Automatically Detecting Room Occupancy

Module 1 | Chapter 2 | Notebook 2

In this notebook we will explore the data that we will use in this chapter for k-nearest neighbors classification. Like all classification algorithms, k-nearest-neighbors is used to predict categorical values, which we call classes. This could be nationality, machine failure or customer churn. In this case, it's whether or not people are in a room. At the end of of this lesson you will have a good understanding of the data for this chapter.

Importing and cleaning the data

Scenario: You work for a *smart-building* provider. You sell sensor technology and data analysis for buildings, to make it possible to automatically identify where objects are located, what temperature settings are needed and whether the alarm should be turned on.

In a new pilot project, sensors already installed to measure temperature, light, humidity and carbon dioxide will be used to predict whether a person is in the room. If this is not the case for a longer period of time, the *smart building* could automatically switch off the lights or heating. This can reduce costs.

In February 2015, data was recorded in one of the company's offices. It is a two-person office, which was equipped with the relevant sensors. A security camera was used to determine when someone was in the room. The resulting data is stored in *occupancy_training.txt*.

To look at the training data, first import `pandas` with its conventional alias `pd`. Then import the data and store it in `df_train`. Despite the file extension *.txt*, this is actually a CSV file. Finally, print the first five rows (all columns) of the data to get a first impression.

In [1]:
```
import pandas as pd
df_train = pd.read_csv('occupancy_training.txt')
df_train.head()
```

Out[1]:

	date	Temperature	Humidity	Light	CO2	HumidityRatio	Occupancy
1	2015-02-04 17:51:00	23.18	27.2720	426.0	721.25	0.004793	1
2	2015-02-04 17:51:59	23.15	27.2675	429.5	714.00	0.004783	1
3	2015-02-04 17:53:00	23.15	27.2450	426.0	713.50	0.004779	1
4	2015-02-04 17:54:00	23.15	27.2000	426.0	708.25	0.004772	1
5	2015-02-04 17:55:00	23.10	27.2000	426.0	704.50	0.004757	1

Based on the entries in the `'date'` column, we can see that one measurement was taken each minute, the first measurement being made on 4th February 2015 at 17:51.

The data dictionary for this data is as follows:

Column number	Column name	Type	Description
0	`'date'`	continuous (`datetime`)	time of the date measurement
1	`'Temperature'`	continuous (`float`)	temperature in ° celsius
2	`'Humidity'`	continuous (`float`)	relative humidity in %
3	`'Light'`	continuous (`float`)	Brightness in lux
4	`'CO2'`	continuous (`float`)	carbon dioxide content in the air in parts per million
5	`'HumidityRatio'`	continuous (`float`)	Specific humidity (mass of water in the air) in kilograms of water per kilogram of dry air
6	`'Occupancy'`	categorical	presence (0 = no one in the room, 1 = at least one person in the room)

Now check that the data types in the columns of `df_train` match those in the data dictionary? Change them if they don't.

In [2]:
```
df_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 8143 entries, 1 to 8143
Data columns (total 7 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   date           8143 non-null   object 
 1   Temperature    8143 non-null   float64
 2   Humidity       8143 non-null   float64
 3   Light          8143 non-null   float64
 4   CO2            8143 non-null   float64
 5   HumidityRatio  8143 non-null   float64
 6   Occupancy      8143 non-null   int64  
dtypes: float64(5), int64(1), object(1)
memory usage: 508.9+ KB
```

As you can see, there are two discrepancies:

- `date` is not a `datetime`
- `Occupancy` is not categorical

To convert a column to the `datetime` type. You need the `pd.to_datetime()` function. You can pass it a `Series` with date entries and you will get a series with `datetime` objects.

Numeric columns are generally interpreted by sklearn as continuous data. Often, however, the data is actually categories coded with numbers; for example, the room size could be coded as follows:
small = 1 , medium = 2, large = 3. In this case, the algorithm would interpret the data as if a large room was exactly three times as large as a small room, although we have no information about the numerical proportions for the different sizes.

So you always have to take care to convert categorical variables to strings or categories. The `my_Series.astype('category')` method helps you with this. It transforms a continuous feature into a categorical one.

Try it out now.

In [3]:
```
# turn date into DateTime
df_train.loc[:,'date'] = pd.to_datetime( df_train.loc[:,'date'])
# turn Occupancy into category
df_train.loc[:,'Occupancy'] =   df_train.loc[:,'Occupancy'].astype('category')
```

Next, it's always a good idea to look at the values directly. Print the eight-value summary of the numerical columns. Do all values make sense?

In [4]:
```
df_train.describe()
```

Out[4]:

	Temperature	Humidity	Light	CO2	HumidityRatio
count	8143.000000	8143.000000	8143.000000	8143.000000	8143.000000
mean	20.619084	25.731507	119.519375	606.546243	0.003863
std	1.016916	5.531211	194.755805	314.320877	0.000852
min	19.000000	16.745000	0.000000	412.750000	0.002674
25%	19.700000	20.200000	0.000000	439.000000	0.003078
50%	20.390000	26.222500	0.000000	453.500000	0.003801
75%	21.390000	30.533333	256.375000	638.833333	0.004352
max	23.180000	39.117500	1546.333333	2028.500000	0.006476

All the values appear to be realistic. The `'count'` row also indicates that all numeric columns contain the same number of values.

Now also print all unique values of the only categorical column (`'Occupancy'`) and its absolute distribution. Use the `my_Series.value_counts()` method for this. Do the categories match those of the data dictionary?

In [8]:
```
df_train.loc[:,'Occupancy'].value_counts()
```

Out[8]:
```
0    6414
1    1729
Name: Occupancy, dtype: int64
```

There are only two categories in the column: `1` (people in the room) and `0` (nobody in the room). This was explained in the data dictionary.

Congratulations: You imported the data and cleaned it a little. In your work as a data scientist, this part can take a lot of time. So prepare yourself in general to spend a lot of time working on this. But we are lucky and can turn to the features right away.

Feature Engineering - adding an additional feature

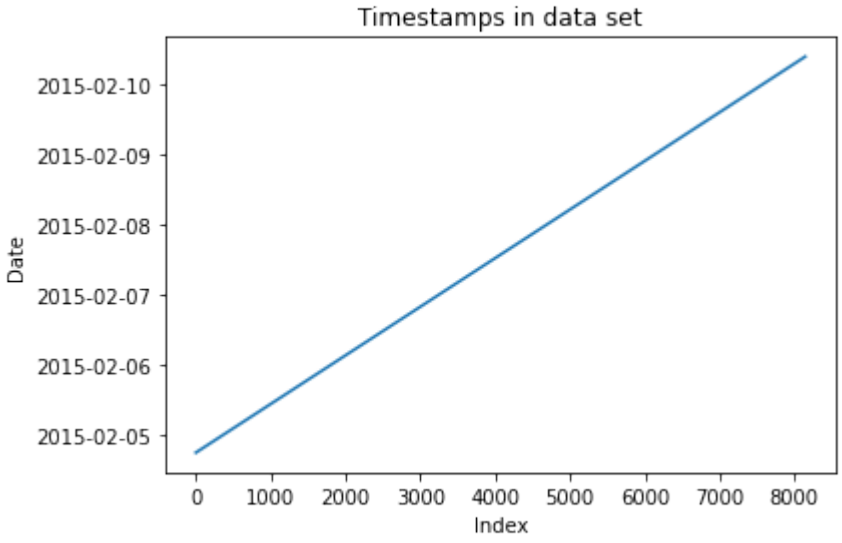
A machine learning algorithm generally only knows the data that you give it. Even if it contains information that may seem obvious to you, it isn't necessarily obvious for the algorithm.

The `date` column is one of these cases. At the moment, the algorithm only has continuous values without breaking it up into hours or days. The following diagram illustrates the view at the time:

In [10]:
```
import matplotlib.pyplot as plt  # module import for plotting

fig, ax = plt.subplots()  # initialise Figure and Axes
df_train.loc[:, 'date'].plot(ax=ax)  # fill Axes with line chart
ax.set(xlabel='Index',  # change x-label
       ylabel='Date',
       title='Timestamps in data set')  # change y-label
```

Out[10]:
```
1      2015-02-04 17:51:00
2      2015-02-04 17:51:59
3      2015-02-04 17:53:00
4      2015-02-04 17:54:00
5      2015-02-04 17:55:00
               ...
8139   2015-02-10 09:29:00
8140   2015-02-10 09:29:59
8141   2015-02-10 09:30:59
8142   2015-02-10 09:32:00
8143   2015-02-10 09:33:00
Name: date, Length: 8143, dtype: datetime64[ns]
```



It would certainly help the machine learning algorithm to know at what time of day the data points were measured. After all, an office is not consistently busy at all times of the day. The best thing to do is to create a new column called `'msm'`, which shows the minutes since midnight. So the algorithm knows where we are in a day. How do we do that?

A `Series` provides access to `datetime` functionality using a `my_Series.dt` attribute. After the attribute you can write the unit you want to extract from the `datetime`. For example, you get the number of seconds for the entries in a `Series`.

In [11]:
```
print('datetime: \n', df_train.iloc[0:3, 0], '\n')     # standard datetime
print('seconds: \n', df_train.iloc[0:3, 0].dt.second)  # seconds
```

```
datetime:
 1   2015-02-04 17:51:00
2   2015-02-04 17:51:59
3   2015-02-04 17:53:00
Name: date, dtype: datetime64[ns]

seconds:
 1     0
2    59
3     0
Name: date, dtype: int64
```

Apparently these are the seconds that have passed since the last full minute. The hours, days, weeks etc. can be accessed in a similar way. You can find the associated attributes here in the official `pandas` documentation:

- year
- month
- day
- hour
- minute
- second
- microsecond
- nanosecond

Now also print the hours since midnight for the first three entries of the `'date'` column.

In [12]:
```
print('datetime: \n', df_train.iloc[0:3, 0], '\n')     # standard datetime
print('seconds: \n', df_train.iloc[0:3, 0].dt.second)  # seconds
print('seconds: \n', df_train.iloc[0:3, 0].dt.hour)  # hours
```

```
datetime:
 1   2015-02-04 17:51:00
2   2015-02-04 17:51:59
3   2015-02-04 17:53:00
Name: date, dtype: datetime64[ns]

seconds:
 1     0
2    59
3     0
Name: date, dtype: int64
seconds:
 1    17
2    17
3    17
Name: date, dtype: int64
```

Now you know everything you need to know to create the new `'msm'` column. It should store the hours of the day multiplied by 60 and summed with the minutes since the last full hour. This corresponds to the minutes since midnight and tells the machine learning algorithm which time of day a data point was measured at.

\begin{equation*} minutes\ since\ midnight = (hours\ since\ midnight * 60) + minutes\ since\ full\ hour \end{equation*}

In [13]:
```
df_train.loc[:, 'msm'] = (df_train.loc[:, 'date'].dt.hour * 60) + df_train.loc[:, 'dat
```

Now visualize the `'msm'` values as a line chart. Can you clearly see where a new day begins?

In [14]:
```
df_train.loc[:, 'msm'].plot()
```

Out[14]:
```
1       1071
2       1071
3       1073
4       1074
5       1075
        ...
8139     569
8140     569
8141     570
8142     572
8143     573
Name: msm, Length: 8143, dtype: int64
```

The graph should now look something like this:

 msm column

You can clearly see that measurements were taken for about a week. The six peaks correspond to 23:39 six times. Afterwards the line immediately falls to zero, because with the new day at 00:00, not a single minute has passed since midnight.

Many algorithms find it difficult to use the `datetime` data type directly. You often have to convert it into a number. Which data types does the `'msm'` column contain?

In [19]:
```
df_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 8143 entries, 1 to 8143
Data columns (total 8 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   date           8143 non-null   datetime64[ns]
 1   Temperature    8143 non-null   float64
 2   Humidity       8143 non-null   float64
 3   Light          8143 non-null   float64
 4   CO2            8143 non-null   float64
 5   HumidityRatio  8143 non-null   float64
 6   Occupancy      8143 non-null   category
 7   msm            8143 non-null   int64
dtypes: category(1), datetime64[ns](1), float64(5), int64(1)
memory usage: 517.0 KB
```

Out[19]:

	date	msm
1	2015-02-04 17:51:00	1071
2	2015-02-04 17:51:59	1071
3	2015-02-04 17:53:00	1073
4	2015-02-04 17:54:00	1074
5	2015-02-04 17:55:00	1075
...	...	...
8139	2015-02-10 09:29:00	569
8140	2015-02-10 09:29:59	569
8141	2015-02-10 09:30:59	570
8142	2015-02-10 09:32:00	572
8143	2015-02-10 09:33:00	573

8143 rows × 2 columns

Congratulations: You have created an additional feature which has two advantages compared to `'date'`:

1) It contains important information (time of day), which otherwise would not have been used

2) It represents this information as `int`, which causes fewer problems than a `datetime`.

It is always a good idea to visualize the data as well. Now we will generate a visualization to summarize the entire data set.

Data visualization

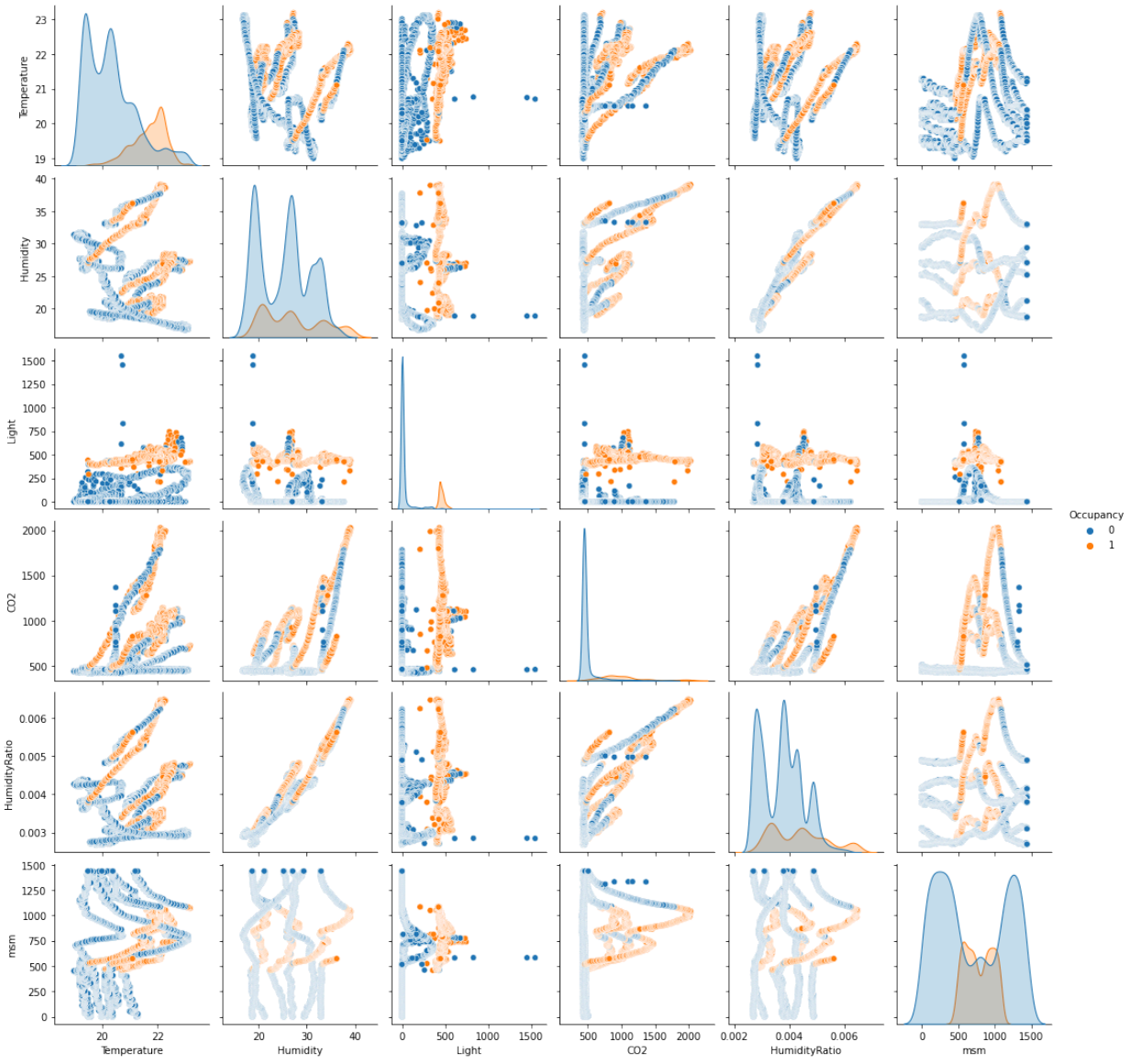
In the lesson *EDA - Understanding Data (Module 1 Chapter 1)* you already used the `seaborn` function `seaborn.pairplot()` to visualize all numerical columns and how they are related.

This function also takes an additional argument `hue`, which differentiates between the categories in `'Occupancy'` using color. So run `seaborn.pairplot()` with `df_train` and set `hue` to `'Occupancy'`. What relationships can you see in the data, even on a purely visual level?

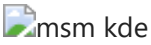
Tip: Also use the `vars` parameter. This specifies which columns are to be displayed. It is important that the `'Occupancy'` column is not displayed, otherwise an error occurs. So pass the columns `['Temperature', 'Humidity', 'Light', 'CO2', 'HumidityRatio', 'msm']` to the `vars` parameter.

In [22]:
```
import seaborn as sns
sns.pairplot(df_train,
             hue='Occupancy',
             vars=['Temperature', 'Humidity', 'Light', 'CO2', 'HumidityRatio', 'msm'],
             # diag_kws={'common_norm': False} # optional, maybe easier to interpret
             #diag_kind='hist'
            )
```

Out[22]:
```
<seaborn.axisgrid.PairGrid at 0x7fa51a8171c0>
```



There is a lot in the *Figure* now. Let's pick out one aspect that is particularly striking. At the bottom right you can see the density histogram for `'msm'`. Your plot should look something like this:


msm kde

You can read a density histogram in the same way as a normal histogram, except that the y-axis no longer shows the absolute frequency but the density (see *Continuous Distributions: Measuring Values (module 0, chapter 3)*). From the orange line (someone present in the room) you can see that in the middle of the day there is usually someone in the room. The small dip in the middle could possibly represent people's lunch break. This shows very clearly that it was worthwhile calculating `'msm'`. This column contains useful information.

Congratulations: You have successfully imported the data, calculated a new column, and then visualized everything. Now that you have become familiar with the data, we can use it in the next lesson to classify whether there are people in a room or not.

Remember:

- Before you start using a machine learning approach, it's important to explore and clean the data.
- You can access the `datetime` functionality within a `Series` using `my_series.dt`

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: Accurate occupancy detection of an office room from light, temperature, humidity and CO2 measurements using statistical learning models. Luis M. Candanedo, Véronique Feldheim. Energy and Buildings. Volume 112, 15 January 2016, Pages 28-39.