# BIKE RENTING PROJECT

*MICHAEL AJITH*

**05 JUNE 2019**

# Contents

# Chapter 1

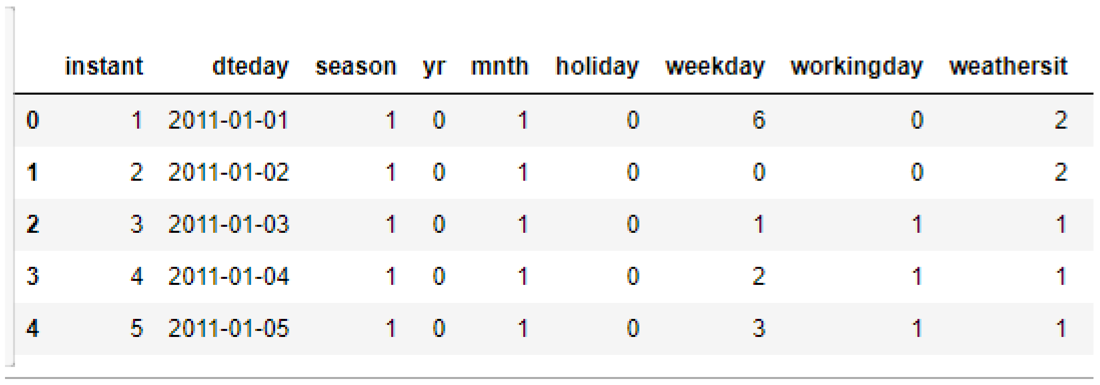
# Introduction

## 1.1 Problem Statement

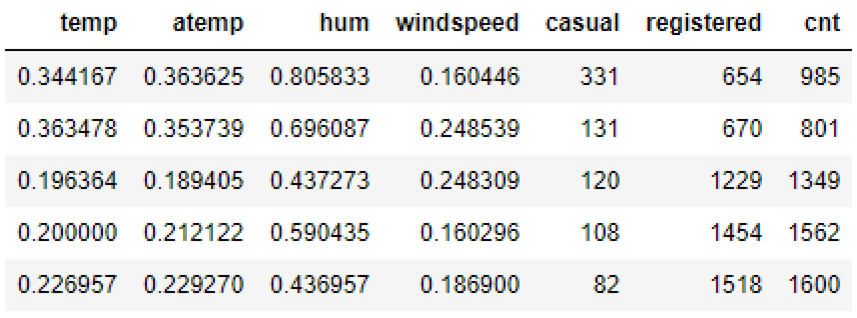
The objective of this Case is to Predication of bike rental count on daily based on the environmental and seasonal settings.

## 1.2 Data

Overview of top five observation of the bike dataset



| | instant | dteday | season | yr | mnth | holiday | weekday | workingday | weathersit |
|---|---|---|---|---|---|---|---|---|---|
| **0** | 1 | 2011-01-01 | 1 | 0 | 1 | 0 | 6 | 0 | 2 |
| **1** | 2 | 2011-01-02 | 1 | 0 | 1 | 0 | 0 | 0 | 2 |
| **2** | 3 | 2011-01-03 | 1 | 0 | 1 | 0 | 1 | 1 | 1 |
| **3** | 4 | 2011-01-04 | 1 | 0 | 1 | 0 | 2 | 1 | 1 |
| **4** | 5 | 2011-01-05 | 1 | 0 | 1 | 0 | 3 | 1 | 1 |



| temp | atemp | hum | windspeed | casual | registered | cnt |
|---|---|---|---|---|---|---|
| 0.344167 | 0.363625 | 0.805833 | 0.160446 | 331 | 654 | 985 |
| 0.363478 | 0.353739 | 0.696087 | 0.248539 | 131 | 670 | 801 |
| 0.196364 | 0.189405 | 0.437273 | 0.248309 | 120 | 1229 | 1349 |
| 0.200000 | 0.212122 | 0.590435 | 0.160296 | 108 | 1454 | 1562 |
| 0.226957 | 0.229270 | 0.436957 | 0.186900 | 82 | 1518 | 1600 |

We have total of 16 columns, in which we have 13 independent variables and 3 dependent variables. **‘casual’, ‘registered’ and ‘cnt’** (dependent variables)are the counting of renting of bikes for a particular day. Casual counting is for non-registered customer, registered counting is for registered customer and cnt is for total counting. i.e. casual + registered.

so we have to choose the target variable as **cnt** and we have to prepare a model to predict the count of bikes on daily basis based on environmental. In the dataset target variable is continuous in nature, so this is a regression problem.

**Attribute Information:**

1. **instant:** Record index

2. **dteday:** Date

3. **season:** Season (1:springer, 2:summer, 3:fall, 4:winter)

4. **yr:** Year (0: 2011, 1:2012)

5. **mnth:** Month (1 to 12)

6. **holiday:** weather day is holiday or not (extracted fromHoliday Schedule)

7. **weekday:** Day of the week

8. **workingday:** If day is neither weekend nor holiday is 1, otherwise is 0.

9. **weathersit:** (extracted fromFreemeteo)

i: Clear, Few clouds, Partly cloudy, Partly cloudy

ii: Mist + Cloudy, Mist + Broken clouds, Mist + Few clouds, Mist

iii: Light Snow, Light Rain + Thunderstorm + Scattered clouds, Light Rain + Scattered clouds

iv: Heavy Rain + Ice Pallets + Thunderstorm + Mist, Snow + Fog

10. **temp**: Normalized temperature in Celsius. The values are derived via (t-t_min)/(t_max-t_min),t_min=-8, t_max=+39 (only in hourly scale)

11. **atemp**: Normalized feeling temperature in Celsius. The values are

derived via (t-t_min)/(t_maxt_min), t_min=-16, t_max=+50 (only in hourly scale)

12. **hum**: Normalized humidity. The values are divided to 100 (max)

So, we have time series in our dataset, every day detail is given with environment condition for 2011 and 2012 with column name ‘dteday’, ‘month’, ‘Day of the week’. We have environmental conditions like season, weather, temperature, humidity and windspeed and as we know these conditions affect a person, whether he/she opt for bike renting or not for that day/condition.

# Chapter 2

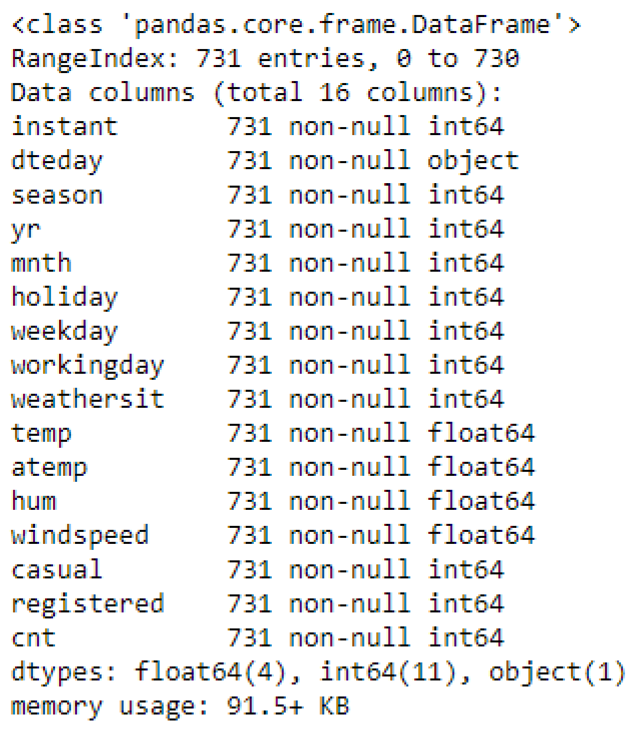
# Methodology

## 2.1 Data Preprocessing: (Exploratory Data Analysis)

Data preprocessing is a data mining technique that involves transforming raw data into an understandable format. Real-world data is often incomplete, inconsistent, and/or lacking in certain behaviors or trends, and is likely to contain many errors. Data preprocessing is a proven method of resolving such issues.

### 2.1.1 Understanding the Data:

Overview of our data and datatypes



```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 731 entries, 0 to 730
Data columns (total 16 columns):
instant       731 non-null int64
dteday        731 non-null object
season        731 non-null int64
yr            731 non-null int64
mnth          731 non-null int64
holiday       731 non-null int64
weekday       731 non-null int64
workingday    731 non-null int64
weathersit    731 non-null int64
temp          731 non-null float64
atemp         731 non-null float64
hum           731 non-null float64
windspeed     731 non-null float64
casual        731 non-null int64
registered    731 non-null int64
cnt           731 non-null int64
dtypes: float64(4), int64(11), object(1)
memory usage: 91.5+ KB
```

We have datatype as object for dteday and rest others have int and float. Time based variables:

‘dteday’, ‘yr’, ‘mnth’, ‘season’, ‘weekday’. Now what should be consider for them either continuous or categorical?

‘**dteday**’ has date of everyday, so all unique it could not be categorical. **‘yr**’ we have two value 0 for 2010 and 1 for 2011, so we can consider it as categorical.

‘**mnth**’: can we consider a month like jan, 2010 and jan 2011 as single category? For a category condition need to be same, like in a school two student are from 5th standard and their class category would be 5, i.e. same for both. We have bike renting dataset and situation for jan 2010 and jan 2011 would be different. So, with one point of view it could be same category for a month but with other point of condition could not be same for jan 2010 and jan 2011. It has 12 values, so consider it as an category , will introduce 11 dimensions to our dataset. So, we will make bins for this column in feature engineering section.

‘**season**’ has four unique values, so we would consider it as category.

‘**weekday**’ : Its same like mnth, so we will make bins for this in feature engineering section.

**Categorical:**

Holiday and working day having value 0 for non-holiday/non-working day 1 for opposite. So it would be our categorical variable.

‘**weathersit**’: it containing 4 unique values, defining different four different condition of weather. Conditions are based on Freemeteo. Our dataset containing only three conditions.

**Continuous Variable:**

‘temp’, ‘atemp’, ‘hum’, ‘windspeed’ are continuous values, and in our dataset they are in normalized format.

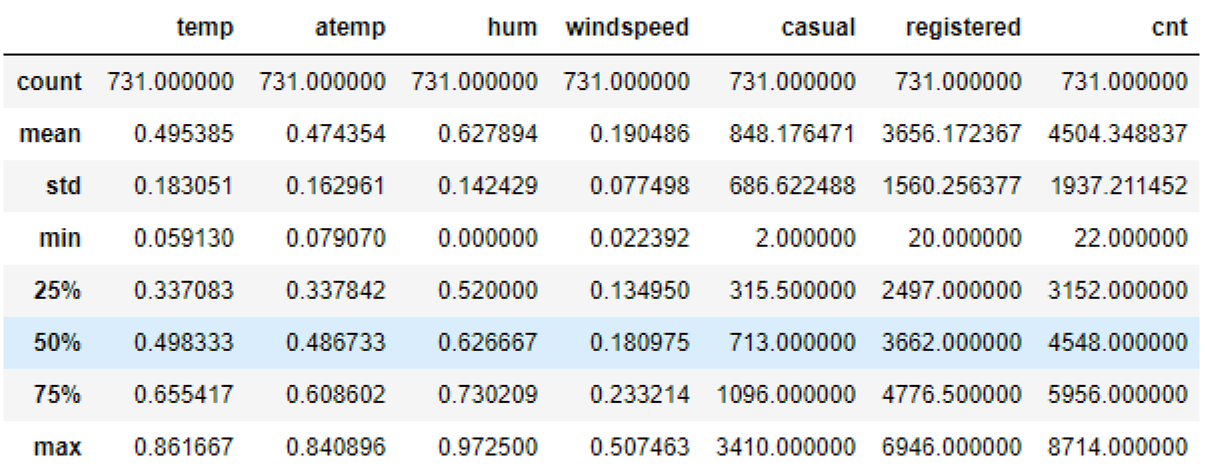
**Target variable:**

‘casual’, ‘registered’ and ‘cnt’ are our target variables and in continuous form.

Casual counting is for non-registered customer, registered counting is for registered customer and cnt is for total counting. i.e. casual + registered.

## Analysis of Numerical Variables:

Looking at summary for our numerical variables



| | temp | atemp | hum | windspeed | casual | registered | cnt |
|---|---|---|---|---|---|---|---|
| count | 731.000000 | 731.000000 | 731.000000 | 731.000000 | 731.000000 | 731.000000 | 731.000000 |
| mean | 0.495385 | 0.474354 | 0.627894 | 0.190486 | 848.176471 | 3656.172367 | 4504.348837 |
| std | 0.183051 | 0.162961 | 0.142429 | 0.077498 | 686.622488 | 1560.256377 | 1937.211452 |
| min | 0.059130 | 0.079070 | 0.000000 | 0.022392 | 2.000000 | 20.000000 | 22.000000 |
| 25% | 0.337083 | 0.337842 | 0.520000 | 0.134950 | 315.500000 | 2497.000000 | 3152.000000 |
| 50% | 0.498333 | 0.486733 | 0.626667 | 0.180975 | 713.000000 | 3662.000000 | 4548.000000 |
| 75% | 0.655417 | 0.608602 | 0.730209 | 0.233214 | 1096.000000 | 4776.500000 | 5956.000000 |
| max | 0.861667 | 0.840896 | 0.972500 | 0.507463 | 3410.000000 | 6946.000000 | 8714.000000 |

### Analysis:

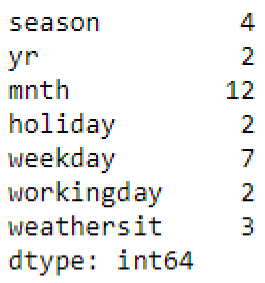
We have four independent continuous variables (temp, atemp, hum and windspeed) and three dependent continuous variables (casual, registered and cnt).

As we can observe from above table, all independent variable are between 0 and 1. Dataset contain normalized values for all four columns.

**'cnt'** the. target variable have minimum **22** counting and maximum **8714** counting.

## Analysis of Categorical Variables:

Number of unique values in each category:



```
season         4
yr             2
mnth          12
holiday        2
weekday        7
workingday     2
weathersit     3
dtype: int64
```

Counting of each unique value in each categorical variable

value counts of categorical column

```
season
3    188
2    184
1    181
4    178
Name: season, dtype: int64
*************************************************************
*******
yr
1    366
0    365
Name: yr, dtype: int64
*************************************************************
*******
mnth
12    62
10    62
8     62
7     62
5     62
3     62
1     62
11    60
9     60
6     60
4     60
2     57
Name: mnth, dtype: int64
```

```
**********************************************************
*******
holiday
0    710
1     21
Name: holiday, dtype: int64
**********************************************************
*******
weekday
6    105
1    105
0    105
5    104
4    104
3    104
2    104
Name: weekday, dtype: int64
**********************************************************
*******
workingday
1    500
0    231
Name: workingday, dtype: int64
**********************************************************
*******
weathersit
1    463
2    247
```

```
3    21
Name: weathersit, dtype: int64
```

**Analysis:**

We have similar counting of unique value for ‘season’, ‘yr’, ‘mnth’ and ‘weekday’. As these data are related to time and we have two year’s data, that is why we have similar counting.

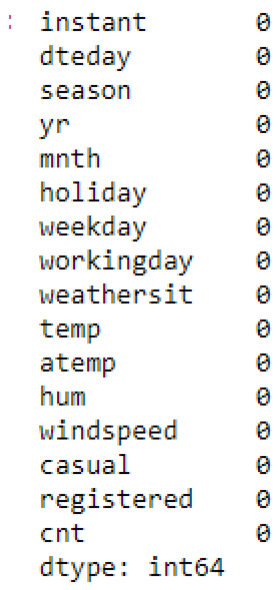
We have 21 holidays and 710 non holiday. This column is highly imbalanced, possibility is that, it would not help our model. We will look for this in feature importance section.

We have working day column, which would be important for us than holiday, we have 500 working day and 231 non-working day. Which would be in approx. ratio of 5:2 (5 working day in a week) and non working day having holiday part also.

**Weathersit** has only three unique values with counting of value 3(Light raining) is 21, counting of value 1(clear weather) is 463 and counting of value 2 (cloudy weather) is 247.

### 2.1.2 Missing value analysis:

Checking missing values for each column in our dataset:



```
: instant       0
  dteday        0
  season        0
  yr            0
  mnth          0
  holiday       0
  weekday       0
  workingday    0
  weathersit    0
  temp          0
  atemp         0
  hum           0
  windspeed     0
  casual        0
  registered    0
  cnt           0
  dtype: int64
```

**Analysis:**

In our dataset we are lucky that we don't have any missing values. In case we have missing values then we should impute it using different method mean, median, KNN

### 2.1.3 Outlier Analysis:

Outliers, are attributed to a rare chance and may not necessarily be fully explainable, Outliers in data can distort predictions and affect the accuracy, if you don't detect and handle them.

The contentious decision to consider or discard an outlier needs to be taken at the time of building the model. Outliers can drastically bias/change the fit estimates and predictions. It is left to the best judgement of the analyst to decide whether treating outliers is necessary and how to go about it.

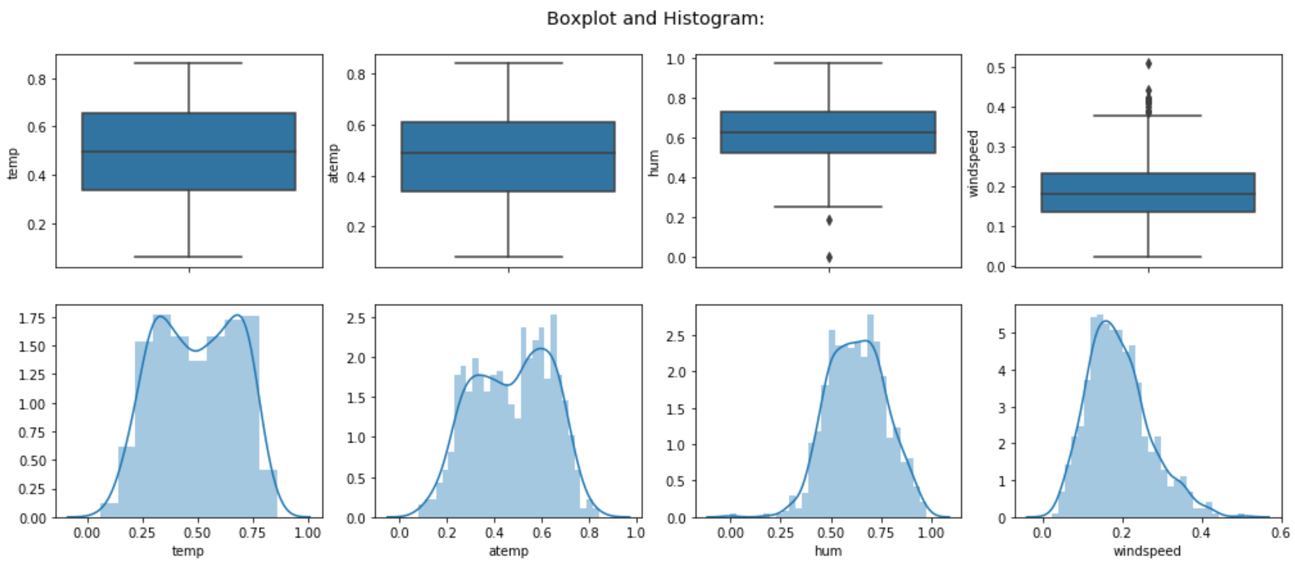
Treating or altering the outlier/extreme values in genuine observations is not a standard operating procedure. If a data point (or points) is excluded from the data analysis, this should be clearly stated on any subsequent rep

Our dataset is based on season and environmental condition. Boxplot finds outliers by calculating distance on a single column only. Suppose for clear weather condition, windspeed is normal maximum time. Now some day windspeed is higher than others day and it has high value. Now, Boxplot may consider it as outlier, as distance from median would be high for this value and due to high windspeed there may be decrease in bike rental counting for that day. So, boxplot method would remove that datapoint, but that data point could be an important predictor.

Let us check for outliers in our numerical dataset and will also look at distribution of data.

So we have numerical columns as temp, atemp, hum and windspeed. Abnormal values may directly affect count of bike renting, so removing outlier would not be a good idea for this dataset as per domain knowledge.

Overview of boxplot and histogram for our numerical data



### Analysis:

We have no outliers for temp and atemp and very less outlier for humidity (hum) and few outliers for windspeed. Distribution is almost normal with little skewness for hum and windspeed and near to normal for temp and atemp with little bimodal effect.

We will make our model with whole data points and with dataset without outliers and then at the end will check which model performance and would decide after that.

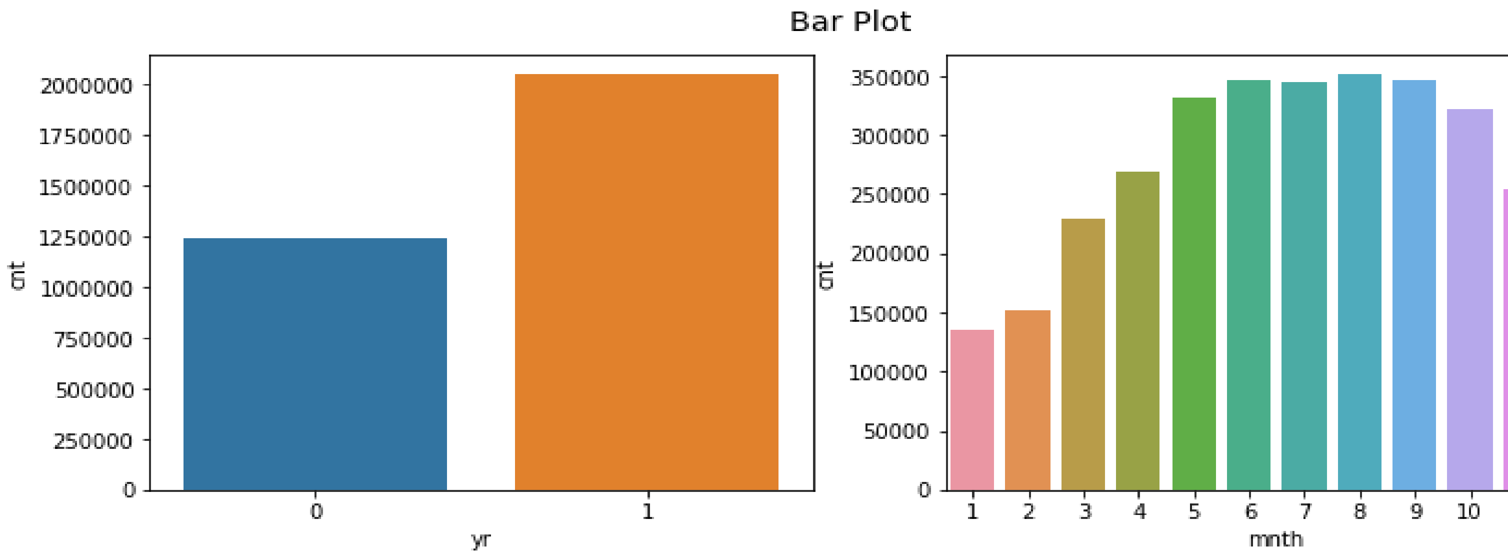
## 2.1.4 Feature Engineering:

We have two columns ‘**mnth’** and ‘**weekday’**. In month column we have range from 1 to 31 and in weekday we have range from 0 to 6

### mnth:

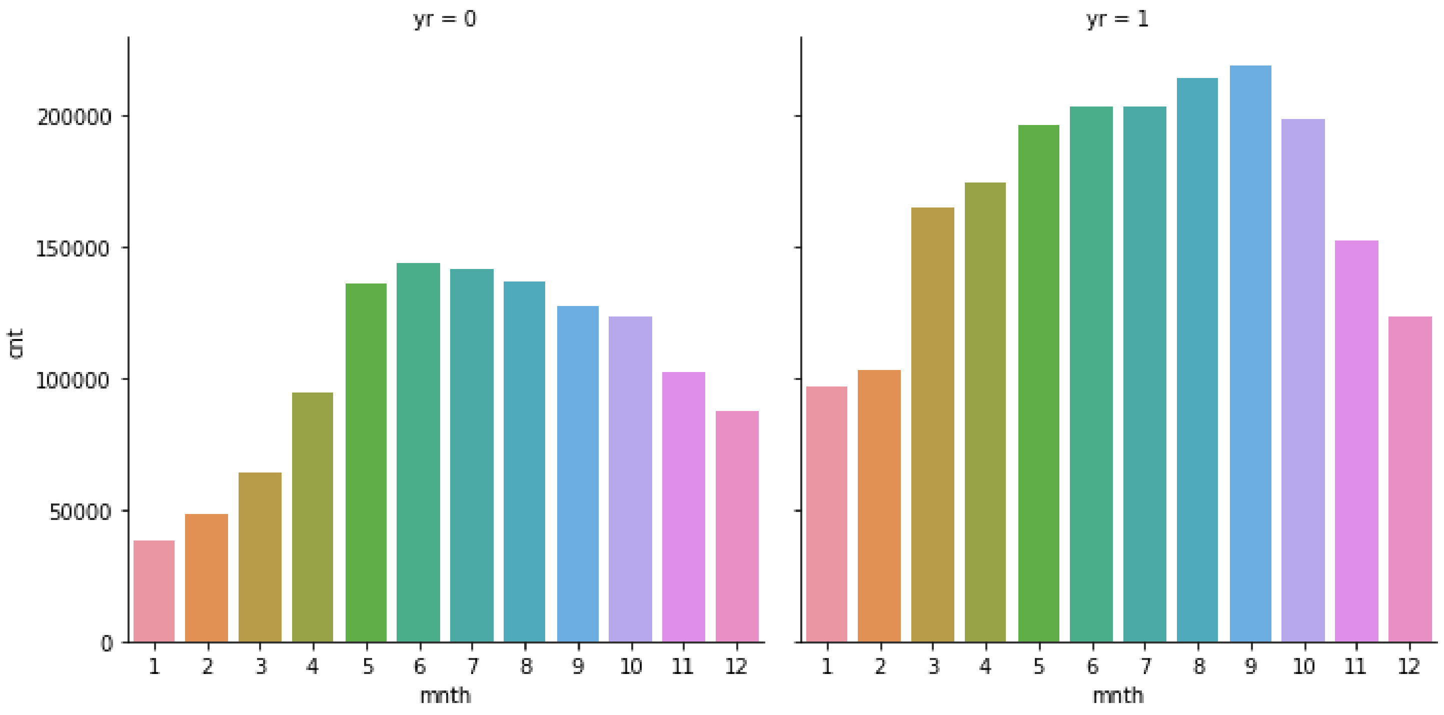
choosingeverymnthasancategory,thenwemay havecurseofdimensionalityand ourmodelwillunderperform.So,we will make a new column for ‘mnth’ as per current values.

Let us see distribution of counting of bike renting according to month as well as with year



As we can see the trend in counting of bike renting for month column. 5th to 10th month has high renting and others are less in comparison.

Above plot figure distribution of month is consolidated for both year 2011 and 2012. Let us see distribution with respect to each year.

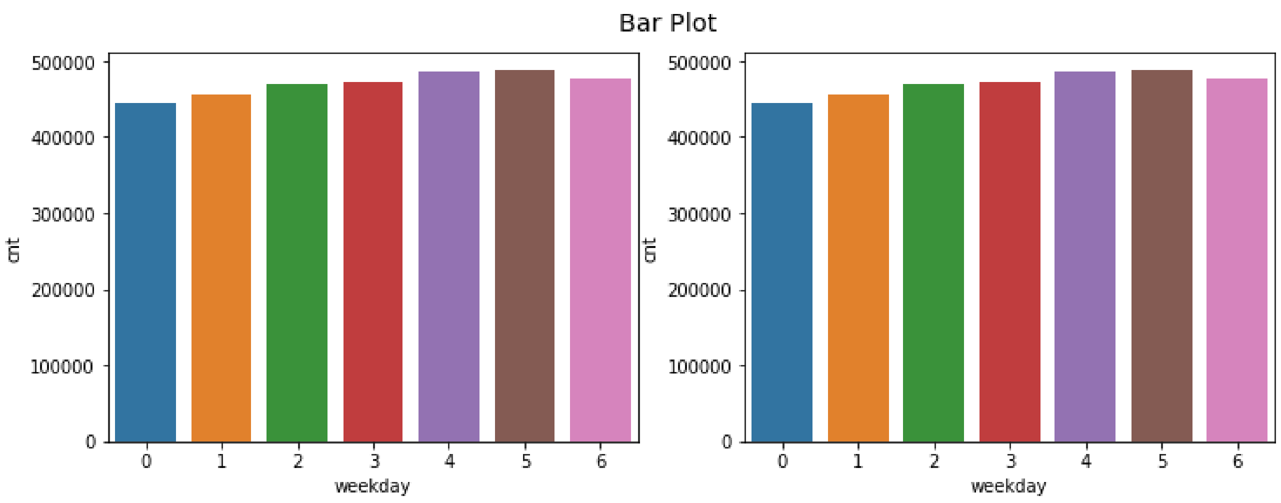


From the plot distribution we can figure out that both the years have similar trends.

As we will categories month in two categories i.e. ‘1’ for month 5th to 10th and ‘0’ for remaining. As 5th to 10th having high values for both 2011 and 2012 and rest have less.

**Weekday:**

Overview of bar plot of counting for each weekday:



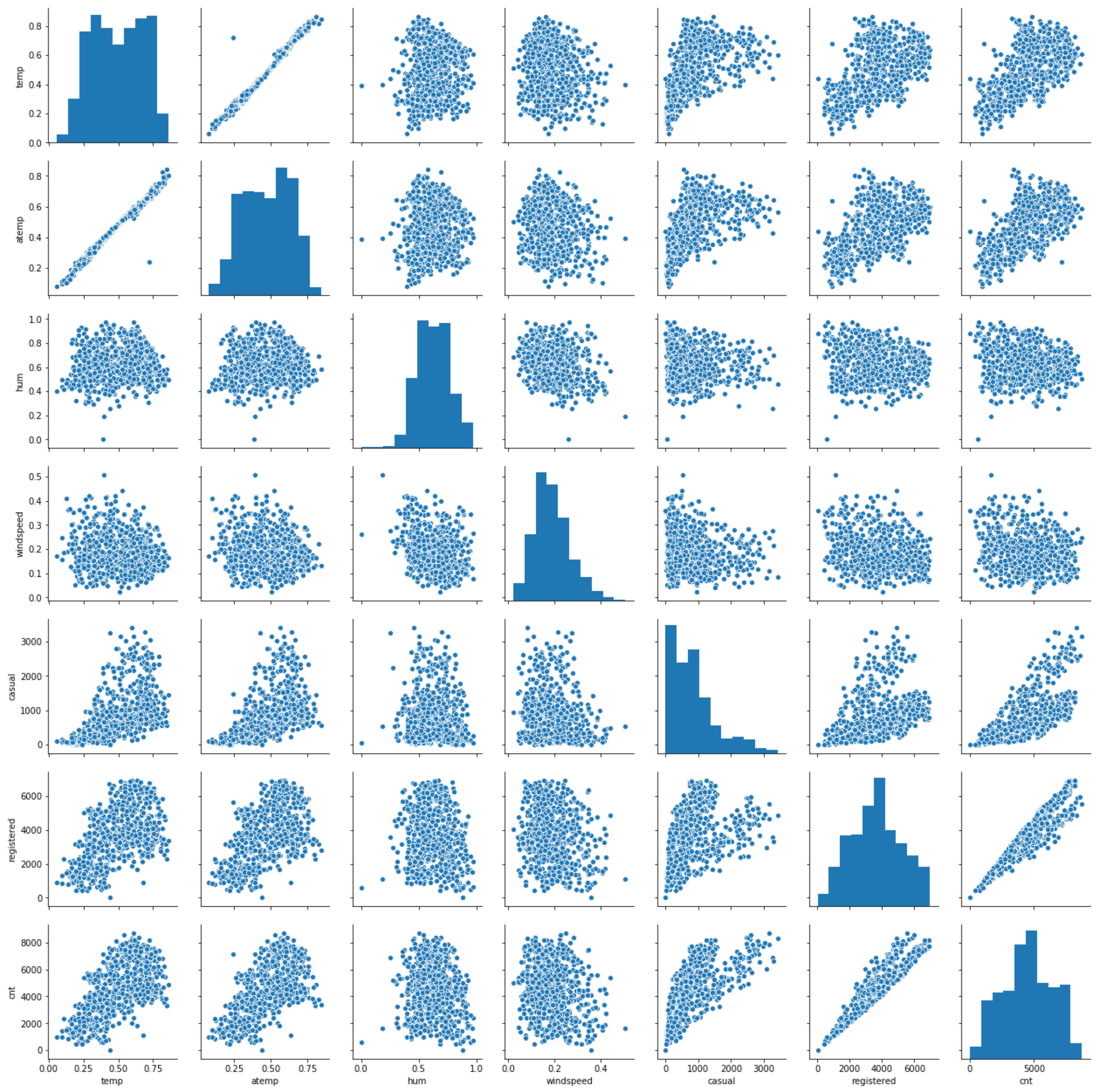
Here, we can see there is also a trend for weekday. We will make two categories for weekday. Value
'0' for weekday 0 and 1 and value '1' for remaining.

We will create dummy variables(with dropping onecolumn) for our category, season and weathersit in python and for R we just have to change them in factor datatype.

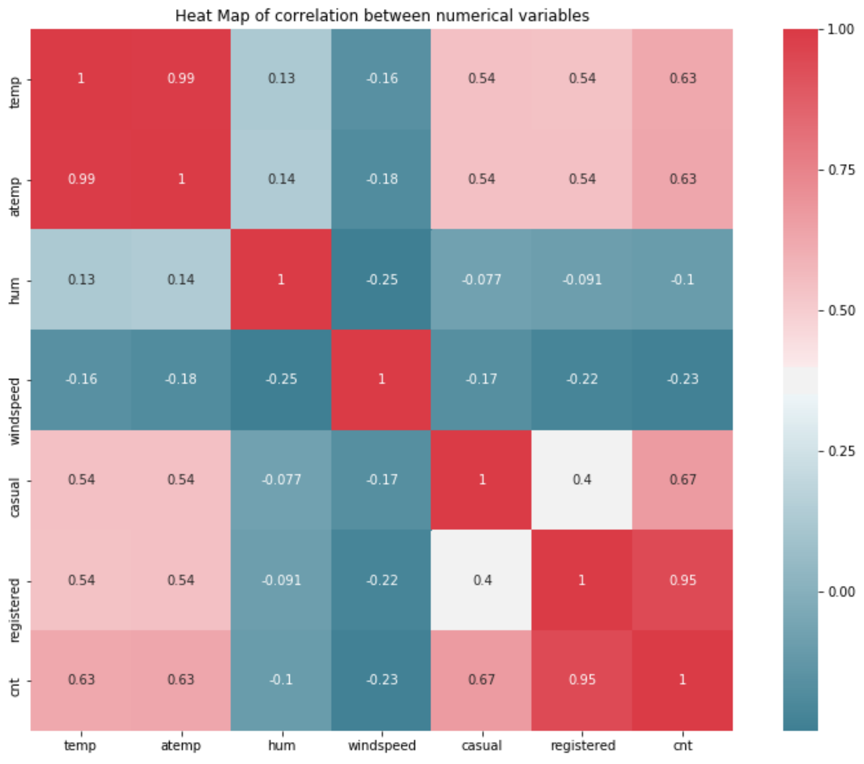
## 2.1.5 Feature selection:

In Feature selection you have to check whether there is multicolinearity between our continuous independent variables.

Overview of scatter plot between of our continuous variables



Overview of heatmap:

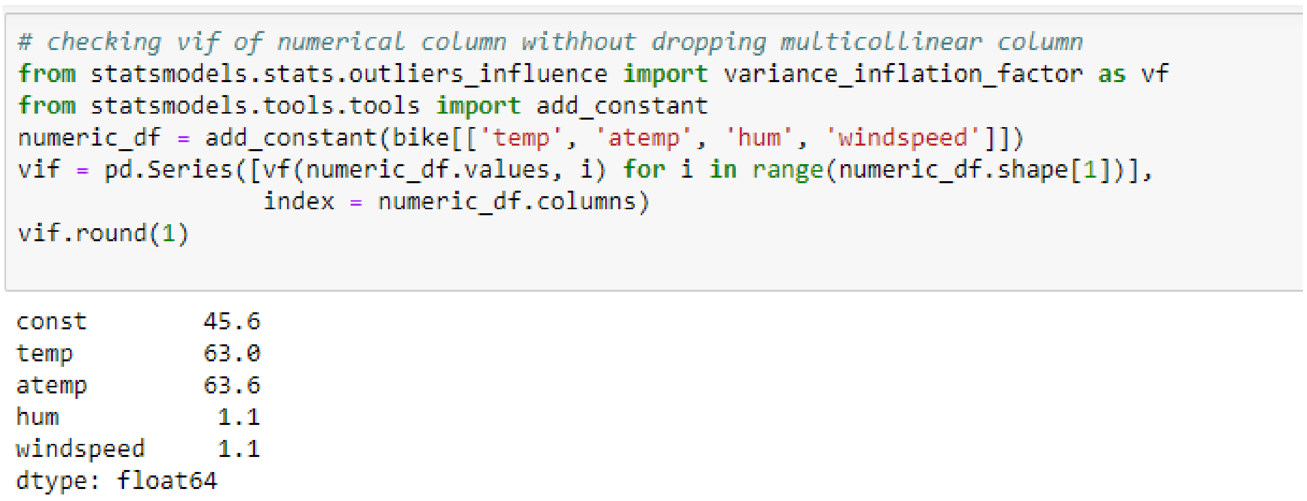


## Analysis:

From heatmap and correlation plot we can figure it out that, there is

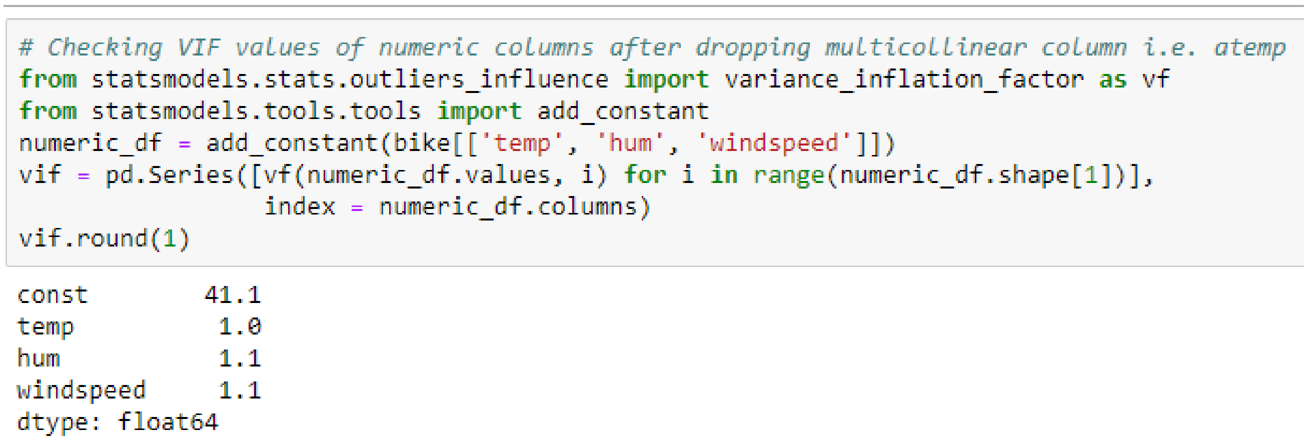
Multicollinearity present between temp and atemp, so we will have to drop one of those columns as per importance of features. Multicollinearity present between registered and cnt, but these are our target variable. We will only use total counting i.e. cnt as our target variable and would drop casual and registered column.

Checking the VIF before and after removing the atemp variable because it has multicollinear with temp and having less importance than temp variable



```
# checking vif of numerical column withhout dropping multicollinear column
from statsmodels.stats.outliers_influence import variance_inflation_factor as vf
from statsmodels.tools.tools import add_constant
numeric_df = add_constant(bike[['temp', 'atemp', 'hum', 'windspeed']])
vif = pd.Series([vf(numeric_df.values, i) for i in range(numeric_df.shape[1])],
                 index = numeric_df.columns)
vif.round(1)
```

```
const        45.6
temp         63.0
atemp        63.6
hum           1.1
windspeed     1.1
dtype: float64
```



```
# Checking VIF values of numeric columns after dropping multicollinear column i.e. atemp
from statsmodels.stats.outliers_influence import variance_inflation_factor as vf
from statsmodels.tools.tools import add_constant
numeric_df = add_constant(bike[['temp', 'hum', 'windspeed']])
vif = pd.Series([vf(numeric_df.values, i) for i in range(numeric_df.shape[1])],
                 index = numeric_df.columns)
vif.round(1)
```

```
const        41.1
temp          1.0
hum           1.1
windspeed     1.1
dtype: float64
```

After removal of atemp we have good value of VIF and now we don’t have multicollinearity

## 2.1.6 Feature Scaling:

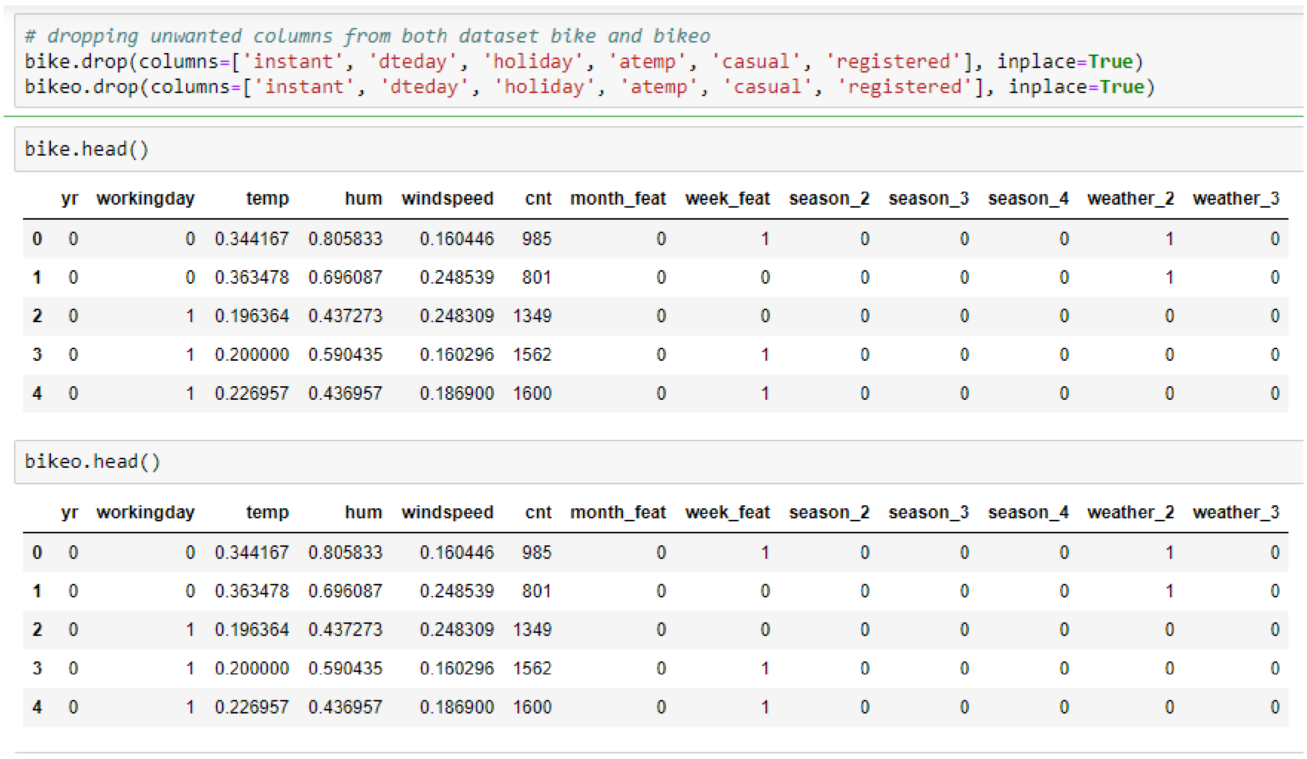
The numerical variables are in normalized form .

All variable are in same range. So we don’t require feature scaling for our dataset.

### 2.1.7 Data After EDA:

We removed ‘instant’, ‘holiday’, ‘dteday’, ‘atemp’, ‘casual’, ‘registered’ and created bins for ‘mnth’ and ‘weekday’. We will make dummy variables for season and weather in python and would change them to factor in R.

We have create another dataset(bikeo) but after removal of outliers from hum and windspeed.



```python
# dropping unwanted columns from both dataset bike and bikeo
bike.drop(columns=['instant', 'dteday', 'holiday', 'atemp', 'casual', 'registered'], inplace=True)
bikeo.drop(columns=['instant', 'dteday', 'holiday', 'atemp', 'casual', 'registered'], inplace=True)
```

```python
bike.head()
```

| | yr | workingday | temp | hum | windspeed | cnt | month_feat | week_feat | season_2 | season_3 | season_4 | weather_2 | weather_3 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 0 | 0 | 0.344167 | 0.805833 | 0.160446 | 985 | 0 | 1 | 0 | 0 | 0 | 1 | 0 |
| **1** | 0 | 0 | 0.363478 | 0.696087 | 0.248539 | 801 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| **2** | 0 | 1 | 0.196364 | 0.437273 | 0.248309 | 1349 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **3** | 0 | 1 | 0.200000 | 0.590435 | 0.160296 | 1562 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| **4** | 0 | 1 | 0.226957 | 0.436957 | 0.186900 | 1600 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |

```python
bikeo.head()
```

| | yr | workingday | temp | hum | windspeed | cnt | month_feat | week_feat | season_2 | season_3 | season_4 | weather_2 | weather_3 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 0 | 0 | 0.344167 | 0.805833 | 0.160446 | 985 | 0 | 1 | 0 | 0 | 0 | 1 | 0 |
| **1** | 0 | 0 | 0.363478 | 0.696087 | 0.248539 | 801 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| **2** | 0 | 1 | 0.196364 | 0.437273 | 0.248309 | 1349 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **3** | 0 | 1 | 0.200000 | 0.590435 | 0.160296 | 1562 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| **4** | 0 | 1 | 0.226957 | 0.436957 | 0.186900 | 1600 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |

The above diagram shows the bike dataset with and without outliers

## 2.2 Modeling

We will now build our model, before proceeding terms used in our codes:

- bike: dataset containing all columns except which we removed in EDA
- bikeo : its same as bike but we removed outliers from it.
- X_train: containing all independent variables (part of bike), used for training model
- y_train : containing target variable (part of bike), used for training model
- X_test : containing independent variable (part of bike), used for testing model
- y_test: containing target variable(part of bike), , used for testing model
- X_traino: containing all independent variables (part of bikeo), used for training model
- y_traino : containing target variable (part of bikeo), used for training model
- X_testo : containing independent variable (part of bikeo), used for testing model
- y_testo: containing target variable (part of bikeo), , used for testing model
- fit_predict_show_performance: user-defined function, which will fit our model on training set, and will calculate and print k-fold (10-fold) cross validation score for explained_variance , and then will make prediction on training and test dataset and will print explained_variance for both training and test dataset.

### 2.2.1 K-fold CV, GridsearchCV and Explained_variance

Before building models on our dataset, we would like to explore three things:

- K-fold cross validation
- GridSearchCV
- Explained_variance (metric fromsklearn)

## K-fold Cross Validation:

K-fold cross validation is used to check performance of model which is checked on K different test dataset. Let us assume, we have built a model and we are checking performance of our model on a test data and our model show accuracy of 95% and now we will check our model on different test data and now accuracy is 80%. So what should we consider for deciding model performance? So in this, K-fold cross validation helps, it would divide our training data in k sets and will build a model using k-1 training set and one left set would be used to test our model performance. In this way it would build k times model and each time there would be different test dataset to check performance and at the end all k model's accuracy(or any other metric) mean value would be considered as model accuracy(any metric).

So, we would use K-fold cross validation technique to get performance of our model.

## GridsearchCV : (Hyperparameter tuning):

Hyperparameter are the parameters which we pass as argument to our building function, like kernel, criterion, n_estimators etc. So to get best values of these gridserchcv is used. In this technique, we make list of these different parameters and then gridsearchcv build model for every combination of these parameters and then check crossvalidation score and based on score it gives the best combination of hyperparameters.

And then we can build our model with the values of hyperparameter given by GridSearchCV.

This is called performance tuning and we would use this to tune our model.

## Explained_variance_score (metric from sklearn):

We have different metrics score for regression to check performance of model. All metrics having difference of y_pred and y_true in their calculation. y_pred is the value predicted by model and y_true is actual value. These metrics tell us deviation of prediction from actual value. We have different metrics for regression:

- Explained_variance
- Mean_absolute_error
- Mean_squared_error
- Mean_squared_log_error
- Median_absolute_error
- R2

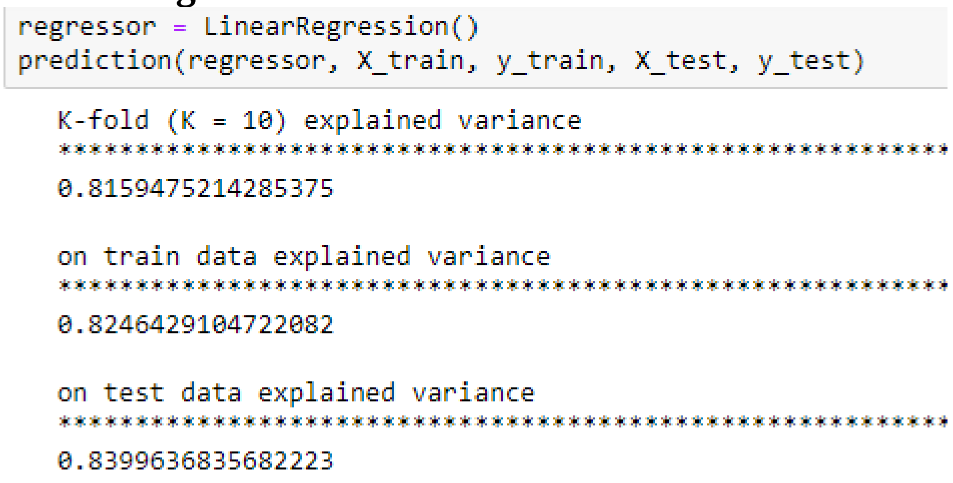
We can use any of them for comparing our model, but we will use explained_variance. Best possible score is 1.0 (perfect prediction) and less is worse.

### 2.2.2 Building models

**Models and performance of models**:

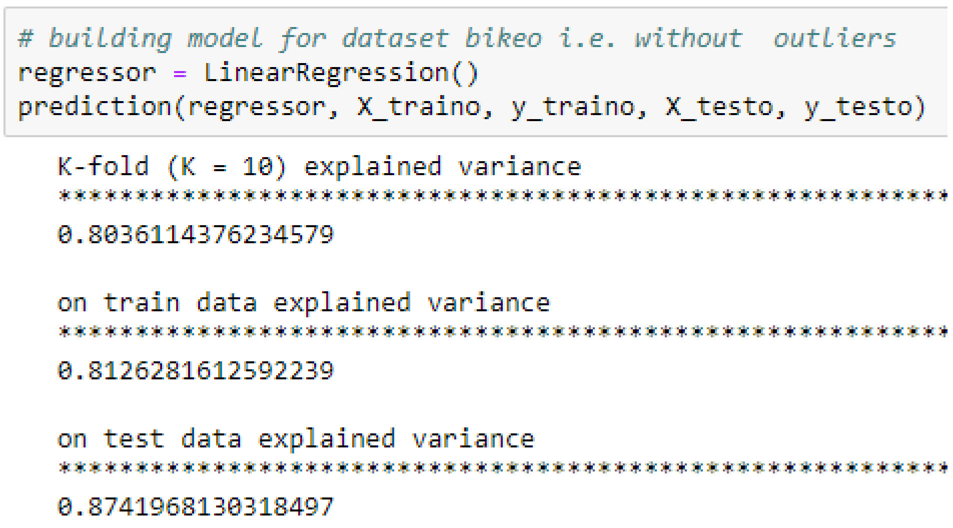
We will now build one by one all models and will check performance of our model and then at the will decide final model which we should use for our project.

**Linear Regression:**



```
regressor = LinearRegression()
prediction(regressor, X_train, y_train, X_test, y_test)
```

```
K-fold (K = 10) explained variance
**********************************************************
0.8159475214285375

on train data explained variance
**********************************************************
0.8246429104722082

on test data explained variance
**********************************************************
0.8399636835682223
```



```
# building model for dataset bikeo i.e. without  outliers
regressor = LinearRegression()
prediction(regressor, X_traino, y_traino, X_testo, y_testo)
```

```
K-fold (K = 10) explained variance
**********************************************************
0.8036114376234579

on train data explained variance
**********************************************************
0.8126281612592239

on test data explained variance
**********************************************************
0.8741968130318497
```

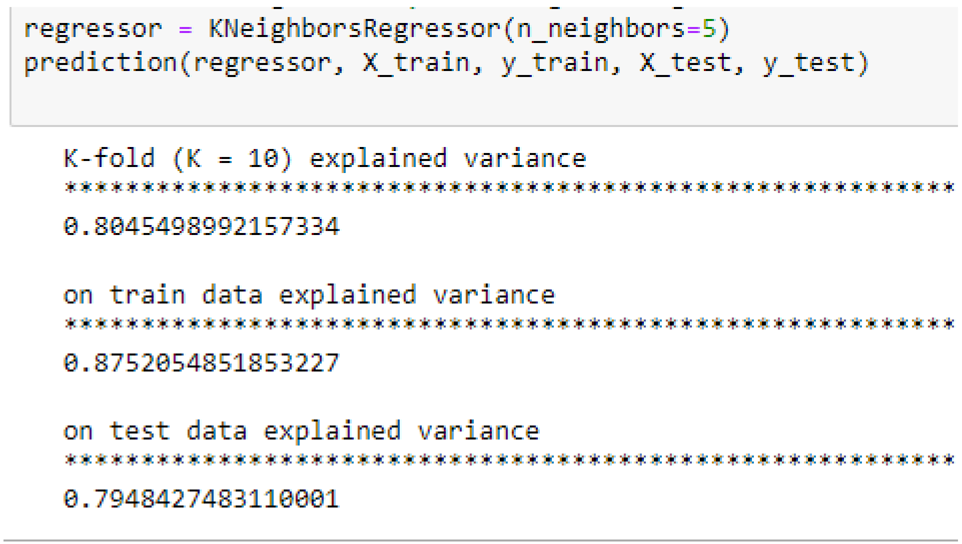
**Analysis:**

As we can see from above results K-fold explained variance for whole dataset is 81.59% and for dataset without outlier is 80.36%.

So, from here we can observe that we don't have any outlier in our dataset, outlier declared by

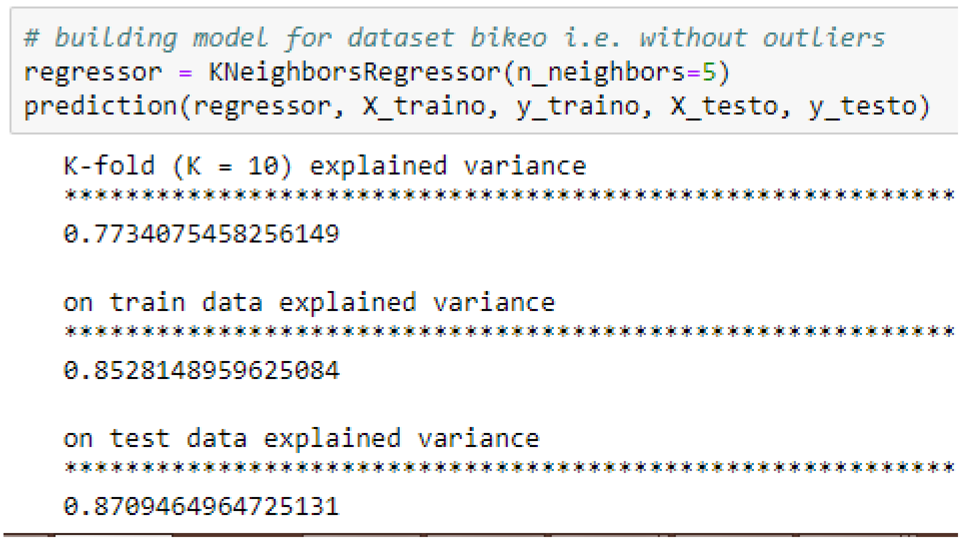
boxplot method is information

## K Nearest Neighbors Regressor:



```
regressor = KNeighborsRegressor(n_neighbors=5)
prediction(regressor, X_train, y_train, X_test, y_test)
```

```
K-fold (K = 10) explained variance
**********************************************************
0.8045498992157334

on train data explained variance
**********************************************************
0.8752054851853227

on test data explained variance
**********************************************************
0.7948427483110001
```



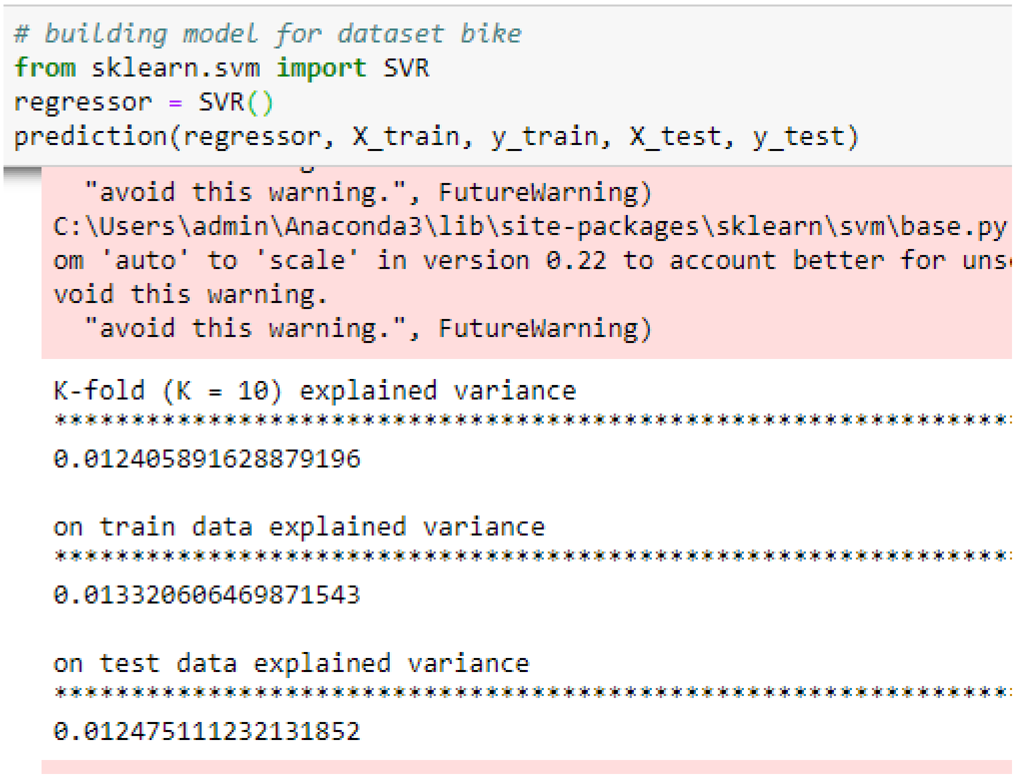
```
# building model for dataset bikeo i.e. without outliers
regressor = KNeighborsRegressor(n_neighbors=5)
prediction(regressor, X_traino, y_traino, X_testo, y_testo)
```

```
K-fold (K = 10) explained variance
**********************************************************
0.7734075458256149

on train data explained variance
**********************************************************
0.8528148959625084

on test data explained variance
**********************************************************
0.8709464964725131
```

## Analysis:

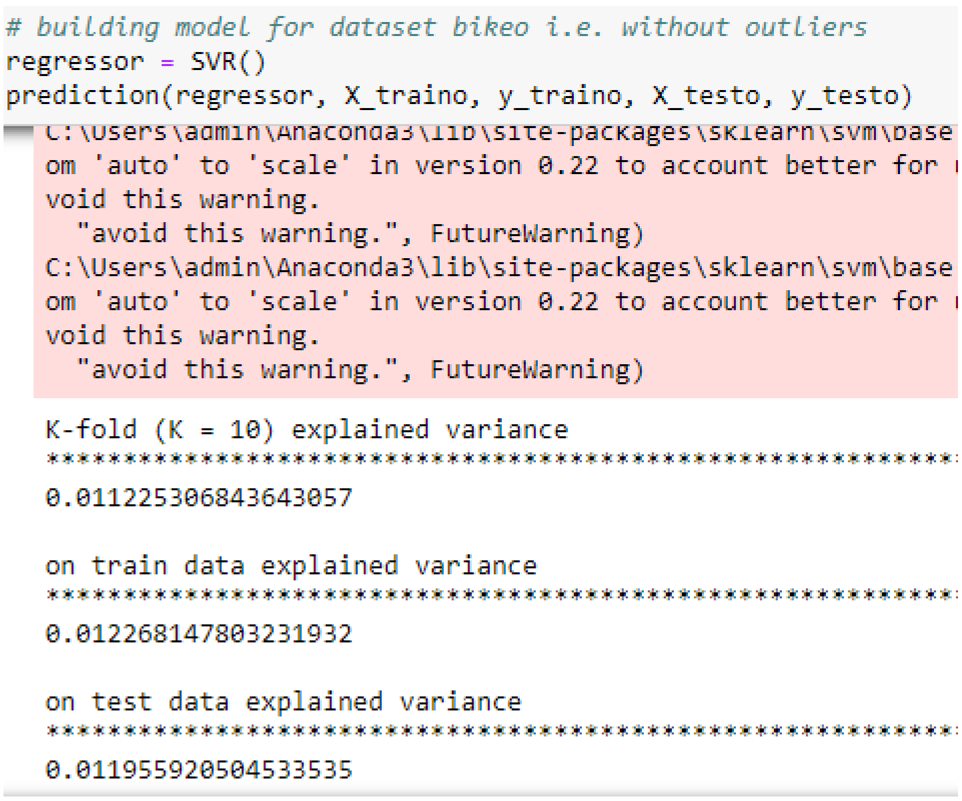
We can observe from above result, KNN regressor showing K-fold explained_variance score 80.45% for whole dataset and 77.34% for dataset without outlier

## Support Vector Regressor:



```
# building model for dataset bike
from sklearn.svm import SVR
regressor = SVR()
prediction(regressor, X_train, y_train, X_test, y_test)
```

```
  "avoid this warning.", FutureWarning)
C:\Users\admin\Anaconda3\lib\site-packages\sklearn\svm\base.py
om 'auto' to 'scale' in version 0.22 to account better for uns
void this warning.
  "avoid this warning.", FutureWarning)

K-fold (K = 10) explained variance
***************************************************************
0.012405891628879196

on train data explained variance
***************************************************************
0.013320606469871543

on test data explained variance
***************************************************************
0.012475111232131852
```



```
# building model for dataset bikeo i.e. without outliers
regressor = SVR()
prediction(regressor, X_traino, y_traino, X_testo, y_testo)
```

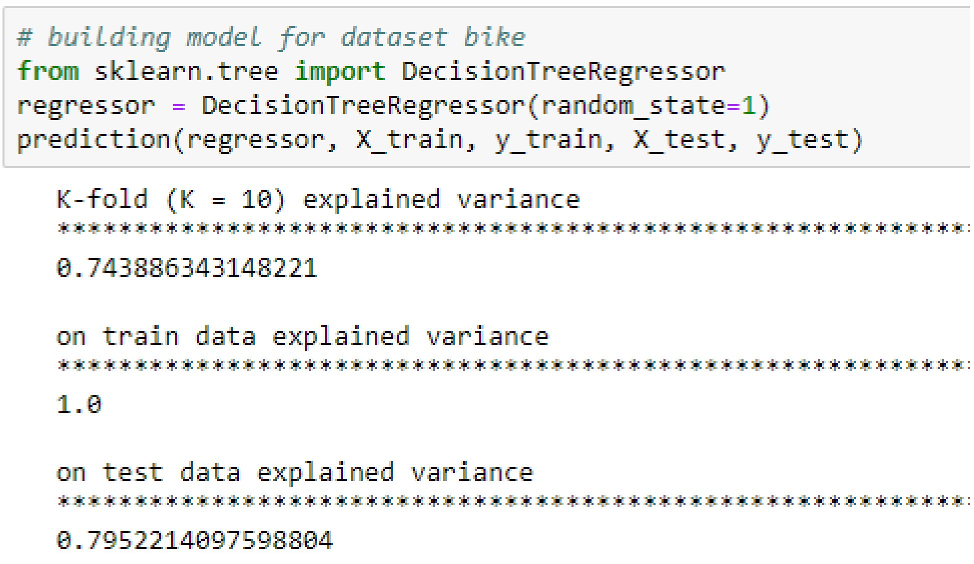
```
C:\Users\admin\Anaconda3\lib\site-packages\sklearn\svm\base
om 'auto' to 'scale' in version 0.22 to account better for
void this warning.
  "avoid this warning.", FutureWarning)
C:\Users\admin\Anaconda3\lib\site-packages\sklearn\svm\base
om 'auto' to 'scale' in version 0.22 to account better for
void this warning.
  "avoid this warning.", FutureWarning)

K-fold (K = 10) explained variance
************************************************************
0.011225306843643057

on train data explained variance
************************************************************
0.012268147803231932

on test data explained variance
************************************************************
0.011955920504533535
```

## Analysis:

Support vector regressor with Gaussian kernel is not giving good accuracy result for

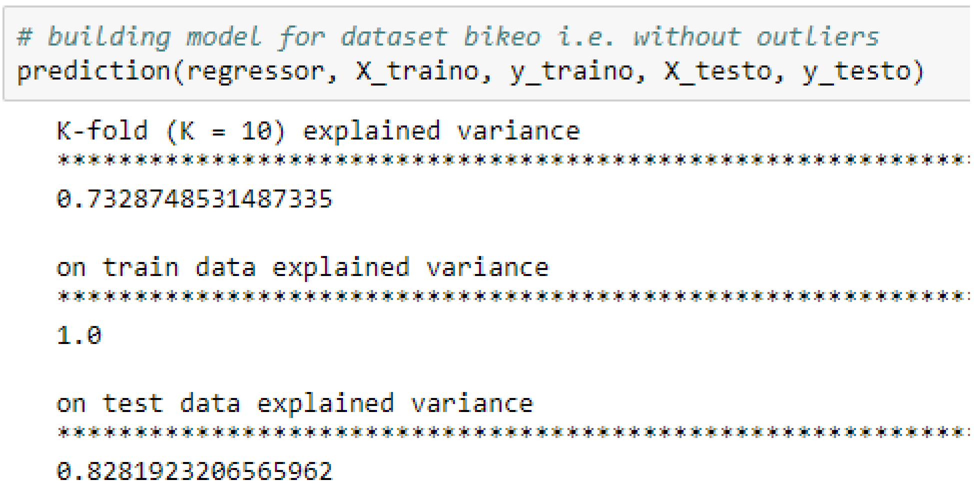
our dataset in any of case.

## Decision Tree Regressor:



```
# building model for dataset bike
from sklearn.tree import DecisionTreeRegressor
regressor = DecisionTreeRegressor(random_state=1)
prediction(regressor, X_train, y_train, X_test, y_test)
```

```
K-fold (K = 10) explained variance
************************************************************
0.743886343148221

on train data explained variance
************************************************************
1.0

on test data explained variance
************************************************************
0.7952214097598804
```



```
# building model for dataset bikeo i.e. without outliers
prediction(regressor, X_traino, y_traino, X_testo, y_testo)
```

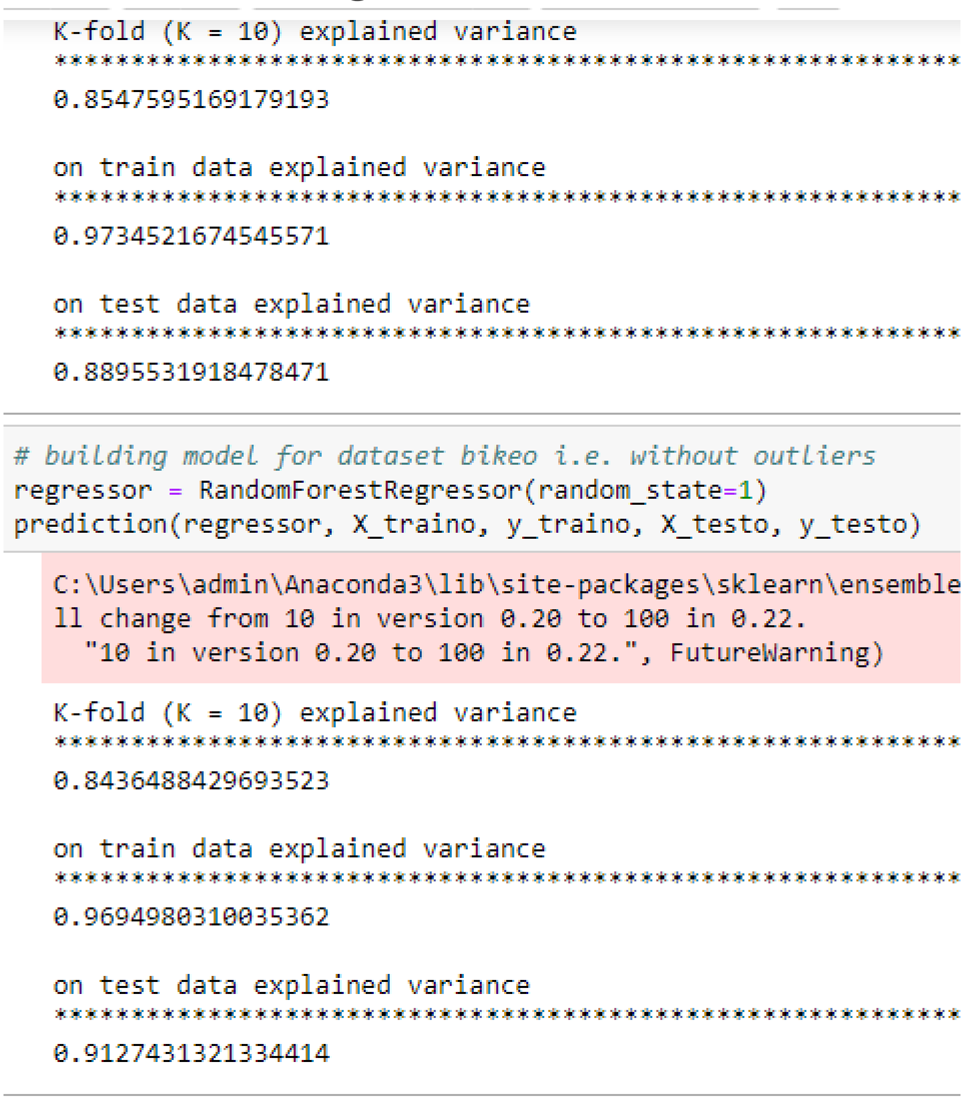
```
K-fold (K = 10) explained variance
************************************************************
0.7328748531487335

on train data explained variance
************************************************************
1.0

on test data explained variance
************************************************************
0.8281923206565962
```

### Analysis:

We can observe from above results Decision tree regressor is explaining K-fold variance 74.38% for whole dataset and 73.28% for dataset without outlier.

Another thing we can observe is Decision tree explaining variance 100% for training dataset. So we have overfitting here.

## Random Forest Regressor:



```
K-fold (K = 10) explained variance
************************************************************
0.8547595169179193

on train data explained variance
************************************************************
0.9734521674545571

on test data explained variance
************************************************************
0.8895531918478471
```

```
# building model for dataset bikeo i.e. without outliers
regressor = RandomForestRegressor(random_state=1)
prediction(regressor, X_traino, y_traino, X_testo, y_testo)
```

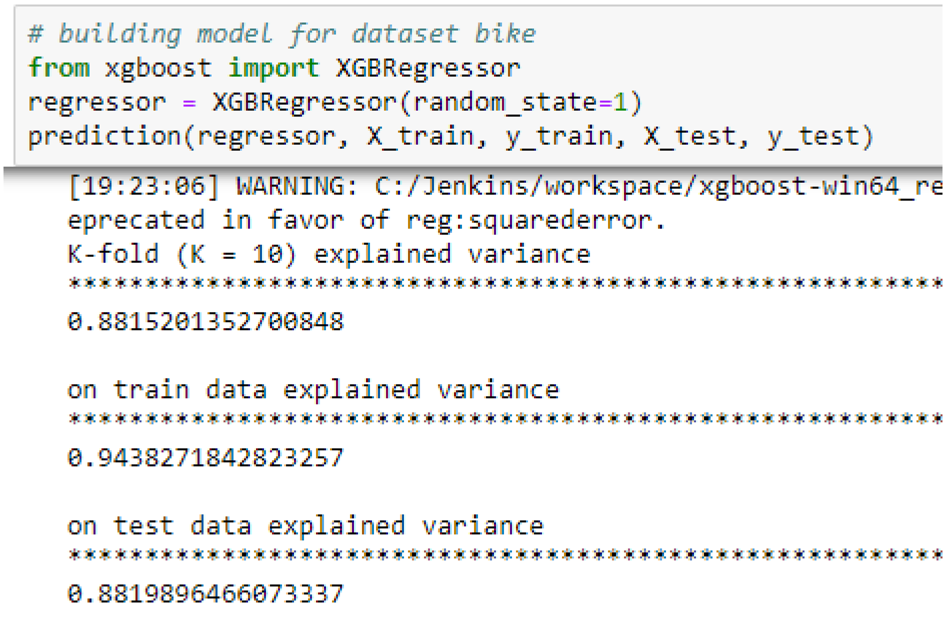
```
C:\Users\admin\Anaconda3\lib\site-packages\sklearn\ensemble
ll change from 10 in version 0.20 to 100 in 0.22.
  "10 in version 0.20 to 100 in 0.22.", FutureWarning)
```

```
K-fold (K = 10) explained variance
************************************************************
0.8436488429693523

on train data explained variance
************************************************************
0.9694980310035362

on test data explained variance
************************************************************
0.9127431321334414
```

### Analysis:

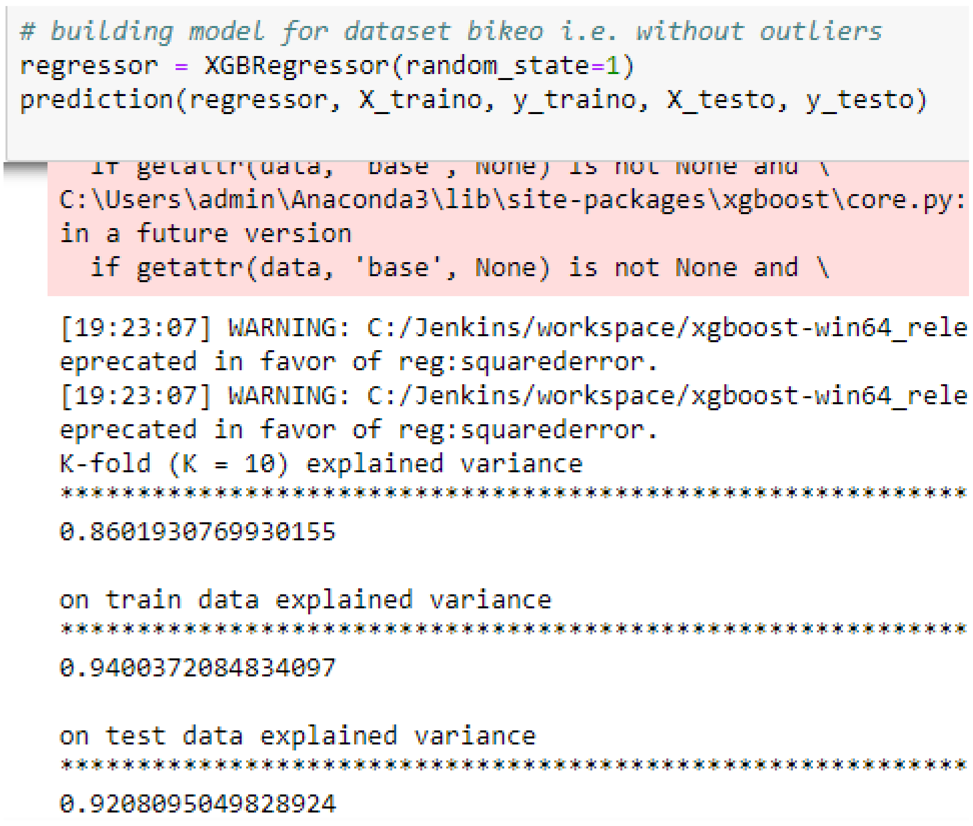
We can observe that Random Forest explained_variance score for k-fold is 85.47% for whole dataset and 84.36% for dataset without outlier. We also have less explained_variance score for training dataset than decision tree and high score for test dataset. So Random forest has helped in removing overfitting.

## XGBRegressor:



```python
# building model for dataset bike
from xgboost import XGBRegressor
regressor = XGBRegressor(random_state=1)
prediction(regressor, X_train, y_train, X_test, y_test)
```

```
[19:23:06] WARNING: C:/Jenkins/workspace/xgboost-win64_re
eprecated in favor of reg:squarederror.
K-fold (K = 10) explained variance
**********************************************************
0.8815201352700848

on train data explained variance
**********************************************************
0.9438271842823257

on test data explained variance
**********************************************************
0.8819896466073337
```



```python
# building model for dataset bikeo i.e. without outliers
regressor = XGBRegressor(random_state=1)
prediction(regressor, X_traino, y_traino, X_testo, y_testo)
```

```
  if getattr(data, 'base', None) is not None and \
C:\Users\admin\Anaconda3\lib\site-packages\xgboost\core.py:
in a future version
  if getattr(data, 'base', None) is not None and \

[19:23:07] WARNING: C:/Jenkins/workspace/xgboost-win64_rele
eprecated in favor of reg:squarederror.
[19:23:07] WARNING: C:/Jenkins/workspace/xgboost-win64_rele
eprecated in favor of reg:squarederror.
K-fold (K = 10) explained variance
************************************************************
0.8601930769930155

on train data explained variance
************************************************************
0.9400372084834097

on test data explained variance
************************************************************
0.9208095049828924
```

**Analysis:**

From the model result we can observe that XGBRegressor explained_variance for K-fold is 88.15% for whole dataset and 86% for dataset without outliers. Also we had best accuracy among the others.

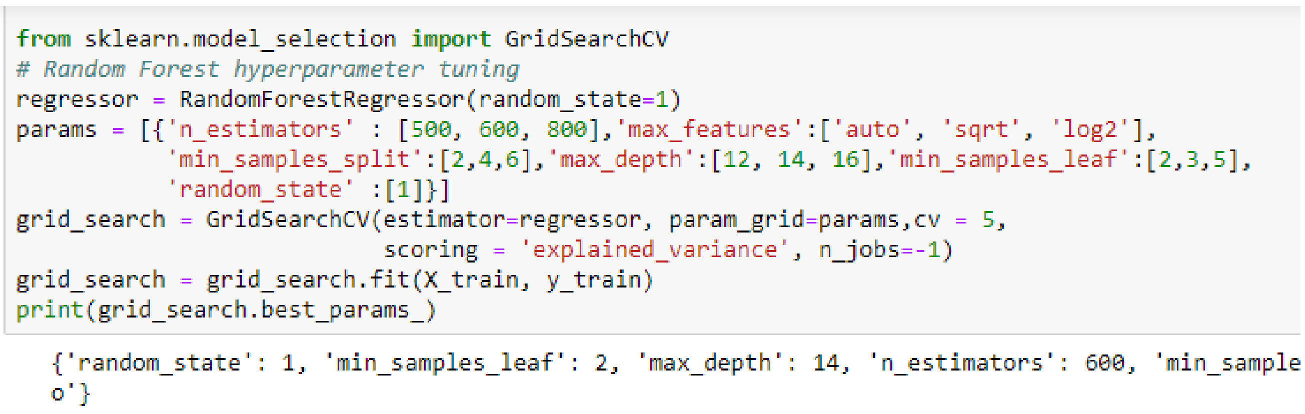
From all model's performances we decided to take whole dataset i.e. bike as final dataset as we have outlier has information which are now necessary for our model performance.

## 2.2.3 Hyperparameter tuning:

We have to tune our two best models ( Random Forest and XGBRegressor). As we see in previous step, outliers shown by boxplot method in actual was the information for our model. So we will tune our model for whole dataset (bike). With the help of hyperparameter tuning we would find optimum values for parameter used in function and would increase our accuracy.

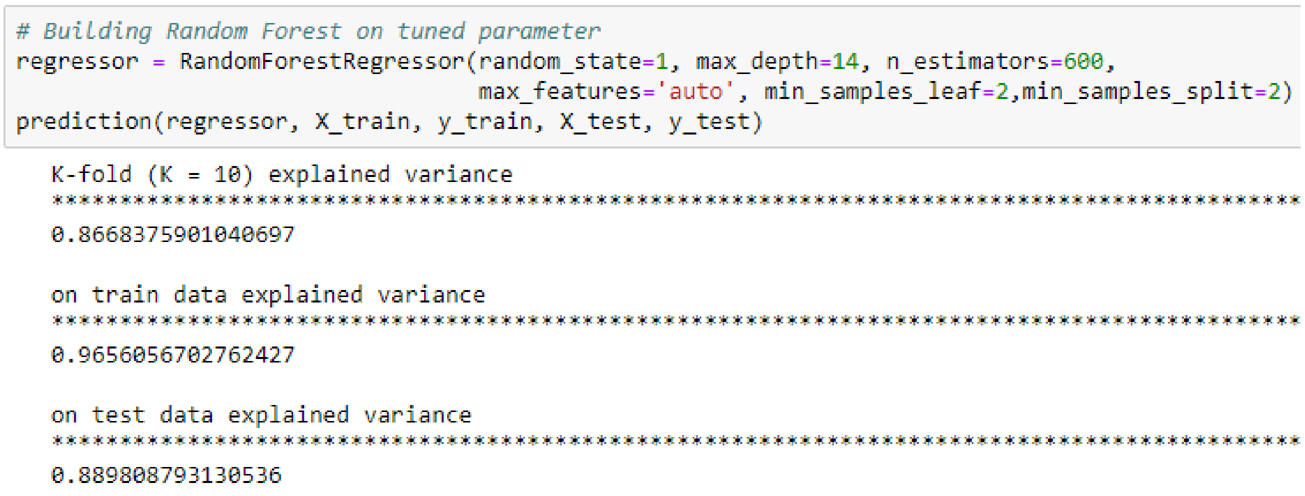
## Random Forest Hyperparameter tuning:

Tuning Random Forest model parameters



```
from sklearn.model_selection import GridSearchCV
# Random Forest hyperparameter tuning
regressor = RandomForestRegressor(random_state=1)
params = [{'n_estimators' : [500, 600, 800],'max_features':['auto', 'sqrt', 'log2'],
           'min_samples_split':[2,4,6],'max_depth':[12, 14, 16],'min_samples_leaf':[2,3,5],
           'random_state' :[1]}]
grid_search = GridSearchCV(estimator=regressor, param_grid=params,cv = 5,
                           scoring = 'explained_variance', n_jobs=-1)
grid_search = grid_search.fit(X_train, y_train)
print(grid_search.best_params_)
```

```
{'random_state': 1, 'min_samples_leaf': 2, 'max_depth': 14, 'n_estimators': 600, 'min_sample
o'}
```



```
# Building Random Forest on tuned parameter
regressor = RandomForestRegressor(random_state=1, max_depth=14, n_estimators=600,
                                  max_features='auto', min_samples_leaf=2,min_samples_split=2)
prediction(regressor, X_train, y_train, X_test, y_test)
```

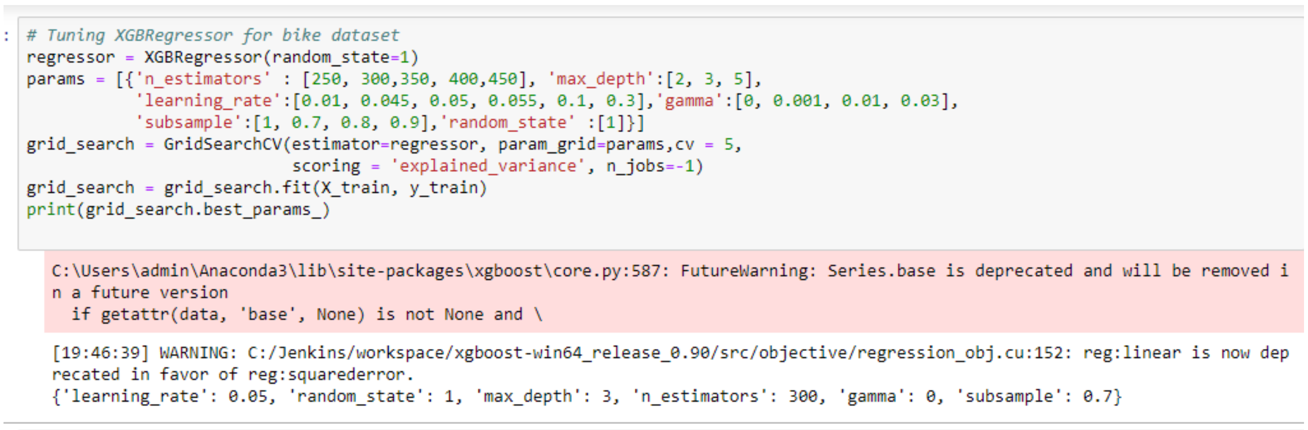
```
K-fold (K = 10) explained variance
********************************************************************************************
0.8668375901040697

on train data explained variance
********************************************************************************************
0.9656056702762427

on test data explained variance
********************************************************************************************
0.889808793130536
```

## Analysis:

From above result (on tuned parameter), we have increased our model explained_variance score for k-fold from 85.47% to 86.68%. Also we can observe that previously it was giving accuracy for training dataset as 97% and now it is giving 96% and on test dataset we have slightly increased accuracy.

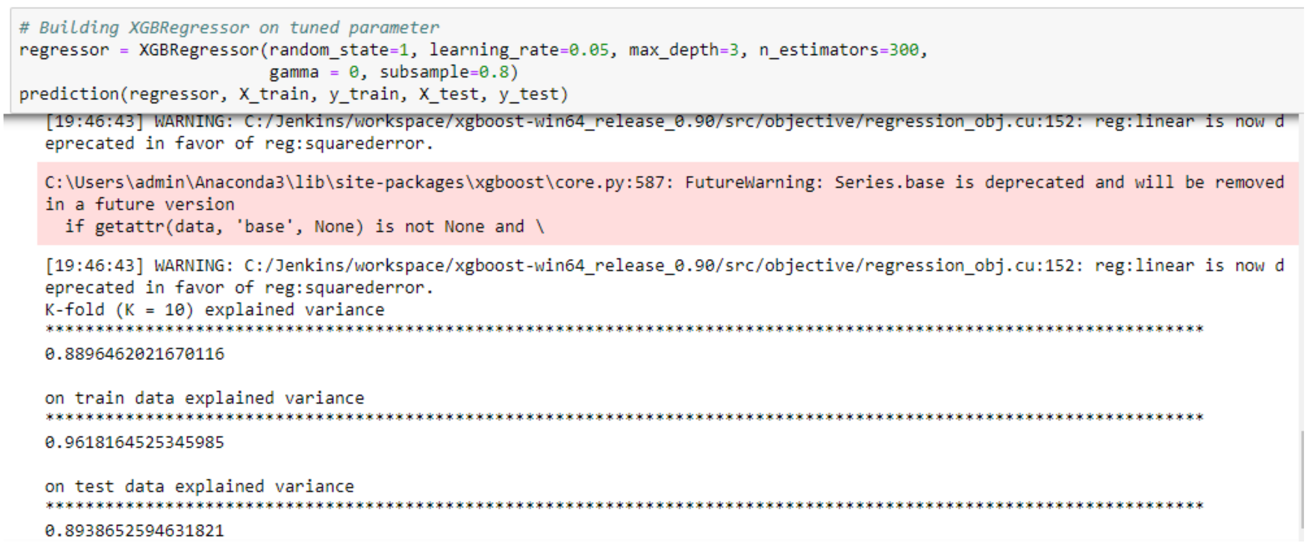
## XGBRegressor Hyperparameter tuning:

Tuning XGBRegressor model  parameters



```
# Tuning XGBRegressor for bike dataset
regressor = XGBRegressor(random_state=1)
params = [{'n_estimators' : [250, 300,350, 400,450], 'max_depth':[2, 3, 5],
           'learning_rate':[0.01, 0.045, 0.05, 0.055, 0.1, 0.3],'gamma':[0, 0.001, 0.01, 0.03],
           'subsample':[1, 0.7, 0.8, 0.9],'random_state' :[1]}]
grid_search = GridSearchCV(estimator=regressor, param_grid=params,cv = 5,
                           scoring = 'explained_variance', n_jobs=-1)
grid_search = grid_search.fit(X_train, y_train)
print(grid_search.best_params_)
```

```
C:\Users\admin\Anaconda3\lib\site-packages\xgboost\core.py:587: FutureWarning: Series.base is deprecated and will be removed i
n a future version
  if getattr(data, 'base', None) is not None and \

[19:46:39] WARNING: C:/Jenkins/workspace/xgboost-win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now dep
recated in favor of reg:squarederror.
{'learning_rate': 0.05, 'random_state': 1, 'max_depth': 3, 'n_estimators': 300, 'gamma': 0, 'subsample': 0.7}
```



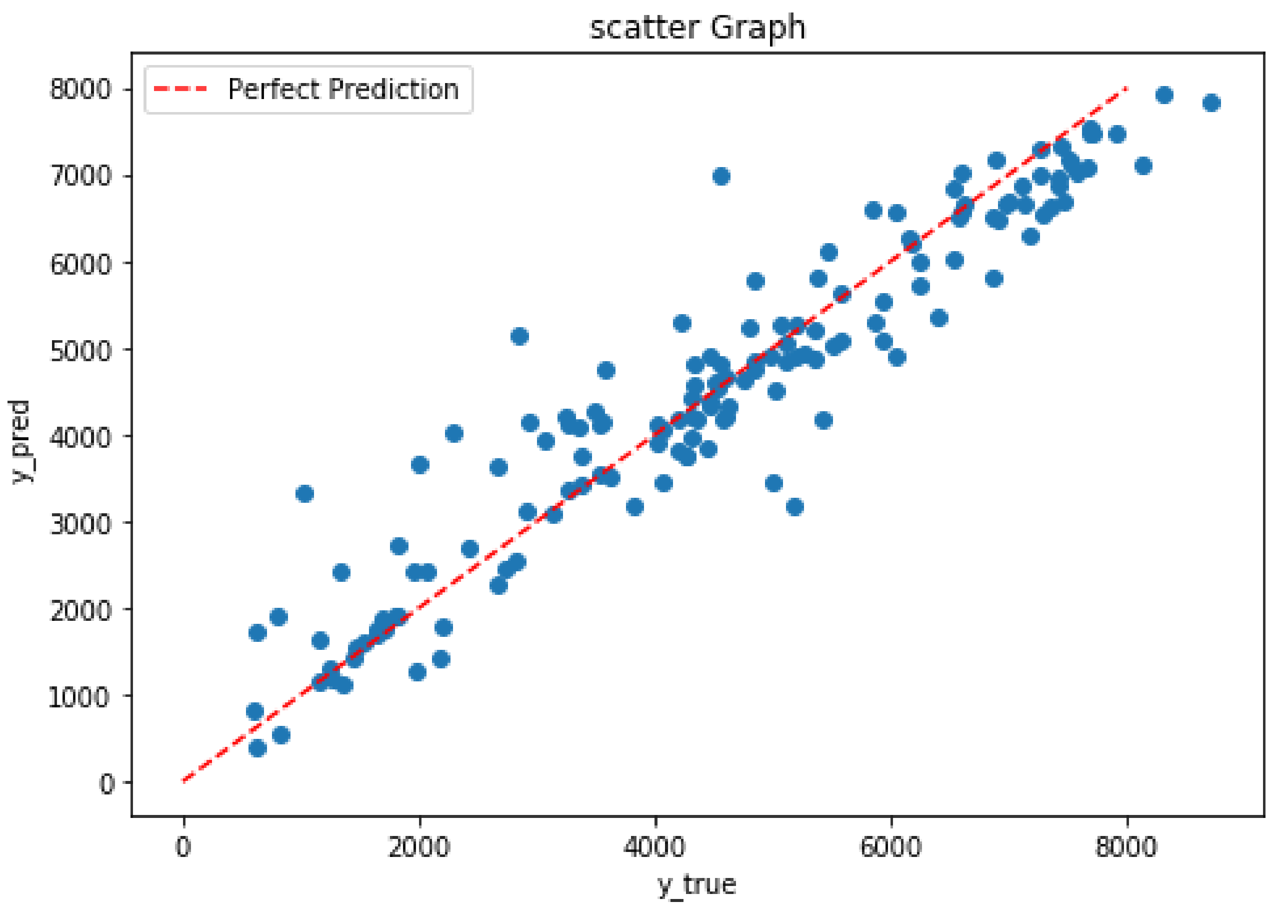
```
# Building XGBRegressor on tuned parameter
regressor = XGBRegressor(random_state=1, learning_rate=0.05, max_depth=3, n_estimators=300,
                         gamma = 0, subsample=0.8)
prediction(regressor, X_train, y_train, X_test, y_test)
```

```
[19:46:43] WARNING: C:/Jenkins/workspace/xgboost-win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now d
eprecated in favor of reg:squarederror.

C:\Users\admin\Anaconda3\lib\site-packages\xgboost\core.py:587: FutureWarning: Series.base is deprecated and will be removed
in a future version
  if getattr(data, 'base', None) is not None and \

[19:46:43] WARNING: C:/Jenkins/workspace/xgboost-win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now d
eprecated in favor of reg:squarederror.
K-fold (K = 10) explained variance
********************************************************************************************************************
0.8896462021670116

on train data explained variance
********************************************************************************************************************
0.9618164525345985

on test data explained variance
********************************************************************************************************************
0.8938652594631821
```

### Analysis:

By  tuning  parameters, we have increased explained_variance  score from 88.15% to 88.96%, which is less as XGBRegressor was giving us  better result without tuning. But  still  we  have  increase  somewhat  with  the  help    of hyperparameter tuning.

## Scatter Graph between y_true and y_pred:



## Analysis:

As we can see, we don’t have perfect predictions. Actually any model can’t have perfect prediction for such type of dataset, where customer taking bike on rent would have some randomness. Assume for two days all situations are same except date and they are nearby dates. Then also there would not be same counting of bike renting even with same situation. So, there would be some randomness in dataset which is natural. So, our model is predicting quite well as we can see from above image. Deviation for most of prediction from original value is low.

# Chapter 3

# Conclusion

## 3.1 Final Model and Training Dataset

### Final Dataset:

- Take whole dataset.
- Remove ‘instant’, ‘dteday’, ‘holiday’, ‘atemp’, ‘casual’, ‘registered’
- Make dummy variables of session and weathersit in python and factor in R.
- Make bins for ‘mnth’ and ‘weekday’ as explained in feature engineering section.
- Training set would be having all columns except ‘cnt’ and test dataset would have only ‘cnt’ column.

### Final Model:

- Use XGBRegressor model using training set.
- Do hyperparameter tuning for training set.
- Build XGBRegressor model with tuned parameters.

## 3.2 End Notes

- All the analysis results and plot diagrams are based on python. In R, result would not be exact same but would be almost same.
- We didn’t done time series analysis. As per problem statement, we have to predict for sessional and environmental condition.
- We can try with time series analysis for this project and can predict. In time series analysis we build model on trend for some specific time (like 2 year or more) and predict for future values like after time for which we have built model. While in this report we predicted for values which are in between.
- Outlier too has information so it should be treated well by gaining domain knowledge and experimenting with model building and checking performance.

- Steps which are done in EDA section is just not based on statistical test, We have done many experiments like whether to drop column or not and then decided to what to do with that specific column.

While doing hyperparameter tuning, after getting result of parameter, do again with finer values of parameter and so on. At the end we will get optimum values.

## Complete Python code:

```
# # **BIKE RENTING PROJECT**

#load libraries-

import os

import numpy as np

import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

from sklearn.linear_model import LinearRegression

from sklearn.neighbors import KNeighborsRegressor

from sklearn.svm import SVR

from sklearn.tree import DecisionTreeRegressor

from sklearn.ensemble import RandomForestRegressor

from xgboost import XGBRegressor

from sklearn.model_selection import GridSearchCV


#set working directory-

os.chdir("D:/MBA REF/edwisor/project/bikerental")


#check current working directory-

os.getcwd()

# Reading/Loading csv file
```

```
bike = pd.read_csv("day.csv")

# #                                 Exploratory Data Analysis

# ##  Understanding the data

#Checking top few observation of dataset

bike.head()

# looking at information of dataset

bike.info()

#numeric variables:

num_columns = ['temp', 'atemp', 'hum', 'windspeed', 'casual', 'registered', 'cnt']


#categorical variables:

cat_columns = ['season', 'yr', 'mnth', 'holiday', 'weekday', 'workingday', 'weathersit']


# looking at  summary for our numerical variables

bike[num_columns].describe()


# unique values of categories variables

bike[cat_columns].nunique()


# counting of each unique values in each categorical variable

print("value counts of categorical column")

print()

for i in cat_columns:

   print(i)

   print(bike[i].value_counts())
print("*****************************************************************
****************************************************")
```

```
# ## Missing value analysis

# checking for missing values in dataset

bike.isnull().sum()

# ## Outlier analysis

# user defined function that will plot boxplot and distribution for four column of dataset

def hist_and_box_plot(col1, col2, col3, col4, data, bin1=30, bin2=30, bin3=30, bin4 =
30, sup =""):

    fig, ax = plt.subplots(nrows = 2, ncols = 4, figsize= (14,6))

    super_title = fig.suptitle("Boxplot and Histogram: "+sup, fontsize='x-large')

    plt.tight_layout()

    sns.boxplot(y = col1, data = data, ax = ax[0][0])

    sns.boxplot(y = col2,data = data, ax = ax[0][1])

    sns.boxplot(y = col3, data = data, ax = ax[0][2])

    sns.boxplot(y = col4, data = data, ax = ax[0][3])

    sns.distplot(data[col1], ax = ax[1][0], bins = bin1)

    sns.distplot(data[col2], ax = ax[1][1], bins = bin2)

    sns.distplot(data[col3], ax = ax[1][2], bins = bin3)

    sns.distplot(data[col4], ax = ax[1][3], bins = bin4)

    fig.subplots_adjust(top = 0.90)

    plt.show()


# plotting boxplot and histogram for our numerical variables

hist_and_box_plot('temp', 'atemp', 'hum', 'windspeed', bin1 = 10, data = bike)
```

```
# ## Feature Engineering

# user defined function to plot bar plot of a column for each y i.e. y1 and y2 wrt


# unique variables of each x i.e. x1 and x2

# X1 and X2 would be categorical variable, y1 and y2 would be continuous

# this funciton will plot barplot for y1 column for each unique values of x1 and

# will do barplot for y2 for each unique values of x2 and method could be mean,sum
etc.

def plot_bar(x1, y1,x2, y2, method = 'sum'):

    fig, ax = plt.subplots(nrows = 1, ncols = 2, figsize= (12,4), squeeze=False)

    super_title = fig.suptitle("Bar Plot ",  fontsize='x-large')

    if(method == 'mean'):

        gp = bike.groupby(by = x1).mean()

        gp2 = bike.groupby(by = x2).mean()

    else:

        gp = bike.groupby(by = x1).sum()

        gp2 = bike.groupby(by = x2).sum()

    gp = gp.reset_index()

    gp2 = gp2.reset_index()

    sns.barplot(x= x1, y = y1, data = gp, ax=ax[0][0])

    sns.barplot(x= x2, y = y2, data = gp2, ax=ax[0][1])

    fig.subplots_adjust(top = 0.90)

    plt.show()

# plotting barplot for count i.e. cnt wrt to yr and month

plot_bar('yr', 'cnt', 'mnth', 'cnt')
```

```
#plotting barplot for count wrt month for each year

gp = bike.groupby(by = ['yr', 'mnth']).sum().reset_index()

sns.factorplot(x= 'mnth', y = 'cnt', data = gp, col = 'yr', kind = 'bar')

# ploting barplot for counting wrt weekday

plot_bar('weekday', 'cnt', 'weekday', 'cnt')


# defining function for making bins in mnth and weekday

def feat_month(row):

    if row['mnth'] <= 4 or row['mnth'] >=11:

        return(0)

    else:

        return(1)

def feat_weekday(row):

    if row['weekday'] < 2:

        return(0)

    else:

        return(1)

bike['month_feat'] = bike.apply(lambda row : feat_month(row), axis=1)

bike = bike.drop(columns=['mnth'])

bike['week_feat'] = bike.apply(lambda row : feat_weekday(row), axis=1)

bike= bike.drop(columns=['weekday'])



# ## Feature Selection

# let us look at correlation plot for each numerical variables

sns.pairplot(bike[num_columns])
```

```
# let us look at heat map for each numerical variable

# with correlation

fig = plt.figure(figsize = (14,10))

corr = bike[num_columns].corr()

sn_plt = sns.heatmap(corr, mask = np.zeros_like(corr, dtype = np.bool), square = True,

            annot= True, cmap = sns.diverging_palette(220, 10, as_cmap= True))

plt.title("Heat Map of correlation between numerical variables")

fg = sn_plt.get_figure()

fg.savefig('heatmap.png')

# chi-square test for each categorical variable

cat_columns = ['season', 'yr', 'month_feat', 'holiday', 'week_feat', 'workingday', 
'weathersit']

# making every combination from cat_columns

factors_paired = [(i,j) for i in cat_columns for j in cat_columns]

# doing chi-square test for every combination

p_values = []

from scipy.stats import chi2_contingency

for factor in factors_paired:

    if factor[0] != factor[1]:

        chi2, p, dof, ex = chi2_contingency(pd.crosstab(bike[factor[0]],

                                                        bike[factor[1]]))

        p_values.append(p.round(3))

    else:

        p_values.append('-')

p_values = np.array(p_values).reshape((7,7))

p_values = pd.DataFrame(p_values, index=cat_columns, columns=cat_columns)
```

```
print(p_values)

# checking importance of feature

drop_col = ['cnt', 'instant','dteday', 'registered', 'casual']

from sklearn.ensemble import ExtraTreesRegressor

reg = ExtraTreesRegressor(n_estimators=200)

X = bike.drop(columns= drop_col)

y = bike['cnt']

reg.fit(X, y)

imp_feat = pd.DataFrame({'Feature': bike.drop(columns=drop_col).columns,

                    'importance':reg.feature_importances_})

imp_feat.sort_values(by = 'importance', ascending=False).reset_index(drop = True)

# checking vif of numerical column withhout dropping multicollinear column

from statsmodels.stats.outliers_influence import variance_inflation_factor as vf

from statsmodels.tools.tools import add_constant

numeric_df = add_constant(bike[['temp', 'atemp', 'hum', 'windspeed']])

vif = pd.Series([vf(numeric_df.values, i) for i in range(numeric_df.shape[1])],

             index = numeric_df.columns)

vif.round(1)

# Checking VIF values of numeric columns after dropping multicollinear column i.e.
atemp

from statsmodels.stats.outliers_influence import variance_inflation_factor as vf

from statsmodels.tools.tools import add_constant

numeric_df = add_constant(bike[['temp', 'hum', 'windspeed']])

vif = pd.Series([vf(numeric_df.values, i) for i in range(numeric_df.shape[1])],

             index = numeric_df.columns)

vif.round(1)
```

```
# Making dummies for each category session and weather

season_dm = pd.get_dummies(bike['season'], drop_first=True, prefix='season')

bike = pd.concat([bike, season_dm],axis=1)

bike = bike.drop(columns = ['season'])

weather_dm = pd.get_dummies(bike['weathersit'], prefix= 'weather',drop_first=True)

bike = pd.concat([bike, weather_dm],axis=1)

bike = bike.drop(columns= ['weathersit'])

# ## Data after    Exploratory Data Analysis

# creating another dataset with dropping outliers i.e. bikeo

bikeo = bike.copy()

# dropping outliers from boxplot method

for i in ['windspeed', 'hum']:

    q75, q25 = np.percentile(bikeo.loc[:,i], [75 ,25])

    iqr = q75 - q25

    min = q25 - (iqr*1.5)

    max = q75 + (iqr*1.5)

    bikeo = bikeo.drop(bikeo[bikeo.loc[:,i] < min].index)

    bikeo = bikeo.drop(bikeo[bikeo.loc[:,i] > max].index)

# dropping unwanted columns from both dataset bike and bikeo

bike.drop(columns=['instant', 'dteday', 'holiday', 'atemp', 'casual', 'registered'],
inplace=True)

bikeo.drop(columns=['instant', 'dteday', 'holiday', 'atemp', 'casual', 'registered'],
inplace=True)

bike.head()

bikeo.head()

#Shape of the dataset with and without outliers
```

```
print('shape of original dataset',bike.shape)

print('shape of dataset after removing outliers', bikeo.shape)

# ## Building models

# making a function which will build model on training set and would show result

# for k-fold cv explained_variance_score and also show result for training and test
dataset

from sklearn.metrics import explained_variance_score

from sklearn.model_selection import cross_val_score

def prediction(regressor, X_train, y_train, X_test, y_test):

    regressor.fit(X_train, y_train)

    y_pred = regressor.predict(X_test)

    ten_performances = cross_val_score(estimator=regressor, X = X_train, y = y_train,
cv = 10, scoring='explained_variance')

    k_fold_performance = ten_performances.mean()

    print("K-fold (K = 10) explained variance")


print("********************************************************************
*****************************************************")

    print(k_fold_performance)

    print()

    print("on train data explained variance")


print("********************************************************************
*****************************************************")

    print(regressor.score(X_train, y_train))

    print()

    print("on test data explained variance")
print("********************************************************************
*****************************************************")
```

```
print(regressor.score(X_test, y_test))

# splitting dataset in train and test for whole dataset after eda i.e. bike

X = bike.drop(columns=['cnt'])

y = bike['cnt']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state =
0)

# splitting dataset in train and test for dataset without outlier after eda i.e. bikeo

X = bikeo.drop(columns=['cnt'])

y = bikeo['cnt']

X_traino, X_testo, y_traino, y_testo = train_test_split(X, y, test_size = 0.2,
random_state = 0)

# #   Linear Regression

# Building model for dataset bike

regressor = LinearRegression()

prediction(regressor, X_train, y_train, X_test, y_test)


# building model for dataset bikeo i.e. without  outliers

regressor = LinearRegression()

prediction(regressor, X_traino, y_traino, X_testo, y_testo)

# #         KNN

# building model for dataset bike

regressor = KNeighborsRegressor(n_neighbors=5)

prediction(regressor, X_train, y_train, X_test, y_test)

# building model for dataset bikeo i.e. without outliers

regressor = KNeighborsRegressor(n_neighbors=5)

prediction(regressor, X_traino, y_traino, X_testo, y_testo)
```

```
# #       SVM
# building model for dataset bike
regressor = SVR()
prediction(regressor, X_train, y_train, X_test, y_test)
# building model for dataset bikeo i.e. without outliers
regressor = SVR()
prediction(regressor, X_traino, y_traino, X_testo, y_testo)
# # Decision Tree Regression
# building model for dataset bike
regressor = DecisionTreeRegressor(random_state=1)
prediction(regressor, X_train, y_train, X_test, y_test)
# building model for dataset bikeo i.e. without outliers
prediction(regressor, X_traino, y_traino, X_testo, y_testo)
# #  Random Forest
# building model for dataset bike
regressor = RandomForestRegressor(random_state=1)
prediction(regressor, X_train, y_train, X_test, y_test)
# building model for dataset bikeo i.e. without outliers
regressor = RandomForestRegressor(random_state=1)
prediction(regressor, X_traino, y_traino, X_testo, y_testo)
# #     XGBRegressor
# building model for dataset bike
regressor = XGBRegressor(random_state=1)
prediction(regressor, X_train, y_train, X_test, y_test)
```

```
# building model for dataset bikeo i.e. without outliers
regressor = XGBRegressor(random_state=1)
prediction(regressor, X_traino, y_traino, X_testo, y_testo)
# #       Hyperparameter tuning
# Tuning Random Forest for bike dataset
# Random Forest hyperparameter tuning
regressor = RandomForestRegressor(random_state=1)
params = [{'n_estimators' : [500, 600, 800],'max_features':['auto', 'sqrt', 'log2'],
          'min_samples_split':[2,4,6],'max_depth':[12, 14, 16],'min_samples_leaf':[2,3,5],
          'random_state' :[1]}]
grid_search = GridSearchCV(estimator=regressor, param_grid=params,cv = 5,
                        scoring = 'explained_variance', n_jobs=-1)
grid_search = grid_search.fit(X_train, y_train)
print(grid_search.best_params_)


# Building Random Forest on tuned parameter
regressor = RandomForestRegressor(random_state=1, max_depth=14,
n_estimators=600,
                              max_features='auto', min_samples_leaf=2,min_samples_split=2)
prediction(regressor, X_train, y_train, X_test, y_test)
# Tuning XGBRegressor for bike dataset
regressor = XGBRegressor(random_state=1)
params = [{'n_estimators' : [250, 300,350, 400,450], 'max_depth':[2, 3, 5],
          'learning_rate':[0.01, 0.045, 0.05, 0.055, 0.1, 0.3],'gamma':[0, 0.001, 0.01, 0.03],
          'subsample':[1, 0.7, 0.8, 0.9],'random_state' :[1]}]
```

```
grid_search = GridSearchCV(estimator=regressor, param_grid=params,cv = 5,
                    scoring = 'explained_variance', n_jobs=-1)


grid_search = grid_search.fit(X_train, y_train)

print(grid_search.best_params_)


# Building XGBRegressor on tuned parameter

regressor = XGBRegressor(random_state=1, learning_rate=0.05, max_depth=3,
n_estimators=300,

                  gamma = 0, subsample=0.8)

prediction(regressor, X_train, y_train, X_test, y_test)


# plotting scatter graph for y_true and y_pred for tuned XGBRegressor model

regressor = XGBRegressor(random_state=1, learning_rate=0.05, max_depth=3,
n_estimators=300,

                  gamma = 0, subsample=0.8)

regressor.fit(X_train, y_train)

y_pred = regressor.predict(X_test)

fig, ax = plt.subplots(figsize=(7,5))

ax.scatter(y_test, y_pred)

ax.plot([0,8000],[0,8000], 'r--', label='Perfect Prediction')

ax.legend()

plt.title("scatter Graph")

plt.xlabel("y_true")

plt.ylabel("y_pred")

plt.tight_layout()

plt.show()
```

## Complete R code:

```
#remove all the objects stored

rm(list=ls())


#set current working directory

setwd("D:/MBA REF/edwisor/project/bikerental")


#Current working directory

getwd()

# importing all required library

Packages <- c("ggplot2", "corrgram", "corrplot", "randomForest",

                "caret", "class", "e1071", "rpart", "mlr","grid",

                "DMwR","usdm","dplyr","caTools","LiblineaR")



lapply(Packages, library, character.only = TRUE)


# Reading/Loading the csv file

bike = read.csv("day.csv")



# ****************************  Exploratory Data Analysis
***************************



# ******************* Understanding the data
**********************************
```

```
# checking datatypes of all columns

class(bike)

dim(bike)

head(bike)

names(bike)

str(bike)

summary(bike)


#numeric variables:

num_columns <- c('temp', 'atemp', 'hum', 'windspeed', 'casual', 'registered', 'cnt')


#categorical varibles:

cat_columns <- c('season', 'yr', 'mnth', 'holiday', 'weekday', 'workingday', 'weathersit')


# ******************  Checking numerical variables
**************************


# Checking numerical statistics of numerical columns:

summary(bike[,num_columns])

# ******************  Checking categorical variables
**************************

#unique values in each category:

lapply(bike[,cat_columns], function(feat) length(unique(feat)))


#counting each unique values in categorical columns:

lapply(bike[,cat_columns], function(feature) table(feature))
```

```
# ***********************  Missing value analysis ***********************


#checking missing value for each column in the bike dataset and storing the counting in
dataframe with column name

missing_val <- data.frame(lapply(bike, function(feat) sum(is.na(feat))))

View(missing_val)


# ********************* Outlier Analysis ************************

#box_plot function to find outliers in  numerical columns

box_plot <- function(column, dataset){

  dataset$x = 1


  ggplot(aes_string(x= 'x', y = column), data = dataset)+

    stat_boxplot(geom = 'errorbar', width = 0.5)+

    geom_boxplot(outlier.size = 2, outlier.shape = 18)+

    labs(y = "", x = column)+

    ggtitle(paste("BOX:",column))

}


#ploting histogram of numerical variable

hist_plot <- function(column, dataset){

  ggplot(aes_string(column), data = dataset)+

    geom_histogram(aes(y=..density..), fill = 'skyblue2')+

    geom_density()+

    labs(x = gsub('\\.', ' ', column))+

    ggtitle(paste("HIST:",gsub('\\.', ' ', column)))

}
```

```
#calling box_plot function and storing all plots in a list

all_box_plots <- lapply(c('temp', 'atemp', 'hum', 'windspeed'),box_plot, dataset = bike)


#calling hist_plot function and storing all plots in a list

all_hist_plots <- lapply(c('temp', 'atemp', 'hum', 'windspeed'),hist_plot, dataset = bike)


# printing all plots in one

gridExtra::grid.arrange(all_box_plots[[1]],all_box_plots[[2]],all_box_plots[[3]],all_box
_plots[[4]],
all_hist_plots[[1]],all_hist_plots[[2]],all_hist_plots[[3]],all_hist_plots[[4]],ncol=4,nrow
=2)



# ************************** Feature Engineering
****************************


#plot barplot of a columns with respect to other column

plot_bar <- function(cat, y, fun){

  gp = aggregate(x = bike[, y], by=list(cat=bike[, cat]), FUN=fun)

  ggplot(gp, aes_string(x = 'cat', y = 'x'))+

    geom_bar(stat = 'identity')+

    labs(y = y, x = cat)+

    ggtitle(paste("Bar plot for",y,"wrt to",cat))

}

# plotting cnt with respect to month

plot_bar('mnth', 'cnt', 'sum')
```

```
# plotting cnt with respect to yr

plot_bar('yr', 'cnt', 'sum')

# plotting cnt with respect to yr

plot_bar('weekday', 'cnt', 'sum')

#creating bins of mnth and weekday

#changing values of month 5th to 10th as 1 and others 0

bike = transform(bike, mnth = case_when(

  mnth <= 4 ~ 0,

  mnth >= 11 ~ 0,

  TRUE   ~ 1

))


colnames(bike)[5] <- 'month_feat'


# changing values of weekday for day 0 and 1 the value will be 0

#and 1 for rest

bike = transform(bike, weekday = case_when(

  weekday < 2 ~ 0,

  TRUE   ~ 1

))

colnames(bike)[7] <- 'week_feat'


#                              Feature Selection
```

```
#correlation plot for numerical feature:

corrgram(bike[,num_columns], order = FALSE,

         upper.panel = panel.pie, text.panel = panel.txt,

         main = "Correlation Plot for bike dataset")


# heatmap plot for numerical features

corrplot(cor(bike[,num_columns]), method = 'color', type = 'lower')


cat_columns <- c('season', 'yr', 'month_feat', 'holiday', 'week_feat', 'workingday',
'weathersit')

#making every combination from cat_columns

combined_cat <- combn(cat_columns, 2, simplify = F)


#doing chi-square test for every combination

for(i in combined_cat){

  print(i)

  print(chisq.test(table(bike[,i[1]], bike[,i[2]])))

}


# finding important features

important_feat <- randomForest(cnt ~ ., data = bike[,-c(1,2,14,15)],

                               ntree = 200, keep.forest = FALSE, importance = TRUE)

importance_feat_df <- data.frame(importance(important_feat, type = 1))


# checking vif of numerical column withhout dropping multicollinear column

vif(bike[,c(10,11,12,13)])
```

```
# Checking VIF values of numeric columns after dropping multicollinear column i.e.
atemp

vif(bike[,c(10,12,13)])


# Making factor datatype to each category

bike[,cat_columns] <- lapply(bike[,cat_columns], as.factor)


# releasing memory of R, removing all variables except dataset

rm(list = setdiff(ls(),"bike"))

# ************************  Data after Exploratory Data Analysis
*******************



# creating another dataset with dropping outliers i.e. bikeo

bikeo <- bike


# removing outliers from hum and windspeed columns

for (i in c('hum', 'windspeed')){

  out_value = bikeo[,i] [bikeo[,i] %in% boxplot.stats(bikeo[,i])$out]

  bikeo = bikeo[which(!bikeo[,i] %in% out_value),]

}


# checking dimension of both dataset

dim(bike)

dim(bikeo)
```

```
# dropping unwanted columns

drop_col <- c('instant', 'dteday', 'holiday', 'atemp', 'casual', 'registered')

bike[,drop_col]<- NULL

bikeo[,drop_col] <- NULL


# ****************************  Building models
********************************


set.seed(1)

split = sample.split(bike$cnt, SplitRatio = 0.80)

train_set = subset(bike, split == TRUE)

test_set = subset(bike, split == FALSE)

split = sample.split(bikeo$cnt, SplitRatio = 0.80)


train_set_wo = subset(bikeo, split == TRUE)

test_set_wo = subset(bikeo, split == FALSE)


# making a function which will train model on training data and would show

# K-fold R2 score , R2 score for test dataset and train dataset

prediction <- function(method, train_data, test_data){

  reg_fit <- caret::train(cnt~., data = train_data, method = method)


  y_pred <- predict(reg_fit, test_data[,-10])

  print("R2 on test dataset")

  print(caret::R2(y_pred, test_data[,10]))

    y_pred <- predict(reg_fit, train_data[,-10])
print("R2 on train dataset")
  print(caret::R2(y_pred, train_data[,10]))


  # creating 10 folds of data
  ten_folds = createFolds(train_data$cnt, k = 10)
  ten_cv = lapply(ten_folds, function(fold) {
    training_fold = train_data[-fold, ]
    test_fold = train_data[fold, ]
    reg_fit <- caret::train(cnt~., data = training_fold, method = method)


    y_pred <- predict(reg_fit, test_fold[,-10])
    return(as.numeric(caret::R2(y_pred, test_fold[,10])))
  })


  sum = 0
  for(i in ten_cv){
    sum = sum + as.numeric(i)
  }
  print("K-fold (K =10) explained variance")
  print(sum/10)
}



# ***************************  Linear Regression
****************************

# building model for dataset bike_data
```

```
prediction('lm', train_set, test_set)


# building model for dataset bike_data_wo i.e. without  outliers

prediction('lm', train_set_wo, test_set_wo)


# *******************************  KNN
***************************************


# building model for dataset bike_data

prediction('knn', train_set, test_set)


# building model for dataset bike_data_wo i.e. without  outliers

prediction('knn', train_set_wo, test_set_wo)

# *********************************  SVM
*************************************


# building model for dataset bike_data

prediction('svmLinear3', train_set, test_set)


# building model for dataset bike_data_wo i.e. without  outliers

prediction('svmLinear3', train_set_wo, test_set_wo)


# ****************************** Decision Tree Regression
********************


# building model for dataset bike_data

prediction('rpart2', train_set, test_set)
```

```
# building model for dataset bike_data_wo i.e. without  outliers

prediction('rpart2', train_set_wo, test_set_wo)


# **********************************  Random Forest
*************************


# building model for dataset bike_data

prediction('rf', train_set, test_set)


# building model for dataset bike_data_wo i.e. without  outliers

prediction('rf', train_set_wo, test_set_wo)


# ***********************************  XGBRegressor
*************************



# building model for dataset bike_data

prediction('xgbTree', train_set, test_set)


# building model for dataset bike_data_wo i.e. without  outliers

prediction('xgbTree', train_set_wo, test_set_wo)


#************************ Hyperparameter tuning
**************************************


#************ Tuning Random Forest for bike_data dataset
****************************
```

```
control <- trainControl(method="repeatedcv", number=10, repeats=3)

reg_fit <- caret::train(cnt~., data = train_set, method = "rf",trControl = control)

reg_fit$bestTune

y_pred <- predict(reg_fit, test_set[,-10])

print(caret::R2(y_pred, test_set[,10]))


# *******************  Tuning XGB for bike_data dataset
*****************************

control <- trainControl(method="repeatedcv", number=10, repeats=3)

reg_fit <- caret::train(cnt~., data = train_set, method = "xgbTree",trControl = control)

reg_fit$bestTune

y_pred <- predict(reg_fit, test_set[,-10])

print(caret::R2(y_pred, test_set[,10]))
```

## References:

- https://learning.edwisor.com/
- https://www.udemy.com/machinelearning/
- https://medium.com/greyatom/why-how-and-when-to-scale-your-features-4b30ab09db5e
- https://towardsdatascience.com/data-pre-processing-techniques-you-should-know-8954662716d6
- https://towardsdatascience.com/machine-learning-general-process-8f1b510bd8af
- https://github.com/topics/machine-learning

**THANK YOU**