yulu

May 9, 2024

1 About

- 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.

2 Objective

2.1 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

2.2 Dataset Link:

Dataset : yulu_data.csv

3 Importing Libraries And Loading the Dataset.

[1]:
```
import numpy as np
import pandas as pd
from scipy import stats
import seaborn as sns
import plotly.graph_objs as go
import plotly.express as px
import datetime as dt
from scipy.stats import ttest_ind,levene,shapiro,f_oneway,chi2_contingency
from statsmodels.graphics.gofplots import qqplot
```

```
import matplotlib.pyplot as plt

df = pd.read_csv("https://d2beiqkhq929f0.cloudfront.net/public_assets/assets/
 ↪000/001/428/original/bike_sharing.csv?1642089089")
df
```

```
[1]:                   datetime  season  holiday  workingday  weather   temp  \
0      2011-01-01 00:00:00       1        0           0        1   9.84
1      2011-01-01 01:00:00       1        0           0        1   9.02
2      2011-01-01 02:00:00       1        0           0        1   9.02
3      2011-01-01 03:00:00       1        0           0        1   9.84
4      2011-01-01 04:00:00       1        0           0        1   9.84
…                      …     …       …          …        …     …
10881  2012-12-19 19:00:00       4        0           1        1  15.58
10882  2012-12-19 20:00:00       4        0           1        1  14.76
10883  2012-12-19 21:00:00       4        0           1        1  13.94
10884  2012-12-19 22:00:00       4        0           1        1  13.94
10885  2012-12-19 23:00:00       4        0           1        1  13.12

        atemp  humidity  windspeed  casual  registered  count
0      14.395        81     0.0000       3          13     16
1      13.635        80     0.0000       8          32     40
2      13.635        80     0.0000       5          27     32
3      14.395        75     0.0000       3          10     13
4      14.395        75     0.0000       0           1      1
…         …         …        …       …       …
10881  19.695        50    26.0027       7         329    336
10882  17.425        57    15.0013      10         231    241
10883  15.910        61    15.0013       4         164    168
10884  17.425        61     6.0032      12         117    129
10885  16.665        66     8.9981       4          84     88

[10886 rows x 12 columns]
```

4 Basic Analysis

```
[2]: df.shape
```

```
[2]: (10886, 12)
```

4.0.1 Analysis:

- Number of rows = 10886
- Number of columns = 12

```
[3]: df.columns
```

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

5 Data Description.

```
[4]: 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
```

- All Columns are Numeric except datetime column.
- There are no missing values in the dataframe.
- Categorical value like season,holiday,weather,etc. are int64 we will convert them to categorical value.
- Convert datetime from object type to datetime type.

```
[5]: df.describe().T
```

```
[5]:              count        mean         std   min      25%      50%       75%  \
     season     10886.0    2.506614    1.116174  1.00   2.0000    3.000    4.0000
     holiday    10886.0    0.028569    0.166599  0.00   0.0000    0.000    0.0000
     workingday 10886.0    0.680875    0.466159  0.00   0.0000    1.000    1.0000
     weather    10886.0    1.418427    0.633839  1.00   1.0000    1.000    2.0000
     temp       10886.0   20.230860    7.791590  0.82  13.9400   20.500   26.2400
     atemp      10886.0   23.655084    8.474601  0.76  16.6650   24.240   31.0600
     humidity   10886.0   61.886460   19.245033  0.00  47.0000   62.000   77.0000
     windspeed  10886.0   12.799395    8.164537  0.00   7.0015   12.998   16.9979
     casual     10886.0   36.021955   49.960477  0.00   4.0000   17.000   49.0000
     registered 10886.0  155.552177  151.039033  0.00  36.0000  118.000  222.0000
```

```
count       10886.0  191.574132  181.144454  1.00  42.0000  145.000  284.0000

                 max
season        4.0000
holiday       1.0000
workingday    1.0000
weather       4.0000
temp         41.0000
atemp        45.4550
humidity    100.0000
windspeed    56.9969
casual      367.0000
registered  886.0000
count       977.0000
```

6 Data Processing.

6.1 Converting 'datetime' to datetime.

[6]:
```
df['datetime']=pd.to_datetime(df['datetime'])
df['datetime']
```

[6]:
```
0       2011-01-01 00:00:00
1       2011-01-01 01:00:00
2       2011-01-01 02:00:00
3       2011-01-01 03:00:00
4       2011-01-01 04:00:00
                 …
10881   2012-12-19 19:00:00
10882   2012-12-19 20:00:00
10883   2012-12-19 21:00:00
10884   2012-12-19 22:00:00
10885   2012-12-19 23:00:00
Name: datetime, Length: 10886, dtype: datetime64[ns]
```

6.2 Converting season,weather,holiday and workingday columns into categorical

[7]:
```
def season_type(x):
    if x==1:
        return 'spring'
    elif x==2:
        return 'summer'
    elif x==3:
        return 'fall'
    else:
        return 'winter'

df['season']=df['season'].apply(lambda x:season_type(x))

df['season']= pd.Categorical(df['season'])
df['weather']=pd.Categorical(df['weather'])
df['holiday']=pd.Categorical(df['holiday'])
df['workingday']=pd.Categorical(df['workingday'])
```

6.3 Statistical Summary after data-type conversion

[8]:
```
df.describe().T
```

[8]:
```
              count                           mean                  min  \
datetime      10886  2011-12-27 05:56:22.399411968  2011-01-01 00:00:00
temp        10886.0                       20.23086                 0.82
atemp       10886.0                      23.655084                 0.76
humidity    10886.0                       61.88646                  0.0
windspeed   10886.0                      12.799395                  0.0
casual      10886.0                      36.021955                  0.0
registered  10886.0                     155.552177                  0.0
count       10886.0                     191.574132                  1.0

                            25%                  50%                  75%  \
datetime    2011-07-02 07:15:00  2012-01-01 20:30:00  2012-07-01 12:45:00
temp                      13.94                 20.5                26.24
atemp                    16.665                24.24                31.06
humidity                   47.0                 62.0                 77.0
windspeed                7.0015               12.998              16.9979
casual                      4.0                 17.0                 49.0
registered                 36.0                118.0                222.0
count                      42.0                145.0                284.0

                            max         std
datetime    2012-12-19 23:00:00         NaN
temp                       41.0     7.79159
atemp                    45.455    8.474601
humidity                  100.0   19.245033
windspeed               56.9969    8.164537
casual                    367.0   49.960477
registered                886.0  151.039033
count                     977.0  181.144454
```

7 Outlier Detection using the z-score method

- We can detect outliers in numeric column using the z-score.

- If the z score of a data point is more than 3, it indicates that the data point is quite different from the other data points. Such a data point can be an outlier.
- z score= (x-mean)/std.deviation.

[9]:
```
outliers={}
for col in df.select_dtypes(include=np.number):

    #finding z-score for each value in a column
    z_score= np.abs((df[col]-df[col].mean()))/df[col].std()

    # if the z score of a value is a grater than 3 than the value is outlier
    column_outliers=df[z_score > 3][col]

    outliers[col]=column_outliers

for col,outlier_values in outliers.items():
    print(f"Outliers for {col} column")
    print(outlier_values)
    print()
```

```
Outliers for temp column
Series([], Name: temp, dtype: float64)

Outliers for atemp column
Series([], Name: atemp, dtype: float64)

Outliers for humidity column
1091    0
1092    0
1093    0
1094    0
1095    0
1096    0
1097    0
1098    0
1099    0
1100    0
1101    0
1102    0
1103    0
1104    0
1105    0
1106    0
1107    0
1108    0
1109    0
1110    0
```

```
1111    0
1112    0
Name: humidity, dtype: int64

Outliers for windspeed column
265      39.0007
613      39.0007
750      43.9989
752      40.9973
753      40.9973
          …
9481     43.0006
9482     43.0006
9484     39.0007
9754     39.0007
10263    43.0006
Name: windspeed, Length: 67, dtype: float64

Outliers for casual column
1384     219
1385     240
1935     196
2127     195
2129     206
        …
10226    195
10227    262
10228    292
10229    304
10230    260
Name: casual, Length: 292, dtype: int64

Outliers for registered column
6611     623
6634     614
6635     638
6649     628
6658     642
        …
10702    670
10726    655
10750    623
10846    652
10870    665
Name: registered, Length: 235, dtype: int64

Outliers for count column
6658     782
```

```
6659     749
6683     746
6779     801
6849     757
        …
9935     834
9944     890
9945     788
10519    743
10534    759
Name: count, Length: 147, dtype: int64
```

7.1 Observations:

- There no outliers in ‘temp’ and ‘atemp’ column.
- Outliers are evident within the ‘humidity’ and ‘windspeed’ columns based on the observations.
- Outliers are noticeable in the counts of casual and registered users, though drawing definite conclusions necessitates analyzing their relationship with independent variables.

8 Univariate Analysis.

8.1 Distribution of Working Day.

[10]:
```
workingday_df=df.groupby(['workingday'],observed=False).
  ↪agg(number_of_cycles_rented=('count','sum')).reset_index()
workingday_df
```

[10]:
```
   workingday  number_of_cycles_rented
0           0                   654872
1           1                  1430604
```

[11]:
```
labels= workingday_df['workingday']
values= workingday_df['number_of_cycles_rented']

colors = ['#C0E0DE','#4F7CAC']

bar_chart = go.Figure(go.Bar(x=labels, y=values, marker_color=colors))
bar_chart
```

8.1.1 Conclusion

- On working days, 68.6% of cycles are rented, whereas on non-working days, 31.4% of cycles are rented.

8.2 Distribution of Season.

```
[12]: season_df=df.groupby(['season'],observed=False).
       ↪agg(number_of_cycles_rented=('count','sum')).reset_index()
      season_df
```

```
[12]:    season  number_of_cycles_rented
      0    fall                   640662
      1  spring                   312498
      2  summer                   588282
      3  winter                   544034
```

```
[13]: labels= season_df['season']
      values= season_df['number_of_cycles_rented']

      colors = ['#D4D2A5','#FCDEBE','#ddbea9','#ffc8dd']

      bar_chart = go.Figure(go.Bar(x=labels, y=values, marker_color=colors))
      bar_chart
```

8.2.1 Conclusion.

- During the fall season, approximately 30.7% of cycles are rented.
- In the summer season, around 28.2% of cycles are rented.
- The winter season records a rental rate of about 26.1% for cycles.
- The lowest rental rate, at just 15%, is observed in the spring season.

8.3 Distribution Of Weather.

```
[14]: weather_df=df.groupby(['weather'],observed=False).
       ↪agg(number_of_cycles_rented=('count','sum')).reset_index()
      weather_df
```

```
[14]:   weather  number_of_cycles_rented
      0       1                  1476063
      1       2                   507160
      2       3                   102089
      3       4                      164
```

```
[15]: labels= weather_df['weather']
      values= weather_df['number_of_cycles_rented']

      #create pie chart
      # Create pie chart
      colors =  ["#b9e769","#efea5a","#f1c453"]

      bar_chart = go.Figure(go.Bar(x=labels, y=values, marker_color=colors))
      bar_chart
```

8.3.1 Conclusion.

- Weather condition 1 experiences the highest rental rate, with approximately 70.8% of cycles rented.
- In weather condition 2, around 24.3% of cycles are rented.
- Weather condition 3 has a rental rate of approximately 4.9% for cycles.
- Weather condition 4 exhibits an exceptionally low rental rate, with only 0.00786% of cycles being rented.

8.4 Trends in Average Cycle rentals(Hourly).

[16]:
```
hour_df=df.groupby(df['datetime'].dt.hour).
 ↪agg(average_cycles_rented=('count','mean')).reset_index()
hour_df
```

[16]:
```
    datetime  average_cycles_rented
0          0              55.138462
1          1              33.859031
2          2              22.899554
3          3              11.757506
4          4               6.407240
5          5              19.767699
6          6              76.259341
7          7             213.116484
8          8             362.769231
9          9             221.780220
10        10             175.092308
11        11             210.674725
12        12             256.508772
13        13             257.787281
14        14             243.442982
15        15             254.298246
16        16             316.372807
17        17             468.765351
18        18             430.859649
19        19             315.278509
20        20             228.517544
21        21             173.370614
22        22             133.576754
23        23              89.508772
```

[17]:
```
fig = px.line(hour_df, x='datetime', y='average_cycles_rented', markers=True)
fig.update_xaxes(tickvals=list(range(25)))
fig.update_layout(title='Average cycles rented in hourly basis',
                  xaxis_title='Hours',
                  yaxis_title='Average cycles rented')
fig.show()
```

8.4.1 Conclusion.

- The highest average count of rental bikes is observed at 5 PM, closely followed by 6 PM and 8 AM. This indicates distinct peak hours during the day when cycling is most popular.
- Conversely, the lowest average count of rental bikes occurs at 4 AM, with 3 AM and 5 AM also showing low counts. These hours represent the early morning period with the least demand for cycling.
- Notably, there is an increasing trend in cycle rentals between 5 AM and 8 AM, suggesting a surge in demand during the early morning hours as people start their day.
- Additionally, there is a decreasing trend in cycle rentals from 5 PM to 11 PM, indicating a gradual decline in demand as the day progresses into the evening and nighttime.

8.5 Trends in Average Cycle rentals(Monthly).

[18]:
```
month_df=df.groupby(df['datetime'].dt.month).
 ↪agg(average_cycles_rented=('count','mean')).reset_index()
month_df
```

[18]:
```
    datetime  average_cycles_rented
0          1              90.366516
1          2             110.003330
2          3             148.169811
3          4             184.160616
4          5             219.459430
5          6             242.031798
6          7             235.325658
7          8             234.118421
8          9             233.805281
9         10             227.699232
10        11             193.677278
11        12             175.614035
```

[19]:
```
fig = px.line(month_df, x='datetime', y='average_cycles_rented', markers=True)
fig.update_xaxes(tickvals=list(range(1,13)),ticktext=['Jan', 'Feb', 'Mar',␣
 ↪'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec'])
fig.update_layout(title='Average cycles rented on monthly basis',
                  xaxis_title='Month',
                  yaxis_title='Average cycles rented')
fig.show()
```

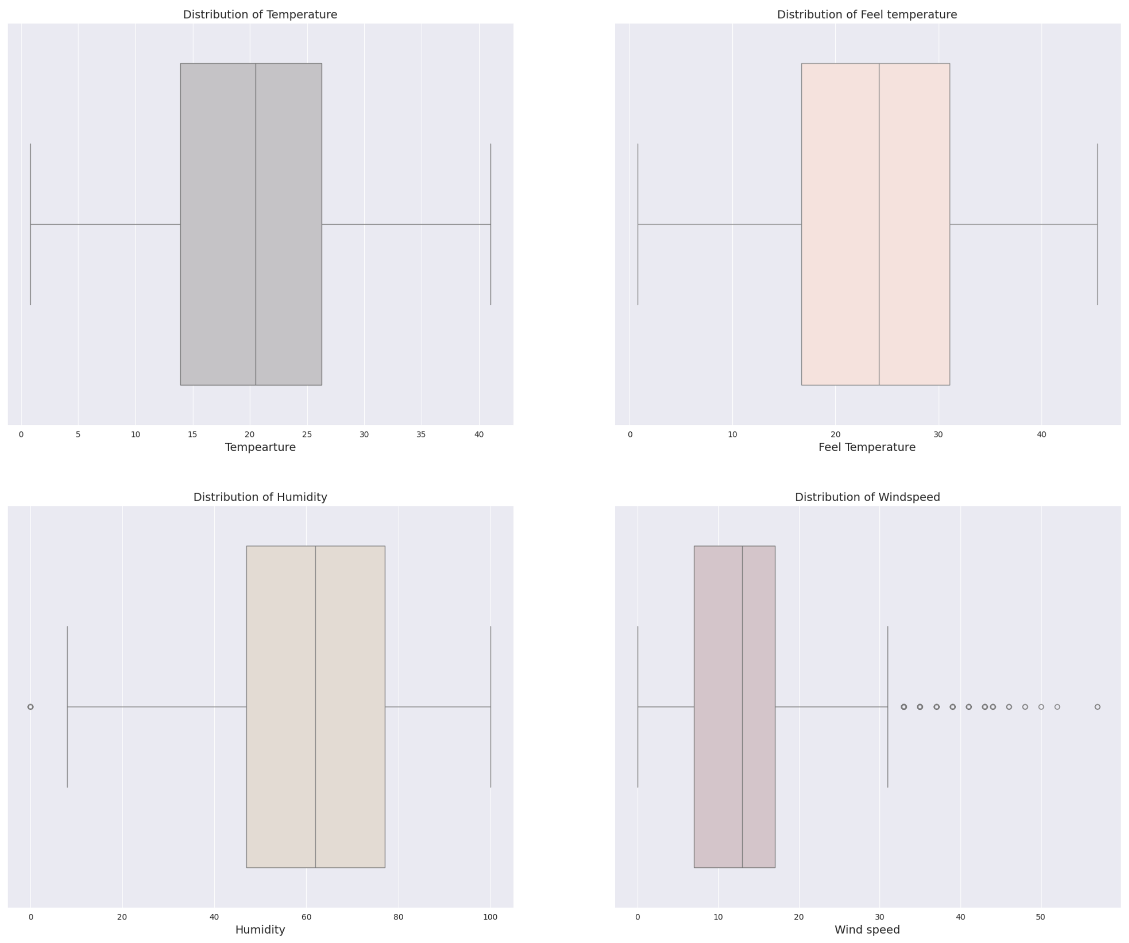
8.5.1 Conclusion.

- The highest average hourly count of rental bikes occurs in June, July, and August, reflecting the peak demand during summer.
- Conversely, the lowest average hourly count of rental bikes is found in January, February, and March, which are the winter months with reduced cycling activity.
- Notably, there is an increasing trend in average bike rentals from February to June, corresponding to the shift from winter to spring and summer.

- Conversely, a decreasing trend in average bike rentals is observed from October to December due to the onset of winter.

8.6 Distribution of temp, atemp, humidity and windspeed

[20]:
```python
sns.set_style('darkgrid')
plt.figure(figsize=(25,20))

# temp column
plt.subplot(2,2,1)
sns.boxplot(data=df,x='temp',color='#C5C3C6')
plt.xlabel('Tempearture',fontsize=14)
plt.title('Distribution of Temperature',fontsize=14)

#feel temperature
plt.subplot(2,2,2)
sns.boxplot(data=df,x='atemp',color='#F9E0D9')
plt.xlabel('Feel Temperature',fontsize=14)
plt.title('Distribution of Feel temperature',fontsize=14)

#Humidity
plt.subplot(2,2,3)

sns.boxplot(data=df,x='humidity',color='#E6DBD0')
plt.xlabel('Humidity',fontsize=14)
plt.title('Distribution of Humidity',fontsize=14)

#Wind Speed
plt.subplot(2,2,4)
sns.boxplot(data=df,x='windspeed',color='#D6C3C9')
plt.xlabel('Wind speed',fontsize=14)
plt.title('Distribution of Windspeed',fontsize=14)


plt.show()
```



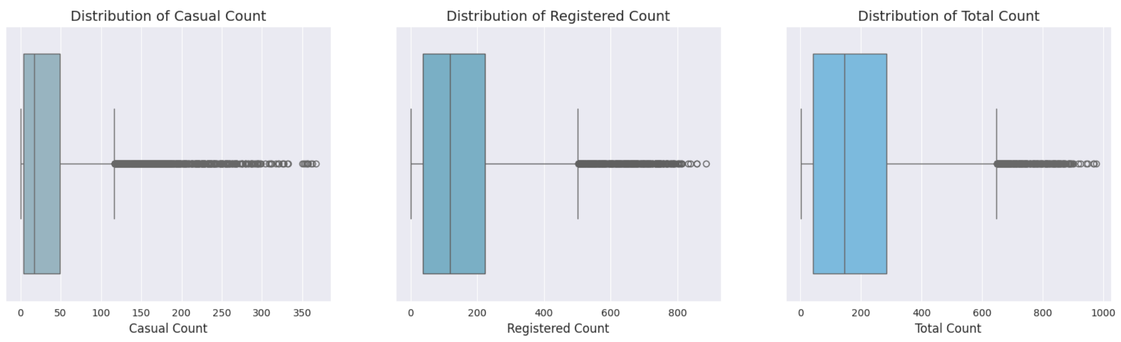
8.6.1 Conclusion.

- No outliers are detected in the 'temp' and 'atemp' columns, suggesting that the temperature-related data points fall within the expected range.
- In the 'humidity' column, a single value is identified as an outlier, implying an unusual humidity measurement distinct from the others.
- The 'windspeed' column contains 12 outlier values, indicating instances where wind speed measurements significantly deviate from the typical range.

8.7 Distribution of Casual count, Registered count and Total count

[21]:
```python
plt.figure(figsize=(20,5))

# Boxplot for temp column
plt.subplot(1,3,1)
sns.boxplot(data=df,x='casual',color='#91B7C7')
plt.xlabel('Casual Count',fontsize=12)
plt.title('Distribution of Casual Count',fontsize=14)
```

```
#Boxplot for feel temperature
plt.subplot(1,3,2)
sns.boxplot(data=df,x='registered',color='#6EB4D1')
plt.xlabel('Registered Count',fontsize=12)
plt.title('Distribution of Registered Count',fontsize=14)

#Boxplot for Humidity
plt.subplot(1,3,3)
sns.boxplot(data=df,x='count',color='#6CBEED')
plt.xlabel('Total Count',fontsize=12)
plt.title('Distribution of Total Count',fontsize=14)



plt.show()
```



8.7.1 Conclusion.

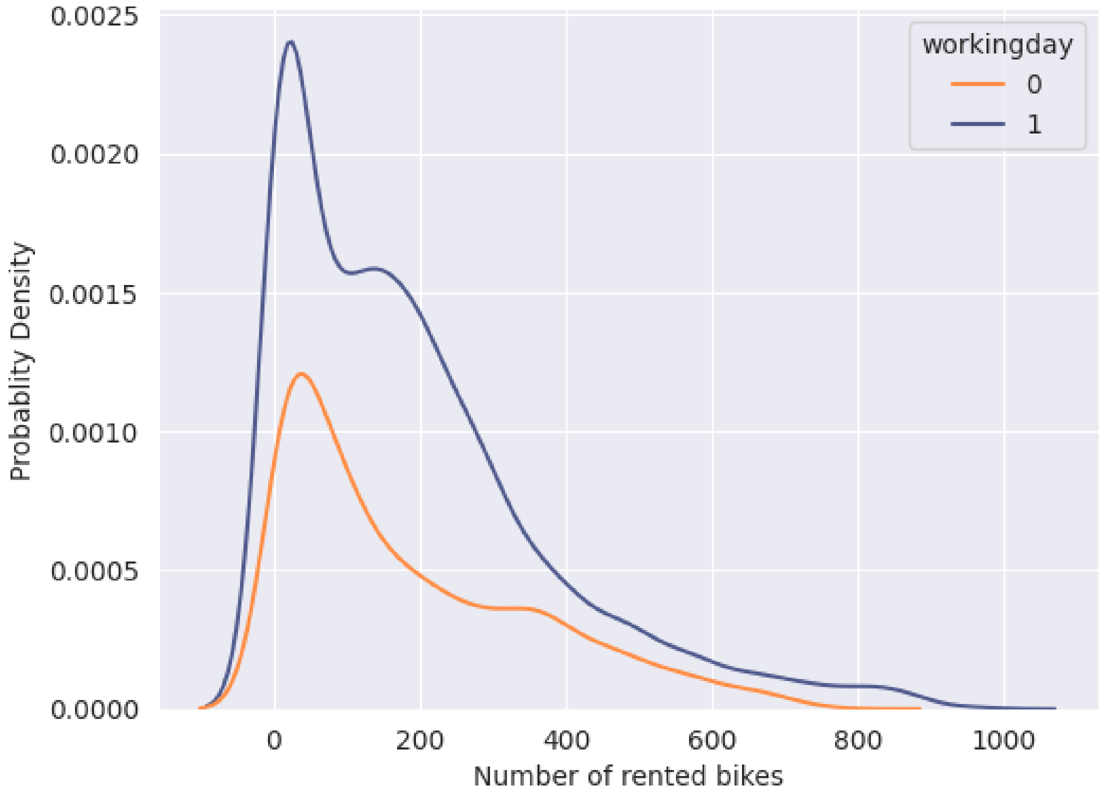
- The box plot clearly indicates the presence of outliers in the number of casual and registered users. However, further analysis against independent variables is needed before making definitive comments.
- The box plot reveal data skewness. As we proceed, we will decide whether to address outliers or perform variable transformation. In this case, given the significant number of outliers, variable transformation, specifically Log Transformation, seems to be a more appropriate approach.

9 Bivariate Analysis.

9.1 Distribution of count of rented bikes across working day

[22]:
```
sns.set_style('darkgrid')
sns.kdeplot(data=df,x='count',hue='workingday',palette=['#FF8C42','#4E598C'])
plt.xlabel('Number of rented bikes')
plt.ylabel('Probablity Density')
```

```
[22]: Text(0, 0.5, 'Probablity Density')
```



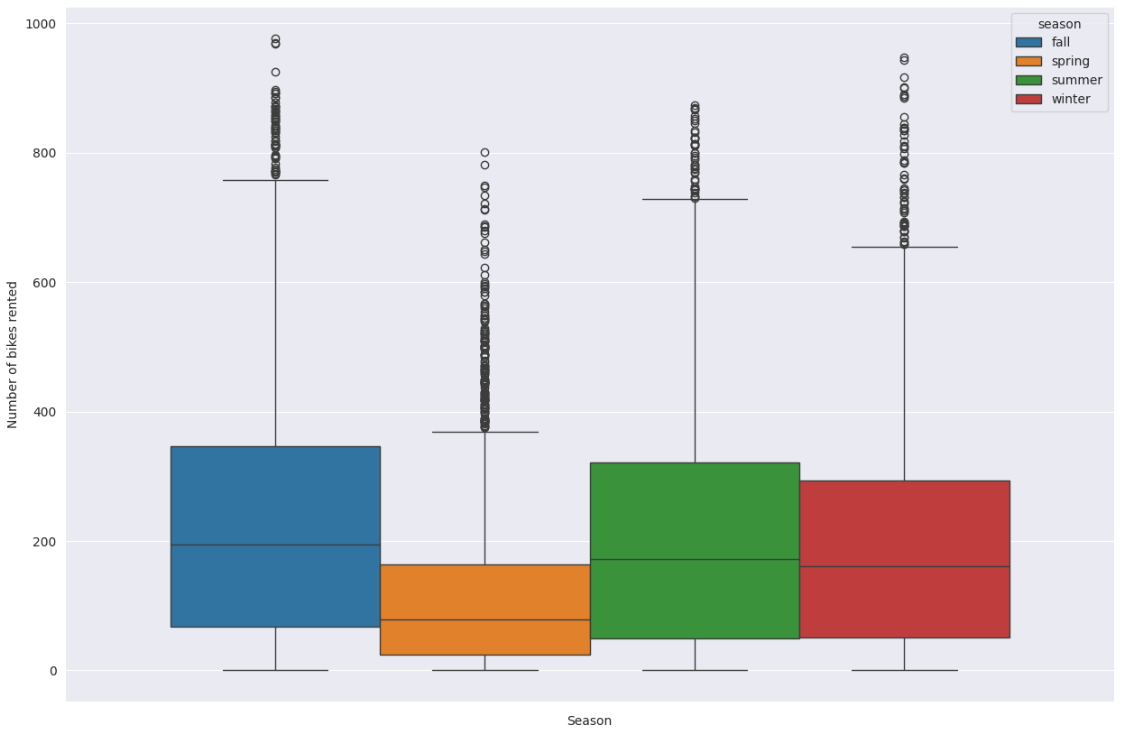
9.1.1 Conclusion.

The probability of renting bikes on a working day appears to be higher than on a non-working day, as evidenced by our univariate analysis, where 68.6% of bike rentals occurred on working days compared to 31.4% on non-working days. However, we will further investigate this through hypothesis testing to determine if the working day indeed has a statistically significant effect on the number of cycles rented."

9.2 Distribution of count of rented bikes across Season

```
[23]: plt.figure(figsize=(15,10))
      sns.boxplot(data=df,y='count',hue='season')
      plt.xlabel('Season')
      plt.ylabel('Number of bikes rented')
```

```
[23]: Text(0, 0.5, 'Number of bikes rented')
```



9.2.1 Conclusion.

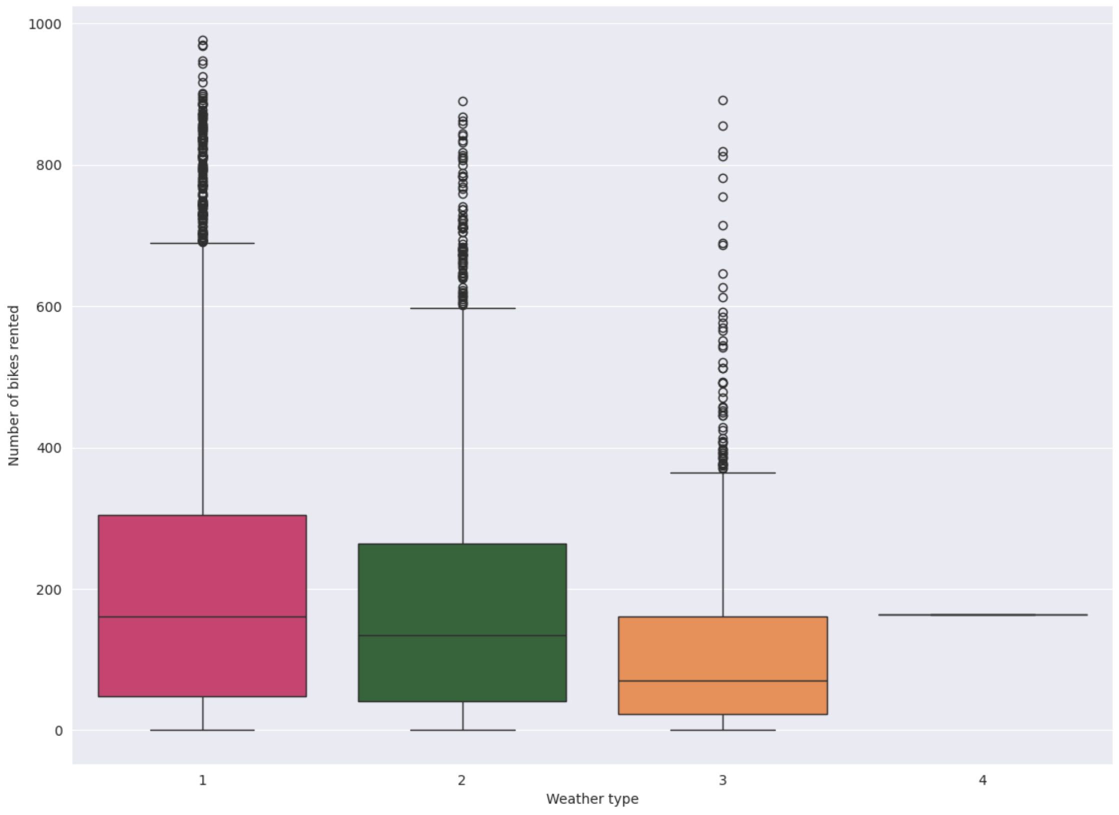
The probability of renting a bike during the fall season appears to be higher compared to other seasons. Conversely, the probability of renting bikes during the winter and spring seasons is lower in comparison to summer and fall.

9.3 Distribution of count of rented bikes across Weather types

[24]:
```
plt.figure(figsize=(30,10))

plt.subplot(1,2,2)
sns.
 ↪boxplot(data=df,y='count',x='weather',palette=['#DB3069','#306B34','#FF8C42','#586BA4'])
plt.xlabel('Weather type')
plt.ylabel('Number of bikes rented')
```

```
/tmp/ipykernel_7078/3995104985.py:4: FutureWarning:



Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same
effect.
```

```
[24]: Text(0, 0.5, 'Number of bikes rented')
```



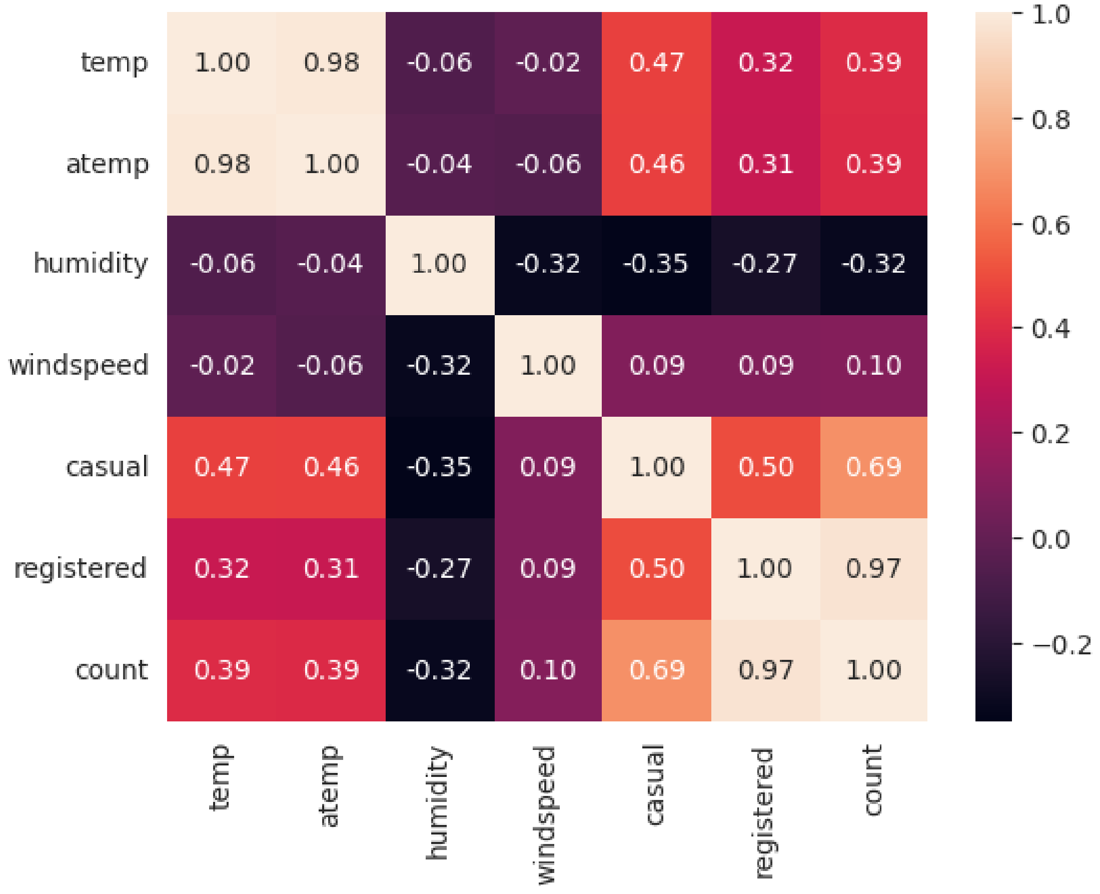
9.3.1 Conclusion.

The probability of renting a bike during weather condition 1 appears to be higher than in other weather types. This is supported by our univariate analysis, where approximately 70.8% of bike rentals occur in weather condition 1, while the remaining weather types collectively account for approximately 29% of bike rentals.

9.4 Heatmap and Correlation

```
[25]: sns.heatmap(df.corr(numeric_only=True),annot=True,fmt='.2f')
```

```
[25]: <Axes: >
```



9.5 Conclusion.

- The weak positive correlation of 0.39 between temperature and the number of bikes rented suggests that, on average, fewer people prefer to use electric cycles during the daytime between 12 PM to 3 PM. This observation aligns with our univariate analysis, where we discovered that the average number of cycles rented during this time frame was lower compared to other times of the day. A similar correlation pattern is also observed in the case of "feels-like" temperature, reinforcing this trend.
- The negative correlation between humidity and the number of cycles rented indicates that people tend to avoid using electric bikes during high humidity conditions.
- The presence of a weak positive correlation between windspeed and the number of cycles rented indicates that there is a subset of individuals who appear to favor using electric cycles during windy conditions for the sheer enjoyment of the experience. While this preference contributes to a slight increase in bike rentals on windier days, it's essential to recognize that this effect is not particularly strong, as indicated by the weak correlation.

10 Hypothesis Testing.

10.1 Effect of working Day on the number of electric cycles rented.

10.1.1 Formulating Null and Alternative Hypotheses

- H0 : Working day does not have an effect on number of cycles rented
- Ha: Working day does have an effect on number of cycles rented

Assumptions of a T Test

- Independence : The observations in one sample are independent of the observations in the other sample.
- Normality : Both samples are approximately normally distributed.
- Homogenity of Variances : Both samples have approximately the same variance.
- Random Sampling : Both samples were obtained using random sampling method

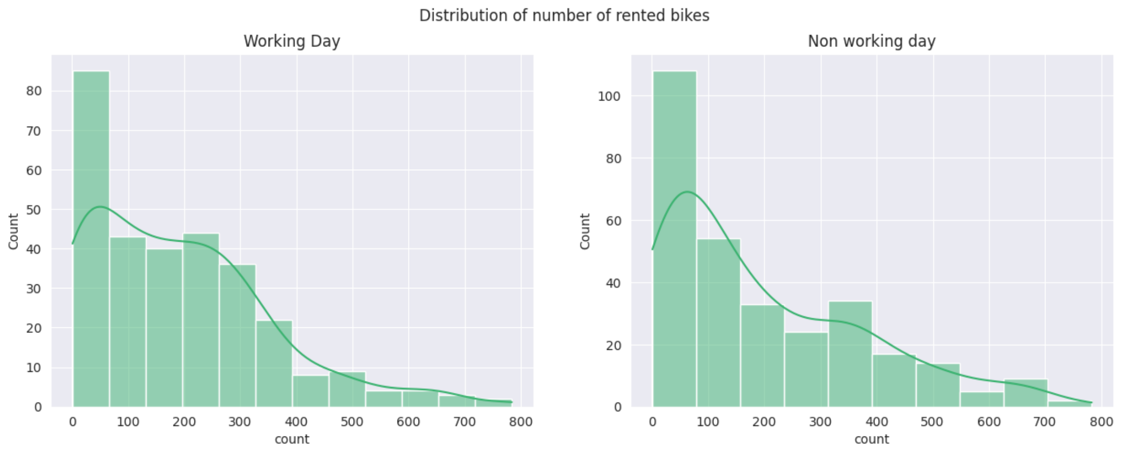
10.1.2 Normality Check: Wilkin Shapiron Test¶

Generate a sample of 300 bike rentals, randomly selected from both working days and non-working days

[26]:
```
workingday_sample=df[df['workingday']==1]['count'].sample(300)
nonworkingday_sample=df[df['workingday']==0]['count'].sample(300)
```

Checking normality using histogram.

[27]:
```
plt.figure(figsize=(15,5))

#histogram for working day sample
plt.subplot(1,2,1)
sns.histplot(workingday_sample,kde=True,color='mediumseagreen')
plt.title('Working Day')

#histogram for non working day sample
plt.subplot(1,2,2)
sns.histplot(nonworkingday_sample,kde=True,color='mediumseagreen')
plt.title('Non working day')

plt.suptitle('Distribution of number of rented bikes')
plt.show()
```

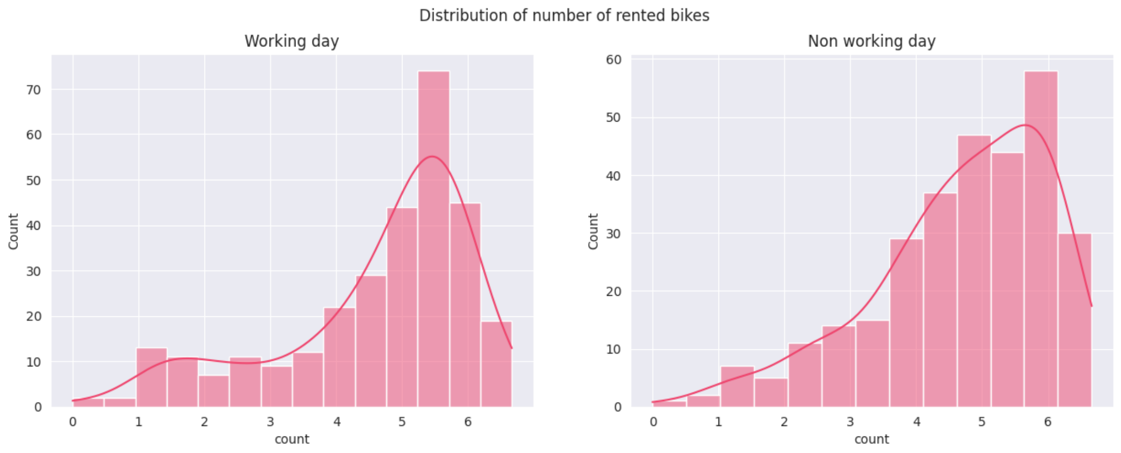


Conclusion.

- The counts of rented cycles on both working and non-working days do not follow a normal distribution.
- We can try to convert the distribution to normal by applying log transformation

Converting sample distribution to normal by applying log transformation

[28]:
```python
plt.figure(figsize=(15,5))

#histogram for working day sample
plt.subplot(1,2,1)
sns.histplot(np.log(workingday_sample),kde=True,color='#ef476f')
plt.title('Working day')

#histogram for non working day sample
plt.subplot(1,2,2)
sns.histplot(np.log(nonworkingday_sample),kde=True,color='#ef476f')
plt.title('Non working day')

plt.suptitle('Distribution of number of rented bikes')
plt.show()
```



Conclusion

- Upon implementing a log transformation on our continuous variables, we observed a substantial improvement in achieving a distribution that closely resembles normality for both the workingday_sample and nonworkingday_sample

Conducting the Wilk-Shapiro Test to assess the normality of the log-normal distribution obtained in the previous step

Performing the Wilk-Shapiro test for the workingday sample

We select the level of signifiance as 5% and the null and alternate hypothesis is as follows:

- H0 : The Working day samples are normally distributed
- Ha: The Working day samples are not normally distributed

[29]:
```
test_stat,p_value= shapiro(np.log(workingday_sample))
print("test stat :",test_stat)
print("p value :",p_value)
alpha = 0.05
if p_value< alpha:
 print("Reject Ho: The working day samples are not normally distributed ")
else:
 print("Fail to Reject Ho: The working day samples are normally distributed")
```

```
test stat : 0.8739916674957608
p value : 5.680348191904423e-15
Reject Ho: The working day samples are not normally distributed
```

Conclusion.

- From the above output, we see that the p value is far less than 0.05, Hence we reject the null hypothesis.
- We have sufficient evidence to say that the working day sample data does not come from normal distribution.

Performing the Wilk-Shapiro test for the non-working day sample

We select the level of significance as 5% and the null and alternate hypothesis is as follows:

- H0 : The non working day samples are normally distributed
- Ha: The non working day samples are not normally distributed

Conclusion.

- From the above output, we see that the p value is far less than 0.05, Hence we reject the null hypothesis.
- We have sufficient evidence to say that the **non working day sample** data does not come from normal distribution.

10.1.3 Homegenity of Variance test : Levene's Test

We select the level of significiance as 5% and the null and alternate hypothesis is as follows:

- H0 : Variance is equal in both working day count and non working day count samples
- Ha: Variances is not equal

[30]:
```
test_stat,p_value= levene(np.log(workingday_sample),np.
 ↪log(nonworkingday_sample))
print("test stat :",test_stat)
print("p value :",p_value)
alpha = 0.05
if p_value< alpha:
 print("Reject Ho: Variance is not equal ")
else:
 print("Fail to Reject Ho: Variance is equal in both working day count and non␣
 ↪working day count samples")
```

```
test stat : 0.1809231476964843
p value : 0.670734121817285
Fail to Reject Ho: Variance is equal in both working day count and non working
day count samples
```

Conclusion

- Since pvalue is not less than 0.05, we fail to reject null hypothesis.
- This means we do not have sufficient evidence to say that variance across workingday count and non workingday count is significantly different thus making the assumption of homogenity of variances true

10.1.4 T-Test and final conclusion

- 3 out of 4 assumptions for T test has been satified.
- Although the sample distribution did not meet the criteria of passing the normality test, we proceed with the T-test as per the given instructions.

For T-Test we select the level of significiance as 5% and the null and alternate hypothesis is as follows:

- H0 : Working day does not have an effect on number of cycles rented

- Ha: Working day does have an effect on number of cycles rented

[31]:
```
t_stat,p_value= ttest_ind(np.log(workingday_sample),np.
 ↪log(nonworkingday_sample),equal_var=True)
print("f stat :",t_stat)
print("p value :",p_value)
alpha = 0.05
if p_value< alpha:
 print("Reject Ho: Working day does have an effect on number of cycles rented ")
else:
 print("Fail to Reject Ho: Working day does not have an effect on number of␣
 ↪cycles rented")
```

```
f stat : -0.6595940855700191
p value : 0.5097682853901802
Fail to Reject Ho: Working day does not have an effect on number of cycles
rented
```

Conclusion

- Since the p-value of our test is greater than alpha which is 0.05, we fail to reject the null hypothesis of this test.
- we do not have sufficient evidence to conclude that working days have a significant effect on the number of cycles rented. This suggests that there is no significant difference in the number of cycles rented on working days versus non-working days.

10.2 Whether No. of cycles rented is similar or different in different weather.

To perform such an analyis, we use ANOVA test:

- ANOVA, which stands for Analysis of Variance, is a statistical technique used to assess whether there is a statistically significant difference among the means of two or more categorical groups. It achieves this by testing for variations in means by examining the variance within and between these groups.

- The 4 different weather conditions are as follows:

  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

We have to check if there is any significant difference in the number of bikes rented across different weather conditions. To analyse this, we use Annova test.

10.2.1 Formulatting Null and Alternate Hypothesis

[32]:
```
df['weather'].value_counts()
```

[32]:
```
weather
1    7192
2    2834
3     859
4       1
Name: count, dtype: int64
```

Conclusion

- Because the weather condition “Heavy Rain + Ice Pallets + Thunderstorm + Mist, Snow + Fog” has only happened once in our dataset, we don’t have enough information to decide if it really affects bike rentals. To prevent any skewed results, it’s best to remove this rare weather type from our analysis.

We shall setup the Null and Alternate Hypothesis to check if there is any effect of weather on the number of cycles rented.

- H0 : The mean number of cycles rented is the same across all three different weather types.
- Ha : There is at least one weather type with a mean number of cycles rented that significantly differs from the overall mean of the dependent variable.

Assumptions for ANOVA Test

- The distributions of data of each group should follow the Gaussian Distribution.
- The variance of each group should be the same or close to each other.
- The total n observations present should be independent of each other.

10.2.2 Normality Test: Shapiro-Wilk Test

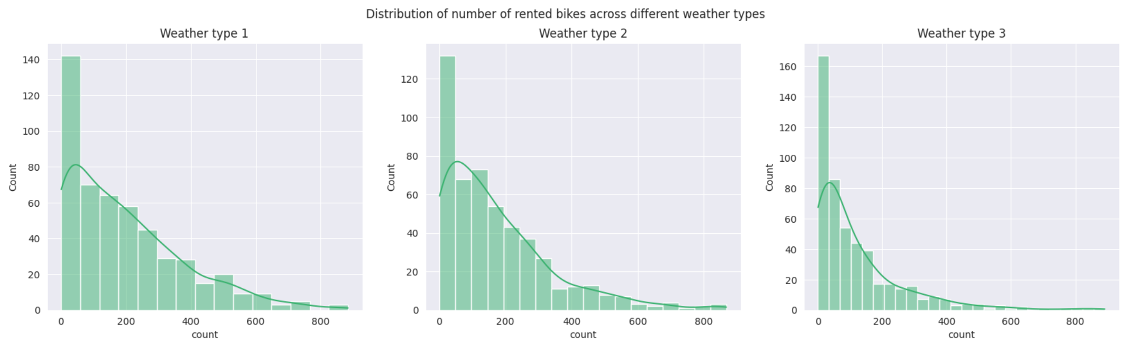
Generate a sample of 300 data points for each weather condition

[33]:
```python
sample_1= df[df['weather']==1]['count'].sample(500)
sample_2 = df[df['weather']==2]['count'].sample(500)
sample_3 = df[df['weather']==3]['count'].sample(500)
```

Checking normality using histogram.

[34]:
```python
plt.figure(figsize=(20,5))

#histogram for weather condition 1
plt.subplot(1,3,1)
sns.histplot(sample_1,kde=True,color='mediumseagreen')
plt.title('Weather type 1')

#histogram for weather condition 2
plt.subplot(1,3,2)
sns.histplot(sample_2,kde=True,color='mediumseagreen')
plt.title('Weather type 2')

#histogram for weather condition 3
plt.subplot(1,3,3)
```

```
sns.histplot(sample_3,kde=True,color='mediumseagreen')
plt.title('Weather type 3')

plt.suptitle('Distribution of number of rented bikes across different weather␣
 ↪types')
plt.show()
```

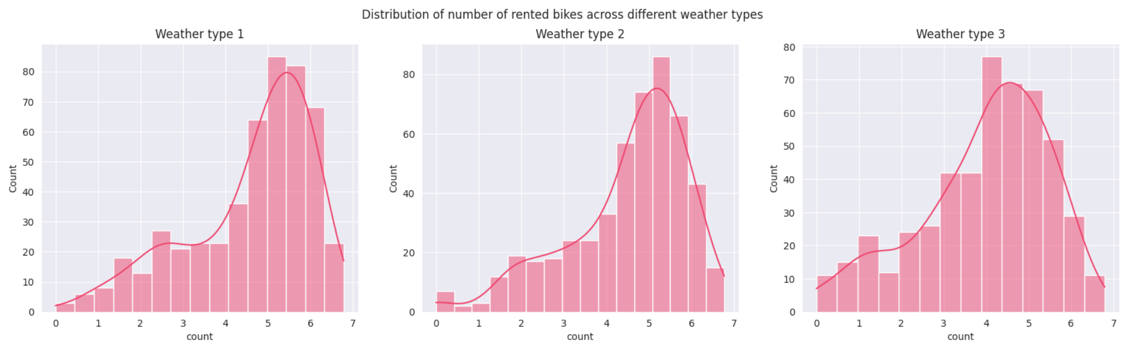


Conclusion

We see that none of the graphs are normally distributed. Hence we apply log transformation to make these distributions near to normal

Converting sample distribution to normal by applying log transformation

[35]:
```
log_1=np.log(sample_1)
log_2=np.log(sample_2)
log_3=np.log(sample_3)

plt.figure(figsize=(20,5))

#histogram for weather condition 1
plt.subplot(1,3,1)
sns.histplot(log_1,kde=True,color='#ef476f')
plt.title('Weather type 1')

#histogram for weather condition 2
plt.subplot(1,3,2)
sns.histplot(log_2,kde=True,color='#ef476f')
plt.title('Weather type 2')

#histogram for weather condition 3
plt.subplot(1,3,3)
sns.histplot(log_3,kde=True,color='#ef476f')
plt.title('Weather type 3')
plt.suptitle('Distribution of number of rented bikes across different weather␣
↪types')
plt.show()
```



Conclusion

After using a log transformation on the data for each weather type, we noticed a substantial improvement in making the data look more like a normal distribution.

Conducting the Shapiro-Wilk Test to assess the normality of the log-normal distribution obtained in the previous step

Shapiro-Wilk Test for weather type 1 sample data

We select the level of signifiance as 5% and the null and alternate hypothesis is as follows:

- H0 : The sample follows a normal distribution
- Ha: The sample does not follow a normal distribution

[36]:
```
test_stat,p_value= shapiro(log_1)
print("test stat :",test_stat)
print("p value :",p_value)
alpha = 0.05
if p_value< alpha:
 print("Reject Ho: The sample does not follow a normal distribution")
else:
 print("Fail to Reject Ho:The sample follows a normal distribution")
```

```
test stat : 0.9085718071930038
p value : 8.922092312235932e-17
Reject Ho: The sample does not follow a normal distribution
```

Conclusion

Even after applying the log transformation, the sample does not conform to a normal distribution, as demonstrated by the Shapiro-Wilk test.

Shapiro-Wilk Test for weather type 2 sample data

We select the level of signifiance as 5% and the null and alternate hypothesis is as follows:

- H0 : The sample follows a normal distribution
- Ha : The sample does not follow a normal distribution

[37]:
```
test_stat,p_value= shapiro(log_2)
print("test stat :",test_stat)
print("p value :",p_value)
alpha = 0.05
if p_value< alpha:
 print("Reject Ho: The sample does not follow a normal distribution")
else:
 print("Fail to Reject Ho:The sample follows a normal distribution")
```

```
test stat : 0.9228439072362071
p value : 2.4832328702298723e-15
Reject Ho: The sample does not follow a normal distribution
```

Conclusion

Even after applying the log transformation, the sample does not conform to a normal distribution, as demonstrated by the Shapiro-Wilk test.

Shapiro-Wilk Test for weather type 3 sample data

We select the level of signifiance as 5% and the null and alternate hypothesis is as follows:

- H0 : The sample follows a normal distribution
- Ha : The sample does not follow a normal distribution

[38]:
```
test_stat,p_value= shapiro(log_3)
print("test stat :",test_stat)
print("p value :",p_value)
alpha = 0.05
if p_value< alpha:
 print("Reject Ho: The sample does not follow a normal distribution")
else:
 print("Fail to Reject Ho:The sample follows a normal distribution")
```

```
test stat : 0.9519949068723
p value : 1.1383494769707192e-11
Reject Ho: The sample does not follow a normal distribution
```

Conclusion

Even after applying the log transformation, the sample does not conform to a normal distribution, as demonstrated by the Shapiro-Wilk test.

Final Conclusion:

None of the weather type samples adhere to a normal distribution even after applying the log-normal transformation, indicating that the normality assumption of the ANOVA test is not met.

10.2.3 Homegenity of Variance test : Levene's Test

We select the level of signifiance as 5% and the null and alternate hypothesis is as follows:

- H0 : The variance is equal across all groups
- Ha : The variance is not equal across the groups

[39]:
```
test_stat,p_value= levene(log_1,log_2,log_3,center='median')
print("test stat :",test_stat)
print("p value :",p_value)
alpha = 0.05
if p_value< alpha:
 print("Reject Ho: Variance is not equal across the groups ")
else:
 print("Fail to Reject Ho: Variance is equal across all groups")
```

```
test stat : 2.29712641991241
p value : 0.10090167959999721
Fail to Reject Ho: Variance is equal across all groups
```

Conclusion

- Since pvalue is not less than 0.05, we fail to reject the null hypothesis.
- This means we do not have sufficient evidence to claim a significant difference in variance across the different weather types. Therefore, the assumption of homogeneity of variances can be considered valid.

10.2.4 ANOVA Test and final Conclusion

For ANOVA Test we select the level of signifiance as 5% and the null and alternate hypothesis is as follows:

- H0 : The mean number of cycles rented is equal across different weather conditions.
- Ha: There is at least one weather condition with a mean number of cycles rented that significantly differs from the others.

[40]:
```
f_stat,p_value= f_oneway(log_1,log_2, log_3)
print("test stat :",f_stat)
print("p value :",p_value)
alpha = 0.05
if p_value< alpha:
 print("Reject Ho: There is at least one weather condition with a mean number␣
 ↪of cycles rented that significantly differs from the others ")
else:
 print("Fail to Reject Ho: The mean number of cycles rented is equal across␣
 ↪different weather conditions")
```

```
test stat : 26.805522452078737
p value : 3.648522627336841e-12
Reject Ho: There is at least one weather condition with a mean number of cycles
rented that significantly differs from the others
```

Final Conclusion:

- Since the p-value obtained from our test is less than the predetermined alpha level of 0.05, we have sufficient evidence to reject the null hypothesis for this test.
- Indeed, this indicates that we have collected sufficient evidence to conclude that there is a significant difference in the mean number of cycles rented across all weather conditions.
- Additionally, this suggests that weather conditions do have a notable effect on the number of cycles rented.

10.3 Whether No. of cycles rented is similar or different in different seasons.

Formulating Null and Alternative Hypotheses

We shall setup the Null and Alternate Hypothesis to check if there is any effect of season on the number of cycles rented.

- H0: All the 4 different seasons have equal means
- Ha: There is atleast one season that differs significantly from the overall mean of dependent variable.

Assumptions for ANOVA Test

- The distributions of data of each group should follow the Gaussian Distribution.
- The variance of each group should be the same or close to each other.
- The total n observations present should be independent of each other.

10.3.1 Normality Test: Wilkin Shapiro Test

Generate a sample of 300 data points for each season

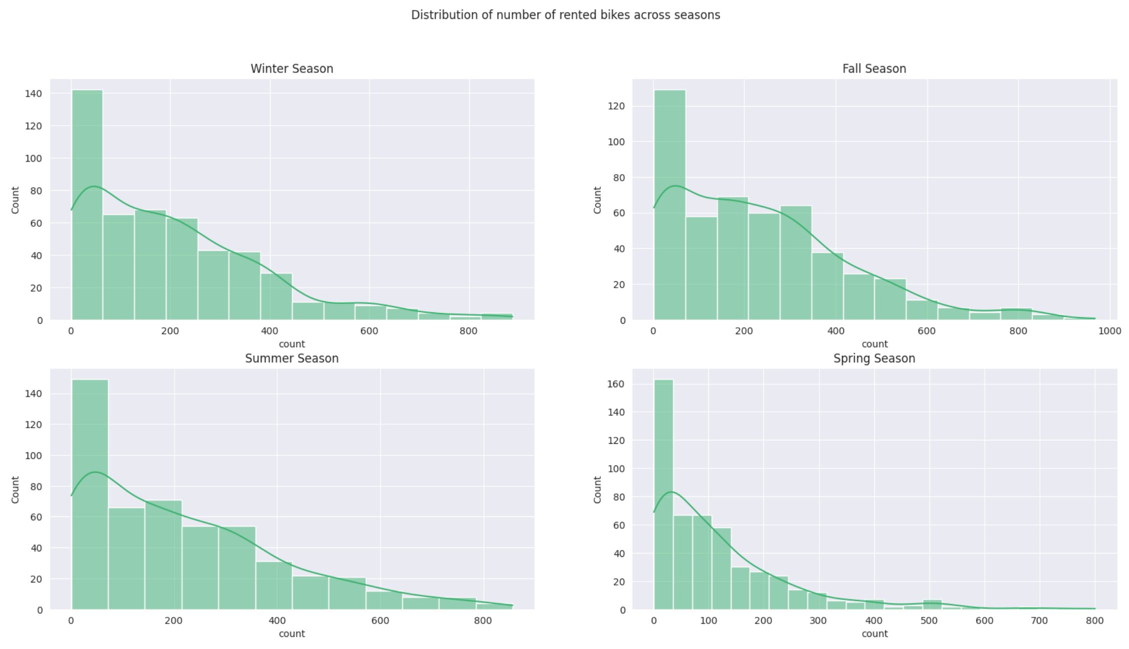
[41]:
```
winter_sample=df[df['season']=='winter']['count'].sample(500)
fall_sample = df[df['season']=='fall']['count'].sample(500)
summer_sample = df[df['season']=='summer']['count'].sample(500)
spring_sample = df[df['season']=='spring']['count'].sample(500)

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

[41]:
```
season
winter    2734
fall      2733
summer    2733
spring    2686
Name: count, dtype: int64
```

Checking normality using histogram

[42]:
```
plt.figure(figsize=(20,10))

#histogram for winter season
plt.subplot(2,2,1)
sns.histplot(winter_sample,kde=True,color='mediumseagreen')
```

```
plt.title('Winter Season')

#histogram for fall season
plt.subplot(2,2,2)
sns.histplot(fall_sample,kde=True,color='mediumseagreen')
plt.title('Fall Season')

#histogram for summer season
plt.subplot(2,2,3)
sns.histplot(summer_sample,kde=True,color='mediumseagreen')
plt.title('Summer Season')

#histogram for spring season
plt.subplot(2,2,4)
sns.histplot(spring_sample,kde=True,color='mediumseagreen')
plt.title('Spring Season')

plt.suptitle('Distribution of number of rented bikes across seasons')
plt.show()
```



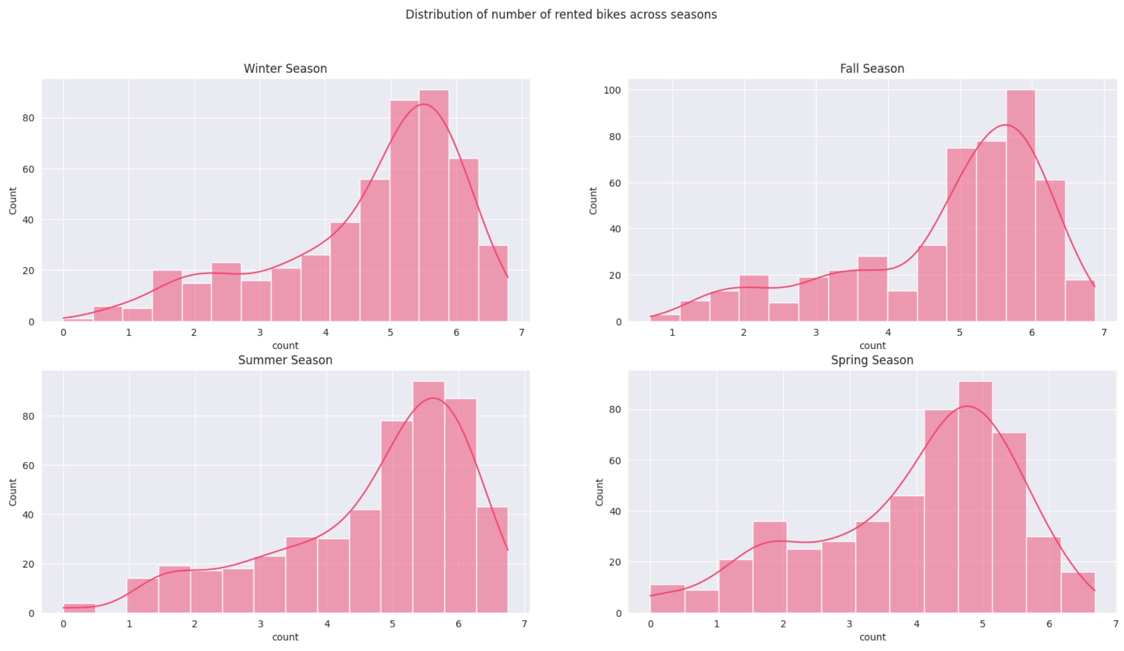
Conclusion

We see that none of the graphs are normally distributed. Hence we apply log transformation to make these distributions near to normal

[43]:
```
log_winter=np.log(winter_sample)
log_fall=np.log(fall_sample)
```

```
log_summer=np.log(summer_sample)
log_spring=np.log(spring_sample)
```

[44]:
```
plt.figure(figsize=(20,10))

#histogram for winter season
plt.subplot(2,2,1)
sns.histplot(log_winter,kde=True,color='#ef476f')
plt.title('Winter Season')

#histogram for fall season
plt.subplot(2,2,2)
sns.histplot(log_fall,kde=True,color='#ef476f')
plt.title('Fall Season')

#histogram for summer season
plt.subplot(2,2,3)
sns.histplot(log_summer,kde=True,color='#ef476f')
plt.title('Summer Season')

#histogram for spring season
plt.subplot(2,2,4)
sns.histplot(log_spring,kde=True,color='#ef476f')
plt.title('Spring Season')

plt.suptitle('Distribution of number of rented bikes across seasons')
plt.show()
```



Conclusion

After applying a log transformation to the samples of each season, it can be inferred that a significant improvement was observed in achieving data distributions that closely resemble normality for each of the seasons.

Conducting the Shapiro-Wilk Test to assess the normality of the log-normal distribution obtained in the previous step

Shapiro-Wilk Test for winter season sample data

We select the level of signifiance as 5% and the null and alternate hypothesis is as follows:

- H0 : The sample follows a normal distribution
- Ha: The sample does not follow a normal distribution

[45]:
```python
test_stat,p_value= shapiro(log_winter)
print("test stat :",test_stat)
print("p value :",p_value)
alpha = 0.05
if p_value< alpha:
 print("Reject Ho: The sample does not follow a normal distribution")
else:
 print("Fail to Reject Ho:The sample follows a normal distribution")
```

```
test stat : 0.9024218384194022
p value : 2.3849119897303933e-17
Reject Ho: The sample does not follow a normal distribution
```

Conclusion

Even after applying the log transformation, the sample does not conform to a normal distribution, as demonstrated by the Shapiro-Wilk test.

Shapiro-Wilk Test for fall season sample data

We select the level of signifiance as 5% and the null and alternate hypothesis is as follows:

- H0 : The sample follows a normal distribution
- Ha: The sample does not follow a normal distribution

[46]:
```python
test_stat,p_value= shapiro(log_fall)
print("test stat :",test_stat)
print("p value :",p_value)
alpha = 0.05
if p_value< alpha:
 print("Reject Ho: The sample does not follow a normal distribution")
else:
 print("Fail to Reject Ho:The sample follows a normal distribution")
```

```
test stat : 0.8872056827208757
p value : 1.159561135718569e-18
```

```
Reject Ho: The sample does not follow a normal distribution
```

Conclusion

Even after applying the log transformation, the sample does not follow a normal distribution, as demonstrated by the Shapiro-Wilk test.

Shapiro-Wilk Test for summer season sample data

We select the level of signifiance as 5% and the null and alternate hypothesis is as follows:

- H0 : The sample follows a normal distribution
- Ha: The sample does not follow a normal distribution

[47]:
```
test_stat,p_value= shapiro(log_summer)
print("test stat :",test_stat)
print("p value :",p_value)
alpha = 0.05
if p_value< alpha:
 print("Reject Ho: The sample does not follow a normal distribution")
else:
 print("Fail to Reject Ho:The sample follows a normal distribution")
```

```
test stat : 0.9021523219545213
p value : 2.254047730923185e-17
Reject Ho: The sample does not follow a normal distribution
```

Conclusion

Even after applying the log transformation, the sample does not follow a normal distribution, as demonstrated by the Shapiro-Wilk test.

Shapiro-Wilk Test for spring season sample data

We select the level of signifiance as 5% and the null and alternate hypothesis is as follows:

- H0 : The sample follows a normal distribution
- Ha: The sample does not follow a normal distribution

[48]:
```
test_stat,p_value= shapiro(log_spring)
print("test stat :",test_stat)
print("p value :",p_value)
alpha = 0.05
if p_value< alpha:
 print("Reject Ho: The sample does not follow a normal distribution")
else:
 print("Fail to Reject Ho:The sample follows a normal distribution")
```

```
test stat : 0.9417087188115643
p value : 4.2212182551141397e-13
Reject Ho: The sample does not follow a normal distribution
```

Conclusion

Even after applying the log transformation, the sample does not follow a normal distribution, as demonstrated by the Shapiro-Wilk test

10.3.2 Homegenity of Variance test : Levene's Test

We select the level of signifiance as 5% and the null and alternate hypothesis is as follows:

- H0 : The variance is equal across all groups
- Ha : The variance is not equal across the groups

[49]:
```
test_stat,p_value=␣
 ↪levene(log_winter,log_fall,log_summer,log_spring,center='median')
print("test stat :",test_stat)
print("p value :",p_value)
alpha = 0.05
if p_value< alpha:
 print("Reject Ho: Variance is not equal across the groups ")
else:
 print("Fail to Reject Ho: Variance is equal across all groups")
```

```
test stat : 2.294698180210158
p value : 0.07602509475155O9
Fail to Reject Ho: Variance is equal across all groups
```

Conclusion

- Since pvalue is not less than 0.05, we fail to reject the null hypothesis.
- This means we do not have sufficient evidence to claim a significant difference in variance across the different seasons. Therefore, the assumption of homogeneity of variances can be considered valid.

10.3.3 ANOVA Test and Final Conclusion

For ANOVA Test we select the level of signifiance as 5% and the null and alternate hypothesis is as follows:

- H0 : The mean number of cycles rented is the same across different seasons
- Ha: At least one season has a mean number of cycles rented that is significantly different from the others.

[50]:
```
f_stat,p_value= f_oneway(log_winter,log_fall, log_summer,log_spring)
print("f stat :",f_stat)
print("p value :",p_value)
alpha = 0.05
if p_value< alpha:
 print("Reject Ho: At least one season has a mean number of cycles rented that␣
 ↪is significantly different from the others. ")
else:
 print("Fail to Reject Ho: The mean number of cycles rented is the same across␣
 ↪different seasons ")
```

```
f stat : 37.11591198393442
p value : 2.448977043081957e-23
Reject Ho: At least one season has a mean number of cycles rented that is
significantly different from the others.
```

Conclusion:

- Since the p-value obtained from our test is less than the predetermined alpha level of 0.05, we have sufficient evidence to reject the null hypothesis for this test.
- Indeed, this implies that we have gathered enough evidence to conclude that there is a significant difference in the mean number of cycles rented across all seasons.

10.4 Is weather type dependent on the season

To perform such an analysis we perform Chi square test

A Chi-Square Test of Independence is used to determine whether or not there is a significant association between two categorical variables

[51]:
```
data=pd.crosstab(df['weather'],df['season'])
data
```

[51]:
```
season   fall  spring  summer  winter
weather
1        1930    1759    1801    1702
2         604     715     708     807
3         199     211     224     225
4           0       1       0       0
```

- there is only one row in our dataset for weather type 4. We lack sufficient information to determine if it truly correlates with the season. To avoid potential biases and skewed results, it's advisable to exclude this rare weather type from our analysis.

[54]:
```
df_removed_weather=df[~(df['weather']==4)]
data=pd.crosstab(df_removed_weather['weather'],df_removed_weather['season'])
data
```

[54]:
```
season   fall  spring  summer  winter
weather
1        1930    1759    1801    1702
2         604     715     708     807
3         199     211     224     225
```

[56]:
```
x_stat,p_value,dof,expected=chi2_contingency(data)
expected.min()
```

[56]:
```
211.88929719797886
```

[57]:
```
(len(expected[expected<5])/len(expected))*100
```

```
[57]: 0.0
```

Conclusion

- All of the data points have expected values greater than 5, indicating that the assumption related to the expected values being greater than 5 is satisfied for the chi-square test.

10.4.1 Chi-Square Test and Final Conclusion

We shall setup Null and alternate Hypotheis to check if Weather is dependent on season

- H0: Weather is not dependent on the season
- Ha: Weather is dependent on the season, meaning they are associated or related

We consider level of significance as 0.05

```
[58]: x_stat,p_value,dof,expected=chi2_contingency(data)
print("X stat :",x_stat)
print("p value :",p_value)
alpha = 0.05
if p_value< alpha:
 print("Reject Ho: Weather is dependent on the season")
else:
 print("Fail to Reject Ho: Weather is not dependent on the season")
```

```
X stat : 46.101457310732485
p value : 2.8260014509929403e-08
Reject Ho: Weather is dependent on the season
```

Final Conclusion

- Since the p-value obtained from our test is less than the predetermined alpha level of 0.05, we have sufficient evidence to reject the null hypothesis for this test.
- Indeed, this suggests that we have gathered enough evidence to conclude that there is a dependence between weather and the season.

11 Recommendations

- Yulu can introduce season-specific promotions and discounts to incentivize bike rentals during peak seasons. For example, offering discounts during summer to encourage more rides can attract additional customers.

- Ensure that bike availability is well-managed to meet the increased demand during peak seasons. This includes bike maintenance, distribution, and tracking to prevent shortages or excess bikes.

- During seasons with adverse weather conditions, such as rain or snow, Yulu can provide weather-ready bikes equipped with features like fenders and all-weather tires. This ensures that riders are comfortable and safe during inclement weather.

- During seasons with adverse weather conditions, such as rain or snow, Yulu can provide weather-ready bikes equipped with features like fenders and all-weather tires. This ensures that riders are comfortable and safe during inclement weather.
- During adverse weather, prioritize rider safety by providing guidelines and recommendations for riding in specific conditions. Ensure bikes are well-maintained and equipped with safety features.