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.

We are provided with dataset of bike rental based on environmental conditions and seasons for two year i.e. 2010 and 2011. We need to predict what could be the counting of bike rental for a given season, month and year with environmental condition.

1.2 Data

Our model will be regression analysis, which will predict total bike rental for a condition. Let us see few data points of our dataset.

instant	dteday	season	yr	mnth	holiday	weekday	workingday	weathersit
1	1/1/2011	1	0	1	0	6	0	2
2	1/2/2011	1	0	1	0	0	0	2
3	1/3/2011	1	0	1	0	1	1	1
4	1/4/2011	1	0	1	0	2	1	1
5	1/5/2011	1	0	1	0	3	1	1
6	1/6/2011	1	0	1	0	4	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
0.2	0.212122	0.590435	0.160296	108	1454	1562
0.226957	0.22927	0.436957	0.1869	82	1518	1600
0.204348	0.233209	0.518261	0.0895652	88	1518	1606

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.

All columns have name as they normally stands for. Details are below:

dteday: Date
yr: Year (0: 2011, 1:2012)
mnth: Month (1 to 12)
weekday: Day of the week
season: Season (1:springer, 2:summer, 3:fall, 4:winter)
holiday: weather day is holiday or not (extracted fromHoliday Schedule)
workingday: If day is neither weekend nor holiday is 1, otherwise is 0.
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
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)
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)
hum: Normalized humidity. The values are divided to 100 (max)
windspeed: Normalized wind speed. The values are divided to 67 (max)
casual: count of casual users
registered: count of registered users
cnt: count of total rental bikes including both casual and registered

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.

So, we would analyze these columns one by one in next section.

Chapter 2

Methodology

2.1 Data Preprocessing: (Exploratory Data Analysis)

In Data Analytics, there is a famous quote by Riley Newman:

> ***"More data beats better models. Better data beats more data."***
>
> *- Riley Newman*

If we have our best model and we feed our data to that model, then it is not guaranteed that model will perform its best. As our data may have lots of noisy data (Garbage) and model will also follow noisy data and thus can produce wrong result because of that noisy data. We cannot remove noise/error completely from our data but we can reduce it with the help of EDA (Exploratory Data Analysis).

EDA involves getting summary of data with numerical statistics and Graphical visualization.

2.1.1 Understanding the Data:

First, 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: 91.5+ KB
```

** Python Code to generate above result

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 5^{th} 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. 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:

	temp	atemp	hum	windspeed	casual	registered	cnt
count	731	731	731	731	731	731	731
mean	0.49539	0.47435	0.62789	0.190486	848.17647	3656.17237	4504.34884
std	0.18305	0.16296	0.14243	0.077498	686.62249	1560.25638	1937.21145
min	0.05913	0.07907	0	0.022392	2	20	22
25%	0.33708	0.33784	0.52	0.13495	315.5	2497	3152
50%	0.49833	0.48673	0.62667	0.180975	713	3662	4548
75%	0.65542	0.6086	0.73021	0.233214	1096	4776.5	5956
max	0.86167	0.8409	0.9725	0.507463	3410	6946	8714

** Python Code to generate above result

Analysis:

We have four independent continuous variables (in dark blue shades) and three dependent continuous variables (in dark orange shades).

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:

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

** Python Code to generate above result

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
```

** Python Code to generate above result

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:

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, linear regression, MICE, missForest etc.

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
```

** Python Code to generate above result

Analysis:

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

2.1.3 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.

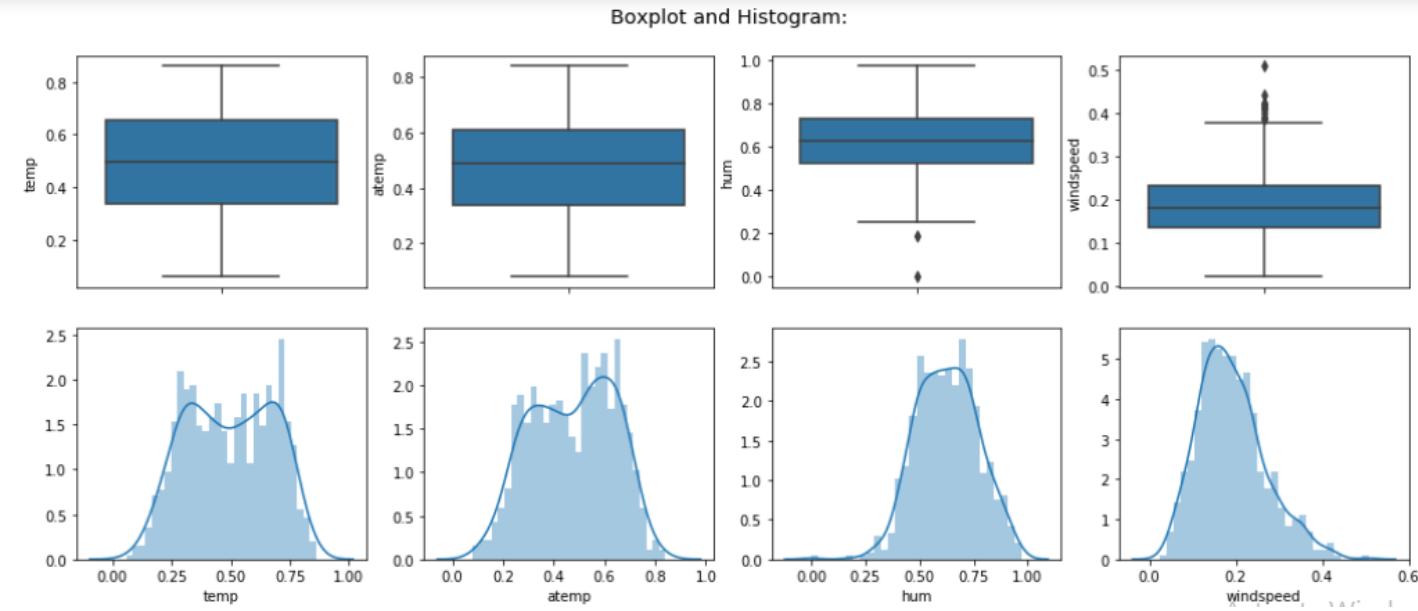
Before treating outlier, we should look at nature of outlier. Is it information or outlier(error)?

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.

Let us see boxplot and histogram for our numerical data.



** Python Code to generate above result

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.

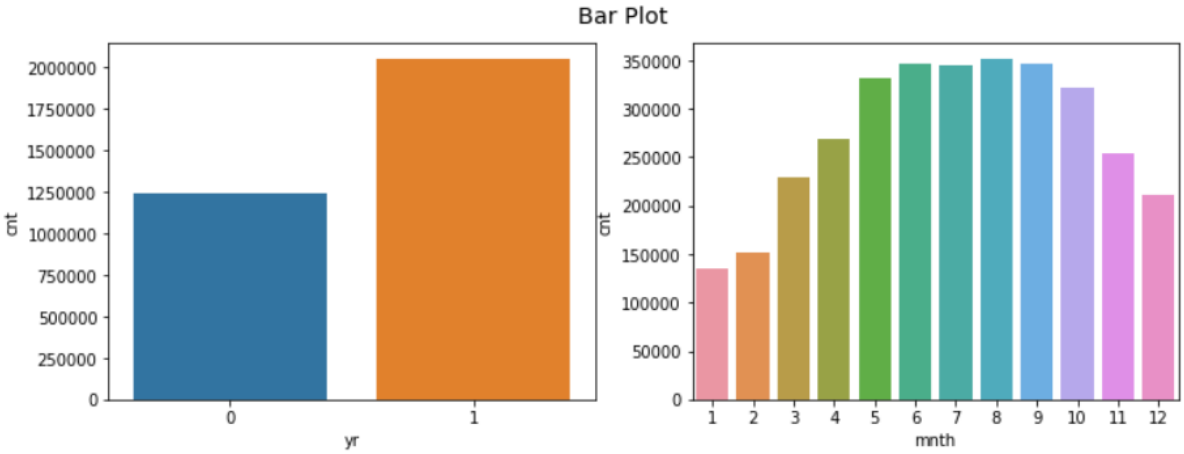
2.1.4 Feature Engineering:

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

For mnth:

Now, if we choose this column as numerical then it would be like 3 is greater than 2 and 2 is greater than 1, which is not the case. Also choosing every mnth as an category, then we may have curse of dimensionality and our model will underperform. So, we will make a new column for 'mnth' as per current values.

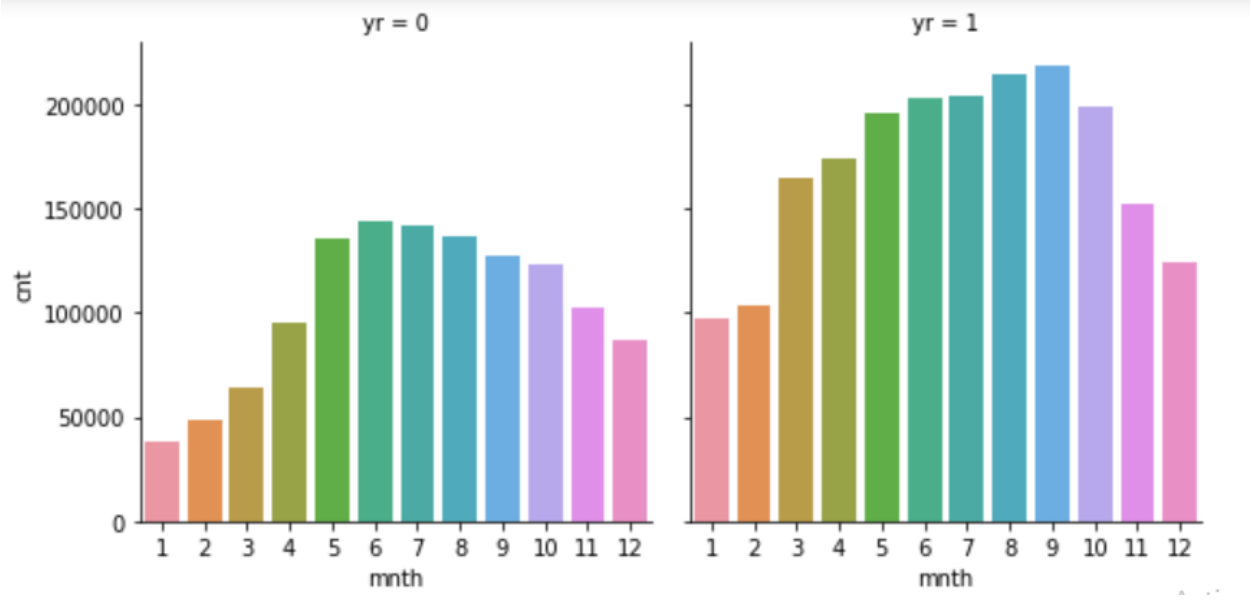
Let us see distribution of counting of bike renting according to month as well as with year



** Python Code to generate above result

We can see a trend in counting of bike renting for month column. 5th to 10th month has high renting and other have less in comparison.

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



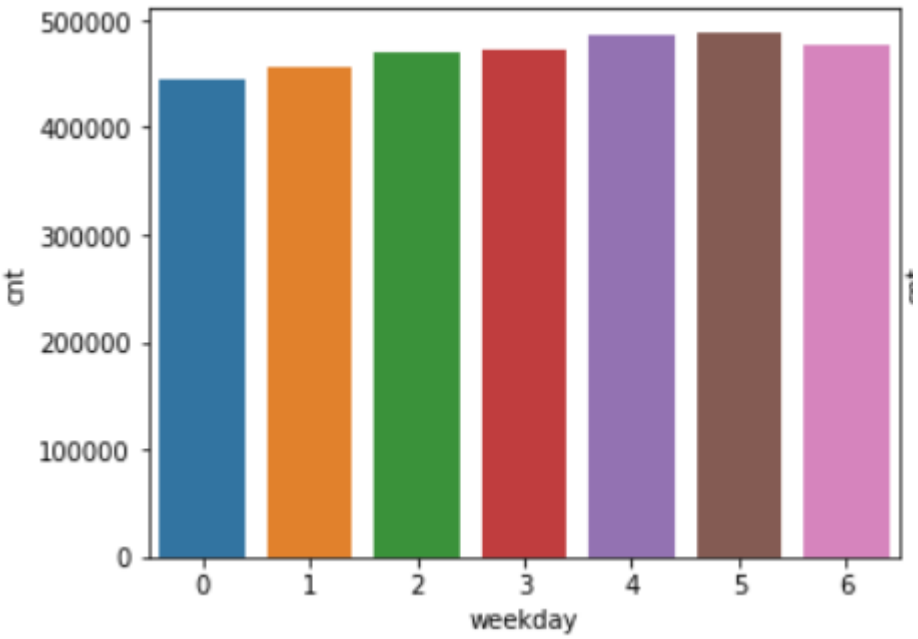
** Python Code to generate above result

From above distribution we can observe that for each year we have similar trend. We will categories month in two categories i.e. '1' for month 5th to 10th and '0' for rest. As 5th to 10th having high values for both 2011 and 2012 and rest have less.

Another point we can think that 5th month of 2011 would not be same 5th month of 2012. Here we have another category of yr which has value 0 for 2011 and 1 for 2012. So our model will learn month value with 1 showing high trend and year with value 1 having high trend, so effect of 5th month of 2011 and 5th month of 2012 would be different on model.

Let us do similar **analysis for weekday**:

Checking bar plot of counting for each weekday:



** Python Code to generate above result

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

Note:

We Just not decided to make binning for month and weekday. For time series data, it is advised that time like year, month, day could not be categorical. So, first we built our model by taking month and day of week as a continuous values and then tried with binning of month and weekday. After that we compared our both model. So, on making bins for 'mnth' we got significant improvement of 1% (explained_variance_score applied on 10 fold) and upon making bins for weekday we got improvement of less than 0.05%.

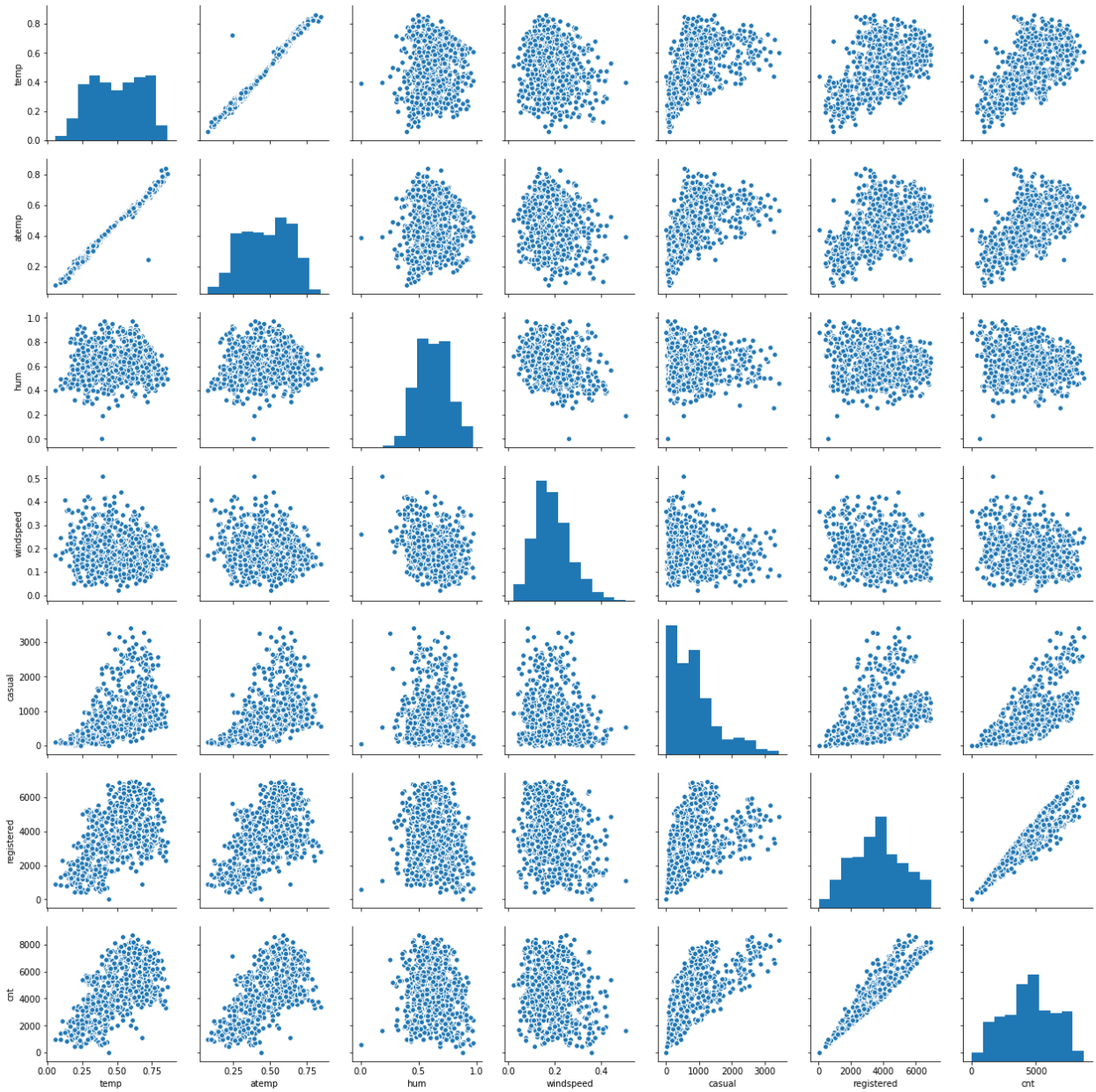
We can see binning of month giving us very good improvement as we can see from bar plot also it has significant difference in both category(made after binning) for month and weekday has very less difference that is why giving us less improvement.

We will make dummy variables(with dropping one column) 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:

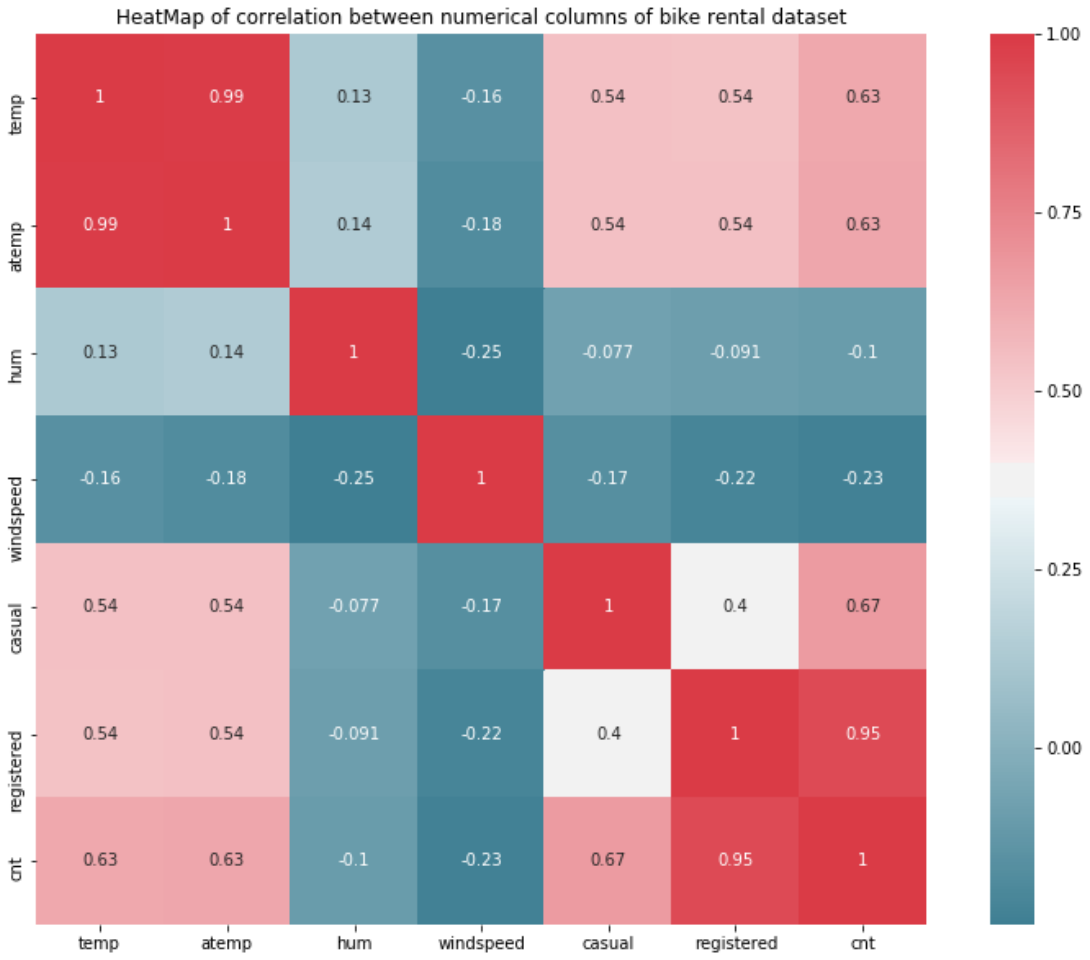
For feature selection of numerical variable we will see if there is multicollinearity between our continuous independent variable.

For this we will plot scatter plot between our continuous variables.



** Python Code to generate above result

Let us see heatmap for the same:



** Python Code to generate above result

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.

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.

	season	yr	month_feat	holiday	week_feat	workingday	weathersit
season	-	1	0	0.683	0.985	0.887	0.021
yr	1	-	0.971	0.995	0.954	0.98	0.127
month_feat	0	0.971	-	0.359	0.972	0.657	0.362
holiday	0.683	0.995	0.359	-	0	0	0.601
week_feat	0.985	0.954	0.972	0	-	0	0.227
workingday	0.887	0.98	0.657	0	0	-	0.254
weathersit	0.021	0.127	0.362	0.601	0.227	0.254	-

** Python Code to generate above result

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:

	Feature	importance
0	yr	0.315736
1	temp	0.165388
2	atemp	0.140998
3	month_feat	0.124285
4	season	0.106907
5	weathersit	0.059081
6	hum	0.033578
7	windspeed	0.023560
8	workingday	0.013887
9	week_feat	0.010608
10	holiday	0.005972

** Python Code to generate above result

Checking VIF for our numerical variables:

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

** Python Code to generate above result

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:

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

** Python Code to generate above result

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.

2.1.6 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.

2.1.7 Data After EDA

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

```
bike_data.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 Code to generate above result

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

```
bike_data_wo.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 Code to generate above result

2.2 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 trianinig model
- X_test : containing independent variabel (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 trianinig model
- X_test_wo : containing independent variabel (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.

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 from sklearn)

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 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.

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:

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
```

** Python Code to generate above result

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
```

** Python Code to generate above result

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 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
```

** Python Code to generate above result

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
```

** Python Code to generate above result

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.

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
```

** Python Code to generate above result

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
```

** Python Code to generate above result

Analysis:

Support vector regressor with Gaussian kernel is not giving good result for our dataset in any of case.

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
```

** Python Code to generate above result

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
```

** Python Code to generate above result

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.

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
```

** Python Code to generate above result

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
```

** Python Code to generate above result

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.

Next model we would use XGBRegressor.

XGBRegressor:

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
```

** Python Code to generate above result

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
```

** Python Code to generate above result

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.

2.2.3 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.

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
```

** Python Code to generate above result

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.

XGBRegressor 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}
```

** Python Code to generate above result

Building model on tuned parameter suggested by GridsearchCV:

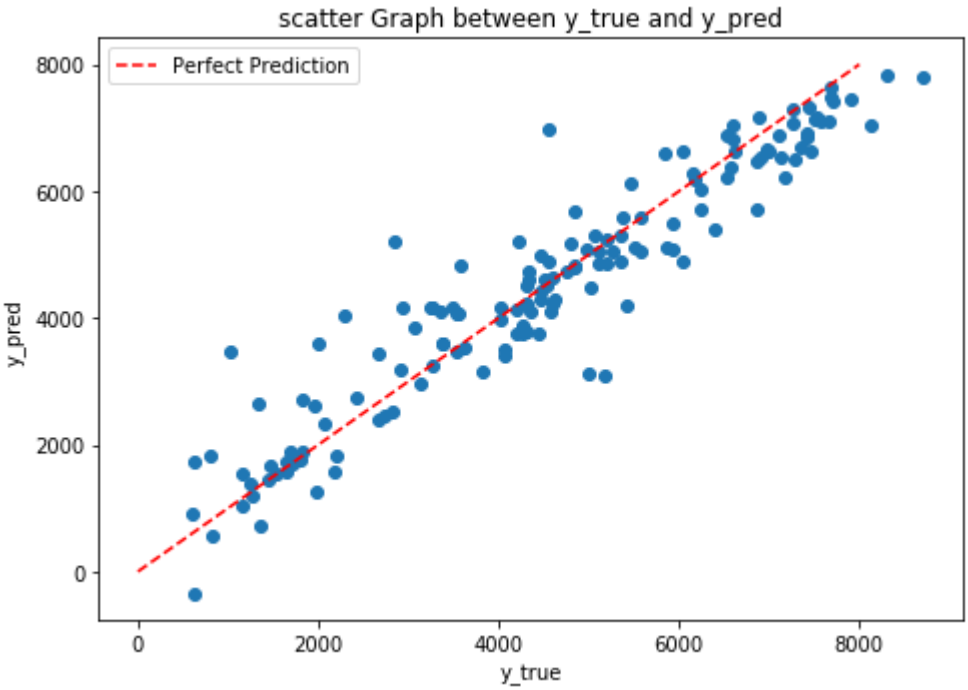
```
K-fold (K = 10) explained variance
==================================
0.8892740001843313

on train data explained variance
==================================
0.9601878208920346

on test data explained variance
==================================
0.8871646831415935
```

** Python Code to generate above result

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.



** Python Code to generate above result

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

3.2 End Notes

- All analysis result is based on python. In R, result would not be exact same but would be almost same.
- We have not done here 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 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.

Complete Python code

```
# Project for prediction of bike renting based on environmental condition
# importing libraries
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# reading csv file
bike_data = pd.read_csv("day.csv")

##############################################
#                                            #
#     2.1 Exploratory Data Analysis          #
#                                            #
##############################################

###################################
#  2.1.1 understanding the data   #
###################################
#Checking few observation of dataset
bike_data.head()

# looking at information of dataset  -> see output
bike_data.info()

numeric_columns = ['temp','atemp','hum','windspeed','casual','registered','cnt']
cat_columns = ['season','yr','mnth','holiday','weekday','workingday','weathersit']

# looking at five point summary for our numerical variables  -> see output
bike_data[numeric_columns].describe()

# unique values of categories variables  -> see output
bike_data[cat_columns].nunique()

# counting of each unique values in each categorical variable  -> see output
print("value counts of categorical column")
print()
for i in cat_columns:
    print(i)
    print(bike_data[i].value_counts())
    print("================================")

###################################
#  2.1.2 Missing value analysis   #     -> see output
###################################
# checking for missing values in dataset
bike_data.isnull().sum()
```

```
###################################
#  2.1.3 outlier analysis         #     -> see output
###################################


# user defined function that will plot boxplot and histogram for four columns
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_data)



###################################
#  2.1.4 Feature Engineering      #
###################################      -> see output
# 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
# willbarplot 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_data.groupby(by = x1).mean()
        gp2 = bike_data.groupby(by = x2).mean()
    else:
        gp = bike_data.groupby(by = x1).sum()
        gp2 = bike_data.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  -> see output
gp = bike_data.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  -> see output
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_data['month_feat'] = bike_data.apply(lambda row : feat_month(row), axis=1)
bike_data = bike_data.drop(columns=['mnth'])
bike_data['week_feat'] = bike_data.apply(lambda row : feat_weekday(row), axis=1)
bike_data = bike_data.drop(columns=['weekday'])

####################################
#  2.1.5 Feature Selection        #
####################################

# let us look at correlation plot for each numerical variables  -> see output
sns.pairplot(bike_data[numeric_columns])

# let us look at heat map for each numerical variable  -> see output
# with correlation
fig = plt.figure(figsize = (14,10))
corr = bike_data[numeric_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("HeatMap of corr. between numerical columns of bike rental dataset")
fg = sn_plt.get_figure()
fg.savefig('heatmap.png')
# chi-square test for each categorical variable  -> see output
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)

# checking importance of feature  -> see output
drop_col = ['cnt', 'instant','dteday', 'registered', 'casual']
from sklearn.ensemble import ExtraTreesRegressor
reg = ExtraTreesRegressor(n_estimators=200)
X = bike_data.drop(columns= drop_col)
y = bike_data['cnt']
reg.fit(X, y)
imp_feat = pd.DataFrame({'Feature': bike_data.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  -> see output
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)

# Checking VIF values of numeric columns after dropping column atemp  -> see output
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)


# Making dummies for each category session and weather
season_dm = pd.get_dummies(bike_data['season'],drop_first=True, prefix='season')
bike_data = pd.concat([bike_data, season_dm],axis=1)
bike_data = bike_data.drop(columns = ['season'])
weather_dm = pd.get_dummies(bike_data['weathersit'], prefix= 'weather',
                            drop_first=True)
```

```
bike_data = pd.concat([bike_data, weather_dm],axis=1)
bike_data = bike_data.drop(columns= ['weathersit'])


###################################
#  2.1.7 Data after EDA           #  -> see output
###################################

# creating another dataset with dropping outliers i.e. bike_data_wo
bike_data_wo = bike_data.copy()
# dropping outliers from boxplot method
for i in ['windspeed', 'hum']:
    q75, q25 = np.percentile(bike_data_wo.loc[:,i], [75 ,25])
    iqr = q75 - q25
    min = q25 - (iqr*1.5)
    max = q75 + (iqr*1.5)
    bike_data_wo=bike_data_wo.drop(bike_data_wo[bike_data_wo.loc[:,i]<min].index)
    bike_data_wo=bike_data_wo.drop(bike_data_wo[bike_data_wo.loc[:,i]>max].index)

# dropping unwanted columns from both dataset bike_data and bike_data_wo
bike_data.drop(columns=['instant', 'dteday', 'holiday', 'atemp', 'casual',
                        'registered'], inplace=True)
bike_data_wo.drop(columns=['instant', 'dteday', 'holiday', 'atemp', 'casual',
                           'registered'], inplace=True)
  -> see output
bike_data.head()
bike_data_wo.head()

print('shape of original dataset',bike_data.shape)
print('shape of dataset after removing outliers', bike_data_wo.shape)

###############################################
#                                             #
#                                             #
#   2.2.2 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 & test
from sklearn.metrics import explained_variance_score
from sklearn.model_selection import cross_val_score
def fit_predict_show_performance(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_data
X = bike_data.drop(columns=['cnt'])
y = bike_data['cnt']
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2,
                                                    random_state = 0)

# splitting dataset in train and test for dataset without outlier i.e. bike_data_wo
X = bike_data_wo.drop(columns=['cnt'])
y = bike_data_wo['cnt']
from sklearn.model_selection import train_test_split
X_train_wo, X_test_wo, y_train_wo, y_test_wo = train_test_split(X,y,test_size=0.2,
                                                                random_state = 0)

#########################
#   Linear Regression   #
#########################

# building model for dataset bike_data  -> see output
from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
fit_predict_show_performance(regressor, X_train, y_train, X_test, y_test)

# building model for dataset bike_data_wo i.e. without  outliers  -> see output
regressor = LinearRegression()
fit_predict_show_performance(regressor, X_train_wo, y_train_wo, X_test_wo,y_test_wo)

#########################
#         KNN           #
#########################

# building model for dataset bike_data  -> see output
from sklearn.neighbors import KNeighborsRegressor
regressor = KNeighborsRegressor(n_neighbors=5)
fit_predict_show_performance(regressor, X_train, y_train, X_test, y_test)

# building model for dataset bike_data_wo i.e. without outliers  -> see output
regressor = KNeighborsRegressor(n_neighbors=5)
fit_predict_show_performance(regressor,X_train_wo,y_train_wo,X_test_wo,y_test_wo)
```

```
#########################
#        SVM            #
#########################

# building model for dataset bike_data  -> see output
from sklearn.svm import SVR
regressor = SVR()
fit_predict_show_performance(regressor, X_train, y_train, X_test, y_test)

# building model for dataset bike_data_wo i.e. without outliers  -> see output
regressor = SVR()
fit_predict_show_performance(regressor,X_train_wo,y_train_wo,X_test_wo,y_test_wo)

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

# building model for dataset bike_data  -> see output
from sklearn.tree import DecisionTreeRegressor
regressor = DecisionTreeRegressor(random_state=1)
fit_predict_show_performance(regressor, X_train, y_train, X_test, y_test)

# building model for dataset bike_data_wo i.e. without outliers  -> see output
fit_predict_show_performance(regressor,X_train_wo,y_train_wo,X_test_wo,y_test_wo)

#########################
#  Random Forest        #
#########################

# building model for dataset bike_data  -> see output
from sklearn.ensemble import RandomForestRegressor
regressor = RandomForestRegressor(random_state=1)
fit_predict_show_performance(regressor, X_train, y_train, X_test, y_test)

# building model for dataset bike_data_wo i.e. without outliers  -> see output
regressor = RandomForestRegressor(random_state=1)
fit_predict_show_performance(regressor,X_train_wo,y_train_wo,X_test_wo,y_test_wo)

#########################
#     XGBRegressor      #
#########################

# building model for dataset bike_data  -> see output
from xgboost import XGBRegressor
regressor = XGBRegressor(random_state=1)
fit_predict_show_performance(regressor, X_train, y_train, X_test, y_test)
# building model for dataset bike_data_wo i.e. without outliers  -> see output
regressor = XGBRegressor(random_state=1)
fit_predict_show_performance(regressor,X_train_wo,y_train_wo,X_test_wo,y_test_wo)
```

```
##############################################
#                                            #
#                                            #
#        Hyperparameter tuning               #
#                                            #
#                                            #
##############################################
################################################
#                                              #
# tuning Random Forest for bike_data dataset   #
#                                              #
################################################

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_)

# building Ranodm Forest on tuned parameter  -> see output
regressor = RandomForestRegressor(random_state=1, max_depth=14, n_estimators=600,
                                  max_features='auto', min_samples_leaf=2,
                                  min_samples_split=2)
fit_predict_show_performance(regressor, X_train, y_train, X_test, y_test)

################################################
#                                              #
# tuning XGBRegressor for bike_data 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  -> see output
regressor = XGBRegressor(random_state=1, learning_rate=0.05, max_depth=3,
                         n_estimators=300,gamma = 0, subsample=0.8)
fit_predict_show_performance(regressor, X_train, y_train, X_test, y_test)
```

```
# plotting scatter graph for y_true and y_pred for tuned XGBRegressor model  -> see output
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 between y_true and y_pred")
plt.xlabel("y_true")
plt.ylabel("y_pred")
plt.tight_layout()
plt.show()
```

Complete R code Link

https://drive.google.com/open?id=1BVANf0fdCpmx00UZOiUUfhCxuXR2Zsvw

References:

http://scikit-learn.org/stable/modules/model_evaluation.html

http://topepo.github.io/caret/index.html