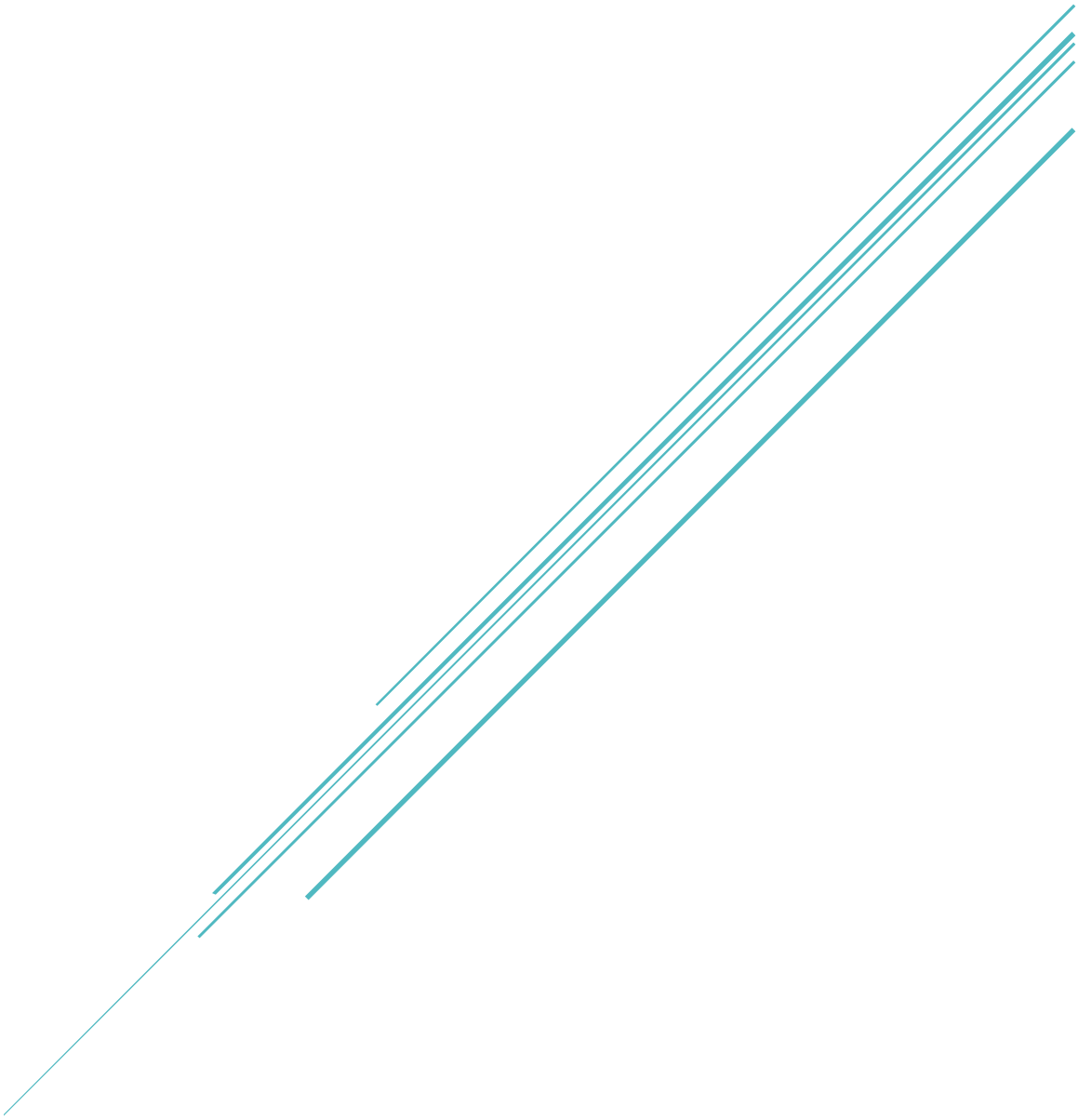
PREDICATION OF BIKE RENTAL COUNT

Predication of Bike Rental Count on Daily Based on the Environmental and Seasonal Settings



Chirag Nayak
15th, September 2018

Contents

Introduction

My objective of the analysis is to find out predictions of bike rental count on daily based on the environmental and seasonal settings, construct statistical models and then try to make prediction on rentals based on the information and models which I have. My exploration and the analysis of the data will be performed in Jupyter Notebook using Python; Tableau will be used for visualization.

Edwisor provides the data I will be look into. This is bike rental data spanning two years dating from January 1, 2011 to December 31, 2012. The dataset is also joined by the weather and season statistics for the corresponding date. However, the column named "hr" as given in the description of the project is not provided in the data set so I have to analyze without it.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
1	instant	dteday	season	yr	mnth	holiday	weekday	workingda	weathersit	temp	atemp	hum	windspeed	casual	registered	cnt
2	1	01-01-2011	1	0	1	0	6	0	2	0.344167	0.363625	0.805833	0.160446	331	654	985
3	2	02-01-2011	1	0	1	0	0	0	2	0.363478	0.353739	0.696087	0.248539	131	670	801
4	3	03-01-2011	1	0	1	0	1	1	1	0.196364	0.189405	0.437273	0.248309	120	1229	1349
5	4	04-01-2011	1	0	1	0	2	1	1	0.2	0.212122	0.590435	0.160296	108	1454	1562
6	5	05-01-2011	1	0	1	0	3	1	1	0.226957	0.22927	0.436957	0.1869	82	1518	1600
7	6	06-01-2011	1	0	1	0	4	1	1	0.204348	0.233209	0.518261	0.0895652	88	1518	1606
8	7	07-01-2011	1	0	1	0	5	1	2	0.196522	0.208839	0.498696	0.168726	148	1362	1510
9	8	08-01-2011	1	0	1	0	6	0	2	0.165	0.162254	0.535833	0.266804	68	891	959
10	9	09-01-2011	1	0	1	0	0	0	1	0.138333	0.116175	0.434167	0.36195	54	768	822
11	10	10-01-2011	1	0	1	0	1	1	1	0.150833	0.150888	0.482917	0.223267	41	1280	1321
12	11	11-01-2011	1	0	1	0	2	1	2	0.169091	0.191464	0.686364	0.122132	43	1220	1263
13	12	12-01-2011	1	0	1	0	3	1	1	0.172727	0.160473	0.599545	0.304627	25	1137	1162
14	13	13-01-2011	1	0	1	0	4	1	1	0.165	0.150883	0.470417	0.301	38	1368	1406
15	14	14-01-2011	1	0	1	0	5	1	1	0.16087	0.188413	0.537826	0.126548	54	1367	1421
16	15	15-01-2011	1	0	1	0	6	0	2	0.233333	0.248112	0.49875	0.157963	222	1026	1248
17	16	16-01-2011	1	0	1	0	0	0	1	0.231667	0.234217	0.48375	0.188433	251	953	1204
18	17	17-01-2011	1	0	1	1	1	0	2	0.175833	0.176771	0.5375	0.194017	117	883	1000
19	18	18-01-2011	1	0	1	0	2	1	2	0.216667	0.232333	0.861667	0.146775	9	674	683
20	19	19-01-2011	1	0	1	0	3	1	2	0.292174	0.298422	0.741739	0.208317	78	1572	1650
21	20	20-01-2011	1	0	1	0	4	1	2	0.261667	0.25505	0.538333	0.195904	83	1844	1927
22	21	21-01-2011	1	0	1	0	5	1	1	0.1775	0.157833	0.457083	0.353242	75	1468	1543
23	22	22-01-2011	1	0	1	0	6	0	1	0.05913	0.07907	0.4	0.17197	93	888	981
24	23	23-01-2011	1	0	1	0	0	0	1	0.096522	0.098839	0.436522	0.2466	150	836	986

Uploading the data set in Jupyter Notebook to analyze it with python

```
import os
os.chdir("E:\\R WorkStation\\Python WorkStation")
os.getcwd()
import pandas as pd
df= pd.read_csv("day.csv", sep = ',')
df
```

As I uploaded the file to Jupyter Notebook, I can see the following Variables

- instant: Record index
- dteday: Date
- season:- Season
  - 1:springer Season
  - 2:summer Season
  - 3:fall Season
  - 4:winter Season
- yr: Year (0: 2011, 1:2012)
- mnth: Month (1 to 12)
- hr: Hour (0 to 23) <Missing in dataset>
- holiday: weather day is holiday or not (extracted from Holiday Schedule)
- weekday: Day of the week.
- 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 count of casual users.
- registered: count of registered users.
- cnt: count of total rental bikes including both casual and registered.

Getting shape of the data set

```
df.shape
```

```
(731, 16)
```

The dataset I will be using contains 731 observations and 16 variables.

Data Cleaning

Removing Irrelevant columns:-

```
df=df.drop(columns=['atemp', 'casual', 'registered'])
```

I removed "atemp" variable since it is not a relatively accurate statistic to acquire because it is the feeling temperature from which we cannot draw any statistical conclusion. I also remove the "casual" and "registered" variable from the dataset because they sum up to "count" and my analysis later will not use them.Now the resultant data set is-

	instant	dteday	season	yr	mnth	holiday	weekday	workingday	weathersit	temp	hum	windspeed	cnt
0	1	2011-01-01	1	0	1	0	6	0	2	0.344167	0.805833	0.160446	985
1	2	2011-01-02	1	0	1	0	0	0	2	0.363478	0.696087	0.248539	801
2	3	2011-01-03	1	0	1	0	1	1	1	0.196364	0.437273	0.248309	1349
3	4	2011-01-04	1	0	1	0	2	1	1	0.200000	0.590435	0.160296	1562
4	5	2011-01-05	1	0	1	0	3	1	1	0.226957	0.436957	0.186900	1600
5	6	2011-01-06	1	0	1	0	4	1	1	0.204348	0.518261	0.089565	1606
6	7	2011-01-07	1	0	1	0	5	1	2	0.196522	0.498696	0.168726	1510
7	8	2011-01-08	1	0	1	0	6	0	2	0.165000	0.535833	0.266804	959
8	9	2011-01-09	1	0	1	0	0	0	1	0.138333	0.434167	0.361950	822
9	10	2011-01-10	1	0	1	0	1	1	1	0.150833	0.482917	0.223267	1321
10	11	2011-01-11	1	0	1	0	2	1	2	0.169091	0.686364	0.122132	1263
11	12	2011-01-12	1	0	1	0	3	1	1	0.172727	0.599545	0.304627	1162
12	13	2011-01-13	1	0	1	0	4	1	1	0.165000	0.470417	0.301000	1406
13	14	2011-01-14	1	0	1	0	5	1	1	0.160870	0.537826	0.126548	1421
14	15	2011-01-15	1	0	1	0	6	0	2	0.233333	0.498750	0.157963	1248
15	16	2011-01-16	1	0	1	0	0	0	1	0.231667	0.483750	0.188433	1204
16	17	2011-01-17	1	0	1	1	1	0	2	0.175833	0.537500	0.194017	1000
17	18	2011-01-18	1	0	1	0	2	1	2	0.216667	0.861667	0.146775	683

Exploratory Data Analysis

Checking for NaN values in table:-

```
df.isnull().any().any()
```

```
False
```

Looks like there is no Nan values in the data frame.

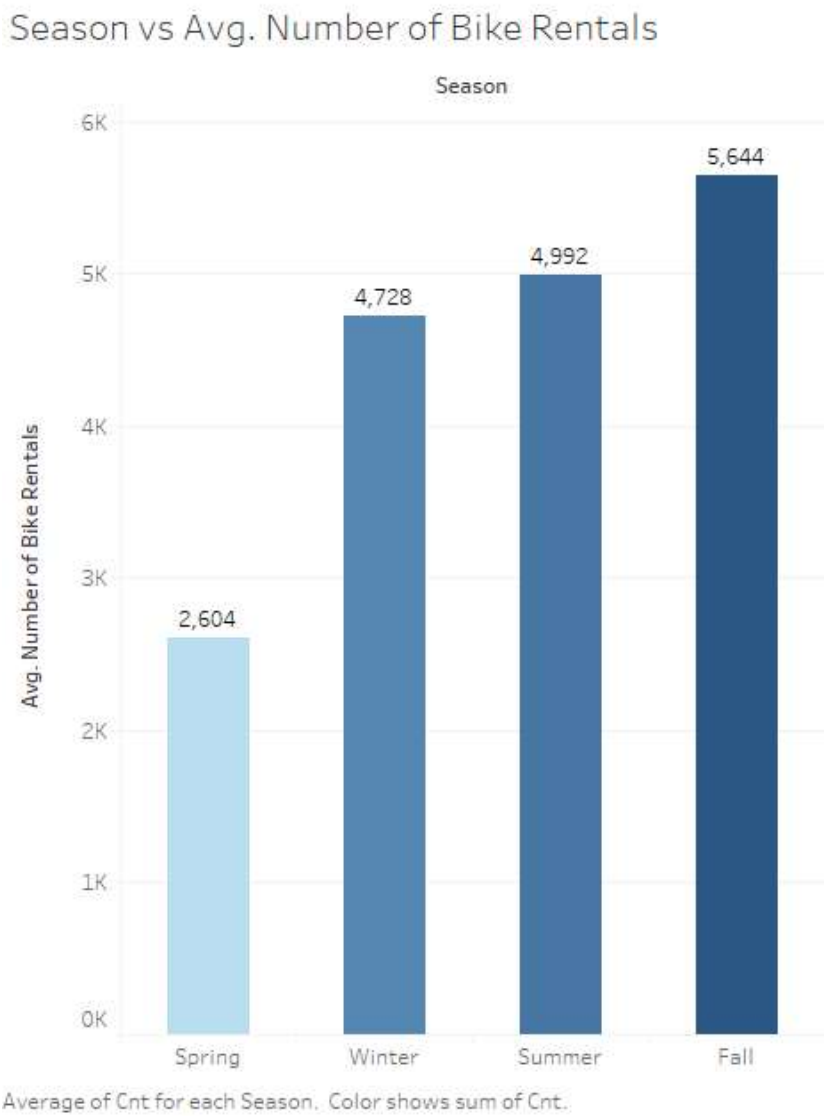
I created a csv file using the relevant variables to explore and get more insight with Tableaue.

```
df.to_csv("day2.csv")
```

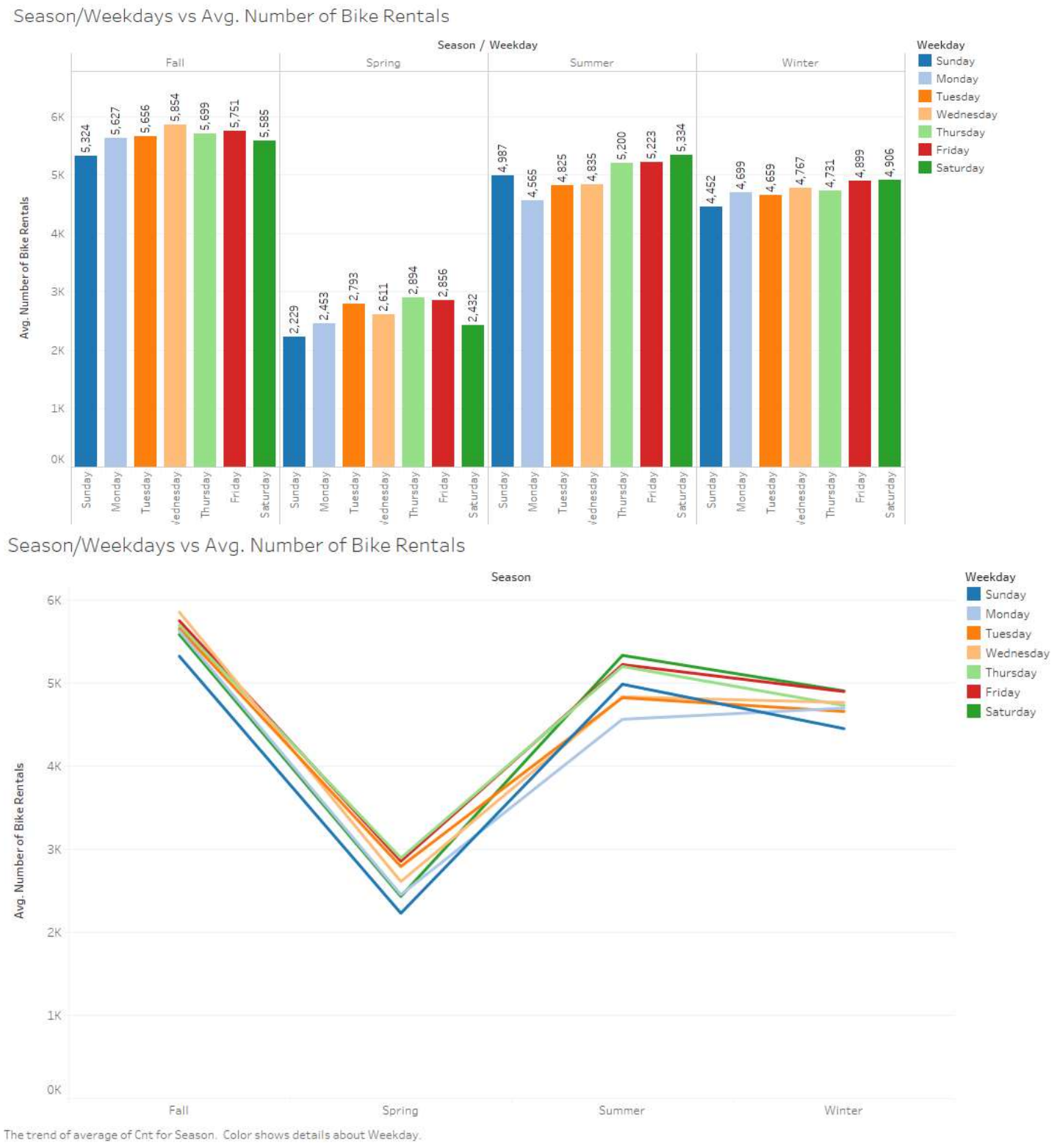
Without building any model or making any predictions, let us first look at the data by itself.

I construct a data frame that summarizes the bike rental count base on the season, month, day of the week, is it a weekday, is it a holiday, and the type of weather then calculating the mean of temperature, humidity, wind speed and rental count. The purpose of this summarization is to find a general relationship between variables regardless of which year the data is from (since the data spans two years and the business is growing.)

Visualization

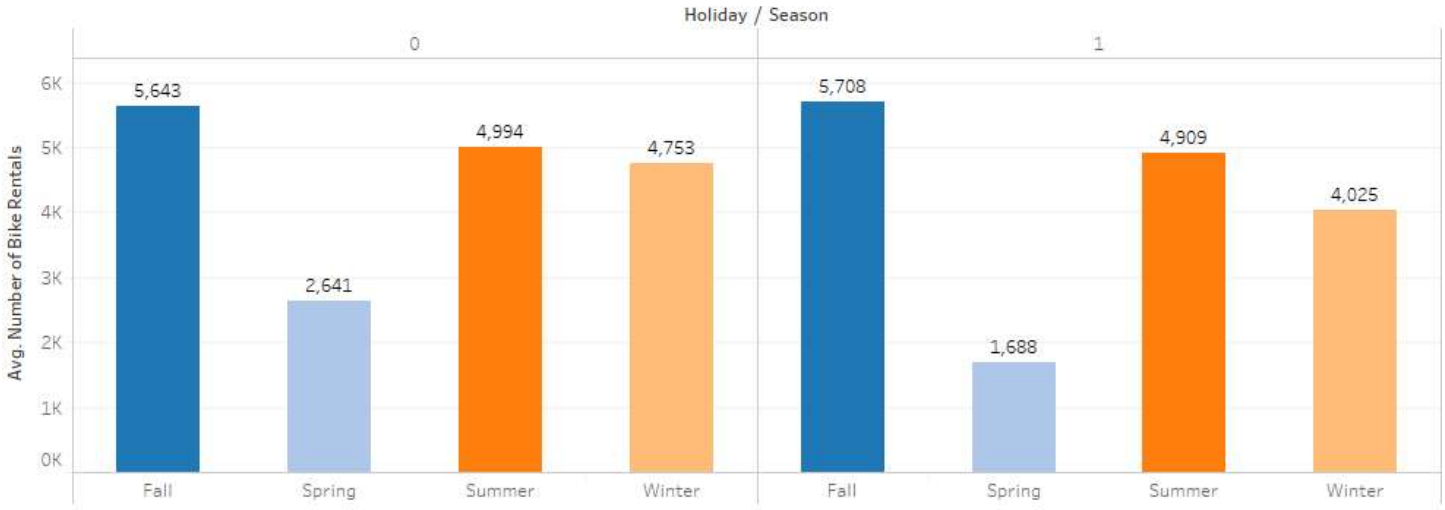


The above Histogram of different seasons against bike rental count reveals that there is a seasonal trend with the rental count. Rental count is generally low in Spring and high in Fall season. Season can be one of the determining factors that affects bike rental count.

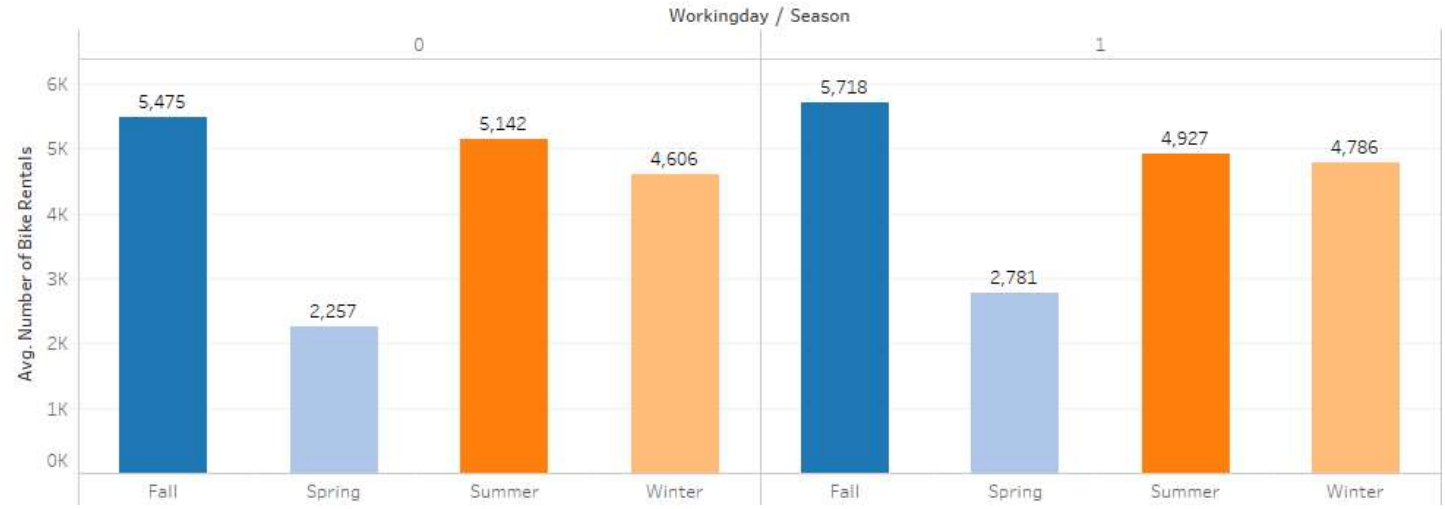


The above histogram and line plot of Season against Avg. Number of Bike Rentals categorize by day of the week shows the difference of rental demand for weekday and weekend. This shows that rental demand is not effected by the weekdays. However, the seasons have more impact on rental demand.

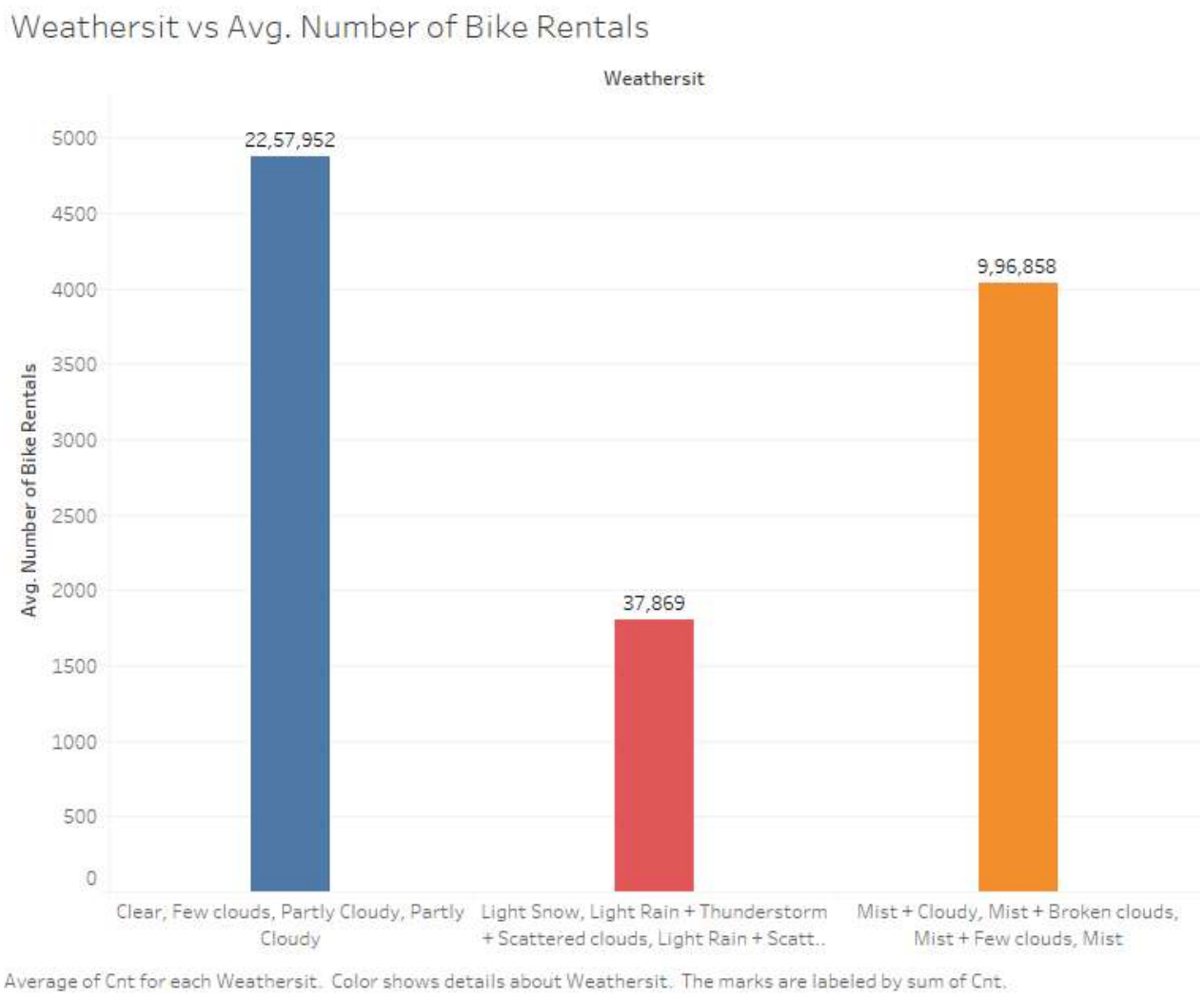
Season/Holidays vs Avg. Number of Bike Rentals



Season/Workingday vs Avg. Number of Bike Rentals



This above histogram compares effects of holidays and workingdays, We can see that the average amounts of bike rental demand are about the same regardless of being a holiday or not and working days or not. We can also see similar seasonality to the season vs bike rental demand; Spring shows the lowest in rental demand and Fall shows the highest.

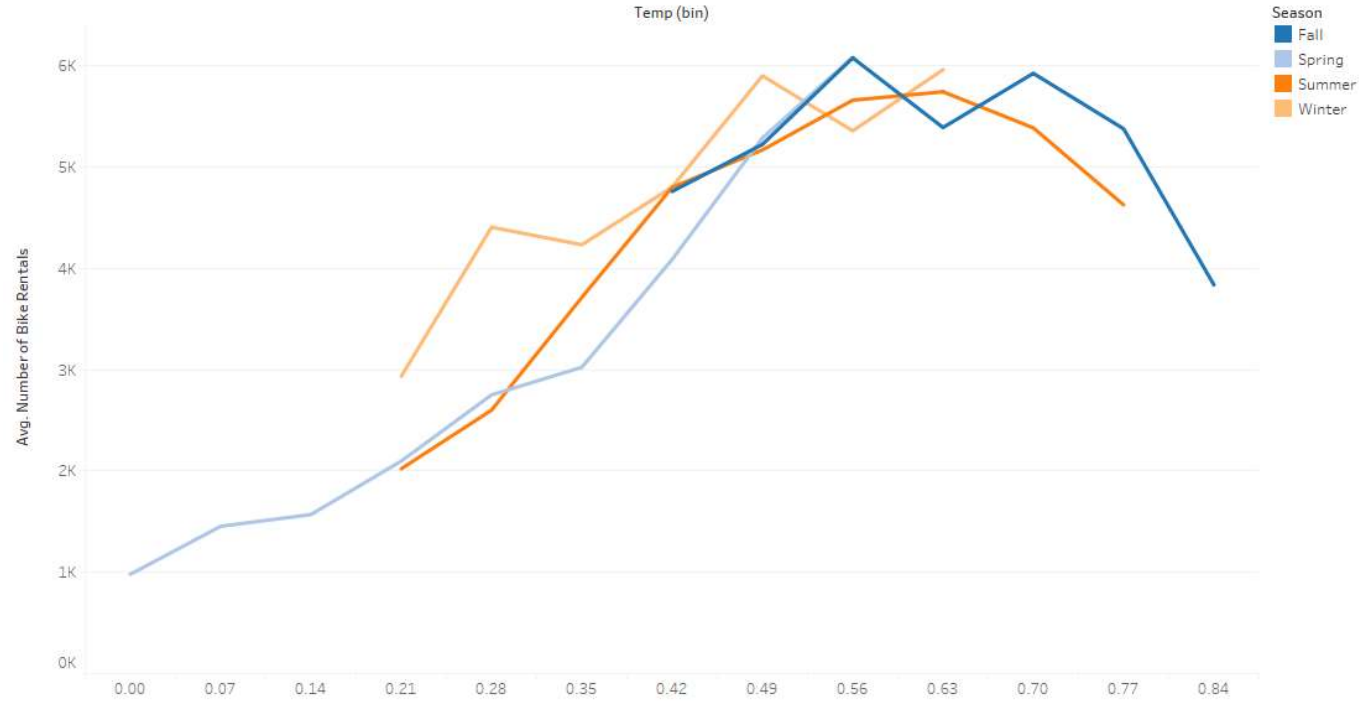


In the above histogram, plotting different type of weather against bike rental demand count, indicates that the demand of bike rental is good in "Mist + Cloudy, Mist + Broken clouds, Mist + Few clouds, Mist", with better overall count in "Clear, Few clouds, Partly Cloudy, Partly Cloudy". "Light Snow, Light Rain + Thunderstorm + Scattered clouds, Light Rain + Scatteredclouds" shows significantly lower average rental count. I believe our dataset do not contain observations in really bad weathers so the histogram for weather type 3 is a placeholder and do not have significant meaning.

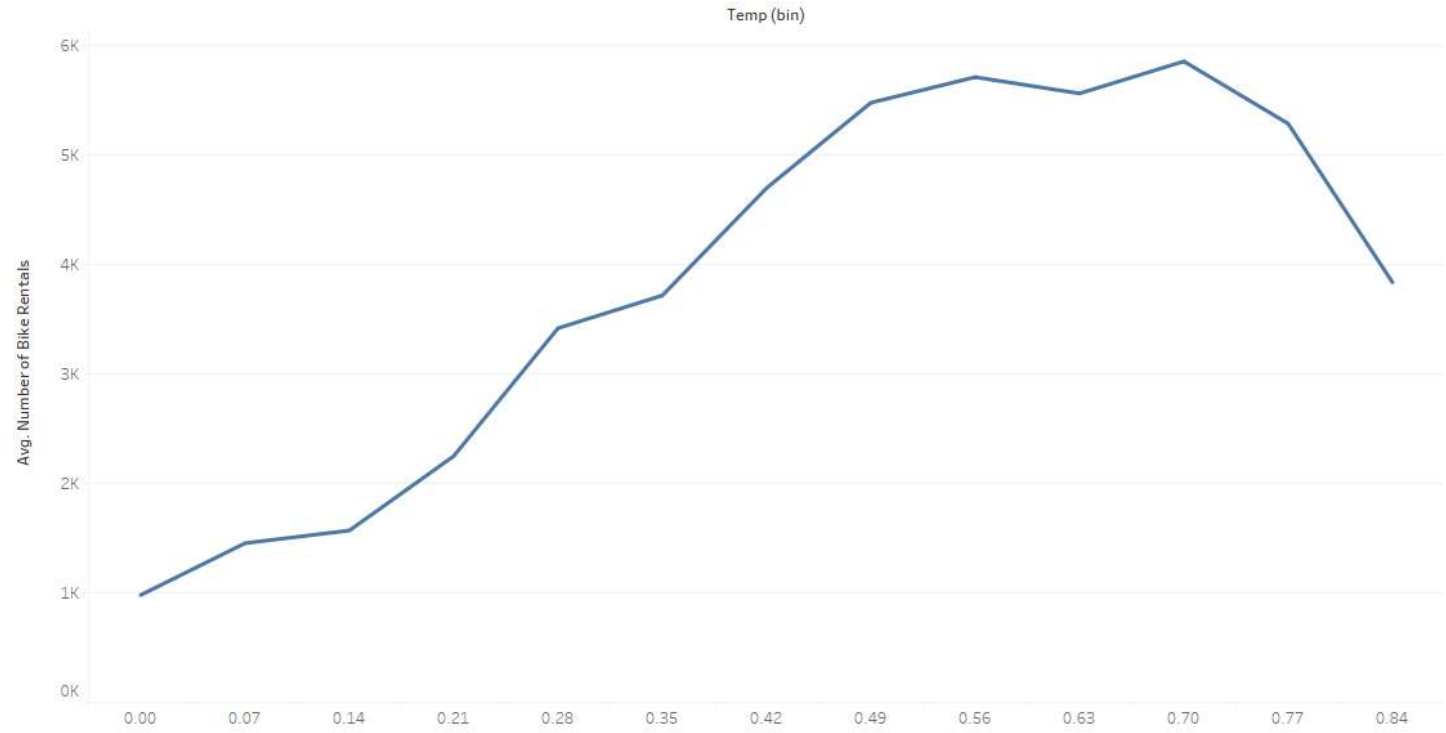
Looks like it is better to analyze the data set by getting more insight from Season's data.

Therefore, I also plot temperature, humidity, and wind speed against bike rental count categorized by different Season types.

Temperature vs Avg. Number of Bike Rentals



The trend of average of Cnt for Temp (bin). Color shows details about Season.

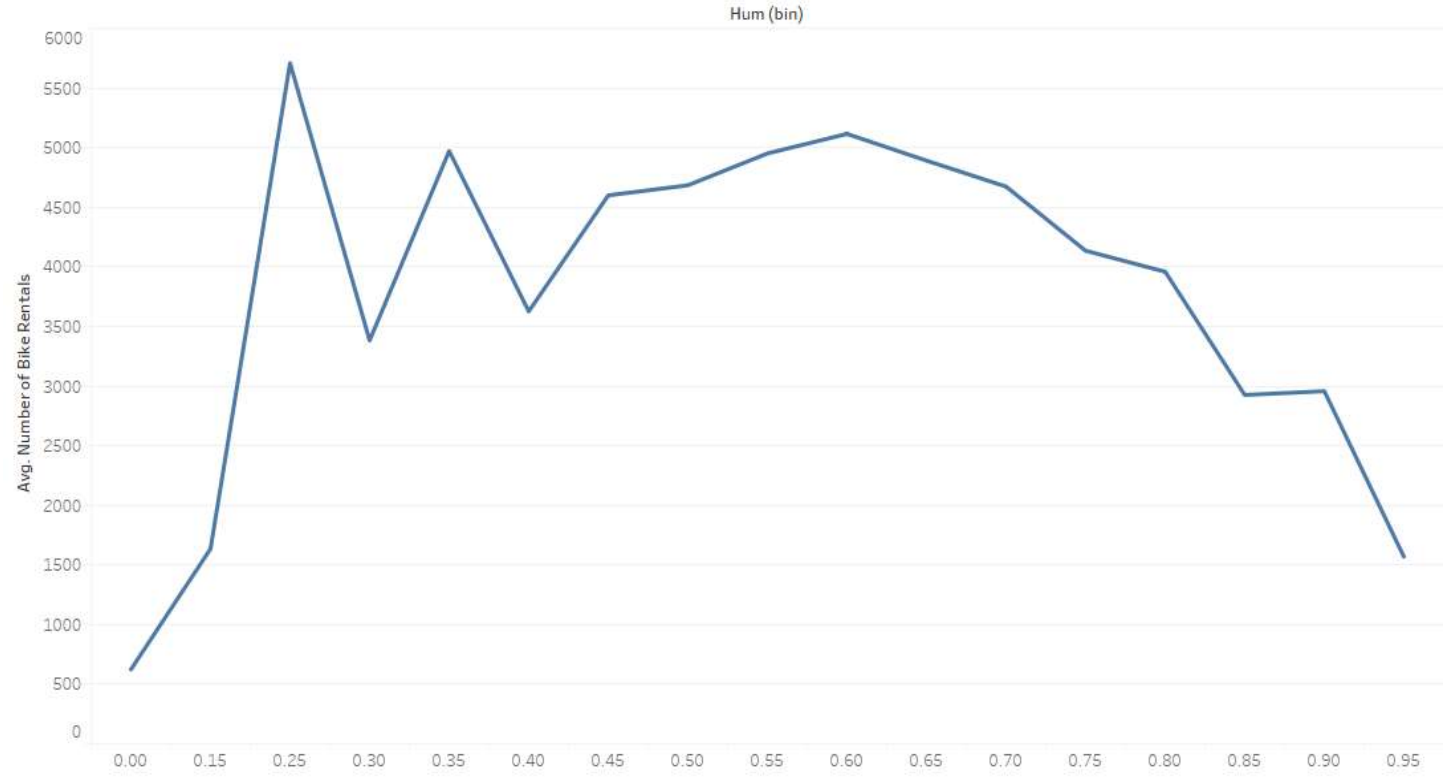


The trend of average of Cnt for Temp (bin).

The temperature plot shows that generally, the warmer the temperature, the higher bike rental demand. However, in Fall Season, the rental count peaks to Maximum; whereas in Summer Season, the rental demand peaks higher around .0.49.

Humidity vs Avg. Number of Bike Rentals

The trend of average of Cnt for Hum (bin). Color shows details about Season.



The trend of average of Cnt for Hum (bin).

The humidity plot shows that generally, the higher the relative humidity, the lower bike rental demand in any Season.

The wind speed plot shows that although people enjoy gentle breeze in good weathers, the bike rental demand is significantly lower no matter the wind speed in Spring and higher in Fall Season.

Correlation matrix

By preparing a correlation matrix, we can have a more straightforward view of what variables are strongly correlated and what is weakly correlated.

```
df.corr()
```

	instant	season	yr	mnth	holiday	weekday	workingday	weathersit	temp	hum	windspeed	cnt
instant	1.000000	0.412224	0.866025	0.496702	0.016145	-0.000016	-0.004337	-0.021477	0.150580	0.016375	-0.112620	0.628830
season	0.412224	1.000000	-0.001844	0.831440	-0.010537	-0.003080	0.012485	0.019211	0.334315	0.205445	-0.229046	0.406100
yr	0.866025	-0.001844	1.000000	-0.001792	0.007954	-0.005461	-0.002013	-0.048727	0.047604	-0.110651	-0.011817	0.566710
mnth	0.496702	0.831440	-0.001792	1.000000	0.019191	0.009509	-0.005901	0.043528	0.220205	0.222204	-0.207502	0.279977
holiday	0.016145	-0.010537	0.007954	0.019191	1.000000	-0.101960	-0.253023	-0.034627	-0.028556	-0.015937	0.006292	-0.068348
weekday	-0.000016	-0.003080	-0.005461	0.009509	-0.101960	1.000000	0.035790	0.031087	-0.000170	-0.052232	0.014282	0.067443
workingday	-0.004337	0.012485	-0.002013	-0.005901	-0.253023	0.035790	1.000000	0.061200	0.052660	0.024327	-0.018796	0.061156
weathersit	-0.021477	0.019211	-0.048727	0.043528	-0.034627	0.031087	0.061200	1.000000	-0.120602	0.591045	0.039511	-0.297391
temp	0.150580	0.334315	0.047604	0.220205	-0.028556	-0.000170	0.052660	-0.120602	1.000000	0.126963	-0.157944	0.627494
hum	0.016375	0.205445	-0.110651	0.222204	-0.015937	-0.052232	0.024327	0.591045	0.126963	1.000000	-0.248489	-0.100659
windspeed	-0.112620	-0.229046	-0.011817	-0.207502	0.006292	0.014282	-0.018796	0.039511	-0.157944	-0.248489	1.000000	-0.234545
cnt	0.628830	0.406100	0.566710	0.279977	-0.068348	0.067443	0.061156	-0.297391	0.627494	-0.100659	-0.234545	1.000000

We can clearly see from the matrix that Temperature and Season has the strongest correlation to bike rental count while all variables considered. We can disregard the significantly high correlation between season and month since it is only natural for them to have high correlation. This correlation matrix also shows that weathersit, windspeed and holiday are negatively correlated to bike rental demand count as shown by previous graphs. Temperature and Season also have high correlation with each other.

From the above analysis. We found that Season and Temperature are two variables, which highly correlated to bike rental count demand. Temperature and Season also have high correlation with each other. So, to get more insight we need to focus on relation between Temperature and Season.

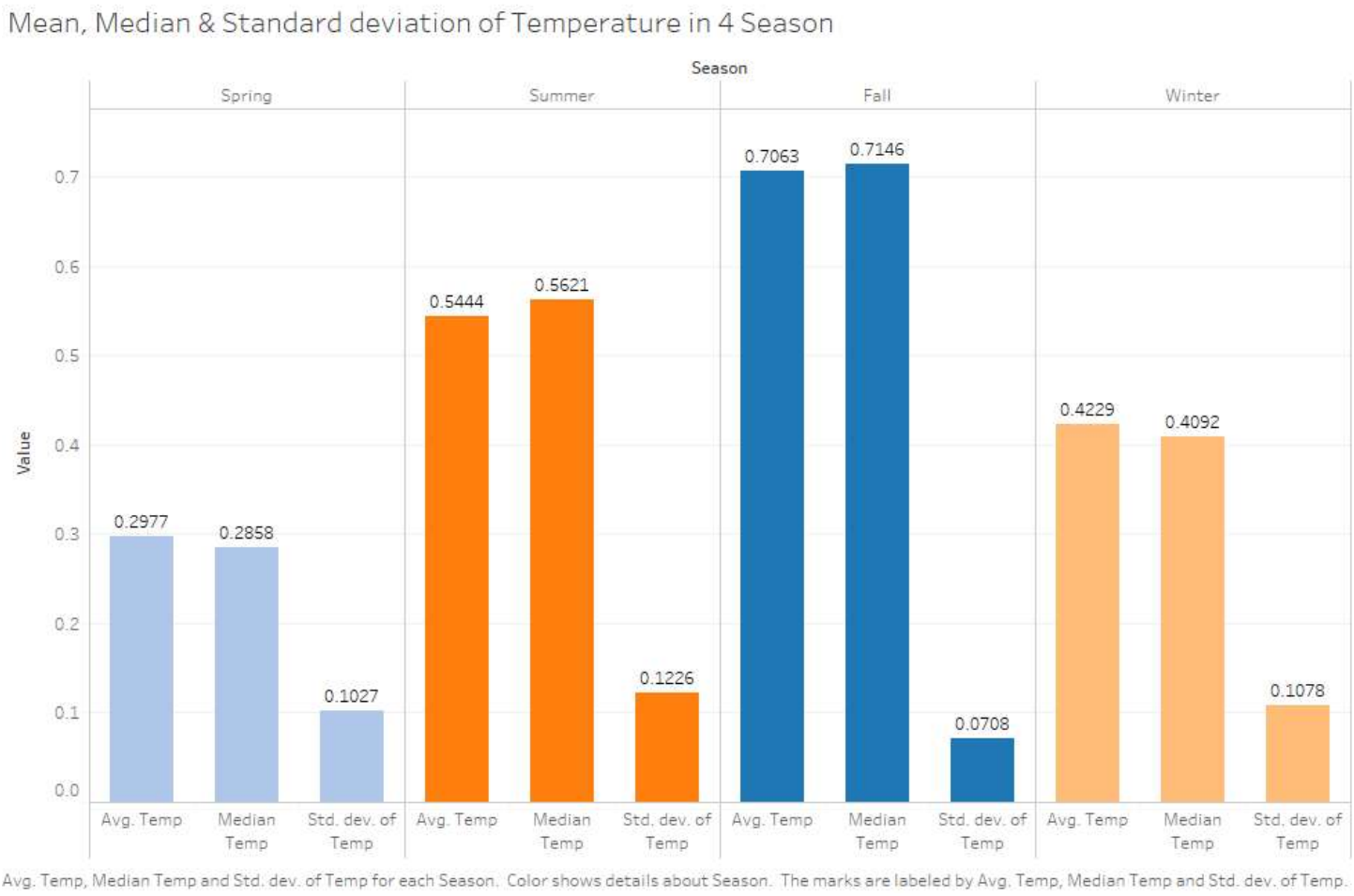
I calculated Mean, Median and Standard deviation of temperature for all 4 different season.

Mean in the average value of temperature in any given season.

Median provides a helpful measure of the center of a dataset. By comparing the median to the mean, you can get an idea of the distribution of a dataset. When the mean and the median are the same, the dataset is more or less evenly distributed from the lowest to highest values. When the mean and the median are different then it is likely the data is not symmetrical but is skewed either to the left or the right.

Standard deviation is a number used to tell how measurements for a group are spread out from the average (mean), or expected value. A low standard deviation means that most of

the numbers are very close to the average. A high standard deviation means that the numbers are far from mean.



As we can see that, the temperature is relatively high across all seasons. The lowest mean temperature was in spring mean temperature of 0.2977 and standard deviation of 0.1027. The highest mean temperature was in fall mean temperature of 0.7063 and standard deviation of 0.0708. In between there is summer with a mean temperature of 0.5444 and standard deviation of 0.1226 and in winter with a mean temperature of 0.4229 and standard deviation of 0.1078.

Because of the mild temperatures in spring and winter, the bike rental demand in these two Season is low and because of the warm weather in summer and fall, the bike rental demand is more in these two Season. This also shows that temperature is highly correlated with the total amount of bike rentals. In this analysis, we show correlation of the Temperature with bike rental count in correlation matrix. This also shows that bike rental count depends on Temperature.

Model Building

Data Preparation

From all previous analysis, I came to know that only Season and Temperature are the two variables, which can be used for predictions. I am not using date because we can see that, the bike rental count depends on season and temperature on a particular given date.

Before diving into building statistical models, I converted all Measurement variable to numeric data type and all dimensions variable to categorical data type.

```
df.season=pd.Categorical(df.season)
df.yr=pd.Categorical(df.yr)
df.mnth=pd.Categorical(df.mnth)
df.holiday=pd.Categorical(df.holiday)
df.weekday=pd.Categorical(df.weekday)
df.workingday=pd.Categorical(df.workingday)
df.weathersit=pd.Categorical(df.weathersit)
df.temp=pd.to_numeric(df.temp)
df.hum=pd.to_numeric(df.hum)
df.windspeed=pd.to_numeric(df.windspeed)
df.cnt=pd.to_numeric(df.cnt)
```

I also did convert the season variable to dummy variables to make machine-learning algorithms understand it better since it is a categorical variable.

```
dum = pd.get_dummies(df.season)
dum
```

	1	2	3	4
0	1	0	0	0
1	1	0	0	0
2	1	0	0	0
3	1	0	0	0
4	1	0	0	0
5	1	0	0	0
6	1	0	0	0

I partition my dataset into two sets, training and testing. Training set will be used to train statistical models and estimate coefficients, while testing set will be used to validate the model we build with the training set. 75% of the complete data is partitioned into training set, sampled uniformly without replacement, and 25% is partitioned in to testing set. Sampling without replacement enables the model we build to extrapolate on the testing data, giving us a better sense of how our statistical models perform.

```
X = df[['temp']]
X= pd.concat([X,dum],axis='columns')
y = df['cnt']
```

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state = 0)
```

The resulting set contains-

```
X_train.shape
```

```
(511, 9)
```

```
X_test.shape
```

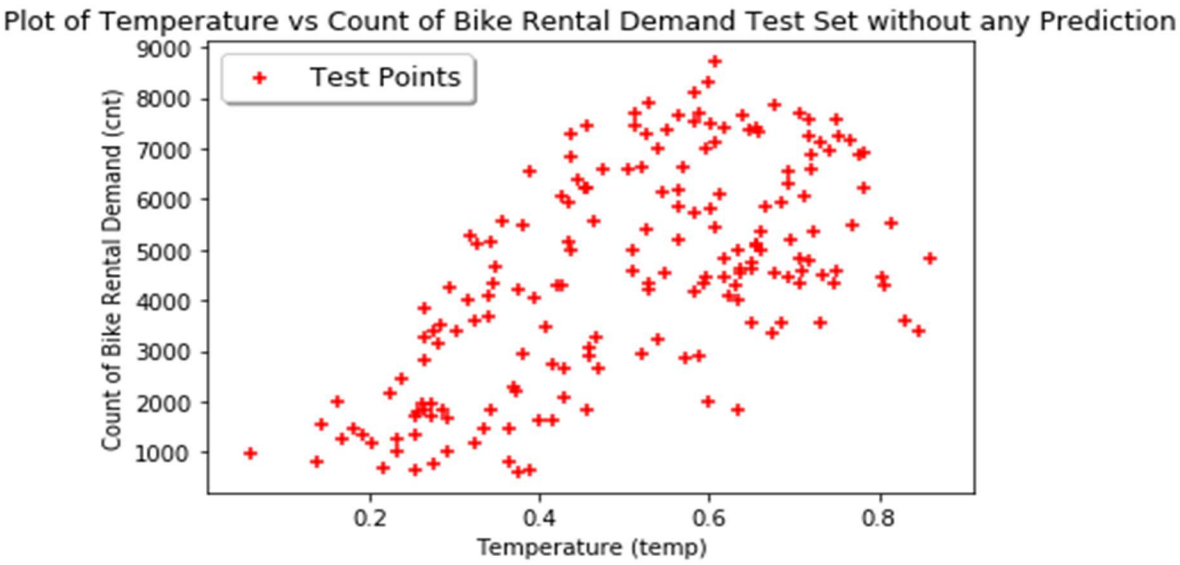
```
(220, 9)
```

511 observations in training set and 220 observations in testing set.

Prediction

Before the prediction should note this graph, which is without any prediction for reference/base line:-

```
%matplotlib inline
plt.title('Plot of Temperature vs Count of Bike Rental Demand Test Set without any Prediction')
plt.xlabel('Temperature (temp)')
plt.ylabel('Count of Bike Rental Demand (cnt)')
plt.scatter(X_test.temp,y_test,color='red',marker='+',label="Test Points")
plt.legend(loc="best",shadow=True,fontsize='large')
```



I will be using '5' Machine Learning Algorithm for prediction:-

- Linear Regression.
- Decision Tree Regressor.
- Random Forest Regressor
- K-Neighbors Regressor
- XGBoost Regressor

For every Machine Learning Algorithm I will be computing k-Fold Cross Validation to know its accuracy. k-Fold Cross Validation will tell us which Machine Learning Algorithm is working most efficiently for this model. I will also be calculating Mean Absolute Error (MAE) and Mean

Squared Error (MSE) for every Machine Learning Algorithm. To know the accuracy of the model better.

K-Fold Cross Validation

Cross-validation, sometimes called rotation estimation, or out-of-sample testing is any of various similar model validation techniques for assessing how the results of a statistical analysis will generalize to an independent data set. It is mainly used in settings where the goal is prediction, and one wants to estimate how accurately a predictive model will perform in practice. In a prediction problem, a model is usually given a dataset of known data on which training is run (training dataset), and a dataset of unknown data (or first seen data) against which the model is tested (called the validation dataset or testing set). The goal of cross-validation is to test the model's ability to predict new data that was not used in estimating it, in order to flag problems like overfitting and to give an insight on how the model will generalize to an independent dataset (i.e., an unknown dataset, for instance from a real problem).

One round of cross-validation involves partitioning a sample of data into complementary subsets, performing the analysis on one subset (called the training set), and validating the analysis on the other subset (called the validation set or testing set). To reduce variability, in most methods multiple rounds of cross-validation are performed using different partitions, and the validation results are combined (e.g. averaged) over the rounds to give an estimate of the model's predictive performance.

In summary, cross-validation combines (averages) measures of fitness in prediction to derive a more accurate estimate of model prediction performance.

In k-fold cross-validation, the original sample is randomly partitioned into k equal size subsamples. Of the k subsamples, a single subsample is retained as the validation data for testing the model, and the remaining k-1 subsamples are used as training data. The cross-validation process is then repeated k times (the folds), with each of the k subsamples used exactly once as the validation data. The k results from the folds can then be averaged (or otherwise combined) to produce a single estimation.

The advantage of this method is that all observations are used for both training and validation, and each observation is used for validation exactly once.
For classification problems, one typically uses stratified k-fold cross-validation, in which the folds are selected so that each fold contains roughly the same proportions of class labels.
In repeated cross-validation, the cross-validation procedure is repeated n times, yielding n random partitions of the original sample. Then results are again averaged (or otherwise combined) to produce a single estimation.

OpenML generates train-test splits given the number of folds and repeats, so that different users can evaluate their models with the same splits. Stratification is applied by default for classification problems (unless otherwise specified). The splits are given as part of the task description as an ARFF file with the row id, fold number, repeat number and the class (TRAIN

or TEST). The uploaded predictions should be labeled with the fold and repeat number of the test instance, so that the results can be properly evaluated and aggregated. OpenML stores both the per fold/repeat results and the aggregated scores.

Mean absolute error

Mean Absolute Error (MAE): MAE measures the average magnitude of the errors in a set of predictions, without considering their direction. It's the average over the test sample of the absolute differences between prediction and actual observation where all individual differences have equal weight.

$$\mathrm{MAE} = \frac{1}{n}\sum_{j=1}^{n}|y_j - \hat{y}_j|$$

Mean Squared Error (MSE)

In statistics, the mean squared error or mean squared deviation of an estimator measures the average of the squares of the errors—that is, the average squared difference between the estimated values and what is estimated. MSE is a risk function, corresponding to the expected value of the squared error loss.

MSE = $1n \sum ni=1(Xi^{\wedge}-Xi)2$

Linear regression

In statistics, **linear regression** is a linear approach to modelling the relationship between a scalar response (or dependent variable) and one or more explanatory variables (or independent variables). The case of one explanatory variable is called simple linear regression. For more than one explanatory variable, the process is called **multiple linear regression**. This term is distinct from multivariate linear regression, where multiple correlated dependent variables are predicted, rather than a single scalar variable.

In linear regression, the relationships are modeled using linear predictor functions whose unknown model parameters are estimated from the data. Such models are called linear models. Most commonly, the conditional mean of the response given the values of the explanatory variables (or predictors) is assumed to be an affine function of those values; less commonly, the conditional median or some other quantile is used. Like all forms of regression analysis, linear regression focuses on the conditional probability distribution of the response given the values of the predictors, rather than on the joint probability distribution of all of these variables, which is the domain of multivariate analysis.

Linear regression was the first type of regression analysis to be studied rigorously, and to be used extensively in practical applications. This is because models which depend linearly on their unknown parameters are easier to fit than models which are non-linearly related to their parameters and because the statistical properties of the resulting estimators are easier to determine.

```
from sklearn import linear_model
reg=linear_model.LinearRegression(n_jobs=5,normalize=tuple)
reg.fit(X_train, y_train)
```

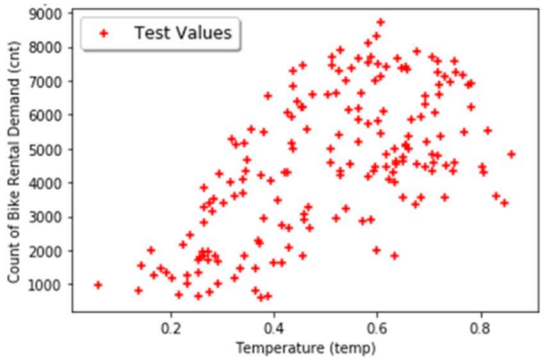
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=5,
         normalize=<class 'tuple'>)
```

```
pl=reg.predict(X_test)
```

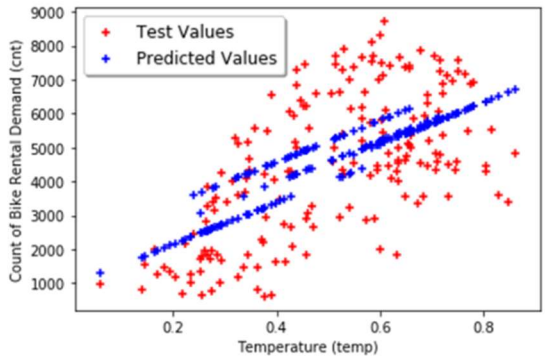
```
%matplotlib inline
plt.title('Plot of Temperature vs Count of Bike Rental Demand Test Set with Predictions using Linear Regression')
plt.xlabel('Temperature (temp)')
plt.ylabel('Count of Bike Rental Demand (cnt)')
plt.scatter(X_test.temp,y_test,color='red',marker='+',label="Test Values")
plt.scatter(X_test.temp,pl,color='blue',marker='+',label="Predicted Values")
plt.legend(loc="best",shadow=True,fontsize='large')
```

Comparing Original values with Predictions using Linear Regression

Original Values



Predated Values using Linear Regression



Computing k-Fold Cross Validation for Linear regression Algorithm

```
from sklearn.model_selection import cross_val_score
accuracies = cross_val_score(estimator = reg, X = X_train, y = y_train, cv = 10)
print('accuracies.mean:-')
print(accuracies.mean())
print('accuracies.std:-')
print(accuracies.std())
```

```
accuracies.mean:-
0.432094900079
accuracies.std:-
0.0497170296907
```

```
accuracies
```

```
array([ 0.41213873,  0.40029026,  0.52846165,  0.44253036,  0.34280751,
        0.45781119,  0.48561911,  0.42474271,  0.38453645,  0.44201101])
```

Computing MAE and MSE for Logistic Regression Algorithm

```
from sklearn.metrics import mean_absolute_error
mean_absolute_error(y_test, reg.predict(X_test))
```

```
1327.974877423602
```

```
from sklearn.metrics import mean_squared_error
mean_squared_error(y_test, reg.predict(X_test))
```

```
2406732.7612735084
```

Decision Trees

A tree has many analogies in real life, and turns out that it has influenced a wide area of machine learning, covering both classification and regression. In decision analysis, a decision tree can be used to visually and explicitly represent decisions and decision making. As the name goes, it uses a tree-like model of decisions. Though a commonly used tool in data mining for deriving a strategy to reach a particular goal, its also widely used in machine learning, which will be the main focus of this article.

A decision tree is drawn upside down with its root at the top. In the image on the left, the bold text in black represents a condition/internal node, based on which the tree splits into branches/ edges. The end of the branch that doesn't split anymore is the decision/leaf, in this case, whether the passenger died or survived, represented as red and green text respectively.

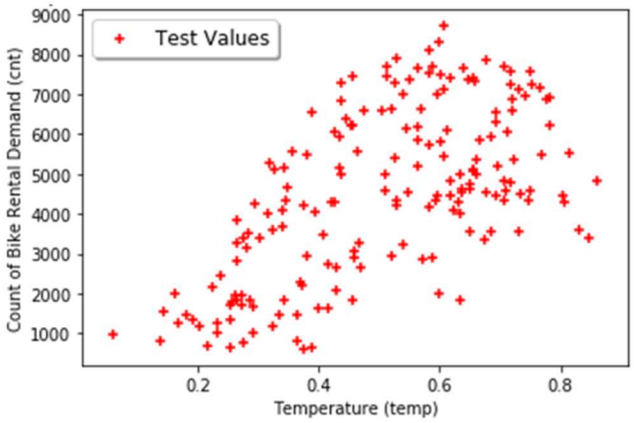
Although, a real dataset will have a lot more features and this will just be a branch in a much bigger tree, but you can't ignore the simplicity of this algorithm. The feature importance is clear and relations can be viewed easily. This methodology is more commonly known as learning decision tree from data and above tree is called Classification tree as the target is to classify passenger as survived or died. Regression trees are represented in the same manner, just they predict continuous values like price of a house. In general, Decision Tree algorithms are referred to as CART or Classification and Regression Trees.

```
from sklearn.tree import DecisionTreeRegressor
reg1 = DecisionTreeRegressor(random_state = 0)
reg1.fit(X_train, y_train)
pdt=reg1.predict(X_test)
```

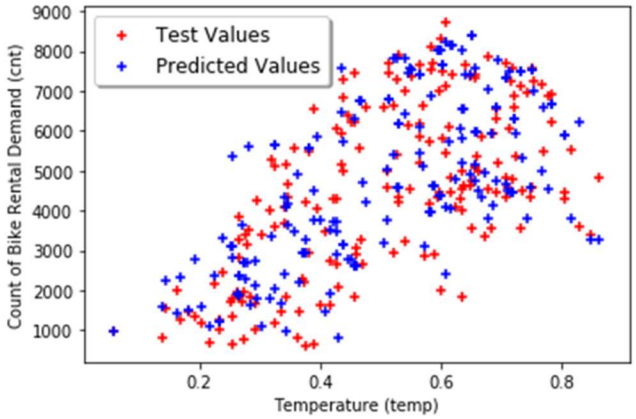
```
%matplotlib inline
plt.title('Plot of Temperature vs Count of Bike Rental Demand Test Set with Predictions using Decision Tree Regressor')
plt.xlabel('Temperature (temp)')
plt.ylabel('Count of Bike Rental Demand (cnt)')
plt.scatter(X_test.temp,y_test,color='red',marker='+',label="Test Values")
plt.scatter(X_test.temp,pdt,color='blue',marker='+',label="Predicted Values")
plt.legend(loc="best",shadow=True,fontsize='large')
```

Comparing Original values with Predictions using Decision Tree Regression

Original Values



Predated Values using Decision Tree Regression



Computing k-Fold Cross Validation for Decision Trees algorithm

```
from sklearn.model_selection import cross_val_score
accuracies = cross_val_score(estimator = reg1, X = X_train, y = y_train, cv = 10)
print('accuracies.mean:-')
print(accuracies.mean())
print('accuracies.std:-')
print(accuracies.std())
```

```
accuracies.mean:-
0.020876507934
accuracies.std:-
0.18860858286
```

```
accuracies
```

```
array([-0.12990179,  0.08404343, -0.12708077,  0.06290323,  0.00904324,
        0.28859999,  0.0600074 ,  0.24090439, -0.39669159,  0.11693755])
```

Computing MAE and MSE for Decision Trees Algorithm

```
from sklearn.metrics import mean_absolute_error
mean_absolute_error(y_test, reg1.predict(X_test))
```

```
1581.360655737705
```

```
from sklearn.metrics import mean_squared_error
mean_squared_error(y_test, reg1.predict(X_test))
```

```
4235798.0874316944
```

Random Forest Algorithm

Random Forest is a flexible, easy to use machine learning algorithm that produces, even without hyper-parameter tuning, a great result most of the time. It is also one of the most used algorithms, because it's simplicity and the fact that it can be used for both classification and regression tasks. In this post, you are going to learn, how the random forest algorithm works and several other important things about it.

Random Forest is a supervised learning algorithm. Like you can already see from its name, it creates a forest and makes it somehow random. The „forest" it builds, is an ensemble of Decision Trees, most of the time trained with the "bagging" method. The general idea of the bagging method is that a combination of learning models increases the overall result.

One big advantage of random forest is, that it can be used for both classification and regression problems, which form the majority of current machine learning systems. I will talk about random forest in classification, since classification is sometimes considered the building block of machine learning.

Random Forest has nearly the same hyperparameters as a decision tree or a bagging classifier. Fortunately, you don't have to combine a decision tree with a bagging classifier and can just easily use the classifier-class of Random Forest. Like I already said, with Random Forest, you can also deal with Regression tasks by using the Random Forest regressor.

Random Forest adds additional randomness to the model, while growing the trees. Instead of searching for the most important feature while splitting a node, it searches for the best feature among a random subset of features. This results in a wide diversity that generally results in a better model.

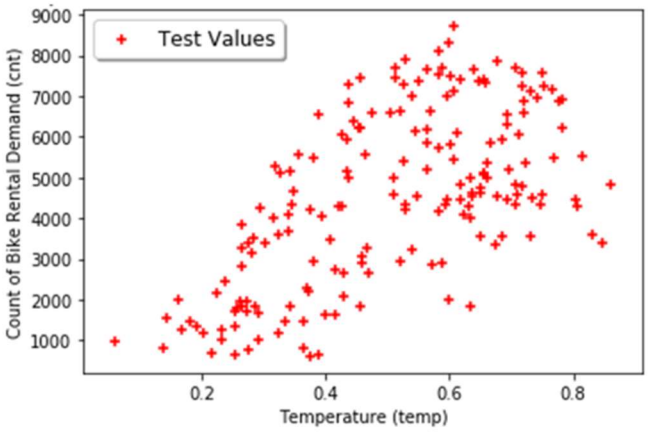
Therefore, in Random Forest, only a random subset of the features is taken into consideration by the algorithm for splitting a node. You can even make trees more random, by additionally using random thresholds for each feature rather than searching for the best possible thresholds (like a normal decision tree does).

```
from sklearn.ensemble import RandomForestRegressor
reg2 = RandomForestRegressor(n_estimators = 350, random_state = 0)
reg2.fit(X_train, y_train)
prf=reg2.predict(X_test)
```

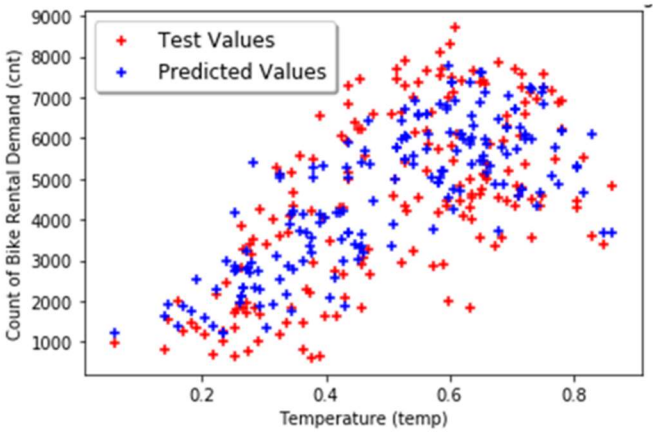
```
%matplotlib inline
plt.title('Plot of Temperature vs Count of Bike Rental Demand Test Set with Predictions using Random Forest Regressor')
plt.xlabel('Temperature (temp)')
plt.ylabel('Count of Bike Rental Demand (cnt)')
plt.scatter(X_test.temp,y_test,color='red',marker='+',label="Test Values")
plt.scatter(X_test.temp,prf,color='blue',marker='+',label="Predicted Values")
plt.legend(loc="best",shadow=True,fontsize='large')
```

Comparing Original values with Predictions using Random Forest Regression

Original Values



Predated Values using Random Forest Regression



Computing k-Fold Cross Validation for Random Forest Algorithm

```
from sklearn.model_selection import cross_val_score
accuracies = cross_val_score(estimator = reg2, X = X_train, y = y_train, cv = 10)
print('accuracies.mean:-')
print(accuracies.mean())
print('accuracies.std:-')
print(accuracies.std())
```

```
accuracies.mean:-
0.315571956926
accuracies.std:-
0.121652273288
```

```
accuracies
```

```
array([ 0.17602239,  0.24118459,  0.41600331,  0.40187424,  0.29672312,
        0.45178427,  0.29377835,  0.37425189,  0.05897763,  0.44511977])
```

Computing MAE and MSE for Random Forest Algorithm

```
from sklearn.metrics import mean_absolute_error
mean_absolute_error(y_test, reg2.predict(X_test))
```

```
1392.2312689429141
```

```
from sklearn.metrics import mean_squared_error
mean_squared_error(y_test, reg2.predict(X_test))
```

```
3107846.2061092304
```

K-Neighbors Regression

In pattern recognition, the k-nearest neighbors algorithm (k-NN) is a non-parametric method used for classification and regression.[1] In both cases, the input consists of the k closest training examples in the feature space. The output depends on whether k-NN is used for classification or regression:

In k-NN classification, the output is a class membership. An object is classified by a majority vote of its neighbors, with the object being assigned to the class most common among its k nearest neighbors (k is a positive integer, typically small). If k = 1, then the object is simply assigned to the class of that single nearest neighbor.

In k-NN regression, the output is the property value for the object. This value is the average of the values of its k nearest neighbors.

k-NN is a type of instance-based learning, or lazy learning, where the function is only approximated locally and all computation is deferred until classification. The k-NN algorithm is among the simplest of all machine learning algorithms.

Both for classification and regression, a useful technique can be used to assign weight to the contributions of the neighbors, so that the nearer neighbors contribute more to the average than the more distant ones. For example, a common weighting scheme consists in giving each neighbor a weight of 1/d, where d is the distance to the neighbor.

The neighbors are taken from a set of objects for which the class (for k-NN classification) or the object property value (for k-NN regression) is known. This can be thought of as the training set for the algorithm, though no explicit training step is required.

A peculiarity of the k-NN algorithm is that it is sensitive to the local structure of the data. The algorithm is not to be confused with k-means, another popular machine learning technique.

```
from sklearn.neighbors import KNeighborsRegressor
reg3 = KNeighborsRegressor(n_neighbors = 40, metric = 'minkowski', p = 2, weights='distance', n_jobs=10)
reg3.fit(X_train, y_train)
```

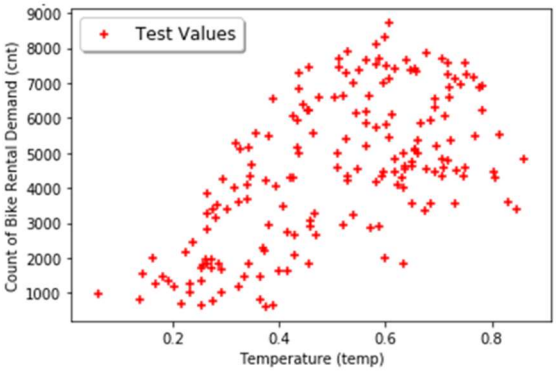
```
KNeighborsRegressor(algorithm='auto', leaf_size=30, metric='minkowski',
          metric_params=None, n_jobs=10, n_neighbors=40, p=2,
          weights='distance')
```

```
pk=reg3.predict(X_test)
```

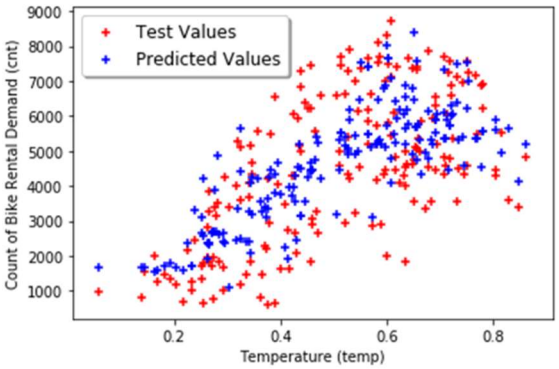
```
%matplotlib inline
plt.title('Plot of Temperature vs Count of Bike Rental Demand Test Set with Predictions using K-Neighbors Regressor')
plt.xlabel('Temperature (temp)')
plt.ylabel('Count of Bike Rental Demand (cnt)')
plt.scatter(X_test.temp,y_test,color='red',marker='+',label="Test Values")
plt.scatter(X_test.temp,pk,color='blue',marker='+',label="Predicted Values")
plt.legend(loc="best",shadow=True,fontsize='large')
```

Comparing Original values with Predictions using K-Neighbors Regression

Original Values



Predated Values using K-Neighbors Regression



Computing k-Fold Cross Validation for K-Neighbors Regression

```
from sklearn.model_selection import cross_val_score
accuracies = cross_val_score(estimator = reg3, X = X_train, y = y_train, cv = 10)
print('accuracies.mean:-')
print(accuracies.mean())
print('accuracies.std:-')
print(accuracies.std())
```

```
accuracies.mean:-
0.34437515205
accuracies.std:-
0.118343375294
```

```
accuracies
```

```
array([ 0.1799869 ,  0.32547038,  0.46088001,  0.37764477,  0.29097164,
        0.50295064,  0.28024602,  0.42245815,  0.13406257,  0.46908043])
```

Computing MAE and MSE for Random Forest Algorithm

```
from sklearn.metrics import mean_absolute_error
mean_absolute_error(y_test, reg3.predict(X_test))
```

```
1392.3630323760951
```

```
from sklearn.metrics import mean_squared_error
mean_squared_error(y_test, reg3.predict(X_test))
```

```
2943557.161200691
```

XGBoost Regressor

XGBoost is an optimized distributed gradient boosting library designed to be highly **efficient**, **flexible** and **portable**. It implements machine-learning algorithms under the Gradient Boosting framework. XGBoost provides a parallel tree boosting (also known as GBDT, GBM) that solve many data science problems in a fast and accurate way. The same code runs on major distributed environment (Hadoop, SGE, MPI) and can solve problems beyond billions of examples.

Gradient boosting

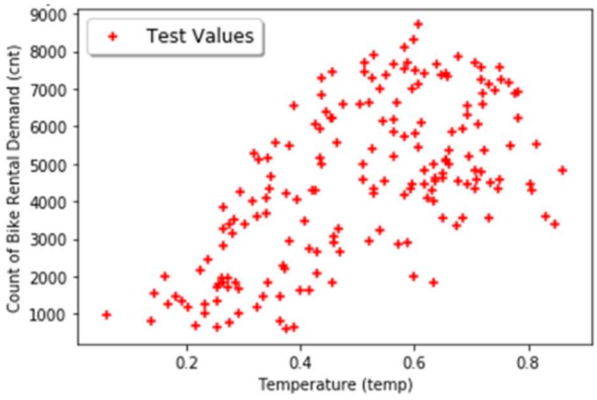
Gradient boosting is a machine learning technique for regression and classification problems, which produces a prediction model in the form of an ensemble of weak prediction models, typically decision trees. It builds the model in a stage-wise fashion like other boosting methods do, and it generalizes them by allowing optimization of an arbitrary differentiable loss function.The idea of gradient boosting originated in the observation by Leo Breiman that boosting can be interpreted as an optimization algorithm on a suitable cost function. Explicit regression gradient boosting algorithms were subsequently developed by Jerome H. Friedman, simultaneously with the more general functional gradient boosting perspective of Llew Mason, Jonathan Baxter, Peter Bartlett and Marcus Frean. The latter two papers introduced the view of boosting algorithms as iterative functional gradient descent algorithms. That is, algorithms that optimize a cost function over function space by iteratively choosing a function (weak hypothesis) that points in the negative gradient direction. This functional gradient view of boosting has led to the development of boosting algorithms in many areas of machine learning and statistics beyond regression and classification.

```
import xgboost as xgb
regxgb = xgb.XGBRegressor()
regxgb.fit(X_train, y_train)
pxgb=regxgb.predict(X_test)
```

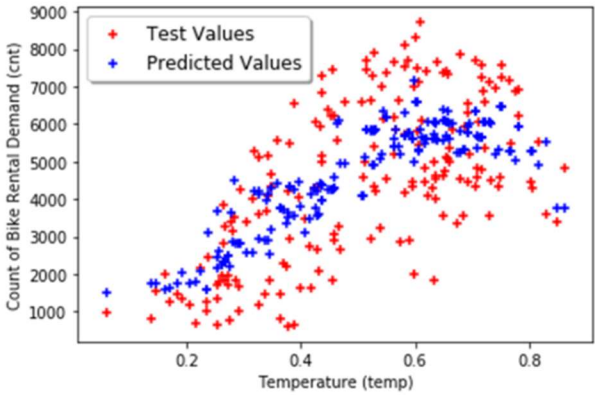
```
%matplotlib inline
plt.title('Plot of Temperature vs Count of Bike Rental Demand Test Set with Predictions using XGBoost XGBRegressor')
plt.xlabel('Temperature (temp)')
plt.ylabel('Count of Bike Rental Demand (cnt)')
plt.scatter(X_test.temp,y_test,color='red',marker='+',label="Test Values")
plt.scatter(X_test.temp,pxgb,color='blue',marker='+',label="Predicted Values")
plt.legend(loc="best",shadow=True,fontsize='large')
```

Comparing Original values with Predictions using XGBoost Regression

Original Values



Predated Values using XGBoost Regression



Computing k-Fold Cross Validation for XGBoost Regressor

```
from sklearn.model_selection import cross_val_score
accuracies = cross_val_score(estimator = regxgb, X = X_train, y = y_train, cv = 10)
print('accuracies.mean:-')
print(accuracies.mean())
print('accuracies.std:-')
print(accuracies.std())
```

```
accuracies.mean:-
0.450112549129
accuracies.std:-
0.0748791047743
```

```
accuracies
```

```
array([ 0.39683532,  0.41437503,  0.51352393,  0.4529075 ,  0.38177962,
        0.52422071,  0.55015607,  0.43628326,  0.30333918,  0.52770487])
```

Computing MAE and MSE for XGBoost Regressor Algorithm

```
from sklearn.metrics import mean_absolute_error
mean_absolute_error(y_test, regxgb.predict(X_test))
```

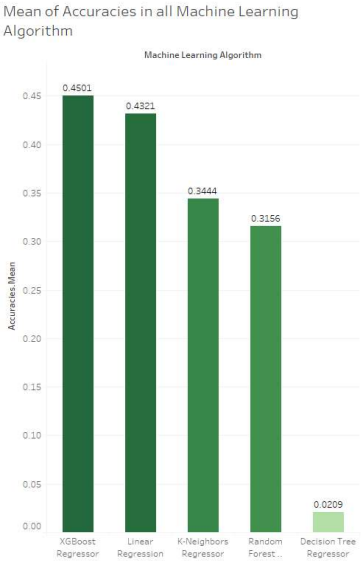
```
1296.0560256040812
```

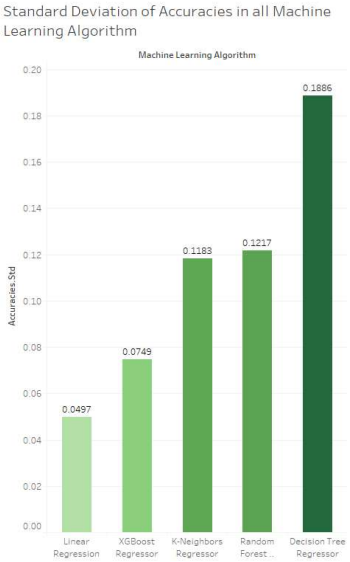
```
from sklearn.metrics import mean_squared_error
mean_squared_error(y_test, regxgb.predict(X_test))
```

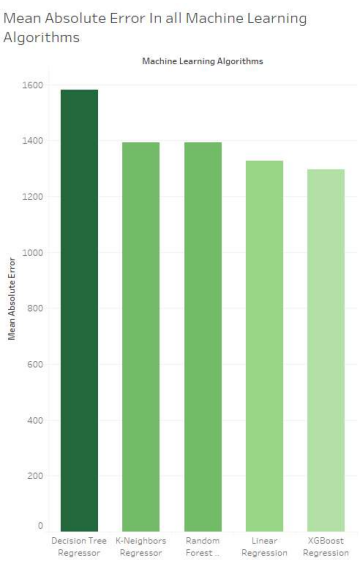
```
2395401.1068375651
```

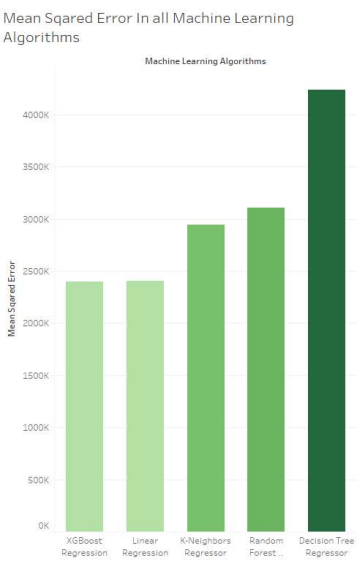
Conclusion

Comparing the Accuracies of all Machine Learning Algorithms obtained from k-Fold Cross Validation.









From the above graph we can see that XGBoost has the highest accuracy mean in comparison with other machine learning algorithms and Linear Regression has slightly lower accuracy mean then XGBoost. The XGBoost has lowest Mean Absolute Error and Mean Squared Error. The XGBoost also have very low standard deviation mean in comparison with other machine learning algorithms this makes it most efficient for predicting the bike rental demand count with respect to Temperature and Seasons. However, Linear Regression will also work efficiently since it has the lowest standard deviation.

The Decision Tree has highest accuracy standard deviation and lowest accuracy mean in comparison with other machine learning algorithms which makes it most worst predictor for bike rental demand count with respect to Temperature and Seasons.

The predictions efficiency can be improved by using Artificial Neural Networks and Convolutional Neural Networks. But, due limitation of time and contains of course I was able to predict the data up to 45% accuracy with XGBoost.

Appendices

Python Code Ran on Jupyter Notebook:-

```
import matplotlib.pyplot as plt

import os

os.chdir("E:\\R WorkStation\\Python WorkStation")

os.getcwd()

import pandas as pd

df= pd.read_csv("day.csv", sep = ',')

df

df.shape

df=df.drop(columns=['atemp', 'casual', 'registered'])

df.isnull().any().any()

df.head(1)

df.corr()

df.season=pd.Categorical(df.season)

df.yr=pd.Categorical(df.yr)

df.mnth=pd.Categorical(df.mnth)

df.holiday=pd.Categorical(df.holiday)

df.weekday=pd.Categorical(df.weekday)

df.workingday=pd.Categorical(df.workingday)

df.weathersit=pd.Categorical(df.weathersit)

df.temp=pd.to_numeric(df.temp)

df.hum=pd.to_numeric(df.hum)

df.windspeed=pd.to_numeric(df.windspeed)

df.cnt=pd.to_numeric(df.cnt)

dum = pd.get_dummies(df.season)

dum
```

```
X = df[['temp']]
X= pd.concat([X,dum],axis='columns')
y = df['cnt']
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state = 0)
%matplotlib inline
plt.title('Plot of Temperature vs Count of Bike Rental Demand Test Set without any Prediction')
plt.xlabel('Temperature (temp)')
plt.ylabel('Count of Bike Rental Demand (cnt)')
plt.scatter(X_test.temp,y_test,color='red',marker='+',label="Test Values")
plt.legend(loc="best",shadow=True,fontsize='large')
from sklearn import linear_model
reg=linear_model.LinearRegression(n_jobs=5,normalize=tuple)
reg.fit(X_train, y_train)
pl=reg.predict(X_test)
%matplotlib inline
plt.title('Plot of Temperature vs Count of Bike Rental Demand Test Set with Predictions using Linear Regression')
plt.xlabel('Temperature (temp)')
plt.ylabel('Count of Bike Rental Demand (cnt)')
plt.scatter(X_test.temp,y_test,color='red',marker='+',label="Test Values")
plt.scatter(X_test.temp,pl,color='blue',marker='+',label="Predicted Values")
plt.legend(loc="best",shadow=True,fontsize='large')
from sklearn.model_selection import cross_val_score
accuracies = cross_val_score(estimator = reg, X = X_train, y = y_train, cv = 10)
print('accuracies.mean:-')
print(accuracies.mean())
print('accuracies.std:-')
print(accuracies.std())
```

```
accuracies
from sklearn.metrics import mean_absolute_error
mean_absolute_error(y_test, reg.predict(X_test))
from sklearn.metrics import mean_squared_error
mean_squared_error(y_test, reg.predict(X_test))
from sklearn.tree import DecisionTreeRegressor
reg1 = DecisionTreeRegressor(random_state = 0)
reg1.fit(X_train, y_train)
pdt=reg1.predict(X_test)
%matplotlib inline
plt.title('Plot of Temperature vs Count of Bike Rental Demand Test Set with Predictions using Decision Tree Regressor')
plt.xlabel('Temperature (temp)')
plt.ylabel('Count of Bike Rental Demand (cnt)')
plt.scatter(X_test.temp,y_test,color='red',marker='+',label="Test Values")
plt.scatter(X_test.temp,pdt,color='blue',marker='+',label="Predicted Values")
plt.legend(loc="best",shadow=True,fontsize='large')
from sklearn.model_selection import cross_val_score
accuracies = cross_val_score(estimator = reg1, X = X_train, y = y_train, cv = 10)
print('accuracies.mean:-')
print(accuracies.mean())
print('accuracies.std:-')
print(accuracies.std())\
accuracies
from sklearn.metrics import mean_absolute_error
mean_absolute_error(y_test, reg1.predict(X_test))
from sklearn.metrics import mean_squared_error
mean_squared_error(y_test, reg1.predict(X_test))
```

```
from sklearn.ensemble import RandomForestRegressor

reg2 = RandomForestRegressor(n_estimators = 350, random_state = 0)

reg2.fit(X_train, y_train)

prf=reg2.predict(X_test)

matplotlib inline

plt.title('Plot of Temperature vs Count of Bike Rental Demand Test Set with Predictions using Random Forest Regressor')

plt.xlabel('Temperature (temp)')

plt.ylabel('Count of Bike Rental Demand (cnt)')

plt.scatter(X_test.temp,y_test,color='red',marker='+',label="Test Values")

plt.scatter(X_test.temp,prf,color='blue',marker='+',label="Predicted Values")

plt.legend(loc="best",shadow=True,fontsize='large')

from sklearn.model_selection import cross_val_score

accuracies = cross_val_score(estimator = reg2, X = X_train, y = y_train, cv = 10)

print('accuracies.mean:-')

print(accuracies.mean())

print('accuracies.std:-')

print(accuracies.std())

accuracies

from sklearn.metrics import mean_absolute_error

mean_absolute_error(y_test, reg2.predict(X_test))

from sklearn.metrics import mean_squared_error

mean_squared_error(y_test, reg2.predict(X_test))

from sklearn.neighbors import KNeighborsRegressor

reg3 = KNeighborsRegressor(n_neighbors = 40, metric = 'minkowski', p = 2, weights='distance', n_jobs=10)

reg3.fit(X_train, y_train)
```

```
%matplotlib inline
plt.title('Plot of Temperature vs Count of Bike Rental Demand Test Set with Predictions using K-Neighbors Regressor')
plt.xlabel('Temperature (temp)')
plt.ylabel('Count of Bike Rental Demand (cnt)')
plt.scatter(X_test.temp,y_test,color='red',marker='+',label="Test Values")
plt.scatter(X_test.temp,pk,color='blue',marker='+',label="Predicted Values")
plt.legend(loc="best",shadow=True,fontsize='large')
from sklearn.model_selection import cross_val_score
accuracies = cross_val_score(estimator = reg3, X = X_train, y = y_train, cv = 10)
print('accuracies.mean:-')
print(accuracies.mean())
print('accuracies.std:-')
print(accuracies.std())
accuracies
from sklearn.metrics import mean_absolute_error
mean_absolute_error(y_test, reg3.predict(X_test))
from sklearn.metrics import mean_squared_error
mean_squared_error(y_test, reg3.predict(X_test))
import xgboost as xgb
regxgb = xgb.XGBRegressor(n_estimators=100,earning_rate=2)
regxgb.fit(X_train, y_train)
pxgb=regxgb.predict(X_test)
```

```
%matplotlib inline
plt.title('Plot of Temperature vs Count of Bike Rental Demand Test Set with Predictions using XGBoost XGBRegressor')
plt.xlabel('Temperature (temp)')
plt.ylabel('Count of Bike Rental Demand (cnt)')
plt.scatter(X_test.temp,y_test,color='red',marker='+',label="Test Values")
plt.scatter(X_test.temp,pxgb,color='blue',marker='+',label="Predicted Values")
plt.legend(loc="best",shadow=True,fontsize='large')
from sklearn.model_selection import cross_val_score
accuracies = cross_val_score(estimator = regxgb, X = X_train, y = y_train, cv = 10)
print('accuracies.mean:-')
print(accuracies.mean())
print('accuracies.std:-')
print(accuracies.std())
accuracies
from sklearn.metrics import mean_absolute_error
mean_absolute_error(y_test, regxgb.predict(X_test))
from sklearn.metrics import mean_squared_error
mean_squared_error(y_test, regxgb.predict(X_test))
```