

# Identifying Outliers

Module 1 | Chapter 5 | Notebook 2

---

In this chapter we will get to grips with the concept of **outliers**. For this we will look at real sensor data from hydraulic pumps. By the end of the chapter, you will know:

- How to identify outliers.
  - What causes outliers.
  - How you can visualize outliers.
- 

## Sensor data as time series in a database

**Scenario:** You work for a company that manufactures hydraulic pumps. A pilot project will investigate how data science could be used to help optimize pump development. In a first step, you will attempt to predict the required cooling capacity based on the temperature values of the machine. You have been provided with a database containing machine data for this purpose: *hydro.db*.

This is a regression problem because we want to predict a continuous number. Before we consider which model to use, let's take a closer look at the data.

First you have to import the data from the database. To do this, import `sqlalchemy` with its conventional alias `sa`. *hydro.db* is an SQLite database located in your current working directory. Create an interface with `sa.create_engine()` and store the result in the `engine` variable.

```
In [1]: import sqlalchemy as sa
engine = sa.create_engine('sqlite:///hydro.db')
connection = engine.connect()
```

Now create an `Inspector` and store it in the variable `inspector`. Use its `my_inspector.get_table_names()` method to print the names of the tables in *hydro.db*.

```
In [2]: sa.inspect(engine).get_table_names()
```

```
Out[2]: ['cooling_efficiency',
         'cooling_power',
         'efficiency_sensor',
         'flow_sensor_1',
         'machine_efficiency',
         'temperature_sensor_1',
         'temperature_sensor_2',
         'temperature_sensor_3',
         'temperature_sensor_4',
         'volume_flow_sensor_1',
         'volume_flow_sensor_2']
```

There are 11 tables containing different sensor data. We now want to get a more detailed overview of the temperature sensors. Let's start with the table named `temperature_sensor_1`.

Create a variable named `connection` to define the connection to the database. Then import `pandas` with the alias `pd` and use an SQL query to import all rows and columns from the `temperature_sensor_1` table in `hydro.db`. Save it as a `DataFrame` with the names `df_temp1`. Then print the first 5 rows of `df_temp1`.

```
In [3]: import pandas as pd
df_temp1 = pd.read_sql('SELECT * FROM temperature_sensor_1', con=connection)
print(df_temp1.shape)
df_temp1.head()
```

```
(1493, 61)
```

```
Out[3]:
```

	0	1	2	3	4	5	6	7	8	9	...	51	52
0	46.055	45.949	45.953	45.875	45.801	45.762	45.723	45.723	45.617	45.629	...	46.125	46.125
1	46.457	46.363	46.379	46.301	46.203	46.152	46.152	46.125	46.062	45.980	...	46.551	46.562
2	44.117	44.031	44.031	43.957	43.859	43.871	43.770	43.691	43.613	43.621	...	44.199	44.141
3	45.871	45.793	45.727	45.715	45.621	45.539	45.488	45.445	45.379	45.375	...	45.953	45.871
4	46.047	45.969	45.969	45.883	45.793	45.727	45.734	45.637	45.621	45.543	...	46.055	46.062

5 rows × 61 columns

As you can see, `temperature_sensor_1` contains 1493 rows and 61 columns. The data describes the cyclical activity of a hydraulic pump. In each cycle, a liquid is sucked in and then pushed somewhere else. The duration of one cycle is 60 seconds. Except for the `'cycle_id'` column, each column represents one second. The columns are already in chronological order, i.e. the first column is the first second, the second column is the 2nd second, etc. The values in the table are temperature values in degrees Celsius (°C) for each second. Each row represents the temperature curve during an entire cycle for a single hydraulic pump with the identification number `'cycle_id'`. So these are records of a temperature sensor with 1493 cycles, where one cycle lasts 60 seconds. This kind of data is also called a **time series**, which is a central part of typical business applications for data science. They represent ordered sequences of observations over time.

**Congratulations:** You have successfully imported and understood the data. Now we want to start with an exploratory data analysis, as usual.

## Line charts as the initial contact to outliers

To get a first impression of the data, we'll start off with a line chart. If you use the `my_df.plot()` method, all the data in each column is displayed as a separate line by default. This is not how we want to present the data here. We want to represent each cycle as a separate line. However, each cycle is represented by one row. So for the visualization we have to swap rows and columns. We can achieve this with the `my_df.T` attribute.

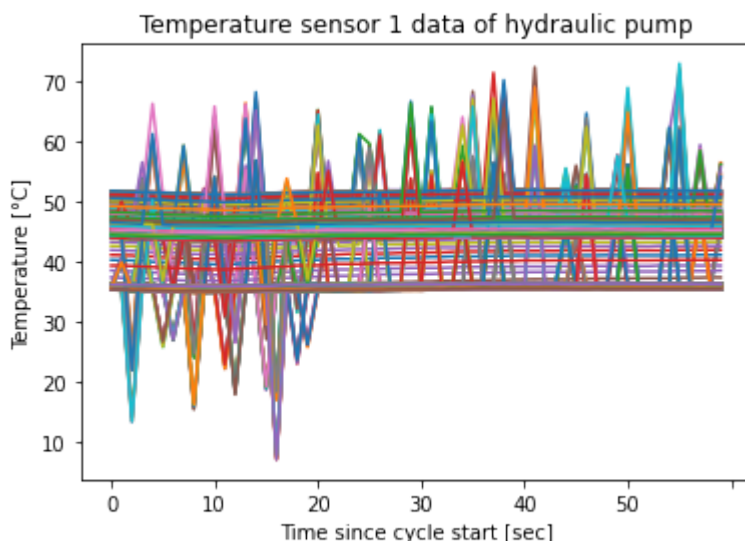
Visualize all the columns (except `'cycle_id'`) with the times on the x-axis, the sensor values on the y-axis and a separate line for each cycle. A legend would be unreadable in this case, so you don't need to show it here.

```
In [4]: import matplotlib.pyplot as plt

# initialise figure and axes
fig, ax = plt.subplots()

# plot line chart
df_temp1.iloc[:, :-1].T.plot(legend=False, ax=ax)

# optimise line chart
ax.set(xlabel='Time since cycle start [sec]',
       ylabel='Temperature [°C]',
       title='Temperature sensor 1 data of hydraulic pump');
#ax.set_xticklabels(range(-10, 71, 10));
```



Except for aesthetic differences, your visualization should look something like this:



You can clearly see that a lot of cycles are around 35°C and also between 45°C and 50°C. However, there are also strange temperature peaks in a cycle that differs greatly from the rest of

the temperature values. These very unusual data points are called **outliers**.

**Remember:** An outlier is a data point that differs greatly from the majority of all other data points.

**Deep Dive:** You should always understand outliers as well as you can, because they directly influence the learning success of your machine learning models. They can even render the predictions of machine learning models completely useless. The reason for this is that a machine learning model looks for a universal structure in the data and outliers create a false idea of this structure. So how you deal with outliers is a key point for every data science project. The origin of the outliers determines how you should handle them. It is very important not to consider outliers as negative, but as a normal property of real data sets. They have their own structure and therefore potentially contain a lot of information about important characteristics of a data set.

To better understand the data, we would advise talking to the hydraulic pump system engineers. It turns out that there are generally four reasons behind outliers:

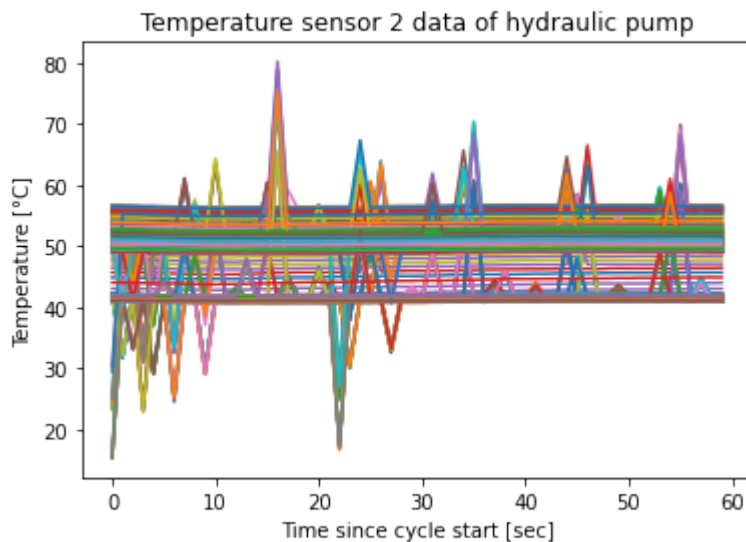
1. Measuring error (the thermometer doesn't always work properly)
2. Data input error (thermometer values aren't always stored correctly)
3. Data corruption (*hydro.db* doesn't work properly)
4. Truthful outliers (the temperature values were actually this unusual)

You should therefore always compare several data sources together to gain a deeper understanding of the outliers. A first working hypothesis would be: The thermometer on sensor 1 is faulty. We can test this hypothesis by creating the same line chart for the second temperature sensor. You will find this data in the database table named *temperature\_sensor\_2*. Import the data into a new `DataFrame` named `df_temp2`. Then draw the line diagram like you did for the first temperature sensor.


```
In [11]: df_temp2 = pd.read_sql('SELECT * FROM temperature_sensor_2', con=connection)

fig, ax = plt.subplots()
# plot line chart
df_temp2.iloc[:, :-1].T.plot(legend=False, ax=ax)

# optimise line chart
ax.set(xlabel='Time since cycle start [sec]',
       ylabel='Temperature [°C]',
       title='Temperature sensor 2 data of hydraulic pump');
ax.set_xticklabels(range(-10, 71, 10));
```



With a little bit of optimization, we end up with this visualization:

 Temperature sensor 2 data

The temperature values are generally higher than from the first temperature sensor. We observe that the second sensor measures higher temperatures on average and that, at least visually, there is not a significantly lower number of outliers. So it is less likely that only a single sensor is damaged and we should doubt our first working hypothesis that there are measurement errors in general.

To be completely sure, we'll visualize the line graphs for all the sensors in the following code cell:

```
In [14]: # initialise figure and axes
fig, axs = plt.subplots(nrows=4,
                        figsize=[6, 12])

for sensor in range(1, 5): # for each thermometer

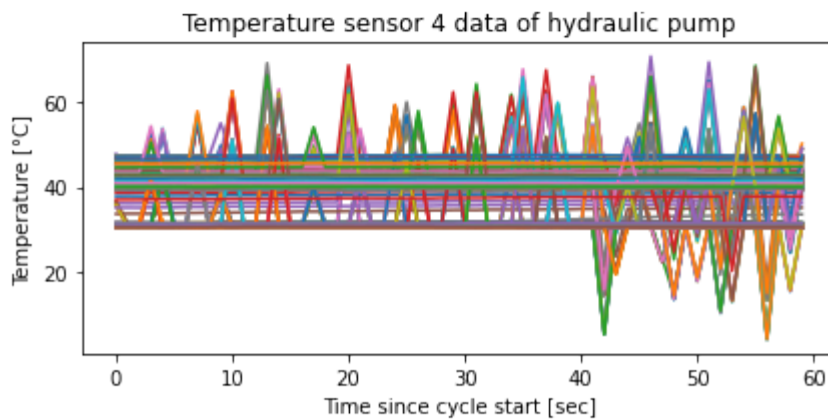
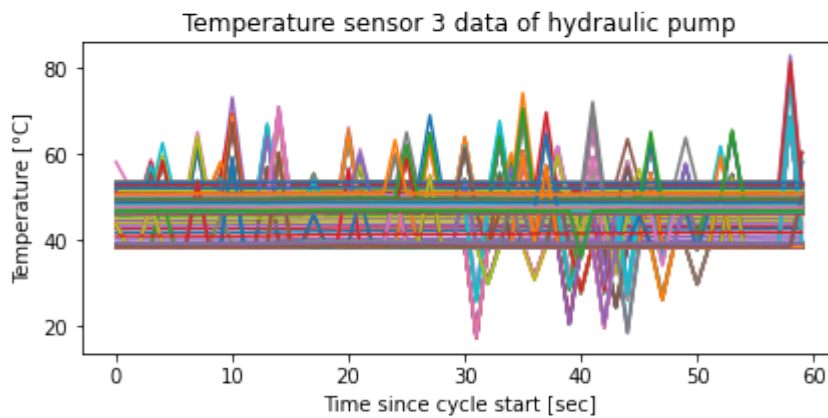
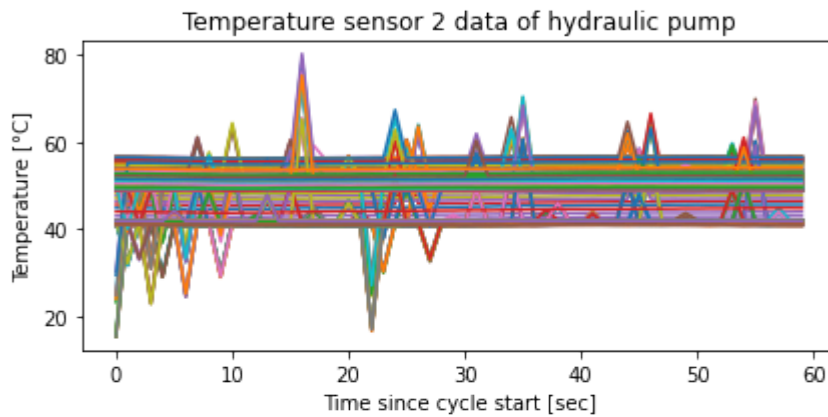
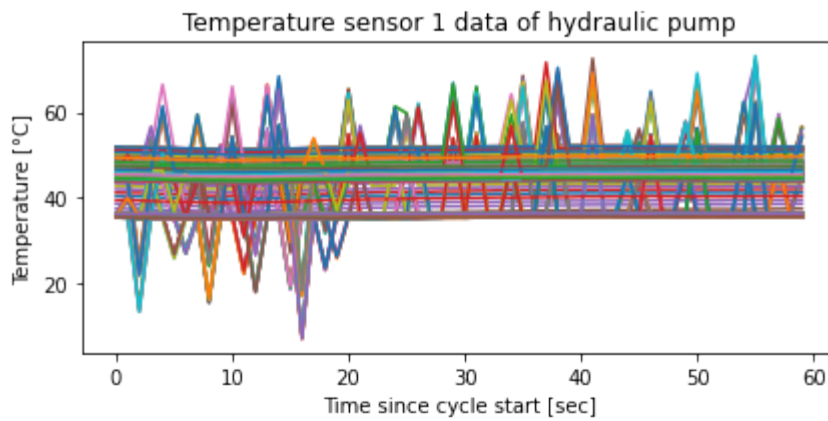
    # read in data from database
    sql_query = '''
    SELECT *
    FROM temperature_sensor_{0}
    '''.format(sensor)
    df = pd.read_sql(sql_query, con=connection)

    # create line chart
    df.iloc[:, :-1].T.plot(legend=False, ax=axs[sensor - 1])

    # optimise line chart
    axs[sensor - 1].set(xlabel='Time since cycle start [sec]',
                       ylabel='Temperature [°C]',
                       title='Temperature sensor {0} data of hydraulic pump'.format(sensor))

    axs[sensor - 1].set_xticklabels(range(-10, 71, 10));

fig.tight_layout() # avoid overlapping axes text
```



You can see very clearly that outliers appear for every thermometer. Since they do not occur at the same time for all sensors, it seems unlikely that this is a truthful outlier. So we can reject our first working hypothesis. In order to formulate a new hypothesis, we should use other exploratory data analysis methods.

**Congratulations.** You've detected outliers in the temperature variations of a hydraulic pump. However, we often use histograms to visualize outliers.

## Visualizing outliers in histograms

You should visualize the existence of outliers in a histogram as often as possible. This provides a better overview of how the data is distributed compared to a line chart. Now let's look directly at an example: Draw a histogram of all temperature data of the 145th cycle (row index `144`) measured by the fourth thermometer.

Tip: Import the data for the 4th sensor from the database and store it in `df_temp4`.

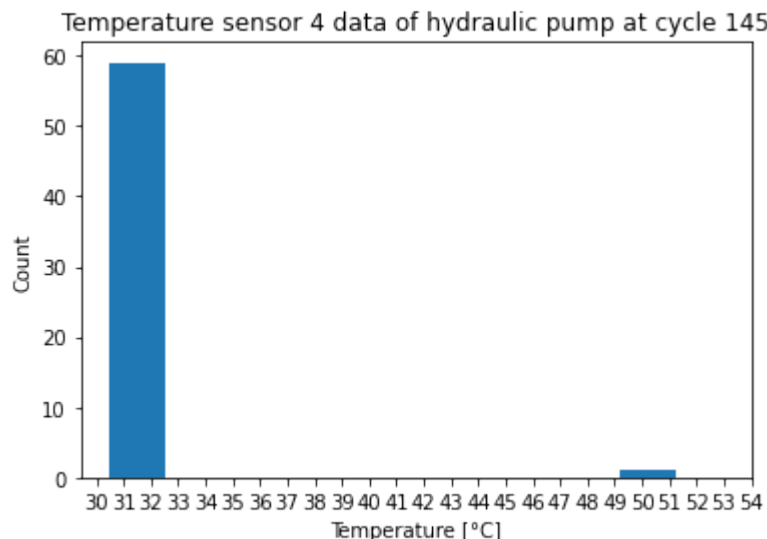
```
In [15]: sql_query = '''
SELECT *
FROM temperature_sensor_4
'''

df_temp4 = pd.read_sql(sql_query, con=connection)

# create histogram for cycle 145
ax = df_temp4.iloc[144, :-1].plot(kind='hist', xticks=range(30,55))

# optimise histogramm
ax.set(xlabel='Temperature [°C]',
      ylabel='Count',
      title='Temperature sensor 4 data of hydraulic pump at cycle 145')
```

```
Out[15]: [Text(0.5, 0, 'Temperature [°C]'),
Text(0, 0.5, 'Count'),
Text(0.5, 1.0, 'Temperature sensor 4 data of hydraulic pump at cycle 145')]
```



You can see that in this cycle almost all temperature values are around 31°C. However, there are a few values around 50°C. There is a discrepancy of about 20°C between the two ranges. The unusually high values therefore have all the characteristics of outliers and we can identify them as such. Here you can see how effective histograms are for finding outliers. You can get a direct feeling for the distance between data points as well as visually estimating the amount of data points in certain areas.

Now, just like the four line charts above, draw the four histograms of the data from the four different temperature sensors in the 145th cycle. You could use a `for` loop again for this, for example.

Tip: Remember that the temperature sensors are numbered starting at 1, whereas Python starts counting at 0 by default.

```
In [16]: fig, axs = plt.subplots(nrows=4,
                                figsize=[6, 12],
                                sharey=True)

    for sensor in range(1, 5): # for each thermometer

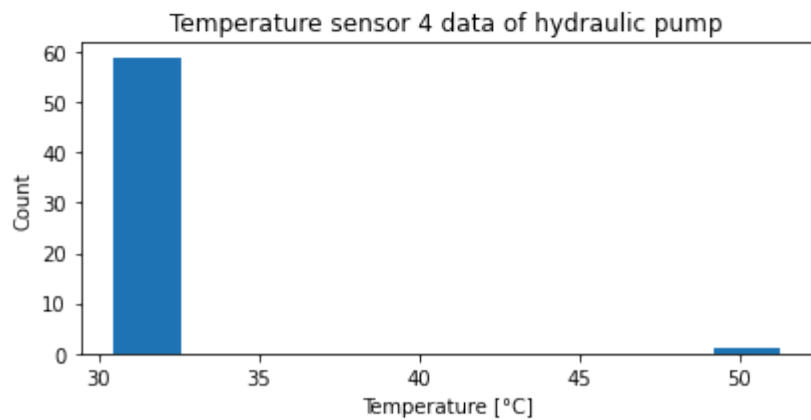
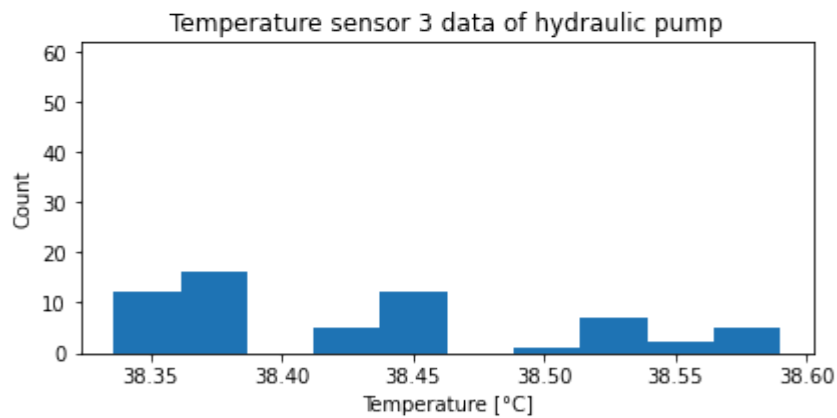
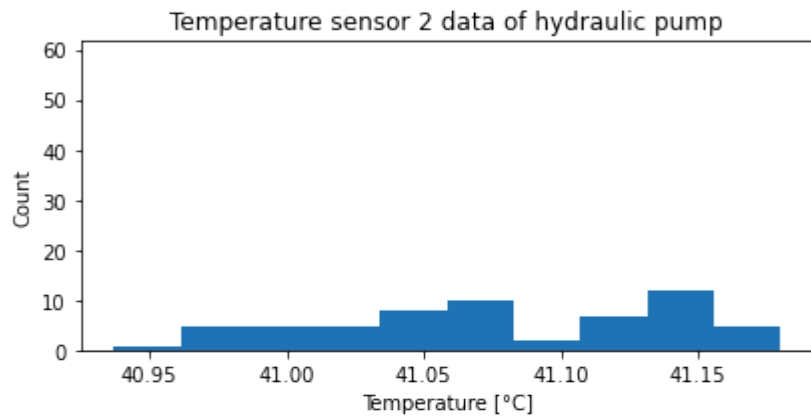
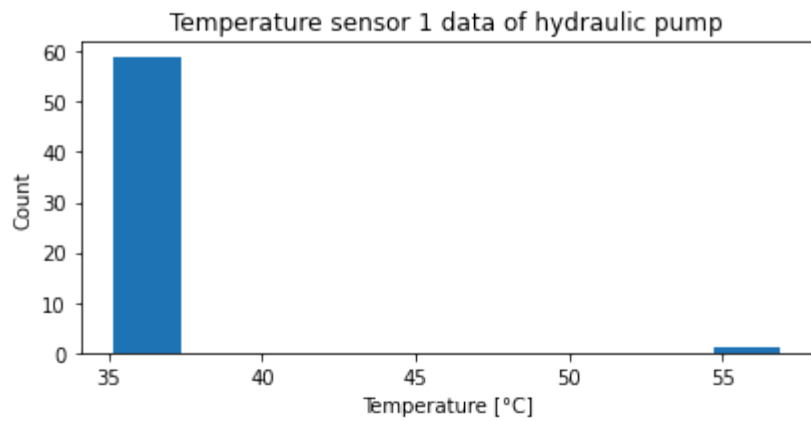
        # read in data from database
        sql_query = '''
        SELECT *
        FROM temperature_sensor_{0}
        '''.format(sensor)
        df = pd.read_sql(sql_query, con=connection)

        # create histogram for cycle 145
        df.iloc[144, :-1].plot(kind='hist',
                                ax=axs[sensor - 1])

        # optimise histogramm
        axs[sensor - 1].set(xlabel='Temperature [°C]',
                            ylabel='Count',
                            title='Temperature sensor {0} data of hydraulic pump'.format(sensor))

    fig.tight_layout() # avoid overlapping axes text
```





In this cycle, there are outliers in the first and fourth temperature sensors. The other two temperature sensor series appear to be widely distributed, at least visually.

Now we have seen several different ways to visualize outliers. In the coming lessons you will learn what to do with them.

For now, all you need to do is close the connection to the database. Close the connection to the engine manually.

```
In [17]: connection.close()
```

**Congratulations:** You got a first impression of the data and the problem this chapter will focus on. Next, we will create several features for each cycle and then use them to make predictions.

**Remember:**

- An outlier is a data point that differs greatly from the majority of all other data points.
- Histograms are particularly well suited for visualizing outliers.
- Outliers can result from faulty data acquisition or storage, but they can also represent valid data points.

---

Do you have any questions about this exercise? Look in the forum to see if they have already been discussed.

---

Found a mistake? Contact Support at [support@stackfuel.com](mailto:support@stackfuel.com).

---

The data was published here first: Nikolai Helwig, Eliseo Pignanelli, Andreas Schütze, 'Condition Monitoring of a Complex Hydraulic System Using Multivariate Statistics', in Proc. I2MTC-2015 - 2015 IEEE International Instrumentation and Measurement Technology Conference, paper PPS1-39, Pisa, Italy, May 11-14, 2015, doi: 10.1109/I2MTC.2015.7151267.