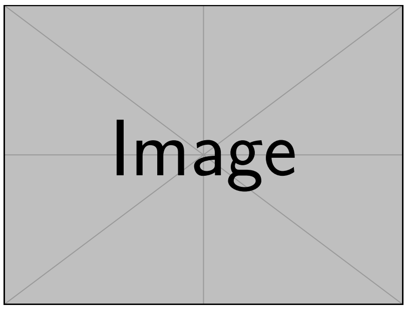
Autumn semester 2025



Data-Mining in Environmental Science

Learning diary

Date:	November 28, 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 μ and population variance σ^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 μ its mean and σ 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:

$$NO_2 : 38.4 \pm 23.2$$
$$O_3 : 37.0 \pm 22.0$$

This makes clear, that the nitrogen dioxide measurement has a larger variability than the 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 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 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.

units?

units

unit

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

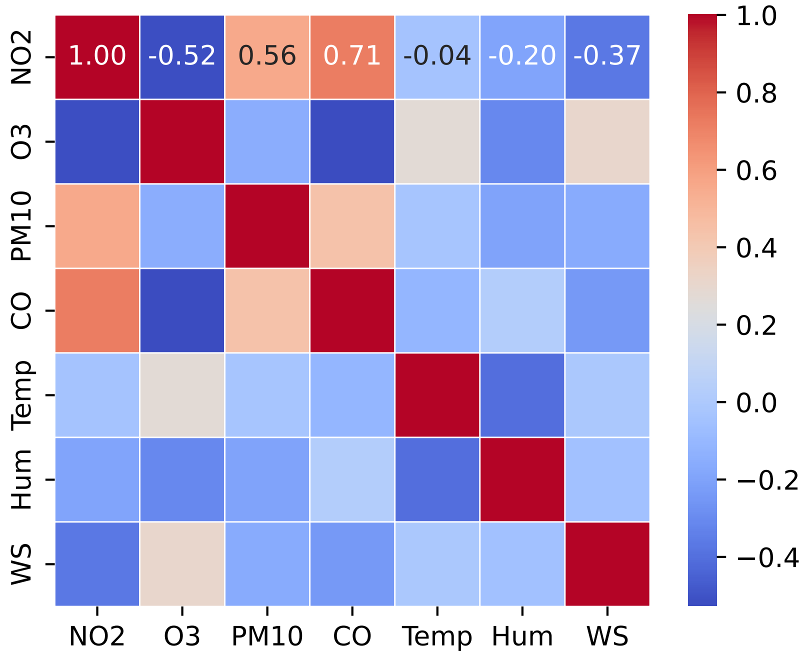


Figure 1: Cross correlation matrix calculated using the mentioned columns of the `toodo.csv` dataset.

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 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

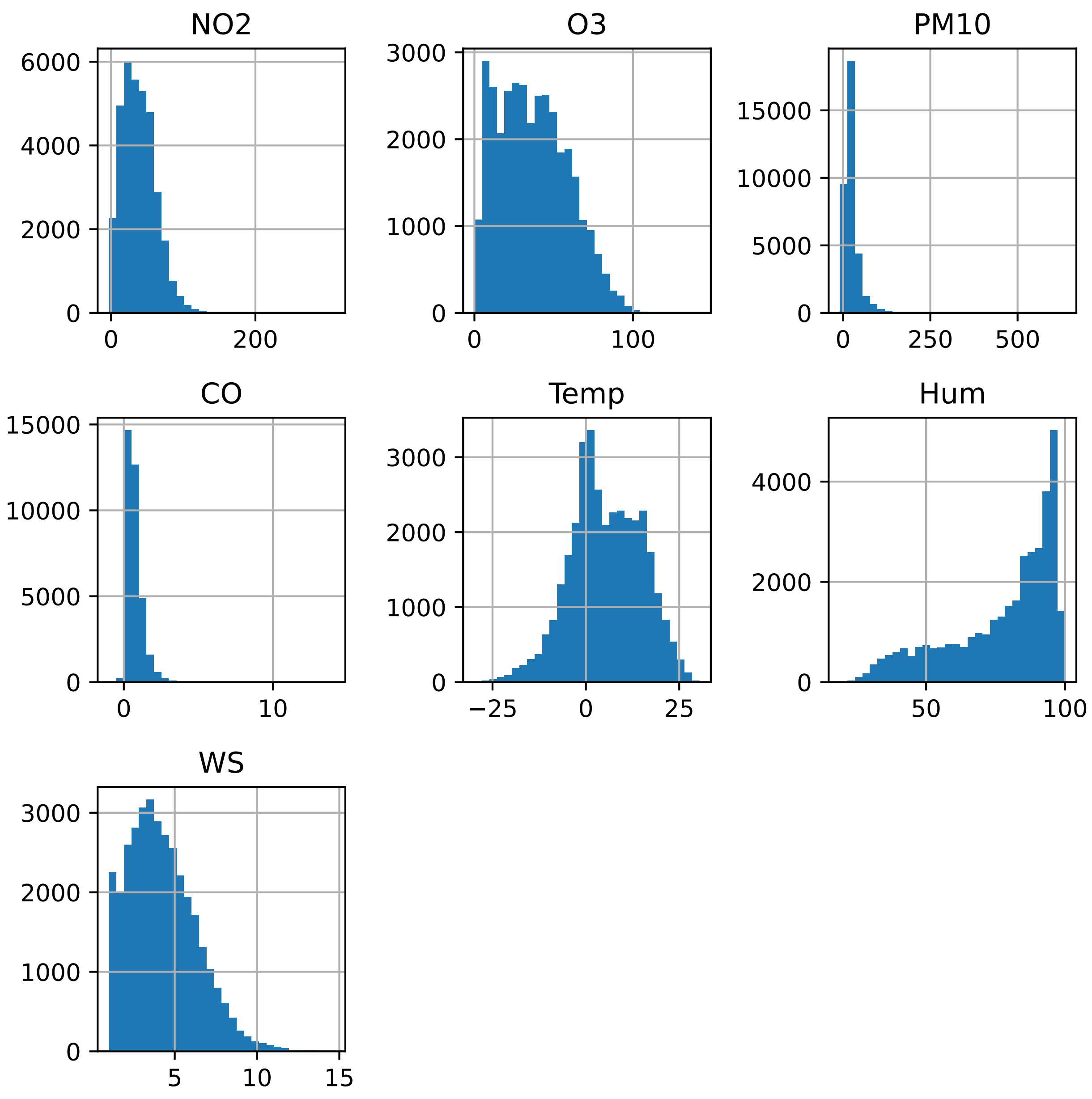


Figure 2: Histogram of the data columns of the `toodo.csv` dataset.

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 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.

```
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 $\Rightarrow$ can alter the distribution of data significantly
- Imputation by linear interpolation $\Rightarrow$ 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_{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 $\Rightarrow$. 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

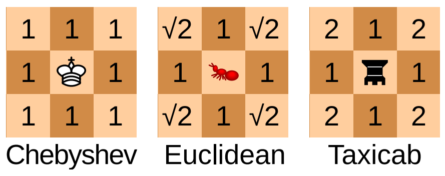


Figure 3: Different cases of the general Minkowski distance illustrated in 2D with the analogy of chess pieces. Taken from [2].

for class-like or binary data.

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 5×5 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

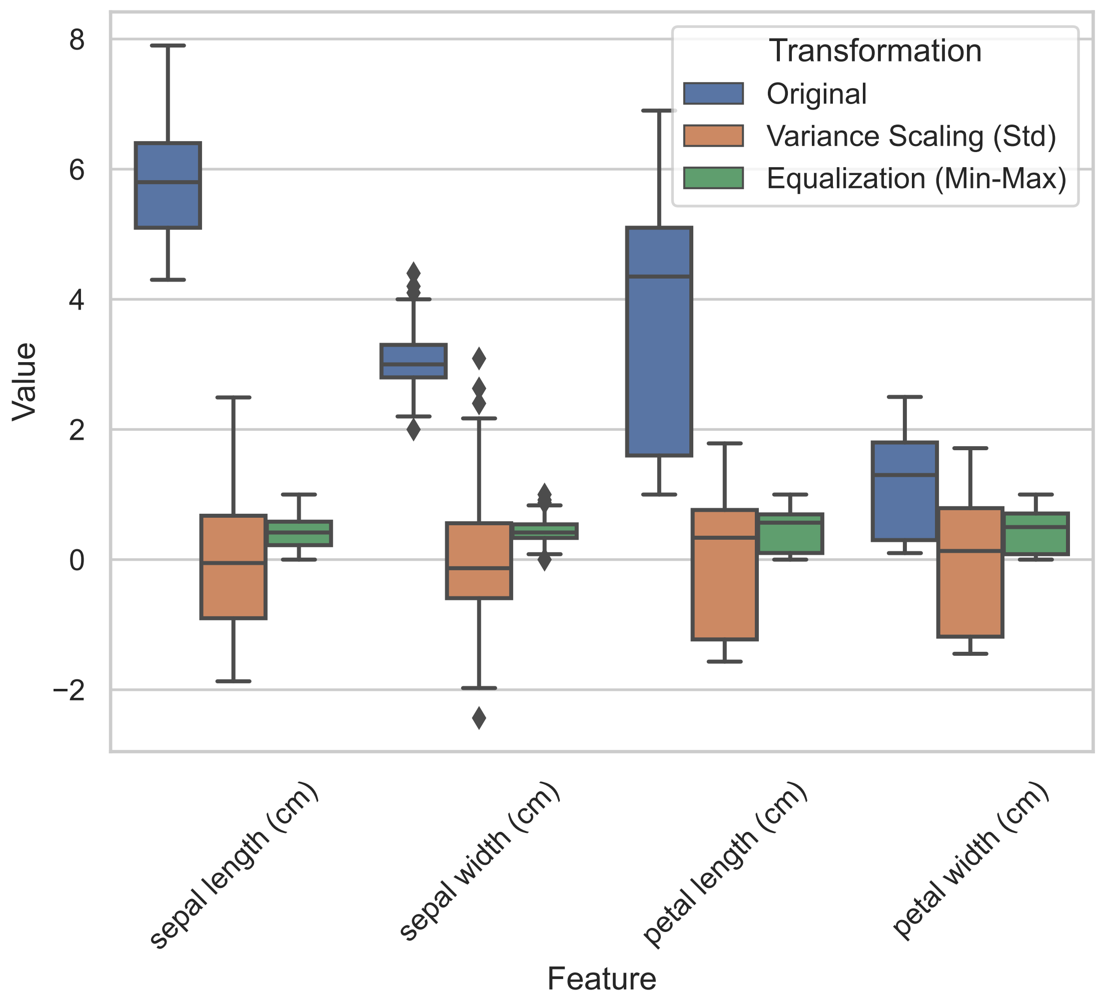


Figure 4: The original, variance scaled and equalized data of the `iris` dataset is shown as boxplots.

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 η 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

1. Import and prepare the data

- Air quality data
- Create standardized datetime, cut columns

2. Describe distributions and outliers Histograms:

- Distributions look more or less normal, but ofc cut, if detector cant meas below 0
- Maybe not normal: PM10, humidity
- Outliers: large No2, PM10, CO values
- No2 much more variable than PM10 which is very peaked with a lot of outliers
- Changing bin number: many bins oversamples data and fine variability appears: very prominent for 03 100 bins. Harder to tell trends as well as low smoothing. Using less bins averages to much and too much detail is lost eg 10 bins C0. 30 quite reasonable

Boxplots:

- Largest IQR is given by 03. It shows high variability in histograms as well. why
- Highest median: Humidity, can also be seen in histogram. Seems to be almost exp. peaked at 100
- Most outliers at PM10, then NO2, as expected from histograms
- NO2 has more spread (IQR) but less extreme outliers than PM10

Scatter plots:

-

3. Explore temporal patterns

- a

4. Study relationships between variables

- a

5. Analyse cycles and dominant frequencies

- a

6. Use dimensionality reduction

- a

Reflection

5. Clustering in Python

6. Predictive modeling

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: November 28, 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.