

# Seasonal Forecast Verification

## Contents

- Learning objectives 🧠
- Target audience 🎯
- Prerequisites and assumed knowledge 🔍
- Outline
- 1. Import the necessary libraries
- 2. Access data with the CDS API
- 3. Compute deterministic and probabilistic products from the hindcast data
- 4. Compute deterministic and probabilistic scores
- 5. Visualize verification plots
- Key Messages to Take Home 📌



PROGRAMME OF  
THE EUROPEAN UNION



This notebook provides a practical introduction on how to produce some verification metrics and scores for seasonal forecasts with data from the [Copernicus Climate Change Service \(C3S\)](#). C3S seasonal forecast products are based on data from several state-of-the-art seasonal prediction systems. In this notebook, as an example, we will focus on data produced by [CMCC SPSv3.5 system](#), which is one of the forecasting systems available through C3S.

The tutorial will demonstrate how to access retrospective forecast (hindcast) data of 2-metre temperature initialized in the period 1993-2016, with a forecast start date in the 1st of March. All these forecasts are 6 months long (from March to August). More details about the role of the hindcasts can be found in [this Copernicus Knowledge Base article](#). Observation data (ERA5 reanalysis) for the same reference period and the same months will also be obtained from the CDS. The tutorial will then show how to compute some deterministic products

(anomalies) and some probabilistic products (probabilities for tercile categories). In addition to the 1-month average data retrieved from the CDS, 3-months aggregations will be also produced. Finally, verification metrics (correlation, area under the ROC curve, and RPS) will be calculated and visualised in a set of plots.

Please see here the full documentation of the [C3S Seasonal Forecast Datasets](#). This notebook will use data from the CDS dataset [seasonal forecast monthly statistics on single levels](#) (as opposed to multiple levels in the atmosphere).

**NOTE:**

An example on the use of the code of this notebook to create verification plots for the operational seasonal forecast systems available in the CDS can be found in a [dedicated page](#) of the documentation hosted in the Copernicus Knowledge Base.

## Learning objectives

1. Learn how to calculate monthly and 3-monthly anomalies.
2. Learn how to compute probabilities for tercile categories.
3. Learn to calculate and visualize the verification metrics correlation, area under the ROC curve and RPS.

## Target audience

**Anyone** interested in learning how to calculate seasonal forecast anomalies and to make seasonal forecast verification.

## Prerequisites and assumed knowledge

1. **Programming Skills:** Familiarity with programming concepts, particularly in Python, as the tutorial involves using loops, defining functions and using multiple different libraries for data manipulation and visualization.

2. **Familiarity with API Usage:** Understanding of how to use Application Programming Interfaces (APIs) will be useful for accessing data through the CDS API.
3. **Familiarity with multidimensional data structure:** Comprehending what are data dimensions and how they are organized in an array.



### Difficulty

4/5

## Run the tutorial





# WEKEO

[WEKEO](#) serves as the official platform of the European Centre for Medium-Range Weather Forecasts (ECMWF), offering access to an extensive array of climate data and tools for analysis and visualization. It provides a robust environment for conducting in-depth analysis and exploration of climate-related datasets. To learn more about WEKEO and its offerings, visit their [website](#).

[Run this notebook on WEKEO](#)

## Possible Cloud Services

While Kaggle, Binder, and Colab are popular options for running notebooks in the cloud, it's essential to note that these are just a few among many available choices. Each platform has its unique features and capabilities, catering to diverse user needs and preferences.

Kaggle	Binder	Colab
 <a href="#">Open in Kaggle</a>	<a href="#">launch</a> <a href="#">binder</a>	 <a href="#">Open in Colab</a>

# Outline

1. Import the necessary libraries
2. Access data with the CDS API
  1. Retrieve hindcast data
  2. Retrieve observations data (ERA5)
3. Compute deterministic and probabilistic products from the hindcast data
  1. Anomalies
  2. Probabilities for tercile categories
4. Compute deterministic and probabilistic scores
  1. Read observations data into a xr.Dataset
  2. Compute deterministic scores
  3. Compute probabilistic scores for tercile categories
5. Visualize verification plots
  1. Correlation
  2. Ranked Probability Score (RPS)
  3. Area under Relative Operating Characteristic (ROC) curve

**WARNING:**

Please take into account that to run this tutorial, around 8GB of RAM and around 5GB of disk space are needed.

## 1. Import the necessary libraries

For downloading the data from the CDS we will need [cdsapi](#). We will be working with data in GRIB format, which we will need to map to the NetCDF Common Data Model. For that, we will need [cfrib](#). To handle NetCDF, we will use [xarray](#), and to deal with dataframes, [pandas](#). We will also need libraries for plotting and viewing data, in this case we will use [matplotlib](#) and [cartopy](#). [xskillscore](#) will be used to calculate the verification metrics.

If you don't have any of the following libraries, you must install it. You can do it like this:

```
!pip install xskillscore
```

```
Requirement already satisfied: xskillscore in /Library/Frameworks/Python.fra
Requirement already satisfied: dask[array] in /Library/Frameworks/Python.fra
Requirement already satisfied: numpy in /Library/Frameworks/Python.framework
Requirement already satisfied: properscoring in /Library/Frameworks/Python.f
Requirement already satisfied: scipy in /Library/Frameworks/Python.framework
Requirement already satisfied: statsmodels in /Library/Frameworks/Python.fra
Requirement already satisfied: xarray>=0.16.1 in /Library/Frameworks/Python.
Requirement already satisfied: xhistogram>=0.3.0 in /Library/Frameworks/Pyth
Requirement already satisfied: packaging>=23.1 in /Library/Frameworks/Python
Requirement already satisfied: pandas>=2.0 in /Library/Frameworks/Python.fra
Requirement already satisfied: click>=8.1 in /Library/Frameworks/Python.fram
Requirement already satisfied: cloudpickle>=1.5.0 in /Library/Frameworks/Pyt
Requirement already satisfied: fsspec>=2021.09.0 in /Library/Frameworks/Pyth
Requirement already satisfied: partd>=1.2.0 in /Library/Frameworks/Python.fr
Requirement already satisfied: pyyaml>=5.3.1 in /Library/Frameworks/Python.f
Requirement already satisfied: toolz>=0.10.0 in /Library/Frameworks/Python.f
Requirement already satisfied: patsy>=0.5.6 in /Library/Frameworks/Python.fr
Requirement already satisfied: python-dateutil>=2.8.2 in /Library/Frameworks
Requirement already satisfied: pytz>=2020.1 in /Library/Frameworks/Python.fr
Requirement already satisfied: tzdata>=2022.7 in /Library/Frameworks/Python.
Requirement already satisfied: locket in /Library/Frameworks/Python.framework
Requirement already satisfied: six in /Library/Frameworks/Python.framework/V
```

```
# CDS API
import cdsapi

# Libraries for working with multi-dimensional arrays
import xarray as xr
import pandas as pd
import numpy as np

# Forecast verification metrics with xarray
import xskillscore as xs

# Date and calendar libraries
from dateutil.relativedelta import relativedelta
import calendar

# Libraries for plotting and geospatial data visualisation
from matplotlib import pyplot as plt
import cartopy.crs as ccrs
import cartopy.feature as cfeature

# Disable warnings for data download via API and matplotlib (do I need both?)
import warnings
warnings.filterwarnings('ignore')
```

## 2. Access data with the CDS API

First we will set up some common variables to be used in this notebook to define the folder containing the data, the variables to be retrieved, the hindcast period and the [nominal start month](#).

```
DATADIR = './data'

config = dict(
    list_vars = ['2m_temperature', ],
    hcstarty = 1993,
    hcendy = 2016,
    start_month = 3,
)
```

After setting up this initial configuration variables, the existence of all the data folders will be checked and directories will be created if needed.

```
import os

SCOREDIR = DATADIR + '/scores'
PLOTSDIR = DATADIR + f'/plots/stmonth{config["start_month"]:02d}'

for directory in [DATADIR, SCOREDIR, PLOTSDIR]:
    # Check if the directory exists
    if not os.path.exists(directory):
        # If it doesn't exist, create it
        os.makedirs(directory)
        print(f'Creating folder {directory}')
```

To access data from the CDS, you will need first to [register](#) at the ADS registration page (if you haven't already done so), [log in](#), and accept the Terms and Conditions at the end of the [Download data](#) tab.

To obtain data programmatically from the CDS, you will need an API Key that can be found in the [api-how-to page](#). Here your key will appear automatically in the black window, assuming you have already registered and logged into the ADS. Your API key is the entire string of characters that appears after `key:`

Now copy your API key into the code cell below, replacing `#####` with your key.

```
CDSAPI_URL = 'https://cds.climate.copernicus.eu/api/v2'
CDSAPI_KEY = '#####'

c = cdsapi.Client(url=CDSAPI_URL, key=CDSAPI_KEY)
```

## 2.A. Retrieve hindcast data

The next step is then to request the seasonal forecast monthly statistics data on single levels with the help of the CDS API. Below, we download a GRIB file containing the retrospective forecasts (hindcasts, or reforecasts) for the variables, start month and years defined in the variable `config`

Running the code block below will download the data from the CDS as specified by the following API keywords:

**Format:** `Grib`

**Originating centre:** `CMCC`

**System:** `35` *this refers to CMCC SPSv3.5*

**Variable:** `2-metre temperature`

**Product type:** `Monthly mean` *all ensemble members will be retrieved*

**Year:** `1993 to 2016`

**Month:** `03` *March*

**Leadtime month:** `1 to 6` *All available lead times (March to August)*

Note we will also be defining some variable, `origin`, with the forecast system details to be included in the `config` variable. Additionally it defines a base name variable `hcst_bname` to be used throughout the notebook.

### NOTE:

The API request below can be generated automatically from the [CDS download page](#). At the end of the download form there is a `Show API request` icon, which allows a copy-paste of the code below. At this point, remember to accept the Terms and Conditions!

```
origin = 'cmcc.s35'
origin_labels = {'institution': 'CMCC', 'name': 'SPSv3.5'}

config['origin'], config['system'] = origin.split('.')

hcst_bname = '{origin}_s{system}_stmonth{start_month:02d}_hindcast{hcstarty}'
hcst_fname = f'{DATADIR}/{hcst_bname}.grib'
print(hcst_fname)

c.retrieve(
    'seasonal-monthly-single-levels',
    {
        'format': 'grib',
        'originating_centre': config['origin'],
        'system': config['system'],
        'variable': config['list_vars'],
        'product_type': 'monthly_mean',
        'year': ['{}'.format(yy) for yy in range(config['hcstarty'], config['
        'month': '{}:02d}'.format(config['start_month']),
        'leadtime_month': ['1', '2', '3', '4', '5', '6'],
    },
    hcst_fname)
```

```
2024-07-04 13:01:09,856 INFO Welcome to the CDS
2024-07-04 13:01:09,857 INFO Sending request to https://cds.climate.copernic
```

```
./data/cmcc_s35_stmonth03_hindcast1993-2016_monthly.grib
```

```
2024-07-04 13:01:10,089 INFO Request is completed
2024-07-04 13:01:10,090 INFO Downloading https://download-0018.copernicus-cl
2024-07-04 13:06:09,177 INFO Download rate 3.6M/s
```

```
Result(content_length=1120527360,content_type=application/x-grib,location=ht
```

## 2.B. Retrieve observations data (ERA5)

Now we will request from the CDS the observation data that will be used as the ground truth against which the hindcast data will be compared. In this notebook we will be using as



observational reference the CDS dataset

`reanalysis-era5-single-levels-monthly-means` which contains [ERA5](#) monthly averaged data from the CDS. In order to compare it with the hindcast data we have just downloaded above, we will ask the CDS to regrid it to the same grid used by the C3S seasonal forecasts.

Running the code block below will download the ERA5 data from the CDS as specified by the following API keywords:

**Format:** `Grib`  
**Variable:** `2-metre temperature`  
**Product type:** `Monthly averaged reanalysis`  
**Year:** `1993 to 2016`  
**Month:** `March to August` *same months contained in the hindcast data*  
**Time:** `00`  
**Grid:** `1x1-degree lat-lon regular grid`  
**Area:** `-89.5 to 89.5 in latitude; 0.5 to 359.5 in longitude`

```
obs_fname = '{fpath}/era5_monthly_stmonth{start_month:02d}_{hcstarty}-{hcendy}.grib'
print(obs_fname)

c.retrieve(
    'reanalysis-era5-single-levels-monthly-means',
    {
        'product_type': 'monthly_averaged_reanalysis',
        'variable': config['list_vars'],
        # NOTE from observations we need to go one year beyond so we have av
        # e.g. Nov.2016 start date forecast goes up to April 2017
        'year': ['{}'.format(yy) for yy in range(config['hcstarty'],config['hcendy']+1)],
        'month': ['{:02d}'.format((config['start_month']+leadm)%12) if confi
        'time': '00:00',
        # We can ask CDS to interpolate ERA5 to the same grid used by C3S se
        'grid': '1/1',
        'area': '89.5/0.5/-89.5/359.5',
        'format': 'grib',
    },
    obs_fname)
```

2024-07-04 13:06:50,265 INFO Welcome to the CDS

2024-07-04 13:06:50,266 INFO Sending request to https://cds.climate.copernic

```
./data/era5_monthly_stmonth03_1993-2016.grib
```

```
2024-07-04 13:06:50,447 INFO Downloading https://download-0015-clone.coperni
2024-07-04 13:06:56,549 INFO Download rate 3M/s
```

```
Result(content_length=19456200, content_type=application/x-grib, location=http
```

### 3. Compute deterministic and probabilistic products from the hindcast data

In this section we will be calculating the different derived products we will be scoring later in the notebook. We will use the reference period defined in

`config["hcstarty"]`, `config["hcendy"]` to calculate:

- anomalies: defined as the difference of a given hindcast to the model climate, computed as the average over all the available members (that is, among all start years and ensemble members)
- probabilities for tercile categories: defined as the proportion of members in a given hindcast lying within each one of the categories bounded by the terciles computed from all available members in the reference period

In addition to the 1-month anomalies and probabilities, a rolling average of 3 months will be also computed to produce anomalies and probabilities for this 3-months aggregation.

**NOTE:**

Some of the seasonal forecast monthly data on the CDS comes from systems using members initialized on different start dates (lagged start date ensembles). In the GRIB encoding used for those systems we will therefore have two different `xarray/cfgrib` keywords for the real start date of each member (`time`) and for the nominal start date (`indexing_time`) which is the one we would need to use for those systems initializing their members with a lagged start date approach. The following line of code will take care of that as long as we include the value `config['isLagged']=True` in the config dictionary as defined in section 2.

```
# For the re-shaping of time coordinates in xarray.Dataset we need to select
# -> burst mode ensembles (e.g. ECMWF SEAS5) use "time". This is the default
# -> lagged start ensembles (e.g. MetOffice GloSea6) use "indexing_time" (s
st_dim_name = 'time' if not config.get('isLagged', False) else 'indexing_time'
```

## 3.A. Anomalies

We will start opening the hindcast data GRIB file we downloaded in the previous section to load it into an `xarray.Dataset` object.

Some minor modifications of the `xr.Dataset` object also happen here:

- We define chunks on the forecastMonth (leadtime) coordinate so `dask.array` can be used behind the scenes.
- We rename some coordinates to align them with the names they will have in the observations `xr.Dataset` and those expected by default in the `xskillscore` package we will be using for the calculation of scores.
- We create a new array named `valid_time` to be a new coordinate (depending on both `start_date` and `forecastMonth`).

After that, both anomalies for 1-month and 3-months aggregations are computed and stored into a netCDF file.

```

# Reading HCST data from file
hcst = xr.open_dataset(hcst_fname, engine='cfgrib', backend_kwargs=dict(time_
hcst = hcst.chunk({'forecastMonth':1, 'latitude':'auto', 'longitude':'auto'})
hcst = hcst.rename({'latitude':'lat', 'longitude':'lon', 'st_dim_name':'start_d

# Re-arranging time metadata in xr.Dataset object

## Add start_month to the xr.Dataset
start_month = pd.to_datetime(hcst.start_date.values[0]).month
hcst = hcst.assign_coords({'start_month':start_month})

## Add valid_time to the xr.Dataset
vt = xr.DataArray(dims=('start_date', 'forecastMonth'), coords={'forecastMont
vt.data = [[pd.to_datetime(std)+relativedelta(months=fcmonth-1) for fcmonth
hcst = hcst.assign_coords(valid_time=vt)

# CALCULATE 3-month AGGREGATIONS
# NOTE rolling() assigns the label to the end of the N month period, so the
hcst_3m = hcst.rolling(forecastMonth=3).mean()
hcst_3m = hcst_3m.where(hcst_3m.forecastMonth>=3, drop=True)

# CALCULATE ANOMALIES (and save to file)
## Computing anomalies 1m
hcmean = hcst.mean(['number', 'start_date'])
anom = hcst - hcmean
anom = anom.assign_attrs(reference_period='{hcstarty}-{hcentdy}'.format(**con

## Computing anomalies 3m
hcmean_3m = hcst_3m.mean(['number', 'start_date'])
anom_3m = hcst_3m - hcmean_3m
anom_3m = anom_3m.assign_attrs(reference_period='{hcstarty}-{hcentdy}'.format

## Saving anomalies 1m/3m to netCDF files
anom.to_netcdf(f'{DATADIR}/{hcst_bname}.1m.anom.nc')
anom_3m.to_netcdf(f'{DATADIR}/{hcst_bname}.3m.anom.nc')

```

```
Ignoring index file './data/cmcc_s35_stmonth03_hindcast1993-2016_monthly.gri
```

## 3.B. Probabilities for tercile categories

Before we start computing the probabilities it will be useful to create a function devoted to computing the boundaries of a given category (`icat`) coming from a given set of quantiles (`quantiles`) of an `xr.Dataset` object.

```
# We define a function to calculate the boundaries of forecast categories de
def get_thresh(ocat, quantiles, xrds, dims=['number', 'start_date']):

    if not all(elem in xrds.dims for elem in dims):
        raise Exception('Some of the dimensions in {} is not present in the
    else:
        if ocat == 0:
            xrds_lo = -np.inf
            xrds_hi = xrds.quantile(quantiles[ocat], dim=dims)

        elif ocat == len(quantiles):
            xrds_lo = xrds.quantile(quantiles[ocat-1], dim=dims)
            xrds_hi = np.inf

        else:
            xrds_lo = xrds.quantile(quantiles[ocat-1], dim=dims)
            xrds_hi = xrds.quantile(quantiles[ocat], dim=dims)

    return xrds_lo, xrds_hi
```

We will start by creating a list with the quantiles which will be the boundaries for our categorical forecast. In this example our product will be based on 3 categories ('below lower tercile', 'normal' and 'above upper tercile') so we will be using as category boundaries the values `quantiles = [1/3., 2/3.]`

The `xr.Dataset` objects `hcst` and `hcst_3m` read in the previous step from the corresponding netCDF files will be used here, and therefore probabilities for both 1-month and 3-months aggregations are computed and stored into a netCDF file.

```
# CALCULATE PROBABILITIES for tercile categories by counting members within
quantiles = [1/3., 2/3.]
numcategories = len(quantiles)+1

for aggr,h in [("1m",hcst), ("3m",hcst_3m)]:
    # Computing tercile probabilities

    l_probs_hcst=list()
    for icat in range(numcategories):
        h_lo,h_hi = get_thresh(icat, quantiles, h)
        probh = np.logical_and(h>h_lo, h<=h_hi).sum('number')/float(h.dims['
        # Instead of using the coordinate 'quantile' coming from the hindcas
        # we will create a new coordinate called 'category'
        if 'quantile' in probh:
            probh = probh.drop('quantile')
        l_probs_hcst.append(probh.assign_coords({'category':icat}))

    # Concatenating tercile probs categories
    probs = xr.concat(l_probs_hcst,dim='category')
    # Saving tercile probs netCDF files
    probs.to_netcdf(f'{DATADIR}/{hcst_bname}.{aggr}.tercile_probs.nc')
```

## 4. Compute deterministic and probabilistic scores

In this section we will produce a set of metrics and scores consistent with the [guidance on forecast verification](#) provided by WMO. Those values can be used to describe statistics of forecast performance over a past period (the reference period defined in the variable `config`), as an indication of expectation of ‘confidence’ in the real-time outputs.

In this notebook metrics and scores for both deterministic (anomalies) and probabilistic (probabilities of tercile categories) products, as those provided in the C3S seasonal forecast graphical products section, will be computed. Specifically:

- Deterministic:
  - linear temporal (Pearson) correlation
- Probabilistic:
  - area under the ROC curve (relative, or ‘receiver’, operating characteristic)
  - RPS (ranked probability score)

Values for both 1-month and 3-months aggregations will be produced and stored into netCDF files.

---

## 4.A. Read observations data into a `xr.Dataset`

Before any score can be calculated we will need to read into an `xr.Dataset` object the observational data from the GRIB file we downloaded in the first section of this notebook (whose name was stored in variable `obs_fname`).

Similarly to what we did with the hindcast data, some minor modifications of the `xr.Dataset` object also happen here:

- We rename some coordinates to align them with the names in the hindcast `xr.Dataset` and those expected by default in the `xskillscore` package we will be using for the calculation of scores.
- We index the reanalysis data with `forecastMonth` to allow an easy like-for-like comparison of valid times.

In addition to the monthly averages we retrieved from the CDS, a rolling average of 3 months is also computed to use as observational reference for the 3-months aggregation on the hindcasts.

```

era5_1deg = xr.open_dataset(obs_fname, engine='cfgrib')

# Renaming to match hindcast names
era5_1deg = era5_1deg.rename({'latitude':'lat','longitude':'lon','time':'sta

# Assign 'forecastMonth' coordinate values
fcmonths = [mm+1 if mm>=0 else mm+13 for mm in [t.month - config['start_mont
era5_1deg = era5_1deg.assign_coords(forecastMonth=('valid_time',fcmonths))
# Drop obs values not needed (earlier than first start date) - this is usefu
era5_1deg = era5_1deg.where(era5_1deg.valid_time>=np.datetime64('{hcstarty}-

# CALCULATE 3-month AGGREGATIONS
# NOTE rolling() assigns the label to the end of the N month period
# NOTE care should be taken with the data available in the "obs" xr.Dataset
era5_1deg_3m = era5_1deg.rolling(valid_time=3).mean()
era5_1deg_3m = era5_1deg_3m.where(era5_1deg_3m.forecastMonth>=3)

# As we don't need it anymore at this stage, we can safely remove 'forecastM
era5_1deg = era5_1deg.drop('forecastMonth')
era5_1deg_3m = era5_1deg_3m.drop('forecastMonth')

```

Ignoring index file './data/era5\_monthly\_stmonth03\_1993-2016.grib.9093e.idx'

## 4.B. Compute deterministic scores

The score used here is temporal correlation (Spearman rank correlation) calculated at each grid point and for each leadtime.

We will start reading the netCDF files containing the anomalies we computed in section 3 of this notebook into a `xr.Dataset` object.

For an easier use of the `xs.spearman_r` function a loop over the hindcast leadtimes (`forecastMonth`) has been introduced, requiring as a final step a concatenation of the values computed for each leadtime.

In addition to the correlation, the p-values will be computed to allow us plotting not only the values of correlation, but also where those values are statistically significant.

### NOTE:

The computations shown here are prepared to cater for anomalies files containing all



ensemble members or files containing only an ensemble mean anomaly. The presence (or absence) of a dimension called `number` in the hindcast `xr.Dataset` object will indicate we have a full ensemble (or an ensemble mean).

```
# Loop over aggregations
for aggr in ['1m', '3m']:

    if aggr=='1m':
        o = era5_1deg
    elif aggr=='3m':
        o = era5_1deg_3m
    else:
        raise BaseException(f'Unknown aggregation {aggr}')

    # Computing deterministic scores

    # Read anomalies file
    h = xr.open_dataset(f'{DATADIR}/{hcst_bname}.{aggr}.anom.nc' )
    is_fullensemble = 'number' in h.dims

    l_corr=list()
    l_corr_pval=list()

    for this_fcmonth in h.forecastMonth.values:
        thishcst = h.sel(forecastMonth=this_fcmonth).swap_dims({'start_date'
        thisobs = o.where(o.valid_time==thishcst.valid_time,drop=True)
        thishcst_em = thishcst if not is_fullensemble else thishcst.mean('nu
        l_corr.append( xs.spearman_r(thishcst_em, thisobs, dim='valid_time')
        l_corr_pval.append ( xs.spearman_r_p_value(thishcst_em, thisobs, dim

    # Concatenating (by fcmonth) correlation
    corr=xr.concat(l_corr,dim='forecastMonth')
    corr_pval=xr.concat(l_corr_pval,dim='forecastMonth')

    # Saving to netCDF file correlation
    corr.to_netcdf(f'{DATADIR}/scores/{hcst_bname}.{aggr}.corr.nc')
    corr_pval.to_netcdf(f'{DATADIR}/scores/{hcst_bname}.{aggr}.corr_pval.nc')
```

## 4.C. Compute probabilistic scores for tercile categories

The scores used here for probabilistic forecasts are the area under the Relative Operating Characteristic (ROC) curve, the Ranked Probability Score (RPS) and the Brier Score (BS). Note that both ROC and BS are appropriate for binary events (applied individually to each

category), while RPS scores all categories together.

We will now read the netCDF files containing the probabilities we computed in section 2 of this notebook into a `xr.Dataset` object. Note the variables `quantiles` and `numcategories` defined there and describing the categories for which the probabilities have been computed will be also needed here.

The step to compute the probabilities for each category in the observation data for each aggregation (1-month or 3-months) is also performed here starting from the `xr.Dataset` objects read in section 3a. (`era5_1deg` and `era5_1deg_3m`).

A loop over the hindcast leadtimes (`forecastMonth`) has been introduced for an easier use of the `xskillscore` functions to compute ROC curve area and RPS. This requires as a final step a concatenation of the values computed for each leadtime.

**NOTE:**

A couple of additional examples, both the calculation of Brier Score (BS) and the calculation of ROC Skill Score using climatology as a reference (which has a ROC value of 0.5), have been also included in the code below, but no plots will be produced for them in section 4 of the notebook.

```

# Loop over aggregations
for aggr in ['1m', '3m']:

    if aggr=='1m':
        o = era5_1deg
    elif aggr=='3m':
        o = era5_1deg_3m
    else:
        raise BaseException(f'Unknown aggregation {aggr}')

    # Computing deterministic scores
    # READ hindcast probabilities file
    probs_hcst = xr.open_dataset(f'{DATADIR}/{hcst_bname}.{aggr}.tercile_pro

    l_roc=list()
    l_rps=list()
    l_rocss=list()
    l_bs=list()
    for this_fcmonth in probs_hcst.forecastMonth.values:
        thishcst = probs_hcst.sel(forecastMonth=this_fcmonth).swap_dims({'st

        # CALCULATE probabilities (tercile categories) from observations
        l_probs_obs=list()
        thiso = o.where(o.valid_time==thishcst.valid_time,drop=True)
        for icat in range(numcategories):
            #print(f'category={icat}')
            o_lo,o_hi = get_thresh(icat, quantiles, thiso, dims=['valid_time
            probo = 1. * np.logical_and(thiso>o_lo, thiso<=o_hi)
            if 'quantile' in probo:
                probo=probo.drop('quantile')
            l_probs_obs.append(probo.assign_coords({'category':icat}))

        thisobs = xr.concat(l_probs_obs, dim='category')

        # Calculate the probabilistic (tercile categories) scores
        thisroc = xr.Dataset()
        thisrps = xr.Dataset()
        thisrocss = xr.Dataset()
        thisbs = xr.Dataset()
        for var in thishcst.data_vars:
            thisroc[var] = xs.roc(thisobs[var],thishcst[var], dim='valid_tim

            thisrps[var] = xs.rps(thisobs[var],thishcst[var], dim='valid_tim

            thisrocss[var] = (thisroc[var] - 0.5) / (1. - 0.5)
            bscat = list()
            for cat in thisobs[var].category:
                thisobscat = thisobs[var].sel(category=cat)
                thishcstcat = thishcst[var].sel(category=cat)
                bscat.append(xs.brier_score(thisobscat, thishcstcat, dim='va
            thisbs[var] = xr.concat(bscat,dim='category')

        l_roc.append(thisroc)

```

```
l_rps.append(thisrps)
l_rocss.append(thisrocss)
l_bs.append(thisbs)

# concat roc
roc=xr.concat(l_roc,dim='forecastMonth')
# concat rps
rps=xr.concat(l_rps,dim='forecastMonth')
# concat rocss
rocss=xr.concat(l_rocss,dim='forecastMonth')
# concat bs
bs=xr.concat(l_bs,dim='forecastMonth')

# writing to netcdf
rps.to_netcdf(f'{DATADIR}/scores/{hcst_bname}.{aggr}.rps.nc')
bs.to_netcdf(f'{DATADIR}/scores/{hcst_bname}.{aggr}.bs.nc')
roc.to_netcdf(f'{DATADIR}/scores/{hcst_bname}.{aggr}.roc.nc')
rocss.to_netcdf(f'{DATADIR}/scores/{hcst_bname}.{aggr}.rocss.nc')
```

## 5. Visualize verification plots

After we have computed some metrics and scores for our seasonal forecast data, in this section some examples of visualization will be introduced.

### NOTE:

In the "Introduction" section of the [C3S seasonal forecasts verification plots](#) page on the Copernicus Knowledge Base you can find some information on how these metrics and scores, consistent with the [guidance on forecast verification](#) provided by WMO, describe statistics of forecast performance over a past period, as an indication of expectation of 'confidence' in the real-time outputs.

## Common variables

We start by setting up some variables to subset the plots we will be producing in this notebook:

**aggr**: to specify the aggregation period (possible values: "1m" or "3m")

`fcmonth`: to specify the leadtime (possible values: integer from 1 to 6)

Additionally variables used for labelling the plots are also defined here. It has been also included a change in the default values of the font size of the titles to improve the visual aspect of the plots.

```
# Subset of plots to be produced
aggr = '1m'
fcmonth = 1

# Common labels to be used in plot titles
VARNAMES = {
    't2m' : '2-metre temperature',
    # 'sst' : 'sea-surface temperature',
    # 'msl' : 'mean-sea-level pressure',
    # 'tp' : 'total precipitation'
}

CATNAMES=['lower tercile', 'middle tercile', 'upper tercile']

# Change titles font size
plt.rc('axes', titlesize=20)
```

## Create plot titles base information

In the following variables we include some relevant information to be later used in the plot titles. Specifically:

- Line 1: Name of the institution and forecast system
- Line 2: Start date and valid date (with appropriate formatting depending on the aggregation, 1-month or 3-months)

```
# PREPARE STRINGS for TITLES
tit_line1 = '{institution} {name}'.format(**origin_labels)
# tit_line2_base = rf'Start month: $\bf{{calendar.month_abbr[config["start_mo
tit_line2_base = f'Start month: {calendar.month_abbr[config["start_month"]].

if aggr=='1m':
    validmonth = config['start_month'] + (fcmonth-1)
    validmonth = validmonth if validmonth<=12 else validmonth-12
#     tit_line2 = tit_line2_base + rf' - Valid month: $\bf{{calendar.month_ab
    tit_line2 = tit_line2_base + f' - Valid month: {calendar.month_abbr[validmonth]}
elif aggr=='3m':
    validmonths = [vm if vm<=12 else vm-12 for vm in [config['start_month']
    validmonths = [calendar.month_abbr[vm][0] for vm in reversed(validmonths)
#     tit_line2 = tit_line2_base + rf' - Valid months: $\bf{{"".join(validmon
    tit_line2 = tit_line2_base + f' - Valid months: {"".join(validmonths)}'
else:
    raise BaseException(f'Unexpected aggregation {aggr}')
```

## 5.A. Correlation

```

# Read the data files
corr = xr.open_dataset(f'{DATADIR}/scores/{hcst_bname}.{aggr}.corr.nc')
corr_pval = xr.open_dataset(f'{DATADIR}/scores/{hcst_bname}.{aggr}.corr_pval.nc')
# RE-ARRANGE the DATASETS longitude values for plotting purposes
corr = corr.assign_coords(lon=((corr.lon + 180) % 360) - 180).sortby('lon')
corr_pval = corr_pval.assign_coords(lon=((corr_pval.lon + 180) % 360) - 180).sortby('lon')

thiscorr = corr.sel(forecastMonth=fcmonth)
thiscorr_pval = corr_pval.sel(forecastMonth=fcmonth)

for var in thiscorr.data_vars:
    fig = plt.figure(figsize=(18,10))
    ax = plt.axes(projection=ccrs.PlateCarree())
    ax.add_feature(cfeature.BORDERS, edgecolor='black', linewidth=0.5)
    ax.add_feature(cfeature.COASTLINE, edgecolor='black', linewidth=2.)
    corrvalues = thiscorr[var].values
    corrpvalvalues = thiscorr_pval[var].values

    if corrvalues.T.shape == (thiscorr[var].lat.size, thiscorr[var].lon.size):
        print('Data values matrices need to be transposed')
        corrvalues = corrvalues.T
        corrpvalvalues = corrpvalvalues.T
    elif corrvalues.shape == (thiscorr[var].lat.size, thiscorr[var].lon.size):
        pass
    # print('Data values matrices shapes are ok')
    else:
        raise BaseException(f'Unexpected data value matrix shape: {corrvalues.shape}')

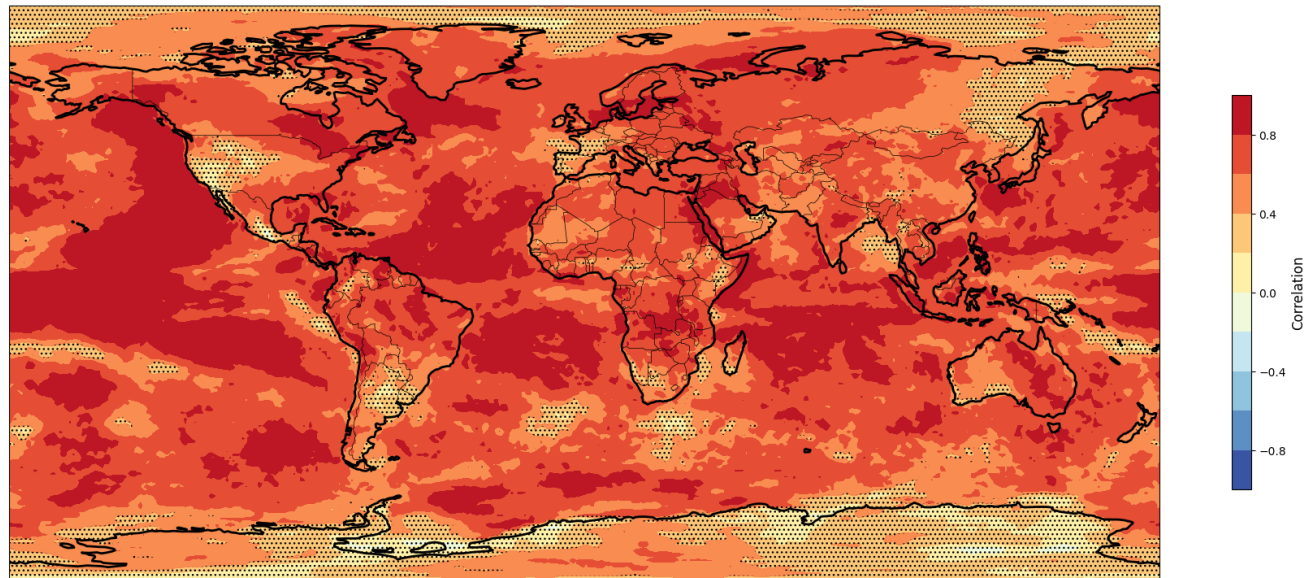
    plt.contourf(thiscorr[var].lon, thiscorr[var].lat, corrvalues, levels=np.linspace(-1, 1, 10))
    cb = plt.colorbar(shrink=0.5)
    cb.ax.set_ylabel('Correlation', fontsize=12)
    origylim = ax.get_ylim()
    plt.contourf(thiscorr_pval[var].lon, thiscorr_pval[var].lat, corrpvalvalues, levels=np.linspace(0, 1, 10))
    # We need to ensure after running plt.contourf() the ylim hasn't changed
    if ax.get_ylim() != origylim:
        ax.set_ylim(origylim)

    plt.title(tit_line1 + f' {VARNAMES[var]} ' + ' (stippling where significant)')

plt.tight_layout()
figname = f'{DATADIR}/plots/stmonth{config["start_month"]:02d}/{hcst_bname}.fig.png'
plt.savefig(figname)

```

CMCC SPSv3.5 2-metre temperature (stippling where significance below 95%)  
 Start month: MAR - Valid month: MAR



## 5.B. Ranked Probability Score (RPS)

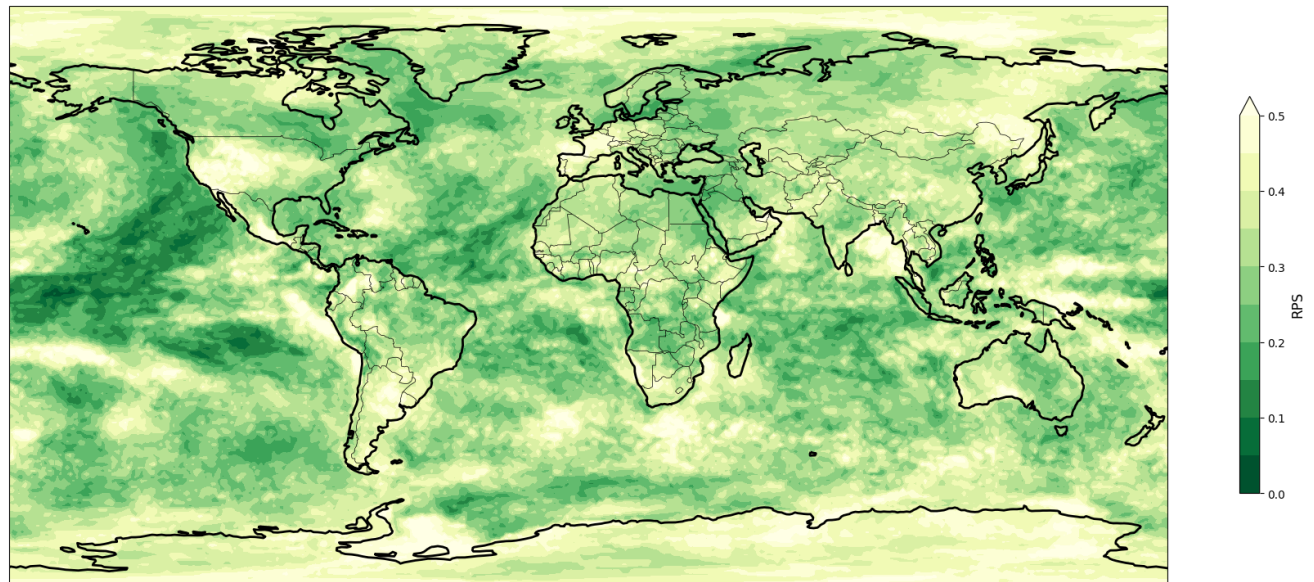
```
# READ scores xr.Datasets
rps = xr.open_dataset(f'{DATADIR}/scores/{hcst_bname}.{aggr}.rps.nc')
# RE-ARRANGE the DATASETS longitude values for plotting purposes
rps = rps.assign_coords(lon=((rps.lon + 180) % 360) - 180)).sortby('lon')

thisrps = rps.sel(forecastMonth=fcmonth)

for var in thisrps.data_vars:
    fig = plt.figure(figsize=(18,10))
    ax = plt.axes(projection=ccrs.PlateCarree())
    ax.add_feature(cfeature.BORDERS, edgecolor='black', linewidth=0.5)
    ax.add_feature(cfeature.COASTLINE, edgecolor='black', linewidth=2.)
    avalues = thisrps[var].values
    cs = plt.contourf(thisrps[var].lon, thisrps[var].lat, avalues, levels=np.li
    cs.cmap.set_under('purple')
    cb = plt.colorbar(shrink=0.5)
    cb.ax.set_ylabel('RPS', fontsize=12)
    plt.title(tit_line1 + f' {VARNAMES[var]} ' + ' (tercile categories)\n' +
    plt.tight_layout()
    figname = f'{DATADIR}/plots/stmonth{config["start_month"]:02d}/{hcst_bna
    plt.savefig(figname)
```



CMCC SPSv3.5 2-metre temperature (tercile categories)  
Start month: MAR - Valid month: MAR



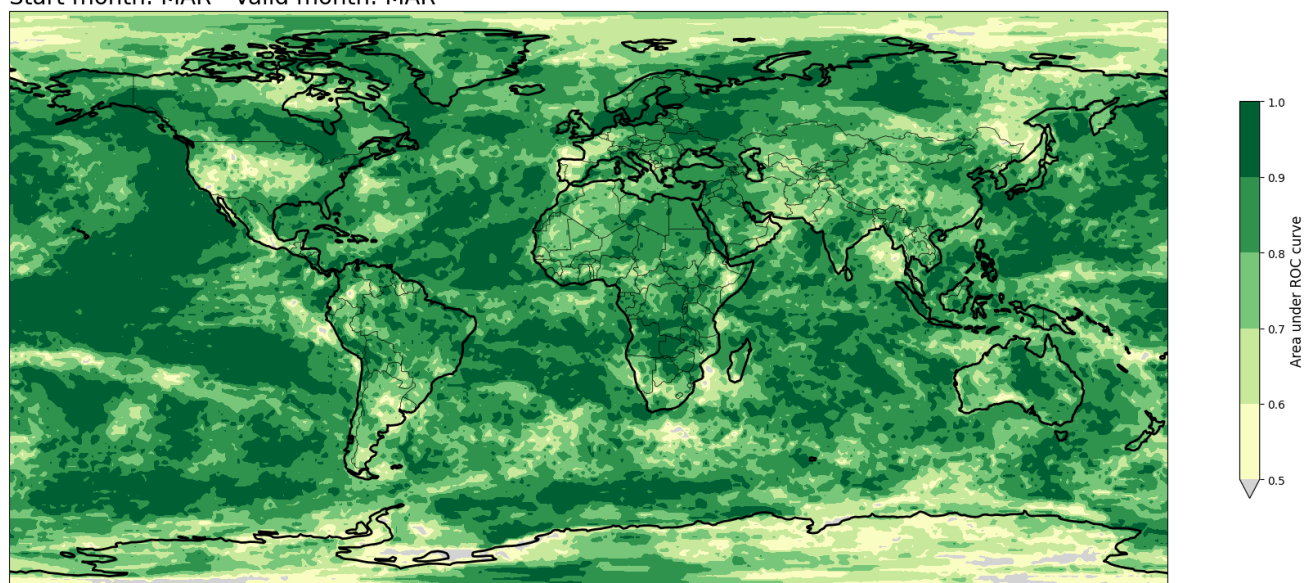
## 5.C. Area under Relative Operating Characteristic (ROC) curve

```
# READ scores xr.Datasets
roc = xr.open_dataset(f'{DATADIR}/scores/{hcst_bname}.{aggr}.roc.nc')
# RE-ARRANGE the DATASETS longitude values for plotting purposes
roc = roc.assign_coords(lon=((roc.lon + 180) % 360) - 180)).sortby('lon')

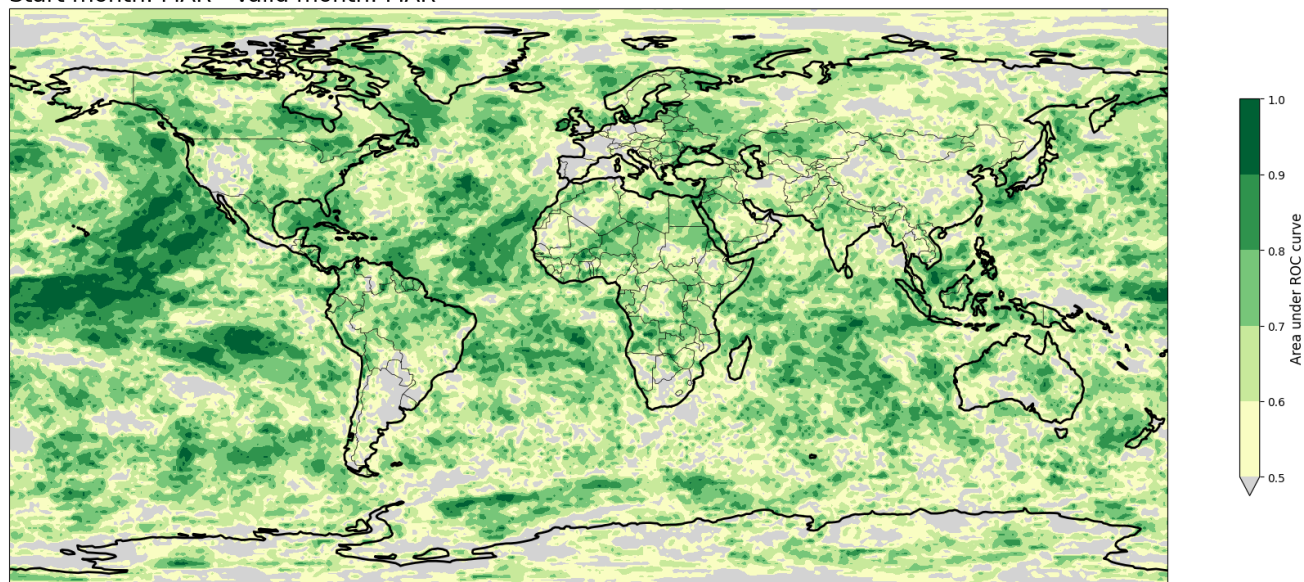
thisroc = roc.sel(forecastMonth=fcmonth)

for var in thisroc.data_vars:
    for icat in thisroc.category.values:
        fig = plt.figure(figsize=(18,10))
        ax = plt.axes(projection=ccrs.PlateCarree())
        ax.add_feature(cfeature.BORDERS, edgecolor='black', linewidth=0.5)
        ax.add_feature(cfeature.COASTLINE, edgecolor='black', linewidth=2.)
        avalues = thisroc.sel(category=icat)[var].values
        cs = plt.contourf(thisroc[var].lon, thisroc[var].lat, avalues, levels=n)
        cs.cmap.set_under('lightgray')
        cb = plt.colorbar(shrink=0.5)
        cb.ax.set_ylabel('Area under ROC curve', fontsize=12)
        plt.title(tit_line1 + f' {VARNAMES[var]} ' + f' ({CATNAMES[icat]})\n'
        plt.tight_layout()
        figname = f'{DATADIR}/plots/stmonth{config["start_month"]}:02d/{hcst}
        plt.savefig(figname)
```

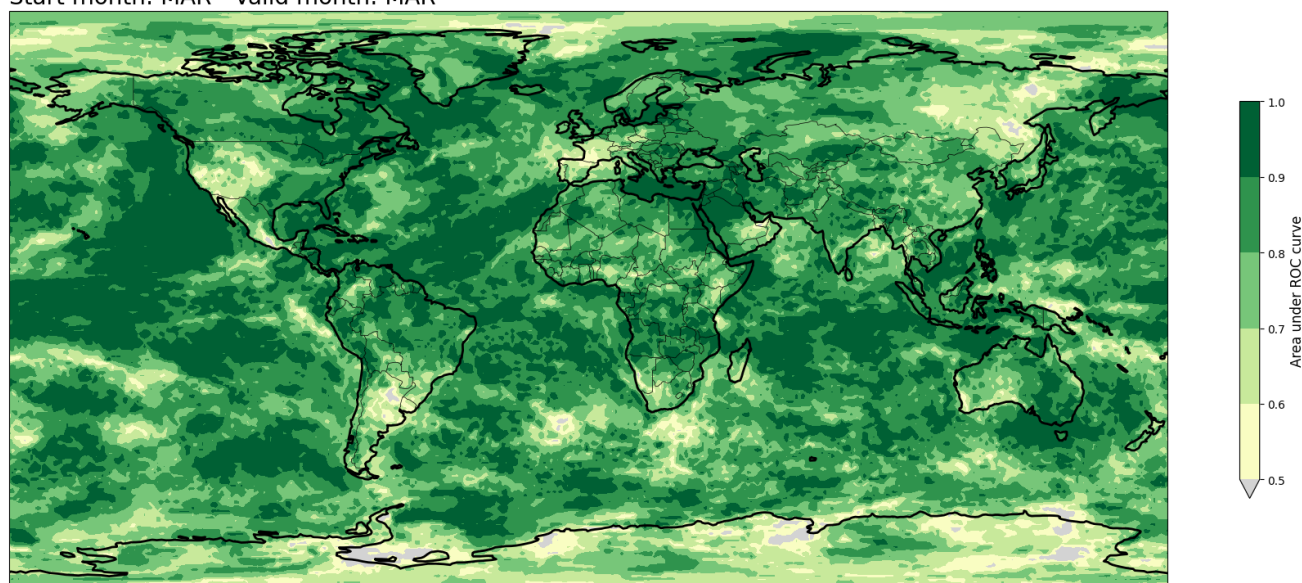
CMCC SPSv3.5 2-metre temperature (lower tercile)  
Start month: MAR - Valid month: MAR



CMCC SPSv3.5 2-metre temperature (middle tercile)  
Start month: MAR - Valid month: MAR



CMCC SPSv3.5 2-metre temperature (upper tercile)  
Start month: MAR - Valid month: MAR



## Key Messages to Take Home

- Seasonal forecast anomalies are defined as the difference of a given hindcast to the model climate, computed as the average over all the available members.
- Probabilities for tercile categories are defined as the proportion of members in a given hindcast lying within each one of the categories bounded by the terciles computed from all available members in the reference period.
- Specific verification metrics and scores are to be computed for both deterministic (anomalies) and probabilistic (probabilities of tercile categories).
- All computed metrics and scores must be spatially visualized.
- Information on how these metrics and scores work as indicators of 'confidence' of the real-time forecast can be found in the "Introduction" section of the [C3S seasonal forecasts verification plots](#) page on the Copernicus Knowledge Base.