Equinor Project Report

Jonas Mundheim Strand

August 4, 2023

Contents

1 INTRODUCTION

The aim for this part of the project is to create Python scripts able to download weather data from historical observation archive and numerical weather prediction models. This report will describe and elaborate the data sources and key features of the scripts. Both historical observation data from offshore/onshore weather stations and numerical weather prediction model data is collected from services operated by Norwegian Meteorological Institute.

2 PROGRAMMING LANGUAGE

All scripts are written in Jupyter notebooks with Python language. Jupyter allows to run sections of the code and store the result internally, this makes it easier to identify possible errors and understand the process of each section. Combined with the markdown function making it possible to write comments and explanations as easy-reference, Jupyter notebook was chosen as the preferred instrument.

3 OBSERVATION DATA

Historical observation data from several meteorological weather stations is accessible through Norwegian Meteorological Institute's channel called Frost. The Frost API provides free access to MET Norway's archive of historical weather and climate data. This data includes quality controlled daily, monthly, and yearly measurements of temperature, precipitation, and wind data. Other information, like metadata about weather stations, is also available through the API.

3.1 SEKLIMA

Seklima.no is a website used to visualise data from weather stations based on the exact same meteorological archive as Frost API. Visit the seklima website https://seklima.met.no/observations/ for a map-overview of all available weather stations with locations and detailed information about each station. This website is a good tool to use for visualising station locations and information like available weather measurements for each station, operating time etc. However, to download large quantities of data the Frost API will be used.

3.2 FROST API

As mentioned, provides the Frost API free access to MET Norway's archive of historical weather and climate data. To get access to this service the user needs to create their own client ID, this is done quick and easy only asking for an email address (create user ID: https://frost.met.no/auth/requestCredentials.html).

3.2.1 ABOUT THE SCRIPT

The Frost API service allows the user to download available weather data from chosen stations for a given time-period set by the user. The script: (**FROST API Retrieval.ipynb**) is created with the intention to retrieve data from one station using the station ID for that unique station for a time-period referred to as (*ref time*) in the script. In the (*data to retrieve*) list, element IDs for the variables to be extracted is listed. The user will be able to change the (*station id*), (*ref time*) and (*element IDs*) in the (*data to retrieve*). Several element IDs can be selected at a time but be aware that only 100.000 observations can be downloaded for each run.

3.2.2 IMPORTANT NOTES

Since the observational data is collected from operating weather stations there are some important things to be aware of.

- **Missing data**

Stations might experience downtime due to maintenance or technical difficulties resulting in missing data for these timestamps. A way to identify missing timestamps is included in the example file: (**GOLIAT 2022.ipynb**)

- **Operating time**

Every station started operating at different times. Check the individual station for operating time and available reference time for data. This also yields for each individual weather variable at the station.

- **Element ID**

To call for the variable to be extracted from the station element IDs is used in the (*data to retrieve*) list. Seklima can be used to check for available weather variables for each station. However, it is not always easy to match the variable from seklima with the element ID needed to extract the variable from Frost. Sometimes one element ID will return several different measurements. This is illustrated for "wind speed" in the (**FROST API Retrieval.ipynb**) script.

- **Quality of data**

The data is run through a quality control system that can assign several quality flags which mean different things. Frost tries to give the user a single value to represent the general quality of the observation. This value is set in the field qualityCode.

Exposure category is used to give information about the quality of the placement of a station. Performance category is used to give information about the quality of a sensor (which is located at a given station). More information about qualityCode and performance and exposure categories with values can be found: https://frost.met.no/dataclarifications.html

- **Adjusted wind speed**

For all offshore weather stations, the wind speed data is adjusted to 10 m.a.s.l regardless of the station's sensor height.

- **Measurements**

The different measurements use different techniques to acquire the measurement value. For example will when calling elemnt ID *Wind speed* be a registerd mean value of the wind speed over the last 10 minutes before the observation time. Description for how the measurement value for the individual element ID is available at https://frost.met.no/elementtable

4 NORA3

4.1 ABOUT THE MODEL

NORA3 dataset is created by the Norwegian Meteorological Institute by running the non-hydrostatic NWP model HARMONIE-AROME (Cy 40h1.2), solving the fully compressible Euler equations. The state-of-the-art reanalysis ERA5 from ECMWF is used as initial and boundary forcing. ERA5 is a global reanalysis product, providing hourly information for 137 vertical layers, and covers the Earth with 0.28125° resolution, corresponding to a horizontal grid of approximately 31×31km. The improved resolution of ERA5 over ECMWF's former reanalysis ERA-Interim (79km) provides a detailed initial and boundary information in the downscaling process. NORA3 covers large parts of the North Atlantic and the Nordic countries in a 3×3km non-staggered horizontal grid (see Fig. 1 for the complete NORA3 domain), dividing the atmosphere into 65 vertical layers. The NORA3 near surface output data are available every hour.

NORA3 provides a deterministic forecast every six hours with lead times up to +9 hours and a spatial grid resolution of 3 km. Lead times +4 to +9 h from runs starting every six hours provide a complete archive for near surface parameters with hourly resolution. The lead times +6 and +9 h provide a complete three-hourly archive for upper atmospheric parameters. The lead time +3 h is also stored. This lead time is meant to be used for correcting accumulated parameters. (The accumulated parameters are summed up from the start of the forecast.)

With a horizontal resolution of 3 km, the nonhydrostatic numerical weather prediction model HARMONIE–AROME runs explicitly resolved deep convection and yields hindcast fields that realistically downscale the ERA5 reanalysis. The wind field is much improved relative to its host analysis, in particular in mountainous areas and along the improved grid-resolving coastlines. NORA3 also performs much better than the earlier hydrostatic 10-km Norwegian Hindcast Archive (NORA10) in complex terrain. NORA3 recreates the detailed structures of mesoscale cyclones with sharp gradients in wind and with clear frontal structures, which are particularly important when modeling polar lows. In extratropical windstorms, NORA3 exhibits significantly higher maximum wind speeds and compares much better to observed maximum wind than do NORA10 and ERA5. The activity of the model is much more realistic than that of NORA10 and ERA5, both over the ocean and in complex terrain.

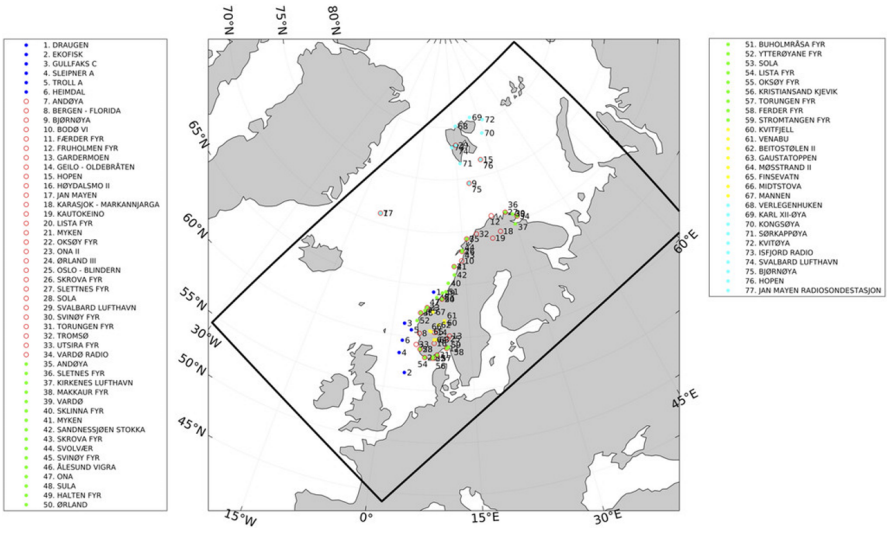


Figure 1: NORA3 domain and location of observation stations

4.2 ABOUT THE SCRIPT

The Jupyter notebook scripts:

- **Download NORA3 wind atoms**
- **Download NORA3 atoms**
- **Download NORA3 fpnc**

Are all based on data from the NORA3 model but with some differences. All the files contains hourly update but with different available variables. The file with the most variables is the **Download NORA3 fpnc**, as shown in appendix A.

Download NORA3 wind atoms uses a data set only containing wind speed and direction. In these files the wind parameters are already rotated to cardinal directions using degrees. An example of how this can be used to create a wind rose is included in the script. Unique for this data set is that both wind speed and direction is already calculted to different heights and it is possible to extract values for heights: 10m, 20m, 50m, 100m, 250m, 500m, 750m

Download NORA3 atoms contains rotated wind parameters but only at 10 m.a.s. These files will however, contain several variables, see appendix. Important to note is that all variables from these files are at a fixed height.

Download NORA3 fpnc is the file with the most available variables, see appendix. The height in these files are described by *hybrid level*, some relevant heights in meters are included as comments in the script. The calculation for this conversion is described in appendix D. Important to note is that the downloading time using these files are much more time consuming than for the two others.

The majority of the values from the variables in the NORA3 data is values from that exact moment. This is the case for wind speed and direction, which is a difference from the observation data which often takes the mean over the last 10 minutes before the observation time

All the scripts contains coordinate suggestions for approximate center of some offshore wind sites. This is found by using https://www.nve.no/energi/energisystem/havvind/havvindkart/ offshore wind site coordinates and then https://www.norgeskart.no/#!?project=norgeskart&layers=1002&zoom=3&lat=7197864.00&lon=396722.00 to estimate the center. This and the nearest weather station is included in the appendix B.

5 AROME-ARCTIC

5.1 ABOUT THE MODEL

The following information about the AROME-Arctic model is retrieved from one of the co-developers of the model, Norwegian Meteorological Institute, https://www.met.no/en/projects/The-weather-model-AROME-Arctic/about

AROME-Arctic is a regional short-range high-resolution forecasting system for the European Arctic with 2.5 km grid spacing and 65 vertical levels (Müller et al. 2017, Køltzow et al. 2019). The model system is based on the HARMONIE-AROME configuration of the ALADIN-HIRLAM numerical weather prediction system (Bengtsson et al. 2017). The HARMONIE-AROME system is part of the code base of the IFS/ARPEGE system of ECMWF and Meteo-France, and it uses the same non-hydrostatic dynamical core as AROME-France described by Seity et al. (2010).

AROME-Arctic is very similar to the Nordic setup MetCoOp (Müller et al. 2017). Since June 2017, the operational AROME-Arctic is based on the cycle version 40h1.1. It issues deterministic forecasts 4 times a day with a lead time of 66 hours.

- **Data assimilation**

AROME-Arctic utilises a three-dimensional variational data assimilation (described by Brousseau et al. 2011) with 3-hourly cycling to assimilate conventional observations, scatterometer ocean surface winds (Valkonen et al. 2017), satellite radiances (Randriamampianina et al. 2019) and atmospheric motion vectors (Randriamampianina et al. 2017). Additionally, the operational system includes a so called "large scale mixing" where the large spatial scales are blended with the ECMWF high resolution model (HRES) forecasts to obtain the background field. Data assimilation of surface variables (2 m temperature, 2 m humidity and snow depth) in the AROME-Arctic is based on optimal interpolation (Giard and Bazile, 2000).

- **Boundary conditions**

The ECMWF HRES forecasts with 1-h interval are used as lateral boundary conditions. Sea ice concentration and sea surface temperature are also obtained from the ECMWF. The surface temperature over sea-ice is subsequently modelled by a 1D sea-ice model (Batrak et al. 2018).

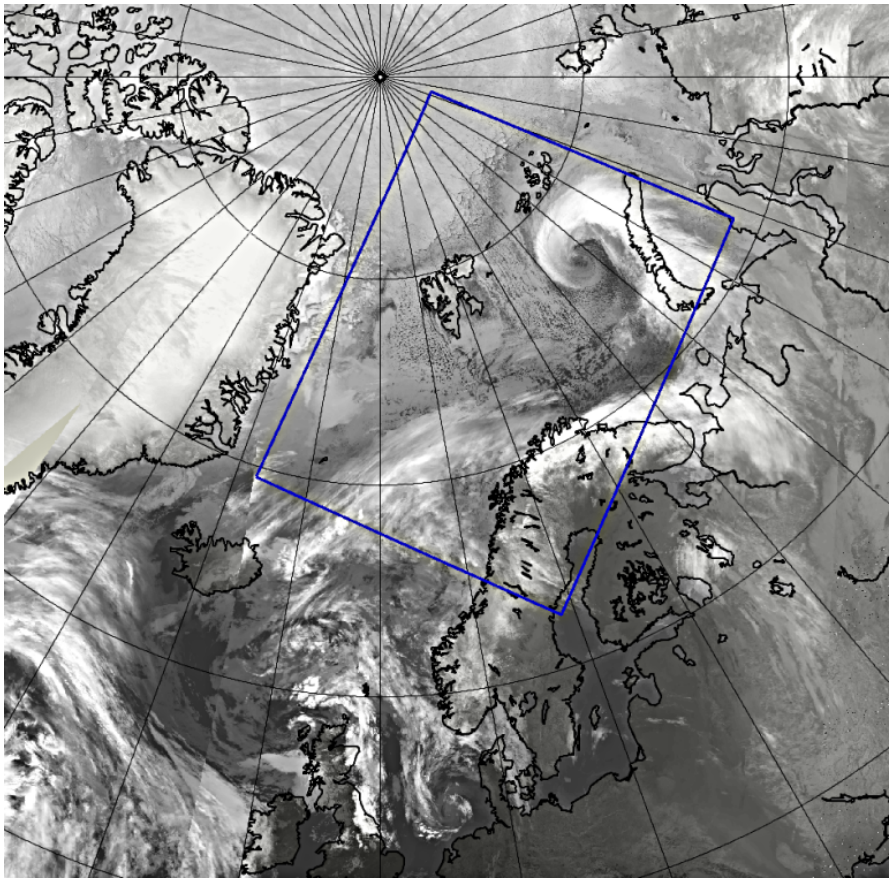


Figure 2: AROME-Arctic model domain

5.2 ABOUT THE SCRIPT

A script similar to the ones created for downloading data from NORA3 is made for downloading data from AROME-Arctic files. These files are available at: https://thredds.met.no/thredds/catalog/aromearcticarchive/catalog.html starting from year 2015.

The model provides a 66h forecast for each file which means that the accuracy of the data will decrease with time in the files. There is also some error in the first 6h in the data due to *spin-up* when processing new information. The script solve this by collecting 3h from each file between time 6h to 9h, before looping to next file.

- **Missing hour-files**

Some days files with a specific hour is missing. The script will solve this by identify the missing file and continue to the next available file. However, for some months there are several files missing resulting in a loss of data. The effect from this is depending on how many total files that are missing. For each file not found in the archive 3 hours of data will be skipped.

- **Different files**

There are several different types of files stored as output from the model. Check the thredds link for available files and what variables each file contains. The files with "extracted" contains rotated wind in cardinal directions (degrees) and a value for the wind speed at 10m.

- **Grid rotation**

The wind components from the model is given in x- and y-parameters. To calculate the cardinal direction of the wind an local grid rotation at the specific coordinate needs to be calculated. The script: (**Alpha local grid rotation AA**) is created for this purpose, calculating the alpha for each coordinate in an AROME-Arctic file. **PS: there are some issues with the code and it does not seem to work!** The script calculating the alpha creates a new ncfile with alpha for all coordinates and is not directly connected to the downloading of the variables from AROME files.

6 SCRIPTS

This chaper will inculde the Jupyter notebook scripts for downlading data from Frost API and NORA3

6.1 FROST API SCRIPT

The following script is the Jupyter notebook for how to download historical observation data from weather stations using Frost API. The script should give an easy-reference of the process. This script acts as an starting point for further development and modifications but it can be used as is. More information at https://frost.met.no/howto.html

6.1.1 FROST API RETRIEVAL

Starts at next page

Seklima data retrival with Frost API

Link to frost webpage (https://frost.met.no/index.html)

1. Create client id for yourself
   - Create client id here (https://frost.met.no/auth/requestCredentials.html)
2. Find weather elements you want to retrieve with:
   - Weather element registrer (https://frost.met.no/elementtable)
3. Change station ID to get station coordinates
   - stations (https://seklima.met.no/stations/)

More information: https://frost.met.no/python_example.html (https://frost.met.no/python_example.html)

In [1]:

```
# Import modules

import numpy as np
import pandas as pd

import requests

import matplotlib.pyplot as plt

from __future__ import print_function
from IPython.display import display, HTML
```

In [2]:

```
# Insert your own client ID here

client_id = 'insert client ID here'

# Insert station ID for the station retrieving data from

station_id = 'SN76956' # insert station ID, EX; 'SN76956' Goliat ,  'SN20926' hjelms
```

Retrieve data from Frost API

This section gets the data from the Frost API service for seklima

- Change ref_time to time period you want data from
- Change data_to_retrieve with element IDs you want to retrieve, element IDs: Weather element registrer (https://frost.met.no/elementtable)

Observations(Element IDs) and time available vary for each station, check this at stations (https://seklima.met.no/stations/)

MAX 100.000 observations can be downloaded each run

In [8]:

```
##############################################

# Change ref_time to time period

ref_time = '2022-01-01/2023-01-01'


# fill this list with elements to retrieve according to frost API frost.met.no spesi
# The elements in this scripts is examples

data_to_retrieve = ['wind_speed'#,
                    #'wind_from_direction'#,
                    #'wind_speed_of_gust'
                    ]

##############################################



# Define endpoint and parameters

endpoint = 'https://frost.met.no/observations/v0.jsonld'
parameters = {
    'sources': f'{station_id}',
    'elements': ','.join(data_to_retrieve),
    'referencetime': f'{ref_time}',
}

# Issue an HTTP GET request
r = requests.get(endpoint, parameters, auth=(client_id,''))
# Extract JSON data
json = r.json()

# Check if the request worked, print out any errors
if r.status_code == 200:
    data = json['data']
    print('Data retrieved from frost.met.no!')
else:
    print('Error! Returned status code %s' % r.status_code)
    print('Message: %s' % json['error']['message'])
    print('Reason: %s' % json['error']['reason'])

```

```
Data retrieved from frost.met.no!
```

Convert to dataframe

Convert the retrieved data into dataframes

In [4]:

```
1
2  # This will return a Dataframe with all of the observations in a table format
3  # This can take some minutes
4
5  df = pd.DataFrame()
6  for i in range(len(data)):
7      row = pd.DataFrame(data[i]['observations'])
8      row['referenceTime'] = data[i]['referenceTime']
9      row['sourceId'] = data[i]['sourceId']
10     df = pd.concat([df, row])
11
12 df = df.reset_index(drop=True)
13
14 display(df)
```

	elementId	value	unit	level	timeOffset	timeResolution	timeSeriesI
0	wind_speed	9.8	m/s	{'levelType': 'height_above_ground', 'unit': '...	PT0H	PT10M	(
1	wind_speed	10.3	m/s	{'levelType': 'height_above_ground', 'unit': '...	PT0H	PT3H	(
2	wind_speed	9.9	m/s	{'levelType': 'height_above_ground', 'unit': '...	PT0H	PT10M	(
3	wind_speed	10.8	m/s	{'levelType': 'height_above_ground', 'unit': '...	PT20M	PT30M	(
4	wind_speed	14.5	m/s	{'levelType': 'height_above_ground', 'unit': '...	PT0H	PT10M	(
...	...	...	...	...	...	...	..
44384	wind_speed	7.6	m/s	{'levelType': 'height_above_ground', 'unit': '...	PT0H	PT10M	(
44385	wind_speed	7.1	m/s	{'levelType': 'height_above_ground', 'unit': '...	PT0H	PT10M	(
44386	wind_speed	9.8	m/s	{'levelType': 'height_above_ground', 'unit': '...	PT20M	PT30M	(
44387	wind_speed	5.6	m/s	{'levelType': 'height_above_ground', 'unit': '...	PT0H	PT10M	(
44388	wind_speed	10.3	m/s	{'levelType': 'height_above_ground', 'unit': '...	PT20M	PT30M	(

44389 rows × 12 columns

Information about:

- PerformanceCategory, exposureCategory, qualityCode etc (https://frost.met.no/dataclarifications.html

In [9]:

```
"""
Some datasets contains several different types of observations with the same elemntI
"timeOffset" but different "timeResolution" and/or "timeOffset".
TimeOffset can be modified in the "getting the data", but keys with a given name can
This shows how to remove rows with unwanted values.

If datafram contains all desired values, skip this!
"""


# Find outliers
unique_time_resolution = df['timeResolution'].unique()
unique_timeOffset = df['timeOffset'].unique()
unique_timeSeriesId = df['timeSeriesId'].unique()
unique_performanceCategory = df['performanceCategory'].unique()
unique_exposureCategory = df['exposureCategory'].unique()
unique_qualityCode = df['qualityCode'].unique()



print(unique_time_resolution)
print(unique_timeOffset)



# Example: remove all observations without 'timeresolution' = PT10M and 'timeOffset'
# Keep what we want
df = df.loc[df['timeResolution'] == 'PT10M']
df = df.loc[df['timeOffset'] == 'PT0H']

display(df)
```

```
['PT10M' 'PT3H' 'PT30M']
['PT0H' 'PT20M']
```

	elementId	value	unit	level	timeOffset	timeResolution	timeSeriesI
0	wind_speed	9.8	m/s	{'levelType': 'height_above_ground', 'unit': '...	PT0H	PT10M	(
2	wind_speed	9.9	m/s	{'levelType': 'height_above_ground', 'unit': '...	PT0H	PT10M	(
4	wind_speed	14.5	m/s	{'levelType': 'height_above_ground', 'unit': '...	PT0H	PT10M	(
6	wind_speed	10.5	m/s	{'levelType': 'height_above_ground', 'unit': '...	PT0H	PT10M	(
7	wind_speed	10.8	m/s	{'levelType': 'height_above_ground', 'unit': '...	PT0H	PT10M	(
...	...	...	...	...	...	...	..
44380	wind_speed	7.1	m/s	{'levelType': 'height_above_ground', 'unit': '...	PT0H	PT10M	(
44382	wind_speed	5.8	m/s	{'levelType': 'height_above_ground', 'unit': '...	PT0H	PT10M	(
44384	wind_speed	7.6	m/s	{'levelType': 'height_above_ground', 'unit': '...	PT0H	PT10M	(
44385	wind_speed	7.1	m/s	{'levelType': 'height_above_ground', 'unit': '...	PT0H	PT10M	(
44387	wind_speed	5.6	m/s	{'levelType': 'height_above_ground', 'unit': '...	PT0H	PT10M	(

26027 rows × 12 columns

In [ ]:

```
1
```

In [10]:

```
"""
Simplify the dataframe
"""

# keep these columns
columns = ['referenceTime','elementId', 'value']
df2 = df[columns].copy()


# Convert the time value to something Python understands
df2['referenceTime'] = pd.to_datetime(df2['referenceTime'])

# Pivot the DataFrame to have 'elementId' as columns and 'referenceTime' as index
df3 = df2.pivot_table(index='referenceTime', columns='elementId', values='value')

# Rename the index label
df3 = df3.rename_axis(['time'])

display(df3)
```

elementId	wind_speed
time	
2022-01-01 00:00:00+00:00	9.8
2022-01-01 00:20:00+00:00	9.9
2022-01-01 00:40:00+00:00	14.5
2022-01-01 01:00:00+00:00	10.5
2022-01-01 01:20:00+00:00	10.8
...	...
2022-12-31 22:20:00+00:00	7.1
2022-12-31 22:40:00+00:00	5.8
2022-12-31 23:00:00+00:00	7.6
2022-12-31 23:20:00+00:00	7.1
2022-12-31 23:40:00+00:00	5.6

26027 rows × 1 columns

Save dataframe as NC-file

Define a function that saves the dataframe as an nc-file.

- Change the output_filename to desired new saved filename

Filesize for time and wind speed (timeresolution: every 20 min) for 12 months: 411 KB

In [11]:

```
"""
This code makes a new optional directory at a desired location and saves the new fil
If the directory already exist it will just save the new file in that directory. Thi
in the same directory by keeping the same directory name and changing only the outpu
"""

import xarray as xr
import os

########################################

# New directory
directory = "weather_data"

# Parent Directory path
parent_dir = "C:/Users/Ida/" # Change path

# Filename new ncfile
output_filename = f'seklima_weather_data_stations_{station_id}_2022' # Choose filena

########################################


# Path
path = os.path.join(parent_dir, directory)

# Create the directory if not already existing
if not os.path.isdir(path):
    os.mkdir(path)


def save_dataframe_to_netcdf(dataframe, output_file):
    dataset = xr.Dataset(data_vars=dataframe.to_dict('series'))
    output_path = f"{parent_dir}{directory}/{output_file}"
    dataset.to_netcdf(output_path)

    print(f"Data saved successfully to {output_path}.")


save_dataframe_to_netcdf(df3, output_filename)
```

```
Data saved successfully to C:/Users/Ida/weather_data/seklima_weather_data_
stations_SN76956_2022.
```

How to use the new ncfile

In [12]:

```
import netCDF4 as nc

filename = 'C:/Users/Ida/weather_data/seklima_weather_data_stations_SN76956_2022'
ncfile = nc.Dataset(filename)

# print variables in the file
for variable in ncfile.variables:
    print(variable)

```

```
time
wind_speed
```

In [13]:

```
# convert timestamps into nice readable dates

import datetime as dt
from datetime import date, timedelta

time = ncfile.variables["time"][:]

ts = np.linspace(0,len(time)-1,len(time))

nice_time = []

for i in range(0,len(ts)):
    t = dt.datetime.utcfromtimestamp(time[i]/10**9)
    nice_time.append(t)

```

In [14]:

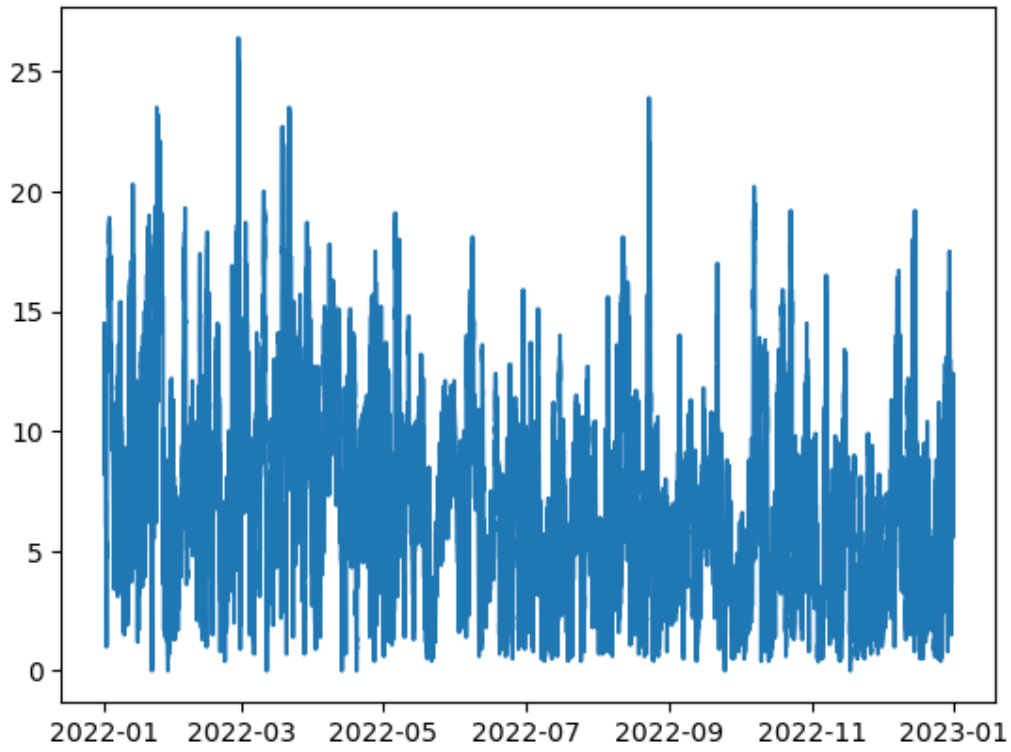
```
# Extract variable
wind_speed = ncfile.variables["wind_speed"][:]

```

In [15]:

```
# Simple plot

plt.plot(nice_time, wind_speed)
```

Out[15]:

```
[<matplotlib.lines.Line2D at 0x220cb783250>]
```



6.2 NORA3 SCRIPTS

Norwegian Meteorological Institute makes different types of files based on NORA3 data available(https://thredds.met.no/thredds/catalog/nora3/catalog.html and https://thredds.met.no/thredds/projects/nora3_subsets.html). The different files will include different variables, these variables are listed in the appendix. This section includes the Jupyter notebook scripts for relevant NORA3 files.

Some of the files will include wind directions rotated to cardinal directions making them easy to use. However, be aware of differences in height for all of the variables. Every script should describe the variables and provide information about important parameters and height differences.

The title in the following subsections represents the filename of the scripts

6.2.1 Download NORA3 wind atoms

Download data from NORA3 and save to ncfile

The following codes is a guide to download data from the NWP NORA3 at a chosen time period and save it as an nc-file. Data available at https://thredds.met.no/thredds/catalog/nora3_subset_atmos/wind_hourly/catalog.html (https://thredds.met.no/thredds/catalog/nora3_subset_atmos/wind_hourly/catalog.html). This script use the subset atmosphere files containing wind_speed and wind_direction at heights above surface (10m, 20m, 50m, 100m, 250m, 500m, 750m). choose a datespan (years, months) and save it as an ncfile.

- wind speed
- wind direction

Data available from: 1985.01 to (current month minus 6-7 months time lag), check link for latest date

Model run 4 times a day, every 6h, the files contains hourly updated data based on these runs

In [1]:

```
# Importing necessary modules

import xarray as xr
import numpy as np

import netCDF4 as nc
import matplotlib.pylab as plt
import pandas as pd

import pyproj
import datetime as dt
from datetime import date, timedelta

import requests
from __future__ import print_function
from IPython.display import display, HTML
```

In [1]:

```
def get_coordinates_center_OWS(location_name):
    locations = {
        'Nordavind_A': {
            'latitude': 71.1314956,
            'longitude': 32.048109
        },
        'Nordavind_B': {
            'latitude': 71.7880587,
            'longitude': 27.7221338
        },
        'Nordavind_C': {
            'latitude': 71.7471898,
            'longitude': 19.9808019
        },
        'Nordavind_D': {
            'latitude': 71.473272,
            'longitude': 18.7614613
        },
        'Utsira_nord': {
            'latitude': 59.2740869,
            'longitude': 4.5441279
        },
        'Sørlige_nordsjø_2': {
            'latitude': 59.7894747,
            'longitude': 4.9314619
        }
    }

    if location_name in locations:
        coordinates = locations[location_name]
        return coordinates['latitude'], coordinates['longitude']
    else:
        return None


```

Seklima station information retrival with frost api

Link to frost webpage (https://frost.met.no/index.html)

1. Create client id for yourself
   - Create client id here (https://frost.met.no/auth/requestCredentials.html)

This is used to retrive the longitude and latitude

In [2]:

```
# Insert your own client ID here
client_id = 'insert client ID'
```

In [3]:

```
'''
Getting the data for the meta dataframe

Using the frost api for seklima
'''

endpoint = 'https://frost.met.no/sources/v0.jsonld'

# Dictionary with elements to retrieve
parameters = {
    'fields': 'name,id,geometry,masl,validFrom'
}

# Issue an HTTP GET request
r = requests.get(endpoint, parameters, auth=(client_id,''))
# Extract JSON data
json = r.json()

# Check if the request worked, print out any errors
if r.status_code == 200:
    data_exp = json['data']
    print('Data retrieved from frost.met.no!')
else:
    print('Error! Returned status code %s' % r.status_code)
    print('Message: %s' % json['error']['message'])
    print('Reason: %s' % json['error']['reason'])
```

```
Data retrieved from frost.met.no!
```

In [4]:

```
'''
Transforming the data from a raw json format retrieved by requests
to a pandas df format

When redefining the query the new columns need to be spesified for the dataframe df
'''

meta_df = pd.DataFrame(columns=['id', 'name', 'lon', 'lat', 'heigh-asl (m)','operati
ignored_values = 0
for i in range(len(data_exp)):
    row = []
    try:
        row.append(data_exp[i]['id'])
        row.append(data_exp[i]['name'])
        row.append(data_exp[i]['geometry']['coordinates'][0])
        row.append(data_exp[i]['geometry']['coordinates'][1])
        row.append(data_exp[i]['masl'])
        row.append(data_exp[i]['validFrom'])
        meta_df.loc[len(meta_df)] = row
    except:
        ignored_values += 1
        continue


print(f'Number of discarded values {ignored_values}')

# Setting the station id as row index2
meta_df = meta_df.set_index('id')

display(meta_df)
```

```
Number of discarded values 88
```

id	name	lon	lat	heigh-asl (m)	operating period (from)
SN47230	ÅKRA UNGDOMSSKOLE	5.196300	59.255500	18	2013-10-29T00:00:00.000Z
SN23670	E16 RYFOSS	8.817500	61.137500	406	2018-01-23T00:00:00.000Z
SN59450	STADLANDET	5.211500	62.146700	75	1923-01-01T00:00:00.000Z
SN12590	E6 MJØSBRUA	10.672500	60.928000	128	2011-01-01T00:00:00.000Z
SN26640	E134 DARBU	9.777300	59.702500	155	2016-04-19T00:00:00.000Z
...	...	...	...	...	...
SN74780	NAMSVATN II	13.540300	64.960800	460	2019-09-01T00:00:00.000Z
SN1292500	NAGYKANIZSA	16.970556	46.456111	139	1979-05-20T00:00:00.000Z
SN1712400	ESKISEHIR HAVALIMANI	30.579700	39.781000	786	1978-05-31T00:00:00.000Z
SN94685	E69 VEDBOTN	25.629700	70.776500	10	2013-07-22T00:00:00.000Z
SN248690	SÖDERTÄLJE	17.628900	59.214200	23	1961-01-01T00:00:00.000Z

1826 rows × 5 columns

- Change station ID to match measurement station to get station coordinates
- [stations (https://seklima.met.no/stations/)](https://seklima.met.no/stations/)

or

- Insert site_name to get coordinates from center of OWS

In [5]:

```
...
#################

station_id = "SN76956" # insert station ID, ex: 'SN76956' Goliat, 'SN20926' hjelms

################

longitude = meta_df.loc[f"{station_id}"]["lon"]
latitude = meta_df.loc[f"{station_id}"]["lat"]

print(f"{station_id}: longitude = {longitude}, latitude = {latitude}")
```

```
SN76956: longitude = 22.25, latitude = 71.3112
```

- Change the site name to one of the locations in get_coordinates

In [3]:

```
# insert the OWS name you want the data extracted from

################

site_name = 'Nordavind_A'

################

latitude, longitude = get_coordinates_center_OWS(site_name) # gets the latitude and
```

In [ ]:

```
# Choose latitude, longitude

latitude =
longitude =
```

Projecting coordinates

Starts by projecting the chosen Latitude and Longitude coordinate to x and y cooridnates used in the model

- No changes needed in this part

In [6]:

```
"""
load in a random file from the NORA3 dataset, this is just used to project latitude/
x- and y-coordinate
"""

filename = "https://thredds.met.no/thredds/dodsC/nora3_subset_atmos/wind_hourly/arom
ncfile   = nc.Dataset(filename)



#######
# Projection
crs_NORA = pyproj.CRS.from_cf(
        {
           "grid_mapping_name": "lambert_conformal_conic",
           "standard_parallel": [66.3, 66.3],
           "longitude_of_central_meridian": -42.0,
           "latitude_of_projection_origin": 66.3,
            "earth_radius": 6371000.0,
        }
)

# Transformer to project from ESPG:4368 (WGS:84) to our lambert_conformal_conic
proj = pyproj.Transformer.from_crs(4326,crs_NORA,always_xy=True)
# Compute projected coordinates of lat/lon point
# Coordinates for OWS chosen earlyer
lat = latitude #N
lon = longitude #E
X,Y = proj.transform(lon,lat)

# Find nearest neighbour
x = ncfile.variables["x"][:]
y = ncfile.variables["y"][:]

Ix = np.argmin(np.abs(x - X)) # Nearest neighbour to LON
Iy = np.argmin(np.abs(y - Y)) # Nearest neighbour to LAT

ncfile.close()
```

In [ ]:

```
# to check for all available variables run the code below

filename = "https://thredds.met.no/thredds/dodsC/nora3_subset_atmos/wind_hourly/arom
ncfile   = nc.Dataset(filename)

for variable in ncfile.variables:
    print(variable)

ncfile.close()
```

Collecting the data

In this section you can choose "start date" and "end date" for the timeperiod of data to be extracted.

This dataset only contains an hourly update of wind speed and direction in cardinal directions at given heights

- Change start_date and end_date
- Change z_i to match desired height in meters above surface

collecting time: wind speed and wind direction hourly data for 12 months approx: 30 sec.

In [7]:

```
################################################
"""
Choose start date and end date (change only year and month)
"""

start_date = date(2022, 1, 1) # choose start date (year, month)
end_date = date(2022, 12, 31) # choose end date (year, month),  PS: includes the ful

"""
choose the height of the extraced data:
0 = 10m, 1 = 20m, 2 = 50m, 3 = 100m, 4 = 250m, 5 = 500m, 6 = 750m
"""

z_i = 2 # The same as 50 m.a.s.l

################################################



#Create empty lists for each variable to be extracted

time = []
wind_speed=[]
wind_direction = []


while start_date <= end_date:

    year = start_date.strftime("%Y")
    month = start_date.strftime("%m")


    opendap_url = f"https://thredds.met.no/thredds/dodsC/nora3_subset_atmos/wind_hou
    ncfile = nc.Dataset(opendap_url)

    times = ncfile.variables["time"][:]
    speed = ncfile.variables["wind_speed"][:, z_i, Iy, Ix]
    direction = ncfile.variables["wind_direction"][:, z_i, Iy, Ix]

    time.extend(times)
    wind_speed.extend(speed)
    wind_direction.extend(direction)



    ncfile.close()

    if month == '12':
            start_date = start_date.replace(year=start_date.year + 1, month=1)
    else:
        start_date = start_date.replace(month=start_date.month + 1)

```

Dictionary

The output from gathering the chosen data is stored in lists. This section creates an dictionary and uses pandas to store in a easy-reference system

- No changes needed

In [8]:

```
# create a dictionary and give name to the lists
d_data = {
     "time" : time,
     "wind_speed": wind_speed,
     "wind_direction": wind_direction,
}

# convert to pandas dataframe
weather_data = pd.DataFrame.from_dict(d_data)
weather_data = weather_data.set_index('time')

display(weather_data)
```

time	wind_speed	wind_direction
1.640995e+09	11.12	351.3
1.640999e+09	10.81	342.6
1.641002e+09	12.42	347.2
1.641006e+09	8.62	350.6
1.641010e+09	9.62	342.7
...	...	...
1.672513e+09	14.32	141.6
1.672517e+09	13.48	143.2
1.672520e+09	14.80	151.1
1.672524e+09	12.89	152.8
1.672528e+09	11.34	154.9

8760 rows × 2 columns

Save as NC-file

Define a function that saves the "weather_data" as an nc-file.

- Change the output_filename to desired new saved filename

Filesize for time, wind speed and wind direction for 12 months: 213 KB

In [ ]:

```
"""
This code is an easy option saving the new file directly in the same directory as th
"""

def save_dataframe_to_netcdf(dataframe, output_file):
    dataset = xr.Dataset(data_vars=dataframe.to_dict('series'))
    dataset.to_netcdf(output_file)

    print(f"Data saved successfully to {output_file}.")

output_filename = 'weather_data_NORA3.nc' #choose filename: default: "weather_data_N
save_dataframe_to_netcdf(weather_data, output_filename)
```

In [9]:

```
"""
This code makes a new optional directory at a desired location and saves the new fil
If the directory already exist it will just save the new file in that directory. Thi
in the same directory by keeping the same directory name and changing only the outpu
"""

import os

# New directory
directory = "weather_data"

# Parent Directory path
parent_dir = "C:/Users/Ida/"

# Filename new ncfile
# {site_name} or {station_id}
output_filename = f'NORA3_wind_weather_data_{station_id}.nc' #choose filename


# Path
path = os.path.join(parent_dir, directory)

# Create the directory if not already existing
if not os.path.isdir(path):
    os.mkdir(path)


def save_dataframe_to_netcdf(dataframe, output_file):
    dataset = xr.Dataset(data_vars=dataframe.to_dict('series'))
    output_path = f"{parent_dir}{directory}/{output_file}"
    dataset.to_netcdf(output_path)

    print(f"Data saved successfully to {output_path}.")


save_dataframe_to_netcdf(weather_data, output_filename)
```

```
Data saved successfully to C:/Users/Ida/weather_data_2022_Goliat/NORA3_win
d_50m_weather_data_GOLIAT_2022.nc.
```

Examples on how to use the new NC-file

This section shows some examples on how the new downloaded and saved ncfile can be used

In [10]:

```
# open the new file

filename = "C:/Users/Ida/weather_data_2022_Goliat/NORA3_wind_50m_weather_data_GOLIAT
ncfile = nc.Dataset(filename)

# File saved at another location

#filename = f"{path}/{output_filename}"
#ncfile = nc.Dataset(filename)

# print all variables in the file
for variable in ncfile.variables:
    print(variable)

```

```
time
wind_speed
wind_direction
```

In [11]:

```
# covert timestamps into nice readable dates

time = ncfile.variables["time"][:]

ts = np.linspace(0,len(time)-1,len(time))

nice_time = []


for i in range(0,len(ts)):
    t = dt.datetime.utcfromtimestamp(time[i])
    nice_time.append(t)

print(nice_time[0], nice_time[-1]) # check that start and end date/time is correct
```

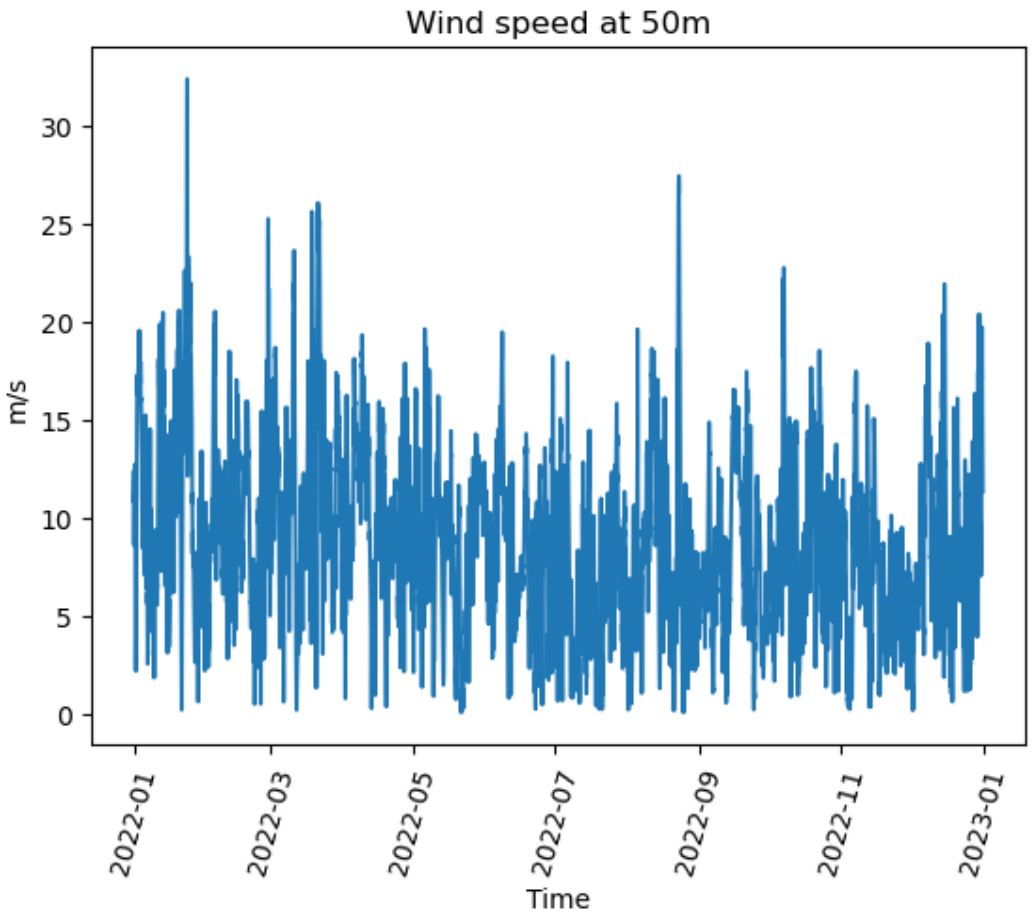
```
2022-01-01 00:00:00 2022-12-31 23:00:00
```

In [12]:

```
# extract chosen variables
ws = ncfile.variables["wind_speed"][:]
direction = ncfile.variables["wind_direction"][:]

```

In [13]:

```
plt.plot(nice_time, ws)
plt.title("Wind speed at 50m")
plt.ylabel("m/s")
plt.xlabel("Time")

plt.tick_params(axis='x', labelcolor='black', labelrotation=75, labelsize=10)
plt.show()
```



Histogram

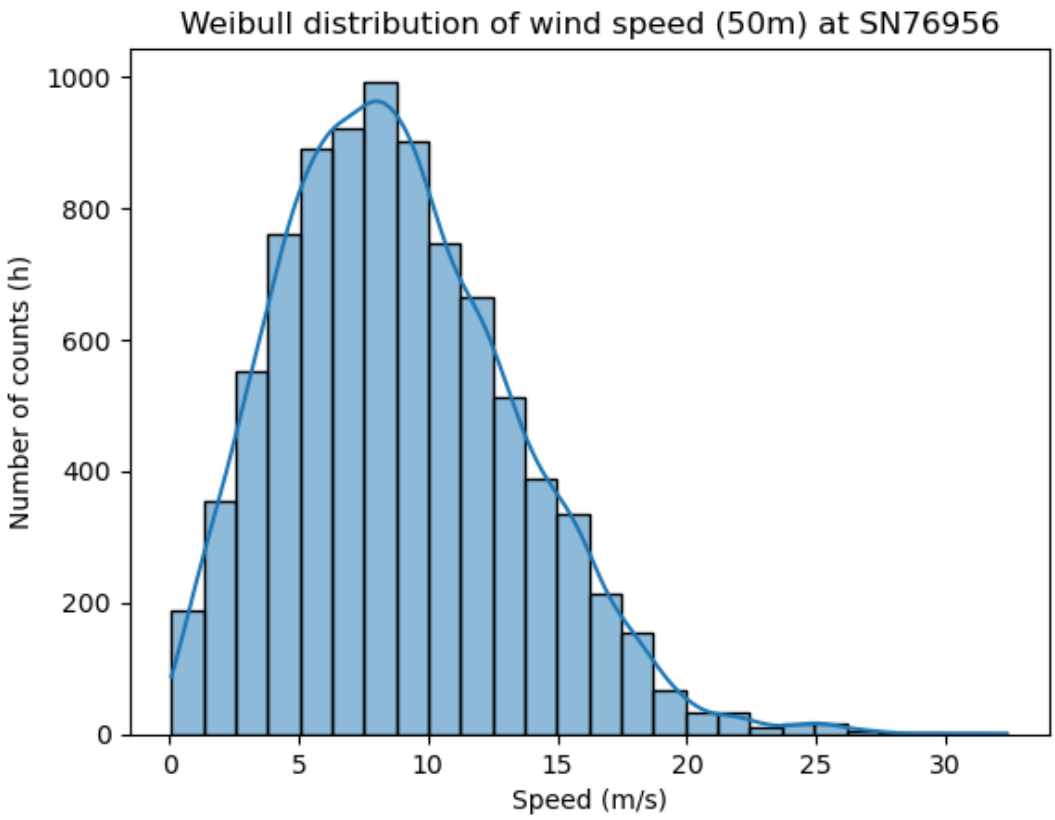
This is an example on how to make a histogram from the wind speeds

In [14]:

```
#install the seaborn package (if not already installed) by removing the "#" and runn
#!pip3 install seaborn

import seaborn as sns

sns.histplot(data = ncfile, x = ws, kde = True, bins = 26)
plt.xlabel("Speed (m/s)")
plt.ylabel("Number of counts (h)")
plt.title(f"Weibull distribution of wind speed (50m) at {station_id}")

```

Out[14]:

```
Text(0.5, 1.0, 'Weibull distribution of wind speed (50m) at SN76956')
```

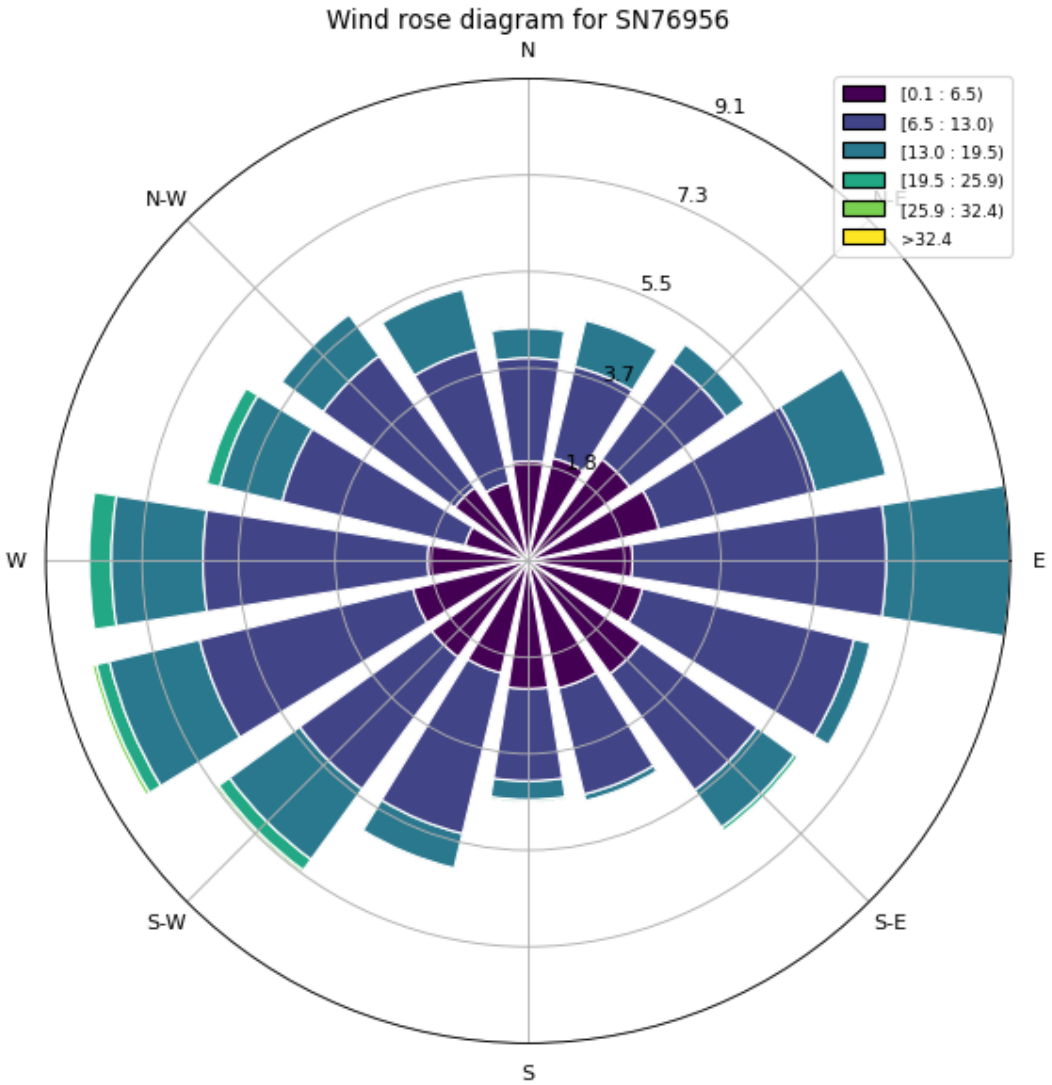


Windrose

the following example show how to make a windrose at the chosen site and timespan. The windrose illustrates the distribution of wind speed and direction together. The direction in degrees is transformed into cardinal directions, 360 - N, 180 - S, 90 - E, 270 - W

In [15]:

```
# run the code below if not already installed the windrose package
#!pip3 install windrose

from windrose import WindroseAxes

speed = ncfile.variables["wind_speed"][:]
direction = ncfile.variables["wind_direction"][:]

ax = WindroseAxes.from_ax()
ax.bar(direction,
       speed,
       normed=True, #get % of number of hours
       opening= 0.8, #width of bars
       edgecolor='white')
ax.set_legend(loc = "best")
plt.title(f"Wind rose diagram for {station_id}")
plt.show()

```



6.2.2 Download NORA3 atoms

Download data from NORA3 and save to ncfile

The following codes is a guide to download data from the NWP NORA3 extracted at one location with coordinates (Lat/Lon) for a chosen time period and save it as an nc-file, available at https://thredds.met.no/thredds/catalog/nora3_subset_atmos/atm_hourly/catalog.html (https://thredds.met.no/thredds/catalog/nora3_subset_atmos/atm_hourly/catalog.html). This script use the subset atmosphere files. Some variables:

- wind_speed and wind_direction at height 10 meters above surface
- Air temperature at 2 meters above surface
- surface air pressure.

choose a datespan (years, months) and save it as a ncfile.

Data available from: 1985.01 to (current month minus 6-7 months time lag), check link for latest date

Model runs 4 times a day, every 6h, the files contains hourly updated data based on these runs

In [28]:

```
# Importing necessary modules

import xarray as xr
import numpy as np

import netCDF4 as nc
import matplotlib.pylab as plt
import pandas as pd

import pyproj
import datetime as dt
from datetime import date, timedelta

import requests
from __future__ import print_function
from IPython.display import display, HTML
```

In [29]:

```
"""
The coordinates in get_coordinates is at the center of the offshore wind sites
"""

def get_coordinates_center_OWS(location_name):
    locations = {
        'Nordavind_A': {
            'latitude': 71.1314956,
            'longitude': 32.048109
        },
        'Nordavind_B': {
            'latitude': 71.7880587,
            'longitude': 27.7221338
        },
        'Nordavind_C': {
            'latitude': 71.7471898,
            'longitude': 19.9808019
        },
        'Nordavind_D': {
            'latitude': 71.473272,
            'longitude': 18.7614613
        },
        'Utsira_nord': {
            'latitude': 59.2740869,
            'longitude': 4.5441279
        },
        'Sørlige_nordsjø_2': {
            'latitude': 59.7894747,
            'longitude': 4.9314619
        }
    }

    if location_name in locations:
        coordinates = locations[location_name]
        return coordinates['latitude'], coordinates['longitude']
    else:
        return None


```

Seklima station information retrival with frost api

Link to frost webpage (https://frost.met.no/index.html)

1. Create client id for yourself
   - Create client id here (https://frost.met.no/auth/requestCredentials.html)

This is used to retrive the longitude and latitude

In [30]:

```
# Insert your own client ID here
client_id = 'insert client ID'
```

In [31]:

```
'''
Getting the data for the meta dataframe

Using the frost api for seklima
'''

endpoint = 'https://frost.met.no/sources/v0.jsonld'

# Dictionary with elements to retrieve
parameters = {
    'fields': 'name,id,geometry,masl,validFrom'
}

# Issue an HTTP GET request
r = requests.get(endpoint, parameters, auth=(client_id,''))
# Extract JSON data
json = r.json()

# Check if the request worked, print out any errors
if r.status_code == 200:
    data_exp = json['data']
    print('Data retrieved from frost.met.no!')
else:
    print('Error! Returned status code %s' % r.status_code)
    print('Message: %s' % json['error']['message'])
    print('Reason: %s' % json['error']['reason'])
```

```
Data retrieved from frost.met.no!
```

In [32]:

```
'''
Transforming the data from a raw json format retrieved by requests
to a pandas df format

When redefining the query the new columns need to be spesified for the dataframe df
'''

meta_df = pd.DataFrame(columns=['id', 'name', 'lon', 'lat', 'heigh-asl (m)','operati
ignored_values = 0
for i in range(len(data_exp)):
    row = []
    try:
        row.append(data_exp[i]['id'])
        row.append(data_exp[i]['name'])
        row.append(data_exp[i]['geometry']['coordinates'][0])
        row.append(data_exp[i]['geometry']['coordinates'][1])
        row.append(data_exp[i]['masl'])
        row.append(data_exp[i]['validFrom'])
        meta_df.loc[len(meta_df)] = row
    except:
        ignored_values += 1
        continue


print(f'Number of discarded values {ignored_values}')

# Setting the station id as row index2
meta_df = meta_df.set_index('id')

display(meta_df)
```

```
Number of discarded values 88
```

id	name	lon	lat	heigh-asl (m)
SN47230	ÅKRA UNGDOMSSKOLE	5.196300	59.255500	18
SN23670	E16 RYFOSS	8.817500	61.137500	406
SN59450	STADLANDET	5.211500	62.146700	75
SN12590	E6 MJØSBRUA	10.672500	60.928000	128
SN26640	E134 DARBU	9.777300	59.702500	155
...	...	...	...	...
SN74780	NAMSVATN II	13.540300	64.960800	460
SN1292500	NAGYKANIZSA	16.970556	46.456111	139
SN1712400	ESKISEHIR HAVALIMANI	30.579700	39.781000	786
SN94805	E69 VEDBOTN	25.629700	70.776500	10
SN248690	SÖDERTÄLJE	17.628900	59.214200	23

1921 rows × 4 columns

- Change station ID to match measurement station to get station coordinates
  - Find station ID at (stations (https://seklima.met.no/stations/))

or

- Insert site_name to get coordinates from center of OWS

In [33]:

```
##################


station_id = "SN76956" # insert station ID, ex: 'SN76956' Goliat ,  'SN20926' hjelms


#################

longitude = meta_df.loc[f"{station_id}"]["lon"]
latitude = meta_df.loc[f"{station_id}"]["lat"]

print(f"{station_id}: longitude = {longitude}, latitude = {latitude}")
```

```
SN76956: longitude = 22.25, latitude = 71.3112
```

In [ ]:

```
# insert the site name you want the data extracted from

#################

site_name = 'Nordavind_C'

#################

latitude, longitude = get_coordinates_center_OWS(site_name) # gets the latitude and
print(f"{site_name}: longitude = {longitude}, latitude = {latitude}")
```

In [ ]:

```
# Or choose a spesific latitude, longitude

latitude =
longitude =
```

Projecting coordinates

Starts by projecting the chosen Latitude and Longitude coordinate to x and y cooridnates used in the model

- No changes needed in this part

In [34]:

```
"""
load in a random file from the NORA3 dataset, this is just used to project latitude/
x- and y-coordinate
"""

filename = "https://thredds.met.no/thredds/dodsC/nora3_subset_atmos/atm_hourly/arome
ncfile   = nc.Dataset(filename)



#######
# Projection
crs_NORA = pyproj.CRS.from_cf(
        {
           "grid_mapping_name": "lambert_conformal_conic",
           "standard_parallel": [66.3, 66.3],
           "longitude_of_central_meridian": -42.0,
           "latitude_of_projection_origin": 66.3,
            "earth_radius": 6371000.0,
        }
)

# Transformer to project from ESPG:4368 (WGS:84) to our lambert_conformal_conic
proj = pyproj.Transformer.from_crs(4326,crs_NORA,always_xy=True)
# Compute projected coordinates of lat/lon point
# Coordinates for OWS chosen earlyer
lat = latitude # N
lon = longitude # E
X,Y = proj.transform(lon,lat)

# Find nearest neighbour
x = ncfile.variables["x"][:]
y = ncfile.variables["y"][:]

Ix = np.argmin(np.abs(x - X)) # Nearest neighbour to LON
Iy = np.argmin(np.abs(y - Y)) # Nearest neighbour to LAT

ncfile.close()
```

In [ ]:

```
# to check for all available variables run the for-loop below

filename = "https://thredds.met.no/thredds/dodsC/nora3_subset_atmos/atm_hourly/arome
ncfile   = nc.Dataset(filename)

for variable in ncfile.variables:
    print(variable)

ncfile.close()
```

Collecting the data

In this section you can choose "start date" and "end date" for the timeperiod of data to be extracted.

This dataset contains an hourly update of wind speed and direction in cardinal directions at given heights
Variables:

- wind_speed and wind_direction at height 10 meters above surface
- Air temperature at 2 meters above surface
- surface air pressure.

- Change start_date and end_date

In [35]:

```
################################################
"""
Choose start date and end date (change only year and month)
"""

start_date = date(2021, 1, 1) # choose start date (year, month)
end_date = date(2021, 12, 1) # choose end date (year, month),  PS: includes the full


################################################


#Create empty lists for each variable to be extracted

time = []
wind_speed=[]
wind_direction = []
air_temp_2m = []
surface_pressure = []


while start_date <= end_date:

    year = start_date.strftime("%Y")
    month = start_date.strftime("%m")


    opendap_url = f"https://thredds.met.no/thredds/dodsC/nora3_subset_atmos/atm_hour
    ncfile = nc.Dataset(opendap_url)

    times = ncfile.variables["time"][:]
    speed = ncfile.variables["wind_speed"][:, 0, Iy, Ix]
    direction = ncfile.variables["wind_direction"][:, 0, Iy, Ix]
    temp = ncfile.variables["air_temperature_2m"][:, 0, Iy, Ix]
    press = ncfile.variables["air_pressure_at_sea_level"][:, 0, Iy, Ix]

    time.extend(times)
    wind_speed.extend(speed)
    wind_direction.extend(direction)
    air_temp_2m.extend(temp)
    surface_pressure.extend(press)

    ncfile.close()

    if month == '12':
            start_date = start_date.replace(year=start_date.year + 1, month=1)
    else:
        start_date = start_date.replace(month=start_date.month + 1)


```

Dictionary

The output from collecting data is stored in lists. This section creates an dictionary and uses pandas to store in a easy-reference system

- Changes can be made in the dictionary "d_data" if variables is ignored or added

In [36]:

```
# create a dictionary and give name to the lists
d_data = {
     "time" : time,
     "wind_speed": wind_speed,
     "wind_direction": wind_direction,
     "air_temperature_2m": air_temp_2m,
     "surface_pressure": surface_pressure
}

# convert to pandas dataframe
weather_data = pd.DataFrame.from_dict(d_data)
weather_data = weather_data.set_index('time')

display(weather_data)
```

time	wind_speed	wind_direction	air_temperature_2m	surface_pressure
1.609459e+09	9.684497	162.840393	275.670532	101637.0625
1.609463e+09	8.857800	162.552078	275.597168	101654.2500
1.609466e+09	8.077759	159.469681	275.534119	101659.1250
1.609470e+09	7.987354	156.853165	275.202850	101663.1875
1.609474e+09	6.451159	134.356888	275.390442	101696.6875
...	...	...	...	...
1.640977e+09	13.336639	5.686159	272.727692	100477.4375
1.640981e+09	11.898799	353.517883	271.726196	100537.7500
1.640984e+09	11.238021	2.237887	272.875122	100569.3125
1.640988e+09	10.014099	2.574722	273.093079	100617.6250
1.640992e+09	10.312613	0.746557	273.183350	100670.0000

8760 rows × 4 columns

Save as NC-file

Define a function that saves the "weather_data" as an nc-file.

- Change the output_filename to desired new saved filename

Filesize for time, wind speed and wind direction for 6 months: 123 KB

In [ ]:

```
"""
This code is an easy option saving the new file directly in the same directory as th
"""

def save_dataframe_to_netcdf(dataframe, output_file):
    dataset = xr.Dataset(data_vars=dataframe.to_dict('series'))
    dataset.to_netcdf(output_file)

    print(f"Data saved successfully to {output_file}.")



output_filename = f'{station_id}_weather_data_NORA3.nc' #choose filename: default: '

save_dataframe_to_netcdf(weather_data, output_filename)
```

In [37]:

```
"""
This code makes a new optional directory at a desired location and saves the new fil
If the directory already exist it will just save the new file in that directory. Thi
in the same directory by keeping the same directory name and changing only the outpu
"""

import os

# New directory
directory = "weather_data"

# Parent Directory path
parent_dir = "C:/Users/Ida/"

# Filename new ncfile
# {site_name} or {station_id}
output_filename = f'{station_id}_y22_atmos_weather_data_NORA3.nc' #choose filename


# Path
path = os.path.join(parent_dir, directory)

# Create the directory if not already existing
if not os.path.isdir(path):
    os.mkdir(path)


def save_dataframe_to_netcdf(dataframe, output_file):
    dataset = xr.Dataset(data_vars=dataframe.to_dict('series'))
    output_path = f"{parent_dir}{directory}/{output_file}"
    dataset.to_netcdf(output_path)

    print(f"Data saved successfully to {output_path}.")


save_dataframe_to_netcdf(weather_data, output_filename)
```

```
Data saved successfully to C:/Users/Ida/weather_data/SN76956_y22_atmos_wea
ther_data_NORA3.nc.
```

Examples on how to use the new NC-file

In [38]:

```
# open the new file

#filename = "weather_data_NORA3.nc" # the name of the new file, file in the working
#ncfile = nc.Dataset(filename)

# File saved at another location

filename = f"{path}/{output_filename}"
ncfile = nc.Dataset(filename)


# print all variables in the file
for variable in ncfile.variables:
    print(variable)

```

```
time
wind_speed
wind_direction
air_temperature_2m
surface_pressure
```

In [40]:

```
# covert timestamps into nice readable dates

time = ncfile.variables["time"][:]

ts = np.linspace(0,len(time)-1,len(time))

nice_time = []


for i in range(0,len(ts)):
    t = dt.datetime.utcfromtimestamp(time[i])
    nice_time.append(t)

print(nice_time[0], nice_time[-1]) # check that start and end date/time is correct

```

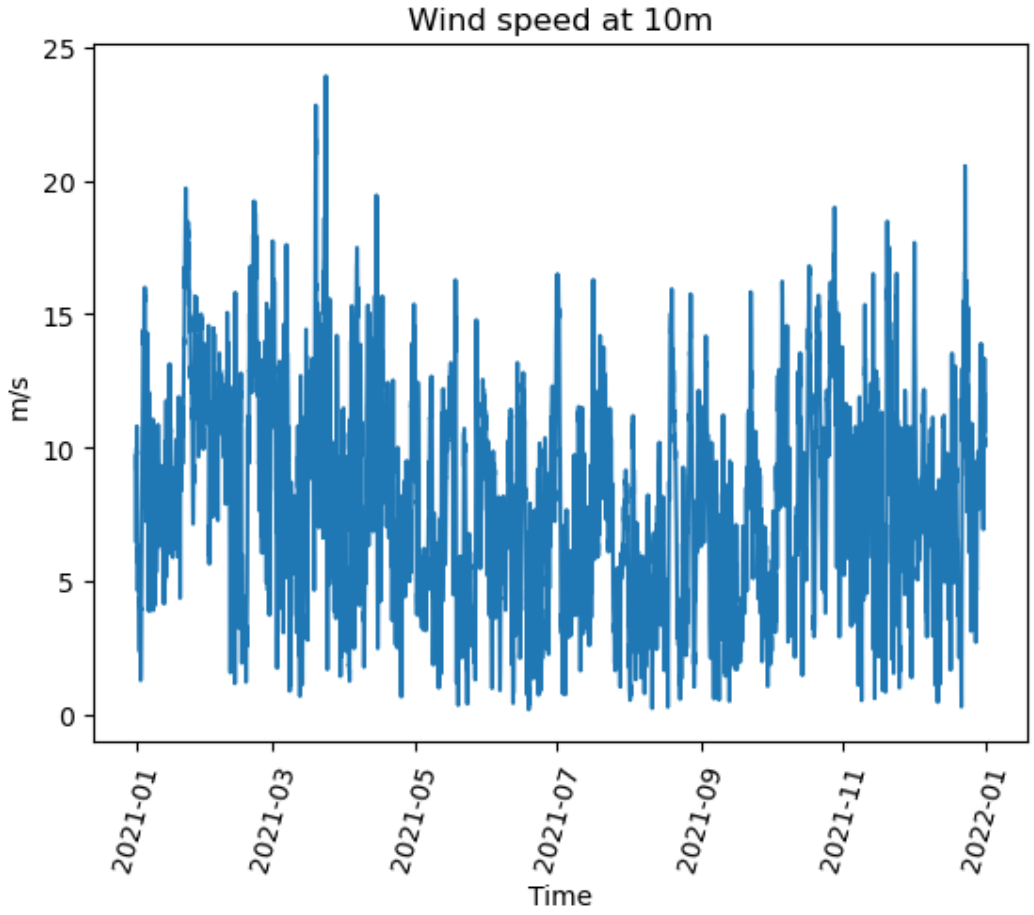
```
2021-01-01 00:00:00 2021-12-31 23:00:00
```

In [41]:

```
# extract variables
ws = ncfile.variables["wind_speed"][:]
direction = ncfile.variables["wind_direction"][:]
Temp = ncfile.variables["air_temperature_2m"][:]
press = ncfile.variables["surface_pressure"][:]

```

In [42]:

```
plt.plot(nice_time, ws)
plt.title("Wind speed at 10m")
plt.ylabel("m/s")
plt.xlabel("Time")

plt.tick_params(axis='x', labelcolor='black', labelrotation=75, labelsize=10)
plt.show()
```



Histogram

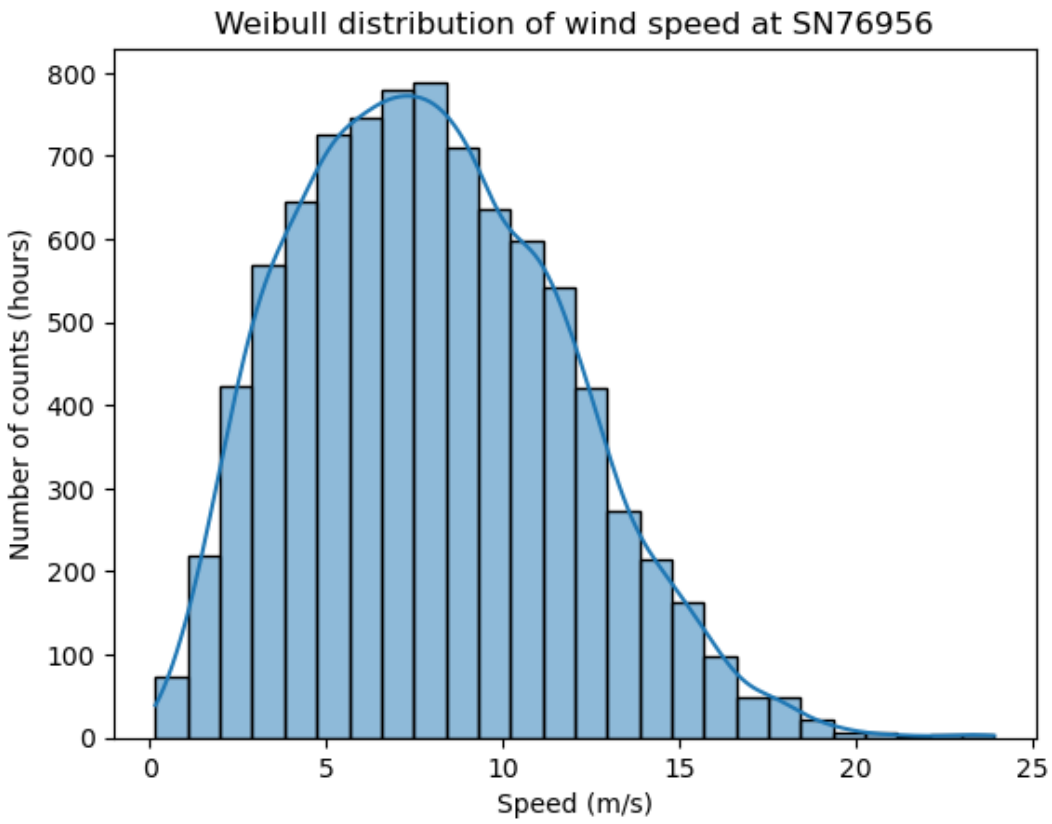
This is an example on how to make a histogram from the wind speeds

In [45]:

```
#install the seaborn package by removing the "#" and running the code below
#!pip3 install seaborn

import seaborn as sns

sns.histplot(data = ncfile, x = ws, kde = True, bins = 26)
plt.xlabel("Speed (m/s)")
plt.ylabel("Number of counts (hours)")
plt.title(f"Weibull distribution of wind speed at {station_id}")

```

Out[45]:

```
Text(0.5, 1.0, 'Weibull distribution of wind speed at SN76956')
```

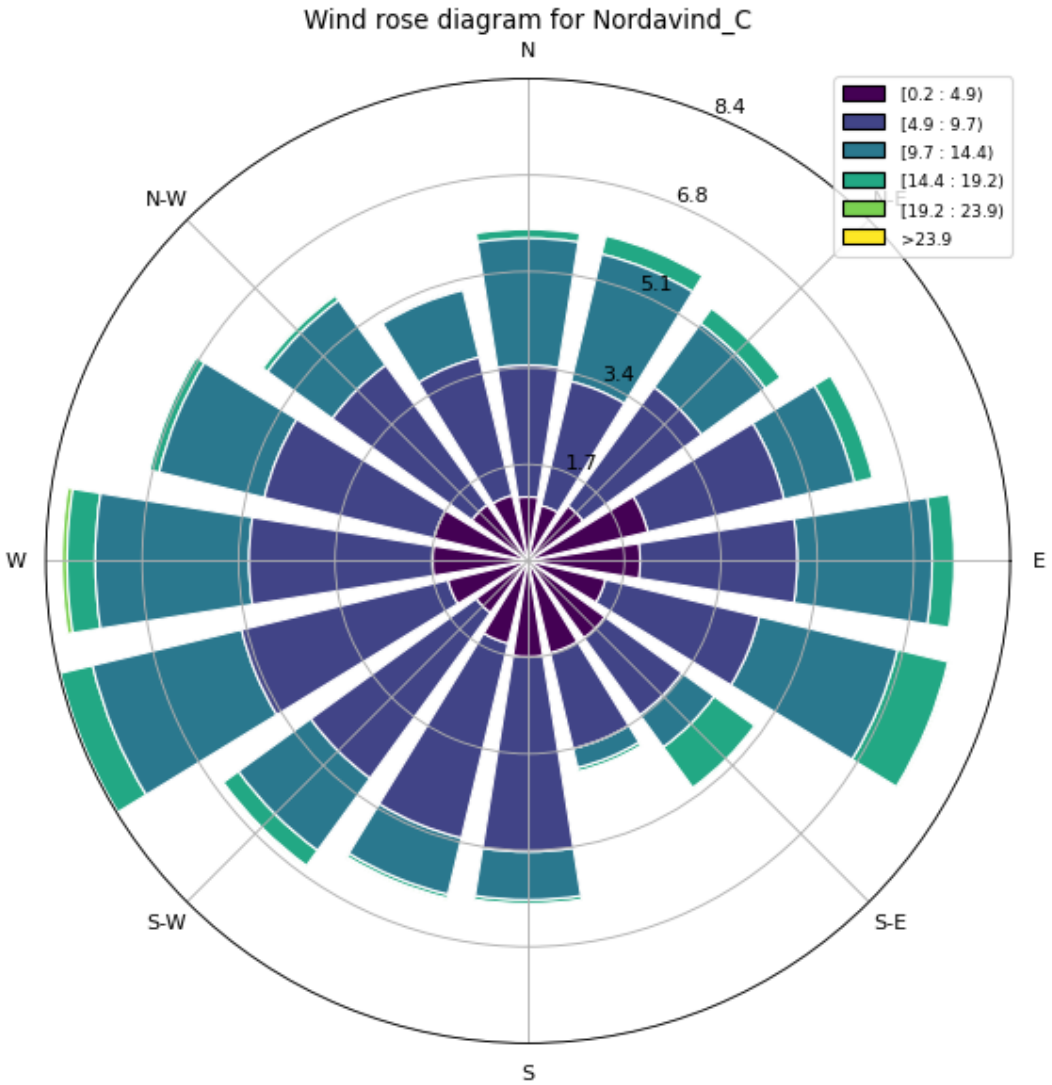


Windrose

the following example show how to make a windrose at the chosen site and timespan. The windrose illustrates the distribution of wind speed and direction together. The direction in degrees is transformed into cardinal directions, 360 = N, 180 = S, 90 = E, 270 = W

In [46]:

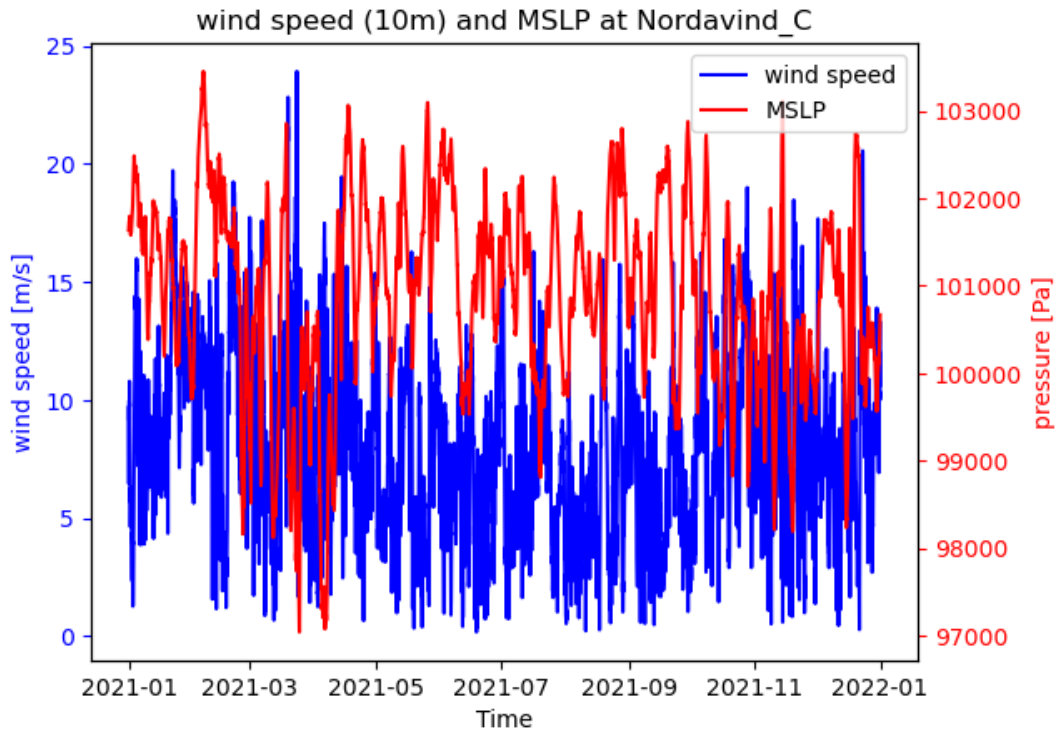
```
# run the code below if not already installed the windrose package
#!pip3 install windrose

from windrose import WindroseAxes

speed = ncfile.variables["wind_speed"][:]
direction = ncfile.variables["wind_direction"][:]

ax = WindroseAxes.from_ax()
ax.bar(direction,
       speed,
       normed=True, #get % of number of hours
       opening= 0.8, #width of bars
       edgecolor='white')
ax.set_legend(loc = "best")
plt.title(f"Wind rose diagram for {site_name}")
plt.show()

```



Plot with two variables

In [47]:

```
# plot med 2 x-akser med samme y-verider
fig, ax1 = plt.subplots()


ax1.plot(nice_time, ws, color="b", label="wind speed") # first plot

ax2 = ax1.twinx() # second plot with same x-value (time)
ax2.plot(nice_time, press, color="r", label="MSLP")


#Label
ax1.set_ylabel("wind speed [m/s]", color="b")
ax2.set_ylabel("pressure [Pa]", color="r")

ax1.set_xlabel("Time")

# color the y-axes
ax1.tick_params(axis='y', colors="b")
ax2.tick_params(axis='y', colors="r")

fig.legend(loc="upper right", bbox_to_anchor=(1,1), bbox_transform=ax1.transAxes)

plt.tick_params(axis='x', labelcolor='black', labelrotation=75, labelsize=10)

plt.title(f"wind speed (10m) and MSLP at {site_name}")

plt.show()
```



6.2.3 Download NORA3 fpnc

Download data from NORA3 and save to ncfile

The following codes is a guide to download data from the NWP NORA3 at a chosen time period and save it as an nc-file. Data available at https://thredds.met.no/thredds/projects/nora3.html (https://thredds.met.no/thredds/projects/nora3.html). This script use the files ending with fp.nc

- Several variables available.
- Containing wind_speed and wind_direction at 10m data from chosen dates and save it to an ncfile.

choose a datespan (years, months, day) and save it as a ncfile.

Data available from: 1969.08.01 to current date, check link for latest date

Model runs 4 times a day, every 6h, the files contains hourly updated data based on these runs

In [47]:

```
# Importing necessary modules

import xarray as xr
import numpy as np

import netCDF4 as nc
import matplotlib.pylab as plt
import pandas as pd

import pyproj
import datetime as dt
from datetime import date, timedelta
```

In [48]:

```
"""
The coordinates in get_coordinates is at the center of the offshore wind sites
"""

def get_coordinates_center_OWS(location_name):
    locations = {
        'Nordavind_A': {
            'latitude': 71.1314956,
            'longitude': 32.048109
        },
        'Nordavind_B': {
            'latitude': 71.7880587,
            'longitude': 27.7221338
        },
        'Nordavind_C': {
            'latitude': 71.7471898,
            'longitude': 19.9808019
        },
        'Nordavind_D': {
            'latitude': 71.473272,
            'longitude': 18.7614613
        },
        'Utsira_nord': {
            'latitude': 59.2740869,
            'longitude': 4.5441279
        },
        'Sørlige_nordsjø_2': {
            'latitude': 59.7894747,
            'longitude': 4.9314619
        }
    }

    if location_name in locations:
        coordinates = locations[location_name]
        return coordinates['latitude'], coordinates['longitude']
    else:
        return None

```

Seklima station information retrival with frost api

Link to frost webpage (https://frost.met.no/index.html)

1. Create client id for yourself
   - Create client id here (https://frost.met.no/auth/requestCredentials.html)

This is used to retrive the longitude and latitude

In [ ]:

```
# Insert your own client ID here
client_id = 'insert client ID'
```

In [ ]:

```
'''
Getting the data for the meta dataframe

Using the frost api for seklima
'''

endpoint = 'https://frost.met.no/sources/v0.jsonld'

# Dictionary with elements to retrieve
parameters = {
    'fields': 'name,id,geometry,masl,validFrom'
}

# Issue an HTTP GET request
r = requests.get(endpoint, parameters, auth=(client_id,''))
# Extract JSON data
json = r.json()

# Check if the request worked, print out any errors
if r.status_code == 200:
    data_exp = json['data']
    print('Data retrieved from frost.met.no!')
else:
    print('Error! Returned status code %s' % r.status_code)
    print('Message: %s' % json['error']['message'])
    print('Reason: %s' % json['error']['reason'])
```

In [ ]:

```
'''
Transforming the data from a raw json format retrieved by requests
to a pandas df format

When redefining the query the new columns need to be spesified for the dataframe df
'''

meta_df = pd.DataFrame(columns=['id', 'name', 'lon', 'lat', 'heigh-asl (m)','operati
ignored_values = 0
for i in range(len(data_exp)):
    row = []
    try:
        row.append(data_exp[i]['id'])
        row.append(data_exp[i]['name'])
        row.append(data_exp[i]['geometry']['coordinates'][0])
        row.append(data_exp[i]['geometry']['coordinates'][1])
        row.append(data_exp[i]['masl'])
        row.append(data_exp[i]['validFrom'])
        meta_df.loc[len(meta_df)] = row
    except:
        ignored_values += 1
        continue


print(f'Number of discarded values {ignored_values}')

# Setting the station id as row index2
meta_df = meta_df.set_index('id')

display(meta_df)
```

- Change station ID to match measurement station to get station coordinates
  - Find station ID at (stations (https://seklima.met.no/stations/))

or

- Insert site_name to get coordinates from center of OWS

In [ ]:

```

#################


station_id = "SN76956" # insert station ID, ex: 'SN76956' Goliat ,  'SN20926' hjelms


################

longitude = meta_df.loc[f"{station_id}"]["lon"]
latitude = meta_df.loc[f"{station_id}"]["lat"]

print(f"{station_id}: longitude = {longitude}, latitude = {latitude}")
```

In [49]:

```
# insert the site name you want the data extracted from

#################

site_name = 'Nordavind_C'

#################

latitude, longitude = get_coordinates_center_OWS(site_name) # gets the latitude and
print(f"{site_name}: longitude = {longitude}, latitude = {latitude}")
```

In [ ]:

```
# Choose latitude, longitude

latitude =
longitude =
```

Projecting coordinates

Starts by projecting the chosen Latitude and Longitude coordinate to x and y cooridnates used in the model

- No changes needed in this part

In [50]:

```
"""
load in a random file from the NORA3 dataset, this is just used to project latitude/
x- and y-coordinate
"""
year =  "2022"
month = "01"
day = "01"
hour = "00"


filename = f"https://thredds.met.no/thredds/dodsC/nora3/{year}/{month}/{day}/{hour}/
ncfile   = nc.Dataset(filename)


#######
# Projection
crs_NORA = pyproj.CRS.from_cf(
        {
           "grid_mapping_name": "lambert_conformal_conic",
           "standard_parallel": [66.3, 66.3],
           "longitude_of_central_meridian": -42.0,
           "latitude_of_projection_origin": 66.3,
            "earth_radius": 6371000.0,
        }
)

# Transformer to project from ESPG:4368 (WGS:84) to our lambert_conformal_conic
proj = pyproj.Transformer.from_crs(4326,crs_NORA,always_xy=True)
# Compute projected coordinates of lat/lon point
# Coordinates for OWS chosen earlyer
lat = latitude #N
lon = longitude #E
X,Y = proj.transform(lon,lat)

# Find nearest neighbour
x = ncfile.variables["x"][:]
y = ncfile.variables["y"][:]

Ix = np.argmin(np.abs(x - X)) # Nearest neighbour to LON
Iy = np.argmin(np.abs(y - Y)) # Nearest neighbour to LAT

ncfile.close()
```

In [ ]:

```

filename = f"https://thredds.met.no/thredds/dodsC/nora3/{year}/{month}/{day}/{hour}/
ncfile   = nc.Dataset(filename)


# Check variables in file
for variable in ncfile.variables:
    print(variable)

ncfile.close()
```

Collecting the data

In this section you can choose "start date" and "end date" for the timeperiod of data to be extracted. Different variables can also be left out or added before running the code. The dataset "fp.nc" contains temperature at 2m, wind speed and direction at 10m and surface air pressure. (run the for-loop in cell 2, to see all available variables)

collecting time, wind speed (10m) and wind direction (10m) hourly data for 6 months approx: 55 min.

In [51]:

```
"""
Choose start and end date
"""
start_date = date(2019, 1, 1) # choose start date (year, month, day)
end_date = date(2019, 1, 2) # choose end date (year, month, day)


delta = timedelta(days=1) # delta (timestep), initilly set to 1 day


"""
Create empty lists for each variable to be extracted
"""
time = []
air_temp_2m=[]
wind_speed = [] # only at 10m
wind_direction = [] # only at 10m
surface_air_pressure = []

x_wind_z = []
y_wind_z = []

z = 0 # chose index for x- and y-wind height, 0 = 20m, 1 = 50m, 2 = 100m, 3 = 250m,



hours = ["00", "06", "12", "18"] # each day have one file with timestamp
leadtime = ["003","004","005","006","007","008"] #"009" if run with 009, this is the

while start_date <= end_date:

    y = start_date.strftime("%Y")
    m = start_date.strftime("%m")
    d = start_date.strftime("%d")
    for i1 in hours:
        for i2 in leadtime:
            opendap_url = f"https://thredds.met.no/thredds/dodsC/nora3/{y}/{m}/{d}/{
            ncfile = nc.Dataset(opendap_url)

            times = ncfile.variables["time"][:] # only one timestamp for each file
            airtemp2m = ncfile.variables["air_temperature_2m"][0, 0, Iy, Ix] #[time(
            speed = ncfile.variables["wind_speed"][0, 0, Iy, Ix]
            direction = ncfile.variables["wind_direction"][0, 0, Iy, Ix]
            sur_p = ncfile.variables["surface_air_pressure"][0, 0, Iy, Ix]

            # xwind_z = ncfile.variables["x_wind_z"][0, z, Iy, Ix]
            # ywind_z = ncfile.variables["y_wind_z"][0, z, Iy, Ix]

            time.append(times)
            air_temp_2m.append(airtemp2m)
            wind_speed.append(speed)
            wind_direction.append(direction)
            surface_air_pressure.append(sur_p)

            # x_wind_z.append(xwind_z)
            # y_wind_z.append(ywind_z)

            ncfile.close()

```

```
60      start_date += delta
```

Dictionary

The output from gathering the chosen data is stored in lists. This section creates an dictionary and uses pandas to store in a easy-reference system

- Changes can be made in the dictionary "d_data" if variables is ignored or added

In [52]:

```
# create a dictionary and give name to the lists
d_data = {
     "time" : time,
     "air_temperature_2m": air_temp_2m,
     "surface_pressure": surface_air_pressure,
     "wind_speed": wind_speed,
     "wind_direction": wind_direction,
    # "x_wind_z": x_wind_z,
    # "y_wind_z": y_wind_z
}

# convert to pandas dataframe
import pandas as pd

weather_data = pd.DataFrame.from_dict(d_data)
#print(weather_data)
```

Save as NC-file

Define a function that saves the "weather_data" as an nc-file.

- Change the output_filename to desired new saved filename

Filesize for time, wind speed and wind direction for 6 months: 93 KB

In [ ]:

```
"""
This code is an easy option saving the new file directly in the same directory as th
"""

def save_dataframe_to_netcdf(dataframe, output_file):
    dataset = xr.Dataset(data_vars=dataframe.to_dict('series'))
    dataset.to_netcdf(output_file)

    print(f"Data saved successfully to {output_file}.")

output_filename = 'weather_data_NORA3.nc' #choose filename: default: "weather_data_N
save_dataframe_to_netcdf(weather_data, output_filename)
```

In [ ]:

```
"""
This code makes a new optional directory at a desired location and saves the new fil
If the directory already exist it will just save the new file in that directory. Thi
in the same directory by keeping the same directory name and changing only the outpu
"""

import os

# New directory
directory = "weather_data"

# Parent Directory path
parent_dir = "C:/Users/Ida/"

# Filename new ncfile
# {site_name} or {station_id}
output_filename = f'{station_id}_full_weather_data_NORA3.nc' # choose filename


# Path
path = os.path.join(parent_dir, directory)

# Create the directory if not already existing
if not os.path.isdir(path):
    os.mkdir(path)


def save_dataframe_to_netcdf(dataframe, output_file):
    dataset = xr.Dataset(data_vars=dataframe.to_dict('series'))
    output_path = f"{parent_dir}{directory}/{output_file}"
    dataset.to_netcdf(output_path)

    print(f"Data saved successfully to {output_path}.")


save_dataframe_to_netcdf(weather_data, output_filename)
```

Examples on how to use the new NC-file

In [54]:

```
# open the new file

filename = "weather_data_NORA3.nc"
ncfile = nc.Dataset(filename)


# print all variables in the file
for variable in ncfile.variables:
    print(variable)

```

```
dim_0
time
wind_speed
wind_direction
```

In [ ]:

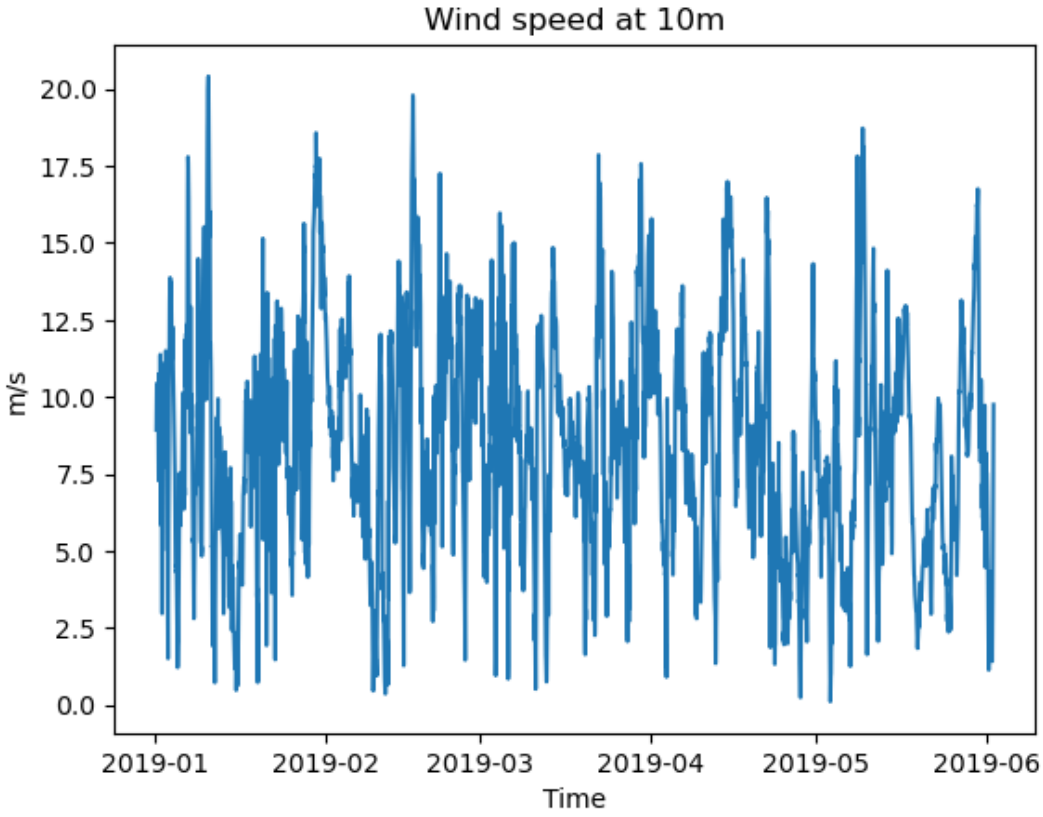
```
# covert timestamps into nice readable dates

time = ncfile.variables["time"][:]

ts = np.linspace(0,len(time)-1,len(time))

nice_time = []


for i in range(0,len(ts)):
    tid = dt.datetime.utcfromtimestamp(time[i])
    nice_time.append(tid)

```

In [56]:

```
# extract variables
ws = ncfile.variables["wind_speed"][:]
direction = ncfile.variables["wind_direction"][:]


```

In [57]:

```
plt.plot(nice_time, ws)
plt.title("Wind speed at 10m")
plt.ylabel("m/s")
plt.xlabel("Time")

plt.tick_params(axis='x', labelcolor='black', labelrotation=75, labelsize=10)
plt.show()
```



Histogram

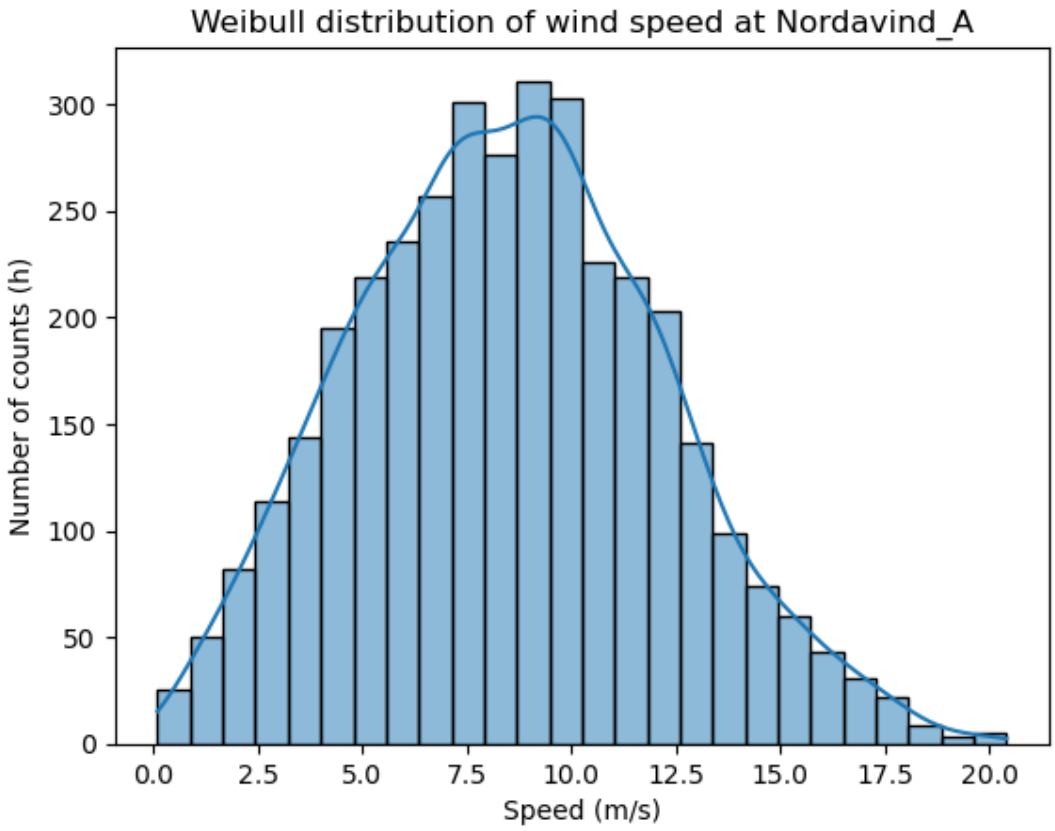
This is an example on how to make a histogram from the wind speeds

In [60]:

```
#install the seaborn package by removing the "#" and running the code below
#!pip3 install seaborn

import seaborn as sns

sns.histplot(data = ncfile, x = ws, kde = True, bins = 26)
plt.xlabel("Speed (m/s)")
plt.ylabel("Number of counts (h)")
plt.title(f"Weibull distribution of wind speed at {site_name}")

```

Out[60]:

```
Text(0.5, 1.0, 'Weibull distribution of wind speed at Nordavind_A')
```

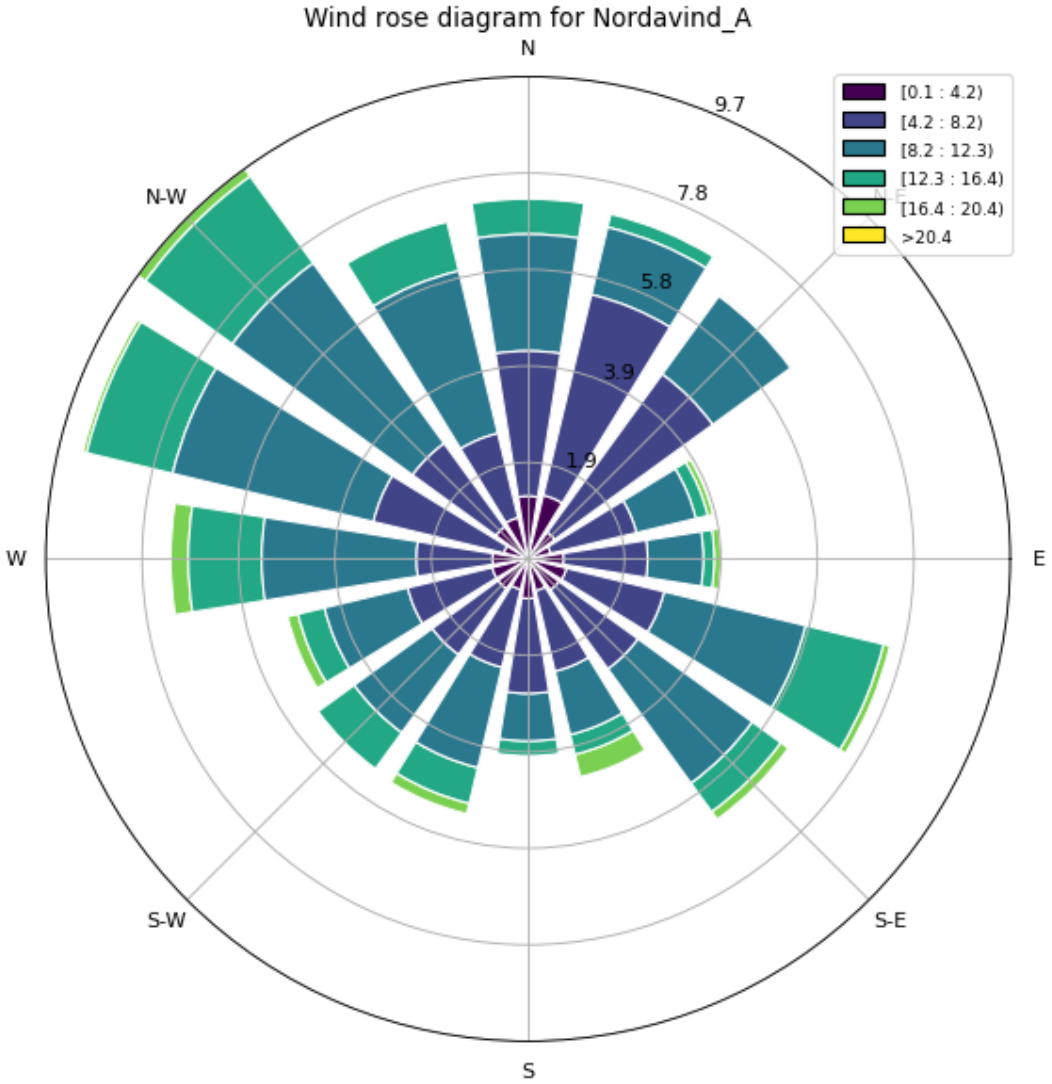


Windrose

the following example show how to make a windrose at the chosen site and timespan. The windrose illustrates the distribution of wind speed and direction together. The direction in degrees is transformed into cardinal directions, 360 - N, 180 - S, 90 - E, 270 - W

In [59]:

```
# run the code below if not already installed the windrose package
#!pip3 install windrose

from windrose import WindroseAxes

speed = ncfile.variables["wind_speed"][:]
direction = ncfile.variables["wind_direction"][:]

ax = WindroseAxes.from_ax()
ax.bar(direction,
       speed,
       normed=True, #get % of number of hours
       opening= 0.8, #width of bars
       edgecolor='white')
ax.set_legend(loc = "best")
plt.title(f"Wind rose diagram for {site_name}")
plt.show()

```



6.3 AROME-ARCTIC SCRIPTS

6.3.1 Retrieve Data AromeArctic

Download data from AROME-Arctic and save as ncfile

The following codes is a guide to download data from the NWP AROME-Arctic extracted at one location with coordinates (Lat/Lon) for a chosen time period and save it as an nc-file, available at https://thredds.met.no/thredds/catalog/aromearcticarchive/catalog.html (https://thredds.met.no/thredds/catalog/aromearcticarchive/catalog.html). This script use the arome_arctic_full files containing several variables.

choose a datespan (years, months, days) and save it as a ncfile.

Data available from: 2015.10.21 to current date

- AROME-Arctic issues deterministic forecasts 4 times a day with a lead time of 66 hours
- The model utilises a three-dimensional variational data assimilation with 3-hourly cycling to assimilate conventional observations, scatterometer ocean surface winds, satellite radiances and atmospheric motion vectors.

(Source and more informatil about the model: https://www.met.no/en/projects/The-weather-model-AROME-Arctic/about (https://www.met.no/en/projects/The-weather-model-AROME-Arctic/about))

In [ ]:

```
# Importing necessary modules

import xarray as xr
import numpy as np

import netCDF4 as nc
import matplotlib.pylab as plt
import pandas as pd

import pyproj
import datetime as dt
from datetime import date, timedelta

import requests
from __future__ import print_function
from IPython.display import display, HTML
```

In [ ]:

```
"""
The coordinates in get_coordinates is at the center of the offshore wind sites
"""

def get_coordinates(location_name):
    locations = {
        'Nordavind_A': {
            'latitude': 71.1314956,
            'longitude': 32.048109
        },
        'Nordavind_B': {
            'latitude': 71.7880587,
            'longitude': 27.7221338
        },
        'Nordavind_C': {
            'latitude': 71.7471898,
            'longitude': 19.9808019
        },
        'Nordavind_D': {
            'latitude': 71.473272,
            'longitude': 18.7614613
        }

    }

    if location_name in locations:
        coordinates = locations[location_name]
        return coordinates['latitude'], coordinates['longitude']
    else:
        return None
```

Seklima station information retrival with frost api

Link to frost webpage (https://frost.met.no/index.html)

1. Create client id for yourself
   - Create client id here (https://frost.met.no/auth/requestCredentials.html)

This is used to retrive the longitude and latitude

PS: Not all stations are included in the model area. Arome-Arctic project northern areas

In [ ]:

```
# Insert your own client ID here
client_id = 'insert client ID'
```

In [ ]:

```
'''
Getting the data for the meta dataframe

Using the frost api for seklima
'''

endpoint = 'https://frost.met.no/sources/v0.jsonld'

# Dictionary with elements to retrieve
parameters = {
    'fields': 'name,id,geometry,masl'
}

# Issue an HTTP GET request
r = requests.get(endpoint, parameters, auth=(client_id,''))
# Extract JSON data
json = r.json()

# Check if the request worked, print out any errors
if r.status_code == 200:
    data_exp = json['data']
    print('Data retrieved from frost.met.no!')
else:
    print('Error! Returned status code %s' % r.status_code)
    print('Message: %s' % json['error']['message'])
    print('Reason: %s' % json['error']['reason'])
```

In [ ]:

```
'''
Transforming the data from a raw json format retrieved by requests
to a pandas df format

When redefining the query the new columns need to be spesified for the dataframe df
'''

meta_df = pd.DataFrame(columns=['id', 'name', 'lon', 'lat', 'heigh-asl (m)'])
ignored_values = 0
for i in range(len(data_exp)):
    row = []
    try:
        row.append(data_exp[i]['id'])
        row.append(data_exp[i]['name'])
        row.append(data_exp[i]['geometry']['coordinates'][0])
        row.append(data_exp[i]['geometry']['coordinates'][1])
        row.append(data_exp[i]['masl'])
        meta_df.loc[len(meta_df)] = row
    except:
        ignored_values += 1
        continue


print(f'Number of discarded values {ignored_values}')

# Setting the station id as row index2
meta_df = meta_df.set_index('id')

display(meta_df)
```

- Change station ID to match measurement station to get station coordinates
  - Find station ID at (stations (https://seklima.met.no/stations/))

or

- Insert site_name to get coordinates from center of OWS

In [ ]:

```

##################


station_id = "SN76956" # insert station ID, ex: 'SN76956' Goliat ,  'SN20926' hjelms


#################

longitude = meta_df.loc[f"{station_id}"]["lon"]
latitude = meta_df.loc[f"{station_id}"]["lat"]

print(f"{station_id}: longitude = {longitude}, latitude = {latitude}")
```

In [ ]:

```
# insert the site name you want the data extracted from

#################

site_name = 'Nordavind_C'

#################

latitude, longitude = get_coordinates_center_OWS(site_name) # gets the latitude and
print(f"{site_name}: longitude = {longitude}, latitude = {latitude}")
```

In [ ]:

```
# Choose latitude, longitude

latitude =
longitude =
```

Projecting coordinates

Starts by projecting the chosen Latitude and Longitude coordinate to x and y cooridnates used in the model

- No changes needed in this part

In [ ]:

```
# Load in a random file to project coordinates
filename = "https://thredds.met.no/thredds/dodsC/aromearcticarchive/"+\
               "2022/01/01/arome_arctic_full_2_5km_20220101T00Z.nc"
ncfile = nc.Dataset(filename)

crs_AA = pyproj.CRS.from_cf(
         {
            "grid_mapping_name": "lambert_conformal_conic",
            "standard_parallel": [77.5, 77.5],
            "longitude_of_central_meridian": -25.0,
            "latitude_of_projection_origin": 77.5,
             "earth_radius": 6371000.0,
         }
)

# Transformer to project from ESPG:4368 (WGS:84) to our lambert_conformal_conic
proj = pyproj.Transformer.from_crs(4326,crs_AA,always_xy=True)

# Compute projected coordinates of lat/lon point
lat = latitude
lon = longitude
X,Y = proj.transform(lon,lat)

# Find nearest neighbour
x = ncfile.variables["x"][:]
y = ncfile.variables["y"][:]

Ix = np.argmin(np.abs(x - X))
Iy = np.argmin(np.abs(y - Y))

ncfile.close()

```

Collecting the data

In this section you can choose "start date" and "end date" for the timeperiod of data to be extracted.

Each file contains a forecast of 66 hours. Due to spin-up error this script retrieves data starting 6h-in in each file extracting 3 hours from the file before jumping to next file. The files contains several variables and this scrips acts as an example, other variables can be added or removed by alternating lists and variables. Variables will be extracted from the chosen height (except variables at a given height, ex: surface_air_pressure)

- Change start_date and end_date
- Choose hyrbid level (height)
- Change empty lists and extracted variables

- PS: x_wind and y_wind is relativ to model, not rotated to cardinal directions.
- (Rotation to cardinal direction is done by using the alpha.nc file)

time to collect ?? months; approx ?? min.

In [ ]:

```
1
2  ####################
3
4  start_date = date(2021, 12, 31) # choose start date (year, month, day)
5  end_date = date(2022, 12, 31) # choose end date (year, month, day)
6
7  delta = timedelta(days=1) # delta (timestep), default set to 1 day
8
9
10 """
11 Choose a hybrid level [64 to 0]:
12 64 = 0 masl
13 63 = 24 masl
14 62 = 48 masl
15 61 = 73 masl
16 60 = 99 masl
17 59 = 127 masl
18 58 = 156 masl
19 57 = 187 masl
20 56 = 221 masl
21 55 = 259 masl
22 """
23
24 hybrid_lvl = 62   # 64 surface, 0 ToA
25
26 ######################
27
28
29
30 """
31 Create empty lists for each variable to be extracted
32 """
33
34 time = []
35 air_temp=[]
36 x_wind = []
37 y_wind = []
38 surface_air_pressure = []
39
40
41 hours = ["00", "03", "06", "09", "12", "15", "18", "21"]
42
43 while start_date <= end_date:
44     y = start_date.strftime("%Y")
45     m = start_date.strftime("%m")
46     d = start_date.strftime("%d")
47
48     for i in hours:
49         opendap_url = f"https://thredds.met.no/thredds/dodsC/aromearcticarchive/{y}/
50
51         try:
52
53             # Extract variables
54             ncfile = nc.Dataset(opendap_url)
55             times = ncfile.variables["time"][6:9]
56             airtemp = ncfile.variables["air_temperature_ml"][6:9, hybrid_lvl, Iy, Ix
57             xwind = ncfile.variables["x_wind_ml"][6:9, hybrid_lvl, Iy, Ix]
58             ywind = ncfile.variables["y_wind_ml"][6:9, hybrid_lvl, Iy, Ix]
59             sur_p = ncfile.variables["surface_air_pressure"][6:9, 0, Iy, Ix]
```

```
            # Add variables to lists
            time.extend(times)
            air_temp.extend(airtemp)
            x_wind.extend(xwind)
            y_wind.extend(ywind)
            surface_air_pressure.extend(sur_p)

            ncfile.close()
        except Exception as e: # sometimes files for 1 day are missing spesific hour
            # If opendap_url is not found or any other exception occurs, skip to the
            print(f"Skipping hour {i} of date {y}-{m}-{d} due to exception: {e}")
            continue

    start_date += delta

```

Rotating wind with Alpha

This section uses the alpha.nc file with rotated local grid in every location. alpha.nc is created by running a seperate code, this needs to be done before running the following.

- This section rotates wind into cardinal directions before saving the file. This can also be skipped in this step and be done when processing the final ncfile.

In [ ]:

```
# Convert x-y to cardinal direction and speed


# Open alpha.nc with rotated local grids and extract alpha from Iy and Ix
ncfile = nc.Dataset("alpha.nc")

alpha = ncfile.variables["alpha"][Iy, Ix]



# Wind direction relative to Earth (wdir) may be calculated as follows:
#   wdir = alpha + 90-atan2(v,u)
# where u and v are model wind relative to model grid

wdir = []
ws = []

for i in range(0,len(x_wind)):
    w = alpha + (90-np.arctan2(y_wind[i], x_wind[i]))
    wdir.append(w)

    speed = np.sqrt(x_wind[i]**2 + y_wind[i]**2)
    ws.append(speed)

```

Dictionary

The output from collecting data is stored in lists. This section creates an dictionary and uses pandas to store in a easy-reference system

- Changes can be made in the dictionary "d_data" if variables are ignored or added

In [ ]:

```
# create a dictionary and give name to the lists
d_data = {
     "time" : time,
     f"air_temperature_Hyb:{hybrid_lvl}": air_temp,
     "surface_pressure": surface_air_pressure,
     f"x_wind_Hyb:{hybrid_lvl}": x_wind,
     f"y_wind_Hyb:{hybrid_lvl}": y_wind,
     f"wind_speed": ws,
     f"wind_direction": wdir
}


# convert to pandas dataframe

weather_data = pd.DataFrame.from_dict(d_data)
weather_data = weather_data.set_index('time')

display(weather_data)
```

Save as NC-file

Define a function that saves the "weather_data" as an nc-file.

- Change the output_filename to desired new saved filename

Filesize for time, wind speed and wind direction for ?? months: ?? KB

In [ ]:

```
"""
This code is an easy option saving the new file directly in the same directory as th
"""

def save_dataframe_to_netcdf(dataframe, output_file):
    dataset = xr.Dataset(data_vars=dataframe.to_dict('series'))
    dataset.to_netcdf(output_file)

    print(f"Data saved successfully to {output_file}.")



output_filename = 'AROME_dir13_weather_data.nc' #choose filename: default: "AROME_we

save_dataframe_to_netcdf(weather_data, output_filename)
```

In [ ]:

```
"""
This code makes a new optional directory at a desired location and saves the new fil
If the directory already exist it will just save the new file in that directory. Thi
in the same directory by keeping the same directory name and changing only the outpu
"""

import os

# New directory
directory = "weather_data_NORA3"

# Parent Directory path
parent_dir = "C:/Users/Ida/"

# Filename new ncfile
# {site_name} or {station_id}
output_filename = 'AROME2_weather_data.nc' #choose filename: default: "weather_data_


# Path
path = os.path.join(parent_dir, directory)

# Create the directory if not already existing
if not os.path.isdir(path):
    os.mkdir(path)


def save_dataframe_to_netcdf(dataframe, output_file):
    dataset = xr.Dataset(data_vars=dataframe.to_dict('series'))
    output_path = f"{parent_dir}{directory}/{output_file}"
    dataset.to_netcdf(output_path)

    print(f"Data saved successfully to {output_path}.")


save_dataframe_to_netcdf(weather_data, output_filename)
```

6.3.2 Alpha local grid rotation AA

Alpha, local grid rotation

This script calculates the local grid rotation needed to transform wind direction in cardinal directions.

- This works for MEPS and AROME-Arctic datasets more information: [https://github.com/metno/NWPdocs/wiki/Examples#extracting-and-plotting-a-variable] (https://github.com/metno/NWPdocs/wiki/Examples#extracting-and-plotting-a-variable%5D)

Calculates "Alpha" at every grid in inputfile and writes as seperate output file (NC-file).

Time: ?? min and Filesize: ?? KB

In [ ]:

```
#!pip3 install geopy

# Both the functions for distance calculates the harversine distance
# The first function use built in module, not showing the process

import geopy.distance

# define a function calculating the Harversine distance

def distance(origin, destination):
    """
    (Source: https://stackoverflow.com/questions/19412462/
            getting-distance-between-two-points-based-on-latitude-longitude)
    """

    d = geopy.distance.geodesic(origin, destination).km

    return d
```

In [ ]:

```
import xarray as xr
import numpy as np
import netCDF4 as nc
"""
This routine calculates the local grid rotation (alpha) from input file,
and writes to a separate output file.
Formula:
  alpha = atan2(dlatykm,dlonykm)*180/pi - 90)

Wind direction relative to Earth (wdir) may later be calculated as follows:
   wdir = alpha + 90-atan2(v, u)
where u(x) and v(y) are model wind relative to model grid

"""


# Function calculating Harversine distance, more elaborate.

def distance(origin, destination):
    """
    (Source: https://stackoverflow.com/questions/19412462/
             getting-distance-between-two-points-based-on-latitude-longitude)

    Calculate the Haversine distance.

    Parameters
    ----------
    origin : tuple of float
        (lat, long)
    destination : tuple of float
        (lat, long)

    Returns
    -------
    distance_in_km : float

    Examples
    --------
    >>> origin = (48.1372, 11.5756)  # Munich
    >>> destination = (52.5186, 13.4083)  # Berlin
    >>> round(distance(origin, destination), 1)
    504.2
    """

    lat1, lon1 = origin
    lat2, lon2 = destination
    radius = 6371  # km

    dlat = np.radians(lat2 - lat1)
    dlon = np.radians(lon2 - lon1)
    a = (np.sin(dlat / 2) * np.sin(dlat / 2) +
         np.cos(np.radians(lat1)) * np.cos(np.radians(lat2)) *
         np.sin(dlon / 2) * np.sin(dlon / 2))
    c = 2 * np.arctan2(np.sqrt(a), np.sqrt(1 - a))
    d = radius * c

    return d
```

In [ ]:

```
import xarray as xr
import numpy as np
import netCDF4 as nc

# use a random arome arctic file
infile = "https://thredds.met.no/thredds/dodsC/aromearcticarchive/"+\
         "2022/01/01/arome_arctic_full_2_5km_20220101T00Z.nc"


nc = xr.open_dataset(infile)

# Variables
xx = nc.x
yy = nc.y
lat = nc.latitude
lon = nc.longitude
alpha=lat #Target matrix


for j in range(0,yy.size-1):
    for i in range(0,xx.size-1):
        # Prevent out of bounds
        if j==yy.size-1:
            j1=j-1; j2=j
        else:
            j1=j; j2=j+1
        if i==xx.size-1:
            i1=i-1; i2=i
        else:
            i1=i; i2=i+1

        dlatykm=distance([lat[j1,i1],lon[j1,i1]],[lat[j2,i1],lon[j1,i1]])
        dlonykm=distance([lat[j1,i1],lon[j1,i1]],[lat[j1,i1],lon[j2,i1]])

        alpha[j,i]=np.arctan2(dlatykm,dlonykm)*180/np.pi - 90


```

In [ ]:

```
import netCDF4 as nc

# Make NetCDF file

outfile = "alpha.nc"

rg = nc.Dataset(outfile, 'w', format='NETCDF4')
x=rg.createDimension("x",xx.size)
y=rg.createDimension("y",yy.size)
alph=rg.createVariable("alpha","f4",("y","x"))
alph[:]=alpha
rg.close()
```

A VARIABLES IN THE NORA3 FILES

- **Download NORA3 wind atoms**
- **Download NORA3 atoms**
- **Download NORA3 fpnc**

In [10]:

```
import netCDF4 as nc

filename = 'https://thredds.met.no/thredds/dodsC/nora3_subset_atmos'\
            '/wind_hourly/arome3kmwind_1hr_202212.nc'

ncfile   = nc.Dataset(filename)


# Check variables in file
for variable in ncfile.variables:
    print(variable)
```

```
forecast_reference_time
projection_lambert
time
x
y
height
latitude
longitude
wind_speed
wind_direction
```

In [11]:

```
import netCDF4 as nc

filename = 'https://thredds.met.no/thredds/dodsC/nora3_subset_atmos'\
            '/atm_hourly/arome3km_1hr_202212.nc'

ncfile   = nc.Dataset(filename)


# Check variables in file
for variable in ncfile.variables:
    print(variable)
```

```
forecast_reference_time
height0
height1
height4
height_above_msl
projection_lambert
time
x
y
height3
air_pressure_at_sea_level
air_temperature_2m
high_type_cloud_area_fraction
latitude
lifting_condensation_level
longitude
low_type_cloud_area_fraction
medium_type_cloud_area_fraction
relative_humidity_2m
wind_direction
wind_speed
surface_net_longwave_radiation
surface_net_shortwave_radiation
precipitation_amount_hourly
fog
```

In [13]:

```
import netCDF4 as nc

filename = 'https://thredds.met.no/thredds/dodsC/nora3/2022/01/01/00/fc2022010100_003_fp.nc

ncfile   = nc.Dataset(filename)


# Check variables in file
for variable in ncfile.variables:
    print(variable)
```

```
time
pressure0
height_above_msl
height0
height1
height2
height3
height4
atmosphere_as_single_layer
top_of_atmosphere
projection_lambert
x
y
forecast_reference_time
longitude
latitude
air_temperature_0m
surface_geopotential
liquid_water_content_of_surface_snow
downward_northward_momentum_flux_in_air
downward_eastward_momentum_flux_in_air
integral_of_toa_net_downward_shortwave_flux_wrt_time
integral_of_surface_net_downward_shortwave_flux_wrt_time
integral_of_toa_outgoing_longwave_flux_wrt_time
integral_of_surface_net_downward_longwave_flux_wrt_time
integral_of_surface_downward_latent_heat_evaporation_flux_wrt_time
integral_of_surface_downward_latent_heat_sublimation_flux_wrt_time
water_evaporation_amount
surface_snow_sublimation_amount_acc
integral_of_surface_downward_sensible_heat_flux_wrt_time
integral_of_surface_downwelling_shortwave_flux_in_air_wrt_time
integral_of_surface_downwelling_longwave_flux_in_air_wrt_time
air_temperature_2m
relative_humidity_2m
specific_humidity_2m
x_wind_10m
y_wind_10m
cloud_area_fraction
x_wind_gust_10m
y_wind_gust_10m
air_temperature_max
air_temperature_min
convective_cloud_area_fraction
high_type_cloud_area_fraction
medium_type_cloud_area_fraction
low_type_cloud_area_fraction
atmosphere_boundary_layer_thickness
hail_diagnostic
rainfall_amount
snowfall_amount
graupelfall_amount
x_wind_pl
y_wind_pl
air_temperature_pl
cloud_area_fraction_pl
geopotential_pl
relative_humidity_pl
upward_air_velocity_pl
air_pressure_at_sea_level
lwe_thickness_of_atmosphere_mass_content_of_water_vapor
x_wind_z
y_wind_z
surface_air_pressure
lifting_condensation_level
```

```
atmosphere_level_of_free_convection
atmosphere_level_of_neutral_buoyancy
wind_direction
wind_speed
precipitation_amount_acc
snowfall_amount_acc
```

B OFFSHORE WIND SITES

B.0.1 Nordavind A

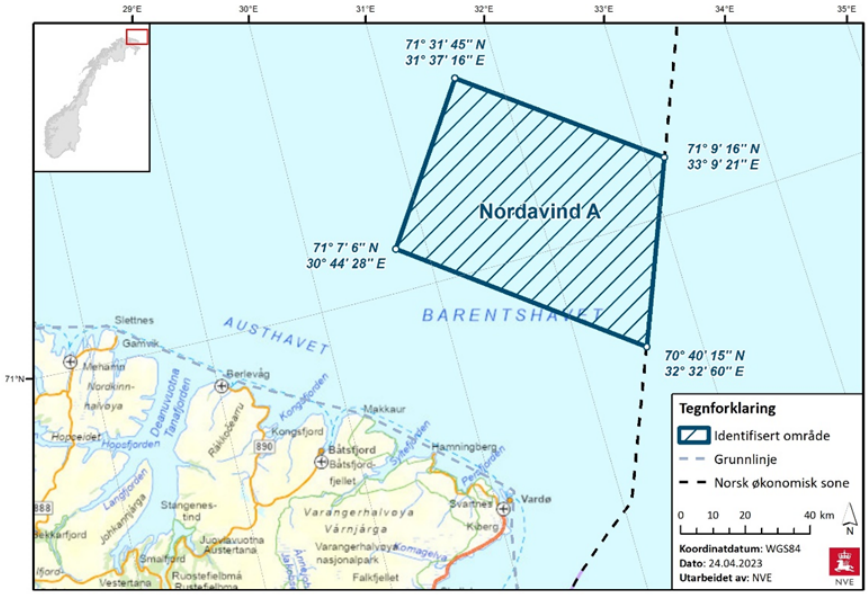


Figure 3: Nordavind A offshore wind site

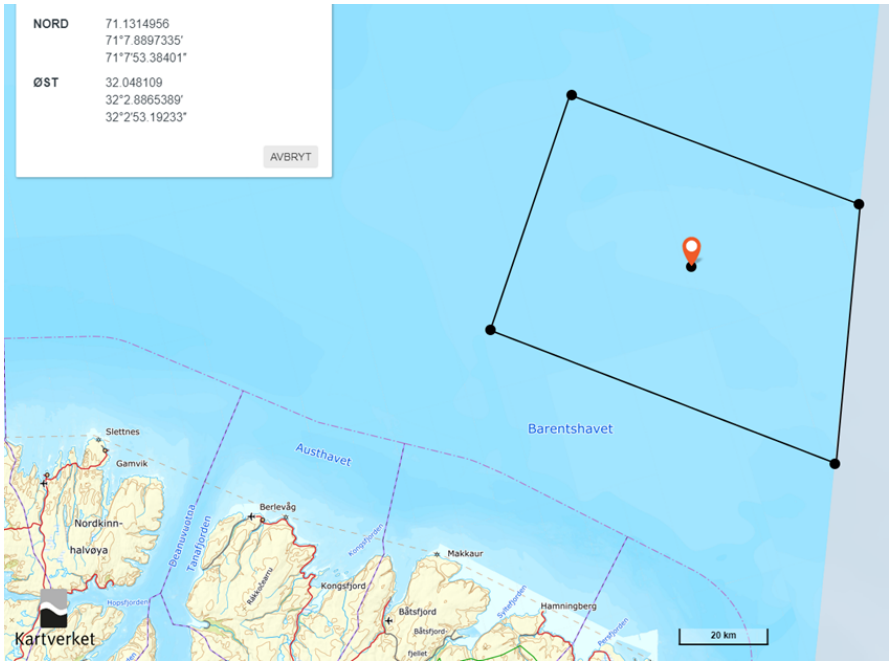


Figure 4: Approximation to center of Nordavind A offshore wind site: (Lat = 71.1314956 N, Lon = 32.048109 E)

Closest weather station (Nordavind A)			
Station ID	**Station name**	**Latitude**	**Longitude**
SN98580	Vardø Lufthavn	70.35°N	31.04°E
SN98550	Vardø Radio	70.3707°N	31.0962°E

B.0.2 Nordavind B

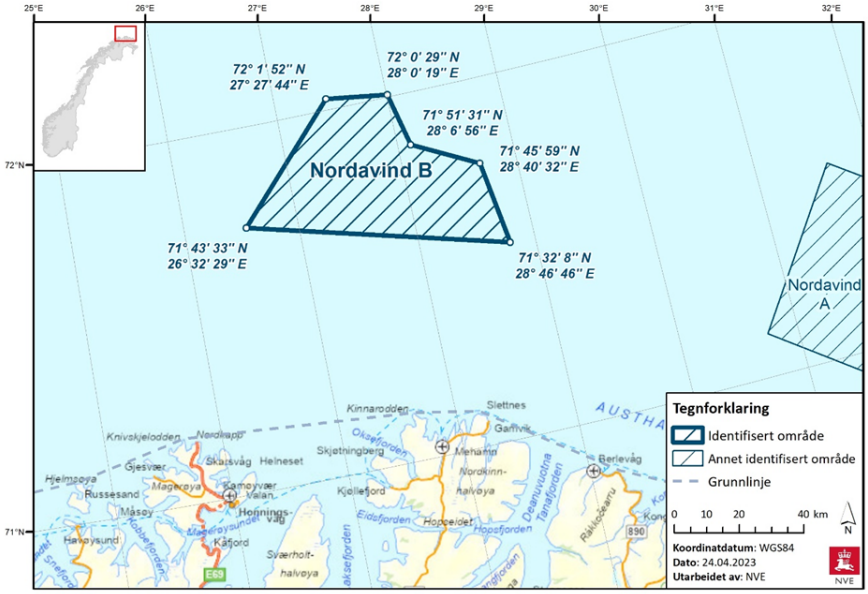


Figure 5: Nordavind B offshore wind site

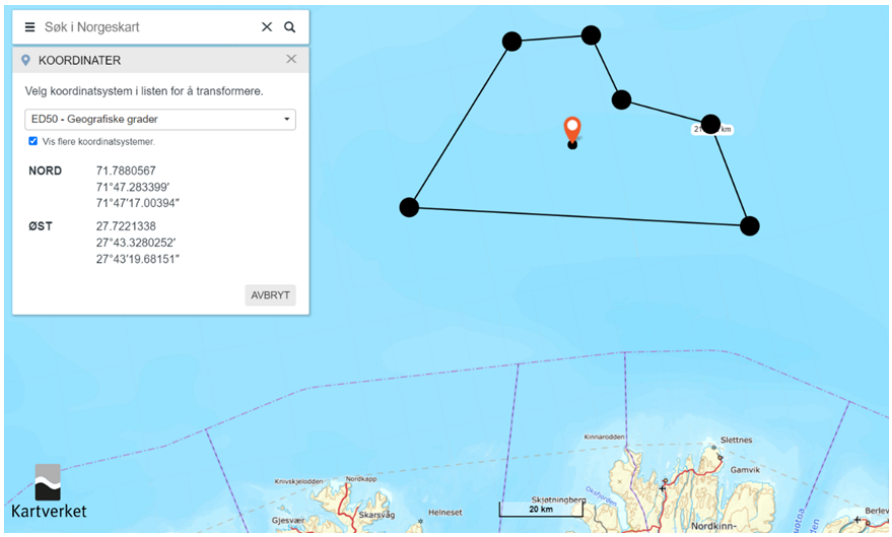


Figure 6: Approximation to center of Nordavind B offshore wind site: (Lat = 71.7880587N, Lon = 27.7221338E)

Closest weather station (Nordavind B)			
Station ID	**Station name**	**Latitude**	**Longitude**
SN76956	Goliat FPSO	71.3112°N	22.25°E
SN96400	Slettnes Fyr	71.0888°N	28.217°E

B.0.3 Nordavind C

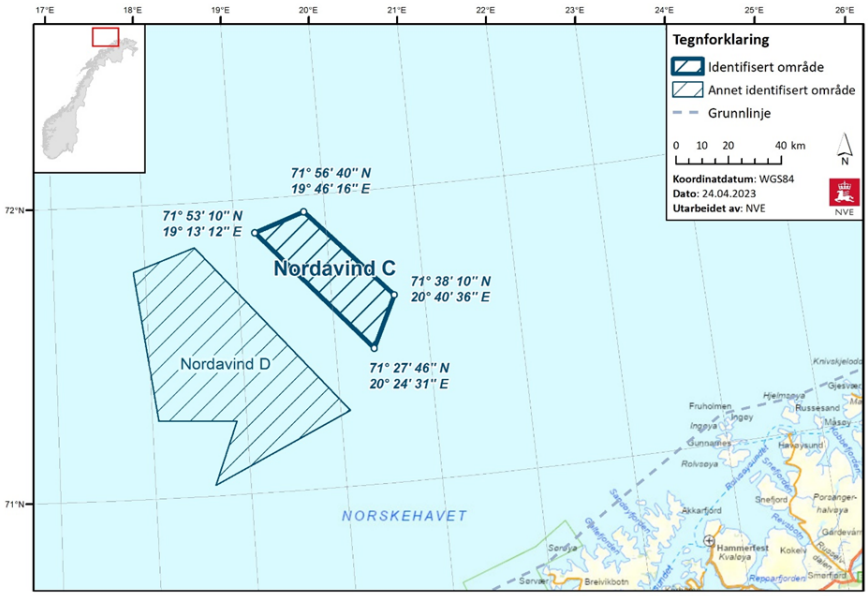


Figure 7: Nordavind C offshore wind site

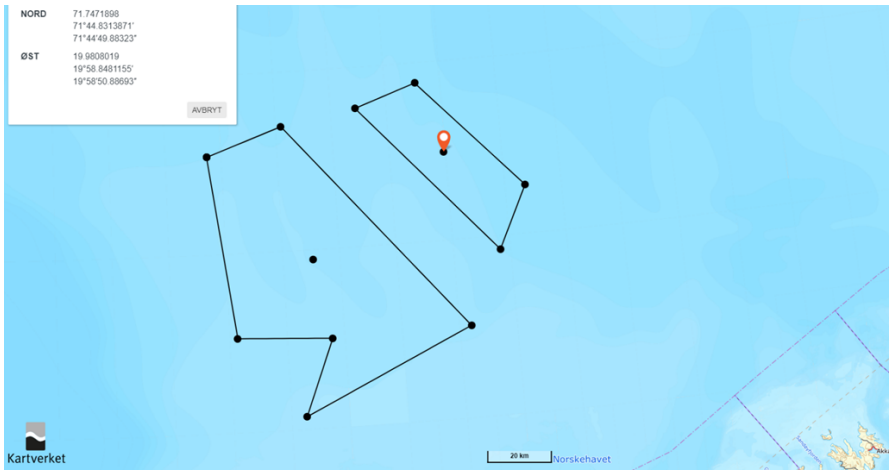


Figure 8: Approximation to center of Nordavind C offshore wind site: (Lat = 71.7471898 N, Lon = 19.9808019 E)

Closest weather station (Nordavind C)			
Station ID	**Station name**	**Latitude**	**Longitude**
SN76956	Goliat FPSO	71.3112°N	22.25°E
SN20926	Hjelmsøybanken	72.49°N	20.15°E

B.0.4 Nordavind D

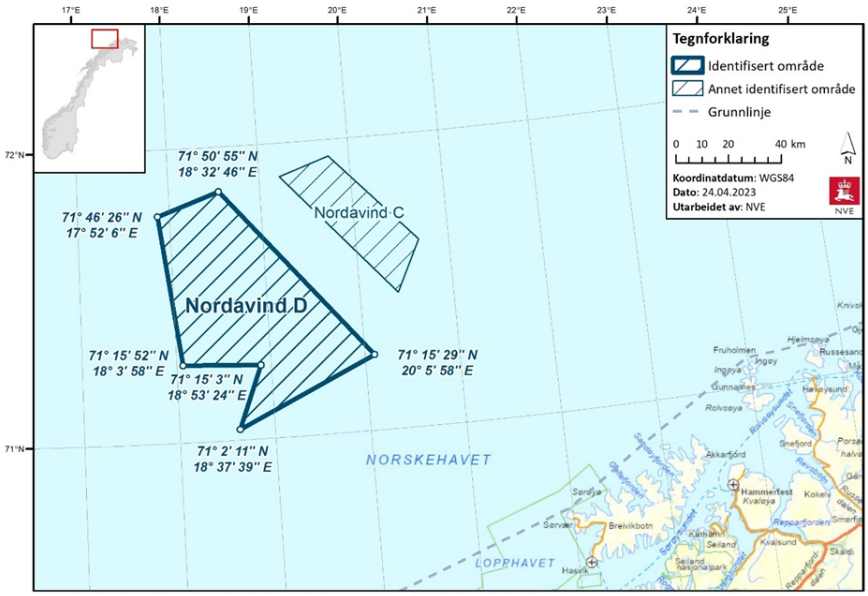


Figure 9: Nordavind D offshore wind site

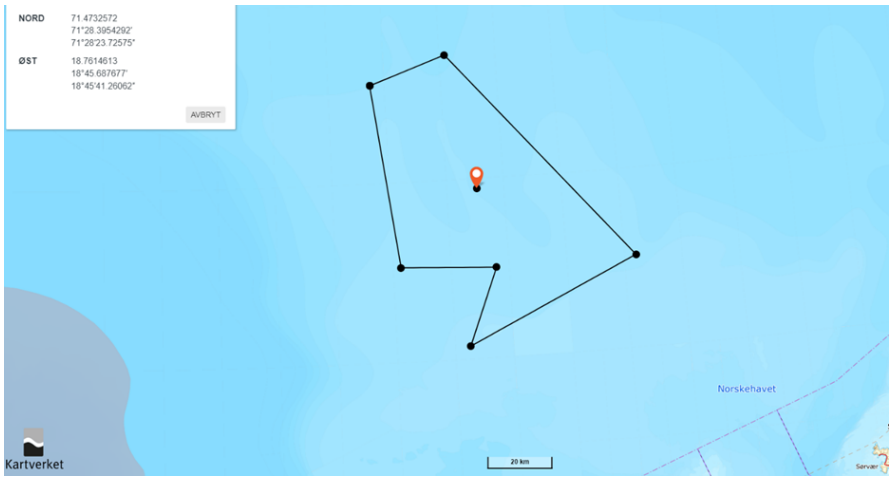


Figure 10: Approximation to center of Nordavind D offshore wind site: (Lat = 71.473272 N, Lon = 18.7614613 E)

Closest weather station (Nordavind D)			
Station ID	**Station name**	**Latitude**	**Longitude**
SN76956	Goliat FPSO	71.3112°N	22.25°E
SN20926	Hjelmsøybanken	72.49°N	20.15°E

B.0.5 Utsira Nord

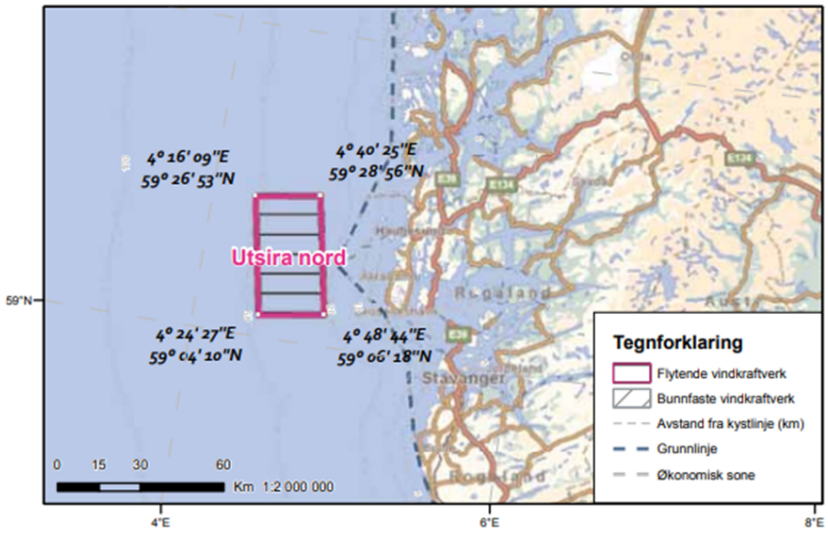


Figure 11: Utsira Nord offshore wind site

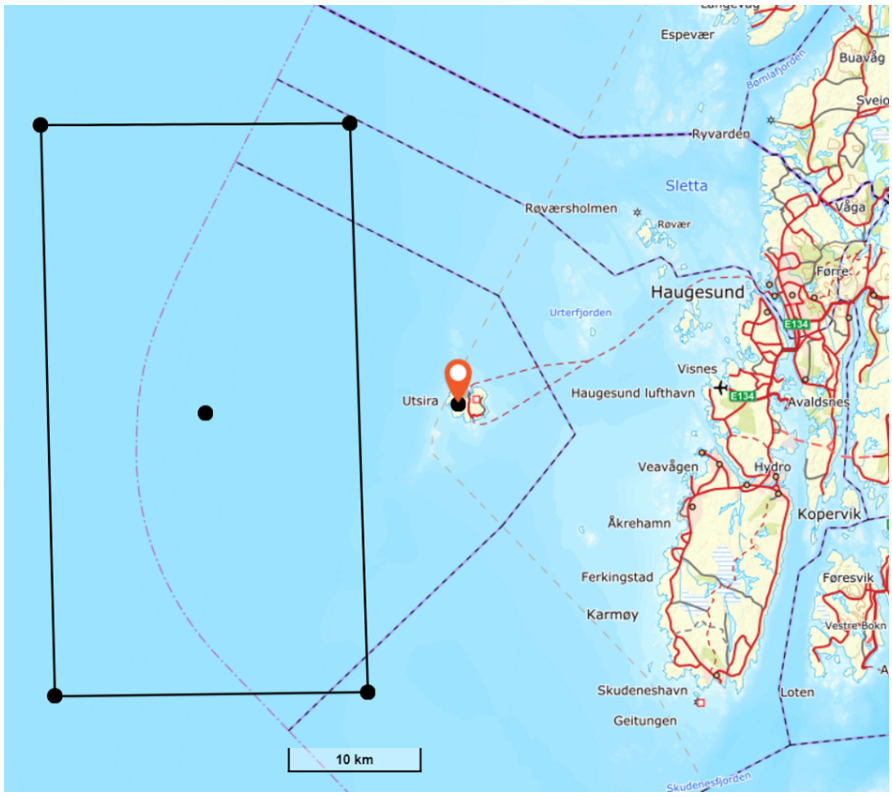


Figure 12: Approximation to center of Utsira Nord offshore wind site: (Lat = 59.2741 N, Lon = 4.544 E), and location of Utsira Fyr

Closest weather station (Utsira Nord)			
Station ID	**Station name**	**Latitude**	**Longitude**
SN47300	Utsira Fyr	59.3065°N	4.8723°E

B.0.6 Sørlige Nordsjø II

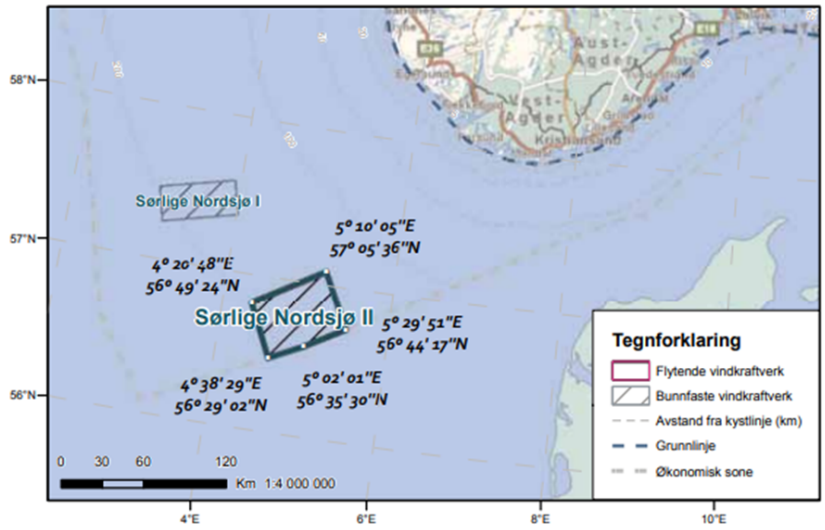


Figure 13: Sørlige Nordsjø II offshore wind site

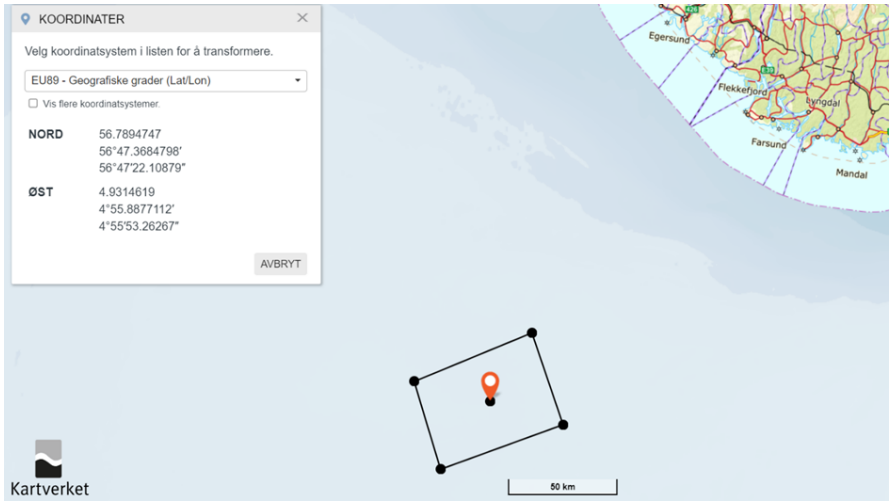


Figure 14: Approximation to center of Sørlige Nordsjø II offshore wind site: (Lat = 59.78947 N, Lon = 4.93146 E)

Closest weather station (Sørlige Nordsjø II)			
Station ID	**Station name**	**Latitude**	**Longitude**
SN76939	Valhall A	56.2781°N	3.3928°E
SN76920	Ekofisk	56.5433°N	3.2243°E

C EXAMPLE: COMPARE DATA (NORA3 VS FROST)

This is an example on how to use the ncfiles from after downloading data using the scipts. This example looks at wind speed data at the location of Goliat for the time period year 2022

Comparison of wind speed at Goliat 2022

This is a comparison between the data retrieved from seklima and NORA3 at Goliat (SN76956) for year 2022

In [2]:

```
import netCDF4 as nc
import numpy as np
```

Load in files

In [3]:

```
filename = "C:/Users/Ida/weather_data_2022_Goliat/NORA3_wind_50m_weather_data_GOLIAT
nora_nc = nc.Dataset(filename)

filename2 = "C:/Users/Ida/weather_data_2022_Goliat/seklima_weather_data_station_GOLI
obs_nc = nc.Dataset(filename2)
```

Convert timestamps

In [4]:

```
# Observations

# covert timestamps into nice readable dates
import datetime as dt
from datetime import date, timedelta

time = obs_nc.variables["time"][:]

ts = np.linspace(0,len(time)-1,len(time))

nice_time_obs = []

for i in range(0,len(ts)):
    t = dt.datetime.utcfromtimestamp(time[i]/10**9)
    nice_time_obs.append(t)

print(nice_time_obs[0], nice_time_obs[-1]) # check that start and end date/time is c
print(len(nice_time_obs))
```

```
2022-01-01 00:00:00 2022-12-31 23:40:00
26027
```

In [5]:

```
# NORA3

# covert timestamps into nice readable dates

time = nora_nc.variables["time"][:]

ts = np.linspace(0,len(time)-1,len(time))

nice_time_N = []


for i in range(0,len(ts)):
    t = dt.datetime.utcfromtimestamp(time[i])
    nice_time_N.append(t)

print(nice_time_N[0], nice_time_N[-1]) # check that start and end date/time is corre
print(len(nice_time_N))
```

```
2022-01-01 00:00:00 2022-12-31 23:00:00
8760
```

Filter whole hours from observation data

In [6]:

```
# Load in variables from observations
wind_speed_obs = obs_nc.variables["wind_speed"][:]
```

In [7]:

```
# NORA3 time is in whole hours, retrieve only whole hours from observationdata
# Observation data is every 20 min

# Initialize a list to store the indexes with whole hours
whole_hour_time = []
whole_hour_wind_speed = []

# Loop over each index
for i in range(len(nice_time_obs)):

    # Check if the hour is zero, indicating a whole hour
    if nice_time_obs[i].minute == 0:
        whole_hour_time.append(nice_time_obs[i])
        whole_hour_wind_speed.append(wind_speed_obs[i])
print(len(whole_hour_time))
```

```
8717
```

Find missing hours

In [8]:

```
"""
Script to locate hours with missing data from OBSERVATION
"""

def generate_all_hours_in_year(year):
    # Create a list of all hours in the given year
    start_date = dt.datetime(year, 1, 1, 0, 0)
    end_date = dt.datetime(year + 1, 1, 1, 0, 0)
    delta = dt.timedelta(hours=1)

    current_date = start_date
    all_hours_in_year = []

    while current_date < end_date:
        all_hours_in_year.append(current_date)
        current_date += delta

    return all_hours_in_year

def find_missing_hours(dates_with_hours, all_hours_in_year):
    # Find missing hours by comparing the two lists
    missing_hours = [hour for hour in all_hours_in_year if hour not in dates_with_ho
    return missing_hours


# Assuming you have a list of dates with hours for a given year (dates_with_hours)
year = 2022  # Change this to the desired year

# Generate all hours in the given year
all_hours_in_year = generate_all_hours_in_year(year)

# Find missing hours from "whole_hour_time"
missing_hours = find_missing_hours(whole_hour_time, all_hours_in_year)



# print amount of missing hours
print(len(missing_hours))

# Print dates with missing hours
for missing_hour in missing_hours:
       print(missing_hour)


```

```
43
2022-02-28 02:00:00
2022-02-28 03:00:00
2022-02-28 04:00:00
2022-02-28 05:00:00
2022-02-28 06:00:00
2022-02-28 07:00:00
2022-02-28 08:00:00
2022-04-06 21:00:00
2022-04-06 22:00:00
2022-04-06 23:00:00
2022-04-07 03:00:00
2022-04-07 04:00:00
2022-04-07 05:00:00
2022-04-07 07:00:00
2022-04-07 08:00:00
2022-12-10 09:00:00
2022-12-10 10:00:00
2022-12-10 11:00:00
2022-12-10 12:00:00
2022-12-10 13:00:00
2022-12-10 14:00:00
2022-12-10 15:00:00
2022-12-10 16:00:00
2022-12-10 17:00:00
2022-12-10 18:00:00
2022-12-10 19:00:00
2022-12-10 20:00:00
2022-12-10 21:00:00
2022-12-10 22:00:00
2022-12-10 23:00:00
2022-12-11 00:00:00
2022-12-11 01:00:00
2022-12-11 02:00:00
2022-12-11 03:00:00
2022-12-11 04:00:00
2022-12-11 05:00:00
2022-12-11 06:00:00
2022-12-11 07:00:00
2022-12-11 08:00:00
2022-12-11 09:00:00
2022-12-11 10:00:00
2022-12-11 11:00:00
2022-12-11 12:00:00
```

Remove missing hours from NORA3 data

In [9]:

```
# Remove the hours with missing data from observation in NORA3 data as well

wind_speed_N = obs_nc.variables["wind_speed"][:]

def remove_missing_hours_from_list(hours_list):
    # removes missing hours
    cleaned_list = [hour for hour in hours_list if hour not in missing_hours]
    return cleaned_list

# removes wind speeds corresponding to the missing hour
cleaned_wind_speeds = []

for date, wind_speed in zip(nice_time_N, wind_speed_N):
    if date not in missing_hours:
        cleaned_wind_speeds.append(wind_speed)



cleaned_time = remove_missing_hours_from_list(nice_time_N)
```

In [10]:

```
# Check lenght to be correct
print(len(cleaned_wind_speeds), len(cleaned_time))
```

```
8717 8717
```

Compare wind speed

In [11]:

```
# Goliat at height 46m and NORA3 extracted at 50m

# Compare mean with raw data, no extrapolation

ave_wind_OBS = sum(whole_hour_wind_speed)/len(whole_hour_wind_speed)
ave_wind_NORA = sum(cleaned_wind_speeds)/len(cleaned_wind_speeds)


print(f'Nora3: {ave_wind_NORA} m/s Obs(raw): {ave_wind_OBS} m/s')

diff = ((ave_wind_OBS-ave_wind_NORA) / ave_wind_OBS ) * 100

print(f'{diff} %')
```

```
Nora3: 8.972261099001928 m/s Obs(raw): 7.089606516003246 m/s
-26.55513502405244 %
```

In [12]:

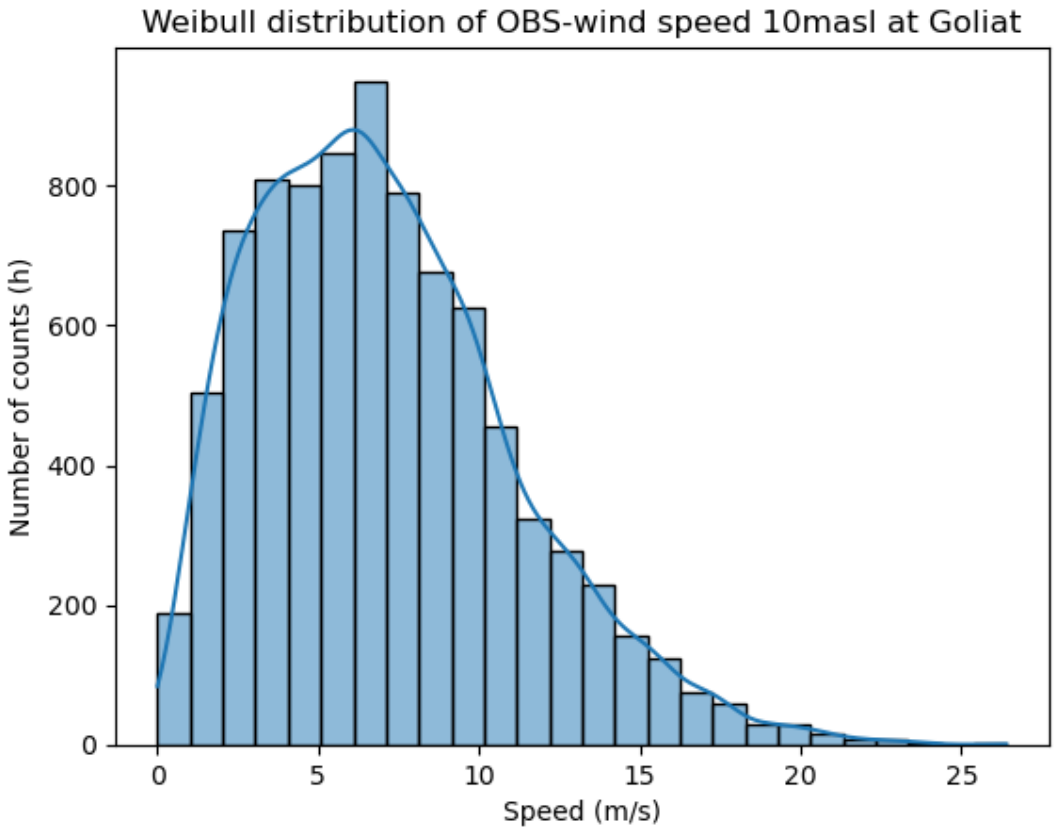
```
# Goliat at height 46m (wind is adjusted to 10m) and NORA3 extracted at 50m

# extrapolate NORA3 data to 10m

z1 = 50 # from height (m)
z2 = 10 # to height (m)

wind_10 = []
a = 0.12


for u in cleaned_wind_speeds:
    u_2 = u*(z2/z1)**a
    wind_10.append(u_2)

ave_wind_10N = sum(wind_10)/len(wind_10)


print(f'Nora3(10m): {ave_wind_10N} Obs: {ave_wind_OBS}')

diff = ((ave_wind_OBS-ave_wind_10N) / ave_wind_OBS ) * 100
print(f'{diff} %')
```

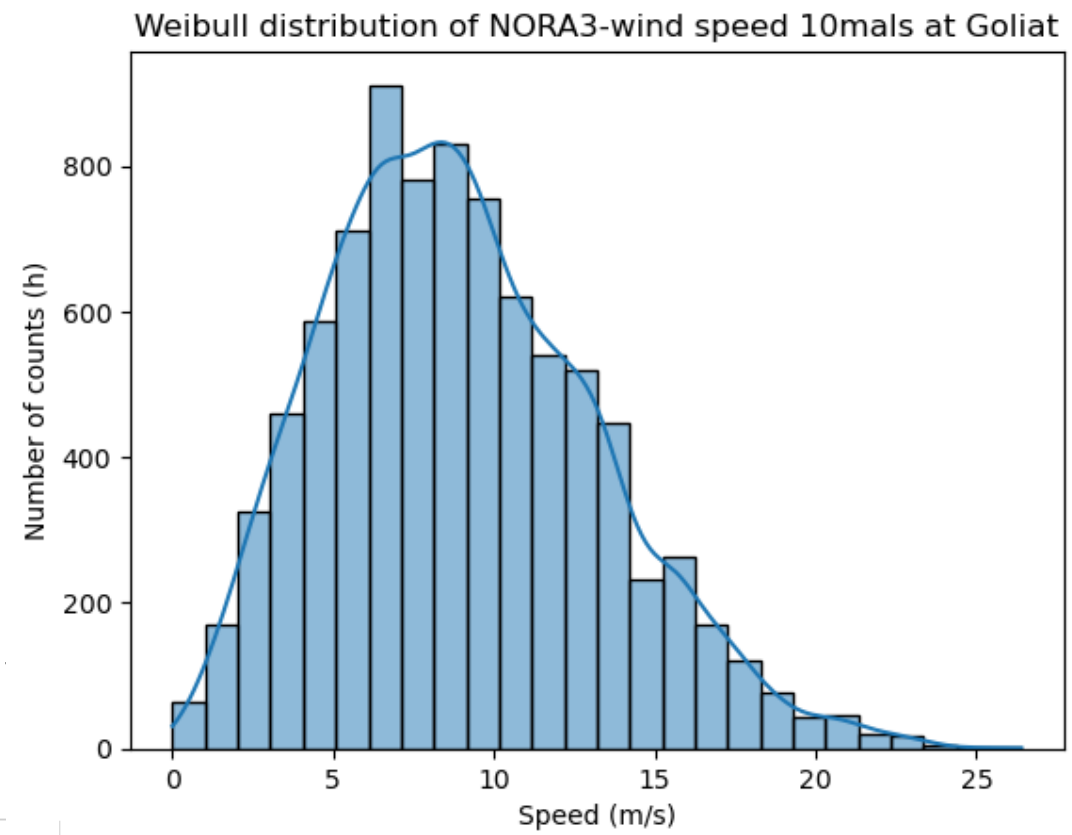
```
Nora3(10m): 7.396487077812552 Obs: 7.089606516003246
-4.328597942870173 %
```

Histogram

In [35]:

```
#install the seaborn package (if not already installed) by removing the "#" and runn
#!pip3 install seaborn

import seaborn as sns
import matplotlib.pylab as plt

sns.histplot(data = obs_nc, x = whole_hour_wind_speed, kde = True, bins = 26)
plt.xlabel("Speed (m/s)")
plt.ylabel("Number of counts (h)")
plt.title("Weibull distribution of OBS-wind speed 10masl at Goliat")
plt.show()

sns.histplot(data = nora_nc, x = cleaned_wind_speeds, kde = True, bins = 26)
plt.xlabel("Speed (m/s)")
plt.ylabel("Number of counts (h)")
plt.title("Weibull distribution of NORA3-wind speed 10mals at Goliat")
plt.show()
```





ervation and

```
# check index for measurements under a certain value.

def get_indexes_with_value(input_list, value):
    return [i for i, item in enumerate(input_list) if item < value] # Change < to >

# Example usage

target_value = 3  # return index if value is under target_value

result_obs = get_indexes_with_value(whole_hour_wind_speed, target_value)
result_N = get_indexes_with_value(cleaned_wind_speeds, target_value)

#print(result_obs, result_N)
print(f'Measurments under value: {target_value} m/s Obs: {len(result_obs)} NORA3: {l
```

```
Measurments under value: 3 m/s Obs: 1353 NORA3: 528
```

D HYBRID TO HEIGHT IN METERS

This scrips shows how to calculate the model height in meters at a specific location from the models hybridlevel

Convert hybrid to height in meters

this is a script converting the hybrid layer into meters at a given location

In [2]:

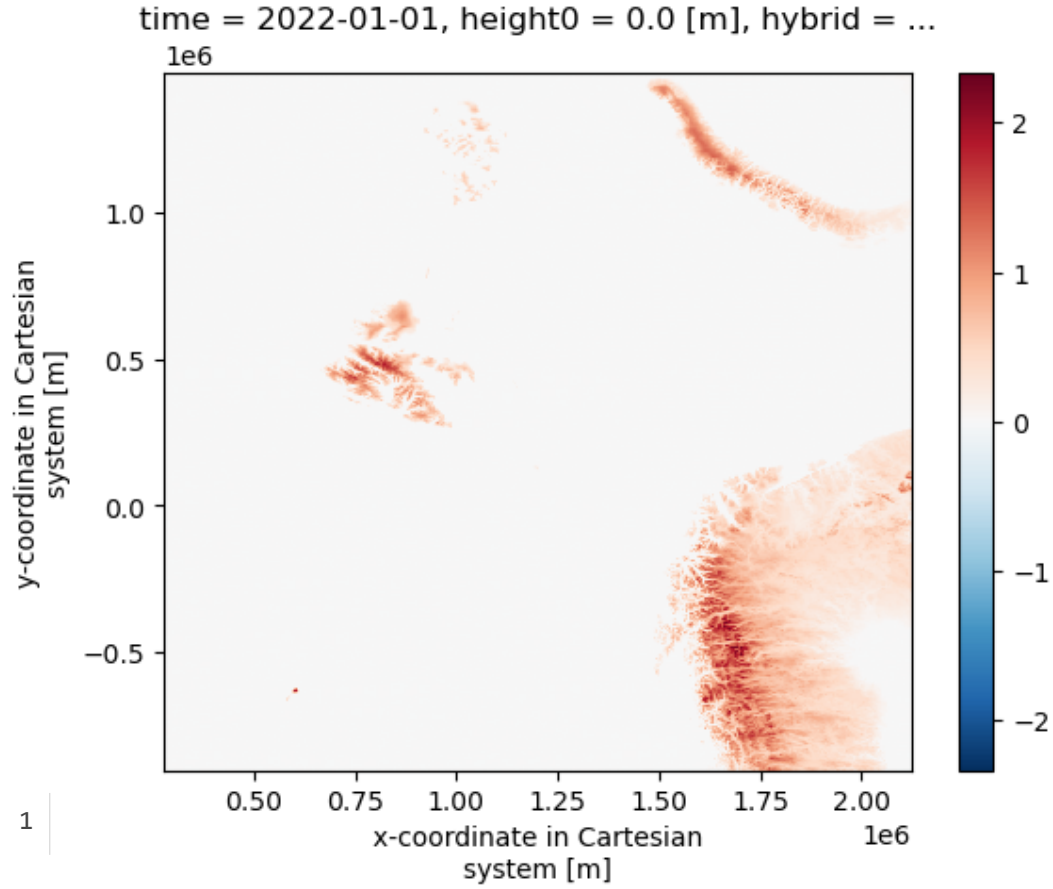
```
import xarray
import numpy as np

import netCDF4 as nc
import matplotlib.pylab as plt

import pyproj



opendap_url = "https://thredds.met.no/thredds/dodsC/aromearcticarchive/"+\
           "2022/01/01/arome_arctic_full_2_5km_20220101T00Z.nc"

#Ask for time=0 to make the calculation smaller, height0=0 to remove the 4th dimens
ds = xarray.open_dataset(opendap_url).isel(time=0,height0=0)

# Formula to calculate pressure from hybrid: p(n,k,j,i) = ap(k) + b(k)*ps(n,j,i)"
ap = ds.ap
b = ds.b
surface_pressure = ds.surface_air_pressure

### test

surface_geo = ds.surface_geopotential


# Note that k = 0 is top of atmosphere (ToA), and k = 64 is surface
pressure_at_k = (ap + (b*surface_pressure))
temperature_at_k = ds.air_temperature_ml
R = 287.058
g = 9.81

height_at_k = xarray.full_like(pressure_at_k, fill_value=0)

max_k = len(height_at_k.hybrid) -1
# Compute the height of the lowest model level
height_at_k.values[max_k,:,:] = surface_geo/g
height_at_k.values[max_k,:,:] *= np.log(surface_pressure / (pressure_at_k.isel(hybr

# Loop over the rest of the model levels
for hybrid in range(max_k - 1,-1,-1):
    height_at_k.values[hybrid,:,:] = (R*temperature_at_k.isel(hybrid=hybrid))/g
    height_at_k.values[hybrid,:,:] *= np.log((pressure_at_k.isel(hybrid = hybrid + 
                                          (pressure_at_k.isel(hybrid = hybrid)))
    height_at_k.values[hybrid,:,:] += height_at_k.isel(hybrid=hybrid + 1)




# Projecting lat lon to x, y

ncfile = nc.Dataset(opendap_url)

crs_AA = pyproj.CRS.from_cf(
         {
            "grid_mapping_name": "lambert_conformal_conic",
            "standard_parallel": [77.5, 77.5],
            "longitude_of_central_meridian": -25.0,
            "latitude_of_projection_origin": 77.5,
```

```
            "earth_radius": 6371000.0,
        }
)

# Transformer to project from ESPG:4368 (WGS:84) to our lambert_conformal_conic
proj = pyproj.Transformer.from_crs(4326,crs_AA,always_xy=True)
# Compute projected coordinates of lat/lon point
lat = 69.67
lon = 22.06
X,Y = proj.transform(lon,lat)

# Find nearest neighbour
x = ncfile.variables["x"][:]
y = ncfile.variables["y"][:]

Ix = np.argmin(np.abs(x - X))
Iy = np.argmin(np.abs(y - Y))


#trekker ut høyde i punktet
H = height_at_k[:,Iy,Ix]

print(H[64])

#####

# finds surface height in the model at location
h2 = ncfile.variables["surface_geopotential"][0,:,:,:]

h2_i_kord = h2[:,Iy,Ix]

høyde = h2_i_kord/9.81 #gir høyde i akk dette punktet! = 866,93moh

# total height m.a.s.l
H1 = H + høyde # H1 kan brukes til å plotte høyde sammen med andre variabler


######
# model height at lowest level

height_at_k.isel(hybrid=64).plot()
```

```
<xarray.DataArray ()>
array(1.28432417)
Coordinates:
    time       datetime64[ns] 2022-01-01
    height0    float32 0.0
    hybrid     float64 0.9985
    x          float32 1.644e+06
    y          float32 -1.779e+05
    longitude  float64 ...
    latitude   float64 ...
```

Out[2]:

```
<matplotlib.collections.QuadMesh at 0x1ceac19da80>
```


time = 2022-01-01, height0 = 0.0 [m], hybrid = ...
1e6
1.0
0.5
0.0
−0.5
y-coordinate in Cartesian system [m]
0.50
0.75
1.00
1.25
1.50
1.75
2.00
1e6
x-coordinate in Cartesian system [m]
2
1
0
−1
−2

1