Load monitoring and control

Data Visualization

SESSION 2: Offline data visualization using Python

Matplotlib and Plotly

December 3, 2020

Table of Contents

Introduction

Until now, we have worked with Arduino, Raspberry Pi, APIs, python, with a common objective: Retrieve data for a specific objective. Using data, we can observe how is the behavior of the load we are controlling and we can schedule its operation according to some specific signals, such as light intensity, market prices, time, etc.

A useful way to see that the load is behaving in the way we want is by using data visualization tools. Data visualization has become a key activity in companies to extract conclusions and define the next steps of the company.

In this session, we will use a wind turbine dataset, using data shared by DTU at the following DOIs: 10.11583/DTU.7856891 and 10.11583/DTU.7856888. We will have historical observations of the V52 Wind turbine. We will work with historical observations, using a CSV file.

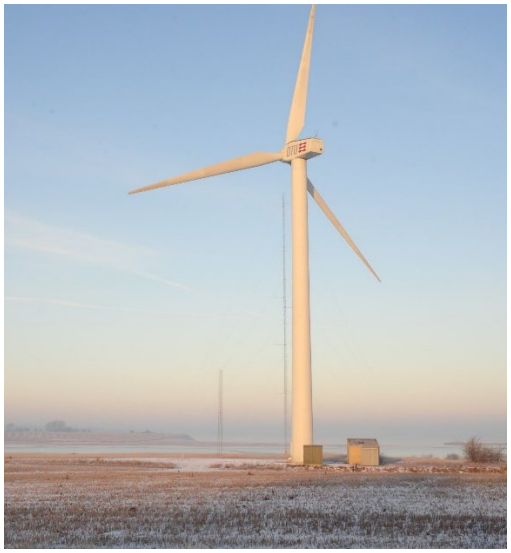


Figure 1: V52 Wind Turbine. Extracted from DTU Course on Data Analytics.

We will use different Python libraries to visualize this data. Specifically, the libraries used in this lab session will be *matplotlib, plotly, and Dash.*

1. Before starting…

1.1. Deliverables for this session

Along with this document, you will find some questions that you will need to answer and submit at the end of the session. You can submit it by uploading a PDF file with all the answers and plots, or you can directly share the *jupyter notebook*.

For this session, we recommend you to use *Jupyter Notebook* or *Spyder*.

1.2. Github Repository

All the material required for this lab session can be found here:

https://github.com/wobniarin/CAPUEE_2019_LAB5_Data_Visualization

1.3. Libraries required

Importing Libraries

```
In [3]: # data processing
        import pandas as pd
        # numerical library
        import numpy as np
        # timer, dates
        import datetime
        # data visualization libraries
        import matplotlib.pyplot as plt
```

1.4. Loading data

There are several ways in which data can be loaded into your script for processing. Data can be stored in such different formats: .xlsx, .csv.

In the previous sessions, we have worked on how to retrieve data using an API. Later, we worked on storing our own data from the sensors that have been capturing data.

Today, we will work on importing the data using a CSV file. Open the file named "V52_ExtensiveData.csv" and take a look at it:

Questions:

1. How many rows do we have?
2. What do these data tell us?

Once we have taken a look at the data, we can import this CSV file into our python script. We will need a specific call for this.

Loading Data

```
In [23]: df = pd.read_csv('./data/V52_ExtensiveData.csv', sep='\t', skiprows=12)
```

We can easily see the structure of the dataset if we take a look at the head of the python script:

First look at the dataset

```
In [24]: df.head()
```

Out[24]:

	Date	Wsp_44m	Wdir_41m	ActPow	RePow	ActPow_std	Wsp_44m_std	Wdir_41m_std	stability
0	201801010000	4.71803	200.743	64.6673	0.000860	33.4251	0.566131	6.45730	1.0
1	201801010010	5.44100	201.768	70.8152	-0.000657	26.3829	0.765691	6.66940	1.0
2	201801010020	5.32178	197.962	80.8037	-0.000617	30.2002	0.603442	6.99113	1.0
3	201801010030	5.95325	204.606	86.1123	-0.002370	43.1192	0.872915	5.47062	1.0
4	201801010040	6.17765	204.398	110.8570	0.001033	29.9507	0.550160	4.93713	1.0

You can also take a look at the dimensions of the Dataframe you just loaded, using the shape attribute.

```
df.shape
```

Questions:

3. What's the shape of the Dataframe?
4. Which type does the Dataframe have?
5. Are all the columns in the right format?
6. Do we have to change any of the types?
7. Are the columns' name useful for us? What's the information they are trying to tell us?
8. Do you know the units of measurement?
9. For how long have we been taking data?
10. What's the time granularity?

When we import data from a CSV file, most of the times we should change some of the columns according to the data type we want.

Questions:

11. Write the commands you have written to change the features' types, if required.

We can take a look at the data by performing and Exploratory Data Analysis (EDA), by using the following commands

```
df.info()
```

```
df.describe()
```

Questions:

12. What are your conclusions after implementing the EDA?
13. Do we have missing values? What can we do with them?

2. Creating plots using Python: matplotlib

2.1. Basic plot

In the data we have loaded, we want to access the active power output from the wind turbine. We can easily select the column we want from the Dataframe by writing the column name in []

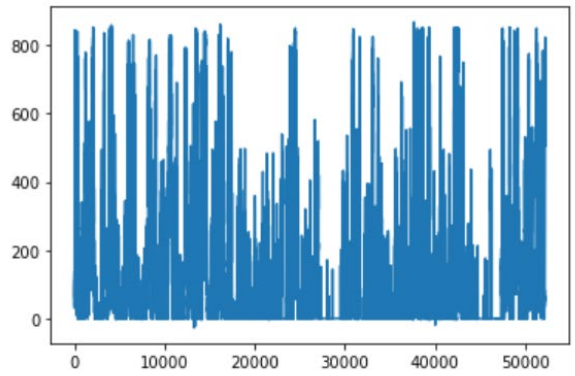
```
df['ActPow']
```

The easiest way to plot in Python is by using matplotlib. We have imported matplotlib before, calling it *plt.* We can access all the attributes of matplotlib (types of graphs, features, etc.), by writing:

plt.

Here we will plot the power output by writing the following command:

```
In [36]: plt.plot(df['ActPow'])
         plt.show()
```



The problem here is that we cannot see the date on which that power output occurred. This can be solved by adding the ['Date'] column.

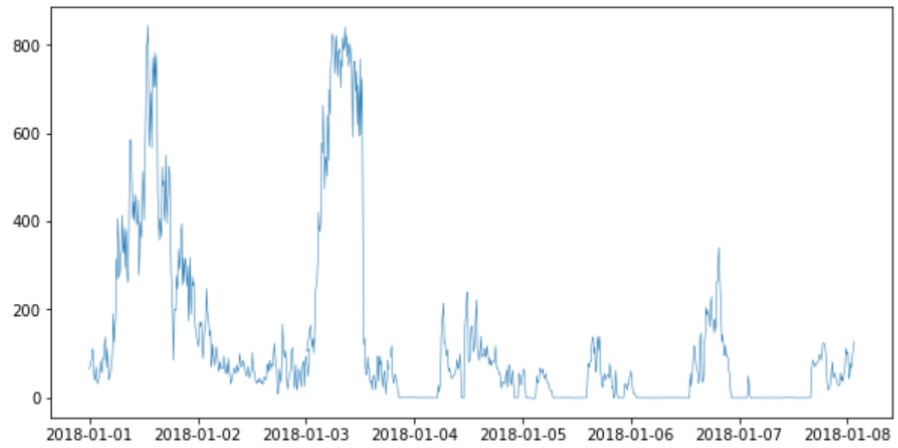
Questions:

14. What's the command you should write to plot the Active Power versus Date?

However, showing the entire yearly consumption does not look very good, but it is possible to select only a subset of data for clearer visualization. Here, picking a week of data to visualize.

```
In [49]: plt.figure(figsize=(10,5))
         plt.plot(df['Date'][0:6*24*7], df['ActPow'][0:6*24*7], linewidth=0.5)
         plt.show
```

```
Out[49]: <function matplotlib.pyplot.show(*args, **kw)>
```



2.2. Adding aesthetics to our plots

2.2.1. Figure Size

At the beginning of the figure we should include the following command:

```
plt.figure(figsize=(20,10))
```

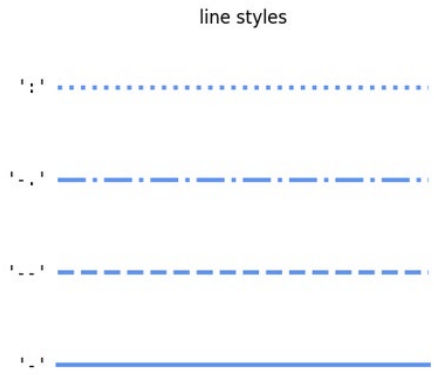
2.2.2. Linewidth

We can change the linewidth by including it on our plot call.

```
plt.figure(figsize=(10,5))
plt.plot(df['Date'][0:6*24*7], df['ActPow'][0:6*24*7], linewidth=0.5)
plt.show()
```

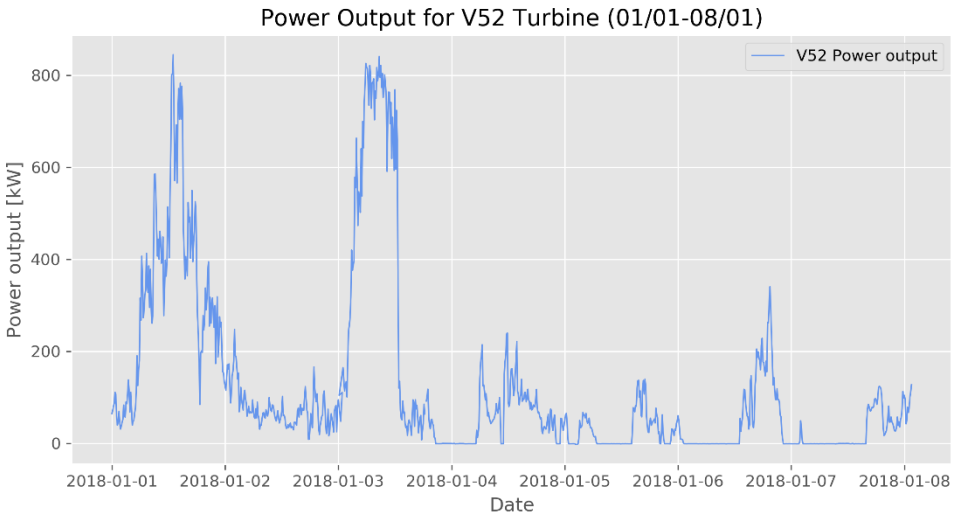
2.2.3. Line-style

By default, the line style will be considered as straight, but there are different options:



2.2.4. Colors

Example:



Questions:

15. Try to create your own plots and submit them with your answers.

3. Interactive data visualization: Plotly

Plotly is a python library that is used to easily create interactive plots.

3.1. Building graphs with Plotly

The native syntax of Plotly includes three main components:

- **Data**
  The data object defines the data that will be displayed in the graph
- **Layout**
  The layout object defines all additional features of the graph like title, axis titles, etc.
- **Figure**
  The figure object brings together the Data and the Layout and creates the graph to be plotted.

You can find full Plotly documentation for Python here.

3.2. Basic interactive plot

This is the full syntax for defining the figure as one object at once:

```
fig = go.Figure(
        data=[
            go.Scatter(x=df_total_year.TIME,
                       y=df_total_year.Value,
                       mode='lines')
        ],
        layout=go.Layout(
            title=dict(text='Total Meat Consumption Evolution'),
            xaxis=dict(title='Year'),
            yaxis=dict(title='Meat consumption, thousand tonnes')
        )
    )
fig.show()
```

Questions:

16. Identify the Data, Layout and Figure objects in the code above.
17. Create an interactive plot for visualizing the active power of V52

3.3. Multiple lines interactive plot

When we want to have an interactive graph with multiple lines, we have to add each line separately. This is a more step by step approach. In this case, we will use the following methods: *add_trace* and *layout.update.*

```
fig = go.Figure()
# create a data trace for each type of meat and add them to figure one by one
fig.add_trace(go.Scatter(x=df_year_type_pivot.TIME,
                         y=df_year_type_pivot.BEEF,
                         mode='lines+markers',
                         name='beef'))
fig.add_trace(go.Scatter(x=df_year_type_pivot.TIME,
                         y=df_year_type_pivot.PIG,
                         mode='lines+markers',
                         name='pig'))
fig.add_trace(go.Scatter(x=df_year_type_pivot.TIME,
                         y=df_year_type_pivot.POULTRY,
                         mode='lines+markers',
                         name='poultry'))
fig.add_trace(go.Scatter(x=df_year_type_pivot.TIME,
                         y=df_year_type_pivot.SHEEP,
                         mode='lines+markers',
                         name='sheep'))
# edit the layout
fig.update_layout(title='Meat Consumption Evolution by Meat Type',
                  xaxis_title='Year',
                  yaxis_title='Meat consumption, thousand tonnes')
fig.show()
```

Questions:

18. Create an interactive plot with the Active Power and the Reactive power of V52 Turbine