Algorithmic Pattern-Detection in Multivariate Outliers

Dominik Pichler

Matriculation Number: 03715311

Bachelor Program: Management and Technology

TUM School of Management

Supervising Chair: Chair of Psychology

Technical University of Munich

Grader: Prof. Dr. Hugo Kehr

Person of Support: Cafer Bakaç

Submission Date: 30.3.2022

Abstract

In this thesis, I have developed a mathematical algorithm that can detect patterns in multivariate outliers to help researchers and scientists understand the driving mechanisms behind the outliers in their data sets. While some suitable algorithms already exist to detect those outliers, the amount and quality of information that can be algorithmically obtained about the origins and mechanisms behind such outliers are still limited. This is the point at which the algorithm presented in this thesis is supposed to provide help. In order to develop such an algorithm, I have mainly focused on existing modern machine learning algorithms ranging from standard clustering to t-SNE, autoencoders, and the Fourier analysis.

After a fundamental analysis, I have intertwined a selected group of existing algorithms into a versatile pattern detection algorithm capable of handling low- and high-dimensional data sets.

Eventually, in a small test case, the final algorithm successfully detected manually added patterns and differentiated between interesting and seemingly random outliers.

Keywords: Pattern detection, Machine Learning, Fourier Analysis, Outliers

Algorithmic Pattern-Detection in Multivariate Outliers

Contents

Introduction

In today's world, scientists and researchers seemingly profit from an ever-growing amount of (readily) available data for all kinds of research, ranging from biology to psychology or even linguistics. While this mostly comes with significant advantages, it does not necessarily mean that scientists and researchers are automatically better off when they have access to relevant data that they can use for their research purposes (Simmons et al., 2011). One reason for this apparent contradiction can be found in the existence of so called *outliers* (Leys et al., 2019). To quote Hawkins (1980), outliers can be defined the following way:

> The intuitive definition of an outlier would be "an outlier is an observation which deviates so much from the other observations as to arouse suspicions that it was generated by a different mechanism." (p.1)

For further illustration, one could imagine a survey that asks its participants about their average monthly income. Presumably, most participants will answer somewhat honestly or exaggerate only to a modest degree, but substantial problems will arise if an (even tiny) subgroup overemphasizes their income to an extreme degree. For example, the entire calculation of important statistic metrics like the mean, the median, or standard deviation will be entirely off. This could mistakenly lead the researchers/scientists to wrong conclusions (Simmons et al., 2011). The case above illustrates an important point, namely that scientists and researchers need to pay attention to the detection of outliers in their data sets, as sometimes nonsignificant results can be caused by outliers (Leys et al., 2019).

While there already exist some suitable (software) solutions to robustly detect outliers in multivariate datasets, for example, algorithms based on a robust version of the *Mahalanobis distance* published by Leys et al. (2018), the amount and quality of information that can be algorithmically obtained about the origins and mechanisms behind such outliers are still limited. Nonetheless, a fundamental understanding of those hidden

mechanisms can be beneficial. It can help researchers and scientists gain deeper insights into their data sets and improve their outlier detection methods. In the example from above, this hidden mechanism could be a particular personality trait that all members of the extreme-exaggerators have in common, leading all of them to overstate their salary.

Consequently, with this thesis, I try to create an algorithm that can reliably discover such hidden mechanisms with the help of modern pattern detection algorithms. Thereby I aim to help researchers and scientists to gain deeper insights into the origins and mechanisms behind the occurrences of outliers in their data sets. In order to do so, some further definitions of outliers and patterns need to be made first.

Outliers

While the mechanisms behind the occurrence of outliers can be manifold, not all of those mechanisms are of interest to researchers and scientists. Sometimes, outliers could arise purely by errors in the data collection, or, occasionally could even occur randomly, without any relevant mechanism behind them. Therefore outliers can (and should) generally be grouped into different classes of outliers: (a) *error outliers*, (b) *interesting outliers*, and (c) *random outliers* (Leys et al., 2019).

Due to the limited scope of this thesis, I am mainly focusing on the so called interesting outliers, because their occurrence is by definition linked to other, initially unknown, non-random variables (Leys et al., 2019). In all later sections, the term outliers will therefore refer to interesting outliers.

Patterns

Following the definition of Concise Oxford English Dictionary, as cited in Choudrey (2002) patterns can generally be defined in the following way:

> The tenth edition of the Concise Oxford English Dictionary defines a pattern as 'an arrangement or sequence regularly found in comparable objects or events'.

> Implicit in this definition is that said objects or events (or, similarly, observations) are not identical, but share fundamental characteristics that make them comparable.(p.2)

This is a suitable definition for this thesis and will be used throughout the following chapters. For better readability and understanding, the set of shared fundamental characteristics in a comparable group will be denoted as $C_i = \{f_{i1}, f_{i2}, \dots\}$ with all measured characteristics as $F = \{f_1, f_2, \dots, f_m\}$ where $C \subseteq F$ must hold true.

In the following, it is assumed that patterns are the best available concept to identify and unmask the hidden mechanism behind outliers, as they can be used to group observations and thereby highlight similarities and dissimilarities between observations.

Statistical Pattern Recognition

In order to be able to develop an mathematical algorithm that can detect patterns, it is first necessary to transform the definition of patterns that was introduced above, into a mathematical framework. This is where *Statistical Pattern Recognition* (SPR) can be useful. SPR assumes that any objects of interest can be characterized by an (in-)finite number of variables, the so called features F (Choudrey, 2002). Those features could be all kind of quantitative measurements like for example the bodyweight and the height of a human being that, mathematically speaking, span an m-dimensional vector space ϑ^m where m equals the number of observed features F of an object and $\boldsymbol{X}$ contains all observations $\vec{x_1} \dots \vec{x_m}$:

$$\vec{x_i} \in \boldsymbol{X} \; where \; \vec{x_i} = \begin{bmatrix} x_{i1} \\ x_{i2} \\ \vdots \\ x_{im} \end{bmatrix} \tag{1}$$

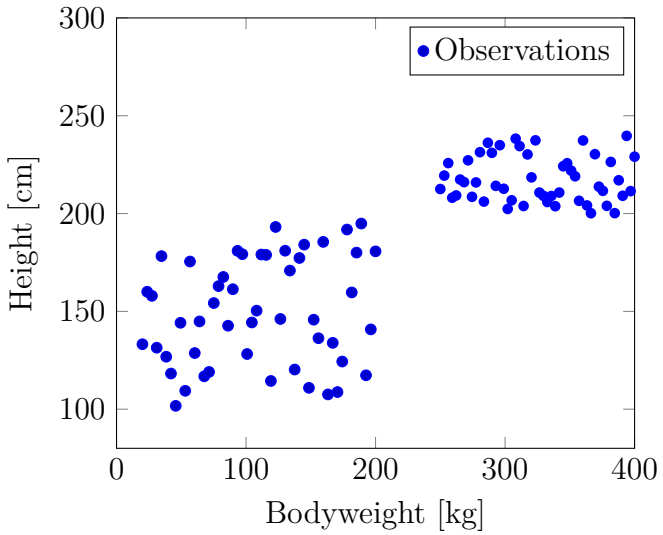
The resulting dimensions form a basis that allows (algorithms) to group objects into different sub-groups/classes by using various mathematical methods introduced in later sections. For further illustration, one could imagine a study that, measures the bodyweight and height of the participants and uses both variables as individual features (and thereby as dimensions):

$$\vec{x_i}^{\intercal} = [bodyweight(Person\ i), height(Person\ i)] \tag{2}$$

Using this notation, all observations could then be plotted in a 2-D vector space as depicted in Figure 1.

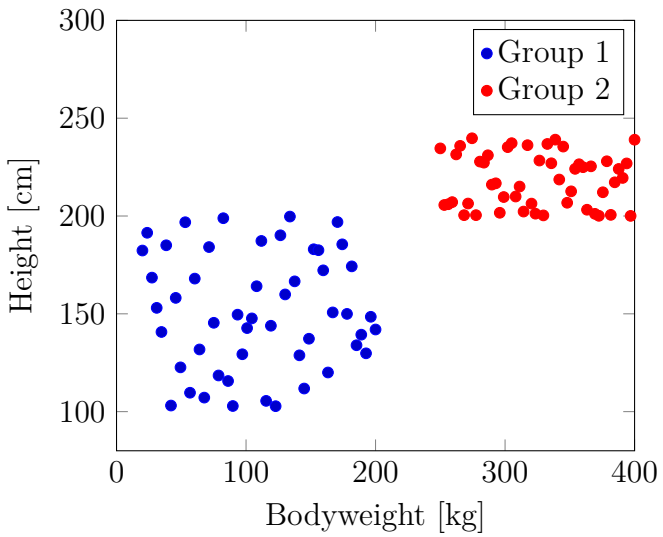
Figure 1

Vector space with two features



After looking at the vector space spanned by the two features, one could already notice that all observations could be intuitively organized into two different groups in which all members share similar attributes. In this case, the fundamental characteristics of members in group one (and thereby the underlying pattern) could be that all group members weight less than 200kg and are smaller than 200cm:

Figure 2
Patterns in a 2-D Vector space



Suppose the number of features (and thereby dimensions) is lower or equal to three. In that case, it might be a manageable task to group and maybe even classify the depicted objects manually. However, for higher dimensional spaces ($n > 3$) or more complex data, the human brain capacity will set the endeavor to halt pretty quickly. That is where novel algorithms are needed. Therefore, the next section will introduce some of the most prominent/practical pattern detection algorithms.

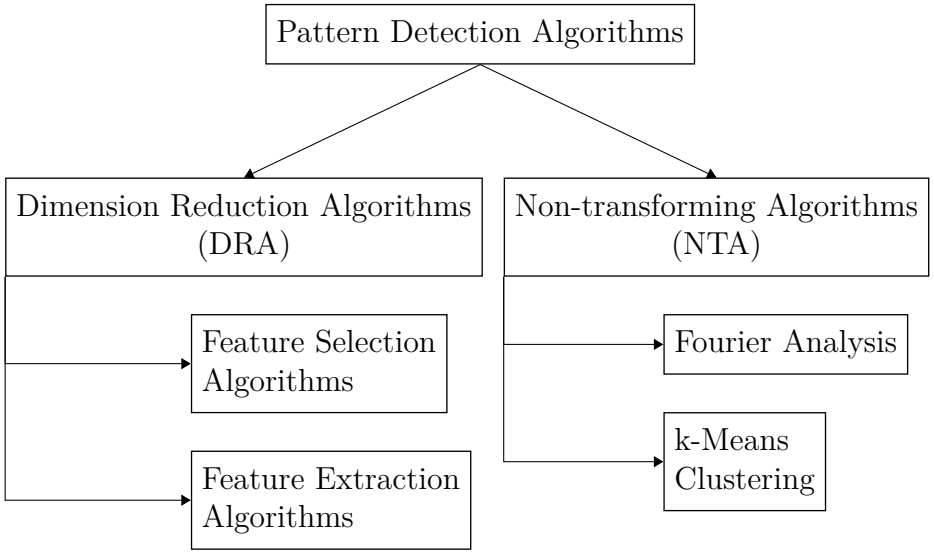
Pattern Detection Algorithms

Thanks to the enormous growth in science and technology, there are currently a lot of different algorithms available that can help researchers and scientists to find hidden patterns in (multivariate) data sets. The methods at hand range from mathematically complicated neural networks to simple clustering algorithms and might overwhelm researchers and scientists looking for quick ways to find patterns in their (outlier) data. In order to help to navigate around potential moments of confusion, this section intends to provide a quick overview of some of the most prominent/practical ones among all those algorithms.

For simplification, the set of the available algorithm has been divided into two different groups of algorithms, namely, those that reduce the number of dimensions and transform the original features and thereby create new vector spaces: *dimensions reduction algorithms* (DRA) and those that work within the original vector space of the observations: *non-transforming algorithms* (NTA).

Figure 3

Taxonomy of the types of pattern detection algorithms used in this thesis



Dimension Reduction Algorithms

As discussed in previous chapters, I assume that objects can be characterized by an (in-)finite number of variables called features, which will span an m-dimensional vector space ϑ^m where m equals the number of observed features. While the original set of features might sufficiently describe the object(s) of interest in many cases, this does not necessarily hold for all cases, primarily when one deals with outliers. Outliers are particularly intriguing because the initially defined set of features often does (by definition) not sufficiently explain their occurrence.

Maybe some outliers do not appear to make any sense because researchers are looking at them with the wrong perspective, meaning the use of unhelpful or insufficient sets of features to describe the outlier.

As introduced earlier in Figure 3, I have further split the DRAs into two groups, on the one hand, the group of *feature selection algorithms*, and on the other hand, the group of *feature extraction algorithms*.

Feature Selection vs. Feature Extraction

While feature selection algorithms do not transform any of the original data from the original vector space ϑ^m to a new vector space μ^n, but rather span a subspace ϑ'^t of ϑ^m through defining a subset of relevant features F' from the original set F, where $F' \subseteq F$ holds true, feature extraction transforms the original observations from the original vector space ϑ^m into a newly created vector space μ^n where in most cases $F' \not\subset F$ holds true.

Feature Selection

Although in general, a very helpful group of algorithms, I assume that feature selection algorithms are not of dominant utility for detecting patterns in multivariate outliers. For completeness, feature selection algorithms are mentioned here, but due to the limited scope of this thesis, they will not be used or discussed any further.

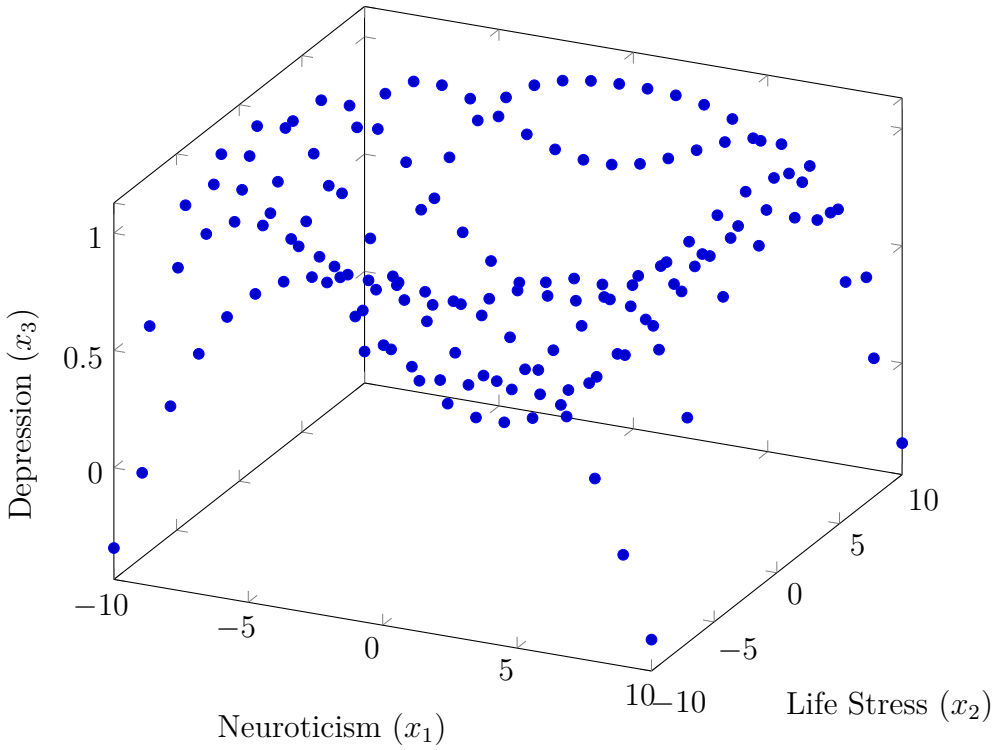
Feature Extraction

Feature extraction rests on the idea that the original dimensions and features of a data set might not be sufficient in explaining the mechanism that hides behind those observations and that other dimensions/features exist that provide more and deeper insights into the data set at hand (Webb, 2002). Therefore the general aim of dimension reduction algorithms is to find those features that can potentially provide deeper insights.

For a better illustration, one could for example imagine a study that gathers data about the degree of depression, neuroticism and life stress in the life of each of their participants, in order to better understand the connections between each of those concepts.

Figure 4

Visualization of a data set with three features



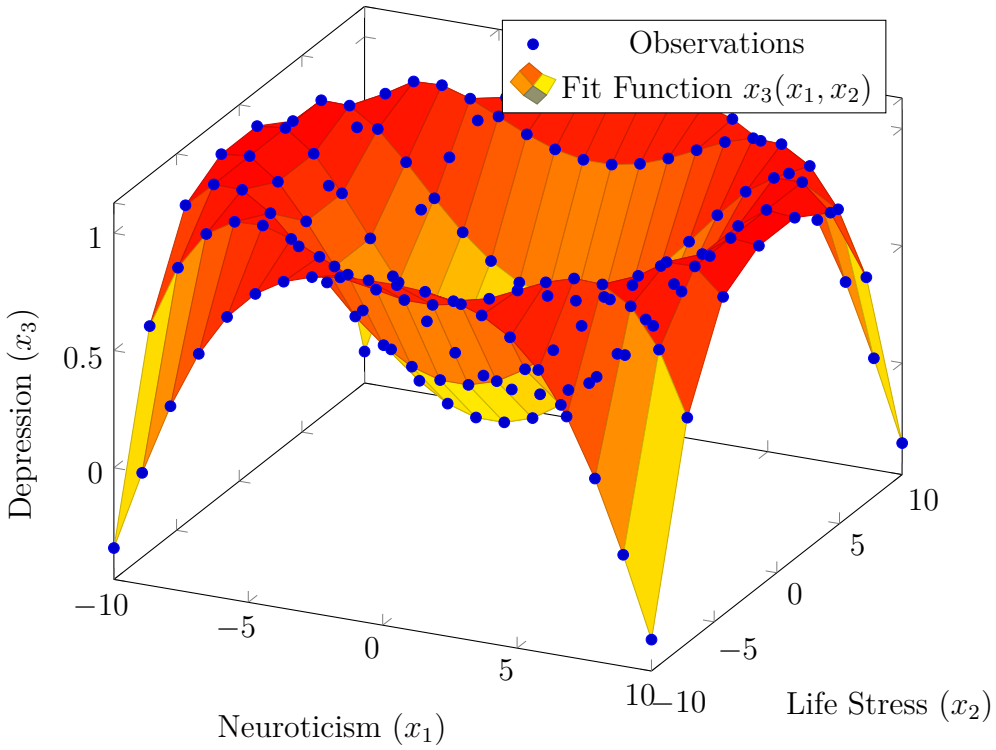
While the plot of the resulting data in Figure 4 does not reveal any clear underlying structure or relationships between the three variables x_1, x_2 and x_3, the application of a dimension reduction algorithm could identify such structures. For example, in this case, the insight generated through a DRA could be, that all of the observations are located on a surface that can be modeled through Equation 3.

$$x_3 = sin(x_1^2 + x_2^2) \tag{3}$$

For further illustration, this case is depicted in Figure 5.

Figure 5

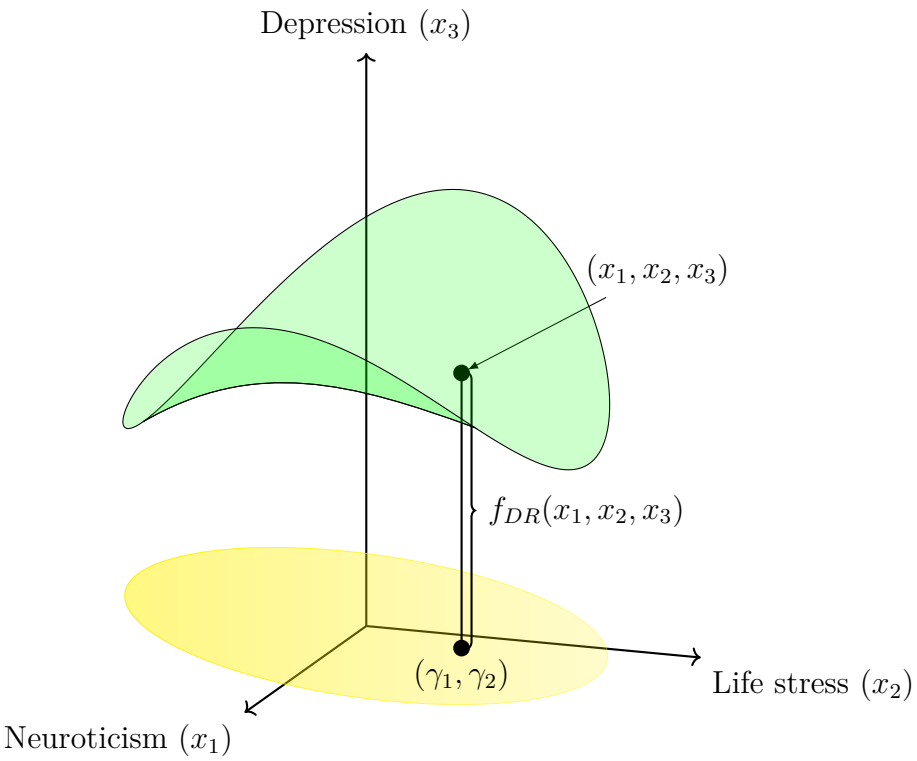
Dimension reduction through non-linear fitting



The potential gains of such a dimension reduction can be at least twofold. Firstly it can provide deeper insights into the underlying structure and thereby into the relationships between each variables which might be used to come up with new hypothesis. For example, Equation 3 could provide a ground for a hypothesis on the impact of neuroticism and life stress on the level of depression in humans. Secondly, the use of DRA can identify new sets of features that highlight similarities and dissimilarities between (groups of) observations more clearly. This is usually done by a coordinate transformation, as illustrated in Figure 6.

Figure 6

Example of an dimension reduction from 3-D to 2-D

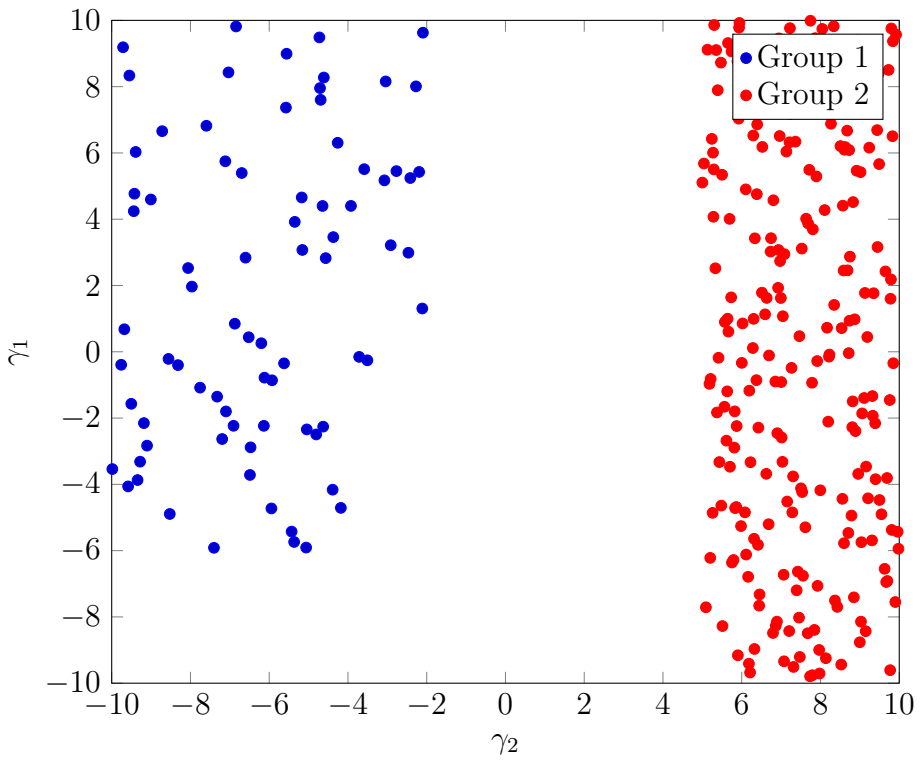


In Figure 6 (γ_1, γ_2) constitute a lower dimensional representation of (x_1, x_2, x_3) that depicts similarities and dissimilarities from the original high dimensional space (x_1, x_2, x_3). This representation (γ_1, γ_2) can be generated through the application of Equation 4.

$$f_{DR} : (x_1, x_2, x_3) \rightarrow (\gamma_1, \gamma_2) \tag{4}$$

The application of the function $f_{DR}(x_1, x_2, x_3)$ to the original, high dimensional data $\boldsymbol{X}$ could then yield clear groups of similarities (patterns), as shown in Figure 7, where γ_1 and γ_2 represent some (yet) unknown features.

Figure 7
Patterns in γ_1, γ_2



Eventually, the identification of the most suitable function $f_{DR}(x_1, x_2, x_3)$ and the subspace $\boldsymbol{\gamma}$ constitutes the main objective of every dimension reduction algorithm. While the group of feature extraction algorithms contains an ever-growing and tremendous amount of different and handy algorithms, this thesis can only introduce some of the most prominent/praised ones due to the limited extent.

Principal Component Analysis (PCA)

Principal component analysis (PCA) goes back as far as 1901 (Pearson, 1901) and is according to Delac et al. (2005) the most widely used subspace projection technique. Jackson (1991) defined PCA in the following way:

> The method of principal components is primarily a data-analytic technique that obtains linear transformations of a group of correlated variables such that

> certain optimal conditions are achieved. The most important of these conditions is that the transformed variables are uncorrelated.(p.1)

When applied to a data set with m Features $F = \{f_1, \ldots, f_m\}$, the PCA algorithm looks for i new features to span a different vector space (where $i < m$). Those new features are called *principal components* ($\boldsymbol{\psi}$) and are, by definition, a set of normalized linear combinations of the original features F :

$$\psi_i = \phi_{1i} f_1 + \phi_{2i} f_2 + \cdots + \phi_{m_i} f_m \tag{5}$$

where $\boldsymbol{\phi}$ constitutes a matrix that contains the so call loading vectors that need to be determined. With PCA, the determination of loading vectors is done using a classic minimization problem, namely, the minimization of the total orthogonal euclidean distances between the PCs and the observations.

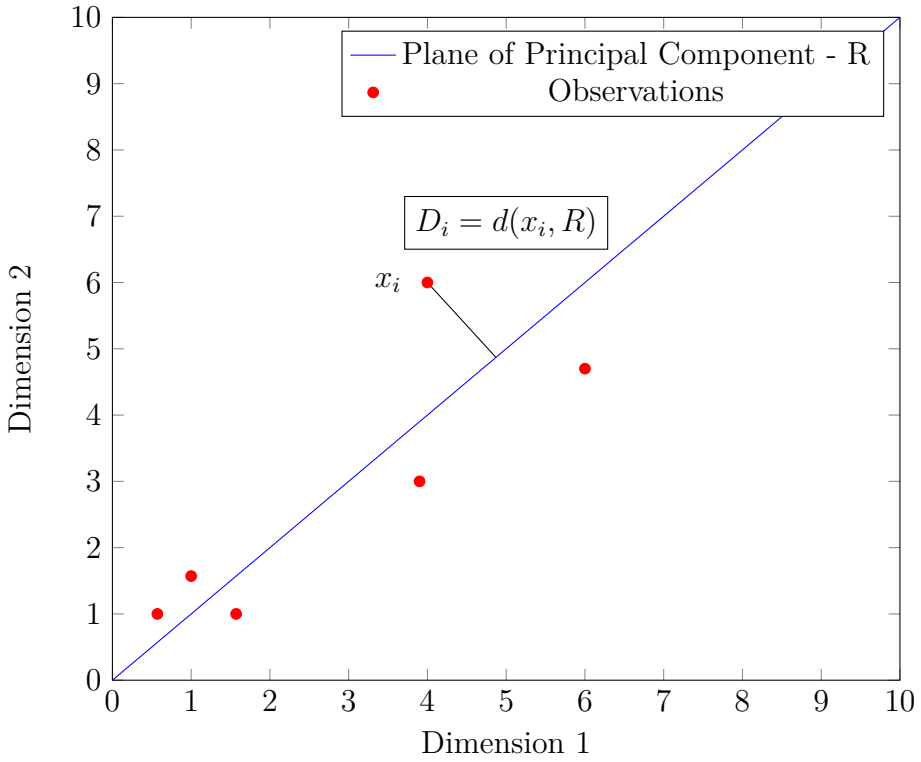
In order to achieve this objection, PCA starts by placing a new linear plane into the existing high dimensional vector space based on F and then starts to rotate this first plane R until the sum of orthogonal euclidean distances between each observation and the plane R reaches a minimum:

$$min(\sum_{n=0}^{N} d(x_n, R)) \tag{6}$$

This is further illustrated in Figure 8. Once the first principal component has been found (using Equation 6), all the other PCs are then, one after the other added orthogonal to all previously defined PCs until the total variance explained by the current set of PCs is sufficiently large.

Figure 8

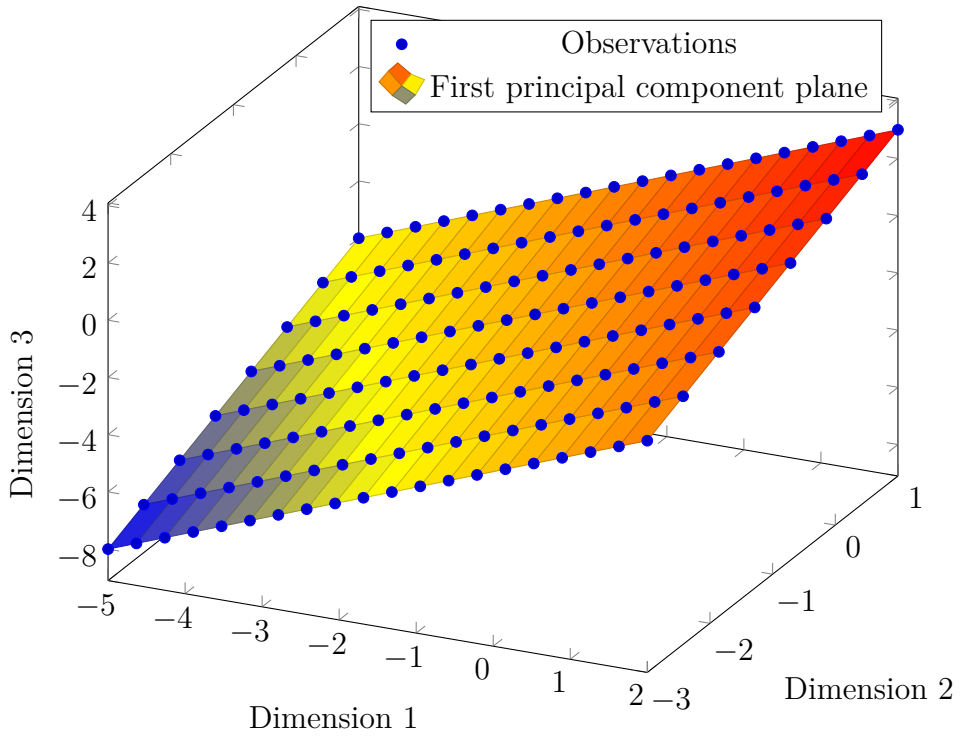
Illustration of Principal Component



To further illustrate, Figure 9 shows an example of a perfect fitting principal component where sum of orthogonal euclidean distances between each observations equals zero.

Figure 9

Dimension Reduction with PCA



Non-negative matrix factorization (NMF)

Non-negative matrix factorization (NMF) was introduced in the infamous paper from Lee and Seung (1999), that starts with the following paragraph:

> Is perception of the whole based on perception of its parts? There is psychological and physiological, evidence for parts-based representations in the brain, and certain computational theories of object recognition rely on such representations. But little is known about how brains or computers might learn the parts of objects. Here we demonstrate an algorithm for non-negative matrix factorization that is able to learn parts of faces and semantic features of text.(p.1)

As this quote already indicates, unlike other methods such as PCA or t-SNE, NMF learns part-based instead of holistically, meaning it identifies fundamental components (Lee

and Seung, 1999). One could imagine a data set that is represented by a Matrix $\mathbf{V}$ of size $n \times m$ where each column represents one feature, and every row represents a distinct observation. Now NMF algorithmically deconstructs the Matrix $\mathbf{V}$ into two Matrices of lower ranks $\mathbf{W}$ and $\mathbf{H}$ (low-rank matrix approximation) where $\mathbf{W}$ can be interpreted as a matrix of basic or fundamental elements and $\mathbf{H}$ as weights or coordinates that are needed to reconstruct the original element.

$$V_{i\mu} \approx \sum_{a=1}^{r} W_{ia} H_{a\mu} \tag{7}$$

The determination of the most suitable deconstruction can be realized through many different methods and is currently a hot topic among scientists and researchers. In the following, this thesis sticks to the prominent *frobenius norm*:

$$\|\mathbf{V} - \mathbf{WH}\|^2 F = \sum i, j(\mathbf{V} - \mathbf{WH})_{ij}^2 \tag{8}$$

and calculates both matrices $\mathbf{W}$ and $\mathbf{H}$ through solving the following minimization problem:

$$\min_{W \in \mathbb{R}^{p \times r}, H \in \mathbb{R}^{r \times n}} \|\mathbf{V} - \mathbf{WH}\|_F^2 \ \textit{such that}\ \mathbf{W} \geq 0 \ \textit{and}\ \mathbf{H} \geq 0 \tag{9}$$

Autoencoders

The relevancy of *autoencoders* can be highlighted by a quote from Lopez Pinaya et al. (2020):

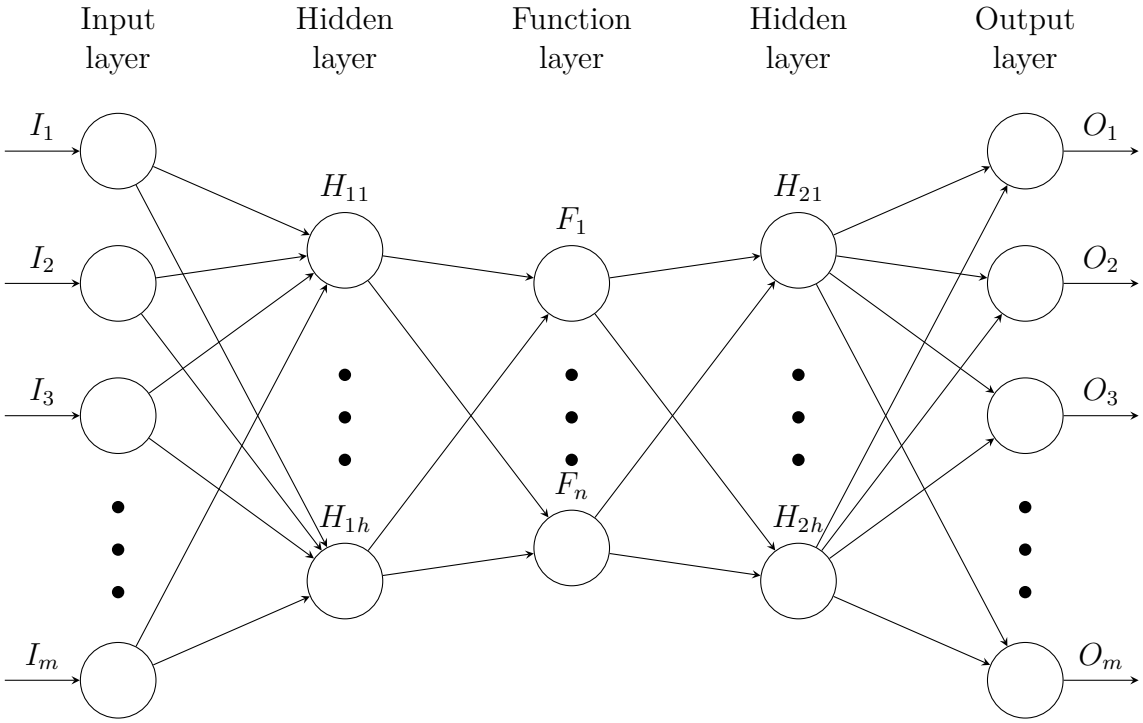
> Autoencoders are neural networks that can automatically learn useful features and representations from the data; this makes them an ideal technique for simplifying the process of feature engineering in machine learning

studies.(p.194)

The goal of an autoencoder is, generally speaking, to identify a function that sufficiently mimics the hidden mechanism behind all observations $x_i \in \boldsymbol{X}$ by attempting to develop a function where: $h_{W,b}(\mathbf{X}) \approx \mathbf{X}$. While there exist many different ways to solve a problem like this, autoencoders do this with the help of two *neural networks* (where one is used as decoder and the other one as encoder). A very simple illustration of an autoencoder can be found in Figure 10.

Figure 10

Illustration of an autoencoder with two hidden layers



where the following needs to hold true:

$$n(I) > n(H) > n(F) < n(H) < n(O) \tag{10}$$

$$n(I) = n(O) \tag{11}$$

In the case of autoencoders, each Input I_i corresponds to precisely one feature f_i. The autoencoder does then try to adjust the internal weights of the neural networks in such a way that the output O is as close to the Input I as possible:

$$min\ C(O) = \sum_{i=1}^{n(I)} |I_i - O_i| \tag{12}$$

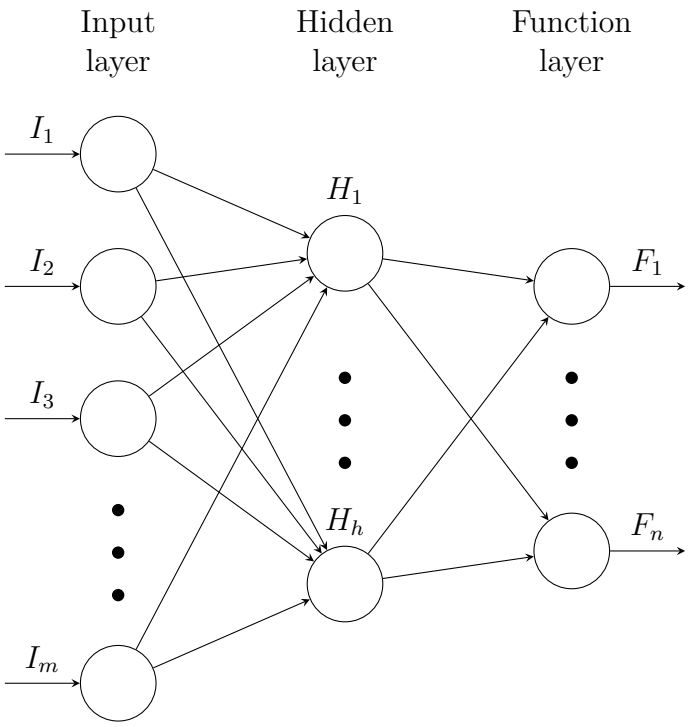
Due to the constraints introduced with Equation 10 and the thereby resulting dimensional bottleneck in the function layer, the autoencoder is forced to find a lower-dimensional representation of the original data with m dimensions that preserves as much original information (contained in I), as possible. Once the autoencoder has been trained enough (meaning $C(O)$ reliably, sufficiently small), the first neural network as depicted in Figure 11 can be used as a feature extraction method, which generates for every high dimensional input $\mathbf{X}$ a corresponding low dimensional output $\mathbf{Y}$ where:

$$I_1 \cup\ I_2 \cup \ldots \cup I_m = \mathbf{X} \tag{13}$$

and

$$F_1 \cup\ F_2 \cup \ldots \cup F_n = \mathbf{Y} \tag{14}$$

Figure 11
Feature extraction with an autoencoder



t-SNE

Initially introduced by van der Maaten and Hinton (2008), t-SNE is an improved version of *stochastic neighbour embedding* that can be used for (non)-linear dimension reduction in high dimensional vector spaces.

Expressed very simplified, t-SNE works the following way: In the first step, instead of using classic euclidean distances, t-SNE calculates *conditional probabilities* to represent similarities between observations in the original vector space (van der Maaten and Hinton, 2008):

$$p_{i|j} = \frac{exp(-\left\|x_i - x_j\right\|^2 / \sigma_i)}{\sum_{k \neq i} exp(-\left\|x_i - x_k\right\|^2 / \sigma_i)} \tag{15}$$

where σ_i is the variance of a Gaussian that is centered at x_i. $p_{i|j}$ can be interpreted

as the probability of observation x_j to be the neighbor of observation x_i. In the second step, the algorithm creates a set of low dimensional counterparts $\mathbf{Y}$ to the original high dimensional data set $\mathbf{X}$ using a *student-t distribution:*

$$q_{i|j} = \frac{exp(1 + \|y_i - y_j\|^2)}{\sum_{k \neq l} exp(1 + \|y_k - y_l\|^2)} \tag{16}$$

In the last step, t-SNE tries to transform the high dimensional similarities between all points $x_i, x_j \in \mathbf{X}$ to the lower dimensional representation $\mathbf{Y}$ with as little information loss as possible. The locations of all points $y_i \in \mathbf{Y}$ is thereby determined by a minimization of the *Kullback-Leibler divergence* of both distributions P and Q:

$$\mathrm{KL}\left(P \parallel Q\right) = \sum_{i \neq j} p_{ij} \log \frac{p_{ij}}{q_{ij}} \tag{17}$$

where the Kullback-Leibler divergence quantifies the proximity of two probability distributions (Shlens, 2014). t-SNE solves this minimization problem with the help of a gradient descent algorithm (van der Maaten and Hinton, 2008).

Non-Transforming Algorithms

While there are a sheer uncountable number of non-transforming algorithms available, explaining and listing all of them would go beyond the scope of this thesis. So as a consequence, this thesis focuses on just two (very deep) concepts: *k-Means clustering* and the *Fourier analysis.*

Clustering

James et al. (2013) defined clustering very generally in the following way:

> Clustering refers to a very broad set of techniques for finding subgroups, or clustering clusters, in a data set. When we cluster the observations of a data set, we seek to partition them into distinct groups so that the observations

> within each group are quite similar to each other, while observations in different groups are quite different from each other. (p. 384)

Among all available partitional algorithms, this endeavour utilizes one of the most prominent and time-proven algorithms, namely the k-Means Clustering Algorithm (James et al., 2013).

k-Means Clustering

Generally speaking, the k-Means clustering algorithm tries to partition a concrete data set into K distinct clusters with no intersections. James et al. (2013) formalized the general criterions for resulting clusters the following way:

Let $C_1, \ldots, C_k$ denote the set of clusters which need to satisfy the following two properties:

$$C_1 \cup\ C_2 \cup \ldots \cup C_k = \mathbf{X} \text{ where } \mathbf{X} \text{ is the set of all observations.} \tag{18}$$

$$\forall\ i, m \in K, i \neq m : C_i \cap\ C_m\ = \varnothing \text{ where } K \text{ is the set of all Cluster-Indices.} \tag{19}$$

(p. 386)

In order to create the optimal clusters, the k-Means algorithm therefore uses the sum of the total variation within each cluster as a measurement for clustering quality, mathematically formulated, this results in a classic minimization problem where $W(C_i)$ denotes the a function that measures the variation within one cluster:

$$min \left\{ \sum_{k=1}^{K} W(C_k) \right\} \tag{20}$$

As with many other algorithms, a big part of the quality of the results depends on the metric that is used. In the case of k-Means clustering, there are multiple possible metrics to calculate the variance within clusters. The most prominent one is the standard

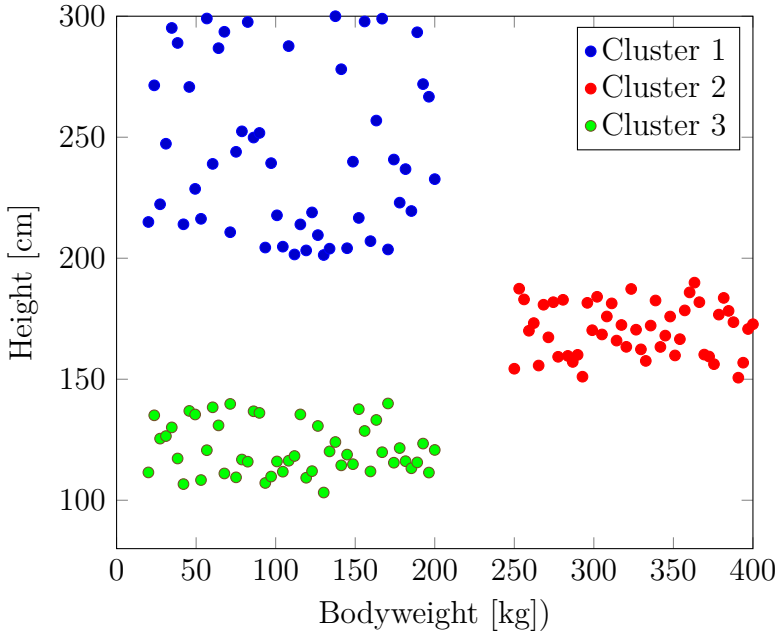
squared euclidean distance which will be used in this thesis in the form of the following function:

$$W(C_k) = \frac{1}{|C_k|} \sum_{i,l \in C_k} \sum_{j=1}^{p} (x_{ij} - x_{lj}) \tag{21}$$

The results of the k-Means clustering with $k = 3$, applied to a data set that contains the bodyweight and the height of several people, could then look like Figure 12 :

Figure 12

k-Means clustering with k = 3



Fourier Analysis

For many years, up to this date, one mathematical concept has been of intense interest among data scientists and researches, namely the so called *Fourier analysis*, as a google search for the terms "fourier analysis data science" on the 24th of March 2022, which yielded a relative high number of about 3.690.000 results, showed. As mentioned by Gelman and Braun (2001) many pattern recognition task rely heavily on the Fourier analysis. Generally used to solve many different problems in engineering, ranging from noise reduction in communications technology to time series analysis in meteorology and

even modern image compression, this mathematical concept has proven it's utility over and over again and even offers a lot of useful functions, when it comes to pattern detection in multivariate outliers.

The classic Fourier analysis was first introduced by Joseph Fourier (1768-1830) and is, among other things, based on the idea that any (continuous) function can be sufficiently approximated by a linear combination of basic trigonometric functions $sin(nx)$ and $cos(nx)$. This concept is better known as *Fourier series*. Mathematically speaking, this results in the following theorem where the Fourier series of a function $f(x)$ can then be formally be given by:

$$f(x) \sim \sum_{n=-\infty}^{\infty} a(n) e^{\frac{-2\pi i n x}{L}} \tag{22}$$

where the coefficients $a(n)$ are defined by:

$$a(n) = \frac{1}{L} \int_a^b f(x) e^{\frac{-2\pi i n x}{L}} dx, n \in \mathbb{Z} \tag{23}$$

To further illustrate this concept, one could imagine a function $\phi(x)$ that consists of an infinite number of rectangular pulses with a wavelength of L added together:

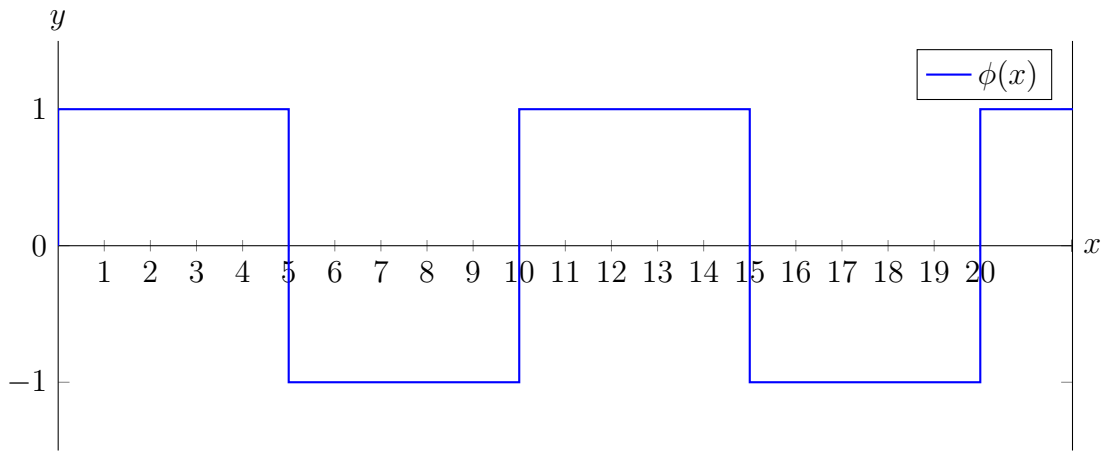
$$\phi(x) = \sum_{n=0}^{\infty} \phi_n(x) \tag{24}$$

with:

$$\phi_n(x) = \begin{cases} -1, \ if \ x \geqslant nL + \frac{L}{2} \ \wedge x \leqslant (n+1)L \\ 0, \ if \ x > (n+1)L \vee x < nL \\ 1, \ if \ x \leqslant nL + \frac{L}{2} \ \wedge x \geqslant nL \end{cases} \tag{25}$$

Figure 13 shows a plot of $\phi(x)$.

Figure 13
$\phi(x) \ for \ L = 10$



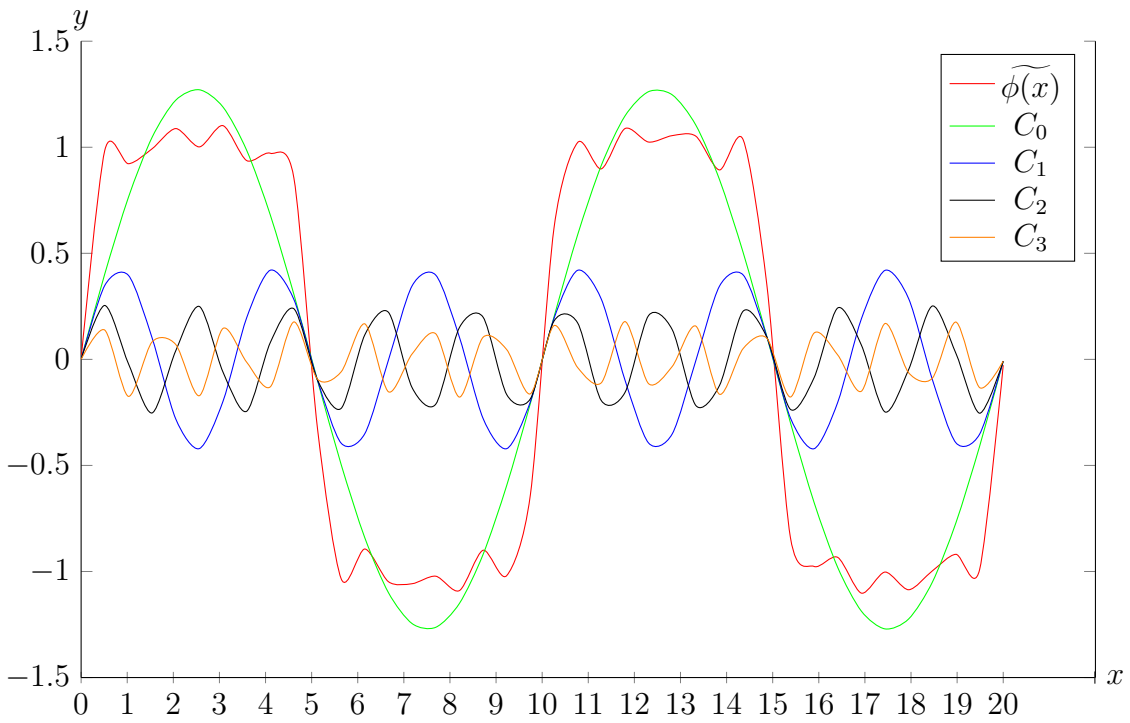
Using the Fourier Analysis, $\phi(x)$ can be sufficiently approximated by the following Fourier Series:

$$\phi(x) \sim \frac{4}{\pi} \sum_{n=1,3,5,7,\ldots} \frac{1}{n} sin(\frac{n\pi x}{L}) \tag{26}$$

which means that the rectangular pulse $\phi(x)$ from above, can be approximated trough a series of $sin(\lambda_i x)$ functions. To provide a more visual demonstration, Figure 14 depicts the Fourier series of $\phi(x)$: $\widetilde{\phi(x)}$ up to the 4th degree where:

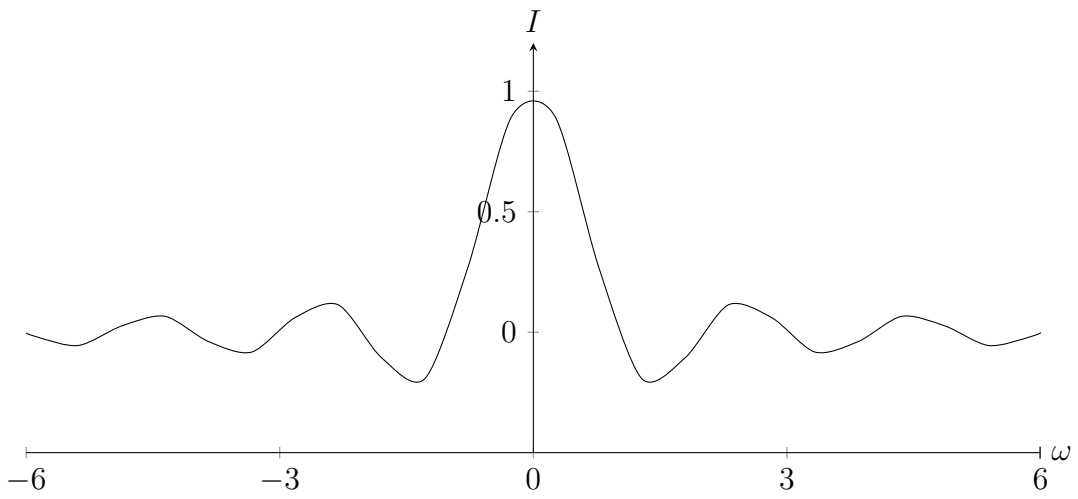
$$\widetilde{\phi(x)} = C_0 + C_1 + C_2 + C_3 \tag{27}$$

Figure 14
Fourier series of $\phi(x)$ with $L = 10$ to the 4th degree



In Figure 14, the original function $\phi(x)$ has been approximated through four sin-functions with four different frequencies. If someone would not stop at the 4th order, but rather integrate over the full space ($n = \{ -\infty; \infty \}$), he would end up with an infinite number of sin-functions with different frequencies. This frequency distribution could be plotted and analysed in terms of the weighted contribution (I) of each frequency to the function-approximation. This interpretation is then called the *frequency domain*. For further illustration, Figure 15 depicts the frequency domain of $\phi(x)$.

Figure 15
Frequency distribution of the Fourier series of $\phi(x)$



Fourier series and pattern detection.

As originally developed to solve heat diffusion problems, someone might righteously ask how the Fourier analysis should be relevant for the detection of patterns? According to the definition of patterns introduced at the beginning of this thesis, patterns can be seen as *sequence regularities*. If combined with the concept of spacial frequencies, one could notice that regularities can be mathematically modeled trough periodic sine or cosine waves. For better illustration, one can imagine three observations in a standard 2-D cartesian coordinate system. Those observations could for example be the representations of one dimensional positions of humans in a waiting queue, where each vector represents the position on the (spacial) x-Axis ($\vec{x_i}[0]$), as well as a count for number of humans at that given spot ($\vec{x_i}[1]$):

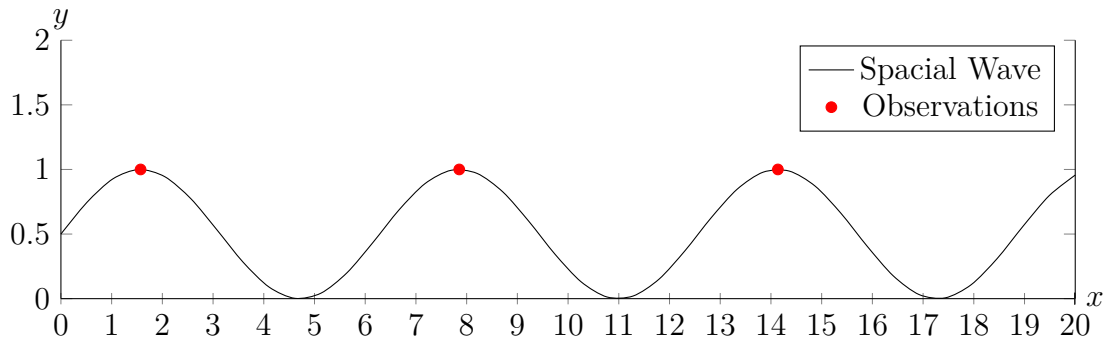
$$\vec{x_1} = \begin{bmatrix} \frac{\pi}{2} \\ 1 \end{bmatrix}, \ \vec{x_2} = \begin{bmatrix} \frac{5\pi}{2} \\ 1 \end{bmatrix}, \ \vec{x_3} = \begin{bmatrix} \frac{9\pi}{2} \\ 1 \end{bmatrix} \tag{28}$$

One could now come to notice that the (euclidean) distance between each observation is the same (in this example 2π), and thereby, the distance can be classified as the characteristics of a pattern at hand (As the constant distance between each observation

can be seen as *sequence regularity*). If now the concept of (spacial) waves gets introduced, those regularities could be approximated through the use of a standard wave function as in Equation 29 and illustrated in Figure 16.

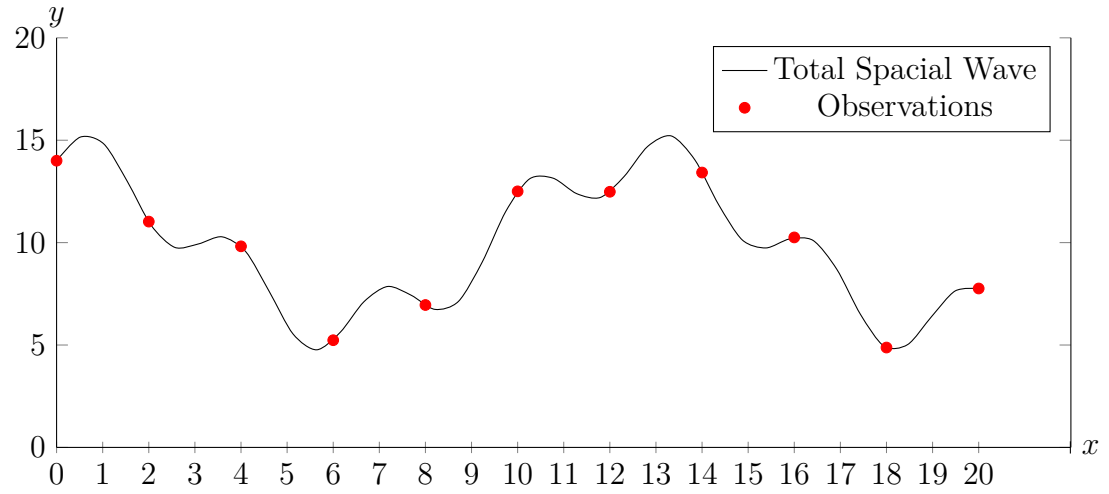
$$y(x) = y_0 * sin(2\pi f x + \varphi_0) \tag{29}$$

Figure 16
Illustration of a Spacial Wave



In this particular case, identifying a pattern (equidistance) was obvious, but in reality, patterns like this are no longer intuitively detectable for the human brain. For example, one could imagine a set of observations from the same scenario as above, but now, the euclidean distance between two observations is not constant anymore:

Figure 17
Spacial Frequency Decomposition



If one tries to sufficiently approximate the seemingly general pattern behind the observations, he will soon conclude that there probably does not exist a single periodic function $y(x) = y_0 * sin(2\pi fx + \varphi_0)$ that sufficiently describes the general pattern. In this case, for example, the following function could be used to model/describe the occurrences of the observations:

$$y(x) = \frac{3 * sin(2x)}{2} + 4 * cos(0.5x) + 10 \tag{30}$$

In this simplified example, things will get quite obvious, but it illustrates a fundamental concept: Sometimes, the whole system can be explained and understood best by focusing on its parts. In this case, $y(x)$ can be elegantly split up into two functions.

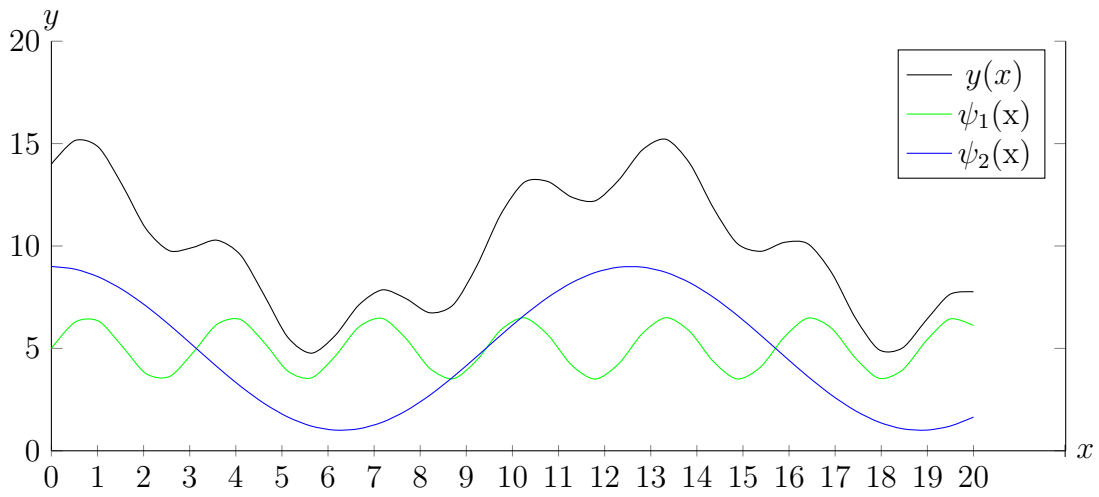
$$\psi_1(x) = \frac{3 * sin(2x)}{2} + 5 \tag{31}$$

$$\psi_2(x) = 4 * cos(0.5x) + 5 \tag{32}$$

where the following holds true:

$$y(x) = \psi_1(x) + \psi_2(x) \tag{33}$$

Figure 18
Spacial Frequency Decomposition



By splitting the original function $y(x)$ up into two fundamental trigonometric functions, new room for insights has been generated. For example that the seemingly pattern-less complex/non-harmonic function $y(x)$ could consist of two harmonic functions $\psi_1(x)$ and $\psi_2(x)$ that follow a intuitive pattern. These new potential insights open up a new domain of possible interpretations of the original observations. For example, in the context of the waiting queue example, one could derive the hypothesis that the spatial distance between each person is not entirely random, but rather conclude that the distances are just superpositions and depend on a second variable (moderator). This second variable could, for example, be a variable indicating membership of different groups. This insight could for instance mean that those humans of interest are not as homogeneous as initially assumed.

Maybe there are two different types of humans, and all humans that belong to group A have agreed to an unconscious rule that every member in this group should keep a distance of $\lambda_{\psi 1}$ to each other while all members of Group B agreed on a distance of $\lambda_{\psi 2}$.

The seemingly chaotic structure behind those initial observations has been broken up into a linear combination of two simple patterns.

Evaluation of Algorithms

Due to the limited scope of this thesis, not all of the algorithms presented above can be included in the final pattern detection algorithm. Therefore, based on their area of application and their advantages/disadvantages, only a subset of all introduced algorithms are considered. Consequently, this section should provide the reader with a quick overview of the (a) the area of application, (b) the advantages, and (c) the disadvantages of all previously introduced algorithms before the next chapter explains the selection criterion for the creation of the final algorithm.

PCA

Area of Application

PCA can be applied to high dimensional data with real variables ($\Re^n$ where $n > 2$). Most suitable for observations where the degree of statistical dispersion differs strongly between different variables.

Advantages

PCA is a reliable and robust way to reduce dimensions in high-dimensional data sets. Due to the algorithms simplification (namely the linear solution space), the algorithm tends to be relatively fast and reliable in most cases.

Disadvantages

One of the biggest disadvantages of PCA is the restriction to linear solution spaces. This restriction means that PCA is not suited to find non-linear relationships between the existing variables.

t-SNE

Area of Application

t-SNE generally can be applied to high dimensional data with real variables ($\Re^n$ where $n > 2$).

Advantages

According to Becht et al. (2018) the t-SNE Algorithm is particularly efficient in highlighting local structures in high dimensional data sets. In addition, due to the non-linear approach, t-SNE avoids overcrowding issues generally more successfully than PCA (Becht et al., 2018).

Disadvantages

t-SNE is relatively slow when compared to other non-linear techniques, such as Uniform Manifold Approximation and Projection (UMAP) (Becht et al., 2018). Additionally van der Maaten and Hinton (2008) claim that t-SNE comes with the following disadvantages/weaknesses:

> Although we have shown that t-SNE compares favorably to other techniques for data visualization, t- SNE has three potential weaknesses: (1) it is unclear how t-SNE performs on general dimensionality reduction tasks, (2) the relatively local nature of t-SNE makes it sensitive to the curse of the intrinsic dimensionality of the data, and (3) t-SNE is not guaranteed to converge to a global optimum of its cost function. (p. 2598)

Non-negative matrix factorization (NMF)

Area of Application

In general, NMF can be applied to any high dimensional data with real, non-negative variables ($\Re^n_{>0} = \{x \in \Re \mid x \geq 0\}$ where $n > 2$)

Advantages

According to (Lee and Seung (1999)) NMF learns, contrary to other prominent methods like PCA or VQ (Vector Quantization), a part-based representation of the data. This part-based representation is particularly advantageous for deconstructing observations into potential sub components.

Disadvantages

Due to the very nature of the algorithm, NMF can not handle variables that allow negative numbers (Lee and Seung, 1999).

Autoencoders

Area of Application

Autoencoders can generally be applied to high dimensional data with real, non-negative variables ($\Re^n$ where $n > 2$)

Advantages

In their paper Hinton and Salakhutdinov (2006) presented a case in which trained autoencoders drastically outperformed the classic PCA method due to their fundamental non-linear structure. Furthermore, autoencoders can be very efficient in removing noise from data sets (Lu et al., 2013),(Gondara, 2016).

Disadvantages

Autoencoders are based on neuronal networks, and they, therefore, typically require a high amount of training to work sufficiently (Barnard, 1992). This requirement means that they most likely only produce satisfactory results for extensive data sets. In addition, if the number of hidden layers gets larger (> 2) the training process may become tedious, as the back-propagation converges slowly and is likely to get stuck in local minima (van Der Maaten et al., 2008).

k-Means clustering

Area of Application

Generally applicable in $\Re^n$, but as Napoleon and Pavalakodi (2011) pointed out, the k-Means clustering often times does not work very well when applied to high dimensional data. Especially for $n > 3$ the complexity might increase significantly as specific metrics to calculate the variations within one cluster might no longer apply to the vector space at hand.

Advantages

According to Ali and Kadhum (2017), using k-Means clustering comes, among other things, with the following advantages: k-Means clustering is (a) fast due to the simplicity of the algorithm and (b) efficient with relatively low computational complexity.

Disadvantages

Due to the very nature of k-Means clustering, the results are highly susceptible to the initial placement of the cluster centers (Celebi et al., 2013). Not very well suited for high dimensional data (Napoleon and Pavalakodi, 2011). Additionally, Celebi et al., 2013 pointed out that k-Means clustering can only detect compact and hyper-spherical clusters that are well separated, and that k-Means clustering is very sensitive to noise.

Fourier Analysis

Area of Application

The Fourier analysis can generally be applied to two dimensional data with real variables ($\Re^2$).

Advantages

The Fourier analysis performs extraordinary well when it is used to uncover information about the two dimensional spacial organisations of data points, when compared to conventional statistic methods (Cullander et al., 1990). Another case in which the Fourier analysis has proven great utility is summarizing certain movement patterns (Silva et al., 1996).

Disadvantages

Generally, the time it takes to compute a Fourier transformation is long (Chan and Pang, 2000).

Methods

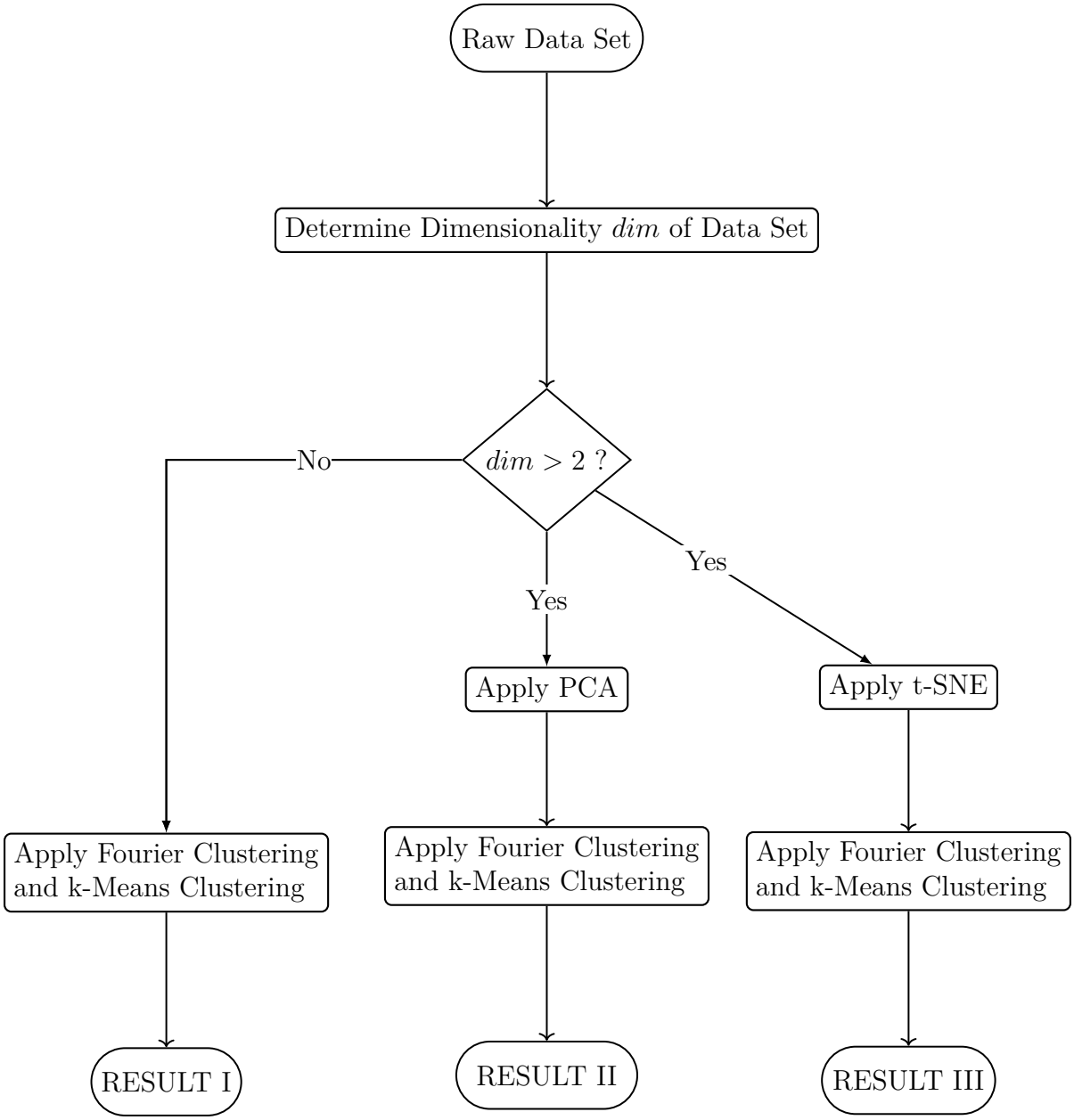
Final Algorithm

Building on top of the previous chapters, I have developed an algorithm that intends to help researchers and scientists to identify patterns in their sets of interesting outliers. In order to do so, I have selected four of the previously presented algorithms, namely (a) PCA, (b) t-SNE, (c) k-Means clustering, and (d) the Fourier analysis based on the earlier evaluation, to build a sophisticated pattern detection algorithm. Unlike other earlier works, I did not just apply each of those algorithms independently of each other but instead intertwined them (for example, recommended by Napoleon and Pavalakodi (2011)) so that all of them can contribute their strengths to a more extensive algorithm.

In a bit more detail, the final algorithm works the following way: In the first step, the algorithm accesses the number of features in the raw data set (and thereby the dimensions of the vector space ϑ^{dim}). If the number of features (dim) exceeds two, the algorithm starts by applying two different feature extraction methods first, namely the PCA and the t-SNE. It is crucial to notice that the algorithm does that in parallel, meaning that both methods get to be applied to a copy of the raw data set independently of each other. This process then yields two 2-D representations (one generated by PCA and one by t-SNE) of the original high(er) dimensional space ϑ^{dim}) on which the clustering methods (Fourier- and k-Means clustering) will then be applied. If the original number of features (dim) equals two, the algorithm starts immediately by applying the clustering methods (Fourier- and k-Means clustering). Once the clustering methods have been applied, the algorithm performs correlation tests within all identified clusters to determine whether the outliers within those clusters tend to follow a particular distribution or appear to be randomly distributed. This is important as the first could indicate the existence of a currently unknown moderator.

A structural overview of the final algorithm can be found in Figure 19. The corresponding implementation in *Python* can be found in Appendix A.

Figure 19
Structural overview of the final algorithm



Generation of Test Data

In order to be able to test the final algorithm, I designed a small test-scenario. For this, I created a data set with 1500 observations that includes three variables. I introduced the third variable as a significant moderator and set the relationship among variables as follows given (outcome variable = y; predictor = x and moderator = w):

```
y <- 2.46960 + (-0.26*x) + (0.44*w) + (0.077*x*w) +rnorm(250, 0, sqrt(1 -
    (0.0676 + 0.1936 + 0.005929)))
```

Later, using the approach specified by Astivia (2022), I introduced Mahalanobis distance-based outliers to predictor and outcome variable for the observations with high values of the moderator (i.e., mean + one standard deviation). I then conducted an outlier analysis using Mahalanobis distance to check if the outliers introduced could be detected in the predictor and outcome variables. Later, I conducted another outlier analysis using Mahalanobis distance to determine if, with the addition of the moderator into the outlier analysis, the outliers become non-outliers. Finally, I introduced some extreme random data to the data set as random outliers. The script that has been used for the data generation can be found in Appendix B.

Results

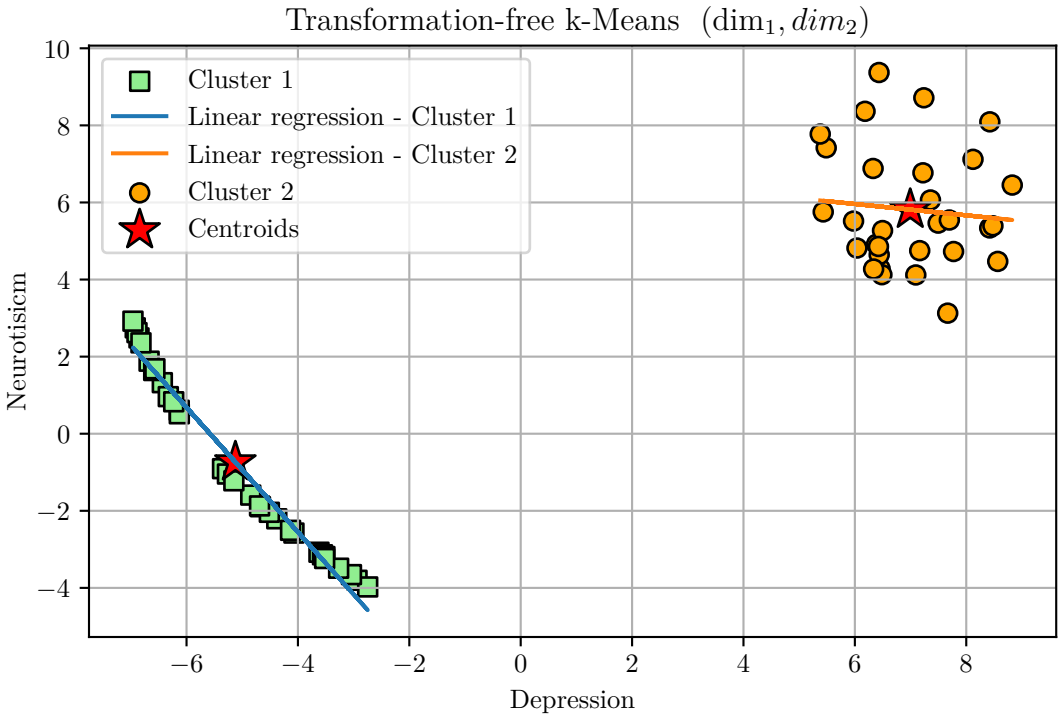
The application of the final algorithm eventually yielded the following results that are depicted in Figure 20 - Figure 24, where each figure shows the results of the clustering and the following regression testing. Below each of those figures are the summaries for the regressions.

As Figure 20 clearly shows, the algorithm was able to differentiate between interesting outliers (Cluster 1) and random outliers (Cluster 2) with the help of k-Means clustering and even managed to identify a correlation between Depression and Neuroticism. On the other hand, PCA + k-Means clustering and t-SNE + k-Means clustering did not produce any significant results, as they could not separate the interesting outliers from the random outliers.

Generally speaking, the distinction between interesting and random outliers is based on distributions within the clusters. If the outliers within one cluster can be fitted sufficiently well through a first-order polynomial (R-squared > 0.9), I have assumed that the outliers are non-random and follow a particular (linear) trend. Vice versa, if that is not the case, I have assumed that the outliers in the respective cluster follow a random distribution.

Figure 20

Result of the final algorithm when applied to dim_1 and dim_2 of the raw data.

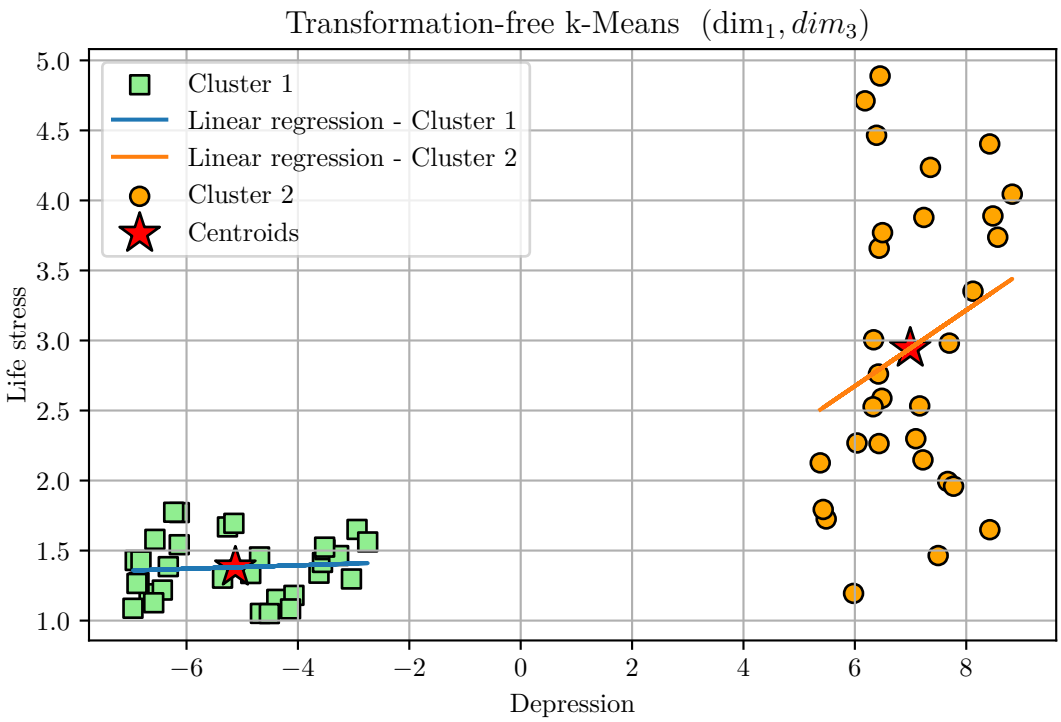


Cluster:	Cluster 1	**R-squared:**	0.974
Model:	OLS	**Adj. R-squared:**	0.973
Method:	Least Squares	**F-statistic:**	1077.
No. Observations:	31	**AIC:**	30.28
Df Residuals:	29	**BIC:**	33.15
Df Model:	1		
Covariance Type:	nonrobust		

Cluster:	Cluster 2	**R-squared:**	0.009
Model:	OLS	**Adj. R-squared:**	-0.027
Method:	Least Squares	**F-statistic:**	0.2465
No. Observations:	30	**AIC:**	114.4
Df Residuals:	28	**BIC:**	117.2
Df Model:	1		
Covariance Type:	nonrobust		

Figure 21

Result of the final algorithm when applied to dim_1 and dim_3 of the raw data.

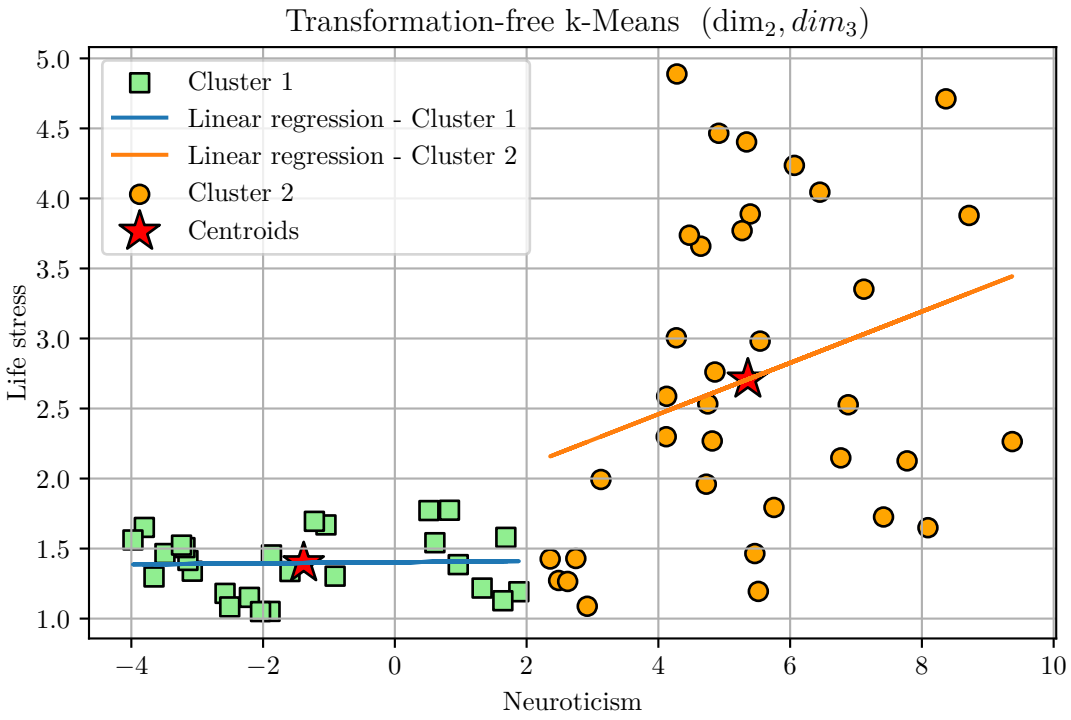


Cluster:	Cluster 1	**R-squared:**	0.007
Model:	OLS	**Adj. R-squared:**	-0.027
Method:	Least Squares	**F-statistic:**	0.1982
No. Observations:	31	**AIC:**	-4.992
Df Residuals:	29	**BIC:**	-2.124
Df Model:	1		
Covariance Type:	nonrobust		

Cluster:	Cluster 2	**R-squared:**	0.064
Model:	OLS	**Adj. R-squared:**	0.030
Method:	Least Squares	**F-statistic:**	1.903
No. Observations:	30	**AIC:**	89.79
Df Residuals:	28	**BIC:**	92.59
Df Model:	1		
Covariance Type:	nonrobust		

Figure 22

Result of the final algorithm when applied to dim_2 and dim_3 of the raw data.

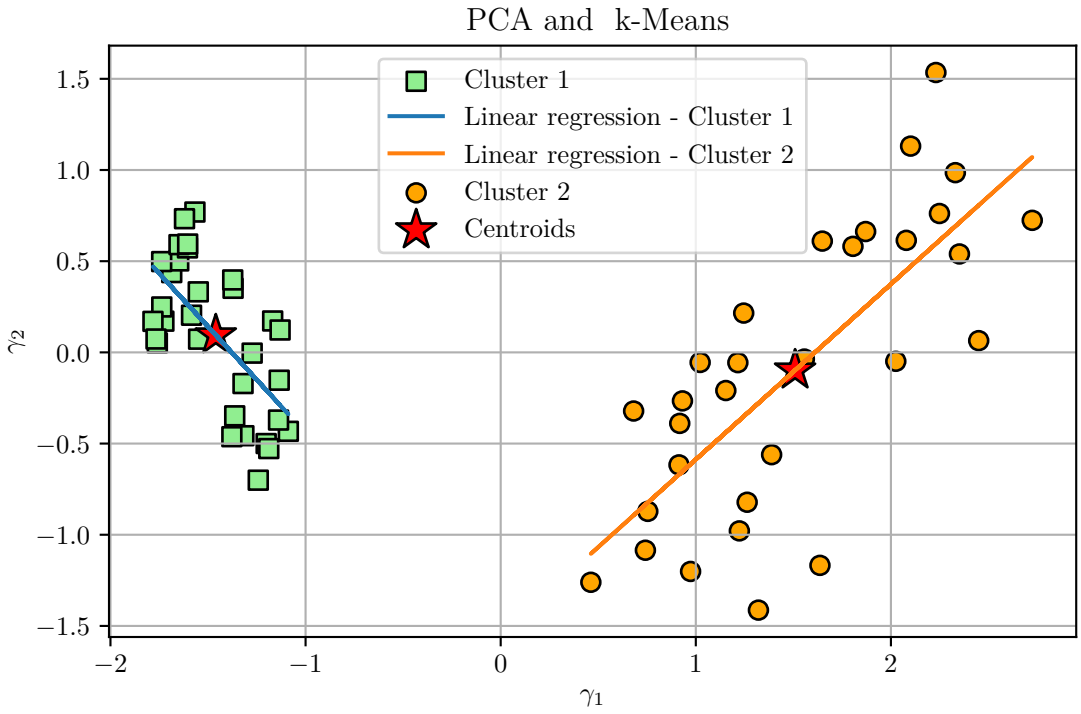


Cluster:	Cluster 1	**R-squared:**	0.001
Model:	OLS	**Adj. R-squared:**	-0.040
Method:	Least Squares	**F-statistic:**	0.02826
No. Observations:	26	**AIC:**	-1.252
Df Residuals:	24	**BIC:**	1.264
Df Model:	1		
Covariance Type:	nonrobust		

Cluster:	Cluster 2	**R-squared:**	0.086
Model:	OLS	**Adj. R-squared:**	0.058
Method:	Least Squares	**F-statistic:**	3.111
No. Observations:	35	**AIC:**	108.5
Df Residuals:	33	**BIC:**	111.7
Df Model:	1		
Covariance Type:	nonrobust		

Figure 23

Result of the PCA branch of the final algorithm when applied to the raw data.

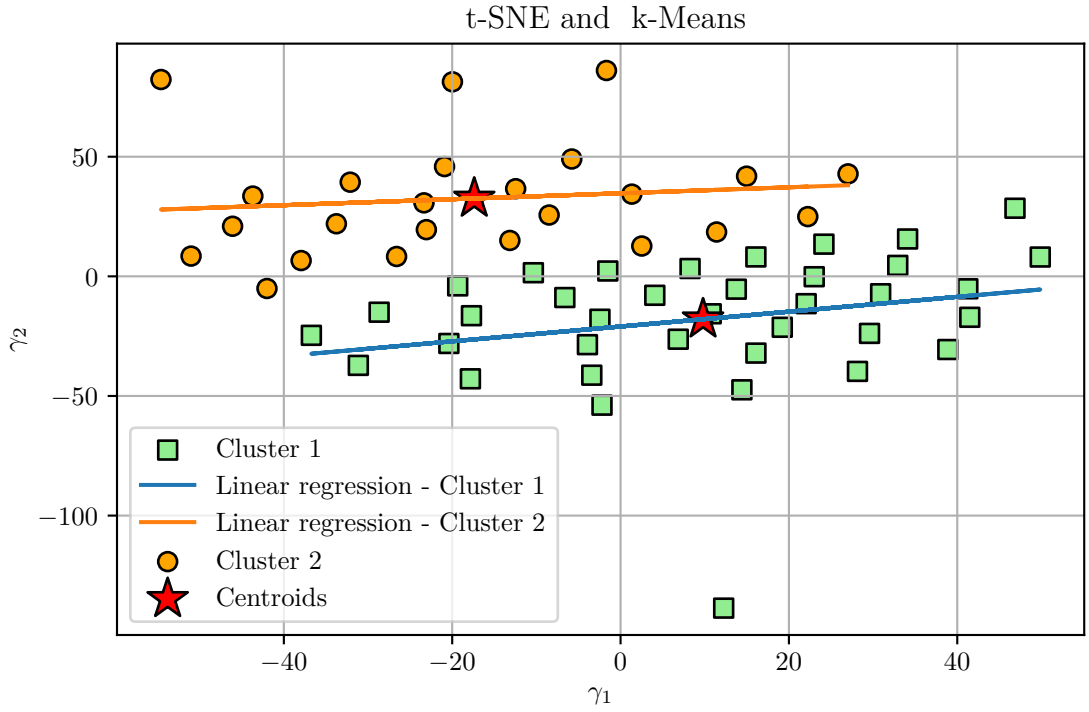


Clustere:	Cluster 1	**R-squared:**	0.419
Model:	OLS	**Adj. R-squared:**	0.399
Method:	Least Squares	**F-statistic:**	20.92
No. Observations:	31	**AIC:**	19.33
Df Residuals:	29	**BIC:**	22.20
Df Model:	1		
Covariance Type:	nonrobust		

Cluster:	Cluster 2	**R-squared:**	0.554
Model:	OLS	**Adj. R-squared:**	0.538
Method:	Least Squares	**F-statistic:**	34.78
No. Observations:	30	**AIC:**	50.13
Df Residuals:	28	**BIC:**	52.93
Df Model:	1		
Covariance Type:	nonrobust		

Figure 24

Result of the t-SNE branch of final algorithm when applied to the raw data.



Cluster:	Cluster 1	**R-squared:**	0.031
Model:	OLS	**Adj. R-squared:**	-0.005
Method:	Least Squares	**F-statistic:**	0.8600
No. Observations:	29	**AIC:**	287.6
Df Residuals:	27	**BIC:**	290.4
Df Model:	1		
Covariance Type:	nonrobust		

Cluster:	Cluster 2	**R-squared:**	0.021
Model:	OLS	**Adj. R-squared:**	-0.012
Method:	Least Squares	**F-statistic:**	0.6348
No. Observations:	32	**AIC:**	311.2
Df Residuals:	30	**BIC:**	314.2
Df Model:	1		
Covariance Type:	nonrobust		

Discussion

In this thesis, I have developed a mathematical algorithm to reliably detect patterns in multivariate outliers. I did so by combing multiple methods that are currently widely used in pattern detection and machine learning. Namely (a) PCA, (b) t-SNE, (c) k-Means Clustering, and (d) Fourier analysis. The results were significant after applying the final algorithm to the testing data set (Appendix B).

More concretely, the final algorithm detected all hidden interesting outliers in the testing data set and was further able to distinguish the interesting outliers from the random outliers that had been previously added. Furthermore, the algorithm was even able to identify a positive correlation between Depression and Neuroticism in the interesting outliers. This can be seen best in Figure 20, where Cluster 1 contains all interesting outliers while Cluster 2 contains all random outliers.

While k-Means clustering performed well and was able to detect all interesting outliers, I could not discover any meaning in the results of the Fourier clustering. I conclude that, although conceptually promising, the Fourier clustering approach would need some further (experimental) investigations.

Limitations and Future Directions

Due to the limited time frame of this thesis, this endeavor faced some limitations. In particular, in the last part, the evaluation and testing only covered a minimal test case of outlier data, as the simulated data that I created only contained positive outliers. Generally speaking, it would therefore be necessary to test the algorithm with more (complex) sets of outliers, as the current test case did not sufficiently exclude the possibility that the algorithm's good performance rests purely on a good suiting test data set. In addition, the algorithm itself still has much room for further improvements that, unfortunately, due to the limited scope of this thesis, have not been introduced in this thesis. Some of those potential improvements include the introduction of other dimension

reduction techniques (for example, some of those presented by van der Maaten and Hinton (2008)) and different clustering techniques besides the classical k-Means clustering, as well as some more fine-tuning of the already included algorithms. An example of an area in the current algorithm that needs some fine-tuning is the initialization method used in the k-Means algorithm. Celebi et al. (2013) offer some interesting and promising methods to improve the standard initialization methods.

Practical Implications

While initially primarily developed for other researchers and scientists, this algorithm could potentially be used in every organization that deals with outliers and wants to understand the hidden mechanisms behind those outliers.

Conclusion

With this thesis, I have aimed to develop a mathematical algorithm that can detect patterns in interesting multivariate outliers. To develop this algorithm, I identified promising, already existing pattern recognition algorithms, evaluated them, and eventually combined the most suitable ones into a final pattern detection algorithm. After developing this algorithm, I created a data set and applied the algorithm to it in order to assess the performance of the algorithm. While the k-Means clustering showed some significant results, the Fourier clustering could not produce any relevant results.

References

Ali, H. H., & Kadhum, L. E. (2017). K-means clustering algorithm applications in data mining and pattern recognition. *International Journal of Science and Research (IJSR)*, *6*(8), 1577–1584.

Astivia. (2022). Simulating multivariate outliers with known mahalanobis' distance. https://psychometroscar.com/simulating-multivariate-outliers/

Barnard, E. (1992). Optimization for training neural nets. *IEEE Transactions on Neural Networks*, *3*(2), 232–240. https://doi.org/10.1109/72.125864

Becht, E., Dutertre, C.-A., Kwok, I. W. H., Ng, L. G., Ginhoux, F., & Newell, E. W. (2018). *Evaluation of UMAP as an alternative to t-SNE for single-cell data* (preprint). Bioinformatics. https://doi.org/10.1101/298430

Celebi, M. E., Kingravi, H. A., & Vela, P. A. (2013). A comparative study of efficient initialization methods for the k-means clustering algorithm. *Expert Systems with Applications*, *40*(1), 200–210. https://doi.org/10.1016/j.eswa.2012.07.021

Chan, C.-H., & Pang, G. (2000). Fabric defect detection by fourier analysis. *IEEE Transactions on Industry Applications*, *36*(5), 1267–1276. https://doi.org/10.1109/28.871274

Choudrey, R. A. (2002). *Variational Methods for Bayesian Independent Component Analysis* (Doctoral dissertation). University of Oxford. Retrieved March 1, 2022, from https://www.robots.ox.ac.uk/~parg/pubs/theses/RizwanChoudrey_thesis.pdf

Cullander, C., Baker, J. R., & Budinger, T. F. (1990). Rapid detection of spatial pattern by fourier analysis. *Computers in Biology and Medicine*, *20*(5), 297–310. https://doi.org/https://doi.org/10.1016/0010-4825(90)90009-E

Delac, K., Grgic, M., & Grgic, S. (2005). Independent comparative study of pca, ica, and lda on the feret data set. *International Journal of Imaging Systems and Technology*, *15*, 252–260. https://doi.org/10.1002/ima.20059

Gelman, L., & Braun, S. (2001). THE OPTIMAL USAGE OF THE FOURIER TRANSFORM FOR PATTERN RECOGNITION. *Mechanical Systems and Signal Processing*, *15*(3), 641–645. https://doi.org/10.1006/mssp.2000.1373

Gondara, L. (2016). Medical image denoising using convolutional denoising autoencoders [arXiv: 1608.04667]. *2016 IEEE 16th International Conference on Data Mining Workshops (ICDMW)*, 241–246. https://doi.org/10.1109/ICDMW.2016.0041 Comment: To appear: 6 pages, paper to be published at the Fourth Workshop on Data Mining in Biomedical Informatics and Healthcare at ICDM, 2016

Hawkins, D. M. (1980). Introduction. In D. M. Hawkins (Ed.), *Identification of Outliers* (pp. 1–1). Springer Netherlands. https://doi.org/10.1007/978-94-015-3994-4_1

Hinton, G. E., & Salakhutdinov, R. R. (2006). Reducing the Dimensionality of Data with Neural Networks. *Science*, *313*(5786), 504–507. https://doi.org/10.1126/science.1127647

Jackson, J. E. (1991). *A user's guide to principal components*. Wiley.

James, G., Witten, D., Hastie, T., & Tibshirani, R. (2013). *An introduction to statistical learning: With applications in r*. Springer. https://faculty.marshall.usc.edu/gareth-james/ISL/

Lee, D. D., & Seung, H. S. (1999). Learning the parts of objects by non-negative matrix factorization. *Nature*, *401*(6755), 788–791. https://doi.org/10.1038/44565

Leys, C., Klein, O., Dominicy, Y., & Christophe, L. (2018). Detecting multivariate outliers: Use a robust variant of the Mahalanobis distance. *Journal of Experimental Social Psychology*, (74), 150–156. https://doi.org/https://doi.org/10.1016/j.jesp.2017.09.011

Leys, C., Ley Christophe, Marie Delacre, Youri L. Mora, & Daniël Lakens. (2019). How to classify, detect, and manage univariate and multivariate outliers, With emphasis on

pre-registration. *International Review of Social Psychology*, (32(1)). https://doi.org/http://doi.org/10.5334/irsp.289

Lopez Pinaya, W. H., Vieira, S., Garcia-Dias, R., & Mechelli, A. (2020). Chapter 11 - autoencoders. In A. Mechelli & S. Vieira (Eds.), *Machine learning* (pp. 193–208). Academic Press. https://doi.org/https://doi.org/10.1016/B978-0-12-815739-8.00011-0

Lu, X., Tsao, Y., Matsuda, S., & Hori, C. (2013). Speech Enhancement Based on Deep Denoising Autoencoder, 5.

Napoleon, D., & Pavalakodi, S. (2011). A New Method for Dimensionality Reduction Using KMeans Clustering Algorithm for High Dimensional Data Set. *International Journal of Computer Applications*, *13*(7), 41–46. https://doi.org/10.5120/1789-2471

Pearson, K. (1901). LIII. On lines and planes of closest fit to systems of points in space [Publisher: Taylor & Francis _eprint: https://doi.org/10.1080/14786440109462720]. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, *2*(11), 559–572. https://doi.org/10.1080/14786440109462720

Shlens, J. (2014). Notes on kullback-leibler divergence and likelihood. *arXiv preprint arXiv:1404.2000*.

Silva, F., Pear, J., Tait, R., & Forest, J. (1996). Fourier analysis of movement patterns in pigeons. *Behavior Research Methods, Instruments, Computers*, *28*, 27–37. https://doi.org/10.3758/BF03203633

Simmons, J. P., Nelson, L. D., & Simonsohn, U. (2011). False-Positive Psychology: Undisclosed Flexibility in Data Collection and Analysis Allows Presenting Anything as Significant. *Psychological Science*, *22*(11), 1359–1366. https://doi.org/10.1177/0956797611417632

van der Maaten, L., & Hinton, G. (2008). Visualizing data using t-SNE. *Journal of Machine Learning Research*, *9*, 2579–2605. http://www.jmlr.org/papers/v9/vandermaaten08a.html

van Der Maaten, L., Eric, P., & Herik, J. V. D. (2008). *Dimensionality reduction: A comparative review* (tech. rep.).

Webb, A. R. (2002). *Statistical pattern recognition, second edition*. Wiley. https://doi.org/10.1002/0470854774

Appendix A

Developed Pattern-Detection-Algorithm

```
import seaborn as sns
import warnings
from sklearn.cluster import KMeans
warnings.filterwarnings('ignore')
from sklearn.metrics import silhouette_score
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.manifold import TSNE
import pandas as pd
from sklearn.metrics import r2_score
from sklearn.metrics import mean_absolute_percentage_error
from scipy.fft import fft, fftfreq
from numpy.fft import fft, ifft
import statsmodels.api as sm
import math
from mpl_toolkits import mplot3d
import numpy as np
import matplotlib


import matplotlib.pyplot as plt
matplotlib.use("pgf")
matplotlib.rcParams.update({
    "pgf.texsystem": "pdflatex",
    'font.family': 'serif',
    'text.usetex': True,
    'pgf.rcfonts': False,
```

```
})
```

```
def load_rawData(path:str) -> pd.DataFrame:
    df_rawData = pd.read_csv(path)
    return df_rawData

def applyFeature_extraction_lin(rawData:pd.DataFrame,label:str) -> pd.DataFrame:

    X = rawData.values
    X_scaled = StandardScaler().fit_transform(X)

    pca = PCA(n_components=2)
    principalComponents = pca.fit_transform(X_scaled)
    df_principal = pd.DataFrame(data=principalComponents
                            , columns=['PC_1', 'PC_2'])

    # plotting a scatter plot
    print("Scatter Plot: ")
    fig = plt.figure(figsize=(10, 7))
    plt.scatter(df_principal["PC_1"], df_principal["PC_2"])
    pcaResult = "results_plots/" + "PCA_for_" + label + ".png"
    pcaResultTeX = "results_plots/" + "PCA_for_" + label + ".pgf"
    plt.title("Result PCA")
    plt.savefig(pcaResult)

    fig.set_size_inches(w=6.5, h=3.5)
```

```
    plt.savefig(pcaResultTeX)
    #plt.show()

    list_explaind_vd = pca.explained_variance_ratio_.tolist()

    expl_variance = 0
    for pc in list_explaind_vd:
        expl_variance += pc

    print('Explained variation per principal component:
        {}'.format(pca.explained_variance_ratio_))
    print("PC1 and PC2 cumulatively explain " + str(round(pc * 100,2)) + " % of
        the total variance")
    if pc < 0.8:
        print("Unable to find helpful PC's in 2-D. Please use a different Method")
    else:
        print("Found helpful PC's in 2-D")


    #Store Data

    label_PCA_Result_data = "results_data/" + "PCA_for_" + label + ".csv"
    df_principal.to_csv(label_PCA_Result_data)


    return df_principal

def applyFeature_extraction_nonLin(rawData:pd.DataFrame,label:str) ->
    pd.DataFrame:
```

```
X = rawData.values


tsne = TSNE(n_components=2, verbose=1, perplexity=40, n_iter=300)
tsne_results = tsne.fit_transform(X)



x_1 = pd.Series(tsne_results[:, 0])
x_2 = pd.Series(tsne_results[:, 1])

df_tsne_result = pd.DataFrame(columns=['C1', 'C2'])


df_tsne_result['C1'] = x_1
df_tsne_result['C2'] = x_2
fig = plt.figure(figsize=(10, 7))

plt.figure(figsize=(16, 10))
plt.title("Result t-SNE")
sns.scatterplot(
    x="C1", y="C2",
    palette=sns.color_palette("hls", 10),
    data=df_tsne_result,
    legend="full",
    alpha=0.3
)

tsneResult = "results_plots/" + "t-SNE_for" + label + ".png"
tsneResultTeX = "results_plots/" + "t-SNE_for" + label + ".pgf"
plt.savefig(tsneResult)
fig.set_size_inches(w=6.5, h=3.5)
```

```
    plt.savefig(tsneResultTeX)
    plt.show()

    label_tsne_Result_data = "results_data/" + "t-SNE_for_" + label + ".csv"
    df_tsne_result.to_csv(label_tsne_Result_data)
    return df_tsne_result


#2-D
def applyFourierTransform(rawData:pd.DataFrame,label) -> pd.DataFrame:


    df_rawData = rawData.copy()
    df_rawData.columns = ['x', 'y']
    df_rawData = df_rawData.sort_values('x')


    x = df_rawData.iloc[:, 0]
    y = df_rawData.iloc[:, 1]



    lst = []
    for i in range(len(x)):
        lst.append(i)


    n_polyDeg = 2
    accurateModel_found = False
```

```
while(accurateModel_found == False):

    mymodel = np.poly1d(np.polyfit(x, y,n_polyDeg ))
    r_2_score = r2_score(y, mymodel(x))
    mape = mean_absolute_percentage_error(y,mymodel(x))

    if(r_2_score > 0.95 and mape < 0.1):
        accurateModel_found = True

    if n_polyDeg > 10:

        print(" ***** Caution @ [FOURIER]: No sufficiently approximating
            model for the observations found ***** ")
        return pd.DataFrame()


        break

    n_polyDeg += 1

print("----- Finished Modelling -----")
print("Sufficiently approximating model found")
print("-- Parameters of Model --")
print("Mape: " + str(mape))
print("R_2: " + str(r_2_score))
print("f(x) is a polynom of order " + str(n_polyDeg))
print("The coeffs for the polynom are: " + str(mymodel.c))
```

```
print("--------------------------------")

print("----- Starting the Fourier Transformation -----")


fig = plt.figure(figsize=(10, 7))

plt.scatter(x,mymodel(x),label = "Original Observations")
plt.plot(x,y, "r-", label = "Model")
plt.savefig("results_plots/Modelling_for_DFT" + label + ".png")
plt.legend()
plt.ylabel("Dim 1")
plt.xlabel("Dim ")
plt.title("Model Fit")

fig.set_size_inches(w=6.5, h=3.5)
plt.savefig("results_plots/Modelling_for_DFT" + label + ".pgf")
plt.show()


# ======= FOURIER
#set manually, could also be stet through the minimal spacial distance
sr = 600
ts = 1.0 / sr
t = np.arange(min(x), max(x), ts)
x = mymodel(t)

X = fft(x)
```

```
N = len(X)
n = np.arange(N)
T = N / sr
freq = n / T

plt.figure(figsize=(12, 6))
plt.subplot(121)

plt.stem(freq, np.abs(X)*ts, 'b', \
        markerfmt=" ", basefmt="-b")
plt.xlabel('Freq (Hz)')
plt.ylabel('FFT Amplitude |X(freq)|')
plt.xlim(0, 10)
plt.title("Frequency Domain")

plt.subplot(122)
plt.plot(t, ifft(X), 'r')
plt.xlabel('Dim 1')
plt.ylabel('Dim 2')
plt.tight_layout()
labelFourier_plot = "results_plots/" + "FourierSeries_of_" + label + ".png"
plt.savefig(labelFourier_plot)
plt.title("Space Domain")

plt.show()

df_coeffs_FourierSeries = pd.DataFrame({'Frequency': freq, 'FFT_Amplitude':
    np.abs(X)*ts},
                                       columns=['Frequency', 'FFT_Amplitude'])
```

```
#---------- CUT-OFF / Qualitiy Testing:


#Nyquist frequency cut off
n_cut = len(df_coeffs_FourierSeries)/2
# pick relevant ones
df_coeffs_FourierSeries = df_coeffs_FourierSeries.iloc[:-int(n_cut), :]


df_coeffs_F_sorted =
    df_coeffs_FourierSeries.sort_values("FFT_Amplitude",ascending=False)


i = 0
n_FComp = 3
y_FourierSeries = 0


list_FSComponents = []
#df_rawData["Cluster"] = n_FComp


plt.figure()
plt.plot(t,x,label= "Original Observations")
df_fourierClustered = df_rawData.copy()
df_fourierClustered["Cluster"] = n_FComp



while i < n_FComp:
    list_xGroupMemebers = []
```

```
comp_i = df_coeffs_F_sorted.iloc[i, 1] * np.sin(2 * np.pi *
    df_coeffs_F_sorted.iloc[i, 0] * t)
if df_coeffs_F_sorted.iloc[i, 0] == 0:
    comp_i += df_coeffs_F_sorted.iloc[i, 1]
    i += 1
    continue

else:
    #binary one
    y_max = max(comp_i)
    j = 0

    #find local maximas
    while j < len(comp_i):
        if math.isclose(comp_i[j],max(comp_i), rel_tol=y_max*0.05) :
            #df_rawData.loc[j,"Cluster"] = i
            list_xGroupMemebers.append(t[j])
        j += 1
        y_FourierSeries += comp_i

    li = np.linspace(5, 5, len(list_xGroupMemebers))
    clusterName = "Cluster_" + str(i)
    plt.scatter(list_xGroupMemebers, li,label=clusterName)
    componentenName = "Component_" + str(i)
    plt.plot(t, 2 * comp_i, label=componentenName)


    # Update in Clustering Version
```

```
        m = 0
        while m < len(df_fourierClustered):

            for element in list_xGroupMemebers:

                if math.isclose(df_fourierClustered.loc[m,"x"],element,
                    rel_tol=abs(element*0.001)):
                    df_fourierClustered.loc[m, "Cluster"] = i
                    #print( str(m) + "Element: " + str(element) + " Is close!
                        to " + str(df_fourierClustered.loc[m,"x"]) )


            m += 1

        i += 1



plt.ylabel('Dim 1')
plt.xlabel('Dim 2')
plt.legend()
plt.title("Fourier Clustering")
plt.show()
```

```
# ====== Plot clustered result:
fourier_clusters = df_fourierClustered['Cluster'].unique()



plt.figure()
for cluster in fourier_clusters:
    df_tmpPlot = df_fourierClustered[df_fourierClustered["Cluster"] ==
        cluster]
    plt.scatter(df_tmpPlot["x"],df_tmpPlot["y"],label = ("Cluster " +
        str(cluster)))
plt.legend()
plt.ylabel("Dim 1")
plt.xlabel("Dim 2")
plt.title("Fourier Clustering")
plt.show()



label_Fourier_Result_data = "results_data/" + "fourierSeries_for_" + label +
    ".csv"
label_FourierClustering = "results_data/" + "fourierClustering_for_" + label
    + ".csv"
df_coeffs_FourierSeries.to_csv(label_Fourier_Result_data)
df_fourierClustered.to_csv(label_FourierClustering)

print("Successfully calculated the Fourier Transformation")
print("Results will be stored in 'results_data/fourierResults.csv' ")
print("-------------------------------")
```

```
    return df_coeffs_FourierSeries



#2-D
def clusterData(rawData:pd.DataFrame,label:str) -> pd.DataFrame:
    n_max = 2
    n_dim = len(rawData.columns)
    if n_dim > n_max:
        raise ValueError("Number of Dimensions of handed Data is too high, please
            apply DRA's first")

    #To numpy array
    X = rawData.values

    # ----------- finding the optimal number Clusters K using the Silhouette
        Method --------
    sil = []
    kmax = 10
    k_start = 2

    for k in range(k_start, kmax + 1):
        kmeans_sil = KMeans(n_clusters=k).fit(X)
        labels = kmeans_sil.labels_
        sil.append(silhouette_score(X, labels, metric='euclidean'))

    k_algorithmic = k_start + sil.index(max(sil))
```

```
#


kmeans = KMeans(
    n_clusters=k_algorithmic,
    init ="random",
    random_state=0,
    n_init= 15,
    max_iter= 100,
)



cluster_km = kmeans.fit_predict(X)
df_result_cluster = df_rawData.copy()
df_result_cluster['Cluster']=pd.Series(cluster_km)

# ------- Regression in clusters -------- :
results_C1 = sm.OLS(X[cluster_km == 0, 1], sm.add_constant(X[cluster_km == 0,
    0])).fit()
results_C2 = sm.OLS(X[cluster_km == 1, 1], sm.add_constant(X[cluster_km == 1,
    0])).fit()


print(label)
print("***************** CLUSTER 1 ***************** ")
x = results_C1.summary().as_latex()


with open('output.txt', 'w') as f_output:
    f_output.write(x)
```

```
print(results_C1.summary())
print("***************** CLUSTER 2 ***************** ")
print(results_C2.summary())
# ------- Ploting the clusters -------- :

fig = plt.figure(figsize=(6.5, 4))

plt.xlabel("Dim1")
plt.ylabel("Dim2")

# plot the 2 clusters
plt.scatter(
    X[cluster_km == 0, 0], X[cluster_km == 0, 1],
    s=50, c='lightgreen',
    marker='s', edgecolor='black',
    label='Cluster 1'
)
plt.plot(X[cluster_km == 0, 0], X[cluster_km == 0, 0] * results_C1.params[1]
    + results_C1.params[0], label="Linear regression - Cluster 1")
plt.plot(X[cluster_km == 1, 0], X[cluster_km == 1, 0] * results_C2.params[1]
    + results_C2.params[0], label="Linear regression - Cluster 2")

plt.scatter(
    X[cluster_km == 1, 0], X[cluster_km == 1, 1],
    s=50, c='orange',
    marker='o', edgecolor='black',
    label='Cluster 2'
)
```

```
# Centroids
plt.scatter(
    kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1],
    s=250, marker='*',
    c='red', edgecolor='black',
    label='Centroids'
)
plt.legend(scatterpoints=1)
plt.title(label)
plt.grid()
clusteringResult = "results_plots/" + "Clustering_for_" + label + ".png"
clusteringResultText = "results_plots/" + "Clustering_for_" + label + ".pgf"
plt.savefig(clusteringResult)

#fig.set_size_inches(w=6.5, h=3.5)
plt.savefig(clusteringResultText)
#plt.show()



#Elbow method for finding the optimal K as additional check for user.
distortions = []
for i in range(1, 11):
    km = KMeans(
        n_clusters=i, init='random',
        n_init=10, max_iter=300,
```

```
            tol=1e-04, random_state=0
        )
        km.fit(X)
        distortions.append(km.inertia_)

    # plt.plot(range(1, 11), distortions, marker='o')
    # plt.xlabel('Number of clusters')
    # plt.ylabel('Distortion')
    # elbowName = "results_plots/" + "Elbow_for_" + label + ".png"
    # plt.savefig(elbowName)
    #
    # label_cluster_Result_data = "results_data/" + "Clustering_for_" + label +
        ".csv"
    # df_result_cluster.to_csv(label_cluster_Result_data)

    label_kMeansClustering = "results_data/" + "k-Means_Clustering_for_" + label
        + ".csv"
    df_result_cluster.to_csv(label_kMeansClustering)
    return df_result_cluster


def autoRun(rawData:pd.DataFrame,label) -> dict:
    df_rawData = rawData
    dict_solutions = {}
    dict_result_clustering = {}
    dict_result_fourier = {}


# ------ 0: Data Prep -------
```

```
    # Non Transformation:
    df_sol_nonTrans = df_rawData.copy()

    # Linear Transformation:
    if len(df_rawData.columns) > 2:
        print("----- Running LinTrans (PCA)")
        df_sol_linTrans = applyFeature_extraction_lin(df_rawData,label)
    else:
        df_sol_linTrans = df_rawData
    # Non-Linear Transformation
    if len(df_rawData.columns) > 2:
        print("----- Running NonLinTrans (t-SNE)")

        df_sol_nonLinTrans = applyFeature_extraction_nonLin(df_rawData,label)

    else:
        df_sol_nonLinTrans = df_rawData



# ------ 1: Clustering -------

    # Non Transformation:
    if len(df_rawData.columns) == 2:
        print("----- Running Clustering for Result of NonTrans")
        label_nt = label + "[NonTrans]"
        df_cluster_nonTrans = clusterData(df_sol_nonTrans,label_nt)
        dict_result_clustering["nonTrans"] = df_cluster_nonTrans
```

```
# Linear Transformation:
if len(df_rawData.columns) > 2:
    print("----- Running Clustering for Result of LinTrans")
    label_lt = label + "[LinTrans]"
    df_cluster_linTrans = clusterData(df_sol_linTrans,label_lt)
    dict_result_clustering["linTrans"] = df_cluster_linTrans

# Non-Linear Transformation
if len(df_rawData.columns) > 2:
    print("----- Running Clustering for Result of NonLinTrans")
    label_nlt = label + "[NonLinTrans]"
    df_cluster_nonLinTrans = clusterData(df_sol_nonLinTrans,label_nlt)
    dict_result_clustering["nonLinTrans"] = df_cluster_nonLinTrans

if len(df_rawData.columns == 3):
    #variational clustering along all axis:

    df_rawData_dim1_2 = pd.DataFrame(columns=["1", "2"])
    df_rawData_dim1_2["1"] = df_rawData.iloc[:, 0]
    df_rawData_dim1_2["2"] = df_rawData.iloc[:, 1]
    label12 = label + "clustering_dim12"
    clusterData(df_rawData_dim1_2,label12)
    applyFourierTransform(df_rawData_dim1_2,label12)

    df_rawData_dim2_3 = pd.DataFrame(columns=["2", "3"])
    df_rawData_dim2_3["2"] = df_rawData.iloc[:, 1]
    df_rawData_dim2_3["3"] = df_rawData.iloc[:, 2]
    label23 = label + "clustering_dim23"
    clusterData(df_rawData_dim2_3, label23)
```

```
        applyFourierTransform(df_rawData_dim2_3, label23)


        df_rawData_dim1_3 = pd.DataFrame(columns=["1", "3"])
        df_rawData_dim1_3["1"] = df_rawData.iloc[:, 0]
        df_rawData_dim1_3["3"] = df_rawData.iloc[:, 2]
        label13 = label + "clustering_dim13"
        clusterData(df_rawData_dim1_3,label13)
        applyFourierTransform(df_rawData_dim1_3, label13)


    dict_solutions["Clustering"] = dict_result_clustering



# ------ 2: Fourier -------


    # Non Transformation:
    if(len(df_sol_nonTrans.columns) == 2):
        print("----- Running Fourier for Result NonTrans")
        label_nt = label + "[NonTrans]"
        df_fourier_nonTrans = applyFourierTransform(df_sol_nonTrans,label_nt)
        dict_result_fourier["nonTrans"] = df_fourier_nonTrans


    # Linear Transformation:
    if len(df_rawData.columns) > 2:
        print("----- Running Fourier for Result of LinTrans")


        label_lt = label + "[LinTrans]"


        df_fourier_linTrans = applyFourierTransform(df_sol_linTrans,label_lt)
```

```
        dict_result_fourier["linTrans"] = df_fourier_linTrans


    # Non-Linear Transformation
        print("----- Running Fourier for Result of NonLinTrans")


        label_nlt = label + "[NonLinTrans]"
        df_fourier_nonLinTrans =
            applyFourierTransform(df_sol_nonLinTrans,label_nlt)
        dict_result_fourier["nonLinTrans"] = df_fourier_nonLinTrans


    dict_solutions["Fourier"] = dict_result_fourier



# ------ 3: Solution -------


    return dict_solutions



if __name__ == '__main__':
    df_rawData = load_rawData("dataSet_test.csv")
    #mpl.use('macosx')


    df_rawData = df_rawData.iloc[:, 1:]


    #3D Plot
    # x = df_rawData.iloc[:,0]
    # y = df_rawData.iloc[:,1]
    # z = df_rawData.iloc[:,2]
```

```
# #clu = df_rawData.iloc[:,3]
#
# fig = plt.figure(figsize=(10, 7))
# ax = plt.axes(projection="3d")
#
# # Creating plot
# ax.scatter3D(x, y, z)#, c= clu > 0,cmap = 'coolwarm')
# #ax.scatter3D(x, y, z, cmap = 'coolwarm')
# ax.set_xlabel('Depression')
# ax.set_ylabel('Neuroticism')
# ax.set_zlabel('Life Stress')
# plt.title("3D Plot of raw data")
# plt.savefig("3D-Plot-allOutliers_kMeans.png")
# fig.set_size_inches(w=6.5, h=3.5)
import tikzplotlib


#plt.savefig('histogram.pgf')
#tikzplotlib.save("test.tex")


#plt.show()


autoRun(df_rawData,"outliers_")
print("Finished")
```

Appendix B

Test Data

```
# libraries we need
library(MASS)
library(interactions)
library(dplyr)
## simulate the first dataset
set.seed(1420)
x1 = runif(n = 1500, min = 1, max = 5)
x2 = runif(n = 1500, min = 1, max = 5)

# correlation between the two
p <- 0.075
U  <- rbinom(n=1500,1,p)

# now get the X3 that correlates with X1
x3  <- x1*U + x2*(1-U)
cor(x1, x3)

# get them to predict Y
y <- 2.46960 + (-0.26*x1) + (0.44*x3) + (0.077*x1*x3) +rnorm(250, 0, sqrt(1 -
    (0.0676 + 0.1936 + 0.005929)))

## bind them together
data <- data.frame(Depression = y, Neuroticism =x1, LifeStress = x3)

### run regressions
```

```
lm1 <- lm(Depression~Neuroticism*LifeStress, data = data )
summary(lm1)

# check the interaction
interact_plot(lm1, pred = Neuroticism, modx = LifeStress,
              modx.labels = c("Low", "Mean (0.00)", "High"),
              color.class = c("darkgray","darkgray","#567001"),
              line.thickness = 1.3)

# check which levels are significant
sim_slopes(lm1, pred = Neuroticism, modx = LifeStress)

# get a different dataset with the slope for low levels
data2 <- data
data2 <- mutate(data2, LifeStress_Categotic = case_when(
  data2$LifeStress<= 1.823353 ~ "YES",
  data2$LifeStress>= 4.138416 ~ "NO"))



## add them errors
######## now create a function to add mahalanobis outliers to a specific data
sim_outliers <- function(N, mu, Sigma, MD) {

  n <- length(mu)
  mu1 <- 0*mu

  #
  L <- chol(Sigma)
  T <- diag(Sigma)
```

```
  Lambda <- diag(T)%*%t(L)
  Y <- matrix(0,N,n)

  for (k in 1:N){
    u <- mvrnorm(1, mu1, Sigma)
    u <- Lambda%*%u
    c <- t(mu1)%*%solve(Sigma)%*%mu1-MD[k]**2
    b <- t(mu1)%*%solve(Sigma)%*%u
    a <- t(u)%*%solve(Sigma)%*%u
    root <- (-b+sqrt(b**2-4*a*c))/(2*a)
    Y[k,] <- root[1]*u
  }
  Y <- as.data.frame(Y + sample(mu, N, replace=TRUE))
  return(Y)
}

###
### EXAMPLE ###
table(data2$LifeStress_Categotic)
N <- 321

Sigma <- unname(matrix(cov(data2[which(data2$LifeStress_Categotic ==
    "YES"),1:2]), 2,2))
Sigma[1,1] <- 1
Sigma[2,2] <- 1
mu <- unname(colMeans(data2[which(data2$LifeStress_Categotic == "YES"),1:2]))
MD <- rep(10,N) ## these are the Mahalanobis' distances

# get the added outliers as a data frame
```

```
x <- sim_outliers(N, mu, Sigma, MD)

# change the colnames
colnames(x) <- colnames(data2)[1:2]
# REPLACE DATA
data2[which(data2$LifeStress_Categotic=="YES"),1:2] <- x

# DO THE MAH DIS with and without moderator
data2$mahal1 <- mahalanobis(data2[,1:2], colMeans(data2[,1:2]), cov(data2[,1:2]))

#create new column in x frame to hold p-value for each Mahalanobis distance
    (.001)
data2$p1 <- pchisq(data2$mahal1, df=1, lower.tail=FALSE)

# with mode
# DO THE MAH DIS with and without moderator
data2$mahal2 <- mahalanobis(data2[,1:3], colMeans(data2[,1:3]), cov(data2[,1:3]))

#create new column in x frame to hold p-value for each Mahalanobis distance
data2$p2 <- pchisq(data2$mahal2, df=2, lower.tail=FALSE)

# Typically a p-value that is less than .001 is considered to be an outlier.
dataOut <- data2[data2$p1 < 0.001,]

############# ADD RANDOM OUTLIERS AS WELL
NRand <- 30
# create vars
set.seed(5646468)
var1 <- rnorm(NRand, mean = 7, sd = 1.5)
```

```
var2 <- rnorm(NRand, mean = 6, sd = 1.5)

# data random outliers
datRand <- data.frame(Depression = var1, Neuroticism =var2)

## now add the outliers to data (5 for low levels, 5 for high levels of the
    moderator)
data2[c(194,3,1147,476,81), 1:2] <- datRand[1:5,1:2] # high
data2[c(8,61,992,1471,1154), 1:2] <- datRand[6:10,1:2] # low
data2[c(45,1,100,227,309,104,71,65,415,510,555,645,775,944,1013,1199,1319,1500,1469,1375),
    1:2] <- datRand[11:30,1:2] # medium

# get the randomly introduced outliers data
dataRand <-
    data2[c(194,3,1147,476,81,8,61,992,1471,1154,45,1,100,227,309,104,71,65,415,51
0,555,645,775,944,1013,1199,1319,1500,1469,1375),]

#combine the two outlier datasets
dataOutliers <- rbind(dataOut, dataRand)

# write dataset
write.csv(dataOutliers, 'DataOutliers.csv')
```

Declaration of Authorship

I hereby declare that the thesis submitted is my own unaided work. All direct or indirect sources used are acknowledged as references. I am aware that the thesis in digital form can be examined for the use of unauthorized aid and in order to determine whether the thesis as a whole or parts incorporated in it may be deemed as plagiarism. For the comparison of my work with existing sources I agree that it shall be entered in a database where it shall also remain after examination, to enable comparison with future theses submitted. Further rights of reproduction and usage, however, are not granted here. This paper was not previously presented to another examination board and has not been published.

Dominik Pichler