서울자전거
SEOUL BIKE 따릉이

PYTHON FOR DATA ANALYSIS PROJECT

SEOUL BIKE SHARING DEMAND DATA ANALYSIS

A project made by

CHMIEL Audrey and DONIER Marie
ESILV A4 - DIA1

TABLE OF CONTENTS

1. INTRODUCTION

- During this project, we had to work on a **dataset** for estimating **Seoul Bike rental demand** based on a panel of **time** and **meteorological criteria**.
- Our purpose is to **visualize** and understand the links between our **target feature (‘Rented Bike Count’)** and the other variables in order to build a **model** using **machine learning** methods **to predict the amount of bike rented in the future** according to the selected features.

2. IMPORTATION OF THE DATASET

- We first imported all the libraries we needed for our project (we kept adding more libraries to the code chunk throughout the project)

0. Importation of the libraries

```
#We first import all the libraries that we will need
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
import numpy as np
import datetime as dt
```

- … and the dataset on our notebook

1. Importation of the dataset

```
df=pd.read_csv("https://archive.ics.uci.edu/ml/machine-learning-databases/00560/SeoulBikeData.csv", encoding="latin1")
```

3. DATA EXPLORATION

We then explored the **features of our dataset**:

- **14 features and 8760 records**
- The target value is **'Rented Bike Count'**

```
df.shape
```

```
(8760, 14)
```

```
df.head()
```

	Date	Rented Bike Count	Hour	Temperature(°C)	Humidity(%)	Wind speed (m/s)	Visibility (10m)
0	01/12/2017	254	0	-5.2	37	2.2	2000
1	01/12/2017	204	1	-5.5	38	0.8	2000

The features related to…

Time periods:

- Date *(from 01/12/2017 to 30/11/2018)*
- Hour
- Seasons
- Holiday
- Functioning Day (FDay)

Weather criteria:

- Temperature (Temp)
- Humidity (Hum)
- Wind speed (Wind)
- Visibility (Vis)
- Dew point temperature (Dew)
- Solar Radiation (Solar)
- Rainfall (Rain)
- Snowfall (Snow)

3. DATA EXPLORATION

- Luckily, our dataset **did not contain any null value**
- Here are the types of each feature:
- All the columns seem **useful** to our project, so we decided not to remove any.

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8760 entries, 0 to 8759
Data columns (total 14 columns):
 #   Column                     Non-Null Count  Dtype  
---  ------                     --------------  -----  
 0   Date                       8760 non-null   object 
 1   Rented Bike Count          8760 non-null   int64  
 2   Hour                       8760 non-null   int64  
 3   Temperature(°C)            8760 non-null   float64
 4   Humidity(%)                8760 non-null   int64  
 5   Wind speed (m/s)           8760 non-null   float64
 6   Visibility (10m)           8760 non-null   int64  
 7   Dew point temperature(°C)  8760 non-null   float64
 8   Solar Radiation (MJ/m2)    8760 non-null   float64
 9   Rainfall(mm)               8760 non-null   float64
 10  Snowfall (cm)              8760 non-null   float64
 11  Seasons                    8760 non-null   object 
 12  Holiday                    8760 non-null   object 
 13  Functioning Day            8760 non-null   object 
dtypes: float64(6), int64(4), object(4)
memory usage: 958.2+ KB
```

4. DATA CLEANING

- We abbreviated and **renamed** the columns

```
df1=df.rename(columns = {'Temperature(°C)': 'Temp', 'Rented Bike Count':'Bike_Count','Humidity(%)': 'Hum',
                     'Wind speed (m/s)':'Wind','Visibility (10m)':'Vis','Dew point temperature(°C)':'Dew',
                    'Solar Radiation (MJ/m2)':'Solar','Rainfall(mm)':'Rain','Snowfall (cm)':'Snow',
                     'Functioning Day':'FDay'})
```

- We checked if there were **missing values** (there was not)

```
df1.isna().sum().sort_values(ascending=True)
```

- We **converted** the datatype of ‘Date’ column from string to datetime

```
from datetime import datetime
df1['Date'] =  pd.to_datetime(df1['Date'], format='%d/%m/%Y')
```

4. DATA CLEANING

Because we thought we lacked some time features, we finally **added some useful columns** to have a better insight of the model

- **'Day_week'**, a categorical variable to know the name of the day
- **'Month',** a categorical variable to know the name of the month

```
df1['Day_week'] = df1['Date'].dt.day_name()
df1['Month'] = df1['Date'].dt.month_name()
```

- **'Number_month',** a numeric value to know the number of each month, which we created with a function returning the number

```
df1["Number_month"] = df1["Month"].apply(lambda x : number_month(x))
```

- **'Part_of_the_day',** a categorical feature returning moments of the day based on the hour.

```
df1["Part_of_the_day"] = df1["Hour"].apply(lambda x : part_of_the_day(int(x)))
```

```
# Function to get the number of a month
def number_month(month):
    if month == "January":
        return 1
    elif month == "February":
        return 2
    elif month == "March":
        return 3
    elif month == "April":
        return 4
    elif month == "May":
        return 5
    elif month == "June":
        return 6
    elif month == "July":
        return 7
    elif month == "August":
        return 8
    elif month == "September":
        return 9
    elif month == "October":
        return 10
    elif month == "November":
        return 11
    else :
        return 12
```

```
def part_of_the_day(hour):
    if(hour < 6):
        return "Night"
    elif (hour < 12):
        return "Morning"
    elif (hour < 18):
        return "Afternoon"
    else :
         return "Evening"
```

5. DATA VISUALIZATION

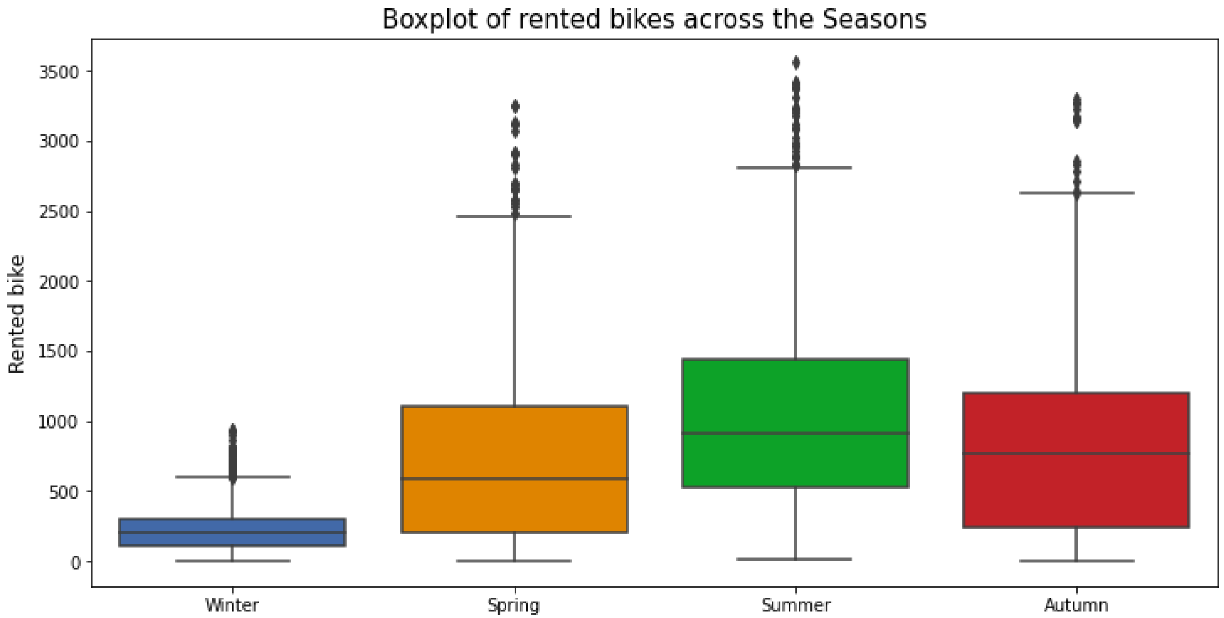
To understand the influence of the weather and the different periods of the year 2017-2018 on the number of rented bike in Seoul, we built several data **visualization models** especially thanks to the **libraries Matplotlib and Seaborn**.

We have split our visualizations in two parts:

- 5.1 Visualizations based on **time variables**
- 5.2 Visualizations based on **weather variables**

5.1 VISUALIZATIONS BASED ON TIME VARIABLES

- **Seasons**



We also notice some outliers, as for some hours of the day, the demand of bikes is higher, no matter the season.

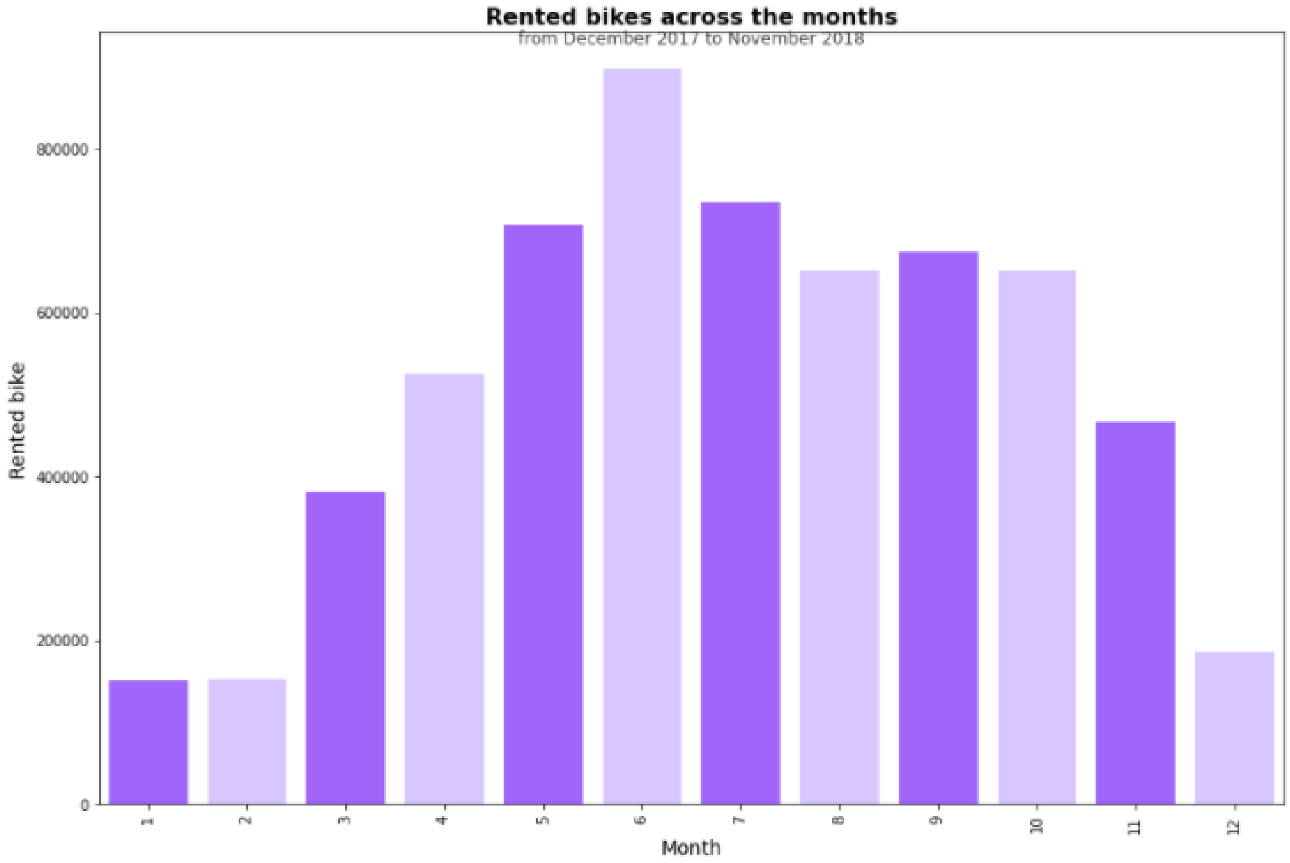
We can see on this boxplot that **bikes are usually more rented in Summer** and are very **less rented during Winter.**

The amount of bikes rented during **Spring and Autumn seems equivalent**, as the seasons do not really differ from each other.

As the box represents 50% of the data breakdown, we know that 500 to 1500 bikes are rented every hour in Seoul during the summer.

5.1 VISUALIZATIONS BASED ON TIME VARIABLES

- **Months**



From this **barplot:**

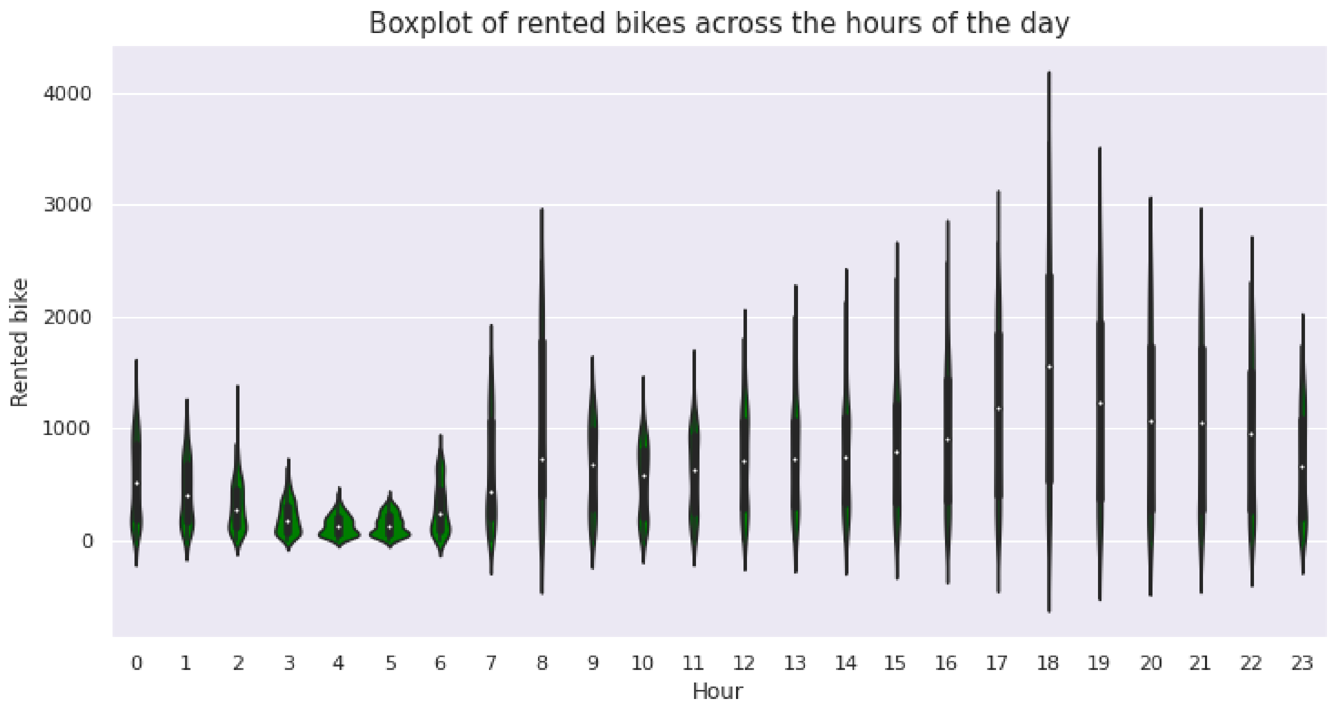
- **Most** bikes are rented in **June, May, July, August, September and October** (>600000 per month), so especially during Summer.

- **Less** bikes are rented during **January & February & December** (<200000 per month), so especially during Winter.

Those months basically confirms the theory said before, that it is **during Summer that bikes are most rented**, where the weather is more warm.

5.1 VISUALIZATIONS BASED ON TIME VARIABLES

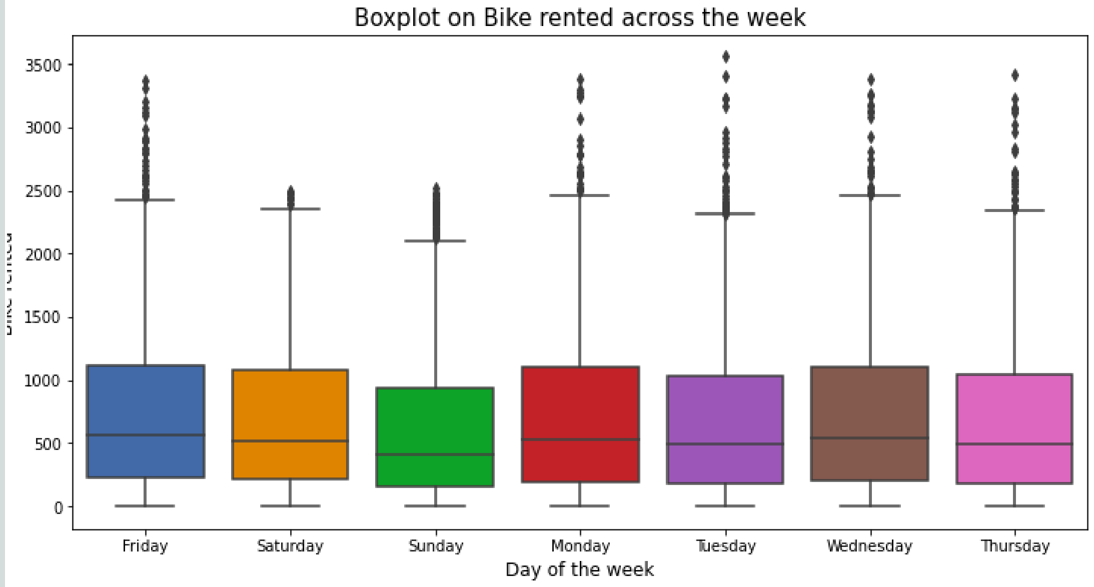
- **Hours**

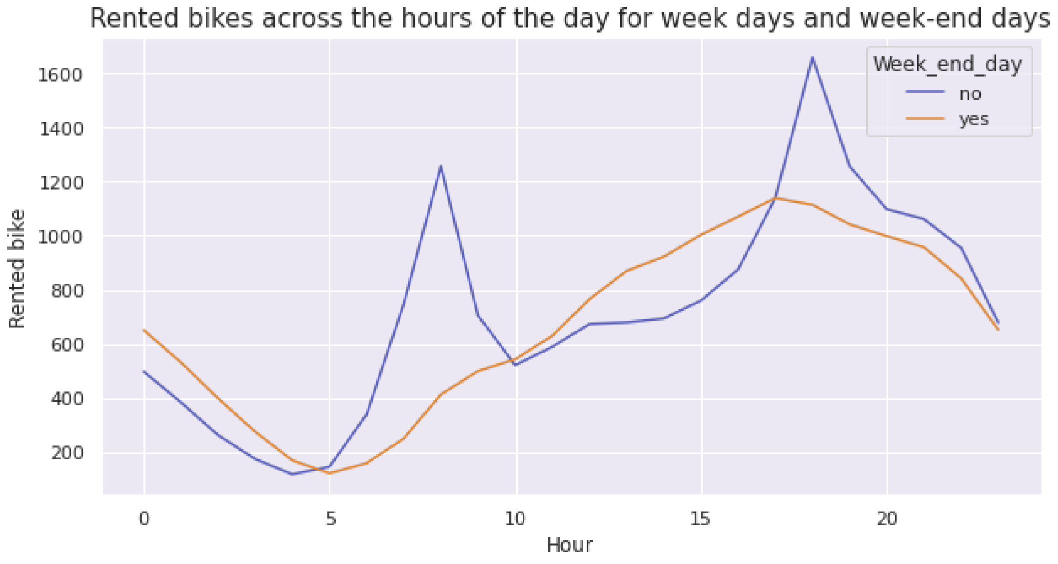


As we can see on the violinplot, the bike is **more used during the evening** (especially near **6pm**) and at **8am** and is **least used early in the morning** (near **4/5am**).
We could suppose that this could be explained because **people commute during those hours** and use bike to to do so.

5.1 VISUALIZATIONS BASED ON TIME VARIABLES

- **Week days / Week end**



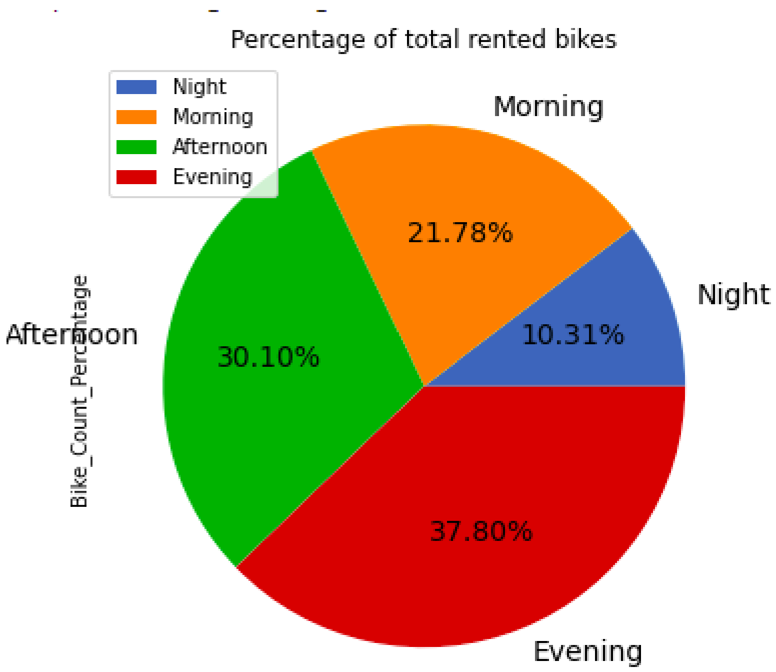


- By looking on the median line of the boxplots, **bike rented demand is slightly higher during the weekdays** and especially on **Monday, Wednesday and Friday** than on Sunday.
- This theory is confirm thanks to the distribution graph, showing **2 peaks throughout the day**, illustrating **commuting periods**, for the weekdays blue curve and no such peaks for the weekend (Saturday & Sunday) orange curve.

So the feature 'Holiday' is therefore important for our future predictions.

5.1 VISUALIZATIONS BASED ON TIME VARIABLES

- **Part of the day**

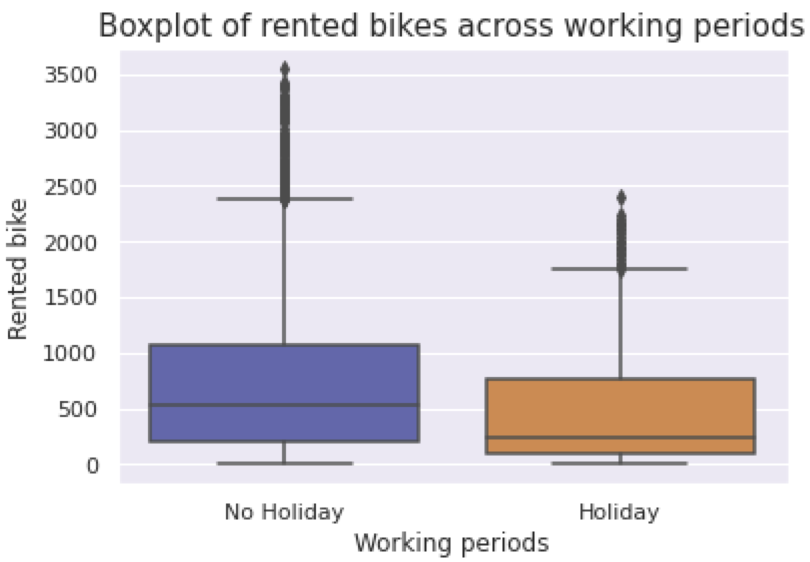


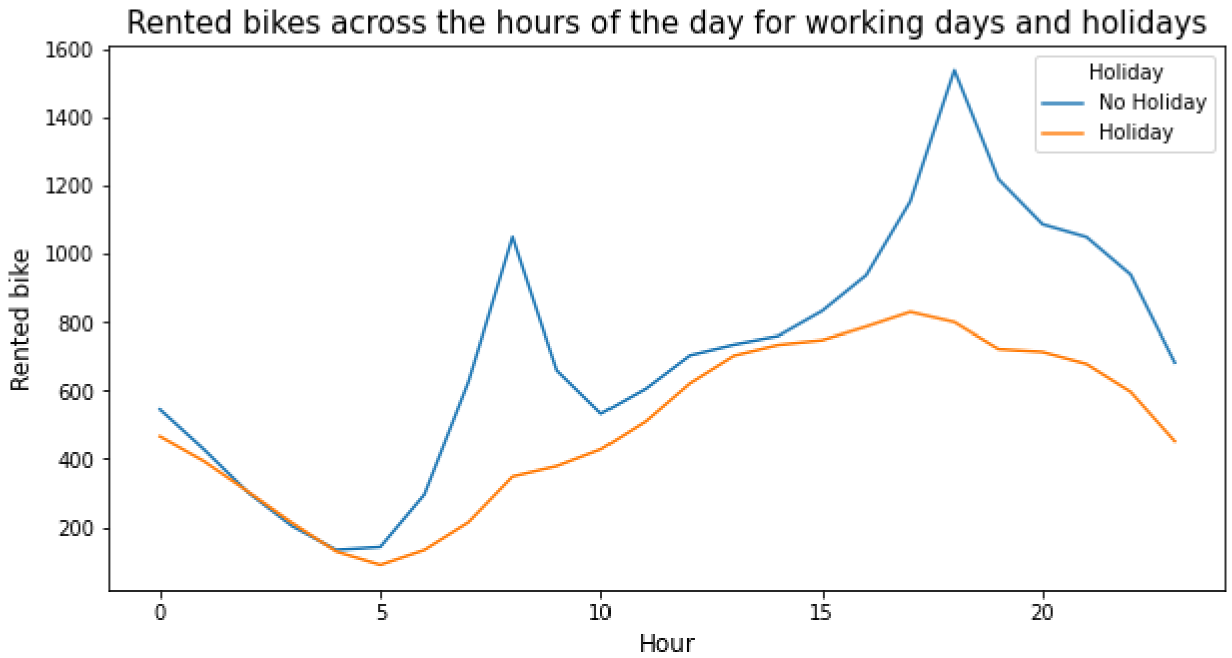
From this pie-chart, we can see that bikes are most rented in the second half of the day (67.90%), i.e. in the **afternoon** and **evening**, confirming the results seen before.

Only 10.31% bikes are rented during the night, because we could safely assumed that most people sleep during this time period

5.1 VISUALIZATIONS BASED ON TIME VARIABLES

- **Working / Holiday periods**

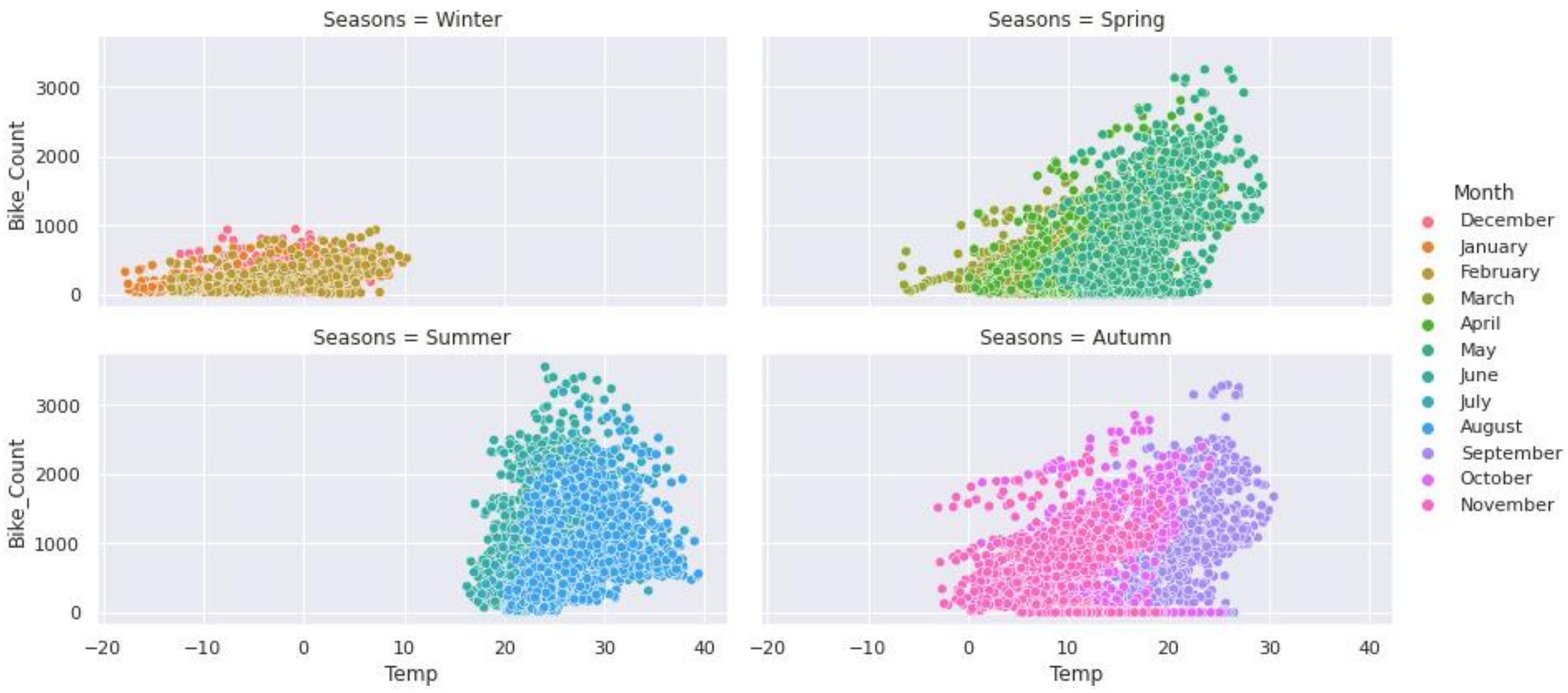




- We can see from this boxplot that **the bikes are more rented during working days than on day off**.

- On the right graph, we notice **two peaks illustrating the commuting movements** observed previously, but during holidays they are very less noticeable because people no longer commute so they use less their bikes. **Demand for bicycles increases** fairly gradually throughout the day **from 5am to 4pm-6pm**.

5.2 VISUALIZATIONS BASED ON WEATHER VARIABLES

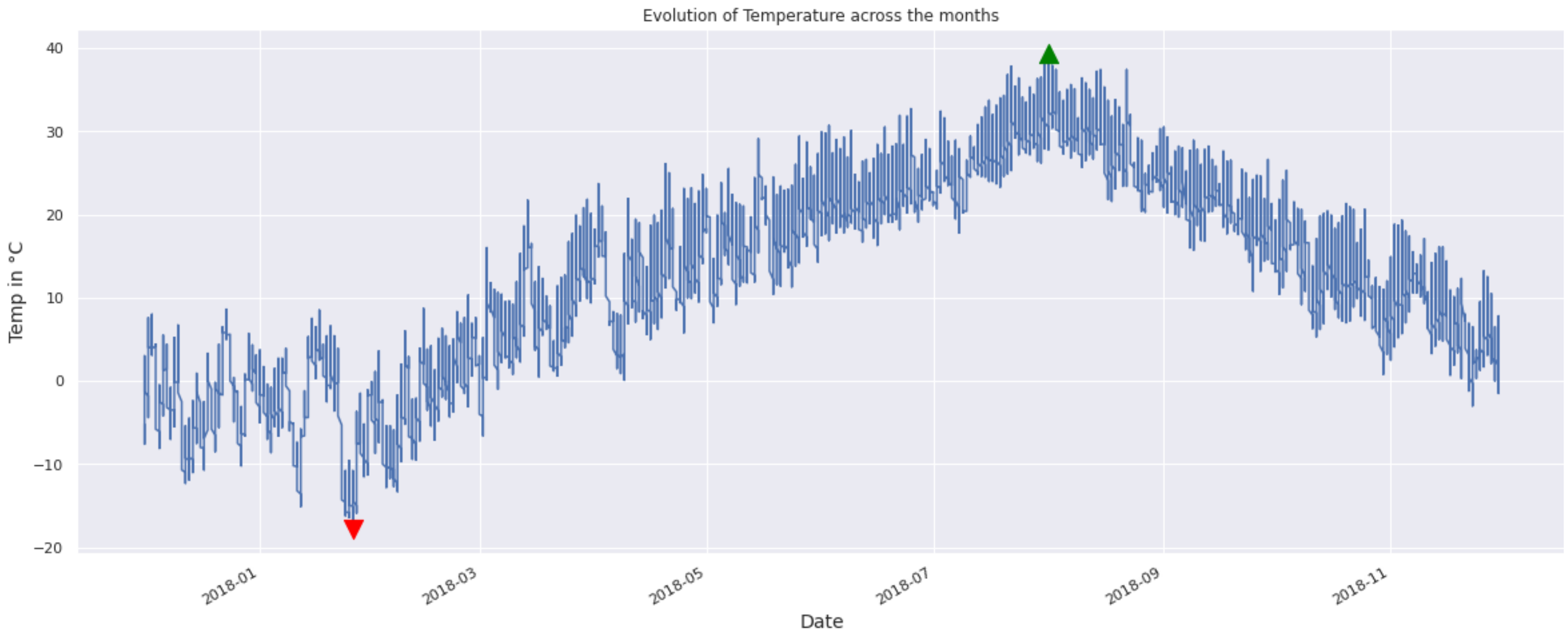


The four scatter plots confirms the **linked impact of the temperature and time period on the amount of bike rented.**

Most bikes are rented when the **weather is warm**, from **15°C to 30°C**, so especially during **Summer**

5.2 VISUALIZATIONS BASED ON WEATHER VARIABLES

- **Temperature across the months**



We can see that the Seoul has more a cold temperate climate

- High temperatures during Summer (20°C-40°C)
- Cold temperatures during Winter (10°C to -20°C)
- **Lowest peak**: 26/01/2018, -17.8°C
- **Highest peak:** 01/08/2018, 39.4°C

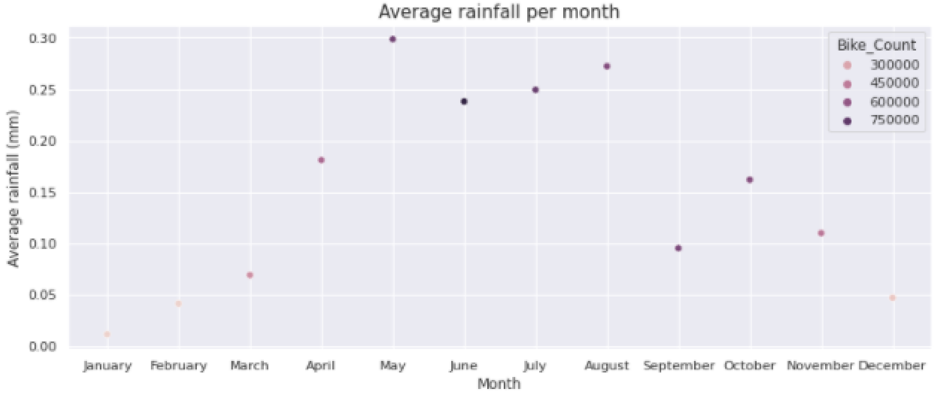
5.2 VISUALIZATIONS BASED ON WEATHER VARIABLES

- **Impact of other features on the target variable**

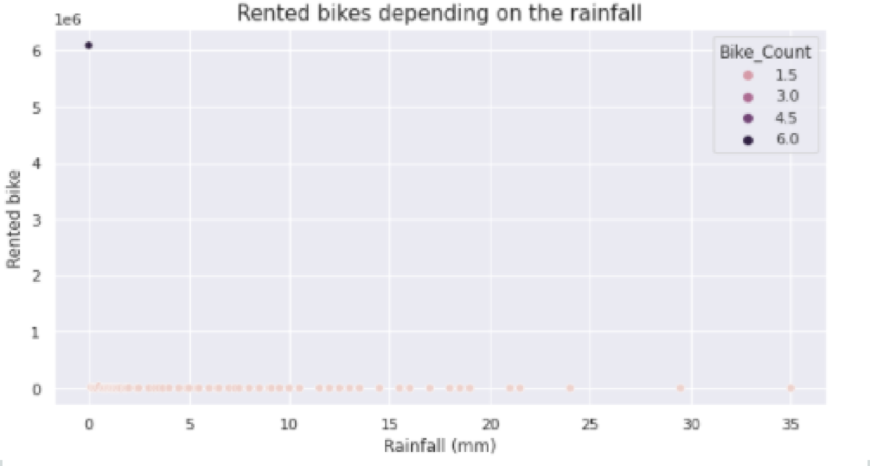
We realized **scatter plots** for each meteorological features, to study the average impact of them on ‘Bike_Count’ per month. We concluded that:

- **Solar radiation, Dew point temperature, Wind and Visibility** may have a **strong influence** on the target variable due to their variation

- **Humidity, Snowfall, Rainfall** have **less influence** on ‘Bike_Count’, so in most of the cases, these weather factors do not stop people from riding their bike.



We noticed that Summer is the rainest season in Seoul.



5. VISUALIZATIONS SUMMARY

- **To resume on time variables:**

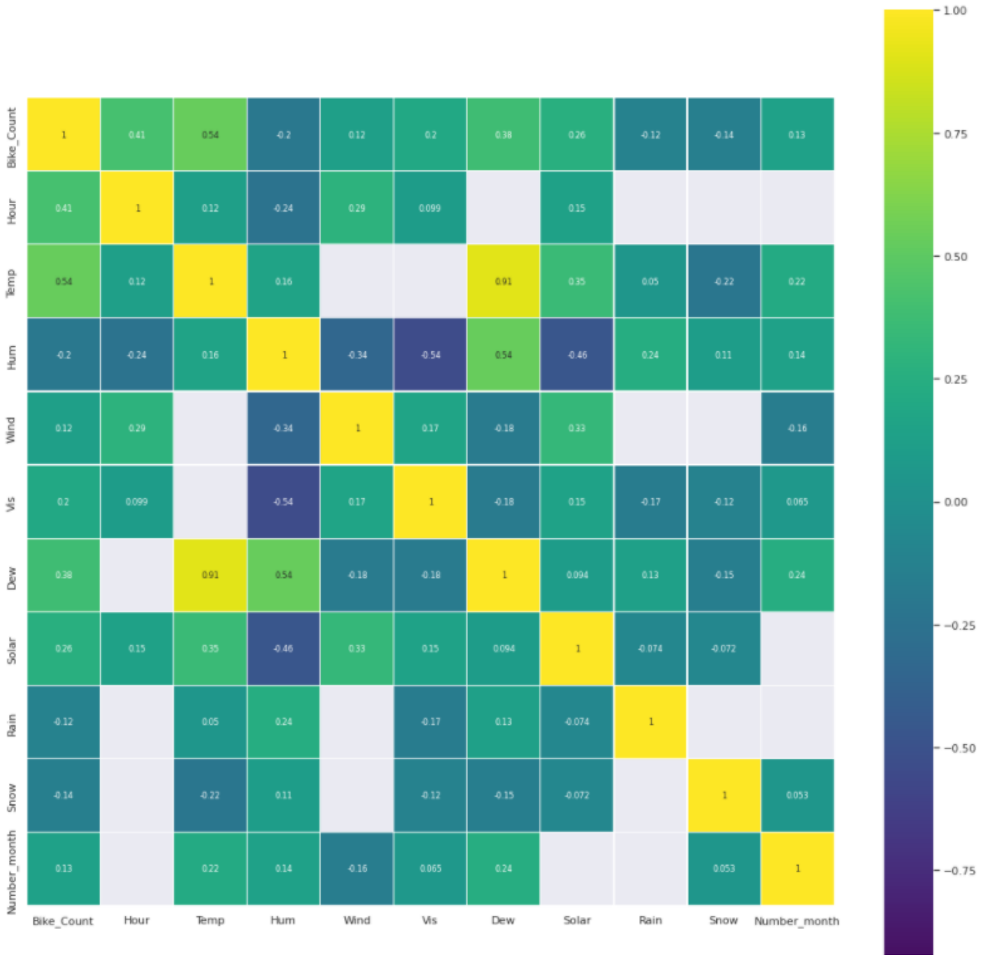
Most bikes are rented:

- During **Summer**, especially in **June**
- The demand increases **from 5am to 4pm-6pm,**.
- Peak at **8am** and **6pm**, corresponding to **commuting hours**
- During **week days** and **work days** than week end and holidays
- **All the features** seem to **have an impact** on the target variable

- **To resume on meteorological factors:**

- **Temperature** has the **highest influence** on bike rented
- **Summer** is the **rainy season** in Seoul but it **does not influence** bike rented
- **Solar radiation, Visibility, Dew point temperature** and **Wind have an influence** on the target variable
- **Humidity, Rainfall and Snowfall do not** seem to **have an influence** on the target variable.

6. DIMENSIONALITY REDUCTION



In order to verify our observations, we made a **Correlation Matrix**

The matrix confirms what was previously said:

- **Snow, Rain and Hum are not correlated** with the target variable
- **Temp and Hour are strongly correlated** with the target variable
- **Dew, Solar and Vis are slightly correlated** with the target variable

We also carried a **PCA analysis** to have more precision on each variable based on the explained variance and we also found that **Wind does not have a strong impact on 'Bike_Count'**

7. FEATURE ENGINEERING

Now that we have selected the features to keep and to delete, we have to modify the dataset before creating our models. We have to **select the useful columns** and **convert the categorical features into numerical values**.

1- We remove the Windspeed, Rainfall, Snowfall and Humidity columns, keep the 'Number_month' column and delete the Month column (because they share the same informations)

```
# We remove the useless columns
del_columns = ['Snow', 'Wind','Rain','Month','Hum']
df_bike = df_bike.drop(del_columns, axis=1)
```

2- We convert 'Holiday' and 'Fday' categorical values to binary

```
[70] df_bike['Holiday'] = [1 if x=='Holiday' else 0 for x in df_bike['Holiday']]
     df_bike['FDay'] = [1 if x=='Yes' else 0 for x in df_bike['FDay']]
```

3- We convert the categorical features 'Seasons', 'Day_week', 'Part_of_the_day' and 'Week_end_day' to numeric values :

```
dummies = ['Seasons','Day_week','Part_of_the_day','Week_end_day']
dummy_data= pd.get_dummies(df_bike[dummies])
```

```
#we concat the 2 frames and drop the old columns
df_bike = pd.concat([df_bike, dummy_data], axis = 1)
df_bike.drop(dummies, axis=1, inplace=True)
```

We convert the 'Date' column values to numeric values

```
df_bike['Date'] = pd.to_numeric(pd.to_datetime(df_bike['Date']))
```

Finally, we check if all values are numeric

```
df_bike.apply(lambda s: pd.to_numeric(s, errors='coerce').notnull().all())
```

```
Date                        True
Bike_Count                  True
Hour                        True
Temp                        True
Vis                         True
Dew                         True
Solar                       True
Holiday                     True
FDay                        True
Number_month                True
Seasons_Autumn              True
Seasons_Spring              True
Seasons_Summer              True
Seasons_Winter              True
Day_week_Friday             True
Day_week_Monday             True
Day_week_Saturday           True
Day_week_Sunday             True
Day_week_Thursday           True
Day_week_Tuesday            True
Day_week_Wednesday          True
Part_of_the_day_Afternoon   True
Part_of_the_day_Evening     True
Part_of_the_day_Morning     True
Part_of_the_day_Night       True
Week_end_day_no             True
Week_end_day_yes            True
dtype: bool
```

8. DATA MODELING

Before creating our models, we **split the data** into a **train** and a **test set**.

```
x,y = df_bike.loc[:,df_bike.columns != 'Bike_Count'], df_bike.loc[:,'Bike_Count']
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.33, random_state = 1)
```

And then, we **scale** the data.

```
sc = StandardScaler()
x_train = sc.fit_transform(x_train)
x_test = sc.transform(x_test)
```

We performed **9 different types of Regression models** :

- Linear Regression
- Ridge Regression
- Lasso Regression
- Support Vector Machine Regression (SVR)
- K-nearest neighbors algorithm for regression
- Decision Tree
- Random Forest
- Gradient Boosting
- Neural Network Regression

8. DATA MODELING

For each model, we tried to change the **hyper parameters** and made a **grid search** to find the best ones. Then, we ran the model with those hyper parameters and printed the **coefficient of determination R^2**, the **Mean Squared Error (MSE)** and the **Root Mean Squared Error (RMSE),**

Here is an example with SVR model:

Cross Validation

```
[ ]  svm_svr = SVR()
     parameters = {"kernel": ['poly', 'rbf', 'sigmoid'],
                   "C": [100, 50, 25, 10, 5, 1.0, 0.1, 0.01],
                   "gamma": ['scale', 'auto']}

     grid_search = GridSearchCV(estimator=svm_svr,
                                param_grid=parameters,
                                cv=10)

     grid_search.fit(x_train, y_train)
     grid_search.best_params_
```

```
{'C': 100, 'gamma': 'scale', 'kernel': 'rbf'}
```

Perform the model

```
[87]  svm_svr = SVR(C=100, gamma='scale', kernel='rbf').fit(x_train, y_train)

      # Prediction
      y_prediction_svm_svr = svm_svr.predict(x_test)

      # Score
      r2_svm_svr = r2_score(y_test, y_prediction_svm_svr)
      MSE_svm_svr = mean_squared_error(y_test, y_prediction_svm_svr)
      print("R2 =", r2_svm_svr)
      print("Mean squared error =", MSE_svm_svr)
      print("Root mean squared error =", np.sqrt(mean_squared_error(y_test, y_prediction_svm_svr)))
```

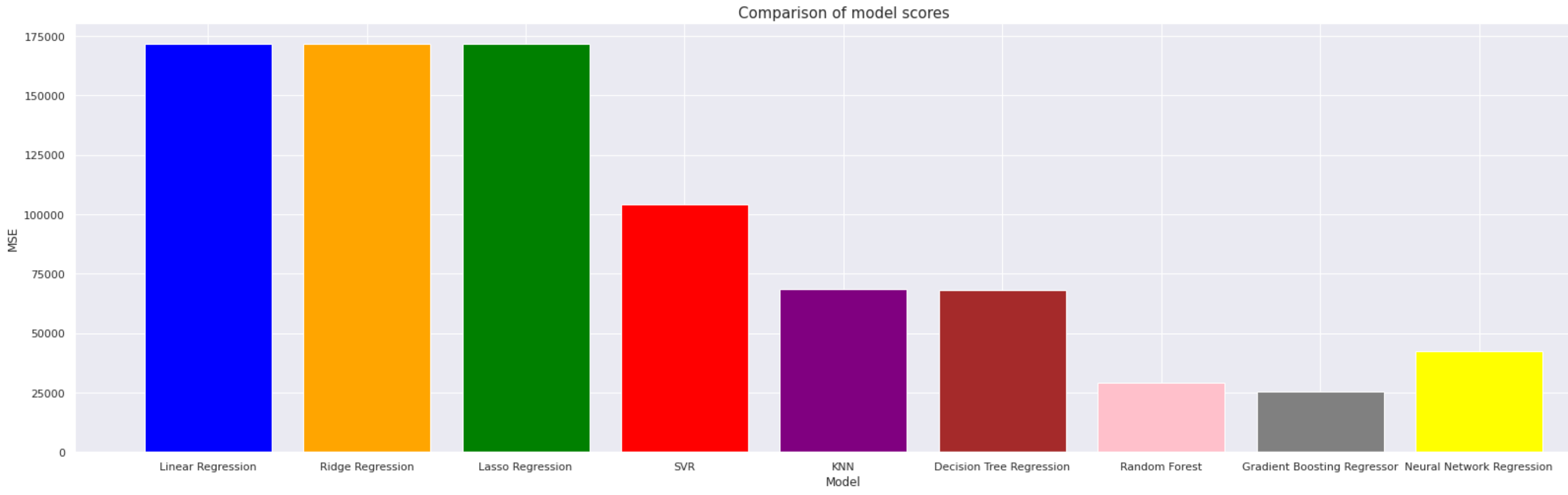
```
R2 = 0.7451979674778288
Mean squared error = 103955.3453147767
Root mean squared error = 322.4210683481721
```

8. DATA MODELING

After creating all our models, we compared the results.

To do so, we compared the value for the MSE :

MSE



	MSE
Gradient Boosting Regressor	25257.171993
Random Forest	29204.167453
Neural Network Regression	42445.137030
Decision Tree Regression	67886.823245
KNN	68431.050344
SVR	103955.345315
Lasso Regression	171524.673323
Ridge Regression	171589.536845
Linear Regression	171603.837349

8. DATA MODELING

We were surprised to find that all of our **MSE scores were insanely high**, so we did some research to understand why they weren't close to 0.

We have found that it is complicated to use MSE scores to assess the performance of our models due to:

- **Very versatile and dispersed data**, because of the features having an extended scale which can vary rapidly from day to day ;

.- **Lots of records (8760),** which we cannot delete because we need the entire dataset to build a model due to the **disparate values depending on weather or time related.**

Even though we refined our models, removed some uncorrelated columns with the target, and scaled the data, the MSE scores remained incredibly high.

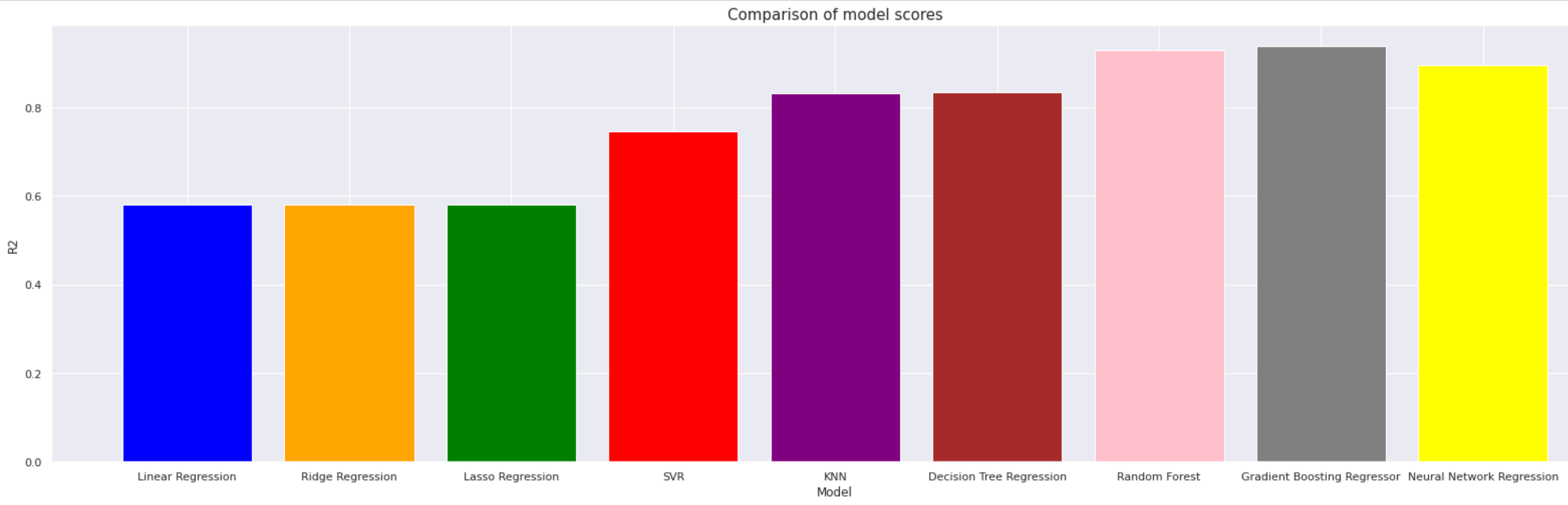
Thus, we decided to **base our assessment of the model's performance on the R2 score** rather than the MSE score due to the peculiarity of our dataset.

	MSE
Gradient Boosting Regressor	25257.171993
Random Forest	29204.167453
Neural Network Regression	42445.137030
Decision Tree Regression	67886.823245
KNN	68431.050344
SVR	103955.345315
Lasso Regression	171524.673323
Ridge Regression	171589.536845
Linear Regression	171603.837349

8. DATA MODELING

And we compared the value for the R^2 :

R^2



	r2
Gradient Boosting Regressor	0.938093
Random Forest	0.928418
Neural Network Regression	0.895964
Decision Tree Regression	0.833605
KNN	0.832271
SVR	0.745198
Lasso Regression	0.579581
Ridge Regression	0.579422
Linear Regression	0.579387

8. DATA MODELING

By looking at those comparisons, we can see that the best result is obtained with the **Gradient Boosting Regressor** model, because its MSE parameter is the lowest and its **R^2 is closest to 1.**

That's why we decided to choose this type of model to make predictions.

In order to try to have a better result, we performed a gradient boosting regressor model with **all features**. Here are the obtained results :

```
R2 = 0.9406712396161091
Mean squared error = 24205.22988672927
Root mean squared error = 155.58030044555534
```

We obtained a better score and decided to keep this model for the predictions,

9. API - FLASK

We had two choices of framework to transform our model into an API : **Django** and **Flask**.

We decided to choose **Flask** because Django is time-consuming, and it doesn't really worth it for a one- or two-pages' API. We didn't need complicated structures for our API, as the user just need to fill in a form, a couple of information (just need to make a single POST request) to obtain the prediction. The user can get predictions for dates from 01.12.2018. We did not define a date limit afterwards, but in the future, it will be necessary to collect more data for subsequent years (after 2017-2018) to continue to generate good predictions.

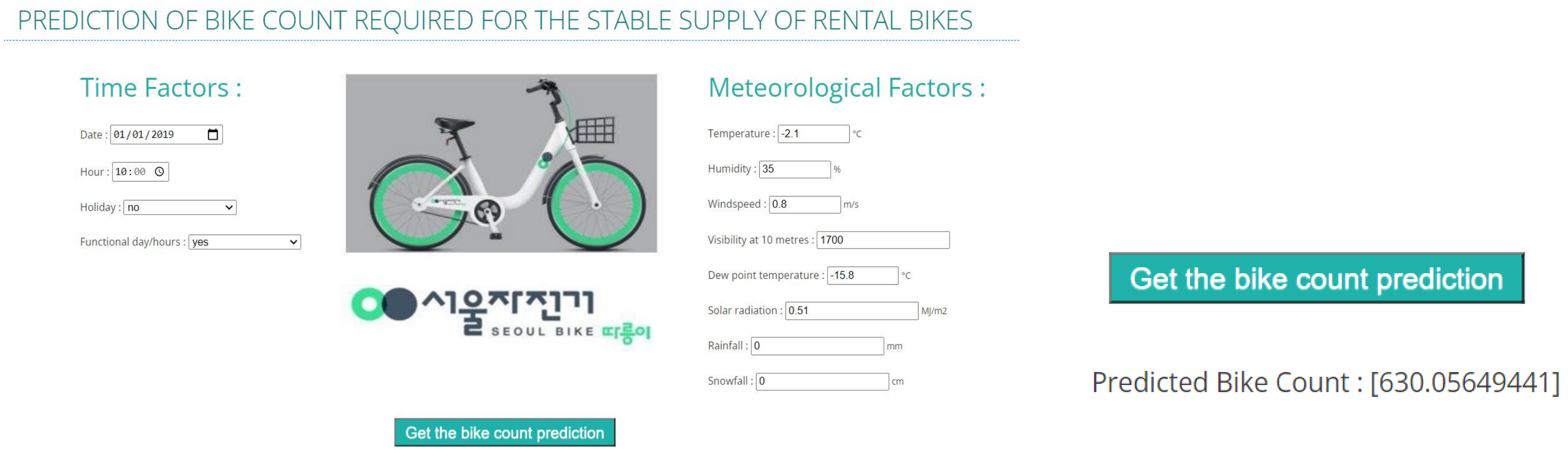
To predict the rented bike count, we used the model (gradient boosting) obtained in the Notebook.

We used **pickle** to serialize the model. This enable us to access to the model without having to run the entire notebook, and to make predictions with our Flask API.

9. API - FLASK

We created an .html template to present our solution.

Once the user has launched the API, all he has to do is fill in all the information requested and press the button at the bottom of the page to get the prediction. It will be displayed just below. He can then fill out the form again as many times as he wants to get more predictions.



10. CONCLUSION

Although it is possible to create a model with a good score and generate predictions, it is important to keep in mind that many parameters are related to the **weather**, which are **relatively random** and can vary a lot from year to year. As a result, **predictions may be distorted** by those weather factors.

In addition, we have in this dataset data ranging from 01.12.2017 to 30.11.2018, i.e. data for 1 year. We could have a **better model** if we had **more data**, for **other years** as well.

We are quite satisfied by the results of our work. With this project, we had the opportunity to work **on a real case and with real data**, which makes the study **more interesting**. We had the occasion to use the skills and knowledge we acquired with our Python and Machine Learning courses. But also learn how to expose our results on a web framework by learning to use Flask which allows to make APIs easily.

We had a **real progression** throughout the project, with an application of everything we learn during this course and even more thanks to our **researches** in order to **seek for the best visualizations and models**.

We have been able to **study the dataset in detail**, **observe the influence of the different features**, make **visualizations** of several types, **interpret them and draw conclusions** for the features to be used for the models to be built. We were able to use **models** we had seen in class but also **discover new ones**. And finally, after creating a performant model, we were able to present it through a **nice interface** by creating a **flask application**.