Bike Renting

(Prediction of bike rental counts)

-Saket Kishore

Introduction

The objective of this project is to create a model which predicts the bike rental count based on various factors such as season, humidity, weather situation etc.

Through various methods, the data will be explored, visualized and cleaned followed by which a model will be developed. The model will be tested and its stability will be checked before finalizing.

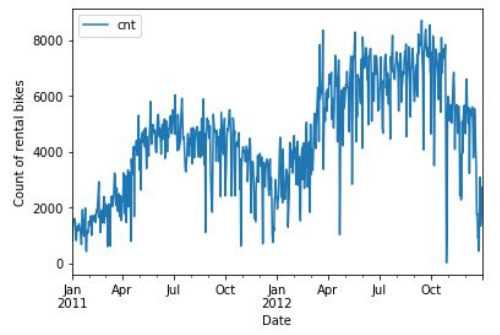
Table Of Contents

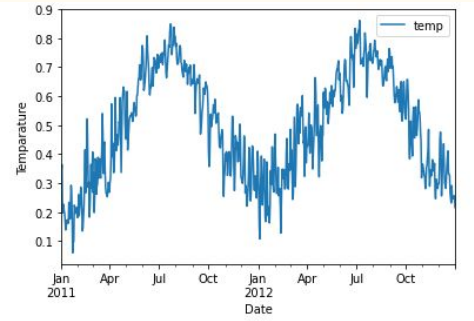
Exploratory Data Analysis

At a glance, we observe that the dataset can potentially be forecasted through a time-series method or a regression method. Let us check which one is suitable.

	instant	dteday	season	yr	mnth	holiday	weekday	workingday	weathersit	temp	atemp	hum	windspeed	casual	registered	cnt
0	1	2011-01-01	1	0	1	0	6	0	2	0.344167	0.363625	0.805833	0.160446	331	654	985
1	2	2011-01-02	1	0	1	0	0	0	2	0.363478	0.353739	0.696087	0.248539	131	670	801
2	3	2011-01-03	1	0	1	0	1	1	1	0.196364	0.189405	0.437273	0.248309	120	1229	1349
3	4	2011-01-04	1	0	1	0	2	1	1	0.200000	0.212122	0.590435	0.160296	108	1454	1562
4	5	2011-01-05	1	0	1	0	3	1	1	0.226957	0.229270	0.436957	0.186900	82	1518	1600

The distribution of the target variable across time has a trend as well as seasonality.





As expected, the distribution of temperature along time is also seasonal in nature. Later in the project we see that temperature proves to be a very important feature in the prediction of the target variable.

So, if we solve this using SARIMAX, we can use temperature as an exogenous variable to support the model thereby increasing its accuracy.

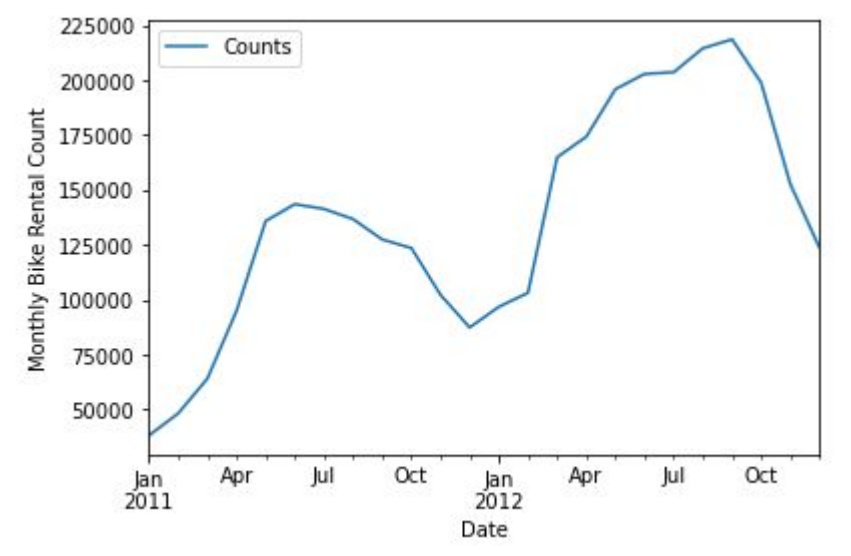
The records we have contains daily data. To use a time series model on this, we must convert this to monthly data. This is necessary because there should be some correlation between two consecutive timestamps with respect to the target variable. This means that the count of bike rentals from a month should be affecting the count of bike rentals from the consecutive month for the model to work. If we leave the data as it is then 2 consecutive days is very small time frame for a timestamp to have any correlation with its consecutive timestamp.

The dataset after converting it from daily to monthly :

date	Counts
2011-01-01	38189
2011-02-01	48215
2011-03-01	64045
2011-04-01	94870
2011-05-01	135821
2011-06-01	143512
2011-07-01	141341
2011-08-01	136691

Now the dataset has 24 records. One for each month and the daily counts for each month have been merged.

Plot after converting daily data to monthly data :



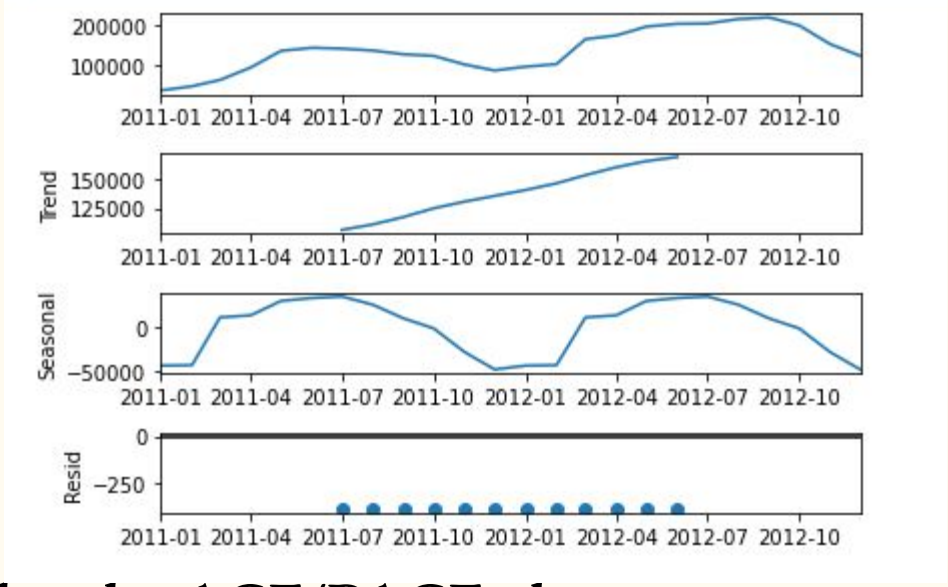
Just as observed before, there is trend and seasonality in the dataset. However the graph is not very well defined. This is probably because of lack of data points. We only have 24 months of data.

Let us remove the trend and seasonality from the data and check if there is really any predictive power in dataset. To do this, we used **seasonal_decompose** from the **statsmodel** library. We use ACF & PACF plots and the Augmented Dickey Fuller Test, to see if there is any local component the in the dataset or if it is only noise.

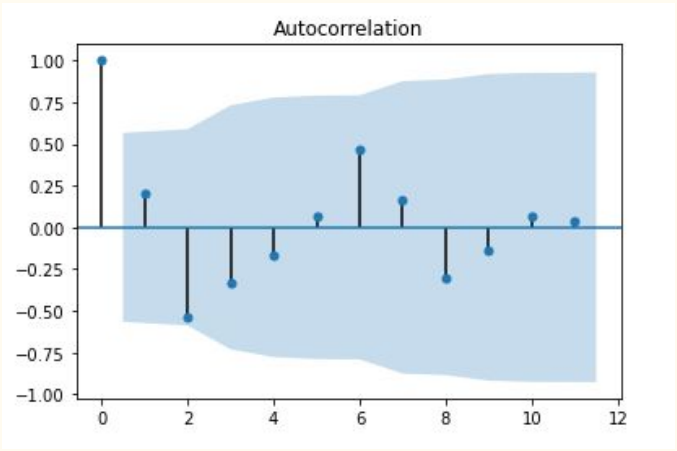
Decomposing/Separating the seasonality and trend from the data :

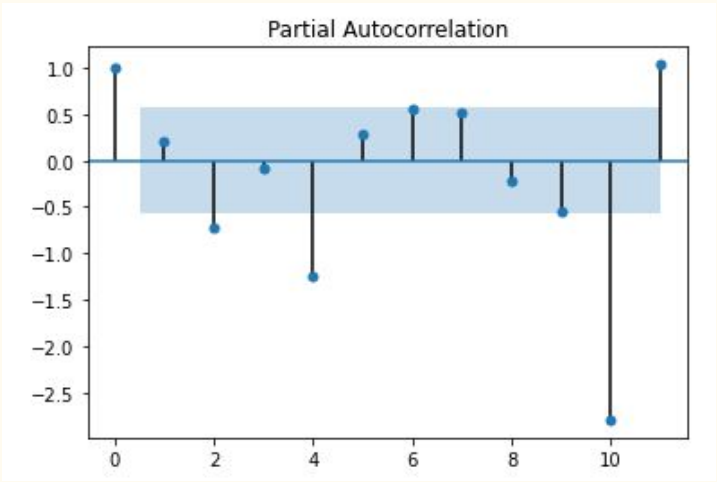
```
from matplotlib import pyplot
result = seasonal_decompose(bike_rental, model='additive')
result.plot()
%%pypylot.show()
```

```
UsageError: Line magic function `%%pypylot.show()` not found
```



Now we plot the ACF/PACF plots :





The 1st lag itself for both the plots is within the confidence intervals indicating the residual mostly consists of noise. We do the augmented dickey fuller test to further confirm this.

Augmented Dickey Fuller Test :

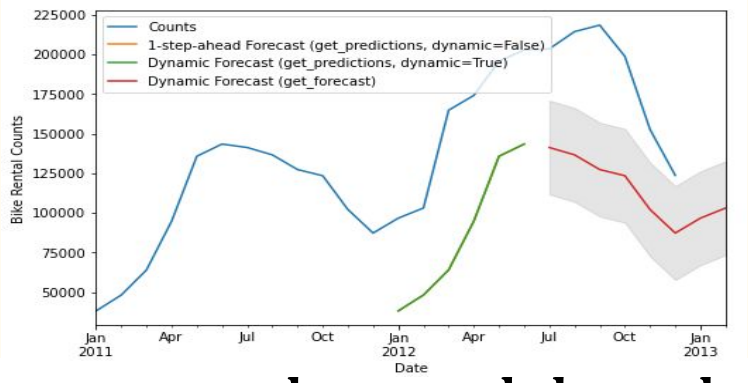
p-value > 0.05: Fail to reject the null hypothesis (H0), the data has a unit root and is non-stationary.

p-value <= 0.05: Reject the null hypothesis (H0), the data does not have a unit root and is stationary.

We get a p-value of 0.41

Thus, we fail to reject the null hypothesis indicating the data is non-stationary. Thus, there is some local component in the data and its not all noise.

However on proceeding with SARIMAX model, the predictions were not good enough as seen in the plot.



The blue line represents the actual data, the green line represents the prediction on the training data and the red line represents forecast on test data and a few months beyond it. Thus, we do not use time-series method to solve this problem.

So let us start solving this problem using a regression technique.

Here is the dataset :

	instant	dteday	season	yr	mnth	holiday	weekday	workingday	weathersit	temp	atemp	hum	windspeed	casual	registered	cnt
0	1	2011-01-01	1	0	1	0	6	0	2	0.344167	0.363625	0.805833	0.160446	331	654	985
1	2	2011-01-02	1	0	1	0	0	0	2	0.363478	0.353739	0.696087	0.248539	131	670	801
2	3	2011-01-03	1	0	1	0	1	1	1	0.196364	0.189405	0.437273	0.248309	120	1229	1349
3	4	2011-01-04	1	0	1	0	2	1	1	0.200000	0.212122	0.590435	0.160296	108	1454	1562
4	5	2011-01-05	1	0	1	0	3	1	1	0.226957	0.229270	0.436957	0.186900	82	1518	1600

From the start on we see a few variables which will not be useful in model building so we drop those first.

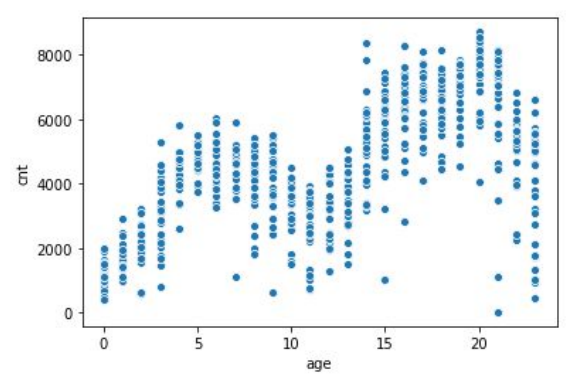
The variable **instant** is just a counter variable. The variables **casual** and **registered** cannot be used because the target variable is the sum total of these two variables and if we already knew these two we could just add them up and there would be no requirement of a machine learning model.

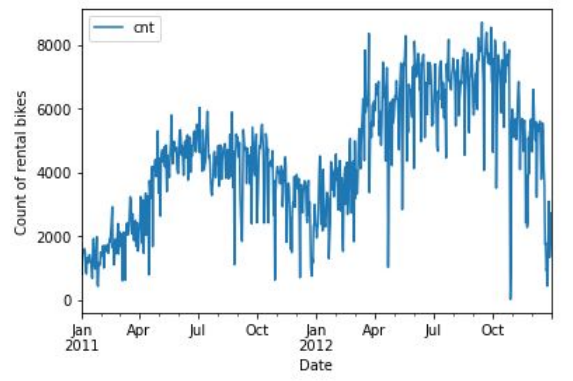
The variable **dteday** is a date-time type variable and would make the problem complex as it cannot directly be fed into a model. To solve this problem we analyse what information this variable is carrying. The most vital part of this variable is that it consists of the age of the company.

Thus, we extract the information of the company's age from this variable and store it in a new variable named 'age'.

The variable age includes the age of the company in months.

As expected, the relationship between age and the target variable has the same pattern as the relationship between dteday and the target variables:





Now we delete the variables - instant, registered, casual and dteday.

Our new dataset :

```
bike_rental = bike_rental.drop(["instant", "dteday", "casual", "registered"], axis=1)
```

```
bike_rental.tail(5)
```

	season	yr	mnth	holiday	weekday	workingday	weathersit	temp	atemp	hum	windspeed	cnt	age
726	1	1	12	0	4	1	2	0.254167	0.226642	0.652917	0.350133	2114	23
727	1	1	12	0	5	1	2	0.253333	0.255046	0.590000	0.155471	3095	23
728	1	1	12	0	6	0	2	0.253333	0.242400	0.752917	0.124383	1341	23
729	1	1	12	0	0	0	1	0.255833	0.231700	0.483333	0.350754	1796	23
730	1	1	12	0	1	1	2	0.215833	0.223487	0.577500	0.154846	2729	23

Missing Value Analysis

There we no missing values present in any variable in our dataset.

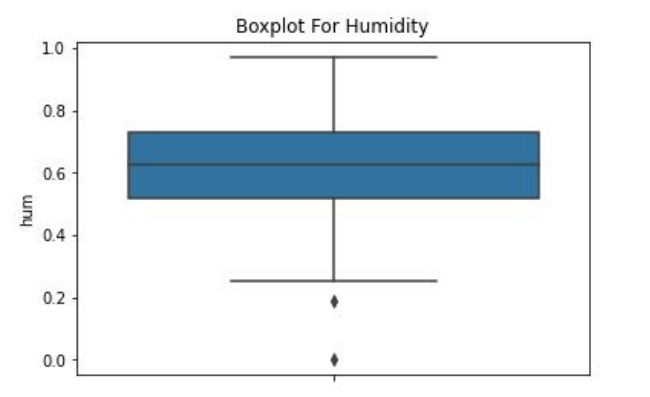
```
pd.DataFrame(bike_rental.isnull().sum())
```

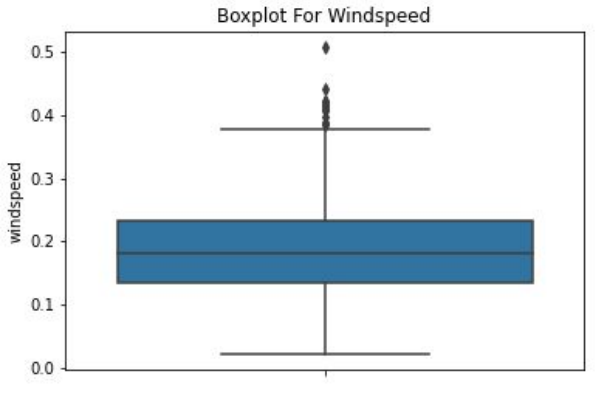
	0
season	0
yr	0
mnth	0
holiday	0
weekday	0
workingday	0
weathersit	0
temp	0
atemp	0
hum	0
windspeed	0
cnt	0
age	0

Outlier Analysis

Outliers are the values in a variables which are either greater than 75 percentile of the data or smaller than 25 percentile of that data.

Here, we have 2 outliers in humidity variable and 13 outliers in windspeed variable.



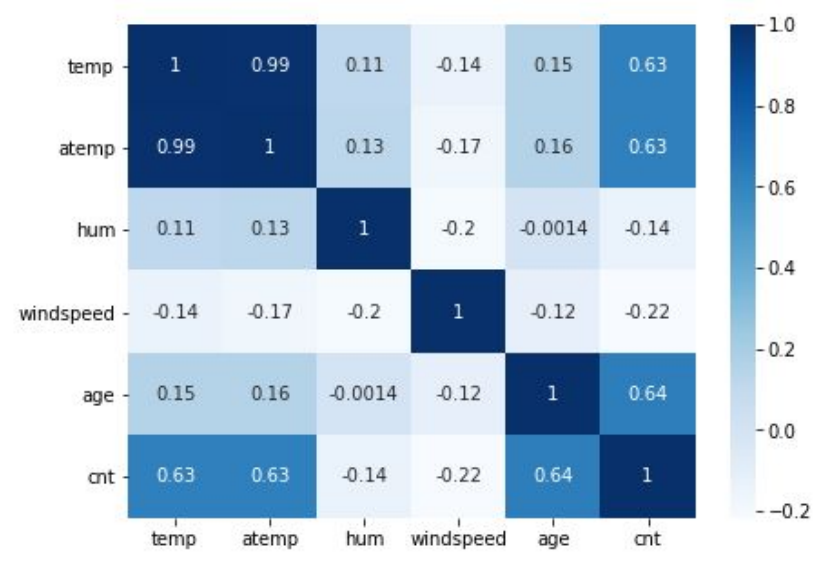


15 rows are less than 2 percent of the total data we have. So we decided to drop the rows which contain the outliers.

Our dataset now contains 717 records which means we dropped 14 records.

Feature Selection

We plot a pearson's correlation plot to check the correlation between all the numerical variables in the dataset.



The variable **atemp** and **temp** have the same correlation coefficient with respect to the target variable however **atemp** is more negatively correlated to **windspeed** than **temp** and more positively correlated with **humidity** than **temp.** This is probably because tempature feels lower when windspeed is high and higher when humidity is high.

So we may drop the variable **atemp** because we already have actual temparature, humidity and windspeed as variables in our dataset.

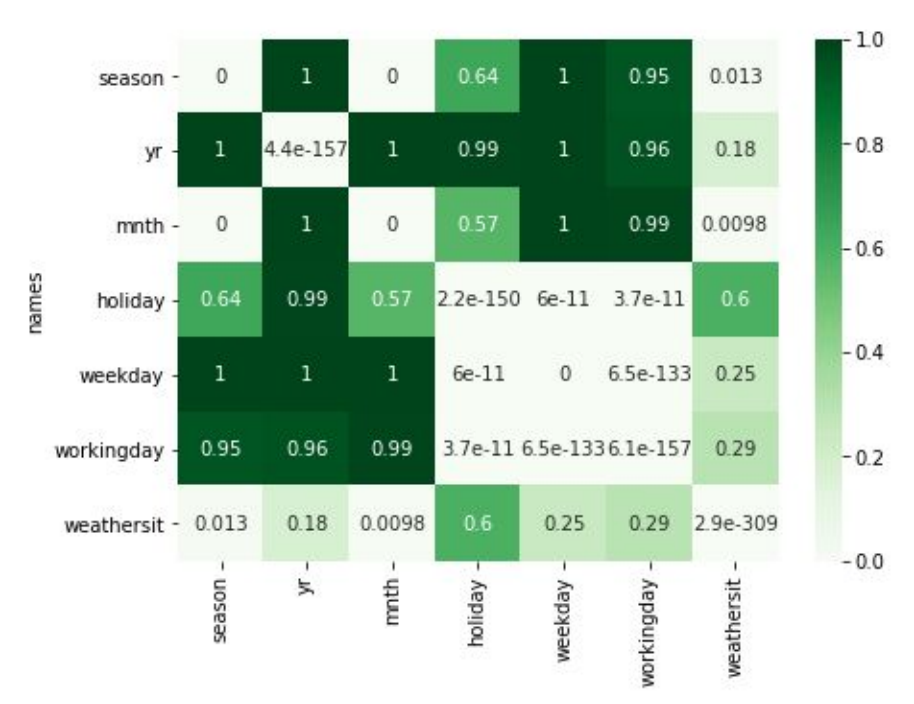
Chi-Square test of independence :

Chi-Square test is a hypothesis testing method to test the independence between categorical variables.

Null Hypothesis(H0) : The variables are independent

Alternate Hypothesis(H1) : The variables are dependent.

This is not a traditional method, but to keep things simple and easily distinguishable, we use nested loops to extract the p-values from the chi-square test and then store them in a dataframe. We then improvise the dataframe a little so that a heatmap can be plot out of it just like we do for pearson's correlations.



The deep green parts of the heatmap represent independence of variables and the white parts represent highly dependent variables. We can also see the p-values easily from the heatmap.

We see that **season, month and weathersit** are highly dependent on each other, **holiday, weekday and workingday** are highly dependent on each other.

We use ExtraTreesRegressor from scikit-learn library to check the feature importance in the prediction of the target variable.

	Feature	importance
0	yr	0.257874
1	temp	0.241403
2	age	0.158705
3	season	0.124579
4	weathersit	0.063312
5	mnth	0.059354
6	hum	0.034446
7	windspeed	0.023372
8	weekday	0.018551
9	workingday	0.012419
10	holiday	0.005984

We see that season is a very important feature, month and weathersit are almost equally important however month has 12 categories in it which may cause a curse of dimensionality problem after one hot encoding is conducted.

We see that holiday is the least important feature, workingday and holiday are obviously nearly carrying the same information. Weekday may become a problem later because it has 7 categories but since its slightly more important, we won't remove it until some further analysis clarifies it for us.

Thus, we drop the variables **mnth** and **holiday** after Chi-Square test of independence.

Variance Inflation Factor(VIF) Test :

In a vif test, if a variable has VIF greater than 5, then it has high correlation with some other variable. As a general rule, the variable with the higher VIF should be removed

Before conducting the VIF test, proper datatype should be assigned to categorical variables so that VIF for each category can be calculated.

We see that **workingday** has a higher VIF value than any category of weekday. The variables **age** and **yr** are probably highly correlated because they are basically carrying the same information, **age** is the age of company in months whereas **yr** is the age of company in years. The variable **age** has a higher VIF value than **yr**. Also, yr was more important than age and weekday was more important than workingday when feature importance was calculated. Thus, we drop the variables **workingday** and **age**.

The VIF values before and after dropping the variables:

	VIF	features
0	64.068643	Intercept
1	2.833669	season[T.2]
2	5.934553	season[T.3]
3	4.929238	season[T.4]
4	11.688286	yr[T.1]
5	5.131975	weekday[T.1]
6	6.369954	weekday[T.2]
7	6.371789	weekday[T.3]
8	6.194917	weekday[T.4]
9	6.292548	weekday[T.5]
10	1.717949	weekday[T.6]
11	8.335645	workingday[T.1]
12	1.675942	weathersit[T.2]
13	1.414428	weathersit[T.3]
14	3.509252	temp
15	2.057216	hum
16	1.173403	windspeed
17	14.056867	age

	VIF	features
0	63.763459	Intercept
1	2.704580	season[T.2]
2	4.785553	season[T.3]
3	1.880710	season[T.4]
4	1.037349	yr[T.1]
5	1.712666	weekday[T.1]
6	1.727481	weekday[T.2]
7	1.735787	weekday[T.3]
8	1.715374	weekday[T.4]
9	1.730637	weekday[T.5]
10	1.717875	weekday[T.6]
11	1.673127	weathersit[T.2]
12	1.413113	weathersit[T.3]
13	3.508908	temp
14	2.038475	hum
15	1.172797	windspeed

After various steps of testing and elimination, we removed the variables which could have caused multi-collinearity and selected the following features for model development :

season, yr, weekday, weathersit, temp, hum and windspeed.

The dataset now looks like this :

	season	yr	weekday	weathersit	temp	hum	windspeed	cnt
0	1	0	6	2	0.344167	0.805833	0.160446	985.0
1	1	0	0	2	0.363478	0.696087	0.248539	801.0
2	1	0	1	1	0.196364	0.437273	0.248309	1349.0
3	1	0	2	1	0.200000	0.590435	0.160296	1562.0
4	1	0	3	1	0.226957	0.436957	0.186900	1600.0

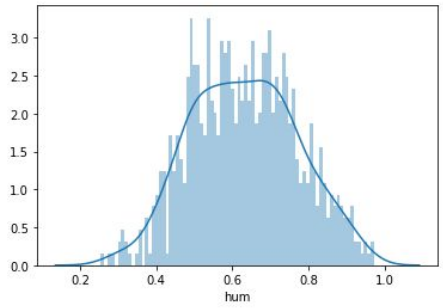
Feature Scaling

We have 3 numerical variables in the dataset now - temp, humidity and windspeed.

To check if these variables are normally distributed, we plot distplots and conduct shapiro test which is a hypothesis testing method.

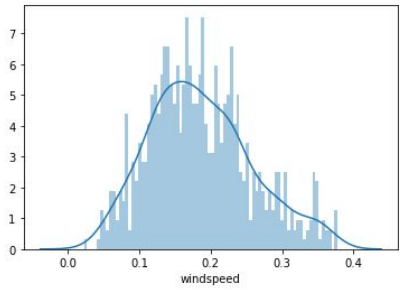
```
sns.distplot(bike_rental['hum'],bins =80)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1456e3e0288>
```



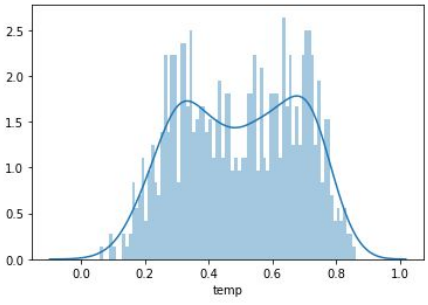

```
sns.distplot(bike_rental['windspeed'],bins =80)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1456e5338c8>
```




```
sns.distplot(bike_rental['temp'],bins =80)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1456e2793c8>
```



Shapiro Test :

Null Hypothesis(H0) : Normally distributed.

Alternate Hypothesis(H1): Not normally distributed.

p <= 0.05: reject H0, not normal.

p > 0.05: fail to reject H0, normal.

```
from scipy import stats

for i in num:
    shapiro_test = stats.shapiro(bike_rental[i])
    print(i)
    print(shapiro_test)
```

```
temp
(0.9643257260322571, 3.325623196964722e-12)
hum
(0.9937424659729004, 0.004474991001188755)
windspeed
(0.9807030558586121, 4.064008152226961e-08)
```

We reject the null hypothesis for all the numerical variables as p-value is smaller than 0.05 for all of them.

Data after performing normalization :

```
bike_rental.head()
```

	season	yr	weekday	weathersit	temp	hum	windspeed	cnt
0	1	0	6	2	0.355170	0.767981	0.388102	985.0
1	1	0	0	2	0.379232	0.615202	0.635752	801.0
2	1	0	1	1	0.171000	0.254904	0.635105	1349.0
3	1	0	2	1	0.175530	0.468123	0.387681	1562.0
4	1	0	3	1	0.209120	0.254464	0.462471	1600.0

Before moving on to Model Development, we do one hot encoding for all the categorical variables.

Also, since a few columns had been deleted due to presence of outliers, the index must be reset.

Before resetting the index :

```
bike_rental.tail()
```

	temp	hum	windspeed	cnt	season_1	season_2	season_3	season_4	yr_0	yr_1	weekday_0	weekday_1	weekday_2	weekday_3	weekday_
726	0.243025	0.555105	0.921356	2114.0	1	0	0	0	0	1	0	0	0	0	
727	0.241986	0.467517	0.374116	3095.0	1	0	0	0	0	1	0	0	0	0	
728	0.241986	0.694316	0.286721	1341.0	1	0	0	0	0	1	0	0	0	0	
729	0.245101	0.319025	0.923102	1796.0	1	0	0	0	0	1	1	0	0	0	
730	0.195259	0.450116	0.372359	2729.0	1	0	0	0	0	1	0	1	0	0	

After resetting the index :

```
bike_rental.tail()
```

	temp	hum	windspeed	cnt	season_1	season_2	season_3	season_4	yr_0	yr_1	weekday_0	weekday_1	weekday_2	weekday_3	weekday_
712	0.243025	0.555105	0.921356	2114.0	1	0	0	0	0	1	0	0	0	0	
713	0.241986	0.467517	0.374116	3095.0	1	0	0	0	0	1	0	0	0	0	
714	0.241986	0.694316	0.286721	1341.0	1	0	0	0	0	1	0	0	0	0	
715	0.245101	0.319025	0.923102	1796.0	1	0	0	0	0	1	1	0	0	0	
716	0.195259	0.450116	0.372359	2729.0	1	0	0	0	0	1	0	1	0	0	

Now we can move on to the model development phase.

Model Development

We will develop six models :

1. Linear Regression
2. Ridge Regression
3. Lasso Regression
4. Decision Tree
5. Random Forest
6. XGBoost Regressor

The models will be tested on 3 metrics :

1. R-Squared Error
2. Adjusted R-Squared Error
3. MAPE Percentage

Hyperparameter tuning will be done on the model which performs the best and then the model will be finalized.

Linear Regression

A linear regression model creates a best fit line such that the cost function or the summation of all the residuals is minimal.

Error Metrics	Performance
R-Squared	0.86
Adjusted R-Squared	0.84
MAPE	17.06

Ridge Regression

A ridge regression is similar to a linear regression but the cost function also includes squared of the slopes. The model tries to create a regression line with minimum cost function. This means that a ridge regression penalizes higher values of slope. This is done to reduce overfitting.

Error Metrics	Performance
R-Squared	0.86
Adjusted R-Squared	0.84
MAPE	17.8

Lasso Regression

The lasso regression works very similar to a ridge regression but instead of using the squared of slopes, it uses the magnitude of slopes or a modulus of slopes. This makes sure that in the final regression line selected, slope of all the variables are included. They might shrink a lot but never become zero.

Error Metrics	Performance
R-Squared	0.86
Adjusted R-Squared	0.84
MAPE	17.07

Decision Tree

The decision tree tries various thresholds for every variable and calculates the sum of squared residuals at each step and then picks the threshold with the minimum sum of squared residual value. This is done for every variable and then the one with the least SSR becomes the root node from where the tree is split in a similar fashion again to find thresholds. The tree stops splitting after a node is left with only a certain number of observations, typically 20. After which the node becomes a leaf and tree is not split anymore. This is done to prevent overfitting.

Error Metrics	Performance
R-Squared	0.86
Adjusted R-Squared	0.84
MAPE	19.57

Random Forest

In a random forest regressor, multiple decision trees are created to produce multiple continuous output and the mean of all these continuous outputs is finalized.

Error Metrics	Performance
R-Squared	0.92
Adjusted R-Squared	0.90
MAPE	15.85

XGBoost

An XGBoost algorithm selects a node which is split based on a threshold. The similarity scores for the nodes and leaves is calculated using which the gain is calculated. The tree is is adjusted for different threshold values until the highest gain value is extracted after which the threshold value is finalized for every node.

Error Metrics	Performance
R-Squared	0.86
Adjusted R-Squared	0.84
MAPE	17.07

Model Evaluation

1 - Linear Regression
2 - Ridge Regression
3 - Lasso Regression
4 - Decision Tree
5 - Random Forest
6 - XGBoost Regressor

Error Metrics	1	2	3	4	5	6
R-Squared	0.86	0.86	0.86	0.86	0.92	0.86
Adjusted R-Squared	0.84	0.84	0.84	0.84	0.90	0.84
MAPE	17.06	17.18	17.07	19.57	15.85	17.07

Hyperparameter Tuning

As random forest performed the best, we use hyperparameter tuning to see if further improvements are possible.

RandomizedSearchCV was used for the task and the following parameters were found to be the best :

```
rf_random.best_params_
```

```
{'n_estimators': 600,
 'min_samples_split': 2,
 'min_samples_leaf': 1,
 'max_depth': 100}
```

On training the model with these parameters, the performance slightly decreased but the change was negligible.

Before	After
Adjusted R squared : 0.90	Adjusted R squared : 0.90
R-Squared : 0.92	R-Squared : 0.92
MAPE : 15.85	MAPE : 16.04

Cross-Validation

Cross validation was performed with R-Squared as the scoring metric to check the model stability.

The model was performing well consistently.

```
cross_val_score(regressor_RF, X_train, y_train, cv=10, scoring='r2')
```

```
array([0.86920989, 0.85389197, 0.76559922, 0.80902693, 0.90635427,
       0.80575107, 0.91166712, 0.84529339, 0.89906403, 0.85358994])
```

```
cross_val_score(regressor_RF, X_train, y_train, cv=10, scoring='r2').mean()
```

```
0.8508155740763005
```

Conclusion

The dataset had 731 records. Various data pre-processing techniques were used to clean the data and select features for model development. Multiple models were developed out of which the Random Forest performed the best. Hyperparameter tuning was done on Random Forest as an attempt to further improve the performance. Cross-Validation was performed to check model stability, the model was performing well consistently.

Thus, **Random Forest** with the default parameters was chosen as the final model.