

Bike Renting Prediction

Ravi Srivastava

25 November, 2019

Contents

1. Introduction

1.1 Problem Statement

1.2 Understanding Data

2. Exploratory Data Analysis

2.1 Insight of Raw Data

2.2 Data Preparation

3. Data Pre Processing

3.1 Missing Value Analysis

3.2 Outlier Analysis

3.3 Feature Selection

3.3.1 Correlation Analysis

3.3.2 Chi Square test of independence

3.3.3 Multicollinearity

3.3.4 Summary of feature Selection

3.4 Feature Scaling

4. Data Modeling

4.1 K-fold CV , GridsearchCV and Explained_variance

4.2 Building Models

4.2.1 Linear Regression

4.2.2 KNN

4.2.3 Support Vector Regressor

4.2.4 Decision Tree Regressor

4.2.5 Random Forest Regressor

4.2.6 XGBRegressor

4.3 Hyperparameter Tuning

4.3.1 Random Forest Hyperparameter Tuning

4.3.2 XGBRegressor Hyperparameter Tuning

5. Conclusion

5.1 Final Model and Dataset

5.2 Final Notes

Appendix A – Complete Python Code

Appendix B – Complete R code Link

References

Chapter 1: Introduction

1.1 Problem Statement

A **bicycle-renting system** is a service in which users can rent/use bicycles available for shared use on a short term basis for a price or free. Currently, there are over 500 bike-sharing programs around the world. Such systems usually **aim to reduce congestion, noise, and air pollution** by providing free/affordable access to bicycles for short-distance trips in an urban area as opposed to motorized vehicles. The number of users on any given day can vary greatly for such systems. The ability to predict the number of hourly users can allow the entities (businesses/governments) that oversee these systems to manage them in a more efficient and cost-effective manner.



So, the aim of this project is to predict the count of bike rentals based on the seasonal and environmental settings. By predicting the count, it would be possible to help accommodate in managing the number of bikes required on a daily basis, and being prepared for high demand of bikes during peak periods.

1.2 Understanding Data

We have total of 16 columns, in which we have 13 independent variables and 3 dependent variables. All columns have name as they normally stands for. Details are below:

Dependent Variables:

‘casual’, ‘registered’ and ‘cnt’ (dependent variables) are the counting of renting of bikes for a particular day.

1. **casual** : count of non-registered users,
2. **registered**: count of registered users,
3. **cnt**: is for total counting i.e., casual + registered users.

Independent Variables:

1. **dteday** : Date
2. **season**: Season (1:springer, 2:summer, 3:fall, 4:winter)
3. **yr**: Year (0: 2011, 1:2012)
4. **mnth**: Month (1 to 12)
5. **hr**: Hour (0 to 23)
6. **holiday**: weather day is holiday or not (extracted from Holiday 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)

1: Clear, Few clouds, Partly cloudy, Partly cloudy

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

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

4: 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

$(t - t_{\min}) / (t_{\max} - t_{\min})$, $t_{\min}=-16$, $t_{\max}=+50$ (only in hourly scale)

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

13. windspeed: Normalized wind speed. The values are divided to 67 (max)

Chapter 2 : Exploratory Data Analysis

2.1 Insight of the raw data

Before starting to process a data-set with algorithms, it's always a good idea to explore it visually. To understand what the raw data is, what it looks like and to understand the parameters, we need to look at various graph plots.

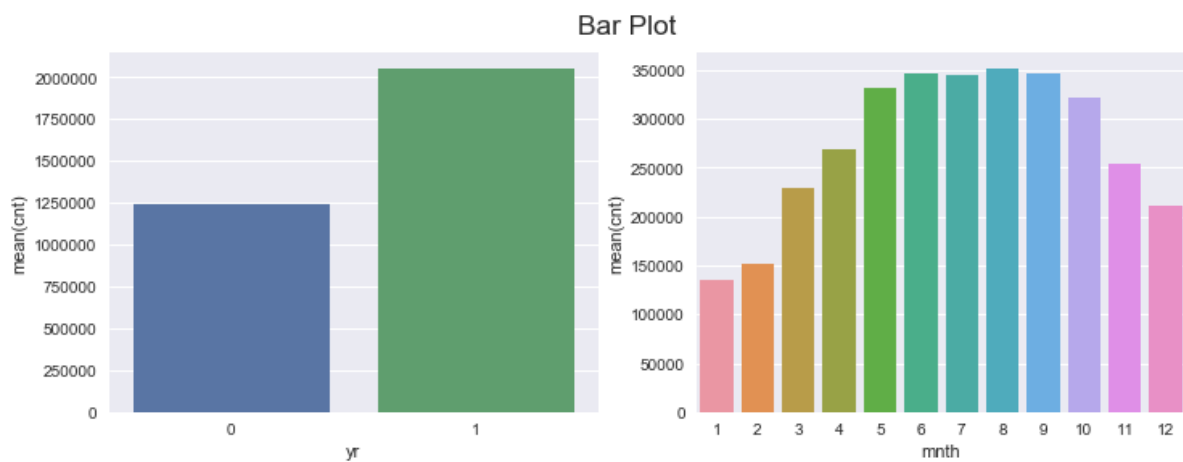


Fig. 2.1

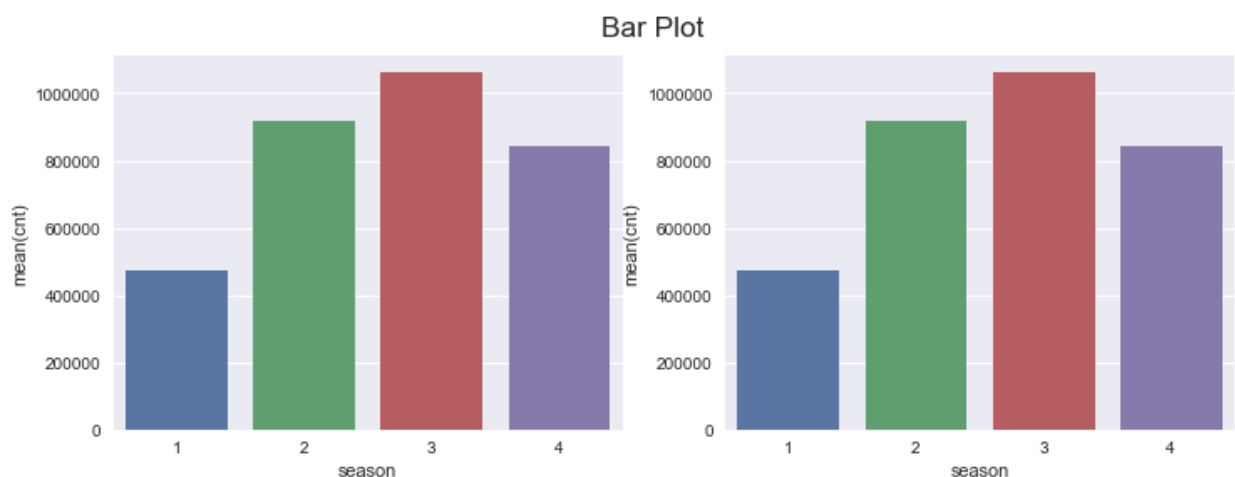


Fig. 2.2

i) Fig 2.2 shows that the rent of bikes is lowest in spring season (season 1) and highest in fall (season 3) this can be concluded from the fact that months (1,2,3,12) , (3,4,5,6) , (6,7,8,9) and (9,10,11,12) are seasons 1,2 ,3,4 respectively.

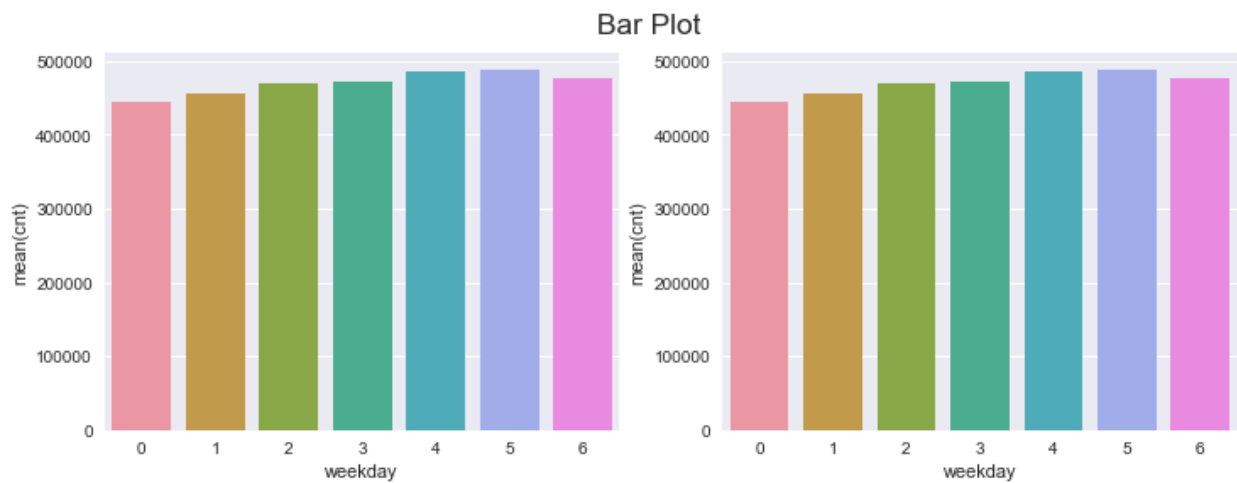


Fig. 2.3

ii) Fig 2.3 shows that the weekdays doesn't affect the rent count much and the mean count is almost same throughout the days.

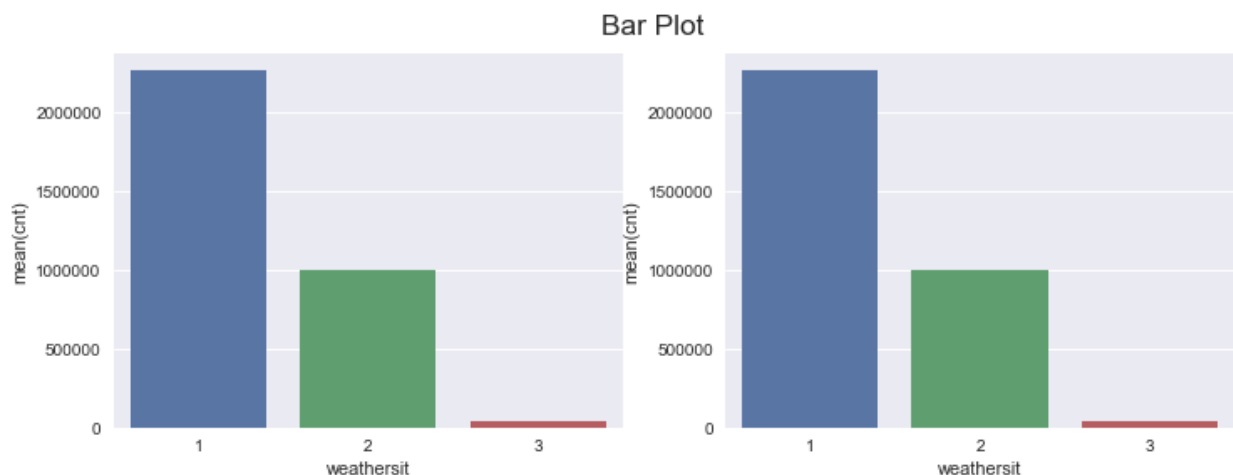


Fig. 2.4

iii) Fig 2.4 suggests that weather situation 1 is completely suitable for bikers, while weather situation 3 completely disrupts the bike renting.

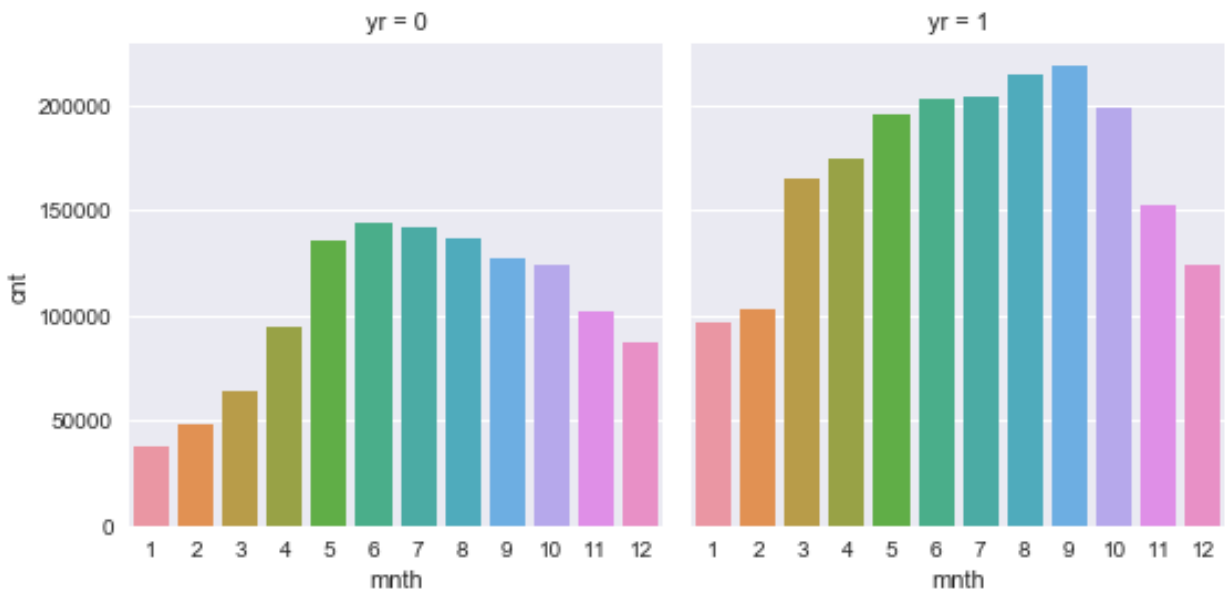


Fig 2.5

iv) Fig 2.5 shows that there is incremental in the bike renting from year 2010 to 2011.

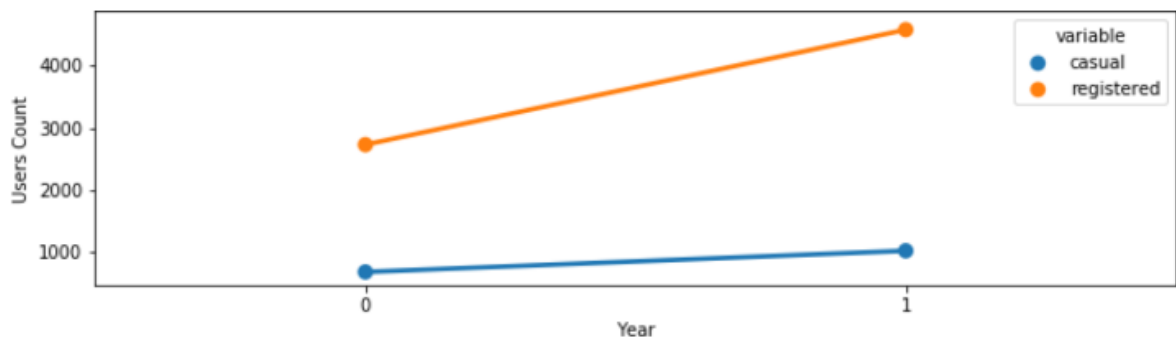


Fig 2.6: Distribution of count by yr divided into casual and registered users.

v) Although the amount of total rent bikers has increased over the year as shown in fig 2.5 , amount of casual users have not increased much but count of registered users have increased a lot (increased sharply).

We are going to use **R** for this project. **ggplot2** and **ggextra** are the packages, using which we can quickly make some plots to investigate how the bicycle usage count is affected by the features available. Now let's look at some graphs.

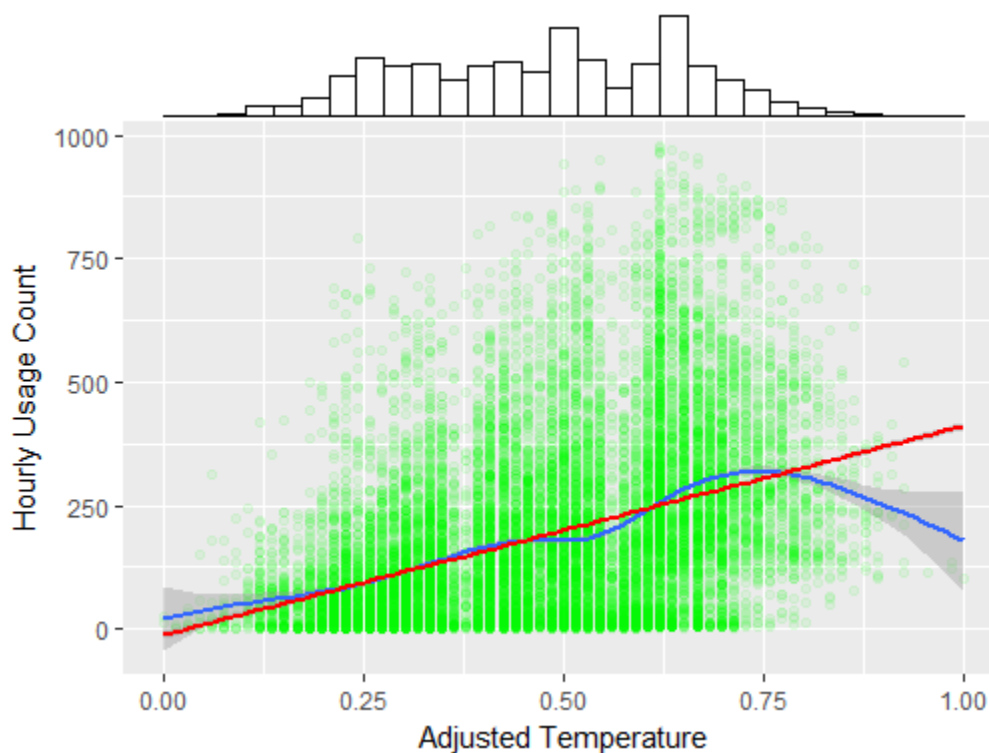


Fig. 2.1.1 Scatter plot- Adjusted Temperature vs Usage

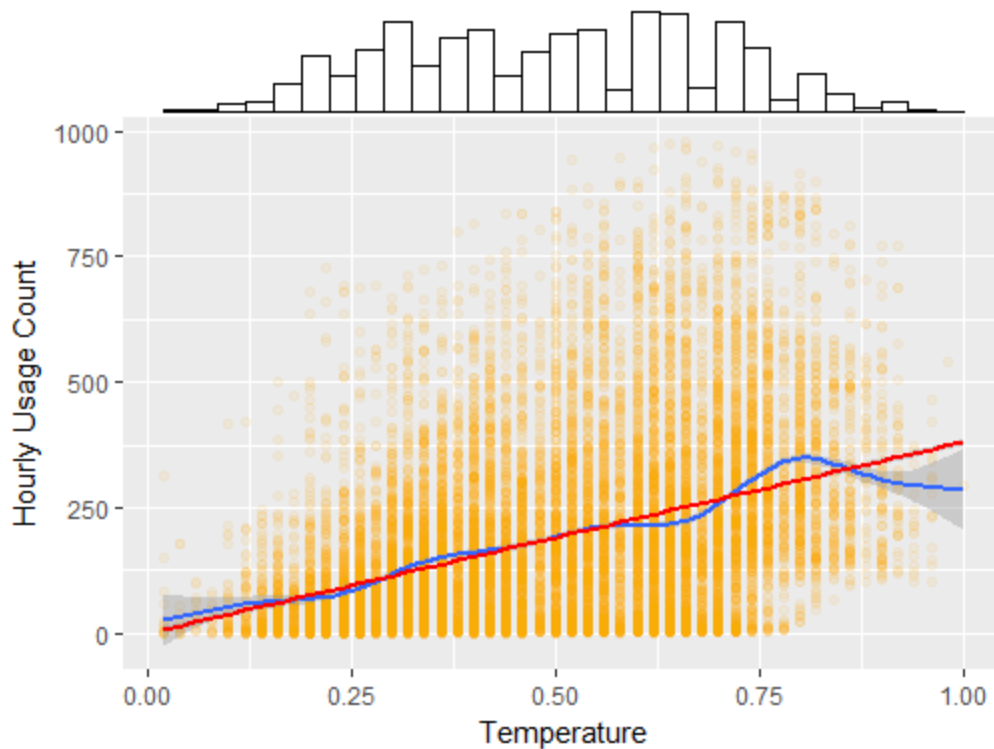


Fig. 2.1.2 Scatter plot- Temperature vs Usage

As seen from the scatter plots above, there is a positive correlation between both temperature-to-usage and adjusted-temperature-to-usage for most of the temperature range, and a linear fit isn't far from the best-fit curve. This should intuitively make sense, as people are not likely to bike outside in cold weather. For the maximum temperatures, which seem to be a small subset of the data, there is a dip in this curve. Once again, this should make sense as users may also be discouraged to bike when it's too hot outside.

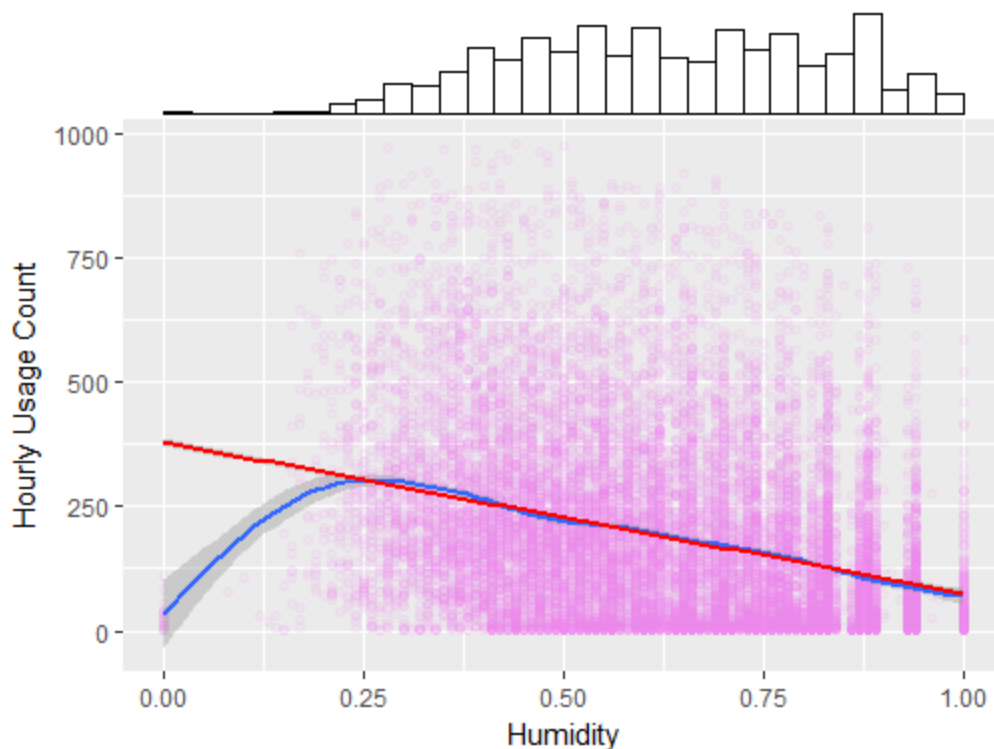


Fig. 2.1.3 Scatter plot- Humidity vs Usage

There seems to be a negative correlation between the humidity and the usage rate, with a linear fit being very close to the best curve fit for all of the data (excluding some outliers with very low humidity). This could be explained by the climate of Washington DC, which is notoriously humid. Higher humidity is correlated with higher chances of rainfall. We could also hypothesize that added perspiration due to high humidity works as a deterrent to riding a bicycle outdoors. Based on the visualizations so far, it wouldn't be unreasonable for us to hypothesize that the weather-situation will affect the bike usage, with rainfall deterring usage. Our next graph, plotted below, supports this hypothesis, to an extent. Note from the histogram (on the x-axis) that there are many more clear days (weather sit 1) than overcast or rainy days (weather sit 2 or 3 respectively).

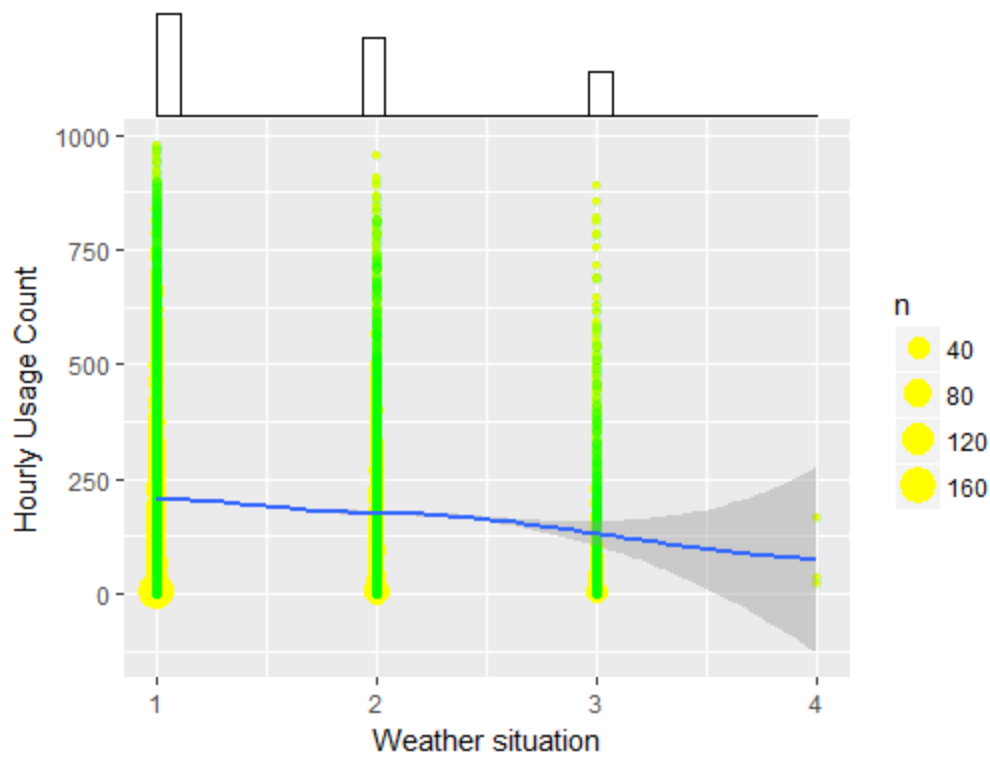


Fig. 2.1.4 Scatter plot- Weather Situation vs Usage

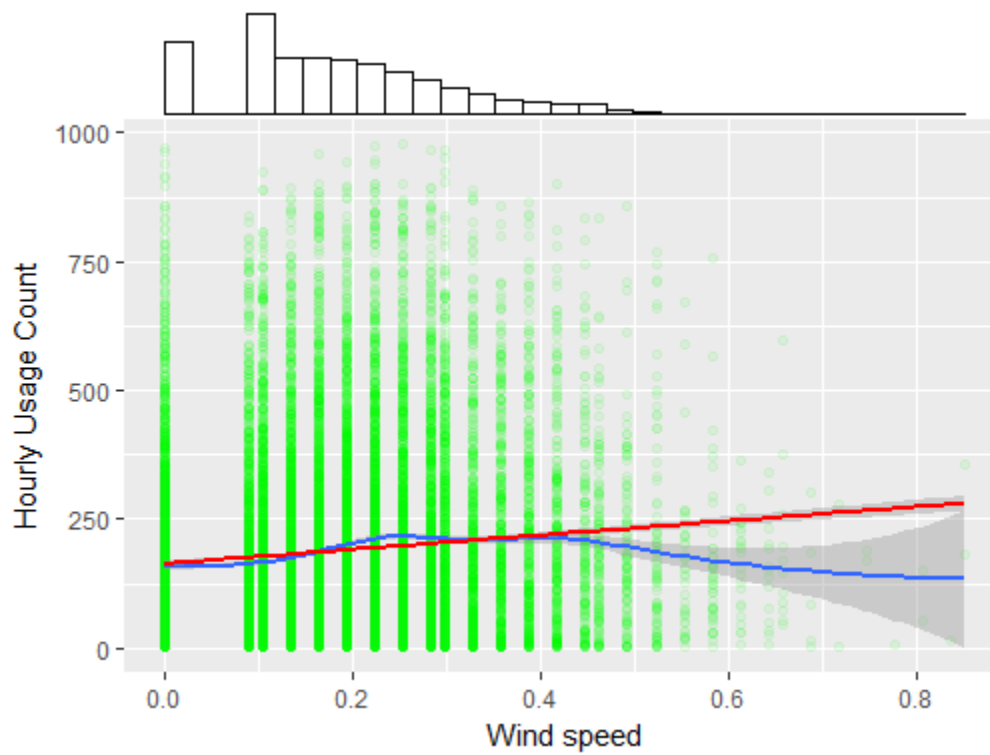


Fig 2.1.5 Scatter plot- Wind Speed vs Usage

Looking at the wind speed data, however, doesn't give us a clear interpretation of how it affects usage. The correlation between the two factors is weak at best.

2.2 Data Preparation

Also we will look at our data and datatype:

```
<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: 88.6+ 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 Freemetoo.

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. So our problem would be regression problem.

Let us now analyze our numerical data- We will check summary of our numerical data or we say five point summary:

The screenshot shows a Jupyter Notebook titled "Project Bike renting prediction". The code cell [86] contains the following code:

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

In [86]: # Looking at five point summary for our numerical variables
bike_data[numerical_columns].describe()
```

The output [86] is a table showing the five-point summary for 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

The code cell [87] contains the following code:

```
In [87]: # unique values of categories variables
bike_data[cat_columns].nunique()
```

The output [87] is a table showing the unique values for categorical variables:

	season	yr	mnth	holiday	weekday	workingday
count	4	2	12	2	7	2

Analysis:

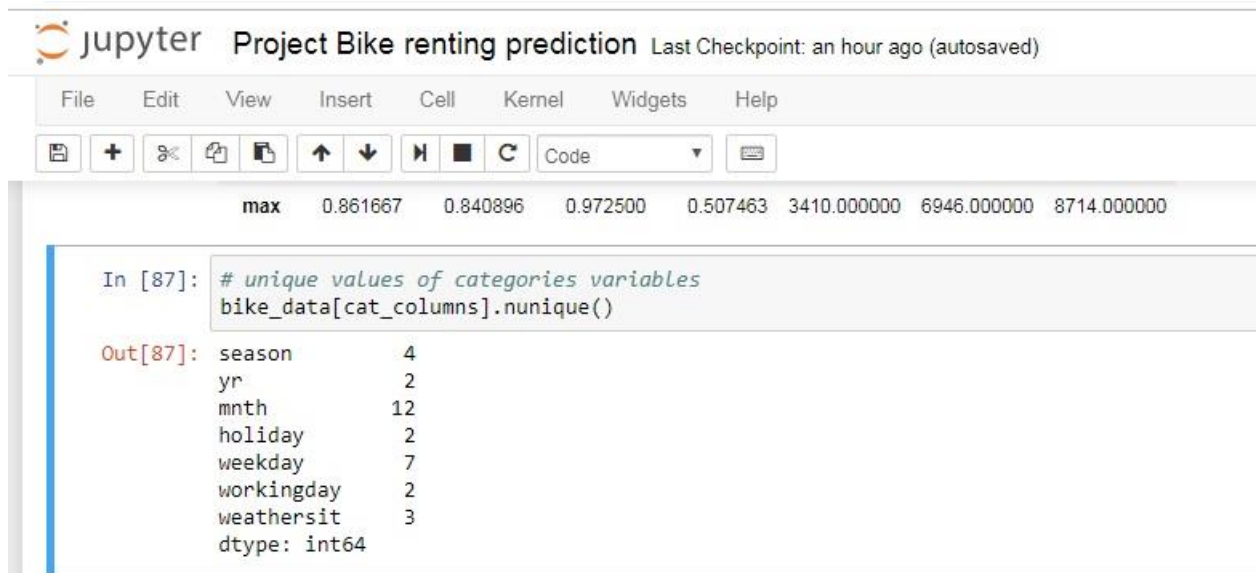
We have four independent continuous variables and three dependent continuous variables.

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

'cnt' i.e. target variable have minimum 22 counting and maximum 8714 counting.

Checking categorical data Now:

Finding number of unique values in each category:



The screenshot shows a Jupyter Notebook titled "Project Bike renting prediction" with a menu bar (File, Edit, View, Insert, Cell, Kernel, Widgets, Help) and a toolbar. Below the toolbar, a row of numerical values is displayed: max, 0.861667, 0.840896, 0.972500, 0.507463, 3410.000000, 6946.000000, 8714.000000. The main code cell shows the following input and output:

```
In [87]: # unique values of categories variables
bike_data[cat_columns].nunique()

Out[87]: season      4
         yr          2
         mnth       12
         holiday      2
         weekday      7
         workingday    2
         weathersit     3
         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.

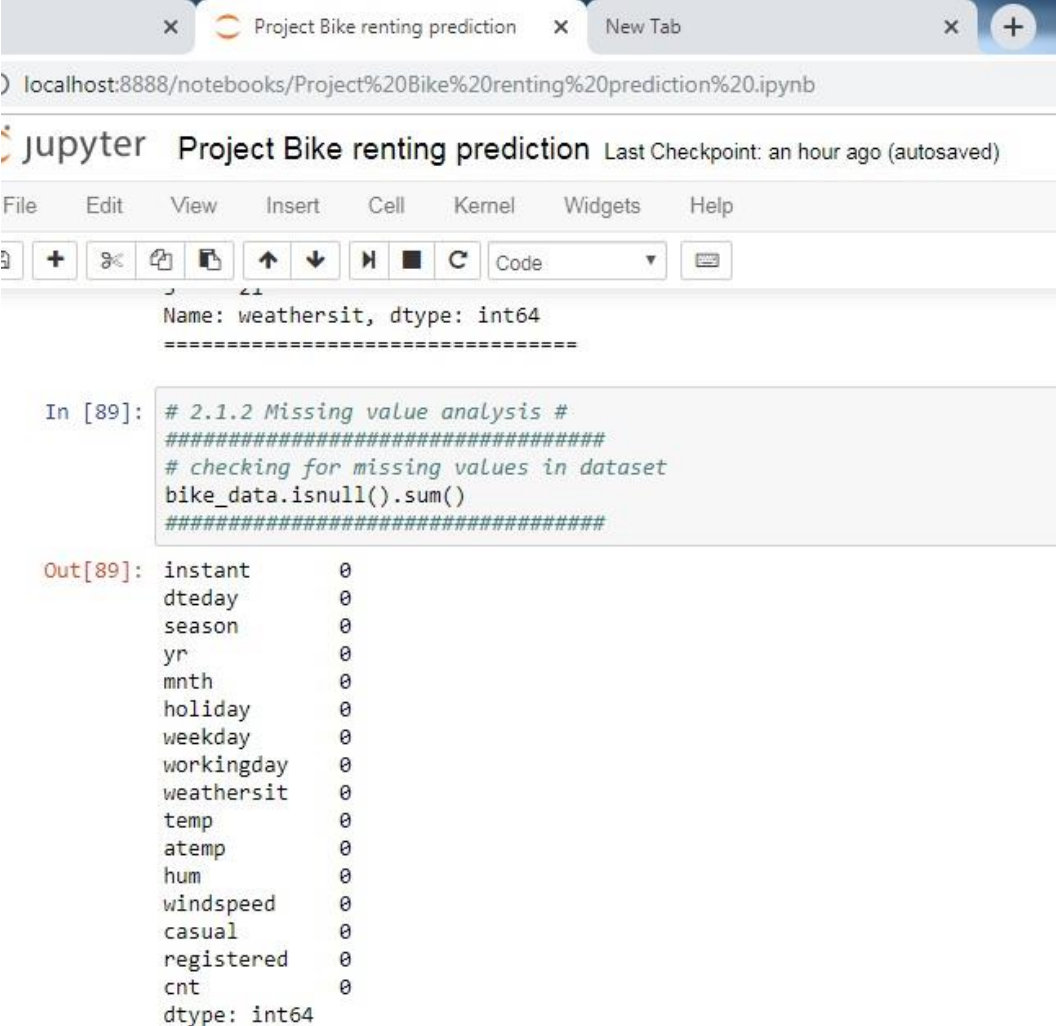
Weathersit has only three unique values with counting of value 3 (Light raining) is 21, counting of value 1 (clear weather) is 463 and counting of value 2 (cloudy weather) is 247.

Chapter 3 : Data Pre-processing

3.1 Missing value Analysis

In this step while checking, it is seen that there are no duplicate rows or empty values present throughout the dataset. Also the numerical variables are scaled by normalizing the data ,so only two variable we need to normalize and also no need of imputing missing values as no missing values present. Therefore, we will continue to next step.

Checking missing values for each column in our dataset:



```
Project Bike renting prediction
localhost:8888/notebooks/Project%20Bike%20renting%20prediction%20.ipynb
Jupyter Project Bike renting prediction Last Checkpoint: an hour ago (autosaved)
File Edit View Insert Cell Kernel Widgets Help
+ < > < > < > < > Code
Name: weathersit, dtype: int64
=====
In [89]: # 2.1.2 Missing value analysis #
#####
# checking for missing values in dataset
bike_data.isnull().sum()
#####
Out[89]: 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
```

Fig 3.1

Analysis:

We don't have any missing values in our dataset.

3.2 Outlier Analysis:

Outlier detection and treatment is always a tricky part especially when our dataset is small. The box plot method detects outlier if any value is present greater than $(Q3 + (1.5 * IQR))$ or less than $(Q1 - (1.5 * IQR))$

Q1 > 25% of data are less than or equal to this value

Q2 or Median -> 50% of data are less than or equal to this value

Q3 > 75% of data are less than or equal to this value

IQR(Inter Quartile Range) = $Q3 - Q1$

So, Boxplot method find approx. 1 % of data as outliers. It looks fine if we think only 1% of data we are treating as outlier and no impact would be after removal of outlier. Then we could be wrong.

Before treating outlier, we should look at nature of outlier. Is it information or outlier(error)?

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.

temp	0
atemp	0
hum	2
windspeed	13
casual	44
registered	0
cnt	0

Fig 3.2 :No. of outliers present

In fig: 3.2 we can clearly see the number of outliers present in the given dataset.

We look for outlier in the dataset by plotting Boxplots. There are outliers present in the data.

Let us see boxplot and histogram for our numerical data.

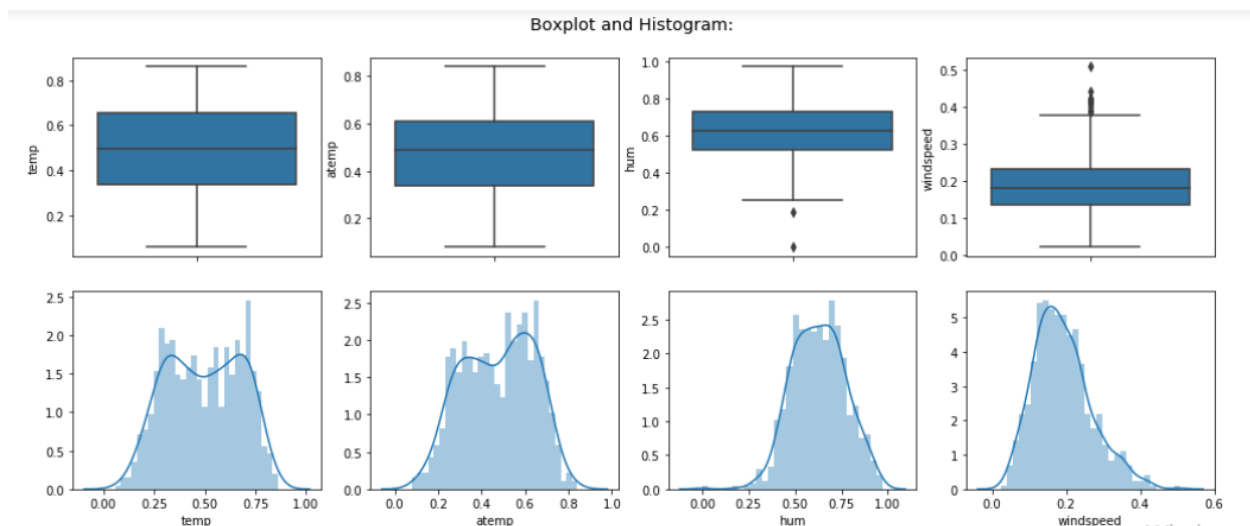


Fig 3.4

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.

To check the outliers present in the data visually

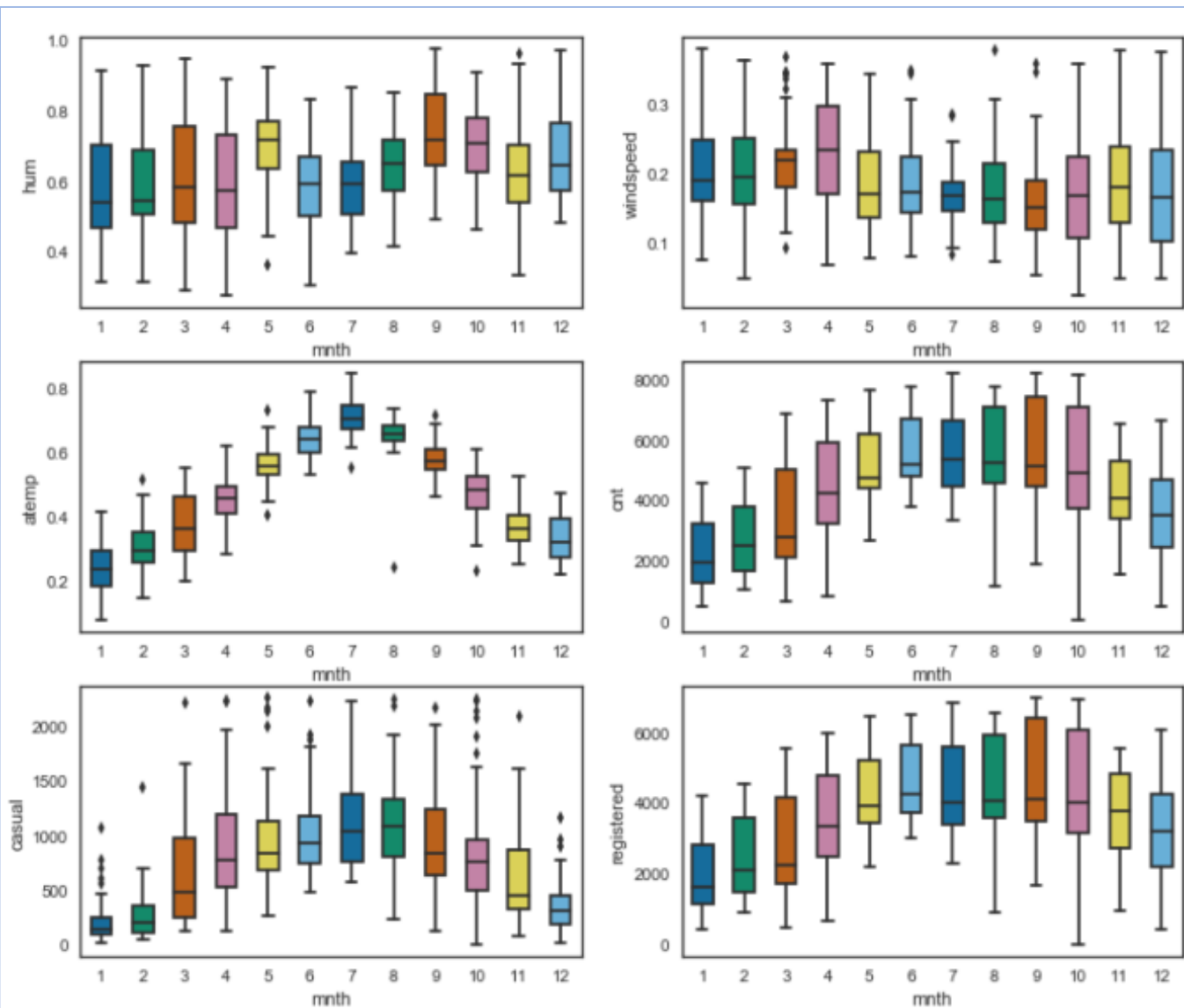


Fig 3.5

Box Plot of different numerical variables with month .

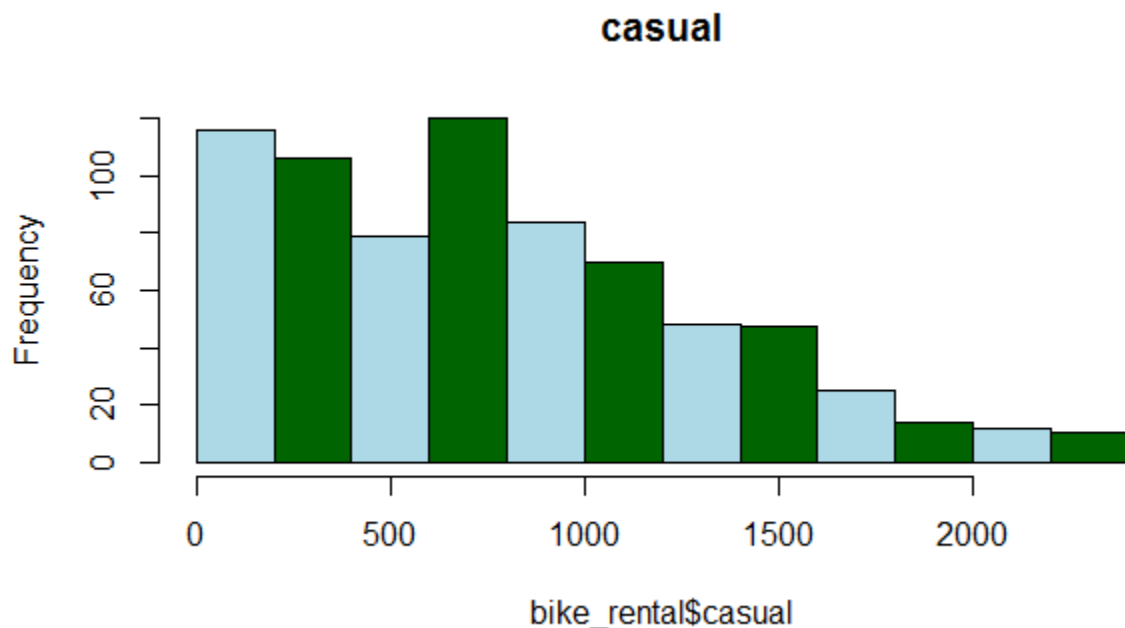
Fig 3.5 shows the number of outliers present in the data , now we need to remove these outliers and replace these outliers with NA and replace these missing values using KNN .

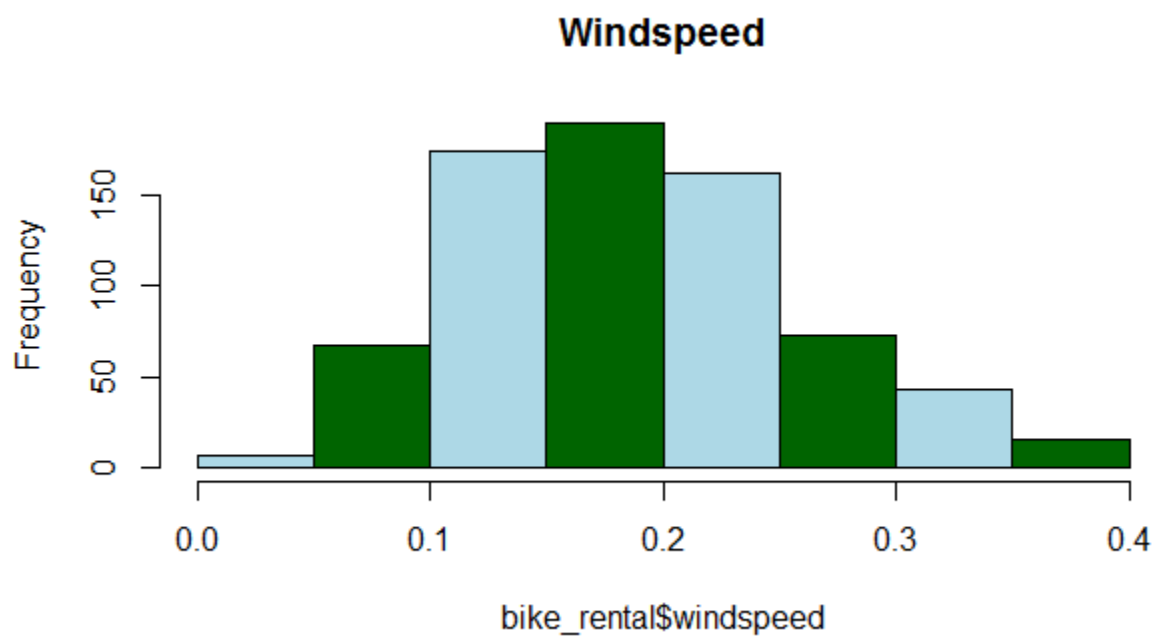
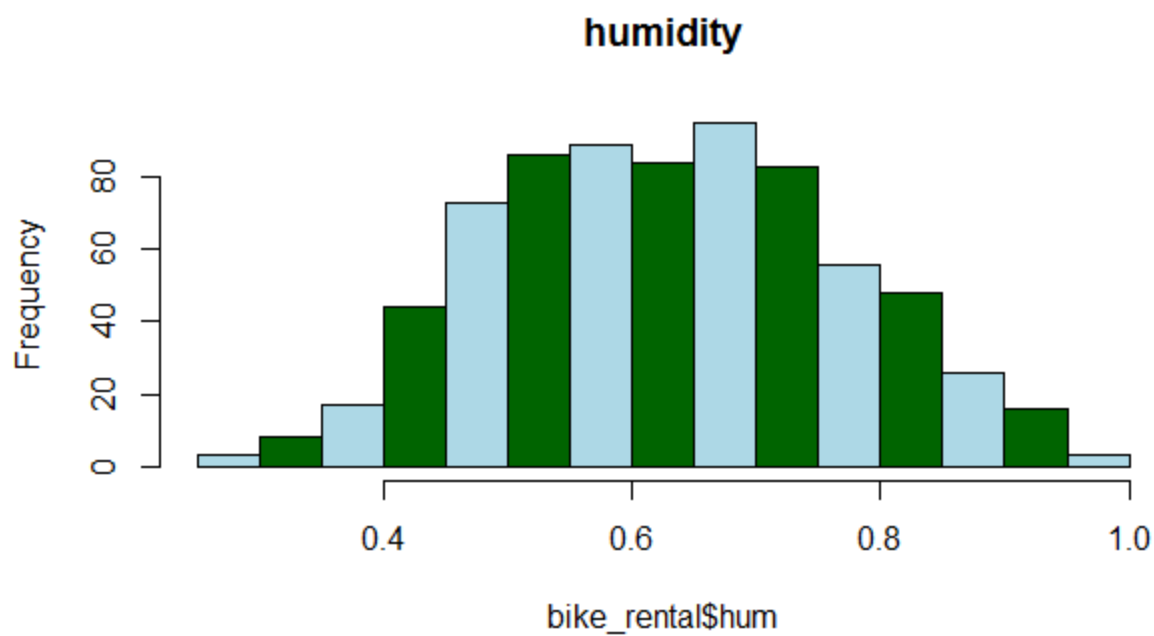
Now we have removed these outliers. This is how we done,

- I I. We replaced them with Nan values or we can say created missing values.
- II II. Then we imputed those missing values with KNN method.
- III III. We checked the performance of each method by checking Standard Deviation of that variable which has outliers before imputation and after imputation.

From above Boxplots we see that only 'hum' and 'windspeed' have outliers in them.

'hum' has 2 outliers and 'windspeed' has 13 outliers.





Figures above show the histogram of variables after removing outliers and impute these outliers with KNN.

3.3 Feature Selection

Feature Selection reduces the complexity of a model and makes it easier to interpret. Features are selected based on their scores in various statistical tests for their correlation with the outcome variable. Correlation plot is used to find out if there is any multicollinearity between variables. The highly collinear variables are dropped and then the model is executed.

In this step we would allow only to pass relevant features to further steps. We remove irrelevant features from the dataset. For selecting the features and understand the inter-relationship between the variables we have performed following tests

3.3.1 Correlation Analysis :

This requires only numerical variables. Therefore, we will filter out only numerical variables and feed it to correlation analysis. We do this by plotting correlation plot for all numerical variables. There should be no correlation between independent variables but there should be high correlation between independent variable and dependent variable. So, we plot the correlation plot. we can see that in correlation plot faded colour like skin colour indicates that 2 variables are highly correlated with each other. As the colour fades correlation values increases.

From below correlation plot we see that:

- 'temp' and 'atemp' are very highly correlated with each other.
- Similarly, 'registered' and 'cnt' are highly correlated with each other.
- We also came to know that--'cnt'='casual'+ 'registered' .

Fig 4.1 shows the correlation analysis table via **heatmap** where we can deduce that temp and atemp are highly correlated , also registered and our target variable are highly correlated .

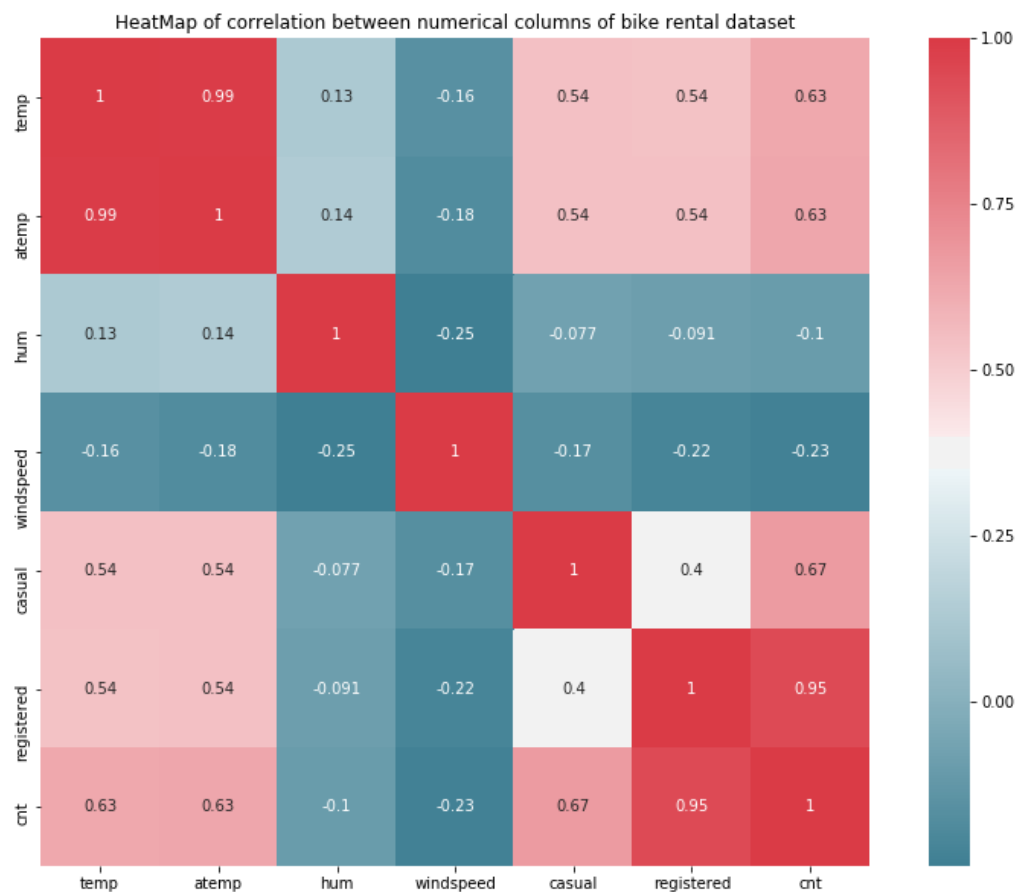


Fig 4.1

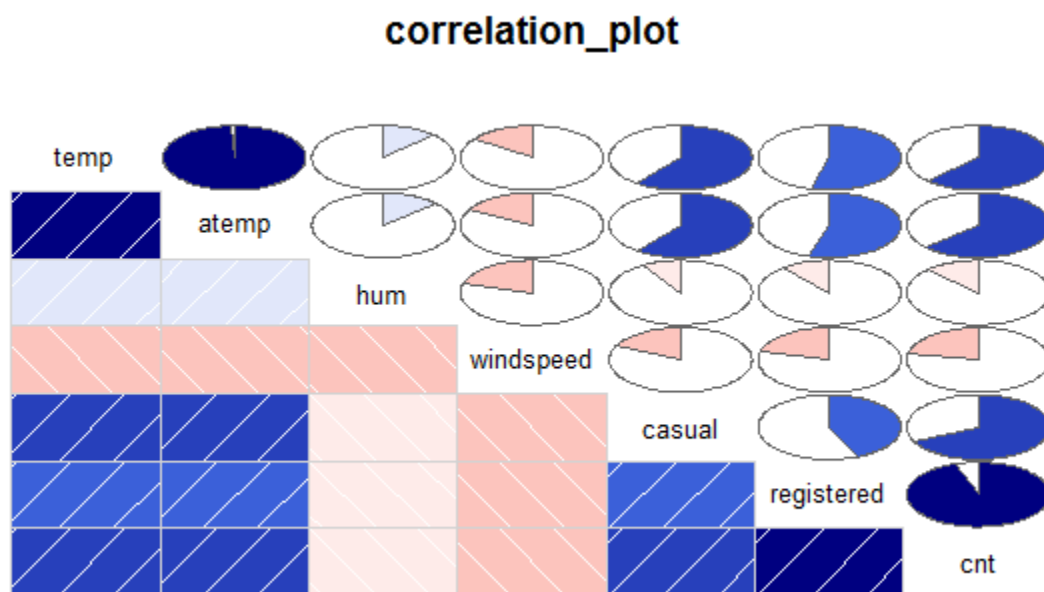
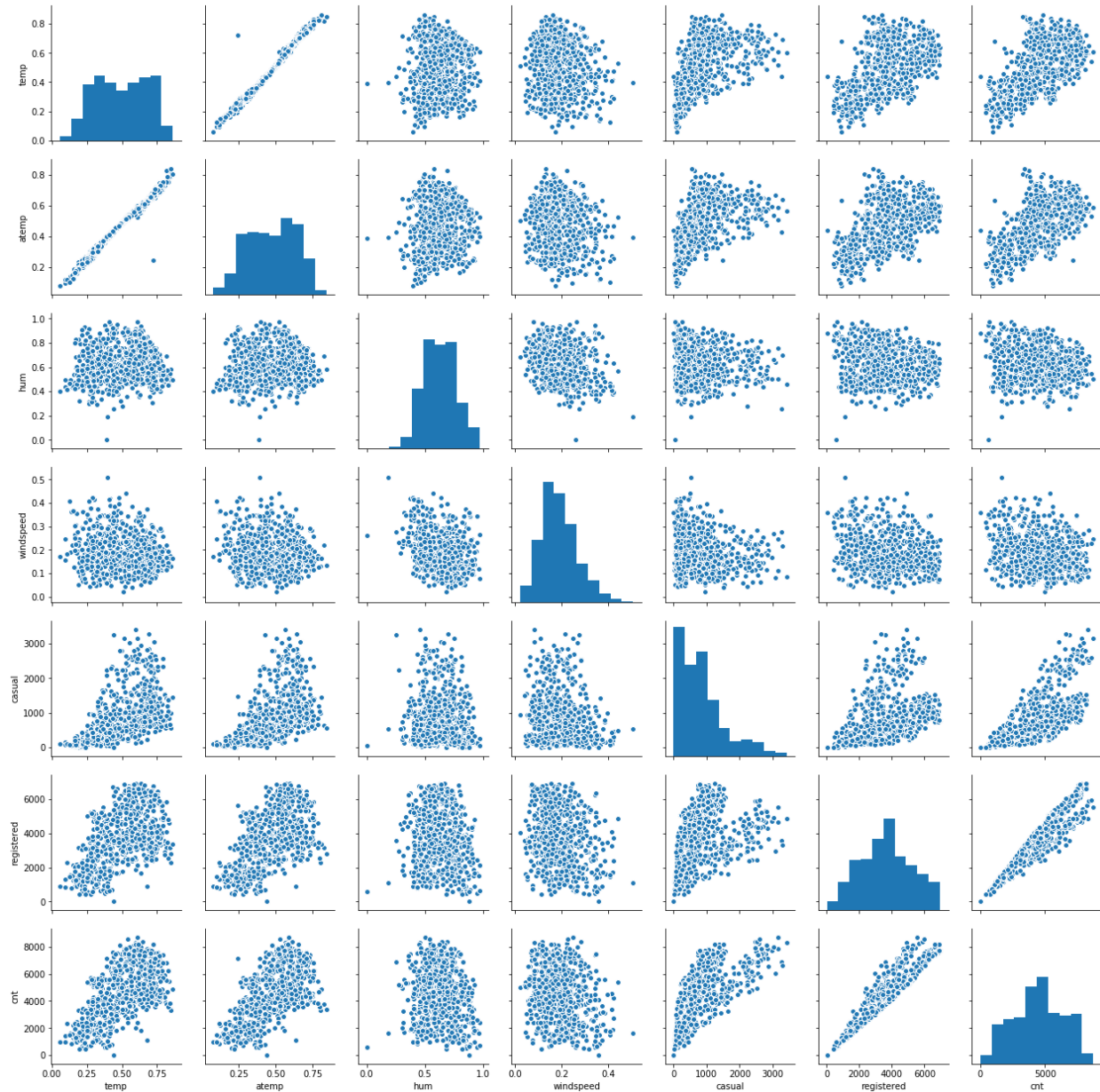


Fig 4.2 Correlation Plot

For multicollinearity between our continuous independent variable we can also look at scatter plot between our continuous variables. It also shows highly correlated variables.



Analysis:

From heatmap and correlation plot we can see that , there is multicollinearity present between temp and atemp, so we will 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.

3.3.2 Chi-Square test of independence – Unlike correlation analysis we will filter out only categorical variables and pass it to Chi-Square test. Chi-square test compares 2 categorical variables in a contingency table to see if they are related or not. Let us analyze for categorical variables also:

We would not use instant and dteday column. As instant is index only and information of dteday i.e. yr, month and day we have already columns for that. Dteday is important factor while doing time series analysis and we are not doing time series analysis.

For getting dependency between categorical variables we would use chi-sq test of independence test. Basically we have mnth , yr, weekday, holiday, workingday, weathersit and season as categorical variables. We will use week_feat (i.e. after making bins of week) and month_feat (i.e. after making bins of month).

Null Hypothesis for Chi-square test of Independence:

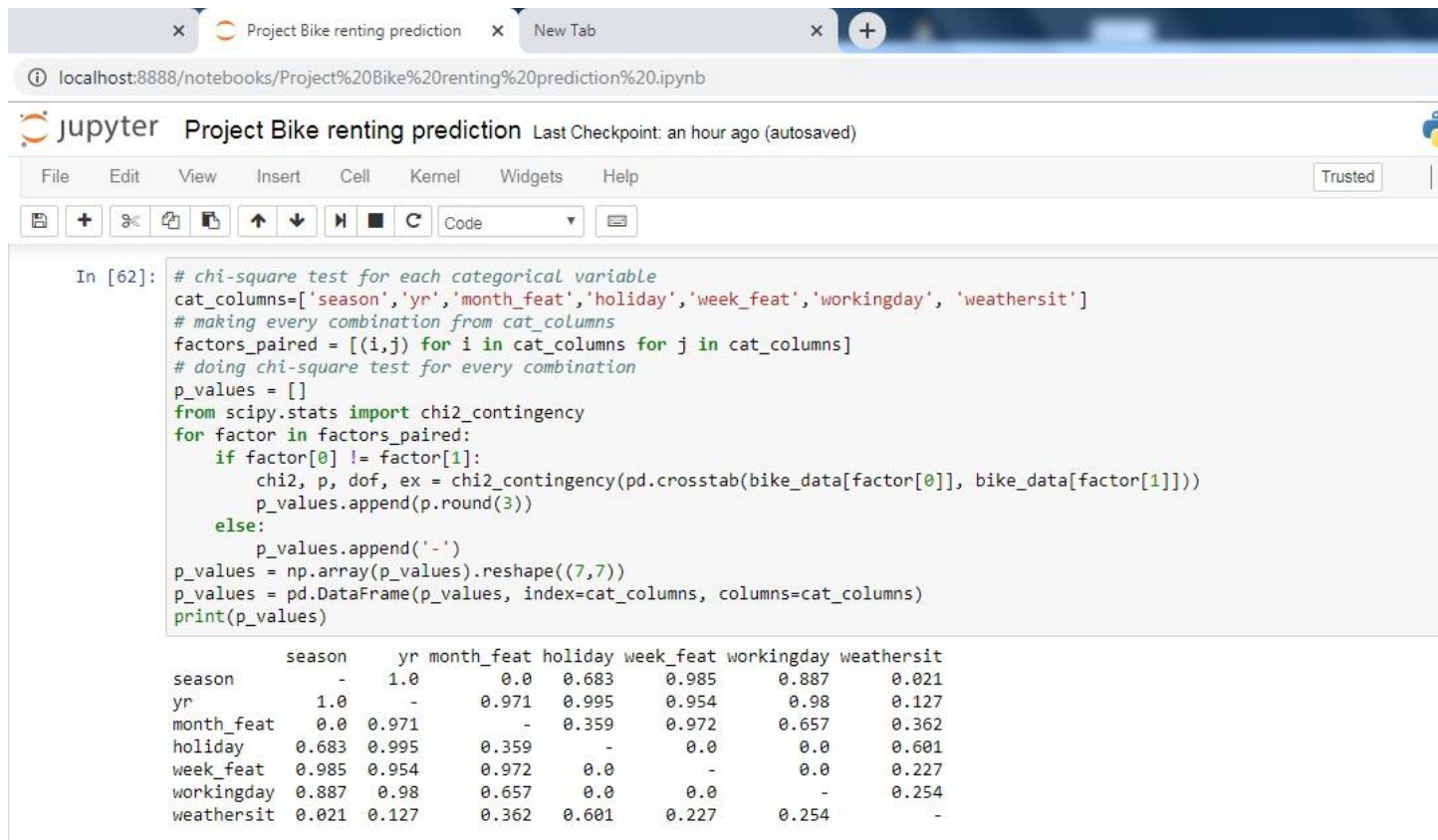
Two variables are independent.

Alternate Hypothesis:

Two variables are not independent.

So, we want that our categorical variable should be independent. If we get p-value less than 0.05, it means we need to reject null hypothesis and accept alternate hypothesis which means variables are dependent. So, we would check for every combination and would print p-value. Here we did the test between categorical independent variables pairwise.

- If $p\text{-value} < 0.05$ then remove the variable,
- If $p\text{-value} > 0.05$ then keep the variable



```
In [62]: # 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_data[factor[0]], bike_data[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)
```

	season	yr	month_feat	holiday	week_feat	workingday	weathersit
season	-	1.0	0.0	0.683	0.985	0.887	0.021
yr	1.0	-	0.971	0.995	0.954	0.98	0.127
month_feat	0.0	0.971	-	0.359	0.972	0.657	0.362
holiday	0.683	0.995	0.359	-	0.0	0.0	0.601
week_feat	0.985	0.954	0.972	0.0	-	0.0	0.227
workingday	0.887	0.98	0.657	0.0	0.0	-	0.254
weathersit	0.021	0.127	0.362	0.601	0.227	0.254	-

Analysis:

Here, from the table we can see some values are less than 0.05 indicating them they are dependent. Holiday is showing collinearity with week and workingday. Month is showing collinearity with season. Weathersit showing collinearity with season.

Now, we need to drop them to remove multicollinearity. But we have to be sure that we are not losing any information. So, we tried with dropping multicollinear column and building models and we got below result:

- On dropping weathersit or season we are losing accuracy with significant amount.
- On dropping holiday we are losing accuracy which is not significant amount.

- On dropping week_feat(weekday) or working day we are losing accuracy with little amount. And if we drop holiday, that information is also included in working day.

We just can't be sure by looking at test result and deciding, we need to get all factors. Here two columns most of the time showing collinearity but some datapoints would not be showing collinearity and at that datapoint there could be abrupt difference in bike renting count column which is very important information for our model.

We will drop only holiday column as we have working day column which has information of holiday also, that is why on dropping holiday we are not losing our accuracy. Also in while getting important features we are getting last ranking for holiday column with very less importance.

Let us check now important feature of our dataset:

jupyter Project Bike renting prediction Last Checkpoint: an hour ago (autosaved)

File Edit View Insert Cell Kernel Widgets Help

Code

```
In [65]: #checking importance of feature
drop_col = ['cnt', 'instant', 'dteday', 'registered', 'casual']
from sklearn.ensemble import ExtraTreesRegressor
reg = ExtraTreesRegressor(n_estimators=200)
X = bike_data.drop(drop_col, axis =1)
y = bike_data['cnt']
reg.fit(X, y)
imp_feat = pd.DataFrame({'Feature': bike_data.drop(drop_col, axis =1).columns, 'importance':reg.feature_importance_})
imp_feat.sort_values(by = 'importance', ascending=False).reset_index(drop = True)
```

Out[65]:

	Feature	importance
0	yr	0.315195
1	temp	0.154627
2	atemp	0.144376
3	month_feat	0.135817
4	season	0.104074
5	weathersit	0.058108
6	hum	0.034741
7	windspeed	0.022831
8	workingday	0.013795
9	week_feat	0.010506
10	holiday	0.005931

3.3.3 Multicollinearity— In regression, "multicollinearity" refers to predictors that are correlated with other predictors. Multicollinearity occurs when your model includes multiple factors that are correlated not just to your response variable, but also to each other.

I. Multicollinearity increases the standard errors of the coefficients.

II. Increased standard errors in turn means that coefficients for some independent variables may be found not to be significantly different from 0.

III. In other words, by overinflating the standard errors, multicollinearity makes some variables statistically insignificant when they should be significant. Without multicollinearity (and thus, with lower standard errors), those coefficients might be significant.

IV. VIF is always greater or equal to 1.

if VIF is 1 --- Not correlated to any of the variables.

if VIF is between 1-5 --- Moderately correlated.

if VIF is above 5 --- Highly correlated.

If there are multiple variables with VIF greater than 5, only remove the variable with the highest VIF.

V. And if the VIF goes above 10, you can assume that the regression coefficients are poorly estimated due to multicollinearity.

We have checked for multicollinearity in our Dataset and VIF values for temp and atemp are above 5.

We have performed VIF test using function `VIF` which is used to check whether variables have multicollinearity .

Checking VIF for our numerical variables:

```
In [66]: # checking vif of numerical column without 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_data[['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)

Out[66]: const      45.6
temp       63.0
atemp      63.6
hum        1.1
windspeed  1.1
dtype: float64
```

We would remove atemp variable as it is multicollinear with temp and having less importance than temp variable.

After removing atemp, let us check again VIF for numerical variables:

```
In [67]: # Checking VIF values of numeric columns after dropping column atemp
from statsmodels.stats.outliers_influence import variance_inflation_factor as vf
from statsmodels.tools.tools import add_constant
numeric_df = add_constant(bike_data[['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)

Out[67]: const      41.1
temp       1.0
hum        1.1
windspeed  1.1
dtype: float64
```

Now, we have good value of vif and don't have multicollinearity. Const is not part of our dataset it was just added as it is required for calculation of VIF.

3.3.4 Summary of Feature Selection :

Conclusions from the various test has been drawn that temp and atemp are highly correlated so we need to drop atleast one variable.

3.4 Feature Scaling:

We have numerical columns temp, hum and windspeed, which are provided in normalized form.

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

Chapter 4 : Data Modeling

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

- bike_data: dataset containing all columns except which we removed in EDA
- bike_data_wo : its same as bike_data but we removed outliers from it.
- X_train: containing all independent variables (part of bike_data), used for training model
- y_train : containing target variable (part of bike_data), used for training model
- X_test : containing independent variable (part of bike_data), used for testing model
- y_test: containing target variable(part of bike_data), , used for testing model
- X_train_wo: containing all independent variables (part of bike_data_wo), used for training model
- y_train_wo : containing target variable (part of bike_data_wo), used for training model
- X_test_wo : containing independent variable (part of bike_data_wo), used for testing model
- y_test_wo: containing target variable (part of bike_data_wo), , 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.

4.1 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 from sklearn)

K- fold cross validation :

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 mertric).

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.

4.2 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.

4.2.1 Linear Regression: Performance of Linear Regression built on X_train, y_train and tested on X_test. Showing K-fold cross validation explained_variance score and score for train and test dataset.

K-fold (K = 10) explained variance

=====

0.8159475214285375

on train data explained variance

=====

0.8246429104722082

on test data explained variance

=====

0.8399636835682225

Performance of Linear Regression built on X_train_wo, y_train_wo and tested on X_test_wo. Showing K-fold cross validation explained_variance score and score for train and test dataset.

K-fold (K = 10) explained variance

=====

0.8036114376234579

on train data explained variance

=====

0.812628161259224

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.

4.2.2 K Near Neighbors Regressor:

Performance of KNN Regressor built on X_train, y_train and tested on X_test. Showing K-fold cross validation explained_variance score and score for train and test dataset.

K-fold (K = 10) explained variance

=====

0.8045498992157334

on train data explained variance

=====

0.8752054851853227

on test data explained variance

=====

0.7948427483110001

Performance of KNN Regressor built on X_train_wo, y_train_wo and tested on X_test_wo. Showing K-fold cross validation explained_variance score and score for train and test dataset.

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 outliers.

4.2.3 Support Vector Regressor:

Performance of SVR(Gaussian kernel) built on X_train, y_train and tested on X_test. Showing K-fold cross validation explained_variance score and score for train and test dataset.

K-fold (K = 10) explained variance

=====

0.012405891628879196

on train data explained variance

=====

0.013320606469871543

on test data explained variance

=====

0.012475111232131852

Performance of SVR(Gaussian kernel) built on X_train_wo, y_train_wo and tested on X_test_wo. Showing K-fold cross validation explained_variance score and score for train and test dataset.

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 result for our dataset in any of case.

4.2.4 Decision Tree Regressor :

Performance of DTR built on X_train, y_train and tested on X_test. Showing K-fold cross validation explained_variance score and score for train and test dataset.

K-fold (K = 10) explained variance

=====

0.743886343148221

on train data explained variance

=====

1.0

on test data explained variance

=====

0.7952214097598804

Performance of DTR built on X_train_wo, y_train_wo and tested on X_test_wo. Showing K-fold cross validation explained_variance score and score for train and test dataset.

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. So, we will use next Random forest which would remove overfitting of decision Tree.

4.2.5 Random Forest Regressor:

Performance of Random Forest Regressor built on X_train, y_train and tested on X_test. Showing K-fold cross validation explained_variance score and score for train and test dataset.

K-fold (K = 10) explained variance

=====

0.8548225224328381

on train data explained variance

=====

0.9734605915767535

on test data explained variance

=====

0.8895839793569554

Performance of RF built on X_train_wo, y_train_wo and tested on X_test_wo. Showing K-fold cross validation explained_variance score and score for train and test dataset.

K-fold (K = 10) explained variance

=====

0.843399951802747

on train data explained variance

```
=====
0.9694980310035362
on test data explained variance
=====
0.9127408418315636
```

Analysis:

We can observe that Random Forest explained_variance score for k-fold is 85.48% for whole dataset and 84.33% 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 helped in removing overfitting.

4.2.6 XGB Regressor :

Performance of XGBRegressor built on X_train, y_train and tested on X_test. Showing K-fold cross validation explained_variance score and score for train and test dataset.

```
K-fold (K = 10) explained variance
=====
0.8815201352700848
on train data explained variance
=====
0.9438271842823257
on test data explained variance
=====
0.8819896466073337
```

Performance of XGBRegressor built on X_train_wo, y_train_wo and tested on X_test_wo. Showing K-fold cross validation explained_variance score and score for train and test dataset.

K-fold (K = 10) explained variance

```
=====
0.8601930769930155
on train data explained variance
=====
0.9400372084834097
on test data explained variance
=====
0.9208095049828924
```

Analysis:

We can observe that XGBRegressor explained_variance for K-fold is 88.15% for whole dataset and 86% for dataset without outliers. Also we have best result among all of above dataset.

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

4.3 Hyperparameter Tuning:

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

4.3.1 Random Forest Hyperparameter tuning

Now, Tuning Random Forest Model for following parameters:

```
from sklearn.ensemble import RandomForestRegressor
regressor = RandomForestRegressor(random_state=1)
params = [{'n_estimators': [400, 600, 800],
            'min_samples_split': [2, 4, 6], 'max_depth': [12, 14, 16], 'min_samples_leaf': [2, 3],
            '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)
grid_search.best_params_

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

Building model on tuned parameter suggested by GridsearchCV:

K-fold (K = 10) explained variance

=====

0.8668317334704365

on train data explained variance

=====

0.965607969664463

on test data explained variance

=====

0.8898201610699171

Analysis:

From above result (on tuned parameter), we have increased our model explained_variance score for k-fold from 85.48% 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.

So, with the help of hyperparameter tuning we reduced overfitting in our model.

4.3.2 XBG Regressor Hyperparameter Tuning:

Now, Tuning XGBRegressor Model for following parameters:

```
regressor = XGBRegressor(random_state=1)
params = [{'n_estimators': [300, 350, 400, 450, 600], '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)
grid_search.best_params_

{'gamma': 0,
 'learning_rate': 0.05,
 'max_depth': 3,
 'n_estimators': 300,
 'random_state': 1,
 'subsample': 0.8}
```

Building model on tuned parameter suggested by GridsearchCV:

K-fold (K = 10) explained variance

=====

0.8892740001843313

on train data explained variance

=====

0.9601878208920346

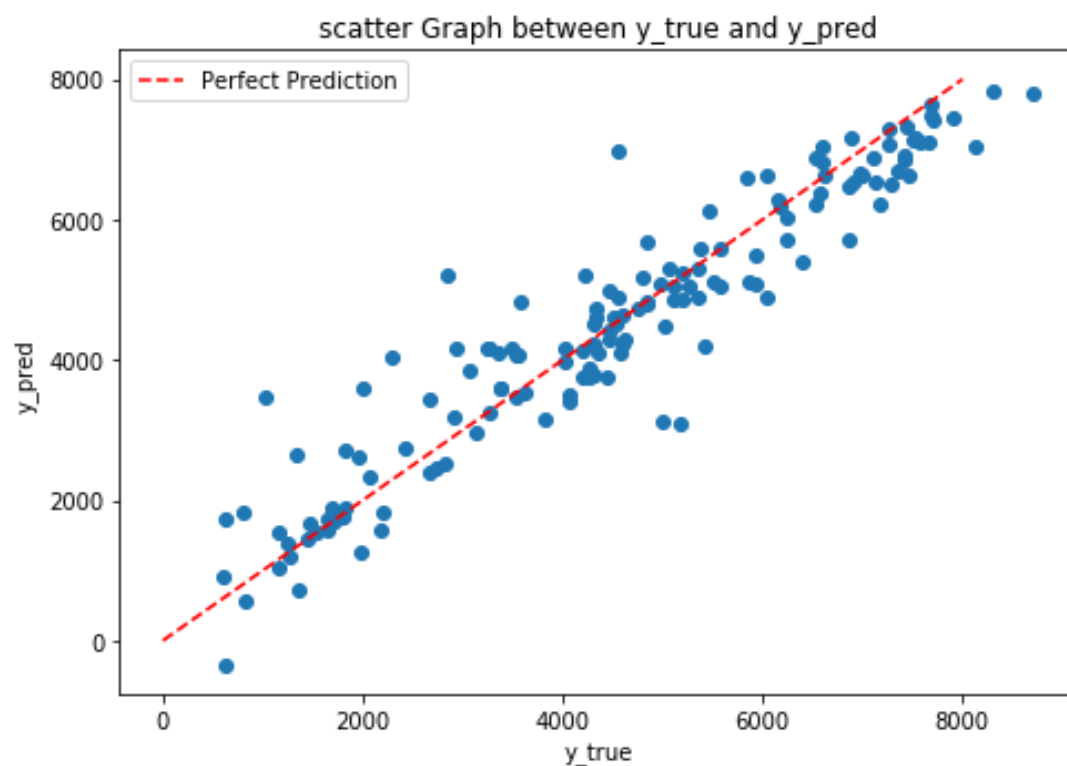
on test data explained variance

=====

0.8871646831415935

Analysis:

From above result (on tuned parameter), we have increased explained_variance score from 88.15% to 88.92%, which is less as XGBRegressor was giving us better result without tuning. But still we have increase somewhat with the help of hyperparameter tuning. Let us check a scatter graph between actual values of test and predicted values. A line at 45 degree , i.e. slope =1 will be perfect prediction and points showing above and below of that line would be error in predictions.



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 5 : Conclusion

5.1 Final Dataset and Final Model:

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 explained in above step.
- Do hyperparameter tuning for training set.
- Build XGBRegressor model with tuned parameters.

5.2 Final Notes

- All analysis result is based on python. In R, result would not be exact same but would be almost same.
- Outlier should be treated well by gaining domain knowledge and experimenting with model building and checking performance.
- Steps done in EDA section is just not based on statistical test, We have done many experiment like dropping 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.

Appendix A – Complete Python Code

- <https://github.com/raviec10/Bike-Renting-Prediction/blob/master/BIKE+RENTING+PROJECT.py>

Appendix B – Complete R code Link

- <https://github.com/raviec10/Bike-Renting-Prediction/blob/master/BIKE%20RENTING%20PROJECT.R>

References

- <https://www.analyticsvidhya.com/blog/2015/06/solution-kaggle-competition-bike-sharing-demand/>
- <https://github.com/>
- <https://www.kaggle.com/>
- <https://rajivsworldlife.wordpress.com/2018/11/24/bike-rentals-a-data-science-and-machine-learning-project-using-a-regression-algorithm/>