Contents

1 Chosen Project

I have chosen for project proposal 1.

The goal of this project is to create a pipeline that takes in data from the knmi.nl pupsub api and combines it into a simple-to-use dataset. This dataset should then be easy to used to train machine learning models. Especially transformer models. I in this report I explore the data and files that this api provides and I propose to improve the quality by creating a pytorch data loader class instance for this dataset.

Fighting with h5 files seems to me like a fun challenge. It is definitely not the easiest file format to work with. Luckily the great python ecosystem contains a library for these files, but they are inherently complex. The live updates coming in from the government give a good reason to explore DAGs which interests me. During the project the api got updated and now only returning .nc files.

2 Ease of access

The files in the dataset are not easy to access. You first need to make an account on the knmi.nl website. After this you need to apply to get two different access tokens. If you get accepted, you can use token 1 to subscribe to a stream where kmni publishes new data. Items in this stream are json documents that contain a link. You can make a specific request to this link with token 2. In the response you get a temporary download link from which you can actually download the data. You should not use an authentication header on this final link because then it does not actually download.

3 H5 file

To become familiar with H5 I read these resources:

- https://docs.fileformat.com/misc/h5/
- https://github.com/christianversloot/machine-learning-articles/blob/main/how-to-use-h5py-and-keras-to-train-with-data-from-hdf5-files.md
- https://docs.hdfgroup.org/hdf5/develop/_f_m_t3.html#AttributeMessage
- https://docs.hdfgroup.org/hdf5/develop/_f_m_t3.html

I also tried out the online h5 file viewer at https://myhdf5.hdfgroup.org/ which was very helpful for my understanding.

After reading these resources I made the following description of H5 in my own words:

3.1 What is H5?

The H5 binary format is like a zip file. A file which contains other files. Files inside an H5 file have two kinds. These two kinds are groups and datasets. Each group and dataset has a name and is located at a certain path in the H5 file.

A group or dataset always contain a bit of json that describes the dataset or group. A dataset is a multidimensional array of a homogeneous type. So for instance a matrix of only integers. These datasets can be loaded as a numpy array.

A group can contain other groups and datasets. Datasets can not contain groups.

Groups and datasets can also contain attributes. Attributes can be used for the metadata of a file, but also as small datasets that are attached to a file. Attributes can contain many different types of data like stings, various types of numbers and more. According to the HDF5 spec the data in a single attribute should not be larger than 64kb.

Figure 1: H5 file system

3.2 H5 file format quality assessment

The fact that the data is in H5 format is good and bad for the quality of the data. An H5 file should be good for quality because the format can store a massive amount of vectors in such a way that it is efficient to load.

However, to use h5 files you need a pretty good understanding of them. This is not good for quality because a data scientist has to actually learn about HDF5 files before they can use them. The average data scientist did not learn about H5 files in their education. Thus, the average data scientist will find it difficult and annoying to load this type of file and load the data into a model. There is a trade-off here between speed of loading and convenience.

Something that also does not help with the quality of h5 files is that there is no easy way to join them together like with a sql database or even a list record in json.

4 The switch to NC files

During the course of this project it seems that the knmi has switched their file format from .h5 to .nc files. At first, I thought that the api could just also return h5 files but after running a script that listens to the data from knmi for two days I observed no further .h5 files any more. So what is an nc file?

4.1 Nc file format

To understand the NC file format I consulted the following resources:

- https://unidata.github.io/netcdf4-python/
- https://stackoverflow.com/questions/1075074/opinions-on-netcdf-vs-hdf5-for-storing-scientific-data
- https://nctoolkit.readthedocs.io/en/v0.1.2/

The NC (NetCDF) a binary file format that is implemented on top of HD5. As such NC is also a binary data format used to store multidimensional arrays and scaler values. The big difference between NetCDF and HD5 is NetCDF is not hierarchical. This means that there are no groups in groups with HD5 but only datasets. `.nc` is like a zip file without any directories. A tree with only leafs. All the dataset are at the root of the tree. This means that any library that can load HD5 should also be able to load NetCDF files. This was indeed the case for the web visualizer for h5 I used earlier.

To illustrate, an h5 file would give you something like this:

```
/
  /weather
    /rain
        - vector
        - image
    /temperature
        - vector
        - image
```

A nc file instead gives you something like this:

```
/
- Rain vector
- Rain image
- Temperature vector
- Temperature image
```

5 Contents of NC files

Even though every h5 loader should be able to load nc files, I looked on the internet for (python) tools specifically made for nc files. At first, I found something called `nctoolkit` (linked above). This had an amazing documentation, but it was sadly unable to load any files for me.

Secondly from this stack overflow post I found netCDF4 which also has great documentation

Using this library I could load an .nc file like this:

```
import netCDF4

db = netCDF4.Dataset("KMDS__OPER_P___10M_OBS_L2_202405161720.nc")
print(db)
print(db.variables)
```

5.1 Exploring the NC files

At this point using the command `ls *.nc -l | wc -l` I found out that I had downloaded 83 files.

Next I wrote a program check the dataset names within an .nc file. You can find the `iter_nc_files` iterator in the appendix.

```
from collections import defaultdict
import glob, netCDF4

counts = defaultdict(int)
file_count = 0
for db in iter_nc_files():
    for key in db.variables:
        counts[key] += 1
    file_count += 1
print(f"Iterated over {file_count} files. ")
print(f"There are {len(counts)} different keys with all the files")
print(f"Each key appearance counts:\n", counts)
```

Find the full printed result in the appendix. The output showed me a lot of information about these .nc files.

Within the 85 files I collected so far there were 103 different databases names found. Every database name appeared 85 times in the files. At least every .nc file has the same database names. This was nice to see because with the h5 files I had noticed differences in the file system layout between different files.

5.1.1 Making a Table of contents

I proceeded to investigate each database further. I found out that each database actually has some very useful attributes. Namely `comment`, `long name`, `unit`, `size`, `shape` and `datatype`.

The comments are either empty or very detailed. An example of a comment for the `zm` dataset:

```
zm Note: The sensor is not installed at equal heights at all types of
measurement sites: At 'AWS' sites the device is installed at 1.80m.
At 'AWS/Aerodrome' and 'Mistpost' (note that this includes site
Voorschoten (06215) which is 'AWS/Mistpost') the device is installed
 at 2.50m elevation. Exceptions are Berkhout AWS (06249), De Bilt AWS
(06260) and Twenthe AWS (06290) where the sensor is installed at 2.50m.
```

Using the toMark package I generated a table with all the database names within each nc file. You can find this table in the appendix. From this table I learned that each database was either a primitive type or an array of 69 values.

5.2 Units

After making that table turned the units column into a set which left me with these units:

- %', 1, J cm-2, W m-2
- cd m-2, code, degree
- degrees Celsius, degrees_east, degrees_north
- ft, hPa, m
- m s-1, min, mm
- mm h-1, octa
- sec, seconds since 1950-01-01 00:00:00

5.3 Station names

Each nc file has a `stations` database. This database always holds 69 strings. From these station databases I finally identified what these nc files actually contain. These 69 values come from 69 weather stations in the Netherlands connected to knmi. So that means that each nc file holds 69 measurements from different weather stations in the Netherlands at a single point in time.

I investigated the values of these station fields. They are string names like this:

```
['06203', '06204', '06205', ..., '06207']
```

Most of the time the station names are the same ones but not always. So far 7% (8/112) of station names list are different. A noteworthy observation is that the station names are always sorted from low to high.

I created a set of the sizes of station databases. The result was `{68,69}`. This shows that sometimes data from less station names are included in a nc file.

There is also a `stationname` database in each nc file. This seems to contain longer station names for humans. I created a set of all the station names across all the nc files. This set can be found in the appendix. The set had exactly 69 items. This means that when there are 69 stations in an nc file, they are at least the same stations.

Finally, I also observed the `FLAMINGO AIRPORT BONAIRE` in this set of stations. This station is located on the island of Bonaire instead of the mainland of the Netherlands. Because of this it might be less useful for the data scientist when predicting the weather in the Netherlands.

5.4 Iso database

I also looked into the `iso_dataset` dataset which is present in all the nc files. This contained a lot of interesting data. The content of this dataset across nc files seems to follow the same schema. Here is an example:

```
|S1 iso_dataset()
    title: KMDS__OPER_P___10M_OBS_L2
    abstract: Most recent 10 minute in situ observations of the Dutch
 ↪  meteorological observation network
    status: ongoing
    type: dataset
    uid: c3a312e2-2d8f-440b-ae7d-3406c9fe2f77
    topic: climatology, meteorology, atmosphere
    keyword: temperature, pressure, relative humidity, visibility, wind
 ↪  speed, wind direction, wind gust
    max-x: 7.4
    min-x: -68.5
    max-y: 55.7
    min-y: 12.0
    temporal_extent: 1950-01-01 and ongoing
    date: 2023-07-01
    dateType: publication date
    statement: KNMI collects observations from automatic weather stations
 ↪  situated in the
                Netherlands and BES islands on locations such as aerodromes
 ↪  and North Sea platforms.
```

```
                In addition, wind data from 5 KNMI wind poles are included.
                The weather stations report every 10 minutes meteorological
↪  parameters such as temperature,
                relative humidity, wind, air pressure, visibility,
↪  precipitation and cloud cover.
                The number of parameters differs per station.
    code: 4326
    codeSpace: EPSG
    accessConstraints: No limitations
    useLimitation: No use limitations
    organisationName_dataset: Royal Netherlands Meteorological Institute
↪   (KNMI)
    email_dataset: opendata@knmi.nl
    role_dataset: pointOfContact
    metadata_id: 0ebaef73-6bff-4a25-a86b-6bf53bc4c505
    organisationName_metadata: Royal Netherlands Meteorological Institute
↪   (KNMI)
    role_metadata: pointOfContact
    email_metadata: opendata@knmi.nl
    url_metadata: https://dataplatform.knmi.nl
    datestamp: 2014-07-23
    language: eng
    metadataStandardName: ISO 19115
    metadataStandardNameVersion: Nederlandse metadatastandaard op ISO 19115
↪   voor geografie 1.2
unlimited dimensions:
current shape = ()
filling on, default _FillValue of   used
```

Most importantly the `metadataStandardName` field.

This seems to indicate that there is some iso standard that this nc file is following.

I found a full version of the standard here But this seems to describe a newer version then the standard (2.1.0) than the one in use by the nc files I get which is 1.2. Apparently the Netherlands are using an old version of this standard.

5.5 Missing values

After loading some nc files using the netCF4 library I found many values of value `--`. I have never seen that before, so I inspected the type of the arrays and I got `numpy.ma.core.MaskedArray` for almost all of them. This is a type of numpy array that supports missing values. Instead of having some sentinel value as your missing values, with masked arrays you include a mask to mark the missing

values. A mask is just a boolean array. If a boolean is true at index i then index i is masked in the original array. Numpy makes sure that a masked value is not included in computations. Here is an example:

```
import numpy as np
import numpy.ma as ma
x = np.array([1, 4, 7])
x.mean(x) # 4.0
mx = ma.masked_array(x, mask=[False, False, True])
mx.mean(ma) # 2.5
```

Torch also has experimental support for masked arrays. See https://pytorch.org/docs/stable/masked.html.

Some arrays apparently never have missing values as these are just normal numpy arrays. An example of such a database is `stations`.

6 Data quality assessment

The quality of the data in terms of accuracy seems to be high as it comes from scientific measurement tools.

The dataset contains good information about the units of data. For instance pressure units or temperature in c. However, the only time unit being `seconds since 1950-01-01 00:00:00` is a bit weird. They could have gone with iso 8601.

There are many missing values. But the fact that we load then as a MaskedArray does not make this a large issue. This is the best way of doing missing values I have ever seen and I will use this myself moving forward when possible.

All vector data seems to be quite usable and in theory very suitable for machine learning models. Ofcourse, some data fields might provide more useful than other data to craft the model but this judgement is up to the data scientist.

One thing to mention is that the shape of the vectors is (69,1). That is a list of 69 lists of lists with 1 value. There seems no reason why it is like that instead of just 69 values.

A large issue with the data is the differing station names. Most of the time the station names same, but sometimes they are not. This means that if you would train on the data naively you would get a timeseries where some of the values of a specific moment are actually from different station in the Netherlands.

Accessing the data is hard as I have hopefully sufficiently described. You first have to connect to the api which is quite some steps. Then you have to extract the data from these file formats that I had

never seen before which is also not trivial. Loading data from nc and h5 formats into something like torch is tricky. Furthermore, this data is only really useful when it is presented as a time series. But, as each file only has the measurements of one moment in time from 69 different stations, the files need to be combined somehow to make it interesting.

7 Improving the Quality

7.1 Improving data quality

All sensor reading data is already a float type which means they can be loaded as numbers. This way we can avoid errors like deciding how to interpret “13.70” and “13,70”.

The data uses seconds after an epoch for dates instead of the true dataformat which is the iso 8601 one. It is a weird epoch, 1950. They choose this because that was when they started measuring. However, I want to create a time sequence dataset. For that reason I think that date strings will not help the data scientist as it is much more about the sequence then the actual date. As such I will leave the seconds after 1950 as the time.

I do not think I have to address the missing values as long as I can convert a masked numpy array into a masked tensor properly.

I will also reshape the data of the data vectors from (69,1) to just 69. That just a list of 69 values instead of a list of 69 lists with 1 value.

7.2 Improving ease of use

I will mainly improve the quality by improving how easy the data will be to load. To do this I want to create a torch dataloader. In the constructor of this dataloader the data scientist should be able to say which keys they want, from which stations and from which date range (from, until). The dataloader will then go download and cache the datasets from that date range and the keys you downloaded. With these three parameters you can exactly control which data you want to download. After the download it is up to the data scientist what they want to do next.

As a nice feature I shall have a default value for each parameter.

- The default value for the stations will be the stations sequence that appears most of the time.
- The default value for the keys will be all the keys.
- The default value of the date range will be the whole of last week.

8 Appendix

8.1 `iter_nc_files` iterator

This is a python iterator that iterates over all .nc files and yields opened versions of them.

```
import glob, netCDF4


def iter_nc_files():
    for file in glob.iglob("*.nc"):
        db = netCDF4.Dataset(file)
        try:
            yield db
        finally:
            db.close()
```

8.2 Result of exploration program 1.

Result of exploration program 1.

```
Iterated over 85 files.
There are 103 different keys with all the files
Each key appearance counts:
defaultdict(<class 'int'>, {'station': 85, 'time': 85,
'wsi': 85, 'stationname': 85, 'lat': 85, 'lon': 85,
'height': 85, 'D1H': 85, 'dd': 85, 'dn': 85, 'dr': 85,
'dsd': 85, 'dx': 85, 'ff': 85, 'ffs': 85, 'fsd': 85,
'fx': 85, 'fxs': 85, 'gff': 85, 'gffs': 85, 'h': 85,
'h1': 85, 'h2': 85, 'h3': 85, 'hc': 85, 'hc1': 85, 'hc2': 85,
'hc3': 85, 'n': 85, 'n1': 85, 'n2': 85, 'n3': 85, 'nc': 85,
'nc1': 85, 'nc2': 85, 'nc3': 85, 'p0': 85, 'pp': 85, 'pg': 85,
'pr': 85, 'ps': 85, 'pwc': 85, 'Q1H': 85, 'Q24H': 85, 'qg': 85, 'qgn': 85,
'qgx': 85, 'qnh': 85, 'R12H': 85, 'R1H': 85, 'R24H': 85, 'R6H': 85,
'rg': 85, 'rh': 85, 'rh10': 85, 'Sav1H': 85, 'Sax1H': 85, 'Sax3H': 85,
'Sax6H': 85, 'sq': 85, 'ss': 85, 'Sx1H': 85, 'Sx3H': 85, 'Sx6H': 85,
't10': 85, 'ta': 85, 'tb': 85, 'tb1': 85, 'Tb1n6': 85, 'Tb1x6': 85,
'tb2': 85, 'Tb2n6': 85, 'Tb2x6': 85, 'tb3': 85, 'tb4': 85, 'tb5': 85,
'td': 85, 'td10': 85, 'tg': 85, 'tgn': 85, 'Tgn12': 85, 'Tgn14': 85,
'Tgn6': 85, 'tn': 85, 'Tn12': 85, 'Tn14': 85, 'Tn6': 85, 'tsd': 85,
'tx': 85, 'Tx12': 85, 'Tx24': 85, 'Tx6': 85, 'vv': 85, 'W10': 85,
'W10-10': 85, 'ww': 85, 'ww-10': 85, 'zm': 85, 'iso_dataset': 85,
'product': 85, 'projection': 85, 'nhc': 85, 'za': 85})
```

8.3 Table of databases in each nc file and their names, sizes, shapes, types and units

name	Long name	size	shape	units	type
station	Station id	69	(69,)	None	string
time	time of measurement	1	(1,)	seconds since 1950-01-01 00:00:00	float64
wsi	Station wsi	69	(69,)	None	string
stationname	Station name	69	(69,)	None	string
lat	station latitude	69	(69,)	degrees_north	float64
lon	station longitude	69	(69,)	degrees_east	float64
height	Station height	69	(69,)	m	float64
D1H	Rainfall Duration in last Hour	69	(69, 1)	min	float64
dd	Wind Direction 10 Min Average with MD	69	(69, 1)	degree	float64
dn	Wind Direction Sensor 10 Min Minimum with MD	69	(69, 1)	degree	float64
dr	Precipitation Duration (Rain Gauge) 10 Min Sum	69	(69, 1)	sec	float64
dsd	Wind Direction 10 Min Std Dev with MD	69	(69, 1)	degree	float64
dx	Wind Direction Sensor 10 Min Maximum with MD	69	(69, 1)	degree	float64
ff	Wind Speed at 10m 10 Min Average with MD	69	(69, 1)	m s-1	float64
ffs	Wind Speed Sensor 10 Min Average with MD	69	(69, 1)	m s-1	float64
fsd	Wind Speed 10 Min Std Dev with MD	69	(69, 1)	m s-1	float64
fx	Wind Gust at 10m Maximum last 10 Min Interval	69	(69, 1)	m s-1	float64
fxs	Wind Gust Sensor Maximum last 10 Min Interval	69	(69, 1)	m s-1	float64

name	Long name	size	shape	units	type
gff	Wind Gust at 10m 10 Min Maximum with MD	69	(69, 1)	m s-1	float64
gffs	Wind Gust Sensor 10 Min Maximum with MD	69	(69, 1)	m s-1	float64
h	Cloud Base	69	(69, 1)	ft	float64
h1	Cloud Base First Layer	69	(69, 1)	ft	float64
h2	Cloud Base Second Layer	69	(69, 1)	ft	float64
h3	Cloud Base Third Layer	69	(69, 1)	ft	float64
hc	Cloud Base Ceilometer Algorithm	69	(69, 1)	ft	float64
hc1	Cloud Base Ceilometer First Layer	69	(69, 1)	ft	float64
hc2	Cloud Base Ceilometer Second Layer	69	(69, 1)	ft	float64
hc3	Cloud Base Ceilometer Third Layer	69	(69, 1)	ft	float64
n	Total cloud cover	69	(69, 1)	octa	float64
n1	Cloud Amount First Layer	69	(69, 1)	octa	float64
n2	Cloud Amount Second Layer	69	(69, 1)	octa	float64
n3	Cloud Amount Third Layer	69	(69, 1)	octa	float64
nc	Total Cloud Cover Ceilometer	69	(69, 1)	octa	float64
nc1	Cloud Amount Ceilometer First Layer	69	(69, 1)	octa	float64
nc2	Cloud Amount Ceilometer Second Layer	69	(69, 1)	octa	float64
nc3	Cloud Amount Ceilometer Third Layer	69	(69, 1)	octa	float64
p0	Air Pressure at Station Level 1 Min Average	69	(69, 1)	hPa	float64
pp	Air Pressure at Sea Level 1 Min Average	69	(69, 1)	hPa	float64

name	Long name	size	shape	units	type
pg	Precipitation Intensity (PWS) 10 Min Average	69	(69, 1)	mm h-1	float64
pr	Precipitation Duration (PWS) 10 Min Sum	69	(69, 1)	sec	float64
ps	Air Pressure at Sensor Level 10 Min Average	69	(69, 1)	hPa	float64
pwc	Corrected Precipitation Type 10 Min Maximum	69	(69, 1)	code	float64
Q1H	Global Radiation 1 Hour Sum	69	(69, 1)	J cm-2	float64
Q24H	Global Radiation 24 Hour Sum	69	(69, 1)	J cm-2	float64
qg	Global Solar Radiation 10 Min Average	69	(69, 1)	W m-2	float64
qgn	Global Solar Radiation 10 Min Minimum	69	(69, 1)	W m-2	float64
qgx	Global Solar Radiation 10 Min Maximum	69	(69, 1)	W m-2	float64
qnh	QNH 1 Min Average	69	(69, 1)	hPa	float64
R12H	Rainfall in last 12 Hours	69	(69, 1)	mm	float64
R1H	Rainfall in last Hour	69	(69, 1)	mm	float64
R24H	Rainfall in last 24 Hours	69	(69, 1)	mm	float64
R6H	Rainfall in last 6 Hours	69	(69, 1)	mm	float64
rg	Precipitation Intensity (Rain Gauge) 10 Min Average	69	(69, 1)	mm h-1	float64
rh	Relative Humidity 1.5m 1 Min Average	69	(69, 1)	%	float64
rh10	Relative Humidity 10 Min Average	69	(69, 1)	%	float64
Sav1H	Wind Speed Average last 1 Hour	69	(69, 1)	m s-1	float64
Sax1H	Wind Speed Maximum last 1 Hour	69	(69, 1)	m s-1	float64
Sax3H	Wind Speed Maximum last 3 Hours	69	(69, 1)	m s-1	float64

name	Long name	size	shape	units	type
Sax6H	Wind Speed Maximum last 6 Hours	69	(69, 1)	m s-1	float64
sq	Squall Indicator	69	(69, 1)	code	float64
ss	Sunshine Duration	69	(69, 1)	min	float64
Sx1H	Wind Gust Maximum last 1 Hour	69	(69, 1)	m s-1	float64
Sx3H	Wind Gust Maximum last 3 Hours	69	(69, 1)	m s-1	float64
Sx6H	Wind Gust Maximum last 6 Hours	69	(69, 1)	m s-1	float64
t10	Ambient Temperature 10 Min Average	69	(69, 1)	degrees Celsius	float64
ta	Ambient Temperature 1.5m 1 Min Average	69	(69, 1)	degrees Celsius	float64
tb	Wet Bulb Temperature 1.5m 10 Min Average	69	(69, 1)	degrees Celsius	float64
tb1	Soil Temperature -5cm 10 Min Average	69	(69, 1)	degrees Celsius	float64
Tb1n6	Soil Temperature -5cm Minimum last 6 Hours	69	(69, 1)	degrees Celsius	float64
Tb1x6	Soil Temperature -5cm Maximum last 6 Hours	69	(69, 1)	degrees Celsius	float64
tb2	Soil Temperature -10cm 10 Min Average	69	(69, 1)	degrees Celsius	float64
Tb2n6	Soil Temperature -10cm Minimum last 6 Hours	69	(69, 1)	degrees Celsius	float64
Tb2x6	Soil Temperature -10cm Maximum last 6 Hours	69	(69, 1)	degrees Celsius	float64
tb3	Soil Temperature -20cm 10 Min Average	69	(69, 1)	degrees Celsius	float64
tb4	Soil Temperature -50cm 10 Min Average	69	(69, 1)	degrees Celsius	float64
tb5	Soil Temperature -100cm 10 Min Average	69	(69, 1)	degrees Celsius	float64

name	Long name	size	shape	units	type
td	Dew Point Temperature 1.5m 1 Min Average	69	(69, 1)	degrees Celsius	float64
td10	Dew Point Temperature 10 Min Average	69	(69, 1)	degrees Celsius	float64
tg	Grass Temperature 10cm 10 Min Average	69	(69, 1)	degrees Celsius	float64
tgn	Grass Temperature 10cm 10 Min Minimum	69	(69, 1)	degrees Celsius	float64
Tgn12	Grass Temperature Minimum last 12 Hours	69	(69, 1)	degrees Celsius	float64
Tgn14	Grass Temperature Minimum last 14 Hours	69	(69, 1)	degrees Celsius	float64
Tgn6	Grass Temperature Minimum last 6 Hours	69	(69, 1)	degrees Celsius	float64
tn	Ambient Temperature 1.5m 10 Min Minimum	69	(69, 1)	degrees Celsius	float64
Tn12	Air Temperature Minimum last 12 Hours	69	(69, 1)	degrees Celsius	float64
Tn14	Air Temperature Minimum last 14 Hours	69	(69, 1)	degrees Celsius	float64
Tn6	Air Temperature Minimum last 6 Hours	69	(69, 1)	degrees Celsius	float64
tsd	SIAM Ambient Temperature 10 Min Average Std Dev	69	(69, 1)	degrees Celsius	float64
tx	Ambient Temperature 1.5m 10 Min Maximum	69	(69, 1)	degrees Celsius	float64
Tx12	Air Temperature Maximum last 12 Hours	69	(69, 1)	degrees Celsius	float64
Tx24	Air Temperature Maximum last 24 Hours	69	(69, 1)	degrees Celsius	float64

name	Long name	size	shape	units	type
Tx6	Air Temperature Maximum last 6 Hours	69	(69, 1)	degrees Celsius	float64
vv	Horizontal Visibility 10 Min Average	69	(69, 1)	m	float64
W10	Past Weather Indicator	69	(69, 1)	code	float64
W10-10	Past Weather Indicator for Previous 10 Min Interval	69	(69, 1)	code	float64
ww	wawa Weather Code	69	(69, 1)	code	float64
ww-10	wawa Weather Code for Previous 10 Min Interval	69	(69, 1)	code	float64
zm	Meteorological Optical Range 10 Min Average	69	(69, 1)	m	float64
iso_dataset	None	1	()	None	S1
product	ADAGUC Data Products Standard	1	()	1	S1
projection	None	1	()	None	S1
nhc	Low and Middle Cloud Amount Ceilometer	1	()	octa	S1
za	Background Luminance 10 Min Average	1	()	cd m-2	S1

S1 is a record type.

8.4 Result of station name counting program

Counting the number of times each station names appears.

```
{'06201': 111, '06203': 111, '06204': 111, '06205': 111, '06207': 111,
 '06208': 111, '06209': 111, '06211': 111, '06214': 111, '06215': 111,
 '06216': 111, '06225': 111, '06229': 111, '06233': 111, '06235': 111,
 '06236': 111, '06237': 111, '06238': 111, '06239': 111, '06240': 111,
 '06242': 111, '06248': 111, '06249': 111, '06251': 111, '06252': 111,
 '06257': 111, '06258': 111, '06260': 111, '06267': 111, '06269': 111,
 '06270': 111, '06273': 111, '06275': 111, '06277': 104, '06278': 111,
 '06279': 111, '06280': 111, '06283': 111, '06285': 111, '06286': 111,
```

```
 '06290': 111, '06308': 111, '06310': 111, '06312': 111, '06313': 111,
 '06315': 111, '06316': 111, '06317': 111, '06319': 111, '06320': 111,
 '06321': 111, '06323': 111, '06324': 111, '06330': 111, '06331': 111,
 '06340': 111, '06343': 111, '06344': 111, '06348': 111, '06350': 111,
 '06356': 110, '06370': 111, '06375': 111, '06377': 111, '06380': 111,
 '06391': 111, '78871': 111, '78873': 111, '78990': 111}
```

Each station name and a set of where the indexes that this station appears at:

```
{'06201': {0}, '06203': {1}, '06204': {2}, '06205': {3}, '06207': {4},
 '06208': {5}, '06209': {6}, '06211': {7}, '06214': {8}, '06215': {9},
 '06216': {10}, '06225': {11}, '06229': {12}, '06233': {13}, '06235': {14},
 '06236': {15}, '06237': {16}, '06238': {17}, '06239': {18},  '06240': {19},
 '06242': {20}, '06248': {21}, '06249': {22}, '06251': {23}, '06252': {24},
 '06257': {25}, '06258': {26}, '06260': {27}, '06267': {28}, '06269': {29},
 '06270': {30}, '06273': {31}, '06275': {32}, '06277': {33},
 '06278': {33, 34}, '06279': {34, 35}, '06280': {35, 36}, '06283': {36, 37},
 '06285': {37, 38}, '06286': {38, 39}, '06290': {40, 39}, '06308': {40, 41},
 '06310': {41, 42}, '06312': {42, 43}, '06313': {43, 44}, '06315': {44, 45},
 '06316': {45, 46}, '06317': {46, 47}, '06319': {48, 47}, '06320': {48, 49},
 '06321': {49, 50}, '06323': {50, 51}, '06324': {51, 52}, '06330': {52, 53},
 '06331': {53, 54}, '06340': {54, 55}, '06343': {56, 55}, '06344': {56, 57},
 '06348': {57, 58}, '06350': {58, 59}, '06356': {59, 60}, '06370': {60, 61},
 '06375': {61, 62}, '06377': {62, 63}, '06380': {64, 63}, '06391': {64, 65},
 '78871': {65, 66}, '78873': {66, 67}, '78990': {67, 68}}
```

8.5 Set of all station names

```
{'TWENTHE AWS', 'ARCEN AWS', 'CABAUW TOWER AWS', 'K14-FA-1C', 'Nieuwkoop
 ↪  BTP',
 'VLIELAND', 'WIJK AAN ZEE AWS', 'MARKNESSE AWS', 'DEELEN', 'OOSTERSCHELDE
  ↪  WP',
 'STAVENISSE', 'HOEK VAN HOLLAND AWS', 'HANSWEERT', 'ROTTERDAM GEULHAVEN',
 'D15-FA-1', 'P11-B', 'CADZAND WP', 'VLISSINGEN AWS', 'ROTTERDAM THE HAGUE
  ↪  AP',
 'BG-OHVS2', 'ELL AWS', 'EINDHOVEN AP', 'Nieuw Vennep BTP', 'OOSTERSCHELDE
  ↪  4',
 'EURO PLATFORM', 'WOENSDRECHT', 'NIEUW BEERTA AWS', 'TEXELHORS WP',
 'STAVOREN AWS', 'A12-CPP', 'HERWIJNEN AWS', 'Hollandse Kust Zuid Alfa
  ↪  (HKZA)',
 'FLAMINGO AIRPORT BONAIRE', 'MAROLLEGAT', 'IJMOND WP', 'BORSSELE ALFA
  ↪  (BSA)',
```

```
'AMSTERDAM/SCHIPHOL AP', 'DE BILT AWS', 'LAUWERSOOG AWS', 'GRONINGEN AP
 ↪  EELDE',
'BERKHOUT AWS', 'WILHELMINADORP AWS', 'F.D. ROOSEVELT AIRPORT ST.
 ↪  EUSTATIUS',
'VLAKTE VAN DE RAAN', 'TERSCHELLING HOORN AWS', 'HOOGEVEEN AWS', 'J6-A',
'VOORSCHOTEN AWS', 'LELYSTAD AP', 'GILZE RIJEN', 'DE KOOY VK',
'HOUTRIBDIJK WP', 'HUPSEL AWS', 'L9-FF-1', 'AWG-1', 'HUIBERTGAT WP',
'LICHTEILAND GOEREE', 'JUANCHO E. YRAUSQUIN AIRPORT SABA', 'IJMUIDEN WP',
'LEEUWARDEN', 'K13-A', 'HEINO AWS', 'Assendelft BTP', 'Muiden BTP',
 ↪  'VOLKEL',
'WIJDENES WP', 'F3-FB-1', 'MAASTRICHT AACHEN AP', 'WESTDORPE AWS'}
```

Set of station names: