IT University of Copenhagen
Big Data Management

Data Pipelines & Data Analytics Lifecycle

Forecasting the Wind Power Production in Orkney

October 24, 2023

1 Introduction

In this assignment you will play the role of a Data Scientist consultant for the Scottish and Southern Electricity Networks (SSEN). Among many other areas, the SSEN provides electricity for the Okrney islands, in Northern Scottland. This archipelago has significant wind and marine energy resources, and it generates over 100% of its net power from renewable sources, coming mainly from wind turbines situated across Orkney. This is good news for Orkney and the environment, but although wind farms provide emissions-free energy, they only generate electricity when the wind blows.

Your job? To design and implement a retraining pipeline (in the non-interleaved sense of the word) - including data fetching, preprocessing, evaluation and model storing - for a small wind energy forecasting system. The goal here is to use weather forecasting data to predict the energy production for Orkney. You will need all your data scientist skills for insights, and you will have some libraries at your disposal, such as `Scikit-Learn` and `MLFlow` (and as many others as you might need).

This document will give you an extremely brief intro to wind power forecasting, describe the data you will be working with, list the requirements for the assignment and provide suggested reading and useful links. You will also have access to a Jupyter Notebook Template, to help you started.

1.1 Wind power forecasting

As you might expect, the estimated wind speed is the primary input to any model that seeks to estimate wind power production. If we make a scatter plot of wind speed against power output, it might look like Figure 1.1. If you squint at the data, you'll see a sigmoid-ish curve, called the "power curve". The power curve is essential for wind power forecasting[1], as it describes the relationship between wind speeds and power output.

[1] If you are interested in learning more about this, check out this website.

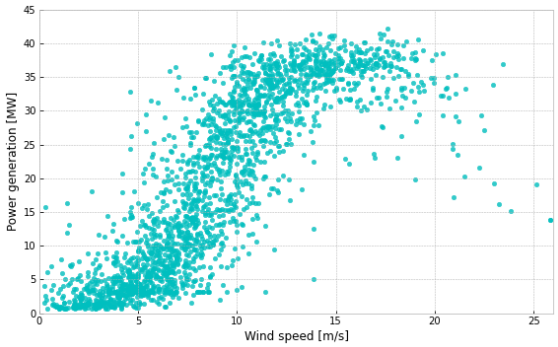


Figure 1.1: Wind speed plotted against power generation, showing a "power curve" for Orkney

But the wind might meet different environments, depending on which direction the wind is coming from, affecting the wind speed or creating turbulence. So we should probably include the wind direction as a feature in our model, increasing the dimensionality of the input.

Wind direction is reported by the direction from which the wind originates in a cardinal direction, and thus encoded as a string in the data. For example, a record containing 'SW' in the wind direction field would indicate that the wind is blowing from the South-West (and to the North-East). Since our models can only handle numeric data, you will need to perform some transformation on this feature.

2 The Data

You will be working with data from two sources:

1. Orkney's renewable power generation - Sourced from Scottish and Souther Electricity Networks (SSEN)
2. Weather forecasts for Orkney- Sourced from the UK MetOffice

The data is stored in an InfluxDB, which is a non-relational time-series database. InfluxDB can be queried using InfluxQL, a "SQL-like" query language for time-series data. InfluxDB does not have tables with rows and columns, instead data is stored in *measurements* with *fields* and *tags*[2].

2.1 Renewable energy generation

The power generation data is sampled every minute from a public website and is stored in the measurement `Generation` with the following "schema":

```
name: Generation
Key       Type
--------  ---------
time      float    (Time of measurement)
ANM       float    (Not relevant for this assignment)
Non-ANM   float    (Not relevant for this assignment)
Total     float    (Renewable power generation in MegaWatts)
```

2.2 Weather forecasts

The weather forecasts are sourced from the UK governmental weather service, the MetOffice. For forecasts we talk about `Source time`, the time at which the forecast was generated,

[2]Tags are always strings and are indexed, while fields are not indexed

`target time`, the time that is forecasted, and `lead time` or `forecast horizon`, the difference between the source time and the target time. There are 3 hours between each timestamp.

The forecasts are stored in the measurement `MetForecasts` with the following "schema":

```
name: MetForecasts
Key          Type
--------     ---------
time         float    (Target time of forecasts)
Speed        float    (Wind speed in M/S)
Direction    string   (Wind direction, e.g. "S" or "NW")
Source_time  integer  (Time of forecast generation)
Lead_hours   string   (Forecast horizon in hours, actually a tag)
```

3 Data Analytics Lifecycle

The Data Analytics lifecycle is the cyclic iterative process with instructions and best practices to use across defined phases while developing a Machine Learning workload. This lifecycle adds clarity and structure for making a machine learning project successful. The steps behind this cycle can be seen in Figure 3.1.

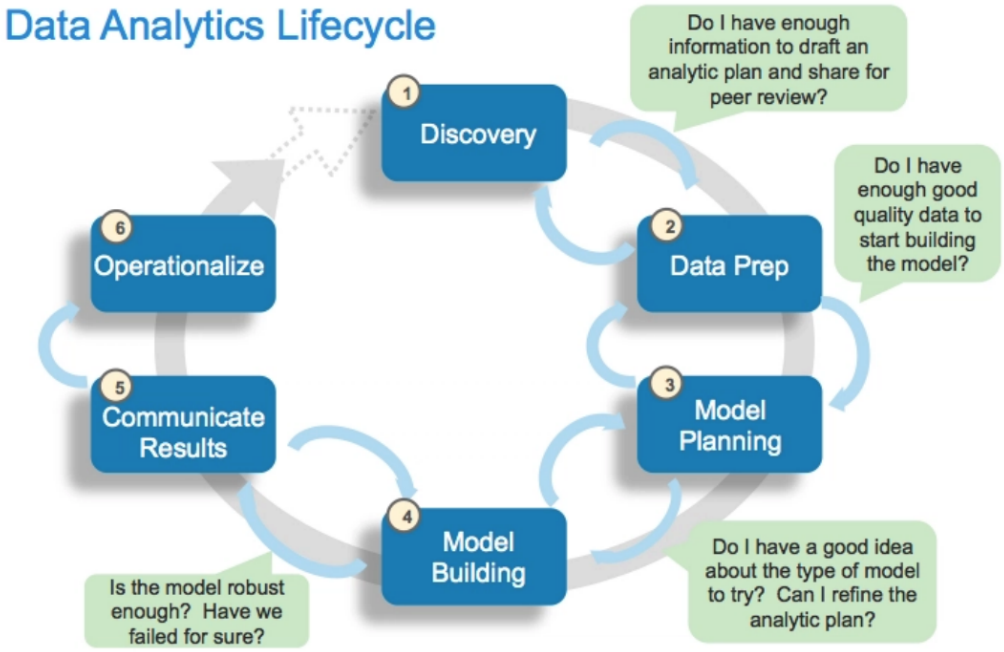


Figure 3.1: The data analytics life cycle. MLflow addresses tracking (2,3,4), reproducibility (4,5), and deployment (6)

How does it fit our project? Since we want to try out different experiments (i.e. different models and different parameters), it could be useful to build a setup that allow us to easily compare the results of these experiments, to select and store the best predictive model. To do this, we will use MLflow, a set of open source tools for the machine learning lifecycle.

MLflow covers three main areas: Experiment tracking, reproducibility, and model deployment. Prior to working on this assignment, we highly recommend you go through the MLflow Tutorials in some detail.

3.1 Experiment tracking

When developing ML models, you go through a lot of experiments, trying many different combinations of models, hyper-parameters and feature extraction. This means you need to keep track of a lot of metrics, and how you created each metric. This is what MLflow Tracking is trying to solve.

In MLflow each experiment can consist of many *runs*, and in each run you can log *parameters*, *metrics*, *tags* and *artifacts*. To get a better explanation of this, visit the MLFlow tracking introduction. A simple experiment could look like this:

```
import mlflow

with mlflow.start_run ():
    mlflow.log_param ("Param one", 1)
    mlflow.log_metric ("Accuracy", 0.5)
```

In this example, we are indicating MLFlow that we want to log a certain parameter (e.g. a polynomial degree), and a certain metric, both with their correspondent value (in reality, it would be useful to have a variable instead of a constant value).

By default, MLflow will log your runs to the default experiment, and will save all your runs to your local file system in the folder mlruns. After running the example above, the directory structure could look like this:

```
mlruns/
    0/
        1f880fec49d64bffa1fdd4e7600f7c5b/
            artifacts/
                meta.yaml
                metrics/
                    Accuracy
                params/
                    Param one

        meta.yaml
```

You don't have to pay much attention to this, but it is useful to identify a few key concepts:

- The experiment ID is 0 (first folder inside "mlruns")
- The run ID is 1f880fec49d64bffa1fdd4e7600f7c5b (first folder after the experiment)
- For each run, all parameters and metrics are saved (inside the "artifacts" folder, for each run folder)

These are all text files, but luckily you don't have to open them in a text editor to see the results. We will make use of a much cleaner and easier to use interface to see and compare results. You can see how the MLFlow user interface looks like in Figure 3.2.

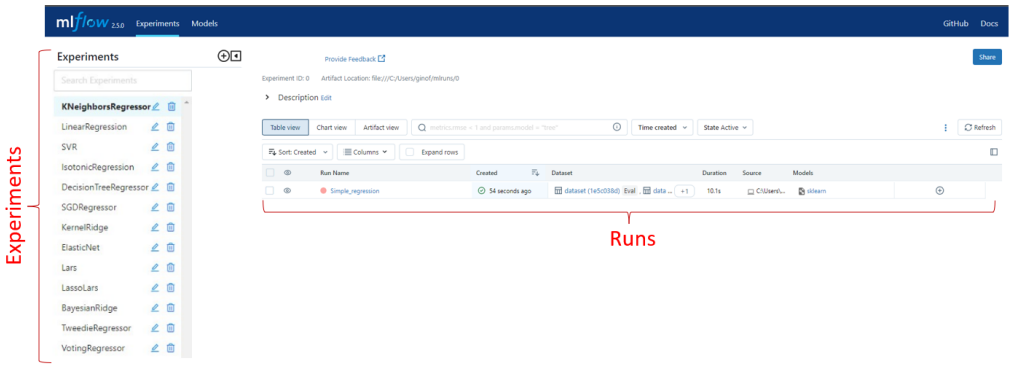


Figure 3.2: MLFlow UI for experiment tracking.

3.2 Reproducibility

Reproducibility refers to the ability of researchers, developers, and practitioners to recreate the exact same results, analyses, and outcomes from a given set of source code and data. In that sense, MLflow Projects is a standardized way to encapsulate an experiment in a reproducible manner, so colleagues or other stakeholders can reproduce your code. This is done by specifying all the dependencies as either a conda or docker environment. Here we will only talk about the conda approach.

An MLflow project consists (at least) of:

- The code you want to package, including the data for your experiments (if needed). Jupyter Notebooks are not supported.
- An environment file specifying the dependencies for the code (in this case a YAML file with the conda environment).
- An MLproject file specifying which environment file to use, parameters accepted, and the entry points to the code.

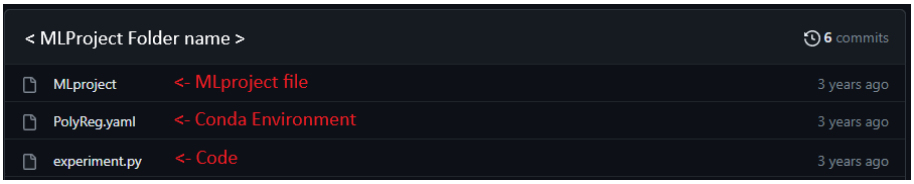


Figure 3.3: Example of MLProject folder.

```yaml
name: PolyReg
channels:
  - defaults
dependencies:
  - scikit-learn>0.23
  - numpy>1.19
  - pip>20
  - python=3.8
  - pip:
    - mlflow
    - matplotlib>3
```

```yaml
name: PolyReg

conda_env: PolyReg.yaml

entry_points:
  main:
    parameters:
      num_samples: {type: int, default: 100}
    command: "python experiment.py {num_samples}"
```

Figure 3.4: On the left, example of the Conda Environment file, specifying the depenencies. On the right, example of the MLproject file, specifying the environment file and code entry points.

We've made a small project that shows an example of polynomial regression, which we will use to show how MLflow projects work. See appendix for more detailed explanation.

3.3 Deployment

Model deployment is the process of putting machine learning models into production, making them available to users, developers or systems, so they can interact with them.

3.3.1 Local

Once you have a model you are satisfied with, you can log and deploy it using MLflow Models format. There's detailed descriptions of the deployment options in the documentation.

Example: As an example, have a look at this repository. This is an MLflow Project that trains a CAISO model for electricity load forecasting and saves it to the local filesystem. The saved model can then be distributed and deployed using MLflow Models.

To access the saved artifacts, you need to clone the repository to your file system first and then navigate to the directory where the project is saved:

```
git clone git@github.com:NielsOerbaek/PolyRegExample.git
cd caiso-mlflow
```

Then, you can use the command "`mlflow run`" and "`mlflow models serve`" to serve the model (e.g. build the necessary endpoints):

```
mlflow run .
mlflow models serve -m <name of model folder>
```

The model will now be served on your local machine. You can then query your model by opening another terminal and running:

```
curl http://127.0.0.1:5000/invocations -H 'Content-Type: application/json'
    -d '{
  "columns": ["Time"],
  "data": [["2021-04-15T20:00:00"]]
}'
```

Notice that we are sending a request (`curl`) to the localhost (in this case 127.0.0.1) to the port 5000 (the default port for MLFlow) with the recently created endpoint "`invocations`", and then passing a JSON formatted dataframe consisting of one column ("Time") with one row ("2021-04-15T20:00:00"). The model will then run its prediction and you will get an answer like `[21.49216666666666]`, meaning that your model thinks that the electricity demand in Orkney at 8 PM on Saturday the 1st of May 2021 will be 21.49 MW. Pretty cool!

3.3.2 To the cloud

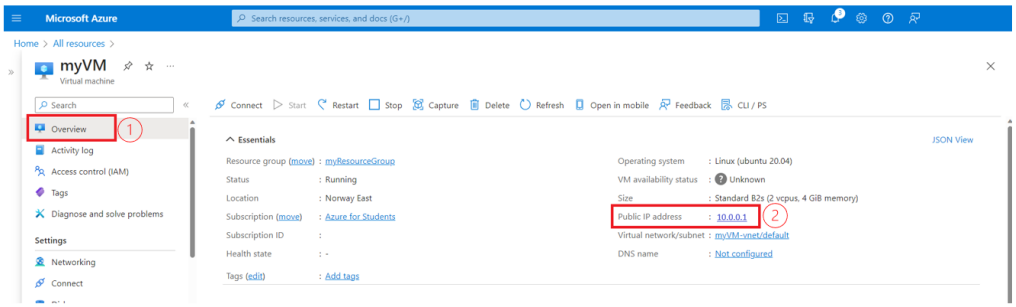
Deploying to your own machine might seem a bit useless. After all, you wouldn't want to open up connections for everyone at your laptop, and then whenever you turn off your computer, the model would be unavailable for the public. The interesting thing comes when deploying to a server and can be queried by many users.

You can look into deploying your model to a cloud-based virtual machine with a public IP address. One such option is to use Microsoft Azure. You might want to take a look at the MLFlow documentation for deployment.

Once you have a virtual machine (if not, follow section Azure VM in the Appendix) you can ssh into it and serve your model, making it available to the public:

1. Push your MLflow project to Github if you develop your project locally.

2. ssh into your VM on Azure `ssh <username>@<Public IP address>`. You can find VM Public IP address in the Overview:



3. Once connected to the VM, install miniconda:

```
wget https://repo.anaconda.com/miniconda/Miniconda3-latest-Linux-
    x86_64.sh
chmod +x Miniconda3-latest-Linux-x86_64.sh
bash ./Miniconda3-latest-Linux-x86_64.sh
```

   Follow the instructions, making sure to accept the License Agreement. If you experience any trouble, you can visit this link for more information.

4. Exit the VM and ssh again so that the conda environment is activated. You should see a (base) written before the command line prompt:

```
(base) azureuser@myVM:~$
```

5. Install MLflow from the "conda-forge" channel:

```
conda install -c conda-forge mlflow
```

6. Download your model to the VM using the GitHub repository:

```
git clone <your-git-repository-url>
```

7. Serve the model:

```
mlflow run <name of folder containing the MLproject file>
mlflow models serve -m <path to model folder> -h 0.0.0.0 -p 5000
```

   (The folder containing the model will usually be located at: `./mlruns/<experiment-nr>/<run-nr>/artifacts/model`).

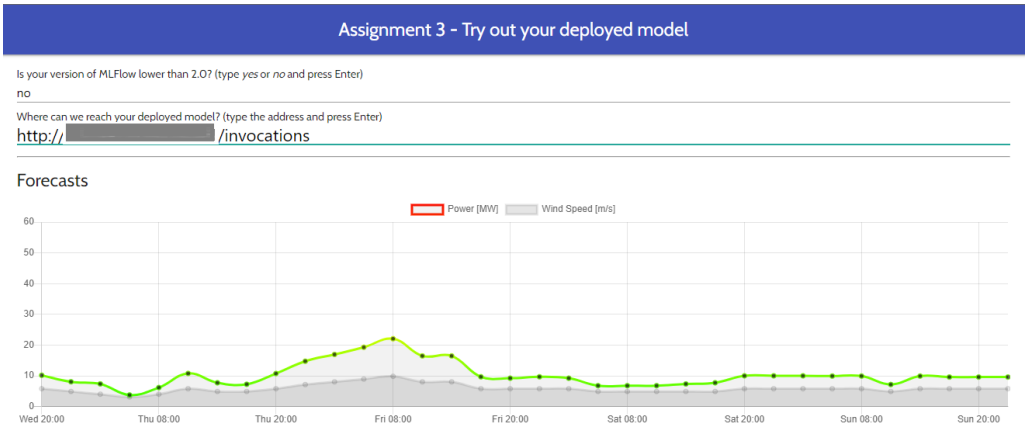
And that's it. Now your model is available to the public. To test manually your model, you can query it with an input to get a prediction:

```
curl http://<Public VM IP address>:5000/invocations -H 'Content-Type:
    application/json' -d '{
    "dataframe_split": {
      "columns": ["Time"],
      "data": [["2020-11-14T20:00:00"]]
     }
}'
```

You should get back a prediction print in the console.
(*NOTE: The input data may differ depending on your model. If your model takes other variables, you should change the content of the query accordingly*).

3.3.3 Trying out your deployed model

We've built a simple website where you can try to use your deployed model in something that resembles a real use case: On https://orkneycloud.itu.dk/mlflow/ you can insert the URL for your invocation endpoint and it will fetch the newest weather forecasts, use your model to make predictions, and draw a little graph.



4 Further options

The MLFlow library has become widely used, and thus new options to connect MLFlow projects with other platforms have appeared. Here, we present two of them for you to explore if interested:

- Azure Machine Learning: a cloud-based platform developed that allows to build, deploy, and manage machine learning models. It provides a comprehensive set of tools and services for every step of the machine learning lifecycle, from data preparation and model training to deployment and monitoring. You can also connect MLFlow with Azure ML. Check the documentation for more information.
- DAGsHub is a platform that integrates version control (such as Git) with experiment tracking. DAGsHub provides additional features for collaboration, tracking experiments, and sharing data science projects. For more information on how to use it with MLFlow, check the documentation.

5 Learning Outcomes

- You should be able to construct a Pipeline that allows you to handle all the incoming data and perform the necessary transformations to pass it to an estimator.
- You should learn to use the MLFlow Library for tracking, reproducibility and deployment of models, in accordance to the Machine Learning Lifecycle.

Note that the purpose of the assignment is not to achieve the best performing model, but to encourage you to **reflect** upon the necessary transformations, pros and cons of each decision made; and to try as many possible models and parameters for comparison.

6 Requirements and Hand-in

6.1 System requirements

For this assignment you should create the data training pipeline, including an `sklearn` Pipeline for preprocessing, in a system that:

- Reads the latest data from the InfluxDB
- Prepares the data for model training, including
  - Aligning the timestamps of the two data sources (e.g. through resampling or an inner join)
  - Handling missing data

  - Altering the wind direction to be a usable feature (by mapping to radians, encoding as categorical, or other)
  - Scaling the data to be within a set range
- Trains a (few) regression model(s) of your choice, (e.g. Linear Regression), and track the experiment using MLFlow.
- Saves the best performing model to disk, logging it along with parameters and metrics using MLFlow.

Once you have saved a model, your work is not done just yet. In a production setting, models should not be static, but kept up to date, once new and possibly better training data appears in the database. Imagine creating and saving a model today, and perhaps in a few weeks, you run your code again and train a new model. Is it better than the first model? To answer that you should also:

- Compare the newly trained model with the currently saved model, and save the best performing model.
- Use the current best model to forecast future power production.

6.2 Hand-in

Your submission should consist of:

- **Report:** describing your solution, including your design choices and their trade-offs. We value concise and well formulated arguments with supplementary figures and code snippets. The report should be in PDF format and have **a maximum length of five pages, including figures**. Although you are free to shape your report as you want, make sure these questions are addressed:
  - Which steps (preprocessing, retraining, evaluation) does your pipeline include?
  - What is the format of the data once it reaches the model?
  - How did you align the data from the two data sources?
  - How did you decide on the type of model and hyperparameters?
  - How do you compare the newly trained model with the stored version?
  - How could the pipeline/system be improved?
  - How would you determine if the wind direction is a useful feature for the model?
  - Currently the system fetches data for the previous 90 days worth of data:
    * How would you determine if this is a good interval?
    * What are the trade-offs when deciding on the interval?
    * Would accuracy necessarily increase by including more data?
  - Your choice of models and evaluation metrics (*What you are logging in MLFlow and why?* **(Section 3.1)**
  - How the evaluation errors change in terms of the cross-validation parameters? (*Are you using MLFlow to track different runs (e.g. with different parameters) of the same model? How is this helpful?*) **(Section 3.1)**
  - The advantages of packaging the experiments/models in the MLFlow formats and a comparison with other reproducibility options. (**Section 3.2**)
  - What are the main reasons to deploy your model, making it available for others (i.e. general public, internal company stakeholders, colleagues, other researchers, etc.)? (**Section 3.3**)

  Naming convention for the report: <*itu_username*>-A2-report.pdf.
- **Code:** supporting the report. You can access the course's GitHub organization[3], clone the repository, and push your code back when it is done. You are welcome to build upon any

[3] You should have received an invitation. You can access it via github.itu.dk with your ITU credentials. If you experience any trouble, please reach out to your TAs.

of the template code, or start from scratch. You should include your repository URL in the report. Regarding your pipeline, if you feel comfortable coding, you could even have a more modular approach, and build your solution as a package that can be imported whenever needed. Regarding MLFlow, *ideally*, you would build your model as an MLProject and deploy it to a VM with public IP, so your model can be used by others (make sure to provide the Public IP address for testing). This is not possible using jupyter notebooks, instead you need scripts. If you do not feel so comfortable with scripts, your code can consist of a Jupyter Notebook (you are welcome to build upon the Jupyter Notebook template provided, or to start from scratch).
Needless to say, your code should run without errors.

- **Extra points:** Setup a recurring job (e.g. using cron) that:
  - Runs your retraining project from the git repository[4]
  - Serves the saved model[5]

In a nutshell: Report with explained design choices, your code (including the pipeline and steps for tracking, reproducing and deploying your model), including repository URL, and the IP address for your model.

6.3 Notes and hints

If you are feeling stuck in your report, it might be useful to read the following section.

- **Joining the data:** How do you join and align the data will affect the amount of data you end up with for training your model. Make sure you explain your decision and reflect upon its pros and cons. Some resources: Pandas Join, Pandas Merge, Pandas DataFrame resample.
- **EDA:** Did you execute any analysis on the data? Explain why (or why not), and how that might affect your future decisions (transformations, estimator, etc.).
- **Wind direction:** How you encode the wind direction is up to you. Some possible options are:
  - OneHotEncoding
  - Transforming to degrees or radians with a custom transformer
  - Converting speed and direction to a 2-dimensional vector (explanation, example).

  Obviously, each method will have some advantages and disadvantages. Make sure to reflect upon those.
- **Scaling:** Think what is the normal range of your features. Is there highly varying magnitudes or values (or units) for your features? How can this affect your model? Should you scale your features? You should explain why would you do it, or why isn't it necessary. Check Scikit-Learn's Preprocessing and Normalization, or this article.
- **Missing values:** Have you included a step to deal with missing values? If so, how are you dealing with that (excluding them, imputing other value, etc.). Explain your decision. Check Scikit-Learn's imputers.
- **Linearity:** Take a look at the power curve in Figure 1.1. Is the relationship between the variables and the target linear? If not, did you include a step to address the nature of the relationship? (You may want to check Polynomial Features.)
- **Splitting and shuffling:** We are dealing with time series data. Is it acceptable to do a random shuffle on the data when splitting for training? If so, why? If not, how can you split the data? Make sure to argue your decision. Check Scikit-Learn's TimeSeries Split, or this article.

[4] To start long-running processes that don't stop when you terminate your ssh connection, have a look at the `nohup` command. To install new services (like MinIO), you might want to use docker. Think about how these steps could be automated.

[5] You might need to kill the previous job. Have a look at `killall` (e.g. `killall -g mlflow`)

- **Pipeline:** From the `sklearn` documentation on Pipelines: *All estimators in a pipeline, except the last one, must be transformers (i.e. must have a transform method). The last estimator may be any type (transformer, classifier, etc.).* This means that you can have your model (predictor) as the last step in your pipeline and have whole process from new data to prediction in one object with the `predict`-method. This is also useful when storing and loading trained models.
- **Feature transformations:** If you want to apply different transformations to different columns (e.g. categorical and numerical), you can use the ColumnTransformer.
- **More sophisticated models:** Training and evaluating sequence-oblivious models is simpler, but there might be some accuracy to gain in using a sequence prediction model, like ARIMA, or a Recurrent Neural Network.
- **Trying models and parameters:** You are probably going to try at least a couple different models, and for each model maybe several parameters. It could be efficient to automate this by iterating over different configurations. Can you think of easy ways to programmatically do this? Maybe a dictionary could help.

7 Further resources

Here is a list of resources for this assignment, including some suggested readings:

- Forecasting wind power Forecasting in Mathematics: Wind Power Forecasting
- The Power Curve of a Wind Turbine
- Hands-On Machine Learning (pages 66 to 76), especially the difference between estimators, transformers and predictors.
- User guide for Scikit-Learn Pipelines
- Scikit-Learn Pipeline API
- InfluxDB and InfluxDB documentation.
- InfluxQL documentation.
- Data Analytics Lifecycle
- MLFlow Tutorials
- MLFlow Documentation
- Docker containers
- Azure Virtual Machines course.

8 Appendixes

8.1 Using MLflow Projects: PolyRegExample repo

We've made a small project that shows an example of polynomial regression, which we will use to show how MLflow projects work. The main experiment, in which we simulate some data and model it using different degrees of polynomial regression, is defined in `experiment.py`:

```
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.pipeline import Pipeline
from matplotlib import pyplot as plt
import numpy as np

import sys
num_samples = int(sys.argv[1]) if len(sys.argv) > 1 else 100

def get_ys(xs):
    signal = -0.1*xs**3 + xs**2 - 5*xs - 5
    noise = np.random.normal(0,100,(len(xs),1))
    return signal + noise

```

```
X = np.random.uniform(-20,20,num_samples).reshape((num_samples,1))
y = get_ys(X)

plt.scatter(X,y,label="data")

for degree in range(1,4):
    model = Pipeline([
        ("Poly", PolynomialFeatures(degree=degree)),
        ("LenReg", LinearRegression())
    ])
    model.fit(X,y)
    plotting_x = np.linspace(-20,20,num=50).reshape((50,1))
    preds = model.predict(plotting_x)
    plt.plot(plotting_x, preds, label=f"degree={degree}")

plt.legend()
plt.show()
```

The Conda environment for this experiment is specified in `PolyReg.yaml`:

```
name: PolyReg
channels:
  - defaults
dependencies:
  - scikit-learn>0.23
  - numpy>1.19
  - pip>20
  - python=3.8
  - pip:
    - mlflow
    - matplotlib>3
```

Finally, the `MLproject`-file specifies how to run the project:

```
name: PolyReg

conda_env: PolyReg.yaml

entry_points:
  main:
    parameters:
      num_samples: {type: int, default: 100}
    command: "python experiment.py {num_samples}"
```

With these three things in place, you can run the experiment using `mlflow run <path to project>`. So if the project is located on your computer, you can navigate to the directory and do:

```
mlflow run .
```

Because this project is hosted as a git repository, you can simply do:

```
mlflow run git@github.com:NielsOerbaek/PolyRegExample.git
```

This will fetch the project, resolve the environment, and run the main entry point with the default parameters.

If you want to run the experiment with 500 samples, instead of the default 100, you can do:

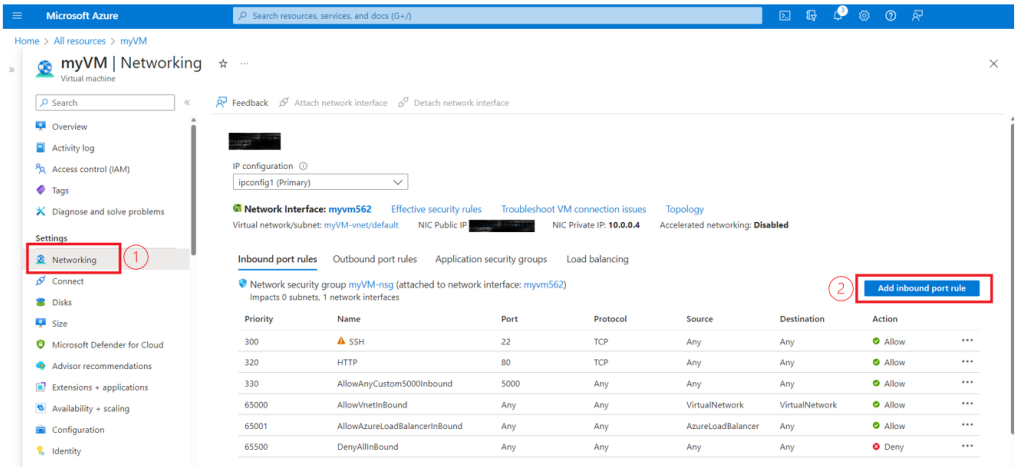
```
mlflow run git@github.com:NielsOerbaek/PolyRegExample.git -P num_samples
    =500
```

8.2 Azure VM

One option for getting a VM with a public IP is to use Microsoft Azure. You should all be eligible for free student credits, giving you some credits to set up your virtual machine.

You can follow these guides to get you started:

1. Setup your student account
2. Setup a VM with a public IP. Pay attention to the following settings:
    a) **Region:** You could pick a VM residing in a region powered by much sustainable energy (like Norway or Sweden).
    b) **Image:** In the Image choose Ubuntu Server 20.04.
    c) **Size:** You should pick one of the cheaper, like "Standard B2s" but ensure the machine has more than 2GB of RAM.
    d) **Authentication type:** We recommend to use a Public key. You can re-use an existing one located on your laptop if you have one. If not, Azure will automatically generate an SSH key pair for you and allow you to download and store it for future use. You will need to move the downloaded key to your ssh folder (usually, for Ubuntu is located at `~/.ssh`).
    e) The rest of the settings are optional (names and security groups). You can leave the defaults if you are not so comfortable setting up virtual machines.
3. Before you connect to your VM, you should open the port 5000, since that is the default port used by MLFlow. To do this, go to your recently created VM in the Azure portal, and on the left, click on "Networking", and the "Add inbound port rule". There, set Port 5000 to allow all connections.



After these steps you should be able to ssh into your VM and serve your model or MLflow UI[6]. If you are inexperienced using SSH, we recommend you read through "CS Missing Semester" on SSH keys and remote machines..

[6]MLflow also has methods for deploying directly to Azure, but we've had mixed experiences with it in the past. Again, have a look at the documentation.