In [ ]:

```
station_name = "Marshall"
test_year = 2021
```

In [ ]:

```
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 [00:08<00:00,  8.29s/it]
```

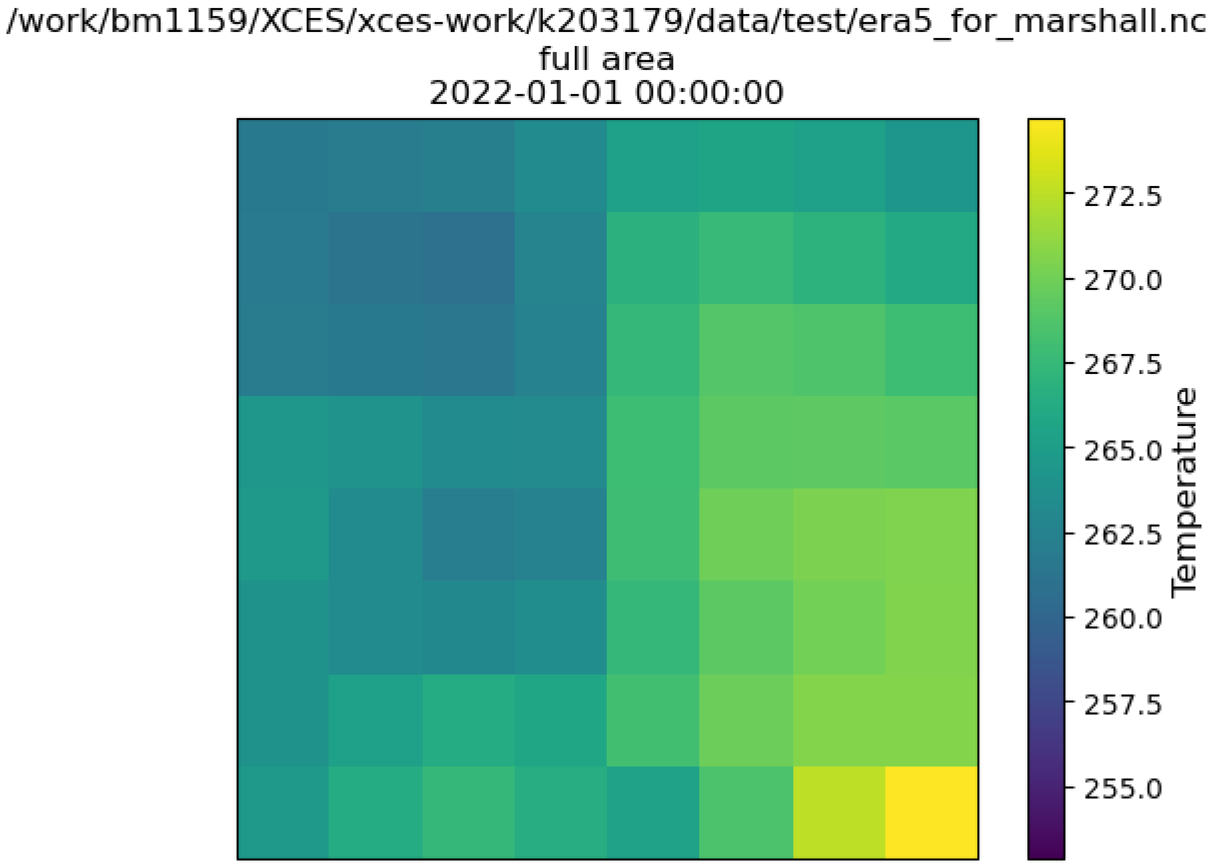
In [ ]:

```
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, ...]

era5_ds = DataSet(era5_file)
output_ds = DataSet(reconstructed_folder_path)
year_ds = DataSet(f"{test_folder_path}/year_at_{station_name.lower()}.nc")
intra_year_ds = DataSet(f"{test_folder_path}/intra_year_at_{station_name.lower()}.nc")
intra_day_ds = DataSet(f"{test_folder_path}/intra_day_at_{station_name.lower()}.nc")

print(era5_ds.time[-1], output_ds.time[-1])

plot_timestep = len(era5_ds.time) - 1

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, :, :]),
)
```

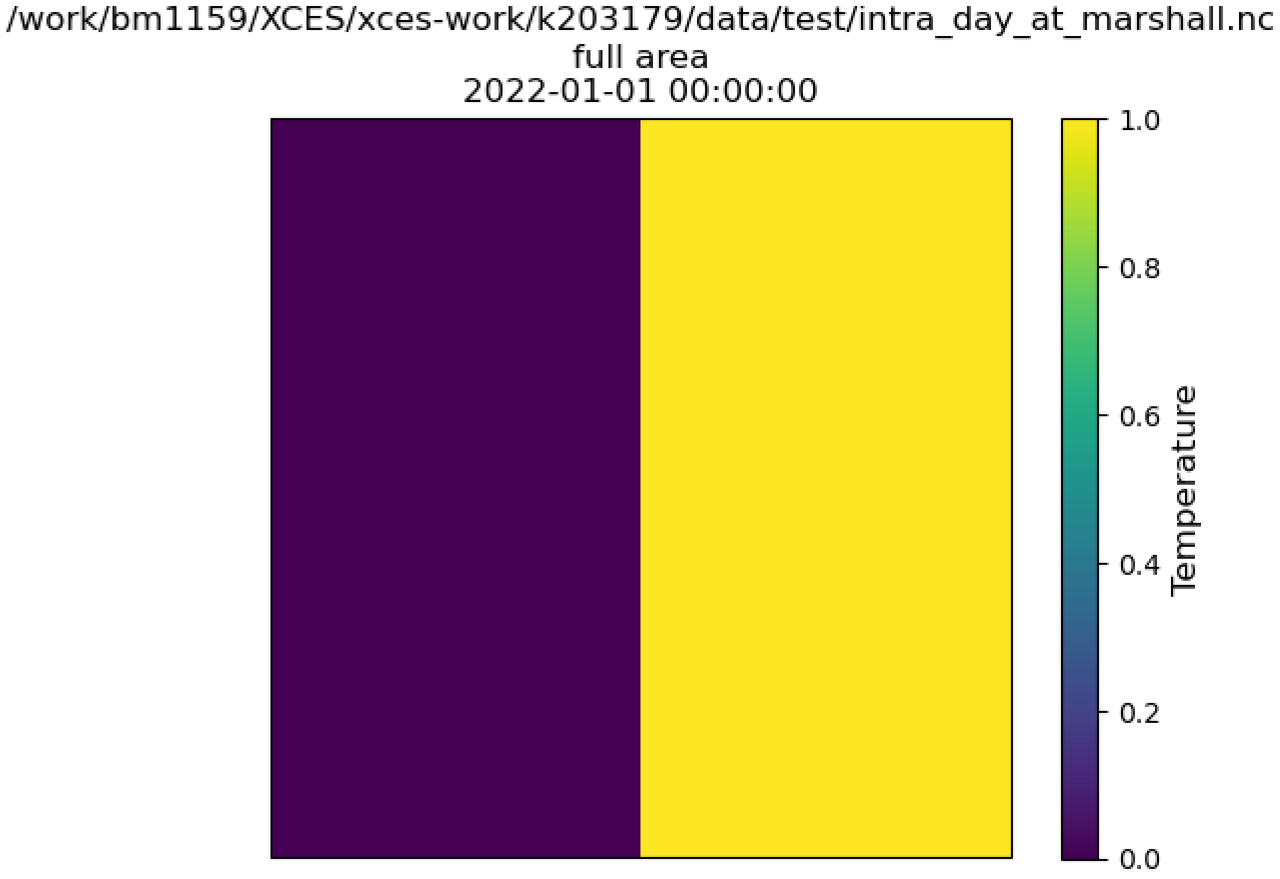
```
8760.0 8760.0
```

In [ ]:

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

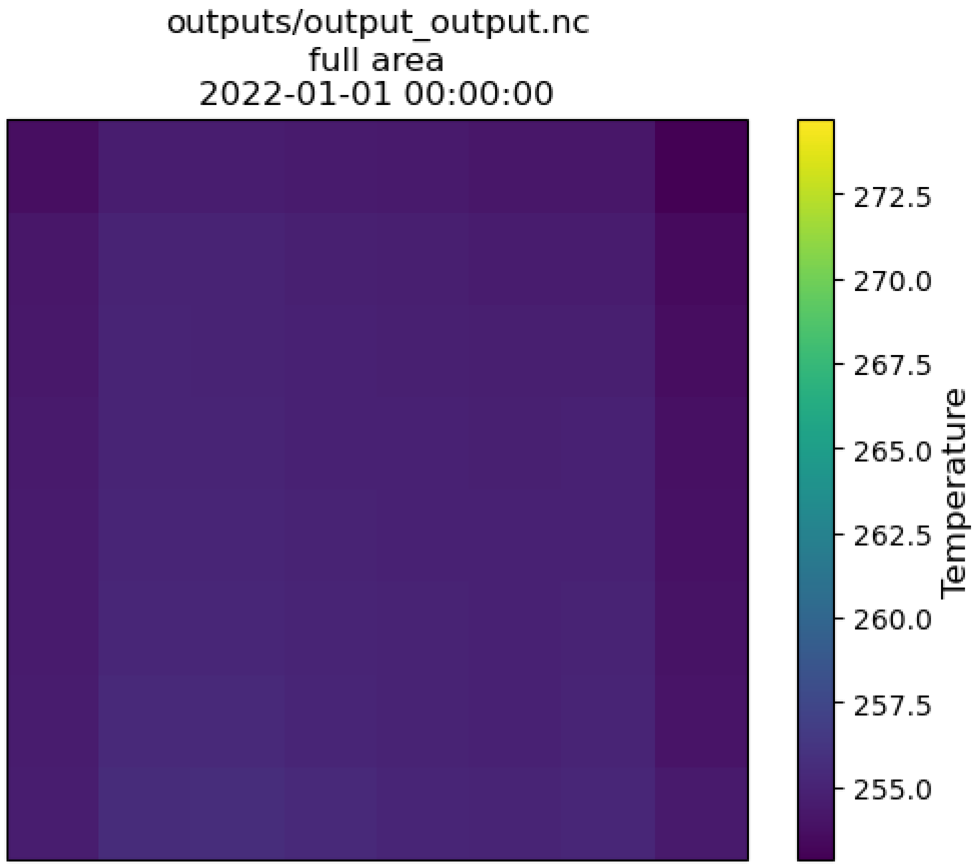


```
In [ ]: plotter = DatasetPlotter(intra_day_ds)
        plotter.time_index_list = [plot_timestep]
        plotter.plot(var="intra_day")
```



In [ ]:

```
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_idx}")
print(f"mid crop: {test_array[left_idx:right_idx+1]}")
```

```
station is at -105.196, 39.9496
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, station_lon % 360)
    lat_left_idx, lat_right_idx, lat_nearest_idx = get_left_right_nearest_elem_in_sorted_array(era5_data.lat.values, station_lat)

    era5_mid_values = era5_data.variables["tas"][:, lon_left_idx:lon_right_idx+1, lat_left_idx:lat_right_idx+1].mean(axis=(1,2))
    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]
    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 [ ]:

```
hourly_df = era_vs_reconstructed_comparision_to_df()

# drop last 1 hour
hourly_df = hourly_df[:-1]

# print a section of the df

start_print_date = "2021-12-31"
end_print_date = "2022-12-31"

hourly_df[start_print_date:end_print_date]
```

Out[ ]:

	era5_mid	era5_nearest	reconstructed_median	reconstructed_mean	reconstructed_min	reconstructed_max	measurements
time							
2021-12-31 00:00:00	275.129761	278.701538	277.828796	277.823212	277.628265	278.027802	277.210000
2021-12-31 01:00:00	275.152008	278.443237	277.659912	277.637207	277.406097	277.850769	277.808333
2021-12-31 02:00:00	274.516663	277.574158	276.237122	276.217896	275.878113	276.529633	278.211017
2021-12-31 03:00:00	274.136444	277.030609	275.665588	275.648621	275.293121	275.975098	278.283333
2021-12-31 04:00:00	274.008026	276.625641	275.115234	275.130188	274.736847	275.544220	278.460000
2021-12-31 05:00:00	272.073303	273.666718	274.044678	274.069946	273.639893	274.553345	278.793333
2021-12-31 06:00:00	270.946594	272.025665	272.802612	272.823822	272.379242	273.343262	278.346667
2021-12-31 07:00:00	270.959534	272.586304	273.480225	273.494080	273.127289	273.988098	276.435000
2021-12-31 08:00:00	270.548676	272.302032	274.385681	274.414154	274.122406	274.855255	276.745000
2021-12-31 09:00:00	269.417847	271.156952	274.199860	274.199158	273.902100	274.603210	275.803333
2021-12-31 10:00:00	269.064728	270.792542	272.341309	272.368591	272.148041	272.689972	275.028333
2021-12-31 11:00:00	269.157471	270.880646	272.796082	272.799744	272.596924	273.114288	276.333333
2021-12-31 12:00:00	269.396210	271.105133	274.779297	274.782715	274.580444	275.123138	274.403333
2021-12-31 13:00:00	269.160004	270.799286	275.311340	275.315796	275.101227	275.686432	270.200000
2021-12-31 14:00:00	268.438232	270.200378	273.509277	273.524414	273.358887	273.843231	270.596667
2021-12-31 15:00:00	268.541504	270.555420	272.140472	272.151337	271.957916	272.504700	271.075000
2021-12-31 16:00:00	269.488220	271.382416	272.197632	272.173706	271.926544	272.483429	271.183333
2021-12-31 17:00:00	270.497253	272.423828	273.830322	273.817322	273.646942	274.031921	271.018333
2021-12-31 18:00:00	270.909241	273.161255	274.540588	274.543396	274.374390	274.731445	271.145000
2021-12-31 19:00:00	270.485809	272.459076	274.277771	274.289612	274.073547	274.490814	271.536667
2021-12-31 20:00:00	269.780762	271.580505	273.365417	273.357788	273.115326	273.638916	271.795000
2021-12-31 21:00:00	269.378296	271.244659	272.192749	272.191956	271.909058	272.560181	270.550000
2021-12-31 22:00:00	266.196259	268.393768	268.680908	268.734985	268.404175	269.313141	269.468333
2021-12-31 23:00:00	266.264435	268.587921	268.030548	268.047974	267.723236	268.546448	268.941667

In [ ]:

Implement plotting method of dataframe

In [ ]:

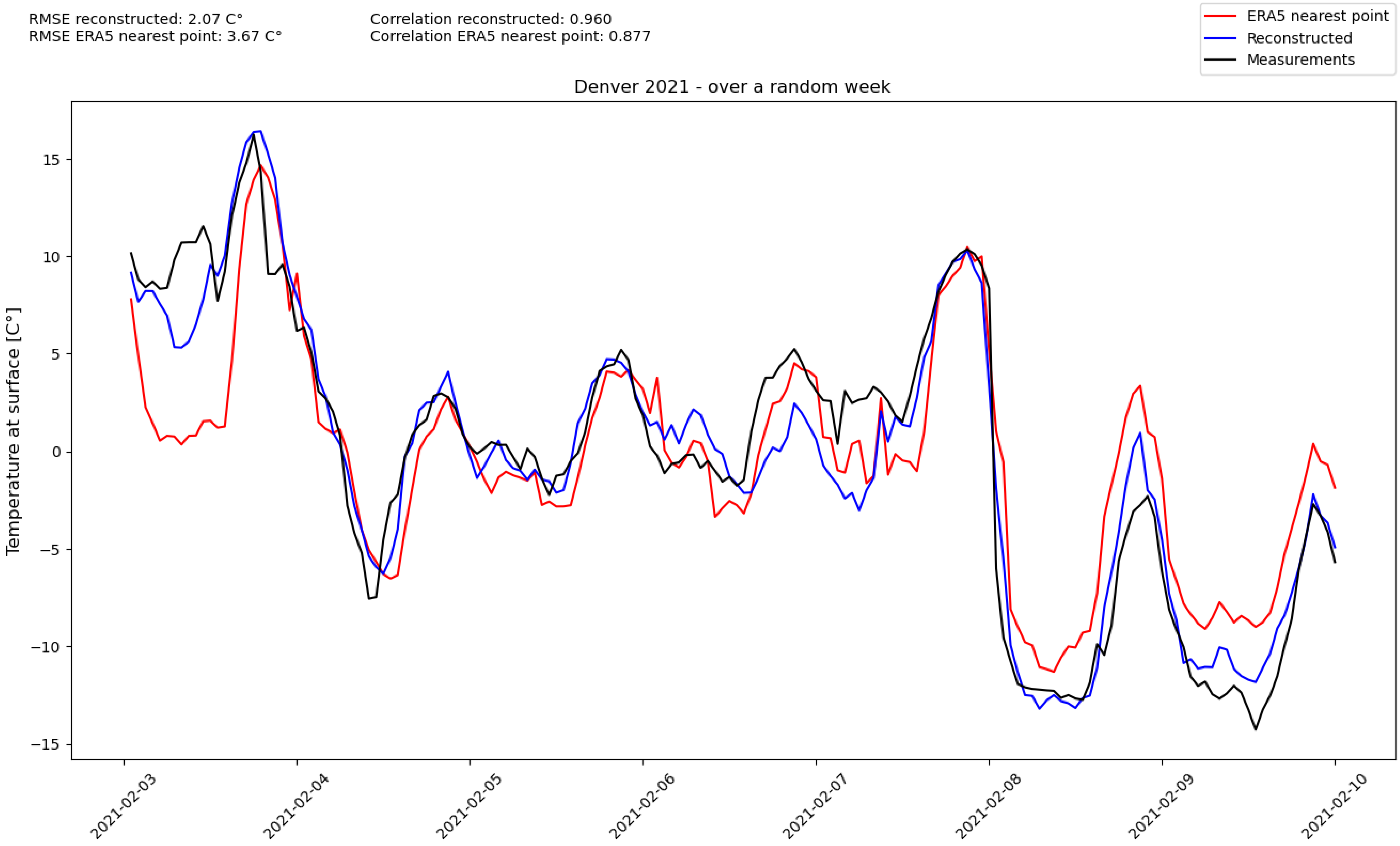
```
def plot_n_steps_of_df(df, as_delta, n=None, title=None, boxplot=False):

    from matplotlib import pyplot as plt

    time = df.index.values
    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])**2) / len(time))
   # 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) / len(time))

    correlation_reconstructed = np.corrcoef(reconstructed_median_values[time_slice], measurements_values[time_slice])[0,1]
   # 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], color="black", linewidth=1)
        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)
```

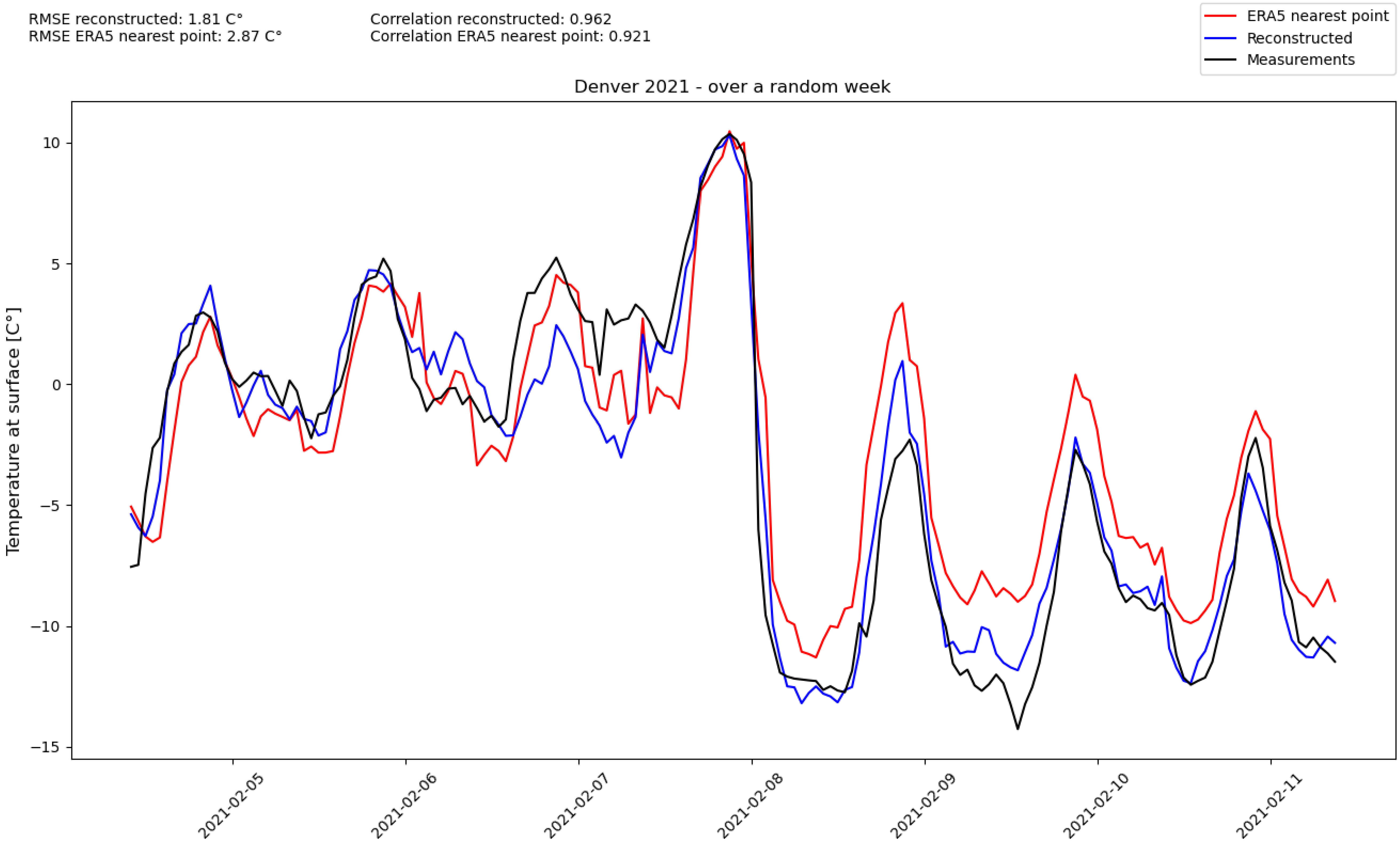
```python
    # font size of legend
    plt.rcParams.update({'font.size': 10})

    # 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 [ ]:

```python
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 hourly steps"
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)
```

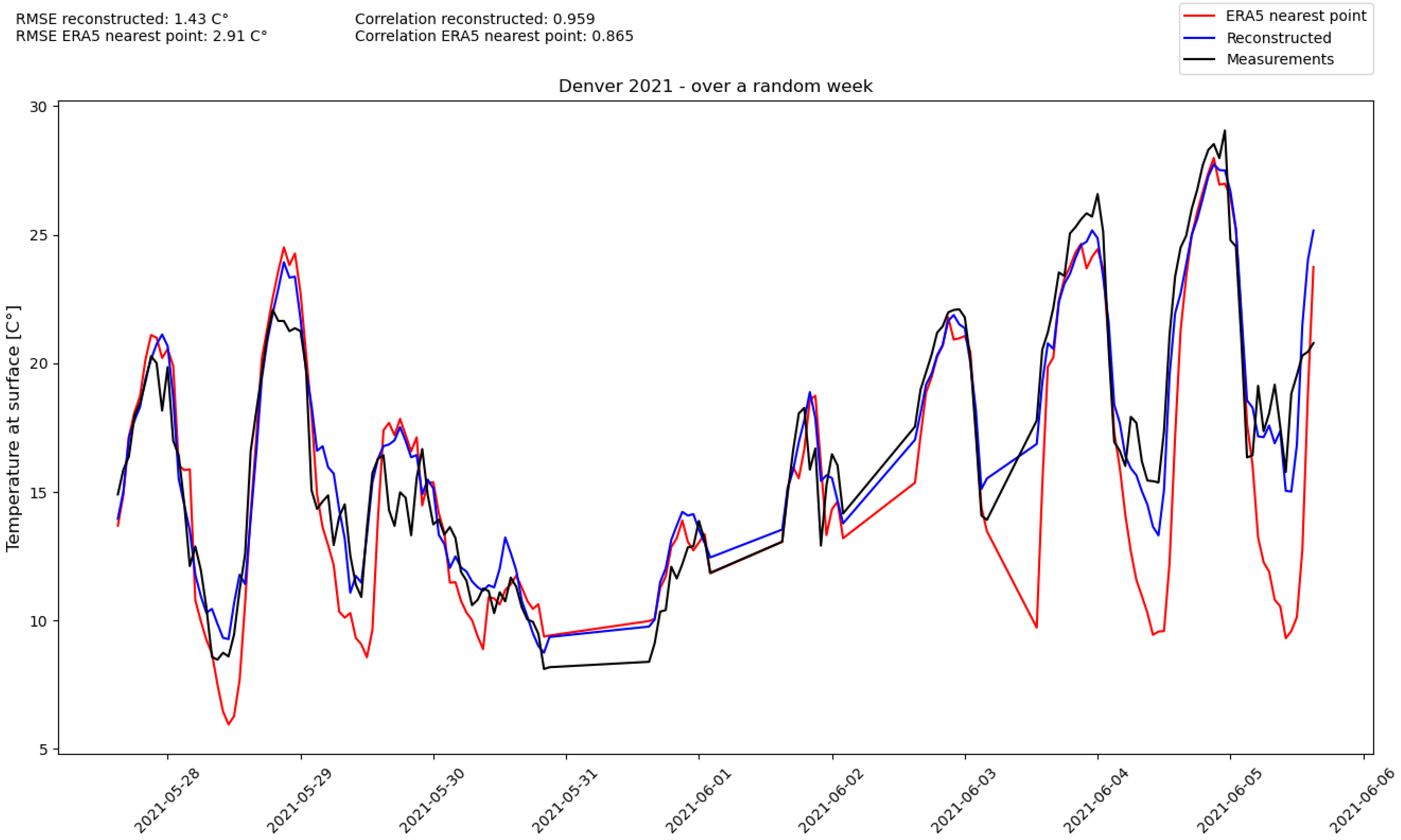

RMSE reconstructed: 2.07 C°
RMSE ERA5 nearest point: 3.67 C°
Correlation reconstructed: 0.960
Correlation ERA5 nearest point: 0.877
ERA5 nearest point
Reconstructed
Measurements
Denver 2021 - over a random week
Temperature at surface [C°]
15
10
5
0
−5
−10
−15
2021-02-03
2021-02-04
2021-02-05
2021-02-06
2021-02-07
2021-02-08
2021-02-09
2021-02-10



RMSE reconstructed: 1.81 C°
RMSE ERA5 nearest point: 2.87 C°

Correlation reconstructed: 0.962
Correlation ERA5 nearest point: 0.921

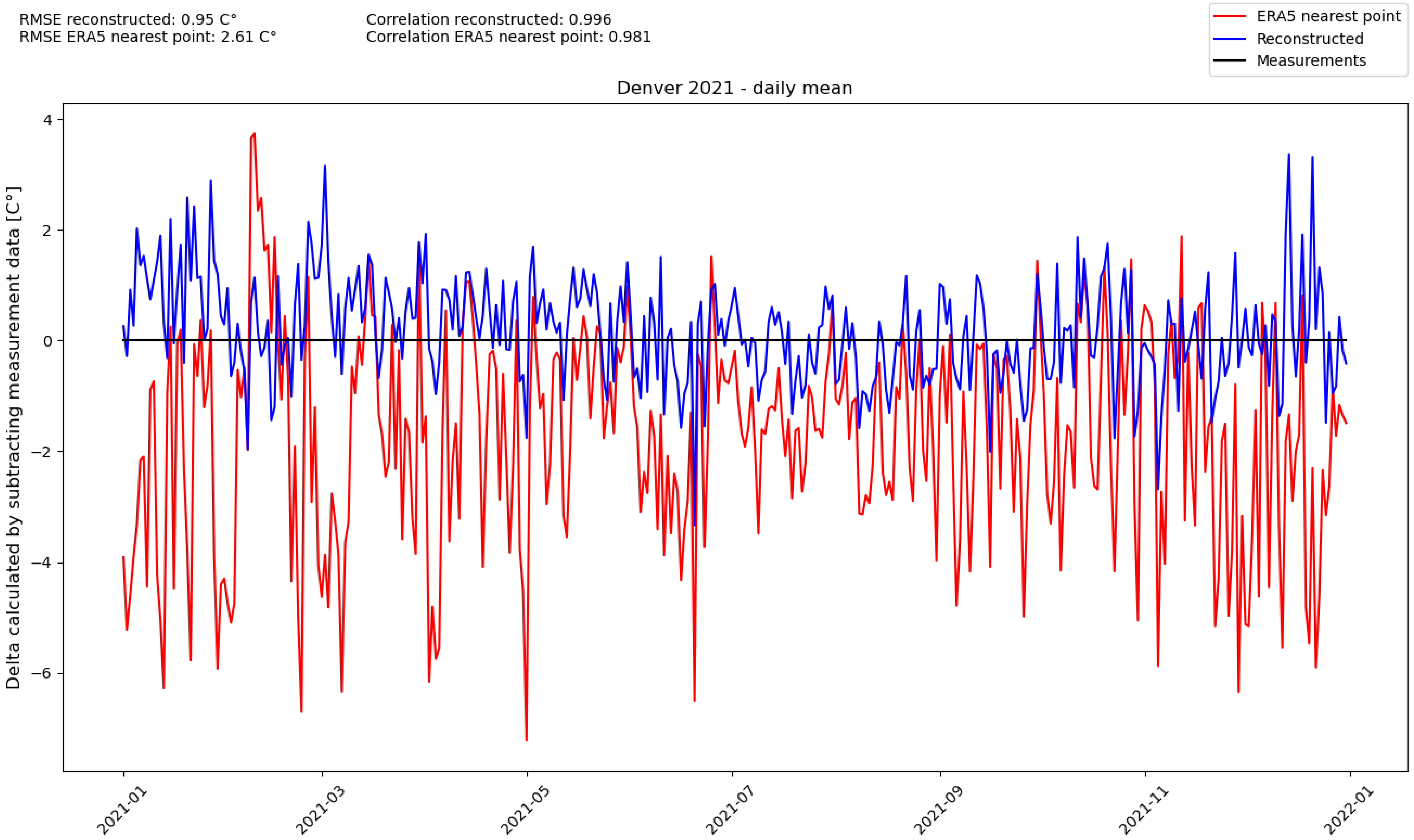
Denver 2021 - over a random week



Resample Data from hourly to daily or monthly

In [ ]:

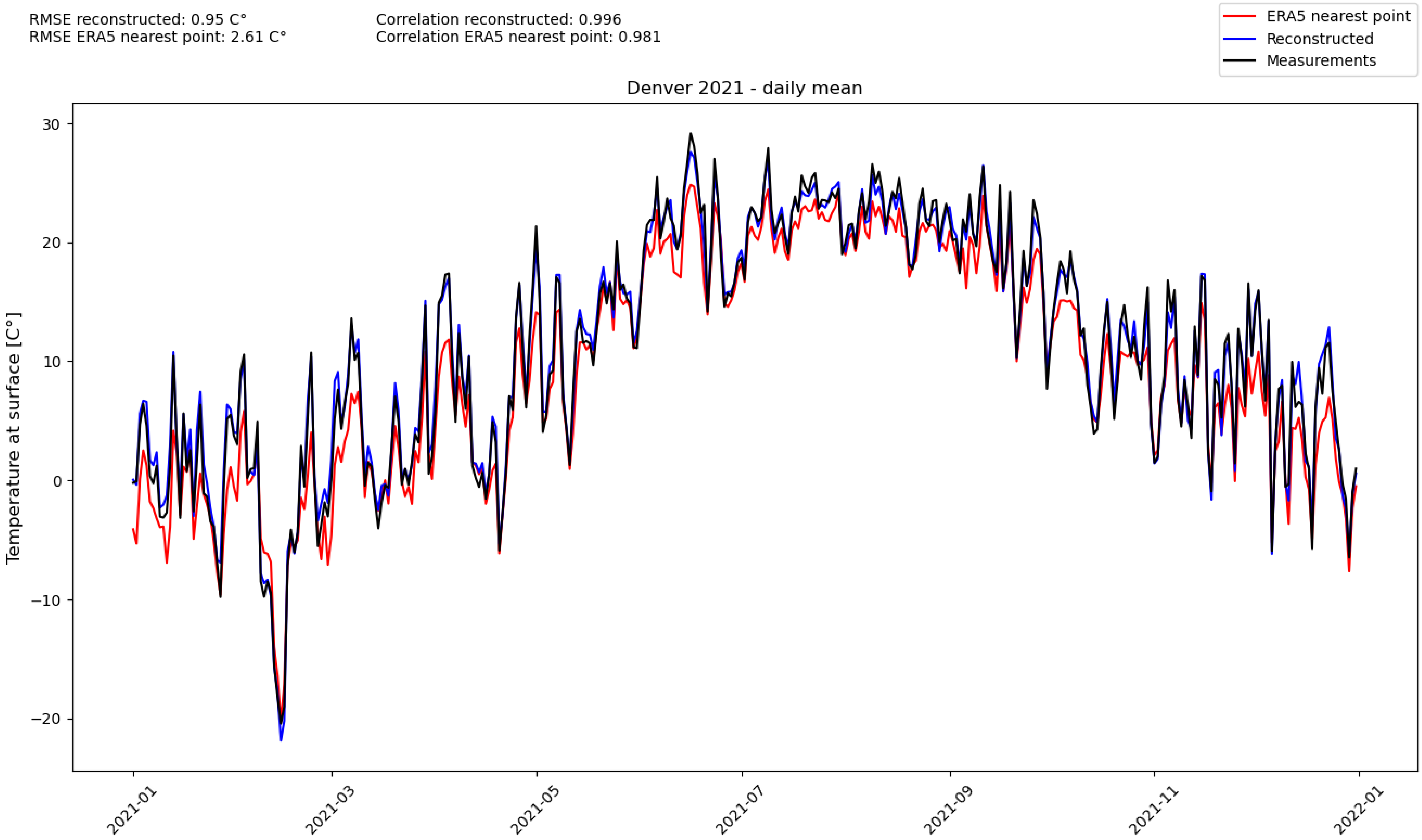
```
# resample to daily mean
daily_df = hourly_df.resample("D").mean()
# drop nans
daily_df = daily_df.dropna()
title = f"{'Denver' if station_name == 'Marshall' else station_name} {test_year} - daily mean"
plot_n_steps_of_df(daily_df, as_delta=True, title=title)
plot_n_steps_of_df(daily_df, as_delta=False,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



RMSE reconstructed: 0.95 C°
RMSE ERA5 nearest point: 2.61 C°

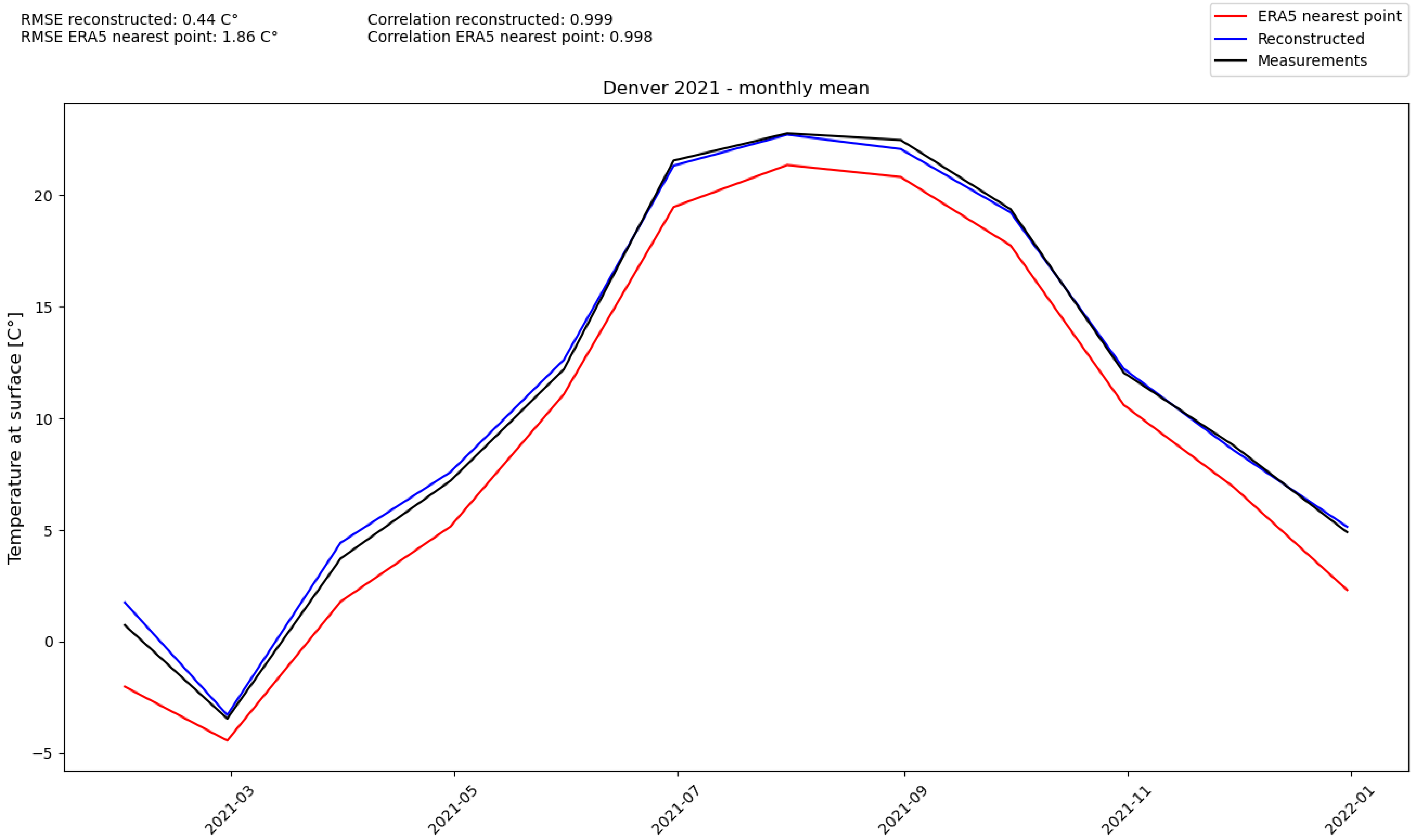
Correlation reconstructed: 0.996
Correlation ERA5 nearest point: 0.981

Denver 2021 - daily mean



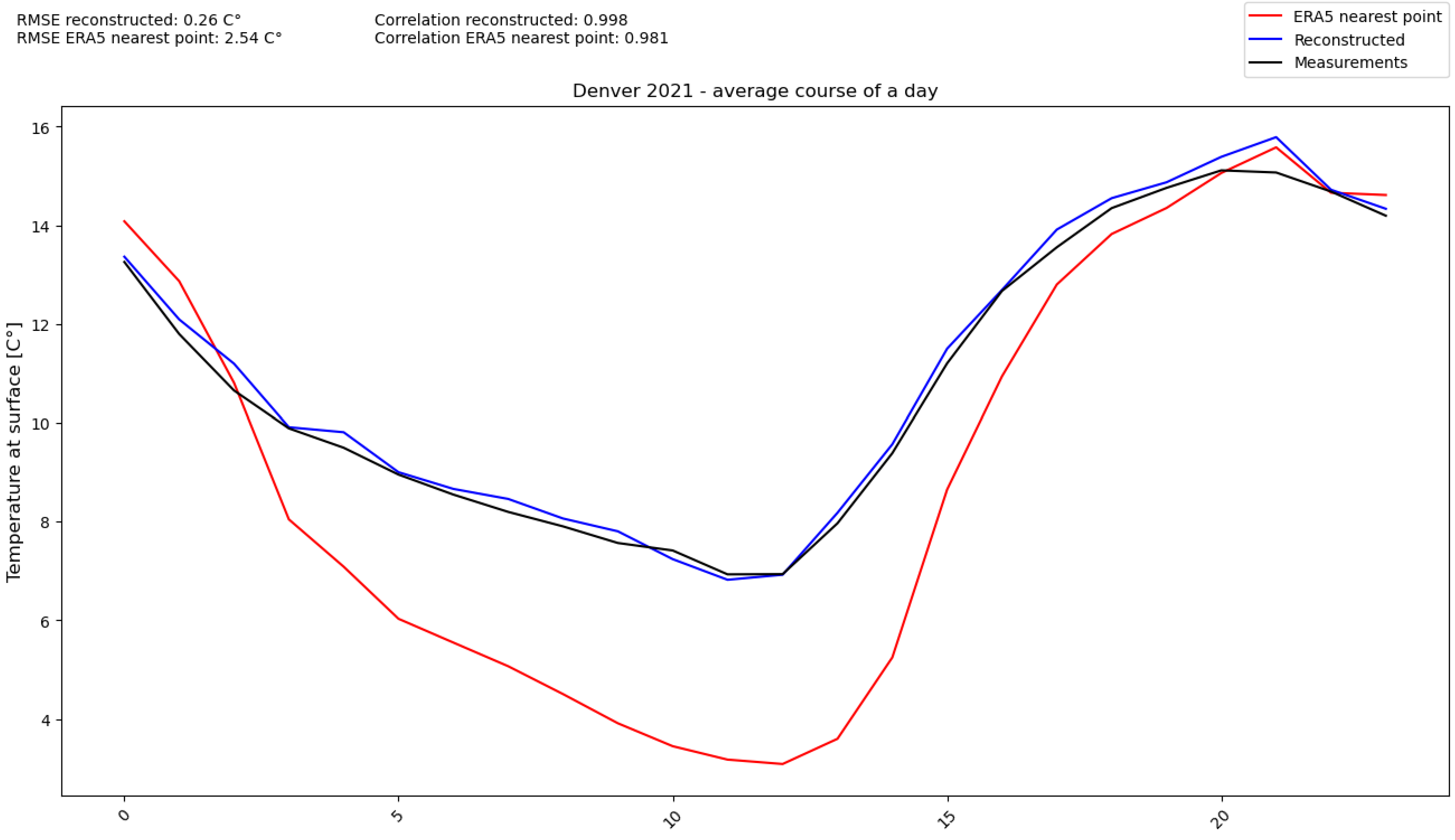
In [ ]:

```
# 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 [ ]: # 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)
```

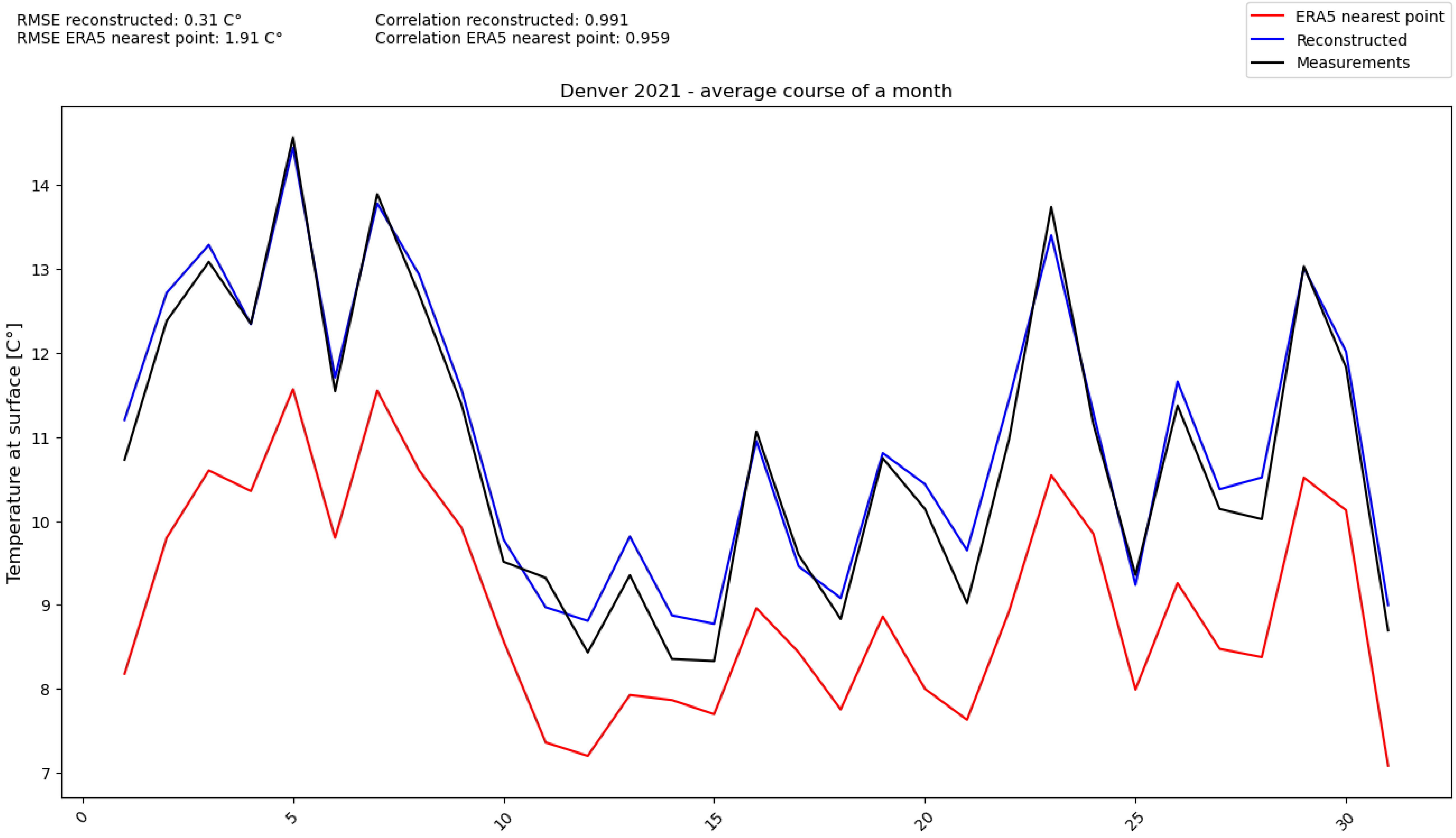


RMSE reconstructed: 0.26 C°
RMSE ERA5 nearest point: 2.54 C°

Correlation reconstructed: 0.998
Correlation ERA5 nearest point: 0.981

Average Course of the month

```
In [ ]: # 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)
```


RMSE reconstructed: 0.31 C°
RMSE ERA5 nearest point: 1.91 C°
Correlation reconstructed: 0.991
Correlation ERA5 nearest point: 0.959
ERA5 nearest point
Reconstructed
Measurements
Denver 2021 - average course of a month
Temperature at surface [C°]
7
8
9
10
11
12
13
14
0
5
10
15
20
25
30