9lh95wcgk

June 19, 2024

#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.

[ ]:
```
#importing all necessary libraries
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from scipy.stats import ttest_ind # T-test for independent samples
from scipy.stats import f_oneway # One-way ANOVA
from scipy.stats import chi2_contingency # Chi-square test of independence
from scipy.stats import shapiro # Shapiro-Wilk’s test for Normality
from scipy.stats import levene # Levene's test for Equality of Variance
from scipy.stats import kruskal # Kruskal-Wallis test for comparing more than 2
 ↪samples
from statsmodels.formula.api import ols # Ordinary Least Squares
from statsmodels.stats.anova import anova_lm # two-way ANOVA
```

#Importing the dataset and doing usual exploratory data analysis steps

[ ]:
```
import warnings
warnings.simplefilter('ignore')
```

[ ]:
```
#Load and Read data
df=pd.read_csv('/content/Yulu - Hypothesis Testing.txt')
df.head()
```

```
[ ]:               datetime  season  holiday  workingday  weather  temp   atemp  \
     0  2011-01-01 00:00:00       1        0           0        1  9.84  14.395
     1  2011-01-01 01:00:00       1        0           0        1  9.02  13.635
     2  2011-01-01 02:00:00       1        0           0        1  9.02  13.635
     3  2011-01-01 03:00:00       1        0           0        1  9.84  14.395
     4  2011-01-01 04:00:00       1        0           0        1  9.84  14.395

        humidity  windspeed  casual  registered  count
     0        81        0.0       3          13     16
     1        80        0.0       8          32     40
     2        80        0.0       5          27     32
     3        75        0.0       3          10     13
     4        75        0.0       0           1      1
```

```
[ ]: print('Total Rows :',df.shape[0])
     print('Total Columns :',df.shape[1])
```

```
Total Rows : 10886
Total Columns : 12
```

```
[ ]: df[df.duplicated()]
```

```
[ ]: Empty DataFrame
     Columns: [datetime, season, holiday, workingday, weather, temp, atemp, humidity,
     windspeed, casual, registered, count]
     Index: []
```

```
[ ]: 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
```

[ ]:
```python
print('Total Null values :',df.isna().sum().sum())
```

```
Total Null values : 0
```

[ ]:
```python
df.describe()
```

[ ]:
```
             season       holiday    workingday       weather         temp  \
count  10886.000000  10886.000000  10886.000000  10886.000000  10886.00000
mean       2.506614      0.028569      0.680875      1.418427     20.23086
std        1.116174      0.166599      0.466159      0.633839      7.79159
min        1.000000      0.000000      0.000000      1.000000      0.82000
25%        2.000000      0.000000      0.000000      1.000000     13.94000
50%        3.000000      0.000000      1.000000      1.000000     20.50000
75%        4.000000      0.000000      1.000000      2.000000     26.24000
max        4.000000      1.000000      1.000000      4.000000     41.00000

              atemp      humidity     windspeed        casual    registered  \
count  10886.000000  10886.000000  10886.000000  10886.000000  10886.000000
mean      23.655084     61.886460     12.799395     36.021955    155.552177
std        8.474601     19.245033      8.164537     49.960477    151.039033
min        0.760000      0.000000      0.000000      0.000000      0.000000
25%       16.665000     47.000000      7.001500      4.000000     36.000000
50%       24.240000     62.000000     12.998000     17.000000    118.000000
75%       31.060000     77.000000     16.997900     49.000000    222.000000
max       45.455000    100.000000     56.996900    367.000000    886.000000

              count
count  10886.000000
mean     191.574132
std      181.144454
min        1.000000
25%       42.000000
50%      145.000000
75%      284.000000
max      977.000000
```

[ ]:
```python
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
```

[ ]:
```
print('season :' , df['season'].unique())
print('holiday :' , df['holiday'].unique())
print('workingday :' , df['workingday'].unique())
print('weather :' , df['weather'].unique())
```

```
season : [1 2 3 4]
holiday : [0 1]
workingday : [0 1]
weather : [1 2 3 4]
```

Insights:

1. The dataset contains no missing values or duplicates, ensuring data integrity.
2. Season, Holiday, Workingday, and Weather are categorical variables with a few distinct values, suitable for analyzing categorical impacts on bike rentals.
3. The categorical columns already have numerical values, so there's no need to convert them; numerical data types are more suitable for analysis.
4. The datetime column can be leveraged to analyze trends and patterns over time, such as peak rental times and seasonal variations.
5. The remaining columns are continuous variables, suitable for analyzing correlations.

#Univariate Analysis

[ ]:
```
df[['temp','atemp', 'humidity', 'windspeed', 'casual', 'registered', 'count']].
 ↪skew()
```

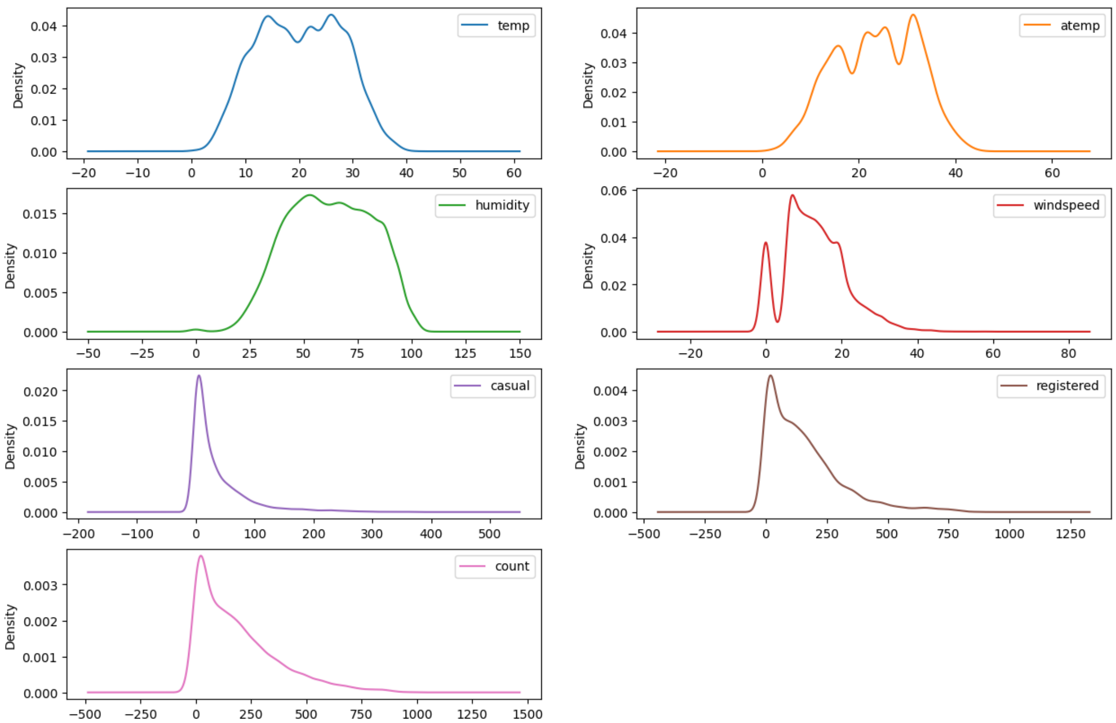
[ ]:
```
temp          0.003691
atemp        -0.102560
humidity     -0.086335
windspeed     0.588767
casual        2.495748
registered    1.524805
count         1.242066
dtype: float64
```

[ ]:
```
df[['temp','atemp', 'humidity', 'windspeed', 'casual', 'registered', 'count']].
 ↪kurt()
```

[ ]:
```
temp         -0.914530
atemp        -0.850076
humidity     -0.759818
windspeed     0.630133
casual        7.551629
```

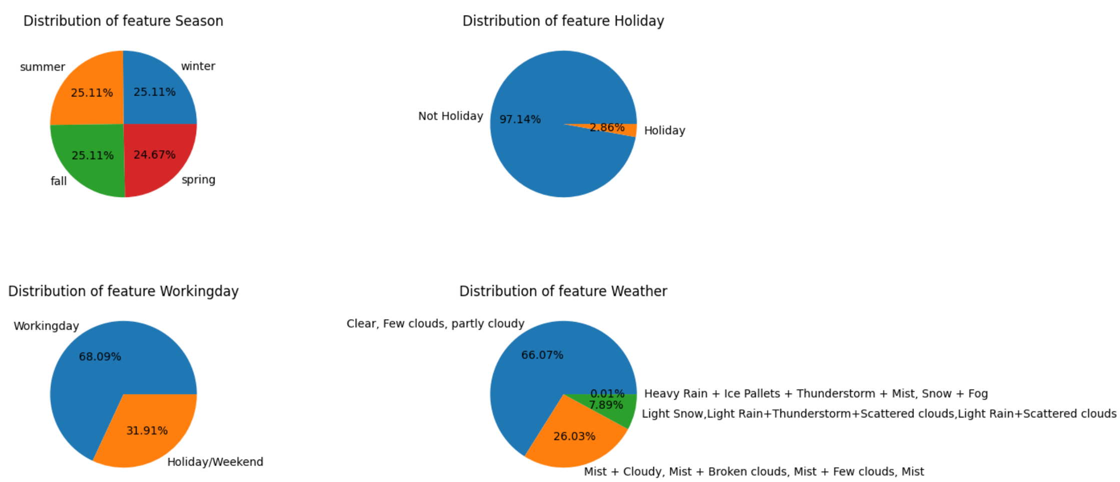
```
registered    2.626081
count         1.300093
dtype: float64
```

[ ]:
```
#kde plot for continuous variables
plt.rcParams["figure.figsize"] = [15, 10]
col = ['temp','atemp', 'humidity', 'windspeed', 'casual', 'registered', 'count']
df[col].plot(kind='density', subplots=True, layout=(4,2), sharex=False)
plt.show()
```



[ ]:
```
plt.figure(figsize=(10,8))
plt.subplot(2,3,1)
plt.title('Distribution of feature Season')
season_df = df['season'].value_counts().reset_index()
season_mapping = {1: 'spring', 2: 'summer', 3: 'fall', 4: 'winter'}
season_df['season'] = season_df['season'].map(season_mapping)
plt.pie(season_df['count'], labels =season_df['season'], autopct='%1.2f%%')

plt.subplot(2,3,3)
plt.title('Distribution of feature Holiday')
holiday_df = df['holiday'].value_counts().reset_index()
holiday_mapping = {0:'Not Holiday', 1:'Holiday'}
holiday_df['holiday'] = holiday_df['holiday'].map(holiday_mapping)
plt.pie(holiday_df['count'], labels =holiday_df['holiday'], autopct='%1.2f%%')
```

```
plt.subplot(2,3,4)
plt.title('Distribution of feature Workingday')
workingday_df = df['workingday'].value_counts().reset_index()
workingday_mapping = {0:'Holiday/Weekend', 1:'Workingday'}
workingday_df['workingday'] = workingday_df['workingday'].map(workingday_mapping)
plt.pie(workingday_df['count'], labels =workingday_df['workingday'], autopct='%1.2f%%')

plt.subplot(2,3,6)
plt.title('Distribution of feature Weather')
weather_df = df['weather'].value_counts().reset_index()
weather_mapping = {1:'Clear, Few clouds, partly cloudy',
                   2:'Mist + Cloudy, Mist + Broken clouds, Mist + Few clouds, Mist',
                   3:'Light Snow,Light Rain+Thunderstorm+Scattered clouds,Light Rain+Scattered clouds',
                   4:'Heavy Rain + Ice Pallets + Thunderstorm + Mist, Snow + Fog'
                  }
weather_df['weather'] = weather_df['weather'].map(weather_mapping)
plt.pie(weather_df['count'], labels =weather_df['weather'], autopct='%1.2f%%')

plt.show()
```



###Insights:

Column Name	Skewness Label	Kurtosis Label
temp	Symmetric	Platykurtic

Column Name	Skewness Label	Kurtosis Label
atemp	Negative Skew	Platykurtic
humidity	Negative Skew	Platykurtic
windspeed	Moderate Positive Skew	Leptokurtic
casual	High Positive Skew	Leptokurtic
registered	Moderate Positive Skew	Leptokurtic
count	Moderate Positive Skew	Leptokurtic

- The dataset is evenly distributed across the seasons, with each season around 25%.
- 97.14% of days are not holidays, 2.86% are holidays (column: holiday) and 68.09% are working days, while 31.91% are holidays or weekends (column: workingday), indicating dependency.
- Weather data is unevenly distributed:
  - 66.07% clear/few clouds/partly cloudy
  - 26.03% mist/cloudy
  - 7.89% light snow/light rain/thunderstorm/scattered clouds
  - 0.01% heavy rain/ice pellets/thunderstorm/mist or snow/fog (1 data point, unsuitable for hypothesis testing.

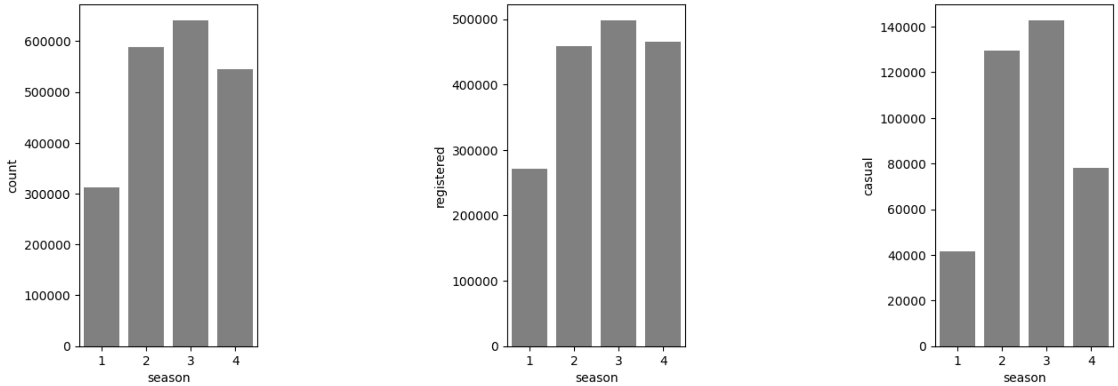
###Analysis Overview

Target Variables: Count, Casual, and Registered.

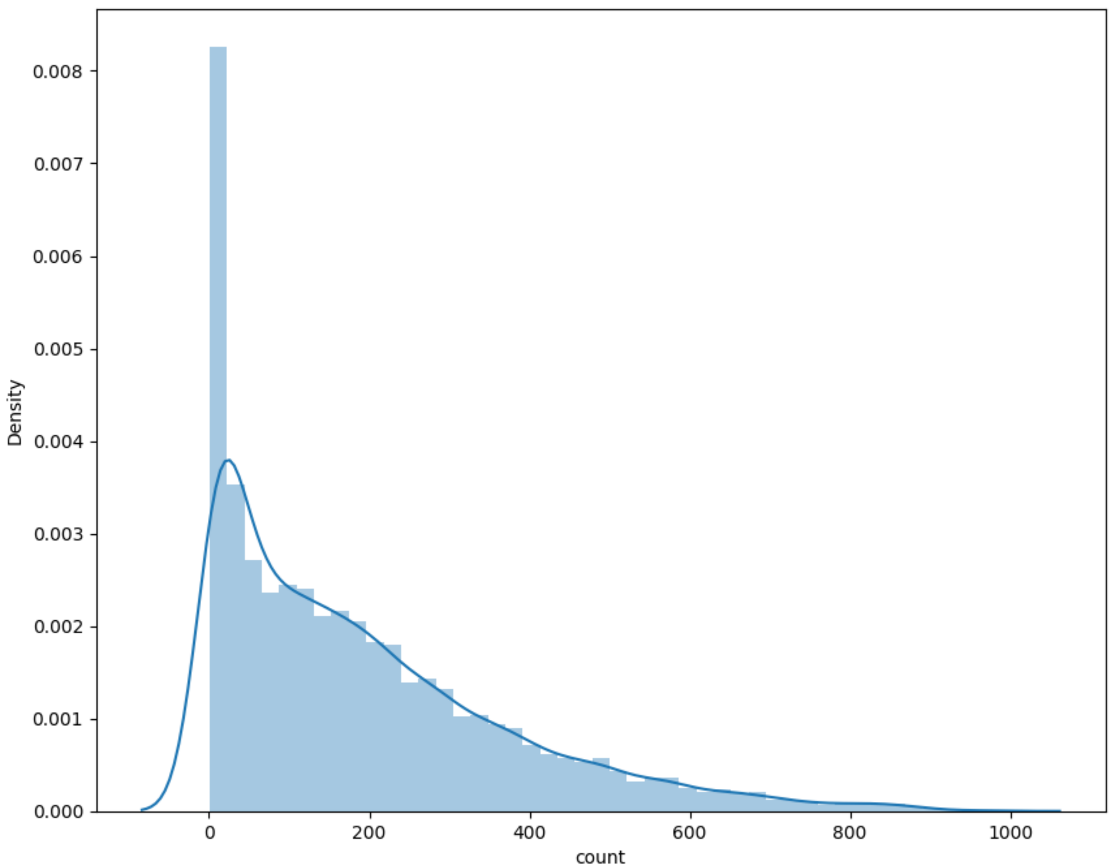
Feature Variables: Workingday, Weather, Season, and other remaining variables.

#Bivariate Analysis

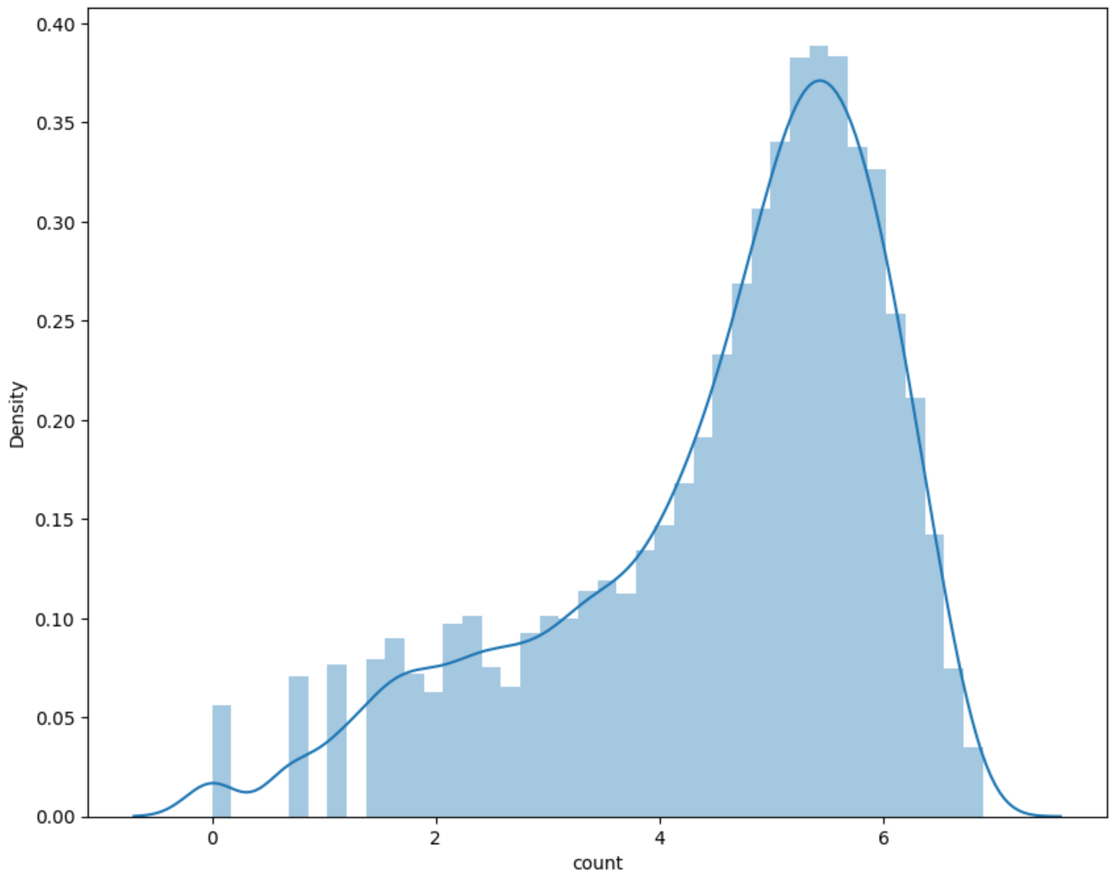
[ ]:
```
season_df = df.groupby('season',observed=True)[['casual','registered','count']].
  ↪sum().reset_index()
plt.figure(figsize=(15,5))
plt.subplot(1,5,1)
sns.barplot(season_df, x='season' , y='count', color='grey')

plt.subplot(1,5,3)
sns.barplot(season_df, x='season' , y='registered', color='grey')

plt.subplot(1,5,5)
sns.barplot(season_df, x='season' , y='casual', color='grey')
plt.show()
```



[ ]:
```
#Distribution of dependent variable - count
plt.figure(figsize=(10,8))
sns.distplot(df['count'])
plt.show()
```



[ ]:
```python
#Distribution of dependent variable - count
plt.figure(figsize=(10,8))
sns.distplot(np.log(df['count']))
plt.show()
```



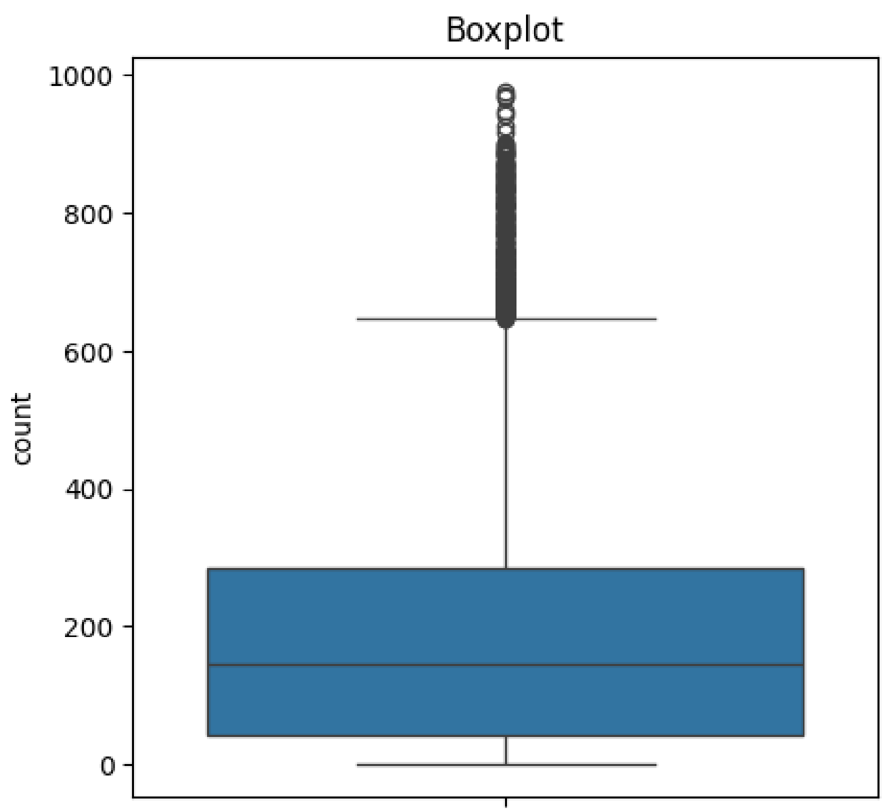
[ ]:
```python
#shapiro test for normality of both raw and logged data
before_log = df['count']
after_log = np.log(df['count'])
s_before,pvalue_before = shapiro(before_log)
s_after,pvalue_after = shapiro(after_log)

print('p_values of:')
print('Raw Data :',pvalue_before)
print('Log Data :',pvalue_after)
```

```
p_values of:
Raw Data : 0.0
Log Data : 0.0
```

[ ]:
```
#outlier Treatment of count variable

print(df['count'].describe())
#Boxplot
plt.figure(figsize=(5,5))
sns.boxplot(y=df['count'])
plt.title('Boxplot')
```

```
count    10886.000000
mean       191.574132
std        181.144454
min          1.000000
25%         42.000000
50%        145.000000
75%        284.000000
max        977.000000
Name: count, dtype: float64
```

[ ]: `Text(0.5, 1.0, 'Boxplot')`



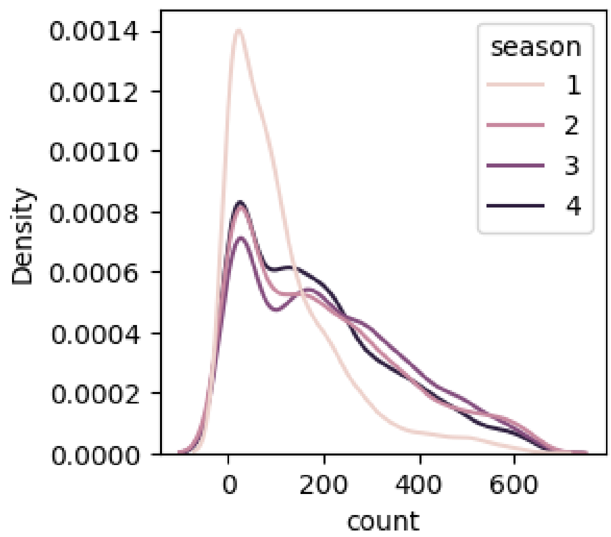
```
[ ]: #Removing outliers

q1 = df['count'].quantile(0.25)
q3 = df['count'].quantile(0.75)
iqr = q3-q1

df_ol = df[(df['count']>(q1-1.5*iqr) ) & (df['count']<(q3+1.5*iqr))]

print("No. of rows : ", df_ol.shape[0])
print("No. of rows : ", df.shape[0])
print("No. of rows dropped : ", df.shape[0]-df_ol.shape[0])
print()

df_ol['count'].describe()
```

```
No. of rows :  10583
No. of rows :  10886
No. of rows dropped :  303
```

```
[ ]: count    10583.000000
mean       175.583483
std        156.180672
min          1.000000
25%         40.000000
50%        138.000000
75%        270.000000
max        646.000000
Name: count, dtype: float64
```

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

```
[ ]: plt.figure(figsize=(3,3))
sns.kdeplot(data=df_ol, x="count", hue="season")
plt.show()
```



[ ]:
```
pd.DataFrame(df_ol.groupby('season')['count'].describe())
```

[ ]:

season	count	mean	std	min	25%	50%	75%	max
1	2670.0	112.795131	116.884929	1.0	24.00	78.0	161.00	644.0
2	2633.0	195.653627	166.170802	1.0	45.00	165.0	299.00	646.0
3	2616.0	210.484327	164.055532	1.0	59.75	185.0	323.25	646.0
4	2664.0	184.404655	154.563069	1.0	48.75	154.0	276.25	646.0

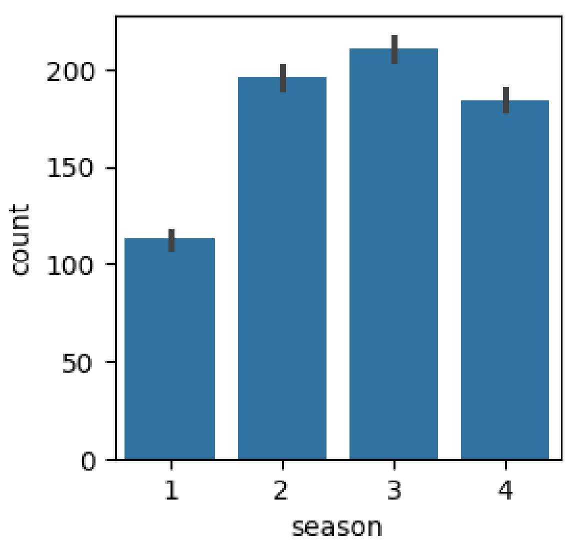
[ ]:
```
# Step 1: Define the null and alternate hypothesis
# H0:  The average no. of shared electric cycles rides in different seasons are␣
↪equal.
# Ha:  The average no. of shared electric cycles rides in different seasons are␣
↪not equal.

# Step 2: Select an appropriate test
#one way Anova test

#step 3: Assumptions:
#Populations are normally distributed

spring = df[df['season'] == 1]['count'].sample(2000)
summer = df[df['season'] == 2]['count'].sample(2000)
fall = df[df['season'] == 3]['count'].sample(2000)
winter = df[df['season'] == 3]['count'].sample(2000)

print('Shapiro test for normality of:')
print('Spring :',shapiro(spring))
```

```
print('Summer :',shapiro(summer))
print('Fall :',shapiro(fall))
print('Winter :',shapiro(winter))
print()

#Equal variance among multiple groups
print('Levene test for equality of variance:')
print('Spring and Summer :',levene(spring,summer))
print('Spring and Fall :',levene(spring,fall))
print('Spring and Winter :',levene(spring,winter))
print('Summer and Fall :',levene(summer,fall))
print('Summer and Winter :',levene(summer,winter))
print('Fall and Winter :',levene(fall,winter))
print()

#step 4 : Find p_value
print('Kruskal Wallis Test:')
stat, p = kruskal(spring, summer, fall, winter)
print('p_value :',p)

print('One way Anova test:')
stat, p = f_oneway(spring, summer, fall, winter)
print('p_value :',p)
print()

#step 5: result
alpha = 0.05
if p > alpha:
    print('Accept Null Hypothesis')
    print('The average no. of shared electric cycles rides in different seasons␣
 ↪are equal.')
else:
    print('Reject Null Hypothesis')
    print('The average no. of shared electric cycles rides in different seasons␣
 ↪are not equal.')
```

```
Shapiro test for normality of:
Spring : ShapiroResult(statistic=0.8094191551208496,
pvalue=1.485376372184306e-43)
Summer : ShapiroResult(statistic=0.8970901966094971,
pvalue=1.195227617853606e-34)
Fall : ShapiroResult(statistic=0.9134268760681152,
pvalue=2.4269936684432486e-32)
Winter : ShapiroResult(statistic=0.913000226020813,
pvalue=2.092175996193804e-32)

Levene test for equality of variance:
```

```
Spring and Summer : LeveneResult(statistic=294.3454839500751,
pvalue=1.019855082332899e-63)
Spring and Fall : LeveneResult(statistic=313.5408524914937,
pvalue=1.3247852182360383e-67)
Spring and Winter : LeveneResult(statistic=300.4239062733827,
pvalue=5.969059777468306e-65)
Summer and Fall : LeveneResult(statistic=0.2777148405555696,
pvalue=0.5982321557392789)
Summer and Winter : LeveneResult(statistic=0.02793487274790259,
pvalue=0.8672704804733106)
Fall and Winter : LeveneResult(statistic=0.12952016259632207,
pvalue=0.7189490793908649)

Kruskal Wallis Test:
p_value : 1.1751274455285527e-115
One way Anova test:
p_value : 4.8885682914181325e-110

Reject Null Hypothesis
The average no. of bike rides in different seasons are not equal.
```

[ ]:
```
plt.figure(figsize=(3,3))
sns.barplot(data=df_ol, x="season", y='count')
plt.show()
```



[ ]:
```
#Lets do pairwise test for season
print('ttest for spring and summer:')
stat, p = ttest_ind(spring, summer)
print('p_value :',p)
```

```python
print()

print('ttest for spring and fall:')
stat, p = ttest_ind(spring, fall)
print('p_value :',p)
print()

print('ttest for spring and winter:')
stat, p = ttest_ind(summer, winter)
print('p_value :',p)
print()

print('ttest for summer and fall:')
stat, p = ttest_ind(summer, winter)
print('p_value :',p)
print()

print('ttest for summer and winter:')
stat, p = ttest_ind(summer, winter)
print('p_value :',p)
print()

print('ttest for fall and winter:')
stat, p = ttest_ind(fall, winter)
print('p_value :',p)
print()
```

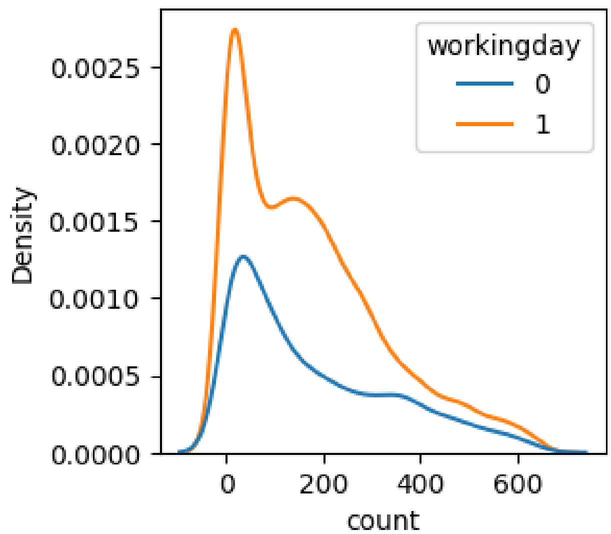
```
ttest for spring and summer:
p_value : 6.288580566538464e-78

ttest for spring and fall:
p_value : 5.649143906269248e-109

ttest for spring and winter:
p_value : 0.005303506000753357

ttest for summer and fall:
p_value : 0.005303506000753357

ttest for summer and winter:
p_value : 0.005303506000753357

ttest for fall and winter:
p_value : 0.539098149769994
```

P_value of ttest for the season fall and winter are greater than significant level. So, we can assume that the average no. of shared electric cycles rides in the seasons fall and winter are same. But still assumtion of ttest got failed for these variables.

[ ]:
```
#Categorical vs Continuous
```

#Check if the demand of bicycles on rent is the same for workingdays vs holidays?

[ ]:
```
plt.figure(figsize=(3,3))
sns.kdeplot(data=df_ol, x="count", hue="workingday")
plt.show()
```



[ ]:
```
pd.DataFrame(df_ol.groupby('workingday')['count'].describe())
```

[ ]:

workingday	count	mean	std	min	25%	50%	75%	max
0	3422.0	180.965517	163.782166	1.0	43.0	124.0	295.75	645.0
1	7161.0	173.011591	152.358993	1.0	38.0	143.0	262.00	646.0

[ ]:
```
# Step 1: Define the null and alternate hypothesis
# H0:  Mean of working day = mean of holiday  (or)
#      Mean of working day <= mean of holiday  (or)
#      Mean of working day >= mean of holiday.
# Ha:  Mean of working day != mean of holiday (or)
#      Mean of working day > mean of holiday  (or)
#      Mean of working day < mean of holiday.

# Step 2: Select an appropriate test
#ttest for independence

#step 3: Assumptions:
```

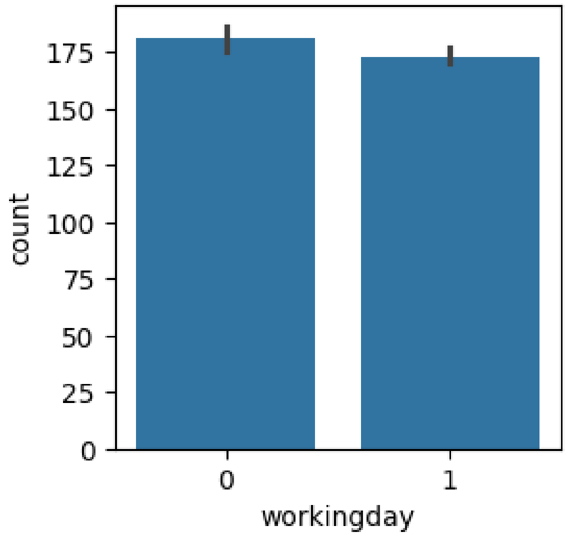
```python
#Populations are normally distributed

workingday = df[df['workingday'] == 1]['count'].sample(3000)
holiday = df[df['workingday'] == 0]['count'].sample(3000)

print('Shapiro test for normality of:')
print('Workingday :',shapiro(workingday))
print('Holiday :',shapiro(holiday))
print()

#Equal variance among multiple groups
print('Levene test for equality of variance:')
print('Workingday :',levene(workingday,holiday))
print()

#step 4 : Find p_value
print('Kruskal Wallis Test:')
stat, p = kruskal(workingday,holiday)
print('p_value :',p)
print()

print('Ttest for independent(workingday = holiday):')
stat1, p1 = ttest_ind(workingday,holiday)
print('p_value :',p1)

#step 5: result
alpha = 0.05
if p1 < alpha:
  print('Reject Null Hypothesis')
  print('The average no. of shared electric cycles rides in different␣
 ↪workingday are not equal.')
else:
    print('Accept Null Hypothesis')
    print('The average no. of shared electric cycles rides in different␣
 ↪workingday are equal.')
print()


print('Ttest for independent(workingday < holiday):')
stat2, p2 = ttest_ind(workingday,holiday, alternative='less')
print('p_value :',p2)

#result
alpha = 0.05
if p2 < alpha:
  print('Reject Null Hypothesis')
```

```
  print('The average no. of shared electric cycles rides in Holidays are more␣
 ↪than Workingday.')
else:
    print('Accept Null Hypothesis')
    print('The average no. of shared electric cycles rides in Holidays are not␣
 ↪more than Workingday.')
print()

print('Ttest for independent(workingday > holiday):')
stat3, p3 = ttest_ind(workingday,holiday, alternative='greater')
print('p_value :',p3)

#result
alpha = 0.05
if p3 < alpha:
  print('Reject Null Hypothesis')
  print('The average no. of shared electric cycles rides in Holidays are less␣
 ↪than Workingday.')
else:
    print('Accept Null Hypothesis')
    print('The average no. of shared electric cycles rides in Holidays are not␣
 ↪less than Workingday.')
```

```
Shapiro test for normality of:
Workingday : ShapiroResult(statistic=0.8663371205329895,
pvalue=5.605193857299268e-45)
Holiday : ShapiroResult(statistic=0.8856452703475952,
pvalue=1.5456322061502732e-42)

Levene test for equality of variance:
Workingday : LeveneResult(statistic=0.545813328143306,
pvalue=0.4600623379335489)

Kruskal Wallis Test:
p_value : 0.8991827240929343

Ttest for independent(workingday = holiday):
p_value : 0.20754368654581107
Accept Null Hypothesis
The average no. of bike rides in different workingday are equal.

Ttest for independent(workingday < holiday):
p_value : 0.8962281567270944
Accept Null Hypothesis
The average no. of bike rides in Holidays are not more than Workingday.

Ttest for independent(workingday > holiday):
```

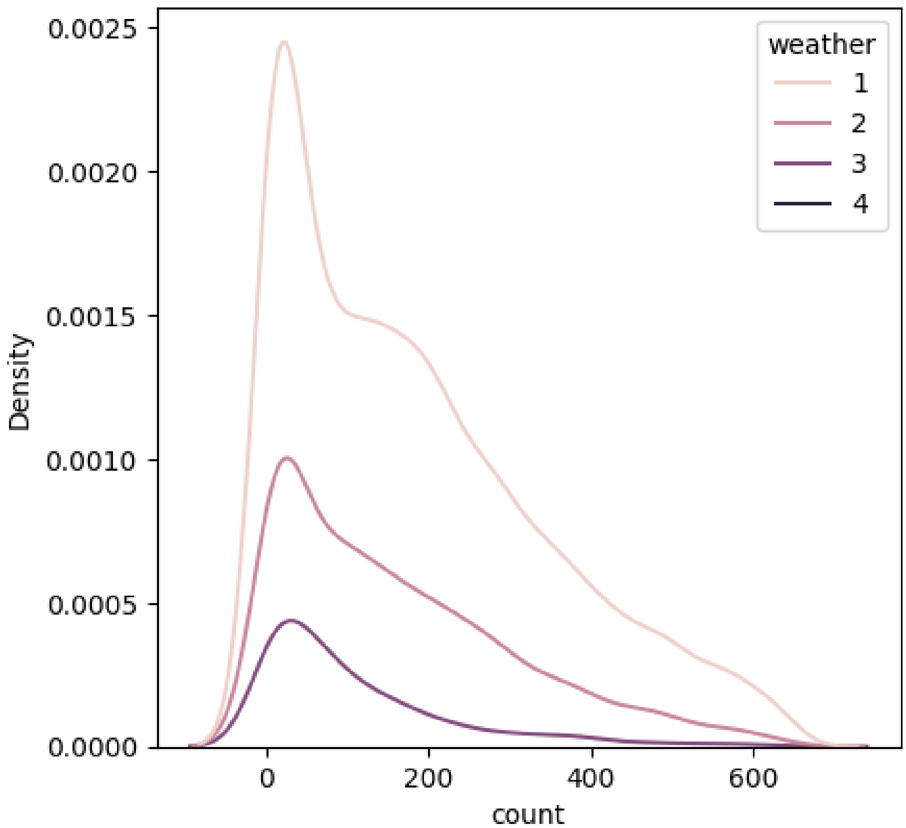
```
p_value : 0.10377184327290553
Accept Null Hypothesis
The average no. of bike rides in Holidays are not less than Workingday.
```

[ ]:
```
plt.figure(figsize=(3,3))
sns.barplot(data=df_ol, x="workingday", y='count')
plt.show()
```



#Check if the demand of bicycles on rent is the same for different weather.

[ ]:
```
plt.figure(figsize=(5,5))
sns.kdeplot(data=df_ol, x="count", hue="weather")
plt.show()
```



[ ]:
```
pd.DataFrame(df_ol.groupby('weather')['count'].describe())
```

[ ]:

	count	mean	std	min	25%	50%	75%	max
weather								
1	6962.0	187.131140	161.333785	1.0	45.0	153.0	286.0	646.0
2	2770.0	166.117690	146.992422	1.0	39.0	130.0	254.0	646.0
3	850.0	111.862353	121.233389	1.0	23.0	70.5	157.0	646.0
4	1.0	164.000000	NaN	164.0	164.0	164.0	164.0	164.0

[ ]:
```
# Step 1: Define the null and alternate hypothesis
# H0:  The average no. of shared electric cycles rides in different weather are
 ↪equal.
# Ha:  The average no. of shared electric cycles rides in different weather are
 ↪not equal.

# Step 2: Select an appropriate test
#one way Anova test

#step 3: Assumptions:
#Populations are normally distributed
```

```python
clear = df[df['weather'] == 1]['count'].sample(2000, replace=True)
mist = df[df['weather'] == 2]['count'].sample(2000, replace=True)
light_snow = df[df['weather'] == 3]['count'].sample(2000, replace=True)

print('Shapiro test for normality of:')
print('Clear :',shapiro(clear))
print('Mist :',shapiro(mist))
print('Light_snow :',shapiro(light_snow))
print()

#Equal variance among multiple groups
print('Levene test for equality of variance:')
print('Clear and Mist :',levene(clear,mist))
print('Clear and Light Snow :',levene(clear,light_snow))
print('Mist and Light Snow :',levene(mist,light_snow))
print()

#step 4 : Find p_value
print('Kruskal Wallis Test:')
stat, p = kruskal(clear, mist, light_snow)
print('p_value :',p)

print('One way Anova test:')
stat, p = f_oneway(clear, mist, light_snow)
print('p_value :',p)
print()

#step 5: result
alpha = 0.05
if p > alpha:
    print('Accept Null Hypothesis')
    print('The average no. of shared electric cycles rides in different weather␣
 ↪are equal.')
else:
    print('Reject Null Hypothesis')
    print('The average no. of shared electric cycles rides in different weather␣
 ↪are not equal.')
```

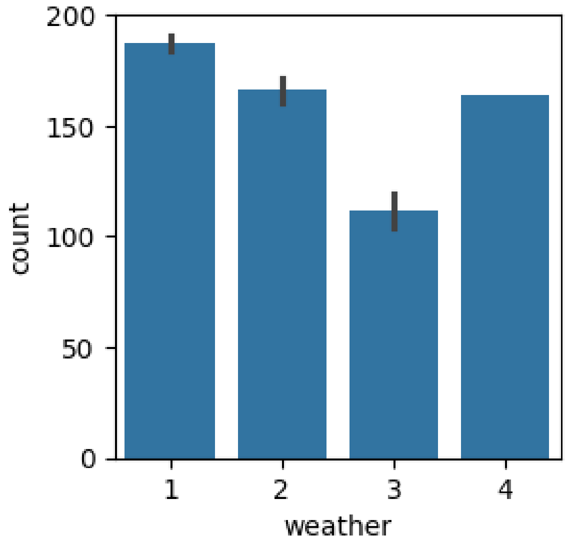
```
Shapiro test for normality of:
Clear : ShapiroResult(statistic=0.8894162774085999,
pvalue=1.238195035824156e-35)
Mist : ShapiroResult(statistic=0.8769956827163696, pvalue=4.063523850582843e-37)
Light_snow : ShapiroResult(statistic=0.7491618394851685, pvalue=0.0)

Levene test for equality of variance:
Clear and Mist : LeveneResult(statistic=24.71486826186925,
```

```
pvalue=6.925637358506148e-07)
Clear and Light Snow : LeveneResult(statistic=180.05566182113418,
pvalue=3.4487753953946214e-40)
Mist and Light Snow : LeveneResult(statistic=81.89195638645549,
pvalue=2.1958779382837034e-19)

Kruskal Wallis Test:
p_value : 2.4715417115327603e-54
One way Anova test:
p_value : 3.859436387031906e-54

Reject Null Hypothesis
The average no. of bike rides in different weather are not equal.
```

[ ]:
```
#Lets do pairwise test for weather
print('ttest for Clear and Mist:')
stat, p = ttest_ind(clear, mist, alternative='greater')
print('p_value :',p)
print()

print('ttest for Clear and Light Snow:')
stat, p = ttest_ind(clear, light_snow, alternative='greater')
print('p_value :',p)
print()

print('ttest for Mist and Light Snow:')
stat, p = ttest_ind(mist, light_snow, alternative='greater')
print('p_value :',p)
print()
```

```
ttest for Clear and Mist:
p_value : 6.448320198104754e-09

ttest for Clear and Light Snow:
p_value : 1.2194794320980146e-53

ttest for Mist and Light Snow:
p_value : 2.433565388806811e-25
```

[ ]:
```
plt.figure(figsize=(3,3))
sns.barplot(data=df_ol, x="weather", y='count')
plt.show()
```




```
[ ]: #categorical vs categorical
```

#Check if Weather and Season are dependent on each other or not.

```
[ ]: print(pd.crosstab(df_ol['weather'],df_ol['season']))
     df_w = df_ol[df_ol['weather'] != 4]
     print('\n',pd.crosstab(df_w['weather'],df_w['season']))
```

```
season      1     2     3     4
weather
1        1744  1720  1842  1656
2         714   690   579   787
3         211   223   195   221
4           1     0     0     0

 season      1     2     3     4
weather
1        1744  1720  1842  1656
2         714   690   579   787
3         211   223   195   221
```

```
[ ]: # Step 1: Define the null and alternate hypothesis
     # H0:  Weather and Season are independent of each other.
     # Ha:  Weather and Season are dependent on each other.

     # Step 2: Select an appropriate test
     #Chi-square test

     #step 3: Assumptions:
```

```
#It is a non parametric test. So, there is no assumption.

#step 4 : Find p_value
print('Chi-square test:')
stat, p, dof, expected = chi2_contingency(pd.
 ↪crosstab(df_w['weather'],df_w['season']))
print('p_value :',p,'\n')

#step 5: result
alpha = 0.05
if p > alpha:
    print('Accept Null Hypothesis')
    print('Weather and Season are independent of each other.')
else:
    print('Reject Null Hypothesis')
    print('Weather and Season are dependent on each other.')
```

```
Chi-square test:
p_value : 6.75312212866461e-08

Reject Null Hypothesis
Weather and Season are dependent on each other.
```

#Check if Season and Working days are dependent on each other or not.

[ ]:
```
pd.crosstab(df_ol['season'],df_ol['workingday'])
```

[ ]:

workingday	0	1
season		
1	852	1818
2	821	1812
3	871	1745
4	878	1786

[ ]:
```
# Step 1: Define the null and alternate hypothesis
# H0:  Workingday and Season are independent of each other.
# Ha:  Workingday and Season are dependent on each other.

# Step 2: Select an appropriate test
#Chi-square test

#step 3: Assumptions:
#It is a non parametric test. So, there is no assumption.

#step 4 : Find p_value
print('Chi-square test:')
stat, p, dof, expected = chi2_contingency(pd.
 ↪crosstab(df_ol['workingday'],df_ol['season']))
```

```
print('p_value :',p)
print()

#step 5: result
alpha = 0.05
if p > alpha:
    print('Accept Null Hypothesis')
    print('Season and Workingday are independent of each other.')
else:
    print('Reject Null Hypothesis')
    print('Season and Workingday are dependent on each other.')
```

```
Chi-square test:
p_value : 0.334351895185102

Accept Null Hypothesis
Season and Workingday are independent of each other.
```

#Check if Weather and Working days are dependent on each other or not.

[ ]:
```
pd.crosstab(df_w['weather'],df_w['workingday'])
```

[ ]:

workingday	0	1
weather		
1	2307	4655
2	891	1879
3	224	626

[ ]:
```
# Step 1: Define the null and alternate hypothesis
# H0:  Workingday and Weather are independent of each other.
# Ha:  Workingday and Weather are dependent on each other.

# Step 2: Select an appropriate test
#Chi-square test

#step 3: Assumptions:
#It is a non parametric test. So, there is no assumption.

#step 4 : Find p_value
print('Chi-square test:')
stat, p, dof, expected = chi2_contingency(pd.
 ↪crosstab(df_w['workingday'],df_w['weather']))
print('p_value :',p)
print()

#step 5: result
alpha = 0.05
if p > alpha:
```

```
    print('Accept Null Hypothesis')
    print('Weather and Workingday are independent of each other.')
else:
    print('Reject Null Hypothesis')
    print('Weather and Workingday are dependent on each other.')
```

```
Chi-square test:
p_value : 0.00033809996118099197

Reject Null Hypothesis
Weather and Workingday are dependent on each other.
```

#Check if Weather and Holiday are dependent on each other or not.

[ ]:

```

```

[ ]:

```
# Step 1: Define the null and alternate hypothesis
# H0:  Holiday and Weather are independent of each other.
# Ha:  Holiday and Weather are dependent on each other.

# Step 2: Select an appropriate test
#Chi-square test

#step 3: Assumptions:
#It is a non parametric test. So, there is no assumption.

#step 4 : Find p_value
print('Chi-square test:')
stat, p, dof, expected = chi2_contingency(pd.
 ↪crosstab(df_w['holiday'],df_w['weather']))
print('p_value :',p)
print()

#step 5: result
alpha = 0.05
if p > alpha:
    print('Accept Null Hypothesis')
    print('Weather and Holiday are independent of each other.')
else:
    print('Reject Null Hypothesis')
    print('Weather and Holiday are dependent on each other.')
```

```
Chi-square test:
p_value : 0.061295163277045574

Accept Null Hypothesis
Weather and Holiday are independent of each other.
```

#Check if Workingday and Holiday are dependent on each other or not.

[ ]:
```
pd.crosstab(df_ol['workingday'],df_ol['holiday'])
```

[ ]:
```
holiday        0    1
workingday
0           3113  309
1           7161    0
```

[ ]:
```
# Step 1: Define the null and alternate hypothesis
# H0:  Holiday and Workingday are independent of each other.
# Ha:  Holiday and Workingday are dependent on each other.

# Step 2: Select an appropriate test
#Chi-square test

#step 3: Assumptions:
#It is a non parametric test. So, there is no assumption.

#step 4 : Find p_value
print('Chi-square test:')
stat, p, dof, expected = chi2_contingency(pd.
 ↪crosstab(df_w['holiday'],df_w['workingday']))
print('p_value :',p)
print()

#step 5: result
alpha = 0.05
if p > alpha:
    print('Accept Null Hypothesis')
    print('Workingday and Holiday are independent of each other.')
else:
    print('Reject Null Hypothesis')
    print('Workingday and Holiday are dependent on each other.')
```

```
Chi-square test:
p_value : 3.6769010835886696e-146

Reject Null Hypothesis
Workingday and Holiday are dependent on each other.
```

#Check if There is a significant interaction effect between weather and workingday on the casual variable or not

[ ]:
```
#step 1:Define the null and alternate hypothesis
# Null Hypotheses (H0):
# H0_1: There is no significant effect of weather on the casual variable.
# H0_2: There is no significant effect of workingday on the casual variable.
# H0_3: There is no significant interaction effect between weather and␣
 ↪workingday on the casual variable.
```

```
# Alternative Hypotheses (H1):
# H1_1: There is a significant effect of weather on the casual variable.
# H1_2: There is a significant effect of workingday on the casual variable.
# H1_3: There is a significant interaction effect between weather and␣
↪workingday on the casual variable.

# step 2: Select an appropriate test
#two way anova

#step 3: Assumptions:
#Populations are normally distributed
#Equal variance among multiple groups

#step 4 : Find p_value
test = ols('casual ~ C(weather)* C(workingday)', data=df).fit()
aov_table = anova_lm(test, typ=2)
print(aov_table)
print()

# step 5: result
alpha = 0.05
if aov_table['PR(>F)'][0] < alpha:
    print('Reject Null Hypothesis')
    print('There is a significant effect of weather on the casual variable.\n')
else:
    print('There is no significant effect of weather on the casual variable.\n')
if aov_table['PR(>F)'][1] < alpha:
    print('Reject Null Hypothesis')
    print('There is a significant effect of workingday on the casual variable.
↪\n')
else:
    print('There is no significant effect of workingday on the casual variable.
↪\n')
if aov_table['PR(>F)'][2] < alpha:
    print('Reject Null Hypothesis')
    print('There is a significant interaction effect between weather and␣
↪workingday on the casual variable.\n')
else:
    print('There is no significant interaction effect between weather and␣
↪workingday on the casual variable.')
```

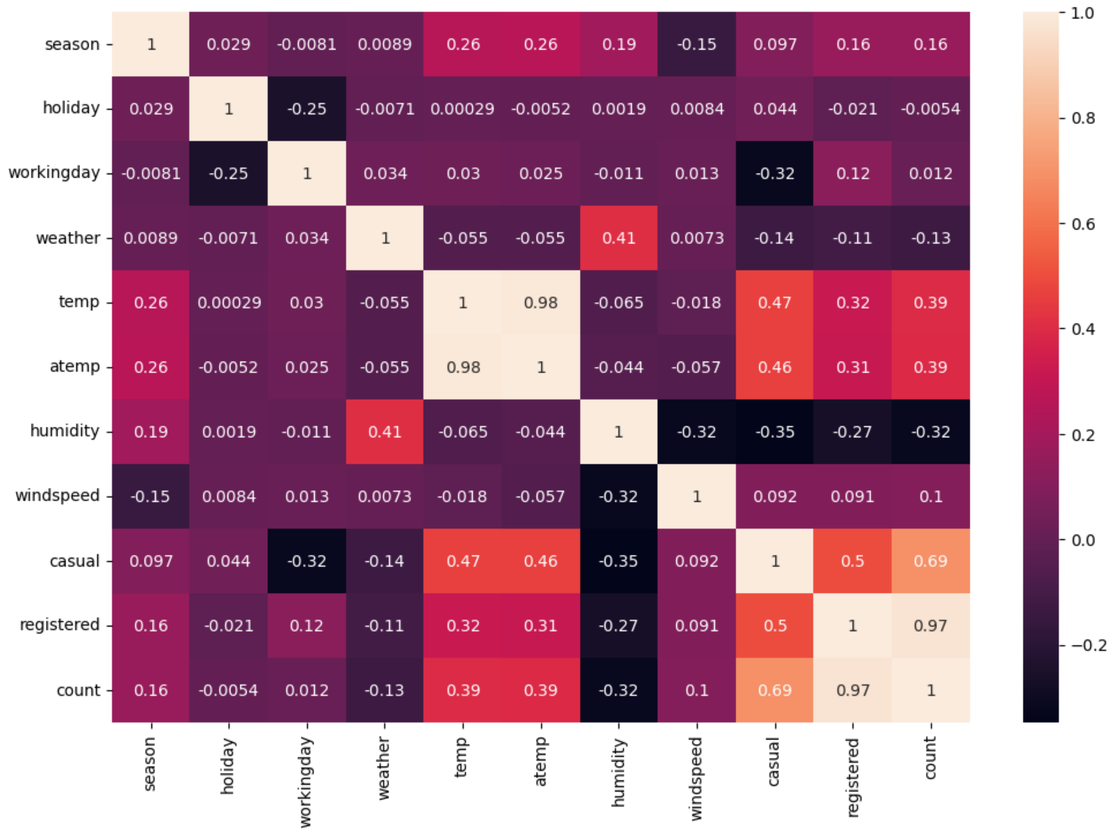
```
                                sum_sq       df            F         PR(>F)
C(weather)                4.274094e+05      3.0    64.840927   1.506375e-41
C(workingday)             2.688005e+06      1.0  1223.366534  4.301997e-254
C(weather):C(workingday)  7.181073e+04      3.0    10.894180   3.845174e-07
Residual                  2.390356e+07  10879.0          NaN            NaN
```

```
Reject Null Hypothesis
There is a significant effect of weather on the casual variable.

Reject Null Hypothesis
There is a significant effect of workingday on the casual variable.

Reject Null Hypothesis
There is a significant interaction effect between weather and workingday on the
casual variable.
```

[ ]:
```python
#Numerical vs Numerical
```

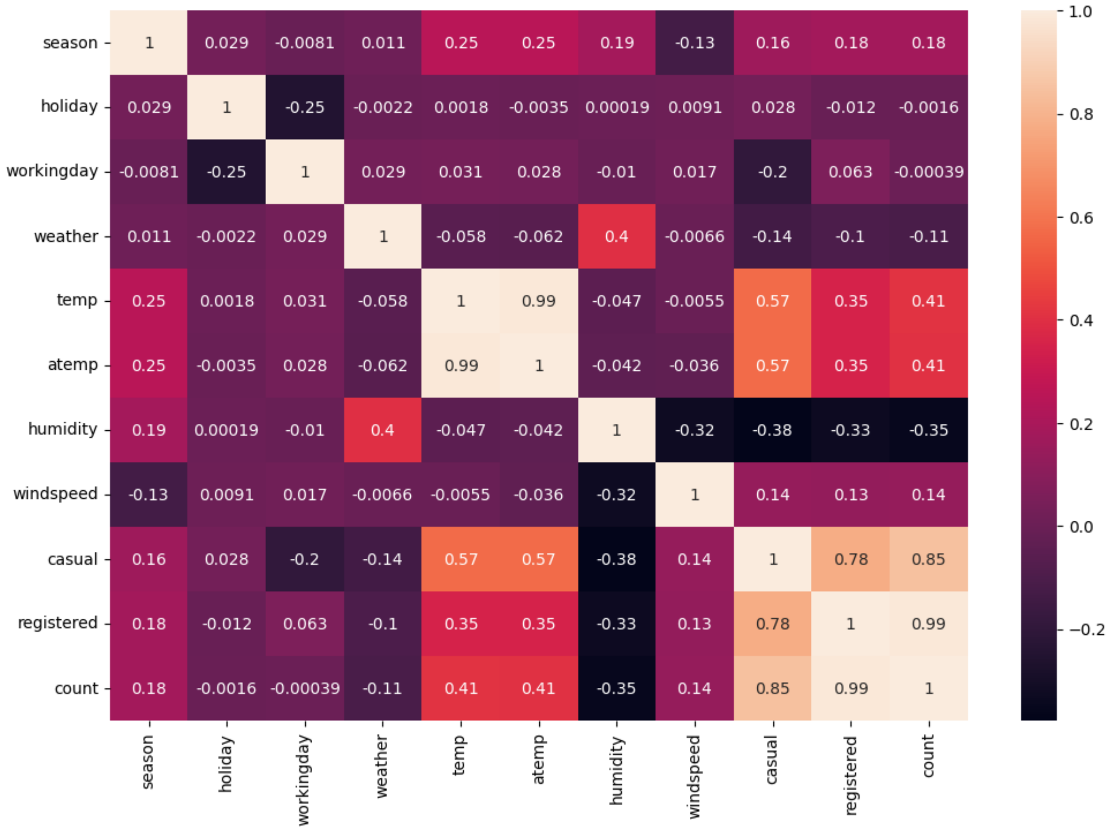
#Correlation Test

[ ]:
```python
# Pearson Correlation using Heatmap

plt.figure(figsize=(12, 8))
sns.heatmap(df[['season', 'holiday', 'workingday', 'weather', 'temp',
       'atemp', 'humidity', 'windspeed', 'casual', 'registered', 'count']].
 ↪corr(method='pearson'),
            annot=True)
plt.show()
```



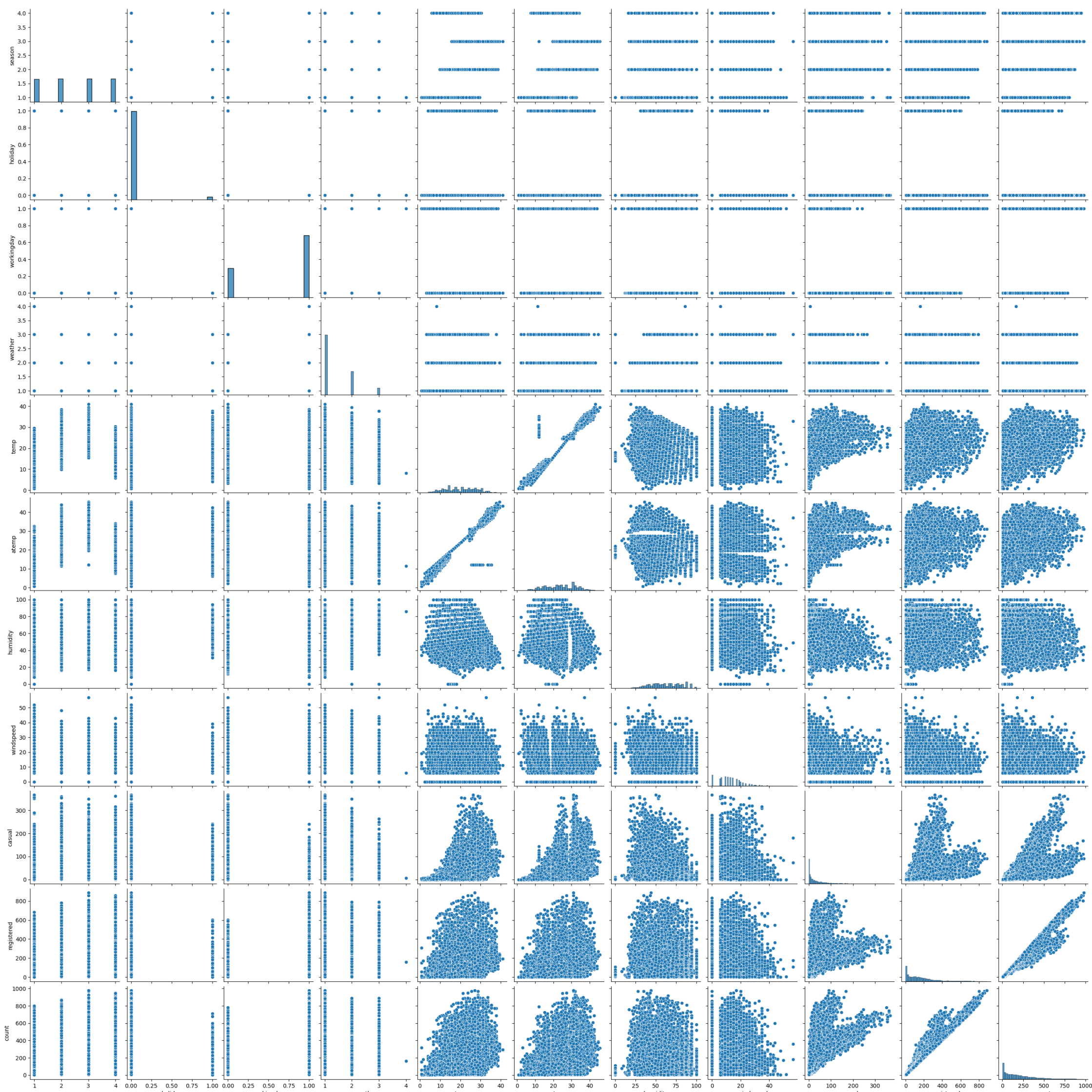
[ ]:

```python
# Spearman Correlation using Heatmap

plt.figure(figsize=(12, 8))
sns.heatmap(df[['season', 'holiday', 'workingday', 'weather', 'temp',
       'atemp', 'humidity', 'windspeed', 'casual', 'registered', 'count']].
 ↪corr(method='spearman'),
             annot=True)
plt.show()
```



[ ]:

```python
sns.pairplot(data=df)
plt.show()
```



- The correlation table for both pearson and spearman are almost same. So we can decide with pearson table which gives most accurate result.

[ ]:

```
                              sum_sq       df            F         PR(>F)
C(weather)              4.274094e+05      3.0    64.840927   1.506375e-41
C(workingday)           2.688005e+06      1.0  1223.366534  4.301997e-254
C(weather):C(workingday) 7.181073e+04     3.0    10.894180   3.845174e-07
Residual                2.390356e+07  10879.0          NaN            NaN

Reject Null Hypothesis
There is a significant effect of weather on the casual variable.
```

```
Reject Null Hypothesis
There is a significant effect of workingday on the casual variable.

Reject Null Hypothesis
There is a significant interaction effect between weather and workingday on the
casual variable.
```

##Insights based on Hypothesis: * The average no. of shared electric cycles rides in different seasons are not equal.But average no. of shared electric cycles rides in the seasons fall and winter are same. * The average no. of shared electric cycles rides in different workingday are equal. * The average no. of shared electric cycles rides in different weather are not equal. * There is a significant interaction effect between weather and workingday on the casual variable. ——————————————————————————-

- The features Weather and Season are dependent on each other.
- The features Season and Workingday are independent of each other.
- The features Weather and Workingday are dependent on each other.
- The features Weather and Holiday are independent of each other.
- The features Working and Holiday are dependent on each other.
-

0.1 Based on the contingency table between workingday and holiday columns, all holidays are covered within the workingday feature, allowing us to proceed with workingday and remove holiday from the list of features.

- There is a very strong positive correlation between Count and Registered variable.
- Similary, a very strong positive correlation between temp and atemp variable.
- There is a good positive correlation between casual and count variable.
- Based on these analysis, we can keep either temp or atemp and either count or registered varialbe in a feature list.
- There is a slight negative correlation between humidity and windspeed, as well as between humidity and count.
- Positive correlation between atemp/temp and casual is slightly higher than between atemp/temp and registered.

#Recommendation: - There is a dip in demand for shared electric cycle during extreme weather conditions such as heavy rain, ice pellets, thunderstorms, mist, snow, and fog. Identify if shared electric cycles are suitable for these conditions or offer alternative transport options during severe weather. - Introduce additional safety measures. - Conduct analysis based on region and weather to gain insights, as weather may be a key factor impacting the dip in revenue. - Offer discounts for rides taken during mild weather conditions to balance out the lower demand during extreme weather. - Conduct a survey through the Yulu app to gather customer preferences on vehicles for different seasons and weather conditions, as well as their expected changes, for more precise analysis.

[ ]:
```

```

[ ]:
```

```