February 17, 2023

Quality assessment of private weather station Netatmo

Viktor Israelsson (vi5015is-s@student.lu.se)
Bachelor of Science, Physics and Meteorology
15 credits, FYSK02

Supervisor SMHI
Heiner Körnich

Supervisor LU
Elna Heimdal Nilsson

Division of Combustion Physics

LUNDS UNIVERSITET

Contents

1 Abstract

Netatmo is a brand of private weather stations that over the past decade, in many countries, have grown to outnumber the number of government based weather stations. In most fields of research, a high number of data points can increase accuracy and precision. For a weather forecast, the more weather condition throughout an area the forecasters know about, the better the forecast. All of Sweden is the area of focus for this thesis.

Weather forecasts is an important function in any society, and even crucial for certain industries that are dependent of weather patterns one way or another. Therefore, if additional data points could be added to weather forecast algorithms, likely improvements of a weather systems predicted track and development could be done (such as; will there be rain? When and how much?). Therefore it makes sense to investigate the basic usefulness of private weather stations.

This bachelor project was built around investigating and evaluating Netatmo stations in Sweden. Netatmo stations are capable of measuring a couple of different types of parameters, namely pressure, temperature, wind strength and direction, precipitation and relative humidity. Precipitation is the selected parameter for this thesis. To this date, precipitation is less investigated in the scientific community, in comparison for example to temperature, which has a rather well evaluated accuracy.

The aims was to evaluate if it is worth doing further investigations of the Netatmo weather station network. The preliminary results provided by this bachelor project show some problems with the precision of the rain gauge featured with Netatmo, but the upside of having so many more points of data show some promise. Especially when handling the data from a statistical approach.

2 Introduction

Professional weather forecasting is, in most countries, retrieved from weather stations owned by the country's governing forecasting agency. Buying, setting up and maintaining these stations is quite expensive, which is why these stations are limited in number. However, over the past decade, private weather stations have become more and more popular. This thesis has made an investigation of the Netatmo weather stations, which is a popular alternative in the private weather station market.

While the Netatmo stations come with multiple challenges in terms of both precision and accuracy, the outnumbering of the professional stations is in itself a good argument to look in to how these private stations might be utilized for professional forecasting runs. In particular, it can be theorized that a higher spatial fidelity of data points might be useful for more local weather forecasts. There may be other reasons to make evaluations of these stations, like for marketing purposes; how well does Netatmo station data perform in comparison to professional stations? Having such generic "benchmark tests" could be valuable in the future. It is fully possible that future implementations of data will be done in ways that one can not foresee today, and that they could make use of such "benchmark tests".

All parameters (pressure, temperature, wind strength and direction, precipitation and relative humidity) a fully equipped Netatmo weather station is able to collect data of come with a range of difficulties in accuracy. If positioned over a surface consisting of anything else that the best practice entails like asphalt as an example, temperatures will likely be measured to be higher than expected. If positioned too high or too low above the ground also plays a roll, winds could become too strong, too weak or too turbulent. Other, perhaps less expected issues, may occur as well. An example of this is getting lag time. The Netatmo stations have a temperature sensor casing, which adds to the time the sensor itself measures the temperature, as heating is slightly slowed down. This lag time is decreased if the casing is removed by around ten minutes, which is not an insignificant time span [3]. In turn, this may be part of explaining why mean temperature in the morning is lower than for a station measuring temperature managed by a professional agency, in this case the UK Met Office.

For this thesis, the precipitation parameter appeared to be a good candidate to analyze. Searching and evaluating available papers that had investigated Netatmo revealed that precipitation was not as well investigated compared to other parameters.

Measuring precipitation correctly can be difficult under certain conditions. For example, if there are very strong winds, a rain gauge device (which is what Netatmo stations can be equipped with) may miss a lot of the rainfall, as precipitation may fall from a strong angle. But precipitation is also known as one of the parameters that needs additional evaluation and investigation [1]. It should also be noted that a Netatmo rain gauge does not measure all types of precipitation, only rain, as it is lacking a heated sensor or any other function that can melt solid precipitation. Since Sweden is a long country, about 1572 km between the most northern- and southern points, the temperatures and climate can differ a lot. Local variations can also be prominent, with rain or snow depending on location of the station. This needs to be taken in to consideration when evaluating rain data.

Being able to accurately measure precipitation has a direct societal importance, like making farming predictions, get an understanding of water reserves etc. Long term understanding how precipitation may change as an effect of climate change is also of great importance.[6]

The Netatmo-data was analyzed in relation to data from SMHI (Swedish Meteorological and Hydrological Institute, operating under the Swedish Ministry of the Environment), as gathered by professionally set up and maintained stations that are able to measure precipitation. Overarching questions, like how single Netatmo stations compare to single SMHI station(s), and how a large number of stations averages compare to one another, was a starting point for this thesis.

SMHI's stake in the thesis was to get an understanding, or at least indication, if it is worth investing time and money making further investigations of the usability of Netatmo stations. And, if possible, get an idea of what areas in weather forecasting (and climate research) the data may show a useful potential.

3 Background

To make accurate forecast runs, various types of data is essential to solve the equations that most forecast algorithms work with. Temperature, pressure, and relative humidity are parameters that the basic version of the Netatmo weather station measures on top of this. A buyer may add instruments to measure precipitation (rain only) and wind strength- and direction. But as this comes with an extra monetary cost, not all stations will be able to provide data with these parameters.

There are studies emerging that support the claim that the relatively new phenomenon of crowd funded weather stations, such as Netatmo, indeed have an area of usefulness. However, expected potential issues in over- and underestimations from readings are also highlighted. This include, but are not limited to:

- general placement such as cover from a building or other high-reaching obstacle in certain wind directions,
- not cleaning the tipping-buckets rain gauge properly from insects, twigs etc. that block from water tipping,
- not levelling the device with the ground properly, or
- careless owners cleaning or handling the device may result in tipping-bucket tips, creating measurements of artificial rain.

All of the above issues are discussed in [4].

In order to work with these unknown numbers of more or less faulty readings for private weather stations, De Vos et al [11] created a quality control algorithm, working with time-intervals where zero observations, high influxes and station outliers are flagged and handled. In regards to the high density of stations, mainly in urban areas, areas with lower density of data points are subject of lower accuracy. Consequently, the filter is not as successful in those areas. Despite this, the filter was successfully tested with a 1-year data set of rainfall in the Netherlands, and it was possible to construct a rainfall map over the country - showing good promise for using private weather stations when measuring rainfall.

3.1 Rain gauges

A Netatmo rain-gauge collects water with the help of a tipping-bucket, as per Figure 1. The tipping-bucket design is one of the most common designs used across the world [8]. On the inside, it operates with the help of tilting buckets, as seen in Figure 2. When it rains, water makes the buckets tilt and the number of bucket hits is measured using a magnet placed on the buckets. It is a fully automated device [7].

Figure 1: *The Netatmo rain gauge casing.*

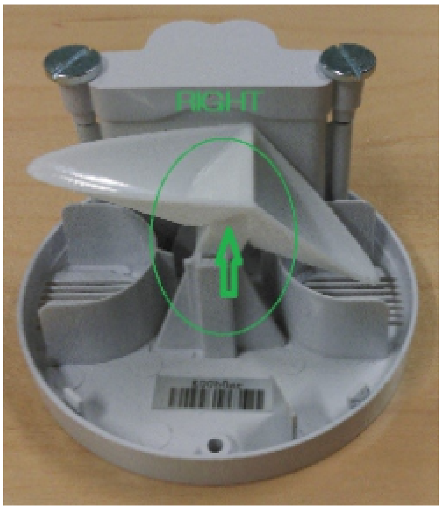


Figure 2: *The insides of a Netatmo rain gauge.*

There are however a number of other devices and methods to measure precipitation. Manual devices require that a bucket is emptied, whilst an automated empty itself one way or another.

SMHI manual stations are simplistic in nature, a jug (Figure 3) is collecting precipitation [8]. Solid precipitation is brought inside, and is carefully melted in a controlled manner, in order to avoid evaporation. The liquid water is measured in mainly the smaller of the measuring glasses (in millimeter) from Figure 4.

Figure 3: *A rain collector, used on manual SMHI precipitation stations.*

Figure 4: *Collected rain is measured manually using measuring glasses.*

There exist other automated designs than the aforementioned tipping-bucket, like optical devices. The basic principle of an optical rain gauge is that the refraction and absorption of an optical ray is changed dependent on the amount of precipitation [8]. The optical devices are used by SMHI, but only as a proof-reading method to secure data measurments. The main type used by SMHI is of the brand Geonor. With this, precipitation is measured using vibrating wire load sensors. Anti-freeze liquid melts solid precipitation, eliminating the need for electrical heating, which in itself can be a source error. A thin layer of oil will aid in preventing evaporation [10].

Figure 5: *SMHI uses automated rain gauges of the brand Geonor. Depicted is model T-200B on a stand, along with wind shields.*

3.2 The wind effect

The most common and largest factor contributing to faulty readings of precipitation is the wind [8]. Depending on wind conditions closer to the ground, the rain may fall with quite an angle, depending on wind strength. This results in a deficiency of rain amount compared to rain falling with irrelevant winds. During such events, (private) weather stations that are not placed according to regular station placement practices [8], may receive less rain than it should. Obstacles like trees, houses etc. that lie too close to the stations, can give the rain gauge unwanted shelter. Nearby buildings or obstacles may also be problematic for the wind, as the micro-meteorological scaled wind field might be affected, with up- and down drafts, unpredictable turbulence and similar.

When it comes to the professional SMHI stations measuring snow, the precision is lower than measuring rain, as snow is even more sensitive to winds than rain. However, winds do affect professional stations as well. Using the Beaufort wind scale, in a class 3 wind (gentle breeze, 3.4 - 5.4 m/s) the loss for rain is 3.5% and for snow 8.5%. For a class 7 event (moderate gale, 13.9 - 17.1 m/s) 12% rain is not gathered, and 35% for snow [2]. The losses for a Netatmo station in a windy scenario is not examined in this thesis.

3.3 Other sources of error and error mitigation

Adhesion is another source for receiving measuring errors (water getting stuck on the rain gauge after emptying). Evaporation, frost (which, in Sweden, is not supposed to not be part of the measurement) are other examples that may lead to faulty readings. Professionals emptying and managing a rain gauge manually can mitigate these problems fairly well, especially minimizing adhesion.

There are around 600 SMHI stations that measures precipitation, a majority of these are being emptied manually. Only about 120 are automatic. These automatic stations are generally at a higher risk of introducing errors [8]. Ways to mitigate these error sources naturally exist, but may not be 100% perfect. Wind screens can for example be set up to help minimizing the wind issues, like the Alter wind shield [2] that is used on self emptying precipitation stations. While the Netatmo rain gauge does empty itself automatically, it does not have a wind shield. These are factors that add potential errors in measurements.

4 Method

This project initially focused on monthly averages in order to get a more general and statistical idea of station performance, comparing Netatmo-data with SMHI-data using temporal averages. Looking specifically at precipitation, the given Netatmo-data ranging from 2015 to 2019, was analyzed through statistical measurements as a function of a temporal range (daily, yearly...). Python scripts were created in order to manage and visualize the data.

The main analyses consist of two parts, single station- and region comparisons. The two methods share some conditions, like data must be recorded throughout the given period of 2016 - 2019 without interruptions, and also have a temporal range of one data point per month.

Gaining access to the Netatmo-data is a paid service, provided by SMHI. An issue this data had was that December-data was unavailable, for all years. This was being worked on by SMHI-personnel to retrieve, but the problem was not resolved for the duration of writing this thesis. Therefore, the analysis is performed with data from December missing in the Netatmo data-set.

4.1 Single station comparisons

Being able to more directly compare Netatmo stations with professionally maintained stations should give a good indication of Netatmo stations performances. In order to achieve this, a script was first created to find Netatmo stations close to SMHI stations geographically. The latitude was set to have a maximum distance of 1/111 of a nautical degree apart, which equals to a maximum distance of 1 km latitude. Longitude condition had to be relaxed a bit, and was set to 20 km. Calculated distances can be seen in table 1.

Table 1

LOCATION	STATION DISTANCE (km)
Gunnarn A	2.5
Hofors	0.6
Komperöd	19.4
Vårgårda D	15.3

Distance between SMHI- and nearby Netatmo stations

Four candidate locations met all criteria, and were selected to be included in the study, as per Figure 6.

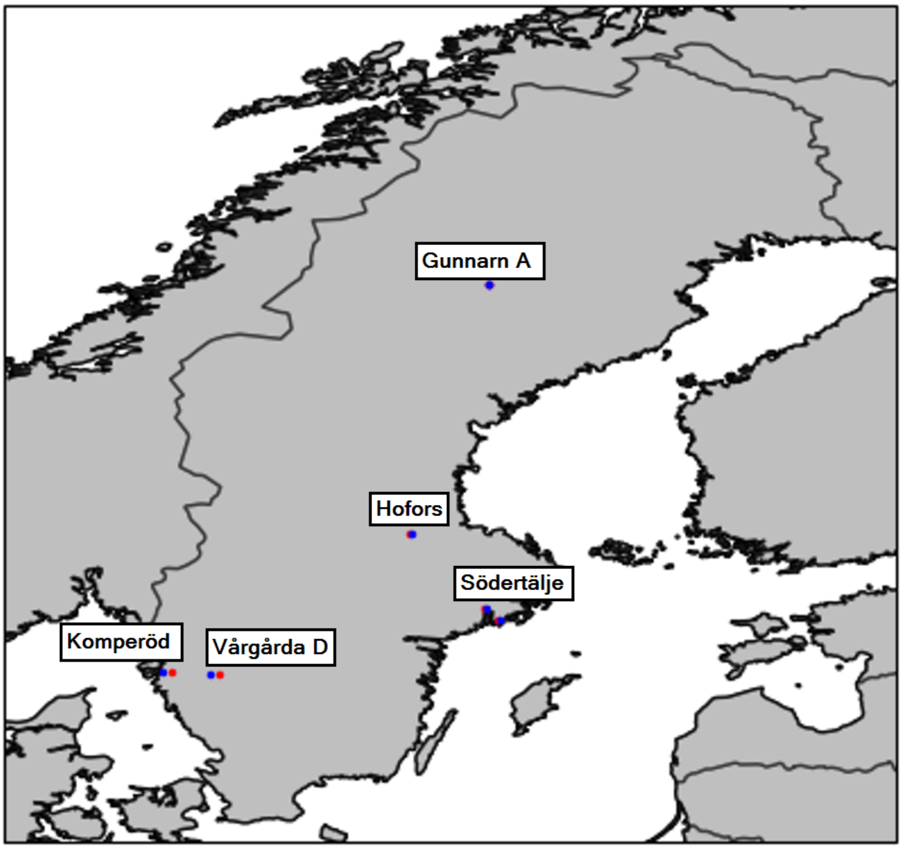
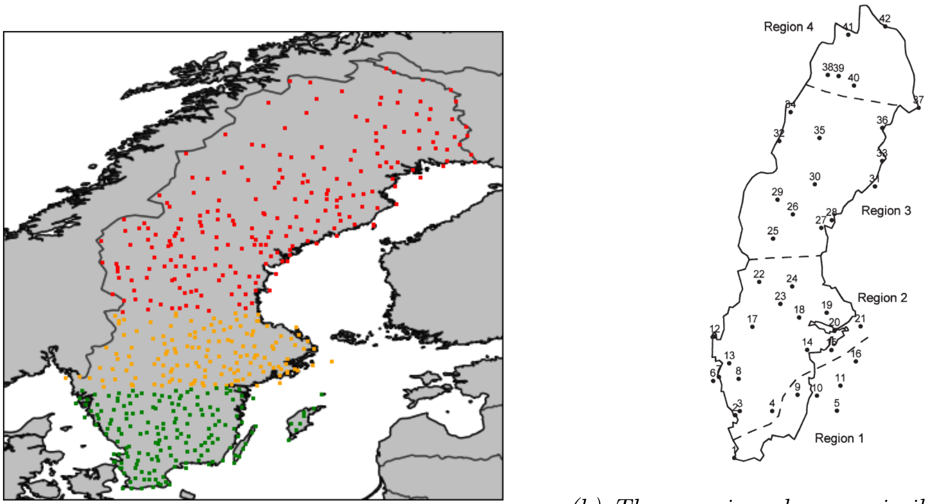


Figure 6: *SMHI stations are shown with red dots, Netatmo stations blue. The four stations near Södertälje were cut, as they were lacking consecutive data in the range 2016 - 2019.*

4.2 Region comparisons

Sweden was split in three regions, as per Figure 7(a). The split is based on the rough, estimated latitude of the regions in Figure 7(b), which in turn is based on regional precipitation variability and is utilized in climate research [5]. The motivation for this selection is simply to have something scientific and meteorological (or climatological) as a basis for the selection. The horizontal latitudinal breaking points are simpler to program, compared to the more complex dashed lines making up the regions in Figure 7(b).



(a) SMHI stations split in to three regions.

(b) These regions have a similar precipitation variability.

Figure 7: *The latitudes of the regions split in (a) was selected to resemble the split of the regions in (b)*

SMHI stations split as follows: 61° >= latitude > 58.6°, giving a "north", a "mid" and a "south" region, with stations coloured red, orange and green respectively.

4.3 Netatmo script data structure

The Netatmo-data is structured in monthly folders. Each station, that for the most part represented by at least two csv-files, also comes with a json-file. The basic parameters a Netatmo station records is temperature, relative humidity and pressure. Instruments to record wind and precipitation (rain) may be purchased as extras, which is why there is less rain- and wind data available in any month. The measured data in the csv-files can be seen in table 2 and an example of filenames in Figure 9.

Table 2

TYPE OF DATA	FILENAME SUBSTRING	UNIT
Temperature	outdoor	° Celcius
Relative humidity	outdoor	%
Pressure	pressure	hPa
Rain	rain	cm
Wind angle	wind	degrees
Wind speed	wind	km/h
Gust angle	wind	degrees
Gust speed	wind	km/h
Timestamp	all data	seconds

All data is marked with a Unix timecode timestamp (which start counting time from 1970-01-01, 00:00). Some files contain multiple types of data, as seen in the "filename substring" column.

The metadata is stored in the json-files, with a nested structure as seen in Figure 8.

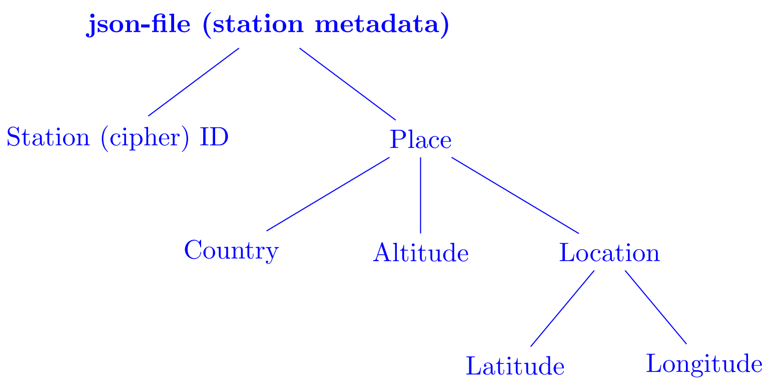


Figure 8: *All nested data is in code represented as a dictionary datatype. The Figure showcase each dictionary's Key, apart from Latitude and Longitude, which are contained in a list, that list being the Location Key's Value.*

An example of a typical station and its related data files can be seen in Figure 9. Each json-file has a number in the filename. This number is, for each month, represented in each related csv-file, and this is the only thing that links the json-file and csv-files. To complicate things, this number is sometimes not consistent throughout the months. The script checks for and deals with eventual changes of the number in the filename.

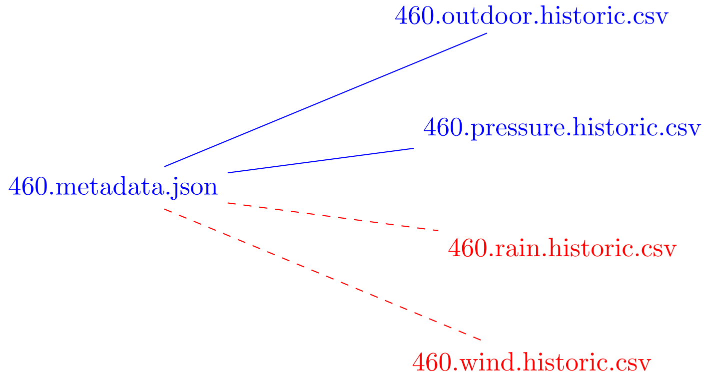


Figure 9: *Blue lines represent standard measurements from a basic Netatmo station, red dotted lines represent data from instruments that can be purchased as extras. The "outdoor" csv contain relative humidity and temperature. Note that it is possible for a station to not record the standard measurements.*

Being able to work with the data in a versatile manner, the scripts have been created to be as flexible as possible. Although rain is the parameter this thesis focuses on, the idea was to build a script in a way so any parameter can be processed and plotted, and then later analyzed. While this feature is not fully met, a large portion of the Netatmo code-base is built as such, only needing a few adjustments in order to operate on any other parameter.

Certain assumed issues and discrepancies in data have been taken in to consideration in the code structure, like if the number in the filename is changed, or if a wanted station is lacking data during a certain period of time. Being able to check if a station is changing its coordinates over time was a wanted feature, but got cut due to lack of time.

After these filters are run, a list of stations having data that is more or less consecutive throughout the period of 2016 - 2019, is used to perform the analysis. In order to achieve these feats, the code has been structured as seen in Figure 10.

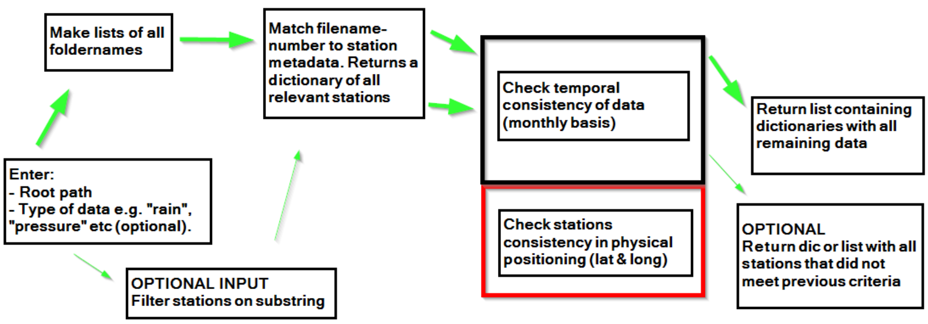
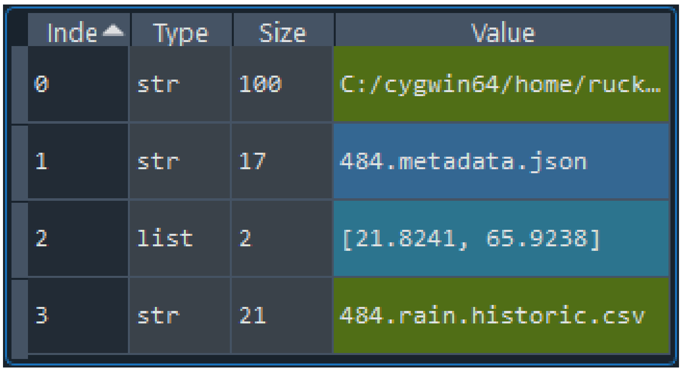


Figure 10: *High level block scheme of how Netatmo-data is managed, in order to make it easily accessible for the purposes of this thesis. The box in red indicates a feature that was cut due to lack of time.*

The end "product" using the script is a list, that contain a number of dictionaries, which represents one month each (dictionaries and months are used interchangeably from hereon). Each dictionary contain a Key and a Value. The Key is the stations unique Cipher ID. The Value is a list, which consist of (in order of appearance): Path on disk, json-filename, and relevant csv-filename(s), as seen in Figure 11.



Inde	Type	Size	Value
0	str	100	C:/cygwin64/home/ruck…
1	str	17	484.metadata.json
2	list	2	[21.8241, 65.9238]
3	str	21	484.rain.historic.csv

Figure 11: *An example of how the dictionaries are structured, viewing the Values of a single station. The Key, i.e. the stations Cipher ID, is not seen in the picture.*

Making use of this now structured list of dictionaries, code was created to use the metadata json-files from each station to split the full list into different regions. After the split, actual data is loaded (see previously discussed in section "Region comparisons" for more info). Next, a separate code was created to calculate the region average for each consecutive month.

A similar approach was done with the individual stations. The structured list of dictionaries is used to identify the Netatmo stations that were found to be close to one another (as discussed in section "Single station comparisons"). In code, this is done using the unique Cipher-ID. Then, data from these relevant stations are being pulled, and similar to the regions, a monthly average is being calculated.

4.4 SMHI-data structure

The SMHI-data was available in one single xlsx-file, as provided by SMHI. The data itself was already accumulated monthly and generally more readily available than the Netatmo-data. The code-base created ended up being functional, but unfortunately not as well structured and versatile as the Netatmo code-base.

Most checks done for the Netatmo code-base was also performed for the SMHI code base, like filtering out all stations that did not have data in every month for the time period of 2016 through 2019. A similar region-split was performed, and after that, a monthly average using all stations in each region respectively was performed and plotted. The data from the four individual, single stations, could simply be plotted outright, as this data already was presented as monthly averages.

4.5 Excel and plots

As a final stage, the now calculated structure of the data was lifted in to Excel-sheets. This was done as a quality-check step (to make sure the monthly averages from SMHI and Netatmo-data were aligned by the right months, among other things). It was simply easier to get a good overview of the Netamo- and SMHI-data listed next to one another in Excel, as seen in Figure 12.

DATE	GUNNARN A NETATMO	SMHI manually
'16-01',	1.56	22.3
'16-02',	10.918	34.5
'16-03',	15.895	17.4
'16-04',	59.697	52.9
'16-05',	26.642	22.3
'16-06',	48.294	66.6
'16-07',	35.661	42.6
'16-08',	86.567	95.4
'16-09',	22.742	20.5
'16-10',	5.458	3.8
'16-11',	9.2	49

Figure 12: *An excerpt from a spread sheet where monthly SMHI- and Netatmo data was listed.*

Another sanity-check was made and tested on the southern region. Using the SMHI-data original xlsx-file, averages from each month in the southern region was accumulated manually and lifted into the Excel-data sheet containing the rest of the results. For the manual calculation, each month would generally have a higher number of stations than what the script would use in the analyzed data, as no condition for the stations being consecutive in every month was applied.

5 Results

Plotted data ranges from January 1, 2016, with the last month being November 2019. As previously mentioned, December was not available in Netatmo-data. These months are left blank in the plots. The SMHI-data from December 2016 was removed in all comparisons with Netatmo-data, in order to get a mutual ending point.

In Figure 13, a comparison between using the SMHI code-base to get the southern region average, and manually calculating the same directly from the SMHI xlsx Excel-file is plotted (stations with incomplete data of the period 2016 - 2019 is removed with the script, but not with the manual method, as discussed in the ”Excel and Plots”-section).

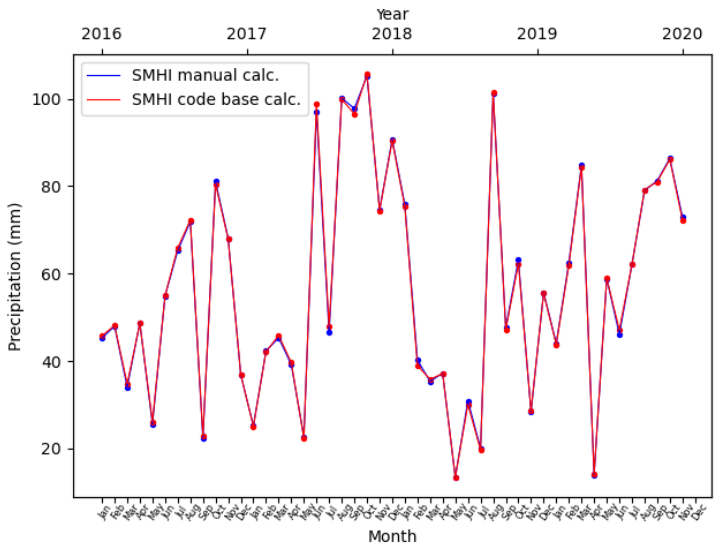
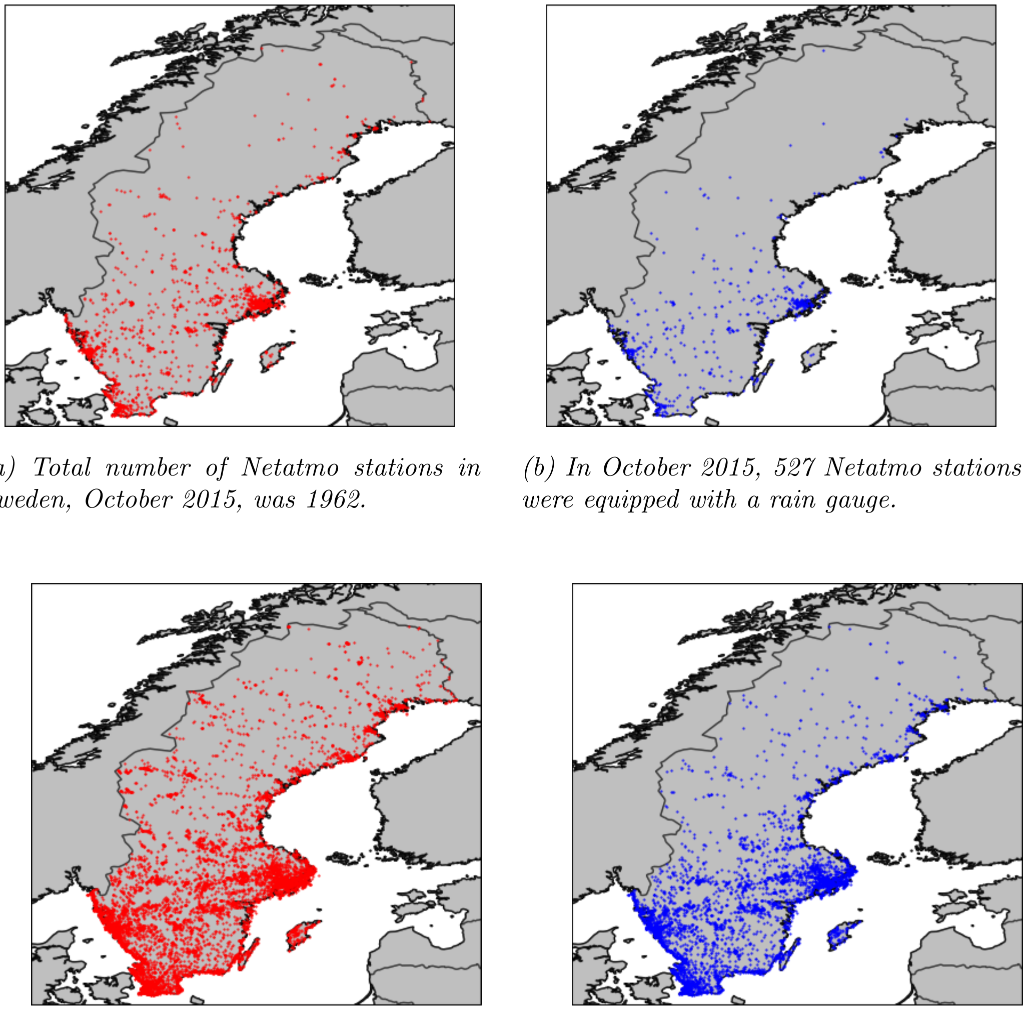


Figure 13: *Region south, plotting the region average from the SMHI Python script results, and the manual calculation for the same averages, performed in Excel and plotted in Python.*

It was noted that the number of Netatmo stations was, during the period of October 2015 to October 2019 on a steep increase. As seen in Figure 15 (a) and (c), the total number of stations have increased by over six times. The number of stations featuring a rain gauge out of these total stations saw an increase of almost ten (9.6) times, as per Figure 15 (b) and (d). This number, both total number of stations and stations featuring rain gauges, has most likely continued to increase since.



(a) Total number of Netatmo stations in Sweden, October 2015, was 1962.

(b) In October 2015, 527 Netatmo stations were equipped with a rain gauge.

(c) Total number of Netatmo stations in Sweden, October 2019, was 11853.

(d) In October 2019, 5058 Netatmo stations were equipped with a rain gauge.

Figure 15: *(a) - (d) show Netatmo station growth from 2015 to 2019, both base package and rain gauge only*

The regional split of the SMHI stations can be seen in Figure 7(a). The total number of stations accumulating precipitation, having any readings in the selected time period for the analysis, was 750. However, only 549 of these stations had data in each month. Out of the 549, 227 belongs to the north region, 148 in the mid region of 148 and 174 in the south region.

In Figures 16, 17 and 18, SMHI and Netatmo monthly average data from all relevant stations is plotted.

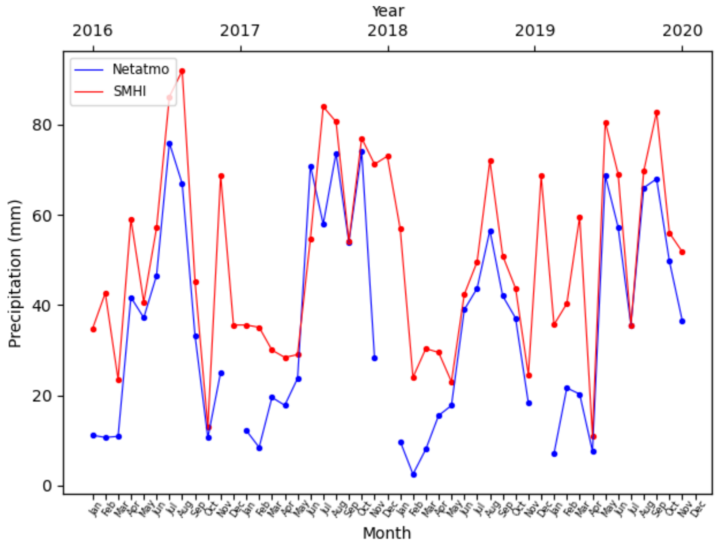


Figure 16: *The northern region, featuring monthly averages of Netatmo- and SMHI-stations in Sweden, located with a latitude equal to or higher than 61. The summer months are generally seen to have a more similar result than the rest of the year.*

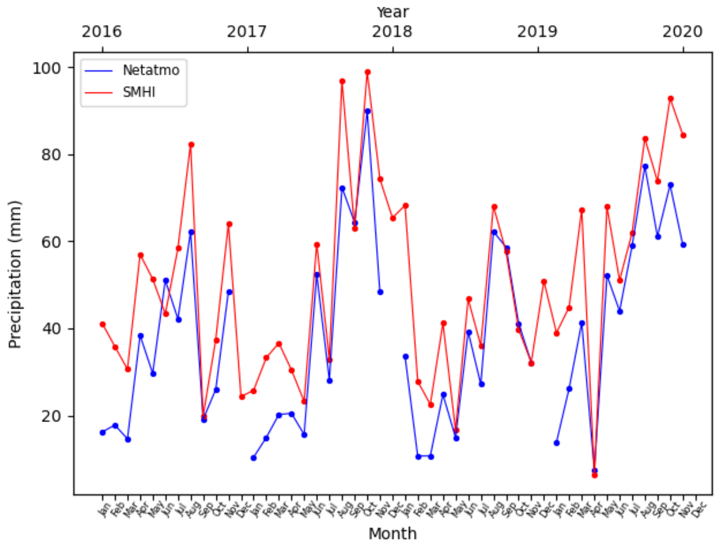


Figure 17: *The mid region, featuring monthly averages of Netatmo- and SMHI-stations in Sweden, located with a latitude less than than 61, and equal to or higher than 58.6. The regions data overall coincide more than that of the northern region, from Figure 16.*

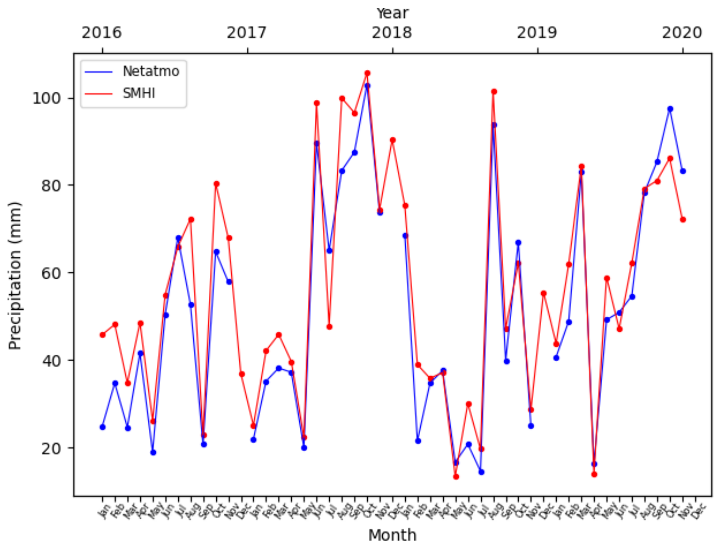


Figure 18: *The southern region, featuring monthly averages of Netatmo- and SMHI-stations in Sweden, located with a latitude less than 58.6. Compared to the other two regions as in Figure 16 and 17, the stations averages are more similar.*

In Figures 19, 20, 21 and 22, SMHI-data from the single stations is plotted, along with the montly average of their respective, nearby Netatmo station.

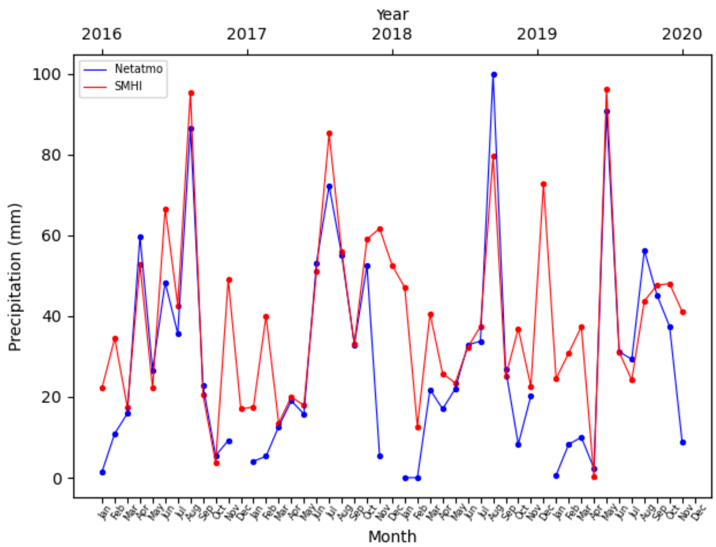


Figure 19: *Gunnarn A, SMHI and Netatmo monthly averages.*

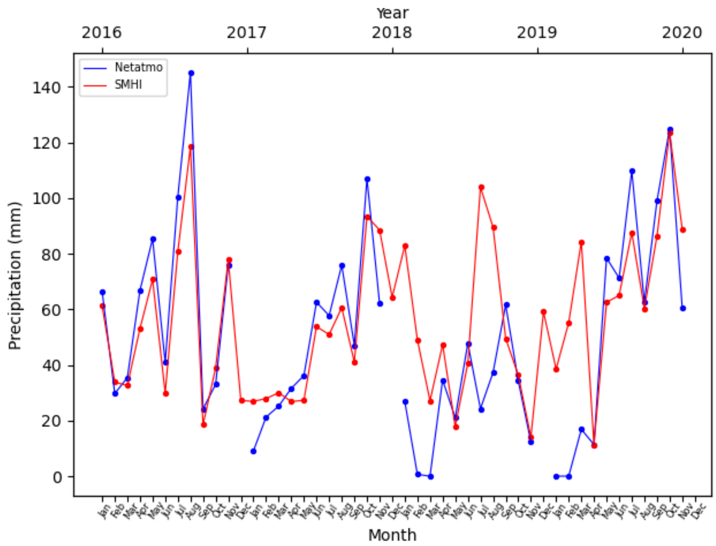


Figure 20: *Hofors, SMHI and Netatmo monthly averages.*

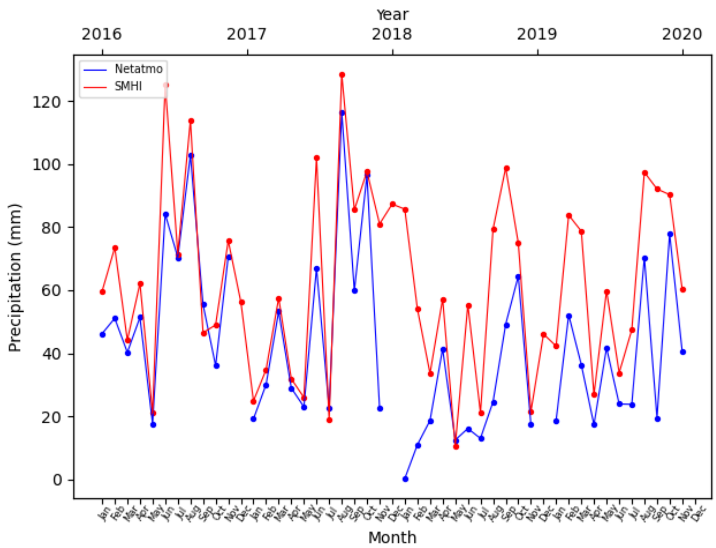


Figure 21: *Vårgårda D, SMHI and Netatmo monthly averages.*

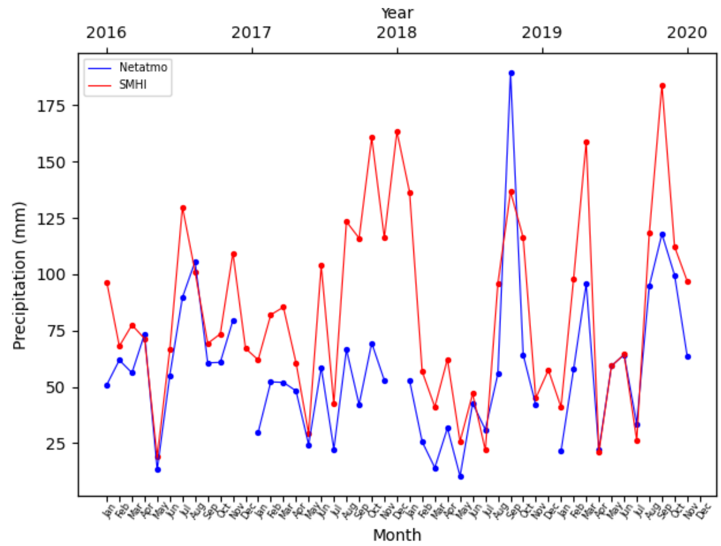


Figure 22: *Komperöd, SMHI and Netatmo monthly averages.*

Figure 23 shows a plot of SMHI and Netatmo regions, where the normalized difference between the stations have been calculated, based on the monthly average data, as seen in Figures 16, 17 and 18.

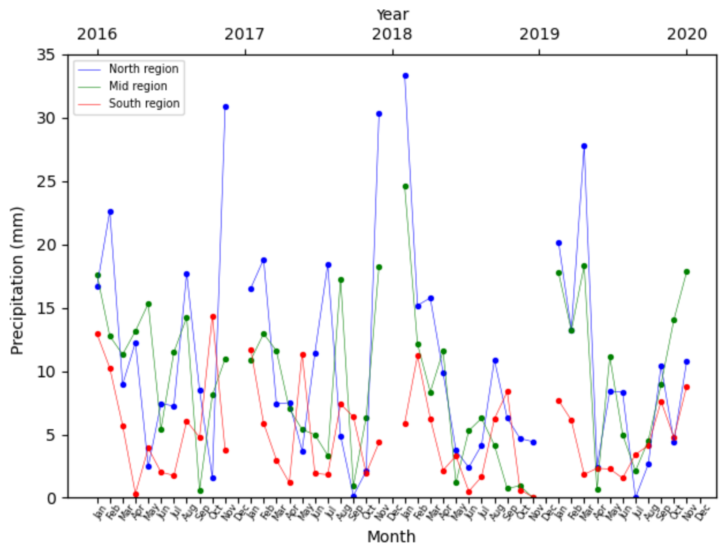


Figure 23: *Normalized difference on monthly averaged data, regions.*

Table 3 shows the mean value of the normalized difference between SMHI and Netatmo on monthly averaged data from all regions, averaged.

Table 3

REGION	NORM. MONTHLY AVG. MONTHLY MEAN (mm)
South	5.00
Mid	9.29
North	10.84

Mean value of normalized difference on monthly averaged data, as calculated using Excel.

Figure 24 shows monthly average normalized difference between the Netatmo and SMHI single stations plotted.

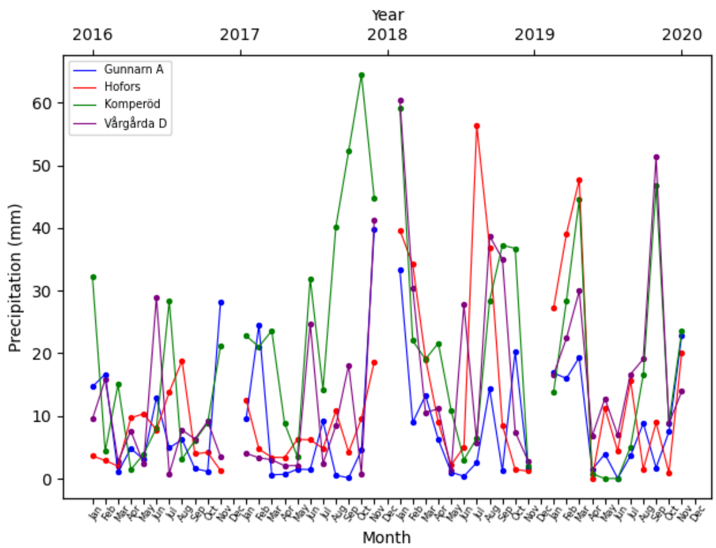


Figure 24: *Normalized difference on monthly averaged data, single stations.*

Table 4 shows the mean value of the monthly average normalized difference between the Netatmo and SMHI single stations plotted.

Table 4

STATION	NORM. MONTHLY AVG. MONTHLY MEAN (mm)
Gunnarn A	8.94
Hofors	12.58
Komperöd	20.35
Vårgårda D	14.36

Mean value of normalized difference on monthly averaged data for the single stations, calculated using Excel.

6 Discussion

Figure 13, showing the difference between the manually calculated monthly average (where some stations was lacking consecutive data throughout the period of 2016 - 2019), and the same region data calculated with the script, turned out to be very small. It seems there still is enough stations to get a stable statistical average. This result imply that, despite the script having filtered out a number of stations to be able to work with the same number of stations every month, the script is very accurate.

The same approach could have been interesting to apply to the Netatmo-data, but this check was down-prioritized in favor of other results for this thesis. If using a later starting date than 2016, and the result of Figure 13 in mind, it seem quite possible that the difference would be minimal though, especially if looking at regions with a large number of stations.

Because of time being short when starting to manage this data, the code-base created to manage the SMHI-data ended up not being as well structured and robust as the Netatmo code-base. Finding errors in the code along the way, this led to some time-consuming extra tasks, that likely could have been avoided if a better initial design of the code would have been structured.

6.1 Regional analysis

Analyzing the region data from Figures 16, 17 and 18, a somewhat mixed result in terms of precision should be apparent between the averaged Netatmo- and SMHI-data. The SMHI-data come from stations that are much more evenly spread out in all of Sweden, compared to the Netatmo-data (as was seen in Figure 7(a) and 15 (b) respectively). As a note, regional geographical differences is not taken into account in this thesis.

Using the quality-controlled SMHI-data as a reference, the expectation was to have Netatmo-data generally showing lower values, as no quality control for setup, maintenance etc, is being performed on these stations. As previously discussed, a faulty placement would most commonly mean that not as much rain is collected for a number of reasons. It should also again be noted that the months of December on the Netatmo results are missing from the delivery, and was not retrieved in time for the duration of this thesis being worked on.

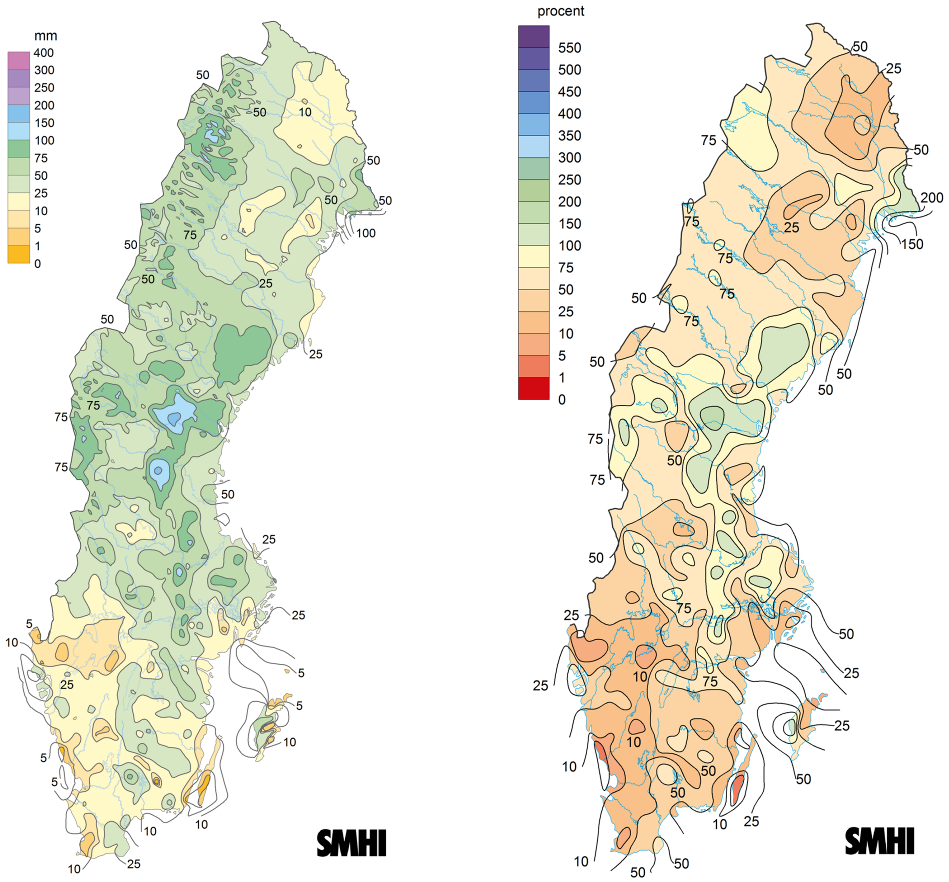
A general pattern can be seen, especially in the northern region, of how the winter- and spring months often show much lower values than the SMHI-data. This is because of the Netatmo rain gauge lacks the feature of melting snow, meaning no (or at least not all) precipitation in the form of snow being measured. For both the regions- and in particular the individual stations, monthly average temperature would have been an interesting addition to more clearly rule out data from periods of sub-zero temperatures.

It is likely that the fewer number of Netatmo stations and uneven spread of the same in the northern region, plays a role. Just geographically speaking, there is a big difference throughout the landscape of the northern part of Sweden. Some months show quite a lot lower amounts of rain, possibly implying rather bad performance. However, looking at the mid region, values are overall a bit closer to what the SMHI-data shows. The south region looks even better, and the Netatmo and SMHI values are starting to look rather similar. At least on average, Netatmo stations perform pretty well when getting a large number of stations to work with over a large region. This points towards a high number of stations simply will minimize the errors and make them insignificant for unchecked and potentially faulty station readings. This also implies that if the stations were indeed checked and served, quality may increase additionally. For the southern region, winter months generally are a lot warmer, and thus much less precipitation falls as snow. This help in making the overall graph over the southern region look a lot more like the SMHI-data, compared to the mid- and northern region.

In Figures 15(a)-(d), there is a clear rise in number of stations over time. And if the usability of the Netatmo stations as base-data for future forecast implementations indeed is increased with the number of stations; crowd sourced, unchecked Netatmo stations could prove to have some usefulness. At least as averaged data seem to show some promise, with an increasing number of stations operational likely improving the accuracy of the results.

Yet another sanity check was done towards the end of this project on the result from the regions, using data from SMHI's service "Månadens väder och vatten i Sverige" (Monthly weather and water in Sweden) of July, 2018 [9].

This was an unusually hot and dry month, as portrayed by the deviation gradient map seen in Figure 25(b). Figure 25(a) shows the monthly, accumulated precipitation amount. Just by making a visual evaluation and comparing these gradient maps to July 2018 regional averages from Figure 16, 17 and 18, the plotted data match up well with the gradient maps. Naturally, the SMHI-data plotted in the region Figures should be based on the same data that was used in Figure 25(b). Figure 25(a), but also just making a visual average assessment of the regions agrees well. Any relevant month for this thesis is represented on the SMHI web page [9].



(a) Gradient map of accumulated precipitation in Sweden, July 2018.

(b) Deviation from average rain amounts in Sweden, July 2018.

Figure 25: *Gradient maps of accumulated precipitation and average rain amounts can be used to make a quick assessment of precipitation. July, 2018, is the month of choice.*

6.2 Individual station analysis

Glancing at the individual stations close to SMHI stations Gunnarn A (Figure 19), Hofors (20), Vårgårda (Figure 21) and Komperöd (Figure 22), a somewhat different story is shown. These stations does not show the accuracy seen in the south region from Figure 18. In this case, good accuracy mean that the values may be off often, but not that much off overall. Looking at precision however, Gunnarn A, being the most northern station of the four, show really high precision during the summer months (i.e. results of individual months is very close to the SMHI-data). These months show a pretty expected behavior, save August 2019, which show a higher accumulation of rain than the nearby SMHI station. This unexpected anomaly appear in a handful of months in the individually selected Netatmo stations, save the one in Vårgårda, Figure 21. The reason for this is unclear, but in most cases, it is a summer-related anomaly. This could for example point towards nearby water sprinklers contaminating the data. Even if the Netatmo and SMHI stations are close, local weather variations such as highly local, convective showers (typical ”summer storms”), could also be a contender for explaining these mostly summer-related anomalies.

6.3 Normalized monthly average difference

The normalized monthly average difference plots of the regions and single stations, Figure 23 and 24, along with table 3 and 4 makes clear the notion that there is quite a difference between the Netatmo and SMHI stations.

However, both the plots and the mean values in the tables fail to take the fact that Netatmo stations can only deal with precipitation falling as rain into account. This is clear looking at the winter months, especially 2017-2018. Table 3 clearly shows how the difference is smaller in the southern region with about 5 mm, while the northern region counts in at around 11 mm.

Concerning the single stations, the differences are higher, as expected. However, surprisingly the most northern station, near the SMHI station Gunnarn A, shows the smallest difference of 8 mm, while the southern stations near Komperöd and Vårgårda show values up to 20 mm. The reason might be related to the respective distance or topographic features and require a more detailed study.

Additionally, looking at a much smaller temporal scale, like hours or less, would likely give a more representative and fair result. As would just looking at the summer months, which is quite obvious in Figure 23, at least when evaluating the southern region’s differences.

7 Conclusion

This thesis has taken an overarching, first step in comparing rain amounts and analyzing Netatmo-data in relation to SMHI-data. The Netatmo data-set lacking quality control, and the SMHI data-set having been quality controlled. A Python-script was created in order to manage the data and make the comparisons, as well as plotting and visualizing the results. The data was calculated as monthly averages, then split regionally. Monthly averages for four individual Netatmo stations and a, respective, nearby SMHI station was also part of the analysis.

Coding in Python has taken a majority of the time spent on this project. As the author was rather new to coding, there are likely a number of improvements and different choices that could be made in order to both make the code more compact, and more efficient, in order to run faster. For example not trying to rush results, even if time is short. However, the Netatmo code-base at least had a more structured design compared to the handling of the SMHI-data, which was a bit more rushed. This mean the Netatmo code-base ended up being a more robust basis if further implementations and features were to be made. It is also possible that looking into some kind of data-base structure feature, the code could have been improved overall. The author received this suggestion late, towards the end of the project, which meant this was not investigated at all.

Concerning precipitation under non-freezing conditions, Netatmo stations in great numbers over a large area do points toward having some areas of usefulness. At the very least when making statistical use of them. Considering the vast amount of stations around, the size of the areas that can be useful should effectively shrink. The statistical number of stations required per square kilometer for being statistically useful is, however, nothing that has been investigated in this thesis.

When it comes to individual stations, the analysis of this thesis imply that single stations data points are quite a bit less reliable than a quality-controlled SMHI station.

8 Outlook

From hereon, further analyzing Netatmo- and SMHI-data, a higher temporal fidelity would be of interest. A better picture of station accuracy could be determined, especially if taking not only temperature into account, but also wind-speed and direction. In terms of individual station performance, at least one proper Netatmo reference station could be set up in close proximity to an SMHI station. Properly setting up and maintaining these stations would naturally give an even better reference of Netatmo's strengths and weaknesses at its best, in comparison to SMHI stations. If setting up more than one reference station, different geographical locations featuring different types of regional yearly weather conditions could be interesting too, in the case that the Netatmo stations instruments perform better or worse under certain conditions.

Future improvement of comparing Netatmo with SMHI-data for precipitation (rain), could fare well from having automated proof-checks of the data. As an example, scripts could be written that check for when each individual Netatmo station is experiencing sub zero temperatures. Simply working with Netatmo's own temperature readings would likely be sufficient to use as threshold to flag rain data as potentially useless.

With a good understanding of how a Netatmo station should perform, nearby enough SMHI stations could be used as reference points and make predictions of how much rain a certain region (containing Netatmo stations) should get. If the Netatmo station(s) gather rain outside a certain range, a script checking wind-conditions could kick in, in order to see if the wind blow with a certain strength and from a certain direction. If this direction (and/or strength) start showing a pattern of lower amounts of rain than predicted, station(s) could be flagged for this. When using the data in other applications (such as weather forecast algorithms), flagged data could be handled, removed or possibly be compensated. An example of this could be using the results as suggested above for close proximity Netatmo- and SMHI stations, to develop compensation-tables to manage these errors.

As for the individual stations analysis, local weather variations could be interesting to investigate in the months where anomalies shows up, at a higher temporal fidelity - weekly, daily, hourly or even more zoomed in (Netatmo rain gauge send data about every fifth minute). This too could serve as quality-control mechanism.

Additionally, The Netatmo rain gauge likely will collect snow when it is snowing, and if there are quick shifts in temperature, this snow may even melt in order to be registered data. The time span however might in that case not be the usual five minutes, rather it can look like a larger amount of precipitation was falling at a certain time, when in fact there was no precipitation at all - the temperature having gone above zero would instead be the trigger, creating a lag-time of sorts for the readings. The potential behavior of such events could be analyzed if making a nearby-station analysis, with a much higher temporal scale of the readings - maybe even down to the five minute mark.

Checking the efficiency of professional wind shields on Netatmo stations could also be an interesting point. Also getting a better understanding of adhesion, frost, evaporation and other slightly smaller problems (in comparison to the wind problem) would be good in order to, whenever needed, update data-sets with compensated values. The spread of Netatmo stations in the northern part of Sweden is fairly bad. If wanting to use Netatmo-data in the future, it might be an idea to see if infrastructure could be shared between SMHI's own professional station, and SMHI-owned Netatmo stations. Thus they would be placed in close proximity of one another, and cover positions where no Netatmo stations may be located over vast distances. This at least might be better than not having a Netatmo station nearby, and also give a bit of redundancy of these, often far out, stations.

Overall, the vast, steadily growing number of Netatmo stations is looking like a cautiously potent complement for being used in future, professional weather data applications and perhaps even forecasts. This data may even provide useful local input to urban heavy rain, rural farming and water managing in remote sites such as for hydro power stations, generally located in the Swedish mountains.

More work needs to be done in order to fully understand the accuracy and precision of the different parameters, however, which is crucial in order to implement Netatmo-data sets in such applications. Considering the vast number and close proximity of these stations in many areas, it would be interesting to look in to how local, short term, weather forecasts as a whole could be improved upon, perhaps starting in smaller regions featuring a high population of Netatmo stations.

References

[1] Potthast R. Acevedo W. and T. Kratzsch. "Netatmo Dataset Evaluation." In: *Deutscher Wetterdienst, Frankfurter Str. 135, 63067 Offenbach, Germany. Provided through personal communication.* (2020).

[2] H Alexandersson. *Korrektion av nederbörd enligt enkel klimatologisk metodik. SMHI, Meteorologi, Nr 111.* 2003.

[3] Jonathan Coney et al. "How useful are crowdsourced air temperature observations? An assessment of Netatmo stations and quality control schemes over the United Kingdom." In: *Meteorological Applications* 29.3 (2022), e2075.

[4] LW De Vos et al. "Hydrometeorological monitoring using opportunistic sensing networks in the Amsterdam metropolitan area." In: *Bulletin of the American Meteorological Society* 101.2 (2020), E167–E185.

[5] Cecilia Hellström et al. "Comparison of climate change scenarios for Sweden based on statistical and dynamical downscaling of monthly precipitation." In: *Climate Research* 19.1 (2001), pp. 45–55.

[6] *Hur klimatet förändras - Nederbörd.* URL: `https://www.klimatanpassning.se/hur-klimatet-forandras/klimateffekter/nederbord-1.21297` (visited on 11/21/2021).

[7] *Netatmo - Help center.* URL: `https://helpcenter.netatmo.com/en-us/smart-home-weather-station-and-accessories/measures-and-calibrations/my-smart-rain-gauge-always-measures-0-mm-of-rain`.

[8] *SMHI - Hur mäts nederbörd?* URL: `https://www.smhi.se/kunskapsbanken/meteorologi/regn/hur-mats-nederbord-1.637`.

[9] *SMHI - Juli 2018 - Långvarig hetta och svåra skogsbränder.* URL: `https://www.smhi.se/klimat/klimatet-da-och-nu/manadens-vader-och-vatten-sverige/manadens-vader-i-sverige/juli-2018-langvarig-hetta-och-svara-skogsbrander-1.137248`.

[10] *T-200B All Weather Precipitation – Rain Gauge.* URL: `https://geonor.com/live/products/weather-instruments/t-200b-weather-precipitation-rain-gauge/`.

[11] Lotte Wilhelmina de Vos et al. "Quality control for crowdsourced personal weather stations to enable operational rainfall monitoring." In: *Geophysical Research Letters* 46.15 (2019), pp. 8820–8829.

APPENDIX A: Main Python code

```python
# -*- coding: utf-8 -*-

"""
Created on Sat Aug 27 21:53:18 2022

@author: ruckl
"""
import os, json
# import pandas as pd
import cartopy.crs as ccrs
#import cartopy.feature as cf
from matplotlib import pyplot as plt
import cartopy.io.shapereader as shpreader
import csv
import re
import copy
# import numpy as np
import time
import datetime
import math

### For execution time check
# Get the start time
st = time.time()

#### PLOT SWEDEN ###
### Downloaded from https://www.naturalearthdata.com/
fname = 'C:/PythonProj/VARIOUS_DATA/Natural_Earth_quick_start/packages/Natural_Earth_quick_start/ne_10m_admin_0_sovereignty/ne_10m_admin_0_sovereignty.shp'
adm1_shapes = list(shpreader.Reader(fname).geometries())
ax = plt.axes(projection=ccrs.PlateCarree())
# plt.title('Sweden')
ax.coastlines(resolution='10m')
ax.add_geometries(adm1_shapes, ccrs.PlateCarree(),
#                   edgecolor='black', facecolor='gray', alpha=0.5)
edgecolor='black', facecolor='gray', alpha=0.5)
ax.set_extent([9, 25, 55, 70], ccrs.PlateCarree())
####################

############## CONFIG LINES ##############
run_locally = True
laptop = True

if run_locally == True:
    local = True
    bi = False
else:
    local = False
    bi = True

if local == True and bi == False:

    if laptop == False:
        data_path_root = 'Q:/exjobb/sverigedata/all_netatmo_data/'
        # Below sets path for SMHI-station data, use the slightly edited version named "TAMPERED_3_monthlyTemperature_Sweden_201510-201911.csv"
        path_and_filename_smhi = "Q:/exjobb/sverigedata/SMHI/TAMPERED_3_monthlyTemperature_Sweden_201510-201911.csv"
        substr = "rain"
        write_path = "Q:/exjobb/sverigedata/Exports/"
        months_to_keep = "Q:/exjobb/sverigedata/SMHI/smhi_relevant_months_clean.csv"
    elif laptop == True:
        data_path_root = 'C:/cygwin64/home/ruckl/sverigedata/Test_Small/'
        # Below sets path for SMHI-station data, use the slightly edited version named "TAMPERED_3_monthlyTemperature_Sweden_201510-201911.csv"
        path_and_filename_smhi = "C:/cygwin64/home/ruckl/sverigedata/SMHI/SMHI_fixed2.csv"
        substr = "rain"
        write_path = "C:/cygwin64/home/ruckl/sverigedata/Exports/"
        months_to_keep = "C:/cygwin64/home/ruckl/sverigedata/SMHI/smhi_relevant_months_clean.csv"

elif local == False and bi == True:
    # data_path_root = 'C:/cygwin64/home/ruckl/sverigedata/Test_Small/'
    # Below sets path for SMHI-station data
    path_and_filename_smhi = "/home/sm_vikis/TAMPERED_3_monthlyTemperature_Sweden_201510-201911.csv"
    substr = "rain"
    data_path_root = '/nobackup/smhid19/users/sm_heiko/NetAtmo/sverigedata/'
    write_path = '/home/sm_vikis/'
############## ############## ##############


class DataPaths:
    ### Call folder_list attribute to get a list of all folders, sub-folders (etc) in a given path
    def __init__(self, data_path_root = data_path_root):
        self.data_path_root = data_path_root
    def excl_dec(self):
        # os.walk(self.data_path_root)
        folder_list = [x[0] for x in os.walk(self.data_path_root, topdown=True)]
        folder_list = folder_list [1:]
        folder_list.sort()
        check_list = [
        "weather-stations-measurements-2014-12-01",
        "weather-stations-measurements-2015-12-01",
        "weather-stations-measurements-2016-12-01",
        "weather-stations-measurements-2017-12-01",
```

```
            "weather-stations-measurements-2018-12-01",
            "weather-stations-measurements-2019-12-01"
            ]
            for i in range(len(check_list)):
                folder_list = [ x for x in folder_list if check_list[i] not in x ]
            return folder_list
    # Run class without excluding specific month as per excl_dec method
    def run(self):
        # os.walk(self.data_path_root)
        folder_list = [x[0] for x in os.walk(self.data_path_root)]
        folder_list = folder_list [1:]
        return folder_list


class StationMetadataJson:
    ### Loads one json file, and makes metadata accessible through class.arguments (is this what its called?)
    def __init__(self, data_path, filename):
        ### data_path & filename parameters should be strings
        data_path = data_path + "/"
        with open(data_path + filename) as file:
            metadataJson = json.load(file)
        self.cipher_id = metadataJson["cipher_id"]
        ### probably possible to use "place"-data and compare with same metadata filename
        ### in other folder to check that it's the same station in other folders
        placeJson = metadataJson["place"]
        self.country = placeJson["country"]
        # self.altitude = placeJson["altitude"]
        self.location = placeJson["location"]
        self.latitude = placeJson["location"][0]
        self.longitude = placeJson["location"][1]


class JsonList:
### Creates list of all json files from given path
    def __init__(self, data_path):
        self.data_path = data_path
    def create_list(self):
        json_files = [pos_json for pos_json in os.listdir(self.data_path) if pos_json.endswith('.json')]
        self.json_files = json_files
        return json_files


class FilteredStations:
### This class takes a path and a part of a filename (e.g. "rain", "pressure", "outdoor" (which includes temp & RH) or
### "wind") and spits out a list of filenames in a list of the .csv type
    def __init__(self, substr, data_path, given_list = []):
        self.data_path = data_path
        self.substr = substr
        self.given_list = given_list
        ### Get list of all csv files from given path, in a list. List elements are strings
        csv_files = [pos_csv for pos_csv in os.listdir(self.data_path) if pos_csv.endswith('.csv') ]
        self.csv_files = csv_files
        # print(csv_files)
    def filterspecstring(self):
    ### This method returns filenames in a list filtered on the given substring
        return [str for str in self.csv_files if
                any(sub in str for sub in [self.substr])]


class MatchFilenameAndID:
    ### Matches json-filename number with all csv's of the same filename number. Leave second parameter blank to work with a
    ### whole folder, or add a list of csv-files to work with (f.ex. using FilteredStations.filterspecstring() ).
    def __init__(self, data_path, csv_files = None):
        self.data_path = data_path
        self.station_dic = {}
        self.csv_files = csv_files
    def string_to_key(self):
    ### Takes the json-list, grabs one json-file at a time and exctracts the "filenamenumber".
    ### The json file's Cipher-ID (which is in the metadata) is then added as "key" in the dictionary,
    ### while the "value" is a list containing: path, json_filename and then all csv's, from
    ### the same folder that has the corresponding "filenamenumber" as the given json.
        print("    Start running: MatchFilenameAndID().sting_to_key()")
        if self.csv_files == None:
            self.csv_files = [pos_csv for pos_csv in os.listdir(self.data_path) if pos_csv.endswith('.csv') ]
        json_list = JsonList(self.data_path).create_list()
        pattern=r'[0-9]+'
    ### Loops through one json-filename at a time, and within this loop, another loop goes through
    ### all csv_files to map all filenames with similar prefix-number in to a dictionary as per above description.
        for index in range(len(json_list)):
            prefix = (re.findall(pattern, json_list[index]))[0] + "."
            json_filename = json_list[index]
            current_values = []
            no_value_list = []
    ### This inside-loop adds csv-filenames that match with the json_filename-value.
            for index2 in range(len(self.csv_files)):
                prefix_csv = (re.findall(pattern, self.csv_files[index2]))[0] + "."
                # Check for filenames with same filename number, but first add data path to the value-list
                if prefix == prefix_csv:
                    location = StationMetadataJson(self.data_path, json_filename).location
```

```python
                    if self.data_path not in current_values:
                        current_values.append(self.data_path )
                    else:
                        pass
                    if json_filename not in current_values:
                        current_values.append(json_filename)
                    else:
                        pass
                    # Adds the location coordinates
                    if location not in current_values:
                        current_values.append(location)
                    # Adds the csv filename and then the json filname both that has the same filenamenumber
                    current_values.append(self.csv_files[index2])
                else:
                    pass
            # Fills up the dictionary
            self.station_dic[StationMetadataJson(self.data_path, json_filename).cipher_id ] = current_values
        ### Removes items in dictionary whose values are just empty lists (if f.ex. having used FilteredStations class to focus on f.ex. "rain").
        print("Clear out empty lists")
        for key, value in self.station_dic.items():
            if value == []:
                no_value_list.append(key)
        for item in range(len(no_value_list)):
            self.station_dic.pop(no_value_list[item])
        print("MatchFilenameAndID().string_to_key() COMPLETE")
        return self.station_dic


class GetAllDics:
    ### This class creates a list of all relevant dictionaries. The dicts. contain matched json- and csv-files
    ### which is with or without a given substr (rain, wind, pressure etc.)
    def __init__(self, data_path_root, substr = None):
        self.substr = substr
        print("Substring/parameter: " + self.substr)
    def run(self):
        self.all_dics_list = []
        datapaths = DataPaths().excl_dec()
        for index in range(len(datapaths)):
            # Check if there's a substring given or not (rain, pressure, wind etc...), then call other classes...
            print("looping: " + str(index) + " time")
            if substr == None:
                self.all_dics_list.append(MatchFilenameAndID(datapaths[index]).string_to_key())
            else:
                filtered = FilteredStations(substr, datapaths[index]).filterspecstring()
                self.all_dics_list.append(MatchFilenameAndID(datapaths[index], filtered).string_to_key())
            print("loop " + str(index) + " complete")
        return self.all_dics_list


class AdjustDics():
    # This class takes a list containing dictionaries, and removes elements based on the given list
    # "invalid_stations_list". That list can f.ex. be created from the class FilteredCheck."SELECTED METHOD"
    def __init__(self, all_dics, invalid_stations_list):
        self.adjusted_dics = copy.deepcopy(all_dics)
        self.all_dics = all_dics
        self.invalid_stations_list = invalid_stations_list
    def run(self):
        for x in range((len(self.adjusted_dics))):
            for i in range(len(self.invalid_stations_list)):
                if self.invalid_stations_list[i] in self.adjusted_dics[x].keys():
                    self.adjusted_dics[x].pop(self.invalid_stations_list[i])
                else:
                    pass
        return self.adjusted_dics


class FilterCheck():
    # Should run class GetAllDics for argument "all_dics".
    def __init__(self, all_dics):
        self.all_dics = all_dics
        current_month_id_list = []
        self.current_month_id_list = current_month_id_list
        next_month_id_list = []
        self.next_month_id_list = next_month_id_list
    def cipher_id(self):
        # Takes all stations from the chronologically first month, compare it with next month in line. Kicks out stations
        # whose cipher-ID isn't in the "next" month. Continues with an updated, possibly smaller list and compares with "next"
        # month, and keeps doing so until comparing with the last month. Returns a list with cipher-ID's who were in all months.
        # Returns two lists of cipher-ID's, index 0 passed- and index 1 failed the check.
        stations_passed = []
        stations_failed = []
        # First loop goes through all the months
        for x in range((len(self.all_dics)) - 1):
            if x < 1:
                # print("first run")
                current_month_list = list(self.all_dics[x].keys())
            else:
```

```
                current_month_list = stations_passed
            stations_passed = []
            next_month_list = list(self.all_dics[x+1].keys())
            for station in range(len(current_month_list)):
                current_month_selected_station_id = current_month_list[station]
                if current_month_selected_station_id in next_month_list:
                    stations_passed.append(current_month_selected_station_id)
                else:
                    if current_month_selected_station_id not in stations_failed:
                        stations_failed.append(current_month_selected_station_id)
        return stations_passed, stations_failed


class RegionSplit():
    # Returns a number of lists filtered on their longitude
    def __init__(self, dic_list):
        self.dic_list = dic_list
        region_1_list = []
        self.region_1_list = region_1_list
        region_2_list = []
        self.region_2_list = region_2_list
        region_3_list = []
        self.region_3_list = region_3_list
        region_1_dict = {}
        self.region_1_dict = region_1_dict
        region_2_dict = {}
        self.region_2_dict = region_2_dict
        region_3_dict = {}
        self.region_3_dict = region_3_dict
    def bands(self):
        for month in range(len(self.dic_list)):
            for key in self.dic_list[month]:
                location = self.dic_list[month][key][2]
                latitude= location[1]
                if latitude > 61:
                    if key not in self.region_1_list:
                        self.region_1_list.append(key)
                elif 61 >= latitude > 58.6:
                    if key not in self.region_2_list:
                        self.region_2_list.append(key)
                else:
                    if key not in self.region_3_list:
                        self.region_3_list.append(key)
        region_1_filtered = AdjustDics(self.dic_list, self.region_2_list + self.region_3_list).run()
        region_2_filtered = AdjustDics(self.dic_list, self.region_1_list + self.region_3_list).run()
        region_3_filtered = AdjustDics(self.dic_list, self.region_1_list + self.region_2_list).run()
        return [region_1_filtered , region_2_filtered , region_3_filtered]


def Wrapper_Id_Band(data_path_root = data_path_root):
    ### Function that CURRENTLY runs cipher-ID check and a region-split (should add extra
     functionality when available)
    substr = "rain"
    all_dics = GetAllDics(data_path_root, substr).run()
    cipher_checked = FilterCheck(all_dics).cipher_id()[1]
    adjusted_dics_cipher = AdjustDics(all_dics, cipher_checked).run()
    [region_filter1 , region_filter2 , region_filter3 ] = RegionSplit(adjusted_dics_cipher).bands()
    return region_filter1 , region_filter2 , region_filter3


class Csv:
    def __init__(self, path, filename):
        self.filename = filename
        self.path = path
        self.path_and_filename = self.path + "/" + self.filename
    def read(self):
        data = []
        for row in csv.reader(open(self.path_and_filename),delimiter=';',skipinitialspace=True):
          data.append(row)
        return data


def read_csv(filename):
    """Reads a CSV file and returns it as a list of rows."""
    data = []
    for row in csv.reader(open(filename),delimiter=';',skipinitialspace=True):
      data.append(row)
    return data


class Monthly():
    def __init__(self, dict_list, cipher_id = None):
        self.dict_list = dict_list
        self.cipher_id = cipher_id
        self.path = []
    def average_plot(self):
        ### Sum all values from a station in a month in a separate list.
        ### This is used for regions, but can be used for whatever that suits.
        months_xaxis = []
        months_xaxis_adjusted = []
        monthly_average_list = []
        for month in range(len(self.dict_list)):
            rain_region_monthly_total = []
            for station in range(len(self.dict_list[month])):
                rain_station_list = []
                rain_station_total = 0
                self.path = list(self.dict_list[month].values())[station][0]
```

```
                if self.path not in months_xaxis:
                    months_xaxis.append(self.path)
                filename = list(self.dict_list[month].values())[station][3]
                rain_data = Csv(self.path, filename).read()
                rain_data = rain_data[1:]
                for i in range(len(rain_data)):
                    rain_station_list.append(rain_data[i][1])
                rain_station_list = [ float(x) for x in rain_station_list ]
                # print(rain_station_list[:10])
                rain_station_total = sum(rain_station_list) #one station total rain
                rain_region_monthly_total.append(rain_station_total)
                rain_region_monthly_total_sum = sum(rain_region_monthly_total)  #all stations summed
 in one month
            divider = len(rain_region_monthly_total)
            rain_region_monthly_total_average = rain_region_monthly_total_sum/divider
            # print(rain_region_monthly_total_average)
            monthly_average_list.append(rain_region_monthly_total_average)
        to_slice = len(self.path) - 11
        months_xaxis = [ x[to_slice:-6] for x in months_xaxis]
        # Adjust months number, as 11 should be 10, 9 --> 8 and so on.
        for i in range(len(months_xaxis)):
            year_and_faulty_month = months_xaxis[i]
            month_value_to_change = int(year_and_faulty_month[-2:])
            month_value_to_change = month_value_to_change - 1
            month_value_changed = str(month_value_to_change)
            if len(month_value_changed ) == 1:
                month_value_changed = "0" + month_value_changed
            year_and_correct_month = year_and_faulty_month[:3] + month_value_changed
            months_xaxis_adjusted.append(year_and_correct_month)
        return months_xaxis_adjusted, monthly_average_list


class Station():
    def __init__(self, dict_list, cipher_id = None):
        self.dict_list = dict_list
        self.cipher_id = cipher_id
        self.path = ""
    def rain_accumulated(self):
        ### Loads csv-data based on a station's cipher-id, sums all precip per month and returns
        ### a list with this monthly precip and a list with the months in question
        rain_station_total = []
        rain_station_accumulated = []
        months_xaxis = []
        months_xaxis_adjusted = []
        for month in range(len(self.dict_list)):
            rain_station_list = []
            filename = self.dict_list[month][self.cipher_id][3]
            self.path = self.dict_list[month][self.cipher_id][0]
            if self.path not in months_xaxis:
                months_xaxis.append(self.path)
            # Load and manage csv-data
            rain_data = Csv(self.path, filename).read()
            rain_data = rain_data[1:]
            for i in range(len(rain_data)):
                rain_station_list.append(rain_data[i][1])
            rain_station_list = [ float(x) for x in rain_station_list ]
            rain_station_total = sum(rain_station_list)
            rain_station_accumulated.append(rain_station_total)
        # Plotting related things below
        to_slice = len(self.path) - 11
        months_xaxis = [ x[to_slice:-6] for x in months_xaxis]
        # Adjust months number, as 11 should be 10, 9 --> 8 and so on.
        for i in range(len(months_xaxis)):
            year_and_faulty_month = months_xaxis[i]
            month_value_to_change = int(year_and_faulty_month[-2:])
            month_value_to_change = month_value_to_change - 1
            month_value_changed = str(month_value_to_change)
            if len(month_value_changed ) == 1:
                month_value_changed = "0" + month_value_changed
            year_and_correct_month = year_and_faulty_month[:3] + month_value_changed
            months_xaxis_adjusted.append(year_and_correct_month)
        return months_xaxis_adjusted, rain_station_accumulated


def plot_ready_data(data_list):
    ### Plots whatever readily baked Netatmo location or region that's wanted (i.e. monthly data here)
    months_xaxis = data_list[0]
    rain_accumulated = data_list[1]
    x_positions = list(range(rain_accumulated))
    x_positions = x_positions[0::5]
    print(x_positions)
    plt.xticks(rain_accumulated, months_xaxis)
    plt.xlabel("Year-Month")
    plt.ylabel("stuff, mm")
    plt.title("One station accum. monthly precip.")
    plt.bar(rain_accumulated, x_positions, width=2, align='center')
    plt.show()


def write_temp(input_list):
    ### Outputs temp/test data as csv.
    temp_path = write_path + "temp/"
    temp_filename = "temp.csv"
    with open(temp_path + temp_filename, 'w') as file:
        writer = csv.writer(file)
        writer.writerow(input_list)
```

```python
def smhi_stations_all(smhi_list_all = read_csv(path_and_filename_smhi)):
    ### Imports all smhi-data and removes the first (header) row
    smhi_list_all = smhi_list_all[1:]
    return smhi_list_all


def smhi_stations_lat():
    ### Filters out SMHI-stations to only have one station/unique latitude (which month we get doesn't
      matter here)
    data_smhi = read_csv(path_and_filename_smhi)
    unique_lat_list = []
    data_smhi = data_smhi[1:]
    for i in range(len(data_smhi)):
        # lat = data_smhi[i][1]
        if len(unique_lat_list) == 0:
            unique_lat_list.append(data_smhi[i])
        else:
            if data_smhi[i-1][1] != unique_lat_list[len(unique_lat_list)-1][1]:
                unique_lat_list.append(data_smhi[i])
    return unique_lat_list


def smhi_stations_name_list(smhi_list_all = smhi_stations_all()):
    ### Uses smhi_stations() to create a list of station names only.
    smhi_names_list = []
    for i in range(len(smhi_list_all)):
        if smhi_list_all[i][4] not in smhi_names_list:
            smhi_names_list.append(smhi_list_all[i][4])
    return smhi_names_list


def smhi_stations_klimatnummer_list(smhi_list_stations = smhi_stations_all()):
    ### Creates a list of one station(element)/klimatnummer only.
    smhi_stations_klimatnummer_list = []
    for i in range(len(smhi_list_stations)):
        if smhi_list_stations[i][3] not in smhi_stations_klimatnummer_list:
            smhi_stations_klimatnummer_list.append(smhi_list_stations[i][3])
    return smhi_stations_klimatnummer_list


def smhi_months_list(smhi_list_stations = smhi_stations_all()):
    # Create list of all months, should appear in numerical order, can this be automatically checked?
    month_list = []
    for i in range(len(smhi_list_stations)):
        if smhi_list_stations[i][6] not in month_list:
            month_list.append(smhi_list_stations[i][6])
        else:
            pass
    return month_list


def smhi_stations_remove_dates(data_smhi = smhi_stations_all()):
    ### Removes elements based on their date, f.ex. December should be removed as that's lacking in
      the Netatmo-data...
    ### Note that the dates in SMHI-data marks the END of a month (the previous one).
    data_smhi_tweaked = [] #copy.deepcopy(data_smhi)
    months_irrelevant_list = ["1/1/2015 6:00","2/1/2015 6:00","3/1/2015 6:00","4/1/2015 6:00","
      5/1/2015 6:00","6/1/2015 6:00","7/1/2015 6:00","8/1/2015 6:00","9/1/2015 6:00","10/1/2015 6:00","
      11/1/2015 6:00","12/1/2015 6:00"]
    for i in range(len(data_smhi)):
        if data_smhi[i][6] not in months_irrelevant_list:
            data_smhi_tweaked.append(data_smhi[i])
        else:
            pass
    return data_smhi_tweaked


def smhi_clean_up_stations(all_data = read_csv(path_and_filename_smhi)):
    ### Removes stations that isn't represented in every month
    passed = []
    failed = []
    station_klimatnummer_list = smhi_stations_klimatnummer_list()
    passed_total = 0
    failed_total = 0
    passed_station_name = []
    failed_station_name = []
    range_of_given_name = 0
    # Pick a station-name that has been checked manually that has data for all months wanted
    for i in range(len(all_data)):
        if all_data[i][4] == "Lund":
            range_of_given_name += 1
    for x in range(len(station_klimatnummer_list)):
        counter = 0
        for i in range(len(all_data)):
            if station_klimatnummer_list[x] == all_data[i][3]:
                counter += 1
            else:
                pass
        if counter == range_of_given_name :
            passed_total +=1
            passed_station_name.append(station_klimatnummer_list[x])
            # print("YES:::" + str(station_klimatnummer_list[x]) + " had " + str(counter) + " counts")
        elif counter != range_of_given_name:
            failed_total +=1
            failed_station_name.append(station_klimatnummer_list[x])
            # print("NO:::" + str(station_klimatnummer_list[x]) + " only had " + str(counter) + "
```

```
    counts")
    for x in range(len(all_data)):
        current_stat = all_data[x][3]
        if current_stat in passed_station_name:
            passed.append(all_data[x])
        elif current_stat in failed_station_name:
            failed.append(all_data[x])
    return passed, failed


def smhi_specific_stations():
    ### Creates lists for specific, manually selected stations
    data_smhi = read_csv(path_and_filename_smhi)
    station_1_smhi = []
    station_2_smhi  = []
    for i in range(len(data_smhi)):
        # print(data_smhi[i][4])
        if data_smhi[i][4] == "Hofors":
            station_1_smhi.append(data_smhi[i])
        elif data_smhi[i][4] == "Gunnarn A":
            station_2_smhi .append(data_smhi[i])
        else:
            pass
    return station_1_smhi, station_2_smhi


def smhi_region_split(data_smhi = smhi_stations_all()):
    ### Splits up original list of data in to three lists based on station latitude.
    data_smhi = data_smhi[1:]
    data_smhi_lat_north = []
    data_smhi_lat_mid = []
    data_smhi_lat_south = []
    for i in range(len(data_smhi)):
        latitude = float(data_smhi[i][1])
        if latitude > 61:
            data_smhi_lat_north.append(data_smhi[i])
        elif 61 >= latitude > 58.6:
            data_smhi_lat_mid.append(data_smhi[i])
        else:
            data_smhi_lat_south.append(data_smhi[i])
    return data_smhi_lat_north, data_smhi_lat_mid, data_smhi_lat_south


def smhi_monthly_average(region_section = read_csv(path_and_filename_smhi)):
    ### Add all obs. values per month, and make average value per region
    month_list = []
    total_data = []
    total_data_averaged = []
    # Create list of all months, should appear in numerical order
    for i in range(len(region_section)):
        if region_section[i][6] not in month_list:
            month_list.append(region_section[i][6])
        else:
            pass
    for i in range(len(month_list)):
        monthly_data = []
        for x in range(len(region_section)):
            if region_section[x][6] == month_list[i]: #and region_section[x] not in monthly_data:
                monthly_data.append(region_section[x])
            else:
                pass
        total_data.append(monthly_data)
    for i in range(len(total_data)):
        monthly_precip_tot = 0
        for x in range(len(total_data[i])):
            # test = int(total_data[i][x][5])
            # print(test)
            monthly_precip_tot += float(total_data[i][x][5])
        monthly_precip_avg = monthly_precip_tot / float(len(total_data[i]))
        total_data_averaged.append(monthly_precip_avg)
    return total_data_averaged


"""
#################### Run SMHI-data: commands here ####################
bad_dates_cleared = smhi_stations_remove_dates()
passed, failed = smhi_clean_up_stations(bad_dates_cleared)
hofors_all_data, gunnarn_a_all_data = smhi_specific_stations()
bad_dates_cleared_hofors = smhi_stations_remove_dates(hofors_all_data)
passed_hofors, failed_hofors = smhi_clean_up_stations(bad_dates_cleared)
bad_dates_cleared_gunnarn_a = smhi_stations_remove_dates(gunnarn_a_all_data)
passed_gunnarn_a, failed_gunnarn_a = smhi_clean_up_stations(bad_dates_cleared)

region_split = smhi_region_split(passed)
north = region_split[0]
mid = region_split[1]
south = region_split[2]

# print(len(smhi_stations_name_list()))
# print(len(smhi_stations_name_list(passed)))
# print(len(smhi_stations_name_list(north)))
# print(len(smhi_stations_name_list(mid)))
# print(len(smhi_stations_name_list(south)))

n_avg = smhi_monthly_average(north)
m_avg = smhi_monthly_average(mid)
s_avg = smhi_monthly_average(south)
```

```
# print("Number of N stations is " + str(len(n_avg)))
# print("Number of N entries is " + str(len(north)))
# print("Number of Mid stations is " + str(len(m_avg)))
# print("Number of Mid entries is " + str(len(mid)))
# print("Number of S stations is " + str(len(s_avg)))
# print("Number of S entries is " + str(len(south)))
"""

"""
######## PLOT REMAINING SMHI STATIONS FOR REGION SPLIT ########
for i in range(len(north)):
    smhi_long = float(north[i][0])
    smhi_lat = float(north[i][1])
    plt.plot(smhi_long, smhi_lat, markersize = 1, color = "red", marker = '.')
for i in range(len(mid)):
    smhi_long = float(mid[i][0])
    smhi_lat = float(mid[i][1])
    plt.plot(smhi_long, smhi_lat, markersize = 1, color = "orange", marker = '.')
for i in range(len(south)):
    smhi_long = float(south[i][0])
    smhi_lat = float(south[i][1])
    plt.plot(smhi_long, smhi_lat, markersize = 1, color = "green", marker = '.')
plt.show()
"""

"""
################ RUN NETATMO-DATA: commands here ###########################################
print("START: 'GetAllDics' to create list of all months, data in each month --> a dict. with station
    ID and ev. parameter selected (rain, RH etc)")
all_dics = GetAllDics(data_path_root, substr).run()
print("FINISHED: 'GetAllDics'")
print("")
print("START: 'FilterCheck().cipher_id' to find stations that isn't present in all months/folders")
cipher_checked = FilterCheck(all_dics).cipher_id()[1]
print("FINISHED: FilterCheck().cipher_id")
print("")
print("START: 'AdjustDics' to remove stations not present in every month")
adjusted_dics_cipher = AdjustDics(all_dics, cipher_checked).run()
print("FINISHED: 'AdjustDics' ")
print("")

vargarda_cipher_id = "enc:16:znrXQ6owWG4Ns2U3aSaVqxAHUelMCjfqVH1F+CPv1kNBPFUiycdYLKqjmi84rpw9"
print("START: 'Station().rain_accumulated' on V rg rda Netatmo to accumulate monthly rain")
vargarda_netatmo_monthly_accumulated_rain = Station(adjusted_dics_cipher, vargarda_cipher_id).
    rain_accumulated()
print("FINISHED: 'Station().rain_accumulated' on V rg rda Netatmo ")
print("")

komperod_cipher_id = "enc:16:Q1BmCn/WNfQtnaYKPQKKDSmBuWT2uNiUPInr0Fp/vNjyrsKfo41mjt5kJJpyGWMY"
print("START: 'Station().rain_accumulated' on Komper d Netatmo to accumulate monthly rain")
komperod_netatmo_monthly_accumulated_rain  = Station(adjusted_dics_cipher, komperod_cipher_id).
    rain_accumulated()
print("FINISHED: 'Station().rain_accumulated' on Komper d Netatmo ")
print("")

hofors_cipher_id = "enc:16:hlma8kYiCfYR+9pD1Vp7Pq4TxHEmT9pqovjUQP1SwkCR27pgRqOzbd2drEzq2imt"
print("START: 'Station().rain_accumulated' on Hofors Netatmo to accumulate monthly rain")
hofors_netatmo_monthly_accumulated_rain  = Station(adjusted_dics_cipher, hofors_cipher_id).
    rain_accumulated()
print("FINISHED: 'Station().rain_accumulated' on Hofors Netatmo ")
print("")

gunnarn_cipher_id = "enc:16:yUdylzG0oXjY5+HqG1E92fyUU03KBEPLhlM2p5XQ8xiNsPrct0NkHuk3/t7HKc4W"
print("START: 'Station().rain_accumulated' on Gunnarn Netatmo to accumulate monthly rain")
gunnarn_netatmo_monthly_accumulated_rain  = Station(adjusted_dics_cipher, gunnarn_cipher_id).
    rain_accumulated()
print("FINISHED: 'Station().rain_accumulated' on Gunnarn Netatmo ")
print("")

print("START: 'Wrapper_Id_Band' to create regions for the data")
region_north, region_mid, region_south = Wrapper_Id_Band()
print("FINISHED: 'Wrapper_Id_Band' ")
print("")

print("START: 'Monthly().average_plot', which on a BI-run returns monthly average rain from the (north
    ) region, doesn't make a plot")
region_north_average = Monthly(region_north).average_plot()
print("FINISHED: 'Monthly().average_plot' (north)")
print("")

print("START: 'Monthly().average_plot', which on a BI-run returns monthly average rain from the (mid)
    region, doesn't make a plot")
region_mid_average = Monthly(region_mid).average_plot()
print("FINISHED: 'Monthly().average_plot' (mid)")
print("")

print("START: 'Monthly().average_plot', which on a BI-run returns monthly average rain from the (south
    ) region, doesn't make a plot")
region_south_average = Monthly(region_south).average_plot()
print("FINISHED: 'Monthly().average_plot' (south)")
print("")

print("START: write data from north region to csv-file")
with open(write_path + "north.csv", 'w') as file:
    writer = csv.writer(file)
    writer.writerow(region_north_average)
print("FINISHED: write data from north region to csv-file")
```

```
print("")

print("START: write data from mid region to csv-file")
with open(write_path + "mid.csv", 'w') as file:
    writer = csv.writer(file)
    writer.writerow(region_mid_average)
print("FINISHED: write data from mid region to csv-file")
print("")

print("START: write data from south region to csv-file")
with open(write_path + "south.csv", 'w') as file:
    writer = csv.writer(file)
    writer.writerow(region_south_average)
print("FINISHED: write data from south region to csv-file")
print("")

print("START: write data from V rg rda D Netatmo station to csv-file")
with open(write_path + "vargarda.csv", 'w') as file:
    writer = csv.writer(file)
    writer.writerow(vargarda_netatmo_monthly_accumulated_rain)
print("FINISHED: write data from V rg rda D Netatmo station to csv-file")
print("")

print("START: write data from Komper d Netatmo station to csv-file")
with open(write_path + "komperod.csv", 'w') as file:
    writer = csv.writer(file)
    writer.writerow(komperod_netatmo_monthly_accumulated_rain)
print("FINISHED: write data from Komper d Netatmo station to csv-file")
print("")

print("START: write data from Hofors Netatmo station to csv-file")
with open(write_path + "hofors.csv", 'w') as file:
    writer = csv.writer(file)
    writer.writerow(hofors_netatmo_monthly_accumulated_rain)
print("FINISHED: write data from Hofors Netatmo station to csv-file")
print("")

print("START: write data from Gunnarn Netatmo station to csv-file")
with open(write_path + "gunnarn.csv", 'w') as file:
    writer = csv.writer(file)
    writer.writerow(gunnarn_netatmo_monthly_accumulated_rain)
print("FINISHED: write data from Gunnarn Netatmo station to csv-file")
print("")

"""

##################### BELOW IS USED TO FIND STATIONS THAT ARE CLOSE TO ONEANOTHER
##################### I.E. NETATMO & SMHI STATIONS
def smhi_stations():
    ### Filters out SMHI-stations to only have one station/unique latitude (which month we get doesn't
      matter here)
    # path_and_filename_smhi = "C:/cygwin64/home/ruckl/sverigedata/SMHI/
     TAMPERED_3_monthlyTemperature_Sweden_201510-201911.csv"
    data_smhi = read_csv(path_and_filename_smhi)
    unique_lat_list = []
    data_smhi = data_smhi[1:]
    for i in range(len(data_smhi)):
        # lat = data_smhi[i][1]
        if len(unique_lat_list) == 0:
            unique_lat_list.append(data_smhi[i])
        else:
            if data_smhi[i-1][1] != unique_lat_list[len(unique_lat_list)-1][1]:
                unique_lat_list.append(data_smhi[i])
    return unique_lat_list


def check_latitude_distance(dict_list, unique_lat_list):
    ### Checks if distance between relevant Netatmo and SMHI-stations is shorter than max_diff_lat &
     max_diff_long
    ### Adjust max_diff_lat to set max distance.
    first_month = dict_list[0]
    max_diff_lat = 1/111 # - Each degree of latitude is approx. 111 km apart.
    # max_diff_lat = 1/222 #USE THIS VALUE AS TEMP ONLY, SHOULD BE ABOUT MAX 500m
    max_diff_long = max_diff_lat*20
    close_stations = []
    for x in range(len(unique_lat_list)):
        smhi_lat = float(unique_lat_list[x][1])
        smhi_long = float(unique_lat_list[x][0])

        for station in range(len(first_month)):
            netatmo_values = list(first_month.values())[station]
            netatmo_key = [list(first_month.keys())[list(first_month.values()).index(netatmo_values)]]
            netatmo_key_and_values = netatmo_key + netatmo_values
            netatmo_position = list(first_month.values())[station][2]
            netatmo_position_long = float(netatmo_position[0])
            netatmo_position_lat = float(netatmo_position[1])

            if abs(smhi_lat - netatmo_position_lat) < max_diff_lat and abs(smhi_long -
     netatmo_position_long) < max_diff_long:
                close_stations.append([unique_lat_list[x], netatmo_key_and_values, ])
                aa = abs(smhi_lat - netatmo_position_lat)*111
                bb = abs(smhi_long - netatmo_position_long)*111
                cc = math.sqrt(aa*aa + bb*bb)
                print("latitude (north - south) distance is " + str(aa))
                print("Longitude (east - west) distance is " + str(bb))
                print("Actual distance is " + str(cc))
                print("coordinates for smhi station is " + str(smhi_lat) + " latitude and " + str(
```

```
     smhi_long))
                print("")
    return close_stations


def wrapper_find_close_stations():
    ### Returns list of stations that are close to one another
    unique_lat_list = smhi_stations()
    all_dics = GetAllDics(data_path_root, substr).run()
    cipher_checked = FilterCheck(all_dics).cipher_id()[1]
    dict_list = AdjustDics(all_dics, cipher_checked).run()
    result = check_latitude_distance(dict_list, unique_lat_list)
    return result


def plot_close_stations(result_list, color = "red", marker = '.'):
    ### Plots SMHI- and Netatmo-stations, that from function "wrapper_find_close_stations" will be
     close to one another.
    for i in range(len(result_list)):
        smhi_long = float(result_list[i][0][0])
        smhi_lat = float(result_list[i][0][1])
        plt.plot(smhi_long, smhi_lat, markersize = 3, marker = marker, color = "red")
        netatmo_long = float(result_list[i][1][3][0])
        netatmo_lat = float(result_list[i][1][3][1])
        plt.plot(netatmo_long, netatmo_lat, markersize = 3, marker = marker, color = "blue")


######## PLOT NEARBY STATIONS ########
### Used to plot nearby stations
result_list = wrapper_find_close_stations()
plot_close_stations(result_list)
plt.show()
######## ######## ######## ######## ########



### Execution time check ###
# Get the end time
et = time.time()
# Get the execution time
elapsed_time = et - st
total_time = str(datetime.timedelta(seconds=elapsed_time))
print()
print('Execution time:', elapsed_time, 'seconds')
print()
print("Total time is: " + total_time + " h:m:s")
```

APPENDIX B: Python code to plot figures

```python
# -*- coding: utf-8 -*-
"""
Created on Tue Oct  4 14:28:25 2022

@author: ruckl
"""
from matplotlib import pyplot as plt
import numpy as np
from matplotlib.ticker import FuncFormatter
from matplotlib.dates import MonthLocator, DateFormatter

x_ax = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26,
    27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48]
months_short = ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec']
months_short_48 = months_short + months_short + months_short + months_short
years = ["2016", "2017", "2018", "2019", "2020"]
fontisize = 6
x_ax_rot = 55
scatter_dot_size = 8
linewidth = 0.8
legend_fontsize = "small"
### WARNING - In Spyder, if collapsing a list, the list must be expanded before deleted, otherwise
    code will magically linger and cause bugs...
netatmo_south = [24.7473098591549,
34.7178366,
24.64316564,
41.59566667,
19.01063934,
50.42926699,
67.96879583,
52.73585606,
20.76099636,
64.75794464,
57.86901993,
None,
21.92859531,
35.17188136,
38.16503784,
37.19439241,
20.01420099,
89.61607323,
65.00179691,
83.30187738,
87.49653144,
102.8753849,
73.68707942,
None,
68.64232053,
21.53185624,
34.73335276,
37.59308866,
16.64257143,
20.76139276,
14.61530064,
93.74489324,
39.88516724,
66.91649055,
25.05786857,
None,
40.70485266,
48.87048765,
82.97346351,
16.23459506,
49.20471094,
50.89070311,
54.63052965,
78.27563403,
85.33269757,
97.54326736,
83.29070483,
None]
smhi_south = [45.77770115,
48.12821839,
34.8,
48.59827586,
26.15287356,
54.95229885,
65.85747126,
72.23735632,
22.9316092,
80.42873563,
67.89367816,
36.85977011,
25.03218391,
42.13448276,
45.84712644,
39.66436782,
22.35574713,
98.75862069,
47.8091954,
99.87643678,
96.52068966,
105.6327586,
74.43396552,
```

```
90.34011494,
75.34517241,
38.94086207,
35.81752874,
37.12586207,
13.45109195,
30.06988506,
19.65936782,
101.4186782,
47.16678161,
62.25735632,
28.66264368,
55.46655172,
43.80425287,
61.93758621,
84.43706897,
14.09155172,
58.90534483,
47.12362069,
62.12229885,
79.17465517,
81.02373563,
86.0958046,
72.26649425,
#73.80390805
None]
smhi_south_manual = [45.23903846,
48.00052885,
34.01875,
48.72355769,
25.55961538,
54.90625,
65.20682927,
71.91504854,
22.44139535,
81.0682243,
68.08110599,
36.83928571,
25.18235294,
42.30776256,
45.29272727,
39.21780822,
22.73243243,
97.09276018,
46.4963964,
100.1333333,
97.76909091,
105.1922727,
74.53013699,
90.74541284,
75.78940092,
40.19087156,
35.3659633,
37.19642202,
13.37627273,
30.91522727,
20.05319635,
101.143379,
47.63287671,
63.10046296,
28.37170507,
55.66559633,
44.06609091,
62.49409091,
84.78783784,
14.00189189,
58.81846847,
46.10495495,
62.08288288,
78.99181818,
81.24331797,
86.32018349,
72.97853881,
#75.14409091
None]
netatmo_mid = [16.20849057,
17.78693798,
14.61463522,
38.35888108,
29.66274884,
51.0123834,
42.12133798,
62.14179808,
19.09813814,
25.99061625,
48.42578437,
None,
10.39389855,
14.91921546,
20.21533405,
20.4749521,
15.60871857,
52.33754878,
28.11288372,
72.36522153,
64.31857313,
89.90553353,
```

```
48.5121136,
None,
33.50373533,
10.69370281,
10.68573664,
24.80440658,
14.93071508,
39.23020619,
27.14774088,
62.23373726,
58.61831787,
41.13452596,
32.1542927,
None,
13.85140084,
26.19098052,
41.30397054,
7.366575714,
52.14489682,
43.9717135,
59.02975522,
77.23375948,
61.13088524,
72.94590094,
59.27209783,
None]
smhi_mid = [41.12851351,
35.83824324,
30.61486486,
56.95743243,
51.33310811,
43.33310811,
58.36351351,
82.29864865,
19.89797297,
37.45945946,
63.95878378,
24.30945946,
25.78648649,
33.23851351,
36.61351351,
30.40945946,
23.23581081,
59.37297297,
32.84256757,
96.80472973,
62.94662162,
98.90743243,
74.33189189,
65.35858108,
68.2447973,
27.86006757,
22.49466216,
41.19635135,
16.62783784,
46.72459459,
36.02439189,
68.10094595,
57.58189189,
39.77547297,
32.15554054,
50.84114865,
38.98141892,
44.85743243,
67.21358108,
6.375945946,
67.91412162,
51.02966216,
62.03533784,
83.61081081,
73.79074324,
92.82310811,
84.51554054,
None]
netatmo_north = [11.1222381,
10.64328571,
10.87396,
41.70857895,
37.16870213,
46.55101754,
75.94420588,
66.89363014,
33.21753247,
10.57455556,
25.08544156,
None,
12.16471951,
8.433853659,
19.51350588,
17.75957447,
23.73440708,
70.87774167,
57.94435338,
73.69615603,
53.88909211,
73.97394737,
28.35463816,
```

```
None,
9.779292208,
2.491651899,
8.02289375,
15.50044385,
17.712125,
38.95240741,
43.67611407,
56.55083498,
41.99772222,
37.10217377,
18.26309524,
None,
7.086531136,
21.60326007,
20.25446875,
7.577176292,
68.67404651,
57.25479858,
35.5159207,
65.98015319,
67.97058836,
49.71203272,
36.57605423,
None]
smhi_north = [34.68577093,
42.64436123,
23.51497797,
59.04669604,
40.65330396,
57.0938326,
86.17136564,
91.9246696,
45.24008811,
12.8876652,
68.72951542,
35.56211454,
35.56167401,
35.06035242,
30.00885463,
28.37577093,
29.00748899,
54.71497797,
83.96651982,
80.62819383,
54.12378855,
76.96519824,
71.21647577,
72.99070485,
56.9369163,
23.95898678,
30.33048458,
29.47863436,
22.99118943,
42.34537445,
49.56295154,
71.95577093,
50.89898678,
43.71757709,
24.53744493,
68.60484581,
35.63784141,
40.32832599,
59.51660793,
11.04700441,
80.56665198,
69.07193833,
35.52656388,
69.71268722,
82.74339207,
55.98299559,
51.80986784,
None]
netatmo_gunnarn = [1.56,
10.918,
15.895,
59.697,
26.642,
48.294,
35.661,
86.567,
22.742,
5.458,
9.2,
None,
4.056,
5.302,
12.634,
19.02,
15.755,
53.152,
72.244,
55.145,
32.897,
52.558,
5.459,
None,
```

```
0,
0,
21.828,
17.001,
21.984,
32.883,
33.777,
99.833,
26.808,
8.263,
20.279,
None,
0.624,
8.266,
9.982,
2.34,
90.72,
31.17,
29.303,
56.24,
45.218,
37.424,
8.735,
None]
smhi_gunnarn = [22.3,
34.5,
17.4,
52.9,
22.3,
66.6,
42.6,
95.4,
20.5,
3.8,
49,
17,
17.5,
40,
13.5,
20,
17.9,
51.1,
85.2,
55.9,
33.1,
59,
61.68,
52.56,
47.06,
12.67,
40.64,
25.82,
23.43,
32.34,
37.41,
79.5,
25,
36.87,
22.58,
72.79,
24.48,
30.87,
37.37,
0.29,
96.28,
31.08,
24.16,
43.73,
47.67,
48.03,
41.12,
# 53.76
None]
netatmo_hofors = [66.421,
29.69,
35.427,
66.813,
85.344,
40.87,
100.283,
145.015,
24.133,
32.995,
75.961,
None,
9.087,
21.239,
25.187,
31.578,
36.172,
62.575,
57.715,
75.878,
47.007,
106.955,
62.114,
None,
```

```
26.861,
0.638,
0,
34.619,
20.963,
47.769,
24.241,
37.271,
61.576,
34.246,
12.286,
None,
0,
0,
16.847,
11.371,
78.427,
71.214,
109.739,
62.435,
98.953,
124.931,
60.478,
None]
smhi_hofors = [61.3,
33.8,
32.6,
53.1,
70.7,
29.9,
80.7,
118.4,
18.5,
38.9,
77.8,
27.2,
26.9,
27.9,
30,
26.8,
27.3,
53.8,
50.9,
60.6,
41,
93.4,
88.5,
64.3,
82.9,
49,
27.1,
47.3,
17.9,
40.6,
103.9,
89.5,
49.5,
36.3,
14,
59.1,
38.5,
55.2,
84.3,
11.4,
62.5,
65,
87.6,
60.3,
86.3,
123.6,
88.9,
# 77.2
None]
netatmo_komperod = [50.904,
61.913,
56.257,
73.427,
13.332,
55.146,
89.688,
105.646,
60.701,
60.903,
79.487,
None,
29.795,
52.217,
52.015,
48.278,
24.442,
58.681,
22.422,
66.559,
42.117,
69.488,
52.924,
None,
```

```
52.823,
25.654,
14.14,
31.714,
10.403,
42.723,
31.108,
55.752,
189.375,
64.337,
42.016,
None,
21.513,
57.873,
95.849,
22.119,
59.287,
64.337,
33.633,
94.839,
117.867,
99.586,
63.731,
None]
smhi_komperod = [96.6,
68.1,
77.5,
71.4,
18.9,
66.6,
129.8,
101.2,
69.3,
73.5,
109.4,
67.1,
62,
82,
85.4,
60.8,
29.4,
103.8,
42.6,
123.3,
116,
160.6,
116.2,
163.3,
136.4,
56.8,
41.1,
62.3,
25.8,
47,
22,
95.9,
136.7,
116.3,
45,
57.5,
41.1,
98,
158.9,
21,
59.3,
64.4,
26.5,
118.3,
184.1,
112.2,
97,
# 157.9
None]
netatmo_vargarda = [46.258,
51.106,
40.299,
51.409,
17.675,
84.133,
70.397,
102.818,
55.449,
36.057,
70.7,
None,
19.19,
30.098,
53.227,
28.987,
23.028,
66.963,
22.523,
116.453,
59.893,
96.455,
22.725,
None,
```

```
0.303,
11.009,
18.685,
41.41,
12.524,
16.16,
13.029,
24.644,
49.187,
64.438,
17.473,
None,
18.786,
52.116,
36.36,
17.372,
41.713,
23.937,
23.836,
70.397,
19.493,
77.871,
40.602,
None]
smhi_vargarda = [59.7,
73.5,
44.2,
62.2,
21,
125.1,
71.5,
113.8,
46.5,
49,
75.7,
56.3,
24.9,
34.8,
57.5,
31.9,
26,
102,
19.1,
128.4,
85.5,
97.6,
81,
87.3,
85.7,
54.1,
33.6,
57.2,
10.7,
55.4,
21.2,
79.4,
98.8,
75,
21.5,
46.1,
42.3,
83.8,
78.7,
26.9,
59.8,
33.8,
47.4,
97.4,
92.1,
90.3,
60.3,
# 100.1
None]

standard_dev_south_test = np.array([12.936676,
10.25116683,
5.726223549,
0.334773536,
3.914394415,
2.015376241,
1.771966898,
6.055490168,
4.769816731,
14.34147902,
3.739460537,
0,
11.72099179,
5.886354996,
2.992328499,
1.201897498,
11.34988359,
1.925091601,
1.835363941,
7.399949778,
6.393657579,
1.945946845,
4.365973911,
```

```
0,
5.878247135,
11.20853723,
6.262228684,
2.180618027,
3.304516645,
0.516997653,
1.662049169,
6.226354574,
8.381987341,
0.633030074,
0.06984595,
0,
7.690878964,
6.125650348,
1.852785006,
2.324191463,
2.284695364,
1.554025782,
3.424217766,
4.124972024,
7.614589839,
4.767677655,
8.818545286,
0])

standard_dev_south = [12.936676,
10.25116683,
5.726223549,
0.334773536,
3.914394415,
2.015376241,
1.771966898,
6.055490168,
4.769816731,
14.34147902,
3.739460537,
None,
11.72099179,
5.886354996,
2.992328499,
1.201897498,
11.34988359,
1.925091601,
1.835363941,
7.399949778,
6.393657579,
1.945946845,
4.365973911,
None,
5.878247135,
11.20853723,
6.262228684,
2.180618027,
3.304516645,
0.516997653,
1.662049169,
6.226354574,
8.381987341,
0.633030074,
0.06984595,
None,
7.690878964,
6.125650348,
1.852785006,
2.324191463,
2.284695364,
1.554025782,
3.424217766,
4.124972024,
7.614589839,
4.767677655,
8.818545286,
None]

standard_dev_mid = [17.62111721,
12.76420036,
11.31387088,
13.15116178,
15.32325799,
5.430067633,
11.48495246,
14.25304573,
0.565568636,
8.109696808,
10.98348922,
None,
10.88420331,
12.95369988,
11.5952639,
7.024757525,
5.393168641,
4.974796155,
3.344391521,
17.28134198,
0.970116218,
6.365303759,
```

```
18.25734032,
None,
24.5656405,
12.13845293,
8.350171314,
11.5908553,
1.200047012,
5.299332865,
6.276740123,
4.14874305,
0.732863837,
0.960995585,
0.000882354,
None,
17.76960619,
13.19917472,
18.32086131,
0.700481027,
11.15052579,
4.990723362,
2.12526785,
4.50925624,
8.951871445,
14.05530798,
17.84980952,
None]

standard_dev_north =  [16.66193385,
22.6281775,
8.93854953,
12.25990017,
2.463985589,
7.454896018,
7.231694016,
17.69961775,
8.501230619,
1.635615514,
30.86102059,
None,
16.54414518,
18.82777784,
7.421332267,
7.506784505,
3.728631974,
11.42879981,
18.40045035,
4.901690939,
0.165955445,
2.115133774,
30.30789603,
None,
33.34547578,
15.17969807,
15.77384875,
9.884073299,
3.732862255,
2.399190004,
4.162622697,
10.89293467,
6.294144533,
4.677796549,
4.436635217,
None,
20.18882511,
13.24062109,
27.76252486,
2.453538989,
8.409341974,
8.35597965,
0.007525859,
2.639300126,
10.44594968,
4.434240373,
10.77193291,
None]

standard_dev_gunnarn = [14.66539464,
16.67499211,
1.064195706,
4.806204792,
3.070257644,
12.94429674,
4.906613955,
6.245874198,
1.585333403,
1.172383043,
28.14284989,
None,
9.506343566,
24.53519109,
0.612354473,
0.692964646,
1.516744046,
1.450983115,
9.161275457,
0.53386562,
0.143542677,
```

```
4.555181884,
39.75425035,
None,
33.27644512,
8.959042918,
13.30209277,
6.235974703,
1.022476406,
0.383958982,
2.568918936,
14.37760218,
1.27844906,
20.22820369,
1.627052704,
None,
16.86873937,
15.98344168,
19.36624052,
1.449568901,
3.931513703,
0.06363961,
3.636650176,
8.845905833,
1.733825827,
7.499574521,
22.89965311,
None]

standard_dev_hofors = [3.621093826,
2.906208871,
1.99899087,
9.69655529,
10.3548717,
7.75696139,
13.8472721,
18.81964698,
3.983132498,
4.175465543,
1.300369371,
None,
12.59569309,
4.710038269,
3.403304938,
3.378556201,
6.273451363,
6.204862005,
4.818932714,
10.8031774,
4.247590435,
9.584832419,
18.65771953,
None,
39.62555691,
34.19709815,
19.16259377,
8.966821092,
2.165868071,
5.069248514,
56.32741908,
36.93148007,
8.53902149,
1.452397329,
1.211981023,
None,
27.22361108,
39.03229432,
47.69647371,
0.020506097,
11.2620897,
4.393961538,
15.65463703,
1.509672978,
8.947022102,
0.941159126,
20.09738893,
None]

standard_dev_komperod = [32.31195147,
4.374869655,
15.02106935,
1.433305445,
3.937170558,
8.099201072,
28.36346721,
3.143796749,
6.080411211,
8.907424123,
21.15168515,
None,
22.77237389,
21.05976126,
23.60675989,
8.854391114,
3.505835421,
31.90395086,
14.26800063,
40.12194587,
```

```
52.24317031,
64.42591305,
44.74288869,
None,
59.09786345,
22.02354781,
19.06359882,
21.62756801,
10.88732311,
3.024295703,
6.440328563,
28.38892305,
37.2468497,
36.74338967,
2.110006635,
None,
13.85010052,
28.37407381,
44.58378966,
0.791252488,
0.009192388,
0.044547727,
5.04379267,
16.58943219,
46.83380344,
8.919444938,
23.5247355,
None]

standard_dev_vargarda = [9.504929353,
15.83494926,
2.758423553,
7.630389276,
2.351130047,
28.9680435,
0.77993878,
7.765446671,
6.327898585,
9.152083069,
3.535533906,
None,
4.037579721,
3.324816085,
3.021467276,
2.059802054,
2.101521354,
24.77490029,
2.420426512,
8.447804715,
18.10688335,
0.809637264,
41.20664767,
None,
60.38479779,
30.46993831,
10.54649764,
11.16521607,
1.289762769,
27.74687009,
5.777769509,
38.71833891,
35.08168874,
7.468461823,
2.847519008,
None,
16.62690885,
22.40397126,
29.93890112,
6.737313411,
12.78944035,
6.974194183,
16.66226419,
19.09400441,
51.34090206,
8.788630183,
13.92858938,
None]

#####################
##################### PLOTTING COMMANDS BELOW. UNCOMMENT WANTED PLOT-TYPE #####################
#####################

# ### PLOT NETATMO VS SMHI SOUTH REGION
# fig, ax1 = plt.subplots()
# plt.scatter(x_ax, netatmo_south, s = scatter_dot_size, color = "blue")
# plt.scatter(x_ax, smhi_south, s = scatter_dot_size, color = "red")
# plt.plot(x_ax, netatmo_south, linewidth = linewidth, color = "blue", label = "Netatmo")
# plt.plot(x_ax, smhi_south, linewidth = linewidth, color = "red", label = "SMHI")
# plt.xlabel('Month')
# plt.ylabel('Precipitation (mm)')
# # plt.title('Netatmo & SMHI monthly average, south region \n')
# plt.legend(loc="upper left", fontsize = legend_fontsize)
# plt.xticks(x_ax, months_short_48, rotation = x_ax_rot, rotation_mode="anchor",fontsize = fontisize)
# ax2 = ax1.twiny()
# ax2.plot(years, [40, 40, 40, 40, 40], color='blue', linewidth = 0)
# fig.tight_layout()
# plt.xlabel('Year')
```

```
# ### PLOT SMHI VS SMHI MANUAL CACLULATIONS SOUTH REGION
# fig, ax1 = plt.subplots()
# plt.scatter(x_ax, smhi_south_manual, s = scatter_dot_size, color = "blue")
# plt.scatter(x_ax, smhi_south, s = scatter_dot_size, color = "red")
# plt.plot(x_ax, smhi_south_manual, linewidth = linewidth, color = "blue")
# plt.plot(x_ax, smhi_south, linewidth = linewidth, color = "red")
# # plt.title('SMHI & SMHI manual calculations, monthly average, south region')
# plt.xlabel('Month')
# plt.ylabel('Precipitation (mm)')
# plt.legend(loc="upper left")
# plt.xticks(x_ax, months_short_48, rotation = x_ax_rot, rotation_mode="anchor",fontsize = fontisize)
# ax2 = ax1.twiny()
# ax2.plot(years, [40, 40, 40, 40, 40], color='blue', linewidth = 0)
# fig.tight_layout()
# plt.xlabel('Year')

# ### PLOT NETATMO VS SMHI MID REGION
# fig, ax1 = plt.subplots()
# plt.scatter(x_ax, netatmo_mid, s = scatter_dot_size, color = "blue")
# plt.scatter(x_ax, smhi_mid, s = scatter_dot_size, color = "red")
# plt.plot(x_ax, netatmo_mid, linewidth = linewidth, color = "blue", label = "Netatmo")
# plt.plot(x_ax, smhi_mid, linewidth = linewidth, color = "red", label = "SMHI")
# plt.xlabel('Month')
# plt.ylabel('Precipitation (mm)')
# # plt.title('Netatmo & SMHI monthly average, mid region \n')
# plt.legend(loc="upper left", fontsize = legend_fontsize)
# plt.xticks(x_ax, months_short_48, rotation = x_ax_rot, rotation_mode="anchor",fontsize = fontisize)
# ax2 = ax1.twiny()
# ax2.plot(years, [40, 40, 40, 40, 40], color='blue', linewidth = 0)
# fig.tight_layout()
# plt.xlabel('Year')

# ### PLOT NETATMO VS SMHI NORTH REGION
# fig, ax1 = plt.subplots()
# plt.scatter(x_ax, netatmo_north, s = scatter_dot_size, color = "blue")
# plt.scatter(x_ax, smhi_north, s = scatter_dot_size, color = "red")
# plt.plot(x_ax, netatmo_north, linewidth = linewidth, color = "blue", label = "Netatmo")
# plt.plot(x_ax, smhi_north, linewidth = linewidth, color = "red", label = "SMHI")
# plt.xlabel('Month')
# plt.ylabel('Precipitation (mm)')
# # plt.title('Netatmo & SMHI monthly average, north region \n')
# plt.legend(loc="upper left", fontsize = legend_fontsize)
# plt.xticks(x_ax, months_short_48, rotation = x_ax_rot, rotation_mode="anchor",fontsize = fontisize)
# ax2 = ax1.twiny()
# ax2.plot(years, [40, 40, 40, 40, 40], color='blue', linewidth = 0)
# fig.tight_layout()
# plt.xlabel('Year')

# ### PLOT NETATMO VS SMHI GUNNARN A
# fig, ax1 = plt.subplots()
# plt.scatter(x_ax, netatmo_gunnarn, s = scatter_dot_size, color = "blue")
# plt.scatter(x_ax, smhi_gunnarn, s = scatter_dot_size, color = "red")
# plt.plot(x_ax, netatmo_gunnarn, linewidth = linewidth, color = "blue", label = "Netatmo")
# plt.plot(x_ax, smhi_gunnarn, linewidth = linewidth, color = "red", label = "SMHI")
# plt.xlabel('Month')
# plt.ylabel('Precipitation (mm)')
# # plt.title('Netatmo & SMHI monthly average, Gunnarn A\n')
# plt.legend(loc="upper left", fontsize = "x-small")
# plt.xticks(x_ax, months_short_48, rotation = x_ax_rot, rotation_mode="anchor",fontsize = fontisize)
# ax2 = ax1.twiny()
# ax2.plot(years, [40, 40, 40, 40, 40], color='blue', linewidth = 0)
# fig.tight_layout()
# plt.xlabel('Year')

# ### PLOT NETATMO VS SMHI HOFORS
# fig, ax1 = plt.subplots()
# plt.scatter(x_ax, netatmo_hofors, s = scatter_dot_size, color = "blue")
# plt.scatter(x_ax, smhi_hofors, s = scatter_dot_size, color = "red")
# plt.plot(x_ax, netatmo_hofors, linewidth = linewidth, color = "blue", label = "Netatmo")
# plt.plot(x_ax, smhi_hofors, linewidth = linewidth, color = "red", label = "SMHI")
# plt.xlabel('Month')
# plt.ylabel('Precipitation (mm)')
# # plt.title('Netatmo & SMHI monthly average, Hofors A\n')
# plt.legend(loc="upper left", fontsize = "x-small")
# plt.xticks(x_ax, months_short_48, rotation = x_ax_rot, rotation_mode="anchor",fontsize = fontisize)
# ax2 = ax1.twiny()
# ax2.plot(years, [40, 40, 40, 40, 40], color='blue', linewidth = 0)
# fig.tight_layout()
# plt.xlabel('Year')

# ## PLOT NETATMO VS SMHI KOMPERD
# fig, ax1 = plt.subplots()
# plt.scatter(x_ax, netatmo_komperod, s = scatter_dot_size, color = "blue")
# plt.scatter(x_ax, smhi_komperod, s = scatter_dot_size, color = "red")
# plt.plot(x_ax, netatmo_komperod, linewidth = linewidth, color = "blue", label = "Netatmo")
# plt.plot(x_ax, smhi_komperod, linewidth = linewidth, color = "red", label = "SMHI")
# plt.xlabel('Month')
# plt.ylabel('Precipitation (mm)')
# # plt.title('Netatmo & SMHI monthly average, Komper d A\n')
# plt.legend(loc="upper left", fontsize = "x-small")
# plt.xticks(x_ax, months_short_48, rotation = x_ax_rot, rotation_mode="anchor",fontsize = fontisize)
# ax2 = ax1.twiny()
# ax2.plot(years, [40, 40, 40, 40, 40], color='blue', linewidth = 0)
# fig.tight_layout()
# plt.xlabel('Year')
```

```
# ### PLOT NETATMO VS SMHI V RG RDA
# fig, ax1 = plt.subplots()
# plt.scatter(x_ax, netatmo_vargarda, s = scatter_dot_size, color = "blue")
# plt.scatter(x_ax, smhi_vargarda, s = scatter_dot_size, color = "red")
# plt.plot(x_ax, netatmo_vargarda, linewidth = linewidth, color = "blue", label = "Netatmo")
# plt.plot(x_ax, smhi_vargarda, linewidth = linewidth, color = "red", label = "SMHI")
# plt.xlabel('Month')
# plt.ylabel('Precipitation (mm)')
# # plt.title('Netatmo & SMHI monthly average, V rg rda D\n')
# plt.legend(loc="upper left", fontsize = "x-small")
# plt.xticks(x_ax, months_short_48, rotation = x_ax_rot, rotation_mode="anchor",fontsize = fontisize)
# ax2 = ax1.twiny()
# ax2.plot(years, [40, 40, 40, 40, 40], color='blue', linewidth = 0)
# fig.tight_layout()
# plt.xlabel('Year')

### PLOT NETATMO VS SMHI SINGLE STATIONS STANDARD DEVIATION
# fig, ax1 = plt.subplots()
# plt.scatter(x_ax, standard_dev_gunnarn, s = scatter_dot_size, color = "blue")
# plt.scatter(x_ax, standard_dev_hofors, s = scatter_dot_size, color = "red")
# plt.scatter(x_ax, standard_dev_komperod, s = scatter_dot_size, color = "green")
# plt.scatter(x_ax, standard_dev_vargarda, s = scatter_dot_size, color = "purple")

# plt.plot(x_ax, standard_dev_gunnarn, linewidth = linewidth, color = "blue", label = "Gunnarn A")
# plt.plot(x_ax, standard_dev_hofors, linewidth = linewidth, color = "red", label = "Hofors")
# plt.plot(x_ax, standard_dev_komperod, linewidth = linewidth, color = "green", label = "Komper d")
# plt.plot(x_ax, standard_dev_vargarda, linewidth = linewidth, color = "purple", label = "V rg rda D")

# plt.xlabel('Month')
# plt.ylabel('Precipitation (mm)')
# # plt.title('Netatmo & SMHI monthly average, V rg rda D\n')
# plt.legend(loc="upper left", fontsize = "x-small")
# plt.xticks(x_ax, months_short_48, rotation = x_ax_rot, rotation_mode="anchor",fontsize = fontisize)
# ax2 = ax1.twiny()
# ax2.plot(years, [40, 40, 40, 40, 40], color='blue', linewidth = 0)
# fig.tight_layout()
# plt.xlabel('Year')

# ### PLOT NETATMO VS SMHI MONTHLY STANDARD DEVIATION REGION SOUTH
# fig, ax1 = plt.subplots()
# ymin = 0
# ymax = 35
# ax1.set(ylim=(ymin, ymax))

# plt.scatter(x_ax, standard_dev_north, s = scatter_dot_size, color = "blue")
# plt.scatter(x_ax, standard_dev_mid, s = scatter_dot_size, color = "green")
# plt.scatter(x_ax, standard_dev_south, s = scatter_dot_size, color = "red")
# plt.plot(x_ax, standard_dev_north, linewidth = 0.4, color = "blue", label = "North region")
# plt.plot(x_ax, standard_dev_mid, linewidth =  0.4, color = "green", label = "Mid region")
# plt.plot(x_ax, standard_dev_south, linewidth =  0.4, color = "red", label = "South region")

# plt.xlabel('Month')
# plt.ylabel('Standard deviation')
# # plt.title('Netatmo & SMHI monthly std average, Standard deviation regions\n')
# plt.legend(loc="upper left", fontsize = "x-small")
# plt.xticks(x_ax, months_short_48, rotation = x_ax_rot, rotation_mode="anchor",fontsize = fontisize)
# ax2 = ax1.twiny()
# ax2.plot(years, [40, 40, 40, 40, 40], color='blue', linewidth = 0)
# fig.tight_layout()
# plt.xlabel('Year')
```