then the constraint time. If "Z" is present to indicate UTC timezone, it is removed.
- **data_max_time** (*pd.Timestamp*) – The max time in the dataset catalog metadata, or if there is a constraint in the metadata such as an ERDDAP catalog allows, and it is more constrained than data_max_time, then the constraint time. If "Z" is present to indicate UTC timezone, it is removed.

ocean_model_skill_assessor.main.**_initial_model_handling**(*model_name*, *paths*, *model_source_name=None*)

Initial model handling.

cf-xarray needs to be able to identify Z, T, longitude, latitude coming out of here.

Parameters

- **model_name** (*str, Catalog*) – Name of catalog for model output, created with `make_catalog` call, or Catalog instance.
- **paths** (Paths) – Paths object for finding paths to use.
- **model_source_name** (*str, optional*) – Use this to access a specific source in the input model_catalog instead of otherwise just using the first source in the catalog.

Returns
Dataset pointing to model output.

Return type
Dataset

ocean_model_skill_assessor.main.**_is_outside_boundary**(*p1*, *lon*, *lat*, *source_name*, *logger=None*)

Checks point to see if is outside model domain.

This currently assumes that the dataset is fixed in space.

Parameters

- **p1** (*shapely.Polygon*) – Model domain boundary
- **lon** (*float*) – Longitude of point to compare with model domain boundary
- **lat** (*float*) – Latitude of point to compare with model domain boundary
- **source_name** (*str*) – Name of dataset within cat to examine.
- **logger** (*optional*) – logger, by default None

Returns
True if lon, lat point is outside the model domain boundary, otherwise False.

Return type
bool

ocean_model_skill_assessor.main.**_narrow_model_time_range**(*dsm*, *user_min_time*, *user_max_time*, *model_min_time*, *model_max_time*, *data_min_time*, *data_max_time*)

Narrow the model time range to approximately what is needed, to save memory.

If user_min_time and user_max_time were input and are not null values and are narrower than the model time range, use those to control time range.

Otherwise use data_min_time and data_max_time to narrow the time range, but add 1 model timestep on either end to make sure to have extra model output if need to interpolate in that range.

Do not deal with time in detail here since that will happen when the model and data are "aligned" a little later. For now, just return a slice of model times, outside of the extract_model code since not interpolating yet. not dealing with case that data is available before or after model but overlapping rename dsm since it has fewer times now and might need them for the other datasets

Parameters

- **dsm** (*xr.Dataset*) – model dataset
- **user_min_time** (*pd.Timestamp*) – If this is input, it will be used as the min time for the model. At this point in the code, it will be a pandas Timestamp though could be "NaT" (a null time value).
- **user_max_time** (*pd.Timestamp*) – If this is input, it will be used as the max time for the model. At this point in the code, it will be a pandas Timestamp though could be "NaT" (a null time value).

- **model_min_time** (*pd.Timestamp*) – Min model time step
- **model_max_time** (*pd.Timestamp*) – Max model time step
- **data_min_time** (*pd.Timestamp*) – The min time in the dataset catalog metadata, or if there is a constraint in the metadata such as an ERDDAP catalog allows, and it is more constrained than data_min_time, then the constraint time.
- **data_max_time** (*pd.Timestamp*) – The max time in the dataset catalog metadata, or if there is a constraint in the metadata such as an ERDDAP catalog allows, and it is more constrained than data_max_time, then the constraint time.

Returns
Model dataset, but narrowed in time.

Return type
xr.Dataset

ocean_model_skill_assessor.main.**_process_model**(*dsm2*, *preprocess*, *need_xgcm_grid*, *kwargs_xroms*, *logger=None*)

Process model output a second time, possibly.

Parameters

- **dsm2** (*xr.Dataset*) – Model output Dataset, already narrowed in time.
- **preprocess** (*bool*) – True to preprocess.
- **need_xgcm_grid** (*bool*) – True if need to find *xgcm* grid object.
- **kwargs_xroms** (*dict*) – Keyword arguments to pass to xroms.
- **logger** (*optional*) – logger, by default None

Returns

- dsm2: Model output, possibly modified
- grid: xgcm grid object or None
- preprocessed: bool that is True if model output was processed in this function

Return type
tuple

ocean_model_skill_assessor.main.**_processed_file_names**(*fname_processed_orig*, *dfd_type*, *user_min_time*, *user_max_time*, *paths*, *ts_mods*, *logger=None*)

Determine file names for base of stats and figure names and processed data and model names

fname_processed_orig: no info about time modifications fname_processed: fully specific name fname_processed_data: processed data file fname_processed_model: processed model file

Parameters

- **fname_processed_orig** (*str*) – Filename based but without modification if user_min_time and user_max_time were input. Does include info about ts_mods if present.
- **dfd_type** (*type*) – pd.DataFrame or xr.Dataset depending on the data container type.
- **user_min_time** (*pd.Timestamp*) – If this is input, it will be used as the min time for the model. At this point in the code, it will be a pandas Timestamp though could be "NaT" (a null time value).

- **user_max_time** (*pd.Timestamp*) – If this is input, it will be used as the max time for the model. At this point in the code, it will be a pandas Timestamp though could be "NaT" (a null time value).
- **paths** (Paths) – Paths object for finding paths to use.
- **ts_mods** (*list*) – list of time series modifications to apply to data and model. Can be an empty list if no modifications to apply.
- **logger** (*logger, optional*) – Logger for messages.

Returns

- fname_processed: base to be used for stats and figure
- fname_processed_data: file name for processed data
- fname_processed_model: file name for processed model
- model_file_name: (unprocessed) model output

Return type
tuple of *Paths*

ocean_model_skill_assessor.main.**_return_data_locations**(*maps*, *dd*, *featuretype*, *logger=None*)

Return lon, lat locations from dataset.

Parameters

- **maps** (*list*) – Each entry is a list of information about a dataset; the last entry is for the present source_name or dataset. Each entry contains [min_lon, max_lon, min_lat, max_lat, source_name] and possibly an additional element containing "maptype".
- **dd** (*Union[pd.DataFrame, xr.Dataset]*) – Dataset
- **featuretype** (*str*) – NCEI feature type for dataset
- **logger** (*optional*) – logger, by default None

Returns

- lons: float or array of floats
- lats: float or array of floats

Return type
tuple

ocean_model_skill_assessor.main.**_return_mask**(*mask*, *dsm*, *lon_name*, *wetdry*, *key_variable_data*, *paths*, *logger=None*)

Find or calculate and check mask.

Parameters

- **mask** (*xr.DataArray or None*) – Values are 1 for active cells and 0 for inactive grid cells in the model dsm.
- **dsm** (*xr.Dataset*) – Model output Dataset
- **lon_name** (*str*) – variable name for longitude in dsm.
- **wetdry** (*bool*) – Adjusts the logic in the search for mask such that if True, selected mask must include "wetdry" in name and will use first time step.
- **key_variable_data** (*str*) – Key name of variable
- **paths** (Paths) – Paths to files and directories for this project.

- **logger** – optional

Returns
Mask

Return type
DataArray

ocean_model_skill_assessor.main.**_return_p1**(*paths*, *dsm*, *mask*, *alpha*, *dd*, *logger=None*)

Find and return the model domain boundary.

Parameters

- **paths** (Paths) – _description_
- **dsm** (*xr.Dataset*) – _description_
- **mask** (*xr.DataArray or None*) – Values are 1 for active cells and 0 for inactive grid cells in the model dsm.
- **alpha** (*int, optional*) – Number for alphashape to determine what counts as the convex hull. Larger number is more detailed, 1 is a good starting point.
- **dd** (*int, optional*) – Number to decimate model output lon/lat, as a stride.
- **skip_mask** (*bool*) – Allows user to override mask behavior and keep it as None. Good for testing. Default False.
- **logger** (*_type_, optional*) – _description_, by default None

Returns
Model domain boundary

Return type
shapely.Polygon

ocean_model_skill_assessor.main.**_select_process_save_model**(*select_kwargs*, *source_name*, *model_source_name*, *model_file_name*, *save_horizontal_interp_weights*, *key_variable_data*, *maps*, *paths*, *logger=None*)

Select model output, process, and save to file

Parameters

- **select_kwargs** (*dict*) – Keyword arguments to send to *em.select()* for model extraction
- **source_name** (*str*) – Name of dataset within cat to examine.
- **model_source_name** (*str*) – Source name for model in the model catalog
- **model_file_name** (*pathlib.Path*) – Path to where to save model output
- **save_horizontal_interp_weights** (*bool*) – Default True. Whether or not to save horizontal interp info like Delaunay triangulation to file. Set to False to not save which is useful for testing.
- **key_variable_data** (*str*) – Name of variable to select, to be interpreted with cf-xarray
- **maps** (*list*) – Each entry is a list of information about a dataset; the last entry is for the present source_name or dataset. Each entry contains [min_lon, max_lon, min_lat, max_lat, source_name] and possibly an additional element containing "maptype".
- **paths** (Paths) – Paths object for finding paths to use.

- **logger** (*logger, optional*) – Logger for messages.

Returns

- model_var: xr.Dataset with selected model output
- skip_dataset: True if we should skip this dataset due to checks in this function
- maps: Same as input except might be missing final entry if skipping this dataset

Return type
tuple

ocean_model_skill_assessor.main.**make_catalog**(*catalog_type*, *project_name*, *catalog_name=None*, *description=None*, *metadata=None*, *kwargs=None*, *kwargs_search=None*, *kwargs_open=None*, *skip_strings=None*, *vocab=None*, *return_cat=True*, *save_cat=False*, *verbose=True*, *mode='w'*, *testing=False*, *cache_dir=None*)

Make a catalog given input selections.

Parameters

- **catalog_type** (*str*) – Which type of catalog to make? Options are "erddap", "axds", or "local".
- **project_name** (*str*) – Subdirectory in cache dir to store files associated together.
- **catalog_name** (*str, optional*) – Catalog name, with or without suffix of yaml. Otherwise a default name based on the catalog type will be used.
- **description** (*str, optional*) – Description for catalog.
- **metadata** (*dict, optional*) – Catalog metadata.
- **kwargs** (*dict, optional*) – Available keyword arguments for catalog types. Find more information about options in the original docs for each type. Some inputs might be required, depending on the catalog type.
- **kwargs_search** (*dict, optional*) – Keyword arguments to input to search on the server before making the catalog. These are not used with `make_local_catalog()`; only for catalog types "erddap" and "axds". Options are:
  - to search by bounding box: include all of min_lon, max_lon, min_lat, max_lat: (int, float). Longitudes must be between -180 to +180.
  - to search within a datetime range: include both of min_time, max_time: interpretable datetime string, e.g., "2021-1-1"
  - to search using a textual keyword: include *search_for* as a string.
  - model_name can be input in place of either the spatial box or the time range or both in which case those values will be found from the model output. model_name should match a catalog file in the directory described by project_name.
- **kwargs_open** (*dict, optional*) – Keyword arguments to save into local catalog for model to pass on to `xr.open_mfdataset` call or `pandas open_csv`. Only for use with `catalog_type=local`.
- **skip_strings** (*list of strings, optional*) – If provided, source_names in catalog will only be checked for goodness if they do not contain one of skip_strings. For example, if *skip_strings=["_base"]* then any source in the catalog whose name contains that string will be skipped.

- **vocab** (*str, Vocab, Path, optional*) – Way to find the criteria to use to map from variable to attributes describing the variable. This is to be used with a key representing what variable to search for.
- **return_cat** (*bool, optional*) – Return catalog. For when using as a Python package instead of with command line.
- **save_cat** (*bool, optional*) – Save catalog to disk into project directory under *catalog_name*.
- **verbose** (*bool, optional*) – Print useful runtime commands to stdout if True as well as save in log, otherwise silently save in log.
- **mode** (*str, optional*) – mode for logging file. Default is to overwrite an existing logfile, but can be changed to other modes, e.g. "a" to instead append to an existing log file.
- **testing** (*boolean, optional*) – Set to True if testing so warnings come through instead of being logged.
- **cache_dir** (*str, Path*) – Pass on to omsa.paths to set cache directory location if you don't want to use the default. Good for testing.

ocean_model_skill_assessor.main.**make_local_catalog**(*filenames*, *filetype=None*, *name='local_catalog'*, *description='Catalog of user files.'*, *metadata=None*, *metadata_catalog=None*, *skip_entry_metadata=False*, *skip_strings=None*, *kwargs_open=None*, *logger=None*)

Make an intake catalog from specified data files, including model output locations.

Pass keywords for xarray for model output into the catalog through `kwargs_xarray`.

`kwargs_open` and `metadata` must be the same for all filenames. If it is not, make multiple catalogs and you can input them individually into the run command.

Parameters

- **filenames** (*list of paths*) – Where to find dataset(s) from which to make local catalog.
- **filetype** (*str, optional*) – Type of the input filenames, if you don't want the function to try to guess. Must be in the form that can go into intake as f"open_{filetype}".
- **name** (*str, optional*) – Name for catalog.
- **description** (*str, optional*) – Description for catalog.
- **metadata** (*dict, optional*) – Metadata for individual source. If input dataset does not include the longitude and latitude position(s), you will need to include it in the metadata as keys *minLongitude*, *minLatitude*, *maxLongitude*, *maxLatitude*.
- **metadata_catalog** (*dict, optional*) – Metadata for catalog.
- **skip_entry_metadata** (*bool, optional*) – This is useful for testing in which case we don't want to actually read the file. If you are making a catalog file for a model, you may want to set this to *True* to avoid reading it all in for metadata.
- **skip_strings** (*list of strings, optional*) – If provided, source_names in catalog will only be checked for goodness if they do not contain one of skip_strings. For example, if *skip_strings=["_base"]* then any source in the catalog whose name contains that string will be skipped.
- **kwargs_open** (*dict, optional*) – Keyword arguments to pass on to the appropriate `intake` `open_*` call for model or dataset.

Returns
Intake catalog with an entry for each dataset represented by a filename.

Return type
Catalog

Examples

Make catalog to represent local or remote files with specific locations:

```
>>> make_local_catalog([filename1, filename2])
```

Make catalog to represent model output:

```
>>> make_local_catalog([model output location], skip_entry_metadata=True, kwargs_
↪open={"drop_variables": "tau"})
```

ocean_model_skill_assessor.main.**run**(*catalogs*, *project_name*, *key_variable*, *model_name*, *vocabs=None*, *vocab_labels=None*, *ndatasets=None*, *kwargs_map=None*, *verbose=True*, *mode='w'*, *testing=False*, *alpha=5*, *dd=2*, *preprocess=False*, *need_xgcm_grid=False*, *xcmocean_options=None*, *kwargs_xroms=None*, *locstream=True*, *interpolate_horizontal=True*, *horizontal_interp_code='delaunay'*, *save_horizontal_interp_weights=True*, *want_vertical_interp=False*, *extrap=False*, *model_source_name=None*, *catalog_source_names=None*, *user_min_time=None*, *user_max_time=None*, *check_in_boundary=True*, *tidal_filtering=None*, *ts_mods=None*, *model_only=False*, *plot_map=True*, *no_Z=False*, *skip_mask=False*, *wetdry=False*, *plot_count_title=True*, *cache_dir=None*, *return_fig=False*, *override_model=False*, *override_processed=False*, *override_stats=False*, *override_plot=False*, *plot_description=None*, *kwargs_plot=None*, *skip_key_variable_check=False*, ***kwargs*)

Run the model-data comparison.

Note that timezones are assumed to match between the model output and data.

To avoid calculating a mask you need to input *skip_mask=True*, *check_in_boundary=False*, and *plot_map=False*.

Parameters

- **catalogs** (*str, list, Catalog*) – Catalog name(s) or list of names, or catalog object or list of catalog objects. Datasets will be accessed from catalog entries.
- **project_name** (*str*) – Subdirectory in cache dir to store files associated together.
- **key_variable** (*str, dict*) – Key in vocab(s) representing variable to compare between model and datasets.
- **model_name** (*str, Catalog*) – Name of catalog for model output, created with `make_catalog` call, or Catalog instance.
- **vocabs** (*str, list, Vocab, PurePath, optional*) – Criteria to use to map from variable to attributes describing the variable. This is to be used with a key representing what variable to search for. This input is for the name of one or more existing vocabularies which are stored in a user application cache. This should be supplied, however it is made optional because it could be provided by setting it outside of the OMSA code.

- **vocab_labels** (*dict, optional*) – Ultimately a dictionary whose keys match the input vocab and values have strings to be used in plot labels, such as "Sea water temperature [C]" for the key "temp". They can be input from a stored file or as a dict.
- **ndatasets** (*int, optional*) – Max number of datasets from each input catalog to use.
- **kwargs_map** (*dict, optional*) – Keyword arguments to pass on to `omsa.plot.map.plot_map` call.
- **verbose** (*bool, optional*) – Print useful runtime commands to stdout if True as well as save in log, otherwise silently save in log.
- **mode** (*str, optional*) – mode for logging file. Default is to overwrite an existing logfile, but can be changed to other modes, e.g. "a" to instead append to an existing log file.
- **testing** (*boolean, optional*) – Set to True if testing so warnings come through instead of being logged.
- **alpha** (*int*) – parameter for alphashape. 0 returns qhull, and higher values make a tighter polygon around the points.
- **dd** (*int*) – number to decimate model points by when calculating model boundary with alphashape. input 1 to not decimate.
- **preprocess** (*bool, optional*) – If True, use function from `extract_model` to preprocess model output.
- **need_xgcm_grid** (*bool*) – If True, try to set up xgcm grid for run, which will be used for the variable calculation for the model.
- **kwargs_xroms** (*dict*) – Optional keyword arguments to pass to xroms.open_dataset
- **locstream** (*boolean, optional*) – Which type of interpolation to do, passed to em.select():
  - False: 2D array of points with 1 dimension the lons and the other dimension the lats.
  - True: lons/lats as unstructured coordinate pairs (in xESMF language, LocStream).
- **interpolate_horizontal** (*bool, optional*) – If True, interpolate horizontally. Otherwise find nearest model points.
- **horizontal_interp_code** (*str*) – Default "xesmf" to use package `xESMF` for horizontal interpolation, which is probably better if you need to interpolate to many points. To use `xESMF` you have install it as an optional dependency. Input "tree" to use BallTree to find nearest 3 neighbors and interpolate using barycentric coordinates. This has been tested for interpolating to 3 locations so far. Input "delaunay" to use a delaunay triangulation to find the nearest triangle points and interpolate the same as with "tree" using barycentric coordinates. This should be faster when you have more points to interpolate to, especially if you save and reuse the triangulation.
- **save_horizontal_interp_weights** (*bool*) – Default True. Whether or not to save horizontal interp info like Delaunay triangulation to file. Set to False to not save which is useful for testing.
- **want_vertical_interp** (*bool*) – This is False unless the user wants to specify that vertical interpolation should happen. This is used in only certain cases but in those cases it is important so that it is known to interpolate instead of try to figure out a vertical level index (which is not possible currently).
- **extrap** (*bool*) – Passed to *extract_model.select()*. Defaults to False. Pass True to extrapolate outside the model domain.

- **model_source_name** (*str, optional*) – Use this to access a specific source in the input model_catalog instead of otherwise just using the first source in the catalog.
- **catalog_source_names** –
- **user_min_time** (*str, optional*) – If this is input, it will be used as the min time for the model
- **user_max_time** (*str, optional*) – If this is input, it will be used as the max time for the model
- **check_in_boundary** (*bool*) – If True, station location will be compared against model domain polygon to check if inside domain. Set to False to skip this check which might be desirable if you want to just compare with the closest model point.
- **tidal_filtering** (*dict,*) – `tidal_filtering["model"]=True` to tidally filter modeling output after em.select() is run, and `tidal_filtering["data"]=True` to tidally filter data.
- **ts_mods** (*list*) – list of time series modifications to apply to data and model.
- **model_only** (*bool*) – If True, reads in model output and saves to cache, then stops. Default False.
- **plot_map** (*bool*) – If False, don't plot map
- **no_Z** (*bool*) – If True, set Z=None so no vertical interpolation or selection occurs. Do this if your variable has no concept of depth, like the sea surface height.
- **skip_mask** (*bool*) – Allows user to override mask behavior and keep it as None. Good for testing. Default False. Also skips mask in p1 calculation and map plotting if set to False and those are set to True.
- **wetdry** (*bool*) – If True, insist that masked used has "wetdry" in the name and then use the first time step of that mask.
- **plot_count_title** (*bool*) – If True, have a count to match the map of the station number in the title, like "0: [station name]". Otherwise skip count.
- **cache_dir** (*str, Path*) – Pass on to omsa.paths to set cache directory location if you don't want to use the default. Good for testing.
- **vocab_labels** – dict with keys that match input vocab for putting labels with units on the plots. User has to make sure they match both the data and model; there is no unit handling.
- **return_fig** (*bool*) – Set to True to return all outputs from this function. Use for testing. Only works if using a single source.
- **override_model** (*bool*) – Flag to force-redo model selection. Default False.
- **override_processed** (*bool*) – Flag to force-redo model and data processing. Default False.
- **override_stats** (*bool*) – Flag to force-redo stats calculation. Default False.
- **override_plot** (*bool*) – Flag to force-redo plot. If True, only redos plot itself if other files are already available. If False, only redos the plot not the other files. Default False.
- **kwargs_plot** (*dict*) – to pass to omsa plot selection and then through the omsa plot selection to the subsequent plot itself for source. If you need more fine options, run the run function per source.
- **skip_key_variable_check** (*bool*) – If True, don't check for key_variable name being in catalog source metadata.

1.4.2 ocean_model_skill_assessor.utils

Utility functions.

Functions

calculate_anomaly(dd_in, monthly_mean[, varname])	Given monthly mean that is indexed by month of year, subtract it from time series to get anomaly.
calculate_distance(lons, lats)	Calculate distance (km), esp for transects.
check_catalog(cat[, source_names, skip_strings])	Check a catalog for required keys.
check_dataframe(dfd, no_Z)	Check dataframe for T, Z, lon, lat; reset indices; parse dates.
check_dataset(ds[, is_model, no_Z])	Check xarray datasets (usually model output) for necessary cf-xarray dims/coords.
coords1Dto2D(dam)	expand 1D coordinates to 2D
find_bbox(ds[, paths, mask, dd, alpha, save])	Determine bounds and boundary of model.
fix_dataset(model_var, ds)	Fill in info necessary to pass *check_dataset()* if possible.
get_mask(dsm, varname[, wetdry])	Return mask that matches x/y coords of var.
kwargs_search_from_model(kwargs_search, paths)	Adds spatial and/or temporal range from model output to dict.
open_catalogs(catalogs[, paths, skip_check, ...])	Initialize catalog objects from inputs.
open_vocab_labels(vocab_labels[, paths])	Open dict of vocab_labels if needed
open_vocabs(vocabs[, paths])	Open vocabularies, can input mix of forms.
read_model_file(fname_processed_model, no_Z, dsm)	_summary_
read_processed_data_file(...)	_summary_
save_processed_files(dfd, ...)	Save processed data and model output into files.
set_up_logging(verbose, paths[, mode, testing])	set up logging
shift_longitudes(dam)	Shift longitudes from 0 to 360 to -180 to 180 if necessary.

ocean_model_skill_assessor.utils.**calculate_anomaly**(*dd_in*, *monthly_mean*, *varname=None*)

Given monthly mean that is indexed by month of year, subtract it from time series to get anomaly.

Should work with both pd.Series/pd.DataFrame and xr. DataArray. Assume that variable in monthly_mean is the same as in the input time series. The way it works for DataArrays is by changing it to a DataFrame. Assumes this is a time series.

Returns dd as the type as DataFrame it is came in as Series and Dataset if it came in DataArray. It is pd.Series in the middle so this probably won't work well for datasets that are more complex than time series.

ocean_model_skill_assessor.utils.**calculate_distance**(*lons*, *lats*)

Calculate distance (km), esp for transects.

ocean_model_skill_assessor.utils.**check_catalog**(*cat*, *source_names=None*, *skip_strings=None*)

Check a catalog for required keys.

Parameters

- **catalogs** (*Catalog*) – Catalog object
- **source_names** (*list*) – Use these source_names instead of list(cat) if entered, for checking.
- **skip_strings** (*list of strings, optional*) – If provided, source_names in catalog will only be checked for goodness if they do not contain one of skip_strings. For example, if

skip_strings=["_base"] then any source in the catalog whose name contains that string will be skipped.

ocean_model_skill_assessor.utils.**check_dataframe**(*dfd*, *no_Z*)

Check dataframe for T, Z, lon, lat; reset indices; parse dates.

ocean_model_skill_assessor.utils.**check_dataset**(*ds*, *is_model=True*, *no_Z=False*)

Check xarray datasets (usually model output) for necessary cf-xarray dims/coords.

If Dataset is model output (*is_model=True*), must have T, Z, vertical, latitude, longitude, and "positive" attribute must be associated with Z or vertical. But, if *no_Z=True*, neither Z, vertical, nor positive attribute need to be present.

If Dataset is not model output (is_model=False), must have T, Z, latitude, longitude. But, if *no_Z=True*, Z does not need to be present.

ocean_model_skill_assessor.utils.**coords1Dto2D**(*dam*)

expand 1D coordinates to 2D

Parameters
dam (*DataArray*) – Model output variable to work on.

Returns
Model output but with 2D coordinates in place of 1D coordinates, if applicable. Otherwise same as input.

Return type
DataArray

ocean_model_skill_assessor.utils.**find_bbox**(*ds*, *paths=None*, *mask=None*, *dd=1*, *alpha=5*, *save=False*)

Determine bounds and boundary of model.

This does not know how to handle a rectilinear 1D lon/lat model with a mask

Parameters

- **ds** (*DataArray*) – xarray Dataset containing model output.
- **paths** (Paths) – Paths object for finding paths to use.
- **mask** (*DataArray, optional*) – Mask with 1's for active locations and 0's for masked.
- **dd** (*int, optional*) – Number to decimate model output lon/lat, as a stride.
- **alpha** (*int, optional*) – Number for alphashape to determine what counts as the convex hull. Larger number is more detailed, 1 is a good starting point.
- **save** (*bool, optional*) – Input True to save.

Returns
Contains the name of the longitude and latitude variables for ds, geographic bounding box of model output (*[min_lon, min_lat, max_lon, max_lat]*), low res and high res wkt representation of model boundary.

Return type
List

Notes

This was originally from the package `model_catalogs`.

ocean_model_skill_assessor.utils.**fix_dataset**(*model_var*, *ds*)

Fill in info necessary to pass *check_dataset()* if possible.

Right now it is only for converting horizontal indices to lon/lat but conceivably could do more in the future. Looks for lon/lat being 2D coords.

Parameters

- **model_var** (*Union[xr.DataArray,xr.Dataset]*) – xarray object that needs some more info filled in
- **ds** (*Union[xr.DataArray,xr.Dataset]*) – xarray object that has info that can be used to fill in model_var

Returns
model_var with more information included, hopefully

Return type
Union[xr.DataArray,xr.Dataset]

ocean_model_skill_assessor.utils.**get_mask**(*dsm*, *varname*, *wetdry=False*)

Return mask that matches x/y coords of var.

If no mask can be identified with *.filter_by_attrs(flag_meanings="land water")*, instead will make one of non-nans for 1 horizontal grid cross-section of varname.

Parameters

- **dsm** (*Dataset*) – Model output
- **varname** (*str*) – Name of variable in dsm.
- **wetdry** (*bool*) – If True, selected mask must include "wetdry" in name and will use first time step.

Returns
mask associated with varname in dsm

Return type
DataArray

ocean_model_skill_assessor.utils.**kwargs_search_from_model**(*kwargs_search*, *paths*)

Adds spatial and/or temporal range from model output to dict.

Examines model output and uses the bounding box of the model as the search spatial range if needed, and the time range of the model as the search time search if needed. They are added into *kwargs_search* and the dict is returned.

Parameters

- **kwargs_search** (*dict*) – Keyword arguments to input to search on the server before making the catalog.
- **paths** (Paths) – Paths object for finding paths to use.

Returns
kwargs_search but with modifications if relevant.

Return type
dict

Raises
KeyError – If all of *max_lon*, *min_lon*, *max_lat*, *min_lat* and *min_time*, *max_time* are already specified along with *model_name*.

ocean_model_skill_assessor.utils.**open_catalogs**(*catalogs*, *paths=None*, *skip_check=False*, *skip_strings=None*)

Initialize catalog objects from inputs.

Parameters

- **catalogs** (*Union[str, Catalog, Sequence]*) – Catalog name(s) or list of names, or catalog object or list of catalog objects.
- **paths** (Paths, *optional*) – Paths object for finding paths to use. Required if any catalog is a string referencing paths.
- **skip_check** (*bool*) – If True, do not check catalogs. Use this for testing as needed. Default is False.
- **skip_strings** (*list of strings, optional*) – If provided, source_names in catalog will only be checked for goodness if they do not contain one of skip_strings. For example, if *skip_strings=["_base"]* then any source in the catalog whose name contains that string will be skipped.

Returns
Catalogs, ready to use.

Return type
list[Catalog]

ocean_model_skill_assessor.utils.**open_vocab_labels**(*vocab_labels*, *paths=None*)

Open dict of vocab_labels if needed

Parameters

- **vocab_labels** (*Union[str, Vocab, Sequence, Path], optional*) – Criteria to use to map from variable to attributes describing the variable. This is to be used with a key representing what variable to search for. This input is for the name of one or more existing vocabularies which are stored in a user application cache.
- **paths** (Paths, *optional*) – Paths object for finding paths to use.

Returns
dict of vocab_labels for plotting

Return type
dict

ocean_model_skill_assessor.utils.**open_vocabs**(*vocabs*, *paths=None*)

Open vocabularies, can input mix of forms.

Parameters

- **vocabs** (*Union[str, Vocab, Sequence, Path]*) – Criteria to use to map from variable to attributes describing the variable. This is to be used with a key representing what variable to search for. This input is for the name of one or more existing vocabularies which are stored in a user application cache.
- **paths** (Paths, *optional*) – Paths object for finding paths to use. Required if any input vocab is a str referencing paths.

Returns
Single Vocab object with vocab stored in vocab.vocab

Return type
Vocab

ocean_model_skill_assessor.utils.**read_model_file**(*fname_processed_model*, *no_Z*, *dsm*)

summary

Parameters

- **fname_processed_model** (*Path*) – Model file path
- **no_Z** (*bool*) – _description_
- **dsm** (*Dataset*) –

Return type
Processed model output (Dataset)

ocean_model_skill_assessor.utils.**read_processed_data_file**(*fname_processed_data*, *no_Z*)

summary

Parameters

- **fname_processed_data** (*Path*) – Data file path
- **no_Z** (*bool*) – _description_

Return type
Processed data (DataFrame or Dataset)

ocean_model_skill_assessor.utils.**save_processed_files**(*dfd*, *fname_processed_data*, *model_var*, *fname_processed_model*)

Save processed data and model output into files.

Parameters

- **dfd** (*Union[xr.Dataset, pd.DataFrame]*) – Processed data
- **fname_processed_data** (*Path*) – Data file path
- **model_var** (*xr.Dataset*) – Processed model output
- **fname_processed_model** (*Path*) – Model file path

ocean_model_skill_assessor.utils.**set_up_logging**(*verbose*, *paths*, *mode='w'*, *testing=False*)

set up logging

ocean_model_skill_assessor.utils.**shift_longitudes**(*dam*)

Shift longitudes from 0 to 360 to -180 to 180 if necessary.

Parameters
dam (*Union[DataArray,Dataset]*) – Object with model output to check

Returns
Return model output with shifted longitudes, if it was necessary.

Return type
Union[DataArray,Dataset]

1.4.3 ocean_model_skill_assessor.paths

Create paths, and handle file locations and vocabulary files.

Classes

Paths([project_name, cache_dir])	Object to manage paths

class ocean_model_skill_assessor.paths.**Paths**(*project_name=None*, *cache_dir=None*)

Bases: object

Object to manage paths

Attributes

ALPHA_PATH
Return path to alphashape polygon.

LOG_PATH
Return path to vocab.

MODEL_CACHE_DIR
Return path to model cache directory.

OUT_DIR
Return path to output directory.

PROCESSED_CACHE_DIR
Return path to processed data-model directory.

PROJ_DIR
Return path to project directory.

VOCAB_DIR
Where to store and find vocabularies.

Methods

CAT_PATH(cat_name)	Return path to catalog.
MASK_PATH(key_variable)	Return path to mask cache for key_variable.
VOCAB_PATH(vocab_name)	Return path to vocab.

property ALPHA_PATH

Return path to alphashape polygon.

CAT_PATH(*cat_name*)

Return path to catalog.

property LOG_PATH

Return path to vocab.

MASK_PATH(*key_variable*)

Return path to mask cache for key_variable.

property MODEL_CACHE_DIR

Return path to model cache directory.

property OUT_DIR

Return path to output directory.

property PROCESSED_CACHE_DIR

Return path to processed data-model directory.

property PROJ_DIR

Return path to project directory.

property VOCAB_DIR

Where to store and find vocabularies. Come from an initial set.

VOCAB_PATH(*vocab_name*)

Return path to vocab.

1.4.4 ocean_model_skill_assessor.stats

Statistics functions.

Functions

compute_bias(obs, model)	Given obs and model signals return bias.
compute_correlation_coefficient(obs, model)	Given obs and model signals, return Pearson product-moment correlation coefficient
compute_descriptive_statistics(model[, ddof])	Given obs and model signals, return the standard deviation
compute_index_of_agreement(obs, model)	Given obs and model signals, return Index of Agreement (Willmott 1981)
compute_mean_square_error(obs, model[, centered])	Given obs and model signals, return mean squared error (MSE)
compute_murphy_skill_score(obs, model[, ...])	Given obs and model signals, return Murphy Skill Score (Murphy 1988)
compute_root_mean_square_error(obs, model[, ...])	Given obs and model signals, return Root Mean Square Error (RMSE)
compute_stats(obs, model)	Compute stats and return as DataFrame
save_stats(source_name, stats, key_variable, ...)	Save computed stats to file.

ocean_model_skill_assessor.stats.**compute_bias**(*obs*, *model*)

Given obs and model signals return bias.

ocean_model_skill_assessor.stats.**compute_correlation_coefficient**(*obs*, *model*)

Given obs and model signals, return Pearson product-moment correlation coefficient

ocean_model_skill_assessor.stats.**compute_descriptive_statistics**(*model*, *ddof=0*)

Given obs and model signals, return the standard deviation

ocean_model_skill_assessor.stats.**compute_index_of_agreement**(*obs*, *model*)

Given obs and model signals, return Index of Agreement (Willmott 1981)

ocean_model_skill_assessor.stats.**compute_mean_square_error**(*obs*, *model*, *centered=False*)

Given obs and model signals, return mean squared error (MSE)

ocean_model_skill_assessor.stats.**compute_murphy_skill_score**(*obs*, *model*, *obs_model=None*)

Given obs and model signals, return Murphy Skill Score (Murphy 1988)

ocean_model_skill_assessor.stats.**compute_root_mean_square_error**(*obs*, *model*, *centered=False*)

Given obs and model signals, return Root Mean Square Error (RMSE)

ocean_model_skill_assessor.stats.**compute_stats**(*obs*, *model*)

Compute stats and return as DataFrame

ocean_model_skill_assessor.stats.**save_stats**(*source_name*, *stats*, *key_variable*, *paths*, *filename=None*)

Save computed stats to file.

1.4.5 ocean_model_skill_assessor.plot.map

Plot map.

Functions

plot_cat_on_map(catalog, paths[, ...])	Plot catalog on map with optional model domain polygon.
plot_map(maps, figname[, extent, p, ...])	Plot and save to file map of model domain and data locations.
setup_ax(ax[, left_labels, right_labels, ...])	Basic plot setup for map.

ocean_model_skill_assessor.plot.map.**plot_cat_on_map**(*catalog*, *paths*, *source_names=None*, *figname=None*, *remove_duplicates=None*, ***kwargs_map*)

Plot catalog on map with optional model domain polygon.

Parameters

- **catalog** (*Union[Catalog,str]*) – Which catalog of datasets to plot on map.
- **paths** (Paths) – Paths object for finding paths to use.
- **source_names** (*list*) – Use these list names instead of list(cat) if input.
- **remove_duplicates** (*function, optional*) – If True, take the set of the source in catalog based on the spatial locations so they are not repeated in the map.
- **remove_duplicates** – Input a function that takes in maps and return maps, and in between removes duplicate entries.

Examples

After creating catalog with *intake-erddap*, look at data locations:

```
>>> omsa.plot.map.plot_cat_on_map(catalog=catalog_name, project_name=project_name)
```

ocean_model_skill_assessor.plot.map.**plot_map**(*maps*, *figname*, *extent=None*, *p=None*, *label_with_station_name=False*, *dd=[0.0, 0.0]*, *annotate=True*, *annotate_fontsize=12*, *figsize=(8, 7)*, *two_maps=None*, *map_font_size=12*, *markersize=5*, *markeredgewidth=0.5*, *linewidth_data=3*, *linewidth_poly=2*, *alpha_marker=1.0*, *colors_data='k'*, *legend=False*, *loc='best'*, *suptitle=None*, *suptitle_fontsize=16*, *tight_layout=True*)

Plot and save to file map of model domain and data locations.

Parameters

- **maps** (*array*) – Info about datasets. [min_lon, max_lon, min_lat, max_lat, source_name]. Can have optional 6th element in the list for what type of representation to use in the plot: "point", "box", or "line". If the type isn't input and maxlons don't equal minlon, assume box to plot instead of line.
- **figname** (*Union[str, PurePath]*) – Map will be saved here.
- **extent** (*optional*) – [min longitude, max longitude, min latitude, max latitude]
- **p** (*Shapely polygon*) – Polygon representing outer boundary of numerical model.
- **label_with_station_name** (*bool*) – If True, use station names to label instead of a counter from 0 to number of stations - 1.
- **dd** (*list*) – Distance [x,y] to push the annotation away from the min lon/lat of a station. To input per station, make list of lists that is the same size as the number of stations.
- **annotate** (*bool*) – True to annotate.
- **annotate_fontsize** (*int*) – Fontsize for annotations
- **figsize** (*tuple*) – Figure size for matplotlib
- **two_maps** (*dict*) – Plot two maps side by side: presumably a zoomed-out version on the left ("extent_left") with a box showing the area enclosed by the magnified map on the right ("extent_right"). The data locations are only plotted in the right-hand map. You can also input "width_ratios" to shift the width between the map and the magnified map.

  Example usage: `two_maps = dict(extent_left=[1,4,1,4], extent_right=[2,3,2,3], width_ratios=[0.67, 1.33])`
- **map_font_size** (*int*) – Font size for grid labels.
- **markersize** (*int*) – Markersize for points. Default is 5.
- **markeredgewidth** (*float*) – Edge width for markers for points (black line). Default is 0.5.
- **linewidth_data** (*int*) – Line width for plotting data when it involves lines.
- **linewidth_poly** (*int*) – Line width for plotting polygon.
- **alpha_marker** (*float*) – alpha for markers for points
- **colors_data** (*str*) – One color to use for all or colors in a list matching number of stations.
- **legend** (*bool*) – True for legend instead of annotations.

- **loc** (*str*) – legend location for matplotlib. Default "best".
- **suptitle** (*str*) – Title for top of the figure, overall.
- **suptitle_fontsize** (*int*) – Fontsize for suptitle. Default is 16.
- **tight_layout** (*bool*) – Whether to use tight_layout when have 2 maps.

ocean_model_skill_assessor.plot.map.**setup_ax**(*ax*, *left_labels=True*, *right_labels=False*, *bottom_labels=False*, *top_labels=True*, *fontsize=12*)

Basic plot setup for map.

1.4.6 ocean_model_skill_assessor.plot.line

Time series plots.

Functions

plot(obs, model, xname, yname[, title, ...])	Plot time series or CTD profile.

ocean_model_skill_assessor.plot.line.**plot**(*obs*, *model*, *xname*, *yname*, *title=None*, *xlabel=None*, *ylabel=None*, *model_label='Model'*, *figname='figure.png'*, *dpi=100*, *figsize=(15, 5)*, *return_plot=False*, ***kwargs*)

Plot time series or CTD profile.

Use for featuretype of timeSeries or profile. Plot obs vs. model as time series line plot or CTD profile.

Parameters

- **obs** (*DataFrame, Dataset*) – Observation time series
- **mode** (*Dataset*) – Model time series to compare against obs
- **xname** (*str*) – Name of variable to plot on x-axis when interpreted with cf-xarray and cf-pandas
- **yname** (*str*) – Name of variable to plot on y-axis when interpreted with cf-xarray and cf-pandas
- **title** (*str, optional*) – Title for plot.
- **xlabel** (*str, optional*) – Label for x-axis.
- **ylabel** (*str, optional*) – Label for y-axis.
- **figname** (*str*) – Filename for figure (as absolute or relative path).
- **dpi** (*int, optional*) – dpi for figure. Default is 100.
- **figsize** (*tuple, optional*) – Figsize to pass to *plt.figure()*. Default is (15,5).
- **return_plot** (*bool*) – If True, return plot. Use for testing.

1.4.7 ocean_model_skill_assessor.plot.surface

Surface plot.

Functions

plot(obs, model, xname, yname, zname, suptitle)	Plot scatter or surface plot.

ocean_model_skill_assessor.plot.surface.**plot**(*obs*, *model*, *xname*, *yname*, *zname*, *suptitle*, *xlabel=None*, *ylabel=None*, *zlabel=None*, *model_title='Model'*, *along_transect_distance=False*, *plot_on_map=False*, *proj=None*, *extent=None*, *kind='pcolormesh'*, *nsubplots=3*, *figname='figure.png'*, *dpi=100*, *figsize=(15, 6)*, *return_plot=False*, *invert_yaxis=False*, *make_Z_negative=None*, ***kwargs*)

Plot scatter or surface plot.

For featuretype of trajectoryProfile or timeSeriesProfile.

Parameters

- **obs** (*DataFrame, Dataset*) – Observation time series
- **mode** (*Dataset*) – Model time series to compare against obs
- **xname** (*str*) – Name of variable to plot on x-axis when interpreted with cf-xarray and cf-pandas
- **yname** (*str*) – Name of variable to plot on y-axis when interpreted with cf-xarray and cf-pandas
- **zname** (*str*) – Name of variable to plot with color when interpreted with cf-xarray and cf-pandas
- **suptitle** (*str, optional*) – Title for plot, over all the subplots.
- **xlabel** (*str, optional*) – Label for x-axis.
- **ylabel** (*str, optional*) – Label for y-axis.
- **zlabel** (*str, optional*) – Label for colorbar.
- **along_transect_distance** – Set to True to calculate the along-transect distance in km from the longitude and latitude, which must be interpretable through cf-pandas or cf-xarray as "longitude" and "latitude".
- **kind** (*str*) – Can be "pcolormesh" for surface plot or "scatter" for scatter plot.
- **nsubplots** (*int, optional*) – Number of subplots. Might always be 3, and that is the default.
- **figname** (*str*) – Filename for figure (as absolute or relative path).
- **dpi** (*int, optional*) – dpi for figure. Default is 100.
- **figsize** (*tuple, optional*) – Figsize to pass to *plt.figure()*. Default is (15,5).
- **return_plot** (*bool*) – If True, return plot. Use for testing.

```
import ocean_model_skill_assessor as omsa
from IPython.display import Code, Image
```

1.5 CLI demo of `ocean-model-skill-assessor` with known data files

This demo runs command line interface (CLI) commands only, which is accomplished in a Jupyter notebook by prefacing commands with `!`. To transfer these commands to a terminal window, remove the `!` but otherwise keep commands the same.

More detailed docs about running with the CLI are *available*.

There are three steps to follow for a set of model-data validation, which is for one variable:

1. Make a catalog for your model output.
2. Make a catalog for your data.
3. Run the comparison.

These steps will save files into a user application directory cache, along with a log. A project directory can be checked on the command line with `omsa proj_path --project_name PROJECT_NAME`.

1.5.1 Make model catalog

Set up a catalog file for your model output. The user can input necessary keyword arguments – through `kwargs_open` – so that `xarray` will be able to read in the model output. Generally it is good to use `skip_entry_metadata` when using the `make_catalog` command for model output since we are using only one model and the entry metadata is aimed at being able to compare datasets.

In the following command,

- `make_catalog` is the function being run from OMSA
- `demo_local` is the name of the project which will be used as the subdirectory name
- `local` is the type of catalog to choose when making a catalog for the model output regardless of where the model output is stored
- "model" is the catalog name which will be used for the file name and in the catalog itself
- Specific `kwargs` to be input to the catalog command are
  - `filenames` which is a string describing where the model output can be found. If the model output is available through a sequence of filenames instead of a single server address, represent them with a single `glob`-style statement, for example, "/filepath/filenameprefix_*.nc".
  - `skip_entry_metadata` use this when running `make_catalog` for model output
- `kwargs_open` all keywords required for `xr.open_dataset` or `xr.open_mfdataset` to successfully read your model output.

```
# get local path for model output sample file from xroms
import xroms
url = xroms.datasets.CLOVER.fetch("ROMS_example_full_grid.nc")
```

```
!omsa make_catalog --project_name demo_local --catalog_type local --catalog_name model --
↪kwargs filenames=$url skip_entry_metadata=True
```

```
[2023-11-27 22:04:51,283] {/home/docs/checkouts/readthedocs.org/user_builds/ocean-model-
↪skill-assessor/checkouts/latest/ocean_model_skill_assessor/main.py:430}
INFO - Catalog saved to /home/docs/.cache/ocean-model-skill-assessor/demo_local/model.
↪yaml with 1 entries.
```

1.5.2 Make data catalog

Set up a catalog of the datasets with which you want to compare your model output. In this example, we use only known data file locations to create our catalog.

In this step, we use the same `project_name` as in the previous step so as to put the resulting catalog file in the same subdirectory, we create a catalog of type "local" since we have known data locations, we call this catalog file "local", input the filenames as a list in quotes (this specific syntax is necessary for inputting a list in through the command line interface), and we input any keyword arguments necessary for reading the datasets.

In the following command:

- `make_catalog` is the function being run from OMSA
- `demo_local` is the name of the project which will be used as the subdirectory name
- `local` is the type of catalog to choose when making a catalog for the known data files
- "local" is the catalog name which will be used for the file name and in the catalog itself
- Specific `kwargs` to be input to the catalog command are
  - `filenames` which is a string or a list of strings pointing to where the data files can be found. If you are using a list, the syntax for the command line interface is `filenames="[file1,file2]"`.
- `kwargs_open` all keywords required for `xr.open_dataset` or `xr.open_mfdataset` or `pandas.open_csv`, or whatever method will ultimately be used to successfully read your model output. These must be applicable to all datasets represted by `filenames`. If they are not, run this command multiple times, one for each set of filenames and `kwargs_open` that match.

```
!omsa make_catalog --project_name demo_local --catalog_type local --catalog_name local --
↪kwargs filenames="[https://erddap.sensors.axds.co/erddap/tabledap/gov_ornl_cdiac_
↪coastalms_88w_30n.csvp?time%2Clatitude%2Clongitude%2Cz%2Csea_water_temperature&time
↪%3E=2009-11-19T012%3A00%3A00Z&time%3C=2009-11-19T16%3A00%3A00Z]" --metadata␣
↪featuretype=timeSeries maptype=point
```

```
[2023-11-27 22:05:01,869] {/home/docs/checkouts/readthedocs.org/user_builds/ocean-model-
↪skill-assessor/checkouts/latest/ocean_model_skill_assessor/main.py:216}
WARNING - Dataset gov_ornl_cdiac_coastalms_88w_30n had a timezone UTC which is being␣
↪removed. Make sure the timezone matches the model output.

[2023-11-27 22:05:01,883] {/home/docs/checkouts/readthedocs.org/user_builds/ocean-model-
↪skill-assessor/checkouts/latest/ocean_model_skill_assessor/main.py:430}
INFO - Catalog saved to /home/docs/.cache/ocean-model-skill-assessor/demo_local/local.
↪yaml with 1 entries.
```

1.5.3 Run comparison

Now that the model output and dataset catalogs are prepared, we can run the comparison of the two.

In this step, we use the same `project_name` as the other steps so as to keep all files in the same subdirectory. We input the data catalog name under `catalog_names` and the model catalog name under `model_name`.

At this point we need to select a single variable to compare between the model and datasets, and this requires a little extra input. Because we don't know anything about the format of any given input data file, variables will be interpreted with some flexibility in the form of a set of regular expressions. In the present case, we will compare the water temperature between the model and the datasets (the model output and datasets selected for our catalogs should contain the variable we want to compare). Several sets of regular expressions, called "vocabularies", are available with the package to be used for this purpose, and in this case we will use one called "general" which should match many commonly-used variable names. "general" is selected under `vocab_names`, and the particular key from the general vocabulary that we are comparing is selected with `key`.

See the vocabulary here.

```
import cf_pandas as cfp

paths = omsa.paths.Paths()
vocab = cfp.Vocab(paths.VOCAB_PATH("general"))
vocab
```

```
{'temp': {'name': '(?i)^(?!.*(air|qc|status|atmospheric|bottom|dew)).*(temp|sst).*'},
↪'salt': {'name': '(?i)^(?!.*(soil|qc|status|bottom)).*(sal|sss).*'}, 'ssh': {'name':
↪'(?i)^(?!.*(qc|status)).*(sea_surface_height|surface_elevation|zeta).*'}, 'u': {'name
↪': 'u$|(?i)(?=.*east)(?=.*vel)'}, 'v': {'name': 'v$|(?i)(?=.*north)(?=.*vel)'}, 'w': {
↪'name': 'w$|(?i)(?=.*up)(?=.*vel)'}, 'water_dir': {'name': '(?i)^(?!.
↪*(qc|status|air|wind))(?=.*dir)(?=.*water)'}, 'water_speed': {'name': '(?i)^(?!.
↪*(qc|status|air|wind))(?=.*speed)(?=.*water)'}, 'wind_dir': {'name': '(?i)^(?!.
↪*(qc|status|water))(?=.*dir)(?=.*wind)'}, 'wind_speed': {'name': '(?i)^(?!.
↪*(qc|status|water))(?=.*speed)(?=.*wind)'}, 'sea_ice_u': {'name': '(?i)^(?!.
↪*(qc|status))(?=.*sea)(?=.*ice)(?=.*u)|(?i)^(?!.*(qc|status))(?=.*sea)(?=.*ice)(?=.
↪*x)(?=.*vel)|(?i)^(?!.*(qc|status))(?=.*sea)(?=.*ice)(?=.*east)(?=.*vel)'}, 'sea_ice_v
↪': {'name': '(?i)^(?!.*(qc|status))(?=.*sea)(?=.*ice)(?=.*v)|(?i)^(?!.*(qc|status))(?=.
↪*sea)(?=.*ice)(?=.*y)(?=.*vel)|(?i)^(?!.*(qc|status))(?=.*sea)(?=.*ice)(?=.*north)(?=.
↪*vel)'}, 'sea_ice_area_fraction': {'name': '(?i)^(?!.*(qc|status))(?=.*sea)(?=.*ice)(?
↪=.*area)(?=.*fraction)'}}
```

In the following command:

- `run` is the function being run from OMSA
- `demo_local` is the name of the project which will be used as the subdirectory name
- `catalog_names` are the names of any catalogs with datasets to include in the comparison. In this case we have just one called "local"
- `model_name` is the name of the model catalog we previously created
- `vocab_names` are the names of the vocabularies to use for interpreting which variable to compare from the model output and datasets. If multiple are input, they are combined together. The variable nicknames need to match in the vocabularies to be interpreted together.
- `key` is the nickname or alias of the variable as given in the input vocabulary

```
!omsa run --project_name demo_local --catalog_names local --model_name model --vocab_
↪names general \
        --key temp \
      --kwargs_map label_with_station_name=True \
         --more_kwargs interpolate_horizontal=False check_in_boundary=False plot_
↪map=True dd=5 alpha=20
```

```
[2023-11-27 22:05:10,950] {/home/docs/checkouts/readthedocs.org/user_builds/ocean-model-
↪skill-assessor/checkouts/latest/ocean_model_skill_assessor/main.py:1803}
INFO - Input parameters: {'catalogs': ['local'], 'project_name': 'demo_local', 'key_
↪variable': 'temp', 'model_name': 'model', 'vocabs': ['general'], 'vocab_labels': None,
↪'ndatasets': None, 'kwargs_map': {'label_with_station_name': True}, 'verbose': True,
↪'mode': 'w', 'testing': False, 'alpha': 20, 'dd': 5, 'preprocess': False, 'need_xgcm_
↪grid': False, 'xcmocean_options': None, 'kwargs_xroms': None, 'locstream': True,
↪'interpolate_horizontal': False, 'horizontal_interp_code': 'delaunay', 'save_
↪horizontal_interp_weights': True, 'want_vertical_interp': False, 'extrap': False,
↪'model_source_name': None, 'catalog_source_names': None, 'user_min_time': None, 'user_
↪max_time': None, 'check_in_boundary': False, 'tidal_filtering': None, 'ts_mods': None,
↪'model_only': False, 'plot_map': True, 'no_Z': False, 'skip_mask': False, 'wetdry':␣
↪False, 'plot_count_title': True, 'cache_dir': None, 'return_fig': False, 'override_
↪model': False, 'override_processed': False, 'override_stats': False, 'override_plot':␣
↪False, 'plot_description': None, 'kwargs_plot': None, 'skip_key_variable_check': False,
↪ 'kwargs': {}, 'paths': <ocean_model_skill_assessor.paths.Paths object at␣
↪0x7ff5ecd33af0>, 'logger': <Logger ocean_model_skill_assessor.utils (INFO)>}
```

```
[2023-11-27 22:05:10,971] {/home/docs/checkouts/readthedocs.org/user_builds/ocean-model-
↪skill-assessor/checkouts/latest/ocean_model_skill_assessor/main.py:1838}
INFO - Note that there are 1 datasets to use. This might take awhile.

[2023-11-27 22:05:10,971] {/home/docs/checkouts/readthedocs.org/user_builds/ocean-model-
↪skill-assessor/checkouts/latest/ocean_model_skill_assessor/main.py:1855}
INFO - Catalog <Intake catalog: local>.

[2023-11-27 22:05:10,971] {/home/docs/checkouts/readthedocs.org/user_builds/ocean-model-
↪skill-assessor/checkouts/latest/ocean_model_skill_assessor/main.py:1870}
INFO -
source name: gov_ornl_cdiac_coastalms_88w_30n (1 of 1 for catalog <Intake catalog: local>
↪.
```

```
[2023-11-27 22:05:11,162] {/home/docs/checkouts/readthedocs.org/user_builds/ocean-model-
↪skill-assessor/checkouts/latest/ocean_model_skill_assessor/main.py:1069}
INFO -
                        User time range: NaT to NaT.
                        Model time range: 2009-11-19 12:00:00 to 2009-11-19 16:00:00.
                        Data time range: 2009-11-19 12:17:00 to 2009-11-19 15:17:00.
                        Data lon range: -88.6 to -88.6.
                        Data lat range: 30.0 to 30.0.

[2023-11-27 22:05:11,163] {/home/docs/checkouts/readthedocs.org/user_builds/ocean-model-
↪skill-assessor/checkouts/latest/ocean_model_skill_assessor/main.py:1946}
INFO - running gov_ornl_cdiac_coastalms_88w_30n for key_variable(s) temp from key_
↪variable_list ['temp']
```

```
[2023-11-27 22:05:11,714] {/home/docs/checkouts/readthedocs.org/user_builds/ocean-model-
↪skill-assessor/checkouts/latest/ocean_model_skill_assessor/main.py:878}
INFO - Processed data file name is /home/docs/.cache/ocean-model-skill-assessor/demo_
↪local/processed/local_gov_ornl_cdiac_coastalms_88w_30n_temp_data.csv.

[2023-11-27 22:05:11,714] {/home/docs/checkouts/readthedocs.org/user_builds/ocean-model-
↪skill-assessor/checkouts/latest/ocean_model_skill_assessor/main.py:879}
INFO - Processed model file name is /home/docs/.cache/ocean-model-skill-assessor/demo_
↪local/processed/local_gov_ornl_cdiac_coastalms_88w_30n_temp_model.nc.

[2023-11-27 22:05:11,714] {/home/docs/checkouts/readthedocs.org/user_builds/ocean-model-
↪skill-assessor/checkouts/latest/ocean_model_skill_assessor/main.py:880}
INFO - model file name is /home/docs/.cache/ocean-model-skill-assessor/demo_local/model_
↪output/local_gov_ornl_cdiac_coastalms_88w_30n_temp.nc.

[2023-11-27 22:05:11,714] {/home/docs/checkouts/readthedocs.org/user_builds/ocean-model-
↪skill-assessor/checkouts/latest/ocean_model_skill_assessor/main.py:2004}
INFO - Figure name is /home/docs/.cache/ocean-model-skill-assessor/demo_local/out/local_
↪gov_ornl_cdiac_coastalms_88w_30n_temp.png.

[2023-11-27 22:05:11,714] {/home/docs/checkouts/readthedocs.org/user_builds/ocean-model-
↪skill-assessor/checkouts/latest/ocean_model_skill_assessor/main.py:2025}
INFO - No previously processed model output and data available for gov_ornl_cdiac_
↪coastalms_88w_30n, so setting up now.

[2023-11-27 22:05:11,717] {/home/docs/checkouts/readthedocs.org/user_builds/ocean-model-
↪skill-assessor/checkouts/latest/ocean_model_skill_assessor/main.py:1363}
INFO - Finding and saving mask to cache to /home/docs/.cache/ocean-model-skill-assessor/
↪demo_local/mask_temp.nc.

[2023-11-27 22:05:11,717] {/home/docs/checkouts/readthedocs.org/user_builds/ocean-model-
↪skill-assessor/checkouts/latest/ocean_model_skill_assessor/utils.py:570}
INFO - Retrieving mask
```

```
[2023-11-27 22:05:14,153] {/home/docs/checkouts/readthedocs.org/user_builds/ocean-model-
↪skill-assessor/checkouts/latest/ocean_model_skill_assessor/main.py:1158}
INFO - Calculating numerical domain boundary.

[2023-11-27 22:05:14,157] {/home/docs/checkouts/readthedocs.org/user_builds/ocean-model-
↪skill-assessor/checkouts/latest/ocean_model_skill_assessor/main.py:969}
WARNING - Dataset gov_ornl_cdiac_coastalms_88w_30n had a timezone UTC which is being␣
↪removed. Make sure the timezone matches the model output.
```

```
[2023-11-27 22:05:14,241] {/home/docs/checkouts/readthedocs.org/user_builds/ocean-model-
↪skill-assessor/checkouts/latest/ocean_model_skill_assessor/main.py:793}
WARNING - the 'vertical' key cannot be identified in dam by cf-xarray. Maybe you need to␣
↪include the xgcm grid and vertical metrics for xgcm grid, but maybe your variable does␣
↪not have a vertical axis.

[2023-11-27 22:05:14,249] {/home/docs/checkouts/readthedocs.org/user_builds/ocean-model-
↪skill-assessor/checkouts/latest/ocean_model_skill_assessor/main.py:667}
INFO - Will not perform vertical interpolation and will find nearest depth to 0.0.
```

(continues on next page)

(continued from previous page)

```
[2023-11-27 22:05:14,249] {/home/docs/checkouts/readthedocs.org/user_builds/ocean-model-
↪skill-assessor/checkouts/latest/ocean_model_skill_assessor/main.py:1452}
INFO - Selecting model output at locations to match dataset gov_ornl_cdiac_coastalms_88w_
↪30n.
```

```
[2023-11-27 22:05:14,307] {/home/docs/checkouts/readthedocs.org/user_builds/ocean-model-
↪skill-assessor/checkouts/latest/ocean_model_skill_assessor/main.py:1484}
INFO -
    Model coordinates found are Coordinates:
    xi_rho      int64 299
    eta_rho     int64 92
    lon_rho     float64 -88.6
    lat_rho     float64 29.97
    s_rho       float64 -0.01667
    npts        int64 0
  * ocean_time  (ocean_time) datetime64[ns] 2009-11-19T12:17:00 2009-11-19T15....

    Output information from finding nearest neighbors to requested points are {'distances
↪': array([0.12069942]), 'eta_rho': array([92]), 'xi_rho': array([299])}.

[2023-11-27 22:05:14,310] {/home/docs/checkouts/readthedocs.org/user_builds/ocean-model-
↪skill-assessor/checkouts/latest/ocean_model_skill_assessor/main.py:1524}
INFO - Trying to drop vertical coordinates time series

[2023-11-27 22:05:14,311] {/home/docs/checkouts/readthedocs.org/user_builds/ocean-model-
↪skill-assessor/checkouts/latest/ocean_model_skill_assessor/main.py:1537}
INFO - Loading model output...
```

```
[2023-11-27 22:05:14,514] {/home/docs/checkouts/readthedocs.org/user_builds/ocean-model-
↪skill-assessor/checkouts/latest/ocean_model_skill_assessor/main.py:1644}
INFO - Saving model output to file...
```

```
[2023-11-27 22:05:14,673] {/home/docs/checkouts/readthedocs.org/user_builds/ocean-model-
↪skill-assessor/checkouts/latest/ocean_model_skill_assessor/main.py:2337}
INFO - model file name is /home/docs/.cache/ocean-model-skill-assessor/demo_local/model_
↪output/local_gov_ornl_cdiac_coastalms_88w_30n_temp.nc.

[2023-11-27 22:05:14,673] {/home/docs/checkouts/readthedocs.org/user_builds/ocean-model-
↪skill-assessor/checkouts/latest/ocean_model_skill_assessor/main.py:2339}
INFO - Reading model output from file.
```

```
[2023-11-27 22:05:14,801] {/home/docs/checkouts/readthedocs.org/user_builds/ocean-model-
↪skill-assessor/checkouts/latest/ocean_model_skill_assessor/main.py:2362}
INFO - Calculating stats for temp.

[2023-11-27 22:05:14,814] {/home/docs/checkouts/readthedocs.org/user_builds/ocean-model-
↪skill-assessor/conda/latest/lib/python3.9/warnings.py:109}
WARNING - /home/docs/checkouts/readthedocs.org/user_builds/ocean-model-skill-assessor/
↪conda/latest/lib/python3.9/site-packages/numpy/lib/function_base.py:2853:␣
↪RuntimeWarning: invalid value encountered in divide
```

(continues on next page)

(continued from previous page)

```
  c /= stddev[:, None]


[2023-11-27 22:05:14,814] {/home/docs/checkouts/readthedocs.org/user_builds/ocean-model-
↪skill-assessor/conda/latest/lib/python3.9/warnings.py:109}
WARNING - /home/docs/checkouts/readthedocs.org/user_builds/ocean-model-skill-assessor/
↪conda/latest/lib/python3.9/site-packages/numpy/lib/function_base.py:2854:␣
↪RuntimeWarning: invalid value encountered in divide
  c /= stddev[None, :]


[2023-11-27 22:05:14,818] {/home/docs/checkouts/readthedocs.org/user_builds/ocean-model-
↪skill-assessor/conda/latest/lib/python3.9/warnings.py:109}
WARNING - /home/docs/checkouts/readthedocs.org/user_builds/ocean-model-skill-assessor/
↪checkouts/latest/ocean_model_skill_assessor/stats.py:100: RuntimeWarning: divide by␣
↪zero encountered in double_scalars
  return float(1 - ((obs - model) ** 2).sum() / ((obs - obs_model) ** 2).sum())
```

```
[2023-11-27 22:05:14,823] {/home/docs/checkouts/readthedocs.org/user_builds/ocean-model-
↪skill-assessor/checkouts/latest/ocean_model_skill_assessor/main.py:2380}
INFO - Saved stats file.
```

```
[2023-11-27 22:05:16,106] {/home/docs/checkouts/readthedocs.org/user_builds/ocean-model-
↪skill-assessor/checkouts/latest/ocean_model_skill_assessor/main.py:2437}
INFO - Made plot for gov_ornl_cdiac_coastalms_88w_30n
.
```

```
[2023-11-27 22:05:39,261] {/home/docs/checkouts/readthedocs.org/user_builds/ocean-model-
↪skill-assessor/checkouts/latest/ocean_model_skill_assessor/main.py:2451}
INFO - Finished analysis. Find plots, stats summaries, and log in /home/docs/.cache/
↪ocean-model-skill-assessor/demo_local.
```

1.5.4 Look at results

Now we can look at the results from our comparison! You can find the location of the resultant files printed at the end of the `run` command output above. Or you can find the path to the project directory while in Python with:

```
paths = omsa.paths.Paths("demo_local")
paths.PROJ_DIR
```

```
PosixPath('/home/docs/.cache/ocean-model-skill-assessor/demo_local')
```

Or you can use a command:

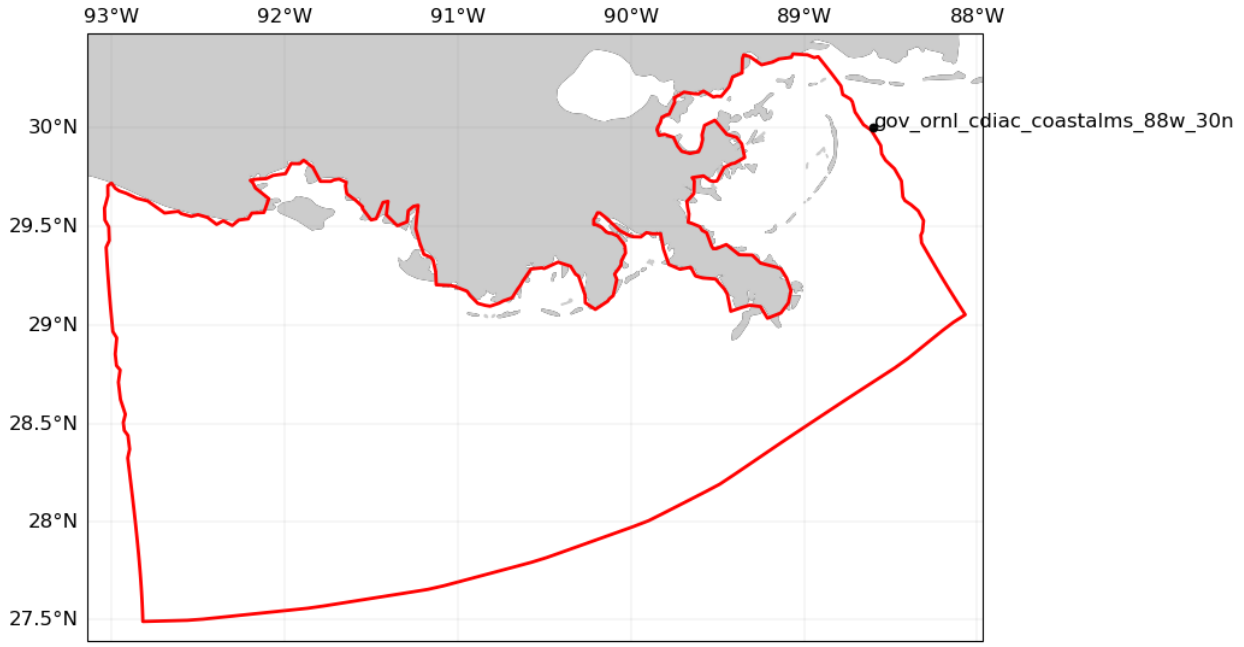
```
!omsa proj_path --project_name demo_local
```

```
/home/docs/.cache/ocean-model-skill-assessor/demo_local
```

Here we know the names of the files so show them inline.

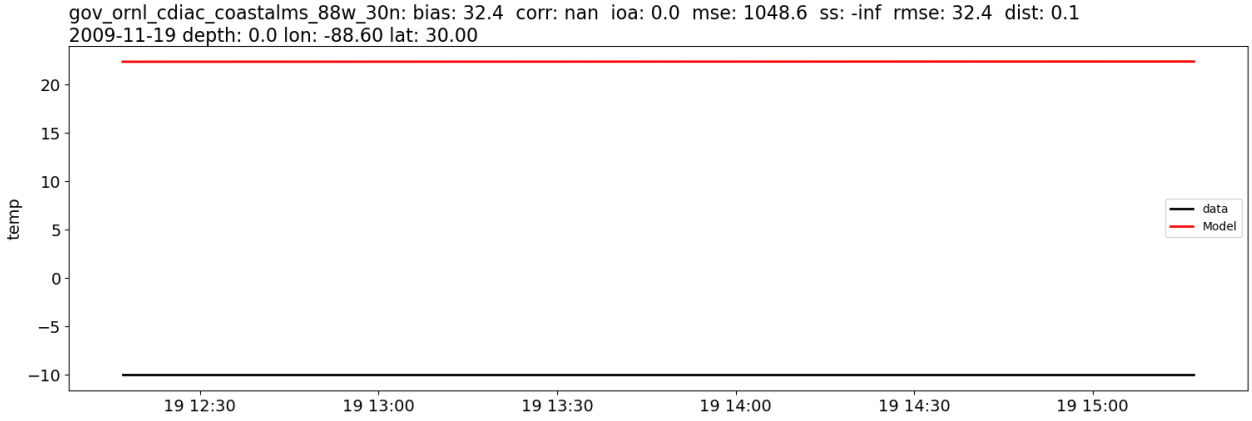
First we see a map of the area around the Mississippi river delta, along with a red line outlining the approximate domain of the numerical model, and 1 black dot indicating 1 data location, marked with a the station name.

```
Image(paths.OUT_DIR / "map.png")
```



Here we see a time series comparison for station "gov_ornl_cdiac_coastalms_88w_30n". It shows in black the temperature values from the data and in red the comparable values from the model. The comparison time range is November 19, 2009 from 12 to 15:30. The lines are not similar because the data is actually missing during this time period. Statistical comparisons are also available in the title text.

```
Image(paths.OUT_DIR / "local_gov_ornl_cdiac_coastalms_88w_30n_temp.png")
```




```
import yaml
with open(paths.OUT_DIR / "local_gov_ornl_cdiac_coastalms_88w_30n_temp.yaml", "r") as␣
↪stream:
    stats = yaml.safe_load(stream)
stats
```

```
{'bias': {'long_name': 'Bias or MSD',
  'name': 'Bias',
  'value': 32.38266357485453},
 'corr': {'long_name': 'Pearson product-moment correlation coefficient',
  'name': 'Correlation Coefficient',
  'value': nan},
 'descriptive': {'long_name': 'Max, Min, Mean, Standard Deviation',
  'name': 'Descriptive Statistics',
  'value': [22.396244923273724,
   22.371082226435345,
   22.383663574854534,
   0.012581348419189453]},
 'dist': {'long_name': 'Distance in km from data location to selected model location',
  'name': 'Distance',
  'value': 0.12069941685270609},
 'ioa': {'long_name': 'Index of Agreement (Willmott 1981)',
  'name': 'Index of Agreement',
  'value': 0.0},
 'mse': {'long_name': 'Mean Squared Error (MSE)',
  'name': 'Mean Squared Error',
  'value': 1048.6370584925387},
 'rmse': {'long_name': 'Root Mean Square Error (RMSE)',
  'name': 'RMSE',
  'value': 32.38266601891417},
 'ss': {'long_name': 'Skill Score (Bogden 1996)',
  'name': 'Skill Score',
  'value': -inf}}
```


1.6 Using OMSA through Command Line Interface (CLI)

Example commands are shown (but not run) below. You can copy these commands directly to a terminal window or command prompt.

This page is focused on explaining all the command line options, not demonstrating a workflow. For a more clear demonstration, check out the *Python package demo* or *CLI demo*.

1.6.1 Make catalog(s) for data and model

There are 3 types of catalogs in OMSA: local, erddap, and axds.

Local catalog

Make a catalog with known local or remote file(s). Also use a local catalog to represent your model output.

Available options

```
omsa make_catalog --project_name PROJ_NAME --catalog_type local --catalog_name CATALOG_
↪NAME --description "Catalog description" --kwargs filenames="[FILE1,FILE2]" --kwargs_
↪open KWARG=VALUE --verbose --mode MODE
```

- `project_name`: Will be used as the name of the directory where the catalog is saved. The directory is located in a user application cache directory, the address of which can be found for your setup with `omsa proj_path --project_name PROJ_NAME`.
- `catalog_type`: Type of catalog to make. Options are "erddap", "axds", or "local".
- `catalog_name`: Name for catalog.
- `description`: Description for catalog.
- `metadata`: Metadata for catalog.
- `kwargs`: Some keyword arguments to make the local catalog. See `omsa.main.make_local_catalog()` for more details.
  - `filenames`: (Required) Where to find dataset(s) from which to make local catalog.
- `kwargs_open`: Keyword arguments to pass on to the appropriate intake open_* call for model or dataset.
- `verbose` Print useful runtime commands to stdout if True as well as save in log, otherwise silently save in log. Log is located in the project directory, which can be checked on the command line with `omsa proj_path --project_name PROJECT_NAME`. Default is True, to turn off use `--no-verbose`.
- `mode` mode for logging file. Default is to overwrite an existing logfile, but can be changed to other modes, e.g. "a" to instead append to an existing log file.

Examples

Basic catalog for single dataset

```
omsa make_catalog --project_name test1 --catalog_type local --catalog_name example_local_
↪catalog --description "Example local catalog description" --kwargs filenames="[https://
↪erddap.sensors.axds.co/erddap/tabledap/aoos_204.csvp?time%2Clatitude%2Clongitude%2Cz
↪%2Csea_water_temperature&time%3E=2022-01-01T00%3A00%3A00Z&time%3C=2022-01-06T00%3A00
↪%3A00Z]"
```

Dataset with no lon/lat

When a dataset does not contain location information, you can input it as metadata to the catalog with the dataset. However, you need to input one filename and one set of metadata per catalog call.

Station page: https://tidesandcurrents.noaa.gov/stationhome.html?id=9455500

```
omsa make_catalog --project_name test1 --catalog_type local --catalog_name example_local_
↪catalog2 --kwargs filenames="[https://api.tidesandcurrents.noaa.gov/api/prod/
↪datagetter?product=water_temperature&application=NOS.COOPS.TAC.PHYSOCEAN&begin_
↪date=20230109&end_date=20230109&station=9455500&time_zone=GMT&units=english&interval=6&
↪format=csv]" --metadata minLongitude=-151.72 maxLongitude=-151.72 minLatitude=59.44␣
↪maxLatitude=59.44
```

Set up model

Use this approach to set up a catalog file for your model output, so that it can be used by OMSA. Use `skip_entry_metadata=True` when running for a model.

```
omsa make_catalog --project_name test1 --catalog_type local --catalog_name model --
↪kwargs filenames="https://www.ncei.noaa.gov/thredds/dodsC/model-ciofs-agg/Aggregated_
↪CIOFS_Fields_Forecast_best.ncd" skip_entry_metadata=True  --kwargs_open drop_
↪variables=ocean_time
```

ERDDAP Catalog

Make a catalog from datasets available from an ERDDAP server using `intake-erddap`.

Available options

```
omsa make_catalog --project_name PROJ_NAME --catalog_type erddap --catalog_name CATALOG_
↪NAME --description "Catalog description" --kwargs server=SERVER --kwargs_search min_
↪lon=MIN_LON min_lat=MIN_LAT max_lon=MAX_LON max_lat=MAX_LAT min_time=MIN_TIME max_
↪time=MAX_TIME search_for=SEARCH_TEXT --verbose --mode MODE
```

- `project_name`: Will be used as the name of the directory where the catalog is saved. The directory is located in a user application cache directory, the address of which can be found for your setup with `omsa proj_path --project_name PROJ_NAME`.
- `catalog_type`: Type of catalog to make. Options are “erddap”, “axds”, or “local”.
- `catalog_name`: Name for catalog.
- `description`: Description for catalog.
- `metadata`: Metadata for catalog.
- `vocab_name`: Name of vocabulary to use from vocab dir. Options are “standard_names” and “general”. See more information *here*.
- `kwargs`: Some keyword arguments to make the ERDDAP catalog. See `intake-erddap.erddap_cat()` for more details.
  - `server`: ERDDAP server address, for example: “http://erddap.sensors.ioos.us/erddap”
  - `category_search`:
  - `use_source_constraints`: Any relevant search parameter defined in kwargs_search will be passed to the source objects as constraints.
  - `protocol`: str, default “tabledap”
  - `query_type`: Specifies how the catalog should apply the query parameters. Choices are `"union"` or `"intersection"`. If the `query_type` is set to `"intersection"`, then the set of results will be the intersection of each individual query made to ERDDAP. This is equivalent to a logical AND of the results. If the value is `"union"` then the results will be the union of each resulting dataset. This is equivalent to a logical OR.
  - other keyword arguments can be passed into the intake `Catalog` class
- `kwargs_search`: Keyword arguments to input to search on the server before making the catalog.

  - `min_lon`, `min_lat`, `max_lon`, `max_lat`: search for datasets within this spatial box
  - `min_time`, `max_time`: search for datasets with data within this time range
  - `model_name`: input a path to the model output to instead select the space and time search specifications based on the model. This input is specific to OMSA, not `intake-erddap`.
  - `search_for`: text-based search

- `verbose` Print useful runtime commands to stdout if True as well as save in log, otherwise silently save in log. Log is located in the project directory, which can be checked on the command line with `omsa proj_path --project_name PROJECT_NAME`. Default is True, to turn off use `--no-verbose`.
- `mode` mode for logging file. Default is to overwrite an existing logfile, but can be changed to other modes, e.g. "a" to instead append to an existing log file.

Examples

Narrow search by input selections

Select a spatial box and time range over which to search catalog:

```
omsa make_catalog --project_name test1 --catalog_type erddap --catalog_name example_
↪erddap_catalogA --description "Example ERDDAP catalog description" --kwargs server=
↪"https://erddap.sensors.ioos.us/erddap" --kwargs_search min_lon=-170 min_lat=53 max_
↪lon=-165 max_lat=56 min_time=2022-1-1 max_time=2022-1-6
```

Narrow search with model output

Input model output to use to create the space search range, but choose time search range. We use the model catalog created in a previous example:

```
omsa make_catalog --project_name test1 --catalog_type erddap --catalog_name example_
↪erddap_catalog --description "Example ERDDAP catalog description" --kwargs server=
↪"https://erddap.sensors.ioos.us/erddap" --kwargs_search model_name=model min_time=2022-
↪1-1 max_time=2022-1-6
```

Narrow search also with `query_type`

You can additionally narrow your search by a text term by adding the `search_for` and `query_type` keyword inputs. This example searches for datasets containing the varaible "sea_surface_temperature" and, somewhere in the dataset metadata, the term "Timeseries". If we had wanted datasets that contain one OR the other, we could use `query_type=union`.

```
omsa make_catalog --project_name test1 --catalog_type erddap --catalog_name cat2 --
↪kwargs server="https://erddap.sensors.ioos.us/erddap" standard_names="[sea_surface_
↪temperature]" search_for="[Timeseries]" query_type=intersection
```

Variable selection by standard_name

Narrow your search by variable. For `intake-erddap` you can filter by the CF `standard_name` of the variable directly with the following.

```
omsa make_catalog --project_name test1 --catalog_type erddap --catalog_name cat1 --
↪kwargs server="https://erddap.sensors.ioos.us/erddap" standard_names="[sea_surface_
↪temperature,sea_water_temperature]"
```

Variable selection by pattern matching with vocab

You can return equivalent results in your catalog by searching with a variable nickname (the keys in the dictionary) along with a dictionary defining a vocabulary of regular expressions for matching what "counts" as a particular variable. To save a custom vocabulary to a location for this command, use the `Vocab` class in `cf-pandas` (docs). A premade set of vocabularies aimed at use by ocean modelers is also available to use by name; see them with command `omsa vocabs`. See more information *here*. Suggested uses:

- axds catalog: vocab_name standard_names
- erddap catalog, IOOS: vocab_name standard_names
- erddap catalog, Coastwatch: vocab_name standard_names
- local catalog: vocab_name general

This is more complicated than simply defining the desired standard_names as shown in the previous example. However, it becomes useful when using other data files or model output which might have different variable names but could be recognized with variable matching through the vocabulary.

The example below uses the pre-defined vocabulary "standard_names" since we are using the IOOS ERDDAP server which uses standard_names as one of its search categories, and will search for matching variables by standard_name and matching the variable nickname "temp". The "standard_names" vocabulary is shown here and includes the standard_names from the previous example (it includes others too but they aren't present on the server). The regular expressions are set up to match exactly those standard_names. This is why we return the same results from either approach.

```
vocab = cfp.Vocab(omsa.VOCAB_PATH("standard_names"))
```

```
omsa make_catalog --project_name test1 --catalog_type erddap --catalog_name cat3 --
↪kwargs server="https://erddap.sensors.ioos.us/erddap" category_search="[standard_name,
↪temp]" --vocab_name standard_names
```

Catalog for Axiom assets

Make a catalog of Axiom Data Science-stored assets using `intake-axds`.

Available options

```
omsa make_catalog --project_name PROJ_NAME --catalog_type axds --catalog_name CATALOG_
↪NAME --description "Catalog description" --kwargs datatype="platform2 standard_names=
↪"[STANDARD_NAME1,STANDARD_NAME2]" page_size=PAGE_SIZE verbose=BOOL --kwargs_search min_
↪lon=MIN_LON min_lat=MIN_LAT max_lon=MAX_LON max_lat=MAX_LAT min_time=MIN_TIME max_
↪time=MAX_TIME search_for=SEARCH_TEXT --verbose --mode MODE
```

- `project_name`: Will be used as the name of the directory where the catalog is saved. The directory is located in a user application cache directory, the address of which can be found for your setup with `omsa proj_path --project_name PROJ_NAME`.
- `catalog_type`: Type of catalog to make. Options are "erddap", "axds", or "local".
- `catalog_name`: Name for catalog.
- `description`: Description for catalog.
- `metadata`: Metadata for catalog.
- `vocab_name`: Name of vocabulary to use from vocab dir. Options are "standard_names" and "general". See more information *here*.
- `kwargs`: Keyword arguments to make the AXDS catalog. See `intake-axds.axds_cat()` for more details.
  - `datatype`: Which type of Axiom asset to search for? Currently only "platform2" works and that is the default.
  - `keys_to_match`: Name of keys to match with system-available variable parameterNames using criteria. To filter search by variables, either input keys_to_match and a vocabulary or input standard_names.
  - `standard_names`: Standard names to select from Axiom search parameterNames. To filter search by variables, either input keys_to_match and a vocabulary or input standard_names.
  - `page_size`: Number of results. Fewer is faster. Note that default is 10. Note that if you want to make sure you get all available datasets, you should input a large number like 50000.
  - `verbose`: Set to True for helpful information.
  - other keyword arguments can be passed into the intake `Catalog` class
- `kwargs_search`: Keyword arguments to input to search on the server before making the catalog.
  - `min_lon`, `min_lat`, `max_lon`, `max_lat`: search for datasets within this spatial box
  - `min_time`, `max_time`: search for datasets with data within this time range
  - `model_name`: input a path to the model output to instead select the space and time search specifications based on the model. This input is specific to OMSA, not `intake-axds`.
  - `search_for`: text-based search
- `verbose` Print useful runtime commands to stdout if True as well as save in log, otherwise silently save in log. Log is located in the project directory, which can be checked on the command line with `omsa proj_path --project_name PROJECT_NAME`. Default is True, to turn off use `--no-verbose`.
- `mode` mode for logging file. Default is to overwrite an existing logfile, but can be changed to other modes, e.g. "a" to instead append to an existing log file.

Examples

Many of the options available for an Axiom catalog are the same as for an ERDDAP catalog, so we show only a few combined examples here.

Narrow search by input selections

Select a box and time range over which to search catalog along with standard_name selections, with `verbose=True`.

```
omsa make_catalog --project_name test1 --catalog_type axds --catalog_name example_axds_
↪catalog1 --description "Example AXDS catalog description" --kwargs page_size=50000␣
↪standard_names="[sea_water_temperature]" verbose=True  --kwargs_search min_lon=-170␣
↪min_lat=53 max_lon=-165 max_lat=56 min_time=2000-1-1 max_time=2002-1-1
```

Same but with vocab

As in the ERDDAP catalog example above, we can instead get the same results by inputting a vocabulary to use, in this case "standard_names" which will map to variable names in Axiom systems, along with the variable nickname from the vocabulary to find: "temp".

```
omsa make_catalog --project_name test1 --catalog_type axds --catalog_name example_axds_
↪catalog2 --description "Example AXDS catalog description" --vocab_name standard_names -
↪-kwargs page_size=50000 keys_to_match="[temp]" --kwargs_search min_lon=-170 min_lat=53␣
↪max_lon=-165 max_lat=56 min_time=2000-1-1 max_time=2002-1-1
```

1.6.2 Run model-data comparison

Note that if any datasets have timezones attached, they are removed before comparison with the assumption that the model output and data are in the same time zone.

For the final step of comparing the model output and datasets, we have to select a variable to compare, which means we have to select one or more vocabularies and one variable key. You should input the vocabularies used to create your catalogs and maybe also the "general" vocabulary if you used any known datasets, not from a specific server, since it then doesn't follow the same convention as the other files.

The datasets need to all cover the same time periods.

Available options

```
omsa run --project_name test1 --catalogs CATALOG_NAME1 CATALOG_NAME2 --vocab_names␣
↪VOCAB1 VOCAB2 --key KEY --model_path PATH_TO_MODEL_OUTPUT --ndatasets NDATASETS --
↪verbose --mode MODE --kwargs_map key_fig=value_fig --more_kwargs key=value key2=value2
```

- `project_name`: Subdirectory in cache dir to store files associated together.
- `catalog_names`: Catalog name(s). Datasets will be accessed from catalog entries.
- `vocab_names`: Criteria to use to map from variable to attributes describing the variable. This is to be used with a key representing what variable to search for. This input is for the name of one or more existing vocabularies which are stored in a user application cache.
- `key`: Key in vocab(s) representing variable to compare between model and datasets.

- `model_name`: name of the model catalog we previously created
- `ndatasets`: Max number of datasets from each input catalog to use.
- `verbose` Print useful runtime commands to stdout if True as well as save in log, otherwise silently save in log. Log is located in the project directory, which can be checked on the command line with `omsa proj_path --project_name PROJECT_NAME`. Default is True, to turn off use `--no-verbose`.
- `mode` mode for logging file. Default is to overwrite an existing logfile, but can be changed to other modes, e.g. "a" to instead append to an existing log file.
- `kwargs_map` are sent to `omsa.plot.map`
- `more_kwargs` are sent to `omsa.run`

Example

Run a model-data comparison for the first 3 datasets in each of the 3 catalogs that we created previously in this notebook. Use vocabularies `standard_names` and `general` for variable matching. Match on the temperature variable with variable nickname "temp".

This example doesn't fully work because the combination of datasets are at different time periods and of different types that don't make sense to compare with the model output. It is shown as a template.

```
omsa run --project_name test1 --catalog_names example_local_catalog example_erddap_
↪catalog example_axds_catalog1 --vocab_names standard_names general --key temp --model_
↪name model --ndatasets 1
```

1.6.3 Utilities

A few handy utilities.

Check location of project

With this you can check all of the project-related files you've created.

```
omsa proj_path --project_name test1
```

Check available vocabularies

```
omsa vocabs
```

Get information about a vocabulary

Return the path to the vocab file and the nicknames of the variables in the file.

```
omsa vocab_info --vocab_name general
```

1.7 Developer documentation

1.7.1 Running tests

You can run the full suite of tests from the base directory with `pytest`. Some tests compare images. To run these someone needs to have previously created expected images for comparison which they can do by running

```
pytest --mpl-generate-path=tests/baseline
```

After checking these, they need to be committed to the repository for future image comparisons. To run the image comparison tests, run

```
pytest --mpl
```

1.7.2 Updating docs

The demo notebooks are compiled by ReadTheDocs with Myst-NB and jupytext. These packages allow for a 1-1 mapping between a Jupyter notebook and a markdown file that can be interpreted for compilation. The markdown version of each demo is what is git-tracked because changes can be tracked better in that format. Note that currently a couple of the notebooks are in fact being stored as Jupyter notebooks so that they can be run locally.

The steps for updating demo pages:

1. If you don't already have a notebook version of the demo you want to update, convert from markdown to notebook with `jupytext [filename].md --to ipynb`.
2. Update notebook.
3. Convert to markdown with `jupytext [filename].ipynb --to myst`.
4. Git commit and push the markdown file.

1.7.3 Vocab demo page

The docs page *How to make and work with vocabularies and vocab labels* is set up differently from the other pages because it contains a widget. Most of the page is an ipython notebook that has been converted to markdown with `Myst-NB`, but the final cell in the page is a raw .html file. One can save the widget state in Jupyter lab under "Settings > Save widget state automatically". It is supposed to be possible to then present the basic widget state with Sphinx but I was not able to get it to work with this setup. However, I was able to export from the "vocab_widget.ipynb" notebook to html and have that properly retain the widget state. As a workaround, then, I embedded the html version of "vocab_widget" in add_vocab.md. Hopefully it will not need to be changed often. That is also why a .ipynb file is saved for the "vocab_widget" demo.

1.7.4 Roadmap

Next steps:

- Extend to be able to compare model output with other dataset types:
  - time series at other depths than surface
  - 2D surface fields and other depth slices
- Handle units (right now assumes units are the same in model and datasets and match what is input with `vocab_labels` for labels on plots)
- Handle time zones. Currently assumes everything in UTC. Removes timezones if present.

- Make dataset handling more flexible such that if a dataset featuretype is amenable, don't require T, Z, lon, or lat to be in separate columns. Currently all are required but in the future e.g. a `timeSeries` dataset could only have the depth defined in the catalog metadata since it doesn't vary.
- add functions to test_plot.py to test scatter and pcolormesh
- expand docs to show other model-data comparison examples, possibly using synthetic data

1.8 What's New

1.8.1 v1.2.0 (November 27, 2023)

- Added capability for running HF Radar as quiver plot, over time or single time.

1.8.2 v1.1.0 (October 13, 2023)

- Continuing to improve functionality of flags to be able to control how model output is extracted
- making code more robust to different use cases

1.8.3 v1.0.0 (October 5, 2023)

- more modularized code structure with much more testing
- requires datasets to include catalog metadata of NCEI feature type and maptype (for plotting):
  - feature types currently included:
    - timeSeries
    - profile
    - trajectoryProfile
    - timeSeriesProfile
  - To be added: grid
- added option for user to input labels for vocab keys to be used in plots
- configuration for handling featuretypes is in `featuretype.py` and `plot.__init__`.
- Added images-based tests for each featuretype, which can be run to compare against expected images with `pytest --mpl`. There is a developer section in the documentation with instructions.
- Added checks for what DataFrame datasets need to include and what catalog source metadata needs to include
- expanded documentation on requirements and information for datasets and catalogs, including using and understanding NCEI feature types
- Full update of docs

1.8.4 v0.9.0 (September 15, 2023)

- improved index handling

1.8.5 v0.8.0 (September 11, 2023)

- `omsa.run` now saves the polygon found for the input model into the project directory.
- bunch of other changes

PYTHON MODULE INDEX

O

INDEX

Symbols

A

C

F

G

K

L

M

O

P

R

S

V