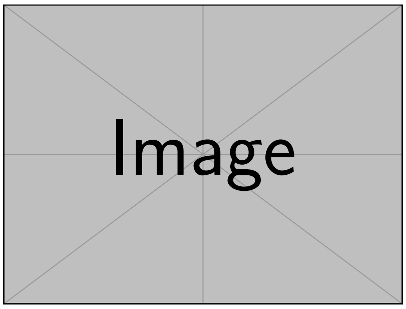
Autumn semester 2025



Data-Mining in Environmental Science

# Learning diary

| | |
|---|---|
| **Date:** | December 7, 2025 |
| **Author:** | Stephen Weybrecht |
| **Student number:** | 2505376 |
| **Supervisor:** | Mikko Kolehmainen |

# Contents

# 1. Orientation

My personal background is that I am an exchange student in physics, studying at the University of Eastern Finland for the autumn semester. As such I have quite a strong background in programming, especially in Python, statistics and machine learning already. As I am quite interested in these topics and my home university offers a rather flexible study plan, I further deepened my knowledge by choosing multiple electives and a Bachelor thesis topic that were deeply connected with data analysis already. Although I expect that many things in the beginning of the course will be topics I already learned, I am very much looking forward to developing a deeper understanding of Data mining and seeing this done in a context I have no previous knowledge yet – namely Environmental science. Looking at the curriculum, there are also many topics I have had no prior experience in which is quite exciting to me. In summary, I expect this course to build nicely on my previous knowledge while additionally providing interesting insights to the field of Environmental science.

As a physics student, I am very used to writing scientific paper-like reports. This is the idea behind many reports of practicals I already needed to write as well as my Bachelor thesis. In these the expression of ones own opinion is actively discouraged. I would even go further and say that conciseness and scientific correctness are virtues hammered into us for years during our studies. Naturally a Learning diary such as this where the expression of a personal opinion and a critical reflection about the topics learned is not only encouraged but actively required is therefore quite a step out of my comfort zone. Still, I am looking forward to experiencing this new concept and seeing how it will shape my learning experience. At least at the time of starting this course this integrated approach of always putting learned things in ones own context, thinking critically and still performing quantitative task during the exercises seems like a very natural way to learn. It will be quite interesting to see how this will change during the course.

My future job prospects as a physicist will most likely revolve about programming and handling large amounts of data, regardless of whether I will pursue a career in industry or I will stay in academia. Jobs as a Data Scientist, Programmer or in the engineering direction are quite common when getting a Masters in physics and experimental physics in academia has mostly been computing, simulation or the analysis of huge amounts of data since many years already. Therefore, having a strong basis in programming, data analysis and visualization are skills one should have after the studies. I expect that this course will deepen my knowledge in Data Mining by not only introducing new concepts but also connecting those learned already on an even deeper level and will thus be a helpful resource for my future.

I will use Large Language Models in the following mainly for getting code suggestions for the exercises and help in the layout of this report (as LaTex can be rather cumbersome at times). The text will mainly be written by myself, although sometimes AI is used for translation and paraphrasing purposes. All code of the exercises as well as the LaTex files to create this report will be made available on a public GitHub repository [1].

# 2. Introduction and basics

## Lecture

The lectures this week dealt with introductory topics regarding the structure of this course, a math and statistics rehearsal as well as an introduction to the topic Data Mining. The math and statistics chapter covers many basic definitions like the axioms and basics properties of probabilities, the definitions of partial derivatives, vectors and matrices and their addition and multiplication properties. Although these are nice to have for completeness sake and should prove helpful for students which do not have a background in statistics yet, for me, they were already known and will therefore not be repeated here. Instead, I will focus on definitions of this chapter which I do not know by heart and which I think will prove useful for the following and will put them into context of what I have already learned.

The sample mean $\hat{\mu}$ and sample variance $\hat{\sigma}^2$ provide unbiased estimators for the population mean $\mu$ and population variance $\sigma^2$ given a certain sample $v_i$ of size $N$, i.e. $i \in (1, N)$. They are defined as

$$\hat{\mu} = \frac{1}{N} \sum_{i=1}^{N} v_I \tag{1}$$

$$\hat{\sigma}^2 = \frac{1}{N-1} \sum_{i=1}^{N} (v_i - \hat{\mu})^2 \tag{2}$$

I recall from a prior course on statistics that the sample variance with the $N-1$ term in the denominator should be used in the case of an *unknown* mean, i.e. if the mean is estimated by Equation 1, as it provides an unbiased estimator and therefore better convergence to the true but unknown sample variance for small $N$. If instead the mean is inferred through different means, the minus 1 term can be dropped. In `numpy` the sample variance can be simply calculated by setting the parameter `ddof=1`:

```python
import numpy as np
# array is a sample array of data
sample_var = np.std(array, ddof=1)
```

Listing 1: Sample variance, calculated in numpy

If there are more than just one random variates a variance can be calculated for each one. Additionally, a so-called covariance between two different variated can also be calculated. Covariances and variances are summarized in a covariance matrix, whose elements are defined as follows:

$$\mathrm{Cov}(x, y) = \frac{1}{n-1} \sum_{i=1}^{N} (x_i - \hat{\mu}_x)(y_i - \hat{\mu}_y) = \hat{\sigma}_x \hat{\sigma}_y \rho_{x,y} \tag{3}$$

Here $\hat{\sigma}_i$ are the standard deviations of the variates $x$ and $y$ according to Equation 2 and $\rho_{x,y}$ is the degree of correlation between $x$ and $y$. It holds that $\rho_{x,y}$ is always between $-1$ and 1 with $-1$ indicating maximum negative correlation, 1 maximum positive correlation and 0 no correlation at all. The covariance matrix (or the reduced correlation matrix obtained when removing all individual standard deviations) therefore indicates correlations between different parameters in a dataset which can be used for explorative data analysis. Another use of it is when fitting a model to a dataset. Here a large degree of correlation between model parameters indicates a surplus of model parameters.

The normal distribution is the most important continuos probability distribution as a lot of measured data follows it. This is due to the central limit theorem stating that means of random variables taken from arbitrary probability distributions will be distributed normally. As measured quantities are usually means over some finite measurement integration time due to a finite detector resolution many data are distributed normally. The normal or Gaussian probability distribution is given by

$$P(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right) \tag{4}$$

with $\mu$ its mean and $\sigma$ its width, corresponding to the population standard deviation in $x$. It is a symmetric distribution and used to formulate confidence intervals as follows:

$$\begin{aligned} &P\left(\mu - 1\sigma < x < \mu + 1\sigma\right) \approx 68\% \\ &P\left(\mu - 2\sigma < x < \mu + 2\sigma\right) \approx 95.5\% \\ &P\left(\mu - 3\sigma < x < \mu + 3\sigma\right) \approx 99.7\% \end{aligned} \tag{5}$$

Further important definitions are those of the Jacobian and Hessian matrices, which are matrices of first order derivates of vector valued functions $\mathbf{f}(\mathbf{x})$ or second order derivates of scalar valued functions $f(\mathbf{x})$ respectively. They are used in optimization algorithms like data fitting or neural network learning. These methods are usually implemented already in various Python packages, however I still list the form of the Jacobian and Hessian here for completeness:

$$\mathbf{J} = \nabla_x \cdot \mathbf{f}(\mathbf{x}) \qquad \text{or element-wise:} \qquad J_{ij} = \frac{\partial f_i}{\partial x_j} \tag{6}$$

$$\mathbf{H} = \nabla_x^2 f(\mathbf{x}) \qquad \text{or element-wise:} \qquad H_{ij} = \frac{\partial^2 f}{\partial x_i \partial x_j} \tag{7}$$

The last part of the lecture considered Data Mining. We discussed that Data Mining is a very structured process, where certain steps are done in order to hopefully get meaningful insights from the data at hand and be able to form a model that generalizes well. The first step is explorative data analysis (EDA), where plotting the data (e.g. in histograms or box plots) or calculating covariance matrices to get a rough idea about correlation are utilized. This step makes a lot of sense to me, especially once one deals with larger and larger datasets. However, this has been quite underutilized during my physics studies so far, as we have usually worked the other way around by first forming a hypothesis (for example a model) and then applying it to the data and doing statistics to see if it fits. It is quite interesting to me to do a different approach in this lecture and to see, where this may lead me. I am looking forward to applying this to real datasets in the following weeks during the tutorials. The next step is descriptive modelling, where one tries to find patters in the data by, fundamentally, playing around with it. One can for example try to fit functions like multivariate Gaussians or use

at end: see if this really was interesting!

clustering methods on the data. Additionally, one can also try to use predictive modelling like classification neural networks to group the data and gather new insights. Hopefully these steps then lead to discovering patterns and rules in the data and finding a strong, i.e. simple model with high degree of generalization and prediction-power instead of remaining with weak models, which do offer some insights but fail to generalize and really understand the data. Weak models are also often classified by many parameters leading to the aforementioned overfitting. During my prior classes where I often used model fitting and machine learning I have often had contacts with overfitting due to fit functions with too many free parameters or overly large neural networks. I therefore know, that avoiding overfitting is one of the most difficult parts in data analysis and am definitely looking forward to learning more about this.

## Exercise

The goal of the first exercise was to get a handle on the provided `toolo.csv` dataset by using the aforementioned technique of EDA. First data from leap years is dropped for better comparability. After this the `.describe` method of `pandas`-dataframes is used to get a summary of the nitrogen dioxide and ozone concentration columns. By this we find mean concentrations and standard deviations of:

$$\begin{aligned} NO_2 &: 38.4 \pm 23.2 \\ O_3 &: 37.0 \pm 22.0 \end{aligned}$$

This makes clear, that the nitrogen dioxide measurement has a larger variability than the units? ozone measurement. Furthermore, this simple analysis indicates already some erroneous data points. The minimum of the $NO_2$ concentration is for example at $-3$ . This is unphysical units and could stem from an erroneous measurement e.g. an error when digitizing the measured concentration. Sometimes measurement devices also deliberately save unphysical values to indicate an error that happened during the measurement. In this case the value of $-3$ could unit be understood as an error code. To see whether this is the case, one would need to take a look at the manufacturers data sheet of the specific detector in question. In principle such data points should be excluded from further analysis to not skew the results one gets from the physical data.

In a next step the columns corresponding to the particulate matter mass under 10 $\mu$m, carbon monoxide concentration, temperature, humidity and wind speed have been analyzed additionally in a similar manner. From this it becomes clear, that the mass and carbon monoxide concentration show similar unphysical negative values as the nitrogen dioxide concentration. As also these unphysical values occur at perfect integer values, a look into the datasheet or the digitization software of the detectors would be helpful to understand these values and see, if they maybe really correspond to error codes. O course, filtering this data out is simple, but it is worth to investigate where this erroneous data stems from, what could be wrong during measurement and also how much data is affected. Additionally, there are some interesting insights to be gained by looking at the mean, standard deviation, minimum and maximum values of the data besides faulty measurements. One can for example see, that there is a seemingly small number of very heavy small particles in the analyzed air, as the mean value of the mass is roughly 25 times smaller than the maximum value. Furthermore, one can see that the carbon monoxide concentration in the air is much smaller than that of any other gas. The temperature and humidity show very large spreads over the dataset that are on the order of a forth of the whole data range. Lastly, one can see that the wind speed is a strictly positive quantity and is therefore not directional. This shows that already a simple analysis requiring only a few lines of code can give valuable insights on the data range, its variability and possible errors during measurement.

Next the correlation between the individual variates is calculated (compare Equation 3) and plotted. This is shown in Figure 1. The strongest positive correlation is present between the nitrogen dioxide and carbon monoxide concentration. This indicates, that the processing leading to an increase or decrease of these gases in the atmosphere are strongly linked such that an increase in one is connected to an increase in the other gas. At this point it is important to keep in mind that this analysis shows only correlation but not causality meaning that we cannot conclusively state which change leads to which but only that they are connected. Similarly, an anticorrelation between the concentrations of nitrogen dioxide and ozone can be seen as the correlation coefficient between those two quantities is negative. This indicates that a rise in one of the quantities is linked to a fall in the other. Additionally, a correlation between particle mass under ten microns and nitrogen dioxide concentration is present, suggesting a link between these two. There are also smaller degrees of anticorrelation between both the wind speed and nitrogen dioxide concentration as well as temperature and humidity. While the latter seems reasonable (higher temperatures seem to be connected with lower relative humidities), the prior is quite hard for me to understand. It could be interesting to investigate this further. It is quite fascinating to me, that such a simple analysis, requiring only a few

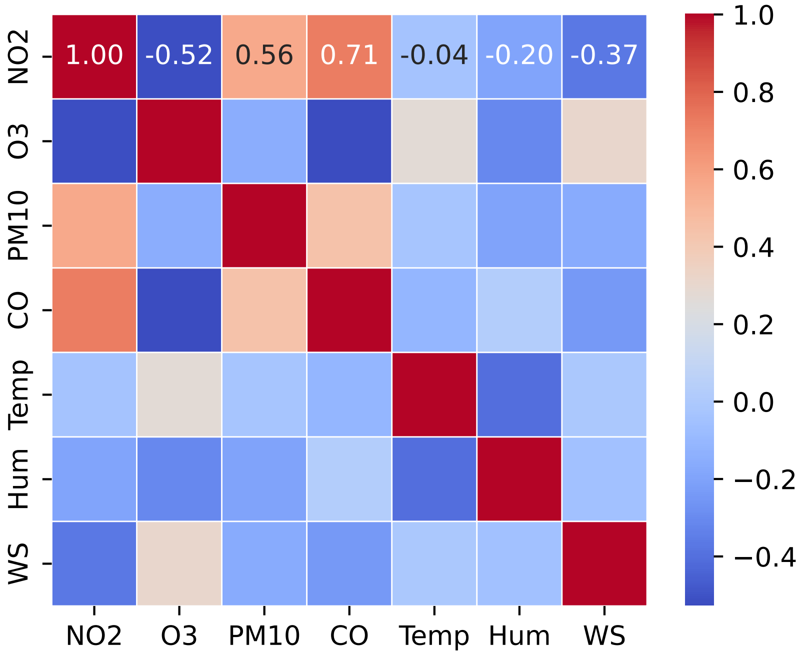


Figure 1: Cross correlation matrix calculated using the mentioned columns of the `toodo.csv` dataset.

simple lines of code can already extract interesting research questions that can be expanded on.

In a last task the individual distributions of the quantities was looked at by plotting histograms of the data shown in Figure 2. While temperature as well as the gas concentrations and wind speed qualitatively look roughly normal distributed (taking into account, that the distribution is cut due to unphysical values), the humidity and particle masses under ten microns seem quite asymmetric. The histogram of the particle masses roughly fits to the realization I already had when looking at the mean, minimum and maximum values of the taken data in a prior step. It seems there are a lot of particles with low mass which influence the mean heavily, but there also remain a few particles of very high mass. In fact, to me the distribution looks similar to an exponential when postulating that very small masses are underrepresented probably due

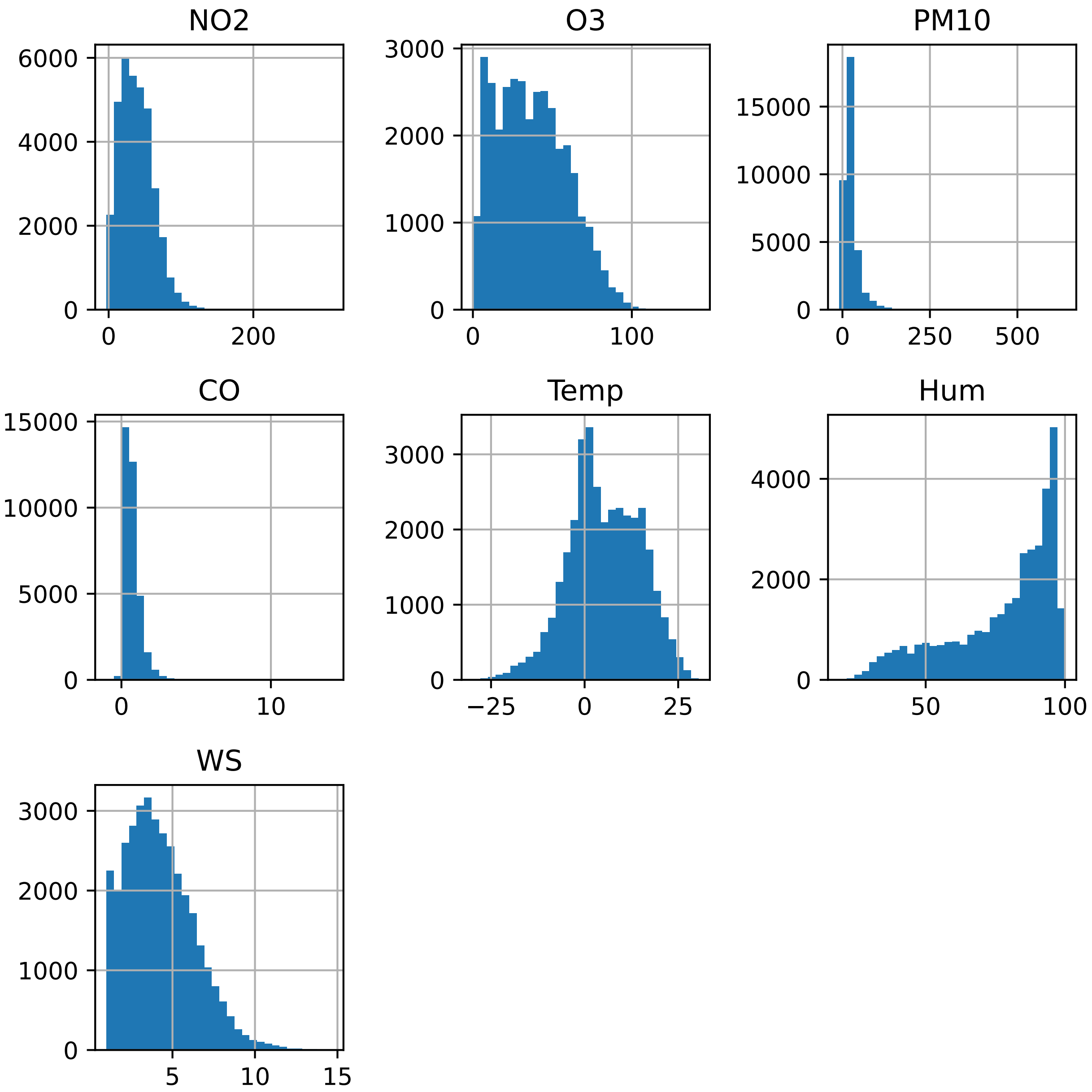


Figure 2: Histogram of the data columns of the `toodo.csv` dataset.

to the detector not being able to measure arbitrarily small particulates accurately. Of course, this would still need to be quantified, but it poses an interesting research idea for further analysis. The humidity diagram is heavily skewed towards high humidities.

## Reflection

Although the topics presented this week were mostly nothing new to me, I found applying explorative data analysis quite interesting. It was quite fascinating to see how much knowledge can already be gained from a complex dataset by just looking at statistical properties of the data and its correlation. This is especially true, as I do not have a background in environmental science and therefore discovering correlations between different gases in the atmosphere or temperature and humidity did always come as a surprise and led to the curiosity to read up on the phenomena happening in the atmosphere that could lead to these links. My learning itself was quite well-structured this week. I was successful in writing the learning diary in multiple sessions and doing small updates at a time which also lead to me thinking more about the presented topics and dataset and also re-discovering topics that I already learned in previous courses which fit nicely into this week's lecture. Having a stronger background in statistics I do however think that the meaning of the sample variance and mean as unbiased *estimators* of the true and unknown population quantities could have been motivated more thoroughly and clearly. I feel like this is one of the most important concepts to understand when starting to discuss statistics, as for people having no background so far these definitions must have come out of nowhere. I do however also understand the time constraints of this course, as it is a synthesis of two different courses which were held prior. Additionally, I think it would have been interesting to be required to do some analysis/coding myself for the exercises. The way the exercises are structured now, the entire code is already provided and only the description part is required. I feel like more interesting and maybe more rewarding insights could be gained if students are asked to perform some exploration on their own. This will maybe be the case in future exercises.

# 3. Environmental data and its pre-processing

## Lecture

The focus of this week's lecture were data pre-processing steps necessary for typical datasets in environmental science. For this we discussed three important steps: Handling missing data, metrics, i.e. a way to compare data and transformations as a way to scale data to similar magnitudes. For each step we talked about the principle behind it and why it is necessary after which a few example algorithms were discussed. In the following I will focus on the idea behind each step and the most interesting and new algorithms to me and will compare their qualities.

The first topic we discussed was missing data. It is clear that efficiently handling these gaps is vital as firstly in any real dataset these are numerous and secondly most further analysis steps cannot handle missing values. These `NaN` values can stem from measurement errors, human mistakes or instrument failures. The most straightforward although very brute-force way is to just drop any entry of the dataframe, that contains at least 1 or some number of `NaN` values. This can be done rather easily as can be seen in Listing 2.

```python
import pandas as pd
df = pd.read_csv(foo.csv)       # read some data
thresh = 1
df = df.dropna(thresh=thresh)   # drop every row with at least thresh NaNs
```

Listing 2: Example for dropping rows with missing values

If one finds out that mainly one variable is problematic, one could also delete this variable (column) from the dataset. Both options have the strong downside of cutting also viable data which generally leads to a loss of predictive power of a resulting model. Another way of dealing with missing values is imputation, meaning filling the missing value with another one based on some algorithm. A few of these are listed here, ordered simple to more complex:

- Imputation with mean, median or a random value ⇒ can alter the distribution of data significantly
- Imputation by linear interpolation ⇒ useful for time-series data
- Nearest neighbor imputation: For each $N$ dimensional feature vector of the dataframe $\mathbf{x}_i$ where the index $i$ represents the row that has $N_{\text{miss}}$ missing values, we calculate a distance to every other feature vector as follows: $d_{ij} = \frac{N}{N-N_{\text{miss}}} \|\mathbf{x}_i - \mathbf{x}_j\|^2$. The missing values are taken from the feature vector with the smallest distance ⇒. The most interesting thing about this metric for me is that there is no introduction of new values, while also taking into account the reliability of data by making vectors with many `NaNs` have a large distance.

It is important to note that the choice of imputation method can heavily influence the model so taking care is needed. A good choice is to use univariate methods like interpolation for short gaps and multivariate approaches like the nearest neighbor imputation for longer gaps in the data.

The next topic concerned metrics, a way to compare how similar or dissimilar two entries of a dataframe are. Metrics are a quite general mathematical concept allowing many different forms with different use cases, however here only a few examples will be named. If the data entries are numeric a simple measure of similarity is the dot product between vectors, while a measure of dissimilarity could be the distance between them. An interesting distance measure has already been shown above, but a very general and often used one is the so called Minkowski distance

$$d(x, y) = \left( \sum_{i=1}^{N} |x_i - y_i|^p \right)^{1/p} \tag{8}$$

which reduces to the normal Euclidean distance for $p = 2$ but also has interesting cases for $p = 1$ (Taxicab/Manhattan distance) or $p \to \infty$ (Chebyshev distance). These are illustrated in Figure 3 for a 2-dimensional case. Additionally, there are also many different metrics used for class-like or binary data.

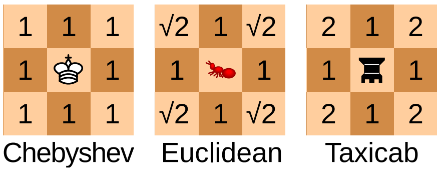


Figure 3: Different cases of the general Minkowski distance illustrated in 2D with the analogy of chess pieces. Taken from [2].

The last topic we covered were transformations which can affect single values, rows or columns of the dataset or even the dataset as a whole. Column transformations are especially important, as different data columns are usually very different in the magnitude of their entries. If no normalization is done, large entries will dominate small ones, which is usually not wanted. Additionally, columns could contain discontinuous data. Some useful and interesting transformations include:

- Discontinuous, cyclic data (like hours of the day) can be handled efficiently by expressing it by its sine and cosine components leading to a range of values between -1 and 1. The additional cosine is needed to distinguish between the rising and falling edge of the sine function. This seems very useful, as it normalizes the data as well.
- Logarithmic pre-scaling of columns can be useful if the data has a large spread over multiple orders of magnitude. This also leads to a rough normalization but alters the distribution.
- Variance scaling, i.e. shifting the data columns by its mean and normalizing by its variance is also a standard technique. It could prove especially useful if the data can be assumed to be Gaussian, as we then expect a standard normal distribution afterward. It does suppress outliers, however.

- Equalization forces the data to be in the range of $(-1, 1)$ but can lead to pronounced outliers. This is a problem I already faced in a previous machine learning project of mine. The `sklearn` documentation provides a great comparison [3].

The main take-away from this lecture and my prior experience with machine learning and analyzing datasets is the following: It is crucially important to think about all pre-processing steps that one performs on their dataset. They can have a huge impact on the statistics of the data after the processing, the model one gains and especially the machine learning efficiency and features that are learned by the neural network.

## Exercise

This weeks exercise deals with distance metrics, scaling of data and binary encoding. In a first step the `iris` dataset provided by the library `sklearn` is imported as it only has numerical values and therefore provides an easy example for the following analysis. Note however, that the exact dataset used does not really matter, as in the following only the effect of transformations is examined, not the data itself. In a first step, five random entries of the dataset are chosen and the Euclidean distances between them are calculated. This results in a $5 \times 5$ matrix containing the distance between any line with any other one in the 5-dimensional feature space. Trivially, the distance between any entry to itself is zero and there are some lines which are more "similar", i.e. have a smaller distance and some which are more "dissimilar". However, the distances are not yet normalized to a range between $(0, 1)$ and are therefore tricky to interpret. In a next step all entries are therefore first normalized (by requiring their $L_2$-norm to be equal to unity) and then the distance matrix is recalculated. Still, all entries have zero-distance to themselves and the ordering of distances between entries remains intact. This means the closest and most far apart entries are still the same, with the added benefit that all distances are now smaller than 1. This leads to a better comparability for the naked eye but is also important for further data analysis steps that might follow. I could imagine working with normalized entries is especially important for algorithms like k-nearest neighbor classification, as outliers due to very different data ranges or even different units will diminish its accuracy. From prior experience I furthermore know that normalization is quite important when one plans to supply the data as an input to a neural network, as these often used activation functions that are only effective at a data range of $(0, 1)$.

In a next step the effect of scaling the data columns with variance and equalization scaling is analyzed. This is shown in Figure 4.
As was already discussed the transformations have different effects. Variance scaling transforms data to have zero-mean (Note: *not* zero-median, the lines in box plots show quartiles of the data) and a variance of one. As such it is especially useful when data is expected to be normally distributed as it will result in data following a standard normal distribution afterward. However, it is important to keep in mind, that variance scaling does not change the underlying distribution as can also be seen in the plot. In other words, skewed data remains skewed, the only effect is a shift and "zoom" of the data. Outliers can have a substantial influence, if they are at very large magnitudes, as they then have a strong influence on the empirical mean and standard deviation which compresses the data strongly after scaling. This

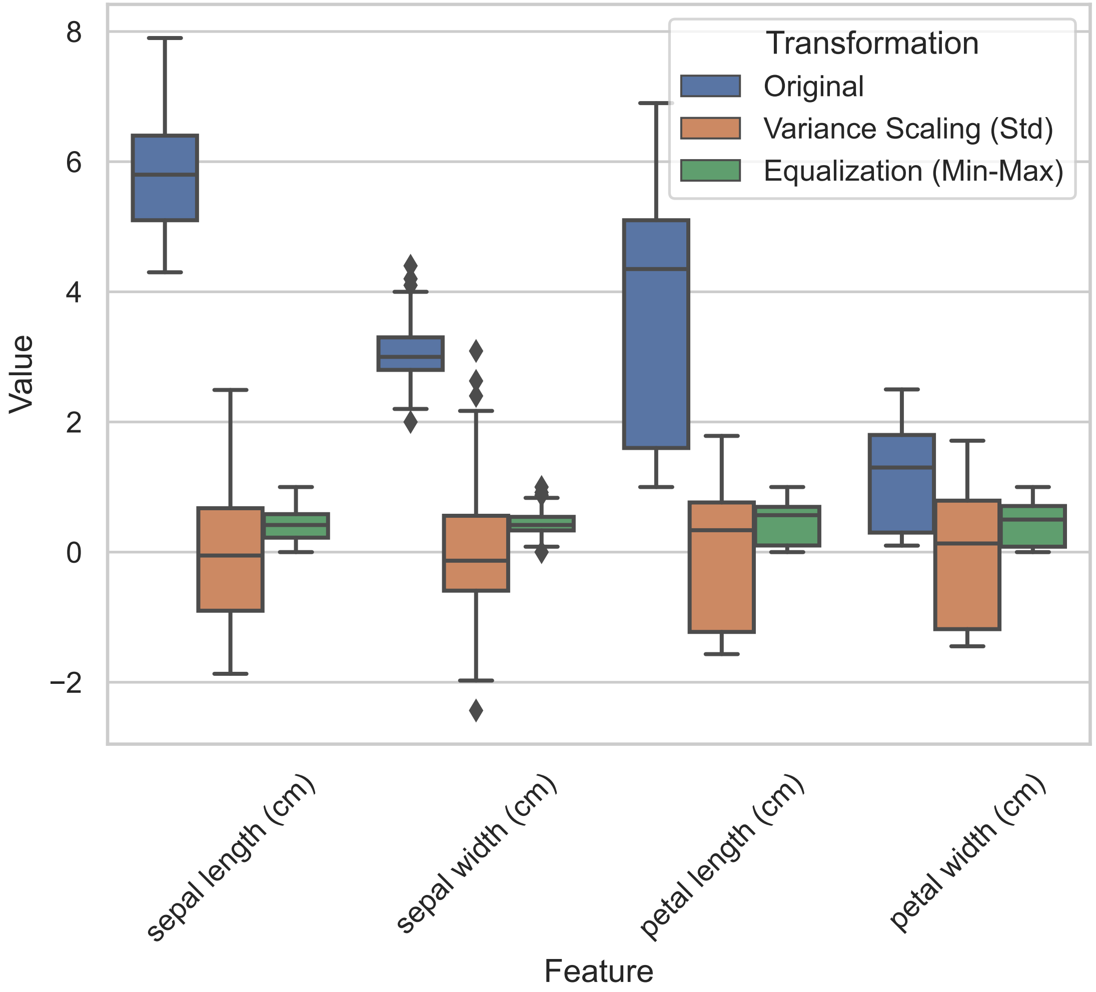


Figure 4: The original, variance scaled and equalized data of the `iris` dataset is shown as boxplots.

is less visible here, but nicely illustrated in [3]. Equalization forces all data to be in the range of $(0, 1)$ and is therefore heavily influenced by large outliers, as these are normalized to one which can lead to an extreme compression of the more common values in the center range. This can be seen in the sepal width column of Figure 4 where 50% of the data lies in a very narrow range after the transformation, which can lead to a difficulty in distinguishing the most important features and an overrepresentation of outliers in a further model. Equalization also keeps the underlying distribution and only shifts and squeezes it. Because of the mentioned effects of standardization I conclude that one has to be quite careful when applying this simple min-max scaling on data with outliers. It is only useful if no extreme outliers are present and data has to be in the unit interval for further analysis (for example sigmoid and tanh activation functions in neural networks). Still there might be scalers which are better suited for the data, see e.g. [3].

As the last step the Dice similarity metric is tested on categorical data. It is defined for binary data $X$, $Y$ as

$$S_{\text{Dice}} = \frac{2\,|X \cap Y|}{|X| + |Y|} \tag{9}$$

where the cardinality of the sets, e.g. $|X|$ is defined as the number of entries that are equal to one. First categorical sample data about cars, including their color, engine and brand is created. This is then transformed to binary data using the handy method `pandas.get_dummies()` and the Dice similarity is calculated between the entries (cars). The results make intuitive sense. Cars 1 and 2 which do not have any attributes in common have a similarity of 0, while cars 1 and 4 share two out of three attributes and therefore have $S_{\text{Dice}} = {}^{2}/_{3}$. It is noteworthy that while the conversion to binary labels creates many more categories this does not influence the Dice similarity negatively as given the cardinalities definition only nonzero entries matter. In my opinion this makes the Dice similarity much more usable, as it makes sense given the original categorical data and is normalized to unity given this data as well. However through the binary representation gained by `pandas.get_dummies()` some information is also lost for future models. A future model cannot know anymore that the three specific colors are just different values of a more general color category which would certainly prove quite useful for generalization. This could be solved by using a different metric that does not require binary data and assigning the different colors e.g. different numeric labels (like 1,2,3 or a three-dimensional color vector). Using binary data can also drastically increase the dimensionality of the dataframe, especially when there are many different values belonging to the categories. This makes further computational steps much more computation heavy and therefore timely.

## Reflection

What I learned this week was quite interesting to me. Although I still knew most of the things presented in the lecture and exercise, there were some new things, like Nearest-neighbor imputation and the Dice metric that i did not know before. I think these will prove quite useful. Furthermore, the topic about normalization and scaling of data was a great starting point for me to read up on the documentation [3] once more. As mentioned numerous times, I am quite fond on statistics and did already have some machine learning projects where input data normalization was quite a challenge. It was nice being reminded of these topics again and seing them used in a different light. For me this repetition in different contexts and making connections to prior courses really helps my understanding, so I am happy to see that there are so many links. Writing the diary this week was more rushed and stressful due to other university courses interfering. This is something I want to do better next week by having a more structured plan for writing. Last week I remarked that it would be nice to do more programming by oneself. While this was still not yet needed this week I quite enjoyed the more open questions about e.g. k-mean-clustering and potential limitations where one needed to think more and read up on some additional limitations of the techniques covered, as this also helped my understanding.

# 4. Data visualization

## Lecture

This week's lecture focussed on visualizing high dimensional data and working with data in time series. Especially the first part contained many new things to me, which is why I will put a bigger emphasis on it in the following paragraphs. If data is low dimensional, visualization via histograms, box plots, scatter plots, or other methods is straightforward and has been discussed in the chapters above already. A more interesting question is how very high dimensional data with maybe hundreds of entries (columns in a dataframe) can be visualized in a 2-dimensional space to get a grasp on clusters and for example prepare the data for clustering algorithms to obtain data classes. We discussed to examples of dimensionality reduction here, namely Sammon mapping and self organizing maps (SOMs). These are quite interesting to me, as I only knew about Principal component analysis (which has the disadvantage of being linear) t-SNE prior to this lecture. Both Sammon mapping and SOMs are nonlinear and can therefore capture the usually complex high dimensional manifolds spanned by the data but are still easy to understand.

Sammon mapping is based on the idea of pairwise conserving distances in the mapping of input vectors $\mathbf{x} \in \mathbb{R}^D$ to output vectors $\mathbf{y} \in \mathbb{R}^d$ with $d < D$ (usually $d = 2$). Specifically, the algorithm is constructed such, that small distances are weighted more strongly. This means, it tries to achieve a mapping that preserves local neighborhoods well and therefore preserves grouping of the data points. Sammon mapping follows these steps:

1. First, pairwise distances between all inputs $d_{ij} = \|\mathbf{x}_i - \mathbf{x}_j\|$ and all outputs $d'_{ij} = \|\mathbf{y}_i - \mathbf{y}_j\|$ are calculated. For the first iteration outputs can be initialized randomly.

2. A cost function is calculated as $E = \frac{1}{\sum_{i<j} d_{ij}} \sum_{i<j} \frac{(d_{ij}-d'_{ij})^2}{d_{ij}}$. Note how the denominator leads to a larger weight for small distances. The prefactor is just for normalization.

3. This cost function is minimized through usual gradient descent and the output mappings are updated as $y_i \leftarrow y_i - \eta \frac{\partial E}{\partial y_i}$, where $\eta$ is a preset learning rate.

check equatoins. they differ from lecture

The downsides of Sammon mapping include its sensitivity to local minima leading to failed mappings (due to random initialization), and that the algorithm is rather slow ($\mathcal{O}(N^2)$ distance calculations). However, it is useful if distances should be preserved well.

Self organizing maps are an unsupervised neural network, where neurons are placed on a usually fixed, 2-dimensional grid. Each neuron has a weight vector $\mathbf{w} \in \mathbb{R}^D$ of the same dimensionality as the data $\mathbf{x} \in \mathbb{R}^D$. The main idea is to find for each input sample (row of the dataset) the closest neuron, i.e. the one where the weight matches closely the input value itself. This neuron and its surrounding are then updated to be more similar to the input. After learning one tries to achieve the situation where each neuron reflects one "typical" input sample with neighboring neurons in the 2-dimensional grid being similar. This way a high dimensional input space is represented in a flattened, although topologically similar space. The following steps are important:

1. First, the best-matching unit (BMU), the closest neuron, is calculated: $\mathrm{BMU}(\mathbf{x}) = \arg\min_i \|\mathbf{x} - \mathbf{w}_i\|$

2. The BMU and its neighbors are updated to be more similar to the input. The update of neuron $i$ is based on the distance of the neuron to the BMU $c$: $\mathbf{w}_i(t+1) = \mathbf{w}_i(t) + \alpha(t)\, h_{ci}(t)\, (\mathbf{x} - \mathbf{w}_i(t))$ with the neighborhood function (Gaussian) $h_{ci}(t) = \exp\left(-\frac{\|\mathbf{r}_c - \mathbf{r}_i\|^2}{2\sigma(t)^2}\right)$. Here $\alpha(t)$ is the learning rate and $\sigma(t)$ the kernel size, specifying the weighting of distances.

After these steps have been repeated for many input samples and the network is trained, similar inputs appear as clusters in the low dimensional representation, while boundaries or gaps in the representation show dissimilar inputs. This can be especially well seen when plotting a heatmap of the so-called U-matrix for neuron $i$, defined as

add boldface vectors here and before

$$\mathbf{U}(i) = \frac{1}{|N(i)|} \sum_{j \in N(i)} \|\mathbf{w}_i - \mathbf{w}_j\| \tag{10}$$

as it contains the average distances between the neuron and its neighbors (The sum ranges over all neighboring neurons). If the entries are small, the values in the neighborhood are quite similar which signals a cluster, while large entries signify the boundary between clusters.

The last topic was dealing with time series data as it is quite abundant in environmental data. Most of the topics discussed were nothing new to me as I have learned about them and used them extensively for example during my Bachelors thesis, which is why I will mainly glance over them. First we discussed examples of visualizing time series data to find important structures. We mentioned:

- Autocorrelation of the signal. It describes the correlation between the signal and itself for different time lags and can therefore be used to get an intuition about cyclic structures. See `numpy.correlate()` for further information.

- Plotting trajectories onto mappings obtained by SOM or Sammon mapping. This seems quite interesting to me, as one should be able to observe, how the data moves between clusters given a certain time lag.

- Fast-Fourier-transforms (FFTs) to transform the time domain signal into frequency domain. See `numpy.fft.rfft()` for real signals.

- The complex result of an FFT can be visualized as a Periodogram, a real estimate of the power spectral density of the signal by either taking the modulus squared of the FFT result and applying proper normalization or using `scipy.signal.periodogram()` directly. This is an important visualization tool, as one can easily identify the most dominant frequencies (like weekly or daily cycles) in the signal as peaks.

Lastly, we discussed filtering the signal to remove these cycles or slow trends for the further analysis. This can be done by:

- A simple moving average. In effect this is a convolution of the signal with a boxcar function or a very primitive lowpass. It is rather primitive as high frequencies are not filtered well but very easy to compute in a first step. Better low-, high- and band pass filters are available e.g. in `scipy.signal.butter()`, but they have limited usability in environmental data.
- Seasonal differencing, i.e. subtracting the value one period (day, week, . . . ) prior is often used, as both slow trends and the filtered frequency are eliminated.
- Slow trends can often be extracting by subtracting a fitted line to the data.

After filtering one should visualize the data again (e.g. with a Periodogram) to see how the filter affected the data.

## Exercise

This week's exercise was about comparing different visualization techniques and finding out, what information they provide about a sample dataset. Here, an air quality dataset was provided. The data is imported as a `pandas` dataframe and an additional standardized time column is created. Then four main topics are covered: First the distributions and outliers are analyzed using histograms and box plots, then the relations between variables are examined with scatter plots. After this, temporal patters are investigated using smoothing, autocorrelations and periodograms and lastly dimensionality reduction is used.

Looking only at the histograms one can see, that most distributions look more or less Gaussian when on keeps in mind that they are sometimes cut due to unphysical values. However, especially the PM10, as well as the humidity values seem to follow a different distribution. Additionally, one can see that many outliers are present in the $NO_2$, PM10 and CO histograms. These are visible as long tails with only a few entries in each bin. It seems also, that $NO_2$ is more variable than PM10, which is more peaked but also has larger outliers. Through changing the bin number, I was able to find that undersampling the data by using too few bins is not good, as much detail is lost, and it is hard to see a trend and outliers this way. On the other hand increasing the bin number too much is also not good as fine variability in the data starts to appear, which makes interpretation harder. Having some amount of averaging by collecting the data in e.g. 30 bins seems optimal.
Next, I took a look at the box plots of the same data shown in Figure 5. These unveiled that $O_3$ has the largest interquartile range (IQR), which could also be seen in the very broad histogram. This could be due to the amount of ozone in the atmosphere changing a lot (e.g. during the seasons) instead of staying at a more or less fixed level. The highest median is present in the humidity data, which was expected from the histogram already as well, as it seems to be nearly exponentially peaked at 100%. The box plots excel at showing the spread and amount of outliers. It is immediately clear that PM10 and $NO_2$ have the largest amount of outliers in the dataset – This was way harder to see when I looked at the histograms. Comparing the both one can see that while $NO_2$ has a larger IQR, it has less extreme outliers than PM10.

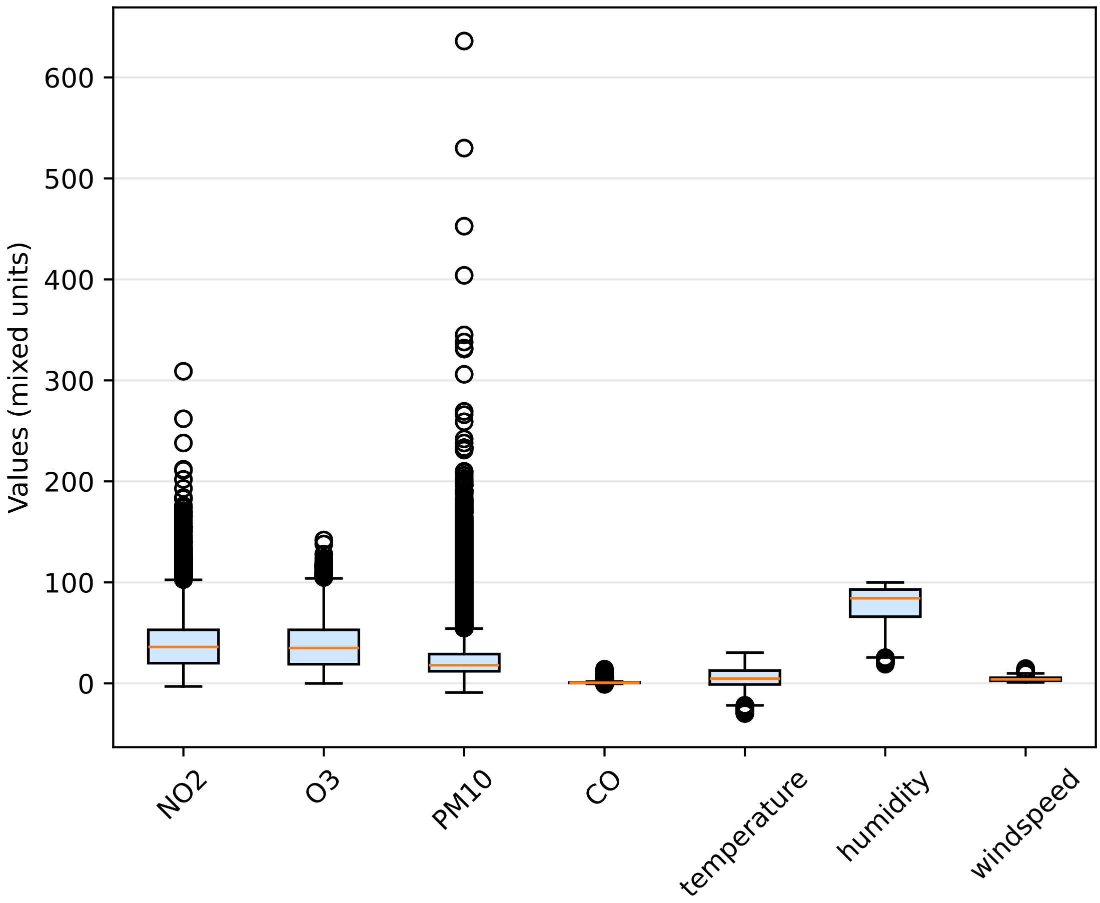


Figure 5: The boxplots of the air quality dataset are shown.

Using scatter plots like the one shown in Figure 6 correlations between variables were examined next. When examining $NO_2$ and humidity, I could see a slight negative correlation, as the slope of a fitted line is negative. As there are many outliers in the $NO_2$ data which seem to influence the fit substantially, but the bulk of the data seems to be quite compact in a cloud, this simple fit does not yet provide compelling evidence in my opinion yet. One should maybe try to remove and quantify outliers first, before repeating the analysis. No clear trend can be seen when looking at the correlation between wind speed and the temperature columns. On the other hand, a strong positive correlation can be seen in Figure 6, where PM10 seems to increase quite linearly with rising $NO_2$ levels over the whole measurement range. Using an additional color scale one can also see a strong anticorrelation between the wind speed and the $NO_2$ measurement. While PM10 and wind speed also seem slightly anticorrelated, this is less evident here, which can also be seen when looking at the two columns separately.

Next, I looked at temporal patters in the data. For this we started with cleaning the date time column and sorting it ascendingly. Then I looked at a simple rolling average filter taken over 24 hours to smooth daily fluctuations. This leads to more smooth, noise resistant curves where trends (or the lack thereof) can be seen more clearly. After smoothing the data seasonal trends can be easily seen in the temperature (peaks in summer), $O_3$ (peaks in spring), PM10

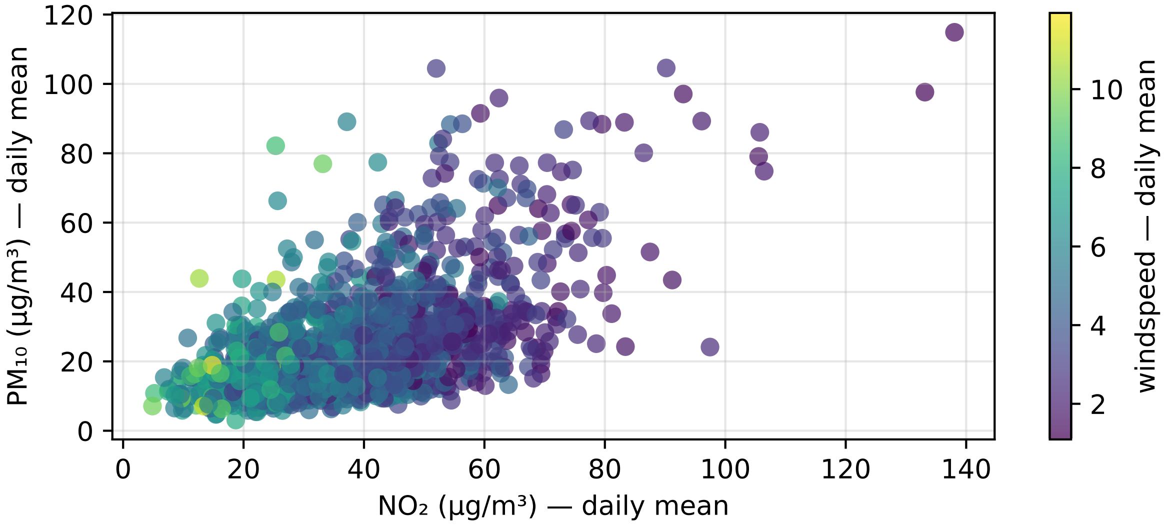


Figure 6: One scatter plot indicating correlation between variables is shown as an example.

(spikes in the winter) and humidity (dips in summer). I could also spot a gradual increase in temperature and decrease in $NO_2$ and PM10 levels over the years by using averaging and a linear fit. This simple example shows, that although a rolling average filter is quite primitive, it can help extract interesting insights to the data quickly and can therefore be used as a straightforward first step.

The next task was to look at the autocorrelation of the signal. For this a function `autocorr_series ()` is called in the provided code but is not defined there. I implemented it myself as follows:

```python
def autocorr_series(y_ac, max_lag):
    autocors = np.zeros(max_lag + 1)
    lags = np.arange(max_lag + 1)
    for lag in lags:
        autocors[lag] = y_ac.autocorr(lag=lag)
    return lags, autocors
```

Listing 3: Implementation of `autocorr_series` function

The autocorrelation shows peaks every 24 hours, indicating a daily cycle in the PM10 values. It should be noted that all peaks at multiples of 24 hours are just higher harmonics that stem from the fact that any 24 hour regularity in the signal also means there are 48,72,. . . hour regularities. In other words: If a value is similar the next day, it is again similar in two days. This shows there is a daily pattern in the emission of small particulates. This could for example be due to traffic, which also has daily repeating patters (many cars during rush hour, few in the night).

More information is revealed when looking at the periodogram of the PM10 data, which is shown in Figure 7. Clear peaks can be seen at 8, 12, 24 and 168 hours period. This indicates that additional to the daily cycle, there are also cycles that are a third, half and 7 days long. Following the traffic-hypothesis as an explanation to the PM10 cycles, the weekly cycle could be due to Sundays having generally less traffic as most people do not need to work. The small 8 and 12-hour cycles could maybe be explained by a peak in traffic during the morning and evening, before and after the 8-12 hour work day is completed. These are however of course

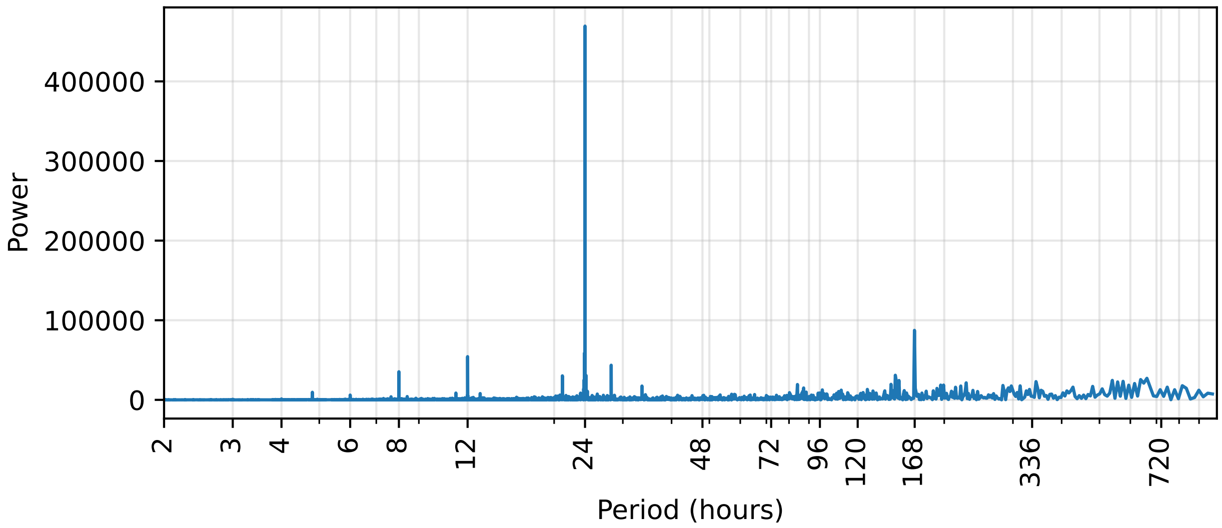


Figure 7: The periodogram of the PM10 data is shown.

only hypotheses which need further testing and more data to be tested. As an example, testing a possible correlation between traffic and this air quality data could be rather interesting. Ozone and nitrogen dioxide data show similar cycles as the PM10 data, although ozone has a much more pronounced 12-hour cycle. This could be due to how ozone is created in the atmosphere.

As a last exercise, a 2-dimensional Sammon plot has been created. As the underlying algorithm tries to keep distances (especially small ones) similar when encoding, one expects that close data points remain close even after the mapping to 2D. I have tried many different settings and plotting color scales for all different columns of the dataframe, however after encoding no clear clusters emerged from the data. Instead, the map produces quite spread out, seemingly random points, indicating low clustering in the data. Through changing the z-score value, I was able to verify that a smaller z-score can be used to remove more outliers.

## Reflection

I found this weeks material to be quite interesting again. Especially the topics about dimensionality reduction were quite new to me. Handling time series data has been something I have done a lot during my prior studies, so I was quite familiar with Fourier transforms and correlations between signals already, still it was nice seeing it applied again in a slightly different manner. However, I found the exercises this week to be a bit too long. Instead of having this many tasks I feel like it would have been more interesting to focus on a few applications specifically and go more into depth about them. Besides the one missing function implementation (see Listing 3), nothing needed to be programmed. For me, the concepts of this week would have been better internalized, if I had been confronted with the (maybe already cleaned and sorted) dataset, and were asked to perform simple steps like a Fourier transform myself, i.e. if the provided notebook would serve more as a guide and less as finished implementation of the data mining process. However, I see that this may be difficult to realize due to different programming backgrounds of the students. I also think, that fewer questions would have sufficed, as it is rather difficult to answer them all on only 2 pages or so. For me this is especially true as the questions often dealt with the specific air quality dataset at hand which while interesting, should not be the main focus of the exercise in my opinion. Instead, it would have been more illustrative to learn about the Data Mining tools themselves more, by programming a bit on my own or needing to compare different approaches, as these tools can be applied to many forms of data.

This week I managed to structure my learning better again. I did a little each day, which definitely helped my understanding and in answering the questions for the exercises. However, I underestimated the amount of time needed for the exercises by a lot this week. As they were a fair bit longer this week, it got quite tight to the end and I needed to work quite a lot two days. Next week I should try to plan more time for the exercise. However, I am quite happy with my progress in general, as I up to now succeeded in working through the lectures, exercises and writing the diary each week without procrastinating.

# 5. Clustering in Python

## Lecture

This week's lecture dealt with clustering data. Clusters represent continuous regions in a $p$-dimensional space with high data density, separated by regions of lower density. They are characterized by their density, variance, shape, and separation. Clustering is needed to discover patterns in the data, compress it, or detect anomalies. We discussed 3 different algorithms this week and the need for validation. These algorithms belong to the following classes:

- **Sequential algorithms**: These are usually quite fast and process points in their order of appearance. We discussed nearest neighbor (NN) clustering.
- **Hierarchical algorithms**: These build a dendrogram (a tree of nested clusters by either agglomerating or dividing the data into groups.
- **Optimization algorithms**: These define a cost function that is iteratively optimized. We discussed k-means. These algorithms are usually well scalable and suited for large datasets.

All algorithms work differently and have their up- and downsides, which are discussed in the following. I focus here on giving a more qualitative view, as most algorithms are already implemented in Python libraries, and it makes understanding much simpler.

The core idea of NN-clustering is that points that are close in space should belong to the same cluster. The procedure is as follows:

1. Present vectors one at a time.
2. For each new vector $\mathbf{x}_i$, find nearest already clustered point $\mathbf{x}_m$.
3. If $d(\mathbf{x}_i, \mathbf{x}_m) \leq \tau$ (threshold), assign to that cluster. Otherwise, create a new cluster.

One can use different metrics to quantify distance, but usually the Euclidean distance is used. The strength of this algorithm is that it is easy, fast and one does not need a predefined number of clusters. However, the result is strongly dependent on input order, highly sensitive to the chosen threshold and tends to lose the global view of the data as only local similarity is used in the metric. It is also prone for chaining together nearby data points and bridge different clusters.

The main idea behind hierarchical clustering is to build a dendrogram based on pairwise distances. Our discussion focused on the agglomerative approach where starting with singletons more and more clusters are merged based on distances. The algorithm is as follows:

1. Compute initial distance matrix $D$.
2. Repeatedly merge the two clusters with minimum inter-cluster distance.
3. Update distance matrix using chosen linkage criterion.

There are different linkage criteria such as single, complete and average linking. The strength of hierarchical clustering is that it produces a full hierarchy instead of single partitions. However, it is computationally quite expensive, it cannot undo early, unideal merges, and is sensitive to the chosen distance metric and linkage choice. To me, it seems, that it may be a sensible algorithm as a first choice for small datasets as the full structure may offer additional insights that can be explored further.

The last algorithm we discussed is the k-means algorithm, an optimization based approach, meaning that the data is partitioned into $k$ groups by iterative optimization. It works roughly as follows:

1. Initialize $k$ centers (randomly).
2. Assign each point to the nearest center.
3. Update centers as means of assigned points.
4. Repeat until assignments no longer change.

It is comparably fast and scalable to large datasets and produces compact clusters when data is roughly spherical. The algorithm is however not well suited for data that is not spherical, struggles with overlapping groups and outliers and is sensitive to initialization. Additionally specifying the number of clusters beforehand is required. This can be however optimized during the clustering validation, which we discussed next.

Validation of the clustering result is extremely important as the procedures are unsupervised and usually require user inputs like the amount of centers $k$, linkage criteria or distance metrics. It is therefore important to find out whether clustering worked well or not. Validation can be done externally (if known labels are present), internally (measure compactness statistically) and relatively (compare clustering obtained by the same algorithm for different parameters). We focused on the relative scheme, which to me also seems to be the most applicable as it can be used for all data regardless of labels. For algorithms requiring some input number $n_c$ (for example $n_c = k$ for k-means) validation looks like this:

1. Choose a range of numbers $n_c \in [n_{\min}, n_{\max}]$.
2. For each $n_c$, run the clustering algorithm $r$ times (different random initializations).
3. Compute a validity index for each run.
4. For each $n_c$, keep the best index value.
5. Plot the index value against $n_c$ and select the best.

The best number is usually the one where the index either changes sharply or has a minima or maxima. I find it very interesting, that using validation one can also find out, whether the data possesses cluster structure at all. If the index does not seem to change at all, this would suggest data that is not clustered. We discussed two different validity indexes. The Davies-Bouldin-Index (DB-index) shows the average similarity between clusters. Naturally, for the clustering to have worked well, one wants a minimum in the index. It is especially good for comparing clusters with different $k$ during k-means clustering as it captures spherical, separated clusters best. Additionally, one can also use the Silhouette measure which assesses how well each data points fits into its assigned cluster. It is ranged between -1 and 1 where a large positive value indicates that the point is much closer to its own cluster than to any other (good clustering), while a value of -1 means the opposite and that the point is probably misclassified. To compare clustering, one can use a Silhouette plot which visually shows outliers as negative values and whether some clusters might be overlapping or poorly defined. In practice, it is best to use both indices and look for an agreement between them.

## Exercise

This week's exercise was a small hands-on example of the workflow behind clustering data based on the `toolo.csv` dataset. After loading the data that was already used in exercise 1, we explored different ways of normalizing the entries first. For this we looked at variance, minmax, and Euclidean scaling (meaning dividing each row by its L2 norm). To see which pre-transform worked best, we then used different metrics. For a start the DB and Silhouette indices are calculated and tabulated. After this k-mean clustering with 7 clusters is employed to compare Sammon maps, box plots and Silhouette plots graphically. All three methods agree that Euclidean scaling is best suited for the dataset at hand. Firstly, the calculated Silhouette index is largest for Euclidean scaling and the calculated DB-index is smallest. Although the difference to the other normalization indices is quite small (roughly in the range of 0.1 for both) it is already a good indication, that Euclidean normalization might be best suited. Additionally, the obtained Sammon mappings show clearly, that the clusters are very tight together for minmax and variance scaling, while they cover a much larger parameter space in the mapping for Euclidean normalization. This can be seen in Figure 8. The distributions of the different variables seem to show a similar variability as well when using L2-normalization. Lastly, when looking at the Silhouette plots, one can see, that for Euclidean normalization nearly all scores are positive and clusters are well separated. For the following tasks, I will therefore use Euclidean normalization. The Silhouette plot also suggests that there may be some overlap between clusters 5 and 6, as they are less clearly separated than the other clusters, which can be seen in the attached plot (Figure 9). The Sammon embedding suggests mostly spherical/globular clusters, although there are also some outliers. All in all, I think k-mean clustering is well suited.

Having decided on preprocessing method and clustering technique, the next sensible step is to find the best number of clusters by comparing the resulting DB-indices. As can be seen in Figure 10, 2 clusters seem to work best. This is also true for different random initializations, the algorithm seems quite stable. As the code has been designed to work with the lowest DB-index and start the search with two clusters, I will therefore do all following analysis with 2 clusters. I will reflect if this is sensible in the end.

In the last step we looked at the temporal evolution of the clusters. As can be seen in Figure 11 exemplary, the 2 clusters can be easily understood. One corresponds to the periods where high emission of pollutants was present, while the other corresponds to low emission periods. This

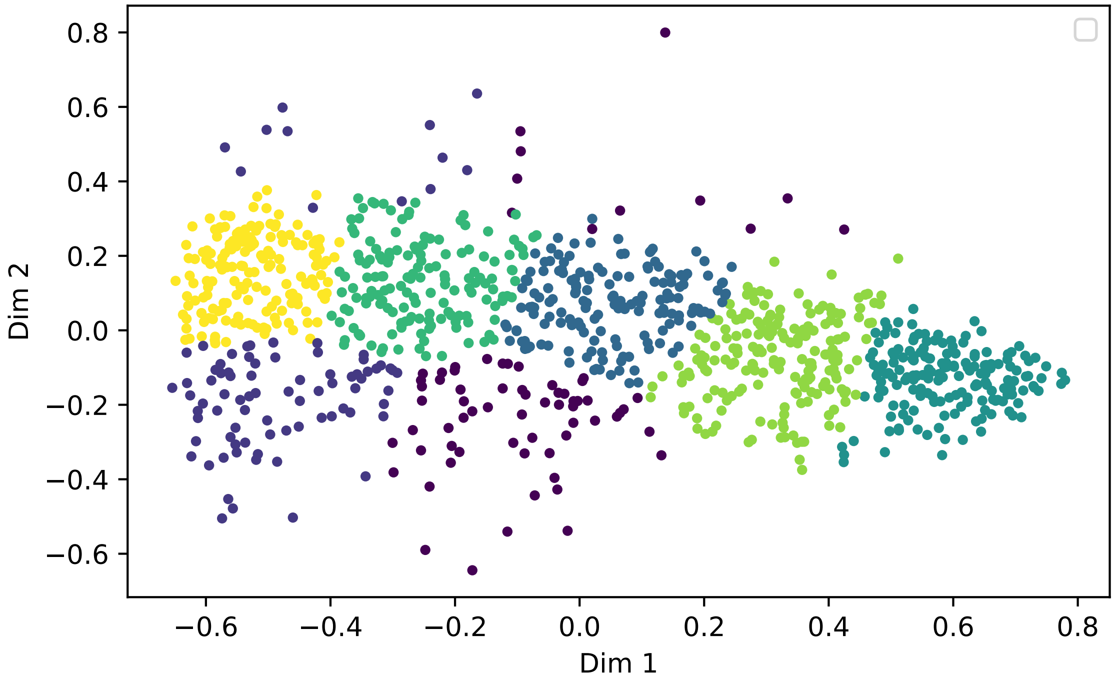


Figure 8: The obtained Sammon mapping of 7 clusters is shown for Euclidean normalization. Note that clusters are well separated and A large parameter space is used. This is not the case for other scalings.

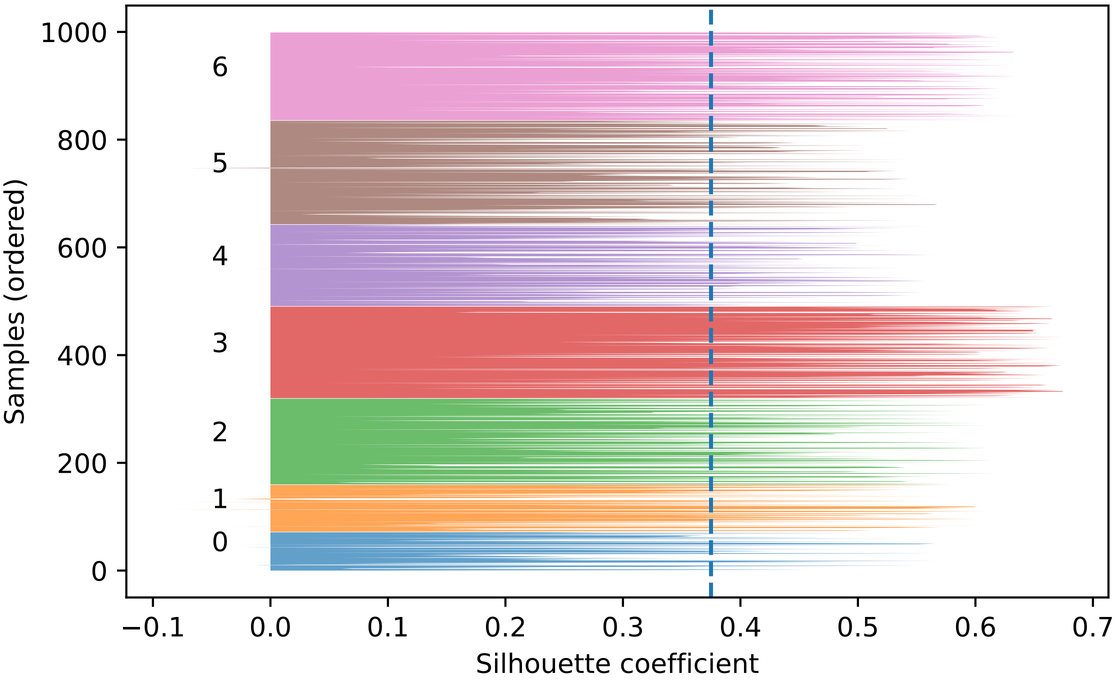


Figure 9: The Silhouette plot when using seven clusters and Euclidean normalization is shown.

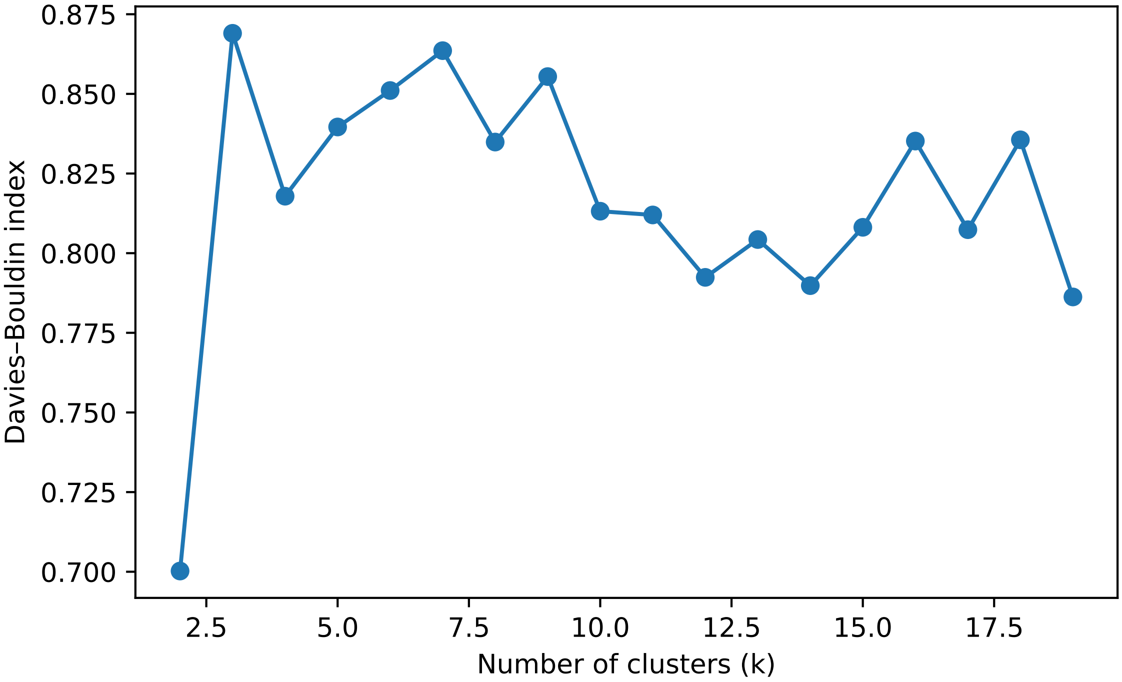


Figure 10: The resulting DBI for different number of clusters can be seen. $k = 2$ performs best.

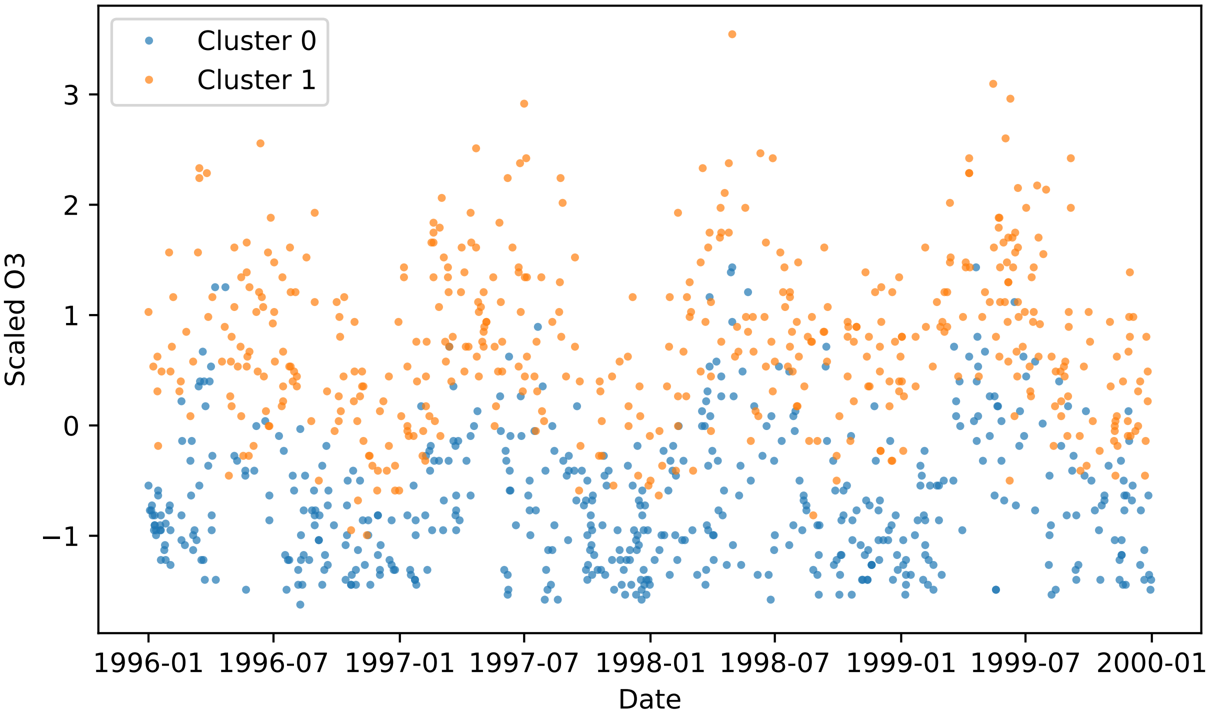


Figure 11: The temporal evolution of $O_3$ clusters can be seen. Trivially, having two clusters works well, as high-emission periods are separated from low-emission ones.

makes intuitive sense: If only two clusters are available periods where there are correlated high emissions (e.g. because of traffic) belong to one group, while the opposite belongs to the other. I think this traffic behavior is exactly what can be seen in the cluster time evolution. One could therefore give the clusters the labels “high emission” and “low emission”. From the analysis of just two clusters not a lot of insights can be gained. One could inspect times of high pollution and try to find policies against that by correlating these times with processes that might be happening. Further analysis, like the correlation with traffic data, is needed.

## Reflection

The lecture material this week was quite interesting, as it was something completely new to me. It felt like it fit well with the topics prior to and after that. The exercises were also a great addition to the lectures. I liked to be able to play around a bit with different normalizations and amount of clusters. However, I think that more interesting insights on the data could be gained when clustering into more than two groups is enforced. In the beginning I was quite surprised, that two clusters had such a low DN-score and performed best. However, after looking at the time traces and visualizing some data it became clear to me, why that is the case. Although one could maybe say that 2 clusters did not really provide much insight, this did still help me understand clustering and plotting the time traces quite a lot by working through my initial confusion. Learning and writing the diary was a bit hectic for me this week. AS my studies as an exchange student soon reach an end, I tried to fit more than one week of lectures and exercises into this week, which made it rather challenging. Still, I am quite happy with my progress.

# 6. Predictive modeling

## Lecture

In this weeks lecture we discussed validation for regression models or classifiers, performance indicators (both visually and numerically), linear regression and the basics behind neural networks (NNs), especially mulitlayer perceptrons (MLPs). As all topics have been a large part in prior courses I took, I will focus on giving a rough overview and highlighting approaches that are new to me and especially interesting.

We started by contrasting different validation approaches which is very important, as evaluating any model on data that has been used in *constructing* the model usually leads to a much too optimistic performance. This is true both for machine learning task (where I were quite aware that validation sets are important) but also when doing simple regression. Splitting validation data from the training set can be done in different ways. For example one can perform hold-out validation, which means simply splitting the dataset into training and testing sets. The training set is used for constructing the model, while the testing set is used purely for evaluation. This approach is simple and fast, but the model performance is usually underestimated as the data used was not used for training. Another approach is k-fold cross-validation, where data is split into $k$ clusters. Training is done on $k-1$ clusters, the remaining one is used for validation. Then the validation cluster is cycled through the whole dataset, such that in the end every cluster has been used for training and validation. The overall performance is then the mean of all cluster performances. As a special case of k-fold cross validation one can also divide the data in as many clusters, as there are data points. This is then called leave-one-out cross-validation. It is numerically more expensive and usually provides an estimate that is too optimistic, but may be the best approach when the data sets are small.

We then went on to talk about performance indicators to evaluate the models we construct. A very simple approach is to use graphical methods that compare the model output with the observations themselves. This can be done using scatter plots or simple line graphs where the model and data are plotted on top of each other. I additionally used an approach extensively for lab courses I did in the past which was not covered in the material. When using regression, one can look at fit residuals, meaning the difference between the fit and data value. If a fit describes the data well, these residuals should be centered around zero and have a spread described by a chi-squared distribution. If one instead wants to evaluate the performance of a model numerically, one could look at the mean error, mean squared error or root mean squared error. These are shown in Equation 11 from left to right.

$$\mathrm{ME} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i) \qquad \mathrm{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i| \qquad \mathrm{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2} \tag{11}$$

Here $y_i$ is a data point, $\hat{y}_i$ the model prediction and $n$ the total amount of data points. They all are of the same unit as the data itself and are therefore well interpretable. One can use the mean error to find the averaged signal deviation together with its direction, meaning one can find whether the model over- or underestimates the data. The MAE omits the directionality. Most often the RMSE is used however, as the squaring of the deviation has the nice property that large deviations contribute more to the error. One can also decompose the RMSE into statistical and systematic components. Other approaches we discussed where the coefficient of determination or $R^2$ value, and the index of agreement (IOA). The index of agreement is widely used, as it is unitless and lies between 0 and 1. It therefore allows for and easy comparison between different models. An index of agreement of 1 indicates a perfect model and one below 0.4 indicates worse performance than random guessing. It is best practice to look at the IOA and RMSE together to find out if a model performs suitably well.

Next we discussed linear regression as a means of fitting curves with some free parameters to data. The summary here is again more qualitative, as fitting algorithms are implemented in many different Python packages and can easily be used. The idea behind fitting is to minimize the sum of all squared residuals, i.e. find parameters $a, b, c, \dots$ such that the following expression becomes minimal:

$$\sum_{i=1}^{N} \left(\hat{y}_i\left(a, b, c, \dots\right) - y_i\right)^2 \tag{12}$$

Here $\hat{y}$ denotes the model output again. How this minimization is done in practice is quite interesting in and of itself, but will not be explained further here. Uses of linear regression are to fit for example linear functions in order to extract simple trends from the data. But more complicated functions like the sum of different harmonics with prominent frequencies extracted from e.g. a periodogram can also be fitted to clean the data of yearly, weekly or hourly trends before further analysis. When doing this complicated fit with necessarily many free parameters, one should be careful to always use model validation to see if there is overfitting. Lastly there are also autoregressive models which employ a sort of "memory" to handle time series data better. An expansion of this are so called ARX models, which can handle the time series structure of different data columns simultaneously. I have never actually worked with such models, but they seems incredibly useful for the time series data that is usually present in environmental science.

During the last part of the lecture we had a quick introduction to neural networks with the example of a multilayer perceptron. This is a densely connected neural network that has one input layer and an output layer. Each node in the vector is assigned a weight vector and each layer provides an additional bias. The feed forward pass through the network updates each neuron in layer $j$ by performing a weighted sum of all neurons in the previous layer $j-1$ as such:

$$x^{(j)} = \varphi\left(\sum_{\mathrm{n}} w_n^{(j-1)} x_n^{(j-1)} + b^{(j-1)}\right) \tag{13}$$

Here $\varphi$ is a nonlinear activation function, the sum ranges over all neurons labelled $n$ in the previous layer, weights are denoted as $w$ and the bias as $b$. After showing the network a sample (or a batch of samples), one trains the network to perform better. Training works by so-called gradient descent during backpropagation. The idea is that a so-called loss function, a function that shows the deviation of the networks output from its desired output, is differentiated with respect to the model parameters. These are then updated towards the direction of steepest descent, meaning the model is adjusted such, that the output gets closer to the desired output. Further details are omitted here again. For training neural networks, validation is extremely important, especially because they can become quite prone to overfitting. To mitigate this, one should also experiment with the number of neurons of the network and their dimensionality and not set it too high. From my experience the best way to advance when it comes to machine learning is just to experiment, try different architectures, cost functions, activations and number of free parameters and see what works best.

## Exercises

This week there were two exercises. The first dealt with creating a linear regression and ARX model and comparing them with each other as well as seeing how their performance change for different parameters. The second exercise deals with deep learning.

For the model comparisoin a finished notebook working with the `AQData.csv` dataset was provided. Only parameters should be adjusted and the models performances should be compared. First, data is loaded and a standardized, sorted date time column is created. The test and validation datasets are then split. One year is always kept for validation, this is varied as a first test to see the influence of the choice of validation data. Four different years are used and compared: 1996, 1997, 1998 and 1999. Then a linear regression model with cosine and sine terms is constructed to model the $NO_2$ concentration. The amount of harmonics is kept fixed for the first part, I used 24 yearly, 24 weekly and 12 daily terms. This model is fit to the data and the changing validation set and its performance is evaluated. The results are shown in Table 1. As expected, the model performance varies slightly between different test years due to random chance. However, this fluctuation is quite small, indicating that overall the years are rather similar, and the model was able to generalize well.

| Year | IOA (Train) | $P_{\mathrm{sys}}$ (Train) | RMSE (Train) | IOA (Test) | $P_{\mathrm{sys}}$ (Test) | RMSE (Test) |
|---|---|---|---|---|---|---|
| 1999 | 0.74 | 0.60 | 18.54 | 0.71 | 0.56 | 18.76 |
| 1998 | 0.73 | 0.61 | 18.48 | 0.74 | 0.58 | 17.98 |
| 1997 | 0.76 | 0.57 | 18.06 | 0.68 | 0.53 | 19.72 |
| 1996 | 0.75 | 0.60 | 16.97 | 0.66 | 0.74 | 22.63 |
| Mean | 0.75 | 0.60 | 18.01 | 0.70 | 0.60 | 19.77 |

Table 1: Index of agreement (IOA), proportion of systematic error ($P_{\mathrm{sys}}$) and RMSE for 1996–1999 and mean.

Next, the test year of 1999 is kept fixed, but the number of harmonics is varied. I performed two test, one where I halved the number of terms (12 yearly and weekly terms, 6 daily ones) and one where I used 50% more terms (36 yearly and weekly terms, 18 daily ones). Again performance tests are done and compared (see Table 2). The test performed above illustrated under- and overfitting clearly. Using too few terms the index of agreement is small, and the systematic error proportion is large, especially for the test data. This is due to the

| # Terms | IOA (Train) | $P_{\text{sys}}$ (Train) | RMSE (Train) | IOA (Test) | $P_{\text{sys}}$ (Test) | RMSE (Test) |
|---|---|---|---|---|---|---|
| 12, 12, 6 | 0.64 | 0.64 | 19.43 | 0.64 | 0.76 | 22.62 |
| 24, 24, 12 | 0.74 | 0.60 | 18.54 | 0.71 | 0.56 | 18.76 |
| 36, 36, 18 | 0.77 | 0.57 | 16.47 | 0.66 | 0.72 | 22.77 |

Table 2: IOA, $P_{\text{sys}}$ and RMSE for different numbers of terms.

model having too few parameters to suitably generalize and perform well. On the other hand, having too many parameters (bottom row) is also a problem as the IOA is again getting smaller with increasing systematic error. This is due to the model overfitting, i.e. losing the ability to generalize to unseen data because it could fit “too well” to the data it was trained on.

Next, an AR- and ARX model are built. As before, predictions should be made on the $NO_2$ concentration. The time lag was varied to see which one is best. For this different prominent peaks that were identified in section 4 are used as time lags, namely 8, 12, 24 and 168 hours. Additionally, an ARX model including also the ozone concentration is trained and tested using the same time lags. The results are shown in It can be seen that increasing the time lag leads to better models, as both the IOA is higher and the systematic error proportion lower. Additionally, including the ozone data also makes the model slightly better still, although it has less of an influence than increasing the memory to capture all prominent peaks. If simple models are preferred one should therefore use an AR model with sufficiently long memory to capture all strong variability. If one wants to create an even better, although more complex model, using an ARX model can be a good option. It should also be noted that fitting sine and cosine regressors leads to similar performance (when one keeps overfitting in mind and tries to mitigate it). This can therefore also be an option. Still, the full ARX model performs best both in terms of IOA and systematics and would therefore be my pick as the model to use. Not that the full tabularized test results showing also different metrics are available in [1] under `Code\topic_5\LR_ARX_results.md`.

| | Train $NO_2$ | | Test $NO_2$ | | Train $NO_2$+$O_3$ | | Test $NO_2$+$O_3$ | |
|---|---|---|---|---|---|---|---|---|
| Lag (hours) | IOA | $P_{\text{sys}}$ | IOA | $P_{\text{sys}}$ | IOA | $P_{\text{sys}}$ | IOA | $P_{\text{sys}}$ |
| 8 | 0.66 | 0.71 | 0.65 | 0.71 | 0.67 | 0.70 | 0.66 | 0.69 |
| 12 | 0.66 | 0.71 | 0.65 | 0.71 | 0.67 | 0.70 | 0.66 | 0.68 |
| 24 | 0.67 | 0.70 | 0.66 | 0.69 | 0.68 | 0.68 | 0.67 | 0.67 |
| 168 | 0.74 | 0.62 | 0.72 | 0.61 | 0.74 | 0.61 | 0.73 | 0.59 |

Table 3: Index of agreement (IOA) and proportion of systematic error ($P_{\text{sys}}$) for AR(X) models with varying time lags.

The second exercise this week dealt with a deep learning task using the same dataset.

## Reflection

# 7. Summary

# 8. Self-evaluation

# References

[1] Stephen Weybrecht. *Data Mining in Environmental Science: Repository.* `https://github.com/stewey0/uef-data-mining`. Source code. Accessed: December 7, 2025.

[2] Cmglee. *Comparison of Chebyshev, Euclidean and taxicab/Manhattan distances for the hypotenuse of a 3-4-5 triangle on a chessboard [image].* `https://commons.wikimedia.org/wiki/File:Minkowski_distance_examples.svg`. Image cropped. Licensed under CC BY-SA 4.0. Accessed: November 22, 2025. n.d.

[3] The scikit-learn developers. *Compare the effect of different scalers on data with outliers.* `https://scikit-learn.org/stable/auto_examples/preprocessing/plot_all_scaling.html`. sklearn documentation. Accessed: November 22, 2025.