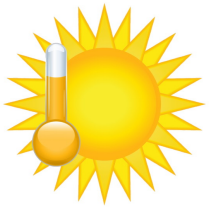
Come Rain  or Shine 

Working with Time Series Data in Python

Using Historical Weather Data
from Hannover, Lower Saxony, Germany

Data provided by

DWD

Written by Brita v. Bartenwerffer
July - October 2021

Come Rain or Shine
Working with Time Series Data in Python

Table of Contents

Introduction

If asked what the most common topics are when making small talk, "weather" is almost certain to lead the list. At least for those of us who live in climes where the weather is rather uncertain and changeable. There is always something to comment on: too warm, too cold, too wet, too dry, too windy. Try finding someone who will simply state that the weather is just right whatever the weather is like! In general, weather is a good conversation starter because it's always happening and people run little risk of getting into a hot debate. Later on, topics might move on to current events, food, sports, entertainment, depending on who one is talking to.

But what, exactly, is weather? Merriam-Webster defines weather as "the state of the atmosphere with respect to heat or cold, wetness or dryness, calm or storm, clearness or cloudiness"[1]. It's sunshine and rain, snow and wind. It's also less obvious air pressure and relative humidity, the latter of which makes for muggy summer days when high. Weather is what's happening right now, or yesterday or tomorrow or on any other specific day, even if it is a long time on the past. Do not confuse this with climate. Climate does not look at the weather on any single day. It refers to weather conditions that are averaged over a long period of time, say 30 or 50 years.

National weather services have usually kept journals of weather conditions for long periods of time. Often, these measurements have been taken by automated weather stations which record temperature, air pressure, moisture and other variables throughout each day. The data would then have been compressed, either into hourly or daily observations, and made available to scientists and often the public for analysis. While some skill and understanding of the data is needed people can check out weather conditions on almost any given day as well as observe long-term trends.

Dataset and Location

My choice for this project was weather data recorded in Hannover, Lower Saxony, Germany. The data is provided by the German national weather service Deutscher Wetterdienst (DWD) and can be downloaded free of charge from this link:

https://www.dwd.de/DE/leistungen/klimadatendeutschland/klarchivtagmonat.html

Datasets are available for quite a few weather stations across Germany but I have chosen Hannover because it's the town where I used to work before the pandemic hit us. There are two files available for download:

- Historical data which ranges from January 1st, 1936 to March 30th, 2020 (the end date may change over time)
- Current data which ranges from January 19th, 2020 up to the current date which was July 21st, 2021 when I started this project (the start date may change over time as well; it seems the current file comprises the past 18 months)

The column names in the data file are not self-explanatory but there is a metadata file provided which explains what each column represents. It also provides

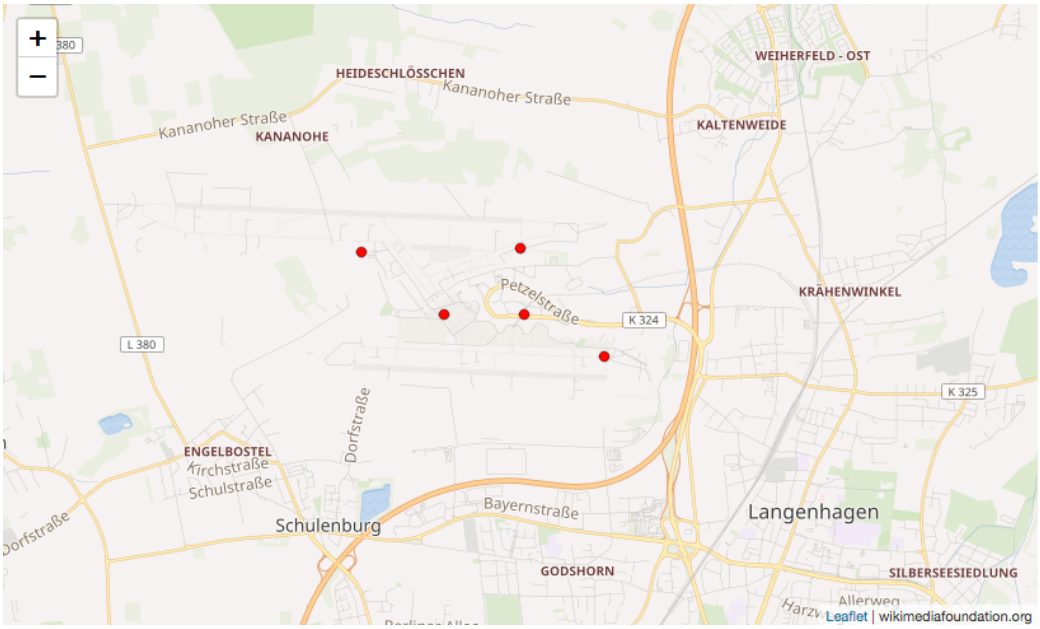
[1] https://www.merriam-webster.com/dictionary/weather

information on how the data for each column was generated during different periods of time. This is an abbreviated overview of all columns:

Parameter	Description	Values in
FM	Daily Mean Windspeed m/s Sensor Network 3	m/sec
FX	Maximum Windspeed Sensor Network 3	m/sec
NM	Daily Mean of Cloud Coverage	eights
PM	Daily Mean of Air Pressure	hpa
RSK	Daily Height of Precipitation, Rainfall	mm
RSKF	Daily Kind of Precipitation	num. code
SDK	Duration of Sunshine, Daily Sum	hour
SHK_TAG	Daily Amount of Snowfall	cm
TGK	Minimum Air Temperature near Ground, at 5cm Height	°C
TMK	Daily Mean of Temperature	°C
TNK	Daily Minimum Air Temperature at Height of 2m	°C
TXK	Daily Maximum Air Temperature at Height of 2m	°C
UPM	Daily Mean of Relative Humidity	%
VPM	Daily Mean of Vapour Pressure	hpa

Unfortunately not all columns are listed in the metadata file but I will come to that later.

The station has moved around a bit during its existence. However, all locations are on the grounds of the local airport "Flughafen Hannover" between the Northern and Southern Runways. This is also provided in the metadata.



The station is currently situated at a crossway leading from the taxiway parallel to the Northern Runway onto the apron (red dot farthest to the left). The airport tower is also located near there.

Importing and Preparing Data

As mentioned above the data is provided in two files: historical up to March last year and current since January last year. Since these are plain text files I decided to merge them using simple Copy & Paste rather than import both separately into pandas dataframes, append current data to historical data and then search for and delete duplicate rows.

Some specifics have to be taken into account while importing:

- The column that should be used as index is MESS_DATUM. The format is given in YYYYMMDD and can be easily converted to datetime64 with the parameter `parse_dates`. This transforms the raw data into time series data.
- Missing values are recorded as -999 and need to be changed to NaN-values using the parameter `na_values`.
- There are a lot of initial white spaces in the data, resulting in column names like " FX". This can be avoided by setting `skipinitialspace` equals True.

These are the first two rows of the dataframe after importing:

	STATIONS_ID	QN_3	FX	FM	QN_4	RSK	RSKF	SDK	SHK_TAG	NM	VPM	PM	TMK	UPM	TXK	TNK	TGK	eor
MESS_DATUM																		
1936-01-01	2014	NaN	NaN	NaN	5	0.3	1	3.1	0	6.7	7.6	993.4	7.9	70.0	9.9	6.9	NaN	eor
1936-01-02	2014	NaN	NaN	NaN	5	0.2	1	2.4	0	6.7	8.0	988.2	7.1	77.0	9.4	5.9	NaN	eor

The entire dataframe has 30,154 rows or observations, distributed across 18 columns.

It seems pretty obvious that the `STATIONS_ID` will be "2014" throughout the data. While the weather station did move it seems unlikely that its designation changed at all during the time. The id also has no influence on weather measurements. So this column can be dropped. Iterating over all columns additionally told me that the last column, `eor`, contains just the one value "eor" (end of record?) so this will not add anything to the analysis and can be dropped.

The columns QN_3, QN_4 and RSKF only contain a few unique values, namely 4, 4 and 6 respectively. There is no information provided for the first two columns and the values are numerical: 1, 3, 5 and 10. They don't provide any immediate clues as to their meaning. A description of the last column is given in the metadata and is defined as a numerical code. There is no hint of how to translate the values 0, 1, 4, 6, 7 and 8 into anything useful although machine learning *might* be able to make sense of it. However, since this is clearly categorical information and not interesting as a predictor I decided to drop these three columns as well.

This left me with 13 columns that didn't have telling names: `FX`, `FM`, `RSK`, `SDK`, `SHK_TAG`, `NM`, `VPM`, `PM`, `TMK`, `UPM`, `TXK`, `TNK`, `TGK`. I decided to rename all columns keeping as closely as possible to the metadata provided. This saves on scrolling time and, more importantly, makes everything more easily readable although it means more typing and thus longer code.

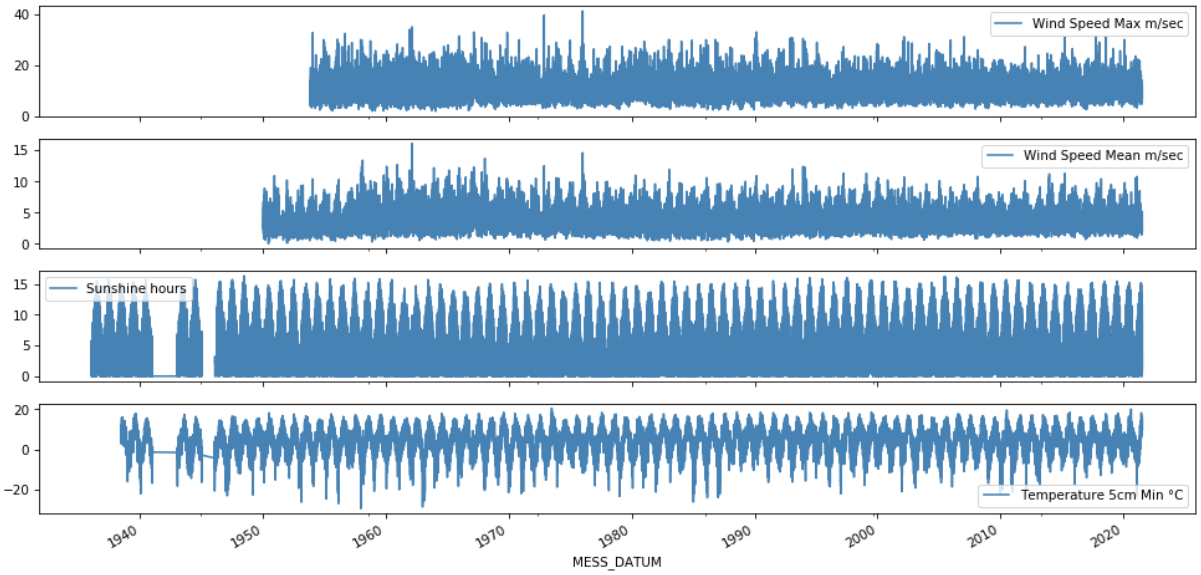
Missing Values

When I was importing the data I already learned that a value of -999 meant that there was no measurement recorded for this date and measurement. A quick check using `.isna().sum()` on the dataframe told me just how many out of 30,154 observations are missing in each column.

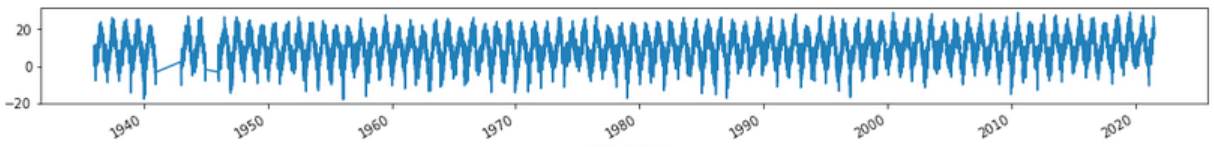
Only four columns actually have missing data:

```
Windspeed Max m/sec          5481
Windspeed Mean m/sec         4057
Sunshine hours                107
Temperature 5cm Min °C        943
```

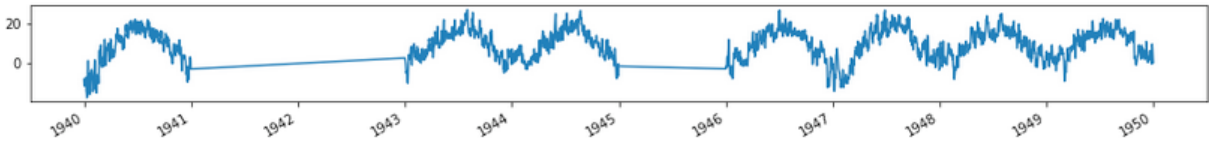
To get some idea of whether the data is missing in whole chunks or just every now and then I decided to plot all four columns:



This made it very clear that there are, indeed, missing whole chunks of data rather than single observations across the entire time. Wind speed doesn't seem to have been measured before 1950 at all and there are also whole years missing from `Sunshine hours` and `Temperature 5cm Min °C`. Neither can be filled using methods like backfill where the next day's value is assumed to fit the observation of this day. Furthermore, for `Sunshine Hours` only 107 days have been counted as missing but the missing chunks are obviously much longer. To see whether this applies to the whole dataframe I decided to plot the mean temperature and see what this looks like.



Again, we observe data missing for several years although no missing data was found in the dataframe. There seem to have been no observations at all for this weather station during these times. The longer phase is during World War II so this might mean that the airport had been bombed and the station destroyed. The second phase is also in or just after the war. Plotting just the decade's data confirms this:



I haven't been able to find any confirmation that the weather station really was destroyed or put out of order by enemy action but it seems likely. Of course, the cause doesn't make any difference to the missing data.

Going forward I checked that the first observation for Wind speed was on November 1st, 1953 and then decided to use only data from January 1st, 1954 onward in order to not to start in the course of the year. This did not take care of all missing data though but it was much reduced:

```
Windspeed Max m/sec          62
Windspeed Mean m/sec         38
Sunshine hours                1
Temperature 5cm Min °C        1
```

Iterating over the observations gave me a list of dates where there was no data for at least one of the columns recorded:

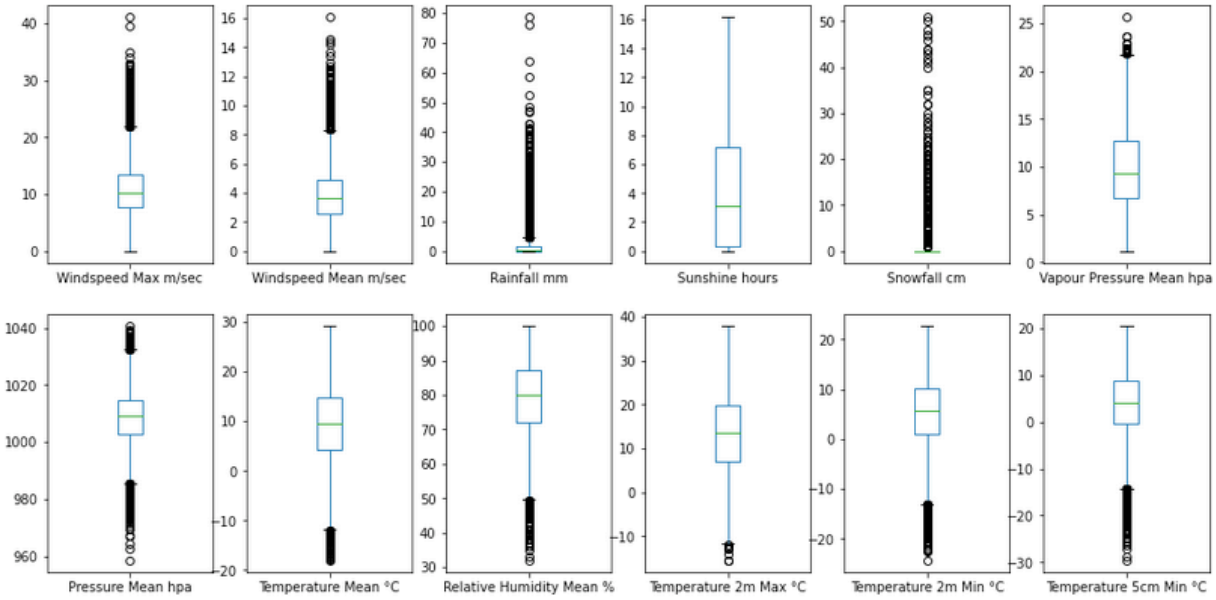
```
['16.06.1981', '17.06.1981', '11.08.1981', '12.08.1981', '13.08.1981',
 '10.09.1981', '11.09.1981', '12.09.1981', '16.09.1981', '17.09.1981',
 '18.09.1981', '24.09.1981', '25.09.1981', '26.09.1981', '27.09.1981',
 '28.09.1981', '29.09.1981', '30.09.1981', '01.10.1981', '02.10.1981',
 '21.10.1981', '01.06.1992', '14.07.1992', '15.07.1992', '19.08.1992',
 '20.08.1992', '29.01.1995', '02.02.1995', '03.02.1995', '04.02.1995',
 '05.02.1995', '21.04.1999', '09.06.1999', '22.07.1999', '03.05.2000',
 '16.08.2000', '15.11.2000', '10.01.2001', '01.04.2001', '02.04.2001',
 '03.04.2001', '04.04.2001', '05.04.2001', '12.09.2001', '13.09.2001',
 '14.09.2001', '15.09.2001', '16.09.2001', '17.09.2001', '18.09.2001',
 '19.09.2001', '01.11.2001', '12.12.2001', '23.01.2002', '24.01.2002',
 '25.05.2004', '14.08.2004', '15.08.2004', '16.08.2004', '23.01.2005',
 '04.08.2008', '05.08.2008', '06.08.2008', '25.04.2014', '26.04.2014',
 '27.04.2014', '28.04.2014', '29.04.2014', '18.05.2014', '13.07.2016',
 '14.07.2016', '18.06.2017', '19.06.2017', '01.07.2017', '02.07.2017',
 '03.07.2017', '16.09.2017', '17.09.2017', '18.09.2017']
```

Unless I have miscounted no more than eight values are missing in a row. It makes sense to fill their values to keep from having to remember to drop missing values for specific plots and algorithms. Some of them are able to ignore missing data but others explicitly require data without NaN-values.

In this case I had to use two methods. Since there are several days in a row missing it seemed more logical to interpolate values linearly rather than just use the previous or next one. After doing so I was still left with missing values albeit fewer. For the rest I used backfilling, filling the missing value with the next available value. This is much less accurate than interpolation but there weren't that many values in comparison to the total amount of observations in the first place so it shouldn't play too big a role for any models using this data. I could also have used interpolation a second time but this only works reliably when there is more than one value missing.

Outliers

The easiest way for humans to spot obvious outliers is by plotting boxplots for all numerical columns, which will visualize outliers clearly:



From the look of it there don't seem to be any obvious outliers that might be attributed to measurement errors or artefacts. There will have been days with high winds, strong rainfall or extreme temperatures, especially taking climate change into consideration. Just remember the flooding we have seen recently!

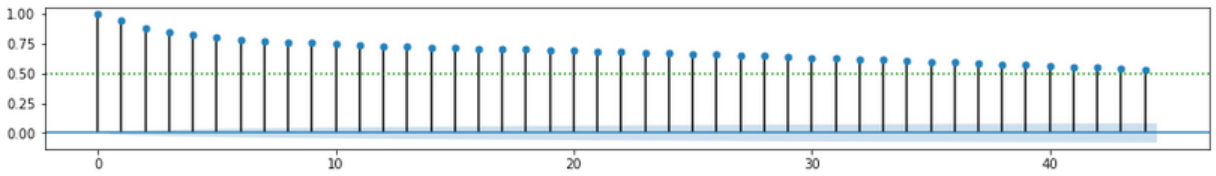
So, in order not to lose any useful data from the time series I decided not to drop any rows with high values. Dropping the rows would also mean that I would need to re-fill these rows because the dataframe would be missing whole days then which is not very useful for a time series and would also distort the actual observations.

Autocorrelation and Partial Autocorrelation

This is a statistical analysis that compares the data to a time shifted or lagged version of itself. Values of correlation are between -1 and +1. If values are below -0.5 or above +0.5 the data is highly correlated and the correlation is statistically significant.

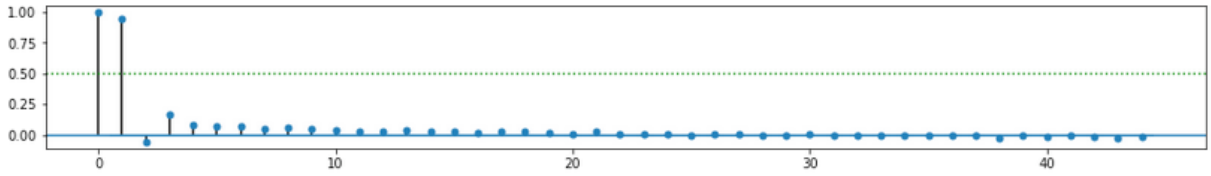
Mean Temperature

Autocorrelation looks at all data points or observations for one lag, e.g. for a lag of 5 the algorithm takes the last five values into account. For the mean temperature a high autocorrelation is observed, regardless of the length of the time shift or lag:



Even for a time shift of 44 days the algorithm thinks it could predict the next day's or even days' value(s) with some accuracy just by looking at all values from the previous 44 days.

This plot looks very different for partial autocorrelation. In this case the algorithm only looks at the one value of each observation but not at any of the intervening ones and both the correlation and the confidence drop to zero very quickly.



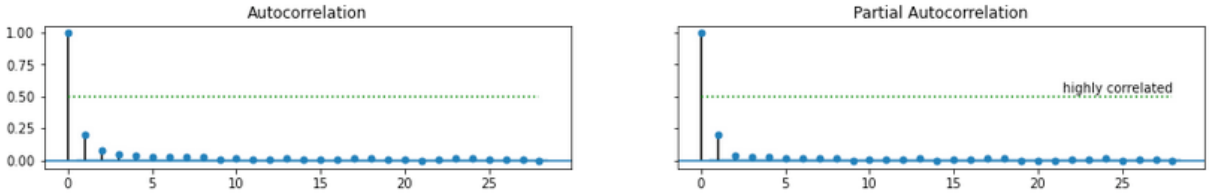
In fact, this time the algorithm is convinced that there is only a strong correlation between two successive days and it would therefore only be able to predict the next day's temperature with any confidence.

Autocorrelation can be considered as showing a certain trend. It is not likely that temperatures will fluctuate wildly over a period of five or six weeks. In general we observe this over the course of the year as well. Except for extreme weather conditions in summer, mostly after thunderstorm fronts passed through, we generally don't see the temperature drop or rise suddenly by a couple of degrees.

Partial Autocorrelation on the other hands shows clearly that the mean temperature on one day is not directly related to the temperatures on the preceding days. We also observe this when we experience a break in the weather. The fact that it is really warm today does not mean that it will be really warm the day after tomorrow (or even tomorrow for that matter).

Rainfall

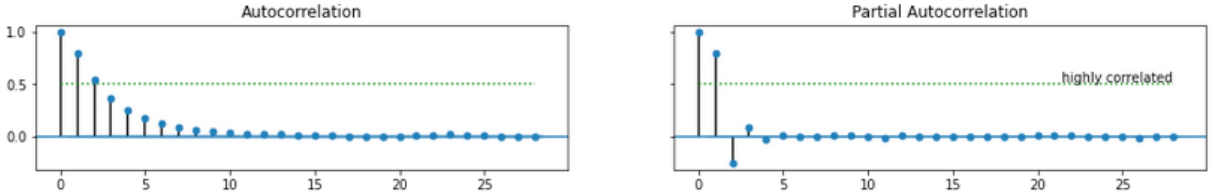
Looking at how much rain falls on any given day both algorithms find that there is no statistically significant correlation at all between the amount of rain on one day and the next:



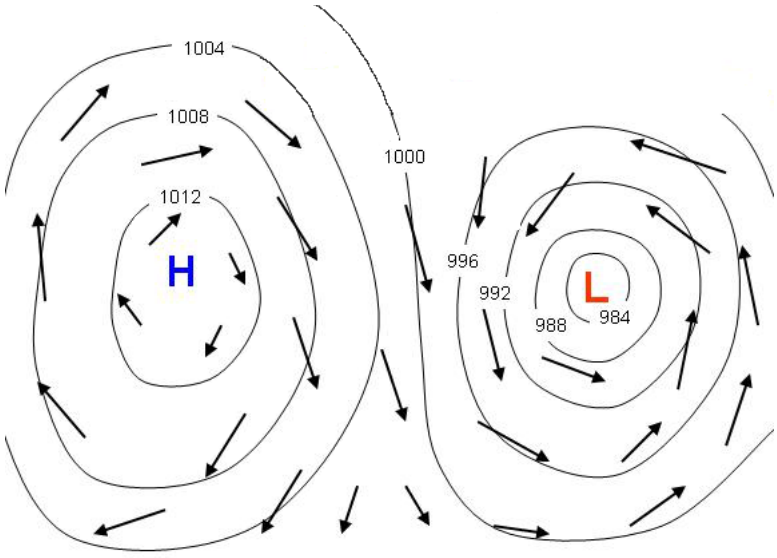
This, again, is something we can observe in daily life. In fact, even weather predictions can be disastrously wrong as the South of Hannover found out in early June this year. There had been warning (alert level "orange") that they were to expect thunderstorms but there had been no prediction of the heavy rainfall and subsequent partial flooding they saw. On the other hand, the national weather service was warning of very heavy rainfalls (alert level "red") in mid-July and they barely had a drizzle!

Air Pressure

Air Pressure is a bit more reliable than rainfall but amazingly enough there isn't any high correlation after the first two days either.



During the course of the year we often experience stable weather conditions with persistent high-pressure regions. These often bring many weeks of clear, cold weather in the winter and dry, hot(-ish) weather in the summer with predominately Easterly winds. This observation gives the impression that there should be a high correlation for autocorrelation over the period of up to 28 days shown in this example. In actual fact however, a high-pressure region is not defined by a constant value of air pressure. It is made up of an onion-like structure of steadily decreasing pressure. The same goes for lower-pressure areas where the pressure is increasing from the centre out. This can be seen nicely in this graphic where the numbers represent the air pressure at this line:



http://cimss.ssec.wisc.edu/satmet/modules/7_weather_forecast/wf-4.html

This means that even for a strong and stable high-pressure region the air pressure itself is fluctuating within a range of values. Provided a pressure area is not moving very rapidly across the country measurements do not fluctuate by that much over a couple of days which is why the autocorrelation algorithm is fairly confident that it could predict the next day's pressure with some accuracy.

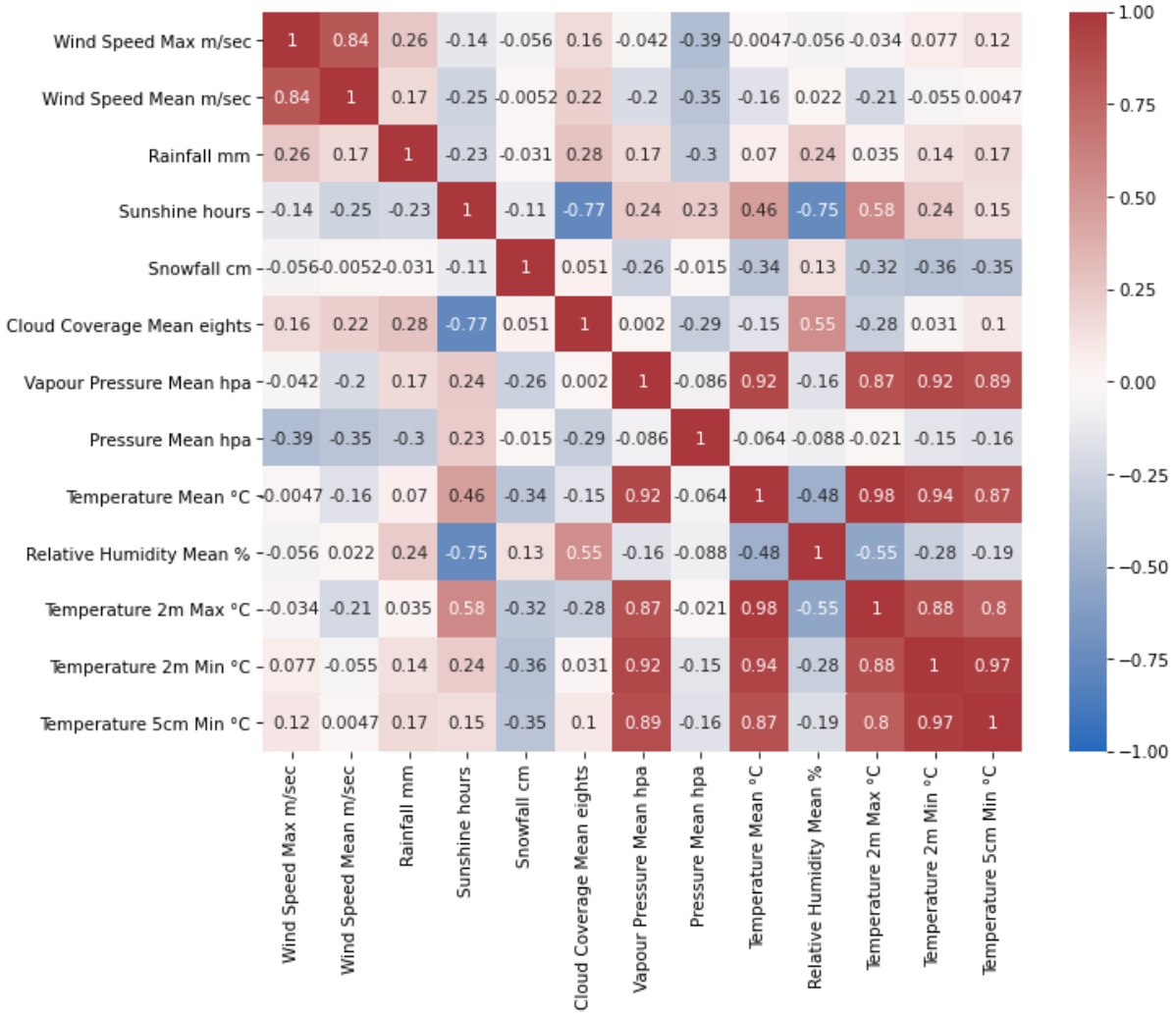
Correlations Between Different Measurements

In everyday life we often find that low pressure areas bring rainfall with them and that high pressure areas usually mean sunnier weather. But is there really a correlation between air pressure and rainfall or sunshine? Is there even a correlation between

the latter two?

Overview

As with autocorrelation for the correlation between columns the values also range from -1 to +1. A positive correlation means that as one value goes up, the other also goes up. For a value of Zero the two values don't move together at all. There is no correlation. A negative correlation means that when one value goes up, the other goes down. The higher the absolute numbers are the higher the correlation between the variables is.



While there are a couple of strong positive correlations they are also the expected ones. If the mean temperature of the day goes up so do the minimum and maximum temperatures. Vapour Pressure is dependent on temperature so there is a high correlation as well. The same goes for wind speeds and maximum wind speeds. If the mean wind speed is high it's quite likely that the maximum wind speed of the day is also high.

Temperatures and hours of sunshine also show a positive correlation but it is not a strong one. Relative humidity, which has a strong correlation to sunshine, also depends on temperature and is comparable to sunshine in this respect. That the correlation isn't any stronger in either case can probably be explained by the seasons as well as the fact that storm fronts in autumn and winter often bring warmer

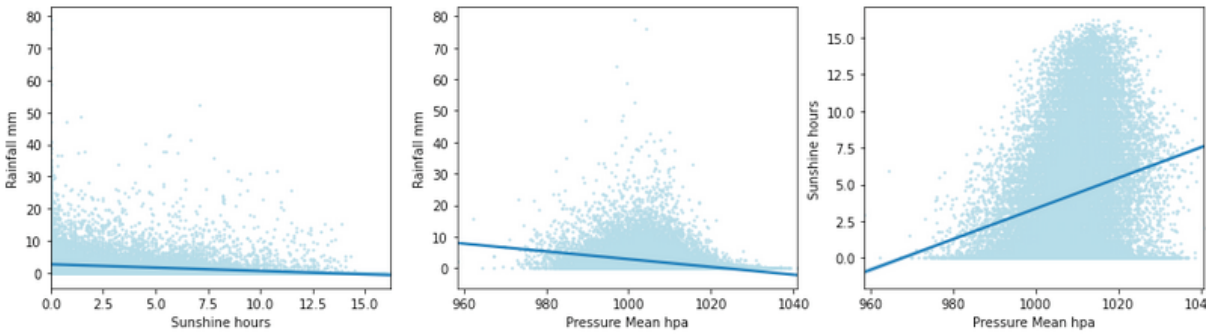
temperatures but little sunshine with them.

There are two relatively high negative correlations at -0.77 and -0.75. The first is an obvious one between sunshine and cloud coverage. With less cloud coverage there is more sunshine. The second is between sunshine and relative humidity. In this case the more sunshine hours we get the less relative humidity there is. We can observe this on summer days: Often days start damp with light clouds or even slightly foggy. With the Sun climbing high in its orbit the humidity goes down and often gives way to a clear day.

A slight negative correlation exists between air pressure and wind speeds. When air pressure goes down, wind speeds go up. In Germany low-pressure regions are often the remnants of hurricanes during hurricane season which hit Northern Europe after having travelled across the Atlantic. Low-pressure regions are also responsible for storm fronts. Both are accompanied by strong winds as the names suggest. When air pressure is high, the weather is usually calm and there is only a refreshing breeze, if that.

Sunshine, Rainfall and Air Pressure

Looking at the initial question: Yes, there *is* a slight negative correlation between sunshine and rainfall. There is also a negative correlation between air pressure and rainfall and a positive one between air pressure and sunshine. To visualize more clearly what this means let's have a look at three scatterplots illustrating the correlations. Neither of the correlations is really strong as can be seen by the slopes of the lines which are all well below 45 degrees.



In the left plot we can see that rainfall goes down while sunshine hours go up. There are a good many outliers and some of them will be related to thunderstorms which bring heavy rain but generally don't last for long.

The plot in the middle shows that as air pressure goes up, i.e. there is a high-pressure region, rainfall generally goes down. However, this plot isn't quite that clear and might also be interpreted as there being a range of air pressure in which it is more likely to rain or which is simply observed more often than the outer ranges.

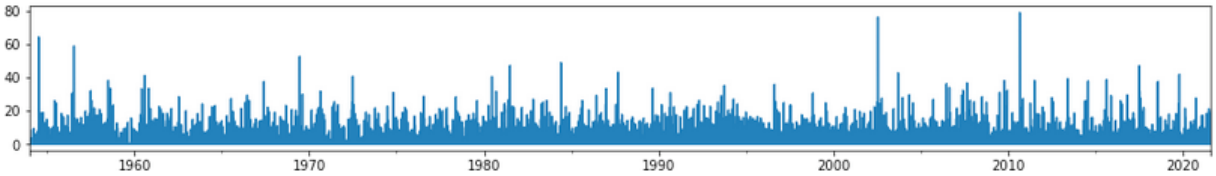
Form the plot on the right it's easy to see that the higher air pressure is the more sunshine there may be. This correlation would probably be even higher if we accounted for the seasons here. Even if the air pressure is the same the days are much shorter in winter and therefore there will be less hours of sunshine even if it is sunny for the whole day.

Seasonality, Trend and Noise

Of course there is seasonality to weather data! And we have all heard that due to climate change there are upward trends with temperatures rising, high winds causing more damage and less rain being responsible for droughts. Can this also be seen in the data from just this one weather station?

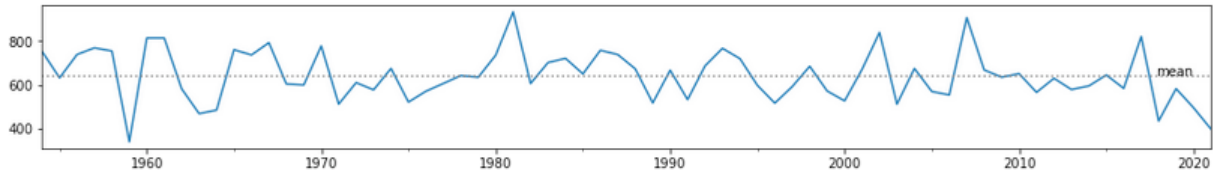
Rainfall

Let's start this analysis off with rainfall by simply plotting all values over time:



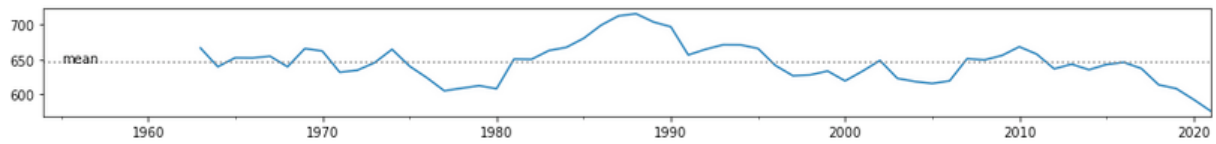
Looking at this busy plot doesn't really give any idea about seasonality or long-term trends. We can see that there are peaks for months (or is it weeks?) in 2003 and 2011 with exceptionally high rainfall. These seem to increase in frequency in more recent years. There seems to a period of below-average rainfall around 1959 and above average around 1994. Otherwise it's hard to judge whether there is an overall up- or downwards trend.

Grouping the data by total rainfall per year makes the plot a whole lot less busy but not any easier to interpret even when adding a line to show the long-term mean:



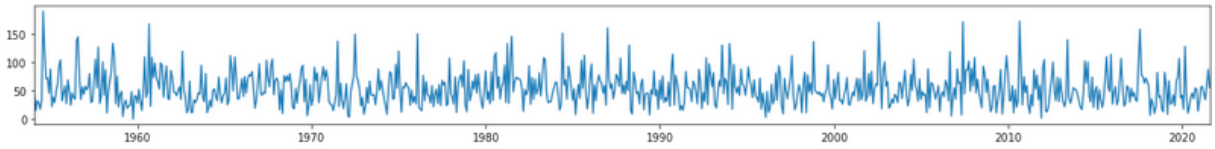
Now we see a peak in 1981 that wasn't apparent in the plot above and the peaks in the last decade get washed out. Seasonality isn't visible at all at this time scale.

This is where the `statsmodels` library can be used. It provides a filter to decompose any given time series using moving averages. In the yearly example above a moving average might be taken every year by calculating the average over the last decade. We can do this ourselves by using the method `.rolling(10)` in pandas and plotting the result:

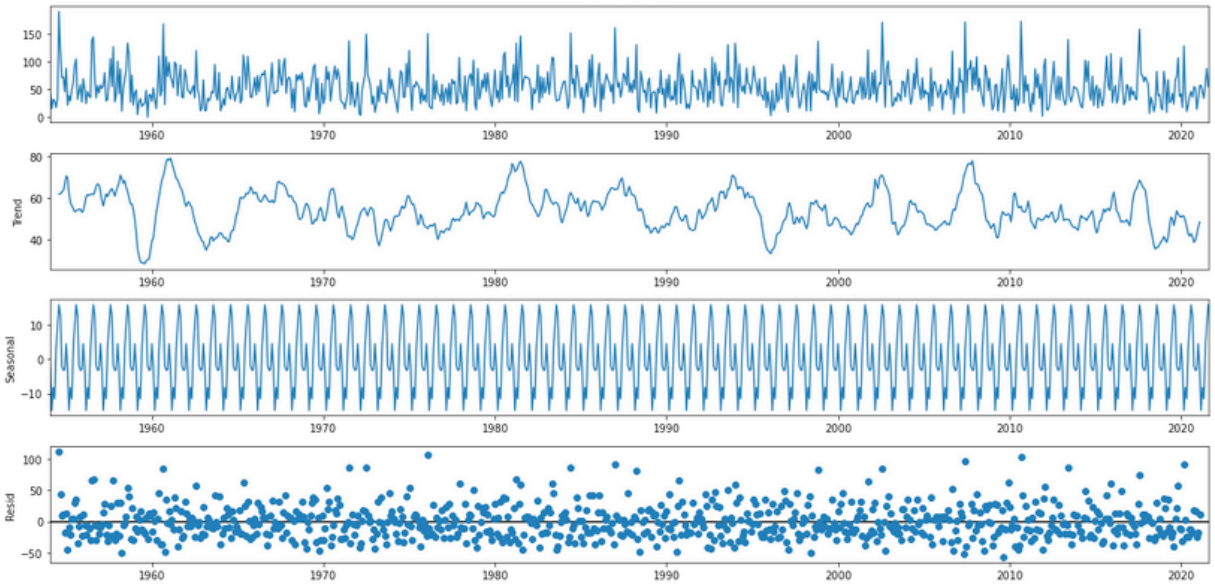


There is no data in this graph for the first nine years because the time spans are not long enough to take an average over ten years. Taking the rolling mean has smoothed out the curve considerably but there still isn't any trend towards less rainfall and droughts visible. Please remember that the strong downward trend at the right is mainly due to looking at only half of the year of 2021. Unless we don't get any rain at all for the rest of the year this will look differently by year's end.

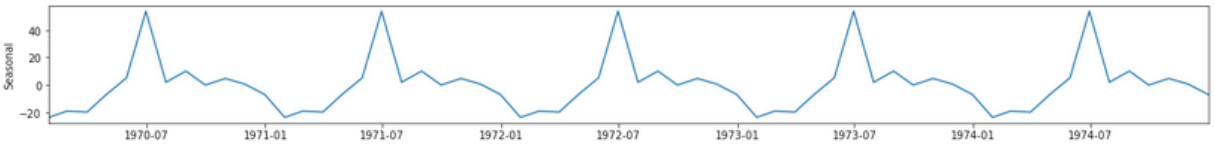
Simply plotting the rolling average for each preceding decade still doesn't show any seasonality. Since seasonality refers to months or seasons it makes sense to resample the data first to a monthly frequency, using the total rainfall per month. This can be easily done by using `.resample('M').sum()` on the data. It makes the original plot for daily measurements much less crowded:



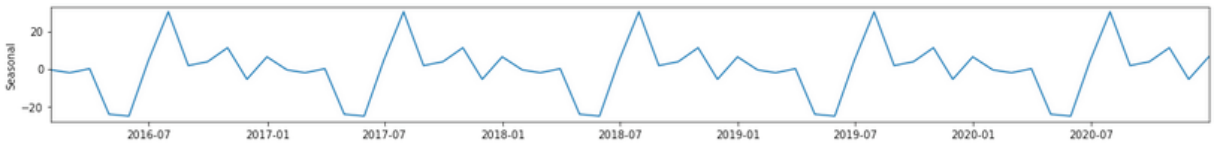
However, it's still near impossible to pick out any seasonal trends within one year due to the large time period plotted. This is where `.tsa.seasonal_decompose` of the `statsmodels` API comes in. With just one line of code it returns the observations, the seasonal component, the trend component and residuals which show by how much data points deviate from the mean value of the dataset. This is also called standard deviation and can be a measure for noise in the data. The more deviation there is the noisier the data. The results can then be plotted:



The algorithm has found a clear seasonal component which indicates that there isn't much rainfall at the beginning of the year and the peaks are in the middle of the year. To make this clearer, here is the seasonal plot for 1970 through 1974 (period chosen for no specific reason):



The model above found a general seasonality of the data for the whole period of almost 65 year. Reducing the data to a much shorter period of time confirmed this but we need to keep in mind that these are all statistical models and not accurate year-by-year renditions of the seasonality. The short-term seasonality actually changes when looking at recent data:



Here, the peak remains in the middle of the year but the dry periods have moved from the beginning of the year to just before the peaks.

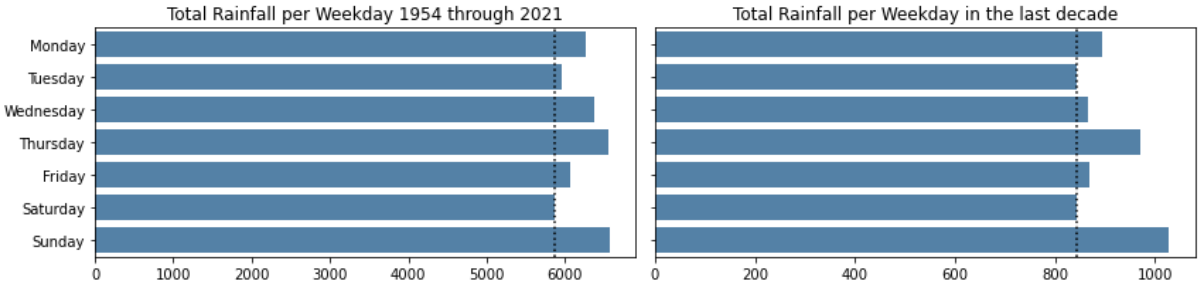
Overall, we can conclude that there is some seasonality to the amount of rain throughout a year but no clear trend over the long-term.

Side Issue: Taking the Day of Week into Account

Looking back over the past two years or so it seems that it's raining more often when I'm off work and less so when I'm not. I remember reading a book[2] years ago that there actually is a scientific explanation why it's raining more often on weekends and it had something to do with aerosols that are emitted into the atmosphere throughout the week. As they accumulate they provide the seeds for clouds which will keep growing. When they are heavy enough and can no longer hold onto the water it's beginning to rain. With less aerosols emitted over the weekend the cycle begins again from Monday. Mind you, I'm not altogether certain I remember this rightly.

Scientists from Switzerland certainly deny this[3] and say that it's pure imagination that it's wetter on weekends. They say that if anything it's a statistical fluke and the distribution throughout the week is uniform.

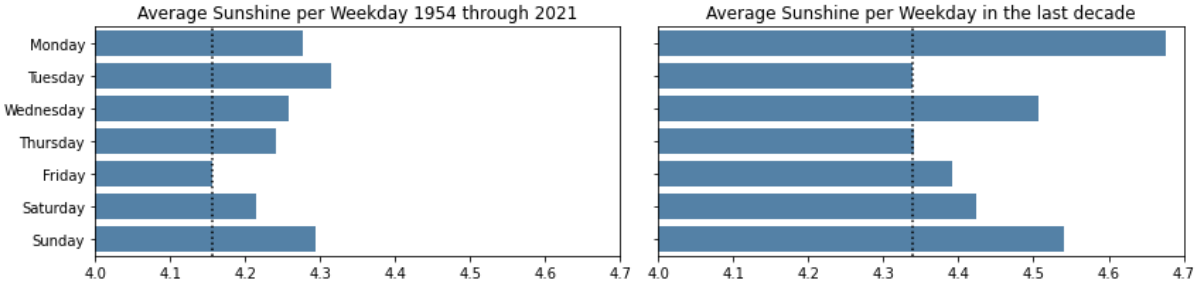
I was interested in seeing whether my data supports my subjective impression that there is more rain on my days off work, both in the long-term and in the shorter term. What I got was a confirmation, sort of:



While the scales are different since I have taken the sum rather than the mean the trend is basically the same for both time periods. The plots for the mean don't look any different since they represent the sum divided by the number of observations. Sundays are really the wettest days overall, decreasing on Monday and even more so on Tuesday. Then a new cycle begins, dropping most of its moisture on Thursday before it calms down again for the next two days. For the long-term Saturday is the driest day, for the shorter term it's actually Tuesday but by only 1.1 mm difference compared to Saturday. This is not even visible in the plot showing the line at the minimum value.

[2] Alex Bojanowski: Nach zwei Tagen Regen folgt Montag: Und andere rätselhafte Phänomene des Planeten Erde (ISBN: 978-3421045348)
[3] https://www.wetter.at/wetter/oesterreich-wetter/regen-am-wochenende-statistitk-erklaerung/133383952

Plotting the distribution for sunshine using mean values gives a surprising result:

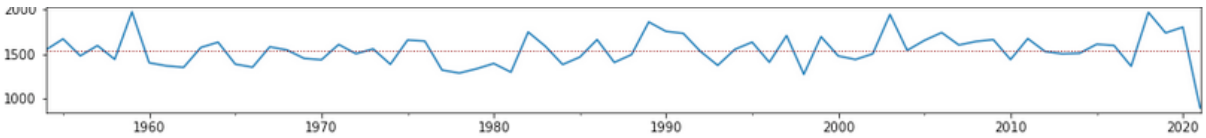


Please be aware that the x-axis has been scaled to show only values above 4 hours in order to make the differences in the days more obvious.

With Sundays being the wettest days followed by Thursdays for both periods I would expect to see the least sunshine on those days. We already found that there is a slight negative correlation between rainfall and sunshine, meaning as one goes up, the other goes down. In spite of this, the day with the least sunshine is Friday for the long-term and Thursday for the shorter term. On top of that, the sunniest day in the short-term is also the third wettest.

Moreover, we can observe that average hours of sunshine on any given day in the past decade alone are higher than in the past 65 years combined. The minimum mean value in the past decade is even higher than the maximum mean value overall. Now, there seems to be a clear trend in the data!

Accumulating the sunshine hours by year and plotting this data doesn't support this view. There is no clear upward trend towards more recent times.



There was even a minimum in 2017 and the years before didn't deviate much from the long-term mean, shown as a red line. I have heard that 2020's sunshiney days were a direct result from less cloud coverage and subsequently less rain because industries were shut down worldwide due to covid-19. This may well be the case but the sunniest year of the decade was 2018 so far.

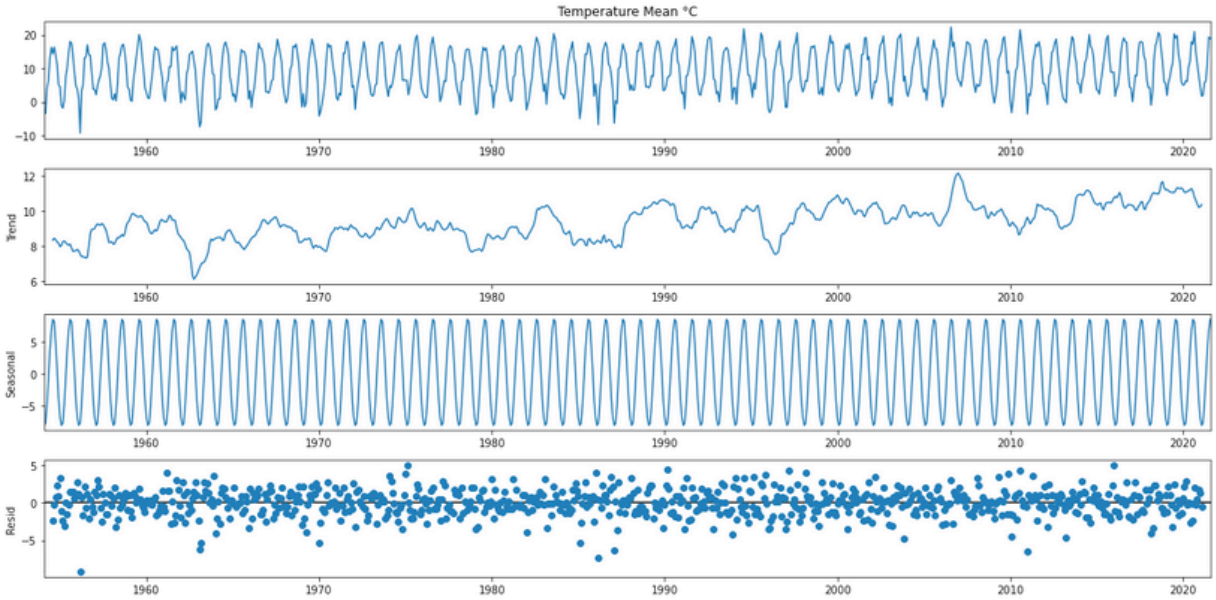
If you are interested in the actual numbers of the last decade have a look at the numbers by each year's end:

2011-12-31	1673.200	2016-12-31	1597.500
2012-12-31	1529.400	2017-12-31	1364.223
2013-12-31	1503.000	2018-12-31	1966.716
2014-12-31	1509.000	2019-12-31	1736.866
2015-12-31	1610.116	2020-12-31	1801.121

For 2021 901.481 hours have added up until July 21st. This is less than half the amount in 2018 so 2021 isn't set to break the record. In fact it may well end up being slightly or even well below last year, thus breaking any very recent upward trend.

Mean Temperature

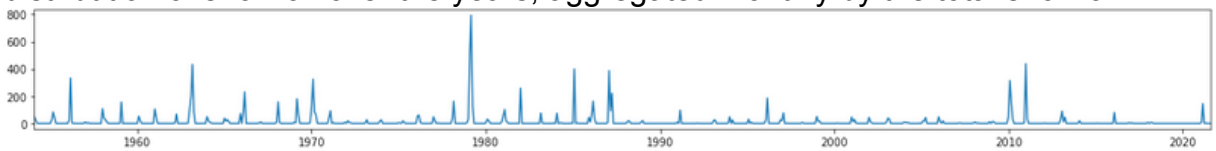
The mean temperature or the rise of temperatures since the beginning of weather records is the most frequent topic in the news when it comes to climate change. When talking about weather records these mostly refer to extreme temperatures, generally extremely high rather than frosty. Let's see what `statsmodels` makes of the data. For this analysis the data has been aggregated to each month's mean:



This time `statsmodels` is not really needed to look at seasonality since this is already obvious from the observations in the top plot. We can also see from that plot that there have been a couple of exceptionally cold winters but not so many exceptionally warm summers. There may be a slight upward trend in temperatures, mainly because we did not have any cold winters over the past decade. Although we may remember the "great snowfall" in early 2021 vividly it wasn't really cold enough for an extended period of time to show up in a mean taken over a month. Here the trend extrapolated by `statsmodels` helps us to see that there really is an upward trend in mean temperatures over the past 65 years. The residuals also show less outliers towards the lower end of the temperatures as time moves forward.

Side Issue: Snowfall

Talking about the "great snowfall" in early 2021 let's have a quick look at the distribution of snowfall over the years, aggregated monthly by the total snowfall:

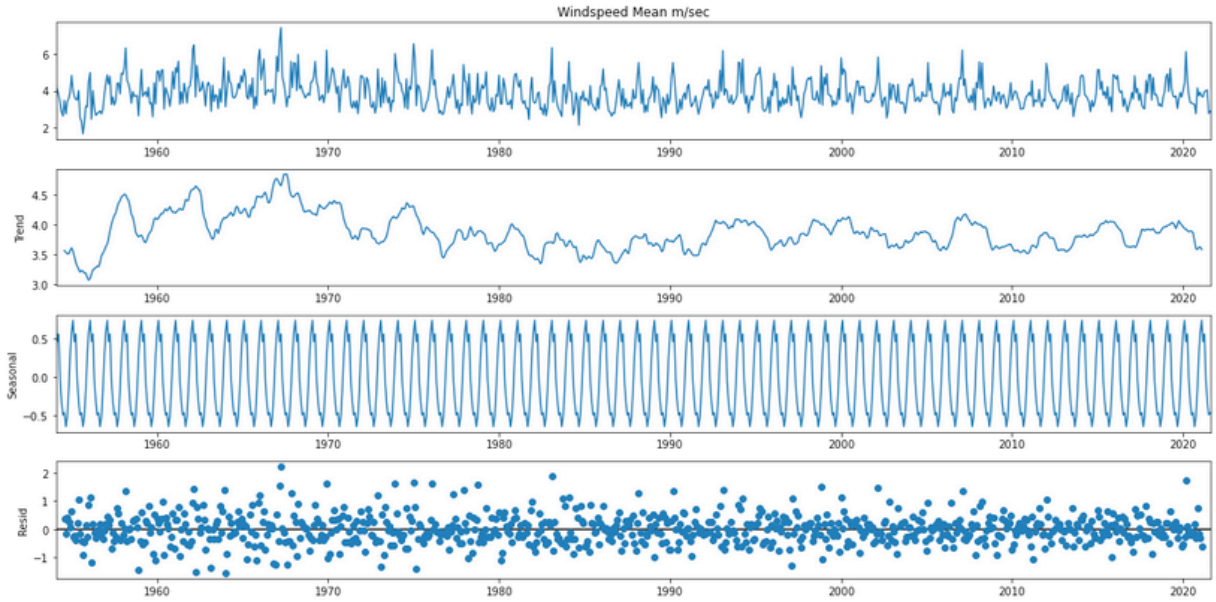


Anyone remembering the very snowy winter of 1978/79? I was in primary school then but I still remember the enormous masses of snow and days off school because of freezing rain. We children built many snowmen and with the help of our parents some grew bigger than we were! This is the only white winter I can remember and the plotted data supports this. I think we don't need to dream of a White Christmas in our latitudes anytime soon as we mostly only see a sprinkling of snow.

Wind Speeds

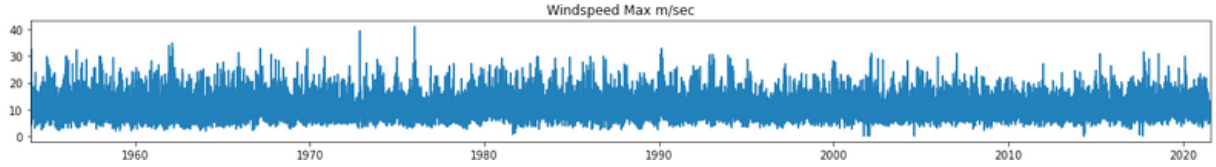
Every time a storm passed through and caused damage or halted rail traffic we can hear on the news that storms are increasing in frequency and strength due to climate change. Looking back over the years my memories support this without having any evidence. When I was a lot younger we were talking about autumn storms, and autumn was when I remember most of them occurring before the weather settled into quieter winter weather. Now we seem to see storms at all times during the year.

The data certainly supports my memories but current observations do not. Again the data has been resampled using the monthly mean.



There is a clear seasonality in the data even without looking at the corresponding plot in that there are seasons when it is more windy overall. However, the data does not support that it is windier nowadays than it used be. In fact, the trend and the residuals show that the distribution has become both more uniform and with less deviation from the long-term average.

The data also provides the maximum wind speeds per day so let's plot these without any aggregation to see if these peaks have increased over the years.



Without downsampling the plot is very dense but it's easily visible that we don't see an increase in wind speed peaks. Overall we see higher wind speeds and more peaks in the first decade of the data than in the last. There also does not seem to be an increase in frequency although this is a bit difficult to judge from such a busy plot. One thing that might be inferred from the plot is that the minimum maximal wind speed has increased from the 1950s and 60s compared to the recent two or three decades.

Adding Another Time Series: Covid-19 Case Numbers

Dataset

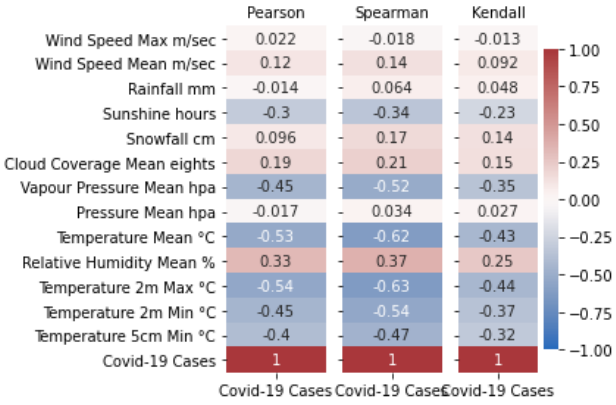
This dataset is provided by the Robert-Koch-Institut Germany (RKI). It can be downloaded from the Covid-19 Datenhub, which can be found here: https://npgeo-corona-npgeo-de.hub.arcgis.com

Since I'm only interested in numbers concerning Hannover, I opted to download only data for Lower Saxony (Niedersachsen), loaded that into a pandas dataframe and then filtered for "Region Hannover". Each date has several rows because the local authorities transmitted cases and deaths separately, accumulated by age groups. This means that the data needed to be grouped by date after importing, the relevant column being "Meldedatum". Being only interested in case numbers, which can be found in the column "AnzahlFall", this returned a pandas series.

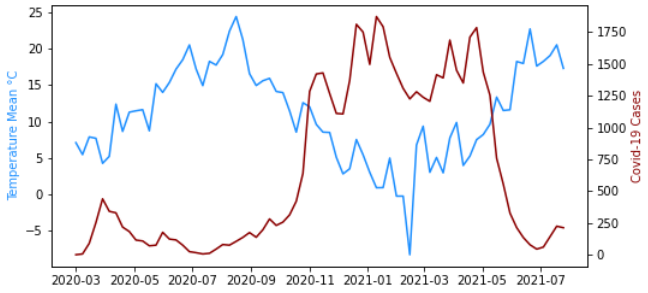
The weather data used above had to be filtered to contain only dates from March 1st, 2020 onwards because no numbers were recorded for covid-19 before this. Having done this, the series containing case numbers could be added to the filtered dataframe as a separate column.

Correlation Between Weather and Covid-19 Case Numbers

Every couple of days we can hear on the news that covid-19 is seasonal, much as any other corona virus like the common cold. Case numbers go down as temperatures go up. Although we are only just into the second summer there is a negative correlation between temperatures, humidity and case numbers as can be seen in the correlation matrices using three different techniques:



At -0.4 to -0.54 the negative linear Pearson correlation isn't really strong but it is statistically significant. The Spearman correlation shows that there is a trend towards a non-linear relationship between the two variables but it isn't much stronger than the linear one. The relationship becomes easily visible with both time lines shown together on one plot using two y-axes because of the different scales of the data:



We need to bear in mind though that from this summer on vaccination rates also influence both infections and case numbers. Recent studies are showing that people who have been fully vaccinated can still get covid-19 but also that cases are less severe. There will certainly be more people who either don't know that they have caught the infection in the first place or who don't bother going to the doctor's to get a test not realizing that their symptoms might point to covid-19, not the common cold.

Stationarity

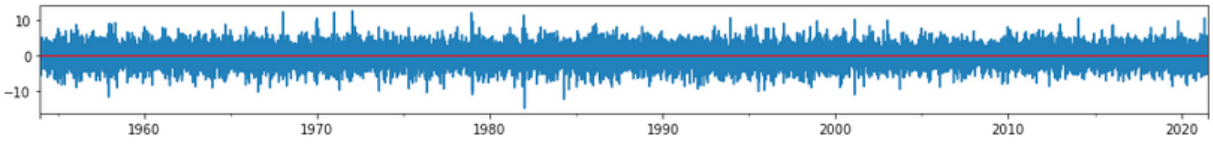
Turning towards modelling it is important to find out whether a given time series is stationary because stationary time series can be modelled much more easily.

A time series is stationary if
- it has no discernible trend
- variance, i.e. by how much it deviates from the mean over time, is constant
- autocorrelation is constant

Mean Temperature

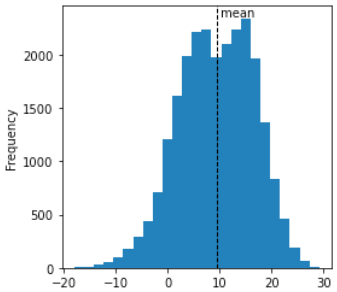
Looking at the plots above it seems obvious that there is a slight upward trend observed and there also seems to be slight downward trend when looking at the autocorrelation. However, neither trend is large, sudden or very obvious.

The variance, on the other hand, is almost certainly constant as can be seen when looking in the changes in temperature per day compared to the previous day. This can be achieved with one line of code using `.diff()` on the time:



There are no changes over time in the amplitudes around the difference of zero and there are few extreme outliers. There is also no trend visible now which means we might be looking at a stationary series after all.

One indication of whether a time series is stationary is a simple histogram:



Except for the dent in the middle just around the mean temperature overall the outline follows a Gaussian normal distribution quite well while being very slightly skewed to the left.

Whether a time series is (not) stationary can be tested using the Augmented Dickey-Fuller Test. I'm not going into the mathematics behind it here. Suffice to say that the test concerns itself with a univariate time series, a series with only one variable, and uses an autoregressive model while testing for different time lags, much as was done above with autocorrelation and partial autocorrelation. It tests whether the series has a time-dependent structure on all levels which would mean that it is not stationary. Because of this the Null Hypothesis (H0) of this test is that the time series under review is not stationary. This will be rejected if the p-value returned is below 0.05 or below a 5% chance that the series is non-stationary.

Once the test is run, it returns test statistics. The most important of these are:

ADF or test statistic	The more negative the value the more likely it is that the time series is stationary
p-value	If below 0.05 reject Null Hypothesis => the time series is stationary
critical test statistics	The ADF should be smaller or more negative than the 5%-value shown here.

For the column `Temperature Mean °C` these are the relevant statistics returned:

```
adf test statistic:     -11.873222515804715
p-value:                6.430163441715136e-22
critical values:        {'1%': -3.430615583017095,
                         '5%': -2.861657379573812,
                         '10%': -2.566832477728774}
```

The ADF value is well below any of the threshold values returned as critical values. The p-value, which measures the probability that the time series is not stationary, is well below 0.05 at 6.43×10^{-22}. Therefore we can reject the Null Hypothesis and assume that this time series is, in fact, stationary.

Using the differences in temperature makes it even more so:

```
adf test statistic:     -44.42254239057009
p-value:                0.0
critical values:        {'1%': -3.430615313635065,
                         '5%': -2.8616572605205115,
                         '10%': -2.5668324143592445}
```

Note that the critical values are still the same but the ADF value is significantly smaller. The decision which of the time series to use depends on the goal of the modelling.

Rainfall

Running the same test on `Rainfall mm` returns these relevant test statistics:

```
adf test statistic:     -49.53430858440807
p-value:                0.0
critical values:        {'1%': -3.4306151415171677,
                         '5%': -2.861657184453057,
                         '10%': -2.5668323738701635}
```

In this case the ADF value by itself in addition to the p-value suggests strongly that this time series is stationary. This is consistent with the earlier analysis in that there was no trend observed. There was also no autocorrelation but that was constant as was the distribution around the mean.

With the ADF value already being near -50 there is no need to take measures to improve the value.

Wind Speeds

Here, we are looking at `Wind Speed Mean m/sec` and `Wind Speed Max m/sec` which had no clear trend but did show that the variance got smoother towards more recent times for mean wind speeds.

We have already seen that the critical values remain the same for each time series so I will only compare the ADF und p-values here:

	Wind Speed Mean m/sec	**Wind Speed Max m/sec**
ADF value	-14.82868348496318	-36.44738312717866
p-value	1.9140716485365906e-27	0.0

The ADF and p-values for maximum wind speeds on any given day shown in the right column are already near perfect for modelling. In fact, using the daily difference instead only improves the results slightly from -36.45 to -38.25.

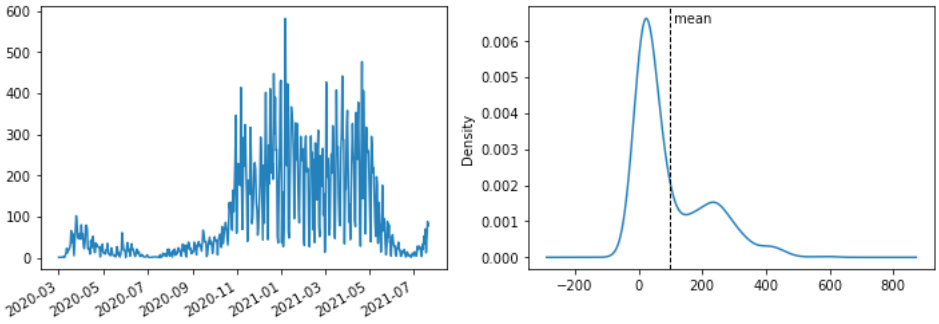
The values for the mean wind speeds per day are more comparable to those we got for the mean temperature. The time series can be used for modelling as is but this time there is a distinct improvement using the daily difference again. The ADF value then is -37.32162967910312 and the p-value 0.0.

Note that there are other transformations possible like taking the log or the square root. In general these should be used only when simpler optimizations don't improve or even worsen the test statistics. In this particular example, taking the square root of the series can hardly be described as improving test statistics. The ADF value improves by -0.2 to -14.90866462975463 and the p-value isn't much different either at 1.4697328977516455e-27. There isn't even one magnitude difference between them. As one of the lecturers said: Simpler is often better!

Counterexample: Covid-19 Case Numbers

So far we have only looked at different columns of the weather data for all years since 1954 and found that the data for all these examples is stationary. Case numbers for covid-19 present a different picture. We only have data available from March 2020 but even so a line plot alone shows at a glance that the data cannot be

considered as stationary:



A stationary line plot would generally look uniform over time while we see few case numbers at the beginning and in recent times with high values in between. Overall high values are rare but the density plot on the right shows a curve that is heavily skewed to the right while the bulk of the values lies to the left of the mean.

Statistical tests like the Augmented Dickey-Fuller Test are only recommended for time series comprising at least two full cycles. In the case of covid-19 we barely have one and half so any results from the test are less meaningful. Nevertheless, the test does confirm that this time series is non-stationary:

```
adf test statistic:     -1.6557841118077146
p-value:                0.45402561221273274
critical values:        {'1%': -3.4439899743408136,
                         '5%': -2.8675550551408353,
                         '10%': -2.569973792117904}
```

Other than for the time series above the ADF value is higher than either of the thresholds provided by the critical values. The p-value is also significantly higher than 0.05 and means that we cannot reject the Null Hypothesis that this is indeed a non-stationary time series.

For modelling this normally means that the data needs to be transformed into a stationary time series using one of the transformations shortly discussed above. I'm not doing this here because there is not enough data for modelling.

Modelling in Theory

With time series data the main goal is to make predictions about the future. For weather data we might want to know what the weather is going to be like in the short-term, this afternoon or in the next couple of days. We might also want to know about future trends, what is now called climate change. Both predictions rely on past data depending on time rather than on regression, classification or clustering of the training data, independent of the time factor. The main algorithms are autoregression and moving average models and a combination thereof. These are statistical models and have nothing to do with classic machine learning.

Both models can be used when autocorrelation (see AUTOCORRELATION AND PARTIAL AUTOCORRELATION) is present in the time series data. In most cases it is at least possible to make a rough guess of tomorrow's data by knowing today's data. Often it's more like an educated guess because many weather parameters don't change rapidly. Both rely on past data to make predictions for the future.

It is important that the time series to model is stationary. If it is not, measures like transformation need to be taken before modelling. They can also only be used for a pandas series (or a numpy array), not for a full pandas dataframe. Except for one model every variable needs to be modelled by itself. It remains to be discussed whether this is always a good approach.

Autoregressive or AR Model

The basis for an autoregressive model is partial autocorrelation. It figures out whether yesterday's observation is significant for today's or whether the day before yesterday's observation has any relevance for today. Partial autocorrelation only looks at a single value at a time. This means that a simple autoregressive model also just looks at one previous value.

Since the weather data has been collected on a daily basis, today's weather would be predicted by only looking at yesterday's values and tomorrow's weather by looking at today's values. This is called an AR(1)-model. Again, I'm not going into the math here. Suffice to say that the model uses the previous value multiplied by a coefficient that is determined by the model using Python's `statsmodels` library. To this a constant is added that represents the uncertainty for the future and is called a shock value. In weather data this can be seen as the difference between the predicted data and the actual observations on the day. Obviously, the shock value is unpredictable for the future but "predicting" past values can give us an idea of how accurate the predictions are. Often there is also another constant added representing white noise.

Whether or not an AR(1) model is a good model depends a lot on the underlying data. Jumping back to autocorrelation we found out that the mean temperature is highly and significantly autocorrelated over a long period of more than one month. This is no longer true when looking at partial autocorrelation which found that only the previous observation is relevant. Is this case an AR(1) model is a good choice.

On the other hand the amount of rainfall was not even autocorrelated for just the previous day and just slightly less so for partial autocorrelation. So for predicting rainfall, an AR model is not at all useful.

For air pressure we see autocorrelation for up to two days but partial autocorrelation for only one day. This would mean that it's possible to make an educated guess about the future by looking at the past two days combined but not by looking just at the day before yesterday. Again, an AR(1) model seems indicated.

Moving Average or MA Model

When talking about SEASONALITY, TREND AND NOISE we already encountered moving averages to even out outliers and make trends visible. A moving average of the data takes an arbitrary number of datapoints and calculates the average of these for every datapoint where the condition applies. Obviously, a moving average for a decade cannot be calculated if there aren't enough datapoints available. Let's say we have 20 years worth of yearly data, the moving average for a decade = 10 years can only be calculated from the tenth year onward since all periods before this were less than 10 years. Since we only have 20 years in this example it might be more helpful to calculate the moving average for five years or half a decade. This decision depends

mostly on the data but can also be determined when thinking about stationarity as this might be one way to make data stationary although certainly not the most common.

A moving average model is based on autocorrelation since it considers an arbitrary number of past observations combined to account for unpredictable events happening during the time of observation. In the case of weather data this is much more robust when the data contains outliers from extreme weather events since they are evened out by taking the longer-term average. As far as the math goes this model is not using the observed values like the AR model but the residuals (or errors) between the calculated moving average and the actual observations. Otherwise the model is quite similar.

Looking at our data from above we find that we could build an moving average model for 44 days in the past (or possibly even slightly more) since all values up to then are considered correlated and significant.[4] However, this is time consuming and likely to introduce a lot of uncertainty compared to a smaller sample of the data. In this case 7 or 8 days seem reasonable because this is where the autocorrelation begins to level off. Using 7 or 8 lags this would represent the order of the model and is called an MA(7) or MA(8) model respectively.

For rainfall it's impossible to use an MA model because there is no significant autocorrelation between even yesterday's rainfall and today's.

Air pressure is significantly autocorrelated for two days so we might use an MA(2) model here. In this case it would be a good idea to check out both an MA(1) and MA(2) model to determine which is better because the significance is only just above 0.5 for the second day, meaning that there is a 50% chance that the air pressure two days ago is relevant for today's.

ARMA Model

The name already lets us make an educated guess: This kind of model combines the autoregressive model and the moving average model into one.

ARMA models can but don't have to use the same order for both model parts. Seeing that it might be better to use only one lag or the previous day when building a model for air pressure the orders for both would be 1 and the order for the ARMA model would be ARMA(1,1). On the other hand, mean temperature would see a huge difference between the orders and might look like this: ARMA(1,8). The first value always stands for AR and the second for MA. Note: An ARMA(1,0) model is the same as an AR(1) model and an ARMA(0,1) model is an MA(1) model.

Combining both models into an ARMA model is particularly useful if the data is persistent but has outliers. Using only autoregression outliers might distort the model beyond usefulness and looking only at the moving average might lose information on persistence. There is also a specialized model to consider:

ARMAX Model

[4] Note: The course modules differ about how to determine the orders visually just by looking at the acf and pacf functions. Some argue that this isn't an MA model at all because the acf just tails off without having a cut-off.

The ARMAX model is basically the same as the ARMA model but introduces another variable called the exogenous inputs or exogenous input terms. For this model we hike back to CORRELATIONS BETWEEN DIFFERENT MEASUREMENTS because ARMAX models make it possible to expand the univariate model by another variable or variables which are, preferably, highly correlated to the original variable. This variable does not have to be a numerical variable but might also represent the day of the week, thus taking into account that some time series behave differently on weekends, e.g. sales in the big city stores.

Building a model for mean temperatures we might also want to take relative humidity into account to improve the model although this is not as highly correlated at -0.48 as vapour pressure at 0.92. That is because vapour pressure depends almost solely on temperature and adds less value to the model than relative humidity, which depends on other factors like the actual amount of sunshine and cloud coverage as well.

Side Issue: ARIMA Model

All of the models above can only be used for stationary data. The ARIMA model is an integrated model that can also be used on non-stationary data like Covid-19 Case Numbers discussed above. AR and MA stand for autoregression and moving average as before while I is a new term and stands for integration. It defines how many steps are necessary to transform the non-stationary data into stationary. If the data is already stationary ARIMA(1,0,1) is the same as ARMA(1,1) with I = 0 telling us that no integration steps are needed to transform the data into stationarity.

I will not use this model here but take and explain the steps to transform the data as needed to keep the subject from becoming too complex and confusing.

Modelling in Practice

Mean Temperature

The `ARMA` class from the `statsmodels.tsa.arima_model` library is used to build a model here. In its simplest form this module can easily be used to build a model for one variable and it just takes this variable, in this case data from the column `Temperature Mean °C`, and the order for both methods: AR and MA. A frequency for the data can be provided if it isn't defined by the datetime index of the pandas series, otherwise this is inferred from the data. There is no need to transform the data in any way because it is already stationary. Once the model is defined it can be fitted. This takes just two lines of code:

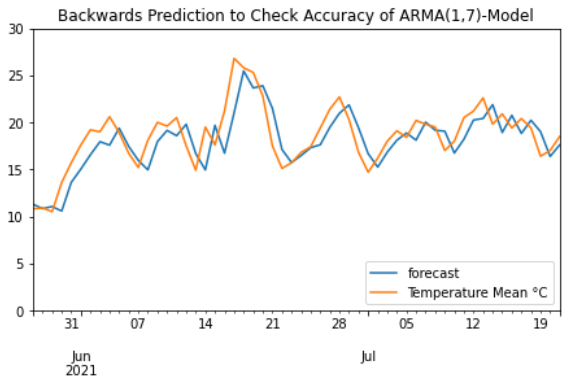
```
model = ARMA(df_54_21['Temperature Mean °C'], order=(1, 7), freq='D')
results = model.fit()
```

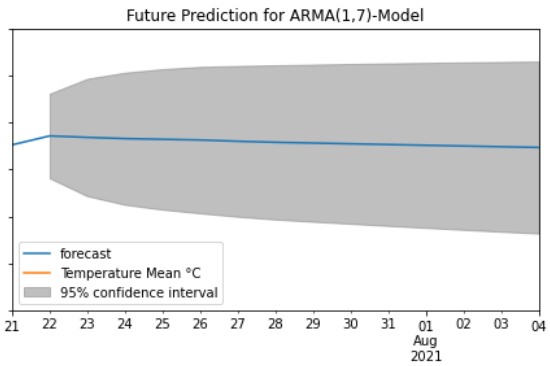
In this code example an ARMA(1,7)-model is built, corresponding to an AR(1)-combined with an MA(7)-model as discussed above. Once the model is fitted, a summary of its results can be printed. It consists of a lot of numbers!

```
                              ARMA Model Results
==============================================================================
Dep. Variable:     Temperature Mean °C   No. Observations:                24674
Model:                     ARMA(1, 7)   Log Likelihood              -55512.502
Method:                        css-mle   S.D. of innovations              2.295
```

```
Date:                Fri, 13 Aug 2021   AIC                         111045.005
Time:                        19:14:12   BIC                         111126.140
Sample:                  01-01-1954   HQIC                        111071.282
                       - 07-21-2021
==========================================================================================
                             coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------------------
const                      9.3729      0.604     15.519      0.000       8.189      10.557
ar.L1.Temperature Mean °C  0.9905      0.001    960.078      0.000       0.989       0.993
ma.L1.Temperature Mean °C -0.0281      0.006     -4.355      0.000      -0.041      -0.015
ma.L2.Temperature Mean °C -0.2388      0.006    -37.051      0.000      -0.251      -0.226
ma.L3.Temperature Mean °C -0.1401      0.007    -21.378      0.000      -0.153      -0.127
ma.L4.Temperature Mean °C -0.0757      0.007    -11.215      0.000      -0.089      -0.062
ma.L5.Temperature Mean °C -0.0578      0.007     -8.831      0.000      -0.071      -0.045
ma.L6.Temperature Mean °C -0.0343      0.006     -5.345      0.000      -0.047      -0.022
ma.L7.Temperature Mean °C -0.0318      0.006     -5.029      0.000      -0.044      -0.019
                                    Roots
=============================================================================
                  Real          Imaginary           Modulus         Frequency
-----------------------------------------------------------------------------
AR.1            1.0096           +0.0000j            1.0096            0.0000
MA.1            1.1585           -0.0000j            1.1585           -0.0000
MA.2            0.9695           -1.4683j            1.7595           -0.1571
MA.3            0.9695           +1.4683j            1.7595            0.1571
MA.4           -1.5146           -0.6760j            1.6586           -0.4332
MA.5           -1.5146           +0.6760j            1.6586            0.4332
MA.6           -0.5735           -1.6895j            1.7842           -0.3021
MA.7           -0.5735           +1.6895j            1.7842            0.3021
-----------------------------------------------------------------------------
```

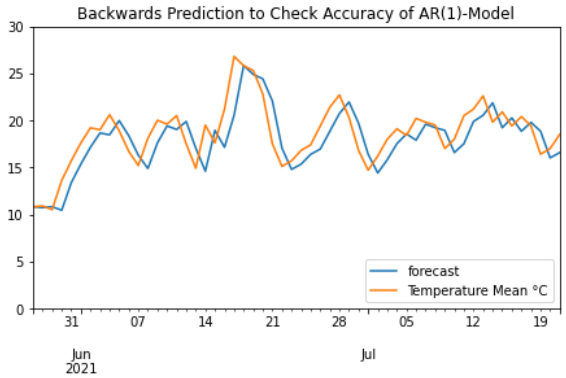
It's hard to draw any conclusions from these statistics. Luckily `statsmodels` provides a method to use on the fitted model both to make predictions into the future and the past. The latter gives us an idea of how well the model fits the data. While it is possible to just get predictions to work with later, for our purposes the predictions can be plotted straight away using the method `.plot_predict()`. In this case I've asked it to plot the predictions backwards for eight weeks or 56 days while the prediction into the future is only for the next two weeks or 14 days:

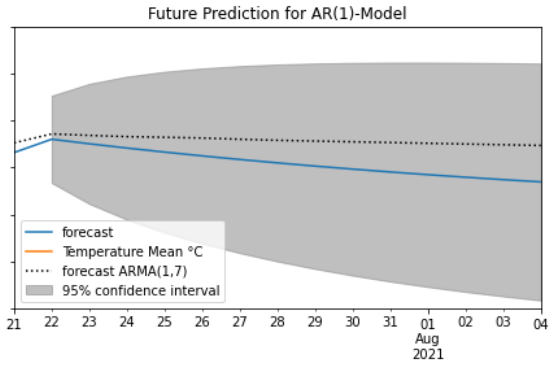




In general the backwards predictions fit the data well enough although they seem to be shifted a day or so into the future.

Let's see if this works any better just using an AR(1) model:

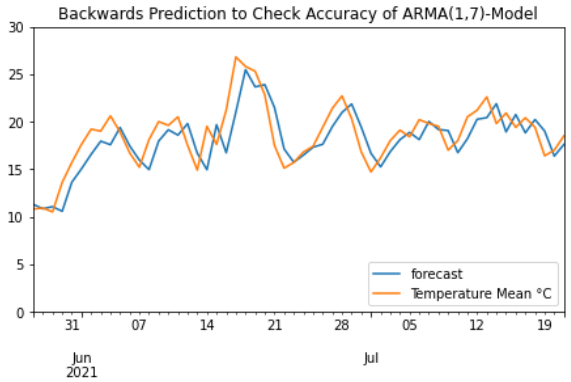


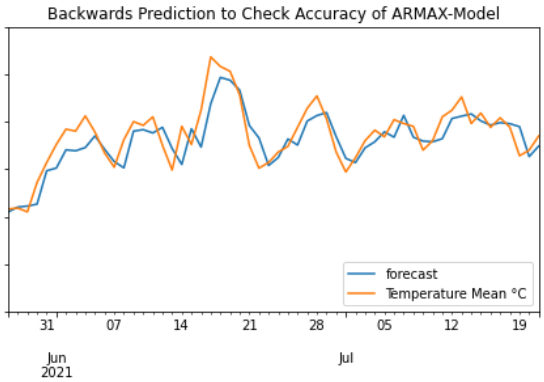


The line for the backwards predictions on the left seems a bit smoother than before, especially around June 18th, 2021, but there is a huge difference on the right when looking at the predictions for the future.

Neither model is able to predict even tomorrow's temperature within a very narrow 95% confidence interval. The confidence interval shows that the temperature could be up to five degrees higher or lower! After that the predicted temperature drops much faster for the AR(1) model than it did for the ARMA(1,7) model. To show this more clearly I've added the predictions from the latter model to the plot. The confidence interval defining the lower and upper bounds increase more rapidly as well but you'll have to compare the separate plots here.

This model is not so very useful but then temperature is by far not the only variable when it comes to weather. We found a high correlation between temperature and relative humidity, and an ARMAX model allows using the relative humidity as an exogenous variable. Since the ARMA(1,7)-model preformed better above it is used here as well, just using the parameter `exog=df_54_21['Relative Humidity Mean %']` to extend the model. Unfortunately `statsmodels` does not allow predictions into the future without providing data for the relative humidity during this period. I don't have that, obviously, so I've opted for plotting backwards predictions for the "pure" ARMA(1,7)-model and the extended one side by side to save on scrolling.





This time the curve is a lot smoother after taking relative humidity into account. Many of the peaks are smoothed out but the predictions still seem to be shifted into the future by a day and the deviation from the actual observations also looks to be bigger than in the original model. In some cases this might be better, depending on the goal, but since I have no way to compare the projected accuracy of predictions without getting more data, this is something I cannot judge here.

For real world predictions I'd prefer a model that does not smooth out the peaks but then I wouldn't be able to use an ARMAX model in the first place because I'd need a crystal sphere to provide the data for future relative humidity the model needs to make predictions for the temperature. Predicting the future values for relative humidity and using this would, of course, be possible but introduce a lot more uncertainty than the original model already has (see 95% confidence interval).

Air Pressure

We found that Air Pressure is highly autocorrelated for the first two days but only for the following day for partial autocorrelation. This points to an ARMA(1,2) model.

First though it's necessary to test the data for stationarity which is again done using the Augmented Dickey-Fuller Test with just one line of code:

```
adfuller(df_54_21['Pressure Mean hpa'])
```

The relevant metrics of the result confirms that air pressure is a stationary series:

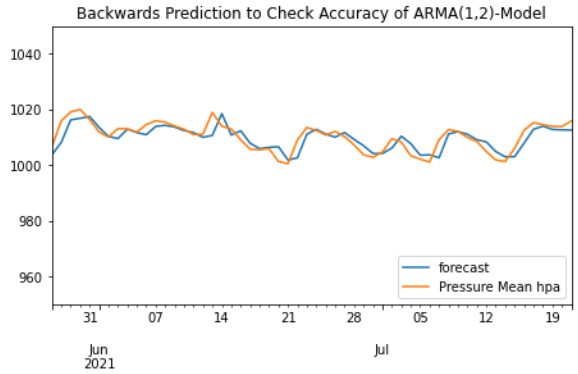
```
adf test statistic: -52.881571161956806
p-value: 0.0
critical values: {'1%': -3.430615098522581,
                  '5%': -2.861657165451611,
                  '10%': -2.5668323637561}
```

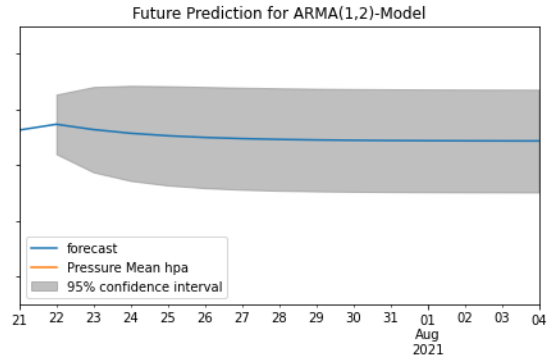
There is no high correlation with any of the other columns in the data so the ARMA-model can be used rather than the extended ARMAX-model.

The model results summary is less crowded than the summary for the temperature but there are still a lot of numbers:

```
ARMA Model Results
==============================================================================
Dep. Variable:      Pressure Mean hpa   No. Observations:                24674
Model:                     ARMA(1, 2)   Log Likelihood              -76913.926
Method:                       css-mle   S.D. of innovations              5.464
Date:                Fri, 13 Aug 2021   AIC                         153837.852
Time:                        21:04:30   BIC                         153878.419
Sample:                    01-01-1954   HQIC                        153850.990
                         - 07-21-2021
==============================================================================
                                coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const                     1008.6514      0.147   6865.620      0.000    1008.363    1008.939
ar.L1.Pressure Mean hpa      0.6803      0.008     82.372      0.000       0.664       0.696
ma.L1.Pressure Mean hpa      0.3426      0.010     32.793      0.000       0.322       0.363
ma.L2.Pressure Mean hpa      0.0077      0.009      0.827      0.408      -0.011       0.026
                                    Roots
=============================================================================
                  Real          Imaginary           Modulus         Frequency
-----------------------------------------------------------------------------
AR.1            1.4699           +0.0000j            1.4699            0.0000
MA.1           -3.1423           +0.0000j            3.1423            0.5000
MA.2          -41.1286           +0.0000j           41.1286            0.5000
-----------------------------------------------------------------------------
```

Plotting the backwards and future predictions show a better fit for the backwards predictions than we have seen with temperature while still being shifted. The 95% confidence interval corridor is more like a hosepipe or sausage casing than it was for temperature but we need to keep in mind that there is also less variability in air pressure in the short-term, extreme weather events excepted.

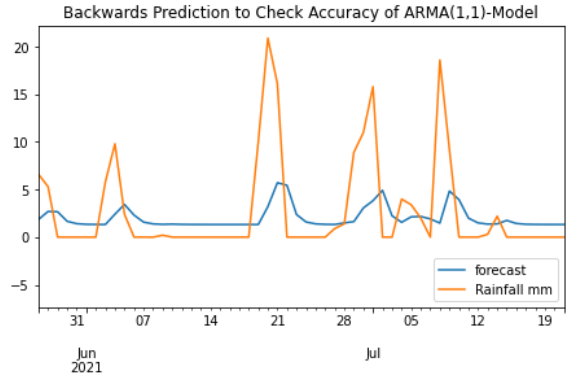


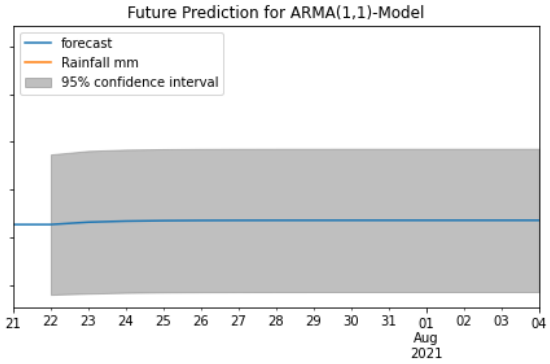


It is interesting to observe that the lines for both the predictions of temperature and the prediction for air pressure have a downwards trend. This is something we often observe in nature during the summer. It's very common for the temperature to go down when air pressure goes down in summer. In winter it is more often the other way around because high pressure is often associated with very cold temperature and clear, bitter nights. I doubt we'll see this in the data though. Looking back at the full line plot when checking for missing data does not display many of these bitterly cold winters and they are not distributed evenly across the years.

Counterexample: Rainfall

This is one column where these models cannot be used even though the data itself is stationary. There is neither significant autocorrelation nor partial autocorrelation for `Rainfall mm`, thus suggesting that today's or even the last couple of days' rainfall has no significance on tomorrow's. This can be seen very clearly in the plots:





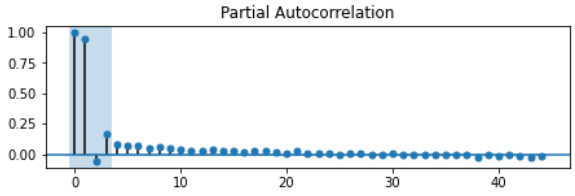
The backwards predictions hardly even begin to fit the observations beyond the fact there are small peaks near where a lot of rainfall came down. The model even found that there was some rainfall every day when in reality there was none. The future predictions are basically a continuation of the flat line of the last couple of days following a very small rise. Although there is absolutely no negative rainfall the confidence interval is happy to include negative values within its bounds. The span between the lowest and the highest possible amount of rain is huge and constant at all times in the future. I did a quick check using either an AR(1)- or an MA(1)-model only rather than the combined one but this doesn't change the results very much.

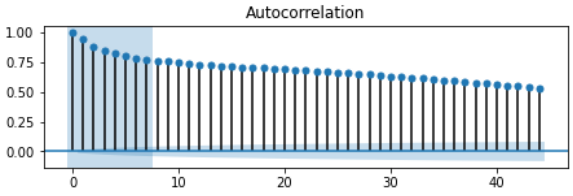
Finding and Revising the Best Model(s)

Grid Search

If you are familiar with classic machine learning you will have come across methods like GridSearchCV to determine the best parameters for the machine learning model. `statsmodels` does not provide such a function but as ARMA models only call for two parameters doing a grid search can be achieved with a simple nested for-loop.

The two parameters to test for are the order for the AR model and the order for the MA model. For the column `Temperature Mean °C` the models built so far used ARMA(1,7) and AR(1) with the latter giving a wider margin of error for the future predictions. Let's revisit the autocorrelation plots where I've marked the time lags I'm most interested in:





Partial correlation, which determines the order for the AR model, has discreet values up to three lags and then tails off. Only the first time lag, 1, is highly correlated but the second being correlated negatively and then jumping back to positive is interesting so these are the lags I'm looking at for modelling.

Autocorrelation for determining the MA model order tails off right from the beginning but all values are highly correlated and statistically significant. I've only marked the first seven lags though because it is doubtful whether tomorrow's weather is dependent on anything that's more than a week ago. Personally I think that the grid search will return a lower number.

I have done a grid search for AR models orders from 0 through 3 and for the MA model orders from 0 through 7. Doing such a grid search means building a model and fitting it combining each of the order parameters, saving each of the results for later analysis. This can take a long time. Using the orders from ARMA(0,0) through ARMA(3,7) took five minutes on my machine.

Once the results are in, the relevant statistics in the results summary are AIC (Akaike Information Criterion) and BIC (Baysian Information Criterion). If you scroll back a couple of pages you'll find these in the summary statistics in the top right. Neither number is meaningful in and of itself. They are only used when comparing models.

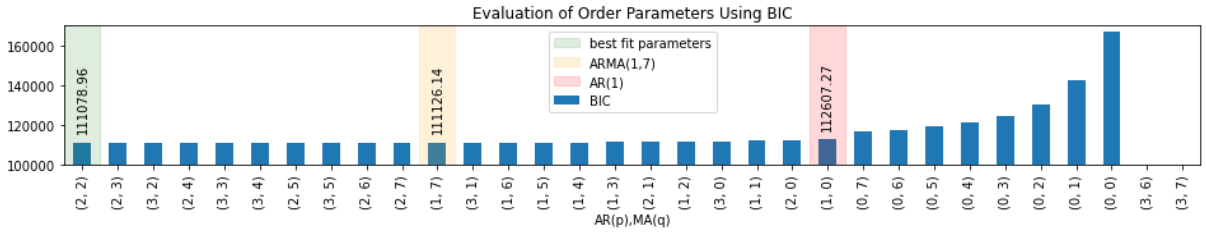
Both are metrics to measure how well each model performs. Lower numbers indicate a better model. Which of the metrics to choose to evaluate the models depends on the goal of the model. The AIC is lowest for models that make the best predictions. The BIC is lowest for models that fit the data best. Since I haven't been working with train-test-splitting the dataset I'm mostly interested in which model fits the data best.

Two of the possible combinations of orders threw a warning so I've marked them as NaN-values. These were AR(p) = 3 and MA(q) = 6 respectively 7. The following table

shows the first and last lines of the two metrics, sorted by BIC ascending:

AR(p)	MA(p)	AIC	**BIC**
2	2	111030.275870	**111078.956902**
2	3	111031.297842	**111088.092380**
...			
0	1	142700.377195	**142724.717711**
0	0	167295.954760	**167312.181770**

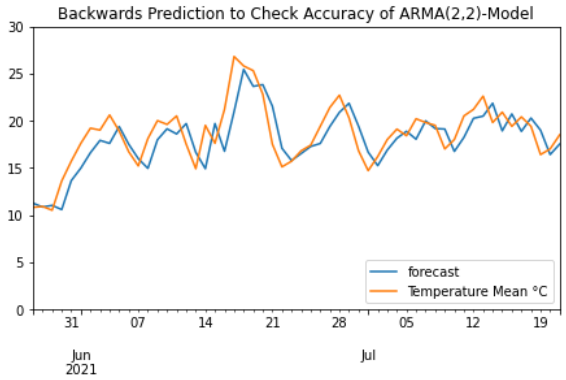
There really isn't a large difference between the best and the second best model. Since the saying is that a picture is worth a thousand words let's visualize how all the possible models compare.

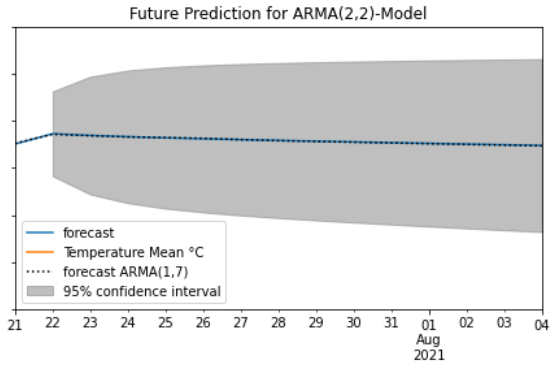


The models used before are marked in yellow and red respectively (the second and third rectangle if you are reading a b/w-printed version). There still isn't a huge difference between all the models until we come to the pure MA or ARMA(0,q) models where a line following the BIC values would look more like a parabola than a straight upward line.

Modelling the Best Order Parameters

As could be expected with the BIC numbers being very close there isn't much if any difference between the visually determined ARMA(1,7) model and the ARMA(2,2) model determined by the grid search.





Comparing the plots to the ARMA(1,7) plots above I can't see any difference for the backwards predictions. Plotting the future predictions from the ARMA(1,7) on top of the predictions from the ARMA(2,2) doesn't show any difference either as is evidenced by the dotted line matching the blue line of the forecast almost exactly. It's hard to say by just looking at the plots whether the confidence intervals match as well but unfortunately there is no method to get the exact numbers, at least that I know of.

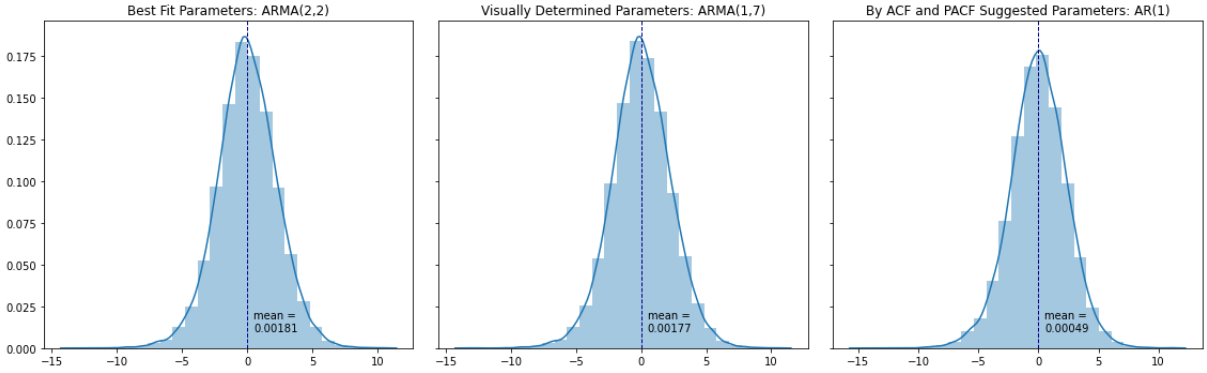
Reviewing the Result Statistics

A common metric to evaluate a model's performance is taking the residuals of the training data: the difference between the predictions and the real observations.

These residuals can be accessed by using `.resid` on the results. This results in another time series for the whole period of time just giving the value of how much the predicted temperature differed from the actual temperature on that day:

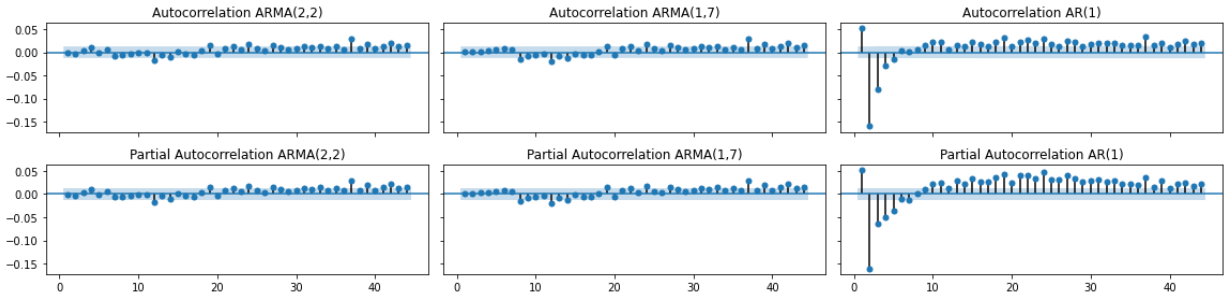
```
MESS_DATUM
1954-01-01   -12.872932
1954-01-02    -0.617203
 [...]
2021-07-20     0.599034
2021-07-21     0.982477
```

Scanning the values doesn't give any idea of how good or bad the model fits the data. There is obviously a huge error for the first value while the others are less than plus or minus one for this small sample. Rather than looking at just the sample it's better to plot the residuals as a histogram to get an overview of the distribution. I've done so for all three models:



All residuals should be centred on zero and show a Gaussian distribution (bell shaped curve). This is complied with in all three cases although the mean isn't exactly zero for either. Nonetheless, the difference is so small it is negligible. The curves are approaching a normal distribution as well with the left histogram coming closes and the right being the most but still only slightly skewed to the left, i.e. trailing off more to the left than to the right.

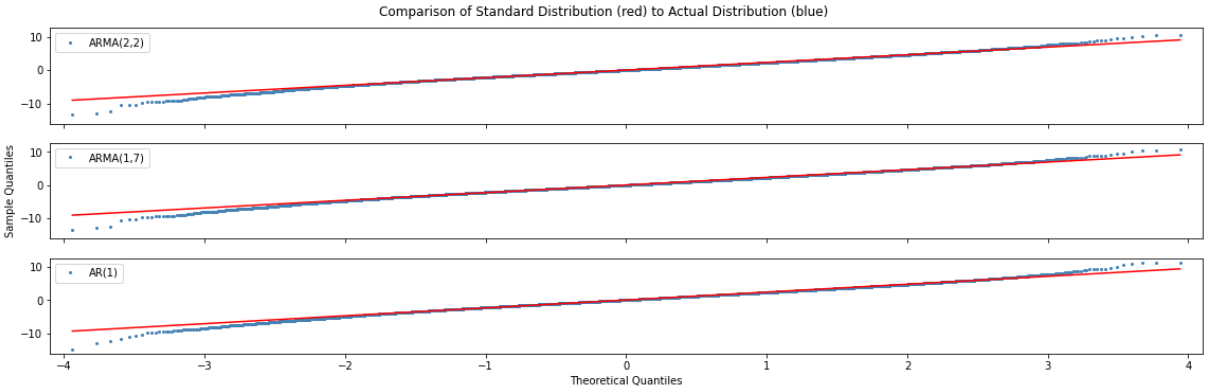
There should also be roughly no (partial) autocorrelation between the residuals since these should represent white noise rather than showing any patterns. Plotting those easily lets us reject the AR(1) model as there is distinct (partial) autocorrelation. Please note that these plots have left out lag 0 which is always 1 in order to improve readability of the plots:



Again we don't observe a lot of difference between the best fit model and the visually determined one.

A last way to review whether the model is a good fit is by comparing the distribution of the values to the standard distribution directly by plotting a q-q-plot (quantile-

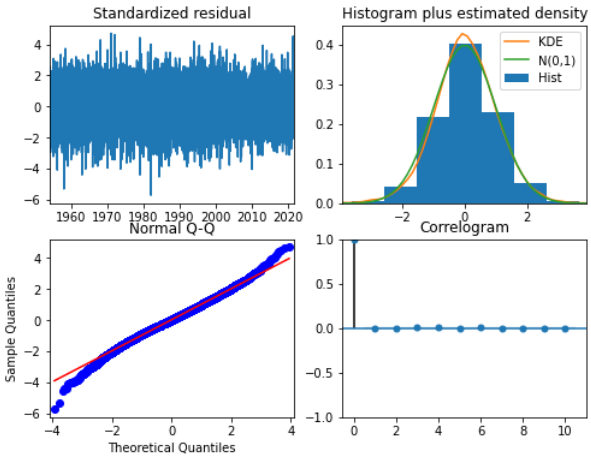
quantile-plot). This is done by using the `sm.qqplot`-function of the `statsmodels api` on the residuals.



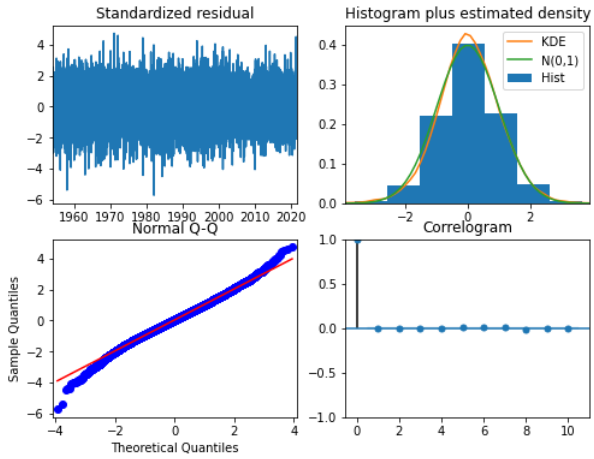
Here, too, there is no marked difference for the top two plots. For the third plot the tails on the left and right are only just going down respectively up slightly more, making this a worse model than the other two.

The ARMA class of `statsmodels` is the simplest class to use for time series models when the data is stationary. A more complex but also more versatile class is the SARIMAX class, which will be discussed in the next chapter. For now let's just note that instead of having to plot each of the metrics by hand it provides the method `.plot_diagnostics()` that does this for any given fitted model results.

Diagnostics for ARMA(2,2)



Diagnostics for ARMA(1,7)



Only being able to plot diagnostics for one fitted model result at a time makes it more difficult to compare them though. In this case I can only see a slight difference between the correlograms but I wouldn't know how to interpret this when having to choose either of the models. It looks very much like there is no correlation at all in the residuals. Only the point at 8 on the x-axis in the right plot is slightly below the line but looks like it would still be inside the light blue area marked on the plots on the previous page. I think I would tend more towards the second model because the histogram looks a bit more even but there doesn't seem to be any difference between the bell curves for the actual distribution KDE and the normal distribution N(0,1). Yet we know already that the second model was not the best fit.

Weather Data is Seasonal

SARIMAX Model

So far, all models have disregarded the fact that weather data is seasonal. With seasonal we don't mean the four seasons summer, autumn, winter and spring but the distribution of values throughout a discreet period of time, here the year. For mean temperatures a "season" is a whole year with a cycle that repeats each year: the cold months spanning late autumn through early spring and the warm months starting in late spring and cooling off in early autumn as seen in MEAN TEMPERATURE.

To model seasonal time series data `statsmodels` provides the class SARIMAX. This acronym stands for **S**easonal **A**uto**r**egressive **I**ntegrated **M**oving **A**verage **E**xogenous Model, which really is quite a mouthful. Roughly speaking it is an ARMAX Model that also accounts for seasonality and the same requirements apply. It can also be used to model non-stationary data like the COVID-19 Case Numbers by providing a value for I but this probably won't ever play a role for weather data.

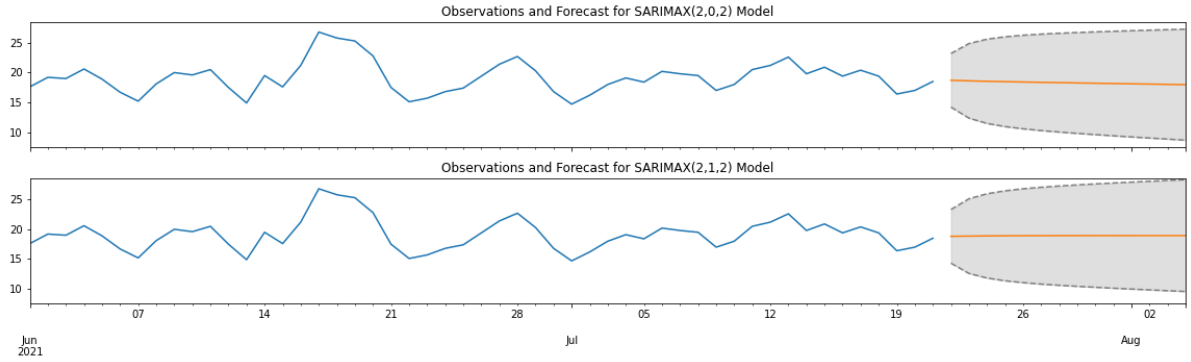
Fitting the ARMA(2,2) or SARIMAX(2,0,2) model used above for `Temperature Mean °C` gives almost the same results in the summary as before:

```
                               SARIMAX Results
==============================================================================
Dep. Variable:     Temperature Mean °C   No. Observations:                24674
Model:                SARIMAX(2, 0, 2)   Log Likelihood             -55544.787
Date:                 Sun, 15 Aug 2021   AIC                         111099.574
Time:                         14:28:24   BIC                         111140.142
Sample:                     01-01-1954   HQIC                        111112.713
                          - 07-21-2021
Covariance Type:                   opg
==============================================================================
[...]
```

There was no transformation needed to make the data stationary but the stationarity could be improved by taking the differences in temperature from day to day rather than the absolute values. With changing just one number in the code it's easy to check whether this improves the model as well. These are the results for the SARIMAX(2,**1**,2) model:

```
                               SARIMAX Results
==============================================================================
Dep. Variable:     Temperature Mean °C   No. Observations:                24674
Model:                SARIMAX(2, 1, 2)   Log Likelihood             -55555.627
Date:                 Sun, 15 Aug 2021   AIC                         111121.254
Time:                         20:34:51   BIC                         111161.821
Sample:                     01-01-1954   HQIC                        111134.392
                          - 07-21-2021
Covariance Type:                   opg
==============================================================================
[...]
```

The bottom line is that there is no marked improvement. This can be seen by comparing the AIC- and BIC-values of both models. The lower the values the better the model. However, at just around 21.7 for each there is virtually no difference between the models. Visualising both models by plotting the predictions, or forecasted values, for both also confirms that there is no marked difference. It might even been argued that taking the differences is worse because the forecast is simply a horizontal line with no trend at all without looking at the AIC or BIC at all:

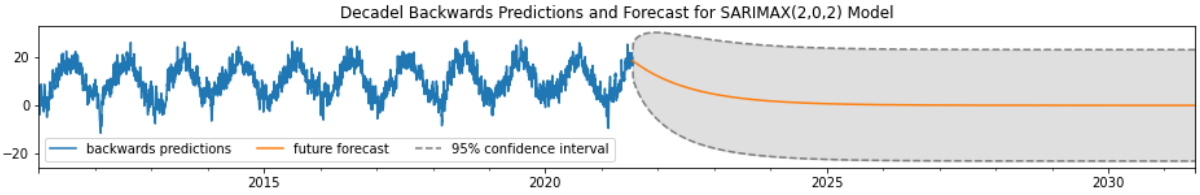


If you prefer to see hard numbers, here are the minimum, mean and maximum values of the predictions for the temperature, the minimum lower and maximum upper bounds of the confidence intervals as well as the spans of each:

model	min temp	mean temp	max temp	lower	upper	temp_span	conf_int_span
SARIMAX(2,0,2)	17.973895	18.302487	18.704235	8.661575	27.286215	0.730340	18.624641
SARIMAX(2,1,2)	18.819373	18.920357	18.946051	9.562252	28.329850	0.126677	18.767598

Both models have been used for forecasting just a fortnight into the future. These short-term forecasts don't need to take seasonality into account. When talking about climate change and how the mean temperature changes over the long-term seasonality can play a vital role.

Looking at the backwards predictions for the SARIMAX(2,0,2) model for the past decade captures the seasonality well but the forecast for the coming decade does not and also shows a distinct downward trend which doesn't fit either the slight upward trend found in SEASONALITY, TREND AND NOISE nor the trend we keep reading and hearing about in the news.



This is where the SARIMAX model comes in. In addition to modelling the data to the training data without knowing anything about seasonality (the model just follows the distribution of the data without analysing it for trends or seasonality) an expanded model can be built that is informed about seasonality. It has to be provided with the length of each season so it can be fitted to data that might or might not be seasonal. Based on the observations for the seasons an ARMA or ARIMA model is built and merged with the general ARMA or ARIMA model. An ARIMA model is only needed if the data is not stationary but this needs to be checked for seasonal data as well as for the actual observations if resampling of the data is necessary to capture the seasons. The rules for the seasonal AR(I)MA models are the same as for non-seasonal data. This means that all the above-mentioned methods for ascertaining autocorrelation and partial autocorrelation as well as stationarity can be used for the seasonally resampled data.

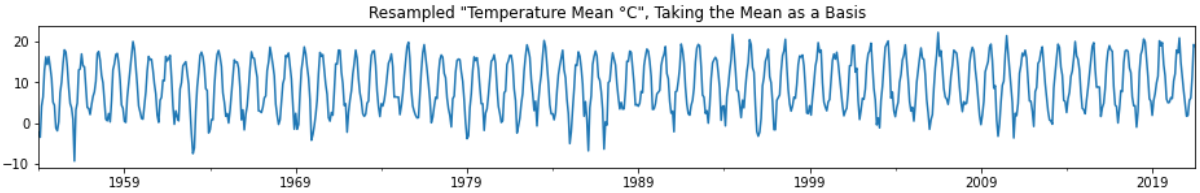
Getting Seasonal

One season in weather data is a whole year. Here we have daily measurements so one season is 365 days long by definition. When forecasting long-term temperature trends this means that the forecast will be off by almost three weeks by the end of the century. A fixed length of 365 days completely misses out on leap years every four years, and this accumulates. The SARIMAX model only accepts one parameter for the seasonal length so the weather data needs to be transformed into the next nearest time length that is static throughout the years: months. Depending on the variable this can be done by taking the mean per month, the median (this is more robust with outliers) or the sum.

As seen in OUTLIERS the data for `Temperature Mean °C` hasn't got that many outliers, and all are in the negative degrees. Therefore the mean temperature is about 0.2 degrees lower than the median:

```
overall mean "Temperature Mean °C": 9.382540325849066
overall median "Temperature Mean °C": 9.6
```

Since there were only a few really cold winter months this small difference is not having much of an effect on the resampled data so it makes sense to stay with the resampling using the mean and not lose data that might be important to the model. The resampled data looks like this:



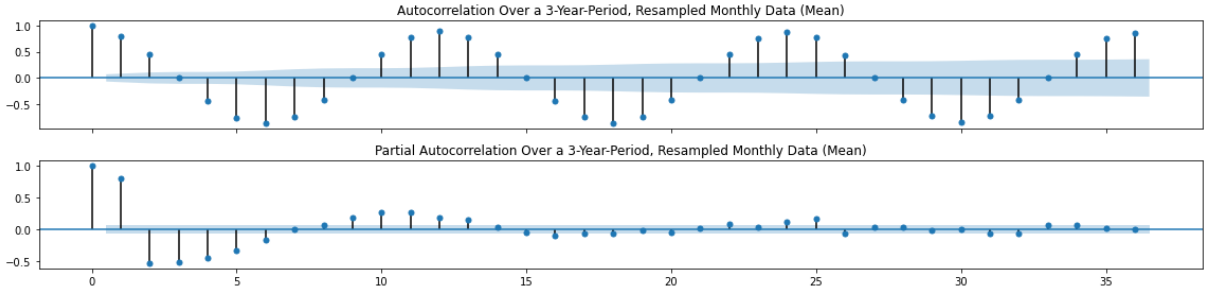
There is no need to check the data for seasonality. This is already obvious from this plot. It is interesting to note that both winters and summers were less extreme in the mid-1970s. Overall the amplitude of the curve doesn't increase either. Colder winters are not necessarily followed by cooler summers and vice versa. This might make it more difficult for the model to predict the future.

The resampled data needs to be checked for stationarity though. Running the Augmented Dickey-Fuller Test returns the follow results:

```
adf test statistic: -4.137969795192533
p-value: 0.0008376447876419999
critical values: {'1%': -3.4386652124595614,
                  '5%': -2.865210127510208,
                  '10%': -2.5687243221835088}
```

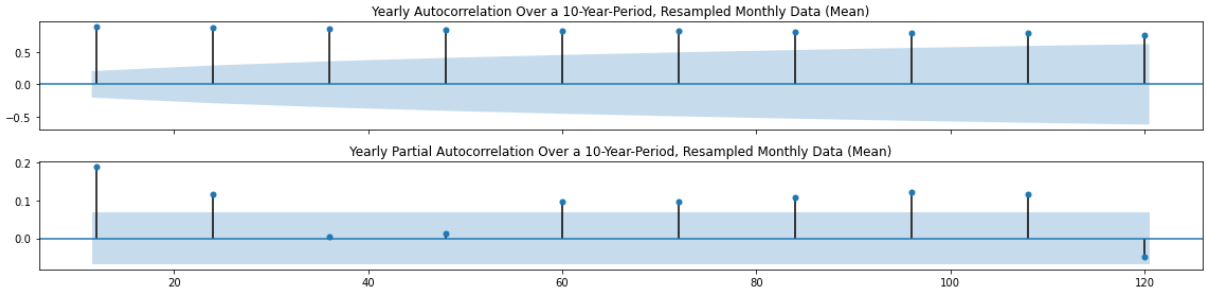
Yes, the data is still stationary but not markedly so. The adf test statistic isn't much below the 1% threshold of the critical values. While the p-value is clearly below 0.05 it is much less so than it was without resampling: `6.430163441715136e-22`. When building the model it might become necessary to transform the data by taking the differences between the months.

To determine the AR and MA model orders information about both autocorrelation and partial autocorrelation is needed. But this time the plots provide little help.



Both plots clearly show seasonality but both are also trailing off with the amplitudes getting smaller faster for partial autocorrelation but still showing seasonality.

For seasonality the orders also need to be determined. To do so the same acf and pacf-functions are used but this time for just the lags that correspond to the length of one season, here 12, 24, 36 months and so on.



These plots are not much easier to interpret than previous ones although the bottom plot might point to a simple AR(1) or AR(2) model to capture seasonal data. Please note that while the x-axis is still labelled by the numbers of months the model order corresponds to this number divided by 12, which is 12/12 = 1 and 24/12 = 2, after which the values drop off sharply. On the other hand it might be that this diagram would also trail off when looking at a much longer timescale. The fact that values from 60 up to 112 are statistically significant again and looking a bit like a wave provides some indication for that.

Using the same method of building a model for each combination and comparing the results of each as was done in the last chapter is time-consuming and error-prone since six parameters need to be found:

- general model: AR, D and MA, where D stands for the number of times transforming the time series by taking the difference needs to per performed to get the optimal stationary data.
- seasonal model: AR, D and MA, with D having the same definition as for the general model.

I did it anyway, confining myself to just three values for each. In order to keep complexity to a minimum I first looped over the order values for the general model. This took just under one minute on my machine. In a second step I did the same for the seasonal model using the best result from the first which was identical for both metrics. This took long enough to go shopping, cook dinner or binge-watch netflix even with having static values for the general model: 55 minutes. Here are the results of both runs:

general model orders by AIC and BIC

AR(p)	D(d)	MA(q)	AIC	BIC
2	1	3	3401.627116	3429.809321
3	1	3	3404.934170	3437.813410
3	0	3	3453.583000	3486.470877
AR(p)	**D(d)**	**MA(q)**	**AIC**	**BIC**
2	1	3	3401.627116	3429.809321
3	1	3	3404.934170	3437.813410
3	0	3	3453.583000	3486.470877

seasonal model orders by AIC and BIC

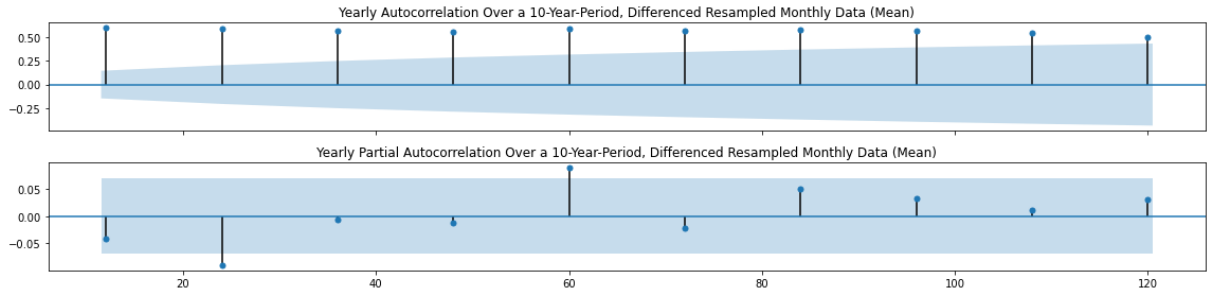
AR(P)	D(D)	MA(Q)	AIC	BIC
0	1	1	3342.125471	3374.900231
1	1	2	3343.483434	3385.622411
0	1	2	3343.547443	3381.004312
AR(P)	**D(D)**	**MA(Q)**	**AIC**	**BIC**
0	1	1	3342.125471	3374.900231
0	1	2	3343.547443	3381.004312
1	1	1	3343.573557	3381.030426

Both threw a lot of warnings, making the results rather unreliable. In fact almost every model order for the seasonal model threw convergence warnings and a couple were skipped altogether after they threw errors. This may well mean that even a model that is designed to deal with seasonal data can't make a good job of unpredictable data like weather measurements.

The favoured model from both runs looks like this:

```
SARIMAX(data, order=(2,1,3), seasonal_order=(0,1,1,12))
```

In both cases the for-loop found that a model with differenced data works better than the monthly mean by itself, as seen by the 1 as the second parameter each. Neither of these respective ARMA model orders is supported by either acf and pacf-plots even after taking the difference and plotting directly for that. The plots for the monthly lags stays pretty much the same while the plots for the yearly lags strongly suggests an AR(2) or AR(5) model, not an MA(1) model:



After building and fitting the model the model statistics support that this isn't a good model:

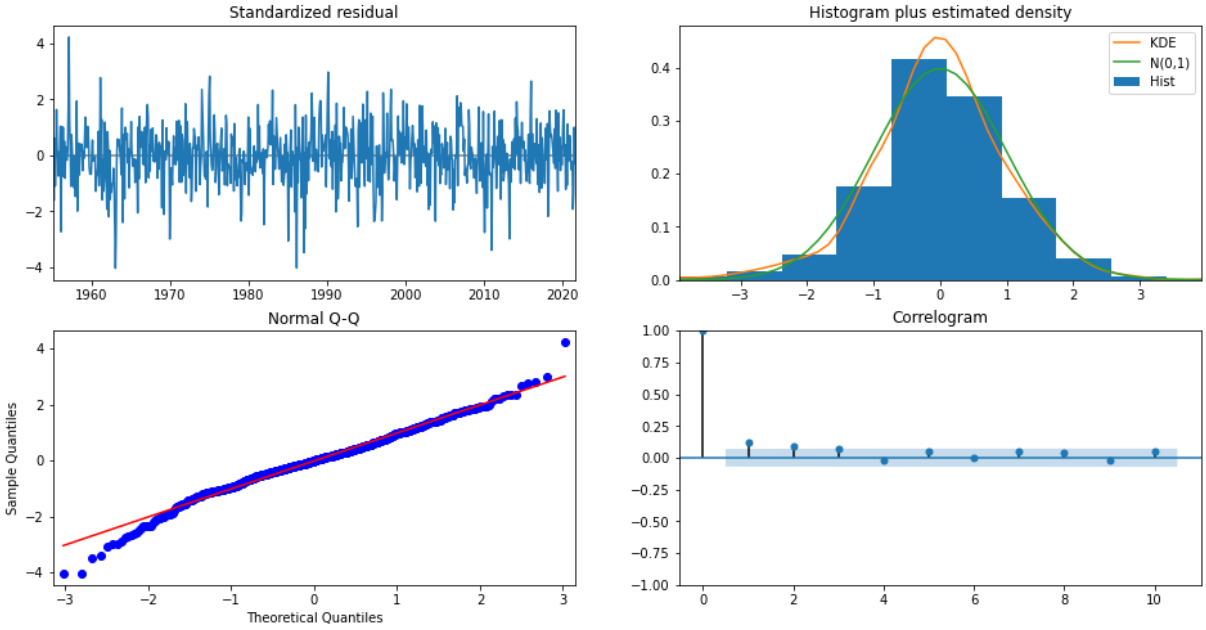
```
                                     SARIMAX Results
==========================================================================================
Dep. Variable:                  Temperature Mean °C   No. Observations:                  811
Model:             SARIMAX(2, 1, 3)x(0, 1, [1], 12)   Log Likelihood               -1676.670
Date:                              Wed, 18 Aug 2021   AIC                           3369.340
Time:                                     14:14:33   BIC                           3406.796
Sample:                                 01-31-1954   HQIC                          3383.731
- 07-31-2021                                         Covariance Type:
opg
==============================================================================
[...]
===================================================================================
Ljung-Box (Q):                       69.55   Jarque-Bera (JB):                 61.91
Prob(Q):                              0.00   Prob(JB):                          0.00
Heteroskedasticity (H):               0.85   Skew:                             -0.24
Prob(H) (two-sided):                  0.19   Kurtosis:                          4.28
===================================================================================
Warnings: [1] Covariance matrix calculated using the outer product of gradients (complex-
step).
```

The middle part of statistics is missing here as it has no significance for the interpretation. The main values are

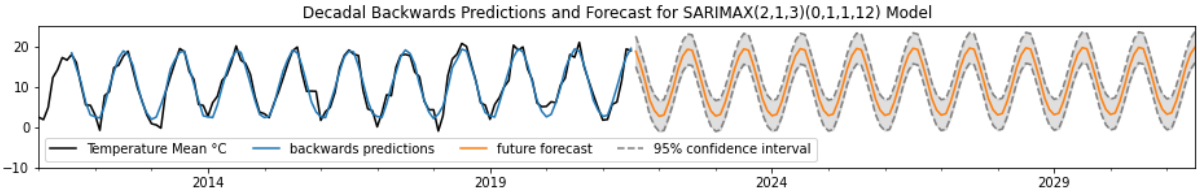
- `Prob(Q)` to support or reject the null hypothesis that there are no correlations in the residuals.

- `Prod(JB)` to support or reject the null hypothesis that the residuals are normally distributed.

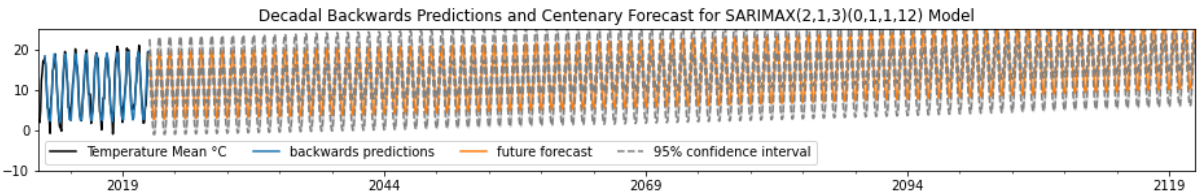
If either of these values is less than 0.05 the corresponding null hypothesis has to be rejected. In this case both have to be rejected, meaning that there are correlations in the residuals and that the residuals are not normally distributed. The plotted diagnostics support this:



Using this model to actually backwards predict and then make a prediction for the future doesn't work too badly and much better than the ARMA(2,2) model did:



It has to be noted that the backwards predictions don't capture the colder winters and the warmer summers. Overall the plot for the predictions looks quite uniform and so does the future forecast. The latter changes drastically when forecasting for a whole century, now supporting both the observed trend in the past projected to the future as well as what a lot of studies have found:



Taking the mean temperature for every year in the forecast it will rise from 10.8 °C in 2022 to 18.3 °C in 2121. A rise of 7.5 degrees does seem extreme, compared to worst-case predictions in the media. However, this is the best this model can do without further tweaking and taking more variables into account.

Returning to the simpler ARMA(2,2) model and expanding this by seasonality returns more realistic results with a yearly mean temperature of 10.6 °C in 2022 and 14.1 °C in 2121. Looking at the model statistics also tells us that this is, in fact, the better model.

model	SARIMAX(2,1,3)(0,1,1)$_{12}$	SARIMAX(2,0,2)(0,1,1)$_{12}$
AIC	3369.340	3333.394
BIC	3406.796	3366.177

Both the AIC and the BIC are lower and for the ARMA(2,2) model there are no correlations in the residuals and while the residuals are still not normally distributed they are more so than before. I think, we can take it for granted that using for-loops to find the optimal parameters in not our best bet in this situation!

Finding the Best Parameters?!

Data Scientists working with complex time series data and SARIMAX models would have to take many breaks and lose a lot of time if they kept using for-loops every time the best parameters needed to be found. So, a package called `pmdarima` was developed and can be used. It was a pain in the neck to install in my environment because it found incompatibilities with packages needed in previous courses. In the end I had to use a different environment, and then I was really disappointed in the results. There was very little introduction to the package and its capabilities in the course so I opted to try four different approaches:

1. Just running the method on the monthly resampled data without adding any parameters at all.
   ```
   ARIMA(order=(2, 1, 5), scoring_args={}, suppress_warnings=True,
   with_intercept=False)
   ```
2. Doing the same as for 1. but without additional differencing:
   ```
   ARIMA(order=(5, 0, 1), scoring_args={}, suppress_warnings=True)
   ```
3. Running the method after adding the parameter seasonal=True and providing the length of the season:
   ```
   ARIMA(order=(2, 1, 0), scoring_args={}, seasonal_order=(1, 0, 0, 12),
   suppress_warnings=True, with_intercept=False)
   ```
4. Doing the same as for 1. but without additional differencing:
   ```
   ARIMA(order=(1, 0, 1), scoring_args={}, seasonal_order=(2, 0, 0, 12),
   suppress_warnings=True)
   ```

Fitting a model for each of the "best" order parameters and printing the summaries gave me some really disappointing results. All of the models performed worse than the SARIMAX(2,0,2)(0,1,1)$_{12}$ model used in the previous steps. They even performed worse than the model derived from the nested for-loop!

Best order parameters (for-loop): SARIMAX(2,1,3)(0,1,1)$_{12}$	AIC 3369.340 BIC 3406.796 Prob(Q): 0.00 Prob(JB): 0.00
Order parameters based on ARMA(2,2): SARIMAX(2,0,2)(0,1,1)$_{12}$	AIC 3333.394 BIC 3366.177 Prob(Q): 0.97 Prob(JB): 0.00
SARIMAX(2, 1, 5) Note: This model failed to converge.	AIC 3379.894 BIC 3417.471 Prob(Q): 0.88

	Prob(JB): 0.00
SARIMAX(5, 0, 1)	AIC 3740.436 BIC 3773.324 Prob(Q): 0.00 Prob(JB): 0.00
$SARIMAX(2, 1, 0)(1, 0, 0)_{12}$	AIC 4010.184 BIC 4028.972 Prob(Q): 0.22 Prob(JB): 0.00
$SARIMAX(1, 0, 1)(2, 0, 0)_{12}$	AIC 3695.122 BIC 3718.613 Prob(Q): 0.90 Prob(JB): 0.00

The original best model ARMA(2,2) expanded by seasonality is not only the best model when looking at the AIC and BIC numbers which confirm that this model has the best backwards predictions and future forecasts but it also has the best distribution of residuals. There are no correlations between the residuals, which means that the residuals are just white noise. The residuals are not normally distributed but when plotting them they are much closer to being normally distributed than any of the other models. So it may be concluded that there either need to be a whole lot more parameters specified when using the `pmdarima` package or that there *is* no better model than the one more or less arrived at by applying some logic and coincidence!

There are a whole lot of parameters that can be set for this method of the `pmdarima` package but documentation is scarce and well beyond the scope of this paper. If you are interested have a look at their website:
https://alkaline-ml.com/pmdarima/modules/generated/pmdarima.arima.auto_arima.html
Some additional information is provided here:
https://alkaline-ml.com/pmdarima/tips_and_tricks.html#tips-and-tricks
but I did not find it immediately helpful. Perhaps someone will offer a dedicated course for this module sometime in the future.

Classic Machine Learning

Time Series data is usually used to predict values in the future but it is actually not restricted to just that. Provided the requirements of the respective model are met time series data can be used for modelling just like any other data can.

For the purpose of this paper let's say that for a discreet number of days temperature wasn't recorded due to the sensor being out of order. All other weather data has been recorded on these days so is there a chance a machine learning model might be used to find out at least what the mean temperatures should have been? It turns out there is.

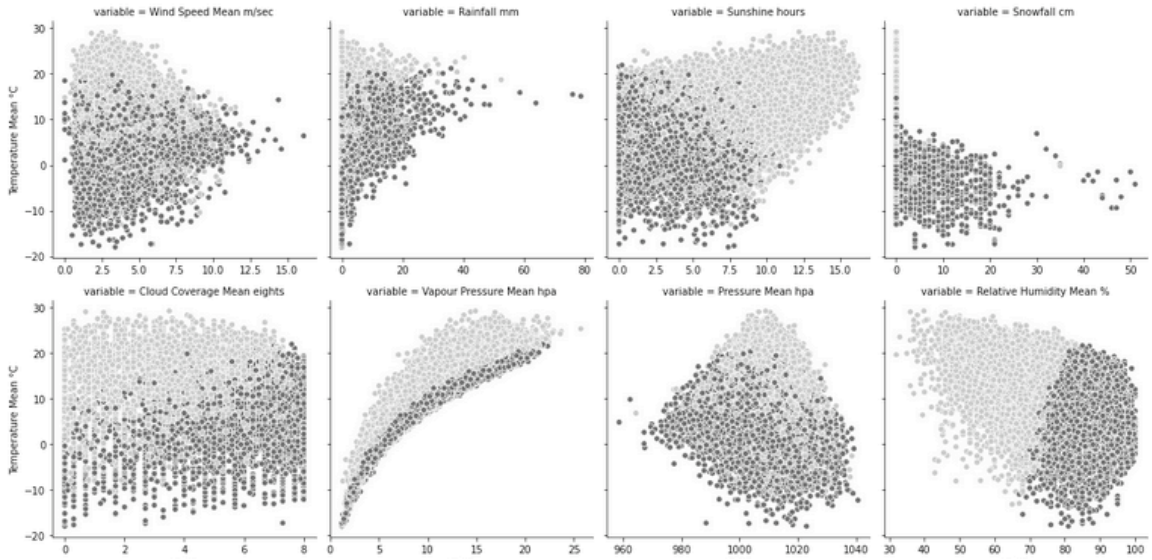
K-Means Clustering

k-means clustering is an unsupervised learning algorithm. The data that is used does not have a target variable. This seems counterintuitive at the first glance because it was stated above that the daily mean temperature is the variable the model is supposed to figure out. You'll see how this can still work in a moment. k-means clustering is used, as the name suggests, to cluster existing data into groups of similar data. Any data point or, in our case, daily observation can only belong to one cluster. It cannot be half in one and half in another. The number of clusters has to be

determined beforehand, either using common sense or cross-validation. For this example I've chosen only two clusters which still illuminate the point well enough.

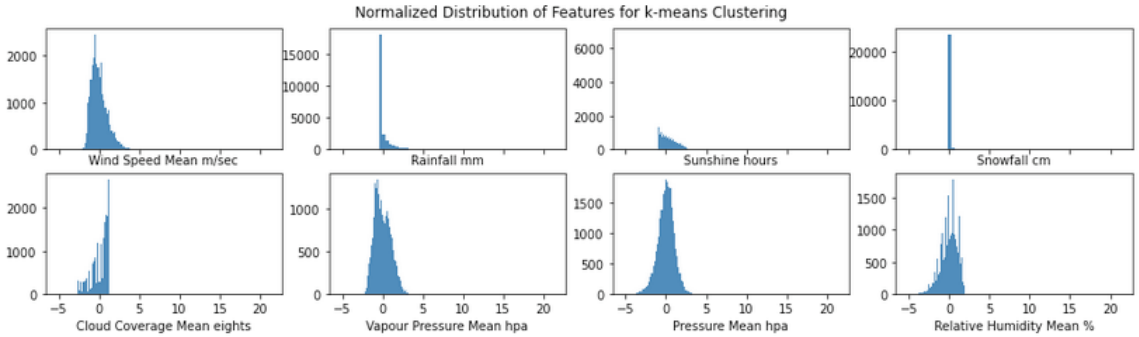
The question was "Can temperature be predicted from all the other measurements?". This means that, contrary to the models used above, all available data can be used for clustering. To keep the model simple for demonstration purposes, minimum and maximum measurements are not used. They are usually directly correlated to the mean measurements. Machine learning models work best if the features are unrelated or independent of each other.

Once the data is clustered using k-means without any additional parameters all features used can be visualized against the measurements for the daily mean temperatures, showing the clusters as colour hues in the scatterplots:



You can see from the plots that actual measurements have been used here in spite of there being huge differences in ranges with a range of just eight for Cloud Coverage Mean and over 80 for Pressure Mean. There are also huge differences between the minimum and maximum values of each variable. This is done here to show the plots more clearly. In order to fit the model these values should be normalized to values centred on 0. This is done to avoid one value being given more weight than any of the others, in this case most likely Pressure Mean which has much higher measurements and a broader range than any of the others.

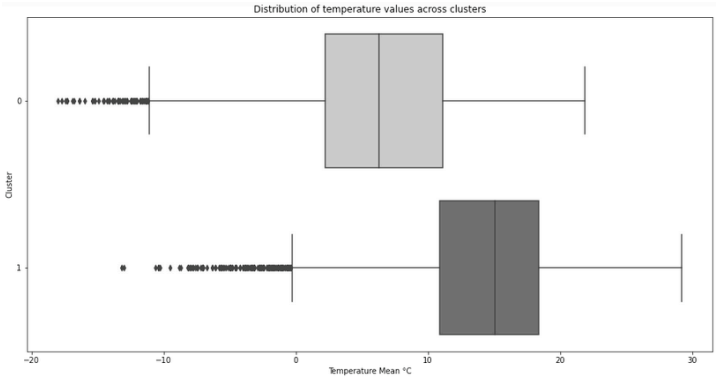
Normalization puts all features on equal footing with a mean of values near 0 and the ranges of values being the same across variables. Here, almost all values are now between -5 and 5.



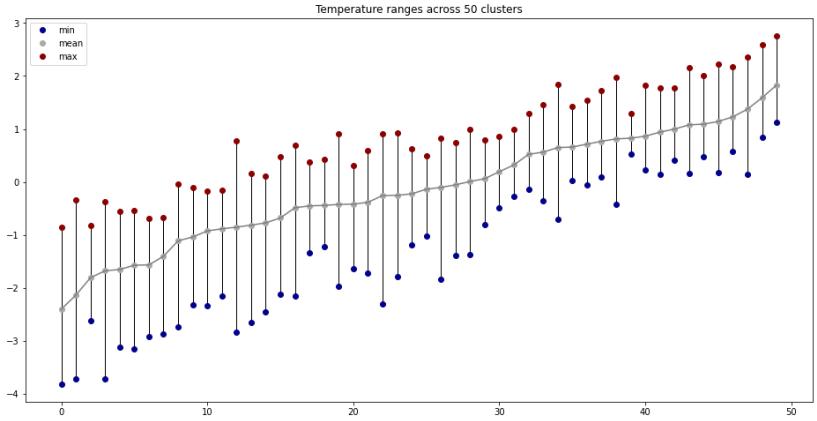
This does make interpretation somewhat more difficult but merging the clusters to the original dataframe can help with that if needed.

Most of the data the model had to work with has been divided neatly into two clusters for each of the variables. Obviously these two clusters are much too broad to be of any value when trying to predict temperature from data where it is missing but the concept is the same using more clusters. However, there are only so many shades of grey that the human eye can distinguish that the plots above wouldn't be of any value for this paper.

The more interesting question is whether the distribution of the mean temperature in both clusters is more or less discreet or to a big extend overlapping. This, too, can be visualized using boxplots:



For the bulk of the data this is the case. There is only a small overlapping area for the highest temperature in the first cluster and the lowest in the second. Obviously there is a lot of uncertainty using only two clusters, especially since each has a lot of outliers and the range of the whiskers is also large, with a range of +- 30°C. So, what temperature to use? This question can be answered much more easily if 50 clusters are used for a full temperature range from -20°C to 30°C. Ordered by the mean value of each the values are now both discreet and with a reasonable distance from one to the next. Please be aware that this plot uses the normalized temperature ranges rather than actual values.



None of the mean values appear twice or more and in a general sense the line connecting them is homogenous with just three or four leaps between the clusters. The actual ranges per cluster between the minimum and the maximum temperatures within this cluster are, perhaps, much broader than to allow reliable predictions. Nonetheless, using the mean value per cluster is a good approximation. Another

problem could be that the number of observations per clusters differs wildly from 912 to just 18. This might lead to wrong predictions if classes with more observations are rated higher than those with few. To find the optimal number of clusters having the smallest temperature ranges and an even distribution amongst classes is out of the scope of this paper though as this was just to show that the possibility exists. It would require extensive feature engineering beyond normalization and parameter tuning which would be better suited for a paper concentrating on these methods.

To go forward, there are two ways to use clusters for prediction:

- The first is to compare new data with missing values for temperature to the existing data per measurements per cluster. This is very time-consuming and not to be recommended for dataset which have more than a handful of variables.
- The second is to use the clusters as another feature in the dataset which is not time-depended in any way for machine learning models. In this case the target variable would not be the mean temperature but the cluster into which the new data falls. Each cluster has a discreet value despite the fact that it is shown as integers. Discreet values correspond to categorical values (because names could be used instead numbers), which means that classification models like decision trees are appropriate here.

Splitting the Data for Training and Testing

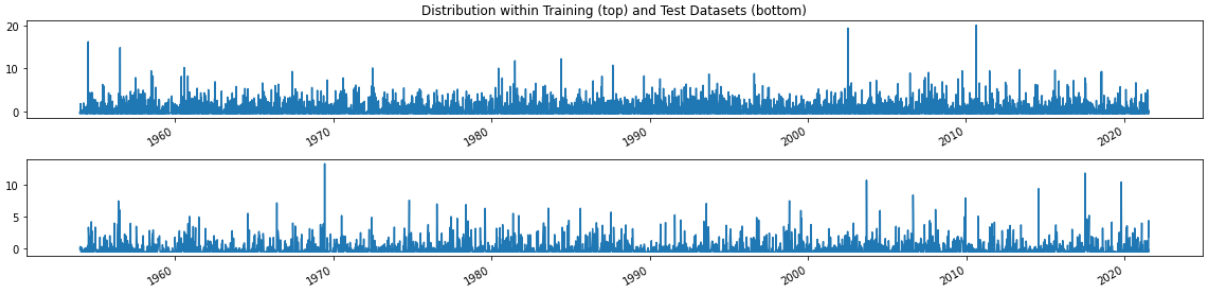
For every model that is used for predictions it is helpful to split the data into a dataset for training and another for testing the model on data it has not seen before but where the true result is known. With more than 24,500 observations across the years from 1954 through summer of 2021 the size for the test dataset can be a generous 20%. In smaller datasets the bulk of the data should go into the training set.

While there are ways to split the data manually it's common practice to use scikit learn's `train_test_split` function for it. As input it takes all the features from the dataframe minus the variable to predict, the target variable (the one which was left out of the features) and the sizes of the training and testing data as rates of the total. A random state can be defined to keep the split the same when rerunning the code.

As mentioned before the number of observations per cluster in our example can vary wildly, so it's important to use stratification when splitting the data into training and test datasets to ensure that this distribution stays the same for both. This is easily achieved by using the parameter `stratify=y`, with y being the column representing the cluster labels 0 and 1 or the target variable. In this case the distribution rates turned out as follows:

Training Dataset		Test Dataset	
Cluster 0	0.607072	Cluster 0	0.607092
Cluster 1	0.392928	Cluster 1	0.392908

Although the time series data is treated like any other data by scikit learn, the distribution over the years was still preserved as plotting the rainfall curves for each dataset shows:



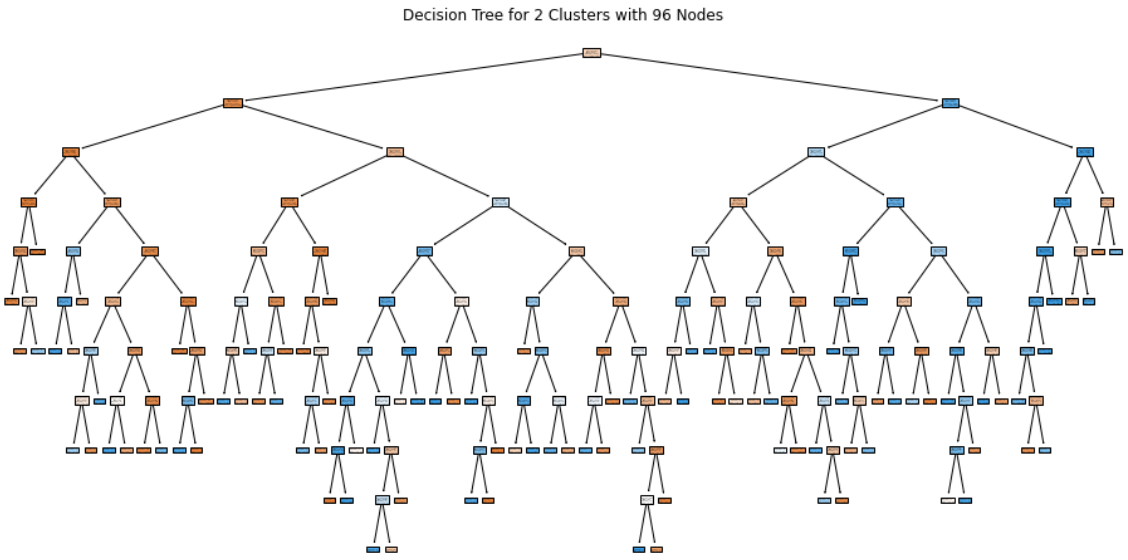
The bottom plot looks "thinner" simply because it has less data per year to plot.

Important Note: Either before or after splitting the dataset the columns with the values that will be missing from additional observations, here temperature, must be dropped. Otherwise the data will not be useful for using on this data as it expects to see the values for temperature.

Decision Tree

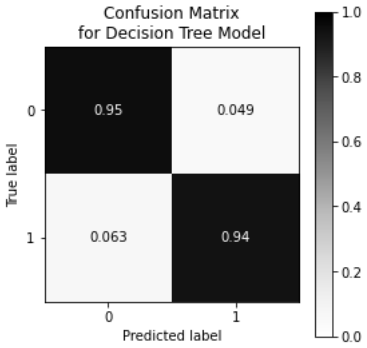
This is the easiest to understand classification model because each decision can be followed down an upside down tree. Based on one criteria or feature at a time the model decides whether some condition is fulfilled or not and than moves on to the next decision node. The answer can always only be yes/True or no/False and, once built, does not vary. This tree can be printed out both in text form and a graphic and be followed from top to bottom for every single observation. However, even with only two clusters as classes to predict, such a tree can become very complex. There are many model parameters that can be specified but for this paper I worked with the defaults, except for the maximum number of leaf nodes the model could use. This number does not correspond to the number of features in the dataset. Any feature can be used more than once, even several times in a row. This is because the split is always binary, e.g. the first question might be: Were there more than 8 hours of sunshine? If yes, were there more than 10 hours of sunshine? and so on.

Using a simple decision tree model with just the maximum number of leaf nodes or decision nodes in this example it turned out that the optimal number of leaf nodes was 96. We would need several DIN A4 pages to get a printout of the tree that was actually readable! On the page you can see what it looks like. Feel free to count all the nodes where decisions are made.



Not as easy to understand as the way the observations have taken down the tree are the weights of the features the model deems most important. For our data `Sunshine hours` has been used twice to split the data at the top level, followed by `Vapour Pressure Mean hpa`. The principle remains the same independent of the number of classes or clusters to predict, but using 10 clusters as classes results in the model regarding `Vapour Pressure Mean hpa` as the most important feature, followed by `Pressure Mean hpa`. While the machine learning modul "scikit learn" or "sklearn" makes it possible to print out the tree and follow it down, I don't know of any way to get an explanation why the model chooses some features above others.

Amazing it may seem, this model actually does a pretty good job predicting the right class for the test dataset using two clusters. It assigned the observations to the correct cluster 94.6 % of the time, without any further tuning of either the dataset's features or the parameters of the decision tree model. The confusion matrix which compares which class label (or cluster) was predicted with the true label shows this very well:



In 6.3% cluster 0 was predicted when it should have been cluster 1, and vice versa in 4.9% cluster 1 was predicted when it should have been cluster 2.

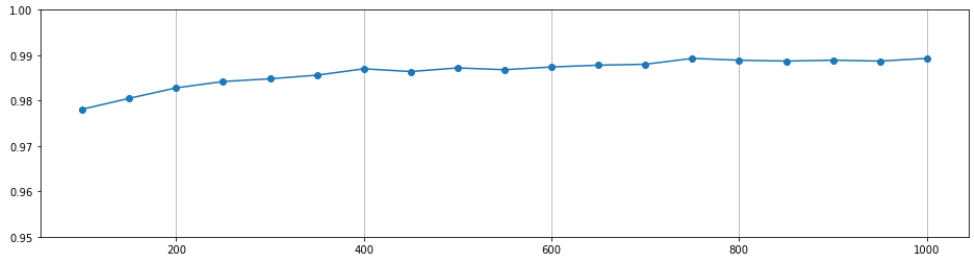
When using the model on the 50 clusters plotted above for using the mean of the temperature range as the final prediction result for each cluster, the accuracy falls to 62.4% and the decision tree becomes more complicated with 101 nodes to use. Extending the search for the optimal number of leaf nodes to 500 results in 491 nodes but only increases the accuracy to 73%. In this case either the model or the cluster assignments using k-means clustering or both would need to be improved significantly.

Gradient Boosting

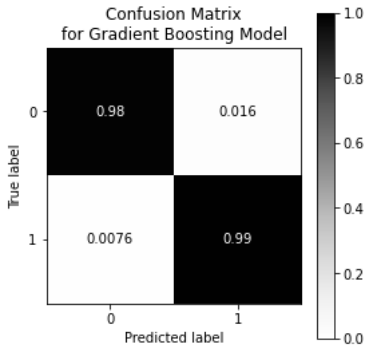
This is another form of supervised tree-based classification that works much like random forests consisting of multiple independent decision trees. The main difference is that gradient boosting does not treat each tree separately as an independent entity but takes previous results into account. This definitely slows down the execution because the trees are built subsequently rather than in parallel but the gain in accuracy makes up for the loss of speed.

Gradient boosting has a lot of parameters to use in the model but for this paper the model just looks at trees built with the same maximum number of leaf nodes (96) identified as the best value for the single decision tree. However, another important factor for optimizing the results is the number of trees used. The default value is 100,

but iterating over a range from 100, 150, 200, up to and including 1000 found that 750 is the optimal number here. The difference in the accuracy scores between the numbers is minimal though after a certain point. Please note that the following plot's y-axis ranges from 0.95 to 1.



The predictions for the test set turned out nearly perfect with a true hit rate of 98.9% when using 750 trees and the same number of leaf or decision nodes as for the single decision tree.



In less than 1% was cluster 0 predicted when it should have been cluster 1. A bit more than twice that often cluster 1 was predicted when it should have been cluster 0. This model promises that with some tweaking it will be able to achieve even better results.

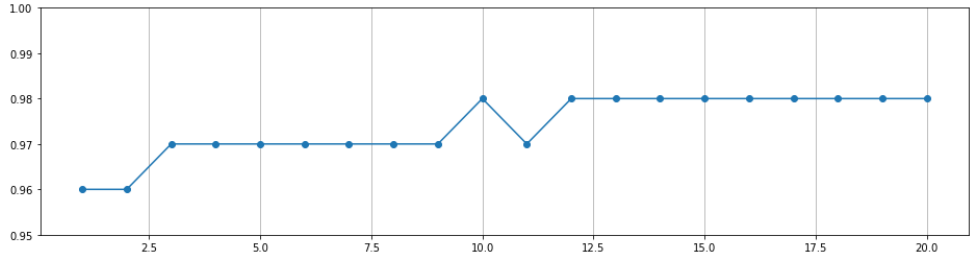
While a single decision tree only took 5 milliseconds of computer time when working with 50 clusters, gradient boosting left me wandering away from the computer in boredom. Building and rating the model alone took so long that I began to wonder whether my computer had given up on me. Including the for-loop to find the optimal number of estimators for 50 clusters, this code took more than half the night at just shy of six hours! On the other hand the accuracy was almost worth the long wait. This model got 89.0% of the classification of the test set right, using 700 trees which is 50 less than for just two classes to predict. This is the best result seen for the three models used in this paper. It has to be said though that the accuracy was only 1.3% better than for the next model so it remains to be seen if better results could be achieved by tweaking the parameters further, thus justifying the long wait.

K-Nearest Neighbors

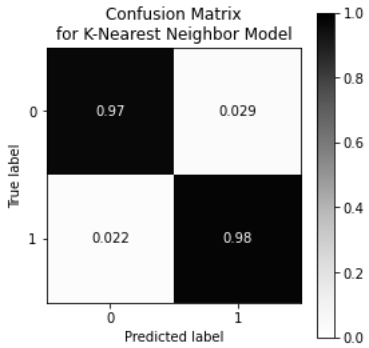
The k-nearest neighbors classifier is related to k-means clustering used above to divide the weather data into separate clusters. It works by placing every observation into the cluster its measurements are least removed from or closes in proximity. This algorithm can be quite time-consuming as it looks at the whole training data every single time it tries to predict the class for one observation. k-nearest neighbors works better when the dataset itself and the number of features used are small. This may require a lot of feature engineering to par down datasets with high dimensionality.

With just eight features there is no need to worry about dimensionality reduction here and the dataset is really small compared to those containing millions and millions of observations. The k-nearest neighbour algorithm is also best used on data that has continuous values on much the same scale. Our weather data is continuous in nature and was normalized before splitting to use with k-means clustering so these requirements are met.

Again a for-loop was used to find the optimal number of neighbours the model should consider. The result wasn't as straightforward as it was for gradient boosting with a sudden drop in accuracy for 11 neighbours:



The predictions for the test set were true in 97,5% of the time measuring the distance to 10 other observations or neighbours so this model worked better for placing new measurements into just one of two classes than a single decision tree but worse than combining many decisions trees with gradient boosting.



It is interesting to see that false predictions for cluster 0 and false predictions for cluster 1 were almost equal at 2.9 and 2.2 % respectively while the single decision tree had 1.5 times more false predictions for cluster 1 than for cluster 0.

This model does quite well using 50 clusters and gets an accuracy score of 87.7%. In this case it uses 20 neighbours, double what was used for just two clusters. It was also still reasonably fast at almost 26 seconds including the for-loop to find the optimal number of neighbours. This shows that the dataset itself is small enough and useful for this kind of algorithm. It might be possible to improve this model by having a closer look at the model's parameters but the gain might not be very big, seeing that the clusters themselves are closely spaced and there might be many borderline cases which might fit into one cluster or the next. It might also be that using an equal number of neighbours rather than an odd might have kicked some observations out altogether because of a tie with 10 neighbours belonging to one cluster and 10 to another even if the accuracy score for 19 neighbours was slightly worse than for 20.

Very Short Wrap-Up

This concludes the topic of working with time series data in python. It has been an interesting and sometimes frustrating journey over a period of many weeks. There were some surprising and also many expected results that could be verified by simply looking out of the window, noting what the weather is like and how it develops over time. Some results were fun like finding out that rain is not keen on certain days of the week, and there were a lot of disappointing results when it came to making predictions. On the other hand, the latter confirms that working with weather data is hellishly complex and depends on so many factors that modelling is far from easy. We see this quite often even with weather forecasts on the news which use far, far more complex models!

The flying visit to classic machine learning showed that time-series data doesn't stand alone and can prove useful in other ways, like making predictions for missing values to enhance a dataset or to model backwards in time when technology was not yet as far developed as it is today and some values might have to be inferred from what data was available at the time. Instead of classification using clusters it would also have been possible to predict temperature directly with regression based models. However, it might be argued that temperatures in a certain range and having exactly on digit behind the comma are not continuous values and thus not a good basis for regression models. I would have liked to have a look at this and will probably revisit the data one day solely for this purpose.

On the whole I enjoyed this journey while learning quite a bit about our local weather and I hope you enjoyed accompanying me as well. Perhaps we will meet again because…

… almost three months have passed since I got the data I worked with here. The summer turned out much less warm and a lot wetter than expected. I'm planning to revisit the data once the year is out and have a look at what has changed since and how the models perform with both more and somewhat different data even if a half year's worth of data won't make that much of a difference in comparison to the 64.5 years already analysed. One thing is clear: The very marked short-term upward trend concerning mean temperatures and the downward trend concerning rainfall have both been broken. It will take a couple of years to figure out whether 2021 was just a freak year or 2020 was the freak year, caused by the massive reduction in aerosols following the first strict (industrial) shutdowns worldwide during the first and second waves of the covid-19 pandemic. Nonetheless, the general trend of rising temperatures is not broken just because one year doesn't fit in. We have definitely seen extreme weather phenomena in 2021, and these, too, are an indication that climate change is a bad as ever.