CZECH TECHNICAL UNIVERSITY
IN PRAGUE

FACULTY OF MECHANICAL ENGINEERING

BACHELOR WORK

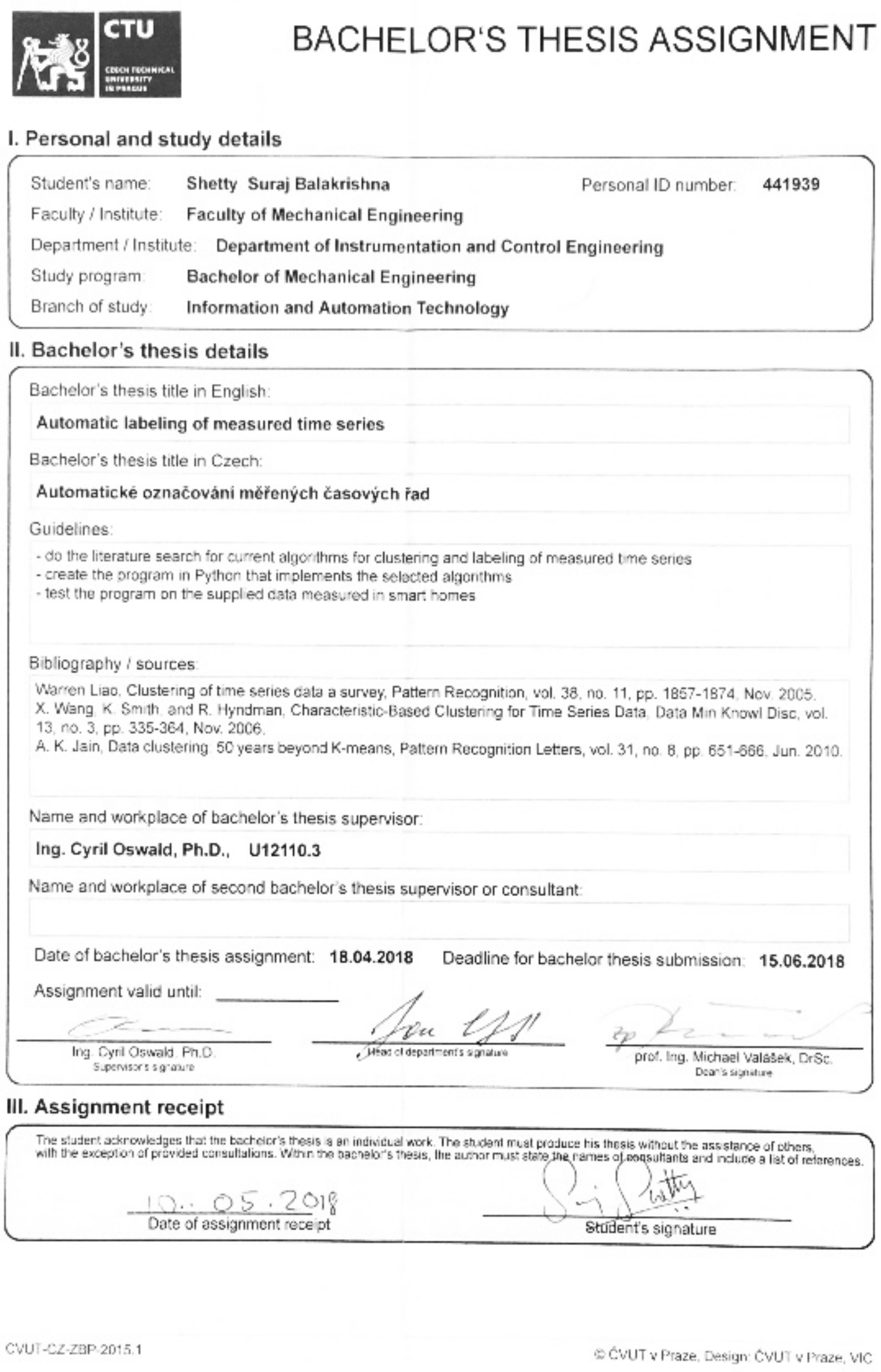
AUTOMATIC LABELLING OF MEASURED TIME SERIES

2018

SURAJ
SHETTY

1. Table of Contents

2. Assignment



BACHELOR'S THESIS ASSIGNMENT

I. Personal and study details

Student's name:	Shetty Suraj Balakrishna
Personal ID number:	441939
Faculty / Institute:	Faculty of Mechanical Engineering
Department / Institute:	Department of Instrumentation and Control Engineering
Study program:	Bachelor of Mechanical Engineering
Branch of study:	Information and Automation Technology

II. Bachelor's thesis details

Bachelor's thesis title in English:

Automatic labeling of measured time series

Bachelor's thesis title in Czech:

Automatické označování měřených časových řad

Guidelines:

- do the literature search for current algorithms for clustering and labeling of measured time series
- create the program in Python that implements the selected algorithms
- test the program on the supplied data measured in smart homes

Bibliography / sources:

Warren Liao, Clustering of time series data a survey, Pattern Recognition, vol. 38, no. 11, pp. 1857-1874, Nov. 2005.
X. Wang, K. Smith, and R. Hyndman, Characteristic-Based Clustering for Time Series Data, Data Min Knowl Disc, vol. 13, no. 3, pp. 335-364, Nov. 2006.
A. K. Jain, Data clustering: 50 years beyond K-means, Pattern Recognition Letters, vol. 31, no. 8, pp. 651-666, Jun. 2010.

Name and workplace of bachelor's thesis supervisor:

Ing. Cyril Oswald, Ph.D., U12110.3

Name and workplace of second bachelor's thesis supervisor or consultant:

Date of bachelor's thesis assignment: **18.04.2018** Deadline for bachelor thesis submission: **15.06.2018**

Assignment valid until: ____________

[illegible] [illegible] [illegible]

Ing. Cyril Oswald, Ph.D.
Supervisor's signature

Head of department's signature

prof. Ing. Michael Valášek, DrSc.
Dean's signature

III. Assignment receipt

The student acknowledges that the bachelor's thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the bachelor's thesis, the author must state the names of consultants and include a list of references.

10. 05. 2018
Date of assignment receipt

[illegible]
Student's signature

CVUT-CZ-ZBP-2015.1

© ČVUT v Praze, Design: ČVUT v Praze, VIC

3. Abstract

This bachelor work aims to conduct a survey of the academic and industrial practices surrounding automatic labelling of measured time series data and further study and classify the different approaches used. Then, the thesis proposes a design and implementation of a program using the data provided by our partner – Energocentrum to label time series temperatures. The analysis, design and implementation are done using Python 3.7.

Cílem této bakalářské práce je provést průzkum akademických a průmyslových postupů, které se týkají automatického označování měřených časových řad a dále studovat a klasifikovat různé použité přístupy. Dále bude v práci představen postup návrhu a implementace programu s využitím dat poskytnutých našim partnerem - Energocentrum pro označení časových řad měřených teplot. Analýza, návrh a implementace jsou prováděny pomocí programu Python 3.7.

4. Introduction

4.1 Description of field

First, what do we mean by data? Webster's define data as a collection of facts, observations or other information related to a question or problem. Data can be structured or unstructured. Structured data is information with a high degree of organization that could be included in databases or spreadsheets and is easily searchable by simple search engine algorithms. Think of how a multiple-choice question forces your answer into a predefined category. Unstructured data is the opposite and is usually text heavy though it may contain video, data or numbers and facts as well. Think of an open field text box that allows you to provide additional comments on a survey. Adding to the complexity Data can also come from a variety of internal and external sources for organizations. The conversation gets interesting when we look at the wide variety of data available to us today, and the powerful analytics that can be applied to that data. Analytics is the science of examining raw data to draw conclusions about the information. It's an exciting field and is dramatically impacting how organizations in many industries are making decisions. The availability of huge volumes of structured and unstructured data sets, combined with advanced computing capabilities. Low cost storage and powerful visualization technology is enabling organizations to gain insight once technologically impossible, or economic impractical. It is also enabling new entities to start-up and scale quickly, which can bring great benefit to the market and to society but can also be very disruptive and challenge the status quo. So where is this data coming from? From market research and social media, to the network of objects we call the internet of things. The world we live in today is creating a constant and ever-increasing stream of data. For most organizations, the data they can access is increasing at a rate of 40% each year which creates significant challenges in the way data is captured and secured, organized, analysed and reported.

Let's quickly touch on some ways that data analytics is impacting business today. First, data is enabling new products and services, creating markets that didn't previously exist and bringing new capabilities to existing markets. Wearables, such as fitness trackers and smart watches are examples of such new products. Second, it is assisting in the advent of sharing economy which is disrupting existing markets with innovative upstarts unseating traditionally secure businesses. Third, data and analytics is driving increased efficiency. For example, retailers can automate and optimize their supply chain. Tailoring offerings for customers, making e-retail services such as same day delivery in major US cities logistically possible. In short, data is providing the organizations the ability to identify growth opportunities, drive innovation, operate more efficiently, and manage risk in new ways. Organizations have always used data in some form to inform their decisions. But the volume, variety and velocity of data available today presents various challenges. Traditional approaches to data analysis, identify a problem or business opportunity, collect data, and use spreadsheets or software to understand it no longer apply. The volume of information today is simply too high, and the time frame's way too short. Confronting these challenges can lead to immense reward. Smart organizations are taking advantage of data analytics to gain a clearer picture of their business. They're using new technology such us data visualization. The presentation of data in a pictorial or graphical format to help decision markers see analytics presented visually and more easily identified new patterns. This kind of insight is powerful and can fundamentally change the speed and sophistication of decision making.

4.2 Scope of the thesis

Organisations deploy various types of algorithms to deal with large data sets and extract appropriate insights and the aim of my project is to evaluate the state of the art and summarise this process and further research and review some of the practices revolving around clustering and labelling algorithms and design and implement a custom algorithm for a local company – Energocentrum, to conduct automated labelling of measured time-series variables.

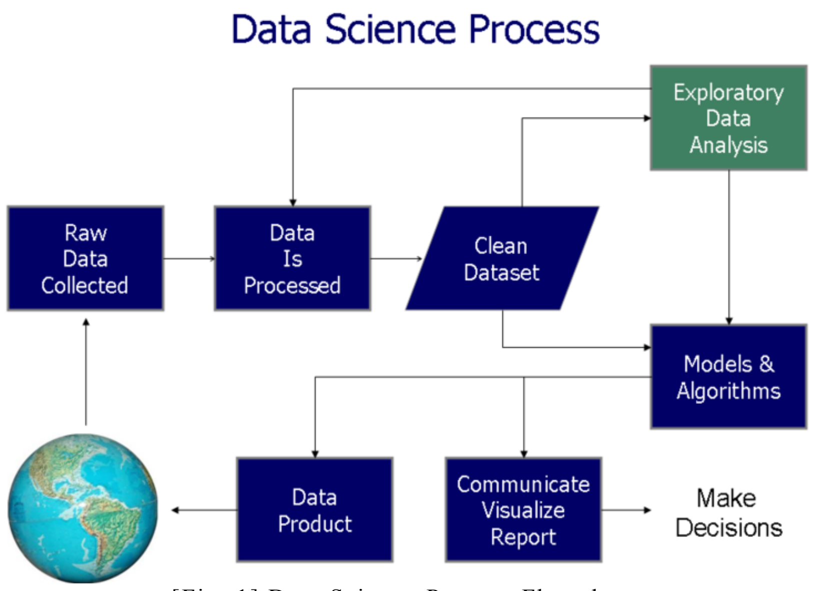
There are three aims to my bachelor work, conduct a research of the state of the art and understand popular practices, design a program within the context of the requirements of my industry partner – Energocentrum and test if the program works as expected.

The ultimate goal of this is to outline some of the use cases of clustering and data labelling algorithms and reflect on the possibilities of implementing them for a “smart system” for

Energocentrum. Principally, not all the applications can be adapted for our particular use, but we can gain perspective on how to tackle different sets of problems.

5. Theory of data science

Data analysis is a process for obtaining raw data and converting it into information useful for decision-making by users. Data is collected and analysed to answer questions, test hypotheses or disprove theories. [1]



[Fig. 1] Data Science Process Flowchart

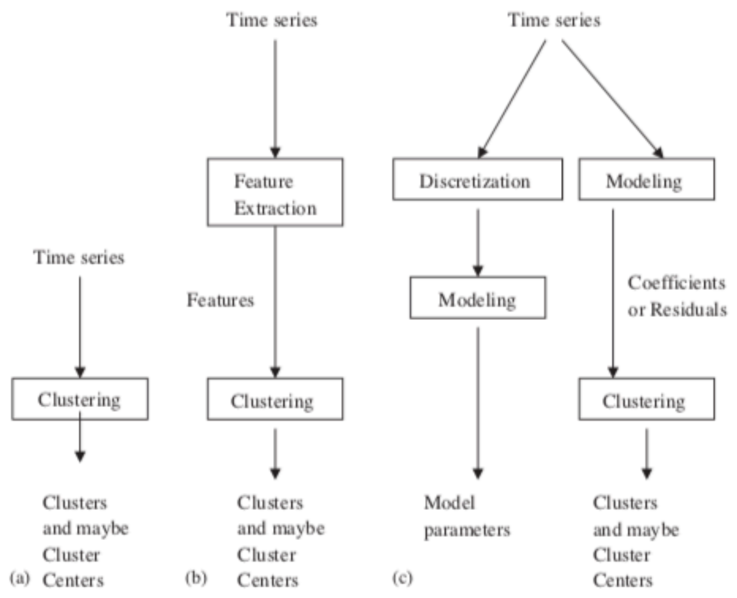
Steps in the Data Science Process. The above figure is a simple linear form of data science process, including five distinct activities that depend on each other. Let's summarize each activity further before we go into the details of each. Acquire includes anything that makes us retrieve data including; finding, accessing, acquiring, and moving data. It includes identification of and authenticated access to all related data. And transportation of data from sources to distributed files systems. It includes waste subset to match the data to regions or times of interest. As we sometimes refer to it as geo-spatial query. The next activity is preparing data, we divide the pre-data activity. Into two steps based on the nature of the activity. Namely, explore data and pre-process data. The first step in data preparation involves literally looking at the data to understand its nature, what it means, its quality and format. It often takes a preliminary analysis of data, or samples of data, to understand it. Therefore, this step is called explore. Once we know more about the data through exploratory analysis, the next step is pre-processing of data for analysis. Pre-processing includes cleaning data, sub-setting or filtering data, creating data, which programs can read and understand, such as modelling raw data into a more defined data model, or packaging it using a specific data format. If there are multiple data sets involved, this step also includes integration of multiple data sources, or streams. The prepared data then would be passed onto the analysis step, which involves selection of analytical techniques to use, building a model of the data, and analysing results. This step can take a couple of iterations on its own or might require data scientists to go back to steps one and two to get more data or package data in a different way. Step four for communicating results includes evaluation of analytical results. Presenting them in a visual way, creating reports that include an assessment of results with respect to success criteria. Activities in this step can often be referred to with terms like interpret, summarize, visualize, or post process. The last step brings us back to the very first reason we do data science, the purpose. Reporting insights from analysis and determining actions from insights based on the purpose you initially defined is what we refer to as the act step. We have now seen all the steps in a typical data science process.

6. Clustering algorithms

The ultimate goal of this section is to outline some of the use cases of clustering and data labelling algorithms and reflect on the possibilities of implementing them for a “smart system” for Energocentrum. Principally, not all the applications can be adapted for our particular use, but we can gain perspective on how to tackle different aspects of the problems.

6.1 Overview

It is important for us to note that the nature of the data we’re working with is not static as the features vary with time and so, our overview will focus on the specifics surrounding clustering for time-series data. The goal of clustering is to identify structure in an unlabelled data set by objectively organizing data into homogeneous groups where the within-group-object similarity is minimized, and between-group-object dissimilarity is maximized [2] resulting in clusters that provide insight into the data set. The procedure we choose to obtain these clusters depends on the requirements of the applications and also the actual type of data. For instance, our "partner" requires us to label temperature values and the type of data we're working with is a time series.



[Fig.2] An illustration of time series clustering approaches: (a) raw-data-based, (b) feature-based, (c) model-based. [2]

As far as time series data are concerned, distinctions can be made as to whether the data are discrete-valued or real-valued, uniformly or non-uniformly sampled, univariate or multivariate, and whether data series are of equal or unequal length. Data that is not uniform can be change to uniformed data and then clustering can be done. We can do this in many ways, from simple sampling method to a rough sampling interval or make a sophisticated model and approximation method. Currently, the majority of clustering algorithms in use were designed to work with static data and the applications for time series data mainly involve converting the time series data into static data and then using the existing clustering methods on this modified data.

In Fig. 2, you can see three of the main procedures for solving such problem: (a) works directly with the time series data whereas (b) involves modifying the given data set to extract some features (from the given data) and then performing clustering and (c) involves converting the raw real time series in order to obtain desired model parameters and then apply traditional clustering methods.

In section 5. – Design and implementation of computational algorithm based on project specifications, we will use a feature-based approach to clustering that will aim to reduce the volume of data as we're working with temperatures with very low variation through our sampling intervals and it is not efficient to consider the entirety of our sampling for our application.

6.2 Applications of time – series clustering

In order to better understand clustering algorithms used for time series data, we have to look at some of the approaches used for other applications. This section aims to investigate approaches in different fields conducted in the past and compare and summarise the same within the above-mentioned approaches.

Generally speaking, there are three fundamental aspects of every particular approach of time-series clustering that is specific to it is application- the clustering algorithm or procedure that is used based on the type of data, the distance measure that is used to correlate the time series data sets (and in order to choose the appropriate measure considerations have to be made that depend on whether the data is discrete-valued or real-valued and whether the time series are equal length or not), and, lastly, the criteria of evaluation we choose in order to stop the clustering process as most of the popular algorithms in use today are iterative (meaning that they are repetitive until) a desirable clustering is obtained.

In the following subsections, we shall look at various applications and compare them based on the particulars mentioned above and also elaborate where necessary.

(a) Raw-data-based approaches

This category houses methods that are compatible with raw data, in either time or frequency domain. Time series themselves are typically sampled at the same intervals but their lengths may or may not be sampled at equal intervals.

(i) Košmelj and Batagelj [3] wrote a paper where they designed an approach for an optimisation problem using clustering multivariate time varying data. They chose to use the relocation clustering procedure as their Clustering algorithm, they used the Euclidean distance as the Distance measure and their Evaluation criteria was the generalised Ward criterion function.

Relocation clustering procedure - The relocation clustering procedure has the following three steps:

Step 1: Start the partition where you have an initial cluster denoted by *C*, having the prescribed *k* number of clusters.
Step 2: Choose an appropriate dissimilarity measure between unit. For each time point compute the dissimilarity matrix and store all resultant matrices computed for all time points for the calculation of trajectory similarity.
Step 3: Find a clustering C', such that C' is better than *C* in terms of the *generalized Ward criterion function*. The clustering C' is obtained from *C* by relocating one member for C_p to C_q or by swapping two members between C_p and C_q, where C_p, $C_q \in C$, p, q =1, 2, ..., k, and $p \neq q$. If no such clustering exists, then stop; else replace *C* by C' and repeat Step 3.

This procedure works only with time series with equal length because the distance between two time series at some cross sections (time points where one series does not have value) is ill defined. This procedure was originally defined for static date and was therefore, modified for their approach. For measuring the dissimilarity between trajectories as required by the procedure, they first introduced a cross-sectional approach- based general model that incorporated the time dimension, and then developed a specific model based on the compound interest idea to determine the time-dependent linear weights. Ultimately, their cross-sectional procedure ignores the correlations between the variables over time and works only with time series of equal length.

Euclidian Distance - Also known as the root mean square distance, and the generalised form Mikowski distance is a basic distance measure between two points. It is computed as follows:

Let x_i and v_j each be a P- dimensional vector. The Euclidean distance is computed as

$$d_E = \sqrt{\sum_{k=1}^{P} (x_{ik} - v_{jk})^2} \tag{1}$$

The root means square distance (or average geometric distance) is simply

$$d_{rms} = \frac{d_E}{n} \tag{2}$$

Mikowski distance is a generalization of Euclidean distance, which is basically defined as

$$d_M = \sqrt[q]{\sum_{k=1}^{P} (x_{ik} - v_{jk})^q} \tag{3}$$

In the above equation, q is a positive integer. A normalized version can be defined if the measured values are normalized via division by the maximum value in the sequence.

Generalised Ward criterion function - The Ward's minimum variance criterion minimizes the total within-cluster variance. To implement this method, at each step we find the pair of clusters that leads to minimum increase in total within-cluster variance after merging. This increase is a weighted squared distance between cluster centers. At the initial step, all clusters are singletons (clusters containing a single point). We can further understand this by defining a problem and iterating some fundamental concept. So, in order to determine the clustering $A^\star \in \Pi_k$, for which

$$P(A^*) = \min_{A \in \Pi_k} P(A) \tag{4}$$

where

$$\Pi_k = \{A: A \text{ is a partition of the set of units } E \text{and } card(A) = k\}$$

and the Ward criterion function P (A) has the form

$$P(A) = \sum_{A \in A} p(A) \tag{5}$$

and

$$p(A) = \sum_{X \in A} d_2{}^2(X, \overline{A}) \tag{6}$$

where $\overline{A}$ is the centre (of gravity) of the cluster A

$$[\overline{A}] = \frac{1}{n_A} \sum_{X \in A} [X] \ , \qquad n_A = card(A), \qquad [X] \in \mathbb{R}^m \tag{7}$$

and $d_2{}^2$ is the squared Euclidean distance. We make a distinction between the (name of) the unit X and its value (description) [X].

However, V. Batagelj used his own earlier proposed generalised form of the Ward criterion function [4] in the aforementioned cross-sectional approach in order to replace the generally used squared Euclidean Distance with the dissimilarity matrix that is calculated in the relocation clustering procedure. He generalised the problem as follows:

Let E ⊂ Ɛ, where Ɛ is the space of units (set of all possible units; the set [Ɛ] of descriptions of units is not necessary a subset of $\mathbb{R}^m$), be a finite set,

$$d : \mathcal{E} \times \mathcal{E} \rightarrow \mathbb{R}_0^+ \tag{8}$$

be a dissimilarity between units and

$$w : \mathcal{E} \rightarrow \mathbb{R}^+ \tag{9}$$

be a weight of units, which is extended to clusters by:

$$\forall X \in \mathcal{E} : w(\{X\}) = w(X) \tag{10}$$

$$A_u \cap A_v = \emptyset \Rightarrow w(A_u \cup A_v) = w(A_u) + w(A_v) \tag{11}$$

Thus, we now obtain the generalized Ward clustering problem where we appropriately alter the formula. And define:

$$p(A) = \frac{1}{2 * w(A)} \sum_{\mathrm{X,Y} \in \mathrm{A}} \mathrm{w(X)} * W(Y) * d(X, Y) \tag{12}$$

Finally, d can now be any dissimilarity on Ɛ and not only the squared Euclidean distance.

And we form a specified number of clusters, the best clustering among all the possible clustering is the one with the minimum generalized Ward criterion function.

(ii) Liao [5] developed a two-step procedure for clustering multivariate time series of equal or unequal length. In the first, in order to convert multivariate real-valued time series data into univariate discrete-valued time series the time is removed from the data and then the next step is to apply the *k-means* or *fuzzy c-means* clustering algorithm. The filtered variable is considered as a state variable process. And then we use the k-means or FCM algorithm to group the converted univariate time series into a number of clusters. These are the expressed as transition probability matrices. In the first step the Euclidean distance is used as the distance measure, but then other distance measures like the symmetric version of Kullback–Liebler distance are used in the second step.

K-means or fuzzy C-means - The k-means (interchangeably called c-means in this study) was first developed more than three decades ago. The main aim behind this algorithm is to minimize an objective function, which is normally chosen to be the total distance between all patterns from their respective cluster centres. Its solution relies on an iterative scheme, which starts with either random assigned initial cluster centres or predefined. The two steps of the c-means algorithm involve the distribution of objects among clusters and the updating of cluster centres are. The algorithm repeats these two steps until the value of the objective function cannot be reduced anymore [6].

Given *n* patterns $\{x_k | k = 1, \ldots, n\}$, c-means determine *c* cluster centres $\{v_i | i=1,\ldots, c\}$, by minimizing the objective function given as function given as

$$\text{Min}\, J_1(U,V) = \sum_{i=1}^{c} \sum_{k=1}^{n} u_{ik} \|x_k - v_i\|^2 \tag{13}$$

(1) $u_{ik} \in \{0,1\}$ $\forall$ i, k, (2) $\sum_{i=1,c} u_{ik} = 1$, $\forall$ k. $\| \cdot \|$ in the above equation conventionally the Euclidean distance measure is used. Albeit, other distance measures could also be used. The iterative solution procedure generally has the following steps:

(1) Choose a cluster center c ($2 \leq c \leq n$) and ε (a small number for stopping the iterative procedure). Set the counter $l = 0$ and the initial cluster centers, $V^{(0)}$, arbitrarily.
(2) Distribute x_k, $\forall$ k to determine $U^{(l)}$ such that J_1 is minimized. This is achieved normally by reassigning x_k to a new cluster that is closest to it.
(3) Revise the cluster centers $V^{(l)}$.
(4) Stop if the change in V is smaller than ε; otherwise, increment l and repeat Steps 2 and 3.

Bezdek [7] formed a more complex objective function subject to fuzzy c-partition constraints

$$\text{Min}\, J_m(U,V) = \sum_{i=1}^{c} \sum_{k=1}^{n} \left(\mu_{ik}\right)^m \|x_k - v_i\|^2 \qquad (14)$$

By differentiating the objective function with respect to v_i (for fixed U) and to μ_{ik} (for fixed V) subject to the conditions, the two equations as a result of this are:

$$v_i = \frac{\sum_{k=1}^{n} \left(\mu_{ik}\right)^m x_k}{\sum_{k=1}^{n} \left(\mu_{ik}\right)^m}, \quad i = 1, \ldots, c. \qquad (15)$$

$$\mu_{ik} = \frac{\left(\frac{1}{\|x_k - v_i\|^2}\right)^{\frac{1}{(m-1)}}}{\sum_{j=1}^{c}\left(\frac{1}{\|x_k - v_i\|^2}\right)^{\frac{1}{(m-1)}}}, \quad i = 1, \dots, c; k = 1, \dots, n. \tag{16}$$

To solve the fuzzy c-means model, an iterative alternative optimization procedure is required. In order to conduct the procedure, again, a few parameters need to be specified - the number of clusters, *c*, and the weighting coefficient. The FCM algorithm has the following steps:

(1) Choose $c(2 \le c \le n)$, $m(1<m<\infty)$, and ε (a small number for stopping the iterative procedure). Set the counter l=0 and initialize the membership matrix, $U^{(l)}$.
(2) Calculate the cluster centre, $v_i^{(l)}$.
(3) Recalculate membership matrix $U^{(l+1)}$ if $x_k \neq v_i^{(l)}$ else, set $\mu_{jk} = 1$ (0) if j = (≠)i.
(4) Calculate $\Delta = \|U^{(l+1)} - U^{(l)}\|$ till you get $\Delta > \varepsilon$.

This group of algorithms works better with time series of equal length because the concept of cluster centers becomes unclear when the same cluster contains time series of unequal length.

Kullback–Liebler distance - Let P_1 and P_2 be matrices of transition probabilities of two Markov chains (MCs) with *s* probability distributions each and $p_{1_{ij}}$ and $p_{2_{ij}}$ be the i − > j transition probability in P_1 and P_2. The asymmetric Kullback–Liebler distance of two probability distributions is

$$d(p_{1_i}, p_{2_i}) = \sum_{j=1}^{s} p_{1_{ij}} \log\left(\frac{p_{1_{ij}}}{p_{2_{ij}}}\right) \tag{17}$$

The symmetric version of Kullback–Liebler distance of two probability distributions is

$$D\left(P_{1_i}, P_{2_i}\right) = \left[\frac{d\left(p_{1_i}, p_{2_i}\right) + d\left(p_{2_i}, p_{1_i}\right)}{2}\right] \tag{18}$$

The average distance between P_1 and P_2 is then

$$d(P_1, P_2) = \sum_{i=1,s} \frac{D\left(p_{1_i}, p_{2_i}\right)}{s} \tag{19}$$

(b) Feature-based approaches

Clustering based on raw data implies working with high dimensional space especially for data collected at fast sampling rates. It is also not desirable to work directly with the raw data that are highly noisy. Several feature-based clustering methods have been proposed to address these concerns. Though most feature extraction methods are generic in nature, the extracted features are usually application dependent. That is, one set of features that work well on one application might not be relevant to another. Some studies even take another feature selection step to further reduce the number of feature dimensions after feature extraction. They all can handle series with unequal length because the feature extraction operation takes care of the issue. For a multivariate time series, features extracted can simply be put together or go through some fusion operation to reduce the dimension and improve the quality of the clustering results, as in classification studies.

(i) In order to map brain activity through Functional MRI using two algorithms: *k-means* and *Ward' s hierarchical clustering* as his clustering procedure, Goutte et al. [8] clustered fMRI time series (P slices of images) in groups of voxels. Instead of using the raw fMRI time series the date set was filtered and the *cross-correlation function* between the fMRI activation and the paradigm (or stimulus) was extracted and used. For each voxel *j* in the image, y_j denotes the measured fMRI time series and *p* is the activation stimulus (assumed a square wave but not limited to), common to all j. The cross-correlation function is defined as where p(i)=0 for i<0 or i>P and T is of the order of the stimulus period. The distance measure used was Euclidean distance

In a subsequent paper Goutte et al. [9] showed that feature-based clustering is an effective meta-analysis tool in assessing the differences and similarities between results procured by particular voxel analyses and this further illustrated the potential of the feature-based clustering method. Using the k- means algorithm, they investigated the performance of different information criteria for determining the optimal number of clusters. Initially, only two features were used - the delay and strength of activation measured on a voxel-by-voxel basis to show that one could identify the regions with significantly different delays and activations.

(ii) Vlachos et al. [10] presented an approach to perform incremental clustering of time series at various resolutions using the Haar wavelet transform. First, the Haar wavelet decomposition is computed for all-time series. Then, the k- means clustering algorithm is applied, starting at the coarse level and gradually progressing to finer levels. The final centers at the end of each resolution are reused as the initial centers for the next level of resolution. Since the length of the data reconstructed from the Haar decomposition doubles as we progress to the next level, each coordinate of the centers at the end of level i is doubled to match the dimensionality of the points on level i + 1. The clustering error is computed at the end of each level as the sum of number of incorrectly clustered objects for each cluster divided by the cardinality of the dataset.

(c) Model-based approaches

This class of approaches considers that each time series is generated by some kind of model or by a mixture of underlying probability distributions. Time series are considered similar when the models characterizing individual series or the remaining residuals after fitting the model are similar. Like feature-based methods, model-based methods are capable of handling series with unequal length as well through the modelling operation. For those methods that use log-likelihood as the distance measure, the model with the highest likelihood is concluded to be the cluster for the data being tested.

(i) Ramoni et al. [11] presented BCD: a Bayesian algorithm for clustering by dynamics. Given a set S of n numbers of univariate discrete-valued time series, BCD transforms each series into a Markov chain (MC) and then clusters similar MCs to discover the most probable set of generating processes. BCD is basically an unsupervised agglomerative clustering method. Considering a partition as a hidden discrete variable C, each state C_k of C represents a cluster of time series, and hence determines a transition matrix. The task of clustering is regarded as a Bayesian model selection problem with the objective to select the model with the maximum posterior probability. Since the same data are used to compare all models and all models are equally likely, the comparison can be based on the marginal likelihood $p(S|MC)$, which is a measure of how likely the data are if the model MC is true. The similarity between two estimated transition matrices is measured as an average of the symmetrized Kullback–Liebler distance between corresponding rows in the matrices. The clustering result is evaluated mainly by a measure of the loss of data information induced by clustering, which is specific to the proposed clustering method.

(ii) They also presented a Bayesian clustering algorithm for multivariate time series [12]. The algorithm searches for the most probable set of clusters given the data using a similarity-based heuristic search method. The measure of similarity is an average of the Kullback–Liebler distances between comparable transition probability tables. The similarity measure is used as a heuristic guide for the search process rather than a grouping criterion. Both the grouping and stopping criteria are based on the posterior probability of the obtained clustering. The objective is to find a maximum posterior probability partition of set of MCs.

7. Design and implementation of computational algorithm based on project specifications

For this section, my aim is to explain the feature-based approach I have designed in Python in order to do labelling of time-series temperature data set provided to me by company I'm working with – Energocentrum. Please note that even though I have attached the source code below for your perusal I will only explain "key areas" of the code as (you will notice later that) a lot of commands are repeated for subsequent data sets. Furthermore, for the sake of visualisation I have reformatted the excerpts of code with spaces, but they're accurately attached in the appendix section below and the files in the CD.

After importing the provided .csv to .xlsx we can see that we have 1848 sensor readings for the month of January taken over a twenty-four-hour period of time in five-minute intervals – this yield 8641 columns of data for us to work with. However, the main aim of the task is to perform labelling of temperature within this dataset, so we have to perform a certain amount of filtering on this dataset, in order to only work with the appropriate values, then extract our chosen features from the data set and additionally adhere to the standard procedure of choosing a clustering procedure, distance measure and evaluation criteria. Finally, we need to graphically visualise our analysis in an intuitive manner through figures to showcase our findings.

Data filtering – the .csv provided to us has some values from sensors that are of no use to our algorithm (and approach) such as 'pump operation', 'energy_meter,actual', 'energy_meter,cumulative' etc., so we first eliminate these in the following manner.

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import norm as sn
from scipy.stats import skew, kurtosis
filename = './data/tagged_data_01_2017.csv' # csv file to load
data = pd.read_csv(filename, delimiter=';', keep_default_na=False,
na_values=['NaN'], dtype='unicode') # load data to pandas DataFrame
```

```
data.set_index('time', inplace = True) # set column 'time' as index

data.index = pd.to_datetime(data.index, format='%d/%m/%y %H:%M') # convert index to
datetime objects
data = data.apply(pd.to_numeric) # convert data id Data Frame to numbers
columns = list(data.columns.values) # save Data Frame header to list
# print(data[columns[0]]) # print timeseries stored in Data Frame data indexed by
list of columns names (header)
#data[columns[0][:]].plot() # plot data with indexing by list of columns names
(header)

data = data[data.index.dayofweek < 5] # filtering weekday
data = data.filter(regex=("^.*temperature.*$")) #filtering the columns for string –
"temperature"
```

The above excerpt is from the source code in the file 'january_process_data.py'. As you can see we first import the relevant libraries, and this yields a desirable data frame for us to work with. The values from the data set are indexed by time and is filtered such that we only use temperature values, also the values of temperature readings from weekends are removed as a lot of sub systems are shut down over the weekend within the building these readings are taken from (as observed by the numerous weekend values relative to the weekday values).

```
data_indoor = data.filter(regex=("indoor_air_temperature.*"))
data_outdoor = data.filter(regex=("outdoor_air_temperature.*"))
data_return = data.filter(regex=("supply_water_temperature.*"))
data_supply = data.filter(regex=("return_water_temperature.*"))
data_service = data.filter(regex=("warm_service_water_temperature.*"))
```

We then further index our data frame in order to distinguish between the different kinds of readings. From our observation of the provided .csv file there are five different types of temperature readings that are expected – indoor temperature, outdoor temperature, water supply temperature, return water supply temperature and lastly, warm service water temperature. We now have appropriately filtered data sets to work with.

Feature extraction – the next step is to extract our chosen features from the data frame. For my approach I will use basic statistical features from the dataset for my clustering algorithm- mean values, standard deviation, skewness and kurtosis. Few things to note are that we have plotted the five temperature types mentioned earlier and have made certain thresholds between which we expect our temperatures to lie, thereby getting rid of corrupt data points within the respective data sets and we run the same set of commands for the different temperature data sets in order to extract the similar features and compare them.

For each of the five data sets of a different temperature type (indoor temperature, outdoor temperature, etc.,) we are a) grouping the values, indexed by the respective statistical feature (mean value, standard deviation, etc.,), based on our observationally established thresholds; we are b) reshaping the data frame into a [n 1] matrix (where n is the number of data points for a given statistical feature and temperature type specific data set); and we are c) eliminating non-integer values. Steps a) and c) are taken in order to eliminate corrupt data points. Lastly, for mean values and standard deviation we are indexing all values by day (so in this case n is equal to the numbers of weekdays in the month) and for skewness and kurtosis we are indexing the values by month (so we expect one value per temperature type per month for each of these two statistical features).

```
indoordaymeans = data_indoor.groupby(data_indoor.index.day).agg(lambda x:
np.nanmean(x[(x>15) & (x<40)]))

indoordaystds = data_indoor.groupby(data_indoor.index.day).agg(lambda x:
np.nanstd(x[(x>15) & (x<40)]))

indoormonthskew = data_indoor.groupby(data_indoor.index.month).agg(lambda x:
skew(x[(x>15) & (x<40)]))

indoormonthkurt= data_indoor.groupby(data_indoor.index.month).agg(lambda x:
kurtosis(x[(x>15) & (x<40)]))

np_in_temp_means_nan = np.reshape(indoordaymeans.as_matrix(), -1)
np_in_temp_stds_nan = np.reshape(indoordaystds.as_matrix(), -1)
np_in_temp_skew_nan = np.reshape(indoormonthskew.as_matrix(), -1)
np_in_temp_kurt_nan = np.reshape(indoormonthkurt.as_matrix(), -1)

np_in_temp_means = np_in_temp_means_nan[~np.isnan(np_in_temp_means_nan)]
np_in_temp_stds = np_in_temp_stds_nan[~np.isnan(np_in_temp_stds_nan)]
np_in_temp_skew = np_in_temp_skew_nan[~np.isnan(np_in_temp_skew_nan)]
np_in_temp_kurt = np_in_temp_skew_nan[~np.isnan(np_in_temp_kurt_nan)
```

For indoor temperatures we are ignoring values of temperature above 40°C and below 15°C, and as mentioned earlier for the mean values and standard deviation values we are indexing the value by days and for skewness and kurtosis we are indexing by month.

```
outdoordaymeans = data_outdoor.groupby(data_outdoor.index.day).agg(lambda x:
np.nanmean(x[x<15]))
outdoordaystds = data_outdoor.groupby(data_outdoor.index.day).agg(lambda x:
np.nanstd(x[x<15]))
outdoormonthskew= data_outdoor.groupby(data_outdoor.index.month).agg(lambda x:
skew(x[x<15]))
outdoormonthkurt= data_outdoor.groupby(data_outdoor.index.month).agg(lambda x:
kurtosis(x[x<15]))

np_out_temp_means_nan = np.reshape(outdoordaymeans.as_matrix(), -1)
np_out_temp_stds_nan = np.reshape(outdoordaystds.as_matrix(), -1)
np_out_temp_skew_nan = np.reshape(outdoormonthskew.as_matrix(), -1)
np_out_temp_kurt_nan = np.reshape(outdoormonthkurt.as_matrix(), -1)

np_out_temp_means = np_out_temp_means_nan[~np.isnan(np_out_temp_means_nan)]
np_out_temp_stds = np_out_temp_stds_nan[~np.isnan(np_out_temp_stds_nan)]
np_out_temp_skew = np_out_temp_skew_nan[~np.isnan(np_out_temp_skew_nan)]
np_out_temp_kurt = np_out_temp_kurt_nan[~np.isnan(np_out_temp_kurt_nan)]
```

In a similar manner, we are ignoring values of temperature above 15 °C for outdoor temperature reading and conducting indexing in keeping with our designed procedure.

Lastly, we have to make a grouping of our data based on the statistical features and saving it for further processing.

```
np.save('./evaluation/data_january',[[np_in_temp_means,np_out_temp_means,np_supply_
temp_means,np_return_temp_means,np_warm_temp_means],
[np_in_temp_stds,np_out_temp_stds,np_supply_temp_stds,np_return_temp_stds,np_warm_t
emp_stds],
[np_in_temp_skew,np_out_temp_skew,np_supply_temp_skew,np_return_temp_skew,np_warm_t
emp_skew],

[np_in_temp_kurt,np_out_temp_kurt,np_supply_temp_kurt,np_return_temp_kurt,np_warm_t
emp_kurt]])

daymeans = indoordaymeans.join
```

```
(outdoordaymeans.join(supplydaymeans.join(returndaymeans.join(warmdaymeans))))

daystds = indoordaystds.join
(outdoordaystds.join(supplydaystds.join(returndaystds.join(warmdaystds))))

monthskew = indoormonthskew.join
(outdoormonthskew.join(supplymonthskew.join(returnmonthskew.join(warmmonthskew))))

monthkurt = indoormonthkurt.join
(outdoormonthkurt.join(supplymonthkurt.join(returnmonthkurt.join(warmmonthkurt))))

daymeans.to_pickle('./evaluation/data_daymeans_january.pkl')
daystds.to_pickle('./evaluation/data_daystds_january.pkl')
monthkurt.to_pickle('./evaluation/data_monthkurt_january.pkl')
monthskew.to_pickle('./evaluation/data_monthskew_january.pkl')
```

This is realised through the above excerpt of our code.

We now have a list containing all our filtered data that is saved in the directory in a '.npy' format and we will use this for training our algorithm or the so-called supervised learning part (and also for visualisation purposes). Additionally, we also have four strings of data for each statistical feature from every temperature type, and we have saved this in our directory in '.pkl' format and we will use this for the unsupervised learning part.

For the aforementioned visualisation purpose, I've written another script – 'january_plot_distributions.py', where we can see the distribution of our filtered data, this gives you rough idea of the spread of the data points we're working with.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import norm as sn
from scipy.stats import skew, kurtosis

data = np.load('./evaluation/data_january.npy')

np_in_temp_means = data[0,0]
np_out_temp_means = data[0,1]
np_supply_temp_means = data[0,2]
```

```
np_return_temp_means = data[0,3]
np_warm_temp_means = data[0,4]
```

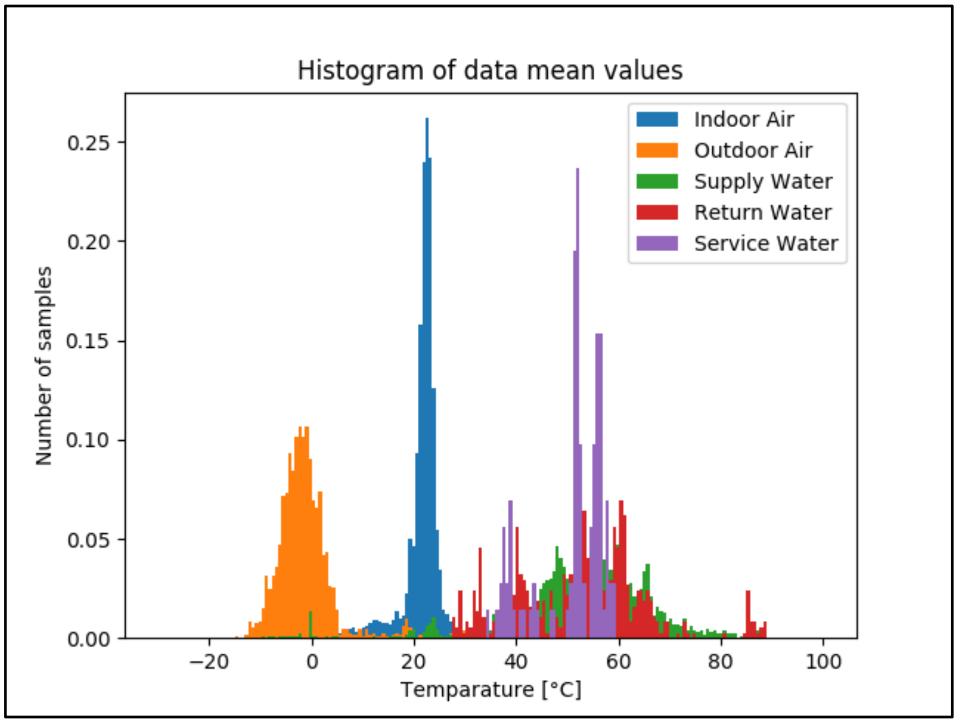
As is the standard procedure, we first import the relevant libraries. Then we load the '.npy' file, appropriately rename and call the datasets from within the list in the file. Although the above excerpt is only for mean values, I have done the same for the other statistical features as well.

```
bins_mean = np.linspace(-30, 100, 200) #creating bins from -30 to 100 to 200
samples ---> 199 bins
```

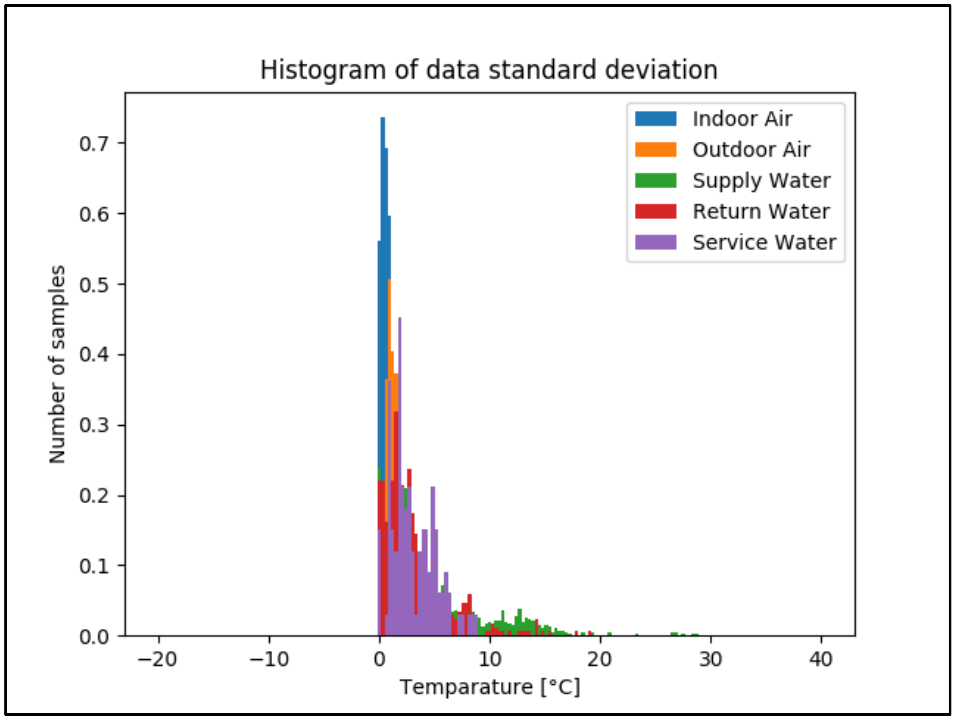
Then we set the intervals and samples for our graph. Again, this is done for all four statistical features.

```
plt.figure()
plt.gca().set_prop_cycle(None)
plt.hist(np_in_temp_means, bins=bins_mean, density=True, label='Indoor Air')
plt.hist(np_out_temp_means, bins=bins_mean, density=True, label='Outdoor Air')
plt.hist(np_supply_temp_means, bins=bins_mean, density=True, label='Supply Water')
plt.hist(np_return_temp_means, bins=bins_mean, density=True, label='Return Water')
plt.hist(np_warm_temp_means, bins=bins_mean, density=True, label='Service Water')
plt.legend()
plt.xlabel('Temparature [°C]')
plt.ylabel('Number of samples')
plt.title('Histogram of data mean values')
```

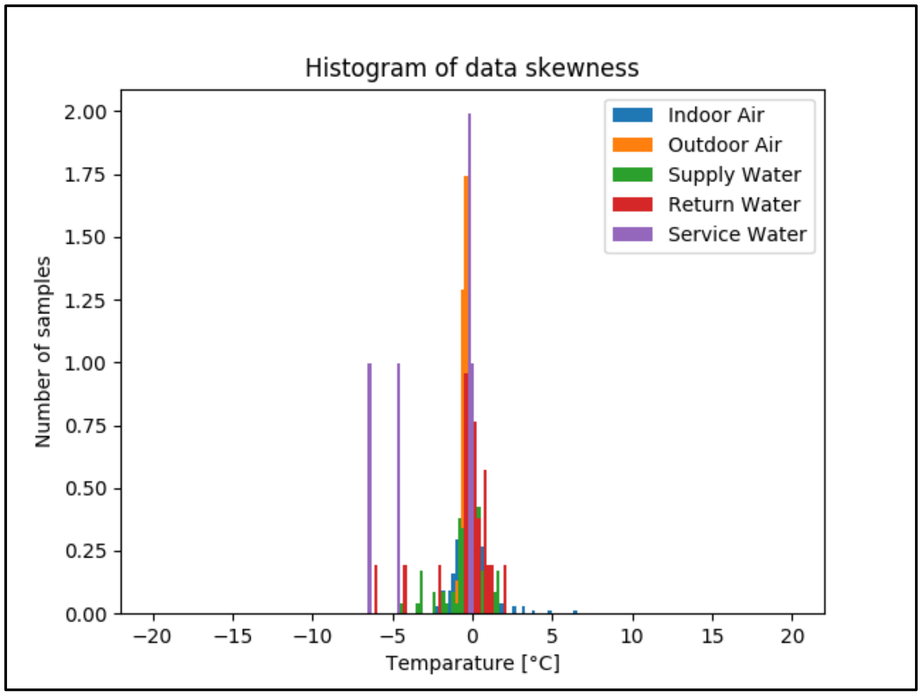
And finally, we can plot the graph through the above excerpt.



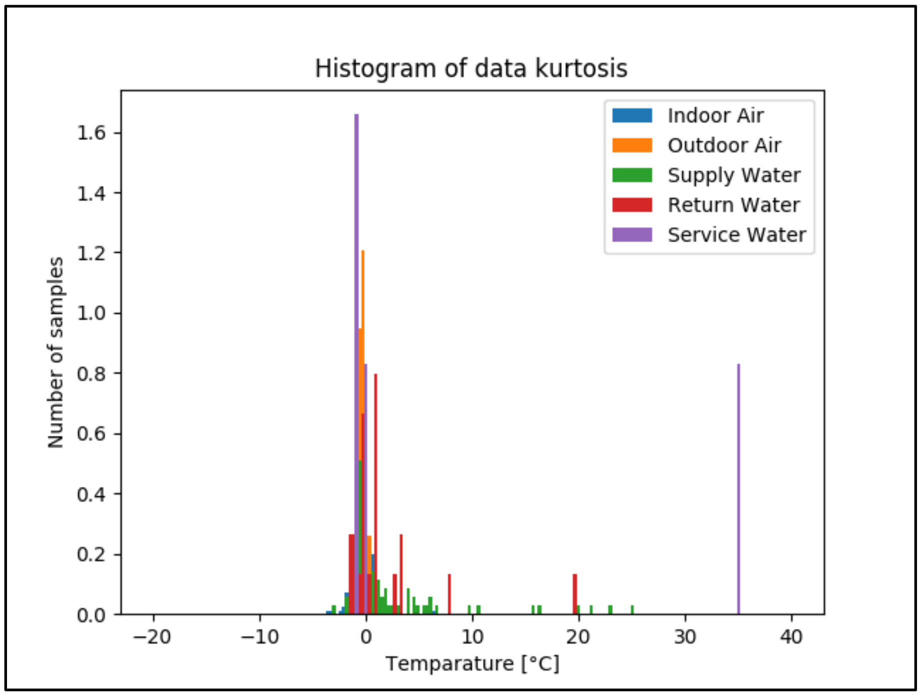
[Fig. 3] Graph of Mean Values



[Fig. 4] Graph of Standard Deviation Values



[Fig. 5] Graph of Skewness Values



[Fig. 6] Graph of Kurtosis Values

Clustering procedure – for my approach, I have chosen to run our extracted statistical features derived from our filtered datasets through the k-means algorithm available in the ‘scikit-learn’ library. The below code is from the ‘january_k_means_train.py’ file.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import norm as sn
from scipy.stats import skew, kurtosis
from sklearn.cluster import KMeans
import pickle

data = np.load('./evaluation/data_january.npy')

np_in_temp_means = data[0,0].reshape(-1, 1) #loading data from array from file
np_out_temp_means = data[0,1].reshape(-1, 1)
np_supply_temp_means = data[0,2].reshape(-1, 1)
np_return_temp_means = data[0,3].reshape(-1, 1)
np_warm_temp_means = data[0,4].reshape(-1, 1)
```

As usual, we first import the relevant libraries. And again, we load the ‘.npy’ file, appropriately rename and call the datasets from within the list in the file while also reordering the data points appropriately (as per the requirement of the algorithm designed by ‘scikit-learn’) so we can then run the clustering procedure.

```
CLUSTERS_NUMBER = 2                                 #setting no. of clusters

kmeans_in_temp_means = KMeans(n_clusters=CLUSTERS_NUMBER,
random_state=0).fit(np_in_temp_means)               #running kmeans training
algorithm for 2 clusters on indoor temp values
kmeans_out_temp_means = KMeans(n_clusters=CLUSTERS_NUMBER,
random_state=0).fit(np_out_temp_means)
kmeans_supply_temp_means = KMeans(n_clusters=CLUSTERS_NUMBER,
random_state=0).fit(np_supply_temp_means)
kmeans_return_temp_means = KMeans(n_clusters=CLUSTERS_NUMBER,
random_state=0).fit(np_return_temp_means)
kmeans_warm_temp_means = KMeans(n_clusters=CLUSTERS_NUMBER,
random_state=0).fit(np_warm_temp_means)
```

The above excerpt shows the configuration set for the mean values, but the same commands are written in the script for the other statistical features as well (for all the temperature types). As seen above, we have set the number of cluster (‘n-clusters’) as two and when we execute the script we will have two centroids per dataset.

```
kmeans_dict = { 'kmeans_in_temp_means': kmeans_in_temp_means,
                'kmeans_out_temp_means': kmeans_out_temp_means,
                'kmeans_supply_temp_means': kmeans_supply_temp_means,
                'kmeans_return_temp_means': kmeans_return_temp_means,
                'kmeans_warm_temp_means': kmeans_warm_temp_means,
```

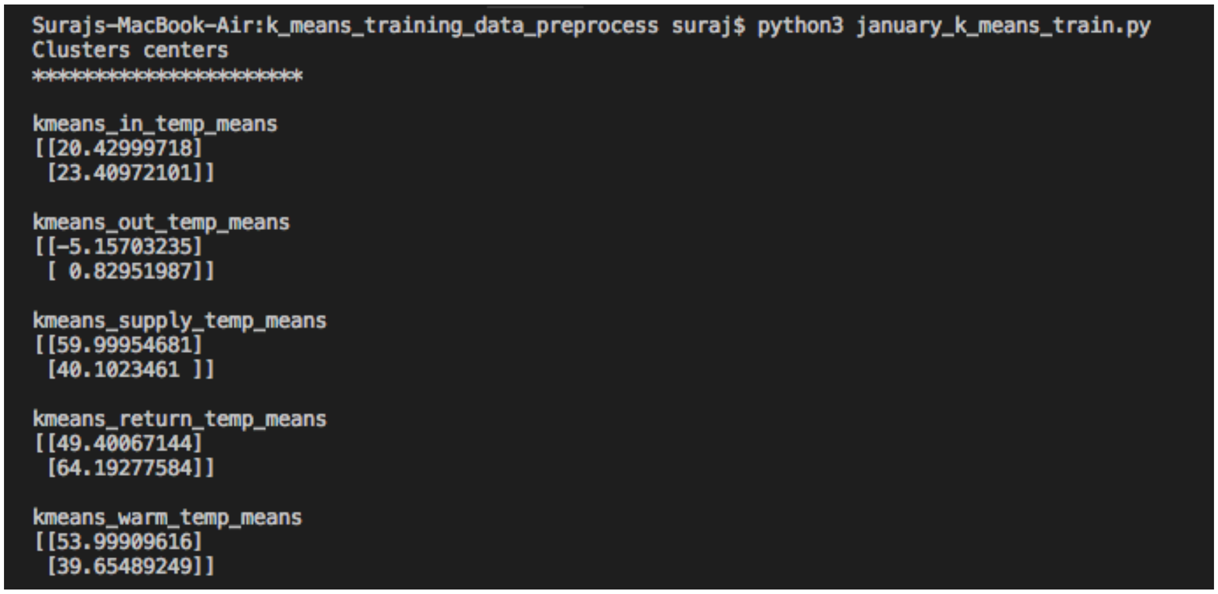
We then save the calculated clusters to a dictionary, which is a type of array, for further use. And this is done for all the different temperature data sets we have. This concludes the supervised learning part of our procedure. We now have centroids for each data set and these centroids will be later used to predict our data labels.

```
print('Clusters centers\n**********************')
for kmeans_name, kmeans_value in kmeans_dict.items():
 #for cycle to show clusters

    print('\n'+kmeans_name)
    print(kmeans_value.cluster_centers_)
      #read cluster centers from kmeans algorithm

with open('./evaluation/kmeans_january.pkl', 'wb') as f:
      #open file, wb--> write binary (way of working with file---> as binary
      format), f: pointer to file
    pickle.dump(kmeans_dict, f)
```

Then we just print the centroid values for our reference and also save them for future use by execute the set of commands in the above excerpt from the script.



```
Surajs-MacBook-Air:k_means_training_data_preprocess suraj$ python3 january_k_means_train.py
Clusters centers
**********************

kmeans_in_temp_means
[[20.42999718]
 [23.40972101]]

kmeans_out_temp_means
[[-5.15703235]
 [ 0.82951987]]

kmeans_supply_temp_means
[[59.99954681]
 [40.1023461 ]]

kmeans_return_temp_means
[[49.40067144]
 [64.19277584]]

kmeans_warm_temp_means
[[53.99909616]
 [39.65489249]]
```

[Fig. 7] Cluster Centers for Mean Values

Distance Measure – the K-means algorithm in the Scikit-learn library uses raw distance between two points so by default this is our distance measure. In order to accommodate this, we have designed our program such that our data is fed to the k-means algorithm in 2-d arrays.

Evaluation criteria – Next we're going to use our centroids to predict the data labels. And we will then create a variables that could be used in confusion matrix based on the Bayesian rule to help visualise the performance of our program and furthermore, we will also use common statistical measures – sensitivity, specificity and accuracy to measure our performance.

From file 'january_k_means_evaluate.py'

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import norm as sn
from scipy.stats import skew, kurtosis
from sklearn.cluster import KMeans
import pickle

with open('./evaluation/kmeans_january.pkl', 'rb') as f:
    kmeans = pickle.load(f)

daymeans = pd.read_pickle('./evaluation/data_daymeans_january.pkl')
daystds = pd.read_pickle('./evaluation/data_daystds_january.pkl')
```

First, we import the relevant libraries and the '.pkl' files from our previous scripts in order to obtain all the statistical features for our different temperature type and also their respective cluster centroids (after running the k-means algorithm).

```
temp_daymeans = daymeans.filter(regex=("^.*temperature.*$"))
temp_daystds = daystds.filter(regex=("^.*temperature.*$"))
```

We have to again, filter out any non-temperature values from the ',pkl' files in the first ('january_process_data.py') script.

```
names= ['indoor', 'outdoor', 'return', 'supply', 'warm_service']
evaluation = []
```

As seen above we have created lists to work with our values and to store them for later use.

```
for column in temp_daymeans:
    try:       #try and if error execute exception

        temp_daymeans_col = temp_daymeans[column].as_matrix()
        temp_daystds_col = temp_daystds[column].as_matrix()

#going through temp_daymeans and prepare it for running the unsupervised part of k-
means algorithm
```

```
        temp_daymeans_col = 
temp_daymeans_col[~np.isnan(temp_daymeans_col)].reshape(-1,1)

 #filtering out nan values and reshape to 2-d array for the kmeans algorithm 
(required standard)

        temp_daystds_col = temp_daystds_col[~np.isnan(temp_daystds_col)].reshape(-
1,1)

    except:
#exception skip iteration

        continue
    score_means = []
#evaluation part; append 'score' values; score---->

    score_means.append(kmeans['kmeans_in_temp_means'].score(temp_daymeans_col))
    score_means.append(kmeans['kmeans_out_temp_means'].score(temp_daymeans_col))
    score_means.append(kmeans['kmeans_supply_temp_means'].score(temp_daymeans_col))
    score_means.append(kmeans['kmeans_return_temp_means'].score(temp_daymeans_col))
    score_means.append(kmeans['kmeans_warm_temp_means'].score(temp_daymeans_col))
# for unknown mean values, calculate the distance from the fitted kmeans of known 
variable (from the training part)

    score_stds = []
    score_stds.append(kmeans['kmeans_in_temp_stds'].score(temp_daystds_col))
    score_stds.append(kmeans['kmeans_out_temp_stds'].score(temp_daystds_col))
    score_stds.append(kmeans['kmeans_supply_temp_stds'].score(temp_daystds_col))
    score_stds.append(kmeans['kmeans_return_temp_stds'].score(temp_daystds_col))
    score_stds.append(kmeans['kmeans_warm_temp_stds'].score(temp_daystds_col))
```

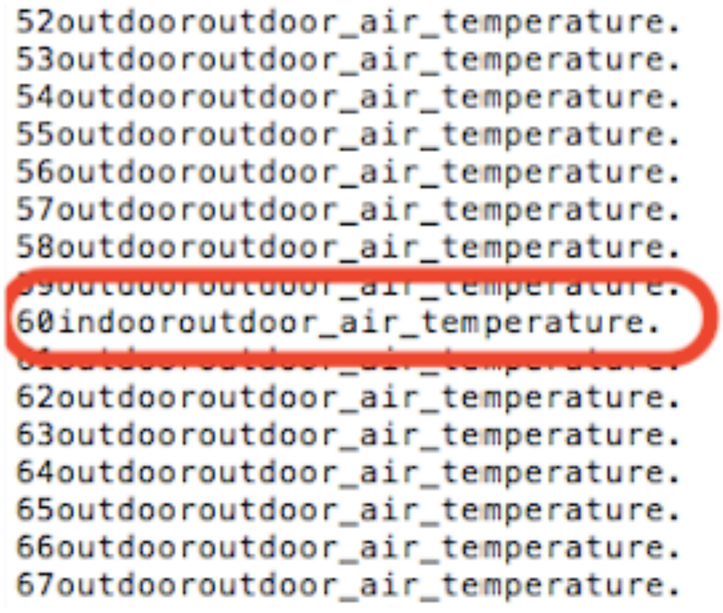
The above excerpt from the script shows that, we have restructured the temperature values of our data sets into columns (or 2-d array) in order to make it work with the k-means algorithm (as mandated by the library), and we have also filtered out non-integer ('nan' values). Then we start the actual evaluation part where we're first getting the distance between all the points in a particular statistical feature's data set and then measuring the distance between those points and the centroids we have obtained from the training part of the program.

```
  scores = np.array(score_means) + np.array(score_stds) + np.array(score_skew) + 
np.array(score_kurt)  #finally, score= sum of all scores from of each feature for 
the data point

    evaluation.append([column, names[np.argmax(scores)]])
#returns name of the variable with high score value (closest to 0 is "winner")

np.save('./evaluation/january_evaluation', evaluation)
#saving to file
```

Finally, we aggregate all these scores and eliminate all except the values with the highest score as these are the values closest to our centroids. And then we save these values in order to evaluate them later.



```
52outdooroutdoor_air_temperature.
53outdooroutdoor_air_temperature.
54outdooroutdoor_air_temperature.
55outdooroutdoor_air_temperature.
56outdooroutdoor_air_temperature.
57outdooroutdoor_air_temperature.
58outdooroutdoor_air_temperature.
59outdooroutdoor_air_temperature.
60indooroutdoor_air_temperature.
61outdooroutdoor_air_temperature.
62outdooroutdoor_air_temperature.
63outdooroutdoor_air_temperature.
64outdooroutdoor_air_temperature.
65outdooroutdoor_air_temperature.
66outdooroutdoor_air_temperature.
67outdooroutdoor_air_temperature.
```

[Fig. 7] The circled value is an example of an incorrectly labelled data point in the 'january_evaluation_stats.py' file.

In the 'january_evaluation_stats.py' file we have two columns the first one being our predicted label and the second one being the actual label of the data.

Next, we have our final script (at least for the basic implementation of my design). From the execution of the file 'january_evaluation_stats.py' the aim is to be able to visualise our results and realise the performance of our design.

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import norm as sn
from scipy.stats import skew, kurtosis

evaluation = np.load('./evaluation/january_evaluation.npy')
```

We again, import the libraries we're using and the results of the evaluation from the previous script.

```python
#preparing of variables for statistic evaluation (based on confusion matrix)

n_indoor = 0
n_notindoor = 0
```

```
n_outdoor = 0
n_notoutdoor = 0

indoor_positive = 0
indoor_falsepositive = 0
indoor_negative = 0
indoor_falsenegative = 0

outdoor_positive = 0
outdoor_falsepositive = 0
outdoor_negative = 0
outdoor_falsenegative = 0
```

We then define the variables we need in order to complete the evaluation and assess the performance of our program. Please refer to the appendix for a comprehensive list of all the variables used.

```
#evaluation contains pairs of values (real variable name and predicted variable
name)
for line in evaluation:
    if 'indoor' in line[0]:
        n_indoor += 1
    else:
        n_notindoor += 1

    if 'outdoor' in line[0]:
        n_outdoor += 1
    else:
        n_notoutdoor += 1
```

```
    if 'indoor' in line[1]:
        if 'indoor' in line[0]:
            indoor_positive +=1
            indoor_and_is_indoor +=1
        else:
            indoor_falsepositive +=1
            if 'outdoor' in line[0]:
                indoor_and_is_outdoor += 1
            if 'return' in line[0]:`
                indoor_and_is_return += 1
            if 'supply' in line[0]:
                indoor_and_is_supply += 1
            if 'service' in line[0]:
                indoor_and_is_service += 1
    else:
        if 'indoor' in line[0]:
            indoor_falsenegative +=1
        else:
            indoor_negative +=1
```

Then we score the variables we have defined based on the results in our 'january_evaluation.npy' file. This step allows us to complete our evaluation and proceed to the visualisation phase of our evaluation.

8. Testing of algorithm

We can now test if our final script functions and also have a look at the results.

```
print('\nindoor', n_indoor, n_notindoor, indoor_positive, indoor_negative,
indoor_falsepositive, indoor_falsenegative)
```

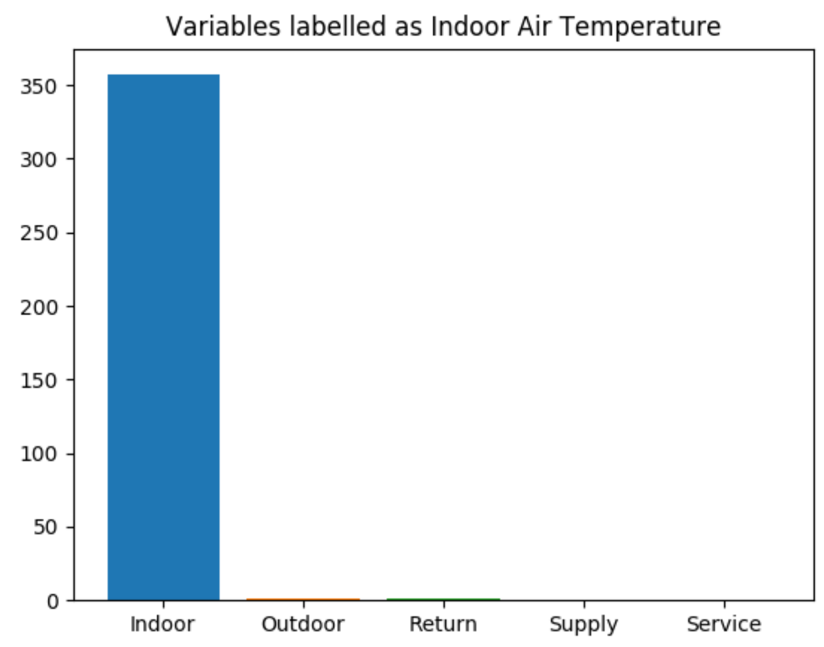
This helps us look at the raw scores of the variables we've created (which are based on the confusion matrix).

```
print('\nSensitivity')
print('indoor', indoor_positive/(indoor_positive+indoor_falsenegative))
```

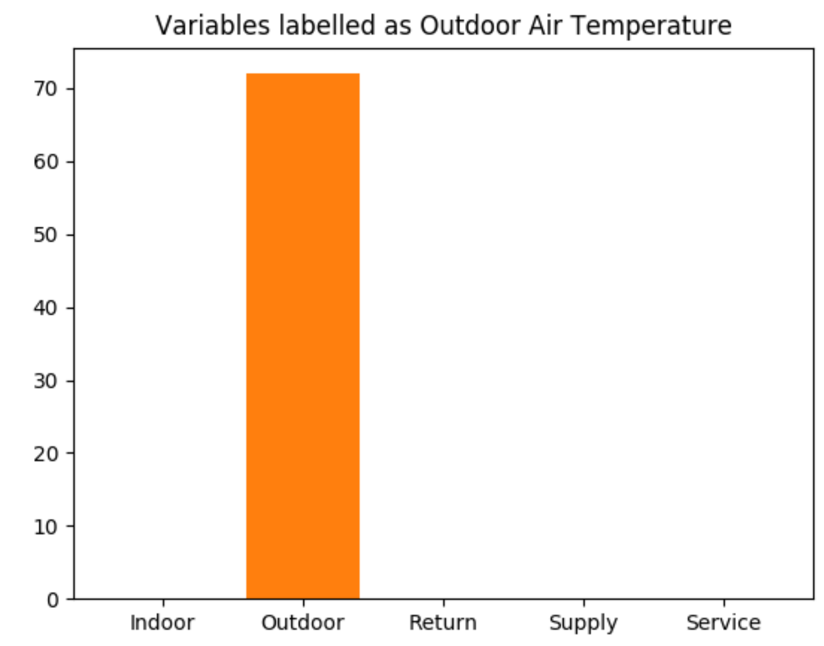
```
print('\nSpecificity')
print('indoor', indoor_negative/(indoor_negative+indoor_falsepositive))
```

```
print('\nAccuracy')
print('indoor',
(indoor_negative+indoor_positive)/(indoor_negative+indoor_falsenegative+indoor_posi
tive+indoor_falsepositive))
```

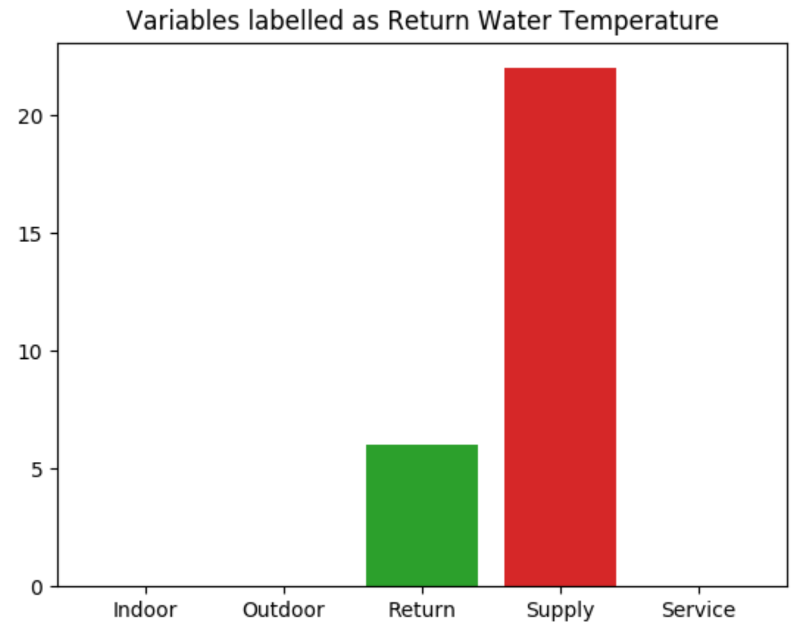
The above excerpts from the scripts shows how we've prepared the statistical performance measures for our analysis. We've arbitrarily chosen three measures (sensitivity, specificity and accuracy) but we can just as easily write a few lines of code to evaluate other statistical performance measures based on our analysis.



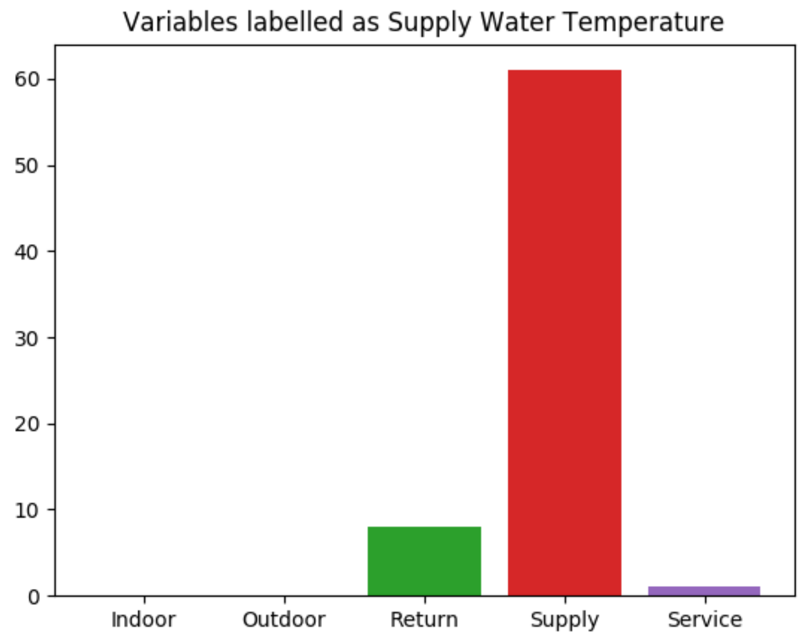
[Fig. 8] Graph of labelled Indoor Air Temperature Values



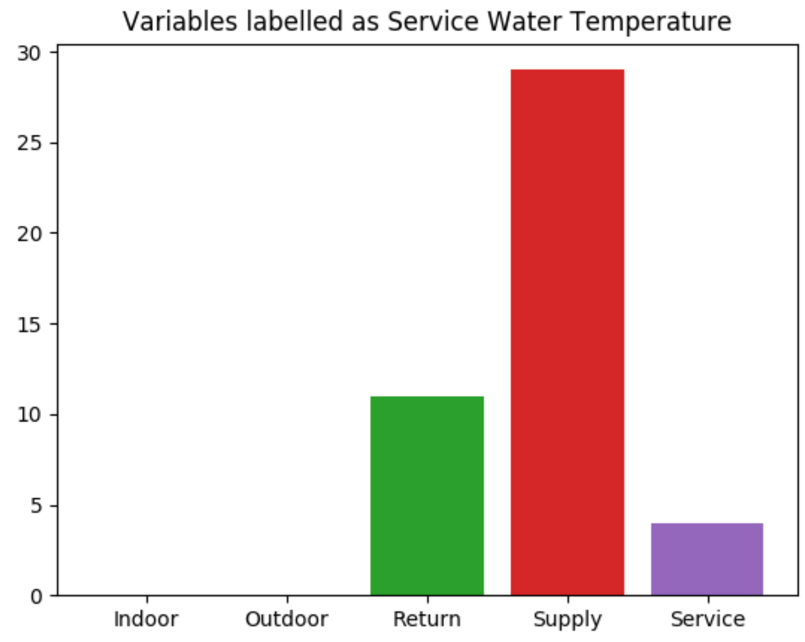
[Fig. 8] Graph of labelled Outdoor Air Temperature Values



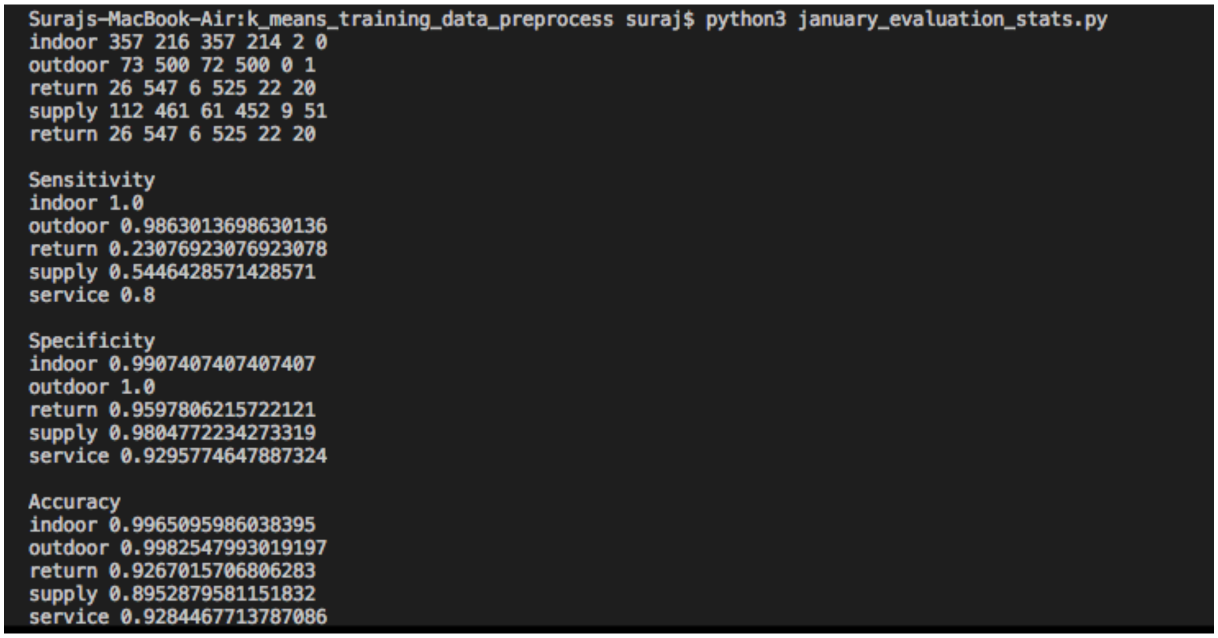
[Fig. 9] Graph of labelled Return Water Temperature Values



[Fig. 10] Graph of labelled Supply Water Temperature Values



[Fig. 10] Graph of labelled Service Water Temperature Values



```
Surajs-MacBook-Air:k_means_training_data_preprocess suraj$ python3 january_evaluation_stats.py
indoor 357 216 357 214 2 0
outdoor 73 500 72 500 0 1
return 26 547 6 525 22 20
supply 112 461 61 452 9 51
return 26 547 6 525 22 20

Sensitivity
indoor 1.0
outdoor 0.9863013698630136
return 0.23076923076923078
supply 0.5446428571428571
service 0.8

Specificity
indoor 0.9907407407407407
outdoor 1.0
return 0.9597806215722121
supply 0.9804772234273319
service 0.9295774647887324

Accuracy
indoor 0.9965095986038395
outdoor 0.9982547993019197
return 0.9267015706806283
supply 0.8952879581151832
service 0.9284467713787086
```

[Fig. 11] Values of Prepared Variables (based on Confusion Matrix) and Values of Statistical Performance Measures

9. Conclusion

9.1 Summary and evaluation

After completion of all discussed sections of this thesis, conduct a research of the state of the art and understand popular practices, design a program within the context of the requirements of my industry partner – Energocentrum and test if the program works as expected, we can conclude several key points regarding the topic.

The feature-based approach is the best approach possible as it can be adapted for various types of requirements. We have designed an algorithm based on the approach within the scope of the bachelor thesis, tested it appropriately and shown that it functions for our particular use case. The algorithm performs reasonably well in predicting the data labels, however it would be much better to train it with a larger data set (such as the temperature readings for the whole year).

Moving forward, this design has to be scaled appropriately in order to be used for a smart system. If we would like to label data points that are not temperature values.
we need to also look into the possibilities of reducing dimensions during the filtering process (for example with Principal Component Analysis). Furthermore, we also need to appropriately redesign our performance measures (if not, use new ones) if the aim is to label other data types in the smart system.

10. References

[1] C. M. Judd, G. H. McClelland, and C. S. Ryan, *Data Analysis: A Model Comparison Approach, Second Edition*. Routledge, 2011.

[2] T. Warren Liao, ‘Clustering of time series data—a survey’, *Pattern Recognit.*, vol. 38, no. 11, pp. 1857–1874, Nov. 2005.

[3] K. Košmelj and V. Batagelj, ‘Cross-sectional approach for clustering time varying data’, *J. Classif.*, vol. 7, no. 1, pp. 99–109, Mar. 1990.

[4] V. BATAGELJ, ‘Generalized Ward and Related Clustering Problems ∗’, p. 8.

[5] T. Warren Liao, ‘A clustering procedure for exploratory mining of vector time series’, *Pattern Recognit.*, vol. 40, no. 9, pp. 2550–2562, Sep. 2007.

[6] *MacQueen, J. B. (1967), Some Methods for Classification and Analysis of MultiVariate Observations, in L. M. Le Cam & J. Neyman, ed., ‘Proc. of the fifth Berkeley Symposium on Mathematical Statistics and Probability’ , University of California Press, , pp. 281-297.* .

[7] J. C. Bezdek, R. Ehrlich, and W. Full, ‘FCM: The fuzzy c-means clustering algorithm’, *Comput. Geosci.*, vol. 10, no. 2–3, pp. 191–203, Jan. 1984.

[8] C. Goutte, P. Toft, E. Rostrup, F. Å. Nielsen, and L. K. Hansen, ‘On Clustering fMRI Time Series’, *NeuroImage*, vol. 9, no. 3, pp. 298–310, Mar. 1999.

[9] C. Goutte, L. K. Hansen, M. G. Liptrot, and E. Rostrup, ‘Feature-space clustering for fMRI meta-analysis’, *Hum. Brain Mapp.*, vol. 13, no. 3, pp. 165–183, Jul. 2001.

[10] M. Vlachos, J. Lin, E. Keogh, and D. Gunopulos, ‘A Wavelet-Based Anytime Algorithm for K-Means Clustering of Time Series’, p. 12.

[11] M. Ramoni, ‘Bayesian Clustering by Dynamics’, p. 31.

[12] M. Ramoni, P. Sebastiani, and P. Cohen, ‘Multivariate Clustering by Dynamics’, p. 6.

11. Appendix

'january_process_data.py'

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import norm as sn
from scipy.stats import skew, kurtosis
filename = './data/tagged_data_01_2017.csv' # csv file to load
data = pd.read_csv(filename, delimiter=';', keep_default_na=False,
na_values=['NaN'], dtype='unicode') # load data to pandas DataFrame

data.set_index('time', inplace = True) # set column 'time' as index

data.index = pd.to_datetime(data.index, format='%d/%m/%y %H:%M') # convert index to
datetime objects
data = data.apply(pd.to_numeric) # convert data id DataFrame to numbers
columns = list(data.columns.values) # save DataFrame header to list
# print(data[columns[0]]) # print timeserie stored in DataFrame data indexed by
list of columns names (header)
#data[columns[0][:]].plot() # plot data with indexing by list of columns names
(header)

data = data[data.index.dayofweek < 5] # filtering weekday
data = data.filter(regex=("^.*temperature.*$")) #filtering the columns for string -
"temperature"

data_indoor = data.filter(regex=("indoor_air_temperature.*"))
data_outdoor = data.filter(regex=("outdoor_air_temperature.*"))
data_return = data.filter(regex=("supply_water_temperature.*"))
data_supply = data.filter(regex=("return_water_temperature.*"))
data_service = data.filter(regex=("warm_service_water_temperature.*"))


indoordaymeans = data_indoor.groupby(data_indoor.index.day).agg(lambda x:
np.nanmean(x[(x>15) & (x<40)]))
indoordaystds = data_indoor.groupby(data_indoor.index.day).agg(lambda x:
np.nanstd(x[(x>15) & (x<40)]))
indoormonthskew = data_indoor.groupby(data_indoor.index.month).agg(lambda x:
skew(x[(x>15) & (x<40)]))
indoormonthkurt= data_indoor.groupby(data_indoor.index.month).agg(lambda x:
kurtosis(x[(x>15) & (x<40)]))
np_in_temp_means_nan = np.reshape(indoordaymeans.as_matrix(), -1)
np_in_temp_stds_nan = np.reshape(indoordaystds.as_matrix(), -1)
```

```
np_in_temp_skew_nan = np.reshape(indoormonthskew.as_matrix(), -1)
np_in_temp_kurt_nan = np.reshape(indoormonthkurt.as_matrix(), -1)
np_in_temp_means = np_in_temp_means_nan[~np.isnan(np_in_temp_means_nan)]
np_in_temp_stds = np_in_temp_stds_nan[~np.isnan(np_in_temp_stds_nan)]
np_in_temp_skew = np_in_temp_skew_nan[~np.isnan(np_in_temp_skew_nan)]
np_in_temp_kurt = np_in_temp_skew_nan[~np.isnan(np_in_temp_kurt_nan)]




outdoordaymeans = data_outdoor.groupby(data_outdoor.index.day).agg(lambda x:
np.nanmean(x[x<15]))
outdoordaystds = data_outdoor.groupby(data_outdoor.index.day).agg(lambda x:
np.nanstd(x[x<15]))
outdoormonthskew= data_outdoor.groupby(data_outdoor.index.month).agg(lambda x:
skew(x[x<15]))
outdoormonthkurt= data_outdoor.groupby(data_outdoor.index.month).agg(lambda x:
kurtosis(x[x<15]))
np_out_temp_means_nan = np.reshape(outdoordaymeans.as_matrix(), -1)
np_out_temp_stds_nan = np.reshape(outdoordaystds.as_matrix(), -1)
np_out_temp_skew_nan = np.reshape(outdoormonthskew.as_matrix(), -1)
np_out_temp_kurt_nan = np.reshape(outdoormonthkurt.as_matrix(), -1)
np_out_temp_means = np_out_temp_means_nan[~np.isnan(np_out_temp_means_nan)]
np_out_temp_stds = np_out_temp_stds_nan[~np.isnan(np_out_temp_stds_nan)]
np_out_temp_skew = np_out_temp_skew_nan[~np.isnan(np_out_temp_skew_nan)]
np_out_temp_kurt = np_out_temp_kurt_nan[~np.isnan(np_out_temp_kurt_nan)]

supplydaymeans = data_supply.groupby(data_supply.index.day).agg(lambda x:
np.nanmean(x[(x>30) & (x<100)]))
supplydaystds = data_supply.groupby(data_supply.index.day).agg(lambda x:
np.nanstd(x[(x>30) & (x<100)]))
supplymonthskew= data_supply.groupby(data_supply.index.month).agg(lambda x:
skew(x[(x>30) & (x<100)]))
supplymonthkurt= data_supply.groupby(data_supply.index.month).agg(lambda x:
kurtosis(x[(x>30) & (x<100)]))
np_supply_temp_means_nan = np.reshape(supplydaymeans.as_matrix(), -1)
np_supply_temp_stds_nan = np.reshape(supplydaystds.as_matrix(), -1)
np_supply_temp_skew_nan = np.reshape(supplymonthskew.as_matrix(), -1)
np_supply_temp_kurt_nan = np.reshape(supplymonthkurt.as_matrix(), -1)
np_supply_temp_means =
np_supply_temp_means_nan[~np.isnan(np_supply_temp_means_nan)]
np_supply_temp_stds = np_supply_temp_stds_nan[~np.isnan(np_supply_temp_stds_nan)]
np_supply_temp_skew = np_supply_temp_skew_nan[~np.isnan(np_supply_temp_skew_nan)]
np_supply_temp_kurt = np_supply_temp_kurt_nan[~np.isnan(np_supply_temp_kurt_nan)]

returndaymeans = data_return.groupby(data_return.index.day).agg(lambda x:
np.nanmean(x[(x>30) & (x<100)]))
```

```
returndaystds = data_return.groupby(data_return.index.day).agg(lambda x:
np.nanstd(x[(x>30) & (x<100)]))
returnmonthskew= data_return.groupby(data_return.index.month).agg(lambda x:
skew(x[(x>30) & (x<100)]))
returnmonthkurt= data_return.groupby(data_return.index.month).agg(lambda x:
kurtosis(x[(x>30) & (x<100)]))
np_return_temp_means_nan = np.reshape(returndaymeans.as_matrix(), -1)
np_return_temp_stds_nan = np.reshape(returndaystds.as_matrix(), -1)
np_return_temp_skew_nan = np.reshape(returnmonthskew.as_matrix(), -1)
np_return_temp_kurt_nan = np.reshape(returnmonthkurt.as_matrix(), -1)
np_return_temp_means =
np_return_temp_means_nan[~np.isnan(np_return_temp_means_nan)]
np_return_temp_stds = np_return_temp_stds_nan[~np.isnan(np_return_temp_stds_nan)]
np_return_temp_skew = np_return_temp_skew_nan[~np.isnan(np_return_temp_skew_nan)]
np_return_temp_kurt = np_return_temp_kurt_nan[~np.isnan(np_return_temp_kurt_nan)]

warmdaymeans = data_service.groupby(data_service.index.day).agg(lambda x:
np.nanmean(x[(x>30) & (x<100)]))
warmdaystds = data_service.groupby(data_service.index.day).agg(lambda x:
np.nanstd(x[(x>30) & (x<100)]))
warmmonthskew= data_service.groupby(data_service.index.month).agg(lambda x:
skew(x[(x>30) & (x<100)]))
warmmonthkurt= data_service.groupby(data_service.index.month).agg(lambda x:
kurtosis(x[(x>30) & (x<100)]))
np_warm_temp_means_nan = np.reshape(warmdaymeans.as_matrix(), -1)
np_warm_temp_stds_nan = np.reshape(warmdaystds.as_matrix(), -1)
np_warm_temp_skew_nan = np.reshape(warmmonthskew.as_matrix(), -1)
np_warm_temp_kurt_nan = np.reshape(warmmonthkurt.as_matrix(), -1)
np_warm_temp_means = np_warm_temp_means_nan[~np.isnan(np_warm_temp_means_nan)]
np_warm_temp_stds = np_warm_temp_stds_nan[~np.isnan(np_warm_temp_stds_nan)]
np_warm_temp_skew = np_warm_temp_skew_nan[~np.isnan(np_warm_temp_skew_nan)]
np_warm_temp_kurt = np_warm_temp_kurt_nan[~np.isnan(np_warm_temp_kurt_nan)]

np.save('./evaluation/data_january',
[[np_in_temp_means,np_out_temp_means,np_supply_temp_means,np_return_temp_means,np_w
arm_temp_means],

[np_in_temp_stds,np_out_temp_stds,np_supply_temp_stds,np_return_temp_stds,np_warm_t
emp_stds],

[np_in_temp_skew,np_out_temp_skew,np_supply_temp_skew,np_return_temp_skew,np_warm_t
emp_skew],

[np_in_temp_kurt,np_out_temp_kurt,np_supply_temp_kurt,np_return_temp_kurt,np_warm_t
emp_kurt]])
```

```
daymeans =
indoordaymeans.join(outdoordaymeans.join(supplydaymeans.join(returndaymeans.join(wa
rmdaymeans))))
daystds =
indoordaystds.join(outdoordaystds.join(supplydaystds.join(returndaystds.join(warmda
ystds))))
monthskew =
indoormonthskew.join(outdoormonthskew.join(supplymonthskew.join(returnmonthskew.joi
n(warmmonthskew))))
monthkurt =
indoormonthkurt.join(outdoormonthkurt.join(supplymonthkurt.join(returnmonthkurt.joi
n(warmmonthkurt))))

daymeans.to_pickle('./evaluation/data_daymeans_january.pkl')
daystds.to_pickle('./evaluation/data_daystds_january.pkl')
monthkurt.to_pickle('./evaluation/data_monthkurt_january.pkl')
monthskew.to_pickle('./evaluation/data_monthskew_january.pkl')

plt.show()
```

‘january_plot_distributions.py’

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import norm as sn
from scipy.stats import skew, kurtosis

data = np.load('./evaluation/data_january.npy')

np_in_temp_means = data[0,0]
np_out_temp_means = data[0,1]
np_supply_temp_means = data[0,2]
np_return_temp_means = data[0,3]
np_warm_temp_means = data[0,4]
np_in_temp_stds = data[1,0]
np_out_temp_stds = data[1,1]
np_supply_temp_stds = data[1,2]
np_return_temp_stds = data[1,3]
np_warm_temp_stds = data[1,4]
np_in_temp_skew = data[2,0]
np_out_temp_skew = data[2,1]
np_supply_temp_skew = data[2,2]
np_return_temp_skew = data[2,3]
np_warm_temp_skew = data[2,4]
np_in_temp_kurt = data[3,0]
np_out_temp_kurt = data[3,1]
np_supply_temp_kurt = data[3,2]
np_return_temp_kurt = data[3,3]
np_warm_temp_kurt = data[3,4]

bins_mean = np.linspace(-30, 100, 200) #creating bins from -30 to 100 to 200
samples ---> 199 bins
bins_skew = np.linspace(-20, 20, 200)
bins_std = np.linspace(-20, 40, 200)
bins_kurt = np.linspace(-20, 40, 200)

plt.figure()
plt.gca().set_prop_cycle(None)
plt.hist(np_in_temp_means, bins=bins_mean, density=True, label='Indoor Air')
plt.hist(np_out_temp_means, bins=bins_mean, density=True, label='Outdoor Air')
plt.hist(np_supply_temp_means, bins=bins_mean, density=True, label='Supply Water')
plt.hist(np_return_temp_means, bins=bins_mean, density=True, label='Return Water')
plt.hist(np_warm_temp_means, bins=bins_mean, density=True, label='Service Water')
plt.legend()
plt.xlabel('Temparature [°C]')
plt.ylabel('Number of samples')
plt.title('Histogram of data mean values')


plt.figure()
plt.gca().set_prop_cycle(None)
plt.hist(np_in_temp_skew, bins=bins_skew, density=True, label='Indoor Air')
plt.hist(np_out_temp_skew, bins=bins_skew, density=True, label='Outdoor Air')
plt.hist(np_supply_temp_skew, bins=bins_skew, density=True, label='Supply Water')
plt.hist(np_return_temp_skew, bins=bins_skew, density=True, label='Return Water')
plt.hist(np_warm_temp_skew, bins=bins_skew, density=True, label='Service Water')
plt.legend()
plt.xlabel('Temparature [°C]')
plt.ylabel('Number of samples')
plt.title('Histogram of data skewness')

plt.figure()
plt.gca().set_prop_cycle(None)
plt.hist(np_in_temp_stds, bins=bins_std, density=True, label='Indoor Air')
plt.hist(np_out_temp_stds, bins=bins_std, density=True, label='Outdoor Air')
plt.hist(np_supply_temp_stds, bins=bins_std, density=True, label='Supply Water')
plt.hist(np_return_temp_stds, bins=bins_std, density=True, label='Return Water')
plt.hist(np_warm_temp_stds, bins=bins_std, density=True, label='Service Water')
plt.legend()
plt.xlabel('Temparature [°C]')
plt.ylabel('Number of samples')
plt.title('Histogram of data standard deviation')
plt.figure()

plt.gca().set_prop_cycle(None)
plt.hist(np_in_temp_kurt, bins=bins_kurt, density=True, label='Indoor Air')
plt.hist(np_out_temp_kurt, bins=bins_kurt, density=True, label='Outdoor Air')
plt.hist(np_supply_temp_kurt, bins=bins_kurt, density=True, label='Supply Water')
plt.hist(np_return_temp_kurt, bins=bins_kurt, density=True, label='Return Water')
plt.hist(np_warm_temp_kurt, bins=bins_kurt, density=True, label='Service Water')
plt.legend()
plt.xlabel('Temparature [°C]')
plt.ylabel('Number of samples')
plt.title('Histogram of data kurtosis')

plt.show()
```

‘january_k_means_train.py’

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import norm as sn
from scipy.stats import skew, kurtosis
from sklearn.cluster import KMeans
import pickle

data = np.load('./evaluation/data_january.npy')

np_in_temp_means = data[0,0].reshape(-1, 1) #loading data from array from file
np_out_temp_means = data[0,1].reshape(-1, 1)
np_supply_temp_means = data[0,2].reshape(-1, 1)
np_return_temp_means = data[0,3].reshape(-1, 1)
np_warm_temp_means = data[0,4].reshape(-1, 1)
np_in_temp_stds = data[1,0].reshape(-1, 1)
np_out_temp_stds = data[1,1].reshape(-1, 1)
np_supply_temp_stds = data[1,2].reshape(-1, 1)
np_return_temp_stds = data[1,3].reshape(-1, 1)
np_warm_temp_stds = data[1,4].reshape(-1, 1)
np_in_temp_skew = data[2,0].reshape(-1, 1)
np_out_temp_skew = data[2,1].reshape(-1, 1)
np_supply_temp_skew = data[2,2].reshape(-1, 1)
np_return_temp_skew = data[2,3].reshape(-1, 1)
np_warm_temp_skew = data[2,4].reshape(-1, 1)
np_in_temp_kurt = data[3,0].reshape(-1, 1)
np_out_temp_kurt = data[3,1].reshape(-1, 1)
np_supply_temp_kurt = data[3,2].reshape(-1, 1)
np_return_temp_kurt = data[3,3].reshape(-1, 1)
np_warm_temp_kurt = data[3,4].reshape(-1, 1)

CLUSTERS_NUMBER = 2 #setting no. of clusters

kmeans_in_temp_means = KMeans(n_clusters=CLUSTERS_NUMBER,
random_state=0).fit(np_in_temp_means)
#running kmeans training algorithm for 2 clusters on indoor temp values
kmeans_out_temp_means = KMeans(n_clusters=CLUSTERS_NUMBER,
random_state=0).fit(np_out_temp_means)
kmeans_supply_temp_means = KMeans(n_clusters=CLUSTERS_NUMBER,
random_state=0).fit(np_supply_temp_means)
kmeans_return_temp_means = KMeans(n_clusters=CLUSTERS_NUMBER,
random_state=0).fit(np_return_temp_means)
kmeans_warm_temp_means = KMeans(n_clusters=CLUSTERS_NUMBER,
random_state=0).fit(np_warm_temp_means)
```

```
kmeans_in_temp_stds = KMeans(n_clusters=CLUSTERS_NUMBER,
random_state=0).fit(np_in_temp_stds)
kmeans_out_temp_stds = KMeans(n_clusters=CLUSTERS_NUMBER,
random_state=0).fit(np_out_temp_stds)
kmeans_supply_temp_stds = KMeans(n_clusters=CLUSTERS_NUMBER,
random_state=0).fit(np_supply_temp_stds)
kmeans_return_temp_stds = KMeans(n_clusters=CLUSTERS_NUMBER,
random_state=0).fit(np_return_temp_stds)
kmeans_warm_temp_stds = KMeans(n_clusters=CLUSTERS_NUMBER,
random_state=0).fit(np_warm_temp_stds)

kmeans_in_temp_skew = KMeans(n_clusters=CLUSTERS_NUMBER,
random_state=0).fit(np_in_temp_skew)
kmeans_out_temp_skew = KMeans(n_clusters=CLUSTERS_NUMBER,
random_state=0).fit(np_out_temp_skew)
kmeans_supply_temp_skew = KMeans(n_clusters=CLUSTERS_NUMBER,
random_state=0).fit(np_supply_temp_skew)
kmeans_return_temp_skew = KMeans(n_clusters=CLUSTERS_NUMBER,
random_state=0).fit(np_return_temp_skew)
kmeans_warm_temp_skew = KMeans(n_clusters=CLUSTERS_NUMBER,
random_state=0).fit(np_warm_temp_skew)

kmeans_in_temp_kurt = KMeans(n_clusters=CLUSTERS_NUMBER,
random_state=0).fit(np_in_temp_kurt)
kmeans_out_temp_kurt = KMeans(n_clusters=CLUSTERS_NUMBER,
random_state=0).fit(np_out_temp_kurt)
kmeans_supply_temp_kurt = KMeans(n_clusters=CLUSTERS_NUMBER,
random_state=0).fit(np_supply_temp_kurt)
kmeans_return_temp_kurt = KMeans(n_clusters=CLUSTERS_NUMBER,
random_state=0).fit(np_return_temp_kurt)
kmeans_warm_temp_kurt = KMeans(n_clusters=CLUSTERS_NUMBER,
random_state=0).fit(np_warm_temp_kurt)

kmeans_dict = { 'kmeans_in_temp_means': kmeans_in_temp_means,
                'kmeans_out_temp_means': kmeans_out_temp_means,
                'kmeans_supply_temp_means': kmeans_supply_temp_means,
                'kmeans_return_temp_means': kmeans_return_temp_means,
                'kmeans_warm_temp_means': kmeans_warm_temp_means,
                'kmeans_in_temp_stds': kmeans_in_temp_stds,
                'kmeans_out_temp_stds': kmeans_out_temp_stds,
                'kmeans_supply_temp_stds': kmeans_supply_temp_stds,
                'kmeans_return_temp_stds': kmeans_return_temp_stds,
                'kmeans_warm_temp_stds': kmeans_warm_temp_stds,
                'kmeans_in_temp_skew': kmeans_in_temp_skew,
                'kmeans_out_temp_skew': kmeans_out_temp_skew,
                'kmeans_supply_temp_skew': kmeans_supply_temp_skew,
                'kmeans_return_temp_skew': kmeans_return_temp_skew,
                'kmeans_warm_temp_skew': kmeans_warm_temp_skew,
                'kmeans_in_temp_kurt': kmeans_in_temp_kurt,
                'kmeans_out_temp_kurt': kmeans_out_temp_kurt,
                'kmeans_supply_temp_kurt': kmeans_supply_temp_kurt,
                'kmeans_return_temp_kurt': kmeans_return_temp_kurt,
                'kmeans_warm_temp_kurt': kmeans_warm_temp_kurt
                }
#save calculated clusters to a dictionary (type of array)

print('Clusters centers\n**********************')
for kmeans_name, kmeans_value in kmeans_dict.items():
#for cycle to show clusters
    print('\n'+kmeans_name)
    print(kmeans_value.cluster_centers_)
#read cluster centers from kmeans algorithm

with open('./evaluation/kmeans_january.pkl', 'wb') as f:
#open file, wb--> write binary (way of working with file---> as binary format), f:
pointer to file
    pickle.dump(kmeans_dict, f)
# save dictionary to file f as pickle library format
```

‘january_k_means_evaluate.py’

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import norm as sn
from scipy.stats import skew, kurtosis
from sklearn.cluster import KMeans
import pickle

with open('./evaluation/kmeans_january.pkl', 'rb') as f:
    kmeans = pickle.load(f)

daymeans = pd.read_pickle('./evaluation/data_daymeans_january.pkl')
daystds = pd.read_pickle('./evaluation/data_daystds_january.pkl')
monthkurt = pd.read_pickle('./evaluation/data_monthkurt_january.pkl')
monthskew = pd.read_pickle('./evaluation/data_monthskew_january.pkl')

temp_daymeans = daymeans.filter(regex=("^.*temperature.*$"))
temp_daystds = daystds.filter(regex=("^.*temperature.*$"))
temp_monthskew = monthskew.filter(regex=("^.*temperature.*$"))
temp_monthkurt = monthkurt.filter(regex=("^.*temperature.*$"))

names= ['indoor', 'outdoor', 'return', 'supply', 'warm_service']
evaluation = []

for column in temp_daymeans:
    try:
#try and if error execute exception
        temp_daymeans_col = temp_daymeans[column].as_matrix()
#going through temp_daymeans and preapare it for running the unsupervised part of
k-means algo
        temp_daystds_col = temp_daystds[column].as_matrix()
        temp_monthkurt_col = temp_monthkurt[column].as_matrix()
        temp_monthskew_col = temp_monthskew[column].as_matrix()

        temp_daymeans_col =
temp_daymeans_col[~np.isnan(temp_daymeans_col)].reshape(-1,1)           #filtering
out nan values and reshape to 2-d array for the kmeans algorithm (required
standard)
        temp_daystds_col = temp_daystds_col[~np.isnan(temp_daystds_col)].reshape(-
1,1)
        temp_monthkurt_col =
temp_monthkurt_col[~np.isnan(temp_monthkurt_col)].reshape(-1,1)
        temp_monthskew_col =
temp_monthskew_col[~np.isnan(temp_monthskew_col)].reshape(-1,1)
```

```
    except:
#exception skip iteration
        continue

    score_means = []
#evaluation part; append 'score' values; score---->
    score_means.append(kmeans['kmeans_in_temp_means'].score(temp_daymeans_col))
# for unknow mean values, calculate the distance from the fitted kmeans of know
variable (from the training part)
    score_means.append(kmeans['kmeans_out_temp_means'].score(temp_daymeans_col))
    score_means.append(kmeans['kmeans_supply_temp_means'].score(temp_daymeans_col))
    score_means.append(kmeans['kmeans_return_temp_means'].score(temp_daymeans_col))
    score_means.append(kmeans['kmeans_warm_temp_means'].score(temp_daymeans_col))

    score_stds = []
    score_stds.append(kmeans['kmeans_in_temp_stds'].score(temp_daystds_col))
    score_stds.append(kmeans['kmeans_out_temp_stds'].score(temp_daystds_col))
    score_stds.append(kmeans['kmeans_supply_temp_stds'].score(temp_daystds_col))
    score_stds.append(kmeans['kmeans_return_temp_stds'].score(temp_daystds_col))
    score_stds.append(kmeans['kmeans_warm_temp_stds'].score(temp_daystds_col))

    score_kurt = []
    score_kurt.append(kmeans['kmeans_in_temp_kurt'].score(temp_monthkurt_col))
    score_kurt.append(kmeans['kmeans_out_temp_kurt'].score(temp_monthkurt_col))
    score_kurt.append(kmeans['kmeans_supply_temp_kurt'].score(temp_monthkurt_col))
    score_kurt.append(kmeans['kmeans_return_temp_kurt'].score(temp_monthkurt_col))
    score_kurt.append(kmeans['kmeans_warm_temp_kurt'].score(temp_monthkurt_col))

    score_skew = []
    score_skew.append(kmeans['kmeans_in_temp_skew'].score(temp_monthskew_col))
    score_skew.append(kmeans['kmeans_out_temp_skew'].score(temp_monthskew_col))
    score_skew.append(kmeans['kmeans_supply_temp_skew'].score(temp_monthskew_col))
    score_skew.append(kmeans['kmeans_return_temp_skew'].score(temp_monthskew_col))
    score_skew.append(kmeans['kmeans_warm_temp_skew'].score(temp_monthskew_col))

    scores = np.array(score_means) + np.array(score_stds) + np.array(score_skew) +
np.array(score_kurt)  #finally, score= sum of all scores from of each feature for
the data point

    evaluation.append([column, names[np.argmax(scores)]])
#returns name of the vairable with high score value (closest to 0 is "winner")

np.save('./evaluation/january_evaluation', evaluation)
#saving to file
```

'january_evaluation_stats.py'

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import norm as sn
from scipy.stats import skew, kurtosis

evaluation = np.load('./evaluation/january_evaluation.npy')

#preparing of variables for statistic evaluation (based on confusion matrix)

n_indoor = 0
n_notindoor = 0
n_outdoor = 0
n_notoutdoor = 0
n_return = 0
n_notreturn = 0
n_supply = 0
n_notsupply = 0
n_service = 0
n_notservice = 0

indoor_positive = 0
indoor_falsepositive = 0
indoor_negative = 0
indoor_falsenegative = 0

outdoor_positive = 0
outdoor_falsepositive = 0
outdoor_negative = 0
outdoor_falsenegative = 0

return_positive = 0
return_falsepositive = 0
return_negative = 0
return_falsenegative = 0

supply_positive = 0
supply_falsepositive = 0
supply_negative = 0
supply_falsenegative = 0

service_positive = 0
service_falsepositive = 0
service_negative = 0
```

```python
service_falsenegative = 0

indoor_and_is_indoor = 0
indoor_and_is_outdoor = 0
indoor_and_is_return = 0
indoor_and_is_supply = 0
indoor_and_is_service = 0

outdoor_and_is_indoor = 0
outdoor_and_is_outdoor = 0
outdoor_and_is_return = 0
outdoor_and_is_supply = 0
outdoor_and_is_service = 0

return_and_is_indoor = 0
return_and_is_outdoor = 0
return_and_is_return = 0
return_and_is_supply = 0
return_and_is_service = 0

supply_and_is_indoor = 0
supply_and_is_outdoor = 0
supply_and_is_return = 0
supply_and_is_supply = 0
supply_and_is_service = 0

service_and_is_indoor = 0
service_and_is_outdoor = 0
service_and_is_return = 0
service_and_is_supply = 0
service_and_is_service = 0

#evaluation contains pairs of values (real variable name and predicted variable
name)
for line in evaluation:
    if 'indoor' in line[0]:
        n_indoor += 1
    else:
        n_notindoor += 1

    if 'outdoor' in line[0]:
        n_outdoor += 1
    else:
        n_notoutdoor += 1

    if 'return' in line[0]:
        n_return += 1
```

```
else:
    n_notreturn += 1

if 'supply' in line[0]:
    n_supply += 1
else:
    n_notsupply += 1

if 'service' in line[0]:
    n_service += 1
else:
    n_notservice += 1

if 'indoor' in line[1]:
    if 'indoor' in line[0]:
        indoor_positive +=1
        indoor_and_is_indoor +=1
    else:
        indoor_falsepositive +=1
        if 'outdoor' in line[0]:
            indoor_and_is_outdoor += 1
        if 'return' in line[0]:
            indoor_and_is_return += 1
        if 'supply' in line[0]:
            indoor_and_is_supply += 1
        if 'service' in line[0]:
            indoor_and_is_service += 1
else:
    if 'indoor' in line[0]:
        indoor_falsenegative +=1
    else:
        indoor_negative +=1

if 'outdoor' in line[1]:
    if 'outdoor' in line[0]:
        outdoor_positive +=1
        outdoor_and_is_outdoor +=1
    else:
        outdoor_falsepositive +=1
        if 'indoor' in line[0]:
            outdoor_and_is_indoor += 1
        if 'return' in line[0]:
            outdoor_and_is_return += 1
        if 'supply' in line[0]:
            outdoor_and_is_supply += 1
        if 'service' in line[0]:
            outdoor_and_is_service += 1
```

```
else:
    if 'outdoor' in line[0]:
        outdoor_falsenegative +=1
    else:
        outdoor_negative +=1

if 'return' in line[1]:
    if 'return' in line[0]:
        return_positive +=1
        return_and_is_return +=1
    else:
        return_falsepositive +=1
        if 'indoor' in line[0]:
            return_and_is_indoor += 1
        if 'outdoor' in line[0]:
            return_and_is_outdoor += 1
        if 'supply' in line[0]:
            return_and_is_supply += 1
        if 'service' in line[0]:
            return_and_is_service += 1
else:
    if 'return' in line[0]:
        return_falsenegative +=1
    else:
        return_negative +=1

if 'supply' in line[1]:
    if 'supply' in line[0]:
        supply_positive +=1
        supply_and_is_supply +=1
    else:
        supply_falsepositive +=1
        if 'indoor' in line[0]:
            supply_and_is_indoor += 1
        if 'outdoor' in line[0]:
            supply_and_is_outdoor += 1
        if 'return' in line[0]:
            supply_and_is_return += 1
        if 'service' in line[0]:
            supply_and_is_service += 1
else:
    if 'supply' in line[0]:
        supply_falsenegative +=1
    else:
        supply_negative +=1

if 'service' in line[1]:
        if 'service' in line[0]:
            service_positive +=1
            service_and_is_service +=1
        else:
            service_falsepositive +=1
            if 'indoor' in line[0]:
                service_and_is_indoor += 1
            if 'outdoor' in line[0]:
                service_and_is_outdoor += 1
            if 'return' in line[0]:
                service_and_is_return += 1
            if 'supply' in line[0]:
                service_and_is_supply += 1
    else:
        if 'service' in line[0]:
            service_falsenegative +=1
        else:
            service_negative +=1

print('\nindoor', n_indoor, n_notindoor, indoor_positive, indoor_negative,
indoor_falsepositive, indoor_falsenegative)
print('outdoor', n_outdoor, n_notoutdoor, outdoor_positive, outdoor_negative,
outdoor_falsepositive, outdoor_falsenegative)
print('return', n_return, n_notreturn, return_positive, return_negative,
return_falsepositive, return_falsenegative)
print('supply', n_supply, n_notsupply, supply_positive, supply_negative,
supply_falsepositive, supply_falsenegative)
print('return', n_return, n_notreturn, return_positive, return_negative,
return_falsepositive, return_falsenegative)

print('\nSensitivity')
print('indoor', indoor_positive/(indoor_positive+indoor_falsenegative))
print('outdoor', outdoor_positive/(outdoor_positive+outdoor_falsenegative))
print('return', return_positive/(return_positive+return_falsenegative))
print('supply', supply_positive/(supply_positive+supply_falsenegative))
print('service', service_positive/(service_positive+service_falsenegative))

print('\nSpecificity')
print('indoor', indoor_negative/(indoor_negative+indoor_falsepositive))
print('outdoor', outdoor_negative/(outdoor_negative+outdoor_falsepositive))
print('return', return_negative/(return_negative+return_falsepositive))
print('supply', supply_negative/(supply_negative+supply_falsepositive))
print('service', service_negative/(service_negative+service_falsepositive))

print('\nAccuracy')
print('indoor',
(indoor_negative+indoor_positive)/(indoor_negative+indoor_falsenegative+indoor_posi
tive+indoor_falsepositive))
print('outdoor',
(outdoor_negative+outdoor_positive)/(outdoor_negative+outdoor_falsenegative+outdoor
_positive+outdoor_falsepositive))
print('return',
(return_negative+return_positive)/(return_negative+return_falsenegative+return_posi
tive+return_falsepositive))
print('supply',
(supply_negative+supply_positive)/(supply_negative+supply_falsenegative+supply_posi
tive+supply_falsepositive))
print('service',
(service_negative+service_positive)/(service_negative+service_falsenegative+service
_positive+service_falsepositive))

names= ['Indoor', 'Outdoor', 'Return', 'Supply', 'Service']

plt.figure()
plt.title('Variables labelled as Indoor Air Temperature')
plt.bar(1, indoor_and_is_indoor)
plt.bar(2, indoor_and_is_outdoor)
plt.bar(3, indoor_and_is_return)
plt.bar(4, indoor_and_is_supply)
plt.bar(5, indoor_and_is_service)
plt.xticks([1, 2, 3, 4, 5], names)

plt.figure()
plt.title('Variables labelled as Outdoor Air Temperature')
plt.bar(1, outdoor_and_is_indoor)
plt.bar(2, outdoor_and_is_outdoor)
plt.bar(3, outdoor_and_is_return)
plt.bar(4, outdoor_and_is_supply)
plt.bar(5, outdoor_and_is_service)
plt.xticks([1, 2, 3, 4, 5], names)

plt.figure()
plt.title('Variables labelled as Return Water Temperature')
plt.bar(1, return_and_is_indoor)
plt.bar(2, return_and_is_outdoor)
plt.bar(3, return_and_is_return)
plt.bar(4, return_and_is_supply)
plt.bar(5, return_and_is_service)
plt.xticks([1, 2, 3, 4, 5], names)

plt.figure()
plt.title('Variables labelled as Supply Water Temperature')
plt.bar(1, supply_and_is_indoor)
plt.bar(2, supply_and_is_outdoor)
plt.bar(3, supply_and_is_return)
plt.bar(4, supply_and_is_supply)
plt.bar(5, supply_and_is_service)
plt.xticks([1, 2, 3, 4, 5], names)

plt.figure()
plt.title('Variables labelled as Service Water Temperature')
plt.bar(1, service_and_is_indoor)
plt.bar(2, service_and_is_outdoor)
plt.bar(3, service_and_is_return)
plt.bar(4, service_and_is_supply)
plt.bar(5, service_and_is_service)
plt.xticks([1, 2, 3, 4, 5], names)

plt.show()
```