Capstone Project

Regression - Bike Sharing Demand Prediction

By Paras

Summary & Problem Statement

A bike sharing service is the one in which a person lends bikes for other people on rent. This can sometimes be a shop where the lender receives payment for providing bikes for use. Nowadays we have an unmanned system in which bikes are available at docks which the user can go to & rent a bike & pay digitally online. In this project we are going to analyse the given dataset for valuable insights that might help the company improve their profits by analyzing various factors given in the dataset. We are also going to deploy a regression model to predict the count of rented bikes which may include a few more steps.

Bike sharing is an important means of travelling in today's world , so it's quite necessary for the bike rental services to provide their services on time. We need to create a system which can predict Bike rental demand based on users behaviour.

Understanding the data

There are around 8760 entries with the following variables :-

Fields	Description
Date	Date
Hour	Hour of the day (0–23)
Temperature	Temperature of the day
Humidity	Humidity measure
Windspeed	Windspeed
Visibility	Visibility measure
Dew Point Temperature	Dew Point Temperature Measure
Solar Radiation	Solar Radiation Measure
Rainfall	Rainfall in mm
Snowfall	Snowfall measure
Seasons	1 = spring, 2 = summer, 3 = fall, 4 = winter
Holiday	Whether a holiday or not
Functional Day	Whether a functional day or not

Workflow

We will approach our project in the following steps :-

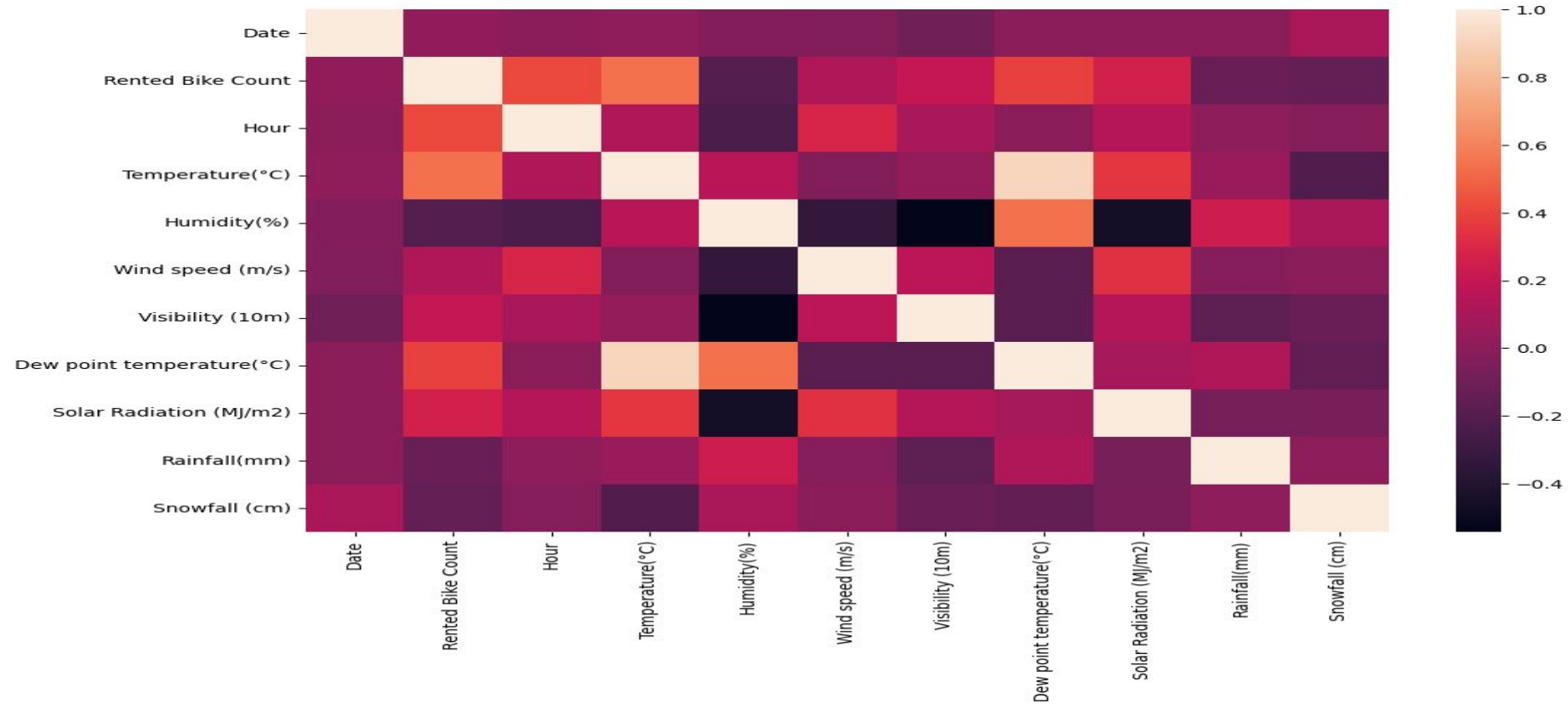
Data Discovery - In this step we import data & understand the data, its variables, values etc

Data Wrangling - In this step we modify the data so that it becomes ready for next procedures

Exploratory Data Analysis - in this step we create a bunch of charts & graphs to gain insights on the dataset that can help us making better decisions for our rental business.

Model Building - in this step we will try multiple machine learning model & test it to choose the best one. We will also do hyperparameter tuning to find the best parameters for our selected model

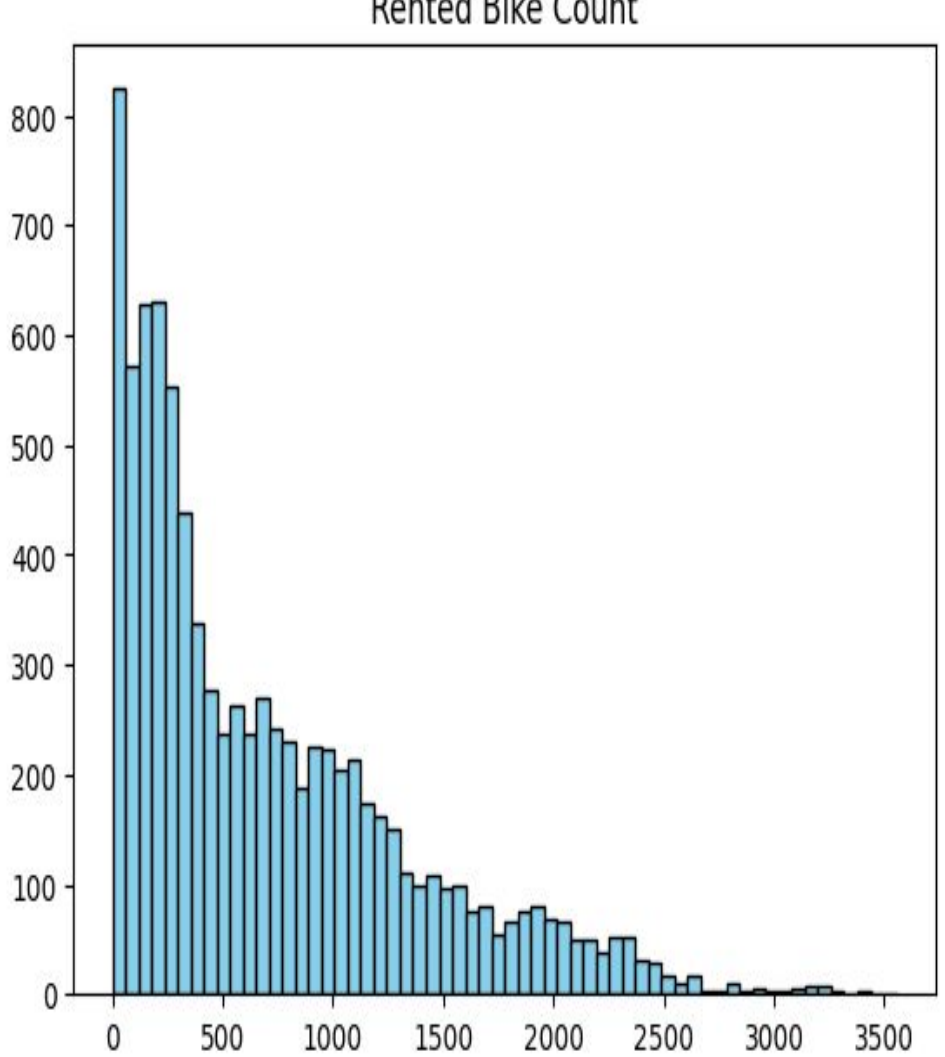
Correlation heatmap



Univariate analysis - Rented bike count

Histograms are better for univariate analysis of numerical values

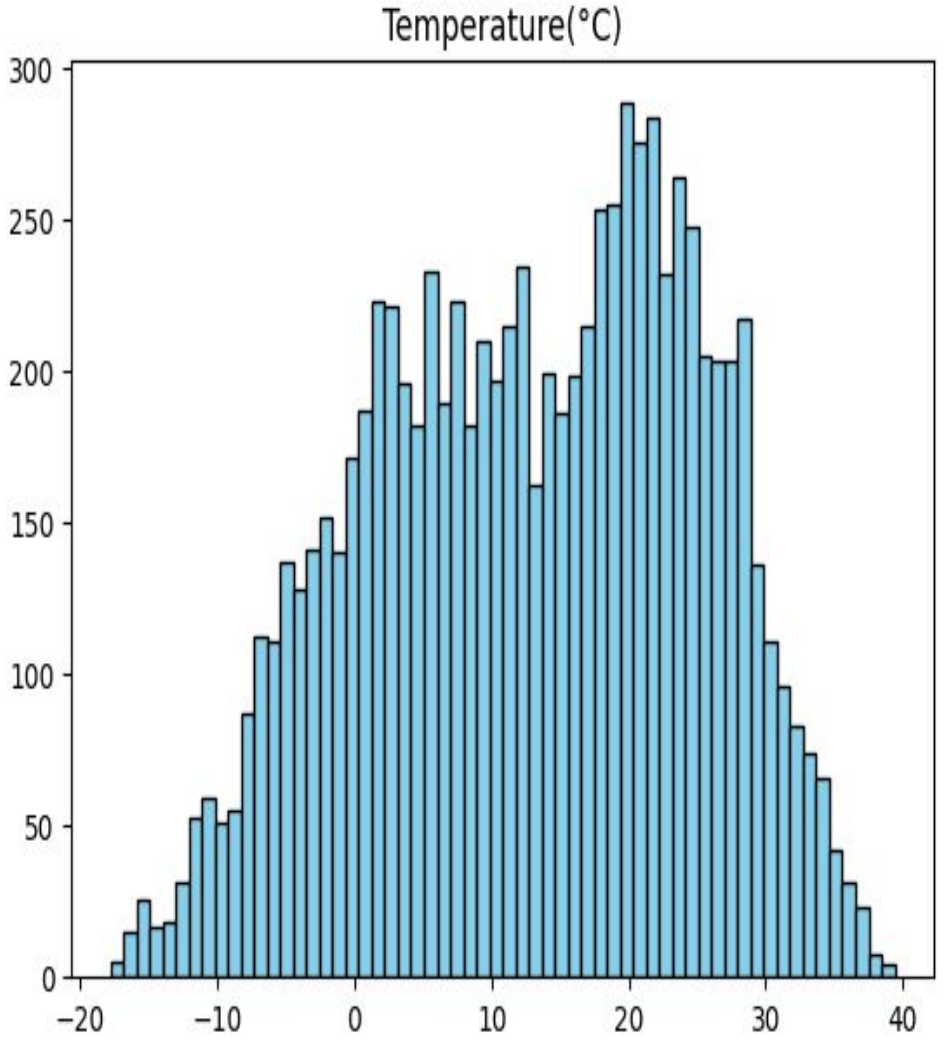
Mostly, the count of rented bikes lies between , 0 to 500



Temperature

Histograms are best for univariate analysis of numerical values

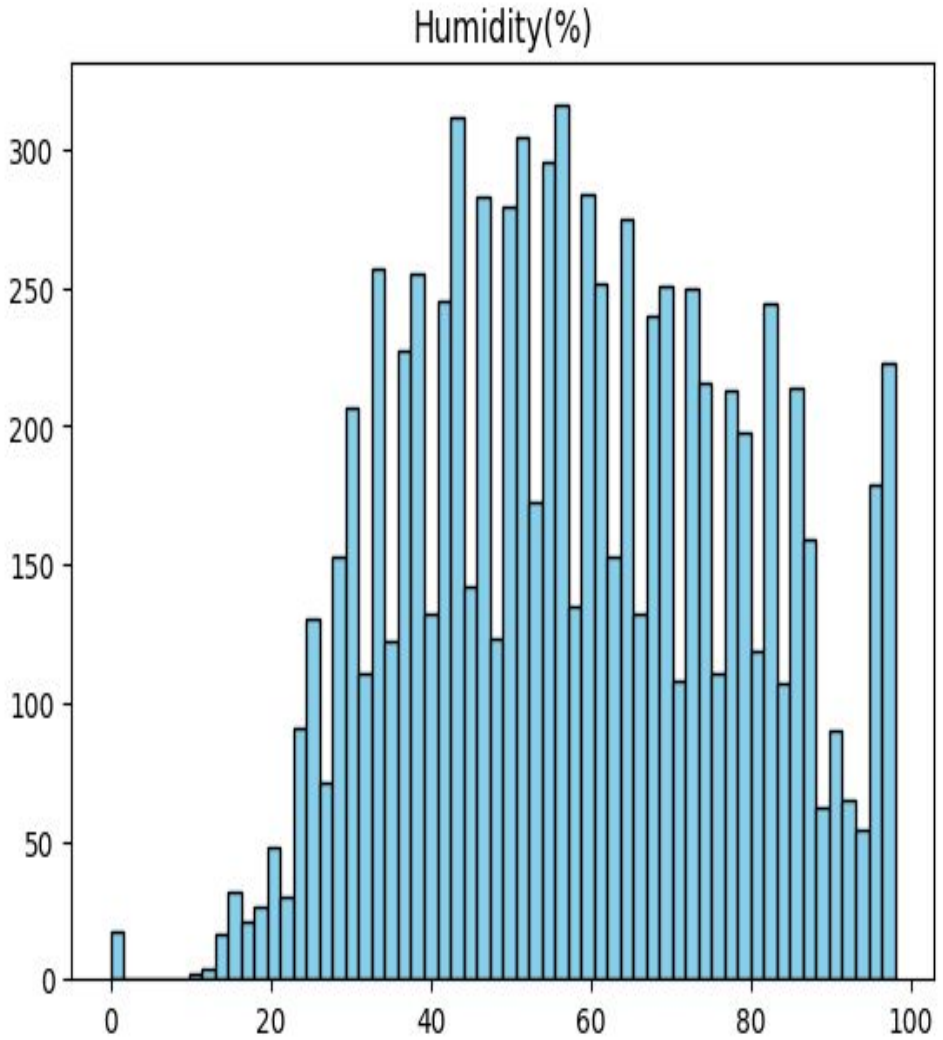
usually the temperature lies between 0 to 30 degree celsius



Humidity

Histograms are best for univariate analysis of numerical values

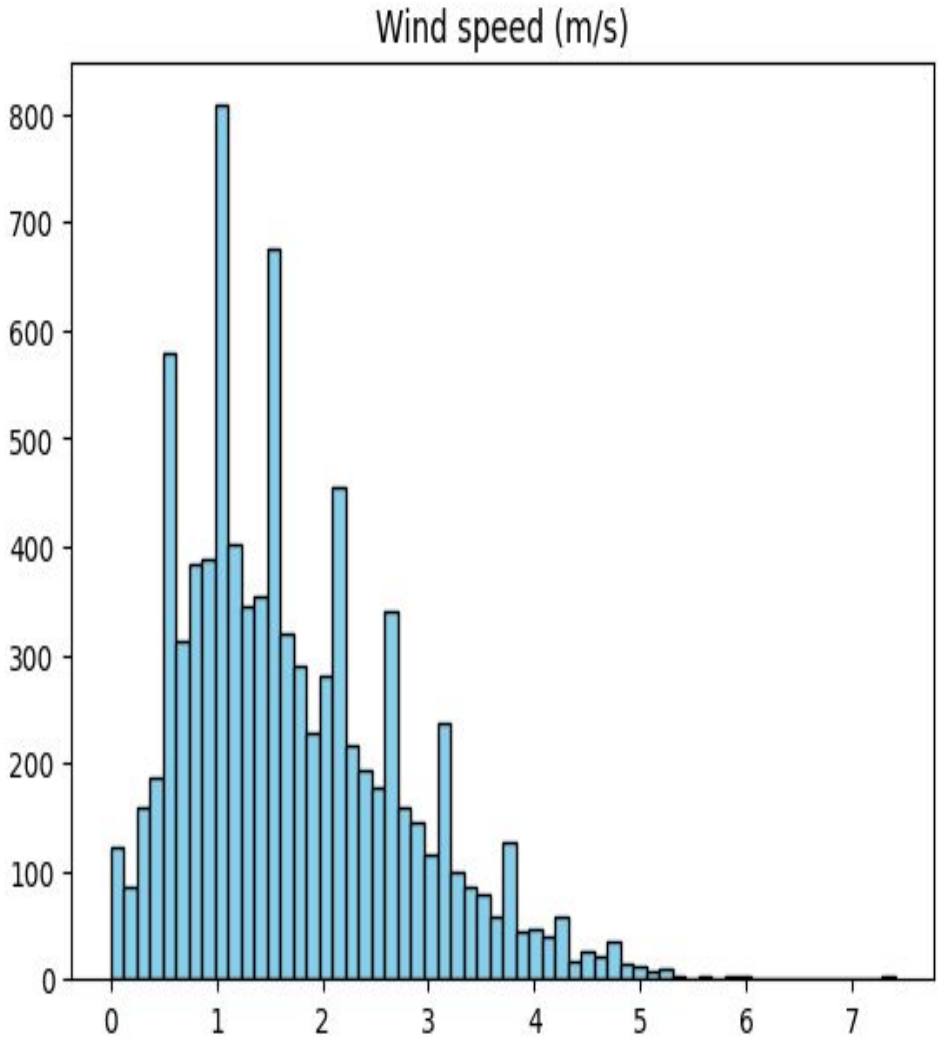
mostly humidity lies between 40-80%



Wind Speed

Histograms are best for univariate analysis of numerical values

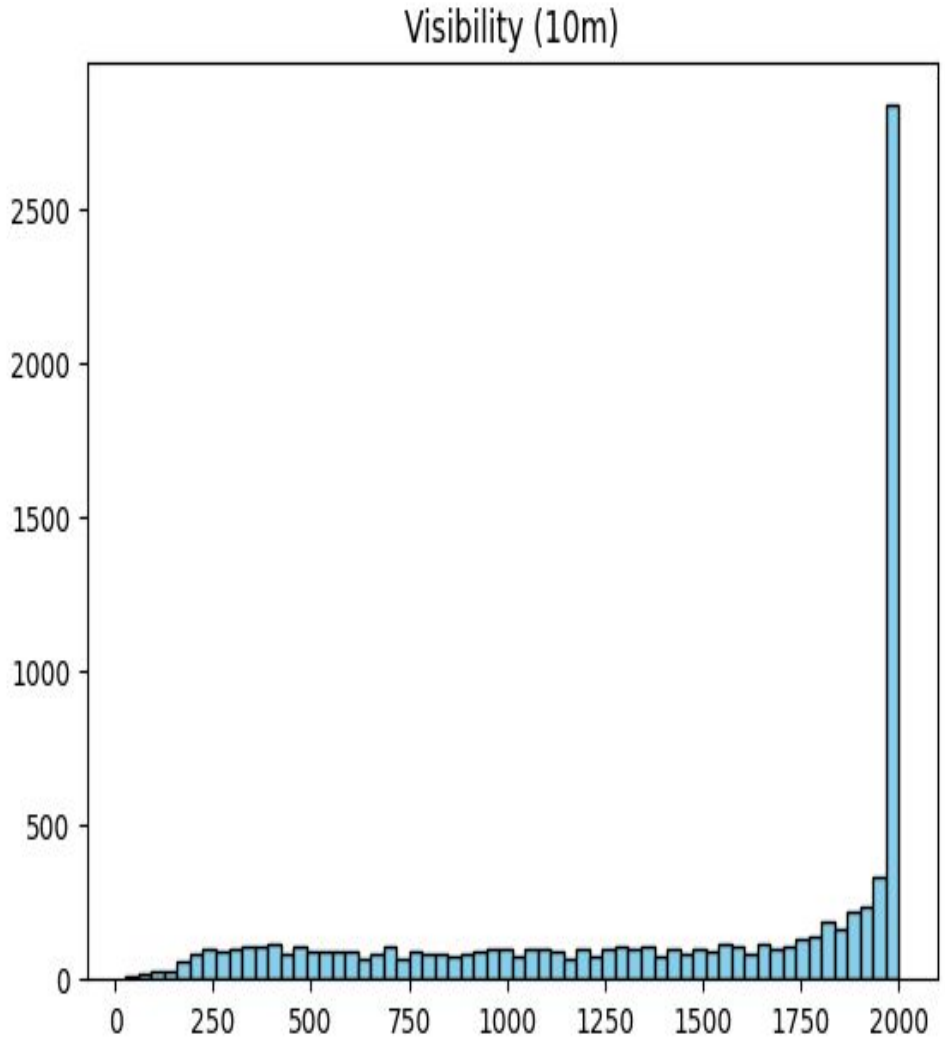
Usually the wind speed is between 0 -2 m/s



Visibility

Histograms are best for univariate analysis of numerical values

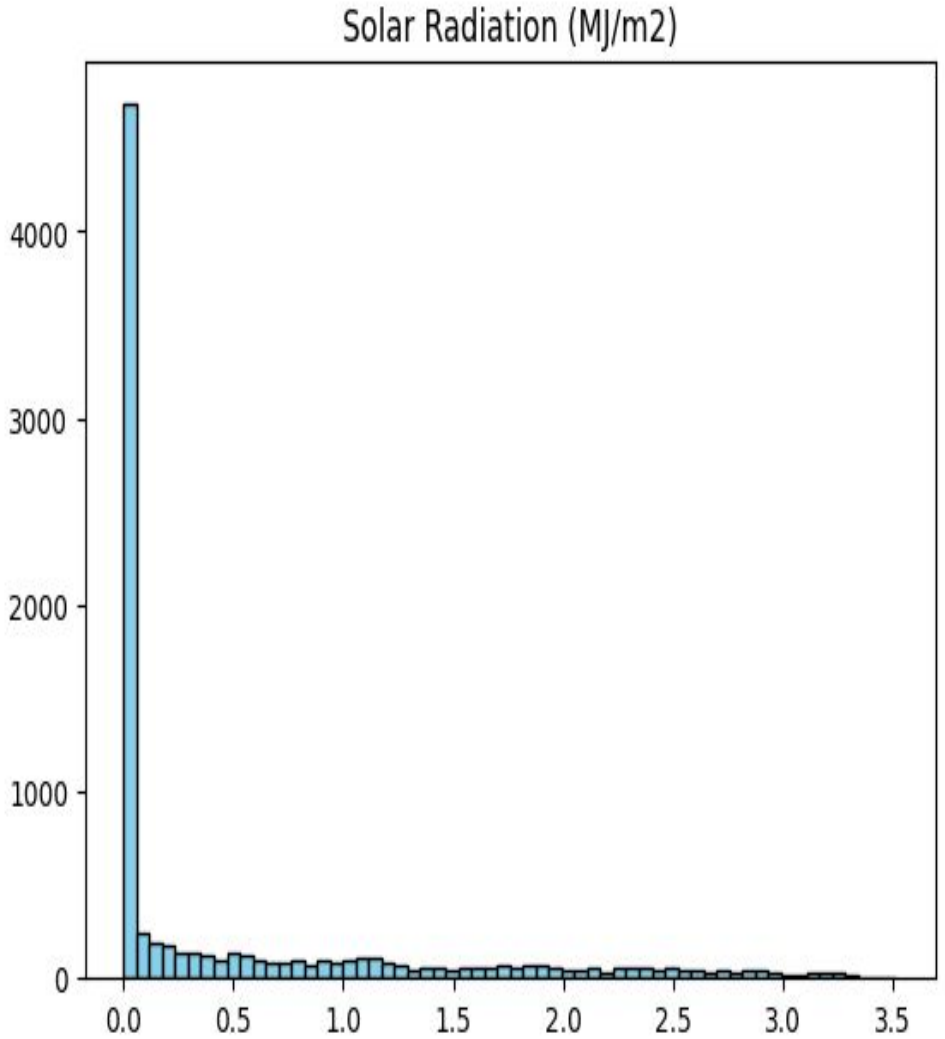
People are more likely to rent when visibility is highest at 2000



Solar radiation

Histograms are best for univariate analysis of numerical values

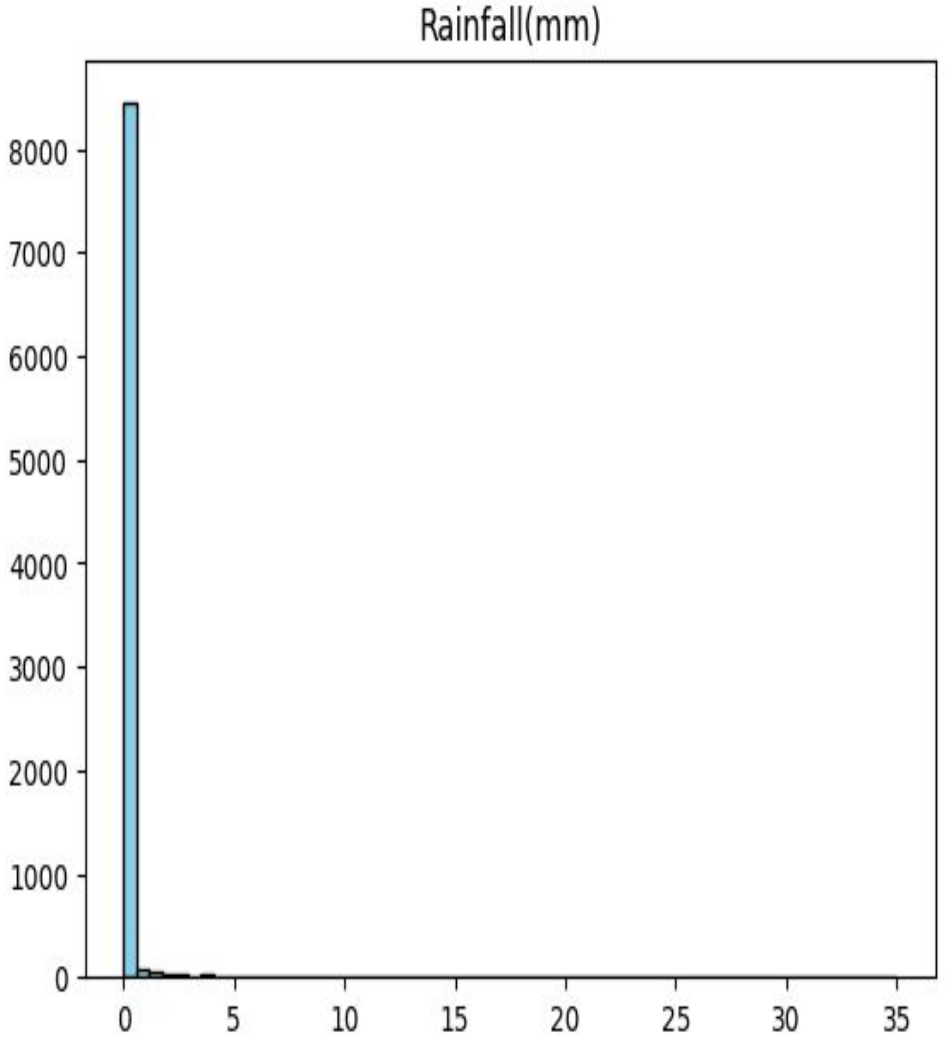
People prefer to rent bikes when there is least solar radiation



RainFall

Histograms are best for univariate analysis of numerical values

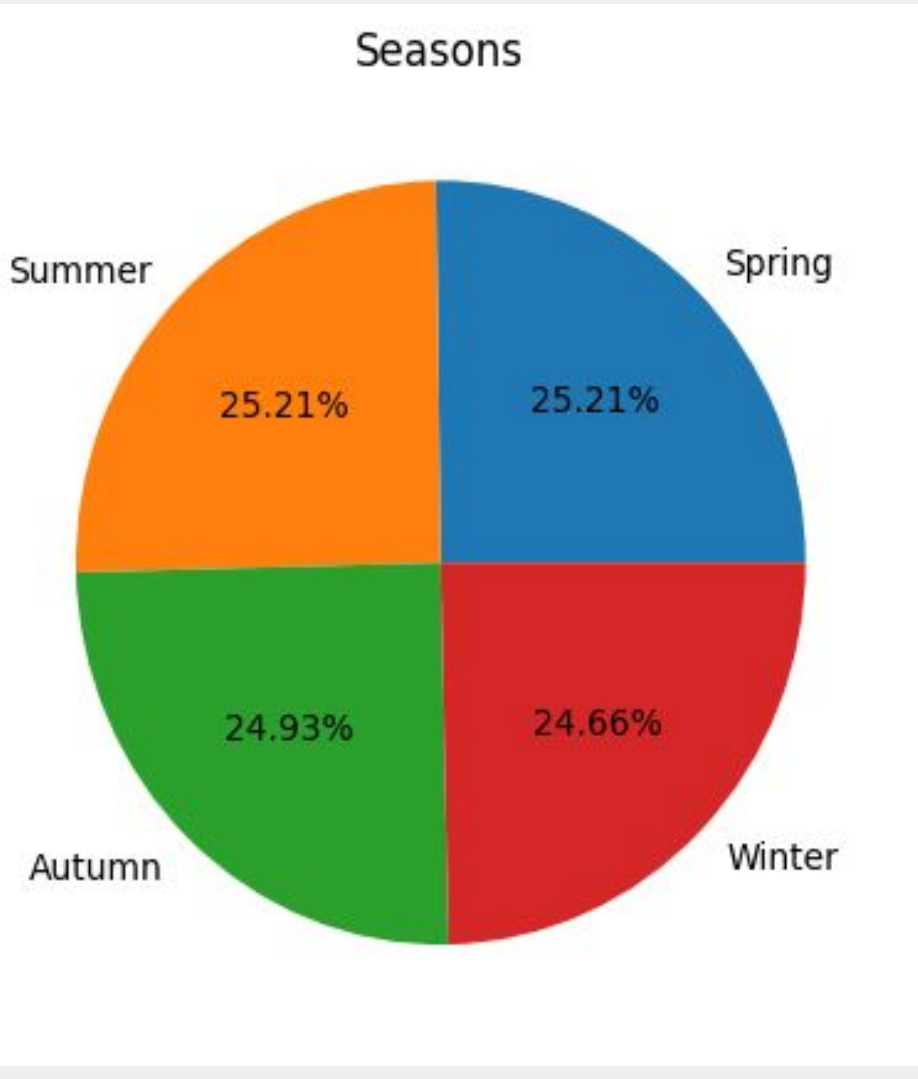
People prefer to rent when the rainfall is least



Seasons

Pie chart is best to display the categorical values in the form of parts-to-whole manner

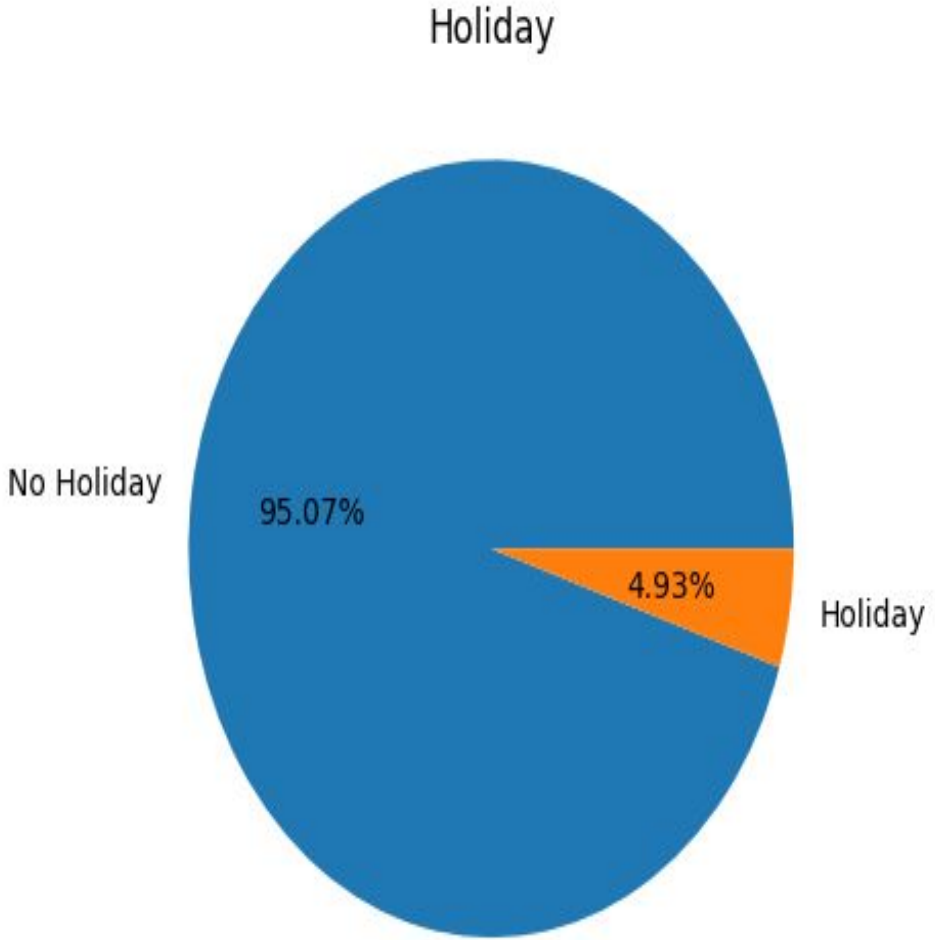
Winter has lowest biker rent count & summers has highest rent count, probably because days are shorter in winter & longer in summers



Holiday

Pie chart is best to display the categorical values in the form of parts-to-whole manner

there's more demand for bikes when there is no holidays

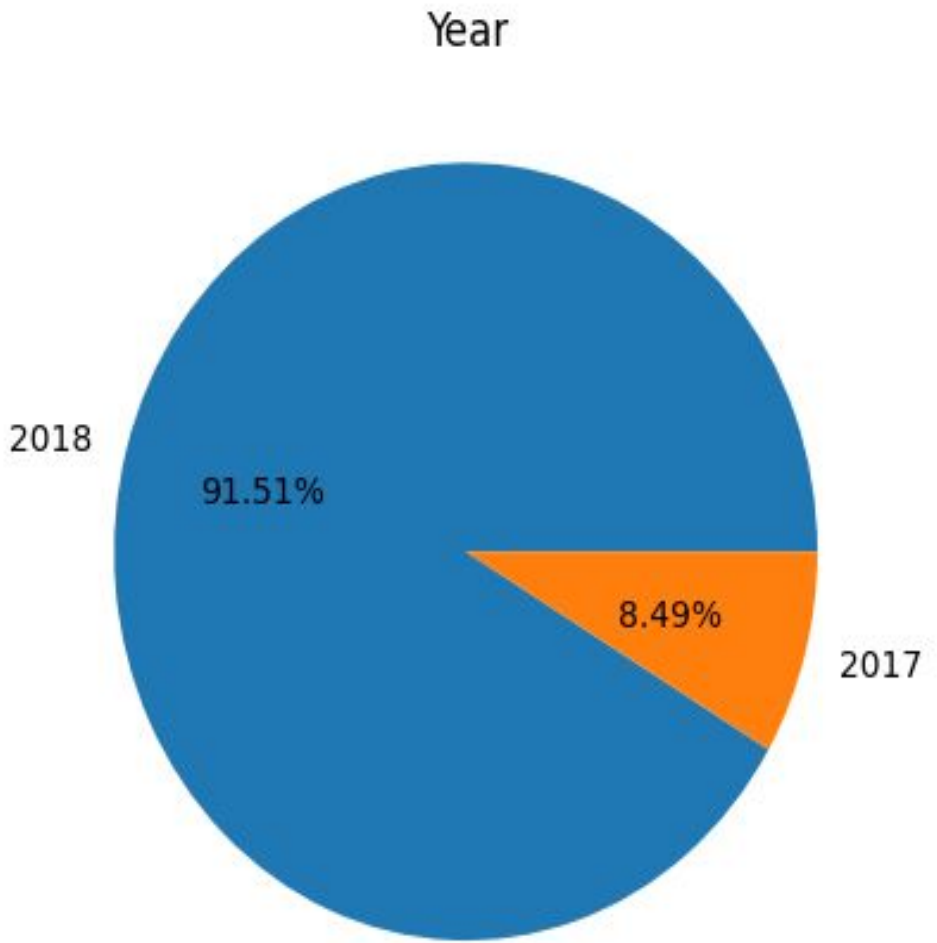


Year

Pie chart is best to display the categorical values in the form of parts-to-whole manner

. What is/are the insight(s) found from the chart?

2018 was busier than 2017 when it comes to bike demand



Total rented bike count per each categorical values

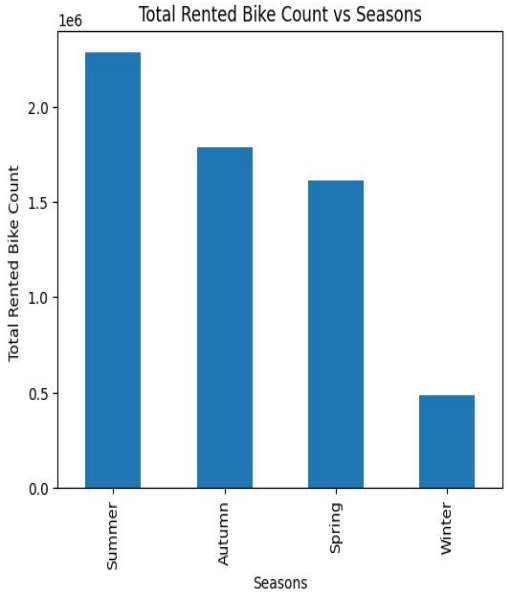
summer has more demand than any other seasons

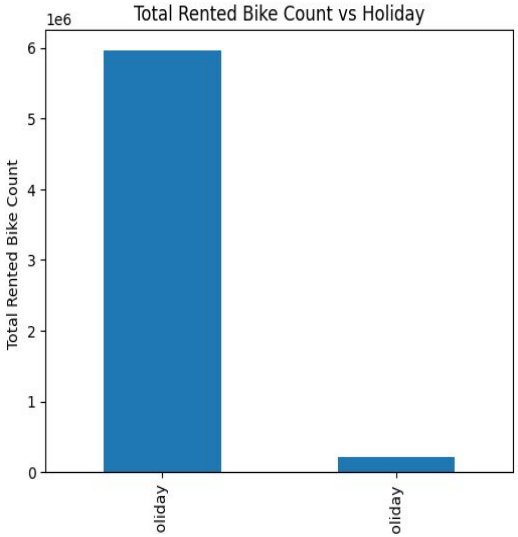
There's more demand when there is no holiday

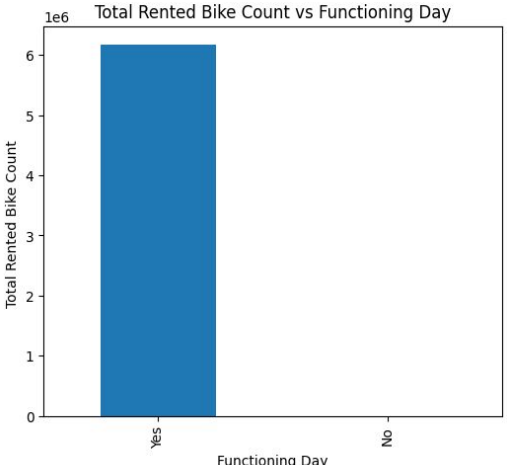
Functioning days has more demands than non functioning days

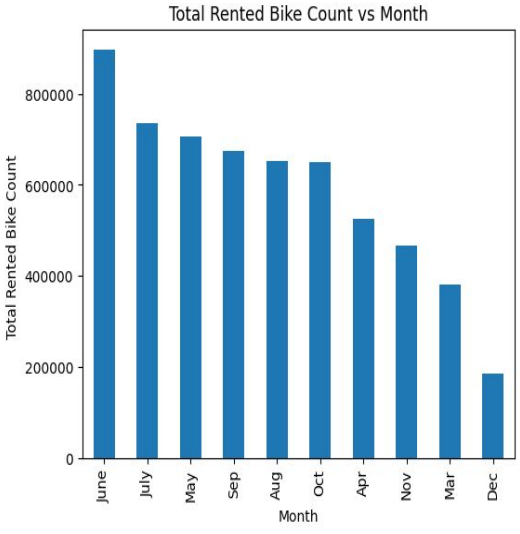
June, july, may has highest demand than other months, probably because these are summer months when days are longer

2018 has seen more demand than 2017









Model Implementation - Dummy Encoding

The machine learning models cannot understand categorical values, so we need to convert these categorical values into numerical values , this is where encoding comes in. In encoding we create columns from unique values from the column & place 0 & 1 based on the category in each entry. In dummy encoding, 1 column is skipped to reduce curse of dimensions.The next slide shows how new columns are made by encoding.

7. ML Model Implementation

Encoding of categorical columns, dummy encoding

```
[ ] df_dum = pd.get_dummies(df)

    df_dum.head(2)
```

	Date	Rented Bike Count	Hour	Temperature(°C)	Humidity(%)	Wind speed (m/s)	Visibility (10m)	Dew point temperature(°C)	Solar Radiation (MJ/m2)	Rainfall(mm)	...	Month_Jan	Month_July	Month_June	Month_Mar	Month_May	Month_Nov	Mo
0	1	254	0	-5.2	37	2.2	2000	-17.6	0.0	0.0	...	0	0	0	0	0	0	
1	1	204	1	-5.5	38	0.8	2000	-17.6	0.0	0.0	...	0	0	0	0	0	0	

2 rows × 33 columns

```
[ ] df_dum.columns , len(df_dum.columns)
```

```
(Index(['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_Autumn', 'Seasons_Spring', 'Seasons_Summer', 'Seasons_Winter',
       'Holiday_Holiday', 'Holiday_No Holiday', 'Functioning Day_No',
       'Functioning Day_Yes', 'Month_Apr', 'Month_Aug', 'Month_Dec',
       'Month_Feb', 'Month_Jan', 'Month_July', 'Month_June', 'Month_Mar',
       'Month_May', 'Month_Nov', 'Month_Oct', 'Month_Sep', 'Year_2017',
       'Year_2018'],
      dtype='object'),
 33)
```

Defining features & train-test split

Here we will define inputs features & output/target variables, then we will do the train test split. We have kept the test size as 0.2 that means 20% of the data will be kept as testing data which will be used in evaluation of the model.

Defining x & y, then train test split

```
[ ]  x = df_dum.drop(['Rented Bike Count'], axis = 1)

     y = df_dum['Rented Bike Count']
```

```
[ ]  x.shape , y.shape
```

```
     ((8760, 32), (8760,))
```

train test split

```
[ ]  from sklearn.model_selection import train_test_split

     x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42)
```

Model Implementation

ML Models

We used the following ml algorithms

linear regression

ridge regression

lasso regression

support vector regression

decision tree regressor

random forest regressor

xgboost regressor

Model Evaluation on training data & testing data

```
model_eval_df = pd.DataFrame(model_train_test_r2_list)
model_eval_df.columns = ['regressor' , 'train_sc' , 'test_sc']
```

```
[ ] model_eval_df
```

	regressor	train_sc	test_sc
0	LinearRegression()	0.593731	0.573092
1	Ridge()	0.593730	0.573123
2	Lasso()	0.593147	0.572377
3	SVR()	-0.045296	-0.051177
4	DecisionTreeRegressor()	1.000000	0.738226
5	(DecisionTreeRegressor(max_features=1.0, rando...	0.983747	0.871077
6	XGBRegressor(base_score=None, booster=None, ca...	0.976386	0.886134

We can see that xgboost regressor works the best , though random forest regressor is a close second

Descision tree is overfitted [good score on training data but bad score on testing data]

and the rest are just very bad

Performance metric used - R2 score

Here we have used r2 score for model evaluation because it is easier to interpret. Its value lies between 0 to 1. The closer the value is to 1 , the better the model is.

Hyperparameter tuning

Hyperparameter tuning is a way by which we can tune our trained model by finding the best parameters for given model. Here we will use randomizedsearch cv wit r2 score for evaluation

```
[ ] # Number of trees
    n_estimators = list(range(100 , 1300 , 100)) # same as prev algo

    # Various learning rate parameters
    learning_rate = ['0.05','0.1', '0.2','0.3','0.5','0.6']

    # Maximum number of levels in tree
    max_depth = list(range(5 , 31 , 6))

    #Subssample parameter values
    subsample=[0.7,0.6,0.8]

    # Minimum child weight parameters
    min_child_weight=[3,4,5,6,7]


    # Create the random grid
    random_grid = {'n_estimators': n_estimators,
                   'learning_rate': learning_rate,
                   'max_depth': max_depth,
                   'subsample': subsample,
                   'min_child_weight': min_child_weight}

    print(random_grid)
```

```
{'n_estimators': [100, 200, 300, 400, 500, 600, 700, 800, 900, 1000, 1100, 1200], 'learning_rate': ['0.05', '0.1', '0.2', '0.3', '0.5', '0.6'], 'max_depth': [5, 11, 17, 2
```

```
[ ]
    xgbr_random = RandomizedSearchCV(estimator = xgbr, param_distributions = random_grid,scoring='r2', n_iter = 10, cv = 2, verbose=2, random_state=42, n_jobs = 1)

    xgbr_random.fit(x_train,y_train)
```

Best score & best parameters

```
[ ]  xgbr_random.best_score_ # r2 score
```

```
0.8860230434691163
```

```
[ ]  xgbr_random.best_params_
```

```
{'subsample': 0.6,
 'n_estimators': 1200,
 'min_child_weight': 5,
 'max_depth': 17,
 'learning_rate': '0.05'}
```

Score analysis

There seems to be not much improvements after hyperparameter tuning.

Thank you