In [1]:

```
import netCDF4 as nc
import numpy as np
from matplotlib import pyplot as plt
import xarray
import glob
from geopy.distance import geodesic
import datetime as dt
import matplotlib as mpl
from matplotlib.ticker import (MultipleLocator, AutoMinorLocator)
```

Funktionen zum einlesen von Daten

In [2]:

```
time_threshold = 3600 # in seconds
distance_threshold = 50 # in km
```

In [3]:

```
def cut_useless_variables(DS):
    """
    Preprocessing the IASI dataset before concatenation.
    Removing undesired dimensions and variables.

    Parameters:
    -----------
    ds : xarray dataset
        Dataset of IASI data.
    """

    # Remove some nasty variables:
    """
    Remaining variables are:
        pressure_levels_temp
        pressure_levels_humidity
        record_start_time
        record_stop_time
        lat
        lon
        atmospheric_temperature
        atmospheric_water_vapor
        surface_temperature
        surface_pressure
        instrument_mode
        flag_cldnes
        flag_iasibad
        flag_itconv
        flag_landsea
        error_data_index
            # temperature_error
            # water_vapour_error
        surface_z        (useful in combination with surface_pressure)
        co_qflag
        co_bdiv

    Remaining dimensions:
        nlt
        nlq
        along_track
        across_track
            # nerr
            # nerrt
            # nerrw
    """
    useless_vars = ['cloud_formation', 'pressure_levels_ozone', 'surface_emissivity_wave
```

```
lengths',
                    'degraded_ins_MDR', 'degraded_proc_MDR', 'solar_zenith', 'satellite_
zenith',
                    'solar_azimuth', 'satellite_azimuth', 'fg_atmospheric_temperature',
                    'fg_atmospheric_water_vapor', 'fg_atmospheric_ozone', 'fg_surface_te
mperature',
                    'atmospheric_ozone', 'integrated_water_vapor', 'integrated_ozone', '
integrated_n2o',
                    'integrated_co', 'integrated_ch4', 'integrated_co2', 'surface_emissi
vity',
                    'number_cloud_formations', 'fractional_cloud_cover', 'cloud_top_temp
erature',
                    'cloud_top_pressure', 'cloud_phase', 'spacecraft_altitude', 'flag_am
subad',
                    'flag_avhrrbad', 'flag_cdlfrm', 'flag_cdltst', 'flag_daynit', 'flag_
dustcld',
                    'flag_fgcheck', 'flag_initia', 'flag_mhsbad', 'flag_numit', 'flag_nw
pbad',
                    'flag_physcheck', 'flag_retcheck', 'flag_satman', 'flag_sunglnt', 'f
lag_thicir',
                    'nerr_values', 'ozone_error', 'co_npca', 'co_nfitlayers', 'co_nbr_va
lues',
                    'co_cp_air', 'co_cp_co_a', 'co_x_co', 'co_h_eigenvalues', 'co_h_eige
nvectors',
                    'temperature_error', 'water_vapour_error']
    DS = DS.drop_vars(useless_vars)

    # useless_dims = ['npct', 'npcw', 'npco', 'nl_co', 'nl_hno3', 'nl_o3', 'nl_so2', 'new
', 'nlo',
                    # 'cloud_formations', 'nerro', 'co_nbr', 'neva_co', 'neve_co']
    # DS = DS.squeeze(useless_dims, drop=True)

    return DS
```

In [4]:

```
def import_single_NYA_RS_radiosonde(
 filename,
 keys='all',
 verbose=0):

 """
 Imports single NYA-RS radiosonde data for Ny Alesund. Converts to SI units
 and interpolates to a height grid with 5 m resolution from 0 to 15000 m.

 Parameters:
 -----------
 filename : str
  Name (including path) of radiosonde data file.
 keys : list of str or str, optional
  This describes which variable(s) will be loaded. Specifying 'all' will import all varia
bles.
  Specifying 'basic' will load the variables the author consideres most useful for his cu
rrent
  analysis.
  Default: 'all'
 verbose : int
  If 0, output is suppressed. If 1, basic output is printed. If 2, more output (more warn
ings,...)
  is printed.
 """

 """
  Loaded values are imported in the following units:
  T: in K
  P: in hPa, will be converted to Pa
  RH: in [0-1]
  Altitude: in m
  time: will be converted to sec since 1970-01-01 00:00:00 UTC
 """
```

```
 file_nc = nc.Dataset(filename)

 if (not isinstance(keys, str)) and (not isinstance(keys, list)):
  raise TypeError("Argument 'key' must be a list of strings or 'all'.")

 if keys == 'all':
  keys = file_nc.variables.keys()
 elif keys == 'basic':
  keys = ['time', 'temp', 'press', 'rh', 'alt']

 sonde_dict = dict()
 for key in keys:
  if not key in file_nc.variables.keys():
   raise KeyError("I have no memory of this key: '%s'. Key not found in radiosonde file."
% key)

  sonde_dict[key] = np.asarray(file_nc.variables[key])
  if key != "IWV" and len(sonde_dict[key]) == 0: # 'and': second condition only evaluate
d if first condition True
   return None

  if key in ['lat', 'lon']: # only interested in the first lat, lon position
   sonde_dict[key] = sonde_dict[key][0]

 # convert units:
 if 'P' in keys:  # from hPa to Pa
  sonde_dict['P'] = sonde_dict['P']*100
 if 'time' in keys: # from int64 to float64
  time_unit = file_nc.variables['time'].units
  time_offset = (dt.datetime.strptime(time_unit[-19:], "%Y-%m-%dT%H:%M:%S") - dt.datetim
e(1970,1,1)).total_seconds()
  sonde_dict['time'] = np.float64(sonde_dict['time']) + time_offset
  sonde_dict['launch_time'] = sonde_dict['time'][0]

 keys = [*keys]  # converts dict_keys to a list
 for key in keys:
  if sonde_dict[key].shape == sonde_dict['time'].shape:
   if key not in ['time', 'lat', 'lon', 'alt']:
    sonde_dict[key + "_ip"] = np.interp(np.arange(0,15001,5), sonde_dict['alt'], sonde_d
ict[key])
   elif key == 'alt':
    sonde_dict[key + "_ip"] = np.arange(0, 15001,5)


 # Renaming variables to a standard convention
 renaming = {'press': 'pres', 'alt': 'height', 'press_ip': 'pres_ip', 'alt_ip': 'height_
ip'}
 for ren_key in renaming.keys():
  if ren_key in sonde_dict.keys():
   sonde_dict[renaming[ren_key]] = sonde_dict[ren_key]

 return sonde_dict


def import_single_PS122_mosaic_radiosonde_level2(
 filename,
 keys='all',
 verbose=0):

 """
 Imports single level 2 radiosonde data created with PANGAEA_tab_to_nc.py
 ('PS122_mosaic_radiosonde_level2_yyyymmdd_hhmmssZ.nc'). Converts to SI units
 and interpolates to a height grid with 5 m resolution from 0 to 15000 m.

 Parameters:
 -----------
 filename : str
  Name (including path) of radiosonde data file.
 keys : list of str or str, optional
  This describes which variable(s) will be loaded. Specifying 'all' will import all varia
bles.
  Specifying 'basic' will load the variables the author consideres most useful for his cu
```

```
rrent
  analysis.
  Default: 'all'
 verbose : int
  If 0, output is suppressed. If 1, basic output is printed. If 2, more output (more warn
ings,...)
  is printed.
 """

 """
  Loaded values are imported in the following units:
  T: in deg C, will be converted to K
  P: in hPa, will be converted to Pa
  RH: in %, will be converted to [0-1]
  Altitude: in m
  q: in kg kg^-1 (water vapor specific humidity)
  time: in sec since 1970-01-01 00:00:00 UTC
 """

 file_nc = nc.Dataset(filename)

 if (not isinstance(keys, str)) and (not isinstance(keys, list)):
  raise TypeError("Argument 'key' must be a list of strings or 'all'.")

 if keys == 'all':
  keys = file_nc.variables.keys()
 elif keys == 'basic':
  keys = ['time', 'T', 'P', 'RH', 'q', 'Altitude']

 sonde_dict = dict()
 for key in keys:
  if not key in file_nc.variables.keys():
   raise KeyError("I have no memory of this key: '%s'. Key not found in radiosonde file."
% key)

  sonde_dict[key] = np.asarray(file_nc.variables[key])
  if key != "IWV" and len(sonde_dict[key]) == 0: # 'and': second condition only evaluate
d if first condition True
   return None

  if key in ['Latitude', 'Longitude']: # only interested in the first lat, lon position
   sonde_dict[key] = sonde_dict[key][0]
  if key == 'IWV':
   sonde_dict[key] = np.float64(sonde_dict[key])

 # convert units:
 if 'RH' in keys: # from percent to [0, 1]
  sonde_dict['RH'] = sonde_dict['RH']*0.01
 if 'T' in keys:  # from deg C to K
  sonde_dict['T'] = sonde_dict['T'] + 273.15
 if 'P' in keys:  # from hPa to Pa
  sonde_dict['P'] = sonde_dict['P']*100
 if 'time' in keys: # from int64 to float64
  sonde_dict['time'] = np.float64(sonde_dict['time'])
  sonde_dict['launch_time'] = sonde_dict['time'][0]

 keys = [*keys]  # converts dict_keys to a list
 for key in keys:
  if sonde_dict[key].shape == sonde_dict['time'].shape:
   if key not in ['time', 'Latitude', 'Longitude', 'ETIM', 'Altitude']:
    sonde_dict[key + "_ip"] = np.interp(np.arange(0,15001,5), sonde_dict['Altitude'], so
nde_dict[key])
   elif key == 'Altitude':
    sonde_dict[key + "_ip"] = np.arange(0, 15001,5)


 # Renaming variables: ['Lat', 'Lon', 'p', 'T', 'RH', 'GeopHgt', 'qv', 'time', ...]
 renaming = {'T': 'temp',  'P': 'pres',  'RH': 'rh',
    'Altitude': 'height', 'h_geom': 'height_geom',
    'Latitude': 'lat',  'Longitude': 'lon',
    'T_ip': 'temp_ip', 'P_ip': 'pres_ip', 'RH_ip': 'rh_ip',
    'Altitude_ip': 'height_ip', 'h_geom_ip': 'height_geom_ip',
```

```
    'IWV': 'iwv'}
 for ren_key in renaming.keys():
  if ren_key in sonde_dict.keys():
   sonde_dict[renaming[ren_key]] = sonde_dict[ren_key]

 return sonde_dict

##!!!

def import_radiosonde_daterange(
 path_data,
 date_start,
 date_end,
 s_version='level_2',
 with_wind=False,
 verbose=0):

 """
 Imports radiosonde data 'mossonde-curM1' and concatenates the files into time series x h
eight.
 E.g. temperature profile will have the dimension: n_sondes x n_height

 Parameters:
 -----------
 path_data : str
  Path of radiosonde data.
 date_start : str
  Marks the first day of the desired period. To be specified in yyyy-mm-dd (e.g. 2021-01-
14)!
 date_end : str
  Marks the last day of the desired period. To be specified in yyyy-mm-dd (e.g. 2021-01-1
4)!
 s_version : str, optional
  Specifies the radiosonde version that is to be imported. Possible options: 'mossonde',
  'psYYMMDDwHH', 'level_2'. Default: 'level_2' (published by Marion Maturilli)
 with_wind : bool, optional
  This describes if wind measurements are included (True) or not (False). Does not work w
ith
  s_version='psYYMMDDwHH'. Default: False
 verbose : int, optional
  If 0, output is suppressed. If 1, basic output is printed. If 2, more output (more warn
ings,...)
  is printed.
 """

 if not isinstance(s_version, str): raise TypeError("s_version in import_radiosonde_dater
ange must be a string.")

 # extract day, month and year from start date:
 date_start = dt.datetime.strptime(date_start, "%Y-%m-%d")
 date_end = dt.datetime.strptime(date_end, "%Y-%m-%d")

 if s_version == 'level_2':
  all_radiosondes_nc = sorted(glob.glob(path_data + "PS122_mosaic_radiosonde_level2*.nc")
)

  # inquire the number of radiosonde files (date and time of launch is in filename):
  # And fill a list which will include the relevant radiosonde files.
  radiosondes_nc = []
  for rs_nc in all_radiosondes_nc:
   rs_date = rs_nc[-19:-3]  # date of radiosonde from filename
   yyyy = int(rs_date[:4])
   mm = int(rs_date[4:6])
   dd = int(rs_date[6:8])
   rs_date_dt = dt.datetime(yyyy,mm,dd)
   if rs_date_dt >= date_start and rs_date_dt <= date_end:
    radiosondes_nc.append(rs_nc)

 elif s_version == 'nya-rs':
  all_radiosondes_nc = sorted(glob.glob(path_data + "NYA-RS_*.nc"))

  # inquire the number of radiosonde files (date and time of launch is in filename):
```

```
  # And fill a list which will include the relevant radiosonde files.
  radiosondes_nc = []
  for rs_nc in all_radiosondes_nc:
   rs_date = rs_nc[-15:-3]  # date of radiosonde from filename
   yyyy = int(rs_date[:4])
   mm = int(rs_date[4:6])
   dd = int(rs_date[6:8])
   rs_date_dt = dt.datetime(yyyy,mm,dd)
   if rs_date_dt >= date_start and rs_date_dt <= date_end:
    radiosondes_nc.append(rs_nc)


 # number of sondes:
 n_sondes = len(radiosondes_nc)

 # count the number of days between start and end date as max. array size:
 n_days = (date_end - date_start).days

 # basic variables that should always be imported:
 if s_version == 'level_2':
  geoinfo_keys = ['lat', 'lon', 'launch_time', 'iwv']
  time_height_keys = ['pres', 'temp', 'rh', 'height', 'rho_v', 'q']
  if with_wind: time_height_keys = time_height_keys + ['wspeed', 'wdir']

 elif s_version == 'nya-rs':
  geoinfo_keys = ['lat', 'lon', 'launch_time']
  time_height_keys = ['pres', 'temp', 'rh', 'height']
  if with_wind: time_height_keys = time_height_keys + ['wspeed', 'wdir']
 else:
  raise ValueError("s_version in import_radiosonde_daterange must be 'nya-rs' or 'level_2
'.")
 all_keys = geoinfo_keys + time_height_keys

 # sonde_master_dict (output) will contain all desired variables on specific axes:
 # Time axis (one sonde = 1 timestamp) = axis 0; height axis = axis 1
 n_height = len(np.arange(0,15001,5)) # length of the interpolated height grid
 sonde_master_dict = dict()
 for gk in geoinfo_keys: sonde_master_dict[gk] = np.full((n_sondes,), np.nan)
 for thk in time_height_keys: sonde_master_dict[thk] = np.full((n_sondes, n_height), np.
nan)

 if s_version == 'level_2':
  all_keys_import = ['Latitude', 'Longitude', 'P', 'T', 'RH', 'Altitude', 'rho_v', 'q',
'time', 'IWV']
  if with_wind: all_keys_import = all_keys_import + ['wdir', 'wspeed']


  # cycle through all relevant sonde files:
  for rs_idx, rs_nc in enumerate(radiosondes_nc):

   if verbose >= 1:
    # rs_date = rs_nc[-19:-3]
    print("Working on Radiosonde, " + rs_nc)

   sonde_dict = import_single_PS122_mosaic_radiosonde_level2(rs_nc, keys=all_keys_import)

   # save to sonde_master_dict:
   for key in all_keys:
    if key in geoinfo_keys:
     sonde_master_dict[key][rs_idx] = sonde_dict[key]

    elif key in time_height_keys:
     sonde_master_dict[key][rs_idx, :] = sonde_dict[key + "_ip"]  # must use the interpo
lated versions!

    else:
     raise KeyError("Key '" + key + "' not found in radiosonde dictionary after importing
it with " +
         "import_single_PS122_mosaic_radiosonde_level2")

 if s_version == 'nya-rs':
  all_keys_import = ['lat', 'lon', 'press', 'temp', 'rh', 'alt', 'time']
```

```
  if with_wind: all_keys_import = all_keys_import + ['wdir', 'wspeed']


  # cycle through all relevant sonde files:
  for rs_idx, rs_nc in enumerate(radiosondes_nc):

   if verbose >= 1:
    # rs_date = rs_nc[-19:-3]
    print("Working on Radiosonde, " + rs_nc)

   sonde_dict = import_single_NYA_RS_radiosonde(rs_nc, keys=all_keys_import)

   # save to sonde_master_dict:
   for key in all_keys:
    if key in geoinfo_keys:
     sonde_master_dict[key][rs_idx] = sonde_dict[key]

    elif key in time_height_keys:
     sonde_master_dict[key][rs_idx, :] = sonde_dict[key + "_ip"]  # must use the interpo
lated versions!

    else:
     raise KeyError("Key '" + key + "' not found in radiosonde dictionary after importing
it with " +
         "import_single_NYA_RS_radiosonde")

 return sonde_master_dict
```

In [5]:

```
def numpydatetime64_to_epochtime(npdt_array):

    """
    Converts numpy datetime64 array to array in seconds since 1970-01-01 00:00:00 UTC (ty
pe:
    float).

    Parameters:
    -----------
    npdt_array : numpy array of type np.datetime64 or np.datetime64 type
        Array (1D) or directly a np.datetime64 type variable.
    """

    sec_epochtime = npdt_array.astype(np.timedelta64) / np.timedelta64(1, 's')

    return sec_epochtime
```

In [6]:

```
def create_launch_time(DS):

    time_dif = np.diff(DS.time.values)
    where_jump = np.argwhere(np.abs(time_dif) > time_threshold).flatten()
    launch_time = np.concatenate((np.array([DS.time.values[0]]), DS.time.values[where_ju
mp+1]))

    return xarray.DataArray(launch_time)
```

Daten einlesen

In [7]:

```
#NyAlesund
path_data = "/Users/charlottebaur/Desktop/Project work/NyAlesund/"
NyAl_sonde_dict = import_radiosonde_daterange(path_data, date_start="2020-08-01", date_en
d="2020-09-30", s_version='nya-rs', with_wind=False)

#IASI
IASI_DS = xarray.open_mfdataset('/Users/charlottebaur/Desktop/Project work/METRS_IASI/*.n
c',concat_dim='along_track', combine='nested', decode_times=False, preprocess=cut_useles
```

```
s_variables)

#Polarstern
path_data = "/Users/charlottebaur/Desktop/Project work/Polarstern/"
date_start = "2020-08-01"  # in yyyy-mm-dd
date_end = "2020-09-30"   # in yyyy-mm-dd
s_version = 'level_2'

PS_sonde_dict = import_radiosonde_daterange(path_data, date_start, date_end, s_version, w
ith_wind=False)
```

```
<ipython-input-4-3e5033df8d41>:59: DeprecationWarning: tostring() is deprecated. Use toby
tes() instead.
  time_unit = file_nc.variables['time'].units
/Users/charlottebaur/opt/anaconda3/lib/python3.8/site-packages/xarray/conventions.py:512:
SerializationWarning: variable 'temperature_error' has multiple fill values {-2147483646,
4294967295}, decoding all values to NaN.
  new_vars[k] = decode_cf_variable(
/Users/charlottebaur/opt/anaconda3/lib/python3.8/site-packages/xarray/conventions.py:512:
SerializationWarning: variable 'water_vapour_error' has multiple fill values {-2147483646
, 4294967295}, decoding all values to NaN.
  new_vars[k] = decode_cf_variable(
/Users/charlottebaur/opt/anaconda3/lib/python3.8/site-packages/xarray/conventions.py:512:
SerializationWarning: variable 'ozone_error' has multiple fill values {-2147483646, 42949
67295}, decoding all values to NaN.
  new_vars[k] = decode_cf_variable(
/Users/charlottebaur/opt/anaconda3/lib/python3.8/site-packages/xarray/conventions.py:512:
SerializationWarning: variable 'surface_z' has multiple fill values {-32767, 32767}, deco
ding all values to NaN.
  new_vars[k] = decode_cf_variable(
/Users/charlottebaur/opt/anaconda3/lib/python3.8/site-packages/xarray/conventions.py:512:
SerializationWarning: variable 'co_cp_air' has multiple fill values {-2, 65535}, decoding
all values to NaN.
  new_vars[k] = decode_cf_variable(
/Users/charlottebaur/opt/anaconda3/lib/python3.8/site-packages/xarray/conventions.py:512:
SerializationWarning: variable 'co_cp_co_a' has multiple fill values {-2, 65535}, decodin
g all values to NaN.
  new_vars[k] = decode_cf_variable(
```

In [15]:

```
print(PS_sonde_dict)
```

```
{'lat': array([78.829, 78.806, 78.835, 78.968, 79.017, 79.037, 79.114, 78.71 ,
       78.656, 78.649, 78.65 , 79.022, 79.047, 79.024, 79.018, 78.812,
       78.714, 78.591, 78.525, 78.42 , 78.386, 78.352, 78.356, 78.796,
       79.41 , 79.753, 79.777, 79.905, 79.881, 79.9  , 79.892, 79.916,
       79.883, 79.882, 79.879, 79.879, 79.908, 79.902, 79.917, 79.921,
       79.946, 79.955, 79.988, 80.001, 80.03 , 80.069, 80.116, 80.182,
       80.233, 80.562, 81.054, 81.277, 81.61 , 82.403, 82.928, 83.397,
       83.992, 84.56 , 85.13 , 85.491, 85.899, 86.36 , 86.662, 86.945,
       87.433, 87.778, 88.118, 88.296, 88.502, 88.916, 88.935, 89.114,
       89.342, 89.644, 89.996, 89.992, 89.821, 89.394, 88.991, 88.581,
       88.154, 87.737, 87.718, 87.717, 87.722, 87.729, 87.749, 87.756,
       87.744, 87.745, 87.757, 87.772, 87.791, 87.798, 87.774, 87.763,
       87.767, 87.776, 87.78 , 87.784, 87.801, 87.827, 87.843, 87.88 ,
       87.906, 87.938, 87.95 , 87.98 , 87.996, 88.025, 88.038, 88.058,
       88.068, 88.093, 88.112, 88.145, 88.168, 88.204, 88.228, 88.253,
       88.266, 88.284, 88.294, 88.311, 88.326, 88.34 , 88.354, 88.375,
       88.392, 88.416, 88.442, 88.474, 88.509, 88.553, 88.591, 88.634,
       88.67 , 88.698, 88.721, 88.743, 88.749, 88.75 , 88.746, 88.742,
       88.733, 88.737, 88.74 , 88.743, 88.752, 88.761, 88.758, 88.746,
       88.734, 88.726, 88.711, 88.696, 88.688, 88.685, 88.675, 88.676,
       88.676, 88.696, 88.71 , 88.729, 88.725, 88.724, 88.71 , 88.713,
       88.708, 88.713, 88.717, 88.735, 88.743, 88.76 , 88.76 , 88.77 ,
       88.772, 88.787, 88.793, 88.811, 88.831, 88.85 , 88.873, 88.908,
       88.935, 88.945, 88.959, 88.987, 89.005, 89.003, 89.009, 89.037,
       89.055, 89.053, 89.047, 89.06 , 89.072, 89.063, 89.079, 89.07 ,
       89.09 , 89.076, 89.07 , 89.034, 89.028, 89.016, 89.031, 89.039,
       89.054, 89.065, 89.075, 89.102, 89.117, 89.153, 89.139, 89.151,
       89.135, 89.146, 89.132, 89.01 , 88.779, 88.595, 88.509, 88.313,
       87.928, 87.502, 87.007, 86.782, 86.555, 86.334, 86.192, 86.005,
```

```
       86.036, 85.974, 85.959, 85.959, 85.98 , 85.984, 85.964, 85.986,
       85.996, 86.01 , 86.012, 86.032, 86.046, 85.979, 85.949, 85.994,
       86.001, 85.958, 85.961, 85.983, 85.979, 85.965, 86.031, 86.032,
       85.964, 85.388, 84.742, 84.59 , 84.417, 84.205, 83.942, 83.73 ,
       83.523, 83.228, 83.253, 83.255, 83.301, 83.157, 82.996, 82.542,
       82.246, 81.667, 81.597, 81.582, 81.589, 81.604, 81.617, 81.625,
       81.775, 81.71 , 81.679, 81.535, 81.318, 81.125, 80.965]), 'lon': array([ -2.234,
-2.41 ,  -2.398,  -2.535,  -2.591,  -2.726,  -2.604,
        -4.622,  -5.038,  -4.296,  -4.244,  -2.965,  -3.3  ,  -3.211,
        -3.445,  -1.676,  -1.16 ,  -1.578,  -2.944,  -6.046,  -7.04 ,
        -7.249,  -7.338,  -6.598,  -5.743,  -5.141,  -4.65 ,  -3.555,
        -3.531,  -3.771,  -3.869,  -4.062,  -4.169,  -4.637,  -4.857,
        -5.237,  -5.306,  -5.477,  -5.541,  -5.54 ,  -5.635,  -5.738,
        -5.891,  -5.94 ,  -6.043,  -6.142,  -6.221,  -6.203,  -6.104,
        -6.347,  -6.934,  -6.483,  -6.134,  -6.591,  -7.92 , -11.76 ,
       -13.4  , -14.285, -15.595, -16.544, -17.791, -20.306, -28.299,
       -35.713, -39.095, -40.265, -37.714, -34.72 , -32.753, -35.557,
       -35.97 , -35.37 , -36.887, -32.423, -77.152, 131.791, 104.918,
       106.547, 103.935, 103.49 , 104.21 , 104.641, 104.488, 104.252,
       104.235, 104.258, 104.497, 104.907, 105.324, 105.589, 105.981,
       106.195, 106.244, 106.023, 105.727, 105.344, 105.152, 104.93 ,
       104.562, 104.267, 104.174, 104.554, 105.077, 105.743, 106.377,
       107.056, 107.622, 108.125, 108.412, 108.668, 108.892, 109.04 ,
       109.108, 109.25 , 109.496, 109.803, 110.451, 111.169, 112.204,
       112.877, 113.513, 113.744, 114.128, 114.248, 114.639, 114.934,
       115.648, 116.302, 116.923, 117.187, 117.716, 118.259, 118.891,
       119.445, 120.065, 120.265, 120.68 , 120.664, 120.707, 120.237,
       119.779, 119.468, 119.379, 119.287, 119.148, 118.765, 118.556,
       118.021, 117.106, 116.016, 114.883, 113.634, 112.477, 112.123,
       111.998, 111.846, 111.568, 111.623, 112.071, 112.298, 112.943,
       113.197, 113.657, 112.877, 111.543, 109.955, 108.936, 107.669,
       107.045, 106.324, 105.605, 104.712, 104.02 , 103.168, 102.603,
       101.656, 101.071, 100.336, 100.247,  99.105,  98.505,  97.872,
        96.529,  95.77 ,  96.111,  96.468,  96.316,  97.075,  99.316,
       101.1  , 101.749, 102.956, 104.827, 106.814, 107.121, 106.907,
       107.437, 107.373, 107.426, 107.998, 108.678, 109.114, 109.147,
       109.144, 108.175, 107.476, 107.163, 107.361, 107.035, 107.198,
       107.014, 107.099, 107.553, 109.181, 110.572, 111.537, 111.55 ,
       111.15 , 110.752, 109.044, 108.379, 105.99 , 105.537, 104.97 ,
        99.679,  99.808,  98.134,  97.855,  95.954,  92.276,  88.923,
        86.875,  85.063,  83.172,  79.045,  75.461,  72.125,  69.103,
        66.224,  61.772,  59.505,  59.498,  59.434,  59.556,  59.628,
        59.239,  55.908,  52.999,  50.192,  48.46 ,  46.858,  43.763,
        40.52 ,  37.403,  35.597,  35.501,  31.917,  31.791,  31.705,
        31.804,  31.81 ,  32.205,  32.183,  32.115,  32.066,  32.055,
        32.029,  32.104,  26.751,  25.14 ,  23.018,  19.351,  15.481,
        12.093,  11.589,  10.632,   9.572,   7.892,   6.415,   5.037,
         2.434,   1.508,   1.316,   1.908,   2.722,   3.481,   4.161]), 'launch_time': a
rray([1.59625777e+09, 1.59627930e+09, 1.59630095e+09, 1.59632248e+09,
       1.59634404e+09, 1.59636572e+09, 1.59638729e+09, 1.59641080e+09,
       1.59643049e+09, 1.59645208e+09, 1.59647373e+09, 1.59649529e+09,
       1.59651698e+09, 1.59653854e+09, 1.59656002e+09, 1.59658193e+09,
       1.59660329e+09, 1.59662507e+09, 1.59664655e+09, 1.59666808e+09,
       1.59668970e+09, 1.59671165e+09, 1.59673377e+09, 1.59675456e+09,
       1.59677612e+09, 1.59679768e+09, 1.59681928e+09, 1.59684110e+09,
       1.59686244e+09, 1.59688412e+09, 1.59690568e+09, 1.59692743e+09,
       1.59694890e+09, 1.59697067e+09, 1.59699212e+09, 1.59701394e+09,
       1.59703548e+09, 1.59705684e+09, 1.59707863e+09, 1.59710018e+09,
       1.59712164e+09, 1.59714324e+09, 1.59716488e+09, 1.59718654e+09,
       1.59720813e+09, 1.59722972e+09, 1.59725128e+09, 1.59727298e+09,
       1.59729465e+09, 1.59731614e+09, 1.59733774e+09, 1.59734846e+09,
       1.59735946e+09, 1.59738088e+09, 1.59740264e+09, 1.59742413e+09,
       1.59744568e+09, 1.59746730e+09, 1.59748900e+09, 1.59751050e+09,
       1.59753309e+09, 1.59755372e+09, 1.59757538e+09, 1.59759691e+09,
       1.59761848e+09, 1.59764007e+09, 1.59766181e+09, 1.59768329e+09,
       1.59770503e+09, 1.59772650e+09, 1.59774807e+09, 1.59776976e+09,
       1.59779143e+09, 1.59781300e+09, 1.59783553e+09, 1.59785641e+09,
       1.59787795e+09, 1.59790082e+09, 1.59792092e+09, 1.59794250e+09,
       1.59796407e+09, 1.59798570e+09, 1.59800713e+09, 1.59802897e+09,
       1.59805051e+09, 1.59807213e+09, 1.59809374e+09, 1.59811531e+09,
       1.59813704e+09, 1.59815851e+09, 1.59818019e+09, 1.59820174e+09,
       1.59822325e+09, 1.59824509e+09, 1.59826652e+09, 1.59828817e+09,
```

```
       1.59830970e+09, 1.59833128e+09, 1.59835317e+09, 1.59837462e+09,
       1.59839618e+09, 1.59841770e+09, 1.59843921e+09, 1.59846097e+09,
       1.59848253e+09, 1.59850413e+09, 1.59852576e+09, 1.59854732e+09,
       1.59856894e+09, 1.59859056e+09, 1.59861213e+09, 1.59863372e+09,
       1.59865536e+09, 1.59867695e+09, 1.59869861e+09, 1.59872005e+09,
       1.59874174e+09, 1.59876338e+09, 1.59878490e+09, 1.59880654e+09,
       1.59882811e+09, 1.59884970e+09, 1.59887136e+09, 1.59889312e+09,
       1.59891455e+09, 1.59893617e+09, 1.59895772e+09, 1.59897945e+09,
       1.59900111e+09, 1.59902260e+09, 1.59904412e+09, 1.59906583e+09,
       1.59908732e+09, 1.59910902e+09, 1.59913065e+09, 1.59915220e+09,
       1.59917377e+09, 1.59919533e+09, 1.59921698e+09, 1.59923898e+09,
       1.59926031e+09, 1.59928170e+09, 1.59930330e+09, 1.59932517e+09,
       1.59934650e+09, 1.59936813e+09, 1.59937893e+09, 1.59938970e+09,
       1.59940097e+09, 1.59941149e+09, 1.59942283e+09, 1.59943330e+09,
       1.59944374e+09, 1.59945461e+09, 1.59946574e+09, 1.59947648e+09,
       1.59948796e+09, 1.59949789e+09, 1.59951922e+09, 1.59954116e+09,
       1.59956256e+09, 1.59958425e+09, 1.59960571e+09, 1.59962726e+09,
       1.59964899e+09, 1.59967062e+09, 1.59969229e+09, 1.59971375e+09,
       1.59973576e+09, 1.59975700e+09, 1.59977859e+09, 1.59980030e+09,
       1.59982192e+09, 1.59984329e+09, 1.59986502e+09, 1.59988676e+09,
       1.59990807e+09, 1.59992972e+09, 1.59994055e+09, 1.59996262e+09,
       1.59997304e+09, 1.59998376e+09, 1.59999489e+09, 1.60000566e+09,
       1.60001619e+09, 1.60002698e+09, 1.60003778e+09, 1.60004930e+09,
       1.60005982e+09, 1.60007018e+09, 1.60008093e+09, 1.60009187e+09,
       1.60010322e+09, 1.60011382e+09, 1.60012405e+09, 1.60013461e+09,
       1.60014624e+09, 1.60016732e+09, 1.60018899e+09, 1.60021055e+09,
       1.60023221e+09, 1.60025380e+09, 1.60027534e+09, 1.60029701e+09,
       1.60031849e+09, 1.60034014e+09, 1.60036177e+09, 1.60038330e+09,
       1.60039640e+09, 1.60040515e+09, 1.60042659e+09, 1.60044812e+09,
       1.60046973e+09, 1.60049136e+09, 1.60051317e+09, 1.60053887e+09,
       1.60055611e+09, 1.60057768e+09, 1.60059943e+09, 1.60062093e+09,
       1.60064247e+09, 1.60066406e+09, 1.60068578e+09, 1.60070737e+09,
       1.60072893e+09, 1.60075035e+09, 1.60077201e+09, 1.60078312e+09,
       1.60079402e+09, 1.60080497e+09, 1.60081558e+09, 1.60082644e+09,
       1.60083699e+09, 1.60084769e+09, 1.60085845e+09, 1.60086952e+09,
       1.60088010e+09, 1.60089088e+09, 1.60090173e+09, 1.60091304e+09,
       1.60092330e+09, 1.60093455e+09, 1.60094583e+09, 1.60095581e+09,
       1.60096647e+09, 1.60097752e+09, 1.60098817e+09, 1.60099862e+09,
       1.60100969e+09, 1.60103126e+09, 1.60105289e+09, 1.60106368e+09,
       1.60107456e+09, 1.60108515e+09, 1.60109610e+09, 1.60111757e+09,
       1.60113945e+09, 1.60116106e+09, 1.60118238e+09, 1.60119093e+09,
       1.60119694e+09, 1.60120403e+09, 1.60121202e+09, 1.60121850e+09,
       1.60122440e+09, 1.60123365e+09, 1.60124016e+09, 1.60124717e+09,
       1.60126889e+09, 1.60129026e+09, 1.60131252e+09, 1.60133372e+09,
       1.60135525e+09, 1.60137706e+09, 1.60138780e+09, 1.60139404e+09,
       1.60139876e+09, 1.60140683e+09, 1.60141339e+09, 1.60142021e+09,
       1.60144164e+09, 1.60146322e+09, 1.60147850e+09, 1.60148493e+09,
       1.60149228e+09, 1.60149965e+09, 1.60150646e+09]), 'iwv': array([22.43222455, 21.95
627232, 22.08131509, 20.97352597, 23.7312653 ,
       25.69425649, 28.81242031, 29.70197522, 29.54071813, 28.71753044,
       28.56985519, 26.25266957, 23.79209625, 17.70874098, 17.29066776,
       17.74563937, 18.40694312, 20.02381401, 16.49162057, 16.08904303,
       18.2424856 , 18.28387616, 16.45064402, 18.94614663, 16.66290787,
       17.06205005, 17.5778332 , 17.34467223, 16.1886767 , 23.67291786,
       24.67460619, 19.37407103, 28.11662954, 27.78274526, 20.87990224,
       24.51724756, 18.45641873, 14.95367803, 13.20376581, 12.6280363 ,
       11.82589711, 14.12467006, 19.50948767, 20.94014565, 22.03793586,
       22.42804193, 23.98050838, 23.37417813,  9.07101001, 12.03166146,
       12.94620866, 11.83749788, 13.23988704, 14.04405579, 19.95267762,
       20.61120883, 13.37930204, 14.7972951 , 13.31130186, 22.71549429,
       14.40088607, 12.51271522, 16.39105067, 17.48720779, 11.88098432,
       17.54694428, 13.87613141, 18.51623073, 19.41662905, 19.16303615,
       17.30699983, 16.8482537 , 15.11942819, 12.78737172, 13.1929759 ,
       13.5570687 , 13.18109153, 13.99580519, 14.40260407, 17.36413412,
       17.61412206, 16.87447966, 12.89401471, 11.57453203, 12.09323182,
       16.10492021, 19.47425587, 11.38064933, 10.46544864, 12.31015755,
       11.90492488, 13.5492894 , 12.9895938 , 10.15292017,  9.0018679 ,
        8.32342352,  9.01610878,  8.67777989,  9.98711314, 15.14906125,
       20.04514753, 17.44872592, 13.74859011, 13.96333244, 16.49646849,
       19.85853207, 17.29272706, 17.60035918, 16.81629504, 18.16295373,
       14.32566934, 14.14347629, 14.03086828, 15.92603293, 15.94587613,
       14.52133943, 14.16492778, 14.1182406 , 13.20591219, 13.90128828,
```

```
       14.16298581, 14.2567637 , 13.94908714, 13.66055884, 12.82278875,
       12.66394442, 11.53464984, 10.97698043, 11.18103853, 11.13705387,
       11.8912602 ,  9.78060391,  9.31680138,  9.97871589,  9.61284443,
        9.94923603, 10.67973481,  9.24556863,  9.07351772,  9.35678668,
        9.18560004,  8.56804989,  8.14987599,  7.69946285,  6.79186525,
        7.18921944,  7.67251591,  9.83420384, 10.53036902, 11.89508411,
       10.94347718,  8.53517381,  6.20207562,  6.68008971,  4.98684368,
        4.99028526,  5.06098045,  5.91317185,  8.0637836 ,  7.89379059,
        8.28444444,  9.17168987, 11.80721675, 11.42329195,  8.40106344,
        5.02751725,  4.40988286,  4.59862603,  4.82146886,  4.76646867,
        4.65720556,  4.87470281,  5.017061  ,  5.84041042,  5.92081915,
        5.48028472,  5.83883757,  6.13199066,  7.77334742,  9.49524672,
       13.73901154, 15.91749756, 18.0160245 , 17.2584508 , 18.40987038,
       10.18035194, 11.53893608, 14.11767476, 14.03150439, 12.81310397,
       12.03731903, 12.04242666,  9.7226601 ,  9.60851298,  9.22467537,
        9.74144612, 10.24078759,  9.2668792 ,  8.21958008,  7.86903867,
        8.28559652,  9.03180389,  8.36457462,  7.20667533,  5.0825794 ,
        5.28860269,  5.21110017,  5.42947751,  6.49763257,  6.29005191,
        7.19875831,  5.85014039,  7.8881543 ,  7.62959835,  4.08490997,
        8.08053287,  7.69751028,  4.89541145,  6.74207634,  7.2379082 ,
        7.48469656,  7.74721072,  6.92998094,  6.50275673,  5.67253551,
        5.50492724,  5.19446151,  4.76151622,  7.73962535, 11.70022613,
       15.48567443, 14.00717427, 13.84353356, 13.36865684,  7.81161712,
        9.10959277,  7.31693705,  8.68481189, 11.42602154, 11.01531841,
       10.33004988, 10.85799158, 10.10390346,  9.73444261,  9.24317221,
        8.8556387 ,  8.49413829,  8.65572989,  8.53451248,  9.86142501,
        9.26918976,  8.17966369,  7.70373234,  8.23179615,  9.02713709,
        8.87157991,  8.10737679,  7.93487933,  7.41598492,  7.17139477,
        6.94009101,  6.34249085,  5.74786734,  5.57856404,  5.02043083,
        3.16815433,  3.32183241,  3.62167854,  3.41320227,  4.94664605,
        5.59681824,  5.68936481,  5.88371439, 11.5263742 , 12.43298431,
       12.5756086 , 12.53970932, 12.89041929, 12.55947488, 11.68826034,
        8.78863916,  7.40816518,  7.26946426,  7.12387929,  6.91019784,
        6.26134583,  5.92378209]), 'pres': array([[101469.  , 101469.  , 101469.  , ...,
12459.  ,  12448.25,
         12439.6 ],
       [101479.  , 101479.  , 101479.  , ...,  12483.  ,  12472.75,
         12463.  ],
       [101399.  , 101399.  , 101399.  , ...,  12501.  ,  12490.25,
         12481.6 ],
       ...,
       [101757.  , 101757.  , 101757.  , ...,  11701.2 ,  11692.2 ,
         11683.2 ],
       [101892.  , 101892.  , 101892.  , ...,  11708.  ,  11698.4 ,
         11689.8 ],
       [101962.  , 101962.  , 101962.  , ...,  11712.5 ,  11705.  ,
         11690.2 ]]), 'temp': array([[275.85, 275.85, 275.85, ..., 225.55, 225.55, 225.55
],
       [276.95, 276.95, 276.95, ..., 225.45, 225.45, 225.45],
       [277.05, 277.05, 277.05, ..., 224.55, 224.55, 224.55],
       ...,
       [267.25, 267.25, 267.25, ..., 222.07, 222.17, 222.25],
       [268.15, 268.15, 268.15, ..., 222.95, 222.95, 222.95],
       [269.05, 269.05, 269.05, ..., 223.75, 223.75, 223.75]]), 'rh': array([[0.99, 0.99,
0.99, ..., 0.01, 0.01, 0.01],
       [1.  , 1.  , 1.  , ..., 0.01, 0.01, 0.01],
       [1.  , 1.  , 1.  , ..., 0.01, 0.01, 0.01],
       ...,
       [0.95, 0.95, 0.95, ..., 0.01, 0.01, 0.01],
       [0.98, 0.98, 0.98, ..., 0.01, 0.01, 0.01],
       [0.94, 0.94, 0.94, ..., 0.01, 0.01, 0.01]]), 'height': array([[0.0000e+00, 5.0000e
+00, 1.0000e+01, ..., 1.4990e+04, 1.4995e+04,
        1.5000e+04],
       [0.0000e+00, 5.0000e+00, 1.0000e+01, ..., 1.4990e+04, 1.4995e+04,
        1.5000e+04],
       [0.0000e+00, 5.0000e+00, 1.0000e+01, ..., 1.4990e+04, 1.4995e+04,
        1.5000e+04],
       ...,
       [0.0000e+00, 5.0000e+00, 1.0000e+01, ..., 1.4990e+04, 1.4995e+04,
        1.5000e+04],
       [0.0000e+00, 5.0000e+00, 1.0000e+01, ..., 1.4990e+04, 1.4995e+04,
        1.5000e+04],
```

```
       [0.0000e+00, 5.0000e+00, 1.0000e+01, ..., 1.4990e+04, 1.4995e+04,
        1.5000e+04]]), 'rho_v': array([[5.76276908e-03, 5.76276908e-03, 5.76276908e-03, .
..,
        8.00394947e-07, 8.00394947e-07, 8.00394947e-07],
       [6.26714779e-03, 6.26714779e-03, 6.26714779e-03, ...,
        7.91836854e-07, 7.91836854e-07, 7.91836854e-07],
       [6.30915332e-03, 6.30915332e-03, 6.30915332e-03, ...,
        7.18454565e-07, 7.18454565e-07, 7.18454565e-07],
       ...,
       [3.02945911e-03, 3.02945911e-03, 3.02945911e-03, ...,
        5.46942448e-07, 5.53067763e-07, 5.58007635e-07],
       [3.33526000e-03, 3.33526000e-03, 3.33526000e-03, ...,
        6.03012291e-07, 6.03012291e-07, 6.03012291e-07],
       [3.41251729e-03, 3.41251729e-03, 3.41251729e-03, ...,
        6.58448795e-07, 6.58448795e-07, 6.58448795e-07]]), 'q': array([[4.50922465e-03, 4
.50922465e-03, 4.50922465e-03, ...,
        4.15917839e-06, 4.16277014e-06, 4.16566464e-06],
       [4.92419675e-03, 4.92419675e-03, 4.92419675e-03, ...,
        4.10497470e-06, 4.10834855e-06, 4.11156265e-06],
       [4.96302030e-03, 4.96302030e-03, 4.96302030e-03, ...,
        3.70434184e-06, 3.70753005e-06, 3.71009933e-06],
       ...,
       [2.28698960e-03, 2.28698960e-03, 2.28698960e-03, ...,
        2.97951339e-06, 3.01655852e-06, 3.04693624e-06],
       [2.52333772e-03, 2.52333772e-03, 2.52333772e-03, ...,
        3.29605353e-06, 3.29875854e-06, 3.30118529e-06],
       [2.58877752e-03, 2.58877752e-03, 2.58877752e-03, ...,
        3.61059570e-06, 3.61290930e-06, 3.61748321e-06]])}
```

Zeit filtern

In [8]:

```
#IASI in Sekunden seit 01.01.2000 > umwandeln in Zeit wie bei NyAlesund

record_start_time = np.zeros(IASI_DS['record_start_time'].shape)
time_diff = (dt.datetime(2000,1,1) - dt.datetime(1970,1,1)).total_seconds()
n_time = IASI_DS['record_start_time'].shape[0]
iasi_record_start_time = IASI_DS.record_start_time.values
n_time_print = int(0.15*n_time)
for i in range(n_time):
    if i%n_time_print == 0: print(i)
    record_start_time[i] = iasi_record_start_time[i] + time_diff


IASI_DS['record_start_time'] = xarray.DataArray(record_start_time, dims=['along_track'])
```

```
0
14857
29714
44571
59428
74285
89142
```

In [12]:

```
n_time_NyAl = len(NyAl_sonde_dict['launch_time'])
time_idx = np.full(n_time_NyAl, np.nan)
for idx in range(n_time_NyAl):
    if np.any(np.abs(record_start_time - NyAl_sonde_dict['launch_time'][idx]) <= time_th
reshold):
        time_idx[idx] = np.argmin(np.abs(record_start_time - NyAl_sonde_dict['launch_tim
e'][idx]))

time_idx = time_idx.astype(np.int32)
```

Koordinaten filtern

In [13]:

```
coords_nyal = (78.924444, 11.928611)
```

In [14]:

```
#Für jeden IASI Pixel (along und across track )
#<-> 2 for loops: berechne Distanz mit geopy Functions
n_along = len(IASI_DS.along_track)
n_across = len(IASI_DS.across_track)
distance_iasi_nyal = np.full((n_along, n_across), np.nan)
IASI_lon = IASI_DS.lon.values
IASI_lat = IASI_DS.lat.values
for j in range(n_across):
    if j%20 == 0: print(j)
    for i in range(n_along):
        if np.abs(IASI_lat[i,j] <= 90):
            distance_iasi_nyal[i,j] = geodesic(coords_nyal, (IASI_lat[i,j], IASI_lon[i,j
])).km
        else:
            distance_iasi_nyal[i,j] = 999999.99

IASI_DS['distance_iasi_nyal'] = xarray.DataArray(distance_iasi_nyal, dims=['along_track',
'across_track'])
```

```
0
20
40
60
80
100
```

In [16]:

```
#NyAlesund from rh to IWP

# constants:
R_d = 287.04   # gas constant of dry air, in J kg^-1 K^-1
R_v = 461.5   # gas constant of water vapour, in J kg^-1 K^-1
M_dv = R_d / R_v # molar mass ratio , in ()
e_0 = 611  # saturation water vapour pressure at freezing point (273.15 K), in Pa
T0 = 273.15  # freezing temperature, in K
g = 9.80665  # gravitation acceleration, in m s^-2

def e_sat(temp):

 """
 Calculates the saturation pressure over water after Goff and Gratch (1946).
 It is the most accurate that you can get for a temperture range from -90°C to +80°C.
 Source: Smithsonian Tables 1984, after Goff and Gratch 1946
 http://cires.colorado.edu/~voemel/vp.html
 http://hurri.kean.edu/~yoh/calculations/satvap/satvap.html

 e_sat_gg_water in Pa.

 Parameters:
 -----------
 temp : array of floats
  Array of temperature (in K).
 """

 e_sat_gg_water = 100 * 1013.246 * 10**(-7.90298*(373.16/temp-1) + 5.02808*np.log10(
            373.16/temp) - 1.3816e-7*(10**(11.344*(1-temp/373.16))-1) + 8.1328e-3 * (10*
*(-3.49149*(373.16/temp-1))-1))

 return e_sat_gg_water

#e_sat(NyAl_temp)

def convert_rh_to_spechum(
 temp,
```

```
 pres,
 relhum):

 """
 Convert array of relative humidity (between 0 and 1) to specific humidity
 in kg kg^-1.

 Saturation water vapour pressure computation is based on: see e_sat(temp).

 Parameters:
 -----------
 temp : array of floats
  Array of temperature (in K).
 pres : array of floats
  Array of pressure (in Pa).
 relhum : array of floats
  Array of relative humidity (between 0 and 1).
 """

 e_sat_water = e_sat(temp)

 e = e_sat_water * relhum
 q = M_dv * e / (e*(M_dv - 1) + pres)

 return q
```

In [17]:

```
n_time_ps = PS_sonde_dict['launch_time'].shape[0]
```

In [18]:

```
#IASI_T zu radiosonden(PS)-pixel-selektieren

n_time_ps = len(PS_sonde_dict['launch_time'])

distance_iasi_ps = np.full((n_along, n_across), 999999.99)
record_start_time = IASI_DS.record_start_time.values
which_PS_sonde = np.full((n_along, n_across), np.nan)
for i in range(n_along):
 if i%1000 == 0: print(i)
 # find Polarstern radiosonde launch that has temporally closest to IASI along track scan
(pixel):
 launch_time_dif = np.abs(PS_sonde_dict['launch_time'] - record_start_time[i])
 idx_ps_time = np.argmin(launch_time_dif)

 if np.abs(PS_sonde_dict['launch_time'][idx_ps_time] - record_start_time[i]) <= 3600:
  # which_PS_sonde[i] = idx_ps_time
  for j in range(n_across):

   if np.abs(IASI_lat[i,j] <= 90):
    disdis = geodesic((PS_sonde_dict['lat'][idx_ps_time], PS_sonde_dict['lat'][idx_ps_ti
me]), (IASI_lat[i,j], IASI_lon[i,j])).km
    if disdis < 50:
     which_PS_sonde[i,j] = idx_ps_time
     distance_iasi_ps[i,j] = disdis

IASI_DS['distance_iasi_ps'] = xarray.DataArray(distance_iasi_ps, dims=['along_track', 'ac
ross_track'])

# remove nans from which_PS_sonde:
which_PS_sonde = which_PS_sonde.astype(np.int32)
which_PS_sonde_nonnan = np.unique(which_PS_sonde[which_PS_sonde >= 0]) # explained below



# what do we have now:
# - distance_iasi_ps (n_along, n_across): distance of IASI to Polarstern for each IASI pi
xel (value is only non-nan if that pixel is temporally within 3600 sec of a
#  Polarstern radiosonde launch AND if it is within 50 km of Polarstern)
# - which_PS_sonde_nonnan (varying dimension): this now indicates which Polarstern sondes
```

```
have IASI pixels that are within 3600 sec of a sonde launch and where IASI
#  pixel is within 50 km of Polarstern

# Later, we want to know if there is a IASI pixel for a given Polarstern launch which is
close enough (i.e. < 50 km) and within
# 3600 sec of that sonde launch. So, we would like to have an array with the shape (n_son
des_ps,2) (or (n_time_ps,2)) that tells
# us the exact along track and across track coordinate of IASI that fulfills these condit
ions.
# So, we can now run through all Polarstern sondes again, and check, if the indicated alo
ng track coordinate has one or more IASI_DS['distance_iasi_ps'] < 50 km
iasi_pixels_for_nya = np.full((n_time_ps,2), 0)

ps_iasi_overlap_pixels = list()
for idx in range(n_time_ps):
    if idx in which_PS_sonde_nonnan: # then it's not a fill value and a IASI pixel with
time offset < 3600 sec exists for the current Polarstern sonde launch:

               # find lines where which_PS_sonde is equal to idx:
        ps_iasi_overlap_pixels.append(np.argwhere(which_PS_sonde == idx))

"""
ps_iasi_overlap_pixels must always be considered together with which_PS_sonde_nonnan:
Example: which_PS_sonde_nonnan[0] is 16: then Polarstern sonde number 16 of your array is
within temporal and spatial range of IASI overpasses
Then, ps_iasi_overlap_pixels[0] tells you which coordinates of IASI_DS overlaps with Pola
rstern: For example,
ps_iasi_overlap_pixels[0] can be

array([[372, 108],
      [372, 109],
      [372, 110],
      [372, 111],
      [373, 108],
      [373, 111]])

--> first column: along_track coordinate that fulfills our requirements ; second column:
across_track coordinate that fulfills requirements
--> 6 pixels of IASI fulfill our requirements in this case. You may now select the Polars
tern temperature profile via:

PS_sonde_dict['temp'][which_PS_sonde_nonnan[0], :];

and the IASI temperature profile(s) (dimensions: along_track, across_track, height):
"""

#n_height_iasi = len(IASI_DS.nlt)
#n_detected_pixels = len(ps_iasi_overlap_pixels[0])
#IASI_T = np.zeros((n_detected_pixels, n_height_iasi))                # here, the T prof
iles for the current Polarstern sonde are saved to
#for idx, ps_ol in enumerate(ps_iasi_overlap_pixels[0]):
#        IASI_T[idx,:] = IASI_DS.atmospheric_temperature[ps_ol[0], ps_ol[1],:]

##IASI_T might include many nans because not all pixels of IASI actually have a temperatu
re profile!
##You might now either average over the number of detected pixels (ignore nans):
#IASI_T_avg = np.nanmean(IASI_T, axis=0)                  # might produce warnings (Runtim
eWarning)
```

```
0
1000
2000
3000
4000
5000
6000
7000
8000
9000
10000
11000
12000
```

13000
14000
15000
16000
17000
18000
19000
20000
21000
22000
23000
24000
25000
26000
27000
28000
29000
30000
31000
32000
33000
34000
35000
36000
37000
38000
39000
40000
41000
42000
43000
44000
45000
46000
47000
48000
49000
50000
51000
52000
53000
54000
55000
56000
57000
58000
59000
60000
61000
62000
63000
64000
65000
66000
67000
68000
69000
70000
71000
72000
73000
74000
75000
76000
77000
78000
79000
80000
81000
82000
83000
84000

```
85000
86000
87000
88000
89000
90000
91000
92000
93000
94000
95000
96000
97000
98000
99000
```

Out[18]:

```
"\nps_iasi_overlap_pixels must always be considered together with which_PS_sonde_nonnan:\
nExample: which_PS_sonde_nonnan[0] is 16: then Polarstern sonde number 16 of your array i
s within temporal and spatial range of IASI overpasses\nThen, ps_iasi_overlap_pixels[0] t
ells you which coordinates of IASI_DS overlaps with Polarstern: For example,\nps_iasi_ove
rlap_pixels[0] can be\n\narray([[372, 108],\n       [372, 109],\n       [372, 110],\n
[372, 111],\n       [373, 108],\n       [373, 111]])\n\n--> first column: along_track coord
inate that fulfills our requirements ; second column: across_track coordinate that fulfil
ls requirements\n--> 6 pixels of IASI fulfill our requirements in this case. You may now
select the Polarstern temperature profile via:\n\nPS_sonde_dict['temp'][which_PS_sonde_no
nnan[0], :];\n\nand the IASI temperature profile(s) (dimensions: along_track, across_trac
k, height):\n"
```

In [19]:

```
#IASI_T zu radiosonden(NyAl)-pixel-selektieren

n_time_NyAl = len(NyAl_sonde_dict['launch_time'])

distance_iasi_NyAl = np.full((n_along, n_across), 999999.99)
record_start_time = IASI_DS.record_start_time.values
which_NyAl_sonde = np.full((n_along, n_across), np.nan)
for i in range(n_along):
 if i%1000 == 0: print(i)
 # find NyAlesund radiosonde launch that has temporally closest to IASI along track scan
(pixel):
 launch_time_dif = np.abs(NyAl_sonde_dict['launch_time'] - record_start_time[i])
 idx_NyAl_time = np.argmin(launch_time_dif)

 if np.abs(NyAl_sonde_dict['launch_time'][idx_NyAl_time] - record_start_time[i]) <= 3600
:
  # which_NyAl_sonde[i] = idx_NyAl_time
  for j in range(n_across):

   if np.abs(IASI_lat[i,j] <= 90):
    disdis = geodesic((NyAl_sonde_dict['lat'][idx_NyAl_time], NyAl_sonde_dict['lat'][idx
_NyAl_time]), (IASI_lat[i,j], IASI_lon[i,j])).km
    if disdis < 50:
     which_NyAl_sonde[i,j] = idx_NyAl_time
     distance_iasi_NyAl[i,j] = disdis

IASI_DS['distance_iasi_NyAl'] = xarray.DataArray(distance_iasi_NyAl, dims=['along_track',
'across_track'])

# remove nans from which_NyAl_sonde:
which_NyAl_sonde = which_NyAl_sonde.astype(np.int32)
which_NyAl_sonde_nonnan = np.unique(which_NyAl_sonde[which_NyAl_sonde >= 0]) # explained
below



# what do we have now:
# - distance_iasi_NyAl (n_along, n_across): distance of IASI to NyAl for each IASI pixel
(value is only non-nan if that pixel is temporally within 3600 sec of a
#  NyAl radiosonde launch AND if it is within 50 km of NyAl)
```

```
# - which_NyAl_sonde_nonnan (varying dimension): this now indicates which NyAl sondes hav
e IASI pixels that are within 3600 sec of a sonde launch and where IASI
#  pixel is within 50 km of NyAl

# Later, we want to know if there is a IASI pixel for a given NyAl launch which is close
enough (i.e. < 50 km) and within
# 3600 sec of that sonde launch. So, we would like to have an array with the shape (n_son
des_NyAl,2) (or (n_time_ps,2)) that tells
# us the exact along track and across track coordinate of IASI that fulfills these condit
ions.
# So, we can now run through all NyAl sondes again, and check, if the indicated along tra
ck coordinate has one or more IASI_DS['distance_iasi_NyAl'] < 50 km
iasi_pixels_for_NyAl = np.full((n_time_ps,2), 0)

NyAl_iasi_overlap_pixels = list()
for idx in range(n_time_NyAl):
    if idx in which_NyAl_sonde_nonnan: # then it's not a fill value and a IASI pixel wit
h time offset < 3600 sec exists for the current NyAl sonde launch:

               # find lines where which_NyAl_sonde is equal to idx:
        NyAl_iasi_overlap_pixels.append(np.argwhere(which_NyAl_sonde == idx))

"""
NyAl_iasi_overlap_pixels must always be considered together with which_PS_sonde_nonnan:
Example: which_PS_sonde_nonnan[0] is 16: then NyAl sonde number 16 of your array is withi
n temporal and spatial range of IASI overpasses
Then, NyAl_iasi_overlap_pixels[0] tells you which coordinates of IASI_DS overlaps with Ny
Al: For example,
NyAl_iasi_overlap_pixels[0] can be

array([[372, 108],
      [372, 109],
      [372, 110],
      [372, 111],
      [373, 108],
      [373, 111]])

--> first column: along_track coordinate that fulfills our requirements ; second column:
across_track coordinate that fulfills requirements
--> 6 pixels of IASI fulfill our requirements in this case. You may now select the NyAl t
emperature profile via:

NyAl_sonde_dict['temp'][which_NyAl_sonde_nonnan[0], :];

and the IASI temperature profile(s) (dimensions: along_track, across_track, height):
"""

#n_height_iasi = len(IASI_DS.nlt)
#n_detected_pixels = len(NyAl_iasi_overlap_pixels[0])
#IASI_T = np.zeros((n_detected_pixels, n_height_iasi))                # here, the T prof
iles for the current NyAl sonde are saved to
#for idx, NyAl_ol in enumerate(NyAl_iasi_overlap_pixels[0]):
#       IASI_T[idx,:] = IASI_DS.atmospheric_temperature[NyAl_ol[0], NyAl_ol[1],:]

##IASI_T might include many nans because not all pixels of IASI actually have a temperatu
re profile!
##You might now either average over the number of detected pixels (ignore nans):
#IASI_T_avg = np.nanmean(IASI_T, axis=0)                # might produce warnings (Runtim
eWarning)
```

```
0
1000
2000
3000
4000
5000
6000
7000
8000
9000
10000
11000
```

```
12000
13000
14000
15000
16000
17000
18000
19000
20000
21000
22000
23000
24000
25000
26000
27000
28000
29000
30000
31000
32000
33000
34000
35000
36000
37000
38000
39000
40000
41000
42000
43000
44000
45000
46000
47000
48000
49000
50000
51000
52000
53000
54000
55000
56000
57000
58000
59000
60000
61000
62000
63000
64000
65000
66000
67000
68000
69000
70000
71000
72000
73000
74000
75000
76000
77000
78000
79000
80000
81000
82000
83000
```

```
84000
85000
86000
87000
88000
89000
90000
91000
92000
93000
94000
95000
96000
97000
98000
99000
```

Out[19]:

```
"\nNyAl_iasi_overlap_pixels must always be considered together with which_PS_sonde_nonnan
:\nExample: which_PS_sonde_nonnan[0] is 16: then NyAl sonde number 16 of your array is wi
thin temporal and spatial range of IASI overpasses\nThen, NyAl_iasi_overlap_pixels[0] tel
ls you which coordinates of IASI_DS overlaps with NyAl: For example,\nNyAl_iasi_overlap_p
ixels[0] can be\n\narray([[372, 108],\n       [372, 109],\n       [372, 110],\n       [372,
111],\n       [373, 108],\n       [373, 111]])\n\n--> first column: along_track coordinate
that fulfills our requirements ; second column: across_track coordinate that fulfills req
uirements\n--> 6 pixels of IASI fulfill our requirements in this case. You may now select
the NyAl temperature profile via:\n\nNyAl_sonde_dict['temp'][which_NyAl_sonde_nonnan[0],
:];\n\nand the IASI temperature profile(s) (dimensions: along_track, across_track, height
):\n"
```

In [20]:

```
IASI_ST = IASI_DS.surface_temperature.values
IASI_SP= IASI_DS.surface_pressure.values
```

In [22]:

```
# height grid auf eine Größe gebracht werden / interpolieren darauf

NyAl_T = NyAl_sonde_dict['temp']
NyAl_alt = NyAl_sonde_dict['height']
NyAl_q = convert_rh_to_spechum(NyAl_sonde_dict['temp'], NyAl_sonde_dict['pres'], NyAl_so
nde_dict['rh'])

IASI_T = IASI_DS.atmospheric_temperature.values
IASI_q = IASI_DS.atmospheric_water_vapor.values
```

In [24]:

```
# for schleife für unterschiedliche shapes IASI_T.shape=(2980, 120, 101) along, across, v
ertical    IASI_DS.pressure_levels_temp.shape = (101,)

IASI_alt = np.zeros(IASI_T.shape)
IASI_DS_pressure_levels_temp = IASI_DS.pressure_levels_temp.values
for i in range(n_along):
    for j in range(n_across):
        IASI_alt[i,j,:] = -(R_d / g) * (IASI_T[i,j,:]*np.log(IASI_DS_pressure_levels_tem
p) - IASI_ST[i,j]*np.log(IASI_SP[i,j]))
```

In [25]:

```
#0 0
#1 1
#2 4
#3 5
#Die 0,1,4,5 hatten die Überschneidenenden along und across wie IASI
```

In [26]:

```
n_sondes_PS = len(which_PS_sonde_nonnan)
```

```
n_height_PS = PS_sonde_dict['temp'].shape[1]

PS_T = np.zeros((n_sondes_PS,n_height_PS))
PS_P = np.zeros((n_sondes_PS,n_height_PS))
PS_q = np.zeros((n_sondes_PS,n_height_PS))

for i, which_PS in enumerate(which_PS_sonde_nonnan):
  #  print(i,which_PS)
    PS_T[i,:] = PS_sonde_dict['temp'][which_PS,:]
    PS_P[i,:] = PS_sonde_dict['pres'][which_PS,:]
    PS_q[i,:] = PS_sonde_dict['q'][which_PS,:]

n_sondes_NyAl = len(which_NyAl_sonde_nonnan)
n_height_NyAl = NyAl_sonde_dict['temp'].shape[1]

NyAl_T = np.zeros((n_sondes_NyAl,n_height_NyAl))
NyAl_P = np.zeros((n_sondes_NyAl,n_height_NyAl))
NyAl_q = np.zeros((n_sondes_NyAl,n_height_NyAl))

for i, which_NyAl in enumerate(which_NyAl_sonde_nonnan):
  #  print(i,which_NyAl)
    NyAl_T[i,:] = NyAl_sonde_dict['temp'][which_NyAl,:]
    NyAl_P[i,:] = NyAl_sonde_dict['pres'][which_NyAl,:]
    NyAl_q[i,:] = NyAl_sonde_dict['rh'][which_NyAl,:]
```

In [27]:

```
len(ps_iasi_overlap_pixels)#IASI überlappt mit PS

n_height_IASI = len(IASI_DS_pressure_levels_temp)
IASI_T_PS = np.zeros((n_sondes_PS,n_height_IASI))
IASI_q_PS = np.zeros((n_sondes_PS,n_height_IASI))
for idx, ps_iasi_overlap in enumerate(ps_iasi_overlap_pixels):
        along_chosen = ps_iasi_overlap[:,0]
        across_chosen = ps_iasi_overlap[:,1]
        IASI_T_PS[idx,:] = np.nanmean(IASI_T[along_chosen, across_chosen,:], axis=0)
        IASI_q_PS[idx,:] = np.nanmean(IASI_q[along_chosen, across_chosen,:], axis=0)
```

```
<ipython-input-27-636a800aa1f5>:9: RuntimeWarning: Mean of empty slice
  IASI_T_PS[idx,:] = np.nanmean(IASI_T[along_chosen, across_chosen,:], axis=0)
<ipython-input-27-636a800aa1f5>:10: RuntimeWarning: Mean of empty slice
  IASI_q_PS[idx,:] = np.nanmean(IASI_q[along_chosen, across_chosen,:], axis=0)
```

In [28]:

```
len(NyAl_iasi_overlap_pixels)#IASI überlappt mit NyAl

n_height_IASI = len(IASI_DS_pressure_levels_temp)
IASI_T_NyAl = np.zeros((n_sondes_NyAl,n_height_IASI))
IASI_q_NyAl = np.zeros((n_sondes_NyAl,n_height_IASI))
for idx, NyAl_iasi_overlap in enumerate(NyAl_iasi_overlap_pixels):
        along_chosen = NyAl_iasi_overlap[:,0]
        across_chosen = NyAl_iasi_overlap[:,1]
        IASI_T_NyAl[idx,:] = np.nanmean(IASI_T[along_chosen, across_chosen,:], axis=0)
        IASI_q_NyAl[idx,:] = np.nanmean(IASI_q[along_chosen, across_chosen,:], axis=0)
```

```
<ipython-input-28-55e26a30ab36>:9: RuntimeWarning: Mean of empty slice
  IASI_T_NyAl[idx,:] = np.nanmean(IASI_T[along_chosen, across_chosen,:], axis=0)
<ipython-input-28-55e26a30ab36>:10: RuntimeWarning: Mean of empty slice
  IASI_q_NyAl[idx,:] = np.nanmean(IASI_q[along_chosen, across_chosen,:], axis=0)
```

In [29]:

```
#interpolation über druck koordinaten
IASI_DS_pressure_levels_humidity = IASI_DS.pressure_levels_humidity.values
IASI_T_PS_grid = np.zeros((n_sondes_PS,n_height_PS))
IASI_q_PS_grid = np.zeros((n_sondes_PS,n_height_PS))

for idx in range(n_sondes_PS):
    IASI_T_PS_grid[idx,:] = np.interp(PS_P[idx,:],IASI_DS_pressure_levels_temp,IASI_T_PS[
idx,:])
```

```
    IASI_q_PS_grid[idx,:] = np.interp(PS_q[idx,:],IASI_DS_pressure_levels_humidity,IASI_q
_PS[idx,:])
```

In [30]:

```
#interpolation über druck koordinaten
IASI_DS_pressure_levels_humidity = IASI_DS.pressure_levels_humidity.values
IASI_T_NyAl_grid = np.zeros((n_sondes_NyAl,n_height_NyAl))
IASI_q_NyAl_grid = np.zeros((n_sondes_NyAl,n_height_NyAl))

for idx in range(n_sondes_NyAl):
    IASI_T_NyAl_grid[idx,:] = np.interp(NyAl_P[idx,:],IASI_DS_pressure_levels_temp,IASI_T
_NyAl[idx,:])
    IASI_q_NyAl_grid[idx,:] = np.interp(NyAl_q[idx,:],IASI_DS_pressure_levels_humidity,I
ASI_q_NyAl[idx,:])
```

RMSE

$$\bar{s}^2 = \frac{1}{n}$$
$$* \left[(T_{Sonde} - \bar{T_{IASI}})^2 + (x_{Sonde} - \bar{T_{IASI}})^2 + ..\right.$$
$$\left. .(x_{Sonde_n} - \bar{T_{IASI}})^2 \right]$$

In [31]:

```
# Polarstern und IASI

def RMSE(X_sonde,X_IASI):

    RMSE_X = np.sqrt( np.nanmean((X_sonde - X_IASI)**2, axis=0) )  # for RMSE PROFILE, w
e average over time (which is axis 0)

    return RMSE_X

RMSE_T_IASI_PS = RMSE(IASI_T_PS_grid, PS_T)
RMSE_pres = np.nanmean(PS_P, axis=0) # zugehöriger Druck
RMSE_q_IASI_PS = RMSE(IASI_q_PS_grid,PS_q)

RMSE_T_IASI_NyAl = RMSE(IASI_T_NyAl_grid, NyAl_T)
RMSE_pres = np.nanmean(NyAl_P, axis=0) # zugehöriger Druck
RMSE_q_IASI_NyAl = RMSE(IASI_q_NyAl_grid,NyAl_q)
```

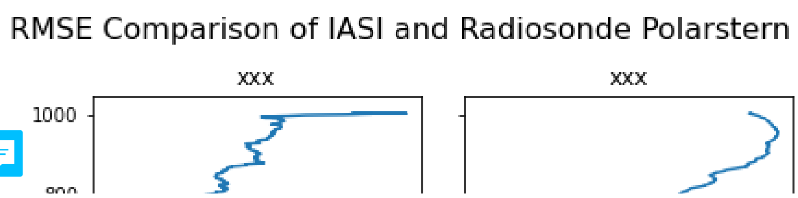
Plot

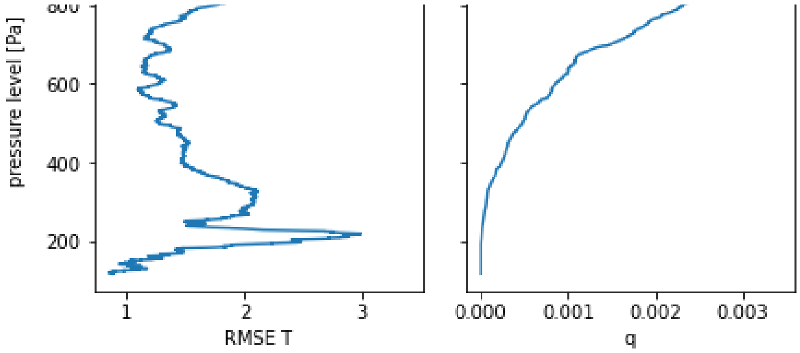
In [32]:

```
fig, (ax1, ax2) = plt.subplots(1, 2, constrained_layout=True, sharey=True)
ax1.plot(RMSE_T_IASI_PS, RMSE_pres)
ax1.set_title('xxx')
ax1.set_xlabel('RMSE T')
ax1.set_ylabel('pressure level [Pa]')

ax2.plot(RMSE_q_IASI_PS, RMSE_pres)
ax2.set_xlabel('q')
ax2.set_title('xxx')

fig.suptitle('RMSE Comparison of IASI and Radiosonde Polarstern', fontsize=16)
```

Out[32]:

```
Text(0.5, 0.98, 'RMSE Comparison of IASI and Radiosonde Polarstern')
```



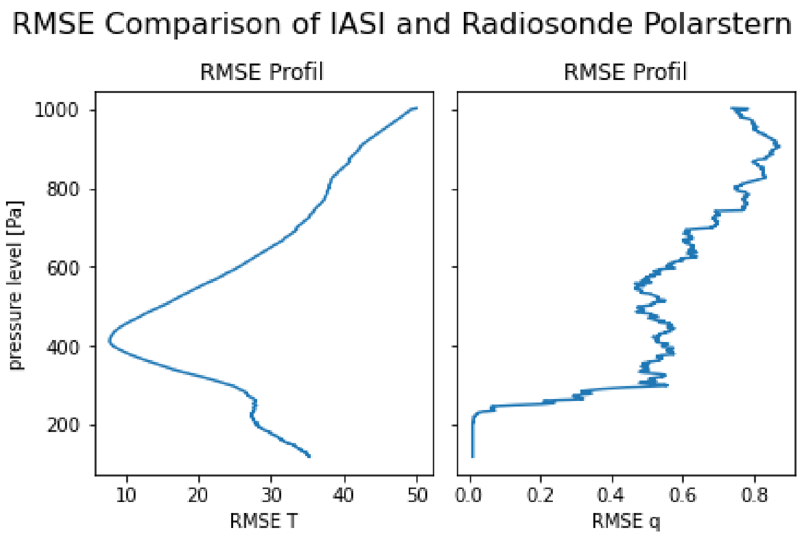


In [33]:

```
fig, (ax1, ax2) = plt.subplots(1, 2, constrained_layout=True, sharey=True)
ax1.plot(RMSE_T_IASI_NyAl, RMSE_pres)
ax1.set_title('RMSE Profil')
ax1.set_xlabel('RMSE T')
ax1.set_ylabel('pressure level [Pa]')

ax2.plot(RMSE_q_IASI_NyAl, RMSE_pres)
ax2.set_xlabel('RMSE q')
ax2.set_title('RMSE Profil')

fig.suptitle('RMSE Comparison of IASI and Radiosonde Polarstern', fontsize=16)
```

Out[33]:

```
Text(0.5, 0.98, 'RMSE Comparison of IASI and Radiosonde Polarstern')
```



In [ ]: