HEATING SYSTEM START-UP DEPENDING ON THE NUMBER OF PEOPLE IN THE ROOM AND THE CONNECTED ELECTRONIC DEVICES

Smart House

HERNANDEZ OLIVAN, JAVIER
VICENTE ARAGUES, ALFONSO

ENSE 3 – SEM – SMART SYSTEMS
2022/2023

Table of contents

1. INTRODUCTION

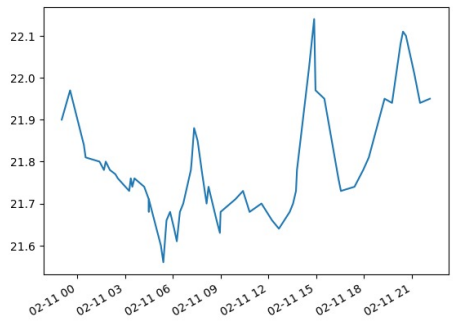
The thermal energy constantly released by the human body corresponds on average to that of a 100-watt light bulb. "Every day, an adult release an average of three kilowatt hours of energy, an amount that could run an LCD TV for 30 hours." Much of this energy is lost in the environment and it is precisely this "waste" that we want to regulate by starting up the heating.

Due to technologies, family life in the living room has changed sharply. Television, home theaters, laptops, autonomous vacuum cleaners, etc. All this set of electronic devices coexist in harmony with the space shared by relatives in the living room on a day-to-day basis. Now, all these devices, as mentioned above, produce heat. If there are too many people in the room and there are too many devices connected, the heat produced can become too high. At this point, sometimes the heating does not need to work at its highest performance, therefore, our goal is to study how the heating should change depending on the members present in the room, as well as the connected electronic devices.

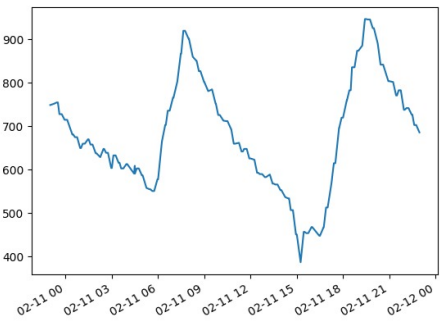
2. DATA COLLECTION

As we are focused on the study of heating, we collect data for a week during winter season. The two data obtained are: electricity consumption (for electronic devices) and CO2 sensor. However, to be more specific with temperature regulation, we can think of more parameters:

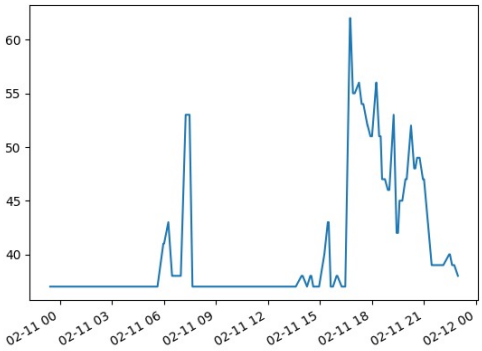
1. Window sensors: if the windows are opened the temperature will drop.
2. Temperature/humidisty sensor: essential to know the temperature regardless of the rest of the parameters.
3. Sound level sensor: generally, if there is a lot of noise it is due to the television to family conversations.
4. Heater production sensors: to know how much energy is being produced.



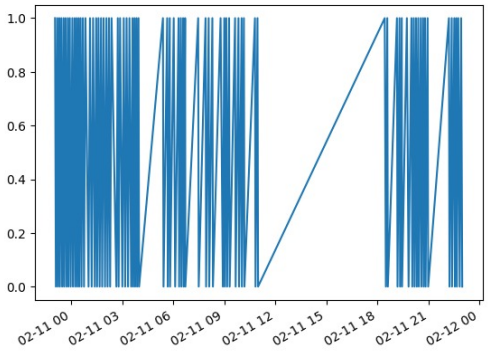
Temperature



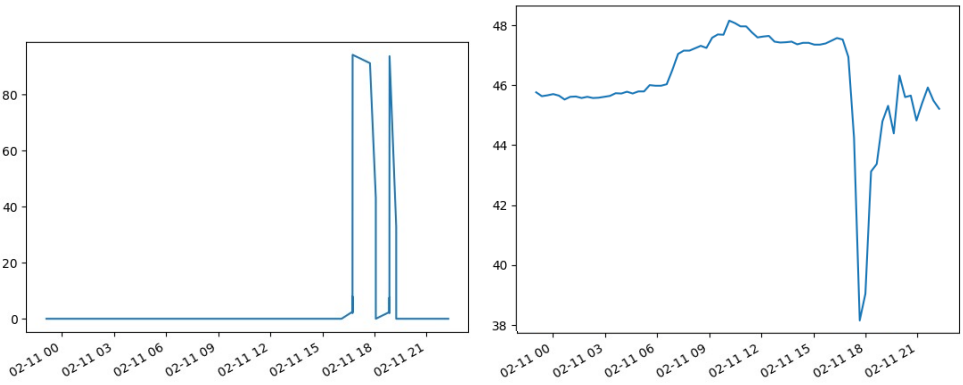
CO_2



Sound Level



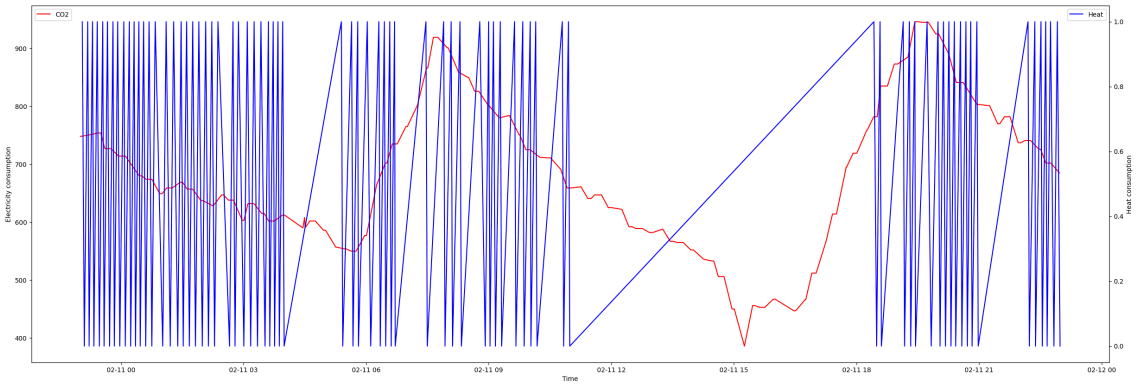
Heating



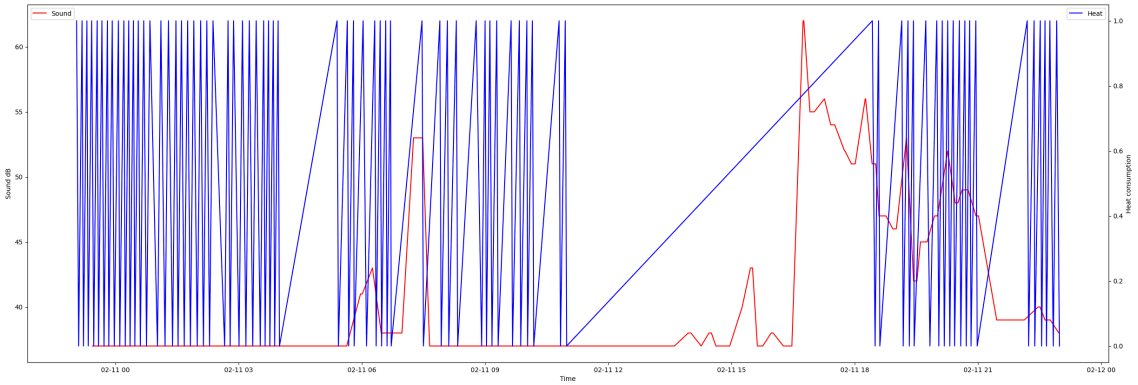
TV consumption (electricity) Humidity

We have a Boolean variable (heating), and the rest are float. We will be interested in estimating, therefore, the float variables. We are going to plot each float with the heat in order to see if it exists a correlation between them.

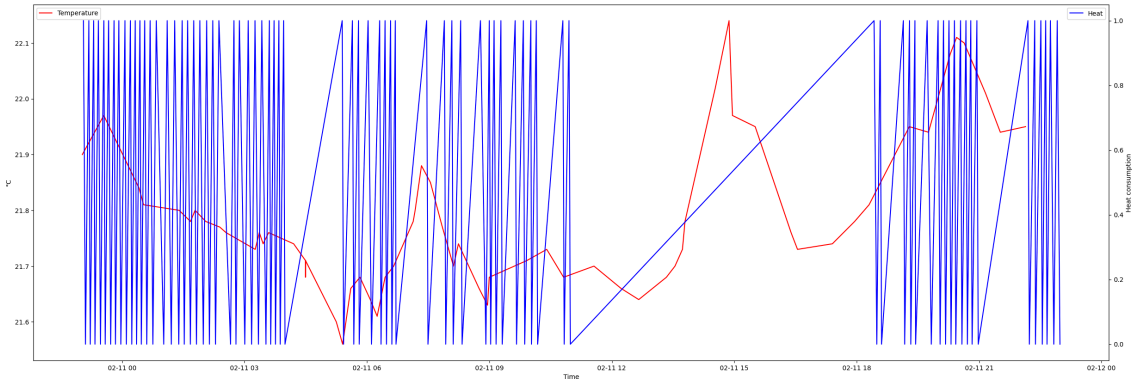
CO2 (red) vs Heat (blue)



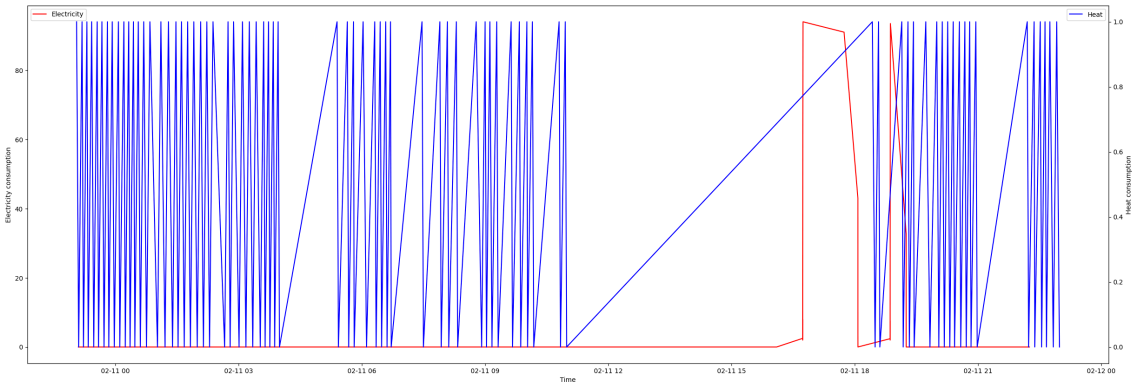
Sound (red) vs Heat (blue)



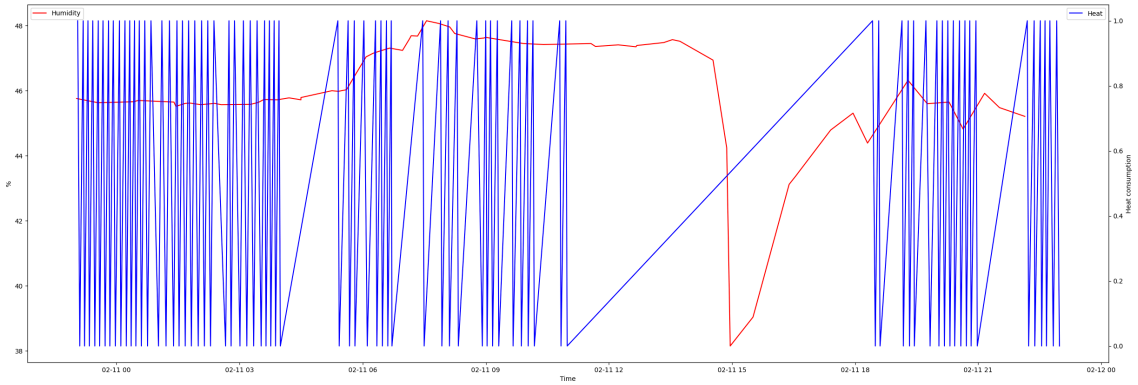
Temperature (red) vs Heat (blue)



Electricity (red) vs Heat (blue)



Humidity (red) vs Heat (blue)



None of the floats maintains a correlation with heat, therefore, we will limit ourselves to estimating the values of the float variables.

3. MODEL IMPLEMENTATION

Within the floats there are two discrete variables (CO2 and sound level) and the other three are continuous (temperature, electricity and humidity), therefore, the same algorithm cannot be used to estimate all the variables.

For continuous variables, a Decision Tree (DT) will be used or, if the results are not as expected, a Random Forest (RF) could be used.

The main advantage of the decision tree classifier is its ability to using different feature subsets and decision rules at different stages of classification. A general DT consists of one root node, a number of internal and leaf nodes, and branches.

Decision Tree Classifier is a simpler model which consists of one root node, several internal and leaf nodes, and branches. It can use different feature subsets and decision rules at different stages of classification. Random Forest Classifier uses a combination of a specific numbers of DT. Considering the RF by default according to sklearn, the number of the trees we are using is 100, as such, DT would have less variance sample to sample but ultimately will not be a strong predictive model comparing it to RF.

There are some advantages to using DT instead of RF. DT is easier to interpret and faster than RF because it can perform well on large datasets, the internal workings are capable of being observed and thus make it possible to reproduce work and can handle both numerical and categorical data. On the contrary, it has some disadvantages comparing to RF. Building DTs require algorithms capable of determining an optimal choice at each node and they are prone to overfitting, especially when a tree is particularly deep.

For continuous variables, the Support Vector Machine (SVM) will be used.

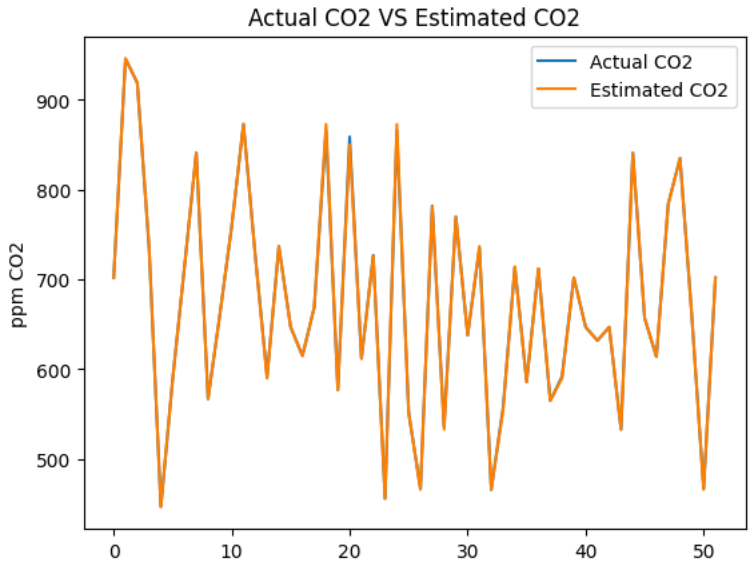
SVM is a supervised machine learning model that uses classification algorithms for two-group classification problems. It works by correlating data to a large feature space so that data points can be categorized, even if the data cannot be linearly separated otherwise. Thus, this model needs more data than RF to make a reliable estimation.

4. RESULTS

'Actual' represents the real variable of the model obtained directly from the data sheet.
'Estimated' represents the predicted variable of the model obtained using sklearn libraries.

1. PREDICT CO2

CO2 is a discrete variable, so we can use a Decision Tree or a Random Forest to estimate it.



The accuracy of the model is 0.75
The main absolute error of the model is 0.7884615384615384
The f-score of the model is 0.7147435897435896
The recall score of the model is 0.75

Looking at the graph above and the results of accuracy and f-score it is strange that they get such different results, because from the graph you could say that the accuracy is practically 1. Therefore, we wanted to show the real values and estimates:

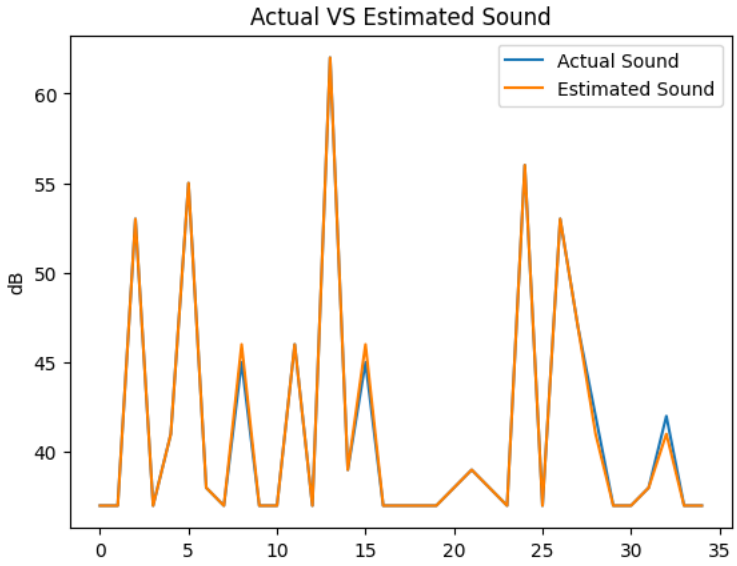
[(702.0, 702.0), (946.0, 946.0), (919.0, 919.0), (735.0, 737.0), (447.0, 447.0), (589.0, 589.0), (714.0, 714.0), (841.0, 841.0), (567.0, 567.0), (661.0, 661.0), (759.0, 759.0), (873.0, 873.0), (727.0, 726.0), (592.0, 590.0), (737.0, 737.0), (647.0, 647.0), (615.0, 615.0), (669.0, 669.0), (866.0, 873.0), (577.0, 577.0), (859.0, 850.0), (612.0, 612.0), (727.0, 726.0), (456.0, 456.0), (866.0, 873.0), (552.0, 552.0), (467.0, 466.0), (782.0, 782.0), (536.0, 533.0), (770.0, 770.0), (638.0, 638.0), (735.0, 737.0), (466.0, 466.0), (557.0, 554.0), (714.0, 714.0), (586.0, 586.0), (712.0, 712.0), (565.0, 565.0), (592.0, 590.0), (702.0, 702.0), (647.0, 647.0), (632.0, 632.0), (647.0, 647.0), (533.0, 533.0), (841.0, 841.0), (657.0, 657.0), (614.0, 614.0), (784.0, 784.0), (835.0, 835.0), (659.0, 659.0), (467.0, 466.0), (702.0, 702.0)]

As can be seen now, of the 52 data that have been estimated, 39 have been correct, which corresponds to 75% accuracy. The difference in values between the real and the estimated is very small, which is why they are not seen in the graph.

The results are the same whether Decision Tree or Random Forest is used.

2. PREDICT SOUND

Sound is a discrete variable, so we can use a Decision Tree to estimate it.

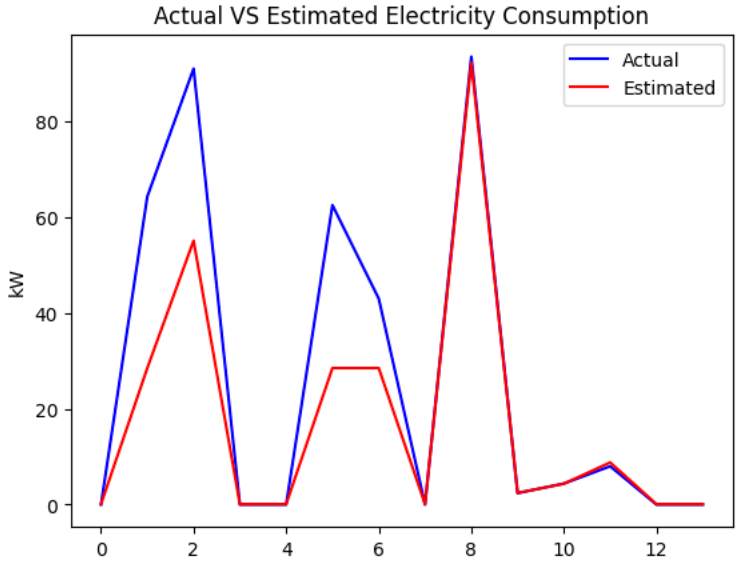


The accuracy of the model is 0.8857142857142857
The main absolute error of the model is 0.11428571428571428
The f-score of the model is 0.8571428571428571
The recall score of the model is 0.8857142857142857

Here the accuracy results agree more with what is shown in the graph.

3. PREDICT ELECTRICITY CONSUMPTION

Electricity consumption is a continuous variable. It is not possible to use DT in this case. We are going to use support vector machine method (SVM) to predict electricity consumption.



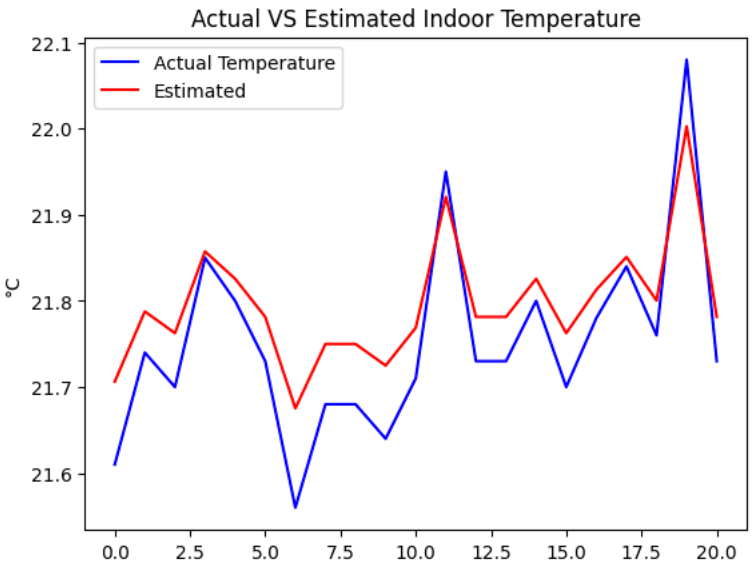
[(0.0, 0.09985213783843605),
(64.3, 28.50895338230529),
(91.0, 55.094965313830784),
(0.0, 0.09985213783843605),
(0.0, 0.09985213783843605),
(62.5, 28.50895338230529),
(43.0, 28.50905440688939),
(0.0, 0.09985213783843605),
(93.5, 92.28548056995628),
(2.4, 2.4463831229914668),
(4.4, 4.362815342428991),
(8.0, 8.80369312351699),
(0.0, 0.09985213783843605),
(0.0, 0.09985213783843605)]

The main absolute error of the model is 8.777069048273075

4. PREDICT INDOOR TEMPERATURE

Temperature is a continuous variable. It is not possible to use DT in this case. We are going to use support vector machine method (SVM) to predict the indoor temperature.

[(21.61, 21.706210079209264),
(21.74, 21.787690805946806),
(21.7, 21.762493357763784),
(21.85, 21.857316237660804),
(21.8, 21.82562670720012),
(21.73, 21.781383142949004),
(21.56, 21.675265098696887),
(21.68, 21.7499315774515),
(21.68, 21.7499315774515),
(21.64, 21.724896806343917),
(21.71, 21.76878404441645),
(21.95, 21.920704765312447),
(21.73, 21.781383142949004),
(21.73, 21.781383142949004),
(21.8, 21.82562670720012),
(21.7, 21.762493357763784),
(21.78, 21.81296678836489),
(21.84, 21.850975303126596),
(21.76, 21.80032048320591),
(22.08, 22.00260159611127),
(21.73, 21.781383142949004)]

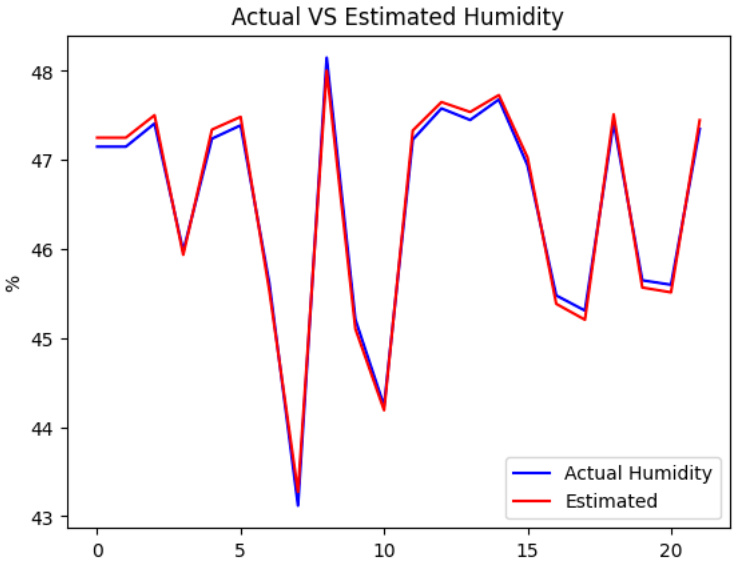


The main absolute error of the model is 0.05346453057974357. It is so small because the differences between the actual and estimated temperature are hundredths of a degree.

5. PREDICT HUMIDITY

Humidity is a continuous variable. It is not possible to use DT in this case. We are going to use support vector machine method (SVM) to predict the humidity.

[(47.15, 47.25127906841871),
(47.15, 47.25127906841871),
(47.41, 47.50365746182406),
(45.98, 45.93603248752759),
(47.24, 47.34188928434136),
(47.39, 47.48532911824882),
(45.62, 45.53617302863984),
(43.12, 43.273855328574754),
(48.15, 48.00482268980761),
(45.21, 45.10470390234627),
(44.24, 44.19110009690814),
(47.23, 47.33197810780237),
(47.58, 47.65087057063801),
(47.45, 47.539705231858015),



(47.68, 47.72959077143271),
(46.94, 47.02917295198536),
(45.48, 45.385744970849125),
(45.31, 45.20737218808709),
(47.42, 47.512746344197964),
(45.65, 45.56880615817021),
(45.6, 45.514494238860905),
(47.35, 47.44808555522553)]

The main absolute error of the model is 0.09228132280770945. It is so small because the differences between the actual and estimated temperature are hundredths of a degree.

5. CONCLUSION

At first, it has not been possible to establish a correlation between turning the boiler on and off (heating) and the variables measured such as sound volume, electricity consumption, humidity and interior temperature, therefore, it is not possible to estimate with these variables whether or not the heat can be turned on. It is true that between these variables there is some correlation, such as the electricity consumption from a television and the volume of sound since the TV emits sound.

On the other hand, predicting continuous variables is more difficult than predicting discrete variables, as can be seen in the previous section. Although the absolute error is less when predicting the continuous ones, if we had to calculate the accuracy (which is possible for continuous variables but is not very adequate) we would see that it would be practically 0 because no exact value matches.

6. ANNEXES CODE PYTHON

1. Read the data from the csv files

```
import pandas as pd # import pandas library

CO2Data = pd.read_csv("CO2/datadoubleremoved.csv", sep=",") # read csv
file
timeCO2 = CO2Data["time"] # get time column
CO2 = CO2Data["value"] # get CO2 column
CO2_max = CO2.max() # get max value of label column
CO2_min = CO2.min() # get min value of label column
print(f"The levels of CO2 we have in our data are from {CO2_min} to
{CO2_max}")

HeatData = pd.read_csv("Heat/datadoubleremoved.csv", sep=",") # read csv
file
timeHeat = HeatData["time"] # get time column
Heat = HeatData["value"] # get heat column
Heat_max = Heat.max() # get max value of label column
Heat_min = Heat.min() # get min value of label column
print(f"The levels of heat consumption we have in our data are from
{Heat_min} to {Heat_max}")

ElecData = pd.read_csv("Electricity/datadoubleremoved.csv", sep=",") #
read csv file
timeElec = ElecData["time"] # get time column
Elec = ElecData["value"] # get electricity column
Elec_max = Elec.max() # get max value of label column
Elec_min = Elec.min() # get min value of label column
print(f"The levels of electric consumption we have in our data are from
{Elec_min} to {Elec_max}")

HumData = pd.read_csv("Humidity/datadoubleremoved.csv", sep=",") # read
csv file
timeHum = HumData["time"] # get time column
Humidity = HumData["value"] # get Humidity column
Humidity_max = Humidity.max() # get max value of label column
Humidity_min = Humidity.min() # get min value of label column
print(f"The levels of humidity we have in our data are from
{Humidity_min} to {Humidity_max}")

SoundData = pd.read_csv("Sound/datadoubleremoved.csv", sep=",") # read
csv file
timeSound = SoundData["time"] # get time column
Sound = SoundData["value"] # get Sound column
Sound_max = Sound.max() # get max value of label column
Sound_min = Sound.min() # get min value of label column
print(f"The levels of sound we have in our data are from {Sound_min} to
{Sound_max}")

TempData = pd.read_csv("Temperature/datadoubleremoved.csv", sep=",") #
read csv file
timeTemp = TempData["time"] # get time column
Temp = TempData["value"] # get Temperature column
Temp_max = Temp.max() # get max value of label column
Temp_min = Temp.min() # get min value of label column
print(f"The levels of temperature indoors we have in our data are from
{Temp_min} to {Temp_max}")
```

2. Plot the different labels with the Heat

```
import matplotlib.pyplot as plt # import matplotlib library

# plot Temperature and Heat in the same plot with shared x axis of
concatenation ordered timeElec and timeHeat
# timeTemp and timeHeat are time dates as strs and they might be unordered
# so we need to order them and concatenate them to plot them in the same
plot
# the format for x axis is: '2022-02-10 23:03:18+00:00'
# we need to convert them to datetime objects to order them
fig, ax1 = plt.subplots(figsize=(30,10)) # create figure and first axis
ax2 = ax1.twinx() # create second axis with shared x axis
ax1.plot(pd.to_datetime(timeTemp),      Temp,      label="Temperature",
color="red") # plot Temperature data
ax2.plot(pd.to_datetime(timeHeat), Heat, label="Heat", color="blue") #
plot heat data
ax1.set_xlabel("Time") # set x label
ax1.set_ylabel("°C") # set y label
ax2.set_ylabel("Heat consumption") # set y label
ax1.legend(loc=2)
ax2.legend(loc=1)
plt.show() # show plot

# plot Elec and Heat in the same plot with shared x axis of concatenation
ordered timeElec and timeHeat
# timeElec and timeHeat are time dates as strs and they might be unordered
# so we need to order them and concatenate them to plot them in the same
plot
# the format for x axis is: '2022-02-10 23:03:18+00:00'
# we need to convert them to datetime objects to order them
fig, ax1 = plt.subplots(figsize=(30,10)) # create figure and first axis
ax2 = ax1.twinx() # create second axis with shared x axis
ax1.plot(pd.to_datetime(timeElec),      Elec,      label="Electricity",
color="red") # plot electricity data
ax2.plot(pd.to_datetime(timeHeat), Heat, label="Heat", color="blue") #
plot heat data
ax1.set_xlabel("Time") # set x label
ax1.set_ylabel("Electricity consumption") # set y label
ax2.set_ylabel("Heat consumption") # set y label
ax1.legend(loc=2)
ax2.legend(loc=1)
plt.show() # show plot

# plot humidity and Heat in the same plot with shared x axis of
concatenation ordered timeElec and timeHeat
# timeHum and timeHeat are time dates as strs and they might be unordered
# so we need to order them and concatenate them to plot them in the same
plot
# the format for x axis is: '2022-02-10 23:03:18+00:00'
# we need to convert them to datetime objects to order them
fig, ax1 = plt.subplots(figsize=(30,10)) # create figure and first axis
ax2 = ax1.twinx() # create second axis with shared x axis
ax1.plot(pd.to_datetime(timeHum),      Humidity,      label="Humidity",
color="red") # plot Humidity data
ax2.plot(pd.to_datetime(timeHeat), Heat, label="Heat", color="blue") #
plot heat data
ax1.set_xlabel("Time") # set x label
ax1.set_ylabel("%") # set y label
ax2.set_ylabel("Heat consumption") # set y label
ax1.legend(loc=2)
ax2.legend(loc=1)
```

```
plt.show() # show plot

# plot CO2 and Heat in the same plot with shared x axis of concatenation
ordered timeElec and timeHeat
# timeCO2 and timeHeat are time dates as strs and they might be unordered
# so we need to order them and concatenate them to plot them in the same
plot
# the format for x axis is: '2022-02-10 23:03:18+00:00'
# we need to convert them to datetime objects to order them
fig, ax1 = plt.subplots(figsize=(30,10)) # create figure and first axis
ax2 = ax1.twinx() # create second axis with shared x axis
ax1.plot(pd.to_datetime(timeCO2), CO2, label="CO2", color="red") # plot
CO2 data
ax2.plot(pd.to_datetime(timeHeat), Heat, label="Heat", color="blue") #
plot heat data
ax1.set_xlabel("Time") # set x label
ax1.set_ylabel("CO2") # set y label
ax2.set_ylabel("Heat consumption") # set y label
ax1.legend(loc=2)
ax2.legend(loc=1)
plt.show() # show plot

# plot sound and Heat in the same plot with shared x axis of concatenation
ordered timeElec and timeHeat
# timeSound and timeHeat are time dates as strs and they might be
unordered
# so we need to order them and concatenate them to plot them in the same
plot
# the format for x axis is: '2022-02-10 23:03:18+00:00'
# we need to convert them to datetime objects to order them
fig, ax1 = plt.subplots(figsize=(30,10)) # create figure and first axis
ax2 = ax1.twinx() # create second axis with shared x axis
ax1.plot(pd.to_datetime(timeSound), Sound, label="Sound", color="red")
# plot Sound data
ax2.plot(pd.to_datetime(timeHeat), Heat, label="Heat", color="blue") #
plot heat data
ax1.set_xlabel("Time") # set x label
ax1.set_ylabel("Sound dB") # set y label
ax2.set_ylabel("Heat consumption") # set y label
ax1.legend(loc=2)
ax2.legend(loc=1)
plt.show() # show plot
```

3. Implement the Decision tree (DT)

```
# Using scikit-learning library to train our model
from re import X
from sklearn.model_selection import train_test_split # import
train_test_split library

#split dataset in features and target variable
feature_CO2 = CO2Data.columns[1:2] # CO2 column
X = CO2Data[feature_CO2] # Feature = CO2
y = CO2Data.value # Target variable = label
x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42) # split data into training and testing data (70%
training and 30% testing)

from sklearn import tree
```

```
from sklearn.tree import DecisionTreeClassifier # Import Decision Tree
Classifier
from sklearn.tree import export_graphviz

classifier        =        tree.DecisionTreeClassifier(random_state=42,
criterion="entropy", max_depth=None) # create decision tree classifier
feature = x_train
occupancy = y_train # set feature and occupancy
classifier = classifier.fit(feature, occupancy) # fit the tree
#print(f"With an occupancy equal to {occupancy}, the accuracy is",
metrics.accuracy_score(y_train, y_test)) # print occupancy and accuracy
print(occupancy)
with open("tree.dot", 'w') as file:
    f = tree.export_graphviz(classifier, out_file=file) # save the tree
as dot file

# calculate the number of nodes and maximum depth of the tree
max_depth = classifier.tree_.max_depth
print(f"The maximum depth of the tree is {max_depth}")

from sklearn.model_selection import train_test_split
from sklearn import tree
from sklearn.tree import DecisionTreeClassifier # Import Decision Tree
Classifier
from sklearn.tree import export_graphviz

from sklearn import metrics
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import f1_score
from sklearn.metrics import recall_score

# plot accuracy with the depth
depth = [] # create an empty list for depth
accuracy = [] # create an empty list for accuracy
error_abs_iter = [] # create an empty list for error_abs_iter
f1_score_iter = [] # create an empty list for f1_score_iter

x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42) # split data into training and testing

for i in range(1, max_depth):
    classifier      =      tree.DecisionTreeClassifier(random_state=0,
criterion="entropy", max_depth=i) # create decision tree classifier
    feature = x_train
    occupancy = y_train # set feature and occupancy
    classifier = classifier.fit(feature, occupancy) # fit the tree
    predict = classifier.predict(x_test) # predict the occupancy
    accuracy.append(metrics.accuracy_score(y_test,     predict))     #
calculate the accuracy
    depth.append(i) # calculate the depth
    error_abs_iter.append(mean_absolute_error(y_test,    predict))    #
calculate the main absolute error
    f1_score_iter.append(f1_score(y_test, predict, average='weighted'))
# calculate the f-score

#plt.plot([i for i in range(1, max_depth)], accuracy)
#plt.xlabel('Depth')
#plt.ylabel('Accuracy')

#plt.plot([i for i in range(1, max_depth)], error_abs_iter)
#plt.xlabel('Depth')
#plt.ylabel('Error')
```

```
plt.plot([i for i in range(1, max_depth)], f1_score_iter)
plt.xlabel('Depth')
plt.ylabel('F-score')
```

4. Predict variables

PREDICT CO2

CO2 is a discrete variable, so we can use a Decision Tree to estimate it.

```
import pandas as pd # import pandas library
CO2Data = pd.read_csv("CO2/datadoubleremoved.csv", sep=",") # read csv
file

time = CO2Data["time"] # get time column
CO2 = CO2Data["value"] # get CO2 column

# Using scikit-learning library to train our model
from sklearn.model_selection import train_test_split # import
train_test_split library
from sklearn import tree
from sklearn.tree import DecisionTreeClassifier # Import Decision Tree
Classifier

# split dataset in features and target variable
feature_CO2 = CO2Data.columns[1:2] # CO2 column
X = CO2Data[feature_CO2] # Features = CO2
y = CO2Data.value # Target variable = label
x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=0) # split data into training and testing
classifier = tree.DecisionTreeClassifier(random_state=0,
criterion="entropy", max_depth=7) # create decision tree classifier
actual_CO2 = y_test
classifier = classifier.fit(x_train, y_train) # fit the tree
estimated_CO2 = classifier.predict(x_test) # predict the CO2

from sklearn import metrics
accuracy = metrics.accuracy_score(y_test, estimated_CO2) # calculate the
accuracy
print(f"The accuracy of the model is {accuracy}") # print the accuracy

from sklearn.metrics import mean_absolute_error
error_abs = mean_absolute_error(y_test, estimated_CO2) # calculate the
main absolute error
print(f"The main absolute error of the model is {error_abs}") # print
the main absolute error

from sklearn.metrics import f1_score
f1_score = f1_score(y_test, estimated_CO2, average='weighted') #
calculate the f-score
print(f"The f-score of the model is {f1_score}") # print the f-score

from sklearn.metrics import recall_score
recall_score = recall_score(y_test, estimated_CO2, average='weighted')
# calculate the recall score
print(f"The recall score of the model is {recall_score}") # print the
recall score

# delete the first columne of the actual CO2
```

```
actual_CO2 = actual_CO2.reset_index(drop=True)

# plot the actual CO2 and the estimated CO2
plt.plot(actual_CO2, label = "Actual CO2")
plt.plot(estimated_CO2, label = "Estimated CO2")
plt.legend()
#plt.xlabel('Time')
plt.ylabel('ppm CO2')
plt.title('Actual CO2 VS Estimated CO2')
plt.show()

[(i, j) for i,j in zip(y_test.to_list(), estimated_CO2)] # print the
actual CO2 and the estimated CO2
```

We can use a Random Forest Classifier to estimate the CO2 too.

```
import pandas as pd # import pandas library
CO2Data = pd.read_csv("CO2/datadoubleremoved.csv", sep=",") # read csv
file

time = CO2Data["time"] # get time column
CO2 = CO2Data["value"] # get CO2 column

# Using scikit-learning library to train our model
from sklearn.model_selection import train_test_split # import
train_test_split library
from sklearn import tree
from sklearn.ensemble import RandomForestClassifier

# split dataset in features and target variable
feature_CO2 = CO2Data.columns[1:2] # CO2 column
X = CO2Data[feature_CO2] # Features = CO2
y = CO2Data.value # Target variable = label
x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=0) # split data into training and testing
classifier = RandomForestClassifier(criterion="entropy")
actual_CO2 = y_test
classifier = classifier.fit(x_train, y_train) # fit the tree
estimated_CO2 = classifier.predict(x_test) # predict the CO2

from sklearn import metrics
accuracy = metrics.accuracy_score(y_test, estimated_CO2) # calculate the
accuracy
print(f"The accuracy of the model is {accuracy}") # print the accuracy

from sklearn.metrics import mean_absolute_error
error_abs = mean_absolute_error(y_test, estimated_CO2) # calculate the
main absolute error
print(f"The main absolute error of the model is {error_abs}") # print
the main absolute error

from sklearn.metrics import f1_score
f1 = f1_score(y_test, estimated_CO2, average='weighted') # calculate the
f-score
print(f"The f-score of the model is {f1}") # print the f-score

from sklearn.metrics import recall_score
recall_score = recall_score(y_test, estimated_CO2, average='weighted')
# calculate the recall score
print(f"The recall score of the model is {recall_score}") # print the
recall score
```

```
# delete the first columne of the actual CO2
actual_CO2 = actual_CO2.reset_index(drop=True)

# plot the actual CO2 and the estimated CO2
plt.plot(actual_CO2, label = "Actual CO2")
plt.plot(estimated_CO2, label = "Estimated CO2")
plt.legend()
plt.xlabel('Time')
plt.ylabel('ppm CO2')
plt.title('Actual CO2 VS Estimated CO2')
plt.show()

[(i, j) for i,j in zip(y_test.to_list(), estimated_CO2)] # print the
actual CO2 and the estimated CO2
```

PREDICT SOUND

Sound is a discrete variable, so we can use a Decision Tree to estimate it.

```
import pandas as pd # import pandas library
SoundData = pd.read_csv("Sound/datadoubleremoved.csv", sep=",") # read
csv file

timeSound = SoundData["time"] # get time column
Sound = SoundData["value"] # get Sound column

# Using scikit-learning library to train our model
from sklearn.model_selection import train_test_split # import
train_test_split library
from sklearn import tree
from sklearn.tree import DecisionTreeClassifier # Import Decision Tree
Classifier

# split dataset in features and target variable
feature_Sound = SoundData.columns[1:2] # sound column
X = SoundData[feature_Sound] # Features = sound
y = SoundData.value # Target variable = label
x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=0) # split data into training and testing
classifier = tree.DecisionTreeClassifier(random_state=0,
criterion="entropy", max_depth=7) # create decision tree classifier
actual_Sound = y_test
classifier = classifier.fit(x_train, y_train) # fit the tree
estimated_Sound = classifier.predict(x_test) # predict the sound

from sklearn import metrics
accuracy = metrics.accuracy_score(y_test, estimated_Sound) # calculate
the accuracy
print(f"The accuracy of the model is {accuracy}") # print the accuracy

from sklearn.metrics import mean_absolute_error
error_abs = mean_absolute_error(y_test, estimated_Sound) # calculate the
main absolute error
print(f"The main absolute error of the model is {error_abs}") # print
the main absolute error

from sklearn.metrics import f1_score
f1_score = f1_score(y_test, estimated_Sound, average='weighted') #
calculate the f-score
print(f"The f-score of the model is {f1_score}") # print the f-score
```

```
from sklearn.metrics import recall_score
recall_score = recall_score(y_test, estimated_Sound,
average='weighted') # calculate the recall score
print(f"The recall score of the model is {recall_score}") # print the
recall score

# delete the first columne of the actual sound
actual_Sound = actual_Sound.reset_index(drop=True)

# plot the actual sound and the estimated sound
plt.plot(actual_Sound, label = "Actual Sound")
plt.plot(estimated_Sound, label = "Estimated Sound")
plt.legend()
plt.ylabel('dB')
plt.title('Actual VS Estimated Sound')
plt.show()

[(i, j) for i,j in zip(y_test.to_list(), estimated_Sound)] # print the
actual sound and the estimated sound
```

PREDICT ELECTRICITY CONSUMPTION

Electricity consumption is a continuous variable. It is not possible to use DT in this case. We are going to use support vector machine method (SVM) to predict electricity consumption.

```
import pandas as pd # import pandas library
ElecData = pd.read_csv("Electricity/datadoubleremoved.csv", sep=",") #
read csv file

time = ElecData["time"] # get time column
Elec = ElecData["value"] # get electricity column

#split dataset in features and target variable
feature_Elec = ElecData.columns[1:2] # electricity column
X = ElecData[feature_Elec] # Features = electricity
y = ElecData.value # Target variable = label
x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=0) # split data into training and testing

# use svm regression to predict the electricity consumption
from sklearn.svm import SVR
svr_rbf = SVR(kernel='rbf', C=1e3, gamma=0.1) # set the kernel to rbf
and fit the model
estimated_Elec = svr_rbf.fit(x_train, y_train).predict(x_test) # predict
the electricity consumption

from sklearn.metrics import mean_absolute_error
error_abs = mean_absolute_error(y_test, estimated_Elec) # calculate the
main absolute error
print(f"The main absolute error of the model is {error_abs}") # print
the main absolute error

plt.plot(x_test["value"].to_list(), label = 'Actual', color='blue')
plt.plot(estimated_Elec, label = 'Estimated', color='red')
plt.ylabel('kW')
plt.legend(loc='upper right')
plt.title('Actual VS Estimated Electricity Consumption')
plt.show()
```

```
[(i, j) for i,j in zip(y_test.to_list(), estimated_Elec)] # print the
actual electricity consumption and the estimated one
```

PREDICT INDOOR TEMPERATURE

Temperature is a continuous variable. It is not possible to use DT in this case. We are going to use support vector machine method (SVM) to predict the indoor temperature.

```
import pandas as pd # import pandas library
TempData = pd.read_csv("Temperature/datadoubleremoved.csv", sep=",") #
read csv file

time = TempData["time"] # get time column
Temp = TempData["value"] # get temperature column

#split dataset in features and target variable
feature_Temp = TempData.columns[1:2] # electricity column
X = TempData[feature_Temp] # Features = electricity
y = TempData.value # Target variable = label
x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=0) # split data into training and testing

# use svm regression to predict the temperature
from sklearn.svm import SVR
svr_rbf = SVR(kernel='rbf', C=1e3, gamma=0.1) # set the kernel to rbf
and fit the model
estimated_Temp = svr_rbf.fit(x_train, y_train).predict(x_test) # predict
the temperature

from sklearn.metrics import mean_absolute_error
error_abs = mean_absolute_error(y_test, estimated_Temp) # calculate the
main absolute error
print(f"The main absolute error of the model is {error_abs}") # print
the main absolute error

plt.plot(x_test["value"].to_list(),  label  =  'Actual  Temperature',
color='blue')
plt.plot(estimated_Temp, label = 'Estimated', color='red')
plt.ylabel('°C')
plt.legend()
plt.title('Actual VS Estimated Indoor Temperature')
plt.show()

[(i, j) for i,j in zip(y_test.to_list(), estimated_Temp)] # print the
actual temperature and the estimated one
```

PREDICT HUMIDITY

Humidity is a continuous variable. It is not possible to use DT in this case. We are going to use support vector machine method (SVM) to predict the humidity.

```
import pandas as pd # import pandas library
HumData = pd.read_csv("Humidity/datadoubleremoved.csv", sep=",") # read
csv file

time = HumData["time"] # get time column
Hum = HumData["value"] # get humidity column
```

```
#split dataset in features and target variable
feature_Hum = HumData.columns[1:2] # humidity column
X = HumData[feature_Hum] # Features = humidity
y = HumData.value # Target variable = label
x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=0) # split data into training and testing

# use svm regression to predict the temperature
from sklearn.svm import SVR
svr_rbf = SVR(kernel='rbf', C=1e3, gamma=0.1) # set the kernel to rbf
and fit the model
estimated_Hum = svr_rbf.fit(x_train, y_train).predict(x_test) # predict
the temperature

from sklearn.metrics import mean_absolute_error
error_abs = mean_absolute_error(y_test, estimated_Hum) # calculate the
main absolute error
print(f"The main absolute error of the model is {error_abs}") # print
the main absolute error

plt.plot(x_test["value"].to_list(),   label   =   'Actual   Humidity',
color='blue')
plt.plot(estimated_Hum, label = 'Estimated', color='red')
plt.ylabel('%')
plt.legend(loc='lower right')
plt.title('Actual VS Estimated Humidity')
plt.show()

[(i, j) for i,j in zip(y_test.to_list(), estimated_Hum)] # print the
actual humidity and the estimated one
```