```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import scipy.stats as stats
import math
from scipy.stats import norm, binom, ttest_1samp, ttest_ind,
ttest_rel, chisquare, chi2_contingency, f_oneway, kruskal, shapiro,
levene, pearsonr, spearmanr

df = pd.read_csv("/content/yolo.txt", parse_dates= [0], dayfirst =
True, na_values ='NA', date_parser = lambda x: pd.to_datetime(x,
format = '%Y-%m-%d %H:%M:%S'))

df.head()
```


{"summary":"{\n \"name\": \"df\",\n \"rows\": 10886,\n \"fields\":
[\n   {\n   \"column\": \"datetime\",\n   \"properties\": {\n
\"dtype\": \"date\",\n   \"min\": \"2011-01-01 00:00:00\",\n
\"max\": \"2012-12-19 23:00:00\",\n   \"num_unique_values\":
10886,\n   \"samples\": [\n   \"2011-07-19 11:00:00\",\n
\"2012-01-16 06:00:00\",\n   \"2011-12-11 18:00:00\"\
n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"season\",\n   \"properties\": {\n   \"dtype\": \"number\",\n
\"std\": 1,\n   \"min\": 1,\n   \"max\": 4,\n
\"num_unique_values\": 4,\n   \"samples\": [\n   2,\n
4,\n   1\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"holiday\",\n   \"properties\": {\n   \"dtype\": \"number\",\
n   \"std\": 0,\n   \"min\": 0,\n   \"max\": 1,\n
\"num_unique_values\": 2,\n   \"samples\": [\n   1,\n
0\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"workingday\",\n   \"properties\": {\n   \"dtype\":
\"number\",\n   \"std\": 0,\n   \"min\": 0,\n
\"max\": 1,\n   \"num_unique_values\": 2,\n   \"samples\":
[\n   1,\n   0\n   ],\n   \"semantic_type\":
\"\",\n   \"description\": \"\"\n   }\n   },\n   {\n
\"column\": \"weather\",\n   \"properties\": {\n   \"dtype\":
\"number\",\n   \"std\": 0,\n   \"min\": 1,\n
\"max\": 4,\n   \"num_unique_values\": 4,\n   \"samples\":
[\n   2,\n   4\n   ],\n   \"semantic_type\":
\"\",\n   \"description\": \"\"\n   }\n   },\n   {\n
\"column\": \"temp\",\n   \"properties\": {\n   \"dtype\":
\"number\",\n   \"std\": 7.791589843987567,\n   \"min\":
0.82,\n   \"max\": 41.0,\n   \"num_unique_values\": 49,\n
\"samples\": [\n   6.56,\n   1.64\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"atemp\",\n   \"properties\": {\

n   \"dtype\": \"number\",\n   \"std\": 8.474600626484948,\n   \"min\": 0.76,\n   \"max\": 45.455,\n   \"num_unique_values\": 60,\n   \"samples\": [\n   14.395,\n   16.665\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"humidity\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 19,\n   \"min\": 0,\n   \"max\": 100,\n   \"num_unique_values\": 89,\n   \"samples\": [\n   29,\n   61\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"windspeed\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 8.164537326838689,\n   \"min\": 0.0,\n   \"max\": 56.9969,\n   \"num_unique_values\": 28,\n   \"samples\": [\n   22.0028,\n   43.0006\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"casual\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 49,\n   \"min\": 0,\n   \"max\": 367,\n   \"num_unique_values\": 309,\n   \"samples\": [\n   287,\n   47\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"registered\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 151,\n   \"min\": 0,\n   \"max\": 886,\n   \"num_unique_values\": 731,\n   \"samples\": [\n   566,\n   9\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"count\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 181,\n   \"min\": 1,\n   \"max\": 977,\n   \"num_unique_values\": 822,\n   \"samples\": [\n   626,\n   256\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   }\n]\n}","type":"dataframe","variable_name":"df"}

Company Profile

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!

Prob statement

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.

```
df.shape
```

```
(10886, 12)
```

```
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
```

No missing values observed hence no imputation required.

```
df.dtypes
```

```
datetime      datetime64[ns]
season                 int64
holiday                int64
workingday             int64
weather                int64
temp                 float64
atemp                float64
humidity               int64
windspeed            float64
casual                 int64
registered             int64
count                  int64
dtype: object
```

datetime column has been changed using parse_function to datetime from object dtype.

```
df['hour'] = df['datetime'].dt.hour
```

```
df['timeslot'] = df['hour'].apply(lambda x: 'Dawn' if x <=4
else("Early Morning"

if x<=9 else ("Noon"

if x<=16 else ("Late Evening"

if x<=21 else "Night"))))

df['month'] = df['datetime'].dt.month.astype('str')

df['year'] = df['datetime'].dt.year.astype('str')
df.head()
```

{"repr_error":"'str' object has no attribute 'empty'","type":"dataframe","variable_name":"df"}

```
df.dtypes
```

```
datetime      datetime64[ns]
season                 int64
holiday                int64
workingday             int64
weather                int64
temp                 float64
atemp                float64
humidity               int64
windspeed            float64
casual                 int64
registered             int64
count                  int64
hour                   int64
timeslot              object
month                 object
year                  object
dtype: object
```

```
df.describe(include = 'all')
```

```
<ipython-input-12-74aa2f970831>:1: FutureWarning: Treating datetime
data as categorical rather than numeric in `.describe` is deprecated
and will be removed in a future version of pandas. Specify
`datetime_is_numeric=True` to silence this warning and adopt the
future behavior now.
  df.describe(include = 'all')
```

{"repr_error":"'str' object has no attribute 'empty'","type":"dataframe"}

Insights: 1. 75% of bike were rented during Winter season by the users.

1. 50% of the bikes were rented during Clear, Few clouds, partly cloudy, partly cloudy.

2. Avg temp and avg a_temp were ~20 and ~23 degree celsius.
3. Mean humidity level was noted as 61 when bikes were rented.
4. Avg windspeed of 12.7 was observed during rent on model.
5. max no. of bikes rented during noon i.e. 3190.
6. max no. of bikes rented during month of May.

```
df.duplicated().sum()
```

```
0
```

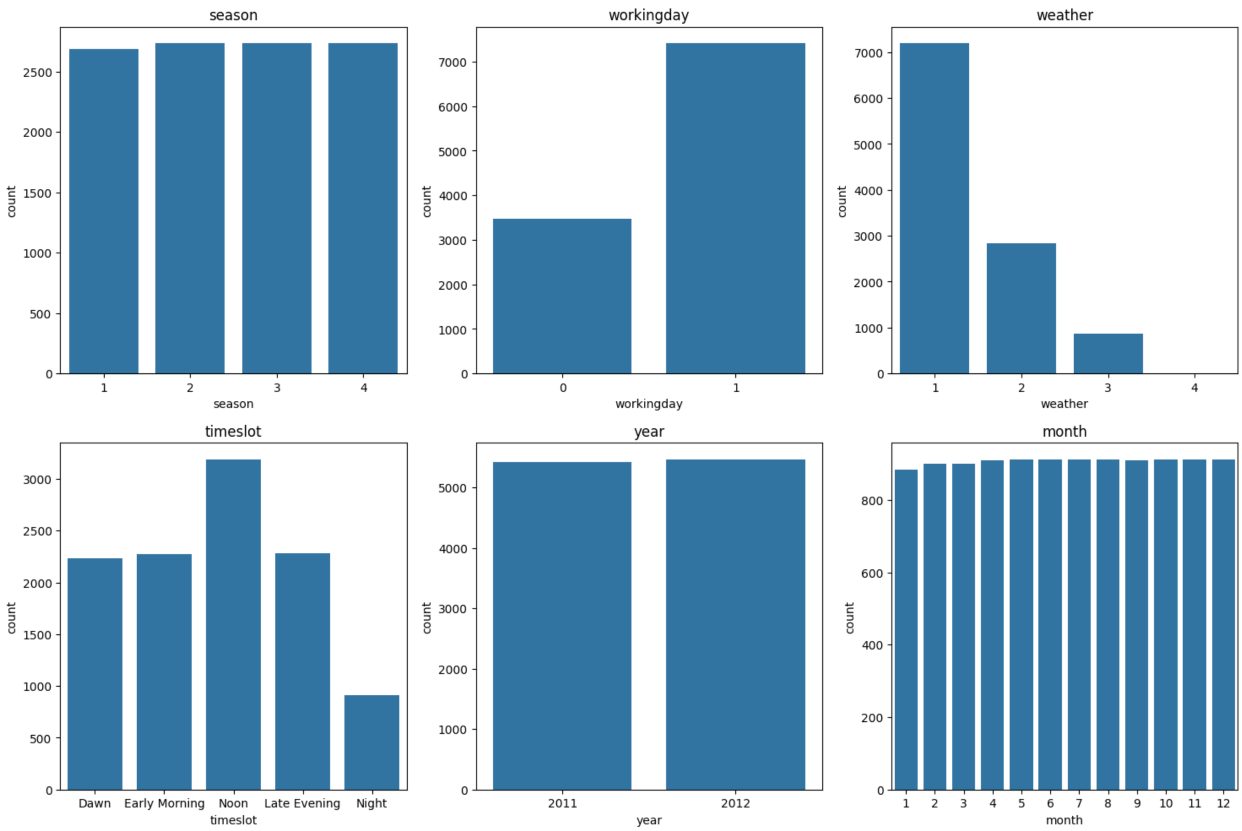
```
# number of unique values in each categorical columns
cat_cols = ['season', 'workingday', 'weather', 'timeslot', 'year',
'month']
df[cat_cols].melt().groupby(['variable', 'value'])[['value']].count()
```

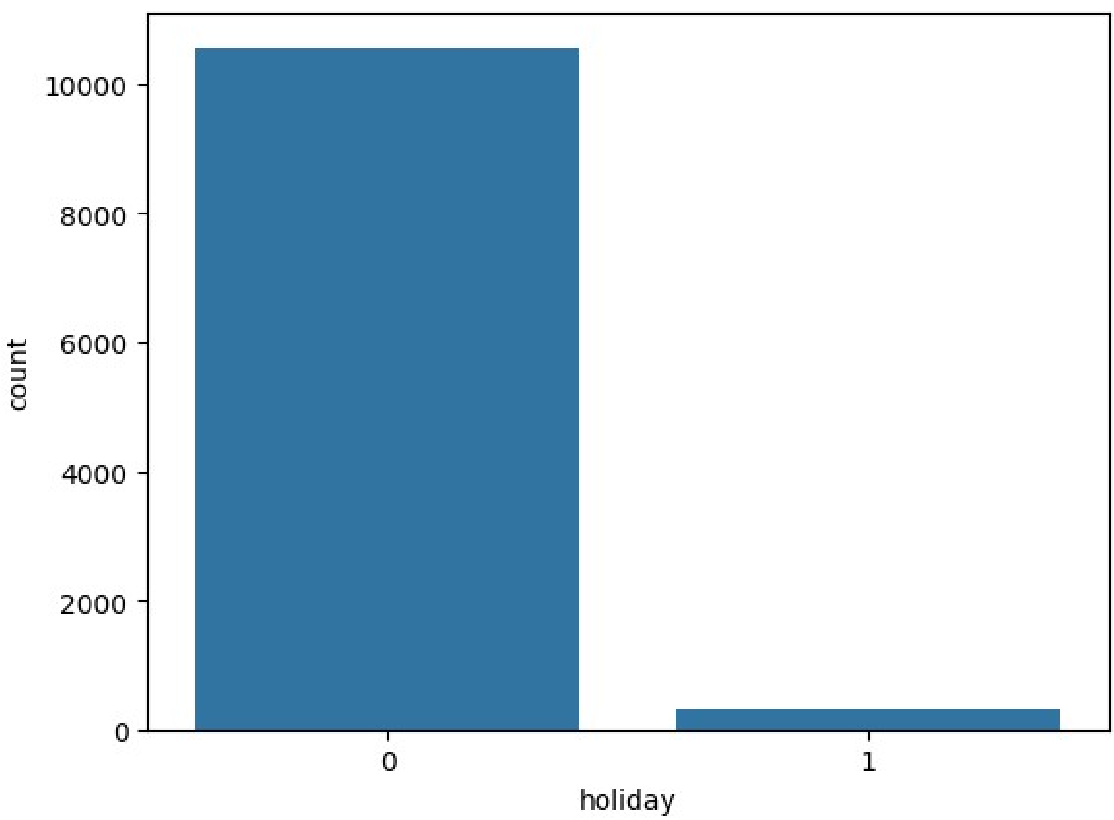

```
{"summary":"{\n  \"name\": \"df[cat_cols]\",\n  \"rows\": 29,\n
\"fields\": [\n    {\n      \"column\": \"value\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
1941,\n        \"min\": 1,\n        \"max\": 7412,\n
\"num_unique_values\": 20,\n        \"samples\": [\n          884,\n
7412,\n          1\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\n}","type":"dataframe"}
```


No duplicates observed

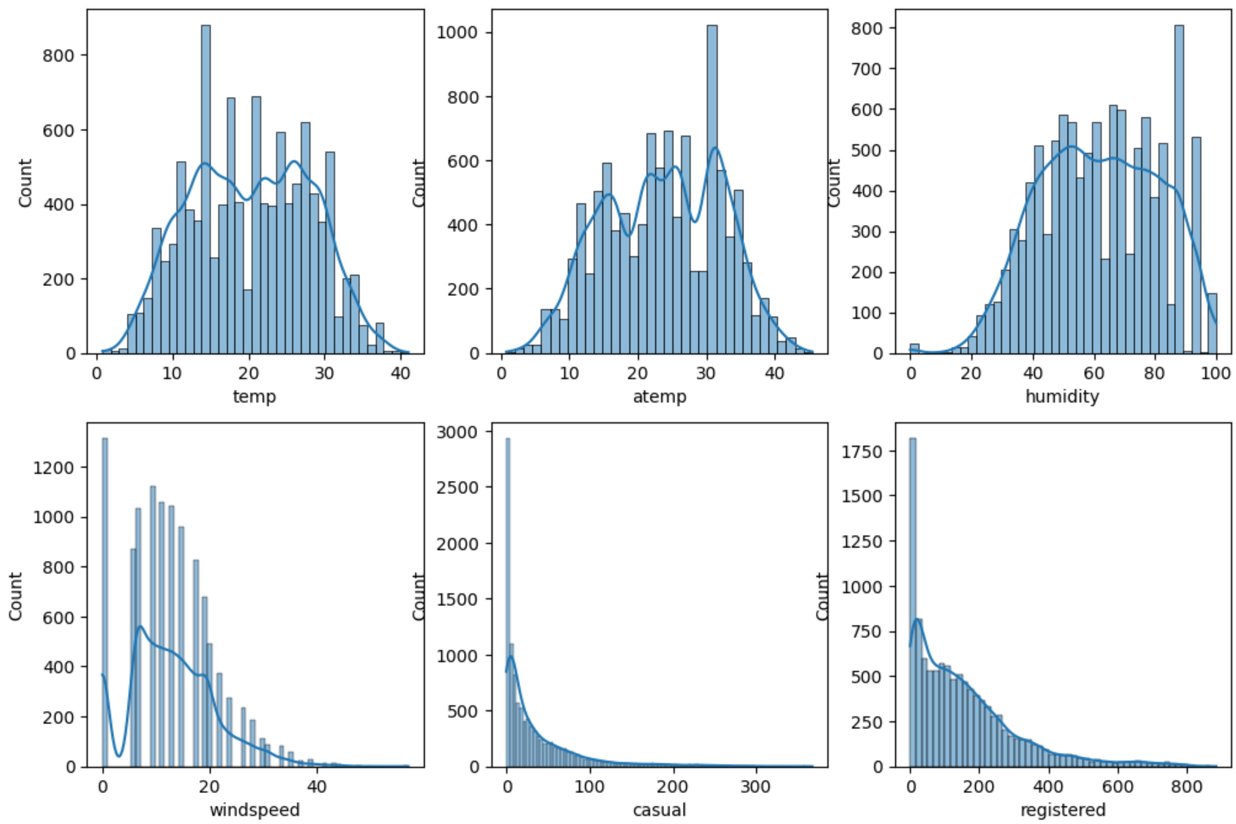
Let's do Graphical analysis :)

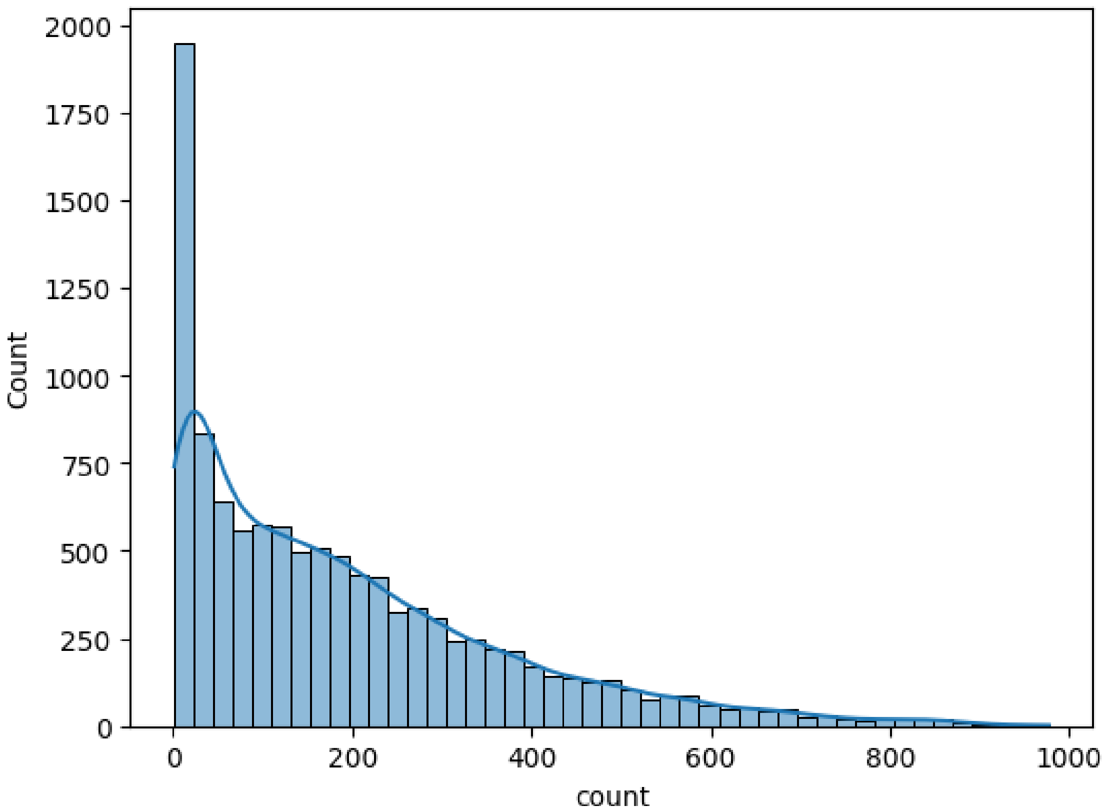
```
cat_cols = ['season', 'workingday', 'weather', 'timeslot', 'year',
'month']
fig, axs = plt.subplots(nrows=2, ncols=3, figsize=(18, 7))
fig.subplots_adjust(top=1.4)
count = 0
for i in range(2):
  for j in range(3):
    sns.countplot(data=df, x= cat_cols[count], ax = axs[i,j])
    axs[i,j].set_title(cat_cols[count])
    count+=1
plt.show()
sns.countplot(data=df, x= 'holiday')
plt.show()
```


season
count
2500
2000
1500
1000
500
0
1
2
3
4
season
workingday
count
7000
6000
5000
4000
3000
2000
1000
0
0
1
workingday
weather
count
7000
6000
5000
4000
3000
2000
1000
0
1
2
3
4
weather
timeslot
count
3000
2500
2000
1500
1000
500
0
Dawn
Early Morning
Noon
Late Evening
Night
timeslot
year
count
5000
4000
3000
2000
1000
0
2011
2012
year
month
count
800
600
400
200
0
1
2
3
4
5
6
7
8
9
10
11
12
month


count
10000
8000
6000
4000
2000
0
0
1
holiday

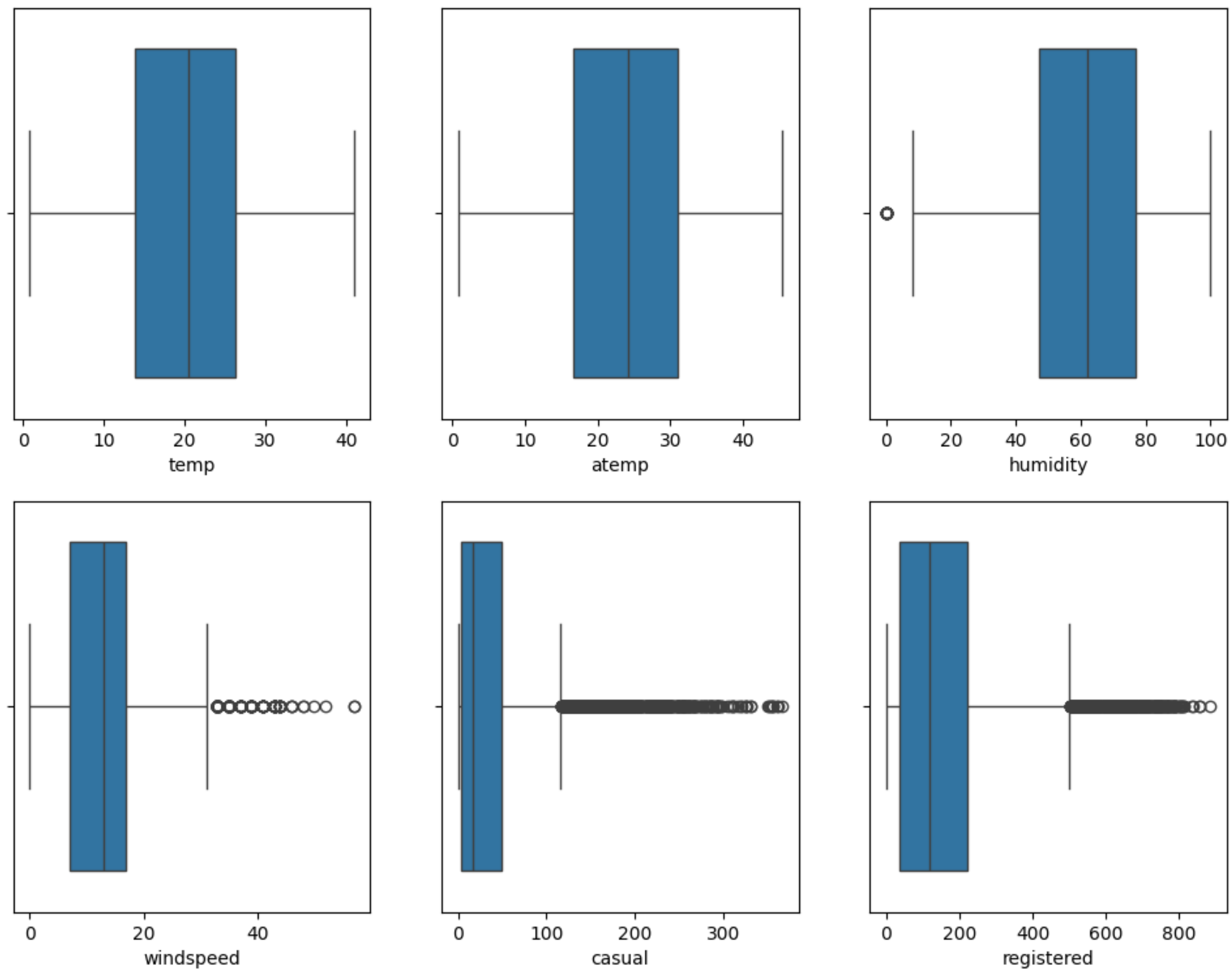
```
# understanding the distribution for numerical variables
num_cols = ['temp', 'atemp', 'humidity', 'windspeed', 'casual',
'registered','count']

fig, axis = plt.subplots(nrows=2, ncols=3, figsize=(12, 8))

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()
```



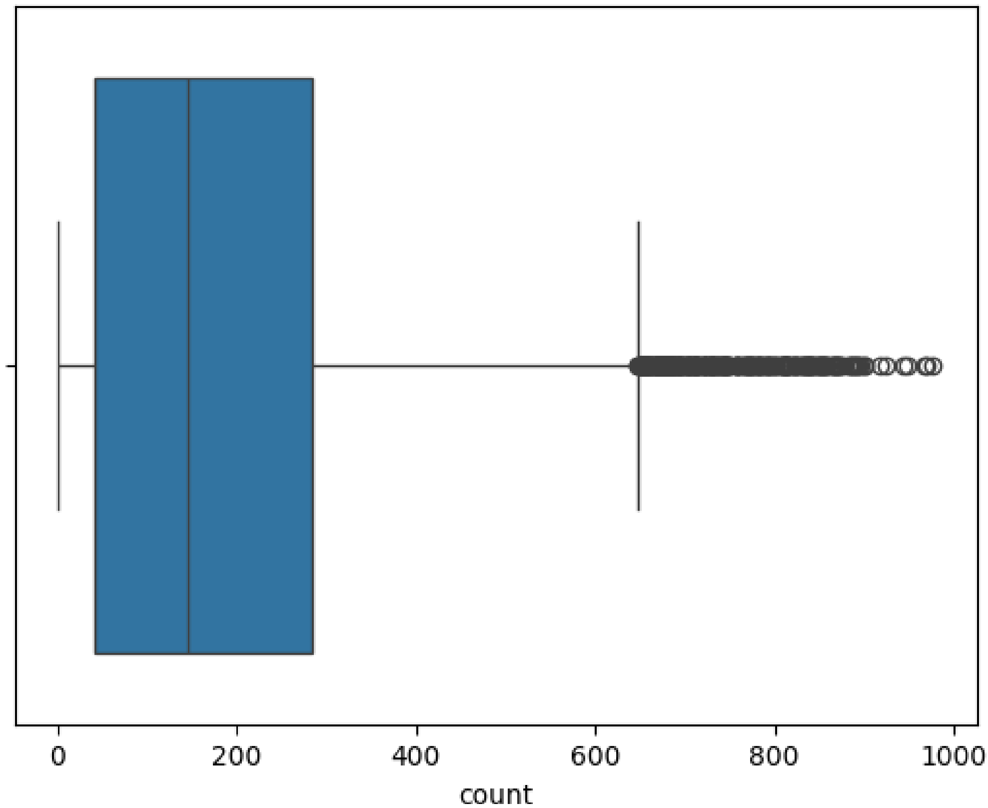


Outlier detection

```
num_cols = ['temp', 'atemp', 'humidity', 'windspeed', 'casual',
'registered','count']
# plotting box plots to detect outliers in the data
fig, axis = plt.subplots(nrows=2, ncols=3, figsize=(12, 9))

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()
```



0
10
20
30
40
temp
0
10
20
30
40
atemp
0
20
40
60
80
100
humidity
0
20
40
windspeed
0
100
200
300
casual
0
200
400
600
800
registered



Seems windspeed, casual, registered and totalcount have some outliers

```
cat_cols = ['season', 'holiday', 'workingday', 'weather', 'timeslot',
'year', 'month']
df[['season', 'holiday','workingday', 'weather', 'timeslot', 'year',
'month']] = df[['season', 'holiday', 'workingday', 'weather',
'timeslot', 'year', 'month']].astype(str)
```

Try establishing a Relationship between the Dependent and Independent Variables

```
df.groupby('season')['count'].sum()

season
1    312498
2    588282
3    640662
4    544034
Name: count, dtype: int64

df.groupby('weather')['count'].sum()

weather
1    1476063
2     507160
3     102089
```

```
4          164
Name: count, dtype: int64
```

```python
df.groupby('workingday')['count'].sum()
```

```
workingday
no      654872
yes    1430604
Name: count, dtype: int64
```

```python
df.groupby('holiday')['count'].sum()
```

```
holiday
no     2027668
yes      57808
Name: count, dtype: int64
```

```python
df.groupby('timeslot')['count'].sum()
```

```
timeslot
Dawn               58642
Early Morning     406571
Late Evening      737257
Night             101727
Noon              781279
Name: count, dtype: int64
```

```python
df.groupby('month')['count'].sum()
```

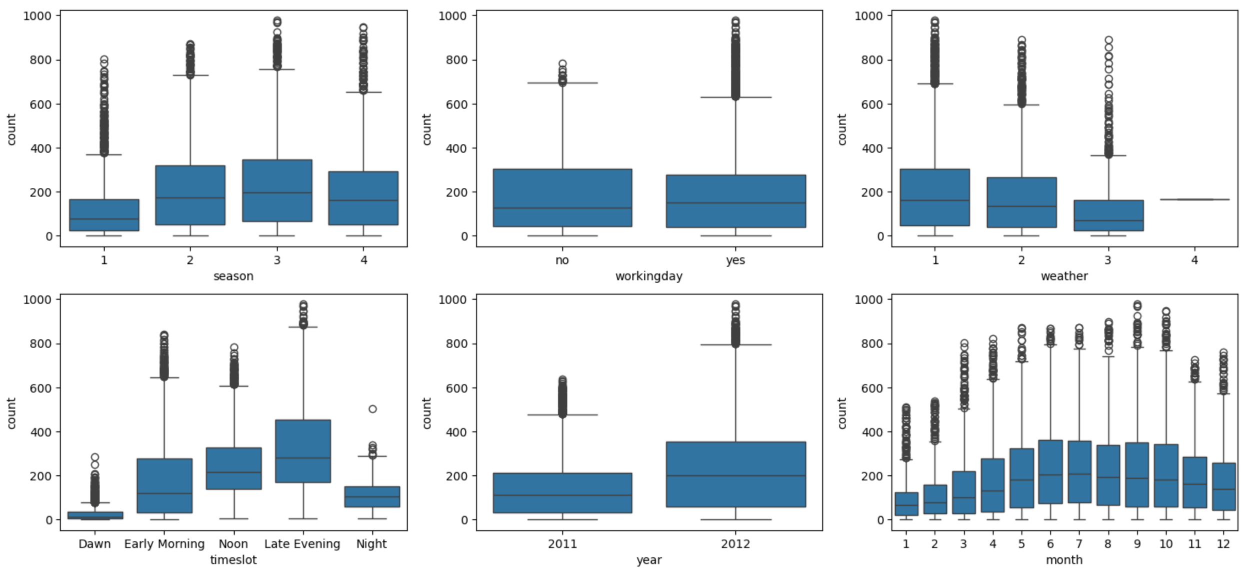
```
month
1      79884
10    207434
11    176440
12    160160
2      99113
3     133501
4     167402
5     200147
6     220733
7     214617
8     213516
9     212529
Name: count, dtype: int64
```

```python
df['season'].value_counts().keys()
```

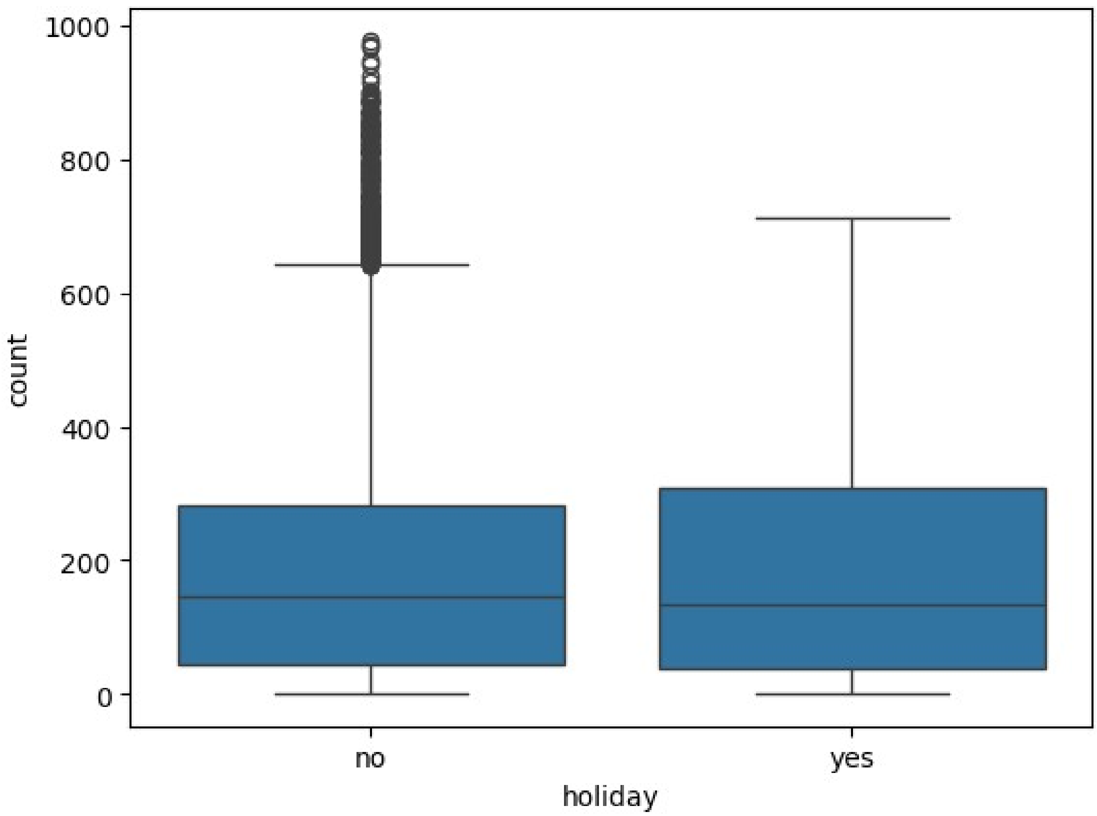
```
Index(['4', '2', '3', '1'], dtype='object')
```

```python
# plotting categorical variables againt count using boxplots
cat_cols = ['season', 'workingday', 'weather', 'timeslot', 'year',
'month']
fig, axis = plt.subplots(nrows=2, ncols=3, figsize=(18, 8))
```

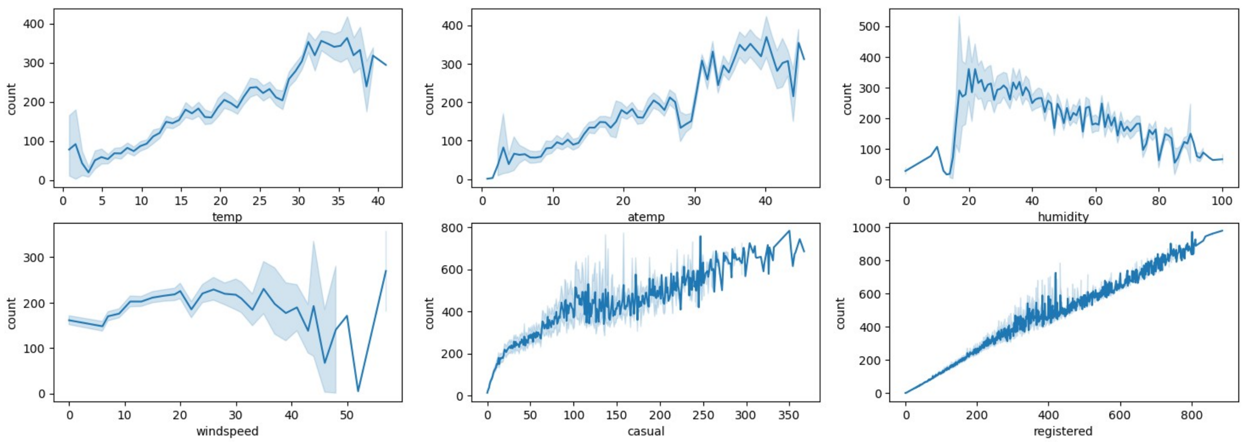
```
index = 0
for row in range(2):
    for col in range(3):
        sns.boxplot(data=df, x=cat_cols[index], y='count',
ax=axis[row, col])
        index += 1

plt.show()
```




```
# Use the value counts of the 'holiday' column as the x-axis values
sns.boxplot(data=df, x= 'holiday', y='count')

<Axes: xlabel='holiday', ylabel='count'>
```

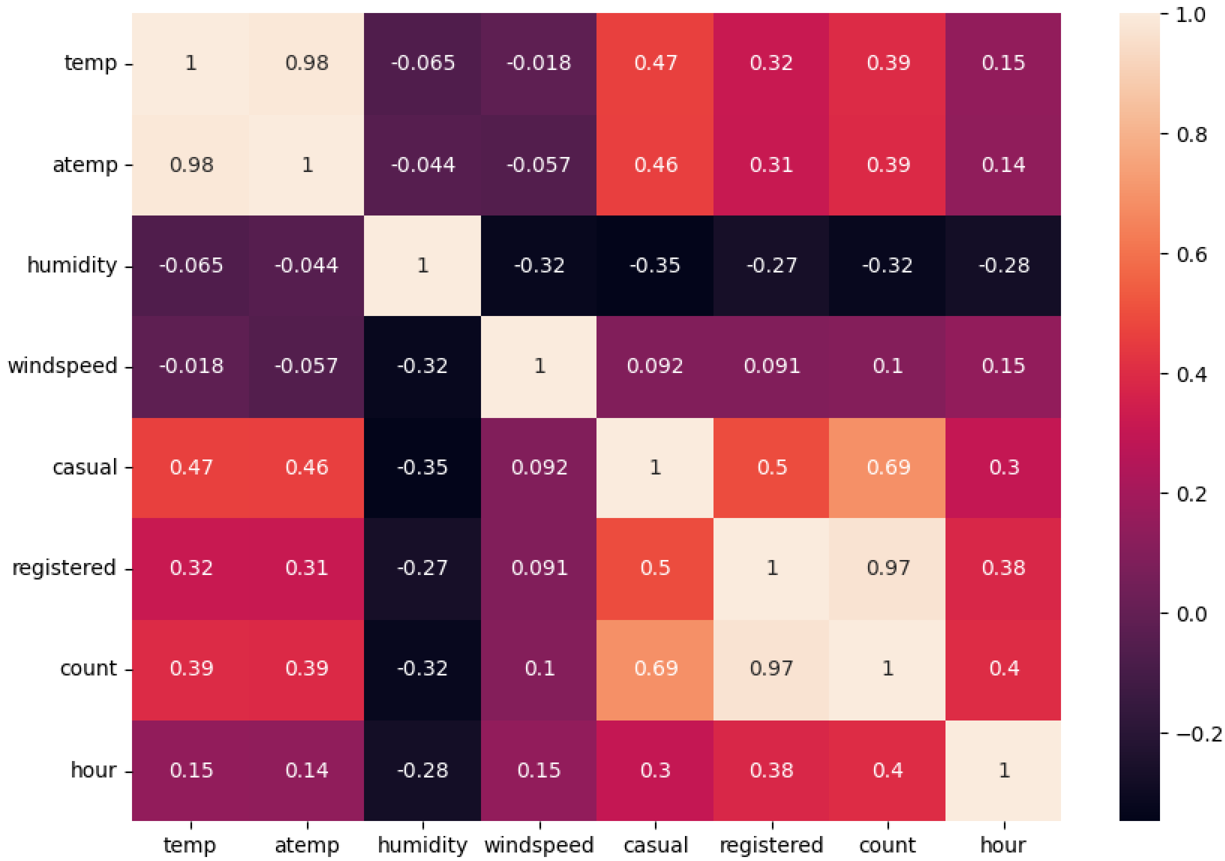



```
# plotting numerical variables againt count using lineplot
fig, axis = plt.subplots(nrows=2, ncols=3, figsize=(18, 6))

index = 0
for row in range(2):
    for col in range(3):
        sns.lineplot(data=df, x=num_cols[index],
y='count',ax=axis[row, col])
        index += 1

plt.show()
```




```
# understanding the correlation between count and numerical variables
plt.figure(figsize = (10,7))
df.corr()['count']
sns.heatmap(df.corr(), annot=True)
plt.show()
```

```
<ipython-input-28-3fa02d5300a5>:3: FutureWarning: The default value of
numeric_only in DataFrame.corr is deprecated. In a future version, it
will default to False. Select only valid columns or specify the value
of numeric_only to silence this warning.
  df.corr()['count']
<ipython-input-28-3fa02d5300a5>:4: FutureWarning: The default value of
numeric_only in DataFrame.corr is deprecated. In a future version, it
will default to False. Select only valid columns or specify the value
of numeric_only to silence this warning.
  sns.heatmap(df.corr(), annot=True)
```



Check if the Weather conditions are significantly different during different Seasons?

```
#weather vs season chisquare test
data_table = pd.crosstab(df['season'], df['weather'], margins = True)
print("Observed values:")
data_table
```

```
Observed values:
```

{"summary":"{\n \"name\": \"data_table\",\n \"rows\": 5,\n \"fields\": [\n   {\n   \"column\": \"1\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 2413,\n   \"min\": 1702,\n   \"max\": 7192,\n   \"num_unique_values\": 5,\n   \"samples\": [\n   1801,\n   7192,\n   1930\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"2\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 953,\n   \"min\": 604,\n   \"max\": 2834,\n

\"num_unique_values\": 5,\n   \"samples\": [\n   708,\n
2834,\n   604\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"3\",\n   \"properties\": {\n   \"dtype\": \"number\",\n
\"std\": 288,\n   \"min\": 199,\n   \"max\": 859,\n
\"num_unique_values\": 5,\n   \"samples\": [\n   224,\n
859,\n   199\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"4\",\n   \"properties\": {\n   \"dtype\": \"number\",\n
\"std\": 0,\n   \"min\": 0,\n   \"max\": 1,\n
\"num_unique_values\": 2,\n   \"samples\": [\n   0,\n
1\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"All\",\n   \"properties\": {\n   \"dtype\": \"number\",\n
\"std\": 3651,\n   \"min\": 2686,\n   \"max\": 10886,\n
\"num_unique_values\": 4,\n   \"samples\": [\n   2733,\n
10886\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   }\n]\
n}","type":"dataframe","variable_name":"data_table"}

```
vals = chi2_contingency(data_table)
vals
```

```
Chi2ContingencyResult(statistic=49.15865559689363,
pvalue=3.1185273325126814e-05, dof=16,
expected_freq=array([[1.77454639e+03, 6.99258130e+02, 2.11948742e+02,
2.46738931e-01,
        2.68600000e+03],
       [1.80559765e+03, 7.11493845e+02, 2.15657450e+02, 2.51056403e-
01,
        2.73300000e+03],
       [1.80559765e+03, 7.11493845e+02, 2.15657450e+02, 2.51056403e-
01,
        2.73300000e+03],
       [1.80625831e+03, 7.11754180e+02, 2.15736359e+02, 2.51148264e-
01,
        2.73400000e+03],
       [7.19200000e+03, 2.83400000e+03, 8.59000000e+02,
1.00000000e+00,
        1.08860000e+04]]))
```

as p_val is less than alpha (0.05), we can reject the H0- both season nd weather are associated.

```
df.dtypes
```

```
datetime      datetime64[ns]
season                object
holiday               object
workingday            object
weather               object
```

```
temp                     float64
atemp                    float64
humidity                   int64
windspeed                float64
casual                     int64
registered                 int64
count                      int64
hour                       int64
timeslot                  object
month                     object
year                      object
dtype: object

df = df.dropna()
```

Check if there any significant difference between the no. of bike rides on Weekdays and Weekends?

```
df['workingday'] = pd.to_numeric(df['workingday'])
df['workingday'] = df['workingday'].replace({0 : 'no', 1 : 'yes'})
df.head()

----------------------------------------------------------------------
-----
ValueError                                Traceback (most recent call
last)
/usr/local/lib/python3.10/dist-packages/pandas/_libs/lib.pyx in
pandas._libs.lib.maybe_convert_numeric()

ValueError: Unable to parse string "no"

During handling of the above exception, another exception occurred:

ValueError                                Traceback (most recent call
last)
<ipython-input-46-f5801fc963cc> in <cell line: 1>()
----> 1 df['workingday'] = pd.to_numeric(df['workingday'])
      2 df['workingday'] = df['workingday'].replace({0 : 'no', 1 :
'yes'})
      3 df.head()

/usr/local/lib/python3.10/dist-packages/pandas/core/tools/numeric.py
in to_numeric(arg, errors, downcast)
    183         coerce_numeric = errors not in ("ignore", "raise")
    184         try:
```

```
--> 185             values, _ = lib.maybe_convert_numeric(
    186                 values, set(), coerce_numeric=coerce_numeric
    187             )

/usr/local/lib/python3.10/dist-packages/pandas/_libs/lib.pyx in
pandas._libs.lib.maybe_convert_numeric()

ValueError: Unable to parse string "no" at position 0
```

```
data_group1 = df[df['workingday']=='no']['count'].values
data_group2 = df[df['workingday']=='yes']['count'].values
data_group1
```

```
array([ 16,  40,  32, ..., 106,  89,  33])
```

```
#Before conducting the two-sample T-Test we need to find if the given
data groups have the same variance. If the ratio of the larger data
groups to the small data group is less than 4:1 then
#we can consider that the given data groups have equal variance.
print(np.var(data_group1), np.var(data_group2))
np.var(data_group2)// np.var(data_group1)
```

```
30171.346098942427 34040.69710674686
```

```
1.0
```

```
stats.ttest_ind(data_group1, data_group2)
```

```
TtestResult(statistic=-1.2096277376026694, pvalue=0.22644804226361348,
df=10884.0)
```

Since p-value is greater than 0.05 so we cannot reject the Null hypothesis. We don't have the sufficient evidence to say that working day has effect on the number of cycles being rented.

```
df['holiday'] = pd.to_numeric(df['holiday'])
df['holiday'] = df['holiday'].replace({0 : 'no', 1 : 'yes'})
df.head()
```


```
{"repr_error":"'str' object has no attribute
'empty'","type":"dataframe","variable_name":"df"}
```


```
data_group3 = df[df['holiday']=='no']['count'].values
data_group4 = df[df['holiday']=='yes']['count'].values
data_group3
```

```
array([ 16,  40,  32, ..., 168, 129,  88])
```

```
print(np.var(data_group3), np.var(data_group4))
np.var(data_group3)// np.var(data_group4)
```

```
32943.901106481346 28233.99150132856
```

```
1.0
```

```
stats.ttest_ind(data_group3, data_group4)
```

```
TtestResult(statistic=0.5626388963477119, pvalue=0.5736923883271103,
df=10884.0)
```

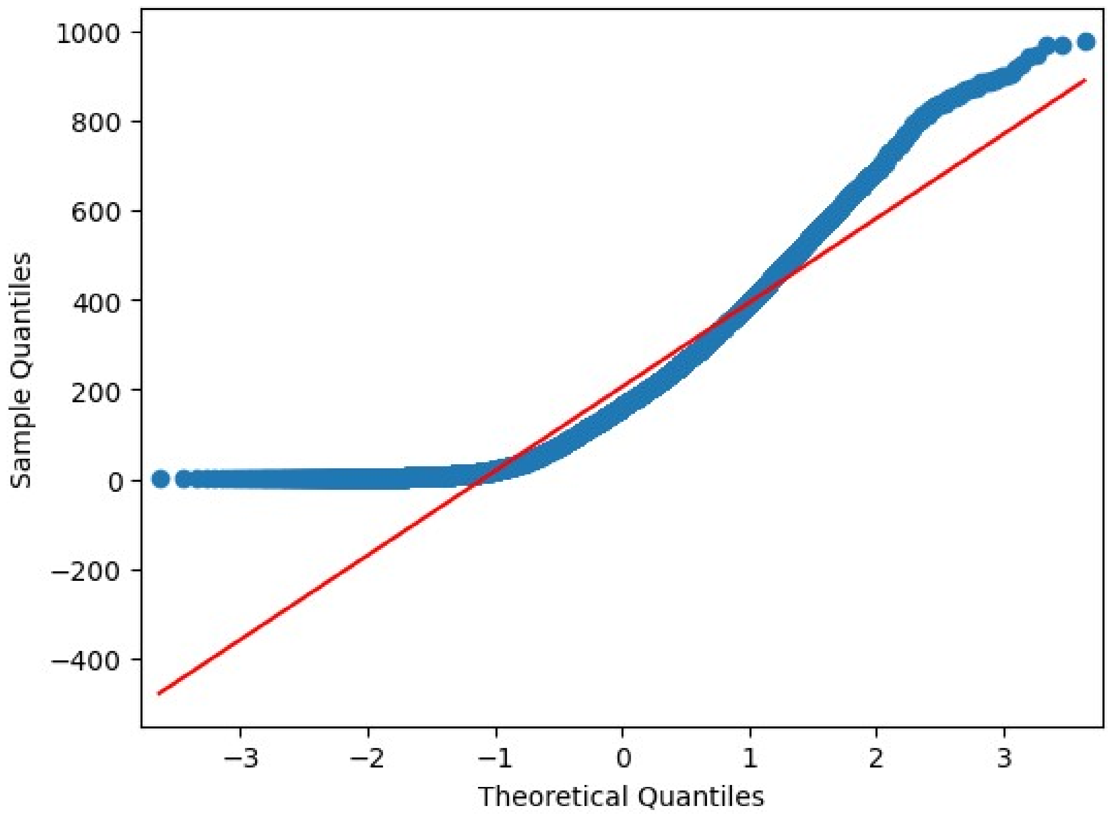
Since p-value is greater than 0.05 so we cannot reject the Null hypothesis. We don't have the sufficient evidence to say that holiday has effect on the number of cycles being rented.

Check if the demand of bicycles on rent is the same for different Weather conditions?

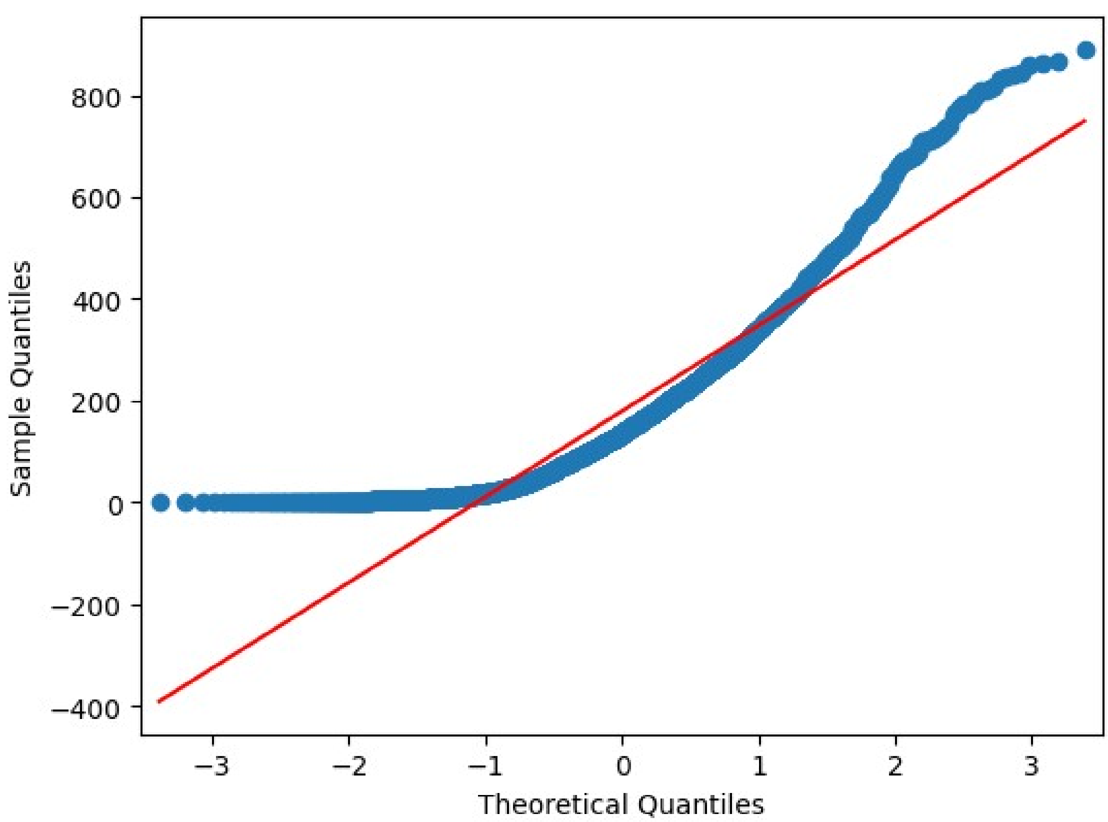
```
# defining the data groups for the ANOVA
df['weather'] = pd.to_numeric(df['weather'])
df['season'] = pd.to_numeric(df['season'])

gp1 = df[df['weather']==1]['count'].values
gp2 = df[df['weather']==2]['count'].values
gp3 = df[df['weather']==3]['count'].values
gp4 = df[df['weather']==4]['count'].values

gp5 = df[df['season']==1]['count'].values
gp6 = df[df['season']==2]['count'].values
gp7 = df[df['season']==3]['count'].values
gp8 = df[df['season']==4]['count'].values
groups=[gp1,gp2,gp3,gp4,gp5,gp6,gp7,gp8]
gp1
```

```
array([ 16,  40,  32, ..., 168, 129,  88])
```

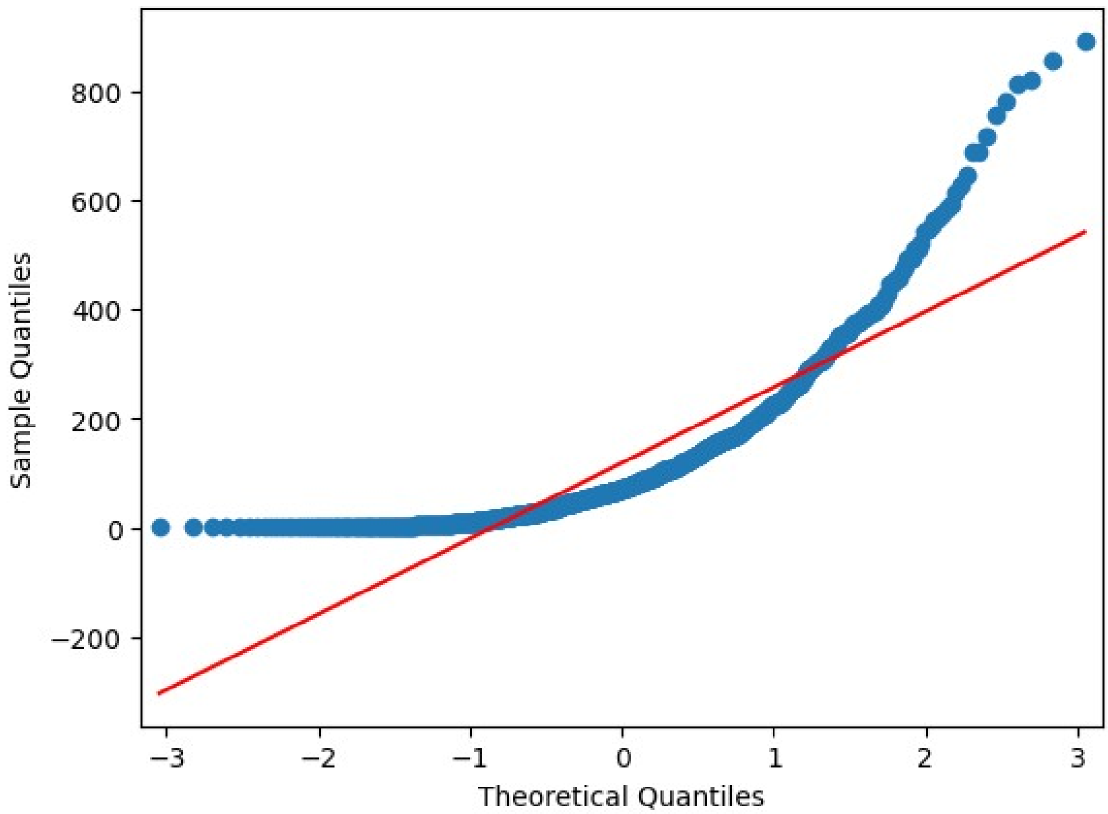
```
#lets  checking the assumptions first
#1 normality using qqplot
from statsmodels.graphics.gofplots import qqplot
index = 0
for row in range(4):
    for col in range(2):
        qqplot(groups[index], line="s")
        index += 1

plt.show()
```

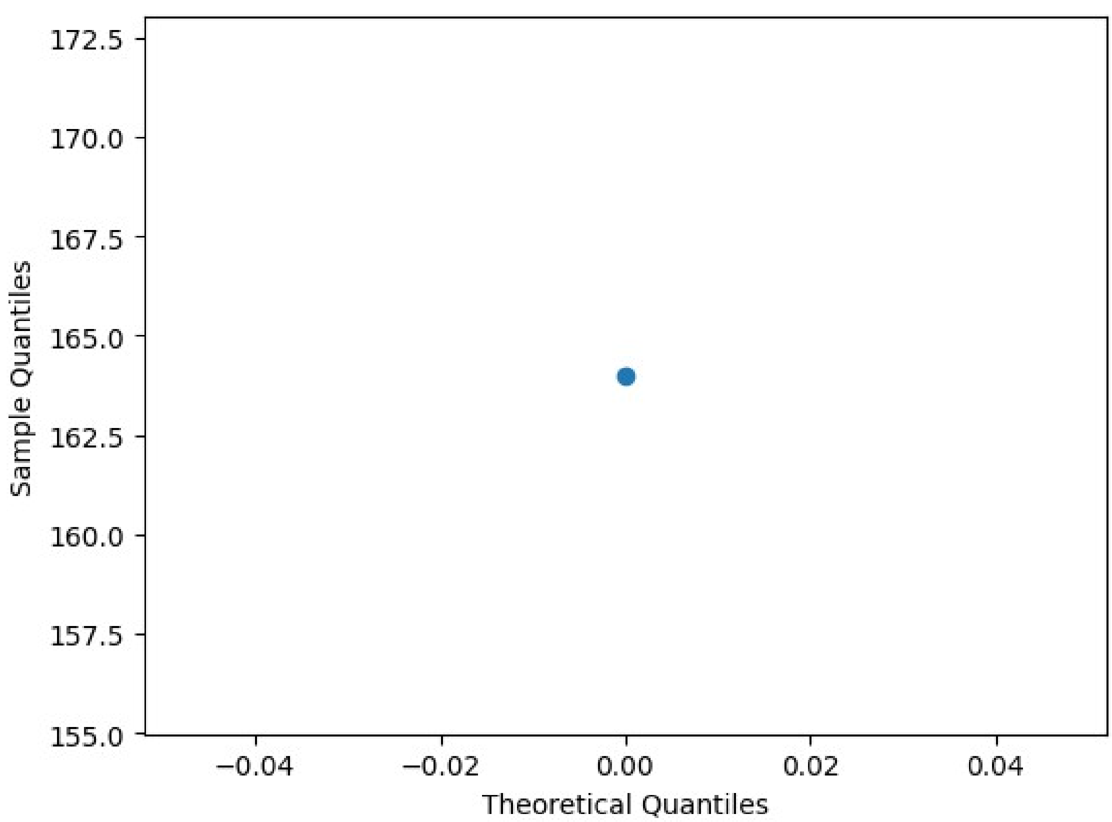


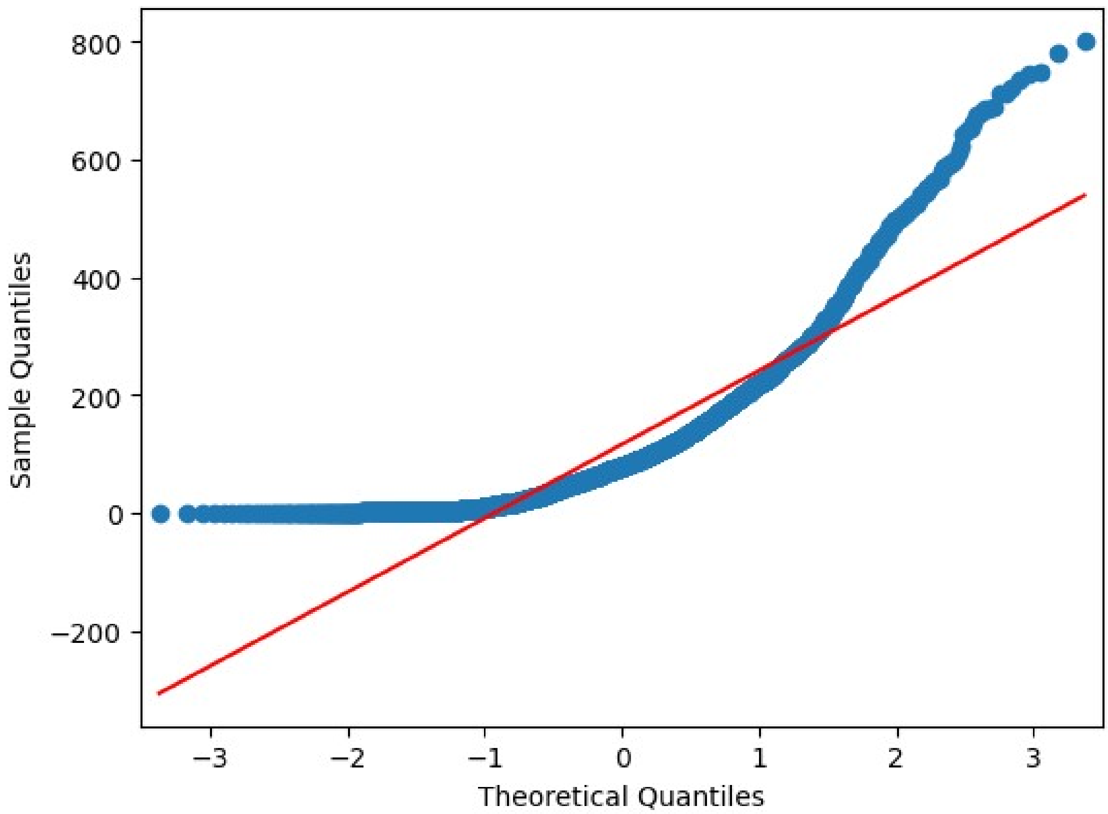
1000
800
600
400
200
0
−200
−400
Sample Quantiles
−3
−2
−1
0
1
2
3
Theoretical Quantiles

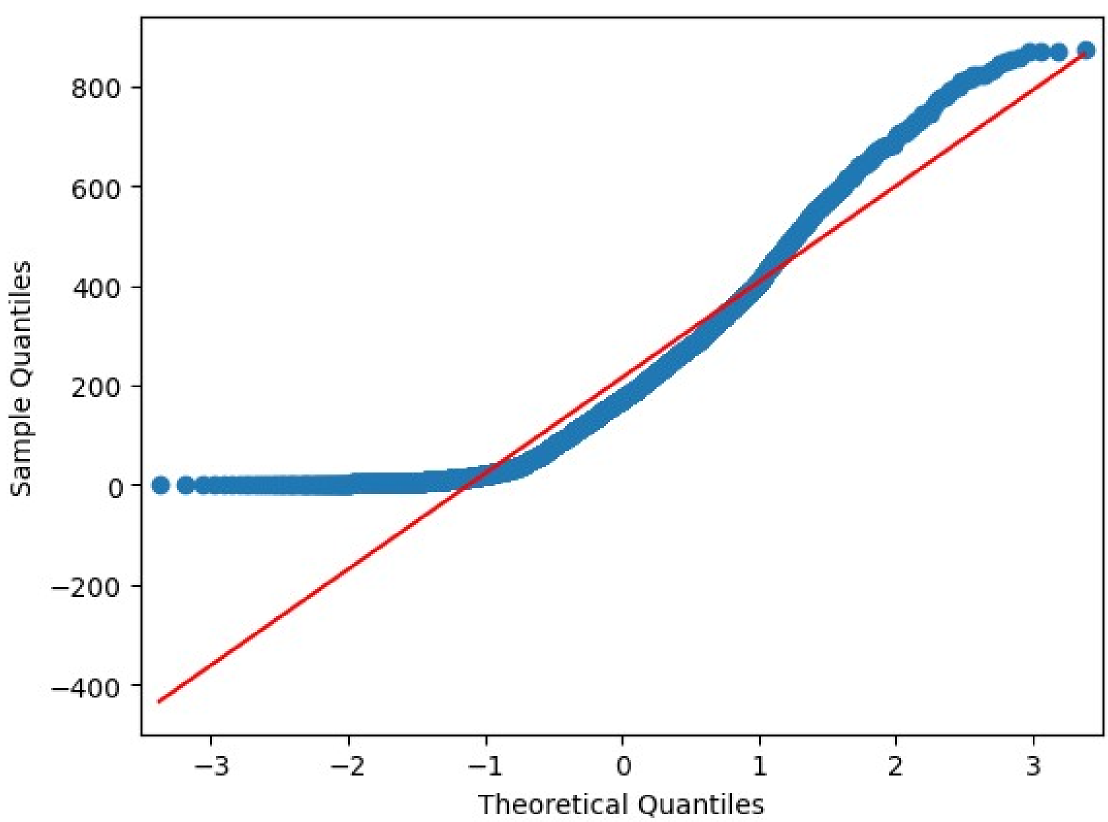


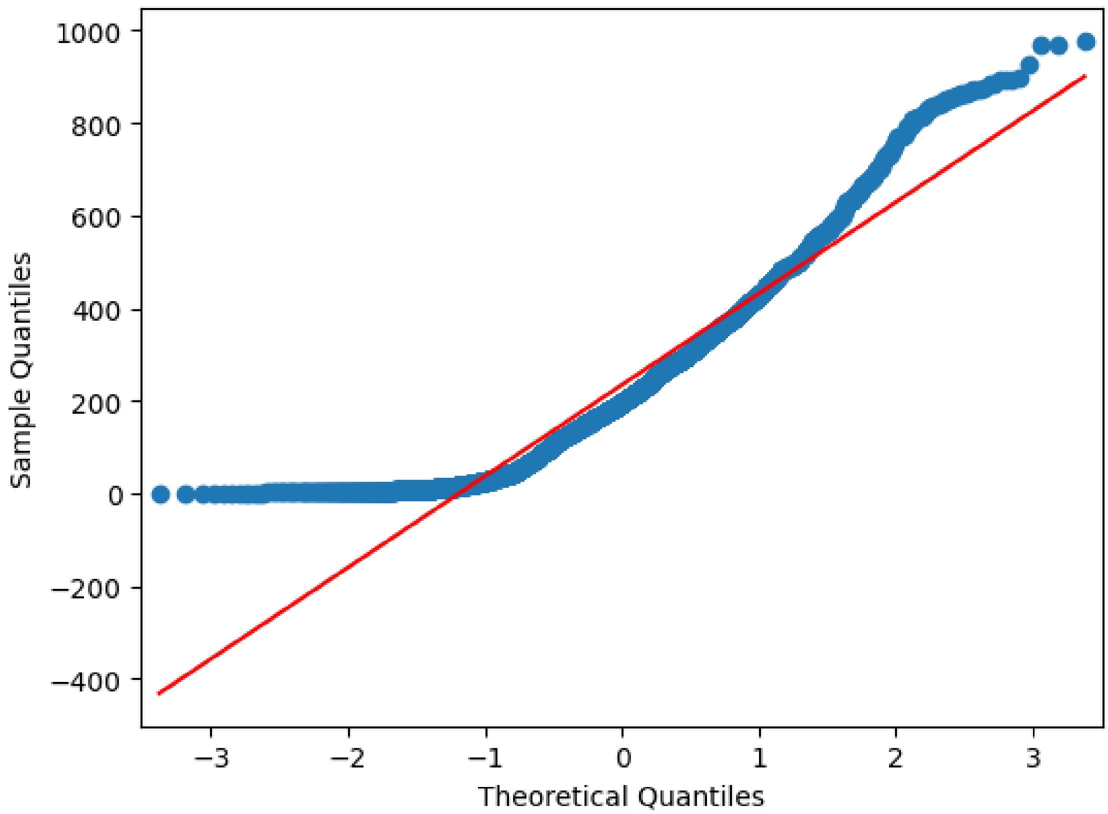
800
600
400
200
0
−200
−400
Sample Quantiles
−3
−2
−1
0
1
2
3
Theoretical Quantiles

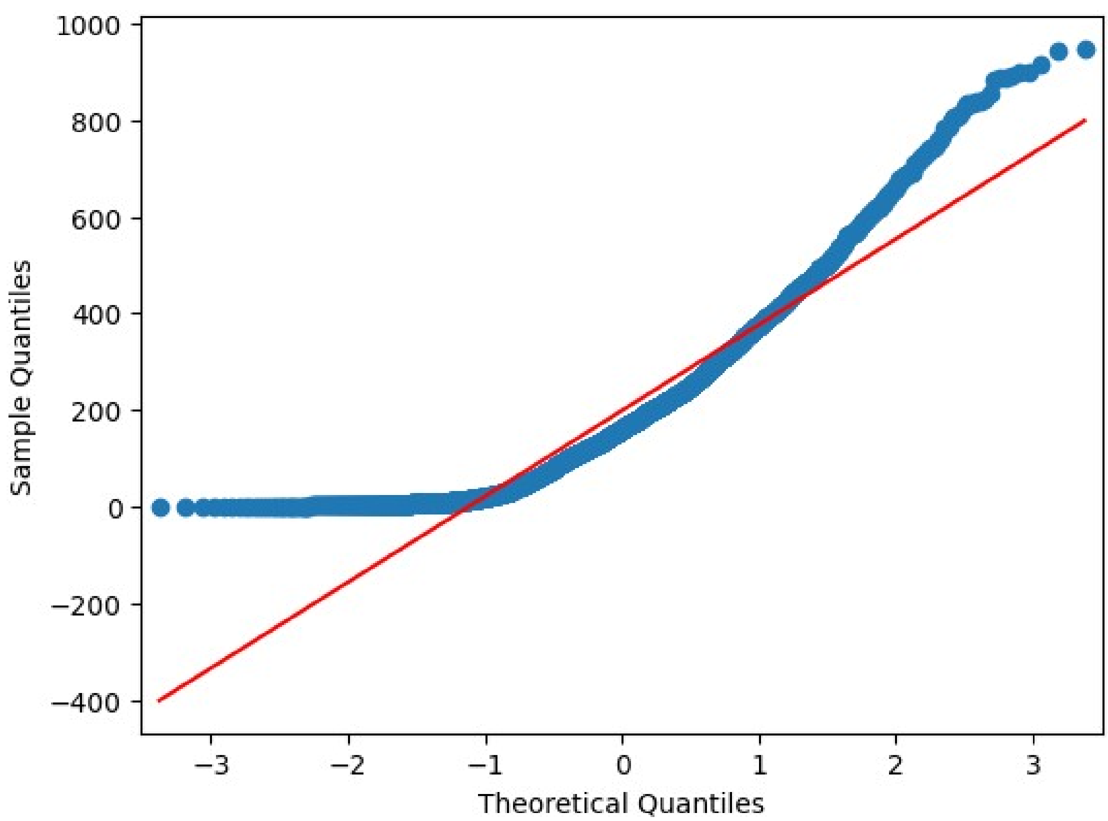

Sample Quantiles
Theoretical Quantiles
800
600
400
200
0
−200
−3
−2
−1
0
1
2
3


Sample Quantiles
Theoretical Quantiles
172.5
170.0
167.5
165.0
162.5
160.0
157.5
155.0
−0.04
−0.02
0.00
0.02
0.04


800
600
400
200
0
−200
Sample Quantiles
−3
−2
−1
0
1
2
3
Theoretical Quantiles


800
600
400
200
0
−200
−400
Sample Quantiles
−3
−2
−1
0
1
2
3
Theoretical Quantiles


1000
800
600
400
200
0
−200
−400
Sample Quantiles
−3
−2
−1
0
1
2
3
Theoretical Quantiles


1000
800
600
400
200
0
−200
−400
Sample Quantiles
−3
−2
−1
0
1
2
3
Theoretical Quantiles

Data is not normally distributed

```
#2 checking variance
#Null Hypothesis: Variances is similar in different weather and
season.

#Alternate Hypothesis: Variances is not similar in different weather
and season.

#Significance level (alpha): 0.05
levene_stat, p_value = stats.levene(gp1,gp2,gp3,gp4,gp5,gp6,gp7,gp8)
print(p_value)
if p_value < 0.05:
    print("Reject the Null hypothesis.Variances are not equal")
else:
  print("Fail to Reject the Null hypothesis.Variances are  equal")
```

```
3.463531888897594e-148
Reject the Null hypothesis.Variances are not equal
```

```
#kruskal wallis test isto be perfomed as assumptions for ANOVA are not
true
from scipy.stats import kruskal
kruskal(gp1,gp2,gp3,gp4)
```

```
KruskalResult(statistic=205.00216514479087, pvalue=3.501611300708679e-
44)
```

p_val is less than alpha, we cn reject the H0 - weather has significant effect on bike rides

Check if the demand of bicycles on rent is the same for different Seasons?

```
kruskal(gp5,gp6,gp7,gp8)
```

```
KruskalResult(statistic=699.6668548181988, pvalue=2.479008372608633e-
151)
```

p_val is less than alpha, we cn reject the H0 - season has significant effect on bike rides

Insights

1. In summer and fall seasons more bikes are rented as compared to other seasons.
2. Whenever its a holiday more bikes are rented.
3. Whenever there is rain, thunderstorm, snow or fog, there were less bikes were rented.
4. Whenever the humidity is less than 20, number of bikes rented is very low.

5. Whenever the temperature is less than 10, number of bikes rented is less.
6. Whenever the windspeed is greater than 35, number of bikes rented is less.

Recommendations

In summer and fall seasons the company should have more bikes in stock to be rented. Because the demand in these seasons is higher as compared to other seasons. With a significance level of 0.05, workingday has no effect on the number of bikes being rented. In very low humid days, company should have less bikes in the stock to be rented. Whenever temperature is less than 10 or in very cold days, company should have less bikes. Whenever the windspeed is greater than 35 or in thunderstorms, company should have less bikes in stock to be rented.