Bike Rental Prediction

Vishal Maurya

23 September 2019

Contents

1 Introduction

1.1 Problem Statement 3
1.2 Data 3

2 Methodology

2.1 Exploratory Data Analysis 6
2.1.1 Descriptive Analysis 6
2.1.1.1 Features Analysis 6
2.1.1.1 Missing Value Analysis 7
2.1.1.3 Target Variable Analysis 7
2.1.2 Visualization 8
2.1.2.1 Attributes Distributions and trends 8
2.1.2.2 Outlier Analysis 16
2.2 Data Preprocessing and Analysis 18
2.2.1 Outlier Handling 18
2.2.2 Feature Selection 18
2.2.3 Feature Engineering 19
2.3 Modeling 21
2.3.1 Model Selection 21
2.3.2 Multilinear & Regularization Regression 21
2.3.3 Random Forest 25
2.3.4 Gradient Boosting 27

3 Conclusion

3.1 Model Evaluation 29
3.1.1 R-squared & Adj R-squared Value 33
3.1.2 Root Mean Squared Error & Root Mean Squared Log Error 33
3.2 Model Selection 34

Appendix A - Extra Figures

Appendix B – Complete Python and R Code

Python Code 45
R Code 66

References

Chapter 1
Introduction

1.1 Problem Statement

The usage of bicycles as a mode of transportation has gained traction in recent years due to with environmental and health issues. The cities across the world have successfully rolled out bike sharing programs to encourage usage of bikes. Under such programs, the riders can rent bicycles using manual or automated stalls spread across the city for defined periods. In most cases, riders can pick up bikes from one location and returned them any other designated place. The bike sharing programs from across the world are hotspots of all sorts of data, ranging from travel time, start and end location, demographics of riders, and so on. This data along with alternate sources of information such as weather, traffic, terrain, season and so on.

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

The objective is to forecast bike rental demand of Bike sharing program in Washington, D.C based on historical usage patterns in relation with weather, environment and other data. We would be interested in predicting the rentals on various factors including season, temperature, weather and building a model that can successfully predict the number of rentals on relevant factors.

1.2 Data

This dataset contains the seasonal and weekly count of rental bikes between years 2011 and 2012 in Capital bikeshare system with the corresponding temperature and humidity information. Bike sharing systems are a new way of traditional bike rentals. The whole process from membership to rental and return back has become automatic. The data was generated by 500 bike-sharing programs and was collected by the Laboratory of Artificial Intelligence and Decision Support (LIAAD), University of Porto. Given below is the description of the data which is a (731, 16) shaped data.

Short description of features

1. instant: Record index
2. dteday: Date
3. season: Season (1:spring, 2:summer, 3:fall, 4:winter)
4. yr: Year (0: 2011, 1:2012)
5. mnth: Month (1 to 12)
6. holiday: weather day is holiday or not (extracted from Holiday Schedule)
7. weekday: Day of the week
8. workingday: If day is neither weekend nor holiday it's 1, otherwise is 0.
9. weathersit: (extracted from Freemeteo)
    a. Clear, Few clouds, Partly cloudy, Partly cloudy
    b. Mist + Cloudy, Mist + Broken clouds, Mist + Few clouds, Mist
    c. Snow, Light Rain + Thunderstorm + Scattered clouds, Light Rain + Scattered clouds
    d. Heavy Rain + Ice Pallets + Thunderstorm + Mist, Snow + Fog
10. temp: Normalized temperature in Celsius.
11. atemp: Normalized feeling temperature in Celsius.
12. hum: Normalized humidity. The values are divided to 100 (max)

13. windspeed: Normalized wind speed. The values are divided to 67 (max)
14. casual: count of casual users
15. registered: count of registered users
16. cnt: count of total rental bikes including both casual and registered

In this project, our task is to build regression models which will used to predict the bike rental count on daily basis. Given below is a sample of the bike sharing dataset:

Fig 1.1: Bike sharing Dataset

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

As you can see in the table below, we have the following 12 variables, using which we must correctly predict the bike rental count, column name has been renamed:

Fig 1.2: Predictor Variables

```
Index(['season', 'year', 'month', 'holiday', 'weekday', 'workingday',
       'weather_condition', 'temp', 'humidity', 'windspeed', 'total_count',
       'isweekend'],
      dtype='object')
```

We have created a new feature:

- isweekend – extracted weekend days from datetime feature.

We have dropped the following variables from our dataset:

1. Instant- it represents the index of a record
2. dteday – it represents date on the given day
3. atemp - Normalized feeling temperature strongly correlated with temp
4. Casual and Registered- they are leakage variables in nature (dependent) and need to drop during model building to avoid bias. (casual + registered = count)

Chapter 2
Methodology

2.1 Exploratory Data Analysis (EDA)

Exploratory data analysis (EDA) is a very important step which takes place after feature engineering and acquiring data and it should be done before any modeling. This is because it is very important for a data scientist to be able to understand the nature of the data without making assumptions. The results of data exploration can be extremely useful in grasping the structure of the data, the distribution of the values, and the presence of extreme values and interrelationships within the data set. It involves the loading dataset, target classes count, data cleaning, typecasting of attributes, missing value analysis, Attributes distributions and trends.

> Purpose of EDA:

1. Summarize the statistics and visualization of data for better understanding. Curbing indication for tendencies of the data, its quality and to formulate assumptions and the hypothesis of our analysis.

2. To create an overall picture of the data with basic statistical description and aspects, and identify

2.1.1 Descriptive Analysis

It is a summary statistic that quantitatively describes or summarizes features of a collection of information, process of condensing key characteristics of the data set into simple numeric metrics. Some of the common metrics used are mean, standard deviation, and correlation.

2.1.1.1 Feature Analysis

Generating profile report using pandas_profiling
For each column the following statistics are presented in an interactive HTML page:
* Essentials: type, unique values, missing values
* Quantile statistics: minimum value, Q1, median, Q3, maximum, range, interquartile range
* Descriptive statistics: mean, mode, standard deviation, sum, median absolute deviation, coefficient of variation, kurtosis, skewness
Most frequent values
* Histogram
* Correlations highlighting of highly correlated variables, Spearman and Pearson matrixes

Fig 2.1: Panda Profiling

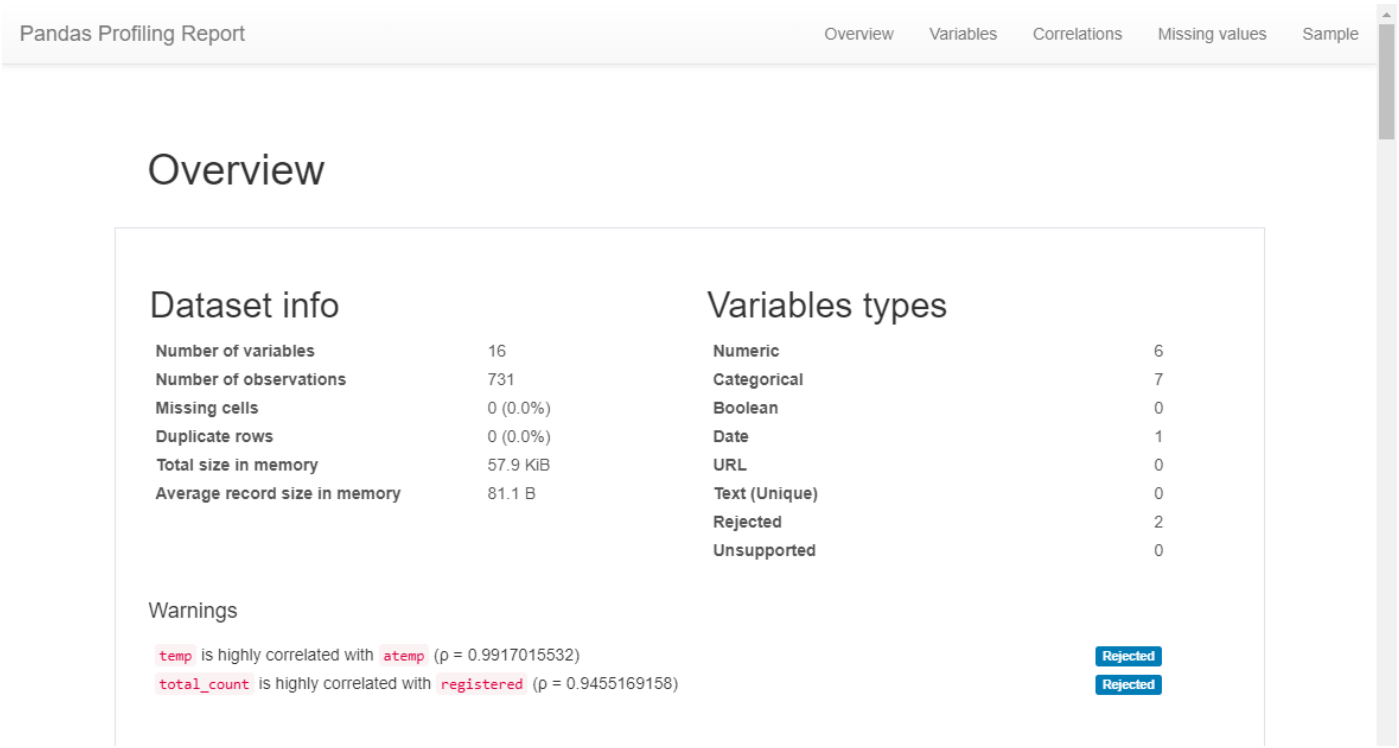


Fig 2.2: Train information

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 731 entries, 0 to 730
Data columns (total 16 columns):
instant              731 non-null int64
datetime             731 non-null datetime64[ns]
season               731 non-null category
year                 731 non-null category
month                731 non-null category
holiday              731 non-null category
weekday              731 non-null category
workingday           731 non-null category
weather_condition    731 non-null category
temp                 731 non-null float64
atemp                731 non-null float64
humidity             731 non-null float64
windspeed            731 non-null float64
casual               731 non-null int64
registered           731 non-null int64
total_count          731 non-null int64
dtypes: category(7), datetime64[ns](1), float64(4), int64(4)
memory usage: 57.9 KB
```

2.1.1.2 Missing value analysis

In this, we need to find out any missing values are present in dataset. We have not found any missing values in both train and test data.

Python and R code as follows:

1. Python

```
# missing value checking
print('Number of missing values:\n',bike_day.isnull().sum())

#  description
bike_day.describe()
```

2. R

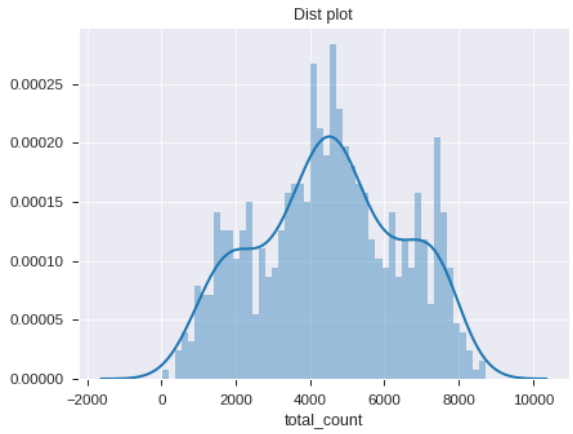
```
# Missing values
cat('Data missing values:',sum(sum(is.na(bike_day)))) # print number of missing values
missing_val<-data.frame(apply(bike_day,2,function(x){sum(is.na(x))}))
names(missing_val)[1]='missing_val'
missing_val
```

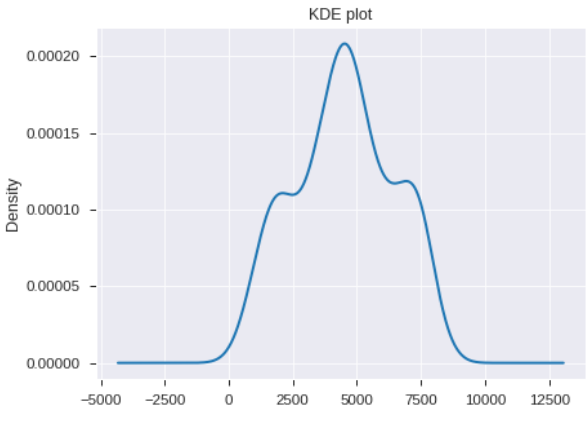
A data.frame: 16 × 1

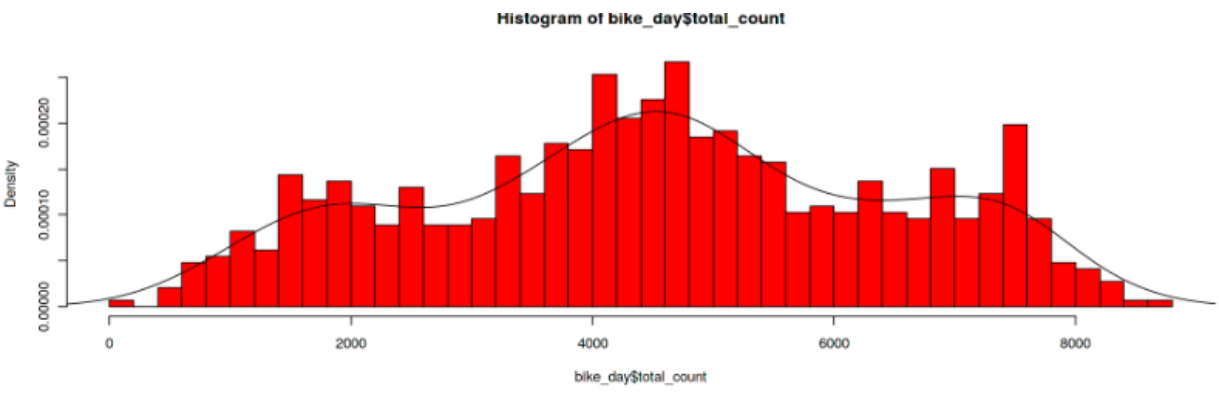
	missing_val
	<int>
instant	0
datetime	0
season	0
year	0
month	0
holiday	0
weekday	0
workingday	0
weather_condition	0
temp	0
atemp	0
humidity	0
windspeed	0
casual	0
registered	0
total_count	0

2.1.1.3 Target Value

Target variable (total count) distribution







2.1.2 Visualization

It is the process of projecting the data, or parts of it, into Cartesian space or into abstract images. With a little domain knowledge, data visualizations can be used to express and demonstrate key relationships in plots and charts that are more visceral to yourself and stakeholders than measures of association or significance. In the data mining process, data exploration is leveraged in many different steps including preprocessing, modeling, and interpretation of results.

One of our main goals for visualizing the data here, is to observe which features are most intuitive in predicting target. The other, is to draw general trend, may aid us in model selection and hyper parameter selection.

2.1.2.1 Attribute Distribution and Trends

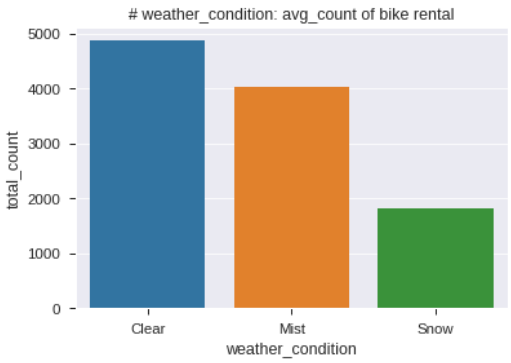
A. Categorical Features

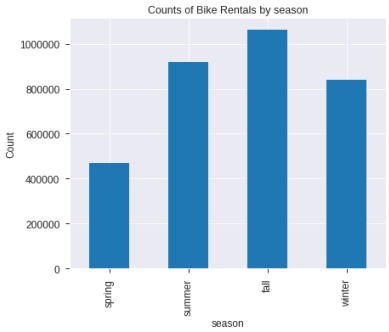
```
categorical_features = categorical_col
print('Categorical features :',', '.join(categorical_features))
```

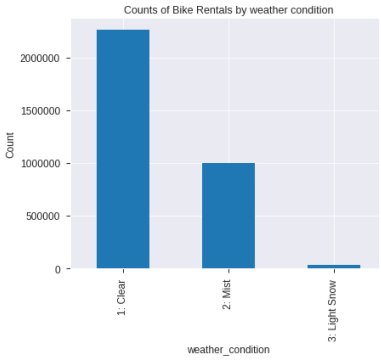
```
Categorical features : season, year, month, holiday, weekday, workingday, weather_condition
```

Key Findings:

1. People like to rent bikes more whenever the sky is clear.
2. the count of number of rented bikes is maximum in fall (Autumn) season and least in spring season.
3. number of bikes rented per season over the years has increased for both casual and registered users.
4. registered users have rented more bikes than casual users overall.
5. casual users travel more over weekends as compared to registered users (Saturday / Sunday).
6. registered users rent more bikes during working days as expected for commute to work / office.
7. demand for bikes are more on working days as compared to holidays (because majority of the bike users are registered)

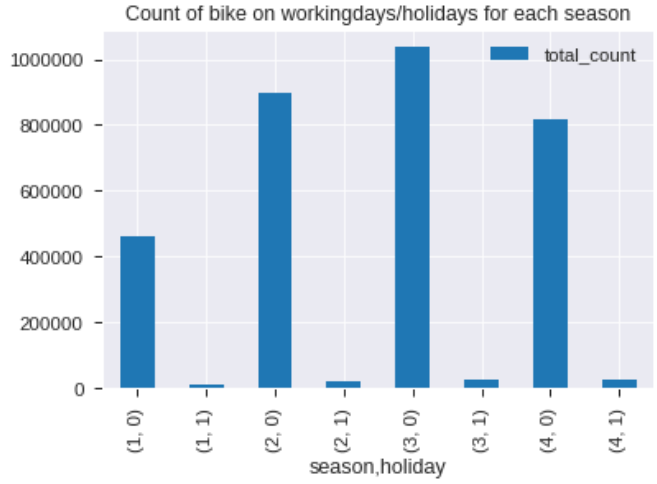






Observation:

1. People like to rent bikes more whenever the sky is clear.
2. the count of number of rented bikes is maximum in fall (Autumn) season and least in spring season.

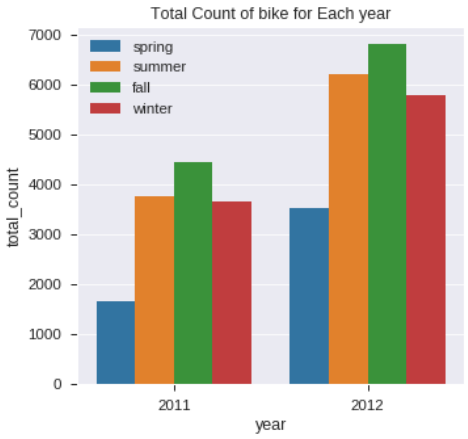


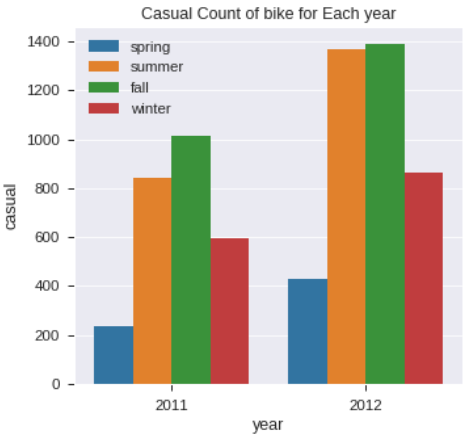
Observation:

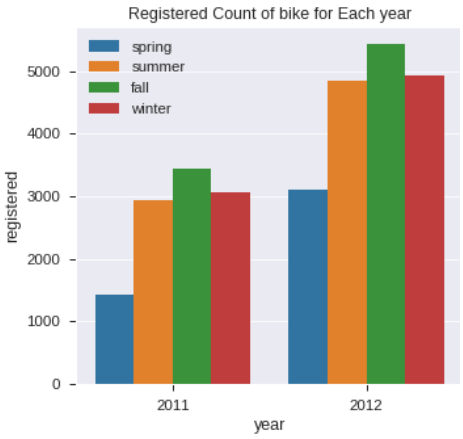
1. For all season, number of workdays rental is much higher than holidays.
2. Both workday and holiday is following same trend over the seasonal rental count and being highest in fall.

Observation:

1. Total, casual, registered rental count increased by next year.
2. All are following same trend over the seasonal rental count and being highest in fall.
3. Casual rental count is lesser than the registered rental counts.

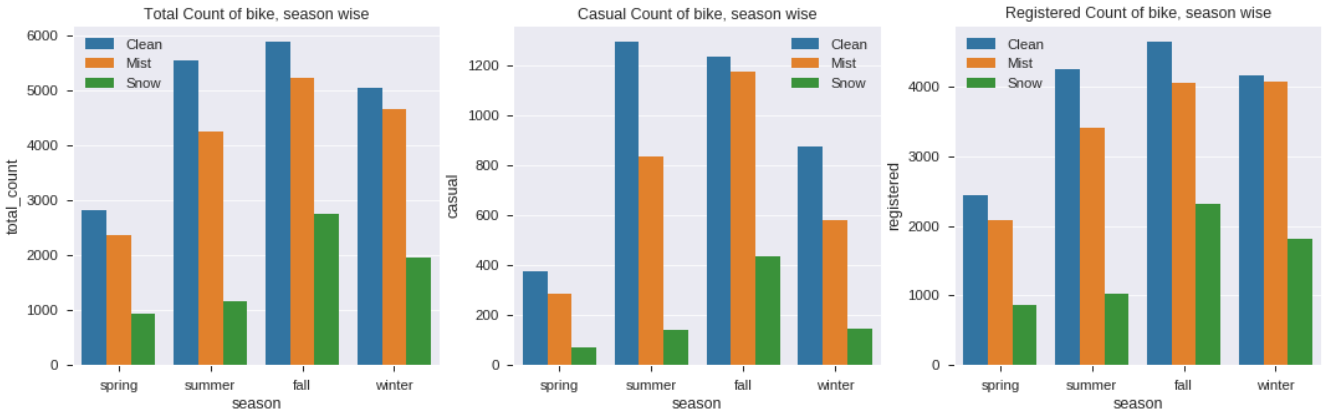






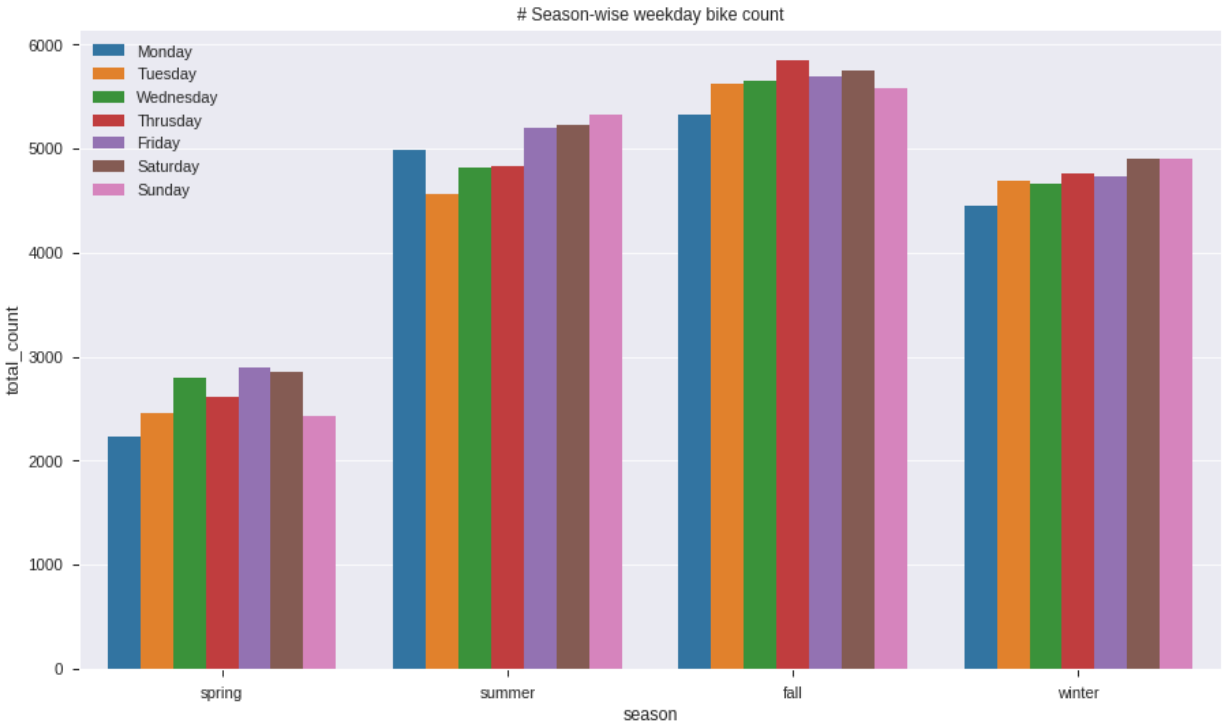
Observation:

1. For each season, people prefer clear weather for renting bike.
2. All are following same trend over the seasonal rental count and being highest in fall on clear weather days.



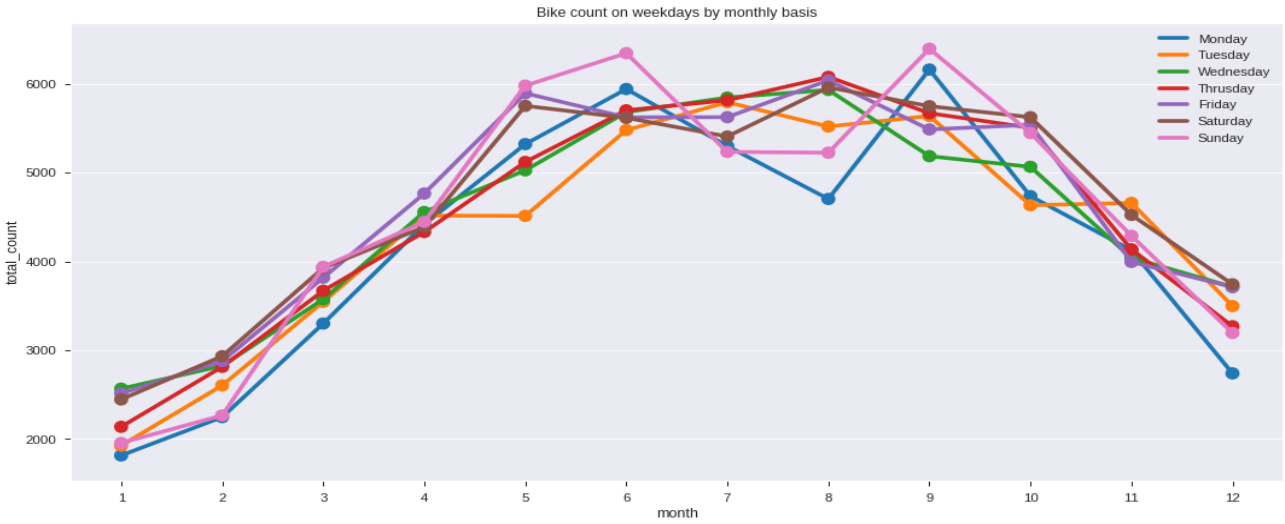
Observation:

1. Count of bike rental on Friday, Saturday & Sunday is always higher than Mondays.



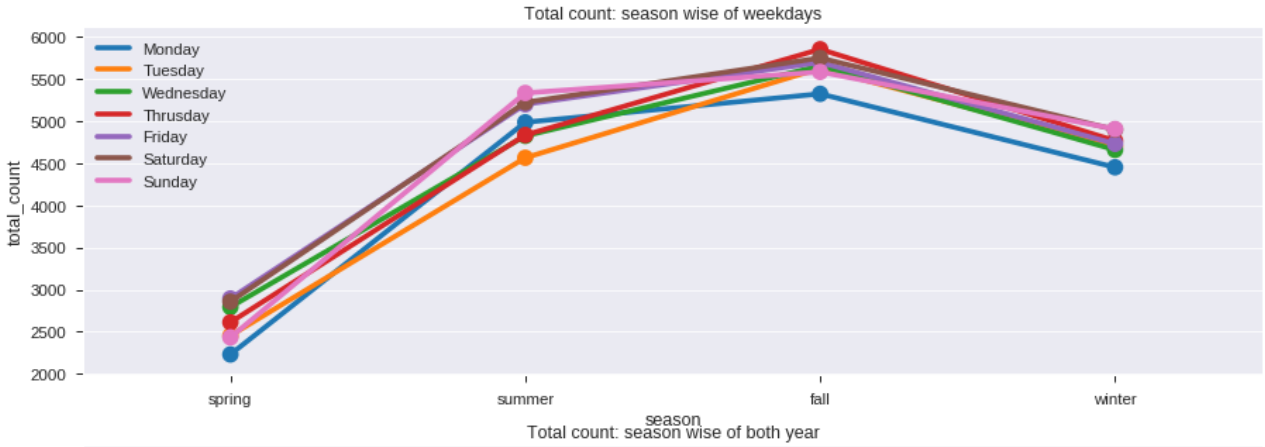
Observation:

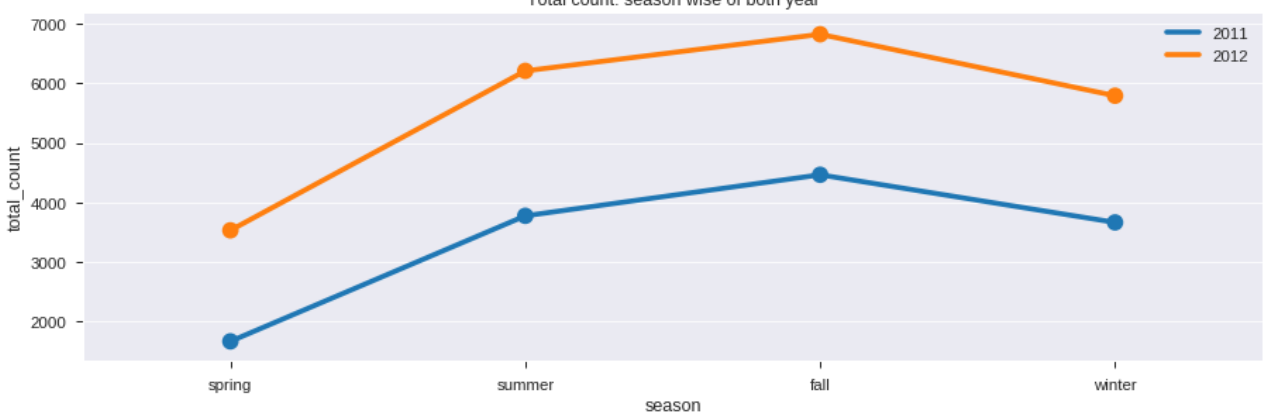
1. Between May (5) to October (10) renting is high for all over the week.
2. Monday being the most fluctuating trend.



Observation:

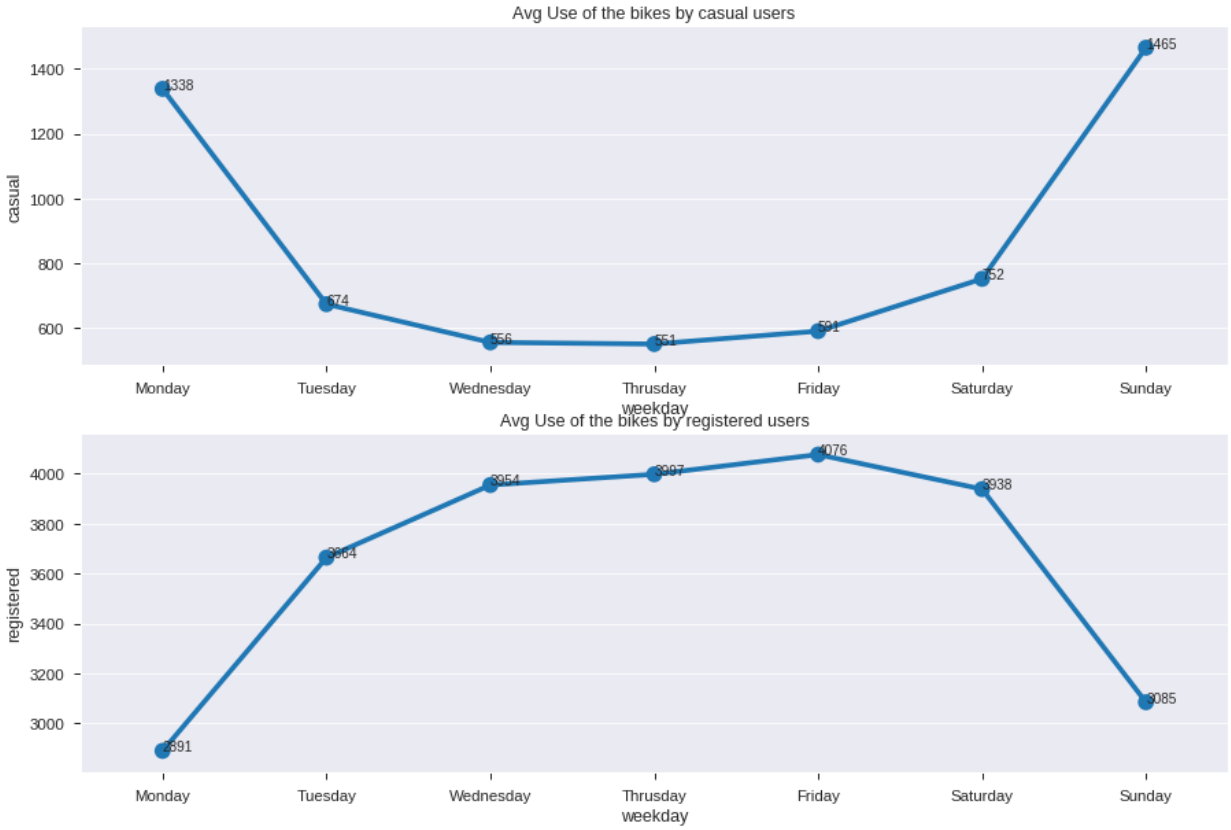
- Overall renting has increased with next year.
- People hardly prefer Monday for renting a bike in all season except summer.





Observation:

- Casual count increases over the weekend, on an average.
- Registered count increases over the weekday on an average.



B. Continuous Features

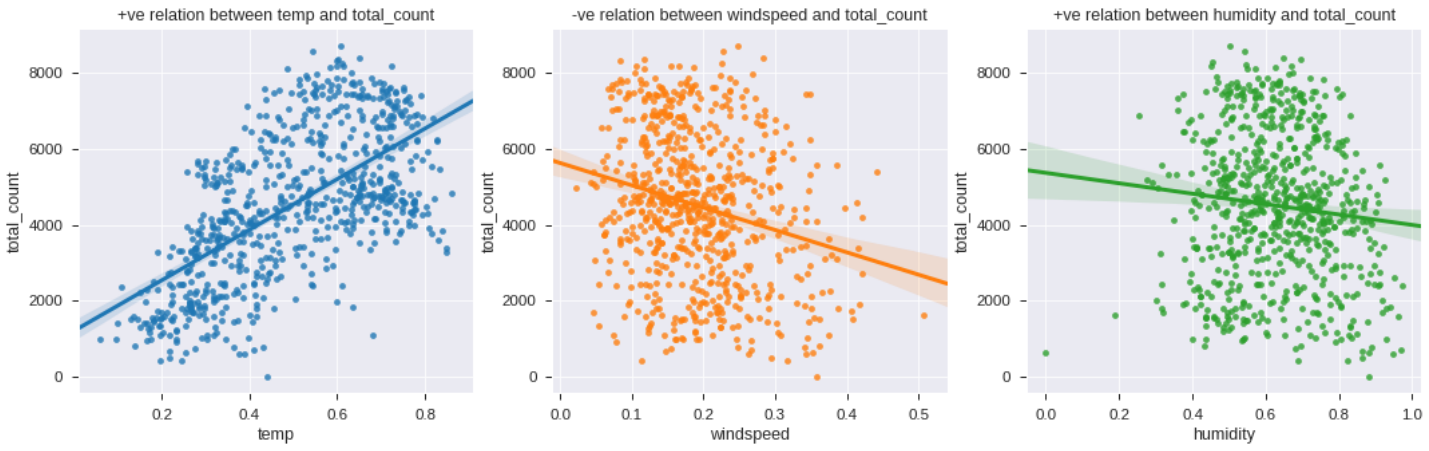
	instant	temp	atemp	humidity	windspeed	casual	registered	total_count
count	731.000000	731.000000	731.000000	731.000000	731.000000	731.000000	731.000000	731.000000
mean	366.000000	0.495385	0.474354	0.627894	0.190486	848.176471	3656.172367	4504.348837
std	211.165812	0.183051	0.162961	0.142429	0.077498	686.622488	1560.256377	1937.211452
min	1.000000	0.059130	0.079070	0.000000	0.022392	2.000000	20.000000	22.000000
25%	183.500000	0.337083	0.337842	0.520000	0.134950	315.500000	2497.000000	3152.000000
50%	366.000000	0.498333	0.486733	0.626667	0.180975	713.000000	3662.000000	4548.000000
75%	548.500000	0.655417	0.608602	0.730209	0.233214	1096.000000	4776.500000	5956.000000
max	731.000000	0.861667	0.840896	0.972500	0.507463	3410.000000	6946.000000	8714.000000

Key Findings:

1. Temp, Atemp looks normally distributed.
2. A strong correlation can be seen for temp and atemp.
3. windspeed, humidity, temp and atemp are all normalized in the dataset already.
4. With increase in temperature, the count of bike rentals increases as shown in reg plot.
5. temp has got positive correlation with count as people like to travel more when the sky is clear.
6. humidity is inversely related to count as expected as when weather is humid people will not like to travel on a bike.
7. windspeed is also having a negative correlation with "count".
8. "atemp" and "temp" variable has got strong correlation with each other. During model building any one of the variables must be dropped since they will exhibit multicollinearity in the data.
9. "weather condition" and count are inversely related. This is because for our data as weather increases from (1 to 4) implies that weather is getting more worse and so lesser people will rent bikes.
10. "registered" and count are highly related which indicates that most of the bikes that are rented are registered.
11. "Casual" and "Registered" are also not considered since they are leakage variables in nature and need to drop during model building to avoid bias. (casual + registered = count).
12. "instant" variable can also be dropped during model building as it indicates index.

Observation: Here we considered "count" vs "temp", "humidity", "windspeed".

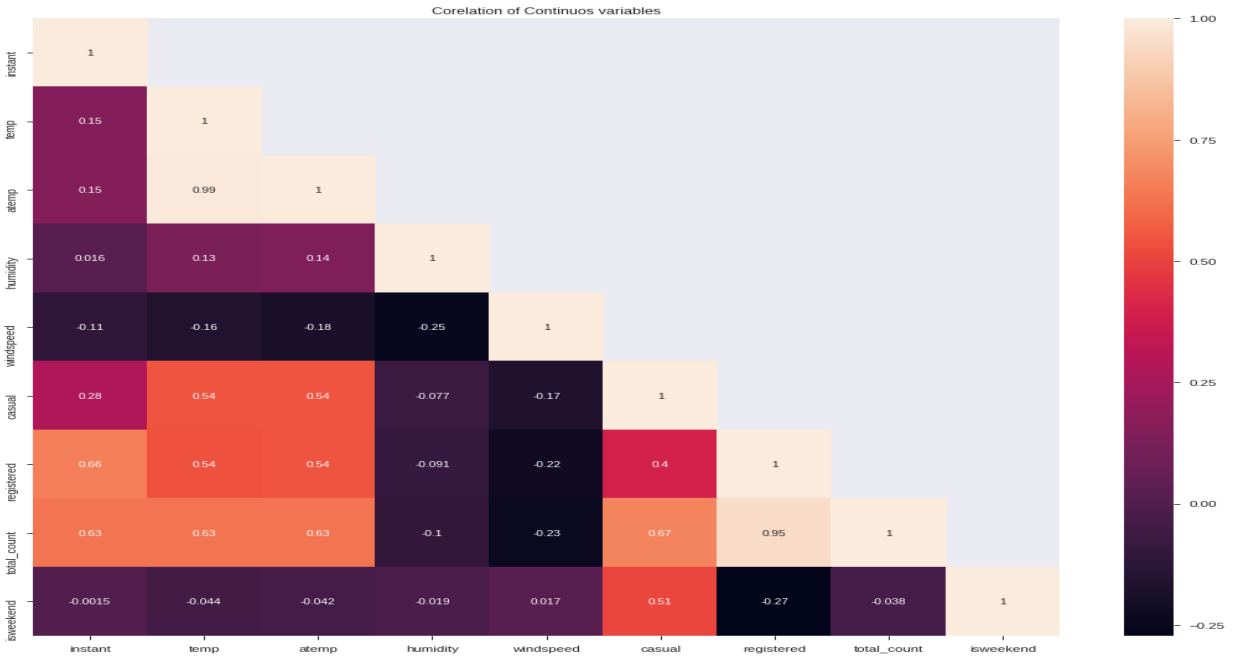
1. A +ve correlation with temperature was observed (sky is clear with increase in temperature)
2. A -ve correlation with humidity and windspeed was observed as people avoid travelling when weather is very windy or humid.



Correlation matrix is telling about linear relationship between attributes and help us to build better models. From the correlation plot, we can observe that some features are positively correlated, and some are negatively correlated to each other. The temp and atemp are highly positively correlated to each other, it means that both are carrying same information. So, we are going to ignore atemp, casual and registered variable for further analysis.

Observation:

- Temp and atemp are highly correlated.
- Casual + registered = Total count

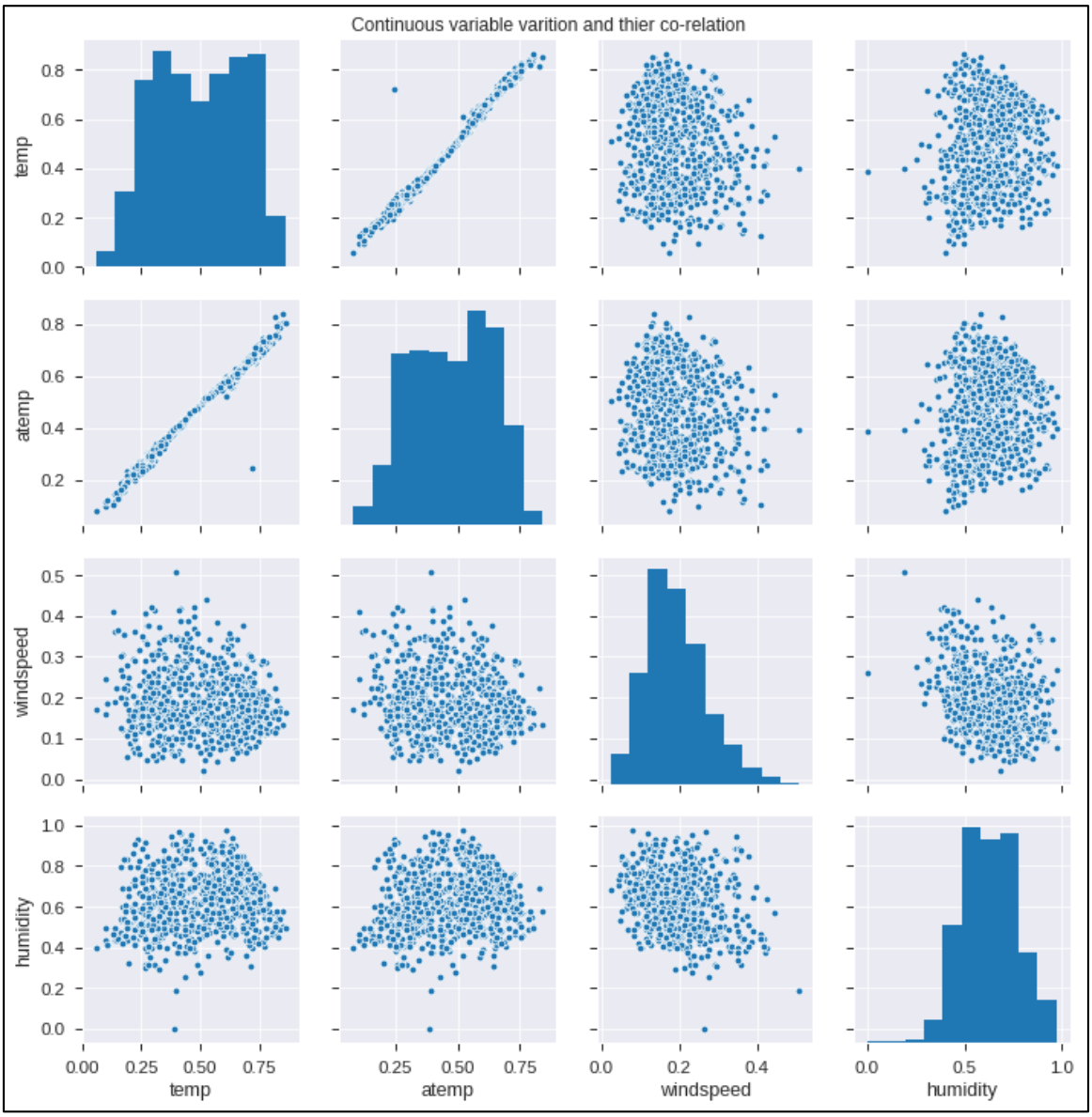


Observation:

- A +ve correlation between humidity and temperature was observed (as temp increases the amount of water vapour present in the air also increases)
- A -ve correlation between windspeed with humidity and temperature was observed (as wind increases, it draws heat from the body, thereby temperature and humidity decreases)

Observation:

- Temp and atemp are highly correlated.
- Windspeed and humidity have skewed distribution.
- One point seems to be outlier in temp and atemp correlation plot.
- Possibility of outliers in Windspeed and humidity.

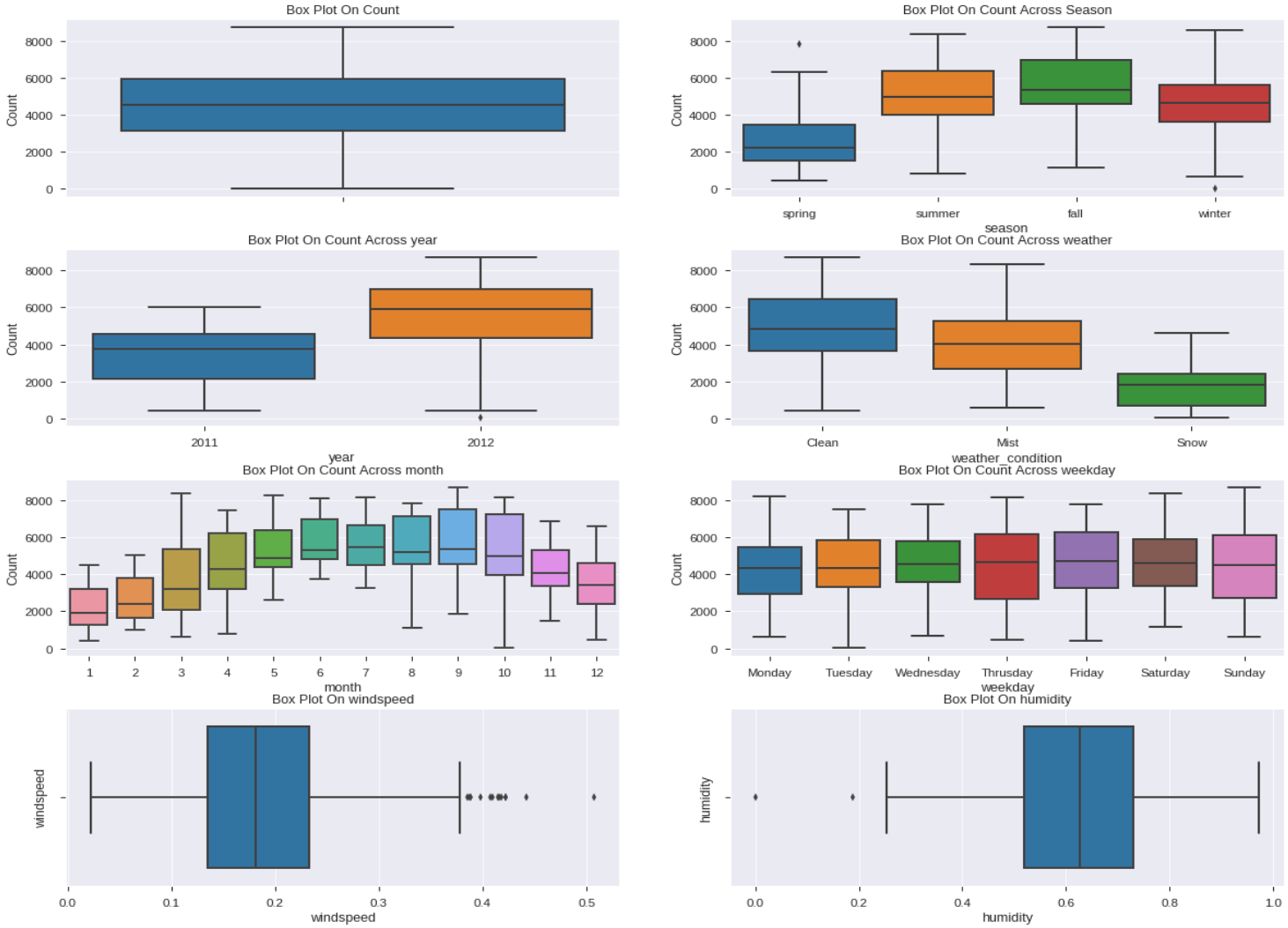


2.1.2.2 Outlier Analysis

A boxplot is a graph that gives you a good indication of how the values in the data are spread out. Although boxplots may seem primitive in comparison to a histogram or density plot, they have the advantage of taking up less space, which is useful when comparing distributions between many groups or datasets.

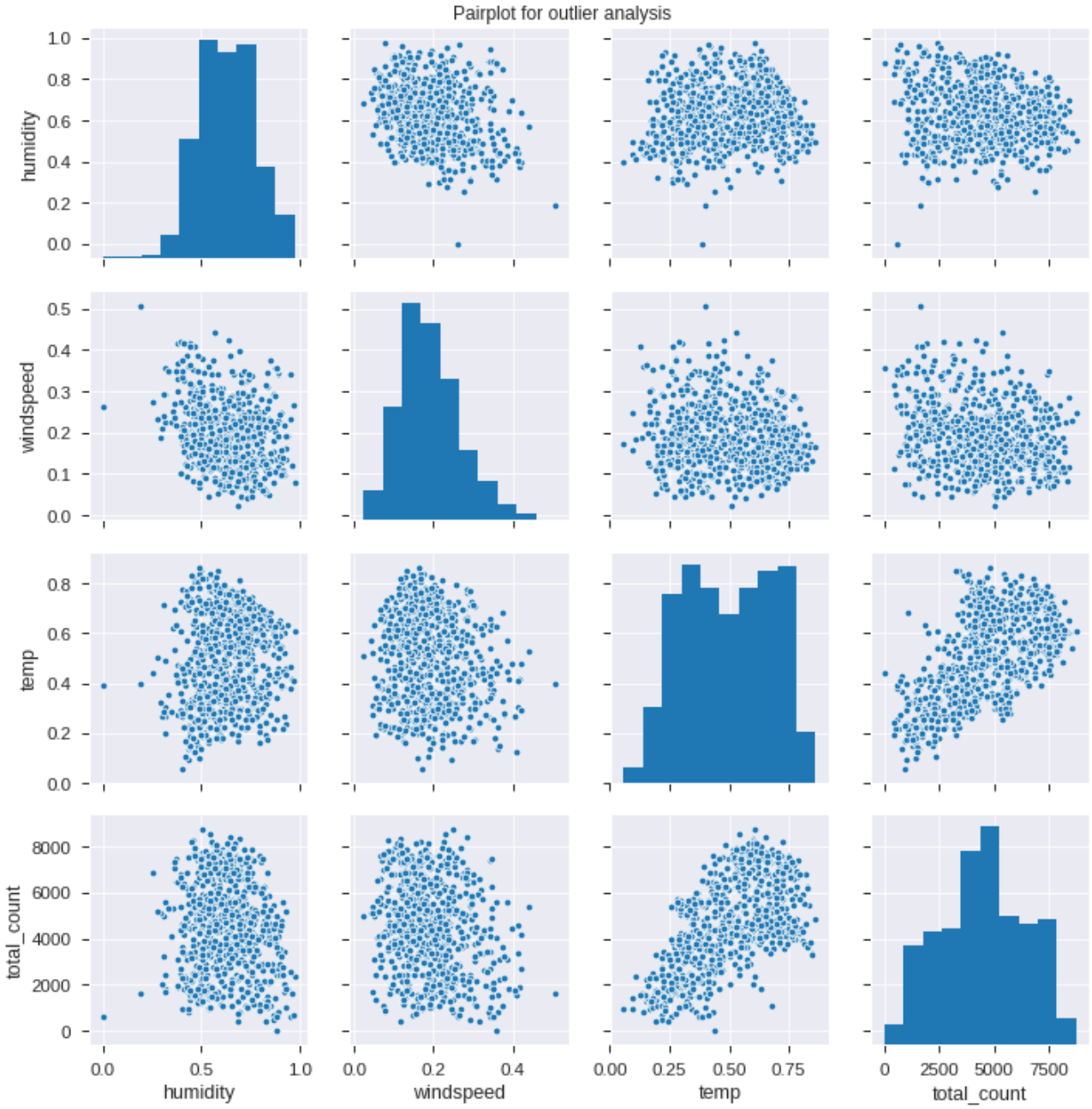
Key Findings:

- Few bivariate outliers were observed from the box plots and pair plots.



Observations:

1. From the box plot, we can observe that few outliers are present in normalized windspeed and humidity variable.
2. Data need to be free from outliers and need to be scaled before applying any outlier sensitive model algorithms.



Observations:

1. From the pair plot, we can observe that no outliers are present in normalized temp, but few outliers are present in normalized windspeed and humidity variable.
2. Data need to be free from outliers and need to be scaled before applying any outlier sensitive model algorithms.

2.2 Data Preprocessing and Analysis

2.2.1 Outlier Handling

```
# finding outliers
wind_humidity = pd.DataFrame(bike_day, columns=['windspeed', 'humidity'])

# get outliers for windspeed and humidity features
for i in ['windspeed', 'humidity']:
    q75, q25 = np.percentile(wind_humidity.loc[:,i], [75,25]) # get q75 and q25
    IQR = q75 - q25 # calculate  IQR for boxplot outlier method
    max = q75+(IQR*1.5) # get max bound
    min = q25-(IQR*1.5) # get min bound
    wind_humidity.loc[wind_humidity.loc[:,i]<min,:i] = np.nan # replacing outliers with NAN
    wind_humidity.loc[wind_humidity.loc[:,i]>max,:i] = np.nan # replacing outliers with NAN

print('Shape after dropping outlier (windspeed,humidity):',wind_humidity.dropna().shape)
print('Shape before dropping outlier (windspeed,humidity):',bike_day[['windspeed','humidity']].shape)
```

```
Shape after dropping outlier (windspeed,humidity): (717, 2)
Shape before dropping outlier (windspeed,humidity): (731, 2)
```

	windspeed	humidity	outlier
44	NaN	0.375833	1
49	NaN	NaN	1
68	NaN	NaN	1
93	NaN	0.426250	1
94	NaN	0.642083	1
292	NaN	0.636250	1
382	NaN	0.443333	1
407	NaN	0.464583	1
420	NaN	0.395833	1
432	NaN	0.567500	1
433	NaN	0.407083	1
450	NaN	0.477917	1
666	NaN	0.694583	1
721	NaN	0.441250	1

Observation:

1. After separating outliers and inliers with IQR method we found 25 rows have outliers.
2. Since, 25 is not very significant, we will drop those 25 rows.

2.2.2 Feature Selection

Feature selection is very important for modelling the dataset. Every dataset has good and unwanted features. The unwanted features will affect performance of model, so we must delete those features. We must select best features by using ANOVA, Chi-Square test and correlation matrix statistical techniques and so on. In this, we are selecting best features by using Correlation matrix.

1. "atemp" and "temp" variable has got strong correlation with each other. During model building any one of the variables must be dropped since they will exhibit multicollinearity in the data.
2. “datetime” variable only contains date which is not important features for prediction.
3. "weather condition" and count are inversely related. This is because for our data as weather increases from (1 to 4) implies that weather is getting more worse and so lesser people will rent bikes.
4. "registered" and count are highly related which indicates that most of the bikes that are rented are registered.
5. "Casual" and "Registered" are also not considered since they are leakage variables in nature and need to drop during model building to avoid bias. (casual + registered = count).
6. "instant" variable can also be dropped during model building as it indicates index.

Python:

```
# categorising features
categorical_features = ["season","holiday","weather_condition","weekday","month","year",'isweekend','workingday']
continous_features = ["temp","humidity","windspeed"]
dropFeatures = ['casual',"datetime","instant","registered","atemp","outlier"]
target=['total_count']
```

```
# drop unwanted features
bike_FE = bike_day.drop(dropFeatures, axis=1)
bike_FE.columns
```

```
Index(['season', 'year', 'month', 'holiday', 'weekday', 'workingday',
       'weather_condition', 'temp', 'humidity', 'windspeed', 'total_count',
       'isweekend'],
      dtype='object')
```

2.2.3 Feature Engineering

1. A new feature "isweekend" was created, which denotes Sat or Sun as 1, and 0 otherwise.

Python:

```
# Generally, 1-Monday and 0-Sunday in datetime , for weekend: Saturday-6 & Sunday-0
# creating new feature 'isweekend' using datetime feature and computing is that date is fall over the weekend
bike_day['isweekend']=bike_day['datetime'].apply( lambda x :1 if (x.weekday()==0) |(x.weekday()==6) else 0 ) # for weekday = 0 or 6, isweekend = 1 el
bike_day[bike_day['isweekend']==1]['weekday'].value_counts() # number of weekend days
```

R:

```
# create new feature 'isweekend' : finding weekend days
# 0: sunday
# 6: saturday
weekend = as.data.frame(lapply(bike_day$weekday,function(x) if (x==0 |x==6) {1} else {0} ), byr
ow=T, 'isweekend' )
bike_day$isweekend = as.factor(t(weekend)) # convert as factor for categorical
```

```
# Count weekends
table(bike_day$weekday==6 ) # Saturday counts
table(bike_day$weekday==0 ) # Sunday counts
```

```
FALSE  TRUE
  626   105

FALSE  TRUE
  626   105
```

2. We will use pd.get_dummies () function for One-Hot Encoding the categorical features and fastDummies function in R, below is the Python and R code:

Python:

```
# create dummy data
dummy_data = bike_FE.copy()

# fucntion for creating dummy features
def get_dummy(df, col):
    df = pd.concat([df, pd.get_dummies(df[col], prefix=col, drop_first=True)], axis=1) # create dummy features and dropping first feature, since it's
    df = df.drop([col], axis = 1)                                                      # drop feature of which dummy is created
    return df                                                                          # return dummy dataframe

# features to create dummy
# get_dummy_features = ["season","weather_condition","weekday","month"]
get_dummy_features = categorical_features

# create dummy for features
for col in get_dummy_features:
    dummy_data = get_dummy(dummy_data, col) # create dummy for all categorical features

dummy_data.head()
```

R:

```
# features to create dummy
get_dummy_features = categorical_features

# Using fastDummies function to create dummy features
dummy_data = fastDummies::dummy_cols(bike_day, select_columns=get_dummy_features, remove_first_
dummy = TRUE) # create dummy, also drop the first feature of each dummy variable
dummy_data = subset(dummy_data, select = -c(season,holiday,weather_condition,weekday,month,yea
r,isweekend,workingday)) # drop original categorical features
str(dummy_data) # print new data information
```

Python Output

```
Data columns (total 30 columns):
temp                    717 non-null float64
humidity                717 non-null float64
windspeed               717 non-null float64
total_count             717 non-null int64
season_2                717 non-null uint8
season_3                717 non-null uint8
season_4                717 non-null uint8
holiday_1               717 non-null uint8
weather_condition_2     717 non-null uint8
weather_condition_3     717 non-null uint8
weekday_1               717 non-null uint8
weekday_2               717 non-null uint8
weekday_3               717 non-null uint8
weekday_4               717 non-null uint8
weekday_5               717 non-null uint8
weekday_6               717 non-null uint8
month_2                 717 non-null uint8
month_3                 717 non-null uint8
month_4                 717 non-null uint8
month_5                 717 non-null uint8
month_6                 717 non-null uint8
month_7                 717 non-null uint8
month_8                 717 non-null uint8
month_9                 717 non-null uint8
month_10                717 non-null uint8
month_11                717 non-null uint8
month_12                717 non-null uint8
year_1                  717 non-null uint8
isweekend_1             717 non-null uint8
workingday_1            717 non-null uint8
dtypes: float64(3), int64(1), uint8(26)
```

R Output

```
'data.frame':   717 obs. of  30 variables:
 $ temp               : num  0.344 0.363 0.196 0.2 0.227 ...
 $ total_count        : int  985 801 1349 1562 1600 1606 1510 959
 $ windspeed          : num  0.16 0.249 0.248 0.16 0.187 ...
 $ humidity           : num  0.806 0.696 0.437 0.59 0.437 ...
 $ season_2           : int  0 0 0 0 0 0 0 0 0 0 ...
 $ season_3           : int  0 0 0 0 0 0 0 0 0 0 ...
 $ season_4           : int  0 0 0 0 0 0 0 0 0 0 ...
 $ holiday_1          : int  0 0 0 0 0 0 0 0 0 0 ...
 $ weather_condition_1: int  0 0 1 1 1 1 0 0 1 1 ...
 $ weather_condition_3: int  0 0 0 0 0 0 0 0 0 0 ...
 $ weekday_0          : int  0 1 0 0 0 0 0 0 1 0 ...
 $ weekday_1          : int  0 0 1 0 0 0 0 0 0 1 ...
 $ weekday_2          : int  0 0 0 1 0 0 0 0 0 0 ...
 $ weekday_3          : int  0 0 0 0 1 0 0 0 0 0 ...
 $ weekday_4          : int  0 0 0 0 0 1 0 0 0 0 ...
 $ weekday_5          : int  0 0 0 0 0 0 1 0 0 0 ...
 $ month_2            : int  0 0 0 0 0 0 0 0 0 0 ...
 $ month_3            : int  0 0 0 0 0 0 0 0 0 0 ...
 $ month_4            : int  0 0 0 0 0 0 0 0 0 0 ...
 $ month_5            : int  0 0 0 0 0 0 0 0 0 0 ...
 $ month_6            : int  0 0 0 0 0 0 0 0 0 0 ...
 $ month_7            : int  0 0 0 0 0 0 0 0 0 0 ...
 $ month_8            : int  0 0 0 0 0 0 0 0 0 0 ...
 $ month_9            : int  0 0 0 0 0 0 0 0 0 0 ...
 $ month_10           : int  0 0 0 0 0 0 0 0 0 0 ...
 $ month_11           : int  0 0 0 0 0 0 0 0 0 0 ...
 $ month_12           : int  0 0 0 0 0 0 0 0 0 0 ...
 $ year_1             : int  0 0 0 0 0 0 0 0 0 0 ...
 $ isweekend_0        : int  0 0 1 1 1 1 1 0 0 1 ...
 $ workingday_1       : int  0 0 1 1 1 1 1 0 0 1 ...
```

2.3 Modeling

2.3.1 Model Selection

Model selection is the process of choosing between different machine learning approaches - e.g. SVM, logistic regression, linear regression, etc. - or choosing between different hyperparameters or sets of features for the same machine learning approach - e.g. deciding between the polynomial degrees/complexities for linear regression.

The dependent variable can fall in either of the four categories:

Nominal, Ordinal, Interval, Ratio

If the dependent variable is Nominal the only predictive analysis that we can perform is Classification, and if the dependent variable is Interval or Ratio, the normal method is to do a Regression analysis, or classification after binning.

We started our model building from the simplest to more complex. Therefore, we use Simple Linear Regression first as our base model.

We used the following models for the evaluation of the right algorithm:

1. Simple Linear Regression
2. Regularization Regression: LASSO, Ridge, and ElasticNet
3. Random Forest Regression
4. Gradient Boosting Machine

2.3.2 Multilinear & Regularization model: Linear Regression| Ridge| Lasso| ElasticNet

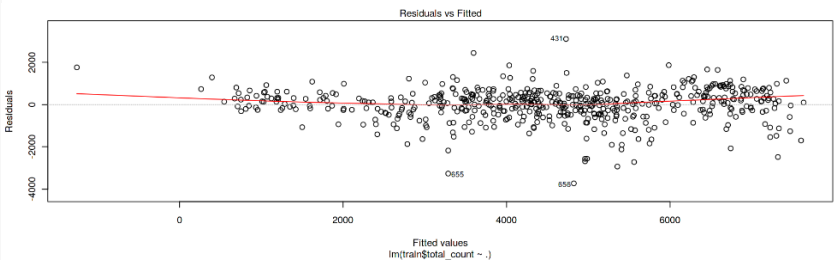
A. Without StandardScale

Python:

```
# liner models
linear_models = {'LinearRegression':LinearRegression(), 'Ridge':Ridge(), 'Lasso':Lasso(), 'ElasticNet':ElasticNet()}

# Score without StandardScale
print('Score without StandardScale:\n')
for regressor_name, regressor in linear_models.items():
    regressor.fit(X_train, y_train) # fit data to models
    y_pred = regressor.predict(X_test) # predict using models

    metrics(regressor_name, regressor, y_pred)
    print("\n")
```

R:

```
# simple linear model

# Set seed to reproduce the results of random sampling
set.seed(42)

# training the lr_model
lr_model = lm(train$total_count~.,data = train)

# Check the summary of the model
summary(lr_model)

# Predict the test cases
lr_predictions = predict(lr_model, test[,-2])
```



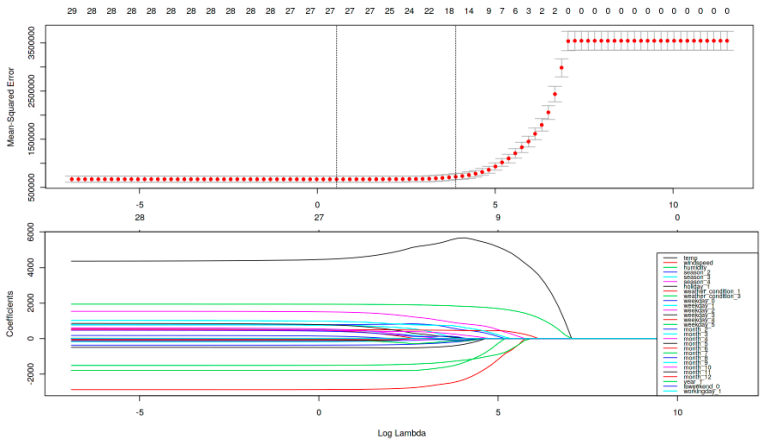
❖ Regularization LASSO, Ridge, ElasticNet with cross validation and hyper-tuning parameter.

Regularization: This is a form of regression, that constrains/ regularizes or shrinks the coefficient estimates towards zero. In other words, *this technique discourages learning a more complex or flexible model, to avoid the risk of overfitting.*

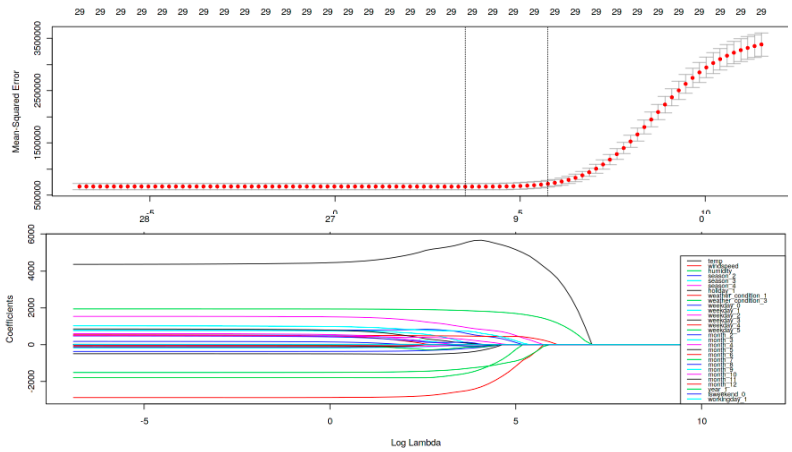
1. **Ridge Regression:**
   - Performs L2 regularization, i.e. adds penalty equivalent to **square of the magnitude** of coefficients
   - Minimization objective = LS Obj + α * (sum of square of coefficients)
2. **Lasso Regression:**
   - Performs L1 regularization, i.e. adds penalty equivalent to **absolute value of the magnitude** of coefficients
   - Minimization objective = LS Obj + α * (sum of absolute value of coefficients)

Note that here 'LS Obj' refers to 'least squares objective', i.e. the linear regression objective without regularization.

```
# lasso model

# Set seed to reproduce the results
set.seed(14)

# Perform 10-fold cross-validation to select lambda ---------------------------
lambda_seq = 10^seq(-3, 5, length.out = 100)

# Setting alpha = 1 implements lasso regression
cv_output = cv.glmnet(as.matrix(train[,-c(2)]), as.matrix(train[,2]), alpha=1, lambda = lambda_seq)

# plot cross-validation result
plot(cv_output)

# print best lambda value
cat('Best lambda: ',cv_output$lambda.min)

# fit model with multiple lambda
lasso_model = glmnet(as.matrix(train[,-c(2)]), as.matrix(train[,2]), alpha=1, lambda = lambda_seq )

# plot model to visualize effect of lambda
plot(lasso_model, xvar = "lambda")
legend("bottomright", lwd = 1, col = 1:6, legend = colnames(train[,-2]), cex = .7)

# Best lambda:  1.707353
```




```
# ridge model

# Set seed to reproduce the results
set.seed(147)

# Perform 10-fold cross-validation to select lambda ---------------------------
#lambda_seq = c( 0.1, 1, 2, 3, 4, 5, 10, 30, 50, 80, 100)
lambda_seq = 10^seq(-3, 5, length.out = 100)

# Setting alpha = 0 implements ridge regression
cv_output = cv.glmnet(as.matrix(train[,-c(2)]), as.matrix(train[,2]), alpha=0, lambda = lambda_seq)

# plot cross-validation result
plot(cv_output)

# print best lambda value
cat('Best lambda: ',cv_output$lambda.min)

# fit model with multiple lambda
ridge_model = glmnet(as.matrix(train[,-c(2)]), as.matrix(train[,2]), alpha=1, lambda = lambda_seq )

# plot model to visualize effect of lambda
plot(ridge_model, xvar = "lambda")
legend("bottomright", lwd = 1, col = 1:6, legend = colnames(train[,-2]), cex = .7)

# Best lambda:  33.51603
```

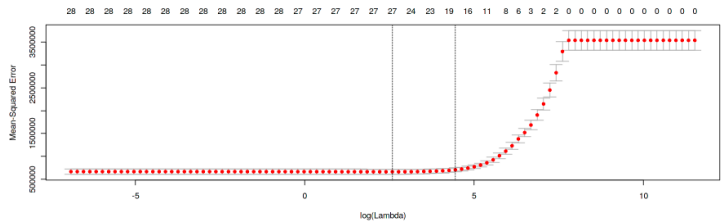


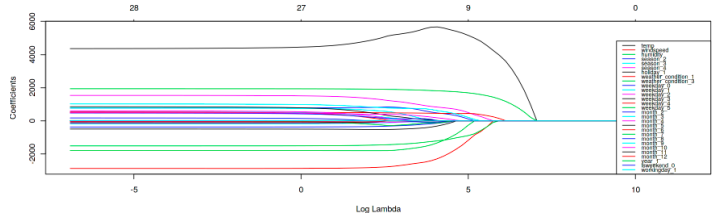
3. **ElasticNet Regression:**
   - ElasticNet is hybrid of Lasso and Ridge Regression techniques. It is trained with L1 and L2 prior as regularizes.
   - Elastic-net is useful when there are multiple features which are correlated. Lasso is likely to pick one of these at random, while elastic-net is likely to pick both.

$$\hat{\beta} = \underset{\beta}{\operatorname{argmin}}(\|y - X\beta\|^2 + \lambda_2\|\beta\|^2 + \lambda_1\|\beta\|_1).$$

- A practical advantage of trading-off between Lasso and Ridge is that, it allows Elastic-Net to inherit some of Ridge's stability under rotation.

```
# elastic net model

# Set seed to reproduce the results
set.seed(17)

# Perform 10-fold cross-validation to select lambda ---------------------------
lambda_seq = 10^seq(-3, 5, length.out = 100)

# Setting alpha = 0.5 implements elastic net regression
cv_output = cv.glmnet(as.matrix(train[,-c(2)]), as.matrix(train[,2]), alpha=0.5, lambda = lambda_seq)

# plot cross-validation result
plot(cv_output)

# print best lambda value
cat('Best lambda: ',cv_output$lambda.min)

# fit model with multiple lambda
elastic_model = glmnet(as.matrix(train[,-c(2)]), as.matrix(train[,2]), alpha=1, lambda = lambda_seq )

# plot model to visualize effect of lambda
plot(elastic_model, xvar = "lambda")
legend("bottomright", lwd = 1, col = 1:6, legend = colnames(train[,-2]), cex = .7)


# Best lambda:  13.21941
```





B. With StandardScale: Python

Python:

```
# Score with StandardScale
print('Score with StandardScale:\n')
simpleRegressor(linear_models)
```

```
# function: simple regressor
def simpleRegressor(model):
    """Simple Regressor function for each model given in model variable by creating pipeline with StandardScale() and fit data to predict.
        model: dict()

    """
    for regressor_name, regressor in linear_models.items():
        pipeline = Pipeline( [('scaler', StandardScaler()), ('model',regressor)] ) # create pipeline of each model with StandardScale
        regressor = pipeline.fit(X_train, y_train)                                  # fit data to the pipeline
        y_pred = regressor.predict(X_test)                                          # Predict using the pipeline model

        metrics(regressor_name, regressor, y_pred)                                  # get metrics : r2, adj r2, rmse, rmsle
        print("\n")
```

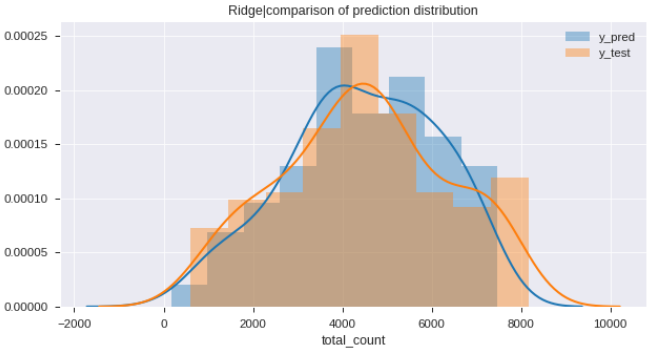
- ❖ Regularization LASSO, Ridge, ElasticNet with cross validation and hyper-tuning parameter.

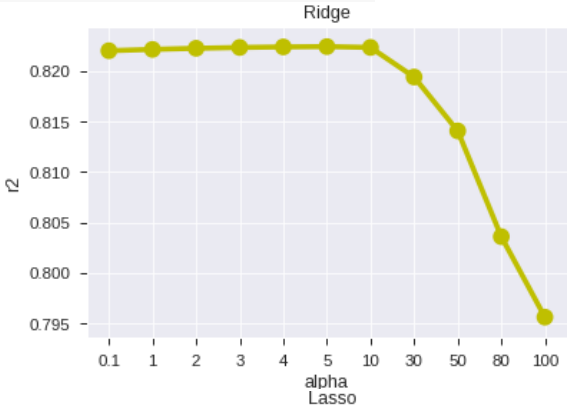
We have taken the regularization models and fitted a pipeline with StandardScale(), and applied gridSearchCV(), provided with hyper-parameters, for getting optimal model.

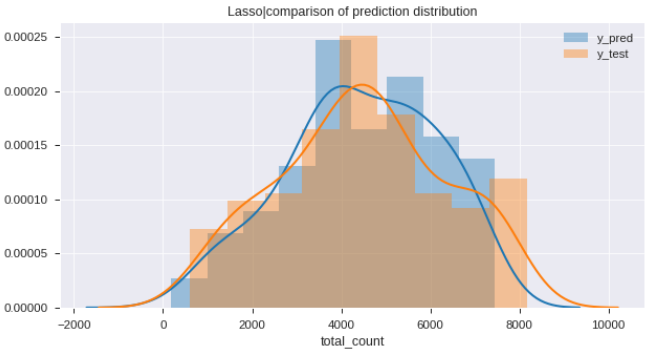
Grid search is the process of performing hyper parameter tuning in order to determine the optimal values for a given model. This is significant as the performance of the entire model is based on the hyper parameter values specified.

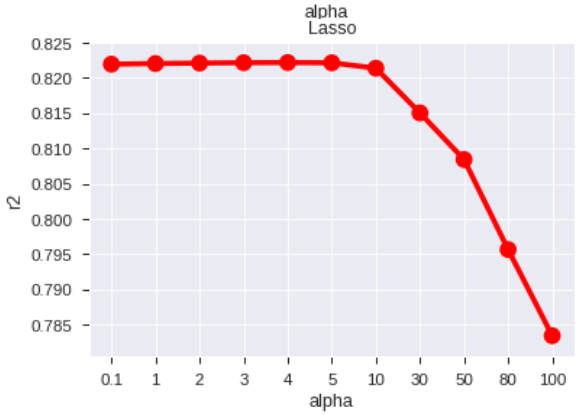
```
# linear regularization model with hypertuning parameters
regularization_linear_models = { 'Ridge':Ridge(), 'Lasso':Lasso(), 'ElasticNet':ElasticNet()}

# Regularization hyper-parameter tunning with GridSearchCV
print('Score with GridSearchCV:\n')
param_grid1 = {'model__alpha' : [0.1, 1, 2, 3, 4, 5, 10, 30, 50, 80, 100],'model__max_iter':[3000] } # parameters for feeding in gridSearch
gridSearchRegressor(regularization_linear_models, param_grid=param_grid1, plot_score_=True, compare_score=True )
```

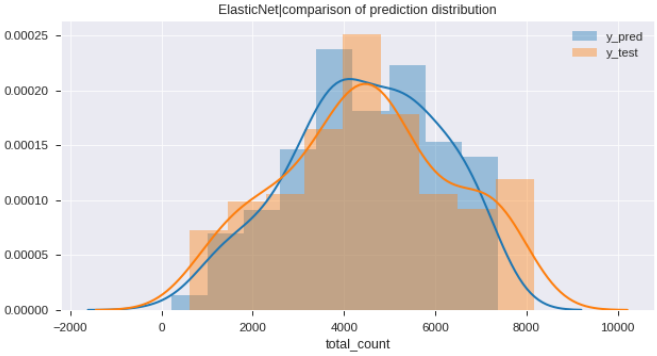
```python
# function: ensemble regressor with hyper-parameter tunning
def gridSearchRegressor(model, param_grid, scoring = 'r2', plot_score_ = False, compare_score = False):
    """GridSearchRegressor function for each model given in model variable by creating pipeline with StandardScale() and fit data to predict.
        model: dict()
        scoring: r2 default
        plot_score_ = optional False default, for plotting socre
        compare_score = optional False default, for comparing the model scores
    """
    best_score = []
    name = []

    for regressor_name, regressor in model.items():
        pipeline = Pipeline( [('scaler', StandardScaler()),  ('model',regressor)] ) # create pipeline of each model with StandardScale
        grid_cv = GridSearchCV(estimator=pipeline, param_grid=param_grid, scoring = scoring, cv = 5) # estimate best parameter using gridSearchCV()
        grid_cv.fit(X_train, y_train) #  # fit data to the pipeline

        name.append(regressor_name)
        best_score.append(grid_cv.best_score_) # append best score from gridSearch

        print(regressor_name)
        print('Best score:',grid_cv.best_score_) # print best score from gridSearch
        print('Best param:',grid_cv.best_params_) # print best parameter from gridSearch

        best_grid = grid_cv.best_estimator_ # get best score from gridSearch best estimator
        y_pred=best_grid.predict(X_test) # predict using best estimator

        metrics(regressor_name,best_grid, y_pred )  # get metrics : r2, adj r2, rmse, rmsle
        plotResiduals(y_test, y_pred, regressor_name)
        distPlot(y_pred, y_test, name=regressor_name)
        if plot_score_:
            plot_score(grid_cv, regressor_name) # plot score
    if compare_score:
        comparePlot(best_score, name)    # plot comparison
```

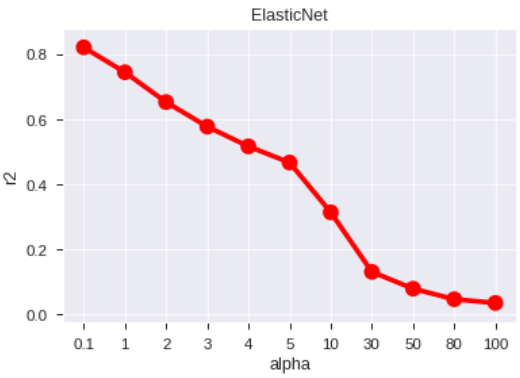












2.3.3 Random Forest

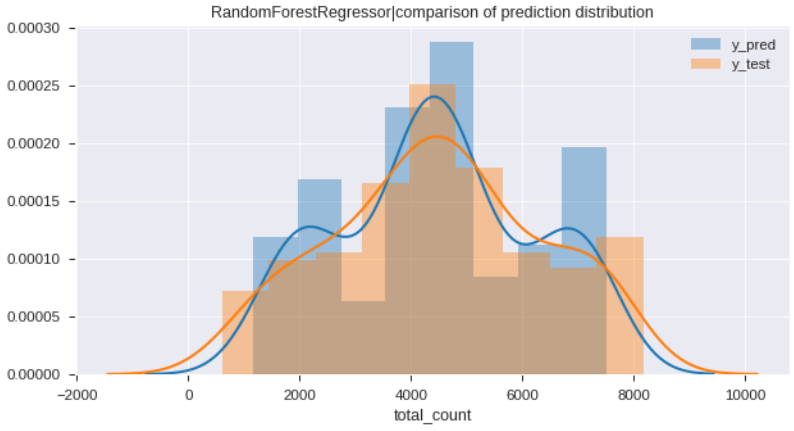
A Random Forest is an ensemble technique capable of performing both regression and classification tasks with the use of multiple decision trees and a technique called **Bootstrap Aggregation**, commonly known as **bagging**. What is bagging you may ask? Bagging, in the Random Forest method, involves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

Python:

```
# function: simple ensemble regressor
def ensembleRegressor(model):
    """Ensemble Regressor function for each model given in model variable by creating pipeline with StandardScale() and fit data to predict.
        model: dict()

    """
    if type(model) == dict:
        for regressor_name_, regressor_ in model.items():
            pipeline = Pipeline( [('scaler', StandardScaler()),  ('model',regressor_)] ) # create pipeline of each model with StandardScale
            regressor_ = pipeline.fit(X_train, y_train)                                   # fit data to the pipeline
            y_pred = regressor_.predict(X_test)                                           # Predict using the pipeline model
            metrics(regressor_name_,regressor_, y_pred )                                  # get metrics : r2, adj r2, rmse, rmsle
            plotResiduals(y_test, y_pred, regressor_name_)                                # plot residual graphs
```

```
# Simple RandomForestRegressor
ensemble_model1 = {'RandomForestRegressor':RandomForestRegressor(random_state=867)}
ensembleRegressor(ensemble_model1)

# RandomForestRegressor
# R^2: 0.8702914169811242
# Adj R^2:  0.8452144242641415
# RMSE RandomForestRegressor: 671.0592021821423
# RMSLE RandomForestRegressor: 0.23251743194514304
```

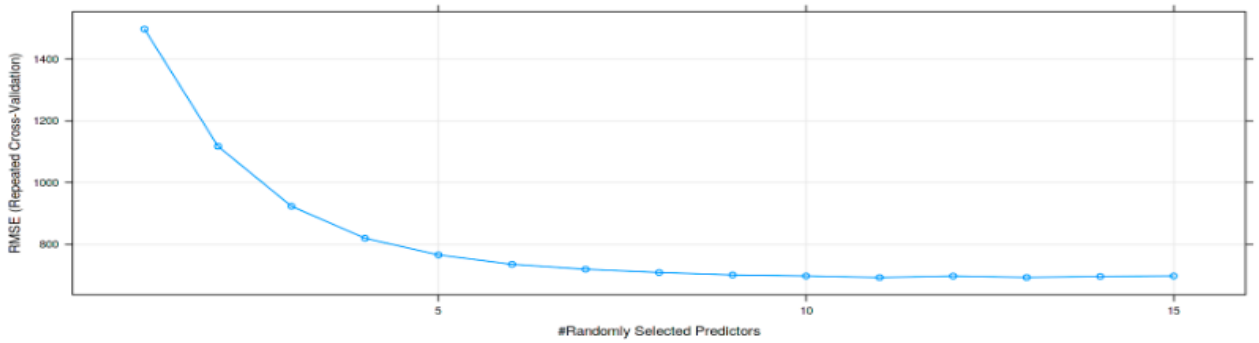
```
# Hyper-parameter tunning: RandomForestRegressor

param_grid = {
    'model__n_estimators' : [10,900],
    'model__max_depth': [5,6,7,10],
    'model__max_features' : ['log2','sqrt','auto'],
    'model__random_state': [897]
}

ensemble_model = {'RandomForestRegressor':RandomForestRegressor(random_state=867) }
gridSearchRegressor(ensemble_model,param_grid=param_grid )
```



R:

```r
# random froset

# Set seed to reproduce the results
set.seed(42)

# training the rf_model
rf_model = randomForest(train$total_count~.,data = train)

# get summary
rf_model

# predict test data using rf_model
rf_prediction = predict(rf_model, as.matrix(test[,-2]))
```

```r
# hyper-paramter tunning for randomforest model

# Set seed to reproduce the results
set.seed(14)

control = trainControl(method="repeatedcv", number=10, repeats=3, search="grid") # create parameter controls
tunegrid = expand.grid(.mtry=c(1:15)) # number of try

# tune the model
rf_gridsearch = train(total_count~., data=train, method="rf", metric='rmse', tuneGrid=tunegrid, trControl=control)
print(rf_gridsearch)
plot(rf_gridsearch)
```



2.3.4 Gradient Boosting

Gradient boosting is a machine learning technique for regression and classification problems, which produces a prediction model in the form of an ensemble of weak prediction models, typically decision trees. It builds the model in a stage-wise fashion like other boosting methods do, and it generalizes them by allowing optimization of an arbitrary differentiable loss function.

A big insight into bagging ensembles and random forest was allowing trees to be greedily created from subsamples of the training dataset. This same benefit can be used to reduce the correlation between the trees in the sequence in gradient boosting models. This variation of boosting is called **stochastic gradient boosting.**

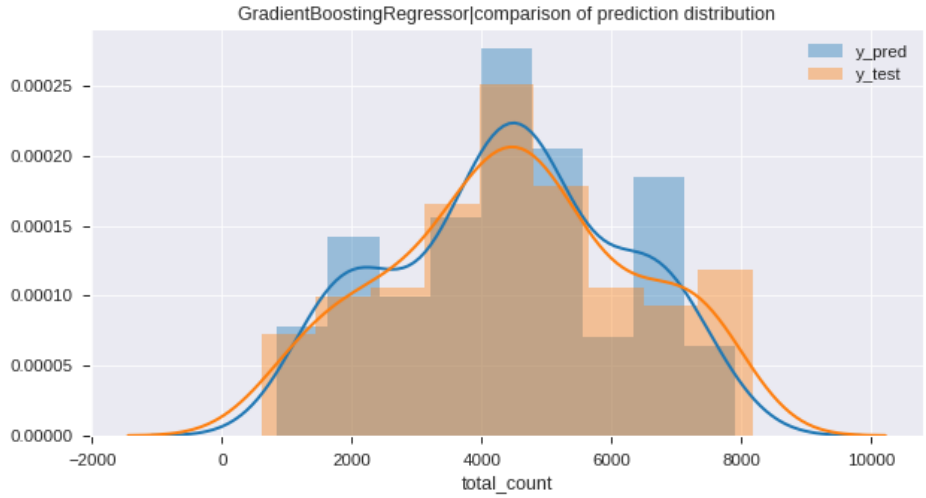
At each iteration a subsample of the training data is drawn at random (without replacement) from the full training dataset. The randomly selected subsample is then used, instead of the full sample, to fit the base learner.

— Stochastic Gradient Boosting

Python:

```
# Simple GradientBoostingRegressor
ensemble_model2 = {'GradientBoostingRegressor':GradientBoostingRegressor(random_state=867)}
ensembleRegressor(ensemble_model2)

# GradientBoostingRegressor
# R^2: 0.8824518010961264
# Adj R^2:  0.8597258159747109
# RMSE GradientBoostingRegressor: 638.8287724807816
# RMSLE GradientBoostingRegressor: 0.19844994226247784
```

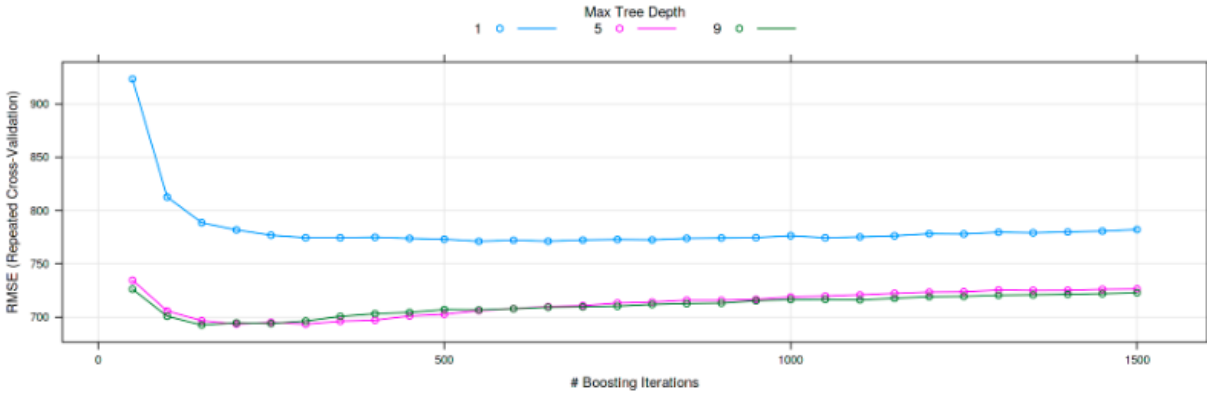
```
# Hyper-parameter tunning: GradientBoostingRegressor

hyper_param = {
                  'model__n_estimators' : [250,400,500,650,800], # The number of boosting stages to perform.
                  'model__max_depth' : [5,6,7,8], # maximum depth of the individual regression estimators
                  'model__max_features' : ['log2','sqrt','auto'], # The number of features to consider when looking for the best split:
                  'model__subsample' : [0.7,0.85,0.9], # The fraction of samples to be used for fitting the individual base learners.  If smaller than 1.0 this results in
                  'model__random_state': [17]

                 }

gbm_ensemble_model = {'GradientBoostingRegressor':GradientBoostingRegressor() }
gridSearchRegressor(gbm_ensemble_model,hyper_param )
```



R:

```r
# gbm

# Set seed to reproduce the results
set.seed(17)

# training the rf_model
gbm_model = gbm(total_count~.,data = train)

# get summary
gbm_model

ntrees = seq(from=100 ,to=10000, by=100) #no of trees-a vector of 100 values

# predict test data using gbm_model
gbm_prediction = round(predict(gbm_model, test[,-2], n.trees = 5000 ))
```

```r
# hyper-paramter tunning for gbm model

# Set seed to reproduce the results
set.seed(14)

fitControl = trainControl(method="repeatedcv", number=10, repeats=3, search="grid") # create parameter controls
ntrees = seq(from=100 ,to=10000, by=100) #no of trees-a vector of 100 values

# hyper-paramters
gbmGrid =  expand.grid(interaction.depth = c(1, 5, 9),
                       n.trees = (1:30)*50,
                       shrinkage = 0.1,
                       n.minobsinnode = 20)

nrow(gbmGrid)

set.seed(825) # Set seed to reproduce the results

# tune the model
gbm_gridsearch = train(total_count~., data = train,
                 method = "gbm",
                 trControl = fitControl,
                 verbose = FALSE,
                 ## Now specify the exact models
                 ## to evaluate:
                 tuneGrid = gbmGrid)

gbm_gridsearch
print(gbm_gridsearch)
plot(gbm_gridsearch)
```



CHAPTER 3

Conclusion

3.1 Model Evaluation

Now, we have 6 models for predicting the target variable, and we need to decide which model is better for this project. There are many metrics used for model evaluation.

There are several criteria that exist for evaluating and comparing models. We can compare the models using any of the following criteria:

1. Predictive Performance
2. Interpretability
3. Computational Efficiency

In our case of Bike Data, the latter two, Interpretability and Computation Efficiency, do not hold much significance. Therefore, we will use Predictive performance as the criteria to compare and evaluate models. Predictive performance can be measured by comparing Predictions of the models with real values of the target variables and calculating some average error measure.

1. **R-squared or Coefficient of Determination (*R2*)**: -
   It defines the degree to which the variance in the dependent variable (or target) can be explained by the independent variable (features). It explains the how much variance of dependent variable which is contributed by all the independent variables.

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_{true} - y_{pred})^2}{\sum_{i=1}^{n}(y_{true} - \bar{y}_{true})^2} = 1 - \frac{variance_{residuals}}{variance_{total}}$$

   Here, $y\ \bar{t}rue$ is the mean of true dependent variable values

2. **Adj R-squared: -**
   Adjusted R-squared measures the variation in the dependent variable (or target), explained by only the features which are helpful in making predictions. Unlike R-squared, the Adjusted R-squared would penalize you for adding features which are not useful for predicting the target.

$$R^2 = 1 - \frac{RSS}{TSS}$$

$$R^2_{adj} = 1 - ((1 - R^2)\frac{N-1}{N-M-1}$$

   Here R^2 is the r-squared calculated, N is the number of rows and M is the number of columns. As the number of feature increases, the value in the denominator decreases.

- If the R2 increases by a significant value, then the adjusted r-squared would increase.
- If there is no significant change in R2, then the adjusted r2 would decrease.

3. **Root Mean Square Error** (RMSE): - It is the square root of the mean of the square of difference between the true and predicted dependent variable values.

$$\text{RMSE}=\sqrt{\frac{\sum_{i=1}^{n}(y_{true}{}^2-y_{pred}{}^2)}{n}}$$

4. **Root Mean Square Log Error** (RMSLE): -

RMSLE penalizes an under-predicted estimate greater than an over-predicted estimate.

$$\text{RMSLE} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(\log(y_i+1)-\log(\hat{y}_i+1))^2}$$

When the differences from predicted and actuals are large the log function helps normalizing this.
By applying logarithms to both prediction and actual numbers, we'll get smoother results by reducing the impact of larger x, while emphasize of smaller x.
RMSLE measures the ratio between actual and predicted.

The problem with R-squared is that it will either stay the same or increase with addition of more variables, even if they do not have any relationship with the output variables. This is where "Adjusted R square" comes to help. Adjusted R-square penalizes you for adding variables which do not improve your existing model. Hence, if you are building Linear regression on multiple variable, it is always suggested that you use **Adjusted R-squared to judge goodness of model**. In case you only have one input variable, R-square and Adjusted R squared would be exactly same. Typically, the more non-significant variables you add into the model, the gap in R-squared and Adjusted R-squared increases.

In case of RMSLE, we take the log of the predictions and actual values. So basically, what changes is the variance that we are measuring. RMSLE is usually used when we don't want to penalize huge differences in the predicted and the actual values when both predicted and true values are huge numbers.

1. If both predicted and actual values are small: RMSE and RMSLE is same.
2. If either predicted or the actual value is big: RMSE > RMSLE
3. If both predicted and actual values are big: RMSE > RMSLE (RMSLE becomes almost negligible)

Python:

```
# function: get metrics
def metrics(regressor_name, regressor, y_pred):
    """Print metrics: r2, adj r2, rmse, rmsle
    parameters:
        regressor_name: list, dataframe, matrix
        regressor: fitted model object
        y_pred: list, dataframe, matrix
    """
    print(regressor_name)                        # print regressor name
    print('R^2:',regressor.score(X_test, y_test))  # Returns the coefficient of determination R^2 of the prediction.
    print('Adj R^2: ', 1 - (1-regressor.score(X_test, y_test))*(len(y_test)-1)/(len(y_test)-X_test.shape[1]-1)) # Returns the coefficient of determination Adj R^2 of the p
    print('RMSE {}: {}'.format(regressor_name,  np.sqrt(mean_squared_error(y_test, y_pred)))) # Retrun rmse score
    print('RMSLE {}: {}'.format(regressor_name,  np.sqrt(mean_squared_log_error(y_test, y_pred)))) # Retrun rmsle score
```

R :

```
cat('\n\nMetrics on Test data:')
cat('\nMAPE:',MAPE(test[,2], lasso_prediction))
cat('\nR^2:',R_squared(test[,2], lasso_prediction))
cat('\nAdj R^2:', Adj_R_squared(r2=R_squared(test[,2], lasso_prediction), n,p))
cat('\nRMSE:',rmse(test[,2], lasso_prediction))
cat('\nRMSLE:',rmsle(test[,2], lasso_prediction))
```

Python:

```
# Score without StandardScale:

# LinearRegression
# R^2: 0.8503150391570842
# Adj R^2:  0.8213759467274537
# RMSE LinearRegression: 720.8842925663198
# RMSLE LinearRegression: 0.2465516805375247


# Ridge
# R^2: 0.8478419303061329
# Adj R^2:  0.8184247034986518
# RMSE Ridge: 726.8151540447384
# RMSLE Ridge: 0.2381610980768202


# Lasso
# R^2: 0.8504890420887281
# Adj R^2:  0.8215835902258822
# RMSE Lasso: 720.4651707841829
# RMSLE Lasso: 0.23878836957603555


# ElasticNet
# R^2: 0.379145172923085
# Adj R^2:  0.25911323968821476
# RMSE ElasticNet: 1468.1536519170734
# RMSLE ElasticNet: 0.46300107772627347
```

```
# Score with StandardScale:

# LinearRegression
# R^2: 0.8503150391570844
# Adj R^2:  0.8213759467274541
# RMSE LinearRegression: 720.8842925663195
# RMSLE LinearRegression: 0.24655168053752335

# Ridge
# R^2: 0.8502971862653109
# Adj R^2:  0.8213546422766044
# RMSE Ridge: 720.9272811386522
# RMSLE Ridge: 0.24640128765416605


# Lasso
# R^2: 0.85043352042173
# Adj R^2:  0.8215173343699311
# RMSE Lasso: 720.5989325995707
# RMSLE Lasso: 0.24569841589499636


# ElasticNet
# R^2: 0.7690031028056619
# Adj R^2:  0.7243437026814232
# RMSE ElasticNet: 895.528789006083
# RMSLE ElasticNet: 0.2952233398791348
```

```
# Score with GridSearchCV:

# Ridge
# Best score: 0.8223825219791182
# Best param: {'model__alpha': 5, 'model__max_iter': 3000}
# Ridge
# R^2: 0.8501161318996594
# Adj R^2:  0.8211385840669269
# RMSE Ridge: 721.3631032400775
# RMSLE Ridge: 0.24571028154769625

# Lasso
# Best score: 0.8220643248274393
# Best param: {'model__alpha': 3, 'model__max_iter': 3000}
# Lasso
# R^2: 0.8505544480598202
# Adj R^2:  0.8215178229750227
# RMSE Lasso: 720.3075640121864
# RMSLE Lasso: 0.24420533245983528
# Best param: {'model__alpha': 3, 'model__max_iter': 3000}

# ElasticNet
# Best score: 0.8209891999677044
# Best param: {'model__alpha': 0.1, 'model__max_iter': 3000}
# ElasticNet
# R^2: 0.8473885489959245
# Adj R^2:  0.7243433797638379
# RMSE ElasticNet: 727.8971844088097
# RMSLE ElasticNet: 0.24186048423041664
# Best param: {'model__alpha': 0.1, 'model__max_iter': 3000}
```

```
# RandomForestRegressor
# R^2: 0.8649320139183166
# Adj R^2:  0.8388188699425245
# RMSE RandomForestRegressor: 684.7825594871541
# RMSLE RandomForestRegressor: 0.2333816742631807
```

```
# RandomForestRegressor
# Best score: 0.8587517092660517
# Best param: {'model__max_depth': 10, 'model__max_features': 'auto', 'model__n_estimators': 900, 'model__random_state': 897}
# RandomForestRegressor
# R^2: 0.8715849042781012
# Adj R^2:  0.8467579857718674
# RMSE RandomForestRegressor: 667.7048312259761
# RMSLE RandomForestRegressor: 0.2349906934957484
```

```
# GradientBoostingRegressor
# R^2: 0.8862719523839226
# Adj R^2:  0.8642845298448143
# RMSE GradientBoostingRegressor: 628.3625167761045
# RMSLE GradientBoostingRegressor: 0.19722626748641778
```

```
# GradientBoostingRegressor
# Best score: 0.8869327747143647
# Best param: {'model__max_depth': 5, 'model__max_features': 'sqrt', 'model__n_estimators': 250, 'model__random_state': 17, 'model__subsample': 0.7}
# GradientBoostingRegressor
# R^2: 0.8870138124626565
# Adj R^2:  0.8651698162054368
# RMSE GradientBoostingRegressor: 626.3097260903672
# RMSLE GradientBoostingRegressor: 0.2143924133958078
```

R:

```
# simple linear model
# MAPE: 15.85646
# R^2: 0.8717872
# Adj R^2: 0.8457981
# RMSE: 746.702
# RMSLE: 0.2219201
```

```
# lasso model
# Metrics on Test data:
# MAPE: 15.881
# R^2: 0.8718666
# Adj R^2: 0.8458936
# RMSE: 746.4707
# RMSLE: 0.2208965
```

```
# ridge model
# Metrics on Test data:
# MAPE: 17.04704
# R^2: 0.8597313
# Adj R^2: 0.8312985
# RMSE: 781.0196
# RMSLE: 0.2163474
```

```
# elasticnet model
# Metrics on Test data:
# MAPE: 16.19882
# R^2: 0.8694822
# Adj R^2: 0.8430259
# RMSE: 753.3842
# RMSLE: 0.215776
```

```
# Call:
#  randomForest(formula = train$total_count ~ ., data = train)
#                Type of random forest: regression
#                      Number of trees: 500
# No. of variables tried at each split: 9

#           Mean of squared residuals: 499178.1
#                     % Var explained: 85.85


# Metrics on Test data:
# MAPE: 13.73607
# R^2: 0.9075762
# Adj R^2: 0.8888417
# RMSE: 633.9769
# RMSLE: 0.1943775
```

```
# gbm(formula = total_count ~ ., data = train)
# A gradient boosted model with gaussian loss function.
# 100 iterations were performed.
# There were 29 predictors of which 11 had non-zero influence.


# Metrics on Test data:
# MAPE: 18.46409
# R^2: 0.8639603
# Adj R^2: 0.8363847
# RMSE: 769.156
# RMSLE: 0.2498177
```

```
# Random Forest

# 538 samples
#  29 predictor

# No pre-processing
# Resampling: Cross-Validated (10 fold, repeated 3 times)
# Summary of sample sizes: 484, 483, 485, 484, 484, 484, ...
# Resampling results across tuning parameters:

#   mtry  RMSE       Rsquared   MAE
#    1    1497.8445  0.7212547  1206.6032
#    2    1117.8434  0.7966798   904.4551
#    3     923.3875  0.8318335   733.4619
#    4     819.7871  0.8523184   631.9902
#    5     765.7639  0.8602762   574.4973
#    6     735.2557  0.8646165   544.6450
#    7     719.2258  0.8659400   525.5161
#    8     708.7438  0.8666198   514.1552
#    9     700.9273  0.8672585   505.7798
#   10     697.4989  0.8669734   502.8166
#   11     692.5154  0.8674696   497.8712              <---- optimal parameter
#   12     697.2182  0.8645921   498.2345
#   13     692.8876  0.8656716   495.1647
#   14     696.0455  0.8637380   495.6083
#   15     697.5091  0.8628047   496.2870

# RMSE was used to select the optimal model using the smallest value.
# The final value used for the model was mtry = 11.
```

```
# Stochastic Gradient Boosting

# 538 samples
#  29 predictor

# No pre-processing
# Resampling: Cross-Validated (10 fold, repeated 3 times)
# Summary of sample sizes: 484, 484, 485, 484, 484, 486, ...
# Resampling results across tuning parameters:

#   interaction.depth  n.trees  RMSE      Rsquared   MAE
#   1                    50     923.3652  0.7839935  727.0566
#   1                   100     812.6623  0.8175004  630.9014
#   1                   150     788.5031  0.8257042  602.6373
#   1                   200     781.7537  0.8288068  592.3475
#   1                   250     776.8015  0.8309069  588.0951
# |.....................................................
#   9                   100     700.7431  0.8623221  507.3497
#   9                   150     692.5382  0.8657030  503.5086          <------ optimal model w
#   9                   200     694.5902  0.8651411  505.3311
#   9                   250     693.9644  0.8652073  506.3707
#   9                   300     696.2700  0.8646263  506.9534
# .....................................................
#   9                  1500     722.8684  0.8551270  526.8665

# Tuning parameter 'shrinkage' was held constant at a value of 0.1

# Tuning parameter 'n.minobsinnode' was held constant at a value of 20
# RMSE was used to select the optimal model using the smallest value.
# The final values used for the model were n.trees = 150, interaction.depth =
#  9, shrinkage = 0.1 and n.minobsinnode = 20.
```

```
# tuned random froset
# Metrics on Test data:
# MAPE: 13.46603
# R^2: 0.9099247
# Adj R^2: 0.8916662
# RMSE: 625.8704
# RMSLE: 0.1940767
```

```
# gbm_gridsearch
# Metrics on Test data:
# MAPE: 13.70147
# R^2: 0.9137555
# Adj R^2: 0.8962735
# RMSE: 612.4171
# RMSLE: 0.2049584
```

3.1.1 R-squared & Adj R-squared

Python:

Model	R-squared	Adj R-squared
Linear Regression	0.8503150391570844	0.8213759467274541
LASSO Regression	0.8505544480598202	0.8215178229750227
Ridge Regression	0.8502971862653109	0.8213546422766044
ElasticNet Regression	0.8473885489959245	0.7243433797638379
Random Forest Regression	0.8715849042781012	0.8467579857718674
Gradient Boosting Regression	0.8870138124626565	0.8651698162054368

R:

Model	R-squared	Adj R-squared
Linear Regression	0.8717872	0.8457981
LASSO Regression	0.8718666	0.8458936
Ridge Regression	0.8597313	0.8312985
ElasticNet Regression	0.8694822	0.8430259
Random Forest Regression	0.9099247	0.8916662
Gradient Boosting Regression	0.9137555	0.8962735

3.1.2 Root Mean Squared Error (RMSE) & Root Mean Squared Log Error

Python:

Model	RMSE	RMSLE
Linear Regression	720.8842925663195	0.24655168053752335
LASSO Regression	720.3075640121864	0.24420533245983528
Ridge Regression	720.9272811386522	0.24640128765416605
ElasticNet Regression	727.8971844088097	0.24186048423041664
Random Forest Regression	667.7048312259761	0.2349906934957484
Gradient Boosting Regression	626.3097260903672	0.2143924133958078

R:

Model	RMSE	RMSLE
Linear Regression	746.702	0.2219201
LASSO Regression	746.4707	0.2208965
Ridge Regression	781.0196	0.2163474
ElasticNet Regression	753.3842	0.215776
Random Forest Regression	625.8704	0.1940767
Gradient Boosting Regression	612.4171	0.2049584

3.2 Model Selection

We have implemented 6 models and out of 6 ensemble models performing best on our error metrics. Followings are the conclusion we made:

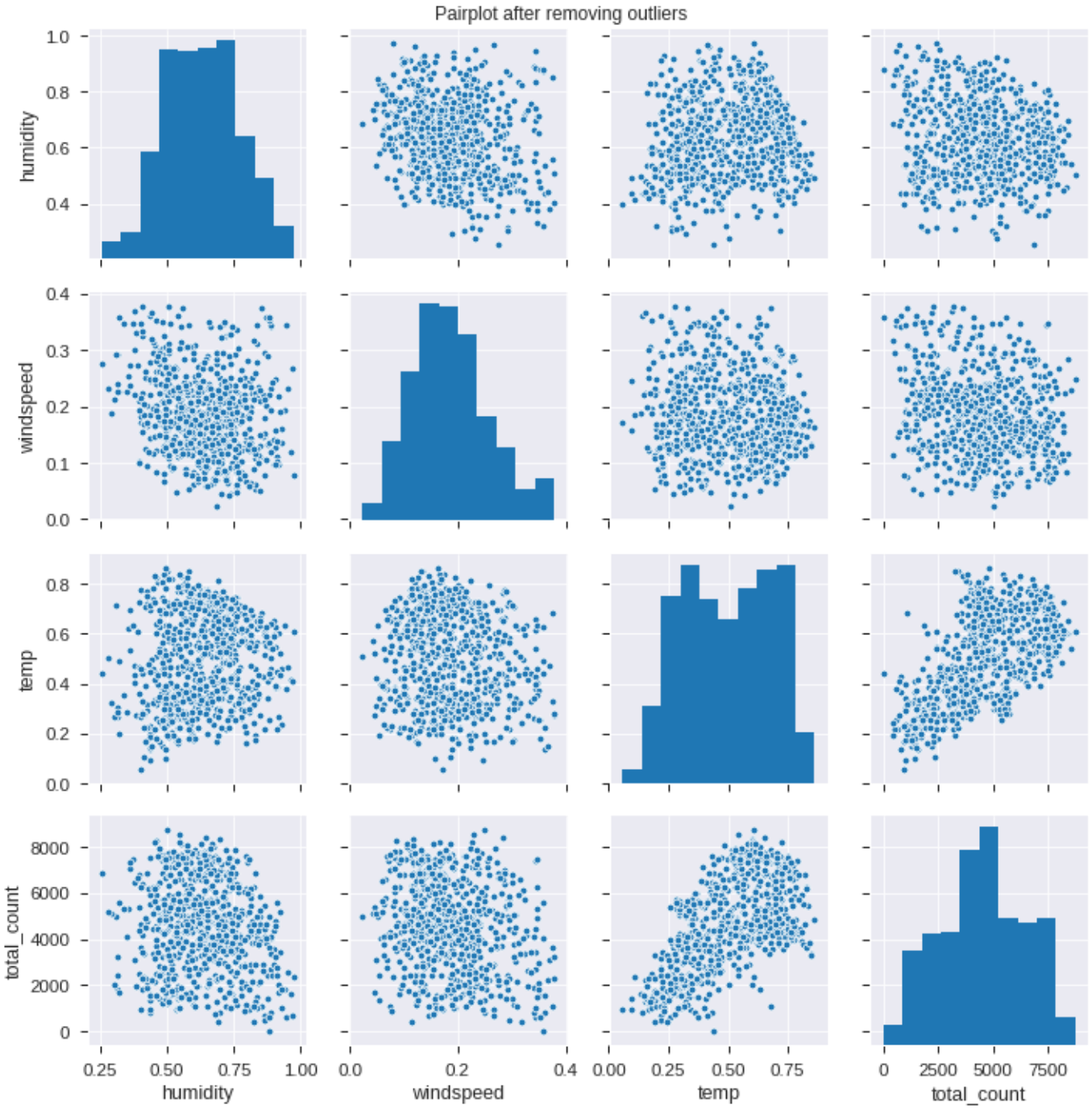
- Gradient Boosting model is performing best in both python and R.

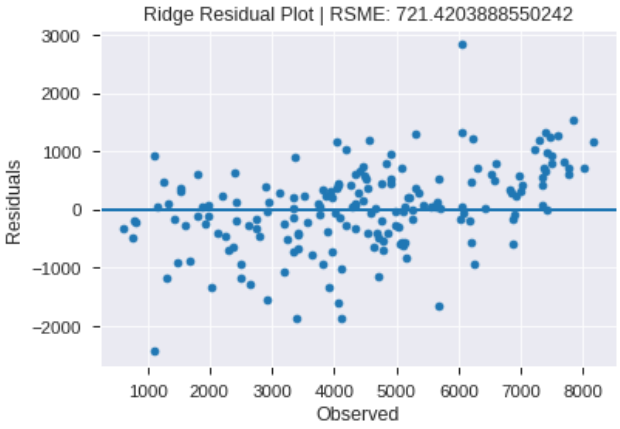
	Python	R
R-squared	0.887013	0.913755
Adj R-squared	0.865169	0.896273
RMSE	626.3097	612.4171
RMSLE	0.214392	0.204958

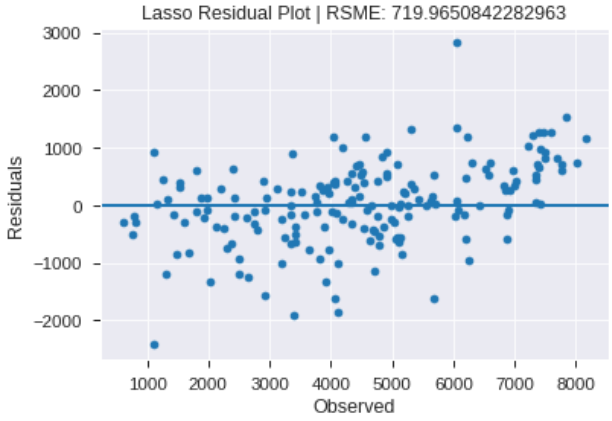
- the dataset contained very less samples (around 731), due to which a large RMSE value was observed while training different models.
- by increasing the no of samples, the model can learn better, and overall error will be reduced.
- **temperature, year, season, humidity,** and **weather** appeared to be the top 5 features affecting bike rental count.

Appendix A - Extra Figures

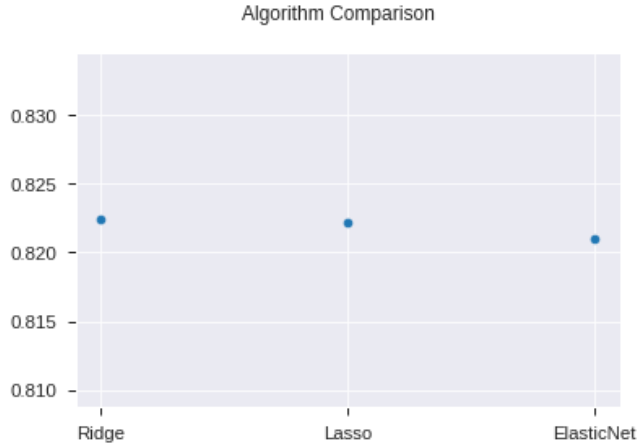
Python:

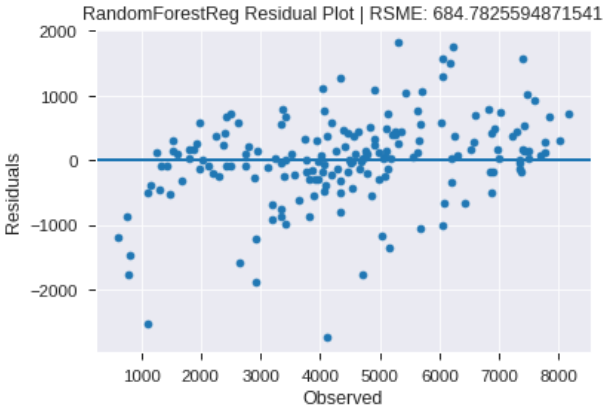


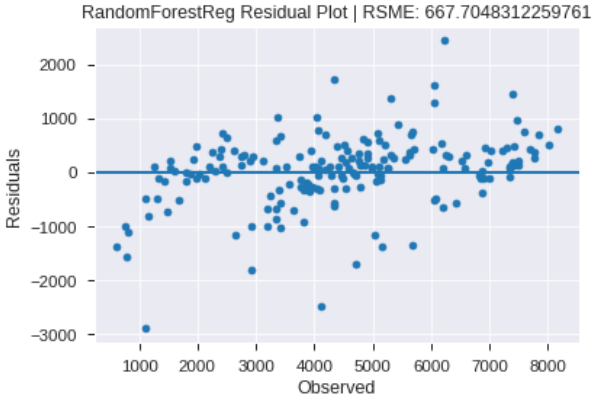

Ridge Residual Plot | RSME: 721.4203888550242
Residuals
Observed

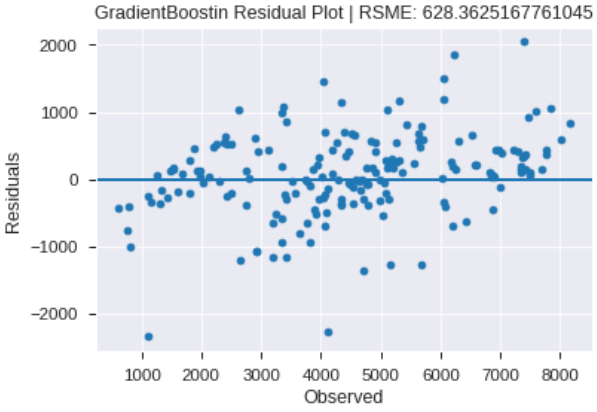

Lasso Residual Plot | RSME: 719.9650842282963
Residuals
Observed

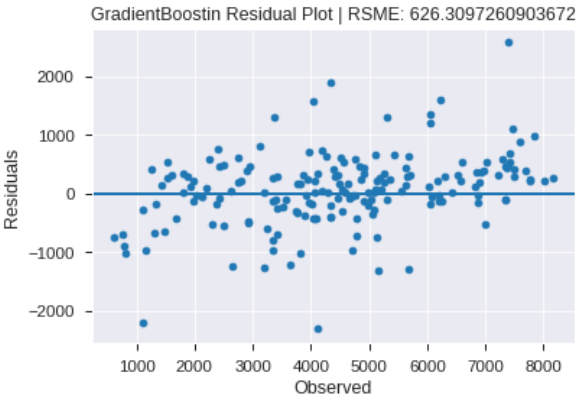

ElasticNet Residual Plot | RSME: 728.0040805247589
Residuals
Observed

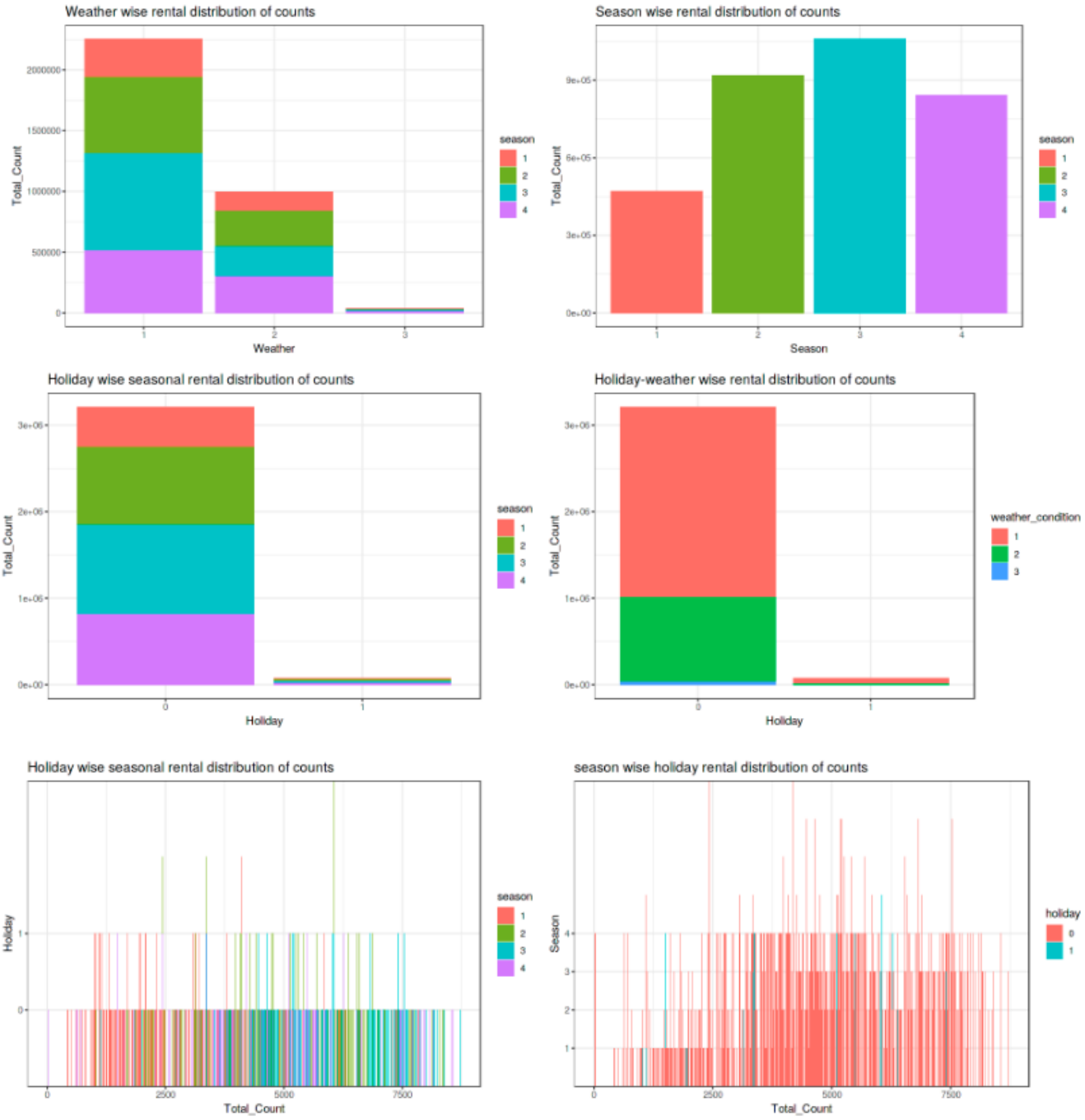

Algorithm Comparison
Ridge
Lasso
ElasticNet

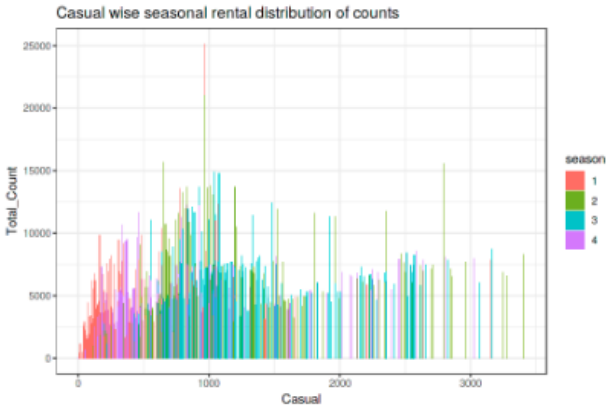

RandomForestReg Residual Plot | RSME: 684.7825594871541
Residuals
Observed

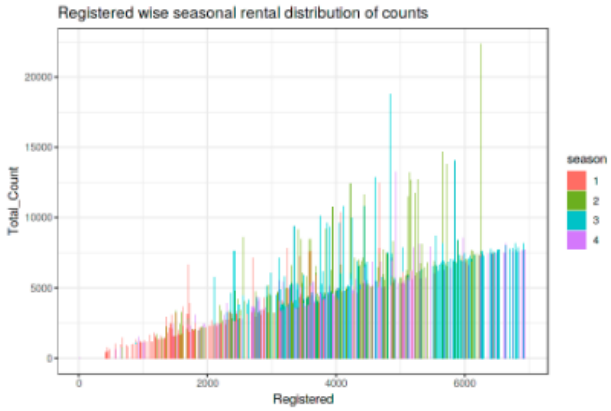

RandomForestReg Residual Plot | RSME: 667.7048312259761
Residuals
Observed

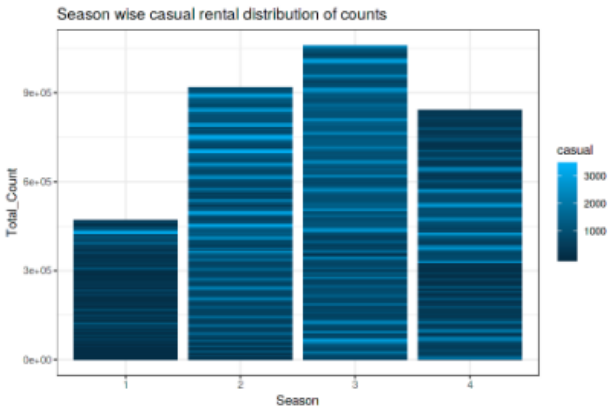

GradientBoostin Residual Plot | RSME: 628.3625167761045
Residuals
Observed


GradientBoostin Residual Plot | RSME: 626.3097260903672
Residuals
Observed

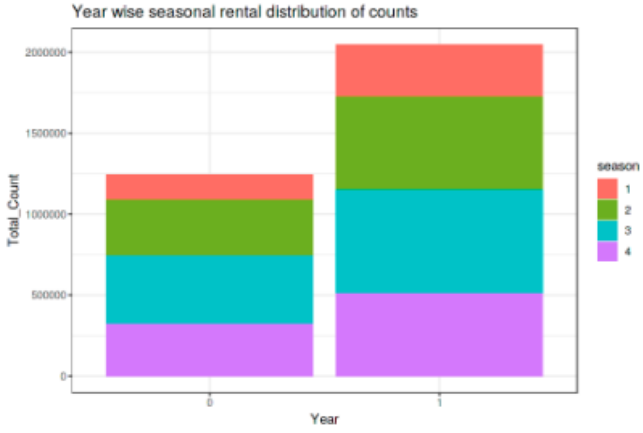

Weather wise rental distribution of counts
Total_Count
Weather
season
1
2
3
4
Season wise rental distribution of counts
Total_Count
Season
season
Holiday wise seasonal rental distribution of counts
Total_Count
Holiday
season
Holiday-weather wise rental distribution of counts
Total_Count
Holiday
weather_condition
1
2
3
Holiday wise seasonal rental distribution of counts
Holiday
Total_Count
season
season wise holiday rental distribution of counts
Season
Total_Count
holiday
0
1

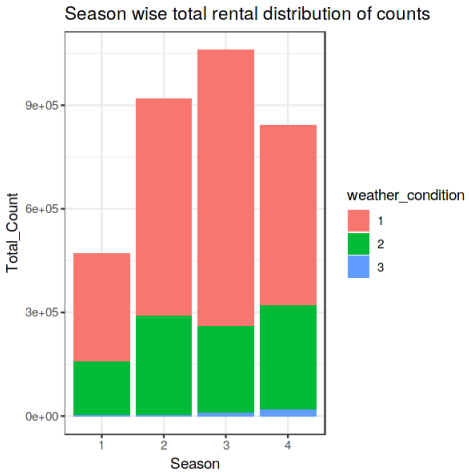

Casual wise seasonal rental distribution of counts
Total_Count
Casual
season
1
2
3
4

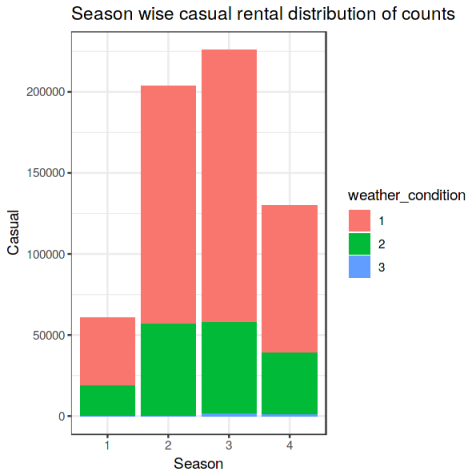

Registered wise seasonal rental distribution of counts
Total_Count
Registered
season
1
2
3
4

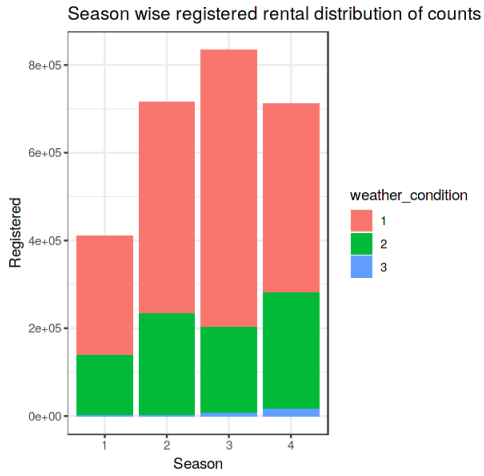

Season wise casual rental distribution of counts
Total_Count
Season
casual
3000
2000
1000

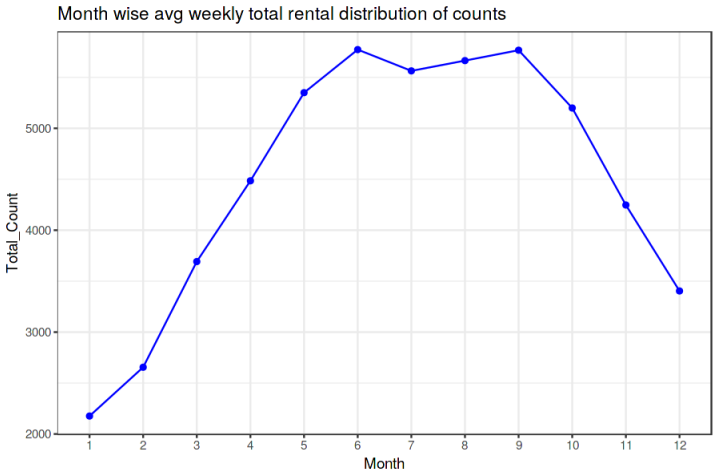

Season wise registered rental distribution of counts
Total_Count
Season
registered
6000
4000
2000

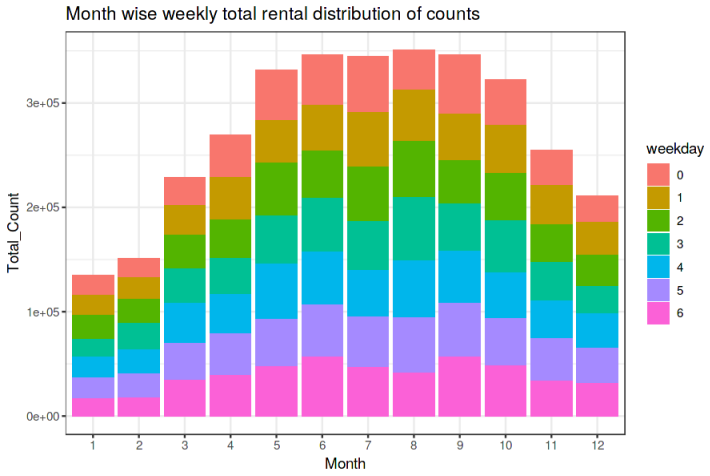

Year wise seasonal rental distribution of counts
Total_Count
Year
season
1
2
3
4

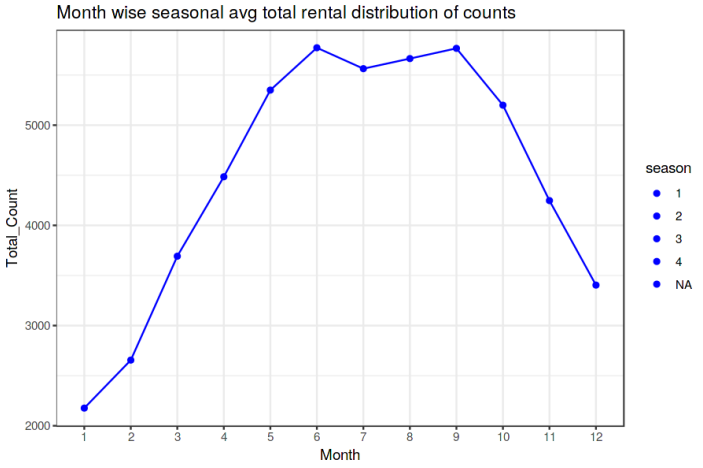

Season wise total rental distribution of counts
Total_Count
9e+05
6e+05
3e+05
0e+00
1
2
3
4
Season
weather_condition
1
2
3

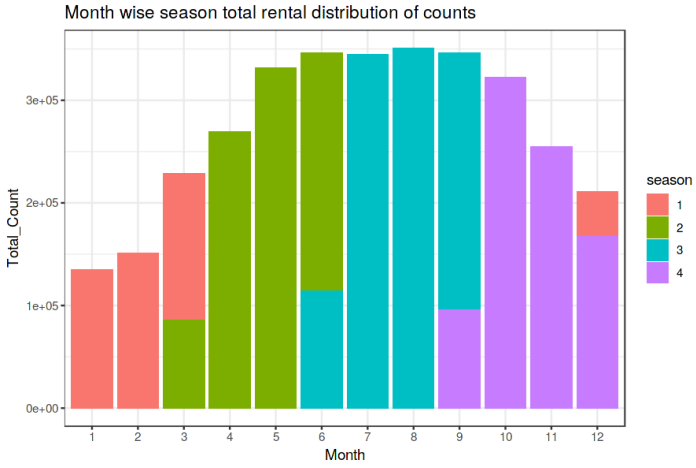

Season wise casual rental distribution of counts
Casual
200000
150000
100000
50000
0
1
2
3
4
Season
weather_condition
1
2
3

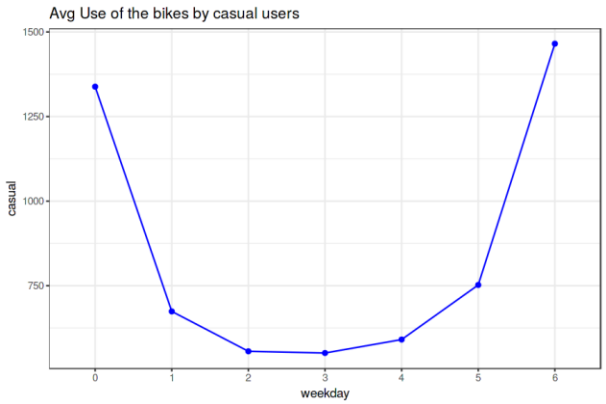

Season wise registered rental distribution of counts
Registered
8e+05
6e+05
4e+05
2e+05
0e+00
1
2
3
4
Season
weather_condition
1
2
3

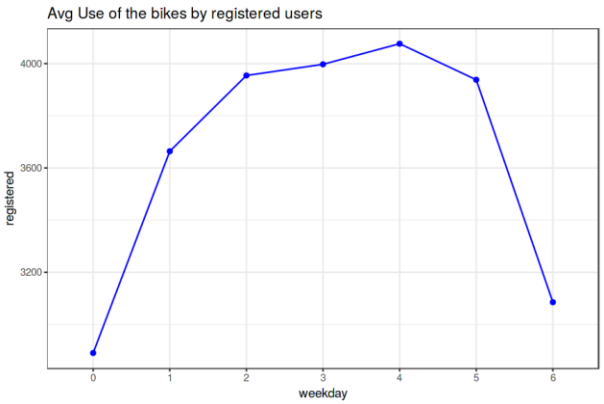

Month wise avg weekly total rental distribution of counts
Total_Count
5000
4000
3000
2000
1
2
3
4
5
6
7
8
9
10
11
12
Month

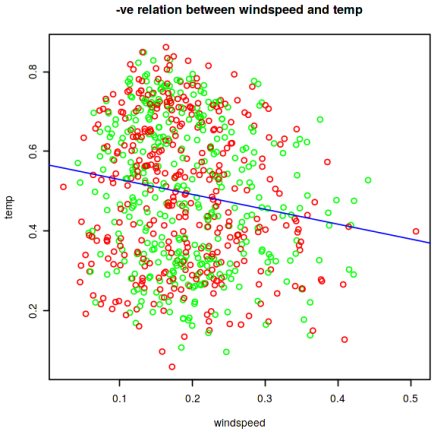

Month wise weekly total rental distribution of counts
Total_Count
3e+05
2e+05
1e+05
0e+00
1
2
3
4
5
6
7
8
9
10
11
12
Month
weekday
0
1
2
3
4
5
6

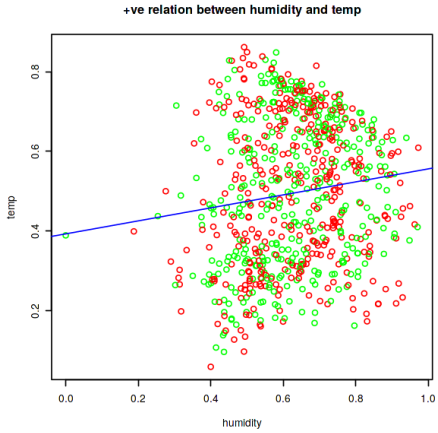

Month wise seasonal avg total rental distribution of counts
Total_Count
5000
4000
3000
2000
1
2
3
4
5
6
7
8
9
10
11
12
Month
season
1
2
3
4
NA

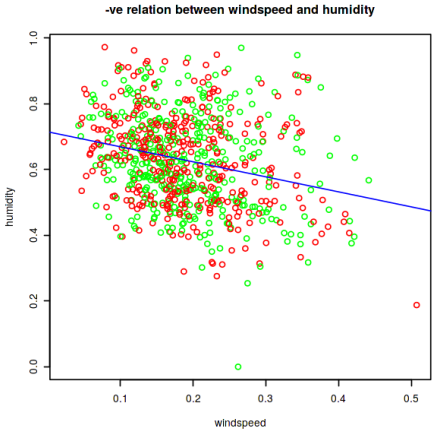

Month wise season total rental distribution of counts
Total_Count
3e+05
2e+05
1e+05
0e+00
1
2
3
4
5
6
7
8
9
10
11
12
Month
season
1
2
3
4

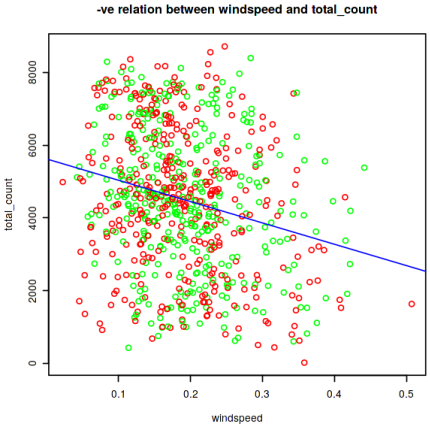

Avg Use of the bikes by casual users
casual
weekday

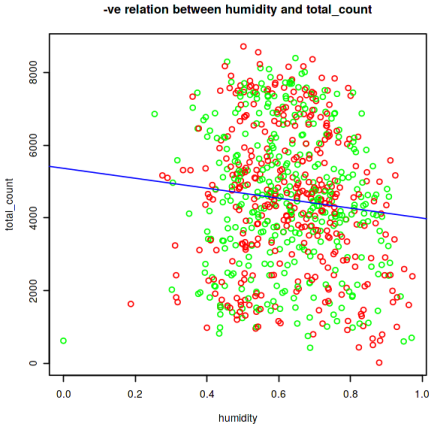

Avg Use of the bikes by registered users
registered
weekday

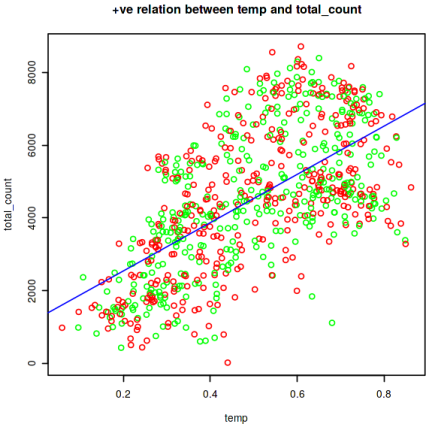

-ve relation between windspeed and temp
temp
windspeed

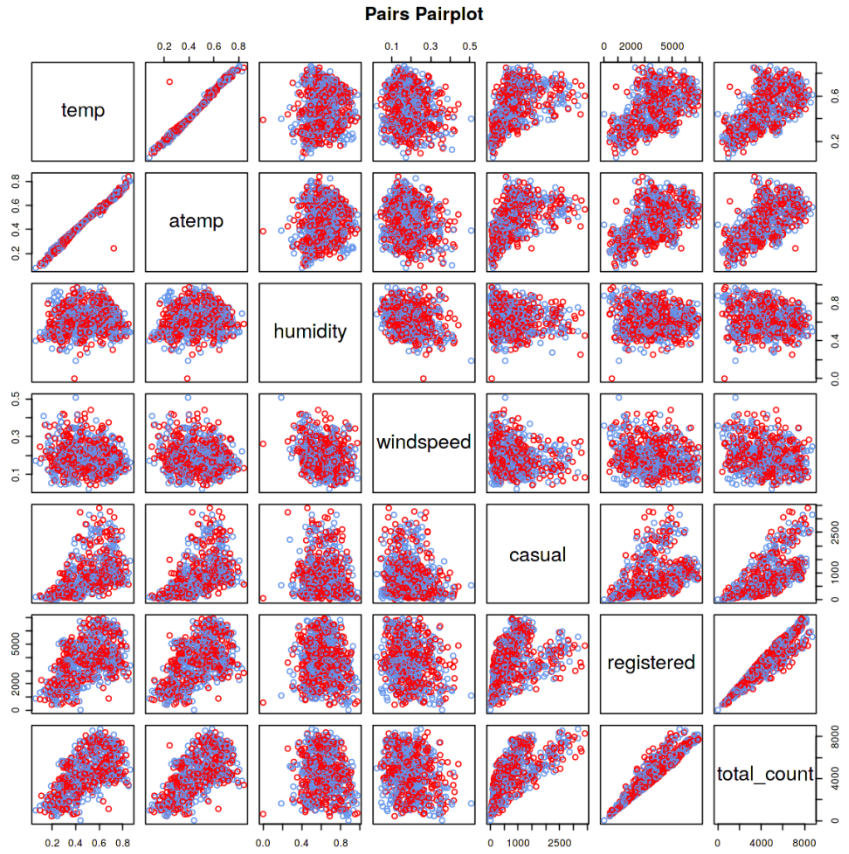

+ve relation between humidity and temp
temp
humidity

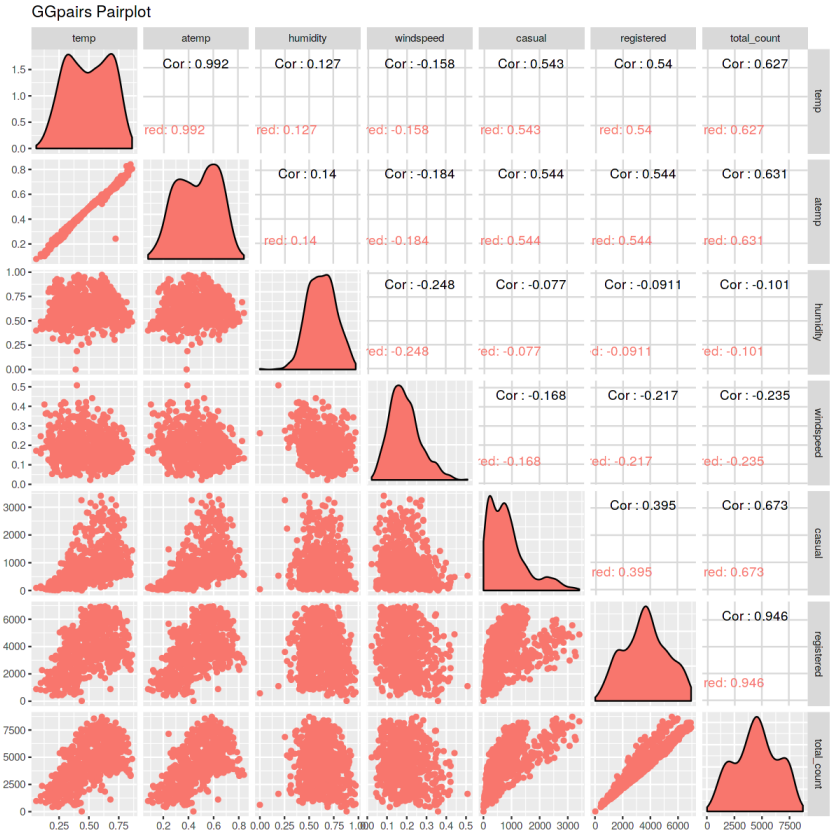

-ve relation between windspeed and humidity
humidity
windspeed

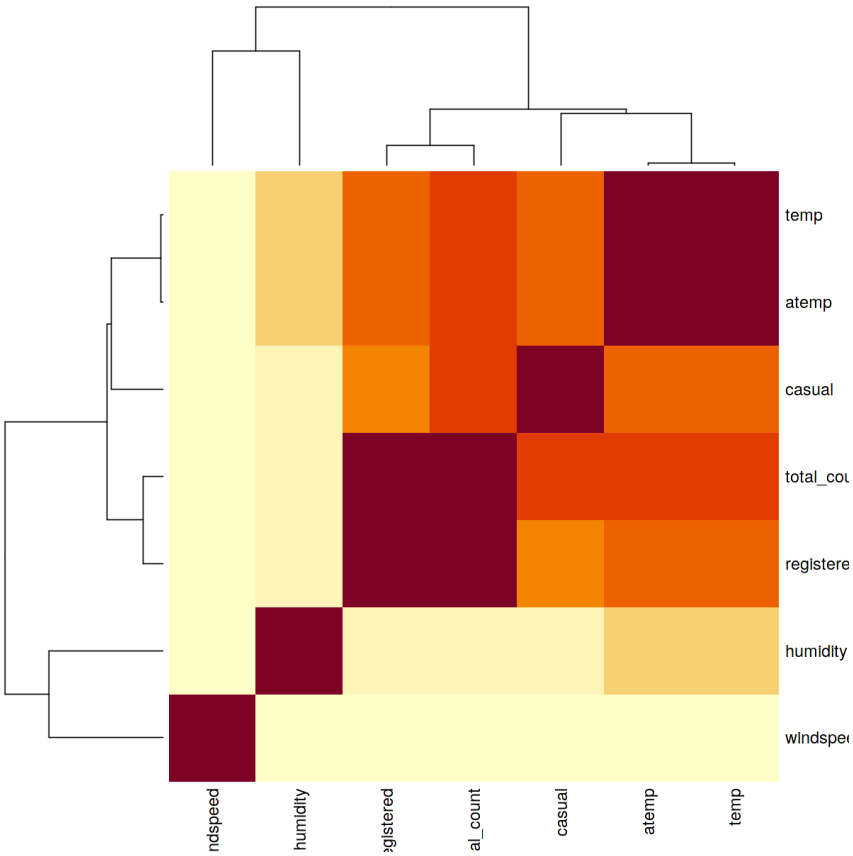

-ve relation between windspeed and total_count
total_count
windspeed

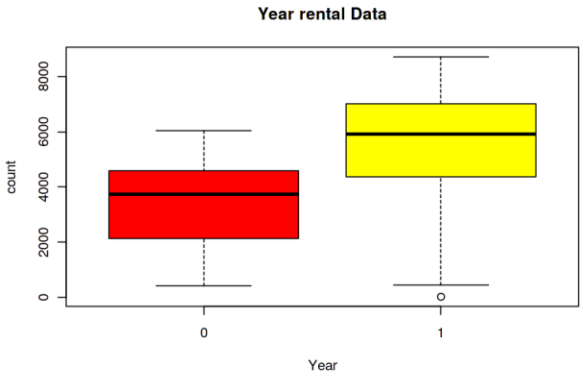

-ve relation between humidity and total_count
total_count
humidity

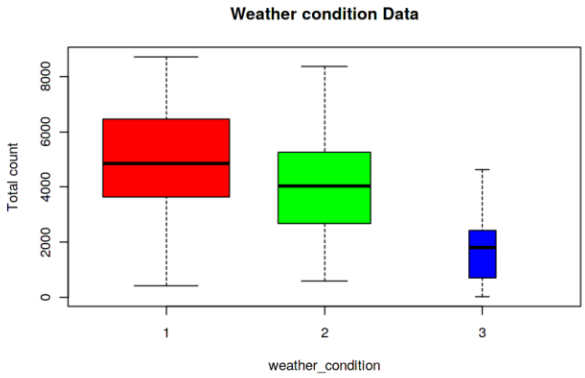

+ve relation between temp and total_count
total_count
temp

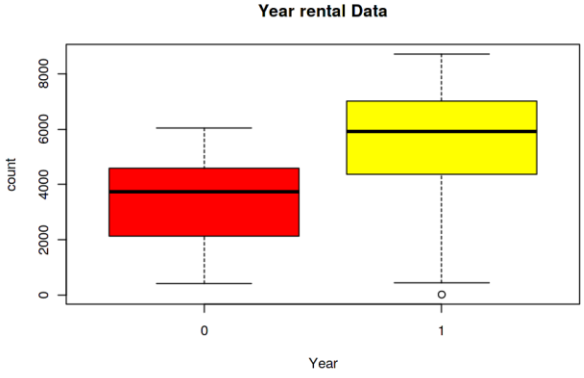

Pairs Pairplot
temp
atemp
humidity
windspeed
casual
registered
total_count

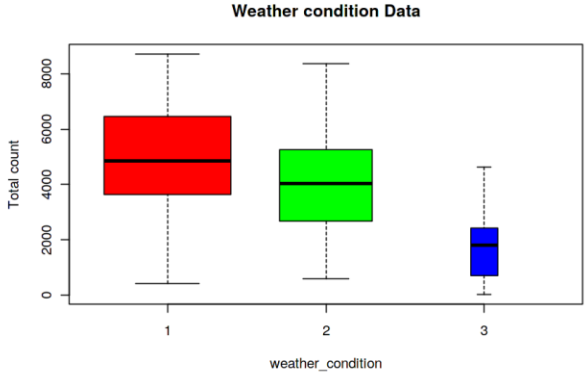

GGpairs Pairplot
temp
atemp
humidity
windspeed
casual
registered
total_count
Cor : 0.992
Cor : 0.127
Cor : -0.158
Cor : 0.543
Cor : 0.54
Cor : 0.627
Cor : 0.14
Cor : -0.184
Cor : 0.544
Cor : 0.631
Cor : -0.248
Cor : -0.077
Cor : -0.0911
Cor : -0.101
Cor : -0.168
Cor : -0.217
Cor : -0.235
Cor : 0.395
Cor : 0.673
Cor : 0.946

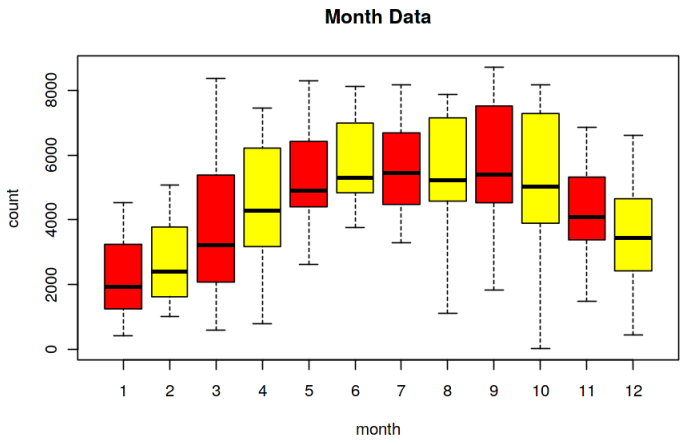

temp
atemp
casual
total_cou
registere
humidity
windspee
ndspeed
humidity
egistered
al_count
casual
atemp
temp

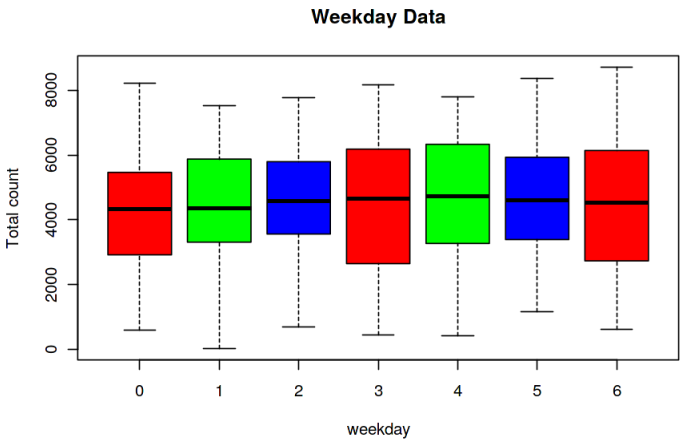

Year rental Data
count
0
2000
4000
6000
8000
0
1
Year

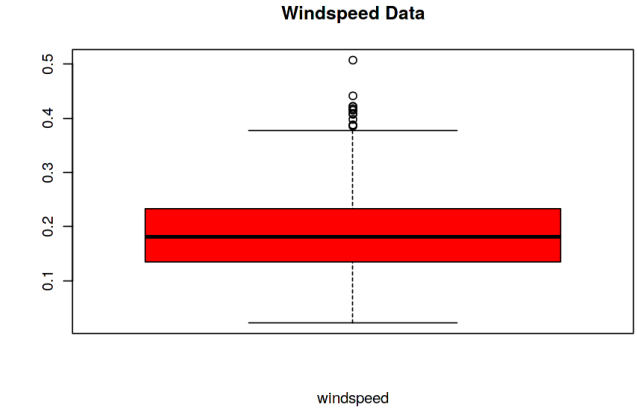

Weather condition Data
Total count
0
2000
4000
6000
8000
1
2
3
weather_condition

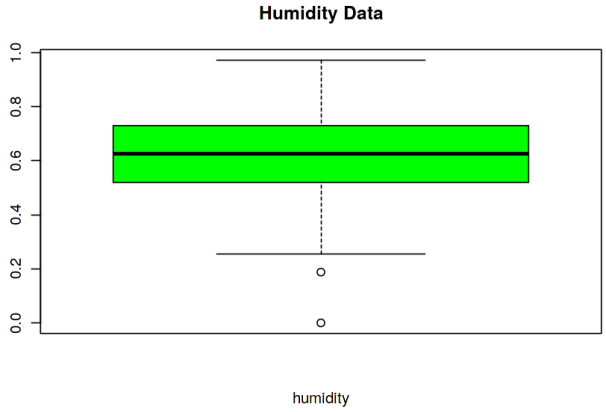

Year rental Data
count
0
2000
4000
6000
8000
0
1
Year

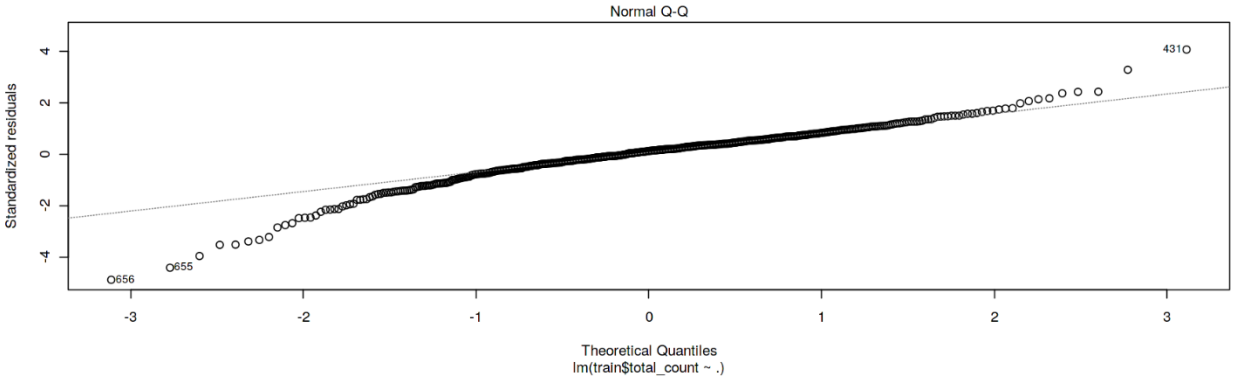

Weather condition Data
Total count
0
2000
4000
6000
8000
1
2
3
weather_condition

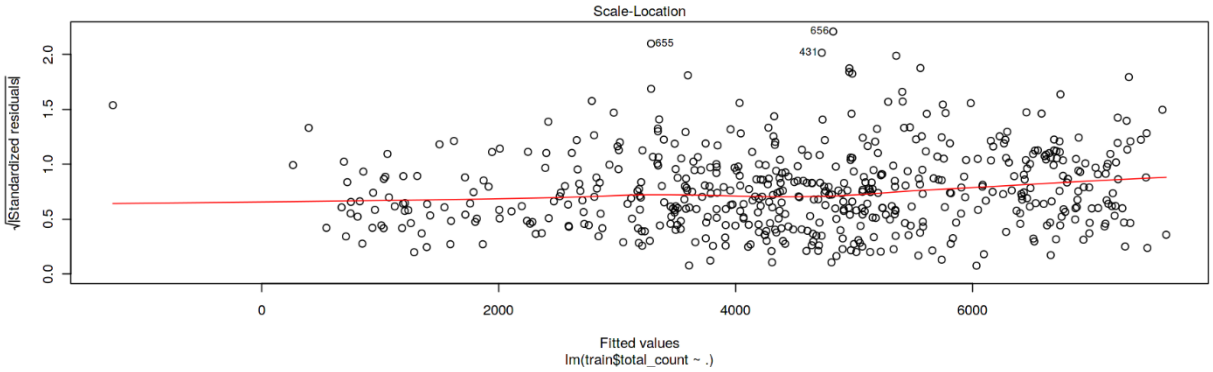

Month Data
count
0
2000
4000
6000
8000
1
2
3
4
5
6
7
8
9
10
11
12
month

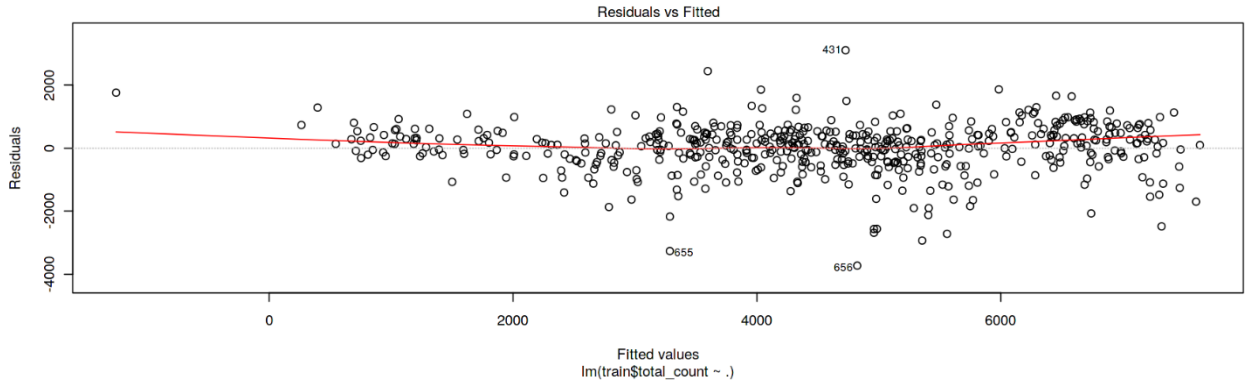

Weekday Data
Total count
0
2000
4000
6000
8000
0
1
2
3
4
5
6
weekday

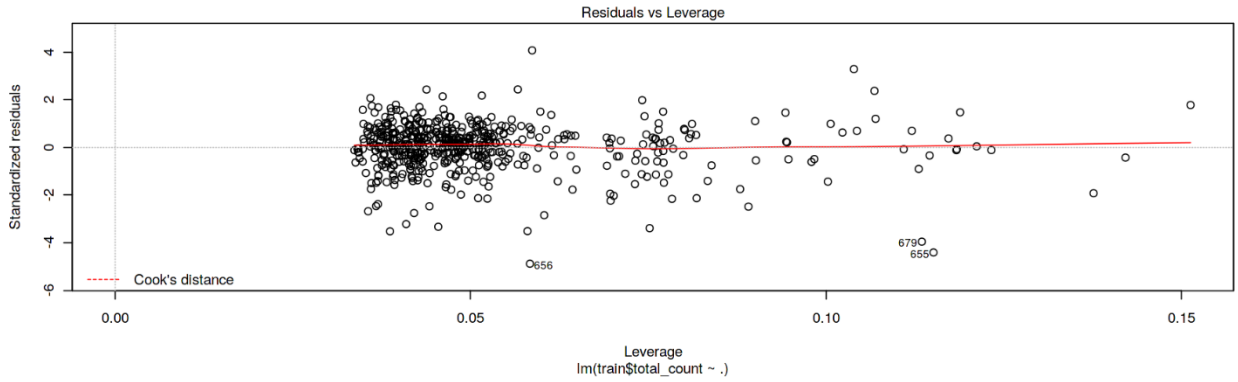

Windspeed Data
0.1
0.2
0.3
0.4
0.5
windspeed


Humidity Data
0.0
0.2
0.4
0.6
0.8
1.0
humidity


Normal Q-Q
Standardized residuals
-4
-2
0
2
4
-3
-2
-1
0
1
2
3
431
655
656
Theoretical Quantiles
lm(train$total_count ~ .)


Scale-Location
√|Standardized residuals|
Fitted values
lm(train$total_count ~ .)


Residuals vs Fitted
Residuals
Fitted values
lm(train$total_count ~ .)


Residuals vs Leverage
Standardized residuals
Cook's distance
Leverage
lm(train$total_count ~ .)

Appendix B – Complete Python and R Code

Python Code

```
# %% [markdown]
# ### **Contents**
#
#     1 Introduction
#         1.1 Problem Statement
#         1.2 Data
#     2 Methodology
#         2.1 Exploratory Data Analysis
#             2.1.1 Descriptive Analysis
#                 2.1.1.1 Features Analysis
#                 2.1.1.1 Missing Value Analysis
#                 2.1.1.3 Target Variable Analysis
#             2.1.2 Visualization
#                 2.1.2.1 Attributes Distributions and trends
#                 2.1.2.2 Outlier Analysis
#             2.2 Data Preprocessing and Analysis
#                 2.2.1 Outlier Handling
#                 2.2.2 Feature Selection
#                 2.2.3 Feature Engineering
#             2.3 Modeling
#                 2.3.1 Random Sampling
#                 2.3.2 Multilinear & Regularization Regression
#                 2.3.3 Random Forest
#                 2.3.4 Gradient Boosting
#     4 Final Model

# %% [markdown]
# # 1. Introduction
#
# The usage of bicycles as a mode of transportation has gained traction in recent
years due to with environmental and health issues. The cities across the world have
successfully rolled out bike sharing programs to encourage usage of bikes. Under such
programs, the riders can rent bicycles using manual or automated stalls spread across
the city for defined periods. In most cases, riders can pick up bikes from one
location and returned them any other designated place.
#
# The bike sharing programs from across the world are hotspots of all sorts of data,
ranging from travel time, start and end location, demographics of riders, and so on.
This data along with alternate sources of information such as weather, traffic,
terrain, season and so on.
#
#
# ## 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. The objective is to forecast bike rental
demand of Bike sharing program in Washington, D.C based on historical usage patterns
in relation with weather, environment and other data. We would be interested in
predicting the rentals on various factors including season, temperature, weather and
building a model that can successfully predict the number of rentals on relevant
factors.
#
```

```
# ## 1.2 Data
#
# This dataset contains the seasonal and weekly count of rental bikes between years
2011 and 2012 in Capital bikeshare system with the corresponding temperature and
humidity information. Bike sharing systems are a new way of traditional bike rentals.
The whole process from membership to rental and return has become automatic. The data
was generated by 500 bike-sharing programs and was collected by the Laboratory of
Artificial Intelligence and Decision Support (LIAAD), University of Porto. Given
below is the description of the data which is a (731, 16) shaped data.
#
# ### short description of features
# 1.  instant: Record index
# 1.  dteday: Date
# 1.  season: Season (1:spring, 2:summer, 3:fall, 4:winter)
# 1.  yr: Year (0: 2011, 1:2012)
# 1.  mnth: Month (1 to 12)
# 1.  holiday: weather day is holiday or not (extracted from Holiday Schedule)
# 1.  weekday: Day of the week
# 1.  workingday: If day is neither weekend nor holiday it's 1, otherwise is 0.
# 1.  weathersit: (extracted from Freemeteo)
# >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
# 1.  temp: Normalized temperature in Celsius.
# 1.  atemp: Normalized feeling temperature in Celsius.
# 1.  hum: Normalized humidity. The values are divided to 100 (max)
# 1.  windspeed: Normalized wind speed. The values are divided to 67 (max)
# 1.  casual: count of casual users
# 1.  registered: count of registered users
# 1.  cnt: count of total rental bikes including both casual and registered

# %% [code]
# Ignore  the warnings
import warnings
warnings.filterwarnings('always')
warnings.filterwarnings('ignore')

# data visualisation and manipulation
import os, sys
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import seaborn as sns # visualization
import matplotlib.pyplot as plt # visualization
import random
import pandas_profiling as pp

# configure font_scale and linewidth for seaborn
sns.set_context('paper', font_scale=1.3, rc={"lines.linewidth": 2})

# preprocessing and metrics
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error, mean_squared_log_error, make_scorer

# model selection
```

```
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV

# regresson model
from sklearn.linear_model import LinearRegression, Ridge, Lasso, ElasticNet
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor

# pipeline builder
from sklearn.pipeline import Pipeline, make_pipeline


# %% [code]
# list files in dir
print(os.listdir('../input/bike-sharing-dataset/'))

# %% [code]
# load data
bike_day = pd.read_csv('../input/bike-sharing-dataset/day.csv')
print('Shape of the data:', bike_day.shape) # shape of data
bike_day.head() # top 5 rows of data

# %% [code]
# data informations
bike_day.info()

# %% [code]
# renaming the columns
bike_day.rename(columns={'dteday':'datetime','yr':'year','mnth':'month','weathersit':
'weather_condition','hum':'humidity','cnt':'total_count'},inplace=True)

#Type casting the datetime and numerical attributes to category
bike_day['datetime']=pd.to_datetime(bike_day.datetime) # datetime conversion
bike_day['season']=bike_day.season.astype('category') # categorical conversion
bike_day['year']=bike_day.year.astype('category')
bike_day['month']=bike_day.month.astype('category')
bike_day['holiday']=bike_day.holiday.astype('category')
bike_day['weekday']=bike_day.weekday.astype('category')
bike_day['workingday']=bike_day.workingday.astype('category')
bike_day['weather_condition']=bike_day.weather_condition.astype('category')

bike_day.info() # data information after typecasting

# %% [markdown]
# # 2. Methodology
#
# ## 2.1 Exploratory Data Analysis
# In this section, we'll explore the attributes and data values. Familiarity with
data will provide more insight knowledge for data pre-processing, analysize how to
use graphical and numerical techniques to begin uncovering the structure of our data.
#
# By looking at data I came across that data is without any missing values however,
casual user variable has outliers in it. Visualizations of the bike rental count base
on the season, month, day of the week, the type of day, is it a weekday, is it a
holiday, and the type of weather, then calculating the mean of temperature, humidity,
wind speed and rental count. The purpose of this summarization is to ?nd a general
relationship between variables regardless of which year the data is from.
```

```
#
# ### 2.1.1 Descriptive Analysis

# %% [markdown]
# ### 2.1.1.1 Feature Analysis
#
# #### Generating profile report using pandas_profiling
# For each column the following statistics are presented in an interactive HTML page:
# * Essentials: type, unique values, missing values
# * Quantile statistics : minimum value, Q1, median, Q3, maximum, range,
interquartile range
# * Descriptive statistics : mean, mode, standard deviation, sum, median absolute
deviation, coefficient of variation, kurtosis, skewness
# * Most frequent values
# * Histogram
# * Correlations highlighting of highly correlated variables, Spearman and Pearson
matrixes

# %% [code]
# profile report generated in the saved repository as a html file
profile = pp.ProfileReport(bike_day)
profile.to_file("profile.html") # saving profile report as html doc
profile

# %% [markdown]
# Click profile to open profile report in new tab.
# <button><a href='./profile.html' ><b>profile</b></a></button>

# %% [code]
#diffrent value counts in each categorical features
categorical_col = ['season', 'year', 'month', 'holiday', 'weekday','workingday',
'weather_condition']
end='\n'+'*'*10+'\n' # end line seperator

for col in categorical_col:
    print(col,':\n',bike_day[col].value_counts(),end=end) # listing frequency of each
value for all of the categorical features

# %% [markdown]
# ### 2.1.1.2 Missing Value Analysis

# %% [code]
# missing value checking
print('Number of missing values:\n',bike_day.isnull().sum())

#  description
bike_day.describe()

# %% [markdown]
# ### 2.1.1.3 Sneakpeak for target variable
#
# Target variable (total_count) distribution
# > target variable is normally distributed, no skewness observed.

# %% [code]
# Distribution of target variable
```

```
_ , ax = plt.subplots(1,2, figsize=(15,5)) # 1 row 2 column subplot

sns.distplot(bike_day.total_count, bins=50, ax=ax[0]) # dependent variable
distribution plot with 50 bins
ax[0].set_title('Dist plot')

ax[1] = bike_day.total_count.plot(kind='kde') # dependent variable KDE plot
ax[1].set_title('KDE plot')


# %% [markdown]
# ### 2.1.2 Visualization
#
# It is the process of projecting the data, or parts of it, into Cartesian space or
into abstract images. With a little domain knowledge, data visualizations can be used
to express and demonstrate key relationships in plots and charts that are more
visceral to yourself and stakeholders than measures of association or significance.
In the data mining process, data exploration is leveraged in many different steps
including preprocessing, modeling, and interpretation of results.
# One of our main goals for visualizing the data here, is to observe which features
are most intiuitive in predicting target. The other, is to draw general trend, may
aid us in model selection and hyper parameter selection.
#
#
# ### 2.1.2.1 Attribute Distribution and Trends
#
# ####  Categorical features
#
#

# %% [code]
categorical_features = categorical_col
print('Categorical features :',', '.join(categorical_features))

# %% [code]
# Holiday wise yearly count of bike rental
bike_day[['season','year', 'total_count', 'holiday']].groupby([ 'year',
'holiday']).sum()

# %% [code]
# weather condition wise  count of bike rental
bike_day[['weather_condition', 'total_count']].groupby(['weather_condition']).sum()

# %% [code]
# weather condition wise  avg_count of bike rental
bike_day[['weather_condition',
'total_count']].groupby(['weather_condition']).mean().round().astype(int)

# %% [code]
# weather condition wise  avg_count of bike rental
ax = sns.barplot(x='weather_condition',y='total_count',data=bike_day, ci=None)
ax.set_title("Weather_condition: avg_count of bike rental") # set title
ax.set_xticklabels(['Clear', 'Mist', 'Snow']) # set x-tick labels
plt.show()

# %% [code]
```

```
f, (ax1, ax2)  =  plt.subplots(nrows=1, ncols=2, figsize=(13, 6)) # 1 row 2 column
subplot

# Counts of Bike Rentals by season
ax1 =
bike_day[['season','total_count']].groupby(['season']).sum().reset_index().plot(kind=
'bar',legend = False, title ="Counts of Bike Rentals by season", stacked=True,
fontsize=12, ax=ax1)
ax1.set_xlabel("season", fontsize=12)  # set x-axis labels
ax1.set_ylabel("Count", fontsize=12)  # set y-axis labels
ax1.set_xticklabels(['spring','summer','fall','winter'])  # set x-tick labels

# Counts of Bike Rentals by weather condition
ax2 =
bike_day[['weather_condition','total_count']].groupby(['weather_condition']).sum().re
set_index().plot(kind='bar', legend = False, stacked=True, title ="Counts of Bike
Rentals by weather condition", fontsize=12, ax=ax2)
ax2.set_xlabel("weather_condition", fontsize=12)  # set x-axis labels
ax2.set_ylabel("Count", fontsize=12)  # set y-axis labels
ax2.set_xticklabels(['1: Clear','2: Mist','3: Light Snow'])  # set x-tick labels

f.tight_layout()

# %% [code]
# Count of bike on workingdays/holidays for each season
bike_day[['season','year', 'total_count', 'holiday']].groupby(['season',
'holiday']).sum().plot(kind='bar') # plotting bar graph
plt.title('Count of bike on workingdays/holidays for each season') # set title

# %% [code]
fig, (axs1,axs2,axs3) = plt.subplots(ncols=3, figsize=(18,5))  # 1 row 3 column
subplot

# Total Count of bike for Each year
sns.barplot(x='year', y='total_count', data=bike_day, hue='season', ax=axs1, ci=None)
# barplot
axs1.set_title('Total Count of bike for Each year') # set tilte
axs1.legend(['spring','summer','fall','winter']) # set legend
axs1.set_xticklabels(['2011', '2012']) # set x-tick label

# Casual Count of bike for Each year
sns.barplot(x='year', y='casual', data=bike_day, hue='season', ax=axs2, ci=None) #
barplot
axs2.set_title('Casual Count of bike for Each year') # set title
axs2.legend(['spring','summer','fall','winter']) # set legend
axs2.set_xticklabels(['2011', '2012']) # set x-tick label

# Registered Count of bike for Each year
sns.barplot(x='year', y='registered', data=bike_day, hue='season', ax=axs3, ci=None)
# barplot
axs3.set_title('Registered Count of bike for Each year') # set title
axs3.legend(['spring','summer','fall','winter']) # set legend
axs3.set_xticklabels(['2011', '2012']) # set x-tick label
```

```
# %% [code]
fig, (axs1,axs2,axs3) = plt.subplots(ncols=3, figsize=(18,5)) # 1 row 3 column
subplot

# Total Count of bike, season wise
sns.barplot(x='season', y='total_count', data=bike_day, hue='weather_condition',
ax=axs1, ci=None) # barplot
axs1.set_title('Total Count of bike, season wise') # set title
axs1.set_xticklabels(['spring','summer','fall','winter']) # set x-tick label
axs1.legend(labels=['Clean', 'Mist', 'Snow']) # set legend

# Casual Count of bike, season wise
sns.barplot(x='season', y='casual', data=bike_day, hue='weather_condition', ax=axs2,
ci=None) # barplot
axs2.set_title('Casual Count of bike, season wise') # set title
axs2.set_xticklabels(['spring','summer','fall','winter']) # set x-tick label
axs2.legend(labels=['Clean', 'Mist', 'Snow']) # set legend

# Registered Count of bike, season wise
sns.barplot(x='season', y='registered', data=bike_day, hue='weather_condition',
ax=axs3, ci=None) # barplot
axs3.set_title('Registered Count of bike, season wise') # set title
axs3.set_xticklabels(['spring','summer','fall','winter']) # set x-tick label
axs3.legend(labels=['Clean', 'Mist', 'Snow']) # set legend


# %% [code]
# Season-wise weekday bike count
plt.figure(figsize=(15,8)) # set figure size

sns.barplot(x='season', y='total_count', data=bike_day, hue='weekday', ci=None) #
barplot
plt.legend(['Monday', 'Tuesday', 'Wednesday', 'Thrusday', 'Friday', 'Saturday',
'Sunday']) # set legend
plt.title('# Season-wise weekday bike count') # set title
plt.xticks(np.arange(4),['spring','summer','fall','winter']) # set x-tick label

# %% [code]
# Season-wise holiday bike count
plt.figure(figsize=(10,8)) # set figure size

sns.barplot(x='season', y='total_count', data=bike_day, hue='holiday', ci=None) #
barplot
plt.legend(title='Day',labels= ['Holiday', 'Workday']) # set legend
plt.title('Season-wise holiday bike count') # set title
plt.xticks(np.arange(4),['spring','summer','fall','winter']) # set x-tick label


# %% [code]
#Consistency of Bike count on weekdays by monthly basis
plt.figure(figsize=(18,8)) # set figure size

sns.pointplot(x='month', y='total_count', data=bike_day, hue='weekday', ci=None) #
pointplot
plt.legend(['Monday', 'Tuesday', 'Wednesday', 'Thrusday', 'Friday', 'Saturday',
'Sunday']) # set legend
```

```
plt.title('Bike count on weekdays by monthly basis') # set title

# %% [code]
_, ax=plt.subplots(nrows=2, ncols=1, figsize=(15,10))

# Total count: season wise of weekdays
sns.pointplot(x='season', y='total_count', data=bike_day, hue='weekday', ci=None,
ax=ax[0]) # pointplot
ax[0].set_title('Total count: season wise of weekdays') # set title
ax[0].legend(['Monday', 'Tuesday', 'Wednesday', 'Thrusday', 'Friday', 'Saturday',
'Sunday']) # set legend
ax[0].set_xticklabels(['spring','summer','fall','winter']) # set x-tick label

# Total count: season wise of both year
sns.pointplot(x='season', y='total_count', data=bike_day, hue='year', ci=None,
ax=ax[1]) # pointplot
ax[1].set_title('Total count: season wise of both year') # set title
ax[1].legend(['2011', '2012']) # set legend
ax[1].set_xticklabels(['spring','summer','fall','winter']) # set x-tick label


# %% [markdown]
# <b>Extract weekend column as a feature by using datetime feature.

# %% [code]
# Generally, 1-Monday and 0-Sunday in datetime , for weekend: Saturday-6 & Sunday-0
# creating new feature 'isweekend' using datetime feature and computing is that date
is fall over the weekend
bike_day['isweekend']=bike_day['datetime'].apply( lambda x :1 if (x.weekday()==0)
|(x.weekday()==6) else 0 ) # for weekday = 0 or 6, isweekend = 1 else 0
bike_day[bike_day['isweekend']==1]['weekday'].value_counts() # number of weekend days

# %% [code]
# Days wise average causal and registered bike count

casual_avg = bike_day.groupby(['weekday'])['casual'].mean().round().astype(int) #
average casual rental on weekday
registered_avg =
bike_day.groupby(['weekday'])['registered'].mean().round().astype(int) # average
casual rental on weekday

print('Total count: casual {}, registered {}'.format(casual_avg.sum(),
registered_avg.sum()))

# %% [code]
# Avg Use of the bikes by casual users on weekdays
_, ax=plt.subplots(nrows=2, ncols=1, figsize=(15,10)) # 2 row 1 col subplot
sns.pointplot(x='weekday', y='casual', data=bike_day, ci=None, ax=ax[0]) # pointplot
ax[0].set(title="Avg Use of the bikes by casual users") # set title
ax[0].set_xticklabels(['Monday', 'Tuesday', 'Wednesday', 'Thrusday', 'Friday',
'Saturday', 'Sunday']) # set x-tick label

for c in ax[0].collections:
    for val,of in zip(casual_avg,c.get_offsets()):
        ax[0].annotate(val, of)                          # set annotations for average
of each weekday for  casual rentals
```

```
# Avg Use of the bikes by registered users on weekday
sns.pointplot(x='weekday', y='registered', data=bike_day, ci=None, ax=ax[1]) # set
pointplot
ax[1].set(title="Avg Use of the bikes by registered users") # set title
ax[1].set_xticklabels(['Monday', 'Tuesday', 'Wednesday', 'Thrusday', 'Friday',
'Saturday', 'Sunday']) # set x-tick label

for c in ax[1].collections:
    for val,of in zip(registered_avg,c.get_offsets()):
        ax[1].annotate(val, of)                          # set annotations for
average of each weekday for registered rentals


# %% [markdown]
# #### Observation: all Categorical features
# - people like to rent bikes more when the sky is clear.
# - the count of number of rented bikes is maximum in fall (Autumn) season and least
in spring season.
# - number of bikes rented per season over the years has increased for both casual
and registered users.
# - registered users have rented more bikes than casual users overall.
# - casual users travel more over weekends as compared to registered users (Saturday
/ Sunday).
# - registered users rent more bikes during working days as expected for commute to
work / office.
# - demand for bikes are more on working days as compared to holidays ( because
majority of the bike users are registered )
#

# %% [markdown]
# ####   Continuous features
#
# <b> A. Pairplot

# %% [code]
# Columns present in dataset after feature adding
bike_day.columns

# %% [code]
# Visualization of continuous variable varition and thier co-relation
ax = sns.pairplot(bike_day[['temp', 'atemp', 'windspeed', 'humidity']] ) # pairplot
ax.fig.suptitle('Continuous variable varition and thier co-relation',  y=1.0) # set
title

# %% [markdown]
# <b> B. Regression plot
#
# > <b> Regression plot of seaborn used to depict the relationship between continous
features and target variable.</b>
#

# %% [code]
# Regresson plots between temp, windspeed and humidity against total_count
_ , ax = plt.subplots(1,3, figsize=(18,5)) # 1 row 3 column subplot
```

```
sns.regplot(x = 'temp', y='total_count', data=bike_day, ax= ax[0]) # Regression plot
ax[0].set_title('+ve relation between temp and total_count') # set title

sns.regplot(x = 'windspeed', y='total_count', data=bike_day, ax= ax[1]) # Regression
plot
ax[1].set_title('-ve relation between windspeed and total_count') # set title

sns.regplot(x = 'humidity', y='total_count', data=bike_day, ax= ax[2]) # Regression
plot
ax[2].set_title('+ve relation between humidity and total_count') # set title

# %% [markdown]
# #### Observation
# Here we considered "count" vs "temp", "humidity", "windspeed".
# - A +ve correlation with temperature was observed ( sky is clear with increase in
temperature )
# - A -ve correlation with humidity and windspeed was observed as people avoid
travelling when weather is very windy or humid.

# %% [code]
# Regresson plots between temp, windspeed and humidity to show thier relation with
each other
_ , ax = plt.subplots(1,3, figsize=(18,5)) # 1 row 3 column subplots

sns.regplot(x = 'temp', y='humidity', data=bike_day, ax= ax[0])# Regression plot
ax[0].set_title('+ve relation between temp and humidity') # set title

sns.regplot(x = 'windspeed', y='humidity', data=bike_day, ax= ax[1])# Regression plot
ax[1].set_title('-ve relation between windspeed and humidity') # set title

sns.regplot(x = 'temp', y='windspeed', data=bike_day, ax= ax[2])# Regression plot
ax[2].set_title('+ve relation between temp and windspeed') # set title

# %% [markdown]
# #### Observation
#
# * A +ve correlation between humidity and temperature was observed (as temp
increases the amount of water vapour present in the air also increases)
# * A -ve correlation between windspeed with humidity and temperature was observed
(as wind increases, it draws heat from the body, thereby temperature and humidity
decreases)

# %% [markdown]
# <b> C. Correlation | Heatmap </b>

# %% [code]
# Calculate Co-variance of new data
bike_corr = bike_day.corr()

# Create mask for upper triangle of co-var matrix

mask1 = np.array(bike_corr)
mask1[np.tril_indices_from(mask1)] = False # setting upper triangle show to False

# %% [code]
# heatmap of continous variables
```

```
_, ax = plt.subplots(1,1, figsize=(15,12)) # 2 row 1 column subplot
sns.heatmap(bike_corr, mask=mask1, annot=True, square=False, ax=ax) # heatmap
ax.set_title('Corelation of Continuos variables') # set title

plt.tight_layout()

# %% [markdown]
# ### Observation: continuous features¶
# * Temp, Atemp looks normally distributed.
# * A strong corelation can be seen for temp and atemp.
# * windspeed, humidity, temp and atemp are all normalised in the dataset already.
# * With increase in temperature, the count of bike rentals increases as shown in reg
plot.

# %% [markdown]
# ### 2.1.2.2  Outlier Analysis: Box Plots, Pair Plots

# %% [code]
# Outlier Analysis
fig, axes = plt.subplots(nrows=4,ncols=2) # 4 row 2 column subplots
fig.set_size_inches(20, 16) # set figure size
plt.subplots_adjust(hspace=0.3) # set hspace to avoid overlapping

# boxplots for categorical and continuous features
sns.boxplot(data=bike_day,y="total_count", ax=axes[0][0])
sns.boxplot(data=bike_day,y="total_count",x="season", ax=axes[0][1])
sns.boxplot(data=bike_day,y="total_count",x="year", ax=axes[1][0])
sns.boxplot(data=bike_day,y="total_count",x="weather_condition", ax=axes[1][1])
sns.boxplot(data=bike_day,y="total_count",x="month", ax=axes[2][0])
sns.boxplot(data=bike_day,y="total_count",x="weekday", ax=axes[2][1])
sns.boxplot(data=bike_day,x="windspeed", ax=axes[3][0])
sns.boxplot(data=bike_day,x="humidity", ax=axes[3][1])


axes[0][0].set(ylabel='Count',title="Box Plot On Count")
axes[0][1].set(ylabel='Count',title="Box Plot On Count Across Season")
axes[0][1].set_xticklabels(['spring','summer','fall','winter'])

axes[1][0].set(ylabel='Count',title="Box Plot On Count Across year")
axes[1][0].set_xticklabels(['2011','2012'])

axes[1][1].set(ylabel='Count',title="Box Plot On Count Across weather")
axes[1][1].set_xticklabels(['Clean','Mist','Snow'])

axes[2][0].set(ylabel='Count',title="Box Plot On Count Across month")
axes[2][1].set(ylabel='Count',title="Box Plot On Count Across weekday")
axes[2][1].set_xticklabels(['Monday', 'Tuesday', 'Wednesday', 'Thrusday', 'Friday',
'Saturday', 'Sunday'])

axes[3][0].set(ylabel='windspeed',title="Box Plot On windspeed")
axes[3][1].set(ylabel='humidity',title="Box Plot On humidity")


# %% [code]
# Pairplot for outlier analysis
```

```
ax =
sns.pairplot(data=bike_day[['humidity','windspeed','temp','total_count']],palette='hl
s')
ax.fig.suptitle('Pairplot for outlier analysis', y=1.0)

# %% [markdown]
# ### Observation:
#
# - temp has got positive correlation with count as people like to travel more when
the sky is clear.
# - humidity is inversely related to count as expected as when weather is humid
people will not like to travel on a bike.
# - windspeed is also having a negative correlation with "count".
# - "atemp" and "temp" variable 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.
# - "weather_condition" and count are inversely related. This is because for our data
as weather increases from (1 to 4) implies that weather is getting more worse and so
lesser people will rent bikes.
# - "registered" and count are highly related which indicates that most of the bikes
that are rented are registered.
# - "Casual" and "Registered" are also not taken into account since they are leakage
variables in nature and need to dropped during model building to avoid bias. (casual
+ registered = count)
# - "instant" variable can also be dropped during model building as it indicates
index.
#

# %% [markdown]
# ### 2.2 Data preprocessing
#
# ### 2.2.1 Outlier handling

# %% [code]
bike_day.head()

# %% [code]
# finding outliers
wind_humidity = pd.DataFrame(bike_day, columns=['windspeed', 'humidity'])

# get outliers for windspeed and humidity features
for i in ['windspeed', 'humidity']:
    q75, q25 = np.percentile(wind_humidity.loc[:,i], [75,25]) # get q75 and q25
    IQR = q75 - q25 # calculate  IQR for boxplot outlier method
    max = q75+(IQR*1.5) # get max bound
    min = q25-(IQR*1.5) # get min bound
    wind_humidity.loc[wind_humidity.loc[:,i]<min,:i] = np.nan # replacing outliers
with NAN
    wind_humidity.loc[wind_humidity.loc[:,i]>max,:i] = np.nan # replacing outliers
with NAN

print('Shape after dropping outlier
(windspeed,humidity):',wind_humidity.dropna().shape)
print('Shape before dropping outlier
(windspeed,humidity):',bike_day[['windspeed','humidity']].shape)
```

```
# %% [code]
# calculating outlier indexs
index=[]
outlier = pd.DataFrame()
for i in range(wind_humidity.shape[0]):
    if wind_humidity.loc[i,].isna().any(): # if either of windspeed or humidity is
NAN, for each column
        outlier.loc[i,'outlier'] = 1 # store index as outlier 1
        index.append(i) # store indices of outliers
    else:
        outlier.loc[i,'outlier'] = 0

wind_humidity['outlier'] = outlier['outlier'].astype(int) # convert outlier column as
integer
wind_humidity.loc[index,] # show outliers with thier respective indices

# %% [code]
bike_day['outlier'] = wind_humidity['outlier'] # add oulier feature in bike data

#dropping all the outliers present in dataframe
bike_day.drop(bike_day[(bike_day.outlier==1) ].index, inplace=True) # dropping all
the outliers
print('Shape after dropping outlier:',bike_day.shape) # shape of the after removing
outlier rows
print(bike_day.info())

# %% [code]
# Visualization after removing outliers
ax =
sns.pairplot(data=bike_day[['humidity','windspeed','temp','total_count']],palette='hl
s') # pairplot of continuous features
ax.fig.suptitle('Pairplot after removing outliers', y=1.0)

# %% [markdown]
# ### 2.2.2 Feature Selection

# %% [code]
# categorising features
categorical_features =
["season","holiday","weather_condition","weekday","month","year",'isweekend','working
day']
continous_features = ["temp","humidity","windspeed"]
dropFeatures = ['casual',"datetime","instant","registered","atemp","outlier"]
target=['total_count']

# %% [code]
# drop unwanted features
bike_FE = bike_day.drop(dropFeatures, axis=1)
bike_FE.columns

# %% [markdown]
# ### 2.2.3 Feature Engineering
# Converting categorical features to numercial features to feed our models using
<b>"pd.get_dummies()"</b>.

# %% [code]
```

```
# create dummy data
dummy_data = bike_FE.copy()

# fucntion for creating dummy features
def get_dummy(df, col):
    df = pd.concat([df, pd.get_dummies(df[col], prefix=col, drop_first=True)],
axis=1) # create dummy features and dropping first feature, since it's redundant
    df = df.drop([col], axis = 1)
# drop feature of which dummy is created
    return df
# return dummy dataframe

# features to create dummy
# get_dummy_features = ["season","weather_condition","weekday","month"]
get_dummy_features = categorical_features

# create dummy for features
for col in get_dummy_features:
    dummy_data = get_dummy(dummy_data, col) # create dummy for all categorical
features

dummy_data.head()

# %% [code]
dummy_data.info()

# %% [markdown]
# # 2.3 Modeling
#
# ### 2.3.1 Sampling
#
# <b> Splitting data</b> in train and test in 75% and 25% of total data respectively

# %% [code]
# splitting data in test and train set
X_train, X_test, y_train, y_test = train_test_split(dummy_data.drop(['total_count'],
axis=1),dummy_data.total_count,test_size=0.25,random_state=14)

print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)

# %% [code]
# function: get metrics
def metrics(regressor_name, regressor, y_pred):
    """Print metrics: r2, adj r2, rmse, rmsle
    parameters:
        regressor_name: list, dataframe, matrix
        regressor: fitted model object
        y_pred: list, dataframe, matrix
    """
    print(regressor_name)                          # print regressor name
    print('R^2:',regressor.score(X_test, y_test))  # Returns the coefficient of
determination R^2 of the prediction.
    print('Adj R^2: ', 1 - (1-regressor.score(X_test, y_test))*(len(y_test)-
1)/(len(y_test)-X_test.shape[1]-1)) # Returns the coefficient of determination Adj
R^2 of the prediction.
```

```
    print('RMSE {}: {}'.format(regressor_name,  np.sqrt(mean_squared_error(y_test,
y_pred)))) # Retrun rmse score
    print('RMSLE {}: {}'.format(regressor_name,
np.sqrt(mean_squared_log_error(y_test, y_pred)))) # Retrun rmsle score

# function: plot scores
def plot_score(grid_cv, name):
    score = pd.DataFrame(grid_cv.cv_results_) # create dataframe of the cv results
    score['alpha'] = score['param_model__alpha'] # rename parameter columns
    score['r2'] = score['mean_test_score']
    sns.pointplot(data=score, x='alpha', y='r2',
color=random.choice(['r','b','g','y']))
    plt.title(name)
    plt.grid(True)
    plt.show()

# function: plot residuals graph
def plotResiduals(y_test, y_pred, name):
    residuals = y_test - y_pred                    # get residuals
    _, ax = plt.subplots()                         # create subplot
    ax.scatter(y_test, residuals)                  # scatter plot for actuals and
residuals
    ax.axhline()                                   # create reg line
    ax.set_xlabel('Observed')
    ax.set_ylabel('Residuals')
    ax.title.set_text(name[:15]+ ' Residual Plot | RSME: ' +
str(np.sqrt(mean_squared_error(y_test, y_pred))))
    plt.show()

# function: plot distribution graph
def distPlot(y_pred, y_test, name):
    plt.figure(figsize=(10,5))
    sns.distplot(y_pred)
    sns.distplot(y_test)
    plt.legend(['y_pred','y_test'])
    plt.title(name+'|comparison of prediction distribution')
    plt.show()

# function: simple regressor
def simpleRegressor(model):
    """Simple Regressor function for each model given in model variable by creating
pipeline with StandardScale() and fit data to predict.
        model: dict()

    """
    for regressor_name, regressor in linear_models.items():
        pipeline = Pipeline( [('scaler', StandardScaler()), ('model',regressor)] ) #
create pipeline of each model with StandardScale
        regressor = pipeline.fit(X_train, y_train)                                  #
fit data to the pipeline
        y_pred = regressor.predict(X_test)                                          #
Predict using the pipeline model

        metrics(regressor_name, regressor, y_pred)                                  #
get metrics : r2, adj r2, rmse, rmsle
        print("\n")
```

```
# function: simple ensemble regressor
def ensembleRegressor(model):
    """Ensemble Regressor function for each model given in model variable by creating
pipeline with StandardScale() and fit data to predict.
        model: dict()

    """
    if type(model) == dict:
        for regressor_name_, regressor_ in model.items():
            pipeline = Pipeline( [('scaler', StandardScaler()),
('model',regressor_)] ) # create pipeline of each model with StandardScale
            regressor_ = pipeline.fit(X_train, y_train)
# fit data to the pipeline
            y_pred = regressor_.predict(X_test)
# Predict using the pipeline model
            metrics(regressor_name_,regressor_, y_pred )
# get metrics : r2, adj r2, rmse, rmsle
            plotResiduals(y_test, y_pred, regressor_name_)
# plot residual graphs


# function: Compare Algorithms
def comparePlot(results, names):
    fig = plt.figure()
    fig.suptitle( 'Algorithm Comparison' )
    ax = fig.add_subplot(111)
    plt.scatter(names, results)
    ax.set_xticklabels(names)
    plt.show()

# function: ensemble regressor with hyper-parameter tunning
def gridSearchRegressor(model, param_grid, scoring = 'r2', plot_score_ = False,
compare_score = False):
    """GridSearchRegressor function for each model given in model variable by
creating pipeline with StandardScale() and fit data to predict.
        model: dict()
        scoring: r2 default
        plot_score_ = optional False default, for plotting socre
        compare_score = optional False default, for comparing the model scores
    """
    best_score = []
    name = []

    for regressor_name, regressor in model.items():
        pipeline = Pipeline( [('scaler', StandardScaler()),  ('model',regressor)] ) #
create pipeline of each model with StandardScale
        grid_cv = GridSearchCV(estimator=pipeline, param_grid=param_grid, scoring =
scoring, cv = 5) # estimate best parameter using gridSearchCV()
        grid_cv.fit(X_train, y_train) #  # fit data to the pipeline

        name.append(regressor_name)
        best_score.append(grid_cv.best_score_) # append best score from gridSearch

        print(regressor_name)
        print('Best score:',grid_cv.best_score_) # print best score from gridSearch
```

```
        print('Best param:',grid_cv.best_params_) # print best parameter from
gridSearch

        best_grid = grid_cv.best_estimator_ # get best score from gridSearch best
estimator
        y_pred=best_grid.predict(X_test) # predict using best estimator

        metrics(regressor_name,best_grid, y_pred )  # get metrics : r2, adj r2, rmse,
rmsle
        plotResiduals(y_test, y_pred, regressor_name)
        distPlot(y_pred, y_test, name=regressor_name)
        if plot_score_:
            plot_score(grid_cv, regressor_name) # plot score
    if compare_score:
        comparePlot(best_score, name)    # plot comparison


# %% [markdown]
# ### 2.3.2 MultiLinear & Regularization model: LinearRegression| Ridge| Lasso|
ElasticNet

# %% [code]
# liner models
linear_models = {'LinearRegression':LinearRegression(), 'Ridge':Ridge(),
'Lasso':Lasso(), 'ElasticNet':ElasticNet()}

# Score without StandardScale
print('Score without StandardScale:\n')
for regressor_name, regressor in linear_models.items():
    regressor.fit(X_train, y_train) # fit data to models
    y_pred = regressor.predict(X_test) # predict using models

    metrics(regressor_name, regressor, y_pred)
    print("\n")

# Score without StandardScale:

# LinearRegression
# R^2: 0.8503150391570842
# Adj R^2:  0.8213759467274537
# RMSE LinearRegression: 720.8842925663198
# RMSLE LinearRegression: 0.2465516805375247


# Ridge
# R^2: 0.8478419303061329
# Adj R^2:  0.8184247034986518
# RMSE Ridge: 726.8151540447384
# RMSLE Ridge: 0.2381610980768202


# Lasso
# R^2: 0.8504890420887281
# Adj R^2:  0.8215835902258822
# RMSE Lasso: 720.4651707841829
# RMSLE Lasso: 0.23878836957603555
```

```
# ElasticNet
# R^2: 0.379145172923085
# Adj R^2:  0.25911323968821476
# RMSE ElasticNet: 1468.1536519170734
# RMSLE ElasticNet: 0.46300107772627347



# Score with StandardScale
print('Score with StandardScale:\n')
simpleRegressor(linear_models)



# Score with StandardScale:

# LinearRegression
# R^2: 0.8503150391570844
# Adj R^2:  0.8213759467274541
# RMSE LinearRegression: 720.8842925663195
# RMSLE LinearRegression: 0.24655168053752335


# Ridge
# R^2: 0.8502971862653109
# Adj R^2:  0.8213546422766044
# RMSE Ridge: 720.9272811386522
# RMSLE Ridge: 0.24640128765416605


# Lasso
# R^2: 0.85043352042173
# Adj R^2:  0.8215173343699311
# RMSE Lasso: 720.5989325995707
# RMSLE Lasso: 0.24569841589499636


# ElasticNet
# R^2: 0.7690031028056619
# Adj R^2:  0.7243437026814232
# RMSE ElasticNet: 895.528789006083
# RMSLE ElasticNet: 0.2952233398791348

# %% [markdown]
# ### Regularizarion models with hyper_tunning parameter:  Ridge| Lasso| ElasticNet

# %% [code]

# linear regularization model with hypertuning parameters
regularization_linear_models = { 'Ridge':Ridge(), 'Lasso':Lasso(),
'ElasticNet':ElasticNet()}

# Regularization hyper-parameter tunning with GridSearchCV
print('Score with GridSearchCV:\n')
```

```
param_grid1 = {'model__alpha' : [0.1, 1, 2, 3, 4, 5, 10, 30, 50, 80,
100],'model__max_iter':[3000] } # parameters for feeding in gridSearch
gridSearchRegressor(regularization_linear_models, param_grid=param_grid1,
plot_score_=True, compare_score=True )


# Score with GridSearchCV:

# Ridge
# Best score: 0.8223825219791182
# Best param: {'model__alpha': 5, 'model__max_iter': 3000}
# Ridge
# R^2: 0.8501161318996594
# Adj R^2:  0.8211385840669269
# RMSE Ridge: 721.3631032400775
# RMSLE Ridge: 0.24571028154769625

# Lasso
# Best score: 0.8220643248274393
# Best param: {'model__alpha': 3, 'model__max_iter': 3000}
# Lasso
# R^2: 0.8505544480598202
# Adj R^2:  0.8215178229750227
# RMSE Lasso: 720.3075640121864
# RMSLE Lasso: 0.24420533245983528
# Best param: {'model__alpha': 3, 'model__max_iter': 3000}

# ElasticNet
# Best score: 0.8209891999677044
# Best param: {'model__alpha': 0.1, 'model__max_iter': 3000}
# ElasticNet
# R^2: 0.8473885489959245
# Adj R^2:  0.7243433797638379
# RMSE ElasticNet: 727.8971844088097
# RMSLE ElasticNet: 0.24186048423041664
# Best param: {'model__alpha': 0.1, 'model__max_iter': 3000}


# %% [markdown]
# ### 2.3.3 Ensemble model:  RandomForestRegressor

# %% [code]
# Simple RandomForestRegressor
ensemble_model1 = {'RandomForestRegressor':RandomForestRegressor(random_state=867)}
ensembleRegressor(ensemble_model1)

# RandomForestRegressor
# R^2: 0.8649320139183166
# Adj R^2:  0.8388188699425245
# RMSE RandomForestRegressor: 684.7825594871541
# RMSLE RandomForestRegressor: 0.2333816742631807

# %% [code]
# Hyper-parameter tunning: RandomForestRegressor

param_grid = {
```

```
    'model__n_estimators' : [10,900],
    'model__max_depth': [5,6,7,10],
    'model__max_features' : ['log2','sqrt','auto'],
    'model__random_state': [897]
}

ensemble_model = {'RandomForestRegressor':RandomForestRegressor(random_state=867) }
gridSearchRegressor(ensemble_model,param_grid=param_grid )

# RandomForestRegressor
# Best score: 0.8587517092660517
# Best param: {'model__max_depth': 10, 'model__max_features': 'auto',
'model__n_estimators': 900, 'model__random_state': 897}
# RandomForestRegressor
# R^2: 0.8715849042781012
# Adj R^2:  0.8467579857718674
# RMSE RandomForestRegressor: 667.7048312259761
# RMSLE RandomForestRegressor: 0.2349906934957484

# %% [markdown]
# ### 2.3.4 Boosting model: GradientBoostingRegressor

# %% [code]
# Simple GradientBoostingRegressor
ensemble_model2 =
{'GradientBoostingRegressor':GradientBoostingRegressor(random_state=867)}
ensembleRegressor(ensemble_model2)

# GradientBoostingRegressor
# R^2: 0.8862719523839226
# Adj R^2:  0.8642845298448143
# RMSE GradientBoostingRegressor: 628.3625167761045
# RMSLE GradientBoostingRegressor: 0.19722626748641778

# %% [code]
# Hyper-parameter tunning: GradientBoostingRegressor

hyper_param = {
                    'model__n_estimators' : [250,400,500,650,800], # The number of
boosting stages to perform.
                    'model__max_depth' : [5,6,7,8], # maximum depth of the individual
regression estimators
                    'model__max_features' : ['log2','sqrt','auto'], # The number of
features to consider when looking for the best split:
                    'model__subsample' : [0.7,0.85,0.9], # The fraction of samples to
be used for fitting the individual base learners.  If smaller than 1.0 this results
in Stochastic Gradient Boosting.
                    'model__random_state': [17]

                   }

gbm_ensemble_model = {'GradientBoostingRegressor':GradientBoostingRegressor() }
gridSearchRegressor(gbm_ensemble_model,hyper_param )

# GradientBoostingRegressor
# Best score: 0.8869327747143647
```

```
# Best param: {'model__max_depth': 5, 'model__max_features': 'sqrt',
'model__n_estimators': 250, 'model__random_state': 17, 'model__subsample': 0.7}
# GradientBoostingRegressor
# R^2: 0.8870138124626565
# Adj R^2:  0.8651698162054368
# RMSE GradientBoostingRegressor: 626.3097260903672
# RMSLE GradientBoostingRegressor: 0.2143924133958078

# %% [markdown]
# # 4. Final Model
#
# ## Model: GradientBoostingRegressor

# %% [code]
# Final model: Tuned GradientBoostingRegressor

# parameters
params = {'max_depth': 5,
          'max_features': 'sqrt',
          'n_estimators': 250,
          'random_state': 147,
          'subsample': 0.85
         }
# regressor
regressor_name = 'GradientBoostingRegressor'

# pipeline
pipeline = Pipeline( [('scaler', StandardScaler()),
('model',GradientBoostingRegressor(**params))] ) # create pipeline
pipeline.fit(X_train, y_train)  # fit data to the pipeline
gbm_y_pred = pipeline.predict(X_test) # make prediction using pipeline

# metrics and plots
print(regressor_name)
metrics(regressor_name, pipeline, gbm_y_pred ) #
plotResiduals(y_test, gbm_y_pred, regressor_name)
distPlot(gbm_y_pred, y_test, name=regressor_name)



# GradientBoostingRegressor
# R^2: 0.8819155688462875
# Adj R^2:  0.8590859121565698
# RMSE GradientBoostingRegressor: 640.28422142774
# RMSLE GradientBoostingRegressor: 0.21093760709288065

# %% [markdown]
#  ##  Submission

# %% [code]
# Final submission
bikeTestPred = pd.DataFrame()
bikeTestPred['y_test'] = y_test
bikeTestPred['gbm_y_pred'] = gbm_y_pred
bikeTestPred['gbm_y_pred'] = bikeTestPred['gbm_y_pred'].astype(int)
```

```
bikeTestPred.to_csv('Bike_Renting_Python.csv')
bikeTestPred

# %% [markdown]
# Download CSV for Test result:
# <a  href="Bike_Renting_Python.csv" target="_blank">download Bike_Renting_Python</a>
#
```

R Code

```
# %% [markdown]
# # 1. Introduction
#
# The usage of bicycles as a mode of transportation has gained traction in recent
years due to with environmental and health issues. The cities across the world have
successfully rolled out bike sharing programs to encourage usage of bikes. Under such
programs, the riders can rent bicycles using manual or automated stalls spread across
the city for defined periods. In most cases, riders can pick up bikes from one
location and returned them any other designated place.
#
# The bike sharing programs from across the world are hotspots of all sorts of data,
ranging from travel time, start and end location, demographics of riders, and so on.
This data along with alternate sources of information such as weather, traffic,
terrain, season and so on.
#
#
# ## 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. The objective is to forecast bike rental
demand of Bike sharing program in Washington, D.C based on historical usage patterns
in relation with weather, environment and other data. We would be interested in
predicting the rentals on various factors including season, temperature, weather and
building a model that can successfully predict the number of rentals on relevant
factors.
#
# ## 1.2 Data
#
# This dataset contains the seasonal and weekly count of rental bikes between years
2011 and 2012 in Capital bikeshare system with the corresponding temperature and
humidity information. Bike sharing systems are a new way of traditional bike rentals.
The wohle process from memberhsip to rental and retrun back has become automatic. The
data was generated by 500 bike-sharing programs and was collected by the Laboratory
of Arti?cial Intelligence and Decision Support (LIAAD), University of Porto. Given
below is the description of the data which is a (731, 16) shaped data.
```

```
#
# ### short description of features
# 1.  instant: Record index
# 1.  dteday: Date
# 1.  season: Season (1:spring, 2:summer, 3:fall, 4:winter)
# 1.  yr: Year (0: 2011, 1:2012)
# 1.  mnth: Month (1 to 12)
# 1.  holiday: weather day is holiday or not (extracted from Holiday Schedule)
# 1.  weekday: Day of the week
# 1.  workingday: If day is neither weekend nor holiday it's 1, otherwise is 0.
# 1.  weathersit: (extracted from Freemeteo)
# >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
# 1.  temp: Normalized temperature in Celsius.
# 1.  atemp: Normalized feeling temperature in Celsius.
# 1.  hum: Normalized humidity. The values are divided to 100 (max)
# 1.  windspeed: Normalized wind speed. The values are divided to 67 (max)
# 1.  casual: count of casual users
# 1.  registered: count of registered users
# 1.  cnt: count of total rental bikes including both casual and registered

# %% [code] {"_execution_state":"idle"}
## Importing packages

library(tidyverse) # metapackage with lots of helpful functions
library(gridExtra) # for creating subplots
library(DMwR) # for reg.eval()
library(ggpubr)
library(glmnet) # linear models
library(Metrics) # error metrics
library(randomForest) # ensemble model
library(GGally) # pairplot
library(mlbench)
library(caret)
library(gbm) # gradient boost model

# set image size
options(repr.plot.width=15, repr.plot.height=5)

# ignore warnings
options(warn=-1)

# list available files/folder
list.files(path = "../input/")


# %% [code]
# load data
bike_day = read.csv('../input//bike-sharing-dataset/day.csv')


# %% [code]
# taking sneak peak to datasets
```

```
cat('Dimension of our Bike data', dim(bike_day),' \n Data feature informations\n')
print(summary(bike_day))

# %% [code]
# viewing top level rows
head(bike_day)


# %% [code]
# list features information
str(bike_day)

# %% [code]
#Rename the columns
names(bike_day)<-
c('instant','datetime','season','year','month','holiday','weekday','workingday','weat
her_condition','temp','atemp','humidity','windspeed','casual','registered','total_cou
nt')
str(bike_day)

# %% [code]
# convert appropriate numerical features to categorical
categorical_col = c('season', 'year', 'month', 'holiday', 'weekday','workingday',
'weather_condition')
bike_day[,categorical_col] = lapply(bike_day[,categorical_col], factor) # convert as
factor
bike_day$datetime = as.Date(bike_day$datetime) # convert as datetime
str(bike_day)

# %% [code]
# different value frequency in each categorical features
for (col in categorical_col){
    print(col)
    print(table(bike_day[col]))
}

# %% [code]
# Missing values
cat('Data missing values:',sum(sum(is.na(bike_day)))) # print number of missing
values
missing_val<-data.frame(apply(bike_day,2,function(x){sum(is.na(x))}))
names(missing_val)[1]='missing_val'
missing_val

# %% [code]
# summary of data
summary(bike_day)

# %% [code]
# Distribution of target variable
hist(bike_day$total_count, breaks = 50, col = 'red', prob=TRUE) # plot histrogram of
target variable
lines(density(bike_day$total_count))  # plot density distribution of target variable


# %% [markdown]
```

```
# # 2. Methodology
# ### 2.1 Exploratory Data Analysis

# %% [code]
# categorical features
cat('Categorical features: ')
col = lapply(categorical_col,function(x) (cat(x,',')))

# %% [code]
#column plot for weather wise rental distribution of counts
p1 = ggplot(bike_day,aes(x=weather_condition,y=total_count,fill=season))+theme_bw(
)+geom_col()+ # plot stacked graph
labs(x='Weather',y='Total_Count',title='Weather wise rental distribution of counts')

#column plot for season wise rental distribution of counts
p2 = ggplot(bike_day,aes(x=season,y=total_count,fill=season))+theme_bw( )+geom_col()+
labs(x='Season',y='Total_Count',title='Season wise rental distribution of counts')

grid.arrange(p1,p2,ncol=2) # arrange in 1 row 2 col plot

# %% [code]
# column plot for season wise rental distribution of counts
p3 = ggplot(bike_day,aes(x=holiday,y=total_count,fill=season))+theme_bw(
)+geom_col()+
labs(x='Holiday',y='Total_Count',title='Holiday wise seasonal rental distribution of
counts')

# column plot for Holiday-weather wise rental distribution of counts
p4 = ggplot(bike_day,aes(x=holiday,y=total_count,fill=weather_condition))+theme_bw(
)+geom_col()+
labs(x='Holiday',y='Total_Count',title='Holiday-weather wise rental distribution of
counts')

# column plot for Holiday wise seasonal rental distribution of counts
p301 = ggplot(bike_day,aes(x=total_count,y=holiday,fill=season))+theme_bw(
)+geom_col()+
labs(x='Total_Count',y='Holiday',title='Holiday wise seasonal rental distribution of
counts')

# column plot for season wise holiday rental distribution
p302 = ggplot(bike_day,aes(x=total_count,y=season,fill=holiday))+theme_bw(
)+geom_col()+
labs(x='Total_Count',y='Season',title='season wise holiday rental distribution of
counts')


grid.arrange(p3,p4,ncol=2) # arrange 2 plot in 1 row
grid.arrange(p301,p302,ncol=2) # arrange 2 plot in 1 row

# %% [code]
# Year, Season, Casual, Registered distribution of rental

# Year wise seasonal rental distribution of counts
p5 = ggplot(bike_day,aes(x=year,y=total_count,fill=season))+theme_bw( )+geom_col()+
labs(x='Year',y='Total_Count',title='Year wise seasonal rental distribution of
counts')
```

```
# Casual wise seasonal rental distribution of counts
p6 = ggplot(bike_day,aes(x=casual,y=total_count,fill=season))+theme_bw( )+geom_col()+
labs(x='Casual',y='Total_Count',title='Casual wise seasonal rental distribution of
counts')

# Registered wise seasonal rental distribution of counts
p7 = ggplot(bike_day,aes(x=registered,y=total_count,fill=season))+theme_bw(
)+geom_col()+
labs(x='Registered',y='Total_Count',title='Registered wise seasonal rental
distribution of counts')

# Season wise casual rental distribution of counts
p8 = ggplot(bike_day,aes(x=season,y=total_count,fill=casual))+theme_bw( )+geom_col()+
labs(x='Season',y='Total_Count',title='Season wise casual rental distribution of
counts')

# Season wise registered rental distribution of counts
p9 = ggplot(bike_day,aes(x=season,y=total_count,fill=registered))+theme_bw(
)+geom_col()+
labs(x='Season',y='Total_Count',title='Season wise registered rental distribution of
counts')

# arranged plot
grid.arrange(p5,ncol=2)
grid.arrange(p6,p7,ncol=2)
grid.arrange(p8,p9,ncol=2)

# %% [code]
# season wise counts
p10 = ggplot(bike_day,aes(x=season,y=total_count,fill=weather_condition))+theme_bw(
)+geom_col()+
labs(x='Season',y='Total_Count',title='Season wise total rental distribution of
counts')

# Season wise casual rental distribution of counts
p11 = ggplot(bike_day,aes(x=season,y=casual,fill=weather_condition))+theme_bw(
)+geom_col()+
labs(x='Season',y='Casual',title='Season wise casual rental distribution of counts')

# Season wise registered rental distribution of counts
p12 = ggplot(bike_day,aes(x=season,y=registered,fill=weather_condition))+theme_bw(
)+geom_col()+
labs(x='Season',y='Registered',title='Season wise registered rental distribution of
counts')

# arrange plot
grid.arrange(p10,p11,p12, ncol=3)


# %% [code]
# Season wise weekdays rental distribution of counts
p13 = ggplot(bike_day,aes(x=weekday,y=total_count,fill=season))+theme_bw(
)+geom_col()+
labs(x='Weekdays',y='Total_Count',title='Season wise weekdays rental distribution of
counts')
```

```
grid.arrange(p13, ncol=1)


# %% [code]
# Month wise avg weekly total rental distribution of counts
p14 = ggplot(bike_day,aes(x=month,y=total_count, fill=weekday))+theme_bw( )+
         stat_summary(fun.y=mean, aes(group=1), geom="point", colour="blue")+
         stat_summary(fun.y=mean, aes(group=1), geom="line", colour="blue")+
         labs(x='Month',y='Total_Count',title='Month wise avg weekly total rental
distribution of counts')

# Month wise weekly total rental distribution of counts
p15 = ggplot(bike_day,aes(x=month,y=total_count, fill=weekday))+theme_bw(
)+geom_col()+
     labs(x='Month',y='Total_Count',title='Month wise weekly total rental
distribution of counts')

# arrange plot
grid.arrange(p14, p15, nrow=1)

# %% [code]
# avg weekly total rental distribution of counts
p16 = ggplot(bike_day,aes(x=weekday,y=total_count, fill=season))+theme_bw( )+
         stat_summary(fun.y=mean, aes(group=1), geom="point", colour="blue")+
         stat_summary(fun.y=mean, aes(group=1), geom="line", colour="blue")+
         labs(x='Weekday',y='Total_Count',title=' avg weekly total rental
distribution of counts')
# Weekday wise season total rental distribution of counts
p17 = ggplot(bike_day,aes(x=weekday,y=total_count, fill=season))+theme_bw(
)+geom_col()+
     labs(x='Weekday',y='Total_Count',title='Weekday wise season total rental
distribution of counts')
# Month wise season total rental distribution of counts
p18 = ggplot(bike_day,aes(x=month,y=total_count, fill=season))+theme_bw(
)+geom_col()+
     labs(x='Month',y='Total_Count',title='Month wise season total rental
distribution of counts')
# Month wise seasonal avg total rental distribution of counts
p1801 = ggplot(bike_day,aes(x=month,y=total_count, fill=season))+theme_bw( )+
        stat_summary(fun.y=mean, aes(group=1), geom="point", colour="blue")+
         stat_summary(fun.y=mean, aes(group=1), geom="line", colour="blue")+
     labs(x='Month',y='Total_Count',title='Month wise seasonal avg total rental
distribution of counts')
# Season wise weekly total rental distribution of counts
p19 = ggplot(bike_day,aes(x=season,y=total_count, fill=weekday))+theme_bw(
)+geom_col()+
     labs(x='Season',y='Total_Count',title='Season wise weekly total rental
distribution of counts')
# Season wise avg weekly total rental distribution of counts
p20= ggplot(bike_day,aes(x=season,y=total_count, fill=weekday))+theme_bw( )+
         stat_summary(fun.y=mean, aes(group=1), geom="point", colour="blue")+
         stat_summary(fun.y=mean, aes(group=1), geom="line", colour="blue")+
         labs(x='Season',y='Total_Count',title='Season wise avg weekly total rental
distribution of counts')

# arrange plots
```

```
grid.arrange(p16,p17, ncol=2)
grid.arrange(p1801,p18, ncol=2)
grid.arrange(p20,p19, ncol=2)

# %% [code]
# create new feature 'isweekend' : finding weekend days
# 0: sunday
# 6: saturday
weekend = as.data.frame(lapply(bike_day$weekday,function(x) if (x==0 |x==6) {1} else
{0} ), byrow=T, 'isweekend' )
bike_day$isweekend = as.factor(t(weekend)) # convert as factor for categorical

# %% [code]
head(bike_day)

# %% [code]
# Count weekends
table(bike_day$weekday==6 ) # Saturday counts
table(bike_day$weekday==0 ) # Sunday counts

# %% [code]
# Avg Use of the bikes by casual users
p21= ggplot(bike_day,aes(x=weekday,y=casual))+theme_bw( )+
         stat_summary(fun.y=mean, aes(group=1), geom="point", colour="blue")+
         stat_summary(fun.y=mean, aes(group=1), geom="line", colour="blue")+
         labs(x='weekday',y='casual',title='Avg Use of the bikes by casual users')
# Avg Use of the bikes by registered users
p22= ggplot(bike_day,aes(x=weekday,y=registered))+theme_bw( )+
         stat_summary(fun.y=mean, aes(group=1), geom="point", colour="blue")+
         stat_summary(fun.y=mean, aes(group=1), geom="line", colour="blue")+
         labs(x='weekday',y='registered',title='Avg Use of the bikes by registered
users')
# arrange plots
grid.arrange(p21,p22, ncol=2)


# %% [code]
# relation between windspeed and temp'
par(mfrow=c(1,3))
plot(temp ~ windspeed,  data=bike_day, col = c('green', 'red'), main = '-ve relation
between windspeed and temp')
abline(lm(temp ~ windspeed, data=bike_day), col='blue')

# relation between humidity and temp'
plot(temp ~ humidity,  data=bike_day, col = c('green', 'red'), main = '+ve relation
between humidity and temp')
abline(lm(temp ~ humidity, data=bike_day), col='blue')

# relation between windspeed and humidity'
plot(humidity ~ windspeed,  data=bike_day, col = c('green', 'red'), main = '-ve
relation between windspeed and humidity')
abline(lm(humidity ~ windspeed, data=bike_day), col='blue')

# %% [code]
# relation between windspeed and total_count'
par(mfrow=c(1,3)) # arrange plots
```

```
plot(total_count ~ windspeed,  data=bike_day, col = c('green', 'red'), main = '-ve
relation between windspeed and total_count')
abline(lm(total_count ~ windspeed, data=bike_day), col='blue')

# relation between humidity and total_count'
plot(total_count ~ humidity,  data=bike_day, col = c('green', 'red'), main = '-ve
relation between humidity and total_count')
abline(lm(total_count ~ humidity, data=bike_day), col='blue')

# relation between total_count and temp'
plot(total_count ~ temp,  data=bike_day, col = c('green', 'red'), main = '+ve
relation between temp and total_count')
abline(lm(total_count ~ temp, data=bike_day), col='blue')

# %% [code]
# Pair plots
options(repr.plot.width=10, repr.plot.height=10) # adjust figure size to 10x10 for
perfect visualization of pairplots

# 1. pairs()
pairs.default(bike_day[,c(10:16)], col = c("red", "cornflowerblue"), main='Pairs
Pairplot')

# 2. ggpairs()
 ggpairs(bike_day[,c(10:16)], mapping=ggplot2::aes(colour = c("red")), title='GGpairs
Pairplot')

# %% [code]
# Correlation Analysis
cormat <- cor(bike_day[,c(10:16)]) # get correlation matrix

summary(cormat[upper.tri(cormat)])
# heatmap
heatmap(cormat, scale = 'none')

# %% [code]
# Boxplot analysis

options(repr.plot.width=15, repr.plot.height=5) # reset figure size to 15x5
par(mfrow=c(1,2)) # subplots: for each row 2 plots

# Total count rental Data
boxplot(bike_day$total_count , data = bike_day, xlab = "Total_count",
          main = "Total count rental Data", varwidth = TRUE, col = c("red"))

# Season rental Data
boxplot(bike_day$total_count ~ bike_day$season, data = bike_day, xlab = "season",ylab
= "Total count",
          main = "Season rental Data", varwidth = TRUE, col =
c("red","green","blue"), n)

# Year rental Data
boxplot(bike_day$total_count ~ bike_day$year, data = bike_day, xlab = "Year",ylab =
"count",
          main = "Year rental Data", varwidth = TRUE, col = c("red", "yellow"))
```

```
# Weather condition Data
boxplot(bike_day$total_count ~ bike_day$weather_condition, data = bike_day, xlab =
"weather_condition",ylab = "Total count",
          main = "Weather condition Data", varwidth = TRUE, col =
c("red","green","blue"))

# Month Data
boxplot(bike_day$total_count ~ bike_day$month, data = bike_day, xlab = "month",ylab =
"count",
          main = "Month Data", varwidth = TRUE, col = c("red", "yellow"))

# Weekday Data
boxplot(bike_day$total_count ~ bike_day$weekday, data = bike_day, xlab =
"weekday",ylab = "Total count",
          main = "Weekday Data", varwidth = TRUE, col = c("red","green","blue"))

# Windspeed Data
boxplot(bike_day$windspeed , data = bike_day, xlab = "windspeed",
          main = "Windspeed Data", varwidth = TRUE, col = c("red"))

# Humidity Data
boxplot(bike_day$humidity , data = bike_day, xlab = "humidity",
          main = "Humidity Data", varwidth = TRUE, col = c("green"))

# %% [markdown]
# <b>Correlation between continous features

# %% [code]
# create a subplot of continous fetures only
sub=data.frame(bike_day$registered,bike_day$casual,bike_day$total_count,bike_day$temp
,bike_day$humidity,bike_day$atemp,bike_day$windspeed)
cor(sub) # print correlation matrix

# %% [markdown]
# ## 2.2 Data Preprocessing

# %% [code]
#create subset for windspeed and humidity variable
wind_humidity = subset(bike_day,select=c('windspeed','humidity'))
#column names of wind_hum
cnames = colnames(wind_humidity)
for(i in cnames){
    val=wind_humidity[,i][wind_humidity[,i] %in%
boxplot.stats(wind_humidity[,i])$out] #outlier values
    wind_humidity[,i][wind_humidity[,i] %in% val]= NA  # Replace outliers with NA
}

# %% [code]
#Remove the windspeed and humidity variable in order to replace imputated data
new_df = subset(bike_day,select=-c(windspeed,humidity))

#Combined new_df and wind_hum data frames
bike_df = cbind(new_df,wind_humidity)
head(bike_df)
```

```
# %% [code]
cat('Shape after dropping outlier (windspeed,humidity):',dim(wind_humidity))
cat('\nShape before dropping outlier (windspeed,humidity):',dim(
drop_na(wind_humidity)))

# %% [code]
# drop outliers
bike_day = drop_na(bike_df)
cat('Shape after dropping outlier bike data:',dim(bike_day),'\n')
head(bike_day)

# %% [markdown]
# ## 2.3 Feature Engg

# %% [code]
# categorising features
categorical_features =
c("season","holiday","weather_condition","weekday","month","year",'isweekend','workin
gday')
continous_features = c("temp","humidity","windspeed")
dropFeatures = c('casual',"datetime","instant","registered","atemp")
target=c('total_count')

# %% [code]
# create subset of categorical features only to create thier dummy features
bike_day  = subset(bike_day, select = -c(casual,datetime,instant,registered,atemp))
names(bike_day)

# %% [code]
# features to create dummy
get_dummy_features = categorical_features

# Using fastDummies function to create dummy features
dummy_data = fastDummies::dummy_cols(bike_day, select_columns=get_dummy_features,
remove_first_dummy = TRUE) # create dummy, also drop the first feature of each dummy
variable
dummy_data = subset(dummy_data, select = -
c(season,holiday,weather_condition,weekday,month,year,isweekend,workingday)) # drop
original categorical features
str(dummy_data) # print new data information

# %% [code]
# copy new feature enginnered data
bike_day = dummy_data
head(bike_day)

# %% [markdown]
# # 3. Modeling

# %% [code]
# Sampling


#Set seed to reproduce the results of random sampling
set.seed(42)
```

```
# Splitting into train and test , 75% and 25% respectivly
n = nrow(bike_day)
trainIndex = sample(1:n, size = round(0.75*n), replace=FALSE)
train = bike_day[trainIndex ,]
test = bike_day[-trainIndex ,]

cat('train dim:',dim(train))
cat('\ntest dim:',dim(test))

# %% [markdown]
# ### Linear model: LinearRegression | Ridge | Lasso | ElasticNet

# %% [code]
# function: calculate MAPE
MAPE = function(actual, pred){
  mean(abs((actual - pred)/actual)) * 100
}

# function: calculate SSR
SSR = function(actual, pred){
    y_hat_cv <- predict(model_cv, X)
    ssr_cv <- t(y - y_hat_cv) %*% (y - y_hat_cv)
    rsq_ridge_cv <- cor(y, y_hat_cv)
}
# function: calculate multiple R-squared
R_squared = function(actual, pred){
    rss <- sum((pred - actual) ^ 2)
    tss <- sum((actual - mean(actual)) ^ 2)
    rsq <- 1 - rss/tss

}
# function: calculate r2 by covariance method
R_sqrd_cov = function(actual, pred){
    cor(train[,2], lr_pred)^2
}

# function: calculate  Adj R-squared
Adj_R_squared = function(r2, n, p){
    adj_r = 1 - ((1-r2)*(n-1))/(n-p-1)
}


# %% [code]
# simple linear model

# Set seed to reproduce the results of random sampling
set.seed(42)

# training the lr_model
lr_model = lm(train$total_count~.,data = train)

# Check the summary of the model
summary(lr_model)

# Predict the test cases
lr_predictions = predict(lr_model, test[,-2])
```

```
# get dimension to calculate rmsle
n = dim(test)[1]
p = dim(test)[2]


print('Metrics on Test data:')
regr.eval(trues = test[,2], preds = lr_predictions, stats =
c("mae","mse","rmse","mape"))
cat('MAPE:',MAPE(test[,2], lr_predictions))
cat('\nR^2:',R_squared(test[,2], lr_predictions))
cat('\nAdj R^2:', Adj_R_squared(r2=R_squared(test[,2], lr_predictions), dim(test)[1],
dim(test)[2]))
cat('\nRMSE:',rmse(test[,2], lr_predictions))
cat('\nRMSLE:',rmsle(test[,2], lr_predictions))

# simple linear model
# MAPE: 15.85646
# R^2: 0.8717872
# Adj R^2: 0.8457981
# RMSE: 746.702
# RMSLE: 0.2219201

# %% [code]
# plot simple linear model graph
plot(lr_model)

# print summary of simple linear model
summary(lr_predictions)

# %% [code]
# lasso model

# Set seed to reproduce the results
set.seed(14)

# Perform 10-fold cross-validation to select lambda ----------------------------
lambda_seq = 10^seq(-3, 5, length.out = 100)

# Setting alpha = 1 implements lasso regression
cv_output = cv.glmnet(as.matrix(train[,-c(2)]), as.matrix(train[,2]), alpha=1, lambda
= lambda_seq)

# plot cross-validation result
plot(cv_output)

# print best lambda value
cat('Best lambda: ',cv_output$lambda.min)

# fit model with multiple lambda
lasso_model = glmnet(as.matrix(train[,-c(2)]), as.matrix(train[,2]), alpha=1, lambda
= lambda_seq )

# plot model to visualize effect of lambda
plot(lasso_model, xvar = "lambda")
legend("bottomright", lwd = 1, col = 1:6, legend = colnames(train[,-2]), cex = .7)
```

```
# Best lambda:  1.707353

# %% [code]
# Set seed to reproduce the results
set.seed(17)

# fit model with best lambda
lasso_best = glmnet(as.matrix(train[,-c(2)]), as.matrix(train[,2]), alpha=1, lambda =
cv_output$lambda.min )

# predict test data after fitting to best lambda
lasso_prediction = predict(lasso_best, as.matrix(test[,-2]))

# get dimension to calculate rmsle
n = dim(test)[1]
p = dim(test)[2]


cat('\n\nMetrics on Test data:')
cat('\nMAPE:',MAPE(test[,2], lasso_prediction))
cat('\nR^2:',R_squared(test[,2], lasso_prediction))
cat('\nAdj R^2:', Adj_R_squared(r2=R_squared(test[,2], lasso_prediction), n,p))
cat('\nRMSE:',rmse(test[,2], lasso_prediction))
cat('\nRMSLE:',rmsle(test[,2], lasso_prediction))

# lasso model
# Metrics on Test data:
# MAPE: 15.881
# R^2: 0.8718666
# Adj R^2: 0.8458936
# RMSE: 746.4707
# RMSLE: 0.2208965

# %% [code]
# ridge model

# Set seed to reproduce the results
set.seed(147)

# Perform 10-fold cross-validation to select lambda ---------------------------
#lambda_seq = c( 0.1, 1, 2, 3, 4, 5, 10, 30, 50, 80, 100)
lambda_seq = 10^seq(-3, 5, length.out = 100)

# Setting alpha = 0 implements ridge regression
cv_output = cv.glmnet(as.matrix(train[,-c(2)]), as.matrix(train[,2]), alpha=0, lambda
= lambda_seq)

# plot cross-validation result
plot(cv_output)

# print best lambda value
cat('Best lambda: ',cv_output$lambda.min)

# fit model with multiple lambda
```

```
ridge_model = glmnet(as.matrix(train[,-c(2)]), as.matrix(train[,2]), alpha=1, lambda
= lambda_seq )

# plot model to visualize effect of lambda
plot(ridge_model, xvar = "lambda")
legend("bottomright", lwd = 1, col = 1:6, legend = colnames(train[,-2]), cex = .7)

# Best lambda:  33.51603

# %% [code]
# Set seed to reproduce the results
set.seed(147)

# fit model with best lambda
ridge_best = glmnet(as.matrix(train[,-c(2)]), as.matrix(train[,2]), alpha=1, lambda =
cv_output$lambda.min )

# predict test data after fitting to best lambda
ridge_prediction = predict(ridge_best, as.matrix(test[,-2]))

# get dimension to calculate rmsle
n = dim(test)[1]
p = dim(test)[2]



cat('\n\nMetrics on Test data:')
cat('\nMAPE:',MAPE(test[,2], ridge_prediction))
cat('\nR^2:',R_squared(test[,2], ridge_prediction))
cat('\nAdj R^2:', Adj_R_squared(r2=R_squared(test[,2], ridge_prediction), n,p))
cat('\nRMSE:',rmse(test[,2], ridge_prediction))
cat('\nRMSLE:',rmsle(test[,2], ridge_prediction))


# ridge model
# Metrics on Test data:
# MAPE: 17.04704
# R^2: 0.8597313
# Adj R^2: 0.8312985
# RMSE: 781.0196
# RMSLE: 0.2163474

# %% [code]
# elastic net model

# Set seed to reproduce the results
set.seed(17)

# Perform 10-fold cross-validation to select lambda ---------------------------
lambda_seq = 10^seq(-3, 5, length.out = 100)

# Setting alpha = 0.5 implements elastic net regression
cv_output = cv.glmnet(as.matrix(train[,-c(2)]), as.matrix(train[,2]), alpha=0.5,
lambda = lambda_seq)

# plot cross-validation result
```

```
plot(cv_output)

# print best lambda value
cat('Best lambda: ',cv_output$lambda.min)

# fit model with multiple lambda
elastic_model = glmnet(as.matrix(train[,-c(2)]), as.matrix(train[,2]), alpha=1,
lambda = lambda_seq )

# plot model to visualize effect of lambda
plot(elastic_model, xvar = "lambda")
legend("bottomright", lwd = 1, col = 1:6, legend = colnames(train[,-2]), cex = .7)


# Best lambda:  13.21941

# %% [code]
# Set seed to reproduce the results
set.seed(14)

# fit model with best lambda
elastic_best = glmnet(as.matrix(train[,-c(2)]), as.matrix(train[,2]), alpha=1, lambda
= cv_output$lambda.min )

# predict test data after fitting to best lambda
elastic_prediction = predict(elastic_best, as.matrix(test[,-2]))

# get dimension to calculate rmsle
n = dim(test)[1]
p = dim(test)[2]


cat('\n\nMetrics on Test data:')
cat('\nMAPE:',MAPE(test[,2], elastic_prediction))
cat('\nR^2:',R_squared(test[,2], elastic_prediction))
cat('\nAdj R^2:', Adj_R_squared(r2=R_squared(test[,2], elastic_prediction), n,p))
cat('\nRMSE:',rmse(test[,2], elastic_prediction))
cat('\nRMSLE:',rmsle(test[,2], elastic_prediction))


# elasticnet model
# Metrics on Test data:
# MAPE: 16.19882
# R^2: 0.8694822
# Adj R^2: 0.8430259
# RMSE: 753.3842
# RMSLE: 0.215776

# %% [markdown]
# ### Ensemble & Boosting models: RandomForest | GBM

# %% [code]
# random froset

# Set seed to reproduce the results
set.seed(42)
```

```
# training the rf_model
rf_model = randomForest(train$total_count~.,data = train)

# get summary
rf_model

# predict test data using rf_model
rf_prediction = predict(rf_model, as.matrix(test[,-2]))

# get dimension to calculate rmsle
n = dim(test)[1]
p = dim(test)[2]

cat('\n\nMetrics on Test data:')
cat('\nMAPE:',MAPE(test[,2], rf_prediction))
cat('\nR^2:',R_squared(test[,2], rf_prediction))
cat('\nAdj R^2:', Adj_R_squared(r2=R_squared(test[,2], rf_prediction), n,p))
cat('\nRMSE:',rmse(test[,2], rf_prediction))
cat('\nRMSLE:',rmsle(test[,2], rf_prediction))


# Call:
#  randomForest(formula = train$total_count ~ ., data = train)
#                Type of random forest: regression
#                      Number of trees: 500
# No. of variables tried at each split: 9

#           Mean of squared residuals: 499178.1
#                     % Var explained: 85.85


# Metrics on Test data:
# MAPE: 13.73607
# R^2: 0.9075762
# Adj R^2: 0.8888417
# RMSE: 633.9769
# RMSLE: 0.1943775

# %% [code]
# gbm

# Set seed to reproduce the results
set.seed(17)

# training the rf_model
gbm_model = gbm(total_count~.,data = train)

# get summary
gbm_model

ntrees = seq(from=100 ,to=10000, by=100) #no of trees-a vector of 100 values

# predict test data using gbm_model
gbm_prediction = round(predict(gbm_model, test[,-2], n.trees = 5000 ))
```

```
# get dimension to calculate rmsle
n = dim(test)[1]
p = dim(test)[2]

cat('\n\nMetrics on Test data:')
cat('\nMAPE:',MAPE(test[,2], gbm_prediction))
cat('\nR^2:',R_squared(test[,2], gbm_prediction))
cat('\nAdj R^2:', Adj_R_squared(r2=R_squared(test[,2], gbm_prediction), n,p))
cat('\nRMSE:',rmse(test[,2], gbm_prediction))
cat('\nRMSLE:',rmsle(test[,2], gbm_prediction))

# gbm(formula = total_count ~ ., data = train)
# A gradient boosted model with gaussian loss function.
# 100 iterations were performed.
# There were 29 predictors of which 11 had non-zero influence.


# Metrics on Test data:
# MAPE: 18.46409
# R^2: 0.8639603
# Adj R^2: 0.8363847
# RMSE: 769.156
# RMSLE: 0.2498177

# %% [markdown]
# ### Hyper-paramter tuning of ensemble models

# %% [code]
# hyper-paramter tunning for randomforest model

# Set seed to reproduce the results
set.seed(14)

control = trainControl(method="repeatedcv", number=10, repeats=3, search="grid") #
create parameter controls
tunegrid = expand.grid(.mtry=c(1:15)) # number of try

# tune the model
rf_gridsearch = train(total_count~., data=train, method="rf", metric='rmse',
tuneGrid=tunegrid, trControl=control)
print(rf_gridsearch)
plot(rf_gridsearch)

# Random Forest

# 538 samples
#  29 predictor

# No pre-processing
# Resampling: Cross-Validated (10 fold, repeated 3 times)
# Summary of sample sizes: 484, 483, 485, 484, 484, 484, ...
# Resampling results across tuning parameters:

#   mtry  RMSE       Rsquared   MAE
#    1    1497.8445  0.7212547  1206.6032
#    2    1117.8434  0.7966798   904.4551
```

```
#    3     923.3875  0.8318335   733.4619
#    4     819.7871  0.8523184   631.9902
#    5     765.7639  0.8602762   574.4973
#    6     735.2557  0.8646165   544.6450
#    7     719.2258  0.8659400   525.5161
#    8     708.7438  0.8666198   514.1552
#    9     700.9273  0.8672585   505.7798
#   10     697.4989  0.8669734   502.8166
#   11     692.5154  0.8674696   497.8712              <---- optimal parameter which
is actually less than our original rf_model
#   12     697.2182  0.8645921   498.2345
#   13     692.8876  0.8656716   495.1647
#   14     696.0455  0.8637380   495.6083
#   15     697.5091  0.8628047   496.2870

# RMSE was used to select the optimal model using the smallest value.
# The final value used for the model was mtry = 11.

# %% [code]
# predict test data using rf_gridsearch
rf_pred = predict(rf_gridsearch, as.matrix(test[,-2]))

cat('\n\nMetrics on Test data:')
cat('\nMAPE:',MAPE(test[,2], rf_pred))
cat('\nR^2:',R_squared(test[,2], rf_pred))
cat('\nAdj R^2:', Adj_R_squared(r2=R_squared(test[,2], rf_pred), n,p))
cat('\nRMSE:',rmse(test[,2], rf_pred))
cat('\nRMSLE:',rmsle(test[,2], rf_pred))

# tuned random froset
# Metrics on Test data:
# MAPE: 13.46603
# R^2: 0.9099247
# Adj R^2: 0.8916662
# RMSE: 625.8704
# RMSLE: 0.1940767

# %% [code]
# hyper-paramter tunning for gbm model

# Set seed to reproduce the results
set.seed(14)

fitControl = trainControl(method="repeatedcv", number=10, repeats=3, search="grid") #
create parameter controls
ntrees = seq(from=100 ,to=10000, by=100) #no of trees-a vector of 100 values

# hyper-paramters
gbmGrid =  expand.grid(interaction.depth = c(1, 5, 9),
                       n.trees = (1:30)*50,
                       shrinkage = 0.1,
                       n.minobsinnode = 20)

nrow(gbmGrid)

set.seed(825) # Set seed to reproduce the results
```

```
# tune the model
gbm_gridsearch = train(total_count~., data = train,
                  method = "gbm",
                  trControl = fitControl,
                  verbose = FALSE,
                  ## Now specify the exact models
                  ## to evaluate:
                  tuneGrid = gbmGrid)

gbm_gridsearch
print(gbm_gridsearch)
plot(gbm_gridsearch)

# Stochastic Gradient Boosting

# 538 samples
#  29 predictor

# No pre-processing
# Resampling: Cross-Validated (10 fold, repeated 3 times)
# Summary of sample sizes: 484, 484, 485, 484, 484, 486, ...
# Resampling results across tuning parameters:

#   interaction.depth  n.trees  RMSE      Rsquared   MAE
#   1                   50      923.3652  0.7839935  727.0566
#   1                  100      812.6623  0.8175004  630.9014
#   1                  150      788.5031  0.8257042  602.6373
#   1                  200      781.7537  0.8288068  592.3475
#   1                  250      776.8015  0.8309069  588.0951
#   1                  300      774.4639  0.8320001  585.3111
#   1                  350      774.4449  0.8324349  584.7461
#   1                  400      774.9251  0.8321007  585.1204
#   1                  450      773.8584  0.8324131  584.4772
#   1                  500      773.0691  0.8329152  583.8875
#   1                  550      771.0937  0.8336384  580.7596
#   1                  600      772.1280  0.8332527  581.8413
#   1                  650      771.1943  0.8336311  580.9198
#   1                  700      772.2819  0.8333544  579.5284
#   1                  750      772.8175  0.8332057  579.4492
#   1                  800      772.4673  0.8330696  579.6616
#   1                  850      773.9123  0.8325743  580.9832
#   1                  900      774.2362  0.8324603  580.9768
#   1                  950      774.5461  0.8323220  581.7716
#   1                 1000      776.3441  0.8314947  583.0538
#   1                 1050      774.3921  0.8323750  581.6075
#   1                 1100      775.1891  0.8320058  581.2110
#   1                 1150      776.2429  0.8316900  582.1212
#   1                 1200      778.1513  0.8309015  584.3019
#   1                 1250      777.9033  0.8310026  583.4613
#   1                 1300      779.5747  0.8302423  584.3113
#   1                 1350      779.0441  0.8303551  583.8664
#   1                 1400      779.8449  0.8302057  584.5140
#   1                 1450      780.6199  0.8298021  584.8280
#   1                 1500      782.1188  0.8291072  585.4991
#   5                   50      734.5991  0.8486256  537.2469
```

```
#    5                     100     705.7227  0.8600561  511.1392
#    5                     150     696.5666  0.8641200  506.7654
#    5                     200     693.4572  0.8654154  503.9453
#    5                     250     695.0371  0.8649859  504.4506
#    5                     300     693.3404  0.8659269  504.3299
#    5                     350     696.1071  0.8648678  506.3286
#    5                     400     697.0409  0.8646030  507.1596
#    5                     450     701.0508  0.8630736  509.7309
#    5                     500     702.6047  0.8625800  510.3181
#    5                     550     706.0559  0.8613689  513.6271
#    5                     600     707.9440  0.8608000  514.9558
#    5                     650     709.8509  0.8600924  516.2359
#    5                     700     710.7190  0.8598858  518.4503
#    5                     750     713.1627  0.8589719  519.9647
#    5                     800     714.0550  0.8586054  520.2812
#    5                     850     715.7761  0.8581567  521.4581
#    5                     900     715.7401  0.8581208  521.9280
#    5                     950     716.5944  0.8577030  522.9020
#    5                    1000     718.8495  0.8568740  523.5233
#    5                    1050     719.7150  0.8566935  524.2363
#    5                    1100     720.8218  0.8562605  524.2308
#    5                    1150     722.3007  0.8556999  525.3797
#    5                    1200     723.4379  0.8553739  525.6739
#    5                    1250     723.7040  0.8553880  525.9925
#    5                    1300     725.3849  0.8547496  526.9246
#    5                    1350     725.0101  0.8548591  526.3578
#    5                    1400     725.1119  0.8549255  526.7508
#    5                    1450     725.9680  0.8545900  527.7801
#    5                    1500     726.3156  0.8545493  527.8939
#    9                      50     726.2360  0.8520688  526.1470
#    9                     100     700.7431  0.8623221  507.3497
#    9                     150     692.5382  0.8657030  503.5086           <------
optimal model which is actually les than our rf_model
#    9                     200     694.5902  0.8651411  505.3311
#    9                     250     693.9644  0.8652073  506.3707
#    9                     300     696.2700  0.8646263  506.9534
#    9                     350     700.6151  0.8631437  510.0052
#    9                     400     703.1501  0.8622154  511.6705
#    9                     450     704.3624  0.8617556  513.3123
#    9                     500     707.0642  0.8607546  515.1844
#    9                     550     706.6695  0.8610575  515.9672
#    9                     600     707.8411  0.8603813  516.1775
#    9                     650     709.2722  0.8598979  517.2824
#    9                     700     709.6714  0.8598398  517.5066
#    9                     750     710.2544  0.8597553  518.0704
#    9                     800     711.8293  0.8590314  518.9023
#    9                     850     712.6644  0.8587927  519.4860
#    9                     900     713.0533  0.8586324  519.8397
#    9                     950     715.3400  0.8577503  521.0784
#    9                    1000     716.3919  0.8573509  522.1011
#    9                    1050     716.5869  0.8572855  522.2637
#    9                    1100     716.0585  0.8575871  522.0127
#    9                    1150     717.6065  0.8570189  523.3644
#    9                    1200     719.1379  0.8564307  524.1781
#    9                    1250     719.4571  0.8562880  524.5529
#    9                    1300     720.4824  0.8559568  525.1753
```

```
#   9                  1350     720.8619  0.8558299  525.4463
#   9                  1400     721.3509  0.8557048  525.6083
#   9                  1450     721.9653  0.8554927  526.3144
#   9                  1500     722.8684  0.8551270  526.8665

# Tuning parameter 'shrinkage' was held constant at a value of 0.1

# Tuning parameter 'n.minobsinnode' was held constant at a value of 20
# RMSE was used to select the optimal model using the smallest value.
# The final values used for the model were n.trees = 150, interaction.depth =
#  9, shrinkage = 0.1 and n.minobsinnode = 20.

# %% [code]
# predict test data using gbm_gridsearch
gbm_pred = predict(gbm_gridsearch, as.matrix(test[,-2]))

cat('\n\nMetrics on Test data:')
cat('\nMAPE:',MAPE(test[,2], gbm_pred))
cat('\nR^2:',R_squared(test[,2], gbm_pred))
cat('\nAdj R^2:', Adj_R_squared(r2=R_squared(test[,2], gbm_pred), n,p))
cat('\nRMSE:',rmse(test[,2], gbm_pred))
cat('\nRMSLE:',rmsle(test[,2], gbm_pred))


#  gbm_gridsearch
# Metrics on Test data:
# MAPE: 13.70147
# R^2: 0.9137555
# Adj R^2: 0.8962735
# RMSE: 612.4171
# RMSLE: 0.2049584

# %% [markdown]
# # 4. Final Model

# %% [code]
# final model

# tuned random froset

# Set seed to reproduce the results
set.seed(14)

control = trainControl(method="repeatedcv", number=10, repeats=3, search="grid") #
create parameter controls
tunegrid = expand.grid(.mtry=c(11))

# tune the model
rf_gridsearch = train(total_count~., data=train, method="rf", metric='rmse',
tuneGrid=tunegrid, trControl=control)
# predict the test data
rf_prediction = predict(rf_gridsearch, as.matrix(test[,-2]))

# get dimension to calculate rmsle
n = dim(test)[1]
p = dim(test)[2]
```

```
cat('\n\nMetrics on Test data:')
cat('\nMAPE:',MAPE(test[,2], rf_prediction))
cat('\nR^2:',R_squared(test[,2], rf_prediction))
cat('\nAdj R^2:', Adj_R_squared(r2=R_squared(test[,2], rf_prediction), n,p))
cat('\nRMSE:',rmse(test[,2], rf_prediction))
cat('\nRMSLE:',rmsle(test[,2], rf_prediction))


# tuned random froset
# Metrics on Test data:
# MAPE: 13.27936
# R^2: 0.9112103
# Adj R^2: 0.8932124
# RMSE: 621.3882
# RMSLE: 0.1911517

# %% [code]
# submission
Bike_predictions=data.frame(test[,2],round(rf_prediction))
write.csv(Bike_predictions,'Bike_Renting_R.CSV',row.names=F)
Bike_predictions

# %% [markdown]
# Download CSV for Test result:
# <a  href="Bike_Renting_R.CSV" target="_blank">download Bike_Renting_Python</a>
#
```

References

- https://www.quora.com/What-is-the-difference-between-an-RMSE-and-RMSLE-logarithmic-error-and-does-a-high-RMSE-imply-low-RMSLE
- https://medium.com/analytics-vidhya/root-mean-square-log-error-rmse-vs-rmlse-935c6cc1802a
- https://medium.com/usf-msds/choosing-the-right-metric-for-machine-learning-models-part-1-a99d7d7414e4
- https://www.analyticsvidhya.com/blog/2019/08/11-important-model-evaluation-error-metrics/
- https://opendatagroup.github.io/Knowledge%20Center/Tutorials/Gradient%20Boosting%20Regressor/
- https://en.wikipedia.org/wiki/Gradient_boosting
- https://machinelearningmastery.com/gentle-introduction-gradient-boosting-algorithm-machine-learning/
- https://www.analyticsvidhya.com/blog/2016/02/complete-guide-parameter-tuning-gradient-boosting-gbm-python/
- https://www.analyticsvidhya.com/blog/2015/08/comprehensive-guide-regression/
- https://www.analyticsvidhya.com/blog/2017/06/a-comprehensive-guide-for-linear-ridge-and-lasso-regression/
- https://medium.com/datadriveninvestor/random-forest-regression-9871bc9a25eb
- https://acadgild.com/blog/random-forest-algorithm-regression