Universiteit Utrecht

Towards a Data-Driven Energy Advice

Creating a base model for gas usage prediction
&
Classifying external heating sources

Martin Doornhein
6209432
Master Thesis
MSc Applied Data Science
Utrecht Univeristy
First reader Prof. dr. Anro Siebes
Second reader: dr. Ing. habil G. Krempl
2 July 2021

Table of Content

Preface

This study was carried out as an assignment for the company Intergas. Within this study, there were two parts. The first part is a collaboration of three Applied Data Science students from University Utrecht. This part aims to create a base prediction model that predicts gas usage using the available data. Within this collaboration part, we have made use of each student's own strengths. This resulted in the following division of tasks:

Name	**Responsibility**
Martin Doornhein	Data analysis in Python and development of methodology of regressor model implementation.
Robin Reijers	Writing of the energy gap, digital energy label and it's related literature research. Together with the exploratory data analysis.
Lesley Rietvelt	Background research (with a focus on the current energy label) and writing.

The second part of this thesis consists of the individual research of each student. This thesis focuses on classifying external heating sources.

Introduction

The European Union has set the goal to become climate neutral by 2050. Each country that falls under the EU, is required to develop national policies that support this goal (RVO, n.d.). In the Netherlands, the The Netherlands Enterprise Agency (RVO) is responsible for implementing these national policies. According to the EU, homes and utility buildings are responsible for 40 percent of our energy consumption and 36 percent of total CO_2 emissions (European Commission, 2020).Therefore, the RVO is implementing policies that help lower the CO_2 emissions of buildings and improve their energy efficiency.

One of these policies is the energy labelling system, which is currently used to advise home owners on their house's energy efficiency (RVO, 2021). To stimulate homeowners to invest in improving energy efficiency and to make this process accessible, the RVO is trying to create a National Digital Platform. Their first step is to automate the process of determining the energy efficiency of Dutch houses. This means the current energy labelling system will be converted into a digital energy label.

In the Netherlands, homeowners are required to provide an energy label upon delivery, sale or rental of the home. When a homeowner wants to register the energy label, an energy advisor inspects their home and calculates its energy efficiency. Eventually, the energy label indicates the overall performance of the house from A++++ to G. Next to this, it contains details on the insulation and installations, offers advice for future improvement, and estimates the energy costs of the homeowner (RVO, 2021).

On 1 January 2021, the method that determines the energy efficiency of a house was updated. Energy labels are now held against new standards and documents (NTA 8800, BRL 9500, and ISSO 82.1) (RVO 2021). Although the new and old methods overlap, NTA 8800 requires more information on the building and advisors have to report every apartment separately. Next to this, more detailed descriptions of, among others, pipe transits and insulation, heating installations, delivery and distribution systems, and cooling installations are required (RVO, 2020a). For example, DGMR (DGMR, n.d.) established that the construction method of a building influences its energy efficiency, and therefore advisors are required to provide information on the construction method from January 2021 onwards (RVO, 2020b). Thus, creating an energy label with the newly required standards and documents is a time-consuming process. An inspection takes, on average, one to two hours to record all the details mentioned above (Energielabel, n.d.). This extensive inspection should theoretically guarantee accuracy. However, the label given is highly influenced by the methods and precision of the advisor and the availability of certain information (for example, whether or not it is possible to retrieve what materials were used for improving insulation) (Radar, 2021).

Next to this, energy labels do not accurately reflect the energy consumption of a house due to the "Energy Gap".

Recently, an increasing amount of research has been done regarding energy labels and the "Energy Gap". This research shows energy labels give an estimate of the energy consumption that often differs significantly from the actual consumption. This difference is called the energy gap. Buildings that are considered inefficient have a lower actual consumption than the estimated consumption, while buildings that are considered energy efficient often have a higher actual consumption than the estimation (Majcen, Itard & Visscher, 2013; van den Brom, Meijer & Visscher, 2018). Policies to reduce energy consumption in order to meet the 2050 goal are based on the theoretical consumption and energy efficiency of buildings. Since these estimations differ from the actual energy efficiency, these policies are less effective in practice than on paper (Macjen, Itard & Visscher, 2013). Several studies called for a better way to address the current energy gap and the usage of energy labels in housing as houses are currently labeled unfairly (Majcen, Itard & Visscher, 2013; van den Brom, Meijer & Visscher, 2018; Boonekamp, 2007; Martens & Spaargaren, 2005; Majcen, Itard & Visscher, 2015).

Intergas is researching the possibility of creating a digital energy labelling method based on their data, which could perhaps decrease the energy gap. Intergas is a company that focuses on developing sustainable heating systems, advising customers based on data, and selling their energy efficient systems (Intergas, n.d.). Over the years, they have collected data from their heaters, resulting in approximately 6TB of information in their database. This data, together with weather measurements from the Royal Dutch Meteorological Institute (KNMI), can be used to create basic and improved models that are able to classify houses based on their energy usage. Since such a model would be based on actual gas consumption, it is possible the estimations are more accurate than current methods.

Earlier research shows several factors influence actual gas usage and energy efficiency. For example, building characteristics such as the surface of the building's floors in m^2 and building year often have an impact on energy consumption (Majcen & Itard, 2014a). Resident's characteristics (such as age, income and social standing) also influence the consumption, which was apparent due to differences in consumption between houses with similar characteristics (Majcen, Itard & Visscher, 2013; Yun & Steemers, 2011; Berkland, 2014; Santin, 2011; Jeeninga, Uyterlinde & Uitzinger, 2001). Lastly, resident's behavior (such as heating patterns, airing and absence) has an impact on how much energy is used as well (Majcen & Itard, 2014b). The data provided by Intergas contains information on some of the important building characteristics, such as the building year and surface area. Next to this, Intergas' data could be used to gain insight into resident's behavior by trying to detect heating patterns and research possibilities for correcting for this behavior.

Research also shows electrical usage is negligible when researching energy labels. The RVO energy label is exclusively taking electrical installations into consideration, while most electricity usage comes from household appliances. The theoretical electricity usage and actual electricity usage of those installations do not differ from each other, or per label class (Majcen & Itard, 2014b).Therefore, electricity usage is negligible in this analysis.

This thesis then aims to explore the opportunities of creating a model based on Intergas' data which can accurately predict the gas consumption of a building in order to eventually assign correct energy labels. Next to this, the individual cases will research possibilities of clustering, classifying and correcting in order to improve the base model. If such a model can indeed be created, future research can improve the accuracy of this model and further explore the possibilities of creating a digital energy label, but that is not within the scope of this project.

According to what was scheduled for this research, another exploratory question was proposed by Intergas. The proposal was a classification task that aimed to classify external heating sources. The question is: "can we classify if there are external heating sources?". This classification should be derived from the available data without any reference data. To classify this property, a certain rule has to be determined. This rule can distinguish, based on the currently available data, whether a house has an external heating source or not. This classification, then, would help gain insight in resident's behavior and could be used to improve the base model.

According to a panel research done by climate agency HIER (2018), 88% of the Dutch people use radiators for indoor climate control. The second highest percentage (19%) of warming is underfloor heating. The third highest percentage is a wood stove (8%). Apparently, people can use different heating sources within one house. Some households will use a gas fueled system as their main system, and use an external heating source, like a woodstove, as an extra system. Radiators use gas as their fuel to provide heating. Underfloor heating could be fueled by gas, but also by electricity. For the wood stove it seems clear that the common fuel is wood. The presence of different heating sources within one household might have an influence on the gas usage of the resident. The expectation is that having a wood stove or underfloor heating fueled by electricity will reduce the gas usage.

This classification task will focus on heating sources that are not fueled by gas. These heating sources should be apparent in a house that also uses heating by gas. This condition is needed to see differences in gas usage for different timepoints. Because the data is collected by central heating systems that are fueled by gas, it is very likely that households use gas as a fuel to heat the house.

Method -- Base model

Since the aim of this project is exploratory, the focus laid on what information the data could provide. However, the large amount of data forced us to make decisions about filtering and rearranging the data. Next, exploratory analysis was done to find correlations and extract information. Eventually, a basic model was created and separate analyses were done to explore further possibilities. The software, data and methods used during this project, together with the decisions that have been made, will be explained in more detail in this chapter.

Software

Intergas' database was built upon Hadoop and consists of approximately 6TB of data. With large amounts of data, it is recommended to use Apache Spark. Spark is fast, offers less reading and writing from and to the disk and, due to the Python API PySpark, is fairly easy to use. It was also possible to work on smaller portions of the data, meaning other Python packages (e.g. Pandas) could be used. Overall, the following software packages were used: PySpark, Pandas, NumPy, Matplotlib, Seaborn, and Plotnine.

Data

The data provided was spread among different datasets. These datasets were used to create a data frame that would be useful for different analyses (see figure 1). This paragraph will discuss the different datasets and the features in it.

Intergas_raw

This dataset contained many columns. Most of its content was about small interval measurement values. For this project, the focus was on 24 hour data. Therefore, a selection of only a few columns was made, which gave information on the gas usage of central heating and warm water. Next to this, all gas use measurements below zero were removed when loading this dataset. This was done to reduce the amount of errors in the data and decrease the computational load while merging datasets.

Gas_use_hourly

This dataset contained information about hourly gas used per m^2 grouped by heater id's. These id's are unique numbers per heater (see table 3).

KNMI_data_24

This dataset is provided by Intergas.

This dataset contained information about the weather for every hour, grouped by time point and neighborhood.

Ig_heater_info

This dataset contained heater ID's, the neighborhood and two house properties from the given house. These properties were house building year and total surface area of residence.

First, the datasets described above were merged to create a data frame that could be used for inspection, exploration and analysis. This was done by using inner join, meaning the intersection of both datasets is used for the newly created data frame. Since this method only adds the overlapping keys, the new data frame has a low amount of missing values. Reducing the number of missing values was desirable, due to the small number of columns. Table 1 shows the order that was used to create the data frame for further analysis.

Table 1. Joining steps for the final dataset.

Dataframe 1	**Dataframe 2**	**Keys**	**New dataframe**
Intergas Raw	Ig_heater_info	Heater_id	Merge_1
Merge_1	Gas_use_hourly	Heater_id, date_day	Merge_2
Merge_2	df_knmi	Neighborhood, date_day	df_24hour

This final data frame contained 28,651,724 rows and 12 columns (table 2). From this data frame, a sample of 10% was taken. Since the whole data frame is too large to work with easily, this sample will be used for testing analysis and visual inspection.

Table 2. Variables in the merged dataset

Variable_name	Type	Meaning
Heater_id	Integer	Unique heater identification number
Neighborhood	Integer	Neighborhood
Date_day	Integer	Year/month/day
Surface	Integer	Surface of total house in m^2
Building_year	Integer	Building year of house
Rain	Double	Amount of rain in 0.1 mm (precipitation <0.05mm = -1)
Sun	Double	Amount of sun in 0.1 hours (sunshine <0.05h = -1)
Temp	Double	Temperature in celsius * 10
Wind	Double	Wind in 0.1 meters / second
T_act	Double	Actual inside house temperature (C)
T_set	Double	Set inside house temperature (C)
Gas_ch	Double	Gas use in m^3 for heating house
Gas_dhw	Double	Gas use in m^3 for hot water
Temp_diff	Double	Inside temperature - outside temperature (C)

A final step that was performed before the filtering process, was dealing with missing values. Based on the first inspection, it seemed that there were missings in 4 columns (see table 2). These columns were: rain, sun, temp and building year. After closer inspection, it became clear that there was a fifth column that had missings which were valued with a zero (n = 74,114). This column was Surface and the missings did overlap 100% with the building year column.

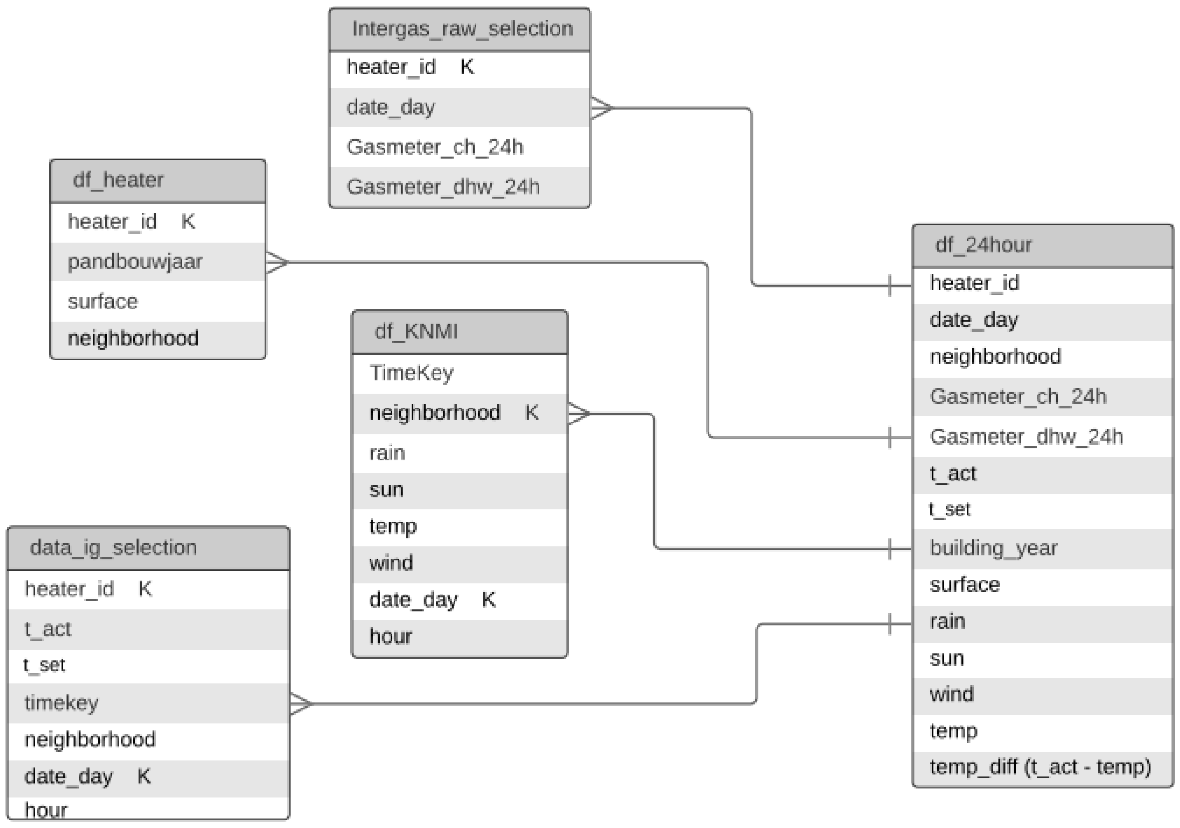
Table 3: columns with missing values

rain	sun	temp	building year
91,229	279,904	44,881	74,114

The missings values came from two different datasets that were used in the merging of datasets. The missings in the weather variables came from the KNMI dataset. A possible explanation for this missings is an error with one, or more of the weather stations. The missings in the variable building year and surface came from the Ig_heater_info dataset. A reasonable explanation for these missings is that these customers did not want to communicate their information. Due to not having a proper NMAR analysis, it was not possible to reverse calculate the missing data. To prevent skewing of the data it was chosen to leave out the rows which contained missing data.

After removing missing values (n = 259,922), the number of rows was: 28,393,420. The missing values were divided among rain (86,290), sun (258,923), temperature (42,533). For building year and surface 74,114 values are missing.

Figure 1. Data frames that were used to get the final dataframe



Filters

Table 3 shows the filters that were used to clean the dataset. These filters were chosen based on a combination of reasoning and the influence of the filter on the data. For the variable surface, the cutoff value of 40 was used. Houses smaller than 40 square meters were removed because these values are very unlikely to be correct. Looking at the data, 40 seemed to be an appropriate cutoff because the number of removals highly increased after applying a value higher than 40.

For both gas_measures, a cutoff value of 40 m^3 per 24 hours was used. Based on information from Intergas, values higher than 40 m^3 are very unlikely and could be considered errors from the heater. These errors can be caused by a momentary loss of power, for example. From the data perspective, these cutoffs did not influence the data much.

For the variables t_act (temperature measured inside) and t_set (temperature set on the thermometer), the cutoff value is equal or smaller than 26 degrees Celsius. These points were chosen because values higher than 26 are unusual. For t_act, there was also a minimum filter with a value of 10 added. This cutoff value was selected based on the data.

Table 4. Filters and number of removed rows after applying

Variable name	**Filter**	**# removed**	**% removed**
Surface min	> 40	299464	1.05%
Surface max	< 600	269998	0.94%
Gas_ch	< 40	7286	0.03%
Gas_dhw	< 40	484	0.002%
T_act	<= 30	33607	0.12
T_act	>= 10	1393515	4.86%
T_set	<= 26	19060	0.07%

The total number of removed rows is 1,978,666 which is 6.91% of the total number of rows (see table 4 for the descriptives).

Lastly, the variable temp_diff (temperature difference) contained values below zero, meaning the outside temperature was higher at these moments than the inside temperature. In this case, there is no gas needed. Therefore, the limit of this variable was modified and all negative values were replaced with zero.

Table 5. Descriptives of the removed rows

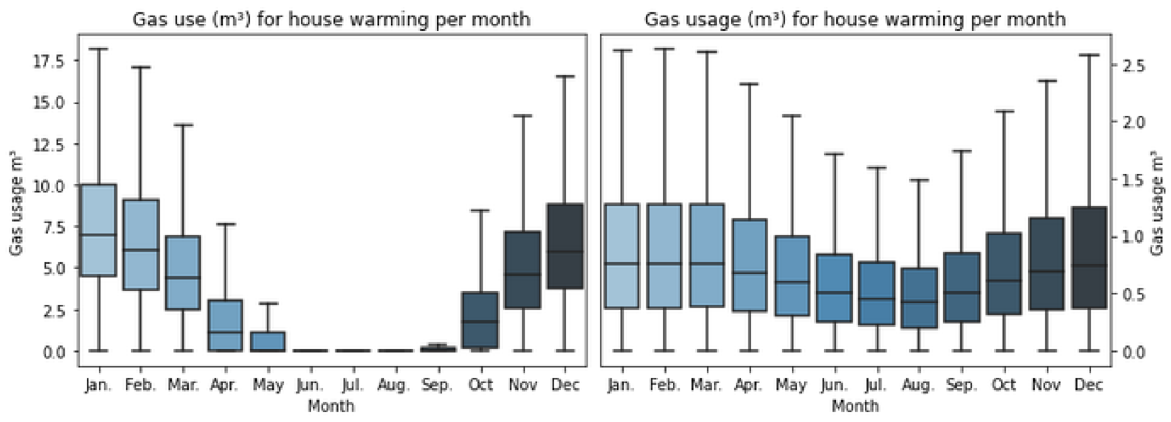
Summary	**Surface**	**Gas_ch**	**Gas_dhw**	**T_act**	**T_set**
Count	1.978.666	1.978.666	1.978.666	1.978.666	1.978.666
Mean	388,27	23,99	34,89	6,56	5,47
Standard deviation	1.553,91	2.471,23	3.610,76	11,27	8,77
Minimum	0	0	0	-323,65	-154,27
Maximum	68.353,0	429.496,73	429.483,71	326,95	325,11

Exploratory data analysis

After applying the filters, the dataset was reduced by a total of 6.91 percent when compared to the total data available. With this exploratory data analysis the make-up of the dataset will be shown. All visualisations are based on the earlier mentioned sample of 10%.

Gas usage for warming houses is lower (0.18 ± 0.73) in the summer months and higher in the winter months (7.10 ± 4.56). This is also true for warm water (0.57 ± 0.50) in summer versus (0.91 ± 0.78) in winter) but with not as much of a difference (figure 2).

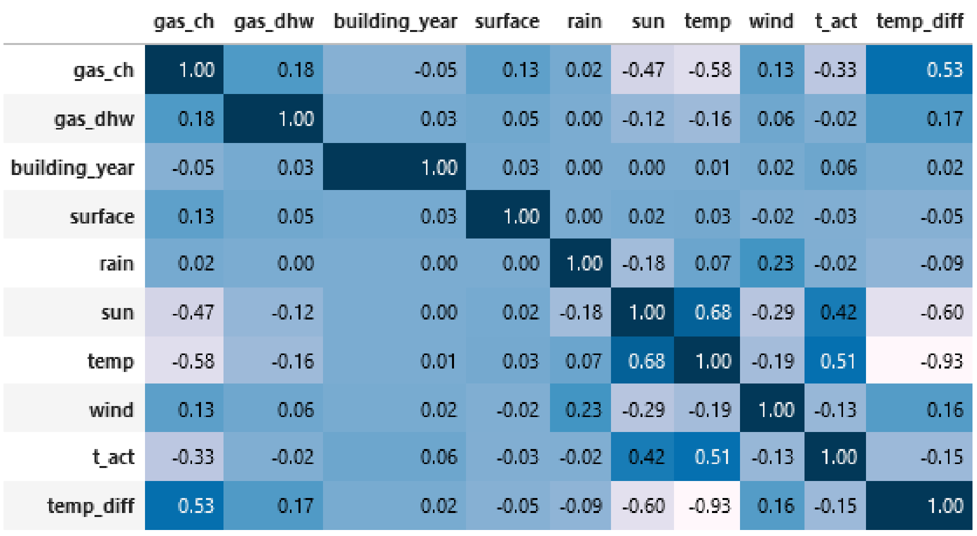
Figure 2: Boxplots of gas usage per 24 hours for house warming (left) and warm water (right)



During the exploratory analysis, correlations were found between certain variables. For example, there is a positive correlation between the average sunshine per day and average temperature (0.68). Another positive correlation (0.53) is temperature difference inside/outside and gas_ch (gas used for heating purposes).

With negative correlations the avg(temp) and temp(diff) are strongly negatively correlated (-0.93), also temperature difference inside/outside and average sunshine are negatively correlated (-0.60). Other negative correlations are between gas usage for heating and average temperature (-0.58) and the average sunshine (-0.47).

Figure 3: Correlations between variables

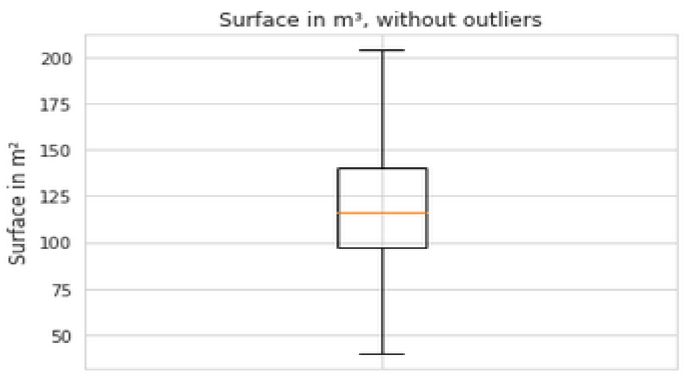


	gas_ch	gas_dhw	building_year	surface	rain	sun	temp	wind	t_act	temp_diff
gas_ch	1.00	0.18	-0.05	0.13	0.02	-0.47	-0.58	0.13	-0.33	0.53
gas_dhw	0.18	1.00	0.03	0.05	0.00	-0.12	-0.16	0.06	-0.02	0.17
building_year	-0.05	0.03	1.00	0.03	0.00	0.00	0.01	0.02	0.06	0.02
surface	0.13	0.05	0.03	1.00	0.00	0.02	0.03	-0.02	-0.03	-0.05
rain	0.02	0.00	0.00	0.00	1.00	-0.18	0.07	0.23	-0.02	-0.09
sun	-0.47	-0.12	0.00	0.02	-0.18	1.00	0.68	-0.29	0.42	-0.60
temp	-0.58	-0.16	0.01	0.03	0.07	0.68	1.00	-0.19	0.51	-0.93
wind	0.13	0.06	0.02	-0.02	0.23	-0.29	-0.19	1.00	-0.13	0.16
t_act	-0.33	-0.02	0.06	-0.03	-0.02	0.42	0.51	-0.13	1.00	-0.15
temp_diff	0.53	0.17	0.02	-0.05	-0.09	-0.60	-0.93	0.16	-0.15	1.00

The most frequently occurring surface area is 110m² with the lowest value being 40 m² due to the filter applied on surface area and the highest value is 600m². The average surface area in square metres is 125.42 (± 52). The outliers are not shown in the boxplot to reduce visual clutter (Figure 4).

Most of the data from Intergas is from residential housing, this would explain the distribution of surface area in housing that is found in the dataset. Meaning that the model possibly could face difficulties when trying to estimate the actual gas consumption in houses with a larger surface area.

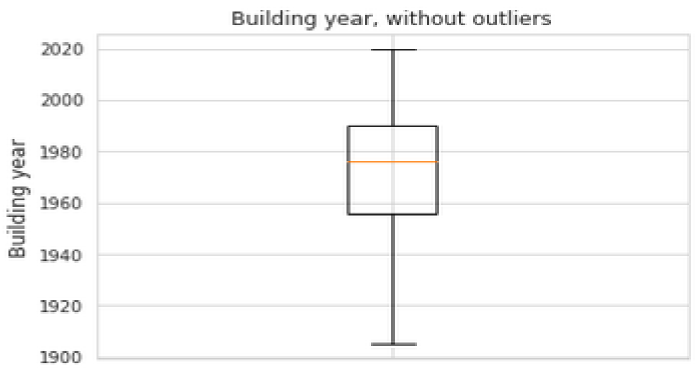
Figure 4: Boxplot of housing per m²



In the dataset the most frequently occurring building year of houses is 1978. The oldest building in the dataset dates from the year 1300 and there are no buildings newer than being built in 2020 (figure 5).Building years of the houses are on average 1971 (± 62). There are no outliers above 2020 in the dataset (figure 5)

This range of years encompasses several different updates of the EU regulations in housing insulation and requirements. Meaning that within the dataset there are houses which adhere to different standards of insulation and specs.

Figure 5: Boxplot of building years



Within the data it is found that summer months (Jun - Aug) have the highest temperatures and the most amount of sun hours for each month. While the winter months (Dec - Feb) have the lowest temperatures and amount of sunshine each month (Figure 6 & 7). This tells us the data itself is in line with the weather patterns as reported from the KNMI and that there are no abnormalities within the general sense of the dataset.

Figure 6: Boxplot of average temperature

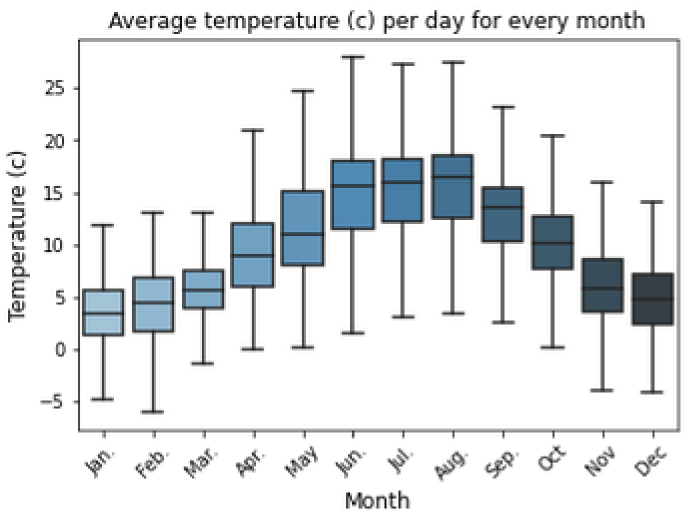
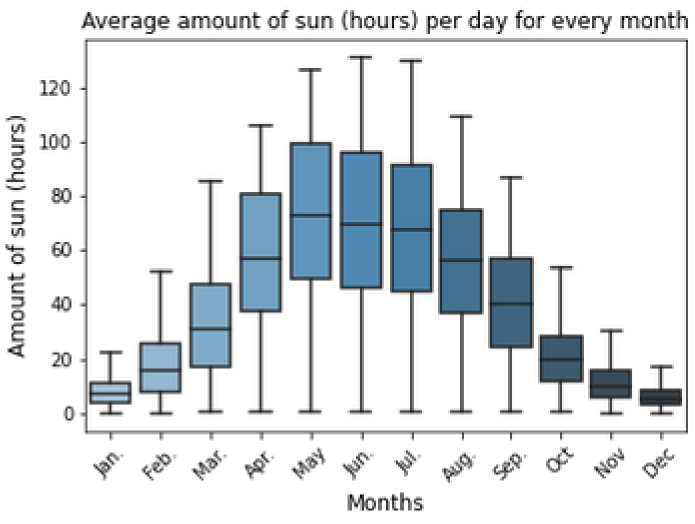
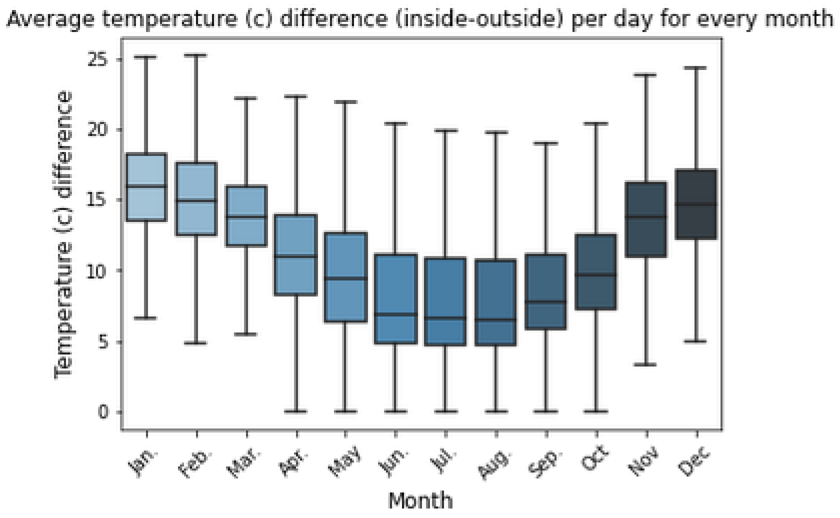


Figure 7: Boxplot of average sun hours



For the average temperature difference (inside - outside) the greatest temperature difference is found in the winter months and when looking at the smallest temperature difference it is found during the summer months (figure 8).

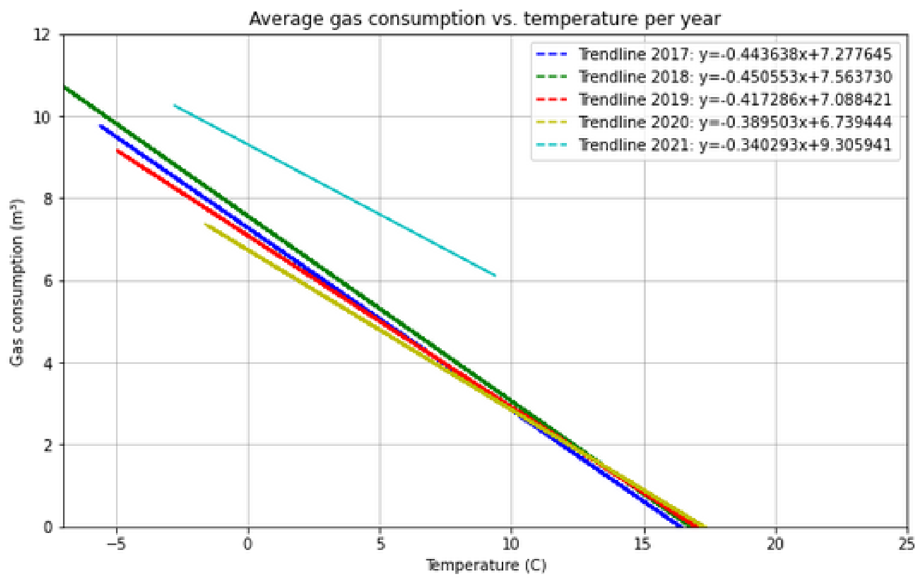
Figure 8: Average temperature difference (inside - outside) in Celcius per day for every month.



For all years there is a negative correlation found between gas consumption and temperature outside. The trendlines are linear, which is expected due to specific heat. Specific heat "is defined as the energy required to raise the temperature of a unit mass by one degree" (Yunus, Cengel & Ghajar, 2020).[1] It costs the same amount of energy to heat up the air in a room from 0 to 1 degree Celsius as it does from 5 to 6 degrees. On average the years 2017 through 2020 each have a similar slope and starting point, but the year of 2021 has a way higher starting point and slope value. This is due the dataset only partially containing 2021 and only the winter months, in these months the gas consumption is the most as previously shown (figure 9). Next to this, we expect the average gas usage to be higher due to working from home during corona.

[1] Yunus A. Çengel and Afshin J. Ghajar, Heat and Mass Transfer: Fundamentals and Applications, (McGraw-Hill Education, New York: 2015), p. 7.

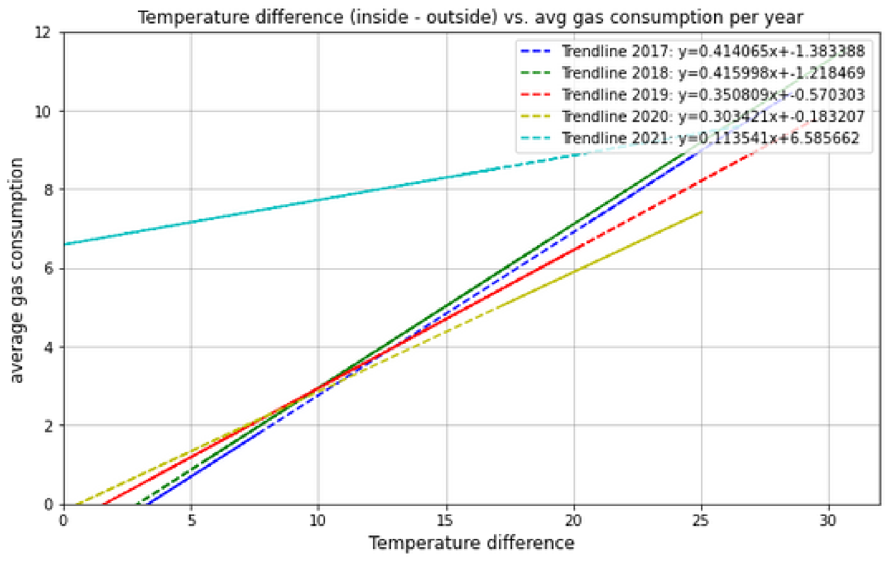
Figure 9: Average gas consumption plotted against temperature per year



The average gas consumption per year is positively correlated with the total temperature difference between inside and outside of homes. Meaning that the higher the temperature difference is, the higher the gas consumption is, tying together the temperatures outside during colder periods together with the increased gas consumption to gain the same amount of heating as one normally would get with higher outside temperatures.

All of the years between 2017 and 2020 show a similar slope profile. In 2021 there is a flatter but increased slope value due to only partly having values from the winter months and not the rest of the year meaning that the data is skewed to more gas consumption then in the previous years (figure 10).

Figure 10: Average gas consumption per year plotted against temperature difference (inside – outside)



Modelling base model

To model the data and predict the gas usage, a random forest regressor model was used. This model aims to predict the total gas usage per year, per m². The Random Forest regressor was chosen for different reasons. First, the model copes well with non-linear relations. Second, it deals well with collinearity. This is desirable because the KNMI-predictors did correlate among each other, for example: sun and temp (*r* = .701). One of the drawbacks of this method is that it is not able to extrapolate predictions. A linear regression model could predict outside the range of the train set.

To feed this random forest model, the earlier described dataframe was grouped by heater_id and years. This resulted in a dataframe with averages of all the variables for every heater id, per year. The outcome variable total gas consumption per m² was included. Some columns, which were no longer of need, were removed (heater_id, year, gas_ch, gas_dhw). To get the data in the right format to train the regressor, the vectorindexer from the PySpark package was used.

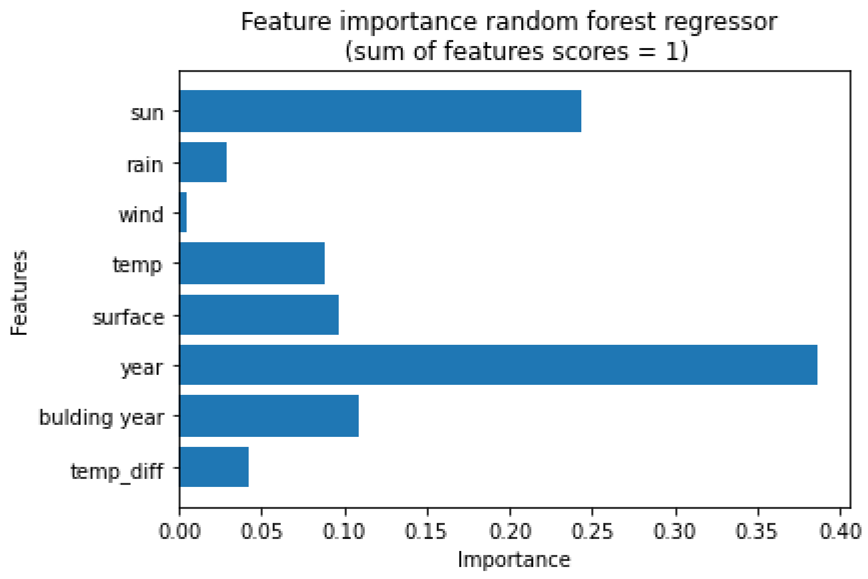
To train and test the model, a proportion of the data (30%) was held out for validation. The train set contained 87,399 rows and the test set contained 37,388 rows. Because there was a lot of data available, the hold out method seemed to be sufficient as a method to determine testing accuracy. Cross validation would be an alternative, but because of the high amount of data, this method would be highly computational expensive.

Results -- Base model

Different random forest regressor models were performed. Every model is trained on the whole dataset.The initial random forest model (r^2 = .460, RMSE = 0.0230) has the goal of being a baseline model. This model aims to predict gas usage per m^2 and was trained on 124,787 observations, which was divided into a train set of 70 % (n = 87,399) and a test set of 30% (n = 37,388). The mean of the outcome column is: 0.044 . In an effort to increase accuracy, an extra filter was added. The dataset was filtered for the number of observations within a year. This filter is applied for the cutoff values > 200 and > 250. For these models, the same train-test-set proportions were applied. The first model (n = 70,993, r^2 = .239, RMSE = 0.015) and second model (n = 64,413, r^2 = .245, RMSE = 0.015) both performed less well and had a lower explained variance. Therefore, the first model is chosen to be the base model.

To inspect the impact of the features on the prediction accuracy, a feature importance plot was visualized (figure 1). The highest scoring features are year and sun. Two features had a relative high influence on the model which were: year and sun. The feature wind had the lowest impact on the model.

Figure 11: Feature importance barplot for random regressor model



Method -- Classification Task

Data

To investigate the option of deriving the classification of external heating sources from the available data, the Intergas_hourly dataset was used. This set contains gas usage data points for every hour in a day. The dataset was combined with the KNMI dataset, to get the hourly weather information, based on the local district of the heater. This combination of datasets could help to provide insight into more specific heating patterns compared to the dataset that was used for the base model. An hourly heating pattern could possibly help to find out whether a consumer uses an external heating source or not. The merge of these two datasets was done with an inner join. This type of join aims to remove nonmatching data, which cannot be used later in the analysis. After this step, the merged dataset was combined with the heater_info dataset. This combining was also done with an inner join, which resulted in a dataset (n = 687,848,004) that contained the same heater IDs as the dataset that was used for the base model. A second advantage of this step was that the house properties were included in the newly created data frame. Applying filters on these house properties help to clean the data, as was done with the dataset created to train the base model. A difference and possible limitation, compared to the base model dataset is that this one only has one variable for gas use. So, there is no distinction between gas use by heating and gas use by warm water.

To assure that the outside temperature is low on average, the data was filtered for the months January and February. This low outside temperature will probably lead to more use of external heating sources. This filtering also overcomes the problem of low gas usage because of the relatively high outside temperature, which was an advice given by the domain expert from Intergas. From the filtered dataset, a sample of 1% (n = 18,252,987) was taken. Unless using a sample does decrease the data points per heater ID, some data reduction was needed because of computational considerations.

After creating the dataset that was needed to do the analysis, some of the earlier mentioned filter steps were applied. The majority of the earlier defined filters could be applied here. The same rules for house properties hold for this dataset. Based on the descriptives of the gas use variable, there was no sign of erroneous values. So, there was no need for any filter on this variable. This resulted in the data frame (n = 16,896,635) that is clean and ready for use.

Table 6: Steps for creating the data frame

Df	Action	New df name
Intergas_hourly	join(df_KNMI, how = inner)	Df_gashourly
Df_gashourly	Filter(month = 1 | month = 2)	Df_winter
Df_winter	Sample(proportion = .01)	Df_winter_sample
Df_winter_sample	Applying filters	Df_winter_sample

Design

To find a rule to classify the heater IDs based on their gas usage, certain expectations were defined. Firstly, when consumers use their external heating source, it is expected that the inside temperature is not lower than usual compared to heating by gas fueled systems. So, we look for normal actual inside temperatures. A second thing to expect is that the gas usage for housewarming is lower than usual, because there is no gas needed for house warming. And finally, these expectations are only true when the outside temperature is on a level that forces people to use some source of heating to warm their houses. These expectations together could be presented as the next rule:

If (outside temp < x) & (inside temp > y) & (gas_usage < z): 1 (external heating source)

else: 0 (no external heating source)

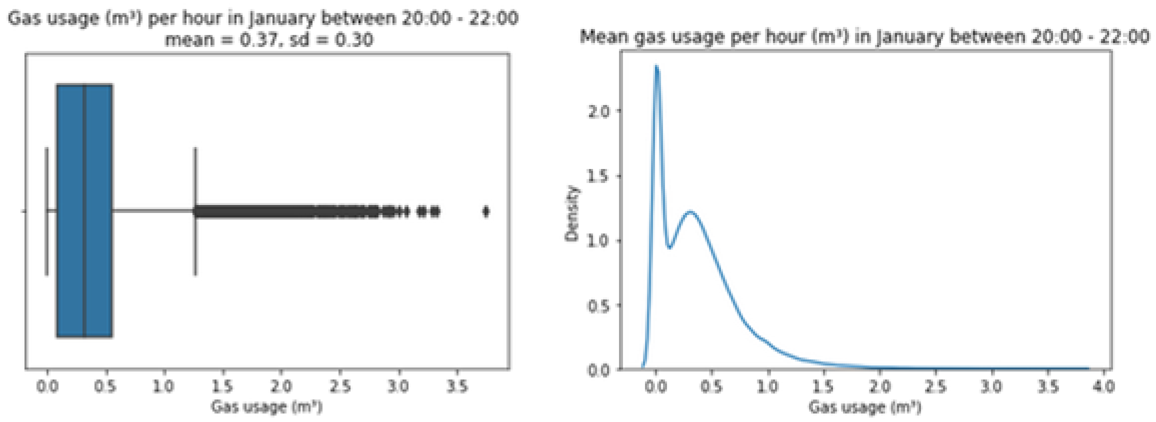
The goal is to find heater IDs that are likely to have an external heating source. This classification is based on the gas usage of a heater. The expectation about gas usage could be different for different types of external heating. For heating sources that are used to provide the regular heating of a house, the expectation is that the gas usage for different timepoints will be low on average. For heating sources that are not used as a regular heating source, like a wood stove, the expectation is that the gas usage will be different for different timepoints. To determine which values are applicable for the classification rule, different timepoints within a certain heater ID were compared. To further reduce the amount of data, only the data for the year 2020 was selected. This year was selected because in 2021 there was the corona pandemic. This pandemic situation could possibly influence the gas usage somehow. Because there was no labeled reference data for external heating sources, an assumption about when people use their external heating source needed to be made. This assumption was made in consultation with one of the domain experts from the heater company. We determined that people are more likely to use their non regular external heating source on Saturday night compared to Tuesday night. This choice is based on the assumption that a wood stove is associated with coziness. We expected that people are more driven to create this cozy experience during Saturday night than Tuesday night. For the regular external heating sources, Tuesday and Saturday night will both score low on average. In line with these assumptions, a subset was made which only included observations for Tuesday between 20:00 and 22:00 and Saturday between 20:00 and 22:00.

Table 7. Steps for creating data frame

Data frame	**Action**	**New data frame**
Df_winter_sample	Filter((dayofweek = 3 | dayofweek = 7) & year = 2020)	Df1
Df1	filter(hour = 20 | hour = 21	Df2
Df2	Groupby(heater_id, Timekey).mean()	df_grouped

In the first attempt to make a distinction between different kinds of heating patterns, heater IDs with anormal patterns were manually selected. Based on this selection it became clear that using an absolute value to classify could be problematic. The problem with this way of working is that there is some spread around the mean (mean = 0.38, SD = 0.36). In other words, there is a lot of difference within gas usage per resident (figure 1).

Figure 12: left: boxplot of gas usage. Right: density plot for gas usage.



To overcome the problem with using absolute values, a more relative variable was needed. This conclusion resulted in a formula that was used to provide insight into how much gas was used proportionally to the inside temperature. This formula is determined in consideration with the domain expert. The formula was:

$$Y = \textit{Gas usage} / (\textit{temperature actual} - \textit{temperature set})$$

To use this variable in the classification task, two points are important:

1. Houses that use external heating sources will have a low gas usage on average. Therefore, the Y in this formula will be low as well. When the temperature difference between t_act and t_set is high, Y will decrease. This is more likely by houses that use external heating sources, because the gas fueled heating system will not activate if the inside temperature is already at the set temperature. So, houses that could possibly be classified as 1, are likely to score low. Values with very small differences between t_act and t_set could be really high. For example: a gas usage of 0.5 divided by .01 = 50.
2. Y needs to be positive to get the classification 1. To get classified as 1, the t_act cannot be lower than the t_set. Because the expectation is that external heating sources will help to increase the t_act, without increasing t_set.

A point to take in consideration is the minimum gas usage variable value. The gas use variable will probably not be zero when there is no consumption by heating. In this variable, the consumption by using warm water is also included. Without using warm water, the consumption by heating water is also often not zero. This is due to a setting in many heating systems that provided warm water immediately. The heating system saves a small amount of water and warms this constantly. As a result, there is immediately warm water when using the warm water tap (Intergas, 2018).

In conclusion, the expectation is that houses that use an external heating source will have a positive Y value that is close to zero. To use this Y variable to make a distinction between heater IDs, a maximum allowed value for Y needs to be determined. This cutoff will determine the line between the classification of 1 versus 0. To find this value, data analysis focused on the distribution of gas usage was done. After determining this maximum allowed value, different classifying rules were inspected and compared. Finally, based on visual inspection, data points within heaters were compared. This inspection aimed to answer the question: “can we classify external heating sources based on the available data?”.

Results -- Classification Task

The selection of heater IDs that probably use an external heating source is selected based on a few elements. The elements contain the real distinguishing value, and some conditions that must be met. The following elements were included in the rules that were inspected:

1. A maximum allowed value for Y. This cutoff does make the distinction between a resident with an external heater and a non-external heater.
2. Gas usage for selected ID must be smaller than the average gas usage (mean = 0.37). This overcomes the problem that residents with a big inside temp difference (t_act – t_set), but also a high gas usage, still fall within the criteria.
3. The inside temperature (t_act) needs to be at a point that makes it proper to assume that people are at home. This value is set at 19 degrees (C) which is close to the reported average 19.9 according to the HIER research (2018) research.

The difference between the different classification rules that were tried, is based on the first element. Determining a cutoff value has a high influence on which IDs were selected. Y needs to be positive, but smaller than a certain value. Y is derived from gas use / (t_act - t_set), the maximum value should be the highest outcome for this Y variable. The same sort of formula was used for determining the cutoff. Beside this formula, some conditions must be met. These conditions are added to every filtering rule:

Cutoff = Y < (f(gas_use)) / x

Condition 1 = Y > 0

Condition 2 = gas_use < gas_use.mean()

Condition 3 = t_act > 19

Different options for f() and x were tried to get the best cutoff. The filtering rules were applied on a dataset (n = 80,706) with 24,213 unique heater IDs. The following rules were considered to be the best:

Cutoff = Y < (gas_usage.mean() – 1SD) / 1.5

This filtering rule did select a subset (n = 9,137) with 5,940 unique heater IDs, which is 25% of the initial data frame.

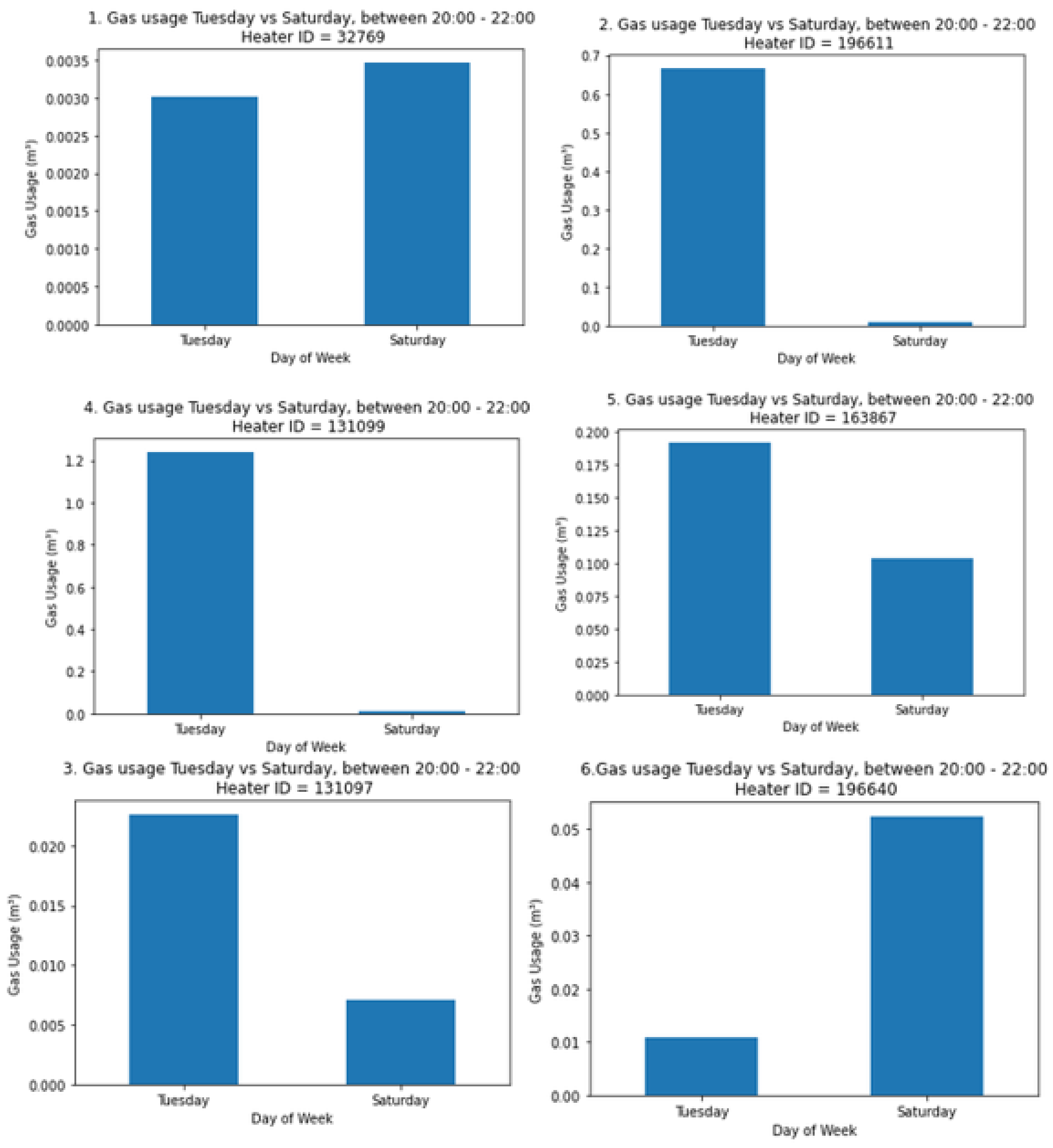
Cutoff = Y < (gas_usage.mean() – 1SD) / 2

This filtering rule did select a subset (n = 8,197) with 5,434 unique heater IDs, which is 22% of the initial data frame.

This choice is made based on the earlier mentioned research by HIER (2018). This research states that 19% of the residents have an underfloor heating system, which could be fueled by electricity, and 8% of the residents have a wood stove. Although these percentages are a very rough indicator for what this result would expect to find, this filtering rule seems to have a percentage which is close to the results of the HIER research. The second filtering rule was selected as being the best rule. This rule was chosen because it has the most strict cutoffs. Beside the expected percentages, it seems to make sense, to expect that the gas usage will be low on average. An important thing to mention is that the filtering rule does only have to select a heater once. After selecting it once, the ID is added to the subset and is therefore selected for classification 1. So, strict cutoffs for gas usage do make sense.

Observations within this subset are likely to be correctly classified as 1 (having an external heating source). From this dataset, the IDs were extracted. The next step was comparing Tuesday with Saturday. This comparison aimed to find out if a resident has an irregular heating pattern, or a low average which is more consistent over time. This inspection was done manually. The first six heater IDs from the extracted list were chosen to be visualized (figure 2).

Figure 13:First 6 heater IDs extracted from the newly created subset. Figures compare gas usage between 20:00 – 22:00 from Tuesday and Saturday.



From figure 2, especially the plots 2 and 4 are remarkable. Both plots show a big difference between Tuesday and Saturday. Based on these observations, it is very likely that we can detect external heating sources. For the other plots, it is very likely that these residents are fueled by a non-gas external heating source, because their gas usage is very low compared to the mean. Although these conclusions seem reasonable, it will never be certain without any reference data.

The research within the framework of a new energy label does not stop here. A further step that could help to scale up the classification of wood stove patterns, could be done by subtracting the Saturday Y value from the Tuesday Y value. Values that are relatively far away from 0 have relatively high differences between these two days. This could indicate a wood stove gas pattern. But, it seems a good first step to classify more accurately before further specifying the labels.

Discussion

Base model

This research aimed to find the answer to the question "Can we build a base model to predict gas usage?". The answer to this question seemed to be "yes". In the process of this research, a random forest regressor model was used to predict resident gas usage. Although this model did not predict quite accurately, the results give a hopeful depiction of what is possible with data and gas usage prediction. Because this model had no baseline or reference point, it is hard to classify its performance as good or bad. Despite this baseline absence, the explained variance gives an indication of its usability and the predictors can be compared among each other.

The regressor model had an explained variance of almost 50 percent. This statistic tells that 50 percent of the variance in the outcome could be explained by the features in the model. This result is interpreted as high. Although these predictors seem to explain quite some variance, there is also more than 50 percent non explained variance. This is according to what was expected. There are a lot of factors that influence one's gas usage. Also, there is a lot of difference between households and their gas usage behavior, which will be highly complex to predict accurately.

For the created regressor model for this research, the feature importance can give sight into what had a substantial impact on the prediction accuracy. Two features scored high on average, which were: year and sun. According to the correlations (see fig x), sun was expected to be a substantial predictor. Sun is ofcourse, highly correlated with temperature. This temperature feature had not a substantial influence on the prediction. Random Forest is a model that is well known for its way of dealing well with collinearity, which was the main consideration to choose for this model. This way of dealing with collinearity is a possible explanation for the high influence of sun and lower influence of temperature. Because they are collinear, they cannot both have a high influence. A more remarkable observation was the influence of year on the prediction within the model. Year is a derived variable from the observation timestamp. Observation year could have an influence because some years will have a lower temperature on average than others. This will probably influence the mean gas usage.

One of the main limitations of this research is the varying amount of records for the year values. Every ID’s year's value is the average of all the 24hour values from this ID, within a certain year. This varying amount of records could have an influence on the results. For example, for some heater ID’s only data points from the winter period are missing. This would give biased estimates and has influence on the averages that were used to train the model. A proposed solution could be to make a distribution to see which values are missing, if for instance the missing values are in a heating curve the slope value of this curve could be used to calculate the missing value. Further research should find a solution for this problem.

The random forest regressor model has the property of being an easily applicable model, which often works well, also without any parameter tuning. This property helped to overcome the problem of complex steps to make an algorithm work. At the same time, the prediction accuracy could probably be improved by tuning these parameters. This parameter tuning could be a good followup research topic.

This explorative study was the first attempt to create a digital energy label. This label aims to predict gas usage, compare residence and possibly also advice on how to improve sustainable housing.

Classification Task

This research aimed to answer the question “Can we classify if there are external heating sources?”. The answer to this question seems to be “yes”. Low regular gas usage is likely to be caused by external heating sources, like an electrical fueled underfloor heater. Also, the irregular gas usage patterns are an indication of external heating, for example by a wood stove. This specification within external heating sources could lead in the future to classify even more accurately. Although these interpretations of the data seem reasonable, there is a high uncertainty caused by the fact that there is no reference data available. Unfortunately, the classification rule could also not be applied over the whole hourly dataset, because of the limited amount of time. Therefore, the improvement on the random forest regressors could not be assessed. Nevertheless, the visual inspection of the IDs that were selected showed a promising look on the possibility of classifying external heating sources. At the same time, it is not expected that accurately classifying external heating sources will increase the explained variance drastically. The variance between resident gas usage does probably depend on a huge set of factors that possibly will be very complex to determine.

A strong point of this research is the interval of the timepoints. Hourly based data frames could determine different gas usage patterns way more accurately than 24hour data. The availability of this data frame was a great opportunity to do these analyses with this level of accuracy. Also, further specifying classes within the classification of external heating sources would be a lot harder without this hourly based dataset.

This study also has some limitations. The first limitation is the fact that the used data is a sample of the initial hourly dataset. 1 % percent of the initial dataset was randomly picked for creating the sample dataset that was used for these analyses. Reducing the data was needed to properly work with it. This reduction obviously resulted in an incomplete number of observations per heater ID. Due to these incomplete records, the results could be biased. This problem could have been passed by using other filters in the early stages of the research. Unfortunately, due to the limited amount of time we had to work with this dataset. Future research should focus on better selection criteria if they want to further improve this classification task.

A second limitation of this research was the formula for determining Y. This formula uses the overall gas usage mean. Using this value, does not take into account that bigger houses on average need more gas to get warm, than smaller houses. As a result, the accuracy of the classification is likely to be less accurate for smaller houses. Future research that focuses on a classification task like this should try to avoid this drawback. A possible solution could be using gas usage per m^2. A drawback of this method is that there is no linear relation between surface and gas usage. This nonlinear relation will result in an advantage for bigger houses, instead of smaller houses.

This classification task is part of a movement towards a data drive energy label. Classifying external heating sources is a piece in the puzzle that tries to create a new method for energy labelling. This new method aims to predict and compare residence energy consumption controlled for consumer behavior. According to what was found in the development of the base model, some external factors, like the amount of sun, seem to have an influence on gas usage. Detecting and classifying different heating patterns is likely to be a first step in controlling for consumer behavior. Future research and development of this data drive energy label will show the capabilities of this system which possibly could lead to the replacement of the nowadays energy label.

This explorative research tried to answer different questions based on datasets provided by Intergas. The methods in this research were chosen based on the experience with data analysis we gained from our courses during the master Applied Data Science. Also, every step within this research is discussed and performed in consideration with our closely involved supervisor.

Reference

Berkland, S. M. (2014). A Comparison of American, Canadian, and European Home Energy Performance in Heating Dominated–Moist Climates Based on Building Codes.

Boonekamp, P. G. (2007). Price elasticities, policy measures and actual developments in household energy consumption–A bottom up analysis for the Netherlands. *Energy Economics*, *29*(2), 133-157.

Bozsaky, D. (2010). The historical development of thermal insulation materials. *Periodica Polytechnica Architecture*, *41*(2), 49-56.

van den Brom, P., Meijer, A., & Visscher, H. (2018). Performance gaps in energy consumption: household groups and building characteristics. *Building Research & Information*, *46*(1), 54-70.

Energielabel. (n.d.). *Veelgestelde vragen*. https://www.energielabel.nl/woningen/veelgestelde-vragen/

European Commission. (2020, 17 Febuary). *In focus: Energy effiency in buildings.* https://ec.europa.eu/info/news/focus-energy-efficiency-buildings-2020-feb-17_en/

European commission. (n.d.). *2050 long term strategy.* www.ec.europa.eu/clima/policies/strategies/2050_en/#tab-0-0

HIER klimaatbureau. (2018, 5 Octobre). *Onderzoeksresultaten Verwarmings-gewoonten van Nederlanders.* https://www.hier.nu/uploads/inline/20181005%20Onderzoeksresultaten%20Verwarmingsgewoonten_HIER%20klimaatbureau.pdf

Intergas. (2018, May). *Kombi Kompakt.* https://www.intergas-verwarming.nl/app/uploads/2018/01/Installatievoorschrift-Kombi-Kompakt-HRE-88557803.pdf

Intergas. (n.d.). *Over Intergas.* https://www.intergas-verwarming.nl/consument/over-intergas/\

Jeeninga, H., Uyterlinde, M. A., & Uitzinger, J. (2001). Energieverbruik van energiezuinige woningen. Effecten van gedrag en besparingsmaatregelen op de spreiding in en de hoogte van het reële energieverbruik.

Martens, S., & Spaargaren, G. (2005). The politics of sustainable consumption: the case of the Netherlands. *Sustainability: science, practice and policy*, *1*(1), 29-42.

Majcen, D., Itard, L. C. M., & Visscher, H. (2013). Theoretical vs. actual energy consumption of labelled dwellings in the Netherlands: Discrepancies and policy implications. *Energy policy*, *54*, 125-136.

Majcen, D., & Itard, L. (2014a). Relatie tussen energielabel, werkelijk energiegebruik en CO2-uitstoot van Amsterdamse corporatiewoningen. *Delft University of Technology (OTB): Rekenkamer Metropool Amsterdam. Von http://resolver. tudelft. nl/uuid: b0b73c48‐4413‐4dda‐8b1b‐748cf65a534b abgerufen*.

Majcen, D., & Itard, L. C. M. (2014b). Relatie tussen huishoudenskenmerken en-gedrag, energielabel en werkelijk energiegebruik in Amsterdamse corporatiewoningen.

Majcen, D., Itard, L., & Visscher, H. (2015). Statistical model of the heating prediction gap in Dutch dwellings: Relative importance of building, household and behavioural characteristics. *Energy and Buildings*, *105*, 43-59.

ODYSSEE. (2015, June). *Energy Efficiency Trends and Policies in the Household and Tertiary Sectors.* https://www.odyssee-mure.eu/publications/archives/energy-efficiency-trends-policies-buildings.pdf

Radar. (2021, 3 March). *Tot 500 euro voor het energielabel, maar wat heb je eraan?* https://radar.avrotros.nl/uitzendingen/gemist/item/tot-500-euro-voor-het-energielabel-maar-wat-heb-je-eraan/

RVO. (2020a, March). *Opnameformulier behorend bij het opnameprotocol.* https://www.rvo.nl/sites/default/files/2020/10/opnameformulier-behorend-bij-het-opnameprotocol-nta-8800-versie-maart-2020.pdf

RVO. (2020b, 29 May). *Energielabels op basis van NTA 8800 bij bouwaanvraag EPC*. https://www.rvo.nl/sites/default/files/2020/06/energielabels-op-basis-van-nta-8800-bij-bouwaanvraag-epc.pdf

RvO. (2021, 29 June). *Energielabel woningen.* https://www.rvo.nl/onderwerpen/duurzaam-ondernemen/gebouwen/wetten-en-regels/bestaande-bouw/energielabel-woningen

Santin, O. G. (2011). Behavioural patterns and user profiles related to energy consumption for heating. *Energy and Buildings*, *43*(10), 2662-2672.

Yun, G. Y., & Steemers, K. (2011). Behavioural, physical and socio-economic factors in household cooling energy consumption. *Applied Energy*, *88*(6), 2191-2200.

Yunus A.. Çengel, & Ghajar, A. J. (2020). *Heat and Mass Transfer: Fundamentals [and] Applications*. McGraw-Hill Education.

Appendix Code

Df_merging

July 2, 2021

```
[23]: import os
      os.environ["SPARK_LOCAL_DIRS"] = "/home/jovyan/work/tmp"
```

```
[24]: from pyspark import SparkContext
      from pyspark.sql import SparkSession, Row, DataFrameWriter, functions as sf
      from pyspark.sql.functions import *

      #Connect to spark context and create session
      context = SparkContext('local[*]')
      session = SparkSession(context)

      # Spark notebook home (/home/jovyan/data is mounted to server /data)
      datadir = '/home/jovyan/work/data'
```

0.1 DF: Gas use hourly

```
[3]: #loading whole df_data_ig
     ig_hourly = session.read.json(datadir+'/ig-gasuse-hourly.json/*.json.gz') #␣
      ↪whole set

     #select necessary cols
     ig_hourly = ig_hourly.select("heater_id", "t_act", "t_set", "TimeKey", "Wijk")
```

```
[4]: #print schema and top rows
     ig_hourly.printSchema()
     ig_hourly.show(5)
```

```
root
 |-- heater_id: long (nullable = true)
 |-- t_act: double (nullable = true)
 |-- t_set: double (nullable = true)
 |-- TimeKey: string (nullable = true)
 |-- Wijk: string (nullable = true)

+---------+-----+-----+----------+----+
|heater_id|t_act|t_set|   TimeKey|Wijk|
+---------+-----+-----+----------+----+
|    10741|21.89| 22.0|2018110901|1400|
|    10741|21.95| 22.0|2018110902|1400|
|    10741|22.06| 22.0|2018110903|1400|
|    10741|22.05| 22.0|2018110904|1400|
|    10741|21.97| 22.0|2018110905|1400|
+---------+-----+-----+----------+----+
only showing top 5 rows
```

[5]:
```
#create new col: dateday, as key to merge with other df's
ig_hourly = ig_hourly.withColumn("date_day", col("TimeKey").substr(0,8))
ig_hourly = ig_hourly.withColumn("hour", col("TimeKey").substr(8,2))
```

[6]:
```
ig_hourly_24 = ig_hourly.groupBy("heater_id", "date_day").agg(avg("t_act"),␣
 ↪avg("t_set"))
ig_hourly_24.show(3)
```

```
+---------+--------+------------------+----------+
|heater_id|date_day|        avg(t_act)|avg(t_set)|
+---------+--------+------------------+----------+
|    10741|20181208|            20.705|  21.02375|
|    10741|20190426| 22.25583333333333|      22.0|
|    10741|20190820|22.980416666666674|   14.4425|
+---------+--------+------------------+----------+
only showing top 3 rows
```

0.2 DF: KNMI data

[8]:
```
#loading whole KNMI df
df_knmi = session.read.json(datadir+'/knmi-hourly.json/*.json.gz')

#get column date_day and hour from TimeKey
df_knmi = df_knmi.withColumn("date_day", col("TimeKey").substr(0,8))

#show
df_knmi.show(4)
```

```
+----------+----+------------------+-----------------+------------------+-------
-----------+--------+
|   TimeKey|Wijk|              rain|              sun|              temp|
wind|date_day|
+----------+----+------------------+-----------------+------------------+-------
-----------+--------+
|2011020220|1001|               0.0|              0.0|
35.90114600398778|101.62519242926264|20110202|
|2011020410|1001|0.9808532599793152|5.792962250180368|
82.17807402757347|121.08183753807421|20110204|
|2011020423|1001|                 0.0|                 0.0|
91.16874426977826|147.29297003266316|20110204|
|2011021404|1001|                 0.0|                 0.0|29.964811250901842|
63.69556992745895|20110214|
+----------+----+--------------------+--------------------+-------------------+-------
-----------+--------+
only showing top 4 rows
```

[10]:
```
# GroupBy Wijk & date_day and get average knmi data
df_avg_knmi = df_knmi.groupBy("Wijk", "date_day").avg("rain", "sun", "temp",␣
↪"wind")

#show
df_avg_knmi.show(4)
```

```
+----+--------+--------------------+------------------+-------------------+-----
-------------+
|Wijk|date_day|           avg(rain)|          avg(sun)|          avg(temp)|
avg(wind)|
+----+--------+--------------------+------------------+-------------------+-----
-------------+
|1001|20110313| 0.03295133200407028|19.038369723259674|
93.00903432208953|33.929621648990526|
|1001|20130222|-0.04352160445389…|  27.1564311402127|-13.227166279583725|
66.24841480430572|
|1002|20131202|                 0.0|12.561415625659238|
52.59694793334723|15.439418152891578|
|1003|20140430| 0.48958753014916007| 67.13607528992402|
136.6616932569032|31.291684519422457|
+----+--------+--------------------+------------------+-------------------+-----
-------------+
only showing top 4 rows
```

0.3 DF: Intergas_raw

[ ]:
```
#loading whole df_data_ig_raw
ig_raw = session.read.json(datadir+'/intergas-raw.json/*.json.gz')
ig_raw.printSchema()
ig_raw.show()
```

[ ]:
```
# select necessary columns
ig_raw_selection = ig_raw.select("stats_24h.gasmeter_ch_24h", "stats_24h.
↪gasmeter_dhw_24h", "date", "date_day", "heater_id")

#filter: gas_ch >0 & gas_dhw > 0
```

```
ig_raw_selection = ig_raw_selection.filter((ig_raw_selection.gasmeter_ch_24h[0]␣
↪>= 0) & (ig_raw_selection.gasmeter_dhw_24h[0] >= 0))

# show
ig_raw_selection.show(5)
ig_raw_selection.printSchema()
```

0.4 DF: ig_heater-info

[ ]:
```
df_heater = session.read.csv(datadir+'/ig-heater-info.csv', header=True)
df_heater.printSchema()
df_heater.show(3)
```

0.5 Join data sets

[ ]:
```
#merge df_data_ig_raw_selection & df_data_heater
merge1 = ig_raw_selection.join(df_heater, ["heater_id"], how='inner')
merge1.show(3)
merge1.printSchema()
```

[ ]:
```
merge2 = merge1.join(ig_hourly_24, ["heater_id", "date_day"], how='inner')
merge2.show(3)
merge2.printSchema()
```

[ ]:
```
df = merge2.join(df_avg_knmi, ['Wijk','date_day'], how='inner')
df.printSchema()
df.show(3)
```

[ ]:
```
df.count()
```

[ ]:
```
#save new dataset
df.write.format('json').save("/home/jovyan/work/data/df_24hour")
```

0.6 Samples from df

[ ]:
```
# get samples do test anlaysis
# 3 samples with 10% of total dataset

#sample 1
df_sample_2 = df.sample(False, .1, seed=102)
df_sample_2.printSchema()
df_sample_2.write.format('json').save("/home/jovyan/work/data/df_sample_2") #␣
↪save

#sample 3
df_sample_3 = df.sample(False, .1, seed=103)
```

```
df_sample_3.printSchema()
df_sample_3.write.format('json').save("/home/jovyan/work/data/df_sample_3") #␣
 ↪save



#sample 1
df_sample_4 = df.sample(False, .1, seed=104)
df_sample_4.printSchema()
df_sample_4.write.format('json').save("/home/jovyan/work/data/df_sample_4") #␣
 ↪save
```

```
root
 |-- wijk: string (nullable = true)
 |-- date_day: string (nullable = true)
 |-- heater_id: long (nullable = true)
 |-- gasmeter_ch_24h: array (nullable = true)
 |    |-- element: double (containsNull = true)
 |-- gasmeter_dhw_24h: array (nullable = true)
 |    |-- element: double (containsNull = true)
 |-- date: string (nullable = true)
 |-- pandbouwjaar: string (nullable = true)
 |-- oppervlakteverblijfsobject: string (nullable = true)
 |-- avg(t_act): double (nullable = true)
 |-- avg(t_set): double (nullable = true)
 |-- avg(rain): double (nullable = true)
 |-- avg(sun): double (nullable = true)
 |-- avg(temp): double (nullable = true)
 |-- avg(wind): double (nullable = true)
```

[1]:
```
# session.stop()
# context.stop()
```

EDA2

July 2, 2021

[1]:
```
import os
os.environ["SPARK_LOCAL_DIRS"] = "/home/jovyan/work/tmp"

from pyspark import SparkContext
from pyspark.sql import SparkSession, Row, DataFrameWriter, functions as sf
from pyspark.sql.functions import *

#Connect to spark context and create session
context = SparkContext('local[*]')
session = SparkSession(context)

# Spark notebook home (/home/jovyan/data is mounted to server /data)
datadir = '/home/jovyan/work/data'
```

[2]:
```
datadir = '/home/jovyan/work/data'
df = session.read.json(datadir+'/df_sample_4/*.json')
```

[3]:
```
from pyspark.sql.functions import *

# add col temp diff
df = df.withColumn('avg(temp)/10', df['avg(temp)']/10)

#filter: t_act < 25
df = df.filter(df['avg(t_act)'] <= 26)
df = df.filter(df['avg(t_act)'] >= 10)
df = df.withColumn('temp_diff', df['avg(t_act)'] - df['avg(temp)/10'])

#filter: t_set < 25
df = df.filter(df['avg(t_set)'] <= 26)

#filter temp = max(temp_diff, 0)
df = df.withColumn('temp_diff', when(col('temp_diff') < 0, 0).
 ↪otherwise(col('temp_diff')))

# add gas outcomes as type double and filter < 40
    # gas_ch
df = df.withColumn('gas_ch', df['gasmeter_ch_24h'].getItem(0))
df = df.filter(df.gas_ch < 40)

    # gas_dhw
df = df.withColumn('gas_dhw', df['gasmeter_dhw_24h'].getItem(0))
df = df.filter(df.gas_dhw < 40)

# change col types
df = df.withColumn("oppervlakteverblijfsobject",␣
 ↪df['oppervlakteverblijfsobject'].cast("double"))
df = df.withColumn("pandbouwjaar", df['pandbouwjaar'].cast("integer"))
df = df.withColumn("wijk", df['wijk'].cast('integer'))

# get col year
df = df.withColumn('year', col('date_day').substr(1, 4))

# get col month
df = df.withColumn('month', col('date_day').substr(5, 2))

# get col gas use m2
df = df.withColumn('total_gas_m2', (df.gas_ch + df.gas_dhw)/df.
 ↪oppervlakteverblijfsobject)

# filter: oppervalkteverblijfsobjct >= 40 & < 600
df = df.filter(df.oppervlakteverblijfsobject >= 40)
df = df.filter(df.oppervlakteverblijfsobject < 600)

# drop unnecessary cols
cols = ['date', 'gasmeter_ch_24h', 'gasmeter_dhw_24h']
df = df.drop(*cols)
df.printSchema()
```

```
root
 |-- avg(rain): double (nullable = true)
 |-- avg(sun): double (nullable = true)
 |-- avg(t_act): double (nullable = true)
 |-- avg(t_set): double (nullable = true)
 |-- avg(temp): double (nullable = true)
 |-- avg(wind): double (nullable = true)
 |-- date_day: string (nullable = true)
 |-- heater_id: long (nullable = true)
 |-- oppervlakteverblijfsobject: double (nullable = true)
 |-- pandbouwjaar: integer (nullable = true)
 |-- wijk: integer (nullable = true)
 |-- avg(temp)/10: double (nullable = true)
 |-- temp_diff: double (nullable = true)
 |-- gas_ch: double (nullable = true)
 |-- gas_dhw: double (nullable = true)
 |-- year: string (nullable = true)
 |-- month: string (nullable = true)
 |-- total_gas_m2: double (nullable = true)
```

[4]:
```
#get months jan. and feb.
df_winter = df.filter((df.month == '01') | (df.month == '02'))
df_winter.show(3)
```

```
+-------------------+------------------+------------------+------------------+--
-----------------+------------------+--------+---------+------------------------
--+------------+-----+--------------------+------------------+-------------------
+------------------+----+-----+--------------------+
|          avg(rain)|          avg(sun)|        avg(t_act)|        avg(t_set)|
avg(temp)|
avg(wind)|date_day|heater_id|oppervlakteverblijfsobject|pandbouwjaar| wijk|
avg(temp)/10|         temp_diff|             gas_ch|
gas_dhw|year|month|        total_gas_m2|
+-------------------+------------------+------------------+------------------+--
-----------------+------------------+--------+---------+------------------------
--+------------+-----+--------------------+------------------+-------------------
+------------------+----+-----+--------------------+
|                0.0|15.066755816416586|             21.38|              21.5|
9.682061231962802|41.490809270779884|20210114|    88503|
170.0|        2002| 1003| 0.9682061231962802| 20.41179387680372|
6.373199999999997|2.5860999999999876|2021|   01| 0.05270176470588226|
|                0.0|15.066755816416586|20.509999999999994|              15.0|
9.682061231962802|41.490809270779884|20210114|   173555|
191.0|        2005| 1003| 0.9682061231962802|19.541793876803716|0.08479999999985
921|1.4487000000000307|2021|   01|0.008028795811517749|
|-0.3218487898902769|
8.142366768458983|19.144166666666667|16.252916666666668|-13.413290924220258|
65.76885105295825|20170208|    53937|                      93.0|
1957|10601|-1.3413290924220258|20.485495759088693|  9.056500000000142|
0.877900000000011|2017|   02| 0.10682150537634573|
+-------------------+------------------+------------------+------------------+--
-----------------+------------------+--------+---------+------------------------
--+------------+-----+--------------------+------------------+-------------------
+------------------+----+-----+--------------------+
only showing top 3 rows
```

[5]:
```
df.describe().show()
```

```
+-------+--------------------+--------------------+-------------------+------------
--------+------------------+------------------+---------------------+------------
------+--------------------------+------------------+-----------------+---------
---------+------------------+-----------------+------------------+--------------
```

```
----+------------------+--------------------+
|summary|          avg(rain)|           avg(sun)|        avg(t_act)|
avg(t_set)|         avg(temp)|         avg(wind)|           date_day|
heater_id|oppervlakteverblijfsobject|      pandbouwjaar|             wijk|
avg(temp)/10|         temp_diff|            gas_ch|           gas_dhw|
year|              month|       total_gas_m2|
+-------+-------------------+--------------------+------------------+------------
--------+------------------+-------------------+--------------------+------------
------+--------------------------+-------------------+------------------+---------
---------+------------------+------------------+-------------------+--------------
----+------------------+--------------------+
|  count|            2605965|            2588867|           2614462|
2614462|           2610253|           2614462|           2614462|
2614462|                   2614462|           2614462|           2614462|
2610253|           2610253|           2614462|           2614462|
2614462|           2614462|              2614462|
|   mean| 0.6931115783215736|  36.41823030408528|20.623722297322303|
16.7576500133664| 90.92280309067996| 43.52875942534622|2.0187832447827507E7|
91796.53089469268|
143.496526627658|1965.1157343269858|66414.51303442162|
9.092280309068038|11.539991806875015|3.338218177927236|0.7693700813016233|
2018.715729278146|6.5941673659819875|0.034840749029792735|
| stddev| 1.4719286503170799| 31.858157405562267|  2.22202899641818|
3.088474349773235|57.609488249718254|21.310360881541918|
13176.763213186694|61660.150918615254|        395.32001428445034|
63.42263794761144| 57914.4196659706| 5.760948824971834|4.9730611498692365|4.2885
17020907146|0.6867816314217753|1.3229080360149592| 3.589469216709949|
0.03853067486617739|
|    min|-0.6414749137889136|0.05661141827097143|
10.0|-0.00791666666666…|-79.39315473007132|3.8886266251021944|
20151004|              1499|                      40.0|              1005|
300|-7.939315473007132|              0.0|               0.0|              0.0|
2015|                01|                  0.0|
|    max| 31.158154904429136|  130.9650152513161|              26.0|
26.0|308.27342108891355|188.09149485061832|            20210201|
204776|                  68353.0|              2020|
197820|30.827342108891354| 32.07211049680499|39.98979999999938|
39.72479999999996|              2021|                12|  0.8909688888888568|
+-------+-------------------+--------------------+------------------+------------
--------+------------------+-------------------+--------------------+------------
------+--------------------------+-------------------+------------------+---------
---------+------------------+------------------+-------------------+--------------
----+------------------+--------------------+
```

[6]:
```
# from pyspark.sql.functions import *
# df_winter.groupBy('heater_id').agg(avg('gas_ch'), avg('temp'))
```

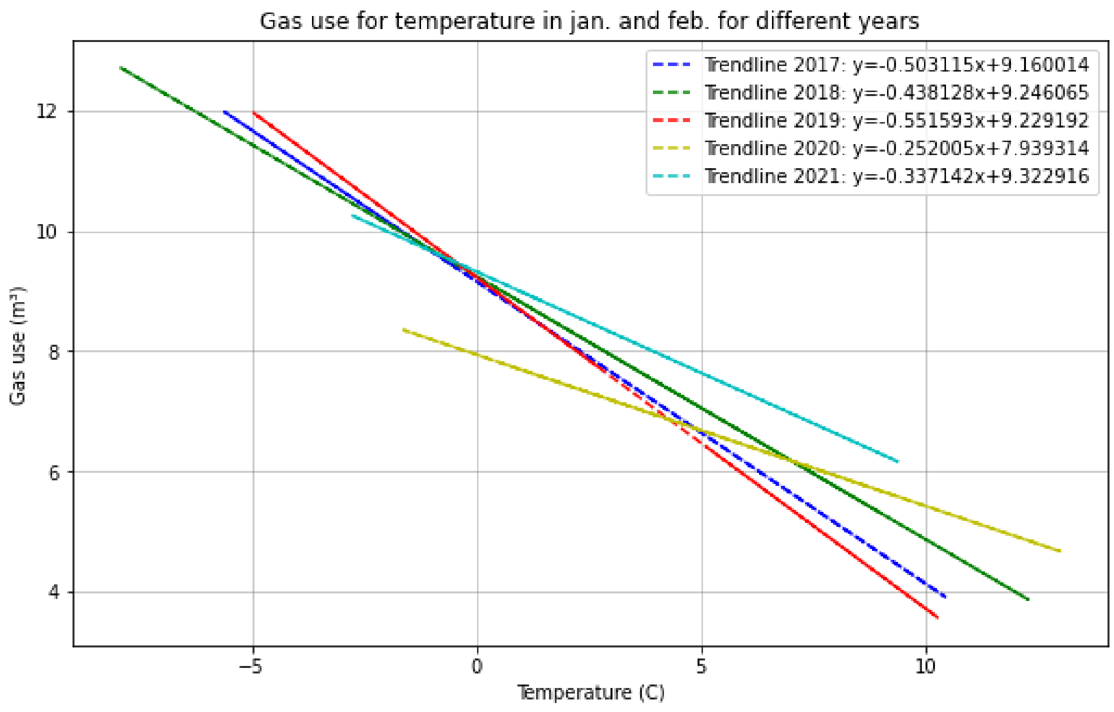
[7]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

[8]:
```
pd_winter = df_winter.toPandas()
```

[9]:
```
fig = plt.figure(figsize= [10,6])
ax = plt.axes()
ax.set_title("Gas use for temperature in jan. and feb. for different years")

# df, x, y year == 2017
df_xy = pd_winter[pd_winter.year == '2017']
df_xy = df_xy.dropna()
x = df_xy['avg(temp)/10'] #filter for year
y = df_xy['gas_ch']     #filter for year

#df, x,  y: year == 2018
df_xy1 = pd_winter[pd_winter.year == '2018']
df_xy1 = df_xy1.dropna()
x1 = df_xy1['avg(temp)/10'] #filter for year
y1 = df_xy1['gas_ch']     #filter for year

# df, x, y: year == 2019
df_xy2 = pd_winter[pd_winter.year == '2019']
df_xy2 = df_xy2.dropna()
x2 = df_xy2['avg(temp)/10'] #filter for year
y2 = df_xy2['gas_ch']     #filter for year

# df, x, y: year == 2020
df_xy3 = pd_winter[pd_winter.year == '2020']
df_xy3 = df_xy3.dropna()
x3 = df_xy3['avg(temp)/10'] #filter for year
y3 = df_xy3['gas_ch']     #filter for year

# df, x, y: year == 2020
df_xy4 = pd_winter[pd_winter.year == '2021']
df_xy4 = df_xy4.dropna()
x4 = df_xy4['avg(temp)/10'] #filter for year
y4 = df_xy4['gas_ch']     #filter for year


#scatter and trendline: 2017
# ax.scatter(x2, y2, alpha=.5)
z = np.polyfit(x, y, 1)
p = np.poly1d(z)
plt.plot(x,p(x),"b--", label= 'Trendline 2017: y=%.6fx+%.6f'%(z[0],z[1]))
```

```
#scatter and trendline: 2018
# ax.scatter(x1, y1, alpha=.5)
z1 = np.polyfit(x1, y1, 1)
p1 = np.poly1d(z1)
plt.plot(x1,p1(x1),"g--", label= 'Trendline 2018: y=%.6fx+%.6f'%(z1[0],z1[1]))

#scatter and trendline: 2019
# ax.scatter(x2, y2, alpha=.5)
z2 = np.polyfit(x2, y2, 1)
p2 = np.poly1d(z2)
plt.plot(x2,p2(x2),"r--", label= 'Trendline 2019: y=%.6fx+%.6f'%(z2[0],z2[1]))

#scatter and trendline: 2020
# ax.scatter(x2, y2, alpha=.5)
z3 = np.polyfit(x3, y3, 1)
p3 = np.poly1d(z3)
plt.plot(x3,p3(x3),"y--", label= 'Trendline 2020: y=%.6fx+%.6f'%(z3[0],z3[1]))

#scatter and trendline: 2021
# ax.scatter(x2, y2, alpha=.5)
z4 = np.polyfit(x4, y4, 1)
p4 = np.poly1d(z4)
plt.plot(x4,p4(x4),"c--", label= 'Trendline 2021: y=%.6fx+%.6f'%(z4[0],z4[1]))

#properties of fig
plt.legend(loc="upper right")
ax.yaxis.grid(True, linestyle='-', which='major', color='grey', alpha=0.5)
ax.xaxis.grid(True, linestyle='-', which='major', color='grey', alpha=0.5)
ax.set_xlabel('Temperature (C)')
ax.set_ylabel('Gas use (m³)')
```

```
[9]: Text(0, 0.5, 'Gas use (m³)')
```



```
[10]: pd_year = df.toPandas()
      pd_year.head()
```

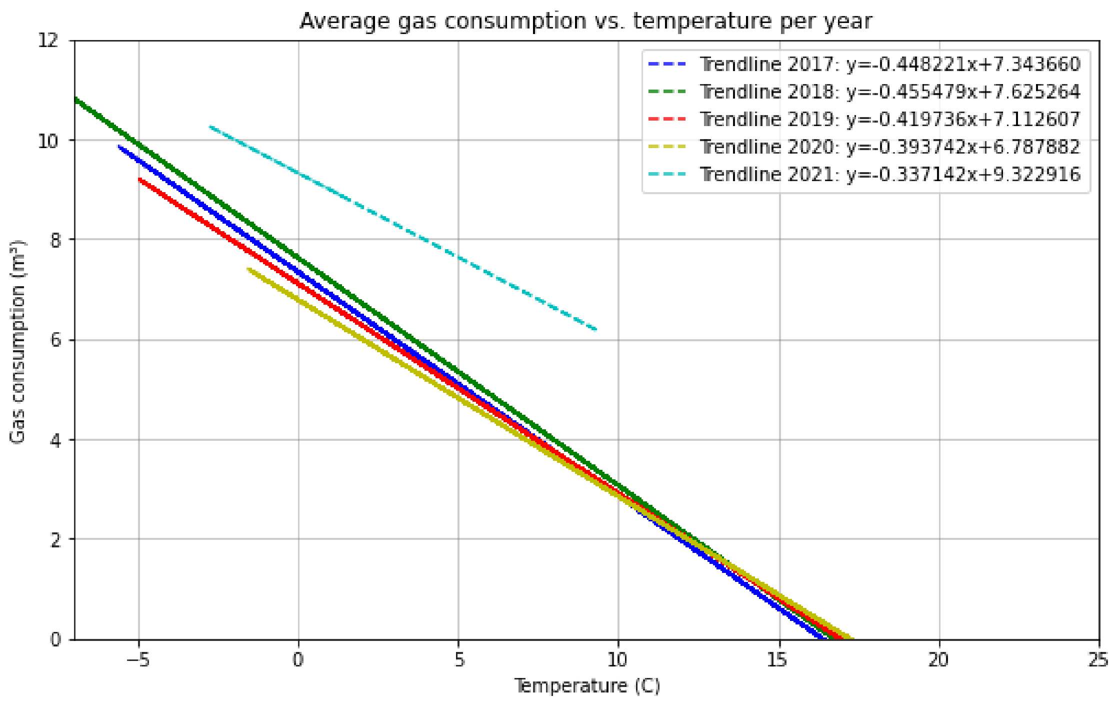
```
[10]:    avg(rain)    avg(sun)  avg(t_act)  avg(t_set)   avg(temp)   avg(wind)  \
      0   0.000000   49.428351   18.926250        17.5   64.497099   36.796227
      1   0.000000  107.616097   23.171667        16.0  192.848343   31.043363
      2   3.040219   11.315445   19.878750        16.0  161.789993   77.275430
      3  -0.008935    6.165509   22.390417        19.0   71.879050   84.686033
      4   4.274873   37.367628   21.940000        17.0  130.892169  107.099784

         date_day  heater_id  oppervlakteverblijfsobject  pandbouwjaar  wijk  \
      0  20170310      12927                       490.0          1830  1001
      1  20190617      12927                       490.0          1830  1001
      2  20160929      40989                       123.0          1959  1002
      3  20161224      54477                        79.0          1956  1002
      4  20170913      54477                        79.0          1956  1002

         avg(temp)/10  temp_diff  gas_ch  gas_dhw  year month  total_gas_m2
      0      6.449710  12.476540  6.8691   1.4980  2017    03      0.017076
      1     19.284834   3.886832  0.0000   0.2331  2019    06      0.000476
      2     16.178999   3.699751  0.0000   0.8656  2016    09      0.007037
      3      7.187905  15.202512  6.5733   0.8204  2016    12      0.093591
      4     13.089217   8.850783  1.4064   0.6214  2017    09      0.025668
```

[11]:

```
fig = plt.figure(figsize= [10,6])
ax = plt.axes()
ax.set_title("Average gas consumption vs. temperature per year")

# df, x, y year == 2017
df_xy = pd_year[pd_year.year == '2017']
df_xy = df_xy.dropna()
x = df_xy['avg(temp)/10'] #filter for year
y = df_xy['gas_ch']    #filter for year

#df, x,  y: year == 2018
df_xy1 = pd_year[pd_year.year == '2018']
df_xy1 = df_xy1.dropna()
x1 = df_xy1['avg(temp)/10'] #filter for year
y1 = df_xy1['gas_ch']    #filter for year

# df, x, y: year == 2019
df_xy2 = pd_year[pd_year.year == '2019']
df_xy2 = df_xy2.dropna()
x2 = df_xy2['avg(temp)/10'] #filter for year
y2 = df_xy2['gas_ch']    #filter for year

# df, x, y: year == 2020
df_xy3 = pd_year[pd_year.year == '2020']
df_xy3 = df_xy3.dropna()
x3 = df_xy3['avg(temp)/10'] #filter for year
y3 = df_xy3['gas_ch']    #filter for year

# df, x, y: year == 2020
df_xy4 = pd_year[pd_year.year == '2021']
df_xy4 = df_xy4.dropna()
x4 = df_xy4['avg(temp)/10'] #filter for year
y4 = df_xy4['gas_ch']    #filter for year


#scatter and trendline: 2017
# ax.scatter(x2, y2, alpha=.5)
z = np.polyfit(x, y, 1)
p = np.poly1d(z)
plt.plot(x,p(x),"b--", label= 'Trendline 2017: y=%.6fx+%.6f'%(z[0],z[1]))

#scatter and trendline: 2018
# ax.scatter(x1, y1, alpha=.5)
z1 = np.polyfit(x1, y1, 1)
p1 = np.poly1d(z1)
plt.plot(x1,p1(x1),"g--", label= 'Trendline 2018: y=%.6fx+%.6f'%(z1[0],z1[1]))
```

```
#scatter and trendline: 2019
# ax.scatter(x2, y2, alpha=.5)
z2 = np.polyfit(x2, y2, 1)
p2 = np.poly1d(z2)
plt.plot(x2,p2(x2),"r--", label= 'Trendline 2019: y=%.6fx+%.6f'%(z2[0],z2[1]))

#scatter and trendline: 2020
# ax.scatter(x2, y2, alpha=.5)
z3 = np.polyfit(x3, y3, 1)
p3 = np.poly1d(z3)
plt.plot(x3,p3(x3),"y--", label= 'Trendline 2020: y=%.6fx+%.6f'%(z3[0],z3[1]))

# scatter and trendline: 2021
# ax.scatter(x2, y2, alpha=.5)
z4 = np.polyfit(x4, y4, 1)
p4 = np.poly1d(z4)
plt.plot(x4,p4(x4),"c--", label= 'Trendline 2021: y=%.6fx+%.6f'%(z4[0],z4[1]))

#properties of fig
plt.legend(loc="upper right")
ax.yaxis.grid(True, linestyle='-', which='major', color='grey', alpha=0.5)
ax.xaxis.grid(True, linestyle='-', which='major', color='grey', alpha=0.5)
ax.set_xlabel('Temperature (C)')
ax.set_ylabel('Gas consumption (m³)')
ax.set_ylim(0,12)
ax.set_xlim(-7, 25)
```

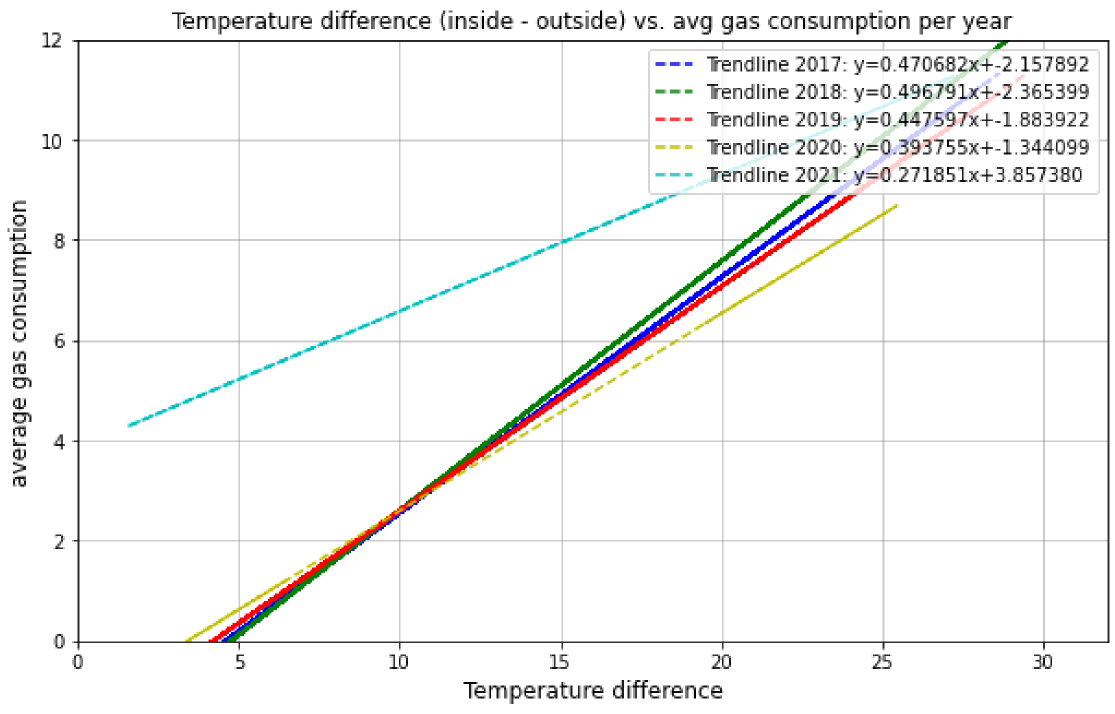
```
[11]: (-7.0, 25.0)
```



[12]:
```
fig = plt.figure(figsize= [10,6])
ax = plt.axes()
ax.set_title("Temperature difference (inside - outside) vs. avg gas consumption␣
 ↪per year", fontsize=12)

# df, x, y year == 2017
df_xy = pd_year[pd_year.year == '2017']
df_xy = df_xy.dropna()
x = df_xy['temp_diff'] #filter for year
y = df_xy['gas_ch']     #filter for year

#df, x,  y: year == 2018
df_xy1 = pd_year[pd_year.year == '2018']
df_xy1 = df_xy1.dropna()
x1 = df_xy1['temp_diff'] #filter for year
y1 = df_xy1['gas_ch']     #filter for year

# df, x, y: year == 2019
df_xy2 = pd_year[pd_year.year == '2019']
df_xy2 = df_xy2.dropna()
x2 = df_xy2['temp_diff'] #filter for year
y2 = df_xy2['gas_ch']     #filter for year

# df, x, y: year == 2020
```

```
df_xy3 = pd_year[pd_year.year == '2020']
df_xy3 = df_xy3.dropna()
x3 = df_xy3['temp_diff'] #filter for year
y3 = df_xy3['gas_ch']    #filter for year

# df, x, y: year == 2020
df_xy4 = pd_year[pd_year.year == '2021']
df_xy4 = df_xy4.dropna()
x4 = df_xy4['temp_diff'] #filter for year
y4 = df_xy4['gas_ch']    #filter for year


#scatter and trendline: 2017
# ax.scatter(x2, y2, alpha=.5)
z = np.polyfit(x, y, 1)
p = np.poly1d(z)
plt.plot(x,p(x),"b--", label= 'Trendline 2017: y=%.6fx+%.6f'%(z[0],z[1]))

#scatter and trendline: 2018
# ax.scatter(x1, y1, alpha=.5)
z1 = np.polyfit(x1, y1, 1)
p1 = np.poly1d(z1)
plt.plot(x1,p1(x1),"g--", label= 'Trendline 2018: y=%.6fx+%.6f'%(z1[0],z1[1]))

#scatter and trendline: 2019
# ax.scatter(x2, y2, alpha=.5)
z2 = np.polyfit(x2, y2, 1)
p2 = np.poly1d(z2)
plt.plot(x2,p2(x2),"r--", label= 'Trendline 2019: y=%.6fx+%.6f'%(z2[0],z2[1]))

#scatter and trendline: 2020
# ax.scatter(x2, y2, alpha=.5)
z3 = np.polyfit(x3, y3, 1)
p3 = np.poly1d(z3)
plt.plot(x3,p3(x3),"y--", label= 'Trendline 2020: y=%.6fx+%.6f'%(z3[0],z3[1]))

# scatter and trendline: 2021
# ax.scatter(x2, y2, alpha=.5)
z4 = np.polyfit(x4, y4, 1)
p4 = np.poly1d(z4)
plt.plot(x4,p4(x4),"c--", label= 'Trendline 2021: y=%.6fx+%.6f'%(z4[0],z4[1]))

#properties of fig
plt.legend(loc="upper right")
ax.yaxis.grid(True, linestyle='-', which='major', color='grey', alpha=0.5)
ax.xaxis.grid(True, linestyle='-', which='major', color='grey', alpha=0.5)
ax.set_xlabel('Temperature difference', fontsize=12)
ax.set_ylabel('average gas consumption', fontsize=12)
ax.set_ylim(0,12)
ax.set_xlim(0,32)
```

```
[12]: (0.0, 32.0)
```



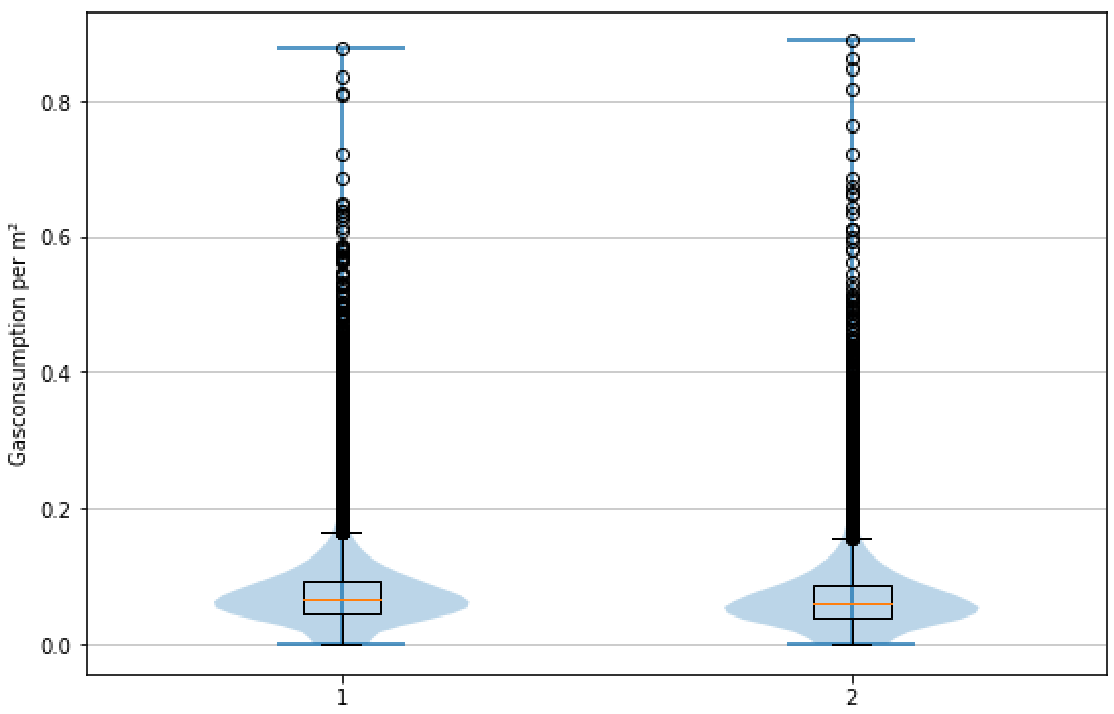
```
[13]: pd_winter
```

```
[13]:         avg(rain)   avg(sun)  avg(t_act)  avg(t_set)  avg(temp)  avg(wind)  \
0        0.000000  15.066756   21.380000   21.500000   9.682061  41.490809
1        0.000000  15.066756   20.510000   15.000000   9.682061  41.490809
2       -0.321849   8.142367   19.144167   16.252917 -13.413291  65.768851
3       -0.321849   8.142367   19.312083   17.500000 -13.413291  65.768851
4        1.355774   6.997916   20.049583   17.000000  27.089749  56.528794
…             …         …          …          …          …          …
457052   0.000000   3.992979   17.539167   17.964583  -3.960250  16.326736
457053   0.000000   3.992979   20.798750   20.497500  -3.960250  16.326736
457054   0.127165   1.879825   19.134583   16.789167  11.616213  96.030642
457055   0.163255   2.370457   16.526667   12.748750   4.753975  48.011423
457056   0.000000   1.764178   17.685000   17.500000   5.646814  33.239672

        date_day  heater_id  oppervlakteverblijfsobject  pandbouwjaar   wijk  \
0       20210114      88503                       170.0          2002   1003
1       20210114     173555                       191.0          2005   1003
```

```
2       20170208      53937                          93.0          1957  10601
3       20170208      56399                          90.0          1962  10601
4       20190128      99315                         152.0          1968  10602
…           …          …                        …           …     …
457052  20170122       8324                         115.0          1978  99522
457053  20170122      12816                          70.0          1975  99522
457054  20160126       8735                         214.0          2002  99524
457055  20180119      79825                          76.0          1998  99538
457056  20200120     194488                         593.0          1996  99561

        avg(temp)/10  temp_diff   gas_ch  gas_dhw  year month  total_gas_m2
0           0.968206  20.411794   6.3732   2.5861  2021    01      0.052702
1           0.968206  19.541794   0.0848   1.4487  2021    01      0.008029
2          -1.341329  20.485496   9.0565   0.8779  2017    02      0.106822
3          -1.341329  20.653412  12.5593   0.7018  2017    02      0.147346
4           2.708975  17.340608   8.6957   0.7347  2019    01      0.062042
…                …         …        …        …     …   …            …
457052     -0.396025  17.935192   9.0891   0.7576  2017    01      0.085623
457053     -0.396025  21.194775  18.4499   1.8151  2017    01      0.289500
457054      1.161621  17.972962   4.3184   1.5474  2016    01      0.027410
457055      0.475398  16.051269   5.1254   0.5776  2018    01      0.075039
457056      0.564681  17.120319  13.2843   0.2582  2020    01      0.022837

[457057 rows x 18 columns]
```

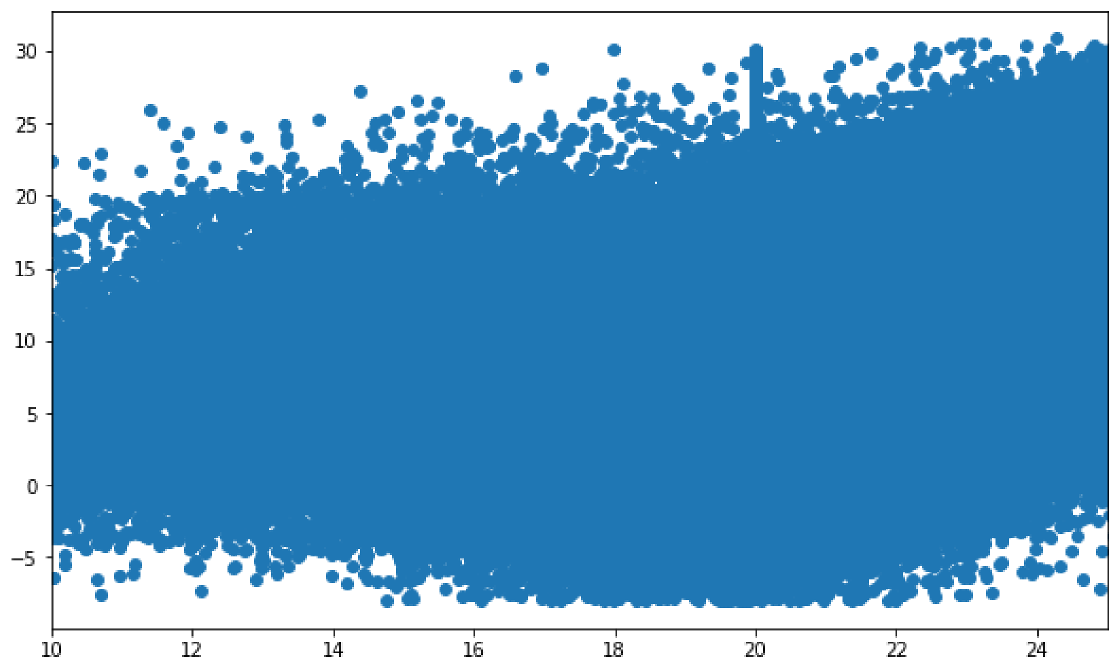
[14]:
```python
jan = pd_winter[pd_winter.month == '01']['total_gas_m2']
feb = pd_winter[pd_winter.month == '02']['total_gas_m2']
winter = [jan, feb]

fig, ax1 = plt.subplots(figsize=(9,6))
ax1.set_title("")

ax1.boxplot(winter)
ax1.violinplot(winter)
ax1.yaxis.grid(True, linestyle='-', which='major', color='grey', alpha=0.5)
# ax1.set_xticks({1: 'Jan', 2: 'Feb.'})
ax1.set_ylabel('Gasconsumption per m²')
```

[14]:
```
Text(0, 0.5, 'Gasconsumption per m²')
```



```
[15]: fig = plt.figure(figsize= [10,6])
      ax = plt.axes()
      # ax.set_title("Temperature difference (inside - outside) vs. avg gas␣
       ↪consumption per year", fontsize=12)

      # df, x, y year == 2017
      xy = pd_year
      xy = xy.dropna()
      x = xy['avg(t_act)'] #filter for year
      y = xy['avg(temp)/10']     #filter for year


      z = np.polyfit(x, y, 1)
      p = np.poly1d(z)
      # plt.plot(x,p(x),"y--")
      plt.scatter(x,y)
      ax.set_xlim(10, 25)
```

```
[15]: (10.0, 25.0)
```



```
[16]: pd_winter.groupby
```

```
[16]: <bound method DataFrame.groupby of          avg(rain)   avg(sun)  avg(t_act)
      avg(t_set)  avg(temp)  avg(wind)  \
      0         0.000000  15.066756   21.380000   21.500000   9.682061  41.490809
      1         0.000000  15.066756   20.510000   15.000000   9.682061  41.490809
      2        -0.321849   8.142367   19.144167   16.252917 -13.413291  65.768851
      3        -0.321849   8.142367   19.312083   17.500000 -13.413291  65.768851
      4         1.355774   6.997916   20.049583   17.000000  27.089749  56.528794
      …            …          …          …          …         …          …
      457052   0.000000   3.992979   17.539167   17.964583  -3.960250  16.326736
      457053   0.000000   3.992979   20.798750   20.497500  -3.960250  16.326736
      457054   0.127165   1.879825   19.134583   16.789167  11.616213  96.030642
      457055   0.163255   2.370457   16.526667   12.748750   4.753975  48.011423
      457056   0.000000   1.764178   17.685000   17.500000   5.646814  33.239672

              date_day  heater_id  oppervlakteverblijfsobject  pandbouwjaar   wijk  \
      0       20210114      88503                       170.0          2002   1003
      1       20210114     173555                       191.0          2005   1003
      2       20170208      53937                        93.0          1957  10601
      3       20170208      56399                        90.0          1962  10601
      4       20190128      99315                       152.0          1968  10602
      …          …          …                            …             …      …
      457052  20170122       8324                       115.0          1978  99522
      457053  20170122      12816                        70.0          1975  99522
```

```
457054  20160126     8735                        214.0          2002  99524
457055  20180119    79825                         76.0          1998  99538
457056  20200120   194488                        593.0          1996  99561

        avg(temp)/10  temp_diff   gas_ch  gas_dhw  year month  total_gas_m2
0           0.968206  20.411794   6.3732   2.5861  2021    01      0.052702
1           0.968206  19.541794   0.0848   1.4487  2021    01      0.008029
2          -1.341329  20.485496   9.0565   0.8779  2017    02      0.106822
3          -1.341329  20.653412  12.5593   0.7018  2017    02      0.147346
4           2.708975  17.340608   8.6957   0.7347  2019    01      0.062042
…                …         …       …       …     …    …            …
457052     -0.396025  17.935192   9.0891   0.7576  2017    01      0.085623
457053     -0.396025  21.194775  18.4499   1.8151  2017    01      0.289500
457054      1.161621  17.972962   4.3184   1.5474  2016    01      0.027410
457055      0.475398  16.051269   5.1254   0.5776  2018    01      0.075039
457056      0.564681  17.120319  13.2843   0.2582  2020    01      0.022837

[457057 rows x 18 columns]>
```

[17]:
```
pd_winter[['avg(t_act)', 'avg(t_set)', 'avg(temp)/10', 'gas_ch']].describe()
```

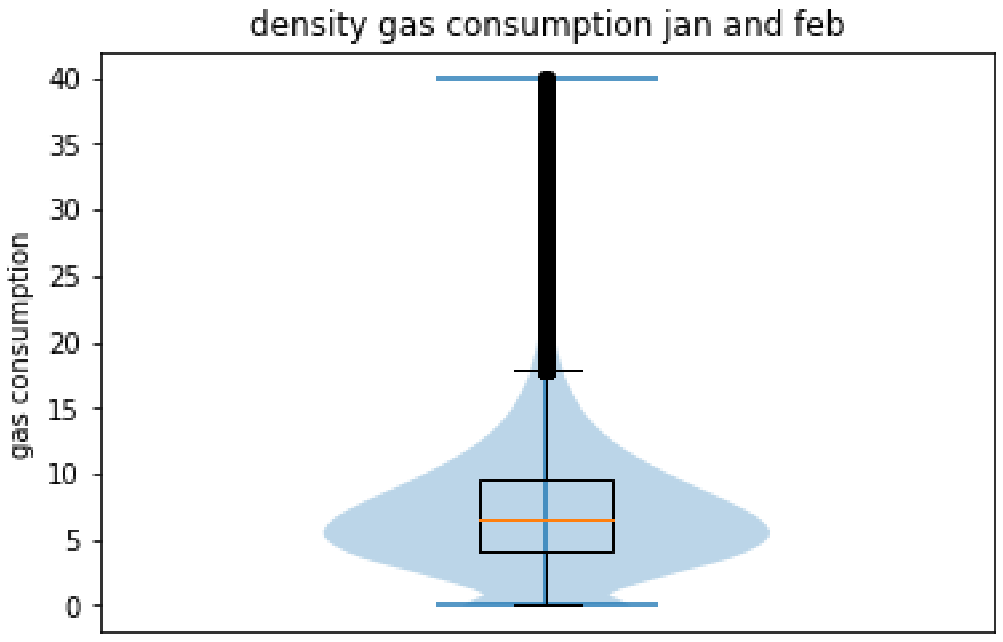
[17]:
```
          avg(t_act)     avg(t_set)   avg(temp)/10         gas_ch
count  457057.000000  457057.000000  456445.000000  457057.000000
mean       19.361823      17.394793       3.890697       7.343040
std         1.774115       2.693917       3.137082       4.709177
min        10.000000       0.000000      -7.939315       0.000000
25%        18.560000      16.000000       1.570685       4.155300
50%        19.551667      17.500000       3.867938       6.589300
75%        20.430000      19.000000       6.200449       9.631200
max        26.000000      26.000000      13.006078      39.989800
```

[18]:
```
pd_winter[['avg(t_act)', 'avg(t_set)', 'avg(temp)/10', 'gas_ch']]
```

[18]:
```
        avg(t_act)  avg(t_set)  avg(temp)/10   gas_ch
0        21.380000   21.500000      0.968206   6.3732
1        20.510000   15.000000      0.968206   0.0848
2        19.144167   16.252917     -1.341329   9.0565
3        19.312083   17.500000     -1.341329  12.5593
4        20.049583   17.000000      2.708975   8.6957
…              …          …             …       …
457052   17.539167   17.964583     -0.396025   9.0891
457053   20.798750   20.497500     -0.396025  18.4499
457054   19.134583   16.789167      1.161621   4.3184
457055   16.526667   12.748750      0.475398   5.1254
457056   17.685000   17.500000      0.564681  13.2843

[457057 rows x 4 columns]
```

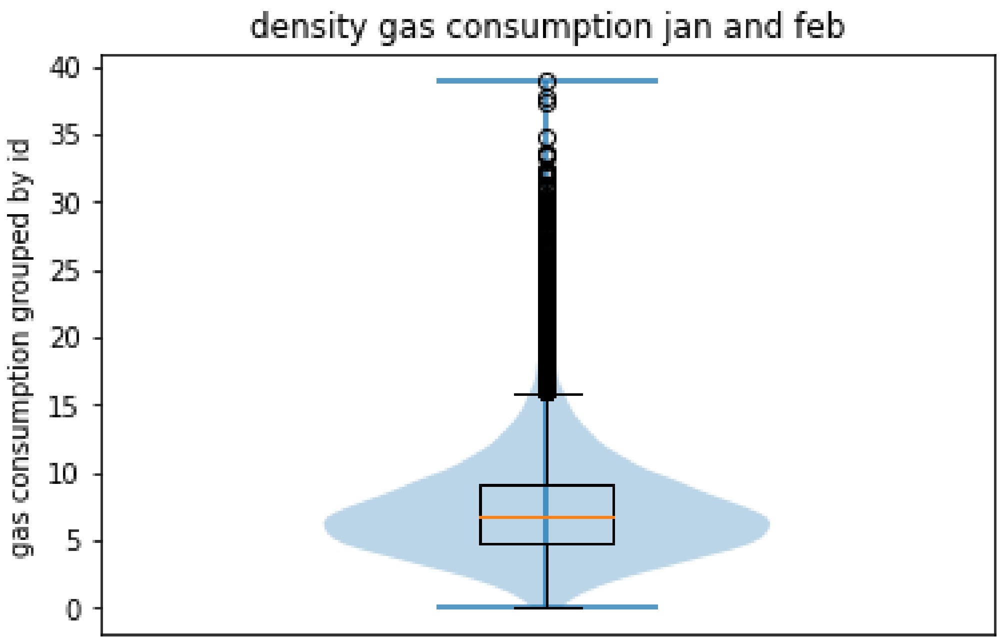
[19]:
```
x = pd_winter['gas_ch']
fig, ax = plt.subplots()
ax.violinplot(x)
ax.boxplot(x)
plt.xticks([])
ax.set_ylabel('gas consumption')
ax.set_title("density gas consumption jan and feb")
```

[19]:
```
Text(0.5, 1.0, 'density gas consumption jan and feb')
```



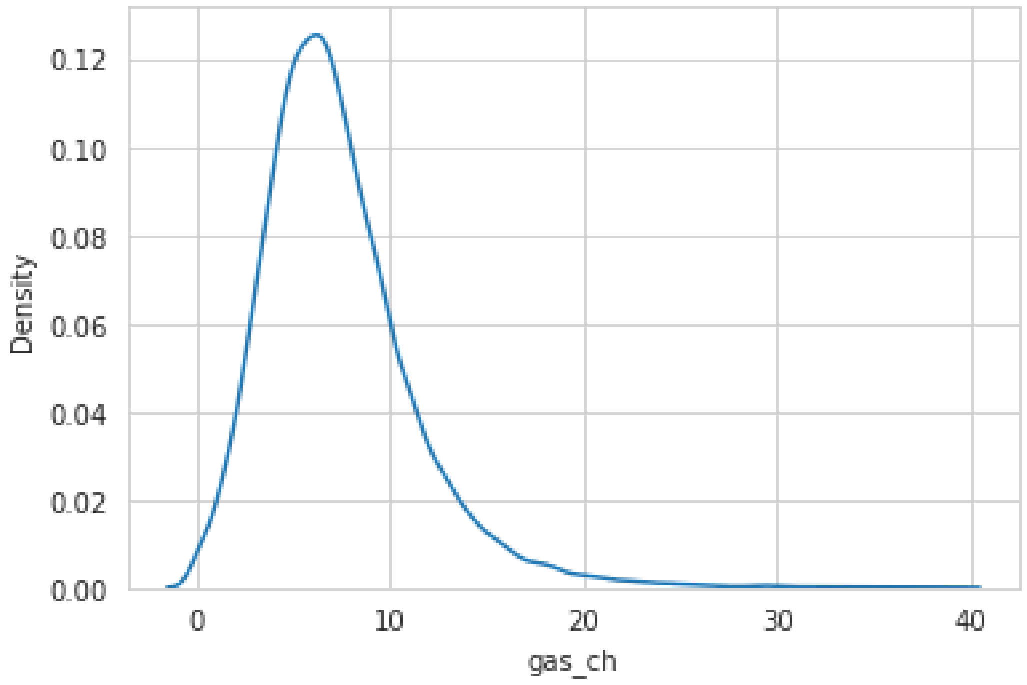
[20]:
```
#de som van grouped id gas_ch geeft een vertekend beeld omdat verschillende ids␣
↪verschillende counts hebben
x = pd_winter.groupby('heater_id')['gas_ch'].mean()


fig, ax = plt.subplots()
ax.violinplot(x)
ax.boxplot(x)
# ax.scatter
plt.xticks([])
ax.set_ylabel('gas consumption grouped by id')
ax.set_title("density gas consumption jan and feb")
```

[20]:
```
Text(0.5, 1.0, 'density gas consumption jan and feb')
```



```
[21]: sns.set_style('whitegrid')
      sns.kdeplot(x)
```

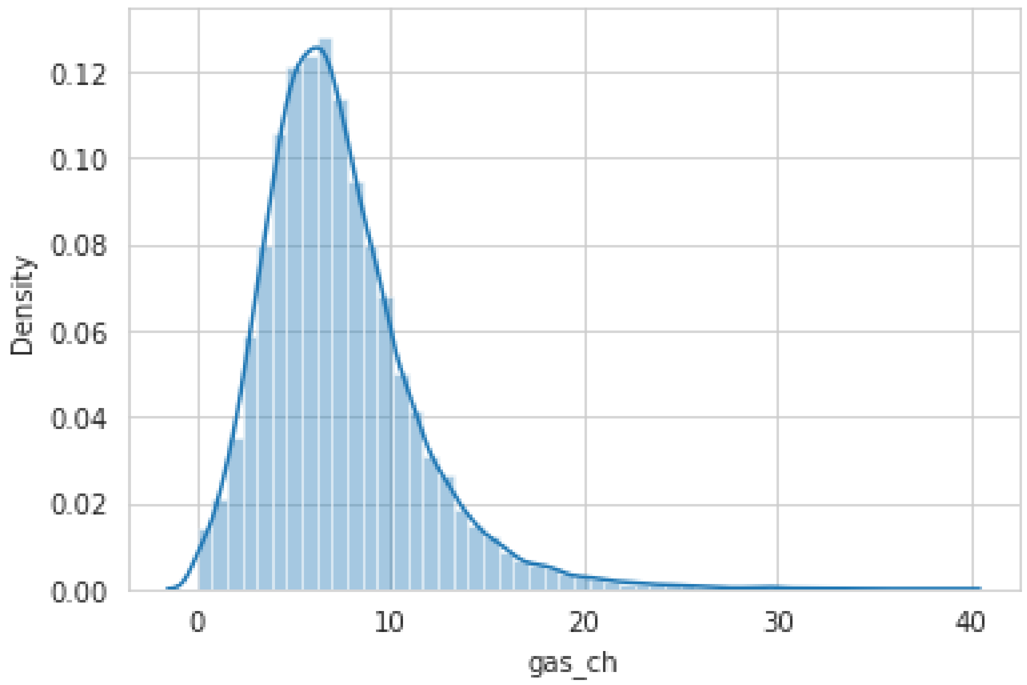
```
[21]: <AxesSubplot:xlabel='gas_ch', ylabel='Density'>
```



[22]:
```
sns.distplot(x, hist=True)
```

```
/opt/conda/lib/python3.8/site-packages/seaborn/distributions.py:2557:
FutureWarning: `distplot` is a deprecated function and will be removed in a
future version. Please adapt your code to use either `displot` (a figure-level
function with similar flexibility) or `histplot` (an axes-level function for
histograms).
  warnings.warn(msg, FutureWarning)
```

[22]:
```
<AxesSubplot:xlabel='gas_ch', ylabel='Density'>
```



[23]:
```
pd_winter.groupby('heater_id').mean().describe()
# mean gas_ch: 7.344812
# range 0-25% = 0 - 4.727518
```

[23]:
```
          avg(rain)      avg(sun)    avg(t_act)    avg(t_set)     avg(temp)  \
count  31423.000000  31160.000000  31468.000000  31468.000000  31445.000000
mean       0.800662     11.935515     19.338394     17.413449     39.837093
std        0.572889      4.784431      1.546166      2.388008     15.513266
min       -0.292772      0.307035     10.144167      0.000000    -66.096203
25%        0.411673      8.846501     18.564211     16.140612     30.836981
50%        0.711325     12.130806     19.447515     17.500000     40.692367
75%        1.066427     15.000140     20.275463     18.815768     50.111305
```

```
max        9.190572     48.766772     25.902250     26.000000     114.916330

          avg(wind)  oppervlakteverblijfsobject  pandbouwjaar          wijk  \
count  31468.000000                31468.000000  31468.000000  31468.000000
mean      51.814216                  147.062413   1964.953763  66058.671825
std       13.186515                  527.493361     62.926572  57696.694006
min        7.909961                   40.000000   1005.000000    300.000000
25%       42.499673                   97.000000   1955.000000  26806.000000
50%       49.729988                  116.000000   1976.000000  43100.000000
75%       59.669280                  141.000000   1989.000000  84502.000000
max      134.214676                68353.000000   2020.000000 197820.000000

       avg(temp)/10     temp_diff        gas_ch       gas_dhw  total_gas_m2
count  31445.000000  31445.000000  31468.000000  31468.000000  31468.000000
mean       3.983709     15.355261      7.344045      0.915435      0.070216
std        1.551327      2.202415      3.918300      0.665643      0.036065
min       -6.609620      0.000000      0.000000      0.000000      0.000000
25%        3.083698     14.017927      4.729456      0.478508      0.047270
50%        4.069237     15.344690      6.734042      0.793855      0.065309
75%        5.011130     16.716805      9.221471      1.206467      0.087053
max       11.491633     28.112954     38.978600     35.816900      0.642479
```

[ ]:

```
[24]: buckets = [0, 4.727518, 13.750750]
bucketnames = ['yes', 'no']


pd_winter['External_Heating'] = pd.cut(x=pd_winter['gas_dhw'], bins=buckets,␣
 ↪labels=bucketnames)
pd_winter.External_Heating.value_counts()
```

```
[24]: yes    435386
no       1164
Name: External_Heating, dtype: int64
```

```
[25]: buckets = [0, 4.727518, 13.750750]
bucketnames = ['yes', 'no']

pd_winter['External_Heating'] = pd.cut(x=pd_winter['gas_dhw'], bins=buckets,␣
 ↪labels=bucketnames)
pd_winter.External_Heating.value_counts()
```

```
[25]: yes    435386
no       1164
Name: External_Heating, dtype: int64
```

[26]:
```
pd_winter.year.value_counts()
```

[26]:
```
2020    144587
2019    100962
2018     72449
2021     72193
2017     46113
2016     20753
Name: year, dtype: int64
```

[ ]:
```
1174/435385
```

[ ]:
```
# session.stop()
# context.stop()
```

RF_gasm2

July 2, 2021

[1]:
```
import pandas as pd
import os
os.environ["SPARK_LOCAL_DIRS"] = "/home/jovyan/work/tmp"

from pyspark import SparkContext
from pyspark.sql import SparkSession, Row, DataFrameWriter, functions as sf
from pyspark.sql.functions import *

#Connect to spark context and create session

context = SparkContext('local[*]')
context.setSystemProperty("spark.driver.memory", "8g")
session = SparkSession(context)
```

[2]:
```
datadir = '/home/jovyan/work/jupyter/data'
df = session.read.json(datadir+'/df_24hour/*.json')
```

[3]:
```
df.printSchema()
df.show()
```

```
root
 |-- avg(rain): double (nullable = true)
 |-- avg(sun): double (nullable = true)
 |-- avg(t_act): double (nullable = true)
 |-- avg(t_set): double (nullable = true)
 |-- avg(temp): double (nullable = true)
 |-- avg(wind): double (nullable = true)
 |-- date: string (nullable = true)
 |-- date_day: string (nullable = true)
 |-- gasmeter_ch_24h: array (nullable = true)
 |    |-- element: double (containsNull = true)
 |-- gasmeter_dhw_24h: array (nullable = true)
 |    |-- element: double (containsNull = true)
 |-- heater_id: long (nullable = true)
 |-- oppervlakteverblijfsobject: string (nullable = true)
 |-- pandbouwjaar: string (nullable = true)
 |-- wijk: string (nullable = true)
```

```
+-------------------+-----------------+-----------------+-----------------+-
-----------------+-----------------+-----------------+--------+-------------
------+-------------------+---------+-------------------------+-----------+--
--+
|          avg(rain)|         avg(sun)|       avg(t_act)|       avg(t_set)|
avg(temp)|        avg(wind)|               date|date_day|    gasmeter_ch_24h|
gasmeter_dhw_24h|heater_id|oppervlakteverblijfsobject|pandbouwjaar|wijk|
+-------------------+-----------------+-----------------+-----------------+-
-----------------+-----------------+-----------------+--------+-------------
------+-------------------+---------+-------------------------+-----------+--
--+
|-0.05294595819723685| 16.43587820098253|19.269583333333333|           19.34|
90.94516994556801|20.256406436265483|2015-10-20 03:32:01|20151020|
[8.621199999999988]|[0.9984000000000037]|    10565|                      490|
1830|1001|
|-0.03940268795338…| 7.113720059929197|17.269999999999996|17.153333333333336|
74.36535687217368| 36.45375808419566|2015-10-28 03:27:38|20151028|
[8.494399999999985]|[0.9584000000000117]|    10565|                      490|
1830|1001|
|               0.0| 5.015029120600641|         18.74875|             17.5|
2.400337425438097| 30.71207452710746|2017-01-24 00:00:02|20170124|
[9.926699999999983]| [1.057800000000043]|    12927|                      490|
1830|1001|
|               0.0| 5.015029120600641|19.330416666666668|             19.0|
2.400337425438097| 30.71207452710746|2017-01-24
00:00:02|20170124|[14.412200000000212]| [1.480899999999906]|    10565|
490|        1830|1001|
| 0.15950629163660526| 25.85703027475181|           21.01|20.002499999999998|
87.00004815705019| 65.48449099504091|2017-05-05
00:00:00|20170505|[10.247299999999996]|[1.8859999999999673]|    10565|
490|        1830|1001|
| 0.15950629163660526| 25.85703027475181|19.143333333333334|             17.5|
87.00004815705019| 65.48449099504091|2017-05-05 00:00:00|20170505|
[4.941600000000108]|[1.3326999999999316]|    12927|                      490|
1830|1001|
| 0.15950629163660526| 25.85703027475181|18.477083333333336|             17.0|
87.00004815705019| 65.48449099504091|2017-05-05
00:00:00|20170505|[3.9876999999999896]|[0.3897000000000004]|    74889|
110|        1900|1001|
|               0.0| 92.27045960665221| 22.03708333333334|
16.5|177.36104142556144| 42.33107086849937|2017-08-14 00:00:00|20170814|
[0.0]|[0.9915999999998348]|    12927|                      490|
1830|1001|
|               0.0| 92.27045960665221|22.044166666666673|
18.5|177.36104142556144| 42.33107086849937|2017-08-14 00:00:00|20170814|
[0.0]|[0.6870000000000118]|    10565|                      490|
1830|1001|
|               0.0| 92.27045960665221|22.216666666666665|
16.0|177.36104142556144| 42.33107086849937|2017-08-14 00:00:00|20170814|
[0.0]|[0.7335000000000065]|    74889|                        110|
1900|1001|
|  3.2666881794837437| 12.37969597763103|21.327142857142853| 8.978095238095236|
136.3339607192684|32.723804110140186|2017-09-30 00:00:00|20170930|
[0.0]|[0.6674999999999898]|    74889|                        110|
1900|1001|
|  3.2666881794837437| 12.37969597763103|22.342916666666664|             22.75|
136.3339607192684|32.723804110140186|2017-09-30
00:00:00|20170930|[3.2569000000003143]|[0.7223999999999933]|    10565|
490|        1830|1001|
|  3.2666881794837437| 12.37969597763103|19.450833333333332|16.749166666666667|
136.3339607192684|32.723804110140186|2017-09-30
00:00:00|20170930|[1.0981000000001586]|[0.6872000000000753]|    12927|
490|        1830|1001|
|-0.04125762574060…| 18.65642163691027|20.848333333333336|
20.49166666666667|54.739008591828146| 43.87506414308755|2018-03-23
00:00:00|20180323|[11.578300000000127]| [1.070799999999963]|    10565|
490|        1830|1001|
|-0.04125762574060…| 18.65642163691027|          17.71125|
17.0|54.739008591828146| 43.87506414308755|2018-03-23 00:00:00|20180323|
[6.214500000000044]|[0.3527000000001408]|    12927|                        490|
1830|1001|
|-0.04125762574060…| 18.65642163691027|17.822916666666668|
16.0|54.739008591828146| 43.87506414308755|2018-03-23 00:00:00|20180323|
[8.033600000000206]|[0.02359999999998…|    74889|                        110|
1900|1001|
|  3.3984857276846356|24.341323885011477|21.069166666666668|
15.5|163.51541030282706| 76.50490330279088|2020-07-04 00:00:00|20200704|
[0.0]|[0.18150000000014…|    12927|                        490|
1830|1001|
|  3.3984857276846356|24.341323885011477|22.149166666666673|
22.0|163.51541030282706| 76.50490330279088|2020-07-04 00:00:00|20200704|
[0.701600000000326]|[0.4733000000001084]|    10565|                        490|
1830|1001|
| 0.06650076269844905|15.264374622653902|16.107916666666668|
16.0|14.224409744023896|28.852744449488398|2016-11-08
00:00:01|20161108|[17.549799999999976]|[0.7513000000000005]|    40989|
123|        1959|1002|
| 0.14786031466780003| 4.544780251412445|21.592499999999998|
19.0|56.046195879670414| 58.73225644094658|2016-12-22 00:00:03|20161222|
[6.527500000000003]|[0.2917000000000005]|    54477|                         79|
1956|1002|
+--------------------+------------------+------------------+------------------+-
------------------+------------------+-------------------+--------+--------------
------+--------------------+---------+-------------------------+------------+--
--+
only showing top 20 rows
```

```
[4]: from pyspark.sql.functions import *

     # add col temp diff
     df = df.withColumn('avg(temp)/10', df['avg(temp)']/10)

     #filter: t_act < 25
     df = df.filter(df['avg(t_act)'] <= 26)
     df = df.filter(df['avg(t_act)'] >= 10)
     df = df.withColumn('temp_diff', df['avg(t_act)'] - df['avg(temp)/10'])

     #filter: t_set < 25
     df = df.filter(df['avg(t_set)'] <= 26)

     #filter temp = max(temp_diff, 0)
     df = df.withColumn('temp_diff', when(col('temp_diff') < 0, 0).
      ↪otherwise(col('temp_diff')))

     # add gas outcomes as type double and filter < 40
         # gas_ch
     df = df.withColumn('gas_ch', df['gasmeter_ch_24h'].getItem(0))
     df = df.filter(df.gas_ch < 40)

         # gas_dhw
     df = df.withColumn('gas_dhw', df['gasmeter_dhw_24h'].getItem(0))
     df = df.filter(df.gas_dhw < 40)

     # change col types
     df = df.withColumn("oppervlakteverblijfsobject",␣
      ↪df['oppervlakteverblijfsobject'].cast("double"))
     df = df.withColumn("pandbouwjaar", df['pandbouwjaar'].cast("integer"))
     df = df.withColumn("wijk", df['wijk'].cast('integer'))

     # get col year
     df = df.withColumn('year', col('date_day').substr(1, 4))

     # get col gas use m2
     df = df.withColumn('total_gas_m2', (df.gas_ch + df.gas_dhw)/df.
      ↪oppervlakteverblijfsobject)

     # filter: oppervalkteverblijfsobjct >= 40
     df = df.filter(df.oppervlakteverblijfsobject >= 40)

     # drop unnecessary cols
     cols = ['date', 'date_day', 'gasmeter_ch_24h', 'gasmeter_dhw_24h']
     df = df.drop(*cols)
```

```
df.printSchema()
```

```
root
 |-- avg(rain): double (nullable = true)
 |-- avg(sun): double (nullable = true)
 |-- avg(t_act): double (nullable = true)
 |-- avg(t_set): double (nullable = true)
 |-- avg(temp): double (nullable = true)
 |-- avg(wind): double (nullable = true)
 |-- heater_id: long (nullable = true)
 |-- oppervlakteverblijfsobject: double (nullable = true)
 |-- pandbouwjaar: integer (nullable = true)
 |-- wijk: integer (nullable = true)
 |-- avg(temp)/10: double (nullable = true)
 |-- temp_diff: double (nullable = true)
 |-- gas_ch: double (nullable = true)
 |-- gas_dhw: double (nullable = true)
 |-- year: string (nullable = true)
 |-- total_gas_m2: double (nullable = true)
```

[5]:
```
df.show(5)
```

```
+--------------------+-----------------+------------------+------------------+--
---------------+------------------+---------+--------------------------+--------
----+----+------------------+------------------+------------------+-------------
-----+----+--------------------+
|           avg(rain)|         avg(sun)|        avg(t_act)|        avg(t_set)|
avg(temp)|
avg(wind)|heater_id|oppervlakteverblijfsobject|pandbouwjaar|wijk|
avg(temp)/10|         temp_diff|            gas_ch|           gas_dhw|year|
total_gas_m2|
+--------------------+-----------------+------------------+------------------+--
---------------+------------------+---------+--------------------------+--------
----+----+------------------+------------------+------------------+-------------
-----+----+--------------------+
|-0.05294595819723685|16.43587820098253|19.269583333333333|
19.34|90.94516994556801|20.256406436265483|    10565|                     490.0|
1830|1001| 9.094516994556802|10.175066338776531|
8.621199999999988|0.9984000000000037|2015| 0.01963183673469386|
|-0.03940268795338…|7.113720059929197|17.269999999999996|17.153333333333336|74
.36535687217368| 36.45375808419566|    10565|                     490.0|
1830|1001| 7.436535687217368| 9.833464312782628|
8.494399999999985|0.9584000000000117|2015|0.019291428571428564|
|                 0.0|5.015029120600641|          18.74875|
17.5|2.400337425438097| 30.71207452710746|    12927|                     490.0|
1830|1001|0.2400337425438097|18.508716257456193| 9.926699999999983|
1.057800000000043|2017|0.022417346938775562|
|                 0.0|5.015029120600641|19.330416666666668|
19.0|2.400337425438097| 30.71207452710746|    10565|                     490.0|
1830|1001|0.2400337425438097| 19.09038292412286|14.412200000000212|
1.4808999999999906|2017| 0.032434897959183392|
| 0.15950629163660526|25.85703027475181|
21.01|20.002499999999998|87.00004815705019| 65.48449099504091|    10565|
490.0|        1830|1001|
8.700004815705018|12.309995184294984|10.247299999999996|1.8859999999999673|2017|
0.0247618367346938|
+--------------------+-----------------+------------------+------------------+--
----------------+------------------+---------+--------------------------+--------
----+----+------------------+------------------+------------------+-------------
-----+----+--------------------+
only showing top 5 rows
```

[6]:
```python
#get df grouped by heater id and year

#check

from pyspark.sql.functions import avg, count
df_rf = df.groupBy("heater_id", 'year').agg(

    count('heater_id').alias('records'),
    avg('temp_diff'),
    avg('pandbouwjaar'),
    avg('year'),
    avg('oppervlakteverblijfsobject'),
    avg('avg(temp)'),
    avg('avg(wind)'),
    avg('avg(rain)'),
    avg('avg(sun)'),
    avg('gas_ch'),
    avg('gas_dhw'),
    avg('total_gas_m2')
)

df_rf.orderBy('heater_id').show(5)
```

```
+---------+----+-------+------------------+-----------------+---------+---------
----------------------+-------------------+-------------------+-------------------+
-------------------+-------------------+-------------------+--------------------+
|heater_id|year|records|
avg(temp_diff)|avg(pandbouwjaar)|avg(year)|avg(oppervlakteverblijfsobject)|
avg(avg(temp))|    avg(avg(wind))|    avg(avg(rain))|     avg(avg(sun))|
avg(gas_ch)|      avg(gas_dhw)|   avg(total_gas_m2)|
+---------+----+-------+------------------+-----------------+---------+---------
----------------------+-------------------+-------------------+-------------------+
------------------+------------------+------------------+-------------------+
|      1499|2016|    306|12.453300370235102|           1967.0|   2016.0|
108.0|
91.51045250257755|44.400704194223465|0.6661683790950261|39.473702595110474|
5.202154575163405| 1.316375816993463| 0.06035676289034138|
|      1499|2019|    326|11.082129498629476|           1967.0|   2019.0|
108.0|100.94631365173376| 47.49253637194074|0.8330010602508482|
41.55707809297405|  4.53654355828222|0.7167487730061337| 0.04864159566007736|
|      1499|2017|    332|11.840301201578082|           1967.0|   2017.0|
108.0| 99.75939344809571| 45.75986844153401| 0.904056034215566|
40.42001451813369|4.7041987951807265|1.3421039156626535| 0.05598428435966094|
|      1499|2015|     88|11.718996655513003|           1967.0|   2015.0|
108.0|
93.66933345192811|54.327702589339765|0.8836471741474242|13.234175291583078|
5.986957954545457|1.4604636363636365| 0.06895760732323235|
|      1499|2018|    329|12.179415351386263|           1967.0|   2018.0|
108.0| 98.31711915703762| 46.32753568881532|0.5933056634898966|
44.00934019763482| 4.363585410334349|1.3278389057750757|0.052698373297309524|
+----------+----+-------+------------------+-----------------+---------+---------
----------------------+------------------+------------------+------------------+
------------------+------------------+------------------+-------------------+
only showing top 5 rows
```

[7]:
```
df_rf.describe().show()
```

```
+-------+------------------+------------------+------------------+--------------
----+-----------------+------------------+--------------------------------+------
------------+------------------+--------------------+------------------+---------
---------+------------------+--------------------+
|summary|         heater_id|              year|           records|
avg(temp_diff)|avg(pandbouwjaar)|
avg(year)|avg(oppervlakteverblijfsobject)|    avg(avg(temp))|    avg(avg(wind))|
avg(avg(rain))|     avg(avg(sun))|       avg(gas_ch)|      avg(gas_dhw)|
avg(total_gas_m2)|
+-------+------------------+------------------+------------------+--------------
----+-----------------+------------------+--------------------------------+------
------------+------------------+--------------------+------------------+---------
---------+------------------+--------------------+
|  count|            124787|            124787|            124787|
124646|           124787|            124787|                          124787|
124646|            124787|             124425|            123586|
124787|            124787|              124787|
|   mean|  99087.9017525864|2018.8882175226586|209.67591175362818|12.60308187622
1438|1965.039146705987|2018.8882175226586|              146.2851498954218|
76.01430218900265| 44.15684397017695| 0.7186161552205141|28.465776488845908|
4.432465428288918|0.8033218272349866|0.044402623350609974|
| stddev|64368.351889833204|1.6370002246718944| 137.1270314794069|
3.555771040507057|63.49026916458678|1.6370002246718944|
478.80513075031365|36.973608106878665|10.292178321381122|0.30009477814046975|
17.10906655587824|3.4483344902300863| 0.583696928374209| 0.03130552131586808|
|    min|             1499|              2015|                 1|
0.0|           1005.0|           2015.0|
40.0|-47.61569050983935| 5.433052112498122|-0.5065557189034849|
0.29311813968648|              0.0|              0.0|                0.0|
|    max|           204776|              2021|
384|23.941872700640655|           2020.0|            2021.0|
68353.0| 264.8715407975971|130.01002399352583| 11.969095836325218|
127.933974674158|38.733149999999114| 36.39265000000012|  0.6505844444444395|
+-------+------------------+------------------+------------------+--------------
----+-----------------+------------------+--------------------------------+------
------------+------------------+--------------------+------------------+---------
---------+------------------+--------------------+
```

0.1 Random Forest

[8]:
```python
from pyspark.ml import Pipeline
from pyspark.ml.regression import RandomForestRegressor
from pyspark.ml.feature import VectorIndexer
from pyspark.ml.evaluation import RegressionEvaluator
```

[9]:
```python
df_rf = df_rf.drop('heater_id', 'year', 'records', 'avg(gas_ch)',␣
 ↪'avg(gas_dhw)')
df_rf.printSchema()
df_rf.show(3)
```

```
root
 |-- avg(temp_diff): double (nullable = true)
 |-- avg(pandbouwjaar): double (nullable = true)
 |-- avg(year): double (nullable = true)
 |-- avg(oppervlakteverblijfsobject): double (nullable = true)
 |-- avg(avg(temp)): double (nullable = true)
 |-- avg(avg(wind)): double (nullable = true)
 |-- avg(avg(rain)): double (nullable = true)
 |-- avg(avg(sun)): double (nullable = true)
 |-- avg(total_gas_m2): double (nullable = true)

+------------------+-----------------+---------+-------------------------------+
------------------+------------------+------------------+-----------------+-----
----------------+
|    avg(temp_diff)|avg(pandbouwjaar)|avg(year)|avg(oppervlakteverblijfsobject)|
avg(avg(temp))|    avg(avg(wind))|    avg(avg(rain))|    avg(avg(sun))|
avg(total_gas_m2)|
+------------------+-----------------+---------+-------------------------------+
------------------+------------------+------------------+-----------------+-----
```

```
----------------+
| 9.164158747196396|           1973.0|   2016.0|
126.0|102.13704489796953|
48.02385066684009|0.7069514931311731|39.15119119918311|0.012848126786946665|
| 9.048692950297404|           1987.0|   2020.0|                          154.0|
111.07649411354359|45.151972067300235|0.7851257747605533|46.43218297051898|0.029
021788270038987|
|10.058170935157975|           2002.0|   2019.0|
191.0|114.71220048427774|
43.44013095070814|0.8057876669096891|48.18929981385622| 0.02468419688665534|
+------------------+-----------------+---------+------------------------------+
------------------+------------------+------------------+-----------------+-----
----------------+
only showing top 3 rows
```

[10]:
```python
from pyspark.sql import Row
from pyspark.ml.linalg import Vectors

def transData(data):
    return data.rdd.map(lambda r: [Vectors.dense(r[:-1]),r[-1]]).
 ↪toDF(['features','label'])
```

[11]:
```python
transformed= transData(df_rf)
transformed.show(5)
```

```
+--------------------+--------------------+
|            features|               label|
+--------------------+--------------------+
|[9.16415874719639…|0.012848126786946665|
|[9.04869295029740…|0.029021788270038987|
|[10.0581709351579…| 0.02468419688665534|
|[9.84885378239807…| 0.03306417096959266|
|[8.39143171627965…|0.028507745038884404|
+--------------------+--------------------+
only showing top 5 rows
```

[12]:
```python
from pyspark.ml import Pipeline
from pyspark.ml.regression import LinearRegression
from pyspark.ml.feature import VectorIndexer
from pyspark.ml.evaluation import RegressionEvaluator

featureIndexer = VectorIndexer(inputCol="features", \
                               outputCol="indexedFeatures",\
                               maxCategories=4).fit(transformed)
```

```
df_rf = featureIndexer.transform(transformed)
df_rf.show(5,True)
```

```
+--------------------+--------------------+--------------------+
|            features|               label|     indexedFeatures|
+--------------------+--------------------+--------------------+
|[9.16415874719639…|0.012848126786946665|[9.16415874719639…|
|[9.04869295029740…|0.029021788270038987|[9.04869295029740…|
|[10.0581709351579…| 0.02468419688665534|[10.0581709351579…|
|[9.84885378239807…| 0.03306417096959266|[9.84885378239807…|
|[8.39143171627965…|0.028507745038884404|[8.39143171627965…|
+--------------------+--------------------+--------------------+
only showing top 5 rows
```

[13]:
```
#train test split
(trainingData, testData) = df_rf.randomSplit([0.7, 0.3], seed=101)
print(f" length trainset {trainingData.count()}")
print(f" length testset {testData.count()}")
```

```
 length trainset 87399
 length testset 37388
```

[14]:
```
## train model

# rf instance
rf = RandomForestRegressor(featuresCol='features')

#pipeline
pipeline = Pipeline(stages=[featureIndexer, rf])

# chain indexer and tree in pipeline
model = pipeline.fit(trainingData)
```

[15]:
```
## make predictions

#predictions
predictions = model.transform(testData)

# results
predictions.select('features', 'label','prediction').show(5)
```

```
+--------------------+--------------------+--------------------+
|            features|               label|          prediction|
+--------------------+--------------------+--------------------+
|[4.53481425735851…|0.003240957015409581|0.018729086650799205|
|[4.70764708103691…| 0.01182340425531915| 0.04232517336995147|
|[6.71956619561564…|0.012668122081025301| 0.02278396983679409|
|[6.92515453303155…|0.009821295814244067|0.027390975126491014|
|[7.04033736046601…|              1.15E-4|0.030364626191401067|
+--------------------+--------------------+--------------------+
only showing top 5 rows
```

[16]:
```
#Select (prediction, true label) and compute test error
evaluator = RegressionEvaluator(
    labelCol="label", predictionCol="prediction", metricName="rmse")

#calculate rmse
rmse = evaluator.evaluate(predictions)
print("Root Mean Squared Error (RMSE) on test data = %g" % rmse)
```

```
Root Mean Squared Error (RMSE) on test data = 0.023003
```

[17]:
```
df_rf.describe().show()
```

```
+-------+--------------------+
|summary|               label|
+-------+--------------------+
|  count|              124787|
|   mean|0.044402623350609974|
| stddev| 0.03130552131586808|
|    min|                 0.0|
|    max|  0.6505844444444395|
+-------+--------------------+
```

[18]:
```
import sklearn.metrics

y_true = predictions.select('label').toPandas()
y_pred = predictions.select('prediction').toPandas()

r2_score = sklearn.metrics.r2_score(y_true, y_pred)
print('r2_score: {:4.3f}'.format(r2_score))
```

```
r2_score: 0.460
```

[19]:
```
model.stages[-1].featureImportances
```

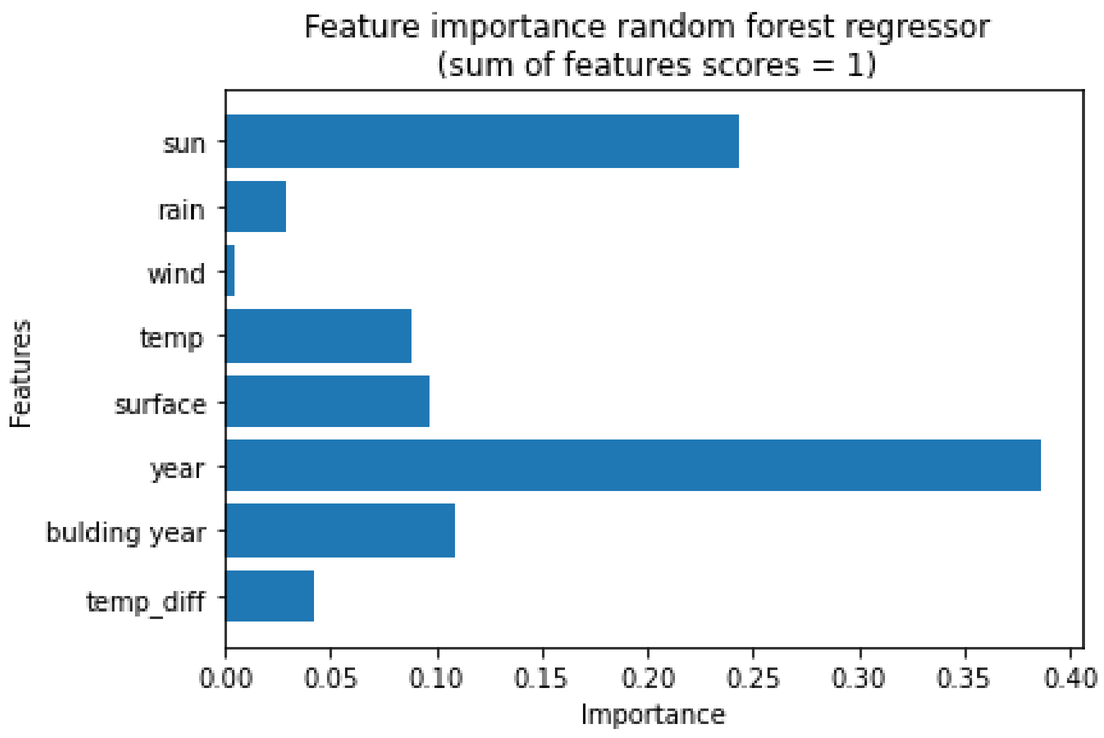
[19]:
```
SparseVector(8, {0: 0.0421, 1: 0.1094, 2: 0.3864, 3: 0.0968, 4: 0.0888, 5:
0.0046, 6: 0.0286, 7: 0.2434})
```

[20]:
```
session.stop()
context.stop()
```

[88]:
```python
labels = ['','temp_diff', 'bulding year', 'year', 'surface', 'temp', 'wind',␣
 ↪'rain', 'sun']

import matplotlib.pyplot as plt

predictors = {0: 0.0421, 1: 0.1094, 2: 0.3864, 3: 0.0968, 4: 0.0888, 5: 0.0046,␣
 ↪6: 0.0286, 7: 0.2434}
fit, ax = plt.subplots()
ax.barh(range(len(predictors)), predictors.values())
ax.set_yticklabels(labels)
ax.set_title("Feature importance random forest regressor \n (sum of features␣
 ↪scores = 1)")
ax.set_ylabel("Features")
ax.set_xlabel("Importance")
# ax.set_xticklabels(['Tuesday', 'Saturday'], rotation=0)
```

```
<ipython-input-88-8617cf2ad3ce>:8: UserWarning: FixedFormatter should only be
used together with FixedLocator
  ax.set_yticklabels(labels)
```

[88]:
```
Text(0.5, 0, 'Importance')
```



[ ]: