Name: **Sabyr Kabylbek**

Batch code: LISUM01

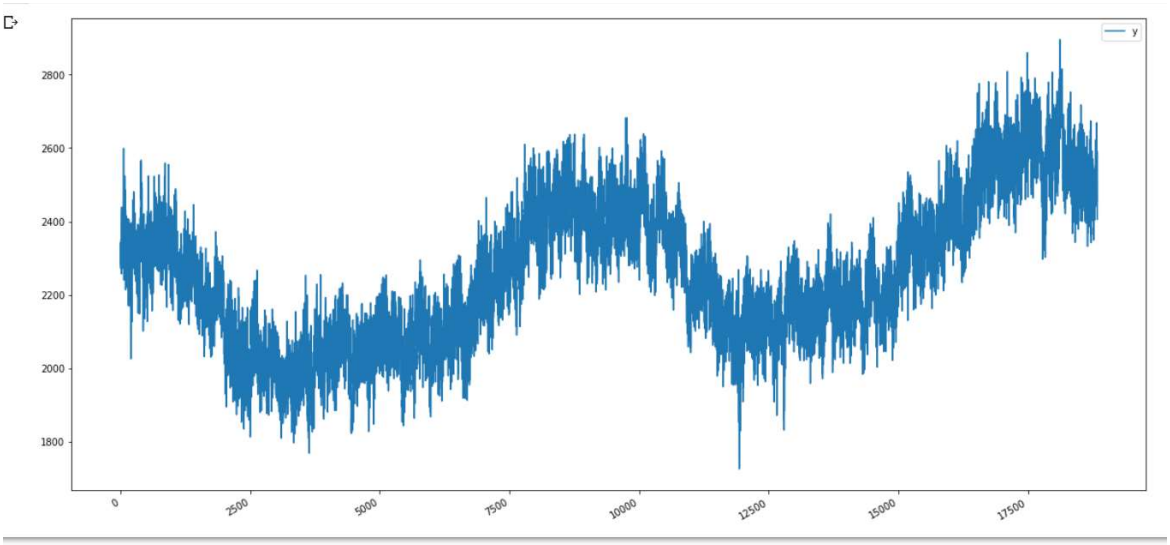
Submission Date: 04.07.2021

Submitted to: Data Glacier

1. The toy data selected.

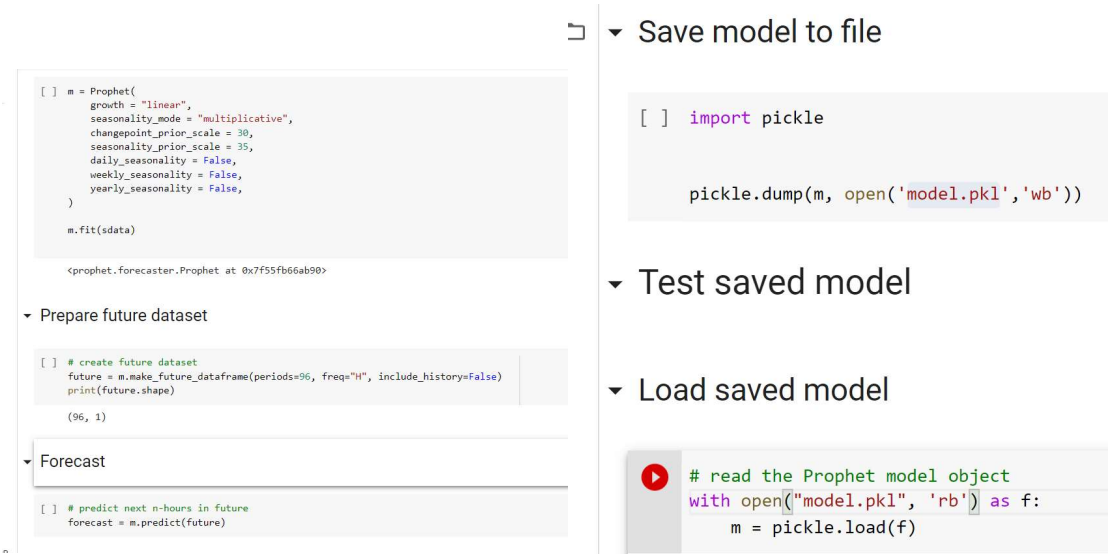
The toy data selected is the electricity consumption of a city in Kazakhstan between 2019 and 2021, which I have already analyzed at my work (I am a Data Scientist in local company).

	y	tempC	FeelsLikeC	ds	dayofweek	quarter	month	dayofmonth	dayofyear	weekofyear	hour	isholidays
0	2286.0	-19	-27	2019-01-01 00:00:00	1	1	1	1	1	1	0	1
1	2297.0	-20	-28	2019-01-01 01:00:00	1	1	1	1	1	1	1	1
2	2300.0	-20	-29	2019-01-01 02:00:00	1	1	1	1	1	1	2	1
3	2342.0	-21	-29	2019-01-01 03:00:00	1	1	1	1	1	1	3	1
4	2271.0	-21	-29	2019-01-01 04:00:00	1	1	1	1	1	1	4	1



For this assignment I have used FBprophet model to make a prediction for TIME-SERIES data. I have dropped all columns except “ds” and “y” – which is necessary for Fbprophet model training. “ds” is date time and “y” is electricity consumption per hour

2. Saving the Model



```
[ ] m = Prophet(
        growth = "linear",
        seasonality_mode = "multiplicative",
        changepoint_prior_scale = 30,
        seasonality_prior_scale = 35,
        daily_seasonality = False,
        weekly_seasonality = False,
        yearly_seasonality = False,
    )

    m.fit(sdata)

    <prophet.forecaster.Prophet at 0x7f55fb66ab90>

Prepare future dataset

[ ] # create future dataset
    future = m.make_future_dataframe(periods=96, freq="H", include_history=False)
    print(future.shape)

    (96, 1)

Forecast

[ ] # predict next n-hours in future
    forecast = m.predict(future)
```

```
Save model to file

[ ] import pickle

    pickle.dump(m, open('model.pkl','wb'))

Test saved model

Load saved model

# read the Prophet model object
with open("model.pkl", 'rb') as f:
    m = pickle.load(f)
```

3. Deployment Process

Firstly, I have created python file in order to apply Flask library for my saved model:

```python
import numpy as np
from flask import Flask, request, jsonify, render_template
import pickle
import pandas as pd
app = Flask(__name__)
model = pickle.load(open('model.pkl', 'rb'))

@app.route('/')
def home():
    return render_template('index.html')

@app.route('/predict',methods=['POST'])
def predict():

    horizon = [int(x) for x in request.form.values()]
    future = model.make_future_dataframe(periods = horizon[0])

    prediction = model.predict(future)

    output = prediction['yhat'][horizon[0]]

    return render_template('index.html', prediction_text='Energy should be MW {}'.format(output))

if __name__ == "__main__":
    app.run(debug=True)
```

Secondly, I have created index.html and style.css files so it can be integrated flask python file:

```html
<!DOCTYPE html>
<html >
<head>
  <meta charset="UTF-8">
  <title>Deployment Flask</title>
  <link href='https://fonts.googleapis.com/css?family=Pacifico' rel='stylesheet' type='text/css'>
<link href='https://fonts.googleapis.com/css?family=Arimo' rel='stylesheet' type='text/css'>
<link href='https://fonts.googleapis.com/css?family=Hind:300' rel='stylesheet' type='text/css'>
<link href='https://fonts.googleapis.com/css?family=Open+Sans+Condensed:300' rel='stylesheet' type='text/css'>
<link rel="stylesheet" href="{{ url_for('static', filename='css/style.css') }}">

</head>

<body style="background: #000;">
 <div class="login">
    <h1>Energy Forecasting</h1>

    <!-- Main Input For Receiving Query to our ML -->
    <form action="{{ url_for('predict')}}"method="post">
        <input type="text" name="future_step" placeholder="future_step" required="required" />

        <button type="submit" class="btn btn-primary btn-block btn-large">Predicted Value</button>
    </form>

   <br>
   <br>
   {{ prediction_text }}

 </div>
</body>
</html>
```

```css
* { -webkit-box-sizing:border-box; -moz-box-sizing:border-box; -ms-box-sizing:border-box; -o-box-sizing:border-box; box-sizing:border-box; }

html { width: 100%; height:100%; overflow:hidden; }

body {
        width: 100%;
        height:100%;
        font-family: 'Helvetica';
        background: #000;
        color: #fff;
        font-size: 24px;
        text-align:center;
        letter-spacing:1.4px;

}
.login {
        position: absolute;
        top: 40%;
        left: 50%;
        margin: -150px 0 0 -150px;
        width:400px;
        height:400px;
}

.login h1 { color: #fff; text-shadow: 0 0 10px rgba(0,0,0,0.3); letter-spacing:1px; text-align:center; }

input {
        width: 100%;
        margin-bottom: 10px;
        background: rgba(0,0,0,0.3);
        border: none;
        outline: none;
        padding: 10px;
        font-size: 13px;
        color: #fff;
        text-shadow: 1px 1px 1px rgba(0,0,0,0.3);
        border: 1px solid rgba(0,0,0,0.3);
        border-radius: 4px;
        box-shadow: inset 0 -5px 45px rgba(100,100,100,0.2), 0 1px 1px rgba(255,255,255,0.2);
        -webkit-transition: box-shadow .5s ease;
        -moz-transition: box-shadow .5s ease;
        -o-transition: box-shadow .5s ease;
        -ms-transition: box-shadow .5s ease;
        transition: box-shadow .5s ease;
}
input:focus { box-shadow: inset 0 -5px 45px rgba(100,100,100,0.4), 0 1px 1px rgba(255,255,255,0.2); }
```

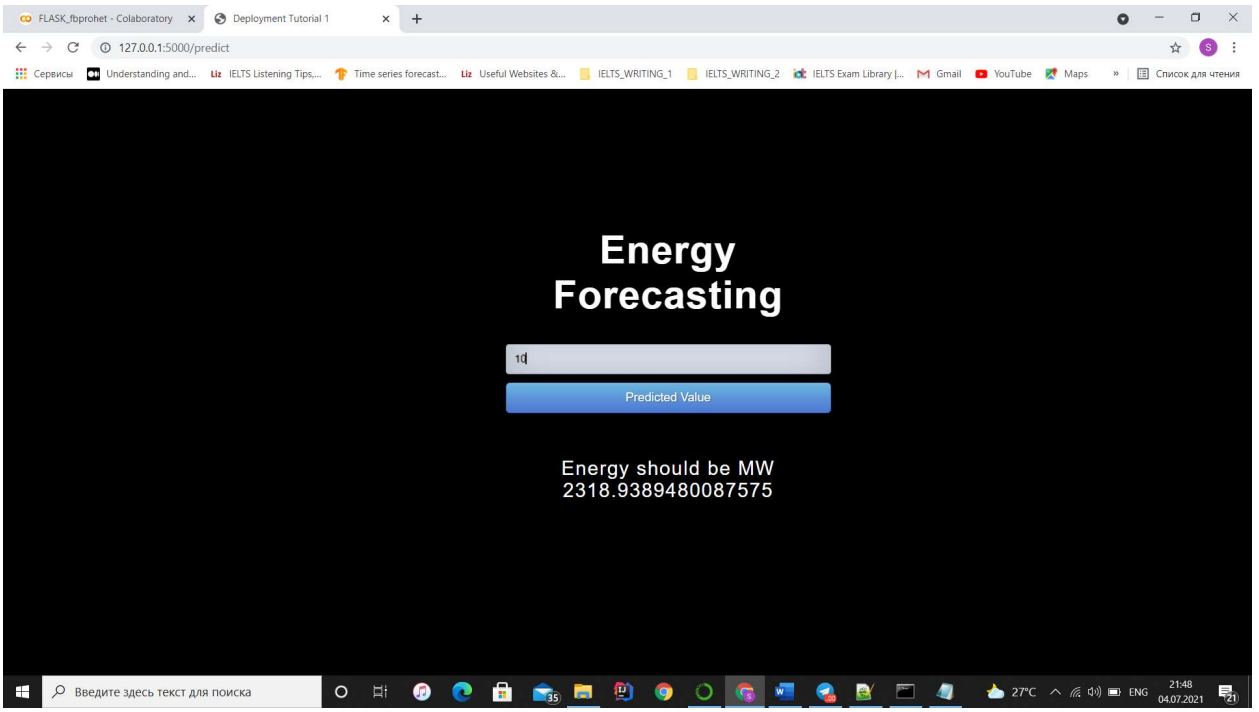
Thirdly, the request python file was created to integrate flask python file and API's

```
import requests

url = 'http://localhost:5000/results'
r = requests.post(url,json={'future_step':2005})

print(r.json())
```

The result of the deployment process:



In my prediction I ask user to enter the step for future prediction, so after getting the future steps from the user, the model provides the predicted value at that step of future.