Date: 05/05/2019

Bike Rental Project Report

Submitted by Purushottam Potnuru

Contents

Chapter 1

1. Introduction

1.1 Problem Statement

Our objective of this Bike Rental Project is to **predict the bike rental count on daily basis, considering the environmental and seasonal settings.**

The predicted values will help the business to meet the demand on particular days with required supply i.e. supply meets demand.

1.2 Dataset

Image 1

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

Image 2

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

The given dataset contains 16 variables and 731 observations. The cnt is the target variable and remaining are the independent variables.

As said above, our primary focus is to determine the count for future test cases using the above shown dataset.

1.3 Exploratory Data Analysis

In Exploratory Data Analysis, we go through different things, like:

- **Brainstorming** – Here, all the team members (if you have a group working together) or only you alone, will actually draw a rough sketch which talks about – the steps you are going to follow to achieve your given objective.

- **Defining Problem Statement** – Although, in this project we are given with problem statement to work with, however, when dealing with real time scenarios, we are sometimes only given with limited information, and the client may ask us in reverse – just find some insights from the given data, to grow my business.

  In that case, we are responsible to define the problem statement by ourselves. We have to understand the complete in and out operations of the client's business and take decisions accordingly.

- Knowing the types of variables in the given dataset, whether they are factor, character or numeric.

- Changing and removing datatypes if required. In our project, we removed four variables as we are convinced, they won't be useful while developing model.

Chapter 2

2. Methodology

Now, we have the dataset and also, we discussed about Exploratory Data Analysis, let's talk about the **methodology** we are going to follow to achieve our goal.

We will be going through:

- Pre-processing which includes missing value analysis, outlier analysis, feature selection and feature scaling.
- Model development, where we will choose what machine learning algorithms to apply.

2.1 Pre-processing

In pre-processing, we actually apply few techniques like missing value analysis, outlier analysis, feature selection, feature scaling.

Why we do that? Well, actually we never get a structured data to work with. Always messy data is handed to us, and we need to clean that data.

The data may have many observations (rows in dataset), where values in few fields will be absent. We can also say, there may be some inconsistent values in a variable (column in dataset), when compared with other variables.

When we go for model development, we should have a structured dataset. We can't go forward for model development, if we don't apply pre-processing techniques on data and convert it into structured format.

2.1.1 Missing Value Analysis

Missing value analysis, as the name suggests, we face with situations, where we are given with dataset, and we have missing values in the observations.

The reason why we have missing values may be plenty, like human error, the user didn't want to share his complete information, the one who was supposed to fill the data may didn't work properly.

But, as discussed earlier, we have to give structured data to the model. In order to achieve that, we perform missing value analysis on top of the data to clean the data, to transform the data from unstructured format to structured format.

We give a line of code and it gives us the total number of missing values in each column. Later, we impute those missing values using mean, median or KNN method for numerical variables or using mode for categorical variables.

After imputing the missing values, we proceed further with outlier analysis. In our project, we don't have any missing values, so we didn't apply it on top of our dataset.

Image 3

```
season         0
year           0
month          0
holiday        0
weekday        0
workingday     0
weather        0
temperature    0
atemp          0
humidity       0
windspeed      0
count          0
dtype: int64
```

We don't have any missing vlaues in the dataset

Another important thing is, we apply missing value techniques only on numerical variables.

2.1.2 Outlier Analysis

Outliers may be defined as the inconsistent values in a variable. For example, a = 1,2,3,4, 20. In a, 20 is inconsistent, in terms of mean. Another important thing is, we apply missing value techniques only on numerical variables.

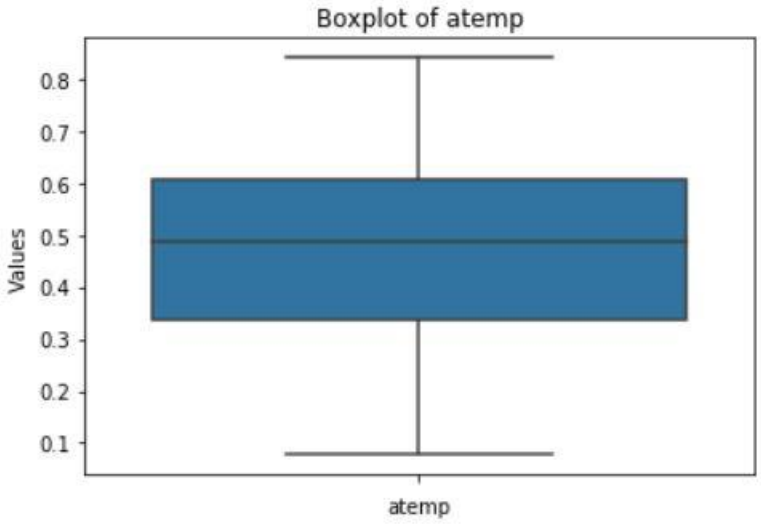
Outliers are used for fraud detection. Let's say, in one bank account, consistently, we observe an amount which ranges from 50,000 to 1,00,000, but in one case 10 lakhs gets deposited. In that case, simply using outlier technique, we can get the inconsistent values.

In our project, we found outliers in humidity and windspeed variables. In other variables the values are consistent with each other.

We used KNN imputation method to impute values in R and median method in Python. Let's go for some visualizations of outliers (we used boxplot technique):

Image 4 & 5

atemp



count

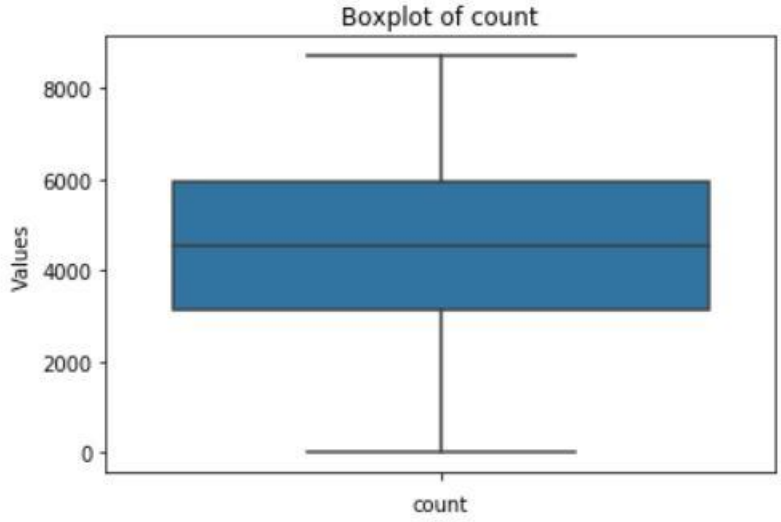


Image 6

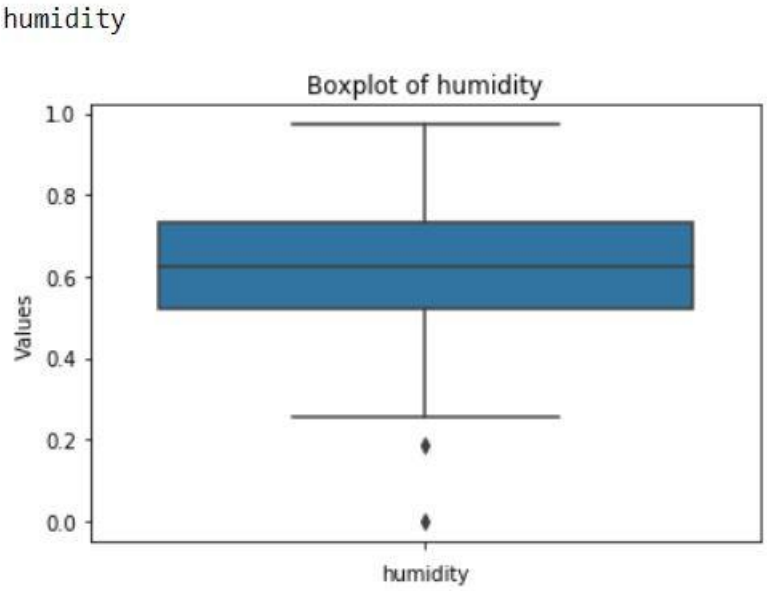
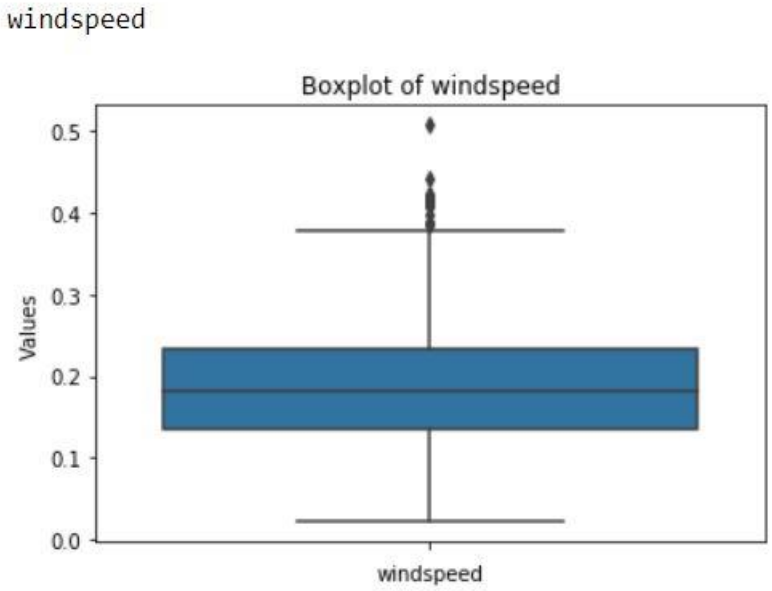


Image 7



You can observe in windspeed (Image 7) and humidity (Image 6) figure, we have small dots, they are the outliers and we imputed them using KNN and median method.

2.1.3 Data Understanding

Understanding visually is easier sometimes and for that purpose we have few libraries in python which allows us to plot awesome visualizations.

We get data, and in order to talk about how few attributes relate with each other, we can use these visualizations.

You can give two variables, to know how they are related with each other. You can also give three variables to understand the relationship between three variables. You got infinite possibilities to plot. Let's have a view on few of the plots we used in our project.

Image 8

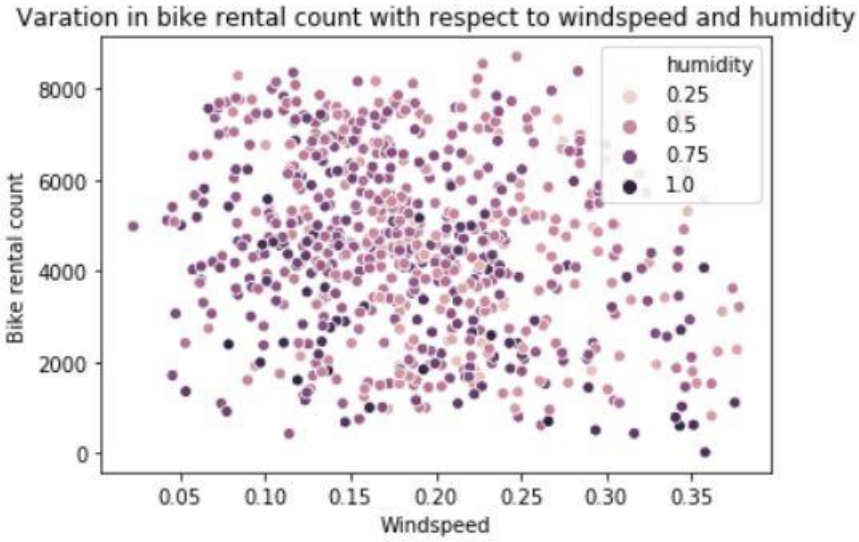


Image 9

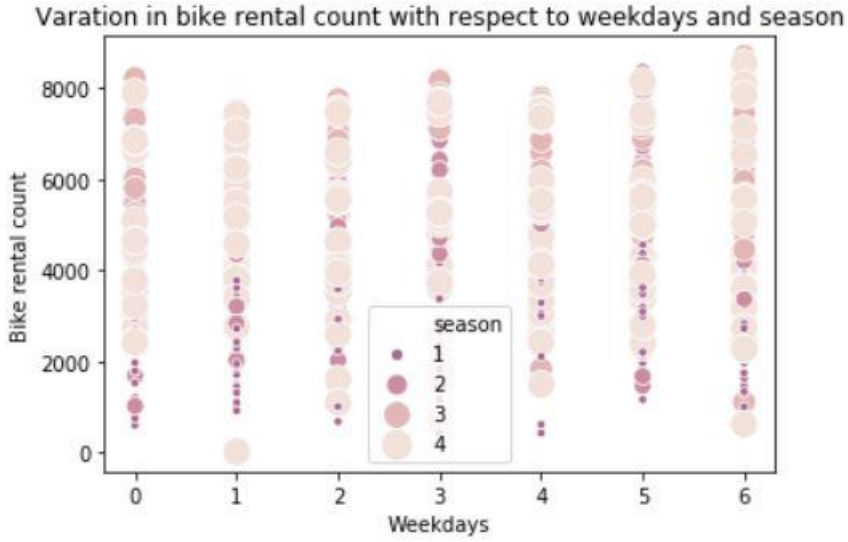


Image 10

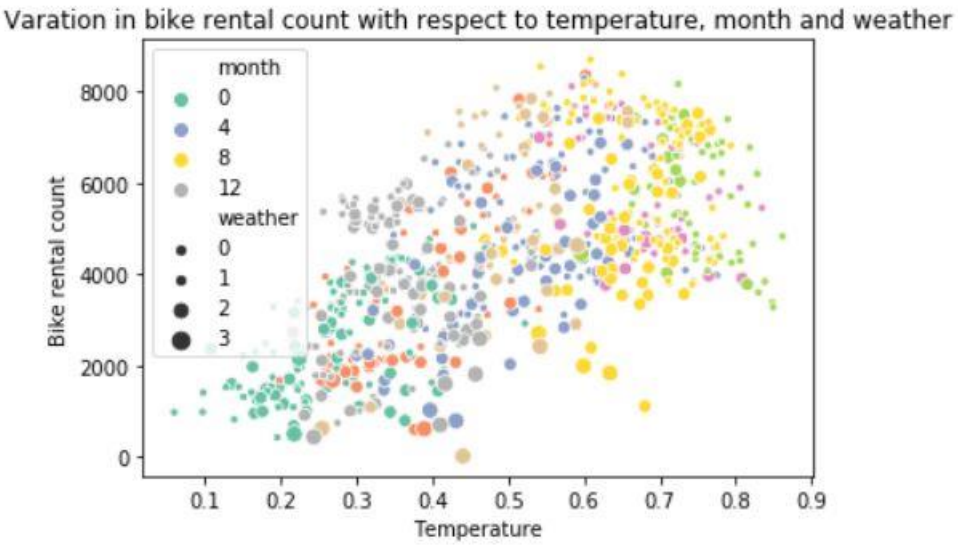
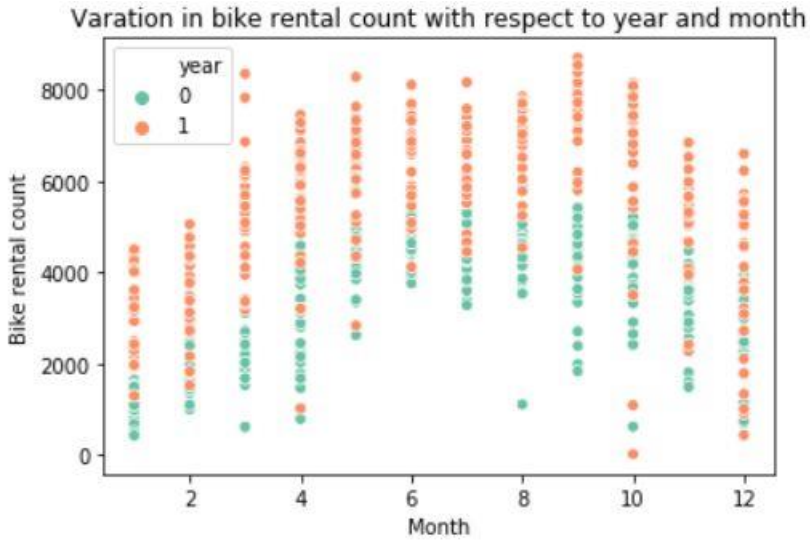


Image 11



Let's talk about our observations from the above-mentioned visualizations,

In image 8**,** count is high when windspeed is from 0.10 to 0.25 and humidity is from 0.5 to 0.7

In image 9, count is high in 4th season and 1st and 6th weekdays.

In image 10, count is high when temperature is from 0.5 to 0.8, in 8th month and weather is 0.

In image 11, count is high in year 1, particularly from season 3 to 12 excluding 9.

2.1.4 Feature Selection

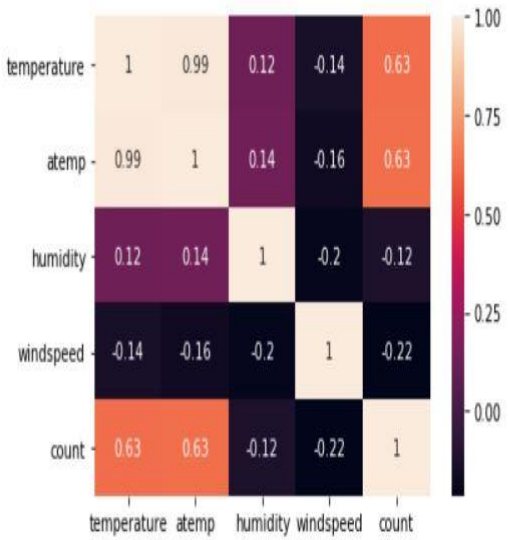
In Feature Selection, we use few techniques to know which variables are important to us, it's all about selecting subset of variables from all variables. Actually, when we are given with a dataset, we perform exploratory data analysis, later data pre-processing to clean the data and transform it from unstructured to structured data to feed into model.

However, after this, we may face a situation where we may have variables which have same information with them about the target variable. Let's talk about a example, in a situation where we are sending five people on a mission. Later, you came to know that, two individuals have the same exact information with them about the mission. Definitely you would drop one, in order to reduce infrastructure, complexity.

The same way, we also drop few variables if they have same information. We always aim that, there should be no independent variable which talks the same as other independent variables but, we appreciate those variables which talks more about the target variable.

In our project, we did correlation analysis between target variable (continuous variable) and other continuous variables. Let's check it:

Image 12



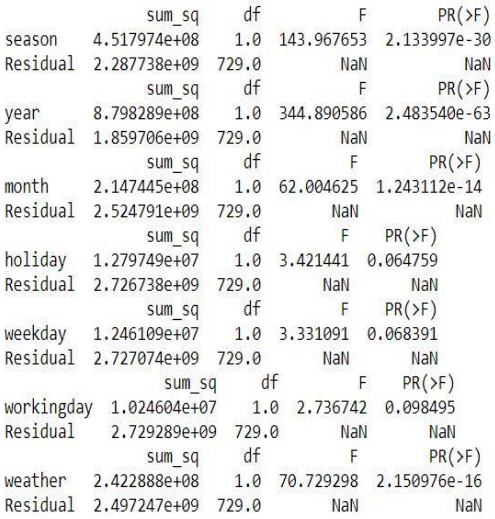
From the analysis diagram, we can understand that, temperature and atemp are highly correlated with each other.

Now, we are going to drop atemp variable as it talks same as temperature.

Next, we did Anova test between categorical variables and target variable (continuous variable)

Let’s check that:

Image 13



	sum_sq	df	F	PR(>F)
season	4.517974e+08	1.0	143.967653	2.133997e-30
Residual	2.287738e+09	729.0	NaN	NaN
	sum_sq	df	F	PR(>F)
year	8.798289e+08	1.0	344.890586	2.483540e-63
Residual	1.859706e+09	729.0	NaN	NaN
	sum_sq	df	F	PR(>F)
month	2.147445e+08	1.0	62.004625	1.243112e-14
Residual	2.524791e+09	729.0	NaN	NaN
	sum_sq	df	F	PR(>F)
holiday	1.279749e+07	1.0	3.421441	0.064759
Residual	2.726738e+09	729.0	NaN	NaN
	sum_sq	df	F	PR(>F)
weekday	1.246109e+07	1.0	3.331091	0.068391
Residual	2.727074e+09	729.0	NaN	NaN
	sum_sq	df	F	PR(>F)
workingday	1.024604e+07	1.0	2.736742	0.098495
Residual	2.729289e+09	729.0	NaN	NaN
	sum_sq	df	F	PR(>F)
weather	2.422888e+08	1.0	70.729298	2.150976e-16
Residual	2.497247e+09	729.0	NaN	NaN

From the result, we can observe, holiday, weekday and workingday has p value > 0.05, by which, we accept null hypothesis.

It’s clear, that holiday, weekday, and working day has p value > 0.05, by which we accept null hypothesis (all means are equal, they carry same information). So, we are going to drop them even.

That’s all about Feature Selection.

2.1.5 Feature Scaling

In Feature Scaling, we try to limit the ranges of variables, so that they can be compared on the same ground.

Let’s talk about one example, consider, two variables, age and income. Age is varied from 1 to 100 but coming to income, it ranges in a large scale.

At this situation, the higher values bias the result towards themselves, in order to overcome a situation like this, we use Feature Scaling, where we limit the ranges of variables.

In our project, we didn’t go for feature scaling as the given data was already approximately symmetric. We also went for distribution check, so let’s have look on that.

Image 14

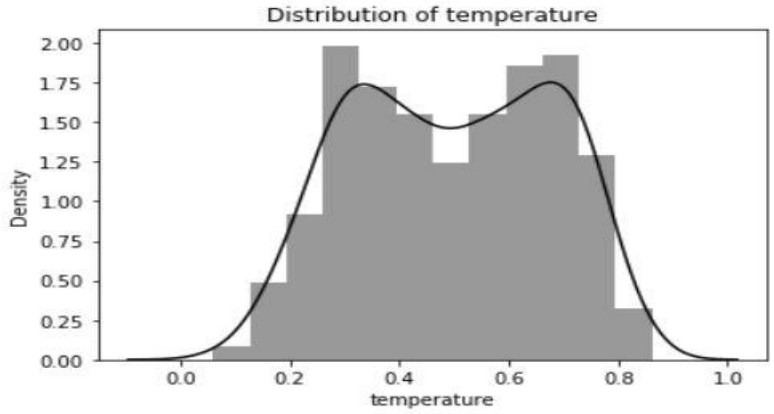


Image 15

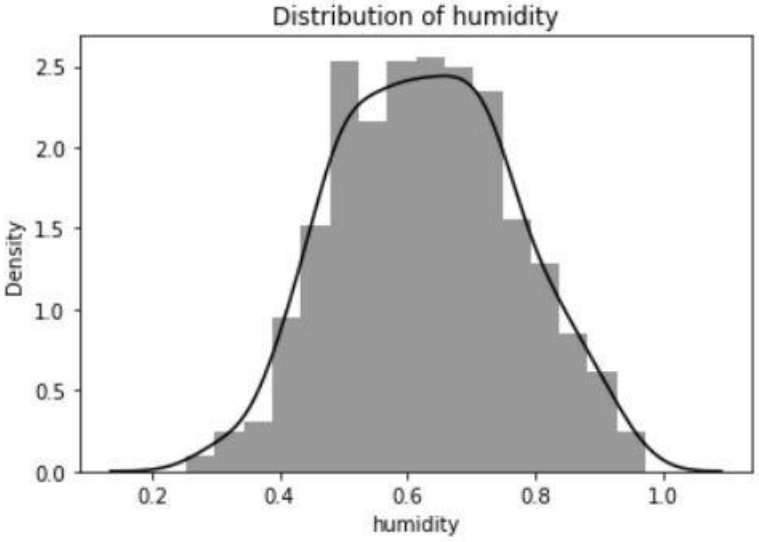


Image 16

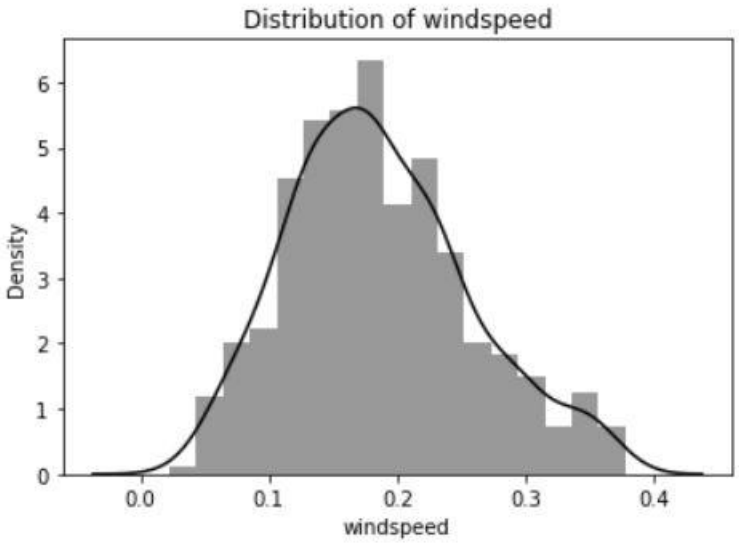
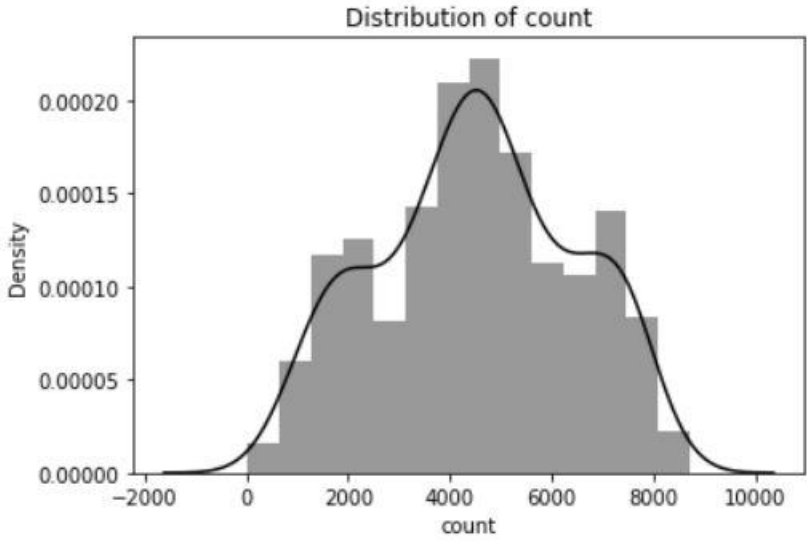


Image 17



Now, let's also check the distribution in numeric values form:

Image 18

	season	year	month	weather	temperature	humidity	windspeed	count
count	731.000000	731.000000	731.000000	731.000000	731.000000	731.000000	731.000000	731.000000
mean	2.496580	0.500684	6.519836	1.395349	0.495385	0.629354	0.186257	4504.348837
std	1.110807	0.500342	3.451913	0.544894	0.183051	0.139566	0.071156	1937.211452
min	1.000000	0.000000	1.000000	1.000000	0.059130	0.254167	0.022392	22.000000
25%	2.000000	0.000000	4.000000	1.000000	0.337083	0.522291	0.134950	3152.000000
50%	3.000000	1.000000	7.000000	1.000000	0.498333	0.627500	0.178802	4548.000000
75%	3.000000	1.000000	10.000000	2.000000	0.655417	0.730209	0.229786	5956.000000
max	4.000000	1.000000	12.000000	3.000000	0.861667	0.972500	0.378108	8714.000000

From the above diagrams and Image 18, it's clear that data is already normalized and don't need any Feature Scaling.

2.2 Model Development

Model Development, is the phase which comes after we are done with applying the exploratory data analysis, data pre processing techniques, on the top of data.

The data, will be in structured format, which was our goal, is now ready to develop model.

After we defined our objective and received the data, we transformed it into our required form, we enter into model development, but before that, let's discuss about model selection.

2.2.1 Model Selection

Model Selection particularly depends on the objective, the problem statement. We have to know at first hand, that, under which category, the problem statement falls.

We have four categories:

- Forecasting
- Classification
- Optimization
- Unsupervised Learning

Our problem statement is - predict the bike rental count on daily basis, considering the environmental and seasonal settings.

Our problem statement is a regression problem (target variable is a continuous variable) and it falls under Forecasting category.

In our project we decided to go with, Linear Regression, Decision Tree and Random Forest.

2.2.2 Linear Regression

Linear Regression is a supervised machine learning algorithm where the predicted output is continuous and has a constant slope. It's used to predict values within a continuous range, (e.g. sales, price) rather than trying to classify them into categories (e.g. cat, dog).

Linear Regression, unlike other algorithms, stores information in terms of coefficients. It is a statistical model. We cannot use this for classification. It describes relationship among variables.

In our project, we get MAPE as 18.800696 and Rsquare as 0.843604.

2.2.3 Decision Tree

Decision Trees are a type of Supervised Machine Learning (that is you explain what the input is and what the corresponding output is in the training data) where the data is continuously split according to a certain parameter.

The tree can be explained by two entities, namely decision nodes and leaves. The leaves are the decisions or the final outcomes. And the decision nodes are where the data is split.

In our project, we get MAPE as 36.948093 and Rsquare as 0.843604.

2.2.4 Random Forest

The Random Forest is a model made up of many decision trees. Rather than just simply averaging the prediction of trees (which we could call a “forest”), this model uses two key concepts that gives it the name *random*:

- Random sampling of training data points when building trees
- Random subsets of features considered when splitting nodes

The random forest combines hundreds or thousands of decision trees, trains each one on a slightly different set of the observations, splitting nodes in each tree considering a limited number of the features. The final predictions of the random forest are made by averaging the predictions of each individual tree.

In our project, we get MAPE as 20.395857 and Rsquare as 0.891176.

Chapter 3

Conclusion

3.1 Model Evaluation

We always need a metric to evaluate the work we did. So, the same way, after we developed our models, we need a metric to validate the model we developed.

There are many metrics to evaluate, even, we have different metrics for classification problem and different metrics for regression problems.

For classification problems, we have metrics like:

- Confusion Matrix.
- Accuracy.
- Recall.
- Specificity.

For regression problems, we have metrics like:

- MSE.
- RMSE.
- MAPE.
- Rsquare.

We are choosing MAPE and Rsquare for our project.

3.1.1 Mean Absolute Percentage Error (MAPE) & Rsquare

We are going to use MAPE and Rsquare as our error metrics to evaluate our models.

MAPE – It is the sum of calculated errors.

Rsquare – Correlation of original and predicted values.

3.2 Model Selection

Finally, it's our Model selection time. We developed three models. Linear Regression, Decision Tree, and Random Forest.

We are going to freeze Random Forest with MAPE as 20.395857 and Rsquare as 0.891176, even though Linear Regression has low MAPE value than Random Forest, because Rsquare is our priority, it's because, high Rsquare is equivalent to minimizing sum of squared errors, on the other hand, minimizing MAPE may yield biased point predictions.

R Result:

Image 19

	Model	MAPE_Values_Train	MAPE_Values_Test	Rsquare_Values_Train	Rsquare_Values_Test
1	Linear Regression	47.51450	16.75479	0.8368754	0.8347779
2	Decision Tree	56.30015	23.70970	0.7939743	0.7521948
3	Random Forest	24.92561	17.37557	0.9680657	0.8679717

Python Result:

Image 20

Model Name	MAPE_Train	MAPE_Test	R-squared_Train	R-squared_Test
Linear Regression	44.444512	18.800696	0.832760	0.843604
Decision Tree	62.260133	36.948093	0.677563	0.654461
Random Forest	19.391452	20.395857	0.980206	0.891176

Appendix A – R Code

```
# Let's clean the R environment first

rm(list = ls())

# Next, set the working directory

setwd("C:/Users/Purushottam/Desktop/Projects")

# Time to check the directory

getwd()

# Importing csv file into R environment

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

### Our objective of this project is - Predication of bike rental count on daily based on the environmental and seasonal settings

### The flow we are gong to work is - 1) Exploratory Data Analysis, 2) Data pre-Processing, 3) Model Development
### 4) Model Evaluation, 5) Result - choosing the best model with lowest MAPE and R-square > 80%

#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~Let's perform EXPLORATORY DATA ANALYSIS~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

class(day) # Its DataFrame
head(day) # Let's have a look on first 6 observations
dim(day) # 731 observations & 16 variables
str(day) # Have a look on the structure
summary(day) # Data seems to be normalized
names(day) # In names, we can see shortcuts like, hum for humidity

# It's time for us to remove variables (if any), which don't add any meaning
# In our data, instant can be dropped as it simply represents the index
# Next, dteday can also be neglected as we are not dealing with time series analysis
# Finally, let's delete casual and registered variables, as our dependent variable is the total of the former variables

day = subset(day, select = -c(instant, dteday, casual, registered))

# Let's check our data after the dimension reduction

dim(day) # We are left with 731 observations and 12 variables

names(day) # The above mentioned four variables are removed

# If we observe the names, they are given with shortcut words. Let's change it for our convenience

names(day)[2] = 'year'
names(day)[3] = 'month'
names(day)[7] = 'weather'
names(day)[8] = 'temperature'
names(day)[10] = 'humidity'
names(day)[12] = 'count'

# We have changed the names and let's have a look on this

names(day)

# It's time to divide our variables into numeric(num_var) and categorical(cat_var)

num_var = c('temperature', 'atemp', 'humidity', 'windspeed', 'count')

cat_var = c('season', 'year', 'month', 'holiday', 'weekday', 'workingday', 'weather')


#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~It's time for DATA PRE PROCESSING~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~


#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~We will start with Missing Value Analysis~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

sum(is.na(day)) # Line of code to know the sum of missing values in dataset

#-->> We got the Missing Values in the whole dataset as 0. No need to perform any missing values operations

# Next, we are going to check for OUTLIERS If outlierts are present we are going to impute them with KNN / Mean / Medain,
# the one which gives the least value

day2=day
day=day2
```

```r
# Boxplot Technique is used to check for the Outliers

for(i in 1:length(num_var)){
  assign(paste0("AB",i),ggplot(aes_string(x="count",y=(num_var[i])),d=subset(day))+
           geom_boxplot(outlier.color = "Red",outlier.shape = 18,outlier.size = 2,
                        fill="Purple")+theme_get()+
           stat_boxplot(geom = "errorbar",width=0.5)+
           labs(x="Bike Count", y=num_var[i])+
           ggtitle("Boxplot of bike count", num_var[i]))
}

gridExtra::grid.arrange(AB1,AB2,AB3,ncol=3)
gridExtra::grid.arrange(AB4,AB5,ncol=2)

#-->> We can observe outliers in windspeed and humidity variables.


# Let's go ahead by replacing outliers with NA

for(i in num_var){
  print(i)
  outlier = day[,i][day[,i] %in% boxplot.stats(day[,i])$out]
  print(length(outlier))
  day[,i][day[,i] %in% outlier] = NA
}

sum(is.na(day))

# We have total 15 outliers. Now, we are going to use Mean / Median / KNN to impute the values. We are using location day[1,11]

# Actual value is 0.160
# Mean = 0.186, after imputing
# Median = 0.179, after imputing
# KNN = 0.160, after imputing

library(DMwR)
library(rpart)


day = knnImputation(day, k = 5)
sum(is.na(day))

#-->> We found out KNN is giving the nearest value as 0.16 when compared with the location day[1,11], which is 0.16, so, KNN
#-->> All the outliers are imputed with KNN values.


#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~Let's go for DATA UNDERSTANDING~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

# Time to plot some graphs, so let's install few libraries

library(ggplot2)
library(scales)
library(psych)
library(gplots)


# Barplot with x axis as season and y axis as count

ggplot(day, aes(x = day$season, y = day$count))+
  geom_bar(stat = "identity", fill = "blue")+
  labs(title = "Number of bikes rented with respect to season", x = "Seasons", y = "Count")+
  theme(panel.background = element_rect("white"))+
  theme(plot.title = element_text(face = "bold"))

#-->> We can observe, season 3, has the highest count and on season 1, we got low count.


# Barplot with x axis as year and y axis as count

ggplot(day, aes(x = day$year, y = day$count))+
  geom_bar(stat = "identity", fill = "red")+
  labs(title = "Number of bikes rented with respect to year", x = "Years", y = "Count")+
  theme(panel.background = element_rect("white"))+
  theme(plot.title = element_text(face = "bold"))

#-->> We can observe, Year 1, has the highest count and on year 0, we got low count.
```

```r
# Barplot with x axis as year and y axis as count

ggplot(day, aes(x = day$year, y = day$count))+
  geom_bar(stat = "identity", fill = "red")+
  labs(title = "Number of bikes rented with respect to year", x = "Years", y = "Count")+
  theme(panel.background = element_rect("white"))+
  theme(plot.title = element_text(face = "bold"))

#-->> We can observe, Year 1, has the highest count and on year 0, we got low count.


# Barplot with x axis as weekday and y axis as count

ggplot(day, aes(x = day$weekday, y = day$count))+
  geom_bar(stat = "identity", fill = "navyblue")+
  labs(title = "Number of bikes rented with respect to days", x = "Days of the week", y = "count")+
  theme(panel.background = element_rect("white"))+
  theme(plot.title = element_text(face = "bold"))

#-->> We can observe, on day 5, we have the highest count and on day 0, we got low count.


#Count with respect to temperature and humidity together

ggplot(day,aes(temperature,count)) +
  geom_point(aes(color=humidity),alpha=0.5) +
  labs(title = "Bikes rented with respect to temperature and humidity", x = "Normalized temperature", y = "Count")+
  scale_color_gradientn(colors=c('blue','light blue','dark blue','light green','yellow','dark orange','black')) +
  theme_bw()

#-->> We can observe, when normalized temperature is between 0.5 to 0.75 and humidity is between 0.50 to 0.75, count is high.


# Count with respect to windspeed and weather together

ggplot(day, aes(x = windspeed, y = count))+
  geom_point(aes(color=weather), alpha=0.5) +
  labs(title = "Bikes rented with respect to windspeed and weather", x = "Windspeed", y = "Count")+
  scale_color_gradientn(colors=c('blue','light blue','dark blue','light green','yellow','dark orange','black')) +
  theme_bw()
#-->> From the plot, we can say,  count is at peak, when windspeed is from 0.1 to 0.3 and weather is from 1.0 to 1.5.


# Count with respect to temperature and season together

ggplot(day, aes(x = temperature, y = count))+
  geom_point(aes(color=season),alpha=0.5) +
  labs(title = "Bikes rented with respect to temperature and season", x = "Normalized temperature", y = "Count")+
  scale_color_gradientn(colors=c('blue','light blue','dark blue','light green','yellow','dark orange','black')) +
  theme_bw()

#-->> From the plot, count is maximum when temperature is 0.50 to 0.75 & season 3 to season 4.


#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~Now, let's move to FEATURE SELECTION~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

day2=day
day=day2

# In Feature Selection, we perform Correlation Analysis and Anova test to find out the varaibles which are to be excluded

# Correlation Analysis is performed between num_var (numeirc independent variables) & count(continuous target variable)

library(corrgram)

corrgram(day[,num_var],order=FALSE,upper.panel = panel.pie,
         text.panel = panel.txt,
         main= "Correlation Analysis between numeric variables")

#-->> From the analysis diagram, we can understand that, temperature and atemp are highly correlated with each other.


# Anova Test is performed between cat_var (categorical independent variables) & count(continuous target variable)

for(i in cat_var){
  print(i)
  Anova_test_result = summary(aov(formula = count~day[,i],day))
  print(Anova_test_result)
}
```

```r
#-->> From the result, we can observe, holiday, weekday and workingday has p value > 0.05, by which, we accept null hypothesis.


# Deleting the below given continuous and categorical variables from day, as we found out they won't add any value to the model.

day = subset(day, select=-c(atemp,holiday,weekday,workingday))


#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~Let's jump to FEATURE SCALING~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

day2=day
day=day2

# Now, what are the numeric and categorical variables left with us

num_var = c("temperature","humidity","windspeed","count")

cat_var = c("season", "year", "month", "weather")


# Skewness test for continuous variables to know the symmetry

library(propagate)

for(i in num_var){
  print(i)
  skew = skewness(day[,i])
  print(skew)
}

#-->> After observing from console, we can conclude, the dataset is approximately symmetric as values are between -0.5 to +0.5.


# We shall go for the summary of dataset where we can see the min and max of continous variables

for(i in num_var){
  print(summary(day[,i]))
`
#-->> From summary, the data seems to be normalized, so there's no need for scaling.


# We shall check for normality as Feature Scaling is all about normality

hist(day$temperature, col="Navyblue", xlab="Temperature", ylab="Frequency",
     main="Temperature Distribution")

hist(day$humidity, col="Blue", xlab="Humidity", ylab="Frequency",
     main="Humidity Distribution")

hist(day$windspeed,col="Dark green",xlab="Windspeed",ylab="Frequency",
     main="Windspeed Distribution")

#-->> As said above, the numeirc variables distribution is approximately symmetric


# Let's save the Pre-processed data for our further reference

write.csv(data,"Bike Rental Data (Pre Processed).csv",row.names=FALSE)


# Till now, we completed Exploratory Data Analysis, plotted some visualizations, worked on Data Pre Processing

#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~Now, We can go for MODEL DEVELOPMENT~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

# Before going there, let's define two functions which are called as Error Metrics.

# As we have regression problem, let's use MAPE and R square as our metrics to evaluate the model

# Let's clean R Environment, as it uses RAM which is limited

library(DataCombine)
rmExcept("day")

# Let's create copy of data for further reference

day2 = day
day = day2
```

```r
# Defining R Square function - Correlation of original and predicted values

Rsquare = function(y,y1){
  cor(y,y1)^2
}

# Defining MAPE function - Mean Absolute Percentage Error (Calculated Errors)

MAPE = function(y,y1){
  mean(abs((y-y1)/y))*100
}

#-->> Error metrics to evaluate the model are defined


# Time to create dummies

cat_var = c("season","year","month","weather")

library(dummies)

day = dummy.data.frame(day, cat_var) # Why? because in regression analysis, it treats all independent variables as numerical


# Now, as we know, we have to divide the data into train and test. So, let's go for that.

set.seed(123)
train_index = sample(1:nrow(day),0.8*nrow(day)) # Using Simple Random Sampling Technique
train= day[train_index,]
test= day[-train_index,]

#-->> We can see our train and test data are available in the environment


#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~We are going for LINEAR REGRESSION MODEL~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

# We have to check for multicollinearity

num_var = c("temperature","humidity","windspeed")

num_var2 = day[,num_var]

library(usdm)

vifcor(num_var2, th = 0.7)

#-->> No variable from the 4 input variables has collinearity problem.

# Code for development of model

LRModel = lm(count~., train)

summary(LRModel)

# Predicting model on train data

LRTrain = predict(LRModel, train[-25])

# Predicting model on test data

LRTest = predict(LRModel, test[-25])

# Calculating MAPE for Train Data

LRMape_Train = MAPE(train[,25], LRTrain)

# Calculating MAPE for Test Data

LRMape_Test = MAPE(test[,25], LRTest)

# Calculating Rsquare for Train Data

LRR2_Train = Rsquare(train[,25], LRTrain)

# Calculating Rsquare for Test Data

LRR2_Test = Rsquare(test[,25], LRTest)
```

```r
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~It's time for DECISION TREE~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

# Code for development of model

library(rpart)

DTModel = rpart(count~., train, method = "anova")

# Predicting model on train data

DTTrain = predict(DTModel, train[-25])

# Predicting model on test data

DTTest = predict(DTModel, test[-25])

# Calculating MAPE for Train Data

DTMape_Train = MAPE(train[,25], DTTrain)

# Calculating MAPE for Test Data

DTMape_Test = MAPE(test[,25], DTTest)

# Calculating Rsquare for Train Data

DTR2_Train = Rsquare(train[,25], DTTrain)

# Calculating Rsquare for Test Data

DTR2_Test = Rsquare(test[,25], DTTest)

#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~Here we go for RANDOM FOREST~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

# Code for development of model

library(randomForest)

RFModel = randomForest(count~., train, ntree = 500, method = "anova")

# Predicting model on train data

RFTrain = predict(RFModel, train[-25])

# Predicting model on test data

RFTest = predict(RFModel, test[-25])

# Calculating MAPE for Train Data

RFMape_Train = MAPE(train[,25], RFTrain)

# Calculating MAPE for Test Data

RFMape_Test = MAPE(test[,25], RFTest)

# Calculating Rsquare for Train Data

RFR2_Train = Rsquare(train[,25], RFTrain)

# Calculating Rsquare for Test Data

RFR2_Test = Rsquare(test[,25], RFTest)
```

```
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~Finally, it's Result Time~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

Result = data.frame("Model" = c("Linear Regression", "Decision Tree", "Random Forest"),
                    "MAPE_Values_Train" = c(LRMape_Train, DTMape_Train, RFMape_Train),
                    "MAPE_Values_Test" = c(LRMape_Test, DTMape_Test, RFMape_Test),
                    "Rsquare_Values_Train" = c(LRR2_Train, DTR2_Train, RFR2_Train),
                    "Rsquare_Values_Test" = c(LRR2_Test, DTR2_Test, RFR2_Test))

# Lets's save the result
write.csv(Result,"Bike Rental Data (Result6 - Imputed KNN Values).csv",row.names=TRUE)

# Finally, the last question - Which model we are going to freeze?

# The answer is - Random Forest as it has the lowest MAPE value = 17.37557 and high Rsqaure value = 0.8679717

# Although, Linear Regression has lowest MAPE value than Random Forest, still we are going with Random Forest,
# because Rsqaure is our priority.
```

Appendix B – Python Code

Project - Bike Rental Count

In [1]:
```python
# As always, let's import all the required libraries

import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from random import randrange,uniform
from sklearn.metrics import r2_score
from scipy import stats
```

In [2]:
```python
# Setting the working directory

os.chdir(r"C:\Users\Purushottam\Desktop\Projects")
```

In [3]:
```python
# Checking the current working directory

os.getcwd()
```

Out[3]: 'C:\\Users\\Purushottam\\Desktop\\Projects'

In [4]:
```python
# Importing csv file into Jupyter notebook

day = pd.read_csv(r"C:\Users\Purushottam\Desktop\Projects\day.csv")
```

In [5]:
```python
# Let's have a look at the first five rows of the imported dataset

day.head()
```

Out[5]:

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

Our objective of this project is - Predication of bike rental count on daily based on the environmental and seasonal settings.

The flow we are gong to work is - 1) Exploratory Data Analysis, 2) Data pre-Processing, 3) Model Development, 4) Model Evaluation, 5) Result - choosing the best model with lowest MAPE and R-square > 80%

Exploratory Data Analysis

In [6]:
```
# Let's get started with Exploratory Data Analysis

#Type of DataFrame
print(type(day))
```

```
<class 'pandas.core.frame.DataFrame'>
```

In [7]:
```
# Knowing Data Types of Varaibles

print(day.dtypes)
```

```
instant        int64
dteday        object
season         int64
yr             int64
mnth           int64
holiday        int64
weekday        int64
workingday     int64
weathersit     int64
temp         float64
atemp        float64
hum          float64
windspeed    float64
casual         int64
registered     int64
cnt            int64
dtype: object
```

In [8]:
```
# Have a look at the dimension of the dataset

print(day.shape)
```

```
(731, 16)
```

In [9]:
```
# Index ranges from 0 to 731

print(day.index)
```

```
RangeIndex(start=0, stop=731, step=1)
```

In [10]:
```
# Total 16 columns are present

print(day.columns)
```

```
Index(['instant', 'dteday', 'season', 'yr', 'mnth', 'holiday', 'weekday',
       'workingday', 'weathersit', 'temp', 'atemp', 'hum', 'windspeed',
       'casual', 'registered', 'cnt'],
      dtype='object')
```

In [11]:
```
# Gives us the unique values present in each variable

print(day.nunique())
```

```
instant       731
dteday        731
season          4
yr              2
mnth           12
holiday         2
weekday         7
workingday      2
weathersit      3
temp          499
atemp         690
hum           595
windspeed     650
casual        606
registered    679
cnt           696
dtype: int64
```

In [12]:
```
# It's time for us to remove variables (if any), which don't add any meaning

# In our data, instant can be dropped as it simply represents the index

# Next, dteday can also be neglected as we are not dealing with time series analysis

# Finally, let's delete casual and registered variables, as our dependent variable is the total of the former variables
day = day.drop(day.columns[[0, 1, 13, 14]], axis = 1)

print(day.shape)
```

```
(731, 12)
```

In [13]:
```
# If we observe the names, they are given with shortcut words. Let's change it for our convenience

day = day.rename(columns = {'yr': 'year', 'mnth': 'month', 'weathersit': 'weather', 'temp': 'temperature',
                            'hum': 'humidity', 'cnt': 'count'})

# We have changed the names and let's have a look on this

print(day.columns)
```

```
Index(['season', 'year', 'month', 'holiday', 'weekday', 'workingday',
       'weather', 'temperature', 'atemp', 'humidity', 'windspeed', 'count'],
      dtype='object')
```

In [14]:
```
# It's time to divide our variables into numeric(num_var) and categorical(cat_var), as we are going -
# For Data pre processing in the next step

num_var = ['temperature', 'atemp', 'humidity', 'windspeed', 'count']

cat_var = ['season', 'year', 'month', 'holiday', 'weekday', 'workingday', 'weather']
```

It's time for DATA PRE PROCESSING

We will start with Missing Value Analysis

In [15]:
```
# Line of code to know the sum of missing values in dataset

day.isnull().sum()
```

Out[15]:
```
season        0
year          0
```

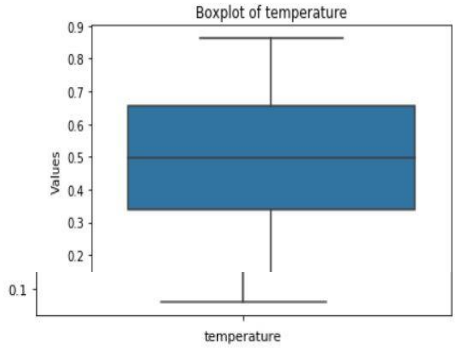
```
month          0
holiday        0
weekday        0
workingday     0
weather        0
temperature    0
atemp          0
humidity       0
windspeed      0
count          0
dtype: int64
```

We don't have any missing vlaues in the dataset

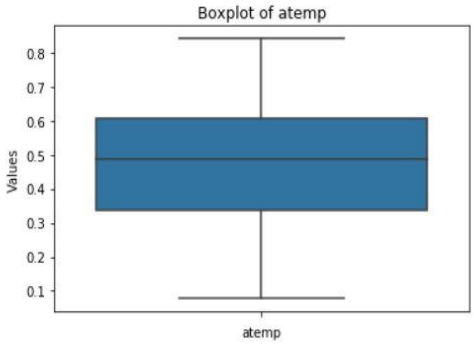
Next, we are going to check for OUTLIERS

In [16]:
```
for i in num_var:
    print(i)
    sns.boxplot(y = day[i])
    plt.xlabel(i)
    plt.ylabel("Values")
    plt.title("Boxplot of " + i)
    plt.show()
```

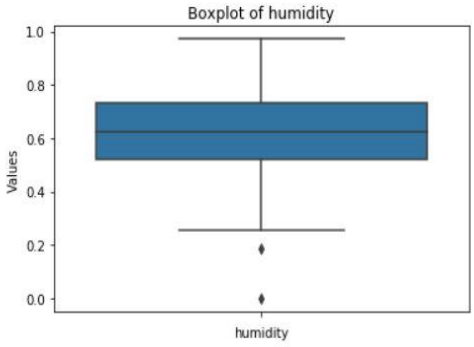
temperature



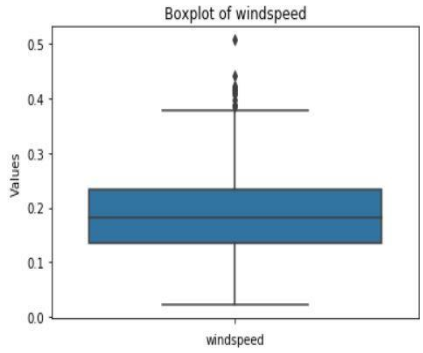
atemp



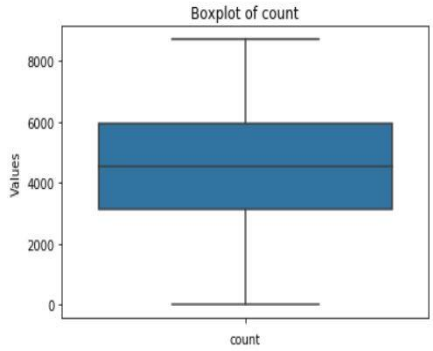
humidity



windspeed



count



We can observe outliers in windspeed and humidity variables.

In [17]:
```
# In this project, we are going to impute the ouliers, but before that, let's obtain the outliers

# Now, we have to calculate Inner Fence, Outer Fence, and IQR

for i in num_var:
    print(i)
    q75, q25 = np.percentile(day.loc[:,i], [75, 25])
    iqr = q75 - q25
    Innerfence = q25 - (iqr*1.5)
    Upperfence = q75 + (iqr*1.5)
    print("Innerfence= "+str(Innerfence))
    print("Upperfence= "+str(Upperfence))
    print("IQR ="+str(iqr))


# Let's go ahead by replacing outliers with NA

    day.loc[day[i]<Innerfence, i] = np.nan
    day.loc[day[i]>Upperfence, i] = np.nan
```

```
temperature
Innerfence= -0.14041600000000015
Upperfence= 1.1329160000000003
IQR =0.3183330000000001
atemp
Innerfence= -0.06829675000000018
Upperfence= 1.0147412500000002
IQR =0.2707595000000001
humidity
Innerfence= 0.20468725
Upperfence= 1.0455212500000002
IQR =0.21020850000000002
windspeed
Innerfence= -0.012446750000000034
Upperfence= 0.38061125
IQR =0.0982645
count
Innerfence= -1054.0
Upperfence= 10162.0
IQR =2804.0
```

In [18]:
```
day.isna().sum()
```

Out[18]:
```
season          0
year            0
month           0
holiday         0
weekday         0
workingday      0
weather         0
temperature     0
atemp           0
humidity        2
windspeed      13
count           0
dtype: int64
```

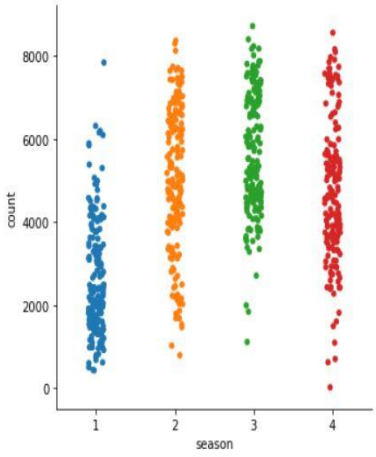
In [19]:
```
# We have total 15 outliers. Now, we are going to use Median to impute the values.

day['humidity'] = day['humidity'].fillna(day['humidity'].median())
day['windspeed'] = day['windspeed'].fillna(day['windspeed'].median())
```

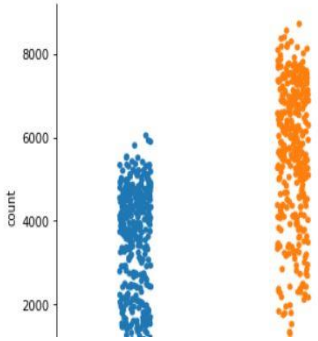
In [20]:
```
# No NA values now

day.isna().sum()
```

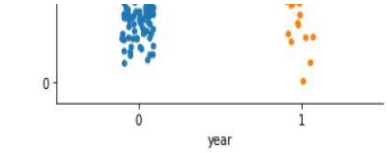
Out[20]:
```
season         0
year           0
month          0
holiday        0
weekday        0
workingday     0
weather        0
temperature    0
atemp          0
humidity       0
windspeed      0
count          0
dtype: int64
```

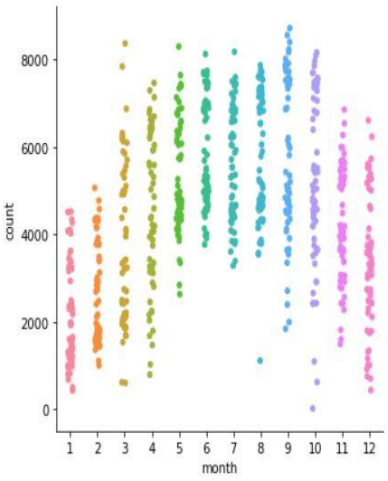
Let's go for DATA UNDERSTANDING

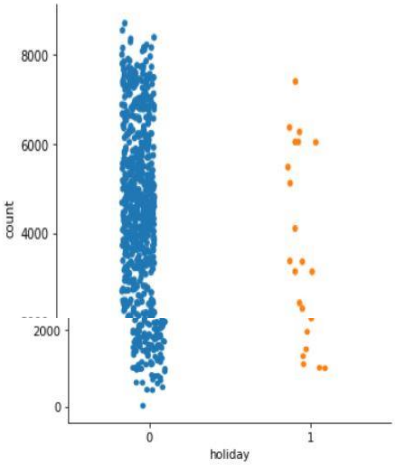
In [21]:
```
for i in cat_var:
    sns.catplot(x = i, y = "count", data=day)
    figname = str(i)+'.pdf'
    plt.savefig(figname)
```

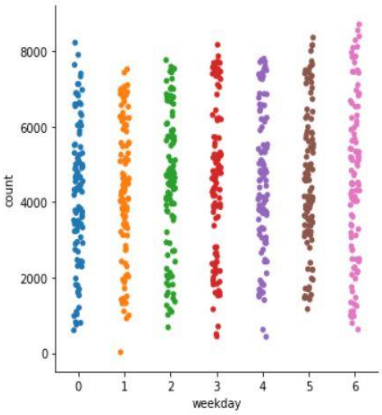


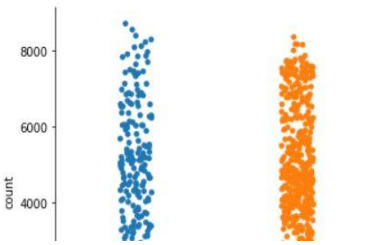


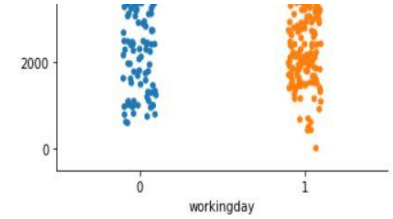

0
0
1
year

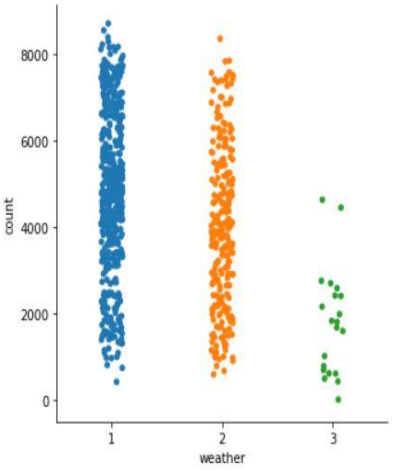

8000
6000
4000
2000
0
count
1 2 3 4 5 6 7 8 9 10 11 12
month


8000
6000
4000
2000
0
count
0
1
holiday


8000
6000
4000
2000
0
count
0 1 2 3 4 5 6
weekday


8000
6000
4000
count





Season 2, 3 and 4 has the highest count

Year 1 has high count than 0

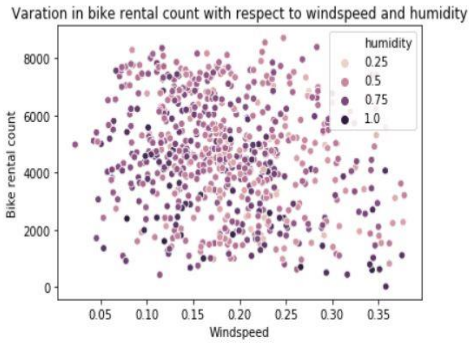
Months 3 to 10 has got pretty good count

Obviously on holidays the count is higher than non-holidays

In weekdays, 0 and 6 has the highest count

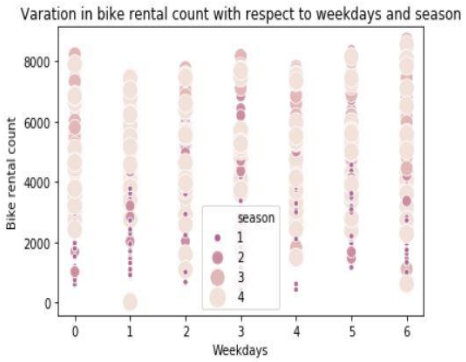
In weather, 1 has the highest count

In [22]:
```
ax = sns.scatterplot(x="windspeed", y="count", hue="humidity",
                     data=day)

plt.title("Varation in bike rental count with respect to windspeed and humidity")
plt.ylabel("Bike rental count")
plt.xlabel("Windspeed")
plt.savefig('Bike count with windspeed and humidity.pdf')
```

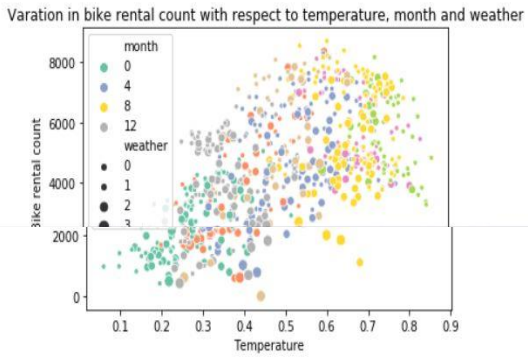


In [23]:
```
cmap = sns.cubehelix_palette(dark=.9, light=.5, as_cmap=True)

ax = sns.scatterplot(x="weekday", y="count",
            hue="season", size="season", sizes = (20, 200), palette=cmap,
             data=day)

plt.title("Varation in bike rental count with respect to weekdays and season")
plt.ylabel("Bike rental count")
plt.xlabel("Weekdays")
```

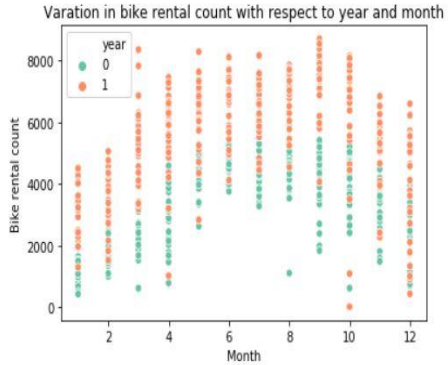
```
plt.savefig('Bike count with weekdays and season.pdf')
```



In [24]:
```
cmap = sns.cubehelix_palette(dark=.3, light=.8, as_cmap=True)

ax = sns.scatterplot(x="temperature", y="count",
                     hue="month", size="weather",
                     palette="Set2",
                     data=day)

plt.title("Varation in bike rental count with respect to temperature, month and weather")
plt.ylabel("Bike rental count")
plt.xlabel("Temperature")
plt.savefig('Bike count with temperature, month and weather.pdf')
```



In [25]:
```
cmap = sns.cubehelix_palette(dark=.3, light=.8, as_cmap=True)

ax = sns.scatterplot(x="month", y="count",
                     hue="year",
                     palette="Set2",
                     data=day)

plt.title("Varation in bike rental count with respect to year and month")
plt.ylabel("Bike rental count")
plt.xlabel("Month")
plt.savefig('Bike count with month and year.pdf')
```



In varation in bike rental count with respect to windspeed and humidity, count is high when windspeed is from 0.10 to 0.25 and humidity is from 0.5 to 0.75

In varation in bike rental count with respect to weekdays and season, count is high in 4th season and 1st and 6th weekdays

In varation in bike rental count with respect to temperature, month and weather, count is high when temperature is from 0.5 to 0.8, in 8th month and weather is 0.

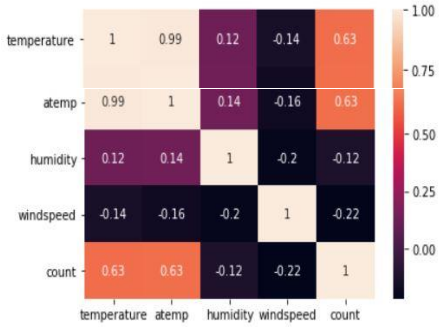
In, varation in bike rental count with respect to year and month, count is high in year 1, particularly from season 3 to 12 excluding 9

Now, let's move to FEATURE SELECTION

In [26]:
```
# In Feature Selection, we perform Correlation Analysis and Anova test to find out the varaibl
# -es which are to be excluded before feeding to the model

# Correlation Analysis is performed between num_var (numeirc independent variables) & count(continuous target variable)

day_cor = day.loc[:, num_var]

# line of code to generate the corrleation

cor_result = day_cor.corr()

print(cor_result)
```

```
             temperature     atemp  humidity  windspeed     count
temperature     1.000000  0.991702  0.123723  -0.138937  0.627494
atemp           0.991702  1.000000  0.137312  -0.164157  0.631066
humidity        0.123723  0.137312  1.000000  -0.200237 -0.121454
windspeed      -0.138937 -0.164157 -0.200237   1.000000 -0.215203
count           0.627494  0.631066 -0.121454  -0.215203  1.000000
```

In [27]:
```
ax = sns.heatmap(cor_result, annot=True)
```



From the analysis diagram, we can understand that, temperature and atemp are highly correlated with each other.

In [28]:
```
# Anova Test is performed between cat_var (categorical independent variables) & count(continuous target variable)

import statsmodels.api as sm
from statsmodels.formula.api import ols

for i in cat_var:
    mod = ols('count' + '~' + i, data = day).fit()
    aov_table = sm.stats.anova_lm(mod, typ = 2)
    print(aov_table)
```

```
               sum_sq     df           F        PR(>F)
season    4.517974e+08    1.0  143.967653  2.133997e-30
Residual  2.287738e+09  729.0         NaN           NaN
               sum_sq     df           F        PR(>F)
year      8.798289e+08    1.0  344.890586  2.483540e-63
Residual  1.859706e+09  729.0         NaN           NaN
               sum_sq     df          F        PR(>F)
month     2.147445e+08    1.0  62.004625  1.243112e-14
Residual  2.524791e+09  729.0        NaN           NaN
               sum_sq     df         F    PR(>F)
holiday   1.279749e+07    1.0  3.421441  0.064759
Residual  2.726738e+09  729.0       NaN       NaN
               sum_sq     df         F    PR(>F)
weekday   1.246109e+07    1.0  3.331091  0.068391
Residual  2.727074e+09  729.0       NaN       NaN
```

```
                 sum_sq     df         F    PR(>F)
workingday  1.024604e+07    1.0  2.736742  0.098495
Residual    2.729289e+09  729.0       NaN       NaN
                sum_sq     df          F        PR(>F)
weather   2.422888e+08    1.0  70.729298  2.150976e-16
Residual  2.497247e+09  729.0        NaN           NaN
```

From the result, we can observe, holiday, weekday and workingday has p value > 0.05, by which, we accept null hypothesis.

In [29]:
```
# Deleting the below given continuous and categorical variables from da-
# -y, as we found out they won't add any value to the model - Dimension Reduction

day = day.drop(['atemp', 'holiday', 'weekday', 'workingday'], axis = 1)

print(day.shape)
```

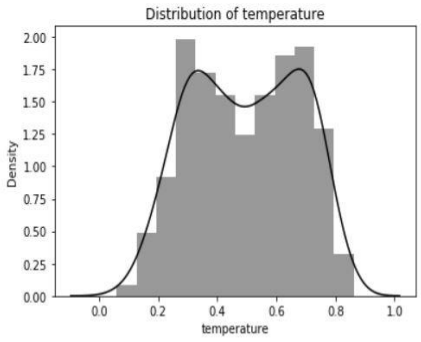
```
(731, 8)
```

In [30]:
```
# Now, what are the numeric and categorical variables left with us

num_var = ["temperature","humidity","windspeed","count"]   # numeric variables

cat_var = ["season", "year", "month", "weather"]   # categorical variables
```

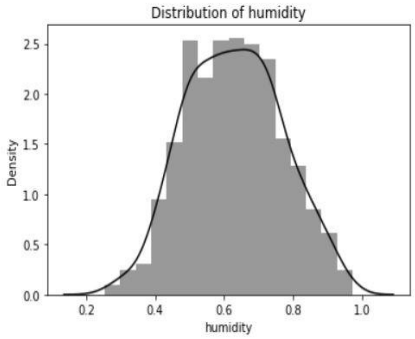
Let's jump to FEATURE SCALING

In [31]:
```
# We shall check for normality as Feature Scaling is all about normality

for i in num_var:
    print(i)
    sns.distplot(day[i], bins = 'auto', color = 'black')
    plt.title("Distribution of "+i)
    plt.ylabel("Density")
    plt.show()
```

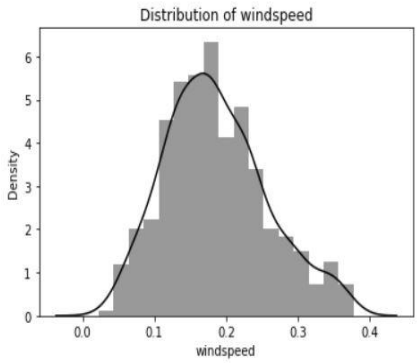
```
temperature
C:\Users\Purushottam\Anaconda3\lib\site-packages\scipy\stats\stats.py:1713: FutureWarning: Using a non-tuple sequence for multi
dimensional indexing is deprecated; use `arr[tuple(seq)]` instead of `arr[seq]`. In the future this will be interpreted as an a
rray index, `arr[np.array(seq)]`, which will result either in an error or a different result.
  return np.add.reduce(sorted[indexer] * weights, axis=axis) / sumval
```



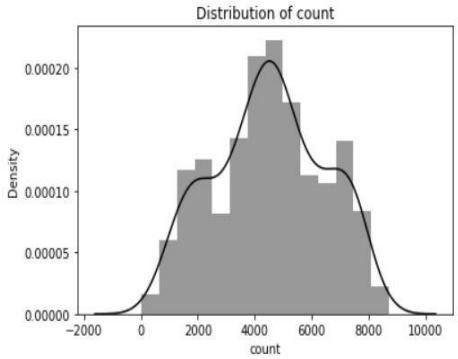

```
humidity
```



windspeed



count



As seen from above, the numeirc variables distribution are, approximately symmetric

In [32]:
```
# We also have a function describe, by which we can have a view on min and max values of each numeric variable

day.describe()
```

Out[32]:

	season	year	month	weather	temperature	humidity	windspeed	count
count	731.000000	731.000000	731.000000	731.000000	731.000000	731.000000	731.000000	731.000000
mean	2.496580	0.500684	6.519836	1.395349	0.495385	0.629354	0.186257	4504.348837
std	1.110807	0.500342	3.451913	0.544894	0.183051	0.139566	0.071156	1937.211452
min	1.000000	0.000000	1.000000	1.000000	0.059130	0.254167	0.022392	22.000000
25%	2.000000	0.000000	4.000000	1.000000	0.337083	0.522291	0.134950	3152.000000
50%	3.000000	1.000000	7.000000	1.000000	0.498333	0.627500	0.178802	4548.000000
75%	3.000000	1.000000	10.000000	2.000000	0.655417	0.730209	0.229786	5956.000000
max	4.000000	1.000000	12.000000	3.000000	0.861667	0.972500	0.378108	8714.000000

Temperature, windspeed and humidity, already seems to be normalized, so, no need for scaling

Now, We can go for MODEL DEVELOPMENT

In [33]:
```
# Let's create a copy of data for further reference

day2 = day.copy()
day = day2.copy()
```

In [34]:
```
# Time to create dummies - Why? because in regression analysis, it treats all independent variables as numerical

day = pd.get_dummies(day, columns = cat_var)

day.shape
```

Out[34]: (731, 25)

In [35]:
```
day.head()
```

Out[35]:

	temperature	humidity	windspeed	count	season_1	season_2	season_3	season_4	year_0	year_1	...	month_6	month_7	month_8	month_9	month_10
0	0.344167	0.805833	0.160446	985.0	1	0	0	0	1	0	...	0	0	0	0	0
1	0.363478	0.696087	0.248539	801.0	1	0	0	0	1	0	...	0	0	0	0	0
2	0.196364	0.437273	0.248309	1349.0	1	0	0	0	1	0	...	0	0	0	0	0
3	0.200000	0.590435	0.160296	1562.0	1	0	0	0	1	0	...	0	0	0	0	0
4	0.226957	0.436957	0.186900	1600.0	1	0	0	0	1	0	...	0	0	0	0	0

5 rows × 25 columns

In [36]:
```
# Let's import few more libraries-

from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
from scipy.stats.stats import pearsonr
```

In [37]:
```
# Before going for Model Development, let's define two functions which are called as Error Metrics.

def MAPE(y_true, y_prediction):
    MAPE = np.mean(np.abs(y_true-y_prediction)/y_true)*100
    return MAPE

def Rsquare(y_true, y_prediction):
    Rsquare = np.corrcoef(y_true,y_prediction)**2
    return Rsquare
```

In [38]:
```
# First let's separate, predictors and target

X = day.drop(['count'], axis = 1)
y = day['count']
```

In [39]:
```
# Now, as we know, we have to divide the data into train and test. So, let's go for that.

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=.20, random_state=0)
```

We are going for LINEAR REGRESSION MODEL

In [40]:
```
# Let's import a library to perform Linear Regression

import statsmodels.api as sm

# Code for development of model

LRModel= sm.OLS(y_train, X_train).fit()

print(LRModel.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  count   R-squared:                       0.833
Model:                            OLS   Adj. R-squared:                  0.827
Method:                 Least Squares   F-statistic:                     140.2
Date:                Thu, 02 May 2019   Prob (F-statistic):          1.63e-203
Time:                        15:20:28   Log-Likelihood:                -4716.2
No. Observations:                 584   AIC:                             9474.
Df Residuals:                     563   BIC:                             9566.
Df Model:                          20
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
temperature  4807.6605    477.418     10.070      0.000    3869.923    5745.398
humidity    -1840.0359    351.762     -5.231      0.000   -2530.963   -1149.109
windspeed   -2692.7145    509.781     -5.282      0.000   -3694.019   -1691.410
season_1     -160.8963    149.431     -1.077      0.282    -454.407     132.615
season_2      735.4147    149.261      4.927      0.000     442.239    1028.591
```

```
season_3      756.5640    170.170      4.446      0.000     422.319    1090.809
season_4     1424.2811    170.259      8.365      0.000    1089.860    1758.702
year_0        409.9681    152.821      2.683      0.008     109.799     710.137
year_1       2345.3954    151.325     15.499      0.000    2048.166    2642.625
month_1        -1.9341    197.841     -0.010      0.992    -390.531     386.663
month_2        45.1383    186.947      0.241      0.809    -322.060     412.337
month_3       510.8770    141.897      3.600      0.000     232.166     789.588
month_4       233.3586    174.311      1.339      0.181    -109.021     575.738
month_5       659.7195    183.392      3.597      0.000     299.503    1019.936
month_6       250.5066    180.098      1.391      0.165    -103.239     604.252
month_7      -222.2685    220.988     -1.006      0.315    -656.331     211.794
month_8       271.1265    207.045      1.310      0.191    -135.548     677.801
month_9       888.8861    173.978      5.109      0.000     547.161    1230.611
month_10      382.5832    187.383      2.042      0.042      14.528     750.639
month_11     -183.6576    194.752     -0.943      0.346    -566.188     198.873
month_12      -78.9721    168.303     -0.469      0.639    -409.550     251.606
weather_1    1643.7280     90.978     18.067      0.000    1465.030    1822.426
weather_2    1302.9232    110.447     11.797      0.000    1085.985    1519.862
weather_3    -191.2876    221.771     -0.863      0.389    -626.886     244.311
==============================================================================
Omnibus:                       97.249   Durbin-Watson:                   1.897
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              248.035
Skew:                          -0.849   Prob(JB):                     1.38e-54
Kurtosis:                       5.704   Cond. No.                     2.55e+16
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The smallest eigenvalue is 1.83e-30. This might indicate that there are
strong multicollinearity problems or that the design matrix is singular.
```

In [41]:
```
# Predicting model on train data

LRTrain = LRModel.predict(X_train)

# Predicting model on test data

LRTest = LRModel.predict(X_test)

# Calculating MAPE for Train Data

LRMape_Train = MAPE(y_train, LRTrain)
# Calculating MAPE for Test Data

LRMape_Test = MAPE(y_test, LRTest)

# Calculating Rsquare for Train Data

LRR2_Train = Rsquare(y_train, LRTrain)

LRR2_Train1 = LRR2_Train.ravel()

LRR2_Train2 = float(LRR2_Train1[1])

# Calculating Rsquare for Test Data

LRR2_Test = Rsquare(y_test, LRTest)

LRR2_Test1 = LRR2_Test.ravel()

LRR2_Test2 = float(LRR2_Test1[1])

print("MAPE for train data="+str(LRMape_Train))
print("MAPE for test data="+str(LRMape_Test))
print("Rsquare for train data="+str(LRR2_Train2))
print("Rsquare for test data="+str(LRR2_Test2))
```

```
MAPE for train data=44.44451231255284
MAPE for test data=18.80069603820695
Rsquare for train data=0.8327600660988468
Rsquare for test data=0.8436040019904952
```

In [42]:
```
# We are saving this to convert it into a dataframe later

fr1= {'Model Name': ['Linear Regression'],'MAPE_Train':[LRMape_Train],'MAPE_Test':[LRMape_Test],'R-squared_Train':[LRR2_Train2],
      'R-squared_Test':[LRR2_Test2]}

result1 = pd.DataFrame(fr1)
```

It's time for DECISION TREE

In [43]:
```
# Let's import a library to perform Decision Tree

from sklearn.tree import DecisionTreeRegressor

# Code for development of model

DTModel = DecisionTreeRegressor(max_depth=2).fit(X_train,y_train)

# Predicting model on train data

DTTrain = DTModel.predict(X_train)

# Predicting model on test data

DTTest = DTModel.predict(X_test)

# Calculating MAPE for Train Data

DTMape_Train = MAPE(y_train, DTTrain)

# Calculating MAPE for Test Data

DTMape_Test = MAPE(y_test, DTTest)

# Calculating Rsquare for Train Data

DTR2_Train = Rsquare(y_train, DTTrain)

DTR2_Train1 = DTR2_Train.ravel()

DTR2_Train2 = float(DTR2_Train1[1])

# Calculating Rsquare for Test Data

DTR2_Test = Rsquare(y_test, DTTest)

DTR2_Test1 = DTR2_Test.ravel()

DTR2_Test2 = float(DTR2_Test1[1])
```

```
DTR2_Test2 = float(DTR2_Test1[1])

print("MAPE for train data="+str(DTMape_Train))
print("MAPE for test data="+str(DTMape_Test))
print("Rsquare for train data="+str(DTR2_Train2))
print("Rsquare for test data="+str(DTR2_Test2))
```

```
MAPE for train data=62.26013293672567
MAPE for test data=36.94809301452646
Rsquare for train data=0.6775629218593647
Rsquare for test data=0.6544606873373333
```

In [44]:
```
# We are saving this to convert it into a dataframe later

fr2 = {'Model Name': ['Decision Tree'],'MAPE_Train':[DTMape_Train],'MAPE_Test':[DTMape_Test],'R-squared_Train':[DTR2_Train2],
      'R-squared_Test':[DTR2_Test2]}
result2 = pd.DataFrame(fr2)
```

In [45]:
```
# We are saving this to convert it into a dataframe later

result =  result1.append(result2)
```

Here we go for RANDOM FOREST

In [46]:
```
# Let's import a library to perform Random Forest

from sklearn.ensemble import RandomForestRegressor

# Code for development of model

RFModel = RandomForestRegressor(n_estimators=100).fit(X_train,y_train)

# Predicting model on train data

RFTrain = RFModel.predict(X_train)
```

```
# Predicting model on test data

RFTest = RFModel.predict(X_test)

# Calculating MAPE for Train Data

RFMape_Train = MAPE(y_train, RFTrain)

# Calculating MAPE for Test Data

RFMape_Test = MAPE(y_test, RFTest)

# Calculating Rsquare for Train Data

RFR2_Train = Rsquare(y_train, RFTrain)

RFR2_Train1 = RFR2_Train.ravel()

RFR2_Train2 = float(RFR2_Train1[1])

# Calculating Rsquare for Test Data

RFR2_Test = Rsquare(y_test, RFTest)

RFR2_Test1 = RFR2_Test.ravel()

RFR2_Test2 = float(RFR2_Test1[1])

print("MAPE for train data="+str(RFMape_Train))
print("MAPE for test data="+str(RFMape_Test))
print("Rsquare for train data="+str(RFR2_Train2))
print("Rsquare for test data="+str(RFR2_Test2))
```

```
MAPE for train data=19.391452297764687
MAPE for test data=20.39585675198898
Rsquare for train data=0.9802057719850207
Rsquare for test data=0.8911757492580972
```

In [47]:
```
# We are saving this to convert it into a dataframe later

fr3 = {'Model Name': ['Random Forest'],'MAPE_Train':[RFMape_Train],'MAPE_Test':[RFMape_Test],'R-squared_Train':[RFR2_Train2],
      'R-squared_Test':[RFR2_Test2]}
result3 = pd.DataFrame(fr3)
```

In [48]:
```
# We are saving this to convert it into a dataframe later

result =  result.append(result3)
```

In [49]:
```
# Finally, here's the result

result

result.set_index('Model Name')
```

Out[49]:

Model Name	MAPE_Train	MAPE_Test	R-squared_Train	R-squared_Test
Linear Regression	44.444512	18.800696	0.832760	0.843604
Decision Tree	62.260133	36.948093	0.677563	0.654461
Random Forest	19.391452	20.395857	0.980206	0.891176

Finally, the last question - Which model we are going to freeze?

The answer is - Random Forest as it has the lowest MAPE value = 20.751683 and high Rsqaure value = 0.882219

Although, Linear Regression has lowest MAPE value than Random Forest, still we are going with Random Forest, because Rsqaure is our priority.

References

- Edwisor.com
- Edwisor Community.
- Google