# Python for data analysis

DELOUPY GREGOIRE DIA2

DENG MUCHAN DIA2

# Introduction:

The goal of this project is to analyze the data set that has been assigned to us in order to show the link between the different variables in our problem.

We decided to work in pairs on the same data set in order to take the one we were most interested in.

We obtained a data set about bike rental in Seoul, South Korea, accessible on the UCI website which is specialized in sharing many datasets in various categories.

# Description of the Dataset

Our Dataset contains information on the number of public bicycles rented per hour from December 1, 2017 until November 31, 2018.

Composed of 8760 instances and 14 attributes, this dataset offers us information about the weather in Seoul, the number of bikes rented and the different periods of the year (seasons, holydays) allowing us to predict the number of bikes needed at each hour of the day.

The dataset has been put online on a South Korean site with for objective of sharing public information. This site is writing in Korean, it was very difficult for us to understand it.

This platform allows all citizens of Seoul to obtain public information in various fields such as industry, economy, health...

# Problématique

The question we wish to answer is therefore :

How many bikes do we need to answer to the demand at T time ?

Seoul is the capital of South Korea; it is also the largest city in Korea with 11 million inhabitants (which is equal to half of the population residing in South Korea).

For means of transportation, the city has 2 airports, and it is mainly by metro that the population moves because it is fast and efficient.

However, given the number of people living in Seoul, the Korean government is making more and more efforts to encourage Seoulites to travel by bicycle, including the construction of more and more bicycle paths. It is therefore important to provide information on the number of bicycles rented.

# Partie 1 : Data-Visualization

In this section we first asked ourselves what factors impact the number of bike rentals.

We have a very small number of attributes (14) so we decided to graphically represent the potential linkage of all these columns in two subparts.

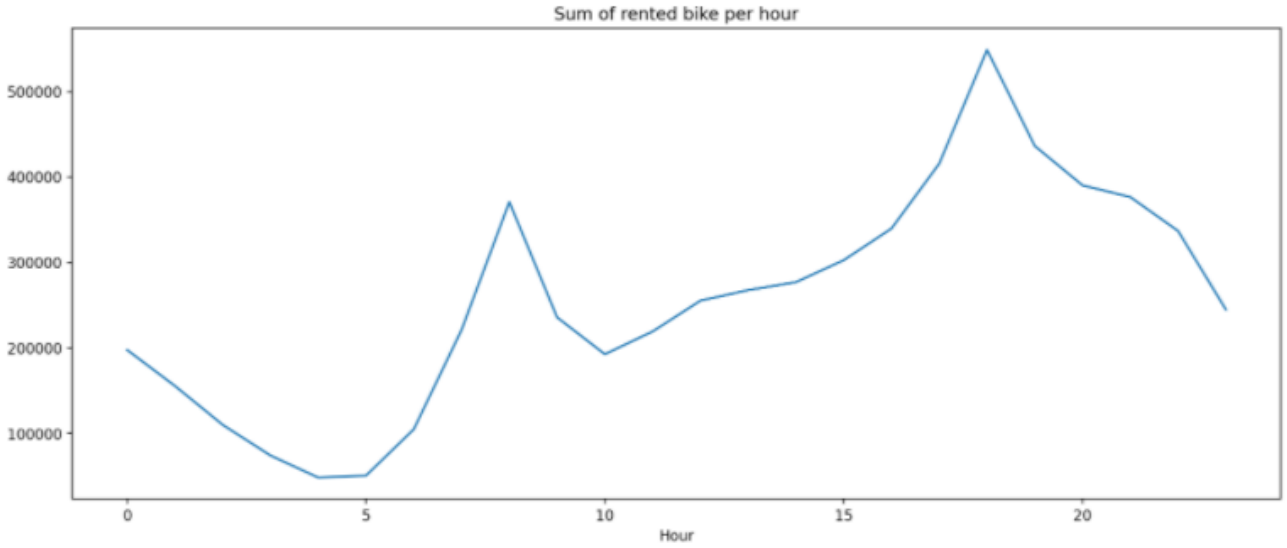
# Partie 1.1: Donnés temporelles

We are interested here in the Seasons, Dates and Hour columns in order to represent the number of bike rentals over time.

We soon realized that all these columns had an impact on the number of bikes rented.

Let's take the examples of hours. We noticed that at some hours, the number of rentals was more important than others.

Here is the graph obtained thanks to our variable 'location_hour' which returns the total number of bikes rented at each hour of the day:



When we are looking at this graph, the first thing to ask is: What are the causes of these differences?

First of all, we can observe a fall from 20h to 00h due to the fact that people are at home having dinner with their families and then sleep until the morning.

In addition, a strong growth can be observed from 5 to 9 am because this is the time when people leave for work.

Finally, from 10am to 7pm the population uses bicycles randomly in order to move around, that’s why its increasing.

Here is the list of variables we have created for this part:

**-Total_location_mois:** Who returns the number of bikes rented per month. We have noticed that the lowest months in terms of rentals are December, January, February (probably due to the weather conditions we will see later...).

**-Moyenne_location_mois:** Does the same as the above variable but as an average, not as a total.

**-Total _location_saisons:** Returns the sum of the number of rentals according to the 4 seasons.

**-Location_Heure**: Explanation on the left

**-Location_vacance:** Total number of rentals if it's vacation or not

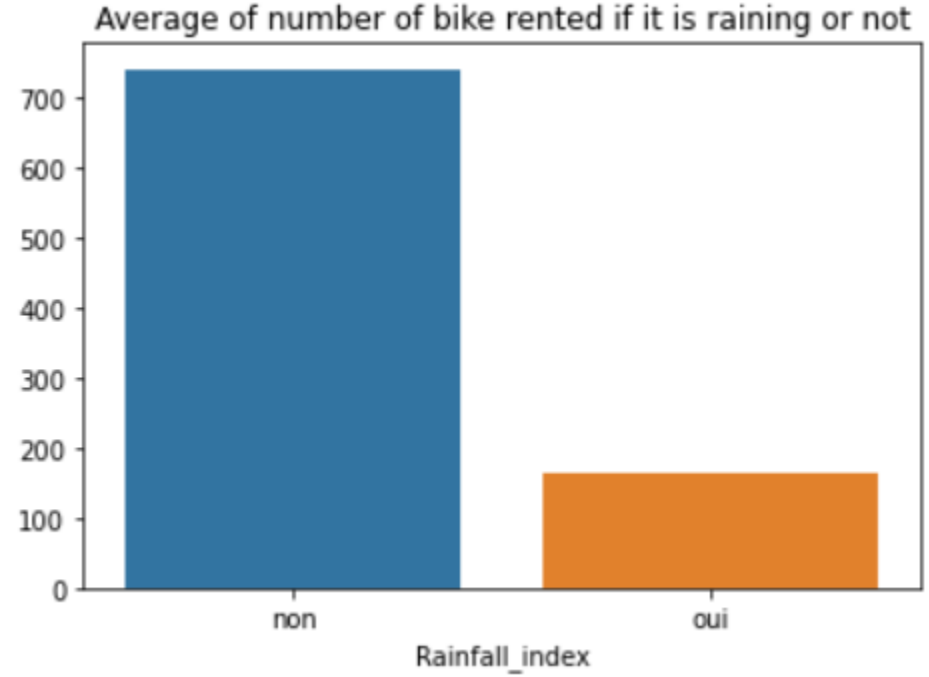
# Part 1.2: Meteorological visualizations

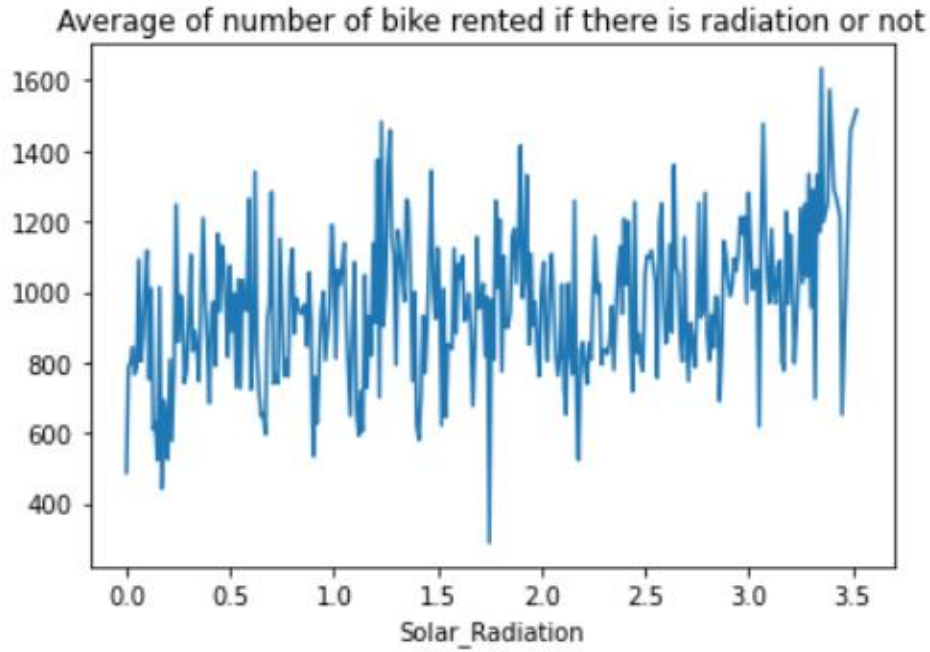
In this section we are interested in columns related to the weather. We used the columns Wind_speed, Humidity, Temperature, Snowfall, Rainfall and Solar_Radiation.

Some of these columns have a greater influence than others on the number of bikes rented.

For example, the **Rainfall column** has a much more important impact than the **Solar_radiation column** because if it rains the population will use less bikes while solar radiation is not taken into account.

We have created graphs that show the difference in the impact of these two columns on the number of bikes rented.




We can clearly see the difference in the impact of these two graphs. The solar radiation graph does not change as the radiation increases.

Unlike the rain graph where you can see a big difference between renting a bike depending on whether it's raining ("yes") or not ("no").

Here are the different variables we have created:

**Total_location_vent:** Represents the total number of rentals bike as a function of the wind speed in m/s.

**Mean_location_vent:** Represents the average bike rental as a function of wind power in m/s

**Humidity_location:** Represents average bike rental as a function of humidity in percent

**Temperature_location:** Represents the average bike rental as a function of temperature in degrees

**Moyenne_location_Snow:** Represents the average bike rental based on the fact there is or not snow on the ground.

**Moyenne_location_Rainfall:** Represents the average bike rental based on the fact there is or not rain.

**Moyenne_location_radiation:** Represents the average bike rental as a function of the value of solar radiation present

We also created a new data_frame (named data_fram) in order to don't change or created column on the velo dataframe.
Here are the 2 columns that we have created inside:

**-Snow_index:** Return 'oui' if it rains or 'non' otherwise

**-Rainfall_index:** Return 'oui' if it snow or 'non' otherwise

# Partie 2:

# Modelisation

In this part we used 3 prediction models in order to predict as accurately as possible the number of bikes rented each hour.

The 3 models are:

- KNN model
- RandomForest model
- GradientBoosting model

# KNN Model

KNN model is a very basic model because it's simple and easily explainable.

We have created the findBestParametersKNN() function that scans the grid several times in order to find the best possible accuracy value.

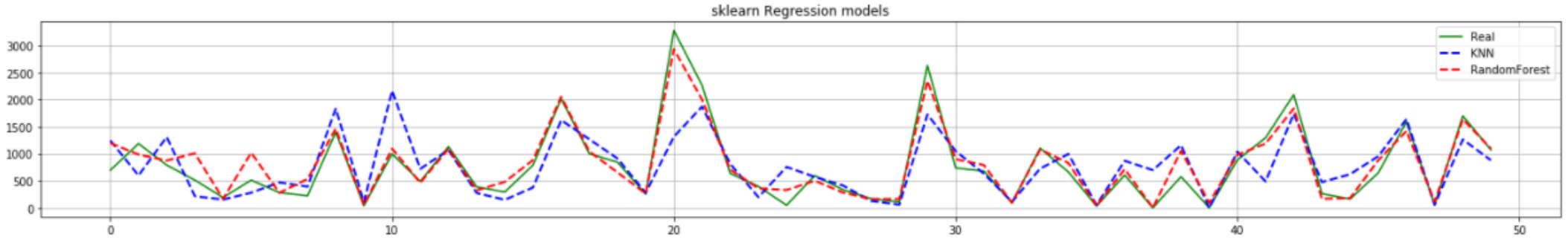
We obtained the best parameters with GridSearch, which k = 3 (very usual value).
The accuracy of the model is 0.765 on the test set. Here is the graph:


It's not very precise, let's try another model.

# The RandomForest Model

Then we selected the RandomForest model which is very stable and has the power of handle large data sets with higher dimensionality, despite his training time wich is a too long.

The methode GridSearchCV has been used here to find the best parameters of this RandomForest model. We obtained a 0.922 accuracy.
Let's visualize it by plotting it on a graph compare to the previous KNN predictions:

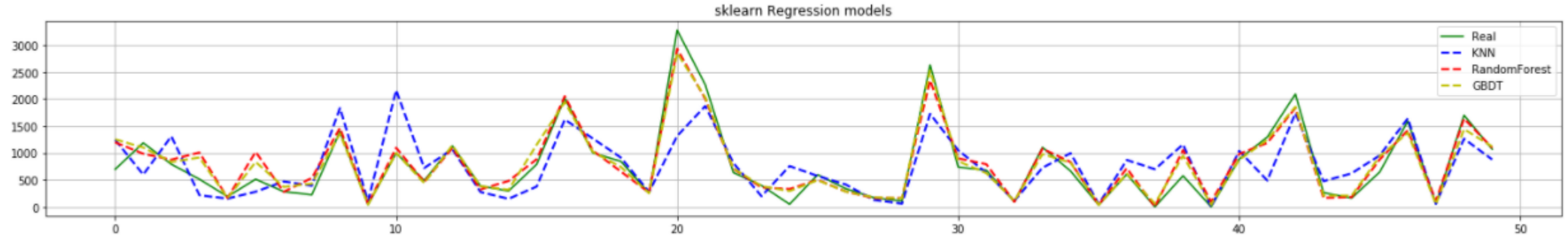



That's much better, we can see on the graph the red line representing the RandomForest model is very close to the green line symbolizing the real data, compare to the blue line which is the predictions of KNN model.

# GradientBoosting Model

GradientBoosting and RandomForest's predictions are very close from each other. It is consistent with the score on GridSearch test set with cross validation cv = 5

There is a precision of 0.938 for this model. RandomForest and Gradient Boosting are definitely better than KNN, but these two are very close in terms of prediction performance, with a slight advantage on the side of Gradient Boosting. Let's see the graph:




But the RandomForest model has much less training time than the Gradient Boosting method, so the choice between these two will be based on that according to the different situation: in case of lack of time, we will favor the Random Forest method, otherwise we prefer to use the Gradient Boosting model.

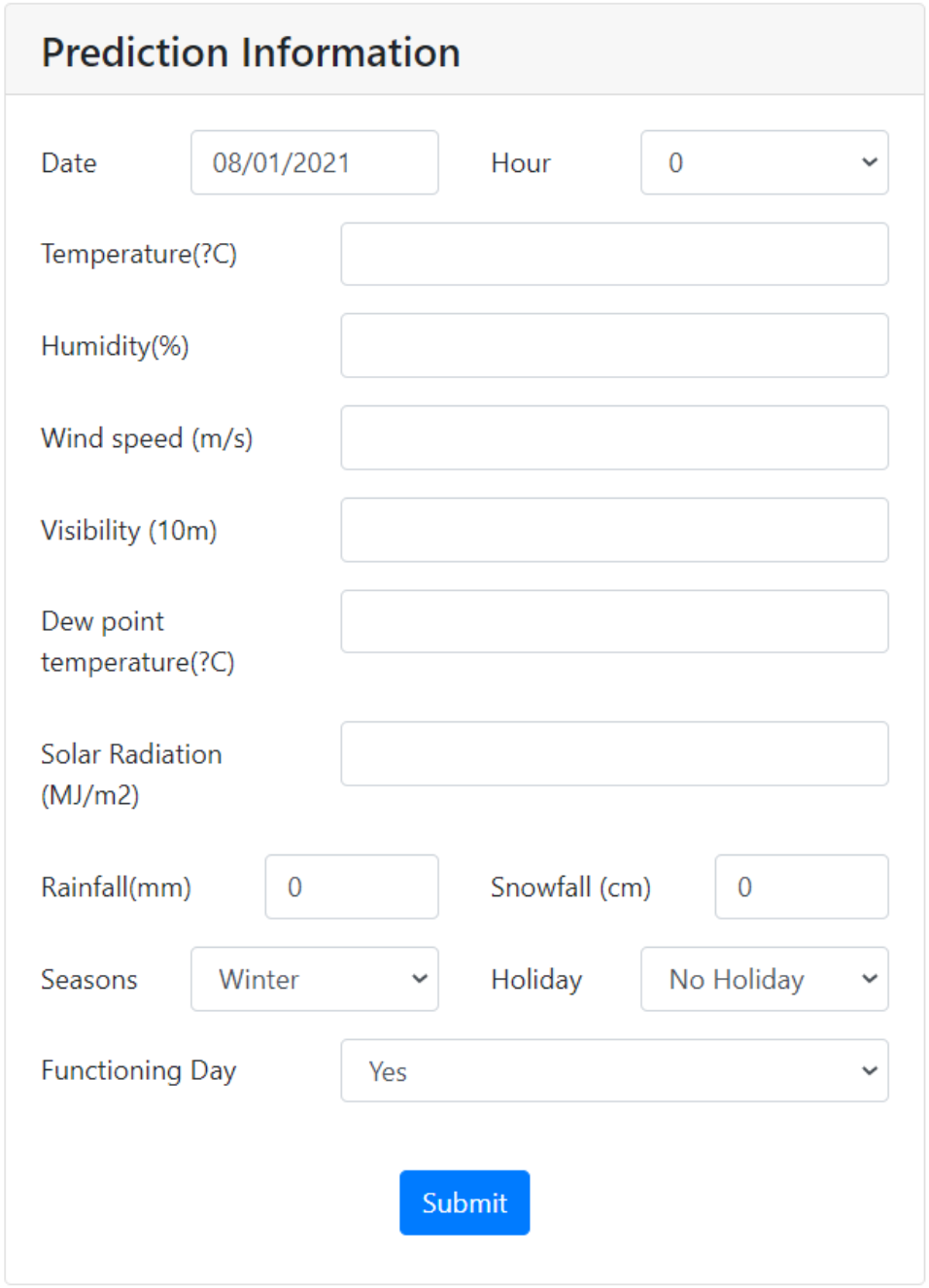
# Part 3:

# Flask Api

In this part we must make an Api to illustrate a prediction model we have chosen for our dataset.

We have chosen to use Flask which we have already seen during the year.

Combined with one of the three models with the hyperparameters we have studied in the previous section : Gradient Boosting.




# Prediction process

Now, if we want to have the prediction of the number of bikes rented.

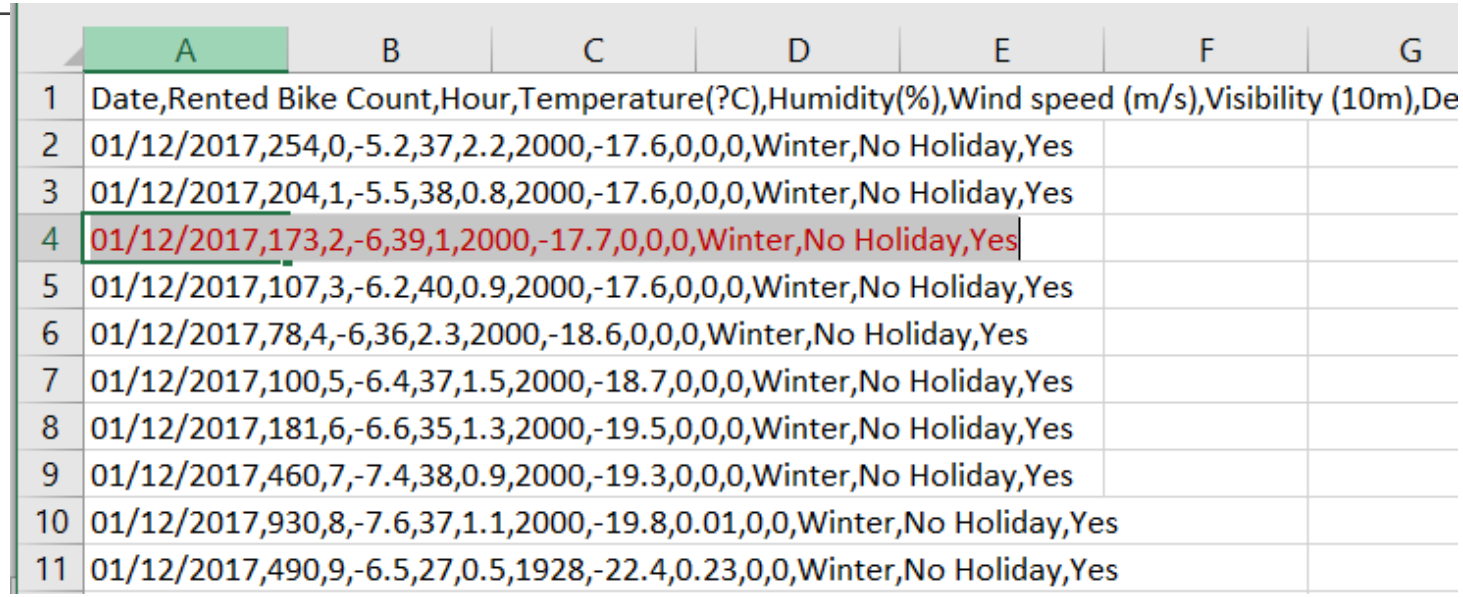
We must fill out the form correctly (with all variables according to their type) and click on “Submit” button.

Some variables are already entered by default, we can change them if we want.

# Test

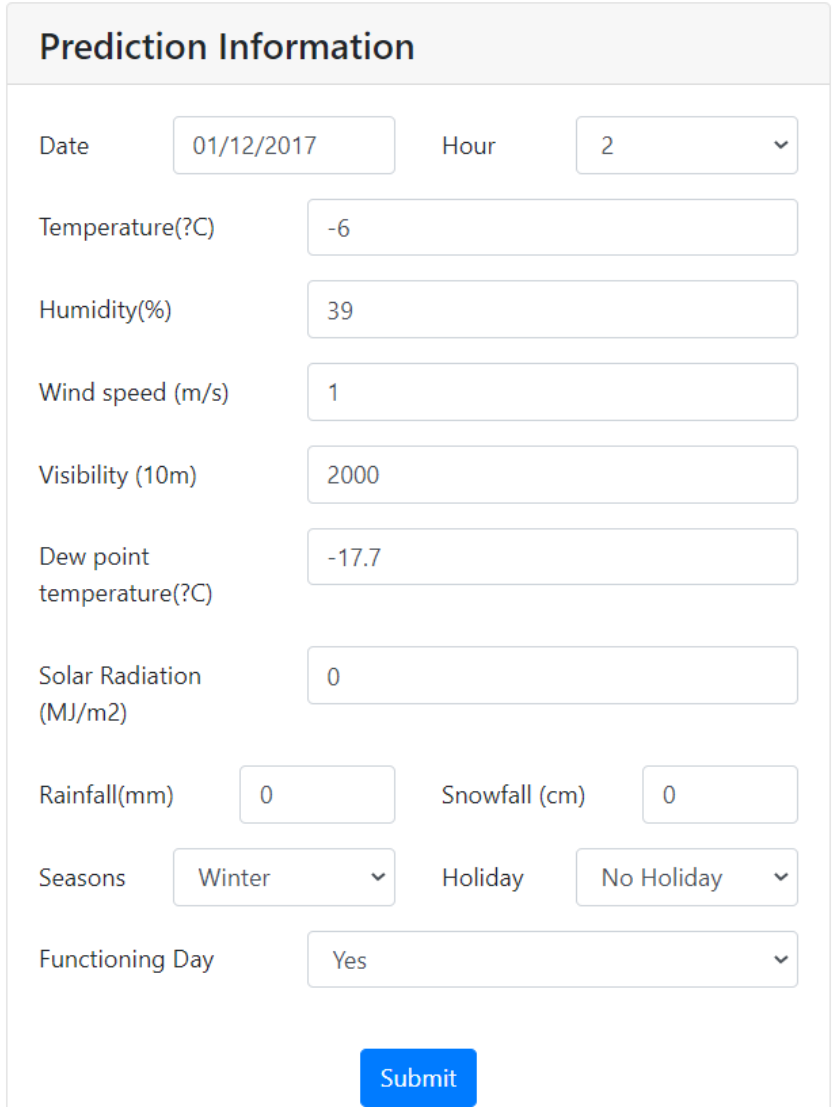
So, let's chose randomly a line from our original dataset, and take its value to put in our previous from.

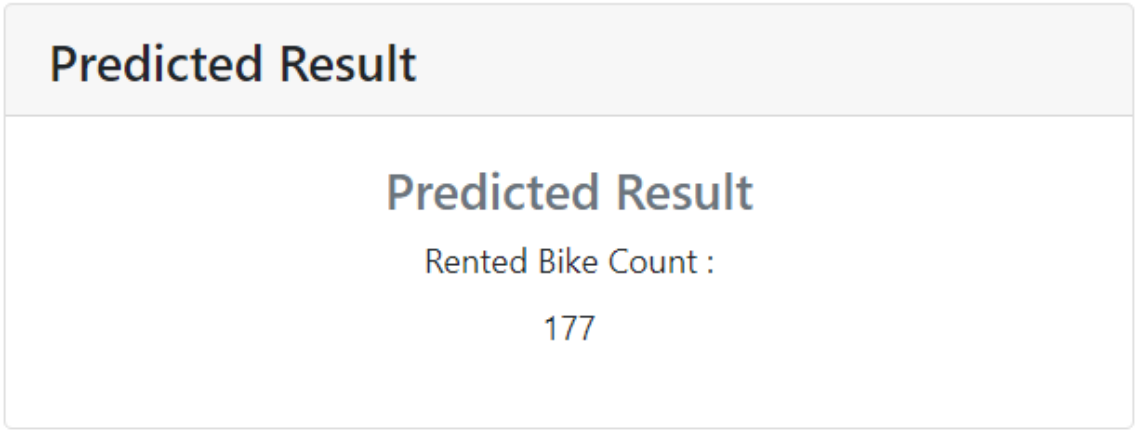
And see if we obtain a value close to 173, which is the second column of out dataset: Rented Bike Count

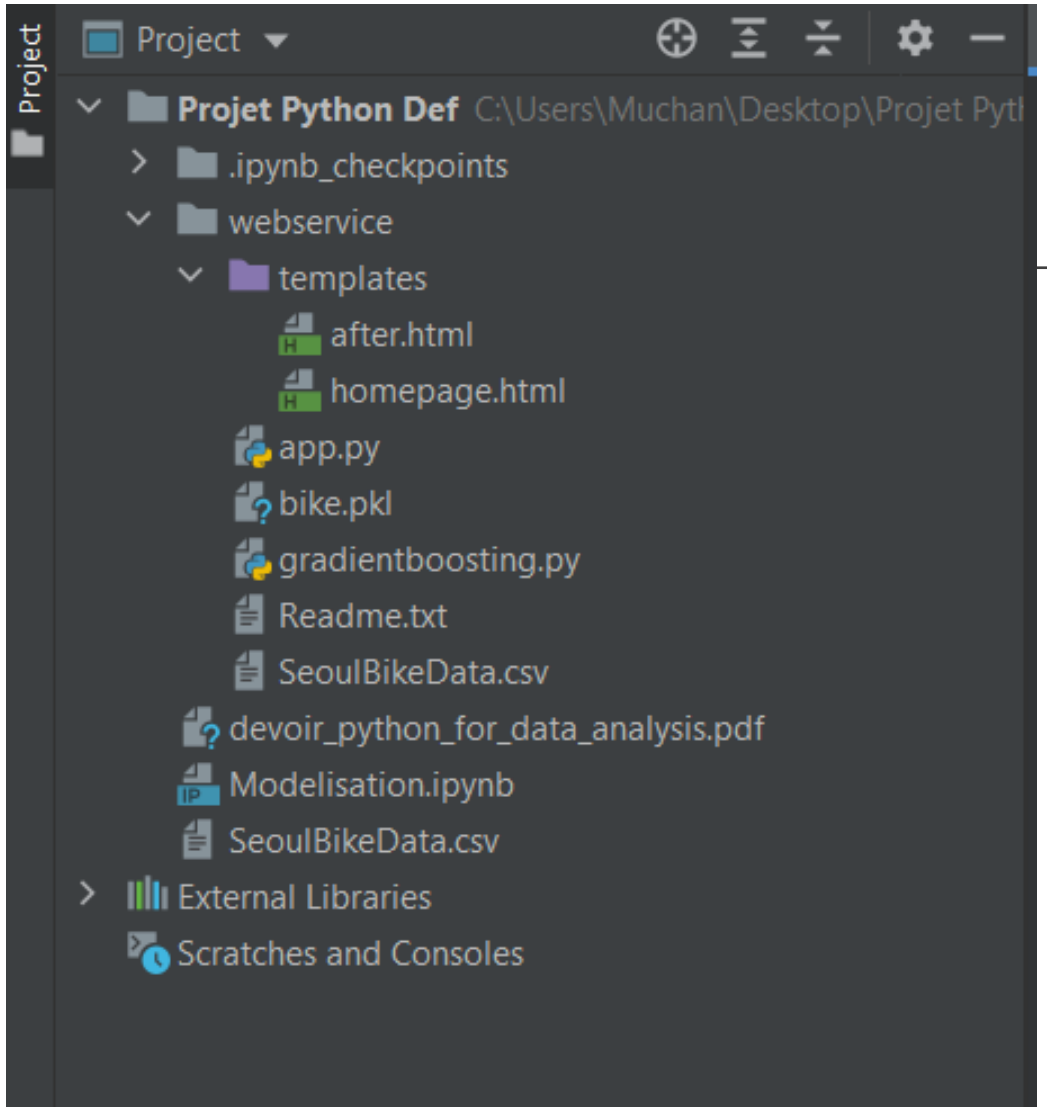


| | A | B | C | D | E | F | G |
|---|---|---|---|---|---|---|---|
| 1 | Date,Rented Bike Count,Hour,Temperature(?C),Humidity(%),Wind speed (m/s),Visibility (10m),De | | | | | | |
| 2 | 01/12/2017,254,0,-5.2,37,2.2,2000,-17.6,0,0,0,Winter,No Holiday,Yes | | | | | | |
| 3 | 01/12/2017,204,1,-5.5,38,0.8,2000,-17.6,0,0,0,Winter,No Holiday,Yes | | | | | | |
| 4 | 01/12/2017,173,2,-6,39,1,2000,-17.7,0,0,0,Winter,No Holiday,Yes | | | | | | |
| 5 | 01/12/2017,107,3,-6.2,40,0.9,2000,-17.6,0,0,0,Winter,No Holiday,Yes | | | | | | |
| 6 | 01/12/2017,78,4,-6,36,2.3,2000,-18.6,0,0,0,Winter,No Holiday,Yes | | | | | | |
| 7 | 01/12/2017,100,5,-6.4,37,1.5,2000,-18.7,0,0,0,Winter,No Holiday,Yes | | | | | | |
| 8 | 01/12/2017,181,6,-6.6,35,1.3,2000,-19.5,0,0,0,Winter,No Holiday,Yes | | | | | | |
| 9 | 01/12/2017,460,7,-7.4,38,0.9,2000,-19.3,0,0,0,Winter,No Holiday,Yes | | | | | | |
| 10 | 01/12/2017,930,8,-7.6,37,1.1,2000,-19.8,0.01,0,0,Winter,No Holiday,Yes | | | | | | |
| 11 | 01/12/2017,490,9,-6.5,27,0.5,1928,-22.4,0.23,0,0,Winter,No Holiday,Yes | | | | | | |

# Click on submit






# How did we build this Api

We used two html templates to respectively present the form and the display of the predicted result: homepage.html and after.html

app.py is the main program and the gradientboosting.py is the prediction model we have chosen

bike.pkl is a file created for the pickle module to store our prediction model which can be used later in our app.py

# Prediction Model

app.py | gradientboosting.py | homepage.html | after.html

```
47
48
49  regressor = GradientBoostingRegressor(learning_rate=0.2,
50                                        max_depth=8,
51                                        max_features=0.8,
52                                        min_samples_leaf=9,
53                                        n_estimators=80).fit(X_train, Y_train.ravel())
54
55
56  pickle.dump(regressor, open('bike.pkl', 'wb'))
57
58
```

we can see that here we used pickle to "wrap" the regressor into bike.pkl to be able to use it later in another context

We have called the bike.pkl in the main program to make prediction on the data collected by html.

And then once we launch the app.py, the html page with the form we saw before appears

app.py | gradientboosting.py | homepage.html | after.html

```
1  from flask import Flask, render_template, request
2  import pickle
3  import pandas as pd
4
5
6  model = pickle.load(open('bike.pkl','rb'))
7
8  app = Flask(__name__)
9
10 @app.route('/')
11 def main():
12     return render_template('homepage.html')
```

Run: app

```
* Serving Flask app "app" (lazy loading)
* Environment: production
  WARNING: This is a development server. Do not use it in a production deployment.
  Use a production WSGI server instead.
* Debug mode: on
* Restarting with windowsapi reloader
* Debugger is active!
* Debugger PIN: 374-861-129
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

Run | TODO | Problems | Terminal | Python Console