Bike Renting

Divyang Gor

January 14, 2020

Contents

Chapter 1

Introduction

1.1 Problem Statement

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

1.2 Data

instant	dteday	season	yr	mnth	holiday	weekday	workingday	weathersit
1	1/1/11	1	0	1	0	6	0	2
2	1/2/11	1	0	1	0	0	0	2
3	1/3/11	1	0	1	0	1	1	1
4	1/4/11	1	0	1	0	2	1	1
5	1/5/11	1	0	1	0	3	1	1

Table 1.1: Daily Data for Bike Renting Column 1 to 9

temp	atemp	hum	windspeed	casual	registered	cnt
0.344167	0.363625	0.805833	0.160446	331	654	985
0.363478	0.353739	0.696087	0.248539	131	670	801
0.196364	0.189405	0.437273	0.248309	120	1229	1349
0.2	0.212122	0.590435	0.160296	108	1454	1562
0.226957	0.22927	0.436957	0.1869	82	1518	1600

Table 1.2: Daily Data for Bike Renting Column 10 to 16

From Tables 1.1 and 1.2 below is the list of predictor Variables with their meaning:

- dteday: Date
- season: Season (1:springer, 2:summer, 3:fall, 4:winter)
- ar (0: 2011, 1:2012)
- Month (1 to 12)
- holiday: weather day is holiday or not (extracted fromHoliday Schedule)
- weekday: Day of the week
- workingday: If day is neither weekend nor holiday is 1, otherwise is 0.
- weathersit: (extracted fromFreemeteo)

  1: Clear, Few clouds, Partly cloudy, Partly cloudy

  2: Mist + Cloudy, Mist + Broken clouds, Mist + Few clouds, Mist

  3: Light Snow, Light Rain + Thunderstorm + Scattered clouds, Light Rain + Scattered clouds

  4: Heavy Rain + Ice Pallets + Thunderstorm + Mist, Snow + Fog
- temp: Normalized temperature in Celsius. The values are derived via $(t - t_m in)/(t_m ax - t_m in)$, $t_m in = -8$, $t_m ax = +39$ (only in hourly scale)
- atemp: Normalized feeling temperature in Celsius. The values are derived via $(t - t_m in)/(t_m ax - t_m in)$, $t_m in = -16$, $t_m ax = +50$ (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

Chapter 2

Methodology

2.1 Pre Processing

Data Pre Processing is also called Exploratory Data Analysis(EDA) which includes data visualization and transformation to data in a systematic way. Below mentioned process can be included in data pre processing:

1. Missing Value Analysis
2. Outline Analysis
3. Data Visualization
4. Standardization and Normalization
5. Feature Selection and Scaling

2.1.1 Missing Value Analysis

In the given data set there is no missing value so this step can be skipped. In case, we have missing value in our data set than we will replace those missing values by Mean, Median, Mode or KNN Imputation which ever is suitable.

2.1.2 Outlier Analysis

In this step we will check for presence of outlines. To get outlines we use a classic approach of removing outliers, Tukey's method. We visualize the outliers using Boxplots. As you can see form the figure 2.1 is the box plot for the variables Boxplot for Temperature, Feeling Temperature, Humidity, Windspeed and figure 2.2. From these diagrams we can conclude that the variables 'humidity' and 'windspeed' has outliers.

In next step all outliers will be replaced by NA's. After converting outliers to NA's on performing missing value analysis it can be observed that 'windspeed' and 'humidity' has missing value percentage 1.778386 and 0.273598 respectively, this is very small amount so we can dropped this NA's. After this step out data is free from outliers.

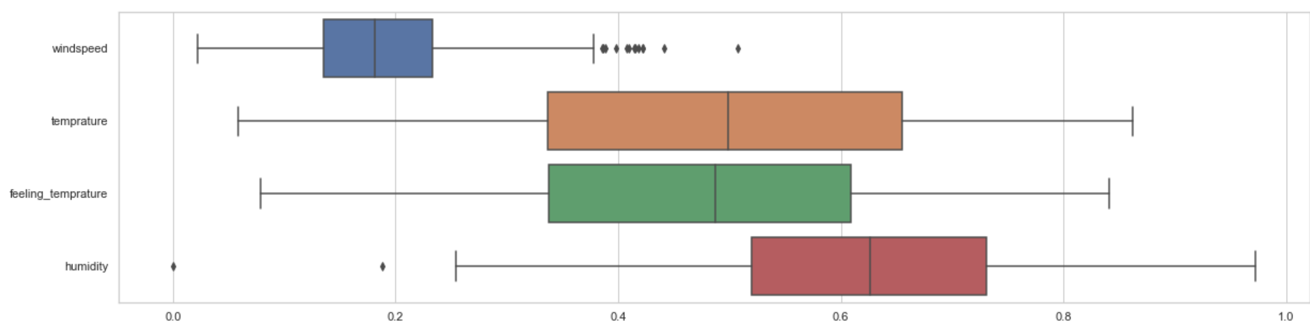


Figure 2.1: Boxplot for Temperature, Feeling Temperature, Humidity, Windspeed

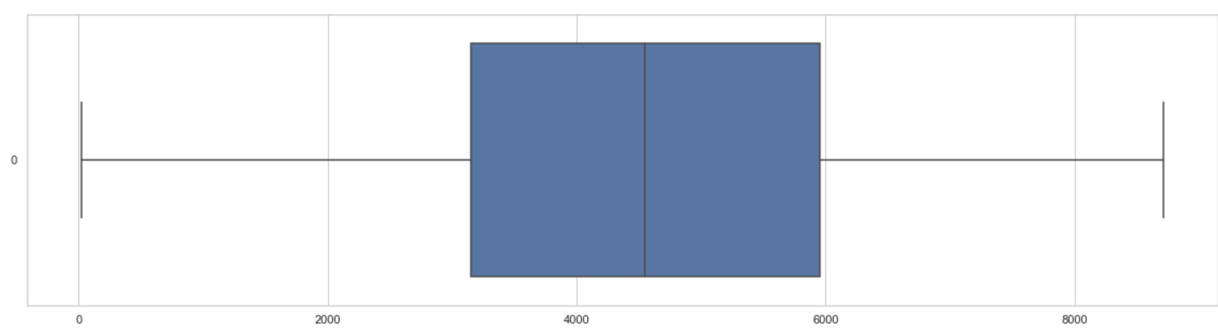


Figure 2.2: Boxplot for Total Count

2.2 Data Visualization

Any predictive modeling requires that we look at the data before we start modeling. The data visualization includes cleaning the data as well as visualizing the data through graphs and plots. To start this process we will first try and look at all the probability distributions of the variables. Most analysis like regression, require the data to be normally distributed. We can visualize that in a glance by looking at the probability distributions or probability density functions of the variable.

In Figure 2.3 it is the probability density function for variables Humidity, Windspeed, Temperature and Feeling Temperature. Figure 2.4 is the plot of Probability Density function for Total Count Variable.The blue lines indicate Kernel Density Estimations (KDE) of the variables. The Black lines represent the normal distribution. So as you can see in the figure most variables either very closely, or somewhat imitate the normal distribution.

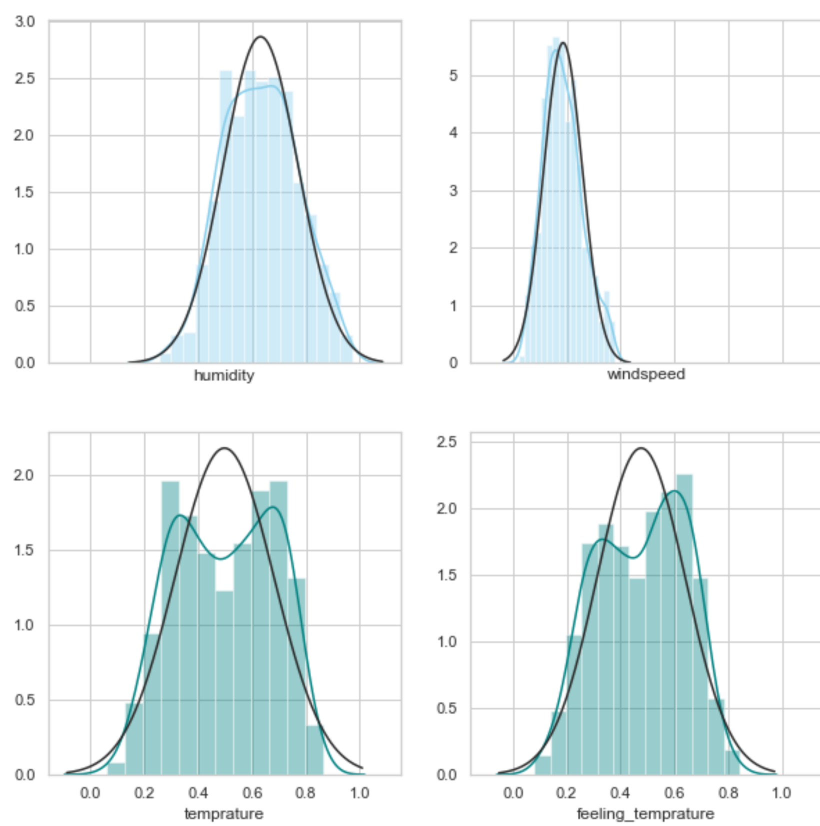


Figure 2.3: Probability Density Function for Humidity, Windspeed, Temperature and Feeling Temperature

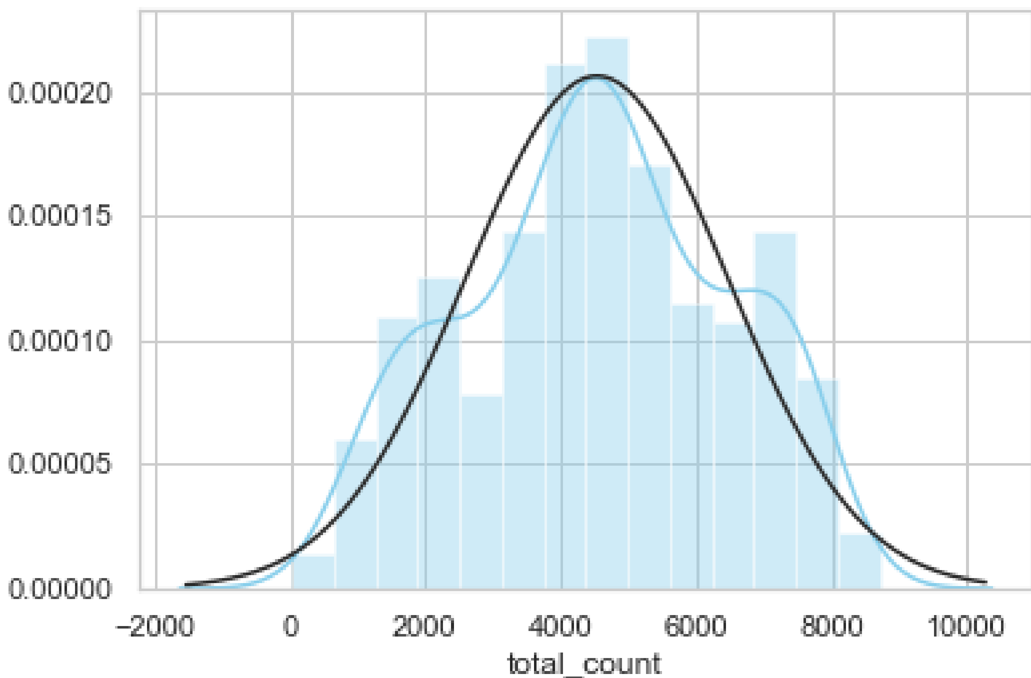


Figure 2.4: Probability Density Function for Total Count

Furthermore, in Figure 2.5 it shows the Bar Plot for Mean Total Count Vs. Holiday, Weekday, Workingday, Season, Month and Year. Form the Bar-plots we can conclude following things:

- Total average count is higher when there is no holiday.
- On all days average total count is almost similar.
- There is no change in total count whether it is holiday or not.
- In springer average total count is minimum and in fall average total count is maximum so we can conclude maximum number of bike users are there in fall the same thing is also visible month wise plot.
- Number of bike users increased in year 2012.

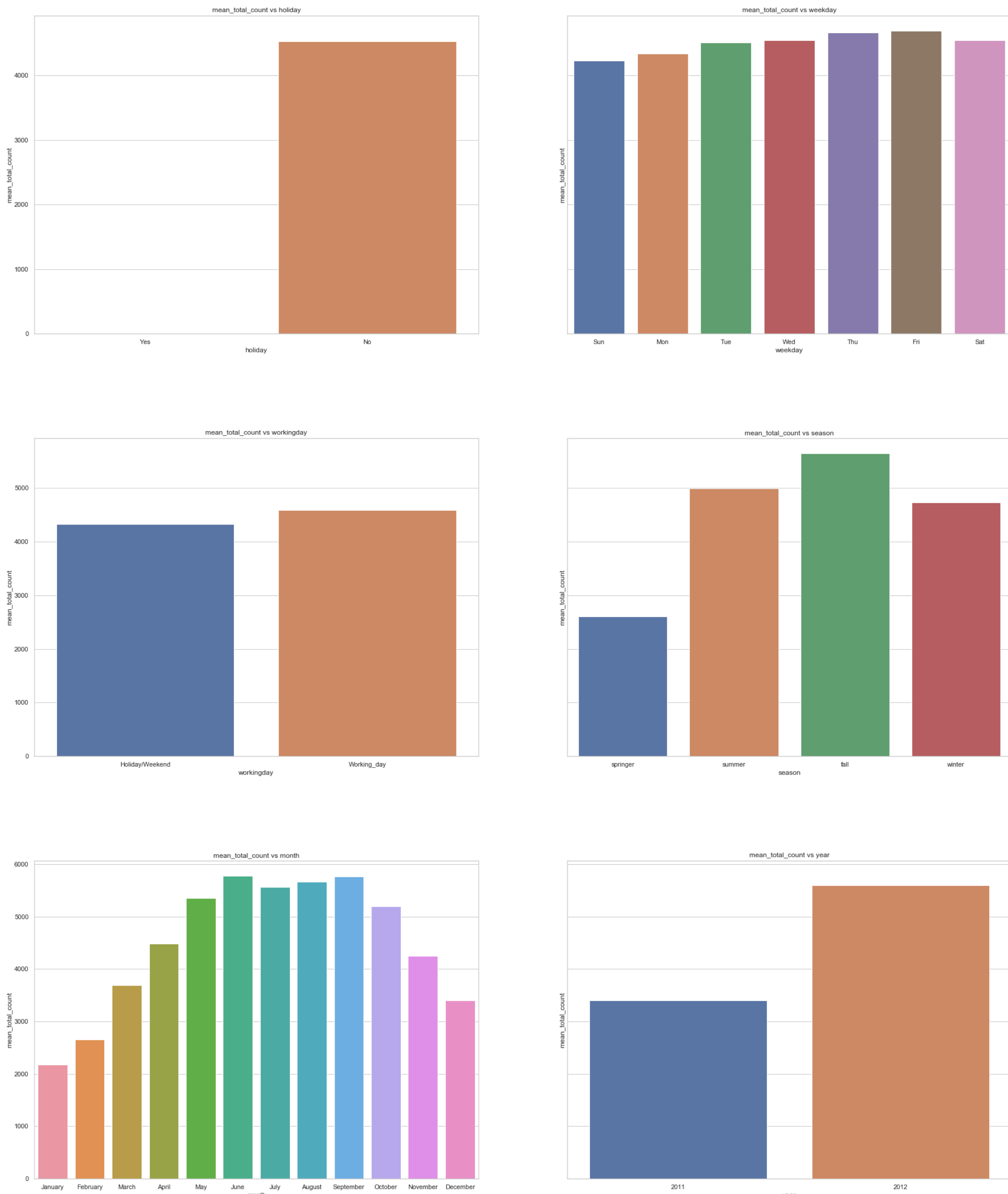


Figure 2.5: Mean Total Cont Vs. Holiday, Weekday, Workingday, Season, Month, Year

2.3 Feature Selection

Before applying any algorithm we need to assess the importance of each predictor variable in our analysis. There is a possibility that many variables in our analysis are not important at all to the problem of class prediction. There are several methods of doing that. Here for continuous variables we will apply method of Correlation and for categorical variables we will apply ANOVA test.

Coefficient of Correlation

Coefficient of correlation is used to derive importance of feature while predicting value for dependent variable. Below Table 2.1 for coefficient of correlation.

	temprature	feeling_temprature	humidity	windspeed	total_count	casual_count	registered_count
temprature	1	0.991738	0.114191	-0.140169	0.625892	0.539714	0.538095
feeling_temprature	0.991738	1	0.126587	-0.166038	0.629204	0.540234	0.541977
humidity	0.114191	0.126587	1	-0.204496	-0.136621	-0.101439	-0.124701
windspeed	-0.140169	-0.166038	-0.204496	1	-0.216193	-0.146178	-0.203677
total_count	0.625892	0.629204	-0.136621	-0.216193	1	0.670547	0.944581
casual_count	0.539714	0.540234	-0.101439	-0.146178	0.670547	1	0.389848
registered_count	0.538095	0.541977	-0.124701	-0.203677	0.944581	0.389848	1

Table 2.1: Coefficient of correlation

Figure 2.6 show the heat map for the Coefficient of Correlation.

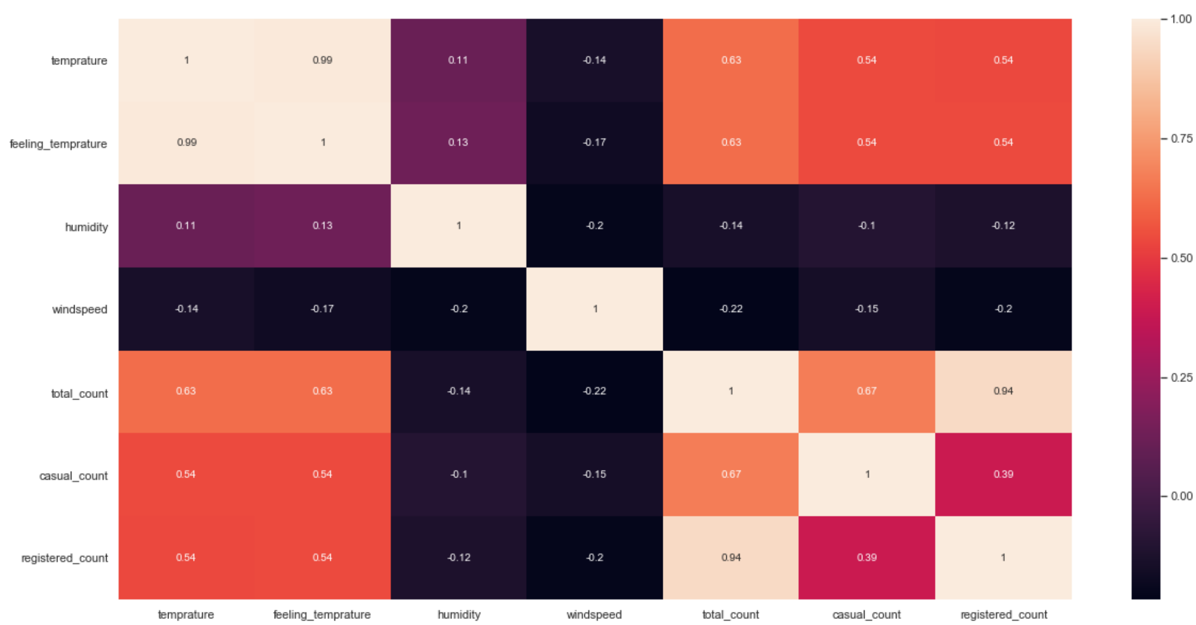


Figure 2.6: Heat Map: Coefficient of Correlation.

ANOVA Test

Table 2.2 is the summary after applying ANOVA test where if PR ¿ 0.05 is the variable that we can drop having least important feature. Here, workingday variable is there with PR value greater than 0.05. So, we will drop this variable.

	df	sum_sq	mean_sq	F	PR(>F)
season	3	9.22E+08	3.07E+08	427.956121	3.75E-157
year	1	8.72E+08	8.72E+08	1214.108425	2.11E-154
month	11	1.84E+08	1.67E+07	23.307849	6.62E-41
holiday	1	3.61E+06	3.61E+06	5.031825	2.52E-02
weekday	6	1.46E+07	2.43E+06	3.383547	2.69E-03
workingday	1	5.55E+04	5.55E+04	0.07736	7.81E-01
weather_condition	2	1.84E+08	9.20E+07	128.142583	4.53E-48
Residual	692	4.97E+08	7.18E+05	NaN	NaN

Table 2.2: ANOVA Summary

In this section we will remove unnecessary data. The variables date and instant which are irrelevant with prediction we are required. So, we will remove this variables.

Chapter 3

Modeling

3.1 Train & Test Data

After completing all data pre processing steps we will move forward for modeling for out data. In this step we will split our data in to two parts. The first part is 80 % of data have been taken as train data and remaining 20% data as test data. Train data will be used to train our model and test data will be used to test our train model.

3.2 Model Selection

Model selection is the process in which we will choose best suitable model from several machine learning approaches or choosing between different hyperparameters or sets of features for the same machine learning approach.For all different problems or data we must apply different models. It is not like that all the time random forest will give you the best fit. Hence, it is mandatory to check for all applicable models on our data depending upon dependent variable. **In this project dependent variable is continuous so we will go for Regression Analysis.** Below are certain qualities you look for in an model:

- Interpretable - can we see or understand why the model is making the decisions it makes?
- Simple - easy to explain and understand
- Accurate
- Fast (to train and test)
- Scalable (it can be applied to a large dataset)

In this project I have applied below mentioned models.

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

Linear Regression

Regression is a technique that displays the relationship between two variables. Linear Regression is the most basic machine learning algorithm. It is a type of supervised learning algorithm, commonly used for predictive analysis. Figure 3.1 is the plot for regression coefficient plot. Table: 3.1 shows regression coefficients:

Parameter	Coefficient
season	530.0433603
year	2059.970108
month	-44.3727868
holiday	-555.4837376
weekday	60.64546803
weather_condition	-545.2975275
temprature	2223.587433
feeling_temprature	3346.575801
humidity	-1134.699551
windspeed	-2348.265555

Table 3.1: Linear Regression Coefficients

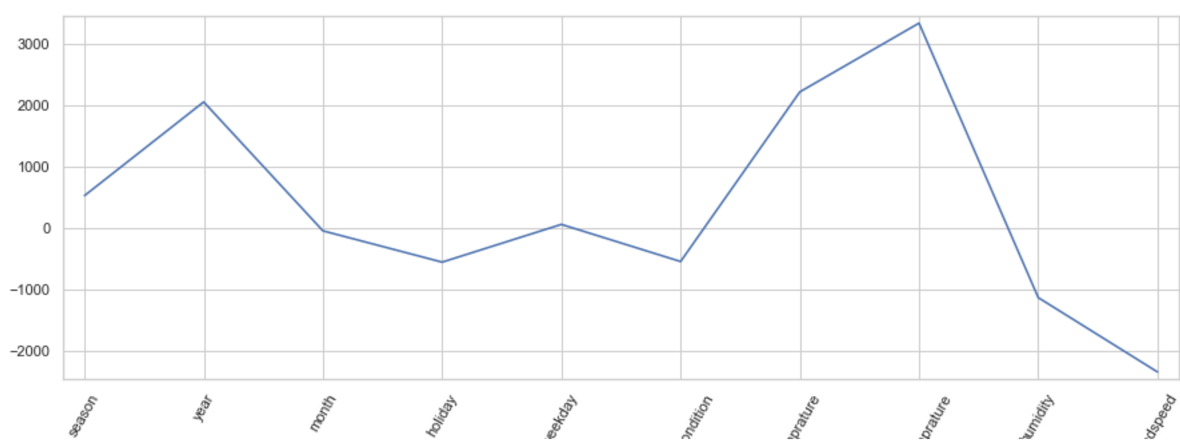


Figure 3.1: Linear Regression coefficient plot

Lasso Regression

Lasso Regression is a type of supervised learning algorithm, commonly used for predictive analysis. Figure 3.2 is the plot for regression coefficient plot. Table: 3.2 shows regression coefficients:

Parameter	Coefficient
season	531.4862176
year	2060.497668
month	-44.08714524
holiday	-526.7434977
weekday	60.79339808
weather_condition	-567.2359241
temprature	2351.626574
feeling_temprature	3156.815934
humidity	-1001.996138
windspeed	-2115.815965

Table 3.2: Lasso Regression Coefficients

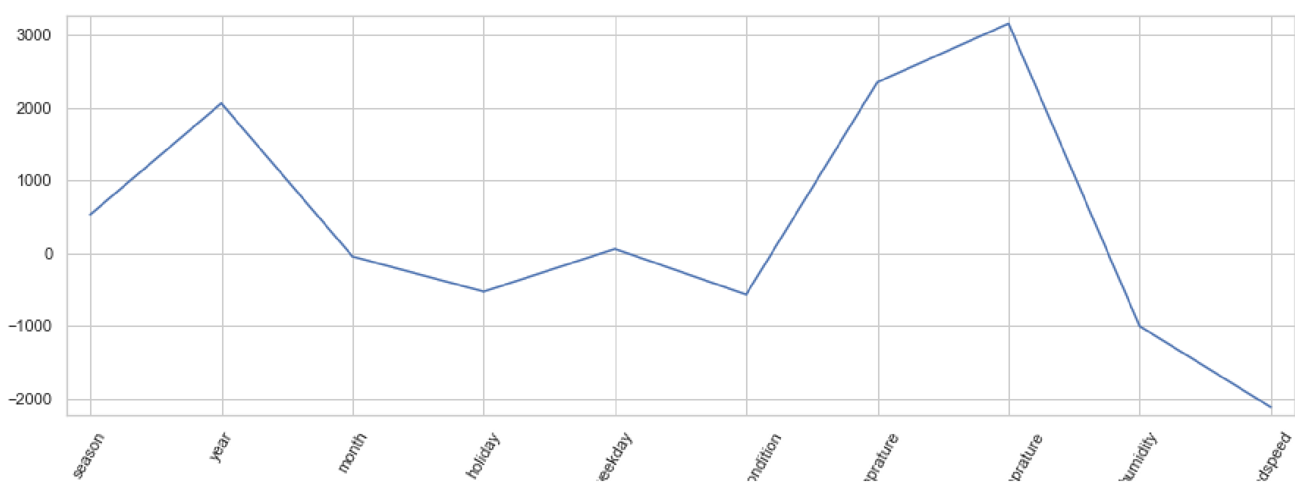


Figure 3.2: Lasso Regression coefficient

Ridge Regression

Ridge Regression is a type of supervised learning algorithm, commonly used for predictive analysis. Figure 3.3 is the plot for regression coefficient plot. Table: 3.3 shows regression coefficients:

Parameter	Coefficient
season	545.5662276
year	2057.528253
month	-45.37369828
holiday	-533.8556
weekday	60.8754719
weather_condition	-604.574905
temprature	2685.405431
feeling_temprature	2623.182193
humidity	-835.8068969
windspeed	-1706.287254

Table 3.3: Ridge Regression Coefficients

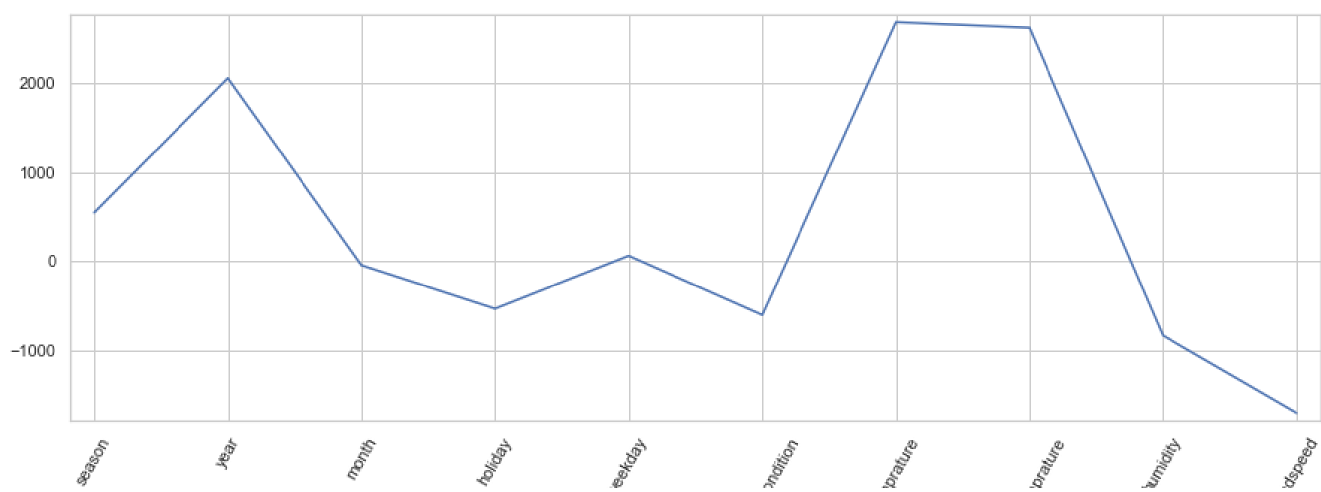


Figure 3.3: Ridge Regression coefficient

Decision Tree Regression

Decision trees are supervised learning algorithms used for both, classification and regression tasks where we will concentrate on classification. Importance of variable in Decision Tree Regression is shown in figure 3.4. The importance of variables is shown in Table: 3.4.

Parameter	Coefficient
season	0.07412835
year	0.29715768
month	0.0230701
holiday	0.00128854
weekday	0.01531377
weather_condition	0.00898129
temprature	0.42817299
feeling_temprature	0.04460997
humidity	0.07540038
windspeed	0.03187694

Table 3.4: Decision Tree Importance of Variable

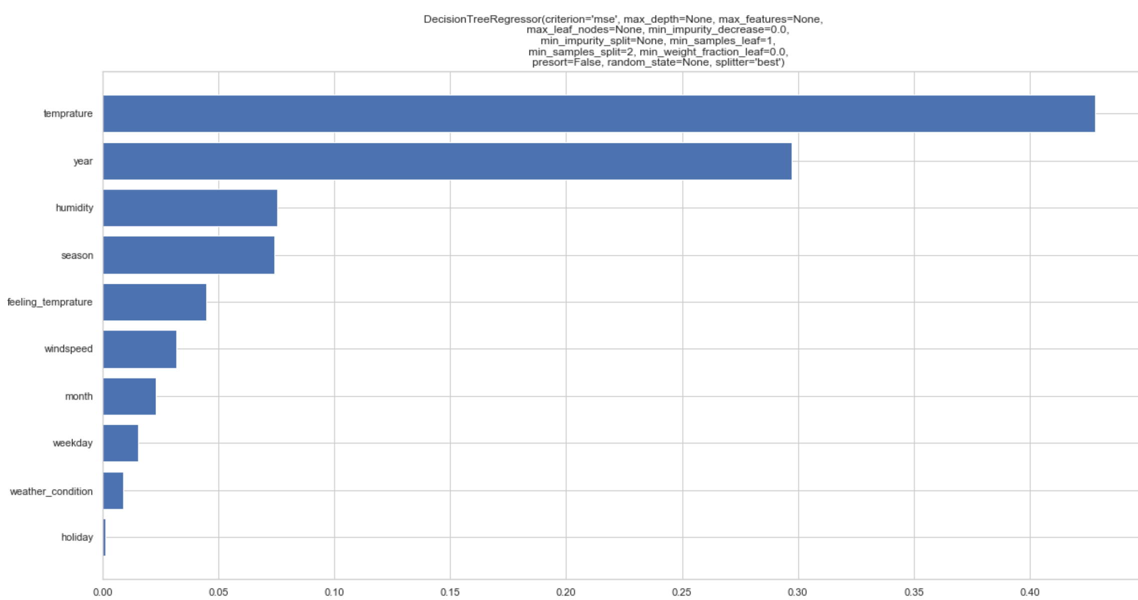


Figure 3.4: Decision Tree Regression Importance of Variable

Random Forest Regression

Random Forest is a learning method that operates by constructing multiple decision trees. The final decision is made based on the majority of the trees and is chosen by the random forest.

There are a lot of benefits to using Random Forest, but one of the main advantages is that it reduces the risk of overfitting and the required training time. Additionally, it offers a high level of accuracy. Random Forest runs efficiently in large databases and produces highly accurate predictions by estimating missing data. Importance of variable in Random Forest Regression is shown in Figure 3.5 & Table 3.5

Parameter	**Coefficient**
season	0.06463119
year	0.28013819
month	0.02478143
holiday	0.00406653
weekday	0.01596164
weather_condition	0.01251542
temprature	0.38464264
feeling_temprature	0.12339215
humidity	0.05891583
windspeed	0.03095498

Table 3.5: Random Forest Regression Importance of variable

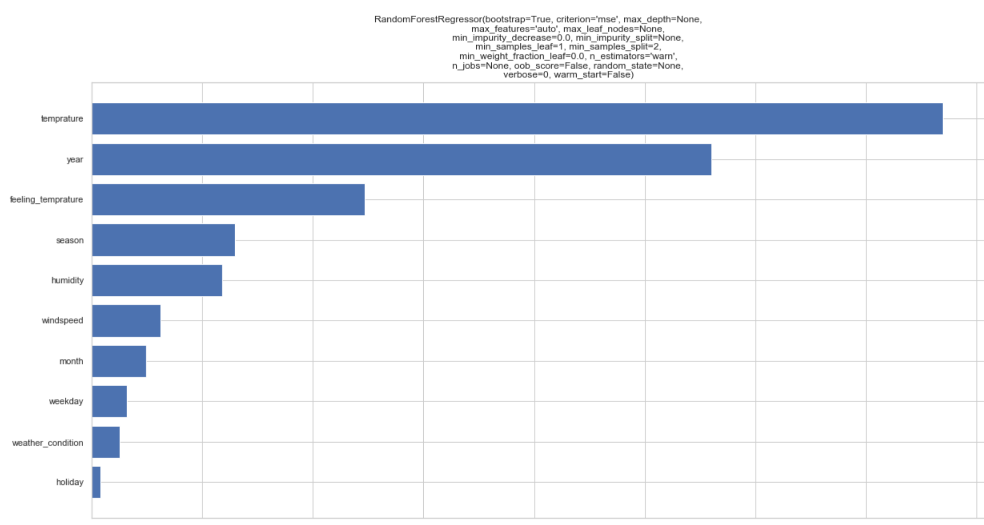


Figure 3.5: Random Forest Regression Importance of variable

XGB Regression

The term ‘Boosting’ refers to a family of algorithms which converts weak learner to strong learners. Boosting is an ensemble method for improving the model predictions of any given learning algorithm. The idea of boosting is to train weak learners sequentially, each trying to correct its predecessor. XG-Boost provides:

- Parallelization of tree construction using all of your CPU cores during training.
- Distributed Computing for training very large models using a cluster of machines.
- Out-of-Core Computing for very large datasets that don’t fit into memory.
- Cache Optimization of data structures and algorithm to make the best use of hardware.

XGBoost (Extreme Gradient Boosting) is an optimized distributed gradient boosting library. It uses gradient boosting (GBM) framework at core. XGBoost is one of the most popular and efficient implementations of the Gradient Boosted Trees algorithm, a supervised learning method that is based on function approximation by optimizing specific loss functions as well as applying several regularization techniques. Importance of variable in XGB Regression is shown in figure 3.6 & Table 3.6.

Parameter	**Coefficient**
season	0.17181417
year	0.4615058
month	0.01699146
holiday	0.01979753
weekday	0.00865801
weather_condition	0.04914481
temprature	0.19019859
feeling_temprature	0.0485683
humidity	0.01941875
windspeed	0.01390261

Table 3.6: XGBoost Regression Importance of variable

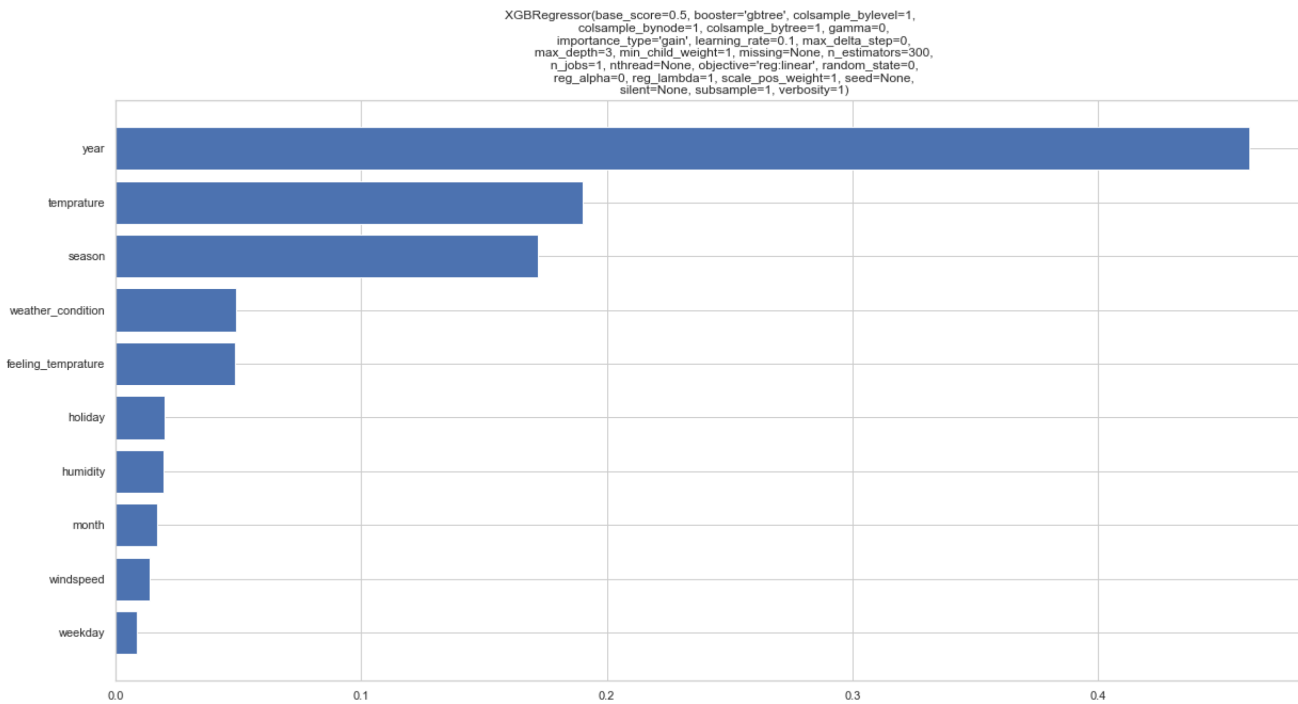


Figure 3.6: XGB Regression Importance of variable

Chapter 4

Conclusion

4.1 Model Evaluation

In order to select amongst models, we need some way of evaluating their performance.

You can't evaluate a model's hypothesis function with the cost function because minimizing the error can lead to overfitting. A good approach is to take your data and split it randomly into a training set and a test set that we already done in 3.1.

For evaluating any regression model below are the main techniques I have used:

- Root Mean Squared Error
- Root Mean Squared Logarithmic Error
- Coefficient of Determination (R^2)

4.1.1 Root Mean Squared Error(RMSE)

RMSE is one of the methods to determine the accuracy of the model on predicting values. RMSE can be calculated form below mentioned mathematical formulae:

$$RMSE = \sqrt{\frac{\sum_{i=1}^{N}(Predict_i - Actual_i)^2}{N}}$$

4.1.2 Root Mean Squared Logarithmic Error (RMSLE)

RMSLE can be calculated from below mentioned mathematical formulae:

$$RMSLE = \sqrt{\frac{1}{n}\sum_{i=1}^{N}(\log{(predicted_i + 1)^2} - \log{(actual_i + 1)^2})}$$

4.1.3 Coefficient of Determination (R^2)

Coefficient of determination R^2 (or r^2), a measure that assesses the ability of a model to predict or explain an outcome in the linear regression setting. More specifically, R^2 indicates the proportion of the variance in the dependent variable (Y) that is predicted or explained by linear regression and the predictor variable (X, also known as the independent variable).

The table 4.1 shows the result of RMSE, RMSLE and R^2 for different models I have used for model selection. After applying several models on our train data this is the time to select best model. The

Model	**RMSE_Test**	**RMSE_Train**	**RMSLE_Test**	**RMSLE_Train**	R^2 **score**
LinearRegression	833.4169	0.49307	877.39306	0.29205	0.81558
DecisionTreeRegressor	942.18824	0.53721	0	0	0.7643
Ridge	836.69768	0.49334	879.35492	0.26856	0.81413
Lasso	834.53671	0.49312	877.6398	0.27606	0.81509
RandomForestRegressor	699.26006	0.47783	247.17488	0.10899	0.87018
XGBRegressor	625.86782	0.44641	432.96809	0.13594	0.896

Table 4.1: Model Evaluation RMSE, RMSLE and R^2

performance can be measured by comparing Predictions of the models with real values and calculating some average error measure which is described in Table 4.1.

4.2 Conclusion

From the table we can conclude that **XGBoost Regression** model gives best result.

Appendix A

Python Code

A.1 Boxplot for Temperature, Feeling Temperature, Humidity, Windspeed Figure 2.1

```
sns.set(style="whitegrid")
%matplotlib inline
plt.figure(figsize = (20,5))
box_plot = sns.boxplot(data=bike_rent[{'temprature','feeling_temprature',
'humidity','windspeed'}],orient='h')
box_plot.figure.savefig("box_plot.png")
```

A.2 Boxplot for Total Count Figure 2.2

```
sns.set(style="whitegrid")
%matplotlib inline
plt.figure(figsize = (20,5))
box_plot_total = sns.boxplot(data=bike_rent['total_count'],orient='h')
box_plot_total.figure.savefig("box_plot_total.png")
```

A.3 Probability Density Function for Humidity, Windspeed, Temperature and Feeling Temperature Figure 2.3

```
f, axes = plt.subplots(2,2, figsize=(10, 10), sharex=True)
sns.distplot( bike_rent["humidity"],fit=norm , color="skyblue", ax=axes[0,0])
sns.distplot( bike_rent["windspeed"],fit=norm , color="skyblue", ax=axes[0,1])
sns.distplot( bike_rent["temprature"],fit=norm , color="teal", ax=axes[1, 0])
sns.distplot( bike_rent["feeling_temprature"],fit=norm , color="teal", ax=axes[1, 1])

Probability Density Function for Total Count Figure
norm_1 = sns.distplot( bike_rent["total_count"],fit=norm , color="skyblue")
```

A.4 Mean Total Cont Vs. Holiday, Weekday, Workingday, Season, Month, Year Figure: 2.4

```
def groupandplot(data,groupby_key,value,sortorder,
        axes,aggregate='mean'):
    agg_data=data.groupby([groupby_key])
    [value].agg(aggregate).reset_index().rename(columns={value:aggregate+'_'+value})
    count_data=data.groupby([groupby_key])['total_count'].count().reset_index()
    .rename(columns={'total_count':'Num_bike_rent'})
    plot = sns.barplot(x=groupby_key,y=aggregate+'_'+value,data=agg_data,order=sortorder,
    ax = axes).set_title(aggregate+'_'+value+" vs "+groupby_key)
f, axes = plt.subplots(1,2,sharex='col', sharey='row', figsize=(30, 10))
groupandplot(plot_bike_rent,'holiday','total_count',['Yes','No'],axes[0])
groupandplot(plot_bike_rent,'weekday','total_count',['Sun','Mon', 'Tue','Wed','Thu',
'Fri','Sat'],axes[1])
f, axes = plt.subplots(1,2,sharex='col', sharey='row', figsize=(30, 10))
groupandplot(plot_bike_rent,'workingday','total_count',['Holiday/Weekend','Working_day'],axes[0
groupandplot(plot_bike_rent,'season','total_count',['springer', 'summer', 'fall',
'winter'],axes[1])
f, axes = plt.subplots(1,2,sharex='col', sharey='row', figsize=(30, 10))
groupandplot(plot_bike_rent,'month','total_count',["January","February","March","April","May",
"June","July","August","September","October","November","December"],axes[0])
groupandplot(plot_bike_rent,'year','total_count',['2011','2012'],axes[1])
```

A.5 Coefficient of correlation Table: 2.1

```
corr = bike_rent[['temprature','feeling_temprature','humidity','windspeed','total_count',
'casual_count','registered_count']].corr()
```

A.6 Heat Map: Coefficient of Correlation 2.6

```
plt.figure(figsize=(20,10))
cor_plot = sns.heatmap(corr, annot=True)
```

Appendix B

Complete Python Code

bike_renting_dug

January 14, 2020

[1]:
```python
from pyforest import *
from statsmodels.formula.api import ols
import statsmodels.api as sm
from fancyimpute import KNN
import scipy.stats as stats
import seaborn as sn
from sklearn.linear_model import LinearRegression,Ridge,Lasso
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import RandomizedSearchCV
from sklearn.model_selection import cross_val_score
from sklearn.metrics import mean_squared_error
from sklearn import metrics
from sklearn.ensemble import RandomForestRegressor
from sklearn.tree import DecisionTreeRegressor
from xgboost import XGBRegressor
import xgboost as xgb
from sklearn.externals import joblib
from sklearn.model_selection import StratifiedKFold
from sklearn.model_selection import cross_val_score
from sklearn.metrics import mean_squared_log_error
import matplotlib.pyplot as plt
from scipy.stats import norm
import matplotlib
from prettytable import PrettyTable
```

```
Using TensorFlow backend.
/Users/divyanggor/anaconda3/lib/python3.7/site-
packages/sklearn/externals/joblib/__init__.py:15: DeprecationWarning:
sklearn.externals.joblib is deprecated in 0.21 and will be removed in 0.23.
Please import this functionality directly from joblib, which can be installed
with: pip install joblib. If this warning is raised when loading pickled models,
you may need to re-serialize those models with scikit-learn 0.21+.
  warnings.warn(msg, category=DeprecationWarning)
```

[2]:
```python
#read data
```

```
bike_rent = pd.read_csv("https://s3-ap-southeast-1.amazonaws.com/
 ↪edwisor-india-bucket/projects/data/DataN0103/day.csv")
```

[3]:
```
#view to 5 rows
bike_rent.head()
```

[3]:
```
   instant      dteday  season  yr  mnth  holiday  weekday  workingday  \
0        1  2011-01-01       1   0     1        0        6           0
1        2  2011-01-02       1   0     1        0        0           0
2        3  2011-01-03       1   0     1        0        1           1
3        4  2011-01-04       1   0     1        0        2           1
4        5  2011-01-05       1   0     1        0        3           1

   weathersit      temp     atemp       hum  windspeed  casual  registered  \
0           2  0.344167  0.363625  0.805833   0.160446     331         654
1           2  0.363478  0.353739  0.696087   0.248539     131         670
2           1  0.196364  0.189405  0.437273   0.248309     120        1229
3           1  0.200000  0.212122  0.590435   0.160296     108        1454
4           1  0.226957  0.229270  0.436957   0.186900      82        1518

    cnt
0   985
1   801
2  1349
3  1562
4  1600
```

[4]:
```
bike_rent.describe()
```

[4]:
```
          instant      season          yr        mnth     holiday     weekday  \
count  731.000000  731.000000  731.000000  731.000000  731.000000  731.000000
mean   366.000000    2.496580    0.500684    6.519836    0.028728    2.997264
std    211.165812    1.110807    0.500342    3.451913    0.167155    2.004787
min      1.000000    1.000000    0.000000    1.000000    0.000000    0.000000
25%    183.500000    2.000000    0.000000    4.000000    0.000000    1.000000
50%    366.000000    3.000000    1.000000    7.000000    0.000000    3.000000
75%    548.500000    3.000000    1.000000   10.000000    0.000000    5.000000
max    731.000000    4.000000    1.000000   12.000000    1.000000    6.000000

       workingday  weathersit        temp       atemp         hum   windspeed  \
count  731.000000  731.000000  731.000000  731.000000  731.000000  731.000000
mean     0.683995    1.395349    0.495385    0.474354    0.627894    0.190486
std      0.465233    0.544894    0.183051    0.162961    0.142429    0.077498
min      0.000000    1.000000    0.059130    0.079070    0.000000    0.022392
25%      0.000000    1.000000    0.337083    0.337842    0.520000    0.134950
50%      1.000000    1.000000    0.498333    0.486733    0.626667    0.180975
75%      1.000000    2.000000    0.655417    0.608602    0.730209    0.233214
max      1.000000    3.000000    0.861667    0.840896    0.972500    0.507463
```

```
            casual    registered          cnt
count   731.000000    731.000000   731.000000
mean    848.176471   3656.172367  4504.348837
std     686.622488   1560.256377  1937.211452
min       2.000000     20.000000    22.000000
25%     315.500000   2497.000000  3152.000000
50%     713.000000   3662.000000  4548.000000
75%    1096.000000   4776.500000  5956.000000
max    3410.000000   6946.000000  8714.000000
```

[5]:
```
#Change Column Names
bike_rent = bike_rent.rename(columns={'dteday':'date','yr':'year','mnth':
 ↪'month','weathersit':'weather_condition','temp':'temprature','atemp':
 ↪'feeling_temprature','hum':'humidity','casual':'casual_count','registered':
 ↪'registered_count','cnt':'total_count'})
```

[6]:
```
#Data Information
bike_rent.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 731 entries, 0 to 730
Data columns (total 16 columns):
instant               731 non-null int64
date                  731 non-null object
season                731 non-null int64
year                  731 non-null int64
month                 731 non-null int64
holiday               731 non-null int64
weekday               731 non-null int64
workingday            731 non-null int64
weather_condition     731 non-null int64
temprature            731 non-null float64
feeling_temprature    731 non-null float64
humidity              731 non-null float64
windspeed             731 non-null float64
casual_count          731 non-null int64
registered_count      731 non-null int64
total_count           731 non-null int64
dtypes: float64(4), int64(11), object(1)
memory usage: 91.5+ KB
```

1 There is no missing data.

[7]:
```
bike_rent.head()
```

[7]:
```
   instant        date  season  year  month  holiday  weekday  workingday  \
0        1  2011-01-01       1     0      1        0        6           0
1        2  2011-01-02       1     0      1        0        0           0
```

```
2          3  2011-01-03       1      0       1          0         1             1
3          4  2011-01-04       1      0       1          0         2             1
4          5  2011-01-05       1      0       1          0         3             1

   weather_condition  temprature  feeling_temprature  humidity  windspeed  \
0                  2    0.344167            0.363625  0.805833   0.160446
1                  2    0.363478            0.353739  0.696087   0.248539
2                  1    0.196364            0.189405  0.437273   0.248309
3                  1    0.200000            0.212122  0.590435   0.160296
4                  1    0.226957            0.229270  0.436957   0.186900

   casual_count  registered_count  total_count
0           331               654          985
1           131               670          801
2           120              1229         1349
3           108              1454         1562
4            82              1518         1600
```

[8]:
```
plot_bike_rent = bike_rent.copy()
```

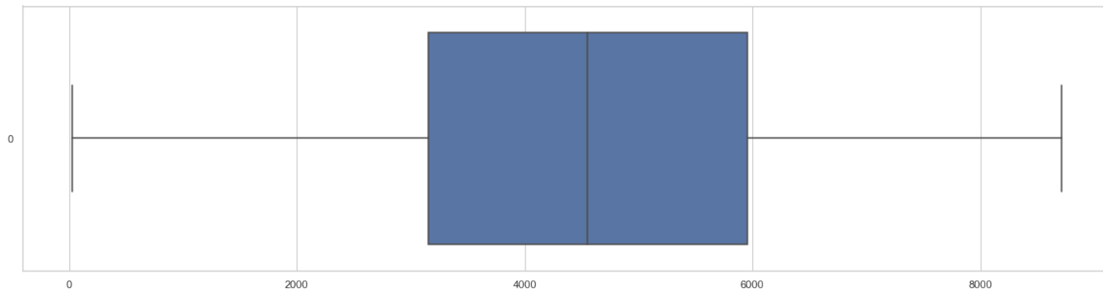
[9]:
```
plot_bike_rent['season']=bike_rent.season.map({1:'springer', 2:'summer', 3:
 ↪'fall', 4:'winter'})
plot_bike_rent['year']=bike_rent.year.map({0: '2011', 1:'2012'})
plot_bike_rent['month']=bike_rent.month.map({1:'January',2:'February',3:
 ↪'March',4:'April',5:'May',6:'June',7:'July',8:'August',9:'September',10:
 ↪'October',11:'November',12:'December'})
plot_bike_rent['holiday']=bike_rent.holiday.map({0:'No',1:'yes'})
plot_bike_rent['weekday']=bike_rent.weekday.map({0:'Sun',1:'Mon', 2:'Tue',3:
 ↪'Wed',4:'Thu',5:'Fri',6:'Sat'})
plot_bike_rent['workingday']=bike_rent.workingday.map({0:'Holiday/Weekend',1:
 ↪'Working_day'})
plot_bike_rent['weather_condition']=bike_rent.weather_condition.map({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'})
```

[10]:
```
temp_var =␣
 ↪['season','year','month','holiday','weekday','workingday','weather_condition']
for var in temp_var:
    bike_rent[var] = bike_rent[var].astype("category")
```

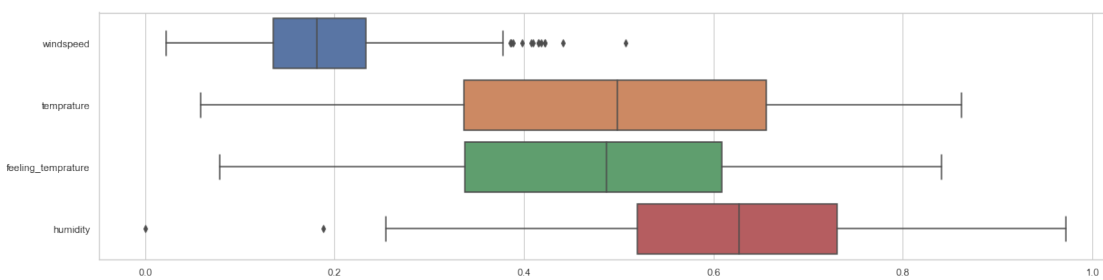
2 Outliner Analysis

In this data set we will check outliner analysis for 'float64' and 'int64' data types only.

[11]:
```
sns.set(style="whitegrid")
%matplotlib inline
plt.figure(figsize = (20,5))
box_plot_total = sns.boxplot(data=bike_rent['total_count'],orient='h')
box_plot_total.figure.savefig("box_plot_total.png")
```



[12]:
```
sns.set(style="whitegrid")
%matplotlib inline
plt.figure(figsize = (20,5))
box_plot = sns.
 ↪boxplot(data=bike_rent[{'temprature','feeling_temprature','humidity','windspeed'}],orient='
box_plot.figure.savefig("box_plot.png")
```



[13]:
```
#Outliner Analysis
q75, q25 = np.percentile(bike_rent['feeling_temprature'], [75 ,25])

#Calculate IQR
iqr = q75 - q25

#Calculate inner and outer fence
minimum = q25 - (iqr*1.5)
maximum = q75 + (iqr*1.5)

#Replace with NA
bike_rent.feeling_temprature[bike_rent.feeling_temprature < minimum] = np.nan
```

```
bike_rent.feeling_temprature[bike_rent.feeling_temprature > maximum] = np.nan
pd.DataFrame(bike_rent.isnull().sum())
```

```
/Users/divyanggor/anaconda3/lib/python3.7/site-
packages/ipykernel_launcher.py:12: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas-
docs/stable/indexing.html#indexing-view-versus-copy
  if sys.path[0] == '':
/Users/divyanggor/anaconda3/lib/python3.7/site-
packages/ipykernel_launcher.py:13: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas-
docs/stable/indexing.html#indexing-view-versus-copy
  del sys.path[0]
```

[13]:

	0
instant	0
date	0
season	0
year	0
month	0
holiday	0
weekday	0
workingday	0
weather_condition	0
temprature	0
feeling_temprature	0
humidity	0
windspeed	0
casual_count	0
registered_count	0
total_count	0

[14]:

```
#Outliner Analysis
q75, q25 = np.percentile(bike_rent['temprature'], [75 ,25])

#Calculate IQR
iqr = q75 - q25

#Calculate inner and outer fence
minimum = q25 - (iqr*1.5)
maximum = q75 + (iqr*1.5)

#Replace with NA
bike_rent.temprature[bike_rent.temprature < minimum] = np.nan
```

```
bike_rent.temprature[bike_rent.temprature > maximum] = np.nan
pd.DataFrame(bike_rent.isnull().sum())
```

```
/Users/divyanggor/anaconda3/lib/python3.7/site-
packages/ipykernel_launcher.py:12: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas-
docs/stable/indexing.html#indexing-view-versus-copy
  if sys.path[0] == '':
/Users/divyanggor/anaconda3/lib/python3.7/site-
packages/ipykernel_launcher.py:13: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas-
docs/stable/indexing.html#indexing-view-versus-copy
  del sys.path[0]
```

[14]:

	0
instant	0
date	0
season	0
year	0
month	0
holiday	0
weekday	0
workingday	0
weather_condition	0
temprature	0
feeling_temprature	0
humidity	0
windspeed	0
casual_count	0
registered_count	0
total_count	0

[15]:

```
#Outliner Analysis
q75, q25 = np.percentile(bike_rent['windspeed'], [75 ,25])

#Calculate IQR
iqr = q75 - q25

#Calculate inner and outer fence
minimum = q25 - (iqr*1.5)
maximum = q75 + (iqr*1.5)

#Replace with NA
bike_rent.windspeed[bike_rent.windspeed < minimum] = np.nan
```

```
bike_rent.windspeed[bike_rent.windspeed > maximum] = np.nan
pd.DataFrame(bike_rent.isnull().sum())
```

```
/Users/divyanggor/anaconda3/lib/python3.7/site-
packages/ipykernel_launcher.py:12: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas-
docs/stable/indexing.html#indexing-view-versus-copy
  if sys.path[0] == '':
/Users/divyanggor/anaconda3/lib/python3.7/site-
packages/ipykernel_launcher.py:13: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas-
docs/stable/indexing.html#indexing-view-versus-copy
  del sys.path[0]
```

[15]:

	0
instant	0
date	0
season	0
year	0
month	0
holiday	0
weekday	0
workingday	0
weather_condition	0
temprature	0
feeling_temprature	0
humidity	0
windspeed	13
casual_count	0
registered_count	0
total_count	0

[16]:

```
#Outliner Analysis
q75, q25 = np.percentile(bike_rent['humidity'], [75 ,25])

#Calculate IQR
iqr = q75 - q25

#Calculate inner and outer fence
minimum = q25 - (iqr*1.5)
maximum = q75 + (iqr*1.5)

#Replace with NA
bike_rent.humidity[bike_rent.humidity < minimum] = np.nan
```

```
bike_rent.humidity[bike_rent.humidity > maximum] = np.nan
pd.DataFrame(bike_rent.isnull().sum())
```

```
/Users/divyanggor/anaconda3/lib/python3.7/site-
packages/ipykernel_launcher.py:12: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas-
docs/stable/indexing.html#indexing-view-versus-copy
  if sys.path[0] == '':
/Users/divyanggor/anaconda3/lib/python3.7/site-
packages/ipykernel_launcher.py:13: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas-
docs/stable/indexing.html#indexing-view-versus-copy
  del sys.path[0]
```

[16]:

	0
instant	0
date	0
season	0
year	0
month	0
holiday	0
weekday	0
workingday	0
weather_condition	0
temprature	0
feeling_temprature	0
humidity	2
windspeed	13
casual_count	0
registered_count	0
total_count	0

[17]:

```
#Missing Value Analysis
missing_value = pd.DataFrame(bike_rent.isnull().sum())
missing_value = missing_value.reset_index()
missing_value = missing_value.rename(columns = {'index':'variables',0:
 ↪'missing_percentage'})
missing_value['missing_percentage']=(missing_value['missing_percentage']/
 ↪len(bike_rent))*100
missing_value = missing_value.sort_values('missing_percentage', ascending=␣
 ↪False)
missing_value
```

[17]:

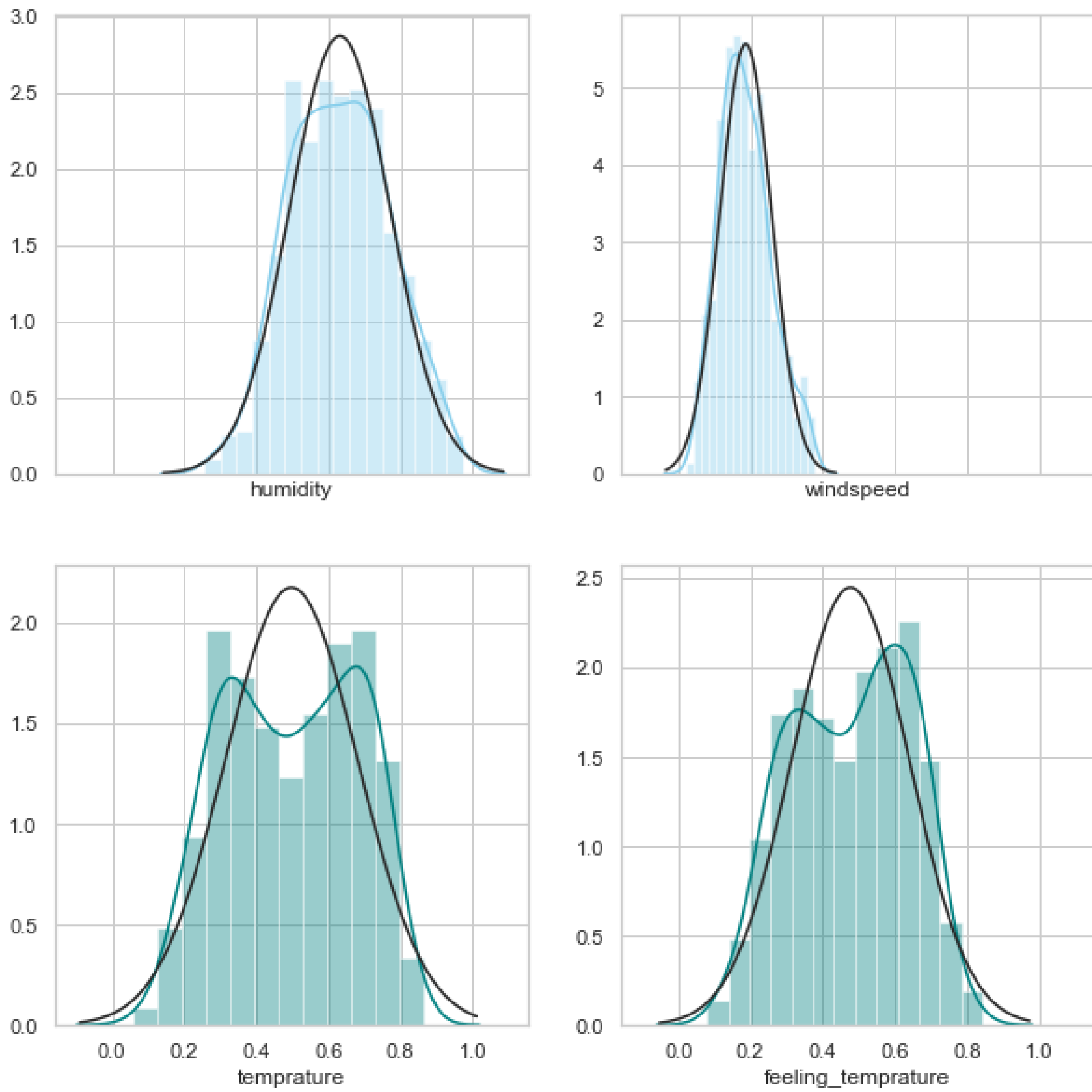
	variables	missing_percentage
12	windspeed	1.778386
11	humidity	0.273598
0	instant	0.000000
1	date	0.000000
2	season	0.000000
3	year	0.000000
4	month	0.000000
5	holiday	0.000000
6	weekday	0.000000
7	workingday	0.000000
8	weather_condition	0.000000
9	temprature	0.000000
10	feeling_temprature	0.000000
13	casual_count	0.000000
14	registered_count	0.000000
15	total_count	0.000000

Missing values are very less in percentage so we can drop those values.

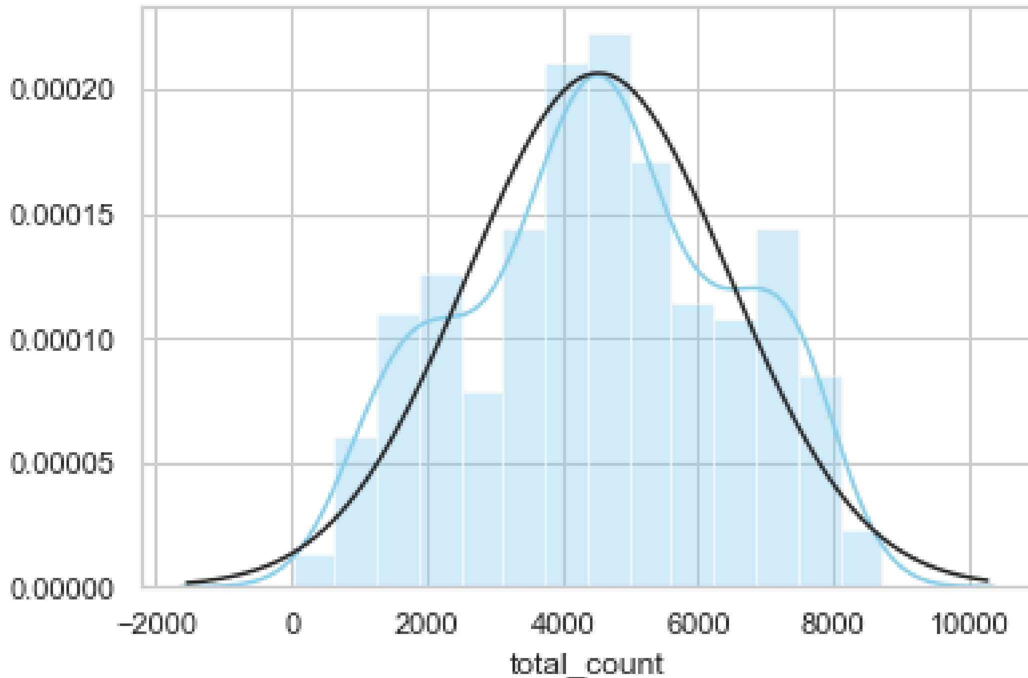
[18]:
```
bike_rent = bike_rent.dropna()
```

3 Data Visulization

[19]:
```
f, axes = plt.subplots(2,2, figsize=(10, 10), sharex=True)
sns.distplot( bike_rent["humidity"],fit=norm , color="skyblue", ax=axes[0,0])
sns.distplot( bike_rent["windspeed"],fit=norm , color="skyblue", ax=axes[0,1])
sns.distplot( bike_rent["temprature"],fit=norm , color="teal", ax=axes[1, 0])
sns.distplot( bike_rent["feeling_temprature"],fit=norm , color="teal",␣
 ↪ax=axes[1, 1])
f.savefig("norm.png")
```

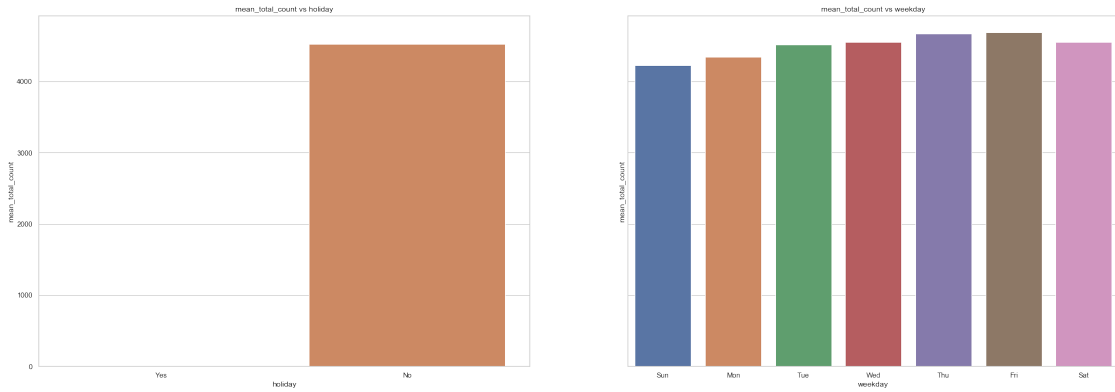


[20]:
```
norm_1 = sns.distplot( bike_rent["total_count"],fit=norm , color="skyblue")
norm_1.figure.savefig("norm_1")
```

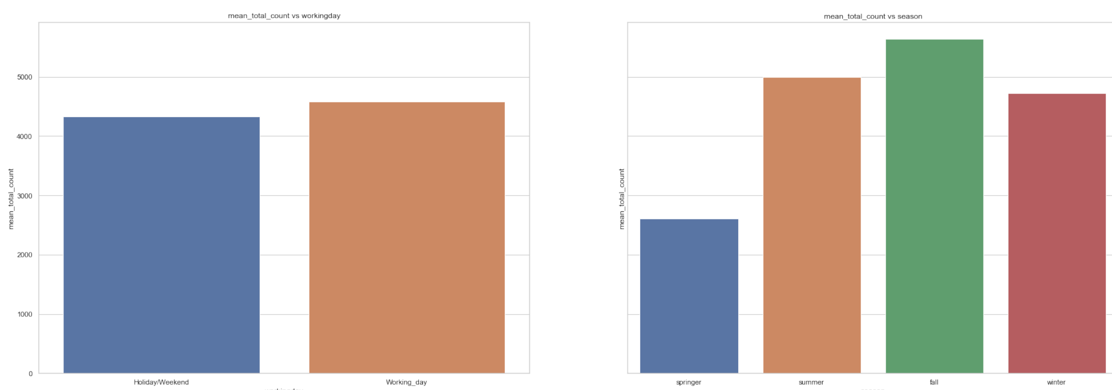


[21]:
```
def groupandplot(data,groupby_key,value,sortorder,axes,aggregate='mean'):
    agg_data=data.groupby([groupby_key])[value].agg(aggregate).reset_index().
 ↪rename(columns={value:aggregate+'_'+value})
    count_data=data.groupby([groupby_key])['total_count'].count().reset_index().
 ↪rename(columns={'total_count':'Num_bike_rent'})
    plot = sns.
 ↪barplot(x=groupby_key,y=aggregate+'_'+value,data=agg_data,order=sortorder,ax␣
 ↪= axes).set_title(aggregate+'_'+value+" vs "+groupby_key)
```

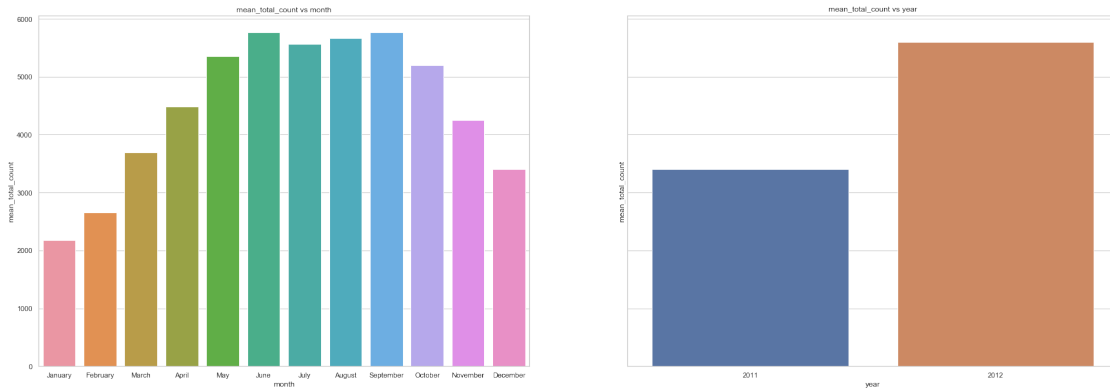
[22]:
```
f, axes = plt.subplots(1,2,sharex='col', sharey='row', figsize=(30, 10))
groupandplot(plot_bike_rent,'holiday','total_count',['Yes','No'],axes[0])
groupandplot(plot_bike_rent,'weekday','total_count',['Sun','Mon',␣
 ↪'Tue','Wed','Thu','Fri','Sat'],axes[1])
f.savefig("b_1.png")
```



[23]:
```
f, axes = plt.subplots(1,2,sharex='col', sharey='row', figsize=(30, 10))
groupandplot(plot_bike_rent,'workingday','total_count',['Holiday/
↪Weekend','Working_day'],axes[0])
groupandplot(plot_bike_rent,'season','total_count',['springer', 'summer',␣
↪'fall', 'winter'],axes[1])
f.savefig("b_2.png")
```



[24]:
```
f, axes = plt.subplots(1,2,sharex='col', sharey='row', figsize=(30, 10))
groupandplot(plot_bike_rent,'month','total_count',["January","February","March","April","May",
groupandplot(plot_bike_rent,'year','total_count',['2011','2012'],axes[1])
f.savefig("b_3.png")
```

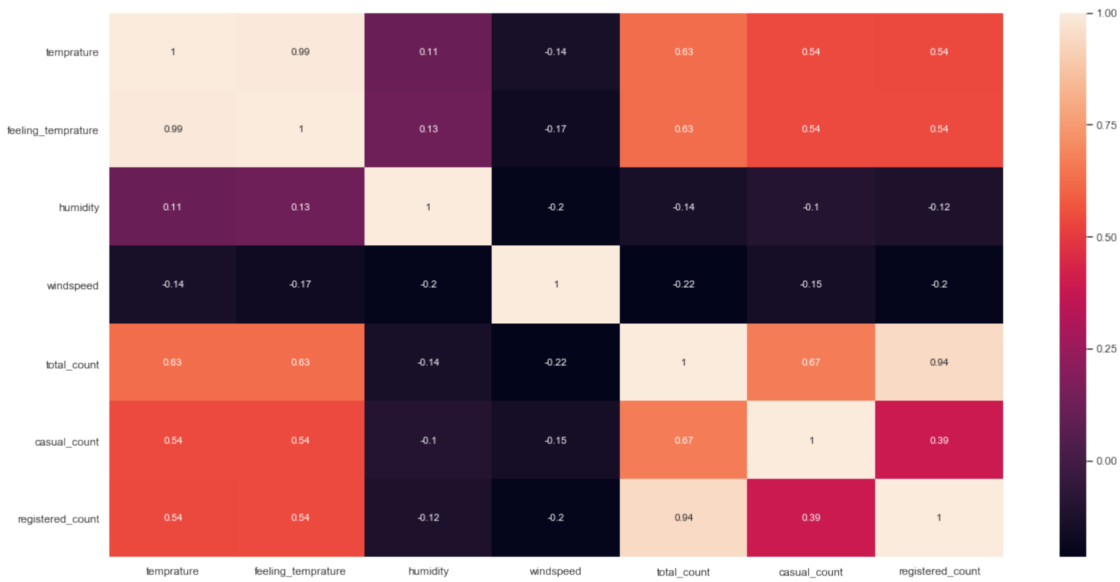


[25]:
```
corr =␣
 ↪bike_rent[['temprature','feeling_temprature','humidity','windspeed','total_count','casual_c
 ↪corr()
corr
```

[25]:
```
                    temprature  feeling_temprature  humidity  windspeed  \
temprature            1.000000            0.991738  0.114191  -0.140169
feeling_temprature    0.991738            1.000000  0.126587  -0.166038
humidity              0.114191            0.126587  1.000000  -0.204496
windspeed            -0.140169           -0.166038 -0.204496   1.000000
total_count           0.625892            0.629204 -0.136621  -0.216193
casual_count          0.539714            0.540234 -0.101439  -0.146178
registered_count      0.538095            0.541977 -0.124701  -0.203677

                    total_count  casual_count  registered_count
temprature             0.625892      0.539714          0.538095
feeling_temprature     0.629204      0.540234          0.541977
humidity              -0.136621     -0.101439         -0.124701
windspeed             -0.216193     -0.146178         -0.203677
total_count            1.000000      0.670547          0.944581
casual_count           0.670547      1.000000          0.389848
registered_count       0.944581      0.389848          1.000000
```

[26]:
```
plt.figure(figsize=(20,10))
cor_plot = sns.heatmap(corr, annot=True)
cor_plot.figure.savefig('cor_plot.png')
```



Looking for continuous variables 'humidity' & 'windspeed' having very small value of coefficient of correlaiton so we will remove those variables.

[27]:
```
bike_rent = bike_rent.
↪drop(['date','instant','casual_count','registered_count'],axis=1)
```

[28]:
```
anova = ols('total_count ~ season + year + month + holiday + weekday +␣
↪workingday +weather_condition', data=bike_rent).fit()
```

[29]:
```
anova.summary()
```

[29]:
```
<class 'statsmodels.iolib.summary.Summary'>
"""
                            OLS Regression Results
==============================================================================
Dep. Variable:            total_count   R-squared:                       0.814
Model:                            OLS   Adj. R-squared:                  0.808
Method:                 Least Squares   F-statistic:                     126.5
Date:                Tue, 14 Jan 2020   Prob (F-statistic):          2.38e-234
Time:                        21:43:55   Log-Likelihood:                -5838.8
No. Observations:                 717   AIC:                         1.173e+04
Df Residuals:                     692   BIC:                         1.184e+04
Df Model:                          24
Covariance Type:            nonrobust
================================================================================
==========
                             coef    std err          t      P>|t|      [0.025
0.975]
--------------------------------------------------------------------------------
----------
Intercept                 1117.4563    137.871      8.105      0.000     846.761
```

```
1388.152
season[T.2]          911.1484    202.128      4.508      0.000    514.291
1308.006
season[T.3]         1126.4273    235.491      4.783      0.000    664.064
1588.791
season[T.4]         1783.4306    200.305      8.904      0.000   1390.153
2176.708
year[T.1]           2145.6381     63.462     33.810      0.000   2021.036
2270.240
month[T.2]           467.3018    159.305      2.933      0.003    154.522
780.082
month[T.3]          1159.7546    172.005      6.743      0.000    822.039
1497.470
month[T.4]          1425.2243    254.978      5.590      0.000    924.601
1925.848
month[T.5]          2207.4806    253.491      8.708      0.000   1709.777
2705.184
month[T.6]          2407.7204    246.461      9.769      0.000   1923.820
2891.621
month[T.7]          2079.2882    281.037      7.399      0.000   1527.501
2631.075
month[T.8]          2211.3636    280.878      7.873      0.000   1659.888
2762.839
month[T.9]          2361.5356    257.461      9.172      0.000   1856.037
2867.034
month[T.10]         1432.5742    252.956      5.663      0.000    935.922
1929.227
month[T.11]          262.2918    253.311      1.035      0.301   -235.059
759.643
month[T.12]          206.2783    202.044      1.021      0.308   -190.415
602.971
holiday[T.1]        -147.8748    176.542     -0.838      0.403   -494.498
198.748
weekday[T.1]        -134.1732     79.694     -1.684      0.093   -290.645
22.298
weekday[T.2]          20.6581     85.494      0.242      0.809   -147.200
188.516
weekday[T.3]          91.6804     85.822      1.068      0.286    -76.822
260.183
weekday[T.4]         114.4572     85.409      1.340      0.181    -53.236
282.150
weekday[T.5]         119.2324     85.018      1.402      0.161    -47.692
286.157
weekday[T.6]         462.2570    118.649      3.896      0.000    229.301
695.213
workingday[T.1]      359.7298     74.099      4.855      0.000    214.244
505.215
```

```
weather_condition[T.2]  -725.7950     68.834    -10.544      0.000    -860.944
-590.646
weather_condition[T.3] -2659.5167    198.905    -13.371      0.000   -3050.046
-2268.988
==============================================================================
Omnibus:                      127.816   Durbin-Watson:                   1.216
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              549.928
Skew:                          -0.750   Prob(JB):                    3.84e-120
Kurtosis:                       7.019   Cond. No.                     2.25e+15
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
[2] The smallest eigenvalue is 3.31e-28. This might indicate that there are
strong multicollinearity problems or that the design matrix is singular.
"""
```

[30]:
```python
aov_table = sm.stats.anova_lm(anova,type = 1)
aov_table
```

[30]:
```
                      df        sum_sq       mean_sq            F  \
season               3.0  9.218466e+08  3.072822e+08   427.956121
year                 1.0  8.717574e+08  8.717574e+08  1214.108425
month               11.0  1.840912e+08  1.673556e+07    23.307849
holiday              1.0  3.612964e+06  3.612964e+06     5.031825
weekday              6.0  1.457678e+07  2.429463e+06     3.383547
workingday           1.0  5.554650e+04  5.554650e+04     0.077360
weather_condition    2.0  1.840186e+08  9.200928e+07   128.142583
Residual           692.0  4.968717e+08  7.180227e+05          NaN

                          PR(>F)
season             3.746448e-157
year               2.113285e-154
month               6.624959e-41
holiday             2.520131e-02
weekday             2.694554e-03
workingday          7.809900e-01
weather_condition   4.527198e-48
Residual                     NaN
```

By anova test 'PR(>F)'>0.05 for variable 'workingday' so we will remove that variable

[31]:
```python
bike_rent = bike_rent.drop(['workingday'],axis=1)
```

[32]:
```python
X = bike_rent.drop('total_count',axis=1).values
y = bike_rent['total_count'].values
X_train_bike, X_test_bike, y_train_bike, y_test_bike = train_test_split(X, y,␣
↪test_size = 0.20, random_state=42)
```

```
print(X_train_bike.shape, X_test_bike.shape, y_train_bike.shape, y_test_bike.
 ↪shape)
```

```
(573, 10) (144, 10) (573,) (144,)
```

3.1 Model Selection

[33]:
```
table = PrettyTable()
table.field_names = ["Model",␣
 ↪"RMSE_Test","RMSE_Train","RMSLE_Test","RMSLE_Train","Rš score"]

models = [
    LinearRegression(),
    DecisionTreeRegressor(),Ridge(),Lasso(),
    RandomForestRegressor( random_state=0, n_estimators=300),
    XGBRegressor(n_estimators=100)
]
for model in models:
    model.fit(X_train_bike, y_train_bike)
    y_pred = model.predict(X_test_bike)
    y_pred_train = model.predict(X_train_bike)
    RMSE_test = np.sqrt(mean_squared_error(y_test_bike, y_pred))
    RMSLE_test = np.sqrt(mean_squared_log_error(y_test_bike, y_pred))
    RMSE_train = np.sqrt(mean_squared_error(y_train_bike, y_pred_train))
    RMSLE_train = np.sqrt(mean_squared_log_error(y_train_bike, y_pred_train))

    mse = mean_squared_error(y_pred, y_test_bike)
    msle = mean_squared_log_error(y_test_bike, y_pred)
    score = model.score(X_test_bike, y_test_bike)

    table.add_row([type(model).__name__, format(RMSE_test, '.
 ↪5f'),format(RMSLE_test,'.5f'),format(RMSE_train,'.5f'),format(RMSLE_train,'.
 ↪5f') ,format(score, '.5f')])

print(table)
```

```
[21:43:56] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.
+-----------------------+-----------+------------+------------+-------------+---
-------+
|         Model         | RMSE_Test | RMSE_Train | RMSLE_Test | RMSLE_Train | Rš
score |
+-----------------------+-----------+------------+------------+-------------+---
-------+
|    LinearRegression   | 833.41690 |  0.49307   | 877.39306  |   0.29205   |
0.81558  |
| DecisionTreeRegressor | 923.33439 |  0.51769   |  0.00000   |   0.00000   |
```

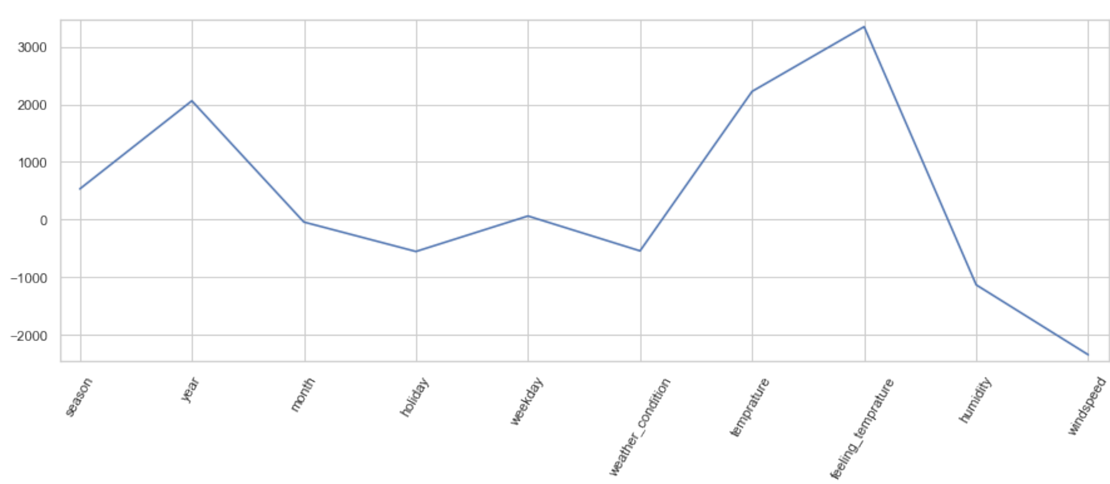
```
0.77364  |
|         Ridge          | 836.69768 |  0.49334   | 879.35492  |   0.26856   |
0.81413  |
|         Lasso          | 834.53671 |  0.49312   | 877.63980  |   0.27606   |
0.81509  |
| RandomForestRegressor | 699.26006 |  0.47783   | 247.17488  |   0.10899   |
0.87018  |
|      XGBRegressor      | 625.86782 |  0.44641   | 432.96809  |   0.13594   |
0.89600  |
+-----------------------+-----------+------------+------------+------------+---
-------+
```

[34]:
```
X_train_bike_df = pd.DataFrame.from_records(X_train_bike)
columns = bike_rent.columns
columns = columns.delete(10)
X_train_bike_df.columns = columns
```

[35]:
```
def plot_regression(model,X_train, y_train):
    reg_coef_m = model.fit(X_train,y_train).coef_
    print(reg_coef_m)
    # Plot the coefficients
    plt.figure(figsize=(15,5))
    plt.plot(range(len(X_train_bike_df.columns)), reg_coef_m)
    plt.xticks(range(len(X_train_bike_df.columns)), X_train_bike_df.columns.
 ↪values, rotation=60)
    plt.margins(0.02)
    plt.show()
```

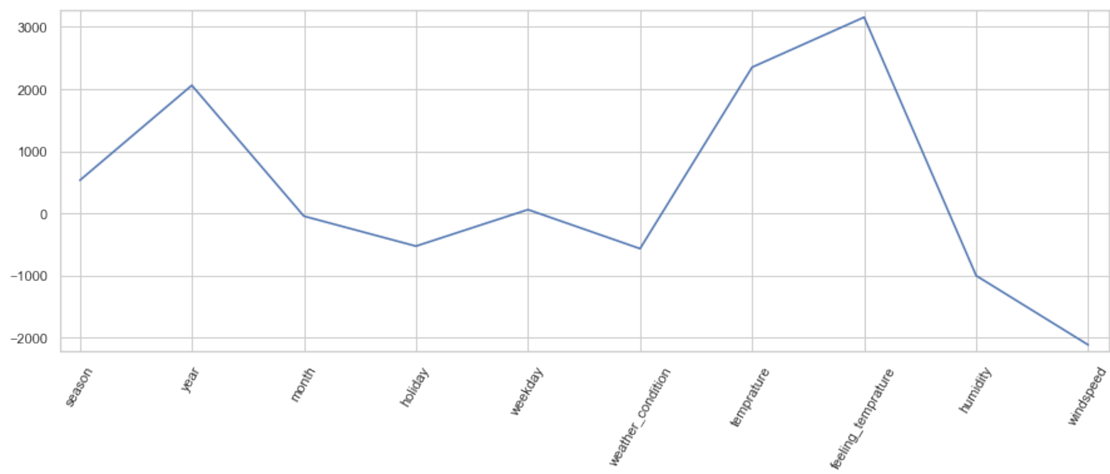
[36]:
```
plot_lr = plot_regression(LinearRegression(),X_train_bike,y_train_bike)
```

```
[  530.04336032  2059.97010786   -44.3727868   -555.48373759
    60.64546803  -545.29752752  2223.58743284  3346.57580094
 -1134.69955114 -2348.26555495]
```



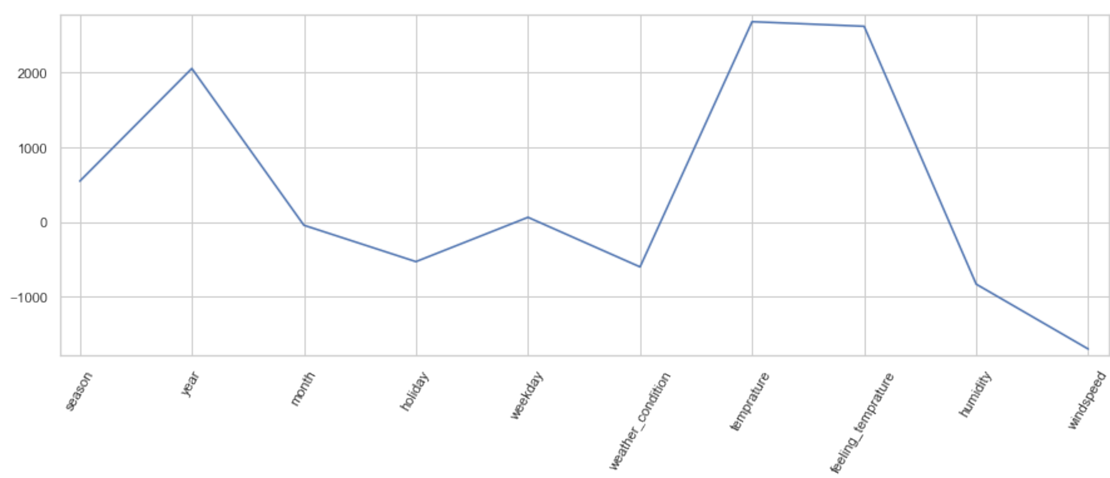
[37]:
```
plot_regression(Lasso(),X_train_bike,y_train_bike)
```

```
[  531.48621761  2060.49766782   -44.08714524  -526.74349773
    60.79339808  -567.23592406  2351.62657408  3156.81593383
 -1001.99613756 -2115.81596532]
```



[38]:
```
plot_regression(Ridge(),X_train_bike,y_train_bike)
```

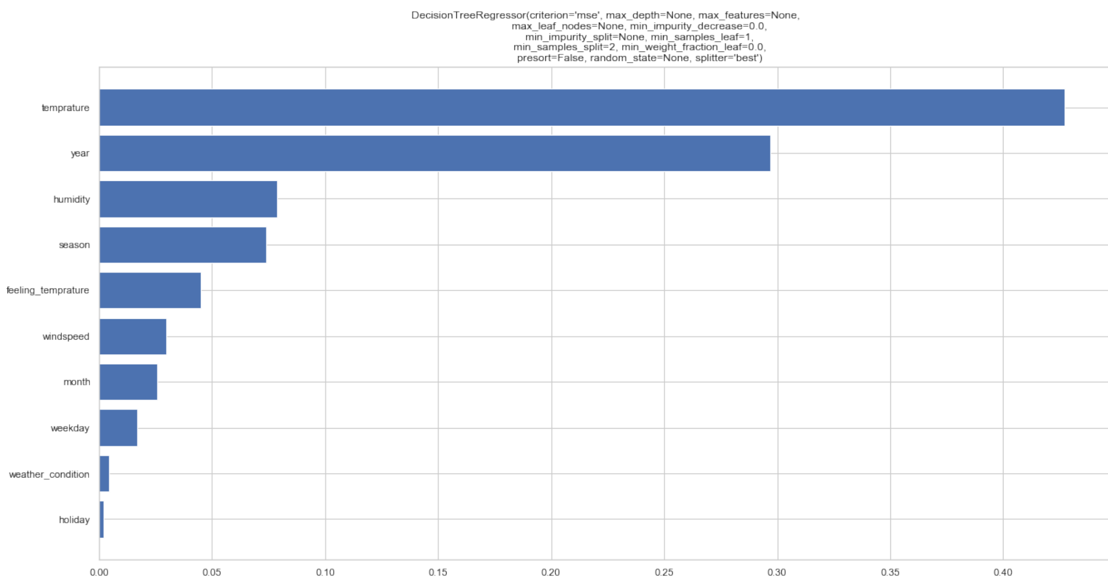
```
[  545.56622755  2057.528253     -45.37369828  -533.85559998
    60.8754719   -604.57490501  2685.40543149  2623.18219349
  -835.80689689 -1706.28725435]
```



[39]:
```
def plot_importance(model, X_train_bike, y_train_bike):
    # Creating plot
    fig = plt.figure(figsize=(20,10))
    plt.title(model)
    tree_features = model.fit(X_train_bike,y_train_bike).feature_importances_
    print(tree_features)
    indices = np.argsort(tree_features)[::1]
    names = [X_train_bike_df.columns[i] for i in indices]
    # Add horizontal bars
    plt.barh(range(pd.DataFrame(X_train_bike).
 ↪shape[1]),tree_features[indices],align = 'center')
    plt.yticks(range(pd.DataFrame(X_train_bike).shape[1]), names)
    plt.show()
```

[40]:
```
print('DecisionTreeRegressor',plot_importance(DecisionTreeRegressor(),X_train_bike,y_train_bik
```

```
[0.07384051 0.29715768 0.02581183 0.00181199 0.01673451 0.00426195
 0.42717015 0.04487751 0.07856211 0.02977176]
```



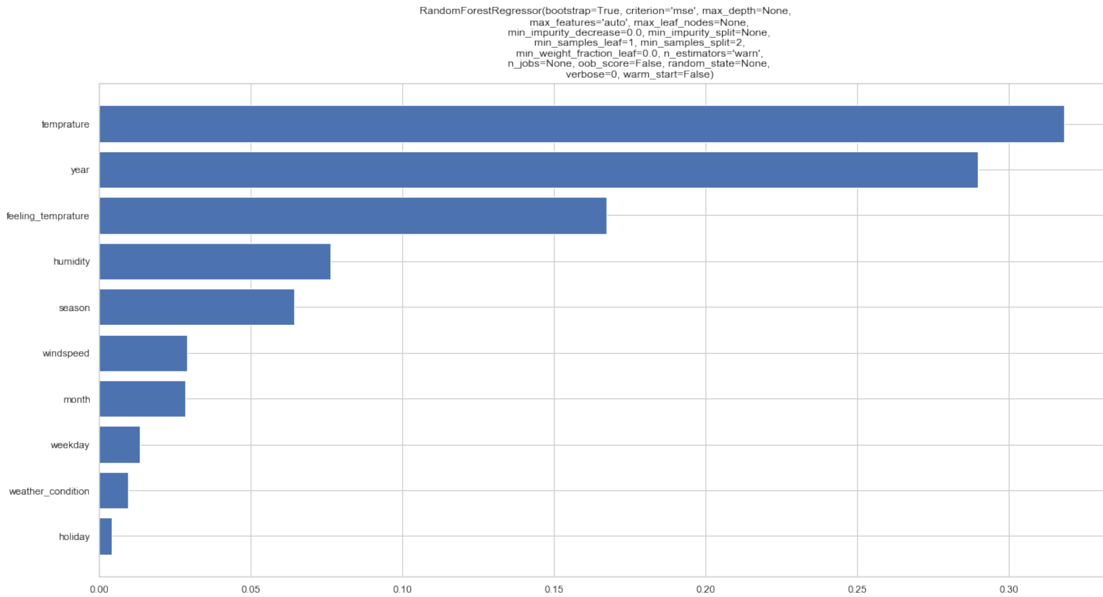

```
DecisionTreeRegressor None
```

[41]:
```
print('RandomForestRegressor',plot_importance(RandomForestRegressor(),X_train_bike,y_train_bik
```

```
[0.06429511 0.28968424 0.02827882 0.00422449 0.01334353 0.00964519
 0.31814405 0.16722259 0.07621522 0.02894676]
```

```
/Users/divyanggor/anaconda3/lib/python3.7/site-
packages/sklearn/ensemble/forest.py:245: FutureWarning: The default value of
```

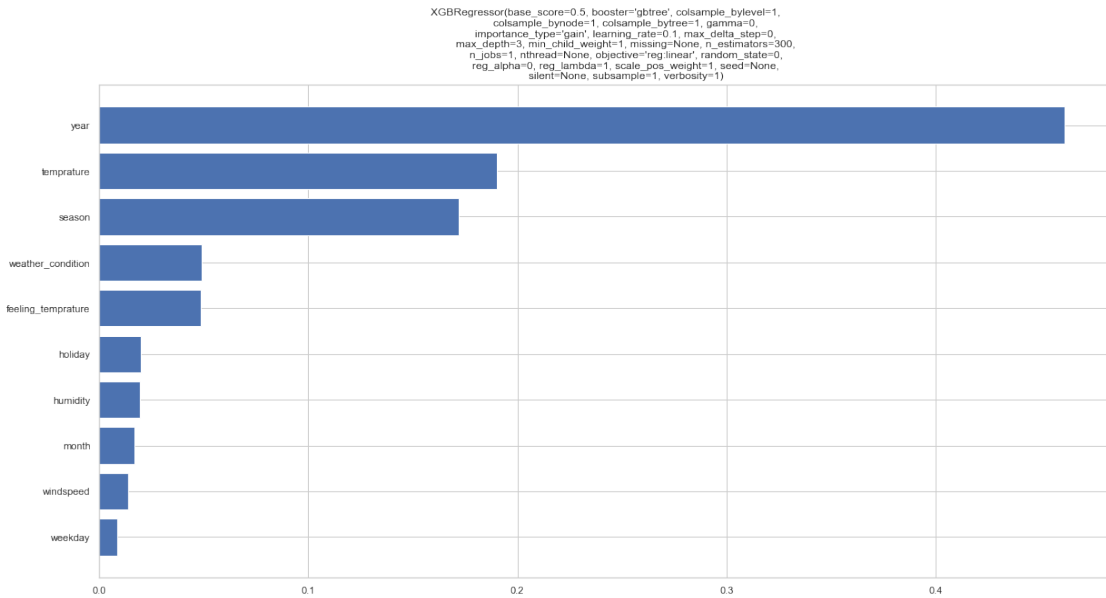
```
n_estimators will change from 10 in version 0.20 to 100 in 0.22.
  "10 in version 0.20 to 100 in 0.22.", FutureWarning)
```




```
RandomForestRegressor None
```

[42]:
```
print('XGBRegressor',plot_importance(XGBRegressor( random_state=0,␣
 ↪n_estimators=300),X_train_bike,y_train_bike))
```

```
[21:43:58] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.
[0.17181417 0.4615058  0.01699146 0.01979753 0.00865801 0.04914481
 0.19019859 0.0485683  0.01941875 0.01390261]
```




```
XGBRegressor None
```

[43]:
```python
XGBRegressor( random_state=0, n_estimators=300).fit(X_train_bike, y_train_bike)
y_pred = model.predict(X_test_bike)
```

```
[21:43:58] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.
```

[44]:
```python
y_pred
```

[44]:
```
array([4248.094  , 3296.7976 , 1410.8702 , 7176.048  , 6958.3223 ,
       6977.85   , 1404.0486 , 6587.996  , 1784.6526 , 3171.3203 ,
       5657.662  , 2046.2527 , 4631.69   , 2575.8713 , 4502.7627 ,
       4259.705  , 4566.099  , 4244.47   , 2552.6697 , 6490.8105 ,
       1920.5403 , 4531.2285 , 3892.5786 , 7383.8022 , 4540.75   ,
       3568.1277 , 4027.5376 , 2494.7432 ,  914.70514, 2984.2283 ,
       1781.3733 , 2507.1846 , 3928.576  , 5725.326  , 2397.875  ,
       2237.4111 , 3701.4094 , 7263.1895 , 4568.774  , 6700.198  ,
       4272.1733 , 6830.1035 , 7742.3257 , 4117.9023 , 5012.8174 ,
       1004.56464, 3245.9207 , 7126.9033 , 4075.3318 , 5050.302  ,
       4178.8115 , 4804.7397 , 7133.397  , 2035.8655 , 4429.6216 ,
       7381.745  , 1412.4048 , 4037.183  , 6463.4443 , 6434.7026 ,
        939.834  , 2570.37   , 2121.776  , 4805.8125 , 4668.6455 ,
       6470.6855 , 4375.8613 , 2965.8577 , 7344.9136 , 5528.701  ,
       6446.9624 , 4576.2007 , 7093.3135 , 4098.6157 , 3627.0098 ,
       3495.6062 , 4687.1484 , 7161.7676 , 2105.5142 , 6033.953  ,
       6579.7363 , 7214.258  , 1049.9711 , 3936.9019 , 5655.86   ,
```

```
6456.4385 , 2571.9753 , 5934.825  , 7506.6646 , 1788.38   ,
5039.5874 , 3246.7844 , 4174.2046 , 3662.6033 , 2261.1848 ,
1398.0944 , 4775.175  , 5814.384  , 6167.957  , 1863.3438 ,
5256.659  , 3058.698  , 3582.9478 , 3949.8013 , 6550.3374 ,
3705.366  , 6962.9136 , 3164.7925 , 1914.5303 , 7231.5337 ,
5160.669  , 3085.4468 , 4412.3936 , 3041.4878 , 6733.004  ,
1746.8986 , 4675.561  , 1233.6024 , 2148.378  , 6544.6333 ,
2829.9878 , 1697.5774 , 4898.067  , 1596.0609 , 3978.226  ,
3463.923  , 5191.241  , 3555.7349 , 3522.7148 , 4462.104  ,
4404.1733 , 6621.9995 , 4123.6646 , 3946.1719 , 1450.2169 ,
4108.158  , 7002.468  , 4072.4956 , 5151.558  , 3811.5137 ,
4088.8467 , 1590.0984 , 3587.084  , 4211.946  ], dtype=float32)
```

Appendix C

Complete R Code

```
rm(list = ls())
setwd("/Users/divyanggor/Documents/Study/Online_Course/Edwisor/Project/project_2/")

# #loading Libraries
x = c("plyr","ggplot2", "corrgram", "DMwR", "usdm", "caret", "randomForest", "e1071",
      "DataCombine", "doSNOW", "inTrees", "rpart.plot", "rpart",'MASS','xgboost','stats',
'gdistance', 'Imap', 'car',"Metrics")
#load Packages
lapply(x, require, character.only = TRUE)
rm(x)

bike_rent= read.csv("day.csv")
summary(bike_rent)
head(bike_rent,5)
#########Changing coulmn Names#########
colnames(bike_rent) = c("instant","date","season","year","month","holiday","weekday",
"workingday","weather_condition","temprature","feeling_temprature","humidity","windspeed","casu

######Missing Value Analysis######
apply(bike_rent,2,function(x){sum(is.na(x))})

#Ther is no Missing Value in data.

plot_bike_rent = bike_rent

#plot_bike_rent$season[plot_bike_rent$season==1]="springer"
```

```
#plot_bike_rent$season[plot_bike_rent$season==2]="summer"
#plot_bike_rent$season[plot_bike_rent$season==3]="fall"
#plot_bike_rent$season[plot_bike_rent$season==4]="winter"
#plot_bike_rent$year[plot_bike_rent$year==0]=2011
#plot_bike_rent$year[plot_bike_rent$year==1]=2012
#head(plot_bike_rent,5)

###########Feature Engineering#########

str(bike_rent)
cols = c('season','year','month','holiday','weekday','workingday','weather_condition')
bike_rent[, cols] = lapply(bike_rent[, cols], factor)
str(bike_rent)

#########Outliner Analysis################
# Boxplot for total_count variable
pl1 = ggplot(bike_rent,aes(y = total_count))
pl1 + geom_boxplot(outlier.colour="red", fill = "grey" ,outlier.shape=18,outlier.size=1,
  notch=FALSE)+ylim(0,100)

boxplot(bike_rent[,"total_count"])
boxplot(bike_rent[,c('temprature','feeling_temprature','humidity','windspeed')])

values = bike_rent[,'windspeed'] %in% boxplot.stats(bike_rent[,'windspeed'])$out
bike_rent[which(values),'windspeed'] = NA

values = bike_rent[,'humidity'] %in% boxplot.stats(bike_rent[,'humidity'])$out
bike_rent[which(values),'humidity'] = NA

apply(bike_rent,2,function(x){sum(is.na(x))})
#here very less number of missing values so we can drop those values.

bike_rent = na.omit(bike_rent)

###############  Feature selection ##############
numeric = sapply(bike_rent,is.numeric) #selecting numeric variables
numeric_data = bike_rent[,numeric]
cnames = colnames(numeric_data)
#Correlation analysis for numeric variables
```

```
cor(numeric_data)
corrgram(bike_rent[,numeric],upper.panel=panel.pie, main = "Correlation Plot")
#Drop unnacessary variables
bike_rent = subset(bike_rent,select=-c(date,instant, casual_count,registered_count))
#Anova Test
aov_results = aov(total_count ~ season + year + month + holiday + workingday+ weekday
+ weather_condition,data = bike_rent)
summary(aov_results)

# workingday has p value greater than 0.05
bike_rent = subset(bike_rent,select=-workingday)
#################### Splitting train into train and validation subsets ###################
set.seed(42)
tr = createDataPartition(bike_rent$total_count,p=0.80,list = FALSE) # 80% in trainin
and 20% in Test Datasets
train_bike_rent = bike_rent[tr,]
test_bike_rent = bike_rent[-tr,]



############# Linear regression #################
lm_model = lm(total_count ~.,data=bike_rent)

summary(lm_model)
plot(lm_model$fitted.values,rstandard(lm_model),main = "Residual plot",
     xlab = "Predicted values of fare_amount",
     ylab = "standardized residuals")

lm_predictions = predict(lm_model,test_bike_rent[,1:10])
qplot(x = test_bike_rent[,11], y = lm_predictions, data = test_bike_rent, color = I("blue"),
geom = "point")
regr.eval(test_bike_rent[,11],lm_predictions)
library(Metrics)
rmsle(lm_predictions,test_bike_rent[,11])

############# Decision Tree ####################

Dt_model = rpart(total_count ~.,data=bike_rent, method = "anova")
summary(Dt_model)
#Predict for new test cases
```

```r
  predictions_DT = predict(Dt_model, test_bike_rent[,1:10])
qplot(x = test_bike_rent[,11], y = predictions_DT, data = test_bike_rent, color = I("blue"),
   geom = "point")
regr.eval(test_bike_rent[,11],predictions_DT)
rmsle(predictions_DT,test_bike_rent[,11])

############ Random forest #####################
rf_model = randomForest(total_count ~.,data=bike_rent)
summary(rf_model)
rf_predictions = predict(rf_model,test_bike_rent[,1:10])
qplot(x = test_bike_rent[,11], y = rf_predictions, data = test_bike_rent, color = I("blue"),
geom = "point")
regr.eval(test_bike_rent[,11],rf_predictions)
rmsle(rf_predictions,test_bike_rent[,11])

########### Improving Accuracy by using Ensemble technique-XGBOOST ##########################
train_data_matrix = as.matrix(sapply(train_bike_rent[-11],as.numeric))
test_data_data_matrix = as.matrix(sapply(test_bike_rent[-11],as.numeric))
xgboost_model = xgboost(data = train_data_matrix,label = train_bike_rent$total_count,
nrounds = 50,verbose = FALSE)
summary(xgboost_model)
xgb_predictions = predict(xgboost_model,test_data_data_matrix)
qplot(x = test_bike_rent[,11], y = xgb_predictions, data = test_bike_rent, color = I("blue"),
 geom = "point")
regr.eval(test_bike_rent[,11],xgb_predictions)
rmsle(xgb_predictions,test_bike_rent[,11])
```

References

- https://medium.com/greyatom/a-quick-guide-to-boosting-in-ml-acf7c1585cb5
- https://xgboost.readthedocs.io/en/latest/tutorials/model.html