

Contents

1	Introduction & Objective	2
2	Data Description	2
3	Data Visualisation and Cleaning	3
3.1	Shape of the data	3
3.2	Viewing first few observations of the data sets	3
3.3	Feature data types and Feature Statistics summary	3
3.4	Checking for missing data	4
3.5	Modifying datasets for ease of use and Categorical data conversion	5
4	Exploratory Data Analysis (EDA)	7
4.1	Plot of count vs. average values across the categorical columns Weather, Season, Working Day and Holiday	7
4.2	Plots of number of bikes rented vs. the temperature with the data split on Working day or Non working day	8
4.3	Hourly Distribution	8
4.4	Monthly distribution	9
4.5	Yearly Distribution	10
4.6	Regression Plots	10
4.7	Correlation analysis	11
5	Time Series Analysis	11
5.1	Test for Trend- Relative Ordering Test	12
5.2	Test for Stationarity- ADF Test	13
6	OLS Fitting	14
6.1	Correlation Among Categorical Variables	14
6.2	Dealing with Multicollinearity	15
6.3	Outlier Detection: Leverage and Influential Points	17
6.4	Inspection of the Normality Assumption of Errors	19
6.4.1	Q-Q Plot	19
6.4.2	Histogram Approach	20
6.4.3	Shapiro-Wilk Test for Normality	21
6.5	Inspection of Autocorrelation among the Errors	22
6.6	Inspection of Homoscedastic Assumptions of Errors	23
6.6.1	Residual vs Fitted Plot	23
6.6.2	Breusch-Pagan Test for Heteroscedasticity	23
6.7	Variable Selection	24
6.8	Final Model Fit	27
6.9	Predictions	27
7	Ridge Regression	28
7.0	Predictions	28
8	Predictions and Error Metrics	29
8.1	R^2 and Adjusted R^2	29
8.2	Mean absolute percentage error(MAPE)	29
8.3	Mean Absolute Deviation Percentage(MADP)	29
8.4	Cumulative Error Percentage(CERPCT)	30
8.5	Comparing model adequacy and error metrics	30

1 Introduction & Objective

Bike sharing systems are a means of renting bicycles where the process of obtaining membership, rental, and bike return is automated via a network of kiosk locations throughout a city. Using these Bike Sharing systems, people rent a bike from one location and return it to a different or same place on need basis. People can rent a bike through membership (mostly regular users) or on demand basis (mostly casual users). This process is controlled by a network of automated kiosk across the city. Currently, there are over 500 bike-sharing programs around the world.

The data generated by these systems makes them attractive for researchers because the duration of travel, departure location, arrival location, and time elapsed is explicitly recorded. Bike sharing systems therefore function as a sensor network, which can be used for studying mobility in a city. Bike-sharing rental process is highly correlated to the environmental and seasonal settings. For instance, weather conditions, precipitation, day of week, season, hour of the day, etc. can affect the rental behaviors. Here, we combine historical usage patterns with weather data in order to forecast bike rental demand in the Capital Bikeshare program in Washington, D.C.

2 Data Description

The core data set is related to the two-year historical log corresponding to years 2011 and 2012 from Capital Bikeshare system, Washington D.C., USA which is publicly available in <http://capitalbikeshare.com/system-data>. We aggregated the data on two hourly and daily basis and then extracted and added the corresponding weather and seasonal information. Weather information are extracted from <http://www.freemeteo.com>.

The day and hour data sets shows daily and hourly rental data respectively, for two years (2011 and 2012). We have bike demand separately as registered or casual users and the sum of both is given as count. Both hour.csv and day.csv have the following 17 fields, except hr which is not available in day.csv. They are described below:

- **instant** : record index
- **dteday** : date
- **season** : season (1:spring, 2:summer, 3:fall, 4:winter)
- **yr** : year (0: 2011, 1:2012)
- **mnth** : month (1 to 12)
- **hr** : hour (0 to 23)
- **holiday** : weather day is holiday or not (extracted from <http://dchr.dc.gov/page/holiday-schedule>)
- **weekday** : day of the week
- **workingday** : if day is neither weekend nor holiday is 1, otherwise is 0.
- **weathersit** :
 - 1: Clear, Few clouds, Partly cloudy, Partly cloudy
 - 2: Mist + Cloudy, Mist + Broken clouds, Mist + Few clouds, Mist
 - 3: Light Snow, Light Rain + Thunderstorm + Scattered clouds, Light Rain + Scattered clouds
 - 4: Heavy Rain + Ice Pallets + Thunderstorm + Mist, Snow + Fog
- **temp** : Normalized temperature in Celsius. The values are divided to 41 (max)
- **atemp**: Normalized feeling temperature in Celsius. The values are divided to 50 (max)
- **hum**: Normalized humidity. The values are divided to 100 (max)
- **windspeed**: Normalized wind speed. The values are divided to 67 (max)
- **casual**: count of casual users
- **registered**: count of registered users
- **cnt**: count of total rental bikes including both casual and registered

3 Data Visualisation and Cleaning

3.1 Shape of the data

The day data set consists 731 rows and 16 columns while the hour data set consists 17379 rows and 17 columns. We set the dteday column as our index column

Load the dataset

```
[ ] # Read the data
    day = pd.read_csv('sample_data/day.csv', parse_dates=True, index_col='dteday')
    hour = pd.read_csv('sample_data/hour.csv', parse_dates=True, index_col='dteday')
```

Shape of data

```
[ ] print('Shape of data: ', day.shape)
    print('Shape of data: ', hour.shape)
```

```
Shape of data: (731, 15)
Shape of data: (17379, 16)
```

Figure 1: Loading datasets, making 'dteday' column as index and checking shape of datasets

3.2 Viewing first few observations of the data sets

```
[ ] day.head()
```

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

```
[ ] hour.head()
```

	instant	season	yr	mnth	hr	holiday	weekday	workingday	weathersit	temp	atemp	hum	windspeed	casual	registered	cnt
dteday																
2011-01-01	1	1	0	1	0	0	6	0	1	0.24	0.2879	0.81	0.0	3	13	16
2011-01-01	2	1	0	1	1	0	6	0	1	0.22	0.2727	0.80	0.0	8	32	40
2011-01-01	3	1	0	1	2	0	6	0	1	0.22	0.2727	0.80	0.0	5	27	32
2011-01-01	4	1	0	1	3	0	6	0	1	0.24	0.2879	0.75	0.0	3	10	13
2011-01-01	5	1	0	1	4	0	6	0	1	0.24	0.2879	0.75	0.0	0	1	1

Figure 2: First few observations of day and hour datasets

3.3 Feature data types and Feature Statistics summary

For the day data set, we have 11 integer type columns and 4 float type columns. For the hour data set, we have 12 integer type columns and 4 float type columns.

<pre> day.info() <class 'pandas.core.frame.DataFrame'> DatetimeIndex: 731 entries, 2011-01-01 to 2012-12-31 Data columns (total 15 columns): # Column Non-Null Count Dtype --- --- 0 instant 731 non-null int64 1 season 731 non-null int64 2 yr 731 non-null int64 3 mnth 731 non-null int64 4 holiday 731 non-null int64 5 weekday 731 non-null int64 6 workingday 731 non-null int64 7 weathersit 731 non-null int64 8 temp 731 non-null float64 9 atemp 731 non-null float64 10 hum 731 non-null float64 11 windspeed 731 non-null float64 12 casual 731 non-null int64 13 registered 731 non-null int64 14 cnt 731 non-null int64 dtypes: float64(4), int64(11) memory usage: 91.4 KB </pre>	<pre> hour.info() <class 'pandas.core.frame.DataFrame'> DatetimeIndex: 17379 entries, 2011-01-01 to 2012-12-31 Data columns (total 16 columns): # Column Non-Null Count Dtype --- --- 0 instant 17379 non-null int64 1 season 17379 non-null int64 2 yr 17379 non-null int64 3 mnth 17379 non-null int64 4 hr 17379 non-null int64 5 holiday 17379 non-null int64 6 weekday 17379 non-null int64 7 workingday 17379 non-null int64 8 weathersit 17379 non-null int64 9 temp 17379 non-null float64 10 atemp 17379 non-null float64 11 hum 17379 non-null float64 12 windspeed 17379 non-null float64 13 casual 17379 non-null int64 14 registered 17379 non-null int64 15 cnt 17379 non-null int64 dtypes: float64(4), int64(12) memory usage: 2.3 MB </pre>
--	--

Figure 3: Data types of variables in day and hour data sets

The feature statistics summary of the datasets are as follows:

[] day.describe()

	instant	season	yr	mnth	holiday	weekday	workingday	weathersit	temp	atemp	hum	windspeed	casual	registered	cnt
count	731.000000	731.000000	731.000000	731.000000	731.000000	731.000000	731.000000	731.000000	731.000000	731.000000	731.000000	731.000000	731.000000	731.000000	731.000000
mean	366.000000	2.496580	0.500684	6.519836	0.028728	2.997264	0.683995	1.395349	0.495385	0.474354	0.627894	0.190486	848.176471	3656.172367	4504.348837
std	211.165812	1.110807	0.500342	3.451913	0.167155	2.004787	0.465233	0.544894	0.183051	0.162961	0.142429	0.077498	686.622488	1560.256377	1937.211452
min	1.000000	1.000000	0.000000	1.000000	0.000000	0.000000	0.000000	1.000000	0.059130	0.079070	0.000000	0.022392	2.000000	20.000000	22.000000
25%	183.500000	2.000000	0.000000	4.000000	0.000000	1.000000	0.000000	1.000000	0.337083	0.337842	0.520000	0.134950	315.500000	2497.000000	3152.000000
50%	366.000000	3.000000	1.000000	7.000000	0.000000	3.000000	1.000000	1.000000	0.498333	0.486733	0.626667	0.180975	713.000000	3662.000000	4548.000000
75%	548.500000	3.000000	1.000000	10.000000	0.000000	5.000000	1.000000	2.000000	0.655417	0.608602	0.730209	0.233214	1096.000000	4776.500000	5956.000000
max	731.000000	4.000000	1.000000	12.000000	1.000000	6.000000	1.000000	3.000000	0.861667	0.840896	0.972500	0.507463	3410.000000	6946.000000	8714.000000

[] hour.describe()

	instant	season	yr	mnth	hr	holiday	weekday	workingday	weathersit	temp	atemp	hum	windspeed	casual	registered	cnt
count	17379.00000	17379.000000	17379.000000	17379.000000	17379.000000	17379.000000	17379.000000	17379.000000	17379.000000	17379.000000	17379.000000	17379.000000	17379.000000	17379.000000	17379.000000	17379.000000
mean	8690.0000	2.501640	0.502561	6.537775	11.546752	0.028770	3.003683	0.682721	1.425283	0.496987	0.475775	0.627229	0.190098	35.676218	153.786869	189.463088
std	5017.0295	1.106918	0.500008	3.438776	6.914405	0.167165	2.005771	0.465431	0.639357	0.192556	0.171850	0.192930	0.122340	49.305030	151.357286	181.387599
min	1.0000	1.000000	0.000000	1.000000	0.000000	0.000000	0.000000	0.000000	1.000000	0.020000	0.000000	0.000000	0.000000	0.000000	0.000000	1.000000
25%	4345.5000	2.000000	0.000000	4.000000	6.000000	0.000000	1.000000	0.000000	1.000000	0.340000	0.333300	0.480000	0.104500	4.000000	34.000000	40.000000
50%	8690.0000	3.000000	1.000000	7.000000	12.000000	0.000000	3.000000	1.000000	1.000000	0.500000	0.484800	0.630000	0.194000	17.000000	115.000000	142.000000
75%	13034.5000	3.000000	1.000000	10.000000	18.000000	0.000000	5.000000	1.000000	2.000000	0.660000	0.621200	0.780000	0.253700	48.000000	220.000000	281.000000
max	17379.0000	4.000000	1.000000	12.000000	23.000000	1.000000	6.000000	1.000000	4.000000	1.000000	1.000000	1.000000	0.850700	367.000000	886.000000	977.000000

Figure 4: Feature statistics summary of day and hour datasets

We can see that on the average, about 189 bikes are getting rented per hour about 4504 bikes are rented per day.

3.4 Checking for missing data

In order to create a reliable dataset we need to adopt Data Cleaning method so that we can increase the quality of our data set. We check for missing values in our datasets


 day.isnull().sum()	[] hour.isnull().sum()
instant 0	instant 0
season 0	season 0
yr 0	yr 0
mnth 0	mnth 0
holiday 0	hr 0
weekday 0	holiday 0
workingday 0	weekday 0
weathersit 0	workingday 0
temp 0	weathersit 0
atemp 0	temp 0
hum 0	atemp 0
windspeed 0	hum 0
casual 0	windspeed 0
registered 0	casual 0
cnt 0	registered 0
dtype: int64	cnt 0
	dtype: int64

Figure 5: Missing data checking

As we can see, there is no missing data in our datasets.

3.5 Modifying datasets for ease of use and Categorical data conversion

We rename some columns into more readable form and add a weekday names column to aid in EDA procedure later. We change the following names:

- yr : year
- hr : hour
- mnth : month
- weathersit : weather
- hum : humidity
- cnt : count

We add the column of weekday names and name id 'dayname'.

Renaming columns into readable names and adding weekday names

```
[ ] #renaming columns into readable names and adding weekday names
day.rename(columns = {'yr':'year', 'mnth':'month', 'weathersit':'weather', 'hum':'humidity', 'cnt':'count'}, inplace = True)
hour.rename(columns = {'yr':'year', 'hr':'hour', 'mnth':'month', 'weathersit':'weather', 'hum':'humidity', 'cnt':'count'}, inplace = True)
day.index.names = ['date']
hour.index.names = ['date']
day['dayname']=day.index.day_name()
hour['dayname']=hour.index.day_name()
```

Figure 6: Renaming columns and adding 'dayname'

The columns 'weekday', 'season', 'holiday', 'workingday', 'weather' represent categories of data and are required to be converted into category data type.

Categorical data conversion

```
[14] # Converting into categorical data
category_list = ['weekday', 'season', 'holiday', 'workingday', 'weather']
for var in category_list:
    day[var] = day[var].astype('category')
    hour[var] = hour[var].astype('category')
```

Figure 7: Converting columns having categories into categorical data type

The temp, atemp, humidity and windspeed are normalised. We convert them to their original value to get a better understanding of the data during EDA.

```
[ ] day['temp']=day['temp']*41
    hour['temp']=hour['temp']*41
    day['atemp']=day['atemp']*50
    hour['atemp']=hour['atemp']*50
    day['humidity']=day['humidity']*100
    hour['humidity']=hour['humidity']*100
    day['windspeed']=day['windspeed']*67
    hour['windspeed']=hour['windspeed']*67
```

Figure 8: Converting numeric columns to original value

4 Exploratory Data Analysis (EDA)

An exploratory data analysis is always helpful to get an insight about the data. So here we will try to visualise different variables by various plots and diagrams and try to analyze them.

4.1 Plot of count vs. average values across the categorical columns Weather, Season, Working Day and Holiday

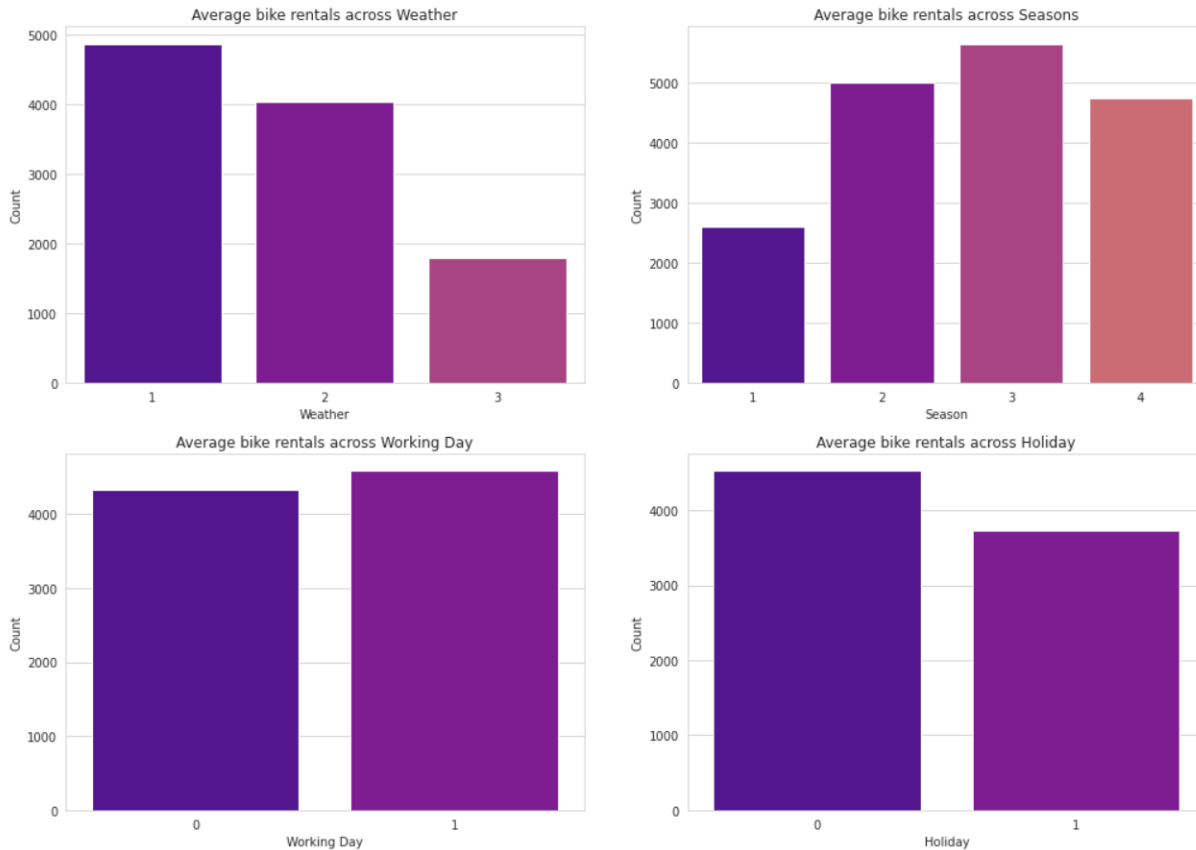


Figure 9: Plot of counts against Categorical variables

4.2 Plots of number of bikes rented vs. the temperature with the data split on Working day or Non working day

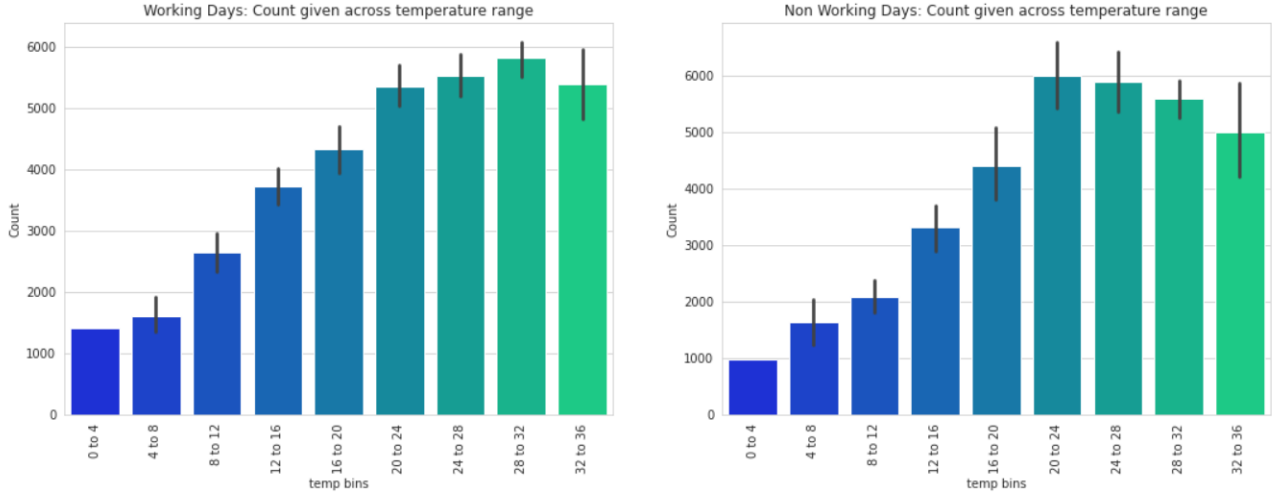


Figure 10: Average number of bikes rented vs. the temperature with the data split on Working day or Non working day

We see that there is a steady increase in the average number of bikes rented with temperature, with a small decrease in number at the highest temperature bin.

4.3 Hourly Distribution

We plot the average number of bikes rented per hour on the basis of weekdays.

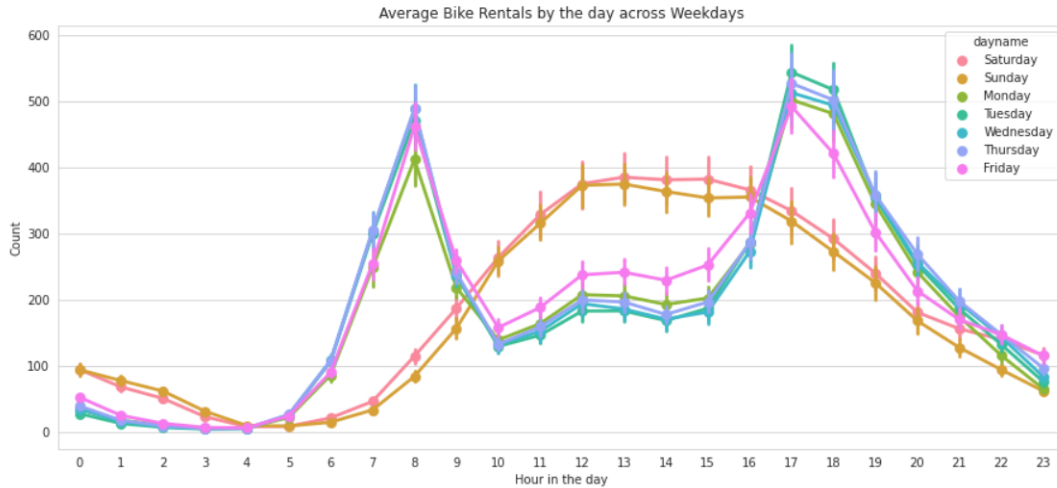


Figure 11: Average number of bikes rented per hour on the basis of weekdays

From this plot, we find that,

- Higher reservations can be seen at around 8am and 5pm (office hours) and close to 0 reservations very early in the morning
- Working Day: There is a peak in the rentals at around 8am and another at around 5pm. These may correspond to working people who typically are registered and go to work on working day which are usually Monday to Friday.
- Non Working Day: There is more or less a uniform rentals across the day with a peak at around 12 noon. These correspond to casual users who rent/drop off bikes uniformly during the day and tour the city on non working days which typically are Saturday and Sunday.

4.4 Monthly distribution

We plot the average number of bikes rented per month.

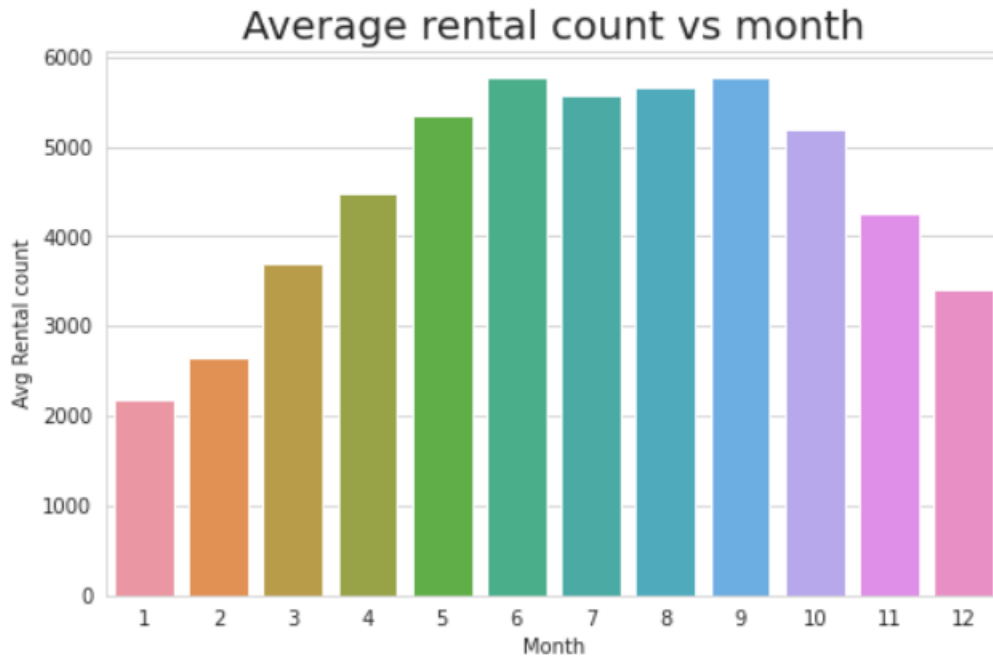


Figure 12: Average rental count per month

We see that the months of June and september have the most number of rentals while January has the least number of rentals.

Next we plot the seasonwise distribution of bike rental counts accross the months

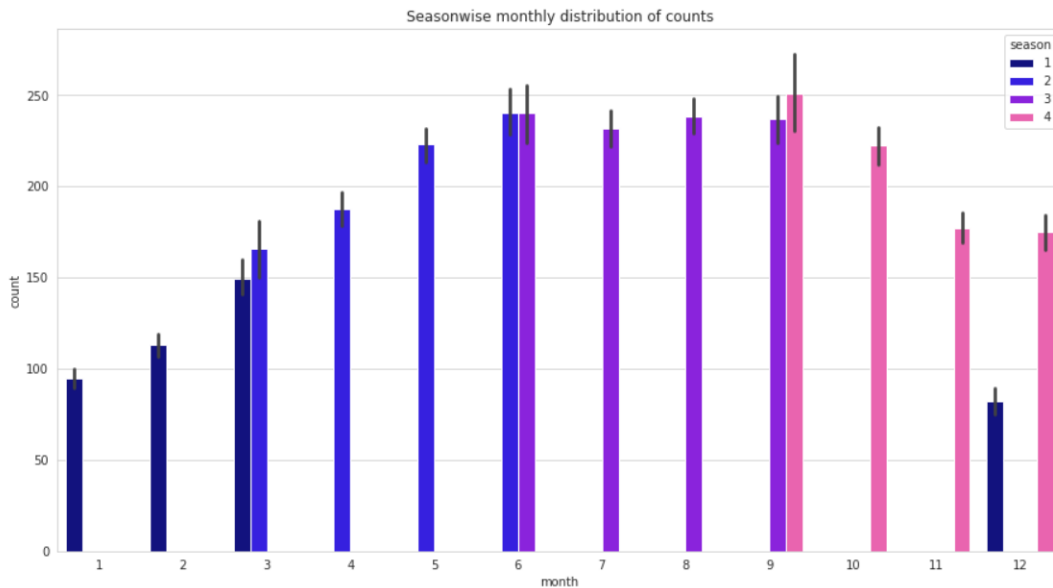


Figure 13: Seasonwise monthly distribution of counts

We have seen that June and September have the highest bike rentals. The month of June sees the transition from Summer to Fall and the month of September sees the transition from Fall to Winter. These are the times when bike rentals are at the highest counts.

4.5 Yearly Distribution

Now we plot the yearly distribution of the number of bikes rented in the years 2011 and 2012 and compare them.



Figure 14: Yearly distribution of bike rental counts

From the above violin plot, we observe that the bike rental count distribution is higher in year 2012 than it was in the year 2011. Also the probability of bikes being rented in the range of 4000-6000 is high in the year 2011 while in the year 2012, the probability is high for the range 6000-8000.

4.6 Regression Plots

We now construct regression plots of the count data with respect to the environmental conditions such as temperature, humidity and windspeed to see how they influence the bike rentals.

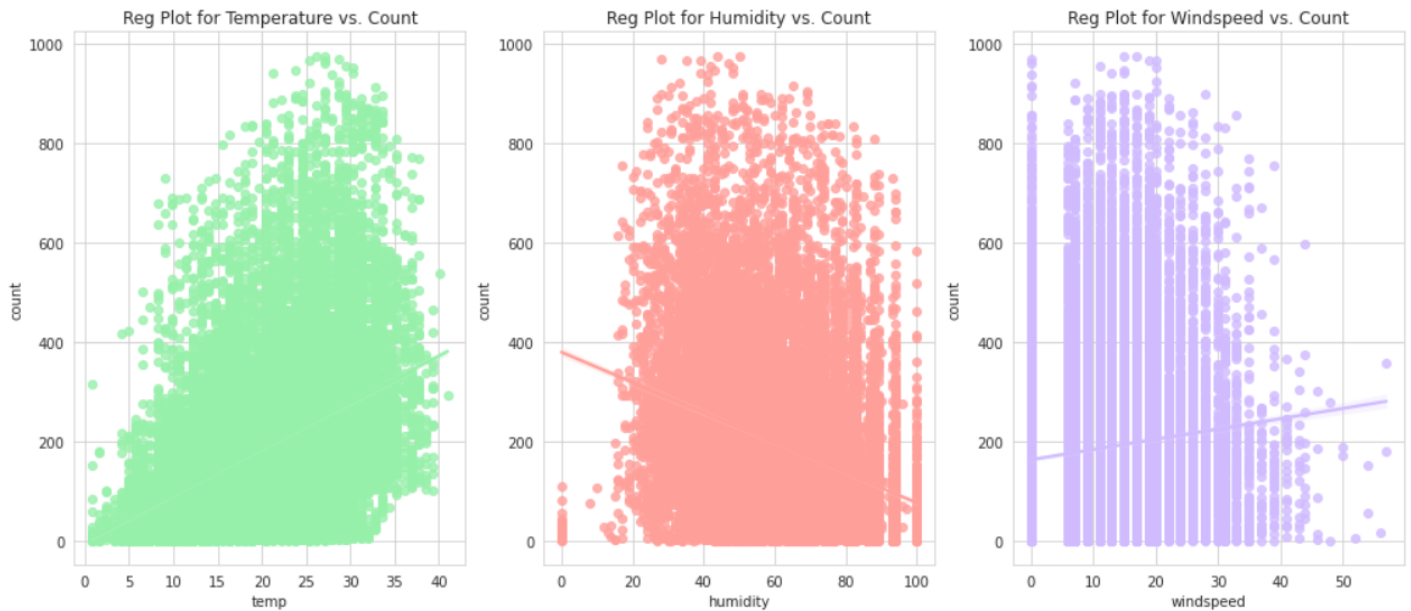


Figure 15: Regression plots of count data vs Temperature, Humidity and Windspeed

We find from the above plots that Temperature and Windspeed are positively related to the count data, while Humidity is negatively correlated.

4.7 Correlation analysis

We create a heat map to understand the correlation among the various columns of the dataset.

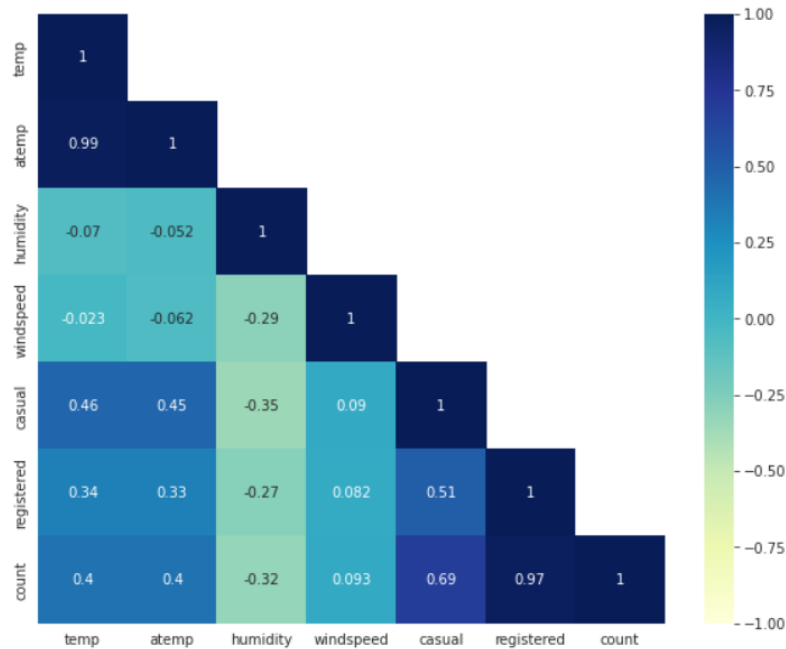


Figure 16: Correlation analysis heat map

We see that the variables atemp and temp have a high correlation. We will further concentrate on this during the multicollinearity check.

5 Time Series Analysis

Since our data is a time series data, we can perform certain time series analysis on the data too. We visualise our data and its various components the trend seasonal variations and residuals.

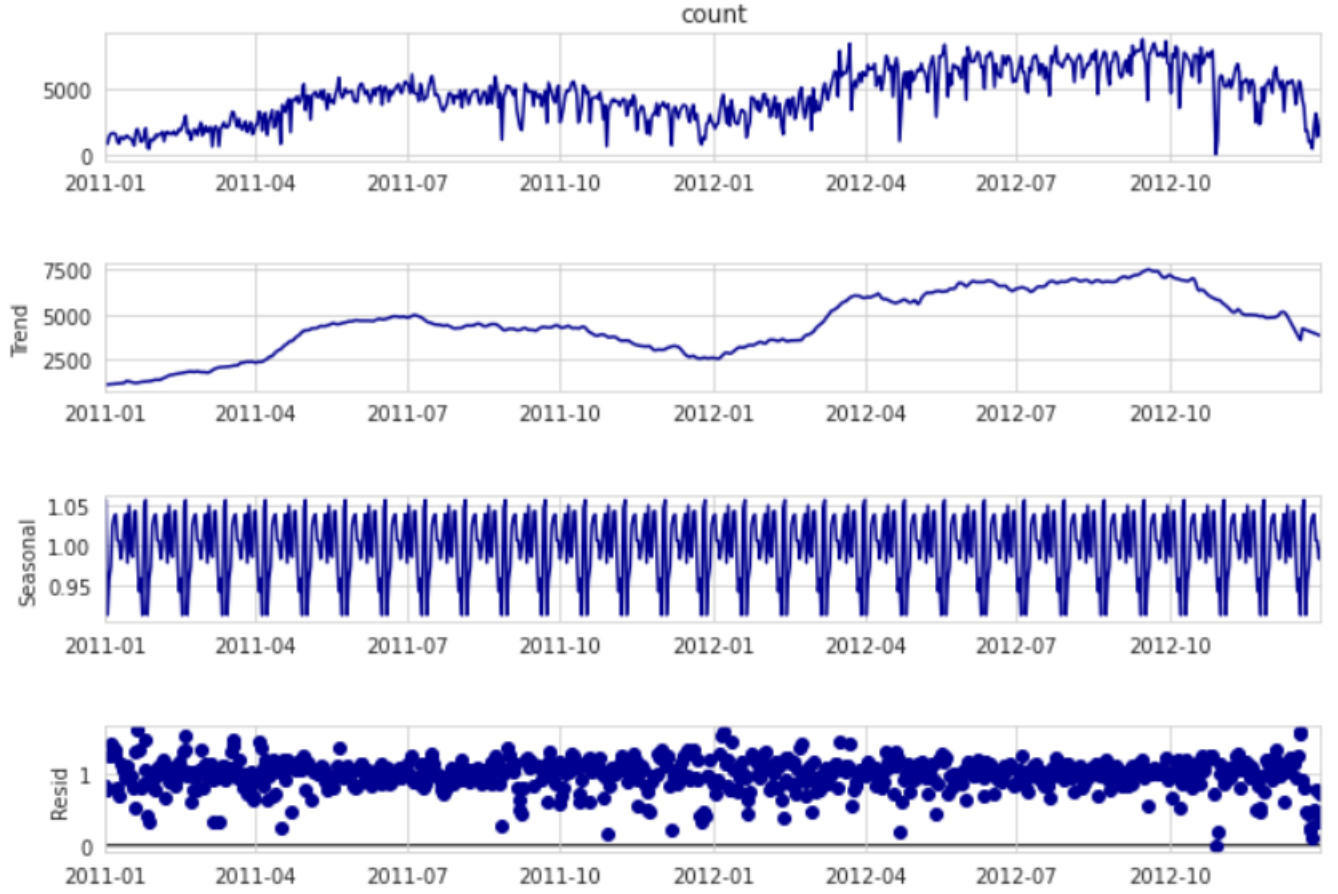


Figure 17: Count data and its various time series components

From the figure we can see a clear trend in data. Also a clear seasonality is visible. We perform the relative ordering test to assure the existence of trend and then we perform the Augmented Dickey Fuller test to test for stationarity in the residuals.

5.1 Test for Trend- Relative Ordering Test

To test,

H_0 : There is no trend in the series, against

H_1 : not H_0

Define,

$$q_{ij} = 1, \text{ if } X_i > X_j$$

$$= 0, \text{ otherwise}$$

Define

$$Q = \sum_i \sum_{j(i < j)} q_{ij}$$

Q is related to Kendall's τ , through the relationship,

$$\tau = 1 - \frac{4Q}{n(n-1)}$$

Under the null hypothesis of no trend, we can show, $E(\tau) = 0$, $Var(\tau) = \frac{2(2n+5)}{9n(n-1)}$

An asymptotic test for H_0 is given by,

$$Z = \frac{\tau - E(\tau)}{\sqrt{Var(\tau)}}$$

We would reject H_0 if $|Z| > \tau_{(\alpha/2)}$, where $\tau_{(\alpha/2)}$ is the upper $\alpha/2$ point of standard normal distribution.

```
#Relative Ordering Test Function
ro.test <- function (y = timeseries){
  n<-length(y)
  q<-0
  for(i in 1:(n-1))
  {
    for(j in (i+1):n)
      if(y[i]>y[j])
        q<-q+1
  }
  eq<-n*(n-1)/4
  tau<-1-(4*q/(n*(n-1)))
  var_tau<-(2*(2*n+5))/(9*n*(n-1))
  z<-tau/sqrt(var_tau)
  if(z>0){
    p_value<-1-pnorm(z)}
  if(z<0){
    p_value<-pnorm(z)}
  cat("      Relative Ordering Test for Presence of Trend \n\n")
  cat("Null Hypothesis: Absence of Trend, and \n")
  cat("Alternative Hypothesis: Presence of Trend. \n\n")
  cat("Test Statistic:",paste(round(z,4)), "\n")
  cat("p_value:", paste(round(p_value,4)), "\n")
  cat("No. of Discordants:",paste(q), "\n")
  cat("Expected No. of Discordants:",paste(eq), "\n")
}
ro.test(rday$count)
```

Relative Ordering Test for Presence of Trend

Null Hypothesis: Absence of Trend, and
Alternative Hypothesis: Presence of Trend.

Test Statistic: 18.2337
p_value: 0
No. of Discordants: 73284
Expected No. of Discordants: 133407.5

Figure 18: Relative ordering test

Here, the p-value is coming out to be 0 (< 0.05). So, we reject H_0 and conclude that, trend is present.

5.2 Test for Stationarity- ADF Test

Augmented Dickey–Fuller Test (ADF) tests the null hypothesis that a unit root is present in a time series. Here, the alternative hypothesis is that the series is stationary. We consider the model as:

$$X_t = \phi_0 + \phi_1 X_{(t-1)} + \epsilon_t$$

To test,

$H_0: \phi_1 = 1$ (series is non- stationary) against,

$H_1: \phi_1 \leq 1$ (series is stationary)

It is easy to see that,

$$X_t - X_{(t-1)} = \phi_0 + (\phi_1 - 1)X_{(t-1)} + \epsilon_t$$

or,

$$\nabla X_t = \phi_0 + \phi^* X_{(t-1)} + \epsilon_t$$

Here, we compare the value of test statistic with the value of Dickey Fuller distribution. If the value of test statistic is less than the value of Dickey Fuller distribution, we reject the null hypothesis and conclude that the series is stationary.

```
# Dickey Fuller test for stationarity: residual for day data
from statsmodels.tsa.stattools import adfuller
X = residual
result = adfuller(X, autolag='AIC')
print(result)
print('ADF Statistic: %f' % result[0])
print('p-value: %f' % result[1])
print('Critical Values:')
for key, value in result[4].items():
    print('\t%s: %.3f' % (key, value))
if result[0] > result[4]["5%"]:
    print("Time Series is Stationary")
else:
    print("Time Series is Stationary")

(-9.07891394388408, 4.106675177701985e-15, 18, 712, {'1%': -3.4395677423210493, '5%': -2.865607873
ADF Statistic: -9.078914
p-value: 0.000000
Critical Values:
    1%: -3.440
    5%: -2.866
    10%: -2.569
Time Series is Stationary
```

Figure 19:

Here, we have got, $p\text{-value}=0.00(<0.05)$, so we reject H_0 and conclude that the series is stationary.

6 OLS Fitting

We first drop the columns which are not required. The casual and registered columns are dropped as we only wish to predict the total bike rental count. We drop the year, month, day and hour columns as these information are already available in the index. For the hour data, we add the hour variable to the date index. We

6.1 Correlation Among Categorical Variables

We check for correlation among the categorical variables, season, holiday, weekday, workingday and weather. We use Cramer's V function for this.

Cramer's V :

Cramér's V is an effect size measurement for the chi-square test of independence. It measures how strongly two categorical fields are associated.

The effect size is calculated in the following manner:

- Determine which field has the fewest number of categories.
- Subtract 1 from the number of categories in this field.
- Multiply the result by the total number of records.
- Divide the chi-square value by the previous result. The chi-square value is obtained from the chi-square test of independence

- Take the square root.

Cramer's $V = \sqrt{\frac{(\chi^2/n)}{\min(c-1, r-1)}}$
 where

- χ^2 : The Chi-square statistic
- n : Total sample size
- r : Number of rows
- c : Number of columns

We create a matrix showing the association values of all the categorical

	season	holiday	weekday	workingday	weather
season	1.00	0.05	0.01	0.03	0.10
holiday	0.05	0.98	0.28	0.24	0.04
weekday	0.01	0.28	1.00	0.94	0.10
workingday	0.03	0.24	0.94	1.00	0.06
weather	0.10	0.04	0.10	0.06	1.00

Figure 20: Cramer's V matrix

From the matrix we see that there is high association between workingday and weekday variables. This is because in general, weekdays sunday (0) and saturday(6) gives non working days (0), while the other days 1-5 give workingdays(1).

So, weekday and workingday variables have high correlation and we need to drop one. We drop weekday variable as it would produce 6 dummy variables and hence increase the number of regressors which in turn would lessen the degrees of freedom and increase the SSE.

6.2 Dealing with Multicollinearity

Multicollinearity refers to a situation in which more than two explanatory variables in a multiple regression model are highly linearly related. There can be more than one reason behind multicollinearity, such as:

- The data collection method employed
- Model specification using too many regressors
- An over-defined model etc.

The consequences of multicollinearity being present in the model can be severe. When one or more regressors are linearly related with each other, the design matrix becomes ill-conditioned producing regression coefficients with large standard errors which can potentially damage the prediction capability of the model. There can be other problems like significant variable becoming insignificant one or regression coefficients appearing with wrong signs from what is expected.

Detection:

There are several methods for knowing the presence of multicollinearity in the model. One such method is to calculate the VIFs of the model.

VIF or Variance Inflation Factor for the j -th regressor is defined as:

$$VIF_j = \frac{1}{1 - R_j^2}$$

Where R_j^2 is the multiple R^2 obtained from regressing X_j on other regressors. The VIF value of 5 or more is an indicator of multicollinearity. Large values of VIF indicate multicollinearity leading to poor estimates of associated regression coefficients.

We next check for multicollinearity among the variables atemp, temp, humidity, windspeed. We have seen before from the correlation map that there is a high likelihood of multicollinearity being present the preliminary model.

```
## {r}
library(car)
vif(model1)
```

	atemp	temp	humidity	windspeed
	63.632351	62.969819	1.079267	1.126768

Figure 21: VIFs of regressors of preliminary model

As we can see from the above R snippet, the VIFs of atemp and temp are high indicating the presence of multicollinearity in the model.

Multicollinearity Diagnostics with Variance Decomposition:

After knowing the presence of multicollinearity in our model, we would like to know the group(s) of variables responsible for it. For doing this we can use Variance Decomposition Method.

Variance Decomposition Method is a method to identify subsets that are involved in multi-collinearity. Variance decomposition proportions, defined as

$$\pi_{kj} = \frac{\frac{v_{kj}^2}{l_k}}{\sum_{k=1}^p \frac{v_{kj}^2}{l_k}}, \forall k, j = 1(1)p$$

where, l_1, l_2, \dots, l_p are eigen values of $X^T X$ and v_1, v_2, \dots, v_p are corresponding orthonormal eigen vectors and $v_j = (v_{j1}, v_{j2}, \dots, v_{jp})^T, j = 1(1)p$.

Now a variance decomposition table is formed with the π_{kj} values along with a column containing the corresponding condition indices arranged in ascending order. So, large proportion in a row corresponding to the maximum condition index indicates the presence of multicollinearity among the corresponding regressors.

Step1:

```
## {r}
library(mctest)
eigprop(model1)
```

Call:
eigprop(mod = model1)

	Eigenvalues	CI (Intercept)	atemp	temp	humidity	windspeed
1	4.6924	1.0000	0.0010	0.0001	0.0001	0.0020
2	0.2009	4.8331	0.0022	0.0015	0.0020	0.0001
3	0.0890	7.2616	0.0224	0.0014	0.0025	0.2684
4	0.0168	16.7055	0.8825	0.0000	0.0010	0.7270
5	0.0009	72.7557	0.0919	0.9969	0.9944	0.0025

=====
Row 5==> atemp, proportion 0.996934 >= 0.50
Row 5==> temp, proportion 0.994400 >= 0.50
Row 4==> humidity, proportion 0.727028 >= 0.50

Figure 22: Variance Decompositon Table on model 1

- The subset (atemp,temp) and are involved in multicollinearity and humidity has proportion of variability greater than 0.5
- atemp has the highest VIFs in the subset (atemp,temp).
- We remove atemp and again fit the model.

Step2:

```
{r}
eigprop(model2)

Call:
eigprop(mod = model2)

Eigenvalues      CI (Intercept) windspeed    temp humidity
1      3.7490    1.0000      0.0017      0.0083 0.0076    0.0031
2      0.1586    4.8613      0.0000      0.5350 0.2201    0.0143
3      0.0755    7.0457      0.0201      0.1220 0.6760    0.2572
4      0.0168   14.9382      0.9782      0.3348 0.0963    0.7254

=====
Row 2==> windspeed, proportion 0.534967 >= 0.50
Row 3==> temp, proportion 0.676021 >= 0.50
Row 4==> humidity, proportion 0.725363 >= 0.50
```

Figure 23: Variance Decompositon Table on model 2

- Although the highest condition index is 14.9382, but the proportion of variability of windspeed, temp and humidity are much than 0.5.
- We will check for the VIFs to decide whether to keep or drop any variable.

```
{r}
vif(model2)

windspeed    temp    humidity
1.084580    1.034283    1.074850
```

Figure 24: VIFs of regressors of second model after dropping atemp variable

- All VIFs are less than 5 now.
- No multicollinearity in data anymore.

6.3 Outlier Detection: Leverage and Influential Points

Detection of Leverage Points :

A leverage point is determined on the basis of location of the point on the x-space and hence remote points impart more effect on the parameters of the model. A point is considered to be a leverage point if:

$$h_{ii} > 2p/n$$

where $h_{ii} = (i, i)^{th}$ element of $X(X^T X)^{-1} X^T$, p is the total number of variables in our model (both response and regressors).

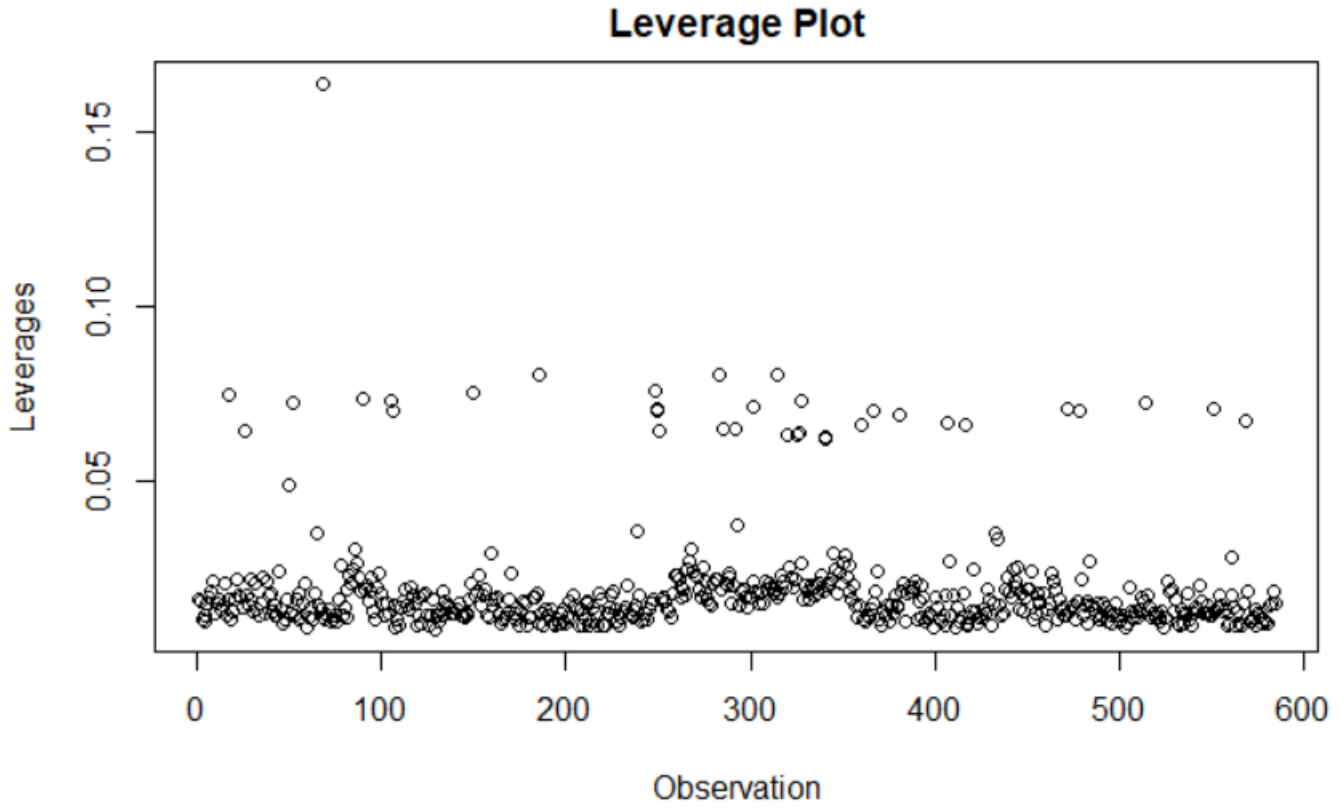


Figure 25: Plot for checking leverages

As we can see, quite a many points apparent have large leverages.

Detection of Influential points:

We use Cook's Distance or Cook's D to estimate the influence of a data point. Technically, Cook's D is calculated by removing the i^{th} data point from the model and recalculating the regression. It summarizes the extent to which all the values in the regression model change when the i^{th} observation is removed. The formula for Cook's distance is:

$$D_i = \frac{\sum_{j=1}^n (\hat{Y}_j - \hat{Y}_{j(i)})^2}{(p+1)s^2}$$

where, $\hat{Y}_{j(i)}$ is the fitted response value obtained when excluding the i^{th} regressor, and s^2 is the MSE. Any point having $D_i > 4/n$ can be investigated to be an influential point and in this case we will be considering them as such.

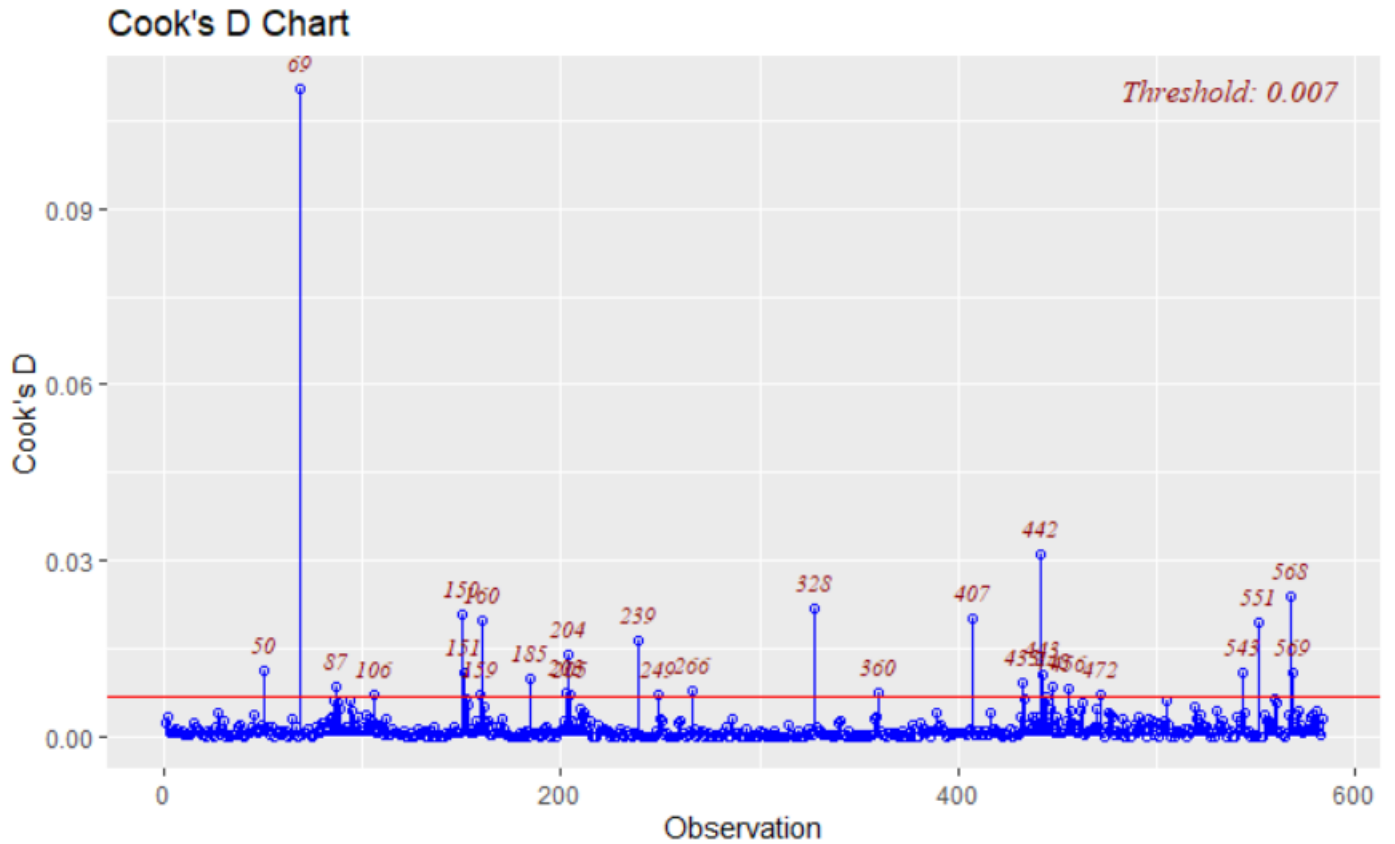


Figure 26: Plot for Cook's Distance

We see that there are substantial number of points which can be regarded as influential points.

Since we are interested in finding the true nature of the dependence of response on the regressors, we will drop these points.

28 data points are dropped which is about 4.79% of our dataset. We fit the MLR model again on this reduced set of data points.

6.4 Inspection of the Normality Assumption of Errors

6.4.1 Q-Q Plot

In this method we would plot the the ordered residuals $e_{(i)}$ against $\Phi^{-1}(\frac{i-0.5}{n})$, $i=1,2,\dots,n$. If the errors are truly from Normal Distribution then the plot will be nearly a straight line.

```
[58] import scipy.stats as stats
      sm.qqplot(model4.resid,line='45',fit=True,dist=stats.norm)
      plt.show()
```

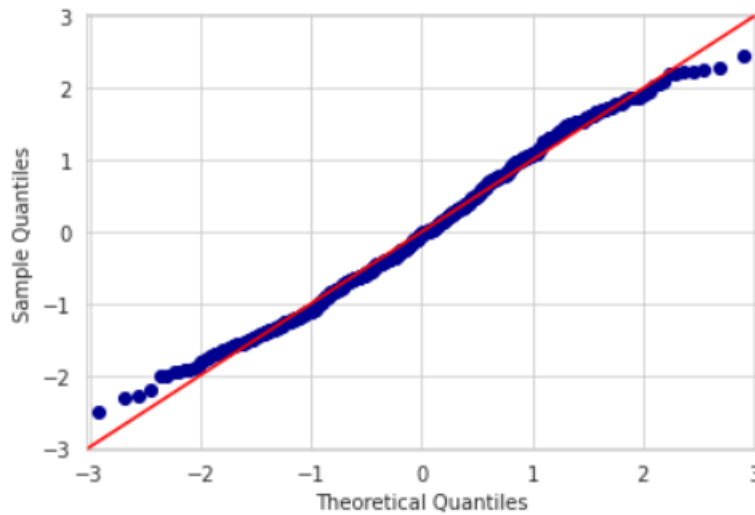


Figure 27: Q-Q Plot

The above Q-Q plot doesnot yield a straight line and the ends seem to deviate away. So, it can be concluded that the residuals do not follow a Normal Distribution.

6.4.2 Histogram Approach

```
[59] #Histogram approach of normality checking
_, bins, _ = plt.hist(model4.resid,density=1,alpha=0.5,ec='k')
plt.xlabel('Residuals')
mu, sigma = stats.norm.fit(model4.resid)
best_fit_line = stats.norm.pdf(bins, mu, sigma)
plt.plot(bins, best_fit_line)
```

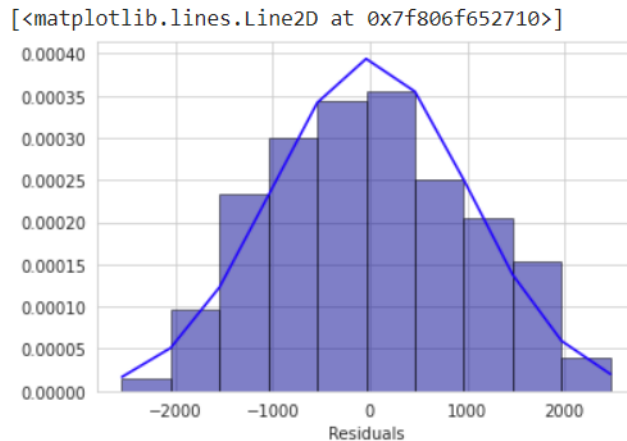


Figure 28: Histogram Plot

The histogram of Residuals seems significantly different from a Normal Curve. From here we could have concluded that our normality assumption for errors donot hold, but we will apply ShaiproWilk Test for Normality to get the final conclusion.

6.4.3 Shapiro-Wilk Test for Normality

Here the null hypothesis is,

H_0 : Errors are normally distributed

The Alternative hypothesis is

H_1 : H_0 is not true

The test Statistic is:

$$W = \frac{\sum_{i=1}^n a_i \hat{e}_{(i)}}{\sum_{i=1}^n (\hat{e}_i - \bar{e})^2}$$

Here, \hat{e}_i are the i^{th} fitted residual

$\hat{e}_{(i)}$ is the i^{th} order statistic

\bar{e} is the sample mean

$(a_1, a_2, \dots, a_n) = \frac{m^T V^{-1}}{C}$

And, $C = (m^T V^{-1} V^{-1} m)^{1/2}$

Here m is made of the expected values of the order statistics of independent and identically distributed random variables sampled from the standard normal distribution; finally, V is the covariance matrix of those normal order statistics. If p-value is greater than chosen level of significance, null hypothesis is accepted (i.e. distribution of error is not significantly different from a normal population).

```
from scipy.stats import shapiro
stat, p = shapiro(model4.resid)
print(f"shapiro statistic: {stat}")
print(f"p-value: {p}")
```

```
shapiro statistic: 0.9896210432052612
p-value: 0.000574766076169908
```

Figure 29: Shapiro wilk Test statistic and p-value

We see that the p-value = $0.000574 < 0.005(\alpha)$

So we reject the null hypothesis at 5% level of significance and conclude on the basis of the given data that the distribution of errors is different from Normal Distribution. So, our assumption is true.

We use the boxcox transformation to transform the dependent variable, and we fit the model again.

```
from scipy import stats
fitted_data, lmbda = stats.boxcox(y_train)
Xc=sm.add_constant(X_train)
model5= sm.OLS(fitted_data,Xc).fit()
print(model5.summary())
```

```

                        OLS Regression Results
=====
Dep. Variable:          y      R-squared:                0.678
Model:                  OLS    Adj. R-squared:           0.673
Method:                 Least Squares    F-statistic:       115.0
Date:                   Mon, 25 Jul 2022    Prob (F-statistic): 2.72e-127
Time:                   12:01:19    Log-Likelihood:    -3848.6
No. Observations:       556    AIC:                7719.
Df Residuals:           545    BIC:                7767.
Df Model:                10
Covariance Type:        nonrobust
=====
                        coef    std err          t      P>|t|      [0.025     0.975]
-----
const                902.3050      77.101     11.703     0.000     750.854    1053.756
temp                 41.8863       2.624     15.965     0.000      36.733     47.040
humidity             -7.0623       1.032     -6.845     0.000     -9.089     -5.036
windspeed           -12.8017       2.315     -5.531     0.000    -17.348     -8.255
season_2             202.4498      37.385      5.415     0.000     129.013     275.886
season_3            -85.6371      53.177     -1.610     0.108    -190.095     18.821
season_4             136.5838      36.923      3.699     0.000      64.055     209.113
holiday_1            -88.3344      81.213     -1.088     0.277    -247.863     71.195
workingday_1         29.1016      23.635      1.231     0.219    -17.325     75.528
weather_2            -65.8208      28.787     -2.286     0.023    -122.368     -9.274
weather_3           -373.3607      80.853     -4.618     0.000    -532.183    -214.538
=====
Omnibus:                 32.861    Durbin-Watson:           0.499
Prob(Omnibus):            0.000    Jarque-Bera (JB):        12.926
Skew:                     0.059    Prob(JB):                0.00156
Kurtosis:                 2.262    Cond. No.                 600.
=====

```

Notes:

[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

Figure 30: Box cox transformation of dependent variable.

6.5 Inspection of Autocorrelation among the Errors

The Durbin-Watson test uses the following hypotheses:

H_0 (null hypothesis) : There is no correlation among the residuals.

H_A (alternative hypothesis) : The residuals are autocorrelated.

The test statistic for the Durbin-Watson test, typically denoted d , is calculated as follows:

$$d = \frac{\sum_{t=2}^n (e_t - e_{t-1})^2}{\sum_{t=1}^n e_t^2}$$

where,

n =total number of observations,

e_t = The t^{th} residual from the regression model.

The test statistic always ranges from 0 to 4 where:

- $d = 2$ indicates no autocorrelation
- $d < 2$ indicates positive serial correlation
- $d > 2$ indicates negative serial correlation

```
{r}
library(car)
durbinWatsonTest(model5)

lag Autocorrelation D-W Statistic p-value
1      0.7795938      0.4391086      0
Alternative hypothesis: rho != 0
```

Figure 31: Durbin Watson Test

We can see that the DW statistic is 0.439 and the $p\text{-value} < 0.05(\alpha)$. So, we can say that autocorrelation is present in the data.

6.6 Inspection of Homoscedastic Assumptions of Errors

6.6.1 Residual vs Fitted Plot

Here we plot the residuals against the fitted responses. If the errors are homoscedastic then we would expect a horizontal band and completely random pattern around $\hat{e}_i = 0$ line. If any pattern is detected this will indicate that the variances may be non constant.

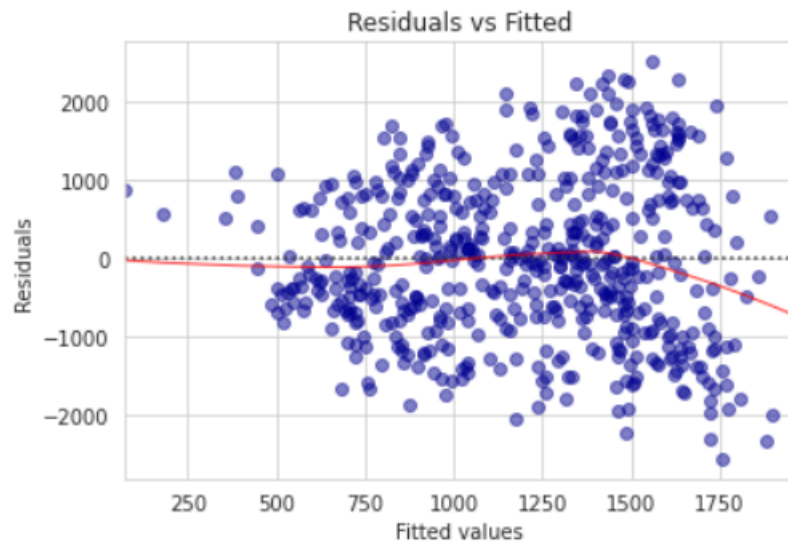


Figure 32: Residuals vs. Fitted Plot

The pattern obtained does not seem random about the horizontal band. We can assume that heteroscedasticity is present in the model.

6.6.2 Breusch-Pagan Test for Heteroscedasticity

The Breusch-Pagan test is used to determine whether or not heteroscedasticity is present in a regression model.

The test uses the following null and alternative hypotheses:

Null Hypothesis (H_0): Homoscedasticity is present (the residuals are distributed with equal variance) Alternative Hypothesis (H_A): Heteroscedasticity is present (the residuals are not distributed with equal variance)

If the p-value of the test is less than some significance level (i.e. $\alpha = .05$) then we reject the null hypothesis and conclude that heteroscedasticity is present in the regression model.

We use the following steps to perform a Breusch-Pagan test:

1. Fit the regression model.
2. Calculate the squared residuals of the model.
3. Fit a new regression model, using the squared residuals as the response values.
4. Calculate the Chi-Square test statistic χ^2 as $n * R_{new}^2$ where:
 - n : The total number of observations
 - R_{new}^2 : The R-squared of the new regression model that used the squared residuals as the response values

If the p-value that corresponds to this Chi-Square test statistic with p (the number of predictors) degrees of freedom is less than some significance level (i.e. $\alpha = .05$) then we reject the null hypothesis and conclude that heteroscedasticity is present. Otherwise, fail to reject the null hypothesis. In this case, it's assumed that homoscedasticity is present.

```
#from statsmodels.compat import lzip
import statsmodels.stats.api as sms
#perform Bresuch-Pagan test
names = ['Lagrange multiplier statistic', 'p-value',
         'f-value', 'f p-value']
test = sms.het_breuschpagan(model5.resid, model5.model.exog)
if test[1]<0.05:
    print('Homoscedasticiy is not present')
else:
    print('Homoscedasticiy is present')

#lzip(names, test)

Homoscedasticiy is not present
```

Figure 33: Breusch-Pagan Test for Heteroscedasticity

We can see that Heteroscedasticity is present in our model.

6.7 Variable Selection

When we fit a MLR model, we use the p-value in the ANOVA table to determine whether the model, as a whole, is significant. A natural question arises which regressors, among a larger set of all potential regressors, are important. We could use the individual p-values of the regressors and refit the model with only significant terms. But the p-values of the regressors are adjusted for the other terms in the model. So, picking out the subset of significant regressors can be somewhat challenging. This procedure of identifying the best subset of regressors to include in the model, among all possible subsets of regressors, is referred to as variable selection.

One approach is to start with a model containing only the intercept. Then using some chosen model fit criterion we slowly add terms to the model, one at a time, whose inclusion gives the most statistically significant improvement of the the model, and repeat this process until none improves the model to a statistically significant extent. This procedure is referred to as forward selection.

Another alternative is backward elimination. Here we start with the full model, then based on some model fit criterion we slowly remove variables one at a time, whose deletion gives the most statistically insignificant deterioration of the model fit, and repeat this process until no further variables can be deleted without a statistically insignificant loss of fit.

A third classical approach is stepwise selection. This is a combination of forward selection (FS) and backward elimination (BE). We start with FS, but at each step we recheck all regressors already entered, for possible deletion by BE method, this is because of the fact that regressor added at an earlier step may now be unnecessary in presence of new regressor.

Here we use stepwise selection method based on AIC criterion to determine the best subset model.
Our MLR model is:

$$Y = X\beta + \epsilon$$

Where we assume that $\epsilon \sim N(0, \sigma^2)$ and $Y \sim N_n(X\beta, \sigma^2 I_n)$
The likelihood function given by,

$$L(\beta, \sigma^2 | y) = (2n)^{-n/2} (\sigma^2)^{-n/2} \exp -\frac{1}{2} (y - X\beta)^T (y - X\beta)$$

So, the general form of the penalized likelihood function is given by,

$$-2\ln \hat{L} + \text{penalty term}$$

$$= n\ln(SSRes) + \text{penalty term}$$

where, $\hat{L} = \max_{\beta, \sigma^2} L(\beta, \sigma^2 | y) = L(\hat{\beta}_{mle}, \hat{\sigma}_{mle}^2)$

The Akaike information criterion (AIC) is a measure of the relative quality of statistical models for a given dataset. Given a collection of models for the data, AIC estimates the quality of each model, relative to each of the other models. Hence, AIC provides a means for model selection.

AIC is founded on information theory: it offers a relative estimate of the information lost when a given model is used to represent the process that generates the data. In doing so, it deals with the trade-off between the goodness of fit of the model and the complexity of the model.

AIC does not provide a test of a model in the sense of testing a null hypothesis, so it can tell nothing about the absolute quality of the model. If all the candidate models fit poorly, AIC will not give any warning of that.

Suppose that we have a statistical model of some n data. Then the AIC value of the model is the given by,

$$AIC = -2\ln(\hat{L}) + 2k$$

where, k = The number of estimated parameters in the model

\hat{L} = The maximized value of the likelihood function for the model

At first we consider all the subset models excluding one regressor at a time, and calculate the AIC value for each of those subset models. Then we discard the variable for which the subset model has the minimum AIC value.

```

{r}
library(MASS)
AIC=stepAIC(model5, direction='both')

```

Start: AIC=10486.39
count ~ temp + humidity + windspeed + season_2 + season_3 + season_4 +
holiday_1 + workingday_1 + weather_2 + weather_3

	Df	Sum of Sq	RSS	AIC
- workingday_1	1	1491339	1206298926	10485
<none>			1204807586	10486
- weather_2	1	5680650	1210488237	10488
- season_3	1	6550410	1211357997	10488
- holiday_1	1	6825841	1211633428	10488
- windspeed	1	40248581	1245056167	10508
- season_2	1	44374148	1249181735	10511
- humidity	1	53665384	1258472970	10516
- weather_3	1	59510484	1264318071	10520
- season_4	1	161775607	1366583193	10576
- temp	1	277214702	1482022289	10636

Figure 34: Step 1

The method considered the full 10 parameter model in the first step. The AIC corresponding to the Full Model is 10486.39. In this step this method compares the AICs by discarding each variable from the full model with the AIC of the

full model. From the table it can be observed that, the AIC corresponding to the model with 14 regressors after discarding the workingday_1 variable is lower than the full model and also it is minimum among all 9 regressors model.

```
Step:  AIC=10485.3
count ~ temp + humidity + windspeed + season_2 + season_3 + season_4 +
        holiday_1 + weather_2 + weather_3
```

	Df	Sum of Sq	RSS	AIC
<none>			1206298926	10485
+ workingday_1	1	1491339	1204807586	10486
- weather_2	1	5325968	1211624893	10486
- season_3	1	6302187	1212601113	10487
- holiday_1	1	9094033	1215392958	10489
- windspeed	1	40624946	1246923871	10508
- season_2	1	43939902	1250238827	10509
- humidity	1	54406989	1260705915	10516
- weather_3	1	58754596	1265053521	10518
- season_4	1	161330371	1367629297	10575
- temp	1	281278954	1487577880	10636

Figure 35: Step 2

This step considers the subset model by discarding workingday_1 from the full model. The AIC corresponding to that model is 10485.3. If any one of the variables is discarded from the current subset model, the AIC is higher than the current model. So, no variable will be discarded any more, the current model is our final model. So, our best subset model chosen by AIC is given by,

`lm(formula = Y ~ temp + humidity + windspeed + season_2 + season_3 + season_4 + holiday_1 + weather_2 + weather_3, data = my_data).`

6.8 Final Model Fit

Since our model errors contains both heteroscedasticity and autocorrelation we use HAC(heteroscedasticity- and autocorrelation-consistent) method to fit our model.

```
fitted_data, fitted_lambda = stats.boxcox(y_train)
Xc=sm.add_constant(X_train)
model6= sm.OLS(fitted_data,Xc).fit(cov_type='HAC',cov_kws={'maxlags':1})
print(model6.summary())
```

```

=====
                        OLS Regression Results
=====
Dep. Variable:          y      R-squared:          0.678
Model:                  OLS    Adj. R-squared:      0.673
Method:                 Least Squares    F-statistic:      89.03
Date:                   Mon, 25 Jul 2022    Prob (F-statistic):  9.01e-108
Time:                   12:01:23    Log-Likelihood:    -3848.6
No. Observations:      556    AIC:              7719.
Df Residuals:          545    BIC:              7767.
Df Model:              10
Covariance Type:       HAC
=====

```

	coef	std err	z	P> z	[0.025	0.975]
const	902.3050	86.222	10.465	0.000	733.313	1071.297
temp	41.8863	3.462	12.100	0.000	35.102	48.671
humidity	-7.0623	1.207	-5.852	0.000	-9.427	-4.697
windspeed	-12.8017	2.361	-5.422	0.000	-17.430	-8.174
season_2	202.4498	53.773	3.765	0.000	97.056	307.843
season_3	-85.6371	71.571	-1.197	0.231	-225.913	54.639
season_4	136.5838	37.039	3.688	0.000	63.989	209.178
holiday_1	-88.3344	46.271	-1.909	0.056	-179.025	2.356
workingday_1	29.1016	25.779	1.129	0.259	-21.424	79.627
weather_2	-65.8208	29.883	-2.203	0.028	-124.390	-7.252
weather_3	-373.3607	56.848	-6.568	0.000	-484.781	-261.940

```

=====
Omnibus:                 32.861    Durbin-Watson:          0.499
Prob(Omnibus):           0.000    Jarque-Bera (JB):        12.926
Skew:                    0.059    Prob(JB):                0.00156
Kurtosis:                2.262    Cond. No.                600.
=====
Notes:
[1] Standard Errors are heteroscedasticity and autocorrelation robust (HAC) using 1 lags and without small sample correction

```

Figure 36: Fitting MLR model using HAC method

Comparing this regression with model5, we find that in both models the estimated coefficients and the R^2 value are the same. But, importantly, note that the HAC standard errors are much greater than the OLS standard errors and therefore the HAC t ratios are much smaller than the OLS t ratios. This shows that OLS had in fact underestimated the true standard errors. Curiously, the d statistics in both models is the same. But we donot need to worry as the HAC procedure has already taken this into account in correcting the OLS standard errors.

6.9 Predictions

Here are few of the predicted values of the test data calculated by the OLS model.

	y_pred
date	
2012-08-07	5135
2012-08-08	5355
2012-08-09	5675
2012-08-10	4511
2012-08-11	4289
	...
2012-12-27	1572
2012-12-28	2332
2012-12-29	1900
2012-12-30	2142
2012-12-31	2128
	Length: 147, dtype: int64

Figure 37: Predicted values from OLS model

7 Ridge Regression

Ridge regression is a model tuning method that is used to analyze any data that suffers from multicollinearity. This method performs L2 regularization. L2 regularization adds an L2 penalty, which equals the square of the magnitude of coefficients. Coefficients are shrunk by the same factor (so none are eliminated).

A tuning parameter (λ) controls the strength of the penalty term. When $\lambda = 0$, ridge regression equals least squares regression. If $\lambda = \infty$, all coefficients are shrunk to zero. The ideal penalty is therefore somewhere in between 0 and ∞ .

Ridge estimators theoretically produce new estimators that are shrunk closer to the “true” population parameters. The ridge function fitting the ridge regression is given by,

$$R(\beta) = \min_{\beta} (Y - X\beta)^T (Y - X\beta) + \lambda \beta^T \beta$$

OLS regression uses the following formula to estimate coefficients:

$$\hat{\beta} = (X^T X)^{-1} X^T Y$$

Ridge regression adds a product of ridge parameter & the identity matrix to the cross product matrix ($X^T X$), forming a new matrix ($X^T X + \lambda I$). The new formula is used to find the coefficients:

$$\tilde{\beta} = (X^T X)^{-1} X^T Y \quad \tilde{\beta} = (X^T X)^{-1} X^T Y \quad \tilde{\beta} = (X^T X)^{-1} X^T Y \quad \tilde{\beta} = (X^T X)^{-1} X^T Y$$

Here are few of the predicted values of the test data calculated by the Ridge regression model.

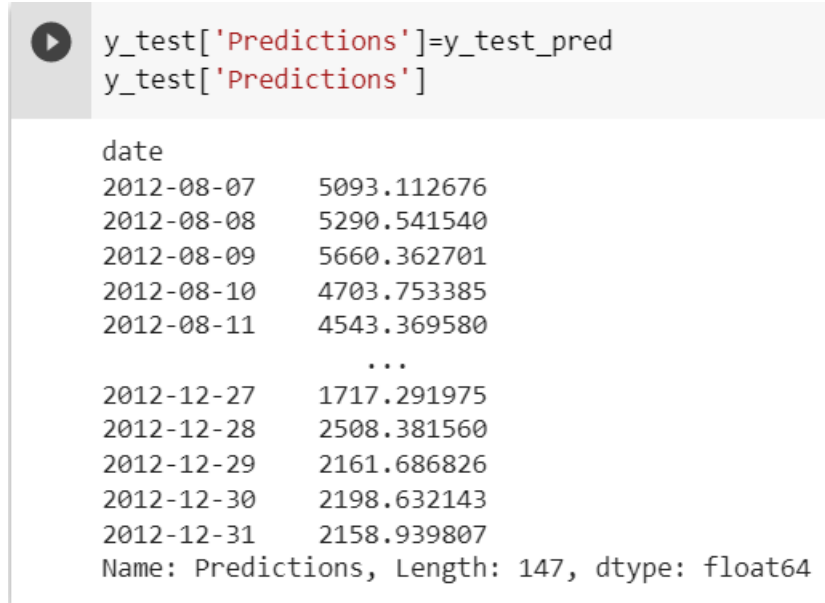


Figure 38: Few of the predicted values of the test data calculated by the Ridge regression model

8 Predictions and Error Metrics

After fitting our models we need to know which one would be the best model for predicting the results in future in our case. For this we use various error metrics to check for the better model.

First we check for the adequacy of the models

8.1 R^2 and Adjusted R^2

R^2 and Adjusted R^2 are used to explain the overall adequacy of the model, where,

$$R^2 = 1 - \frac{SS_{Res}}{SS_{Tot}}$$

$$R^2_{adj} = 1 - \frac{SS_{Res}/(n-p-1)}{SS_{Tot}/(n-1)}$$

8.2 Mean absolute percentage error(MAPE)

The mean absolute percentage error (MAPE), also known as mean absolute percentage deviation (MAPD), is a measure of prediction accuracy of a forecasting method in statistics. It usually expresses the accuracy as a ratio defined by the formula:

$$MAPE = \frac{1}{n} \sum_{t=1}^n \left| \frac{O_t - P_t}{O_t} \right| * 100\%$$

where, O_t = observed values

P_t = predicted values

8.3 Mean Absolute Deviation Percentage(MADP)

The mean absolute deviation percentage (MADP), also known as mean absolute error percentage (MAEP), is another measure of prediction accuracy of a model. It usually expresses the accuracy as a ratio defined by the formula:

$$MADP = \frac{\sum_{t=1}^n |O_t - P_t|}{\sum_{t=1}^n |O_t|} * 100\%$$

where, O_t = observed values

P_t = predicted values

8.4 Cumulative Error Percentage(CERRPCT)

The Cumulative Error Percentage(CERRPCT) is another way to measure the prediction accuracy of the model. It is given by the formula:

$$CERRPCT = \frac{\sum_{t=1}^n (O_t - P_t)}{\sum_{t=1}^n O_t} * 100\%$$

8.5 Comparing model adequacy and error metrices

ModelsMetrics	MLR Model	Ridge Regression
R^2	0.678	0.663
MAPE	80.30433	86.44154
MADP	34.19479	34.74267
CERRPCT	31.98366	32.24110

From the above table we see that The MLR Model gives a better fit to the data and also has lesser error in predicting bike counts. So, the MLR would be the preferred model in this case.