```
!gdown 1ZpORKGL4HpjAB_H7F0l7DNddKEUCZHF3 -O 'yulu_data.csv'
```

```
Downloading...
From: https://drive.google.com/uc?id=1ZpORKGL4HpjAB_H7F0l7DNddKEUCZHF3
To: /content/yulu_data.csv
100% 648k/648k [00:00<00:00, 115MB/s]
```

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from scipy.stats import ttest_ind, f_oneway, shapiro, levene, kruskal, chi2_contingency
from statsmodels.graphics.gofplots import qqplot
```

```
yu = pd.read_csv('yulu_data.csv')
```

```
yu.head()
```

	datetime	season	holiday	workingday	weather	temp	atemp	h
0	2011-01-01 00:00:00	1	0	0	1	9.84	14.395	
1	2011-01-01	1	0	0	1	9.02	13.635	

```
yu.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
```



```
yu.shape
```

```
(10886, 12)
```

```
yu.isna().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
```

```
np.any(yu.isna().any(axis=1))
```

```
False
```

```
yu.loc[yu["season"] == 1,"season"] = "spring"
yu.loc[yu["season"] == 2,"season"] = "summer"
yu.loc[yu["season"] == 3,"season"] = "fall"
yu.loc[yu["season"] == 4,"season"] = "winter"
```

```
season = yu["season"].value_counts().index.sort_values()
season
```

```
Index(['fall', 'spring', 'summer', 'winter'], dtype='object')
```

```
yu.describe()
```

	holiday	workingday	weather	temp	
count	10886.000000	10886.000000	10886.000000	10886.00000	10886
mean	0.028569	0.680875	1.418427	20.23086	23
std	0.166599	0.466159	0.633839	7.79159	8
min	0.000000	0.000000	1.000000	0.82000	0
25%	0.000000	0.000000	1.000000	13.94000	16
50%	0.000000	1.000000	1.000000	20.50000	24

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

	datetime	season
count	10886	10886
unique	10886	4
top	2011-01-01 00:00:00	winter

```
yu.shape
```

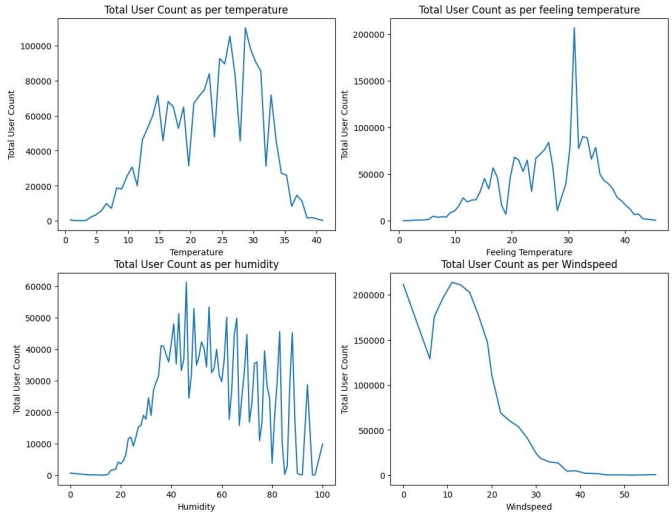
```
(10886, 12)
```

```
# Total User Count as per Continuous Variable(s) temp, atemp, humidity, windspeed
fig = plt.figure(figsize=(13,10))
```

Saved successfully!

```
yu_temp = yu.groupby('temp')['count'].agg(['sum']).sort_values(by ='temp').reset_index()
plt.xlabel("Temperature")
plt.ylabel("Total User Count")
plt.title("Total User Count as per temperature")
plt.plot(yu_temp['temp'],yu_temp['sum'] )

plt.subplot(2,2, 2)

yu_atemp = yu.groupby('atemp')['count'].agg(['sum']).sort_values(by ='atemp').reset_index()
plt.xlabel("Feeling Temperature")
plt.ylabel("Total User Count")
plt.title("Total User Count as per feeling temperature")
plt.plot(yu_atemp['atemp'],yu_atemp['sum'] )

plt.subplot(2,2, 3)

yu_humidity = yu.groupby('humidity')['count'].agg(['sum']).sort_values(by ='humidity').reset_index()
plt.xlabel("Humidity")
plt.ylabel("Total User Count")
plt.title("Total User Count as per humidity")
plt.plot(yu_humidity['humidity'],yu_humidity['sum'] )

plt.subplot(2,2, 4)

yu_windspeed = yu.groupby('windspeed')['count'].agg(['sum']).sort_values(by ='windspeed').reset_index()
plt.xlabel("Windspeed")
plt.ylabel("Total User Count")
plt.title("Total User Count as per Windspeed")
plt.plot(yu_windspeed['windspeed'],yu_windspeed['sum'] )
```

```
[<matplotlib.lines.Line2D at 0x7f1a639af850>]
```

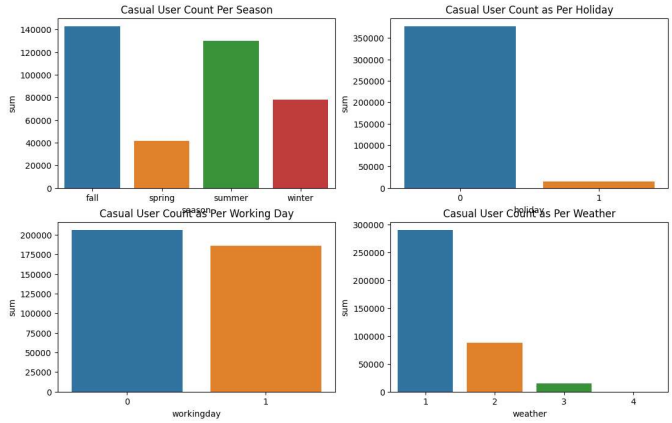



```
# Casual User Count as per Categorical Variable(s) Season, Holiday , WorkingDay, Weather
fig = plt.figure(figsize=(13,8))
```

Saved successfully!

```
yu_season = yu.groupby('season')['casual'].agg(['sum']).sort_values(by ='season').reset_index()
plt.xlabel("Season")
plt.ylabel("Casual User Count")
plt.title("Casual User Count Per Season")
sns.barplot(x = yu_season["season"] , y = yu_season["sum"]  )

plt.subplot(2,2, 2)

yu_holiday = yu.groupby('holiday')['casual'].agg(['sum']).sort_values(by ='holiday').reset_index()
plt.xlabel("Holiday")
plt.ylabel("Casual User Count")
plt.title("Casual User Count as Per Holiday")
sns.barplot(x = yu_holiday["holiday"] , y = yu_holiday["sum"]  )

plt.subplot(2,2, 3)

yu_workingDay = yu.groupby('workingday')['casual'].agg(['sum']).sort_values(by ='workingday').reset_index()
plt.xlabel("Working Day")
plt.ylabel("Casual User Count")
plt.title("Casual User Count as Per Working Day")
sns.barplot(x = yu_workingDay["workingday"] , y = yu_workingDay["sum"]  )

plt.subplot(2,2, 4)

yu_weather = yu.groupby('weather')['casual'].agg(['sum']).sort_values(by ='weather').reset_index()
plt.xlabel("Weather")
plt.ylabel("Casual User Count")
plt.title("Casual User Count as Per Weather")
sns.barplot(x = yu_weather["weather"] , y = yu_weather["sum"]  )
```

```
<Axes: title={'center': 'Casual User Count as Per Weather'},
xlabel='weather', ylabel='sum'>
```

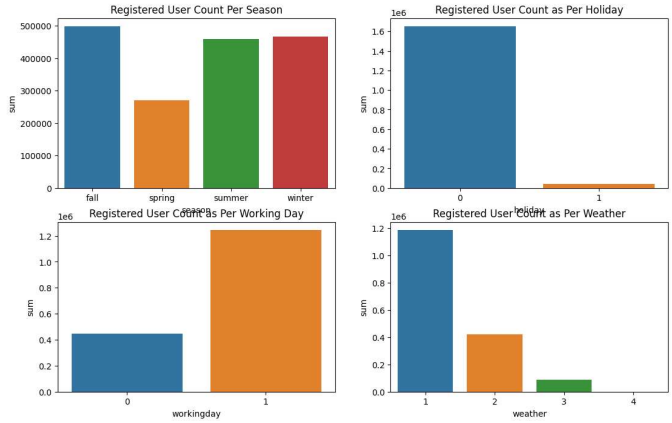



```
# Registered User Count as per Categorical Variable(s) Season, Holiday, WorkingDay, Weather
fig = plt.figure(figsize=(13,8))

plt.subplot(2,2, 1)

yu_season = yu.groupby('season')['registered'].agg(['sum']).sort_values(by ='season').reset_index()
plt.xlabel("Season")
plt.ylabel("Registered User Count")
plt.title("Registered User Count Per Season")
sns.barplot(x = yu_season["season"] , y = yu_season["sum"]  )
```

Saved successfully!

```
yu_holiday = yu.groupby('holiday')['registered'].agg(['sum']).sort_values(by ='holiday').reset_index()
plt.xlabel("Holiday")
plt.ylabel("Registered User Count")
plt.title("Registered User Count as Per Holiday")
sns.barplot(x = yu_holiday["holiday"] , y = yu_holiday["sum"]  )

plt.subplot(2,2, 3)

yu_workingDay = yu.groupby('workingday')['registered'].agg(['sum']).sort_values(by ='workingday').reset_index()
plt.xlabel("Working Day")
plt.ylabel("Registered User Count")
plt.title("Registered User Count as Per Working Day")
sns.barplot(x = yu_workingDay['workingday'], y = yu_workingDay['sum'] )

plt.subplot(2,2, 4)

yu_weather = yu.groupby('weather')['registered'].agg(['sum']).sort_values(by ='weather').reset_index()
plt.xlabel("Weather")
plt.ylabel("Registered User Count")
plt.title("Registered User Count as Per Weather")
sns.barplot(x = yu_weather['weather'], y = yu_weather['sum'] )
```

```
<Axes: title={'center': 'Registered User Count as Per Weather'},
xlabel='weather', ylabel='sum'>
```

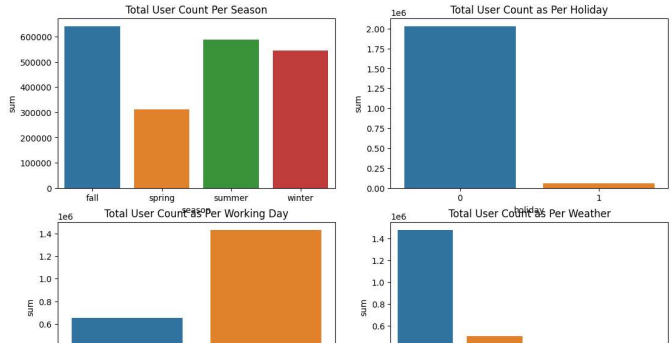



```
# Total User Count as per Categorical Variable(s) Season, Holiday, WorkingDay, Weather
fig = plt.figure(figsize=(13,8))

plt.subplot(2,2, 1)

yu_season = yu.groupby('season')['count'].agg(['sum']).sort_values(by ='season').reset_index()
plt.xlabel("Season")
plt.ylabel("Total User Count")
plt.title("Total User Count Per Season")
sns.barplot(x = yu_season["season"] , y = yu_season["sum"]  )

plt.subplot(2,2, 2)

yu_holiday = yu.groupby('holiday')['count'].agg(['sum']).sort_values(by ='holiday').reset_index()
plt.xlabel("Holiday")
plt.ylabel("Total User Count")
plt.title("Total User Count as Per Holiday")
sns.barplot(x = yu_holiday["holiday"] , y = yu_holiday["sum"]  )
```

Saved successfully!

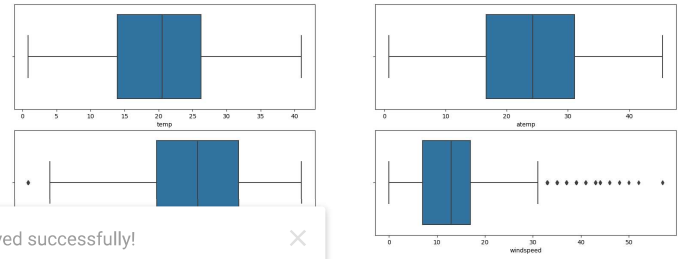
```
yu_workingDay = yu.groupby('workingday')['count'].agg(['sum']).sort_values(by ='workingday').reset_index()
plt.xlabel("Working Day")
plt.ylabel("Total User Count")
plt.title("Total User Count as Per Working Day")
sns.barplot(x = yu_workingDay['workingday'], y = yu_workingDay['sum'] )

plt.subplot(2,2, 4)

yu_weather = yu.groupby('weather')['count'].agg(['sum']).sort_values(by ='weather').reset_index()
plt.xlabel("Weather")
plt.ylabel("Total User Count")
plt.title("Total User Count as Per Weather")
sns.barplot(x = yu_weather['weather'], y = yu_weather['sum'] )
```

```
<Axes: title={'center': 'Total User Count as Per Weather'},
xlabel='weather', ylabel='sum'>
```




```
#Outlier

fig = plt.figure(figsize=(20,7))

plt.subplot(2,2, 1)
sns.boxplot(data = yu , x ='temp' )

plt.subplot(2,2, 2)
sns.boxplot(data = yu , x = 'atemp' )

plt.subplot(2,2, 3)
sns.boxplot(data = yu , x = 'humidity' )

plt.subplot(2,2, 4)
sns.boxplot(data = yu , x = 'windspeed' )
```

```
<Axes: xlabel='windspeed'>
```



Saved successfully!

Hypothesis Testing

Working Day has an effect on the number of electric cycles rented ?

2 Sample T-Test

```
# HO : Working Day has no effect on the number of electric cycles rented
# HA : Working Day has effect on the number of electric cycles rented

electricCylesRentedOnWorkingDay = yu[yu['workingday'] == 1]['count']
```

```
electricCylesRentedOnNonWorkingDay = yu[yu['workingday'] == 0]['count']

# Let alpha be 0.05
alpha = 0.05
t_stat, p_value = ttest_ind(electricCylesRentedOnWorkingDay,electricCylesRentedOnNonWorkingDay) # 2 Sample T-Test

if p_value < alpha:
  print('Reject HO : Working Day has effect on the number of electric cycles rented')
else:
  print(p_value,'Cant Reject HO : Dont have sufficient evidence to say Working Day has effect on the number of electric cycles rented')
```

```
0.22644804226361348 Cant Reject HO : Dont have sufficient evidence to say Working Day has effect on the number of electric cycles r
```

Holiday has an effect on the number of electric cycles rented ?

Lets check using 2 Sample T-Test

```
# HO : Holiday has no effect on the number of electric cycles rented
# HA : Holiday has effect on the number of electric cycles rented

electricCylesRentedOnHoliday = yu[yu['holiday'] == 1]['count']
electricCylesRentedOnNonHoliDay = yu[yu['holiday'] == 0]['count']

# Let alpha be 0.05
alpha = 0.05
t_stat, p_value = ttest_ind(electricCylesRentedOnHoliday,electricCylesRentedOnNonHoliDay) # 2 Sample T-Test

if p_value < alpha:
  print('Reject HO : Holiday has effect on the number of electric cycles rented')
else:
  print(p_value,'Cant Reject HO : Dont have sufficient evidence to say Holiday has effect on the number of electric cycles rented')
```

```
0.5736923883271103 Cant Reject HO : Dont have sufficient evidence to say Holiday has effect on the number of electric cycles rented
```

No. of cycles rented similar or different in different seasons ?

Saved successfully!

ANNOVA Test

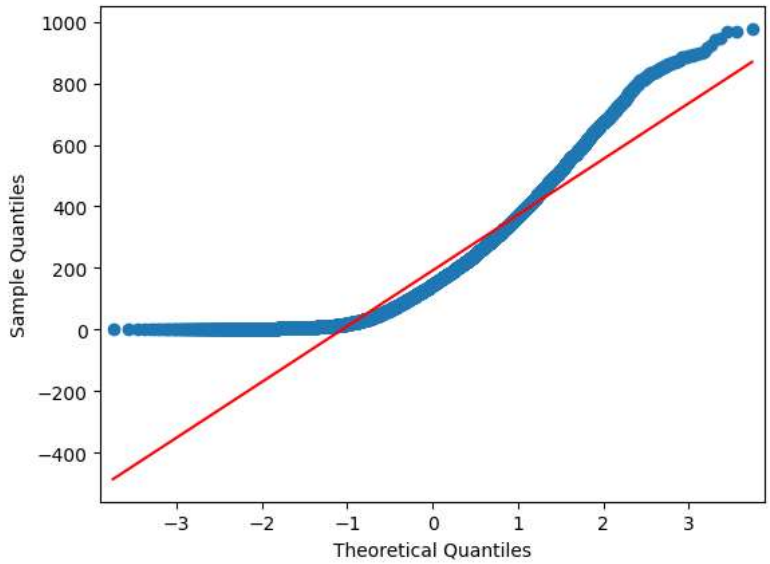
```
# HO : No. of cycles rented similar in different seasons
# HA : No. of cycles rented different in different seasons
# Let alpha be 0.05
alpha = 0.05

electricCylesRentedInSpringSeason = yu[yu['season'] == 'spring']['count']
electricCylesRentedInSummerSeason = yu[yu['season'] == 'summer']['count']
electricCylesRentedInFallSeason = yu[yu['season'] == 'fall']['count']
electricCylesRentedInWinterSeason = yu[yu['season'] == 'winter']['count']

# Performed ANNOVA Test
t_stat, p_value = f_oneway(electricCylesRentedInSpringSeason,electricCylesRentedInSummerSeason,electricCylesRentedInFallSeason, electricC


if p_value < alpha:
  print(p_value,'Reject HO : No. of cycles rented different in different seasons')
else:
  print('No. of cycles rented similar in different seasons')
```

```
6.164843386499654e-149 Reject HO : No. of cycles rented different in different seasons
```

Checking assumptions of ANNOVA

1. Gaussian or not
    - QQ Plot
    - Sharpio Test
2. Equal Variance in diferent groups or not
    - Levenes Test

```
# QQ Plot

qqplot(yu["count"], line = "s")
plt.show()
```




```
# Sharpio Test

# HO: Data is Gaussian
# HA: Data is not Gaussian

test_stat, p_value = shapiro(electricCylesRentedInSpringSeason)
print(p_value)
```

```
0.0
```

```
test_stat, p_value = shapiro(electricCylesRentedInSummerSeason)
print(p_value)
```



```
test_stat, p_value = shapiro(electricCylesRentedInFallSeason)
print(p_value)
```

```
1.043458045587339e-36
```

```
test_stat, p_value = shapiro(electricCylesRentedInWinterSeason)
print(p_value)
```

```
1.1301682309549298e-39
```

▾ The p-value of Sharipo Test is less then significance level of 5 percent. Thereby indicating to reject HO.

```
# Variance is equal or not in different season.
# LEVENES Test

# HO: Variances are equal
# HA: Variances are not equal

alpha = 0.05
levene_stat, p_value = levene(electricCylesRentedInSpringSeason, electricCylesRentedInSummerSeason, electricCylesRentedInFallSeason, elec
print(p_value)
if p_value < 0.05:
    print("Variances are not equal")
```

```
1.0147116860043298e-118
Variances are not equal
```

▾ If assumptions of ANOVA don't hold, we do Kruskal Wallis Test

```
# HO : No. of cycles rented similar in different seasons
# HA : No. of cycles rented different in different seasons
```

```
_,p_value = kruskal(electricCylesRentedInSpringSeason, electricCylesRentedInSummerSeason, electricCylesRentedInFallSeason, electricCylesF

print(p_value)

if p_value < alpha:
  print(p_value, 'Reject HO : No. of cycles rented different in different seasons')
else:
  print(p_value, 'No. of cycles rented similar in different seasons')
```

```
2.479008372608633e-151
2.479008372608633e-151 Reject HO : No. of cycles rented different in different seasons
```

No. of cycles rented similar or different in different weather ?

ANNOVA Test

```
# HO : No. of cycles rented similar in different weather
# HA : No. of cycles rented different in different weather
# Let alpha be 0.05
alpha = 0.05

electricCylesRentedInW1 = yu[yu['weather'] == 1]['count']
electricCylesRentedInW2 = yu[yu['weather'] == 2]['count']
electricCylesRentedInW3 = yu[yu['weather'] == 3]['count']
electricCylesRentedInW4 = yu[yu['weather'] == 4]['count']

# Performed ANNOVA Test
t_stat, p_value = f_oneway(electricCylesRentedInW1, electricCylesRentedInW2 ,electricCylesRentedInW3 , electricCylesRentedInW4 )


if p_value < alpha:
  print(p_value,'Reject HO : No. of cycles rented different in different weather')
else:
  print(p_value,'No. of cycles rented similar in different weather')
```

```
5.482069475935669e-42 Reject HO : No. of cycles rented different in different weather
```

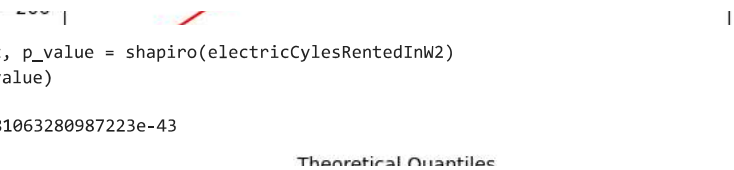

0VA

1. Gaussian or not
   - QQ Plot
   - Sharpio Test
2. Equal Variance in diferent groups or not
   - Levenes Test

```
# QQ Plot

qqplot(yu["count"], line = "s")
plt.show()
```

```
# Sharpio Test

# HO: Data is Gaussian
# HA: Data is not Gaussian

test_stat, p_value = shapiro(electricCylesRentedInW1)
print(p_value)
```

```
0.0
/usr/local/lib/python3.10/dist-packages/scipy/stats/_morestats.py:1816: UserWarning: p-value may not be accurate for N > 5000.
  warnings.warn("p-value may not be accurate for N > 5000.")
```



```
test_stat, p_value = shapiro(electricCylesRentedInW2)
print(p_value)
```

```
9.781063280987223e-43
```



```
test_stat, p_value = shapiro(electricCylesRentedInW3)
print(p_value)
```

```
3.876090133422781e-33
```

```
#test_stat, p_value = shapiro(electricCylesRentedInW4)
#print(p_value)
# Data must be at least length 3 for shapiro Test.
```

The p-value of Sharipo Test is less then significance level of 5 percent. Thereby indicating to reject HO.

```
# Variance is equal or not in different weather.
# LEVENES Test

# HO: Variances are equal
# HA: Variances are not equal
```

Saved successfully!

```
icCylesRentedInW1, electricCylesRentedInW2 ,electricCylesRentedInW3 , electricCylesRentedInW4 )
print(p_value)
if p_value < 0.05:
    print(p_value, "Reject HO: Variances are not equal")
else:
  print(p_value, 'Variances are not equal')
```

```
3.504937946833238e-35
3.504937946833238e-35 Reject HO: Variances are not equal
```

If assumptions of ANOVA don't hold, we do Kruskal Wallis Test

```
# HO : No. of cycles rented similar in different weather.
# HA : No. of cycles rented different in different weather.

_,p_value = kruskal(electricCylesRentedInW1, electricCylesRentedInW2 ,electricCylesRentedInW3 , electricCylesRentedInW4 )

print(p_value)

if p_value < alpha:
  print(p_value, 'Reject HO : No. of cycles rented different in different weather')
else:
  print(p_value, 'No. of cycles rented similar in different weather')
```

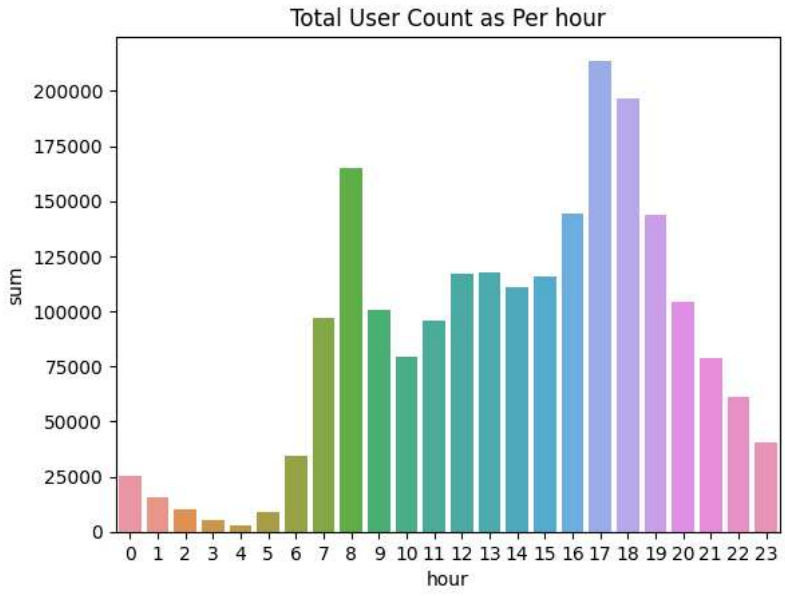
```
3.501611300708679e-44
3.501611300708679e-44 Reject HO : No. of cycles rented different in different weather
```

No. of cycles rented similar or different in different hour of the day ?

```
yu['datetime'] = pd.to_datetime(yu['datetime'])
yu['hour'] = yu['datetime'].dt.hour
yu['hour'].value_counts()
```

```
yu_hour = yu.groupby('hour')['count'].agg(['sum']).sort_values(by ='hour').reset_index()
plt.xlabel("hour")
plt.ylabel("Total User Count")
plt.title("Total User Count as Per hour")
sns.barplot(x = yu_hour["hour"] , y = yu_hour["sum"]  )
```

```
<Axes: title={'center': 'Total User Count as Per hour'}, xlabel='hour', ylabel='sum'>
```




```
# HO : No. of cycles rented similar in different hour of the day
# HA : No. of cycles rented different in different hour of the day
alpha = 0.05

electricCylesRentedInH0 = yu[yu['hour'] == 0]['count']
electricCylesRentedInH1 = yu[yu['hour'] == 1]['count']
electricCylesRentedInH2 = yu[yu['hour'] == 2]['count']
electricCylesRentedInH3 = yu[yu['hour'] == 3]['count']
electricCylesRentedInH4 = yu[yu['hour'] == 4]['count']
```

Saved successfully!

```
electricCylesRentedInH5 = yu[yu['hour'] == 5]['count']
electricCylesRentedInH6 = yu[yu['hour'] == 6]['count']
electricCylesRentedInH7 = yu[yu['hour'] == 7]['count']
electricCylesRentedInH8 = yu[yu['hour'] == 8]['count']

electricCylesRentedInH9 = yu[yu['hour'] == 9]['count']
electricCylesRentedInH10 = yu[yu['hour'] == 10]['count']
electricCylesRentedInH11 = yu[yu['hour'] == 11]['count']
electricCylesRentedInH12 = yu[yu['hour'] == 12]['count']

electricCylesRentedInH13 = yu[yu['hour'] == 13]['count']
electricCylesRentedInH14 = yu[yu['hour'] == 14]['count']
electricCylesRentedInH15 = yu[yu['hour'] == 15]['count']
electricCylesRentedInH16 = yu[yu['hour'] == 16]['count']

electricCylesRentedInH17 = yu[yu['hour'] == 17]['count']
electricCylesRentedInH18 = yu[yu['hour'] == 18]['count']
electricCylesRentedInH19 = yu[yu['hour'] == 19]['count']
electricCylesRentedInH20 = yu[yu['hour'] == 20]['count']

electricCylesRentedInH21 = yu[yu['hour'] == 21]['count']
electricCylesRentedInH22 = yu[yu['hour'] == 22]['count']
electricCylesRentedInH23 = yu[yu['hour'] == 23]['count']


# Performed ANNOVA Test
t_stat, p_value = f_oneway(
electricCylesRentedInH0 ,
electricCylesRentedInH1 ,
electricCylesRentedInH2 ,
electricCylesRentedInH3 ,
electricCylesRentedInH4 ,
electricCylesRentedInH5 ,
electricCylesRentedInH6 ,
electricCylesRentedInH7 ,
electricCylesRentedInH8 ,
electricCylesRentedInH9 ,
electricCylesRentedInH10 ,
electricCylesRentedInH11 ,
```

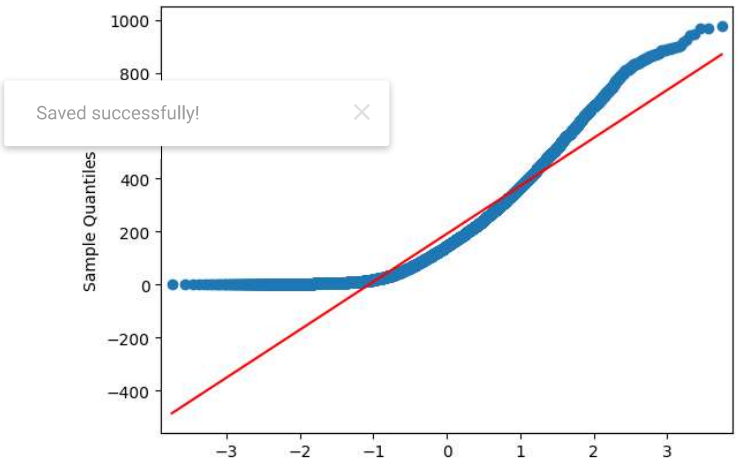
```
electricCylesRentedInH12 ,
electricCylesRentedInH13 ,
electricCylesRentedInH14 ,
electricCylesRentedInH15 ,
electricCylesRentedInH16 ,
electricCylesRentedInH17 ,
electricCylesRentedInH18 ,
electricCylesRentedInH19 ,
electricCylesRentedInH20 ,
electricCylesRentedInH21 ,
electricCylesRentedInH22 ,
electricCylesRentedInH23
)

if p_value < alpha:
  print(p_value,'Reject HO : No. of cycles rented different in different hour of the day')
else:
  print(p_value,'No. of cycles rented similar in different hour of the day')
```

```
0.0 Reject HO : No. of cycles rented different in different hour of the day
```

Checking assumptions of ANNOVA

1. Gaussian or not
   - QQ Plot
   - Sharpio Test
2. Equal Variance in diferent groups or not
   - Levenes Test

```
# QQ Plot

qqplot(yu["count"], line = "s")
plt.show()
```




```
# Sharpio Test

# HO: Data is Gaussian
# HA: Data is not Gaussian

test_stat_0, p_value_0 = shapiro(electricCylesRentedInH0)
test_stat_1, p_value_1 = shapiro(electricCylesRentedInH1)
test_stat_2, p_value_2 = shapiro(electricCylesRentedInH2)
test_stat_3, p_value_3 = shapiro(electricCylesRentedInH3)

test_stat_4, p_value_4 = shapiro(electricCylesRentedInH4)
test_stat_5, p_value_5 = shapiro(electricCylesRentedInH5)
test_stat_6, p_value_6 = shapiro(electricCylesRentedInH6)
test_stat_7, p_value_7 = shapiro(electricCylesRentedInH7)

test_stat_8, p_value_8 = shapiro(electricCylesRentedInH8)
test_stat_9, p_value_9 = shapiro(electricCylesRentedInH9)
test_stat_10, p_value_10 = shapiro(electricCylesRentedInH10)
test_stat_11, p_value_11 = shapiro(electricCylesRentedInH11)
```

```
test_stat_12, p_value_12 = shapiro(electricCylesRentedInH12)
test_stat_13, p_value_13 = shapiro(electricCylesRentedInH13)
test_stat_14, p_value_14 = shapiro(electricCylesRentedInH14)
test_stat_15, p_value_15 = shapiro(electricCylesRentedInH15)

test_stat_16, p_value_16 = shapiro(electricCylesRentedInH16)
test_stat_17, p_value_17 = shapiro(electricCylesRentedInH17)
test_stat_18, p_value_18 = shapiro(electricCylesRentedInH18)
test_stat_19, p_value_19 = shapiro(electricCylesRentedInH19)

test_stat_20, p_value_20 = shapiro(electricCylesRentedInH20)
test_stat_21, p_value_21 = shapiro(electricCylesRentedInH21)
test_stat_22, p_value_22 = shapiro(electricCylesRentedInH22)
test_stat_23, p_value_23 = shapiro(electricCylesRentedInH23)
print(p_value_0, p_value_1, p_value_2, p_value_3, p_value_4, p_value_5, p_value_6, p_value_7, p_value_8, p_value_9, p_value_10, p_value_1
```

```
7.727760389366201e-20 1.706304718043419e-23 6.039973178430634e-25 2.4333970649677658e-24 5.520921786687956e-15 3.152888228274975e-1
```

The p-value of Sharipo Test is less then significance level of 5 percent. Thereby indicating to reject HO.

```
# Variance is equal or not in different hour of the day.
# LEVENES Test

# HO: Variances are equal
# HA: Variances are not equal

alpha = 0.05
levene_stat, p_value = levene(
electricCylesRentedInH0 ,
electricCylesRentedInH1 ,
electricCylesRentedInH2 ,
electricCylesRentedInH3 ,
electricCylesRentedInH4 ,
electricCylesRentedInH5 ,
electricCylesRentedInH6 ,
electricCylesRentedInH7 ,
electricCylesRentedInH8 ,
electricCylesRentedInH9 ,
```

Saved successfully!

```
electricCylesRentedInH13 ,
electricCylesRentedInH14 ,
electricCylesRentedInH15 ,
electricCylesRentedInH16 ,
electricCylesRentedInH17 ,
electricCylesRentedInH18 ,
electricCylesRentedInH19 ,
electricCylesRentedInH20 ,
electricCylesRentedInH21 ,
electricCylesRentedInH22 ,
electricCylesRentedInH23
)
print(p_value)
if p_value < 0.05:
    print(p_value, "Reject HO: Variances are not equal")
else:
  print(p_value, 'Variances are not equal')
```

```
0.0
0.0 Reject HO: Variances are not equal
```

If assumptions of ANOVA don't hold, we do Kruskal Wallis Test

```
# HO : No. of cycles rented similar in different hour of the day
# HA : No. of cycles rented different in different hour of the day

_,p_value = kruskal(
electricCylesRentedInH0 ,
electricCylesRentedInH1 ,
electricCylesRentedInH2 ,
electricCylesRentedInH3 ,
electricCylesRentedInH4 ,
electricCylesRentedInH5 ,
electricCylesRentedInH6 ,
```

```
electricCylesRentedInH7 ,
electricCylesRentedInH8 ,
electricCylesRentedInH9 ,
electricCylesRentedInH10 ,
electricCylesRentedInH11 ,
electricCylesRentedInH12 ,
electricCylesRentedInH13 ,
electricCylesRentedInH14 ,
electricCylesRentedInH15 ,
electricCylesRentedInH16 ,
electricCylesRentedInH17 ,
electricCylesRentedInH18 ,
electricCylesRentedInH19 ,
electricCylesRentedInH20 ,
electricCylesRentedInH21 ,
electricCylesRentedInH22 ,
electricCylesRentedInH23
)

print(p_value)

if p_value < alpha:
  print(p_value, 'Reject HO : No. of cycles rented different in different hour of the day')
else:
  print(p_value, 'No. of cycles rented similar in different hour of the day')
```

```
0.0
0.0 Reject HO : No. of cycles rented different in different hour of the day
```

Weather is dependent on season ?

```
weatherSeasonCrossTab = pd.crosstab(yu['weather'] , yu['season'])
weatherSeasonCrossTab
```

weather	season	fall	spring	summer	winter
1		1930	1759	1801	1702
2					807
3					225
4		0	1	0	0

Saved successfully!

Since Both weather and season are categorical variables, we will be performing Chi-Square Test.

Note : Before performing Chi-Square Test we need to remove Row 4, since it has value less than 5 which is the violation of the test assumptions.

```
weatherSeasonCrossTab = weatherSeasonCrossTab.loc[[1,2,3],:]
weatherSeasonCrossTab
```

weather	season	fall	spring	summer	winter
1		1930	1759	1801	1702
2		604	715	708	807
3		199	211	224	225

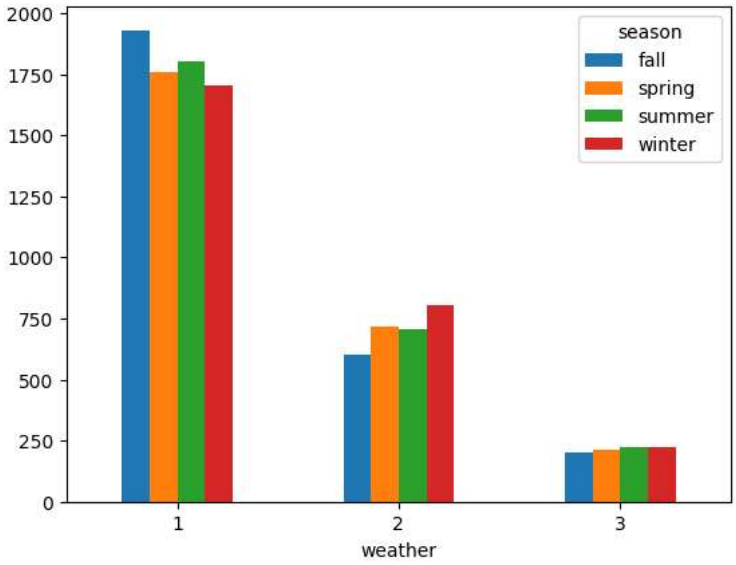
Chi-Square Test

```
# HO : Weather and Season are independent.
# HA : Weather and Season are dependent.

#Significance Level
alpha = 0.05
result = chi2_contingency(weatherSeasonCrossTab)

if result.pvalue < alpha:
```

```
  print(result.pvalue, 'Reject HO : Weather and Season are dependent.')
else:
  print(result.pvalue, 'Weather and Season are independent.')
```

```
    2.8260014509929403e-08 Reject HO : Weather and Season are dependent.
```

```
# Creating barplot
barplot = weatherSeasonCrossTab.plot.bar(rot=0)
```



Saved successfully!

0s completed at 11:09 AM