INDEX

Importing necessary Libraries

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from scipy.stats import poisson, binom, expon, geom
from scipy.stats import norm
from datetime import datetime

# for t-test
from scipy.stats import ttest_1samp
from scipy.stats import ttest_ind

# anova
from scipy.stats import f_oneway
from scipy.stats import kruskal
from scipy.stats import levene
from scipy.stats import shapiro

# chisquare
from scipy.stats import chi2_contingency
from scipy.stats import chisquare # Statistical test (chistat, pvalue)
```

Understanding/Overviewing Data

```
df = pd.read_csv("bike_sharing.csv")
df
```

	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

10886 rows × 12 columns

Next steps: Generate code with df | View recommended plots | New interactive sheet

```
df.shape
```

```
(10886, 12)
```

```
df.describe()
```

	season	holiday	workingday	weather	temp	atemp	humidity	windspeed	casual	re
count	10886.000000	10886.000000	10886.000000	10886.000000	10886.00000	10886.000000	10886.000000	10886.000000	10886.000000	1086
mean	2.506614	0.028569	0.680875	1.418427	20.23086	23.655084	61.886460	12.799395	36.021955	15
std	1.116174	0.166599	0.466159	0.633839	7.79159	8.474601	19.245033	8.164537	49.960477	15
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	3
50%	3.000000	0.000000	1.000000	1.000000	20.50000	24.240000	62.000000	12.998000	17.000000	11
75%	4.000000	0.000000	1.000000	2.000000	26.24000	31.060000	77.000000	16.997900	49.000000	22
max	4.000000	1.000000	1.000000	4.000000	41.00000	45.455000	100.000000	56.996900	367.000000	88

```
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.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

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

```
np.int64(0)
```

```
dtobject = pd.to_datetime(df["datetime"])
df["date"] = dtobject.dt.date
df["time"] = dtobject.dt.time
df
```

	datetime	season	holiday	workingday	weather	temp	atemp	humidity	windspeed	casual	registered	count	date	time
0	2011-01-01 00:00:00	1	0	0	1	9.84	14.395	81	0.0000	3	13	16	2011-01-01	00:00:00
1	2011-01-01 01:00:00	1	0	0	1	9.02	13.635	80	0.0000	8	32	40	2011-01-01	01:00:00
2	2011-01-01 02:00:00	1	0	0	1	9.02	13.635	80	0.0000	5	27	32	2011-01-01	02:00:00
3	2011-01-01 03:00:00	1	0	0	1	9.84	14.395	75	0.0000	3	10	13	2011-01-01	03:00:00
4	2011-01-01 04:00:00	1	0	0	1	9.84	14.395	75	0.0000	0	1	1	2011-01-01	04:00:00
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...

Next steps: Generate code with df | View recommended plots | New interactive sheet

```
from datetime import date
def is_weekend_weekday(check_date):
  if check_date.weekday() >= 5:  # 5 is Saturday, 6 is Sunday
      return 1  # It's a weekend
  else:
      return 0 # It's a weekday
df["Weekends"] = df["date"].apply(is_weekend_weekday )
df.head(5)
```

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

Next steps: Generate code with df | View recommended plots | New interactive sheet

Exploratory Data Analysis

Categorical Features

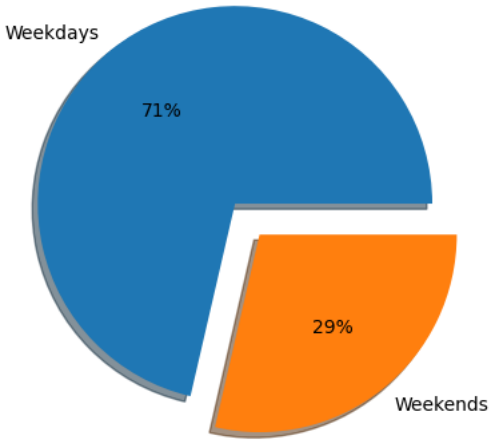
- Weekend Analysis
- Workingday Analysis
- Season Analysis
- Weather Analysis

```
weekend_data = df.groupby("Weekends")["count"].sum()
weekend_data
```

Weekends	count
0	1488412
1	597064

dtype: int64

```
plt.pie(weekend_data,labels=[ "Weekdays","Weekends"  ] , explode=[0.2, 0] ,shadow = True , autopct='%1.0f%%' )
plt.show()
```



- **Interpretations**
  - Higher Rental Volume on Weekdays:A significant majority of bike usage (71%) occurs on weekdays, suggesting that Yulu bikes are primarily used for daily commuting (e.g., office, college).
  - Lower Weekend Usage:Rentals drop on weekends (only 29%)

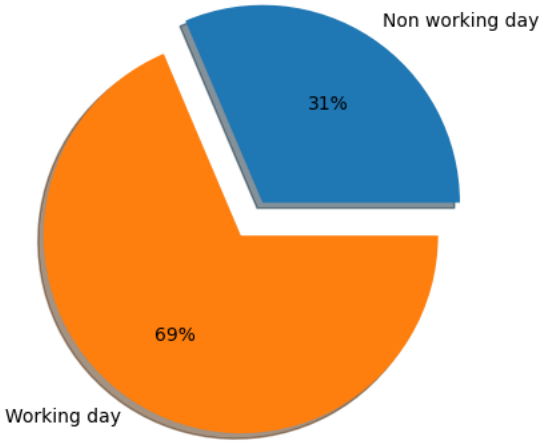
However, this chart shows total volume, not average daily usage. we are going to pair it with a t-test (in the later part) to confirm whether average weekday vs. weekend usage is statistically different.

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

workingday	count
0	654872
1	1430604

dtype: int64

```
plt.pie(workingday_data , labels=["Non working day" , "Working day" ] , explode=[0.2, 0] ,shadow = True , autopct='%1.0f%%' )
plt.show()
```



- **Interpretations**
  - Higher Rental Volume on working day:A significant majority of bike usage (69%) occurs on working days, suggesting that Yulu bikes are primarily used for daily commuting (e.g., office, college).
  - Lower non working day Usage:Rentals drop on non working day (only 31%)

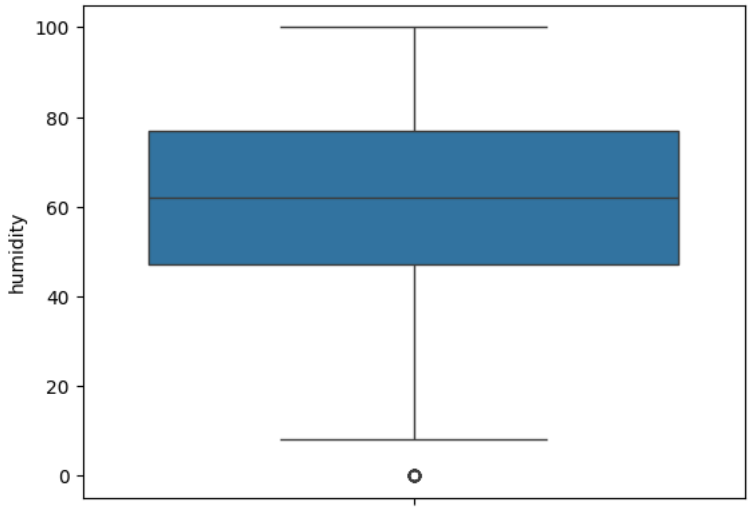
However, this chart shows total volume, not average daily usage .To confirm whether average working day vs. non working day usage is statistically different.

Numerical Features

- Weekend Analysis
- Workingday Analysis
- Season Analysis
- Weather Analysis

```
sns.boxplot(df['humidity'])
```

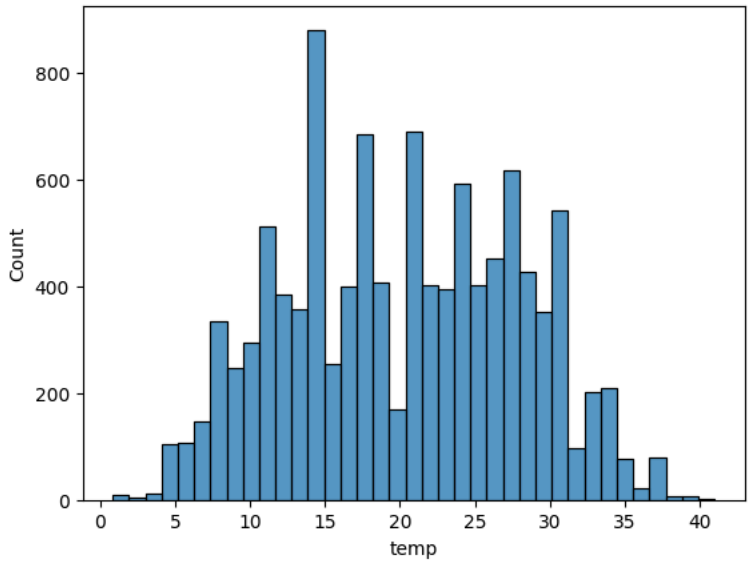
```
<Axes: ylabel='humidity'>
```



- **Interpretations**
  - Moderate to high humidity is dominant, so Yulu should ensure .Electric bikes are well-equipped for moist environments.
  - Protective gear (e.g., seat covers) might be important in humid regions. Humidity can affect user comfort and battery efficiency in electric bikes, so should be considered in demand forecasting models.

```
sns.histplot(x=df['temp'])
```

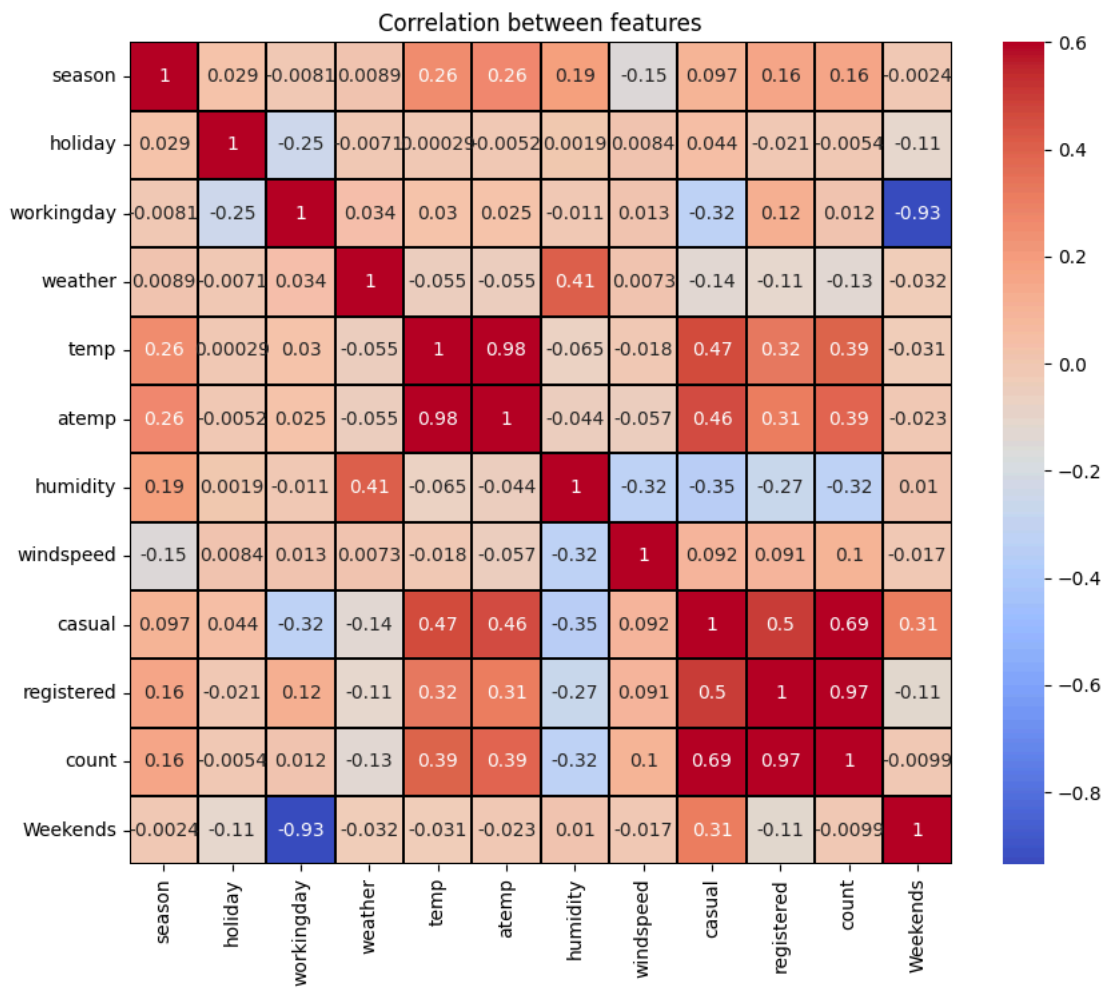
```
<Axes: xlabel='temp', ylabel='Count'>
```



- **Interpretations**
  - Temperature is well distributed between 5°C and 35°C.
  - Peak ridership seems to occur around 15°C–20°C.
  - The dataset contains few extreme temperature days, which may slightly affect normality but are realistic.

Establishing a Relationship between Variables

```
heatmap_df=df.select_dtypes(exclude=["object"])
plt.figure(figsize=(10,8))

correlation_values = heatmap_df.corr(method = 'pearson')
sns.heatmap(correlation_values, vmax = .6, linewidths=0.01, square=True, annot=True, cmap="coolwarm", linecolor="black")
plt.title('Correlation between features');
```



- **Interpretaions**
  - temp and atemp are almost perfectly correlated → one should be removed before running linear regression or other parametric models to avoid inflated coefficients.
  - Since registered users drive demand, retention and engagement strategies should target this segment.
  - Slightly lower demand on humid and poor-weather days — useful for real-time demand forecasting.

Weekdays- Weekends Demand Analysis

- Check if there any significant difference between the no. of bike rides on Weekdays and Weekends?
- For this we are employing T-Test (independent)

```
# bike ride on weekends
import random

weekend_df = df.loc[df["Weekends"]==1]

weekend_data = weekend_df.groupby(df["date"])["count"].sum().reset_index()
print(weekend_data)
weekend_daily=weekend_data["count"].values
weekend_daily
# a=np.array(random.choices(weekend_data["count"],k=200))
# a
```

```
           date  count
0    2011-01-01    985
1    2011-01-02    801
2    2011-01-08    959
3    2011-01-09    822
4    2011-01-15   1248
```

```
..          ...    ...
127  2012-12-02   4649
128  2012-12-08   5582
129  2012-12-09   3228
130  2012-12-15   5047
131  2012-12-16   3786

[132 rows x 2 columns]
array([ 985,  801,  959,  822, 1248, 1204, 1005, 1623, 1472, 1589, 1635,
       2077,  605, 2132, 2417, 3117, 2252, 3249, 2455, 2895,  795, 3744,
       3351, 4714, 4333, 3409, 4553, 5342, 4906, 4966, 4460, 5119, 4744,
       5119, 4649, 5336, 4881, 5923, 5302, 4294, 3785, 4150, 3820, 4484,
       4940, 5345, 5046, 4511, 4274, 2429, 2918, 5409, 5511, 5217, 5041,
       3926, 3649, 4067, 3717, 3663, 3614, 3485, 3190, 2743, 2739, 2431,
       2294, 4521, 3425, 2493, 2311, 2832, 2947, 2169, 1529, 4318, 2689,
       4066, 3423, 4118, 4911, 7836, 5892, 6041, 6857, 5169, 7460, 7132,
       6883, 6359, 7429, 6118, 8294, 8120, 7641, 7498, 6598, 7702, 6978,
       5531, 4840, 4672, 6969, 6031, 6824, 5464, 6299, 6544, 7865, 4549,
       6140, 5810, 5976, 8227, 8714, 7333, 7965, 3510, 7109, 6639, 5138,
       5107, 6536, 6852, 5629, 4669, 5191, 4649, 5582, 3228, 5047, 3786])
```

```
# bike ride on weekdays
import random

weekdays_df = df.loc[df["Weekends"]==0]

weekdays_data = weekdays_df.groupby(df["date"])["count"].sum().reset_index()
print(weekdays_data)
weekdays_daily=weekdays_data["count"].values
weekdays_daily
```

```
           date  count
0    2011-01-03   1349
1    2011-01-04   1562
2    2011-01-05   1600
3    2011-01-06   1606
4    2011-01-07   1510
..          ...    ...
319  2012-12-13   5532
320  2012-12-14   5611
321  2012-12-17   4585
322  2012-12-18   5557
323  2012-12-19   5267

[324 rows x 2 columns]
array([1349, 1562, 1600, 1606, 1510, 1321, 1263, 1162, 1406, 1421, 1000,
        683, 1650, 1360, 1526, 1550, 1708, 1712, 1530, 1605, 1538, 1746,
       1913, 1815, 2115, 2475, 2927, 1851, 2134, 1685, 1944, 1872, 2133,
       1891,  623, 1977, 2046, 2056, 2192, 2744, 3239, 2227, 3115, 1795,
       2808, 3141, 1471, 3348, 2034, 2162, 3267, 3126, 3429, 3204, 4401,
       4451, 2633, 4433, 4608, 4362, 4803, 4182, 4864, 4105, 3958, 4123,
       3855, 4575, 3974, 4968, 5312, 4548, 4833, 4401, 3915, 4586, 5020,
       4891, 5180, 3767, 4844, 5362, 6043, 4665, 4629, 4592, 4040, 4086,
       4258, 4342, 5084, 5538, 4458, 4541, 4266, 4845, 3574, 4576, 4866,
       4326, 4602, 4780, 4792, 4905, 4338, 4725, 4694, 3805, 4153, 5115,
       4727, 3351, 2710, 1996, 1842, 3544, 4713, 4763, 4785, 3659, 4760,
       4539, 3570, 4456, 4826, 4765, 4985, 5117, 4563, 2416, 2913, 3644,
       4570, 4748, 2424, 4068, 4186, 3974, 4046, 4035, 4205, 4109, 2933,
       3368, 4486, 4195, 1817, 3053, 3392, 3727, 3940, 3811, 2594,  705,
       3322, 3620, 3310, 3523, 3740, 3709, 3577, 3403, 1951, 2236, 2368,
       3272, 4098, 2376, 3598, 2177, 4097, 3214, 2298, 2935, 3376, 3292,
       4579, 3761, 4151, 3784, 4375, 2802, 3830, 3831, 3422, 3922, 4169,
       3005, 4154, 4990, 3194, 3333, 3956, 4916, 5382, 4569, 5298, 5847,
       6312, 6192, 4378, 6153, 5936, 6772, 6436, 6457, 6460, 5585, 5918,
       4862, 5409, 6398, 6370, 6691, 4367, 6565, 5740, 6169, 6421, 6296,
       6273, 5728, 4717, 6572, 7030, 2843, 5115, 7424, 7384, 7639, 4127,
       6998, 7001, 7055, 7494, 7736, 6664, 4972, 7421, 7363, 7665, 5099,
       6825, 6227, 6660, 7403, 6241, 6207, 6569, 6290, 7264, 7446, 7499,
       6830, 6786, 5713, 6591, 7580, 7261, 7175, 7013, 7273, 7534, 7286,
       5786, 6883, 6784, 7347, 7605, 7148, 6034, 6864, 7112, 6203, 7504,
       7525, 7767, 7870, 7804, 8009, 6869, 4073, 7591, 6778, 4639, 7572,
       7328, 8156, 5478, 6392, 7691, 7570, 7282, 5875, 7534, 7461, 7509,
       5424, 5986, 5847, 5259, 5686, 5035, 5315, 5992, 6269, 4094, 5495,
       5445, 5698, 5499, 6234, 6606, 5729, 5375, 5008, 5170, 5501, 5319,
       5532, 5611, 4585, 5557, 5267])
```

```
# Ho = there is no significance difference between bikerides in weekdays and weekends
# Ha = there is  significance difference between bikerides in weekdays and weekends

t_stats, pvalue = ttest_ind(weekdays_daily, weekend_daily, alternative ='two-sided')
print(t_stats, pvalue)
```

```
0.3657962794180527 0.7146874742951812
```

Since the p-value (0.7147) is much greater than the typical significance level of 0.05, we fail to reject the null hypothesis.

Conclusion:

There is no statistically significant difference between the average number of bike rides on weekdays and weekends.

Business Insight:

- Despite the higher total ride volume on weekdays (71%), the average daily ride
- count on weekdays vs. weekends does not differ significantly. This suggests:
  - The difference in total ride volume is likely due to more weekdays in the data, not a meaningful daily usage difference.
  - Yulu can consider uniform service planning across the week.
  - Marketing or resource allocation doesn't necessarily need to favor weekdays heavily.

Season - Demand Relation

- Checking if the demand of bicycles on rent is the same for different Seasons?
- For different seasons we are going to apply One-way ANOVA test Check assumptions of the test using -
  - Shapiro Test
  - Levene Test

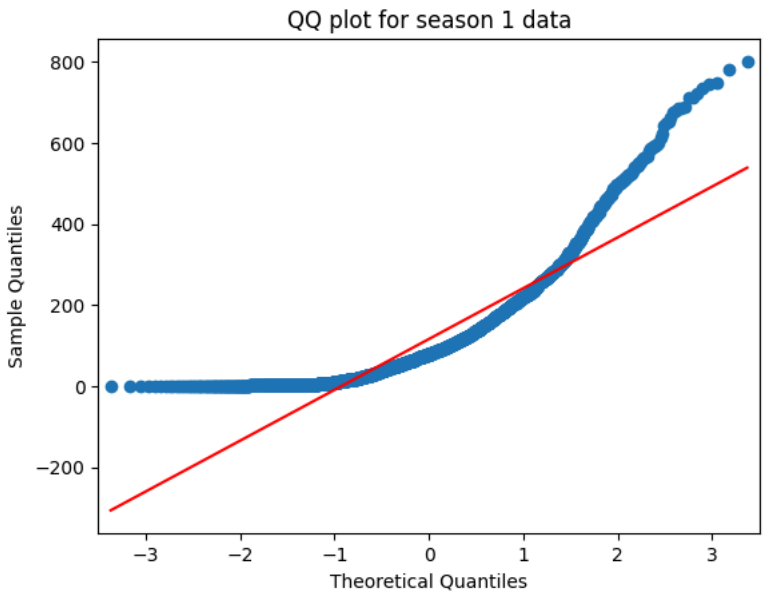
```
season1_data=df[df["season"]==1]["count"]
print("Shapiro test for  season 1" , shapiro(season1_data))

season2_data=df[df["season"]==2]["count"]
print("Shapiro test for  season 2" ,shapiro(season2_data))

season3_data=df[df["season"]==3]["count"]
print("Shapiro test for  season 3" ,shapiro(season3_data))

season4_data=df[df["season"]==4]["count"]
print("Shapiro test for  season 4" ,shapiro(season4_data))
```

```
Shapiro test for  season 1 ShapiroResult(statistic=np.float64(0.8087378401253588), pvalue=np.float64(8.749584618867662e-49))
Shapiro test for  season 2 ShapiroResult(statistic=np.float64(0.9004818080893252), pvalue=np.float64(6.039374406270491e-39))
Shapiro test for  season 3 ShapiroResult(statistic=np.float64(0.9148166372899196), pvalue=np.float64(1.043680518918597e-36))
Shapiro test for  season 4 ShapiroResult(statistic=np.float64(0.8954637482095505), pvalue=np.float64(1.1299244409282836e-39))
```

```
levene(season1_data, season2_data , season3_data  , season4_data)
```

```
LeveneResult(statistic=np.float64(187.7706624026276), pvalue=np.float64(1.0147116860043298e-118))
```

```
# Verifying with QQ plot for one of the season
from statsmodels.graphics.gofplots import qqplot
qqplot(season1_data ,  line="s")
plt.title("QQ plot for season 1 data")
plt.show()
```



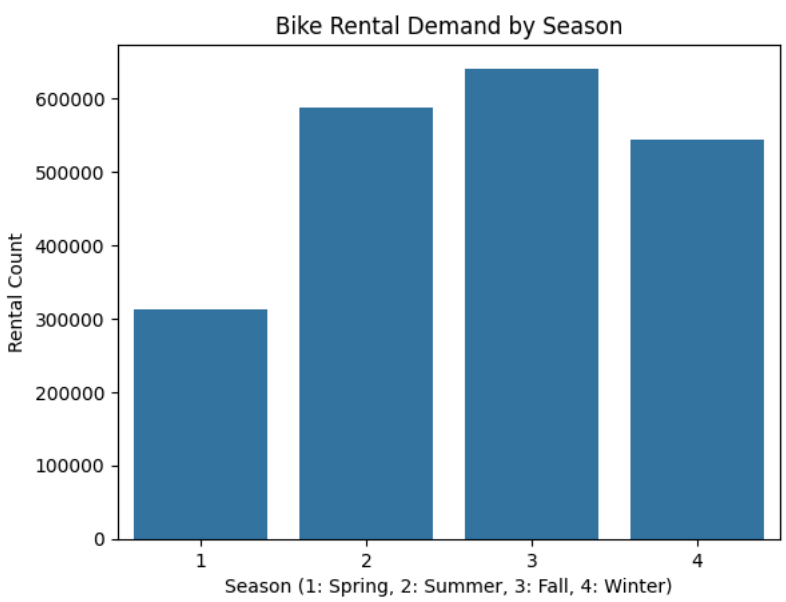
Our Conclusion: Though the Shapiro-Wilk and Levene's test indicate deviations from normality and homogeneity, ANOVA was still applied due to the robustness of the method in large samples

```
# Applying One Way Annova Test
test_statistic, p_value = f_oneway(season1_data,season2_data,season3_data,season4_data)
print(test_statistic, p_value)
```

```
236.94671081032106 6.164843386499654e-149
```

- Hypothesis:
  - H_0 (Null Hypothesis): The mean bike rental demand is the same across all seasons.
  - Ha (Alternative Hypothesis): The mean demand differs for at least one season.
- Since the p-value is far less than 0.05 (i.e 6.164843386499654e-149), we reject the null hypothesis.
- Conclusion: There is a statistically significant difference in average bike rental demand across different seasons.

```
season_data = df.groupby("season")["count"].sum().reset_index()
sns.barplot(x="season" ,y="count", data=season_data )
plt.title("Bike Rental Demand by Season")
plt.xlabel("Season (1: Spring, 2: Summer, 3: Fall, 4: Winter)")
plt.ylabel("Rental Count")
plt.show()
```



- The bar chart illustrates the total rental counts for each season, alongside the results from a One-Way ANOVA test.
  - Highest Demand in Fall (Season 3)
  - Summer (Season 2) also shows strong demand
  - Winter (Season 4) experiences moderate demand
  - Spring (Season 1) has the lowest rental volume

Business recommendations for Yulu

- Optimize Fleet Size Seasonally:Increase bike availability and charging logistics during Fall and Summer.Consider maintenance or cost-saving operations during Spring, when demand is low.
- Seasonal Promotions:Offer discounts or reward programs in Spring and Winter to stimulate demand.Leverage high seasons (Fall, Summer) for revenue maximization without heavy discounts.
- Targeted Marketing:Educate users about comfort and convenience of Yulu during winter to reduce seasonal drop-off.

Weather Conditions - Demand Relation

```
weather1_data=df[df["weather"]==1]["count"]
print("Shapiro test for  weather 1" , shapiro(weather1_data))

weather2_data=df[df["weather"]==2]["count"]
```

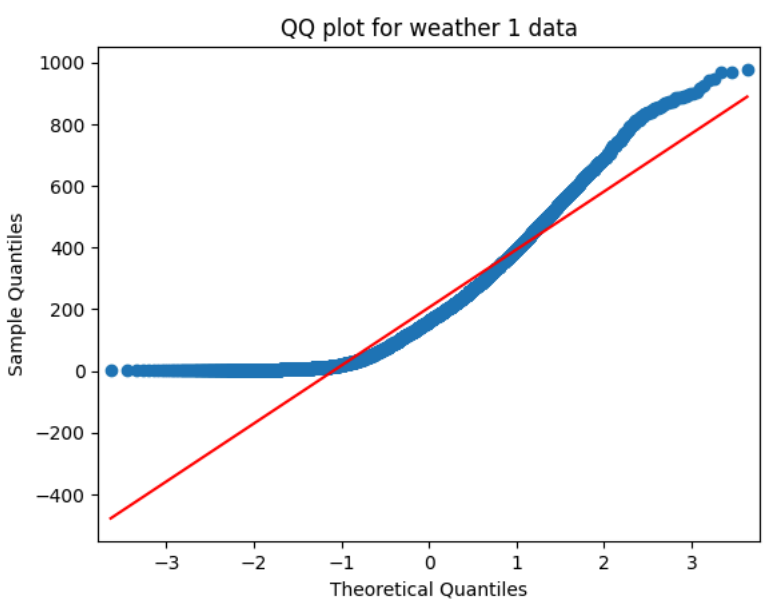
```
print("Shapiro test for weather 2" ,shapiro(weather2_data))

weather3_data=df[df["weather"]==3]["count"]
print("Shapiro test for  weather 3" ,shapiro(weather3_data))


# Shapiro not applicable as there is only one data in this group
weather4_data=df[df["weather"]==4]["count"]
print("Shapiro test for  weather 4" ,shapiro(weather4_data))
```

```
Shapiro test for  weather 1 ShapiroResult(statistic=np.float64(0.8909259459740138), pvalue=np.float64(1.5964921477006555e-57))
Shapiro test for weather 2 ShapiroResult(statistic=np.float64(0.8767694973495206), pvalue=np.float64(9.777839106111785e-43))
Shapiro test for  weather 3 ShapiroResult(statistic=np.float64(0.7674327906035717), pvalue=np.float64(3.875893017396149e-33))
Shapiro test for  weather 4 ShapiroResult(statistic=np.float64(nan), pvalue=np.float64(nan))
/usr/local/lib/python3.11/dist-packages/scipy/stats/_axis_nan_policy.py:586: UserWarning: scipy.stats.shapiro: For N > 5000, compute
  res = hypotest_fun_out(*samples, **kwds)
/tmp/ipython-input-255-4204849334.py:13: SmallSampleWarning: One or more sample arguments is too small; all returned values will be
  print("Shapiro test for  weather 4" ,shapiro(weather4_data))
```

```
levene(season1_data, season2_data , season3_data  , season4_data)
```

```
LeveneResult(statistic=np.float64(187.7706624026276), pvalue=np.float64(1.0147116860043298e-118))
```

```
# Verifying with QQ plot for one of the season
from statsmodels.graphics.gofplots import qqplot
qqplot(weather1_data ,  line="s")
plt.title("QQ plot for weather 1 data")
plt.show()
```



Our Conclusion: Though the Shapiro-Wilk and Levene's test indicate deviations from normality and homogeneity, ANOVA was still applied due to the robustness of the method in large samples

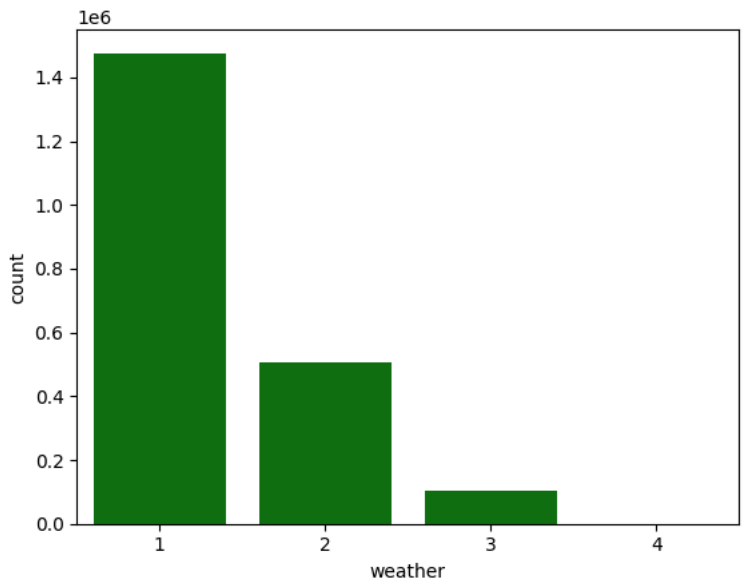
```
# Applying One Way Annova Test
test_statistic, p_value = f_oneway(weather1_data,weather2_data,weather3_data,weather4_data)
print(test_statistic, p_value)
```

```
65.53024112793271 5.482069475935669e-42
```

- Hypothesis:
  - H_0 (Null Hypothesis): The mean bike rental demand is the same across all weather.
  - Ha (Alternative Hypothesis): The mean demand differs for at least one weather.
- Since the p-value is far less than 0.05 (i.e 5.482069475935669e-42), we reject the null hypothesis.
- Conclusion: There is a statistically significant difference in average bike rental demand across different weather.

```
weather_data = df.groupby("weather")["count"].sum().reset_index()
plt.title("Bike Rental Demand by Weather")
```

```
sns.barplot(x = "weather" , y = "count" , data=weather_data , color="green")
plt.show()
```



Graphical Analysis

- Majority of Rides Occur in Good Weather (Code 1):
  - Over 1.4 million rides were recorded when weather was clear or partly cloudy.
  - Indicates strong user preference for renting in favorable conditions.
- Demand Drops with Worsening Weather:
  - Weather 2 (misty/cloudy) sees a sharp drop — approximately one-third of demand compared to clear weather.
  - Weather 3 (light rain/snow) leads to a dramatic fall in rentals — around 100k rides only.
  - Weather 4 (severe conditions) is almost unused — indicating negligible usage during extreme weather events.

Business Insights:

- Weather Sensitivity is High:Demand is clearly weather-dependent — even mild deterioration in weather causes significant drop-offs.This insight is critical for forecasting and real-time demand modeling.
- Operational Planning:During bad weather (codes 3 & 4), Yulu should Reduce fleet deployment , Shift maintenance schedules to low-demand days, Avoid aggressive marketing
- User Communication & Safety and building trust:In misty or rainy weather, consider in-app notifications to promote safety (e.g., ride carefully, reduce speed).Offer limited-time discounts during light rain if safety allows, to maintain baseline demand.
- Weather-Aware Demand Forecasting:Weather data should be included as a core feature in machine learning models to predict ride demand.

Season Types and Weather Conditions Relation

```
weather_season = pd.crosstab(index=df['season'],columns=df['weather'])
weather_season # This will give the count of each weather condition for each season type
```

season	1	2	3	4
1	1759	715	211	1
2	1801	708	224	0
3	1930	604	199	0
4	1702	807	225	0

Next steps: Generate code with weather_season | View recommended plots | New interactive sheet

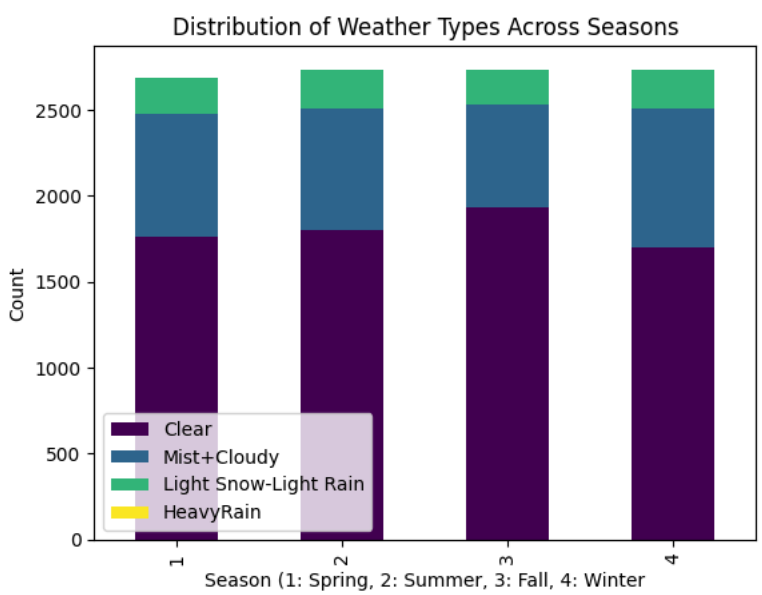
```
chi_stat, p_value, df, exp_freq = chi2_contingency(weather_season) # chi_stat, p_value, df, expected value

print("chi_stat:",chi_stat)
print("p_value:",p_value)
```

```
print("df:",df)
print("exp_freq:",exp_freq)
```

```
chi_stat: 49.158655596893624
p_value: 1.549925073686492e-07
df: 9
exp_freq: [[1.77454639e+03 6.99258130e+02 2.11948742e+02 2.46738931e-01]
 [1.80559765e+03 7.11493845e+02 2.15657450e+02 2.51056403e-01]
 [1.80559765e+03 7.11493845e+02 2.15657450e+02 2.51056403e-01]
 [1.80625831e+03 7.11754180e+02 2.15736359e+02 2.51148264e-01]]
```

- Hypothesis:
  - H_0 (Null Hypothesis): Weather conditions are NOT significantly different during different Seasons.
  - Ha (Alternative Hypothesis): Weather conditions are significantly different during different Seasons.
- Since the p-value is far less than 0.05 (i.e 1.549925073686492e-07), we reject the null hypothesis.
- Conclusion: Weather conditions are not independent of season — meaning the distribution of weather types significantly varies across different seasons.

```
weather_season.plot(kind='bar', stacked=True, colormap='viridis' )
plt.title("Distribution of Weather Types Across Seasons")
plt.xlabel("Season (1: Spring, 2: Summer, 3: Fall, 4: Winter")
plt.legend(["Clear" , "Mist+Cloudy" , "Light Snow-Light Rain"  , "HeavyRain"])
plt.ylabel("Count")
plt.show()
```



Graphical Analysis :

- The majority of days in every season experience clear or partly cloudy weather (Weather Type 1).Especially in Fall (Season 3), the share of clear weather is the highest — aligning with high rental demand observed earlier.
- Mist + Cloudy Conditions (Weather Type 2):Present consistently across all seasons, forming the second-largest weather category.More prevalent in Winter (Season 4) and Spring (Season 1).
- Light Snow / Light Rain (Weather Type 3):Appears only sparingly across all seasons.Usage likely drops during these days, as observed in rental behavior.
- Heavy Rain or Extreme Conditions (Weather Type 4):Almost negligible in all seasons — indicating these are rare events.Reinforces why rentals drop to nearly zero during such weather.

Business Insights for Yulu:

- Weather Distribution Varies by Season:Planning demand should consider seasonal weather patterns, not just season labels.For example, Winter may have more misty days, while Fall enjoys more clear skies.
- Optimize Operations Seasonally:Prepare for lower usage days during misty/cloudy seasons (especially Winter).Increase marketing and fleet availability during clear-weather-dominant seasons like Fall.
- Enhance Forecast-Driven Planning:Build a weather-season matrix to help predict real-time demand dips.Offer incentives on misty or mildly rainy days to retain usage.
- Data-Driven Communication:App notifications or alerts can be tuned seasonally to inform users about ideal riding days or offer rain-safe riding tips.

.