

Business Case : Yulu - Hypothesis Testing

Yulu :

Yulu is an Indian micro-mobility service founded in 2017 that provides electric bikes and bicycles for rent, aimed at making urban commuting more sustainable and affordable. Users can rent these vehicles via a mobile app, which allows them to locate and unlock bikes using QR codes. Yulu operates in cities like Bangalore, Delhi, and Mumbai, focusing on reducing traffic congestion and pollution. The company uses a dockless model with easy battery-swapping options to ensure the availability of its electric vehicles.

Business Problem :

The company wants to know that Which variables play a crucial role in forecasting the demand for shared electric cycles in the Indian market, and how effectively do these variables explain the variations in electric cycle demand?

```
#Required Libraries importing:
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
```

```
!pip install pandas_profiling
from ydata_profiling import ProfileReport
```

```
Collecting multimethod<2,>=1.4 (from ydata-profiling->pandas_profiling)
  Downloading multimethod-1.12-py3-none-any.whl.metadata (9.6 kB)
Requirement already satisfied: statsmodels<1,>=0.13.2 in /usr/local/lib/python3.10/dist-packages (from ydata-profiling->pandas_profiling)
Requirement already satisfied: typeguard<5,>=3 in /usr/local/lib/python3.10/dist-packages (from ydata-profiling->pandas_profiling)
Collecting imagehash==4.3.1 (from ydata-profiling->pandas_profiling)
  Downloading ImageHash-4.3.1-py2.py3-none-any.whl.metadata (8.0 kB)
Requirement already satisfied: wordcloud>=1.9.1 in /usr/local/lib/python3.10/dist-packages (from ydata-profiling->pandas_profiling)
Collecting dacite>=1.8 (from ydata-profiling->pandas_profiling)
  Downloading dacite-1.8.1-py3-none-any.whl.metadata (15 kB)
Requirement already satisfied: numba<1,>=0.56.0 in /usr/local/lib/python3.10/dist-packages (from ydata-profiling->pandas_profiling)
Collecting PyWavelets (from imagehash==4.3.1->ydata-profiling->pandas_profiling)
  Downloading pywavelets-1.7.0-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (9.0 kB)
Requirement already satisfied: pillow in /usr/local/lib/python3.10/dist-packages (from imagehash==4.3.1->ydata-profiling->pandas_profiling)
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.10/dist-packages (from jinja2<3.2,>=2.11.1->ydata-profiling)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib<3.10,>=3.5->ydata-profiling)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-packages (from matplotlib<3.10,>=3.5->ydata-profiling)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib<3.10,>=3.5->ydata-profiling)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib<3.10,>=3.5->ydata-profiling)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib<3.10,>=3.5->ydata-profiling)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib<3.10,>=3.5->ydata-profiling)
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.10/dist-packages (from matplotlib<3.10,>=3.5->ydata-profiling)
Requirement already satisfied: llvmlite<0.44,>=0.43.0dev0 in /usr/local/lib/python3.10/dist-packages (from numba<1,>=0.56.0->ydata-profiling)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas!=1.4.0,<3,>1.1->ydata-profiling)
Requirement already satisfied: tzdata>=2022.1 in /usr/local/lib/python3.10/dist-packages (from pandas!=1.4.0,<3,>1.1->ydata-profiling)
Requirement already satisfied: joblib>=0.14.1 in /usr/local/lib/python3.10/dist-packages (from phik<0.13,>=0.11.1->ydata-profiling)
Requirement already satisfied: annotated-types>=0.4.0 in /usr/local/lib/python3.10/dist-packages (from pydantic>=2->ydata-profiling)
Requirement already satisfied: pydantic-core==2.20.1 in /usr/local/lib/python3.10/dist-packages (from pydantic>=2->ydata-profiling)
Requirement already satisfied: typing-extensions>=4.6.1 in /usr/local/lib/python3.10/dist-packages (from pydantic>=2->ydata-profiling)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2.24.0->ydata-profiling)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2.24.0->ydata-profiling)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2.24.0->ydata-profiling)
```

Stored in directory: /root/.cache/pip/wheels/dd/91/29/a79cecb328d01739e64017b6fb9a1ab9d8cb1853098ec5966d

Successfully built htmlmin

Installing collected packages: htmlmin, PyWavelets, multimethod, dacite, imagehash, visions, phik, ydata-profiling, pandas_profil

Successfully installed PyWavelets-1.7.0 dacite-1.8.1 htmlmin-0.1.12 imagehash-4.3.1 multimethod-1.12 pandas_profiling-3.6.6 phik-

Downlading The Given "YULU" Dataset :

!gdown https://d2beiqkhq929f0.cloudfront.net/public_assets/assets/000/001/428/original/bike_sharing.csv?1642089089

Download...

From: https://d2beiqkhq929f0.cloudfront.net/public_assets/assets/000/001/428/original/bike_sharing.csv?1642089089

To: /content/bike_sharing.csv?1642089089

100% 648k/648k [00:00<00:00, 10.2MB/s]

```
df=pd.read_csv("/content/bike_sharing.csv?1642089089")
df
```

10886 rows × 12 columns

	datetime	season	holiday	workingday	weather	temp	atemp	humidity	windspeed	casual	registered	count
0	2011-01-01 00:00:00	1	0	0	1	9.84	14.395	81	0.0000	3	13	16
1	2011-01-01 01:00:00	1	0	0	1	9.02	13.635	80	0.0000	8	32	40
2	2011-01-01 02:00:00	1	0	0	1	9.02	13.635	80	0.0000	5	27	32
3	2011-01-01 03:00:00	1	0	0	1	9.84	14.395	75	0.0000	3	10	13
4	2011-01-01 04:00:00	1	0	0	1	9.84	14.395	75	0.0000	0	1	1
...
10881	2012-12-19 19:00:00	4	0	1	1	15.58	19.695	50	26.0027	7	329	336
10882	2012-12-19 20:00:00	4	0	1	1	14.76	17.425	57	15.0013	10	231	241
10883	2012-12-19 21:00:00	4	0	1	1	13.94	15.910	61	15.0013	4	164	168
10884	2012-12-19 22:00:00	4	0	1	1	13.94	17.425	61	6.0032	12	117	129
10885	2012-12-19 23:00:00	4	0	1	1	13.12	16.665	66	8.9981	4	84	88

ProfileReport(df)

Summarize dataset: 100%

70/70 [00:28<00:00, 1.39it/s, Completed]

Generate report structure: 100%

1/1 [00:12<00:00, 12.71s/it]

Render HTML: 100%

1/1 [00:02<00:00, 2.01s/it]

Overview

Brought to you by YData (https://ydata.ai/?utm_source=opensource&utm_medium=ydataprofiling&utm_campaign=report)

Dataset statistics

Number of variables	12
Number of observations	10886
Missing cells	0
Missing cells (%)	0.0%
Duplicate rows	0
Duplicate rows (%)	0.0%
Total size in memory	1020.7 KiB
Average record size in memory	96.0 B

Variable types

DateTime	1
Categorical	4
Numeric	7

Alerts

atemp is highly overall correlated with casual and 2 other fields (casual, season, temp)	High correlation
casual is highly overall correlated with atemp and 3 other fields (atemp, count, registered, temp)	High correlation
count is highly overall correlated with casual and 1 other fields (casual, registered)	High correlation
registered is highly overall correlated with casual and 1 other fields (casual, count)	High correlation
season is highly overall correlated with atemp and 1 other fields (atemp, temp)	High correlation
temp is highly overall correlated with atemp and 2 other fields (atemp, casual, season)	High correlation

Exploratory Data Analysis:

a. Examine dataset structure, characteristics, and statistical summary.

df.shape

(10886, 12)

df.size

130632

df.info()

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

```
11 count      10886 non-null int64
dtypes: float64(3), int64(8), object(1)
memory usage: 1020.7+ KB
```

```
df.describe()
```

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

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

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

```
df.columns
```

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

```
df.head()
```

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

Insight:Retreving first 5 rows of the data

```
df.tail()
```

	datetime	season	holiday	workingday	weather	temp	atemp	humidity	windspeed	casual	registered	count
10881	2012-12-19 19:00:00	4	0	1	1	15.58	19.695	50	26.0027	7	329	336
10882	2012-12-19 20:00:00	4	0	1	1	14.76	17.425	57	15.0013	10	231	241
10883	2012-12-19 21:00:00	4	0	1	1	13.94	15.910	61	15.0013	4	164	168
10884	2012-12-19 22:00:00	4	0	1	1	13.94	17.425	61	6.0032	12	117	129
10885	2012-12-19 23:00:00	4	0	1	1	13.12	16.665	66	8.9981	4	84	88

Insight:Retrieving last 5 rows of the data

b. Identify missing values and perform Imputation using an appropriate method.

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

```

0
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

Insights: Observed no null values present in the given dataset.

C. Identify and remove duplicate records.

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

```
0
```

Insights: Observed that there is no duplicated values present in the given dataset.

```
df.dtypes
```


```

0
datetime    object
season      int64
holiday     int64
workingday  int64
weather     int64
temp        float64
atemp       float64
humidity    int64
windspeed   float64
casual       int64
registered  int64
count       int64

```

dtype: object


```
df.nunique()
```



	0
datetime	10886
season	4
holiday	2
workingday	2
weather	4
temp	49
atemp	60
humidity	89
windspeed	28
casual	309
registered	731
count	822

```
df.describe()
```

```
df.head(3)
```



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

d. Analyzing the distribution of Numerical & Categorical variables, separately


Note: - season: 1: spring, 2: summer, 3: fall, 4: winter

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

```
def seasons_name(x):
    if x == 1:
        return "spring"
    elif x == 2:
        return 'summer'
    elif x == 3:
        return "fall"
    else:
        return "winter"
```

```
df['season'] = df['season'].apply(seasons_name)
```

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



	datetime	season	holiday	workingday	weather	temp	atemp	humidity	windspeed	casual	registered	count
9872	2012-10-15 17:00:00	winter	0	1	1	22.96	26.515	64	16.9979	96	670	766
9196	2012-09-06 13:00:00	fall	0	1	2	29.52	34.850	79	7.0015	39	152	191
3078	2011-07-17 04:00:00	fall	0	0	1	25.42	29.545	73	11.0014	4	6	10
2189	2011-05-18 03:00:00	summer	0	1	3	21.32	25.000	94	19.9995	6	3	9
4176	2011-10-06 01:00:00	winter	0	1	1	19.68	23.485	63	12.9980	5	17	22
5233	2011-12-12 03:00:00	winter	0	1	1	7.38	12.120	74	0.0000	0	2	2
1825	2011-05-02 23:00:00	summer	0	1	2	22.96	26.515	83	12.9980	16	48	64
6104	2012-02-10 14:00:00	spring	0	1	2	13.12	16.665	39	6.0032	8	138	146
7321	2012-05-04 10:00:00	summer	0	1	2	25.42	29.545	73	8.9981	45	154	199
6196	2012-02-14 10:00:00	spring	0	1	2	10.66	13.635	60	8.9981	8	90	98

```
df.season.unique()
```



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

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



	count
season	
winter	2734
summer	2733
fall	2733
spring	2686

df.season.unique()

```
df.season.nunique()
```



4

holiday : whether day is a holiday or not

```
def holiday_or_not(x):
    if x == 0:
        return "No Holiday"
    else:
        return "Holiday"
```

```
df["holiday"] = df["holiday"].apply(holiday_or_not)
```

```
df['holiday'].value_counts()
```



	count
holiday	
No Holiday	10575
Holiday	311

df.workingday.unique()

workingday :

if day is neither weekend nor holiday is 1, otherwise is 0.

```
def workingday_or_not(x):
    if x==0:
        return "No Working Day"
    else:
        return "Working Day"
```

```
df["workingday"] = df["workingday"].apply(workingday_or_not)
```

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



	count
workingday	
Working Day	7412
No Working Day	3474

df.weather.unique()

weather:

1: Clear, Few clouds, partly cloudy

2: Mist + Cloudy, Mist + Broken clouds, Mist + Few clouds, Mist

3: Light Snow, Light Rain + Thunderstorm + Scattered clouds, Light Rain + Scattered clouds

4: Heavy Rain + Ice Pellets + Thunderstorm + Mist, Snow + Fog

```
def weather_mapping(x):
    if x == 1:
        return "Clear"
    elif x == 2:
        return "Cloudy"
    elif x == 3:
        return "Light Rain"
    else:
        return "Heavy Rain"
```


```
df["weather"] = df["weather"].apply(weather_mapping)
```

```
df['weather'].value_counts()
```




	count
weather	
Clear	7192
Cloudy	2834
Light Rain	859
Heavy Rain	1

```
df.info()
```



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

```
df.describe()
```



	datetime	temp	atemp	humidity	windspeed	casual	registered	count
count	10886	10886.00000	10886.000000	10886.000000	10886.000000	10886.000000	10886.000000	10886.000000
mean	2011-12-27 05:56:22.399411968	20.23086	23.655084	61.886460	12.799395	36.021955	155.552177	191.574132
min	2011-01-01 00:00:00	0.82000	0.760000	0.000000	0.000000	0.000000	0.000000	1.000000
25%	2011-07-02 07:15:00	13.94000	16.665000	47.000000	7.001500	4.000000	36.000000	42.000000
50%	2012-01-01 20:30:00	20.50000	24.240000	62.000000	12.998000	17.000000	118.000000	145.000000
75%	2012-07-01 12:45:00	26.24000	31.060000	77.000000	16.997900	49.000000	222.000000	284.000000
max	2012-12-19 23:00:00	41.00000	45.455000	100.000000	56.996900	367.000000	886.000000	977.000000
std	NaN	7.79159	8.474601	19.245033	8.164537	49.960477	151.039033	181.144454

Insights :

The dataset contains 10,886 observations of bike rental data, with key statistical metrics summarized below:

Datetime: The data covers the period from January 1, 2011, to December 19, 2012, with the median datetime on January 1, 2012, at 20:30, and an average datetime of December 27, 2011, at 05:56:22.

Temperature (temp): Temperatures range from 0.82°C to 41.00°C, with a mean of 20.23°C, a standard deviation of 7.79°C, and a median of 20.50°C.

"Feels like" Temperature (atemp): The "feels like" temperature ranges from 0.76°C to 45.45°C, with a mean of 23.66°C, a standard deviation of 8.47°C, and a median of 24.24°C.

Humidity: Humidity ranges from 0% to 100%, with an average of 61.89%, a standard deviation of 19.25%, and a median of 62%.

Windspeed: Wind speeds vary between 0 and 56.997 km/h, with an average of 12.80 km/h, a standard deviation of 8.16 km/h, and a median of 13.00 km/h.

Casual Users: The number of casual users ranges from 0 to 367, with a mean of 36.02, a standard deviation of 49.96, and a median of 17.

Registered Users: Registered users vary from 0 to 886, with a mean of 155.55, a standard deviation of 151.04, and a median of 118.

Total Bike Rentals (count): The total hourly bike rentals range from 1 to 977, with an average of 191.57, a standard deviation of 181.14, and a median of 145.

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

	season	holiday	workingday	weather
count	10886	10886	10886	10886
unique	4	2	2	4
top	winter	No Holiday	Working Day	Clear
freq	2734	10575	7412	7192

Insights :

The dataset contains 10,886 entries and includes four categorical variables: season, holiday, working day, and weather.

The 'season' variable has four categories, with 'winter' being the most frequent, appearing 2,734 times.

The 'holiday' variable is split into two categories, with 'No Holiday' being the dominant category, occurring 10,575 times.

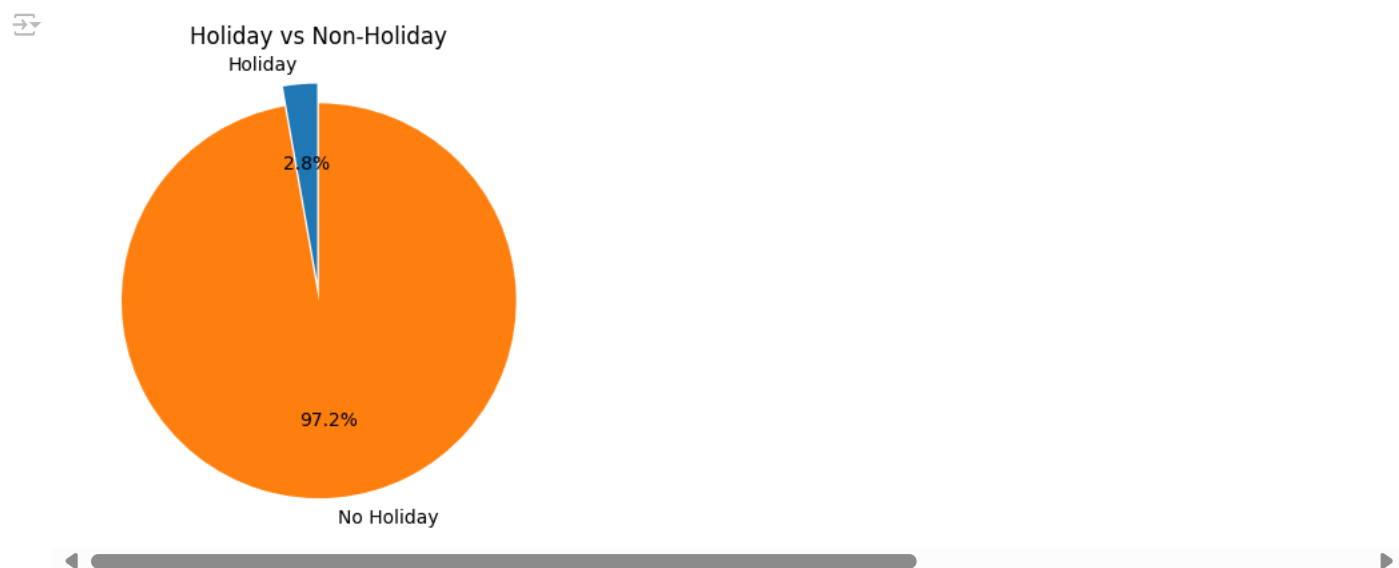
The 'working day' variable also has two categories, with 'Working Day' being the most common, recorded 7,412 times.

The 'weather' variable features four categories, with 'clear' conditions being the most frequent, observed 7,192 times.

```
holiday_counts = df.groupby("holiday")["count"].sum()
holiday_counts
```

	count
holiday	
Holiday	57808
No Holiday	2027668

```
plt.pie(holiday_counts, labels=holiday_counts.index, autopct='%1.1f%%', startangle=90, explode=[0.1, 0])
plt.title("Holiday vs Non-Holiday")
plt.show()
```



Insights :

In this dataset, approximately 97.2% of the days are non-holidays, and these days show higher bicycle rental counts.

```
workingdaycounts = df.groupby("workingday")["count"].sum().reset_index()
workingdaycounts
```

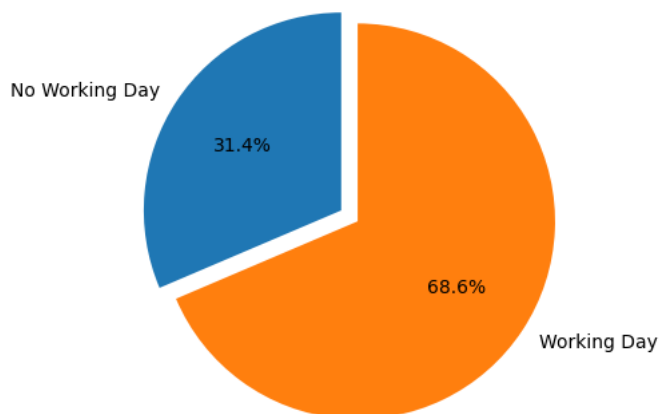


	workingday	count
0	No Working Day	654872
1	Working Day	1430604

```
plt.pie(workingdaycounts["count"], labels=workingdaycounts["workingday"], autopct='%1.1f%%', startangle=90, explode=[0.1, 0])
plt.title("Working Day vs No Working Day")
plt.show()
```



Working Day vs No Working Day



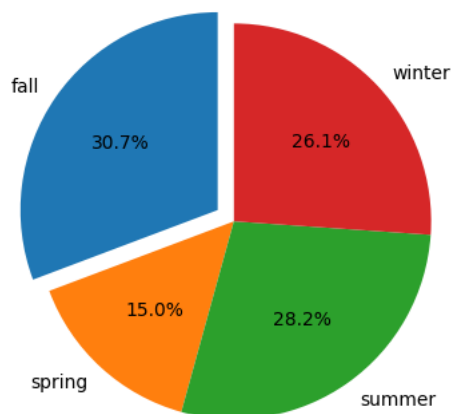
Insights:

In this dataset, around 68.6% of the days are classified as working days, which tend to have higher bicycle rental counts.

```
season_counts=df.groupby("season")["count"].sum().reset_index()
plt.pie(season_counts["count"], labels=season_counts["season"], autopct='%1.1f%%', startangle=90, explode=[0.1, 0,0,0])
plt.title("Season")
plt.show()
```



Season



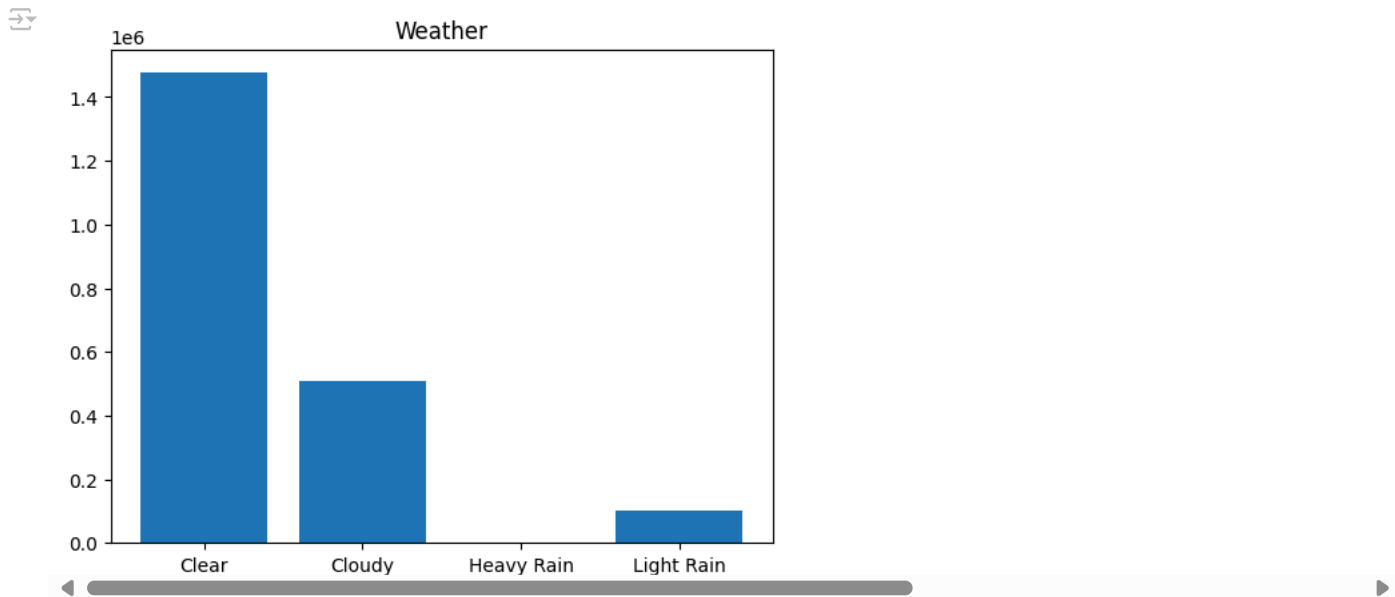
Insights : when the season is "fall" bicycle rentals are higher when compared with other seasons.

```
weather_counts=df.groupby("weather")["count"].sum().reset_index()
weather_counts
```



	weather	count
0	Clear	1476063
1	Cloudy	507160
2	Heavy Rain	164
3	Light Rain	102089

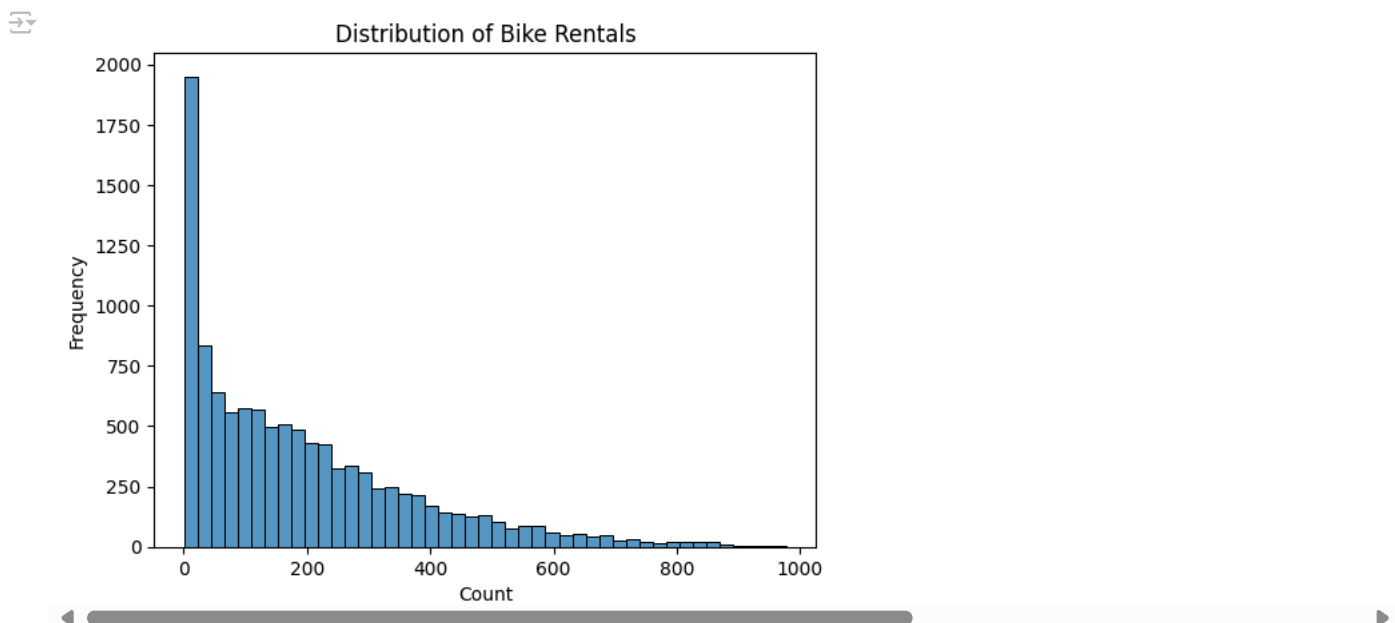
```
plt.bar(weather_counts["weather"],weather_counts["count"])
plt.title("Weather")
plt.show()
```



Insights:

Observed that when the weather is clear, bicycle rentals are higher when we compared with other variations in weather.

```
sns.histplot(df["count"])
plt.title("Distribution of Bike Rentals")
plt.xlabel("Count")
plt.ylabel("Frequency")
plt.show()
```

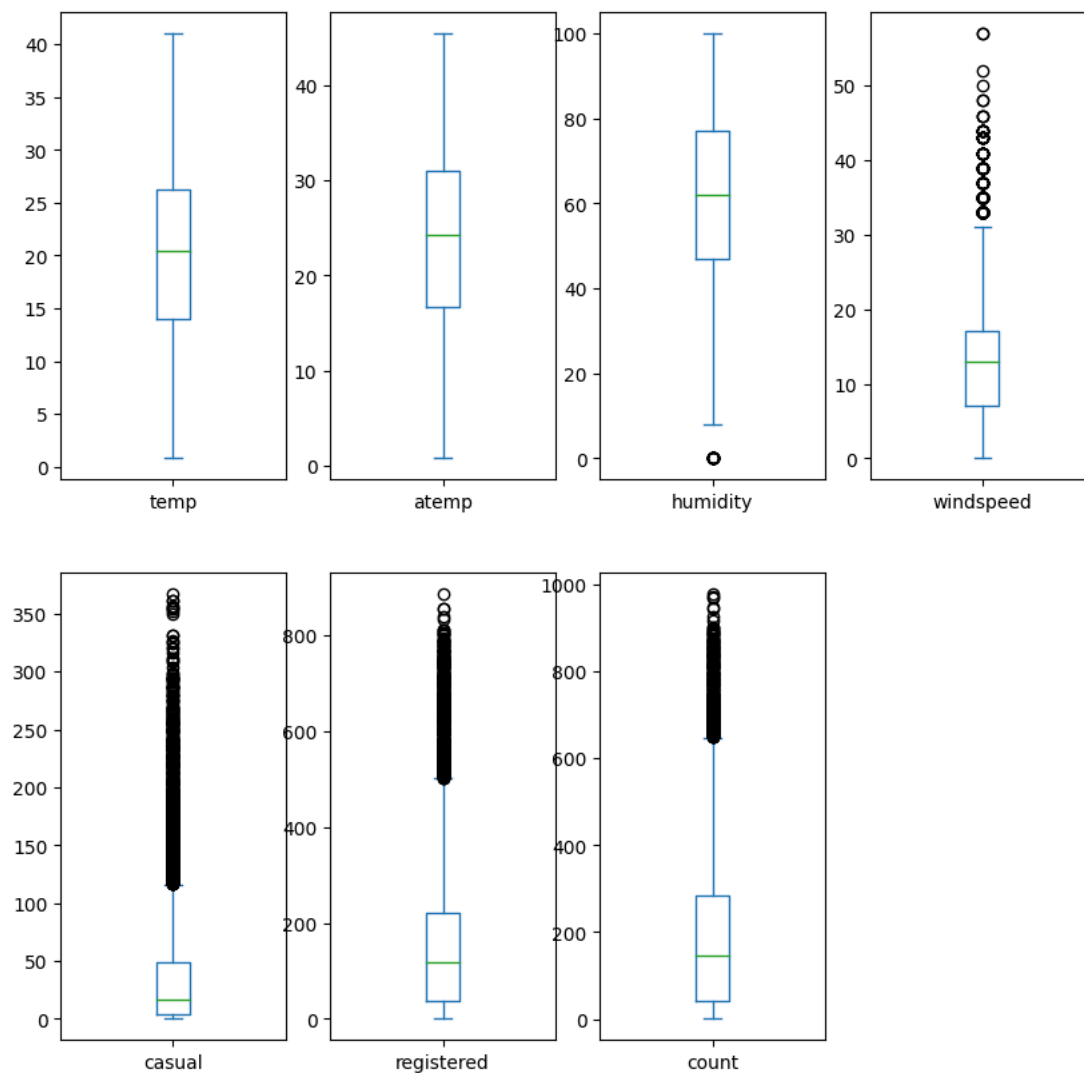


```
df.info()
```

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

```
11 count      10886 non-null int64
dtypes: datetime64[ns](1), float64(3), int64(4), object(4)
memory usage: 1020.7+ KB
```

```
df.plot(kind='box',subplots=True,layout=(2,4),figsize=(10,10))
plt.show()
plt.suptitle("Outliers for the attributes")
```



```
Text(0.5, 0.98, 'Outliers for the attributes')
```

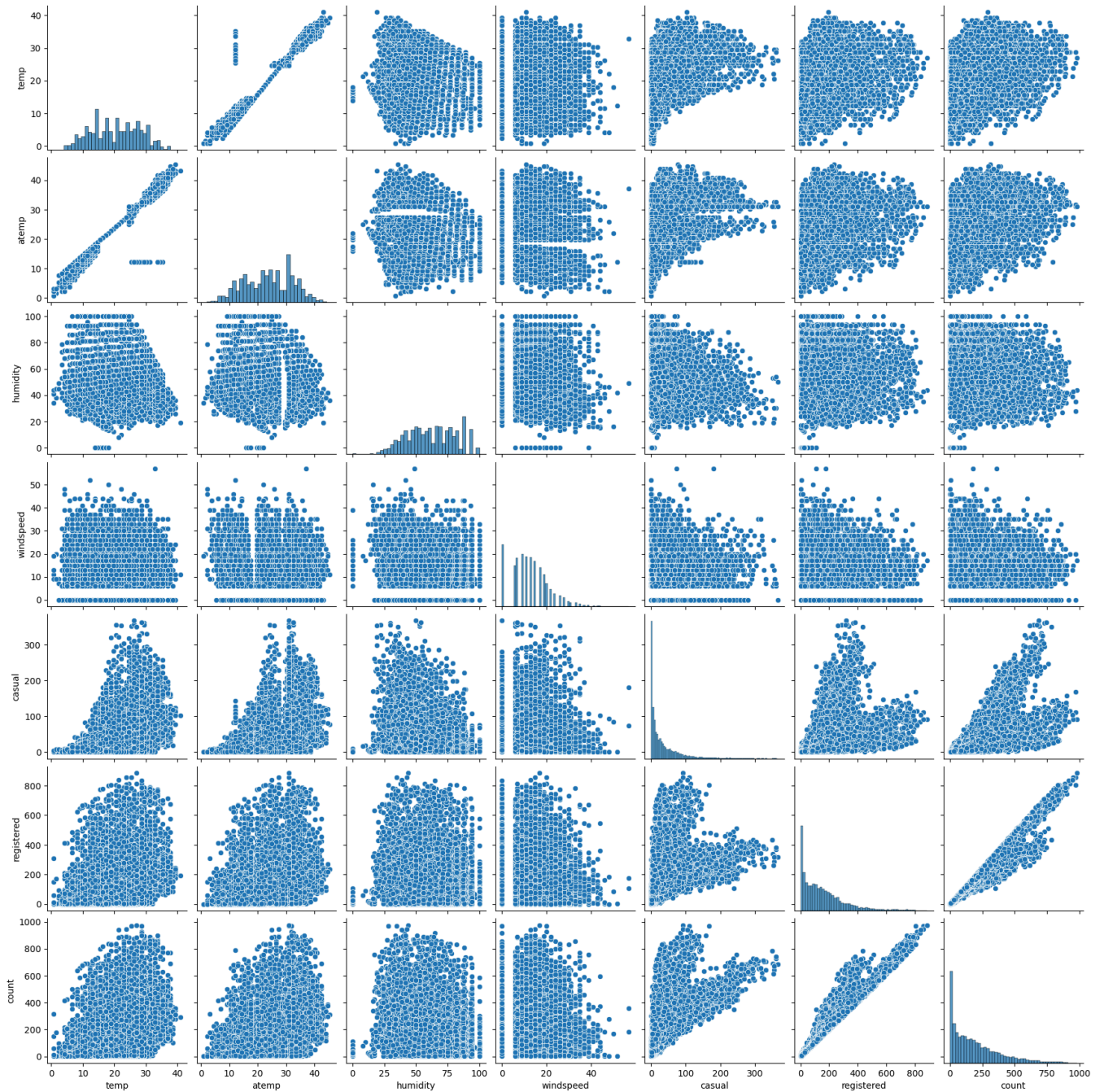
```
Figure size 640x480 with 0 Axes
```

Insights:

Observed that there are significant number of outliers in the casual, registered, count, and windspeed data and to avoid losing valuable information, we will keep these outliers.

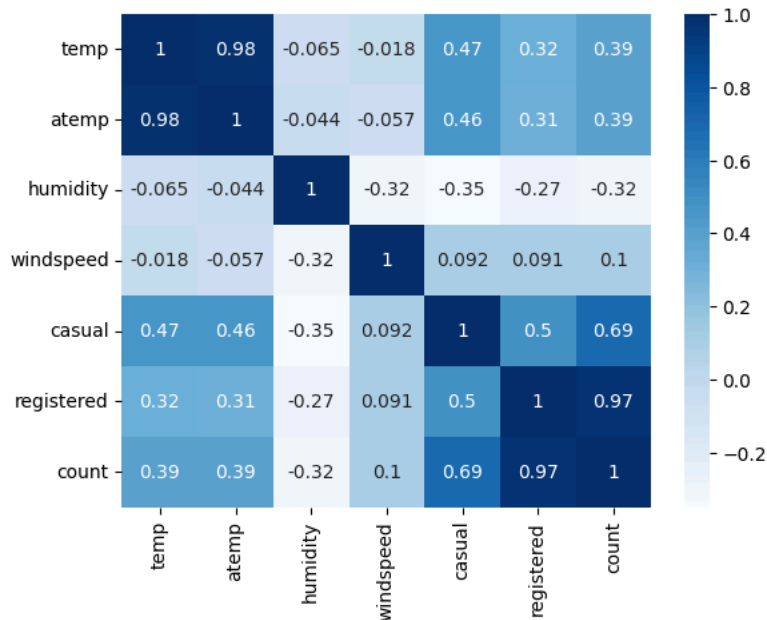
```
sns.pairplot(df)
```

 <seaborn.axisgrid.PairGrid at 0x7be3b498cac0>



```
corr_df=df.select_dtypes(include=['number'])
corr_df.corr()
sns.heatmap(corr_df.corr(),annot=True,cmap='Blues')
```

<Axes: >



Insights:

Observed that "atemp" is highly correlated with "temp" and we can drop that atemp column.

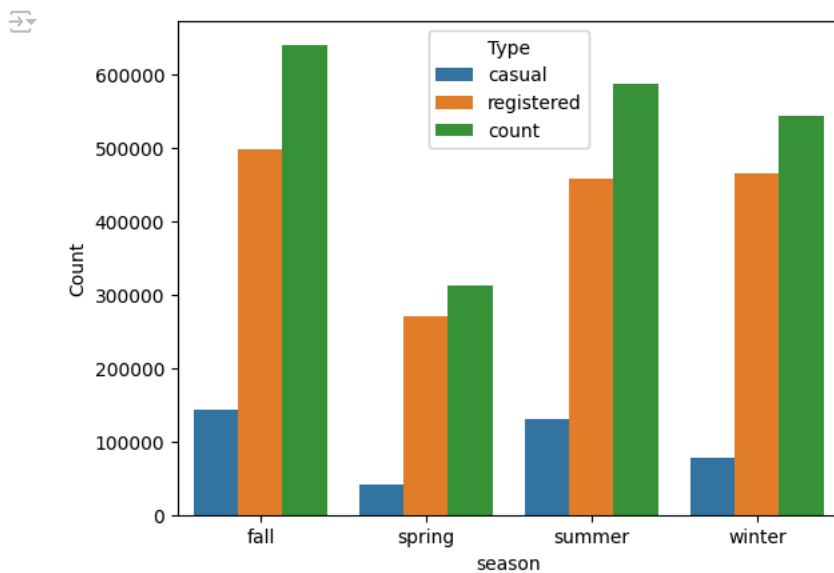
```
df.drop(['atemp'],axis=1,inplace=True)
```

```
df["Month"]=df["datetime"].dt.month_name().astype("category")
df["weekday"]=df["datetime"].dt.day_name().astype("category")
df["year"]=df["datetime"].dt.year
df["Day"]=df["datetime"].dt.day
df["Hour"]=df["datetime"].dt.hour
```

```
df.head(2)
```

	datetime	season	holiday	workingday	weather	temp	humidity	windspeed	casual	registered	count	Month	weekday	year	Day
0	2011-01-01 00:00:00	spring	No Holiday	No Working Day	Clear	9.84	81	0.0	3	13	16	January	Saturday	2011	1

```
seasonpercounts=df.groupby('season')[['casual','registered','count']].sum().reset_index()
seasonmelted=pd.melt(seasonpercounts,id_vars=['season'],var_name='Type',value_name='Count')
sns.barplot(x='season',y='Count',data=seasonmelted,hue='Type')
plt.show()
```



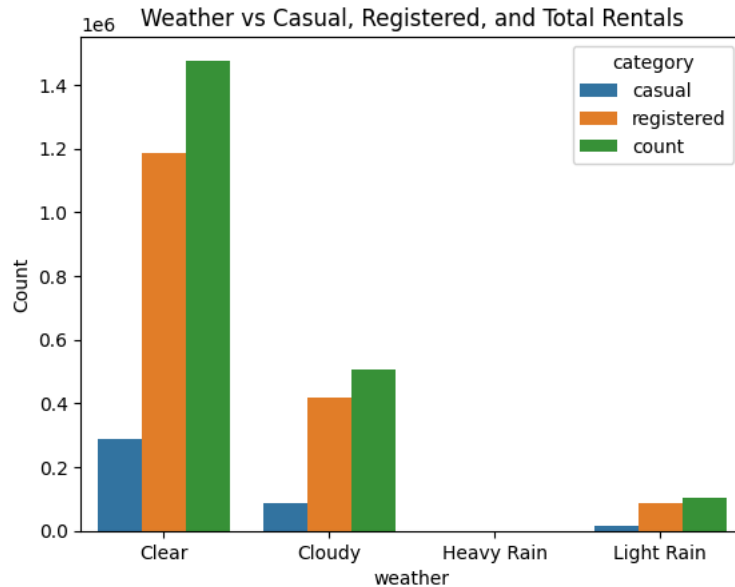
Insights:

Observed that during fall season there is a peak in demand.

winter and summer are both approximately equal in demand.

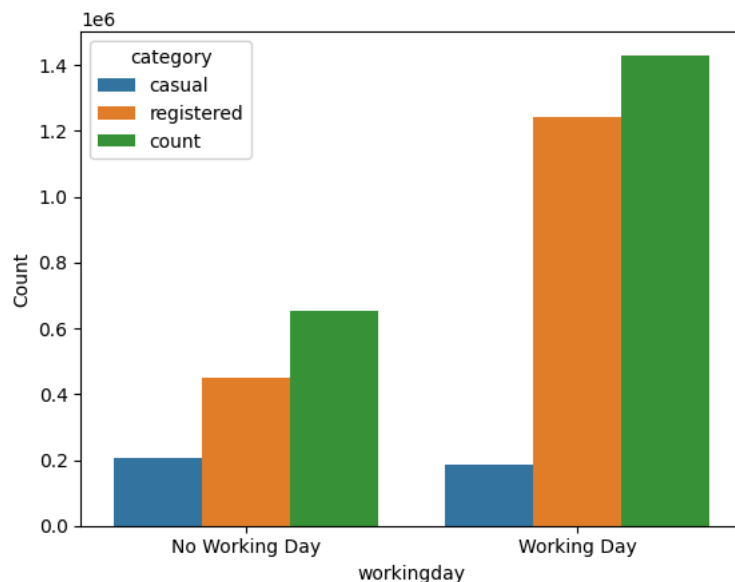
In spring season lower in demand.

```
weatherpercounts=df.groupby("weather")[["casual", "registered", "count"]].sum().reset_index()
weathermelted=pd.melt(weatherpercounts,id_vars="weather",var_name="category",value_name="Count")
sns.barplot(data=weathermelted,x="weather",y="Count",hue="category")
plt.title("Weather vs Casual, Registered, and Total Rentals")
plt.show()
```



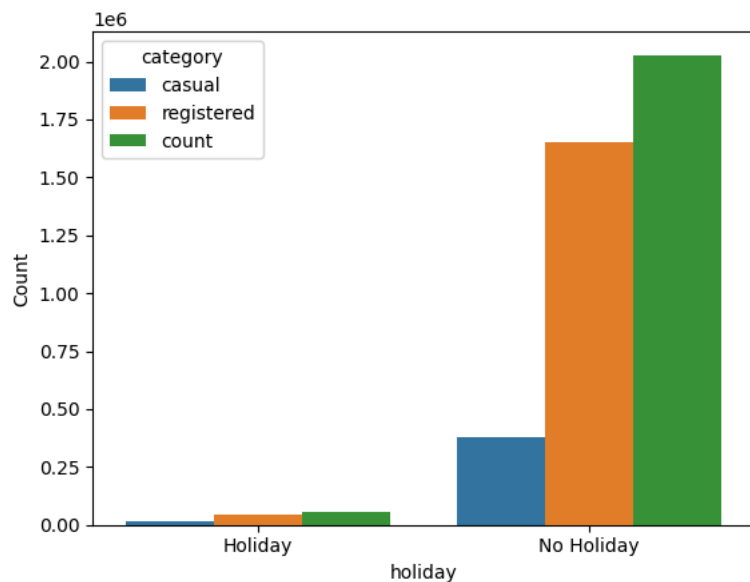
Insights: "Demand peaked during clear weather conditions and declined progressively as the weather deteriorated."

```
workingdaypercounts=df.groupby("workingday")[["casual", "registered", "count"]].sum().reset_index()
workingdaymelted=pd.melt(workingdaypercounts,id_vars="workingday",var_name="category",value_name="Count")
sns.barplot(data=workingdaymelted,x="workingday",y="Count",hue="category")
plt.show()
```



Insights: Cumulative demand is greater on a weekday.

```
holidaypercounts=df.groupby("holiday")[["casual", "registered", "count"]].sum().reset_index()
holidaymelted=pd.melt(holidaypercounts,id_vars="holiday",var_name="category",value_name="Count")
sns.barplot(data=holidaymelted,x="holiday",y="Count",hue="category")
plt.show()
```

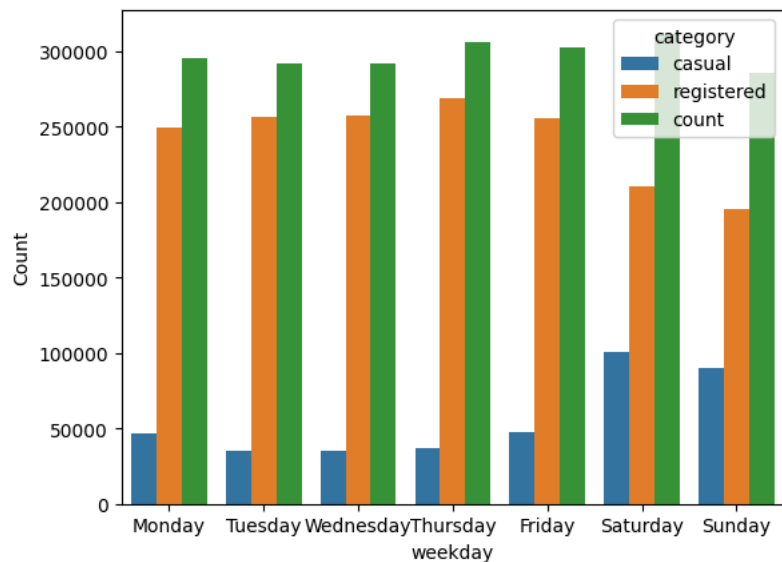


Insights : Demand is more on Non -holiday days

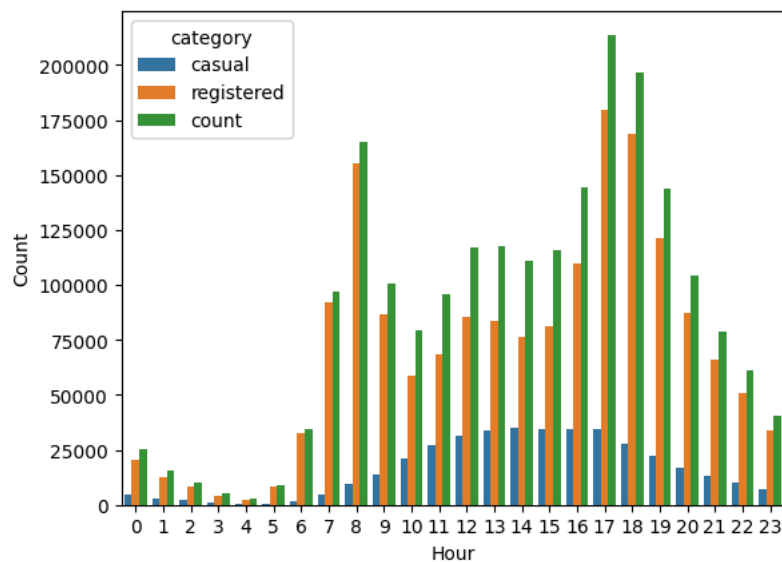
```
yearpercounts=df.groupby("year")[["casual","registered","count"]].sum().reset_index()
yearmelted=pd.melt(yearpercounts,id_vars="year",var_name="category",value_name="Count")
sns.barplot(data=yearmelted,x="year",y="Count",hue="category")
plt.show()
```



```
weekdaypercounts=df.groupby("weekday")[["casual","registered","count"]].sum().reset_index()
weekdaymelted=pd.melt(weekdaypercounts,id_vars="weekday",var_name="category",value_name="Count")
weekdayorder = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday']
weekdaymelted['weekday'] = pd.Categorical(weekdaymelted['weekday'], categories=weekdayorder, ordered=True)
sns.barplot(data=weekdaymelted,x="weekday",y="Count",hue="category")
plt.show()
```

```
hourpercounts=df.groupby("Hour")["casual","registered","count"].sum().reset_index()
hourmelted=pd.melt(hourpercounts,id_vars="Hour",var_name="category",value_name="Count")
sns.barplot(data=hourmelted,x="Hour",y="Count",hue="category")
plt.show()
```



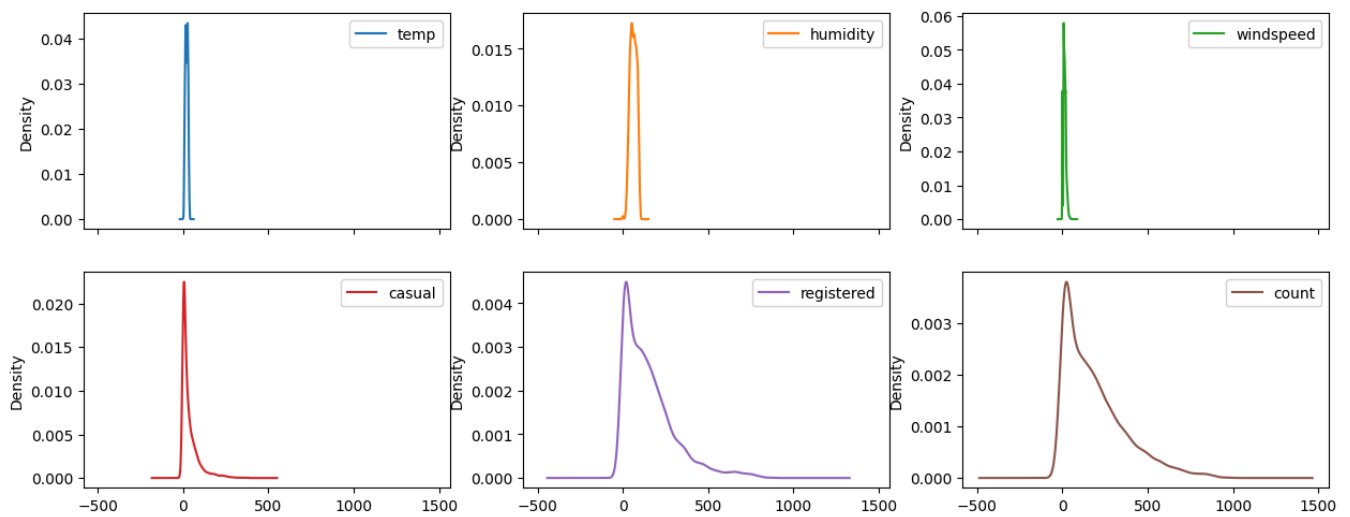
Insights:

Bicycle rentals peak in the evening, with high demand also in the morning, though slightly lower.

The afternoon sees moderate demand, while night and late-night hours have the least demand.

#Checking Distributions

```
df.drop(['datetime','Month','Day','Hour','weekday','year'],axis=1).plot(kind='kde',subplots=True,layout=(4,3),figsize=(15,12))
plt.show()
```



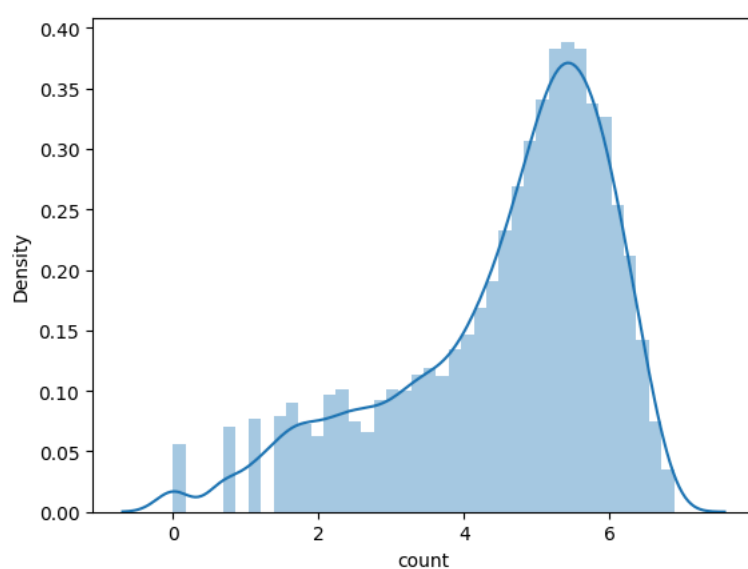
Insights:

Observed that windspeed,casual,registered,count were not normally distributed.

```
df.head(2)
```

	datetime	season	holiday	workingday	weather	temp	humidity	windspeed	casual	registered	count	Month	weekday	year	Day
0	2011-01-01 00:00:00	spring	No Holiday	No Working Day	Clear	9.84	81	0.0	3	13	16	January	Saturday	2011	1

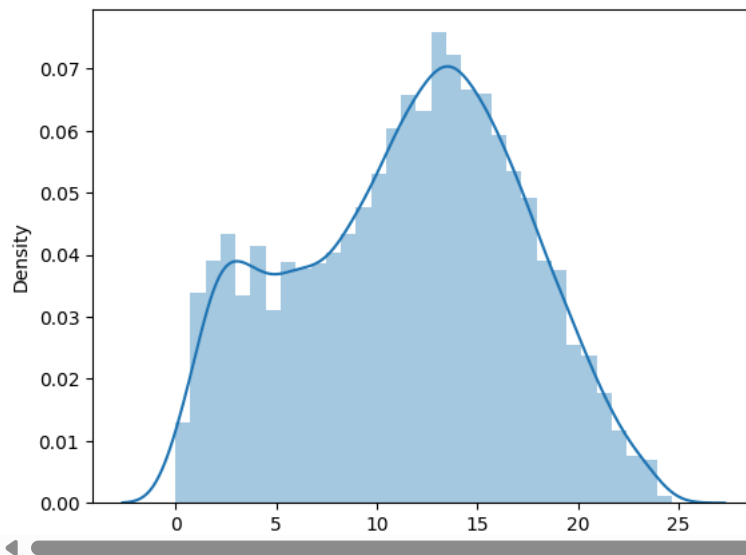
```
#To convert skewed distribution to normal distribution ,using lognormal
sns.distplot(np.log(df["count"]))
plt.show()
```



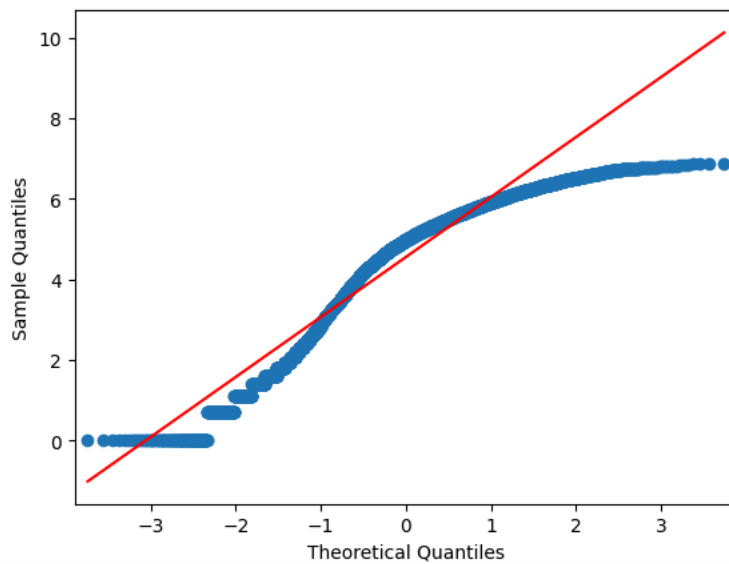
```
#To convert skewed distribution to normal distribution using boxcox
from scipy.stats import boxcox
transformed_value,lambda_value=boxcox(df["count"])
print("Transformed Value:",transformed_value)
print("Lambda Value:",lambda_value)
```

```
Transformed Value: [ 4.43314555  6.98267447  6.29227413 ... 12.79939319 11.52196593
 9.85125877]
Lambda Value: 0.3156702511500272
```

```
sns.distplot(transformed_value)
plt.show()
```



```
#qq plot to check normality
from statsmodels.graphics.gofplots import qqplot
qqplot(np.log(df["count"]),line="s")
plt.show()
```

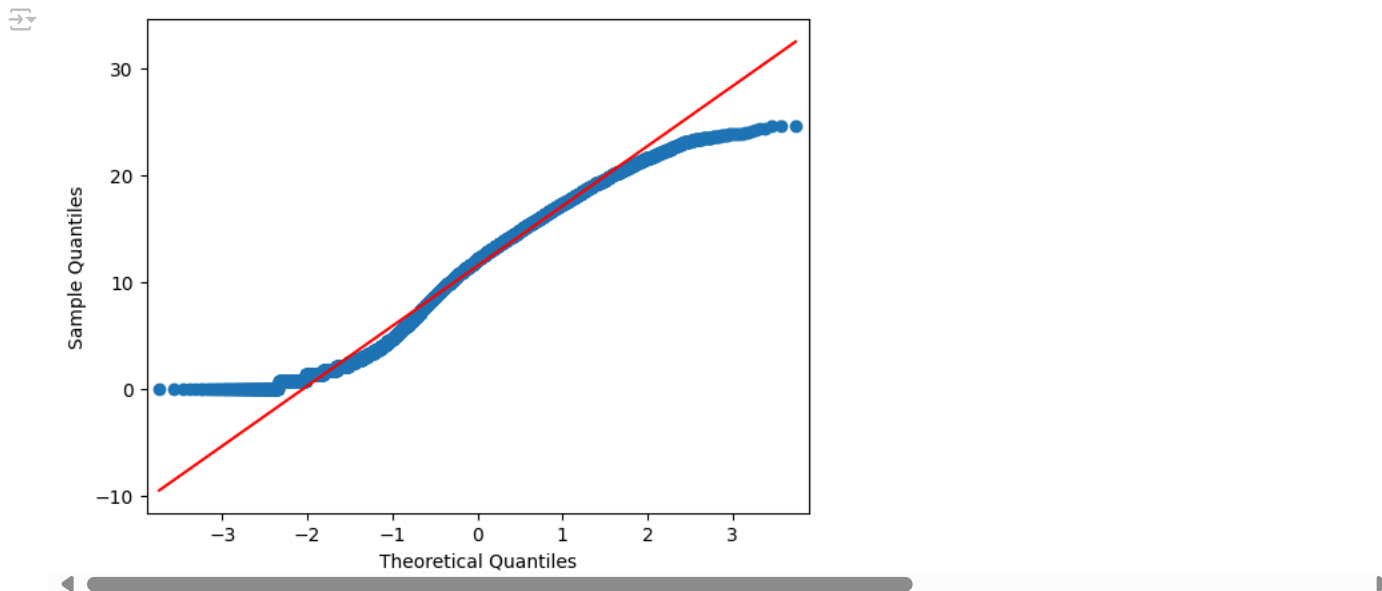


```
#To check normality -using shapiro test
#H0=Data is normally distributed
#HA=Data is not normally distributed
from scipy.stats import shapiro
tstats,pvalue= shapiro(np.log(df["count"]))
print("tstats:",tstats)
print("pvalue:",pvalue)
if pvalue < 0.05:
    print("Reject Null Hypothesis")
    print("Data is not normally distributed")
else:
    print("Accept Null Hypothesis")
    print("Data is normally distributed")
```



```
tstats: 0.9154006238583487
pvalue: 8.020775999667864e-61
Reject Null Hypothesis
Data is not normally distributed
```

```
qqplot(transformed_value,line="s")
plt.show()
```



```
# Shapiro's test to test for normality
# H0 : Data is Gaussian
# Ha : Data is not Gaussian

alpha=0.05

test_stats, p_value = shapiro(transformed_value)
print("Test statistic:",test_stats)
print("p-value:",p_value)

if p_value>alpha:
    print("Data is Gaussian")
else:
    print("Data is not Gaussian")
```

```
Test statistic: 0.9789787158306741
p-value: 4.082269779728272e-37
Data is not Gaussian
```

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

```
df["weekend"] = df["weekday"].apply(lambda x: 1 if x in ["Saturday", "Sunday"] else 0)
df["weekday"] = df["weekday"].apply(lambda x: 1 if x in ["Monday", "Tuesday", "Wednesday", "Thursday", "Friday"] else 0)
```

```
from scipy.stats import ttest_ind,ttest_rel
```

```
#Hypothesis Testing:
# H0 : there is no significant difference between the no. of bike rides on Weekdays and Weekends
# Ha : there is significant difference between the no. of bike rides on Weekdays and Weekends
alpha = 0.05
tstats,pvalue=ttest_ind(df[df["weekday"]==1]["count"],df[df["weekend"]==1]["count"])
print("tstats:",tstats)
print("pvalue:",pvalue)
if pvalue < 0.05:
    print("Reject Null Hypothesis")
    print("there is significant difference between the no. of bike rides on Weekdays and Weekends")
else:
    print("Fail to reject Null Hypothesis")
    print("there is no significant difference between the no. of bike rides on Weekdays and Weekends")
```

```
tstats: 1.0354386367292092
pvalue: 0.3004871142922829
Fail to reject Null Hypothesis
there is no significant difference between the no. of bike rides on Weekdays and Weekends
```

Insights :

Observed that the test is "Fail to reject Null Hypothesis" where $p > 0.05$.

There is no significant difference between the no. of bike rides on Weekdays and Weekends

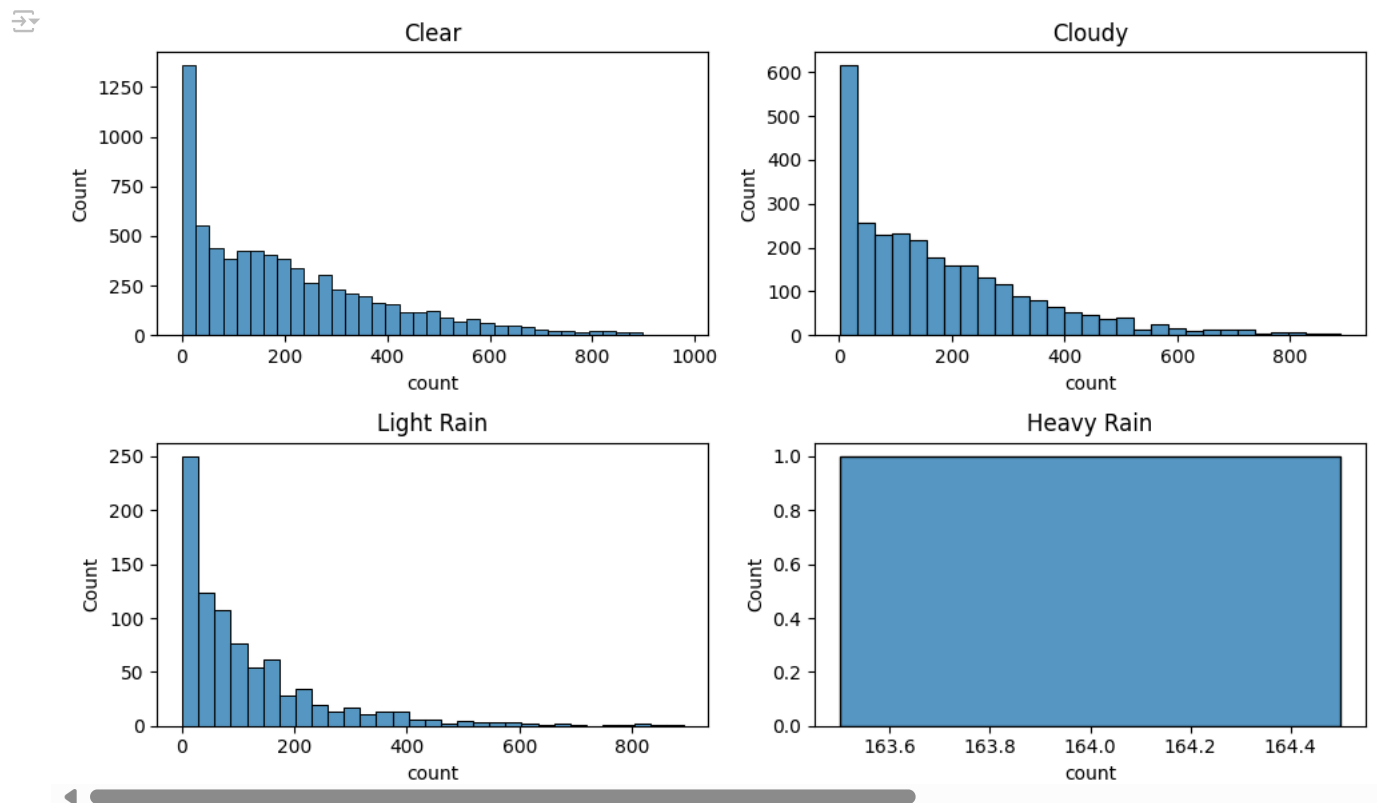
4. Check if the demand of bicycles on rent is the same for different Weather conditions?

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

weather	count
Clear	7192
Cloudy	2834
Light Rain	859
Heavy Rain	1

```
df_Clear = df[df["weather"]=="Clear"]["count"]
df_Cloudy = df[df["weather"]=="Cloudy"]["count"]
df_Light_Rain = df[df["weather"]=="Light Rain"]["count"]
df_Heavy_Rain = df[df["weather"]=="Heavy Rain"]["count"]
```

```
plt.figure(figsize=(10,6))
plt.subplot(2,2,1)
sns.histplot(df_Clear)
plt.title("Clear")
plt.subplot(2,2,2)
sns.histplot(df_Cloudy)
plt.title("Cloudy")
plt.subplot(2,2,3)
sns.histplot(df_Light_Rain)
plt.title("Light Rain")
plt.subplot(2,2,4)
sns.histplot(df_Heavy_Rain)
plt.title("Heavy Rain")
plt.tight_layout()
plt.show()
```



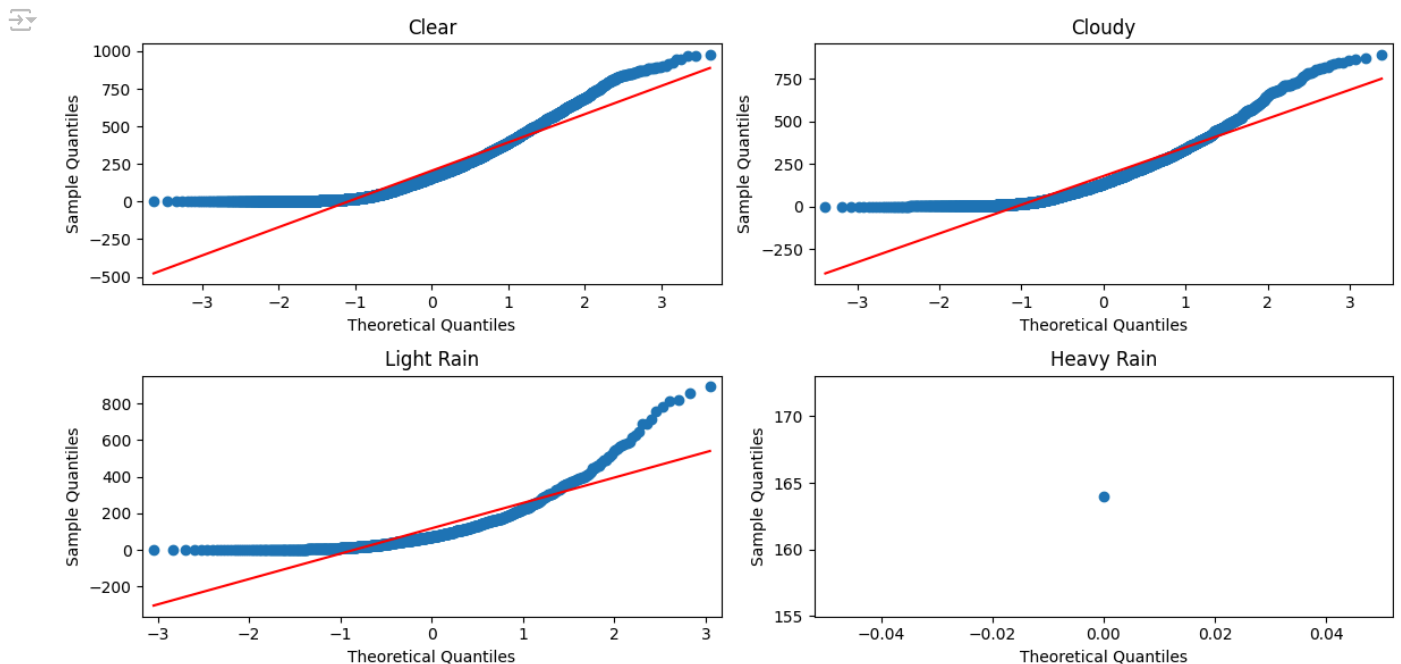
Insights: Observed that the data is not normally distributed...distributions are not normal.

```
#Using qqplot ,checking normal distributions for the given weather conditions
fig,axes=plt.subplots(2,2,figsize=(12,6)) #adjusting figures in a better size to fit in 2*2
```

```

qqplot(df_Clear, line="s", ax=axes[0,0])
axes[0,0].set_title("Clear")
qqplot(df_Cloudy, line="s", ax=axes[0,1])
axes[0,1].set_title("Cloudy")
qqplot(df_Light_Rain, line="s", ax=axes[1,0])
axes[1,0].set_title("Light Rain")
qqplot(df_Heavy_Rain, line="s", ax=axes[1,1])
axes[1,1].set_title("Heavy Rain")
plt.tight_layout()
plt.show()

```



Insights: From above plots too, Distributions are not normal

```

# Shapiro-Wilk test - to check the normality of data
print(shapiro(df_Clear))
print(shapiro(df_Cloudy))
print(shapiro(df_Light_Rain))
#print(shapiro(df_Heavy_Rain))

ShapiroResult(statistic=0.8909259459740138, pvalue=1.5964921477006555e-57)
ShapiroResult(statistic=0.8767694973495206, pvalue=9.777839106111785e-43)
ShapiroResult(statistic=0.7674327906035717, pvalue=3.875893017396149e-33)

```

Observed that by using shapiro test too the distributions are not normal. Note: PValue < 0.05 ..so rejecting null hypothesis means we are accepting alternative hypothesis.

```

#To check the variability of the data in each group - Levene's Test
#H0: Initially all the groups variances are equal
#HA: Variances are not equal
alpha=0.05
from scipy.stats import levene
levене_test, pvalue = levene(df_Clear, df_Cloudy, df_Light_Rain, df_Heavy_Rain)
print("levене_test:", levene_test)
print("pvalue:", pvalue)
if pvalue < 0.05:
    print("Reject Null Hypothesis")
    print("Variances are not equal")
else:
    print("Fail to reject Null Hypothesis")
    print("Variances are equal")

```

```

levене_test: 54.85106195954556
pvalue: 3.504937946833238e-35
Reject Null Hypothesis
Variances are not equal

```

Insights: From the above value we can say that the variances are unequal on the weather conditions.

```
#Anova Test
#H0:All groups have the same mean
#HA:At least one group has a different
from scipy.stats import f_oneway
f_stats,p_value=f_oneway(df_Clear,df_Cloudy,df_Light_Rain,df_Heavy_Rain)
print("f_stats:",f_stats)
print("pvalue:",p_value)
if pvalue < 0.05:
    print("Reject Null Hypothesis")
    print("At least one group has a different mean")
else:
    print("Fail to reject Null Hypothesis")
    print("All groups have the same mean")
```

```
f_stats: 65.53024112793271
pvalue: 5.482069475935669e-42
Reject Null Hypothesis
At least one group has a different mean
```

Insights : Observed that pvalue < 0.05. Demand is dependent on different weather conditions

```
#Checking Kruskal wallis test:
#H0:All groups have the same median
#HA:At least one group has a different median
from scipy.stats import kruskal
stats,p_value=kruskal(df_Clear,df_Cloudy,df_Light_Rain,df_Heavy_Rain)
print("f_stats:",f_stats)
print("pvalue:",p_value)
if pvalue < 0.05:
    print("Reject Null Hypothesis")
    print("At least one group has a different median")
else:
    print("Fail to reject Null Hypothesis")
    print("All groups have the same median")
```

```
f_stats: 65.53024112793271
pvalue: 3.501611300708679e-44
Reject Null Hypothesis
At least one group has a different median
```

Insights : Observed that pvalue < 0.05. Demand is dependent on different weather conditions

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

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

```
count
season
winter    2734
summer    2733
fall       2733
spring     2686
```

```
df.head(2)
```

```
datetime season holiday workingday weather temp humidity windspeed casual registered count Month weekday year Day
0 2011-01-01 spring No Holiday No Working Day Clear 9.84 81 0.0 3 13 16 January 0 2011 1
```

```
df_Winter = df[df["season"]=="winter"]["count"]
df_Spring = df[df["season"]=="spring"]["count"]
df_Summer = df[df["season"]=="summer"]["count"]
df_Fall = df[df["season"]=="fall"]["count"]
```

```
plt.figure(figsize=(15, 6)) # Set the figure size

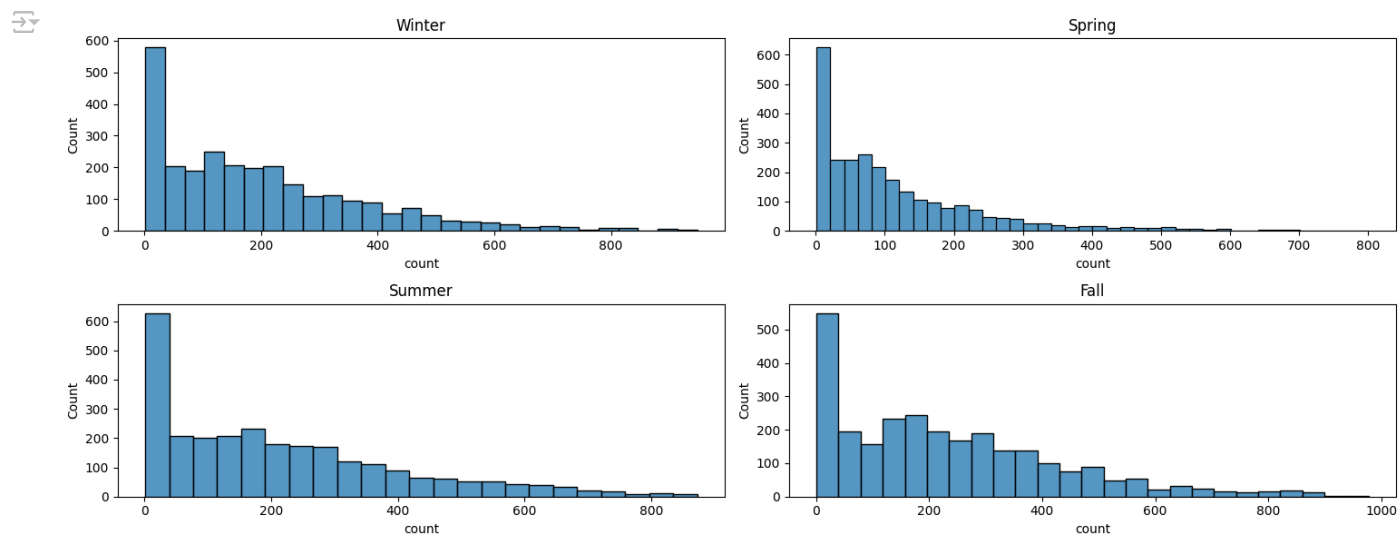
plt.subplot(2,2,1)
sns.histplot(df_Winter)
plt.title("Winter")

plt.subplot(2,2,2)
sns.histplot(df_Spring)
plt.title("Spring")

plt.subplot(2,2,3)
sns.histplot(df_Summer)
plt.title("Summer")

plt.subplot(2,2,4)
sns.histplot(df_Fall)
plt.title("Fall")

plt.tight_layout()
plt.show()
```

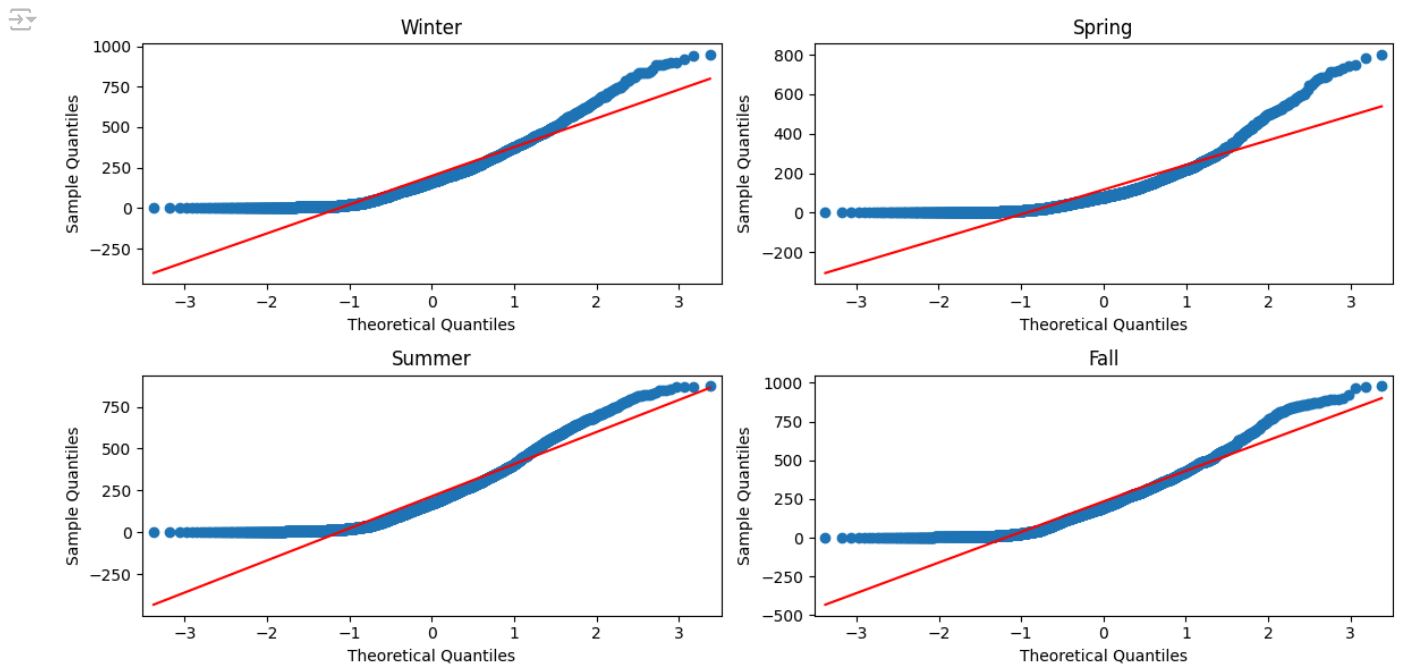


Insights:-Observed that distributions are not normal

```
from statsmodels.graphics.gofplots import qqplot

fig, axes = plt.subplots(2, 2, figsize=(12, 6)) #adjusting figures in a better size to fit in 2*2

qqplot(df_Winter, line="s", ax=axes[0, 0])
axes[0, 0].set_title("Winter")
qqplot(df_Spring, line="s", ax=axes[0, 1])
axes[0, 1].set_title("Spring")
qqplot(df_Summer, line="s", ax=axes[1, 0])
axes[1, 0].set_title("Summer")
qqplot(df_Fall, line="s", ax=axes[1, 1])
axes[1, 1].set_title("Fall")
plt.tight_layout()
plt.show()
```

Insights: From the above plots, the distributions are not normal.

```
# Shapiro-Wilk test - to check the normality of data
print(shapiro(df_Winter))
print(shapiro(df_Spring))
print(shapiro(df_Summer))
print(shapiro(df_Fall))
```

```
ShapiroResult(statistic=0.8954637482095505, pvalue=1.1299244409282836e-39)
ShapiroResult(statistic=0.8087378401253588, pvalue=8.749584618867662e-49)
ShapiroResult(statistic=0.9004818080893252, pvalue=6.039374406270491e-39)
ShapiroResult(statistic=0.9148166372899196, pvalue=1.043680518918597e-36)
```

Insights: Observed that distributions were not normal

Start coding or generate with AI.

```
#levene's test - to check the variability of the data with in each group
alpha=0.05
from scipy.stats import levene
stats,pvalue = levene(df_Winter,df_Spring,df_Summer,df_Fall)
print("stats:",stats)
print("pvalue:",pvalue)
if pvalue < 0.05:
    print("Reject Null Hypothesis")
    print("Variances are not equal")
else:
    print("Fail to reject Null Hypothesis")
    print("Variances are equal")
```

```
stats: 187.7706624026276
pvalue: 1.0147116860043298e-118
Reject Null Hypothesis
Variances are not equal
```

```
#Annova test-All groups have the same mean
#H0:All groups have the same mean
#HA:At least one group has a different
from scipy.stats import f_oneway
stats,p_value=f_oneway(df_Winter,df_Spring,df_Summer,df_Fall)
print("stats:",stats)
print("pvalue:",p_value)
if pvalue < 0.05:
    print("Reject Null Hypothesis")
    print("At least one group has a different mean")
```

```

else:
    print("Fail to reject Null Hypothesis")
    print("All groups have the same mean")

stats: 236.94671081032104
pvalue: 6.164843386499654e-149
Reject Null Hypothesis
At least one group has a different mean

```

Insights:

Observed that /we can say that there is a demand on the different seasons which is dependent.

```

#Kruskal wall's test
#H0:All groups have the same median
#HA:At least one group has a different median
from scipy.stats import kruskal

f_stats,p_value=kruskal(df_Winter,df_Spring,df_Summer,df_Fall)

print("f_stats:",f_stats)
print("p_value:",p_value)

alpha = 0.05
if pvalue < 0.05:
    print("Reject Null Hypothesis")
    print("At least one group has a different median")
else:
    print("Fail to reject Null Hypothesis")
    print("All groups have the same median")

f_stats: 699.6668548181988
p_value: 2.479008372608633e-151
Reject Null Hypothesis
At least one group has a different median

```


Insights:Demand is dependent on different seasons. pvalue -2.479008372608633e-151<0.05 .so rejecting null hypothesis..means we are considering alternative hypothesis.

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


```

df1=pd.crosstab(df[ 'weather' ],df[ 'season' ])
df1

```



	season	fall	spring	summer	winter
weather					
Clear	1930	1759	1801	1702	
Cloudy	604	715	708	807	
Heavy Rain	0	1	0	0	
Light Rain	199	211	224	225	



```

#chisquare test
#H0 : Weather conditions are independent of Seasons
# Ha : Weather conditions are dependent on Seasons
from scipy.stats import chi2_contingency
stats,p_value,dof,expected=chi2_contingency(df1)
print("stats:",stats)
print("p_value:",p_value)

alpha = 0.05
if p_value < alpha:
    print("Rejecting H0")
    print("Weather conditions are dependent on Seasons")
else:
    print("Failed to Reject H0")
    print("Weather conditions are independent of Seasons")

stats: 49.15865559689363
p_value: 1.5499250736864862e-07
Rejecting H0

```

Weather conditions are dependent on Seasons

Insights:

Pvalue<0.05..rjecting null hypothesis.. so we are accepting Ha hypothesis.

Weather conditions are dependent on Seasons

Overall Insights:

Demand Influences:

Working Days vs. Holidays: Demand remains relatively unaffected by whether it's a working day or a holiday.

Weekend Patterns: There is a noticeable increase in casual users on weekends.

Weather and Seasonal Effects: Weather and season are key drivers of demand fluctuations.

Clear Weather: Highest demand.

Light Rain or Snow: Lowest demand.

Seasonal Trends: Fall, summer, and winter see higher demand, while spring experiences the lowest.

Humidity: Has minimal influence on demand.

Temperature: A significant factor; demand increases as temperature rises.

Wind Speed: Plays a minor role.

Time of Day: Has a subtle impact; demand peaks in the evening, followed by morning and afternoon, with the lowest during night and late-night hours.

Data Characteristics:

Distributions: Variables such as "temp," "atemp," and "humidity" exhibit normal distributions. Outliers: Found in "humidity," "casual," "registered," and "count."

Impact of Seasons and Weather:

ANOVA Results: Confirm higher rentals in summer and fall, with spring showing significantly lower rentals. Weather Impact: Rentals drop during rainy, stormy, snowy, and foggy conditions.

Correlation Insights:

Positive Correlations: Between "count" and "registered," "count" and "casual," and "count" and temperature-related variables ("temp"/"atemp").

Negative Correlation: Between "count" and humidity.

Distribution Patterns: "Count," "registered," and "casual" align with a log-normal distribution.

Yearly and Monthly Trends:

Year-on-Year Growth: Rentals surged from 2011 to 2012.

Monthly Peaks: June sees the highest rentals, while January has the fewest.

Weather Influence: Rentals are lowest on snowy days and highest on clear days.

Statistical Analysis:

2-Sample T-Test: Indicates no significant impact of working days on rental counts.

ANOVA Analysis: Shows a substantial effect of weather and seasons on rentals, with clear weather and fall leading in demand, and spring lagging.

Chi-Square Test: Confirms that weather patterns are dependent on the season

Overall Recommendations:

Convert Casual Users to Registered Users:

Encourage casual users to become registered members by offering weekend sign-up discounts, aiming to grow the registered user base.

Optimize Bicycle Fleet Management:

Align bicycle deployment with demand trends to reduce costs. Scale back the number of bicycles during the low-demand spring season, and increase availability during the higher-demand summer, fall, and winter seasons.

Increase Casual User Engagement:

Implement targeted marketing strategies, such as first-time user discounts and referral incentives. Focus on promoting Yulu as a convenient commuting option on weekdays to boost casual user numbers.

Promotions During Peak Hours:

Introduce special deals during peak business hours to drive customer engagement. Tailor marketing efforts to account for weather and seasonal factors, and offer weather-specific features. Use customer data to customize promotions for different user segments.

Seasonal Marketing Campaigns:

Launch seasonal promotions to attract diverse customer groups, such as offering student discounts during summer or targeting school hours to appeal to students.

Start coding or [generate](#) with AI.

Double-click (or enter) to edit

Double-click (or enter) to edit

Double-click (or enter) to edit

Double-click (or enter) to edit

Double-click (or enter) to edit

Double-click (or enter) to edit

Double-click (or enter) to edit

Double-click (or enter) to edit