Identifying Outliers in Linear Trends

Module 1 | Chapter 5 | Notebook 6

In this lesson you'll expand on your ability to identify outliers. After we worked with the distance to the mean and median in the previous lesson, in this lesson we'll use the distance to the regression line. By the end of this lesson you will be able to:

- Use RANSAC regression to detect outliers

Outliers according to RANSAC regression

Scenario: You work for a company that manufactures hydraulic pumps. A large number of outliers were found in the data in a pilot project. As a result, management asks you to identify the times when the temperature measurements were unusual. Then the engineers will look into these afterwards.

We'll start by reading the temperature data from the database, see *Robust Feature Engineering*. The four DataFrames `df_temp1`, `df_temp2`, `df_temp3` and `df_temp4` each represent temperatures in degrees Celsius. Each row represents a machine cycle and every column represents a point in time during the cycle.

In [1]:
```
# module import
import sqlalchemy as sa
import pandas as pd

# connection to database
engine = sa.create_engine('sqlite:///hydro.db')
connection = engine.connect()

# read out temperature data of sensor
sql_query = '''
SELECT *
FROM temperature_sensor_{}
'''
df_temp1 = pd.read_sql(sql_query.format(1), con=connection)
df_temp2 = pd.read_sql(sql_query.format(2), con=connection)
df_temp3 = pd.read_sql(sql_query.format(3), con=connection)
df_temp4 = pd.read_sql(sql_query.format(4), con=connection)
df_temp1.head()
```

Out[1]:

	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	46
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	46
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	44
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	45
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	46

5 rows × 61 columns

In the last lesson we saw that:

1. Robust methods are preferable when identifying outliers.
2. Working with median and *median absolute deviation* cannot tell the difference between continuous shifts in values and true outliers.

Therefore, we need an approach to outlier detection, which:

1. is robust against outliers.
2. can tell the difference between continuous shifts in values and outliers.

Fortunately, we learned about this kind of approach in the *Robust Regression* lesson: RANSAC regression.

If you are using the mean value and standard deviation to identify outliers, data points are considered outliers if they are unlikely according to a normal distribution. However, if you are using a regression approach for outlier detection as in this lesson, you are also using a different understanding of outliers. Then outliers are data points which are far away from the regression line. So they are not really predicted by the regression model. You have probably realized that it's worth thinking carefully about what exactly you mean by an outlier before you get down to work identifying them.

Let's look at the cycle `'C_456'` (row index 144) and temperature sensor 4 to make the regression-based approach clearer:

In [2]:
```
df_temp4.iloc[[144], :]
```

Out[2]:

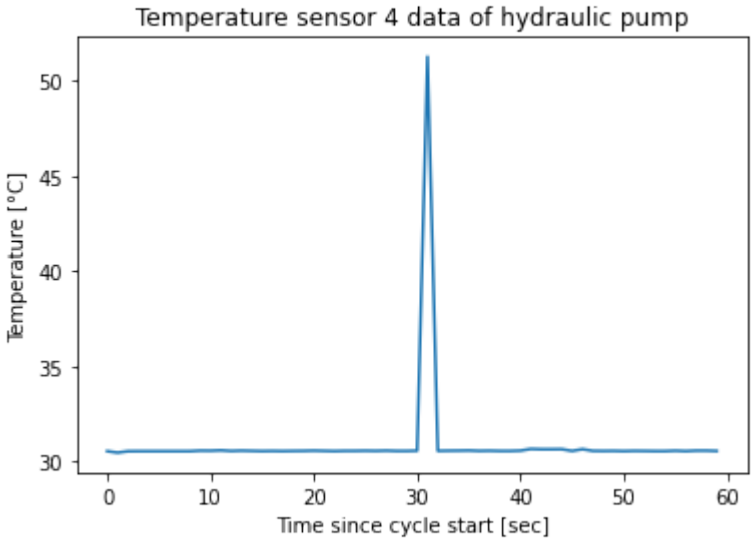
	0	1	2	3	4	5	6	7	8	9	...	51	52
144	30.551	30.473	30.547	30.551	30.551	30.551	30.551	30.551	30.551	30.578	...	30.566	30.562

1 rows × 61 columns

Here's a reminder of how the recorded temperature curve looks:

In [3]:
```
import matplotlib.pyplot as plt
fig, ax = plt.subplots()
df_temp4.iloc[[144], :-1].T.plot(legend=False,
                                 ax=ax)
ax.set(xlabel='Time since cycle start [sec]',
       ylabel='Temperature [°C]',
      title='Temperature sensor 4 data of hydraulic pump')
ax.set_xticklabels(range(-10, 71, 10))
```

Out[3]:
```
[Text(-10.0, 0, '-10'),
 Text(0.0, 0, '0'),
 Text(10.0, 0, '10'),
 Text(20.0, 0, '20'),
 Text(30.0, 0, '30'),
 Text(40.0, 0, '40'),
 Text(50.0, 0, '50'),
 Text(60.0, 0, '60'),
 Text(70.0, 0, '70')]
```

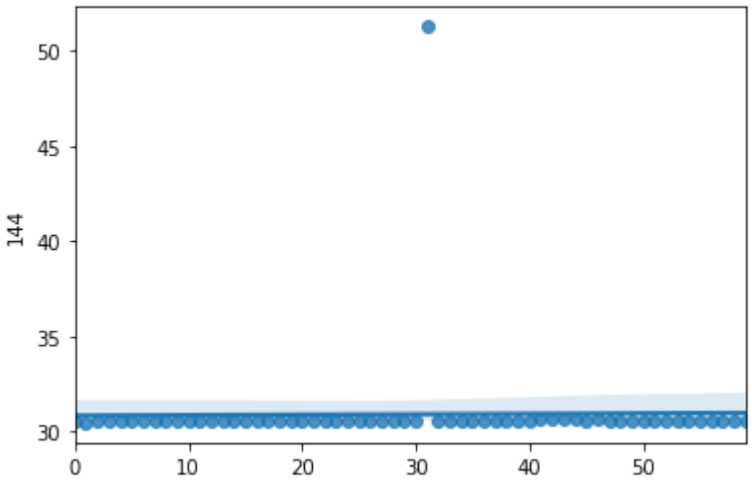


Look at the data's distribution with a scatterplot that shows the time in the cycle in seconds on the x-axis and the temperature on the y-axis. Insert a regression line.

Tip: Use `sns.regplot()` and set the `x` parameter to `x=list(range(60))`. For the `y` parameter, use all the values from the 145th row (row index 144) up to but excluding the last column and convert to `float` (`df_temp4.iloc[144, :-1].astype(float)`).

In [5]:
```
import seaborn as sns
sns.regplot(x=list(range(60)), y=df_temp4.iloc[[144], :-1].astype(float))
```

Out[5]:
```
<AxesSubplot:ylabel='144'>
```



You can see how most points hardly deviate from the regression line. There is only one exception: an outlier at around 51°C. Its residual is extremely large, i.e. there is a strong deviation from the predicted value to the actual observed value and as a result, the data point is far from the regression line.

This should mean that a RANSAC regression will identify it as an outlier. We can try this out right now. Import `RANSACregressor` from `sklearn.linear_model` again.

In [11]:
```
from sklearn.linear_model import RANSACRegressor
```

A data point is considered an outlier by default if it deviates one *median absolute deviation* from the regression line. In our case, this value is perhaps a little low. Because the target values are represented in degrees Celsius, we can decide for ourselves which temperature distance makes a data point an outlier. Let's choose 1°C as the value above which the distance to the regression line is so large that the value is considered an outlier.

Now instantiate the RANSAC regression model defining outliers as being over 1°C from the regression line (`residual_threshold` parameter). Name the model `model_ransac` .

In [12]:
```
model_ransac = RANSACRegressor(residual_threshold=1)
```

Then you can define the feature matrix (`features`) and the target vector (`target`). For the feature matrix, use a `DataFrame` with just one column named whatever you like, containing the values `range(60)` . Make sure that the target vector uses all the columns except for the last one. In this example use the line with line item number `144` .

In [13]:
```
features = pd.DataFrame({'x': range(60)})
target = df_temp4.iloc[144, :-1]
```

Then fit the model to the data.

In [14]:
```
model_ransac.fit(features, target)
```

Out[14]:
```
RANSACRegressor(residual_threshold=1)
```

The model variable `model_ransac` now has a new attribute: `my_model.inlier_mask_`. It shows which data points **do not** count as outliers, what we call *inliers*. Which ones are they in this case? Print the `my_model.inlier_mask_` attribute of `model_ransac`.

In [15]:
```
model_ransac.inlier_mask_
```

Out[15]:
```
array([ True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True, False,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True])
```

Visualized with a scatter plot, it looks like this:

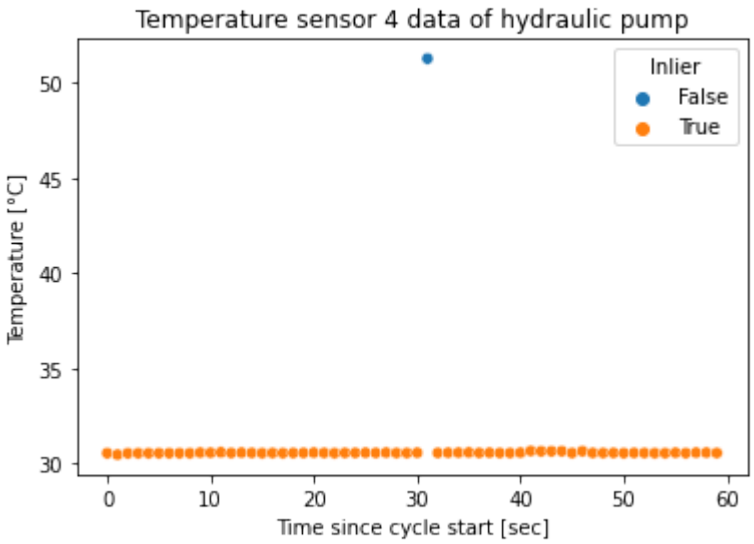
In [16]:
```
# module import (in case it wasn't done before)
import seaborn as sns

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

# draw scatter plot
sns.scatterplot(x=list(range(60)),
                y=df_temp4.iloc[144, :-1],
                hue=model_ransac.inlier_mask_, #coloring inliers (orange) and outliers
                ax=ax)

# optimise plot
ax.set(xlabel='Time since cycle start [sec]',
       ylabel='Temperature [°C]',
      title='Temperature sensor 4 data of hydraulic pump')
ax.legend(title='Inlier')
```

Out[16]:
```
<matplotlib.legend.Legend at 0x7f076deaa160>
```



You can clearly see how the data point already described above differs considerably from all the others and is the only one that is recognized as an outlier (blue).

Congratulations: You have learned how to identify outliers using `RANSAC` regression. This allows you to detect outliers despite a gradual shift in the values. This knowledge is the next step in identifying outliers in our data.

Using RANSAC regression to identify outliers

With a `for` loop, you can now follow this approach for all the cycles in `df_temp4`. Note that in the following code

- a new target vector is generated with each iteration. A new feature matrix is not necessary, as these are always the numbers 0 to 59.
- the Boolean mask in `model_ransac.inlier_mask_` is inverted with `~`. So `True` becomes `False`, and vice versa. This turns the mask into an outlier mask.

In [17]:
```
# copy DataFrame for structure
df_temp4_outliers_mask = df_temp4.copy()

# instantiate RANSAC regression model with adjusted outlier threshold
model_ransac = RANSACRegressor(residual_threshold=1)

for cycle in range(len(df_temp4)):  # for each cycle

    # target vector for this cycle
    target = df_temp4.iloc[cycle, :-1]

    # fit model with data of this cycle
    model_ransac.fit(features, target)

    # save outlier mask of this cycle
    df_temp4_outliers_mask.iloc[cycle, :-1] = ~model_ransac.inlier_mask_

df_temp4_outliers_mask.head()
```

Out[17]:

	0	1	2	3	4	5	6	7	8	9	...	51	52	53	54	55	
0	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	F
1	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	F
2	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	F
3	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	F
4	False	False	False	False	False	False	False	False	False	False	...	True	False	False	False	False	F

5 rows × 61 columns

Now define a function called `outlierdection`, which we will use to use RANSAC regression to mask the outliers for all cycles. We can then apply this function to each temperature sensor individually, so that for each temperature sensor we generate a `DataFrame` displaying only the cycles containing outliers.

You can use the previous code cell as a guide. Note in the following code, we want to create a function with `def function_name(parameter_name):`. The code that should be part of the the function must be indented.

1. In the function you should first create a new `DataFrame` based on the `DataFrame` of a temperature sensor, so that you adopt the same format (indices, column and row labels). The best way to do this is to copy the `DataFrame` using `my_df.copy()`. However, since we only want to work with the `True` / `False` values in the `DataFrame`, there is a problem with the `'cycle_id'` column for the machine cycles, because the values contained in the cells are not Boolean coded. Because we need the outliers for each cycle, the `'cycle_id'` column is well suited as the index for the new `DataFrame`. You can do this with `my_df.set_index()`. Name the new `DataFrame` `temp_values`. Then create a copy of the `temp_values` `DataFrame`, and call it `df_outliers_mask`.
2. In the next step, we'll instantiate a RANSAC regression with the parameter `residual_threshold=1`. It specifies the maximal residual (difference between the predicted value from the actual observed value) for a data sample to be classified as an *inlier* (non-outlier). In our case this means that all values that are more than 1°C away from the regression line are classified as outliers.
3. Now we'll write a loop similar to the previous one, but this time within the function and therefore quite general, so that it should later be applicable to every `DataFrame` of a temperature sensor. You can use the previous code cell as a guide. The largest difference is that in step 1, we specified the index of the new `DataFrame` as the `'cycle_id'` column. Remember this when selecting your columns, that the last column in the original `DataFrame` is now the index in the new `DataFrame`, i.e. our `DataFrame` now only contains the temperature sensor data.
4. We'll filter out just the value that contain outliers in four steps. We find the outliers with `temp_values[df_outliers_mask]`. Assign the output to a new variable named `outlier_values` This will just give us the columns that contain outliers.
5. In the last step, we want to set all inliers to `NaN` values. We can do this by using `outlier_values[~outlier_values.isna().all(axis=1)]`. We need the tilde (~), because we want to select the inliers and not the outliers to be replaced with `NAN`. `axis=1` specifies that we want to proceed row by row. This way we can make sure we only output cycles that actually contain outliers with our index `'cycle_id'`. Then only the outlier values are represented numerically, all inliers are shown as `NaN`. Finally name the output from this `output`, which you have to return in the next line of the function (`return output`).

This task will be something of a challenge. If you need additional help, take a look at the code cell's *hint* for more detailed tips on how to solve this task.

In [ ]:
```
def outlierdetection(df_temp):
    """Detecting outliers for temperature DataFrames. Only rows with outliers will be
    df_temp = placeholder for read temperature sensor DataFrame"""

    # copy DataFrame for structure
    temp_values = df_temp.copy().set_index('cycle_id')
```

```
    df_outliers_mask = temp_values.copy()

    # instantiate RANSAC regression model with adjusted outlier threshold
    model_ransac = RANSACRegressor(residual_threshold=1)

    for cycle in range(len(df_temp)):  # for each cycle
        # target vector for this cycle
        features = pd.DataFrame({'x': range(60)})
        target = temp_values.iloc[cycle, :]

        # fit model with data of this cycle
        model_ransac.fit(features, target)

        # save outlier mask of this cycle
        df_outliers_mask.iloc[cycle, :] = ~model_ransac.inlier_mask_

    # filter only outlier values
    outlier_values = temp_values[df_outliers_mask]

    # return only rows that contain outliers
    output = outlier_values[~outlier_values.isna().all(axis=1)]
    return output
```

Now apply your function to each individual temperature sensor and save the output per temperature sensor as `my_tempsensor_outliers`. For example, the masked outliers of `df_temp1` should be stored in the variable `df_temp1_outliers`. Proceed in the same way for all the other temperature sensors.

In [ ]:
```
df_temp1_outliers = outlierdetection(df_temp1)
df_temp2_outliers = outlierdetection(df_temp2)
df_temp3_outliers = outlierdetection(df_temp3)
df_temp4_outliers = outlierdetection(df_temp4)
```

Now we have a `DataFrame` for each temperature sensor, which only contains the cycles containing outliers. In addition, only the values representing outliers are displayed numerically. All the other values (i.e. the inliers) are shown as `NaN`.

To enable the engineers to investigate the outliers more closely, we can now write each individual temperature `DataFrame` to an Excel file. For each outlier `DataFrame` per temperature sensor, we could therefore create an Excel file with `pd.to_excel()`, generating 4 separate files. However, we want to write the data from all four temperature sensors into a single Excel file as sheets in a workbook. It's useful here to create several Excel sheets. We'll use `pd.ExcelWriter` for this.

We'll define a small *writer* function using `with pd.ExcelWriter('my_file.xlsx') as my_writer:`. As you can see, this creates an xlsx file. Name the file you want to create `temp_outliers.xlsx` and the writer function `writer`.

The Excel file has now been created, but it's still empty. Now we have to create Excel sheets which only show the outliers for each temperature sensor. One sheet should represent one temperature sensor at a time. So we need 4 sheets in the Excel workbook. We can create the first sheet for temperature sensor 1 using `my_df_outliers.to_excel(my_writer,`

`sheet_name='my_sheet_name')`. The name of the sheet for the first temperature sensor should be `df_temp1_outliers`. Do this in the same way for all the other temperature sensors.

In [ ]:
```
with pd.ExcelWriter('temp_outliers.xlsx') as writer:
    df_temp1_outliers.to_excel(writer, sheet_name='df_temp1_outliers')
    df_temp2_outliers.to_excel(writer, sheet_name='df_temp2_outliers')
    df_temp3_outliers.to_excel(writer, sheet_name='df_temp3_outliers')
    df_temp4_outliers.to_excel(writer, sheet_name='df_temp4_outliers')
```

Now you've created an Excel workbook containing four sheets. This was saved in the current working directory on our server. It should now be listed in your home directory (in the DataLab click on the folder symbol in the upper left corner). However, you cannot open this file because it is not UTF-8 encoded, which cannot be opened live in the DataLab.

But please take a look at the following graphic: It should give you an insight into the xlsx file we just created. Each sheet represents a temperature sensor from the data set. Excel cannot display `NaN` values, so all *inlier* cells stored with `NaN` have been emptied. So, the workbook only contains machine cycles with outliers (at least 1°C deviation from the regression line) and when exactly these occurred. We could now forward the xlsx file directly to the engineers for further research.

The three methods for identifying outliers that we learned in this chapter can also be applied to multidimensional data. So far, we have only applied them to each data series individually. However, it's possible that an outlier could only become clear when you combine the information of multiple data series.

The regression-based approach with RANSAC regression can also be extended to several dimensions. Then the distance to a regression line is no longer evaluated to detect outliers, but the distance to a regression plane.

We have also seen that DBSCAN can also be used to identify outliers in multidimensional space, see *Clustering with DBSCAN (Chapter 3)*. In this case, a data point would be considered an outlier if it is relatively isolated and too far away from other data points.

Otherwise, `sklearn` offers you other approaches for detecting outliers, see the scikit-learn page on the subject.

Now you might get the feeling that there are so many ways to detect outliers that in the end you don't even know what to do. So remember the most important premise of good outlier detection: first and foremost, you should think about what you actually mean by an outlier - i.e., what does it mean when a data point is "too unusual"? In this chapter you have seen two different approaches:

- Outliers are unlikely data points (*median absolute deviation* distance from the median)
- Outliers are data points that deviate significantly from the model (*RANSAC regression*)

You should therefore always consider what exactly constitutes outliers in your company's data.

Now you're done, close the connection to the database.

In [ ]:

Congratulations: You've identified outliers in the temperature data. You used a regression model to do that. Now the engineers can investigate what was going on at those times. Your company thanks you very much for your detailed analysis of the data.

Remember:

- Outliers are `false` in `model_ransac.inlier_mask_`.
- You can create multiple sheets in an Excel workbook with `pd.ExcelWriter()` and `pd.to_excel()`.

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.

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.