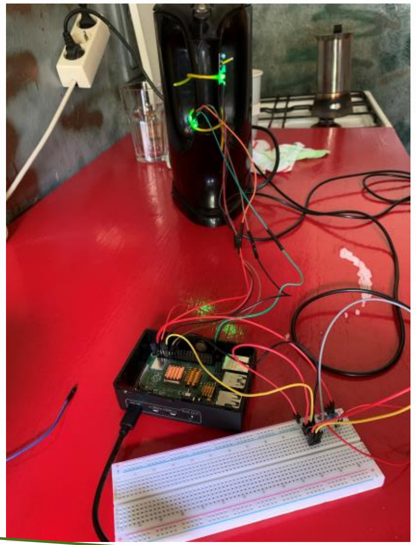
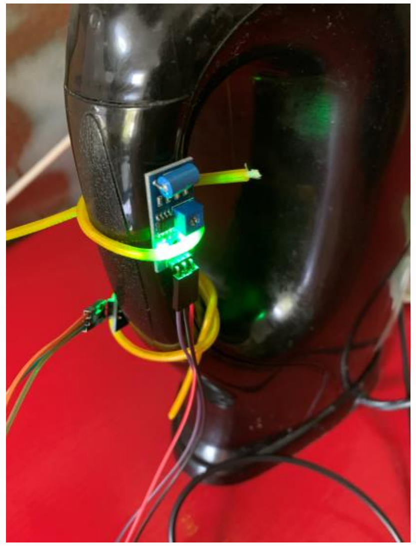
# INTERNET OF THINGS AND MACHINE LEARNING USING THE EXAMPLE OF PREDICTIVE MAINTENANCE

## WHAT IS THE INTERNET OF THINGS?

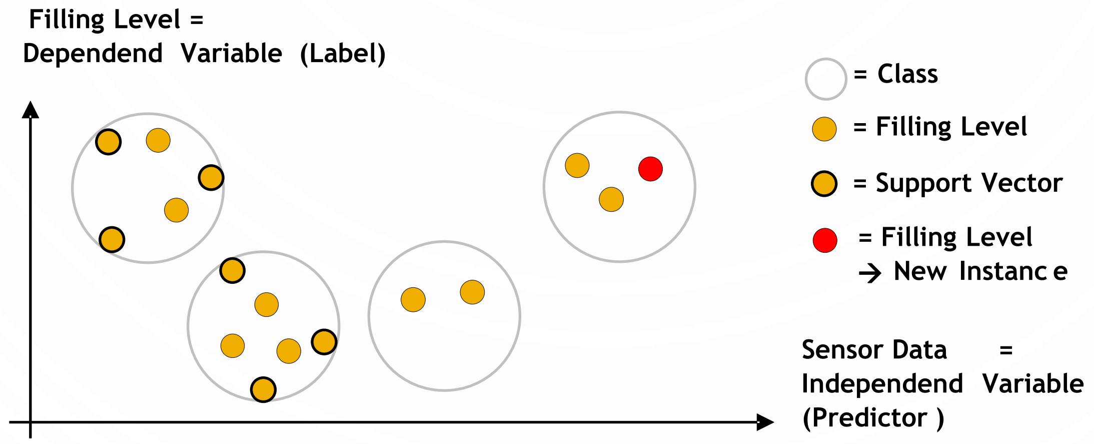
"The Internet of Things (IoT) refers to the connectivity of physical objects, equipped with sensors and actuators, to the Internet via data communication technology, enabling interaction with and/or among these objects."

## **PROJECT QUESTION:** IS IT POSSIBLE TO PREDICT THE FILLING LEVEL OF AN ELECTRIC KETTLE WITH SENSOR DATA?

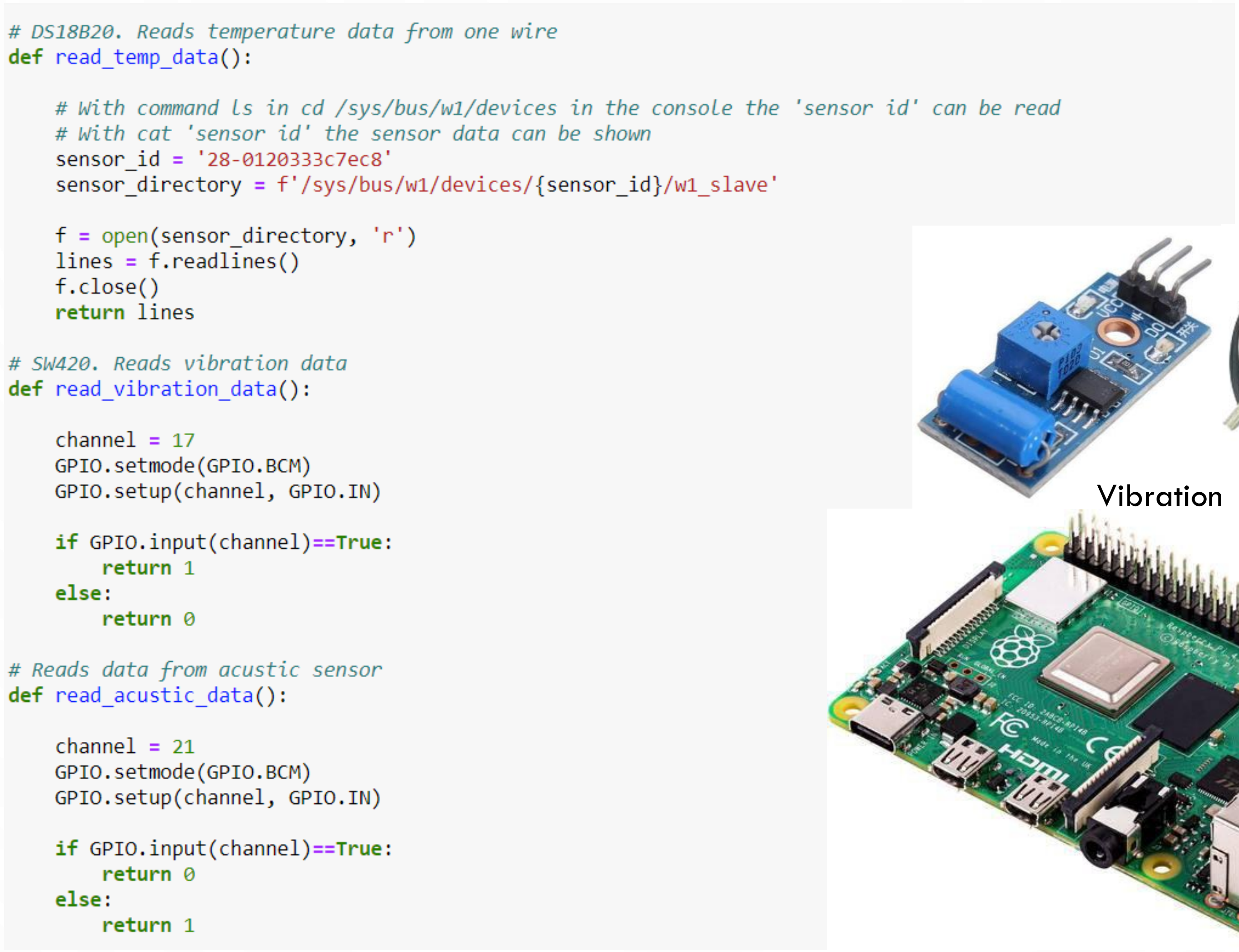



✓ Prediction with Machine Learning Approach

# PREDICTION BASED ON SUPPORT VECTOR MACHINE!



# DATA GATHERING WITH RASPBERRY PI AND SENSORS

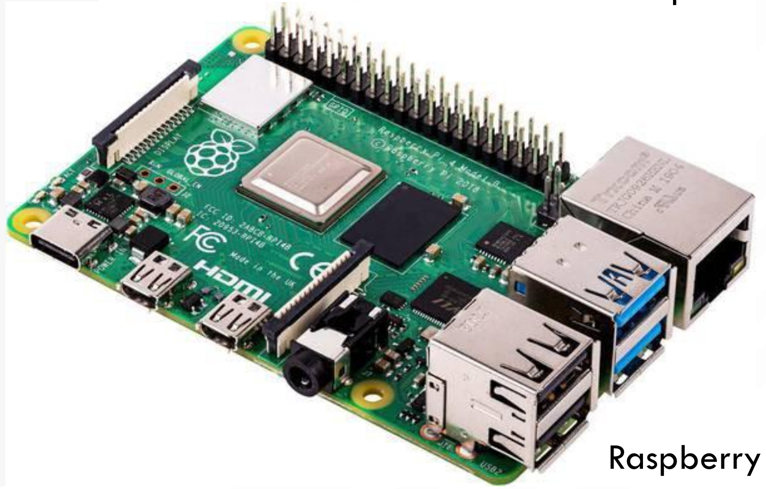


```python
# DS18B20. Reads temperature data from one wire
def read_temp_data():

    # With command ls in cd /sys/bus/w1/devices in the console the 'sensor id' can be read
    # With cat 'sensor id' the sensor data can be shown
    sensor_id = '28-0120333c7ec8'
    sensor_directory = f'/sys/bus/w1/devices/{sensor_id}/w1_slave'

    f = open(sensor_directory, 'r')
    lines = f.readlines()
    f.close()
    return lines

# SW420. Reads vibration data
def read_vibration_data():

    channel = 17
    GPIO.setmode(GPIO.BCM)
    GPIO.setup(channel, GPIO.IN)

    if GPIO.input(channel)==True:
        return 1
    else:
        return 0

# Reads data from acustic sensor
def read_acustic_data():

    channel = 21
    GPIO.setmode(GPIO.BCM)
    GPIO.setup(channel, GPIO.IN)

    if GPIO.input(channel)==True:
        return 0
    else:
        return 1
```

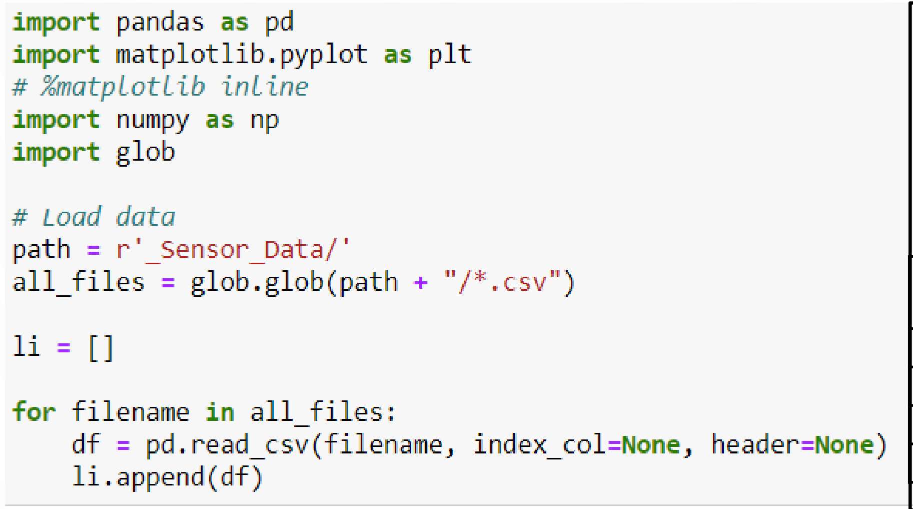
Sound

Vibration

Temperature


Raspberry Pi

## DATA IMPORT



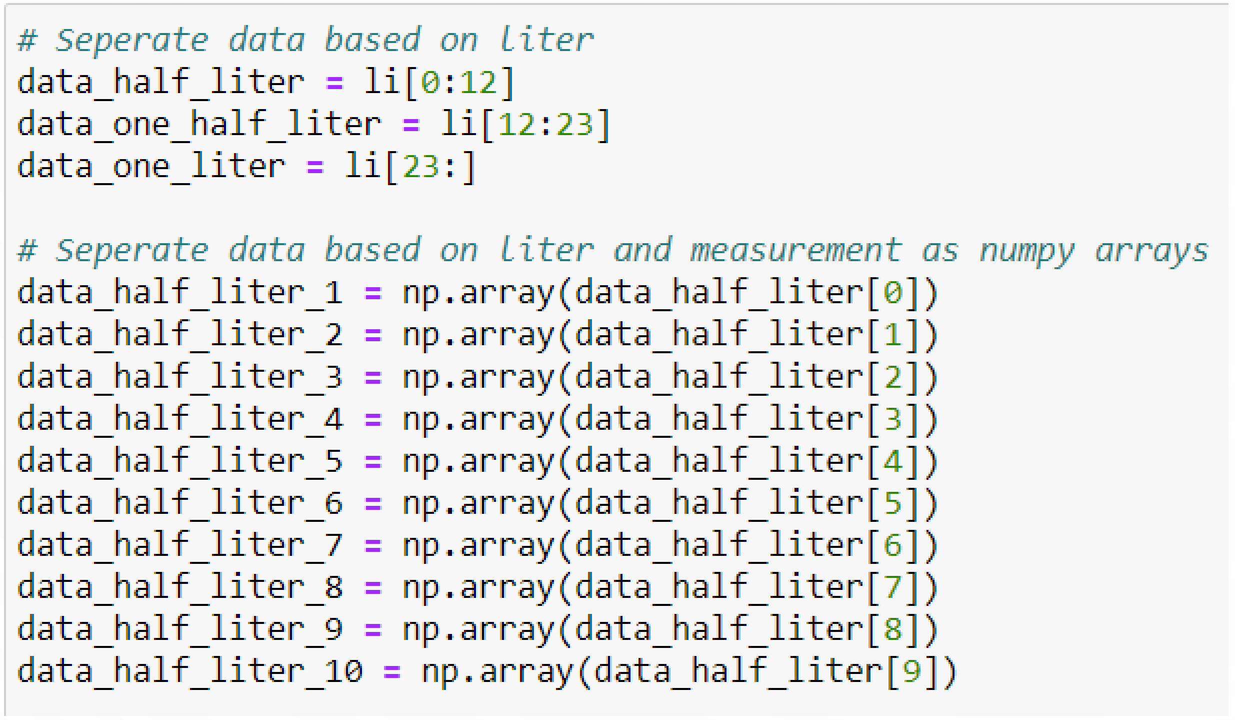
```
import pandas as pd
import matplotlib.pyplot as plt
# %matplotlib inline
import numpy as np
import glob

# Load data
path = r'_Sensor_Data/'
all_files = glob.glob(path + "/*.csv")

li = []

for filename in all_files:
    df = pd.read_csv(filename, index_col=None, header=None)
    li.append(df)
```

- Gathering data for filling levels 0.5, 1.0 and 1.5 litre
- In sum about 6700 landmarks
  (i.e. about 220 landmarks per measurement)

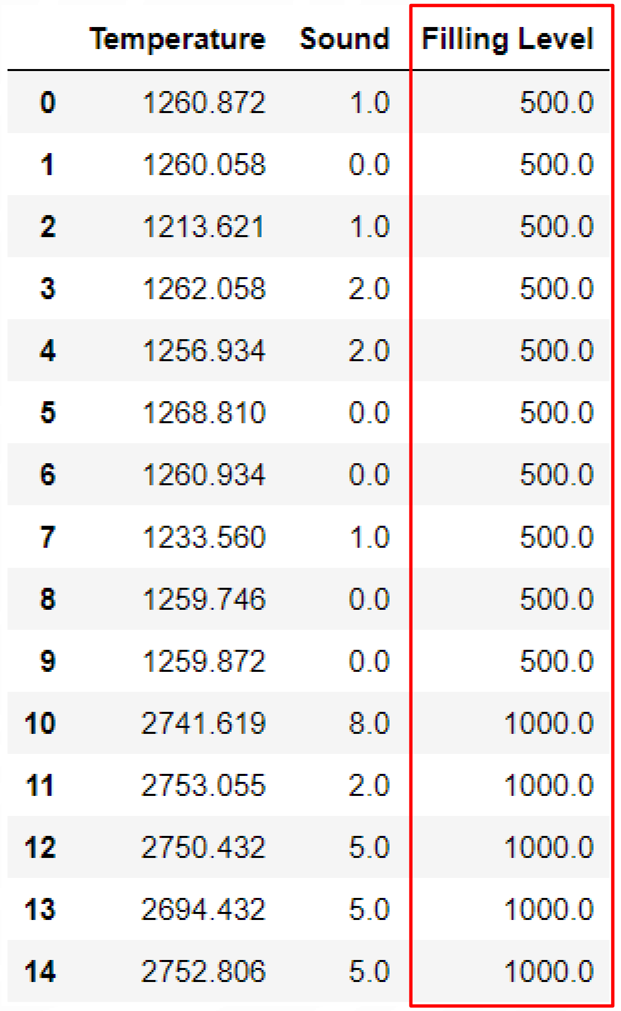
| Time Stamp | Temperature Raw | Temperature | Vibration | Acustic |
|---|---|---|---|---|
| 2020-09-16 15:18:14.564406 | 92437 | 92.437 | 0 | 0 |
| 2020-09-16 15:18:15.524435 | 92375 | 92.375 | 0 | 0 |
| 2020-09-16 15:18:16.484355 | 92312 | 92.312 | 0 | 1 |
| … | … | … | … | … |

Excluding vibration sensor, since the sensor was not sensitive enough

## DATA PREPARATION AND LABELING

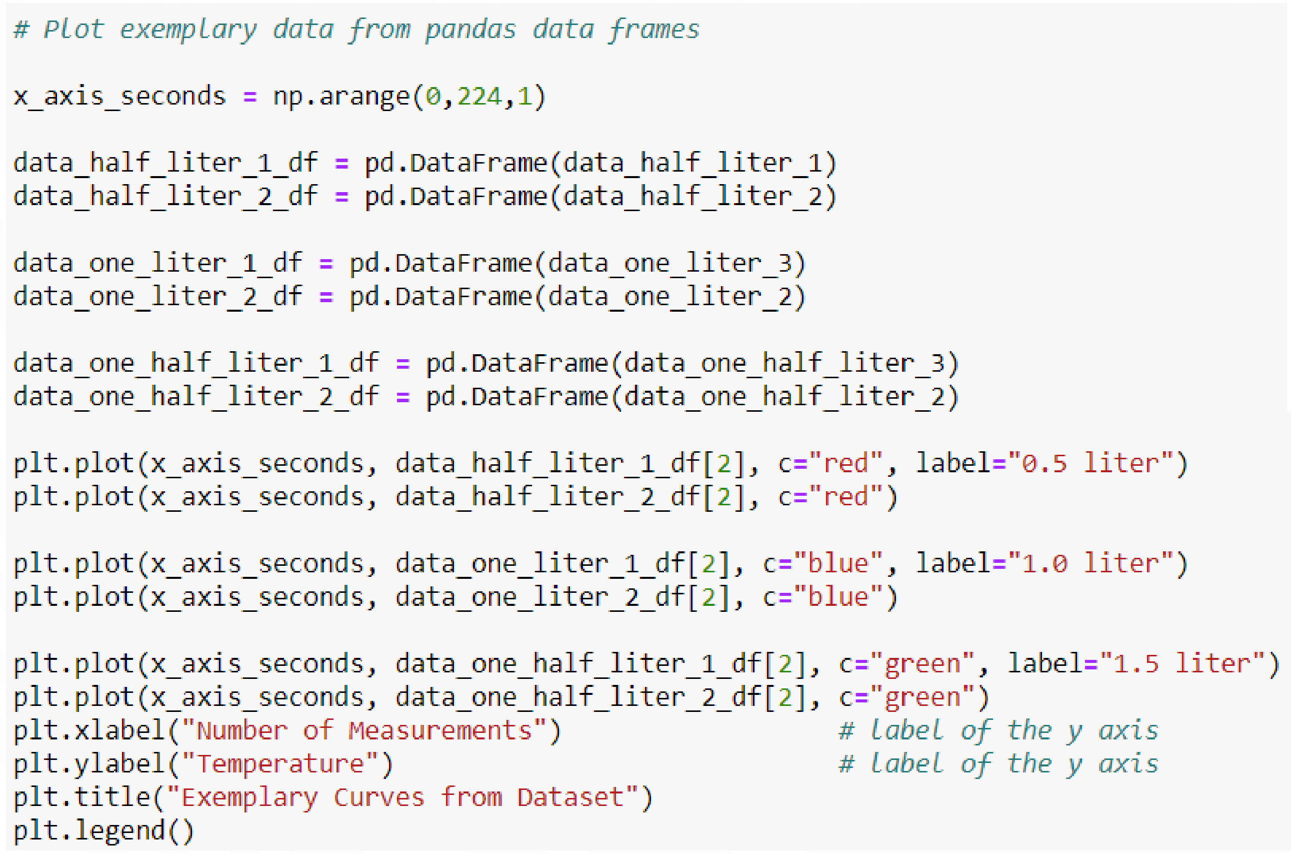


```
# Seperate data based on liter
data_half_liter = li[0:12]
data_one_half_liter = li[12:23]
data_one_liter = li[23:]

# Seperate data based on liter and measurement as numpy arrays
data_half_liter_1 = np.array(data_half_liter[0])
data_half_liter_2 = np.array(data_half_liter[1])
data_half_liter_3 = np.array(data_half_liter[2])
data_half_liter_4 = np.array(data_half_liter[3])
data_half_liter_5 = np.array(data_half_liter[4])
data_half_liter_6 = np.array(data_half_liter[5])
data_half_liter_7 = np.array(data_half_liter[6])
data_half_liter_8 = np.array(data_half_liter[7])
data_half_liter_9 = np.array(data_half_liter[8])
data_half_liter_10 = np.array(data_half_liter[9])
```

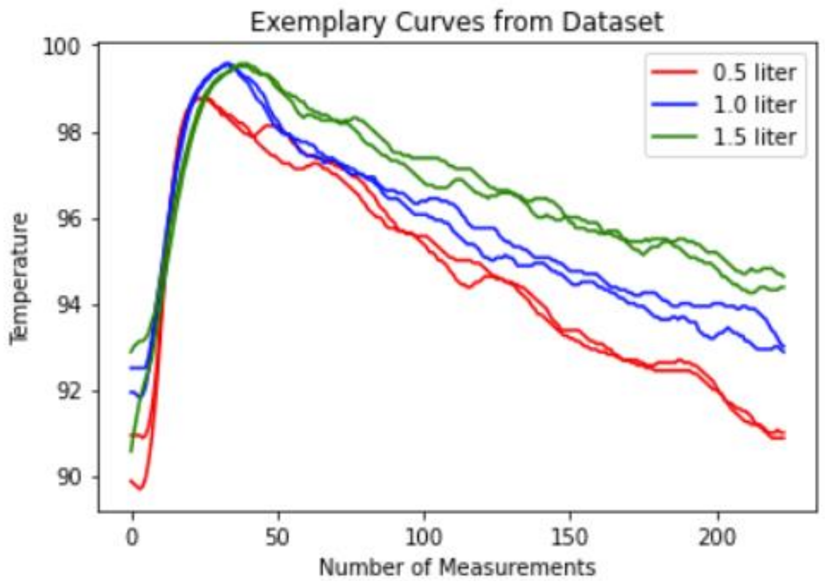


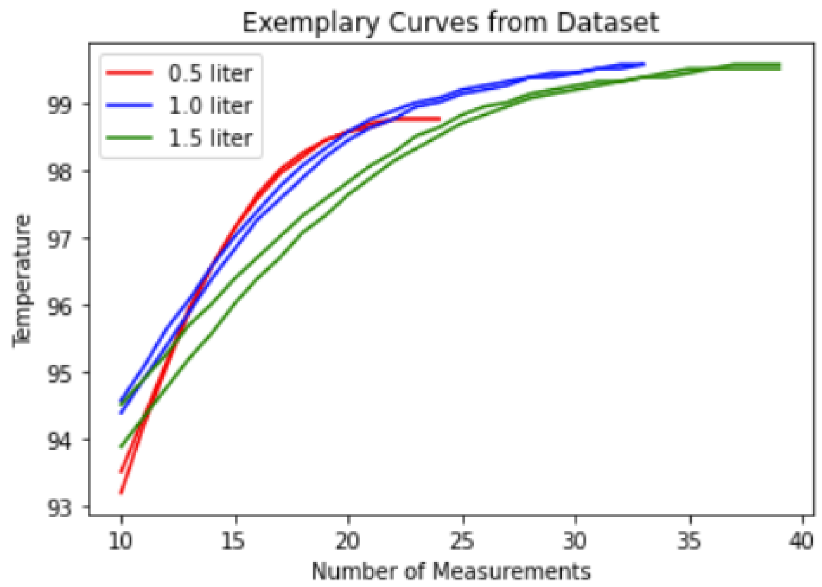
| | Temperature | Sound | Filling Level |
|---|---|---|---|
| **0** | 1260.872 | 1.0 | 500.0 |
| **1** | 1260.058 | 0.0 | 500.0 |
| **2** | 1213.621 | 1.0 | 500.0 |
| **3** | 1262.058 | 2.0 | 500.0 |
| **4** | 1256.934 | 2.0 | 500.0 |
| **5** | 1268.810 | 0.0 | 500.0 |
| **6** | 1260.934 | 0.0 | 500.0 |
| **7** | 1233.560 | 1.0 | 500.0 |
| **8** | 1259.746 | 0.0 | 500.0 |
| **9** | 1259.872 | 0.0 | 500.0 |
| **10** | 2741.619 | 8.0 | 1000.0 |
| **11** | 2753.055 | 2.0 | 1000.0 |
| **12** | 2750.432 | 5.0 | 1000.0 |
| **13** | 2694.432 | 5.0 | 1000.0 |
| **14** | 2752.806 | 5.0 | 1000.0 |

## DATA VISUALIZATION

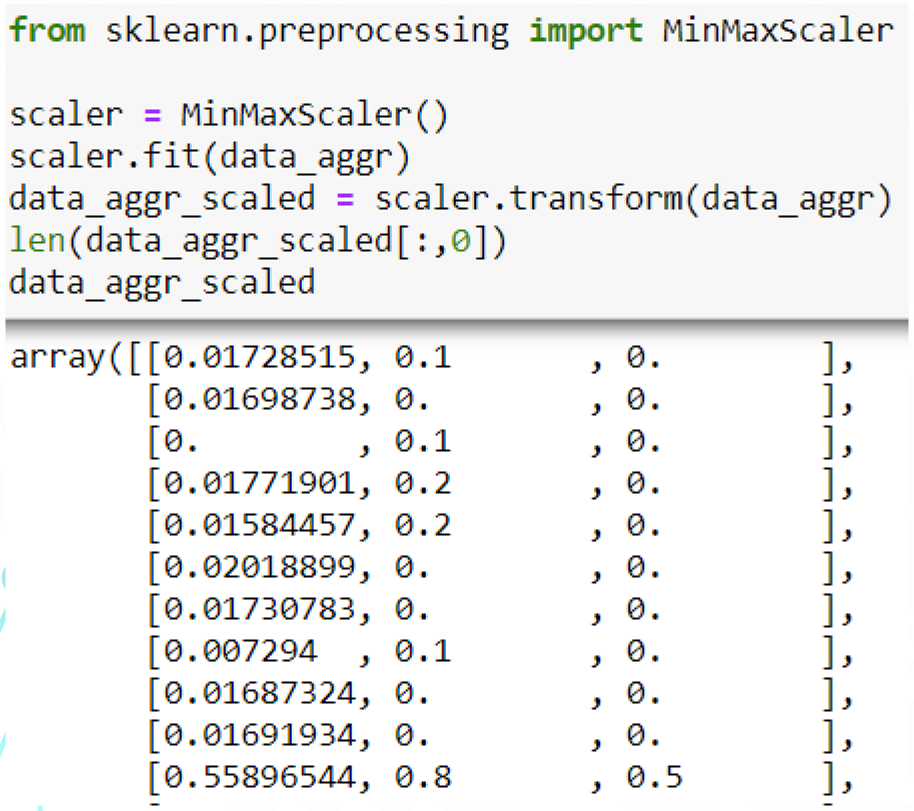


```
# Plot exemplary data from pandas data frames

x_axis_seconds = np.arange(0,224,1)

data_half_liter_1_df = pd.DataFrame(data_half_liter_1)
data_half_liter_2_df = pd.DataFrame(data_half_liter_2)

data_one_liter_1_df = pd.DataFrame(data_one_liter_3)
data_one_liter_2_df = pd.DataFrame(data_one_liter_2)

data_one_half_liter_1_df = pd.DataFrame(data_one_half_liter_3)
data_one_half_liter_2_df = pd.DataFrame(data_one_half_liter_2)

plt.plot(x_axis_seconds, data_half_liter_1_df[2], c="red", label="0.5 liter")
plt.plot(x_axis_seconds, data_half_liter_2_df[2], c="red")

plt.plot(x_axis_seconds, data_one_liter_1_df[2], c="blue", label="1.0 liter")
plt.plot(x_axis_seconds, data_one_liter_2_df[2], c="blue")

plt.plot(x_axis_seconds, data_one_half_liter_1_df[2], c="green", label="1.5 liter")
plt.plot(x_axis_seconds, data_one_half_liter_2_df[2], c="green")
plt.xlabel("Number of Measurements")                    # label of the y axis
plt.ylabel("Temperature")                               # label of the y axis
plt.title("Exemplary Curves from Dataset")
plt.legend()
```



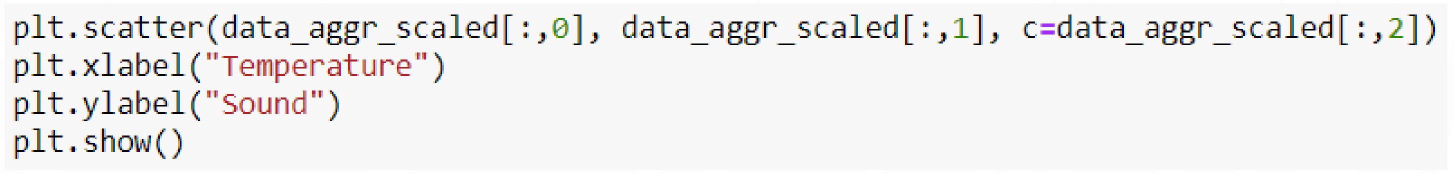


# STANDARDIZATION OF DATA



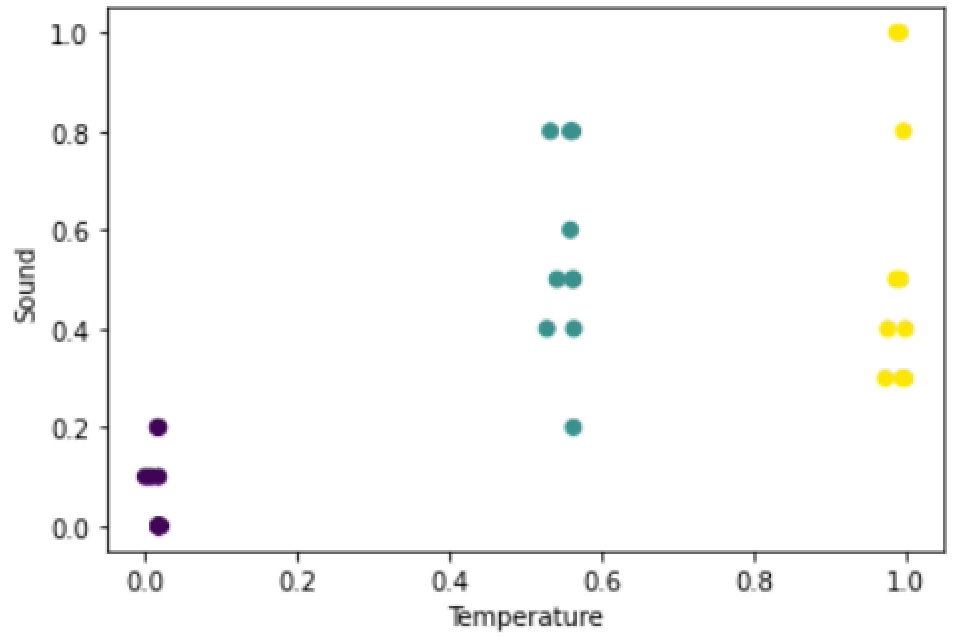
```
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()
scaler.fit(data_aggr)
data_aggr_scaled = scaler.transform(data_aggr)
len(data_aggr_scaled[:,0])
data_aggr_scaled
```

```
array([[0.01728515, 0.1       , 0.        ],
       [0.01698738, 0.        , 0.        ],
       [0.        , 0.1       , 0.        ],
       [0.01771901, 0.2       , 0.        ],
       [0.01584457, 0.2       , 0.        ],
       [0.02018899, 0.        , 0.        ],
       [0.01730783, 0.        , 0.        ],
       [0.007294  , 0.1       , 0.        ],
       [0.01687324, 0.        , 0.        ],
       [0.01691934, 0.        , 0.        ],
       [0.55896544, 0.8       , 0.5       ],
```

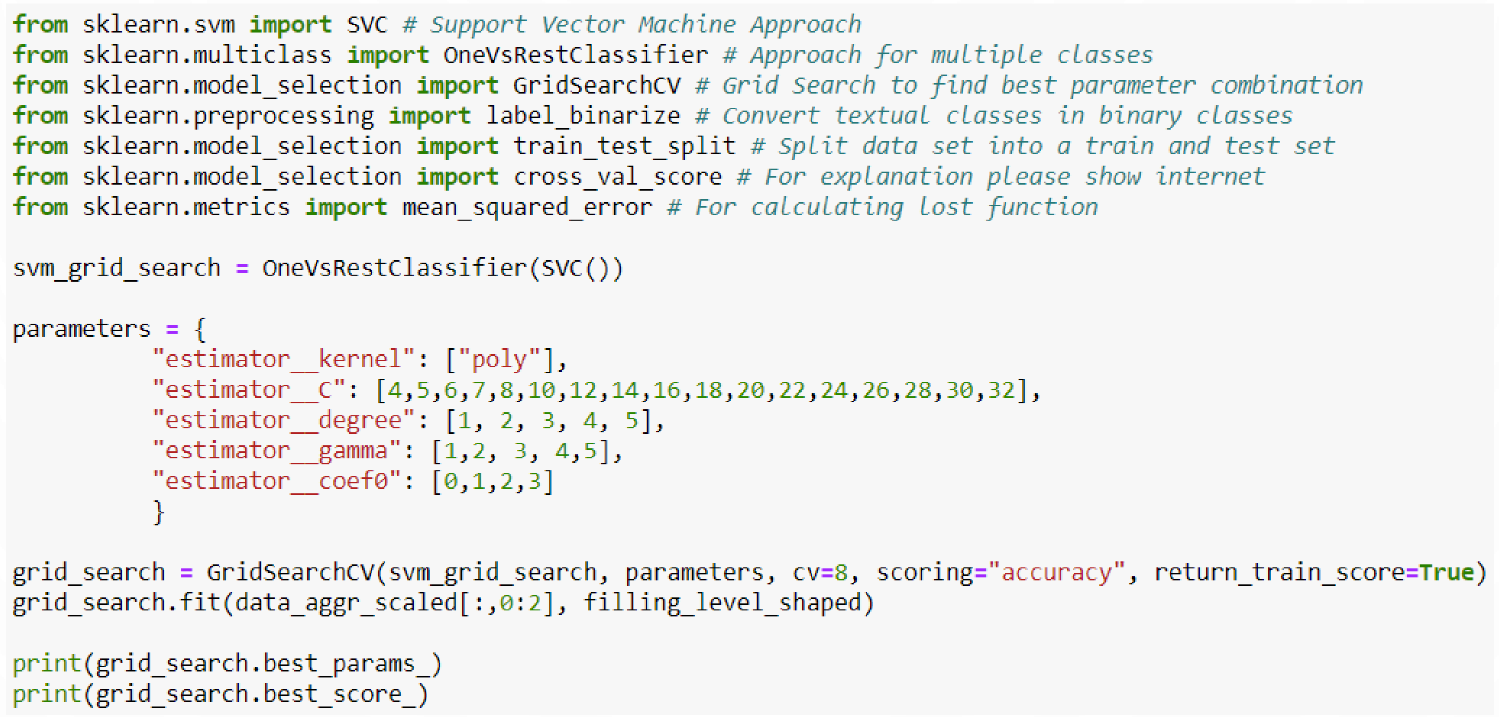


```
plt.scatter(data_aggr_scaled[:,0], data_aggr_scaled[:,1], c=data_aggr_scaled[:,2])
plt.xlabel("Temperature")
plt.ylabel("Sound")
plt.show()
```

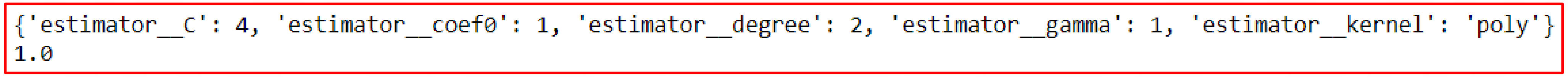
Visualiziation Combination Sound and Temperature



## SELECT, APPLY AND FINE-TUNE MODEL (POLY)

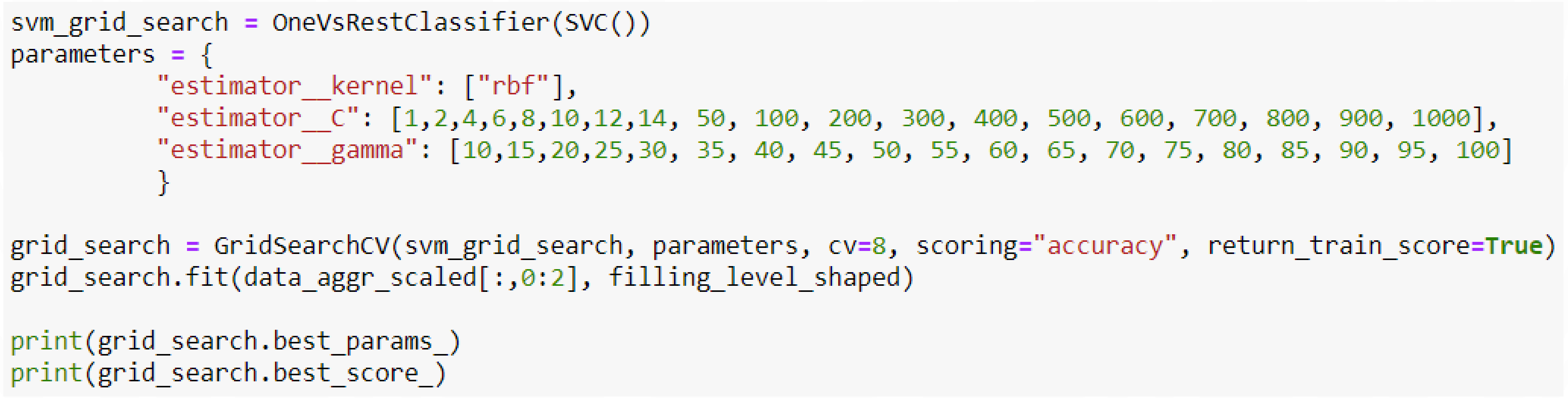


```python
from sklearn.svm import SVC # Support Vector Machine Approach
from sklearn.multiclass import OneVsRestClassifier # Approach for multiple classes
from sklearn.model_selection import GridSearchCV # Grid Search to find best parameter combination
from sklearn.preprocessing import label_binarize # Convert textual classes in binary classes
from sklearn.model_selection import train_test_split # Split data set into a train and test set
from sklearn.model_selection import cross_val_score # For explanation please show internet
from sklearn.metrics import mean_squared_error # For calculating lost function

svm_grid_search = OneVsRestClassifier(SVC())

parameters = {
          "estimator__kernel": ["poly"],
          "estimator__C": [4,5,6,7,8,10,12,14,16,18,20,22,24,26,28,30,32],
          "estimator__degree": [1, 2, 3, 4, 5],
          "estimator__gamma": [1,2, 3, 4,5],
          "estimator__coef0": [0,1,2,3]
          }

grid_search = GridSearchCV(svm_grid_search, parameters, cv=8, scoring="accuracy", return_train_score=True)
grid_search.fit(data_aggr_scaled[:,0:2], filling_level_shaped)

print(grid_search.best_params_)
print(grid_search.best_score_)
```

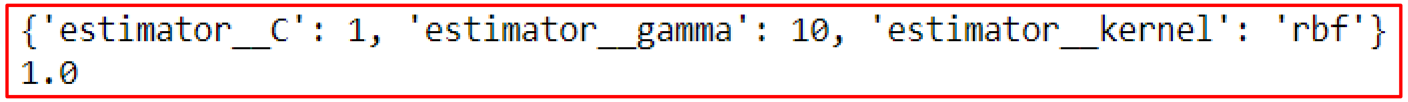


```
{'estimator__C': 4, 'estimator__coef0': 1, 'estimator__degree': 2, 'estimator__gamma': 1, 'estimator__kernel': 'poly'}
1.0
```

# SELECT, APPLY AND FINE-TUNE MODEL (RBF)

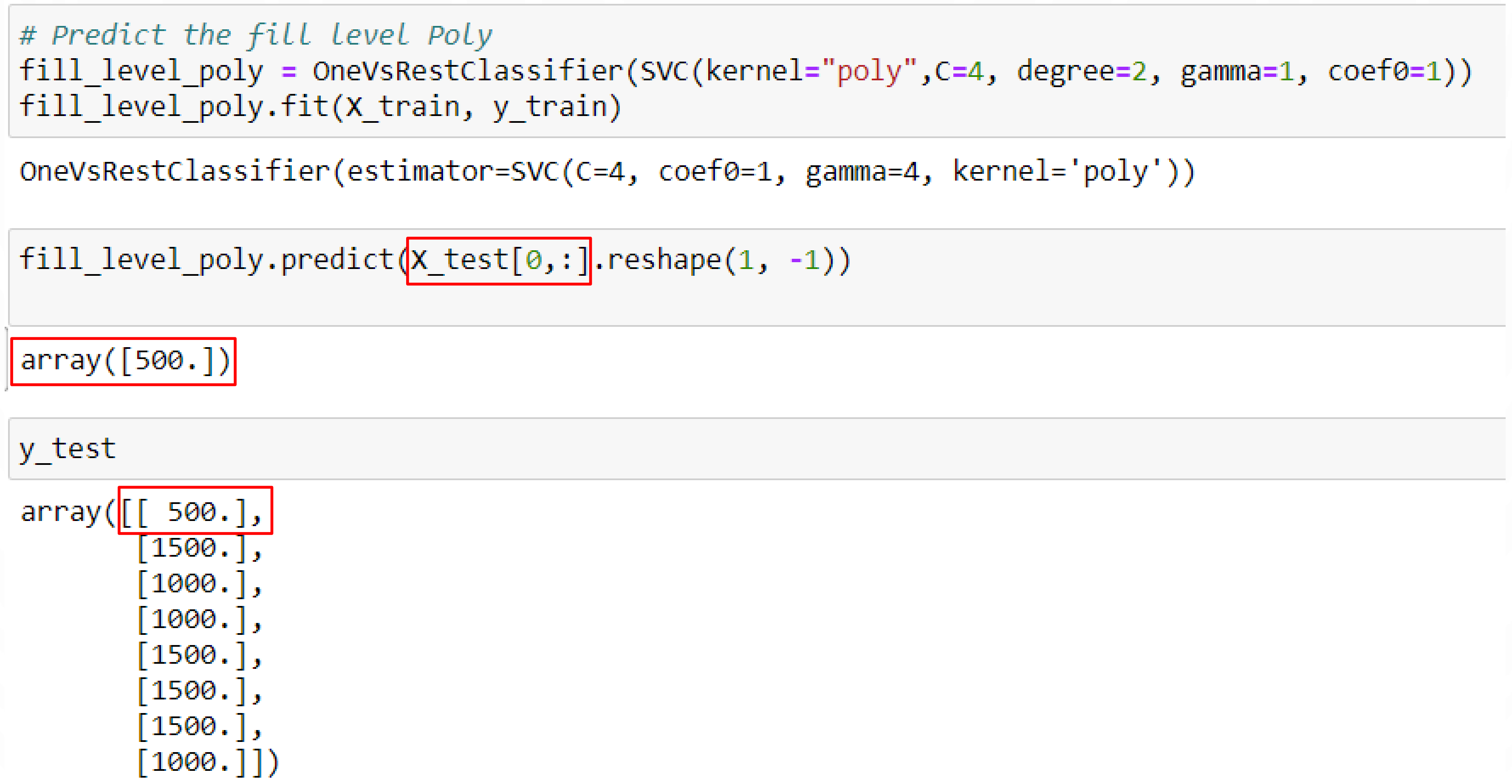


```python
svm_grid_search = OneVsRestClassifier(SVC())
parameters = {
          "estimator__kernel": ["rbf"],
          "estimator__C": [1,2,4,6,8,10,12,14, 50, 100, 200, 300, 400, 500, 600, 700, 800, 900, 1000],
          "estimator__gamma": [10,15,20,25,30, 35, 40, 45, 50, 55, 60, 65, 70, 75, 80, 85, 90, 95, 100]
          }

grid_search = GridSearchCV(svm_grid_search, parameters, cv=8, scoring="accuracy", return_train_score=True)
grid_search.fit(data_aggr_scaled[:,0:2], filling_level_shaped)

print(grid_search.best_params_)
print(grid_search.best_score_)
```



```
{'estimator__C': 1, 'estimator__gamma': 10, 'estimator__kernel': 'rbf'}
1.0
```

## PREDICTION WITH THE MODEL (POLY)



```
# Predict the fill level Poly
fill_level_poly = OneVsRestClassifier(SVC(kernel="poly",C=4, degree=2, gamma=1, coef0=1))
fill_level_poly.fit(X_train, y_train)
```

```
OneVsRestClassifier(estimator=SVC(C=4, coef0=1, gamma=4, kernel='poly'))
```

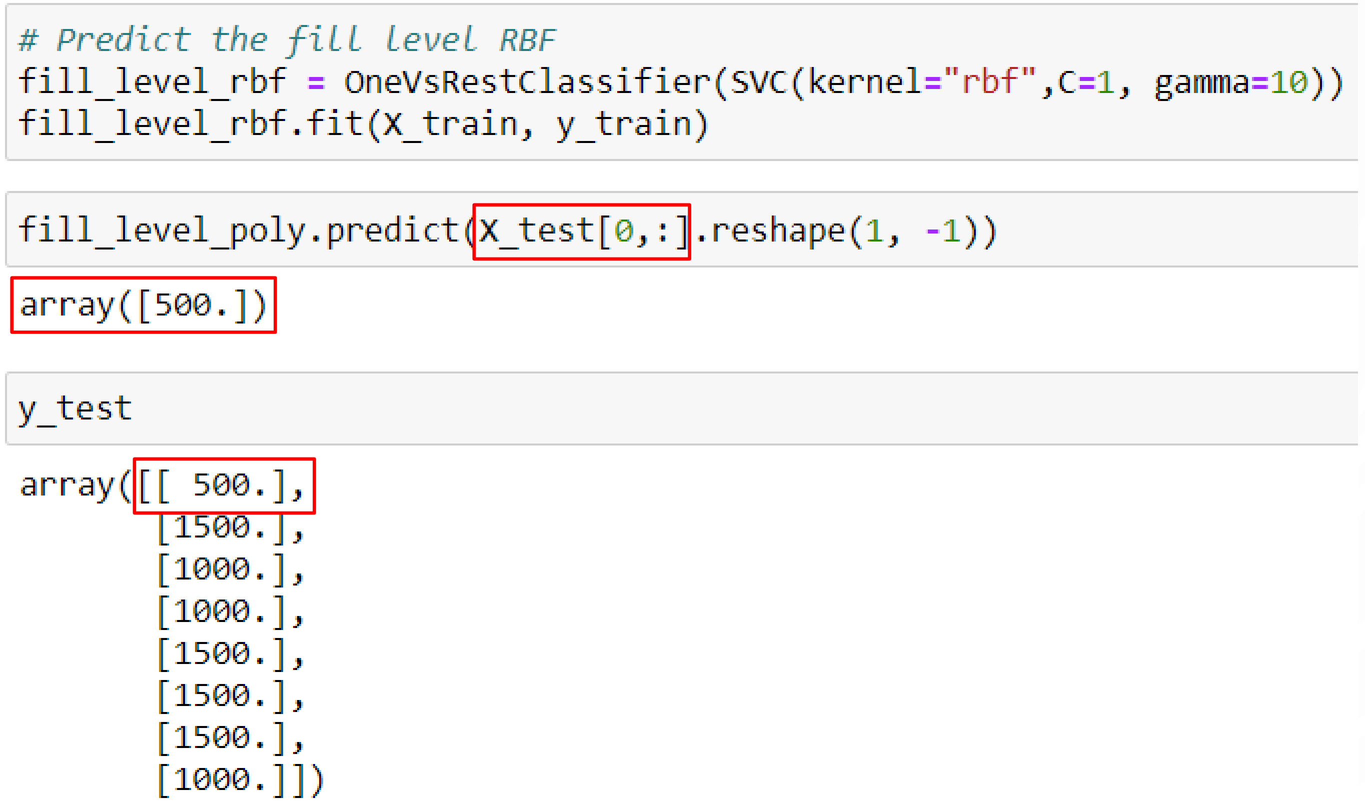
```
fill_level_poly.predict(X_test[0,:].reshape(1, -1))
```

```
array([500.])
```

```
y_test
```

```
array([[ 500.],
       [1500.],
       [1000.],
       [1000.],
       [1500.],
       [1500.],
       [1500.],
       [1000.]])
```

## PREDICTION WITH THE MODEL (RBF)



```
# Predict the fill level RBF
fill_level_rbf = OneVsRestClassifier(SVC(kernel="rbf",C=1, gamma=10))
fill_level_rbf.fit(X_train, y_train)
```

```
fill_level_poly.predict(X_test[0,:].reshape(1, -1))
```

```
array([500.])
```

```
y_test
```

```
array([[ 500.],
       [1500.],
       [1000.],
       [1000.],
       [1500.],
       [1500.],
       [1500.],
       [1000.]])
```

# CONCLUSION

**Learnings**

- High effort in data acquisition using Raspberry Pis and associated sensor technology
- Simple approaches are sometimes more effective
- Some sensors do not contribute to the explanation of the model, since they are unsuitable

- End-to-end data analytics or machine learning project was implemented
  - From big picture to data collection and evaluation to presentation
  - Especially how to collect data with hardware and sensors
  - Challenge solved: Unstructured initial real world use case