Yulu - Hypothesis Testing

September 4, 2023

1 Business Case: Yulu - Confidence Interval and CLT

- Yulu Data_Set
- Yulu Project Sebmision Link

[ ]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

import math
from scipy import stats
from scipy.stats import norm, binom, geom, poisson  # Distributions
from scipy.stats import zscore, ttest_1samp, ttest_ind, ttest_rel  # Z-Test &
↪T-Test
from scipy.stats import chisquare, chi2_contingency, chi2  # ChiSquare Test
from scipy.stats import f_oneway, kruskal, shapiro, levene, ks_2samp  # Anova
↪Test
from scipy.stats import pearsonr, spearmanr  # Co-Relation Test
from scipy.stats import shapiro  # Co-Relation Test

from statsmodels.graphics.gofplots import qqplot

import warnings
warnings.filterwarnings("ignore")

sns.set_theme(style="darkgrid")
```

1.1 Mindset

- Evaluation will be kept lenient, so make sure you attempt this case study.
- It is understandable that you might struggle with getting started on this. Just brainstorm, discuss with peers, or get help from TAs.
- There is no right or wrong answer. We have to become comfortable with dealing with uncertainty in business.
  - This is exactly the skill we want to develop.

1.2 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.

1.3 About DataSet

- datetime: datetime
- season: season `(1: spring, 2: summer, 3: fall, 4: winter)`
- holiday: whether day is a holiday or not (extracted from http://dchr.dc.gov/page/holiday-schedule)
- workingday: if day is neither weekend nor holiday is 1, otherwise is 0.
- weather:
  - 1: `Clear, Few clouds, partly cloudy, 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`
- temp: temperature in Celsius
- atemp: feeling temperature in Celsius
- humidity: humidity
- windspeed: wind speed
- casual: count of casual users
- registered: count of registered users
- count: count of total rental bikes including both casual and registered

[ ]:
```
yulu_raw = pd.read_csv("./bike_sharing.csv", parse_dates=[0], dayfirst=True)
yulu_raw.head(5)
```

[ ]:
```
             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
```

[ ]:
```
confidence_interval = 99/100
# # confidence_interval = 99%
significance_level = 1-confidence_interval
# $ significance_level = 1%
```

1.4 Business Problem

The company wants to know:

- Which variables are significant in predicting the demand for shared electric cycles in the Indian market?
- How well those variables describe the electric cycle demands

1.5 Exploring The Data Set

[ ]:
```
yulu_raw.shape
```

[ ]:
```
(10886, 12)
```

[ ]:
```
yulu_raw.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10886 entries, 0 to 10885
Data columns (total 12 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   datetime    10886 non-null  datetime64[ns]
 1   season      10886 non-null  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: datetime64[ns](1), float64(3), int64(8)
memory usage: 1020.7 KB
```

1.5.1 Missing value detection & fill with relevent data.

[ ]:
```
yulu_raw.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
```

1.5.2 Check for Outliers

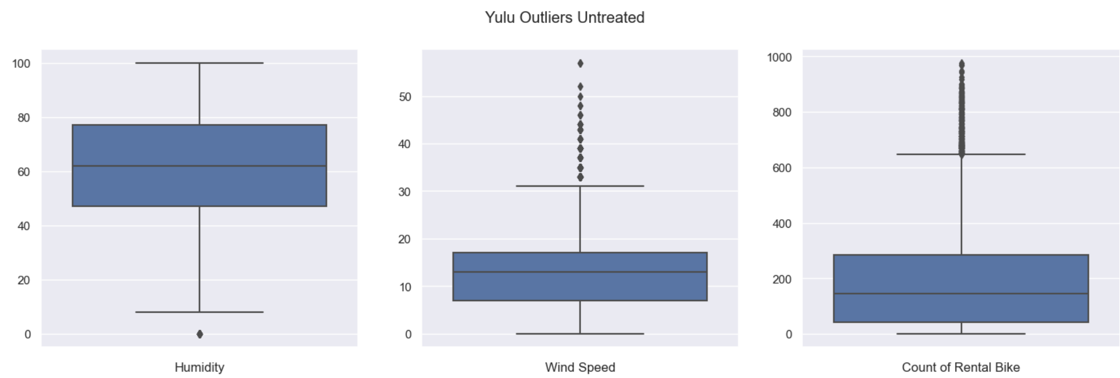
[ ]:
```
def check_outlier(df, x):
    Q1 = df[x].quantile(0.25)
    Q3 = df[x].quantile(0.75)
    IQR = Q3 - Q1
    lower = Q1 - 1.5*IQR
    upper = Q3 + 1.5*IQR
    lower_outlier = df[x][df[x] < lower]
    upper_outlier = df[x][df[x] > upper]

    return {
        'lower': {
            'list': lower_outlier,
            'length': len(lower_outlier)
        },
        'upper': {
            'list': upper_outlier,
            'length': len(upper_outlier)
        }}
```

[ ]:
```
for i in yulu_raw[['humidity', 'windspeed', 'count', 'temp', 'atemp', 'casual',␣
↪'registered']].columns:
    # print(i)
    outlier = check_outlier(yulu_raw, i)
    print("{} : \n\t'Lower Outliers': {} \n\t'Higher Outliers': {} \n\t'Mean -␣
↪Median': {}\n".format(i,
            outlier['lower']['length'], outlier['upper']['length'],␣
↪round(yulu_raw[i].mean()-yulu_raw[i].median(), 4)))
```

```
humidity :
        'Lower Outliers': 22
        'Higher Outliers': 0
        'Mean - Median': -0.1135

windspeed :
        'Lower Outliers': 0
        'Higher Outliers': 227
        'Mean - Median': -0.1986

count :
        'Lower Outliers': 0
        'Higher Outliers': 300
        'Mean - Median': 46.5741

temp :
        'Lower Outliers': 0
        'Higher Outliers': 0
        'Mean - Median': -0.2691

atemp :
        'Lower Outliers': 0
        'Higher Outliers': 0
        'Mean - Median': -0.5849

casual :
        'Lower Outliers': 0
        'Higher Outliers': 749
        'Mean - Median': 19.022

registered :
        'Lower Outliers': 0
        'Higher Outliers': 423
        'Mean - Median': 37.5522
```

[ ]:
```
plt.figure(figsize=(18, 5)).suptitle(" Yulu Outliers Untreated")
plt.subplot(1, 3, 1)
sns.boxplot(yulu_raw, y='humidity')
plt.xlabel("Humidity")
plt.ylabel("")
plt.subplot(1, 3, 2)
sns.boxplot(yulu_raw, y='windspeed')
plt.xlabel("Wind Speed")
plt.ylabel("")
plt.subplot(1, 3, 3)
sns.boxplot(yulu_raw, y='count')
```

```
plt.xlabel("Count of Rental Bike")
plt.ylabel("")
plt.show()
```



1.5.3 Remove Outliers

[ ]:
```
def remove_outlier(df, x):
    Q1 = df[x].quantile(0.25)
    Q3 = df[x].quantile(0.75)
    IQR = Q3 - Q1
    lower = Q1 - 1.5*IQR
    upper = Q3 + 1.5*IQR

    # print((lower-df[x].mean())/df[x].std())
    # print((upper-df[x].mean())/df[x].std())
    # print(df.shape, x, lower, upper)

    return df[(df[x] > lower) & (df[x] < upper)]
```

[ ]:
```
for i in yulu_raw[['humidity', 'windspeed', 'count']].columns:
    # print(i)
    yulu_raw = remove_outlier(yulu_raw, i)
    print(yulu_raw.shape)
```

```
(10864, 12)
(10638, 12)
(10352, 12)
```

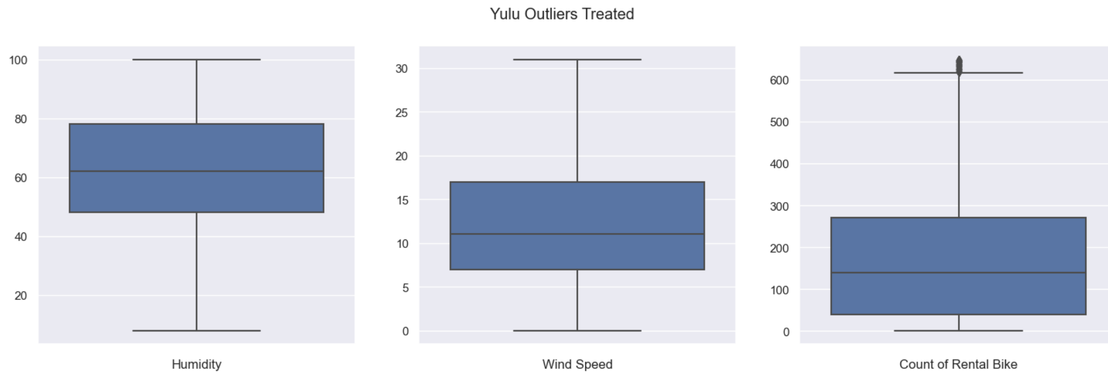
[ ]:
```
# humidity_z = np.abs(stats.zscore(yulu_raw['humidity']))
# windspeed_z = np.abs(stats.zscore(yulu_raw['windspeed']))
# count_z = np.abs(stats.zscore(yulu_raw['count']))
# threshold = 3
# yulu_raw[((humidity_z < 3) & (humidity_z > -3))]
# yulu_raw = yulu_raw[((windspeed_z < 3) & (windspeed_z > -3))]
```

```
# yulu_raw = yulu_raw[((count_z < 3) & (count_z > -3))]
# yulu_raw.shape
```

[ ]:
```
for i in yulu_raw[['humidity', 'windspeed', 'count']].columns:
    # print(i)
    outlier = check_outlier(yulu_raw, i)
    print("{} : \n\t'Lower Outliers': {} \n\t'Higher Outliers': {} \n\t'Mean -␣
 ↪Median': {}\n".format(i,
            outlier['lower']['length'], outlier['upper']['length'],␣
 ↪round(yulu_raw[i].mean()-yulu_raw[i].median(), 4)))
```

```
humidity :
        'Lower Outliers': 0
        'Higher Outliers': 0
        'Mean - Median': 0.6235

windspeed :
        'Lower Outliers': 0
        'Higher Outliers': 0
        'Mean - Median': 1.2663

count :
        'Lower Outliers': 0
        'Higher Outliers': 77
        'Mean - Median': 37.3668
```

[ ]:
```
plt.figure(figsize=(18, 5)).suptitle(" Yulu Outliers Treated")
plt.subplot(1, 3, 1)
sns.boxplot(yulu_raw, y='humidity')
plt.xlabel("Humidity")
plt.ylabel("")
plt.subplot(1, 3, 2)
sns.boxplot(yulu_raw, y='windspeed')
plt.xlabel("Wind Speed")
plt.ylabel("")
plt.subplot(1, 3, 3)
sns.boxplot(yulu_raw, y='count')
plt.xlabel("Count of Rental Bike")
plt.ylabel("")
plt.show()
```



1.5.4 Consolidated Data

[ ]:
```
df_yulu = yulu_raw.copy()
```

[ ]:
```
df_yulu.columns
```

[ ]:
```
Index(['datetime', 'season', 'holiday', 'workingday', 'weather', 'temp',
       'atemp', 'humidity', 'windspeed', 'casual', 'registered', 'count'],
      dtype='object')
```

Time Slots of the Day

[ ]:
```
df_yulu["time_slot"] = yulu_raw['datetime'].dt.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")
       )
      )
     )
)
```

Set Name to Categorical Columns

[ ]:
```
def season_convert(x):
    return 'Spring' if (int(x) == 1) else ("Summer" if (int(x) == 2) else␣
 ↪("Fall" if (int(x) == 3) else "Winter"))


def holiday_convert(x):
    return 'Yes' if (int(x) == 1) else 'No'
```

```
def workingday_convert(x):
    return 'Yes' if (int(x) == 1) else 'No'


def weather_convert(x):
    return 'Clear or Cloudy' if (int(x) == 1) else ("Mist" if (int(x) == 2)␣
 ↪else ("Light Rain or Snow" if (int(x) == 3) else "Heavy Rain or Snow"))
```

[ ]:
```
df_yulu['season'] = yulu_raw.apply(
    lambda row: season_convert(row['season']), axis=1)
df_yulu['holiday'] = yulu_raw.apply(
    lambda row: holiday_convert(row['holiday']), axis=1)
df_yulu['workingday'] = yulu_raw.apply(
    lambda row: workingday_convert(row['workingday']), axis=1)
df_yulu['weather'] = yulu_raw.apply(
    lambda row: weather_convert(row['weather']), axis=1)
```

[ ]:
```
df_yulu = df_yulu[['datetime', 'time_slot', 'season', 'holiday', 'workingday',␣
 ↪'weather', 'temp',
                    'atemp', 'humidity', 'windspeed', 'casual', 'registered',␣
 ↪'count']]
df_yulu.head(5)
```

[ ]:
```
             datetime time_slot  season holiday workingday          weather  \
0 2011-01-01 00:00:00      Dawn  Spring      No         No  Clear or Cloudy
1 2011-01-01 01:00:00      Dawn  Spring      No         No  Clear or Cloudy
2 2011-01-01 02:00:00      Dawn  Spring      No         No  Clear or Cloudy
3 2011-01-01 03:00:00      Dawn  Spring      No         No  Clear or Cloudy
4 2011-01-01 04:00:00      Dawn  Spring      No         No  Clear or Cloudy

   temp   atemp  humidity  windspeed  casual  registered  count
0  9.84  14.395        81        0.0       3          13     16
1  9.02  13.635        80        0.0       8          32     40
2  9.02  13.635        80        0.0       5          27     32
3  9.84  14.395        75        0.0       3          10     13
4  9.84  14.395        75        0.0       0           1      1
```

Binning of Countinuous Columns

[ ]:
```
df_yulu_grouped = df_yulu.copy()
```

[ ]:
```
def temp_categorise(df):
    df["temp_group"] = pd.cut(x=df['temp'], bins=[-5, 0, 5, 10, 15, 20, 25, 30,␣
 ↪35, 40, 45],
                              labels=["< 0.0", '0.1 to 5.0', "5.1 to 10.0", "10.
 ↪1 to 15.0", "15.1 to 20.0", "20.1 to 25.0", "25.1 to 30.0", "30.1 to 35.0",␣
 ↪"35.1 to 40.0", "> 40.0"])
    df['temp_group'] = df['temp_group'].astype('category')
```

```
    return df


def atemp_categorise(df):
    df["atemp_group"] = pd.cut(x=df['atemp'], bins=[-5, 0, 5, 10, 15, 20, 25,␣
 ↪30, 35, 40, 45, 50],
                               labels=["< 0.0", '0.1 to 5.0', "5.1 to 10.0",␣
 ↪"10.1 to 15.0", "15.1 to 20.0", "20.1 to 25.0", "25.1 to 30.0", "30.1 to 35.
 ↪0", "35.1 to 40.0", "40.1 to 45.0", "> 45.0"])
    df['atemp_group'] = df['atemp_group'].astype('category')
    return df


def humidity_categorise(df):
    df["humidity_group"] = pd.cut(x=df['humidity'], bins=[-1, 10, 20, 30, 40,␣
 ↪50, 60, 70, 80, 90, 100],
                                  labels=['0.0-10.0 %', '10.1-20.0 %', '20.1-30.
 ↪0 %', '30.1-40.0 %', '40.1-50.0 %', '50.1-60.0 %', '60.1-70.0 %', '70.1-80.0␣
 ↪%', '80.1-90.0 %', '90.1-100.0 %'])
    df['humidity_group'] = df['humidity_group'].astype('category')
    return df


def windspeed_categorise(df):
    df["windspeed_group"] = pd.cut(x=df['windspeed'], bins=[-1.0, 10, 20, 30,␣
 ↪40, 50, 60],
                                   labels=['0.0-10.0', '10.1-20.0', '20.1-30.
 ↪0', '30.1-40.0', '40.1-50.0', '50.1-60.0'])
    df['windspeed_group'] = df['windspeed_group'].astype('category')
    return df


def casualUser_categorise(df):
    df["casual_group"] = pd.cut(x=df[df['casual'] != 0]['casual'], bins=[-1.0,␣
 ↪100, 200, 300, 400],
                                labels=['0-100', '101-200', '201-300',␣
 ↪'301-400'])
    df['casual_group'] = df['casual_group'].astype('category')
    return df


def registeredUser_categorise(df):
    df["registered_group"] = pd.cut(x=df[df['registered'] != 0]['registered'],␣
 ↪bins=[-1.0, 100, 200, 300, 400, 500, 600, 700],
                                    labels=['0-100', '101-200', '201-300',␣
 ↪'301-400', '401-500', '501-600', '601-700'])
```

```python
    df['registered_group'] = df['registered_group'].astype('category')
    return df


def count_categorise(df):
    df["count_group"] = pd.cut(x=df['count'], bins=[-1.0, 100, 200, 300, 400,␣
 ↪500, 600, 700],
                                labels=['0-100', '101-200', '201-300',␣
 ↪'301-400', '401-500', '501-600', '601-700'])
    df['count_group'] = df['count_group'].astype('category')
    return df
```

[ ]:
```python
df_yulu_grouped = temp_categorise(df_yulu_grouped)
df_yulu_grouped = atemp_categorise(df_yulu_grouped)
df_yulu_grouped = humidity_categorise(df_yulu_grouped)
df_yulu_grouped = windspeed_categorise(df_yulu_grouped)
df_yulu_grouped = casualUser_categorise(df_yulu_grouped)
df_yulu_grouped = registeredUser_categorise(df_yulu_grouped)
df_yulu_grouped = count_categorise(df_yulu_grouped)
```

[ ]:
```python
df_yulu_grouped.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 10352 entries, 0 to 10885
Data columns (total 20 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   datetime          10352 non-null  datetime64[ns]
 1   time_slot         10352 non-null  object
 2   season            10352 non-null  object
 3   holiday           10352 non-null  object
 4   workingday        10352 non-null  object
 5   weather           10352 non-null  object
 6   temp              10352 non-null  float64
 7   atemp             10352 non-null  float64
 8   humidity          10352 non-null  int64
 9   windspeed         10352 non-null  float64
 10  casual            10352 non-null  int64
 11  registered        10352 non-null  int64
 12  count             10352 non-null  int64
 13  temp_group        10352 non-null  category
 14  atemp_group       10352 non-null  category
 15  humidity_group    10352 non-null  category
 16  windspeed_group   10352 non-null  category
 17  casual_group      9391 non-null   category
 18  registered_group  10339 non-null  category
 19  count_group       10352 non-null  category
dtypes: category(7), datetime64[ns](1), float64(3), int64(4), object(5)
```

```
memory usage: 1.2+ MB
```

Updated Yulu Data Set

[ ]:
```
df_yulu.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 10352 entries, 0 to 10885
Data columns (total 13 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   datetime    10352 non-null  datetime64[ns]
 1   time_slot   10352 non-null  object
 2   season      10352 non-null  object
 3   holiday     10352 non-null  object
 4   workingday  10352 non-null  object
 5   weather     10352 non-null  object
 6   temp        10352 non-null  float64
 7   atemp       10352 non-null  float64
 8   humidity    10352 non-null  int64
 9   windspeed   10352 non-null  float64
 10  casual      10352 non-null  int64
 11  registered  10352 non-null  int64
 12  count       10352 non-null  int64
dtypes: datetime64[ns](1), float64(3), int64(4), object(5)
memory usage: 1.1+ MB
```

1.5.5 Conversion of categorical attributes to 'category'.

[ ]:
```
df_yulu['time_slot'] = df_yulu['time_slot'].astype('category')
df_yulu['season'] = df_yulu['season'].astype('category')
df_yulu['holiday'] = df_yulu['holiday'].astype('category')
df_yulu['workingday'] = df_yulu['workingday'].astype('category')
df_yulu['weather'] = df_yulu['weather'].astype('category')
df_yulu.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 10352 entries, 0 to 10885
Data columns (total 13 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   datetime    10352 non-null  datetime64[ns]
 1   time_slot   10352 non-null  category
 2   season      10352 non-null  category
 3   holiday     10352 non-null  category
 4   workingday  10352 non-null  category
 5   weather     10352 non-null  category
 6   temp        10352 non-null  float64
 7   atemp       10352 non-null  float64
```

```
 8   humidity    10352 non-null  int64
 9   windspeed   10352 non-null  float64
 10  casual      10352 non-null  int64
 11  registered  10352 non-null  int64
 12  count       10352 non-null  int64
dtypes: category(5), datetime64[ns](1), float64(3), int64(4)
memory usage: 779.3 KB
```

1.5.6 Statitical Summary

Descriptive Statistits

[ ]:
```
df_yulu[['time_slot', 'season', 'holiday', 'workingday', 'weather']].describe()
```

[ ]:
```
       time_slot  season holiday workingday          weather
count      10352   10352   10352      10352            10352
unique         5       4       2          2                4
top         Noon  Winter      No        Yes  Clear or Cloudy
freq        3044    2645   10049       6992             6821
```

[ ]:
```
df_yulu[['temp', 'atemp', 'humidity', 'windspeed',
         'casual', 'registered', 'count']].describe()
```

[ ]:
```
               temp         atemp     humidity     windspeed        casual  \
count  10352.000000  10352.000000  10352.000000  10352.000000  10352.000000
mean      20.114397     23.551453     62.623454     12.267712     34.136785
std        7.784824      8.443686     18.869422      7.458563     47.224470
min        0.820000      0.760000      8.000000      0.000000      0.000000
25%       13.940000     16.665000     48.000000      7.001500      4.000000
50%       20.500000     24.240000     62.000000     11.001400     16.000000
75%       26.240000     31.060000     78.000000     16.997900     46.000000
max       41.000000     45.455000    100.000000     31.000900    355.000000

         registered         count
count  10352.000000  10352.000000
mean     142.230004    176.366789
std      127.388471    156.952045
min        0.000000      1.000000
25%       34.000000     40.000000
50%      114.000000    139.000000
75%      212.000000    271.000000
max      629.000000    649.000000
```

Unique Count

[ ]:
```
df_yulu.nunique()
```

[ ]:
```
datetime      10352
time_slot         5
season            4
```

```
holiday            2
workingday         2
weather            4
temp              49
atemp             60
humidity          87
windspeed         16
casual           285
registered       586
count            643
dtype: int64
```

Mean

[ ]:
```python
df_yulu[['temp', 'atemp', 'humidity', 'windspeed',
         'casual', 'registered', 'count']].mean()
```

[ ]:
```
temp           20.114397
atemp          23.551453
humidity       62.623454
windspeed      12.267712
casual         34.136785
registered    142.230004
count         176.366789
dtype: float64
```

Median

[ ]:
```python
df_yulu[['temp', 'atemp', 'humidity', 'windspeed',
         'casual', 'registered', 'count']].median()
```

[ ]:
```
temp           20.5000
atemp          24.2400
humidity       62.0000
windspeed      11.0014
casual         16.0000
registered    114.0000
count         139.0000
dtype: float64
```

Mode

[ ]:
```python
for i in df_yulu.columns:
    print(i, ':', df_yulu[i].mode()[0])
```

```
datetime : 2011-01-01 00:00:00
time_slot : Noon
season : Winter
holiday : No
workingday : Yes
weather : Clear or Cloudy
```

```
temp : 14.76
atemp : 31.06
humidity : 88
windspeed : 0.0
casual : 0
registered : 3
count : 5
```

1.6 Uni Variate Analysis

1.6.1 Time Slot

[ ]:
```
df_yulu['time_slot'].unique()
```

[ ]:
```
['Dawn', 'Early Morning', 'Noon', 'Late Evening', 'Night']
Categories (5, object): ['Dawn', 'Early Morning', 'Late Evening', 'Night',
'Noon']
```

[ ]:
```
df_yulu['time_slot'].value_counts()
```

[ ]:
```
Noon             3044
Dawn             2207
Early Morning    2168
Late Evening     2035
Night             898
Name: time_slot, dtype: int64
```

[ ]:
```
df_yulu['time_slot'].value_counts(normalize=True)*100
```

[ ]:
```
Noon             29.404946
Dawn             21.319552
Early Morning    20.942813
Late Evening     19.658037
Night             8.674652
Name: time_slot, dtype: float64
```

Statitical Analysis

[ ]:
```
df_yulu['time_slot'].describe()
```

[ ]:
```
count      10352
unique         5
top         Noon
freq        3044
Name: time_slot, dtype: object
```

[ ]:
```
df_yulu['time_slot'].mode()[0]
```

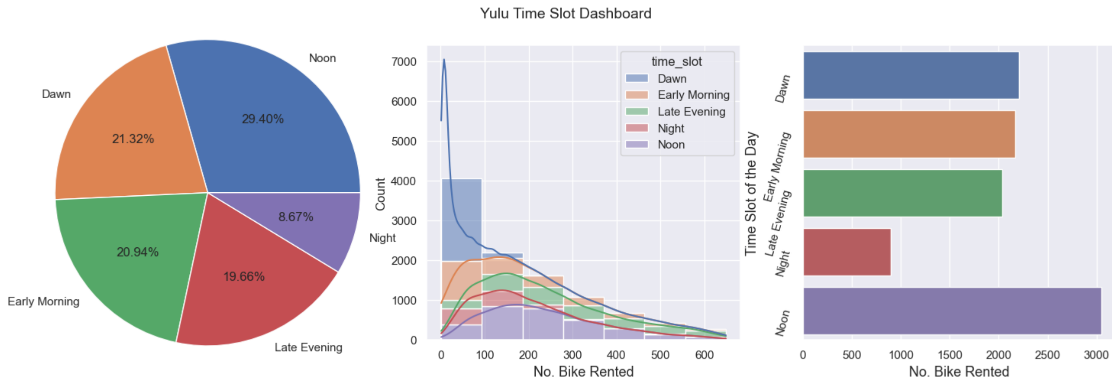
[ ]:
```
'Noon'
```

[ ]:
```
df_yulu.groupby('time_slot')["count"].describe()
```

[ ]:
```
               count        mean         std   min     25%    50%     75%  \
time_slot
Dawn          2207.0   26.500227   33.237161   1.0    5.00   12.0   34.00
Early Morning 2168.0  161.540129  154.698997   1.0   29.75  112.0  258.25
Late Evening  2035.0  286.374447  157.208013  11.0  165.00  262.0  397.00
Night          898.0  112.462138   65.169543   4.0   61.25  105.0  152.00
Noon          3044.0  240.893890  135.166752   3.0  138.00  216.0  323.00

                 max
time_slot
Dawn           283.0
Early Morning  649.0
Late Evening   649.0
Night          502.0
Noon           647.0
```

Plot the Graph

[ ]:
```
plt.figure(figsize=(18, 5)).suptitle(
    "Yulu Time Slot Dashboard", fontsize=14)

plt.subplot(1, 3, 1)
plt.pie(df_yulu['time_slot'].value_counts().values, labels=df_yulu['time_slot'].
 ↪value_counts(
).index, radius=1.3, autopct='%1.2f%%',)  # type: ignore

plt.subplot(1, 3, 2)
sns.histplot(data=df_yulu, x='count', bins=7,
             hue='time_slot', kde=True, multiple="stack")
plt.xlabel('No. Bike Rented', fontsize=13)
plt.ylabel("Count", fontsize=12)

plt.subplot(1, 3, 3)
sns.countplot(df_yulu, y='time_slot')
plt.xlabel("No. Bike Rented", fontsize=13)
plt.ylabel('Time Slot of the Day', fontsize=13)
plt.xticks(rotation=0, fontsize=11)
plt.yticks(rotation=75, fontsize=11)

plt.show()
```



1.6.2 Season

[ ]:
```
df_yulu['season'].unique()
```

[ ]:
```
['Spring', 'Summer', 'Fall', 'Winter']
Categories (4, object): ['Fall', 'Spring', 'Summer', 'Winter']
```

[ ]:
```
df_yulu['season'].value_counts()
```

[ ]:
```
Winter    2645
Fall      2598
Summer    2579
Spring    2530
Name: season, dtype: int64
```

[ ]:
```
df_yulu['season'].value_counts(normalize=True)*100
```

[ ]:
```
Winter    25.550618
Fall      25.096600
Summer    24.913060
Spring    24.439722
Name: season, dtype: float64
```

Statitical Analysis

[ ]:
```
df_yulu['season'].describe()
```

[ ]:
```
count      10352
unique         4
top       Winter
freq        2645
Name: season, dtype: object
```

[ ]:
```
df_yulu['season'].mode()[0]
```

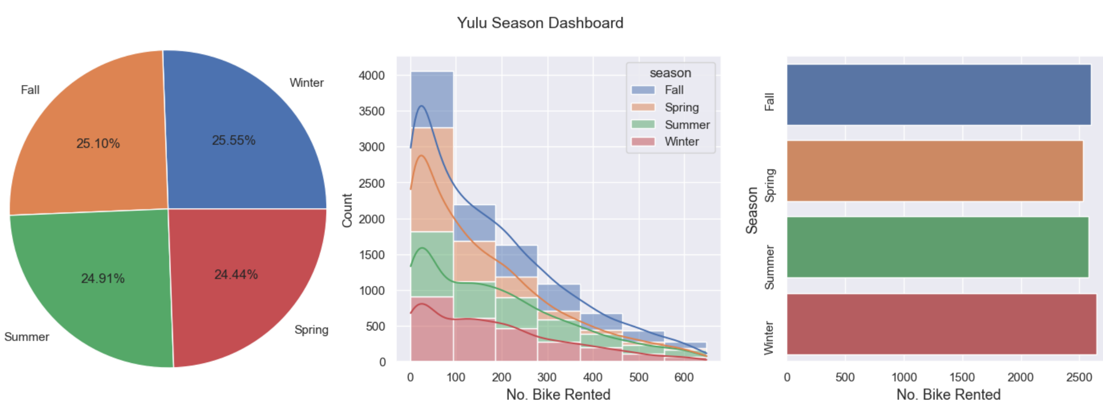
```
[ ]: 'Winter'
```

```
[ ]: df_yulu.groupby('season')["count"].describe()
```

```
[ ]:          count        mean         std  min   25%    50%    75%    max
     season
     Fall    2598.0  210.633564  164.663413  1.0  58.0  185.5  323.0  649.0
     Spring  2530.0  112.774308  116.946626  1.0  23.0   78.0  161.0  648.0
     Summer  2579.0  195.945328  167.073444  1.0  43.0  165.0  300.0  647.0
     Winter  2645.0  184.446503  155.068939  1.0  48.0  154.0  278.0  648.0
```

Plot the Graph

```
[ ]: plt.figure(figsize=(18, 5)).suptitle(
         "Yulu Season Dashboard", fontsize=14)

     plt.subplot(1, 3, 1)
     plt.pie(df_yulu['season'].value_counts().values, labels=df_yulu['season'].
      ↪value_counts(
     ).index, radius=1.3, autopct='%1.2f%%',)  # type: ignore

     plt.subplot(1, 3, 2)
     sns.histplot(data=df_yulu, x='count', bins=7,
                  hue='season', kde=True, multiple="stack")
     plt.xlabel('No. Bike Rented', fontsize=13)
     plt.ylabel("Count", fontsize=12)

     plt.subplot(1, 3, 3)
     sns.countplot(df_yulu, y='season')
     plt.xlabel("No. Bike Rented", fontsize=13)
     plt.ylabel('Season', fontsize=13)
     plt.xticks(rotation=0, fontsize=11)
     plt.yticks(rotation=90, fontsize=11)

     plt.show()
```



1.6.3 Holiday

[ ]:
```
df_yulu['holiday'].unique()
```

[ ]:
```
['No', 'Yes']
Categories (2, object): ['No', 'Yes']
```

[ ]:
```
df_yulu['holiday'].value_counts()
```

[ ]:
```
No     10049
Yes      303
Name: holiday, dtype: int64
```

[ ]:
```
df_yulu['holiday'].value_counts(normalize=True)*100
```

[ ]:
```
No     97.073029
Yes     2.926971
Name: holiday, dtype: float64
```

Statitical Analysis

[ ]:
```
df_yulu['holiday'].describe()
```

[ ]:
```
count     10352
unique        2
top          No
freq      10049
Name: holiday, dtype: object
```

[ ]:
```
df_yulu['holiday'].mode()[0]
```

[ ]:
```
'No'
```

[ ]:
```
df_yulu.groupby('holiday')["count"].describe()
```

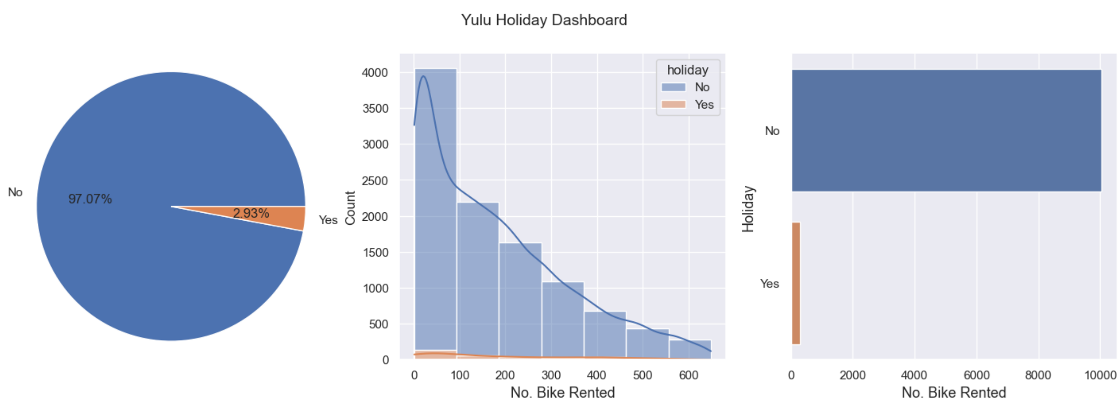
[ ]:

	count	mean	std	min	25%	50%	75%	max
holiday								
No	10049.0	176.178426	156.702198	1.0	40.0	139.0	270.0	649.0
Yes	303.0	182.613861	165.174199	1.0	36.5	126.0	308.0	597.0

Plot the Graph

[ ]:
```
plt.figure(figsize=(18, 5)).suptitle(
    "Yulu Holiday Dashboard", fontsize=14)

plt.subplot(1, 3, 1)
plt.pie(df_yulu['holiday'].value_counts().values, labels=df_yulu['holiday'].
 ↪value_counts(
```

```
).index, radius=1.1, autopct='%1.2f%%',)  # type: ignore

plt.subplot(1, 3, 2)
sns.histplot(data=df_yulu, x='count', bins=7,
             hue='holiday', kde=True, multiple="stack")
plt.xlabel('No. Bike Rented', fontsize=13)
plt.ylabel("Count", fontsize=12)

plt.subplot(1, 3, 3)
sns.countplot(df_yulu, y='holiday')
plt.xlabel("No. Bike Rented", fontsize=13)
plt.ylabel('Holiday', fontsize=13)
plt.xticks(rotation=0, fontsize=11)
plt.yticks(rotation=0, fontsize=11)

plt.show()
```



1.6.4 Working Day

[ ]:
```
df_yulu['workingday'].unique()
```

[ ]:
```
['No', 'Yes']
Categories (2, object): ['No', 'Yes']
```

[ ]:
```
df_yulu['workingday'].value_counts()
```

[ ]:
```
Yes    6992
No     3360
Name: workingday, dtype: int64
```

[ ]:
```
df_yulu['workingday'].value_counts(normalize=True)*100
```

```
[ ]: Yes    67.542504
     No     32.457496
     Name: workingday, dtype: float64
```

Statitical Analysis

```
[ ]: df_yulu['workingday'].describe()
```

```
[ ]: count      10352
     unique         2
     top          Yes
     freq        6992
     Name: workingday, dtype: object
```

```
[ ]: df_yulu['workingday'].mode()[0]
```

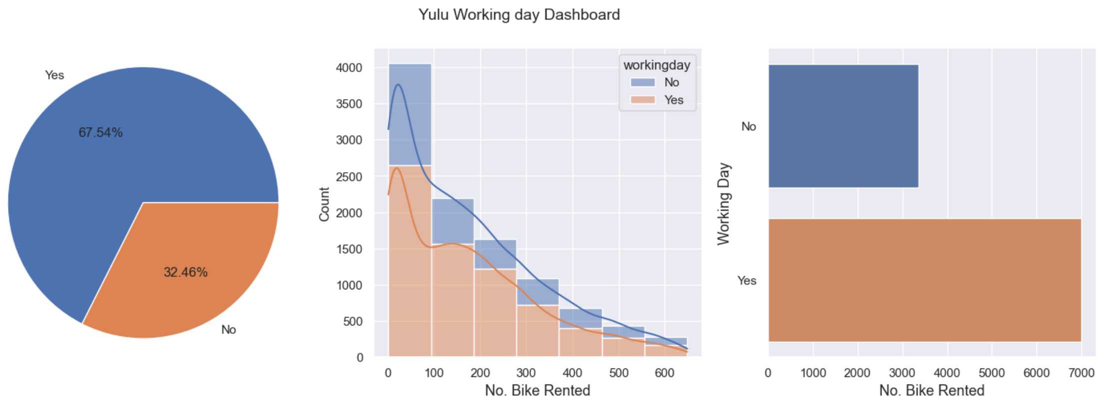
```
[ ]: 'Yes'
```

```
[ ]: df_yulu.groupby('workingday')["count"].describe()
```

```
[ ]:              count        mean         std  min   25%    50%    75%    max
     workingday
     No          3360.0  182.189881  165.030731  1.0  43.0  125.0  297.0  648.0
     Yes         6992.0  173.568507  152.851331  1.0  38.0  144.0  263.0  649.0
```

Plot the Graph

```
[ ]: plt.figure(figsize=(18, 5)).suptitle(
         "Yulu Working day Dashboard", fontsize=14)

     plt.subplot(1, 3, 1)
     plt.pie(df_yulu['workingday'].value_counts().values,␣
      ↪labels=df_yulu['workingday'].value_counts(
     ).index, radius=1.1, autopct='%1.2f%%',)  # type: ignore

     plt.subplot(1, 3, 2)
     sns.histplot(data=df_yulu, x='count', bins=7,
                  hue='workingday', kde=True, multiple="stack")
     plt.xlabel('No. Bike Rented', fontsize=13)
     plt.ylabel("Count", fontsize=12)

     plt.subplot(1, 3, 3)
     sns.countplot(df_yulu, y='workingday')
     plt.xlabel("No. Bike Rented", fontsize=13)
     plt.ylabel('Working Day', fontsize=13)
     plt.xticks(rotation=0, fontsize=11)
     plt.yticks(rotation=0, fontsize=11)

     plt.show()
```



1.6.5 Weather

[ ]:
```
df_yulu['weather'].unique()
```

[ ]:
```
['Clear or Cloudy', 'Mist', 'Light Rain or Snow', 'Heavy Rain or Snow']
Categories (4, object): ['Clear or Cloudy', 'Heavy Rain or Snow', 'Light Rain or
Snow', 'Mist']
```

[ ]:
```
df_yulu['weather'].value_counts()
```

[ ]:
```
Clear or Cloudy       6821
Mist                  2733
Light Rain or Snow     797
Heavy Rain or Snow       1
Name: weather, dtype: int64
```

[ ]:
```
df_yulu['weather'].value_counts(normalize=True)*100
```

[ ]:
```
Clear or Cloudy       65.890649
Mist                  26.400696
Light Rain or Snow     7.698995
Heavy Rain or Snow     0.009660
Name: weather, dtype: float64
```

Statitical Analysis

[ ]:
```
df_yulu['weather'].describe()
```

[ ]:
```
count                 10352
unique                    4
top         Clear or Cloudy
freq                   6821
Name: weather, dtype: object
```

[ ]:
```
df_yulu['weather'].mode()[0]
```

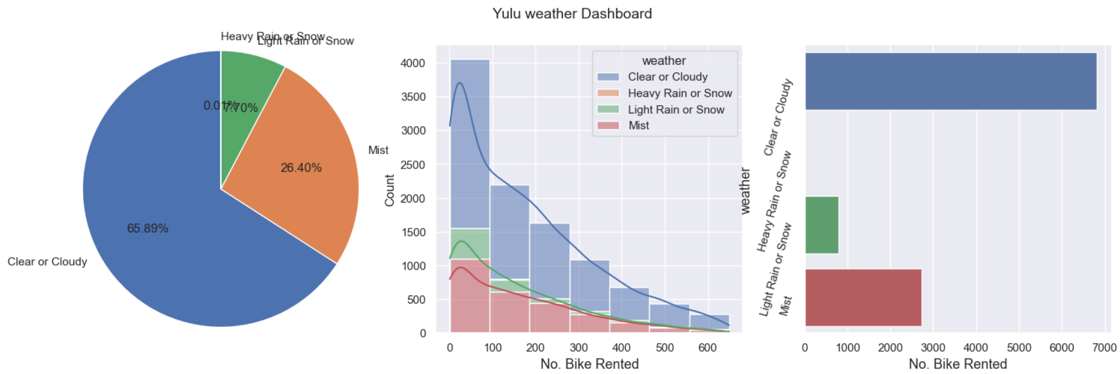
[ ]:
```
'Clear or Cloudy'
```

[ ]:
```
df_yulu.groupby('weather')["count"].describe()
```

[ ]:
```
                     count        mean         std    min    25%    50%  \
weather
Clear or Cloudy     6821.0  187.822607  162.329150    1.0   44.0  153.0
Heavy Rain or Snow     1.0  164.000000         NaN  164.0  164.0  164.0
Light Rain or Snow   797.0  113.562108  121.648539    1.0   24.0   73.0
Mist                2733.0  166.095134  147.163209    1.0   39.0  130.0

                      75%    max
weather
Clear or Cloudy     288.0  649.0
Heavy Rain or Snow  164.0  164.0
Light Rain or Snow  158.0  646.0
Mist                254.0  648.0
```

Plot the Graph

[ ]:
```
plt.figure(figsize=(18, 5)).suptitle(
    "Yulu weather Dashboard", fontsize=14)

plt.subplot(1, 3, 1)
plt.pie(df_yulu['weather'].value_counts().values, labels=df_yulu['weather'].
 ↪value_counts(
).index, radius=1.2, autopct='%1.2f%%', rotatelabels=False, startangle=90,␣
 ↪counterclock=True)  # type: ignore

plt.subplot(1, 3, 2)
sns.histplot(data=df_yulu, x='count', bins=7,
             hue='weather', kde=True, multiple="stack")
plt.xlabel('No. Bike Rented', fontsize=13)
plt.ylabel("Count", fontsize=12)

plt.subplot(1, 3, 3)
sns.countplot(df_yulu, y='weather')
plt.xlabel("No. Bike Rented", fontsize=13)
plt.ylabel('weather', fontsize=13)
plt.xticks(rotation=0, fontsize=11)
plt.yticks(rotation=75, fontsize=11)

plt.show()
```



1.6.6 Temperatue

[ ]:
```
df_yulu['temp'].unique().shape
```

[ ]:
```
(49,)
```

[ ]:
```
df_yulu['temp'].value_counts().head(5)
```

[ ]:
```
14.76    452
26.24    425
28.70    400
13.94    393
18.86    389
Name: temp, dtype: int64
```

[ ]:
```
df_yulu['temp'].value_counts(normalize=True).head(5)*100
```

[ ]:
```
14.76    4.366306
26.24    4.105487
28.70    3.863988
13.94    3.796368
18.86    3.757728
Name: temp, dtype: float64
```

Statitical Analysis

[ ]:
```
df_yulu['temp'].describe()
```

[ ]:
```
count    10352.000000
mean        20.114397
std          7.784824
min          0.820000
25%         13.940000
50%         20.500000
```

```
75%          26.240000
max          41.000000
Name: temp, dtype: float64
```

[ ]:
```
df_yulu['temp'].mean()
```

[ ]:
```
20.1143972179289
```

[ ]:
```
df_yulu['temp'].median()
```

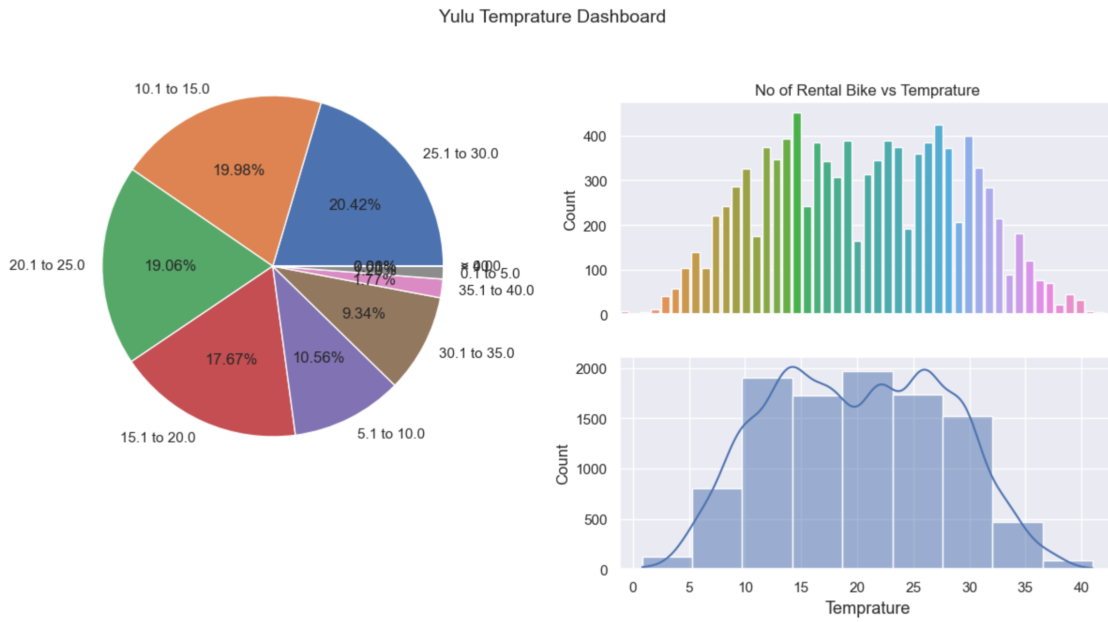
[ ]:
```
20.5
```

[ ]:
```
df_yulu_grouped['temp_group'].mode()[0]
```

[ ]:
```
'25.1 to 30.0'
```

Plot the Graph

[ ]:
```
plt.figure(figsize=(15, 10)).suptitle("Yulu Temprature Dashboard", fontsize=14)

plt.subplot(2, 2, 1)
plt.pie(df_yulu_grouped['temp_group'].value_counts().values,␣
 ↪labels=df_yulu_grouped['temp_group'].value_counts(
).index, radius=1.3, autopct='%1.2f%%')  # type: ignore

plt.subplot(3, 2, 2)
sns.countplot(df_yulu, x='temp')
plt.title('No of Rental Bike vs Temprature', fontsize=12)
plt.ylabel('Count', fontsize=12)
plt.xlabel('', fontsize=11)
plt.xticks([], fontsize=11)
plt.yticks(rotation=0, fontsize=11)

plt.subplot(3, 2, 4)
# sns.countplot(df_yulu_grouped, x='temp_group')
# plt.ylabel('Count', fontsize=12)
# plt.xlabel('', fontsize=11)
# plt.xticks(rotation=30, fontsize=11)
# plt.yticks(rotation=0, fontsize=11)

sns.histplot(data=df_yulu, x='temp', bins=9, kde=True, multiple="stack")
plt.xlabel('Temprature', fontsize=13)
plt.ylabel("Count", fontsize=12)

plt.show()
```



1.6.7 Feeling Temperature

[ ]:
```
df_yulu['atemp'].unique().shape
```

[ ]:
```
(60,)
```

[ ]:
```
df_yulu['atemp'].value_counts().head(5)
```

[ ]:
```
31.060    607
25.760    409
22.725    389
20.455    385
16.665    375
Name: atemp, dtype: int64
```

[ ]:
```
df_yulu['atemp'].value_counts(normalize=True).head(5)*100
```

[ ]:
```
31.060    5.863601
25.760    3.950927
22.725    3.757728
20.455    3.719088
16.665    3.622488
Name: atemp, dtype: float64
```

Statitical Analysis

[ ]:
```
df_yulu['atemp'].describe()
```

[ ]:

```
count    10352.000000
mean        23.551453
std          8.443686
min          0.760000
25%         16.665000
50%         24.240000
75%         31.060000
max         45.455000
Name: atemp, dtype: float64
```

[ ]:

```
df_yulu['atemp'].mean()
```

[ ]:

```
23.551453342349305
```

[ ]:

```
df_yulu['atemp'].median()
```

[ ]:

```
24.24
```

[ ]:

```
df_yulu_grouped['atemp_group'].mode()[0]
```

[ ]:

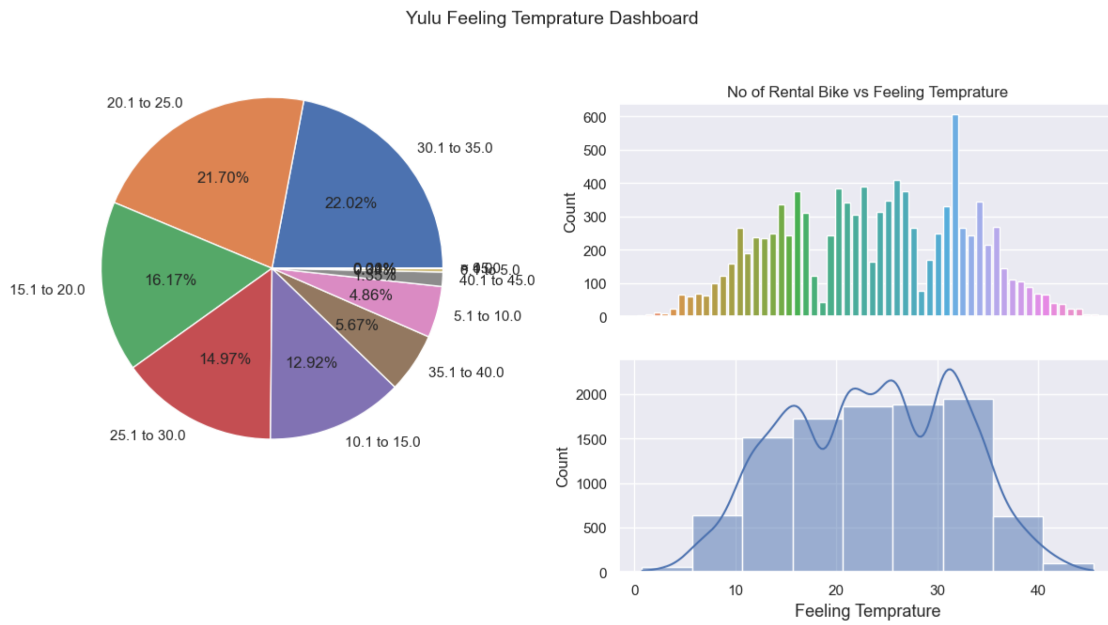
```
'30.1 to 35.0'
```

Plot the Graph

[ ]:

```
plt.figure(figsize=(15, 10)).suptitle(
    "Yulu Feeling Temprature Dashboard", fontsize=14)

plt.subplot(2, 2, 1)
plt.pie(df_yulu_grouped['atemp_group'].value_counts().values,␣
 ↪labels=df_yulu_grouped['atemp_group'].value_counts(
).index, radius=1.3, autopct='%1.2f%%')  # type: ignore

plt.subplot(3, 2, 2)
sns.countplot(df_yulu, x='atemp')
plt.title('No of Rental Bike vs Feeling Temprature', fontsize=12)
plt.ylabel('Count', fontsize=12)
plt.xlabel('', fontsize=11)
plt.xticks([], fontsize=11)
plt.yticks(rotation=0, fontsize=11)

plt.subplot(3, 2, 4)
# sns.countplot(df_yulu_grouped, x='atemp_group')
# plt.ylabel('Count', fontsize=12)
# plt.xlabel('', fontsize=11)
# plt.xticks(rotation=30, fontsize=10)
# plt.yticks(rotation=0, fontsize=11)

sns.histplot(data=df_yulu, x='atemp', bins=9, kde=True, multiple="stack")
```

```
plt.xlabel('Feeling Temprature', fontsize=13)
plt.ylabel("Count", fontsize=12)

plt.show()
```



1.6.8 Humidity

[ ]:
```
df_yulu['humidity'].unique().shape
```

[ ]:
```
(87,)
```

[ ]:
```
df_yulu['humidity'].value_counts().head(5)
```

[ ]:
```
88    354
94    320
83    309
87    283
70    250
Name: humidity, dtype: int64
```

[ ]:
```
df_yulu['humidity'].value_counts(normalize=True).head(5)*100
```

[ ]:
```
88    3.419629
94    3.091190
83    2.984930
87    2.733771
70    2.414992
```

```
Name: humidity, dtype: float64
```

Statitical Analysis

```
[ ]: df_yulu['humidity'].describe()
```

```
[ ]: count    10352.000000
     mean        62.623454
     std         18.869422
     min          8.000000
     25%         48.000000
     50%         62.000000
     75%         78.000000
     max        100.000000
     Name: humidity, dtype: float64
```

```
[ ]: df_yulu['humidity'].mean()
```

```
[ ]: 62.6234544049459
```

```
[ ]: df_yulu['humidity'].median()
```

```
[ ]: 62.0
```

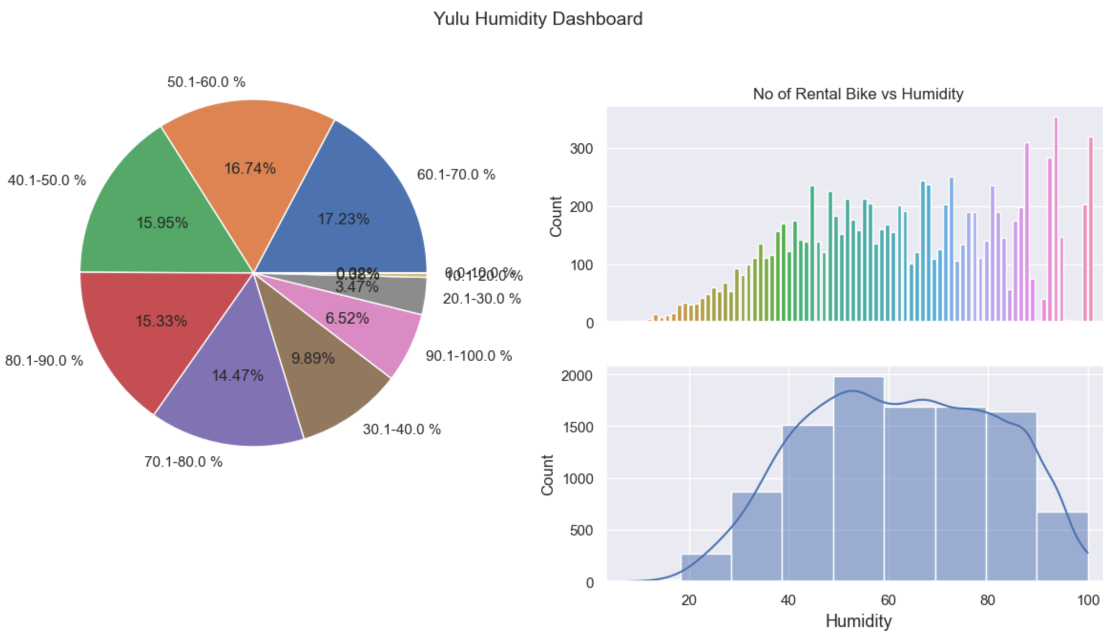
```
[ ]: df_yulu_grouped['humidity_group'].mode()[0]
```

```
[ ]: '60.1-70.0 %'
```

Plot the Graph

```
[ ]: plt.figure(figsize=(15, 10)).suptitle("Yulu Humidity Dashboard", fontsize=14)

plt.subplot(2, 2, 1)
plt.pie(df_yulu_grouped['humidity_group'].value_counts().values,␣
 ↪labels=df_yulu_grouped['humidity_group'].value_counts(
).index, radius=1.3, autopct='%1.2f%%')  # type: ignore

plt.subplot(3, 2, 2)
sns.countplot(df_yulu, x='humidity')
plt.title('No of Rental Bike vs Humidity', fontsize=12)
plt.ylabel('Count', fontsize=12)
plt.xlabel('', fontsize=11)
plt.xticks([], fontsize=11)
plt.yticks(rotation=0, fontsize=11)

plt.subplot(3, 2, 4)
# sns.countplot(df_yulu_grouped, x='humidity_group')
# plt.ylabel('Count', fontsize=12)
# plt.xlabel('', fontsize=11)
```

```
# plt.xticks(rotation=30, fontsize=11)
# plt.yticks(rotation=0, fontsize=11)

sns.histplot(data=df_yulu, x='humidity', bins=9, kde=True, multiple="stack")
plt.xlabel('Humidity', fontsize=13)
plt.ylabel("Count", fontsize=12)

plt.show()
```



1.6.9 Wind Speed

[ ]:
```
df_yulu['windspeed'].unique().shape
```

[ ]:
```
(16,)
```

[ ]:
```
df_yulu['windspeed'].value_counts().head(5)
```

[ ]:
```
0.0000     1285
8.9981     1093
11.0014    1019
12.9980    1013
7.0015     1009
Name: windspeed, dtype: int64
```

[ ]:
```
df_yulu['windspeed'].value_counts(normalize=True).head(5)*100
```

```
[ ]: 0.0000     12.413060
     8.9981     10.558346
     11.0014     9.843509
     12.9980     9.785549
     7.0015      9.746909
     Name: windspeed, dtype: float64
```

Statitical Analysis

```
[ ]: df_yulu['windspeed'].describe()
```

```
[ ]: count    10352.000000
     mean        12.267712
     std          7.458563
     min          0.000000
     25%          7.001500
     50%         11.001400
     75%         16.997900
     max         31.000900
     Name: windspeed, dtype: float64
```

```
[ ]: df_yulu['windspeed'].mean()
```

```
[ ]: 12.26771164992272
```

```
[ ]: df_yulu['windspeed'].median()
```

```
[ ]: 11.0014
```

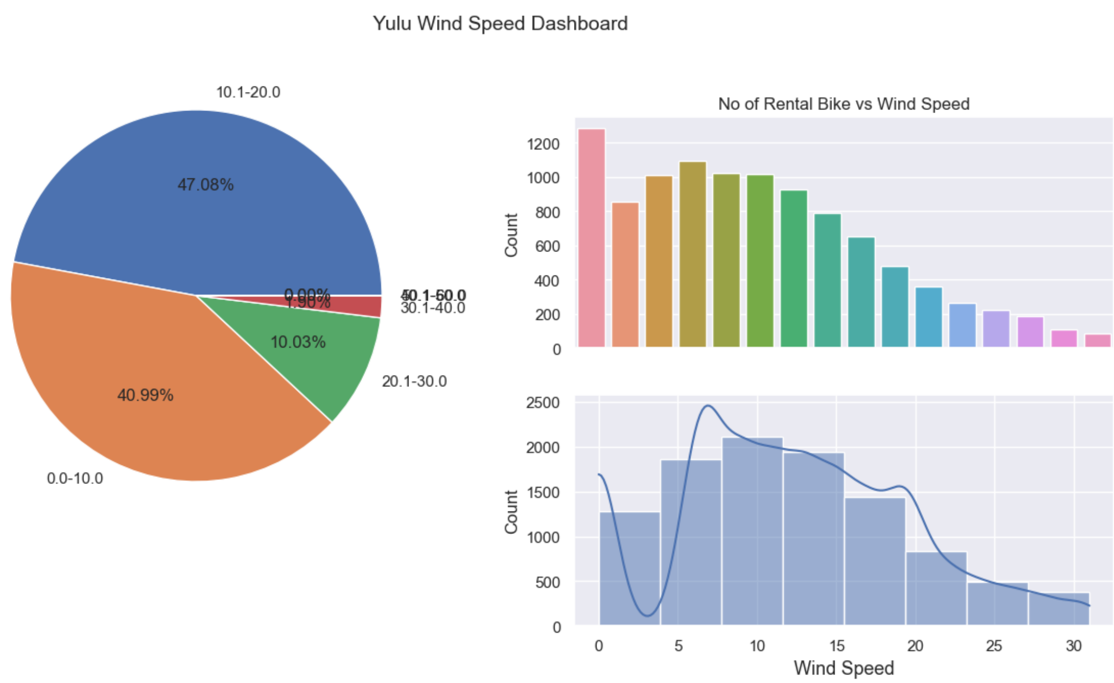
```
[ ]: df_yulu_grouped['windspeed_group'].mode()[0]
```

```
[ ]: '10.1-20.0'
```

Plot the Graph

```
[ ]: plt.figure(figsize=(15, 10)).suptitle("Yulu Wind Speed Dashboard", fontsize=14)

     plt.subplot(2, 2, 1)
     plt.pie(df_yulu_grouped['windspeed_group'].value_counts().values,␣
      ↪labels=df_yulu_grouped['windspeed_group'].value_counts(
     ).index, radius=1.3, autopct='%1.2f%%')  # type: ignore

     plt.subplot(3, 2, 2)
     sns.countplot(df_yulu, x='windspeed')
     plt.title('No of Rental Bike vs Wind Speed', fontsize=12)
     plt.ylabel('Count', fontsize=12)
     plt.xlabel('', fontsize=11)
     plt.xticks([], fontsize=11)
     plt.yticks(rotation=0, fontsize=11)
```

```python
plt.subplot(3, 2, 4)
# sns.countplot(df_yulu_grouped, x='windspeed_group')
# plt.ylabel('Count', fontsize=12)
# plt.xlabel('', fontsize=11)
# plt.xticks(rotation=30, fontsize=11)
# plt.yticks(rotation=0, fontsize=11)

sns.histplot(data=df_yulu, x='windspeed', bins=8, kde=True, multiple="stack")
plt.xlabel('Wind Speed', fontsize=13)
plt.ylabel("Count", fontsize=12)

plt.show()
```



1.6.10 Bike Rent by Casual user

[ ]:
```python
df_yulu['casual'].unique().shape
```

[ ]: (285,)

[ ]:
```python
df_yulu['casual'].value_counts().head(5)
```

[ ]:
```
0    961
1    645
2    475
```

```
3    433
4    343
Name: casual, dtype: int64
```

[ ]:
```
df_yulu['casual'].value_counts(normalize=True).head(5)*100
```

[ ]:
```
0    9.283230
1    6.230680
2    4.588485
3    4.182767
4    3.313369
Name: casual, dtype: float64
```

Statitical Analysis

[ ]:
```
df_yulu['casual'].describe()
```

[ ]:
```
count    10352.000000
mean        34.136785
std         47.224470
min          0.000000
25%          4.000000
50%         16.000000
75%         46.000000
max        355.000000
Name: casual, dtype: float64
```

[ ]:
```
df_yulu['casual'].mean()
```

[ ]:
```
34.13678516228748
```

[ ]:
```
df_yulu['casual'].median()
```

[ ]:
```
16.0
```

[ ]:
```
df_yulu_grouped['casual_group'].mode()[0]
```

[ ]:
```
'0-100'
```

Check for Outliers

[ ]:
```
check_outlier(df_yulu, 'casual')['upper']
```

[ ]:
```
{'list': 1173     144
 1174     149
 1175     124
 1311     126
 1312     174
         …
 10610    122
```

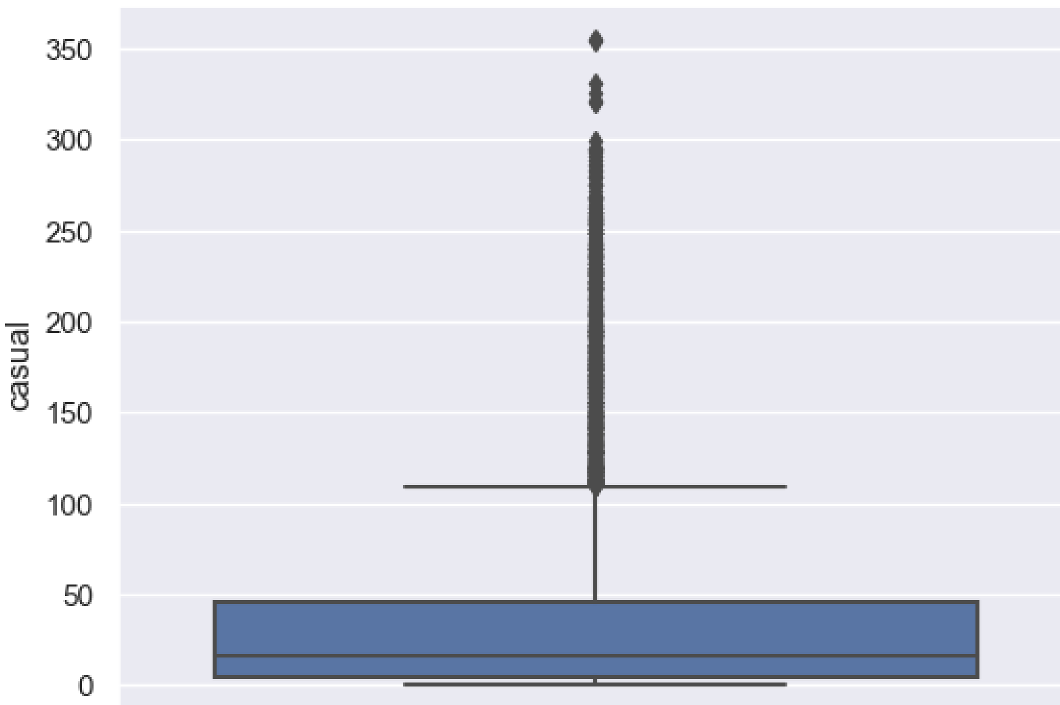
```
10611    148
10612    164
10613    167
10614    139
Name: casual, Length: 740, dtype: int64,
'length': 740}
```

[ ]:
```
check_outlier(df_yulu, 'casual')['lower']
```

[ ]:
```
{'list': Series([], Name: casual, dtype: int64), 'length': 0}
```

[ ]:
```
sns.boxplot(df_yulu, y='casual')
```

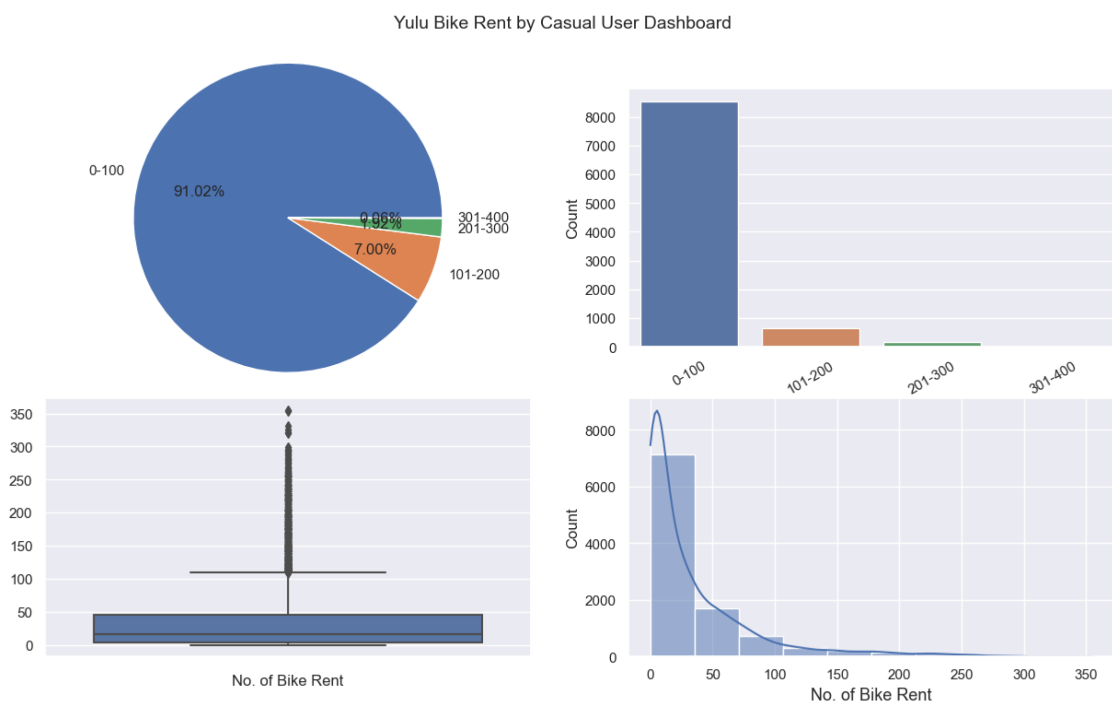
[ ]:
```
<Axes: ylabel='casual'>
```



Plot the Graph

[ ]:
```
plt.figure(figsize=(15, 8)).suptitle(
    "Yulu Bike Rent by Casual User Dashboard", fontsize=14)

plt.subplot(2, 2, 1)
plt.pie(df_yulu_grouped['casual_group'].value_counts().values,␣
 ↪labels=df_yulu_grouped['casual_group'].value_counts(
).index, radius=1.5, autopct='%1.2f%%')  # type: ignore
```

```
plt.subplot(2, 2, 3)
sns.boxplot(df_yulu, y="casual")
plt.xlabel('No. of Bike Rent', fontsize=12)
plt.ylabel('')

plt.subplot(2, 2, 2)
sns.countplot(df_yulu_grouped, x='casual_group')
plt.ylabel('Count', fontsize=12)
plt.xlabel('', fontsize=11)
plt.xticks(rotation=30, fontsize=11)
plt.yticks(fontsize=11)

plt.subplot(2, 2, 4)
sns.histplot(data=df_yulu, x='casual', bins=10, kde=True, multiple="stack")
plt.xlabel('No. of Bike Rent', fontsize=13)
plt.ylabel("Count", fontsize=12)

plt.show()
```



1.6.11 Bike Rent by Registered Users

```
[ ]: df_yulu['registered'].unique().shape
```

[ ]: (586,)

[ ]:
```
df_yulu['registered'].value_counts().head(5)
```

[ ]:
```
3    194
4    190
5    173
6    152
2    144
Name: registered, dtype: int64
```

[ ]:
```
df_yulu['registered'].value_counts(normalize=True).head(5)*100
```

[ ]:
```
3    1.874034
4    1.835394
5    1.671175
6    1.468315
2    1.391036
Name: registered, dtype: float64
```

Statitical Analysis

[ ]:
```
df_yulu['registered'].describe()
```

[ ]:
```
count    10352.000000
mean       142.230004
std        127.388471
min          0.000000
25%         34.000000
50%        114.000000
75%        212.000000
max        629.000000
Name: registered, dtype: float64
```

[ ]:
```
df_yulu['registered'].mean()
```

[ ]: 142.23000386398763

[ ]:
```
df_yulu['registered'].median()
```

[ ]: 114.0

[ ]:
```
df_yulu_grouped['registered_group'].mode()[0]
```

[ ]: '0-100'

Check for Outliers

[ ]:
```
check_outlier(df_yulu, 'registered')['upper']
```

[ ]: {'list': 1844   485
 1987   539
 2011   532
 2012   480
 2059   540
         …
 10832   493
 10855   533
 10856   512
 10879   536
 10880   546
 Name: registered, Length: 231, dtype: int64,
 'length': 231}

[ ]:
```
check_outlier(df_yulu, 'registered')['lower']
```

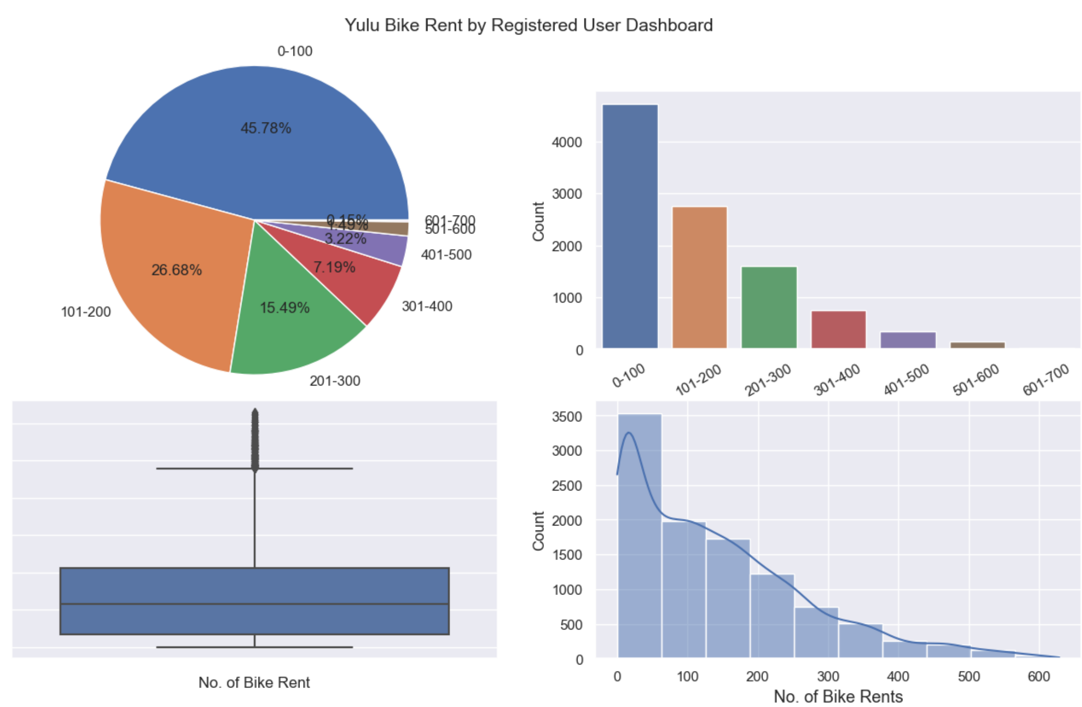
[ ]: {'list': Series([], Name: registered, dtype: int64), 'length': 0}

[ ]:
```
sns.boxplot(df_yulu, y='registered')
```

[ ]: <Axes: ylabel='registered'>

Plot the Graph

```
[ ]: plt.figure(figsize=(15, 8)).suptitle(
         "Yulu Bike Rent by Registered User Dashboard", fontsize=14)

     plt.subplot(2, 2, 1)
     plt.pie(df_yulu_grouped['registered_group'].value_counts().values,␣
      ↪labels=df_yulu_grouped['registered_group'].value_counts(
     ).index, radius=1.5, autopct='%1.2f%%')  # type: ignore

     plt.subplot(2, 2, 3)
     sns.boxplot(df_yulu, y="registered")
     plt.xlabel('No. of Bike Rent', fontsize=12)
     plt.ylabel('')

     plt.subplot(2, 2, 2)
     sns.countplot(df_yulu_grouped, x='registered_group')
     plt.ylabel('Count', fontsize=12)
     plt.xlabel('', fontsize=11)
     plt.xticks(rotation=30, fontsize=11)
     plt.yticks(fontsize=11)

     plt.subplot(2, 2, 4)
     sns.histplot(data=df_yulu, x='registered', bins=10, kde=True, multiple="stack")
     plt.xlabel('No. of Bike Rents', fontsize=13)
     plt.ylabel("Count", fontsize=12)

     plt.show()
```



1.6.12 Total Rental Bike Count

[ ]:
```
df_yulu['count'].unique().shape
```

[ ]:
```
(643,)
```

[ ]:
```
df_yulu['count'].value_counts().head(5)
```

[ ]:
```
5    166
4    147
3    140
6    134
2    129
Name: count, dtype: int64
```

[ ]:
```
df_yulu['count'].value_counts(normalize=True).head(5)*100
```

[ ]:
```
5    1.603555
4    1.420015
3    1.352396
6    1.294436
2    1.246136
Name: count, dtype: float64
```

Statitical Analysis

[ ]:
```
df_yulu['count'].describe()
```

[ ]:
```
count    10352.000000
mean       176.366789
std        156.952045
min          1.000000
25%         40.000000
50%        139.000000
75%        271.000000
max        649.000000
Name: count, dtype: float64
```

[ ]:
```
df_yulu['count'].mean()
```

[ ]:
```
176.36678902627511
```

[ ]:
```
df_yulu['count'].median()
```

[ ]:
```
139.0
```

[ ]:
```
df_yulu_grouped['count_group'].mode()[0]
```

[ ]:
```
'0-100'
```

Check for Outliers

[ ]:
```
check_outlier(df_yulu, 'count')['upper']
```

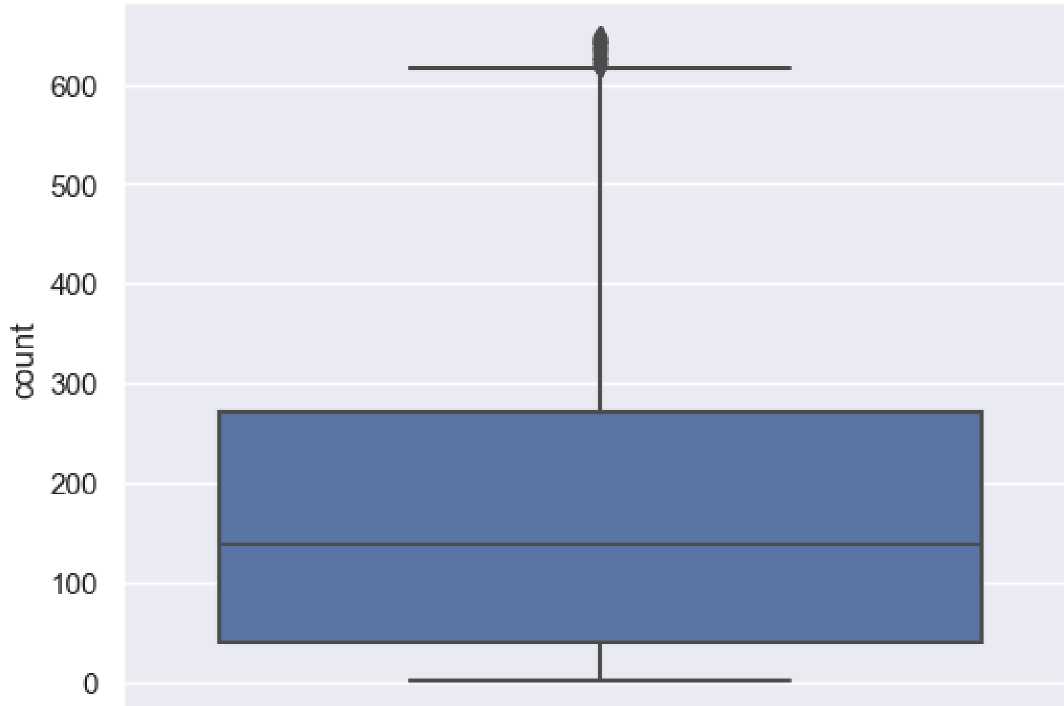
[ ]:
```
{'list': 2587     638
 3619     628
 4456     620
 4480     625
 6610     644
         …
 10204    627
 10318    646
 10423    619
 10750    636
 10759    622
 Name: count, Length: 77, dtype: int64,
 'length': 77}
```

[ ]:
```
check_outlier(df_yulu, 'count')['lower']
```

[ ]:
```
{'list': Series([], Name: count, dtype: int64), 'length': 0}
```

[ ]:
```
sns.boxplot(df_yulu, y='count')
```

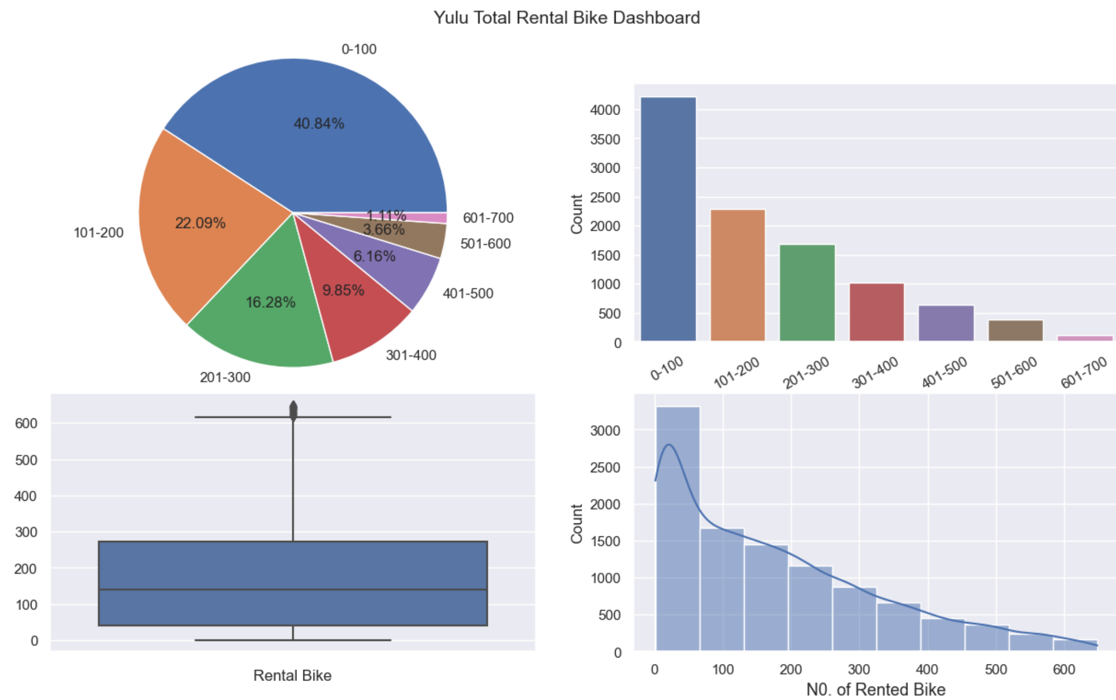
[ ]:
```
<Axes: ylabel='count'>
```



Plot the Graph

[ ]:
```
plt.figure(figsize=(15, 8)).suptitle(
    "Yulu Total Rental Bike Dashboard", fontsize=14)

plt.subplot(2, 2, 1)
plt.pie(df_yulu_grouped['count_group'].value_counts().values,␣
 ↪labels=df_yulu_grouped['count_group'].value_counts(
).index, radius=1.5, autopct='%1.2f%%')  # type: ignore

plt.subplot(2, 2, 3)
sns.boxplot(df_yulu, y="count")
plt.xlabel('Rental Bike', fontsize=12)
plt.ylabel('')

plt.subplot(2, 2, 2)
sns.countplot(df_yulu_grouped, x='count_group')
plt.ylabel('Count', fontsize=12)
plt.xlabel('', fontsize=11)
plt.xticks(rotation=30, fontsize=11)
plt.yticks(fontsize=11)

plt.subplot(2, 2, 4)
sns.histplot(data=df_yulu, x='count', bins=10, kde=True, multiple="stack")
```

```
plt.xlabel('NO. of Rented Bike', fontsize=13)
plt.ylabel("Count", fontsize=12)

plt.show()
```



1.7 Relation Between Indipendent Variable

[ ]:
```
df_yulu.columns
```

[ ]:
```
Index(['datetime', 'time_slot', 'season', 'holiday', 'workingday', 'weather',
       'temp', 'atemp', 'humidity', 'windspeed', 'casual', 'registered',
       'count'],
      dtype='object')
```

[ ]:
```
dependent_variables = ['count']
independent_variables = ['workingday, weather, season']
```

1.7.1 Relation Between Wether & Season

[ ]:
```
Ho = "There is no relation between Weather & Season"
Ha = "Weather & Season are co-related"
```

[ ]:
```
stats, p_value, dof, data = chi2_contingency(
    pd.crosstab(df_yulu['weather'], df_yulu['season'],))
```

```
stats, p_value
```

[ ]: (50.437100227285605, 8.913970213670006e-08)

```
[ ]: print(Ha if (p_value < significance_level) else Ho)
```

Weather & Season are co-related

1.7.2 Relation Between Wether & Working Day

```
[ ]: Ho = "There is no relation between Weather & Working Day"
     Ha = "Weather & Working Day are co-related"
```

```
[ ]: stats, p_value, dof, data = chi2_contingency(
         pd.crosstab(df_yulu['weather'], df_yulu['workingday'],))
     stats, p_value
```

[ ]: (14.81140809965818, 0.001985116190288393)

```
[ ]: print(Ha if (p_value < significance_level) else Ho)
```

Weather & Working Day are co-related

1.7.3 Relation Between Working Day & Season

```
[ ]: Ho = "There is no relation between Working Day & Season"
     Ha = "Working Day & Season are co-related"
```

```
[ ]: stats, p_value, dof, data = chi2_contingency(
         pd.crosstab(df_yulu['workingday'], df_yulu['season'],))
     stats, p_value
```

[ ]: (2.1088184699125585, 0.5501307880329931)

```
[ ]: print(Ha if (p_value < significance_level) else Ho)
```

There is no relation between Working Day & Season

1.8 Bootstraping:

```
[ ]: def filter_col_as_array(df, col, match, result_col, inverse=False):
         if inverse:
             return df[df[col] != match][result_col]
         else:
             return df[df[col] == match][result_col]
```

```
[ ]: class CLT_Interval:
```

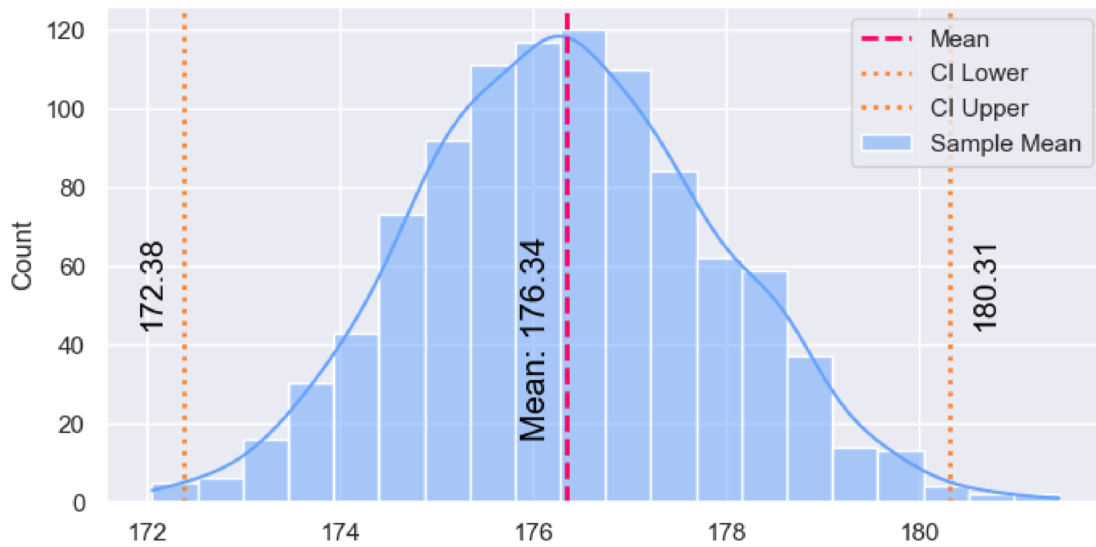
```
    def __init__(self, mean, std_error, z_critical, margin_of_error,␣
↪confidence_interval_lower, confidence_interval_upper, sample_df,␣
↪significance_level, samples_size):
        self.mean = mean
        self.std_error = std_error
        self.z_critical = z_critical
        self.margin_of_error = margin_of_error
        self.confidence_interval_lower = confidence_interval_lower
        self.confidence_interval_upper = confidence_interval_upper
        self.sample_df = sample_df
        self.significance_level = significance_level
        self.samples_size = samples_size
```

[ ]:
```
def bootstraping(df, samples, cl, plot=True):
    bootstrap_sample_means = []

    for _ in range(samples):
        bootstrap_sample = df.sample(n=len(df), replace=True)
        bootstrap_sample_mean = bootstrap_sample.mean()
        bootstrap_sample_means.append(bootstrap_sample_mean)

    bootstrap_mean = np.mean(bootstrap_sample_means)
    bootstrap_std_error = np.std(bootstrap_sample_means)

    z_critical = norm.ppf((1 + cl) / 2)

    # Calculate the margin of error
    margin_of_error = z_critical * bootstrap_std_error

    # Calculate the confidence interval
    confidence_interval_lower = bootstrap_mean - margin_of_error
    confidence_interval_upper = bootstrap_mean + margin_of_error

    if (plot):
        plt.figure(figsize=(8, 4))
        sns.histplot(data=bootstrap_sample_means, bins=20,
                     color='#66a3ff', label='Sample Mean', kde=True)
        plt.axvline(bootstrap_mean, color='#ff0066',
                    linestyle='dashed', linewidth=2, label='Mean')
        plt.axvline(confidence_interval_lower, color='#ff8533',
                    linestyle='dotted', linewidth=2, label='CI Lower')
        plt.axvline(confidence_interval_upper, color='#ff8533',
                    linestyle='dotted', linewidth=2, label='CI Upper')
        plt.annotate(f'Mean: {bootstrap_mean:.2f}', rotation='vertical', xy=(
            bootstrap_mean, 0), xytext=(-20, 30), textcoords='offset points',␣
↪color='blacK', fontsize=15)
```

```
        plt.annotate(f'{confidence_interval_lower:.2f}', rotation='vertical',␣
↪xy=(
            confidence_interval_lower, 0), xytext=(-20, 80), textcoords='offset␣
↪points', color='blacK', fontsize=14)
        plt.annotate(f'{confidence_interval_upper:.2f}', rotation='vertical',␣
↪xy=(
            confidence_interval_upper, 0), xytext=(10, 80), textcoords='offset␣
↪points', color='blacK', fontsize=14)
        plt.legend()

    return CLT_Interval(bootstrap_mean, bootstrap_std_error, z_critical,␣
↪margin_of_error, confidence_interval_lower, confidence_interval_upper,␣
↪bootstrap_sample_means, cl, samples)
```

[ ]:
```
df = df_yulu['count']
samples = 1000
confidence_level = 0.99
clt_interval = bootstraping(df, samples, cl=confidence_level, plot=True)

print("Bootstrap Mean:", clt_interval.mean)
print("Bootstrap Standard Error:", clt_interval.std_error)
print("Z-Critical Value:", clt_interval.z_critical)
print("Margin of Error:", clt_interval.margin_of_error)
print("Bootstrap Confidence Interval:",
      (clt_interval.confidence_interval_lower, clt_interval.
↪confidence_interval_upper))
```

```
Bootstrap Mean: 176.34169300618237
Bootstrap Standard Error: 1.5393571100012733
Z-Critical Value: 2.5758293035489004
Margin of Error: 3.9651211525676278
Bootstrap Confidence Interval: (172.37657185361473, 180.30681415875)
```



1.9 Questions

- Working Day has effect on number of electric cycles rented
- No. of cycles rented similar or different in different seasons
- No. of cycles rented similar or different in different weather
- Weather is dependent on season (check between 2 predictor variable)

1.9.1 Validate Target variable's Data is Gaussian or not

[ ]:
```
stats, p_value = shapiro(df_yulu['count'])
round(p_value, 4)
```

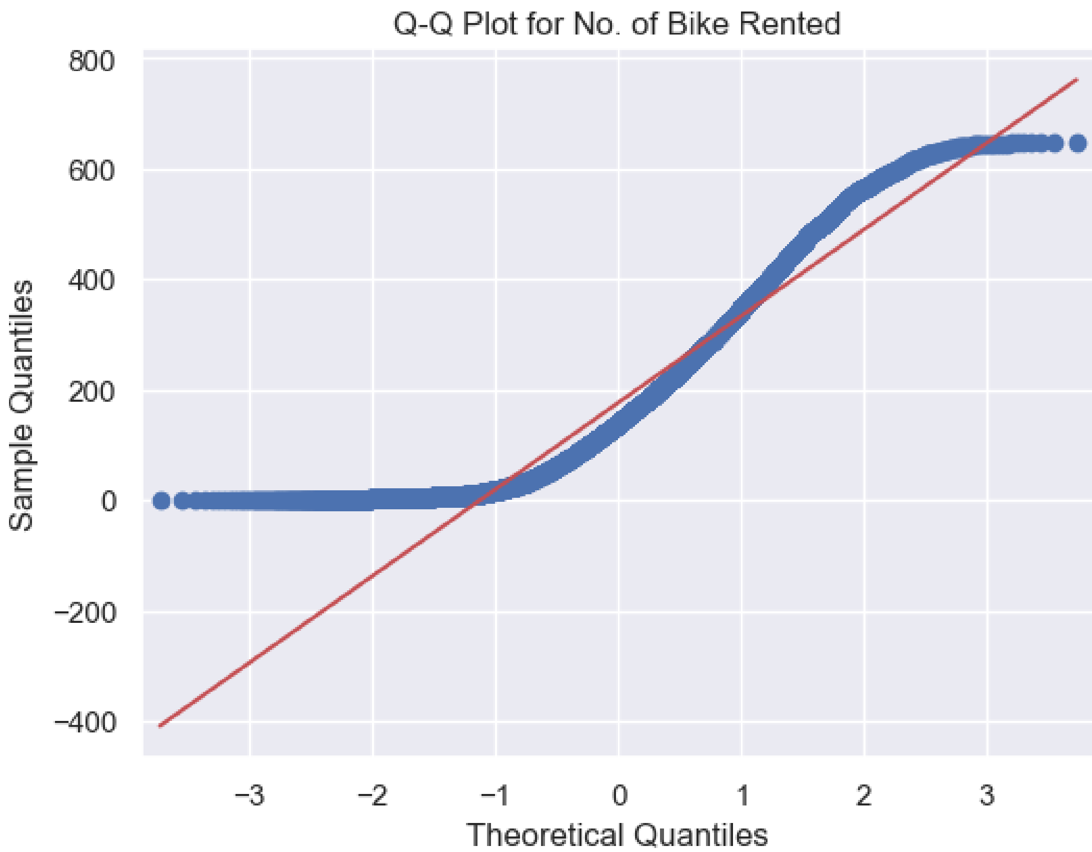
[ ]: 0.0

[ ]:
```
"Data is not Gaussian" if (
    p_value < significance_level) else "Data is Gaussian"
```

[ ]: 'Data is not Gaussian'

[ ]:
```
plt.figure(figsize=(15, 5)).suptitle(
    "Properties of No. of Bike Rented")

plt.subplot(121)
sns.histplot(df_yulu['count'], kde=True)
plt.ylabel("Count")
plt.xlabel("No. Bike Rented")

plt.subplot(122)
sns.histplot(np.log(df_yulu['count']), kde=True)
```

```
plt.ylabel("Count")
plt.xlabel("Log of No. Bike Rented")

qqplot(df_yulu['count'], line='s')
plt.title("Q-Q Plot for No. of Bike Rented")

plt.show()
```



1.9.2 Q1. Working Day has effect on number of electric cycles rented

[ ]:
```
rented_bike_on_Working_day = df_yulu.groupby(
    'workingday').aggregate(
        mean=('count', 'mean'),
        count=('count', 'count')).reset_index()
rented_bike_on_Working_day
```

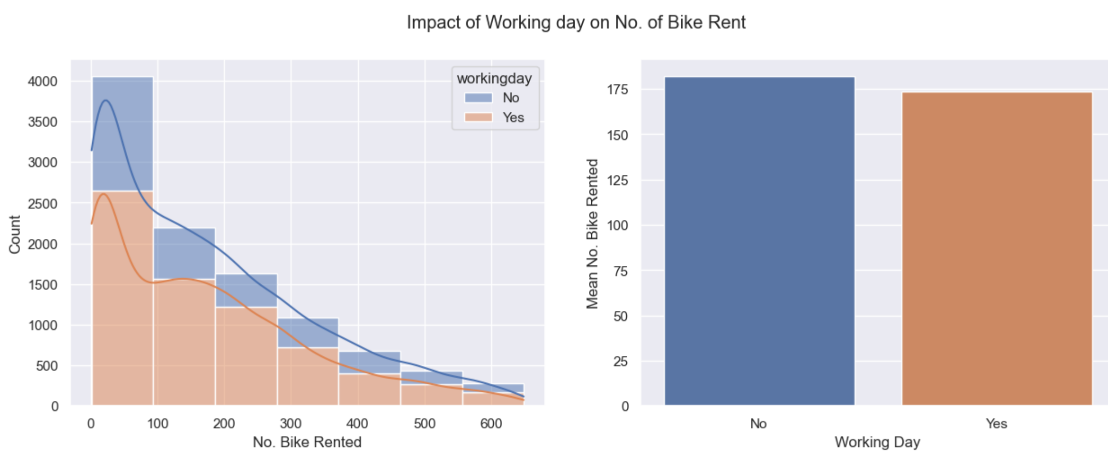
[ ]:

	workingday	mean	count
0	No	182.189881	3360
1	Yes	173.568507	6992

[ ]:
```
rent_on_wd = filter_col_as_array(
    df=df_yulu, col='workingday', match='Yes', result_col='count')
rent_on_non_wd = filter_col_as_array(
    df=df_yulu, col='workingday', match='No', result_col='count')
```

Plot the Graph to Support Assumptions

[ ]:
```
plt.figure(figsize=(15, 5)).suptitle(
    "Impact of Working day on No. of Bike Rent")

plt.subplot(121)
sns.histplot(data=df_yulu, x='count', bins=7,
             hue='workingday', kde=True, multiple="stack")
plt.ylabel("Count")
plt.xlabel("No. Bike Rented")

plt.subplot(122)
sns.barplot(x=rented_bike_on_Working_day['workingday'],
            y=rented_bike_on_Working_day['mean'])
plt.ylabel("Mean No. Bike Rented")
plt.xlabel("Working Day")
plt.show()
plt.show()
```



Hypothesis Testing Here, we found that mean of Bike rented in Working day is less than non-working day.

[ ]:
```
Ho = "Working Day has no effect on number of electric cycles rented"
Ha = "Working Day has effect on number of electric cycles rented"
```

Using T-Test

[ ]:
```
t_statistic, p_value = ttest_ind(
    rent_on_wd, rent_on_non_wd)
round((t_statistic*100), 4), round((p_value*100), 4)
```

[ ]: (-261.7523, 0.887)

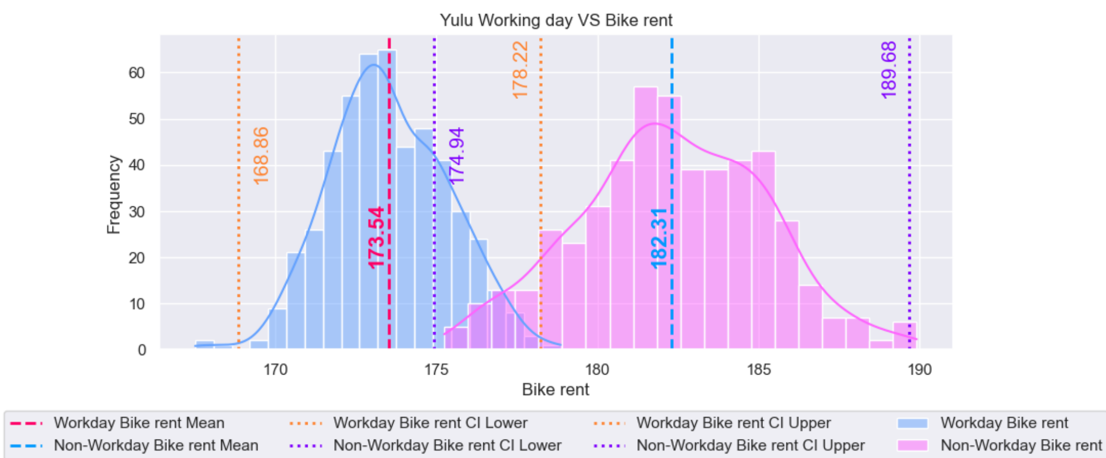
[ ]:
```
print(Ha if (p_value < significance_level) else Ho)
```

Working Day has effect on number of electric cycles rented

Use the Central limit theorem to compute the interval

```
[ ]: samples = 500
wd_clt_interval = bootstraping(
    rent_on_wd, samples, cl=confidence_interval, plot=False)
non_wd_clt_interval = bootstraping(
    rent_on_non_wd, samples, cl=confidence_interval, plot=False)

plt.figure(figsize=(10, 4))

sns.histplot(data=wd_clt_interval.sample_df, bins=20,
             color='#66a3ff', label='Workday Bike rent', kde=True)
sns.histplot(data=non_wd_clt_interval.sample_df, bins=20,
             color='#ff66ff', label='Non-Workday Bike rent', kde=True)

plt.axvline(wd_clt_interval.mean, color='#ff0066',
            linestyle='dashed', linewidth=2, label='Workday Bike rent Mean')
plt.axvline(non_wd_clt_interval.mean, color='#0099ff',
            linestyle='dashed', linewidth=2, label='Non-Workday Bike rent Mean')

plt.annotate(f'{wd_clt_interval.mean:.2f}', rotation='vertical', xy=(
    wd_clt_interval.mean, 0), xytext=(-15, 60), textcoords='offset points',␣
↪color='#ff0066', fontsize=15, weight='bold')
plt.annotate(f'{non_wd_clt_interval.mean:.2f}', rotation='vertical', xy=(
    non_wd_clt_interval.mean, 0), xytext=(-15, 60), textcoords='offset points',␣
↪color='#0099ff', fontsize=15, weight='bold')

plt.axvline(wd_clt_interval.confidence_interval_lower, color='#ff8533',
            linestyle='dotted', linewidth=2, label='Workday Bike rent CI Lower')
plt.axvline(non_wd_clt_interval.confidence_interval_lower, color='#8000ff',
            linestyle='dotted', linewidth=2, label='Non-Workday Bike rent CI␣
↪Lower')

plt.annotate(f'{wd_clt_interval.confidence_interval_lower:.2f}',␣
↪rotation='vertical', xy=(
    wd_clt_interval.confidence_interval_lower, 0), xytext=(10, 120),␣
↪textcoords='offset points', color='#ff8533', fontsize=14)
plt.annotate(f'{non_wd_clt_interval.confidence_interval_lower:.2f}',␣
↪rotation='vertical', xy=(
    non_wd_clt_interval.confidence_interval_lower, 0), xytext=(10, 120),␣
↪textcoords='offset points', color='#8000ff', fontsize=14)

plt.axvline(wd_clt_interval.confidence_interval_upper, color='#ff8533',
            linestyle='dotted', linewidth=2, label='Workday Bike rent CI Upper')
plt.axvline(non_wd_clt_interval.confidence_interval_upper, color='#8000ff',
```

```
            linestyle='dotted', linewidth=2, label='Non-Workday Bike rent CI␣
 ↪Upper')

plt.annotate(f'{wd_clt_interval.confidence_interval_upper:.2f}',␣
 ↪rotation='vertical', xy=(
    wd_clt_interval.confidence_interval_upper, 0), xytext=(-20, 180),␣
 ↪textcoords='offset points', color='#ff8533', fontsize=14)
plt.annotate(f'{non_wd_clt_interval.confidence_interval_upper:.2f}',␣
 ↪rotation='vertical', xy=(
    non_wd_clt_interval.confidence_interval_upper, 0), xytext=(-20, 180),␣
 ↪textcoords='offset points', color='#8000ff', fontsize=14)

plt.xlabel('Bike rent')
plt.ylabel('Frequency')
plt.title('Yulu Working day VS Bike rent')
plt.legend(bbox_to_anchor=(0.5, -0.35),
           loc='lower center', borderaxespad=0, ncol=4)


plt.show()
```



Insight

Working Day has effect on number of electric cycles rented.

```
Ho -> Working Day has no effect on number of electric cycles rented
Ha -> Working Day has effect on number of electric cycles rented
```

```
Using T-Test p_value found that 0.887%.
```

- With Confidence interval of 99% & Sample Size of 500
  - Mean Bike rent on Working day is 173.48 with a Intervals of (168.92 - 178.04).
  - Mean Bike rent on Non-Working day is 182.21 with a Intervals of (175.20 - 189.22)
  - As 0.887% < 1% Thus Rejecting Ho & Accept Ha.

As per T-Test we can Conclude "Working Day has effect on number of electric cycles rented".

1.9.3 Q2. No. of cycles rented similar or different in different seasons

[ ]:
```
rented_bike_on_seasons = df_yulu.groupby(
    'season').aggregate(
        mean=('count', 'mean'),
        count=('count', 'count')).reset_index()
rented_bike_on_seasons
```

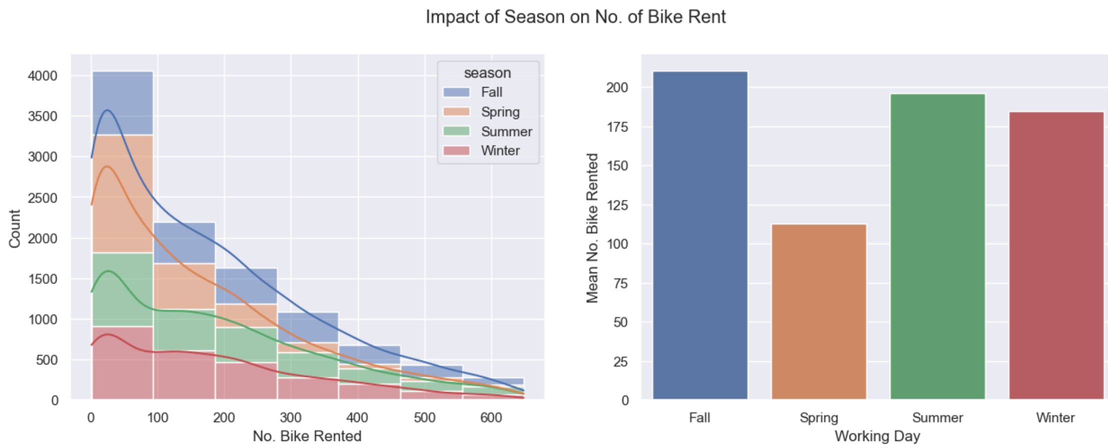
[ ]:

	season	mean	count
0	Fall	210.633564	2598
1	Spring	112.774308	2530
2	Summer	195.945328	2579
3	Winter	184.446503	2645

[ ]:
```
rent_on_summer = filter_col_as_array(
    df=df_yulu, col='season', match='Summer', result_col='count')
rent_on_fall = filter_col_as_array(
    df=df_yulu, col='season', match='Fall', result_col='count')
rent_on_winter = filter_col_as_array(
    df=df_yulu, col='season', match='Winter', result_col='count')
rent_on_spring = filter_col_as_array(
    df=df_yulu, col='season', match='Spring', result_col='count')
```

Plot the Graph to Support Assumptions

[ ]:
```
plt.figure(figsize=(15, 5)).suptitle(
    "Impact of Season on No. of Bike Rent")

plt.subplot(121)
sns.histplot(data=df_yulu, x='count', bins=7,
             hue='season', kde=True, multiple="stack")
plt.ylabel("Count")
plt.xlabel("No. Bike Rented")

plt.subplot(122)
sns.barplot(x=rented_bike_on_seasons['season'],
            y=rented_bike_on_seasons['mean'])
plt.ylabel("Mean No. Bike Rented")
plt.xlabel("Working Day")
plt.show()
plt.show()
```



Hypothesis Testing Here, we found that No. of Bike rented are different in different seasons

[ ]:
```
Ho = "No. of Bike rented are Similar in different seasons"
Ha = "No. of Bike rented are different in different seasons"
```

Using ANOVA

[ ]:
```
t_statistic, p_value = f_oneway(
    rent_on_summer, rent_on_fall, rent_on_winter, rent_on_spring)
round((t_statistic*100), 4), round((p_value*100), 4)
```

[ ]: (20711.2224, 0.0)

[ ]:
```
print(Ha if (p_value < significance_level) else Ho)
```

No. of Bike rented are different in different seasons

Using Kruskal

[ ]:
```
t_statistic, p_value = kruskal(
    rent_on_summer, rent_on_fall, rent_on_winter, rent_on_spring)
round((t_statistic*100), 4), round((p_value*100), 4)
```

[ ]: (59158.8977, 0.0)

[ ]:
```
print(Ha if (p_value < significance_level) else Ho)
```

No. of Bike rented are different in different seasons

Using Levene

[ ]:
```
t_statistic, p_value = levene(
    rent_on_summer, rent_on_fall, rent_on_winter, rent_on_spring)
round((t_statistic*100), 4), round((p_value*100), 4)
```

[ ]: (17240.3736, 0.0)

[ ]:
```
print(Ha if (p_value < significance_level) else Ho)
```

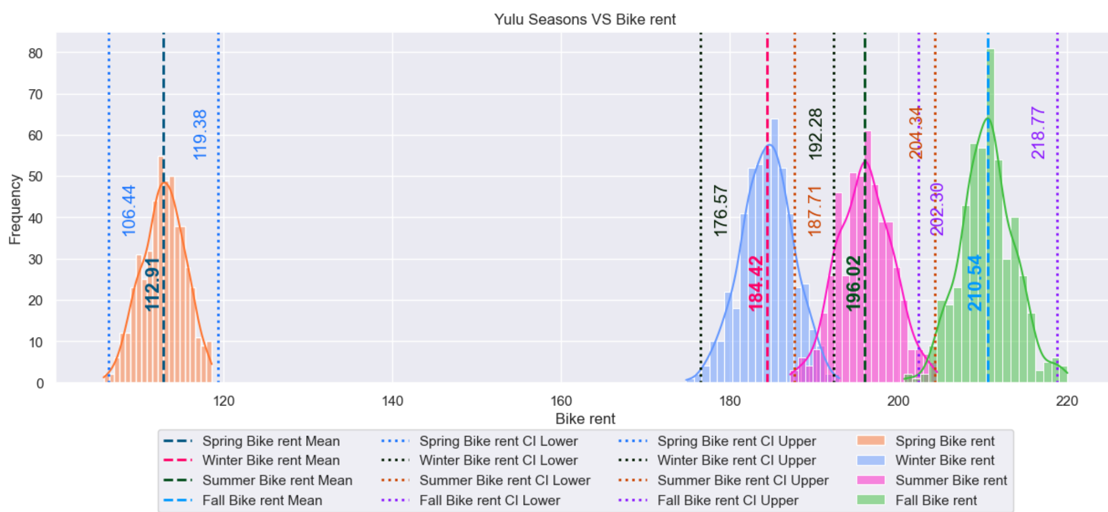
```
No. of Bike rented are different in different seasons
```

Use the Central limit theorem to compute the interval

[ ]:
```
samples = 500
spring_clt_interval = bootstraping(
    rent_on_spring, samples, cl=confidence_interval, plot=False)
winter_clt_interval = bootstraping(
    rent_on_winter, samples, cl=confidence_interval, plot=False)
summer_clt_interval = bootstraping(
    rent_on_summer, samples, cl=confidence_interval, plot=False)
fall_clt_interval = bootstraping(
    rent_on_fall, samples, cl=confidence_interval, plot=False)

plt.figure(figsize=(15, 5))

sns.histplot(data=spring_clt_interval.sample_df, bins=20,
             color='#ff7733', label='Spring Bike rent', kde=True)
sns.histplot(data=winter_clt_interval.sample_df, bins=20,
             color='#6699ff', label='Winter Bike rent', kde=True)
sns.histplot(data=summer_clt_interval.sample_df, bins=20,
             color='#ff1ac6', label='Summer Bike rent', kde=True)
sns.histplot(data=fall_clt_interval.sample_df, bins=20,
             color='#40bf40', label='Fall Bike rent', kde=True)

plt.axvline(spring_clt_interval.mean, color='#005580',
            linestyle='dashed', linewidth=2, label='Spring Bike rent Mean')
plt.axvline(winter_clt_interval.mean, color='#ff0066',
            linestyle='dashed', linewidth=2, label='Winter Bike rent Mean')
plt.axvline(summer_clt_interval.mean, color='#004d1a',
            linestyle='dashed', linewidth=2, label='Summer Bike rent Mean')
plt.axvline(fall_clt_interval.mean, color='#0099ff',
            linestyle='dashed', linewidth=2, label='Fall Bike rent Mean')

plt.annotate(f'{spring_clt_interval.mean:.2f}', rotation='vertical', xy=(
    spring_clt_interval.mean, 0), xytext=(-15, 60), textcoords='offset points',␣
 ↪color='#005580', fontsize=15, weight='bold')
plt.annotate(f'{winter_clt_interval.mean:.2f}', rotation='vertical', xy=(
    winter_clt_interval.mean, 0), xytext=(-15, 60), textcoords='offset points',␣
 ↪color='#ff0066', fontsize=15, weight='bold')
plt.annotate(f'{summer_clt_interval.mean:.2f}', rotation='vertical', xy=(
    summer_clt_interval.mean, 0), xytext=(-15, 60), textcoords='offset points',␣
 ↪color='#004d1a', fontsize=15, weight='bold')
plt.annotate(f'{fall_clt_interval.mean:.2f}', rotation='vertical', xy=(
```

```
    fall_clt_interval.mean, 0), xytext=(-15, 60), textcoords='offset points',␣
 ↪color='#0099ff', fontsize=15, weight='bold')

plt.axvline(spring_clt_interval.confidence_interval_lower, color='#1a75ff',
            linestyle='dotted', linewidth=2, label='Spring Bike rent CI Lower')
plt.axvline(winter_clt_interval.confidence_interval_lower, color='#001a00',
            linestyle='dotted', linewidth=2, label='Winter Bike rent CI Lower')
plt.axvline(summer_clt_interval.confidence_interval_lower, color='#cc4400',
            linestyle='dotted', linewidth=2, label='Summer Bike rent CI Lower')
plt.axvline(fall_clt_interval.confidence_interval_lower, color='#8c1aff',
            linestyle='dotted', linewidth=2, label='Fall Bike rent CI Lower')

plt.annotate(f'{spring_clt_interval.confidence_interval_lower:.2f}',␣
 ↪rotation='vertical', xy=(
    spring_clt_interval.confidence_interval_lower, 0), xytext=(10, 120),␣
 ↪textcoords='offset points', color='#1a75ff', fontsize=14)
plt.annotate(f'{winter_clt_interval.confidence_interval_lower:.2f}',␣
 ↪rotation='vertical', xy=(
    winter_clt_interval.confidence_interval_lower, 0), xytext=(10, 120),␣
 ↪textcoords='offset points', color='#001a00', fontsize=14)
plt.annotate(f'{summer_clt_interval.confidence_interval_lower:.2f}',␣
 ↪rotation='vertical', xy=(
    summer_clt_interval.confidence_interval_lower, 0), xytext=(10, 120),␣
 ↪textcoords='offset points', color='#cc4400', fontsize=14)
plt.annotate(f'{fall_clt_interval.confidence_interval_lower:.2f}',␣
 ↪rotation='vertical', xy=(
    fall_clt_interval.confidence_interval_lower, 0), xytext=(10, 120),␣
 ↪textcoords='offset points', color='#8c1aff', fontsize=14)

plt.axvline(spring_clt_interval.confidence_interval_upper, color='#1a75ff',
            linestyle='dotted', linewidth=2, label='Spring Bike rent CI Upper')
plt.axvline(winter_clt_interval.confidence_interval_upper, color='#001a00',
            linestyle='dotted', linewidth=2, label='Winter Bike rent CI Upper')
plt.axvline(summer_clt_interval.confidence_interval_upper, color='#cc4400',
            linestyle='dotted', linewidth=2, label='Summer Bike rent CI Upper')
plt.axvline(fall_clt_interval.confidence_interval_upper, color='#8c1aff',
            linestyle='dotted', linewidth=2, label='Fall Bike rent CI Upper')

plt.annotate(f'{spring_clt_interval.confidence_interval_upper:.2f}',␣
 ↪rotation='vertical', xy=(
    spring_clt_interval.confidence_interval_upper, 0), xytext=(-20, 180),␣
 ↪textcoords='offset points', color='#1a75ff', fontsize=14)
plt.annotate(f'{winter_clt_interval.confidence_interval_upper:.2f}',␣
 ↪rotation='vertical', xy=(
    winter_clt_interval.confidence_interval_upper, 0), xytext=(-20, 180),␣
 ↪textcoords='offset points', color='#001a00', fontsize=14)
```

```
plt.annotate(f'{summer_clt_interval.confidence_interval_upper:.2f}',␣
    ↪rotation='vertical', xy=(
        summer_clt_interval.confidence_interval_upper, 0), xytext=(-20, 180),␣
    ↪textcoords='offset points', color='#cc4400', fontsize=14)
plt.annotate(f'{fall_clt_interval.confidence_interval_upper:.2f}',␣
    ↪rotation='vertical', xy=(
        fall_clt_interval.confidence_interval_upper, 0), xytext=(-20, 180),␣
    ↪textcoords='offset points', color='#8c1aff', fontsize=14)

plt.xlabel('Bike rent')
plt.ylabel('Frequency')
plt.title('Yulu Seasons VS Bike rent')
plt.legend(bbox_to_anchor=(0.5, -0.37),
           loc='lower center', borderaxespad=0, ncol=4)


plt.show()
```



Insight

No. of cycles rented similar or different in different seasons

```
Ho -> No. of Bike rented are Similar in different seasons
Ha -> No. of Bike rented are different in different seasons
```

`Using ANOVA, Kruskal & Levene Test Test p_value found that 0.0%.`

- With Confidence interval of 99% & Sample Size of 500
  - Mean Bike rent on Spring Season is 112.87 with a Intervals of (106.51 - 119.23).

- Mean Bike rent on Winter Season is 184.68 with a Intervals of (176.95 - 192.40).
- Mean Bike rent on Summer Season is 195.84 with a Intervals of (186.98 - 204.69).
- Mean Bike rent on Fall Season is 210.84 with a Intervals of (202.32 - 219.37).
- As 0% < 1% Thus Rejecting Ho & Accept Ha.

As per Anova Test we can Conclude "No. of Bike rented are different in different seasons".

1.9.4 Q3. No. of cycles rented similar or different in different weather

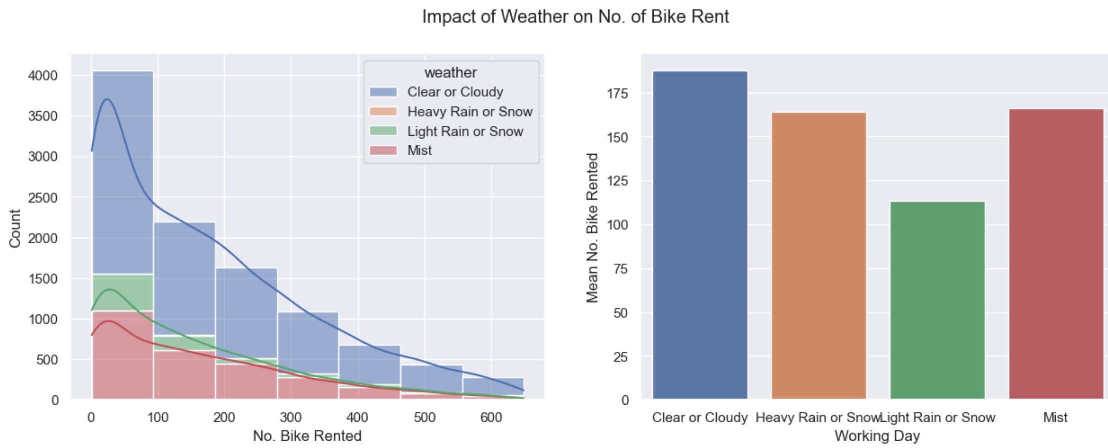
[ ]:
```
rented_bike_on_weather = df_yulu.groupby(
    'weather').aggregate(
        mean=('count', 'mean'),
        count=('count', 'count')).reset_index()
rented_bike_on_weather
```

[ ]:
```
              weather        mean  count
0     Clear or Cloudy  187.822607   6821
1  Heavy Rain or Snow  164.000000      1
2  Light Rain or Snow  113.562108    797
3                Mist  166.095134   2733
```

[ ]:
```
rent_on_clear_cloudy = filter_col_as_array(
    df=df_yulu, col='weather', match='Clear or Cloudy', result_col='count')
rent_on_heavy_rain_snow = filter_col_as_array(
    df=df_yulu, col='weather', match='Heavy Rain or Snow', result_col='count')
rent_on_light_rain_snow = filter_col_as_array(
    df=df_yulu, col='weather', match='Light Rain or Snow', result_col='count')
rent_on_mist = filter_col_as_array(
    df=df_yulu, col='weather', match='Mist', result_col='count')
```

Plot the Graph to Support Assumptions

[ ]:
```
plt.figure(figsize=(15, 5)).suptitle(
    "Impact of Weather on No. of Bike Rent")

plt.subplot(121)
sns.histplot(data=df_yulu, x='count', bins=7,
             hue='weather', kde=True, multiple="stack")
plt.ylabel("Count")
plt.xlabel("No. Bike Rented")

plt.subplot(122)
sns.barplot(x=rented_bike_on_weather['weather'],
            y=rented_bike_on_weather['mean'])
plt.ylabel("Mean No. Bike Rented")
plt.xlabel("Working Day")
plt.show()
plt.show()
```



Hypothesis Testing Here, we found that No. of Bike rented are different in different weather

[ ]:
```
Ho = "No. of Bike rented are Similar in different weather"
Ha = "No. of Bike rented are different in different weather"
```

Using `ANOVA`

[ ]:
```
t_statistic, p_value = f_oneway(
    rent_on_clear_cloudy, rent_on_heavy_rain_snow, rent_on_light_rain_snow,␣
 ↪rent_on_mist)
round((t_statistic*100), 4), round((p_value*100), 4)
```

[ ]: (5954.9314, 0.0)

[ ]:
```
print(Ha if (p_value < significance_level) else Ho)
```

```
No. of Bike rented are different in different weather
```

Using `Kruskal`

[ ]:
```
t_statistic, p_value = kruskal(
    rent_on_clear_cloudy, rent_on_heavy_rain_snow, rent_on_light_rain_snow,␣
 ↪rent_on_mist)
round((t_statistic*100), 4), round((p_value*100), 4)
```

[ ]: (16758.251, 0.0)

[ ]:
```
print(Ha if (p_value < significance_level) else Ho)
```

```
No. of Bike rented are different in different weather
```

Using `Levene`

[ ]:
```
t_statistic, p_value = levene(
    rent_on_clear_cloudy, rent_on_heavy_rain_snow, rent_on_light_rain_snow,␣
 ↪rent_on_mist)
round((t_statistic*100), 4), round((p_value*100), 4)
```

[ ]:
```
(5740.2193, 0.0)
```

[ ]:
```
print(Ha if (p_value < significance_level) else Ho)
```

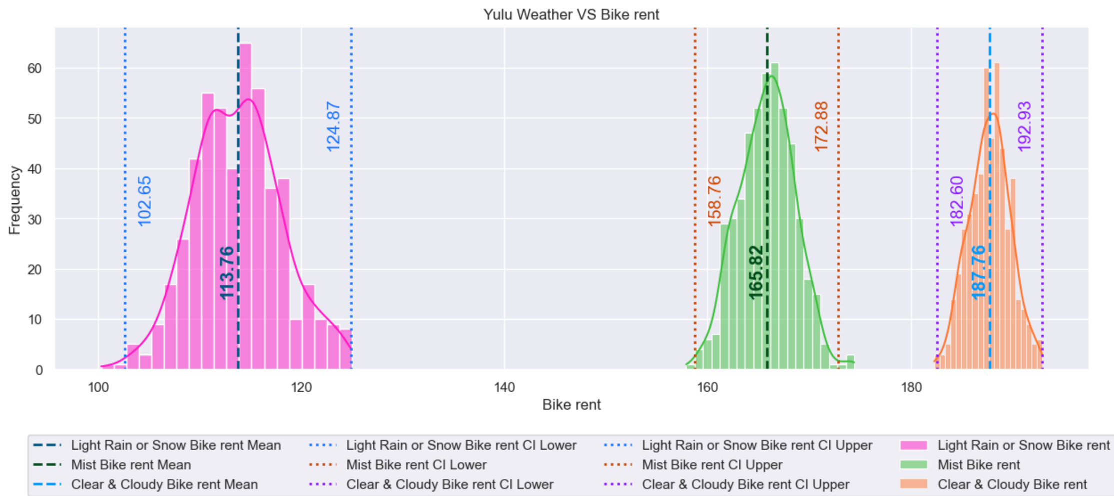
```
No. of Bike rented are different in different weather
```

Use the `Central limit` theorem to compute the interval

[ ]:
```
samples = 500
light_rain_snow_clt_interval = bootstraping(
    rent_on_light_rain_snow, samples, cl=confidence_interval, plot=False)
# heavy_rain_snow_clt_interval = bootstraping(
#     rent_on_heavy_rain_snow, samples, cl=confidence_interval, plot=False)
mist_clt_interval = bootstraping(
    rent_on_mist, samples, cl=confidence_interval, plot=False)
clear_cloudy_clt_interval = bootstraping(
    rent_on_clear_cloudy, samples, cl=confidence_interval, plot=False)

plt.figure(figsize=(15, 5))

sns.histplot(data=light_rain_snow_clt_interval.sample_df, bins=20,
             color='#ff1ac6', label='Light Rain or Snow Bike rent', kde=True)
# sns.histplot(data=heavy_rain_snow_clt_interval.sample_df, bins=20,
#               color='#6699ff', label='Heavy Rain or Snow Bike rent', kde=True)
sns.histplot(data=mist_clt_interval.sample_df, bins=20,
             color='#40bf40', label='Mist Bike rent', kde=True)
sns.histplot(data=clear_cloudy_clt_interval.sample_df, bins=20,
             color='#ff7733', label='Clear & Cloudy Bike rent', kde=True)

plt.axvline(light_rain_snow_clt_interval.mean, color='#005580',
            linestyle='dashed', linewidth=2, label='Light Rain or Snow Bike␣
 ↪rent Mean')
# plt.axvline(heavy_rain_snow_clt_interval.mean, color='#ff0066',
#              linestyle='dashed', linewidth=2, label='Heavy Rain or Snow Bike␣
 ↪rent Mean')
plt.axvline(mist_clt_interval.mean, color='#004d1a',
            linestyle='dashed', linewidth=2, label='Mist Bike rent Mean')
plt.axvline(clear_cloudy_clt_interval.mean, color='#0099ff',
            linestyle='dashed', linewidth=2, label='Clear & Cloudy Bike rent␣
 ↪Mean')

plt.annotate(f'{light_rain_snow_clt_interval.mean:.2f}', rotation='vertical',␣
 ↪xy=(
```

```
    light_rain_snow_clt_interval.mean, 0), xytext=(-15, 60), textcoords='offset␣
↪points', color='#005580', fontsize=15, weight='bold')
# plt.annotate(f'{heavy_rain_snow_clt_interval.mean:.2f}', rotation='vertical',␣
↪xy=(
#     heavy_rain_snow_clt_interval.mean, 0), xytext=(-15, 60),␣
↪textcoords='offset points', color='#ff0066', fontsize=15, weight='bold')
plt.annotate(f'{mist_clt_interval.mean:.2f}', rotation='vertical', xy=(
    mist_clt_interval.mean, 0), xytext=(-15, 60), textcoords='offset points',␣
↪color='#004d1a', fontsize=15, weight='bold')
plt.annotate(f'{clear_cloudy_clt_interval.mean:.2f}', rotation='vertical', xy=(
    clear_cloudy_clt_interval.mean, 0), xytext=(-15, 60), textcoords='offset␣
↪points', color='#0099ff', fontsize=15, weight='bold')

plt.axvline(light_rain_snow_clt_interval.confidence_interval_lower,␣
↪color='#1a75ff',
            linestyle='dotted', linewidth=2, label='Light Rain or Snow Bike␣
↪rent CI Lower')
# plt.axvline(heavy_rain_snow_clt_interval.confidence_interval_lower,␣
↪color='#001a00',
#             linestyle='dotted', linewidth=2, label='Heavy Rain or Snow Bike␣
↪rent CI Lower')
plt.axvline(mist_clt_interval.confidence_interval_lower, color='#cc4400',
            linestyle='dotted', linewidth=2, label='Mist Bike rent CI Lower')
plt.axvline(clear_cloudy_clt_interval.confidence_interval_lower,␣
↪color='#8c1aff',
            linestyle='dotted', linewidth=2, label='Clear & Cloudy Bike rent CI␣
↪Lower')

plt.annotate(f'{light_rain_snow_clt_interval.confidence_interval_lower:.2f}',␣
↪rotation='vertical', xy=(
    light_rain_snow_clt_interval.confidence_interval_lower, 0), xytext=(10,␣
↪120), textcoords='offset points', color='#1a75ff', fontsize=14)
# plt.annotate(f'{heavy_rain_snow_clt_interval.confidence_interval_lower:.2f}',␣
↪rotation='vertical', xy=(
#     heavy_rain_snow_clt_interval.confidence_interval_lower, 0), xytext=(10,␣
↪120), textcoords='offset points', color='#001a00', fontsize=14)
plt.annotate(f'{mist_clt_interval.confidence_interval_lower:.2f}',␣
↪rotation='vertical', xy=(
    mist_clt_interval.confidence_interval_lower, 0), xytext=(10, 120),␣
↪textcoords='offset points', color='#cc4400', fontsize=14)
plt.annotate(f'{clear_cloudy_clt_interval.confidence_interval_lower:.2f}',␣
↪rotation='vertical', xy=(
    clear_cloudy_clt_interval.confidence_interval_lower, 0), xytext=(10, 120),␣
↪textcoords='offset points', color='#8c1aff', fontsize=14)
```

```
plt.axvline(light_rain_snow_clt_interval.confidence_interval_upper,␣
↪color='#1a75ff',
            linestyle='dotted', linewidth=2, label='Light Rain or Snow Bike␣
↪rent CI Upper')
# plt.axvline(heavy_rain_snow_clt_interval.confidence_interval_upper,␣
↪color='#001a00',
#             linestyle='dotted', linewidth=2, label='Heavy Rain or Snow Bike␣
↪rent CI Upper')
plt.axvline(mist_clt_interval.confidence_interval_upper, color='#cc4400',
            linestyle='dotted', linewidth=2, label='Mist Bike rent CI Upper')
plt.axvline(clear_cloudy_clt_interval.confidence_interval_upper,␣
↪color='#8c1aff',
            linestyle='dotted', linewidth=2, label='Clear & Cloudy Bike rent CI␣
↪Upper')

plt.annotate(f'{light_rain_snow_clt_interval.confidence_interval_upper:.2f}',␣
↪rotation='vertical', xy=(
    light_rain_snow_clt_interval.confidence_interval_upper, 0), xytext=(-20,␣
↪180), textcoords='offset points', color='#1a75ff', fontsize=14)
# plt.annotate(f'{heavy_rain_snow_clt_interval.confidence_interval_upper:.2f}',␣
↪rotation='vertical', xy=(
#     heavy_rain_snow_clt_interval.confidence_interval_upper, 0), xytext=(-20,␣
↪180), textcoords='offset points', color='#001a00', fontsize=14)
plt.annotate(f'{mist_clt_interval.confidence_interval_upper:.2f}',␣
↪rotation='vertical', xy=(
    mist_clt_interval.confidence_interval_upper, 0), xytext=(-20, 180),␣
↪textcoords='offset points', color='#cc4400', fontsize=14)
plt.annotate(f'{clear_cloudy_clt_interval.confidence_interval_upper:.2f}',␣
↪rotation='vertical', xy=(
    clear_cloudy_clt_interval.confidence_interval_upper, 0), xytext=(-20, 180),␣
↪textcoords='offset points', color='#8c1aff', fontsize=14)

plt.xlabel('Bike rent')
plt.ylabel('Frequency')
plt.title('Yulu Weather VS Bike rent')
plt.legend(bbox_to_anchor=(0.5, -0.37),
           loc='lower center', borderaxespad=0, ncol=4)


plt.show()
```



Insight

No. of cycles rented similar or different in different weather

```
Ho -> No. of Bike rented are Similar in different weather
Ha -> No. of Bike rented are different in different weather
```

`Using ANOVA, Kruskal & Levene Test Test p_value found that 0.0%.`

- With Confidence interval of 99% & Sample Size of 500
  - Mean Bike rent on `Light Snow, Light Rain + Thunderstorm + Scattered clouds, Light Rain + Scattered clouds` Weather is 113.18 with a Intervals of (102.31 - 124.04).
  - Mean Bike rent on `Mist + Cloudy, Mist + Broken clouds, Mist + Few clouds, Mist` Weather is 166.20 with a Intervals of (158.78 - 173.61).
  - Mean Bike rent on `Clear, Few clouds, partly cloudy, partly cloudy` Weather is 187.83 with a Intervals of (182.51 - 193.14).
  - As only 1 record available for `Heavy Rain + Ice Pallets + Thunderstorm + Mist, Snow + Fog` Weather, we are not going to consider this for Confidence Interval.
  - As 0% < 1% Thus Rejecting Ho & Accept Ha.

As per Anova Test we can Conclude "No. of Bike rented are different in different weather".

1.9.5 Q4. Weather is dependent on season

[ ]:
```
pd.crosstab(df_yulu['season'], df_yulu['weather'], margins=True)
```

[ ]:

weather	Clear or Cloudy	Heavy Rain or Snow	Light Rain or Snow	Mist	All
season					
Fall	1838	0	185	575	2598

```
Spring               1648                    1                  185    696    2530
Summer               1687                    0                  213    679    2579
Winter               1648                    0                  214    783    2645
All                  6821                    1                  797   2733   10352
```

Find Probability

Probability of a Season & Weather across all Combination *"Season Weather"*

```
[ ]: pd.crosstab(df_yulu['season'], df_yulu['weather'],
                 margins=True, normalize=True)
```

```
[ ]: weather  Clear or Cloudy  Heavy Rain or Snow  Light Rain or Snow      Mist  \
     season
     Fall            0.177550            0.000000            0.017871  0.055545
     Spring          0.159196            0.000097            0.017871  0.067233
     Summer          0.162964            0.000000            0.020576  0.065591
     Winter          0.159196            0.000000            0.020672  0.075638
     All             0.658906            0.000097            0.076990  0.264007

     weather       All
     season
     Fall     0.250966
     Spring   0.244397
     Summer   0.249131
     Winter   0.255506
     All      1.000000
```

Probability of Weather for given Seasons *"Weather / Seasons"*

```
[ ]: pd.crosstab(df_yulu['season'], df_yulu['weather'],
                 normalize='index', margins=True)*100
```

```
[ ]: weather  Clear or Cloudy  Heavy Rain or Snow  Light Rain or Snow       Mist
     season
     Fall           70.746728            0.000000            7.120862  22.132410
     Spring         65.138340            0.039526            7.312253  27.509881
     Summer         65.412951            0.000000            8.259015  26.328034
     Winter         62.306238            0.000000            8.090737  29.603025
     All            65.890649            0.009660            7.698995  26.400696
```

Probability of Season's for given Weather *"Season / Weather"*

```
[ ]: pd.crosstab(df_yulu['weather'], df_yulu['season'],
                 normalize='index', margins=True)*100
```

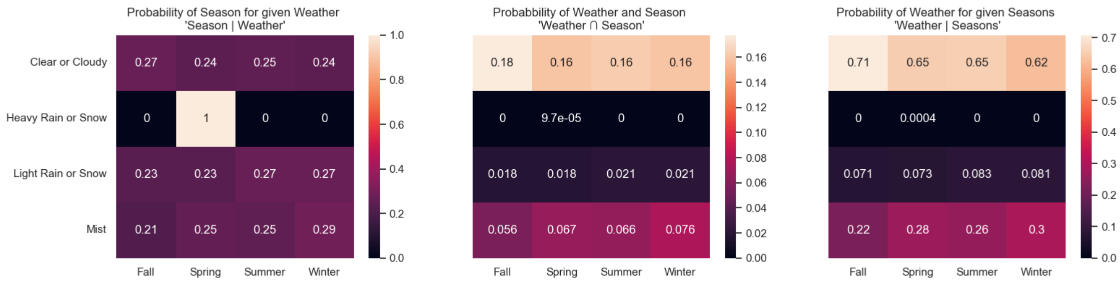
```
[ ]: season                    Fall     Spring     Summer     Winter
     weather
     Clear or Cloudy      26.946196  24.160680  24.732444  24.160680
```

```
Heavy Rain or Snow   0.000000  100.000000   0.000000   0.000000
Light Rain or Snow  23.212045   23.212045  26.725220  26.850690
Mist                21.039151   25.466520  24.844493  28.649835
All                 25.096600   24.439722  24.913060  25.550618
```

Plot the Graph to Support Assumptions

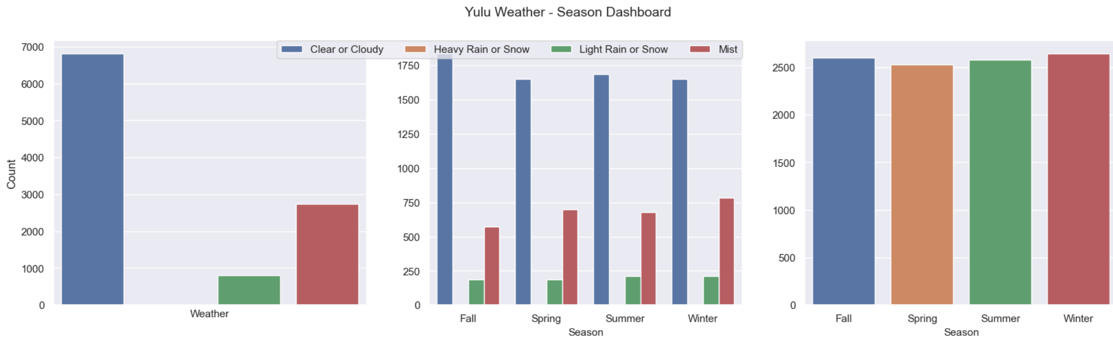
Heat Map

[ ]:
```python
plt.figure(figsize=(18, 4))
# plt.suptitle("Relation of Weather & Season")

plt.subplot(131)
sns.heatmap(pd.crosstab(
    df_yulu['weather'], df_yulu['season'], normalize='index'), annot=True)
plt.title("Probability of Season for given Weather \n'Season | Weather'", fontsize=12)
plt.ylabel("")
plt.xlabel("")

plt.subplot(132)
sns.heatmap(pd.crosstab(
    df_yulu['weather'], df_yulu['season'], normalize='all'), annot=True)
plt.title("Probabbility of Weather and Season \n'Weather   Season'", fontsize=12)
plt.yticks([])
plt.ylabel("")
plt.xlabel("")

plt.subplot(133)
sns.heatmap(pd.crosstab(
    df_yulu['weather'], df_yulu['season'], normalize='columns'), annot=True)
plt.title(
    "Probability of Weather for given Seasons \n'Weather | Seasons'", fontsize=12)
plt.yticks([])
plt.ylabel("")
plt.xlabel("")

plt.show()
```



Descriptive Plot

[ ]:
```
plt.figure(figsize=(20, 5)).suptitle(
    "Yulu Weather - Season Dashboard", fontsize=14)

plt.subplot(1, 3, 1)
sns.countplot(df_yulu, x='weather')
plt.ylabel('Count', fontsize=12)
plt.xlabel('Weather', fontsize=11)
plt.yticks(rotation=0, fontsize=11)
plt.xticks([], fontsize=11)

plt.subplot(1, 3, 2)
sns.countplot(df_yulu, x='season', hue='weather')
plt.ylabel('', fontsize=12)
plt.xlabel('Season', fontsize=11)
plt.yticks(rotation=0, fontsize=11)
plt.xticks(fontsize=11)
plt.legend(borderaxespad=0, ncol=4)

plt.subplot(1, 3, 3)
sns.countplot(df_yulu, x='season')
plt.ylabel('', fontsize=12)
plt.xlabel('Season', fontsize=11)
plt.yticks(rotation=0, fontsize=11)
plt.xticks(fontsize=11)

plt.show()
```



Hypothesis Testing Here, we found that Weather is Dependent on Season

[ ]:
```
Ho = "Weather is Independent of Season"
Ha = "Weather is Dependent on Season"
```

Using ChiSquare Test

[ ]:
```
stats, p_value, dof, data = chi2_contingency(
    pd.crosstab(df_yulu['season'], df_yulu['weather']))
round((stats*100), 4), round((p_value*100), 4)
```

[ ]:
```
(5043.71, 0.0)
```

[ ]:
```
print(Ha if (p_value < significance_level) else Ho)
```

```
Weather is Dependent on Season
```

Insight

Weather is dependent on season

```
Ho -> Weather is Independent of Season
Ha -> Weather is Dependent on Season
```

`Using Chi-Square Test p_value found that 0.0%.`

- As 0% < 1% Thus Rejecting Ho & Accept Ha.

As per Chi-Square Test we can Conclude “Weather is Dependent on Season”.

- This is also absorbed from Heat Map & Probability plot
  - As, Co-Relation Co-efficient for each Weather are mostly equal, except “Heavy Rain or Snow”.

1.10 Feature Engineering

[ ]:
```
yulu_feature = pd.read_csv(
    "./bike_sharing.csv", parse_dates=[0], dayfirst=True)
```

[ ]:
```
yulu_feature = remove_outlier(yulu_feature, 'count')
```

[ ]:
```
yulu_feature["time_slot"] = yulu_feature['datetime'].dt.hour.apply(
    lambda x:
    (1 if (x <= 4) else
     (2 if (x <= 9) else
      (3 if (x <= 16) else
       (4 if (
           x <= 21) else 5)
       )
      )
     )
)
```

[ ]:
```
yulu_feature.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 10583 entries, 0 to 10885
Data columns (total 13 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   datetime    10583 non-null  datetime64[ns]
 1   season      10583 non-null  int64
 2   holiday     10583 non-null  int64
 3   workingday  10583 non-null  int64
 4   weather     10583 non-null  int64
 5   temp        10583 non-null  float64
 6   atemp       10583 non-null  float64
 7   humidity    10583 non-null  int64
 8   windspeed   10583 non-null  float64
 9   casual      10583 non-null  int64
 10  registered  10583 non-null  int64
 11  count       10583 non-null  int64
 12  time_slot   10583 non-null  int64
dtypes: datetime64[ns](1), float64(3), int64(9)
memory usage: 1.1 MB
```

[ ]:
```
yulu_feature['count'].describe()
```

[ ]:
```
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
```

1.10.1 Segmentation Using Inner Quartile Range

[ ]:
```
# count_Q1 = yulu_feature['count'].quantile(0.25)
# count_median = yulu_feature['count'].median()
# count_mean = yulu_feature['count'].mean()
# count_Q3 = yulu_feature['count'].quantile(0.75)
# count_IQR = count_Q3 - count_Q1
# count_lower = count_Q1 - 1.5*count_IQR
# count_upper = count_Q3 + 1.5*count_IQR
```

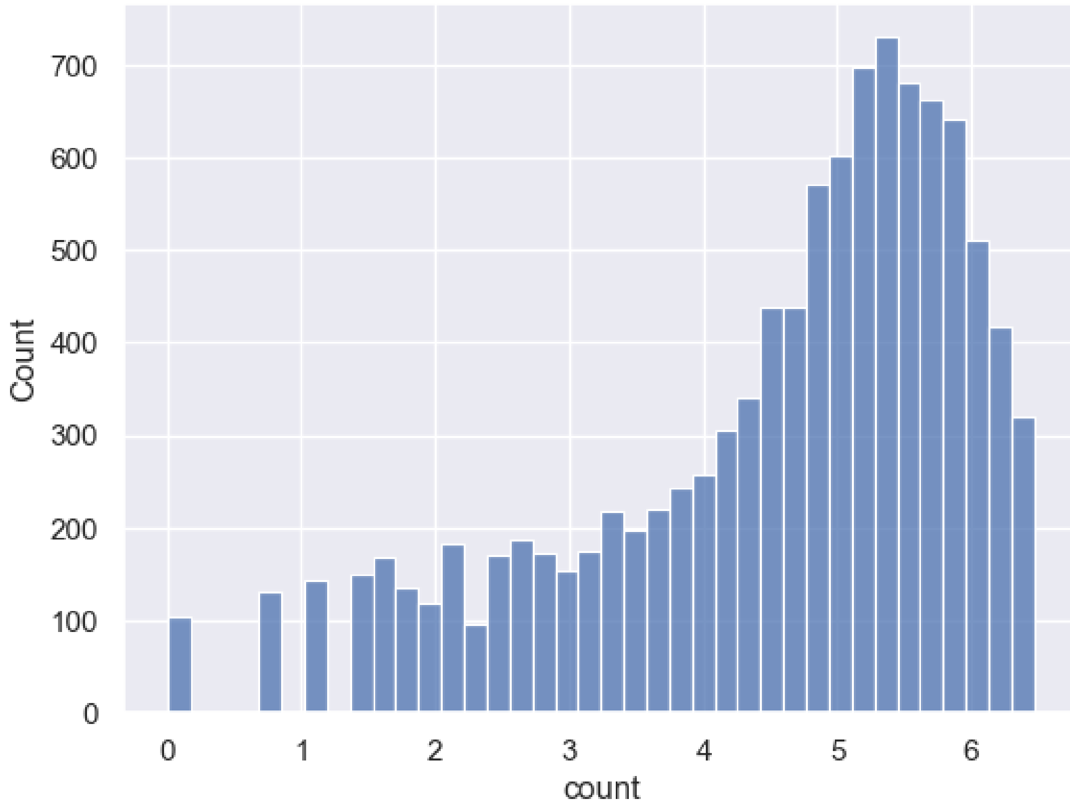
[ ]:
```
# def count_to_feature(x):
#     if (int(x) < count_lower):
#         return "Outliers"
#     elif (int(x) < count_Q1):
#         return 'Bed'
#     elif (int(x) < count_median):
#         return 'Not Good'
#     elif (int(x) < count_mean):
#         return 'Average'
#     elif (int(x) < count_Q3):
#         return 'Good'
#     elif (int(x) < count_upper):
#         return 'Exceptional'
#     else:
#         return "Outliers"


# yulu_feature['performance'] = yulu_feature.apply(
#     lambda row: count_to_feature(row['count']), axis=1)
```

[ ]:
```
# def target_feature(x):
#     if ((x == 'Outliers') or (x == 'Bed') or (x == 'Not Good')):
#         return 0
#     else:
#         return 1

# yulu_feature['target'] = yulu_feature.apply(
#     lambda row: target_feature(row['performance']), axis=1)
```

1.10.2 Segmentation Using Z-Score

[ ]:
```
sns.histplot(np.log(yulu_feature['count']))
```

[ ]: <Axes: xlabel='count', ylabel='Count'>



[ ]:
```
def count_to_feature(x):
    if (x < -1):
        return 'Bed'
    elif (x < -0.5):
        return 'Not Good'
    elif (x < 0.5):
        return 'Average'
    elif (x < 1):
        return 'Good'
    elif (x < 1.5):
        return 'Exceptional'
    else:
        return "Outliers"
    return (x)
```

```
yulu_feature['performance'] = zscore(yulu_feature['count']).apply(
    lambda x: count_to_feature(x))
```

[ ]:
```
yulu_feature['performance'].describe()
```

[ ]:
```
count        10583
unique           6
top        Average
freq          3415
Name: performance, dtype: object
```

[ ]:
```
yulu_feature['target'] = zscore(yulu_feature['count']).apply(
    lambda x: 1 if -(0.5) < x < (1.5) else 0)
```

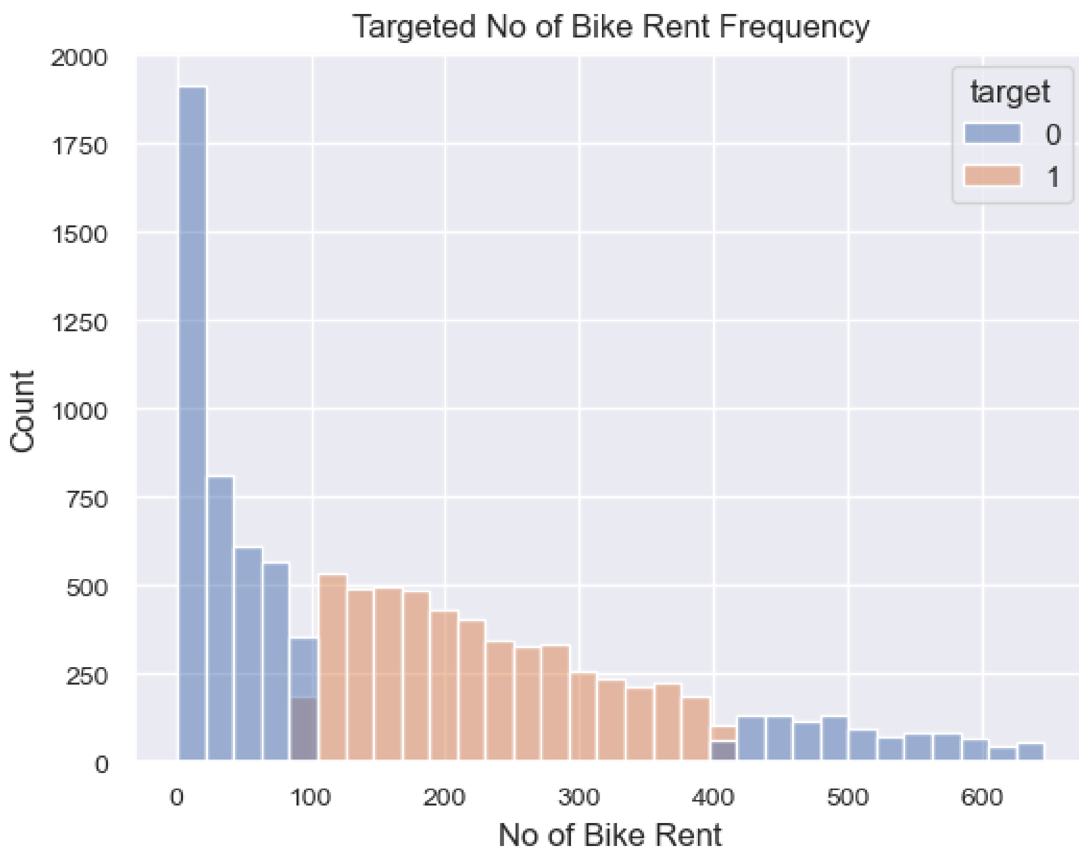
1.10.3 Test The Segmentation

[ ]:
```
yulu_feature.groupby(['target', 'performance'])['count'].count(
).reset_index().sort_values(['target', 'count'], ascending=False).
 ↪reset_index(drop=True)
```

[ ]:

	target	performance	count
0	1	Average	3415
1	1	Good	1088
2	1	Exceptional	751
3	0	Not Good	2461
4	0	Bed	1799
5	0	Outliers	1069

[ ]:
```
sns.histplot(data=yulu_feature, x='count', hue='target')
plt.title("Targeted No of Bike Rent Frequency")
plt.ylabel('Count', fontsize=12)
plt.xlabel('No of Bike Rent', fontsize=12)
plt.yticks(rotation=0, fontsize=10)
plt.xticks(fontsize=10)
# plt.legend(borderaxespad=0, ncol=4)
plt.show()
```



1.10.4 Co-Relation of Targeted Variable & Independent Variable

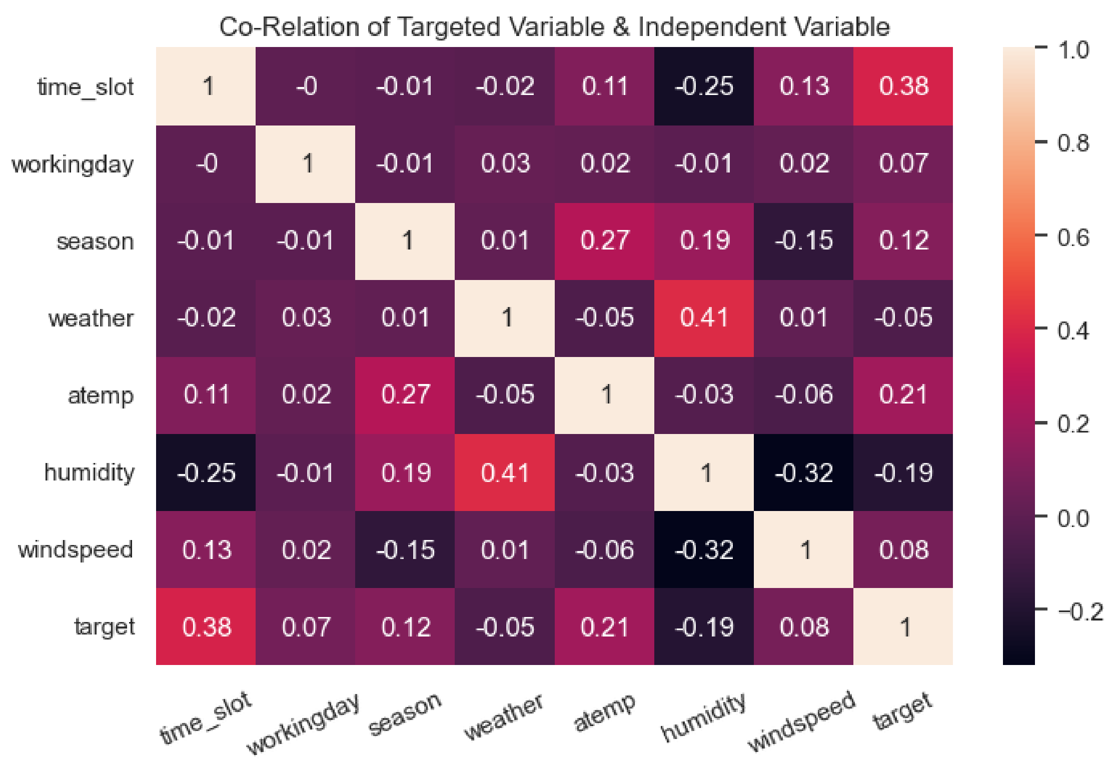
[ ]:
```
corelation_of_effective_variables = round(yulu_feature[['time_slot',␣
↪'workingday', 'season',  'weather',
                                                        'atemp', 'humidity',␣
↪'windspeed', 'target'
                                                        ]].corr(), 2)
```

[ ]:
```
corelation_of_effective_variables
```

[ ]:
```
            time_slot  workingday  season  weather  atemp  humidity  \
time_slot        1.00       -0.00   -0.01    -0.02   0.11     -0.25
workingday      -0.00        1.00   -0.01     0.03   0.02     -0.01
season          -0.01       -0.01    1.00     0.01   0.27      0.19
weather         -0.02        0.03    0.01     1.00  -0.05      0.41
atemp            0.11        0.02    0.27    -0.05   1.00     -0.03
humidity        -0.25       -0.01    0.19     0.41  -0.03      1.00
windspeed        0.13        0.02   -0.15     0.01  -0.06     -0.32
target           0.38        0.07    0.12    -0.05   0.21     -0.19
```

```
            windspeed  target
time_slot        0.13    0.38
workingday       0.02    0.07
season          -0.15    0.12
weather          0.01   -0.05
atemp           -0.06    0.21
humidity        -0.32   -0.19
windspeed        1.00    0.08
target           0.08    1.00
```

[ ]:
```
plt.figure(figsize=(8, 5))
plt.title("Co-Relation of Targeted Variable & Independent Variable")
sns.heatmap(corelation_of_effective_variables, annot=True)
plt.xticks(rotation=25)
plt.show()
```



2 END