CRA2022_Python_plotting_tools

July 19, 2022

1 Python tools for nice plots

Climate Risk Assessment 2022, Jérôme Kopp (adapted from Pauline Rivoire)

[4]:
```
# importing libraries as an alias so that we know which function belongs to␣
 ↪which library
import xarray as xr
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import pandas as pd
import xclim as xc
from IPython.display import Image
from IPython import display

# Geographic plotting
import cartopy.crs as ccrs
import cartopy.feature as cfeature;
```

[5]:
```
#path_to_data='/climriskdata/'
path_to_data='/lhome/cra2022/climriskdata/'
```

A clear and adapted communication of your results is key in science. The less time the reader spends on understanding your plots, the faster s.he can focus on the message you want to deliver. Readability details, like font size, colormaps, annotations, can make a huge difference.

1.1 Colormaps

1.1.1 Categories of colormaps

There are 3 categories of colormaps: * Sequential colormaps: one continuous sequence of colors (e.g. viridis),

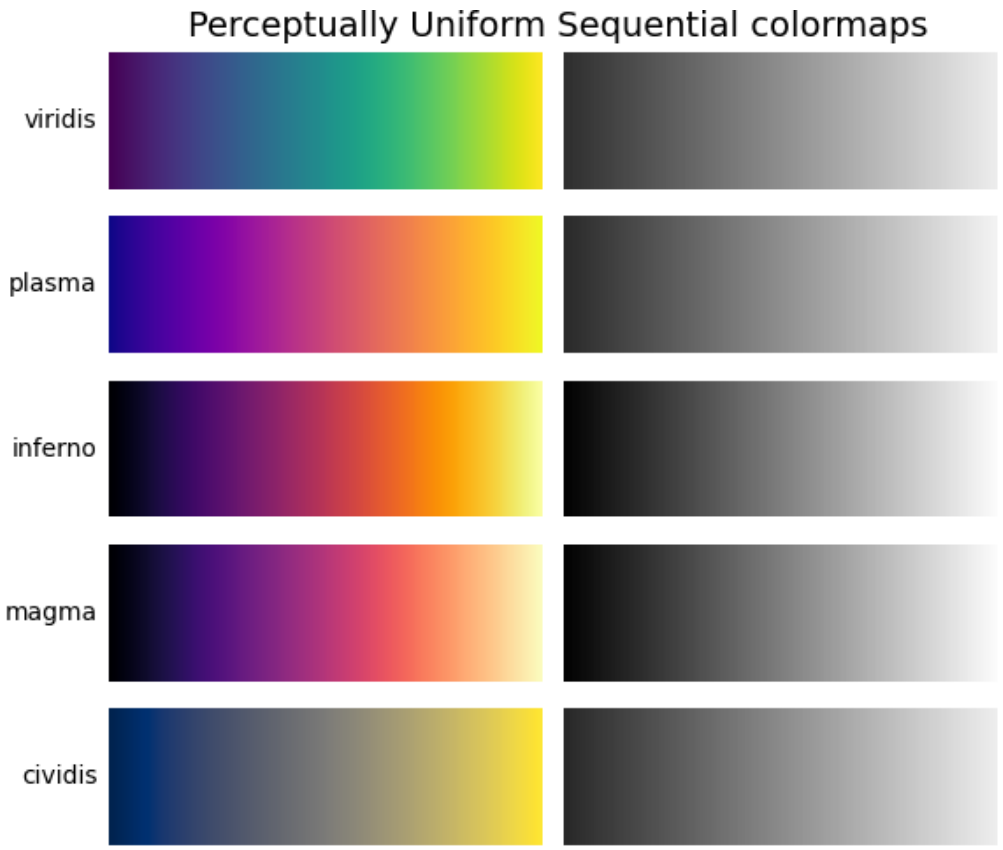
- Divergent colormaps: usually containing two distinct colors, to highlight positive and negative deviations from a mean (e.g. RdBu),
- Qualitative colormaps: mix colors with no particular sequence (e.g. rainbow or jet).

One has to choose wisely a colormap from a large variety. It has to be suitable for the type of data you want to plot. Human eye perception is also an important parameter to take into account.

Here are some colorbars provided by `matplotlib` (reference: https://matplotlib.org/stable/tutorials/colors/colormaps.html)

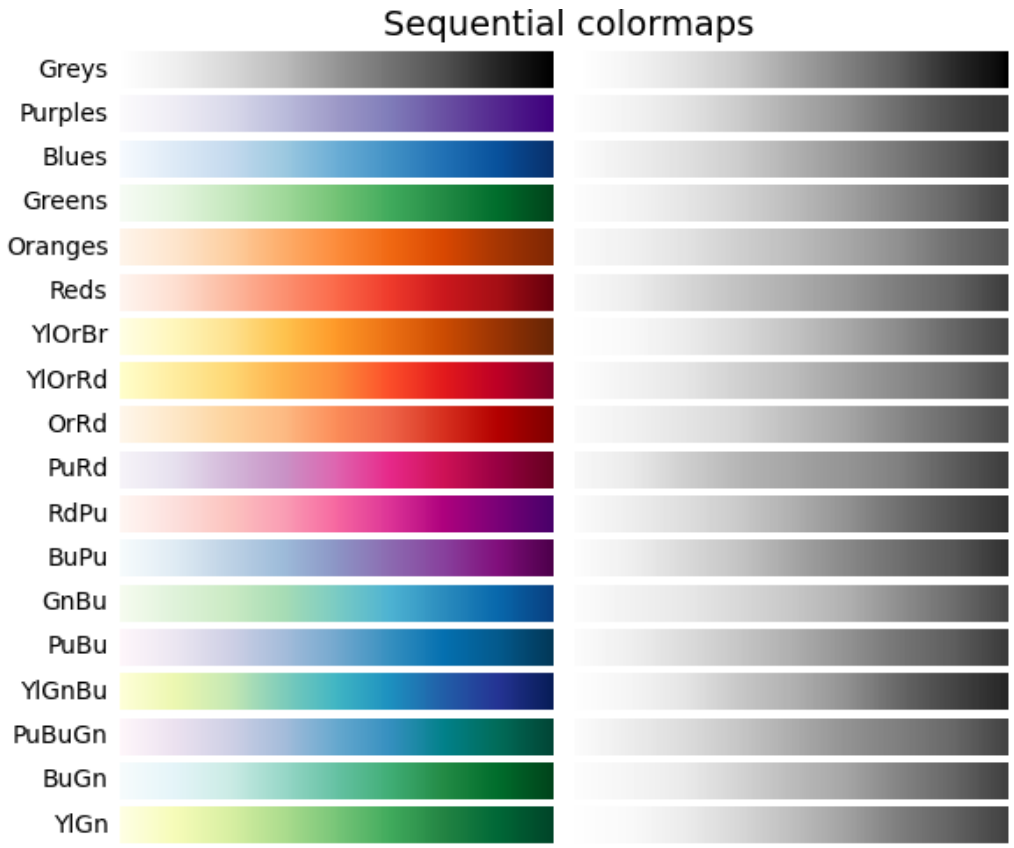
[3]:
```
display.Image("https://matplotlib.org/stable/_images/sphx_glr_colormaps_015.
↪png")
```

[3]:



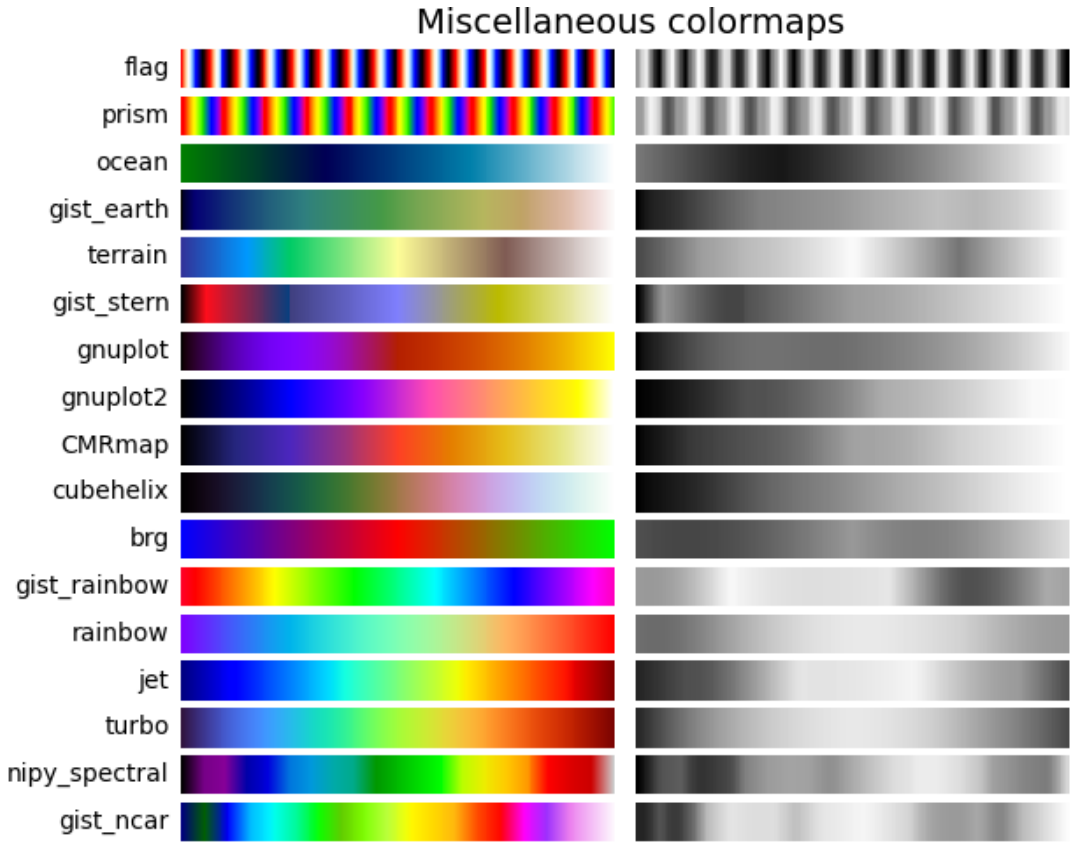
[4]:
```
display.Image("https://matplotlib.org/stable/_images/sphx_glr_colormaps_016.
↪png")
```

[4]:



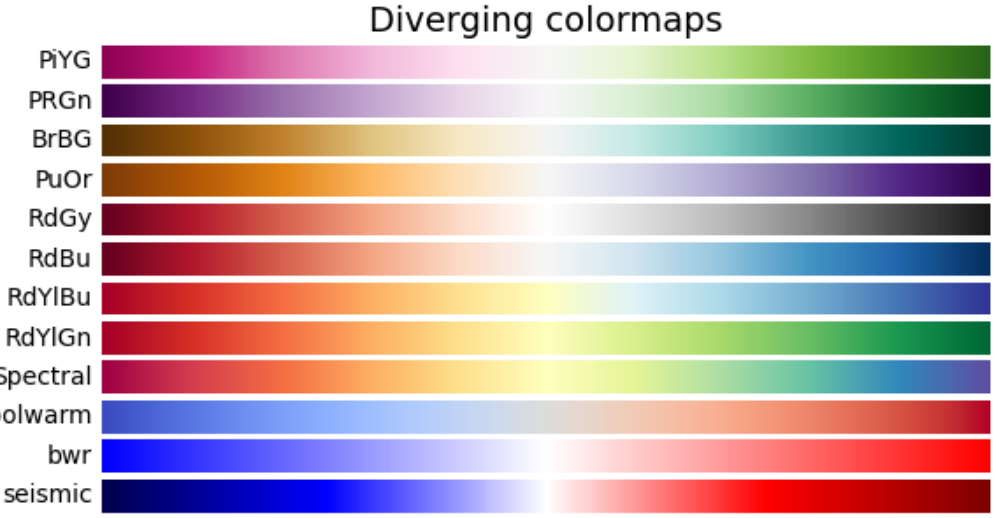
[5]:
```
display.Image("https://matplotlib.org/stable/_images/sphx_glr_colormaps_021.
↪png")
```

[5]:



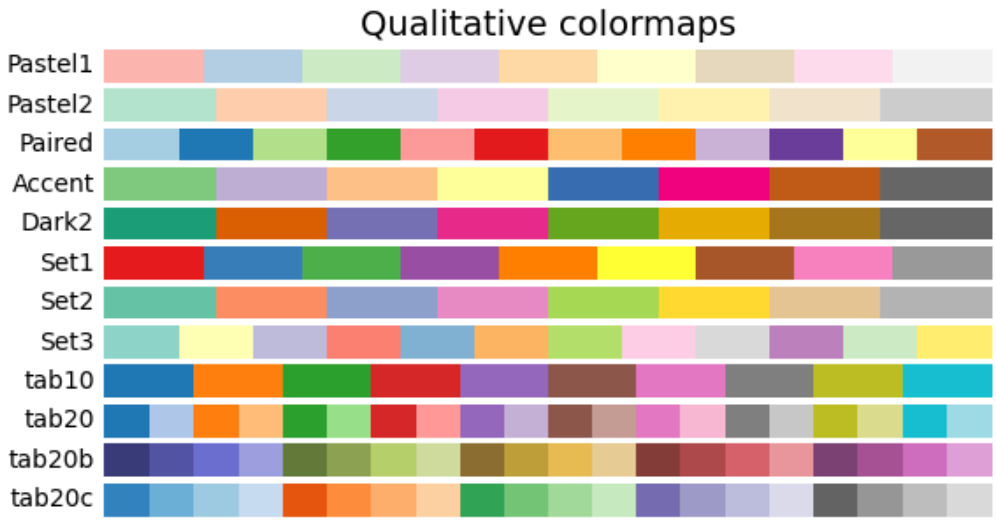
[6]:
```
display.Image("https://matplotlib.org/stable/_images/sphx_glr_colormaps_004.
↪png")
```

[6]:



[7]:
```
display.Image("https://matplotlib.org/stable/_images/sphx_glr_colormaps_006.
↪png")
```

[7]:



1.1.2 *#endtherainbow*

[6]:
```
#Don't pay too much attention on the cells of this section, just run them
class Tweet(object):
    def __init__(self, embed_str=None):
        self.embed_str = embed_str

    def _repr_html_(self):
        return self.embed_str
```

[7]:
```
Tweet("""
<blockquote class="twitter-tweet"><p lang="en" dir="ltr">Difference between␣
↪color palettes with monotonic and non-monotonic lightness functions. Data␣
↪are 72 hours of surface temperature. Non-monotonic palette is more dramatic,␣
↪but perceptually misrepresents temperature gradients.  <a href="https://
↪twitter.com/hashtag/EarthEngine?src=hash&amp;
↪ref_src=twsrc%5Etfw">#EarthEngine</a> lava lamp.<a href="https://t.co/
↪3kuHcuusgr">https://t.co/3kuHcuusgr</a> <a href="https://t.co/
↪rCzSzIsNyy">pic.twitter.com/rCzSzIsNyy</a></p>&mdash; Justin Braaten␣
↪(@jstnbraaten) <a href="https://twitter.com/jstnbraaten/status/
↪1210631357317074944?ref_src=twsrc%5Etfw">December 27, 2019</a></blockquote>␣
↪<script async src="https://platform.twitter.com/widgets.js"␣
↪charset="utf-8"></script>
```

```
""")
```

[7]:
```
<__main__.Tweet at 0x7fe4b0ee6a90>
```

[10]:
```
Tweet("""
<blockquote class="twitter-tweet"><p lang="en" dir="ltr">Transitions between␣
 ↪some colors are accompanied by a major change in lightness trajectory (e.g.,␣
 ↪yellow to red) which can make the data represented by these colors appear to␣
 ↪have a steeper gradient than the same data delta represented by a different␣
 ↪color transition.</p>&mdash; Justin Braaten (@jstnbraaten) <a href="https://
 ↪twitter.com/jstnbraaten/status/1210683184561577984?
 ↪ref_src=twsrc%5Etfw">December 27, 2019</a></blockquote> <script async␣
 ↪src="https://platform.twitter.com/widgets.js" charset="utf-8"></script>
""")
```

[10]:
```
<__main__.Tweet at 0x7f07ed29ce80>
```

- The uneven brightness can potentially emphasize unimportant parts of the dataset.

There is even a hashtag #endtherainbow ! https://twitter.com/JoannaMerson/status/1210765303107342337. See also the article "Rainbow Color Map (Still) Considered Harmful" https://ieeexplore.ieee.org/document/4118486

- Generally speaking, *qualitative colormaps* shouldn't be used to represent *quantitative data*.

1.2 Using the existing colormaps

1.2.1 Example with a sequential colormap

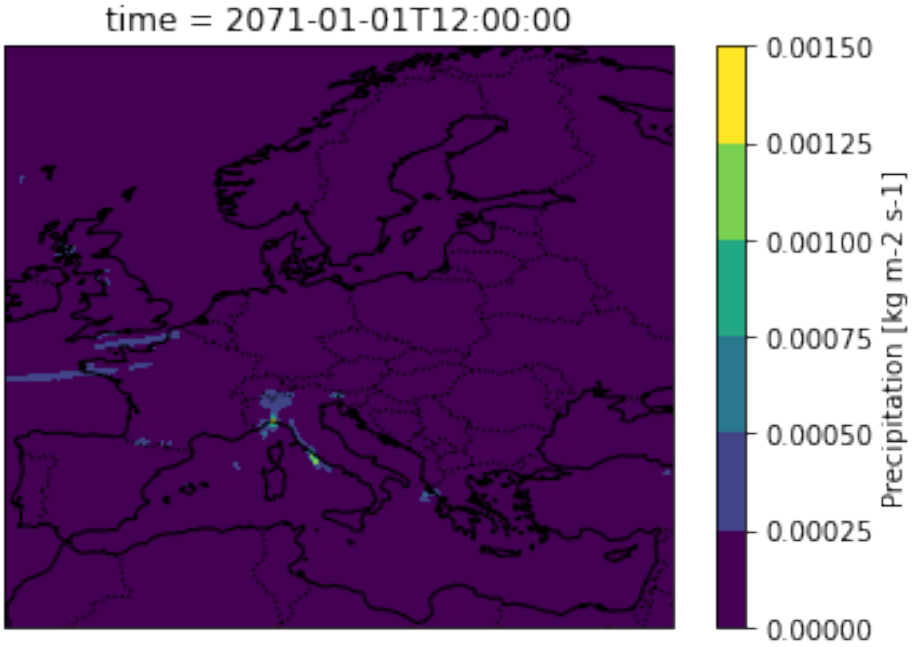
[8]:
```
date = '20710101-20711231'
myPrecipfile = xr.open_dataset(path_to_data+'EUR-11S/
 ↪ICHEC-EC-EARTH_CLMcom-CCLM4-8-17_v1/rcp85/pr/
 ↪pr_EUR-11_ICHEC-EC-EARTH_rcp85_r12i1p1_CLMcom-CCLM4-8-17_v1_day_' +␣
 ↪date+'_LL.nc')
myPrecipfile
```

[8]:
```
<xarray.Dataset>
Dimensions:    (time: 365, bnds: 2, lon: 471, lat: 409)
Coordinates:
  * time       (time) datetime64[ns] 2071-01-01T12:00:00 … 2071-12-31T12:00:00
  * lon        (lon) float64 -10.0 -9.9 -9.8 -9.7 -9.6 … 36.7 36.8 36.9 37.0
  * lat        (lat) float64 30.0 30.1 30.2 30.3 30.4 … 70.5 70.6 70.7 70.8
Dimensions without coordinates: bnds
Data variables:
    time_bnds  (time, bnds) datetime64[ns] …
    pr         (time, lat, lon) float32 …
Attributes: (12/31)
    CDI:                          Climate Data Interface version ?? (http:/…
```

```
history:                    Thu Mar 12 16:58:17 2020: cdo selyear,207…
source:                     CLMcom-CCLM4-8-17
institution:                Climate Limited-area Modelling Community …
Conventions:                CF-1.4
institute_id:               CLMcom
…                           …
project_id:                 CORDEX
table_id:                   Table day (Sept 2013) 0cf1782745489246c9f…
modeling_realm:             atmos
realization:                12
cmor_version:               2.9.1
CDO:                        Climate Data Operators version 1.9.3 (htt…
```

Without specifying any colorbar, we get the default one (viridis, for positive data)

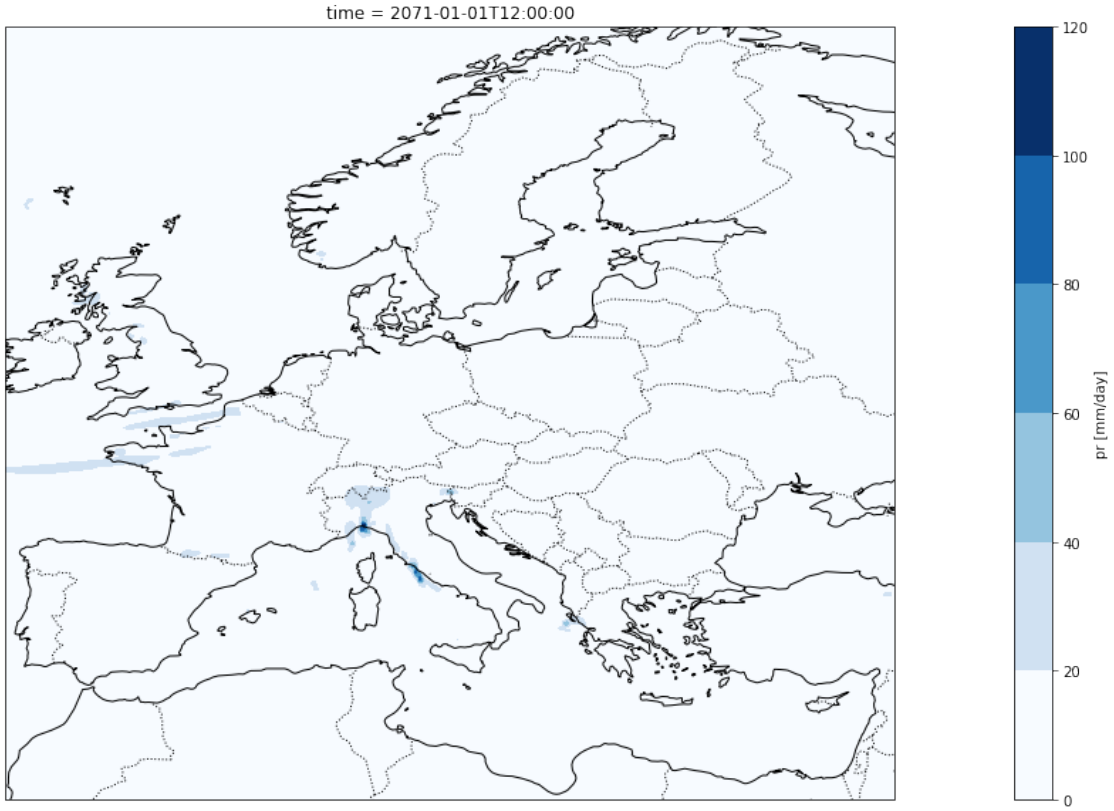
[9]:
```
#fig = plt.figure(figsize=(30,10))
ax = plt.axes(projection=ccrs.PlateCarree())


#Include a ready-to-use colormap with cmap=<colormap_name>
#!\ to use a color vector you made yourself (e.g. the MeteoSwiss one), use␣
 ↪"colors=" instead of "cmap="
a = myPrecipfile.pr.isel(time=0).plot.contourf(ax=ax, transform=ccrs.
 ↪PlateCarree())


ax.add_feature(cfeature.COASTLINE, linestyle='-')
ax.add_feature(cfeature.BORDERS, linestyle=':');
```



This sequential colormap is not very suitable in our context of daily precipitation. Let's change it, and let's also change the precipitation units to mm/day.

[10]:
```
#from notebook Python_climate_indices.ipynb
pr_mm = myPrecipfile.pr * 86400
pr_mm.attrs['units'] = 'mm/day'
data_precip = pr_mm.isel(time=0)
```

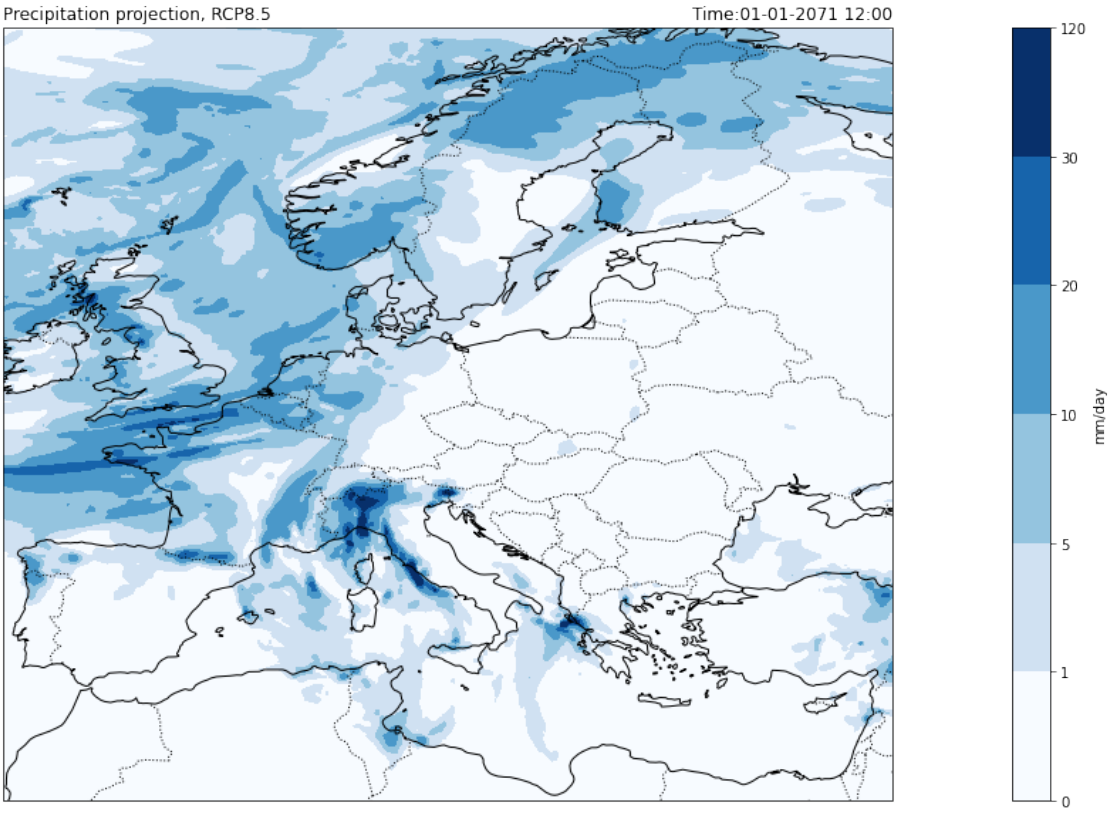
[11]:
```
fig = plt.figure(figsize=(30,10))
ax = plt.axes(projection=ccrs.PlateCarree())


#Include a ready-to-use colormap with cmap=<colormap_name>
a = data_precip.plot.contourf(ax=ax, transform=ccrs.PlateCarree(), cmap="Blues")


ax.add_feature(cfeature.COASTLINE, linestyle='-')
ax.add_feature(cfeature.BORDERS, linestyle=':');
```



- Let's now change the levels to emphasize extreme precipitation

[12]:
```
# Nice time values for the plots (from notebook Plotting.ipynb)
dt = pd.to_datetime(data_precip.time.data)
nice_time = dt.strftime('%d-%m-%Y %H:%M')
```

[13]:
```
fig = plt.figure(figsize=(30,10))
ax = plt.axes(projection=ccrs.PlateCarree())

my_levels = [0,1,5,10,20,30,120]

a = data_precip.plot.contourf(ax=ax, transform=ccrs.PlateCarree(), levels =␣
 ↪my_levels, cmap="Blues",
                               add_colorbar=False) #remove the automatic colorbar


ax.add_feature(cfeature.COASTLINE, linestyle='-')
ax.add_feature(cfeature.BORDERS, linestyle=':')

#Extract the colorbar from the plot a with axes ax.
#"fraction=" = colorbar size, label=colorbar label
cbar = fig.colorbar(a, ax=ax, fraction = 0.1, label=r'mm/day')

# setting the title
ax.set_title('') # to remove xarray auto-title
ax.set_title('Time:{}'.format(nice_time), loc='right');
ax.set_title('Precipitation projection, RCP8.5', loc='left');
```



1.2.2 Example with a diverging colormap

If you provide data with positive and negative values, the default colorbar will be a diverging colorbar. 0 will automatically be set as the center of the colorbar, and the colors above or below are assigned to values in a symetric way. This only works if the user doesn't define the `levels`. Let's illustrate this with temperature data:

[14]:
```
ds_tasmax=xr.open_dataset(path_to_data + "EUR-11S/
  ↪ICHEC-EC-EARTH_CLMcom-CCLM4-8-17_v1/rcp85/tasmax/
  ↪tasmax_EUR-11_ICHEC-EC-EARTH_rcp85_r12i1p1_CLMcom-CCLM4-8-17_v1_day_"+date+"_LL.
  ↪nc")
ds_tasmax
```

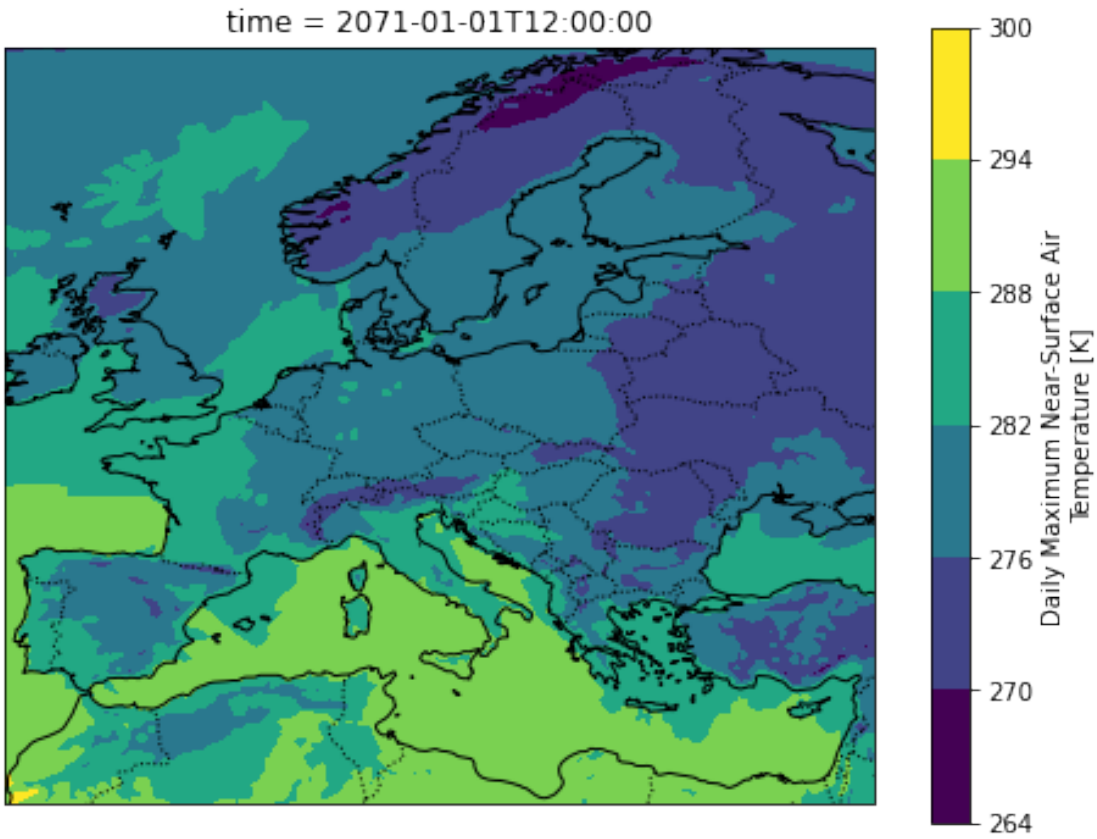
[14]:
```
<xarray.Dataset>
Dimensions:    (time: 365, bnds: 2, lon: 471, lat: 409)
Coordinates:
  * time       (time) datetime64[ns] 2071-01-01T12:00:00 … 2071-12-31T12:00:00
  * lon        (lon) float64 -10.0 -9.9 -9.8 -9.7 -9.6 … 36.7 36.8 36.9 37.0
  * lat        (lat) float64 30.0 30.1 30.2 30.3 30.4 … 70.5 70.6 70.7 70.8
Dimensions without coordinates: bnds
Data variables:
```

```
    time_bnds  (time, bnds) datetime64[ns] …
    tasmax     (time, lat, lon) float32 …
Attributes: (12/31)
    CDI:                           Climate Data Interface version ?? (http:/…
    history:                       Thu Mar 12 16:56:00 2020: cdo selyear,207…
    source:                        CLMcom-CCLM4-8-17
    institution:                   Climate Limited-area Modelling Community …
    Conventions:                   CF-1.4
    institute_id:                  CLMcom
    …                              …
    project_id:                    CORDEX
    table_id:                      Table day (Sept 2013) 0cf1782745489246c9f…
    modeling_realm:                atmos
    realization:                   12
    cmor_version:                  2.9.1
    CDO:                           Climate Data Operators version 1.9.3 (htt…
```

A quick look at the raw data:

[15]:
```
fig = plt.figure(figsize=(8,6))
ax = plt.axes(projection=ccrs.PlateCarree())

ds_tasmax.tasmax.isel(time=0).plot.contourf(ax=ax, transform=ccrs.
 ↪PlateCarree());

ax.add_feature(cfeature.COASTLINE, linestyle='-')
ax.add_feature(cfeature.BORDERS, linestyle=':')
```

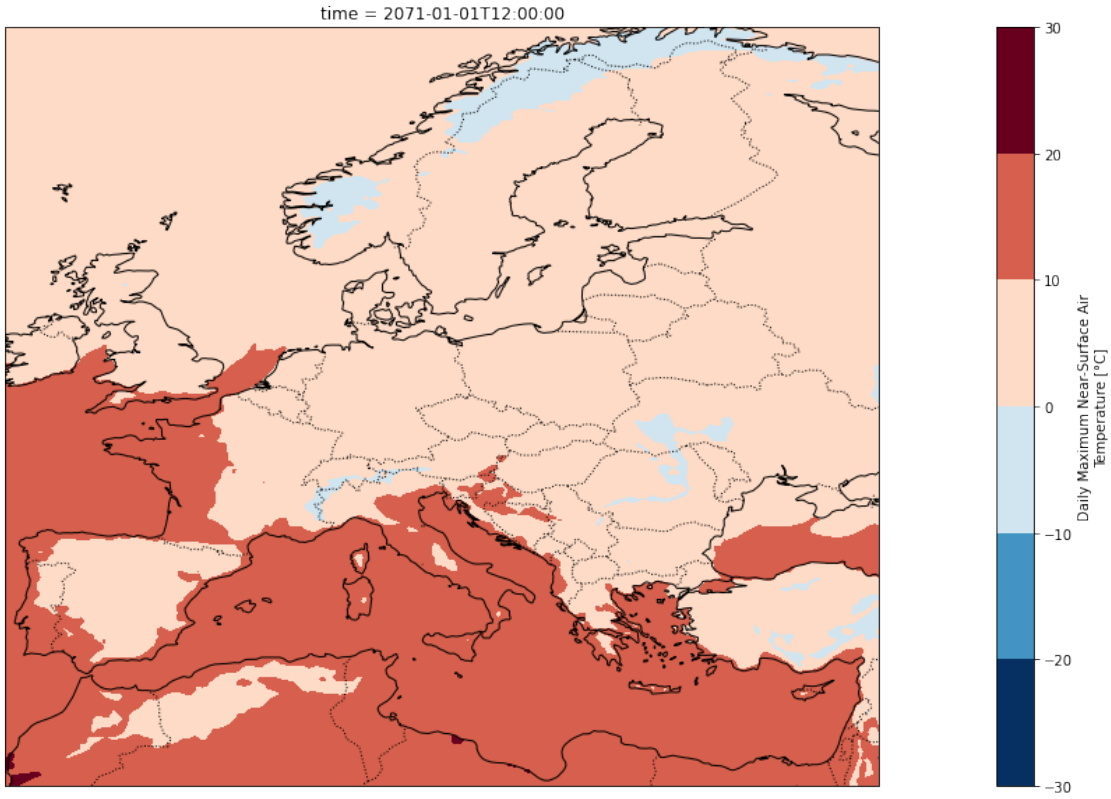
[15]:
```
<cartopy.mpl.feature_artist.FeatureArtist at 0x7fe47aaf7610>
```



Let's convert the unit to °C, to get positive and negative values. This will change the automatic colorbar

[19]:
```
fig = plt.figure(figsize=(30,10))
ax = plt.axes(projection=ccrs.PlateCarree())

tasmax_degC_xclim = xc.units.convert_units_to(ds_tasmax.tasmax, 'degC') #from␣
↪notebook Python_climate_indices.ipynb

tasmax_data = tasmax_degC_xclim.isel(time=0)

#tasmax_data.plot.contourf(ax=ax, transform=ccrs.PlateCarree());
tasmax_data.plot.contourf(ax=ax, transform=ccrs.PlateCarree(), cmap="RdBu_r");

ax.add_feature(cfeature.COASTLINE, linestyle='-')
ax.add_feature(cfeature.BORDERS, linestyle=':')
```

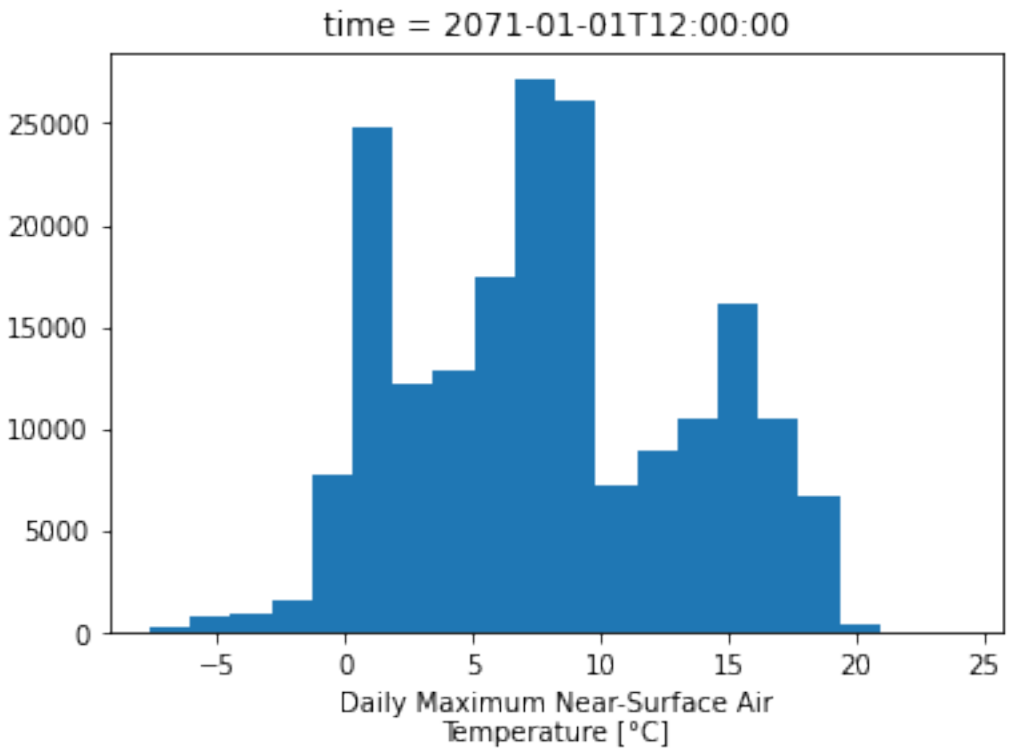
[19]: <cartopy.mpl.feature_artist.FeatureArtist at 0x7fe47a8c7460>



Remark: if you want to reverse the ordre of the colors in a colormap, add `"_r"` at the end of the name. This works to reverse any pre-defined colormap.

If we plot a histogram of our data, we see that almost all temperature values are positive, and that most values are between 0° and 10°C. We might want to change the levels to have more informative representation of the data. But in this case we would have to make a customised diverging colormap.

[20]:
```
tasmax_data.plot.hist(bins=20);
```



1.3 Create and use your own colormaps

Using `seaborn`colormap for custom diverging colormap… And more!

1.3.1 Precipitation

- Extract RGB codes from a ready-to-use colormap (here from a simple `seaborn` colormap)

[23]:
```
color_ex = sns.color_palette("Blues", n_colors=5)
color_ex
```

[23]:
```
[(0.8406920415224913, 0.9016378316032295, 0.9586620530565167),
 (0.6718954248366014, 0.8143790849673203, 0.9006535947712418),
 (0.41708573625528644, 0.6806305267204922, 0.8382314494425221),
 (0.21568627450980393, 0.5294117647058824, 0.7542483660130719),
 (0.06251441753171857, 0.35750865051903113, 0.6429065743944637)]
```

- `append()` command to manually add an element at the end of a list.

Here it can be used to complete the sequential colormap with a color for values out of certain boundaries

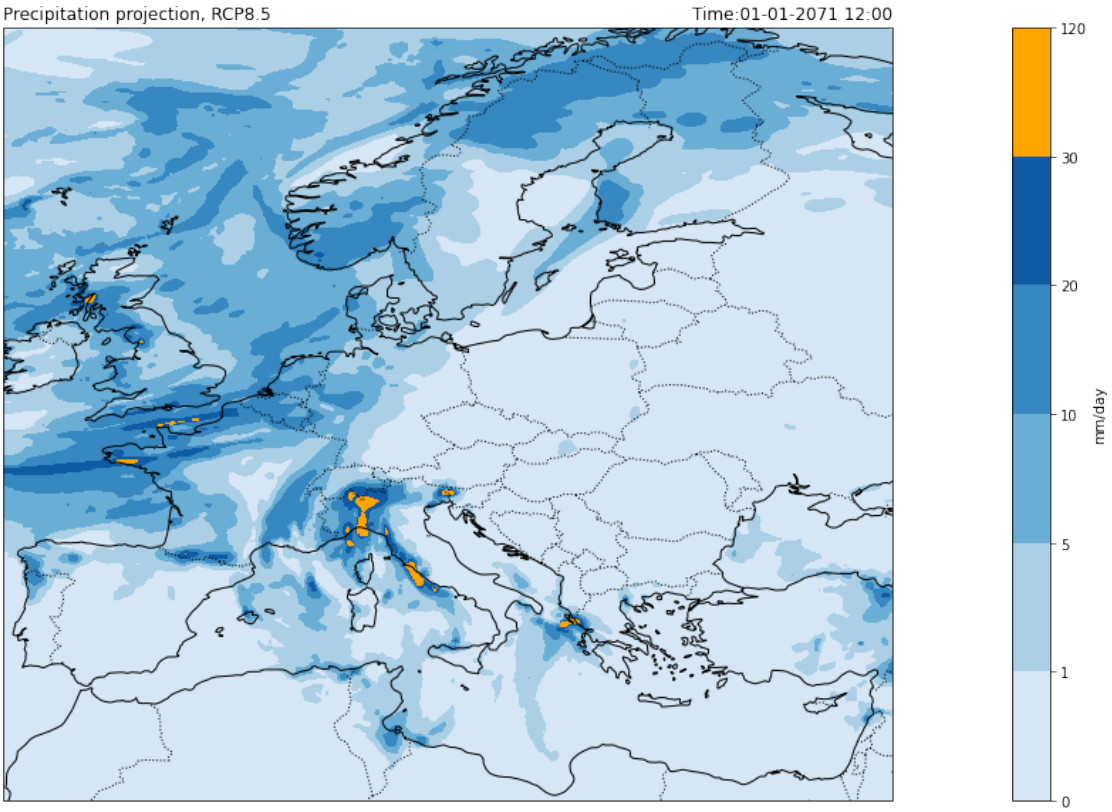
[24]:
```
color_ex.append("Orange")
```

[25]:
```
color_ex
```

[25]:
```
[(0.8406920415224913, 0.9016378316032295, 0.9586620530565167),
 (0.6718954248366014, 0.8143790849673203, 0.9006535947712418),
 (0.41708573625528644, 0.6806305267204922, 0.8382314494425221),
 (0.21568627450980393, 0.5294117647058824, 0.7542483660130719),
 (0.06251441753171857, 0.35750865051903113, 0.6429065743944637),
 'Orange']
```

Let's use this colormap to highlight extreme daily precipitation. In the function `contourf`, we used the argument `cmap` to indicate the wanted "ready-to-use" color palette. For a customized colormap, we have to use the argument `colors`.

[26]:
```
fig = plt.figure(figsize=(30,10))
ax = plt.axes(projection=ccrs.PlateCarree())

my_levels = [0,1,5,10,20,30,120]

a = data_precip.plot.contourf(ax=ax, transform=ccrs.PlateCarree(), levels =␣
 ↪my_levels,
                              colors=color_ex, add_colorbar=False)

ax.add_feature(cfeature.COASTLINE, linestyle='-')
ax.add_feature(cfeature.BORDERS, linestyle=':')

cbar = fig.colorbar(a, ax=ax, fraction = 0.1, label=r'mm/day')


ax.set_title('')
ax.set_title('Time:{}'.format(nice_time), loc='right');
ax.set_title('Precipitation projection, RCP8.5', loc='left');
```



1.3.2 Temperature

We now go back to our temperature example. We first select a red sequential colormap and a blue sequential colormap (note the use of `-r` to reverse the order):

[27]:
```
color_t_pos = sns.color_palette("Reds", n_colors=7)
color_t_pos
```

[27]:
```
[(0.9960476739715494, 0.8778623606305267, 0.8227758554402153),
 (0.9882352941176471, 0.732072279892349, 0.6299269511726259),
 (0.9881891580161477, 0.5707035755478662, 0.44521337946943484),
 (0.9835755478662053, 0.4127950788158401, 0.28835063437139563),
 (0.9344867358708189, 0.2286812764321415, 0.17139561707035755),
 (0.7925720876585928, 0.09328719723183392, 0.11298731257208766),
 (0.6403844675124952, 0.05720876585928489, 0.08149173394848133)]
```

[28]:
```
color_t_neg = sns.color_palette("Blues_r", n_colors=2)
color_t_neg
```

[28]:
```
[(0.21568627450980393, 0.5294117647058824, 0.7542483660130719),
 (0.6718954248366014, 0.8143790849673203, 0.9006535947712418)]
```

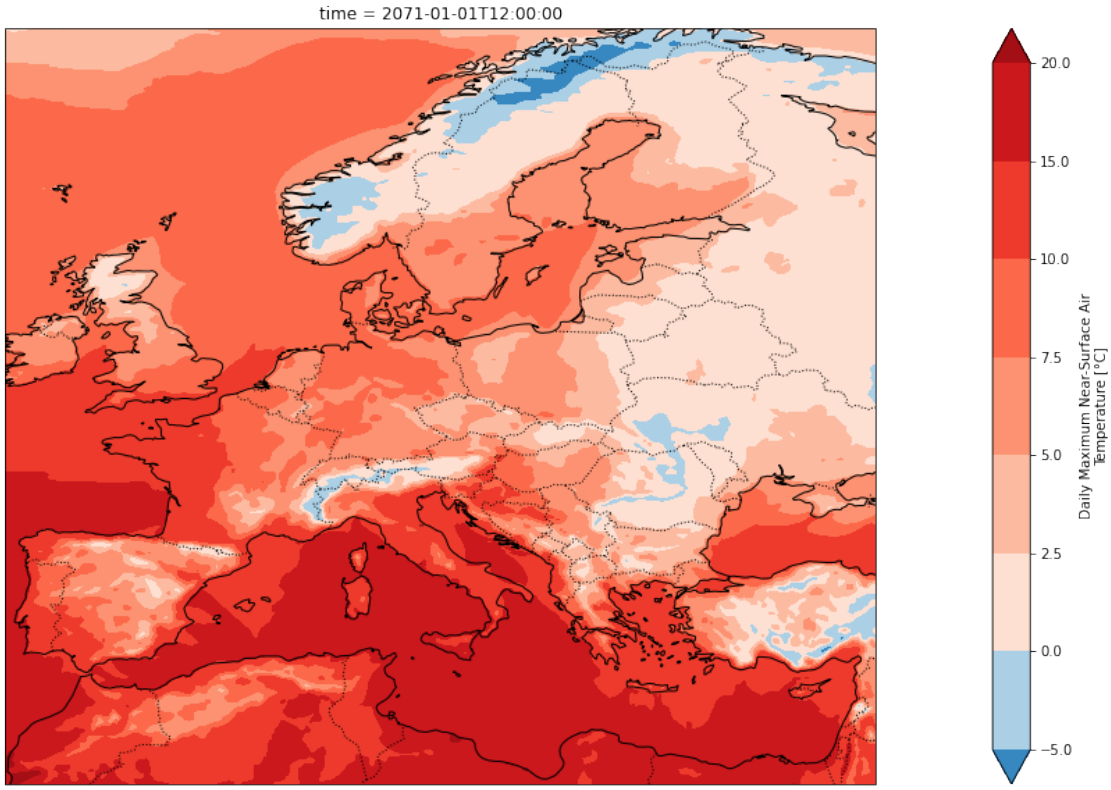
We then use the `extend()` method of a list to construct our colormap (`append()` only works with a single element)

[29]:
```
color_t_neg.extend(color_t_pos)
color_t_neg
```

[29]:
```
[(0.21568627450980393, 0.5294117647058824, 0.7542483660130719),
 (0.6718954248366014, 0.8143790849673203, 0.9006535947712418),
 (0.9960476739715494, 0.8778623606305267, 0.8227758554402153),
 (0.9882352941176471, 0.732072279892349, 0.6299269511726259),
 (0.9881891580161477, 0.5707035755478662, 0.44521337946943484),
 (0.9835755478662053, 0.4127950788158401, 0.28835063437139563),
 (0.9344867358708189, 0.2286812764321415, 0.17139561707035755),
 (0.7925720876585928, 0.09328719723183392, 0.11298731257208766),
 (0.6403844675124952, 0.05720876585928489, 0.08149173394848133)]
```

[30]:
```
fig = plt.figure(figsize=(30,10))
ax = plt.axes(projection=ccrs.PlateCarree())

c_levels = [-5,0,2.5,5,7.5,10,15,20]
tasmax_data.plot.contourf(ax=ax, transform=ccrs.PlateCarree(), levels=c_levels,␣
↪colors=color_t_neg);

ax.add_feature(cfeature.COASTLINE, linestyle='-')
ax.add_feature(cfeature.BORDERS, linestyle=':')
```

[30]:
```
<cartopy.mpl.feature_artist.FeatureArtist at 0x7fe47a62c970>
```



More information about seaborn color palettes/colormaps : https://seaborn.pydata.org/tutorial/color_palettes.html

1.3.3 A last example: MeteoSwiss precipitation return levels colormap

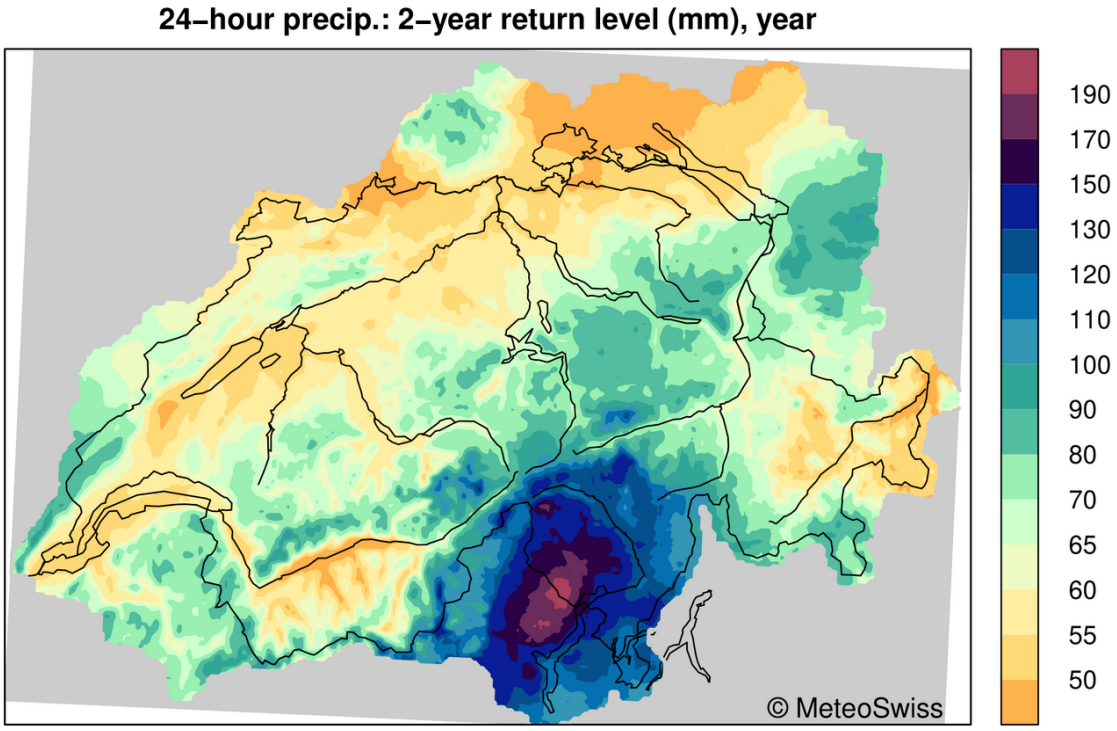
As a last example, let's see how to create manually a colormap: the one inspired from the precipitation return levels colorbar used by MeteoSwiss: https://www.meteoswiss.admin.ch/home/climate/swiss-climate-in-detail/extreme-value-analyses/maps-of-extreme-precipitation.html

It is composed of 3 different shades of color, to represent low, intermediate and high precipitation return levels:

- Yellow/brown has the connotation of dry conditions (color of the vegetation)
- Green to blue: the bluer the wetter
- Black/purple to indicate wet extremes

[31]:
```
Image(filename="/lhome/cra2022/cra2022_shared/Figures/
↪spatxs-prec_24-hour-sum_year_X002_retlev.png")
```

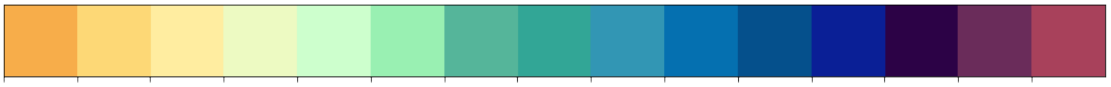
[31]:



[32]:
```
# RGB codes corresponding to MeteoSwiss colors:
my_MS_colors = [(247/255,173/255,74/255),(253/255,216/255,118/255),(255/255,237/
↪255,160/255),
                (237/255,250/255,194/255),(205/255,255/255,205/255),(153/255,240/
↪255,178/255),
                (85/255,181/255,154/255),(50/255,166/255,150/255),(50/255,150/
↪255,180/255),
                (5/255,112/255,176/255),(5/255,80/255,140/255),(10/255,31/255,150/
↪255),
                (44/255,2/255,70/255),(106/255,44/255,90/255),(168/255,65/255,91/
↪255)]
```

You have many options to find the RGB codes of the different colors composing an image. All graphic editors (Paint, GIMP, Photoshop, Inkscape, …) or online ressources can do the job (see e.g.: https://www.ginifab.com/feeds/pms/color_picker_from_image.php).

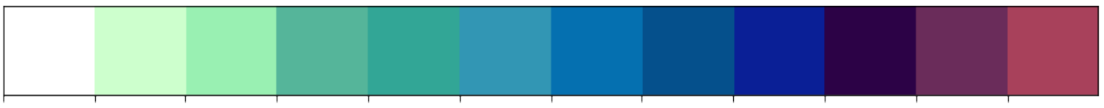
[33]:
```
sns.palplot(my_MS_colors)
```



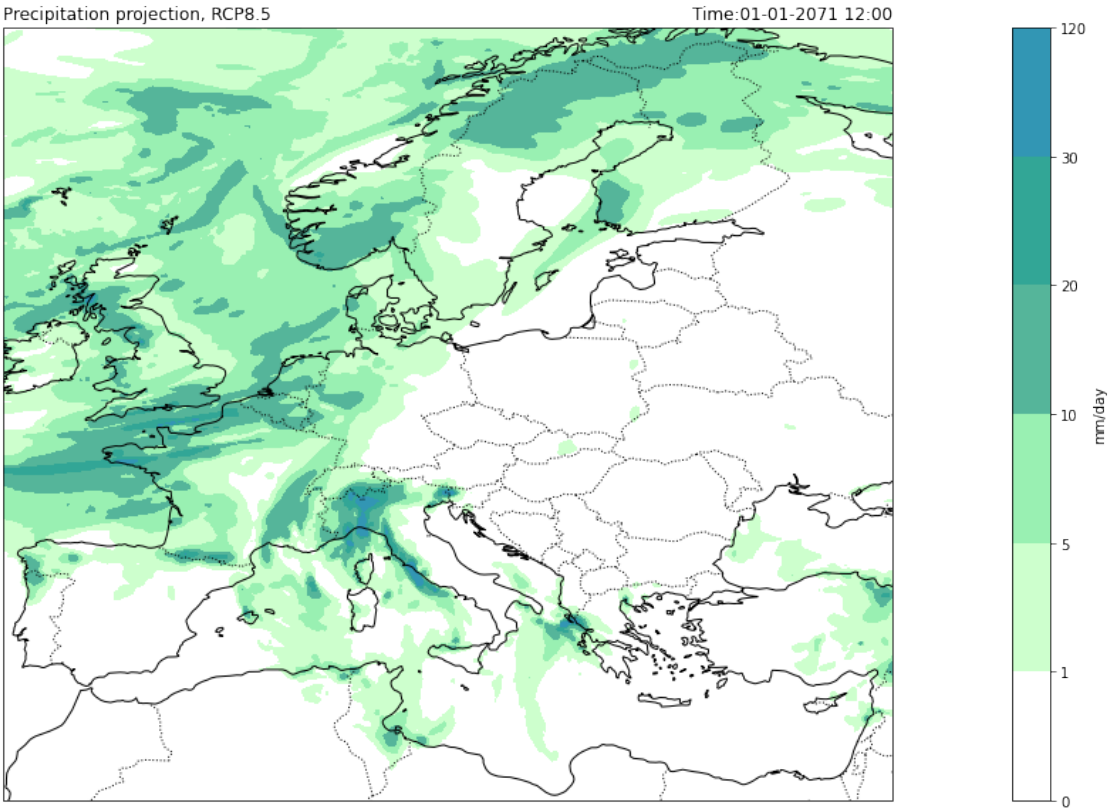
Let's use a part of this color for our precipitation plot and add white at the beginning to represent precipitation < 1 [mm]

[34]:
```
my_precip_color = [(1,1,1)] + my_MS_colors[4:] # (1,1,1) is the RGB code for␣
↪white
```

[35]:
```
sns.palplot(my_precip_color)
```



[36]:
```
fig = plt.figure(figsize=(30,10))
ax = plt.axes(projection=ccrs.PlateCarree())

my_levels = [0,1,5,10,20,30,120]

a = data_precip.plot.contourf(ax=ax, transform=ccrs.PlateCarree(), levels =␣
↪my_levels,
                              colors=my_precip_color, add_colorbar=False)

ax.add_feature(cfeature.COASTLINE, linestyle='-')
ax.add_feature(cfeature.BORDERS, linestyle=':')

cbar = fig.colorbar(a, ax=ax, fraction = 0.1, label=r'mm/day')


ax.set_title('')
ax.set_title('Time:{}'.format(nice_time), loc='right');
ax.set_title('Precipitation projection, RCP8.5', loc='left');
```

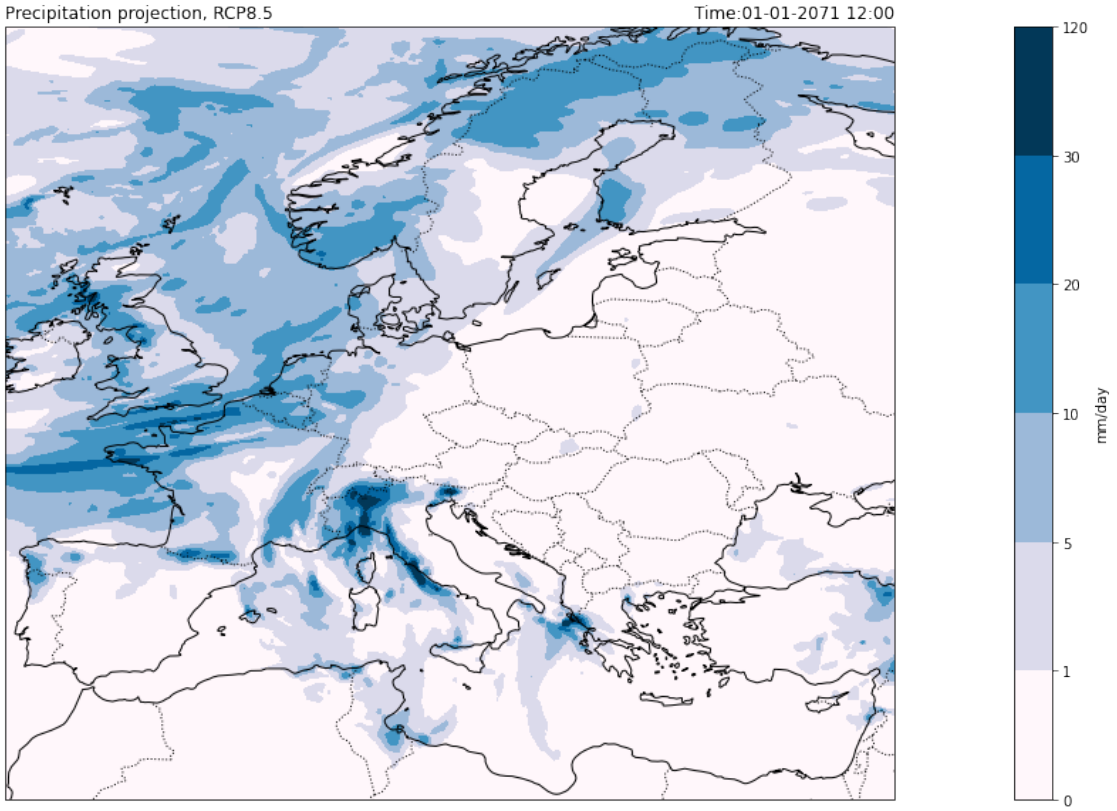


Remark: the last 6 colors were not used, because the number of levels we gave is smaller than the number of colors in our "home-made" colorbar.

With a "ready-to-use" sequential colormap, the 2 extreme colors are always used, and as many intermediate colors as necessary are selected.

[37]:
```
#exemple with "PuBu"

fig = plt.figure(figsize=(30,10))
ax = plt.axes(projection=ccrs.PlateCarree())

my_levels = [0,1,5,10,20,30,120]

a = data_precip.plot.contourf(ax=ax, transform=ccrs.PlateCarree(), levels =␣
 ↪my_levels,
                              cmap="PuBu", add_colorbar=False)

ax.add_feature(cfeature.COASTLINE, linestyle='-')
ax.add_feature(cfeature.BORDERS, linestyle=':')
cbar = fig.colorbar(a, ax=ax, fraction = 0.1, label=r'mm/day')
```

```
ax.set_title('')
ax.set_title('Time:{}'.format(nice_time), loc='right');
ax.set_title('Precipitation projection, RCP8.5', loc='left');
```



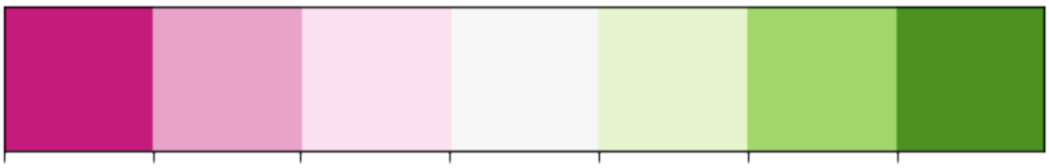
1.3.4 Great ressource for the choice of colors

http://colorbrewer2.org

- Experiment with different colormaps and get useful advices on their use based on several criteria (e.g.: color blindness)
- Get color codes values to use in your own plots

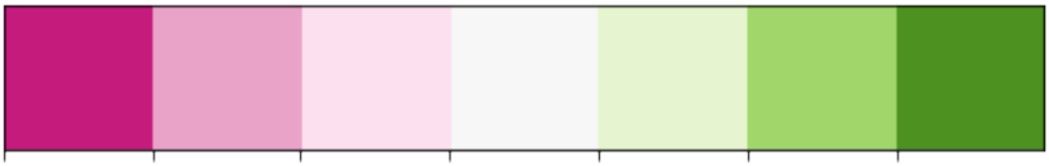
[38]:
```
# HEX
my_colors =␣
↪["#c51b7d","#e9a3c9","#fde0ef","#f7f7f7","#e6f5d0","#a1d76a","#4d9221"]
```

[39]:
```
sns.palplot(my_colors)
```



[40]:
```
#Alternative definition: RGB (red blue green, don't forget to divid by 255)
My_colors = [(197/255,27/255,125/255),(233/255,163/255,201/255),(253/255,224/
↪255,239/255),(247/255,247/255,247/255),
            (230/255,245/255,208/255),(161/255,215/255,106/255),(77/255,146/
↪255,33/255)]
```

[41]:
```
sns.palplot(My_colors)
```



1.4 Design tools

1.4.1 Adapt font size

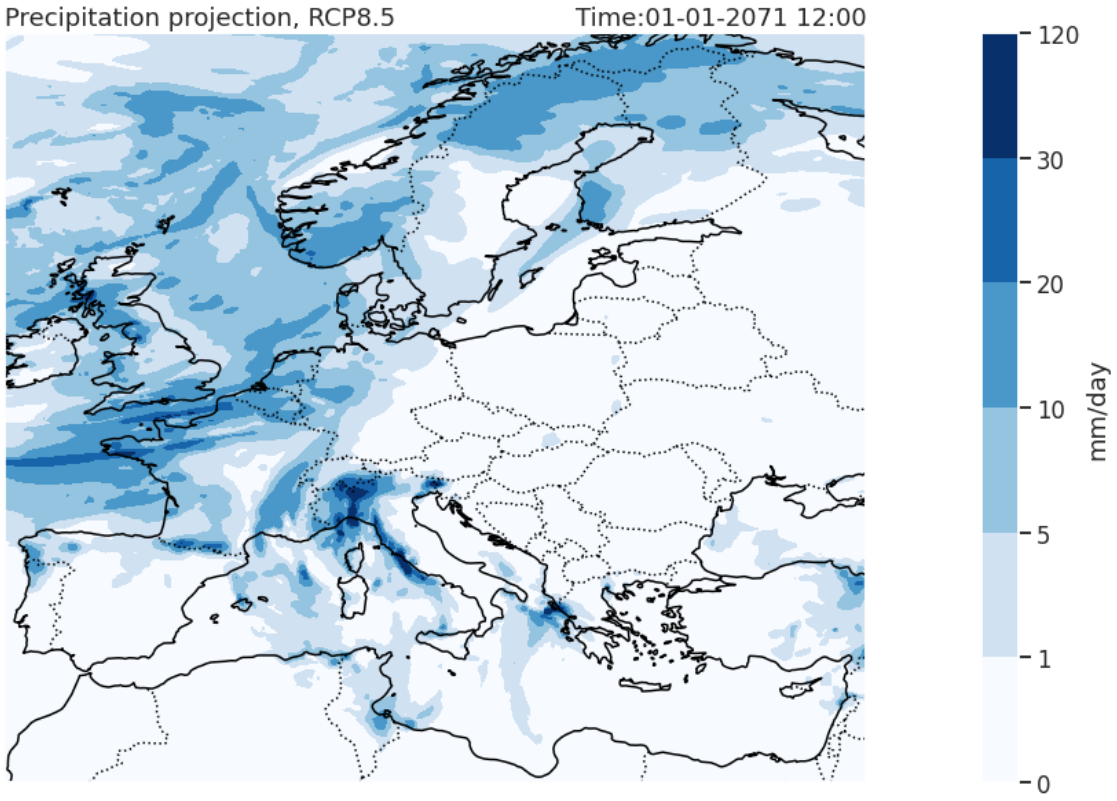
- The `seaborn` command `set` is a first solution to have a font size adapted to the context (`notebook`, `paper`, `talk` or `poster`):

[42]:
```
sns.set(context="talk")
```

[43]:
```
fig = plt.figure(figsize=(30,10))
ax = plt.axes(projection=ccrs.PlateCarree())

my_levels = [0,1,5,10,20,30,120]

a = data_precip.plot.contourf(ax=ax, transform=ccrs.PlateCarree(), levels =␣
↪my_levels, cmap="Blues", add_colorbar=False)

ax.add_feature(cfeature.COASTLINE, linestyle='-')
ax.add_feature(cfeature.BORDERS, linestyle=':')

cbar = fig.colorbar(a, ax=ax, fraction = 0.1, label=r'mm/day')
```

```
ax.set_title('')
ax.set_title('Time:{}'.format(nice_time), loc='right');
ax.set_title('Precipitation projection, RCP8.5', loc='left');
```

```
/tmp/ipykernel_705509/744308171.py:11: MatplotlibDeprecationWarning: Auto-
removal of grids by pcolor() and pcolormesh() is deprecated since 3.5 and will
be removed two minor releases later; please call grid(False) first.
  cbar = fig.colorbar(a, ax=ax, fraction = 0.1, label=r'mm/day')
```



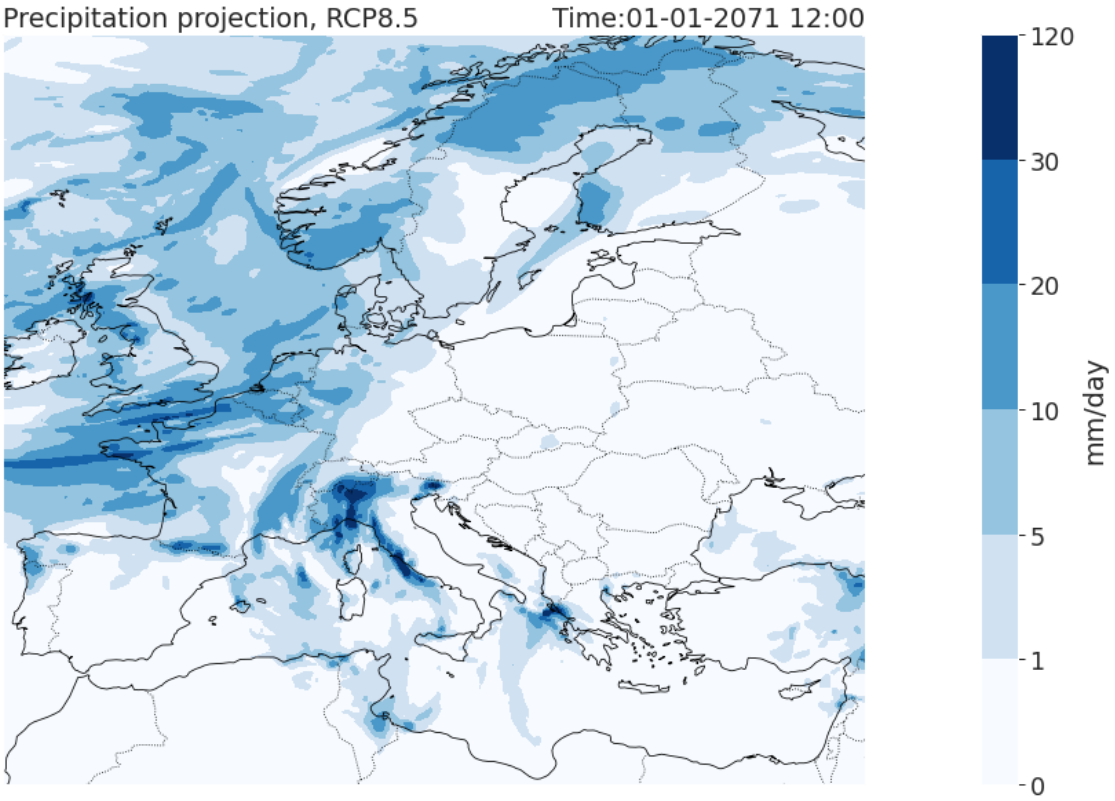
With the context `talk`, border lines are thicker, for a clearer image during a presentation. To just increase the font size and not the lines, you can play with the parameter `font_scale`

[44]:
```
sns.set(context="paper", font_scale=2)
```

[45]:
```
fig = plt.figure(figsize=(30,10))
ax = plt.axes(projection=ccrs.PlateCarree())

my_levels = [0,1,5,10,20,30,120]

a = data_precip.plot.contourf(ax=ax, transform=ccrs.PlateCarree(), levels =␣
 ↪my_levels, cmap="Blues", add_colorbar=False)
```

```
ax.add_feature(cfeature.COASTLINE, linestyle='-')
ax.add_feature(cfeature.BORDERS, linestyle=':')

cbar = fig.colorbar(a, ax=ax, fraction = 0.1, label=r'mm/day')


ax.set_title('')
ax.set_title('Time:{}'.format(nice_time), loc='right');
ax.set_title('Precipitation projection, RCP8.5', loc='left');
```

```
/tmp/ipykernel_705509/744308171.py:11: MatplotlibDeprecationWarning: Auto-
removal of grids by pcolor() and pcolormesh() is deprecated since 3.5 and will
be removed two minor releases later; please call grid(False) first.
  cbar = fig.colorbar(a, ax=ax, fraction = 0.1, label=r'mm/day')
```

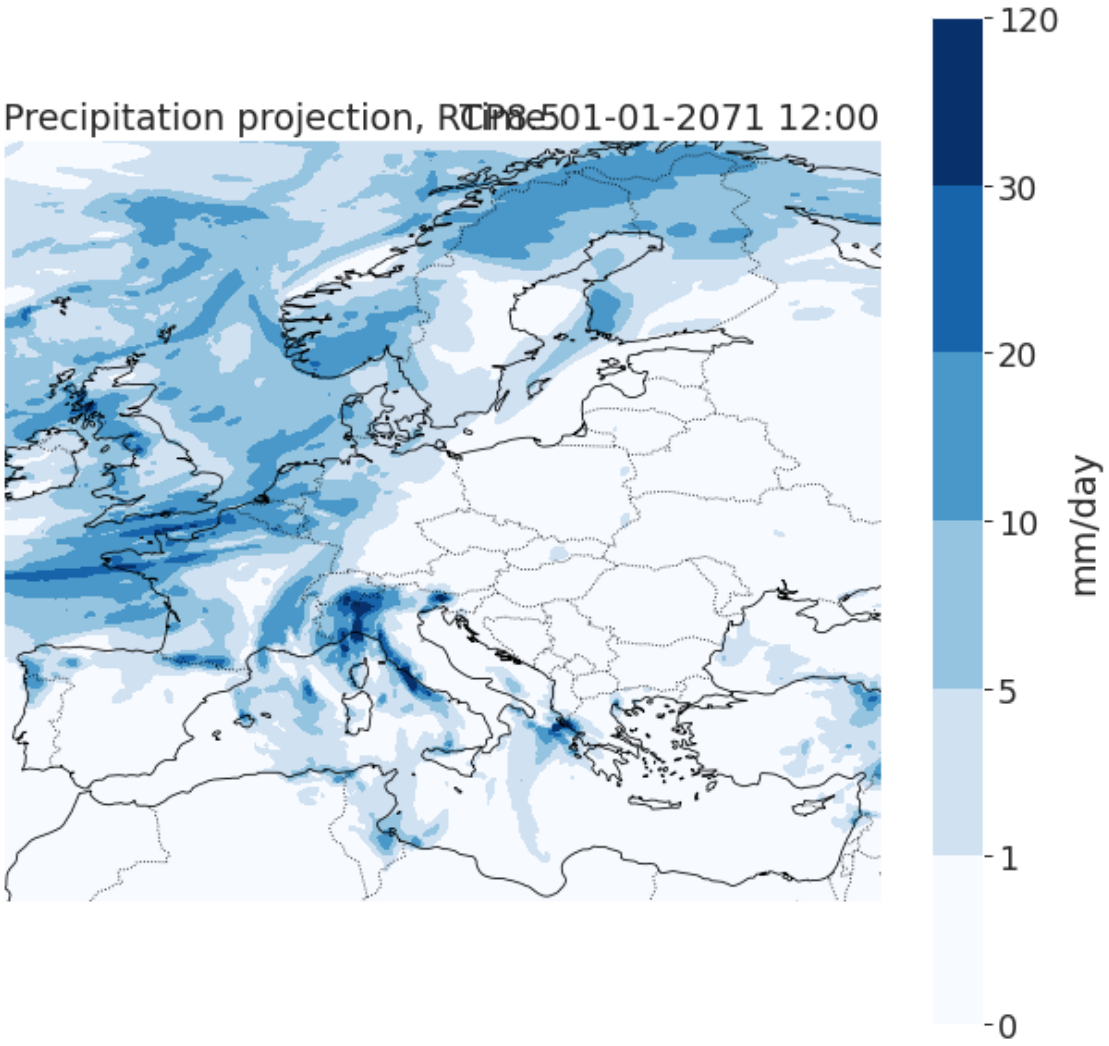


- The font size of each label/title/text can be further adapted individually (see e.g.: https://matplotlib.org/3.5.0/tutorials/text/text_intro.html)
- But beware that changing the figsize and DPI of the figures will most likely require to adapt the sizes of the elements inside the figure (fonts, but also line, marker, colorbar)
- For a discussion on the topic: https://stackoverflow.com/questions/47633546/relationship-between-dpi-and-figure-size
- NB: The DPI (dots per inches) is the image resolution. It determines how many pixels the

figure comprises. The default dpi in matplotlib is 100. A good DPI for printing/saving figures is 300.

[46]:
```
#Reduced figsize
fig = plt.figure(figsize=(10,10))
ax = plt.axes(projection=ccrs.PlateCarree())

my_levels = [0,1,5,10,20,30,120]

a = data_precip.plot.contourf(ax=ax, transform=ccrs.PlateCarree(), levels =␣
 ↪my_levels, cmap="Blues", add_colorbar=False)

ax.add_feature(cfeature.COASTLINE, linestyle='-')
ax.add_feature(cfeature.BORDERS, linestyle=':')

cbar = fig.colorbar(a, ax=ax, fraction = 0.1, label=r'mm/day')


ax.set_title('')
ax.set_title('Time:{}'.format(nice_time), loc='right');
ax.set_title('Precipitation projection, RCP8.5', loc='left');
```

```
/tmp/ipykernel_705509/384421102.py:12: MatplotlibDeprecationWarning: Auto-
removal of grids by pcolor() and pcolormesh() is deprecated since 3.5 and will
be removed two minor releases later; please call grid(False) first.
  cbar = fig.colorbar(a, ax=ax, fraction = 0.1, label=r'mm/day')
```



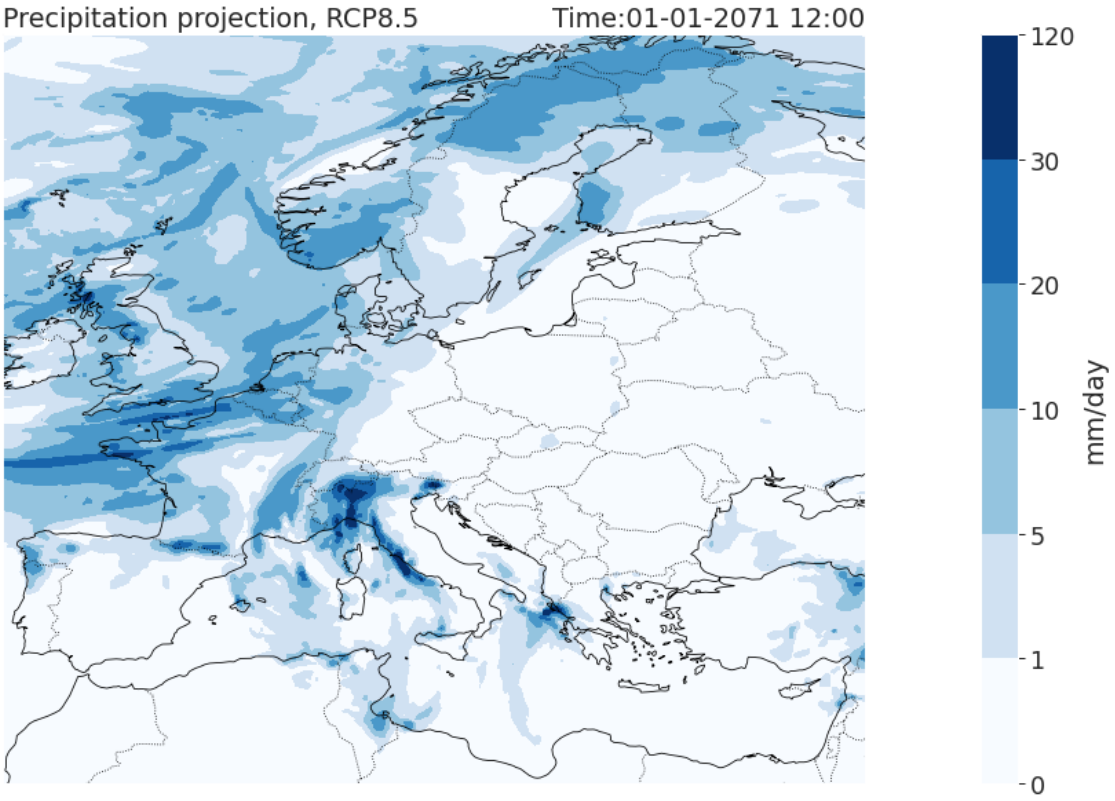
1.4.2 Remove white space around your plot

- Use `bbox_inches="tight"` and `pad_inches=0` in `savefig()` command (change path to your home!)
- Don't forget to change the paths in the savefig command!

[49]:
```
fig = plt.figure(figsize=(30,10))
ax = plt.axes(projection=ccrs.PlateCarree())

my_levels = [0,1,5,10,20,30,120]

a = data_precip.plot.contourf(ax=ax, transform=ccrs.PlateCarree(), levels =␣
 ↪my_levels, cmap="Blues", add_colorbar=False)

ax.add_feature(cfeature.COASTLINE, linestyle='-')
```

```
ax.add_feature(cfeature.BORDERS, linestyle=':')

cbar = fig.colorbar(a, ax=ax, fraction = 0.1, label=r'mm/day')


ax.set_title('')
ax.set_title('Time:{}'.format(nice_time), loc='right');
ax.set_title('Precipitation projection, RCP8.5', loc='left');


# to remove white space around the plot: bbox_inches="tight",pad_inches=0
# Adapt the path accordingly!

#plt.savefig("/lhome/cra2022/cra2022_shared/test_precip_tight"+ nice_time+".
 ↪png",  bbox_inches="tight",pad_inches=0);
#plt.savefig("/lhome/cra2022/cra2022_shared/test_precip_"+ nice_time+".png");
```

```
/tmp/ipykernel_2901023/1088635975.py:11: MatplotlibDeprecationWarning: Auto-
removal of grids by pcolor() and pcolormesh() is deprecated since 3.5 and will
be removed two minor releases later; please call grid(False) first.
  cbar = fig.colorbar(a, ax=ax, fraction = 0.1, label=r'mm/day')
```



1.4.3 Restrict plot to areas of interest

- `xr.where()`: set to *NaN* data that doesn't satisfy a given condition (NaN=not a number).
- E.g.: set precipitation below 1mm to *NaN*
- See also Lecture_Numpy_Xarray.ipynb and Python_climate_indices.ipynb

[47]:
```
data_precip.where(data_precip >1)
```

[47]:
```
<xarray.DataArray 'pr' (lat: 409, lon: 471)>
array([[       nan,        nan,        nan, …,        nan,        nan,
               nan],
       [       nan,        nan,        nan, …,        nan,        nan,
               nan],
       [       nan,        nan,        nan, …,        nan,        nan,
               nan],
       …,
       [       nan,        nan,        nan, …, 1.19392882, 1.12383059,
        1.17262781],
       [       nan,        nan,        nan, …,        nan,        nan,
               nan],
       [       nan,        nan,        nan, …,        nan,        nan,
               nan]])
Coordinates:
    time     datetime64[ns] 2071-01-01T12:00:00
  * lon      (lon) float64 -10.0 -9.9 -9.8 -9.7 -9.6 … 36.7 36.8 36.9 37.0
  * lat      (lat) float64 30.0 30.1 30.2 30.3 30.4 … 70.4 70.5 70.6 70.7 70.8
Attributes:
    units:    mm/day
```

Let's use the command **where** with temperature data to keep only precipitation at grid points where the max temperature is below 0°C, and plot it.

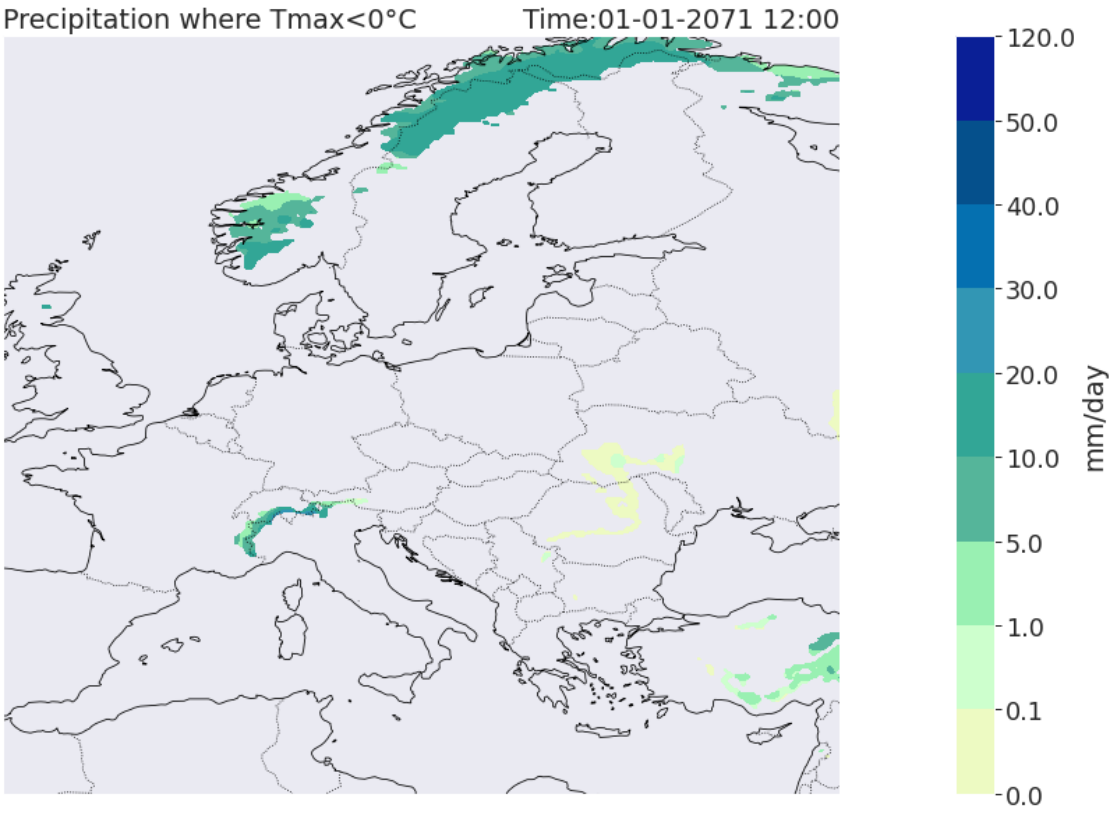
[48]:
```
data_precip.where(tasmax_degC_xclim.isel(time=0) < 0)
```

[48]:
```
<xarray.DataArray 'pr' (lat: 409, lon: 471)>
array([[nan, nan, nan, …, nan, nan, nan],
       [nan, nan, nan, …, nan, nan, nan],
       [nan, nan, nan, …, nan, nan, nan],
       …,
       [nan, nan, nan, …, nan, nan, nan],
       [nan, nan, nan, …, nan, nan, nan],
       [nan, nan, nan, …, nan, nan, nan]])
Coordinates:
    time     datetime64[ns] 2071-01-01T12:00:00
  * lon      (lon) float64 -10.0 -9.9 -9.8 -9.7 -9.6 … 36.7 36.8 36.9 37.0
  * lat      (lat) float64 30.0 30.1 30.2 30.3 30.4 … 70.4 70.5 70.6 70.7 70.8
Attributes:
```

```
    units:    mm/day
```

[49]:
```
fig = plt.figure(figsize=(30,10))
ax = plt.axes(projection=ccrs.PlateCarree())

my_levels = [0,0.1, 1, 5, 10,20,30, 40, 50,120]

my_colors = [(237/255,250/255,194/255),(205/255,255/255,205/255),(153/255,240/
↪255,178/255),
             (85/255,181/255,154/255),(50/255,166/255,150/255),(50/255,150/
↪255,180/255),
             (5/255,112/255,176/255),(5/255,80/255,140/255),(10/255,31/255,150/
↪255),
             (44/255,2/255,70/255),(106/255,44/255,90/255),(168/255,65/255,91/
↪255)]


a = data_precip.where(tasmax_degC_xclim.isel(time=0) < 0).plot.contourf(ax=ax,␣
↪transform=ccrs.PlateCarree(), levels = my_levels,
                                                                        ␣
↪colors=my_colors, add_colorbar=False)


ax.add_feature(cfeature.COASTLINE, linestyle='-')
ax.add_feature(cfeature.BORDERS, linestyle=':')


cbar = fig.colorbar(a, ax=ax, fraction = 0.1, label=r'mm/day')


ax.set_title('')
ax.set_title('Time:{}'.format(nice_time), loc='right');
ax.set_title('Precipitation where Tmax<0°C', loc='left');
```

```
/tmp/ipykernel_705509/3079342499.py:20: MatplotlibDeprecationWarning: Auto-
removal of grids by pcolor() and pcolormesh() is deprecated since 3.5 and will
be removed two minor releases later; please call grid(False) first.
  cbar = fig.colorbar(a, ax=ax, fraction = 0.1, label=r'mm/day')
```



1.4.4 Hatching

Hatching (or stipplings) can be used to show additional information on a contourf plot.

Let's show the areas with a population density greather than 1'000'000 people per 0.1x0.1 lon/lat square. (e.g. relevant for climate change impact on population)

For additionnal information, see: https://matplotlib.org/stable/gallery/images_contours_and_fields/contourf_ha

[50]:
```
ds_pop = xr.open_dataset('/lhome/cra2022/climriskdata/EUR-11S/
  ↪Estimated_population/Estimated_population_2091_LL.nc')
ds_pop
```

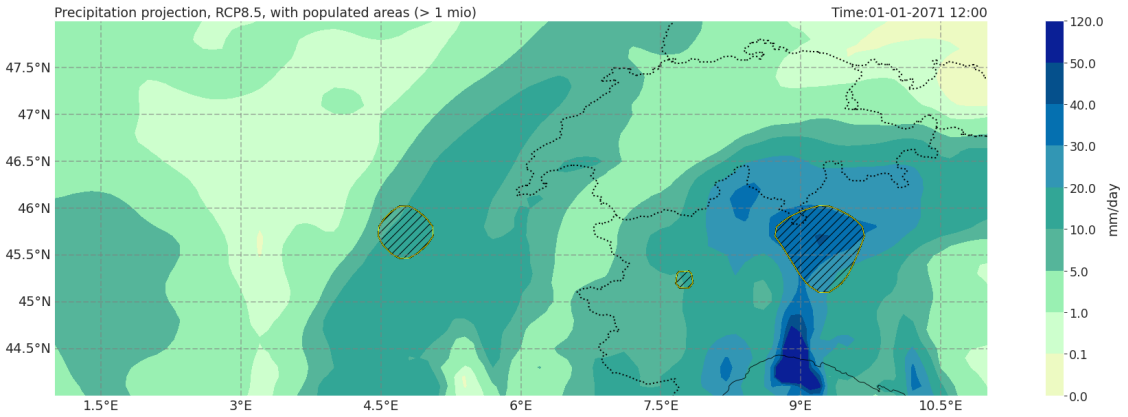
[50]:
```
<xarray.Dataset>
Dimensions:     (lon: 471, lat: 409)
Coordinates:
  * lon         (lon) float64 -10.0 -9.9 -9.8 -9.7 -9.6 … 36.7 36.8 36.9 37.0
  * lat         (lat) float64 30.0 30.1 30.2 30.3 30.4 … 70.5 70.6 70.7 70.8
Data variables:
    population  (lat, lon) float32 …
Attributes:
    CDI:          Climate Data Interface version 1.6.4 (http://code.zmaw.de/p…
```

```
    Conventions:  CF-1.4
    history:      Wed Feb 13 17:30:05 2019: cdo mul tmp1.nc Land_Surface_Mask…
    created_by:   R, packages ncdf and raster (version 2.0-12)
    date:         2012-11-08 14:47:12
    CDO:          Climate Data Operators version 1.6.4 (http://code.zmaw.de/p…
```

[52]:
```
fig = plt.figure(figsize=(30,10))
ax = plt.axes(projection=ccrs.PlateCarree())
ax.set_extent([1,11, 44, 48], crs=ccrs.PlateCarree())
my_levels = [0,0.1, 1, 5, 10,20,30, 40, 50,120]

my_colors = [(237/255,250/255,194/255),(205/255,255/255,205/255),(153/255,240/
 ↪255,178/255),
             (85/255,181/255,154/255),(50/255,166/255,150/255),(50/255,150/
 ↪255,180/255),
             (5/255,112/255,176/255),(5/255,80/255,140/255),(10/255,31/255,150/
 ↪255),
             (44/255,2/255,70/255),(106/255,44/255,90/255),(168/255,65/255,91/
 ↪255)]


a = data_precip.plot.contourf(ax=ax, transform=ccrs.PlateCarree(), levels =␣
 ↪my_levels, colors=my_colors, add_colorbar=False)

# Areas with > 1'000'000 people are hatched with the pattern ['//']
# whereas areas with less than 1'000'000 are left transparent ['']
d = ds_pop.population.plot.contourf(ax=ax, transform=ccrs.
 ↪PlateCarree(),levels=[0,1000000], colors='none', hatches=['','//'],␣
 ↪add_colorbar=False)

# Hatch color has to be changed afterwards has edgecolor
d.collections[1].set_edgecolor('black')

# Add a contour for clarity
ds_pop.population.plot.contour(ax=ax, transform=ccrs.PlateCarree(),␣
 ↪levels=[1000000], colors = 'yellow', linewidths=1, add_colorbar=False)

ax.add_feature(cfeature.COASTLINE, linestyle='-')
ax.add_feature(cfeature.BORDERS, linestyle=':', linewidth=2)

cbar = fig.colorbar(a, ax=ax, fraction = 0.1, label=r'mm/day')

gl = ax.gridlines(crs=ccrs.PlateCarree(), draw_labels=True,
                  linewidth=2, color='gray', alpha=0.5, linestyle='--')
gl.top_labels = False # suppress gridline labels on the top
gl.right_labels = False # suppress gridline labels at the right edge
```

```
ax.set_title('')
ax.set_title('Time:{}'.format(nice_time), loc='right');
ax.set_title('Precipitation projection, RCP8.5, with populated areas (> 1␣
 ↪mio)', loc='left');
```

```
/tmp/ipykernel_705509/529527054.py:27: MatplotlibDeprecationWarning: Auto-
removal of grids by pcolor() and pcolormesh() is deprecated since 3.5 and will
be removed two minor releases later; please call grid(False) first.
  cbar = fig.colorbar(a, ax=ax, fraction = 0.1, label=r'mm/day')
```



The yellow hatched domains correctly show the urban areas of Lyon, Torino and Milano (frow West to East).