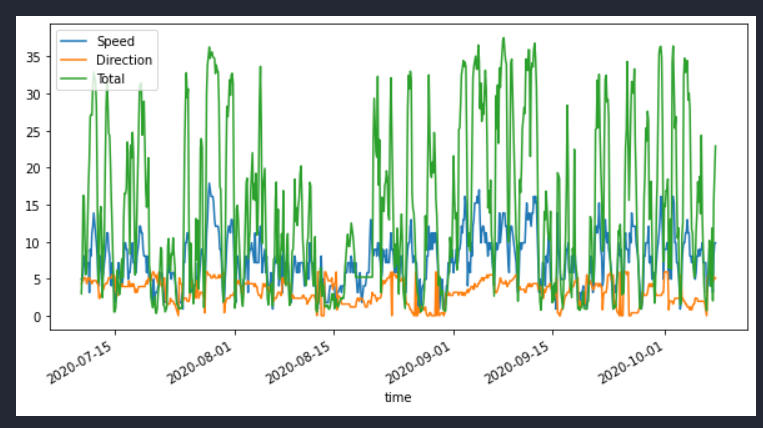


**Assignment 1**: *Data Pipelines*

***Forecasting the wind power production in Orkney***



*Executed by Asger C. Schliemann-Haug  
email:* [*ACSC@ITU.DK*](mailto:ACSC@ITU.DK) *Student No:17565  
Course: Big Data Management*

1. **Introduction**

The aim of this assignment is to design and implement a data pre-processing pipeline for wind power production forecasting using weather forecast data and windpower from in Orkney, Scotland. The town has approximately 20,000 inhabitants and they are able to cover their annual energy consumption with a production of about 120% of the level consumed.

The forecasting system uses sklearn. The wind speed is the primary input parameter to the model, closely related to the power output. Furthermore, the wind direction is an important input-parameter, which increases the dimensionality of the input.

1. **Data type and source**

Data on Orkney’s renewable energy generation is obtained from the Scottish and Souther Electricity Networks (SSEN). Every minute a data sample is taken from a public website. The collected data is then stored in the measurement “Generation” and follows the schema:

1. name : Generation
2. Key Type
3. -------- ---------
4. Time float (time of measurement)
5. ANM float (not relevant for this assignment)
6. Non – ANM float (not relevant for this assignment)
7. Total float (renewable power generation in MW)

The weather forecast provides the parameters wind speed and direction. The forecasts are sourced from the MetOffice, which is the UK’s governmental weather service. There are 3 hours between each timestamp and the data is stored in the measurement “MetForecasts”, following the schema:

1. name : MetForecasts
2. Key Type
3. -------- ---------
4. Time float (Target time of forecasts)
5. Speed float (Wind speed in M/S)
6. Direction string (Wind direction , e.g. "S" or "NW ")
7. Source\_time integer (Time of forecast generation)
8. Lead\_hours string (Forecast horizon in hours , actually a tag)

The data is stored in an InfluxDB (non-relational time-series database). It does not have tables with rows and columns, but the data is stored in measurements with fields and tags.

The Lead time string is the difference between the source time (time at which forecast was generated) and the target time (time that is forecasted). The wind direction is encoded as a string, however, it is naturally circular. Thus, the text string needs to either be mapped to degrees or radians, or encoded as categorical vector.

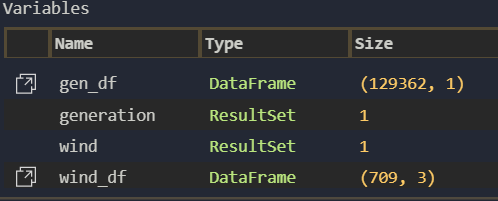
1. **Data quality dimensions**
   1. **Accuracy of the data of Power Generation dataset**

In general Wind Power Data is quite accurate. Mainly due to multiple reasons such as but not limited to:

* + 1. Power Meters installed at the voltage levels of Wind Power generation needs a good accuracy class, since they are used for tariff.[[1]](#footnote-1)
    2. The producers PID(E) controllers needs the Wind Power Data to control the power generation of non-fluctuating power sources, such as coal, gas, biomass power plants, in order to stabilize the frequency on the electricity power grid. Non-high integrity data could result in blackouts.[[2]](#footnote-2)
  1. **Accuracy of the data of Weather Forecasting**

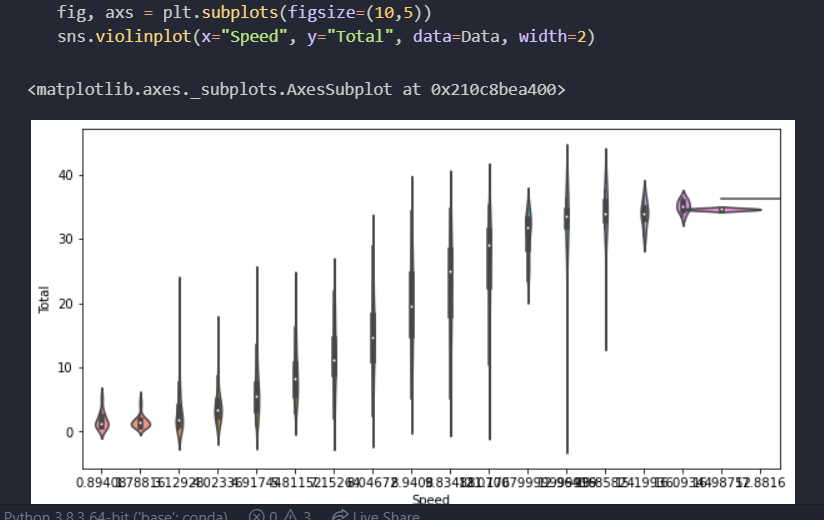
In general complex models with a marked uncertainty. The closer the Lead\_hours the higher the accuracy. The auther makes the assumption that the forecast data is fairly constant in-between the occurrence of updated source\_time (Time of Forecast Generation)

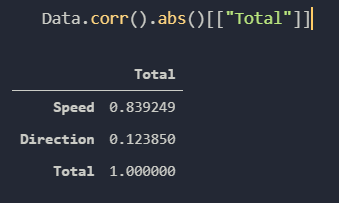
* 1. **Completeness of the data**

The two datasets are vastly different in size, as the Power generation**(*gen\_df***) is logged for every minute, whereas the weather **forecasting(*wind\_df)*** is generated every 180min or 3 hours.   
The difference in log-interval, results in the DataFrames of 129362 vs. 709 rows, as shown on the picture below.

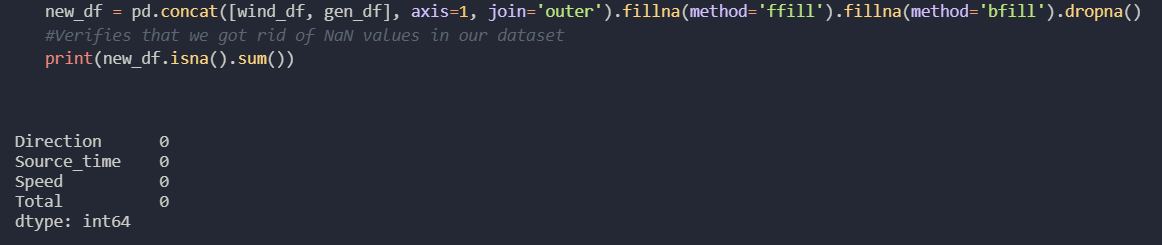
One could choose to perform an inner join to align the data on their timestamps, which is an easy fix, but also results in a significant derating of the data-amount for power generation.

Another option would to make an outer join and handle the massive amount of Nan spots in the data. Simply replacing Nan values with averages or means is a horrible idea when it comes to model Wind Power generation. This is due to the fact that power generated from a wind turbine is a cubic relation of wind velocity given by the relation below[[3]](#footnote-3):

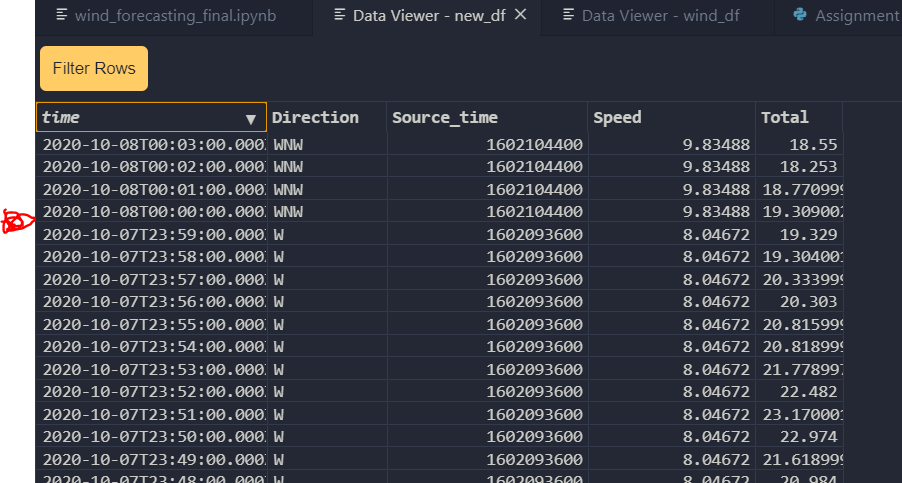
The Violin plot to the right shows the relation brilliantly. Take carefull note of the saturation of the power curve, once the generators reach maximum power output at almost betz limit turbine blade pitch. Asking for data correlation, also states clearly implies windspeed as the most predominant factor.



In order to keep as much data as possible for later wrangling, it was chosen to make an outer join and handle the Nan-values with fillna\_functions, where the missing “Direction” and ”Speed” rows are filled out with the latest value, until a new value is encountered. It is considered a fair assumption that weather stays fairly consistent in the 180min intervals.



The red mark, shows where the fillna function encounters a new value and starts now applies the updated value in eliminating the “missing data”. One could argue that a decision tree could be applied in the pipeline to do this, provided the author had access to deeper python skills.



**The rest less relevant Quality dimensions (skipped for now its 06:15 AM after all)**

* 1. **Consistency of the data – consistent.**
  2. **Timeliness – Its timely.**
  3. **Believability – I Believe it.**
  4. **Interpretability – language is translated in data access layer in SQL-like query strings.**

1. **Data pre-processing pipeline**

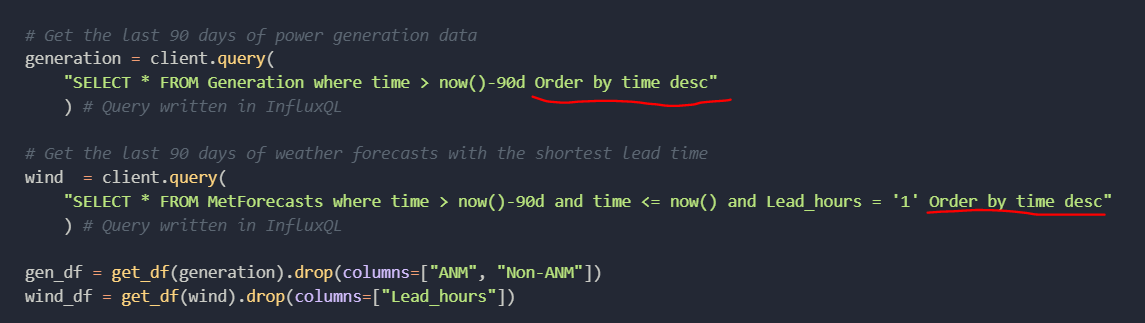
**4.1 Requirements**

For this assignment you should create the data preproccesing pipeline using sklearn in a system

that:

• Reads the latest data from the InfluxDB

* **ACSC: this was conducted by simply adding a “Order by time DESC” statement to the SQL string query. This ensures we start processing the latest data and not the 3 months old stuff.**



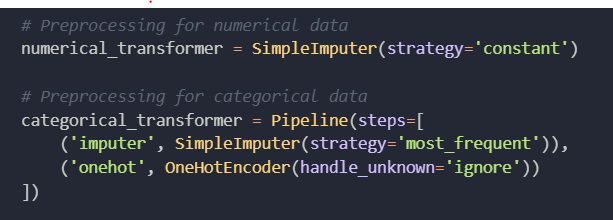
• Prepares the data for model training, including

**–** Aligning the timestamps of the two data sources (e.g. through resampling)

**ACSC: Please refer to Section 3.C of this report.**

**–** Handling missing data

**ACSC: Although the basic fillna function from the panda package did handle all the missing the data, the assignment still requires us to apply the sklearn package. Therefore, SimpleImputer was applied to both numerical data and categorial data in the Pipeline transformers.**



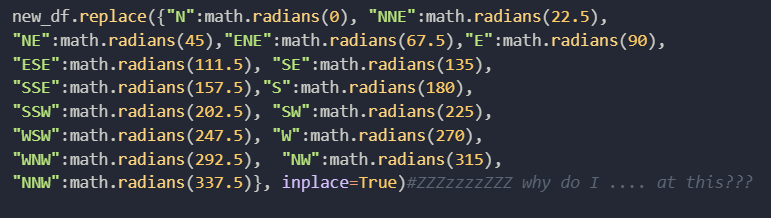
**–** Altering the wind direction to be a usable feature (by mapping to radians or encoding

as categorical)

**ACSC: The “Direction” is stored in the categorial\_column, since it's a String (object).**

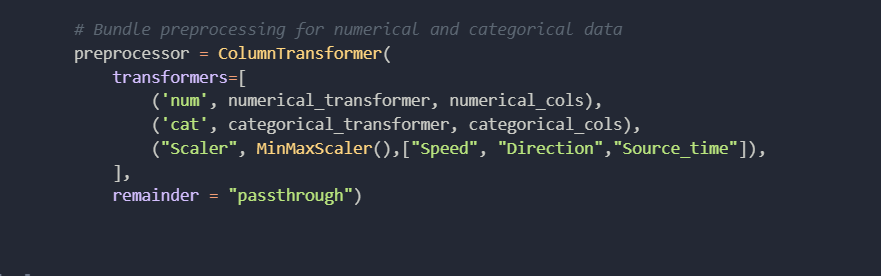
**The OneHotEncoder should ideally convert these objects to some primitive date type once called.**

**For some reason this did not work 100% of the time, which would later throw errors on wrong datatypes for scaling transformer etc.   
Therefore tedious and manual coding was applied to convert the “Direction” column into Radians. This is of course not a feasible solution in the long run!**

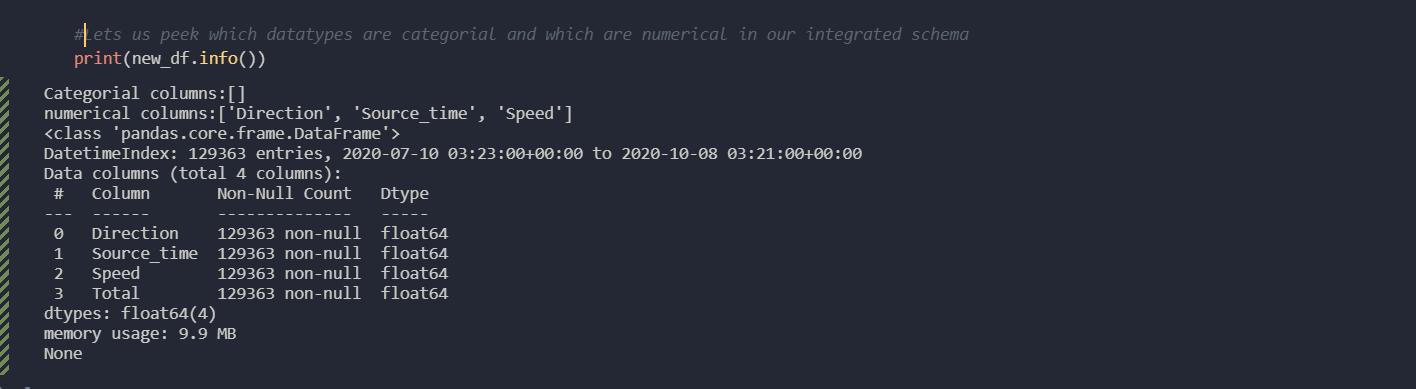


**–** Scaling the data to be within a set range

**ACSC The work however paid of since, the MinMaxScaler worked “Direction”**

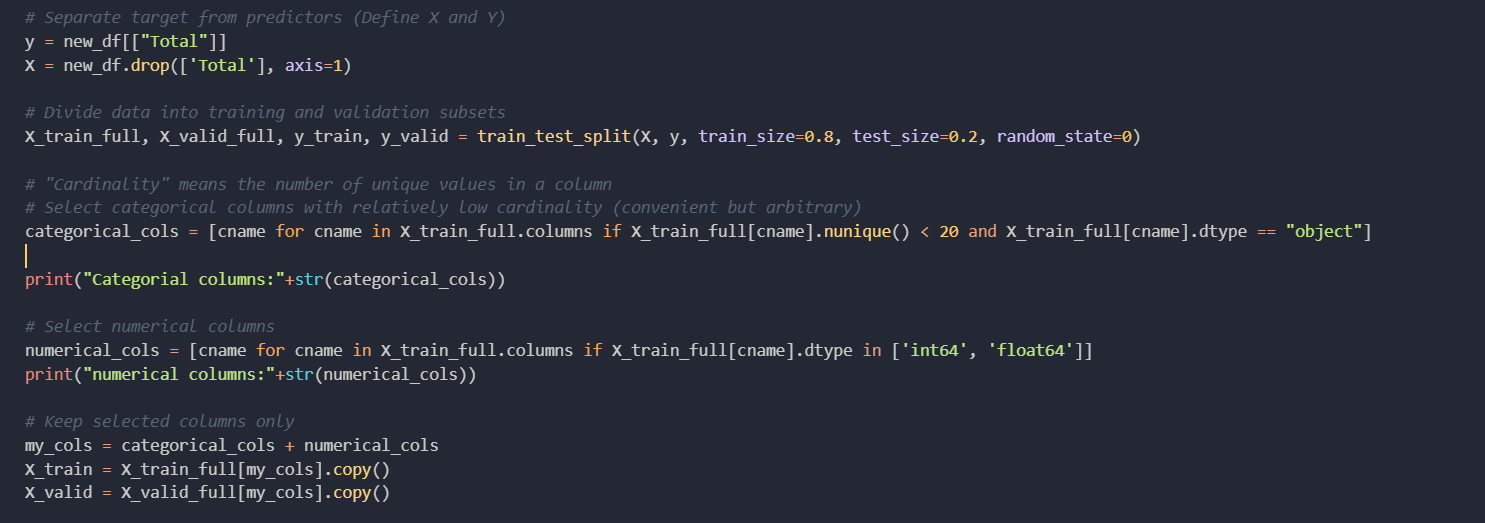


**The categorial column kind of became obsolete in the since only float64 Dtypes where left after the conversion… oh well..**

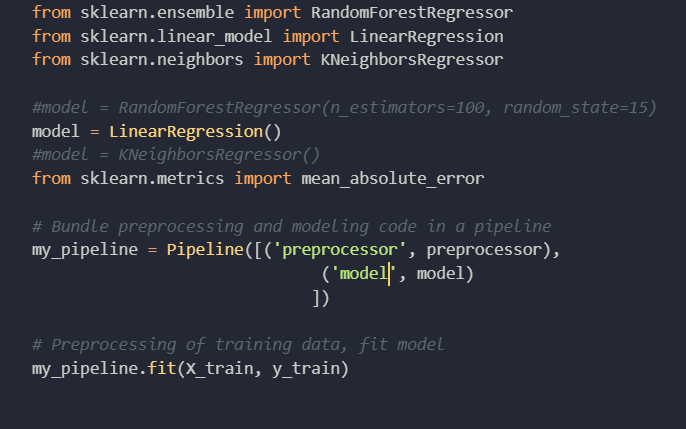


• Trains a linear regression model

**ACSC: The data was split into a 80/20 percent train vs test size.**



**Then the date was further prepared for numerical and categorial transformer preprocessors.**

**The pipeline was finally defined as the preprocessor and the model which was used to fit the train data. With this parameter design, one can easily switch out different models.**

**The Linear Regression model was tested against mean absolute error score giving the value of 3.78.   
  
It was more intuitive to see how the model predicts the actual data visually.**



• Uses the model to forecast wind power production

**ACSC: The pipline was applied to forecast “Total” from the latest weather data. This date is orginally returned as an ndArray, and can be formatted into a DataFrame that can be plotted to give an indication of the models characteristics. Its evident that “Speed” and “Direction” can now predict the power production given the weather data.**



Much of the scaffolding for this is provided in template.py, so you primarily need to focus

on the pipeline.

**ACSC: the scaffolding template was converted into a Jupityr notebook and further developed in VSC code.**

The source code can be found as a Jupiter notebook in the following GitHub repository: <https://github.com/asgerhaug/Data-Science-tutorial-VS-Code-Titanic.git>

**4.2 Improvements**

**ACSC: The OneHotEncoder should ideally work for the categorial data types, and this is a must to get it working for future projects. In addition it was annoying to have to drop the index, due to it being bound to the timestamp. Somehow it would be great if that could be handled by dropping the timestamp in the pipeline.**

**ACSC: Although interesting literature, I did not find this reading very helpful… perhaps some more live coding recordings would be ideal. Many of us SD’s are still novice in the python syntax, the author considered in this category as well…**

**Suggested reading and useful links**

• Page 66-76 ofHands-OnMachine Learning, especially the difference between estimators,

transformers and predictors.2

• User guide for sklearn pipelines

• sklearns Pipeline API

• sklearns LinearRegressionmodel

**ACSC: Super cool ideas. But out of time.**

1. **Extentions**

If you think wind power forecasting is extremely interesting and soo much fun (which you

of course do, i mean, why wouldn’t you?), here’s some ideas for extending your model and

pipeline. These aren’t especially connected to the course, so there is no expectation for you

to do any of these.

• The power curve in Figure 1.1 does not look especially linear, does it? So maybe the linear

regression is not a great fit. Add polynomial features to the wind speed input to handle

the non-linearity.

• How accurate is your model? Evaluate your model using non-shuffled K-fold cross validation

• What works better, converting the wind direction to radians or to a categorical encoding?

• The current model is oblivious to the time sequence of the data. Exchange the linear regressionmodel

with a Auto-RegressiveMoving-Average eXogenous input (ARMAX) model

or a Recurrent Neural Network with Long Short-TermMemory cells

1. Source: IEC Standard 62053 [↑](#footnote-ref-1)
2. Source: Author is an Electrical Engineer [↑](#footnote-ref-2)
3. Source: Cengel, Turner, Cimbala, 2008, Fundamentals of thermal fluid sciences 3rd edition. [↑](#footnote-ref-3)