Large Scale Data Analysis

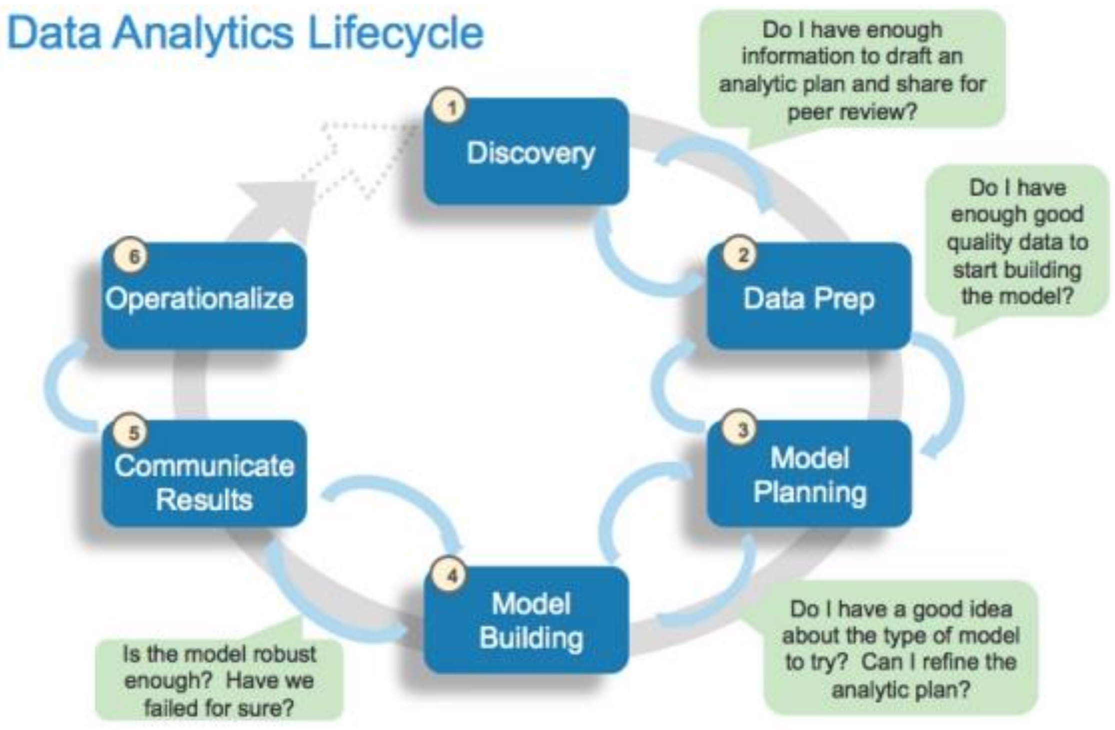
BSLASDA1KU

Andreas Magnus Benggaard

andbe@itu.dk

Assignment 2: ML Lifecycle

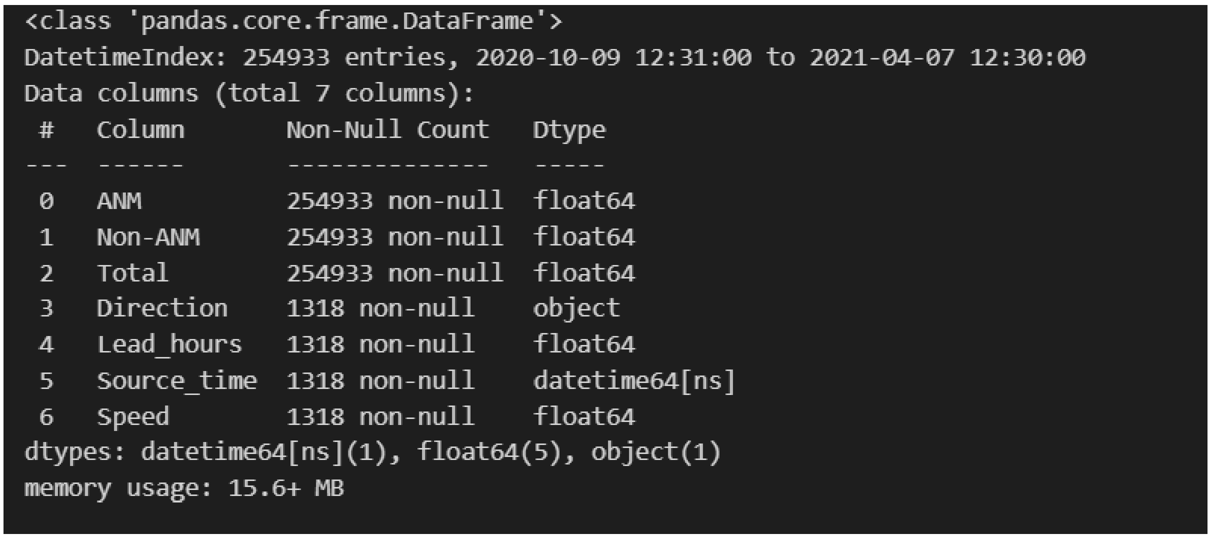
Tracking, reproducibility, and deployment using MLFlow.


Data Analytics Lifecycle
Do I have enough information to draft an analytic plan and share for peer review?
1 Discovery
2 Data Prep
Do I have enough good quality data to start building the model?
3 Model Planning
Do I have a good idea about the type of model to try? Can I refine the analytic plan?
4 Model Building
Is the model robust enough? Have we failed for sure?
5 Communicate Results
6 Operationalize

LINK TO GITHUB REPO: https://github.itu.dk/andbe/LSDA_A2.git

This report revisits work done from the first assignment of Large Scale Data Analysis. This project aims to extend the work we did regarding the wind power forecasting system by incorporating a machine learning pipeline. This model uses recent weather and energy data from Orkney to build a model predicting wind energy generation based on weather conditions like wind speed and direction. The best-preforming model is then saved and, in this assignment, will be served into a Azure virtual machine, allowing others to forecast the energy generation for future weeks. The assignment will utilize MLFlow, which will help manage the machine learning lifecycle to log results based off the best model and parameters. Allowing others to easily reproduce the results and deploy the developed pipeline through the cloud.

In this case, we did not have to get the data from a live database. We had to load a dataset given to use which was a .JSON file. This data set stores 180 days of weather data and energy production that was been inner joined. This sample ranges from 2020-10-09 until 2021-04-07. The relevant columns for this assignment are Speed, Direction and Total, which is important to note when designing the pipeline. The Speed column stores windspeed in m/s whereas Direction labels the direction of the wind. Total is the measurement for the total wind energy production. Below is a screenshot of the data frames information.


```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 254933 entries, 2020-10-09 12:31:00 to 2021-04-07 12:30:00
Data columns (total 7 columns):
 #   Column       Non-Null Count   Dtype
---  ------       --------------   -----
 0   ANM          254933 non-null  float64
 1   Non-ANM      254933 non-null  float64
 2   Total        254933 non-null  float64
 3   Direction    1318 non-null    object
 4   Lead_hours   1318 non-null    float64
 5   Source_time  1318 non-null    datetime64[ns]
 6   Speed        1318 non-null    float64
dtypes: datetime64[ns](1), float64(5), object(1)
memory usage: 15.6+ MB
```

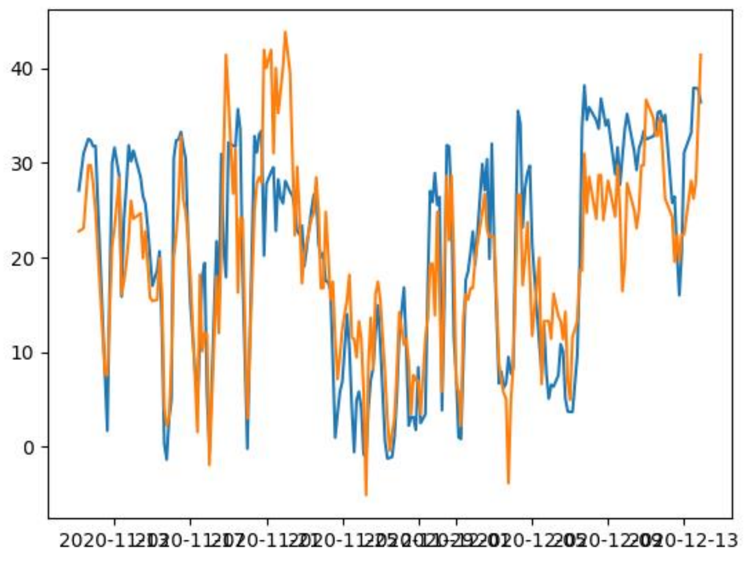
The next step was creating the pipeline. I tried to keep it as simple as possible in this assignment, using the same pipeline and preprocessing as used in the first assignment. Before implementing the pipeline, It is important to note that I am using the template file given to use through the LDSA github repository. The first thing I did was to use Pandas.Dropna()

function to remove all the NA values in the dataset. This left the dataset with 1784531 rows. Secondly, I used the Column transformer from the previous assignment to one hot encode and Simple impute the important columns which were Direction and Speed. These two columns needed to be transformed in order for the machine learning algorithms to train. The only regressor was linear regression in this case, this was because I wanted to keep it as simple as possible in order to focus my time on the more important steps in this assignment.

We also had to implement more evaluation metrics. I decided to use three different evaluation metrics. These were Mean absolute error, mean squared error and the R2 score. I chose more than 3 metrics, because it provides a more comprehensive understanding of the model's performance. You can get different perspectives of the performance through different metrics because each metric is looking for something specific. If for example, accuracy was bad, you would know that the specific model was poor at guessing correctly classified instances. After running the main script the following metric results can be seen in the table below.

MAE	5.204
MSE	49.524
R2	0.4168

And an illustration of the model:



As you can see from the metrics, the model's performance is quite poor, and not able to accurately predict. There could be several solutions to solve this issue, the two could be adding more algorithms and choosing the best model or giving the model more data to train on.

The next step was making the script file able to run through MLflow. For MLflow to work I had to create a conda environment and add a file called “MLProject”. This is so, when I or others want to run the experiment, the correct environment, packages and versions of software are downloaded ready to run the model without errors. Also, to make running the script useful, the model and metrics were logged, this was to be able to compare models and make sure that when new data comes in the newest best model is run, this is not really relevant when I am only using linear regression in this case. MLflow is a useful tool to both track experiments and package models. MLflow allows you to log and track all experiments that have been performed including parameters, metrics and artifacts. This can help improve results. MlFlow allows you to package models making it very easy to deploy into cloud services such as azures virtual machine.

The two files mentioned above contain information that MLFlow needs to know before running the main script. The environment file, conda.yaml file contains all the necessary channels, python versions and all the pip installs that the environment needs to run to remove errors. The MLProject files contain which environment needs to be run and the entry point along with its command to access the main script and run the actual pipeline.

To make the code I have created useful, we had to be able to make it reproducible and deploy it to the cloud. So the MlFlow project was pushed to a GitHub repository[1]. This would make it easy to deploy it to the azure VM. The azure was set up following the instructions given, and miniconda along with MLFlow was installed into the virtual machine. The next step was to ssh into the virtual machine. This was done using the following command on powershell:

```
ssh -i C:\Users\andre\sample andbe@20.164.43.41
```

important to note, It would not be possible for others to use the command above since I am using private and public keys to access the virtual machine. Then running mlflow with the GitHub to run the model.

```
mlflow run https://github.itu.dk/andbe/LSDA_A2.git
```

[1] https://github.itu.dk/andbe/LSDA_A2.git

The model was able to be served both locally and on the virtual machine. On the other hand, I was unable to curl the query to get a prediction out, both on the local machine and on the virtual machines. I attempted served curl variations including the ones linked on the website and none worked.

In conclusion, MLFlow is a sort of all-in-one package when you must create a machine learning pipeline. By making the pipeline comprehensive and flexible, it’s a solution for simplifying the complexity of a machine learning lifecycle and allowing everyone to easily implement different models.