YULU BUSINESS CASE

About Yulu

Yulu is India's leading micro-mobility service provider, which offers unique vehicles for the daily commute. Starting off as a mission to eliminate traffic congestion in India, Yulu provides the safest commute solution through a user-friendly mobile app to enable shared, solo and sustainable commuting.

Yulu zones are located at all the appropriate locations (including metro stations, bus stands, office spaces, residential areas, corporate offices, etc) to make those first and last miles smooth, affordable, and convenient!

Yulu has recently suffered considerable dips in its revenues. They have contracted a consulting company to understand the factors on which the demand for these shared electric cycles depends. Specifically, they want to understand the factors affecting the demand for these shared electric cycles in the Indian market.

Problem Statement

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?

About the dataset

1. datetime: datetime
2. season: season (1: spring, 2: summer, 3: fall, 4: winter)
3. holiday: whether day is a holiday or not (extracted from [http://dchr.dc.gov/page/holiday-schedule (http://dchr.dc.gov/page/holiday-schedule)](http://dchr.dc.gov/page/holiday-schedule))
4. workingday: if day is neither weekend nor holiday is 1, otherwise is 0.
5. weather: Clear, Few clouds, partly cloudy, partly cloudy ; Mist + Cloudy, Mist + Broken clouds, Mist + Few clouds, Mist ; Light Snow, Light Rain + Thunderstorm + Scattered clouds, Light Rain + Scattered clouds ; Heavy Rain + Ice Pallets + Thunderstorm + Mist, Snow + Fog
6. temp: temperature in Celsius
7. atemp: feeling temperature in Celsius
8. humidity: humidity
9. windspeed: wind speed
10. casual: count of casual users
11. registered: count of registered users
12. count: count of total rental bikes including both casual and registered

In [1]:
```
# Importing required packages to be used

import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from scipy.stats import norm,stats,chi2_contingency,levene,kruskal,shap
from IPython.display import display
```

In [2]:
```
# Importing and Reading top 10 data

df_yulu=pd.read_csv('https://d2beiqkhq929f0.cloudfront.net/public_asset
df_yulu.head(10)
```

Out[2]:

	datetime	season	holiday	workingday	weather	temp	atemp	humidity	windspeed	casual
0	2011-01-01 00:00:00	1	0	0	1	9.84	14.395	81	0.0000	3
1	2011-01-01 01:00:00	1	0	0	1	9.02	13.635	80	0.0000	8
2	2011-01-01 02:00:00	1	0	0	1	9.02	13.635	80	0.0000	5
3	2011-01-01 03:00:00	1	0	0	1	9.84	14.395	75	0.0000	3
4	2011-01-01 04:00:00	1	0	0	1	9.84	14.395	75	0.0000	0
5	2011-01-01 05:00:00	1	0	0	2	9.84	12.880	75	6.0032	0
6	2011-01-01 06:00:00	1	0	0	1	9.02	13.635	80	0.0000	2
7	2011-01-01 07:00:00	1	0	0	1	8.20	12.880	86	0.0000	1
8	2011-01-01 08:00:00	1	0	0	1	9.84	14.395	75	0.0000	1
9	2011-01-01 09:00:00	1	0	0	1	13.12	17.425	76	0.0000	8

General Analysis

In [3]:
```
# Defining the shape and dimension of data

a = df_yulu.shape
b = df_yulu.ndim

print("Shape of dataset -->", a)
print("Dimension of dataset -->",b)
```

```
Shape of dataset --> (10886, 12)
Dimension of dataset --> 2
```

Remarks: -

1. There are 10886 rows and 12 columns present in the dataset.
2. Its 2-d dataset by nature

In [4]:
```
# Checking general info of cols

df_yulu.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
```

Remarks: -

1. There is 1 string type column (datetime)
2. Rest all are integer and float type

In [5]:
```
# Overall stats of entire dataset

df_yulu.describe(include="all")
```

Out[5]:

	datetime	season	holiday	workingday	weather	temp	
count	10886	10886.000000	10886.000000	10886.000000	10886.000000	10886.00000	10886.(
unique	10886	NaN	NaN	NaN	NaN	NaN	
top	2011-01-01 00:00:00	NaN	NaN	NaN	NaN	NaN	
freq	1	NaN	NaN	NaN	NaN	NaN	
mean	NaN	2.506614	0.028569	0.680875	1.418427	20.23086	23.(
std	NaN	1.116174	0.166599	0.466159	0.633839	7.79159	8..
min	NaN	1.000000	0.000000	0.000000	1.000000	0.82000	0.
25%	NaN	2.000000	0.000000	0.000000	1.000000	13.94000	16.(
50%	NaN	3.000000	0.000000	1.000000	1.000000	20.50000	24.:
75%	NaN	4.000000	0.000000	1.000000	2.000000	26.24000	31.(
max	NaN	4.000000	1.000000	1.000000	4.000000	41.00000	45..

In [6]:
```
# Categorical Data Description

df_yulu.describe(include= object).T
```

Out[6]:

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

In [7]:
```
# Check on null/empty values

df_yulu.isna().sum().sort_values(ascending=False)
```

Out[7]:
```
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
```

Remarks: -

1. This is a check on the number of nulls/empty values present in the entire dataset
2. There are no nulls/empty values present in the dataset for any of the column values

In [8]:
```
# Finding duplicate rows based on all columns

duplicate_rows = df_yulu[df_yulu.duplicated()]
# Display the duplicate rows
print("Duplicate Rows:")
print(duplicate_rows)

# Count the total number of duplicate rows
num_duplicates = len(duplicate_rows)
print(f"Total number of duplicate rows: {num_duplicates}")
```

```
Duplicate Rows:
Empty DataFrame
Columns: [datetime, season, holiday, workingday, weather, temp, atem
p, humidity, windspeed, casual, registered, count]
Index: []
Total number of duplicate rows: 0
```

Remarks: -

1. This is solely targetting the number of duplicates across any of the data
2. There are no duplicates present in the entire dataset

In [9]:
```
# Checking for value ranges (e.g., numeric columns should not have nega
numeric_columns = df_yulu.select_dtypes(include=['int', 'float']).colur
value_range_issues = (df_yulu[numeric_columns] < 0).any()
print("\nValue Range Issues:")
print(value_range_issues)
```

```
Value Range Issues:
season        False
holiday       False
workingday    False
weather       False
temp          False
atemp         False
humidity      False
windspeed     False
casual        False
registered    False
count         False
dtype: bool
```

Remarks: -

1. This is for checking if there is any oddity or unexpected data/anomalies present under the integer columns
2. There are no anomaly values and only numeric data is present across all the rows for these integer columns

In [10]:
```
df_yulu.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
```

Non-Graphical Analysis

Uniques and Value counts

In [11]:
```
# Unique values for datetime
print("Total Unique values are: -")
print(df_yulu['datetime'].nunique())
print("Unique values are: -")
print(df_yulu['datetime'].unique())
# Value counts
print("\nValue counts are: -")
print(df_yulu['datetime'].value_counts().reset_index().to_string(index=
```

```
Total Unique values are: -
10886
Unique values are: -
['2011-01-01 00:00:00' '2011-01-01 01:00:00' '2011-01-01 02:00:00'
...
 '2012-12-19 21:00:00' '2012-12-19 22:00:00' '2012-12-19 23:00:00']

Value counts are: -
              index  datetime
2011-01-01 00:00:00         1
2012-05-01 21:00:00         1
2012-05-01 13:00:00         1
2012-05-01 14:00:00         1
2012-05-01 15:00:00         1
2012-05-01 16:00:00         1
2012-05-01 17:00:00         1
2012-05-01 18:00:00         1
2012-05-01 19:00:00         1
2012-05-01 20:00:00         1
2012-05-01 22:00:00         1
```

Remarks: -

1. There are 10886 unique values for product type in the entire dataset
2. The unique counts of each values are as displayed above:- 2011-01-01 00:00:00 = 1 time it occurred in the dataset ,etc.

In [12]:
```
# Unique values for season
print("Total Unique values are: -")
print(df_yulu['season'].nunique())
print("Unique values are: -")
print(df_yulu['season'].unique())
# Value counts
print("\nValue counts are: -")
print(df_yulu['season'].value_counts().reset_index().to_string(index=Fa
```

```
Total Unique values are: -
4
Unique values are: -
[1 2 3 4]

Value counts are: -
 index  season
     4    2734
     2    2733
     3    2733
     1    2686
```

Remarks: -

There are 4 unique values for product type in the entire dataset The unique counts of each values are as displayed above:- 4 = 2734 time it occurred in the dataset ,etc.

In [13]:
```
# Unique values for holiday
print("Total Unique values are: -")
print(df_yulu['holiday'].nunique())
print("Unique values are: -")
print(df_yulu['holiday'].unique())
# Value counts
print("\nValue counts are: -")
print(df_yulu['holiday'].value_counts().reset_index().to_string(index=
```

```
Total Unique values are: -
2
Unique values are: -
[0 1]

Value counts are: -
 index  holiday
     0    10575
     1      311
```

Remarks: -

There are 2 unique values for product type in the entire dataset The unique counts of each values are as displayed above:- 0 = 10575 time it occurred in the dataset ,etc.

In [14]:
```
# Unique values for workingday
print("Total Unique values are: -")
print(df_yulu['workingday'].nunique())
print("Unique values are: -")
print(df_yulu['workingday'].unique())
# Value counts
print("\nValue counts are: -")
print(df_yulu['workingday'].value_counts().reset_index().to_string(inde
```

```
Total Unique values are: -
2
Unique values are: -
[0 1]

Value counts are: -
 index  workingday
     1        7412
     0        3474
```

Remarks: -

There are 2 unique values for product type in the entire dataset The unique counts of each values are as displayed above:- 1 = 7412 time it occurred in the dataset ,etc.

In [15]:
```
# Unique values for weather
print("Total Unique values are: -")
print(df_yulu['weather'].nunique())
print("Unique values are: -")
print(df_yulu['weather'].unique())
# Value counts
print("\nValue counts are: -")
print(df_yulu['weather'].value_counts().reset_index().to_string(index=
```

```
Total Unique values are: -
4
Unique values are: -
[1 2 3 4]

Value counts are: -
 index  weather
     1     7192
     2     2834
     3      859
     4        1
```

Remarks: -

There are 4 unique values for product type in the entire dataset The unique counts of each values are as displayed above:- 1 = 7192 time it occurred in the dataset ,etc.

In [16]:
```
# Unique values for temp
print("Total Unique values are: -")
print(df_yulu['temp'].nunique())
print("Unique values are: -")
print(df_yulu['temp'].unique())
# Value counts
print("\nValue counts are: -")
print(df_yulu['temp'].value_counts().reset_index().to_string(index=Fals
```

```
Total Unique values are: -
49
Unique values are: -
[ 9.84  9.02  8.2  13.12 15.58 14.76 17.22 18.86 18.04 16.4  13.94 1
2.3
 10.66  6.56  5.74  7.38  4.92 11.48  4.1   3.28  2.46 21.32 22.96 2
3.78
 24.6  19.68 22.14 20.5  27.06 26.24 25.42 27.88 28.7  30.34 31.16 2
9.52
 33.62 35.26 36.9  32.8  31.98 34.44 36.08 37.72 38.54  1.64  0.82 3
9.36
 41.  ]

Value counts are: -
 index  temp
 14.76   467
 26.24   453
 28.70   427
 13.94   413
 18.86   406
 22.14   403
 25.42   403
 16.40   400
 22.96   395
 27.06   394
 24.60   390
 12.30   385
 21.32   362
 17.22   356
 13.12   356
 29.52   353
 10.66   332
 18.04   328
 20.50   327
 30.34   299
  9.84   294
 15.58   255
  9.02   248
 31.16   242
  8.20   229
 27.88   224
 23.78   203
 32.80   202
 11.48   181
 19.68   170
  6.56   146
 33.62   130
  5.74   107
  7.38   106
 31.98    98
 34.44    80
 35.26    76
  4.92    60
 36.90    46
  4.10    44
 37.72    34
 36.08    23
  3.28    11
  0.82     7
 38.54     7
 39.36     6
```

```
  2.46     5
  1.64     2
 41.00     1
```

Remarks: -

There are 49 unique values for product type in the entire dataset The unique counts of each values are as displayed above:- 14.76 = 467 times it occurred in the dataset ,etc.

In [17]:
```
# Unique values for temp
print("Total Unique values are: -")
print(df_yulu['atemp'].nunique())
print("Unique values are: -")
print(df_yulu['atemp'].unique())
# Value counts
print("\nValue counts are: -")
print(df_yulu['atemp'].value_counts().reset_index().to_string(index=Fal
```

```
Total Unique values are: -
60
Unique values are: -
[14.395 13.635 12.88  17.425 19.695 16.665 21.21  22.725 21.97  20.45
5
 11.365 10.605  9.85   8.335  6.82   5.305  6.06   9.09  12.12   7.57
5
 15.91   3.03   3.79   4.545 15.15  18.18  25.    26.515 27.275 29.54
5
 23.485 25.76  31.06  30.305 24.24  18.94  31.82  32.575 33.335 28.79
 34.85  35.605 37.12  40.15  41.665 40.91  39.395 34.09  28.03  36.36
5
 37.88  42.425 43.94  38.635  1.515  0.76   2.275 43.18  44.695 45.45
5]

Value counts are: -
 index  atemp
31.060    671
25.760    423
22.725    406
20.455    400
26.515    395
16.665    381
25.000    365
33.335    364
21.210    356
30.305    350
15.150    338
21.970    328
24.240    327
17.425    314
31.820    299
34.850    283
27.275    282
32.575    272
11.365    271
14.395    269
29.545    257
19.695    255
15.910    254
12.880    247
13.635    237
34.090    224
12.120    195
28.790    175
23.485    170
10.605    166
35.605    159
 9.850    127
18.180    123
36.365    123
37.120    118
 9.090    107
37.880     97
28.030     80
 7.575     75
38.635     74
 6.060     73
39.395     67
 6.820     63
 8.335     63
```

```
18.940     45
40.150     45
40.910     39
 5.305     25
42.425     24
41.665     23
 3.790     16
 4.545     11
 3.030      7
43.940      7
 2.275      7
43.180      7
44.695      3
 0.760      2
 1.515      1
45.455      1
```

Remarks: -

There are 60 unique values for product type in the entire dataset The unique counts of each values are as displayed above:- 31.060 = 671 times it occurred in the dataset ,etc.

In [18]:
```
# Unique values for humidity
print("Total Unique values are: -")
print(df_yulu['humidity'].nunique())
print("Unique values are: -")
print(df_yulu['humidity'].unique())
# Value counts
print("\nValue counts are: -")
print(df_yulu['humidity'].value_counts().reset_index().to_string(index=
```

```
Total Unique values are: -
89
Unique values are: -
[ 81  80  75  86  76  77  72  82  88  87  94 100  71  66  57  46  42
39
  44  47  50  43  40  35  30  32  64  69  55  59  63  68  74  51  56
52
  49  48  37  33  28  38  36  93  29  53  34  54  41  45  92  62  58
61
  60  65  70  27  25  26  31  73  21  24  23  22  19  15  67  10   8
12
  14  13  17  16  18  20  85   0  83  84  78  79  89  97  90  96  91]

Value counts are: -
 index  humidity
    88       368
    94       324
    83       316
    87       289
    70       259
    65       253
    46       247
    66       246
    77       244
    49       234
    55       224
    52       218
    56       208
    69       207
    93       205
    61       205
    62       202
    82       200
    74       197
    73       195
    43       193
    78       192
    50       190
    53       186
    41       184
    59       178
    81       174
    58       168
    40       167
    54       164
    60       162
    51       161
    44       151
    89       150
    37       149
    79       149
   100       148
    47       146
    57       145
    76       144
    45       143
    72       136
    42       133
    68       131
    36       129
    48       129
```

```
64      128
38      127
39      126
75      113
67      110
71      107
35      107
33      104
63      104
34       93
31       80
84       75
29       65
32       64
28       61
30       60
80       60
27       49
86       40
26       39
23       37
24       37
25       32
 0       22
22       18
21       16
19       15
20       10
16        8
18        7
17        6
85        4
15        4
90        4
14        2
92        2
13        1
12        1
 8        1
10        1
97        1
96        1
91        1
```

Remarks: -

There are 89 unique values for product type in the entire dataset The unique counts of each values are as displayed above:- 88 = 368 times it occurred in the dataset ,etc.

In [19]:
```
# Unique values for windspeed
print("Total Unique values are: -")
print(df_yulu['windspeed'].nunique())
print("Unique values are: -")
print(df_yulu['windspeed'].unique())
# Value counts
print("\nValue counts are: -")
print(df_yulu['windspeed'].value_counts().reset_index().to_string(inde
```

```
Total Unique values are: -
28
Unique values are: -
[ 0.      6.0032 16.9979 19.0012 19.9995 12.998  15.0013  8.9981 11.0
014
 22.0028 30.0026 23.9994 27.9993 26.0027  7.0015 32.9975 36.9974 31.0
009
 35.0008 39.0007 43.9989 40.9973 51.9987 46.0022 50.0021 43.0006 56.9
969
 47.9988]

Value counts are: -
  index  windspeed
 0.0000       1313
 8.9981       1120
11.0014       1057
12.9980       1042
 7.0015       1034
15.0013        961
 6.0032        872
16.9979        824
19.0012        676
19.9995        492
22.0028        372
23.9994        274
26.0027        235
27.9993        187
30.0026        111
31.0009         89
32.9975         80
35.0008         58
39.0007         27
36.9974         22
43.0006         12
40.9973         11
43.9989          8
46.0022          3
56.9969          2
47.9988          2
51.9987          1
50.0021          1
```

Remarks: -

There are 28 unique values for product type in the entire dataset The unique counts of each values are as displayed above:- 8.9981= 1120 times it occurred in the dataset ,etc.

In [20]:
```
# Unique values for casual
print("Total Unique values are: -")
print(df_yulu['casual'].nunique())
print("Unique values are: -")
print(df_yulu['casual'].unique())
# Value counts
print("\nValue counts are: -")
print(df_yulu['casual'].value_counts().reset_index().to_string(index=Fa
```

```
Total Unique values are: -
309
Unique values are: -
[  3   8   5   0   2   1  12  26  29  47  35  40  41  15   9   6  11
4
   7  16  20  19  10  13  14  18  17  21  33  23  22  28  48  52  42
24
  30  27  32  58  62  51  25  31  59  45  73  55  68  34  38 102  84
39
  36  43  46  60  80  83  74  37  70  81 100  99  54  88  97 144 149
124
  98  50  72  57  71  67  95  90 126 174 168 170 175 138  92  56 111
89
  69 139 166 219 240 147 148  78  53  63  79 114  94  85 128  93 121
156
 135 103  44  49  64  91 119 167 181 179 161 143  75  66 109 123 113
65
  86  82 132 129 196 142 122 106  61 107 120 195 183 206 158 137  76
115
```

Remarks: -

There are 309 unique values for product type in the entire dataset The unique counts of each values are as displayed above:- 1 = 667 times it occurred in the dataset ,etc.

In [21]:
```
# Unique values for registered
print("Total Unique values are: -")
print(df_yulu['registered'].nunique())
print("Unique values are: -")
print(df_yulu['registered'].unique())
# Value counts
print("\nValue counts are: -")
print(df_yulu['registered'].value_counts().reset_index().to_string(inde
```

```
Total Unique values are: -
731
Unique values are: -
[ 13  32  27  10   1   0   2   7   6  24  30  55  47  71  70  52  26
31
  25  17  16   8   4  19  46  54  73  64  67  58  43  29  20   9   5
3
  63 153  81  33  41  48  53  66 146 148 102  49  11  36  92 177  98
37
  50  79  68 202 179 110  34  87 192 109  74  65  85 186 166 127  82
40
  18  95 216 116  42  57  78  59 163 158  51  76 190 125 178  39  14
15
  56  60  90  83  69  28  35  22  12  77  44  38  75 184 174 154  97
214
  45  72 130  94 139 135 197 137 141 156 117 155 134  89  80 108  61
124
 132 196 107 114 172 165 105 119 183 175  88  62  86 170 145 217  91
195
 152  21 126 115 223 207 123 236 128 151 100 198 157 168  84  99 173
121
 159  93  23 212 111 193 103 113 122 106  96 249 218 194 213 191 142
224
 244 143 267 256 211 161 131 246 118 164 275 204 230 243 112 238 144
185
 101 222 138 206 104 200 129 247 140 209 136 176 120 229 210 133 259
147
 227 150 282 162 265 260 189 237 245 205 308 283 248 303 291 280 208
286
 352 290 262 203 284 293 160 182 316 338 279 187 277 362 321 331 372
377
 350 220 472 450 268 435 169 225 464 485 323 388 367 266 255 415 233
467
 456 305 171 470 385 253 215 240 235 263 221 351 539 458 339 301 397
271
 532 480 365 241 421 242 234 341 394 540 463 361 429 359 180 188 261
254
 366 181 398 272 167 149 325 521 426 298 428 487 431 288 239 453 454
345
 417 434 278 285 442 484 451 252 471 488 270 258 264 281 410 516 500
343
 311 432 475 479 355 329 199 400 414 423 232 219 302 529 510 348 346
441
 473 335 445 555 527 273 364 299 269 257 342 324 226 391 466 297 517
486
 489 492 228 289 455 382 380 295 251 418 412 340 433 231 333 514 483
276
 478 287 381 334 347 320 493 491 369 201 408 378 443 460 465 313 513
292
 497 376 326 413 328 525 296 452 506 393 368 337 567 462 349 319 300
515
 373 399 507 396 512 503 386 427 312 384 530 310 536 437 505 371 375
534
 469 474 553 402 274 523 448 409 387 438 407 250 459 425 422 379 392
430
 401 306 370 449 363 389 374 436 356 317 446 294 508 315 522 494 327
495
 404 447 504 318 579 551 498 533 332 554 509 573 545 395 440 547 557
623
 571 614 638 628 642 647 602 634 648 353 322 357 314 563 615 681 601
543
```

```
 577 354 661 653 304 645 646 419 610 677 618 595 565 586 670 656 626
581
 546 604 596 383 621 564 309 360 330 549 589 461 631 673 358 651 663
538
 616 662 344 640 659 770 608 617 584 307 667 605 641 594 629 603 518
665
 769 749 499 719 734 696 688 570 675 405 411 643 733 390 680 764 679
531
 637 652 778 703 537 576 613 715 726 598 625 444 672 782 548 682 750
716
 609 698 572 669 633 725 704 658 620 542 575 511 741 790 644 740 735
560
 739 439 660 697 336 619 712 624 580 678 684 468 649 786 718 775 636
578
 746 743 481 664 711 689 751 745 424 699 552 709 591 757 768 767 723
558
 561 403 502 692 780 622 761 690 744 857 562 702 802 727 811 886 406
787
 496 708 758 812 807 791 639 781 833 756 544 789 742 655 416 806 773
737
 706 566 713 800 839 779 766 794 803 788 720 668 490 568 597 477 583
501
 556 593 420 541 694 650 559 666 700 693 582]

Value counts are: -
 index  registered
     3         195
     4         190
     5         177
     6         155
     2         150
     1         135
     7         126
     9         114
     8         114
    11          87
    10          72
    14          67
    19          66
    23          61
    22          59
    12          57
    16          56
    15          54
    20          53
    13          52
    21          52
    18          51
    24          51
    30          49
    95          48
    26          47
    17          46
    28          46
    48          45
    31          44
    43          44
    34          43
    27          43
    49          42
   115          42
    64          42
```

```
104    41
130    41
142    41
108    41
 88    40
 86    40
 25    40
 55    40
 39    39
 53    39
176    39
 29    39
141    38
144    38
 41    38
 33    38
168    37
116    37
120    37
156    37
 99    37
112    37
 50    37
137    36
 38    36
 72    36
127    36
119    35
150    35
 96    35
134    35
 68    35
 81    35
110    35
 58    35
102    34
111    34
148    34
178    34
 83    34
161    34
 47    34
 89    34
 92    34
 80    34
 61    34
 74    34
 94    34
 62    34
 70    33
 54    33
103    33
 84    33
 46    33
 76    33
107    33
 66    33
114    33
 57    33
162    33
128    32
133    32
```

121	32
139	32
126	32
63	32
98	32
32	32
78	32
163	32
155	32
118	32
100	31
158	31
44	31
35	31
106	31
91	31
40	31
85	30
169	30
105	30
65	30
183	30
97	30
71	30
37	30
51	30
73	30
184	30
79	30
175	29
67	29
153	29
170	29
82	29
87	29
207	29
186	29
59	29
189	29
93	29
123	28
218	28
203	28
77	28
56	28
45	28
60	28
36	28
132	28
101	27
135	27
192	27
202	27
188	27
147	27
136	27
52	27
152	27
109	27
42	26
174	26
69	26

```
 90    26
157    26
154    26
244    26
124    26
177    26
209    26
230    26
187    26
166    25
256    25
138    25
151    25
214    25
211    24
171    24
146    24
204    24
220    24
149    24
122    24
190    24
143    23
129    23
206    23
145    23
252    23
180    23
 75    23
241    23
225    23
159    23
197    23
198    23
117    23
167    22
240    22
232    22
165    22
131    22
125    22
191    22
179    22
173    22
164    22
196    21
248    21
255    21
212    21
260    21
253    21
262    21
265    20
185    20
217    20
160    20
181    20
113    20
223    20
222    20
194    20
224    20
```

```
200    20
235    20
210    20
182    19
243    19
270    19
226    19
208    19
193    19
228    19
172    19
199    18
234    18
216    18
213    17
140    17
245    17
254    17
221    17
201    17
238    17
195    16
264    16
239    16
258    16
219    16
237    16
274    15
215    15
233    15
  0    15
247    15
246    15
229    15
251    15
288    15
227    15
299    15
323    15
289    14
278    14
273    14
279    14
284    14
236    14
205    13
292    13
231    13
286    13
362    13
300    13
282    13
368    13
296    13
287    12
320    12
335    12
261    12
275    12
250    12
267    12
316    12
```

```
310    12
312    12
242    12
342    11
249    11
257    11
298    11
283    11
314    11
266    11
354    11
301    11
337    11
259    11
352    11
263    11
317    11
334    10
346    10
302    10
343    10
327    10
276    10
369    10
280    10
347    10
281    10
325    10
377    10
330    10
277    10
268    10
351    10
349    10
271    10
272    10
391    10
361    10
372     9
311     9
397     9
318     9
381     9
308     9
329     9
321     9
294     9
356     9
467     9
324     9
378     9
371     9
415     8
291     8
388     8
348     8
315     8
333     8
399     8
305     8
303     8
338     8
```

393	8
357	8
432	8
350	8
297	8
341	8
285	8
309	8
345	8
269	8
374	8
363	8
426	8
340	8
364	7
408	7
483	7
355	7
386	7
319	7
480	7
365	7
328	7
306	7
331	7
344	7
313	7
290	7
304	7
489	7
376	6
379	6
380	6
367	6
437	6
332	6
366	6
359	6
442	6
429	6
540	6
339	6
458	6
414	6
453	6
307	6
435	6
358	6
470	5
449	5
326	5
375	5
474	5
472	5
539	5
360	5
409	5
387	5
398	5
462	5
421	5
486	5

```
670    5
484    5
473    5
322    5
400    5
478    5
533    5
451    5
418    5
412    5
434    5
417    5
454    5
293    5
382    5
461    4
402    4
523    4
665    4
504    4
448    4
336    4
498    4
677    4
353    4
389    4
652    4
390    4
564    4
401    4
436    4
749    4
697    4
411    4
625    4
640    4
445    4
510    4
413    4
488    4
465    4
471    4
460    4
491    4
514    4
394    4
450    4
464    4
455    4
516    4
547    3
628    3
642    3
543    3
647    3
648    3
575    3
423    3
475    3
554    3
557    3
410    3
```

```
734    3
617    3
456    3
385    3
463    3
383    3
604    3
428    3
447    3
586    3
405    3
487    3
733    3
419    3
646    3
661    3
767    3
404    3
433    3
757    3
370    3
525    3
430    3
452    3
425    3
459    3
506    3
424    3
515    3
507    3
396    3
512    3
711    3
534    3
384    3
505    3
493    3
580    3
295    3
744    3
508    3
619    2
675    2
688    2
692    2
719    2
758    2
668    2
769    2
578    2
708    2
490    2
746    2
741    2
603    2
664    2
629    2
594    2
740    2
605    2
700    2
787    2
```

```
608    2
745    2
712    2
716    2
576    2
548    2
444    2
403    2
812    2
698    2
544    2
572    2
655    2
598    2
715    2
669    2
649    2
481    2
857    2
531    2
684    2
679    2
725    2
704    2
542    2
737    2
468    2
743    2
573    2
662    2
509    2
438    2
407    2
392    2
446    2
495    2
579    2
551    2
545    2
469    2
395    2
440    2
623    2
571    2
602    2
634    2
563    2
553    2
536    2
681    2
527    2
485    2
532    2
431    2
500    2
479    2
529    2
441    2
466    2
530    2
517    2
443    2
```

```
497    2
567    2
373    2
503    2
427    2
615    2
582    2
595    2
601    2
546    2
653    2
656    2
549    2
663    2
581    2
596    2
618    2
639    1
791    1
833    1
781    1
756    1
789    1
742    1
584    1
807    1
613    1
667    1
416    1
555    1
496    1
641    1
406    1
886    1
811    1
727    1
802    1
702    1
562    1
673    1
631    1
690    1
651    1
773    1
806    1
622    1
693    1
770    1
666    1
559    1
650    1
694    1
541    1
420    1
593    1
556    1
501    1
583    1
477    1
597    1
568    1
616    1
```

```
521    1
720    1
788    1
803    1
794    1
766    1
779    1
839    1
800    1
713    1
566    1
706    1
538    1
761    1
561    1
780    1
492    1
660    1
439    1
739    1
560    1
735    1
644    1
790    1
610    1
511    1
680    1
764    1
620    1
658    1
645    1
633    1
637    1
577    1
614    1
609    1
638    1
750    1
682    1
778    1
782    1
672    1
703    1
537    1
643    1
494    1
522    1
621    1
502    1
589    1
726    1
558    1
723    1
513    1
768    1
518    1
591    1
709    1
552    1
699    1
751    1
624    1
```

```
689           1
499           1
696           1
626           1
659           1
775           1
718           1
786           1
570           1
422           1
678           1
565           1
636           1
```

Remarks: -

There are 731 unique values for product type in the entire dataset The unique counts of each values are as displayed above:- 3 = 195 times it occurred in the dataset ,etc.

In [22]:
```
# Unique values for count
print("Total Unique values are: -")
print(df_yulu['count'].nunique())
print("Unique values are: -")
print(df_yulu['count'].unique())
# Value counts
print("\nValue counts are: -")
print(df_yulu['count'].value_counts().reset_index().to_string(index=Fal
```

```
Total Unique values are: -
822
Unique values are: -
[ 16  40  32  13   1   2   3   8  14  36  56  84  94 106 110  93  67
35
  37  34  28  39  17   9   6  20  53  70  75  59  74  76  65  30  22
31
   5  64 154  88  44  51  61  77  72 157  52  12   4 179 100  42  57
78
  97  63  83 212 182 112  54  48  11  33 195 115  46  79  71  62  89
190
 169 132  43  19  95 219 122  45  86 172 163  69  23   7 210 134  73
50
  87 187 123  15  25  98 102  55  10  49  82  92  41  38 188  47 178
155
  24  18  27  99 217 130 136  29 128  81  68 139 137 202  60 162 144
158
 117  90 159 101 118 129  26 104  91 113 105  21  80 125 133 197 109
161
 135 116 176 168 108 103 175 147  96 220 127 205 174 121 230  66 114
216
 243 152 199  58 166 170 165 160 140 211 120 145 256 126 223  85 206
124
 255 222 285 146 274 272 185 191 232 327 224 107 119 196 171 214 242
148
 268 201 150 111 167 228 198 204 164 233 257 151 248 235 141 249 194
259
 156 153 244 213 181 221 250 304 241 271 282 225 253 237 299 142 313
310
 207 138 280 173 332 331 149 267 301 312 278 281 184 215 367 349 292
303
 339 143 189 366 386 273 325 356 314 343 333 226 203 177 263 297 288
236
 240 131 452 383 284 291 309 321 193 337 388 300 200 180 209 354 361
306
 277 428 362 286 351 192 411 421 276 264 238 266 371 269 537 518 218
265
 459 186 517 544 365 290 410 396 296 440 533 520 258 450 246 260 344
553
 470 298 347 373 436 378 342 289 340 382 390 358 385 239 374 598 524
384
 425 611 550 434 318 442 401 234 594 527 364 387 491 398 270 279 294
295
 322 456 437 392 231 394 453 308 604 480 283 565 489 487 183 302 547
513
 454 486 467 572 525 379 502 558 564 391 293 247 317 369 420 451 404
341
 251 335 417 363 357 438 579 556 407 336 334 477 539 551 424 346 353
481
 506 432 409 466 326 254 463 380 275 311 315 360 350 252 328 476 227
601
 586 423 330 569 538 370 498 638 607 416 261 355 552 208 468 449 381
377
 397 492 427 461 422 305 375 376 414 447 408 418 457 545 496 368 245
596
 563 443 562 229 316 402 287 372 514 472 511 488 419 595 578 400 348
587
 497 433 475 406 430 324 262 323 412 530 543 413 435 555 523 441 529
532
 585 399 584 559 307 582 571 426 516 465 329 483 600 570 628 531 455
389
```

```
 505 359 431 460 590 429 599 338 566 482 568 540 495 345 591 593 446
485
 393 500 473 352 320 479 444 462 405 620 499 625 395 528 319 519 445
512
 471 508 526 509 484 448 515 549 501 612 597 464 644 712 676 734 662
782
 749 623 713 746 651 686 690 679 685 648 560 503 521 554 541 721 801
561
 573 589 729 618 494 757 800 684 744 759 822 698 490 536 655 643 626
615
 567 617 632 646 692 704 624 656 610 738 671 678 660 658 635 681 616
522
 673 781 775 576 677 748 776 557 743 666 813 504 627 706 641 575 639
769
 680 546 717 710 458 622 705 630 732 770 439 779 659 602 478 733 650
873
 846 474 634 852 868 745 812 669 642 730 672 645 694 493 668 647 702
665
 834 850 790 415 724 869 700 793 723 534 831 613 653 857 719 867 823
403
 693 603 583 542 614 580 811 795 747 581 722 689 849 872 631 649 819
674
 830 814 633 825 629 835 667 755 794 661 772 657 771 777 837 891 652
739
 865 767 741 469 605 858 843 640 737 862 810 577 818 854 682 851 848
897
 832 791 654 856 839 725 863 808 792 696 701 871 968 750 970 877 925
977
 758 884 766 894 715 783 683 842 774 797 886 892 784 687 809 917 901
887
 785 900 761 806 507 948 844 798 827 670 637 619 592 943 838 817 888
890
 788 588 606 608 691 711 663 731 708 609 688 636]

Value counts are: -
 index  count
     5    169
     4    149
     3    144
     6    135
     2    132
     7    118
     1    105
     8     99
    10     95
    11     95
     9     83
    12     76
    16     73
    14     66
    13     65
    20     61
    17     58
    15     57
    28     54
    21     54
    23     53
    26     48
    24     44
    64     43
    31     43
    25     42
```

18	42
33	41
35	39
30	39
27	39
32	39
19	38
29	38
62	37
52	37
22	36
95	36
34	36
118	35
75	35
46	35
78	35
37	35
41	35
108	34
87	34
89	34
71	34
124	34
39	34
154	34
72	34
36	33
123	33
90	33
47	33
45	33
106	32
165	32
93	32
84	32
69	32
38	32
88	31
153	31
86	30
129	30
134	30
50	30
70	30
119	30
43	30
51	30
178	30
48	30
54	30
74	29
126	29
55	29
102	29
181	29
140	29
59	29
57	29
44	29
99	28
113	28

```
53     28
56     28
190    28
40     28
168    28
151    28
205    28
114    28
79     28
202    27
147    27
130    27
141    27
152    27
148    27
136    27
63     27
179    27
121    26
170    26
233    26
167    26
120    26
224    26
107    26
171    26
185    26
172    26
110    26
112    25
232    25
73     25
66     25
207    25
92     25
127    25
139    25
162    25
60     25
98     24
109    24
189    24
133    24
150    24
76     24
105    24
138    24
96     24
49     24
182    24
159    24
274    24
211    24
219    23
122    23
201    23
214    23
42     23
203    23
276    23
180    23
111    23
```

```
164    23
217    23
157    23
 82    23
175    23
 68    23
101    23
144    23
176    23
184    23
 94    23
188    23
204    22
128    22
222    22
228    22
209    22
191    22
149    22
 61    22
226    22
174    22
 67    22
125    22
230    22
135    22
103    21
206    21
177    21
 77    21
267    21
155    21
256    21
116    21
272    21
145    21
292    21
117    21
260    21
268    20
173    20
332    20
198    20
244    20
280    20
248    20
 65    20
213    20
281    20
229    20
 97    20
 83    20
210    20
161    20
220    20
196    20
223    20
132    20
195    20
218    20
187    20
166    20
```

```
137    20
104    19
169    19
212    19
115    19
239    19
308    19
286    19
285    19
146    19
283    19
237    18
 91    18
 80    18
238    18
251    18
299    18
216    18
270    18
163    18
241    18
343    18
 58    18
259    18
263    18
192    18
215    17
254    17
258    17
328    17
300    17
235    17
183    17
282    17
331    17
304    17
142    17
197    17
194    17
160    16
315    16
193    16
234    16
245    16
131    16
264    16
100    16
265    16
225    16
297    15
156    15
288    15
227    15
337    15
243    15
158    15
 81    15
271    15
208    14
306    14
312    14
 85    14
```

```
246    14
287    14
199    14
291    14
385    14
404    14
367    14
143    14
334    14
250    14
221    14
374    14
363    14
382    13
313    13
330    13
377    13
247    13
349    13
361    13
420    13
253    13
266    13
294    13
296    12
358    12
290    12
355    12
353    12
275    12
372    12
370    12
365    12
278    12
302    12
310    12
284    12
240    12
428    12
314    12
356    12
321    12
200    12
255    12
257    12
327    12
269    12
323    12
262    12
231    11
236    11
261    11
186    11
317    11
318    11
463    11
303    11
319    11
417    11
295    11
359    11
396    11
```

```
390    11
354    11
289    11
421    11
298    11
277    11
347    11
402    10
366    10
414    10
249    10
466    10
362    10
357    10
351    10
273    10
406    10
325    10
242    10
459    10
360    10
492    10
338    10
342    10
389    10
341    10
324     9
329     9
585     9
408     9
336     9
499     9
376     9
305     9
311     9
495     9
320     9
346     9
425     9
371     9
392     9
373     9
398     9
410     9
386     9
301     9
388     9
322     9
387     9
419     8
339     8
279     8
309     8
432     8
483     8
326     8
400     8
348     8
380     8
441     8
384     8
350     8
```

```
435    8
436    8
596    8
397    8
293    8
352    8
513    8
457    8
316    8
447    8
446    8
375    8
452    8
391    8
333    8
381    8
369    8
335    8
566    8
427    8
497    7
430    7
423    7
449    7
470    7
489    7
378    7
514    7
500    7
488    7
416    7
340    7
498    7
487    7
579    7
413    7
454    7
486    7
467    7
379    7
479    7
480    7
568    7
451    7
394    7
456    7
405    7
512    7
505    7
395    7
411    7
472    6
503    6
563    6
555    6
443    6
433    6
593    6
399    6
345    6
383    6
307    6
```

```
520     6
465     6
445     6
455     6
412     6
473     6
364     6
409     6
407     6
496     6
252     6
401     6
476     6
627     6
493     6
539     6
569     6
453     6
527     6
678     6
668     6
478     6
564     6
547     6
502     6
525     6
461     6
418     6
344     6
558     6
491     6
450     6
448     5
458     5
442     5
481     5
509     5
546     5
571     5
508     5
464     5
368     5
531     5
550     5
517     5
562     5
440     5
617     5
681     5
560     5
586     5
590     5
460     5
642     4
462     4
444     4
654     4
439     4
507     4
730     4
501     4
526     4
```

```
504    4
536    4
490    4
646    4
729    4
704    4
671    4
573    4
541    4
635    4
522    4
679    4
686    4
576    4
713    4
662    4
743    4
644    4
615    4
494    4
393    4
516    4
638    4
538    4
523    4
511    4
552    4
537    4
518    4
485    4
434    4
559    4
437    4
565    4
545    4
438    4
544    4
540    4
578    4
556    4
431    4
482    4
567    3
610    3
542    3
474    3
692    3
643    3
553    3
468    3
626    3
632    3
607    3
580    3
653    3
403    3
694    3
659    3
551    3
770    3
812    3
424    3
```

```
710    3
506    3
641    3
524    3
604    3
647    3
415    3
557    3
724    3
601    3
673    3
618    3
744    3
515    3
554    3
471    3
623    3
484    3
651    3
584    3
543    3
605    3
475    3
570    3
528    3
426    3
582    3
835    3
839    3
620    3
595    3
592    3
549    3
591    3
597    3
834    2
715    2
814    2
672    2
784    2
645    2
723    2
687    2
572    2
619    2
669    2
745    2
731    2
868    2
852    2
766    2
808    2
534    2
469    2
674    2
649    2
631    2
633    2
872    2
689    2
581    2
795    2
```

```
811      2
772      2
837      2
858      2
613      2
640      2
614      2
737      2
810      2
583      2
693      2
682      2
823      2
856      2
719      2
884      2
634      2
702      2
782      2
656      2
429      2
561      2
529      2
521      2
616      2
628      2
598      2
738      2
721      2
648      2
422      2
533      2
698      2
822      2
759      2
684      2
800      2
757      2
611      2
677      2
680      2
706      2
676      2
477      2
602      2
712      2
705      2
622      2
530      2
600      2
813      2
594      2
612      1
842      1
892      1
734      1
886      1
970      1
877      1
925      1
797      1
725      1
```

774	1
863	1
871	1
683	1
977	1
792	1
968	1
783	1
758	1
696	1
532	1
894	1
701	1
750	1
650	1
519	1
838	1
688	1
609	1
708	1
599	1
663	1
711	1
691	1
608	1
606	1
588	1
788	1
890	1
888	1
817	1
943	1
809	1
625	1
637	1
670	1
827	1
798	1
844	1
948	1
806	1
761	1
900	1
791	1
887	1
901	1
917	1
785	1
685	1
832	1
849	1
747	1
655	1
624	1
603	1
660	1
658	1
867	1
857	1
781	1
775	1
831	1

```
748      1
793      1
700      1
869      1
776      1
790      1
850      1
666      1
665      1
575      1
639      1
769      1
717      1
630      1
732      1
779      1
733      1
846      1
722      1
819      1
897      1
830      1
848      1
851      1
749      1
854      1
818      1
577      1
862      1
746      1
690      1
843      1
873      1
587      1
741      1
767      1
865      1
739      1
652      1
891      1
777      1
771      1
657      1
661      1
794      1
755      1
667      1
801      1
629      1
825      1
589      1
636      1
```

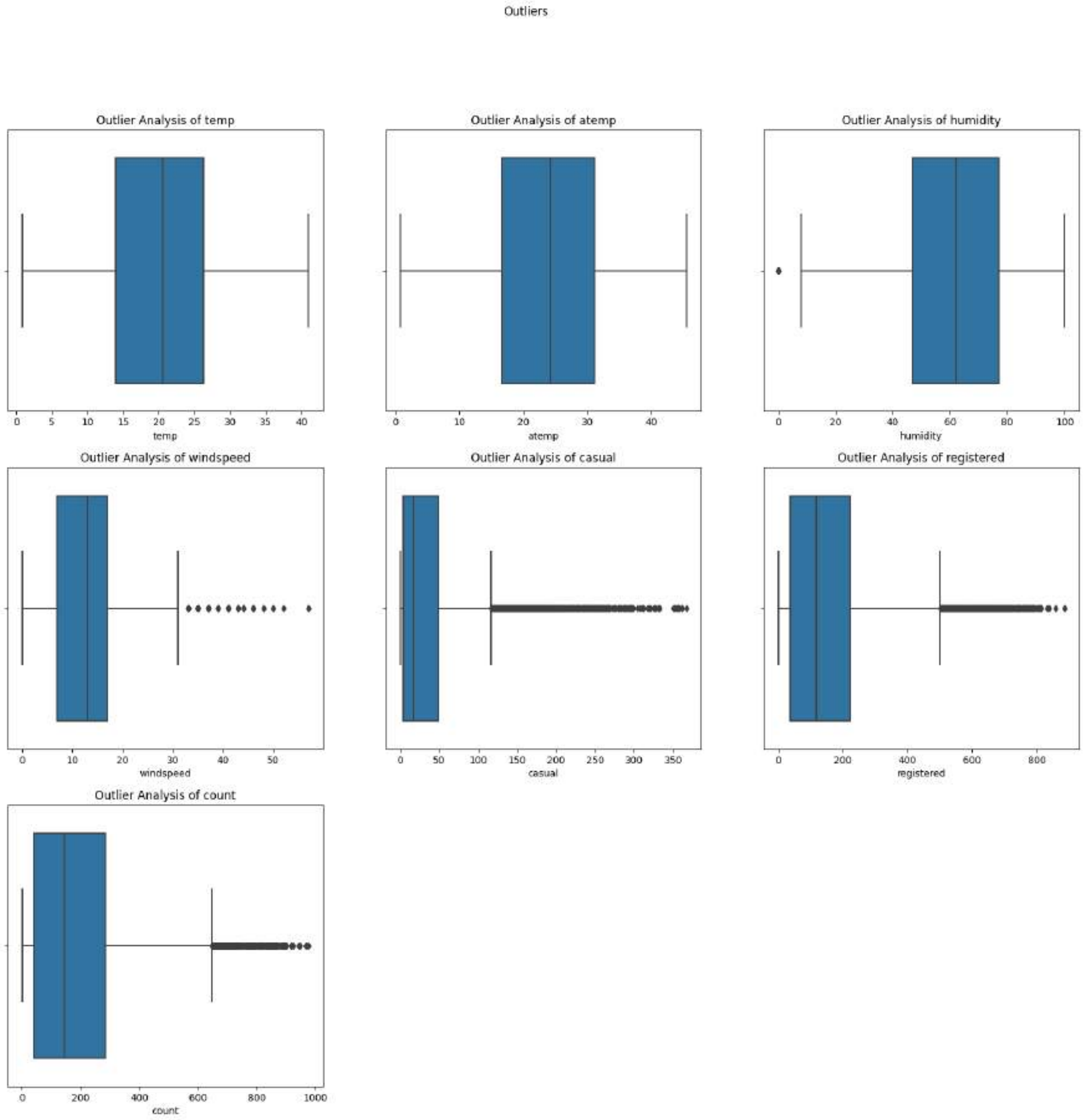
Remarks: -

There are 822 unique values for product type in the entire dataset The unique counts of each values are as displayed above:- 5 = 169 times it occurred in the dataset ,etc.

Univariate Analysis

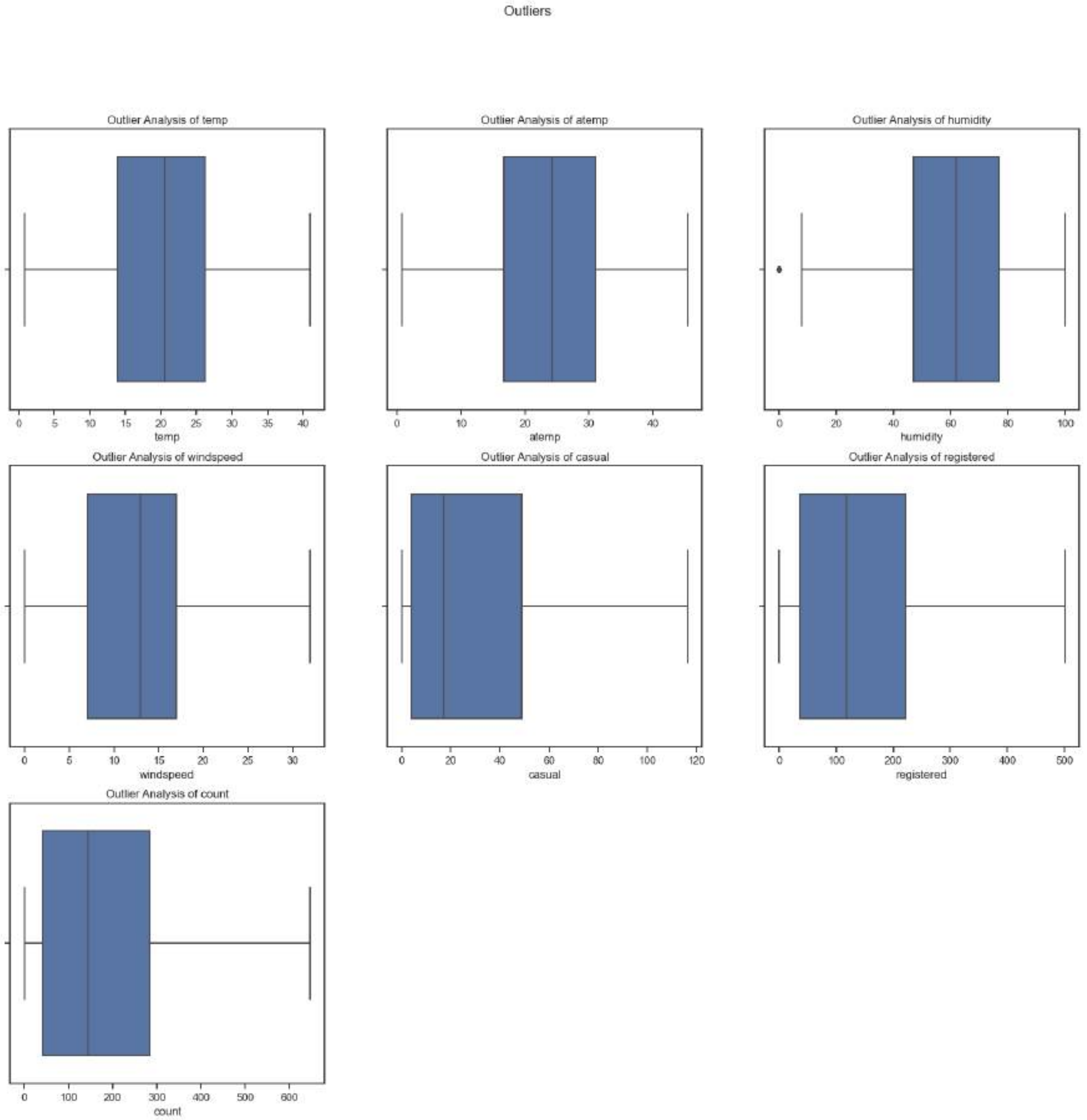
Outlier Checks and Treatment

In [23]:
```
# Outliers handling

plt.figure(figsize = (20,18))
plt.suptitle("Outliers")
features = ['temp', 'atemp','humidity', 'windspeed', 'casual', 'registe
for i in range(len(features)):
     plt.subplot(3, 3, i+1)
     sns.boxplot(x = df_yulu[features[i]])
     plt.title('Outlier Analysis of {}'.format(features[i]))
plt.show()
```



In [24]:
```
# Handling above outliers for last 4 plots

def handle_outliers(df, columns_list):
    for col in columns_list:
        Q1, Q2, Q3 = df[col].quantile([0.25, 0.5, 0.75])
        IQR = Q3 - Q1
        lower_whisker = Q1 - (1.5 * IQR)
        upper_whisker = Q3 + (1.5 * IQR)
# Replacing outliers with lower or upper whisker values
        df[col] = df[col].apply(lambda x: lower_whisker if x < lower_wh

handle_outliers(df_yulu, ["windspeed", "casual", "registered", "count"]
```

In [40]:
```
# Outliers are handled

plt.figure(figsize = (20,18))
plt.suptitle("Outliers")
features = ['temp', 'atemp','humidity', 'windspeed', 'casual', 'registe
for i in range(len(features)):
     plt.subplot(3, 3, i+1)
     sns.boxplot(x = df_yulu[features[i]])
     plt.title('Outlier Analysis of {}'.format(features[i]))
plt.show()
```



Handling column values

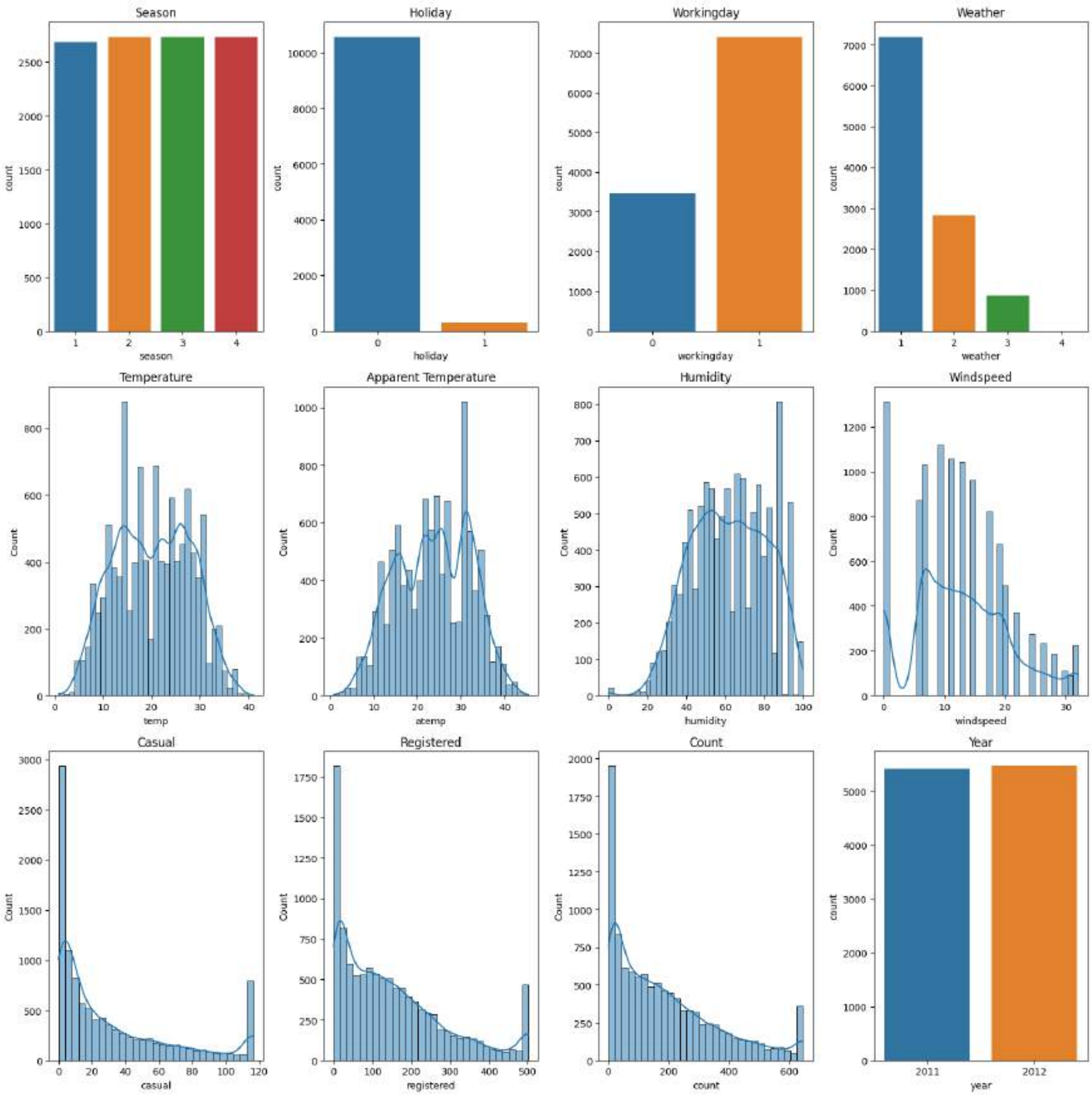
In [25]:
```
# Creating few columns from datetime field for better analysis of the d
df_yulu['datetime']=pd.to_datetime(df_yulu['datetime'])

df_yulu["year"] = df_yulu["datetime"].dt.year
df_yulu["month"] = df_yulu["datetime"].dt.month
df_yulu["day"] = df_yulu["datetime"].dt.day
df_yulu["hour_of_the_day"] = df_yulu["datetime"].dt.hour
df_yulu['quarter'] = df_yulu['datetime'].dt.quarter

# Dropping the datetime column, as the required data has been extracted
df_yulu.drop(["datetime"],axis=1, inplace=True)

# Modifying the data type of following few features, as per the underst
df_yulu["season"] = df_yulu["season"].astype("object")
df_yulu["holiday"] = df_yulu["holiday"].astype("object")
df_yulu["workingday"] = df_yulu["workingday"].astype("object")
df_yulu["weather"] = df_yulu["weather"].astype("object")
df_yulu["year"] = df_yulu["year"].astype("object")
df_yulu["month"] = df_yulu["month"].astype("object")
df_yulu["day"] = df_yulu["day"].astype("object")
df_yulu["hour_of_the_day"] = df_yulu["hour_of_the_day"].astype("object'
df_yulu["quarter"] = df_yulu["quarter"].astype("object")
```

In [97]:
```
# Column updation review

df_yulu.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10886 entries, 0 to 10885
Data columns (total 16 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   season           10886 non-null  object 
 1   holiday          10886 non-null  object 
 2   workingday       10886 non-null  object 
 3   weather          10886 non-null  object 
 4   temp             10886 non-null  float64
 5   atemp            10886 non-null  float64
 6   humidity         10886 non-null  int64  
 7   windspeed        10886 non-null  float64
 8   casual           10886 non-null  float64
 9   registered       10886 non-null  float64
 10  count            10886 non-null  float64
 11  year             10886 non-null  object 
 12  month            10886 non-null  object 
 13  day              10886 non-null  object 
 14  hour_of_the_day  10886 non-null  object 
 15  quarter          10886 non-null  object 
dtypes: float64(6), int64(1), object(9)
memory usage: 1.3+ MB
```

Graphical analysis

In [26]:
```
# Setting up a 4x4 grid of subplots for the plots
fig, axes = plt.subplots(nrows=3, ncols=4, figsize=(16, 16))

# Plot 1: Season (Categorical)
sns.countplot(data=df_yulu, x='season', ax=axes[0, 0])
axes[0, 0].set_title('Season')

# Plot 2: Holiday (Categorical)
sns.countplot(data=df_yulu, x='holiday', ax=axes[0, 1])
axes[0, 1].set_title('Holiday')

# Plot 3: Workingday (Categorical)
sns.countplot(data=df_yulu, x='workingday', ax=axes[0, 2])
axes[0, 2].set_title('Workingday')

# Plot 4: Weather (Categorical)
sns.countplot(data=df_yulu, x='weather', ax=axes[0, 3])
axes[0, 3].set_title('Weather')

# Plot 5: Temperature (Continuous)
sns.histplot(data=df_yulu, x='temp', ax=axes[1, 0], kde=True)
axes[1, 0].set_title('Temperature')

# Plot 6: Apparent Temperature (Continuous)
sns.histplot(data=df_yulu, x='atemp', ax=axes[1, 1], kde=True)
axes[1, 1].set_title('Apparent Temperature')

# Plot 7: Humidity (Continuous)
sns.histplot(data=df_yulu, x='humidity', ax=axes[1, 2], kde=True)
axes[1, 2].set_title('Humidity')

# Plot 8: Windspeed (Continuous)
sns.histplot(data=df_yulu, x='windspeed', ax=axes[1, 3], kde=True)
axes[1, 3].set_title('Windspeed')

# Plot 9: Casual (Continuous)
sns.histplot(data=df_yulu, x='casual', ax=axes[2, 0], kde=True)
axes[2, 0].set_title('Casual')

# Plot 10: Registered (Continuous)
sns.histplot(data=df_yulu, x='registered', ax=axes[2, 1], kde=True)
axes[2, 1].set_title('Registered')

# Plot 11: Count (Continuous)
sns.histplot(data=df_yulu, x='count', ax=axes[2, 2], kde=True)
axes[2, 2].set_title('Count')

# Plot 12: Year (Categorical)
sns.countplot(data=df_yulu, x='year', ax=axes[2, 3])
axes[2, 3].set_title('Year')

# Adjusting the layout and displaying the plots
plt.tight_layout()
plt.show()
```

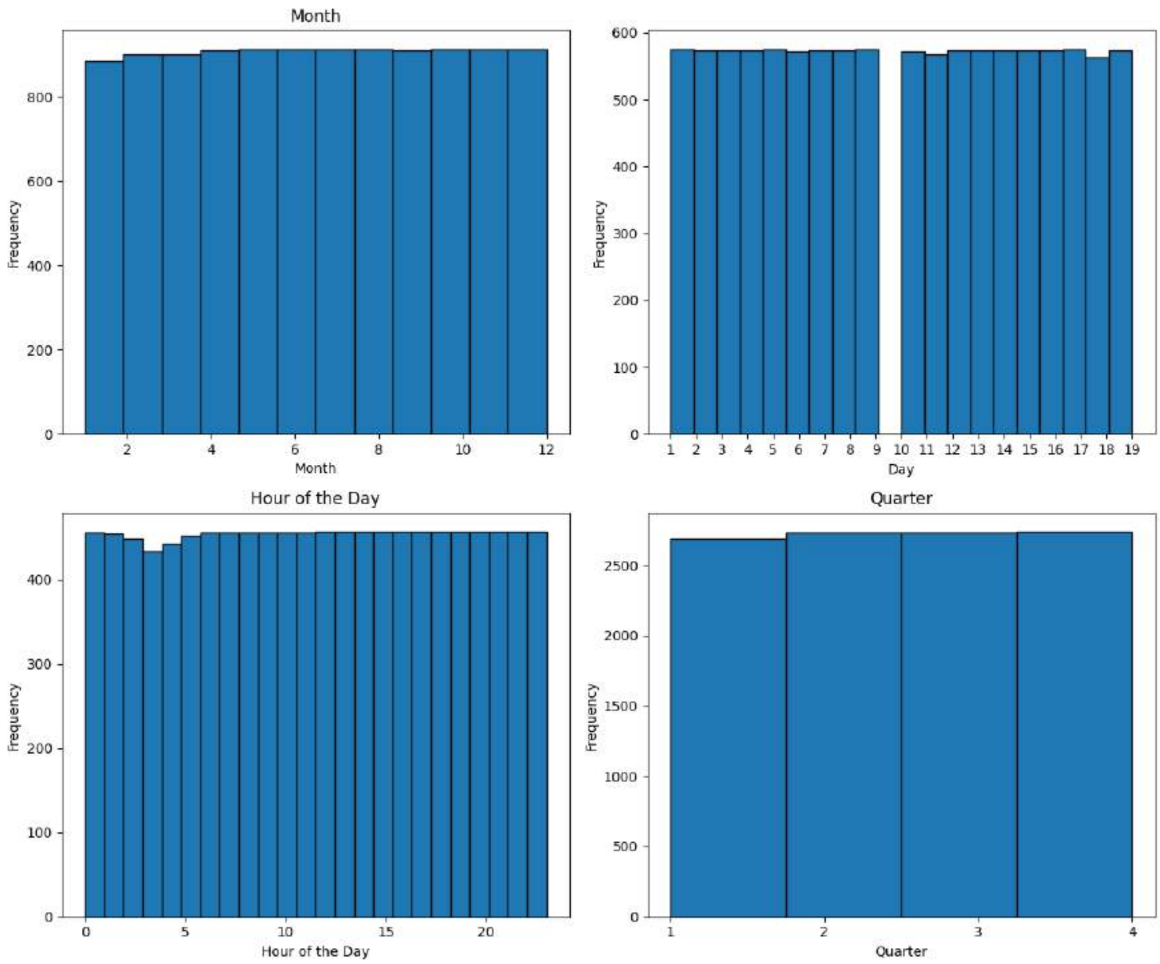


Observations

1. Graph 1: - Season 1 has less number of counts as compared to 2,3 & 4 which are almost same.
2. Graph 2: - Holiday 1 is the least and the maximum Holiday is 0, i.e., less holiday are present compared to working days in dataset.
3. Graph 3: - Working Day 0 is the least and the maximum Working Day is 1, i.e., less holiday are present compared to working days in dataset.
4. Graph 4: - Clear, Few clouds, partly cloudy, partly cloudy has max occurence and Heavy Rain + Ice Pallets + Thunderstorm + Mist, Snow + Fog has least.
5. Graph 5: - The max temp recorded is around 15° cel and the entire distribution is unevenly distributed.
6. Graph 6: - The max feel temp recorded is around 32° cel and the entire distribution is unevenly distributed.
7. Graph 7: - The max humidity is around 83 and the least recorded lies between 5-10.
8. Graph 8: - The max windspeed recorded is around 1 and least is around 31.
9. Graph 9: - The max count of casual users is 1 and there is a decline in the data points with a sudden peak towards the end (120).
10. Graph 10: - The max count of registered users is 1 and there is a decline in the data points with a sudden peak towards the end (500).

11. Graph 11: - The max count of total rental bikes including both casual and registered is around 1 and there is a decline in the data points with a sudden peak towards the end (620).

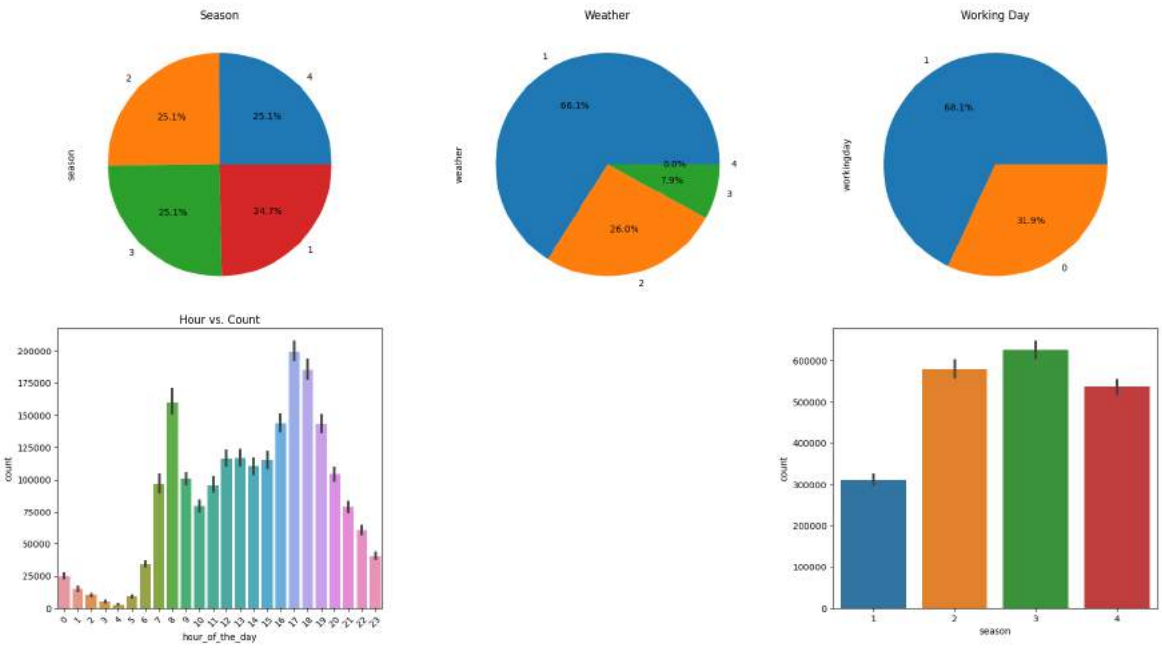
In [27]:
```
# Setting up a 2x2 grid of subplots for the histograms
fig, axes = plt.subplots(nrows=2, ncols=2, figsize=(12, 10))

# Plot 1: Month (Categorical)
axes[0, 0].hist(df_yulu['month'], bins=12, edgecolor='black')
axes[0, 0].set_title('Month')
axes[0, 0].set_xlabel('Month')
axes[0, 0].set_ylabel('Frequency')

# Plot 2: Day (Categorical)
axes[0, 1].set_xlabel('Day')
axes[0, 1].set_ylabel('Frequency')
axes[0, 1].hist(df_yulu['day'], bins=20, edgecolor='black')
whole_numbers = [int(x) for x in df_yulu['day'].unique()]
axes[0, 1].set_xticks(whole_numbers)
axes[0, 1].set_xticklabels(whole_numbers)

# Plot 3: Hour of the Day (Categorical)
axes[1, 0].hist(df_yulu['hour_of_the_day'], bins=24, edgecolor='black')
axes[1, 0].set_title('Hour of the Day')
axes[1, 0].set_xlabel('Hour of the Day')
axes[1, 0].set_ylabel('Frequency')

# Plot 4: Quarter (Categorical)
axes[1, 1].hist(df_yulu['quarter'], bins=4, edgecolor='black')
axes[1, 1].set_title('Quarter')
axes[1, 1].set_xlabel('Quarter')
axes[1, 1].set_ylabel('Frequency')
whole_numbers = [int(x) for x in df_yulu['quarter'].unique()]
axes[1, 1].set_xticks(whole_numbers)
axes[1, 1].set_xticklabels(whole_numbers)

# Adjusting the layout and displaying the plots
plt.tight_layout()
plt.show()
```



Observations

1. Graph 1: - January month has slight less users followed by February until there is saturation in counts from April onwards.
2. Graph 2: - There is not much difference in counts of days and its almost uniform across dataset and least is at 18.
3. Graph 3: - There is not much difference in hours of days and its almost uniform across dataset and least is at
4. Graph 4: - Average count of rented cycles is lower in first quarter, then average is same across rest 3 quarters.

In [28]:
```
# Creating subplots for the pie charts and bar plots
fig, axes = plt.subplots(2, 3, figsize=(18, 10))

# Plot 1: Pie Chart for 'season'
df_yulu['season'].value_counts().plot.pie(autopct='%1.1f%%', ax=axes[0,
axes[0, 0].set_title('Season')

# Plot 2: Pie Chart for 'weather'
df_yulu['weather'].value_counts().plot.pie(autopct='%1.1f%%', ax=axes[(
axes[0, 1].set_title('Weather')

# Plot 3: Pie Chart for 'workingday'
df_yulu['workingday'].value_counts().plot.pie(autopct='%1.1f%%', ax=axe
axes[0, 2].set_title('Working Day')

# Plot 4: Bar Plot for 'hour' vs. 'count'
sns.barplot(data=df_yulu, x='hour_of_the_day', y='count', estimator=sur
axes[1, 0].set_title('Hour vs. Count')
axes[1, 0].set_xticklabels(axes[1, 0].get_xticklabels(), rotation=45)

# Plot 6: Bar Plot for 'season' vs. 'count'
sns.barplot(data=df_yulu, x='season', y='count', estimator=sum, ax=axes
axes[1, 1].set_title('Season vs. Count')

# Dropping unused axes
fig.delaxes(axes[1][1])

# Adjusting the layout and displaying the plots
plt.tight_layout()
plt.show()
```

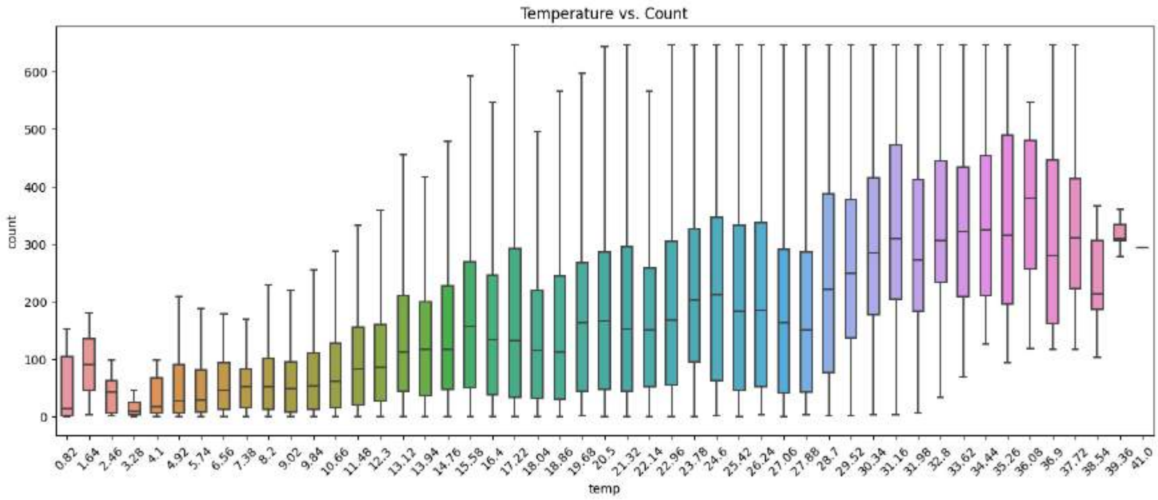


Observations

1. Graph 1: - Spring (1 season) has less spread as compared to rest 3 seasons which are same w.r.t percentage distribution.
2. Graph 2: - Average count of rented cycles is higher in clean/cloudy weather, followed by mist/cloudy weather.
3. Graph 3: - Working days has 68.1% distribution compared to non-working days (31.9%).

4. Graph 4: - Average count of rented cycles stays considerably higher from 7am till 9am, and then start decreasing till 3pm, and again start increasing till 7pm, with highest at 5pm.
5. Graph 5: - The max season is of fall followed by summer

In [29]:
```
# Creating a figure
plt.figure(figsize=(16, 6))

# Plotting the boxplot with whole numbers on the x-axis
sns.boxplot(data=df_yulu, x='temp', y='count', width=0.5, showfliers=Fa
plt.title('Temperature vs. Count')

# Setting the x-axis to display numbers
plt.xticks(rotation=45)

plt.show()
```



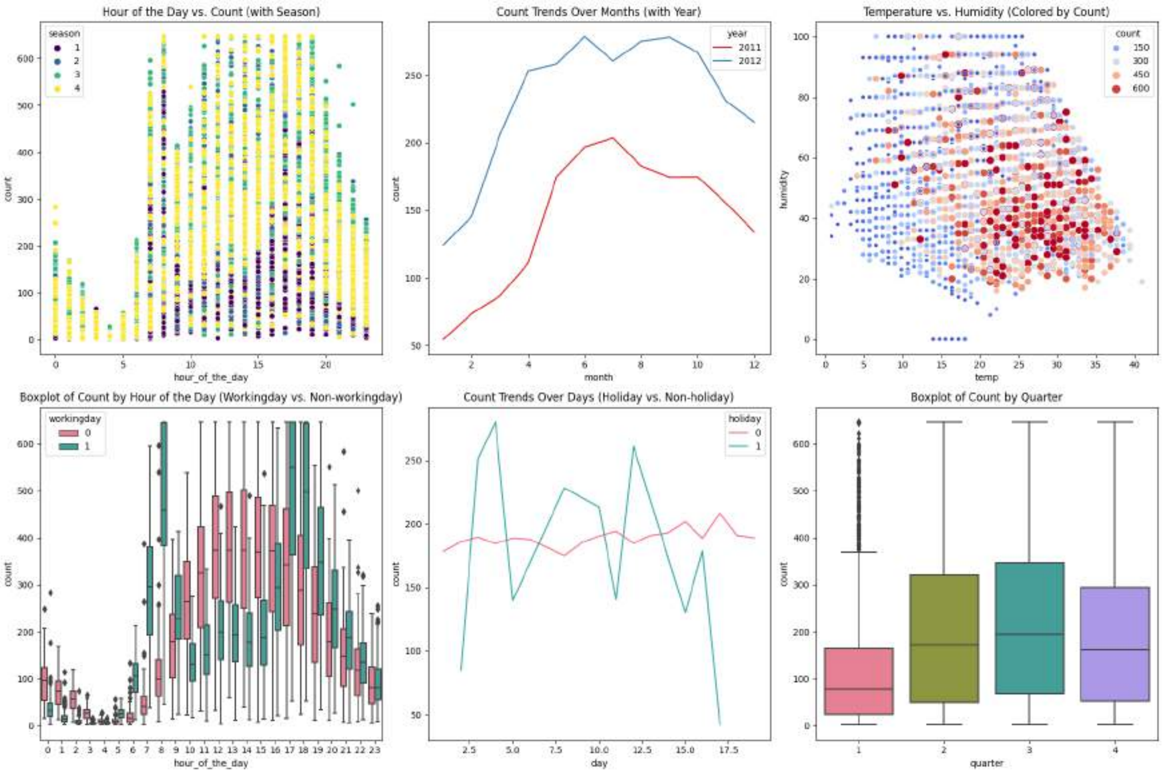
Observations

The count median of temp lies maximum around 100-150 for max observed temp which falls between ~ 14° to ~ 28°. Also the max temp observed (upper whiskers) is around 17°.

Bivariate Analysis

In [30]:
```
# Creating a 2x3 grid of subplots
fig, axes = plt.subplots(2, 3, figsize=(18, 12))

# Plot 1 - Scatter plot with 'season' as hue
sns.scatterplot(data=df_yulu, x='hour_of_the_day', y='count', hue='seas
axes[0, 0].set_title('Hour of the Day vs. Count (with Season)')

# Plot 2 - Line plot with 'year' as hue
sns.lineplot(data=df_yulu, x='month', y='count', hue='year', palette='S
axes[0, 1].set_title('Count Trends Over Months (with Year)')

# Plot 3 - Scatter plot with 'count' as color scale
sns.scatterplot(data=df_yulu, x='temp', y='humidity', hue='count', pale
axes[0, 2].set_title('Temperature vs. Humidity (Colored by Count)')

# Plot 4 - Box plots with 'workingday' as hue
sns.boxplot(data=df_yulu, x='hour_of_the_day', y='count', hue='working
axes[1, 0].set_title('Boxplot of Count by Hour of the Day (Workingday v

# Plot 5 - Line plot with 'holiday' as hue
sns.lineplot(data=df_yulu, x='day', y='count', hue='holiday', palette=
axes[1, 1].set_title('Count Trends Over Days (Holiday vs. Non-holiday)

# Plot 6 - Box plots for 'quarter'
sns.boxplot(data=df_yulu, x='quarter', y='count', palette='husl', ax=ax
axes[1, 2].set_title('Boxplot of Count by Quarter')

# Adjusting the layout and displaying the plots
plt.tight_layout()
plt.show()
```

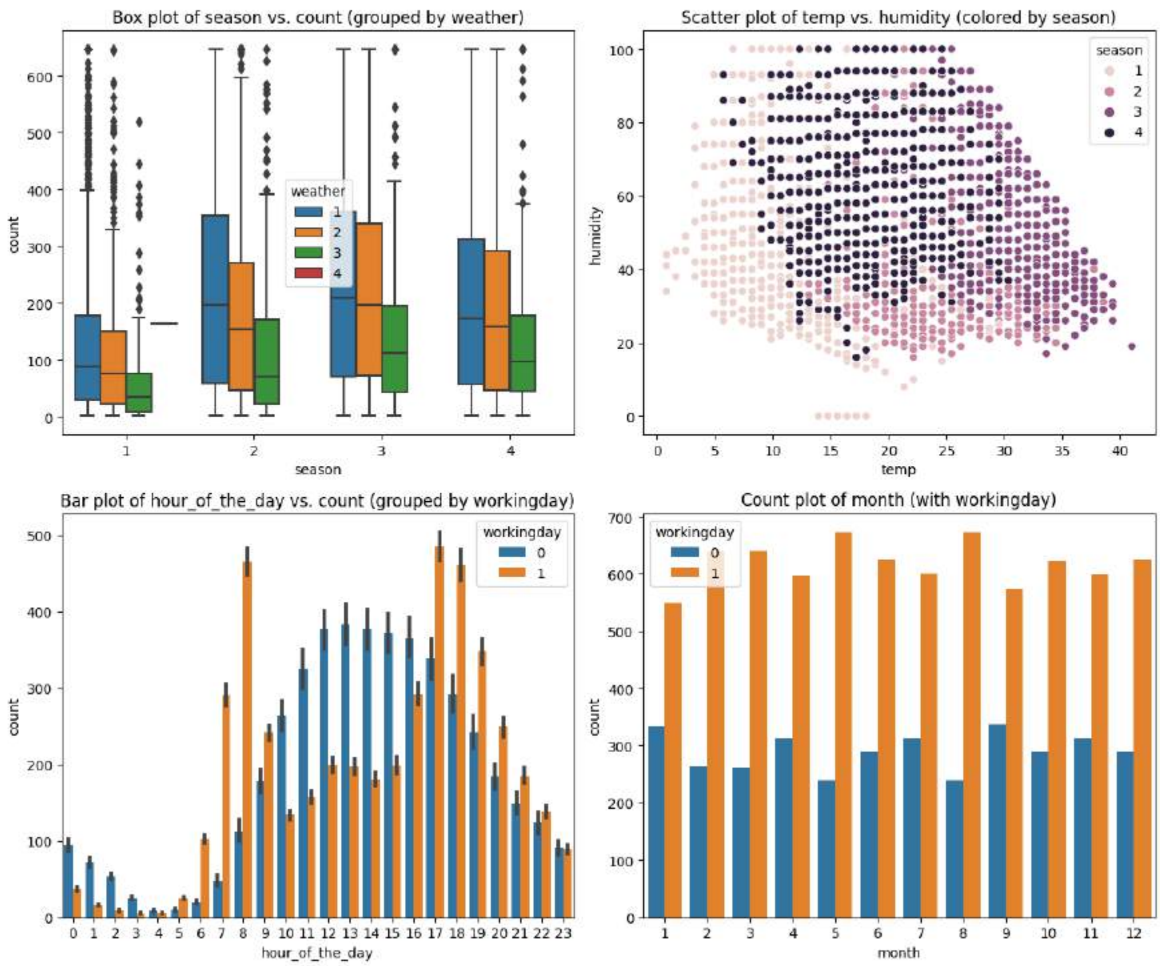
```
/var/folders/3x/g_8g6pcj3cv64hq5g80q28tw0000gp/T/ipykernel_36376/1606
522860.py:12: FutureWarning:

The `ci` parameter is deprecated. Use `errorbar=None` for the same ef
fect.

  sns.lineplot(data=df_yulu, x='month', y='count', hue='year', palett
e='Set1', ci=None, ax=axes[0, 1])
/var/folders/3x/g_8g6pcj3cv64hq5g80q28tw0000gp/T/ipykernel_36376/1606
522860.py:24: FutureWarning:

The `ci` parameter is deprecated. Use `errorbar=None` for the same ef
fect.

  sns.lineplot(data=df_yulu, x='day', y='count', hue='holiday', palet
te='husl', ci=None, ax=axes[1, 1])
```



Observations

1. Graph 1: - The max distributions are w.r.t season 4 and between 12-5 pm. The less distributions are w.r.t season 3.
2. Graph 2: - The trend is observed more in 2012 compared to 2011 with peak during the months of jun(2012) 7 jul(2011) and after that there is a decline seeing in the trend when winter approaches.
3. Graph 3: - The count of temp is more for temp ranges ~ 22-30° (count >600) and the distribution of counts is more w.r.t 150.
4. Graph 4: - During working days, the peak hours of usage is basically around 8am (people going to office, etc.) & then around 6-7pm (people going back home).
5. Graph 5: - The peak usage is during working days/non-holiday as compared to holidays(peak is at during evening).
6. Graph 6: - The median quarter count is around 180 for 2, 3 & 4. The highest is for quarter 3.

In [31]:
```
# Creating a 2x2 grid of subplots
fig, axes = plt.subplots(2, 2, figsize=(12, 10))

# Plot 1 - Box plot of 'season' vs. 'count' grouped by 'weather'
sns.boxplot(x='season', y='count', hue='weather', data=df_yulu, ax=axes
axes[0, 0].set_title('Box plot of season vs. count (grouped by weather)

# Plot 2 - Scatter plot of 'temp' vs. 'humidity' colored by 'season'
sns.scatterplot(x='temp', y='humidity', hue='season', data=df_yulu, ax=
axes[0, 1].set_title('Scatter plot of temp vs. humidity (colored by sea

# Plot 3 - Bar plot of 'hour_of_the_day' vs. 'count' grouped by 'workin
sns.barplot(x='hour_of_the_day', y='count', hue='workingday', data=df_y
axes[1, 0].set_title('Bar plot of hour_of_the_day vs. count (grouped by

# Plot 4 - Count plot of 'month' with 'workingday' as hue
sns.countplot(x='month', hue='workingday', data=df_yulu, ax=axes[1, 1])
axes[1, 1].set_title('Count plot of month (with workingday)')

# Adjusting the layout and displaying the plots
plt.tight_layout()
plt.show()
```



Observations

Graph 1: - Another way of expressing season w.r.t weather based on counts.

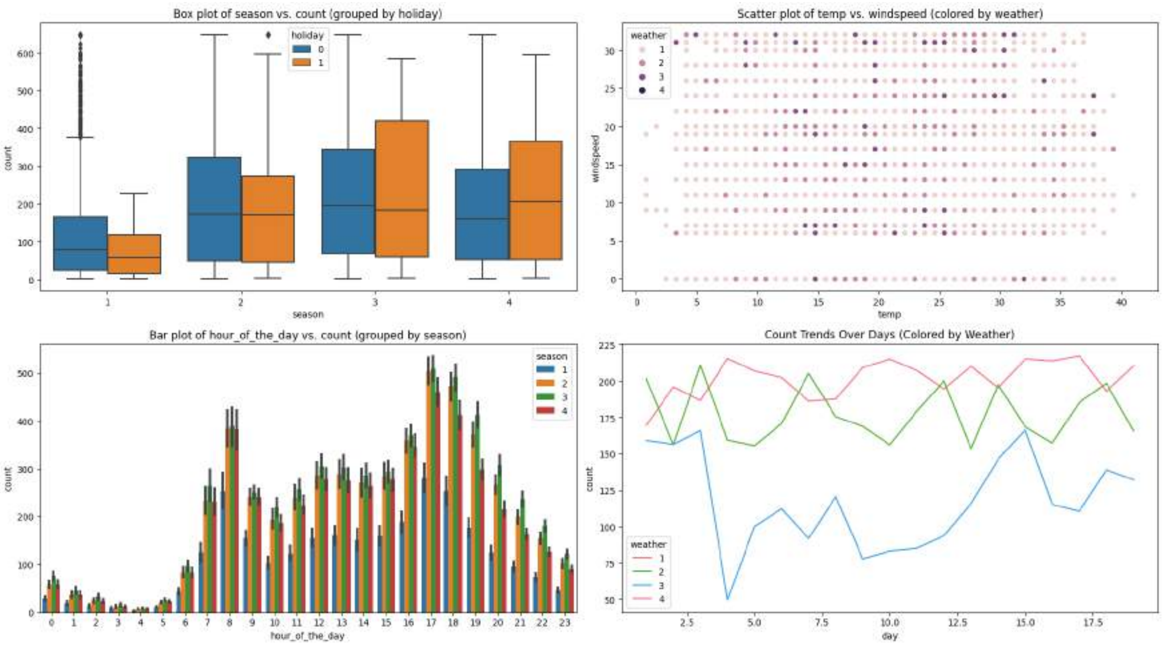
Graph 2: - The max distribution is w.r.t season 3 and 4 with high humidity that lies between temp ranges 12-25°.

Graph 3: - Another way of expressing working day w.r.t hours of day based on counts.

Graph 4: - Another way of expressing working day w.r.t month based on counts.

In [32]:
```
# Create a 2x3 grid of subplots
fig, axes = plt.subplots(2, 2, figsize=(18, 10))

# Plot 1 - Box plot of 'season' vs. 'count' grouped by 'holiday'
sns.boxplot(x='season', y='count', hue='holiday', data=df_yulu, ax=axes
axes[0, 0].set_title('Box plot of season vs. count (grouped by holiday)

# Plot 2 - Scatter plot of 'temp' vs. 'windspeed' colored by 'weather'
sns.scatterplot(x='temp', y='windspeed', hue='weather', data=df_yulu, a
axes[0, 1].set_title('Scatter plot of temp vs. windspeed (colored by we

# Plot 3 - Bar plot of 'hour_of_the_day' vs. 'count' grouped by 'season
sns.barplot(x='hour_of_the_day', y='count', hue='season', data=df_yulu,
axes[1, 0].set_title('Bar plot of hour_of_the_day vs. count (grouped by

# Plot 4 - Line plot of 'day' vs. 'count' colored by 'weather'
sns.lineplot(x='day', y='count', hue='weather', data=df_yulu, palette=
axes[1, 1].set_title('Count Trends Over Days (Colored by Weather)')

# Adjusting the layout and displaying the plots
plt.tight_layout()
plt.show()
```

```
/var/folders/3x/g_8g6pcj3cv64hq5g80q28tw0000gp/T/ipykernel_36376/2892
442352.py:17: FutureWarning:

The `ci` parameter is deprecated. Use `errorbar=None` for the same ef
fect.

  sns.lineplot(x='day', y='count', hue='weather', data=df_yulu, palet
te='husl', ci=None, ax=axes[1, 1])
```



Observations: -

1. Graph 1: - The median of season 2 and 3 for both holiday/non-holiday lies around 180 and highest is for season 3 and holiday.

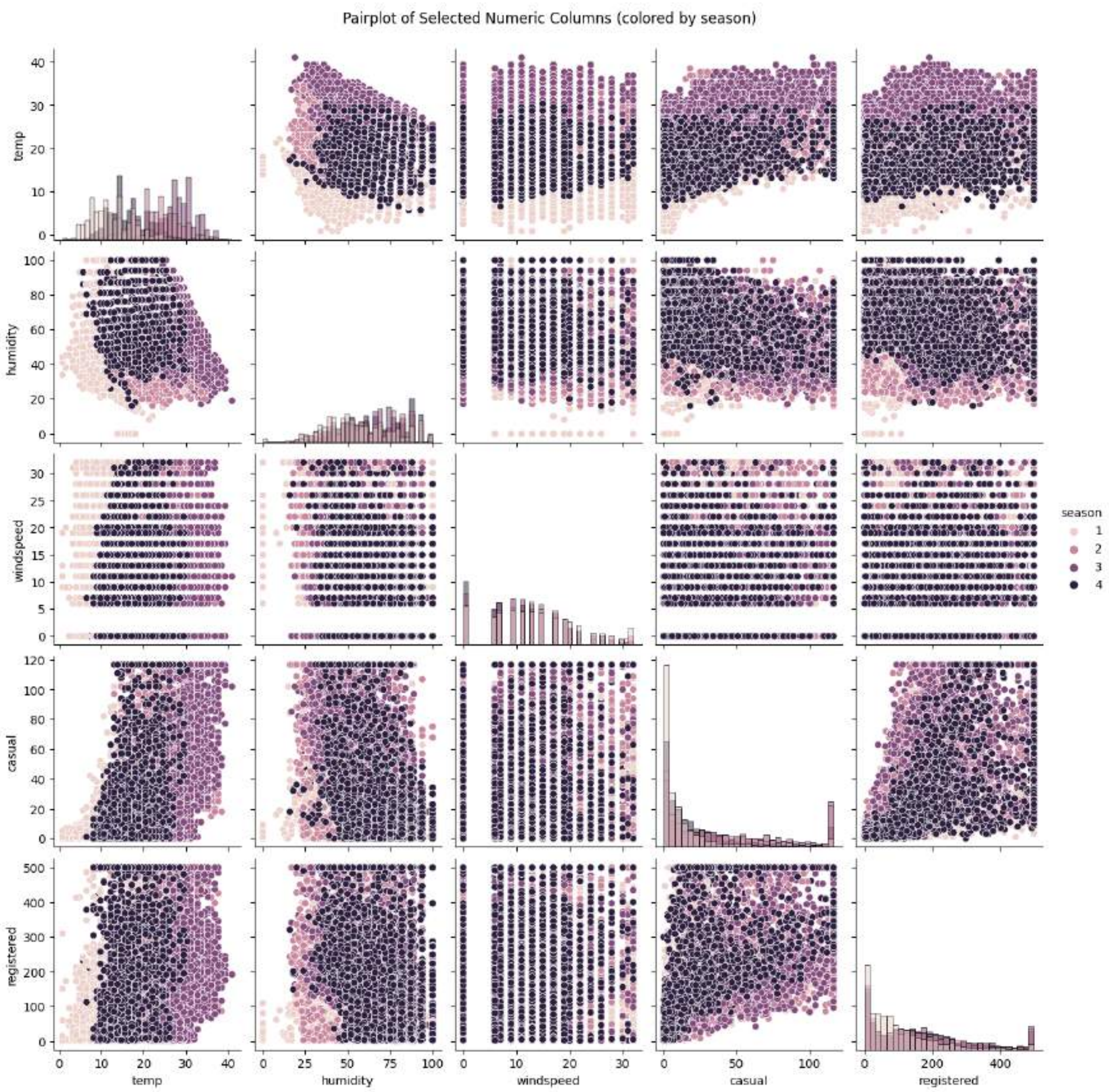
2. Graph 2: - Another way of plotting weather and temp and windspeed. The distribution is scattered and for temp around 25° the weather 3(light snow,etc.) has more windspeed around 32.
3. Graph 3: - The max season w.r.t 2 and 3 has more count during hours of day at 5pm.
4. Graph 4: - This also resembles a graph depicting relation between days, weather & their count. The weather 4 has a dip around day 4th (=50) and the distribution of all weather is not uniform

Multivariate Analysis

In [33]:
```
# Plot 1 - Pair plot for selected numeric columns with 'hue' as 'seasor
selected_columns = ['temp', 'humidity', 'windspeed', 'casual', 'registe
g = sns.pairplot(df_yulu, vars=selected_columns, hue='season', diag_kir
g.fig.suptitle('Pairplot of Selected Numeric Columns (colored by seasor
```

```
/Users/0438mpind/anaconda3/lib/python3.10/site-packages/seaborn/axisg
rid.py:118: UserWarning: The figure layout has changed to tight
  self._figure.tight_layout(*args, **kwargs)
```

Out[33]:
```
Text(0.5, 1.02, 'Pairplot of Selected Numeric Columns (colored by sea
son)')
```

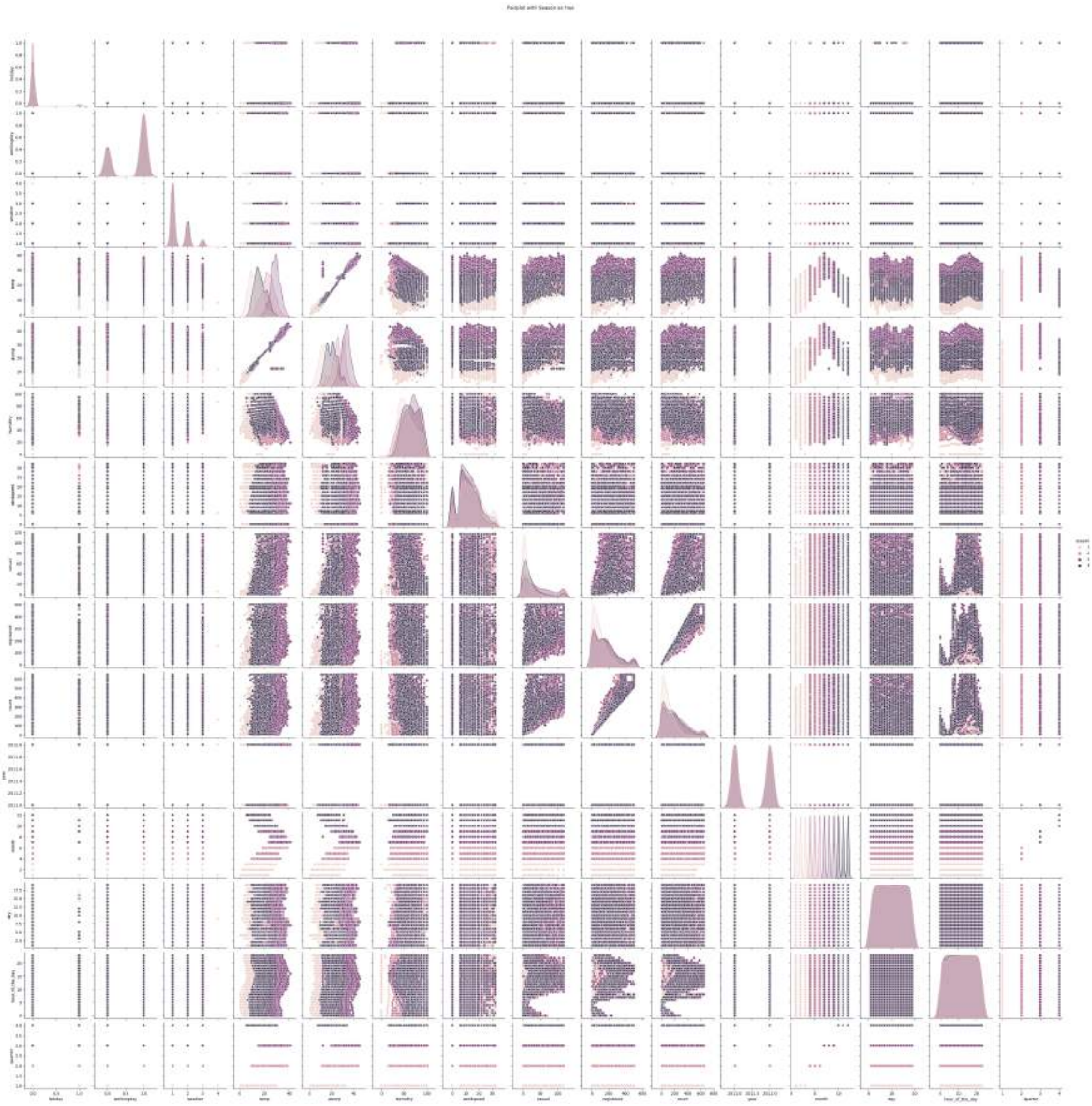


Observations: -

The pair plots of all integer columns are plotted above. The max correlation of season 4 is observed w.r.t humidity and registered users, etc.

In [34]:
```
# Create a pairplot with 'season' as hue
sns.pairplot(df_yulu, hue='season', diag_kind='kde', markers=['o', 's',
plt.suptitle('Pairplot with Season as Hue', y=1.02)
plt.show()
```

```
/Users/0438mpind/anaconda3/lib/python3.10/site-packages/seaborn/axisg
rid.py:118: UserWarning: The figure layout has changed to tight
  self._figure.tight_layout(*args, **kwargs)
```



Observations: -

Pair plots for all columns

In [35]:
```
# Checking the data correlation

corr_data = df_yulu.corr()
corr_data
```

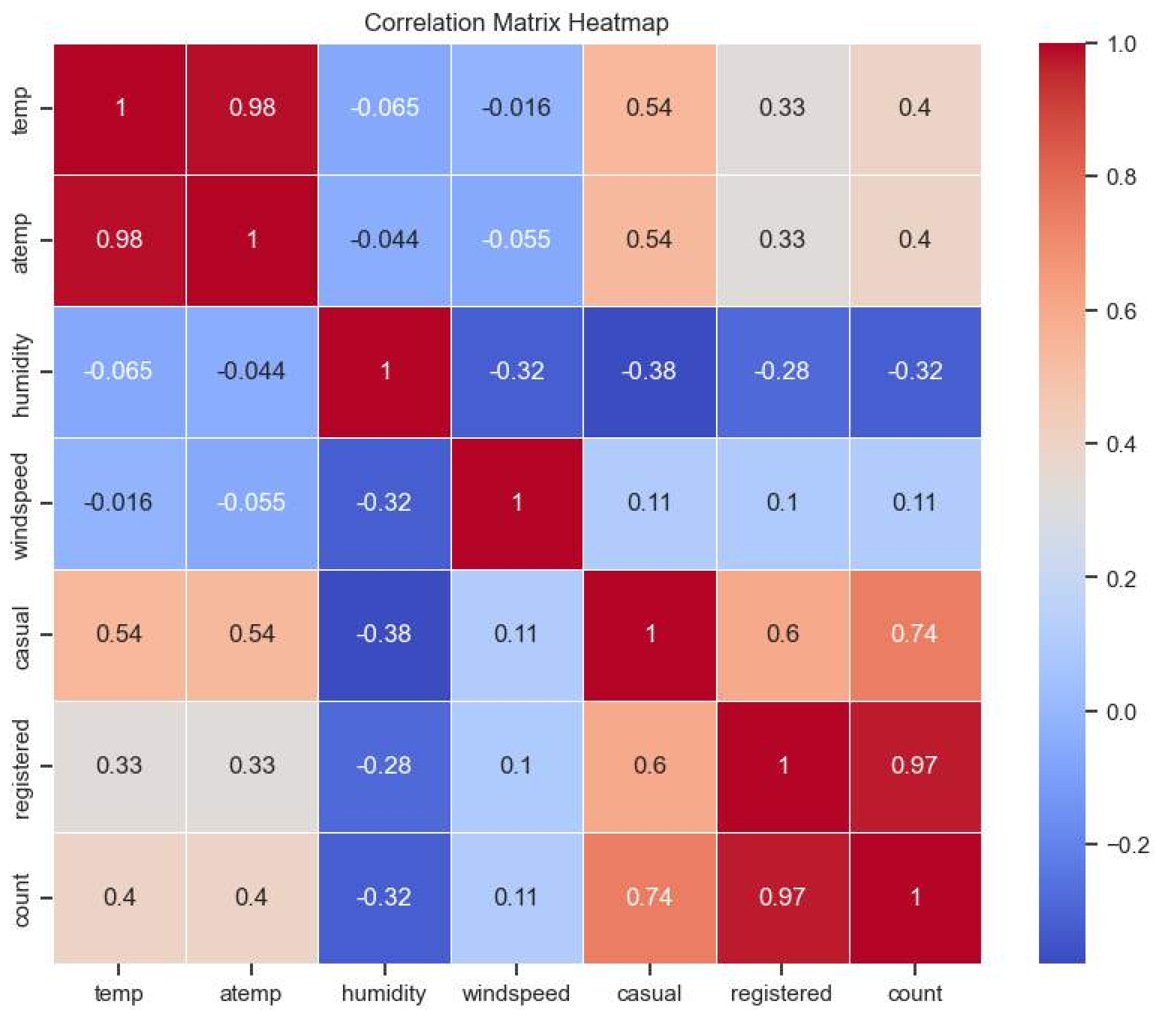
```
/var/folders/3x/g_8g6pcj3cv64hq5g80q28tw0000gp/T/ipykernel_36376/5834
79191.py:1: FutureWarning: The default value of numeric_only in DataF
rame.corr is deprecated. In a future version, it will default to Fals
e. Select only valid columns or specify the value of numeric_only to
silence this warning.
  corr_data = df_yulu.corr()
```

Out[35]:

	temp	atemp	humidity	windspeed	casual	registered	count
temp	1.000000	0.984948	-0.064949	-0.015521	0.542221	0.330598	0.399567
atemp	0.984948	1.000000	-0.043536	-0.055305	0.535456	0.326758	0.395062
humidity	-0.064949	-0.043536	1.000000	-0.320072	-0.377872	-0.282891	-0.323456
windspeed	-0.015521	-0.055305	-0.320072	1.000000	0.110620	0.103144	0.109054
casual	0.542221	0.535456	-0.377872	0.110620	1.000000	0.599660	0.744425
registered	0.330598	0.326758	-0.282891	0.103144	0.599660	1.000000	0.971975
count	0.399567	0.395062	-0.323456	0.109054	0.744425	0.971975	1.000000

In [41]:
```
# Heatmap of the correlation matrix for numeric variables

corr_matrix = df_yulu.corr()
plt.figure(figsize=(10, 8))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', linewidths=.5)
plt.title('Correlation Matrix Heatmap')
plt.show()
```

```
/var/folders/3x/g_8g6pcj3cv64hq5g80q28tw0000gp/T/ipykernel_36376/3054
630516.py:3: FutureWarning: The default value of numeric_only in Data
Frame.corr is deprecated. In a future version, it will default to Fal
se. Select only valid columns or specify the value of numeric_only to
silence this warning.
  corr_matrix = df_yulu.corr()
```

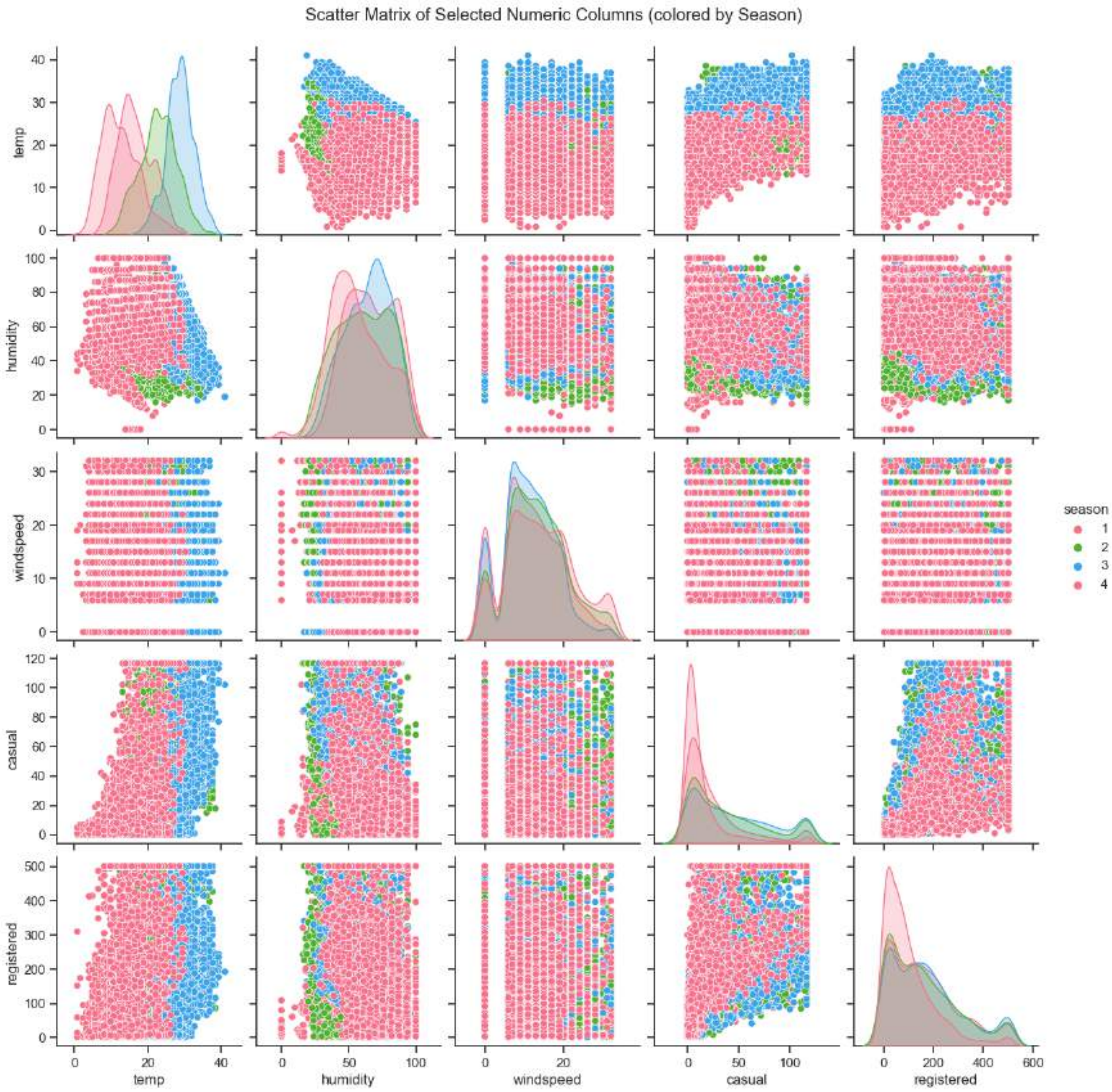


Observations

1. Very High Correlation (> 0.9) exists between columns [atemp, temp] and [count, registered]
2. High positively / negatively correlation (0.7 - 0.9) does not exist between any columns.
3. Moderate positive correlation (0.5 - 0.7) exists between columns [casual, count], [casual, registered].
4. Low Positive correlation (0.3 - 0.5) exists between columns [count, temp], [count, atemp], [casual, atemp]
5. Negligible correlation exists between all other combinations of columns.

In [37]:
```
# Scatter matrix of selected numeric columns

selected_columns = ['temp', 'humidity', 'windspeed', 'casual', 'registe
sns.set_theme(style="ticks")
sns.pairplot(df_yulu, vars=selected_columns, hue='season', palette='hus
plt.suptitle('Scatter Matrix of Selected Numeric Columns (colored by Se
plt.show()
```

```
/Users/0438mpind/anaconda3/lib/python3.10/site-packages/seaborn/axisg
rid.py:118: UserWarning: The figure layout has changed to tight
  self._figure.tight_layout(*args, **kwargs)
```



Observations: -

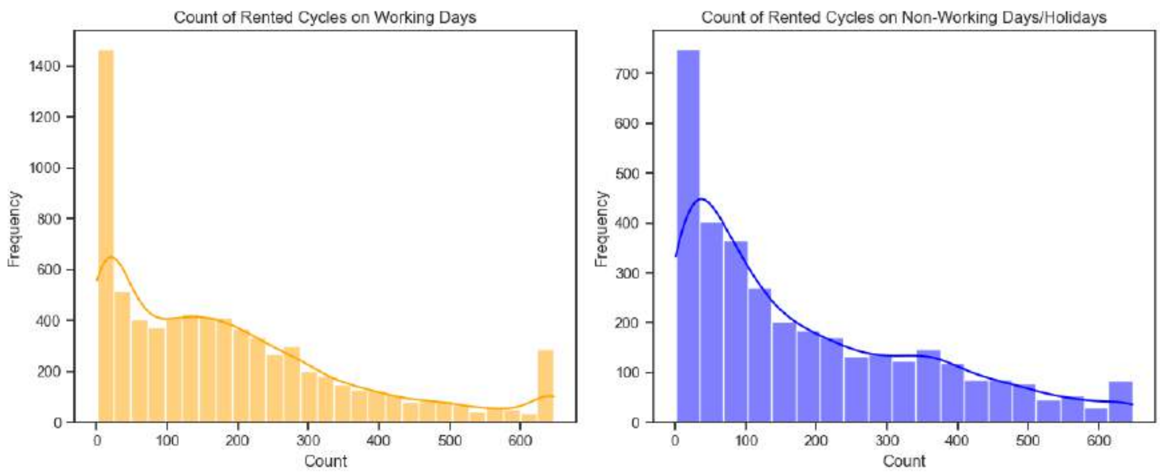
Another pair plot for plotting numeric data

Hypothesis Testing

Q. Whether Working Day/Holiday has an effect on the count of rented electric cycles

In [38]:
```
# Dataframe for count of renting bikes w.r.t working day as working_day
count_working_day = df_yulu[df_yulu['workingday'] == 1]['count']

# Dataframe for count of renting bikes w.r.t non-working days as nonwo
count_non_working_day = df_yulu[df_yulu['workingday'] == 0]['count']
```

In [84]:
```
# Plotting distribution plots for above

# Setting up the subplots
fig, axes = plt.subplots(1, 2, figsize=(12, 5))

# Creating normal distribution plots for working days and non-working d
sns.histplot(count_working_day, kde=True, ax=axes[0],color='orange')
axes[0].set_title('Count of Rented Cycles on Working Days')
axes[0].set_xlabel('Count')
axes[0].set_ylabel('Frequency')

sns.histplot(count_non_working_day, kde=True, ax=axes[1], color='blue')
axes[1].set_title('Count of Rented Cycles on Non-Working Days/Holidays
axes[1].set_xlabel('Count')
axes[1].set_ylabel('Frequency')

# Adjust layout
plt.tight_layout()
plt.show()
```



In [73]:
```
# Checking for the normal distribution with Wilkin-Shapiro Test [Workir

# Null Hypothesis(H0) : Data is distributed normally
# Alternate Hypothesis(Ha) : Data is not distributed normally

alpha = 0.05
workingday_smp = count_working_day.sample(100)
shapiro_stat,p_value = shapiro(workingday_smp)

print("Confidence Interval= 95%")
print("Wilkin-Shapiro test with Test Statistics: {}, and p-value: {}".
if p_value < alpha:
    print("Reject Ho: Data is not distributed normally")
else:
    print("Failed to reject Ho: Data is distributed normally")
```

```
Confidence Interval= 95%
Wilkin-Shapiro test with Test Statistics: 0.8483414649963379, and p-v
alue: 1.0303542019585166e-08
Reject Ho: Data is not distributed normally
```

In [74]:
```
# Checking for the normal distribution with Wilkin-Shapiro Test [Holida

# Null Hypothesis(H0) : Data is distributed normally
# Alternate Hypothesis(Ha) : Data is not distributed normally

nonworkingdays_smp = count_non_working_day.sample(100)
shapiro_stat,p_value = shapiro(nonworkingdays_smp)
alpha = 0.05

print("Confidence Interval= 95%")
print("Wilkin-Shapiro test with Test Statistics: {}, and p-value: {}".
if p_value < alpha:
    print("Reject Ho: Data is not distributed normally")
else:
    print("Failed to reject Ho: Data is distributed normally")
```

```
Confidence Interval= 95%
Wilkin-Shapiro test with Test Statistics: 0.8896470069885254, and p-v
alue: 4.755342501994164e-07
Reject Ho: Data is not distributed normally
```

In [46]:
```
# Checking for equal/similar variance among the two groups

print("Variance for Working Day group is : {}\n Variance for Non-Workir
print("The Ratio of the above two is : {}".format(np.var(count_working_
```

```
Variance for Working Day group is : 29774.454644861602
 Variance for Non-Working Day group is : 29617.640765896675
The Ratio of the above two is : 1.0052946107424428
```

Setting up Null and Alternate Hypothesis for above question

Null Hypothesis (H0) = Working Day doesn't have impact on the count of rented electric cycles,i.e., count of rented electric cycles is same on working day and on non-working day.

Alternate Hypothesis (Ha) = Working Day has an impact on the count of rented electric cycles,i.e., count of rented electric cycles is more on working day than on non-working day.

Significance Value (alpha) = 0.05

Since data is not distributed normally (above plots), 2 categories are here, with working day being the categorical data and Count being the numerical Also, the ratio between in variance

In [47]:

```
# Performing a two-sample t-test
t_stat, p_value = ttest_ind(count_working_day, count_non_working_day)

# Set the significance level (alpha)
alpha = 0.05

# Check if the p-value is less than alpha
if p_value < alpha:
    print("Reject the null hypothesis.")
    print("Working day has a significant effect on the number of electi
    print(f"(p-value = {p_value})")
else:
    print("Fail to reject the null hypothesis.")
    print("Working day does not have a significant effect on the numbe
    print(f"(p-value = {p_value})")
```

```
Fail to reject the null hypothesis.
Working day does not have a significant effect on the number of elect
ric cycles rented, i.e., count of rented electric cycles is same as o
n working day than on holidays.
(p-value = 0.7517611135576576)
```

Observations

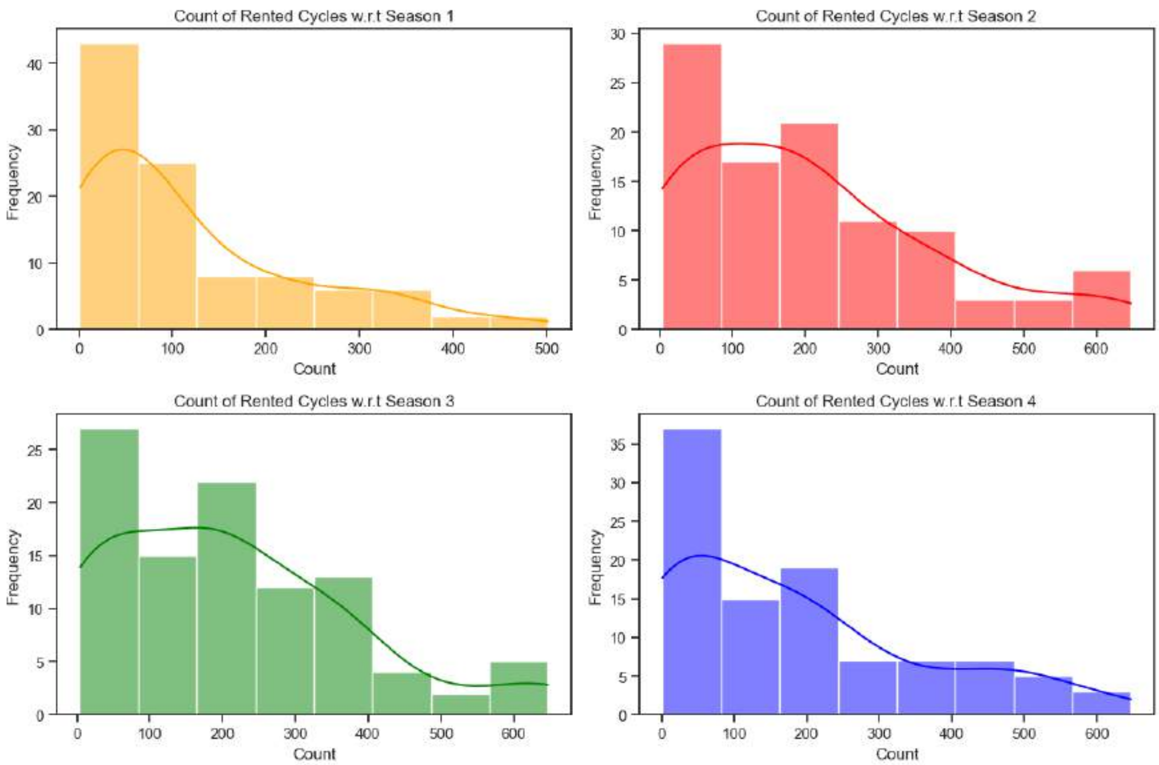
1. 2 different datasets are created for working days and non-working days(holidays). After this, we have checked the data for normality in the distribution using Wilkin-Shapiro test. Its observed that the data doesn't follow normal distribution.
2. Set up the Null and Alternate hypothesis, as these data have two categories and the data is independent from each other, therefore, used Two Sample Independent T-Test on the data and checked the result with 95% confidence interval.
3. Its observed that since p-value is less than the significance level of 0.05, we can reject the null hypothesis and can conclude that Working Day has more usage of rented electric cycles,i.e., count of rented electric cycles is more on working day than on holidays.

Q. ANNOVA to check if No. of cycles rented is similar or different in different 1. weather 2. season

Whether Season has an effect on the count of rented electric cycles

In [48]:
```
# Creating different categories for each different type of season based

season1_smps = df_yulu.loc[(df_yulu["season"]==1),"count"]
season2_smps = df_yulu.loc[(df_yulu["season"]==2),"count"]
season3_smps = df_yulu.loc[(df_yulu["season"]==3),"count"]
season4_smps = df_yulu.loc[(df_yulu["season"]==4),"count"]
```

In [88]:
```
# Plotting distribution plots for the different seasons

# Setting up the subplots
fig, axes = plt.subplots(2, 2, figsize=(12, 8))

# Creating normal distribution plots for the different seasons
sns.histplot(season1_smps, kde=True, ax=axes[0, 0], color='orange')
axes[0, 0].set_title('Count of Rented Cycles w.r.t Season 1')
axes[0, 0].set_xlabel('Count')
axes[0, 0].set_ylabel('Frequency')

sns.histplot(season2_smps, kde=True, ax=axes[0, 1], color='red')
axes[0, 1].set_title('Count of Rented Cycles w.r.t Season 2')
axes[0, 1].set_xlabel('Count')
axes[0, 1].set_ylabel('Frequency')

sns.histplot(season3_smps, kde=True, ax=axes[1, 0], color='green')
axes[1, 0].set_title('Count of Rented Cycles w.r.t Season 3')
axes[1, 0].set_xlabel('Count')
axes[1, 0].set_ylabel('Frequency')

sns.histplot(season4_smps, kde=True, ax=axes[1, 1], color='blue')
axes[1, 1].set_title('Count of Rented Cycles w.r.t Season 4')
axes[1, 1].set_xlabel('Count')
axes[1, 1].set_ylabel('Frequency')

# Adjust layout
plt.tight_layout()
plt.show()
```



In [89]:

```
# checking for the normal distribution with Wilkin-Shapiro Test [season

# Null Hypothesis(H0) : Data is distributed normally
# Alternate Hypothesis(Ha) : Data is not distributed normally

season1_smps = season1_smps.sample(100)
shapiro_stat,p_value = shapiro(season1_smps)
alpha = 0.05

print("Confidence Interval= 95%")
print("Wilkin-Shapiro test with Test Statistics: {}, and p-value: {}".
if p_value < alpha:
    print("Reject Ho: Data is not distributed normally")
else:
    print("Failed to reject Ho: Data is distributed normally")
```

```
Confidence Interval= 95%
Wilkin-Shapiro test with Test Statistics: 0.849433422088623, and p-va
lue: 1.130151972006388e-08
Reject Ho: Data is not distributed normally
```

In [90]:

```
# checking for the normal distribution with Wilkin-Shapiro Test [season

# Null Hypothesis(H0) : Data is distributed normally
# Alternate Hypothesis(Ha) : Data is not distributed normally

season2_smps = season2_smps.sample(100)
shapiro_stat,p_value = shapiro(season2_smps)
alpha = 0.05

print("Confidence Interval= 95%")
print("Wilkin-Shapiro test with Test Statistics: {}, and p-value: {}".
if p_value < alpha:
    print("Reject Ho: Data is not distributed normally")
else:
    print("Failed to reject Ho: Data is distributed normally")
```

```
Confidence Interval= 95%
Wilkin-Shapiro test with Test Statistics: 0.9117276072502136, and p-v
alue: 5.259171757643344e-06
Reject Ho: Data is not distributed normally
```

In [91]:
```
# checking for the normal distribution with Wilkin-Shapiro Test [seasor

# Null Hypothesis(H0) : Data is distributed normally
# Alternate Hypothesis(Ha) : Data is not distributed normally

season3_smps = season3_smps.sample(100)
shapiro_stat,p_value = shapiro(season3_smps)
alpha = 0.05

print("Confidence Interval= 95%")
print("Wilkin-Shapiro test with Test Statistics: {}, and p-value: {}".
if p_value < alpha:
    print("Reject Ho: Data is not distributed normally")
else:
    print("Failed to reject Ho: Data is distributed normally")
```

```
Confidence Interval= 95%
Wilkin-Shapiro test with Test Statistics: 0.9155954122543335, and p-v
alue: 8.280203473987058e-06
Reject Ho: Data is not distributed normally
```

In [92]:
```
# checking for the normal distribution with Wilkin-Shapiro Test [seasor

# Null Hypothesis(H0) : Data is distributed normally
# Alternate Hypothesis(Ha) : Data is not distributed normally

season4_smps = season4_smps.sample(100)
shapiro_stat,p_value = shapiro(season4_smps)
alpha = 0.05

print("Confidence Interval= 95%")
print("Wilkin-Shapiro test with Test Statistics: {}, and p-value: {}".
if p_value < alpha:
    print("Reject Ho: Data is not distributed normally")
else:
    print("Failed to reject Ho: Data is distributed normally")
```

```
Confidence Interval= 95%
Wilkin-Shapiro test with Test Statistics: 0.8788182139396667, and p-v
alue: 1.6179961903617368e-07
Reject Ho: Data is not distributed normally
```

In [53]:
```
# Checking for equal variance among the different groups with Levene's

# Null Hypothesis(H0) : Variance among the groups are equal.
# Alternate Hypothesis(Ha) : Variance among the groups are not equal.

alpha = 0.05
levene_stat,p_value = levene(season1_smps,season2_smps,season3_smps,sea

print("Confidence Interval= 95%")
print("Levene test with Test Statistic: {}, and p-value: {}".format(lev
if p_value < alpha:
    print("Reject Ho: Variance among the groups are not equal")
else:
    print("Failed to reject Ho: Variance among the groups are equal")
```

```
Confidence Interval= 95%
Levene test with Test Statistic: 4.755535415374494, and p-value: 0.00
2860238426954299
Reject Ho: Variance among the groups are not equal
```

Setting up Null and Alternate Hypothesis for above question

Null Hypothesis (H0) = Season doesn't have impact on the count of rented electric cycles,i.e., No. of cycles are same in different seasons.

Alternate Hypothesis (Ha) = Season does have impact on the count of rented electric cycles,i.e., No. of cycles are different in different seasons.

Significance Value = 0.05

Data is not distributed normally, and variance among the groups are not equal, this violates the assumption of ANOVA. Also, we have more than two categories here, with season being the categorical data and Count being the numerical. Thus applying Kruskal-Wallis test for same.

In [54]:
```
#  Implementation of Kruskal-Wallis Test

test_statistic,p_value = kruskal(season1_smps,season2_smps,season3_smps

alpha = 0.05
print("Kruskal-Wallis Test with Test Statistics: {}, and p-value: {}".

print("Confidence Interval: 95%")
if p_value < alpha:
    print("Reject Ho: Season does have impact on the count of rented e
else:
    print("Failed to reject Ho: Season doesn't have impact on the count
```

```
Kruskal-Wallis Test with Test Statistics: 20.046051288787385, and p-v
alue: 0.00016605282486259227
Confidence Interval: 95%
Reject Ho: Season does have impact on the count of rented electric cy
cles,i.e., No. of cycles are different in different seasons.
```

In [55]:
```
#  Implementation of ANOVA (f_oneway) Test

test_statistic,p_value = f_oneway(season1_smps,season2_smps,season3_smp
print("ANOVA with Test Statistics: {}, and p-value: {}".format(test_sta
alpha = 0.05
print("Confidence Interval: 95%")
if p_value < alpha:
    print("Reject Ho: Season does have impact on the count of rented e
else:
    print("Failed to reject Ho: Season doesn't have impact on the count
```

```
ANOVA with Test Statistics: 6.7676998183168315, and p-value: 0.000184
55583004407485
Confidence Interval: 95%
Reject Ho: Season does have impact on the count of rented electric cy
cles,i.e., No. of cycles are different in different seasons.
```

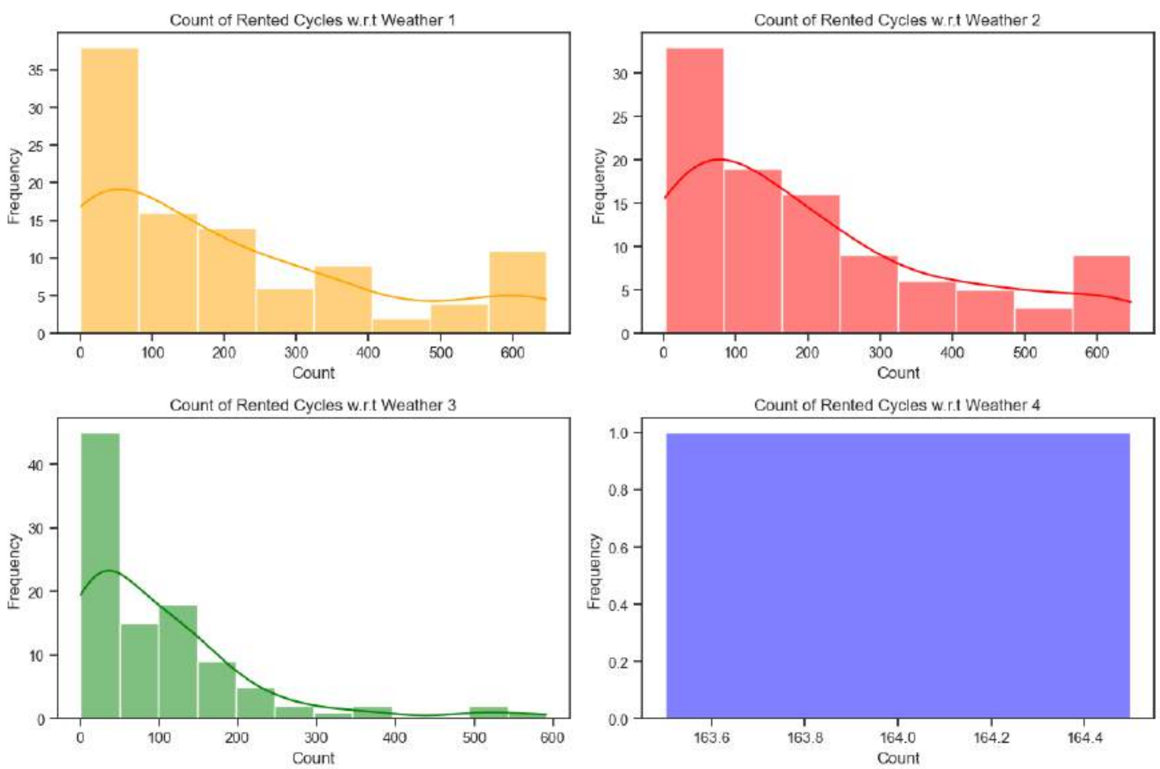
Observations:

1. 4 different datasets for each season. After this, we have checked the data for normality in the distribution using Wilkin-Shapiro test, unfortunately, the data doesn't follow normal distribution. We also checked for the assumption for equal variance among these groups using Levene's test. We found that variance among the groups are not equal.
2. Set-up of the Null and Alternate hypothesis, as these data have more than two categories and the data is not normally distributed for each seasons and variance among the groups are also not equal, this violates the assumptions of ANOVA. Therefore, we have used Kruskal-Wallis Test on the data and checked the result with 95% confidence interval. We also used ANOVA (f_oneway) on the data.
3. We found in both the tests that: Since p-value is less than the significance level of 0.05, we can reject the null hypothesis and can conclude that Season does have impact on the count of rented electric cycles,i.e., No. of cycles is different in different seasons.

Whether Weather has an effect on the count of rented electric cycles

In [56]:
```
# Creating different categories for each different type of weather base

weather1_smps = df_yulu.loc[(df_yulu["weather"]==1),"count"]
weather2_smps = df_yulu.loc[(df_yulu["weather"]==2),"count"]
weather3_smps = df_yulu.loc[(df_yulu["weather"]==3),"count"]
weather4_smps = df_yulu.loc[(df_yulu["weather"]==4),"count"]
```

In [93]:
```
# Plotting distribution plots for the different weather

# Setting up the subplots
fig, axes = plt.subplots(2, 2, figsize=(12, 8))

# Creating normal distribution plots for the different seasons
sns.histplot(weather1_smps, kde=True, ax=axes[0, 0], color='orange')
axes[0, 0].set_title('Count of Rented Cycles w.r.t Weather 1')
axes[0, 0].set_xlabel('Count')
axes[0, 0].set_ylabel('Frequency')

sns.histplot(weather2_smps, kde=True, ax=axes[0, 1], color='red')
axes[0, 1].set_title('Count of Rented Cycles w.r.t Weather 2')
axes[0, 1].set_xlabel('Count')
axes[0, 1].set_ylabel('Frequency')

sns.histplot(weather3_smps, kde=True, ax=axes[1, 0], color='green')
axes[1, 0].set_title('Count of Rented Cycles w.r.t Weather 3')
axes[1, 0].set_xlabel('Count')
axes[1, 0].set_ylabel('Frequency')

sns.histplot(weather4_smps, kde=True, ax=axes[1, 1], color='blue')
axes[1, 1].set_title('Count of Rented Cycles w.r.t Weather 4')
axes[1, 1].set_xlabel('Count')
axes[1, 1].set_ylabel('Frequency')
# No proper distribution for plot 4 since only 1 datapoint is present

# Adjust layout
plt.tight_layout()
plt.show()
```



In [94]:
```
# checking for the normal distribution with Wilkin-Shapiro Test [weathe

# Null Hypothesis(H0) : Data is distributed normally
# Alternate Hypothesis(Ha) : Data is not distributed normally

weather1_smps = weather1_smps.sample(100)
shapiro_stat,p_value = shapiro(weather1_smps)
alpha = 0.05

print("Confidence Interval= 95%")
print("Wilkin-Shapiro test with Test Statistics: {}, and p-value: {}".
if p_value < alpha:
    print("Reject Ho: Data is not distributed normally")
else:
    print("Failed to reject Ho: Data is distributed normally")
```

```
Confidence Interval= 95%
Wilkin-Shapiro test with Test Statistics: 0.8514386415481567, and p-v
alue: 1.3407565724321557e-08
Reject Ho: Data is not distributed normally
```

In [95]:
```
# checking for the normal distribution with Wilkin-Shapiro Test [weathe

# Null Hypothesis(H0) : Data is distributed normally
# Alternate Hypothesis(Ha) : Data is not distributed normally

weather2_smps = weather2_smps.sample(100)
shapiro_stat,p_value = shapiro(weather2_smps)
alpha = 0.05

print("Confidence Interval= 95%")
print("Wilkin-Shapiro test with Test Statistics: {}, and p-value: {}".
if p_value < alpha:
    print("Reject Ho: Data is not distributed normally")
else:
    print("Failed to reject Ho: Data is distributed normally")
```

```
Confidence Interval= 95%
Wilkin-Shapiro test with Test Statistics: 0.8781110048294067, and p-v
alue: 1.5110263973383553e-07
Reject Ho: Data is not distributed normally
```

In [96]:
```
# checking for the normal distribution with Wilkin-Shapiro Test [weathe

# Null Hypothesis(H0) : Data is distributed normally
# Alternate Hypothesis(Ha) : Data is not distributed normally

weather3_smps = weather3_smps.sample(100)
shapiro_stat,p_value = shapiro(weather3_smps)
alpha = 0.05

print("Confidence Interval= 95%")
print("Wilkin-Shapiro test with Test Statistics: {}, and p-value: {}".
if p_value < alpha:
    print("Reject Ho: Data is not distributed normally")
else:
    print("Failed to reject Ho: Data is distributed normally")
```

```
Confidence Interval= 95%
Wilkin-Shapiro test with Test Statistics: 0.7768545150756836, and p-v
alue: 5.1138940304618075e-11
Reject Ho: Data is not distributed normally
```

In [60]:
```
# Checking for equal variance among different groups with Levene's Test

# Null Hypothesis(H0) : Variance among the groups are equal.
# Alternate Hypothesis(Ha) : Variance among the groups are not equal.

alpha = 0.05
levene_stat,p_value = levene(weather1_smps,weather2_smps,weather3_smps,

print("Confidence Interval: 95%")
print("Levene test with Test Statistic: {}, and p-value: {}".format(lev
if p_value < alpha:
    print("Reject Ho: Variance among the groups are not equal")
else:
    print("Failed to reject Ho: Variance among the groups are equal")
```

```
Confidence Interval: 95%
Levene test with Test Statistic: 9.447580126113285, and p-value: 5.56
7145134828813e-06
Reject Ho: Variance among the groups are not equal
```

Setting up Null and Alternate Hypothesis

Null Hypothesis (H0) = Weather doesn't have impact on the count of rented electric cycles,i.e., No. of cycles are same in different weather.

Alternate Hypothesis (Ha) = Weather does have impact on the count of rented electric cycles,i.e., No. of cycles are different in different weather.

Significance Value = 0.05

Also, weather 4 has only one data point, so its not able to perform wilkin-shapiro test on it.

Data is not distributed normally, and variance among the groups are not equal, this violates the assumption of ANOVA. Moreover, we have more than two categories here, with season being the categorical data and count being the numerical. Thus applying Kruskal-Wallis test.

In [61]:
```
# Implementation of Kruskal-Wallis Test

alpha = 0.05
test_statistic,p_value = kruskal(weather1_smps,weather2_smps,weather3_s
print("Kruskal-Wallis Test with Test Statistics: {}, and p-value: {}".

print("Confidence Interval= 95%")
if p_value < alpha:
    print("Reject Ho: Weather does have impact on the count of rented e
else:
    print("Failed to reject Ho: Weather doesn't have impact on the cour
```

```
Kruskal-Wallis Test with Test Statistics: 18.546331617290353, and p-v
alue: 0.00033927153180920495
Confidence Interval= 95%
Reject Ho: Weather does have impact on the count of rented electric c
ycles,i.e., No. of cycles is different in different weather.
```

In [62]:
```
# Implementation of ANOVA(f_oneway)

test_statistic,p_value = f_oneway(weather1_smps,weather2_smps,weather3_
print("Kruskal-Wallis Test with Test Statistics: {}, and p-value: {}".
alpha = 0.05
print("Confidence Interval= 95%")
if p_value < alpha:
    print("Reject Ho: Weather does have impact on the count of rented e
else:
    print("Failed to reject Ho: Weather doesn't have impact on the cour
```

```
Kruskal-Wallis Test with Test Statistics: 7.973475018053148, and p-va
lue: 3.953231010727268e-05
Confidence Interval= 95%
Reject Ho: Weather does have impact on the count of rented electric c
ycles,i.e., No. of cycles is different in different weather.
```

Observations

1. 4 different datasets for each weather category. After this, checked the data for normality in the distribution using Wilkin-Shapiro test, unfortunately, the data doesn't follow normal distribution. We also checked for the assumption for equal variance among these groups using Levene's test. We found that variance among the groups are not equal.
2. Set-up of the Null and Alternate hypothesis, as these data have more than two categories and the data is not normally distributed for each weather and variance among the groups are also not equal, this violates the assumptions of ANOVA. Therefore, we have used Kruskal-Wallis Test on the data and checked the result with 95% confidence interval. We also used ANOVA (f_oneway) on the data.
3. We found in both the tests that: Since p-value is less than the significance level of 0.05, we can reject the null hypothesis and can conclude that Weather does have impact on the count of rented electric cycles,i.e., No. of cycles is different in different weather.

Q. Whether Weather is dependent on the season?

In [63]:
```
# Creating a separate dataframe consisting of season,weather and corres
weather_df = df_yulu[["season","weather","count"]]
weather_df = pd.crosstab(index=weather_df['season'],columns=weather_df
data=weather_df.values
```

Setting up Null and Alternate Hypothesis

Null Hypothesis (H0) = Weather doesn't depend on the season.

Alternate Hypothesis (Ha) = Weather does depend on the season.

Significance Value = 0.05

We have two categories to compare here for the test of independence, thus applying chisquare test(chi2_contingency).

In [64]:
```
# Implementation of Chi-Square Test of Independence

test_statistic,p_value,dof,exp_freq = chi2_contingency(data)
print("Chi-Square Test of Independence with Test Statistic: {}, p-value
alpha = 0.05
print("Confidence Interval: 95%")
if p_value < alpha:
    print("Reject Ho: Weather does depend on the season.")
else:
    print("Failed to reject Ho: Weather doesn't depend on the season.")
```

```
Chi-Square Test of Independence with Test Statistic: 660.273112912428
7, p-value: 2.3873348201023556e-136, and Degree of Freedom: 9
Confidence Interval: 95%
Reject Ho: Weather does depend on the season.
```

Observations

1. Created a dataset of season and weather with their corresponding count of rented cycles. After this, used crosstab to create a dataframe with season in row axis and weather on column axis, and put the rows in a 2D numpy array.
2. Set-up of the Null and Alternate hypothesis, as these data have two categories to compare for dependency, therefore, used Chi-Square test of independence on the derived data and checked the result with 95% confidence interval.
3. Its found that: Since p-value is less than the significance level of 0.05, we can reject the null hypothesis and can conclude that Weather does depend on the season.

In [65]:
```
# Perform ANOVA for 'weather'
weather_categories = df_yulu['weather'].unique()
weather_data = [df_yulu[df_yulu['weather'] == cat]['count'] for cat in

# Perform the ANOVA test
weather_anova = stats.f_oneway(*weather_data)

# Perform ANOVA for 'season'
season_categories = df_yulu['season'].unique()
season_data = [df_yulu[df_yulu['season'] == cat]['count'] for cat in se

# Perform the ANOVA test
season_anova = stats.f_oneway(*season_data)

# Set the significance level (alpha)
alpha = 0.05

# Check the p-values for both tests
print("Case 1:- Weather")
if weather_anova.pvalue < alpha:
    print(f"Reject the null hypothesis for weather. Number of cycles re
else:
    print(f"Fail to reject the null hypothesis for weather. Number of c

print("\n")
print("Case 2:- Season")
if season_anova.pvalue < alpha:
    print(f"Reject the null hypothesis for season. Number of cycles ren
else:
    print(f"Fail to reject the null hypothesis for season. Number of cy
```

```
Case 1:- Weather
Reject the null hypothesis for weather. Number of cycles rented is di
fferent for different weather categories (p-value = 8.034967610817961
e-44)


Case 2:- Season
Reject the null hypothesis for season. Number of cycles rented is dif
ferent for different season categories (p-value = 7.771506553957677e-
153)

/var/folders/3x/g_8g6pcj3cv64hq5g80q28tw0000gp/T/ipykernel_36376/1029
627645.py:6: DeprecationWarning: Please use `f_oneway` from the `scip
y.stats` namespace, the `scipy.stats.stats` namespace is deprecated.
  weather_anova = stats.f_oneway(*weather_data)
/var/folders/3x/g_8g6pcj3cv64hq5g80q28tw0000gp/T/ipykernel_36376/1029
627645.py:13: DeprecationWarning: Please use `f_oneway` from the `sci
py.stats` namespace, the `scipy.stats.stats` namespace is deprecated.
  season_anova = stats.f_oneway(*season_data)
```

Q. Chi-square test to check if Weather is dependent on the season

In [66]:
```
# Creating a contingency table of observed frequencies
cont_tbl = pd.crosstab(df_yulu['season'], df_yulu['weather'])
cont_tbl
```

Out[66]:

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

In [67]:
```
# Creating a separate dataframe consisting of season,weather and corres
weather_sm = df_yulu[["season","weather","count"]]
weather_sm = pd.crosstab(index=weather_sm['season'],columns=weather_sm
data=weather_sm.values
```

Setting up Null and Alternate Hypothesis

Null Hypothesis (H0) = Weather doesn't depend on the season.

Alternate Hypothesis (Ha) = Weather does depend on the season.

Significance Value = 0.05

We have two categories to compare here for the test of independence, thus applying chisquare test(chi2_contingency).

In [68]:
```
# Implementation of Chi-Square Test of Independence

alpha = 0.05
test_statistic,p_value,dof,exp_freq = chi2_contingency(data)
print("Chi-Square Test of Independence with Test Statistics: {}, p-valu

print("Confidence Interval: 95%")

if p_value < alpha:
    print("Reject Ho: Weather does depend on the season.")
else:
    print("Failed to reject Ho: Weather doesn't depend on the season.")
```

```
Chi-Square Test of Independence with Test Statistics: 660.27311291242
87, p-value: 2.3873348201023556e-136, and Degree of Freedom: 9
Confidence Interval: 95%
Reject Ho: Weather does depend on the season.
```

Observations

1. We have created a dataset of season and weather with their corresponding count of rented cycles. After this, we have used crosstab to create a dataframe with season in row axis and weather on column axis, and put the rows in a 2D numpy array.
2. Set-up of the Null and Alternate hypothesis, as these data have two categories to compare for dependency, therefore, we have used Chi-Square test of independence on the derived data and checked the result with 95% confidence interval.
3. We found that: Since p-value is less than the significance level of 0.05, we can reject the null hypothesis and can conclude that Weather does depend on the season.

Business Insights

1. Electric cycle rentals are most popular among customers when the weather is clear or cloudy, primarily from January to August. Businesses can capitalize on this trend by ensuring cycle availability during these specific periods or seasons.
2. It has been noticed that cycle rentals decrease significantly during rainy, stormy, snowy, or foggy weather conditions. Additionally, cycle rentals are extremely low when humidity drops below 20.
3. Likewise, when the temperature falls below 10 degrees, there is a drop in the number of cycle rentals, and when the wind speed exceeds 35, cycle rentals also decrease.
4. Registered users and casual riders exhibit distinct rental preferences. Registered users are more inclined to rent during regular workdays, particularly during office hours, while casual riders show a preference for renting on holidays. This differentiation can guide service providers in allocating more cycle during peak hours to cater to varied consumer needs.
5. The number of registered riders significantly surpasses that of casual riders, offering valuable insights into customer retention and the quality of services provided by the business.
6. Through hypothesis testing, it can be concluded that weather is influenced by the season, a fact that is readily apparent. Additionally, both weather and season have an impact on the quantity of cycle rented during specific periods.
7. Peak hours for cycle rentals tend to coincide with typical working hours and commuting patterns. This implies that businesses can enhance their cycle availability during the morning and evening rush hours to serve the commuter demographic effectively.
8. Among 100 users, approximately 19 fall into the category of casual users, while the remaining 81 are registered users.
9. The average hourly cycle rental count is 144 for the year 2011 and 239 for the year 2012. This indicates an annual growth rate of 65.41% in the demand for electric vehicles on an hourly basis.
10. The count of rental cycle exhibits a seasonal trend, with elevated demand during spring and summer, a mild decrease in the autumn, and a more significant drop during the winter months.
11. Throughout the day, there is a noticeable fluctuation in cycle counts, characterized by low numbers in the early morning, a sudden surge in the morning, a peak count in the afternoon, and a gradual decrease in the evening and night time.
12. Humidity levels exceed 40 for more than 80% of the time, indicating that humidity mostly fluctuates between optimal and excessively moist conditions.
13. The average hourly cycle rental count exhibits no statistically significant difference between working and non-working days.

14. Hourly total cycle rentals exhibit statistical disparities across various weather conditions.
15. There are statistically significant variations in the hourly total number of cycle rentals among different seasons.

Recommendations

1. In summer and fall seasons, during clear or cloudy weather, the company should have more cycles in stock to be rented, as the demand during these seasons is higher as compared to other seasons.
2. With a significance level of 0.05, working day does have effect on the number of cycles being rented. Therefore, it will be wise to put more cycles on roads to cater the needs of consumers (for registered ones). A nominal count of cycles available during holidays will cater the needs of casual riders.
3. As mentioned above, days when temperature is less than 10, company should take out cycles from roads for maintenance, this will provide ample time for repairs.
4. Similarly, days when windspeed is greater than 35 or in thunderstorms, company should take out cycles from roads for maintenance.
5. Dynamic Pricing: Implement dynamic pricing strategies that adjust rental rates based on factors such as weather conditions, peak hours, and demand. Offering discounts during unfavorable weather or off-peak hours can attract more riders.
6. Weather-Responsive Promotions: Launch weather-responsive promotions to incentivize riders during inclement weather. For example, offer reduced rates or special deals during rainy or hot seasons to encourage ridership.
7. Seasonal Marketing Campaigns: Develop seasonal marketing campaigns that align with the patterns of ridership. Tailor advertising and promotions to coincide with the peak seasons in each region.
8. Fleet Optimization: Analyze data to optimize the distribution and maintenance of the bicycle fleet. Ensure that cycles are readily available during peak hours and seasons, and conduct preventative maintenance during off-peak times.
9. User Segmentation: Implement user-specific strategies. For casual users, focus on ease of access and user-friendly interfaces. For registered users, introduce loyalty programs and incentives for frequent usage.
10. Geographic Expansion: Utilize data to identify underserved areas with high demand for Yulu services. Plan geographic expansion into these regions, considering demographic and traffic patterns.
11. Public-Private Partnerships: Collaborate with local governments, transportation authorities, and private businesses to integrate Yulu services into existing transportation networks, making it convenient for commuters.
12. Safety Initiatives: Invest in safety measures, including safety training for riders, enhanced security for bicycles, and collaboration with local authorities to ensure safe riding conditions.
13. Feedback Mechanism: Implement a robust feedback system to gather insights from users about their experiences and expectations. Act on this feedback to improve services continually.
14. Sustainability Promotion: Promote the environmental benefits of cycling as a sustainable mode of transportation. Highlight the positive impact on reducing carbon emissions and air pollution.
15. Data-Driven Decision Making: Establish a data-driven culture within the organization. Use advanced analytics and machine learning to predict demand, optimize operations, and improve decision-making processes.

16. Community Building: Foster a community around Yulu services by organizing cycling events, community rides, and partnerships with local cycling clubs. Engage users in the brand and promote a sense of belonging.
17. Accessibility Enhancements: Improve accessibility for riders with disabilities. Ensure that Yulu services are inclusive and compliant with accessibility standards.
18. Collaborative Advertising: Collaborate with local businesses, tourism boards, and other organizations for co-marketing opportunities. Leverage partnerships to expand the reach of promotional campaigns.
19. Improve Weather Data Collection: Given the lack of records for extreme weather conditions, consider improving the data collection process for such scenarios. Having more data on extreme weather conditions can help to understand customer behavior and adjust the operations accordingly, such as offering specialized bike models for different weather conditions or implementing safety measures during extreme weather.
20. R&D and Innovation: Invest in research and development to innovate and offer new features and services, such as smart bike locks, IoT integration, and mobile app enhancements.
21. Special Occasion Discounts: Since Yulu focusses on providing a sustainable solution for vehicular pollution, it should give special discounts on the occassions like Zero Emissions Day (21st September), Earth day (22nd April), World Environment Day (5th June) etc in order to attract new users.
22. Weather-based Promotions: Recognize the impact of weather on bike rentals. Create weather-based promotions that target customers during clear and cloudy weather, as these conditions show the highest rental counts. Yulu can offer weather-specific discounts to attract more customers during these favorable weather conditions

In [ ]:

```
------EOF----
```

In [ ]:

```

```