Business Case: Yulu - Hypothesis Testing

About YULU:

Yulu is India's leading micro-mobility service provider, which offers unique vehicles for the daily commute. Starting off as a mission to eliminate traffic congestion in India, Yulu provides the safest commute solution through a user-friendly mobile app to enable shared, solo and sustainable commuting. Yulu zones are located at all the appropriate locations (including metro stations, bus stands, office spaces, residential areas, corporate offices, etc) to make those first and last miles smooth, affordable, and convenient! Yulu has recently suffered considerable dips in its revenues. They have contracted a consulting company to understand the factors on which the demand for these shared electric cycles depends. Specifically, they want to understand the factors affecting the demand for these shared electric cycles in the Indian market.

Project Goal:

1. Which variables are significant in predicting the demand for shared electric cycles in the Indian market?
2. How well those variables describe the electric cycles demands
3. How to handle categorical & numerical features
4. How to deal with missing data
5. Feature Engineering

Steps to be followed while doing this project:

1. Import the dataset and do usual exploratory data analysis steps like checking the structure & characteristics of the dataset(i.e to understand the patterns in the data)
2. Trying to establish relation between the dependent and independent variable (Dependent: Count & Independent: Workingday, Weather, Season etc.,)
3. Selecting appropriate test to check weather:

```
* Working Day has effect on numeric of electric cycles rented
* No of cycles rented similar or different in different seasons
* No of cycles rented or different in different weather Weather is dependent on season
```

4. Set up Null Hypothesis(H0)
5. State the alternate hypothesis(Ha)
6. Check assumptions of the test(Normality, Equal Variance). Need to check using Histogram/ Q-Q plot / Statistical method like levene's test, Shapiro-wilk test(optional)
7. Set a significance level(Alpha)
8. Calculate test Statistics
9. Decision to accept or reject null hypothesis
10. Inference from the analysis

```
import numpy as np, pandas as pd, seaborn as sns, matplotlib.pyplot as plt
from scipy.stats import norm, poisson, binom
from statsmodels.stats.weightstats import ztest
from scipy.stats import ttest_1samp, ttest_ind
from scipy.stats import f_oneway
from scipy.stats import chi2_contingency
from scipy.stats import spearmanr
from scipy.stats import t
import plotly.express as px
```

```
!wget https://d2beiqkhq929f0.cloudfront.net/public_assets/assets/000/001/428/original/bike_sharing.csv
```

```
--2023-08-15 19:52:12--  https://d2beiqkhq929f0.cloudfront.net/public_assets/assets/000/001/428/original/bike_sharing.csv
Resolving d2beiqkhq929f0.cloudfront.net (d2beiqkhq929f0.cloudfront.net)... 18.172.139.46, 18.172.139.94, 18.172.139.61, ...
Connecting to d2beiqkhq929f0.cloudfront.net (d2beiqkhq929f0.cloudfront.net)|18.172.139.46|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 648353 (633K) [text/plain]
Saving to: 'bike_sharing.csv'
```

```
bike_sharing.csv    100%[===================>] 633.16K  --.-KB/s    in 0.05s

2023-08-15 19:52:12 (12.0 MB/s) - 'bike_sharing.csv' saved [648353/648353]
```

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

	datetime	season	holiday	workingday	weather	temp	atemp	humidity	windspeed	casual	registered	count
0	2011-01-01 00:00:00	1	0	0	1	9.84	14.395	81	0.0	3	13	16
1	2011-01-01 01:00:00	1	0	0	1	9.02	13.635	80	0.0	8	32	40
2	2011-01-01 02:00:00	1	0	0	1	9.02	13.635	80	0.0	5	27	32
3	2011-01-01 03:00:00	1	0	0	1	9.84	14.395	75	0.0	3	10	13
4	2011-01-01 04:00:00	1	0	0	1	9.84	14.395	75	0.0	0	1	1

Step 1: Define Problem Statement and perform Exploratory Data Analysis

```
df.shape
```

```
(10886, 12)
```

Initial Observations:

1. Here we have in total 10886 records of bike rented(each record shows how many bikes were rented during that specific hour of the day.)
2. All the above data with 12 features such as datetime, season, holiday etc.,

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10886 entries, 0 to 10885
Data columns (total 12 columns):
 #   Column      Non-Null Count  Dtype  
---  ------      --------------  -----  
 0   datetime    10886 non-null  object 
 1   season      10886 non-null  int64  
 2   holiday     10886 non-null  int64  
 3   workingday  10886 non-null  int64  
 4   weather     10886 non-null  int64  
 5   temp        10886 non-null  float64
 6   atemp       10886 non-null  float64
 7   humidity    10886 non-null  int64  
 8   windspeed   10886 non-null  float64
 9   casual      10886 non-null  int64  
 10  registered  10886 non-null  int64  
 11  count       10886 non-null  int64  
dtypes: float64(3), int64(8), object(1)
memory usage: 1020.7+ KB
```

```
df.dtypes == "object"
```

```
datetime       True
season        False
holiday       False
workingday    False
weather       False
temp          False
atemp         False
humidity      False
windspeed     False
casual        False
registered    False
count         False
dtype: bool
```

Now we need to convert datetime from categorical to numerical, here datetime is is only object type variable and remaining all features are in numerical format

```
df.datetime.value_counts()
```

```
2011-01-01 00:00:00    1
2012-05-01 21:00:00    1
2012-05-01 13:00:00    1
```

```
2012-05-01 14:00:00    1
2012-05-01 15:00:00    1
                      ..
2011-09-02 04:00:00    1
2011-09-02 05:00:00    1
2011-09-02 06:00:00    1
2011-09-02 07:00:00    1
2012-12-19 23:00:00    1
Name: datetime, Length: 10886, dtype: int64
```

```
#Checking the characteristics of the Data
df.describe()
```

	season	holiday	workingday	weather	temp	atemp	humidity	windspeed	casual	re
count	10886.000000	10886.000000	10886.000000	10886.000000	10886.00000	10886.000000	10886.000000	10886.000000	10886.000000	108
mean	2.506614	0.028569	0.680875	1.418427	20.23086	23.655084	61.886460	12.799395	36.021955	1
std	1.116174	0.166599	0.466159	0.633839	7.79159	8.474601	19.245033	8.164537	49.960477	1
min	1.000000	0.000000	0.000000	1.000000	0.82000	0.760000	0.000000	0.000000	0.000000	
25%	2.000000	0.000000	0.000000	1.000000	13.94000	16.665000	47.000000	7.001500	4.000000	
50%	3.000000	0.000000	1.000000	1.000000	20.50000	24.240000	62.000000	12.998000	17.000000	1
75%	4.000000	0.000000	1.000000	2.000000	26.24000	31.060000	77.000000	16.997900	49.000000	2
max	4.000000	1.000000	1.000000	4.000000	41.00000	45.455000	100.000000	56.996900	367.000000	8

```
df.describe(include = "object")
```

	datetime
count	10886
unique	10886
top	2011-01-01 00:00:00
freq	1

Trying to establish relation between Dependent(count) & Independent variables(Workingday/ Weather/ Season etc) i.e Bi-Variate Analysis From the above Univariate graphical and numerical representation datatype of the following attributes needs to be changed to proper data types i.e

- datetime > to datetime
- season > to categorical
- holiday > to categorical
- workingday > to categorical
- weather > to categorical

```
df["datetime"] = pd.to_datetime(df["datetime"])
df["datetime"]
```

```
0       2011-01-01 00:00:00
1       2011-01-01 01:00:00
2       2011-01-01 02:00:00
3       2011-01-01 03:00:00
4       2011-01-01 04:00:00
                ...
10881   2012-12-19 19:00:00
10882   2012-12-19 20:00:00
10883   2012-12-19 21:00:00
10884   2012-12-19 22:00:00
10885   2012-12-19 23:00:00
Name: datetime, Length: 10886, dtype: datetime64[ns]
```

```
cat_cols= ['season', 'holiday', 'workingday', 'weather']
for col in cat_cols:
  df[col] = df[col].astype('object')
df.iloc[:, 1:].describe(include='all')
```

	season	holiday	workingday	weather	temp	atemp	humidity	windspeed	casual	registered	
count	10886.0	10886.0	10886.0	10886.0	10886.00000	10886.000000	10886.000000	10886.000000	10886.000000	10886.000000	10886
unique	4.0	2.0	2.0	4.0	NaN	NaN	NaN	NaN	NaN	NaN	
top	4.0	0.0	1.0	1.0	NaN	NaN	NaN	NaN	NaN	NaN	
freq	2734.0	10575.0	7412.0	7192.0	NaN	NaN	NaN	NaN	NaN	NaN	
mean	NaN	NaN	NaN	NaN	20.23086	23.655084	61.886460	12.799395	36.021955	155.552177	191
std	NaN	NaN	NaN	NaN	7.79159	8.474601	19.245033	8.164537	49.960477	151.039033	181
min	NaN	NaN	NaN	NaN	0.82000	0.760000	0.000000	0.000000	0.000000	0.000000	1
25%	NaN	NaN	NaN	NaN	13.94000	16.665000	47.000000	7.001500	4.000000	36.000000	42
50%	NaN	NaN	NaN	NaN	20.50000	24.240000	62.000000	12.998000	17.000000	118.000000	145

```
# minimum datetime and maximum datetime
df['datetime'].min(), df['datetime'].max()
```

```
(Timestamp('2011-01-01 00:00:00'), Timestamp('2012-12-19 23:00:00'))
```

```
# number of unique values in each categorical columns
df[cat_cols].melt().groupby(['variable', 'value'])[['value']].count()
```

		value
variable	**value**	
holiday	**0**	10575
	1	311
season	**1**	2686
	2	2733
	3	2733
	4	2734
weather	**1**	7192
	2	2834
	3	859
	4	1
workingday	**0**	3474
	1	7412

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10886 entries, 0 to 10885
Data columns (total 12 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   datetime    10886 non-null  datetime64[ns]
 1   season      10886 non-null  object
 2   holiday     10886 non-null  object
 3   workingday  10886 non-null  object
 4   weather     10886 non-null  object
 5   temp        10886 non-null  float64
 6   atemp       10886 non-null  float64
 7   humidity    10886 non-null  int64
 8   windspeed   10886 non-null  float64
 9   casual      10886 non-null  int64
 10  registered  10886 non-null  int64
 11  count       10886 non-null  int64
dtypes: datetime64[ns](1), float64(3), int64(4), object(4)
memory usage: 1020.7+ KB
```

Now Categorical variables are changed to object type and datetime attribute is changed to datetime variable & numerical attributes are not changed.

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

```
datetime      0
season        0
```

```
holiday       0
workingday    0
weather       0
temp          0
atemp         0
humidity      0
windspeed     0
casual        0
registered    0
count         0
dtype: int64
```

Observations:

1. No nulll values till now.
2. Working days(at 10575) are more compared to holidays(at 311)
3. Almost all 4 seasons counts are similar
4. Weather 1(Clear, Few clouds, partly cloudy, partly cloudy) is highest with 7192 and weather 4(Heavy Rain + Ice Pallets + Thunderstorm + Mist, Snow + Fog) is lowest at 1
5. No of working days is 7412 while non working days is 3474

```
#Unique values per columns
df.nunique()
```

```
datetime      10886
season            4
holiday           2
workingday        2
weather           4
temp             49
atemp            60
humidity         89
windspeed        28
casual          309
registered      731
count           822
dtype: int64
```

```
#Data Pre-processing Feature Extraction
df["weather"].replace({1:"Clear", 2:"Cloudy", 3:"Little Rain", 4:"Heavy Rain"}, inplace = True)
df["season"].replace({1:"Spring", 2:"Summer", 3:"Fall", 4:"Winter"}, inplace = True)
df["workingday"].replace({1:"Yes", 0:"No"}, inplace = True)
df["datetime"] = pd.to_datetime(df["datetime"])
df["holiday"].replace({1:"Yes", 0:"No"}, inplace = True)
df["day"] = df["datetime"].dt.day_name()
df["date"] = df["datetime"].dt.date
df["hour"] = df["datetime"].dt.hour
df["Month"] = df["datetime"].dt.month
df["Month_name"] = df["datetime"].dt.month_name()
df["year"] = df["datetime"].dt.year
```

```
df
```

	datetime	season	holiday	workingday	weather	temp	atemp	humidity	windspeed	casual	registered	count	day	dat
0	2011-01-01 00:00:00	Spring	No	No	Clear	9.84	14.395	81	0.0000	3	13	16	Saturday	2011 01-0
1	2011-01-01 01:00:00	Spring	No	No	Clear	9.02	13.635	80	0.0000	8	32	40	Saturday	2011 01-0
2	2011-01-01 02:00:00	Spring	No	No	Clear	9.02	13.635	80	0.0000	5	27	32	Saturday	2011 01-0
3	2011-01-01 03:00:00	Spring	No	No	Clear	9.84	14.395	75	0.0000	3	10	13	Saturday	2011 01-0
	2011-01-													2011

Describing Statistical summary of independent numerical features:

```
pd.DataFrame(df["atemp"].describe()).T
```

	count	mean	std	min	25%	50%	75%	max
atemp	10886.0	23.655084	8.474601	0.76	16.665	24.24	31.06	45.455

```
def get_temp(temp):
  if temp <= 12: return "Very low"
  elif temp > 12 and temp < 24: return "Low"
  elif temp >= 24 and temp < 35: return "Moderate"
  elif temp >= 35: return "High"
```

2012-12-

```
df["temperature"] = pd.Series(map(get_temp, df["atemp"]))
```

10886 rows × 18 columns

```
pd.DataFrame(df["humidity"].describe()).T
```

	count	mean	std	min	25%	50%	75%	max
humidity	10886.0	61.88646	19.245033	0.0	47.0	62.0	77.0	100.0

```
def get_humidity(H):
  if 0 <= H <= 10:
    return "10%"
  elif 11 <= H <= 20:
    return "20%"
  elif 21 <= H <= 30:
    return "30%"
  elif 31 <= H <= 40:
    return "40%"
  elif 41 <= H <= 50:
    return "50%"
  elif 51 <= H <= 60:
    return "60%"
  elif 61 <= H <= 70:
    return "70%"
  elif 71 <= H <= 80:
    return "80%"
  elif 81 <= H <= 90:
    return "90%"
  elif 91 <= H <= 100:
    return "100%"
df["gethumidity"] = pd.Series(map(get_humidity,df["humidity"]))
pd.DataFrame(df["windspeed"].describe()).T
```

	count	mean	std	min	25%	50%	75%	max
windspeed	10886.0	12.799395	8.164537	0.0	7.0015	12.998	16.9979	56.9969

```
df["windspeed_category"] = pd.qcut(df["windspeed"],8)
df["windspeed_category"] = df["windspeed_category"].astype("object")
```

Data information:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10886 entries, 0 to 10885
Data columns (total 21 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   datetime            10886 non-null  datetime64[ns]
 1   season              10886 non-null  object
 2   holiday             10886 non-null  object
 3   workingday          10886 non-null  object
 4   weather             10886 non-null  object
 5   temp                10886 non-null  float64
 6   atemp               10886 non-null  float64
 7   humidity            10886 non-null  int64
 8   windspeed           10886 non-null  float64
 9   casual              10886 non-null  int64
 10  registered          10886 non-null  int64
 11  count               10886 non-null  int64
 12  day                 10886 non-null  object
 13  date                10886 non-null  object
 14  hour                10886 non-null  int64
 15  Month               10886 non-null  int64
 16  Month_name          10886 non-null  object
 17  year                10886 non-null  int64
 18  temperature         10886 non-null  object
 19  gethumidity         10886 non-null  object
 20  windspeed_category  10886 non-null  object
dtypes: datetime64[ns](1), float64(3), int64(7), object(10)
memory usage: 1.7+ MB
```

Statistical summary about categorical data:

```
df.describe(include=["object","category"])
```

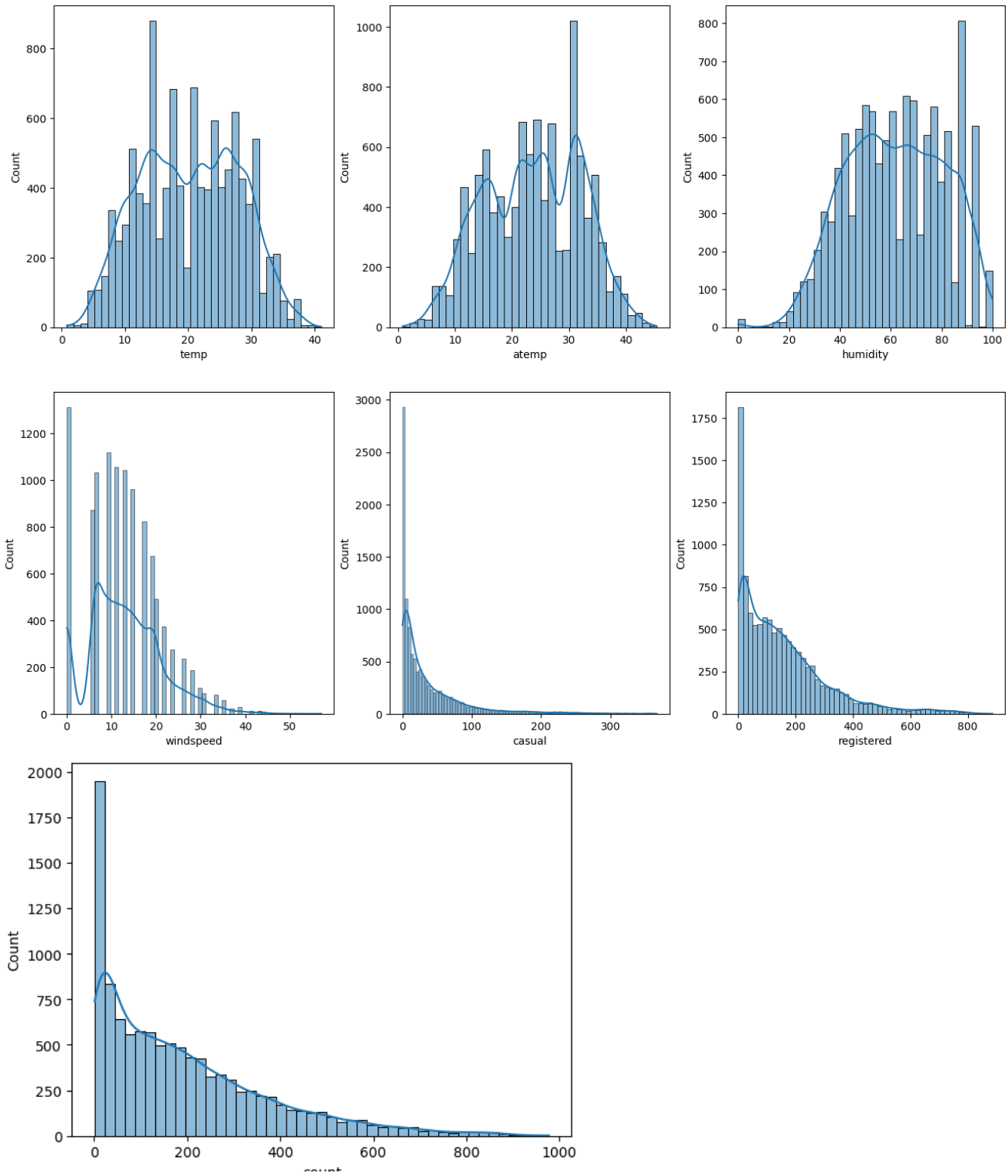
	season	holiday	workingday	weather	day	date	Month_name	temperature	gethumidity	windspeed_category
count	10886	10886	10886	10886	10886	10886	10886	10886	10886	10886
unique	4	2	2	4	7	456	12	4	10	8
top	Winter	No	Yes	Clear	Saturday	2011-01-01	May	Moderate	70%	(-0.001, 6.003]
freq	2734	10575	7412	7192	1584	24	912	4767	1845	2185

Observations:

1. Moderate level Temperature frequency is highest in given data
2. 70% is the highest humidity level
3. May month stands top
4. Most preferable windspeed 8-12

Univariate Analysis:

```
# understanding the distribution for numerical variables
num_cols = ['temp', 'atemp', 'humidity', 'windspeed', 'casual', 'registered','count']
fig, axis = plt.subplots(nrows=2, ncols=3, figsize=(16, 12))
index = 0
for row in range(2):
  for col in range(3):
    sns.histplot(df[num_cols[index]], ax=axis[row, col], kde=True)
    index += 1
plt.show()
sns.histplot(df[num_cols[-1]], kde=True)
plt.show()
```



Observations:

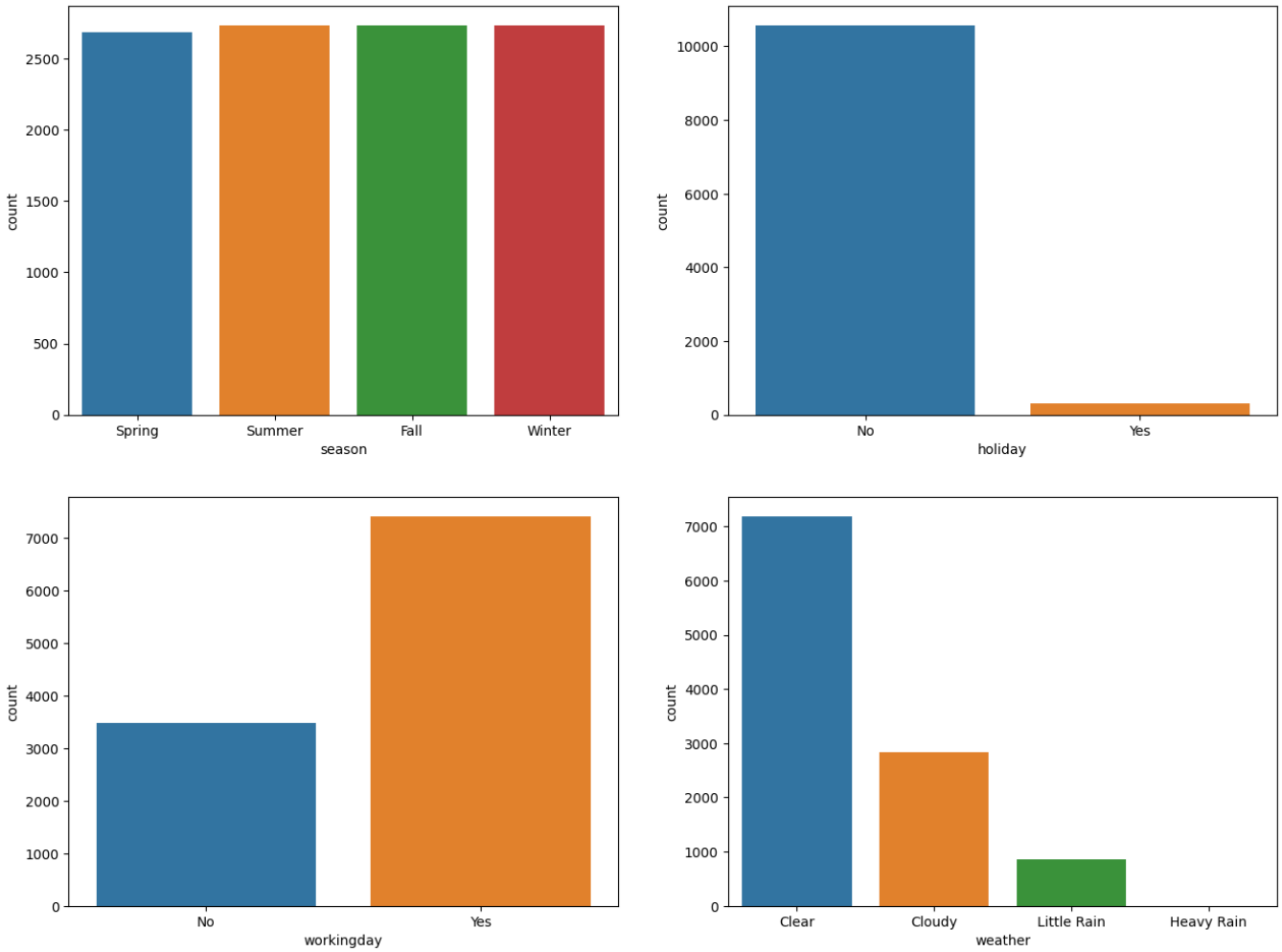
1. Casual, registered and count almost look like a log normal distribution
2. temp, atemp and humidity mostly looks like a normal distribution
3. Windspeed is very similar to binomial distribution

```
# plotting box plots to detect outliers in the data
fig, axis = plt.subplots(nrows=2, ncols=3, figsize=(16, 12))
index = 0
for row in range(2):
  for col in range(3):
    sns.boxplot(x=df[num_cols[index]], ax=axis[row, col])
    index += 1
plt.show()
sns.boxplot(x=df[num_cols[-1]])
plt.show()
```

Observations:

1. From above plot windspeed, casual, registered and count have outliers in the data

```
# countplot of each categorical column
fig, axis = plt.subplots(nrows=2, ncols=2, figsize=(16, 12))
index = 0
for row in range(2):
  for col in range(2):
    sns.countplot(data=df, x=cat_cols[index], ax=axis[row, col])
    index += 1
plt.show()
```



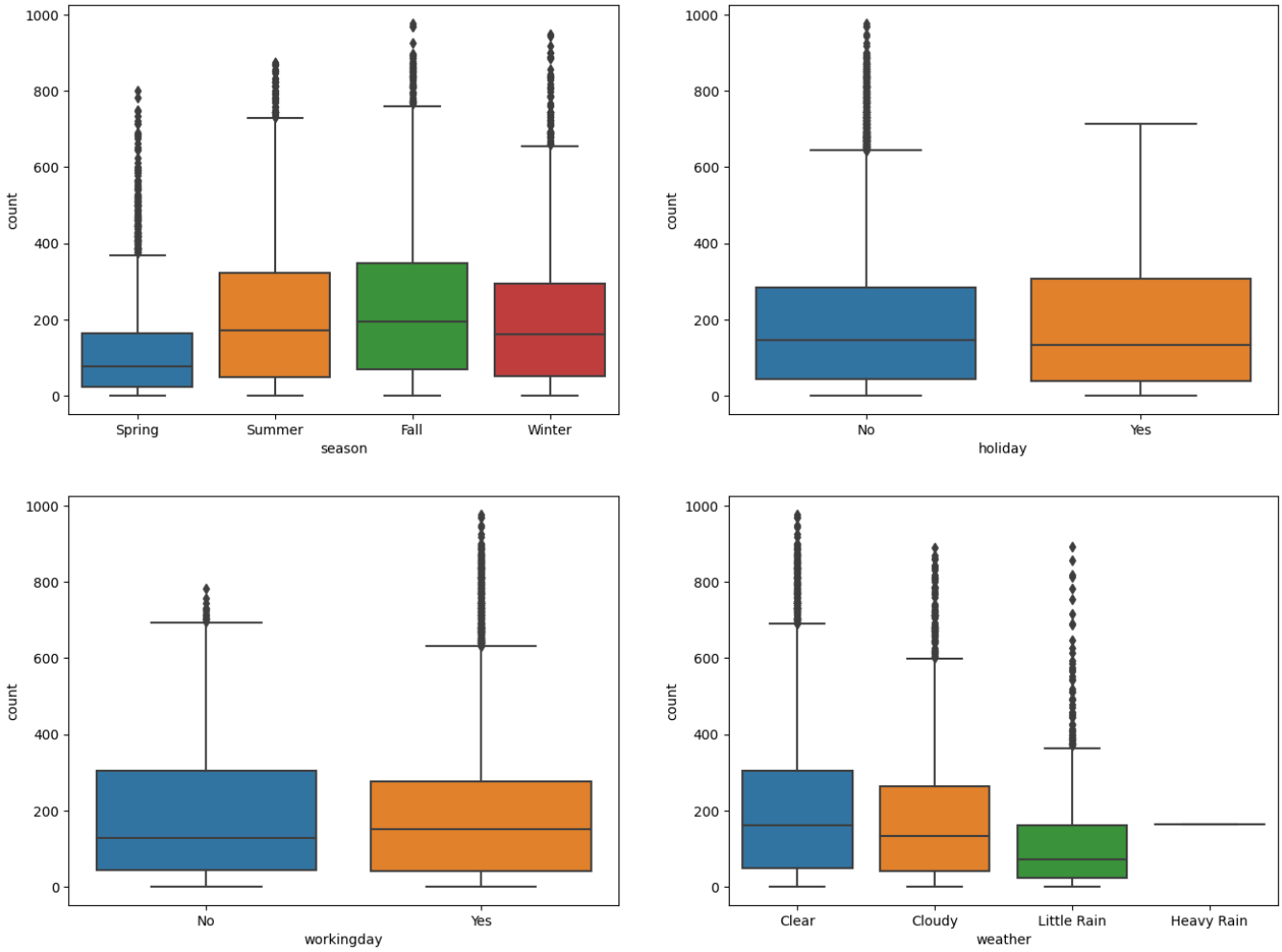
Observations:

1. As per the above plots the season days are more or less same in number
2. There are more number of working days than holidays
3. Mostly weather is Clear, Few clouds, partly cloudy, partly cloudy
4. Overall the data looks very common

Bivariate Analysis

```
# plotting categorical variables againt count using boxplots
fig, axis = plt.subplots(nrows=2, ncols=2, figsize=(16, 12))
index = 0
for row in range(2):
```

```
    for col in range(2):
      sns.boxplot(data=df, x=cat_cols[index], y='count', ax=axis[row, col])
      index += 1
plt.show()
```



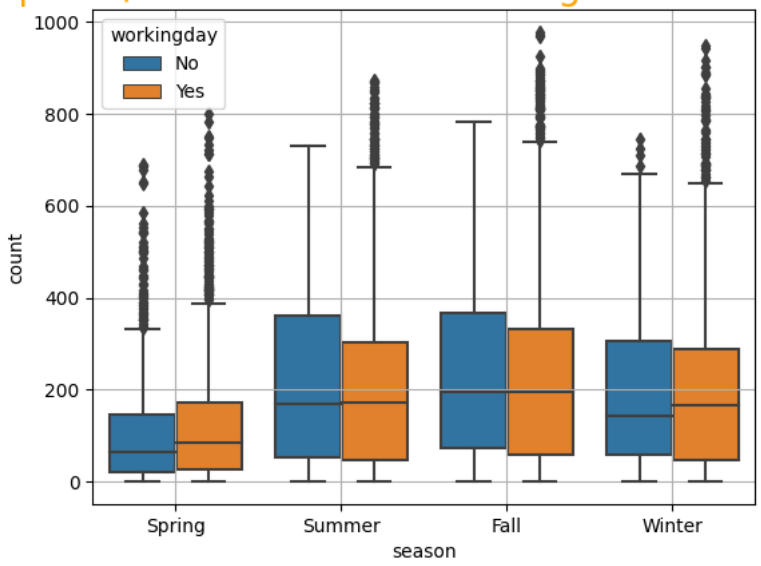
Multivariate Analysis:

```
sns.boxplot(data = df, x = "season", y = "count", hue = "holiday")
plt.grid()
plt.title("Boxplot b/w season & count along with holiday season" , fontsize = 20 , color = "orange" )
plt.show()
```

Boxplot b/w season & count along with holiday season

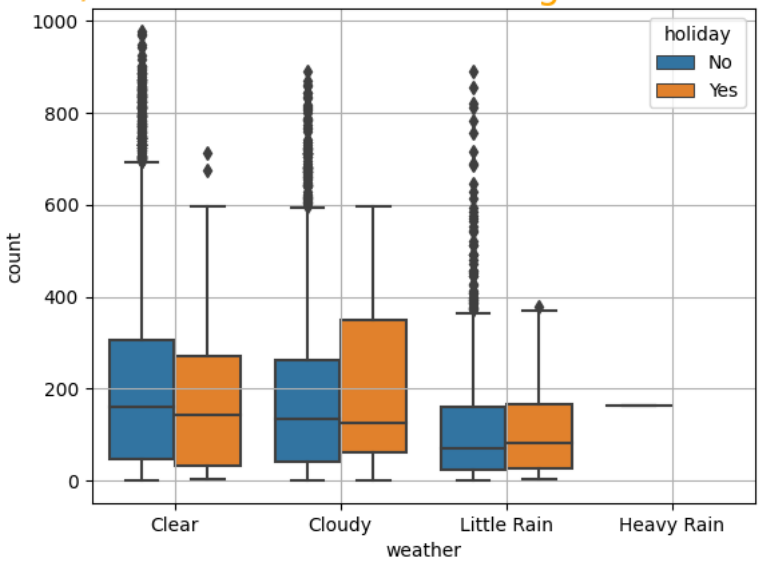

```
sns.boxplot(data = df, x = "season", y = "count", hue = "workingday")
plt.grid()
plt.title("Boxplot b/w season & count along with workingday" , fontsize = 20 , color = "orange" )
plt.show()
```

Boxplot b/w season & count along with workingday

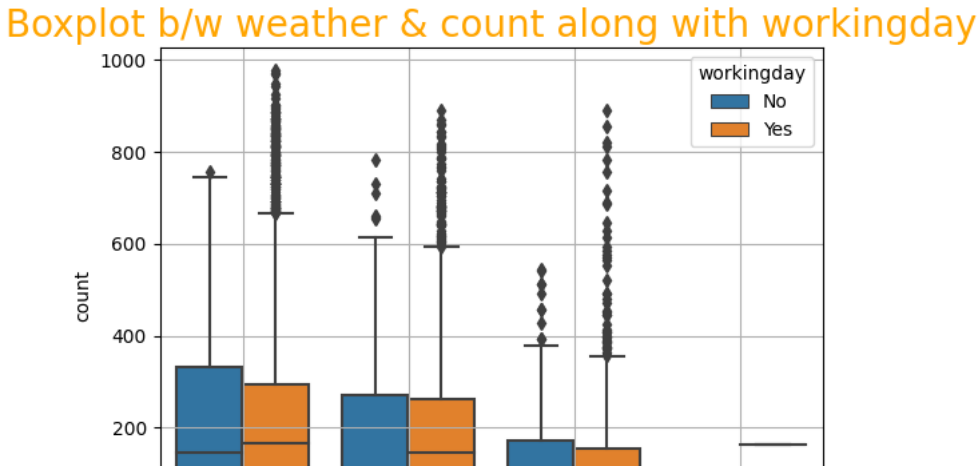



```
sns.boxplot(data = df, x = "weather", y = "count", hue = "holiday")
plt.grid()
plt.title("Boxplot b/w weather & count along with holiday season" , fontsize = 20 , color = "orange" )
plt.show()
```

Boxplot b/w weather & count along with holiday season



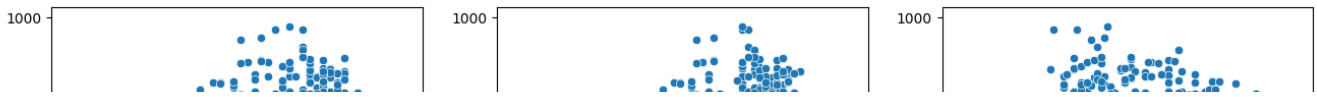

```
sns.boxplot(data = df, x = "weather", y = "count", hue = "workingday")
plt.grid()
plt.title("Boxplot b/w weather & count along with workingday" , fontsize = 20 , color = "orange" )
plt.show()
```



Observations:

1. As per the above plots, w.r.t seasons more number of bikes were rented on summer and fall seasons next comes winter and then comes spring season
2. More bikes are rented whenever there is a holiday
3. Slightly more number of bikes were rented on a non workingday/holiday
4. Less number of bikes were rented on a Heavy Rain + Ice Pallets + Thunderstorm + Mist, Snow + Fog day

```
# plotting numerical variables againt count using scatterplot
fig, axis = plt.subplots(nrows=2, ncols=3, figsize=(16, 12))
index = 0
for row in range(2):
  for col in range(3):
    sns.scatterplot(data=df, x=num_cols[index], y='count', ax=axis[row, col])
    index += 1
plt.show()
```

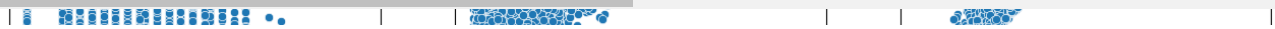


Observations:

1. No of bikes rented fell when the humidity is less than 20
2. Less no of bikes were rented when windspeed is greater than 35
3. No of nikes rented fell when temp, atemp fall below 10
4. The range of casual users ranges from 0 to 300
5. The range of registered users ranges from 0 to 800 approx.

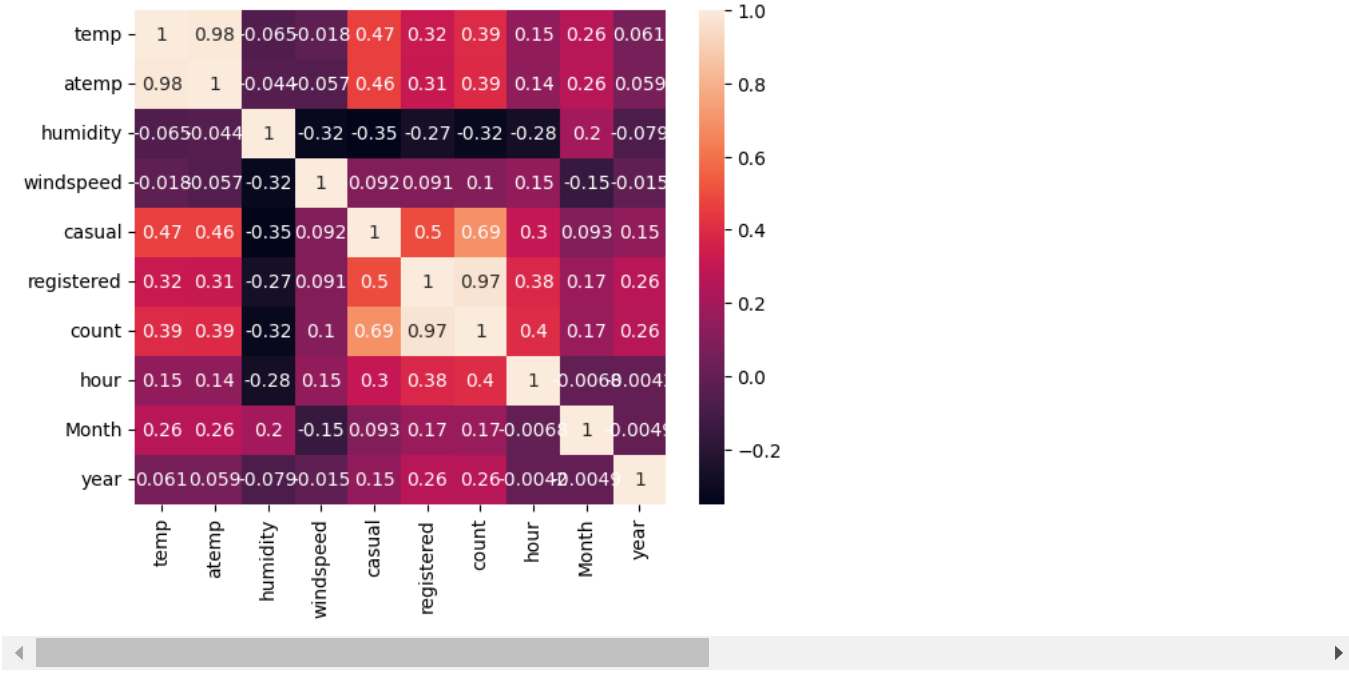
```
# understanding the correlation between count and numerical variables
df.corr()['count']
```

```
<ipython-input-36-d24438c865eb>:2: FutureWarning: The default value of numeric_only in DataFrame.corr is deprecated. In a future ve
  df.corr()['count']
temp          0.394454
atemp         0.389784
humidity     -0.317371
windspeed     0.101369
casual        0.690414
registered    0.970948
count         1.000000
hour          0.400601
Month         0.166862
year          0.260403
Name: count, dtype: float64
```



```
sns.heatmap(df.corr(), annot=True)
plt.show()
```

```
<ipython-input-37-6522c2b4e5f9>:1: FutureWarning: The default value of numeric_only in DataFrame.corr is deprecated. In a future ve
  sns.heatmap(df.corr(), annot=True)
```



Observations:

1. Correlation between Temperature and Number of Cycles Rented for all customers : 0.39
2. Correlation between Temperature and Number of Cycles Rented for casual subscribers : 0.46
3. Correlation between Temperature and Number of Cycles Rented for registered subscribers : 0.31
4. Correlation between Temperature and Number of Cycles Rented for registered subscribers : 0.31
5. Humidity has a negative correlation with the number of cycles rented which is -0.32

Pre-processed Data Sample :

```
df.sample(10)
```

	datetime	season	holiday	workingday	weather	temp	atemp	humidity	windspeed	casual	...	count	day	date	hour
6078	2012-02-09 12:00:00	Spring	No	Yes	Clear	10.66	11.365	48	22.0028	10	...	158	Thursday	2012-02-09	12
1647	2011-04-14 13:00:00	Summer	No	Yes	Clear	22.96	26.515	37	15.0013	38	...	159	Thursday	2011-04-14	13
1996	2011-05-10 02:00:00	Summer	No	Yes	Clear	18.04	21.970	58	8.9981	1	...	5	Tuesday	2011-05-10	2
3026	2011-07-15 00:00:00	Fall	No	Yes	Clear	25.42	29.545	73	15.0013	23	...	80	Friday	2011-07-15	0
1178	2011-03-13 19:00:00	Spring	No	No	Clear	14.76	16.665	50	23.9994	28	...	101	Sunday	2011-03-13	19
4192	2011-10-06 17:00:00	Winter	No	Yes	Clear	22.96	26.515	52	8.9981	63	...	568	Thursday	2011-10-06	17
9576	2012-10-03 09:00:00	Winter	No	Yes	Cloudy	25.42	28.790	83	12.9980	30	...	362	Wednesday	2012-10-03	9
388	2011-01-17 16:00:00	Spring	Yes	No	Cloudy	8.20	10.605	47	11.0014	6	...	82	Monday	2011-01-17	16
6333	2012-03-01 03:00:00	Spring	No	Yes	Clear	18.86	22.725	94	15.0013	0	...	3	Thursday	2012-03-01	3
7272	2012-05-02 09:00:00	Summer	No	Yes	Cloudy	22.96	26.515	88	0.0000	26	...	315	Wednesday	2012-05-02	9

10 rows × 21 columns

About the features :

dependent variables : count / registerd / casual

independent variables : workingday / holiday / weather / seasons /temperature /humidity /windspeed.

Outlier detection in Dataset :

```
def detect_outliers(df):
  length_before = len(df)
  Q1 = np.percentile(df,25)
  Q3 = np.percentile(df,75)
  IQR = Q3-Q1
  upperbound = Q3+1.5*IQR
  lowerbound = Q1-1.5*IQR
  if lowerbound < 0:
    lowerbound = 0
  length_after = len(df[(df>lowerbound)&(df<upperbound)])
  return f"{np.round((length_before-length_after)/length_before,4)} % Outliers data from input data found"
rentedCyclesPerHour = df["count"]
rentedCyclesPerHour
```

```
0         16
1         40
2         32
3         13
4          1
        ...
10881    336
10882    241
10883    168
10884    129
10885     88
Name: count, Length: 10886, dtype: int64
```

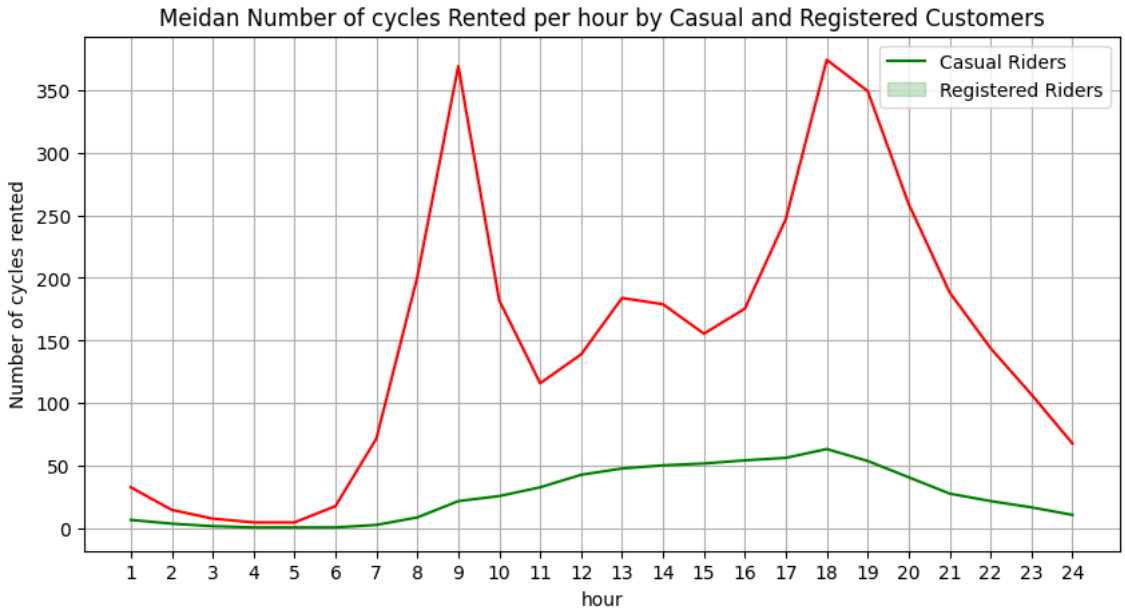
```
detect_outliers(rentedCyclesPerHour)
```

```
'0.0278 % Outliers data from input data found'
```

Number of cycles rented by : casual users and registered users

Average Number of Cycles rented by Casual vs Registered Subscribes :

```
registered_per_hour_median = df.groupby("hour")["registered"].median()
casual_per_hour_median = df.groupby("hour")["casual"].median()
registered_per_hour_median = registered_per_hour_median.reset_index()
casual_per_hour_median = casual_per_hour_median.reset_index()
casual_per_hour_median["hour"]+= 1
registered_per_hour_median["hour"]+= 1
median_count_perHr = registered_per_hour_median.merge(casual_per_hour_median,on="hour")
plt.figure(figsize=(10,5))
sns.lineplot(x = median_count_perHr["hour"],
  y = median_count_perHr["casual"],color="g",legend='auto')
sns.lineplot(x = median_count_perHr["hour"],
  y = median_count_perHr["registered"],color="r",legend='auto')
plt.legend(["Casual Riders","Registered Riders"])
plt.title("Meidan Number of cycles Rented per hour by Casual and Registered Customers")
plt.grid()
plt.xticks(np.arange(1,25,1))
plt.ylabel("Number of cycles rented")
plt.show()
```



Observations:

1. registered customers seems to be using rental cycles mostly for work-commute purposes.
2. registered cycle counts seems to be much higher than the casual customers.

```
print("Casual Users (in %) :")
(df["casual"].sum()/df["count"].sum())*100
```

```
Casual Users (in %) :
18.8031413451893
```

```
print("Registered Users (in %) : ")
(df["registered"].sum()/df["count"].sum())*100
```

```
Registered Users (in %) :
81.1968586548107
```

Observations:

1. 81% cycles had been rented by registered customers.
2. 19% cycles had been rented by casual customers.

Using Bootrsapping : Confidence Interval of Mean Number of cycles Rented by Casual And

Registered Customers :

```
def Confidence_Interval_Bootstrapping(df, confidence=95 , sample_size = 30000,trials = 200):
  '''
  data : array
  confidence level : Required Confidence Level
  Sample Size : length of Sample Size
  Trials : How many times we take sample sample from data.
  '''
  print("Data Distribution before Sampling/Bootstrap: Data Distribution After Sampling/Bootstraping")
  bootstrapped_mean= np.empty(trials)
  for i in range(trials):
    btssample = df.sample(n=sample_size,replace=True)
    bootstrapped_mean[i] = np.mean(btssample)
  print()
  sample_mean = np.mean(bootstrapped_mean)
  sample_std = np.std(df)
  standard_error = sample_std/np.sqrt(sample_size)
  talfa_by2 = t.ppf((1-((1-(confidence)/100)/2)),df = sample_size-1)
  margin_of_error = talfa_by2*standard_error
  print("sample mean :",sample_mean)
  print("sample standard deviation :",sample_std)
  print("sample size: ",sample_size)
  plt.figure(figsize=(16,5))
  plt.subplot(121)
  sns.distplot(df,bins = 15)
  plt.subplot(122)
  sns.distplot(bootstrapped_mean,bins = 15)
  lower_ = sample_mean - margin_of_error
  upper_ = sample_mean + margin_of_error
  CI = (lower_,upper_)
  plt.axvline(x = lower_,c = "r")
  plt.axvline(x = upper_,c = "r")
  plt.show()
  print("Confidence Interval : ",CI)
```

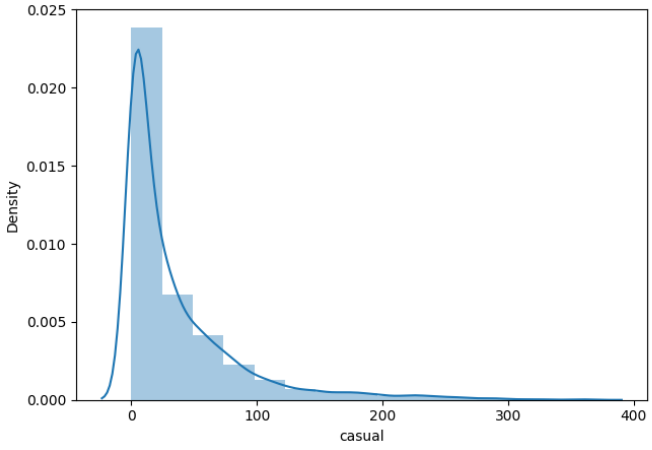
Confidence Interval of Average Number of Cycles Rented by Registered Customers

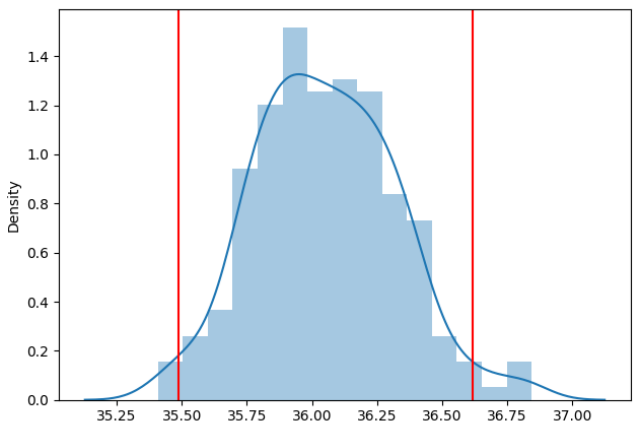
```
Confidence_Interval_Bootstrapping(df["registered"])
```

```
Data Distribution before Sampling/Bootstrap: Data Distribution After Sampling/Bootstraping

sample mean : 155.58436933333334
sample standard deviation : 151.03209561628546
sample size:  30000
<ipython-input-45-6c1c3af38187>:24: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(df,bins = 15)
<ipython-input-45-6c1c3af38187>:26: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
```

Confidence Interval of Average Number of Cycles Rented by Casual Customers

```
Confidence_Interval_Bootstrapping(df["casual"])

Data Distribution before Sampling/Bootstrap: Data Distribution After Sampling/Bootstraping

sample mean : 36.052960000000006
sample standard deviation : 49.958181807631085
sample size:  30000
<ipython-input-45-6c1c3af38187>:24: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(df,bins = 15)
<ipython-input-45-6c1c3af38187>:26: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(bootstrapped_mean,bins = 15)
```



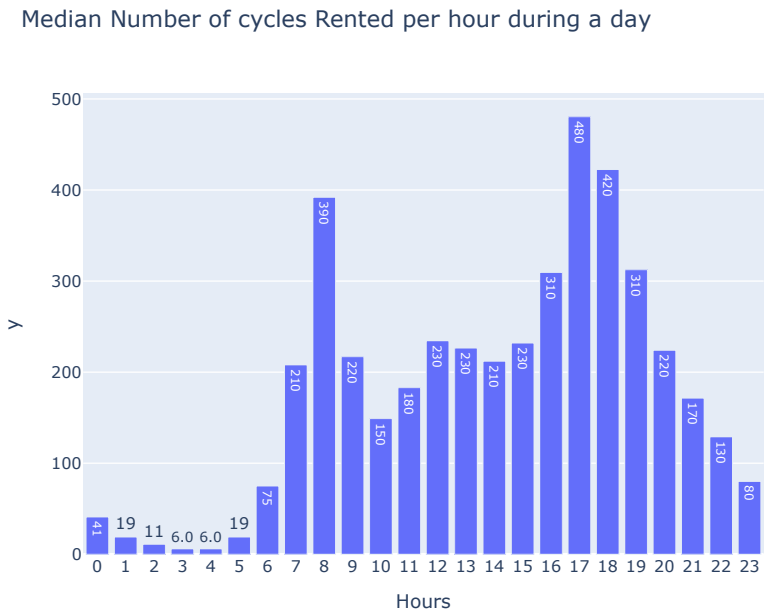



```
Confidence Interval :  (35.48761753189735, 36.61830246810266)
```

Hourly median number of cycles rented during the day :

```
fig = px.bar(y = df.groupby("hour")["count"].median(),
             x = df.groupby("hour")["count"].median().index,text_auto='.2s',
             labels={
```

```
                "x": "Hours",
                "Y": "Median value of Number of cycles rented", }
              ,title="Median Number of cycles Rented per hour during a day"
)

fig.update_layout(
xaxis = dict(
tickmode = 'linear',
tick0 = 0,
dtick = 1
)
)
fig.show()
```



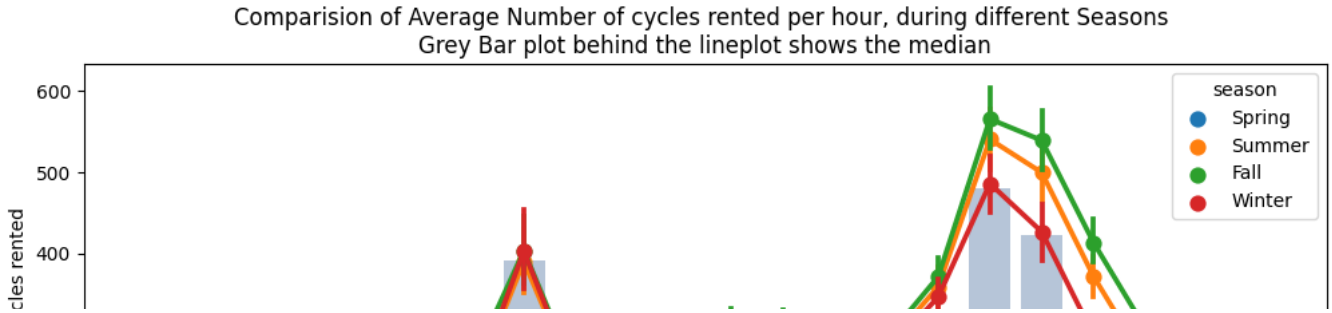
Observations:

1. from above bar chart : shows the median value of number of cycles were rented during perticular hour of the day.
2. Median of number of cycles rented are higher during morning 7 to 9 am to evening 4 to 8pm .

Effect of seasons on number of cycles rented during hours :

```
plt.figure(figsize=(12,5))
sns.barplot(y = df.groupby("hour")["count"].median(),
x = df.groupby("hour")["count"].median().index,
color="lightsteelblue")
sns.pointplot(x = df["hour"],
y= df["count"],
hue=df["season"],
ci=95)
plt.title("Comparision of Average Number of cycles rented per hour, during different Seasons \nGrey Bar plot behind the lineplot shows th
plt.xticks(rotation = 90)
plt.ylabel("Number of cycles rented")
plt.show()
```

```
<ipython-input-50-e06b53c4a59e>:5: FutureWarning:

The `ci` parameter is deprecated. Use `errorbar=('ci', 95)` for the same effect.
```



Observations:

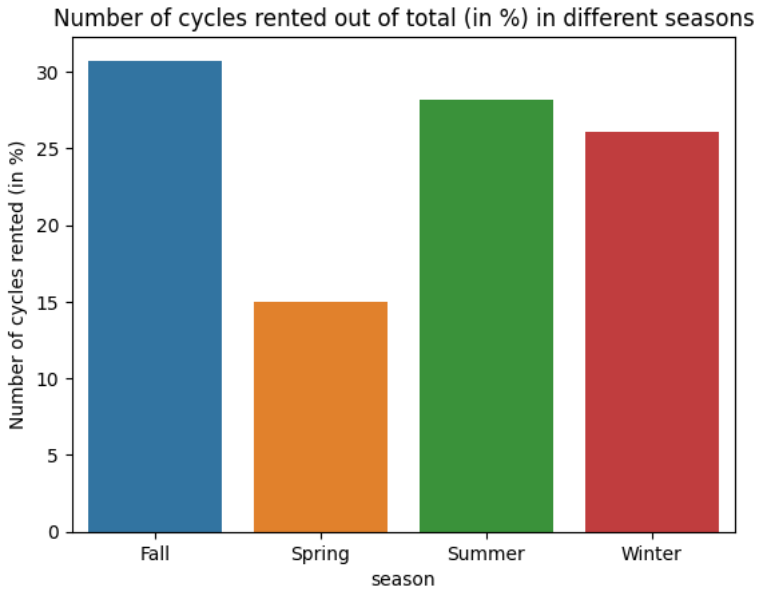
1. during the morning 7-9am and afternoon 4pm to 7pm , the cycles rent counts is increasing.
2. during the spring season , looks like people prefer less likely to rent the cycle.

Number of cycles rented during differnet seasons (in %) :`


```
season_wise_rent_percentage = df.groupby("season")["count"].sum()/np.sum(df["count"])*100
season_wise_rent_percentage
```

```
season
Fall      30.720181
Spring    14.984493
Summer    28.208524
Winter    26.086802
Name: count, dtype: float64
```

```
sns.barplot(x= season_wise_rent_percentage.index,
y = season_wise_rent_percentage)
plt.ylabel("Number of cycles rented (in %)")
plt.title("Number of cycles rented out of total (in %) in different seasons")
plt.show()
```

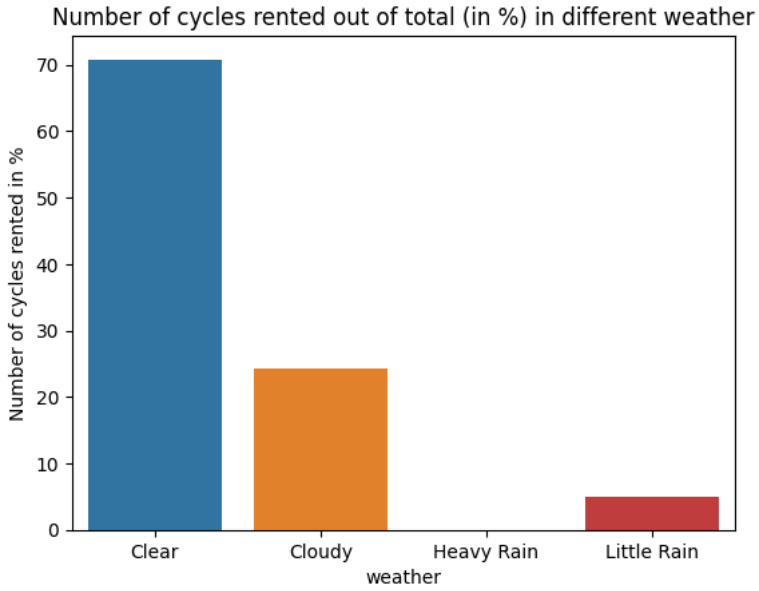


weather effect on cycle rental median counts hourly :

```
weather_wise_rent_percentage = df.groupby("weather")["count"].sum()/np.sum(df["count"])*100
weather_wise_rent_percentage
```

```
weather
Clear          70.778230
Cloudy         24.318669
Heavy Rain      0.007864
Little Rain     4.895237
Name: count, dtype: float64
```

```
sns.barplot(x= weather_wise_rent_percentage.index,
y = weather_wise_rent_percentage)
plt.title("Number of cycles rented out of total (in %) in different weather")
plt.ylabel("Number of cycles rented in %")
plt.show()
```




```
plt.figure(figsize=(12,5))
sns.barplot(y = df.groupby("hour")["count"].median(),
x = df.groupby("hour")["count"].median().index,
color="lightsteelblue")
sns.pointplot(x = df["hour"],
y= df["count"],
hue=df["weather"],
ci=95)
plt.title("Comparision of Average Number of cycles rented per hour, during different weather type\nGrey Bar plot behind the lineplot show
plt.xticks(rotation = 90)
plt.ylabel("Average Number of cycles rented")
plt.show()
```

```
<ipython-input-56-9aa9a807cfa4>:5: FutureWarning:

The `ci` parameter is deprecated. Use `errorbar=('ci', 95)` for the same effect.
```

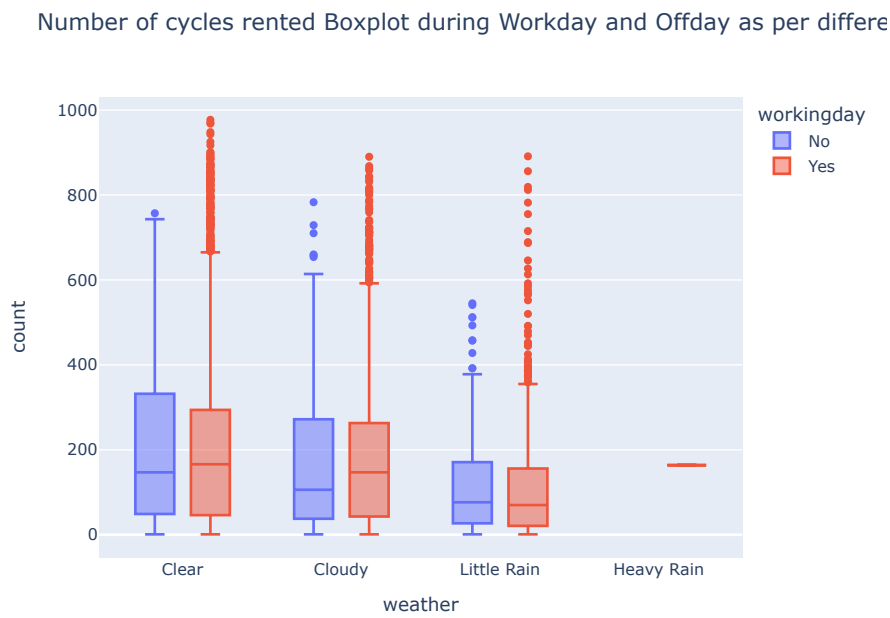
Observations:

1. 70% of the cycles were rented when it was clear weather.
2. 24% when it was cloudy weather .
3. during rainy weather , only around 5% of the cycles were rented.

DISTRIBUTIONS and Comparision of number of cycles rented during working days and off day , across different

Boxplot - distribution of number of bike rented , during different weather as per workingday or not!

```
fig = px.box(df, x="weather", y="count", color="workingday",
title="Number of cycles rented Boxplot during Workday and Offday as per different weather conditions")
fig.show()
```

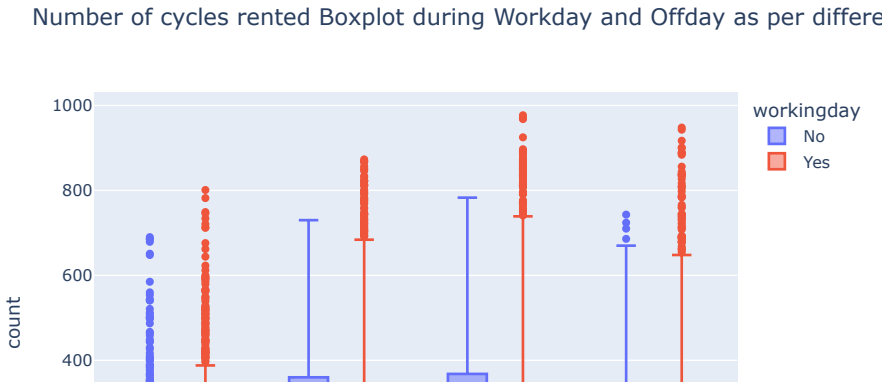


Observations:

1. from above boxplot, we can say , there's no significant activity during heavy rain weather.
2. High activity during clear and cloudy weather.

Boxplot - distribution of number of bike rented , during different seasons as per workingday or not!

```
fig = px.box(df, x="season", y="count", color="workingday",
title="Number of cycles rented Boxplot during Workday and Offday as per different seasons")
fig.show()
```

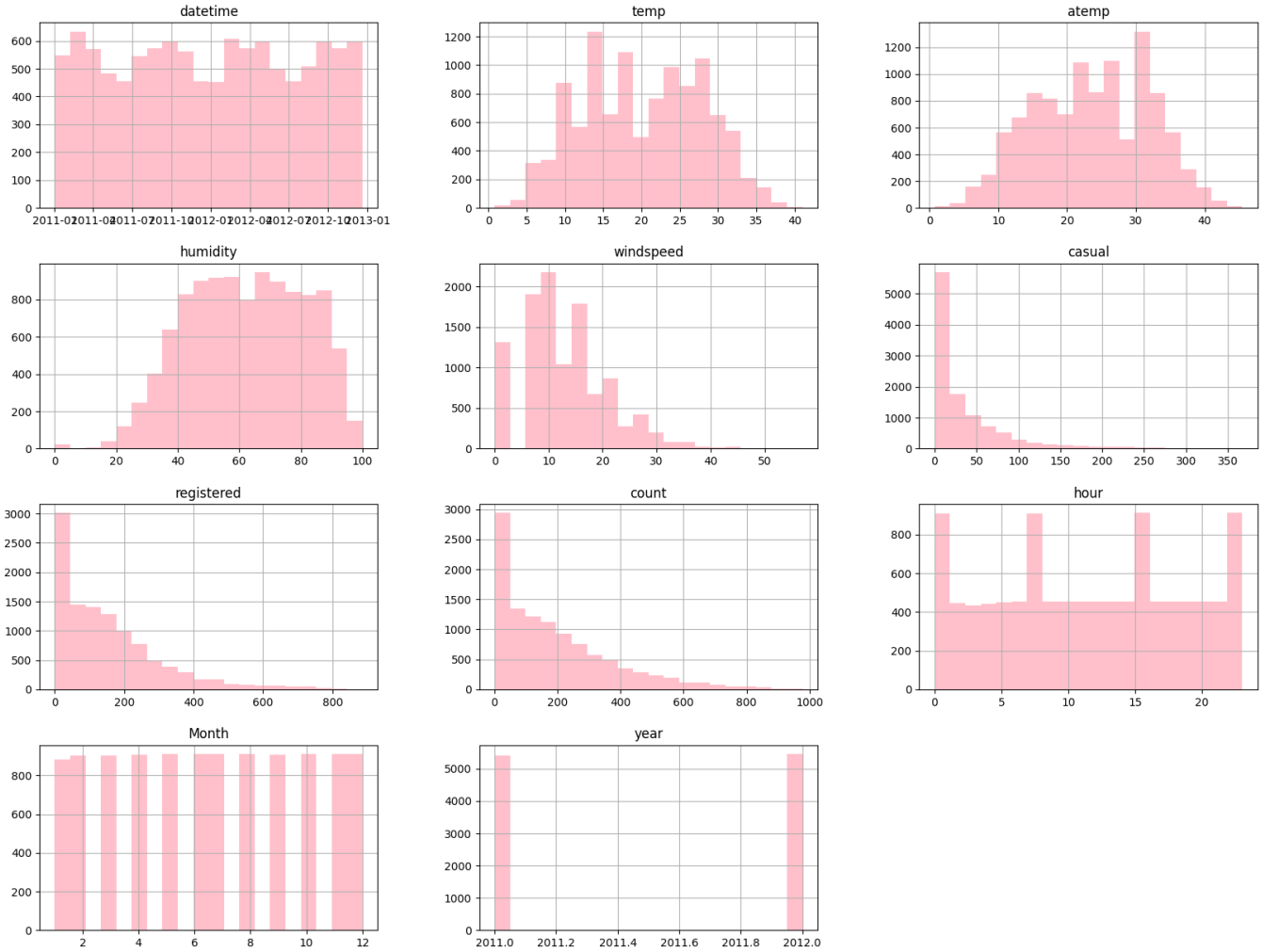


Observation:

1. during spring season , number of bike rented were lower than summer and fall.

overview on distributions of Numerical Features :

```
df.hist(bins=20,figsize=(20,15),color='pink')
plt.show()
```



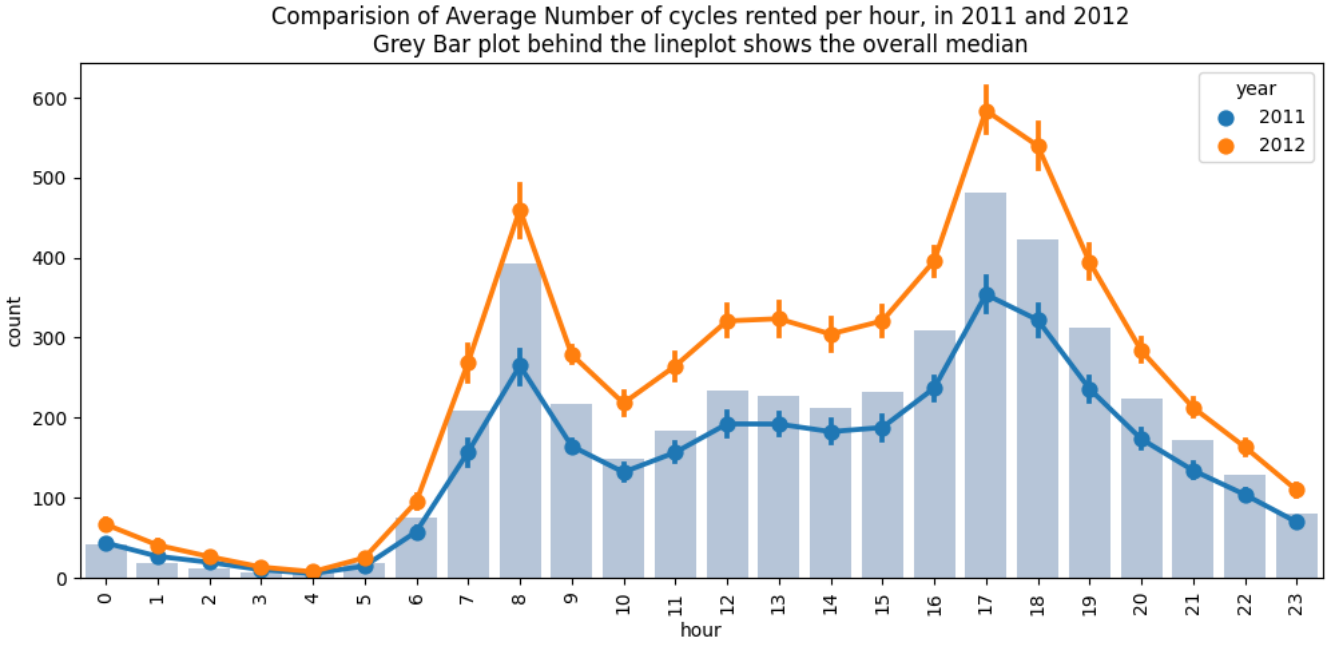
Observations from above distribution plots of number of bikes rented , are not normally distributed.

1. also that there are outliers in the data and overall distributions are heavily right skewed .
2. data need to be tranformed for hypothesis test calculations further.

Yearly difference in number of bike rental :

```
plt.figure(figsize=(12,5))
sns.barplot(y = df.groupby("hour")["count"].median(),
x = df.groupby("hour")["count"].median().index,
color="lightsteelblue")
sns.pointplot(x = df["hour"],
y= df["count"],
hue=df["year"],
ci=95)
plt.title("Comparision of Average Number of cycles rented per hour, in 2011 and 2012\nGrey Bar plot behind the lineplot shows the overall
plt.xticks(rotation = 90)
plt.show()
```

```
<ipython-input-61-761f13910469>:5: FutureWarning:

The `ci` parameter is deprecated. Use `errorbar=('ci', 95)` for the same effect.
```



hourly average bike rented in year 2011 and 2012:

```
df.groupby("year")["count"].median()
```

```
year
2011    111.0
2012    199.0
Name: count, dtype: float64
```

```
(((199-111)/111))*100
```

```
79.27927927927928
```

Observation:

1. from 2011 , there's 79.27% hike in hourly median number of bike rental.

```
df.groupby("year")["casual"].median()
```

```
year
2011    13.0
2012    20.0
Name: casual, dtype: float64
```

```
df.groupby("year")["registered"].median()
```

```
year
2011     91.0
2012    161.0
Name: registered, dtype: float64
```

```
(((161-91)/91))*100
```

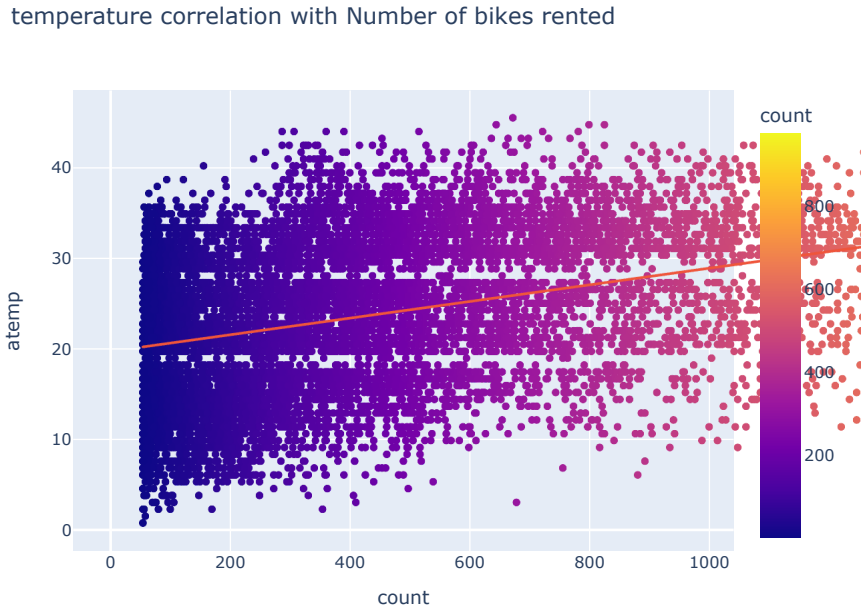
```
76.92307692307693
```

Observations:

1. in registered customers , 76% hike in hourly median cycle rental from 2011 to 2012.
2. in 2011 , median number of hourly rental were 13 , and in 2012 , its 20.

Number and cycles rented and temperature correlation :

```
fig = px.scatter(df, x="count", y="atemp", color="count", trendline="ols",
title="temperature correlation with Number of bikes rented")
fig.show()
```



Observation:

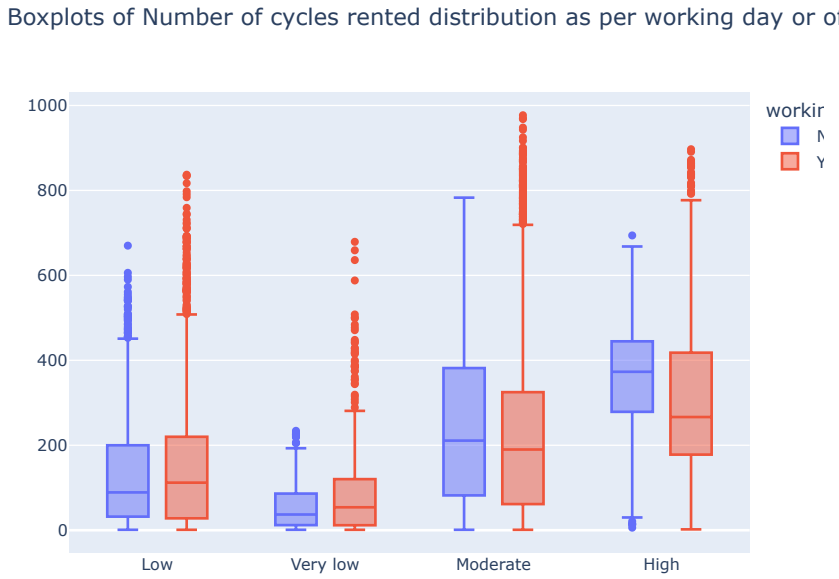
1. from scatter plot , there's a positive correlation across temperature and number of bikes rented.

After categorising the temperature as low, verylow, moderate, high :

```
df["temperature"].value_counts()
```

```
Moderate    4767
Low         4318
Very low    1014
High         787
Name: temperature, dtype: int64
```

```
fig = px.box(df, x="temperature", y="count", color="workingday",
title= "Boxplots of Number of cycles rented distribution as per working day or offday in different temperatures")
fig.show()
```



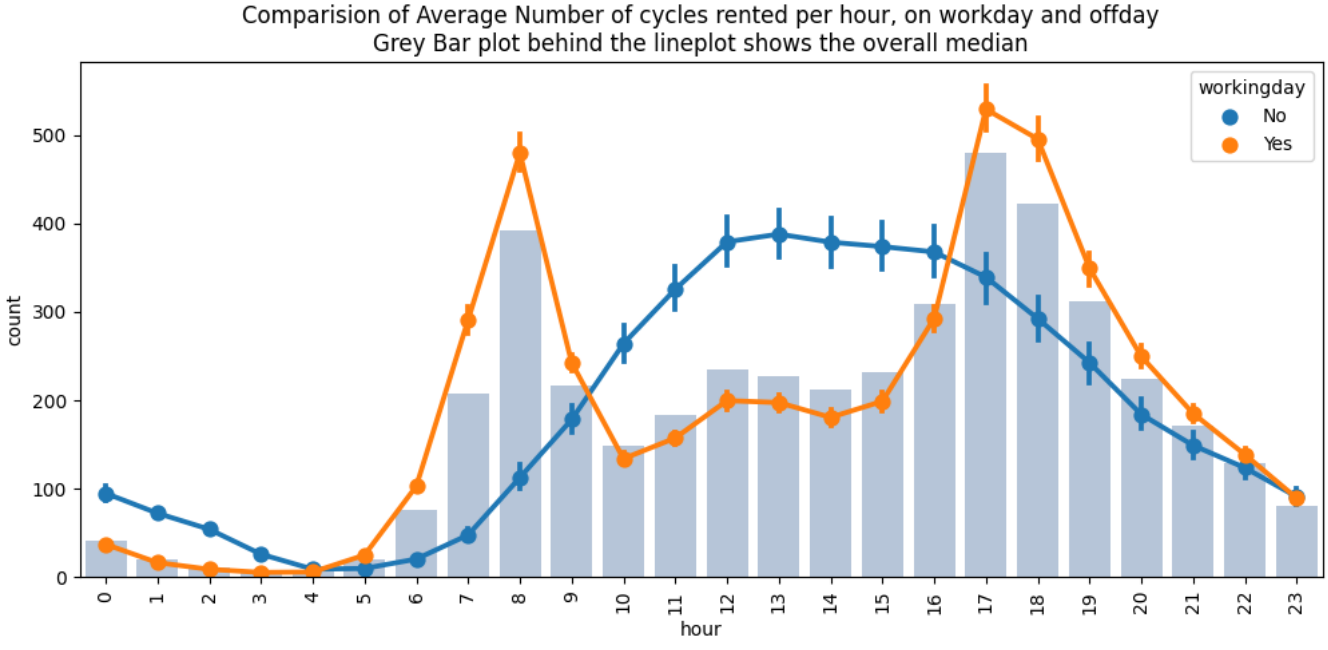
Observations: from above boxplot :

1. number of bike rented during moderate to high temerature is significantly higher than lower temperature.

offday vs working day number of cycles rented trend during a day :

```
plt.figure(figsize=(12,5))
sns.barplot(y = df.groupby("hour")["count"].median(),
x = df.groupby("hour")["count"].median().index,
color="lightsteelblue")
sns.pointplot(x = df["hour"],
y= df["count"],
hue=df["workingday"],
ci=95)
plt.title("Comparision of Average Number of cycles rented per hour, on workday and offday\nGrey Bar plot behind the lineplot shows the ov
plt.xticks(rotation = 90)
plt.show()
```

```
<ipython-input-71-dfe9c657d9aa>:5: FutureWarning:


The `ci` parameter is deprecated. Use `errorbar=('ci', 95)` for the same effect.
```



Observations:

1. number of cycles rented changed as per working day and off-day . trend is opposit.
2. on off days , number of cycles rented increases during the day time ! which is opposite of during working days.
3. from above plot it looks like, working day count of cycle rented seems to be higher than offday! lets do a AB test : weather mean of rented cycled on working day and offdays are same or not !

hourly median number of cycles rented during

```
df.groupby("workingday")["count"].median()
```

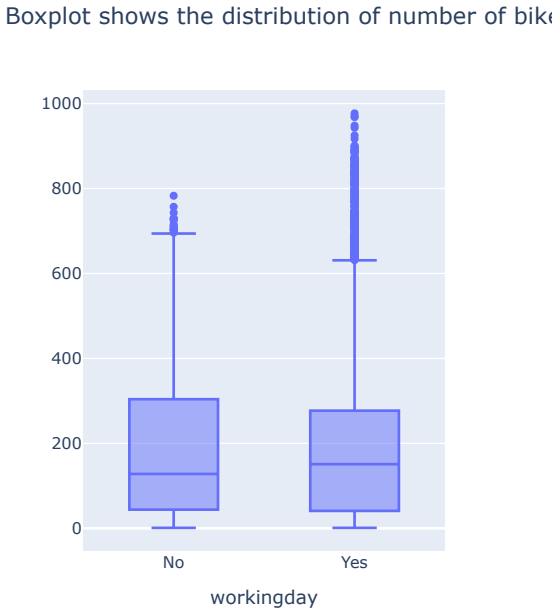
```
workingday
No     128.0
Yes    151.0
Name: count, dtype: float64
```

```
df.groupby("workingday")["count"].mean()
```

```
workingday
No     188.506621
Yes    193.011873
Name: count, dtype: float64
```

Boxplot : number of bikes rented during working day and off-day :

```
fig = px.box(df, x="workingday", y="count",
title="Boxplot shows the distribution of number of bikes rented on offdays and workingdays")
fig.show()
```



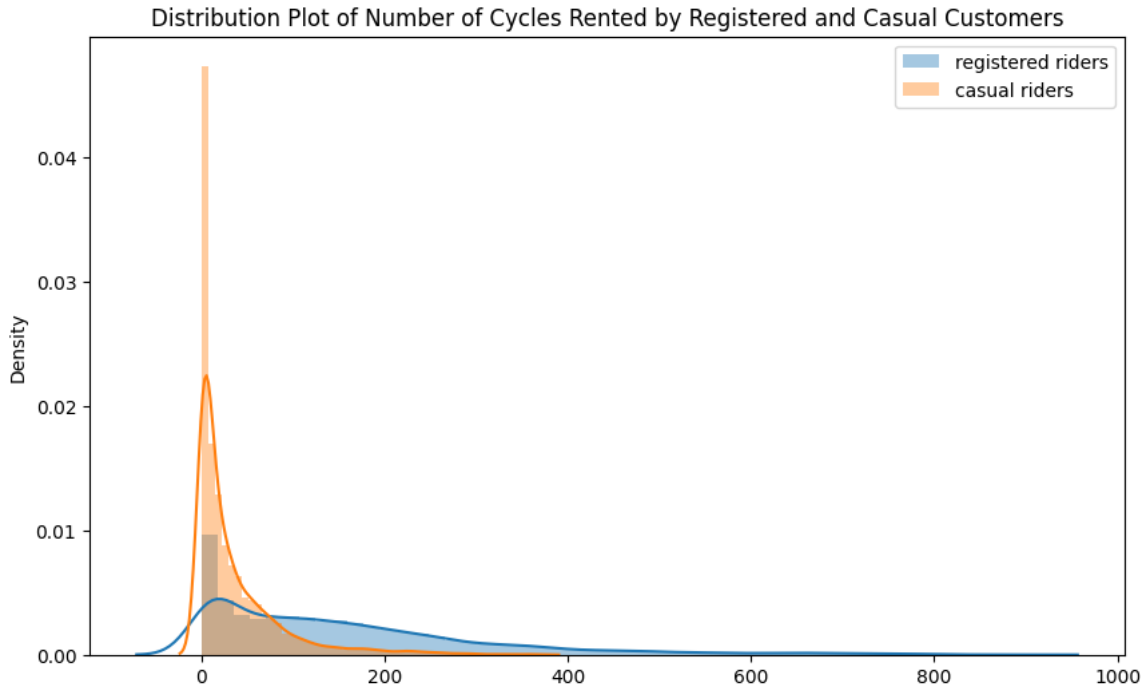
Observations:

1. from above boxplot, distributions of hourly number of bike rented during working day and off day seems similar .
2. though there are more outliers in workinday category.

Distribution Plot of Number of Cycles Rented by Registered and Casual Customers

```
plt.figure(figsize=(10,6))
sns.distplot(df["registered"], label = "registered riders")
sns.distplot(df["casual"], label = "casual riders")
plt.title("Distribution Plot of Number of Cycles Rented by Registered and Casual Customers")
plt.legend()
plt.show()
```

```
<ipython-input-75-646a37852dc4>:2: UserWarning:


`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751


<ipython-input-75-646a37852dc4>:3: UserWarning:


`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751
```



Step 2: Hypothesis Testing

Testing if mean number of electric cycles rented on workday is equal to on offday !

If working day and offday has an effect on the number of electric cycles rented.

distribution of number of bikes rented as per working day or offday (in percentages

```
df.groupby("workingday")["count"].sum()/np.sum(df["count"])*100
```

```
workingday
No     31.40156
Yes    68.59844
Name: count, dtype: float64
```

```
workingday = df.loc[df["workingday"]=="Yes"]["count"]
offday = df.loc[df["workingday"]=="No"]["count"]
```

Establishing Hypothesis :

- H0: average # of cycles rented on workingdays = average # of cycles rented on offday
- Ha: average # of cycles rented on workingdays != average # of cycles rented on offday

calulating Test Statistic :

```
ttest_ind(workingday, offday, alternative = "two-sided")
```

```
Ttest_indResult(statistic=1.2096277376026694, pvalue=0.22644804226361348)
```

As the significance("alpha") is 5%

p value > 0.05

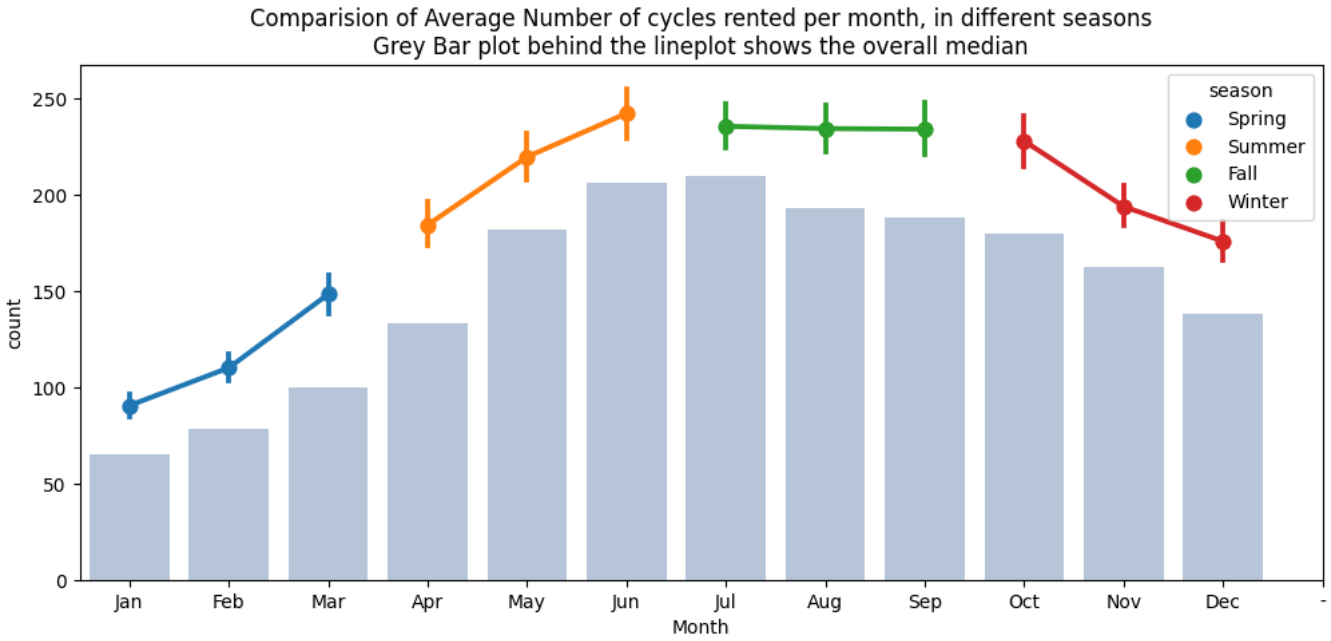
Observations:

1. we failed to reject null Hypothesis
2. mean of number of cycles rented on working days are equal as the cycles rented on offdays.

Month and season wise , effect on median and average number of cycles rented:

```
plt.figure(figsize=(12,5))
sns.barplot(y = df.groupby("Month")["count"].median(),
x = df.groupby("Month")["count"].median().index,
color="lightsteelblue")
sns.pointplot(x = df["Month"],
y= df["count"],
hue=df["season"],
ci=95)
plt.title("Comparision of Average Number of cycles rented per month, in different seasons\nGrey Bar plot behind the lineplot shows the ov
plt.xticks([0,1,2,3,4,5,6,7,8,9,10,11,12],["Jan","Feb","Mar","Apr","May","Jun","Jul","Aug","Sep","Oct","Nov","Dec","-"])
plt.show()
```

```
<ipython-input-81-ebb41f701576>:5: FutureWarning:


The `ci` parameter is deprecated. Use `errorbar=('ci', 95)` for the same effect.
```



Observations:

1. cycle rental counts decreased during winter season and opering spring seaosn .
2. During Summer season , count increase and stays a constant till pre-winter season .
3. From May to November the number of cycles rented are increasing

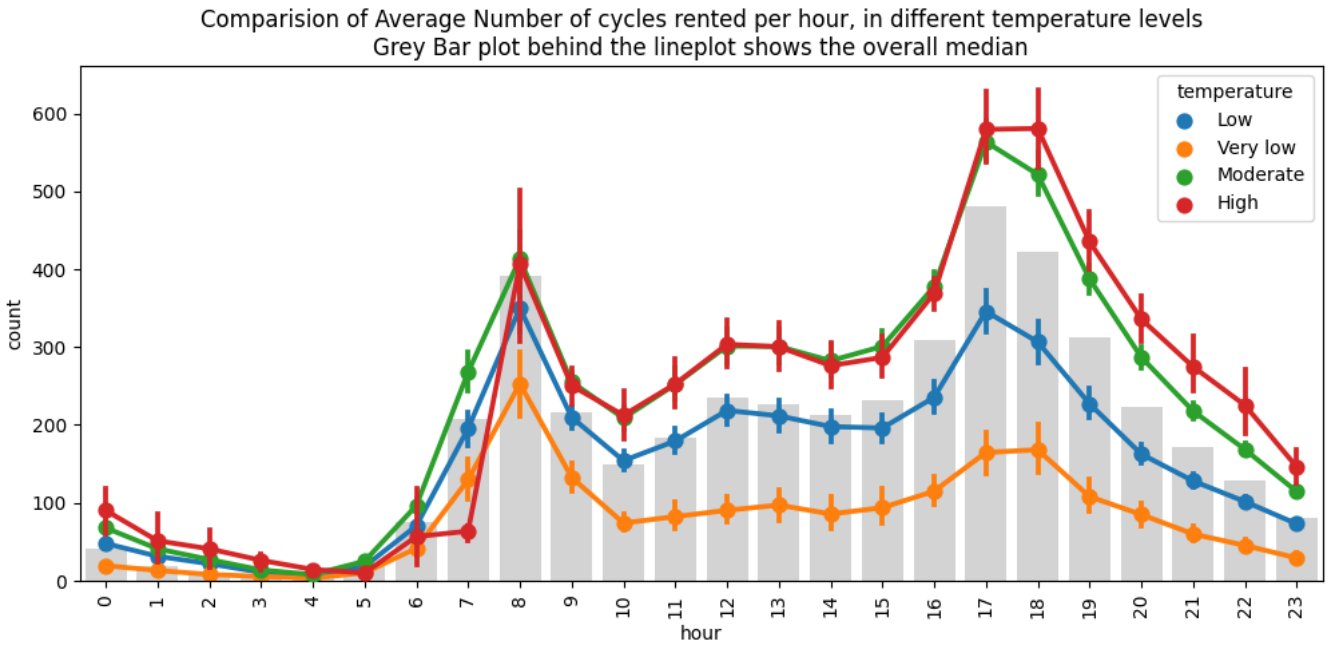
Temperature effect on cycle rental

```
temperature_wise_rent_percentage = df.groupby("temperature")["count"].sum()/np.sum(df["count"])*100
temperature_wise_rent_percentage
```

```
temperature
High        12.487269
Low         30.172248
Moderate    53.538617
Very low     3.801866
Name: count, dtype: float64
```

```
plt.figure(figsize=(12,5))
sns.barplot(y = df.groupby("hour")["count"].median(),
x = df.groupby("hour")["count"].median().index,
color="lightgrey")
sns.pointplot(x = df["hour"],
y= df["count"],
hue=df["temperature"],
ci=95)
plt.title("Comparision of Average Number of cycles rented per hour, in different temperature levels\nGrey Bar plot behind the lineplot sh
plt.xticks(rotation = 90)
plt.show()
```

```
<ipython-input-83-22ec9719d85c>:5: FutureWarning:


The `ci` parameter is deprecated. Use `errorbar=('ci', 95)` for the same effect.
```

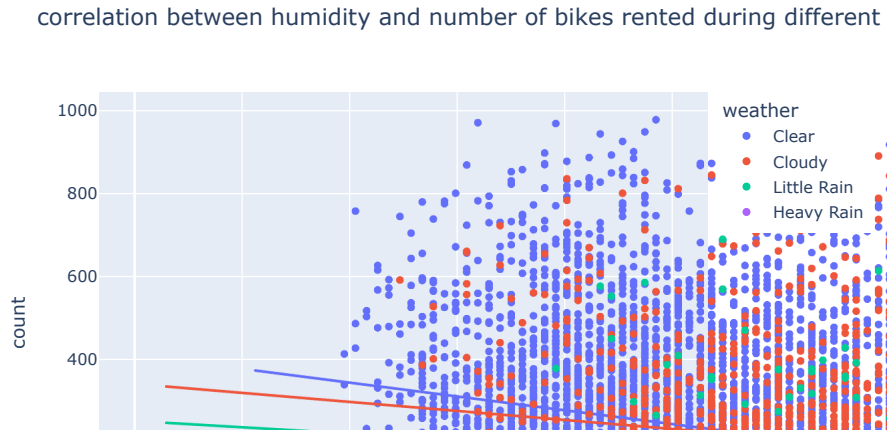


Observations:

1. Average Number of Bikes rented are higher in moderate to high temperature, which decreases when temperature is low to very low

humidity vs count

```
fig = px.scatter(df, y="count", x="humidity", color="weather", trendline="ols",
title=" correlation between humidity and number of bikes rented during different weather")
fig.show()
```



Observation:

1. Scatter plot above , shows kind of a negative correlation , between humidity and number of bikes rented.

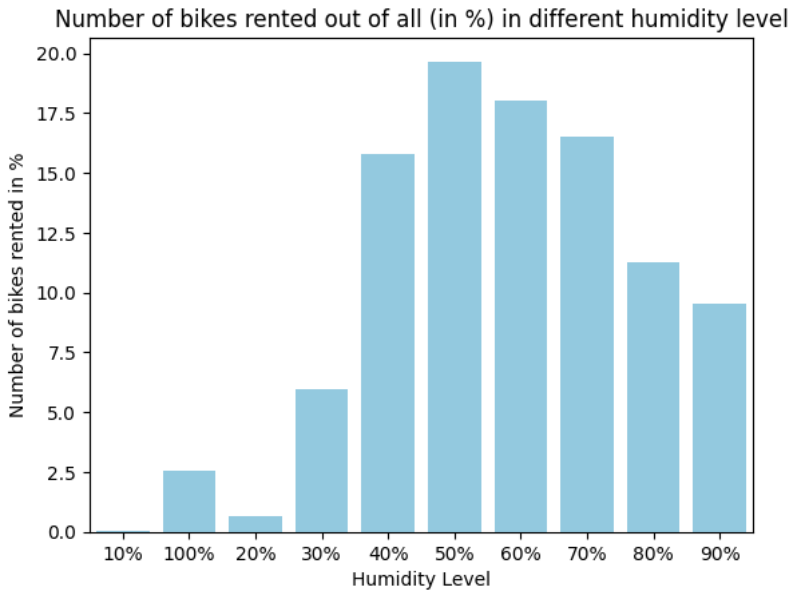
```
humidity_wise_rent_percentage = df.groupby("gethumidity")["count"].sum()/np.sum(df["count"])*100
humidity_wise_rent_percentage
```

```
gethumidity
10%      0.038696
100%     2.565314
20%      0.635970
30%      5.942528
40%     15.798887
50%     19.659541
60%     18.030512
70%     16.507215
80%     11.268459
90%      9.552879
Name: count, dtype: float64
```

Observations:

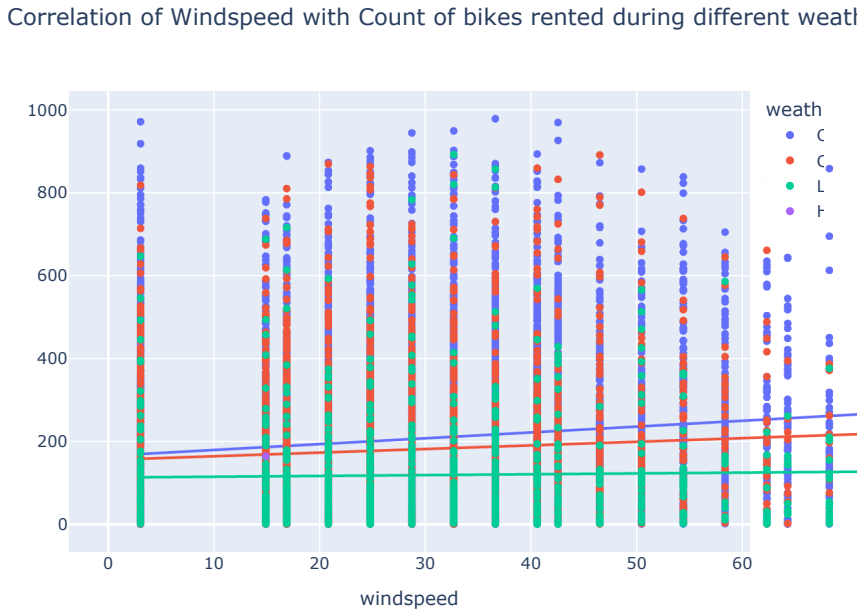
1. Counts are increasing from humidity level of 40% to 70% .
2. 40 to 70% humidity level seems to be most comfortable for cycling.

```
sns.barplot(x= humidity_wise_rent_percentage.index,
y = humidity_wise_rent_percentage,color="skyblue")
plt.title("Number of bikes rented out of all (in %) in different humidity level")
plt.ylabel("Number of bikes rented in %")
plt.xlabel("Humidity Level")
plt.show()
```



Windspeed vs count :

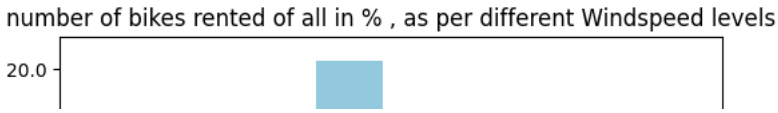
```
fig = px.scatter(df, y="count", x="windspeed", color="weather", trendline="ols",
title= "Correlation of Windspeed with Count of bikes rented during different weather")
fig.show()
```




```
windspeed_wise_rent_percentage = df.groupby("windspeed_category")["count"].sum()/np.sum(df["count"])*100
windspeed_wise_rent_percentage
```

```
windspeed_category
(-0.001, 6.003]     16.325482
(6.003, 7.002]       8.421435
(7.002, 8.998]       9.433002
(8.998, 12.998]     20.356743
(12.998, 15.001]     9.715336
(15.001, 16.998]     8.488901
(16.998, 22.003]    15.682703
(22.003, 56.997]    11.576398
Name: count, dtype: float64
```

```
sns.barplot(x= windspeed_wise_rent_percentage.index,
y = windspeed_wise_rent_percentage,color="skyblue")
plt.title("number of bikes rented of all in % , as per different Windspeed levels")
plt.ylabel("Number of bikes rented of all in %")
plt.xticks(rotation =90)
plt.show()
```



Observations:from above, plot:

1. windspeed are categorised in different groups .
2. Windspeed increases , the number of bike rented are decreases.
3. Most often windspeed is 8 to 24.

Test for Independence between few categorical features:

If Weather is dependent on the season

chi-square test : for independence : weather and season are categorical variables for dependency : chi square test :

H0: weather and seasons are independent

Ha: weather and seasons are dependent

```
temp_data = df[df["weather"].isin(["Little Rain","Clear","Cloudy"])]
```

```
observed = pd.crosstab(index = temp_data["season"],
columns = temp_data["weather"],
values= temp_data["count"],
aggfunc=np.sum
)
```

```
observed
```

season	Clear	Cloudy	Little Rain
Fall	470116	139386	31160
Spring	223009	76406	12919
Summer	426350	134177	27755
Winter	356588	157191	30255

```
pd.crosstab(index = temp_data["season"],
columns = temp_data["weather"],
values= temp_data["count"],
aggfunc=np.sum,
margins=True )
```

season	Clear	Cloudy	Little Rain	All
Fall	470116	139386	31160	640662
Spring	223009	76406	12919	312334
Summer	426350	134177	27755	588282
Winter	356588	157191	30255	544034
All	1476063	507160	102089	2085312

```
row_sum = np.array(np.sum(observed,axis = 1))
col_sum = np.array(np.sum(observed,axis = 0))
expected = []
for i in row_sum:
  expected.append((i*col_sum)/np.sum(np.sum(observed,axis = 0)))
expected
```

```
[array([453484.88557396, 155812.72247031,  31364.39195574]),
 array([221081.86259035,  75961.44434981,  15290.69305984]),
```

```
       array([416408.3330293 , 143073.60199337,  28800.06497733]),
       array([385087.91880639, 132312.23118651,  26633.8500071 ])]
```

```
expected = pd.DataFrame(expected,columns=observed.columns)
expected
```

weather	Clear	Cloudy	Little Rain
0	453484.885574	155812.722470	31364.391956
1	221081.862590	75961.444350	15290.693060
2	416408.333029	143073.601993	28800.064977
3	385087.918806	132312.231187	26633.850007

```
chi2_contingency(observed)
```

```
Chi2ContingencyResult(statistic=10838.372332480214, pvalue=0.0, dof=6, expected_freq=array([[453484.88557396, 155812.72247031,
31364.39195574],
       [221081.86259035,  75961.44434981,  15290.69305984],
       [416408.3330293 , 143073.60199337,  28800.06497733],
       [385087.91880639, 132312.23118651,  26633.8500071 ]]))
```

Observations: As p value < alpha(0.05)

Reject Null Hypothesis : Weather and Season are dependent variables From ChiSquare test of independece :

We reject Null hyothesis as independence:

Conclusion that weather and seasons are Dependent Features.

If weather and temperature are dependent :

for dependency : chi square test :

H0: weather and temperature are independent

Ha: weather and temperature are dependent

```
observed_temp_weather = pd.crosstab(index=temp_data["weather"],
columns= temp_data["temperature"],
values=temp_data["casual"],
aggfunc=np.sum)
chi2_contingency(observed_temp_weather)
```

```
Chi2ContingencyResult(statistic=2979.8035003021923, pvalue=0.0, dof=6, expected_freq=array([[4.86162054e+04, 6.12071816e+04,
1.76870280e+05, 3.20633337e+03],
       [1.46311468e+04, 1.84204269e+04, 5.32294737e+04, 9.64952610e+02],
       [2.51264783e+03, 3.16339152e+03, 9.14124664e+03, 1.65714015e+02]]))
```

Observations: As p value < alpha(0.05)

Reject Null Hypothesis : Weather and Season are dependent variables From ChiSquare test of independece :

We reject Null hyothesis as independence:

Conclusion that weather and temperature are Dependent Features.

If Weather and Humidity Level are dependent :

for dependency : chi square test :

H0: weather and Humidity are independent

Ha: weather and Humidity are dependent

```
weathervshumidity = pd.crosstab(index=temp_data["weather"],
columns= temp_data["gethumidity"])
weathervshumidity
```

gethumidity	10%	100%	20%	30%	40%	50%	60%	70%	80%	90%
weather										
Clear	2	111	52	382	1031	1400	1327	1248	938	701
Cloudy	2	289	2	32	110	331	447	527	475	619
Little Rain	20	282	0	0	5	19	40	70	124	299

```
chi2_contingency(weathervshumidity)
```

```
Chi2ContingencyResult(statistic=2722.2373985014465, pvalue=0.0, dof=18, expected_freq=array([[  15.85741847,  450.61497474,
35.67919155,  273.54046853,
        757.19173174, 1156.27009646, 1198.55654571, 1219.03904456,
       1015.53550758, 1069.71502067],
       [   6.24859899,  177.56435462,   14.05934773,  107.78833257,
        298.37060175,  455.62700965,  472.28994028,  480.36104731,
        400.17069362,  421.5200735 ],
       [   1.89398254,   53.82067065,    4.26146073,   32.6711989 ,
         90.43766651,  138.10289389,  143.15351401,  145.59990813,
        121.29379881,  127.76490583]]))
```

Observations: As p value < alpha(0.05)

Reject Null Hypothesis : Weather and humidity are dependent variables From

ChiSquare test of independece :

We reject Null hyothesis as independence:

Conclusion that weather and humidity are Dependent Features.

checking if the distribution of number of cycles rented are similar in different weather.

If Average No. of cycles rented is similar or different in different weather

```
df["weather"].unique()
```

```
array(['Clear', 'Cloudy', 'Little Rain', 'Heavy Rain'], dtype=object)
```

```
plt.figure(figsize=(12,5))
sns.pointplot(x = df["hour"],
y= df["count"],
hue=df["weather"],
ci=95)
plt.title("Comparision of Average Number of cycles rented per hour, during different weather type")
plt.show()
```

<ipython-input-102-3151a7b82505>:2: FutureWarning:

we have 4 different weather here, to check if there's significant differnece between 4 weathers , we can perform

"anova test" :

H0: μ1 = μ2 = μ3 = μ3. The mean Count in every season is same.

Ha: Atleast one of mean of count is not same

Anova and all parametric tests assume-

Normality:- Values in each sampled groups are assumed to be drawn from normally distributed populations. We can use normal probability plot or Q-Q plot to check normality.

Homogeneity of variance:- All the c group variances are equal, that is $\sigma_1^2 = \sigma_2^2 = \sigma_3^2 = \ldots = \sigma c^2$.

Assumptions Test for Anova

Anova assumes normality and also it assumes variances is same across all groups.

Normality test

Shapiro-Wilk's test

We will test the

null hypothesis: count follows normal distribution

against the

alternative hypothesis: count doesn't follow normal distribution

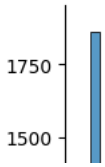
```
from scipy.stats import shapiro
# find the p-value
w, p_value = shapiro(df['count'])
print('The p-value is', p_value)
```

```
The p-value is 0.0
/usr/local/lib/python3.10/dist-packages/scipy/stats/_morestats.py:1816: UserWarning:

p-value may not be accurate for N > 5000.
```

As mentioned on documentation of scipy, when no of data points>5000, this test fails. let's move to another test

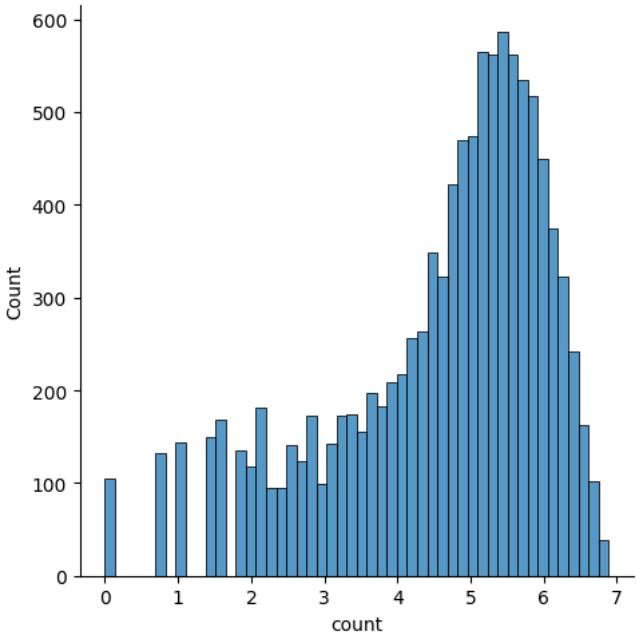
```
#Let's do normality test using distplot.
sns.displot(df['count'],bins=50)
```

```
<seaborn.axisgrid.FacetGrid at 0x7d2e8e9d16f0>
```




```
log_transform = np.log(df['count'])
sns.displot(log_transform,bins=50)
```

```
<seaborn.axisgrid.FacetGrid at 0x7d2e971db5b0>
```

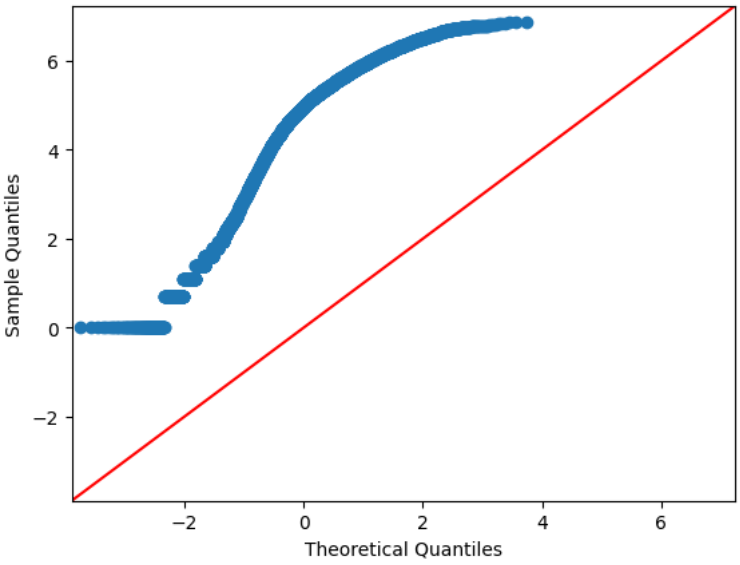


Observation: It is visible from distplot that count is not normally distributed but log_count is normally distributed

Another Test for normality

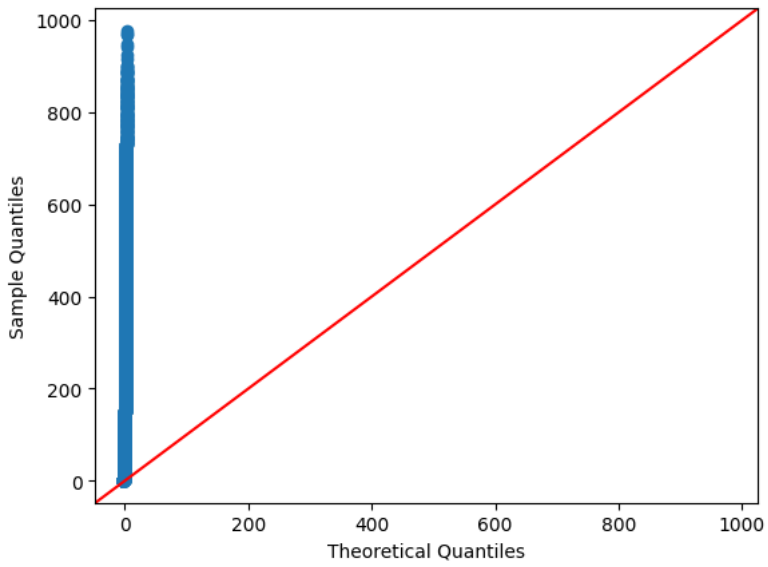
QQ-PLOT

```
import numpy as np
import statsmodels.api as sm
import pylab as py
sm.qqplot(log_transform, line ='45')
py.show()
```




```
import numpy as np
import statsmodels.api as sm
```

```
import pylab as py
sm.qqplot(df['count'], line ='45')
py.show()
```



Observations: None of the two Variables are Normally distributed as per QQ-plot. But still log_tranform is better. So we Will continue our analysis with log_transform as we have huge data.

Test for Equality of Variances

Levene's test

We will test the

null hypothesis : All the count variances are equal

against the

alternative hypothesis : At least one variance is different from the rest.

```
df['log_count']=np.log(df['count'])
```

```
df.groupby('season')['log_count'].describe()
```

season	count	mean	std	min	25%	50%	75%	max
Fall	2733.0	4.860311	1.378662	0.0	4.219508	5.273000	5.849325	6.884487
Spring	2686.0	3.984206	1.539737	0.0	3.178054	4.356709	5.099866	6.685861
Summer	2733.0	4.703267	1.462172	0.0	3.891820	5.147494	5.771441	6.771936
Winter	2734.0	4.652650	1.421134	0.0	3.931826	5.081404	5.683580	6.854355

Here we see that std across all groups are almost same. so it should not have any Variances. Let's check statistically

```
from scipy.stats import levene
statistic, p_value = levene(
df[df['season']=="Fall"]['log_count'],
df[df['season']=="Spring"]['log_count'],
df[df['season']=="Summer"]['log_count'],
df[df['season']=="Winter"]['log_count']
)
# find the p-value
print('The p-value is ',p_value)
```

```
The p-value is  2.3678125658230693e-06
```

Observations:

1. As p value< 0.05, Reject Ho and Variances are different. FAILED TO TEST WITH ANNOVA

2. Since the datasets for tests, are not normally distributed, and having significance varinace between weathers, we cannot perform anova test.

Inferences and Recommendations :

1. There is a positive Correlation between Temperature and Number of cycles rented.
2. Demand increases with the rise in the temperature from moderate to not very high.
3. As per shows in the chats in the file , till certain level of humidity level , demand increases , when humidity is too low or very high , there are very few observations.
4. Humidity level , 40% to 70% highest records have been observed.
5. As per hourly average number of cycles rented by registered and casual customer plots ,
6. Registered Customers seems to be using rental cycles mostly for work commute purposes.
7. registered customers are much higher than the casual customers. 81% customers are Registered and 19% only are casual riders. Which is good thing for a consistent business. Though it is recommended to introduce more go-to offers and strategical execution to attract more casual riders, that further increase chances of converting to consistent users.
8. Confidence interval of average number of cycles rented by registered customers is (153,157) and casual customers is (35,37).
9. Demand for cycles increases during the rush hours specifically during working days , from morning 7 to 9 am and in evening 4 to 8pm.
10. on off days demands are higher from 10 am to evening 7pm.
11. Though it is concluded from statistical tests, that demand on weekdays and off-days are similar. We can say demand is equal with 95% confidene.
12. During spring season , customers prefer less likely to rent cycle. demand increases in summer and fall season.