Report of project
“Price prediction for Turbine(ex works) on randomly dataset using DataIKU”

organization «General Electric Renewable Energy»

Performed by:
signature Bogdan Gorelkin
«___» _____2021 y.

For information:
GE
signature Toth Marton
«___»_____2021 y.

For information:
GE
signature Ruslan Galimov
«___»_____2021 y.

Belfort 2021

Table of contents

The purpose of this work

The main goal of this work is to build model in Dataiku Data Science Studio (DSS) which is able to predict output based on input data. Dataiku is the platform democratizing access to data and enabling enterprises to build their own path to AI in a human-centric way. The present report results in comparison of two graphs, presenting original data and predicted data.

The work was performed in a free version of the DSS installed under the control of the operating system ubuntu and supporting up to 3 users. The data for analysis was prepared using the Python programming language in Google Colab.

1. Introduction

Being able to accurately predict the production of an organization's products is a difficult, but quite doable task. If a company has an understanding of the products of tomorrow, it remains a competitive organization.

The complexity of the task lies not only in the implementation of the model, but also in maintaining the confidentiality of the data. In addition, it is important to distribute tasks among employees and prevent data leakage. A reasonable solution would be to use one tool that is able to store data, analyze, make predictions and at the same time control the access of each employee, allowing you to solve subtasks to achieve a common goal.

DataIKU has extensive functionality and is able to solve all these difficulties. Creating a demo model showing some of the capabilities of the DSS requires:

1. Create a dataset that is theoretically similar to the real one.
2. Install the DataIKU platform.
3. Import data to the local platform server.
4. Create a model and train it on the imported dataset.
5. Perform model evaluation on data loaded from outside.

2. Creating a dataset

The dataset consists of three parts, which are created separately according to the conditions of *Tables 2.1, 2.2 and 2.3*. It was decided to create a fake dataset using the Python programming language in the google collab environment.

Table 2.1 – Dataset part one

	Example, kE/1 turbine (baseData)	Min Max Range
Turbine EXW	2000 kE	40%
Transport	1000 kE	40%
Customizations	1000 kE	40%
Hub activities	500 kE	40%
Hub rental	500 kE	40%
Installation	500 kE	40%
Commissioning	700 kE	40%
Commissioning vessel	1000 kE	40%
Installation vessel	700 kE	40%

Table 2.2 – Dataset part two

	Project launch year NTP	Installation year	Distance from shore	Distance from hub	Number of units	Site area	Weather	Losses of production
Example (baseData)	2021	2023	30km	40km	46 turbines	100km	60	14%
Min Max Range	Up to 2025	Up to 2030	0-150	0-150	10 to 300	30 to 300	1 to 100	10% to 20%

Table 2.1 – Dataset part three

	Example (baseData)	Min Max Range
Project launch year NTP	2021	Up to 2025
Installation yea	2023	Up to 2030
Distance from shore	30km	0-150
Distance from hub	40km	0-150
Number of units	46 turbines	10 to 300
Site area	100km	30 to 300
Weather	60	1 to 100
Losses of production	14%	10% to 20%

To dynamically generate data, we will create a function called *dataGenerator*, which we will use to create the entire dataset.

```
1. import numpy as np
2. import random
3. import matplotlib.pyplot as plt
4. plt.style.use('fivethirtyeight')
5. noise = 5
6.
```

```
7. def dataGenerator(minValue,maxValue,baseData,size,type):
8.   dataset = []
9.   if type == '%':
10.       for i in range(0,size):
11.         my_randoms = round(random.uniform(minValue, maxValue),2)
12.         percent = my_randoms
13.         datainp = round((baseData * (percent/100)),2)
14.         data = baseData + datainp
15.         dataset.append(data)
16.       return dataset
17.     if type == 'other':
18.       for i in range(0,size):
19.         my_randoms = round(random.uniform(minValue, maxValue),2)
20.         data = baseData + my_randoms
21.         dataset.append(data)
22.       return dataset
23.     if type == 'int':
24.       for i in range(0,size):
25.         my_randoms = random.randint(minValue, maxValue)
26.         data = baseData + my_randoms
27.         dataset.append(data)
28.       return dataset
29.     if type == 'seq %':
30.       startData = baseData+(baseData/100*minValue)
31.       lastData = baseData+(baseData/100*maxValue)
32.       deltaData = lastData-startData
33.       trendSumm = deltaData/size
34.       seqData = startData
35.       for i in range(1,size+1):
36.         my_randoms = round(random.uniform(-noise, noise),2)
37.         percent = my_randoms
38.         datainp = round((seqData * (percent/100)),2)
39.         #seqData = seqData+trendSumm*i # ±5%
40.         data = (seqData + datainp)+trendSumm*i
41.         dataset.append(data)
42.       return dataset
43.     if type == 'GAUSS':
44.       if abs(minValue)==abs(maxValue):
45.         deviation = abs(baseData/100*minValue)
46.         dataset = np.random.normal(baseData,deviation/3,size)
47.         dataset = sorted(dataset)
48.       else:
49.         print("your deviation of the base isn't same (minValue != m
    axValue)")
50.       return dataset
51.     else:
52.       print("wrong type to generate the data")
```

2.1. Generating part one of the dataset

Following Pareto's law, was created the first part of data using a Gaussian distribution (*Figure 2.1*).

```
1. size = 1000
2. Turbine_EXW = np.array(dataGenerator(-40,40,20000,size,'GAUSS'))
3. Transport = np.array(dataGenerator(-40,40,1000,size,'GAUSS'))
4. Customizations = np.array(dataGenerator(-40,40,1000,size,'GAUSS'))
5. Hub_activities = np.array(dataGenerator(-40,40,500,size,'GAUSS'))
6. Hub_rental = np.array(dataGenerator(-40,40,500,size,'GAUSS'))
7. Installation = np.array(dataGenerator(-40,40,500,size,'GAUSS'))
8. Commisioning = np.array(dataGenerator(-40,40,700,size,'GAUSS'))
9. Commisioning_vessel = np.array(dataGenerator(-
   40,40,1000,size,'GAUSS'))
10.   Installation_vessel = np.array(dataGenerator(-
   40,40,700,size,'GAUSS'))
```

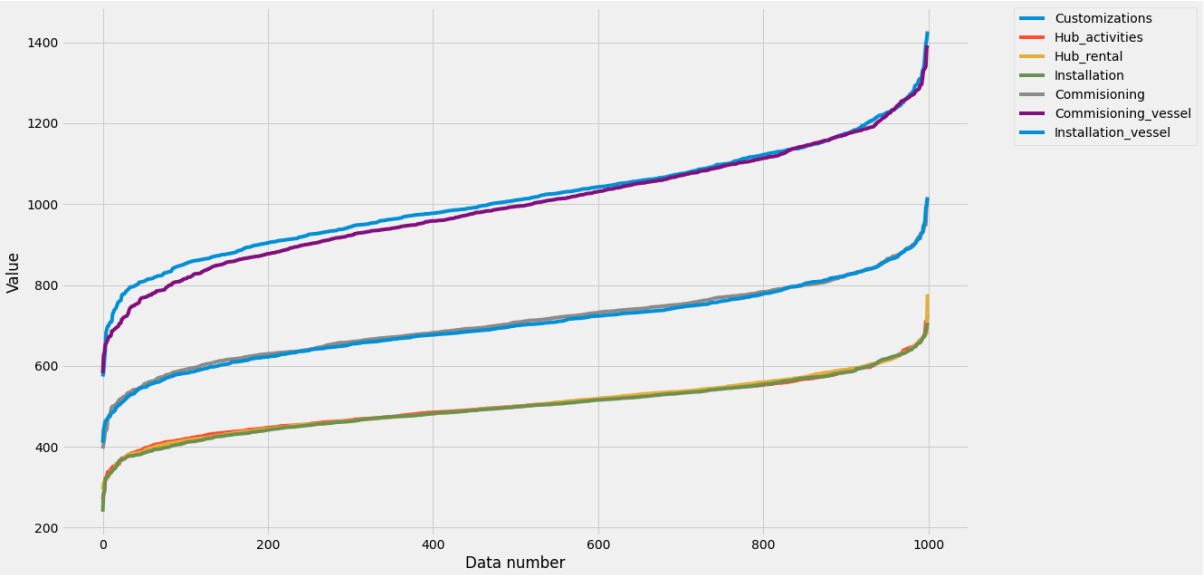

Figure 2.1 – First part of data

2.2. Generating part two of the dataset

Since the data in Table 2.2 is more random, the dataset is created using a continuous uniform distribution. The generation result is shown in (*Figure 2.2).*

```
1. Wind_speed = np.array(dataGenerator(-2,2,10,size,'other'))
2. Air_density = np.array(dataGenerator(1.1,1.3,1.12,size,'other'))
3. Water_depth = np.array(dataGenerator(0,60,24,size,'other'))
4. chosen_idx = np.random.choice(192, replace = True, size = size)
5. counties = countries_csv.iloc[chosen_idx]
6. counties = np.array(counties['alpha2'])
7. Inflation = np.array(dataGenerator(1,5.5,2,size,'other')) #For infla
   tion data in percentage
```

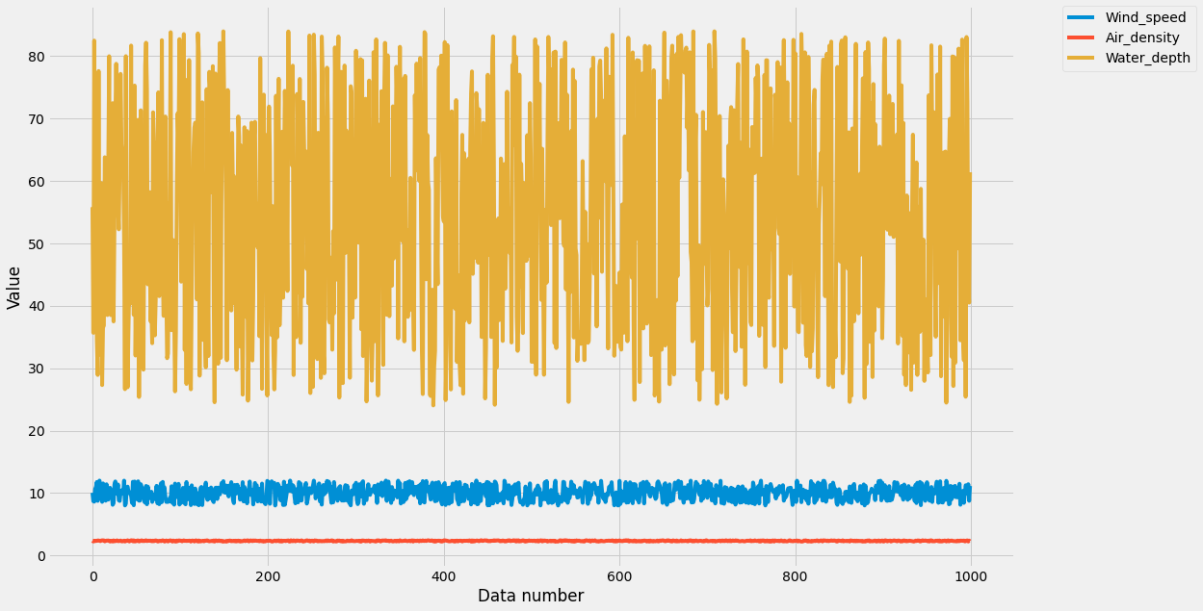

Figure 2.2 – Second part of data

2.3. Breadth-First Spanning Tree Construction

The dataset for the third part is generated according to laws similar to the previous section *(Figure 2.3)*.

```
1. Project_launch_year_NTP=np.array(dataGenerator(0,4,2021,size,'int'))
2. Installation_year = np.array(dataGenerator(0,7,2023,size,'int'))
3. Distance_from_shore = np.array(dataGenerator(0,150,30,size,'other'))
4. Distance_from_hub = np.array(dataGenerator(0,150,40,size,'other'))
5. Number_of_units = np.array(dataGenerator(10,300,0,size,'int'))
6. Site_area = np.array(dataGenerator(30,300,0,size,'int'))
7. Weather = np.array(dataGenerator(1,100,0,size,'int'))
8. Losses_of_production = np.array(dataGenerator(10,20,0,size,'int')
9. Equity_share = np.array(dataGenerator(30,100,0,size,'int'))
```

Figure 2.3 – Third part of data

3. DataIKU

Installation of DSS on Windows is possible only using a virtual machine, therefore the installation is performed under the control of the Ubuntu operating system, using the dual boot.

After successfully installing the DSS platform, you need to start the localhost *(Figure 3.1)*. Now you can connect to the platform by entering the server IP address and port number into the line of any supporting browser. You can also connect to the server from any other device that is on the same network as the server. In the future, access to the server can be organized from the outside

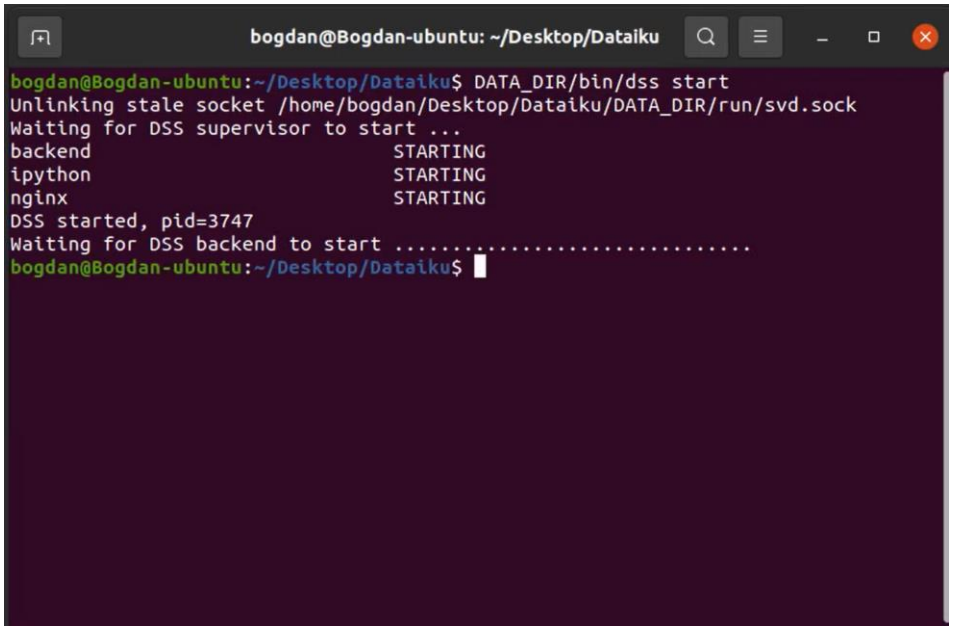

Figure 3.1 – Starting of localhost

Uploading data to the DSS platform can be done in many ways. After this point is completed, you can start analyzing the data. When building a model, in addition to choosing and setting up a predictive algorithm, you can determine on which data the predicted value depends. *Figure 3.2* shows what the "Turbine_EXW" parameter depends on. It is for him that we will train our model.

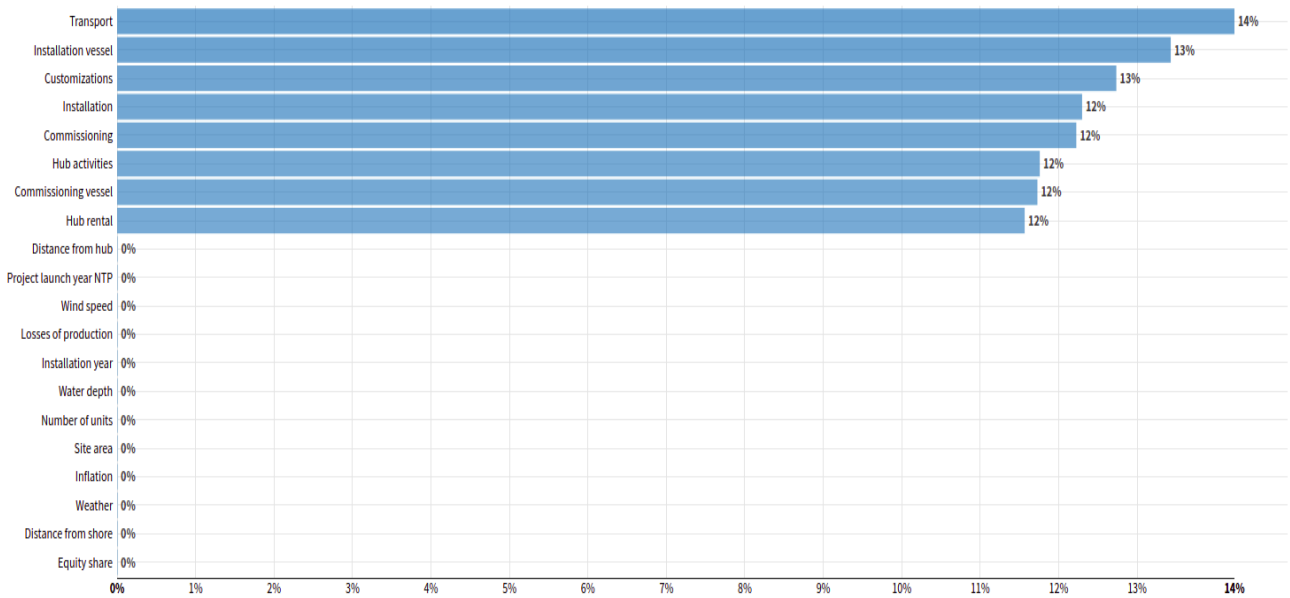

Figure 3.2 – Variables importance

At this stage, everything is ready for building and training the model *(Figure 3.3)*. Training takes place on data from the "training_dataset" block. The area highlighted in red is only needed to assess the prediction quality.

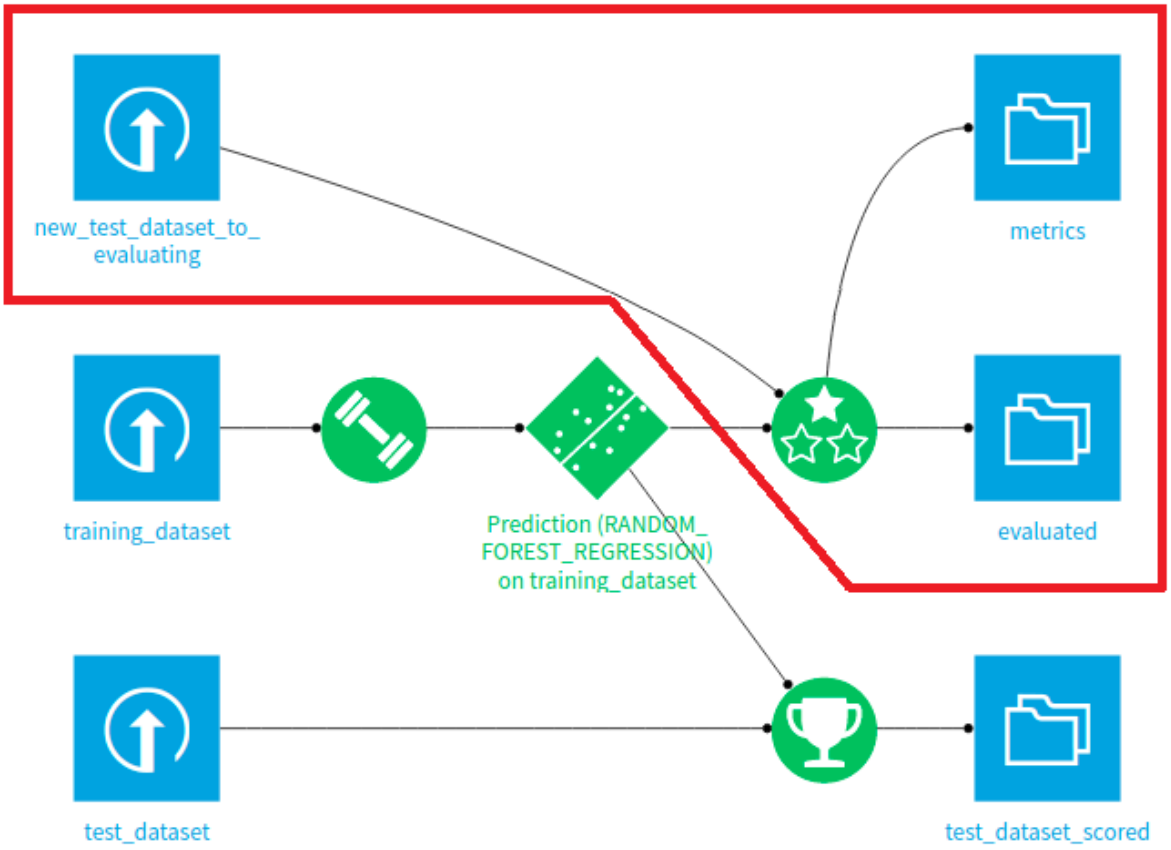

Figure 3.3 – model in Flow

In addition to the predictor, the input to the evaluator is data other than the training set. At the output, a new set of forecast data is created, on the basis of which a graph is built showing the input data and the forecast *(Figure 3.4). Table 3.1* shows the performance indicators of the predictions obtained.

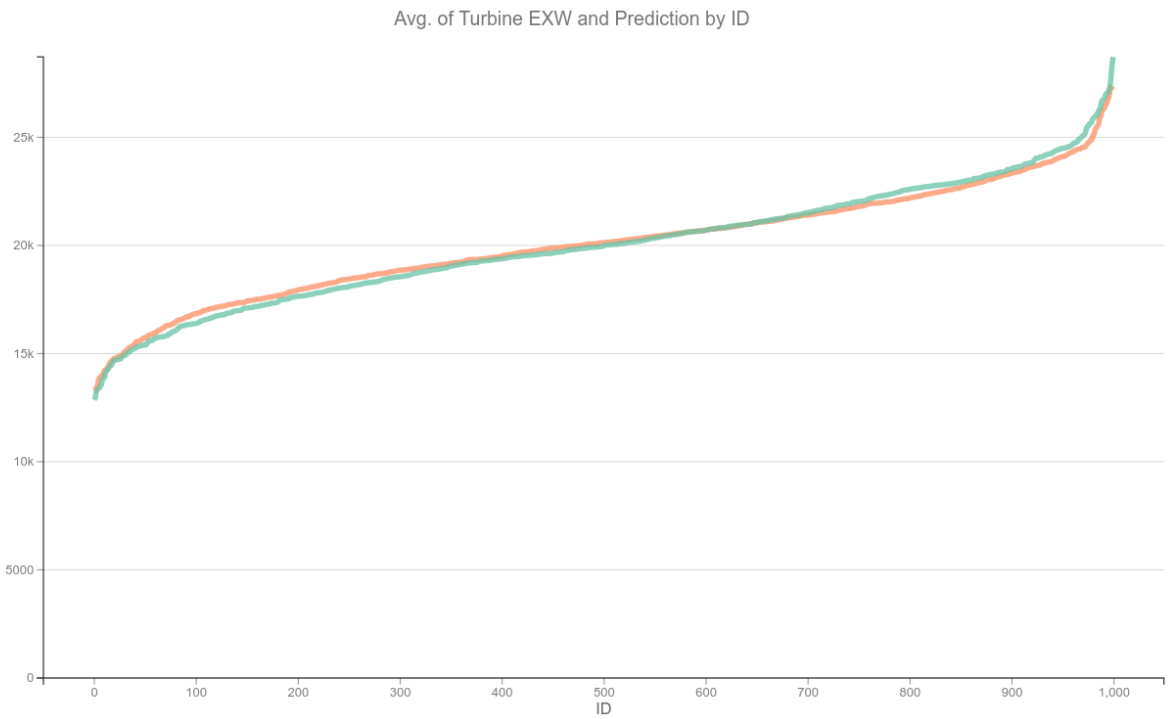

Figure 3.4 – Comparison of original and predicted data

Table 3.1 – Metrics

evs	mae	mse	mape	rmse	rmsle	r2	pearson
0.99	233.92	74679.59	0.012	273.27	0.014	0.99	0.99

Also, an editable dataset has been added to this model, which is connected to the scorer input. It is enough to enter the known data into the "test_data" block and the scorer will give a recommendation for "Turbine_EXW" based on the trained model.