Bike Sharing Case Study

A bike-sharing system is a service in which bikes are made available for shared use to individuals on a short term basis for a price or free. Many bike share systems allow people to borrow a bike from a "dock" which is usually computer-controlled wherein the user enters the payment information, and the system unlocks it. This bike can then be returned to another dock belonging to the same system.

Problem Statement

A US bike-sharing provider **BoomBikes** has recently suffered considerable dips in their revenues due to the ongoing Corona pandemic. The company is finding it very difficult to sustain in the current market scenario. So, it has decided to come up with a mindful business plan to be able to accelerate its revenue as soon as the ongoing lockdown comes to an end, and the economy restores to a healthy state.

The company wants to know:

- Which variables are significant in predicting the demand for shared bikes.
- How well those variables describe the bike demands

Aim of the case study

To model the demand for shared bikes with the available independent variables. It will be used by the management to understand how exactly the demands vary with different features. They can accordingly manipulate the business strategy to meet the demand levels and meet the customer's expectations. Further, the model will be a good way for management to understand the demand dynamics of a new market.

Step 1: Reading and Understanding the Data

Let us first import the required libraries and then read the dataset

In [1]:

```
# importing libraries

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import calendar
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from sklearn.feature_selection import RFE
from sklearn.linear_model import LinearRegression
import statsmodels.api as sm
from statsmodels.stats.outliers_influence import variance_inflation_factor
```

```
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import r2_score

# Supress Warnings

import warnings
warnings.filterwarnings('ignore')
```

In [2]:
```
# reading dataset

df = pd.read_csv("day.csv")
df.head()
```

Out[2]:

	instant	dteday	season	yr	mnth	holiday	weekday	workingday	weathersit	temp	atemp
0	1	01-01-2018	1	0	1	0	6	0	2	14.110847	18.18125
1	2	02-01-2018	1	0	1	0	0	0	2	14.902598	17.68695
2	3	03-01-2018	1	0	1	0	1	1	1	8.050924	9.47025
3	4	04-01-2018	1	0	1	0	2	1	1	8.200000	10.60610
4	5	05-01-2018	1	0	1	0	3	1	1	9.305237	11.46350

In [3]:
```
# check the shape

df.shape
```

Out[3]:
```
(730, 16)
```

The dataset has 730 rows and 16 columns.

In [4]:
```
# check the columns

df.columns
```

Out[4]:
```
Index(['instant', 'dteday', 'season', 'yr', 'mnth', 'holiday', 'weekday',
       'workingday', 'weathersit', 'temp', 'atemp', 'hum', 'windspeed',
       'casual', 'registered', 'cnt'],
      dtype='object')
```

In [5]:
```
# check info about columns

df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 730 entries, 0 to 729
Data columns (total 16 columns):
 #   Column      Non-Null Count  Dtype  
---  ------      --------------  -----  
 0   instant     730 non-null    int64  
 1   dteday      730 non-null    object 
 2   season      730 non-null    int64  
 3   yr          730 non-null    int64  
 4   mnth        730 non-null    int64  
 5   holiday     730 non-null    int64  
 6   weekday     730 non-null    int64  
 7   workingday  730 non-null    int64  
 8   weathersit  730 non-null    int64  
 9   temp        730 non-null    float64
 10  atemp       730 non-null    float64
 11  hum         730 non-null    float64
 12  windspeed   730 non-null    float64
 13  casual      730 non-null    int64  
 14  registered  730 non-null    int64  
 15  cnt         730 non-null    int64  
dtypes: float64(4), int64(11), object(1)
memory usage: 91.4+ KB
```

We can see all the columns have 730 values which means there are no missing values.

In [6]:
```
# check if there is any null value

df.isnull().sum()
```

Out[6]:
```
instant       0
dteday        0
season        0
yr            0
mnth          0
holiday       0
weekday       0
workingday    0
weathersit    0
temp          0
atemp         0
hum           0
windspeed     0
casual        0
registered    0
cnt           0
dtype: int64
```

All columns have 0 null values.

Data Sanity Check

Let's check if there is any anomaly when we check the equation `casual + registered = cnt` as cnt should be equal to number of casual bookings plus regustered ones.

In [7]:
```
((df.cnt == df.casual + df.registered) == False).sum()
```

Out[7]: 0

So this shows there are no such values.

We can drop following columns:

- `instant` as it is an index
- `dteday` as it is redundant ; we already have mnth & yr
- `casual` & `registered` as these values will be populated when the user actually books the bike. These variable will not help in making predictions of bike booking.

In [8]:
```
df = df.drop( columns = ['instant','dteday','casual','registered'])
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 730 entries, 0 to 729
Data columns (total 12 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   season      730 non-null    int64
 1   yr          730 non-null    int64
 2   mnth        730 non-null    int64
 3   holiday     730 non-null    int64
 4   weekday     730 non-null    int64
 5   workingday  730 non-null    int64
 6   weathersit  730 non-null    int64
 7   temp        730 non-null    float64
 8   atemp       730 non-null    float64
 9   hum         730 non-null    float64
 10  windspeed   730 non-null    float64
 11  cnt         730 non-null    int64
dtypes: float64(4), int64(8)
memory usage: 68.6 KB
```

Now we are left with 12 columns.

In [9]:
```
# lets check the stats here on a high level

df.describe()
```

Out[9]:

	season	yr	mnth	holiday	weekday	workingday	weathersit	tem
count	730.000000	730.000000	730.000000	730.000000	730.000000	730.000000	730.000000	730.00000
mean	2.498630	0.500000	6.526027	0.028767	2.997260	0.683562	1.394521	20.31925
std	1.110184	0.500343	3.450215	0.167266	2.006161	0.465405	0.544807	7.50672
min	1.000000	0.000000	1.000000	0.000000	0.000000	0.000000	1.000000	2.42434
25%	2.000000	0.000000	4.000000	0.000000	1.000000	0.000000	1.000000	13.81188
50%	3.000000	0.500000	7.000000	0.000000	3.000000	1.000000	1.000000	20.46582
75%	3.000000	1.000000	10.000000	0.000000	5.000000	1.000000	2.000000	26.88061
max	4.000000	1.000000	12.000000	1.000000	6.000000	1.000000	3.000000	35.32834

Since the data set is clean, we can now proceed towards visualization of data.

Step 2: Visualising the Data

Let's now visualize the data in graphs and plots.

- We can check if there is some multicollinearity among the data
- Identify if some predictors directly have a strong association with the outcome variable

In [10]:

```
# Checking number of numerical and categorical values

numerical_var = df.dtypes[df.dtypes !='object'].index
print("Number of Numerical Variables : ", len(numerical_var))
print("Numerical variables : ",numerical_var)

categorical_var = df.dtypes[df.dtypes =='object'].index
print("Number of Categorical Variables : " ,len(categorical_var))
print("Categorical variables : ",categorical_var)
```

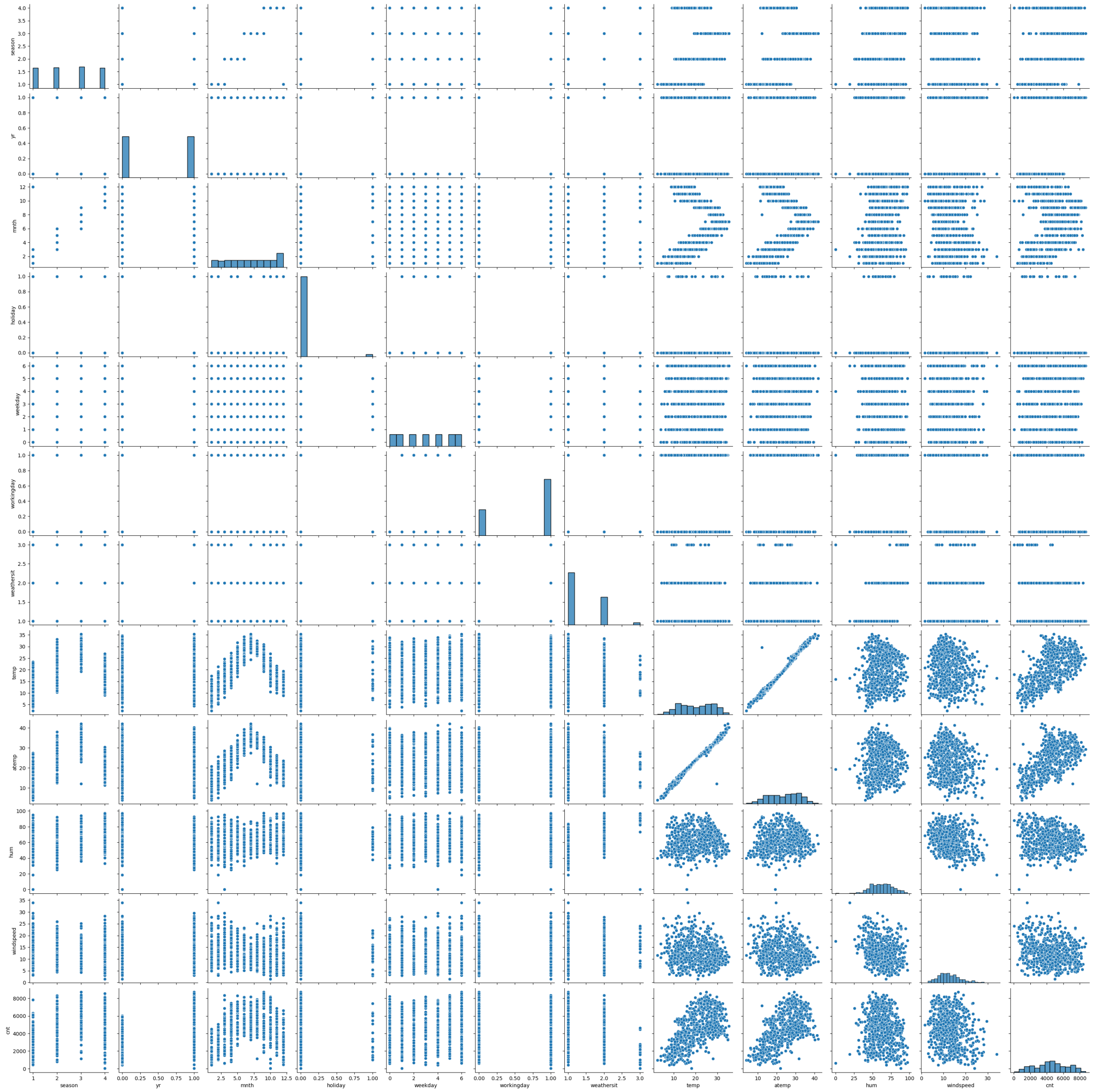
```
Number of Numerical Variables :  12
Numerical variables :  Index(['season', 'yr', 'mnth', 'holiday', 'weekday', 'workingd
ay',
       'weathersit', 'temp', 'atemp', 'hum', 'windspeed', 'cnt'],
      dtype='object')
Number of Categorical Variables :  0
Categorical variables :  Index([], dtype='object')
```

There are no categorical values as such but there are binary encoding values in some columns like workingday, holiday etc.

Lets do a pair plot first.

In [11]:

```
sns.pairplot(df)
plt.show()
```



Since this pairplot at this point of time is not giving any proper indication.
Lets work towards creating categorical variables for `season` , `weathersit` , `mnth` and `weekday` .

Season

In [12]:
```
# values in season column

df.season.value_counts()
```

Out[12]:
```
3    188
2    184
1    180
4    178
Name: season, dtype: int64
```

As per data dictionary :

1 : spring

2 : summer

3 : fall

4 : winter

In [13]:
```
# Defining the mapping function

def season_map(x):
    return x.map({1:'spring', 2:'summer', 3:'fall', 4:'winter'})

# Applying the function to the dataset
df[['season']] = df[['season']].apply(season_map)
df.season.value_counts()
```

Out[13]:
```
fall      188
summer    184
spring    180
winter    178
Name: season, dtype: int64
```

This is same as the earlier values.

Weathersit

In [14]:
```
# values in weathersit column

df.weathersit.value_counts()
```

Out[14]:
```
1    463
2    246
3     21
Name: weathersit, dtype: int64
```

As per data dictionary :

- 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

since the texts are very lengthy here, lets use following terms for each category:

- 1 : clear
- 2 : misty
- 3 : light_rain
- 4 : heavy_rain

In [15]:
```
# Defining the mapping function

def weather_map(x):
    return x.map({1:'clear', 2:'misty', 3:'light_rain', 4:'heavy_rain'})

# Applying the function to the dataset
```

```
df[['weathersit']] = df[['weathersit']].apply(weather_map)
df.weathersit.value_counts()
```

Out[15]:

```
clear          463
misty          246
light_rain      21
Name: weathersit, dtype: int64
```

Mnth

In [16]:

```
# values in mnth column

df.mnth.value_counts()
```

Out[16]:

```
1     62
3     62
5     62
7     62
8     62
10    62
12    62
4     60
6     60
9     60
11    60
2     56
Name: mnth, dtype: int64
```

Here the numbers denote the month of the year for e.g. 1 : Jan, 2: Feb.... 12: Dec
So lets convert them to categorical values as well.

In [17]:

```
df.mnth=df.mnth.apply(lambda x:calendar.month_abbr[x])
df.mnth.value_counts()
```

Out[17]:

```
Jan    62
Mar    62
May    62
Jul    62
Aug    62
Oct    62
Dec    62
Apr    60
Jun    60
Sep    60
Nov    60
Feb    56
Name: mnth, dtype: int64
```

Weekday

In [18]:

```
df.weekday.value_counts()
```

Out[18]:
```
6    105
0    105
1    105
2    104
4    104
5    104
3    103
Name: weekday, dtype: int64
```

In [19]:
```
df.head(10)
```

Out[19]:

	season	yr	mnth	holiday	weekday	workingday	weathersit	temp	atemp	hum	winds
0	spring	0	Jan	0	6	0	misty	14.110847	18.18125	80.5833	10.74
1	spring	0	Jan	0	0	0	misty	14.902598	17.68695	69.6087	16.65
2	spring	0	Jan	0	1	1	clear	8.050924	9.47025	43.7273	16.63
3	spring	0	Jan	0	2	1	clear	8.200000	10.60610	59.0435	10.73
4	spring	0	Jan	0	3	1	clear	9.305237	11.46350	43.6957	12.52
5	spring	0	Jan	0	4	1	clear	8.378268	11.66045	51.8261	6.00
6	spring	0	Jan	0	5	1	misty	8.057402	10.44195	49.8696	11.30
7	spring	0	Jan	0	6	0	misty	6.765000	8.11270	53.5833	17.87
8	spring	0	Jan	0	0	0	clear	5.671653	5.80875	43.4167	24.25
9	spring	0	Jan	0	1	1	clear	6.184153	7.54440	48.2917	14.95

Here, we can see that working day is 0 for weekday value 0 & 6. And it is 1 for rest. This tells weekday 0 is Sunday and 6 is Saturday. So let's do encoding according to this.

In [20]:
```
weekdays_dic ={0:'Sun', 1:'Mon',2:'Tue',3:'Wed',4:'Thu',5:'Fri', 6:'Sat'}

df[['weekday']]=df[['weekday']].apply(lambda x:x.map(weekdays_dic))
df.weekday.value_counts()
```

Out[20]:
```
Sat    105
Sun    105
Mon    105
Tue    104
Thu    104
Fri    104
Wed    103
Name: weekday, dtype: int64
```

In [21]:
```
df.head()
```

Out[21]:

	season	yr	mnth	holiday	weekday	workingday	weathersit	temp	atemp	hum	winds
0	spring	0	Jan	0	Sat	0	misty	14.110847	18.18125	80.5833	10.74
1	spring	0	Jan	0	Sun	0	misty	14.902598	17.68695	69.6087	16.65
2	spring	0	Jan	0	Mon	1	clear	8.050924	9.47025	43.7273	16.63
3	spring	0	Jan	0	Tue	1	clear	8.200000	10.60610	59.0435	10.73
4	spring	0	Jan	0	Wed	1	clear	9.305237	11.46350	43.6957	12.52

Since, data conversion to categorical values are done, lets look into the data visualization now.

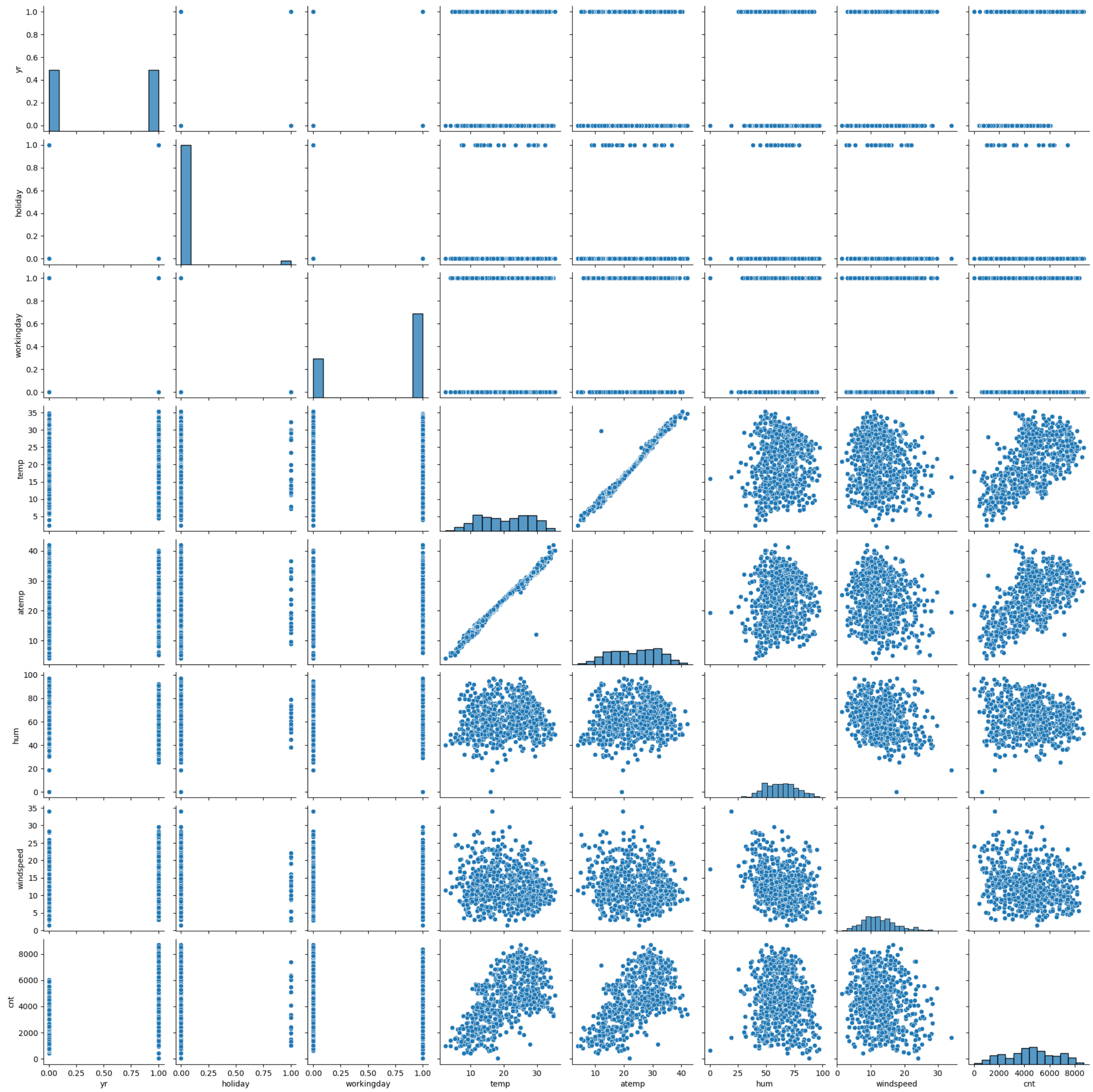
Lets do the pairplot again now

Visualising Numeric Variables

Let's make a pairplot of all the numeric variables

In [22]:

```
sns.pairplot(df)
plt.show()
```



Inference:

- temp and atemp are highly correlated : we can drop one of them going forward based on VIF and p-value.
- both temp and atemp shows linear relationship with target variable `cnt`
- increase in himidity is resulting in more bike bookings
- less wind speed is resulting in more bike bookings

In [23]:
```
# lets find the correlation matrix

df.corr()
```

Out[23]:

	yr	holiday	workingday	temp	atemp	hum	windspeed	cnt
yr	1.000000	0.008195	-0.002945	0.048789	0.047215	-0.112547	-0.011624	0.569728
holiday	0.008195	1.000000	-0.252948	-0.028764	-0.032703	-0.015662	0.006257	-0.068764
workingday	-0.002945	-0.252948	1.000000	0.053470	0.052940	0.023202	-0.018666	0.062542
temp	0.048789	-0.028764	0.053470	1.000000	0.991696	0.128565	-0.158186	0.627044
atemp	0.047215	-0.032703	0.052940	0.991696	1.000000	0.141512	-0.183876	0.630685
hum	-0.112547	-0.015662	0.023202	0.128565	0.141512	1.000000	-0.248506	-0.098543
windspeed	-0.011624	0.006257	-0.018666	-0.158186	-0.183876	-0.248506	1.000000	-0.235132
cnt	0.569728	-0.068764	0.062542	0.627044	0.630685	-0.098543	-0.235132	1.000000

In [24]:

```
# heatmap

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

Inference:

- This also shows `temp` & `atemp` are very much correlated (0.99)
- `Cnt` has a positive correlation with `yr`

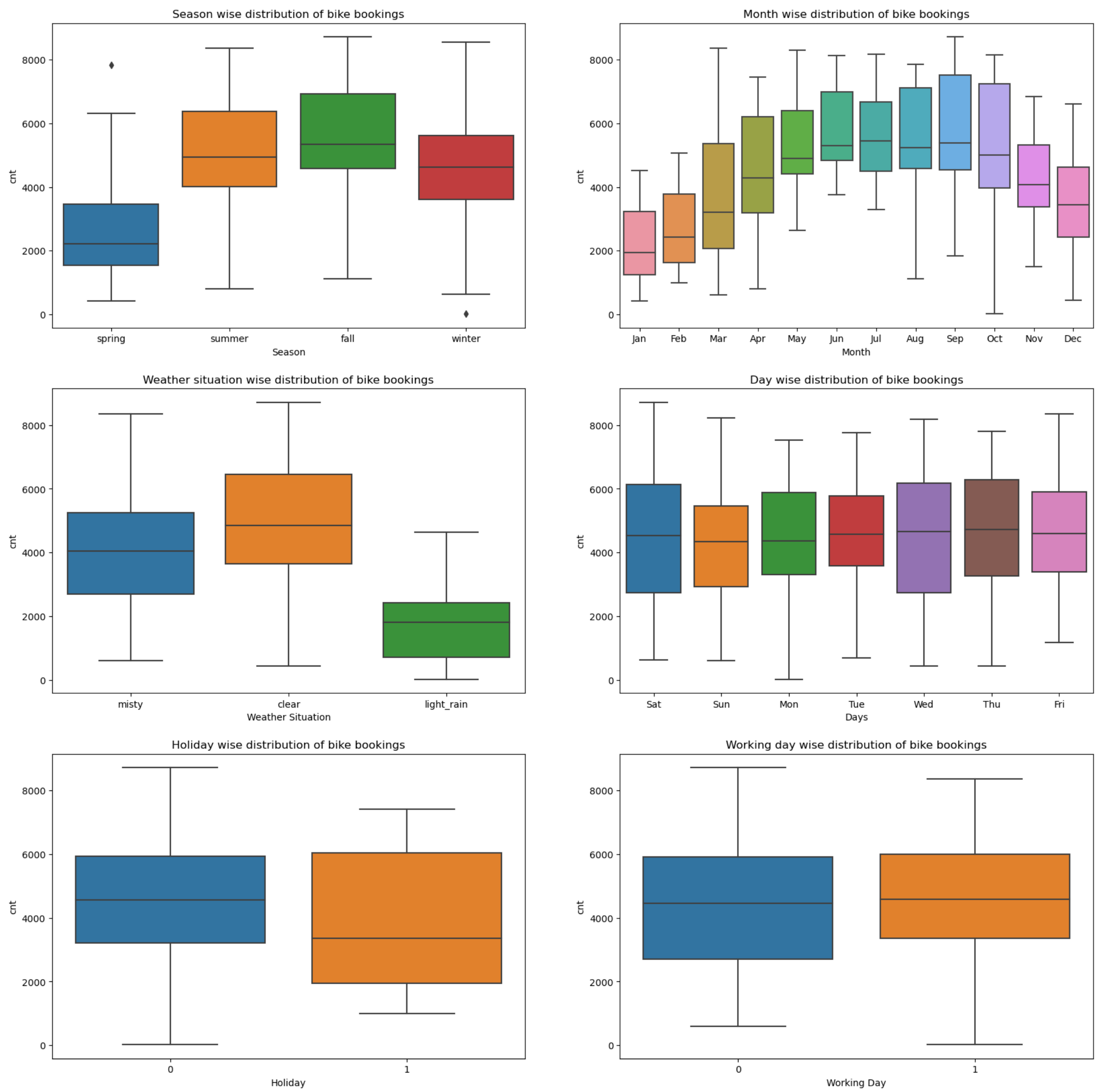
Visualising Categorical Variables

As there are a few categorical variables as well. Let's make a boxplot for some of these variables.

In [25]:

```
plt.figure(figsize=(20, 20))
plt.subplot(3,2,1)
```

```
sns.boxplot(x = 'season', y = 'cnt', data = df).set(title="Season wise distribution of
plt.subplot(3,2,2)
sns.boxplot(x = 'mnth', y = 'cnt', data = df).set(title="Month wise distribution of bi
plt.subplot(3,2,3)
sns.boxplot(x = 'weathersit', y = 'cnt', data = df).set(title="Weather situation wise
plt.subplot(3,2,4)
sns.boxplot(x = 'weekday', y = 'cnt', data = df).set(title="Day wise distribution of b
plt.subplot(3,2,5)
sns.boxplot(x = 'holiday', y = 'cnt', data = df).set(title="Holiday wise distribution
plt.subplot(3,2,6)
sns.boxplot(x = 'workingday', y = 'cnt', data = df).set(title="Working day wise distri

plt.show()
```



Inference:

Season

- fall season has the most number of bookings done, followed by summer & winter

- spring has the lowest no of bookings

Month
- most number of bookings were done in the month from august to october
- least number of bookings were done in the month from november to february
- this shows that month can be a good parameter for booking

Weather
- clear days have the most bookings
- light rain has the least
- there is no data of heavy rain

Weekday
- most bookings are done in wednesday, saturday and least on tuesday
- though there is not much clear pattern of this on the count variable

Holiday
- More bookings were done on holidays

Working Day
- Slightly more bookings were done on non-working day than working day

Step 3: Data Preparation

There are no yes/no values which means we don't need to do binary conversion to 0/1. Though we need to create dummy variables for the categorical variables.

Dummy Variables

1. season

In [26]:

```
# Get the dummy variables for the column 'season' and store it in a new variable - 'te

temp = pd.get_dummies(df.season)
temp.head()
```

Out[26]:

	fall	spring	summer	winter
0	0	1	0	0
1	0	1	0	0
2	0	1	0	0
3	0	1	0	0
4	0	1	0	0

Now, we don't need four columns. We can drop the fall column, as the type of season can be identified with the 3 columns where —

`000` will correspond to fall
`100` will correspond to spring
`010` will correspond to summer
`001` will correspond to winter

In [27]:
```
# Let's drop the first column from temp using 'drop_first = True'
temp = pd.get_dummies(df.season, drop_first = True)

# Add the results to the original dataframe
df = pd.concat([df, temp], axis = 1)

# Now let's see the head of our dataframe.
df.head()
```

Out[27]:

	season	yr	mnth	holiday	weekday	workingday	weathersit	temp	atemp	hum	winds
0	spring	0	Jan	0	Sat	0	misty	14.110847	18.18125	80.5833	10.74
1	spring	0	Jan	0	Sun	0	misty	14.902598	17.68695	69.6087	16.65
2	spring	0	Jan	0	Mon	1	clear	8.050924	9.47025	43.7273	16.63
3	spring	0	Jan	0	Tue	1	clear	8.200000	10.60610	59.0435	10.73
4	spring	0	Jan	0	Wed	1	clear	9.305237	11.46350	43.6957	12.52

In [28]:
```
# Drop 'season' as we have created the dummies for it
df.drop(['season'], axis = 1, inplace = True)

df.head()
```

Out[28]:

	yr	mnth	holiday	weekday	workingday	weathersit	temp	atemp	hum	windspeed	c
0	0	Jan	0	Sat	0	misty	14.110847	18.18125	80.5833	10.749882	9
1	0	Jan	0	Sun	0	misty	14.902598	17.68695	69.6087	16.652113	8
2	0	Jan	0	Mon	1	clear	8.050924	9.47025	43.7273	16.636703	13
3	0	Jan	0	Tue	1	clear	8.200000	10.60610	59.0435	10.739832	15
4	0	Jan	0	Wed	1	clear	9.305237	11.46350	43.6957	12.522300	16

2. mnth

In [29]:
```
# Get the dummy variables for the column 'mnth' and store it in a new variable - 'temp
temp = pd.get_dummies(df.mnth, drop_first = True)

# Add the results to the original dataframe
df = pd.concat([df, temp], axis = 1)
```

```python
# Drop 'mnth' as we have created the dummies for it
df.drop(['mnth'], axis = 1, inplace = True)

# Now let's see the head of our dataframe.
df.head()
```

Out[29]:

	yr	holiday	weekday	workingday	weathersit	temp	atemp	hum	windspeed	cnt	...
0	0	0	Sat	0	misty	14.110847	18.18125	80.5833	10.749882	985	...
1	0	0	Sun	0	misty	14.902598	17.68695	69.6087	16.652113	801	...
2	0	0	Mon	1	clear	8.050924	9.47025	43.7273	16.636703	1349	...
3	0	0	Tue	1	clear	8.200000	10.60610	59.0435	10.739832	1562	...
4	0	0	Wed	1	clear	9.305237	11.46350	43.6957	12.522300	1600	...

5 rows × 24 columns

3. weekday

In [30]:

```python
# Get the dummy variables for the column 'weekday' and store it in a new variable - 't
temp = pd.get_dummies(df.weekday, drop_first = True)

# Add the results to the original dataframe
df = pd.concat([df, temp], axis = 1)

# Drop 'weekday' as we have created the dummies for it
df.drop(['weekday'], axis = 1, inplace = True)

# Now let's see the head of our dataframe.
df.head()
```

Out[30]:

	yr	holiday	workingday	weathersit	temp	atemp	hum	windspeed	cnt	spring	...	M
0	0	0	0	misty	14.110847	18.18125	80.5833	10.749882	985	1	...	
1	0	0	0	misty	14.902598	17.68695	69.6087	16.652113	801	1	...	
2	0	0	1	clear	8.050924	9.47025	43.7273	16.636703	1349	1	...	
3	0	0	1	clear	8.200000	10.60610	59.0435	10.739832	1562	1	...	
4	0	0	1	clear	9.305237	11.46350	43.6957	12.522300	1600	1	...	

5 rows × 29 columns

4. weathersit

In [31]:

```python
# Get the dummy variables for the column 'weathersit' and store it in a new variable -
temp = pd.get_dummies(df.weathersit, drop_first = True)

# Add the results to the original dataframe
df = pd.concat([df, temp], axis = 1)
```

```
# Drop 'weathersit' as we have created the dummies for it
df.drop(['weathersit'], axis = 1, inplace = True)

# Now let's see the head of our dataframe.
df.head()
```

Out[31]:

	yr	holiday	workingday	temp	atemp	hum	windspeed	cnt	spring	summer	...	Oct
0	0	0	0	14.110847	18.18125	80.5833	10.749882	985	1	0	...	0
1	0	0	0	14.902598	17.68695	69.6087	16.652113	801	1	0	...	0
2	0	0	1	8.050924	9.47025	43.7273	16.636703	1349	1	0	...	0
3	0	0	1	8.200000	10.60610	59.0435	10.739832	1562	1	0	...	0
4	0	0	1	9.305237	11.46350	43.6957	12.522300	1600	1	0	...	0

5 rows × 30 columns

Step 4: Splitting the Data into Training and Testing Sets

As we know, the first basic step for regression is performing a train-test split.

In [32]:

```
df.head()
```

Out[32]:

	yr	holiday	workingday	temp	atemp	hum	windspeed	cnt	spring	summer	...	Oct
0	0	0	0	14.110847	18.18125	80.5833	10.749882	985	1	0	...	0
1	0	0	0	14.902598	17.68695	69.6087	16.652113	801	1	0	...	0
2	0	0	1	8.050924	9.47025	43.7273	16.636703	1349	1	0	...	0
3	0	0	1	8.200000	10.60610	59.0435	10.739832	1562	1	0	...	0
4	0	0	1	9.305237	11.46350	43.6957	12.522300	1600	1	0	...	0

5 rows × 30 columns

We have 30 columns now with all numeric values.

In [33]:

```
# We specify this so that the train and test data set always have the same rows, respe

np.random.seed(0)
df_train, df_test = train_test_split(df, train_size = 0.7, test_size = 0.3, random_sta
```

In [34]:

```
df_train.shape
```

Out[34]:

```
(510, 30)
```

In [35]:
```
df_test.shape
```
Out[35]:
```
(219, 30)
```

The problem here can occur because of no scaling present for variables like temp, atemp, hum, windspeed.
If we don't scale them, the model might be biased towards one kind of variable because of its high values.
So, lets first do scaling of these variables using `MinMaxScaler`.

Rescaling the Features

We need to rescale columns : temp, atemp, hum, windspeed, cnt

In [36]:
```
scaler = MinMaxScaler()

rescale_vars = ['temp', 'atemp', 'hum', 'windspeed','cnt']

df_train[rescale_vars] = scaler.fit_transform(df_train[rescale_vars])

df_train.head()
```
Out[36]:

	yr	holiday	workingday	temp	atemp	hum	windspeed	cnt	spring	summer	...
653	1	0	1	0.509887	0.501133	0.575354	0.300794	0.864243	0	0	...
576	1	0	1	0.815169	0.766351	0.725633	0.264686	0.827658	0	0	...
426	1	0	0	0.442393	0.438975	0.640189	0.255342	0.465255	1	0	...
728	1	0	0	0.245101	0.200348	0.498067	0.663106	0.204096	1	0	...
482	1	0	0	0.395666	0.391735	0.504508	0.188475	0.482973	0	1	...

5 rows × 30 columns

In [37]:
```
df_train.describe()
```

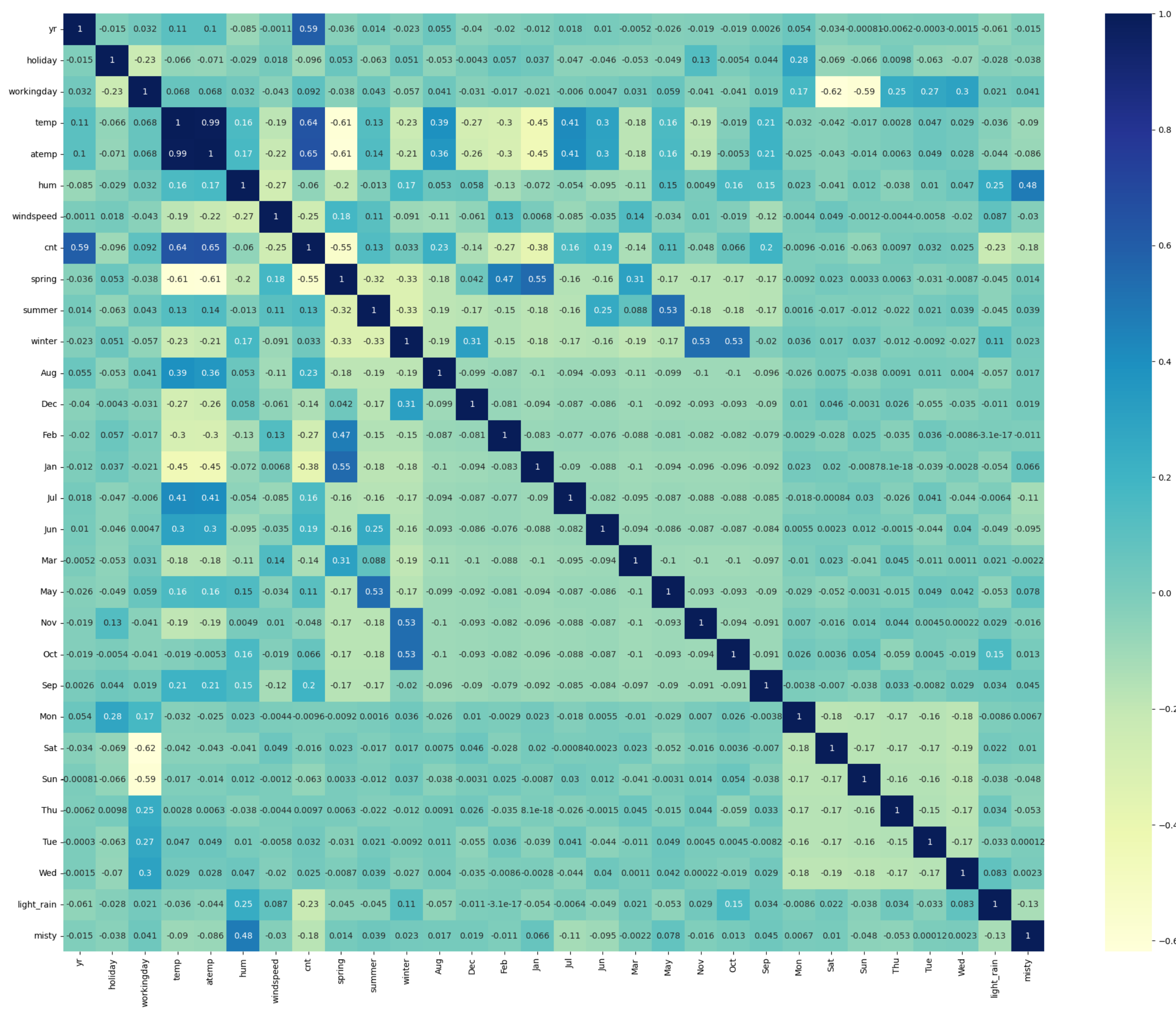
Out[37]:

	yr	holiday	workingday	temp	atemp	hum	windspeed	cı
count	510.000000	510.000000	510.000000	510.000000	510.000000	510.000000	510.000000	510.00000
mean	0.507843	0.025490	0.676471	0.537262	0.512989	0.650369	0.320768	0.51362
std	0.500429	0.157763	0.468282	0.225844	0.212385	0.145882	0.169797	0.22459
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.00000
25%	0.000000	0.000000	0.000000	0.339853	0.332086	0.538643	0.199179	0.35642
50%	1.000000	0.000000	1.000000	0.540519	0.526811	0.653714	0.296763	0.51863
75%	1.000000	0.000000	1.000000	0.735215	0.688457	0.754830	0.414447	0.68471
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.00000

8 rows × 30 columns

In [38]:

```
# Let's check the correlation coefficients to see which variables are highly correlate

plt.figure(figsize = (26, 20))
sns.heatmap(df_train.corr(), annot = True, cmap="YlGnBu")
plt.show()
```

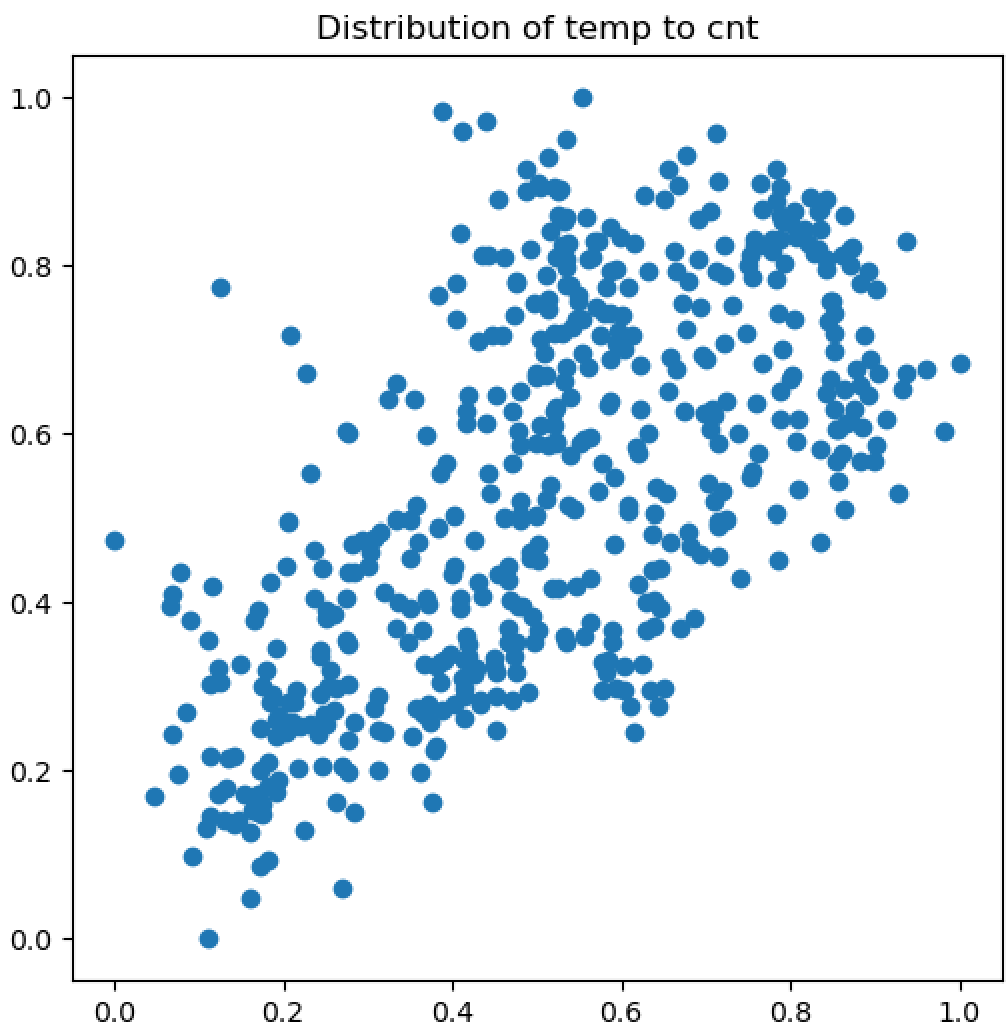


We can see here,

- `temp` and `atemp` are highly correlated (0.99)
- `cnt` has a high correlation with `temp` & `atemp` followed by `yr`
- `spring` is negatively correlated with `cnt`
- `workingday` has a negative correlation with sat,sun
- `hum` and `misty` are also positively correlated (0.48)
- We can see a good correlation between May and summer & oct,nov and winter which is fairly acceptable.

  Let's see a pairplot for cnt with temp, atemp and yr

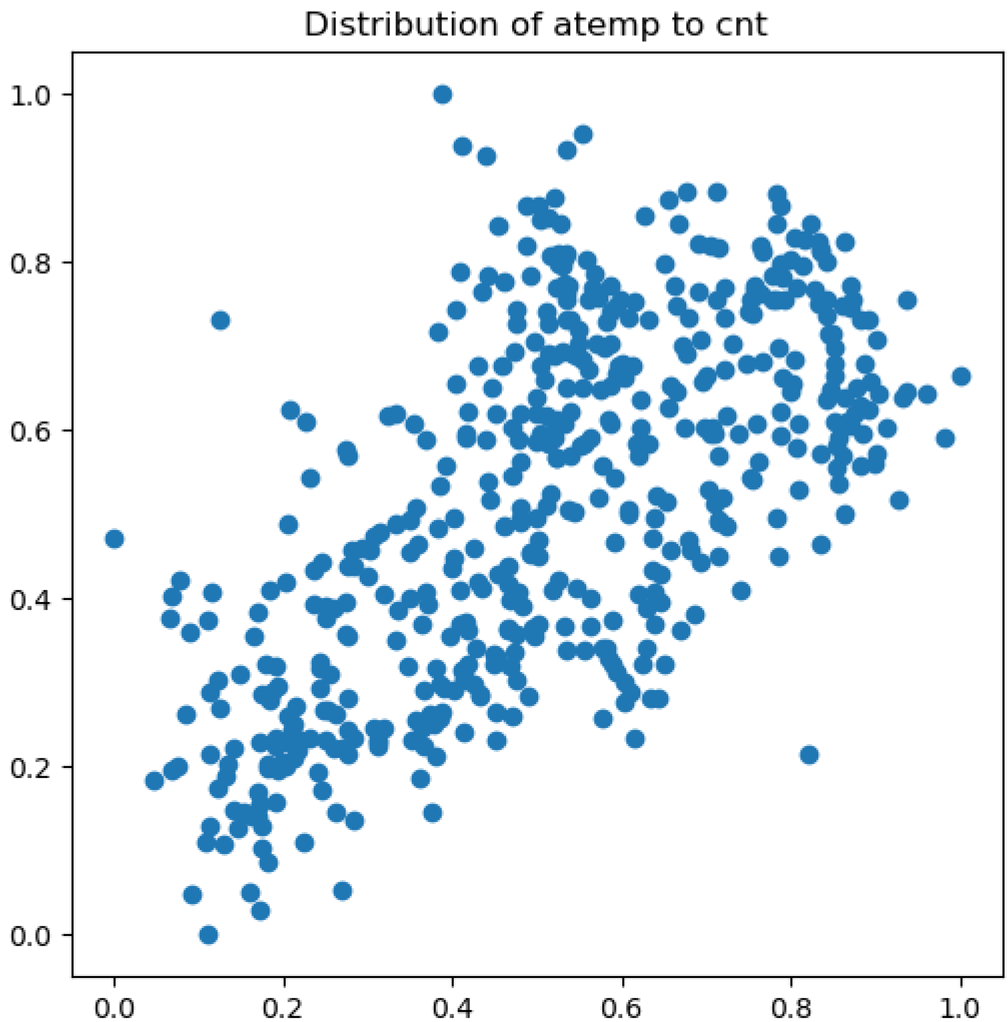
In [39]:
```
#temp

plt.figure(figsize=[6,6])
plt.scatter(df_train.cnt, df_train.temp)
plt.title("Distribution of temp to cnt", fontsize = 12)
plt.show()
```



This shows a linear relationship.

In [40]:

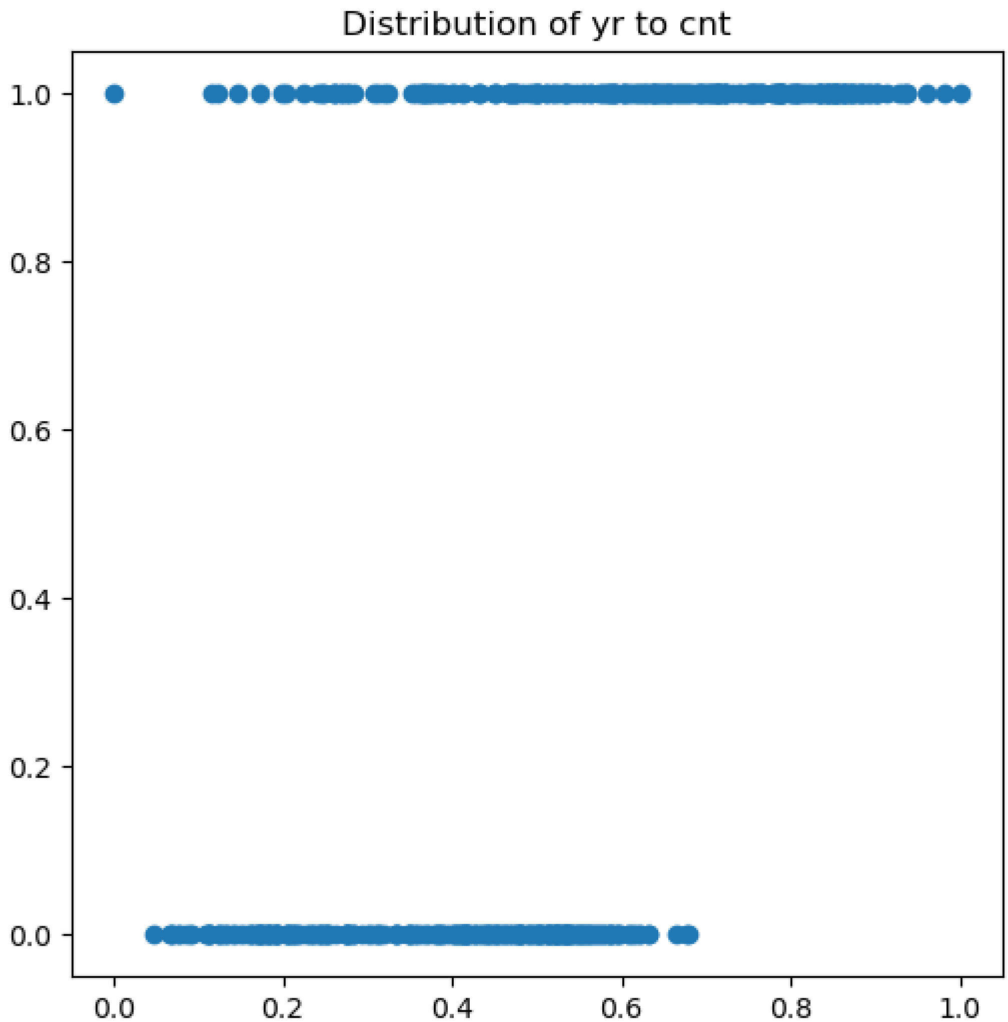
```
#atemp

plt.figure(figsize=[6,6])
plt.scatter(df_train.cnt, df_train.atemp)
plt.title("Distribution of atemp to cnt", fontsize = 12)
plt.show()
```



This shows a linear relationship.

In [41]:

```
#yr

plt.figure(figsize=[6,6])
plt.scatter(df_train.cnt, df_train.yr)
plt.title("Distribution of yr to cnt", fontsize = 12)
plt.show()
```



Inference

- Number of bikes booked in year 2019 is more compared to 2018.

So, we pick `temp` as the first variable and we'll try to fit a regression line to that.

Dividing into X and Y sets for the model building

In [42]:
```
y_train = df_train.pop('cnt')
X_train = df_train
```

Step 5: Building a linear model

RFE

Recursive feature elimination

In [43]:
```
# Running RFE with the output number of the variable equal to 20
lm = LinearRegression()
lm.fit(X_train, y_train)
```

```
rfe = RFE(lm, n_features_to_select=20)

# running RFE
rfe = rfe.fit(X_train, y_train)
```

In [44]:
```
# selected columns

list(zip(X_train.columns,rfe.support_,rfe.ranking_))
```

Out[44]:
```
[('yr', True, 1),
 ('holiday', True, 1),
 ('workingday', True, 1),
 ('temp', True, 1),
 ('atemp', True, 1),
 ('hum', True, 1),
 ('windspeed', True, 1),
 ('spring', True, 1),
 ('summer', True, 1),
 ('winter', True, 1),
 ('Aug', False, 8),
 ('Dec', True, 1),
 ('Feb', True, 1),
 ('Jan', True, 1),
 ('Jul', True, 1),
 ('Jun', True, 1),
 ('Mar', False, 4),
 ('May', False, 10),
 ('Nov', True, 1),
 ('Oct', False, 6),
 ('Sep', True, 1),
 ('Mon', False, 9),
 ('Sat', True, 1),
 ('Sun', False, 2),
 ('Thu', False, 7),
 ('Tue', False, 3),
 ('Wed', False, 5),
 ('light_rain', True, 1),
 ('misty', True, 1)]
```

In [45]:
```
col = X_train.columns[rfe.support_]
col
```

Out[45]:
```
Index(['yr', 'holiday', 'workingday', 'temp', 'atemp', 'hum', 'windspeed',
       'spring', 'summer', 'winter', 'Dec', 'Feb', 'Jan', 'Jul', 'Jun', 'Nov',
       'Sep', 'Sat', 'light_rain', 'misty'],
      dtype='object')
```

In [46]:
```
# eliminated columns

X_train.columns[~rfe.support_]
```

Out[46]:
```
Index(['Aug', 'Mar', 'May', 'Oct', 'Mon', 'Sun', 'Thu', 'Tue', 'Wed'], dtype='objec
t')
```

Building model using statsmodel, for the detailed statistics

In [47]:
```
# Creating X_test dataframe with RFE selected variables
X_train_rfe = X_train[col]
```

```
In [48]: # Adding a constant variable

         X_train_rfe = sm.add_constant(X_train_rfe)
```

```
In [49]: # Running the linear model
         lm = sm.OLS(y_train,X_train_rfe).fit()
```

```
In [50]: #Let's see the summary of our linear model
         print(lm.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                    cnt   R-squared:                       0.852
Model:                            OLS   Adj. R-squared:                  0.846
Method:                 Least Squares   F-statistic:                     140.8
Date:                Sun, 27 Aug 2023   Prob (F-statistic):         3.29e-188
Time:                        17:04:19   Log-Likelihood:                 525.70
No. Observations:                 510   AIC:                            -1009.
Df Residuals:                     489   BIC:                            -920.5
Df Model:                          20
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          0.2848      0.038      7.536      0.000       0.211       0.359
yr             0.2306      0.008     28.915      0.000       0.215       0.246
holiday       -0.0510      0.027     -1.892      0.059      -0.104       0.002
workingday     0.0439      0.011      3.833      0.000       0.021       0.066
temp           0.4630      0.135      3.439      0.001       0.198       0.728
atemp          0.0120      0.135      0.089      0.929      -0.254       0.278
hum           -0.1523      0.038     -4.018      0.000      -0.227      -0.078
windspeed     -0.1897      0.026     -7.318      0.000      -0.241      -0.139
spring        -0.0507      0.022     -2.309      0.021      -0.094      -0.008
summer         0.0402      0.016      2.548      0.011       0.009       0.071
winter         0.1038      0.018      5.760      0.000       0.068       0.139
Dec           -0.0470      0.018     -2.584      0.010      -0.083      -0.011
Feb           -0.0320      0.021     -1.490      0.137      -0.074       0.010
Jan           -0.0607      0.021     -2.837      0.005      -0.103      -0.019
Jul           -0.0567      0.019     -3.047      0.002      -0.093      -0.020
Jun           -0.0176      0.017     -1.020      0.308      -0.051       0.016
Nov           -0.0451      0.019     -2.409      0.016      -0.082      -0.008
Sep            0.0697      0.017      4.154      0.000       0.037       0.103
Sat            0.0540      0.014      3.746      0.000       0.026       0.082
light_rain    -0.2568      0.026     -9.817      0.000      -0.308      -0.205
misty         -0.0598      0.010     -5.784      0.000      -0.080      -0.040
==============================================================================
Omnibus:                       80.192   Durbin-Watson:                   2.034
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              213.551
Skew:                          -0.779   Prob(JB):                     4.25e-47
Kurtosis:                       5.761   Cond. No.                         87.1
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly spec
ified.
```

Lets check the VIF now.

In [51]:
```
X_train_rfe.columns
```

Out[51]:
```
Index(['const', 'yr', 'holiday', 'workingday', 'temp', 'atemp', 'hum',
       'windspeed', 'spring', 'summer', 'winter', 'Dec', 'Feb', 'Jan', 'Jul',
       'Jun', 'Nov', 'Sep', 'Sat', 'light_rain', 'misty'],
      dtype='object')
```

In [52]:
```
X_train_new = X_train_rfe.drop(['const'], axis=1)
```

In [53]:
```
# Calculate the VIFs for the new model
from statsmodels.stats.outliers_influence import variance_inflation_factor

vif = pd.DataFrame()
X = X_train_new
vif['Features'] = X.columns
vif['VIF'] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]
vif['VIF'] = round(vif['VIF'], 2)
vif = vif.sort_values(by = "VIF", ascending = False)
vif
```

Out[53]:

	Features	VIF
3	temp	386.48
4	atemp	369.31
5	hum	33.45
7	spring	5.73
2	workingday	5.40
6	windspeed	5.08
9	winter	4.28
8	summer	3.10
12	Jan	2.45
19	misty	2.33
0	yr	2.11
17	Sat	2.00
11	Feb	1.94
15	Nov	1.88
13	Jul	1.72
10	Dec	1.71
14	Jun	1.45
16	Sep	1.45
18	light_rain	1.29
1	holiday	1.21

Dropping the variable and updating the model

Both `temp` and `atemp` has very high VIF values. But `atemp` has higher p-value.
Lets drop `atemp` and rebuild the model

In [54]:
```
X_train_new = X_train_rfe.drop(["atemp"], axis = 1)

# Adding a constant variable
X_train_lm = sm.add_constant(X_train_new)

# Running the linear model
lm = sm.OLS(y_train,X_train_lm).fit()

#Let's see the summary of our linear model
print(lm.summary())
```

```
                            OLS Regression Results                            
==============================================================================
Dep. Variable:                    cnt   R-squared:                       0.852
Model:                            OLS   Adj. R-squared:                  0.846
Method:                 Least Squares   F-statistic:                     148.5
Date:                Sun, 27 Aug 2023   Prob (F-statistic):          2.67e-189
Time:                        17:04:20   Log-Likelihood:                 525.70
No. Observations:                 510   AIC:                            -1011.
Df Residuals:                     490   BIC:                            -926.7
Df Model:                          19                                         
Covariance Type:            nonrobust                                         
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          0.2847      0.038      7.544      0.000       0.211       0.359
yr             0.2306      0.008     28.946      0.000       0.215       0.246
holiday       -0.0511      0.027     -1.899      0.058      -0.104       0.002
workingday     0.0439      0.011      3.837      0.000       0.021       0.066
temp           0.4744      0.040     11.809      0.000       0.395       0.553
hum           -0.1521      0.038     -4.021      0.000      -0.226      -0.078
windspeed     -0.1902      0.025     -7.487      0.000      -0.240      -0.140
spring        -0.0504      0.022     -2.322      0.021      -0.093      -0.008
summer         0.0405      0.015      2.624      0.009       0.010       0.071
winter         0.1041      0.018      5.886      0.000       0.069       0.139
Dec           -0.0469      0.018     -2.585      0.010      -0.083      -0.011
Feb           -0.0320      0.021     -1.492      0.136      -0.074       0.010
Jan           -0.0608      0.021     -2.843      0.005      -0.103      -0.019
Jul           -0.0566      0.019     -3.050      0.002      -0.093      -0.020
Jun           -0.0177      0.017     -1.027      0.305      -0.051       0.016
Nov           -0.0450      0.019     -2.411      0.016      -0.082      -0.008
Sep            0.0698      0.017      4.171      0.000       0.037       0.103
Sat            0.0540      0.014      3.749      0.000       0.026       0.082
light_rain    -0.2569      0.026     -9.855      0.000      -0.308      -0.206
misty         -0.0599      0.010     -5.794      0.000      -0.080      -0.040
==============================================================================
Omnibus:                       80.049   Durbin-Watson:                   2.034
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              213.122
Skew:                          -0.777   Prob(JB):                     5.26e-47
Kurtosis:                       5.759   Cond. No.                         24.4
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly spec
ified.
```

In [55]:
```
X_train_new = X_train_new.drop(['const'], axis=1)
```

In [56]:
```
vif = pd.DataFrame()
X = X_train_new
vif['Features'] = X.columns
vif['VIF'] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]
vif['VIF'] = round(vif['VIF'], 2)
vif = vif.sort_values(by = "VIF", ascending = False)
vif
```

Out[56]:

	Features	VIF
4	hum	33.43
3	temp	23.29
6	spring	5.62
2	workingday	5.40
5	windspeed	4.83
8	winter	4.12
7	summer	2.96
11	Jan	2.44
18	misty	2.33
0	yr	2.11
16	Sat	2.00
10	Feb	1.94
14	Nov	1.88
12	Jul	1.72
9	Dec	1.71
13	Jun	1.44
15	Sep	1.44
17	light_rain	1.29
1	holiday	1.21

Dropping the variable and updating the model

`hum` has a very high VIF. Lets rebuild the model after dropping `hum`

In [57]:
```
X_train_new = X_train_rfe.drop(["atemp","hum"], axis = 1)

# Adding a constant variable
X_train_lm = sm.add_constant(X_train_new)

# Running the linear model
lm = sm.OLS(y_train,X_train_lm).fit()
```

```
#Let's see the summary of our linear model
print(lm.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                    cnt   R-squared:                       0.847
Model:                            OLS   Adj. R-squared:                  0.842
Method:                 Least Squares   F-statistic:                     151.1
Date:                Sun, 27 Aug 2023   Prob (F-statistic):          5.82e-187
Time:                        17:04:20   Log-Likelihood:                 517.42
No. Observations:                 510   AIC:                            -996.8
Df Residuals:                     491   BIC:                            -916.4
Df Model:                          18
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          0.2167      0.034      6.327      0.000       0.149       0.284
yr             0.2346      0.008     29.251      0.000       0.219       0.250
holiday       -0.0480      0.027     -1.758      0.079      -0.102       0.006
workingday     0.0471      0.012      4.068      0.000       0.024       0.070
temp           0.4246      0.039     10.942      0.000       0.348       0.501
windspeed     -0.1620      0.025     -6.535      0.000      -0.211      -0.113
spring        -0.0596      0.022     -2.720      0.007      -0.103      -0.017
summer         0.0316      0.016      2.036      0.042       0.001       0.062
winter         0.0901      0.018      5.117      0.000       0.056       0.125
Dec           -0.0560      0.018     -3.065      0.002      -0.092      -0.020
Feb           -0.0359      0.022     -1.651      0.099      -0.079       0.007
Jan           -0.0704      0.022     -3.266      0.001      -0.113      -0.028
Jul           -0.0484      0.019     -2.587      0.010      -0.085      -0.012
Jun           -0.0060      0.017     -0.348      0.728      -0.040       0.028
Nov           -0.0473      0.019     -2.492      0.013      -0.085      -0.010
Sep            0.0647      0.017      3.822      0.000       0.031       0.098
Sat            0.0589      0.015      4.035      0.000       0.030       0.088
light_rain    -0.2996      0.024    -12.393      0.000      -0.347      -0.252
misty         -0.0837      0.009     -9.741      0.000      -0.101      -0.067
==============================================================================
Omnibus:                       84.646   Durbin-Watson:                   2.035
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              235.303
Skew:                          -0.806   Prob(JB):                     8.03e-52
Kurtosis:                       5.911   Cond. No.                         22.6
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly spec
ified.
```

In [58]:

```
X_train_new.columns
```

Out[58]:

```
Index(['const', 'yr', 'holiday', 'workingday', 'temp', 'windspeed', 'spring',
       'summer', 'winter', 'Dec', 'Feb', 'Jan', 'Jul', 'Jun', 'Nov', 'Sep',
       'Sat', 'light_rain', 'misty'],
      dtype='object')
```

In [59]:

```
X_train_new = X_train_new.drop(['const'], axis=1)
```

In [60]:

```
# Calculate the VIFs for the new model

vif = pd.DataFrame()
X = X_train_new
```

```
vif['Features'] = X.columns
vif['VIF'] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]
vif['VIF'] = round(vif['VIF'], 2)
vif = vif.sort_values(by = "VIF", ascending = False)
vif
```

Out[60]:

	Features	VIF
3	temp	8.40
2	workingday	5.38
4	windspeed	4.74
5	spring	4.58
7	winter	3.12
6	summer	2.34
10	Jan	2.25
0	yr	2.08
15	Sat	2.00
9	Feb	1.89
13	Nov	1.83
11	Jul	1.71
17	misty	1.60
8	Dec	1.58
14	Sep	1.40
12	Jun	1.37
1	holiday	1.21
16	light_rain	1.09

Dropping the variable and updating the model

We will drop `Jun` now as it has a high p-value.

In [61]:

```
X_train_new = X_train_rfe.drop(["atemp","hum","Jun"], axis = 1)

# Adding a constant variable
X_train_lm = sm.add_constant(X_train_new)

# Running the linear model
lm = sm.OLS(y_train,X_train_lm).fit()

#Let's see the summary of our linear model
print(lm.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                    cnt   R-squared:                       0.847
Model:                            OLS   Adj. R-squared:                  0.842
Method:                 Least Squares   F-statistic:                     160.3
Date:                Sun, 27 Aug 2023   Prob (F-statistic):          4.80e-188
Time:                        17:04:20   Log-Likelihood:                 517.36
No. Observations:                 510   AIC:                            -998.7
Df Residuals:                     492   BIC:                            -922.5
Df Model:                          17
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          0.2187      0.034      6.474      0.000       0.152       0.285
yr             0.2348      0.008     29.327      0.000       0.219       0.250
holiday       -0.0477      0.027     -1.747      0.081      -0.101       0.006
workingday     0.0472      0.012      4.082      0.000       0.025       0.070
temp           0.4202      0.037     11.445      0.000       0.348       0.492
windspeed     -0.1619      0.025     -6.537      0.000      -0.211      -0.113
spring        -0.0600      0.022     -2.741      0.006      -0.103      -0.017
summer         0.0307      0.015      2.008      0.045       0.001       0.061
winter         0.0901      0.018      5.121      0.000       0.056       0.125
Dec           -0.0567      0.018     -3.118      0.002      -0.092      -0.021
Feb           -0.0365      0.022     -1.688      0.092      -0.079       0.006
Jan           -0.0714      0.021     -3.342      0.001      -0.113      -0.029
Jul           -0.0468      0.018     -2.582      0.010      -0.082      -0.011
Nov           -0.0477      0.019     -2.527      0.012      -0.085      -0.011
Sep            0.0655      0.017      3.905      0.000       0.033       0.098
Sat            0.0588      0.015      4.038      0.000       0.030       0.087
light_rain    -0.2995      0.024    -12.400      0.000      -0.347      -0.252
misty         -0.0834      0.009     -9.763      0.000      -0.100      -0.067
==============================================================================
Omnibus:                       85.173   Durbin-Watson:                   2.037
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              237.529
Skew:                          -0.810   Prob(JB):                     2.64e-52
Kurtosis:                       5.925   Cond. No.                         21.9
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly spec
ified.
```

In [62]:

```python
X_train_new = X_train_new.drop(['const'], axis=1)

# Calculate the VIFs for the new model

vif = pd.DataFrame()
X = X_train_new
vif['Features'] = X.columns
vif['VIF'] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]
vif['VIF'] = round(vif['VIF'], 2)
vif = vif.sort_values(by = "VIF", ascending = False)
vif
```

Out[62]:

	Features	VIF
3	temp	7.30
2	workingday	5.35
4	windspeed	4.73
5	spring	4.55
7	winter	3.07
6	summer	2.33
10	Jan	2.24
0	yr	2.08
14	Sat	1.99
9	Feb	1.89
12	Nov	1.83
11	Jul	1.60
8	Dec	1.58
16	misty	1.58
13	Sep	1.36
1	holiday	1.20
15	light_rain	1.09

Dropping the variable and updating the model

We will drop `Feb` now as it has a high p-value.

In [63]:

```
X_train_new = X_train_rfe.drop(["atemp","hum","Jun","Feb"], axis = 1)

# Adding a constant variable
X_train_lm = sm.add_constant(X_train_new)

# Running the linear model
lm = sm.OLS(y_train,X_train_lm).fit()

#Let's see the summary of our linear model
print(lm.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                    cnt   R-squared:                       0.846
Model:                            OLS   Adj. R-squared:                  0.841
Method:                 Least Squares   F-statistic:                     169.5
Date:                Sun, 27 Aug 2023   Prob (F-statistic):          1.49e-188
Time:                        17:04:21   Log-Likelihood:                 515.89
No. Observations:                 510   AIC:                            -997.8
Df Residuals:                     493   BIC:                            -925.8
Df Model:                          16
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          0.2055      0.033      6.242      0.000       0.141       0.270
yr             0.2345      0.008     29.241      0.000       0.219       0.250
holiday       -0.0499      0.027     -1.827      0.068      -0.104       0.004
workingday     0.0475      0.012      4.096      0.000       0.025       0.070
temp           0.4360      0.036     12.254      0.000       0.366       0.506
windspeed     -0.1604      0.025     -6.467      0.000      -0.209      -0.112
spring        -0.0701      0.021     -3.327      0.001      -0.112      -0.029
summer         0.0340      0.015      2.237      0.026       0.004       0.064
winter         0.0917      0.018      5.212      0.000       0.057       0.126
Dec           -0.0474      0.017     -2.731      0.007      -0.082      -0.013
Jan           -0.0520      0.018     -2.882      0.004      -0.087      -0.017
Jul           -0.0478      0.018     -2.630      0.009      -0.083      -0.012
Nov           -0.0429      0.019     -2.291      0.022      -0.080      -0.006
Sep            0.0669      0.017      3.989      0.000       0.034       0.100
Sat            0.0598      0.015      4.097      0.000       0.031       0.088
light_rain    -0.2987      0.024    -12.347      0.000      -0.346      -0.251
misty         -0.0834      0.009     -9.751      0.000      -0.100      -0.067
==============================================================================
Omnibus:                       80.436   Durbin-Watson:                   2.034
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              221.289
Skew:                          -0.770   Prob(JB):                     8.87e-49
Kurtosis:                       5.836   Cond. No.                         21.2
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly spec
ified.
```

In [64]:

```
X_train_new = X_train_new.drop(['const'], axis=1)

# Calculate the VIFs for the new model

vif = pd.DataFrame()
X = X_train_new
vif['Features'] = X.columns
vif['VIF'] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]
vif['VIF'] = round(vif['VIF'], 2)
vif = vif.sort_values(by = "VIF", ascending = False)
vif
```

Out[64]:

	Features	VIF
3	temp	7.21
2	workingday	5.33
4	windspeed	4.72
5	spring	3.17
7	winter	3.03
6	summer	2.33
0	yr	2.08
13	Sat	1.99
11	Nov	1.81
9	Jan	1.68
10	Jul	1.59
15	misty	1.57
8	Dec	1.48
12	Sep	1.36
1	holiday	1.20
14	light_rain	1.09

Dropping the variable and updating the model

We will drop `Nov` now as it has a high p-value.

In [65]:

```
X_train_new = X_train_rfe.drop(["atemp","hum","Jun","Feb","Nov"], axis = 1)

# Adding a constant variable
X_train_lm = sm.add_constant(X_train_new)

# Running the linear model
lm = sm.OLS(y_train,X_train_lm).fit()

#Let's see the summary of our linear model
print(lm.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                    cnt   R-squared:                       0.845
Model:                            OLS   Adj. R-squared:                  0.840
Method:                 Least Squares   F-statistic:                     178.9
Date:                Sun, 27 Aug 2023   Prob (F-statistic):         1.47e-188
Time:                        17:04:21   Log-Likelihood:                 513.18
No. Observations:                 510   AIC:                            -994.4
Df Residuals:                     494   BIC:                            -926.6
Df Model:                          15
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          0.1876      0.032      5.841      0.000       0.125       0.251
yr             0.2343      0.008     29.102      0.000       0.219       0.250
holiday       -0.0573      0.027     -2.103      0.036      -0.111      -0.004
workingday     0.0470      0.012      4.036      0.000       0.024       0.070
temp           0.4571      0.035     13.244      0.000       0.389       0.525
windspeed     -0.1614      0.025     -6.481      0.000      -0.210      -0.112
spring        -0.0614      0.021     -2.947      0.003      -0.102      -0.020
summer         0.0400      0.015      2.655      0.008       0.010       0.070
winter         0.0817      0.017      4.771      0.000       0.048       0.115
Dec           -0.0318      0.016     -1.982      0.048      -0.063      -0.000
Jan           -0.0467      0.018     -2.600      0.010      -0.082      -0.011
Jul           -0.0477      0.018     -2.613      0.009      -0.084      -0.012
Sep            0.0728      0.017      4.373      0.000       0.040       0.106
Sat            0.0597      0.015      4.073      0.000       0.031       0.088
light_rain    -0.2945      0.024    -12.156      0.000      -0.342      -0.247
misty         -0.0827      0.009     -9.626      0.000      -0.100      -0.066
==============================================================================
Omnibus:                       73.300   Durbin-Watson:                   2.054
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              186.508
Skew:                          -0.728   Prob(JB):                     3.16e-41
Kurtosis:                       5.580   Cond. No.                         20.5
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly spec
ified.
```

In [66]:

```
X_train_new = X_train_new.drop(['const'], axis=1)

# Calculate the VIFs for the new model

vif = pd.DataFrame()
X = X_train_new
vif['Features'] = X.columns
vif['VIF'] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]
vif['VIF'] = round(vif['VIF'], 2)
vif = vif.sort_values(by = "VIF", ascending = False)
vif
```

Out[66]:

	Features	VIF
3	temp	7.12
2	workingday	5.29
4	windspeed	4.68
5	spring	3.17
6	summer	2.33
7	winter	2.21
0	yr	2.08
12	Sat	1.99
9	Jan	1.67
10	Jul	1.59
14	misty	1.57
11	Sep	1.35
8	Dec	1.30
1	holiday	1.17
13	light_rain	1.09

Dropping the variable and updating the model

We will drop `Dec` now as it has a high p-value.

In [67]:

```
X_train_new = X_train_rfe.drop(["atemp","hum","Jun","Feb","Nov","Dec"], axis = 1)

# Adding a constant variable
X_train_lm = sm.add_constant(X_train_new)

# Running the linear model
lm = sm.OLS(y_train,X_train_lm).fit()

#Let's see the summary of our linear model
print(lm.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                    cnt   R-squared:                       0.843
Model:                            OLS   Adj. R-squared:                  0.839
Method:                 Least Squares   F-statistic:                     190.3
Date:                Sun, 27 Aug 2023   Prob (F-statistic):          7.33e-189
Time:                        17:04:21   Log-Likelihood:                 511.16
No. Observations:                 510   AIC:                            -992.3
Df Residuals:                     495   BIC:                            -928.8
Df Model:                          14
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          0.1737      0.031      5.525      0.000       0.112       0.235
yr             0.2344      0.008     29.019      0.000       0.218       0.250
holiday       -0.0562      0.027     -2.058      0.040      -0.110      -0.003
workingday     0.0465      0.012      3.983      0.000       0.024       0.069
temp           0.4728      0.034     14.037      0.000       0.407       0.539
windspeed     -0.1563      0.025     -6.292      0.000      -0.205      -0.107
spring        -0.0597      0.021     -2.861      0.004      -0.101      -0.019
summer         0.0434      0.015      2.890      0.004       0.014       0.073
winter         0.0797      0.017      4.650      0.000       0.046       0.113
Jan           -0.0389      0.018     -2.215      0.027      -0.073      -0.004
Jul           -0.0482      0.018     -2.635      0.009      -0.084      -0.012
Sep            0.0753      0.017      4.522      0.000       0.043       0.108
Sat            0.0584      0.015      3.980      0.000       0.030       0.087
light_rain    -0.2917      0.024    -12.027      0.000      -0.339      -0.244
misty         -0.0826      0.009     -9.592      0.000      -0.100      -0.066
==============================================================================
Omnibus:                       67.959   Durbin-Watson:                   2.066
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              166.078
Skew:                          -0.690   Prob(JB):                     8.64e-37
Kurtosis:                       5.431   Cond. No.                         20.0
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly spec
ified.
```

In [68]:

```python
X_train_new = X_train_new.drop(['const'], axis=1)

# Calculate the VIFs for the new model

vif = pd.DataFrame()
X = X_train_new
vif['Features'] = X.columns
vif['VIF'] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]
vif['VIF'] = round(vif['VIF'], 2)
vif = vif.sort_values(by = "VIF", ascending = False)
vif
```

Out[68]:

	Features	VIF
3	temp	7.07
2	workingday	5.24
4	windspeed	4.67
5	spring	3.08
6	summer	2.33
0	yr	2.08
7	winter	1.99
11	Sat	1.97
8	Jan	1.62
9	Jul	1.59
13	misty	1.57
10	Sep	1.35
1	holiday	1.17
12	light_rain	1.09

Dropping the variable and updating the model

We will drop `Jan` now as it has a high p-value.

In [69]:

```
X_train_new = X_train_rfe.drop(["atemp","hum","Jun","Feb","Nov","Dec","Jan"], axis = 1

# Adding a constant variable
X_train_lm = sm.add_constant(X_train_new)

# Running the linear model
lm = sm.OLS(y_train,X_train_lm).fit()

#Let's see the summary of our linear model
print(lm.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                    cnt   R-squared:                       0.842
Model:                            OLS   Adj. R-squared:                  0.838
Method:                 Least Squares   F-statistic:                     203.0
Date:                Sun, 27 Aug 2023   Prob (F-statistic):          5.73e-189
Time:                        17:04:21   Log-Likelihood:                 508.65
No. Observations:                 510   AIC:                            -989.3
Df Residuals:                     496   BIC:                            -930.0
Df Model:                          13
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          0.1577      0.031      5.134      0.000       0.097       0.218
yr             0.2336      0.008     28.839      0.000       0.218       0.250
holiday       -0.0571      0.027     -2.085      0.038      -0.111      -0.003
workingday     0.0463      0.012      3.947      0.000       0.023       0.069
temp           0.4920      0.033     15.056      0.000       0.428       0.556
windspeed     -0.1491      0.025     -6.032      0.000      -0.198      -0.101
spring        -0.0653      0.021     -3.139      0.002      -0.106      -0.024
summer         0.0465      0.015      3.101      0.002       0.017       0.076
winter         0.0859      0.017      5.058      0.000       0.053       0.119
Jul           -0.0500      0.018     -2.723      0.007      -0.086      -0.014
Sep            0.0758      0.017      4.532      0.000       0.043       0.109
Sat            0.0580      0.015      3.936      0.000       0.029       0.087
light_rain    -0.2904      0.024    -11.931      0.000      -0.338      -0.243
misty         -0.0835      0.009     -9.669      0.000      -0.100      -0.067
==============================================================================
Omnibus:                       66.977   Durbin-Watson:                   2.059
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              163.728
Skew:                          -0.681   Prob(JB):                     2.80e-36
Kurtosis:                       5.419   Cond. No.                         19.5
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly spec
ified.
```

In [70]:

```python
X_train_new = X_train_new.drop(['const'], axis=1)

# Calculate the VIFs for the new model

vif = pd.DataFrame()
X = X_train_new
vif['Features'] = X.columns
vif['VIF'] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]
vif['VIF'] = round(vif['VIF'], 2)
vif = vif.sort_values(by = "VIF", ascending = False)
vif
```

Out[70]:

	Features	VIF
3	temp	6.97
2	workingday	5.20
4	windspeed	4.65
5	spring	2.49
6	summer	2.32
0	yr	2.07
7	winter	1.99
10	Sat	1.96
8	Jul	1.58
12	misty	1.56
9	Sep	1.35
1	holiday	1.17
11	light_rain	1.08

Dropping the variable and updating the model

We will drop `Jul` now as it has a high p-value.

In [71]:

```
X_train_new = X_train_rfe.drop(["atemp","hum","Jun","Feb","Nov","Dec","Jan","Jul"], ax

# Adding a constant variable
X_train_lm = sm.add_constant(X_train_new)

# Running the linear model
lm = sm.OLS(y_train,X_train_lm).fit()

#Let's see the summary of our linear model
print(lm.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                    cnt   R-squared:                       0.839
Model:                            OLS   Adj. R-squared:                  0.836
Method:                 Least Squares   F-statistic:                     216.5
Date:                Sun, 27 Aug 2023   Prob (F-statistic):          1.48e-188
Time:                        17:04:21   Log-Likelihood:                 504.87
No. Observations:                 510   AIC:                            -983.7
Df Residuals:                     497   BIC:                            -928.7
Df Model:                          12
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          0.1484      0.031      4.832      0.000       0.088       0.209
yr             0.2342      0.008     28.729      0.000       0.218       0.250
holiday       -0.0551      0.028     -1.998      0.046      -0.109      -0.001
workingday     0.0475      0.012      4.033      0.000       0.024       0.071
temp           0.4793      0.033     14.724      0.000       0.415       0.543
windspeed     -0.1492      0.025     -5.997      0.000      -0.198      -0.100
spring        -0.0540      0.021     -2.632      0.009      -0.094      -0.014
summer         0.0615      0.014      4.378      0.000       0.034       0.089
winter         0.0982      0.016      5.961      0.000       0.066       0.131
Sep            0.0893      0.016      5.559      0.000       0.058       0.121
Sat            0.0586      0.015      3.954      0.000       0.030       0.088
light_rain    -0.2914      0.024    -11.894      0.000      -0.340      -0.243
misty         -0.0822      0.009     -9.470      0.000      -0.099      -0.065
==============================================================================
Omnibus:                       71.158   Durbin-Watson:                   2.092
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              170.059
Skew:                          -0.729   Prob(JB):                     1.18e-37
Kurtosis:                       5.424   Cond. No.                         19.5
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly spec
ified.
```

In [72]:

```python
X_train_new = X_train_new.drop(['const'], axis=1)

# Calculate the VIFs for the new model

vif = pd.DataFrame()
X = X_train_new
vif['Features'] = X.columns
vif['VIF'] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]
vif['VIF'] = round(vif['VIF'], 2)
vif = vif.sort_values(by = "VIF", ascending = False)
vif
```

Out[72]:

	Features	VIF
3	temp	5.70
2	workingday	5.20
4	windspeed	4.65
5	spring	2.40
0	yr	2.07
6	summer	2.00
9	Sat	1.96
7	winter	1.83
11	misty	1.56
8	Sep	1.24
1	holiday	1.17
10	light_rain	1.08

Dropping the variable and updating the model

We will drop `spring` now as it has a high p-value.

In [73]:
```
X_train_new = X_train_rfe.drop(["atemp","hum","Jun","Feb","Nov","Dec","Jan","Jul","spr

# Adding a constant variable
X_train_lm = sm.add_constant(X_train_new)

# Running the linear model
lm = sm.OLS(y_train,X_train_lm).fit()

#Let's see the summary of our linear model
print(lm.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                    cnt   R-squared:                       0.837
Model:                            OLS   Adj. R-squared:                  0.834
Method:                 Least Squares   F-statistic:                     232.8
Date:                Sun, 27 Aug 2023   Prob (F-statistic):         2.92e-188
Time:                        17:04:22   Log-Likelihood:                 501.34
No. Observations:                 510   AIC:                            -978.7
Df Residuals:                     498   BIC:                            -927.9
Df Model:                          11
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          0.0849      0.019      4.443      0.000       0.047       0.122
yr             0.2329      0.008     28.455      0.000       0.217       0.249
holiday       -0.0571      0.028     -2.059      0.040      -0.112      -0.003
workingday     0.0479      0.012      4.037      0.000       0.025       0.071
temp           0.5477      0.020     27.822      0.000       0.509       0.586
windspeed     -0.1543      0.025     -6.181      0.000      -0.203      -0.105
summer         0.0868      0.010      8.443      0.000       0.067       0.107
winter         0.1321      0.010     12.830      0.000       0.112       0.152
Sep            0.0992      0.016      6.317      0.000       0.068       0.130
Sat            0.0591      0.015      3.963      0.000       0.030       0.088
light_rain    -0.2893      0.025    -11.745      0.000      -0.338      -0.241
misty         -0.0818      0.009     -9.375      0.000      -0.099      -0.065
==============================================================================
Omnibus:                       64.326   Durbin-Watson:                   2.102
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              138.181
Skew:                          -0.698   Prob(JB):                     9.87e-31
Kurtosis:                       5.134   Cond. No.                         12.1
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly spec
ified.
```

In [74]:

```python
X_train_new = X_train_new.drop(['const'], axis=1)

# Calculate the VIFs for the new model

vif = pd.DataFrame()
X = X_train_new
vif['Features'] = X.columns
vif['VIF'] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]
vif['VIF'] = round(vif['VIF'], 2)
vif = vif.sort_values(by = "VIF", ascending = False)
vif
```

Out[74]:

	Features	VIF
3	temp	4.84
2	workingday	4.35
4	windspeed	3.55
0	yr	2.02
8	Sat	1.76
5	summer	1.57
10	misty	1.53
6	winter	1.42
7	Sep	1.21
1	holiday	1.12
9	light_rain	1.08

Now we can see, the VIFs and p-values both are within an acceptable range. So we go ahead and make our predictions using this model only.

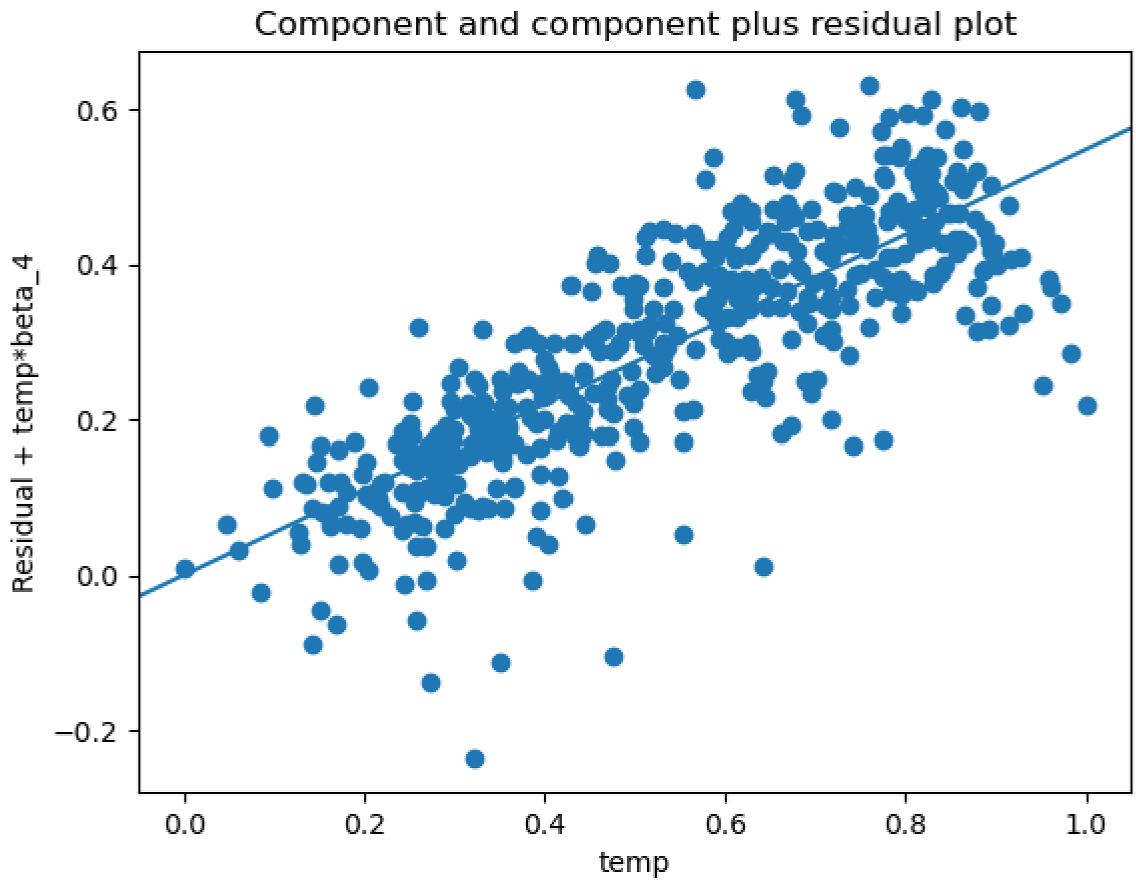
In [75]:
```
lm.params
```

Out[75]:
```
const          0.084901
yr             0.232913
holiday       -0.057089
workingday     0.047862
temp           0.547727
windspeed     -0.154257
summer         0.086798
winter         0.132127
Sep            0.099248
Sat            0.059118
light_rain    -0.289291
misty         -0.081844
dtype: float64
```

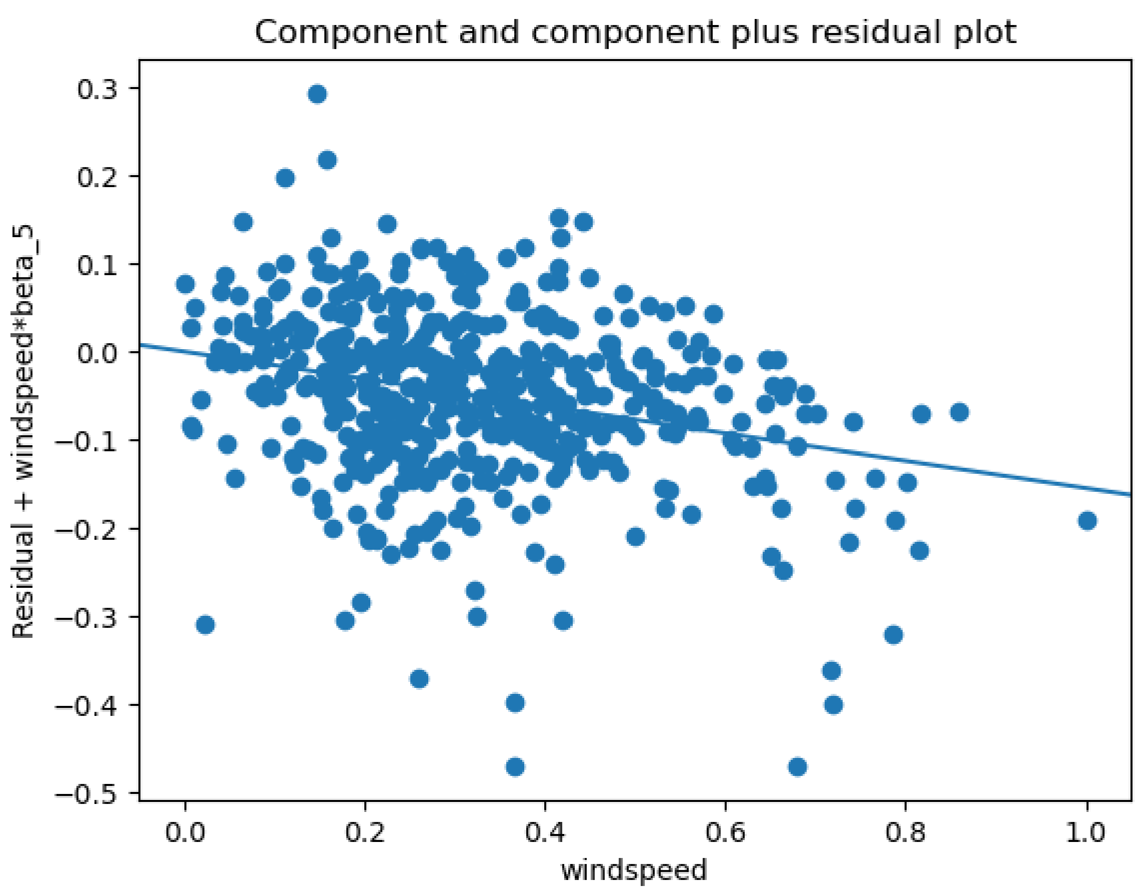
Check the various assumptions.

1. Linear relationship

In [76]:
```
sm.graphics.plot_ccpr(lm,"temp")
plt.show()
```



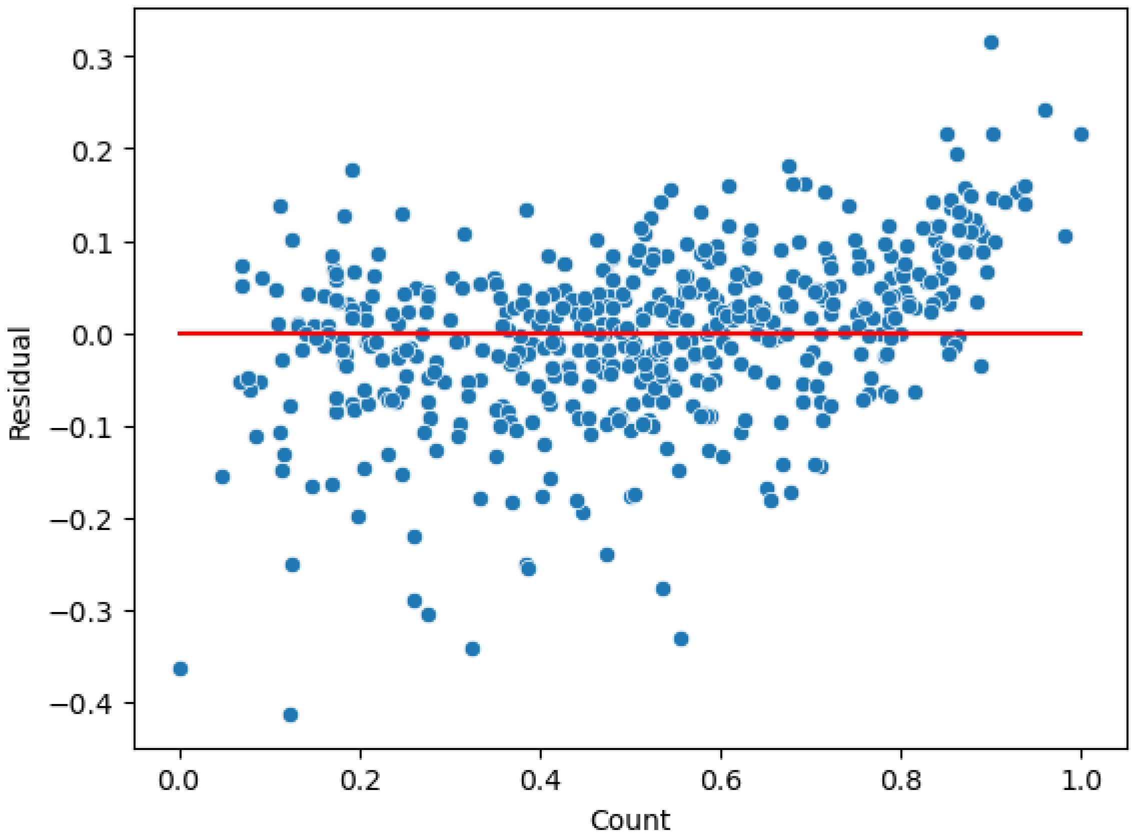
In [77]:
```
sm.graphics.plot_ccpr(lm,"windspeed")
plt.show()
```



We can see a linear relationship here.

2. Homoscedasticity

In [78]:
```
y_train_pred = lm.predict(X_train_lm)
residual = y_train - y_train_pred
sns.scatterplot(x=y_train,y=residual)
plt.plot(y_train, (y_train - y_train), '-r')
plt.xlabel("Count")
plt.ylabel("Residual")
plt.show()
```

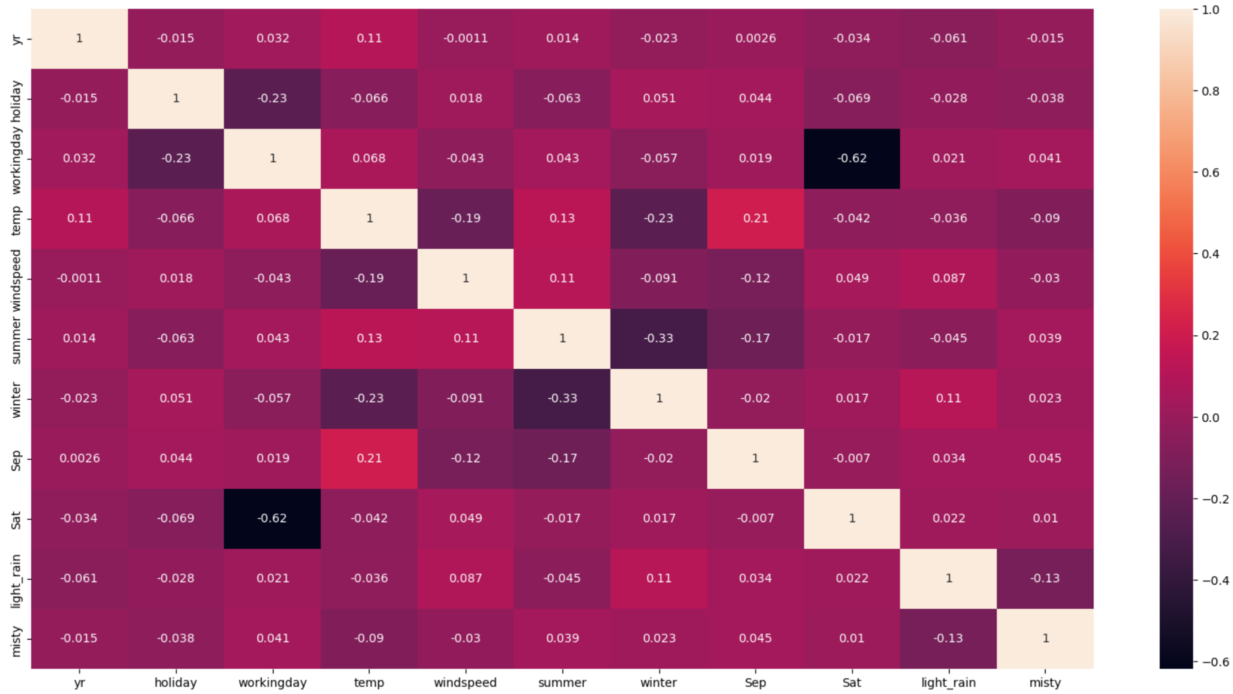


There seems to be no pattern or trend in the residual values.

3. No muticollinearity

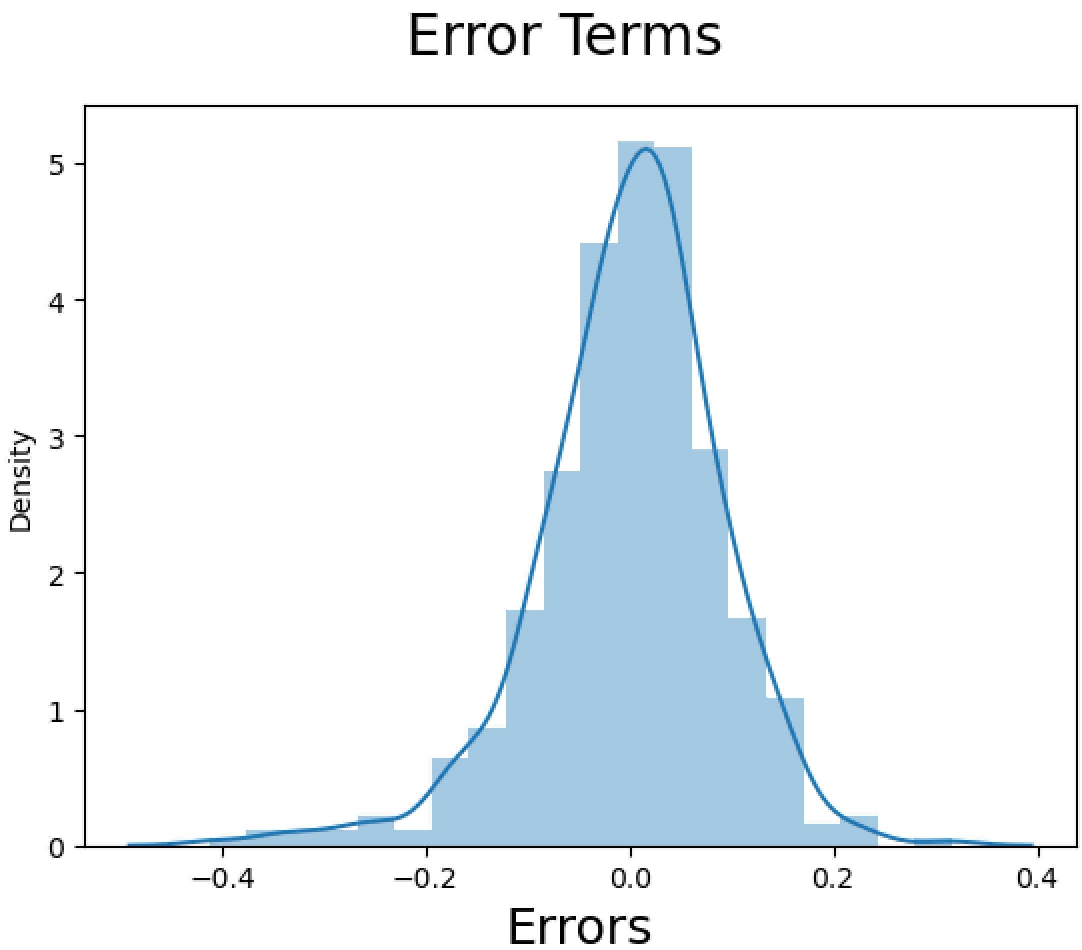
we saw the VIF values and all are less than 5.
Lets check from heatmap now.

In [79]:
```
plt.figure(figsize = (20,10))
sns.heatmap(X_train_new.corr(),annot = True)
plt.show()
```



4. Error terms are normally distributed

In [80]:
```
# Plot the histogram of the error terms
fig = plt.figure()
sns.distplot(residual,bins=20)
fig.suptitle('Error Terms', fontsize = 20)
plt.xlabel('Errors', fontsize = 18)
plt.show()
```



It is a normal distribution.

Step 6: Making Predictions Using the Final Model

Now that we have fitted the model and checked the normality of error terms, it's time to go ahead and make predictions using the final, i.e. lr_14 model.

Applying the scaling on the test sets

In [81]:
```
rescale_vars = ['temp','atemp', 'hum', 'windspeed','cnt']

df_test[rescale_vars] = scaler.transform(df_test[rescale_vars])
```

In [82]:
```
df_test.describe()
```

Out[82]:

	yr	holiday	workingday	temp	atemp	hum	windspeed	cı
count	219.000000	219.000000	219.000000	219.000000	219.000000	219.000000	219.000000	219.00000
mean	0.479452	0.036530	0.698630	0.558941	0.532991	0.638508	0.313350	0.52059
std	0.500722	0.188034	0.459904	0.233698	0.217888	0.148974	0.159947	0.21843
min	0.000000	0.000000	0.000000	0.046591	0.025950	0.261915	-0.042808	0.04820
25%	0.000000	0.000000	0.000000	0.354650	0.344751	0.527265	0.198517	0.37753
50%	0.000000	0.000000	1.000000	0.558691	0.549198	0.627737	0.299459	0.52427
75%	1.000000	0.000000	1.000000	0.759096	0.714132	0.743928	0.403048	0.67274
max	1.000000	1.000000	1.000000	0.984424	0.980934	1.002146	0.807474	0.96330

8 rows × 30 columns

Dividing into X_test and y_test

In [83]:
```
y_test = df_test.pop('cnt')
X_test = df_test
```

In [84]:
```
X_train_new.columns
```

Out[84]:
```
Index(['yr', 'holiday', 'workingday', 'temp', 'windspeed', 'summer', 'winter',
       'Sep', 'Sat', 'light_rain', 'misty'],
      dtype='object')
```

In [85]:
```
X_test=X_test[X_train_new.columns]
```

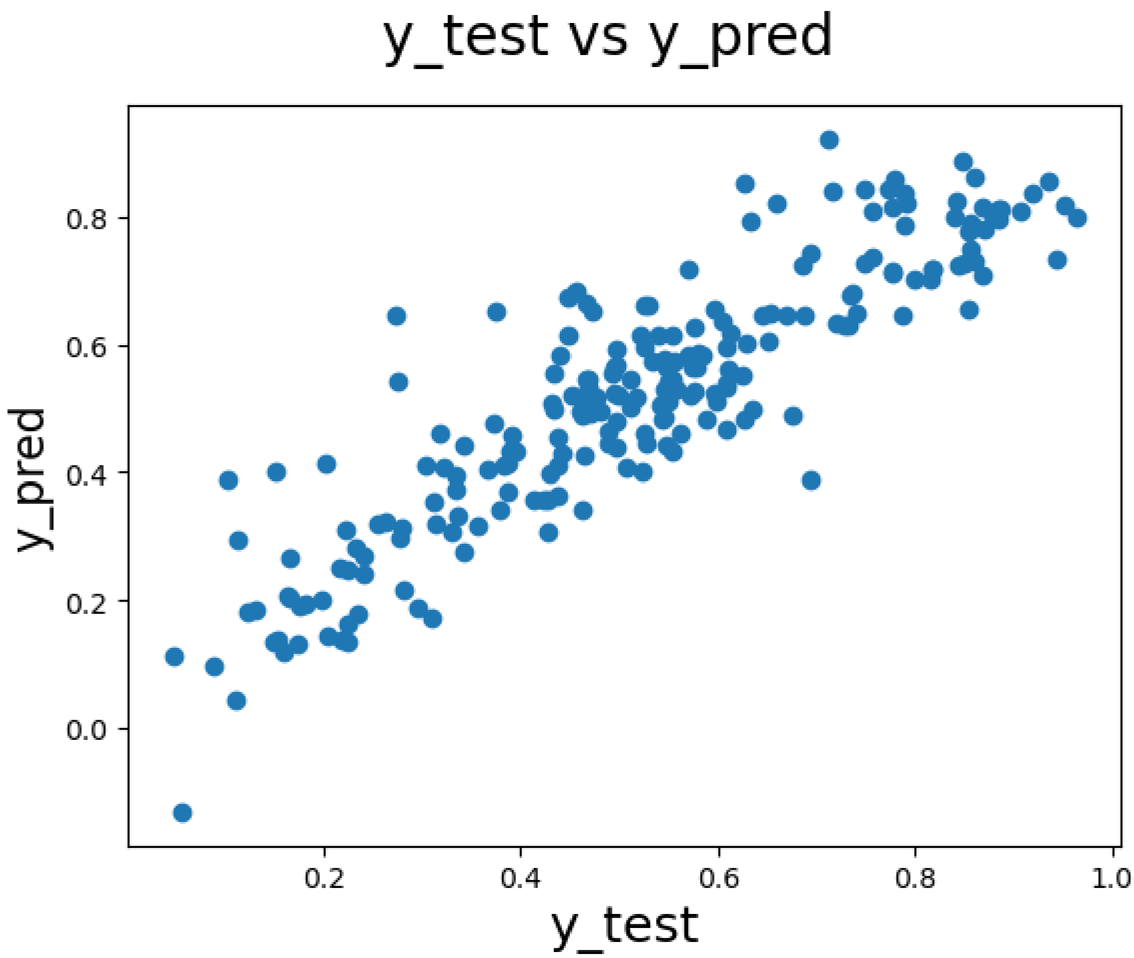
In [86]:
```
# Adding constant variable to test dataframe
X_test_lm = sm.add_constant(X_test)
```

In [87]:
```
# Making predictions using the lm model

y_pred= lm.predict(X_test_lm)
```

Step 7: Model Evaluation

Let's now plot the graph for actual versus predicted values.

In [88]:
```
# Plotting y_test and y_pred to understand the spread

fig = plt.figure()
plt.scatter(y_test, y_pred)
fig.suptitle('y_test vs y_pred', fontsize = 20)              # Plot heading
plt.xlabel('y_test', fontsize = 18)                          # X-label
plt.ylabel('y_pred', fontsize = 16)
plt.show()
```



R square calculation

1. test data set

In [89]:
```
r2_test = r2_score(y_test, y_pred)
print(r2_test)
```

```
0.796288126082227
```

1. train data set

In [90]:
```
r2_train = r2_score(y_train,y_train_pred)
print(r2_train)
```

```
0.8371634859985762
```

Adjusted R square calculation

1. test data set

In [91]:
```
n = X_test.shape[0]
p = X_test.shape[1]
```

```
adj_r2_test = 1-(1-r2_test)*(n-1)/(n-p-1)
print(adj_r2_test)
```

```
0.7854628574199298
```

1. train data set

In [92]:
```
n = X_train.shape[0]
p = X_train.shape[1]

adj_r2_train = 1-(1-r2_train)*(n-1)/(n-p-1)
print(adj_r2_train)
```

```
0.8273254466109901
```

We can see that there is not much deviations in the values of r square and adjusted r square when we compare for both test and training data set.
The difference is less tha 5% which suggested it is a good model

RMSE
Root Mean Squared Error

In [93]:
```
rmse =np.sqrt(mean_squared_error(y_test, y_pred))
print(rmse)
```

```
0.09836391656259555
```

In [94]:
```
mean_absolute_error(y_test, y_pred)
```

Out[94]:
```
0.07559984799794739
```

We can see that mean squared error and mean absolute error values are 0.0983 and 0.0755 respectively which indicates that the model we derived is good.

Final Model

In [95]:
```
lm.params
```

Out[95]:
```
const          0.084901
yr             0.232913
holiday       -0.057089
workingday     0.047862
temp           0.547727
windspeed     -0.154257
summer         0.086798
winter         0.132127
Sep            0.099248
Sat            0.059118
light_rain    -0.289291
misty         -0.081844
dtype: float64
```

Based on the coefficients we can say, below are the list of important variables in the order of there importance:

- **temp** : 0.547727 indicates that temperature has a significant impact on bookings
- **light_rain** : -0.289291 indicates that snow & rain are negatively impacting the bookings
- **yr** : 0.232913 indicates that the bookings of bike has increased over year
- **windspeed** : -0.154257 indicates that bike bookings decreases with increase in windspeed
- **winter** : 0.132127 indicates that bike booking is preferrable during winter season

Inference

- We can say that with increase in temperature and good weather condition positively affects the bike booking.
- We can dock more bikes based on the weather forcast(Clear > Misty > Rainy)
- Summer & Winters have more bike bookings, so we can promote, advertise more during these seasons
- Month september and day saturday also shows positive relationship with bike bookings
- There is a increase in bike bookings over the year. Once the lockdown opens, we can see the increase in bookings as well
- We can offer discounts may be to increase the bike bookings during spring season, cloudy and rainy weather etc.

We can see that the equation of our best fitted line is:

$ cnt = (0.547727 *temp) + (0.232913* yr) + (0.132127 *winter) + (0.099248* Sep) + (0.086798 *summer) + (0.059118* Sat) + (0.047862 *workingday) - (0.289291* light_rain) - (0.154257 *windspeed) - (0.057089* holiday) - (0.081844 * misty)

In [ ]:

```

```