ex_3_Davide

May 28, 2025

[1]:

```python
import pandas as pd
import matplotlib.pyplot as plt
from scipy.interpolate import griddata
import numpy as np
```

[2]:

```python
import pandas as pd

cols = [
    "Stations_id", "von_datum", "bis_datum", "Stationshoehe",
    "geoBreite", "geoLaenge", "Stationsname", "Bundesland"
]

def repair_row(tokens: list[str]) -> list[str] | None:
    """
    tokens is the raw list of strings that made pandas complain.
    Return:
        • list of exactly 8 strings  → pandas keeps the row
        • None                       → pandas skips the row
    """
    if len(tokens) < 8:
        return None

    fixed      = tokens[:6]
    bundesland = tokens[-1]
    name       = " ".join(tokens[6:-1])
    return fixed + [name, bundesland]



df = pd.read_csv(
    "zehn_min_rr_Beschreibung_Stationen.txt",
    sep=r"\s+",      # tab OR  2 spaces  → keeps single blanks inside names
    skiprows=2,
    names=cols,
    engine="python",
    encoding="cp1252",
    keep_default_na=False,
    on_bad_lines=repair_row
)


df["Stations_id"]   = df["Stations_id"].astype(int)
df["von_datum"]     = pd.to_datetime(df["von_datum"], format="%Y%m%d")
df["bis_datum"]     = pd.to_datetime(df["bis_datum"], format="%Y%m%d")
df["Stationshoehe"] = pd.to_numeric(df["Stationshoehe"])
df["geoBreite"]     = pd.to_numeric(df["geoBreite"])
df["geoLaenge"]     = pd.to_numeric(df["geoLaenge"])

len(df)
```

[2]:

```
1067
```

[3]:

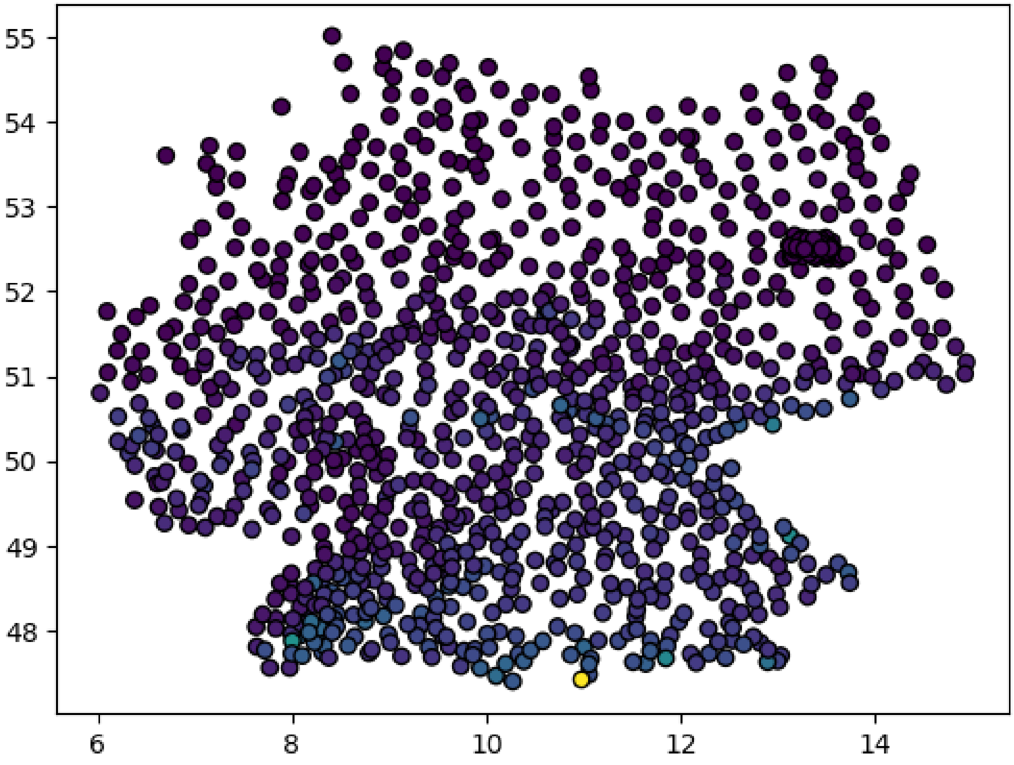
```
main_df = df
main_df
```

[3]:

```
      Stations_id  von_datum  bis_datum  Stationshoehe  geoBreite  geoLaenge  \
0              44 2007-02-08 2024-04-22             44    52.9336     8.2370
1              53 2005-08-31 2024-04-22             60    52.5850    13.5634
2              73 2007-02-13 2024-04-22            374    48.6183    13.0620
3              78 2004-10-10 2024-04-22             64    52.4853     7.9125
4              87 2004-10-19 2024-04-22            158    51.0950    11.0479
…           …        …        …             …        …        …
1062        19172 2020-08-20 2024-04-22             48    54.0246     9.3880
1063        19207 2023-03-30 2024-04-22             16    53.8178    12.0645
1064        19299 2021-03-22 2024-04-22            463    49.8713    11.7883
1065        19897 2023-12-31 2024-04-22             37    52.5040    13.4550
1066        19898 2023-12-31 2024-04-22             39    52.4970    13.2820

                    Stationsname              Bundesland
0                   Großenkneten           Niedersachsen
1                    Ahrensfelde             Brandenburg
2          Aldersbach-Kramersepp                  Bayern
3                      Alfhausen           Niedersachsen
4                     Alperstedt               Thüringen
…                          …                      …
1062                      Wacken      Schleswig-Holstein
1063               Gülzow-Prüzen  Mecklenburg-Vorpommern
1064               Speichersdorf                  Bayern
1065  Berlin-Friedrichshain-Nord                  Berlin
1066             Berlin-Halensee                  Berlin

[1067 rows x 8 columns]
```

[4]:
```
plt.scatter(df['geoLaenge'],df['geoBreite'], c= df["Stationshoehe"], ␣
↪cmap="viridis", edgecolor="k")
```

[4]:
```
<matplotlib.collections.PathCollection at 0x7faccdc88dd0>
```



[ ]:
```
```

[5]:
```
df_10m = pd.read_csv('10min_processed.csv')
df_10m['date'] = pd.to_datetime(df_10m['date'] , format="%Y%m%d%H%M")
df_10m = df_10m[df_10m['rain'] != -999 ]

df_10m
```

[5]:
```
        stationid                 date  rain
0            6303  2024-04-20 00:00:00   0.0
1            6303  2024-04-20 00:10:00   0.0
2            6303  2024-04-20 00:20:00   0.0
3            6303  2024-04-20 00:30:00   0.0
4            6303  2024-04-20 00:40:00   0.0
…              …                    …     …
150986       7429  2024-04-20 23:10:00   0.0
150987       7429  2024-04-20 23:20:00   0.0
```

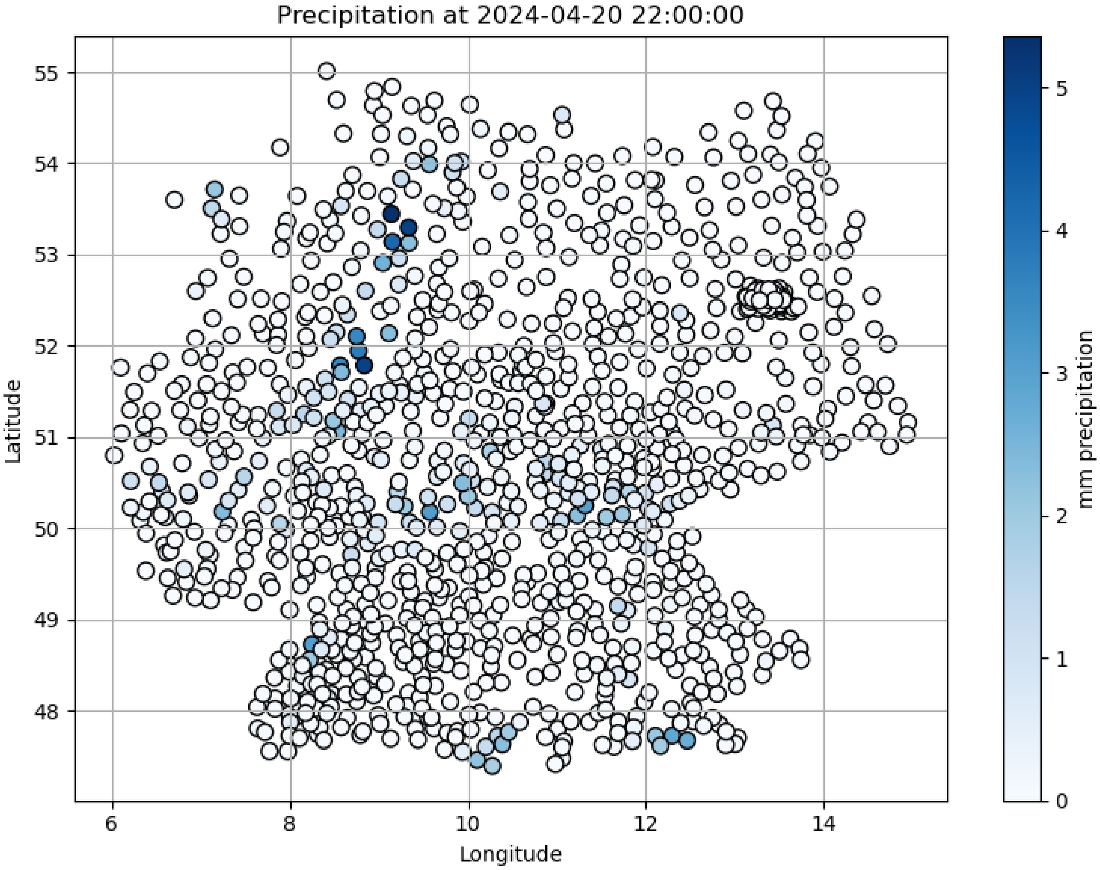
```
150988       7429 2024-04-20 23:30:00   0.0
150989       7429 2024-04-20 23:40:00   0.0
150990       7429 2024-04-20 23:50:00   0.0

[150645 rows x 3 columns]
```

[6]:

```
#Aggregate and plot

df_10m["date"] = pd.to_datetime(df_10m["date"])

df_10m_agg = (
    df_10m
      .set_index("date")
      .groupby("stationid")
      .resample("h")
      ["rain"]
      .sum()
      .reset_index()
)


all_hours    = df_10m_agg["date"].unique()
all_stations = main_df["Stations_id"].unique()
full_index = pd.MultiIndex.from_product(
    [all_hours, all_stations],
    names=["date", "stationid"]
)

hourly = (
    df_10m_agg.set_index(["date", "stationid"])[["rain"]]
        .reindex(full_index, fill_value=0)
        .reset_index()
)


geo_cols = ["Stations_id", "geoBreite", "geoLaenge"]
hourly = hourly.merge(main_df[geo_cols],
                      left_on="stationid",
                      right_on="Stations_id",
                      how="left")
```

[7]:

```
def plot_hour(ts, cmap="Blues"):
    """
    Scatterplot: all stations at the chosen hour.
    Colour = precipitation (mm).
    """
    hour   = pd.to_datetime(ts).floor("h")
    subset = hourly[hourly["date"] == hour]


    fig, ax = plt.subplots(figsize=(8, 6))
    sc = ax.scatter(
        subset["geoLaenge"],
        subset["geoBreite"],
        c=subset["rain"],
        cmap=cmap,
        s=60,
        edgecolor="k"
    )
    cb = fig.colorbar(sc, ax=ax)
    cb.set_label("mm precipitation")
    ax.set_title(f"Precipitation at {hour}")
    ax.set_xlabel("Longitude")
    ax.set_ylabel("Latitude")
    ax.grid(True)
    plt.tight_layout()
    plt.show()
plot_hour("2024-04-20 22:00")
```



[8]:

```
ts = "2024-04-20 22:00"                      # <- change to taste
ts = pd.to_datetime(ts).floor("H")
subset = hourly[hourly["date"] == ts]
subset


loaded  = np.load("griddata.npz")
geolat  = loaded["geolat"]
geolong = loaded["geolong"]
ind     = loaded["ind"]

grid_points = np.column_stack([geolong.ravel(), geolat.ravel()])
src_points = subset[["geoLaenge", "geoBreite"]].to_numpy()         # (M, 2)
src_values = subset["rain"].to_numpy()


interp = griddata(
    src_points, src_values,
    grid_points,
    method="linear"
).reshape(geolat.shape)

interp_masked = np.where(ind, interp, np.nan)
```

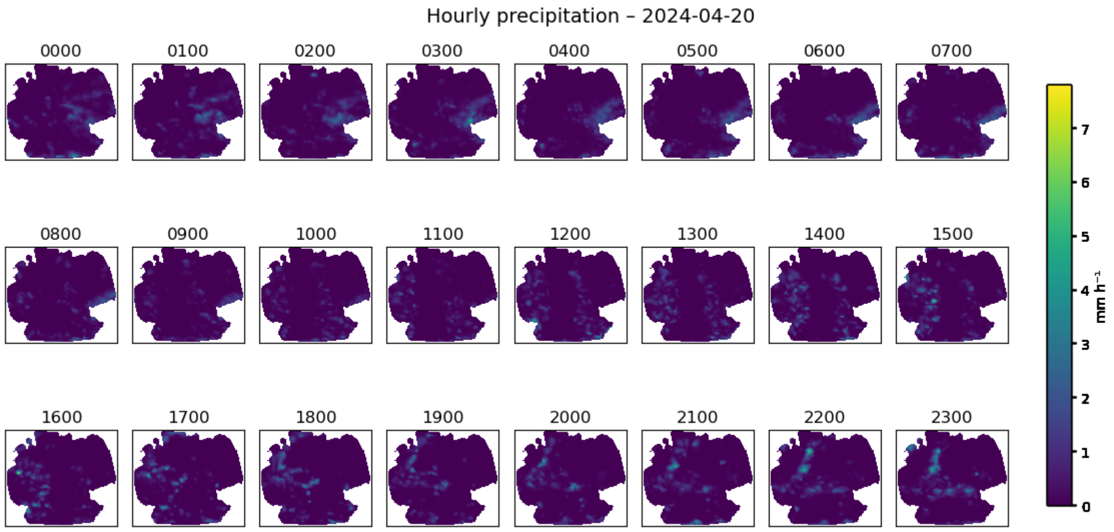
```
/tmp/ipykernel_322/3703775058.py:2: FutureWarning: 'H' is deprecated and will be
removed in a future version, please use 'h' instead.
  ts = pd.to_datetime(ts).floor("H")
```

[9]:

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.interpolate import griddata
import pandas as pd

def daily_precip_mosaic(day, hourly_df, geolong, geolat, ind,
                        method="linear", cmap="viridis",
                        n_cols=8, figsize=(12, 6)):
    """
    Make an 00-23 UTC mosaic of interpolated precipitation maps.

    Parameters
    ----------
    day         : str or datetime-like
        The calendar day to plot, e.g. "2024-04-20".
    hourly_df   : DataFrame
```

```
        Must have columns ["hour", "stationid", "rain", "geoBreite",␣
↪"geoLaenge"].
    geolong,
    geolat      : 2-D arrays (same shape) of target grid coordinates.
    ind         : 2-D boolean array – True where interpolation is valid.
    method      : "linear", "cubic" or "nearest" → scipy.griddata method.
    cmap        : Matplotlib colormap name.
    n_cols      : Panels per row in the mosaic.
    figsize     : Overall figure size.
    """
    day = pd.to_datetime(day).normalize()
    hours = pd.date_range(day, periods=24, freq="H")
    src  = {h: hourly_df[hourly_df["date"] == h] for h in hours}

    vmax = max(s["rain"].max() if not s.empty else 0 for s in src.values())
    vmin = 0

    grid_pts = np.column_stack([geolong.ravel(), geolat.ravel()])

    n_rows = int(np.ceil(24 / n_cols))
    fig, axes = plt.subplots(
        n_rows, n_cols,
        figsize=figsize,
        subplot_kw=dict(aspect="equal", xticks=[], yticks=[])
    )
    axes = axes.ravel()

    #interpolation

    for i, h in enumerate(hours):
        ax  = axes[i]
        sub = src[h]


        vals = griddata(
            sub[["geoLaenge", "geoBreite"]].to_numpy(),
            sub["rain"].to_numpy(),
            grid_pts,
            method=method
        ).reshape(geolat.shape)

        vals = np.where(ind, vals, np.nan)

        im = ax.imshow(
            vals,
            extent=[geolong.min(), geolong.max(),
                    geolat.min(),  geolat.max()],
            origin="lower",
            vmin=vmin, vmax=vmax, cmap=cmap
        )
        fig.subplots_adjust(right=0.86)
        cbar_ax = fig.add_axes([0.88, 0.15, 0.02, 0.7])  # [left, bottom, width, height]
        fig.colorbar(im, cax=cbar_ax, label="mm h ¹")
        ax.set_title(h.strftime("%H%M"))

    for j in range(i + 1, len(axes)):
        fig.delaxes(axes[j])

    plt.tight_layout(rect=[0, 0, 0.86, 1])   # keep the colour-bar area free
    fig.suptitle(f"Hourly precipitation – {day.date()}", y=0.98, fontsize=14)
    plt.show()


loaded = np.load("griddata.npz")
daily_precip_mosaic("2024-04-20", hourly,
                     loaded["geolong"], loaded["geolat"], loaded["ind"])
```

```
/tmp/ipykernel_322/1735644380.py:27: FutureWarning: 'H' is deprecated and will
be removed in a future version, please use 'h' instead.
  hours = pd.date_range(day, periods=24, freq="H")
/tmp/ipykernel_322/1735644380.py:74: UserWarning: This figure includes Axes that
are not compatible with tight_layout, so results might be incorrect.
  plt.tight_layout(rect=[0, 0, 0.86, 1])   # keep the colour-bar area free
```

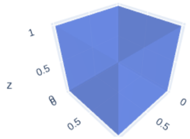


```
[ ]:
```

[10]:

```
import numpy as np
import plotly.graph_objects as go

# vertices
points = np.array(
    [[0,0,0],[0,1,0],[0,1,1],[0,0,1],
     [1,0,0],[1,1,0],[1,1,1],[1,0,1]],
    dtype=float
)

triangles = np.array([
    [0, 1, 5], [0, 5, 4],
    [3, 7, 6], [3, 6, 2],
    [0, 3, 2], [0, 2, 1],
    [4, 5, 6], [4, 6, 7],
    [0, 4, 7], [0, 7, 3],
    [1, 2, 6], [1, 6, 5]
])

fig_cube = go.Figure(
    data=go.Mesh3d(
        x=points[:,0], y=points[:,1], z=points[:,2],
        i=triangles[:,0], j=triangles[:,1], k=triangles[:,2],
        color="royalblue", opacity=0.5, flatshading=True
    )
)
fig_cube.update_layout(title="Unit cube - 12-triangle surface mesh")
fig_cube.show()
```

Unit cube – 12-triangle surface mesh



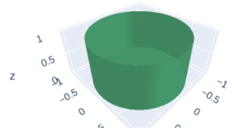
[11]:

```
def disk_mesh(n_r=15, n_th=40):
    """Return (points, triangles) approximating the unit disk z=0."""
    r = np.linspace(0, 1, n_r)                        # radii
    th = np.linspace(0, 2*np.pi, n_th, endpoint=False)  # angles
    R, T = np.meshgrid(r, th, indexing="ij")    # rectangular grid

    # polar → cartesian (z = 0)
    x = (R*np.cos(T)).ravel()
    y = (R*np.sin(T)).ravel()
    z = np.zeros_like(x)

    pts = np.column_stack([x, y, z])

    # connectivity: two triangles per quad in (r, ) grid
    tri_list = []
    for ir in range(n_r-1):
        for it in range(n_th):
            a  = ir*n_th + it
            b  = a + n_th                      # next radius ring
            a1 = ir*n_th + (it+1) % n_th       # wrap angle
            b1 = a1 + n_th
            tri_list += [[a, a1, b1], [a, b1, b]]

    return pts, np.array(tri_list, dtype=int)

disk_pts, disk_tri = disk_mesh()

fig_disk = go.Figure(
    go.Mesh3d(
        x=disk_pts[:,0], y=disk_pts[:,1], z=disk_pts[:,2],
        i=disk_tri[:,0], j=disk_tri[:,1], k=disk_tri[:,2],
        color="tomato", opacity=0.6
    )
).update_layout(title="Triangular mesh of unit disk (z=0)")
fig_disk.show()
```

Triangular mesh of unit disk (z=0)
1
0
z
-1
-0.5
0
-1
-0.5
0

[12]:
```
def cylinder_side(n_z=20, n_th=60):
    z = np.linspace(0, 1, n_z)
    th = np.linspace(0, 2*np.pi, n_th, endpoint=False)
    Z, T = np.meshgrid(z, th, indexing="ij")

    x = np.cos(T).ravel()
    y = np.sin(T).ravel()
    z = Z.ravel()

    pts = np.column_stack([x, y, z])

    tri = []
    for iz in range(n_z-1):
        for it in range(n_th):
            a  = iz*n_th + it
            b  = a + n_th
            a1 = iz*n_th + (it+1) % n_th
            b1 = a1 + n_th
            tri += [[a, a1, b1], [a, b1, b]]

    return pts, np.array(tri, int)

side_pts, side_tri = cylinder_side()
top_pts, top_tri = disk_mesh()
top_pts[:,2] = 1.0                      # lift to z = 1
bot_pts, bot_tri = disk_mesh()          # z=0 already

# Offset triangle indices before merging
bot_tri_off = bot_tri
top_tri_off = top_tri + len(bot_pts)
side_tri_off = side_tri + len(bot_pts) + len(top_pts)

all_pts = np.vstack([bot_pts, top_pts, side_pts])
all_tri = np.vstack([bot_tri_off, top_tri_off, side_tri_off])

fig_cyl = go.Figure(
    go.Mesh3d(
        x=all_pts[:,0], y=all_pts[:,1], z=all_pts[:,2],
        i=all_tri[:,0], j=all_tri[:,1], k=all_tri[:,2],
        color="seagreen", opacity=0.65
    )
).update_layout(title="Closed cylinder (side + top/bottom)")
fig_cyl.show()
```

Closed cylinder (side + top/bottom)



[ ]:

```
```