BoomBikes Bike Sharing

In [87]:
```
#importing libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

import seaborn as sns
%matplotlib inline

import sklearn
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from sklearn.feature_selection import RFE
from sklearn.preprocessing import PolynomialFeatures
from sklearn.metrics import r2_score

import statsmodels.api as sm
```

In [88]:
```
#import dataset
biking = pd.read_csv('C:/2nd Semester - Course/Data Analytics/Project/day.csv')
biking.head()
```

Out[88]:

	instant	dteday	season	yr	mnth	holiday	weekday	workingday	weathersit	temp	atemp	hum	windspeed	casual	registered	c
0	1	01-01-2018	1	0	1	0	6	0	2	14.110847	18.18125	80.5833	10.749882	331	654	9
1	2	02-01-2018	1	0	1	0	0	0	2	14.902598	17.68695	69.6087	16.652113	131	670	8
2	3	03-01-2018	1	0	1	0	1	1	1	8.050924	9.47025	43.7273	16.636703	120	1229	13
3	4	04-01-2018	1	0	1	0	2	1	1	8.200000	10.60610	59.0435	10.739832	108	1454	15
4	5	05-01-2018	1	0	1	0	3	1	1	9.305237	11.46350	43.6957	12.522300	82	1518	16

In [89]:
```
#checking last rows
biking.tail()
```

Out[89]:

	instant	dteday	season	yr	mnth	holiday	weekday	workingday	weathersit	temp	atemp	hum	windspeed	casual	registered
725	726	27-12-2019	1	1	12	0	4	1	2	10.420847	11.33210	65.2917	23.458911	247	1867
726	727	28-12-2019	1	1	12	0	5	1	2	10.386653	12.75230	59.0000	10.416557	644	2451
727	728	29-12-2019	1	1	12	0	6	0	2	10.386653	12.12000	75.2917	8.333661	159	1182
728	729	30-12-2019	1	1	12	0	0	0	1	10.489153	11.58500	48.3333	23.500518	364	1432
729	730	31-12-2019	1	1	12	0	1	1	2	8.849153	11.17435	57.7500	10.374682	439	2290

In [90]:
```
#rows & columns
biking.shape
```

Out[90]: (730, 16)

In [91]:
```
#information
biking.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
```

In [92]:
```
#checking the numerical columns
biking.describe()
```

Out[92]:

	instant	season	yr	mnth	holiday	weekday	workingday	weathersit	temp	atemp	hum	win
count	730.000000	730.000000	730.000000	730.000000	730.000000	730.000000	730.000000	730.000000	730.000000	730.000000	730.000000	730
mean	365.500000	2.498630	0.500000	6.526027	0.028767	2.997260	0.683562	1.394521	20.319259	23.726322	62.765175	12
std	210.877136	1.110184	0.500343	3.450215	0.167266	2.006161	0.465405	0.544807	7.506729	8.150308	14.237589	5
min	1.000000	1.000000	0.000000	1.000000	0.000000	0.000000	0.000000	1.000000	2.424346	3.953480	0.000000	1
25%	183.250000	2.000000	0.000000	4.000000	0.000000	1.000000	0.000000	1.000000	13.811885	16.889713	52.000000	9
50%	365.500000	3.000000	0.500000	7.000000	0.000000	3.000000	1.000000	1.000000	20.465826	24.368225	62.625000	12
75%	547.750000	3.000000	1.000000	10.000000	0.000000	5.000000	1.000000	2.000000	26.880615	30.445775	72.989575	15
max	730.000000	4.000000	1.000000	12.000000	1.000000	6.000000	1.000000	3.000000	35.328347	42.044800	97.250000	34

Analysis

-We have all columns as float or int type except date type

-Further, there are some fields that are categorical in nature like season, mnth, holiday, weekdayetc.

In [93]:
```
#checking duplicates
biking.duplicated().sum()
```

Out[93]: 0

Analysis

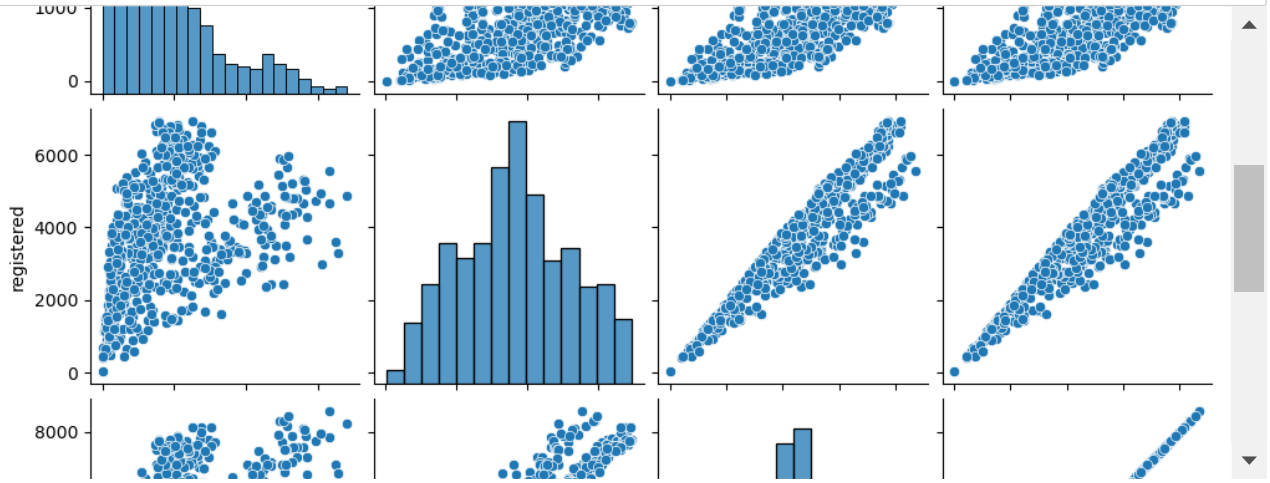
-There are no null values or missing data present.

-All columns have numerical(int & float) data except date.

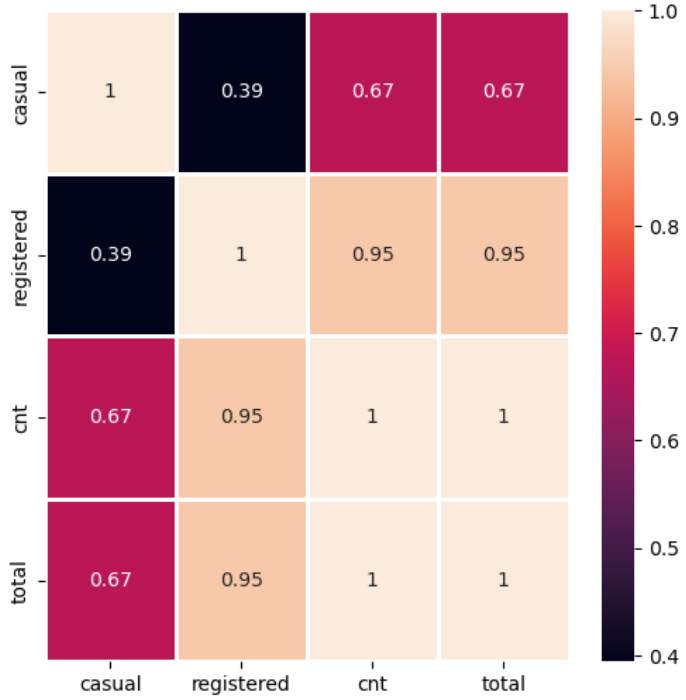
-There are no duplicates values.

-Further, there are some fields that are categorical in nature like season, mnth, holiay , weekday etc.

In [94]:
```
##analysing columns
biking_cnt = biking[['casual', 'registered', 'cnt']]
biking_cnt['total'] = biking_cnt['casual'] + biking_cnt['registered']
sns.pairplot(biking_cnt)
plt.show
```



In [95]:
```
#checking correlation
plt.figure(figsize = (6,6))
ax = sns.heatmap(biking_cnt.corr(), annot = True, linewidth = 1)
plt.show()
```



Analysis

As per the analysis we donot require some of the columns instant - record index - which doesnot have any significance.

temp and atemp are related to each other, so will be using one that is temp.

casual & registered - cnt includes both causal and registered.These columns contains the count of bike booked by different categories of customers. Further, we need to count total number of bikes and not to go with category.

dteday - As we have separate columns for 'year', 'month', so therefore no need of this column.

In [96]:
```
#dropping columns
biking.drop(['instant','atemp','casual','registered','dteday'], axis = 1, inplace = True)
```

In [97]:
```
#shape
biking.shape
```

Out[97]: (730, 11)

In [98]:
```
#Categorical Values
categorical_variables = ['season', 'yr', 'mnth', 'holiday', 'weekday', 'workingday', 'weathersit']
biking[categorical_variables] = biking[categorical_variables].astype('category')

#Numerical Values
numerical_varibales = ['temp', 'hum', 'windspeed', 'cnt']
```

In [99]:
```
#getting insights of categorical varibale
biking.describe(include = ['category'])
```

Out[99]:

	season	yr	mnth	holiday	weekday	workingday	weathersit
count	730	730	730	730	730	730	730
unique	4	2	12	2	7	2	3
top	3	0	1	0	0	1	1
freq	188	365	62	709	105	499	463

In [100]:
```
#getting insights of numeric variable
biking.describe()
```

Out[100]:

	temp	hum	windspeed	cnt
count	730.000000	730.000000	730.000000	730.000000
mean	20.319259	62.765175	12.763620	4508.006849
std	7.506729	14.237589	5.195841	1936.011647
min	2.424346	0.000000	1.500244	22.000000
25%	13.811885	52.000000	9.041650	3169.750000
50%	20.465826	62.625000	12.125325	4548.500000
75%	26.880615	72.989575	15.625589	5966.000000
max	35.328347	97.250000	34.000021	8714.000000

Step 2 - Visualising The Data

Below categorical columns are having following characterstics and can be maped with respective values:

season : column is having four seasons as (1:spring, 2:summer, 3:fall, 4:winter)

mnth : column is having 12 categorical vlaues donating for months Jan to Dec

weathersit : is having for categorical values(1:Clear_FewClouds, 2:Mist_Cloudy, 3:LightSnow_LightRain, 4:HeavyRain_IcePallets)

weekday : column having 7 varibales (0 to 6) denoting (0:Sun, 1:Mon, 2:Tue, 3:Wed, 4:Thu, 5:Fri, 6:Sat)

yr, holiday, workingday are having binary value.So we will not map these columns.

In [101]:
```
#Mapping the season column as given
biking['season'] = biking['season'].map({1 : 'spring', 2 : 'summer', 3: 'fall', 4 : 'winter'})

#Mapping month column as described
biking['mnth'] = biking['mnth'].map({1 : 'Jan', 2 : 'Feb', 3 : 'March', 4 : 'April', 5 : 'May', 6 : 'June', 7 : 'July
                                     9 : 'Sept', 10 : 'Oct', 11 : 'Nov', 12 : 'Dec'})

#Mapping weathersit column as describes
biking['weathersit'] = biking['weathersit'].map({1 : 'Clear_FewClouds', 2 : 'Mist_Cloudy', 3 : 'LightSnow_LightRain',

#Mapping weekday columns as describes
biking['weekday'] = biking['weekday'].map({0 : 'Sun', 1 : 'Mon', 2 : 'Tue', 3 : 'Wed', 4 : 'Thu', 5 : 'Fri', 6 : 'Sat
```

In [102]:
```
biking.head()
```

Out[102]:

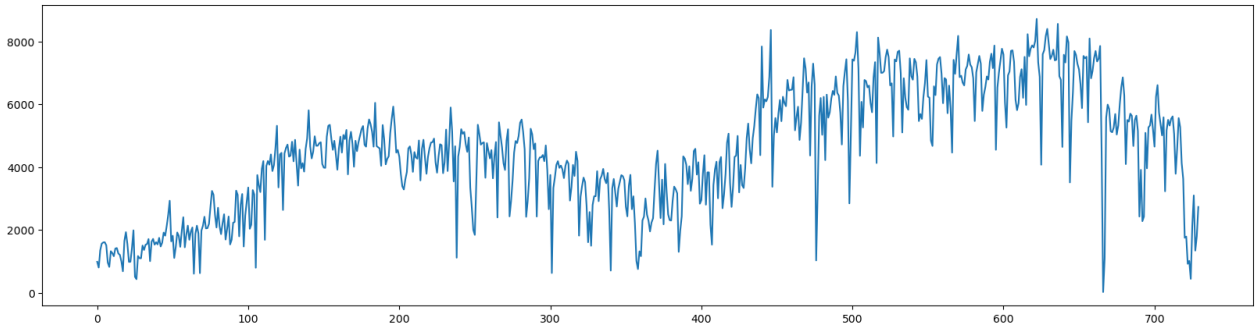
	season	yr	mnth	holiday	weekday	workingday	weathersit	temp	hum	windspeed	cnt
0	spring	0	Jan	0	Sat	0	Mist_Cloudy	14.110847	80.5833	10.749882	985
1	spring	0	Jan	0	Sun	0	Mist_Cloudy	14.902598	69.6087	16.652113	801
2	spring	0	Jan	0	Mon	1	Clear_FewClouds	8.050924	43.7273	16.636703	1349
3	spring	0	Jan	0	Tue	1	Clear_FewClouds	8.200000	59.0435	10.739832	1562
4	spring	0	Jan	0	Wed	1	Clear_FewClouds	9.305237	43.6957	12.522300	1600

In [103]:
```
biking.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 730 entries, 0 to 729
Data columns (total 11 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   season      730 non-null    category
 1   yr          730 non-null    category
 2   mnth        730 non-null    category
 3   holiday     730 non-null    category
 4   weekday     730 non-null    category
 5   workingday  730 non-null    category
 6   weathersit  730 non-null    category
 7   temp        730 non-null    float64
 8   hum         730 non-null    float64
 9   windspeed   730 non-null    float64
 10  cnt         730 non-null    int64
dtypes: category(7), float64(3), int64(1)
memory usage: 29.4 KB
```

2.1 Performing Univariate Analysis

In [104]:
```
#visualise the pattern of demand (target variable - 'cnt') over the period of 2 years
plt.figure(figsize=(20,5))
plt.plot(biking.cnt)
plt.show()
```

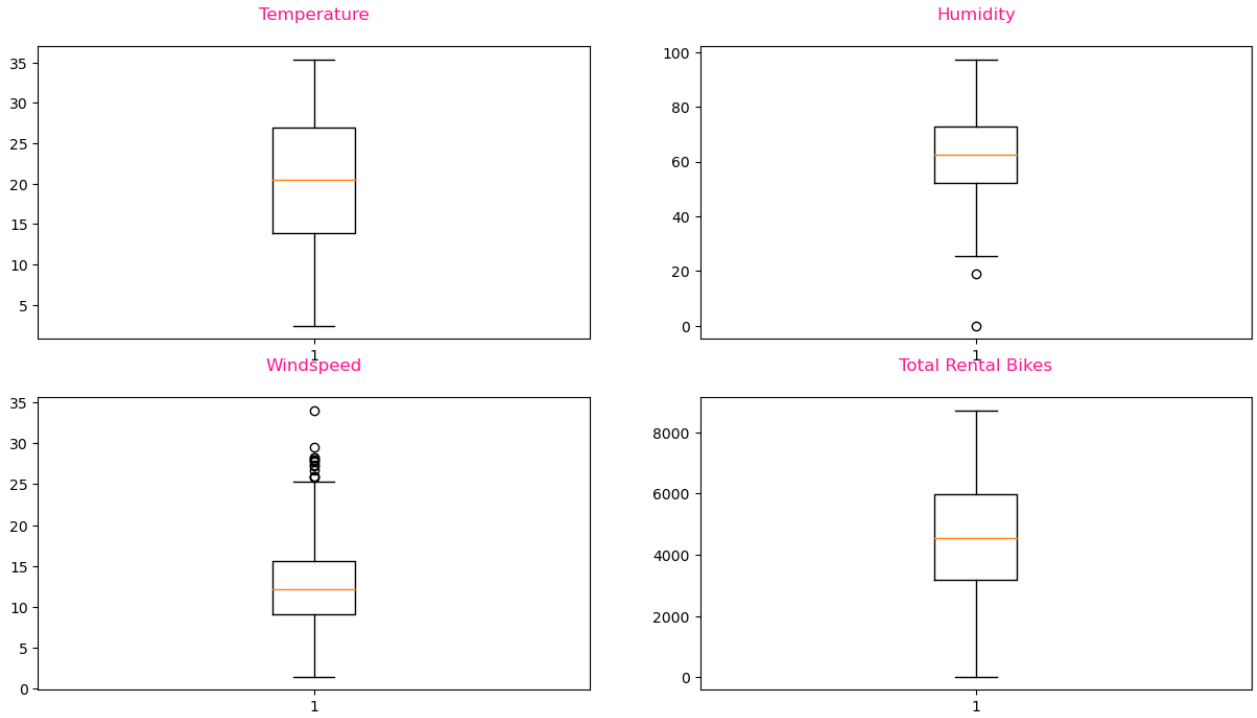


Analysis

-Over time, there has been a growth, and currently, there are comparably fewer users.

In [105]:
```
#Visualising numberical variables

#selecting numerical variables
#var = biking.select_dtypes(exclude = 'category').columns

plt.figure(figsize = (15,8))
plt.subplot(2,2,1)
plt.boxplot(biking['temp'])
plt.title("Temperature\n", fontdict = {'fontsize' : 12, 'fontweight' : 5, 'color' : '#FF1493'})

plt.subplot(2,2,2)
plt.boxplot(biking['hum'])
plt.title("Humidity\n", fontdict = {'fontsize' : 12, 'fontweight' : 5, 'color' : '#FF1493'})

plt.subplot(2,2,3)
plt.boxplot(biking['windspeed'])
plt.title("Windspeed\n", fontdict = {'fontsize' : 12, 'fontweight' : 5, 'color' : '#FF1493'})

plt.subplot(2,2,4)
plt.boxplot(biking['cnt'])
plt.title("Total Rental Bikes\n", fontdict = {'fontsize' : 12, 'fontweight' : 5, 'color' : '#FF1493'})
```

Out[105]:
```
Text(0.5, 1.0, 'Total Rental Bikes\n')
```



Analysis

- Humidity and Windspeed have few outliers.

In [106]:
```
#calculating percentage of outliers for hum and windspeed.

#function  to get outlier percentage

def cal_outlier_percentage(x):
    iqr = biking[x].quantile(0.75) - biking[x].quantile(0.25)
    UL = biking[x].quantile(0.75) + iqr * 1.5
    LL = biking[x].quantile(0.25) - iqr * 1.5
    cal = round((((biking[x] < LL).sum() + (biking[x] > UL).sum())/len(biking[x]) * 100),2)
    return(cal)

print('Percentage of outlier (hum) : ', cal_outlier_percentage('hum'))
print('Percentage of outlier (windspeed) : ' , cal_outlier_percentage('windspeed'))
```

```
Percentage of outlier (hum) :  0.27
Percentage of outlier (windspeed) :  1.78
```

In [107]:
```
#Finding the IQR for hum
Q1 = biking['hum'].quantile(0.25)
Q3 = biking['hum'].quantile(0.75)
IQR = Q3-Q1
upper_limit = Q3+1.5*IQR
lower_limit = Q1-1.5*IQR
print('lower limit: ', lower_limit)
print('upper limit: ', upper_limit)
print('IQR:' , IQR)
```

```
lower limit:  20.515637499999997
upper limit:  104.4739375
IQR: 20.989575000000002
```

In [108]:
```
biking[(biking['hum']>upper_limit) | (biking['hum']<lower_limit)]
```

Out[108]:

	season	yr	mnth	holiday	weekday	workingday	weathersit	temp	hum	windspeed	cnt
49	spring	0	Feb	0	Sat	0	Clear_FewClouds	16.365847	18.7917	34.000021	1635
68	spring	0	March	0	Thu	1	LightSnow_LightRain	15.952731	0.0000	17.545759	623

In [109]:
```
biking = biking[(biking['hum'] >= lower_limit) & (biking['hum'] <= upper_limit)]
```

In [110]:
```
#Finding the IQR for windspeed
Q1 = biking['windspeed'].quantile(0.25)
Q3 = biking['windspeed'].quantile(0.75)
IQR = Q3-Q1
upper_limit = Q3+1.5*IQR
lower_limit = Q1-1.5*IQR
print('lower limit: ', lower_limit)
print('upper limit: ', upper_limit)
print('IQR:' , IQR)
```

```
lower limit:  -0.8584374999999991
upper limit:  25.514638499999997
IQR: 6.593268999999999
```

In [111]:
```
biking[(biking['windspeed']>upper_limit) | (biking['windspeed']<lower_limit)]
```

Out[111]:

	season	yr	mnth	holiday	weekday	workingday	weathersit	temp	hum	windspeed	cnt
44	spring	0	Feb	0	Mon	1	Clear_FewClouds	17.015000	37.5833	27.999836	1913
93	summer	0	April	0	Mon	1	Clear_FewClouds	23.506653	42.6250	25.833257	3115
94	summer	0	April	0	Tue	1	Mist_Cloudy	16.980847	64.2083	26.000489	1795
292	winter	0	Oct	0	Thu	1	Clear_FewClouds	19.509153	63.6250	28.292425	4195
382	spring	1	Jan	0	Wed	1	Clear_FewClouds	12.436653	44.3333	27.833743	3376
407	spring	1	Feb	0	Sun	0	Clear_FewClouds	5.227500	46.4583	27.417204	1529
420	spring	1	Feb	0	Sat	0	Clear_FewClouds	11.924153	39.5833	28.250014	2732
431	spring	1	March	0	Thu	1	Clear_FewClouds	21.627500	56.7500	29.584721	5382
432	spring	1	March	0	Fri	1	Mist_Cloudy	16.844153	40.7083	27.791600	4569
449	summer	1	March	0	Mon	1	Clear_FewClouds	18.279153	47.7917	25.917007	5558
665	winter	1	Oct	0	Sun	0	Mist_Cloudy	19.577500	69.4583	26.666536	4459
720	spring	1	Dec	0	Sat	0	Clear_FewClouds	10.899153	44.1250	27.292182	1749

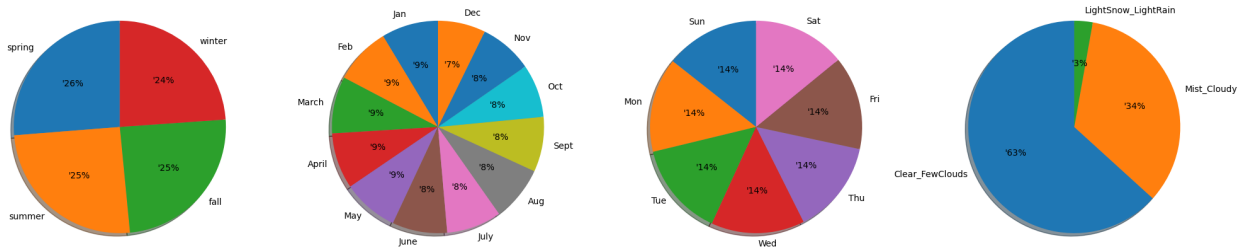
In [112]:
```
biking = biking[(biking['windspeed'] >= lower_limit) & (biking['windspeed'] <= upper_limit)]
```

Analysis

As there percentage is low and insignificant and therefore we have removed them using trimming concept.

In [113]:
```
plt.figure(figsize = (25,20))
plt.subplot(1,4,1)
plt.pie(biking['season'].value_counts(), startangle = 90,  autopct = "'%1.0f%%", labels = ['spring', 'summer', 'fall'

plt.subplot(1,4,2)
plt.pie(biking['mnth'].value_counts(), startangle = 90,  autopct = "'%1.0f%%", labels = ['Jan', 'Feb', 'March', 'Apri

plt.subplot(1,4,3)
plt.pie(biking['weekday'].value_counts(), startangle = 90,  autopct = "'%1.0f%%", labels = ['Sun', 'Mon', 'Tue', 'Wed

plt.subplot(1,4,4)
plt.pie(biking['weathersit'].value_counts(), startangle = 90,  autopct = "'%1.0f%%", labels = ['Clear_FewClouds', 'Mi

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

Out[113]:
```
([<matplotlib.patches.Wedge at 0x224cf93ebe0>,
  <matplotlib.patches.Wedge at 0x224cf93eca0>,
  <matplotlib.patches.Wedge at 0x224cfecc1f0>],
 [Text(-1.005814032276962, -0.4453516952642695, 'Clear_FewClouds'),
  Text(1.04097500761019, 0.35548703707866075, 'Mist_Cloudy'),
  Text(0.09640561500029646, 1.0957672916255599, 'LightSnow_LightRain')],
 [Text(-0.5486258357874337, -0.24291910650778334, "'63%"),
  Text(0.567804549605558, 0.19390202022472403, "'34%"),
  Text(0.052584880909252604, 0.597691249977578, "'3%")])
```



Analysis

1) Seasons: The business climate is essentially the same across the four seasons.

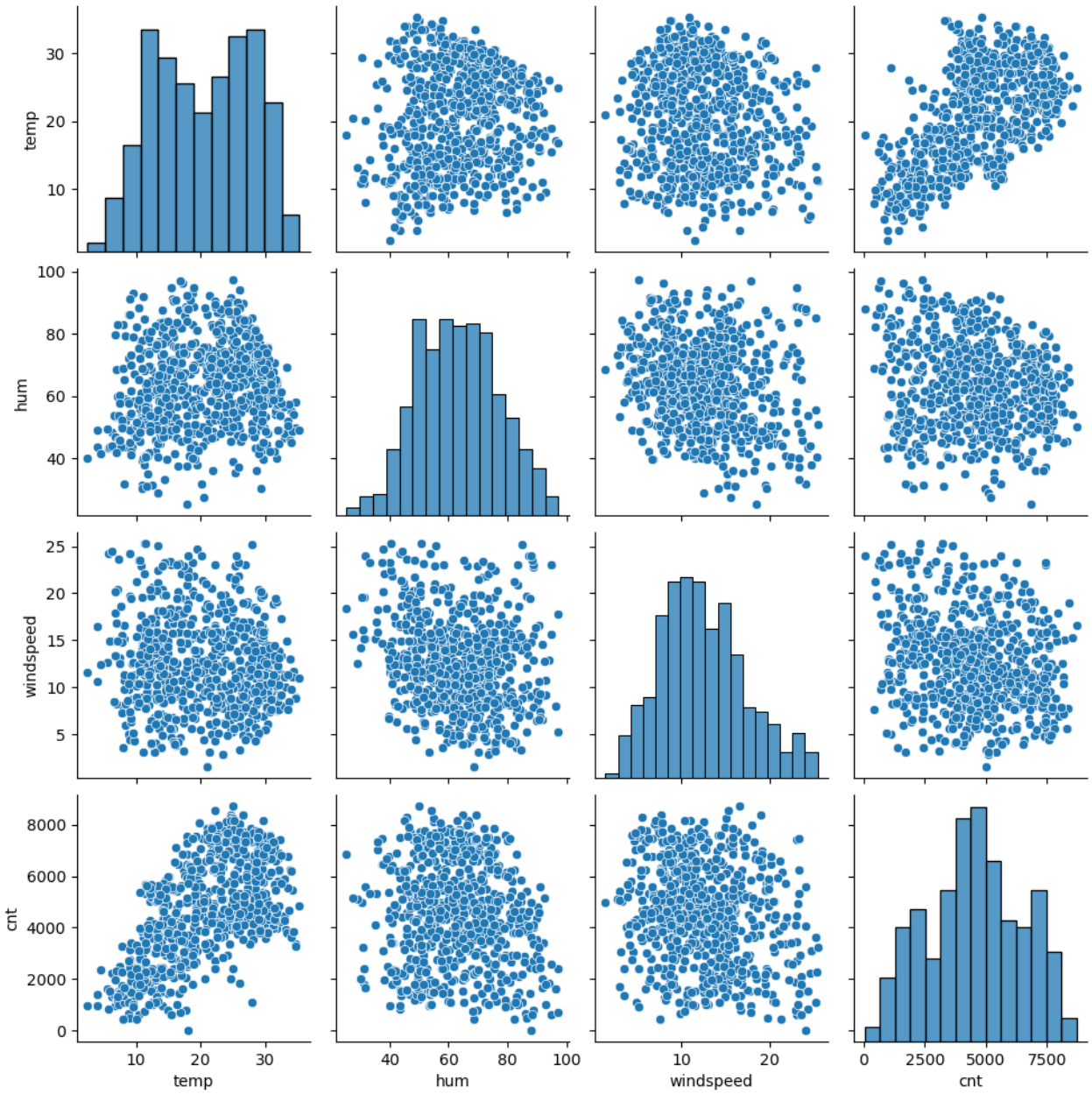
2) Monthly: Each monthly % is the same.

3) Weekdays: There is a similar percentage of business on all weekdays.

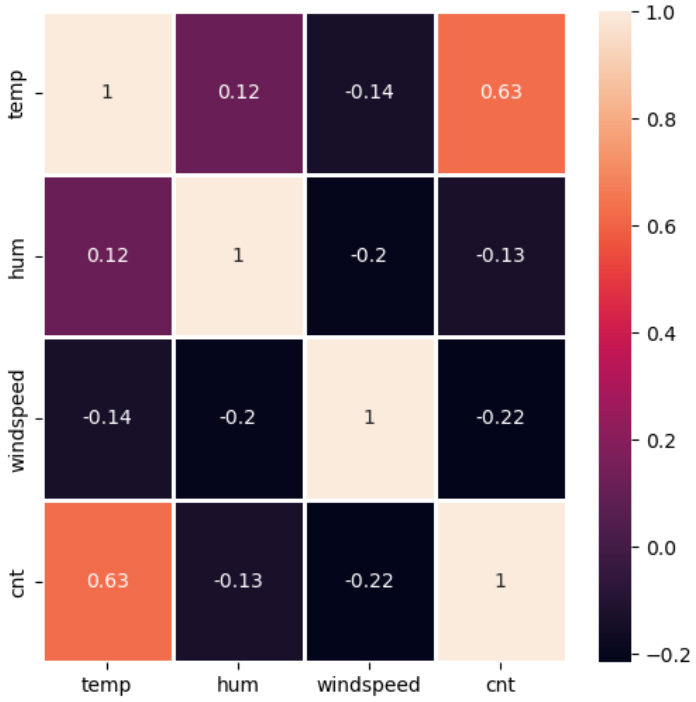
4) Weather: Despite the fact that there are four different weather conditions, most individuals prefer to bike when the sky is clear, partially cloudy, or cloudless. The opposite is true when there is heavy rain, ice pallets, a thunderstorm, mist, snow, and fog.

2.2 Bivariate Analysis

In [115]:
```
#visualising the numeric variable
sns.pairplot(biking)
plt.show()
#The probability density function (PDF) of a random variable can be estimated in statistics using kernel density
#estimation (KDE), a non-parametric method.
```



In [116]:
```
#checking correlation

plt.figure(figsize = (6,6))
ax = sns.heatmap(biking.corr(), annot = True, linewidth = 1)
```

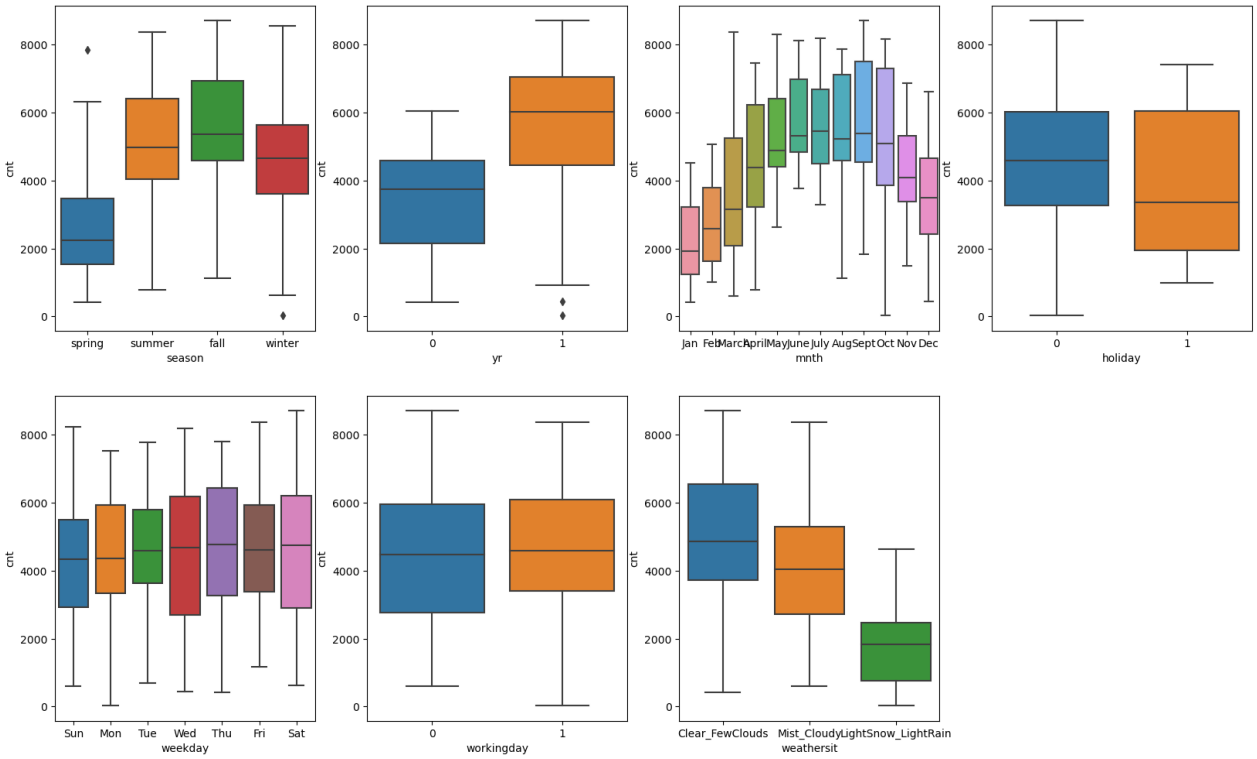


Analysis

temp has highest positive correlation with target variable cnt.

In [117]:
```
#Visualising the categorical variables
plt.figure(figsize = (20, 12))
plt.subplot(2,4,1)
sns.boxplot(x = 'season', y = 'cnt', data = biking)
plt.subplot(2,4,2)
sns.boxplot(x = 'yr', y = 'cnt', data = biking)
plt.subplot(2,4,3)
sns.boxplot(x = 'mnth', y = 'cnt', data = biking)
plt.subplot(2,4,4)
sns.boxplot(x = 'holiday', y = 'cnt', data = biking)
plt.subplot(2,4,5)
sns.boxplot(x = 'weekday', y = 'cnt', data = biking)
plt.subplot(2,4,6)
sns.boxplot(x = 'workingday', y = 'cnt', data = biking)
plt.subplot(2,4,7)
sns.boxplot(x = 'weathersit', y = 'cnt', data = biking)
```

Out[117]: <AxesSubplot:xlabel='weathersit', ylabel='cnt'>



Analysis

-There are 4 seasons (1 : spring, 2: summer, 3 : fall, 4 : winter) and if you see the boxplot summer and fall have many users that took bikes on rent.

-There are 2 years ((0:2018, 1 : 2019)) and according to chart, there are more users in 2019.

-Whenever there is no holiday, more people are taking bikes on rent.

-Weekdays have almost similar number of users.

-Weathersit - There are four classifications, however the majority of individuals prefer to bike when the sky is clear, has few clouds, or is partially cloudy. Contrarily, when there is a thunderstorm, heavy rain, ice pallets, mist, snow, and fog.

Analysis

1) According to season, in summer & fall most people rent the bicycles.

2) According to month, most bookings are happening between May to October.

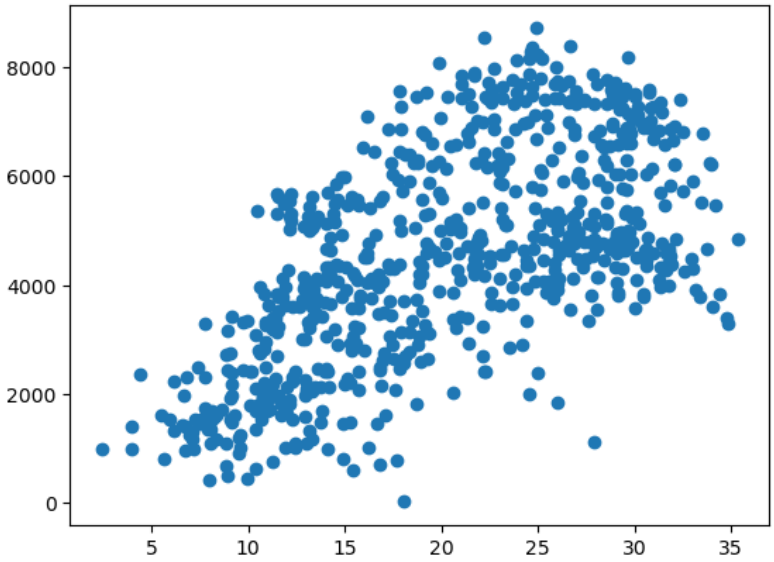
3) Accouding to weather, when the sky is clear, most bookings were done.

4) Bookings were happening on all days of the week.

5) During non-holiday hours, bicycle rentals were more.

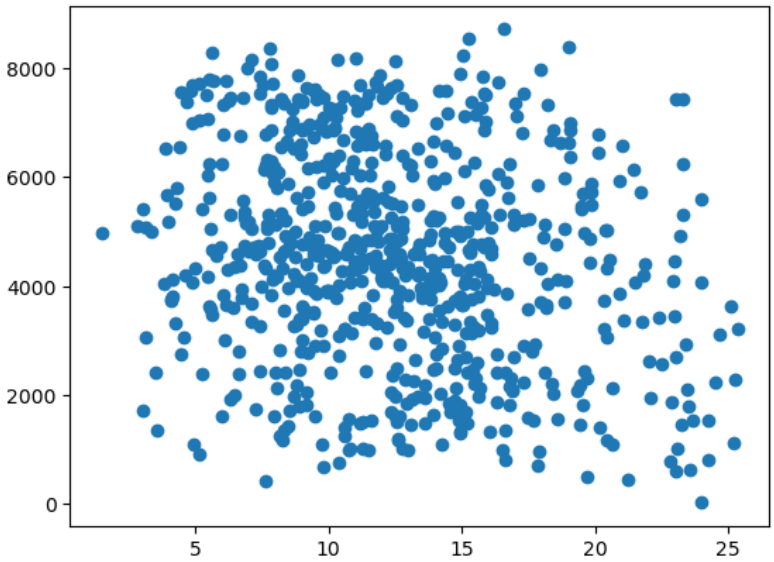
In [118]:
```
#temperature
plt.scatter('temp', 'cnt', data=biking)
```

Out[118]: <matplotlib.collections.PathCollection at 0x224d0c93f10>



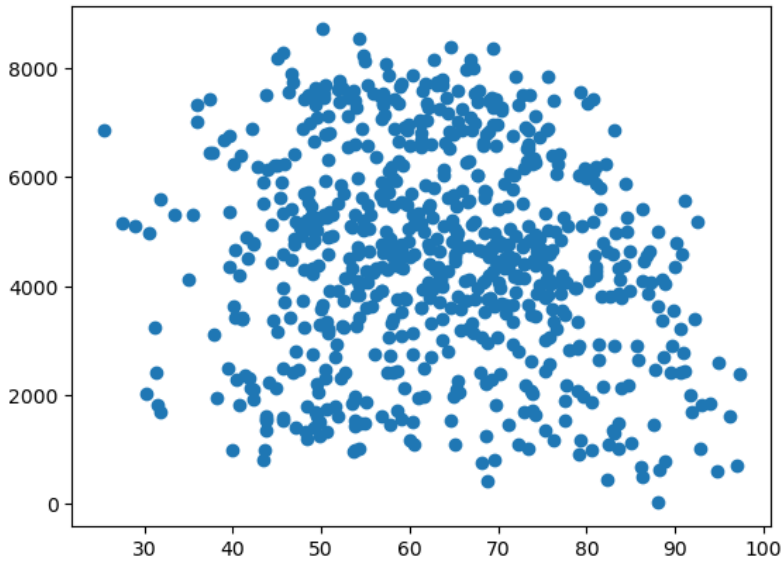
In [119]:
```
#wind
plt.scatter('windspeed', 'cnt', data=biking)

#wind speeds increase with a greater temperature.It is highly correlated with temp
```

Out[119]: <matplotlib.collections.PathCollection at 0x224d06e6f10>



In [120]:
```
#humidity

plt.scatter('hum', 'cnt', data=biking)
#Temperature being directly proportional to humidity
```

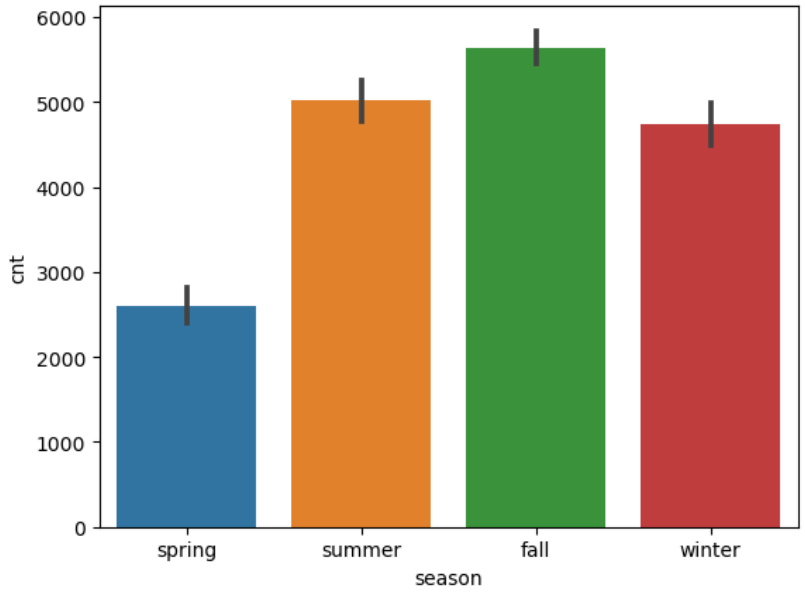
Out[120]: <matplotlib.collections.PathCollection at 0x224d0573520>



In [121]:
```
sns.barplot('season', 'cnt', data=biking)
```

```
C:\Users\ar_is\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variables as
keyword args: x, y. From version 0.12, the only valid positional argument will be `data`, and passing other argument
s without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```

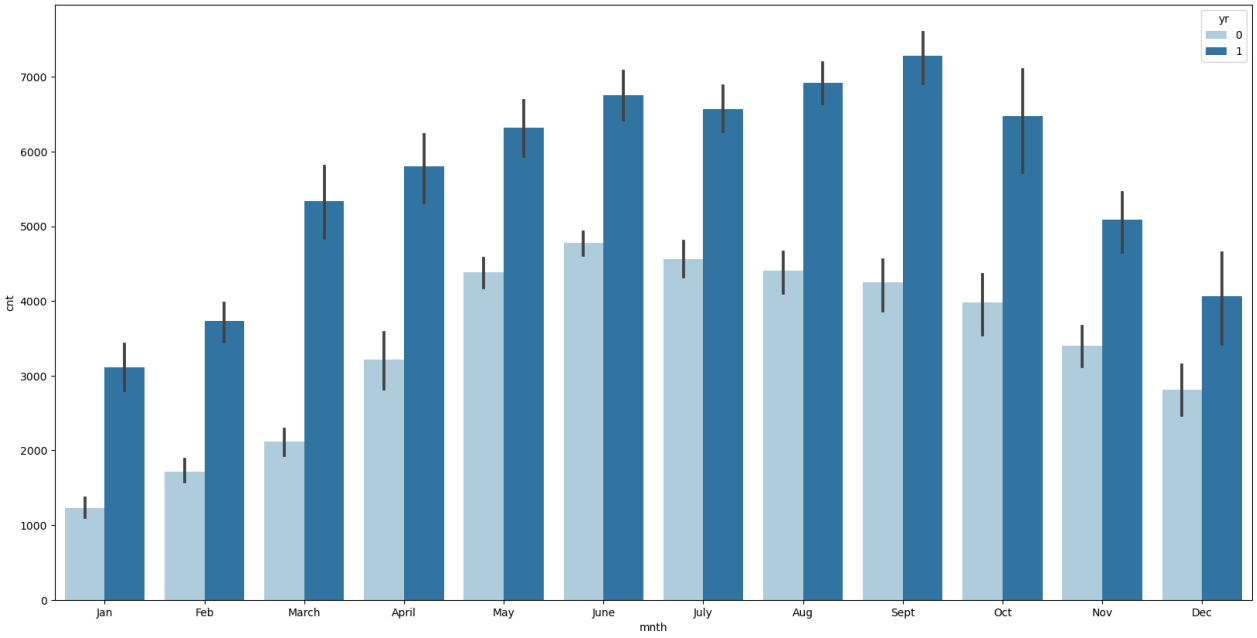
Out[121]: <AxesSubplot:xlabel='season', ylabel='cnt'>



In [122]:
```
plt.figure(figsize=(20,10))
sns.barplot('mnth', 'cnt',hue='yr', data=biking,palette = 'Paired')
```

```
C:\Users\ar_is\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variables as
keyword args: x, y. From version 0.12, the only valid positional argument will be `data`, and passing other argument
s without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```

Out[122]: <AxesSubplot:xlabel='mnth', ylabel='cnt'>



STEP 3 - Data Preparation

In [53]:
```
#Get Dummy Values for the categorical variables
dummy_vars = pd.get_dummies(biking[['season', 'weekday','mnth','weathersit']], drop_first=True)
```

In [54]:
```
# concat the dummy df with original df
biking = pd.concat([biking,dummy_vars], axis = 1)
```

In [55]:
```
# drop season column
biking.drop(['season','weekday','mnth','weathersit'], axis=1, inplace=True)
```

In [56]:
```
biking.head()
```

Out[56]:

	yr	holiday	workingday	temp	hum	windspeed	cnt	season_summer	season_fall	season_winter	...	mnth_May	mnth_June	mn
0	0	0	0	0.355170	0.767981	0.388102	0.110792	0	0	0	...	0	0	
1	0	0	0	0.379232	0.615202	0.635752	0.089623	0	0	0	...	0	0	
2	0	0	1	0.171000	0.254904	0.635105	0.152669	0	0	0	...	0	0	
3	0	0	1	0.175530	0.468123	0.387681	0.177174	0	0	0	...	0	0	
4	0	0	1	0.209120	0.254464	0.462471	0.181546	0	0	0	...	0	0	

5 rows × 29 columns

In [57]:
```
#Columns & Rows
biking.shape
```

Out[57]: (716, 29)

In [58]:
```
#info
biking.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 716 entries, 0 to 729
Data columns (total 29 columns):
 #   Column                          Non-Null Count  Dtype
---  ------                          --------------  -----
 0   yr                              716 non-null    category
 1   holiday                         716 non-null    category
 2   workingday                      716 non-null    category
 3   temp                            716 non-null    float64
 4   hum                             716 non-null    float64
 5   windspeed                       716 non-null    float64
 6   cnt                             716 non-null    float64
 7   season_summer                   716 non-null    uint8
 8   season_fall                     716 non-null    uint8
 9   season_winter                   716 non-null    uint8
 10  weekday_Mon                     716 non-null    uint8
 11  weekday_Tue                     716 non-null    uint8
 12  weekday_Wed                     716 non-null    uint8
 13  weekday_Thu                     716 non-null    uint8
 14  weekday_Fri                     716 non-null    uint8
 15  weekday_Sat                     716 non-null    uint8
 16  mnth_Feb                        716 non-null    uint8
 17  mnth_March                      716 non-null    uint8
 18  mnth_April                      716 non-null    uint8
 19  mnth_May                        716 non-null    uint8
 20  mnth_June                       716 non-null    uint8
 21  mnth_July                       716 non-null    uint8
 22  mnth_Aug                        716 non-null    uint8
 23  mnth_Sept                       716 non-null    uint8
 24  mnth_Oct                        716 non-null    uint8
 25  mnth_Nov                        716 non-null    uint8
 26  mnth_Dec                        716 non-null    uint8
 27  weathersit_Mist_Cloudy          716 non-null    uint8
 28  weathersit_LightSnow_LightRain  716 non-null    uint8
dtypes: category(3), float64(4), uint8(22)
memory usage: 62.0 KB
```

In [59]:
```
#As three columns are comig under categorical dtype so these have to be converted into uint8
# Convert categorical columns to numeric
biking[['yr','holiday','workingday']]= biking[['yr','holiday','workingday']].astype('uint8')
biking.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 716 entries, 0 to 729
Data columns (total 29 columns):
 #   Column                          Non-Null Count  Dtype
---  ------                          --------------  -----
 0   yr                              716 non-null    uint8
 1   holiday                         716 non-null    uint8
 2   workingday                      716 non-null    uint8
 3   temp                            716 non-null    float64
 4   hum                             716 non-null    float64
 5   windspeed                       716 non-null    float64
 6   cnt                             716 non-null    float64
 7   season_summer                   716 non-null    uint8
 8   season_fall                     716 non-null    uint8
 9   season_winter                   716 non-null    uint8
 10  weekday_Mon                     716 non-null    uint8
 11  weekday_Tue                     716 non-null    uint8
 12  weekday_Wed                     716 non-null    uint8
 13  weekday_Thu                     716 non-null    uint8
 14  weekday_Fri                     716 non-null    uint8
 15  weekday_Sat                     716 non-null    uint8
 16  mnth_Feb                        716 non-null    uint8
 17  mnth_March                      716 non-null    uint8
 18  mnth_April                      716 non-null    uint8
 19  mnth_May                        716 non-null    uint8
 20  mnth_June                       716 non-null    uint8
 21  mnth_July                       716 non-null    uint8
 22  mnth_Aug                        716 non-null    uint8
 23  mnth_Sept                       716 non-null    uint8
 24  mnth_Oct                        716 non-null    uint8
 25  mnth_Nov                        716 non-null    uint8
 26  mnth_Dec                        716 non-null    uint8
 27  weathersit_Mist_Cloudy          716 non-null    uint8
 28  weathersit_LightSnow_LightRain  716 non-null    uint8
dtypes: float64(4), uint8(25)
memory usage: 61.6 KB
```

3.2 - Scaling of Dataset

1) During EDA we could observe that there is different range of data in the data set. So it becomes important to scale the data.

2) Here we will be using Min-Max scaling (normalisation) to scale both training and tesing dataset.

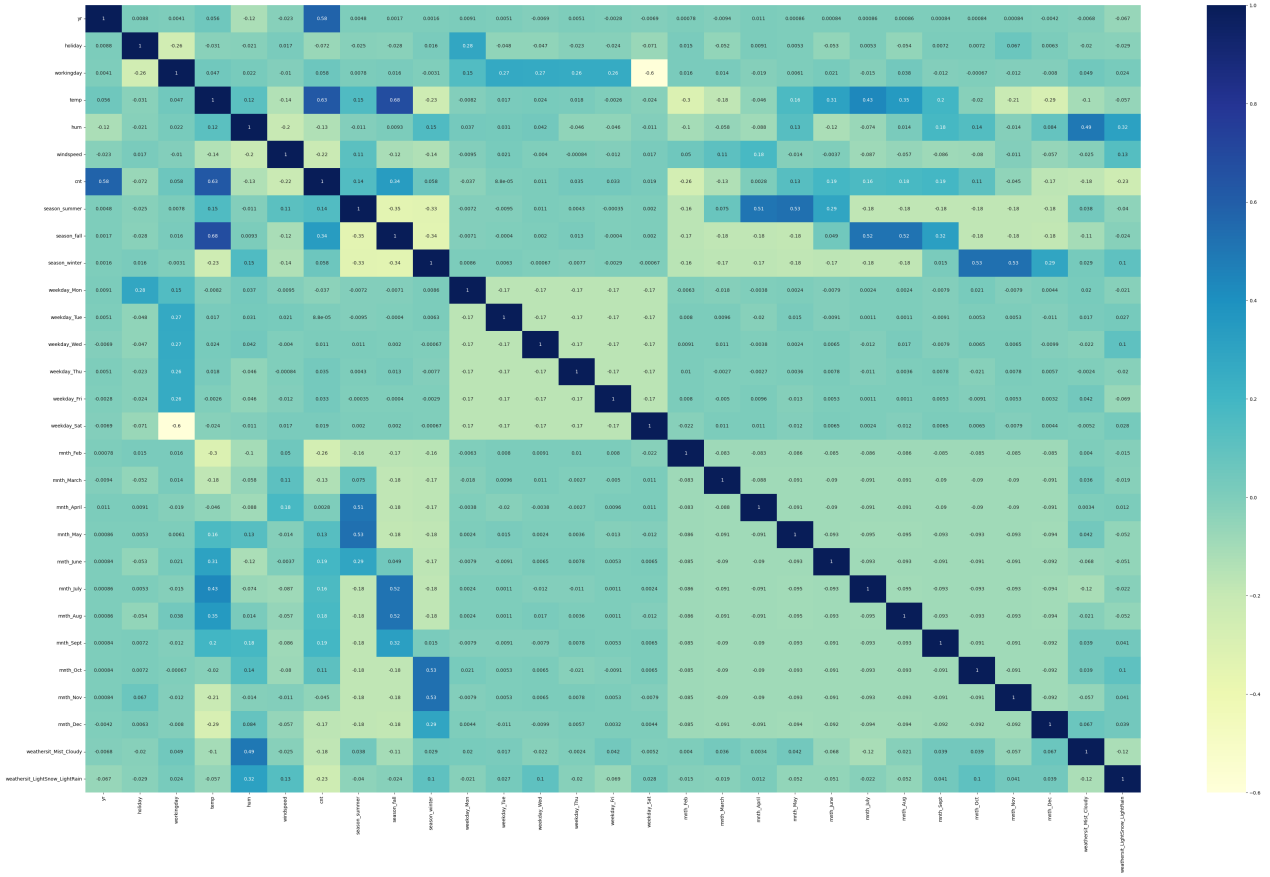
In [60]:
```
# 1. Instantiate an object
scaler = MinMaxScaler()

#Create a list of numeric vars
num_vars = ['temp', 'hum', 'windspeed', 'cnt']

#2. Fit on data
biking[num_vars] = scaler.fit_transform(biking[num_vars])
biking.head()
```

Out[60]:

	yr	holiday	workingday	temp	hum	windspeed	cnt	season_summer	season_fall	season_winter	...	mnth_May	mnth_June	mı
0	0	0	0	0.355170	0.767981	0.388102	0.110792	0	0	0	...	0	0	
1	0	0	0	0.379232	0.615202	0.635752	0.089623	0	0	0	...	0	0	
2	0	0	1	0.171000	0.254904	0.635105	0.152669	0	0	0	...	0	0	
3	0	0	1	0.175530	0.468123	0.387681	0.177174	0	0	0	...	0	0	
4	0	0	1	0.209120	0.254464	0.462471	0.181546	0	0	0	...	0	0	

5 rows × 29 columns

In [66]:
```
# Let's check the correlation coefficients to see which variables are highly correlated
#heatmap
plt.figure(figsize = (50, 30))
sns.heatmap(biking.corr(), annot = True, cmap = 'YlGnBu')
plt.show()
```



Step 4 : Making Prediction

4.1 - Splitting data into train and test

In [67]:
```
X = biking[['temp', 'hum', 'windspeed']]
y = biking['cnt']
```

In [68]:
```
from sklearn.model_selection import train_test_split
```

In [69]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.40, random_state=5)
```

In [70]:
```
from sklearn.linear_model import LinearRegression
```

In [71]:
```
lm = LinearRegression()
lm.fit(X_train,y_train)
rfe = RFE(lm)
rfe = rfe.fit(X_train, y_train)
```

In [72]:
```
list(zip(X_train.columns, rfe.support_, rfe.ranking_))
```

Out[72]:
```
[('temp', True, 1), ('hum', False, 2), ('windspeed', False, 3)]
```

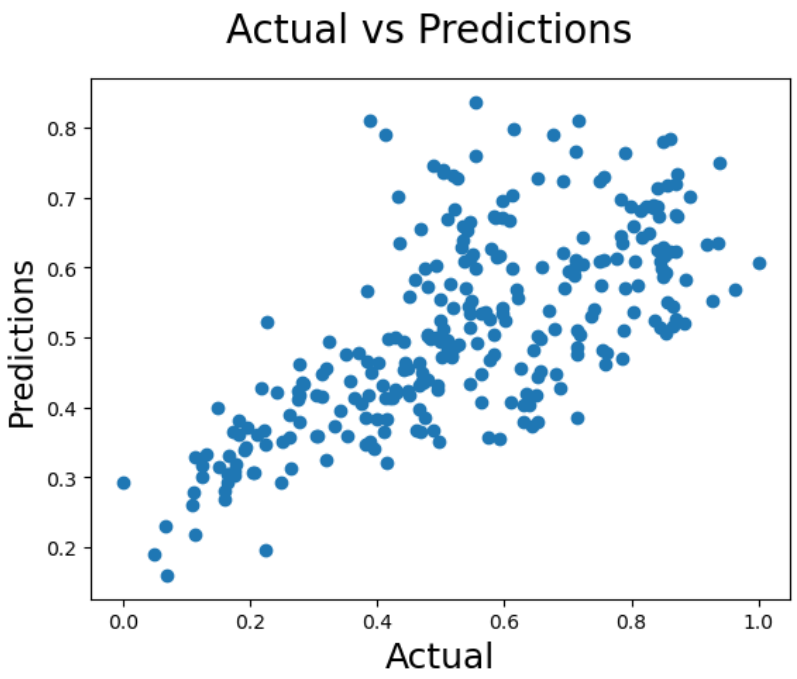
In [73]:
```
predictions = lm.predict(X_test)
```

Step -5 Model Evaluation

Step 5.1 Comparison

In [74]:
```
fig = plt.figure()
plt.scatter(y_test,predictions)
fig.suptitle('Actual vs Predictions', fontsize=20)
plt.xlabel('Actual', fontsize=18)
plt.ylabel('Predictions', fontsize=16)
```

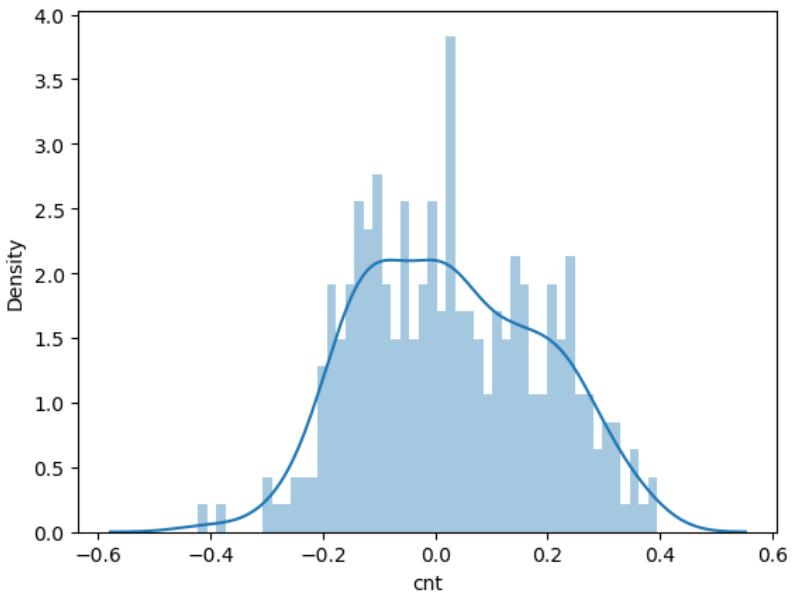
Out[74]:
```
Text(0, 0.5, 'Predictions')
```



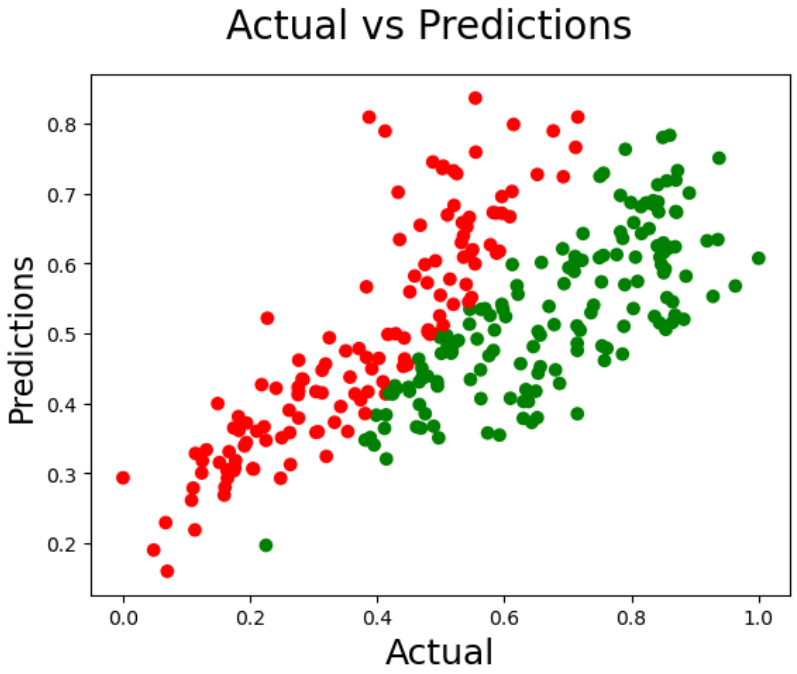
In [75]:
```
fig = plt.figure()
sns.distplot((y_test - predictions),bins=50)
```

```
C:\Users\ar_is\anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `distplot` is a deprecated
function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level fun
ction with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

Out[75]:
```
<AxesSubplot:xlabel='cnt', ylabel='Density'>
```



In [76]:
```
fig = plt.figure()
plt.scatter(y_test, predictions, c=['r' if y < p else 'g' for y, p in zip(y_test, predictions)])
fig.suptitle('Actual vs Predictions', fontsize=20)
plt.xlabel('Actual', fontsize=18)
plt.ylabel('Predictions', fontsize=16)
plt.show()
#red  - when actual value is less than predicted value and green is when actual value is greater than oredicted value
```



5.2 Evaluating Model

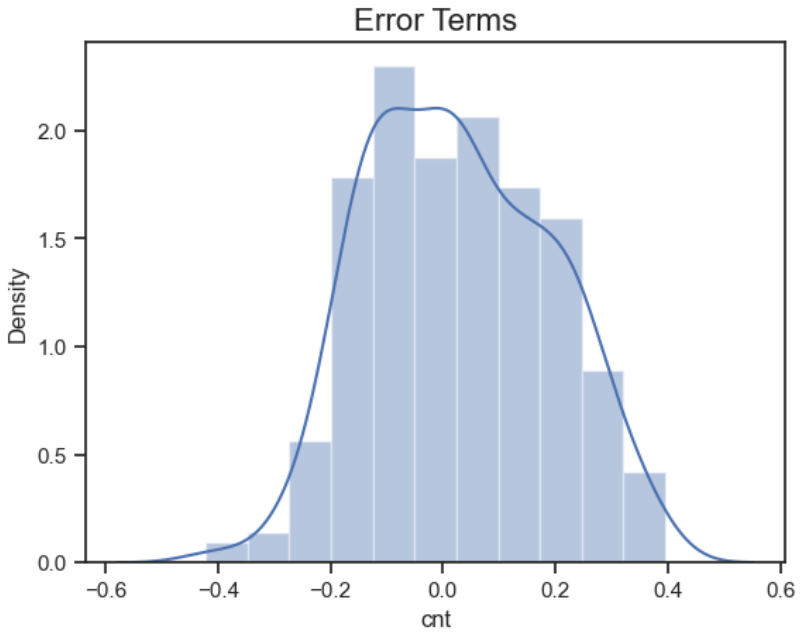
In [1025]:
```
from sklearn import metrics
```

In [1026]:
```
print('MAE:', metrics.mean_absolute_error(y_test, predictions))
print('MSE:', metrics.mean_squared_error(y_test, predictions))
print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test, predictions)))
```

```
MAE: 0.13497363534024703
MSE: 0.02681432396283249
RMSE: 0.16375079835784767
```

In [1027]:
```
res_test = y_test - predictions
plt.title('Error Terms', fontsize=16)
sns.distplot(res_test)
plt.show()
```

```
C:\Users\ar_is\anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `distplot` is a deprecated
function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level fun
ction with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```



In [1028]:
```
lm = sm.OLS(y_train,X_train).fit()
```

In [1029]:
```
lm.summary()
```

	coef	std err	t	P>|t|	[0.025	0.975]
temp	0.7753	0.033	23.708	0.000	0.711	0.840
hum	0.0231	0.035	0.650	0.516	-0.047	0.093
windspeed	0.1223	0.033	3.748	0.000	0.058	0.186

Omnibus:	2.533	**Durbin-Watson:**	1.996
Prob(Omnibus):	0.282	**Jarque-Bera (JB):**	2.256
Skew:	0.087	**Prob(JB):**	0.324
Kurtosis:	2.691	**Cond. No.**	4.48

Notes:
[1] R^2 is computed without centering (uncentered) since the model does not contain a constant.
[2] Standard Errors assume that the covariance matrix of the errors is correctly specified.

In [ ]:

Step -6 Residual Analysis of Train Data

In [1030]:
```
y_train_pred = lm.predict(X_train)
```

In [1031]:
```
fig = plt.figure()
sns.distplot((y_train - y_train_pred), bins = 20)
fig.suptitle('Error Terms', fontsize = 20)                  # Plot heading
plt.xlabel('Errors', fontsize = 18)
```

```
C:\Users\ar_is\anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `distplot` is a deprecated
function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level fun
ction with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

Out[1031]: Text(0.5, 0, 'Errors')