Seoul Rented Bike Prediction

Problem

- The dataset shows how many bikes are rented every hour in december 2017 and in 2018.
- The goal is, thus, to find a way to ***predict how many bikes will be rented at a given time*** to prevent from any lack of bikes.

SeoulBikeData

- There are: 8760 rows & 14 columns

Attribute Information:

Date : year-month-day
Rented Bike count - Count of bikes rented at each hour
Hour - Hour of he day
Temperature-Temperature in Celsius
Humidity - %
Windspeed - m/s
Visibility - 10m
Dew point temperature - Celsius
Solar radiation - MJ/m2
Rainfall - mm
Snowfall - cm
Seasons - Winter, Spring, Summer, Autumn
Holiday - Holiday/No holiday
Functional Day - NoFunc(Non Functional Hours), Fun(Functional hours)

- Our target: the 'Rented Bike Count' column

	Date	Rented Bike Count	Hour	Temperature(°C)	Humidity(%)	Wind speed (m/s)	Visibility (10m)	Dew point temperature(°C)	Solar Radiation (MJ/m2)	Rainfall(mm)	Snowfall (cm)	Seasons	Holiday	Functioning Day
0	01/12/2017	254	0	-5.2	37	2.2	2000	-17.6	0.0	0.0	0.0	Winter	No Holiday	Yes
1	01/12/2017	204	1	-5.5	38	0.8	2000	-17.6	0.0	0.0	0.0	Winter	No Holiday	Yes
2	01/12/2017	173	2	-6.0	39	1.0	2000	-17.7	0.0	0.0	0.0	Winter	No Holiday	Yes
3	01/12/2017	107	3	-6.2	40	0.9	2000	-17.6	0.0	0.0	0.0	Winter	No Holiday	Yes
4	01/12/2017	78	4	-6.0	36	2.3	2000	-18.6	0.0	0.0	0.0	Winter	No Holiday	Yes

Data cleaning

- ❑ Removing ‘non number values’;
- ❑ Removing symbols in the column names;
- ❑ Deleting "Functioning Day" column.

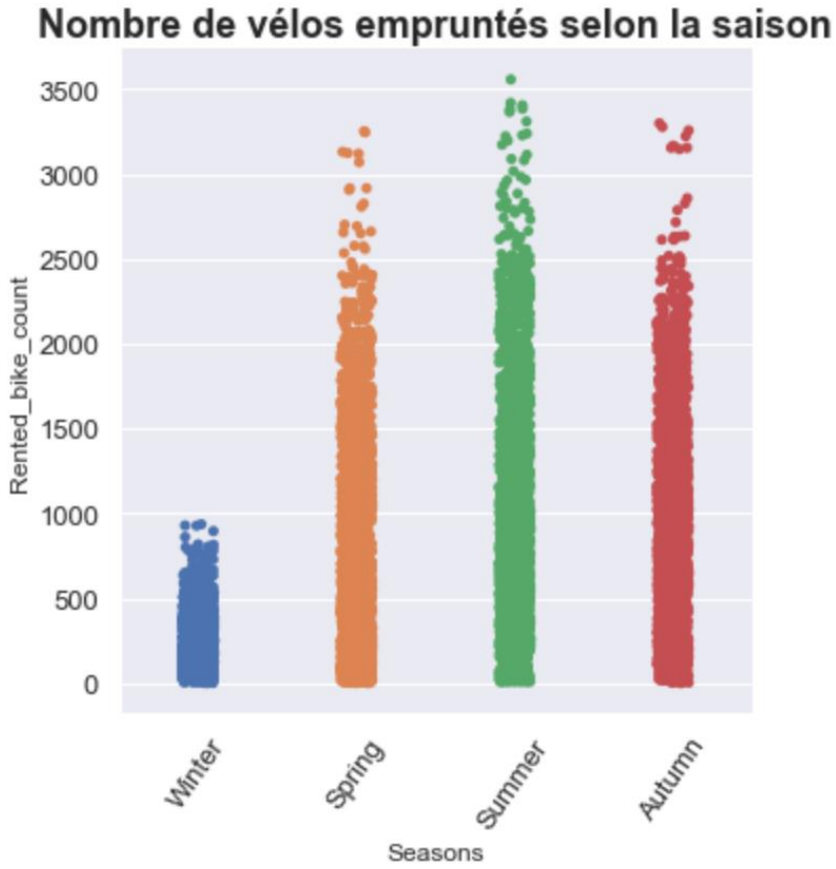
```
data.rename(columns = {"Rented Bike Count":"Rented_bike_count",
                       "Temperature(°C)" : "Temperature",
                       "Humidity(%)": "Humidity",
                       "Wind speed (m/s)": "Wind_speed",
                       "Visibility (10m)":"Visibility",
                       "Dew point temperature(°C)": "Dew_point_temperature",
                       "Solar Radiation (MJ/m2)": "Solar_Radiation",
                       "Rainfall(mm)": "Rainfall",
                       "Snowfall (cm)": "Snowfall"}, inplace=True)
```

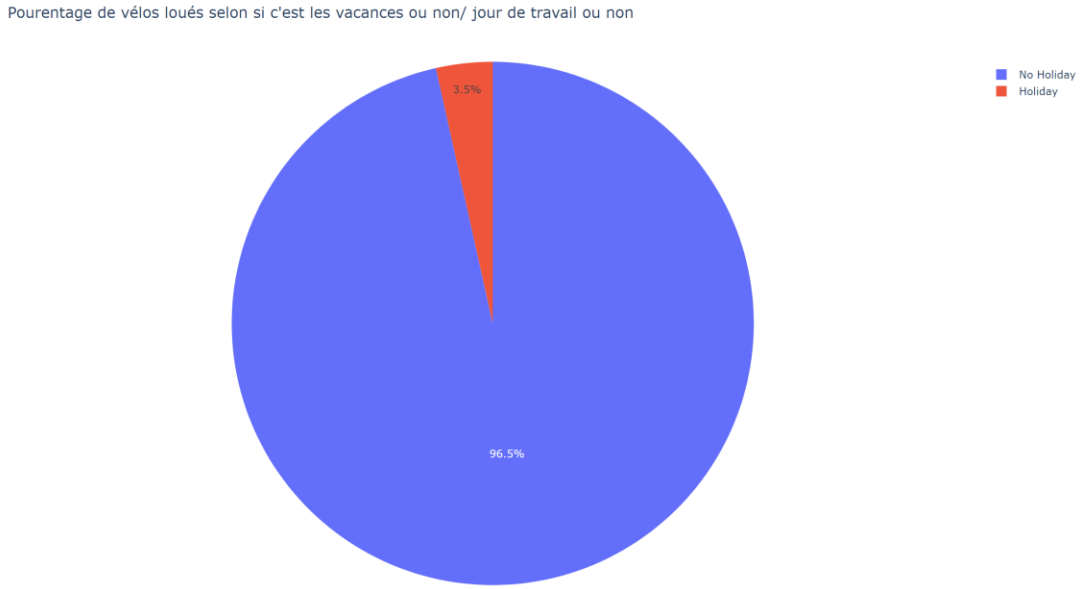
```
Entrée [121]: data[data.isna()].sum()

Out[121]: Date                    0.0
          Rented_bike_count       0.0
          Hour                    0.0
          Temperature             0.0
          Humidity                0.0
          Wind_speed              0.0
          Visibility              0.0
          Dew_point_temperature   0.0
          Solar_Radiation         0.0
          Rainfall                0.0
          Snowfall                0.0
          Seasons                 0.0
          Holiday                 0.0
          Functioning Day         0.0
          dtype: float64
```

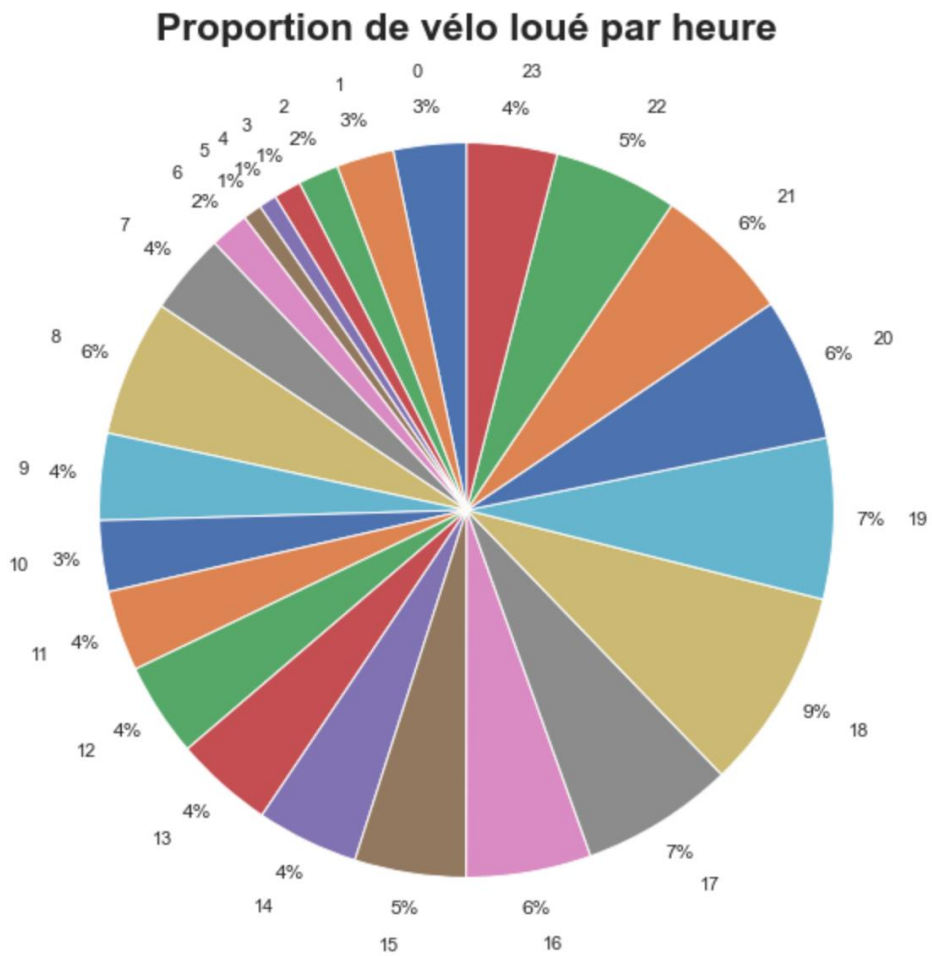
```
data[(data["Rented_bike_count"]!=0) & (data["Functioning Day"]=="No")]
```

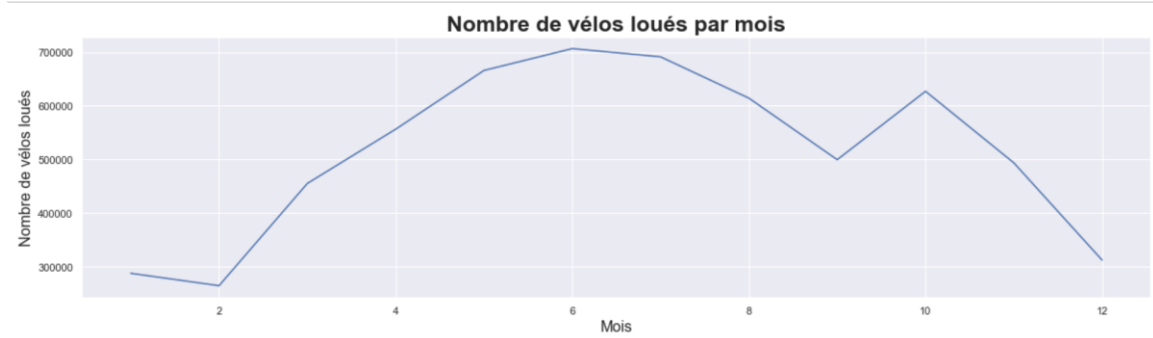
Date	Rented_bike_count	Hour	Temperature	Humidity	Wind_speed	Visibility	Dew_point_temperature

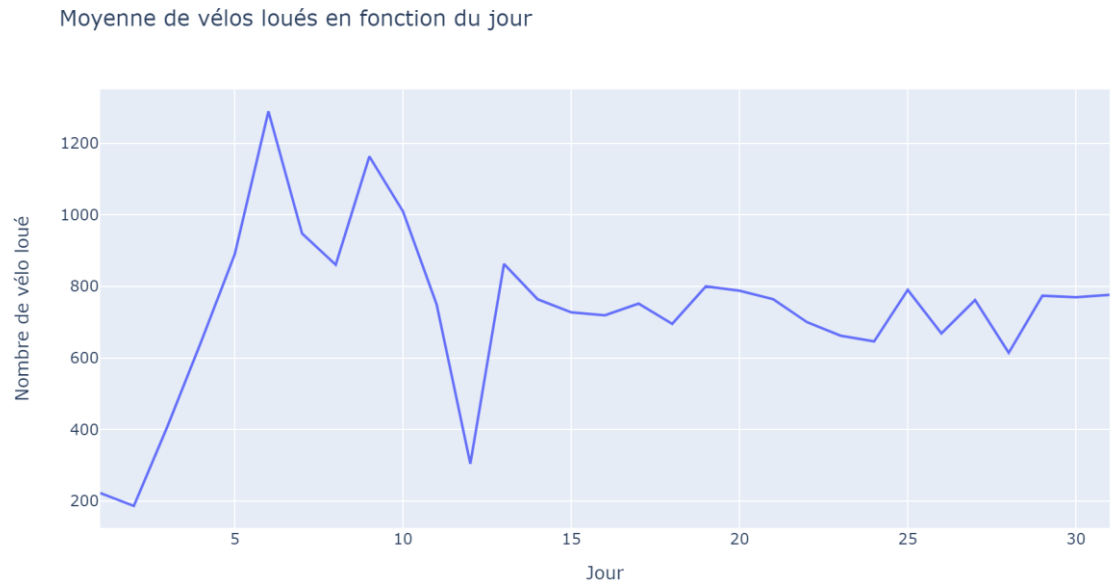
Data visualisations






Proportion de vélo loué par heure
0 3%
1 3%
2 2%
3 1%
4 1%
5 1%
6 2%
7 4%
8 6%
9 4%
10 3%
11 4%
12 4%
13 4%
14 4%
15 5%
16 6%
17 7%
18 9%
19 7%
20 6%
21 6%
22 5%
23 4%


Nombre de vélos loués par mois
Nombre de vélos loués
700000
600000
500000
400000
300000
2
4
6
8
10
12
Mois


Moyenne de vélos loués en fonction du jour
Nombre de vélo loué
1200
1000
800
600
400
200
5
10
15
20
25
30
Jour

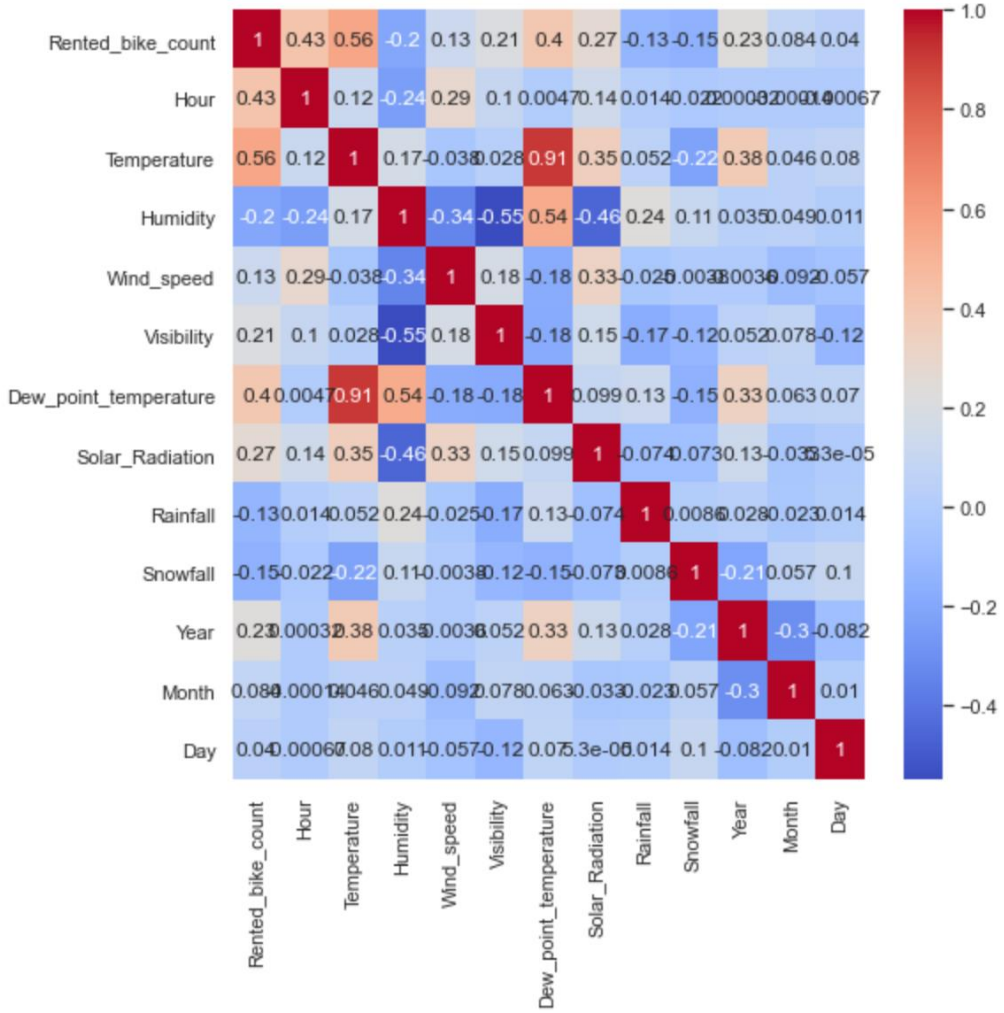
Splitting the "Date" into columns: month, year and day and hours to do our visualisations and prediction

```
Entrée [143]: data.columns
```

```
Out[143]: Index(['Rented_bike_count', 'Temperature', 'Humidity', 'Wind_speed',
                 'Visibility', 'Solar_Radiation', 'Rainfall', 'Snowfall', 'Year',
                 'Month', 'Day', 'Hour_0', 'Hour_1', 'Hour_10', 'Hour_11', 'Hour_12',
                 'Hour_13', 'Hour_14', 'Hour_15', 'Hour_16', 'Hour_17', 'Hour_18',
                 'Hour_19', 'Hour_2', 'Hour_20', 'Hour_21', 'Hour_22', 'Hour_23',
                 'Hour_3', 'Hour_4', 'Hour_5', 'Hour_6', 'Hour_7', 'Hour_8', 'Hour_9',
                 'Seasons_Autumn', 'Seasons_Spring', 'Seasons_Summer', 'Seasons_Winter',
                 'Holiday_Holiday', 'Holiday_No Holiday'],
                dtype='object')
```

Correlation and statistical analysis

- "Temperature" highly correlated with "Dew_point_temperature", we have dropped the latter



See if variables are related to our target

- **Pearson test** between our target "Rented Bike Count" and the other quantitative variables.

	Temperature
pearson_coef	0.56274
p_value	0.00000
	Humidity
pearson_coef	-2.019727e-01
p_value	1.241129e-78
	Wind_speed
pearson_coef	1.250219e-01
p_value	7.704373e-31
	Visibility
pearson_coef	2.123228e-01
p_value	6.895319e-87
	Solar_Radiation
pearson_coef	2.738616e-01
p_value	1.692193e-145
	Rainfall
pearson_coef	-1.286261e-01
p_value	1.469221e-32
	Snowfall
pearson_coef	-1.516108e-01
p_value	1.049056e-44

- **Anova test** between our target "Rented Bike Count" and qualitative variables.

	df	sum_sq	mean_sq	F	PR(>F)
Seasons	3.0	8.273756e+08	2.757919e+08	875.601073	0.0
Residual	8461.0	2.664998e+09	3.149743e+05	NaN	NaN

	df	sum_sq	mean_sq	F	PR(>F)
Holiday	1.0	1.714688e+07	1.714688e+07	41.756703	1.090780e-10
Residual	8463.0	3.475227e+09	4.106377e+05	NaN	NaN

Prediction model: RandomForestRegressor

```
X= data.drop(columns = 'Rented_bike_count')
X1 = dataset1.drop(columns = 'Rented_bike_count')
X2= dataset2.drop(columns = "Rented_bike_count")
y = data["Rented_bike_count"]
```

```
#Split des données
X_train,X_test,y_train,y_test = train_test_split(X,y, test_size=0.2,random_state=42)
X1_train,X1_test,y1_train,y1_test = train_test_split(X1,y, test_size=0.2,random_state=42)
X2_train,X2_test,y2_train,y2_test = train_test_split(X2,y, test_size=0.2,random_state=42)
```

```
from sklearn.ensemble import RandomForestRegressor

rf_model=RandomForestRegressor()
rf_model.fit(X_train,y_train)

y_pred=rf_model.predict(X_test)
```

```
print('score train: ', rf_model.score(X_train, y_train)*100,"%")
print('score test: ', rf_model.score(X_test, y_test)*100,"%")
```

```
score train:  98.08323764062936 %
score test:  85.27606812026087 %
```

```
r2_score(y_test,y_pred)
```

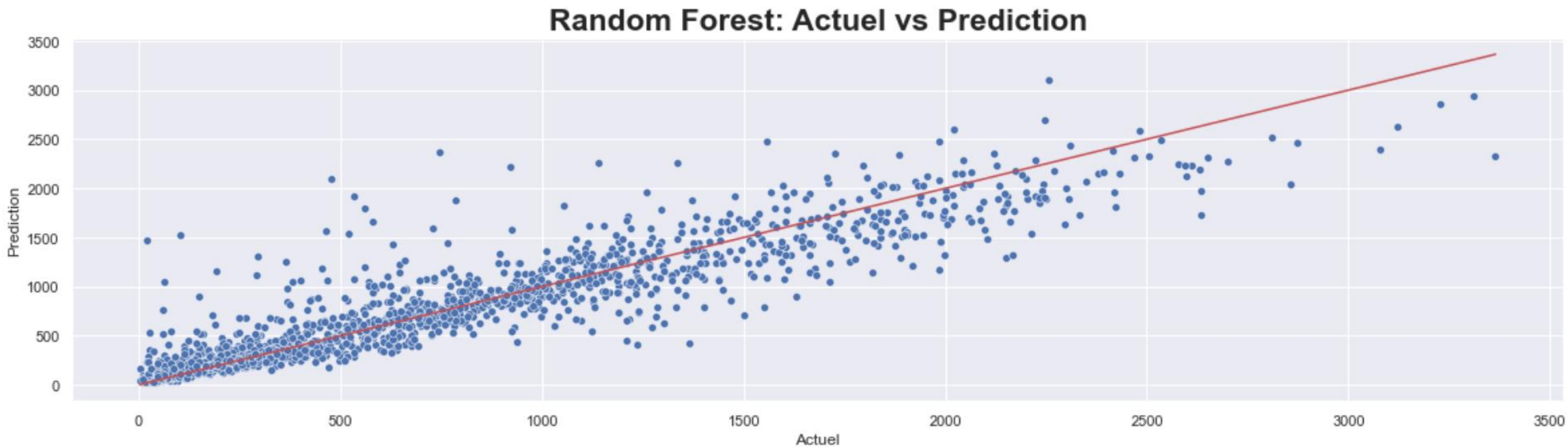
```
0.8527606812026087
```

```
mean_squared_error(y_test,y_pred,squared = False)
```

```
240.42493334537656
```

```
fig,ax = plt.subplots()

scatter = sns.scatterplot(x=y_test, y=y_pred)
line = sns.lineplot(x=[min(y_test),max(y_test)], y=[min(y_test),max(y_test)],color="r")

plt.title('Random Forest: Actuel vs Prediction',fontsize=23, fontweight="bold")
plt.xlabel('Actuel')
plt.ylabel('Prediction')
```

```
Text(0, 0.5, 'Prediction')
```



Other models

Linear Regression

```
score train :  0.6614760895806244
score test : 0.6652773441455361
mean absolute error is  : 502040.6058586664
mean squared error is  : 885073432553.3466
Root mean squared error is  : 940783.414263531
R2 score is  : 0.5630734308741695
```

Bagging

```
score train:  97.32067315385038 %
score test:  83.63407222927695 %
```

```
from sklearn.metrics import r2_score
r2_score(y_test,y_pred)
```

```
0.8363407222927696
```

```
from sklearn.metrics import mean_squared_error
mean_squared_error(y_test,y_pred,squared = False)
```

```
253.47662715501144
```

Best model

```
Cross Score Validation pour RandomForestRegressor 86.14802299742188 %
Cross Score Validation pour DecisionTreeRegressor 74.79894741831353 %
Cross Score Validation pour BaggingRegressor 84.88473872918703 %
```