Plotting

July 19, 2022

`Aim:` *Notebook for the purpose of Climate Risk Assessment lecture:* `Xarray and Plotting`

`Author:` Mubashshir Ali

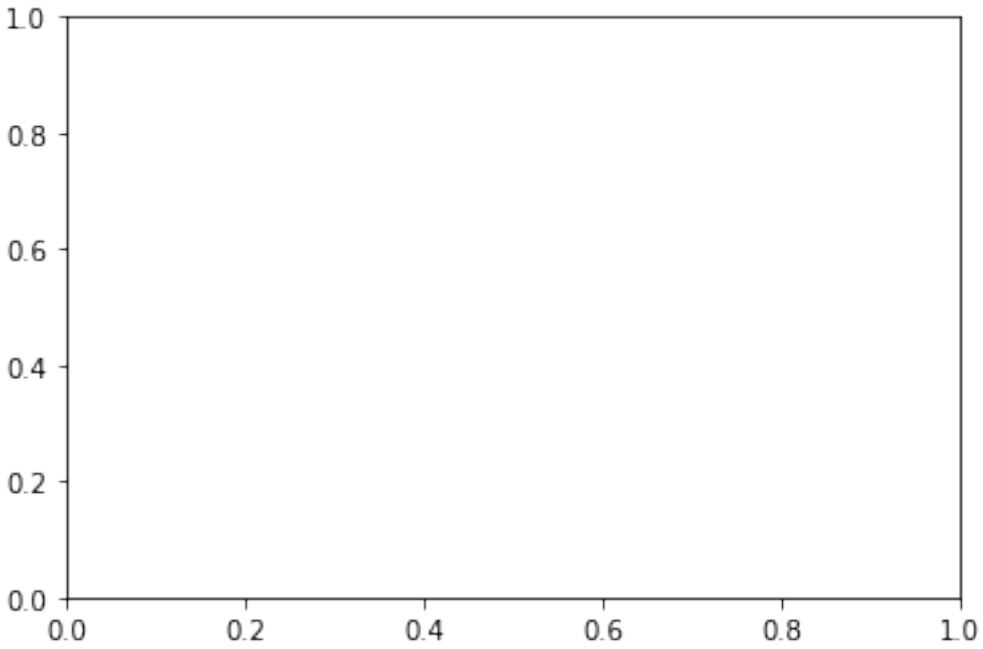
1 Imports

[2]:
```
# importing libraries as an alias so that we know which function belongs to␣
 ↪which library
from IPython.display import Image
import matplotlib.pyplot as plt
import numpy as np
```

2 Plotting a simple canvas

Let's first plot a simple canvas on which we will draw figures later-on

[3]:
```
fig = plt.figure()
ax = plt.axes()
```



You can use `shift + tab` key to view the syntax of a function and other options it provides. Test it below.

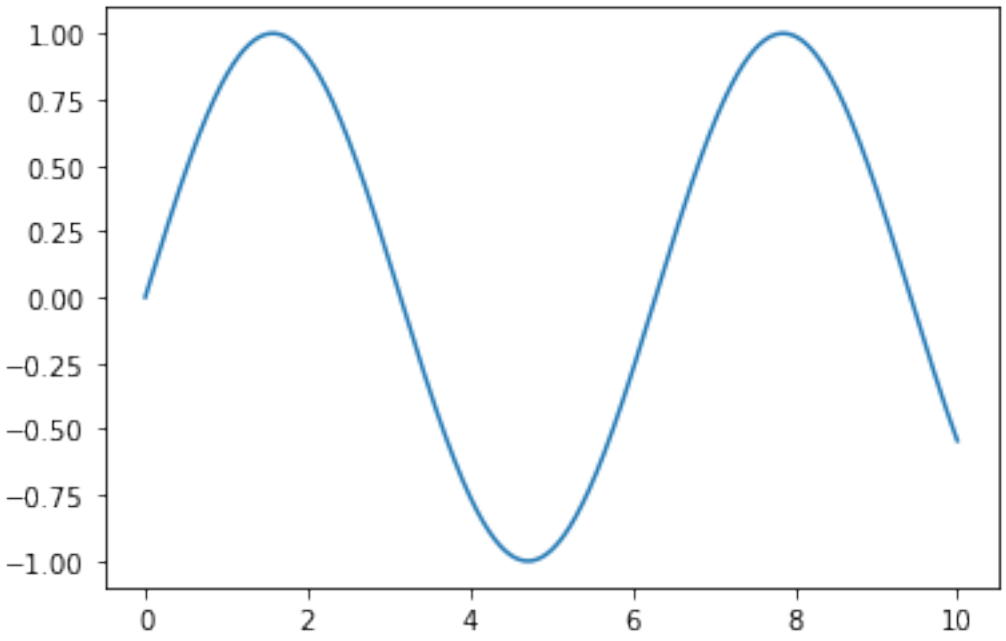
[4]:
```
plt.figure();
```

```
<Figure size 432x288 with 0 Axes>
```

Now, let's plot a simple function, the classical `sine` curve example

[5]:
```
# create an array of points
x = np.linspace(0, 10, 1000) # use shift + tab to see the syntax
y = np.sin(x) # our function

# lets create our canvas as we did above
fig = plt.figure()
# add axes
ax = plt.axes()
ax.plot(x, y)
```

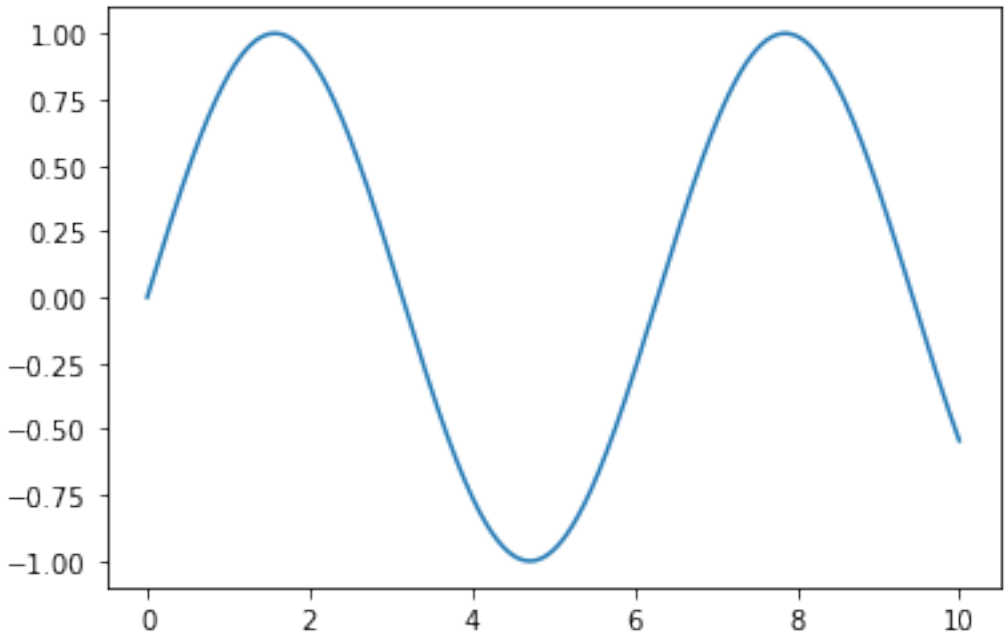
[5]:
```
[<matplotlib.lines.Line2D at 0x7fda850ccfa0>]
```



This could also be plotted lazily in just one line...

[6]:
```
plt.plot(x, np.sin(x))
```

[6]: [<matplotlib.lines.Line2D at 0x7fda85182160>]



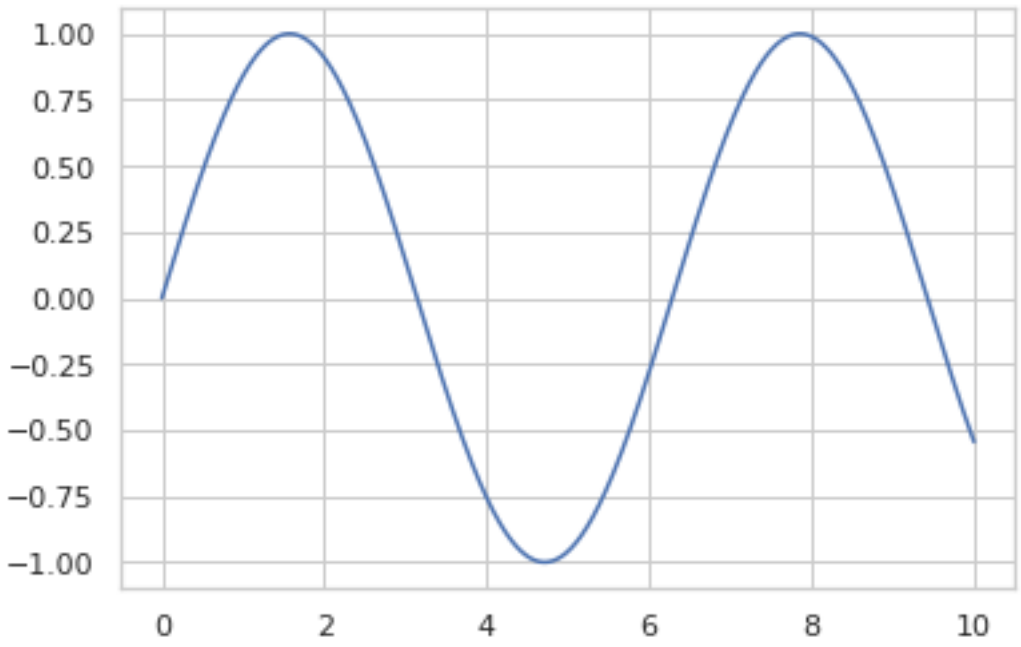
2.1 Making nicer plots with seaborn

[7]:
```
import seaborn as sns
sns.set(style='whitegrid') # sets default plotting styles, check contextual␣
 ↪help for more options
```

Same plot as above

[8]:
```
plt.plot(x, np.sin(x))
```

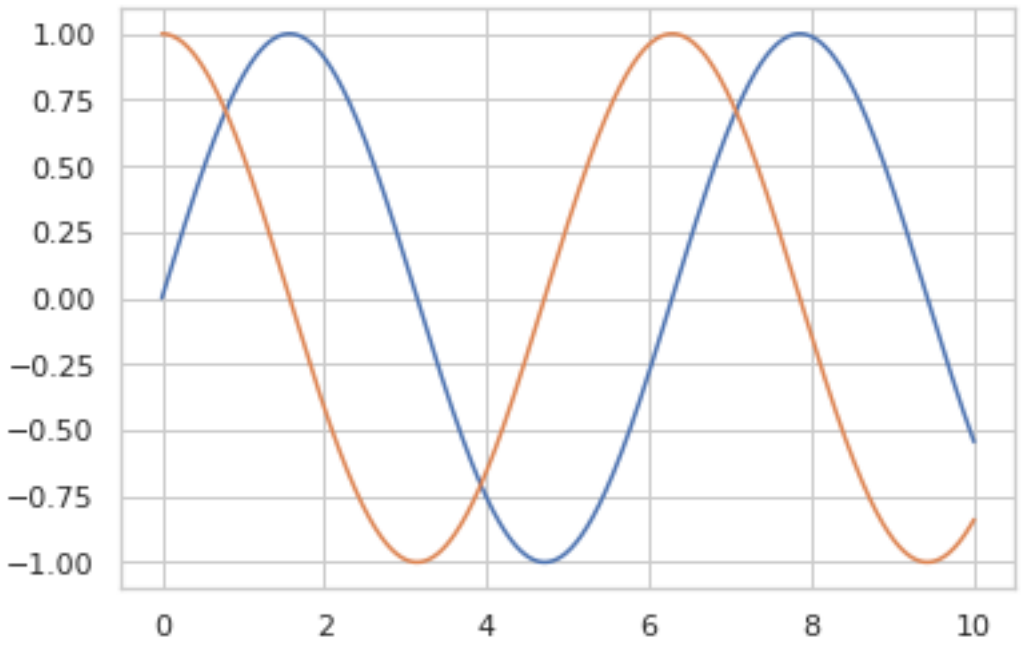
[8]: [<matplotlib.lines.Line2D at 0x7fda07f70790>]



2.2 Ploting multiple lines

[9]:
```
# lets do the lazy ploting of multiple lines
plt.plot(x, np.sin(x))
plt.plot(x, np.cos(x))
```

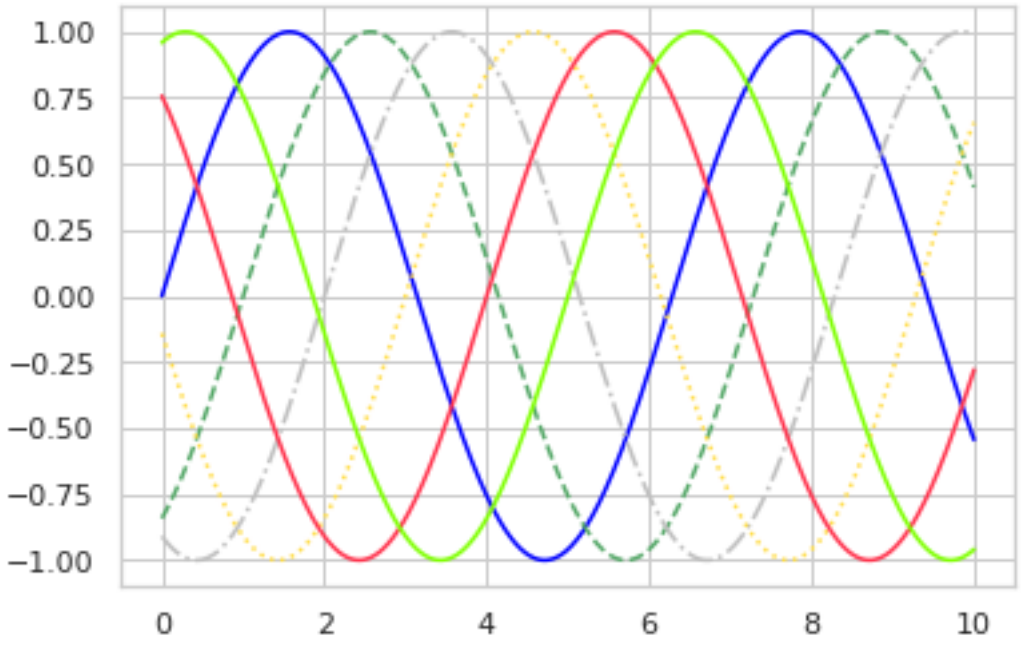
[9]:
```
[<matplotlib.lines.Line2D at 0x7fda07ed9a90>]
```



Note that matplotlib automatically assigned different colours. Pretty smart, right? What if you don't like those colours? Let's see how we can customize it…

2.3 Adjusting colors and linestyles

[10]:
```
plt.plot(x, np.sin(x - 0), color='blue', linestyle='-')          # specify color␣
 ↪by name
plt.plot(x, np.sin(x - 1), color='g', linestyle='--')             # short color␣
 ↪code (rgbcmyk)
plt.plot(x, np.sin(x - 2), color='0.75', linestyle='-.')          # Grayscale␣
 ↪between 0 and 1
plt.plot(x, np.sin(x - 3), color='#FFDD44', linestyle=':')       # Hex code␣
 ↪(RRGGBB from 00 to FF)
plt.plot(x, np.sin(x - 4), color=(1.0,0.2,0.3)) # RGB tuple, values 0 and 1
plt.plot(x, np.sin(x - 5), color='chartreuse'); # all HTML color names supported
```



Homework Task: Find a website with a list of HTML color names

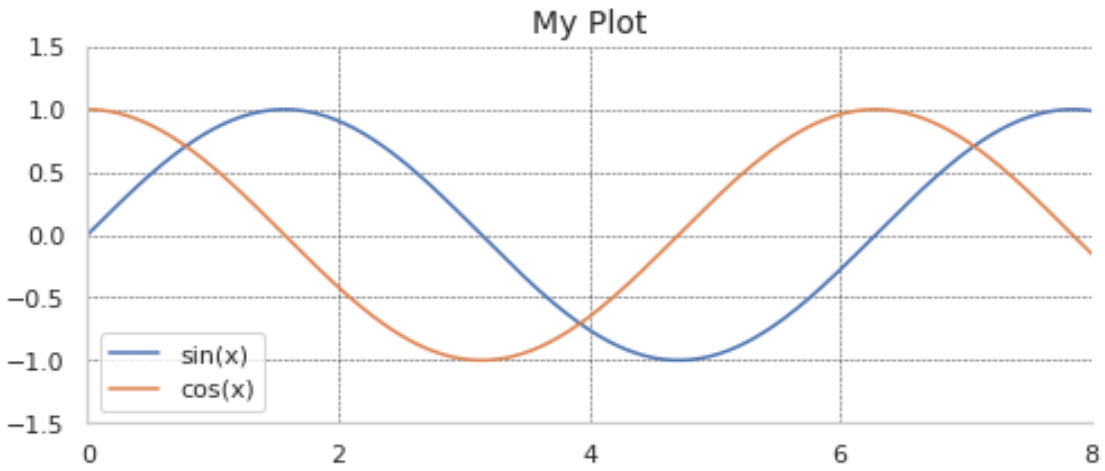
2.4 Figure elements and Aesthetics

[11]:
```
Image(url="https://matplotlib.org/_images/sphx_glr_anatomy_001.png")
```

[11]:
```
<IPython.core.display.Image object>
```

[13]:
```
# customize our plot by controlling some figure elements

x = np.linspace(0, 10, 1000)
y1 = np.sin(x) # our function
y2 = np.cos(x)

# lets create our canvas as we did above
fig = plt.figure(figsize=(8,8))
# adding axis. For multiple plots add another axis separately
ax = fig.add_subplot(1,1,1, aspect=1) # rows, columns, number

ax.plot(x, y1, label='sin(x)')
ax.plot(x, y2, label='cos(x)')
ax.set_title('My Plot', fontsize=14)

# tweaking minor parameters to our liking
ax.set(xlim=(0,8), ylim=(-1.5,1.5), xticks=np.arange(0,8.1, 2))
ax.legend()
```

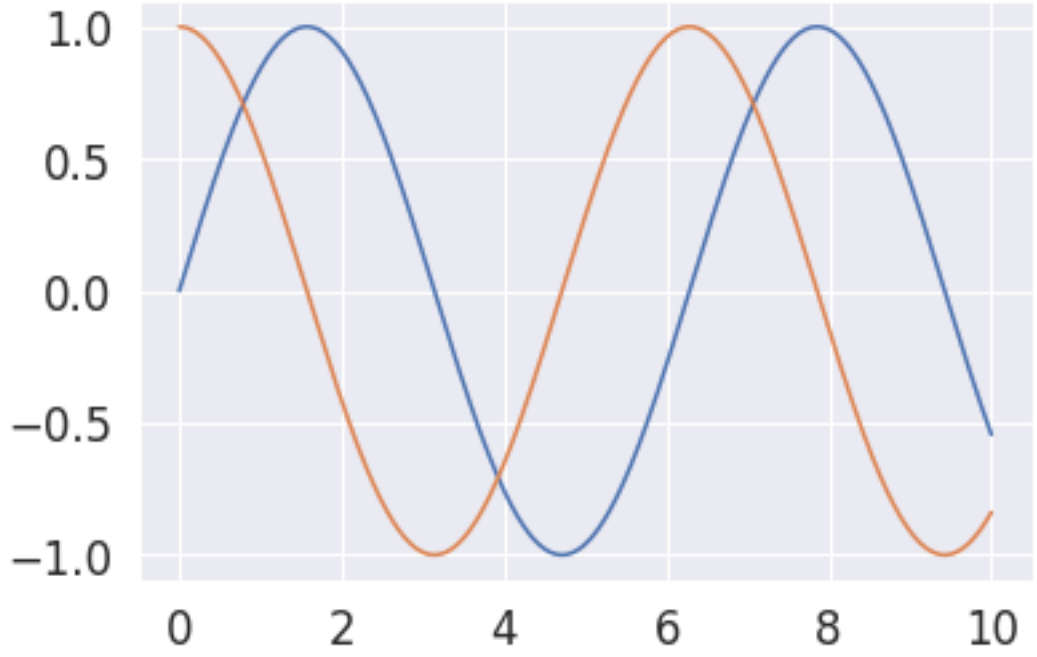
```
ax.grid(linestyle="--", linewidth=0.5, color='.25', zorder=-10)
sns.despine(ax=ax)
```



[14]:
```
sns.set(style='darkgrid', font_scale=1.5)
```

[15]:
```
# customize our plot by controlling some figure elements

plt.plot(x, np.sin(x))
plt.plot(x, np.cos(x))
```

[15]: [<matplotlib.lines.Line2D at 0x7fda07d59880>]



More examples on controlling figure aesthetics

[16]:
```
# changed to ticks
# I personally prefer it for geographic plotting
# feel free to play around
sns.set(style='ticks')
```

3 Geographic ploting

So far we have seen plots in the Cartesian coordinate system(x-y). Python also provides powerful libraries such as `Cartopy` to enable geographic ploting.

Let's import the libary as an alias

[17]:
```
import cartopy.crs as ccrs # for geographic plotting
import cartopy.feature as cfeature
```

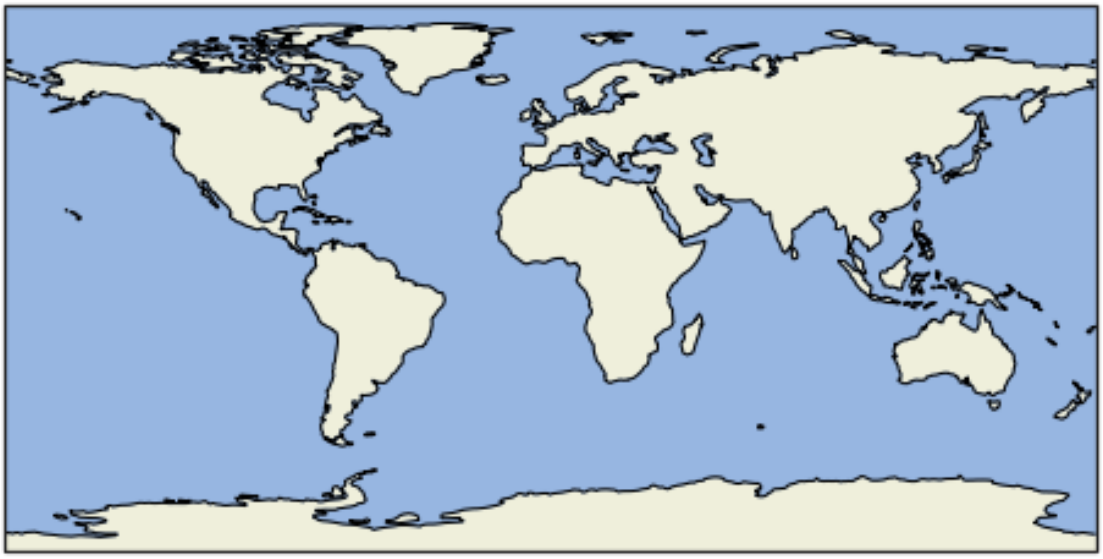
Let's plot a simple map

[18]:
```
fig = plt.figure(figsize=[6,6])
# What are the units of the figure size? Try to find out.
#
# Geographic maps can have different projection system.
# Now, we will specify that we want the axes of our figure in PlateCarree␣
 ↪projection.
# We won't go into details of projection
ax = plt.axes(projection=ccrs.PlateCarree())
```

Oops! That doesn't look like the map we imagined. So, lets add ocean and land to our geograhical map. We can do that by calling `ax.add_feature()` .

[19]:
```
fig = plt.figure(figsize=[8,6])
ax = plt.axes(projection=ccrs.PlateCarree())

ax.add_feature(cfeature.LAND)
ax.add_feature(cfeature.OCEAN)
ax.add_feature(cfeature.COASTLINE)
ax.set_global()
```

```
/usr/local/miniconda3/lib/python3.9/site-packages/cartopy/io/__init__.py:241:
DownloadWarning: Downloading:
https://naciscdn.org/naturalearth/110m/physical/ne_110m_land.zip
  warnings.warn('Downloading: {}'.format(url), DownloadWarning)
/usr/local/miniconda3/lib/python3.9/site-packages/cartopy/io/__init__.py:241:
DownloadWarning: Downloading:
https://naciscdn.org/naturalearth/110m/physical/ne_110m_ocean.zip
  warnings.warn('Downloading: {}'.format(url), DownloadWarning)
```



3.1 Adding more features

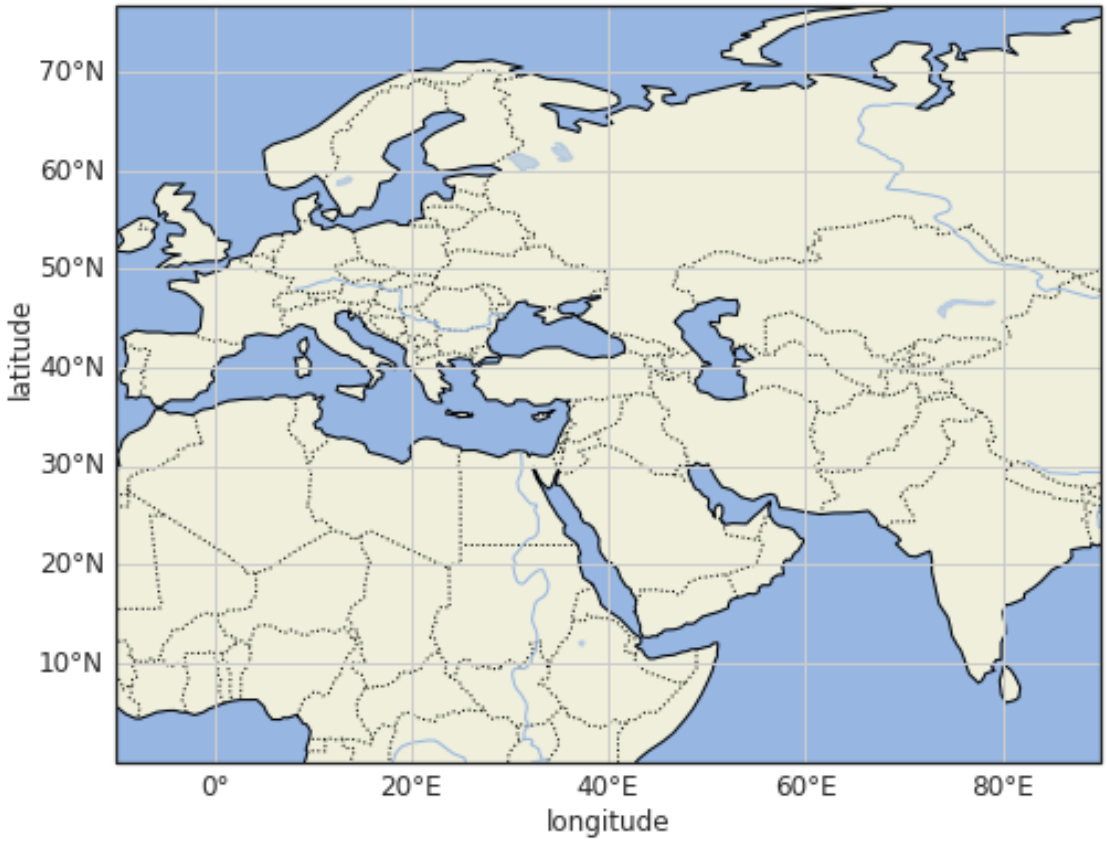
We can also add features like country borders, rivers, lakes, etc.

[20]:
```
fig = plt.figure(figsize=[8,8])
ax = plt.axes(projection=ccrs.PlateCarree())

# with set_extent we can zoom in by specyfing a smaller region
ax.set_extent([-10, 90, 0, 70])
```

```
# add features
ax.add_feature(cfeature.LAND)
ax.add_feature(cfeature.OCEAN)
ax.add_feature(cfeature.COASTLINE)
ax.add_feature(cfeature.BORDERS, linestyle=':')
ax.add_feature(cfeature.LAKES, alpha=0.5)
ax.add_feature(cfeature.RIVERS)

# grid lines
gridlines = ax.gridlines(draw_labels=True, )
gridlines.top_labels=False
gridlines.right_labels=False

## Note: Normal labeling doesn't work as Cartopy's labeling takeover the␣
 ↪matplotlib commands
# ax.set_xlabel('longitude')
# ax.set_ylabel('latitude')


# Here is a work around
ax.text(-0.08, 0.55, 'latitude', va='bottom', ha='center',
        rotation='vertical', rotation_mode='anchor',
        transform=ax.transAxes)
ax.text(0.5, -0.1, 'longitude', va='bottom', ha='center',
        rotation='horizontal', rotation_mode='anchor',
        transform=ax.transAxes)
```

[20]: Text(0.5, -0.1, 'longitude')

```
/usr/local/miniconda3/lib/python3.9/site-packages/cartopy/io/__init__.py:241:
DownloadWarning: Downloading:
https://naciscdn.org/naturalearth/110m/physical/ne_110m_lakes.zip
  warnings.warn('Downloading: {}'.format(url), DownloadWarning)
/usr/local/miniconda3/lib/python3.9/site-packages/cartopy/io/__init__.py:241:
DownloadWarning: Downloading: https://naciscdn.org/naturalearth/110m/physical/ne
_110m_rivers_lake_centerlines.zip
  warnings.warn('Downloading: {}'.format(url), DownloadWarning)
```



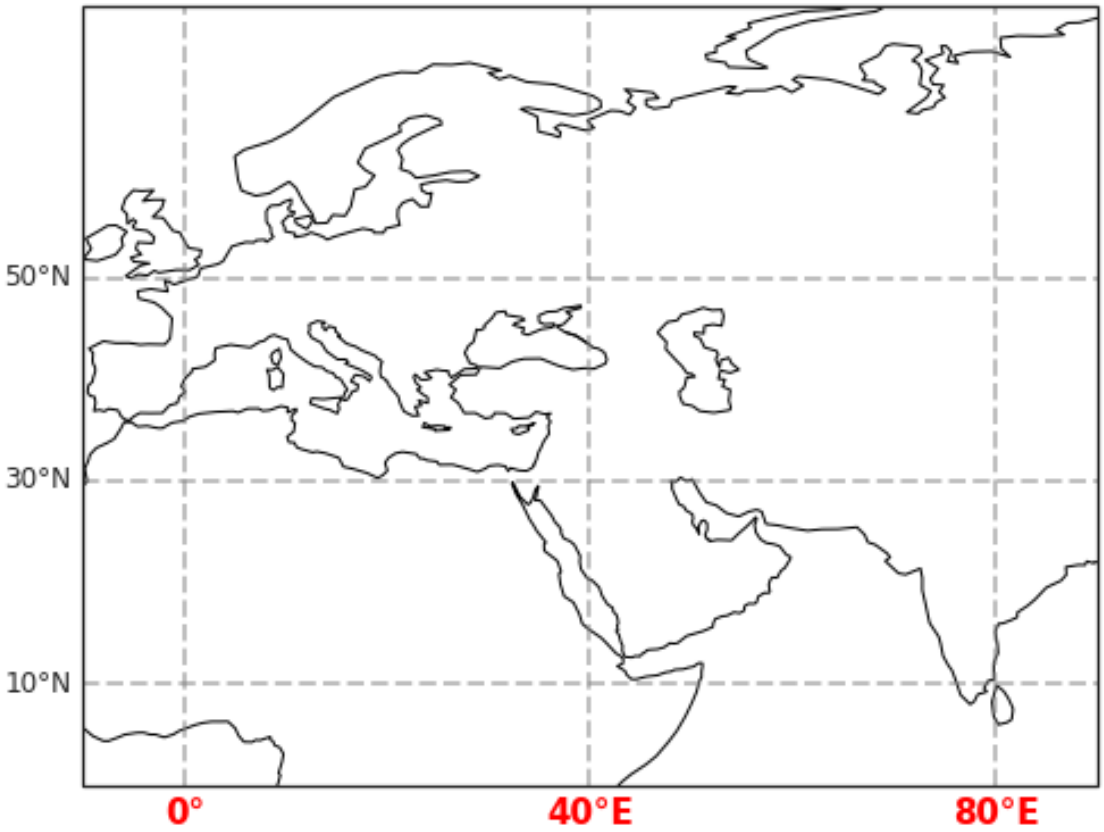
3.1.1 Drawing gridlines at only specific points and some additional gridline controls

For additional control over gridlines, e.g. fontsize, colour, etc.

[21]:
```
import matplotlib.ticker as mticker
from cartopy.mpl.ticker import (LongitudeFormatter, LatitudeFormatter,
                                LatitudeLocator)
```

[22]:
```
fig = plt.figure(figsize=[8,8])
ax = plt.axes(projection=ccrs.PlateCarree())
ax.set_extent([-10, 90, 0, 70])
ax.add_feature(cfeature.COASTLINE)

gridlines = ax.gridlines(crs=ccrs.PlateCarree(), # projection info is needed if␣
 ↪you plot in different projections
                                                 # not needed for our use case
                         draw_labels=True, linewidth=2, color='gray', alpha=0.
 ↪5, linestyle='--'
                        )
gridlines.top_labels=False
```

```
gridlines.right_labels=False

gridlines.xlocator = mticker.FixedLocator([0, 40, 80])
gridlines.ylocator = mticker.FixedLocator([10, 30, 50])
gridlines.xformatter = LongitudeFormatter() # check out syntax using Contextual␣
 ↪help for more info
gridlines.yformatter = LatitudeFormatter()
gridlines.xlabel_style = {'size': 15, 'color': 'red', 'weight': 'bold'}
```



3.2 Plotting NetCDF file

3.2.1 Quick Recap

[29]:
```
Image(url='http://xarray.pydata.org/en/stable/_images/dataset-diagram.png')
```

[29]:
```
<IPython.core.display.Image object>
```

[30]:
```
import xarray as xr
# library to work with climate model data format
```

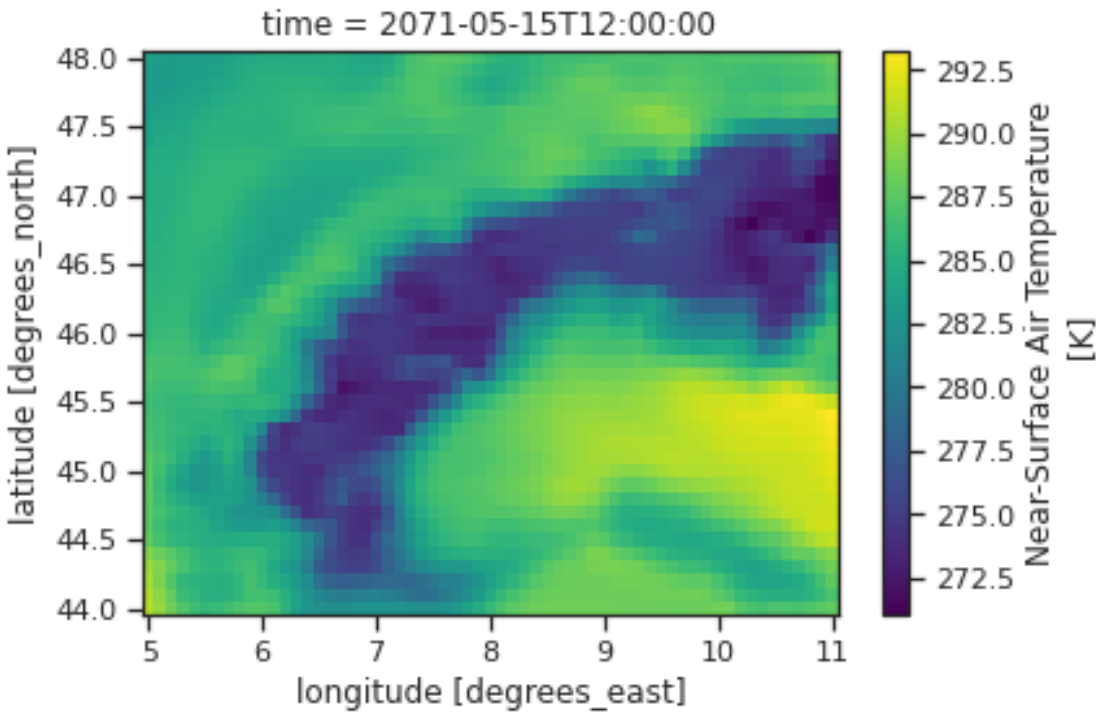
```
# file to be plotted
# /scratch3/climriskdata/EUR-11N/ICHEC-EC-EARTH_SMHI-RCA4_v1/rcp85/tas/
 ↪reduced_tas_EUR-11_ICHEC-EC-EARTH_rcp85_r12i1p1_SMHI-RCA4_v1_day_20710101-20751231_LL.
 ↪nc'
file1 = '~/cra2022/climriskdata/EUR-11N/ICHEC-EC-EARTH_SMHI-RCA4_v1/rcp85/tas/
 ↪reduced_tas_EUR-11_ICHEC-EC-EARTH_rcp85_r12i1p1_SMHI-RCA4_v1_day_20710101-20751231_LL.
 ↪nc'
# lets load the data by using open_dataset function of xarray
ds1 = xr.open_dataset(file1)
```

[32]:
```
ds1
```

[32]:
```
<xarray.Dataset>
Dimensions:    (time: 1826, bnds: 2, lon: 61, lat: 41)
Coordinates:
  * time       (time) datetime64[ns] 2071-01-01T12:00:00 … 2075-12-31T12:00:00
  * lon        (lon) float64 5.0 5.1 5.2 5.3 5.4 … 10.6 10.7 10.8 10.9 11.0
  * lat        (lat) float64 44.0 44.1 44.2 44.3 44.4 … 47.7 47.8 47.9 48.0
Dimensions without coordinates: bnds
Data variables:
    time_bnds  (time, bnds) datetime64[ns] 2071-01-01 2071-01-02 … 2076-01-01
    tas        (time, lat, lon) float32 …
Attributes: (12/25)
    CDI:                          Climate Data Interface version ?? (http:/…
    history:                      Fri Mar 13 11:12:41 2020: cdo sellonlatbo…
    institution:                  Swedish Meteorological and Hydrological I…
    Conventions:                  CF-1.4
    contact:                      rossby.cordex@smhi.se
    creation_date:                2013-07-03-T20:13:36Z
    …                             …
    references:                   http://www.smhi.se/en/Research/Research-d…
    tracking_id:                  18802e76-c2be-49a8-bbe6-c460dcd5960a
    rossby_comment:               201139: CORDEX Europe 0.11 deg | RCA4 v1 …
    rossby_run_id:                201139
    rossby_grib_path:             /nobackup/rossby16/rossby/joint_exp/corde…
    CDO:                          Climate Data Operators version 1.9.3 (htt…
```

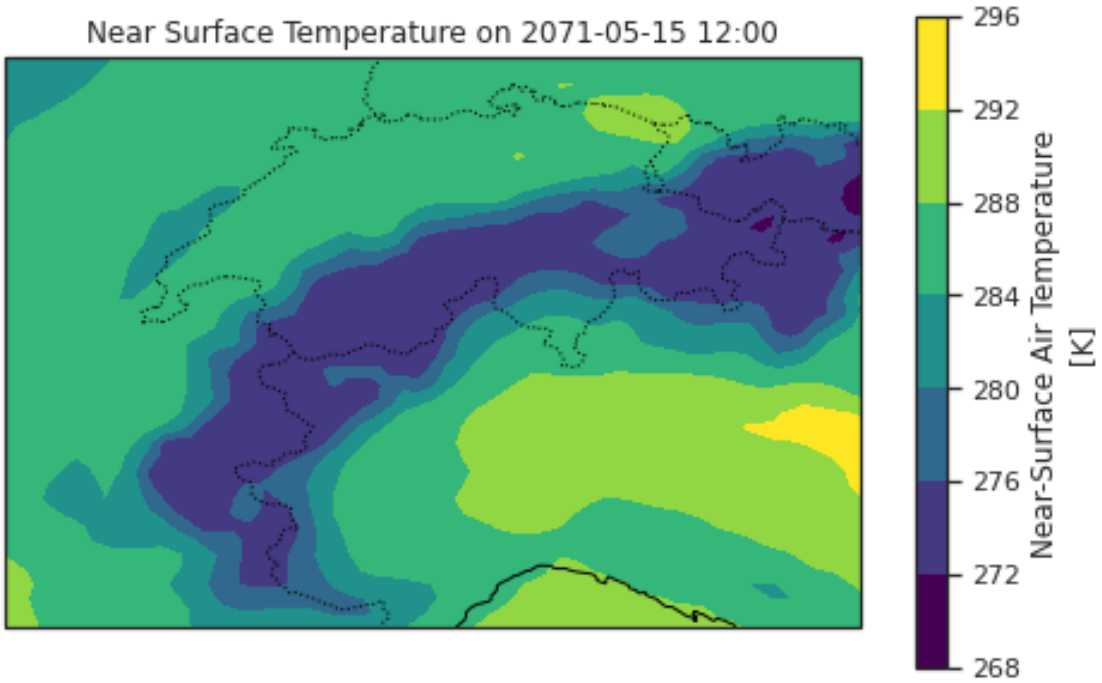
[33]:
```
data_to_plot = ds1.tas.sel(time='2071-05-15 12:00')
data_to_plot.plot()
```

[33]:
```
<matplotlib.collections.QuadMesh at 0x7f2699cdcac0>
```



Most of the time we need more control to make plots for publication quality. So, now we're going to have a look as to how we can control more aspects of our plot rather than letting `xarray` do it for us. We will try to stick with `xarray's` easy plotting method as much as possible that means, letting it choose most of the things for us. But, we will add more details wherever required to make the plot fit to our needs.

[35]:
```
fig = plt.figure(figsize=(8, 5))
ax = plt.axes(projection=ccrs.PlateCarree())



# now we're going to tell xarray that we want contourf plot specifically!

data_to_plot.plot.contourf(ax=ax, transform=ccrs.PlateCarree())
# transform means we're telling projection of our data to cartopy, not␣
 ↪important at this stage
# data we use here is by default in PlateCarree(lat, lon)

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

# setting the title

ax.set_title('Near Surface Temperature on 2071-05-15 12:00');
```



More colour-bars options

3.3 Levels

Lets say we want to see minute temperature variations and thus, we would like to divide our color bar into smaller segments. For that one could just use `levels` arguments. Of course, one can also make there own custom segments and specify it as a list.

[39]:
```
# fixing default time values
import pandas as pd # library to work with tabular data
# pandas also provide nice datetime manipulation functions
dt = pd.to_datetime(data_to_plot.time.data)
nice_time = dt.strftime('%d-%m-%Y %H:%M')
nice_time
```

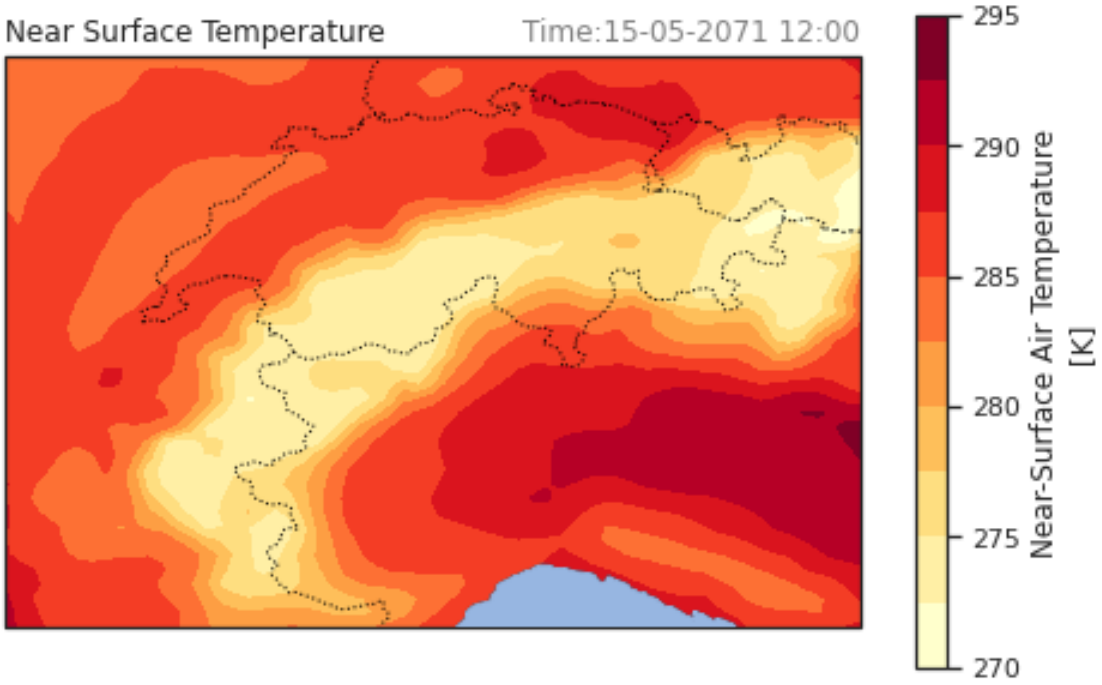
[39]:
```
'15-05-2071 12:00'
```

[38]:
```
data_to_plot.time.data
```

[38]:
```
array('2071-05-15T12:00:00.000000000', dtype='datetime64[ns]')
```

[43]:
```
fig = plt.figure(figsize=(8, 5))
ax = plt.axes(projection=ccrs.PlateCarree())

# setting geographical boundaries of our map using the data provided
```

```
ax.set_extent([5,11, 44, 48])
## Note: so far xarray was setting geographical boundaries on its own but we␣
 ↪can also specify the boundaries manually

data_to_plot.plot.contourf(ax=ax, transform=ccrs.PlateCarree(), cmap='YlOrRd',␣
 ↪levels=10) # added levels argument


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

# Adding ocean so that we only emphasize on Land
ax.add_feature(cfeature.OCEAN, zorder=10)

ax.set_title('')
ax.set_title('Near Surface Temperature', loc='left', fontsize=12);
ax.set_title('Time:{}'.format(nice_time), loc='right', fontsize=12,␣
 ↪color='grey');
```

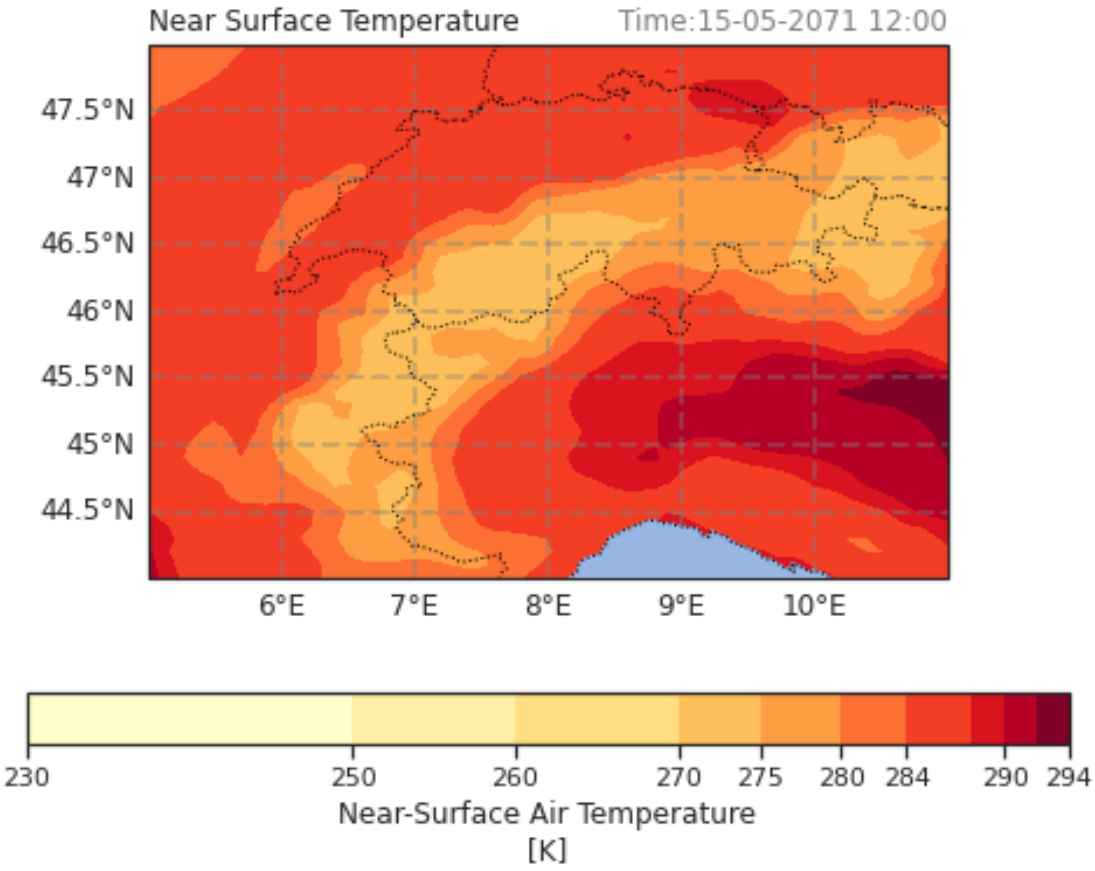


3.3.1 making custom levels

As mentioned above, we can also define custom levels for our color-bar

[47]:
```
# Irregular levels to illustrate the use of a proportional colorbar spacing
levels1 = [230, 250, 260, 270, 275, 280, 284, 288, 290, 292, 294]
ticks = [230, 250, 260, 270, 275, 280, 284, 290,  294]
```

[49]:
```
fig = plt.figure(figsize=(8,6))
ax = plt.axes(projection=ccrs.PlateCarree())

ax.add_feature(cfeature.COASTLINE, linestyle=':', zorder=20)
ax.add_feature(cfeature.BORDERS, linestyle=':', zorder=25)
ax.add_feature(cfeature.OCEAN, zorder=10)

gl = ax.gridlines(crs=ccrs.PlateCarree(), draw_labels=True,
                  linewidth=2, color='gray', alpha=0.4, linestyle='--')
gl.top_labels = False # suppress gridline labels on the top
gl.right_labels = False # suppress gridline labels at the right edge

data_to_plot.plot.contourf(ax=ax, transform=ccrs.PlateCarree(), cmap='YlOrRd',
                           levels=levels1,
                           # using cbar_kwargs one can control properties of␣
 ↪color bar
                           cbar_kwargs={'ticks':ticks,'spacing': 'proportional',
                                        'orientation': 'horizontal',
                                       # 'shrink':0.8
                                       }
                           # add shrink and see the difference
                             )

ax.set_title('')
ax.set_title('Near Surface Temperature', loc='left', fontsize=12);
ax.set_title('Time:{}'.format(nice_time), loc='right', fontsize=12,␣
 ↪color='grey');
```



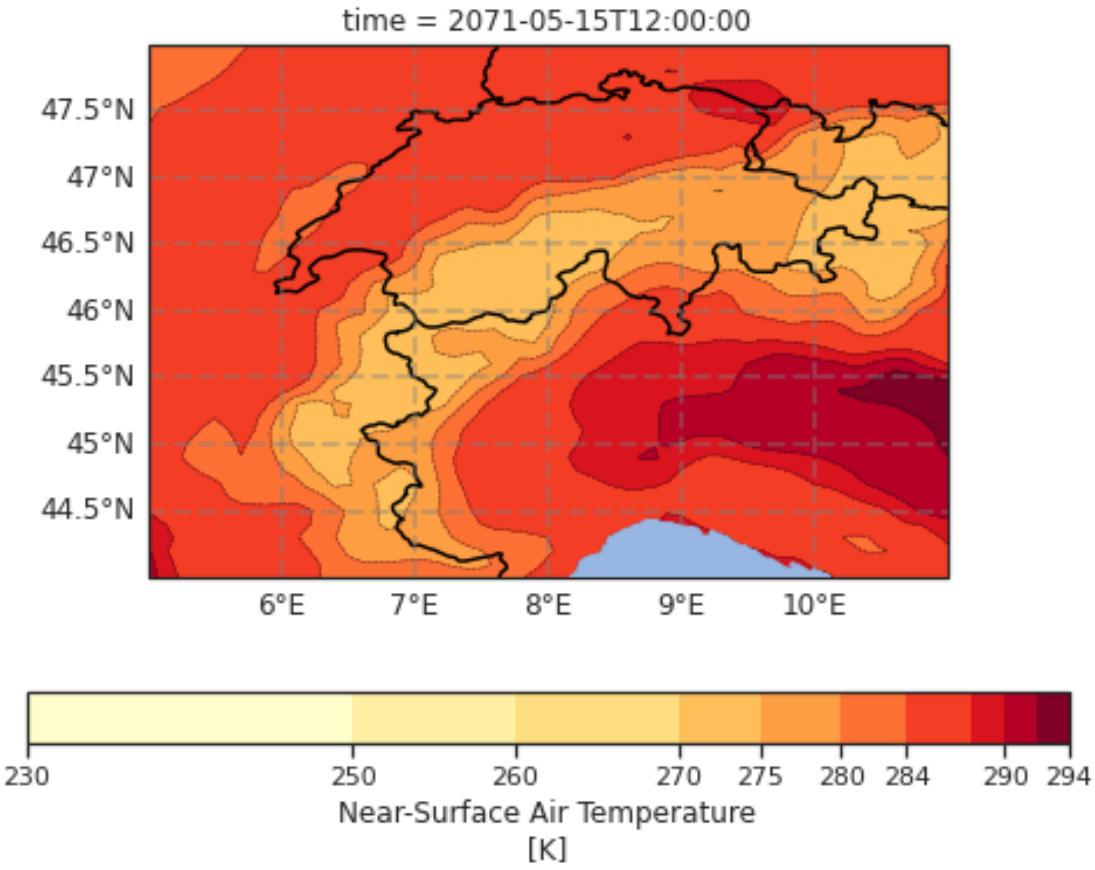
3.4 Plotting Contours

Contours are lines representing points with a common numerical value.

[54]:
```
fig = plt.figure(figsize=(8,6))
ax = plt.axes(projection=ccrs.PlateCarree())



gl = ax.gridlines(crs=ccrs.PlateCarree(), draw_labels=True,
                  linewidth=2, color='gray', alpha=0.4, linestyle='--')
gl.top_labels = False # suppress gridline labels on the top
gl.right_labels = False # suppress gridline labels at the right edge



data_to_plot.plot.contourf(ax=ax, transform=ccrs.PlateCarree(), cmap='YlOrRd',
                           levels=levels1,
```

```
                            # using cbar_kwargs one can control properties of␣
 ↪color bar
                            cbar_kwargs={'ticks':ticks,'spacing': 'proportional',
                                         'orientation': 'horizontal',
                                        # 'shrink':0.8
                                        }
                            # add shrink and see the difference
                              )

data_to_plot.plot.contour(ax=ax, transform=ccrs.PlateCarree(),
                            levels=levels1, colors='k', linestyles='dashed',
                           linewidths=0.5, alpha=0.8,
                           )

ax.add_feature(cfeature.COASTLINE, linestyle=':')
ax.add_feature(cfeature.BORDERS, linestyle='-', linewidths=1.5, )
ax.add_feature(cfeature.OCEAN, zorder=10)
```

[54]: <cartopy.mpl.feature_artist.FeatureArtist at 0x7f26983f8e80>



`Homework:` Find out different `linestyles` options.

Summary so far: * For non geographical plotting we use `matplotlib` library which produces high quality scientific plots. * For geographic plotting we add `cartopy` to our `matplotlib` plots to take care of geographical axis and map projections. * For quick geographical plotting, use high-level plotting features of `xarray`. * For absolute control of your geographical plots use `cartopy` directly.

[ ]:

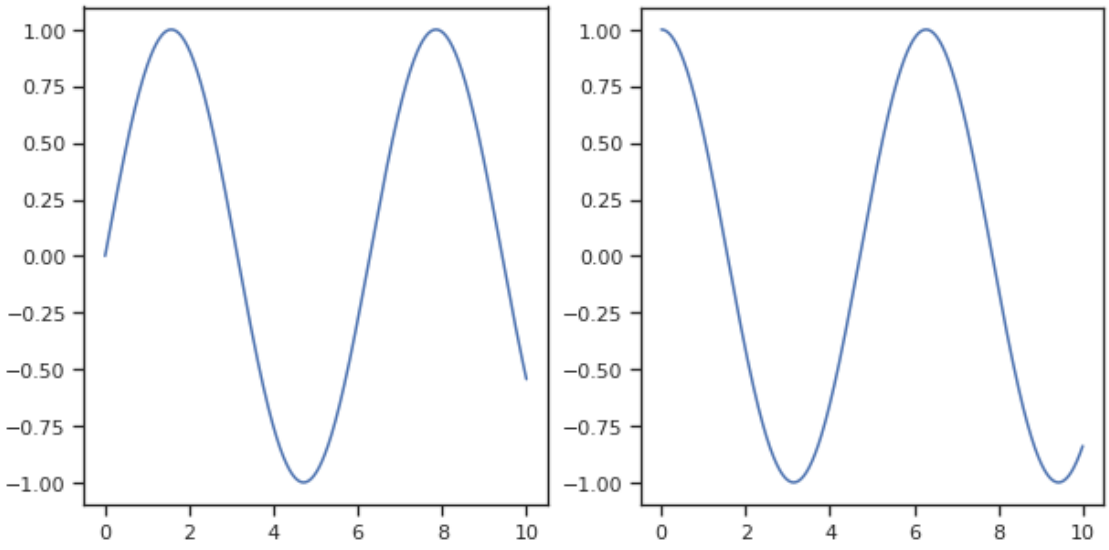
4 Plotting multiple files using for loop

[ ]:
```
# custom code
# list of all files in the sub-directory sorted according to file name
all_files=['path/to/file1.nc', 'path/to/file2.nc', 'path/to/file3.nc',]
save_names=['plot1', 'plot2', 'plot3']

# loop over all files
for i in range(len(all_files)):
    # open file
    ds = xr.open_dataset(all_files[i])

    data_to_plot=all_files[i]
    # paste your plotting routine below #



    plt.savefig('/save/to/my_directory/'+ save_names[i] +'.png', dpi=100,␣
 ↪bbox_inches='tight' )
    plt.close() # make sure not to crash the notebook by keeping 20-30 plots␣
 ↪open
    print('Finished plotting for {}'.format(all_files[i]))
```

5 Making Subplots

[55]:
```
fig, [ax1, ax2] = plt.subplots(nrows=1, ncols=2, figsize=(10,5))
x = np.linspace(0, 10, 1000)
y1 = np.sin(x) # our function
y2 = np.cos(x)
ax1.plot(x, y1)
ax2.plot(x, y2)
```

[55]:
```
[<matplotlib.lines.Line2D at 0x7f2698291070>]
```



5.1 Geographic subplots

[56]:
```
data_list = [ds1.tas.isel(time=0), ds1.tas.isel(time=1), ds1.tas.isel(time=-2),␣
↪ds1.tas.isel(time=-1)]
```

[57]:
```
len(data_list)
```

[57]:
```
4
```

[58]:
```
data_list[0]
```

[58]:
```
<xarray.DataArray 'tas' (lat: 41, lon: 61)>
array([[282.3606 , 281.92233, 281.38867, …, 281.33218, 281.44113, 281.81854],
       [282.17743, 281.61105, 280.8188 , …, 280.02493, 280.3555 , 280.87204],
       [281.76346, 281.17737, 280.14703, …, 280.2751 , 280.7882 , 281.293  ],
       …,
       [279.64423, 279.542  , 279.47067, …, 279.77545, 279.8155 , 279.86526],
       [279.83334, 279.6843 , 279.56104, …, 279.0776 , 279.01532, 278.93643],
       [279.9746 , 279.82288, 279.6683 , …, 278.4002 , 278.3426 , 278.20987]],
      dtype=float32)
Coordinates:
    time     datetime64[ns] 2071-01-01T12:00:00
  * lon      (lon) float64 5.0 5.1 5.2 5.3 5.4 5.5 … 10.6 10.7 10.8 10.9 11.0
  * lat      (lat) float64 44.0 44.1 44.2 44.3 44.4 … 47.6 47.7 47.8 47.9 48.0
Attributes:
    standard_name:  air_temperature
    long_name:      Near-Surface Air Temperature
    units:          K
```

```
    cell_methods:   time: mean
```

Lets plot anomalies with respect to mean temperature data

[59]:
```
mean_temp=ds1.tas.mean() # mean of whole data
```

[60]:
```
mean_temp
```

[60]:
```
<xarray.DataArray 'tas' ()>
array(281.4577, dtype=float32)
```

[67]:
```
axes.flatten()
```

[67]:
```
array([<GeoAxesSubplot:title={'left':'My subplot'}, xlabel='longitude
[degrees_east]', ylabel='latitude [degrees_north]'>,
       <GeoAxesSubplot:title={'left':'My subplot'}, xlabel='longitude
[degrees_east]', ylabel='latitude [degrees_north]'>,
       <GeoAxesSubplot:title={'left':'My subplot'}, xlabel='longitude
[degrees_east]', ylabel='latitude [degrees_north]'>,
       <GeoAxesSubplot:title={'left':'My subplot','center':'My plot'},
xlabel='longitude [degrees_east]', ylabel='latitude [degrees_north]'>],
      dtype=object)
```

[69]:
```
{'my_class':1}
```

[69]:
```
{'my_class': 1}
```

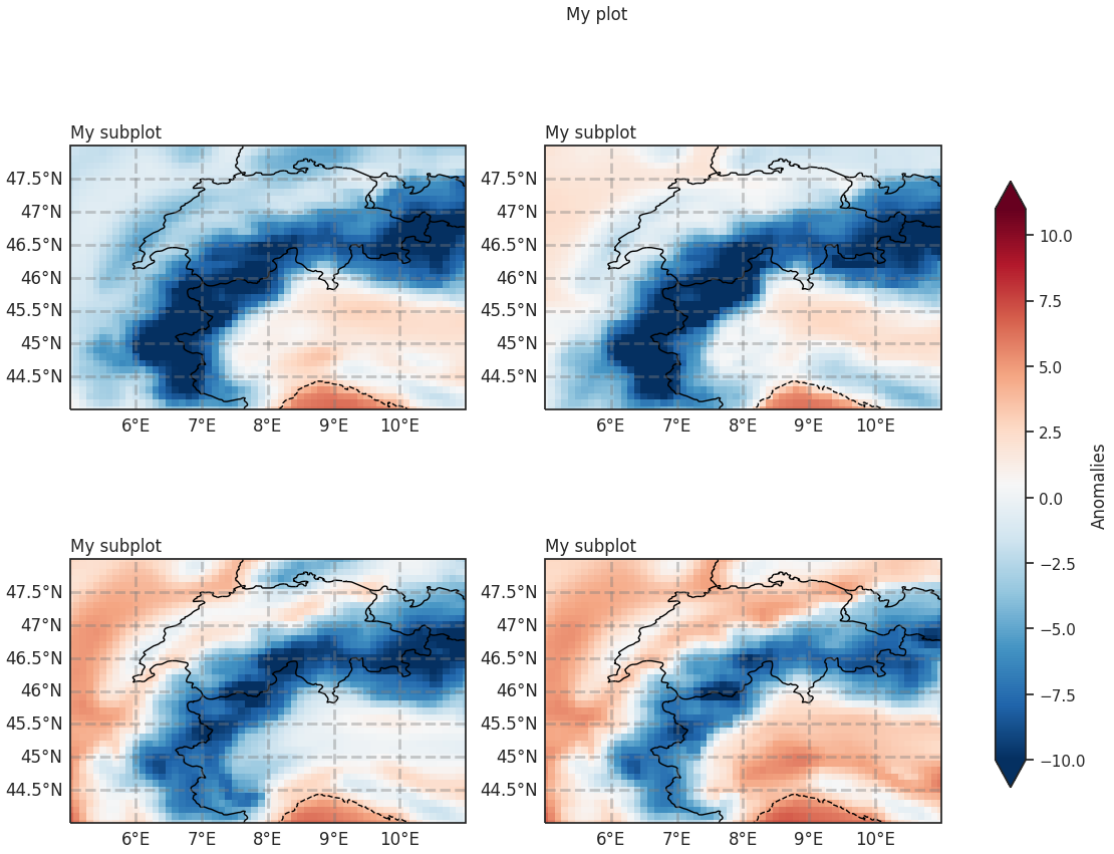
[70]:
```
dict(my_class=1)
```

[70]:
```
{'my_class': 1}
```

[68]:
```
projection=ccrs.PlateCarree()
fig, axes = plt.subplots(nrows=2,ncols=2, figsize=(14,10),  dpi=100,
                                         subplot_kw=dict(projection=ccrs.
 ↪PlateCarree())
                                        )

# loop over figure axis and data items inside the data list
# zip attaches the two lists together
for ax, data in zip(axes.flat, data_list):
    p = (data - mean_temp).plot.pcolormesh(ax=ax, cmap='RdBu_r', vmin=-10,␣
 ↪vmax=11,
                                                              add_colorbar=False)
# Note: vmin, vmax is very important so that colorbar is consistent for all␣
 ↪subplots
# Or you can give custom levels
    gl = ax.gridlines(crs=ccrs.PlateCarree(), draw_labels=True,
```

```
                  linewidth=2, color='gray', alpha=0.4, linestyle='--')
    gl.top_labels= False # suppress gridline labels on the top
    gl.right_labels = False # suppress gridline labels at the right edge
    ax.set_extent([5,11, 44, 48], crs=ccrs.PlateCarree())
    ax.coastlines(linestyles='dashed')
    ax.add_feature(cfeature.BORDERS)
    ax.set_title('') # to suppress xarray's default subtitle
    ax.set_title('My subplot', loc='left')


# if you don't like the default spacing between subplots you can control the␣
 ↪finer settings with plt.subplots_adjust()
# plt.subplots_adjust(wspace = 0.3,  # the amount of width reserved for space␣
 ↪between subplots,
                  # right = 0.8,   # the right side of the subplots of the␣
 ↪figure
              # expressed as a fraction of the average axis width
#                   hspace = 0.0  # the amount of height reserved for space␣
 ↪between subplots,
              # expressed as a fraction of the average axis height
#                  )
plt.colorbar(p, ax=axes.flat, shrink=0.8, extend='both', label="Anomalies",
            orientation='vertical');

plt.suptitle('My plot', fontsize=12)
# uncomment to save
# change file extension to ".pdf" or ".png" as per need
#plt.savefig('/my_dir/save_my_plot.jpg', dpi=300, bbox_inches='tight',
#        pad_inches=0.05)
```

[68]: Text(0.5, 0.98, 'My plot')



Note : Adding common colorbar with this method is more precise in the sense that one creates a specific colorbar axes with height, width and location info to place the colorbar but the above solution we used is more general . In any case, I had to play with figure `size` and cbar `shrink` to get it right.

[ ]:

[71]:
```
ds1.nbytes/10**9
```

[71]:
```
0.018311944
```

[76]:
```
ds1
```

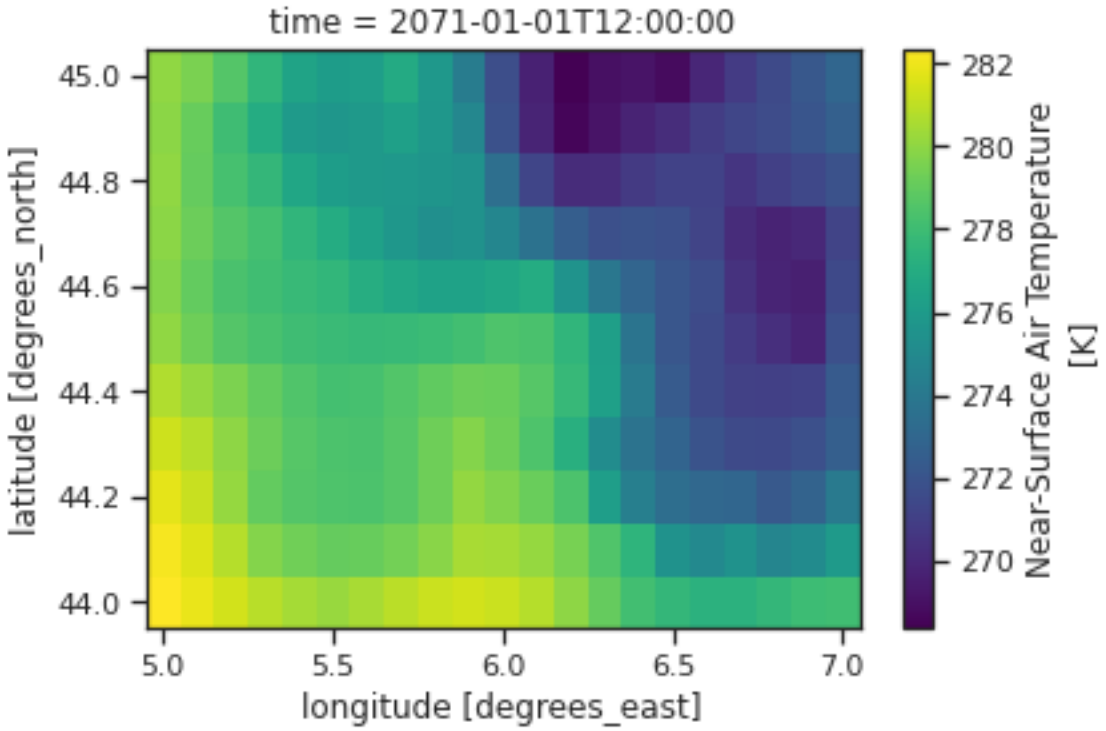
[76]:
```
<xarray.Dataset>
Dimensions:    (time: 1826, bnds: 2, lon: 61, lat: 41)
Coordinates:
  * time       (time) datetime64[ns] 2071-01-01T12:00:00 … 2075-12-31T12:00:00
  * lon        (lon) float64 5.0 5.1 5.2 5.3 5.4 … 10.6 10.7 10.8 10.9 11.0
  * lat        (lat) float64 44.0 44.1 44.2 44.3 44.4 … 47.7 47.8 47.9 48.0
Dimensions without coordinates: bnds
```

```
Data variables:
    time_bnds  (time, bnds) datetime64[ns] 2071-01-01 2071-01-02 … 2076-01-01
    tas        (time, lat, lon) float32 282.4 281.9 281.4 … 284.1 283.6 283.0
Attributes: (12/25)
    CDI:                            Climate Data Interface version ?? (http:/…
    history:                        Fri Mar 13 11:12:41 2020: cdo sellonlatbo…
    institution:                    Swedish Meteorological and Hydrological I…
    Conventions:                    CF-1.4
    contact:                        rossby.cordex@smhi.se
    creation_date:                  2013-07-03-T20:13:36Z
    …                               …
    references:                     http://www.smhi.se/en/Research/Research-d…
    tracking_id:                    18802e76-c2be-49a8-bbe6-c460dcd5960a
    rossby_comment:                 201139: CORDEX Europe 0.11 deg | RCA4 v1 …
    rossby_run_id:                  201139
    rossby_grib_path:               /nobackup/rossby16/rossby/joint_exp/corde…
    CDO:                            Climate Data Operators version 1.9.3 (htt…
```

[77]:
```
my_data=ds1.sel(lat=slice(44, 45), lon=slice(5,7))
```

[79]:
```
my_data.tas.isel(time=0).plot()
```

[79]:
```
<matplotlib.collections.QuadMesh at 0x7f2696c5e790>
```

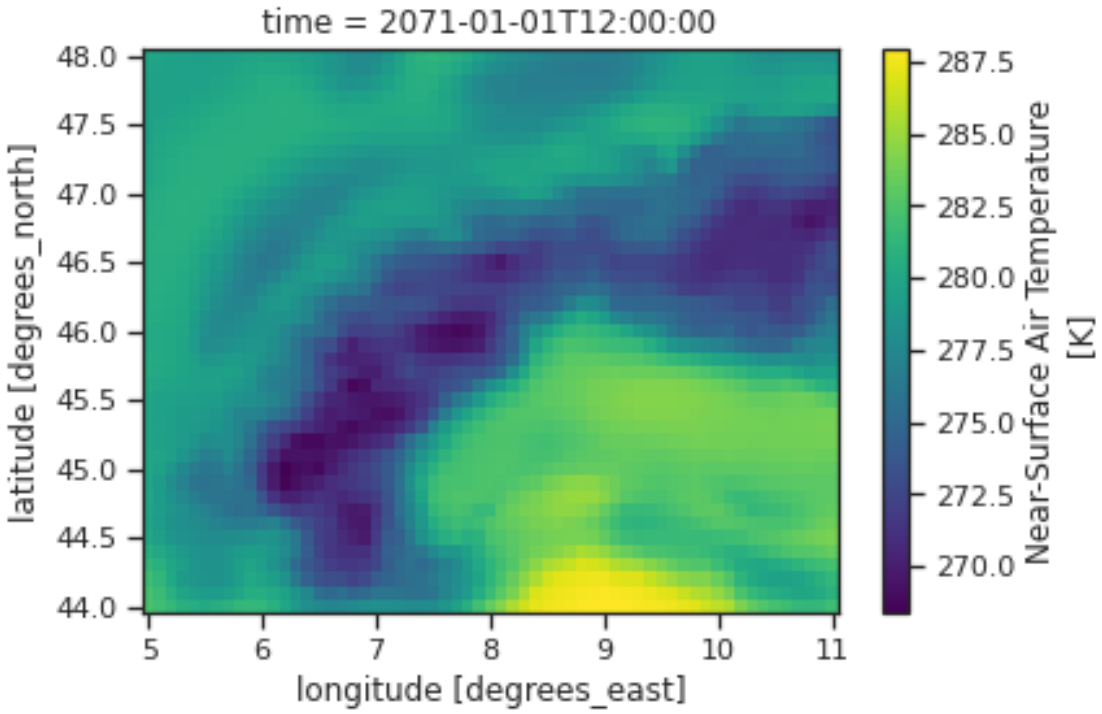


[80]:
```
my_data2 = my_data.copy()
```

[81]:
```
del my_data2
```

[75]:
```
ds1.isel(time=0).tas.plot()
```

[75]:
```
<matplotlib.collections.QuadMesh at 0x7f269651f340>
```



6 Closing and deleting dataset

[ ]:
```
ds1.close()
del ds1
```

It will only delete ds1 variable, it won't delete the original data stroed on disk. Note: Xarray never modifies original data on the disk in all the operations we looked above.

7 VIMP: Please shtudown notebooks properly

`File -> Hub control panel -> Stop my server`

8 References & Further Reading

xarray

Python Data Science Handbook - Visualization