SEOUL BIKE DATA PROJECT

A project done by CAMARA Abdoul-Karim

PYTHON FOR DATA ANALYSIS

DATASET INFORMATION

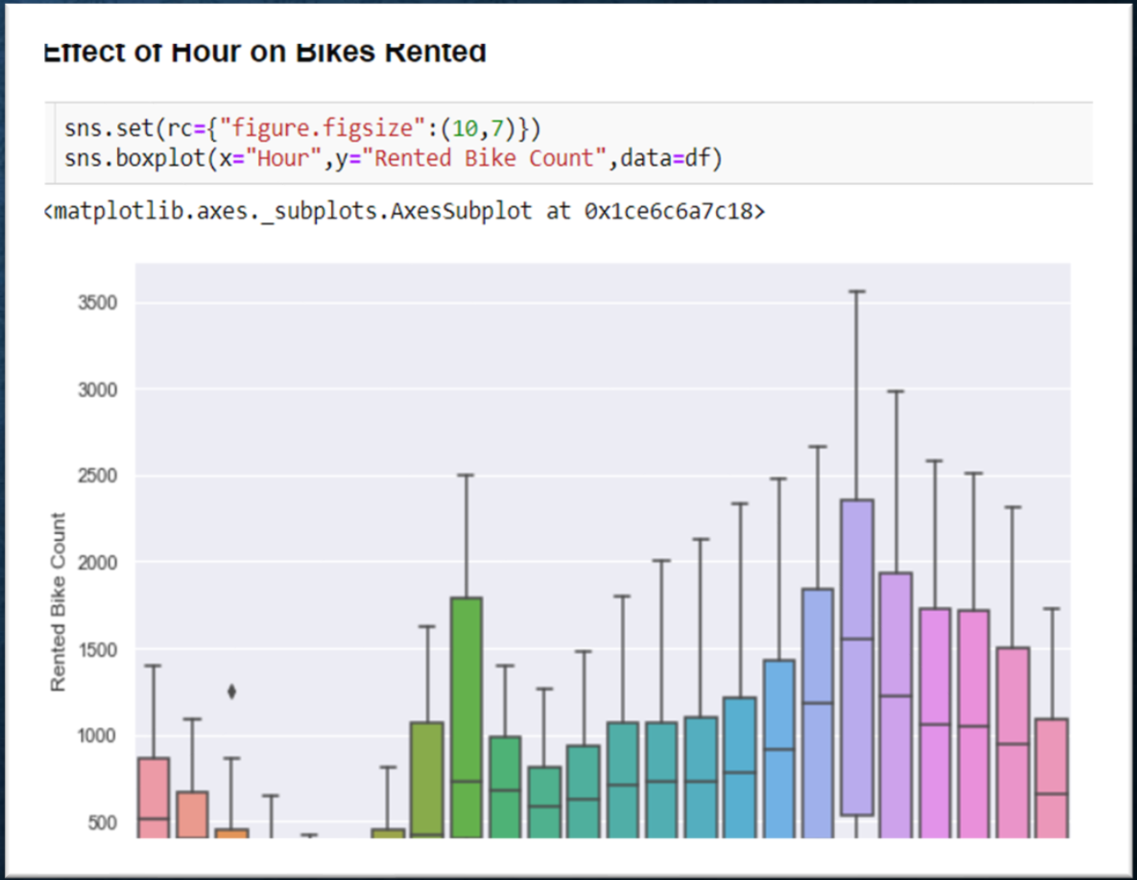
- We found he dataset of seoulbike in a csv file which we can easily read and work within our project.
- Dataset contains 8760 rows and 14 columns
- This dataset contains the information of the bikes rented at each data along with different weather conditions at that day.

```
df = pd.read_csv("SeoulBikeData.csv",encoding='latin-1')
df.head()
```

	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	Functionin Da
0	01/12/2017	254	0	-5.2	37	2.2	2000	-17.6	0.0	0.0	0.0	Winter	No Holiday	Ye
1	01/12/2017	204	1	-5.5	38	0.8	2000	-17.6	0.0	0.0	0.0	Winter	No Holiday	Ye
2	01/12/2017	173	2	-6.0	39	1.0	2000	-17.7	0.0	0.0	0.0	Winter	No Holiday	Ye
3	01/12/2017	107	3	-6.2	40	0.9	2000	-17.6	0.0	0.0	0.0	Winter	No Holiday	Ye
4	01/12/2017	78	4	-6.0	36	2.3	2000	-18.6	0.0	0.0	0.0	Winter	No Holiday	Ye

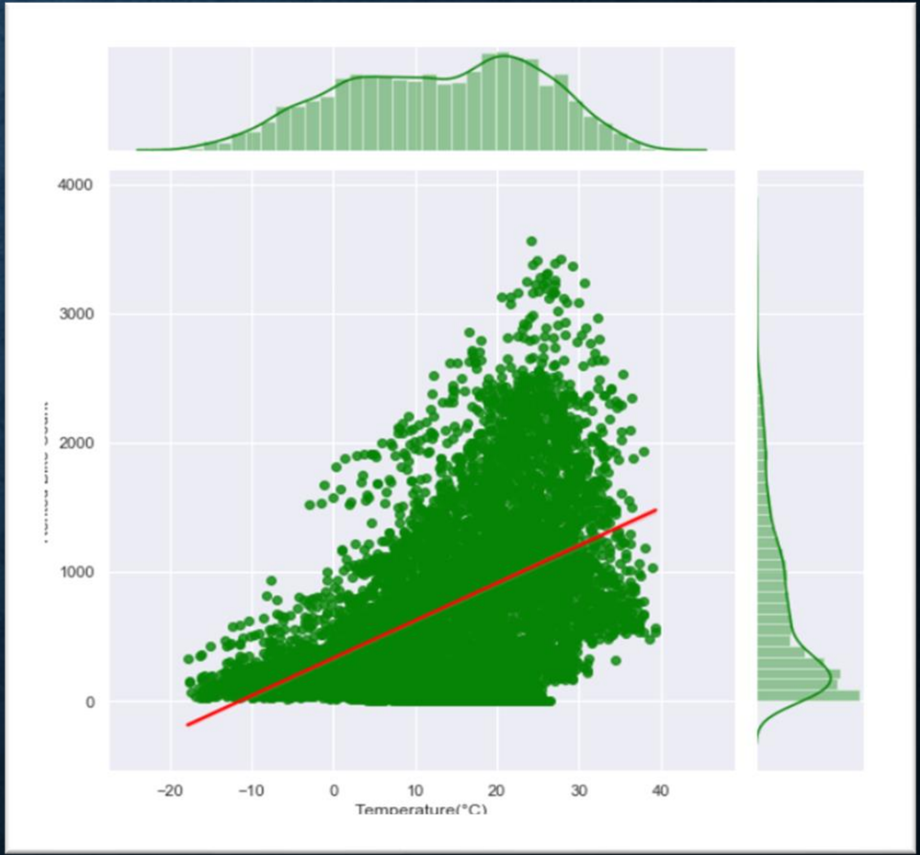
DATA VISUALIZATION

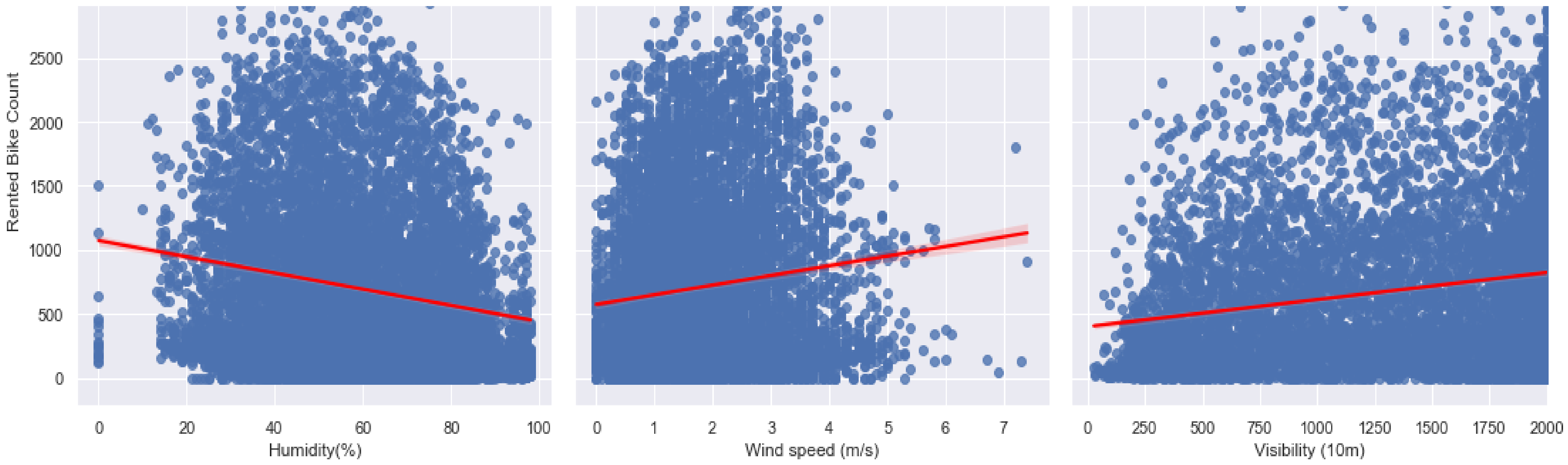
- We have plotted some visualizations from the dataset to show the relationship of different variables with the target variable which is the number of rented bikes at each day.



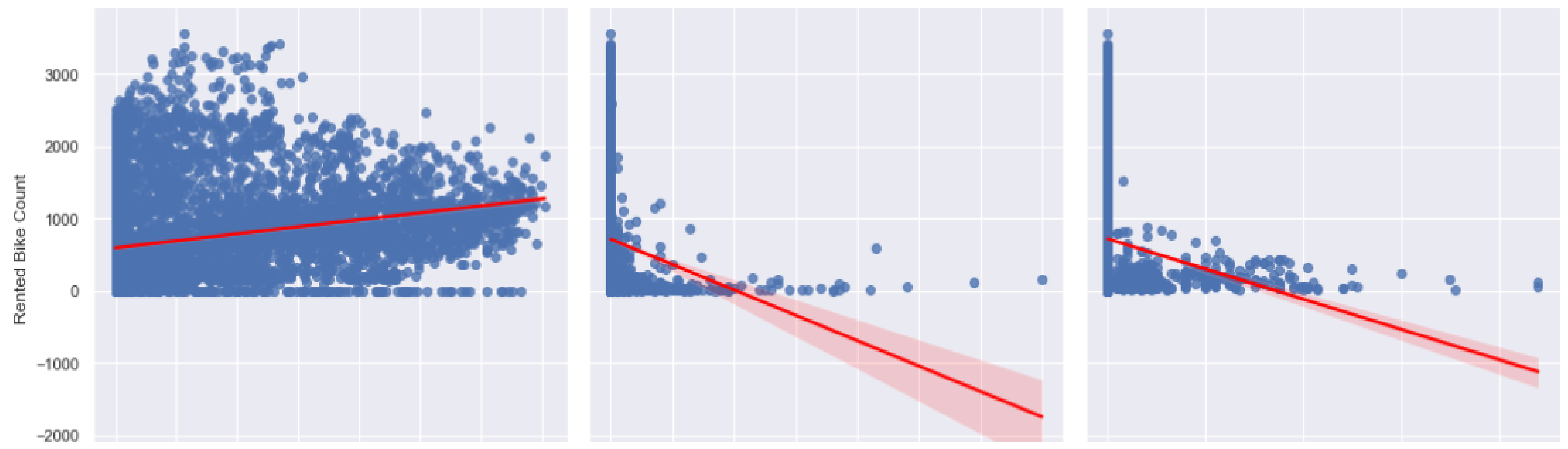
INFLUENCE OF TEMPERATURE ON RENTED BIKES

- This graph shows the positive correlation between temperature of the day and the number of bikes rented.





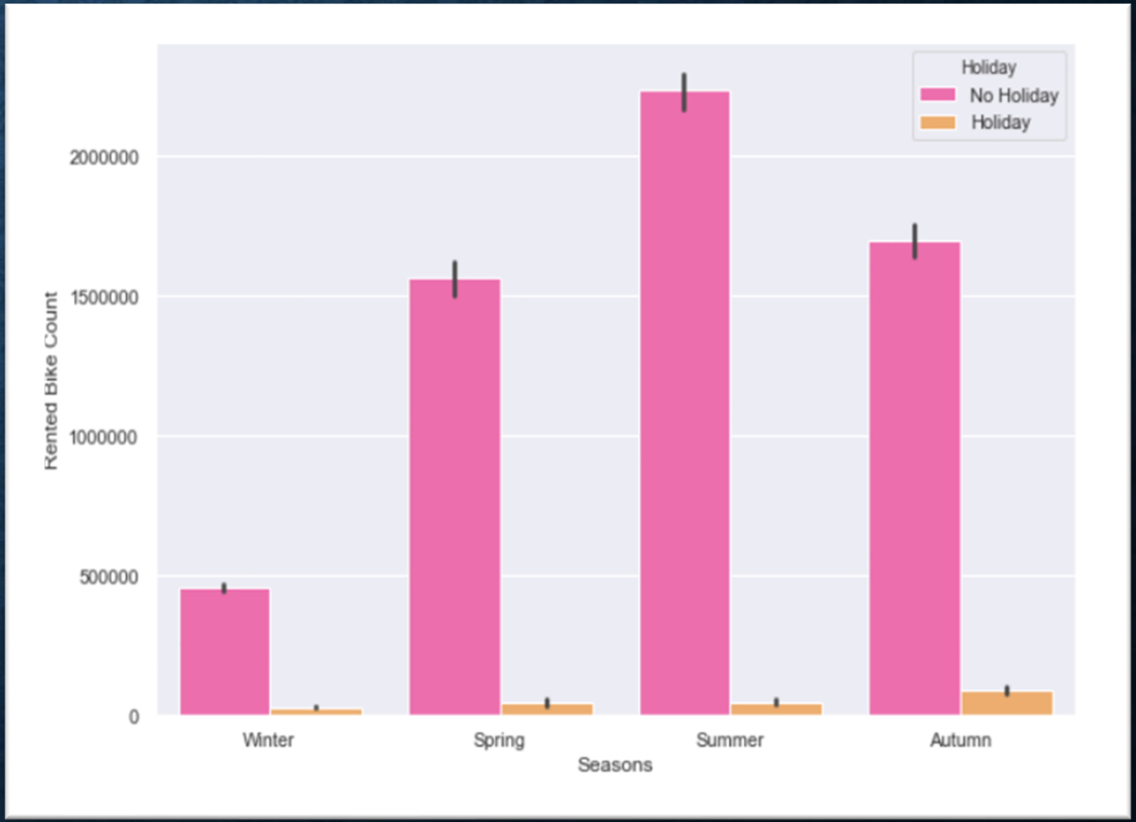
RELATION OF NUMBER OF BIKES RENTED WITH HUMIDITY, WIND SPEED AND VISIBILITY



EFFECT OF SOLAR RADIATION, RAINFALL AND SNOWFALL ON RENTED BIKES

RELATIONSHIP OF RENTED BIKES WITH SEASON AND HOLIDAY

- We can clearly see from the graph that holiday has a significant effect on rented number of bikes
- So we can expect more rented bikes on a non holiday and less rented bikes on a holiday
- When it comes to seasons, we see more number of bikes rented in summer as compared to other seasons



MACHINE LEARNING

- In this project we want to predict the number of rented bikes given features which are in the dataset.
- For machine learning model its better to do some feature engineering and make the dataset appropriate on which the machine learning model can be trained.

FEATURE EXTRACTION

- First, we convert the Date column into proper datetime datatype
- Then we created features of month and day of week to be used for training

Feature Extraction

```
df["Date"] = pd.to_datetime(df["Date"])
```

```
df["Month"] = df["Date"].dt.month
```

```
df["Weekday"] = df["Date"].dt.dayofweek
```

DATA ENCODING

```
season_dict = {"Spring":0,"Summer":1,"Autumn":2,"Winter":3}
holiday_dict = {"No Holiday":0,"Holiday":1}
function_dict = {"Yes":1,"No":0}
```

```
df["Seasons"] = df["Seasons"].map(season_dict)
df["Holiday"] = df["Holiday"].map(holiday_dict)
df["Functioning Day"] = df["Functioning Day"].map(function_dict)
```

- We need to encode the columns so that they can be converted into integer datatype so that we can train the model on them since machine learning models only work with integer and float datatypes.

```python
features.info()
```

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

```
<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)
```

DATA COMPARISON AFTER PROCESSING AND ENCODING

DATA NORMALIZATION

- Data normalization is necessary for better performance of machine learning models
- We have used Min Max Scaler to make sure all the features values are between 0 to 1

Normalization

```
from sklearn.preprocessing import MinMaxScaler
sc = MinMaxScaler()
```

```
sc.fit(X_train)
```

```
MinMaxScaler(copy=True, feature_range=(0, 1))
```

```
X_train_sc = sc.transform(X_train)
X_test_sc = sc.transform(X_test)
```

```
from sklearn.linear_model import LinearRegression
```

```
model_1 = LinearRegression()
model_1.fit(X_train_sc,y_train)
```

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
```

```
pred_1 = model_1.predict(X_test_sc)
least_squares_score = r2_score(y_test,pred_1)
print("Lease Squares Score :",least_squares_score)
```

```
Lease Squares Score : 0.5156853869262186
```

LEAST SQUARES LINEAR REGRESSION

- Training a simple linear regression model on features to predict the number of rented bikes

LASSO REGRESSION WITH GRID SEARCH

```
from sklearn.linear_model import Lasso
```

```
pram_1 = {'alpha':[0.01,0.02,0.05,0.1,1],'max_iter':[1000,3000,5000]}
model_2 = Lasso()
clf_2 = GridSearchCV(model_2,pram_1)
```

```
clf_2.fit(X_train_sc,y_train)
```

```
GridSearchCV(cv=None, error_score=nan,
             estimator=Lasso(alpha=1.0, copy_X=True, fit_intercept=True,
                             max_iter=1000, normalize=False, positive=Fa
                             precompute=False, random_state=None,
                             selection='cyclic', tol=0.0001, warm_start=
             iid='deprecated', n_jobs=None,
             param_grid={'alpha': [0.01, 0.02, 0.05, 0.1, 1],
                         'max_iter': [1000, 3000, 5000]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=Fal
             scoring=None, verbose=0)
```

```
clf_2.best_params_
```

```
{'alpha': 0.05, 'max_iter': 1000}
```

- In lasso regression we have applied grid search on two parameters first number of iterations and second alpha which is for regularization.
- We have found best max iteration number to be 1000 and alpha to be 0.05

POLYNOMIAL AND RIDGE REGRESSION WITH GRID

```
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import Ridge
```

```
poly = PolynomialFeatures(degree=2)
features_poly = poly.fit_transform(features.values)
X_trainp, X_testp, y_train, y_test = train_test_split(features_poly,labels.values,test_size=0.1,random_state=0)
```

```
pram_2 = {'alpha':[10,50,75,100,200]}
model_3 = Ridge()
clf_3 = GridSearchCV(model_3,pram_2)
```

```
clf_3.fit(X_trainp,y_train)
```

```
GridSearchCV(cv=None, error_score=nan,
             estimator=Ridge(alpha=1.0, copy_X=True, fit_intercept=True,
                             max_iter=None, normalize=False, random_state=None,
                             solver='auto', tol=0.001),
             iid='deprecated', n_jobs=None,
             param_grid={'alpha': [10, 50, 75, 100, 200]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring=None, verbose=0)
```

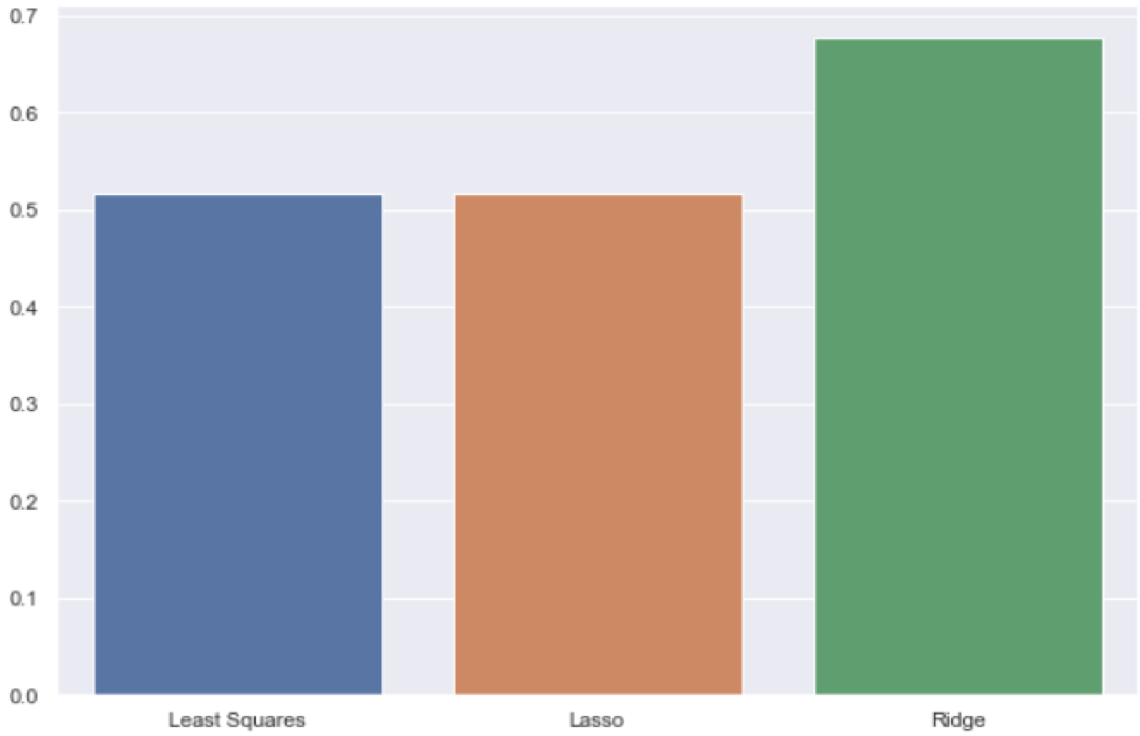
```
clf_3.best_params_
```

```
{'alpha': 75}
```

MODEL PERFORMANCE COMPARISON

```
sns.barplot(["Least Squares","Lasso","Ridge"],[least_squares_score,least_squares_score,ridge_score])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1ce6862a9e8>
```



SAVING THE MODEL TO BE USED IN DJANGO API

```
model_3=clf_3.best_estimator_
```

```
pred_3 = model_3.predict(X_testp)
ridge_score = r2_score(y_test,pred_3)
print("Ridge Regression Score :",ridge_score)
```

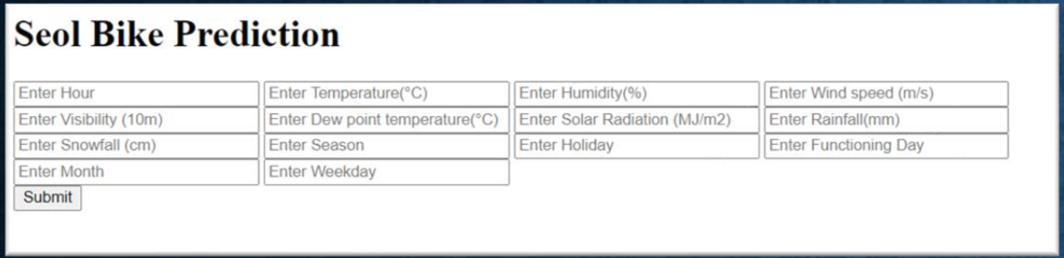
```
Ridge Regression Score : 0.6771534930531571
```

```
model_save = open("model.dat","wb")
pickle.dump(model_3,model_save)
model_save.close()
```

- We have picked the best performing model and saved the model to be used in Django API

DJANGO API

- First, we created a Django project and setup our files which are required for making of the api.
- We created the html page to take the required input values which have to be used for the prediction of the rented bikes.
- After making the html page we created our views files where we processed all the input variables and use the saved model to make the prediction and return the Jason response of that prediction.



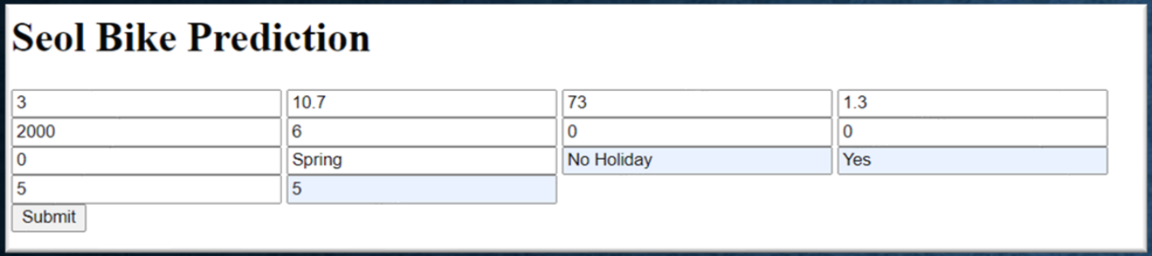

```
(djangodev) D:\PyCharm Projects\1_Work\Project_Abdoul\model_api>python manage.py runserver
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).
January 10, 2021 - 16:14:46
Django version 3.1.4, using settings 'model_api.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.
```

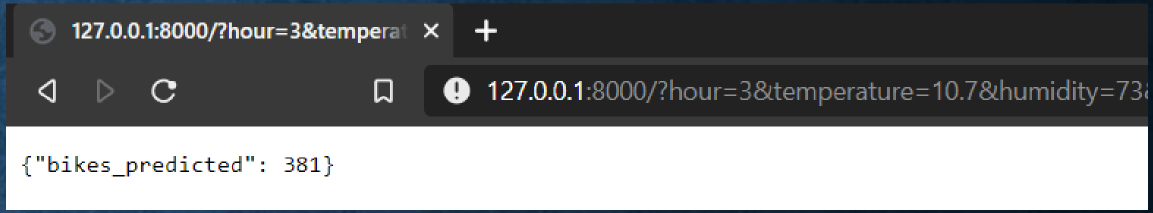
RUNNING SERVER

```python
        functioning_day = function_dict[functioning_day]
        month = float(month)
        weekday = float(weekday)
        print(hour, temperature, humidity,wind_speed,visibility,dew_point_temperature,
              solar_radiation,rainfall,snowfall,seasons,holiday,functioning_day,month,weekday)

        # print(os.getcwd())
        model = pickle.load(open("model.dat",'rb'))

        poly = pickle.load(open("poly.dat",'rb'))

        poly_feature = poly.transform([[hour, temperature, humidity,
                                        wind_speed,visibility,dew_point_temperature,
                                        solar_radiation,rainfall,snowfall,seasons,holiday,
                                        functioning_day,month,weekday]])
        # print(poly_feature)
        # Making prediction
        bikes = int(model.predict(poly_feature))

        return JsonResponse({'bikes_predicted':bikes})
    else:
        return render(request,"seol_bike_prediction.html")
```

```python
from django.shortcuts import render
import sklearn
from django.http import JsonResponse
import pickle
import os

def make_prediction(request):

    if len(request.GET) == 15:
        d = request.GET
        hour, temperature, humidity,wind_speed,visibility,dew_point_temperature,\
        solar_radiation,rainfall,snowfall,seasons,holiday,functioning_day,month,weekday,_ = d.values()

        season_dict = {"Spring": 0, "Summer": 1, "Autumn": 2, "Winter": 3}
        holiday_dict = {"No Holiday": 0, "Holiday": 1}
        function_dict = {"Yes": 1, "No": 0}
        hour = float(hour)
        temperature = float(temperature)
        humidity = float(humidity)
        wind_speed = float(wind_speed)
        visibility = float(visibility)
        dew_point_temperature = float(dew_point_temperature)
        solar_radiation = float(solar_radiation)
        rainfall = float(rainfall)
        snowfall = float(snowfall)
        seasons = season_dict[seasons]
        holiday = holiday_dict[holiday]
        functioning_day = function_dict[functioning_day]
```

TAKING AND PROCESSING THE HTML INPUT





PREDICTION USING DJANGO API

- After entering all the details for that day, the user will click on Submit button the get the prediction on number of bikes for that day.