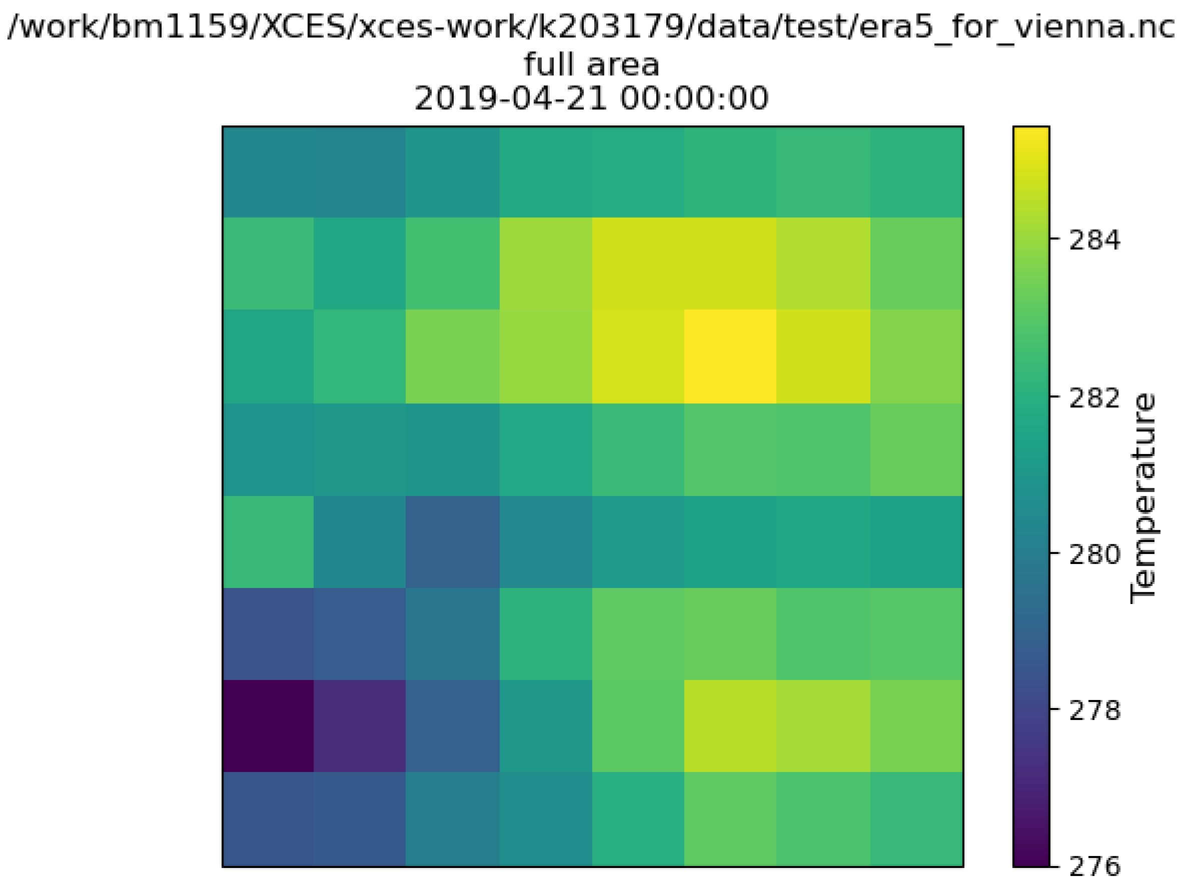
In [ ]:
```python
station_name = "Vienna"
test_year = 2019
```

In [ ]:
```python
from climatereconstructionai import evaluate

evaluate(f"test_args_{station_name.lower()}.txt")
```

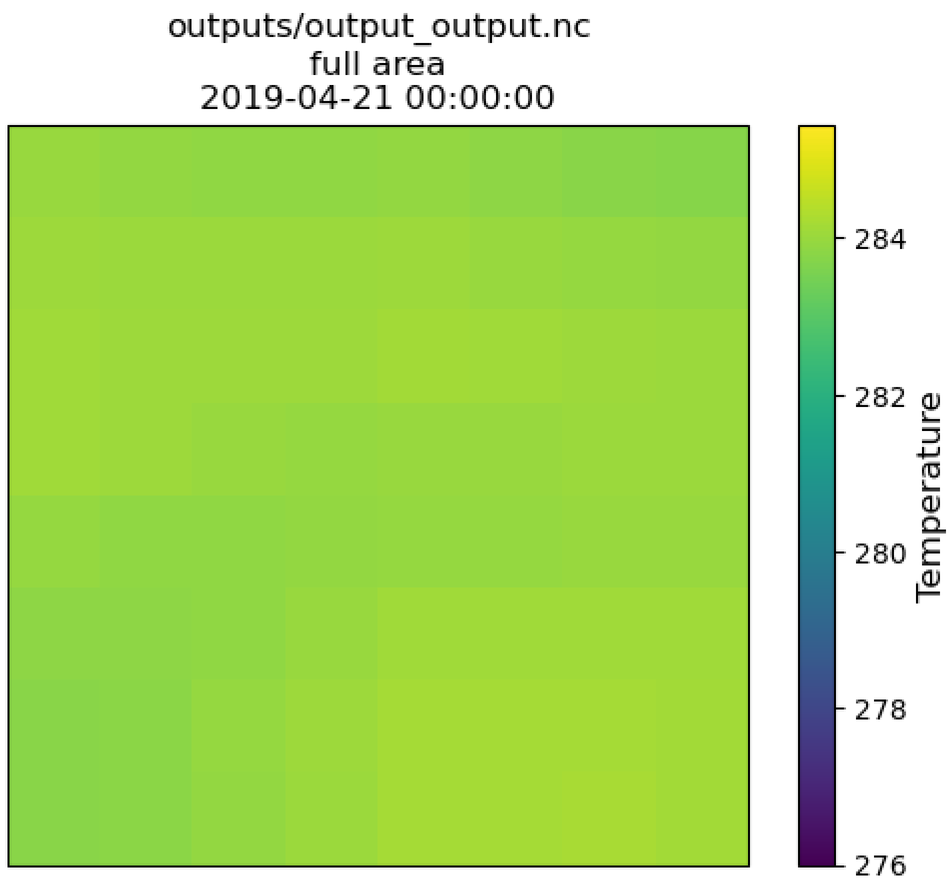
```
/home/k/k203179/.conda/envs/crai/lib/python3.10/site-packages/climatereconstructionai/utils/normalizer.py:10: RuntimeWarning: Mean of empty slice
  img_mean.append(np.nanmean(np.array(img_data[i])))
/home/k/k203179/.conda/envs/crai/lib/python3.10/site-packages/numpy/lib/nanfunctions.py:1879: RuntimeWarning: Degrees of freedom <= 0 for slice.
  var = nanvar(a, axis=axis, dtype=dtype, out=out, ddof=ddof,
100%|██████████| 1/1 [00:01<00:00,  1.31s/it]
```

In [ ]:
```python
import xarray as xr
from utils import DataSet, DatasetPlotter
import numpy as np
import os

test_folder_path = "/work/bm1159/XCES/xces-work/k203179/data/test"
reconstructed_folder_path = "outputs/output_output.nc"
era5_file = f"{test_folder_path}/era5_for_{station_name.lower()}.nc"

# get measurements values

measurements_data = xr.open_dataset(test_folder_path + f"/reality_{station_name.lower()}.nc")

# plot era5 and output at timesteps [x, ...]
plot_timestep = 2000

era5_ds = DataSet(era5_file)
output_ds = DataSet(reconstructed_folder_path)

vmin = min(
    np.min(era5_ds.dataset.variables['tas'][plot_timestep, :, :]),
    np.min(output_ds.dataset.variables['tas'][plot_timestep, :, :]),
)

vmax = max(
    np.max(era5_ds.dataset.variables['tas'][plot_timestep, :, :]),
    np.max(output_ds.dataset.variables['tas'][plot_timestep, :, :]),
)
```

In [ ]:
```python
plotter = DatasetPlotter(era5_ds)
plotter.time_index_list = [plot_timestep]
plotter.vmin = vmin
plotter.vmax = vmax
plotter.plot()
```



In [ ]:
```python
plotter = DatasetPlotter(output_ds)
plotter.time_index_list = [plot_timestep]
plotter.vmin = vmin
plotter.vmax = vmax
plotter.plot()
```




```
In [ ]: # get coordinates from measurements nc file
import numpy as np

station_lon, station_lat = measurements_data.lon.values[0], measurements_data.lat.values[0]
print(f"station is at {station_lon}, {station_lat}")

# get nearest coordinates in era5
def get_left_right_nearest_elem_in_sorted_array(array, value):
    length = len(array)
    left = len(list(filter(lambda x: x <= value, array))) - 1
    right = length - len(list(filter(lambda x: x >= value, array)))
    nearest = min(left, right, key=lambda x: abs(array[x] - value))
    return left, right, nearest

test_array = [1, 2, 3, 4, 5, 6, 7, 8]
test_search = 5.51

print(f"searching for {test_search} in {test_array}")
left_idx, right_idx, nearest_idx =  get_left_right_nearest_elem_in_sorted_array(test_array, test_search)
print(f"idx left to {test_search} is {left_idx}, idx right to {test_search} is {right_idx}, nearest idx is {nearest
print(f"mid crop: {test_array[left_idx:right_idx+1]}")
```

```
station is at 16.3609, 48.2303
searching for 5.51 in [1, 2, 3, 4, 5, 6, 7, 8]
idx left to 5.51 is 4, idx right to 5.51 is 5, nearest idx is 5
mid crop: [5, 6]
```

```
In [ ]: def era_vs_reconstructed_comparision_to_df():
    era5_data = xr.open_dataset(era5_file)
    reconstructed_data = xr.open_dataset(reconstructed_folder_path)


    lon_left_idx, lon_right_idx, lon_nearest_idx = get_left_right_nearest_elem_in_sorted_array(era5_data.lon.values
    lat_left_idx, lat_right_idx, lat_nearest_idx = get_left_right_nearest_elem_in_sorted_array(era5_data.lat.values

    era5_mid_values = era5_data.variables["tas"][:, lon_left_idx:lon_right_idx+1, lat_left_idx:lat_right_idx+1].mea
    era5_nearest_values = era5_data.variables["tas"][:, lon_nearest_idx, lat_nearest_idx]

    reconstructed_data_values = reconstructed_data.variables["tas"].stack(grid=['lat', 'lon']).values
    measurements_data_values = measurements_data.variables["tas"][...].mean(axis=(1,2))


    # timeaxis
    time = measurements_data.variables["time"][:]

    import pandas as pd

    # create dataframe with all values
    df = pd.DataFrame()

    df["time"] = time

    # index should be time
    df.set_index("time", inplace=True)

    df["era5_mid"] = era5_mid_values
    df["era5_nearest"] = era5_nearest_values
    df["reconstructed_median"] = [np.median(x) for x in reconstructed_data_values]
    df["reconstructed_mean"] = [np.mean(x) for x in reconstructed_data_values]
    df["reconstructed_min"] = [np.min(x) for x in reconstructed_data_values]
```

```python
    df["reconstructed_max"] = [np.max(x) for x in reconstructed_data_values]

    df["measurements"] = measurements_data_values

    return df
```

Generate Dataframe

- makes resampling easier

In [ ]:
```python
hourly_df = era_vs_reconstructed_comparision_to_df()

# print a section of the df

start_print_date = "2019-04-21"
end_print_date = "2019-04-21"

hourly_df[start_print_date:end_print_date]
```

Out[ ]:

time	era5_mid	era5_nearest	reconstructed_median	reconstructed_mean	reconstructed_min	reconstructed_max	measur
2019-04-21 00:00:00	281.406372	280.432373	284.001465	283.990601	283.727692	284.201416	283
2019-04-21 01:00:00	280.800140	279.936676	283.812561	283.776123	283.431946	283.993988	282.
2019-04-21 02:00:00	280.060394	280.054199	282.701294	282.693054	282.369843	282.944214	281.
2019-04-21 03:00:00	279.517120	279.784851	281.980469	281.971832	281.689941	282.160065	280.
2019-04-21 04:00:00	279.187622	279.550659	281.416077	281.388916	281.051300	281.546021	281.
2019-04-21 05:00:00	280.308258	280.445496	283.613403	283.639099	283.515961	283.837982	283.
2019-04-21 06:00:00	283.347443	283.139465	288.494354	288.521118	288.371063	288.712097	286.
2019-04-21 07:00:00	286.524933	286.101959	291.023682	291.021179	290.863251	291.219788	288.
2019-04-21 18:00:00	290.378723	291.001373	289.913208	289.853516	289.448486	290.241760	288.
2019-04-21 19:00:00	286.881592	288.088379	289.000793	289.017365	288.728027	289.315979	286.
2019-04-21 20:00:00	285.289551	287.079651	287.705994	287.718567	287.312225	288.023468	286.
2019-04-21 21:00:00	283.730835	285.270081	285.158081	285.165588	284.867493	285.605743	285.
2019-04-21 22:00:00	282.893738	284.848328	285.697388	285.711426	285.477386	285.949066	284.
2019-04-21 23:00:00	281.976074	283.376648	284.158142	284.178528	283.864807	284.515320	282.

Implement plotting method of dataframe

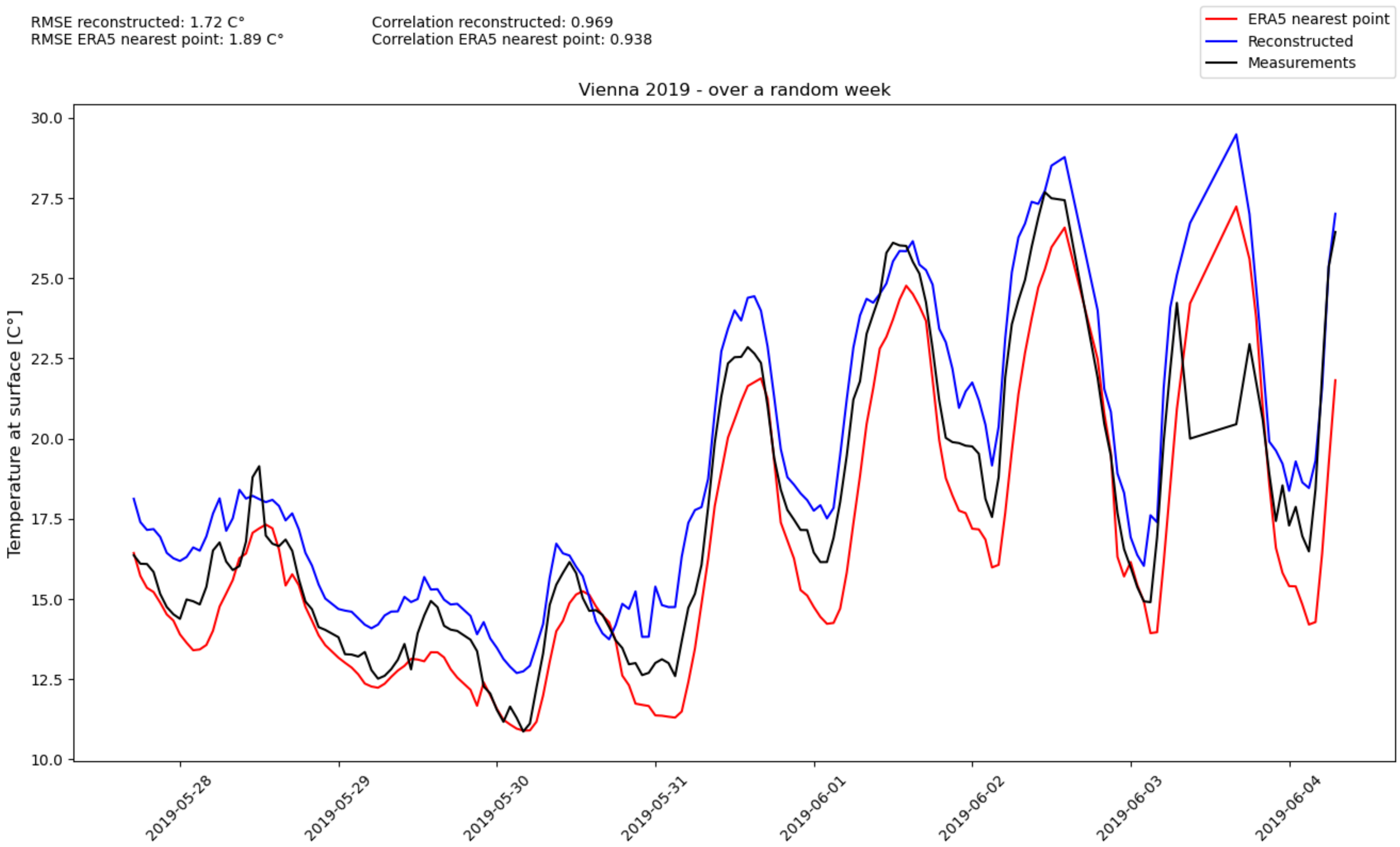
In [ ]:
```python
def plot_n_steps_of_df(df, as_delta, n=None, title=None, boxplot=False):

    from matplotlib import pyplot as plt

    time = df.index.values
```

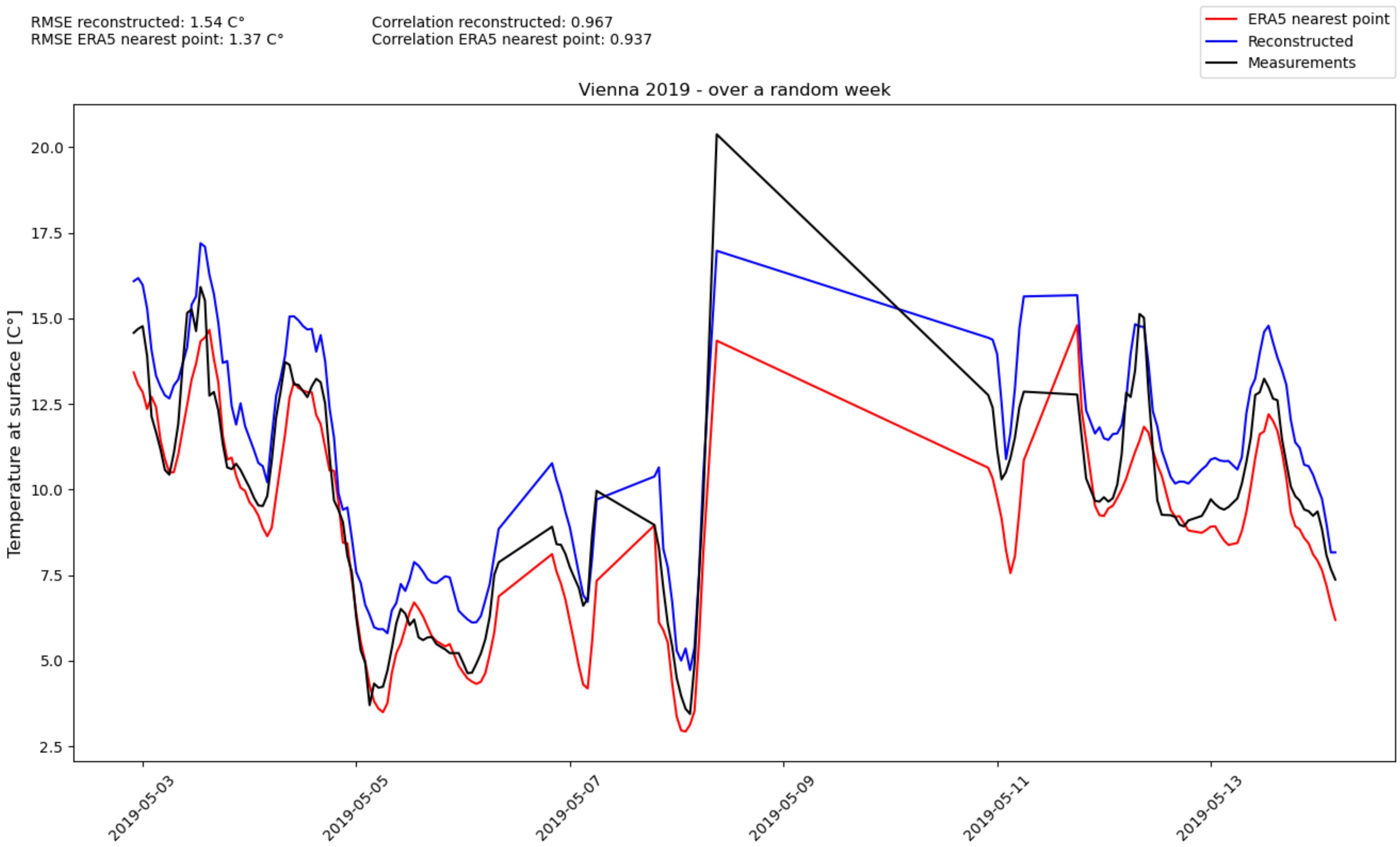
```python
    if n is None:
        n = len(df)

    # random slice of n consecutive datapoints
    import random
    slice_start = random.randint(0, len(time) - n)
    time_slice = slice(slice_start, slice_start + n)

    time = time[time_slice]

   # era5_mid_values = df["era5_mid"].values -273.15
    era5_nearest_values = df["era5_nearest"].values - 273.15
    reconstructed_mean_values = df["reconstructed_mean"].values - 273.15
    reconstructed_median_values = df["reconstructed_median"].values - 273.15
    reconstructed_min_values = df["reconstructed_min"].values - 273.15
    reconstructed_max_values = df["reconstructed_max"].values - 273.15

    measurements_values = df["measurements"].values - 273.15

    rmse_reconstructed = np.sqrt(np.sum((reconstructed_median_values[time_slice] - measurements_values[time_slice])
   # rmse_era5_mid = np.sqrt(np.sum((era5_mid_values[time_slice] - measurements_values[time_slice])**2) / len(time)
    rmse_era5_nearest = np.sqrt(np.sum((era5_nearest_values[time_slice] - measurements_values[time_slice])**2) / le

    correlation_reconstructed = np.corrcoef(reconstructed_median_values[time_slice], measurements_values[time_slice
   # correlation_era5_mid = np.corrcoef(era5_mid_values[time_slice], measurements_values[time_slice])[0,1]
    correlation_era5_nearest = np.corrcoef(era5_nearest_values[time_slice], measurements_values[time_slice])[0,1]

    if as_delta:
     #   era5_mid_values = era5_mid_values - measurements_values
        era5_nearest_values = era5_nearest_values - measurements_values
        reconstructed_mean_values = reconstructed_mean_values - measurements_values
        reconstructed_median_values = reconstructed_median_values - measurements_values
        reconstructed_min_values = reconstructed_min_values - measurements_values
        reconstructed_max_values = reconstructed_max_values - measurements_values
        measurements_values = measurements_values - measurements_values

        # y-axis title, temperature difference
        plt.ylabel("Delta calculated by subtracting measurement data [C°]")

    else:
        plt.ylabel("Temperature at surface [C°]")


    plt.plot(time, era5_nearest_values[time_slice], label="ERA5 nearest point", color="red")
    # plt.plot(time, era5_mid_values[time_slice], label="ERA5 nearest 4 points")

    if boxplot:
        for i in range(len(time)):
            plt.vlines(time[i], reconstructed_min_values[time_slice][i], reconstructed_max_values[time_slice][i], c
        plt.scatter(time, reconstructed_median_values[time_slice], label="Reconstructed", color="blue", s=8)
    else:
        plt.plot(time, reconstructed_median_values[time_slice], label="Reconstructed", color="blue")

    plt.plot(time, measurements_values[time_slice], label="Measurements", color="black")

    # x-axis labels 90 degrees
    plt.xticks(rotation=45)

    # title
    if title is not None:
        plt.title(title)


    # font size of legend
    plt.rcParams.update({'font.size': 10})

    # font size of axis labels
    plt.rcParams.update({'axes.labelsize': 12})

    plt.legend()
    # position legend below chart to the right
    plt.legend(bbox_to_anchor=(1, 1.15), loc='upper right', borderaxespad=0.)


    # text below diagram with RMSE and Correlation in fontsize 10
    plt.text(0.1,0.95, f"RMSE reconstructed: {rmse_reconstructed:.2f} C°\n" +
             f"RMSE ERA5 nearest point: {rmse_era5_nearest:.2f} C°",

             fontsize=10, transform=plt.gcf().transFigure)

    plt.text(0.3, 0.95, f"Correlation reconstructed: {correlation_reconstructed:.3f}\n" +
             f"Correlation ERA5 nearest point: {correlation_era5_nearest:.3f}",

             fontsize = 10, transform=plt.gcf().transFigure)

    # figure size A4 landscape
    plt.gcf().set_size_inches(16, 8)
```

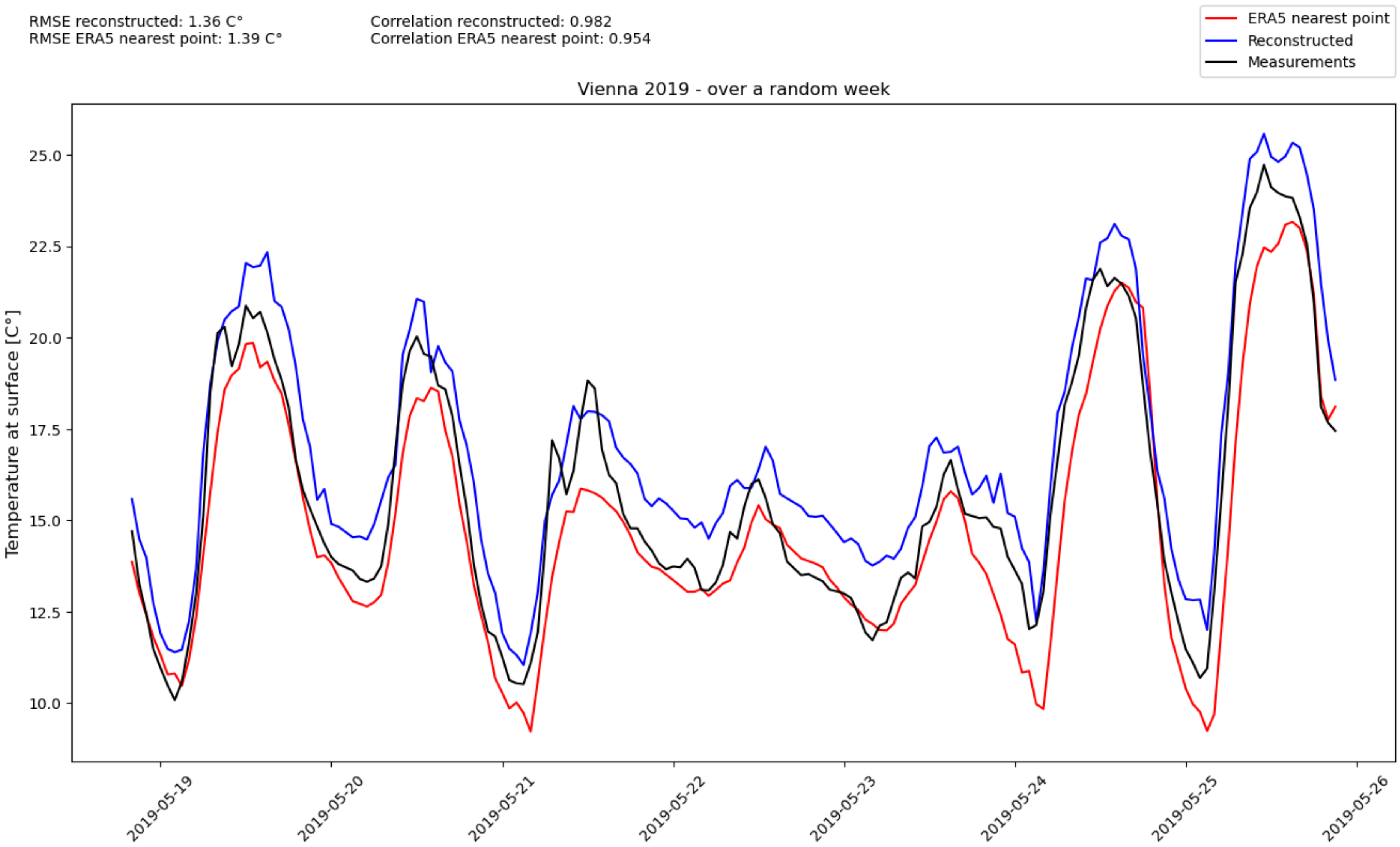
```
    plt.show()
```

Plot Hourly (deltas), so errors against real measurements

In [ ]:
```
n = 168
if n == 168:
    title = f"{'Denver' if station_name == 'Marshall' else station_name} {test_year} - over a random week"
else:
    title = f"{'Denver' if station_name == 'Marshall' else station_name} {test_year} - {n} random consecutive hourl
plot_n_steps_of_df(hourly_df, as_delta=False, n=n, title=title)
plot_n_steps_of_df(hourly_df, as_delta=False, n=n, title=title)
plot_n_steps_of_df(hourly_df, as_delta=False, n=n, title=title)
```



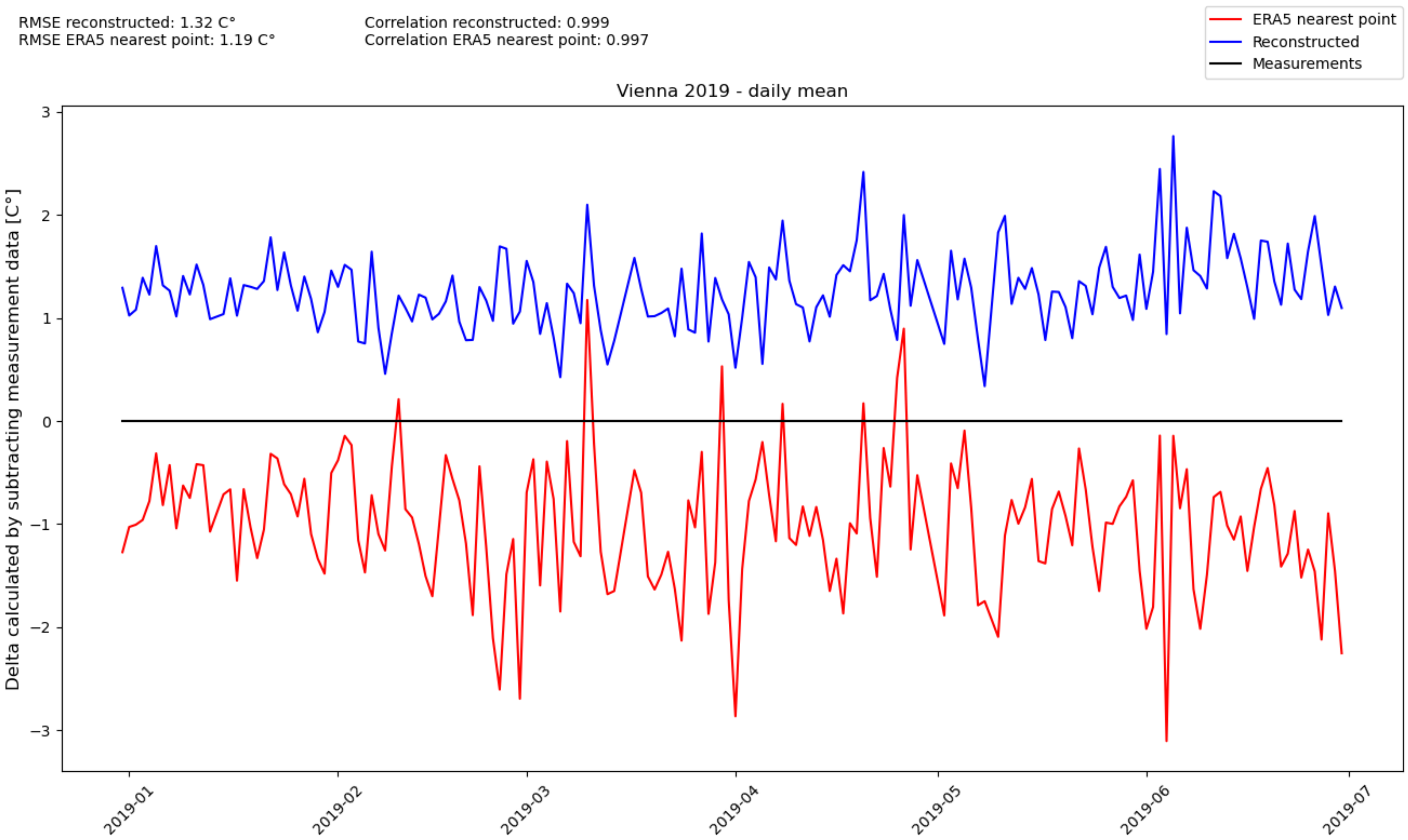


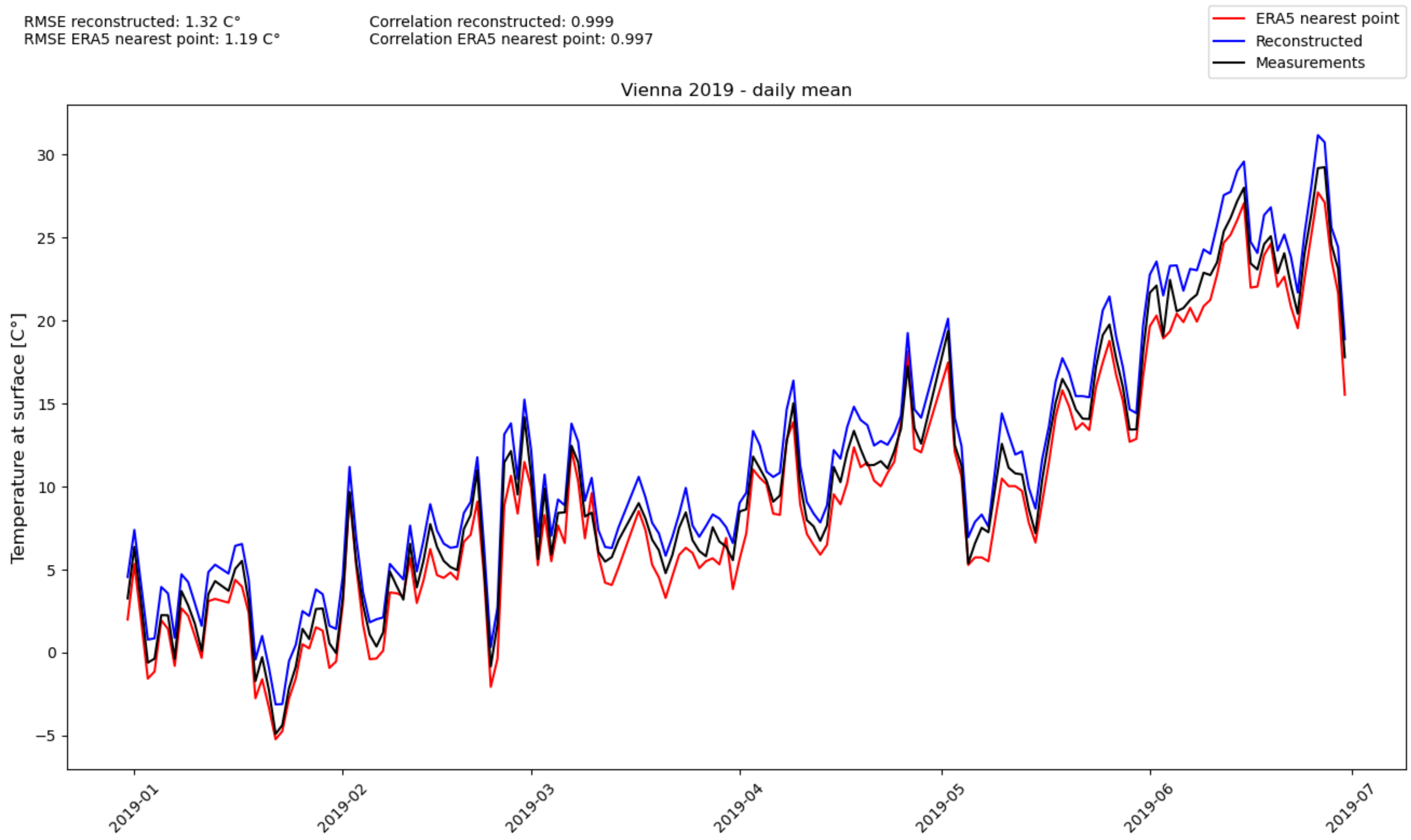


Resample Data from hourly to daily or monthly

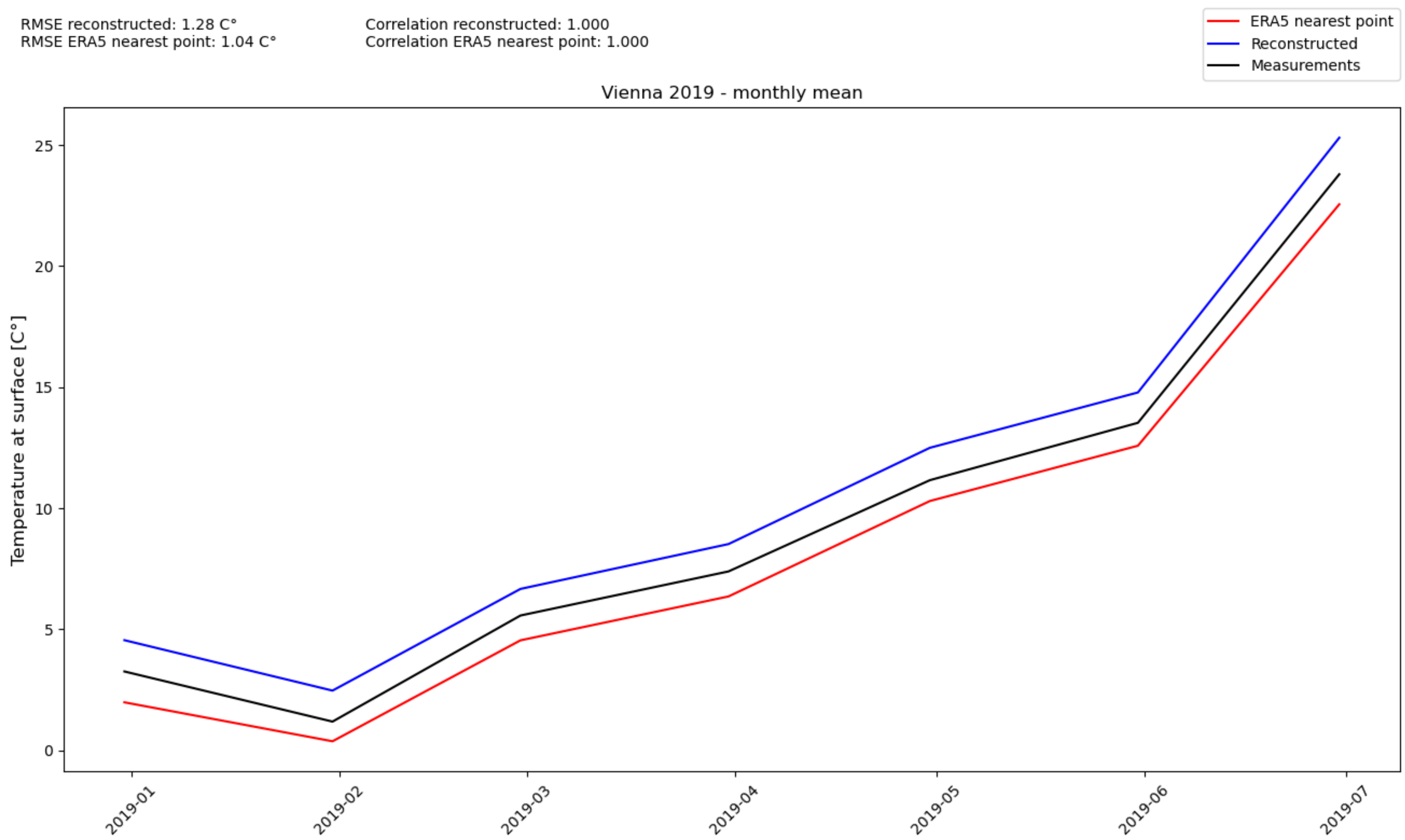
In [ ]:

```
# drop reconstructed column
daily_df = hourly_df.resample("D").mean()
# drop nans
daily_df = daily_df.dropna()
title = f"{'Denver' if station_name == 'Marshall' else station_name} {test_year} - daily mean"
plot_n_steps_of_df(daily_df, as_delta=True, title=title)
plot_n_steps_of_df(daily_df, as_delta=False, title=title)
```



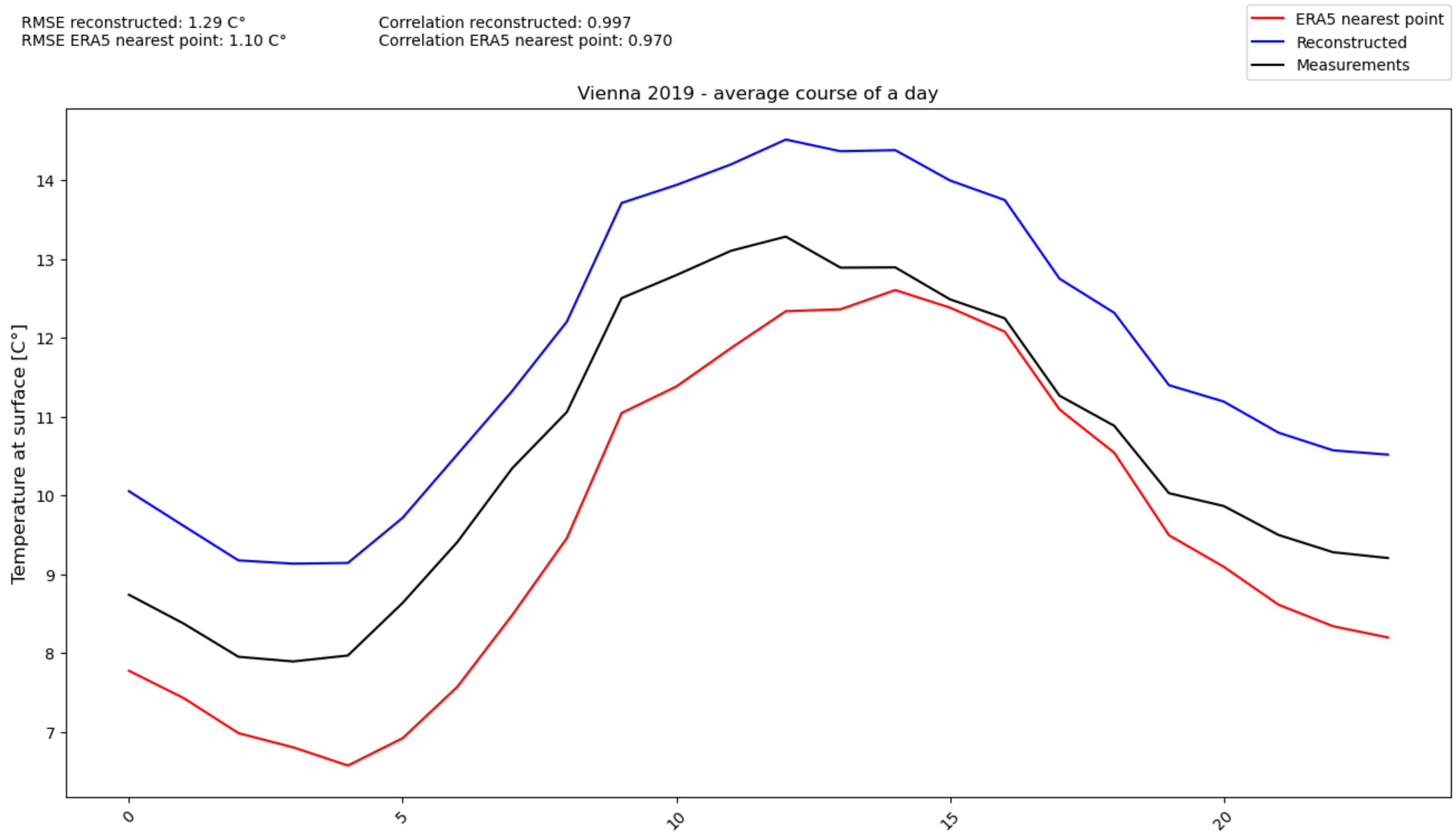


In [ ]:
```python
# resample rows to monthly mean
df = hourly_df.resample("M").mean()
title = f"{'Denver' if station_name == 'Marshall' else station_name} {test_year} - monthly mean"
plot_n_steps_of_df(df, as_delta = False, title=title)
```



Average Course of the day

In [ ]:
```python
# calculate the mean of each 24 hours over the whole year

day_course_df = hourly_df.groupby(hourly_df.index.hour).mean()
title = f"{'Denver' if station_name == 'Marshall' else station_name} {test_year} - average course of a day"
plot_n_steps_of_df(day_course_df, as_delta = False, title=title)
```



Average Course of the month

In [ ]:
```python
# calculate the mean of each day of a month over the whole year

month_course_df = hourly_df.groupby(hourly_df.index.day).mean()
title = f"{'Denver' if station_name == 'Marshall' else station_name} {test_year} - average course of a month"
plot_n_steps_of_df(month_course_df, as_delta = False, title=title)
```