Stanford University

Final Project Report

Thomas Sounack

Professor: Eric Darve

ME 343

Contents

1 Description of the project 2

2 Technical problem of interest and its difficulties 2

3 Algorithm description 2

4 Published literature and available approaches 3

5 Algorithm implementation 3

6 Algorithm verification 3

7 Datasets, problem settings and parameters 4

8 Results 4

9 Conclusion and limitations 5

1 Description of the project

The goal of this project is to design a model that can predict wind speeds at different altitudes in the context of implementing offshore wind turbines. When building offshore wind turbines, on-site studies have to be made during several months to several years in order to determine which specific spots would provide the best wind speeds for the turbines.

Currently, lidar buoys are used to estimate wind speeds at various altitudes. This solution is rather costly (several thousand dollars per week for one buoy), which is why the objective is to replace the lidar buoys with a model that would compute wind speeds at various altitudes using simple sensors at sea level and satellite data. This is possible as several physical equations link conditions at sea level with wind speeds (such as the difference in temperature between the sea and the air, the rugosity - charnock - of the sea, etc...).

Using data from a lidar buoy and the ERA5 satellite, the project will consist in designing a multi-output regression machine learning model that can predict wind speeds at different heights.

This project is the continuation of an internship I did last summer in an offshore wind company. During this internship, I used XGBoost to accomplish a similar task without a good understanding of ML models and hyperparameter optimization. Through this project, I hope to be able to obtain similar or better performance with a model I built and optimized by myself, using the content I learned during this course.

2 Technical problem of interest and its difficulties

The technical problem of interest in this project is to design a machine learning model that can predict wind speeds at different altitudes in the context of implementing offshore wind turbines. The main difficulty lies in the fact that there are several physical equations that link conditions at sea level with wind speeds, and it can be challenging to accurately capture all the relevant variables and their interactions in a model.

Furthermore, predicting wind speeds at multiple altitudes simultaneously is needed as offshore wind turbines can have very different sizes, which adds another layer of complexity to the problem. Therefore, a multi-output regression machine learning model will be used to tackle this problem. This approach allows for predicting wind speeds at multiple altitudes simultaneously, taking into account the interdependence between the different altitudes.

3 Algorithm description

Since this problem requires multi-output regression and is complex to model, designing a deep learning architecture that has native support for multiple outputs appears to be an interesting approach, and has the added benefit of applying concepts taught in class during the quarter.

The model therefore relies on a sequential neural network whose structure is detailed in section 5. We arbitrarily set the model as site-dependent to limit the complexity of the problem.

4 Published literature and available approaches

During my internship, I used three papers to understand the problem at play and have a good understanding of the physical equations behind it:

- A paper on Offshore vertical wind shear by the DTU (1)
- A paper on Southern New England's vertical wind shear conditions (2)
- A study on the northern french coast wind conditions (3)

In addition to these resources, two articles from the website *Machine Learning Mastery* -(4) and (5)- have helped me understand how to perform multi output regression with pytorch.

5 Algorithm implementation

To implement this algorithm, we use a neural network model which consists of two {linear, activation, batch normalization} blocks and a block composed only of an activation layer.

To identify the best parameters we perform a random search, as we have seen in lectures that it is more effective than a grid search. The parameters that are being searched are the activation type, the batch size, the learning rate and the number of neurons in the first and second layer (the others are imposed by the input and output size). The results are as follows:

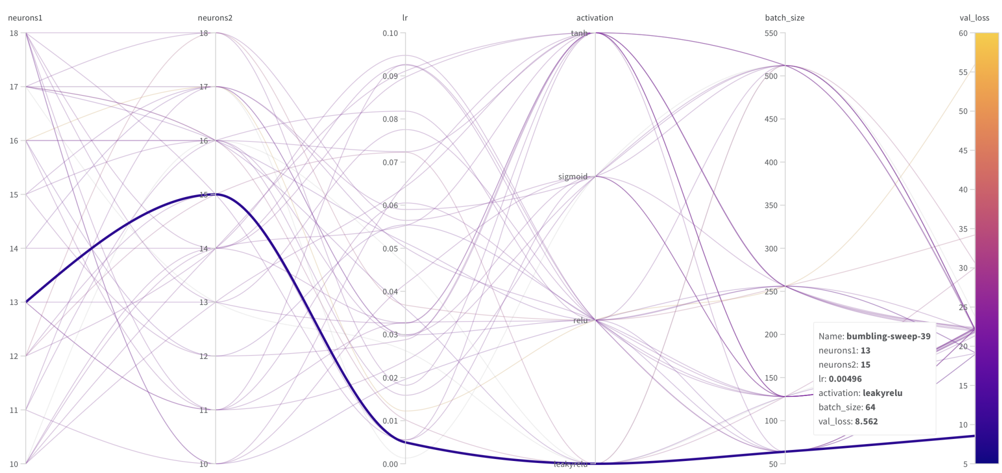


Figure 1: *Random search over the parameter space, obtained through wandb*

This allows us to identify the best configuration for our network, which will be used to evaluate the model. The code to achieve this is adapted from homework 6, and can be found in section 9.

6 Algorithm verification

To ensure that the code is working correctly, several steps have been implemented:

- The data has been imported in an auxiliary python script and tested before moving on to implementing the model. This ensures that the model receives correct data.

- Wandb is used to ensure that the model is learning properly, providing a feedback on the validation and training loss at each step.
- Finally, the model's performance is tested after the learning process (see section 8).

7 Datasets, problem settings and parameters

The dataset used comes from several sources: the buoy data (lidar + anemometer) is the one I used during my internship and belongs to the company, while the satellite data (ERA5) is open-source and available online at the desired location. Both the location and the buoy data cannot be shared in compliance with the confidentiality agreement I signed.

To ensure that the algorithm is working properly, a good sanity check is to plot the correlation matrix between the inputs and the outputs of the model.

	WS	Wdir	Delta T	p140209	msl	t2m	d2m	blh	sst	chnk	ssr	month
WS 40m	0.962580	0.056202	-0.037145	0.022448	-0.206344	-0.172457	-0.161730	0.410736	-0.189806	0.530855	-0.108940	-0.051696
WS 57m	0.951707	0.054832	-0.020029	0.014973	-0.209202	-0.166127	-0.154084	0.406570	-0.192516	0.530396	-0.108120	-0.060123
WS 77m	0.940810	0.054939	-0.006381	0.008025	-0.211938	-0.160547	-0.147685	0.404326	-0.195268	0.530346	-0.106775	-0.066966
WS 97m	0.932843	0.055677	0.001790	0.002735	-0.214116	-0.156652	-0.143227	0.404060	-0.197461	0.531089	-0.105499	-0.071202
WS 117m	0.926523	0.056475	0.008007	-0.001786	-0.215435	-0.153157	-0.139339	0.404111	-0.198920	0.532066	-0.104665	-0.073964
WS 137m	0.920922	0.056712	0.013130	-0.005586	-0.216315	-0.150151	-0.135964	0.403874	-0.200065	0.532769	-0.103706	-0.075780
WS 157m	0.916063	0.056748	0.017214	-0.008947	-0.217058	-0.147485	-0.133095	0.404001	-0.200589	0.533665	-0.103337	-0.076571
WS 177m	0.911754	0.057688	0.020273	-0.012046	-0.218251	-0.145436	-0.130836	0.405020	-0.200790	0.535067	-0.103945	-0.076427
WS 197m	0.907898	0.058215	0.022791	-0.014380	-0.219248	-0.144229	-0.129290	0.405963	-0.201181	0.536236	-0.104781	-0.076545

Figure 2: *Input - Output correlation matrix*

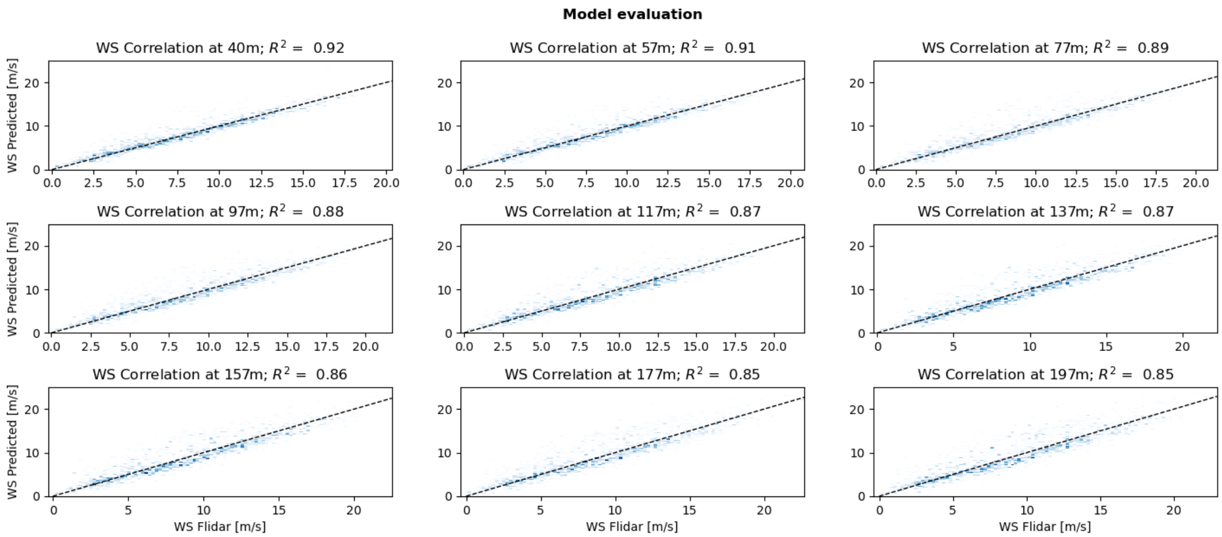
As expected, the wind speeds are highly correlated with WS (the wind speed at sea level). Other parameters like the charnok (rugosity of the sea) are also important.

We can note that (as expected) the correlation is different for the various heights. Indeed, it makes sense that trying to infer wind speeds using data at sea level would produce different results based on the desired height. This observation implies that the benchmark needs to evaluate the different heights separately.

To perform this benchmark on the validation set, we can use correlation graphs at each height along with an R^2 value for the $y = x$ line. This is mostly a visual feedback, but is useful to see if our model has a strong tendency to overestimate or underestimate wind speeds (which was a recurrent problem during my internship because of outliers). For more quantitative data, we can use a table that displays at each height the R^2 value; the average error and the average absolute error to quantify the error in m/s; the standard deviation to determine how spread-out the data is; and the average relative error to have a unitless metric of the model's performance at this height.

8 Results

Using the benchmark described above, we get the following graph:



	40m	57m	77m	97m	117m	137m	157m	177m	197m
R²	0.92	0.91	0.89	0.88	0.87	0.87	0.86	0.85	0.85
Avg. error (m/s)	0.16	0.17	0.18	0.18	0.18	0.18	0.19	0.2	0.2
Avg. absolute error (m/s)	0.81	0.92	1.02	1.1	1.17	1.24	1.3	1.36	1.41
Standard deviation (m/s)	4.04	4.16	4.29	4.4	4.49	4.58	4.66	4.74	4.8
Avg. relative error (%)	13.87	14.67	15.59	16.43	17.32	18.22	19.18	20.26	21.27

Figure 3: *Evaluating the performance of the model*

Visually, we can see that the data seems to match the correlation line well. This is confirmed by relatively high R^2 values and low error scores.

As expected, the accuracy decreases with the height. However, even at the maximal height of 197m, we have a relatively good score (21.27%) compared to the results obtained with XGBoost during my internship (29.3%). This means that the random search was effective in finding a good optimum for the hyperparameters of the neural network.

9 Conclusion and limitations

By using a random search to build our neural network, we were able to achieve satisfactory wind speed predictions. We can also note that the lower accuracy scores for the last heights is in part due to the lidar being less precise for higher measurements.

However, we can note that our predictions are still off by more than 20% for the higher wind speeds. In the context of predicting power outputs of potential wind turbines, this model could hardly replace lidar measurements. Since we have optimized the model, further improving the accuracy would require identifying and correcting the noise in our data (especially since we are using data from different sources). A good start would be to look into the precision of the lidar during rainy days (which is known to be lower than usual) as well as the precision of the ERA5 satellite.

References

[1] A. Pena Diaz, T. Mikkelsen, S.-E. Gryning, C. Hasager, A. Hahmann, M. Badger, I. Karagali, and M. Courtney, *Offshore vertical wind shear: Final report on NORSEWInD's work task 3.1*, ser. DTU Wind Energy E. Denmark: DTU Wind Energy, 2012, no. 0005.

[2] D. Borvarán, A. Peña, and R. Gandoin, "Characterization of offshore vertical wind shear conditions in southern new england," *Wind Energy*, vol. 24, no. 5, pp. 465–480, 2021. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/we.2583

[3] R Husson, L De Montera, H. Berger, P. Appelghem, "Etude de potentiel de vent à partir du sarao4 normandie 2021 -dgec 05," 2021.

[4] J. Brownlee, "Deep learning models for multi-output regression," Available at https://machinelearningmastery.com/deep-learning-models-for-multi-output-regression/ (2020/08/18).

[5] M. A. I. Khan, "Multi-target predictions with multilinear regression in pytorch," Available at https://machinelearningmastery.com/multi-target-predictions-with-multilinear-regression-in-pytorch/ (2022/12/14).

Appendix - Code

```
# %%
import torch
import torch.nn as nn
import torch.optim as optim
import pprint
import numpy as np
import os
import wandb
import pandas as pd
import matplotlib.pyplot as plt
import scipy.stats as st
from matplotlib.gridspec import GridSpec
device = torch.device("cpu")
from extract_data import *

# %%
X_data, y_data = create_dataframes()

# %%
df_corr = pd.DataFrame()
for column in y_data:
    corrM = X_data.join(y_data[column]).corr()
    corrM.drop(corrM.tail(1).index,inplace=True)
    df_corr[column] = corrM[column]

df_corr = df_corr.T
df_corr

# %%
# Hyperparameters
num_epochs = 250
log_freq = 15
```

```python
n_train = int(0.8 * len(X_data))

# Data loader
X_train = torch.from_numpy(X_data[:n_train].values).float().to(device)
y_train = torch.from_numpy(y_data[:n_train].values).float().to(device)
X_val = torch.from_numpy(X_data[n_train:].values).float().to(device)
y_val = torch.from_numpy(y_data[n_train:].values).float().to(device)
train_dataset = torch.utils.data.TensorDataset(X_train, y_train)
test_dataset = torch.utils.data.TensorDataset(X_val, y_val)

# %%
#wandb hyperparameter dictionary
sweep_configuration = {
    "method": "random",
    "name": "random_search",
    "metric": {"goal": "minimize", "name": "val_loss"},
    "parameters":
    {
        "lr": {"min": 0.0001, "max": 0.1},
        "batch_size": {"values": [64, 128, 256, 512]},
        "neurons1": {"values": [10, 11, 12, 13, 14, 15, 16, 17, 18]},
        "neurons2": {"values": [10, 11, 12, 13, 14, 15, 16, 17, 18]},
        "activation": {"values": ["tanh", "sigmoid", "relu", "leakyrelu"]}
    },
    "run_cap": 150
}
pprint.pprint(sweep_configuration)
project_name = "cme216_final_project"
group_name = "randomsearch"
sweep_id = wandb.sweep(sweep_configuration, project=project_name)

# %%
class Network(nn.Module):
    def __init__(self, neurons1, neurons2, activation):
        super().__init__()
        self.network = nn.Sequential(
            nn.Linear(12, neurons1),
            activation,
            nn.BatchNorm1d(neurons1),
            nn.Linear(neurons1, neurons2),
            activation,
            nn.BatchNorm1d(neurons2),
            nn.Linear(neurons2, 9)
            )

    def forward(self,x):
        y_pred = self.network(x)
        return y_pred

# %%
import time
t1 = time.time()
# ==============================================================================
# Training
# ==============================================================================
# Train the model
def train(config=None):
    # Initialize the new wandb run
    wandb.init(config=config, project=project_name, group=group_name)
```

```python
    config = wandb.config
    train_loader = torch.utils.data.DataLoader(dataset=train_dataset,
                                              batch_size=config.batch_size,
                                                 shuffle=True)
    total_step = len(train_loader)
    loss_list = []

    activation = None
    if config.activation == "tanh":
        activation = nn.Tanh()
    elif config.activation == "sigmoid":
        activation = nn.Sigmoid()
    elif config.activation == "relu":
        activation = nn.ReLU()
    else:
        activation = nn.LeakyReLU()

    model = Network(config.neurons1, config.neurons2, activation)
    criterion = nn.MSELoss()
    optimizer = torch.optim.Adam(model.parameters(), lr=config.lr)
    for epoch in range(num_epochs):
        for i, (train_x, train_y) in enumerate(train_loader):
            # Run the forward pass
            model.train()
            output = model(train_x)
            loss = criterion(output, train_y)
            loss_list.append(loss.item())
            # Backprop and perform Adam optimisation
            optimizer.zero_grad()
            loss.backward()
            optimizer.step()

        if (epoch+1) % log_freq == 0:
            # Calculate the validation loss
            model.eval()
            with torch.no_grad():
                X_val_pred = model(X_val)
                val_loss = criterion(X_val_pred, y_val)

            # diff_ = (X_val_pred - y_val.unsqueeze(1)).detach().cpu().numpy().squeeze()
            # diff_vec = np.reshape(diff_, (diff_.shape[0], -1))
            # val_l2_pt_error = np.mean(np.linalg.norm(diff_vec, axis=1) / np.linalg.norm(np.reshape(y_val.detach().cpu().numpy(), (y_val.shape[0], -1)), axis=1), axis=0) * 100
            # rel_error = 100 * np.linalg.norm(diff_vec, axis=1) / np.linalg.norm(np.reshape(y_val.detach().cpu().numpy(), (y_val.shape[0], -1)), axis=1)

            wandb.log({"val_loss": val_loss.item(), "train_loss": loss.item(), "epoch": epoch})
            print (f"Epoch [{epoch+1}/{num_epochs}], Step [{i+1}/{total_step}], \
                    Training Loss: {loss.item():.4f}, Validation Loss: {val_loss.item():.4f}")

    # Save the model checkpoint (optional)
    save_path = os.path.join(wandb.run.dir, "model.ckpt")
    torch.save(model.state_dict(), save_path)

wandb.agent(sweep_id, train)
```

```
t2 = time.time()
print(f"Total time taken: {t2-t1}")
wandb.finish()

# %% [markdown]
# Best performance obtained with:
# - activation: "leakyrelu"
# - batch_size: 64
# - lr: 0.0049603811225372675
# - neurons1: 13
# - neurons2: 15

# %%
train_loader = torch.utils.data.DataLoader(dataset=train_dataset,
                                           batch_size=64,
                                           shuffle=True)
total_step = len(train_loader)
model = Network(13, 15, nn.LeakyReLU())
criterion = nn.MSELoss()
optimizer = torch.optim.Adam(model.parameters(), lr=0.00496)
for epoch in range(num_epochs):
    for i, (train_x, train_y) in enumerate(train_loader):
        # Run the forward pass
        model.train()
        output = model(train_x)
        loss = criterion(output, train_y)
        # Backprop and perform Adam optimisation
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

# %%
heights = [40,57,77,97,117,137,157,177,197]

df_X = pd.DataFrame(model(X_val).detach().numpy())
df_y = pd.DataFrame(y_val.numpy())
df_X.columns = [f"WS {z}m" for z in heights]
df_y.columns = [f"WS {z}m" for z in heights]


fig = plt.figure(figsize=(15, 10))
fig.tight_layout(pad=3)
spec = GridSpec(int(np.ceil(len(y_val[0])/3))+2,3, figure=fig)
column_headers = []

row_headers = ["R ", "Avg. error (m/s)", "Avg. absolute error (m/s)", "Standard
    deviation (m/s)", "Avg. relative error (%)"]

cell_text = np.zeros((len(row_headers), len(heights)))

r = 0
c = 0

for i, z in enumerate(heights):

    if len(heights)%3 == 1:

        if c > 2 and i != len(heights)-1:
            c = 0
```

```
                r += 1

            if i == 9:
                c = 1
                r += 1

        if len(heights)%3 != 1:

            if c > 2:
                c = 0
                r += 1

        column_headers += [str(z)+"m"]


        # R   value
        lr = st.linregress(df_X[f"WS {z}m"], df_y[f"WS {z}m"]) #scipy stats built-in
        linear regression function

        cell_text[0][i] = '{: 0.2f}'.format(lr.rvalue**2)


        # Mean error
        df_subtract = df_y[f"WS {z}m"].subtract(df_X[f"WS {z}m"], axis = 0)
        subtract_avg = df_subtract.mean()

        cell_text[1][i] = '{: 0.2f}'.format(subtract_avg)


        # Mean absolute error
        df_subtract_abs = df_subtract.abs()
        subtract_abs_avg = df_subtract_abs.mean()

        cell_text[2][i] = '{: 0.2f}'.format(subtract_abs_avg)


        # Standard deviation
        cell_text[3][i] = '{: 0.2f}'.format(np.std(df_y[f"WS {z}m"]))


        # Average relative error
        df_relative_error = 100 * df_subtract_abs.divide(df_X[f"WS {z}m"], axis = 0)
        relative_error_avg = df_relative_error.mean()
        cell_text[4][i] = '{: 0.2f}'.format(relative_error_avg)



        ## Histogram

        ax = fig.add_subplot(spec[r, c])

        ax.hist2d(df_X[f"WS {z}m"], df_y[f"WS {z}m"], 100, cmin=0.0001, cmap='Blues')

        if c == 0:
            ax.set_ylabel("WS Predicted [m/s]")

        if r == int(np.ceil(len(heights)/3))-1:
            ax.set_xlabel("WS Flidar [m/s]")

```

```
    ax.set_ylim([0, 25])

    xpoints = ypoints = ax.get_xlim()
    ax.axline((xpoints[0], ypoints[0]), (xpoints[1], ypoints[1]), linestyle='--',
    color='k', lw=1)

    # on the title, I write the height of the plot and the R2 value
    ax.set_title(f"WS Correlation at {z}m; $R^2$ = {lr.rvalue**2: 0.2f}")

    c += 1


rcolors = plt.cm.Blues(np.full(len(row_headers), 0.1))
ccolors = plt.cm.Blues(np.full(len(column_headers), 0.1))

ax = fig.add_subplot(spec[int(np.ceil(len(heights)/3)):int(np.ceil(len(heights)/3)
    )+2, :])

ytable = ax.table(cellText=cell_text,
            rowLabels=row_headers,
            rowColours=rcolors,
            rowLoc='center',
            colColours=ccolors,
            colLabels=column_headers,
            loc='center',
            bbox=[0.1, 0.05, 0.9, 0.9])

ax.axis('tight')
ax.axis('off')

fig.suptitle("Model evaluation", fontweight="bold")


cellDict = ytable.get_celld()
for i in range(0, len(cell_text[0])):
    cellDict[(0,i)].set_height(.15)
    for j in range(1, len(cell_text)+1):
        cellDict[(j,i)].set_height(.1)

for i in range(1, len(cell_text)+1):
    cellDict[(i,-1)].set_height(.1)

fig.subplots_adjust(left=0.05, bottom=0.04, right=0.95, top=0.92, hspace=0.5,
    wspace=0.2)

plt.show()
```

Listing 1: main.ipynb

```
import pandas as pd

def create_dataframes():
    # Extract Satellite Data
    sat_data = pd.read_csv("A_ERA.csv", sep=",", index_col=0, parse_dates=True)
    sat_data.index = pd.to_datetime(sat_data.index)
    sat_data.drop(["longitude", "latitude"], axis=1, inplace=True)
    sat_data.dropna(inplace=True)

```

```python
    # Extract Buoy Data
    buoy_data = pd.read_csv("A_MeteoStation_Data.csv", sep=";", index_col=0, parse_dates=True)
    buoy_data = buoy_data.loc[:,["wind_speed [m/s]", "air_temperature [ C ]", "wind_direction [  ]"]]
    buoy_data.index = pd.to_datetime(buoy_data.index)
    buoy_data = buoy_data.groupby(pd.Grouper(freq="1h")).mean()
    buoy_data.dropna(inplace=True)

    # Extract Anemometer Data
    df_temp = buoy_data.merge(sat_data, right_index=True, left_index=True)
    df_temp.dropna(inplace=True)
    df_subtract = df_temp["air_temperature [ C ]"].subtract(df_temp["sst"]) + 273.15
    anemo_data = pd.DataFrame([df_temp["wind_speed [m/s]"], df_temp["wind_direction [  ]"], df_subtract]).transpose()
    anemo_data.columns = ["WS", "Wdir", "Delta T"]

    df_temp_lidar = pd.read_csv("A_Wind_Data.txt", sep="\t", parse_dates=True, index_col=0)
    df_temp_lidar = df_temp_lidar.loc[:,df_temp_lidar.columns.str.contains("wind_speed ")]
    df_temp_lidar = df_temp_lidar.groupby(pd.Grouper(freq="1h")).mean()
    df_temp_lidar.dropna(inplace=True)


    heights_A = [40,57,77,97,117,137,157,177,197]
    regex = ""
    for i in range(len(heights_A)):
        z = heights_A[i]
        regex += str(z)
        if i!=len(heights_A)-1:
            regex += "|"

    lidar_data = df_temp_lidar.filter(regex=regex)
    lidar_data.columns = [f"WS {z}m" for z in heights_A]

    df_temp_X = anemo_data.merge(sat_data, right_index=True, left_index=True)
    df_temp_X["month"] = df_temp_X.index.month
    df_temp_X = df_temp_X[df_temp_X["WS"] != 0]
    df_temp_X.dropna(inplace=True)

    df_temp_y = lidar_data.groupby(pd.Grouper(freq="1h")).mean()
    df_temp_y.dropna(inplace=True)

    df_temp_Xy = df_temp_X.merge(df_temp_y, right_index=True, left_index=True)
    df_temp_Xy.dropna(inplace=True)
    df_temp_Xy = df_temp_Xy.sample(frac=1)

    Xy_data = df_temp_Xy
    X_data = df_temp_Xy.loc[:, ~df_temp_Xy.columns.str.contains("WS ")]
    y_data = df_temp_Xy.loc[:, df_temp_Xy.columns.str.contains("WS ")]

    return X_data, y_data
```
Listing 2: extract_data.py