DHBW
Duale Hochschule
Baden-Württemberg

Measuring crowd mood at parties

Semester Paper

from the Course of Studies Allgemeine Informatik (Computer Science)

at the Cooperative State University Baden-Württemberg Heidenheim

by

Benedikt Holland

19.09.2022

Time of Project	01.01.2022 - 31.03.2022, 01.07.2022 - 19.09.2022
Student ID, Course	8778697, TINF2019AI
Reviewer	Prof. Dr. Andreas Mahr

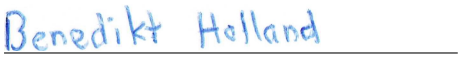
Author's declaration

Hereby I solemnly declare:

1. that this Semester Paper , titled *Measuring crowd mood at parties* is entirely the product of my own scholarly work, unless otherwise indicated in the text or references, or acknowledged below;
2. I have indicated the thoughts adopted directly or indirectly from other sources at the appropriate places within the document;
3. this Semester Paper has not been submitted either in whole or part, for a degree at this or any other university or institution;
4. I have not published this Semester Paper in the past;
5. the printed version is equivalent to the submitted electronic one.

I am aware that a dishonest declaration will entail legal consequences.

Heidenheim, 19.09.2022



Benedikt Holland

Abstract

In this thesis sensor data collected from parties is used to engineer a measure that quantifies crowd mood. An infrared camera combined with supervised learning is used to determine the crowd size. Air properties like temperature, humidity and the concentration of various chemicals are used to determine the physical activity of the crowd and infer its general mood based on that. Biases that affect the data like the ventilation system, fog machine and cigarette smoke have to be addressed.

Contents

Acronyms

VOC	Volatile Organic Compounds
TVOC	Total Volatile Organic Compounds
MAE	Mean Absolute Error
MSE	Mean Squared Error
SDA	Serial Data
SCL	Serial Clock
GPCLK	General Purpose Clock

List of Figures

Listings

1 Artificial intelligence in the music industry

As artificial intelligence is conquering the world by storm, more and more application areas are opening up. Especially online content-based platforms like YouTube and Netflix profit considerably from this development. The infamous recommendation algorithms are taking more control over our lives every day. As the largest streaming service in the world even Spotify introduced its own recommendation algorithm in 2015 called *Spotify's Discover Weekly*. This algorithm is based on collaborative filtering which works by recommending music that other users with a similar profile listen to. Other recommendation algorithms have also been proposed like knowledge-based, asking the user directly for their preferences, content-based, looking at content information and labels of tracks the user liked, and hybrid systems based on the above methods [Wan20, p. 2].

Although effective at optimizing the single user experience, the mentioned algorithms struggle to address multiple users or even crowds. Music after all is not only a personal entertainment method, but also particular effective at mass entertainment like in the form of a concert or party. Whether at a private event or in a club, the success of a party always depends on the mood of the guests. Since the financial success of the celebration often depends on this mood, everything is done to maximize it. Offers, themes and especially a very good and therefore expensive DJ. Although the DJ brings more skills than simply selecting songs, the selection alone makes a lot of difference [SDP12, p. 2]. Given this expensive investment, the question of whether this activity can be optimized or even automated falls. Companies like Spotify have now developed very sophisticated algorithms to tailor auto play to individual users, although not to an entire crowd like found at a party.

This study will aim to answer the question if it is possible to engineer a measure of crowd mood based on sensor data from a single party location.

2 Literature

Although crowd behavior analysis has been the subject of many studies over the years ([Zei+09][HS18]), studies specifically addressing crowd mood are sparse. Crowd behavior analysis mainly focuses on crowd movement and reaction to address safety concerns, while this study is more interested in subjective crowd mood. The most prominent studies on crowd mood are [Zha+17], which used computer vision techniques to determine crowd mood based on crowd movement and [Wak+15], which used Twitter posts to determine crowd mood in a suburban area in Japan. To the author's knowledge this is the first study to use crowd mood detection on parties to particulary enhance music selection. In the following sections the techniques and measures used for this study will be defined.

2.1 Supervised learning

Supervised learning is a subcategory of machine learning that deals with labeled data. It can be used to make predictions on the labels of new data based on training data. In classification algorithms a label is predicted by determining the likelihood that the label applies to any datapoint between 0 and 1 while regression algorithms predict values on a continuous or discrete scale [Loh11]. This thesis will only make use of regression supervised learning. First the metrics to calculate error and performance are defined before introducing 4 algorithms for supervised learning.

2.1.1 Error metrics

To evaluate any supervised learning model, metrics for performance and errors have to be defined. The simplest error metric is the Mean Absolute Error (MAE), which is the average absolute difference between every prediction of the model and the actual value. The absolute difference avoids errors of opposite sign canceling each other out. Similarly, the Mean Squared Error (MSE) is defined as the average difference between every prediction of the model and the actual value squared. By squaring the difference instead of taking the absolute value, outliers are punished more in addition to negative errors being avoided.

$$\text{MAE} = \frac{\sum_{i=1}^{n} |y_i - \hat{y}_i|}{n}, \quad \text{MSE} = \frac{\sum_{i=1}^{n} (y_i - \hat{y}_i)^2}{n} \tag{2.1}$$

n is the size of the dataset, y_i is the true value for the i-th input and $\hat{y}_i$ is the predicted value of the model for the i-th input [NA20].

2.1.2 R^2 score

The r^2 score or coefficient of determination of a model represents "the proportion of variance [...] that has been explained by the independent variables in the model" [scic] and is a measure of how well a model is able to generalize. It is defined as

$$R^2(y, \hat{y}) = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \overline{y})^2} \tag{2.2}$$

with $\overline{y}$ being the average of all true values. The r^2 score is defined as 0 for a constant model that predicts $\overline{y}$ for all inputs and can be at most 1 [NA20].

2.1.3 Decision trees

Decision trees are machine-learning algorithms that use recursive partitioning to fit simple prediction models based on comparisons on the smaller partitions . In every step a line is fitted through the data that splits the data with the lowest squared error. This line represents the rule for this node, in every child node this process is repeated for the reduced dataset until a termination criterion is met. These criteria can be either that the maximum tree depth is reached, the leaf node has less than the minimum number of samples required to split the node, or the leaf node has reached the minimum number of samples required to be in each node. In figure 2.1 two decision trees were fitted to a sine curve with noise. The first tree had a maximum depth of 2, meaning the data was only split into 4 regions, while the second tree had a maximum depth of 5. Each vertical line represents the split in the dataset and each horizontal line is the predicted value for this partition. For example, input values between 2 and 3 are predicted to result in a target of 0.5. Some of the advantages and disadvantages of decision trees include [scib][Ped+11]:

Advantages	Disadvantages
White box approach is easy to understand	Prone to overfitting
Can be visualized	Might not generalize well
Works well on raw data	Unstable towards small variations
Has a low complexity of $O(logn)$	Predictions are not continuous

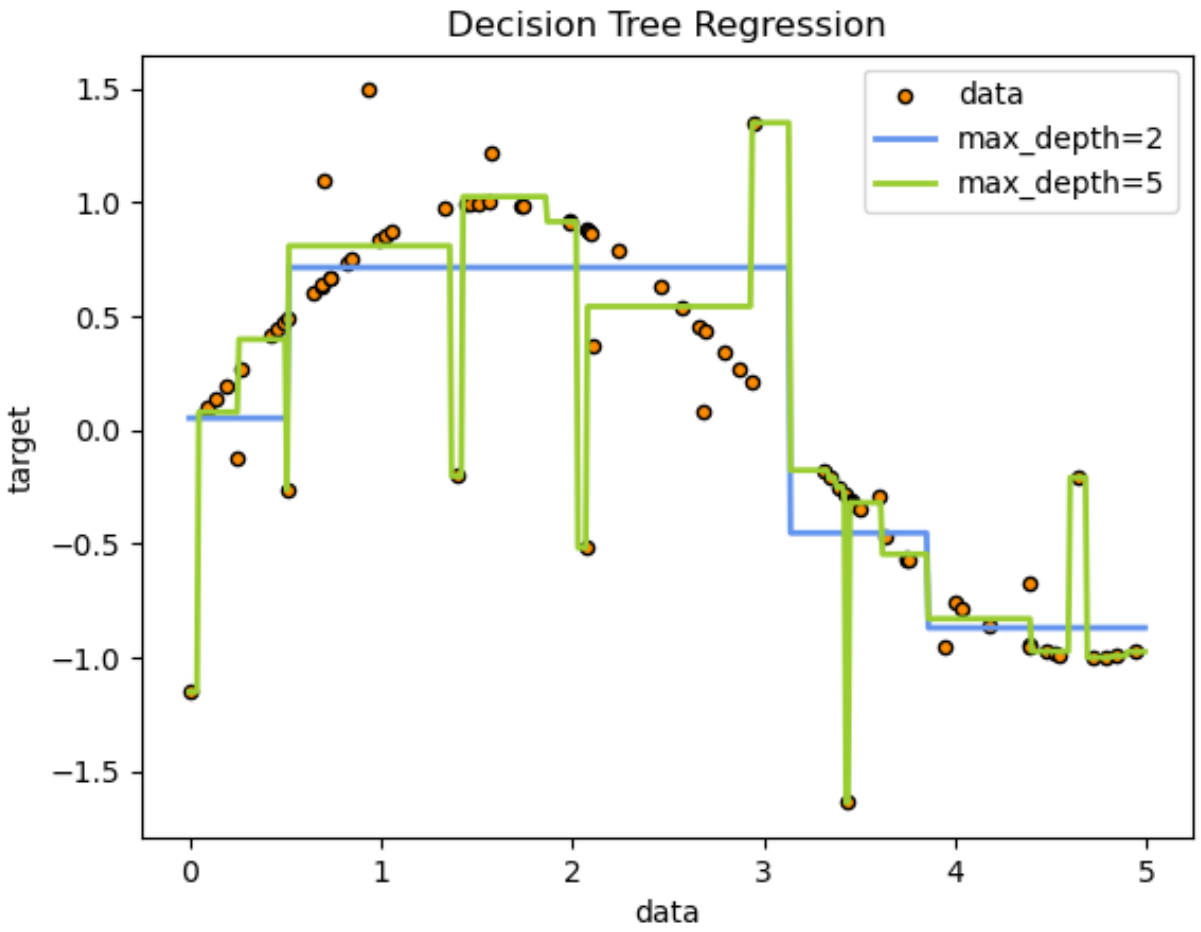


Figure 2.1: Fitting a decision tree to a sine curve with noise [scib]

2.1.4 Random forest

Random forest is an algorithm that uses many decision trees at once to try to tackle some of the disadvantages of the decision tree algorithm. The training set is bootstrapped, which means that the samples are randomly drawn with replacement, for each tree in the forest. When the forest is tasked with predicting a value, all trees vote on the result and the result with the majority is chosen as the prediction. The random forest is less prone to overfitting than single decision trees, however they are more difficult to understand and visualize than decision trees [Bre01].

2.1.5 AdaBoost

AdaBoost works similarly to random forests by using many weak learners, which may be small decision trees, to vote on predictions. However, AdaBoost relies on boosting instead of bootstrapping. In each iteration a decision tree is fitted to the whole dataset and the mean squared error for the tree is calculated. The tree is assigned a weight according

to this mean squared error and this weight is applied to the dataset. Every datapoint that got labeled correctly gets its weight decreased while every datapoint that got labeled incorrectly gets its weight increased. The subsequent trees are therefore encouraged to focus more on the mislabeled data and less on the correctly labeled data, leading the algorithm to correct its own errors with every iteration. To predict values all trees cast a vote, however the votes are not unitary but weighted according to the mean squared error of each tree. This means that high error trees contribute less to the overall prediction and low error trees more [scia][FS97]. AdaBoost is less prone to overfitting but has the disadvantage that noisy data and outliers negatively affect it.

2.1.6 Linear regression

Linear regression models use a linear function to predict values and respresents one of the simplest way of regression.

$$\phi(x_1, x_2, \ldots, x_K) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_K x_K \tag{2.3}$$

The function can have any number of inputs or dimension K and will predict a single output ϕ. The weights β will be adjusted in a way that minimizes the mean squared error of the predicted value and the actual value [SL12, p. 4][HWZ16]. This model is very simple and powerful if the underlying relationship of the variables is truly linear. However it is badly affected by outliers.

3 Experiment

The experiment will take place in an insulated closed rooms of dimensions 5 by 3 meters as shown in figure 3.1. The windows cannot be opened and there is only one door that leads to an adjacent room. Besides the dancefloor that is open for the guests, there is also a bar in the room that is operated by a bartender. Because of the small size and the limited openings of the room the temperature and air quality usually shift drastically over the course of a party. To avoid the room getting heated up too much a ventilation system has been installed that can be manually turned on by the bartender. There is also a fog machine installed that can be manually turned on. Smoking inside is discouraged but not strictly prohibited. Regular private parties take place in this room every 2 to 4 weeks.

In this chapter the assumptions made for data collection will be stated. Based on that

Figure 3.1: View into the room of dimensions 5 by 3 meters from the bar

the sensors will be selected. Finally, the hardware setup, wiring and algorithm for data collection will be described.

3.1 Assumptions

This chapter will state all assumptions made in setting up the experiment and engineering the crowd mood measure.

3.1.1 Physical activity

The first assumption of this thesis is that better mood will result in increased physical activity, like dancing, shouting, and singing. Of course, this assumption does not hold true for every individual, but in crowds of people it tends to apply. It is important to note that this causality does not necessarily apply in reverse. Meaning that a high physical activity crowd does not have to be in a good mood. For example, if a fight or panic breaks out in the crowd, the physical activity will be very high, but the crowd mood will be negative instead of positive. The physical activity measure therefore operates on a spectrum of crowd moods which may look like this.

- Low activity - bored
- Medium activity - happy
- High activity - energetic
- Very high activity - aggressive

Depending on the type of party, we want the crowd to have a different mood. The only mood all types want to avoid is boredom, every party wants their guests to be at least happy. If the party is aimed at making sales an energetic mood is preferred, because guest will be more engaged in dancing and singing and will therefore drink more to replace lost water in their body. As mentioned in general aggressive crowd mood has to be avoided, however there are exceptions. The best example for this are so-called “mosh pits” often encountered on heavy metal concerts. Here the individuals will push each other in an aggressive manner, however the goal is not to harm each other but to let off steam. In our case actual fights are rare and aggressive mood is most often caused by these mosh pits, we will therefore assume that higher physical activity is always preferable. The spectrum is comparable to the arousal–valence emotion plane proposed by [Tha90] as a measure for crowd mood. Here the arousal dimension determines the strength of the crowd emotion,

while the valence dimension determines if the reaction was positive or negative [Zha+17, p. 2]. For this thesis the model was linearized to correspond with physical activity.

3.1.2 Measurability

Indoor air quality measures that are caused by human body activity include humidity, CO2 and Total Volatile Organic Compounds (TVOC) concentration. TVOC is the measure of total volume of indoor Volatile Organic Compounds (VOC) concentration [ZS20, p. 3]. VOC cover a huge range of hazardous gaseous or vaporized organic chemicals. The chemical of most interest for this study is alcohol [Wol95, p. 20]. CO2 concentrations are raised by human respiration and humidity is raised by either breath or sweating. The second assumption is that physical activity will directly affect the measurable properties of the air in the room. Increased physical activity will accelerate the metabolism of individuals, which will heat up the body and increase breathing. The body will try to cool down by sweating, this will increase the temperature and humidity in the vicinity. The increased breathing will also lead to a higher CO2 concentration in the room. The frequent consumption of alcoholic drinks will raise the alcohol concentration in each breath of the individuals. If the room were perfectly insulated these measures would directly correlate, however there are biases in place. Firstly, because the room slowly loses heat due to the walls and people opening and closing the door repeatedly. This is exacerbated by the ventilation system in the room, which will be turned on periodically to rapidly cool the room down if it gets too hot. The air quality measures humidity, CO2 and TVOC concentration are not affected by insulation, however the opening of the door as well as the ventilation system will affect them. However, the only door in the room leads to another room instead of outside and its immediate closure is guaranteed by a door closer. Because of this, the third assumption is that we can ignore the biases introduced by the insulation and door, as their effects are miniscule and nearly constant over time. The biases introduced by the ventilation system and the fog machine cannot be ignored and have to be assessed later.

3.1.3 Social milieu

In order to be able to meaningfully assess the impact of song selection over multiple parties, it will be assumed that the social milieu will stay approximately the same. This means that even though the people attending each party might differ, the characteristics of the crowd as a whole stay approximately constant. A significant majority of the guests are regular guests with a relatively constant music taste. The way the crowd reacts to each song will therefore also stay approximately constant from party to party and only change slowly over a time scale of multiple months. For this reason, it will be assumed that songs

that had a positive or negative effect on the crowd's mood at the start of data collection, will still have a similar effect on the crowd's mood at the end of data collection 2 months later.

3.1.4 Crowd size

The last aspect to discuss is crowd size. Crowd size itself can be interpreted as a measure for crowd mood. In this setup people will usually leave the room when bored while people outside hearing good music will usually come inside to dance and sing. However sometimes people will just leave to get a break or smoke a cigarette regardless of mood and some people will enter the room just to get a drink as the only bar is located there. Not to mention that the total amount of people attending the party will fluctuate between parties due to factors outside of our control. Even when accounting for this by measuring "percentage of people attending the party in the room" instead, there are too many factors besides mood why people might leave or enter the room. For this reason, we will dismiss crowd size as a reliable measure of crowd mood.
However, crowd size is still needed as an intermediate measure to calculate the final measure. This is because all the air properties measured are totals. If we assume that crowd mood is mostly independent from crowd size, we have to normalize air property measures to increase in measure per person. The assumption here is that a crowd in a good mood will increase temperature and the concentrations of air properties faster than a crowd in a bad mood, due to a difference in total physical activity.

3.1.5 Ventilation system

These measures cannot increase indefinitely as they will eventually reach an equilibrium, where the insulation and door will decrease the measure as fast as it increases. This is where the ventilation system comes into play. This equilibrium is not pleasant for a human as the temperatures would most often exceed 30°C. When a certain threshold is reached the bartender will turn on the ventilation system to rapidly cool down the room, in order to avoid that people, leave the room. The fresh air from outside will also decrease the humidity, CO2 and TVOC concentrations in the air. Because this process is human controlled, the threshold varies and cannot be exactly determined. However, it is guaranteed that the ventilation system will be activated intermittently rather than continuously. Firstly, because the cooling system of the motor is not effective at low voltages, leading it to eventually overheat. Continuous ventilation at high enough voltages to avoid overheating would cool down the room to much, which would be unpleasant for the guests. Secondly the ventilation system interferes with the fog machine and will clear

out any fog produced by it, which is not desirable. We will therefore assume that any decrease above a certain threshold in the measures is caused by the ventilation system.

3.2 Sensor selection

As discussed in the last chapter, in order to determine physical human activity certain air properties, have to be measured. Namely temperature, humidity, CO2 and TVOC concentrations. In addition to air quality an infrared sensor will be used to determine crowd size. The sensors will be installed on a Raspberry Pi 3 Model B.

DHT22

The DHT22 is a commonly deployed temperature and humidity sensor. It is precalibrated and outputs a digital signal, which is needed for the Raspberry. It can measure temperatures from -40°C to 80°C and relative humidity from 0 to 100%. Its resolution is 0.1% and 0.1°C respectively and it has a sensing period of 2 seconds on average [Ltd]. The DHT22 has the advantage that both temperature and humidity measurements can be done in a single sensor.

SGP30

The SGP30 is a gas sensor for measuring CO2 equivalent and TVOC concentration. It can measure CO2 equivalent in a range of 400ppm to 60000ppm and TVOC concentration from 0ppb to 60000ppb. Output signals are sent over a I^2C bus and the sensor can be manually calibrated [Sen]. To provide a constant baseline in order to be able to compare data from multiple parties, the sensor was calibrated by running for 24 hours without a set baseline. The final baseline was logged and manually set for all subsequent runs. This sensor was chosen as it combines the ability to measure both CO2 and alcohol concentrations in one product.

AMG8833

The AMG8833 is an infrared camera with a resolution of 8x8 pixels. The infrared camera will be used to determine the number of people in the room. It can output up to 10 images per second over the I^2C bus and can measure temperatures between 0°C and 80°C. It has a resolution of 0.25°C and can detect a human up to 7 meters away [Pan].

3.3 Hardware setup

In this section the selected sensors will be wired to the Raspberry Pi and placed into a case. As mentioned in the last chapter, the DHT22 has a digital output signal while the SGP30 and AMG8833 both have an I^2C interface. As illustrated in figure 3.2 all sensors were connected to the 3.3 voltage supply at pin 1 and to the ground at pin 6. The digital output of the DHT22 was connected to the GPCLK at pin 7. A resistor was placed between the 3.3V and data line to stabilize the data transfer. The SGP30 and AMG8833 sensors were connected to the Raspberry Pi's I^2C bus by connecting the SDA and SCL lines on pin 3 and 5, respectively.

The Raspberry Pi and sensors were mounted into a custom-made wood casing as shown in

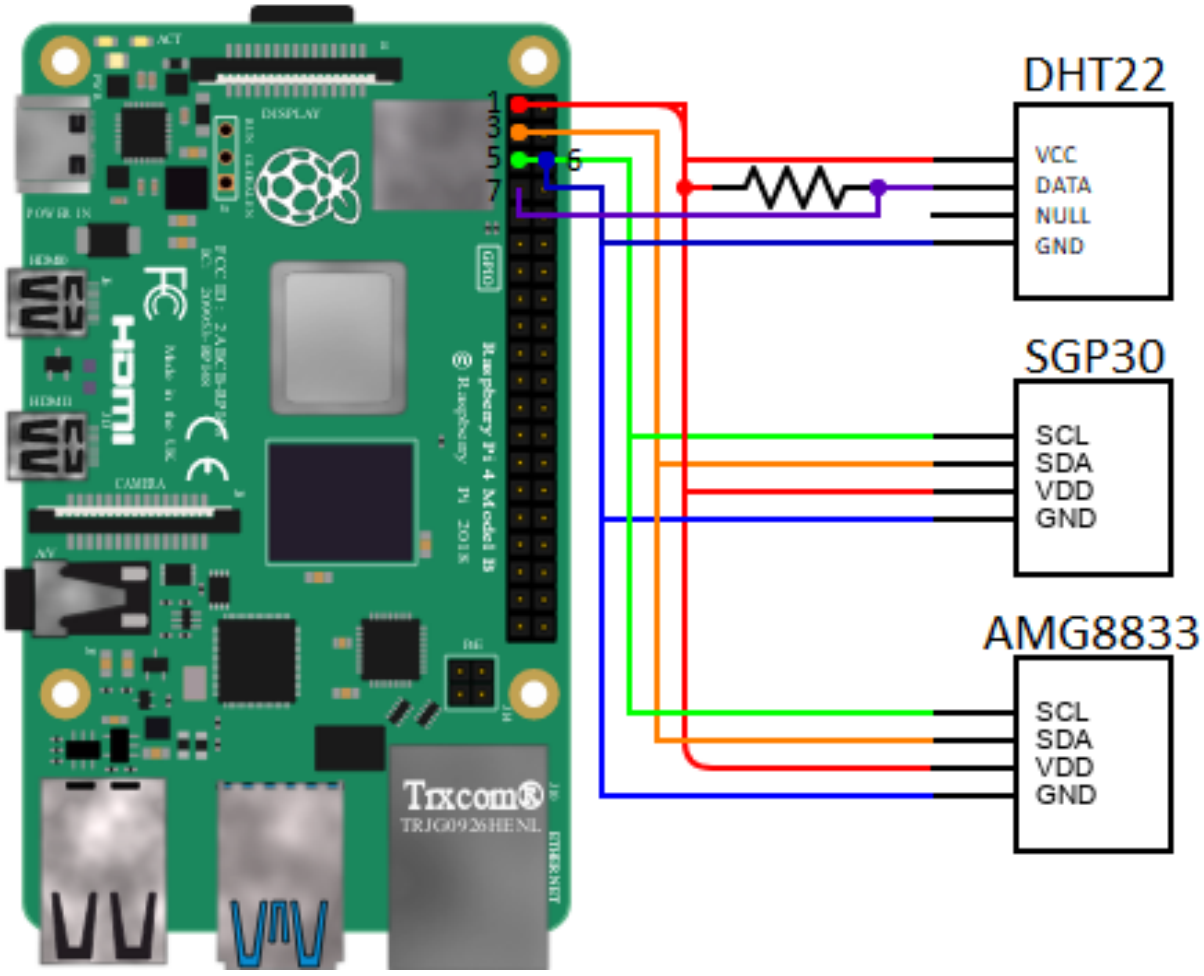
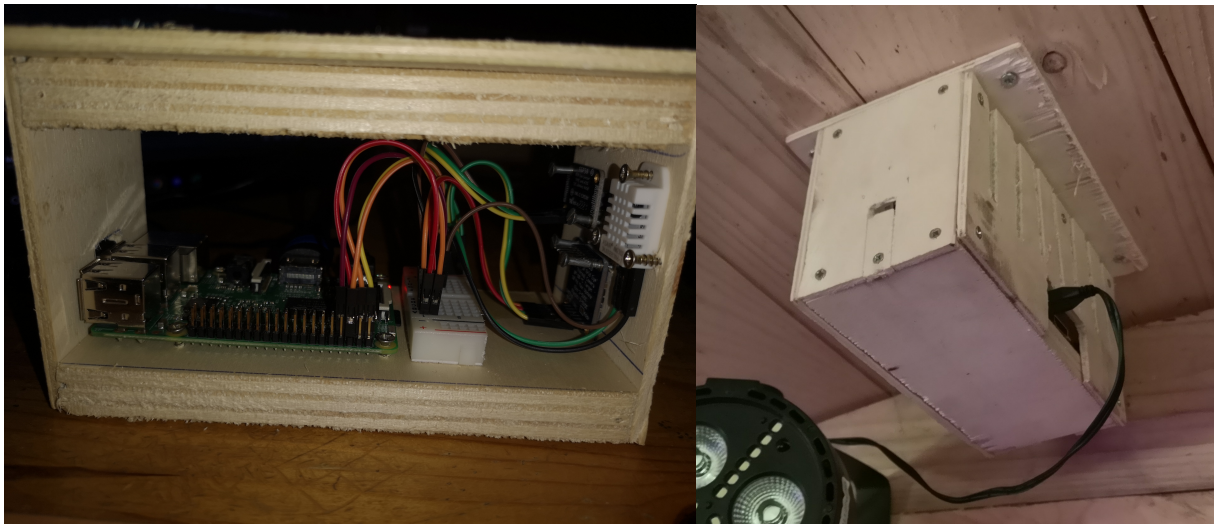


Figure 3.2: Circuit diagram with wires: pin 1 3.3V supply voltage (red); pin 6 ground (blue); pin 3 Serial Data (SDA) (orange); pin 5 Serial Clock (SCL) (green); pin 7 General Purpose Clock (GPCLK) (purple)

figure 3.3a. A circuit board was also placed inside the casing for the wiring to be realized as shown in figure 3.2. The sensors were mounted on the front wall, the AMG8833 had an opening on the front wall for the infrared camera. The wooden case had ventilation

slots on the sides and periphery slots for USB, HDMI, and power connections on the left side and back as illustrated in figure 3.3b. It was mounted on ceiling at the back of the bar in order to avoid damage or destruction by the guests. The sensor had to be installed



(a) (b)

Figure 3.3: (a) Side view of the interior of the sensor with Raspberry Pi and circuit board on the floor and the DHT22, SGP30 and AMG8833 sensors installed on the front wall; (b) Sensor installed on the ceiling with the AMG8833 infrared camera on the front and ventilation and periphery slots on the side and back

tilted slightly to the right as the ceiling was curved and the central position was already occupied by the ventilation system.

3.4 Algorithm

Two algorithms had to be programmed for the experiment. First the data collection algorithm run on the Raspberry Pi that fetches data from each sensor and save it locally. Second the algorithm that saves each song played run on the PC of the DJ. Both programs are run on startup.

3.4.1 Sensor data collection

The program run on the Raspberry Pi had to be resistant to errors as there is no way to check on the status of the program during live testing. The program first loaded the existing csv file containing all data if possible or create a new one. The headers of the csv were timestamp, temperature, humidity, eCO2, TVOC and all 64 infrared pixels. It then initialized all sensors one by one. The main loop ran at first every 60 seconds and

later every 10 seconds and included filling each of the column for a complete datapoint. First the timestamp was included, then the DHT22 was checked for temperature and humidity readings. After that, the SGP30 was checked for CO2 and TVOC readings and the AMG8833 for all 64 infrared pixels, both via the I^2C bus. Finally, the new datapoint was appended to the existing dataframe and the dataframe was saved to the csv file. Logging was used to find potential bugs and all likely exceptions had to be caught. Error cases included

- Csv file not found
- Initialization of sensor failed
- Sensor not found
- Error while reading from sensor

and had to be tracked for each of the three sensors. The program needed to be able to reliably run for a day at most. The exhaustive source code is listed in the appendix.

3.4.2 Song data collection

The script for collecting the song data was written in Windows PowerShell, due to the simplicity of the task and the fact that PowerShell is preinstalled on every Windows PC.

```
$song = $null
$old_song = $null
while ($true) {
    $processes = Get-Process spotify
    $title = (Out-String -InputObject $processes.mainWindowTitle) -
        replace "`n","" -replace "`r",""
    if (-Not $title.Contains('Spotify')) {
        $song = $title
    }
    if ($song -ne $old_song -and $song -ne "Drag") {
        $old_song = $song
        $time = Get-Date
        $newrow = [PSCustomObject] @{"timestamp" = $time; "song" =
            $song}
        $Addr = $newrow
        $Addr | Export-Csv -Path songs.csv -Force -NoTypeInformation
            -Append
        Write-Output "$time, $song"
    }
    Start-Sleep -Seconds 5
```

```
18 }
```

Listing 3.1: PowerShell script for collection song data on the DJ's PC

As visible in listing 3.1 the script takes advantage of the fact that Spotify, the program used by the DJ, updates the process name according to the currently playing song. The script will search for all processes named spotify and then extract their title. Spotify has many processes running at once and most of them are not related to the song, therefore if the extracted title contains the string "Spotify" it will be disregarded. The program checks every 5 seconds for all process titles and if the title is different from the last one, the new title will be saved into a csv file together with a timestamp. Additionally, a bug was discovered where sometimes the process name "Drag" will be passed, which is not a song name. This bug was fixed by simply disregarding the change if the title of the process is "Drag".

4 Data analysis

The goal of this chapter is to clean the data and assess the biases stated in chapter 3.1. This will enable us to engineer the crowd mood measure in the next chapter.

4.1 Data cleaning

In this section the data will be cleaned to be suitable for data analysis. Simple data cleaning like transforming datatypes or setting faulty values to Nan will not be mentioned in detail. Instead, time synchronization and recording frequency issues will be handled and the song data will be joined with the sensor data.

4.1.1 Data structure

As described in chapter 3.4 the recordings are saved as a csv file with the following columns: Timestamp, temperature, humidity, CO2 equivalent, TVOC concentration and pixels 1-64 of the infrared camera. Most handling of faulty readings is already done beforehand by error handling, however the DHT22 sensor does not throw errors when encountering faulty readings. Instead, it records a temperature of -50 and a humidity of 1, which corresponds to faulty readings according to the datasheet [Ltd]. These values have to be manually set to Nan. Additionally, csv files do not save datatypes, we therefore have manually convert the timestamp column to the datetime format. Missing data will not be removed, as there is not a single entry in the dataset that has missing data on all three sensors.

4.1.2 Time synchronization

Due to the fact that the Raspberry Pi had no built in time server, its clock needed to be synchronized after every startup. This synchronization was done by connecting it to the internet via Wi-Fi or mobile Hotspot. However sometimes this synchronization did not happen for several hours depending on the circumstances. The sensor could therefore not wait for an internet connect before it started collecting data, as this might result in vast data loss over several hours. This means the suboptimal solution to start collecting data immediately was chosen, which lead to many datapoints having wrong timestamps. This

had to be manually cleaned up afterwards. After every restart the clock would be set a few minutes back from its shutdown time, which means detection of the datapoints with wrong timestamps would be easy. 13 datapoints were found where the timestamp of the datapoint decreased compared to its predecessor. These datapoints could not be cleaned up automatically, as the Raspberry Pi would sometimes have no internet connection for hours. The faulty timestamps will be assumed to belong to the next startup and corrected accordingly.

1. Identify the start of a series with faulty timestamp by looking for datapoints that have an earlier timestamp than their predecessor
2. Determine the size of the faulty series by looking for the next large time skip. This is highly subjective as sometimes the sensor would restart after a few minutes and sometimes it would be shut down for weeks. For this reason, the task could not be automated
3. Adjust the timestamps of the faulty series by keeping their relative dates but increasing them until the last entry of the series has a timestamp one minute or 10 seconds smaller than the first entry after the time skip, depending on the recording frequency.

Additionally for any faulty timestamps this method did not detect, a visual analysis was carried out. The measures temperature, humidity and average infrared temperature were plotted for each date, and any sudden jumps in the measure at the start or end of a session were manually investigated.

4.1.3 Song data

During cleaning of the played songs data, a song named “Drag” caught our attention. As it turns out this is not a song but actually a bug in the script, probably caused by Spotify loading new songs. The bug was fixed and the entries were removed. The end time of each song had to be manually set, as only the start time was recorded. If another song started within 10 minutes, the end time was assumed to be the start time of the next song. If no other song started within 10 minutes, it was assumed that there was a break in the recording. The exact duration of each song could have been gathered online, however the exact value is not relevant for the analysis. Therefore, these entries were set to the average song duration of the dataset of 2:43 minutes. The song dataframe contained start and end time of each song, as well as the song name and manual recordings on the number of people in the room, if available. It was joined on the data dataframe which contained the sensor readings recorded every 10 seconds. For every recording within the start and

end time of the song, the name of the song as well as the number of people observed was saved in the new columns “song” and “num_people’.

```
for song in songs.values:
# song: ["start", "song_name", "num_people", "end", "length"]
index = np.where((data["timestamp"] >= song[0]) & (data["timestamp"]
    < song[3]))
for i in index:
data.loc[i, "song"] = song[1]
data.loc[i, "num_people"] = song[2]
```

Listing 4.1: Joining the song data with the sensor data

4.1.4 Recording frequency

The initial recording frequency of the sensor data was set to one reading every 60 seconds, however after several live tests this frequency turned out to be too low. Therefore, starting at party 5 the frequency was set to one reading every 10 seconds. This heavily skews statistically measures in favor of party 5 and 6. To account for this, either the low frequency data has to be expanded to contain 6 separate entries for each reading. Or the high frequency dataset has to be contracted to contain the average of 6 readings in one entry. The first approach is problematic, as there is no information about the values between recordings. The second approach is suboptimal as we would lose valuable information. We will therefore opt to separate the data into two datasets, one with recordings every 60 seconds and one every 10 seconds. For direct comparisons with the first dataset, the average per minute of the second dataset will be used.

4.2 Assessing biases

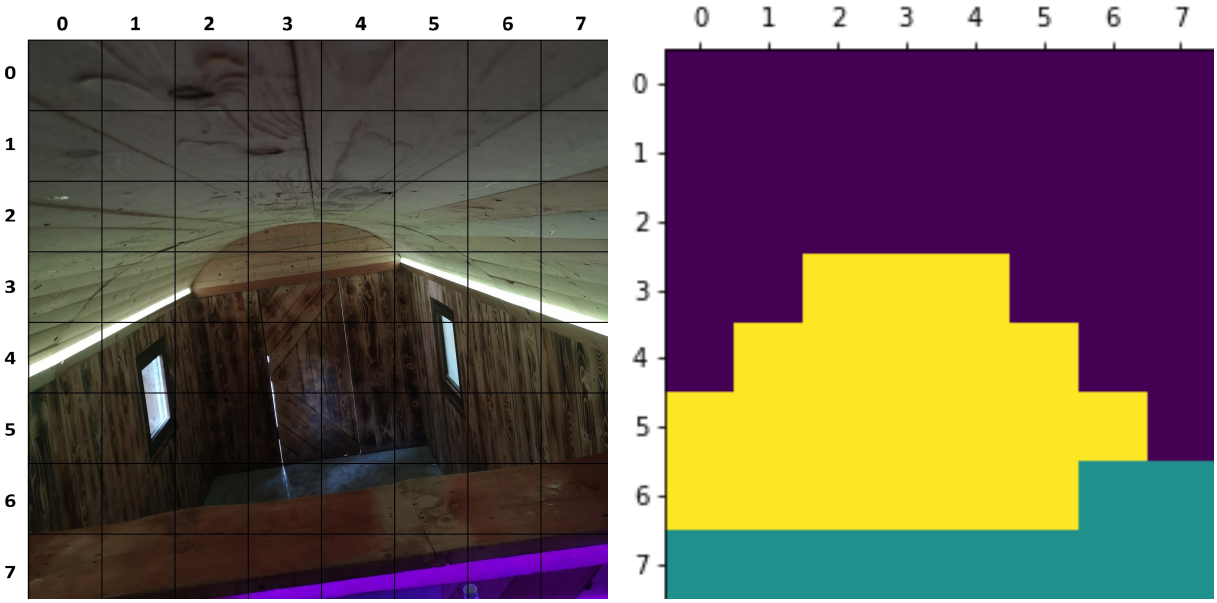
A test was conducted to assess the impact of the most prominent biases, namely the fog machine, ventilation system and cigarette smoke. First the infrared data generated by the AMG8833 8x8 infrared camera will be assessed, then the air property measures generated by the DHT22 and SGP30 sensors.

4.2.1 Infrared data

As discussed in chapter 3.3 the sensor had to be installed at the back of the room. This resulted in the suboptimal field of view of the infrared camera as visible in figure 4.1.

Only 33% of the pixels were aimed at the dance floor, 16% were aimed at the bar and the remaining 51% at the ceiling. The dance floor pixels are of most interest, as they would directly depict the heat signatures of individual guest. Because the bar is always staffed by a single bartender, the bar pixels provide little value. The ceiling pixel would also provide little value besides measuring residual heat in the wood. It has to be noted that these designations are not absolute, neighboring pixels might still be weakly affected by human activity. For example, by people resting their arm on the bar counter or reaching their hands up at the ceiling.

Three tests were conducted to determine the visibility of people in the room under varying



(a) (b)

Figure 4.1: (a) Approximate overlay of the infrared pixels on an image; (b) Pixel designations: ceiling (purple), dance floor (yellow), bar (turquoise)

levels of fog in the air. The first test had no fog, the second test a medium amount and the third test a large amount. One person was present, and he was changing his position after every datapoint. As visible in figure 4.2 the person is clearly visible in the infrared image when standing in front of the bar. This is indicated by one or two pixels measuring 27.75°C or above. When standing 1 meter away from the bar the person is still visible by a single pixel measuring 27.25°C. When standing at the right side at the end of the room the person is still visible by two pixels measuring 26.25 and 26.5°C respectively. A human might be able to recognize the silhouette of the person standing at the left side at the back of the room at the pixels vertical axis 2 and horizontal axis 4 and 5. However there are a total of 6 pixels in this image measuring the same value of 26°C. It is therefore safe to assume that this person is in general not recognizable. The same is true for the person standing in the doorway, this image is indistinguishable from random noise. The fog machine was fired once in figure 4.3 and a medium amount of fog was in the room.

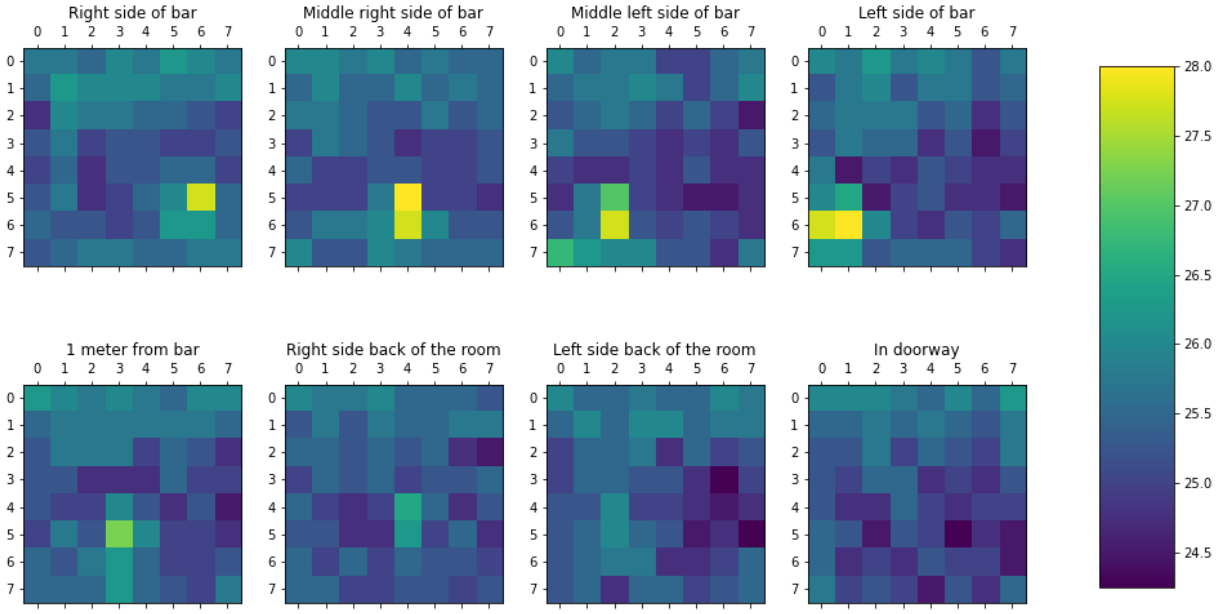


Figure 4.2: Infrared images of a person standing at different positions in the room without fog

The visibility was approximately 2 meters. The person standing inside or directly in front of the bar is still visible by pixels measuring 27.75°C or above. The person standing 1 meter distant from the bar is depicted by a pixel measuring 27.25°C. However, when the person is standing 2 or more meters away from the bar, he becomes unrecognizable. The fog machine was fired twice in figure 4.4 and the room was fully saturated with fog. The visibility has gone down to approximately 0.5 meters. The person standing inside or up to one meter away from the bar is slightly less visible by pixels measuring 27.25°C or above. The heat signature seems less consistent here, as it fluctuates in intensity between images. The person standing 2 meters away from the bar is visible by a pixel measuring 26.50°C. Like in the last test the person standing 3 or above meters away from the bar is not visible in the image.

In conclusion the fog machine has no significant impact on the ability to estimate the number of people based on infrared date. The heat signature of the person has only gone down by 0.5°C between the first and third test. The person standing at the back of the room was sometimes still visible in the first test, while the second and third tests only had visibility up to 2 meters away from the bar. However even without fog recognizing people more than 2 meters away was inconsistent. This means that regardless of fog the infrared camera is likely to only be able to estimate the number of people in the front half of the room. However, it is assumed that this does not affect the final measure as the crowd is usually evenly spread out across the room.

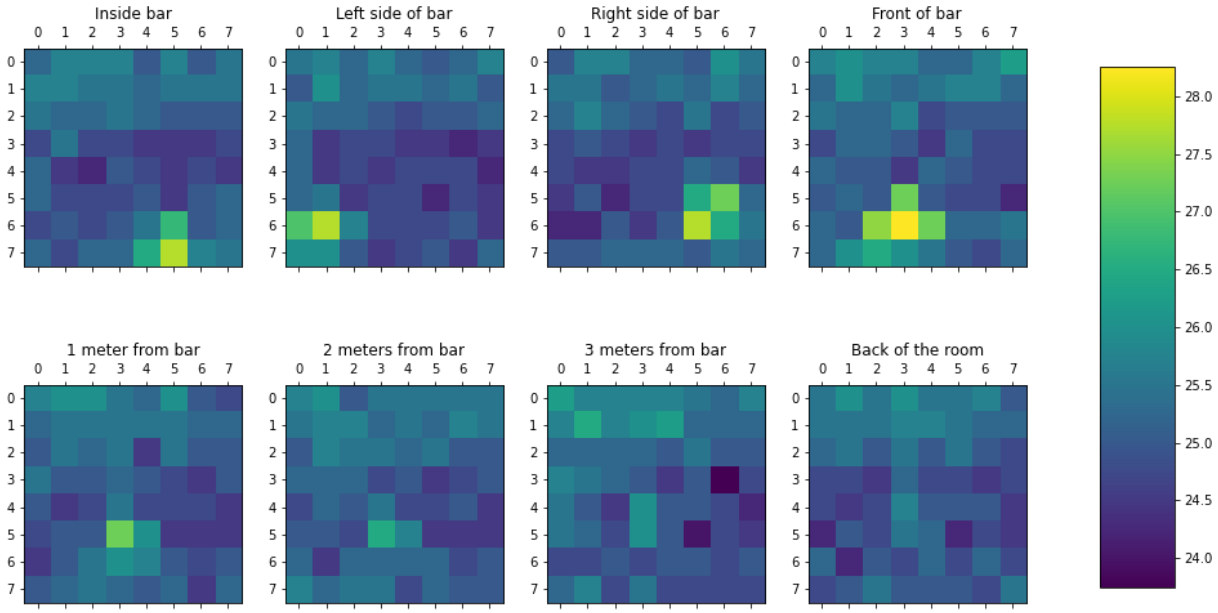


Figure 4.3: Infrared images of a person standing at different positions in the room with a medium amount of fog

4.2.2 Air properties

Next the impact of biases on the air properties temperature, humidity, CO2 equivalent and TVOC concentration will be assessed. A test was conducted to determine the impact of the ventilation system, fog machine and cigarette smoke on the various measures.

Humidity

As visible in figure 4.5 the ventilation system had a large impact on the humidity in the room. However, depending on how long the ventilation system is active, the humidity will return to its prior average with only minor differences. Before the ventilation system was activated the first time, humidity was on average 59%, it then dropped to a low of 56% when the ventilation system was active for 72 seconds and returned to an average of 58% afterwards. The second time the ventilation system was activated for 4 minutes, which reduced humidity to a low of 52% and returned to an average of 56% afterwards. The fog machine had a miniscule impact on humidity, after activation the humidity would dip briefly to 0.7% below average before returning to its prior average within 60 seconds. Smoking a cigarette had no impact on humidity, the small spike at the end was likely due to smoke and therefore high humidity breath being blown directly at the sensor.

Temperature

As visible in figure 4.6 the impact of the ventilation system on room temperature was less severe. The room was heating up approximately 0.15°C per minute due to sun exposure.

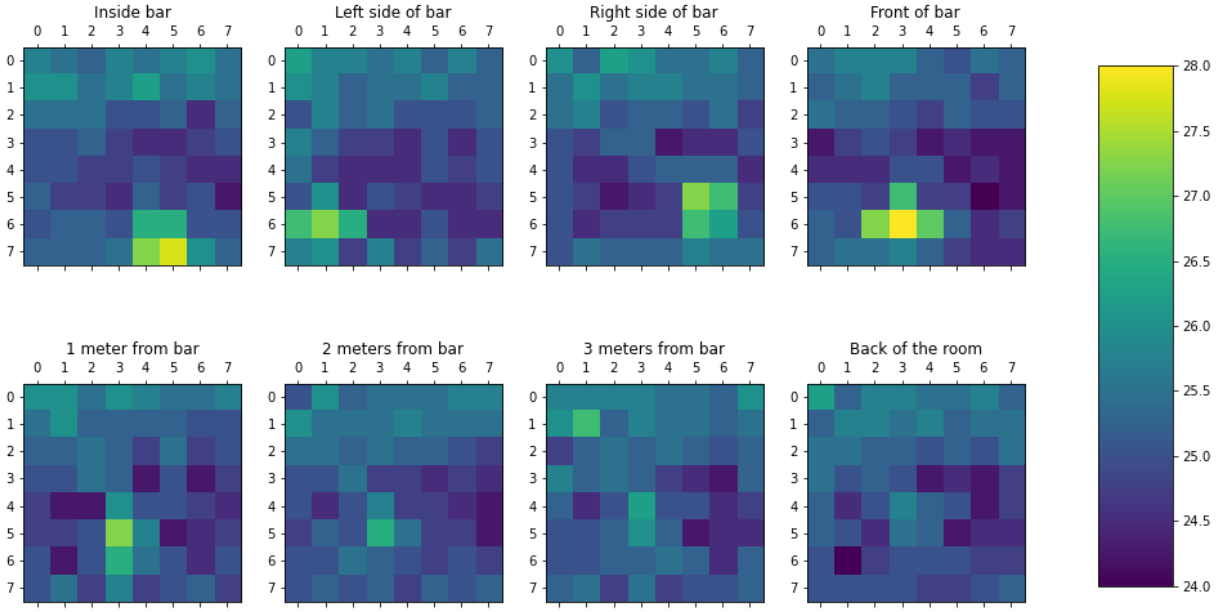


Figure 4.4: Infrared images of a person standing at different positions in the room with a large amount of fog

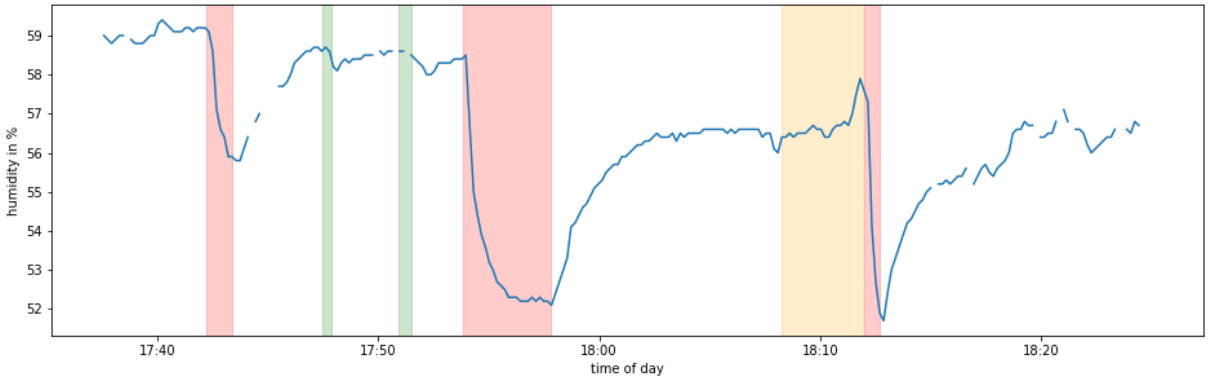


Figure 4.5: Testing the influence of the ventilation system (red), fog machine (green) or smoking a cigarette (orange) on humidity

The first activation did not cool down the room but lead the temperature to roughly stay constant at 24.3°C for 137 seconds. The second activation cooled down the room from 25.3°C to 24.5°C. It has to be noted that the ventilation system will have a more severe impact when the difference between inside and outside temperature is higher, which is the case during parties. Outside temperature were approximately 23°C during testing. The fog machine and cigarette smoke had little to no impact on temperature. The average infrared temperature of all pixels was greatly influenced by the ventilation system, but the fog machine and cigarette smoke had little impact. The average infrared temperature was reduced by 2°C or more exactly while the ventilation system was active. This suggests that the stream of cold air might be visible in the infrared image, this will be investigated later.

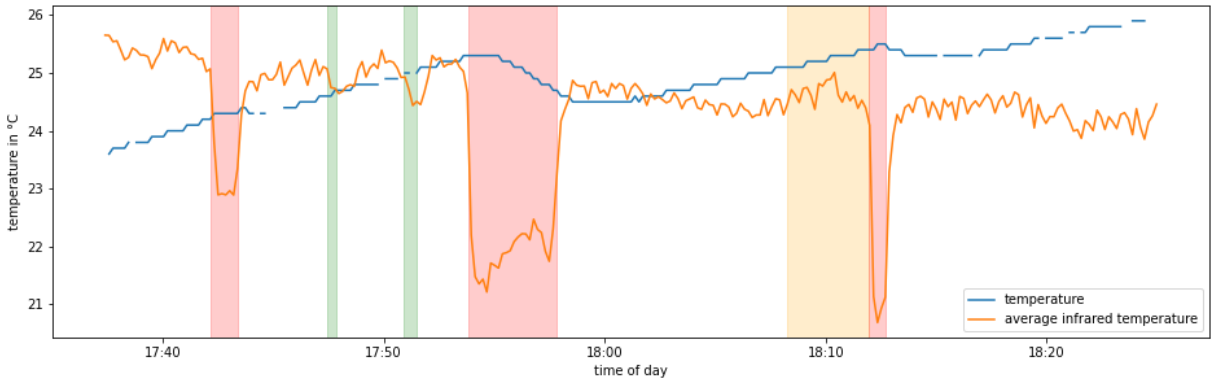


Figure 4.6: Testing the influence of the ventilation system (red), fog machine (green) or smoking a cigarette (orange) on temperature and average infrared temperature

CO2 and TVOC

As visible in figure 4.7 the CO2 equivalent and TVOC concentration measures proved

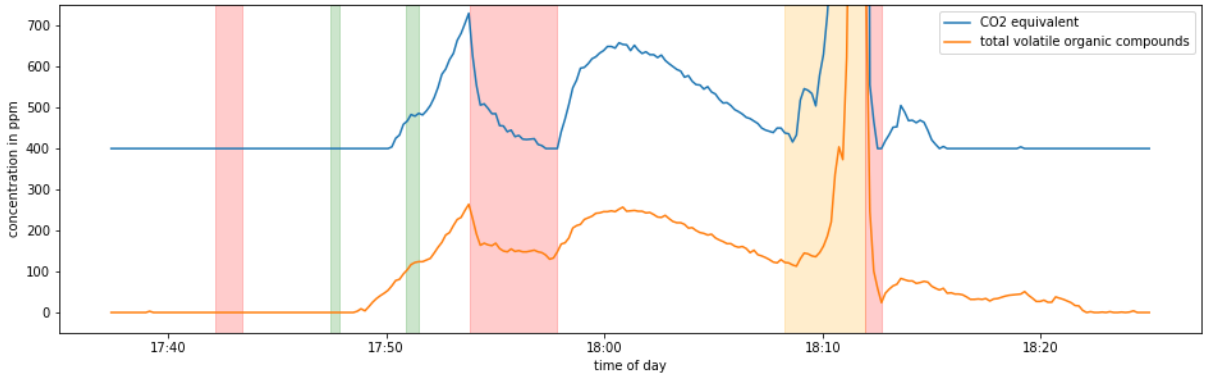


Figure 4.7: Testing the influence of the ventilation system (red), fog machine (green) or smoking a cigarette (orange) on CO2 equivalent and total volatile organic compounds concentration

to be the most volatile. These are the only measures that are significantly influenced by the fog machine. The first activation of the fog machine raised CO2 concentration from the base of 400ppm to 459ppm and the TVOC concentration from the base of 0ppm to 94ppm. The second activation subsequently raised CO2 concentration to 730ppm and TVOC concentration to 264ppm. The first activation of the ventilation system had no impact on CO2 and TVOC concentrations as they were already at base level. The second activation reduced CO2 concentrations from 730ppm back to its base level of 400ppm and TVOC concentration from 264ppm to 130ppm. The second activation did not completely clear the fog in the room, which is why CO2 returned to 730ppm and TVOC to 264ppm within 3 minutes. However, because the fog was now thin it decayed within 8 minutes reducing the CO2 concentration to 439ppm and TVOC concentration to 121ppm. So far, these measures correlated highly with the actual concentration of fog in the room, only getting temporarily disrupted by the ventilation system. A cigarette was lit approximately 1 meter away from the sensor, which raised CO2 and TVOC concentration immediately

to 546ppm and 145ppm respectively. When breathing cigarette smoke directly at the sensor to simulate a room saturated with smoke, the CO2 concentration skyrocketed to 25412ppm, 58 times the median of the entire dataset. TVOC concentration was also raised to 3244ppm, 21 times the median of the dataset.

Conclusion

In conclusion the fog machine had less impact on most of the measures than anticipated. Cigarette smoke as well seems to be only relevant for CO2 and TVOC concentration. Therefore, it will be assumed that the biases introduced by the fog machine and cigarette smoke can be ignored for temperature, humidity and infrared data. This means the only bias to consider for them is the ventilation system. CO2 and TVOC concentration are problematic to work with, as there are greatly influenced by the fog machine, ventilation system as well as cigarette smoke. It is assumed that human activity is the smallest factor out of all that contributes to CO2 and TVOC concentration levels. The only use these measures might be able to reliably provide is to estimate the thickness of fog in the room, when the spikes generated by cigarette smoke are removed. For this reason, temperature, humidity and infrared data will be preferred as a measure for calculating crowd mood in subsequent chapters.

4.2.3 Detecting ventilation activity

As discussed in the last chapter, the sudden drop in average infrared temperature during ventilation might be an indicator that infrared data can be used to determine ventilation activity. If exact ventilation times are known, removing biases based on the ventilation system becomes vastly easier.

When assessing the infrared images in figure 4.8 before, during and after ventilation, no stream of cold air is recognizable. All pixels seem to drop uniformly to a lower temperature and return to their prior values immediately after ventilation. This might indicate that the stream of cold air completely blocks the field of view of the infrared camera when the ventilation system is active. There is no testing data available to determine if the ventilation system also blocks the heat signatures of humans. figure 4.9 shows infrared images of a crowd of size 11 in the first image and size 13 in images 2 to 4. The ventilation system was engaged for a total of 182 seconds 5 seconds after image 1. All images are 60 seconds apart from one another. As visible the ventilation system does not seem to disrupt the infrared camera's ability to detect the crowd heat signature. In fact, all pixels are approximately uniformly 4.8°C colder in image 2 than in image 1, with the standard deviation only getting raised from 1.2 in the first to 1.33 in the second image. The standard deviation in images 2 to 4 stays approximately constant at 1.45 within an

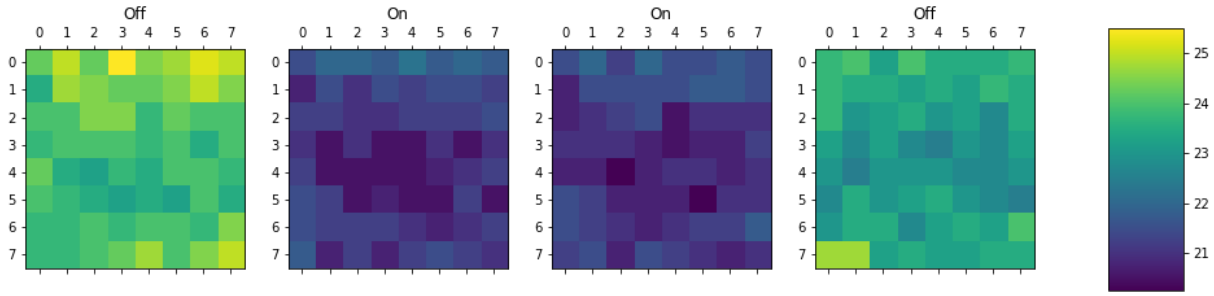


Figure 4.8: Testing the influence of the status of the ventilation system on infrared data

error of 0.12. The raise in standard deviation between image 1 and 2 can be contributed to the increase in number of people in the room from 11 to 13. In conclusion the heat signature of the crowd is still visible when the ventilation system is active, with the key figure of standard deviation staying approximately the same. Additionally sudden drops and spikes in average infrared temperature can be used to determine ventilation system activity.

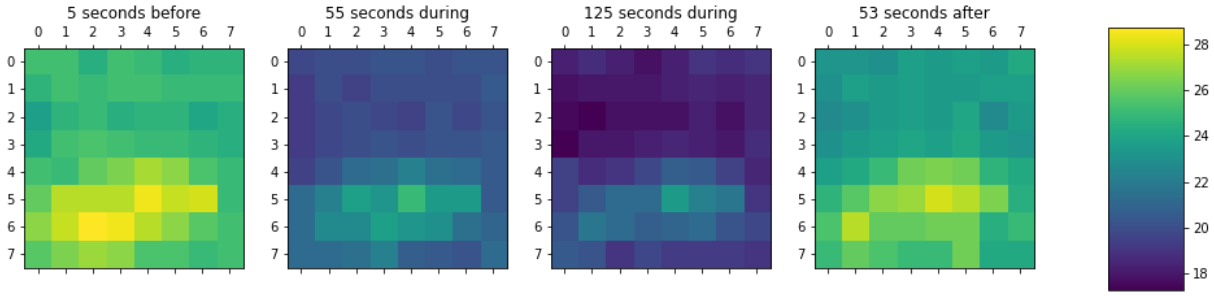


Figure 4.9: Examining infrared images of a crowd before, during and after activating the ventilation system

5 Supervised Learning

This chapter will use supervised learning techniques to build models based on infrared data to determine ventilation activity and predict the number of people in the room. All the models used in this chapter are from the Python library Scikit-learn [Ped+11]. Finally the crowd mood measure based on those predictions will be defined.

5.1 Predicting crowd size

In order to calculate a consistent measure, we have to normalize any quantity used to the crowd size. This data is only partially available. On a single party the number of people in the room was manually counted approximately every 3 minutes for 3 hours. Additionally in a 2-hour test with a single person the number of people was recorded every 10 seconds. This training data is incomplete as it does not contain sufficient data for crowd sizes between 2 and 6 people. Supervised learning will be used to create a model based on those manual measurements and the data from the infrared sensor to estimate the number of people in the room for all other parties.

5.1.1 Standard deviation model

As discovered in chapter 4.2.3 the standard deviation of the infrared pixels seems to be preserved regardless of ventilation. This allows us to build a model without having to account for ventilation bias. As illustrated in figure 5.1 there seems to be a linear relationship between crowd size and infrared standard deviation across all 64 pixels. This correlation is very strong at 86%. In this case the suboptimal field of view of the infrared camera is actually an advantage, as the pixels aimed at the ceiling provide a base value for the calculation of the standard deviation. If all pixels were aimed at the dance floor, standard deviation would decrease with crowd size once the room is more than half full.

Four models were trained to predict crowd size, a decision tree model, a random forest model, a AdaBoost model and a linear regression model. The decision tree model performed the worst on the training data with a r2 score of only 61%. The AdaBoost model performed better with a r2 score of 71%. The best was the linear regression model with a r2 score of 79% and the random forest model with a r2 score of 87%. The random forest model outperforms the linear regression model by 8%, however a random forest model is not

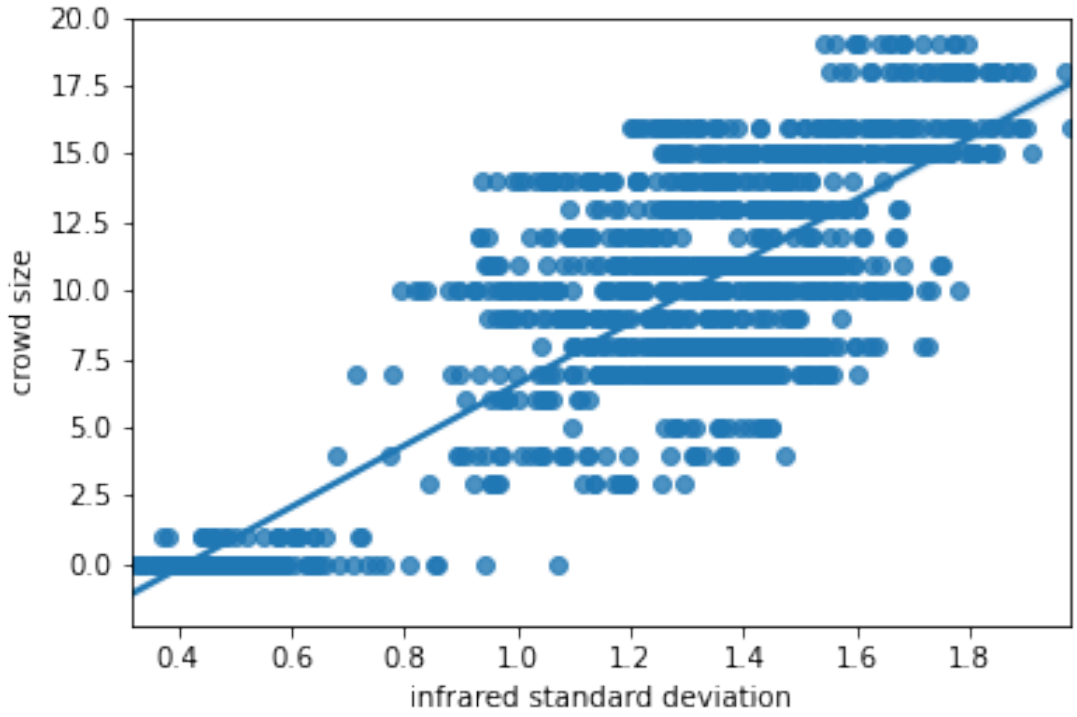


Figure 5.1: Scatterplot with crowd size and infrared standard deviation showing a linear relationship

optimal in this case because decision trees cannot generalize well to predict new values not present in the training data as discussed in chapter 2.1.3. The training set contains little data corresponding to crowd sizes of 2 to 6 people and no data above 19, the random forest model will not be able to make accurate predictions for these crowds. The best match is therefore the linear regression model, as it is able to reliable predict new values if the underlying relationship is truly linear.

The biggest problem with this model is when the ceiling gets heated up by sun exposure. Because the room is mostly shadowed by trees during the day, this only comes to effect at sundown between 7:30pm until the effect cools off at approximately 11pm. Additionally if the sensor runs long enough, the effect is also visible starting at sunrise at approximately 9am. This effect is of course dependent on the weather. Additionally, the effect seems to apply when the party is over, and the room slowly cools down. Heat rises up and escapes through the roof, which results in the ceiling being hotter than the floor and walls. Under normal circumstances the effect is not relevant, as parties start at 10pm and end before sunrise. The ceiling pixels cannot be simply excluded from the calculation of the standard deviation to solve the problem, as they are needed as a basis to guarantee that crowd size scales linearly with standard deviation. One possible solution is to identify infrared data where the ceiling pixel average is greater than the dancefloor pixel average and exclude them from the prediction. This does not account for the edge case that there is both a

large crowd and a hot ceiling, which would predict less people than are actual in the room due to a lower standard deviation. However, this is not relevant in this case because large crowds usually only gather after 11pm when the effect is already gone. The excluded predictions cannot be assumed to be 0, as the effect could be active while people are in the room. As illustrated in figure 5.2 the predictions vary greatly from case to case. The only

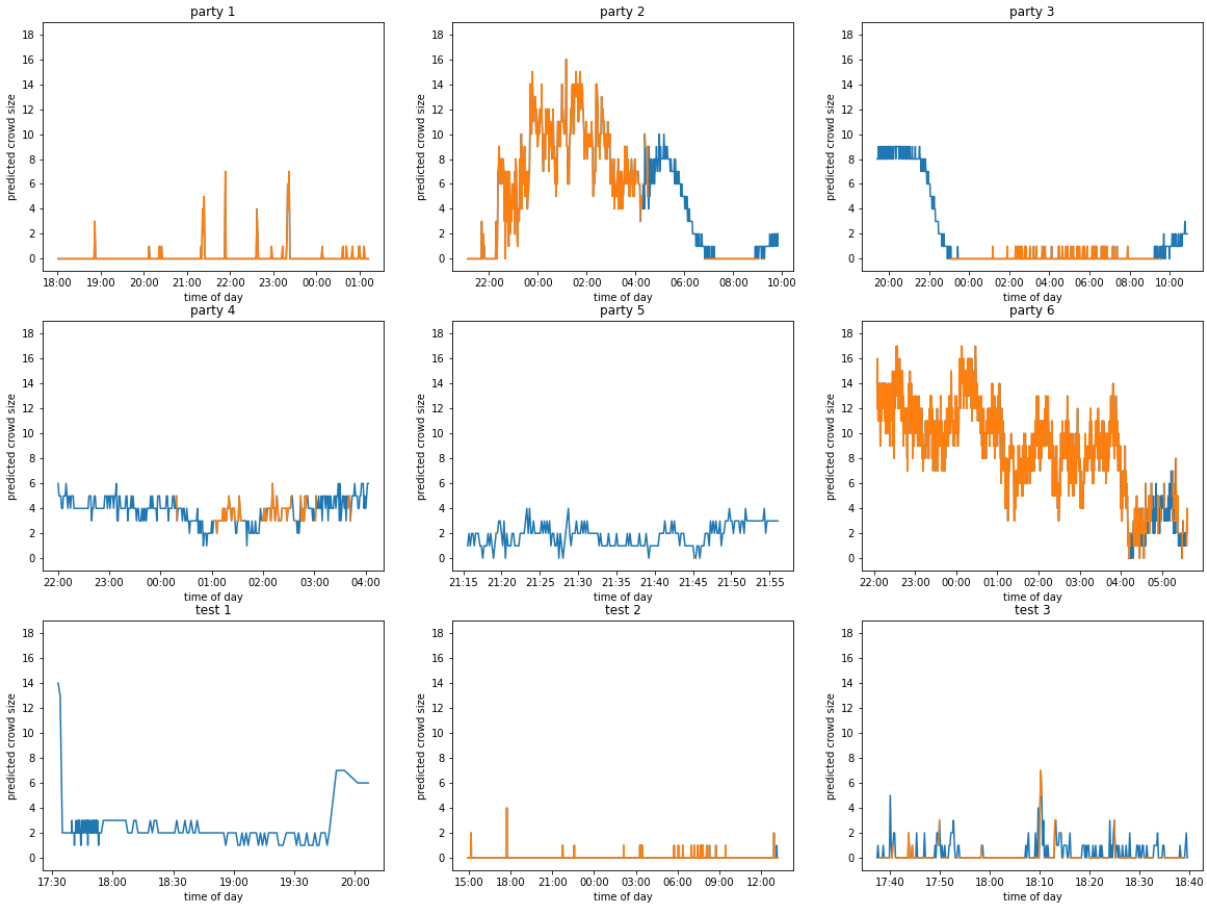


Figure 5.2: Predicted crowd sizes by the standard deviation model for every party and test: original predictions (blue), hot ceiling excluded (orange)

parties were crowd sizes regularly exceeded 10 were party 2 and party 6. This matches real life observations, as the other parties were poorly attended. For party 1 and 3 the predictions match the expected values, except for the hot ceiling effect being active at the beginning and end of party 3. The predictions for party 4 and 5 are expected to look like party 1 and 3, however the algorithm predicts on average 2 people constantly in the room. A second model will be built to try to fill in the missing values and uncover potential discrepancies.

5.1.2 Pixel model

For this model the individual pixels of the infrared camera will be used as a direct input for the model. This is in general not preferable to the standard deviation model as the complexity is greatly increased. This prohibits us from easily finding and addressing the weaknesses of the model. However, because the standard deviation model cannot handle infrared images with a hot ceiling, we will try to fill these missing values with this model. As illustrated in figure 5.3 the pixel model does not seem to be influenced by hot ceilings.

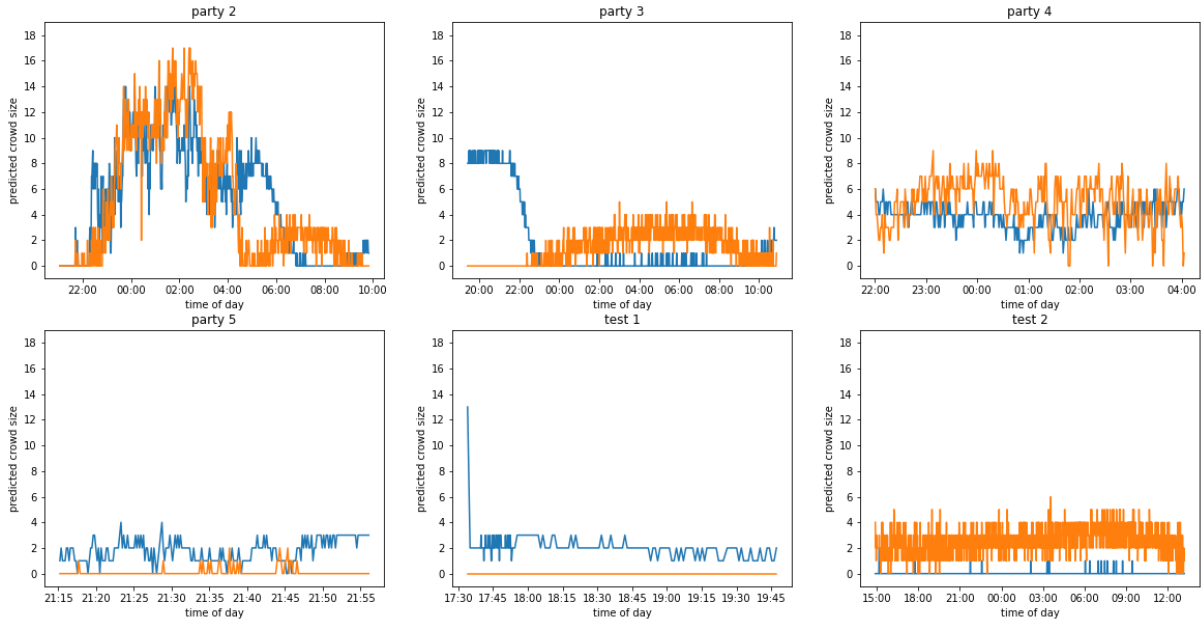


Figure 5.3: Predicted crowd sizes by the standard deviation model (orange) and pixel model (blue) for selected parties and tests

The model also has a lower mean squared error on testing data at 6.19 compared to the 7.47 of the standard deviation mode. The predictions for party 5, test 1 and party 3 up until 23pm are much more realistic and constant, compared to the predictions made by the standard deviation model. However, the model seems to have a bug that causes predictions to slowly rise and fall again over night when no one is in the room, visible at party 2 after 5am and party 3 after 23pm. Because the model has 64 input dimensions, it is difficult to find the cause of this bug. This is likely a symptom of insufficient training data, as there is no training data on the room cooling down after a party. In general, the model seems to be more susceptible to noise, as the predictions change sharply from minute to minute with no apparent reason. However, because the model seems to be able to reliable predict crowd size at datapoints where the standard deviation model is influenced by a hot ceiling, we will opt to combine the models by using the pixel model to fill in the missing predictions from the standard deviation model.

5.1.3 Combined model

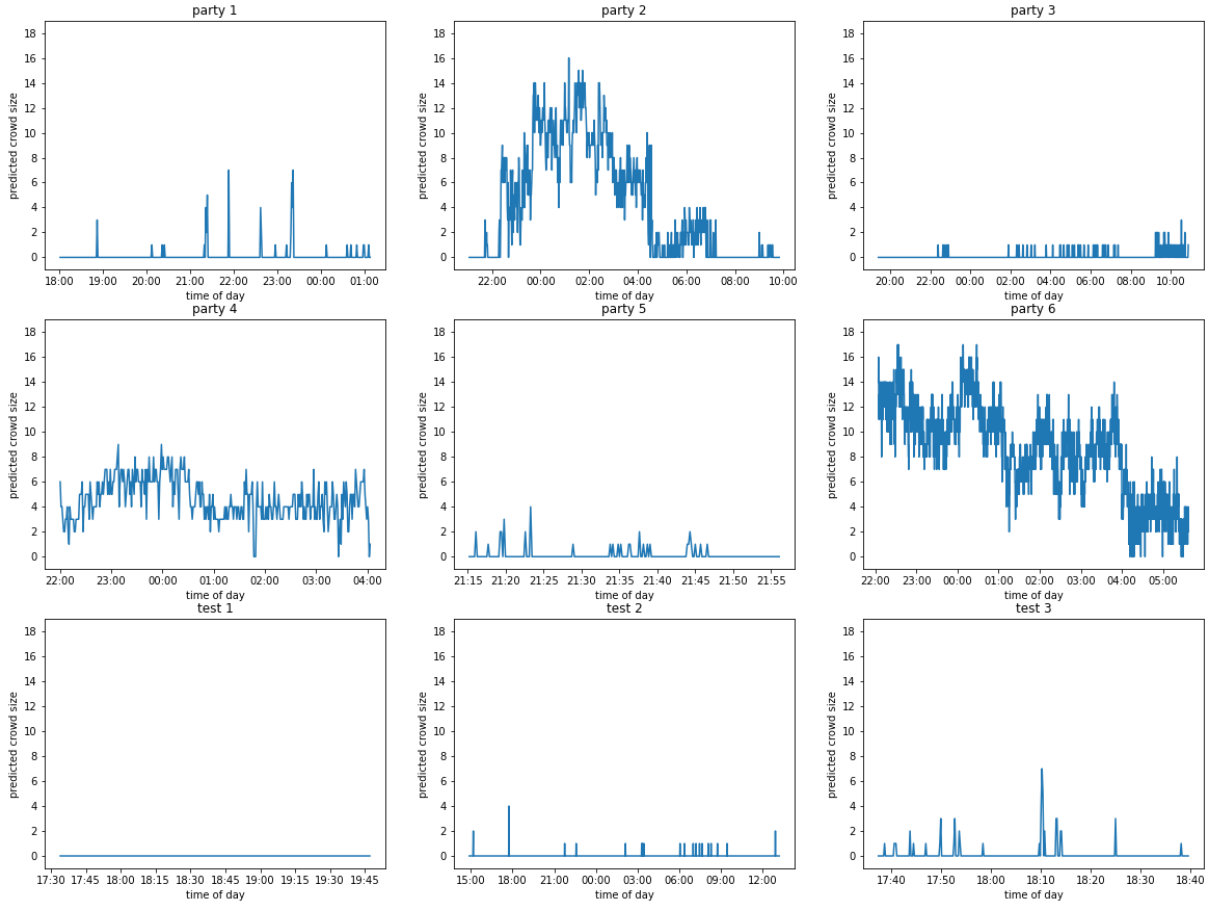


Figure 5.4: Predicted crowd sizes by the combined model for every party and test

As illustrated in figure 5.4 the combined model combines the strengths of both models to both accurately predict large crowds and empty rooms. The biggest problem with this approach is that there is no way to prove the correctness of the model beyond training accuracy, the model only seems intuitively correct. Party 1 was a pool party, so guests primarily stayed outside, only coming inside sporadically to get drinks. The large spikes of 8 people seem too high, but other than that the combined model predicts expected values for crowd size. Party 2 was well visited, with the crowd size peaking at 16 people at 1am. The crowd size then slowly decreases to approximately 8 until the party was closed at 5am. The slow rise in crowd size after that is presumably due to the bug in the pixel model discussed earlier. It will be assumed that party 2 closed at 5am and the predicted crowd size will be manually set to 0 for all datapoints after that. Party 3 was a pool party too, therefore people only went inside to get drinks sporadically. It is highly unlikely that people still were present after 8am, it will be therefore assumed that the predictions there are caused by the sunrise and will be set manually to 0. Party 4 was at a different party location, any predictions made by the model can therefore not be applied to it. It is interesting to see that the predictions are plausible, however the validity of the

predictions is impossible to address. Party 5 only contains data for half an hour, however crowd sizes between 0 and 4 are plausible for this time period. The predictions for party 6 matches expectations, as it was the best attended out of all parties. The model was trained on data from this party, it is therefore reasonable to assume that these predictions match closely match reality. Test 1 did not include any people and the expected crowd size of 0 was reliably predicted. Tests 2 and 3 contained at most 1 person. This means the model wrongly predicted too many people in the room in 12 datapoints. However, it can reliably predict the crowd size of an empty room. This combined model will be used for the calculation of the final measures in the next chapters.

5.2 Predicting ventilation activity

As discovered in chapter 4.2.2, ventilation activity greatly affects the measures. To address this bias a model will be built to predict ventilation activity based on sudden drops in humidity and average infrared temperature as discussed in the last chapter. First attempts using a random forest model were unsuccessful. As visible in figure 5.5 the model is

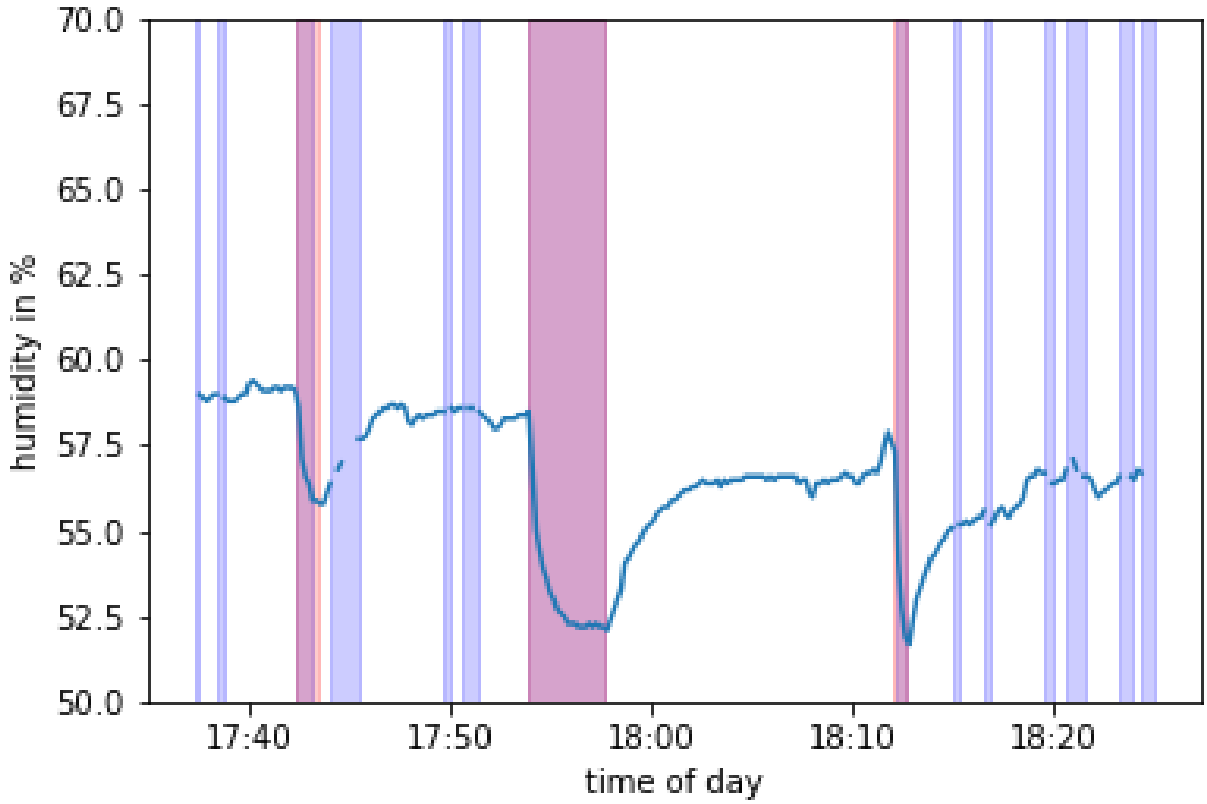


Figure 5.5: Predicting ventilation activity on training data using a random forest model: predicted ventilation (blue), predicted and actual ventilation (purple), actual ventilation (orange)

too greedy when trained on the limited training data available. In this case the model

would be preferred to have a high precision, as any false positive in predicting ventilation activity would result in a loss of data. Only ventilation activity with significant impact on the measures should be filtered out. To address this a model was built manually, which assumed that ventilation activity started once both the average infrared temperature and humidity suddenly decreased sharply and lasted until both measures suddenly increased sharply again.

```
1  pred_ven = list()
2  for i in range(data.shape[0]):
3  idx = data.index[i]
4  time_diff = data["timestamp"].diff().loc[idx]
5  if pd.isnull(time_diff) or time_diff > pd.Timedelta("01:00:00"):
6  ven = 0
7  elif data.loc[idx]["inf_mean_diff"] < -1.28 and (data.loc[idx:idx+4][
       "humidity_diff"] <= -1.19).any():
8  ven = 1
9  elif data.loc[idx]["inf_mean_diff"] > 0.77 and (data.loc[idx:idx+5]["
       humidity_diff"] >= 0.2).any():
10 ven = 0
11 pred_ven.append(ven)
12 data["ventilation_pred"] = pred_ven
```

Listing 5.1: Custom model to predict ventilation activity

The model was programmed as shown in listing 4.2. The begin of ventilation activity in line 8 was marked by a sudden decrease in average infrared temperature by more than 1.28°C followed by a decrease in humidity by at least 1.19% in any of the next 3 datapoints. Ventilation activity lasted until one of two termination criteria were met. Either the sensor was shut down for more than 1 hour in line 6. Or the average infrared temperature suddenly increased by more than 0.77°C followed by an increase in humidity of at least 0.2% in any of the following 4 datapoints in line 10. The thresholds were manually fine-tuned to achieve a training accuracy of 100% and meet expectations in the testing data. As illustrated in figure 5.6 these expectations were that there is no predicted ventilation activity in party 1, party 3, party 4, test 1 and test 2, as there are no sudden changes in both average infrared temperature and humidity. The model correctly predicts the training data of test 3 and the expected ventilation activity in party 5. As illustrated in figure 5.7 the predictions for ventilation activity for party 2 seems to match intuition. Every sudden drop in both average infrared temperature and humidity is predicted to be caused by ventilation activity. Of course, it is impossible to assess the validity of the model beyond intuition, as there is no data to compare to. As illustrated in figure 5.8 party 6 is more difficult to assess, as there are multiple drops in average infrared temperature and humidity that are not predicted to be caused by ventilation activity. However, at closer

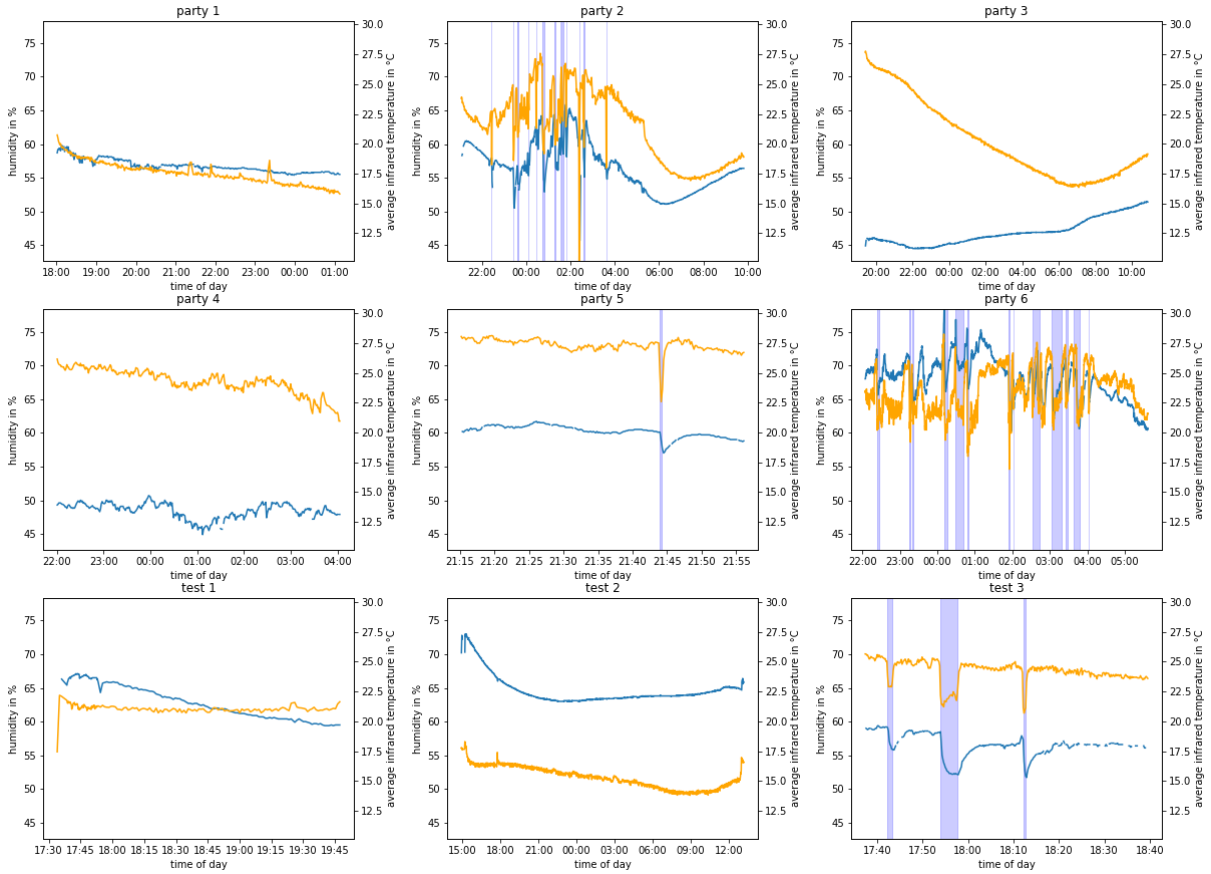


Figure 5.6: Predicting ventilation activity for every party and test using a custom-built model: humidity (blue), average infrared temperature (orange), predicted ventilation activity (marked blue)

inspection this prediction might be correct. The drop in average infrared temperature of 1.37°C at approximately 11:30pm only coincides with a gradual drop in humidity of 0.2% every 10 seconds instead of a sudden drop. The drop at approximately 23:50pm was only a decrease of 0.86°C in average infrared temperature, therefore not sufficient to count as ventilation activity according to the model. This underlines the difficulty in assessing the validity of the model, as it is highly biased by the expectations of the model builder.

5.3 Measure

In the last sections models were built to predict both crowd size and ventilation activity. The predictions of those models will now be used to engineer a measure for crowd mood. Because the ventilation model cannot be easily verified, we will create 3 measures for every attribute of interest. The first measure is based on the uncleaned data, with all ventilation biases still in place. The second measure will ignore any data that is affected by the ventilation bias as predicted by the ventilation model. The third measure will only

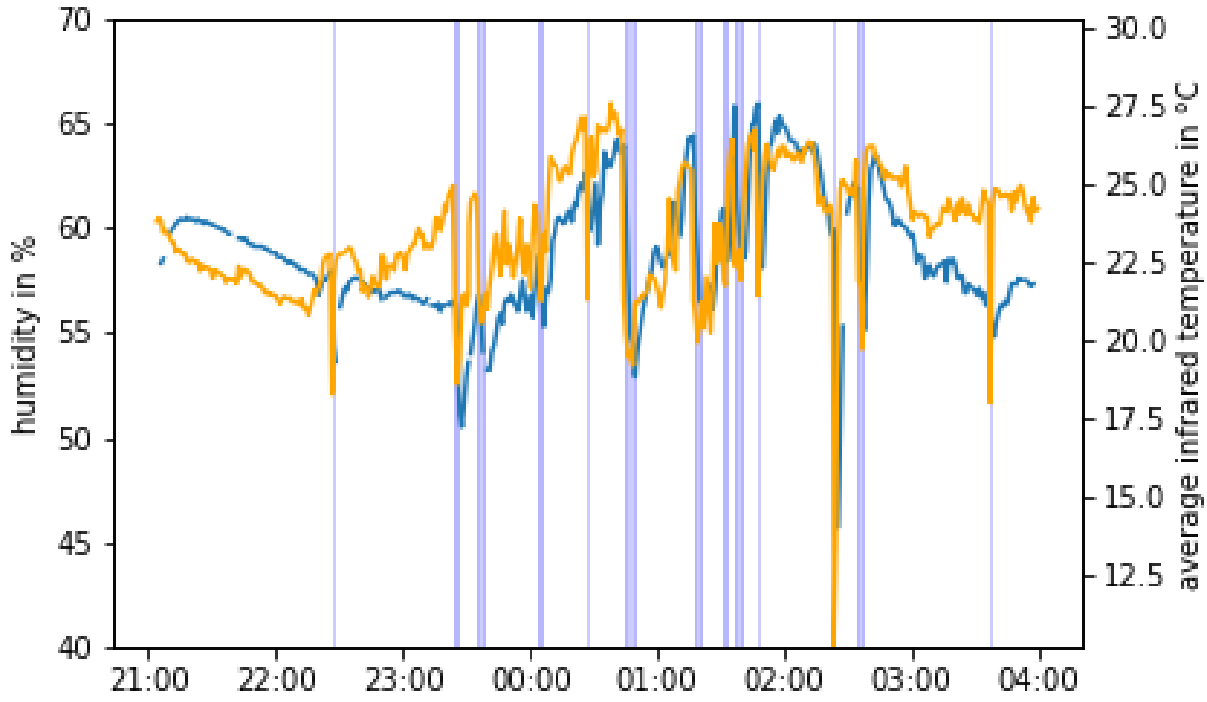


Figure 5.7: Predicting ventilation activity for party 2 using a custom-built model: humidity (blue), average infrared temperature (orange), predicted ventilation activity (marked blue)

consider increase in measure and will therefore consider any decrease as getting caused by the ventilation bias. By creating these three measures the impact of the ventilation bias and bias cleaning on the final measure will be determined. The four attributes that will be considered are temperature, humidity, CO2 equivalent and Total Volatile Organic Compounds (TVOC) concentration. As discussed in chapter 4.2.2 CO2 equivalent and TVOC concentration are massively influenced by multiple biases and might not be suitable as a final measure. In order to be able to compare the two datasets with recording frequency 10 and 60 seconds, the measure will be on a per minute basis.

5.3.1 Song frequency

The largest problem is that despite the dataset having 7278 entries and 371 distinct songs, the average song only got played 1.4 times in total. This is in part due to the fact that only less than half of the datapoints at 3429 have song data associated. This means that for the average song no statistically meaningful statement can be given on their impact on crowd mood as the sample size is too small. As illustrated in figure 5.9 273 songs got played only once, 71 twice and only 27 more than twice. Because a sample size of one is likely to be significantly impacted by bias, those entries will have to be dismissed. Two rankings will be established, the first with all songs that have been played two or more

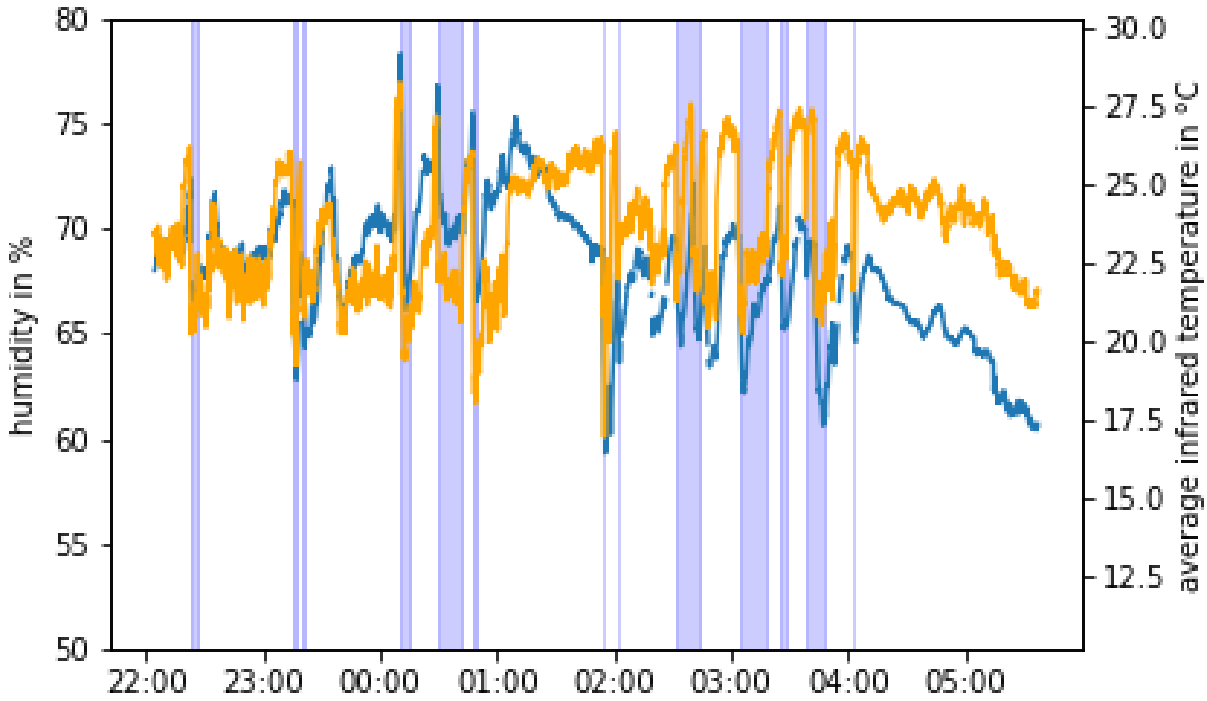


Figure 5.8: Predicting ventilation activity for party 6 using a custom-built model: humidity (blue), average infrared temperature (orange), predicted ventilation activity (marked blue)

times, and the second with all songs that have been played three or more times. The first might still be subject to biases, however the likelihood of the song getting influenced by the same bias twice is reduced. The second has an even lower likelihood, at the cost of even less songs that are considered. The first ranking reduces the number of rows involved to 1786 and the second ranking to 831.

5.3.2 Definition

In total there are 3 measures for 4 attributes resulting in a total of 12 measures each on 2 datasets. To achieve this the measure was grouped by song while applying the mean function. Then the dataframe was sorted by the measure with descending values and each song was given a score according to the placement in the ranking starting at 0. To determine the similarity of the 12 rankings the mean squared error between them was calculated one by one. The mean squared error was used instead of the mean absolute error to punish the distance between rankings more. This way ranking a single song different by 10 places would be punished 10 times more than ranking 10 songs different by a single ranking. The results of these calculations are illustrated as a matrix in figure 5.10 with each axis representing the measure number between 0 and 12. Measures 0-2 correspond to ranking by temperature, 3-5 to humidity, 6-8 to CO2 equivalent and 9-12 to TVOC

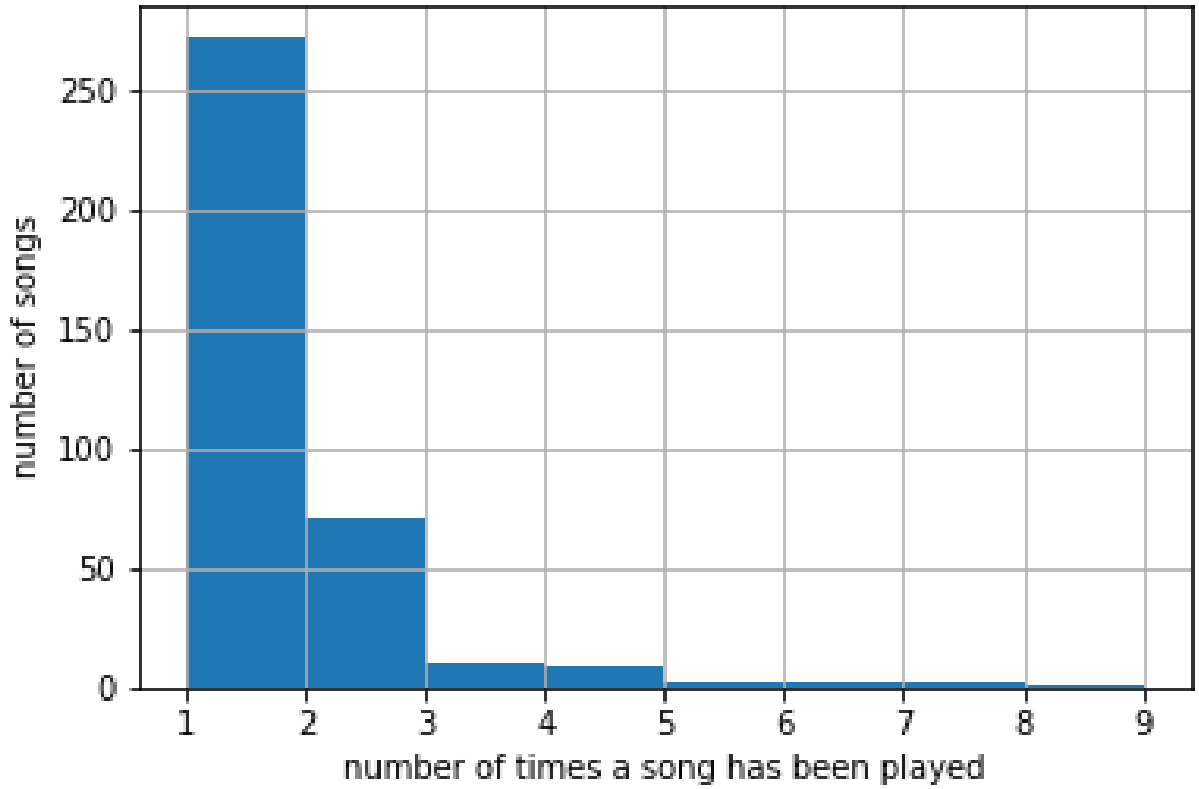
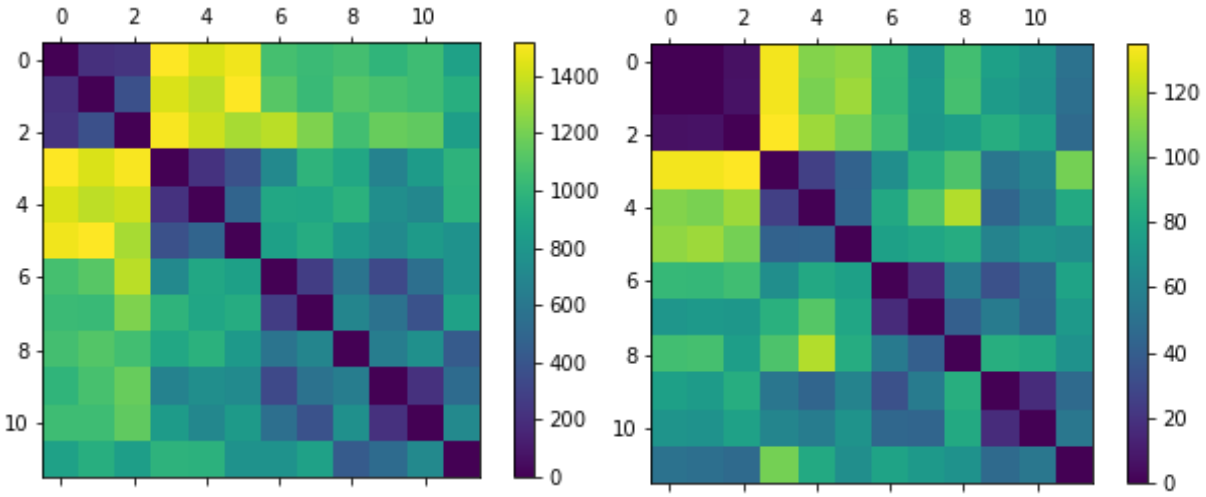


Figure 5.9: Histogram of how many times a song got played

concentration. The first of each group is the raw measure, the second is cleaned by using the ventilation model and the third is cleaned by ignoring decreases. This means the value at e.g. [5,9] represents the mean squared error between measure number 5 which is the humidity measure containing only increases and measure number 9 which is the uncleaned CO2 equivalent measure. The diagonals of the matrices represent a comparison of the measures with themselves and are therefore always 0. The matrices are also symmetric with respect to their diagonal.

There are a number of insights that can be extracted from these matrices. First, there is a cluster of low mean squared error visible at both top left corners, between the first 3 measures corresponding to temperature regardless of the level of bias cleaning. This confirms the discovery made in chapter 4.2.2 that temperature is relatively resistant to the ventilation bias. The result of this fact is that trying to clean the bias by removing datapoints that are predicted to be affected and increasing the threshold on how many samples each song is required to have, has minimal impact on the actual ranking. The rankings for the smaller datasets have even lower mean squared errors, highlighting the fact that averaged out over more samples the effect of the biases is diminished. Second, the mean squared error is significantly lower for the smaller dataset than for the large dataset. This anomaly can be explained by the fact that rankings on the small dataset have less possible combinations than the ranking on the large dataset. Because the number of combinations scales factorially while the number of entries only scales linearly, larger



(a) (b)

Figure 5.10: Matrices of the mean squared error between the measures with the axes corresponding to the measure number from 0 to 12: (a) only including songs that got played at least twice; (b) only including songs that got played at least three times

datasets have on average a much greater mean squared error than smaller ones. The largest discrepancy is between the rankings predicted by the temperature and humidity measures. This effect is mitigated in the smaller dataset, indicating that it might be largely caused by the small sample size per song in the larger dataset. The influence of the bias cleaning attempts is also visible here, as the raw humidity measure still has a high mean squared error when compared to the rankings of the temperature measures but declines sharply once the bias cleaning is applied. It is to be expected that the CO2 equivalent and TVOC concentration measures only have a medium mean squared error between them, as the measures have a correlation of 96% overall. However, the fact that their mean squared error with respect to the temperature measures is actually lower than the mean squared error of humidity with respect to the temperature measures, even when uncleaned. The largest bias that affects CO2 equivalent and TVOC concentration is cigarette smoke and the fact that the rankings according to those measures are still quite similar to the ranking predict by the temperature measure hints at an additional behavior of the crowd. It could be speculated that an increase in crowd mood might also influence the crowd's likelihood to smoke cigarettes, however there is not enough data to confirm this hypothesis.

5.3.3 Conclusion

The fact that the rankings of all three measures on temperature are so similar, especially for the smaller dataset, suggests that the effect of biases on temperature are minimal and can be ignored. However, there is still a large mean squared error in the large dataset present, because of this reason only the small dataset will be used for reliable predictions.

The measures based on humidity heavily contradict the predictions by the temperature measures and bias cleaning has an effect on the ranking that cannot be neglected. As we cannot assume that either the ventilation model or the model that ignores decreases are correct, it will be concluded that humidity is not a reliable measure for indicating crowd mood. CO2 equivalent and TVOC concentration are difficult to assess as a measure for crowd mood. Contrary to expectations their rankings were actually more similar to the rankings based on the temperature measures, which might be due to crowd mood encouraging smoking cigarettes as discussed. However, like humidity the measures are heavily affected by bias cleaning, so without confirmation on the validity of the ventilation model these measures are not reliable. The final measure is therefore defined as

$$\text{crowd mood} = \frac{\text{average difference in temperature per minute}}{\text{crowd size}} \tag{5.1}$$

without assessing the ventilation bias. The crowd mood is undefined for crowd sizes of 0. This measure can now be applied to any song that has been played at least three times to determine the average raise in mood cause by that song. However due to the lack of data only 27 out of 371 can currently be ranked with this measure.

6 Conclusion

This thesis concludes that crowd mood on parties can be measured as the average change in temperature per minute per person in a small, closed room of $15m^2$ with no windows and only one door. It is also evident that the effect of air conditioning on the measure can be mitigated by including three or more separate instances of playing a song in the dataset. This way a DJ may continuously improve their playlist by adding songs with a high crowd mood score and removing songs with a low score once enough data is collected. A linear regression model was built which was able to reliably predict the crowd size based on infrared data with only 4 hours of training data. Attempts to manually build a model to predict ventilation system activity based on average infrared temperature and humidity ultimately failed due to the multitude of factors affecting the model and the resulting inability to prove its validity. More training and testing data is needed to further increase the accuracy of these models.

This thesis can be improved upon by including additional sensors to predict crowd mood. A light barrier can be used to accurately measure crowd size instead of having to rely on a linear regression model based on infrared data. The difficulty here is that the light barrier needs to be installed at the door and is therefore susceptible to getting damaged or destroyed by the crowd. To replace the ventilation activity model a module can be deployed to measure the ventilation system voltage. This way the ventilation bias can be accurately addressed. A ban on smoking indoors would be difficult to enforce but might allow the CO2 equivalent and Total Volatile Organic Compounds (TVOC) concentration measures to be used in calculating crowd mood. On the other hand, a potential relationship between crowd mood and increased smoking could be further investigated.

Further indicators of crowd mood could also include measuring floor vibration to determine dancing activity or deploying audio and noise canceling the music to determine the volume of the crowd. The omnipresence of smartphones could be used to scrape social medias for hashtags and location to determine the posting activity in relation to songs or free Wi-Fi could be offered to monitor the network traffic in a similar manner. Finally electronic payment system could be analyzed to determine the effect of song selection on profits.

Unsupervised learning techniques like clustering or principal component analysis could be used to extract more insights from the data. To automate the process of optimizing song selection on the basis of the crowd mood measure defined, a system for fully autonomous crowd entertainment based on e.g., Reinforcement Learning could be developed.

In conclusion the subject of using sensor data to measure crowd mood and optimize music se-

lection is promising but should be investigated further in future studies. More data is needed to prove the validity of the results of this study.

Bibliography

[Bre01] Leo Breiman. "Random forests". In: *Machine learning* 45.1 (2001), pp. 5–32. DOI: 10.1023/A:1010933404324. URL: https://link.springer.com/article/10.1023/a:1010933404324.

[FS97] Yoav Freund and Robert E Schapire. "A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting". In: *Journal of Computer and System Sciences* 55.1 (1997), pp. 119–139. ISSN: 0022-0000. DOI: 10.1006/jcss.1997.1504. URL: https://www.sciencedirect.com/science/article/pii/S002200009791504X.

[HS18] Milad Haghani and Majid Sarvi. "Crowd behaviour and motion: Empirical methods". In: *Transportation Research Part B: Methodological* 107 (2018), pp. 253–294. ISSN: 0191-2615. DOI: 10.1016/j.trb.2017.06.017. URL: https://www.sciencedirect.com/science/article/pii/S0191261517303788.

[HWZ16] Ting Hu, Qiang Wu, and Ding-Xuan Zhou. "Convergence of Gradient Descent for Minimum Error Entropy Principle in Linear Regression". In: *IEEE Transactions on Signal Processing* 64.24 (2016), pp. 6571–6579. DOI: 10.1109/TSP.2016.2612169. URL: https://ieeexplore.ieee.org/document/7572876.

[Loh11] Wei-Yin Loh. "Classification and regression trees". In: *Wiley interdisciplinary reviews: data mining and knowledge discovery* 1.1 (2011), pp. 14–23. DOI: 10.1002/widm.8. URL: https://wires.onlinelibrary.wiley.com/doi/abs/10.1002/widm.8.

[Ltd] Aosong Electronics Co. Ltd. *Digital-output relative humidity & temperature sensor/module DHT22*. URL: https://www.sparkfun.com/datasheets/Sensors/Temperature/DHT22.pdf.

[NA20] M. Z. Naser and Amir Alavi. "Insights into Performance Fitness and Error Metrics for Machine Learning". In: *CoRR* abs/2006.00887 (2020). DOI: 10.1007/s44150-021-00015-8. arXiv: 2006.00887. URL: https://arxiv.org/abs/2006.00887.

[Pan] Panasonic. *Infrared Array Sensor Grid-EYE (AMG88)*. URL: https://cdn.sparkfun.com/assets/4/1/c/0/1/Grid-EYE_Datasheet.pdf.

[Ped+11] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830. URL: https://www.jmlr.org/papers/volume12/pedregosa11a/pedregosa11a.pdf?ref=https://githubhelp.com.

[scia] scikit-learn. *AdaBoost*. URL: https://scikit-learn.org/stable/modules/ensemble.html#adaboost.

[scib] scikit-learn. *Decision Trees*. URL: https://scikit-learn.org/stable/modules/tree.html#tree.

[scic] scikit-learn. *Regression metrics*. URL: https://scikit-learn.org/stable/modules/model_evaluation.html#r2-score.

[SDP12] Yading Song, Simon Dixon, and Marcus Pearce. "A Survey of Music Recommendation Systems and Future Perspectives". In: (2012). DOI: 10.1.1.414.6614. URL: https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.414.6614&rep=rep1&type=pdf.

[Sen] Sensirion. *Datasheet SGP30 Sensirion Gas Platform*. URL: https://www.mouser.com/pdfdocs/Sensirion_Gas_Sensors_SGP30_Datasheet_EN-1148053.pdf.

[SL12] George AF Seber and Alan J Lee. *Linear regression analysis*. John Wiley & Sons, 2012. URL: https://books.google.com/books?hl=de&lr=&id=X2Y6OkXl8ysC&oi=fnd&pg=PR5&dq=linear+regression+analysis&ots=senQD-qRir&sig=IQ9ac7wOOSTq0EruZJtQOxLSCbY.

[Tha90] Robert E Thayer. *The biopsychology of mood and arousal*. Oxford University Press, 1990. URL: https://books.google.com/books?hl=de&lr=&id=ORiwiDNqcbEC&oi=fnd&pg=PR9&dq=The+Biopsychology+of+Mood+and+Arousal&ots=5c_yA4xZyf&sig=oqCqJdLL0b84Uu1YUIH8QOlGSLw.

[Wak+15] Shoko Wakamiya et al. "Measuring crowd mood in city space through twitter". In: *International Symposium on Web and Wireless Geographical Information Systems*. Springer. 2015, pp. 37–49. DOI: 10.1007/978-3-319-18251-3_3. URL: https://link.springer.com/chapter/10.1007/978-3-319-18251-3_3.

[Wan20] Yu Wang. "A Hybrid Recommendation for Music Based on Reinforcement Learning". In: *Advances in Knowledge Discovery and Data Mining*. Ed. by Hady W. Lauw et al. Cham: Springer International Publishing, 2020, pp. 91–103. ISBN: 978-3-030-47426-3. DOI: 10.1007/978-3-030-47426-3_8. URL: https://link.springer.com/chapter/10.1007/978-3-030-47426-3_8.

[Wol95] Peder Wolkoff. "Volatile organic compounds". In: *Indoor Air, Suppl* 3 (1995), pp. 1–73.

[Zei+09] Kathryn M. Zeitz et al. "Crowd Behavior at Mass Gatherings: A Literature Review". In: *Prehospital and Disaster Medicine* 24.1 (2009), pp. 32–38. DOI: 10.1017/S1049023X00006518. URL: https://www.cambridge.org/core/journals/prehospital-and-disaster-medicine/article/abs/crowd-behavior-at-mass-gatherings-a-literature-review/06478D2F18259B8E2EC0D39FA11937C0.

[Zha+17] Yanhao Zhang et al. "Exploring Coherent Motion Patterns via Structured Trajectory Learning for Crowd Mood Modeling". In: *IEEE Transactions on Circuits and Systems for Video Technology* 27.3 (2017), pp. 635–648. DOI: 10.1109/TCSVT.2016.2593609. URL: https://ieeexplore.ieee.org/document/7517342.

[ZS20] He Zhang and Ravi Srinivasan. "A Systematic Review of Air Quality Sensors, Guidelines, and Measurement Studies for Indoor Air Quality Management". In: *Sustainability* 12.21 (2020). ISSN: 2071-1050. DOI: 10.3390/su12219045. URL: https://www.mdpi.com/2071-1050/12/21/9045.

Appendix

```
import pandas as pd
import numpy as np
import time
import logging
import sys
# Raspberry libraries
import board
import busio
# Device libraries
import adafruit_dht
import adafruit_sgp30
import adafruit_amg88xx

DELAY = 60
SAVE_INTERVAL = 1
FREQUENCY = 100000
MOX_BASELINE = [0x9c61, 0x94e7]

logging.basicConfig(filename='party.log', filemode='a', format='%(
    asctime)s - %(name)s - %(levelname)s - %(message)s')
logger = logging.getLogger()
logger.setLevel(logging.INFO)
logging.info("Delay: {0}s, Save interval: {1}".format(DELAY,
    SAVE_INTERVAL))

columns = ["timestamp", "temperature", "humidity", "eCO2", "TVOC"]
for i in range(64):
  columns.append("inf" + str(i))
try:
  data = pd.read_csv("party.csv", index_col = 0)
  logging.info("party.csv found")
except Exception:
  data = pd.DataFrame(columns=columns, dtype="float64")
  logging.warning("party.csv not found, creating new one...")
count = 0

# DHT22 temperature & humidity sensor
try:
  dhtDevice = adafruit_dht.DHT22(board.D4)
  logging.info("DHT22 initialised")
```

```
39    temperature_on = True
40  except RuntimeError as e:
41    logging.error(e)
42    temperature_on = False
43
44  # SGP30 CO2 equivalent & total volatile organic compound
45  try:
46    i2c = busio.I2C(board.SCL, board.SDA, frequency=FREQUENCY)
47    sgp30 = adafruit_sgp30.Adafruit_SGP30(i2c)
48    sgp30.set_iaq_baseline(MOX_BASELINE[0], MOX_BASELINE[1])
49    sgp30.iaq_init()
50    logging.info("SGP30 initialised")
51    co2_on = True
52  except Exception as e:
53    logging.error(e)
54    co2_on = False
55
56  # AMG8833 8x8 infrared sensor
57  try:
58    i2c = busio.I2C(board.SCL, board.SDA, frequency=FREQUENCY)
59    amg = adafruit_amg88xx.AMG88XX(i2c)
60    logging.info("AMG8833 initialised")
61    infrared_on = True
62  except Exception as e:
63    logging.error(e)
64    infrared_on = False
65
66  # Main loop
67  while True:
68    try:
69      datapoint = pd.Series(index = columns, dtype="float64")
70      datapoint["timestamp"] = pd.Timestamp.now()
71      # temperature + humidity
72      if temperature_on:
73        try:
74          datapoint["temperature"] = dhtDevice.temperature
75          datapoint["humidity"] = dhtDevice.humidity
76        except RuntimeError:
77          datapoint["temperature"] = np.nan
78          datapoint["humidity"] = np.nan
79      else:
80        datapoint["temperature"] = np.nan
81        datapoint["humidity"] = np.nan
82        # DHT22 temperature & humidity sensor
83        try:
84          dhtDevice = adafruit_dht.DHT22(board.D4)
85          logging.info("DHT22 initialised")
```

```
86          temperature_on = True
87        except Exception as e:
88          logging.error(e)
89          temperature_on = False
90
91      # CO2 equivalent + total volatile organic compounds
92      if co2_on:
93        try:
94          datapoint["eCO2"] = sgp30.eCO2
95          datapoint["TVOC"] = sgp30.TVOC
96        except Exception:
97          datapoint["eCO2"] = np.nan
98          datapoint["TVOC"] = np.nan
99      else:
100       datapoint["eCO2"] = np.nan
101       datapoint["TVOC"] = np.nan
102       # SGP30 CO2 equivalent & total volatile organic compound
103       try:
104         i2c = busio.I2C(board.SCL, board.SDA, frequency=FREQUENCY)
105         sgp30 = adafruit_sgp30.Adafruit_SGP30(i2c)
106         sgp30.iaq_init()
107         logging.info("SGP30 initialised")
108         co2_on = True
109       except Exception as e:
110         logging.error(e)
111         co2_on = False
112
113     # 8x8 infrared picture
114     if infrared_on:
115       i = 0
116       for row in amg.pixels:
117         for temp in row:
118           temp_column = "inf" + str(i)
119           assert temp_column in columns
120           i += 1
121           try:
122             datapoint[temp_column] = temp
123           except Exception:
124             datapoint[temp_column] = np.nan
125     else:
126       # AMG8833 8x8 infrared sensor
127       try:
128         i2c = busio.I2C(board.SCL, board.SDA, frequency=FREQUENCY)
129         amg = adafruit_amg88xx.AMG88XX(i2c)
130         logging.info("AMG8833 initialised")
131         infrared_on = True
132       except Exception as e:
```

```
133         logging.error(e)
134         infrared_on = False
135 
136     logger.info(datapoint[["timestamp", "temperature", "humidity", "
            eCO2", "TVOC", "inf0"]])
137     data = data.append(datapoint, ignore_index=True)
138     count += 1
139     if count >= SAVE_INTERVAL:
140       data.to_csv("party.csv")
141       count = 0
142     time.sleep(DELAY)
143 
144   except KeyboardInterrupt:
145     data.to_csv("party.csv")
146     logging.warning("Keyboard␣interrupt")
147     break
```

Listing 1: Program for data collection run on the Raspberry Pi

```
1  #!/usr/bin/env python
2  # coding: utf-8
3 
4  # In[1]:
5 
6 
7  import pandas as pd
8  import matplotlib.pyplot as plt
9  import matplotlib.dates as md
10 import math
11 import numpy as np
12 from sklearn.linear_model import LinearRegression
13 from sklearn.ensemble import RandomForestRegressor, AdaBoostRegressor
14 from sklearn.tree import DecisionTreeRegressor
15 from sklearn.metrics import accuracy_score, mean_squared_error
16 from sklearn.model_selection import train_test_split, GridSearchCV,
       KFold
17 from sklearn.preprocessing import PolynomialFeatures
18 import seaborn as sns
19 
20 
21 # In[2]:
22 
23 
24 data1 = pd.read_csv("party1.csv", index_col=0)
25 data2 = pd.read_csv("party.csv",index_col=0)
26 songs = pd.read_csv("songs.csv")
27 songs_people = pd.read_csv("songs_people.csv")
```

```
28
29
30 # In[3]:
31
32
33 data = data1.append(data2, ignore_index=True)
34
35
36 # In[4]:
37
38
39 data["timestamp"] = pd.to_datetime(data["timestamp"])
40
41
42 # In[5]:
43
44
45 data.head()
46
47
48 # In[6]:
49
50
51 data.info()
52
53
54 # Cleaning faulty timestamps caused by sensor shutdowns
55
56 # In[7]:
57
58
59 time_errors = data[data["timestamp"].diff() < pd.Timedelta("0")].
       index
60 time_errors
61
62
63 # In[8]:
64
65
66 data.loc[time_errors[0], "timestamp"] = pd.to_datetime('2022-05-09␣
       17:33:00')
67 data.loc[time_errors[0]+1, "timestamp"] = pd.to_datetime('2022-05-09␣
       17:34:00')
68
69
70 # In[9]:
71
```

```
72
73 for i in range(10):
74     data.loc[314+i, "timestamp"] = pd.to_datetime('2022-05-11␣
           10:{}:00'.format(30+i))
75
76
77 # In[10]:
78
79
80 for i in range(8):
81     temp = data.loc[314+i].copy()
82     data.loc[314+i] = data.loc[324+i]
83     data.loc[324+i] = temp
84
85
86 # In[11]:
87
88
89 for i in range(4):
90     temp1 = data.loc[322+2*i].copy()
91     temp2 = data.loc[323+2*i].copy()
92     data.loc[322+2*i] = data.loc[324+2*i]
93     data.loc[323+2*i] = data.loc[325+2*i]
94     data.loc[324+2*i] = temp1
95     data.loc[325+2*i] = temp2
96
97
98 # In[12]:
99
100
101 data.loc[335, "timestamp"] = pd.to_datetime('2022-05-11␣10:44:24')
102
103
104 # In[13]:
105
106
107 for i in range(3):
108     data.loc[343+i, "timestamp"] = pd.to_datetime("2022-05-23␣
           15:{}:00".format(45+i))
109
110
111 # In[14]:
112
113
114 data.loc[383, "timestamp"] = pd.to_datetime('2022-05-24␣15:09:00')
115
116
```

```
# In[15]:


data.loc[1671, "timestamp"] = pd.to_datetime('2022-05-25 12:56:00')


# In[16]:


data.loc[2253, "timestamp"] = pd.to_datetime('2022-06-11 23:31:00')


# In[17]:


for i in range(81):
    h = 19 + math.floor((i+25)/60)
    m = (i+25)%60
    data.loc[2858+i, "timestamp"] = pd.to_datetime("2022-06-16 
        {}:{}:00".format(h, m))


# In[18]:


data.loc[3764, "timestamp"] = pd.to_datetime("2022-06-18 15:40:00")


# In[19]:


for i in range(197):
    h = math.floor((i+47)/60)
    m = (i+47)%60
    data.loc[4003+i, "timestamp"] = pd.to_datetime("2022-06-19 
        {}:{}:00".format(h, m))


# In[20]:


for i in range(5):
    data.loc[time_errors[12]+i, "timestamp"] = pd.to_datetime("
        2022-07-30 21:{}:00".format(9+i))


# Investigating large gaps in time between datapoints
```

```
161
162 # In[21]:
163
164
165 time_skips = data[data["timestamp"].diff() > pd.Timedelta("01:00:00")
        ].index
166 time_skips
167
168
169 # In[22]:
170
171
172 skip_errors = [2, 5,10, 11, 14, 27, 18]
173
174
175 # In[23]:
176
177
178 data.loc[114, "timestamp"] = pd.to_datetime("2022-05-07␣11:25:41")
179
180
181 # In[24]:
182
183
184 for i, idx in enumerate([375,376]):
185     data.loc[idx, "timestamp"] = pd.to_datetime("2022-05-24␣14:5{}:24
            ".format(i+3))
186
187
188 # In[25]:
189
190
191 for i in range(2):
192     data.loc[4206+i, "timestamp"] = pd.to_datetime("2022-07-23␣
            21:15:{}3".format(1+i))
193
194
195 # In[26]:
196
197
198 for i in range(4):
199     data.loc[2108+i, "timestamp"] = pd.to_datetime("2022-06-11␣
            21:0{}:46".format(3+i))
200
201
202 # In[27]:
203
```

```
204
205 data.iloc[314:322, 1:] = np.nan
206
207
208 # In[28]:
209
210
211 assert (data["timestamp"].diff()[1:] >= pd.Timedelta("0")).all()
212
213
214 # Cleaning erroneous values
215
216 # In[29]:
217
218
219 data["temperature"].hist();
220
221
222 # In[30]:
223
224
225 data[data["temperature"] < 10]["temperature"].unique()
226
227
228 # In[31]:
229
230
231 data.loc[data["temperature"] == -50, "temperature"] = np.nan
232
233
234 # In[32]:
235
236
237 data["temperature"].hist();
238
239
240 # In[33]:
241
242
243 data["humidity"].hist();
244
245
246 # In[34]:
247
248
249 data[data["humidity"] < 30]["humidity"].unique()
250
```

```
251
252 # In[35]:
253
254
255 data.loc[data["humidity"] == 1, "humidity"] = np.nan
256
257
258 # In[36]:
259
260
261 data["humidity"].hist();
262
263
264 # In[37]:
265
266
267 data["eCO2"].hist();
268
269
270 # Defining columns used for statistics
271
272 # In[38]:
273
274
275 data["inf_mean"] = data.loc[:,data.columns[5:69]].mean(axis=1)
276 data["timestamp_diff"] = data["timestamp"].diff()
277 data["inf_mean_diff"] = data["inf_mean"].diff()
278 data["humidity_diff"] = data["humidity"].diff()
279 data["TVOC_diff"] = data["TVOC"].diff()
280 data["eCO2_diff"] = data["eCO2"].diff()
281 data["temp_diff"] = data["temperature"].diff()
282 data["humidity_diff_shift"] = data["humidity_diff"].shift(-1)
283
284
285 # In[39]:
286
287
288 trans_day = data[data["timestamp"].diff() > pd.Timedelta("00:01:02")
        ].index
289 len(trans_day)
290
291
292 # In[40]:
293
294
295 data.loc[trans_day, "inf_mean_diff"] = np.nan
296
```

```
297
298 # In[41]:
299
300
301 data["date"] = data["timestamp"].apply(lambda x: x.floor("d"))
302 data["hour"] = data["timestamp"].apply(lambda x: x.hour)
303 data["hour_date"] = data["timestamp"].apply(lambda x: x.floor("h"))
304
305
306 # Cleaning song data
307
308 # In[42]:
309
310
311 songs.head()
312
313
314 # In[43]:
315
316
317 songs["timestamp"] = pd.to_datetime(songs["timestamp"], dayfirst=True
        )
318 songs.dtypes
319
320
321 # In[44]:
322
323
324 songs_people.sample(10)
325
326
327 # In[45]:
328
329
330 songs_people["timestamp"] = pd.to_datetime(songs_people["timestamp"],
         dayfirst=True)
331 songs_people.dtypes
332
333
334 # In[46]:
335
336
337 songs = songs.merge(songs_people.drop("song", axis=1), on="timestamp"
        , how="left")
338
339
340 # In[47]:
```

```
341
342
343 songs.head()
344
345
346 # In[48]:
347
348
349 songs.groupby("song").count().sort_values("timestamp", ascending=
        False).head(10)
350
351
352 # In[49]:
353
354
355 # 'Drag' is not a song but a bug in the script, probably caused when
        Spotify is loading new songs
356 songs = songs[songs["song"] != 'Drag']
357
358
359 # In[50]:
360
361
362 songs["end"] = songs["timestamp"].shift(-1)
363 songs["length"] = songs["end"] - songs["timestamp"]
364
365
366 # In[51]:
367
368
369 songs.head(2)
370
371
372 # In[52]:
373
374
375 songs.sort_values("length", ascending=False).head(20)
376 # Length > 1 day are just the time between partys, length > 10
        minutes indicate missing date, e.g. the script stopped working
377
378
379 # In[53]:
380
381
382 #All lengths above 10 minutes are faulty
383 songs.loc[(songs["length"] > pd.Timedelta("00:10:00")) | (songs["
        length"] < pd.Timedelta("00:00:00")), "length"] = np.nan
```

```
384
385
386 # In[54]:
387
388
389 # We could fetch the song duration from the Spotify API, however this
        is not reliable as the song could have been skipped halfway.
390 # In fact precisely the fact that this data is faulty indicates that
       the laptop might have crashed
391 # We will just assume the song to have average length
392 avg_length = songs["length"].mean().floor('S')
393 avg_length
394
395
396 # In[55]:
397
398
399 songs.loc[songs["length"].isna(), "length"] = avg_length
400
401
402 # In[56]:
403
404
405 songs.rename({"timestamp": "start"}, axis=1, inplace=True)
406
407
408 # In[57]:
409
410
411 songs[songs["end"].isna()]
412
413
414 # In[58]:
415
416
417 songs["end"] = songs["start"] + songs["length"]
418
419
420 # In[59]:
421
422
423 songs.reset_index(drop=True, inplace=True)
424
425
426 # In[60]:
427
428
```

```
songs.head()


# Merging song data and sensor data

# In[61]:


for song in songs.values:
    # song: ["start", "song_name", "num_people", "end", "length"]
    index = np.where((data["timestamp"] >= song[0]) & (data["
        timestamp"] < song[3]))
    for i in index:
        data.loc[i, "song"] = song[1]
        data.loc[i, "num_people"] = song[2]


# In[62]:


data["tdiff"] = data["temperature"].diff()
data.loc[trans_day, "tdiff"] = np.nan


# In[63]:


limits = np.zeros((8,8))
limits[6:8] = 0.5
limits[6, 0:6] = 1
limits[5, 0:7] = 1
limits[4, 1:6] = 1
limits[3, 2:5] = 1
plt.matshow(limits);


# In[64]:


# x=7-6: standing behind bar
# x=6-5: standing at bar
# x=4: standing in the middle of the room
# x=4-3: standing at the back of the room


# In[65]:

```

```
475
476 floor_pixels = list()
477 for x in range(8):
478     for y in range(8):
479         if limits[x,y] > 0:
480             floor_pixels.append("inf{}".format(y+x*8))
481
482
483 # In[66]:
484
485
486 ceiling_pixels = data.columns[5:69]
487 ceiling_pixels = ceiling_pixels.drop(floor_pixels)
488 ceiling_pixels
489
490
491 # In[67]:
492
493
494 data["minute"] = data["timestamp"].apply(lambda x: x.floor('T'))
495 songs["date"] = songs["start"].apply(lambda x: x.floor("D"))
496 data["inf_std"] = data.iloc[:, 5:69].std(axis=1)
497 data["inf_std_floor"] = data.loc[:, floor_pixels].std(axis=1)
498
499
500 # Building models to predict crowd size
501
502 # In[68]:
503
504
505 data_inf = data[data["inf0"].notna()]
506
507
508 # In[69]:
509
510
511 # This is the data of test 3
512 data.loc[4440:4790, "num_people"] = 0
513 data.loc[(data["timestamp"] >= '2022-07-29␣17:40:09') & (data["
        timestamp"] <='2022-07-29␣17:41:59'), "num_people"] = 1
514 data.loc[(data["timestamp"] >= '2022-07-29␣17:48:50') & (data["
        timestamp"] <='2022-07-29␣17:50:00'), "num_people"] = 1
515 data.loc[(data["timestamp"] >= '2022-07-29␣17:51:53') & (data["
        timestamp"] <='2022-07-29␣17:53:00'), "num_people"] = 1
516
517
518 # In[70]:
```

```
519
520
521 party_people = data[data["num_people"].notna()]
522
523
524 # In[71]:
525
526
527 party_people["inf_std"].corr(party_people["num_people"])
528
529
530 # In[72]:
531
532
533 sns.regplot(x=party_people["inf_std"], y=party_people["num_people"])
534 plt.ylabel("crowd␣size")
535 plt.xlabel("infrared␣standard␣deviation");
536
537
538 # In[73]:
539
540
541 sns.regplot(x=party_people["humidity"], y=party_people["num_people"])
        ;
542
543
544 # In[74]:
545
546
547 plt.plot(party_people["num_people"]);
548
549
550 # In[75]:
551
552
553 X_train_std, X_test_std, y_train_std, y_test_std = train_test_split(
        party_people["inf_std"].values.reshape(-1,1), party_people["
        num_people"].values, test_size=0.2)
554 X_train_std_floor, X_test_std_floor, y_train_std_floor,
        y_test_std_floor = train_test_split(party_people["inf_std_floor"
        ].values.reshape(-1,1), party_people["num_people"].values,
        test_size=0.2)
555 X_train_hum, X_test_hum, y_train_hum, y_test_hum = train_test_split(
        party_people[party_people["humidity"].notna()]["humidity"].values
        .reshape(-1,1), party_people[party_people["humidity"].notna()]["
        num_people"].values, test_size=0.2)
556
```

```
557
558 # In[76]:
559
560
561 reg_mod_std = LinearRegression().fit(X_train_std, y_train_std)
562 reg_mod_std_floor = LinearRegression().fit(X_train_std_floor,
        y_train_std_floor)
563 reg_mod_hum = LinearRegression().fit(X_train_hum, y_train_hum)
564
565
566 # In[77]:
567
568
569 tree_mod = DecisionTreeRegressor()
570 forest_mod = RandomForestRegressor()
571 ada_mod = AdaBoostRegressor()
572 reg_mod = LinearRegression()
573
574
575 # In[78]:
576
577
578 party_people.count().max()
579
580
581 # In[79]:
582
583
584 X_train, X_test, y_train, y_test = train_test_split(party_people.iloc
        [:,5:69], party_people["num_people"], test_size = 0.2)
585 X_test, X_cross, y_test, y_cross = train_test_split(X_test, y_test,
        test_size=0.5)
586
587
588 # In[80]:
589
590
591 tree_mod.fit(X_train, y_train)
592 forest_mod.fit(X_train, y_train)
593 ada_mod.fit(X_train, y_train)
594 reg_mod.fit(X_train, y_train);
595
596
597 # In[81]:
598
599
600 from sklearn.metrics import r2_score
```

```
601 labels = ["decision␣tree", "random␣forest", "AdaBoost", "Linear␣
        regression"]
602 models = [tree_mod, forest_mod, ada_mod, reg_mod]
603 for label, mod in zip(labels, models):
604     pred = mod.predict(X_cross)
605     print(label, mean_squared_error(y_cross, pred), r2_score(y_cross,
            pred))
606
607
608 # In[82]:
609
610
611 print(mean_squared_error(y_test_std, reg_mod_std.predict(X_test_std))
        )
612 print(mean_squared_error(y_test_std_floor, reg_mod_std_floor.predict(
        X_test_std_floor)))
613 print(mean_squared_error(y_test_hum, reg_mod_hum.predict(X_test_hum))
        )
614
615
616 # Model to predict ventilation activity
617
618 # In[83]:
619
620
621 ven = 0
622 pred_ven = list()
623 for i in range(data.shape[0]):
624     idx = data.index[i]
625     time_diff = data["timestamp"].diff().loc[idx]
626     if pd.isnull(time_diff) or time_diff > pd.Timedelta("01:00:00"):
627         ven = 0
628     elif data.loc[idx]["inf_mean_diff"] < -1.28 and (data.loc[idx:idx
            +4]["humidity_diff"] <= -1.19).any():
629         ven = 1
630     elif data.loc[idx]["inf_mean_diff"] > 0.77 and (data.loc[idx:idx
            +5]["humidity_diff"] >= 0.2).any():
631         ven = 0
632     pred_ven.append(ven)
633 data["ventilation_pred"] = pred_ven
634
635
636 # In[84]:
637
638
639 # Crowd size model predictions
```

```
data["num_people_pred_std"] = data_inf["inf_std"].apply(lambda x:
    math.floor(reg_mod_std.predict([[x]])[0]))
data["num_people_pred"] = data_inf.iloc[:,5:69].apply(lambda x: math.
    floor(reg_mod.predict(x.values.reshape(1,-1))[0]), axis=1)
data["num_people_pred_rf"] = data_inf.iloc[:,5:69].apply(lambda x:
    tree_mod.predict(x.values.reshape(1,-1))[0], axis=1)
data["num_people_pred_std_floor"] = data_inf["inf_std_floor"].apply(
    lambda x: math.floor(reg_mod_std_floor.predict([[x]])[0]))
data["num_people_pred_hum"] = data[data["humidity"].notna()]["
    humidity"].apply(lambda x: reg_mod_hum.predict([[x]])[0])

# Negative predictions will be set to 0
data.loc[data["num_people_pred"] < 0, "num_people_pred"] = 0
data.loc[data["num_people_pred_std"] < 0, "num_people_pred_std"] = 0
data.loc[data["num_people_pred_std_floor"] < 0, "
    num_people_pred_std_floor"] = 0
data.loc[data["num_people_pred_hum"] < 0, "num_people_pred_hum"] = 0

data["num_people_pred_std_comb"] = data[["num_people_pred_std", "
    num_people_pred_std_floor"]].min(axis=1)
data["num_people_pred_std1"] = data["num_people_pred_std"]
data.loc[(data["num_people_pred_std"] > 0) & (data[floor_pixels].mean
    (axis=1) <= data[ceiling_pixels].mean(axis=1)), "
    num_people_pred_std1"] = np.nan
data["num_people_pred_std2"] = data["num_people_pred_std1"]
data.loc[data["num_people_pred_std2"].isna(), "num_people_pred_std2"]
     = data[data["num_people_pred_std2"].isna()]["num_people_pred"]
data.loc[data.query("(timestamp >= '2022-06-12 05:00:00') & (
    timestamp < '2022-06-13 00:00:00')").index, "num_people_pred_std2
    "] = 0
data.loc[data.query("(timestamp >= '2022-06-17 08:00:00') & (
    timestamp < '2022-06-18 00:00:00')").index, "num_people_pred_std2
    "] = 0

# Defining views for every party and every test
test1 = data.query("(timestamp < '2022-05-10 00:00:00') & (timestamp
    >= '2022-05-09 00:00:00')")
test2 = data.query("(timestamp >= '2022-05-24 00:00:00') & (timestamp
    < '2022-05-26 00:00:00')")
party1 = data.query("(timestamp >= '2022-05-27 00:00:00') & (
    timestamp < '2022-05-29 00:00:00')")
party2 = data.query("(timestamp >= '2022-06-11 00:00:00') & (
    timestamp < '2022-06-13 00:00:00')")
party3 = data.query("(timestamp >= '2022-06-16 00:00:00') & (
    timestamp < '2022-06-18 00:00:00')")
party4 = data.query("(timestamp >= '2022-06-18 22:00:00') & (
    timestamp < '2022-06-20 00:00:00')")
```

```
party5 = data.query("(timestamp >= '2022-07-23 00:00:00') & (timestamp < '2022-07-24 00:00:00')")
test3 = data.query("(timestamp >= '2022-07-29 00:00:00') & (timestamp < '2022-07-29 20:00:00')")
party6 = data.query("(timestamp >= '2022-07-29 20:00:00') & (timestamp < '2022-07-30 07:00:00')")
party_people = data[data["num_people"].notna()]


# Investigating dataset

# In[85]:


xfmt = md.DateFormatter('%H:%M')


# In[86]:


fig, ax = plt.subplots(2,3, figsize=(16,8))
plt.subplots_adjust(hspace=0.3)
plt.suptitle("party1 {}".format(party1.date.min().date()))
barchart = party1.groupby("song").count()["timestamp"].sort_values(ascending=False).head(12)
for a in ax.flatten():
    a.xaxis.set_major_formatter(xfmt)
    a.tick_params(labelrotation=20);
ax[0,0].plot(party1.set_index("timestamp")["temperature"]);
ax[0,0].set_title("Temperature");
ax[0,1].plot(party1.set_index("timestamp")["humidity"]);
ax[0,1].set_title("Humidity");
ax[0,2].plot(party1.set_index("timestamp")["eCO2"]);
ax[0,2].set_title("CO2 Equivalent");
ax[1,0].plot(party1.set_index("timestamp")["TVOC"]);
ax[1,0].set_title("Total volatile organic compounds");
ax[1,1].plot(party1.set_index("timestamp")["inf_mean"]);
ax[1,1].set_title("Average infrared temperature");


# In[87]:


fig, ax = plt.subplots(2,3, figsize=(16,8))
plt.subplots_adjust(hspace=0.3)
plt.suptitle("party2 {}".format(party2.date.min().date()))
```

```
barchart = party2.groupby("song").count()["timestamp"].sort_values(
    ascending=False).head(12)
for a in ax.flatten():
    a.xaxis.set_major_formatter(xfmt)
    a.tick_params(labelrotation=20);
ax[0,0].plot(party2.set_index("timestamp")["temperature"]);
ax[0,0].set_title("Temperature");
ax[0,1].plot(party2.set_index("timestamp")["humidity"]);
ax[0,1].set_title("Humidity");
ax[0,2].plot(party2.set_index("timestamp")["eCO2"]);
ax[0,2].set_title("CO2 Equivalent");
ax[1,0].plot(party2.set_index("timestamp")["TVOC"]);
ax[1,0].set_title("Total volatile organic compounds");
ax[1,1].plot(party2.set_index("timestamp")["inf_mean"]);
ax[1,1].set_title("Average infrared temperature");


# In[88]:


fig, ax = plt.subplots(2,3, figsize=(16,8))
plt.subplots_adjust(hspace=0.3)
plt.suptitle("party3 {}".format(party3.date.min().date()))
barchart = party3.groupby("song").count()["timestamp"].sort_values(
    ascending=False).head(12)
for a in ax.flatten():
    a.xaxis.set_major_formatter(xfmt)
    a.tick_params(labelrotation=20);
ax[0,0].plot(party3.set_index("timestamp")["temperature"]);
ax[0,0].set_title("Temperature");
ax[0,1].plot(party3.set_index("timestamp")["humidity"]);
ax[0,1].set_title("Humidity");
ax[0,2].plot(party3.set_index("timestamp")["eCO2"]);
ax[0,2].set_title("CO2 Equivalent");
ax[1,0].plot(party3.set_index("timestamp")["TVOC"]);
ax[1,0].set_title("Total volatile organic compounds");
ax[1,1].plot(party3.set_index("timestamp")["inf_mean"]);
ax[1,1].set_title("Average infrared temperature");


# In[89]:


fig, ax = plt.subplots(2,3, figsize=(16,8))
plt.subplots_adjust(hspace=0.3)
plt.suptitle("party4 {}".format(party4.date.min().date()))
```

```
barchart = party4.groupby("song").count()["timestamp"].sort_values(
    ascending=False).head(12)
for a in ax.flatten():
    a.xaxis.set_major_formatter(xfmt)
    a.tick_params(labelrotation=20);
ax[0,0].plot(party4.set_index("timestamp")["temperature"]);
ax[0,0].set_title("Temperature");
ax[0,1].plot(party4.set_index("timestamp")["humidity"]);
ax[0,1].set_title("Humidity");
ax[0,2].plot(party4.set_index("timestamp")["eCO2"]);
ax[0,2].set_title("CO2 Equivalent");
ax[1,0].plot(party4.set_index("timestamp")["TVOC"]);
ax[1,0].set_title("Total volatile organic compounds");
ax[1,1].plot(party4.set_index("timestamp")["inf_mean"]);
ax[1,1].set_title("Average infrared temperature");


# In[90]:


fig, ax = plt.subplots(2,3, figsize=(16,8))
plt.subplots_adjust(hspace=0.3)
plt.suptitle("party5 {}".format(party5.date.min().date()))
barchart = party5.groupby("song").count()["timestamp"].sort_values(
    ascending=False).head(12)
for a in ax.flatten():
    a.xaxis.set_major_formatter(xfmt)
    a.tick_params(labelrotation=20);
ax[0,0].plot(party5.set_index("timestamp")["temperature"]);
ax[0,0].set_title("Temperature");
ax[0,1].plot(party5.set_index("timestamp")["humidity"]);
ax[0,1].set_title("Humidity");
ax[0,2].plot(party5.set_index("timestamp")["eCO2"]);
ax[0,2].set_title("CO2 Equivalent");
ax[1,0].plot(party5.set_index("timestamp")["TVOC"]);
ax[1,0].set_title("Total volatile organic compounds");
ax[1,1].plot(party5.set_index("timestamp")["inf_mean"]);
ax[1,1].set_title("Average infrared temperature");


# In[91]:


fig, ax = plt.subplots(2,3, figsize=(16,8))
plt.subplots_adjust(hspace=0.3)
plt.suptitle("party6 {}".format(party5.date.min().date()))
```

```
barchart = party6.groupby("song").count()["timestamp"].sort_values(
    ascending=False).head(12)
for a in ax.flatten():
    a.xaxis.set_major_formatter(xfmt)
    a.tick_params(labelrotation=20);
ax[0,0].plot(party6.set_index("timestamp")["temperature"]);
ax[0,0].set_title("Temperature");
ax[0,1].plot(party6.set_index("timestamp")["humidity"]);
ax[0,1].set_title("Humidity");
ax[0,2].plot(party6.set_index("timestamp")["eCO2"]);
ax[0,2].set_title("CO2 Equivalent");
ax[1,0].plot(party6.set_index("timestamp")["TVOC"]);
ax[1,0].set_title("Total volatile organic compounds");
ax[1,1].plot(party6.set_index("timestamp")["inf_mean"]);
ax[1,1].set_title("Average infrared temperature");


# In[92]:


fig, ax = plt.subplots(2,3, figsize=(16,8))
plt.subplots_adjust(hspace=0.3)
plt.suptitle("test1")
for a in ax.flatten():
    a.xaxis.set_major_formatter(xfmt)
    a.tick_params(labelrotation=20);
ax[0,0].plot(test1.set_index("timestamp")["temperature"]);
ax[0,0].set_title("Temperature");
ax[0,1].plot(test1.set_index("timestamp")["humidity"]);
ax[0,1].set_title("Humidity");
ax[0,2].plot(test1.set_index("timestamp")["eCO2"]);
ax[0,2].set_title("CO2 Equivalent");
ax[1,0].plot(test1.set_index("timestamp")["TVOC"]);
ax[1,0].set_title("Total volatile organic compounds");
ax[1,1].plot(test1.set_index("timestamp")["inf_mean"]);
ax[1,1].set_title("Average infrared temperature");


# In[93]:


fig, ax = plt.subplots(2,3, figsize=(16,8))
plt.subplots_adjust(hspace=0.3)
plt.suptitle("test2")
for a in ax.flatten():
    a.xaxis.set_major_formatter(xfmt)
    a.tick_params(labelrotation=20);
```

```
ax[0,0].plot(test2.set_index("timestamp")["temperature"]);
ax[0,0].set_title("Temperature");
ax[0,1].plot(test2.set_index("timestamp")["humidity"]);
ax[0,1].set_title("Humidity");
ax[0,2].plot(test2.set_index("timestamp")["eCO2"]);
ax[0,2].set_title("CO2 Equivalent");
ax[1,0].plot(test2.set_index("timestamp")["TVOC"]);
ax[1,0].set_title("Total volatile organic compounds");
ax[1,1].plot(test2.set_index("timestamp")["inf_mean"]);
ax[1,1].set_title("Average infrared temperature");


# In[94]:


fig, ax = plt.subplots(2,3, figsize=(16,8))
plt.subplots_adjust(hspace=0.3)
plt.suptitle("test3")
for a in ax.flatten():
    a.xaxis.set_major_formatter(xfmt)
    a.tick_params(labelrotation=20);
ax[0,0].plot(test3.set_index("timestamp")["temperature"]);
ax[0,0].set_title("Temperature");
ax[0,1].plot(test3.set_index("timestamp")["humidity"]);
ax[0,1].set_title("Humidity");
ax[0,2].plot(test3.set_index("timestamp")["eCO2"]);
ax[0,2].set_title("CO2 Equivalent");
ax[1,0].plot(test3.set_index("timestamp")["TVOC"]);
ax[1,0].set_title("Total volatile organic compounds");
ax[1,1].plot(test3.set_index("timestamp")["inf_mean"]);
ax[1,1].set_title("Average infrared temperature");


# Investigating the ventilation activity model

# In[95]:


fig, ax = plt.subplots()
party6_time = party6.set_index("timestamp")
ax.plot(party6_time["humidity"])
ax.set_ylim([50,80])
ax.fill_between(party6_time.index, 0, 1, where=party6_time["
    ventilation_pred"], alpha=0.2, transform=ax.get_xaxis_transform()
    , color="blue")
ax.xaxis.set_major_formatter(xfmt)
ax.set_ylabel("humidity in %")
```

```
ax2 = ax.twinx()
ax2.plot(party6_time["inf_mean"], color="orange")
ax2.set_ylim([data["inf_mean"].min(), data["inf_mean"].max()])
ax2.set_ylabel("average infrared temperature in C")
ax2.xaxis.set_major_formatter(xfmt);


# In[96]:


fig, ax = plt.subplots()
party2_trim = party2.query("timestamp < '2022-06-12 04:00:00'").
    set_index("timestamp")
ax.plot(party2_trim["humidity"])
ax.set_ylim([40,70])
ax.fill_between(party2_trim.index, 0, 1, where=party2_trim["
    ventilation_pred"], alpha=0.2, transform=ax.get_xaxis_transform()
    , color="blue")
ax.xaxis.set_major_formatter(xfmt)
ax.set_ylabel("humidity in %")
ax2 = ax.twinx()
ax2.plot(party2_trim["inf_mean"], color="orange")
ax2.set_ylim([data["inf_mean"].min(), data["inf_mean"].max()])
ax2.set_ylabel("average infrared temperature in C")
ax2.xaxis.set_major_formatter(xfmt);


# In[97]:


fig, ax = plt.subplots(3,3, figsize=(20,15))
plt.subplots_adjust(wspace=0.3)
labels=["party 1", "party 2", "party 3", "party 4", "party 5", "party
    6", "test 1", "test 2", "test 3"]
for index, party in enumerate([party1, party2, party3, party4, party5
    , party6, test1, test2, test3]):
    x, y = math.floor(index/3), index%3
    ax[x,y].plot(party.set_index("timestamp")["humidity"])
    ax[x,y].fill_between(party.set_index("timestamp").index, 0, 1,
        where=party.set_index("timestamp")["ventilation_pred"], alpha
        =0.2, transform=ax[x,y].get_xaxis_transform(), color="blue")
    #ax[x,y].plot(party.set_index("timestamp")["num_people_pred_std1
        "])
    #ax[x,y].plot(party.set_index("timestamp")["num_people_pred"])
    ax[x,y].set_ylim([data["humidity"].min(),data["humidity"].max()])
    ax[x,y].xaxis.set_major_formatter(xfmt)
    ax[x,y].set_xlabel("time of day")
```

```
927     ax[x,y].set_title(labels[index])
928     ax[x,y].set_ylabel("humidity␣in␣%")
929     ax2 = ax[x,y].twinx()
930     ax2.plot(party.set_index("timestamp")["inf_mean"], color="orange"
            )
931     ax2.set_ylim([data["inf_mean"].min(), data["inf_mean"].max()])
932     ax2.set_ylabel("average␣infrared␣temperature␣in␣C")
933     ax2.xaxis.set_major_formatter(xfmt);
934
935
936 # Looking for correlation in the dataset
937
938 # In[98]:
939
940
941 data["eCO2diff"] = data["eCO2"].diff()
942
943
944 # In[99]:
945
946
947 data.loc[data["eCO2diff"] >= 0, "eCO2diff"] = np.nan
948
949
950 # In[100]:
951
952
953 data.corr()
954
955
956 # In[101]:
957
958
959 data.corr().to_csv("data_corr.csv")
960
961
962 # eCO2 and TVOC are strongly correlated 96%
963 # temperature is strongly correlated with infrared data 90% with a
        cyclic behaivor
964 #
965 # eCO2 and TVOC are weakly correlated to temperature, humidity and
        infrared data 40%
966 # humidity is weakly correlated with temperature (anti), CO2 and TVOC
         35%
967 # hour is weakly anti correlated with eCO2 and TVOC
968 #
969 # num_people is not correlated with anything
```

```
970
971 # In[103]:
972
973
974 party_people.corr()["num_people"].sort_values(ascending=False)
975
976
977 # In[104]:
978
979
980 from pandas.core.indexing import IndexingError
981 # Function to plot infrared images
982 def inf_matrix(df, n_cols=4, labels=None):
983     n_rows = math.ceil(df.shape[0]/n_cols)
984     fig, ax = plt.subplots(n_rows,n_cols, figsize=(16,4*n_rows))
985     plt.subplots_adjust(hspace=0.5, right=0.8)
986     try:
987         tmin, tmax = df.iloc[:,5:69].min().min(), df.iloc[:,5:69].max
                ().max()
988     except IndexingError:
989         tmin, tmax = df[5:69].min(), df[5:69].max()
990     for i, val in enumerate(df.values):
991         x, y = math.floor(i/n_cols), i%n_cols
992         if n_rows == 1:
993             axis = ax[y]
994         else:
995             axis = ax[x,y]
996         cax = axis.matshow(np.rot90(val[5:69].reshape(8,8).astype("
                float"), k=3), vmin=tmin, vmax=tmax)
997         if labels != None and len(labels) > i:
998             axis.set_title(labels[i])
999     fig.colorbar(cax, ax=ax)
1000
1001
1002 # In[105]:
1003
1004
1005 inf_matrix(data.iloc[4457:4465], labels=["Right␣side␣of␣bar", "Middle
     ␣right␣side␣of␣bar", "Middle␣left␣side␣of␣bar", "Left␣side␣of␣bar
     ", "1␣meter␣from␣bar", "Right␣side␣back␣of␣the␣room", "Left␣side␣
     back␣of␣the␣room", "In␣doorway"])
1006
1007
1008 # In[106]:
1009
1010
```

```
inf_matrix(data.iloc[4506:4516].drop(4507).drop(4510), labels=["Inside bar", "Left side of bar", "Right side of bar", "Front of bar", "1 meter from bar", "2 meters from bar", "3 meters from bar", "Back of the room"])


# In[107]:


# fog is not visible
# standing at bar left side
# firing the fog machine once has no significant impact
data.iloc[4460,5:69].max(), data.iloc[4508, 5:69].max()


# In[108]:


# standing at bar right side
data.iloc[4457,5:69].max(), data.iloc[4511, 5:69].max()
# no impact


# In[109]:


#standing at bar middle
data.iloc[4458,5:69].max(), data.iloc[4509, 5:69].max()
# no impact


# In[110]:


# 1 meter from bar
data.iloc[4461,5:69].max(), data.iloc[4512, 5:69].max()
# no impact


# In[111]:


# 2 meter from bar
data.iloc[4461,5:69].max(), data.iloc[4513, 5:69].max()
# 0.75 degree difference


```

```
# In[112]:


# 3 meter from bar
data.iloc[4461,5:69].max(), data.iloc[4514, 5:69].max()
# 0.75 degree difference


# In[113]:


# 3 meter from bar
data.iloc[4461,5:69].max(), data.iloc[4515, 5:69].max()
# 1.25 degree difference


# In[114]:


inf_matrix(data.iloc[4524:4532], labels=["Inside bar", "Left side of bar", "Right side of bar", "Front of bar", "1 meter from bar", "2 meters from bar", "3 meters from bar", "Back of the room"])


# In[115]:


# fog is not visible
# standing at bar left side
# 0.75 degrees difference
data.iloc[4460,5:69].max(), data.iloc[4525, 5:69].max()


# In[116]:


# standing at bar right side
data.iloc[4457,5:69].max(), data.iloc[4526, 5:69].max()
# no impact


# In[117]:


#standing at bar middle
data.iloc[4458,5:69].max(), data.iloc[4527, 5:69].max()
# no impact
```

```
1100
1101
1102 # In[118]:
1103
1104
1105 # 1 meter from bar
1106 data.iloc[4461,5:69].max(), data.iloc[4528, 5:69].max()
1107 # no impact
1108
1109
1110 # In[119]:
1111
1112
1113 # 2 meter from bar
1114 data.iloc[4461,5:69].max(), data.iloc[4529, 5:69].max()
1115 # 0.75 degree difference
1116
1117
1118 # In[120]:
1119
1120
1121 # 3 meter from bar
1122 data.iloc[4461,5:69].max(), data.iloc[4530, 5:69].max()
1123 # 1 degree difference
1124
1125
1126 # In[121]:
1127
1128
1129 # 3 meter from bar
1130 data.iloc[4461,5:69].max(), data.iloc[4531, 5:69].max()
1131 # 1 degree difference
1132
1133
1134 # In[122]:
1135
1136
1137 # conclusion: fog only affects readings >2 meters from bar by
        reduction of maximum 1.25 degrees
1138
1139
1140 # In[123]:
1141
1142
1143 fig, ax = plt.subplots(4, figsize=((16,21)))
1144 smoke_test = test3.query("timestamp␣<␣'2022-07-29␣18:25:00'").
        set_index("timestamp")
```

```
ax[0].plot(smoke_test["humidity"]);
ax[1].plot(smoke_test["temperature"]);
ax[1].plot(smoke_test["inf_mean"]);
ax[2].plot(smoke_test["eCO2"]);
ax[2].plot(smoke_test["TVOC"]);
ax[3].plot(smoke_test["inf_mean_diff"])
ax[3].plot(smoke_test["humidity_diff"].shift(-1))
for a in ax.flatten():
    a.xaxis.set_major_formatter(xfmt)
    a.axvspan(pd.to_datetime("2022-07-29 17:42:12"),pd.to_datetime("
        2022-07-29 17:43:24"), color="red", alpha=0.2)
    a.axvspan(pd.to_datetime("2022-07-29 17:47:27"),pd.to_datetime("
        2022-07-29 17:47:52"), color="green", alpha=0.2)
    a.axvspan(pd.to_datetime("2022-07-29 17:50:55"),pd.to_datetime("
        2022-07-29 17:51:29"), color="green", alpha=0.2)
    a.axvspan(pd.to_datetime("2022-07-29 17:53:50"),pd.to_datetime("
        2022-07-29 17:57:48"), color="red", alpha=0.2)
    a.axvspan(pd.to_datetime("2022-07-29 18:08:15"),pd.to_datetime("
        2022-07-29 18:11:57"), color="orange", alpha=0.2)
    a.axvspan(pd.to_datetime("2022-07-29 18:11:58"),pd.to_datetime("
        2022-07-29 18:12:42"), color="red", alpha=0.2)
    a.set_xlabel("time of day")
ax[0].set_ylabel("humidity in %")
ax[1].set_ylabel("temperature in C")
ax[2].set_ylabel("concentration in ppm")
ax[1].legend(["temperature", "average infrared temperature"], loc="
    lower right")
ax[2].legend(["CO2 equivalent", "total volatile organic compounds"],
    loc="upper right")
ax[2].set_ylim([-50,750])
ax[3].set_ylabel("change in average infrared temperature");


# In[124]:


data["ventilation"] = np.nan
data.loc[test3.index, "ventilation"] = 0
data.loc[(data["timestamp"] >= "2022-07-29 17:42:12") & (data["
    timestamp"] <= "2022-07-29 17:43:24"), "ventilation"] = 1
data.loc[(data["timestamp"] >= "2022-07-29 17:53:50") & (data["
    timestamp"] <= "2022-07-29 17:57:48"), "ventilation"] = 1
data.loc[(data["timestamp"] >= "2022-07-29 18:11:58") & (data["
    timestamp"] <= "2022-07-29 18:12:42"), "ventilation"] = 1


# In[125]:
```

```
1181
1182
1183 #from sklearn.model_selection import GridSearchCV
1184 #parameters = {"max_depth": range(1,20,1), "min_samples_split":range
     (2,20,1), "min_samples_leaf": range(1,20,1)}
1185 #cv_mod = GridSearchCV(estimator=RandomForestClassifier(), param_grid
     =parameters, scoring="accuracy")
1186 #cv_mod.best_estimator_
1187
1188 #results max_depth=2, min_samples_split=4
1189
1190
1191 # In[126]:
1192
1193
1194 from sklearn.ensemble import RandomForestClassifier
1195 cv_mod = RandomForestClassifier(max_depth=2, min_samples_split=4)
1196 train_cols = ["inf_mean_diff", "inf_mean", "humidity_diff", "
     humidity_diff_shift", "humidity", "ventilation"]
1197 train = data.loc[test3.index][train_cols].dropna()
1198 cv_mod.fit(train[train_cols[:-1]], train["ventilation"])
1199
1200
1201 # In[127]:
1202
1203
1204 data.loc[(data[train_cols[:-1]].notna()).all(axis=1), "
     ventilation_pred2"] = cv_mod.predict(data[train_cols[:-1]].dropna
     ())
1205
1206
1207 # In[128]:
1208
1209
1210 test3 = data.query("(timestamp␣>=␣'2022-07-29␣00:00:00')␣&␣(timestamp
     ␣<␣'2022-07-29␣20:00:00')")
1211 smoke_test = test3.query("timestamp␣<␣'2022-07-29␣18:25:00'").
     set_index("timestamp")
1212 fig, ax = plt.subplots()
1213 ax.fill_between(smoke_test.index, 0, 1, where=smoke_test["
     ventilation_pred2"], alpha=0.2, transform=ax.get_xaxis_transform
     (), color="blue")
1214 ax.fill_between(smoke_test.index, 0, 1, where=smoke_test["ventilation
     "], alpha=0.2, transform=ax.get_xaxis_transform(), color="red")
1215 plt.plot(smoke_test["humidity"]);
1216 plt.ylim([50,70]);
1217 plt.xlabel("time␣of␣day")
```

```
plt.ylabel("humidity in %")
ax.xaxis.set_major_formatter(xfmt)


# In[129]:


test3 = data.query("(timestamp >= '2022-07-29 00:00:00') & (timestamp < '2022-07-29 20:00:00')")
smoke_test = test3.query("timestamp < '2022-07-29 18:25:00'").set_index("timestamp")
fig, ax = plt.subplots()
ax.fill_between(smoke_test.index, 0, 1, where=smoke_test["ventilation_pred"], alpha=0.2, transform=ax.get_xaxis_transform(), color="blue")
ax.fill_between(smoke_test.index, 0, 1, where=smoke_test["ventilation"], alpha=0.2, transform=ax.get_xaxis_transform(), color="red")
plt.plot(smoke_test["humidity"]);
plt.ylim([50,70]);
plt.xlabel("time of day")
plt.ylabel("humidity in %")
ax.xaxis.set_major_formatter(xfmt)


# In[130]:


fig, ax = plt.subplots(3,3, figsize=(20,15))
labels=["party 1", "party 2", "party 3", "party 4", "party 5", "party 6", "test 1", "test 2", "test 3"]
for index, party in enumerate([party1, party2, party3, party4, party5, party6, test1, test2, test3]):
    x, y = math.floor(index/3), index%3
    ax[x,y].plot(party.set_index("timestamp")["humidity"])
    ax[x,y].plot(party.set_index("timestamp")["inf_mean"])
    ax[x,y].fill_between(party.set_index("timestamp").index, 0, 1, where=party.set_index("timestamp")["ventilation_pred"], alpha=0.2, transform=ax[x,y].get_xaxis_transform(), color="blue")
    #ax[x,y].plot(party.set_index("timestamp")["num_people_pred_std1"])
    #ax[x,y].plot(party.set_index("timestamp")["num_people_pred"])
    ax[x,y].xaxis.set_major_formatter(xfmt)
    ax[x,y].set_xlabel("time of day")
    ax[x,y].set_title(labels[index]);


# In[131]:
```

```
1255
1256
1257 fig, ax = plt.subplots()
1258 ax.plot(party6.set_index("timestamp")["humidity"])
1259 ax.plot(party6.set_index("timestamp")["inf_mean"])
1260 ax.fill_between(party6.set_index("timestamp").index, 0, 1, where=
         party6.set_index("timestamp")["ventilation_pred"], alpha=0.2,
         transform=ax.get_xaxis_transform(), color="blue")
1261 ax.xaxis.set_major_formatter(xfmt);
1262
1263
1264 # In[132]:
1265
1266
1267 party2 = data.query("(timestamp␣>=␣'2022-06-11␣00:00:00')␣&␣(
         timestamp␣<␣'2022-06-13␣00:00:00')")
1268 party2_test = party2.query("timestamp␣<␣'2022-06-12␣03:00:00'").
         set_index("timestamp")
1269 fig, ax = plt.subplots()
1270 ax.fill_between(party2_test.index, 0, 1, where=party2_test["
         ventilation_pred"], alpha=0.2, transform=ax.get_xaxis_transform()
         , color="red")
1271 plt.plot(party2_test["humidity_diff_shift"])
1272 plt.plot(party2_test["inf_mean_diff"], alpha=0.5);
1273
1274
1275 # In[133]:
1276
1277
1278 inf_matrix(data.iloc[4466:4478])
1279
1280
1281 # In[134]:
1282
1283
1284 inf_matrix(data.iloc[4636:4642].drop(4638).drop(4639), labels=["Off",
          "On", "On", "Off"])
1285
1286
1287 # In[135]:
1288
1289
1290 inf_matrix(data.iloc[4548:4560])
1291
1292
1293 # In[136]:
1294
```

```
1295
1296 ven_test = party2.query("(timestamp > '2022-06-12 00:00:00') & (
         timestamp < '2022-06-12 01:00:00')")
1297 plt.plot(ven_test.set_index("timestamp")["inf_mean"]);
1298
1299
1300 # In[137]:
1301
1302
1303 inf_matrix(ven_test)
1304
1305
1306 # In[138]:
1307
1308
1309 inf_matrix(data.iloc[2306:2310], labels=["1", "2", "3", "4"])
1310
1311
1312 # In[139]:
1313
1314
1315 data.iloc[2306:2310][["inf_std", "inf_mean"]]
1316
1317
1318 # In[140]:
1319
1320
1321 inf_matrix(data.iloc[[5184,5190,5196,5202]], labels=["5 seconds
         before", "55 seconds during", "125 seconds during", "53 seconds
         after"])
1322
1323
1324 # In[141]:
1325
1326
1327 data.iloc[[5184,5190,5196,5202]][["num_people", "inf_std", "inf_mean"
         ]]
1328
1329
1330 # In[142]:
1331
1332
1333 party_people["num_people"].corr(party_people["inf_std"])
1334
1335
1336 # In[143]:
1337
```

```
1338
1339 plt.plot(party6.groupby("hour_date").mean()["TVOC"]);
1340 plt.plot(party6.set_index("timestamp")["TVOC"]);
1341
1342
1343 # In[144]:
1344
1345
1346 data["eCO2"].corr(data["TVOC"])
1347
1348
1349 # In[145]:
1350
1351
1352 inf_matrix(data.iloc[5038:5054])
1353
1354
1355 # In[146]:
1356
1357
1358 plt.plot(party6.set_index("timestamp")["inf_std"]);
1359
1360
1361 # In[147]:
1362
1363
1364 inf_matrix(data.iloc[2858:2862])
1365
1366
1367 # In[148]:
1368
1369
1370 fig, ax = plt.subplots(3,3, figsize=(20,15))
1371 for index, party in enumerate([party1, party2, party3, party4, party5
        , party6, test1, test2, test3]):
1372     x, y = math.floor(index/3), index%3
1373     ax[x,y].plot(party.set_index("timestamp")["num_people_pred_rf"])
1374     ax[x,y].plot(party.set_index("timestamp")["num_people_pred"])
1375     ax[x,y].xaxis.set_major_formatter(xfmt)
1376     ax[x,y].set_ylim((0,20))
1377     ax[x,y].set_title(index);
1378
1379
1380 # In[149]:
1381
1382
1383 from matplotlib.ticker import MaxNLocator
```

```
fig, ax = plt.subplots(3,3, figsize=(20,15))
labels=["party 1", "party 2", "party 3", "party 4", "party 5", "party 6", "test 1", "test 2", "test 3"]
for index, party in enumerate([party1, party2, party3, party4, party5, party6, test1, test2, test3]):
    x, y = math.floor(index/3), index%3
    ax[x,y].plot(party.set_index("timestamp")["num_people_pred_std"])
    ax[x,y].plot(party.set_index("timestamp")["num_people_pred_std1"])
    #ax[x,y].plot(party.set_index("timestamp")["num_people_pred"])
    ax[x,y].xaxis.set_major_formatter(xfmt)
    ax[x,y].set_ylim((-1,19))
    ax[x,y].yaxis.set_major_locator(MaxNLocator(integer=True))
    ax[x,y].set_ylabel("predicted crowd size")
    ax[x,y].set_xlabel("time of day")
    ax[x,y].set_title(labels[index]);


# In[150]:


from matplotlib.ticker import MaxNLocator
fig, ax = plt.subplots(2,3, figsize=(20,10))
labels=["party 2", "party 3", "party 4", "party 5", "test 1", "test 2"]
for index, party in enumerate([party2, party3, party4, party5, test1, test2]):
    x, y = math.floor(index/3), index%3
    ax[x,y].plot(party.set_index("timestamp")["num_people_pred_std"])
    ax[x,y].plot(party.set_index("timestamp")["num_people_pred"])
    #ax[x,y].plot(party.set_index("timestamp")["num_people_pred"])
    ax[x,y].xaxis.set_major_formatter(xfmt)
    ax[x,y].set_ylim((-1,19))
    ax[x,y].yaxis.set_major_locator(MaxNLocator(integer=True))
    ax[x,y].set_ylabel("predicted crowd size")
    ax[x,y].set_xlabel("time of day")
    ax[x,y].set_title(labels[index]);


# In[151]:


#combinedmodel
from matplotlib.ticker import MaxNLocator
fig, ax = plt.subplots(3,3, figsize=(20,15))
labels=["party 1", "party 2", "party 3", "party 4", "party 5", "party 6", "test 1", "test 2", "test 3"]
```

```
for index, party in enumerate([party1, party2, party3, party4, party5
    , party6, test1, test2, test3]):
    x, y = math.floor(index/3), index%3
    ax[x,y].plot(party.set_index("timestamp")["num_people_pred_std2"
        ])
    #ax[x,y].plot(party.set_index("timestamp")["num_people_pred"])
    ax[x,y].xaxis.set_major_formatter(xfmt)
    ax[x,y].set_ylim((-1,19))
    ax[x,y].yaxis.set_major_locator(MaxNLocator(integer=True))
    ax[x,y].set_ylabel("predicted␣crowd␣size")
    ax[x,y].set_xlabel("time␣of␣day")
    ax[x,y].set_title(labels[index]);


# In[152]:


inf_matrix(data.iloc[party4[party4["num_people_pred_std1"].notna()].
    index[:20]])


# In[153]:


inf_matrix(data.iloc[3939-4:3939+4])


# In[154]:


inf_matrix(data.iloc[party3[party3["num_people_pred_std"] ==1].index
    [:4]])#, labels=list(data.iloc[party3.index[:4]]["
    num_people_pred_std"].values))


# In[197]:


plt.matshow(np.rot90(data.mean(numeric_only=True)[5:69].values.
    reshape(8,8).astype("float"), k=3));


# In[156]:


plt.matshow(np.rot90(data.std()[5:69].values.reshape(8,8).astype("
    float"), k=3));
```

```
1465
1466
1467 # In[157]:
1468
1469
1470 fig, ax = plt.subplots(3,3, figsize=(20,15))
1471 for index, party in enumerate([party1, party2, party3, party4, party5
         , party6, test1, test2, test3]):
1472     x, y = math.floor(index/3), index%3
1473     ax[x,y].plot(party.set_index("timestamp")["num_people_pred_std"])
1474     ax[x,y].plot(party.set_index("timestamp")["
             num_people_pred_std_floor"])
1475     ax[x,y].xaxis.set_major_formatter(xfmt)
1476     ax[x,y].set_ylim((0,20))
1477     ax[x,y].set_title(index);
1478
1479
1480 # In[158]:
1481
1482
1483 fig, ax = plt.subplots(3,3, figsize=(20,15))
1484 for index, party in enumerate([party1, party2, party3, party4, party5
         , party6, test1, test2, test3]):
1485     x, y = math.floor(index/3), index%3
1486     ax[x,y].plot(party.set_index("timestamp")["
             num_people_pred_std_comb"])
1487     ax[x,y].xaxis.set_major_formatter(xfmt)
1488     ax[x,y].set_ylim((0,20))
1489     ax[x,y].set_title(index);
1490
1491
1492 # In[159]:
1493
1494
1495 fig, ax = plt.subplots(3,3, figsize=(20,15))
1496 for index, party in enumerate([party1, party2, party3, party4, party5
         , party6, test1, test2, test3]):
1497     x, y = math.floor(index/3), index%3
1498     ax[x,y].plot(party.set_index("timestamp")["
             num_people_pred_std_comb"])
1499     ax[x,y].plot(party.set_index("timestamp")["num_people_pred_hum"])
1500     ax[x,y].xaxis.set_major_formatter(xfmt)
1501     ax[x,y].set_ylim((0,20))
1502     ax[x,y].set_title(index);
1503
1504
1505 # In[160]:
```

```

plt.scatter(data["inf_std"], data["humidity"], s=1);

# In[161]:

plt.scatter(data["inf_std"], data["temperature"], s=1);

# In[162]:

index = party1[party1["num_people_pred_std"] > 0].index

# In[163]:

inf_matrix(data.iloc[index], labels=list(data.iloc[index]["hour"].
    values))

# In[164]:

fig, ax = plt.subplots()
plt.plot(party6.set_index("timestamp")["inf_mean_diff"]);
plt.axhline(y=0.93)
plt.axhline(y=-1.23)
ax.xaxis.set_major_formatter(xfmt);

# In[165]:

fig, ax = plt.subplots()
ax.plot(party6.set_index("timestamp")["humidity"])
ax.plot(party6.set_index("timestamp")["inf_mean"]+32)
ax.set_ylim([50,80])
ax.fill_between(party6.set_index("timestamp").index, 0, 1, where=
    party6.set_index("timestamp")["ventilation_pred"], alpha=0.2,
    transform=ax.get_xaxis_transform(), color="blue")
ax.xaxis.set_major_formatter(xfmt);

```

```
# In[166]:


inf_matrix(data.iloc[party4.index[[0,100,200,300]]])


# In[167]:


data_minute = data.groupby("minute").mean().copy(deep=True)


# In[168]:


data_minute["crowd_size"] = data_minute["num_people_pred_std2"]
time_skip = (pd.Series(data_minute.index).diff() > pd.Timedelta("
    00:01:02")).values
measures = ["temperature", "humidity", "eCO2", "TVOC"]

for col in measures:
    diff_col = "{}_diff".format(col)
    meas_col = "{}_mood".format(col)
    data_minute[diff_col] = data_minute[col].diff()
    data_minute.loc[time_skip, diff_col] = np.nan
    data_minute[meas_col] = data_minute[diff_col] / data_minute["
        crowd_size"]
    data_minute.loc[~np.isfinite(data_minute[meas_col]), meas_col] =
        np.nan
    data_minute[meas_col+"_ven"] = data_minute[data_minute["
        ventilation_pred"] != 1][meas_col]
    data_minute[meas_col+"_0"] = data_minute[meas_col]
    data_minute.loc[(data_minute[meas_col] < 0), meas_col+"_0"] = 0


# In[169]:


test1_m = data_minute.query("(index␣<␣'2022-05-10␣00:00:00')␣&␣(index
    ␣>=␣'2022-05-09␣00:00:00')")
test2_m = data_minute.query("(index␣>=␣'2022-05-24␣00:00:00')␣&␣(
    index␣<␣'2022-05-26␣00:00:00')")
party1_m = data_minute.query("(index␣>=␣'2022-05-27␣00:00:00')␣&␣(
    index␣<␣'2022-05-29␣00:00:00')")
party2_m = data_minute.query("(index␣>=␣'2022-06-11␣00:00:00')␣&␣(
    index␣<␣'2022-06-13␣00:00:00')")
```

```
party3_m = data_minute.query("(index >= '2022-06-16 00:00:00') & (
    index < '2022-06-18 00:00:00')")
party4_m = data_minute.query("(index >= '2022-06-18 22:00:00') & (
    index < '2022-06-20 00:00:00')")
party5_m = data_minute.query("(index >= '2022-07-23 00:00:00') & (
    index < '2022-07-24 00:00:00')")
test3_m = data_minute.query("(index >= '2022-07-29 00:00:00') & (
    index < '2022-07-29 20:00:00')")
party6_m = data_minute.query("(index >= '2022-07-29 20:00:00') & (
    index < '2022-07-30 07:00:00')")


# In[170]:


mood_cols = data_minute.columns[-12:]
mood_cols


# In[171]:


data = data.join(data_minute[mood_cols], on="minute")


# In[172]:


measures = data.groupby(["hour_date", "song"]).mean()[mood_cols].
    reset_index(drop=True)


# In[173]:


measures


# In[174]:


for suffix in ["", "_ven", "_0"]:
    data["mood{}".format(suffix)] = (data["temperature_mood{}".format
        (suffix)] / data["temperature_mood{}".format(suffix)].max())
        + (data["humidity_mood".format(suffix)] / data["humidity_mood
        ".format(suffix)].max())

```

```
1626
1627 # In[175]:
1628
1629
1630 song_count = data.groupby(["hour_date", "song"]).mean().groupby("song
         ").count().sort_values("temperature",ascending=False)
1631 song_count
1632
1633
1634 # In[176]:
1635
1636
1637 data.shape[0]
1638
1639
1640 # In[177]:
1641
1642
1643 data["song"].nunique()
1644
1645
1646 # In[178]:
1647
1648
1649 data[data["song"].notna()].shape[0]
1650
1651
1652 # In[179]:
1653
1654
1655 song_count[song_count["temperature"] > 1].shape[0]
1656
1657
1658 # In[180]:
1659
1660
1661 song_count[song_count["temperature"] == 2].shape[0]
1662
1663
1664 # In[181]:
1665
1666
1667 song_count["temperature"].hist(bins=8)
1668 plt.ylabel("number␣of␣songs")
1669 plt.xlabel("number␣of␣times␣a␣song␣has␣been␣played");
1670
1671
```

```
# In[182]:


ranking1 = data[data["song"].isin(song_count[song_count["temperature"
    ] >= 2].index)]
ranking1.shape[0]


# In[183]:


ranking2 = data[data["song"].isin(song_count[song_count["temperature"
    ] >= 3].index)]
ranking1.shape[0]


# In[184]:


ranking1.groupby("song").mean()[["mood", "temperature_mood", "
    humidity_mood"]].sort_values("temperature_mood", ascending=False)
    .head(10)


# In[185]:


ranking2.groupby("song").mean()[["mood", "temperature_mood", "
    humidity_mood"]].sort_values("temperature_mood", ascending=False)
    .head(10)


# In[186]:


ranking = pd.DataFrame()
ranking["song"] = ranking1["song"].unique()
ranking = ranking.set_index("song")


# In[187]:


data.columns[-16:-4]


# In[188]:
```

```
1713
1714
1715 for measure in data.columns[-16:-4]:
1716     ranking.loc[ranking1.groupby("song").mean().sort_values(measure,
         ascending=False).index.values, measure+"_r1"] = range(
         ranking1["song"].nunique())
1717     ranking.loc[ranking2.groupby("song").mean().sort_values(measure,
         ascending=False).index.values, measure+"_r2"] = range(
         ranking2["song"].nunique())
1718
1719
1720 # In[189]:
1721
1722
1723 ranking.columns.values
1724
1725
1726 # In[190]:
1727
1728
1729 error_matrix1 = pd.DataFrame()
1730 for col1 in ranking.columns[range(0,24,2)]:
1731     error_list = pd.Series(dtype="float64")
1732     for col2 in ranking.columns[range(0,24,2)]:
1733         rank_notna = ranking[(ranking[col1].notna()) & (ranking[col2
             ].notna())]
1734         try:
1735             error = mean_squared_error(rank_notna[col1], rank_notna[
                 col2])
1736         except ValueError:
1737             error = np.nan
1738         error_list[col2] = error
1739     error_matrix1[col1] = error_list
1740
1741
1742 # In[191]:
1743
1744
1745 error_matrix2 = pd.DataFrame()
1746 for col1 in ranking.columns[range(1,24,2)]:
1747     error_list = pd.Series(dtype="float64")
1748     for col2 in ranking.columns[range(1,24,2)]:
1749         rank_notna = ranking[(ranking[col1].notna()) & (ranking[col2
             ].notna())]
1750         try:
1751             error = mean_squared_error(rank_notna[col1], rank_notna[
                 col2])
```

```
            except ValueError:
                error = np.nan
            error_list[col2] = error
        error_matrix2[col1] = error_list


# In[192]:


fig, ax = plt.subplots()
cax = ax.matshow(error_matrix1);
#cbar_ax = fig.add_axes([0.85, 0.15, 0.05, 0.7])
fig.colorbar(cax);#, cax=cbar_ax)


# In[193]:


fig, ax = plt.subplots()
cax = ax.matshow(error_matrix2);
#cbar_ax = fig.add_axes([0.85, 0.15, 0.05, 0.7])
fig.colorbar(cax);#, cax=cbar_ax)


# In[194]:


data["TVOC"].corr(data["eCO2"])


# In[195]:


data.tail(1)


# In[196]:


data["num_people"].notna().sum()/6/60
```

Listing 2: Jupyter notebook used for data analysis