Bike Rent

Albert Abraham

13 September 2019

Contents

Introduction

1.1 Problem Statement

The objective of this Case is to Prediction of daily bike rental count, based on the environmental and seasonal settings.

1.2 Data

Our task is to build a regression model that will predict the bike count based on different seasonal and environmental variables.

The details of data attributes in the dataset are as follows -
instant: Record index
dteday: Date
season: Season (1:spring, 2:summer, 3:fall, 4:winter)
yr: Year (0: 2011, 1:2012)
mnth: Month (1 to 12)
hr: Hour (0 to 23)
holiday: weather day is a holiday or not (extracted fromHoliday Schedule)
weekday: Day of the week
workingday: If day is neither weekend nor holiday then it's value is 1, otherwise is 0.
weathersit: (extracted fromFreemeteo)
1: Clear, Few clouds, Partly cloudy, Partly cloudy
2: Mist + Cloudy, Mist + Broken clouds, Mist + Few clouds, Mist
3: Light Snow, Light Rain + Thunderstorm + Scattered clouds, Light Rain + Scattered
clouds
4: Heavy Rain + Ice Pallets + Thunderstorm + Mist, Snow + Fog
temp: Normalized temperature in Celsius. The values are derived via (t-t_min)/(t_max-t_min),
t_min=-8, t_max=+39 (only in hourly scale)

atemp: Normalized feeling temperature in Celsius. The values are derived via
(t-t_min)/(t_maxt_min), t_min=-16, t_max=+50 (only in hourly scale)
hum: Normalized humidity. The values are divided to 100 (max)
windspeed: Normalized wind speed. The values are divided to 67 (max)
casual: count of casual users
registered: count of registered users
cnt: count of total rental bikes including both casual and registered

As a first step lets do three simple steps on the dataset

- Size of the dataset
- Get a glimpse of data by printing few rows of it.
- What type of variables contribute our data

Shape Of The Dataset

```
dailyData.shape
```

```
(731, 16)
```

Sample Of First Few Rows

```
dailyData.head(2)
```

	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

Variables Data Type ¶

```
dailyData.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
```

Methodology

2.1 Feature Engineering

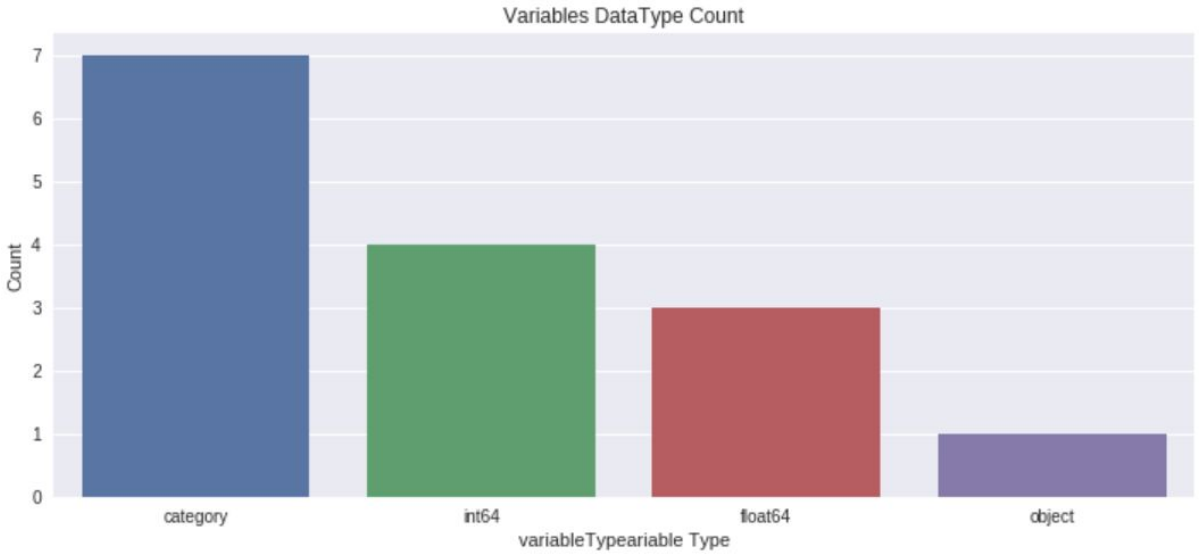
The columns "season","holiday","workingday" and "weathersit" should be of "categorical" data type.But the current data type is "int" for those columns. Let us transform the dataset in the following ways so that we can get started up with our EDA

- Coerce the datatype of "season","holiday","workingday" and weather to category.

Coercing To Category Type

```
categoryVariableList = ["weekday","month","season","weather","holiday","workingday"]
for var in categoryVariableList:
    dailyData[var] = dailyData[var].astype("category")
```

Visualization Of Variables DataType Count



2.2 Missing Value Analysis

Missing values can be handled in different ways:

- Remove entire rows with missing values
- Derive missing values from other variables
- Replace missing values with mean of that variable
- Replace missing value with median value of that variable

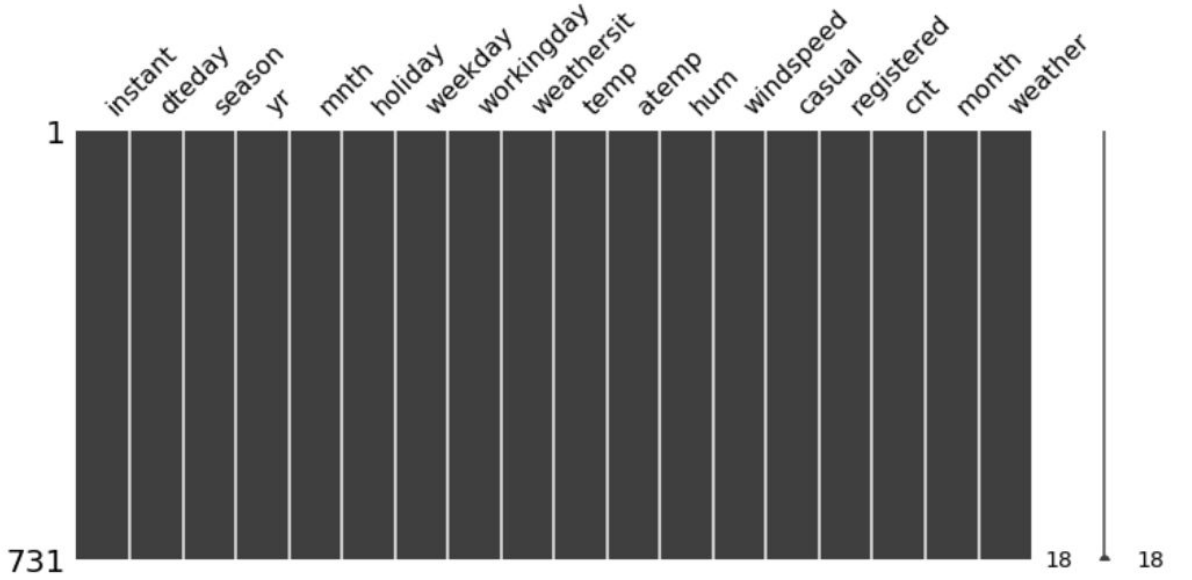
Once we get the hang of the data and columns, next step we generally take is to find out whether we have any missing values in our data. Luckily we don't have any missing values in the dataset. One way which I generally prefer to visualize missing values in the dataset is through "missingno".

It's a quiet handy library to quickly visualize variables for missing values. As I mentioned earlier we got lucky this time as there are no missing values in the dataset.

Skewness In Distribution

```
msno.matrix(dailyData,figsize=(12,5))
```

```
<matplotlib.axes._subplots.AxesSubplot at 0xb1d1765ef0>
```



```
### change dtype of fare_amount to float
train_df['fare_amount'] = train_df['fare_amount'].values.astype(np.float64)
```

```
#Remove rows fare_amount less than 0 and passenger count less than 0
train_df = train_df[(train_df['fare_amount'] > 0) & (train_df['passenger_count'] >= 1)]
test_df = test_df[test_df['passenger_count'] > 0]
```

```
train_df.count()
```

```
fare_amount          15980
pickup_datetime      15980
pickup_longitude     15980
pickup_latitude      15980
dropoff_longitude    15980
dropoff_latitude     15980
passenger_count      15980
dtype: int64
```

2.3 Outlier Analysis

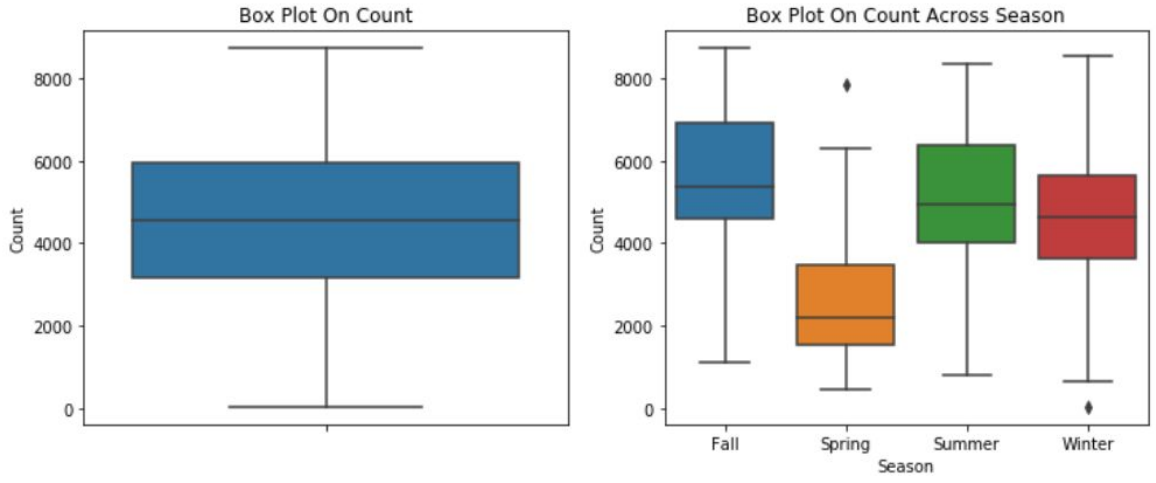
In statistics, an **outlier** is an observation point that is distant from other observations.
The above definition suggests that outlier is something which is separate/different from the crowd.
Outliers can skew the data so it is good to remove it before training the model.
For detection of outliers we can use mathematical methods, like z-scores, IQR, and graphical methods like box plots and scatter plots.
For Detection we have used boxplots and for outlier removal we have used IQR.

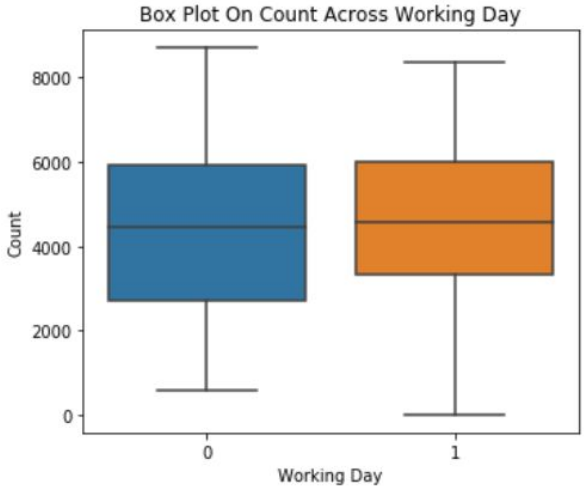
The **interquartile range** (**IQR**), also called the **midspread** or **middle 50%**, or technically **H-spread**, is a measure of statistical dispersion, being equal to the difference between 75th and 25th percentiles, or between upper and lower quartiles, IQR = *Q*3 − *Q*1.In other words, the IQR is the first quartile subtracted from the third quartile; these quartiles can be clearly seen on a box plot on the data.

It is a measure of the dispersion similar to standard deviation or variance, but is much more robust against outliers.

At first look, "count" variable contains lot of outlier data points which skews the distribution towards right (as there are more data points beyond Outer Quartile Limit).But in addition to that, following inferences can also be made from the simple box plots given below:

- Spring season has got relatively lower count.The dip in median value in boxplot gives evidence for it.
- Most of the outlier points are mainly contributed from "Working Day" than "Non Working Day".



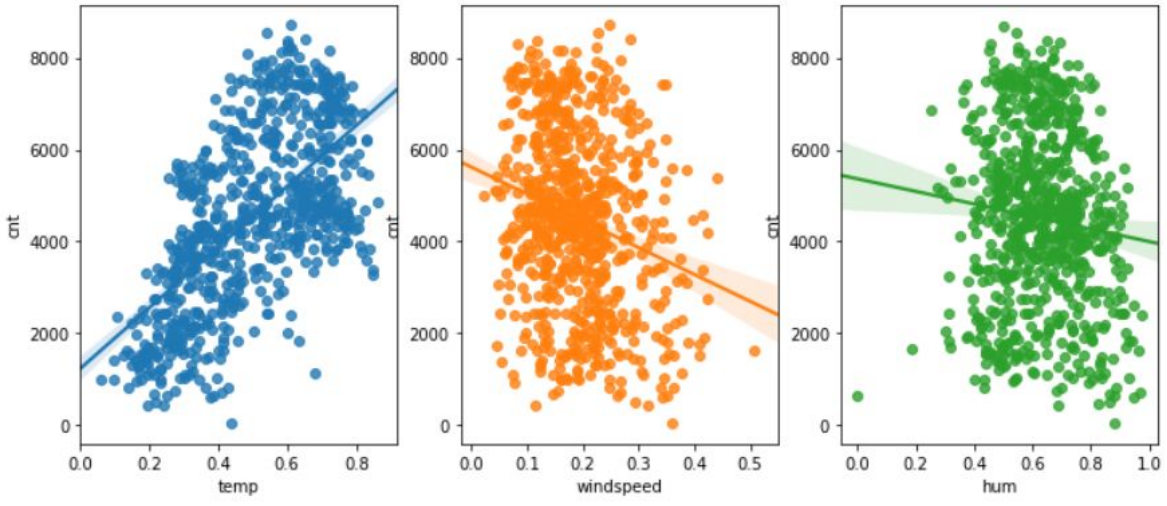


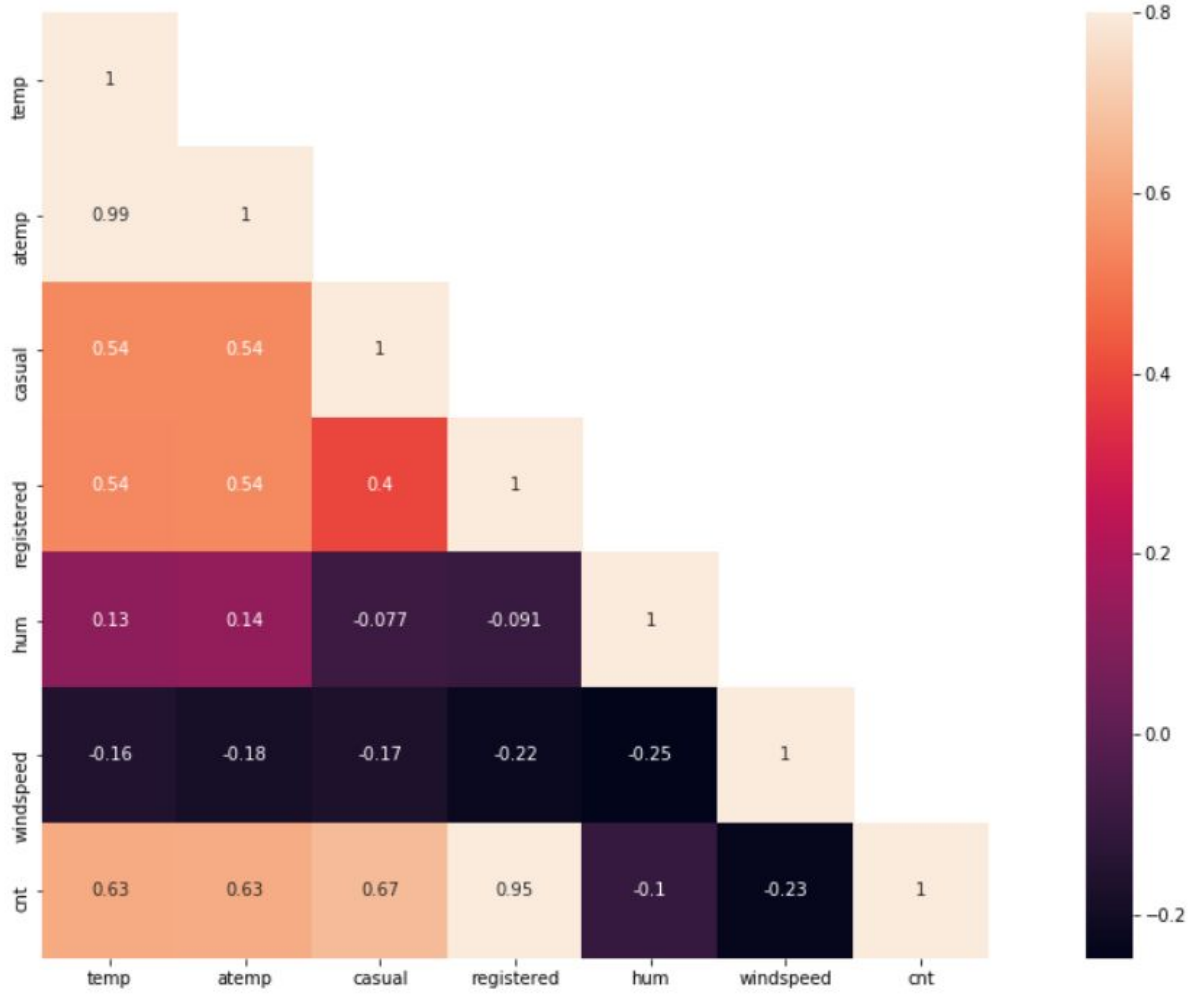
2.4 Correlation Analysis

One common to understand how a dependent variable is influenced by features (numerical) is to find a correlation matrix between them. Lets plot a correlation plot between "count" and ["temp","atemp","humidity","windspeed"].

- temp and humidity features has got positive and negative correlation with count respectively.Although the correlation between them are not very prominent still the count variable has little dependence on "temp" and "humidity".
- wind speed is not gonna be really useful numerical feature and it is visible from it correlation value with "count"
- "atemp" is variable is not taken into since "atemp" and "temp" has got a strong correlation with each other. During model building any one of the variables has to be dropped since they will exhibit multicollinearity in the data.
- "Casual" and "Registered" are also not taken into account since they are leakage variables in nature and need to be dropped during model building.

Regression plot in seaborn is one useful way to depict the relationship between two features. Here we consider "count" vs "temp", "humidity", "windspeed".




temp
atemp
casual
registered
hum
windspeed
cnt
1
0.99 1
0.54 0.54 1
0.54 0.54 0.4 1
0.13 0.14 -0.077 -0.091 1
-0.16 -0.18 -0.17 -0.22 -0.25 1
0.63 0.63 0.67 0.95 -0.1 -0.23 1
temp
atemp
casual
registered
hum
windspeed
cnt
0.8
0.6
0.4
0.2
0.0
−0.2

2.5 Model Development

After wrangling, exploring, analysing and preprocessing the data now it's finally time to model the data. As discussed earlier this is a regression problem where we want to generate a model that can predict car fares. We will use three models:

1. Multiple Linear Regression Model
2. Random Forest Model
3. Gradient Boosting Model

2.5.1 Multiple Linear Regression Model

Multiple linear regression (MLR), also known simply as multiple regression is a statistical technique that uses several explanatory variables to predict the outcome of a response variable. The goal of multiple linear regression (MLR) is to model the linear regression between the explanatory (independent) variables and response (dependent) variable.

The Formula for Multiple Linear Regression Is

$$y_i = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + ... + \beta_p x_{ip} + \epsilon$$

where, for $i = n$ observations:

$y_i = \text{dependent variable}$

$x_i = \text{expanatory variables}$

$\beta_0 = \text{y-intercept (constant term)}$

$\beta_p = \text{slope coefficients for each explanatory variable}$

$\epsilon = \text{the model's error term (also known as the residuals)}$

Linear Regression Model

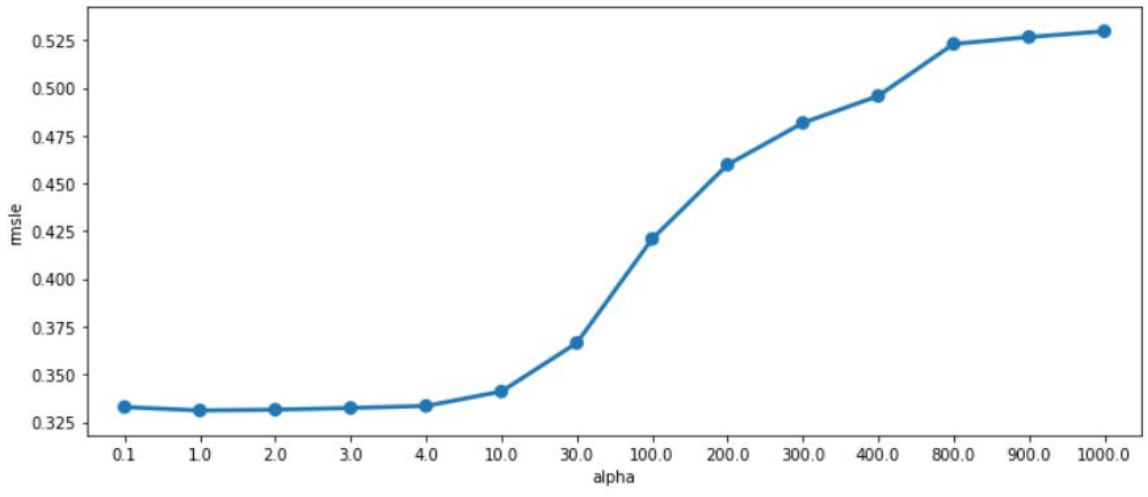
```
from sklearn.linear_model import LinearRegression,Ridge,Lasso
from sklearn.model_selection import GridSearchCV
from sklearn import metrics
import warnings
pd.options.mode.chained_assignment = None
warnings.filterwarnings("ignore", category=DeprecationWarning)

# Initialize logistic regression model
lModel = LinearRegression()

# Train the model
yLabelsLog = np.log1p(yLabels)
lModel.fit(X = dataTrain,y = yLabelsLog)

# Make predictions
preds = lModel.predict(X= dataTrain)
print ("RMSLE Value For Linear Regression: ",rmsle(np.exp(yLabelsLog),np.exp(preds),False))
```

```
RMSLE Value For Linear Regression:  0.33521788718861467
```

Regularization Model - Ridge

```
ridge_m_ = Ridge()
ridge_params_ = { 'max_iter':[3000],'alpha':[0.1, 1, 2, 3, 4, 10, 30,100,200,300,400,800,900,1000]}
rmsle_scorer = metrics.make_scorer(rmsle, greater_is_better=False)
grid_ridge_m = GridSearchCV( ridge_m_,
                            ridge_params_,
                            scoring = rmsle_scorer,
                            cv=5,
                            return_train_score=True)
yLabelsLog = np.log1p(yLabels)
grid_ridge_m.fit( dataTrain, yLabelsLog )
preds = grid_ridge_m.predict(X= dataTrain)
print (grid_ridge_m.best_params_)
print ("RMSLE Value For Ridge Regression: ",rmsle(np.exp(yLabelsLog),np.exp(preds),False))

fig,ax= plt.subplots()
fig.set_size_inches(12,5)
df = pd.DataFrame(grid_ridge_m.cv_results_)
df["alpha"] = df["params"].apply(lambda x:x["alpha"])
df["rmsle"] = df["mean_test_score"].apply(lambda x:-x)
sn.pointplot(data=df,x="alpha",y="rmsle",ax=ax)
```

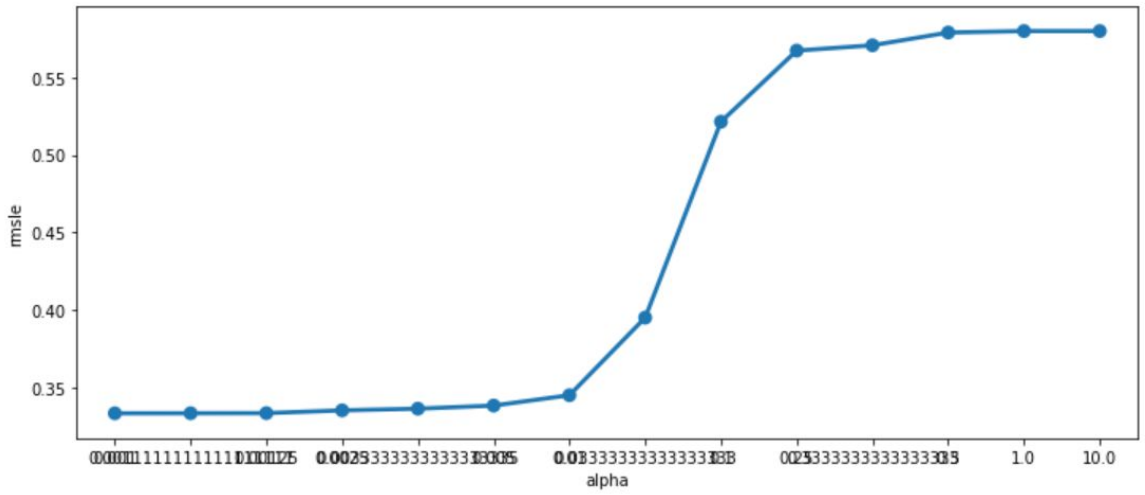
```
{'alpha': 1, 'max_iter': 3000}
RMSLE Value For Ridge Regression:  0.3360321720078531
```



Regularization Model - Lasso ¶

```
lasso_m_ = Lasso()

alpha  = 1/np.array([0.1, 1, 2, 3, 4, 10, 30,100,200,300,400,800,900,1000])
lasso_params_ = { 'max_iter':[3000],'alpha':alpha}

grid_lasso_m = GridSearchCV( lasso_m_,lasso_params_,scoring = rmsle_scorer,cv=5, return_train_score=True)
yLabelsLog = np.log1p(yLabels)
grid_lasso_m.fit( dataTrain, yLabelsLog )
preds = grid_lasso_m.predict(X= dataTrain)
print (grid_lasso_m.best_params_)
print ("RMSLE Value For Lasso Regression: ",rmsle(np.exp(yLabelsLog),np.exp(preds),False))

fig,ax= plt.subplots()
fig.set_size_inches(12,5)
df = pd.DataFrame(grid_lasso_m.cv_results_)
df["alpha"] = df["params"].apply(lambda x:x["alpha"])
df["rmsle"] = df["mean_test_score"].apply(lambda x:-x)
sn.pointplot(data=df,x="alpha",y="rmsle",ax=ax)
```

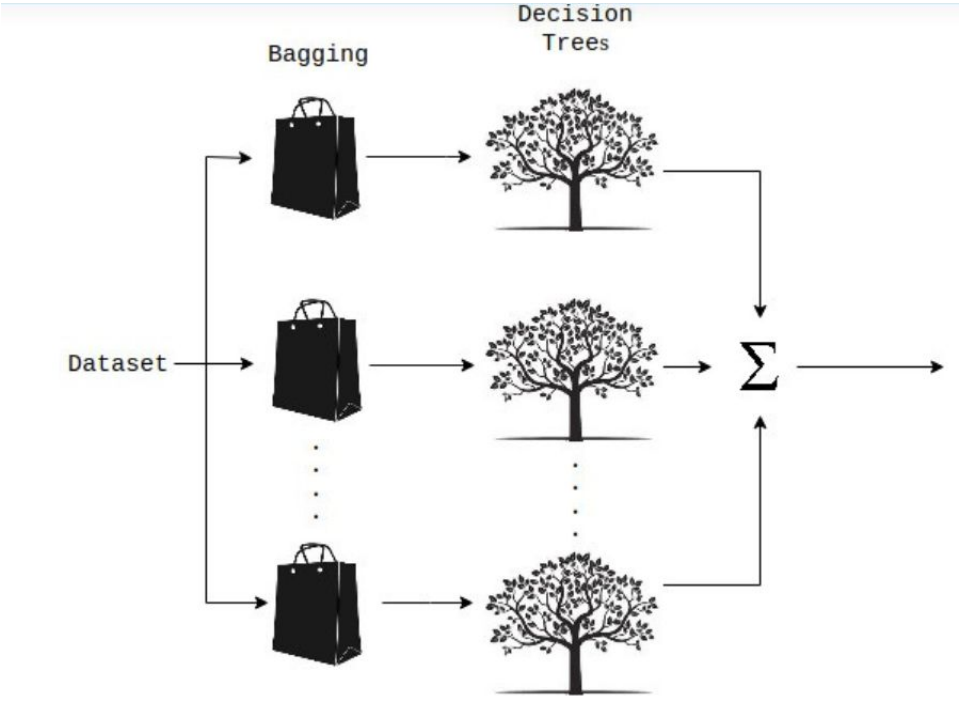
```
{'alpha': 0.001, 'max_iter': 3000}
RMSLE Value For Lasso Regression:  0.33582638744647086
```



2.5.2 Random Forest

A random forest is a meta estimator that fits a number of classifying decision trees on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting.

A Random Forest is an ensemble technique capable of performing both regression and classification tasks with the use of multiple decision trees and a technique called **Bootstrap Aggregation**, commonly known as **bagging**. Bagging, in the Random Forest method, involves training each decision tree on a different data sample where sampling is done with replacement.



Random Forest Regression: Process

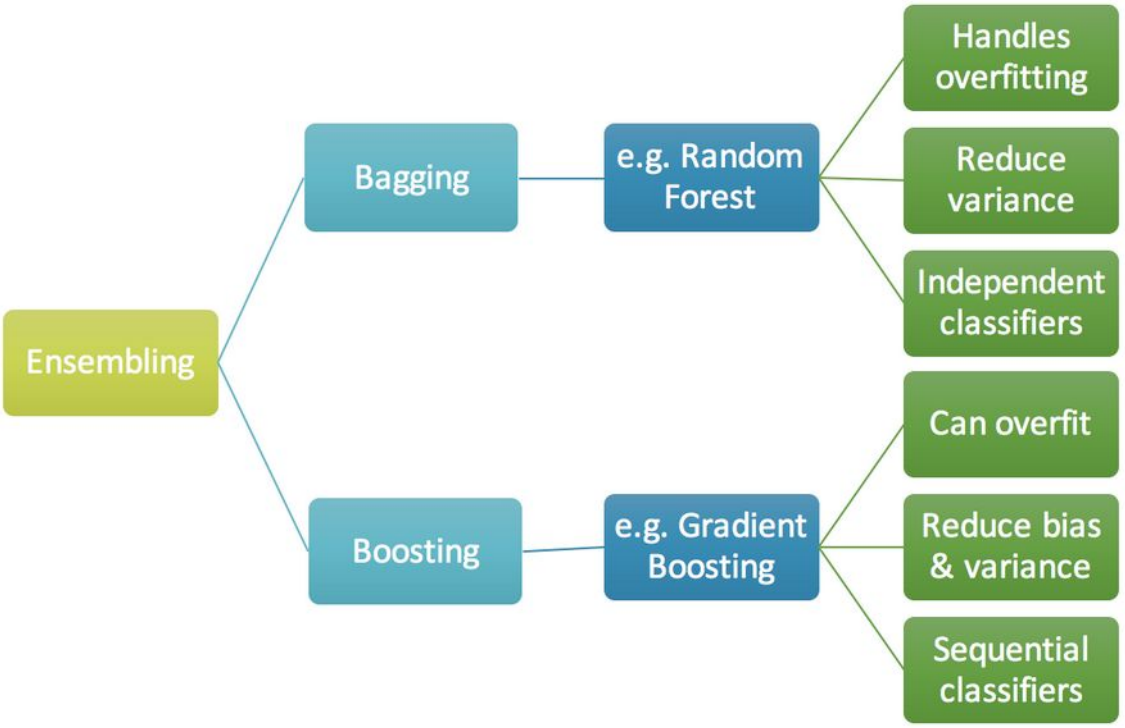
Ensemble Models - Random Forest

```
from sklearn.ensemble import RandomForestRegressor
rfModel = RandomForestRegressor(n_estimators=500)
yLabelsLog = np.log1p(yLabels)
rfModel.fit(dataTrain,yLabelsLog)
preds = rfModel.predict(X= dataTrain)
print ("RMSLE Value For Random Forest: ",rmsle(np.exp(yLabelsLog),np.exp(preds),False))
```

```
RMSLE Value For Random Forest:  0.10970537226166112
```

2.5.3 Gradient Boosting

Gradient boosting is a machine learning technique for regression and classification problems, which produces a prediction model in the form of an ensemble of weak prediction models, typically decision trees.



Ensemble Model - Gradient Boost

```
from sklearn.ensemble import GradientBoostingRegressor
gbm = GradientBoostingRegressor(n_estimators=500,alpha=0.01); ### Test 0.41
yLabelsLog = np.log1p(yLabels)
gbm.fit(dataTrain,yLabelsLog)
preds = gbm.predict(X= dataTrain)
print ("RMSLE Value For Gradient Boost: ",rmsle(np.exp(yLabelsLog),np.exp(preds),False))
```

```
RMSLE Value For Gradient Boost:  0.038430691022472034
```

Conclusion

3.1 Model Evaluation

Mean Absolute Error (MAE) and Root mean squared error (RMSE) are two of the most common metrics used to measure accuracy for continuous variables.

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

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

Root mean squared error (RMSE): RMSE is a quadratic scoring rule that also measures the average magnitude of the error. It's the square root of the average of squared differences between prediction and actual observation.

$$\mathrm{RMSE} = \sqrt{\frac{1}{n}\sum_{j=1}^{n}(y_j - \hat{y}_j)^2}$$

Taking the square root of the average squared errors has some interesting implications for RMSE. Since the errors are squared before they are averaged, the RMSE gives a relatively high weight to large

errors. This means the RMSE should be more useful when large errors are particularly undesirable.

Hence, we will use Root Mean Square Error (RMSE) as the evaluation metric as it fits our case and we want to avoid large errors.

Root Mean Square Log Error (RMSLE):
When we see the formulation of the RMSE, it just looks like a difference of a log function. In reality, that small difference of log is the primary factor that gives RMSLE the unique properties of its own.

$$\sqrt{\frac{1}{n}\sum_{i=1}^{n}(\log(x_i+1) - \log(y_i+1))^2}$$

Relative Error

If we only consider the internal part of the RMLSE, we find that it is fundamentally a calculation relative error.

$$\log(x_i+1) - \log(y_i+1) = \frac{\log(x_i+1)}{\log(y_i+1)}$$

In the case of RMSE, the presence of outliers can explode the error term to a very high value. But, in the case of RMLSE the outliers are drastically scaled down therefore nullifying their effect.

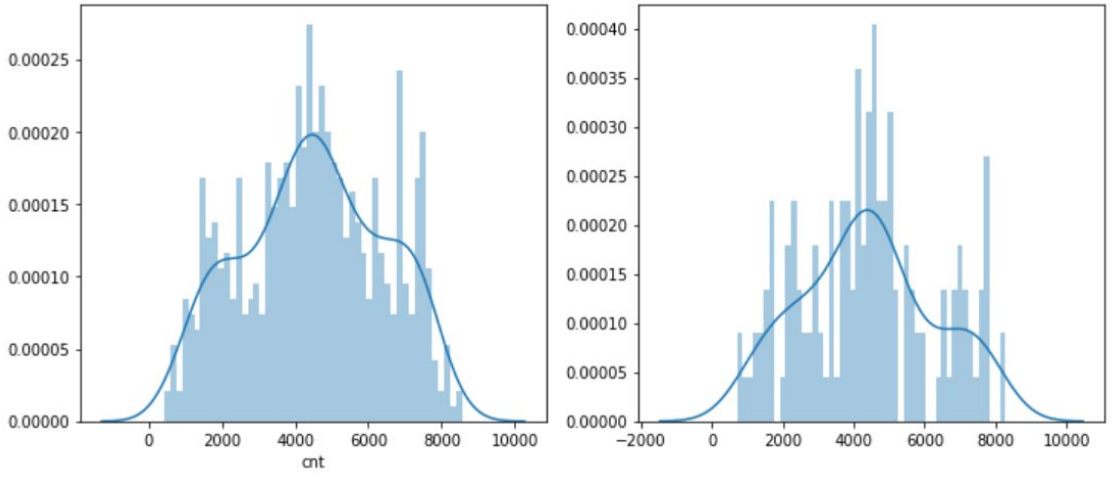
So, we will use RMSLE as our evaluation metric.

RMSLE Scorer

```
def rmsle(y, y_,convertExp=True):
    if convertExp:
        y = np.exp(y),
        y_ = np.exp(y_)
    log1 = np.nan_to_num(np.array([np.log(v + 1) for v in y]))
    log2 = np.nan_to_num(np.array([np.log(v + 1) for v in y_]))
    calc = (log1 - log2) ** 2
    return np.sqrt(np.mean(calc))
```

```
RMSLE Value For Linear Regression:  0.266715736758359

{'alpha': 0.1, 'max_iter': 3000}
RMSLE Value For Ridge Regression:  0.26804294612421037

{'alpha': 0.001, 'max_iter': 3000}
RMSLE Value For Lasso Regression:  0.26907374271311

RMSLE Value For Random Forest:  0.08317392332306514

RMSLE Value For Gradient Boost:  0.033400497679542444
```

Lets compare the distribution of train and test results. More or less the distribution of train and test looks identical. It confirms visually that our model has not predicted really bad and not suffering from major overfitting problem.



So, we will select Gradient Boost model.

Annexure

Python Code:

Bike Rent

In [1]:
```
import pylab
import calendar
import numpy as np
import pandas as pd
import seaborn as sn
from scipy import stats
import missingno as msno
from datetime import datetime
import matplotlib.pyplot as plt
import warnings
pd.options.mode.chained_assignment = None
warnings.filterwarnings("ignore", category=DeprecationWarning)
%matplotlib inline
```

Lets Read In The Dataset

In [2]:
```
dailyData = pd.read_csv("day.csv")
```

Data Summary

As a first step lets do three simple steps on the dataset

- Size of the dataset
- Get a glimpse of data by printing few rows of it.
- What type of variables contribute our data

Shape Of The Dataset

In [3]:
```
dailyData.shape
```

Out[3]:
```
(731, 16)
```

Sample Of First Few Rows

In [4]:
```
dailyData.head(2)
```

Out[4]:

	instant	dteday	season	yr	mnth	holiday	weekday	workingday	weathersit	temp	at
0	1	2011-01-01	1	0	1	0	6	0	2	0.344167	0.36
1	2	2011-01-02	1	0	1	0	0	0	2	0.363478	0.35

Variables Data Type

In [5]:
```
dailyData.dtypes
```

Out[5]:
```
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
```

Feature Engineering

As we see from the above results, the columns "season","holiday","workingday" and "weather" should be of "categorical" data type.But the current data type is "int" for those columns. Let us transform the dataset in the following ways so that we can get started up with our EDA

- Coerce the datatype of "season","holiday","workingday" and weather to category.

Creating New Columns From "Datetime" Column

In [6]:
```
dailyData["weekday"] = dailyData.dteday.apply(lambda dateString : calendar.day
_name[datetime.strptime(dateString,"%Y-%m-%d").weekday()])
dailyData["month"] = dailyData.dteday.apply(lambda dateString : calendar.month
_name[datetime.strptime(dateString,"%Y-%m-%d").month])
dailyData["season"] = dailyData.season.map({1: "Spring", 2 : "Summer", 3 : "Fa
ll", 4 :"Winter" })
dailyData["weather"] = dailyData.weathersit.map({1: " Clear + Few clouds + Par
tly cloudy + Partly cloudy",\
                                                 2 : " Mist + Cloudy, Mist + Broken clo
uds, Mist + Few clouds, Mist ", \
                                                 3 : " Light Snow, Light Rain + Thunder
storm + Scattered clouds, Light Rain + Scattered clouds", \
                                                 4 :" Heavy Rain + Ice Pallets + Thunde
rstorm + Mist, Snow + Fog " })
```

Coercing To Category Type

In [7]:
```
categoryVariableList = ["weekday","month","season","weather","holiday","workin
gday"]
for var in categoryVariableList:
    dailyData[var] = dailyData[var].astype("category")
```

In [8]:
```
dailyData.dtypes
```

Out[8]:
```
instant          int64
dteday          object
season        category
yr               int64
mnth             int64
holiday       category
weekday       category
workingday    category
weathersit       int64
temp           float64
atemp          float64
hum            float64
windspeed      float64
casual           int64
registered       int64
cnt              int64
month         category
weather       category
dtype: object
```

Missing Values Analysis

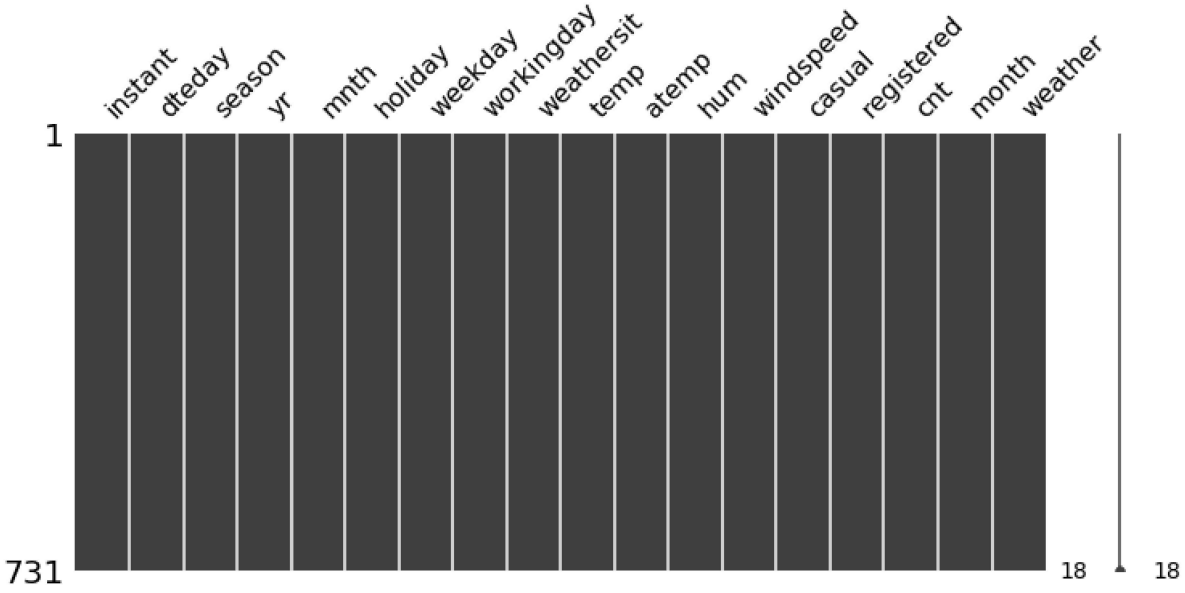
Once we get hang of the data and columns, next step we generally take is to find out whether we have any missing values in our data. Luckily we dont have any missing value in the dataset. One way which I generally prefer to visualize missing value in the dataset is through "missingno".

Its a quiet handy library to quickly visualize variables for missing values. As I mentioned earlier we got lucky this time as there no missing value in the dataset.

Skewness In Distribution

```
In [9]: msno.matrix(dailyData,figsize=(12,5))
```

```
Out[9]: <matplotlib.axes._subplots.AxesSubplot at 0x534b98cc50>
```



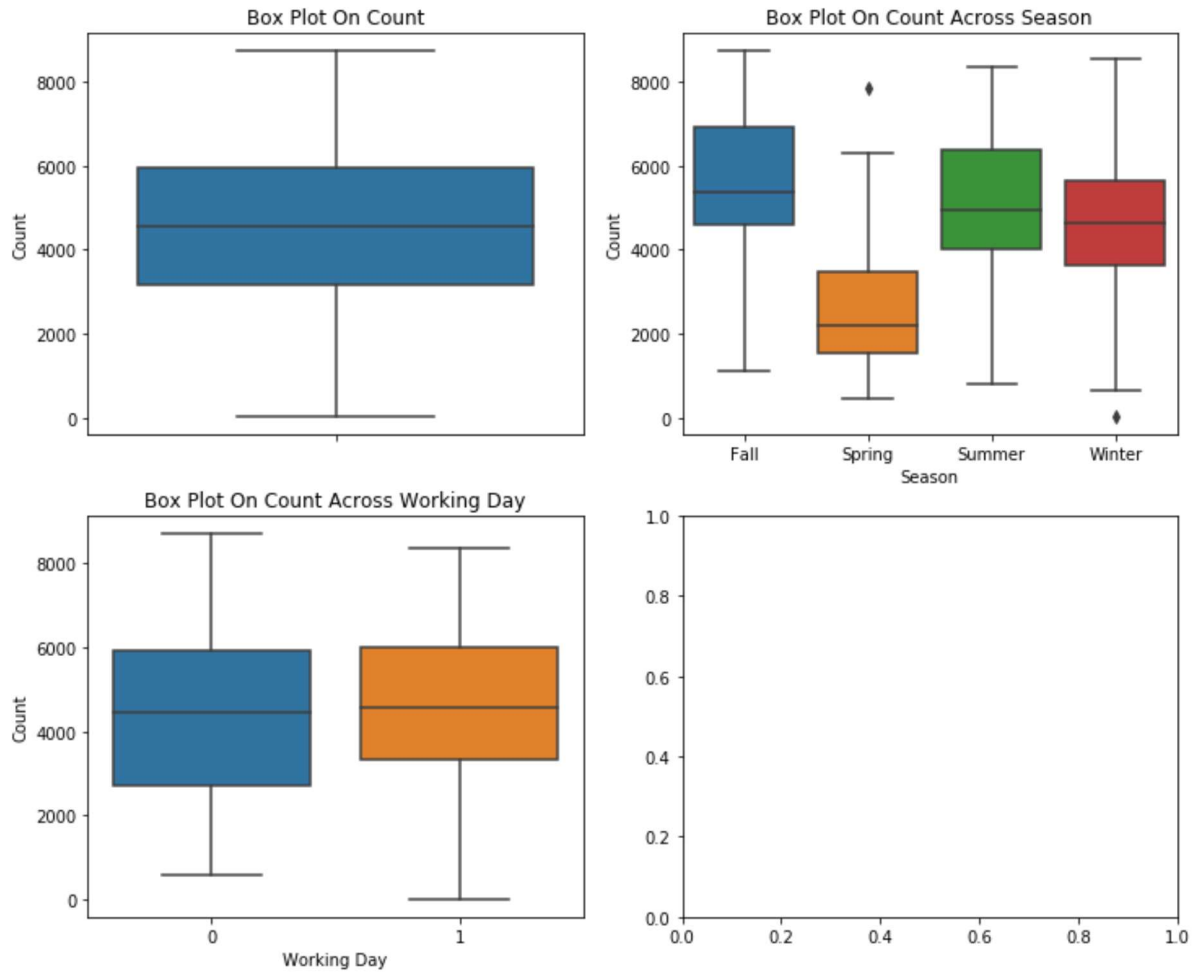
Outliers Analysis

At first look, "count" variable contains lot of outlier data points which skews the distribution towards right (as there are more data points beyond Outer Quartile Limit).But in addition to that, following inferences can also been made from the simple boxplots given below.

- Spring season has got relatively lower count.The dip in median value in boxplot gives evidence for it.
- Most of the outlier points are mainly contributed from "Working Day" than "Non Working Day".

In [10]:
```
fig, axes = plt.subplots(nrows=2,ncols=2)
fig.set_size_inches(12, 10)
sn.boxplot(data=dailyData,y="cnt",orient="v",ax=axes[0][0])
sn.boxplot(data=dailyData,y="cnt",x="season",orient="v",ax=axes[0][1])
sn.boxplot(data=dailyData,y="cnt",x="workingday",orient="v",ax=axes[1][0])

axes[0][0].set(ylabel='Count',title="Box Plot On Count")
axes[0][1].set(xlabel='Season', ylabel='Count',title="Box Plot On Count Across Season")
axes[1][0].set(xlabel='Working Day', ylabel='Count',title="Box Plot On Count Across Working Day")
```

Out[10]:
```
[Text(0, 0.5, 'Count'),
 Text(0.5, 0, 'Working Day'),
 Text(0.5, 1.0, 'Box Plot On Count Across Working Day')]
```



Lets Remove Outliers In The Count Column

In [11]:
```
dailyDataWithoutOutliers = dailyData[np.abs(dailyData["cnt"]-dailyData["cnt"].mean())<=(3*dailyData["cnt"].std())]
```

In [12]:
```
print ("Shape Of The Before Ouliers: ",dailyData.shape)
print ("Shape Of The After Ouliers: ",dailyDataWithoutOutliers.shape)
```

```
Shape Of The Before Ouliers:  (731, 18)
Shape Of The After Ouliers:  (731, 18)
```

Correlation Analysis

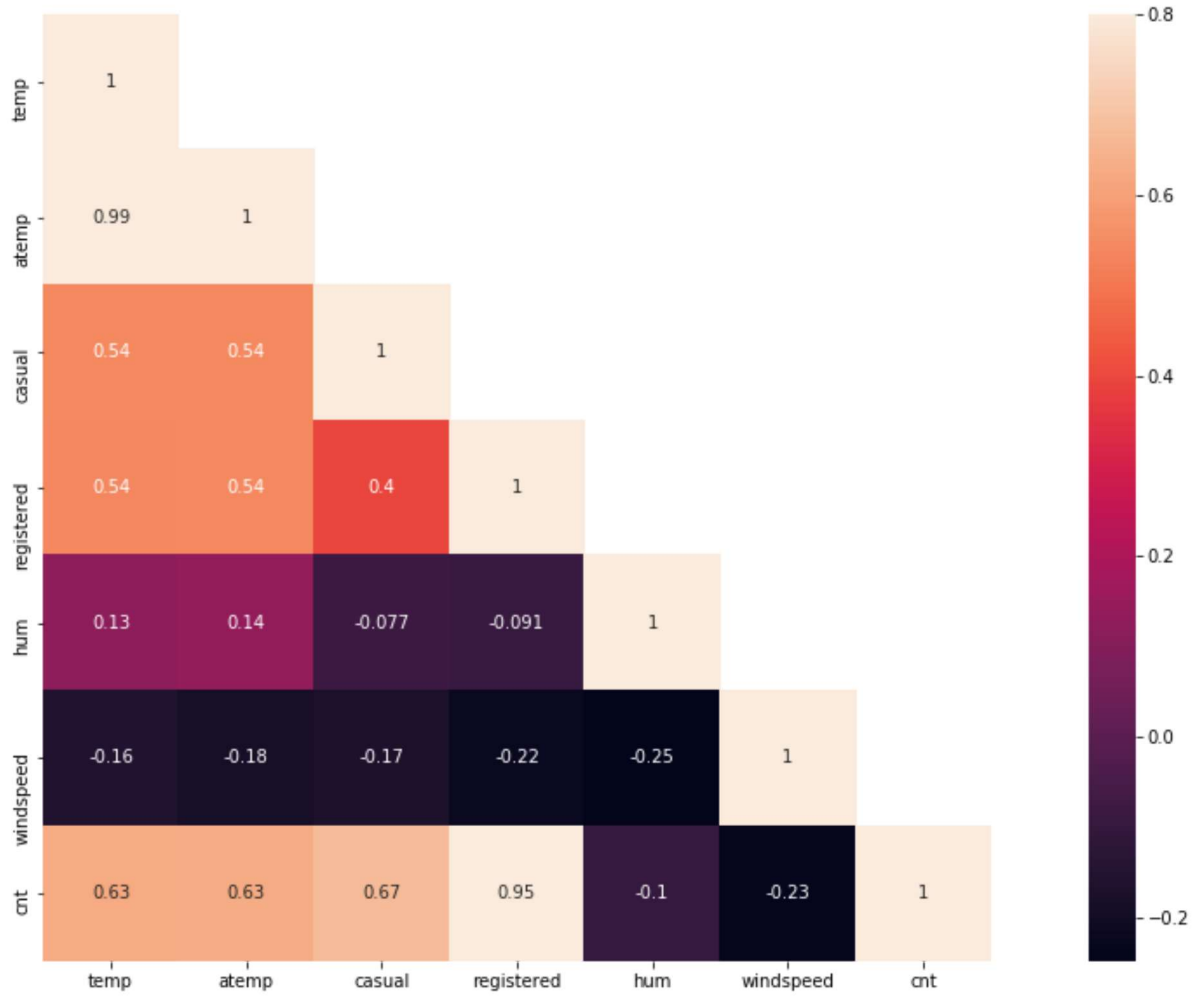
One common to understand how a dependent variable is influenced by features (numerical) is to fibd a correlation matrix between them. Lets plot a correlation plot between "count" and ["temp","atemp","humidity","windspeed"].

- temp and humidity features has got positive and negative correlation with count respectively.Although the correlation between them are not very prominent still the count variable has got little dependency on "temp" and "humidity".
- windspeed is not gonna be really useful numerical feature and it is visible from it correlation value with "count"
- "atemp" is variable is not taken into since "atemp" and "temp" has got strong correlation with each other. During model building any one of the variable has to be dropped since they will exhibit multicollinearity in the data.
- "Casual" and "Registered" are also not taken into account since they are leakage variables in nature and need to dropped during model building.

Regression plot in seaborn is one useful way to depict the relationship between two features. Here we consider "count" vs "temp", "humidity", "windspeed".

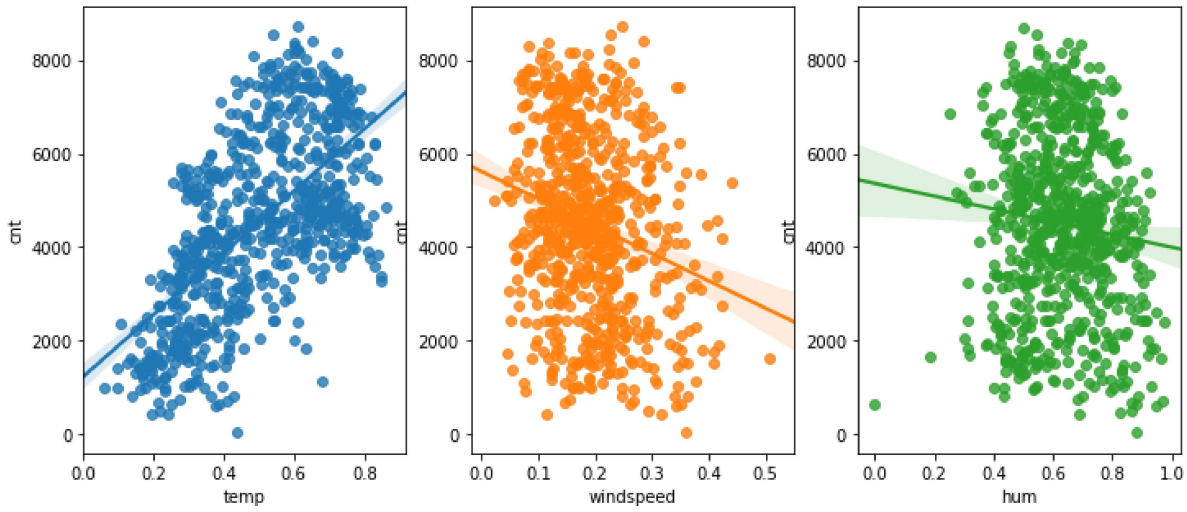
In [13]:
```
corrMatt = dailyData[["temp","atemp","casual","registered","hum","windspeed",
"cnt"]].corr()
mask = np.array(corrMatt)
mask[np.tril_indices_from(mask)] = False
fig,ax= plt.subplots()
fig.set_size_inches(20,10)
sn.heatmap(corrMatt, mask=mask,vmax=.8, square=True,annot=True)
```

Out[13]: <matplotlib.axes._subplots.AxesSubplot at 0x534c29d390>



In [14]:
```
fig,(ax1,ax2,ax3) = plt.subplots(ncols=3)
fig.set_size_inches(12, 5)
sn.regplot(x="temp", y="cnt", data=dailyData,ax=ax1)
sn.regplot(x="windspeed", y="cnt", data=dailyData,ax=ax2)
sn.regplot(x="hum", y="cnt", data=dailyData,ax=ax3)
```

Out[14]: <matplotlib.axes._subplots.AxesSubplot at 0x5340721588>



Visualizing Distribution Of Data

As it is visible from the below figures that "count" variable is skewed towards right. It is desirable to have Normal distribution as most of the machine learning techniques require dependent variable to be Normal. One possible solution is to take log transformation on "count" variable after removing outlier data points. After the transformation the data looks lot better but still not ideally following normal distribution.

In [15]:
```
fig,axes = plt.subplots(ncols=2,nrows=2)
fig.set_size_inches(12, 10)
sn.distplot(dailyData["cnt"],ax=axes[0][0])
stats.probplot(dailyData["cnt"], dist='norm', fit=True, plot=axes[0][1])
sn.distplot(np.log(dailyDataWithoutOutliers["cnt"]),ax=axes[1][0])
stats.probplot(np.log1p(dailyDataWithoutOutliers["cnt"]), dist='norm', fit=Tru
e, plot=axes[1][1])
```

Out[15]: ((array([-3.10612952, -2.83371839, -2.68121219, -2.57340905, -2.48915191,
-2.41955673, -2.36001798, -2.30782877, -2.26125818, -2.21912992,
-2.18060696, -2.14507173, -2.11205508, -2.08119197, -2.05219258,
-2.0248228 , -1.99889075, -1.97423711, -1.95072808, -1.92825019,
-1.90670633, -1.88601273, -1.8660966 , -1.84689427, -1.82834975,
-1.81041348, -1.79304141, -1.77619419, -1.75983653, -1.74393663,
-1.72846577, -1.71339788, -1.69870925, -1.68437825, -1.67038506,
-1.6567115 , -1.64334086, -1.63025771, -1.6174478 , -1.60489794,
-1.59259587, -1.58053022, -1.56869036, -1.55706641, -1.54564912,
-1.53442983, -1.52340042, -1.51255328, -1.50188124, -1.49137757,
-1.4810359 , -1.47085025, -1.46081495, -1.45092464, -1.44117426,
-1.431559 , -1.4220743 , -1.41271583, -1.40347947, -1.39436132,
-1.38535765, -1.3764649 , -1.36767969, -1.35899879, -1.35041911,
-1.3419377 , -1.33355173, -1.32525852, -1.31705546, -1.30894008,
-1.30091001, -1.29296295, -1.28509673, -1.27730922, -1.26959842,
-1.26196238, -1.25439922, -1.24690714, -1.2394844 , -1.23212934,
-1.22484033, -1.21761582, -1.21045431, -1.20335435, -1.19631454,
-1.18933352, -1.18240998, -1.17554267, -1.16873035, -1.16197185,
-1.155266 , -1.14861171, -1.14200789, -1.13545351, -1.12894754,
-1.12248901, -1.11607697, -1.10971049, -1.10338867, -1.09711064,
-1.09087556, -1.08468261, -1.07853098, -1.07241989, -1.06634859,
-1.06031635, -1.05432244, -1.04836618, -1.04244688, -1.03656388,
-1.03071654, -1.02490423, -1.01912634, -1.01338227, -1.00767144,
-1.0019933 , -0.99634727, -0.99073283, -0.98514945, -0.9795966 ,
-0.97407381, -0.96858056, -0.96311639, -0.95768082, -0.9522734 ,
-0.94689368, -0.94154123, -0.93621562, -0.93091643, -0.92564325,
-0.92039569, -0.91517335, -0.90997585, -0.90480282, -0.89965388,
-0.89452869, -0.88942689, -0.88434814, -0.87929209, -0.87425842,
-0.86924681, -0.86425694, -0.85928849, -0.85434116, -0.84941466,
-0.84450869, -0.83962296, -0.83475719, -0.8299111 , -0.82508442,
-0.8202769 , -0.81548825, -0.81071823, -0.80596659, -0.80123308,
-0.79651745, -0.79181947, -0.78713889, -0.7824755 , -0.77782907,
-0.77319937, -0.76858618, -0.76398929, -0.75940848, -0.75484356,
-0.75029432, -0.74576055, -0.74124205, -0.73673864, -0.73225012,
-0.72777631, -0.72331702, -0.71887206, -0.71444126, -0.71002444,
-0.70562143, -0.70123206, -0.69685616, -0.69249356, -0.6881441 ,
-0.68380762, -0.67948396, -0.67517297, -0.67087448, -0.66658836,
-0.66231445, -0.6580526 , -0.65380267, -0.64956452, -0.64533801,
-0.64112299, -0.63691932, -0.63272689, -0.62854555, -0.62437516,
-0.62021561, -0.61606676, -0.61192849, -0.60780067, -0.60368318,
-0.59957591, -0.59547872, -0.5913915 , -0.58731414, -0.58324652,
-0.57918853, -0.57514005, -0.57110098, -0.5670712 , -0.56305061,
-0.5590391 , -0.55503657, -0.55104291, -0.54705802, -0.5430818 ,
-0.53911414, -0.53515496, -0.53120414, -0.5272616 , -0.52332724,
-0.51940096, -0.51548267, -0.51157228, -0.5076697 , -0.50377484,
-0.4998876 , -0.4960079 , -0.49213565, -0.48827077, -0.48441317,
-0.48056276, -0.47671946, -0.4728832 , -0.46905388, -0.46523142,
-0.46141575, -0.45760679, -0.45380445, -0.45000867, -0.44621936,
-0.44243644, -0.43865984, -0.43488949, -0.43112532, -0.42736724,
-0.42361519, -0.41986909, -0.41612887, -0.41239447, -0.40866581,
-0.40494282, -0.40122544, -0.39751359, -0.39380721, -0.39010623,
-0.38641059, -0.38272022, -0.37903506, -0.37535503, -0.37168008,
-0.36801015, -0.36434516, -0.36068506, -0.35702979, -0.35337928,
-0.34973347, -0.34609231, -0.34245573, -0.33882367, -0.33519607,
-0.33157289, -0.32795405, -0.3243395 , -0.32072918, -0.31712304,
-0.31352101, -0.30992305, -0.3063291 , -0.3027391 , -0.299153 ,
-0.29557074, -0.29199227, -0.28841753, -0.28484648, -0.28127906,

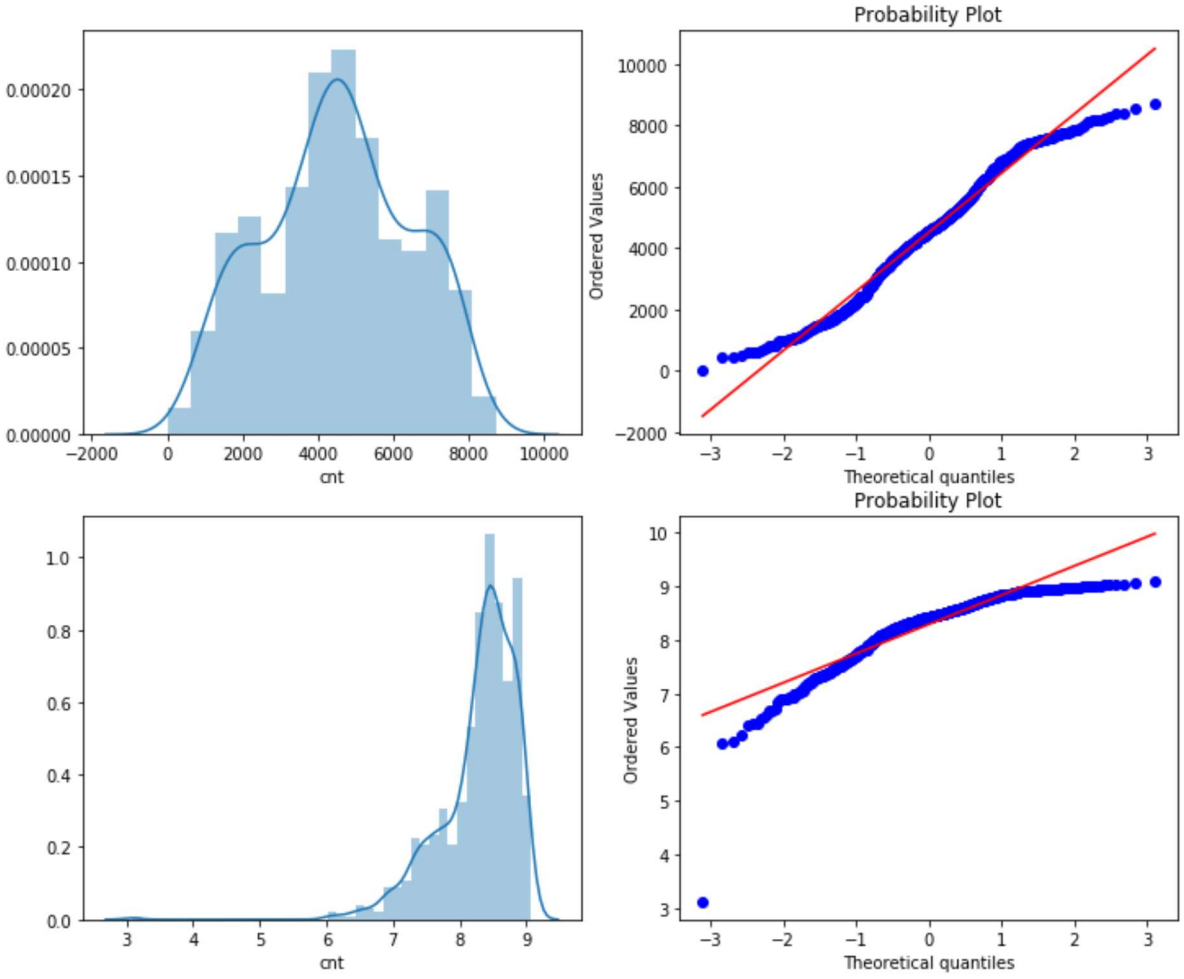
```
-0.27771521, -0.27415488, -0.27059803, -0.2670446 , -0.26349453,
-0.25994779, -0.25640431, -0.25286405, -0.24932695, -0.24579297,
-0.24226206, -0.23873416, -0.23520924, -0.23168723, -0.2281681 ,
-0.22465178, -0.22113825, -0.21762744, -0.21411931, -0.21061382,
-0.20711091, -0.20361054, -0.20011267, -0.19661724, -0.19312421,
-0.18963354, -0.18614518, -0.18265908, -0.17917519, -0.17569349,
-0.17221391, -0.16873641, -0.16526095, -0.16178749, -0.15831598,
-0.15484638, -0.15137863, -0.14791271, -0.14444857, -0.14098615,
-0.13752543, -0.13406635, -0.13060888, -0.12715297, -0.12369857,
-0.12024565, -0.11679416, -0.11334407, -0.10989532, -0.10644788,
-0.1030017 , -0.09955675, -0.09611298, -0.09267034, -0.08922881,
-0.08578833, -0.08234887, -0.07891038, -0.07547282, -0.07203616,
-0.06860034, -0.06516534, -0.0617311 , -0.05829759, -0.05486477,
-0.0514326 , -0.04800103, -0.04457003, -0.04113955, -0.03770955,
-0.03428   , -0.03085085, -0.02742207, -0.0239936 , -0.02056542,
-0.01713748, -0.01370974, -0.01028217, -0.00685471, -0.00342734,
 0.        ,  0.00342734,  0.00685471,  0.01028217,  0.01370974,
 0.01713748,  0.02056542,  0.0239936 ,  0.02742207,  0.03085085,
 0.03428   ,  0.03770955,  0.04113955,  0.04457003,  0.04800103,
 0.0514326 ,  0.05486477,  0.05829759,  0.0617311 ,  0.06516534,
 0.06860034,  0.07203616,  0.07547282,  0.07891038,  0.08234887,
 0.08578833,  0.08922881,  0.09267034,  0.09611298,  0.09955675,
 0.1030017 ,  0.10644788,  0.10989532,  0.11334407,  0.11679416,
 0.12024565,  0.12369857,  0.12715297,  0.13060888,  0.13406635,
 0.13752543,  0.14098615,  0.14444857,  0.14791271,  0.15137863,
 0.15484638,  0.15831598,  0.16178749,  0.16526095,  0.16873641,
 0.17221391,  0.17569349,  0.17917519,  0.18265908,  0.18614518,
 0.18963354,  0.19312421,  0.19661724,  0.20011267,  0.20361054,
 0.20711091,  0.21061382,  0.21411931,  0.21762744,  0.22113825,
 0.22465178,  0.2281681 ,  0.23168723,  0.23520924,  0.23873416,
 0.24226206,  0.24579297,  0.24932695,  0.25286405,  0.25640431,
 0.25994779,  0.26349453,  0.2670446 ,  0.27059803,  0.27415488,
 0.27771521,  0.28127906,  0.28484648,  0.28841753,  0.29199227,
 0.29557074,  0.299153  ,  0.3027391 ,  0.3063291 ,  0.30992305,
 0.31352101,  0.31712304,  0.32072918,  0.3243395 ,  0.32795405,
 0.33157289,  0.33519607,  0.33882367,  0.34245573,  0.34609231,
 0.34973347,  0.35337928,  0.35702979,  0.36068506,  0.36434516,
 0.36801015,  0.37168008,  0.37535503,  0.37903506,  0.38272022,
 0.38641059,  0.39010623,  0.39380721,  0.39751359,  0.40122544,
 0.40494282,  0.40866581,  0.41239447,  0.41612887,  0.41986909,
 0.42361519,  0.42736724,  0.43112532,  0.43488949,  0.43865984,
 0.44243644,  0.44621936,  0.45000867,  0.45380445,  0.45760679,
 0.46141575,  0.46523142,  0.46905388,  0.4728832 ,  0.47671946,
 0.48056276,  0.48441317,  0.48827077,  0.49213565,  0.4960079 ,
 0.4998876 ,  0.50377484,  0.5076697 ,  0.51157228,  0.51548267,
 0.51940096,  0.52332724,  0.5272616 ,  0.53120414,  0.53515496,
 0.53911414,  0.5430818 ,  0.54705802,  0.55104291,  0.55503657,
 0.5590391 ,  0.56305061,  0.5670712 ,  0.57110098,  0.57514005,
 0.57918853,  0.58324652,  0.58731414,  0.5913915 ,  0.59547872,
 0.59957591,  0.60368318,  0.60780067,  0.61192849,  0.61606676,
 0.62021561,  0.62437516,  0.62854555,  0.63272689,  0.63691932,
 0.64112299,  0.64533801,  0.64956452,  0.65380267,  0.6580526 ,
 0.66231445,  0.66658836,  0.67087448,  0.67517297,  0.67948396,
 0.68380762,  0.6881441 ,  0.69249356,  0.69685616,  0.70123206,
 0.70562143,  0.71002444,  0.71444126,  0.71887206,  0.72331702,
 0.72777631,  0.73225012,  0.73673864,  0.74124205,  0.74576055,
 0.75029432,  0.75484356,  0.75940848,  0.76398929,  0.76858618,
```

```
        0.77319937,  0.77782907,  0.7824755 ,  0.78713889,  0.79181947,
        0.79651745,  0.80123308,  0.80596659,  0.81071823,  0.81548825,
        0.8202769 ,  0.82508442,  0.8299111 ,  0.83475719,  0.83962296,
        0.84450869,  0.84941466,  0.85434116,  0.85928849,  0.86425694,
        0.86924681,  0.87425842,  0.87929209,  0.88434814,  0.88942689,
        0.89452869,  0.89965388,  0.90480282,  0.90997585,  0.91517335,
        0.92039569,  0.92564325,  0.93091643,  0.93621562,  0.94154123,
        0.94689368,  0.9522734 ,  0.95768082,  0.96311639,  0.96858056,
        0.97407381,  0.9795966 ,  0.98514945,  0.99073283,  0.99634727,
        1.0019933 ,  1.00767144,  1.01338227,  1.01912634,  1.02490423,
        1.03071654,  1.03656388,  1.04244688,  1.04836618,  1.05432244,
        1.06031635,  1.06634859,  1.07241989,  1.07853098,  1.08468261,
        1.09087556,  1.09711064,  1.10338867,  1.10971049,  1.11607697,
        1.12248901,  1.12894754,  1.13545351,  1.14200789,  1.14861171,
        1.155266  ,  1.16197185,  1.16873035,  1.17554267,  1.18240998,
        1.18933352,  1.19631454,  1.20335435,  1.21045431,  1.21761582,
        1.22484033,  1.23212934,  1.2394844 ,  1.24690714,  1.25439922,
        1.26196238,  1.26959842,  1.27730922,  1.28509673,  1.29296295,
        1.30091001,  1.30894008,  1.31705546,  1.32525852,  1.33355173,
        1.3419377 ,  1.35041911,  1.35899879,  1.36767969,  1.3764649 ,
        1.38535765,  1.39436132,  1.40347947,  1.41271583,  1.4220743 ,
        1.431559  ,  1.44117426,  1.45092464,  1.46081495,  1.47085025,
        1.4810359 ,  1.49137757,  1.50188124,  1.51255328,  1.52340042,
        1.53442983,  1.54564912,  1.55706641,  1.56869036,  1.58053022,
        1.59259587,  1.60489794,  1.6174478 ,  1.63025771,  1.64334086,
        1.6567115 ,  1.67038506,  1.68437825,  1.69870925,  1.71339788,
        1.72846577,  1.74393663,  1.75983653,  1.77619419,  1.79304141,
        1.81041348,  1.82834975,  1.84689427,  1.8660966 ,  1.88601273,
        1.90670633,  1.92825019,  1.95072808,  1.97423711,  1.99889075,
        2.0248228 ,  2.05219258,  2.08119197,  2.11205508,  2.14507173,
        2.18060696,  2.21912992,  2.26125818,  2.30782877,  2.36001798,
        2.41955673,  2.48915191,  2.57340905,  2.68121219,  2.83371839,
        3.10612952]),
array([3.13549422, 6.06842559, 6.09130988, 6.228511  , 6.40687999,
       6.43615037, 6.44254017, 6.52795792, 6.55961524, 6.62671775,
       6.67959919, 6.68710861, 6.7129562 , 6.82546004, 6.86693328,
       6.88959131, 6.89365635, 6.89467004, 6.90875478, 6.91373735,
       6.91968385, 6.92165818, 6.93537045, 7.00033446, 7.00033446,
       7.00215595, 7.01031187, 7.01750614, 7.05875815, 7.05875815,
       7.06304816, 7.09423485, 7.13009851, 7.14203657, 7.17165682,
       7.18387072, 7.18690102, 7.20191632, 7.20785987, 7.215975  ,
       7.24921506, 7.25629724, 7.25981961, 7.27724773, 7.28000825,
       7.28756064, 7.2943773 , 7.29505642, 7.31055016, 7.31455283,
       7.32052696, 7.33106031, 7.33302301, 7.3336764 , 7.33758774,
       7.33888813, 7.34213173, 7.34665516, 7.35436233, 7.3714893 ,
       7.37838371, 7.38150189, 7.38212437, 7.38274645, 7.39264752,
       7.40000952, 7.40913644, 7.42892719, 7.43011414, 7.43011414,
       7.43484788, 7.44366368, 7.4460015 , 7.46565531, 7.46737107,
       7.48885296, 7.49331725, 7.49387389, 7.49997654, 7.50273821,
       7.50439156, 7.50549227, 7.51479976, 7.51914996, 7.52402142,
       7.53155238, 7.5352967 , 7.54538975, 7.55695057, 7.55903826,
       7.56423848, 7.57301726, 7.57660977, 7.58578882, 7.58984151,
       7.58984151, 7.59387784, 7.59940133, 7.61529834, 7.6182511 ,
       7.62413059, 7.62900389, 7.63916117, 7.63916117, 7.65681009,
       7.65728279, 7.66011432, 7.66528472, 7.66575343, 7.66622193,
       7.67925143, 7.68248245, 7.6861623 , 7.69302575, 7.70074779,
       7.70120018, 7.7088596 , 7.71289096, 7.72001794, 7.73105314,
```

7.73848812, 7.74022952, 7.7419679 , 7.74586823, 7.7702232 ,
7.77359447, 7.78155596, 7.78447324, 7.79028238, 7.79069603,
7.79317435, 7.7935868 , 7.7935868 , 7.79399909, 7.79399909,
7.79564654, 7.79646924, 7.79688034, 7.80628929, 7.81278282,
7.81439963, 7.81843027, 7.82164313, 7.82284529, 7.85049318,
7.8613418 , 7.87625888, 7.8860814 , 7.88645727, 7.89729647,
7.90248744, 7.90507285, 7.91205689, 7.91315519, 7.9157132 ,
7.91717199, 7.91753635, 7.92515751, 7.93487157, 7.93844555,
7.94058383, 7.9490915 , 7.95296679, 7.97108575, 7.97728199,
7.9776251 , 7.97899637, 7.98207488, 7.98412196, 7.98480339,
7.98888225, 8.00636757, 8.00836557, 8.02420749, 8.02910705,
8.02910705, 8.03008409, 8.03786623, 8.04430541, 8.04494705,
8.04782936, 8.04878828, 8.05261482, 8.05959233, 8.06808963,
8.06934237, 8.07246737, 8.07558264, 8.07558264, 8.07992777,
8.08332861, 8.08456242, 8.08641028, 8.09193346, 8.09346227,
8.0974263 , 8.09955428, 8.10500554, 8.10862327, 8.111328 ,
8.11192806, 8.11641707, 8.11731246, 8.11731246, 8.12237124,
8.12355784, 8.12474302, 8.12799506, 8.1285852 , 8.12946976,
8.13270649, 8.13446757, 8.13827264, 8.13856474, 8.13914868,
8.14031554, 8.14815644, 8.15651023, 8.15708379, 8.16365618,
8.16650032, 8.16735199, 8.1727291 , 8.17329344, 8.18060095,
8.18172046, 8.18255926, 8.18841131, 8.19063168, 8.19257047,
8.19284713, 8.19450551, 8.19533367, 8.19560957, 8.20028826,
8.20111164, 8.20248245, 8.20521843, 8.20631073, 8.20794694,
8.21878716, 8.22094117, 8.22362718, 8.22710823, 8.2281769 ,
8.22897764, 8.22977775, 8.23270601, 8.23429964, 8.23695005,
8.23880117, 8.23880117, 8.23906533, 8.23932943, 8.24433405,
8.24590926, 8.24826745, 8.25088114, 8.25114214, 8.25348803,
8.2550489 , 8.25738566, 8.26049286, 8.26178468, 8.26204284,
8.26744896, 8.27078101, 8.27154837, 8.272826 , 8.27461195,
8.27563105, 8.27918978, 8.28020423, 8.28324144, 8.28374675,
8.2839993 , 8.28778003, 8.28778003, 8.28979058, 8.29679587,
8.30003171, 8.30300938, 8.30325712, 8.30424747, 8.30573114,
8.30869192, 8.31066091, 8.31090676, 8.31115255, 8.3123806 ,
8.3123806 , 8.31287139, 8.31556648, 8.317522 , 8.31825433,
8.31849832, 8.31849832, 8.3202046 , 8.32117831, 8.32336569,
8.32385113, 8.32457885, 8.32554831, 8.32579053, 8.33110455,
8.33134542, 8.331827 , 8.33206771, 8.33567131, 8.33878397,
8.33973977, 8.33997857, 8.34045601, 8.34093323, 8.34188697,
8.34188697, 8.34426736, 8.34782735, 8.35678967, 8.35866628,
8.35960327, 8.36053938, 8.36053938, 8.36520683, 8.36706773,
8.36753242, 8.36846114, 8.37077917, 8.37170488, 8.37262974,
8.37401542, 8.37424618, 8.37447689, 8.37539919, 8.37562963,
8.37632063, 8.37862054, 8.38022734, 8.38091517, 8.38114435,
8.38206052, 8.38389034, 8.38457567, 8.38526052, 8.38731227,
8.38958707, 8.38981426, 8.38981426, 8.39705739, 8.40110871,
8.40223117, 8.4026798 , 8.40290405, 8.40290405, 8.40312824,
8.40648507, 8.40849377, 8.40893961, 8.41027591, 8.41360888,
8.41405243, 8.41449579, 8.41670965, 8.42068229, 8.42112272,
8.42266271, 8.42288251, 8.42376125, 8.42595471, 8.42726848,
8.42748728, 8.42858053, 8.42879904, 8.42945428, 8.43076346,
8.43098149, 8.43185314, 8.43228868, 8.43294164, 8.43446354,
8.43576619, 8.44031215, 8.44052811, 8.44139148, 8.44246965,
8.44440742, 8.4446225 , 8.4446225 , 8.44612674, 8.4469853 ,
8.44719982, 8.44805745, 8.44891435, 8.44955654, 8.45062595,
8.45105339, 8.45276133, 8.45425339, 8.45723085, 8.45829208,
8.4585042 , 8.45914026, 8.46083446, 8.46125756, 8.46484671,

8.46568935, 8.46779284, 8.46779284, 8.46821301, 8.46884293,
8.46926266, 8.47093981, 8.47240501, 8.47345027, 8.4740769 ,
8.47449444, 8.4749118 , 8.47553752, 8.47720418, 8.48198044,
8.48342956, 8.48384321, 8.48467   , 8.48487659, 8.48570252,
8.4859089 , 8.48941081, 8.48982199, 8.49023301, 8.49331025,
8.4953565 , 8.49821422, 8.49841804, 8.49943647, 8.50045387,
8.50065722, 8.50532302, 8.51057132, 8.51097389, 8.51177856,
8.51298435, 8.51438926, 8.51539157, 8.51559191, 8.51659301,
8.51899157, 8.51939077, 8.5213844 , 8.52257866, 8.5243674 ,
8.52555811, 8.52654929, 8.5267474 , 8.52892411, 8.52971447,
8.53405031, 8.53464011, 8.53699582, 8.53758388, 8.53856322,
8.54012816, 8.54012816, 8.54051902, 8.54090972, 8.54090972,
8.54305585, 8.54461379, 8.5461693 , 8.55062797, 8.55082137,
8.55275337, 8.55487426, 8.55487426, 8.55699066, 8.55699066,
8.55737498, 8.55986947, 8.56140145, 8.56712556, 8.56788631,
8.5680764 , 8.5680764 , 8.56940606, 8.5752734 , 8.57602798,
8.57659353, 8.57791193, 8.57791193, 8.57847642, 8.57922858,
8.57998018, 8.58241898, 8.58354257, 8.5841039 , 8.5872788 ,
8.58969988, 8.59100112, 8.59600437, 8.59600437, 8.59858883,
8.59877318, 8.60263667, 8.60520407, 8.6059364 , 8.6061194 ,
8.60867788, 8.61177583, 8.61250337, 8.61286694, 8.61468281,
8.61540824, 8.61830469, 8.61848544, 8.61956926, 8.62299361,
8.62317351, 8.62461159, 8.62568879, 8.62748155, 8.62801875,
8.63266244, 8.63586472, 8.63657495, 8.63675243, 8.64276801,
8.64593815, 8.64611397, 8.648046 , 8.648046 , 8.65067458,
8.65329627, 8.65347081, 8.65538869, 8.65591111, 8.6633693 ,
8.66664714, 8.66750795, 8.66974259, 8.673855 , 8.673855 ,
8.67778026, 8.67863154, 8.68152048, 8.68202943, 8.68372406,
8.68592279, 8.68676718, 8.68895923, 8.69567405, 8.69734573,
8.6983474 , 8.70483391, 8.70533113, 8.70649036, 8.70682132,
8.70682132, 8.70847449, 8.71177265, 8.71505996, 8.71915396,
8.72160234, 8.72274287, 8.72485756, 8.72745412, 8.7311749 ,
8.73182058, 8.73294952, 8.73359406, 8.73423818, 8.73681053,
8.73729211, 8.73777346, 8.73793386, 8.73809423, 8.73905592,
8.74353163, 8.74416939, 8.74687532, 8.74782861, 8.74830491,
8.74909825, 8.75036628, 8.75778366, 8.75951172, 8.76295892,
8.76389701, 8.76748487, 8.76981787, 8.77307495, 8.77353938,
8.78431535, 8.78523362, 8.78523362, 8.78645668, 8.7896601 ,
8.79026911, 8.79072563, 8.79361207, 8.79361207, 8.79452185,
8.7946734 , 8.79588497, 8.79860565, 8.80086724, 8.8040249 ,
8.80462523, 8.80777107, 8.80866806, 8.81507309, 8.82040407,
8.8206994 , 8.82158488, 8.82173238, 8.82246957, 8.8227643 ,
8.82834762, 8.82834762, 8.82849413, 8.82922635, 8.83244179,
8.83287946, 8.83317113, 8.83375422, 8.83419132, 8.83491939,
8.83521046, 8.83637393, 8.83695516, 8.83695516, 8.83782636,
8.8381166 , 8.84000107, 8.84188199, 8.84894   , 8.8493705 ,
8.85066092, 8.85352256, 8.8539511 , 8.85466493, 8.85566343,
8.85808422, 8.85950548, 8.8616336 , 8.86205868, 8.86869478,
8.86925752, 8.86967937, 8.87206651, 8.87248718, 8.87472777,
8.8784974 , 8.88419463, 8.89041055, 8.89082358, 8.89206163,
8.89329814, 8.89384722, 8.89439599, 8.89959436, 8.90027635,
8.90054901, 8.90095787, 8.90218345, 8.90259164, 8.90381521,
8.90435854, 8.90598677, 8.90720619, 8.90842414, 8.90977567,
8.91072066, 8.91139511, 8.91220384, 8.91260796, 8.91328114,
8.91422282, 8.91502927, 8.91529795, 8.91556655, 8.91717664,
8.91744473, 8.91757875, 8.91824859, 8.92199141, 8.92252496,
8.9226583 , 8.92332474, 8.92399074, 8.92611897, 8.92731411,

```
        8.92731411, 8.92784483, 8.93208044, 8.93234457, 8.9334004 ,
        8.93366418, 8.93485034, 8.93498205, 8.93669269, 8.94115288,
        8.94141463, 8.94455025, 8.94793611, 8.94819608, 8.94871583,
        8.94936514, 8.95079214, 8.95169917, 8.95338147, 8.95376929,
        8.95751051, 8.95776801, 8.96251983, 8.96661139, 8.9686509 ,
        8.97030495, 8.9709404 , 8.97563018, 8.98293776, 8.98844604,
        8.99850761, 9.00220858, 9.00663173, 9.00797936, 9.00871366,
        9.01529825, 9.0234082 , 9.03157249, 9.03551068, 9.05438807,
        9.07280096])),
(0.5432509051940598, 8.283711727194532, 0.929679930253805))
```

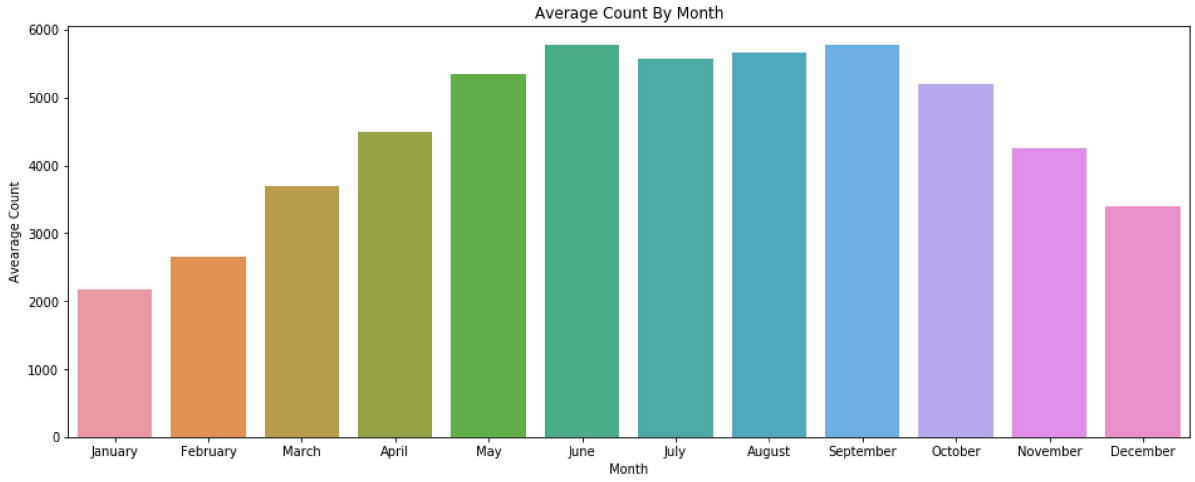


Visualizing Count Vs (Month,Season,Weekday,Usertype)

- It is quiet obvious that people tend to rent bike during summer season since it is really conducive to ride bike at that season.Therefore June, July and August has got relatively higher demand for bicycle.

In [16]:
```
fig,(ax1)= plt.subplots(nrows=1)
fig.set_size_inches(16,6)
sortOrder = ["January","February","March","April","May","June","July","August"
,"September","October","November","December"]
hueOrder = ["Sunday","Monday","Tuesday","Wednesday","Thursday","Friday","Satur
day"]

monthAggregated = pd.DataFrame(dailyData.groupby("month")["cnt"].mean()).reset
_index()
monthSorted = monthAggregated.sort_values(by="cnt",ascending=False)
sn.barplot(data=monthSorted,x="month",y="cnt",ax=ax1,order=sortOrder)
ax1.set(xlabel='Month', ylabel='Avearage Count',title="Average Count By Month"
)
```

Out[16]:
```
[Text(0, 0.5, 'Avearage Count'),
 Text(0.5, 0, 'Month'),
 Text(0.5, 1.0, 'Average Count By Month')]
```



So we have visualized the data to a greater extent.So lets go and build some models and see how close we can predict the results.

Filling 0's In windspeed Using Random Forest

Lets Read In Train And Test Data

In [17]:
```
data = pd.read_csv("day.csv")
```

Feature Engineering

In [18]:
```
data["weekday"] = data.dteday.apply(lambda dateString : datetime.strptime(date
String,"%Y-%m-%d").weekday())
data["month"] = data.dteday.apply(lambda dateString : datetime.strptime(dateSt
ring,"%Y-%m-%d").month)
data["year"] = data.dteday.apply(lambda x : x.split()[0].split("-")[0])
```

Coercing To Categorical Type

In [19]:
```
categoricalFeatureNames = ["season","holiday","workingday","weathersit","weekd
ay","month","year"]
numericalFeatureNames = ["temp","hum","windspeed","atemp"]
dropFeatures = ['instant','casual","dteday","registered",'mnth', 'yr']
```

In [20]:
```
for var in categoricalFeatureNames:
    data[var] = data[var].astype("category")
```

In [21]:
```
dteday = data.dteday
data  = data.drop(dropFeatures,axis=1)
```

In [22]:
```
data.head(2)
```

Out[22]:

	season	holiday	weekday	workingday	weathersit	temp	atemp	hum	windspeed
0	1	0	5	0	2	0.344167	0.363625	0.805833	0.160446
1	1	0	6	0	2	0.363478	0.353739	0.696087	0.248539

Splitting Train And Test Data

In [23]:
```
x_vars = list(data.columns)
x_vars.remove('cnt')
print(x_vars)
```

```
['season', 'holiday', 'weekday', 'workingday', 'weathersit', 'temp', 'atemp',
'hum', 'windspeed', 'month', 'year']
```

In [24]:
```
from sklearn.model_selection import train_test_split
X = data[x_vars]
Y = data.cnt
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.2)
print(X_train.shape, y_train.shape)
print(X_test.shape, y_test.shape)
```

```
(584, 11) (584,)
(147, 11) (147,)
```

In [25]:
```
dataTrain = X_train
yLabels = y_train
```

RMSLE Scorer

In [26]:
```
def rmsle(y, y_,convertExp=True):
    if convertExp:
        y = np.exp(y),
        y_ = np.exp(y_)
    log1 = np.nan_to_num(np.array([np.log(v + 1) for v in y]))
    log2 = np.nan_to_num(np.array([np.log(v + 1) for v in y_]))
    calc = (log1 - log2) ** 2
    return np.sqrt(np.mean(calc))
```

Linear Regression Model

In [27]:
```
from sklearn.linear_model import LinearRegression,Ridge,Lasso
from sklearn.model_selection import GridSearchCV
from sklearn import metrics
import warnings
pd.options.mode.chained_assignment = None
warnings.filterwarnings("ignore", category=DeprecationWarning)

# Initialize logistic regression model
lModel = LinearRegression()

# Train the model
yLabelsLog = np.log1p(yLabels)
lModel.fit(X = dataTrain,y = yLabelsLog)

# Make predictions
preds = lModel.predict(X= dataTrain)
print ("RMSLE Value For Linear Regression: ",rmsle(np.exp(yLabelsLog),np.exp(p
reds),False))
```

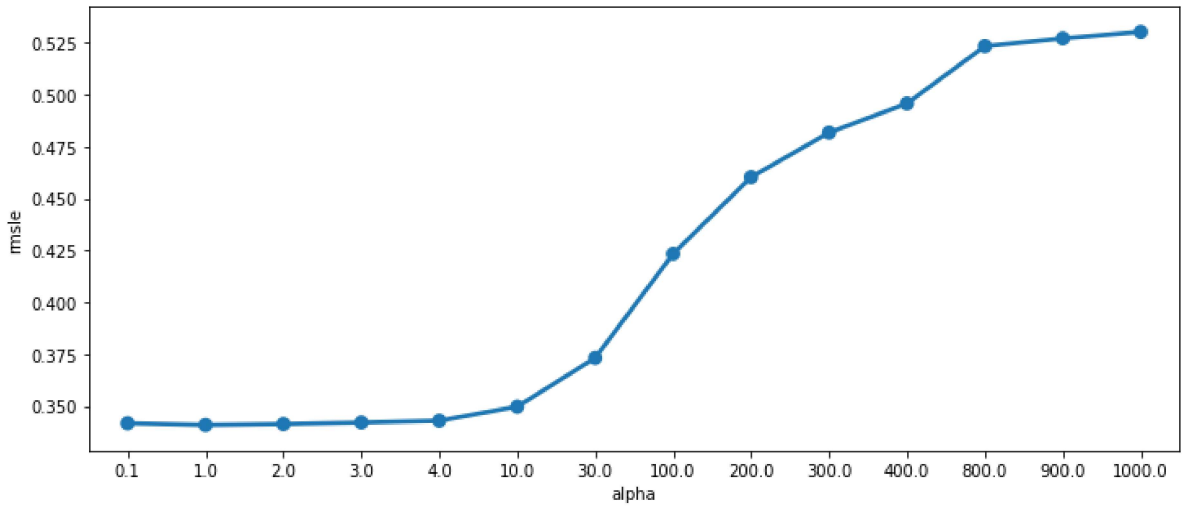
```
RMSLE Value For Linear Regression:  0.34362984831234766
```

Regularization Model - Ridge

In [28]:
```
ridge_m_ = Ridge()
ridge_params_ = { 'max_iter':[3000],'alpha':[0.1, 1, 2, 3, 4, 10, 30,100,200,3
00,400,800,900,1000]}
rmsle_scorer = metrics.make_scorer(rmsle, greater_is_better=False)
grid_ridge_m = GridSearchCV( ridge_m_,
                            ridge_params_,
                            scoring = rmsle_scorer,
                            cv=5,
                            return_train_score=True)
yLabelsLog = np.log1p(yLabels)
grid_ridge_m.fit( dataTrain, yLabelsLog )
preds = grid_ridge_m.predict(X= dataTrain)
print (grid_ridge_m.best_params_)
print ("RMSLE Value For Ridge Regression: ",rmsle(np.exp(yLabelsLog),np.exp(pr
eds),False))

fig,ax= plt.subplots()
fig.set_size_inches(12,5)
df = pd.DataFrame(grid_ridge_m.cv_results_)
df["alpha"] = df["params"].apply(lambda x:x["alpha"])
df["rmsle"] = df["mean_test_score"].apply(lambda x:-x)
sn.pointplot(data=df,x="alpha",y="rmsle",ax=ax)
```

```
{'alpha': 1, 'max_iter': 3000}
RMSLE Value For Ridge Regression:  0.34432163148375283
```

Out[28]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x534eb6a0b8>
```



In [29]:
```
df["alpha"] = df["params"].apply(lambda x:x["alpha"])
df["rmsle"] = df["mean_test_score"].apply(lambda x:-x)
sn.pointplot(data=df,x="alpha",y="rmsle",ax=ax)
```

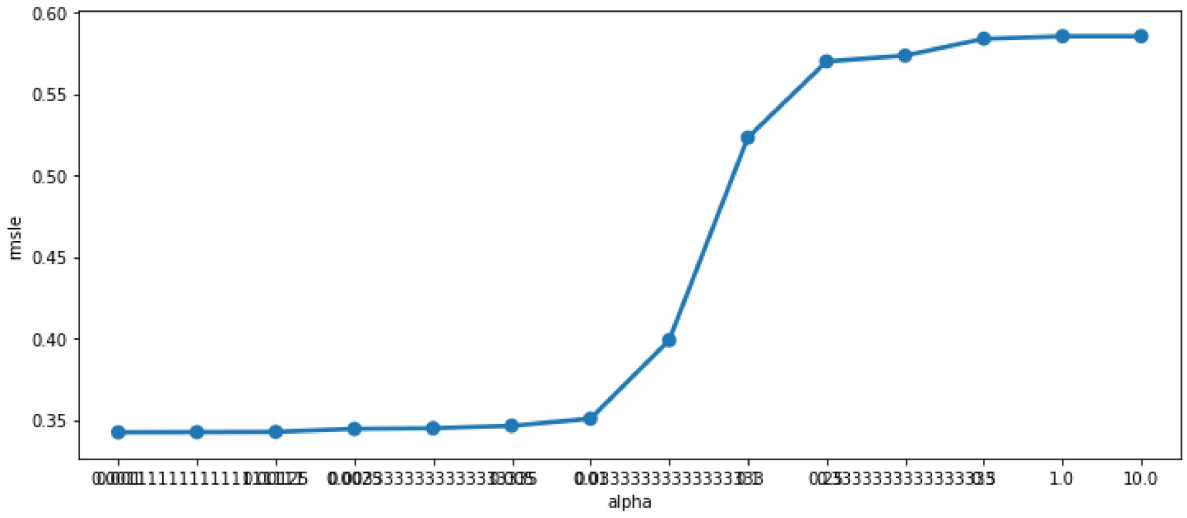
Out[29]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x534eb6a0b8>
```

Regularization Model - Lasso

In [30]:
```
lasso_m_ = Lasso()

alpha  = 1/np.array([0.1, 1, 2, 3, 4, 10, 30,100,200,300,400,800,900,1000])
lasso_params_ = { 'max_iter':[3000],'alpha':alpha}

grid_lasso_m = GridSearchCV( lasso_m_,lasso_params_,scoring = rmsle_scorer,cv=
5, return_train_score=True)
yLabelsLog = np.log1p(yLabels)
grid_lasso_m.fit( dataTrain, yLabelsLog )
preds = grid_lasso_m.predict(X= dataTrain)
print (grid_lasso_m.best_params_)
print ("RMSLE Value For Lasso Regression: ",rmsle(np.exp(yLabelsLog),np.exp(pr
eds),False))

fig,ax= plt.subplots()
fig.set_size_inches(12,5)
df = pd.DataFrame(grid_lasso_m.cv_results_)
df["alpha"] = df["params"].apply(lambda x:x["alpha"])
df["rmsle"] = df["mean_test_score"].apply(lambda x:-x)
sn.pointplot(data=df,x="alpha",y="rmsle",ax=ax)
```

```
{'alpha': 0.001, 'max_iter': 3000}
RMSLE Value For Lasso Regression:  0.3442459713628952
```

Out[30]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x534f19ce48>
```



Ensemble Models - Random Forest

In [31]:
```
from sklearn.ensemble import RandomForestRegressor
rfModel = RandomForestRegressor(n_estimators=500)
yLabelsLog = np.log1p(yLabels)
rfModel.fit(dataTrain,yLabelsLog)
preds = rfModel.predict(X= dataTrain)
print ("RMSLE Value For Random Forest: ",rmsle(np.exp(yLabelsLog),np.exp(preds
),False))
```

```
RMSLE Value For Random Forest:  0.10899995853128484
```

Ensemble Model - Gradient Boost

In [32]:
```
from sklearn.ensemble import GradientBoostingRegressor
gbm = GradientBoostingRegressor(n_estimators=500,alpha=0.01); ### Test 0.41
yLabelsLog = np.log1p(yLabels)
gbm.fit(dataTrain,yLabelsLog)
preds = gbm.predict(X= dataTrain)
print ("RMSLE Value For Gradient Boost: ",rmsle(np.exp(yLabelsLog),np.exp(pred
s),False))
```

```
RMSLE Value For Gradient Boost:  0.036058642339735825
```

Lets compare the distribution of train and test results. More or less the distribution of train and test looks identical. It confirms visually that our model has not predicted really bad and not suffering from major overfitting problem.

In [33]:
```
predsTest = gbm.predict(X= X_test)
fig,(ax1,ax2)= plt.subplots(ncols=2)
fig.set_size_inches(12,5)
sn.distplot(yLabels,ax=ax1,bins=50)
sn.distplot(np.exp(predsTest),ax=ax2,bins=50)
```

Out[33]: <matplotlib.axes._subplots.AxesSubplot at 0x535120e9e8>