yulu-hypothesis-testing

March 1, 2024

#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!

#Business Problem

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.

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?

[129]:
```
!pip install matplotlib
```

```
Requirement already satisfied: matplotlib in /usr/local/lib/python3.10/dist-
packages (3.7.1)
Requirement already satisfied: contourpy>=1.0.1 in
/usr/local/lib/python3.10/dist-packages (from matplotlib) (1.2.0)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-
packages (from matplotlib) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in
/usr/local/lib/python3.10/dist-packages (from matplotlib) (4.49.0)
Requirement already satisfied: kiwisolver>=1.0.1 in
/usr/local/lib/python3.10/dist-packages (from matplotlib) (1.4.5)
Requirement already satisfied: numpy>=1.20 in /usr/local/lib/python3.10/dist-
packages (from matplotlib) (1.25.2)
Requirement already satisfied: packaging>=20.0 in
/usr/local/lib/python3.10/dist-packages (from matplotlib) (23.2)
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.10/dist-
```

```
packages (from matplotlib) (9.4.0)
Requirement already satisfied: pyparsing>=2.3.1 in
/usr/local/lib/python3.10/dist-packages (from matplotlib) (3.1.1)
Requirement already satisfied: python-dateutil>=2.7 in
/usr/local/lib/python3.10/dist-packages (from matplotlib) (2.8.2)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-
packages (from python-dateutil>=2.7->matplotlib) (1.16.0)
```

[2]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import random
```

[4]:
```
yulu = pd.read_csv('bike_sharing.csv') ##hiding data path to maintain data␣
 ↪privacy.
yulu.head()
```

[4]:
```
              datetime  season  holiday  workingday  weather  temp   atemp  \
0  2011-01-01 00:00:00       1        0           0        1  9.84  14.395
1  2011-01-01 01:00:00       1        0           0        1  9.02  13.635
2  2011-01-01 02:00:00       1        0           0        1  9.02  13.635
3  2011-01-01 03:00:00       1        0           0        1  9.84  14.395
4  2011-01-01 04:00:00       1        0           0        1  9.84  14.395

   humidity  windspeed  casual  registered  count
0        81        0.0       3          13     16
1        80        0.0       8          32     40
2        80        0.0       5          27     32
3        75        0.0       3          10     13
4        75        0.0       0           1      1
```

#DataSet

The company collected hourly data for a period of 2 years.

If we look at the data, we can see the data in these columns:

1. Season -> change on monthly basis.
2. Holiday, workingday -> change on daily basis.
3. Datetime, weather, temp, atemp, humidity, windspeed, casual, registered and count -> change on an hourly basis.

#Features

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)
4. workingday: if day is neither weekend nor holiday, then 1, otherwise is 0.

5. weather:

- 1 -> Clear, Few clouds, partly cloudy, partly cloudy
- 2 -> Mist + Cloudy, Mist + Broken clouds, Mist + Few clouds, Mist
- 3 -> Light Snow, Light Rain + Thunderstorm + Scattered clouds, Light Rain + Scattered clouds
- 4 -> Heavy Rain + Ice Pallets + Thunderstorm + Mist, Snow + Fog

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

[130]:
```
yulu.shape
```

[130]:
```
(10886, 12)
```

[131]:
```
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
```

[ ]:
```
yulu.isna().sum()
```

[ ]:
```
datetime      0
season        0
holiday       0
workingday    0
weather       0
```

```
temp          0
atemp         0
humidity      0
windspeed     0
casual        0
registered    0
count         0
dtype: int64
```

So, as we can see, there are no nulls in our dataset, which is good. Now we will be checking what kind of data we have in our dataframe in the individual features. It will help us have a sense of the data we are dealing with.

```
[ ]: for column in yulu:
       print('\033[1;4m' + column + '\033[0m', end= ':- No. of Unique Values: ')
       print(yulu[column].nunique(), end= ';  Unique Values: ')
       print(yulu[column].unique())
       print()
```

```
datetime:- No. of Unique Values: 10886;  Unique Values: ['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']

season:- No. of Unique Values: 4;  Unique Values: [1 2 3 4]

holiday:- No. of Unique Values: 2;  Unique Values: [0 1]

workingday:- No. of Unique Values: 2;  Unique Values: [0 1]

weather:- No. of Unique Values: 4;  Unique Values: [1 2 3 4]

temp:- No. of Unique Values: 49;  Unique Values: [ 9.84  9.02  8.2
13.12 15.58 14.76 17.22 18.86 18.04 16.4  13.94 12.3
 10.66  6.56  5.74  7.38  4.92 11.48  4.1   3.28  2.46 21.32 22.96 23.78
 24.6  19.68 22.14 20.5  27.06 26.24 25.42 27.88 28.7  30.34 31.16 29.52
 33.62 35.26 36.9  32.8  31.98 34.44 36.08 37.72 38.54  1.64  0.82 39.36
 41.  ]

atemp:- No. of Unique Values: 60;  Unique Values: [14.395 13.635 12.88
17.425 19.695 16.665 21.21  22.725 21.97  20.455
 11.365 10.605  9.85   8.335  6.82   5.305  6.06   9.09  12.12   7.575
 15.91   3.03   3.79   4.545 15.15  18.18  25.    26.515 27.275 29.545
 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  36.365
 37.88  42.425 43.94  38.635  1.515  0.76   2.275 43.18  44.695 45.455]

humidity:- No. of Unique Values: 89;  Unique Values: [ 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]
```

windspeed:- No. of Unique Values: 28; Unique Values: [0.   6.0032

```
16.9979 19.0012 19.9995 12.998  15.0013  8.9981 11.0014
 22.0028 30.0026 23.9994 27.9993 26.0027  7.0015 32.9975 36.9974 31.0009
 35.0008 39.0007 43.9989 40.9973 51.9987 46.0022 50.0021 43.0006 56.9969
 47.9988]
```

casual:- No. of Unique Values: 309; Unique Values: [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
 150 188 193 180 127 154 108  96 110 112 169 131 176 134 162 153 210 118
 141 146 159 178 177 136 215 198 248 225 194 237 242 235 224 236 222  77
  87 101 145 182 171 160 133 105 104 187 221 201 205 234 185 164 200 130
 155 116 125 204 186 214 245 218 217 152 191 256 251 262 189 212 272 223
 208 165 229 151 117 199 140 226 286 352 357 367 291 233 190 283 295 232
 173 184 172 320 355 326 321 354 299 227 254 260 207 274 308 288 311 253
 197 163 275 298 282 266 220 241 230 157 293 257 269 255 228 276 332 361
 356 331 279 203 250 259 297 265 267 192 239 238 213 264 244 243 246 289
 287 209 263 249 247 284 327 325 312 350 258 362 310 317 268 202 294 280
 216 292 304]
```

registered:- No. of Unique Values: 731; Unique Values: [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]
```

count:- No. of Unique Values: 822; Unique Values: [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]
```

Checking for duplicates

[138]:
```
duplicate = yulu[yulu.duplicated()]

print("Duplicate Rows :")
```

```
Duplicate Rows :
```

Descriptive Statistics for Features

[139]:
```
yulu.describe()
```

[139]:
```
             season       holiday    workingday       weather         temp  \
count  10886.000000  10886.000000  10886.000000  10886.000000  10886.00000
mean       2.506614      0.028569      0.680875      1.418427     20.23086
std        1.116174      0.166599      0.466159      0.633839      7.79159
min        1.000000      0.000000      0.000000      1.000000      0.82000
25%        2.000000      0.000000      0.000000      1.000000     13.94000
50%        3.000000      0.000000      1.000000      1.000000     20.50000
75%        4.000000      0.000000      1.000000      2.000000     26.24000
```

```
max        4.000000      1.000000      1.000000      4.000000     41.00000

                atemp      humidity     windspeed        casual    registered  \
count  10886.000000  10886.000000  10886.000000  10886.000000  10886.000000
mean      23.655084     61.886460     12.799395     36.021955    155.552177
std        8.474601     19.245033      8.164537     49.960477    151.039033
min        0.760000      0.000000      0.000000      0.000000      0.000000
25%       16.665000     47.000000      7.001500      4.000000     36.000000
50%       24.240000     62.000000     12.998000     17.000000    118.000000
75%       31.060000     77.000000     16.997900     49.000000    222.000000
max       45.455000    100.000000     56.996900    367.000000    886.000000

              count
count  10886.000000
mean     191.574132
std      181.144454
min        1.000000
25%       42.000000
50%      145.000000
75%      284.000000
max      977.000000
```

[ ]:
```
yulu['season'].value_counts(normalize=True)
```

[ ]:
```
4    0.251148
2    0.251056
3    0.251056
1    0.246739
Name: season, dtype: float64
```

[ ]:
```
yulu['holiday'].value_counts(normalize=True)
```

[ ]:
```
0    0.971431
1    0.028569
Name: holiday, dtype: float64
```

[ ]:
```
yulu['workingday'].value_counts(normalize=True)
```

[ ]:
```
1    0.680875
0    0.319125
Name: workingday, dtype: float64
```

[ ]:
```
yulu['weather'].value_counts(normalize=True)
```

[ ]:
```
1    0.660665
2    0.260334
3    0.078909
```

```
4    0.000092
Name: weather, dtype: float64
```

As we can see from the data that we have counts of electric cycles rented per hour and also have weather, temperature, humidity values which change every year; we can intutively say, these factors impact the no. of cycles rented.

To enable better analysis i am going to add a few more columns to our existing data frame.

```
[ ]: yulu["datetime"] = pd.to_datetime(yulu['datetime'])
     yulu["date"] =yulu['datetime'].dt.day
     yulu['month']=yulu['datetime'].dt.month
     yulu['year'] = yulu['datetime'].dt.year
     yulu['day'] = yulu['datetime'].dt.day_name()
     yulu['time'] = yulu['datetime'].dt.time
     yulu["onlydate"] =yulu['datetime'].dt.date
     yulu['weekend']= np.where(((yulu['workingday']== 0) & (yulu['holiday']== 0)) ,␣
      ↪1, 0)
```

```
[ ]: yulu.head(7)
```

```
[ ]:             datetime  season  holiday  workingday  weather  temp   atemp  \
     0 2011-01-01 00:00:00       1        0           0        1  9.84  14.395
     1 2011-01-01 01:00:00       1        0           0        1  9.02  13.635
     2 2011-01-01 02:00:00       1        0           0        1  9.02  13.635
     3 2011-01-01 03:00:00       1        0           0        1  9.84  14.395
     4 2011-01-01 04:00:00       1        0           0        1  9.84  14.395
     5 2011-01-01 05:00:00       1        0           0        2  9.84  12.880
     6 2011-01-01 06:00:00       1        0           0        1  9.02  13.635

        humidity  windspeed  casual  registered  count  date  month  year  \
     0        81     0.0000       3          13     16     1      1  2011
     1        80     0.0000       8          32     40     1      1  2011
     2        80     0.0000       5          27     32     1      1  2011
     3        75     0.0000       3          10     13     1      1  2011
     4        75     0.0000       0           1      1     1      1  2011
     5        75     6.0032       0           1      1     1      1  2011
     6        80     0.0000       2           0      2     1      1  2011

             day      time    onlydate  weekend
     0  Saturday  00:00:00  2011-01-01        1
     1  Saturday  01:00:00  2011-01-01        1
     2  Saturday  02:00:00  2011-01-01        1
     3  Saturday  03:00:00  2011-01-01        1
     4  Saturday  04:00:00  2011-01-01        1
     5  Saturday  05:00:00  2011-01-01        1
     6  Saturday  06:00:00  2011-01-01        1
```

```
[ ]: yulu['weekend'].value_counts(normalize=True)
```

```
[ ]: 0    0.709443
     1    0.290557
     Name: weekend, dtype: float64
```

#**Inferences:**

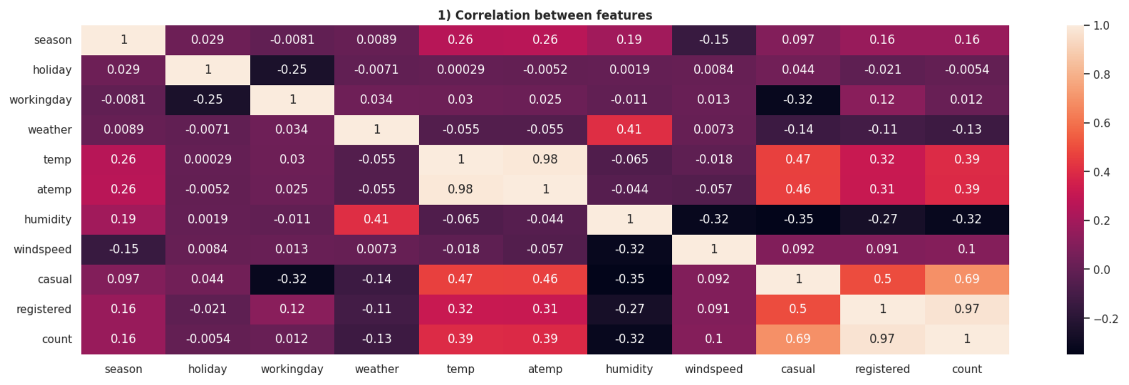
1. **Datetime:** Count of cycle rented and weather conditions data of approx 10.8K hours spread over a span of 2 years recorded.
2. **Season:** We have 4 seasons, equally distributed.
3. **workingday:** 68% of these hours have been recorded from working days and 31% from non working days.
4. **weather:** weather -1 which is-> Clear, Few clouds, partly cloudy, partly cloudy is the most frequently occuring weather, followed by weather -2. out of these 10.8K hours, only 1 hour recorded a weather of category 4.
5. **Temp:** The mean temperature is 20.23 degrees, maximum temp climbing upto 41 degrees and minimum being 0.82 degrees.
6. **Atemp:** The mean temperature felt is 23.65 degrees, maximum temp climbing upto 45 degrees and minimum being 0.7 degrees.
7. **Humidity:** The mean humidity is 61 g/m^3, maximum humidity climbing upto 100 g/m^3 and minimum being 0 g/m^3.
8. **Windspeed:** The 25 percentile to 75 percentile windspeed is between 7km/hr and 17km/hr.
9. **Casual:** average count of casual users on a hourly basis is 36 users. maximum casual users recorded in an hour is 367 and minimum is 0.
10. **Registered:** average count of registered users on an hourly basis is 155 users. maximum registered users in an hour is 886 and minimum is zero.
11. **Count:** average count of total rental bikes including both casual and registered on an hourly basis is 191.

#**Visual Analysis**

#1. **Correlation**

```
[140]: plt.figure(figsize = (21,6))
       sns.heatmap(yulu.corr(), annot=True)
       plt.title('1) Correlation between features',fontweight="bold")
       plt.show()
```

```
<ipython-input-140-4ce8190faa5b>:2: FutureWarning: The default value of
numeric_only in DataFrame.corr is deprecated. In a future version, it will
default to False. Select only valid columns or specify the value of numeric_only
to silence this warning.
  sns.heatmap(yulu.corr(), annot=True)
```



#**Inferences**

The heatmap for correlation shows:

1. It shows a correlation between temp, atemp and count of bikes rented.
2. It shows a correlation between season and temperature and humidity.
3. It shows a correlation between humidity and weather.
4. It shows a correlation between count and month.

#2. **Univariate Analysis**

```
[ ]: holiday_count=yulu.groupby(['holiday'])['count'].sum()
     holiday_count
```

```
[ ]: holiday
     0    2027668
     1      57808
     Name: count, dtype: int64
```

```
[ ]: workingday_count=yulu.groupby(['workingday'])['count'].sum()
     workingday_count
```

```
[ ]: workingday
     0     654872
     1    1430604
     Name: count, dtype: int64
```

```
[ ]: weather_count=yulu.groupby(['weather'])['count'].sum()
     weather_count
```

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

```
Name: count, dtype: int64
```

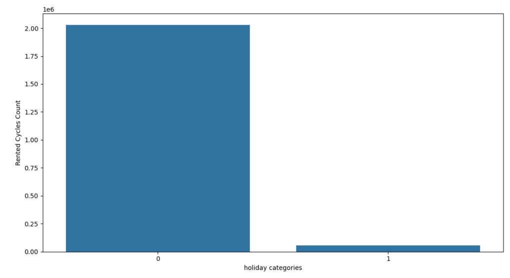
[ ]:
```
season_count=yulu.groupby(['season'])['count'].sum()
season_count
```

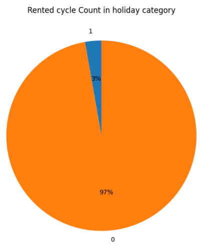
[ ]:
```
season
1    312498
2    588282
3    640662
4    544034
Name: count, dtype: int64
```

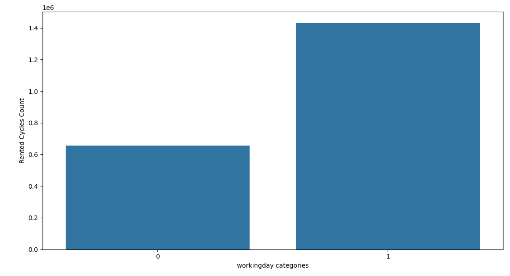
[ ]:
```
weekend_count=yulu.groupby(['weekend'])['count'].sum()
weekend_count
```

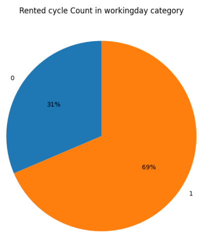
[ ]:
```
weekend
0    1488412
1     597064
Name: count, dtype: int64
```

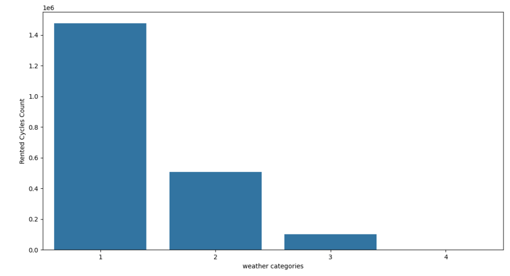
[ ]:
```
colnames = ['holiday', 'workingday', 'weather', 'season', 'weekend', 'month',␣
 ↪'year']
plt.figure(figsize=(20, len(colnames) * 6))
k = 1

for colname in colnames:
    plt.subplot(len(colnames), 2, k)
    s = df.groupby(colname)['count'].sum().sort_values(ascending=True)
    s = s.reset_index()
    g = sns.barplot(data=s, x=s[colname], y='count')
    g.set_ylabel("Rented Cycles Count")
    g.set_xlabel(f"{colname} categories")
    k += 1

    plt.subplot(len(colnames), 2, k)
    plt.pie(s['count'], labels=s[colname], autopct='%.0f%%', startangle=90)
    plt.title(f'Rented cycle Count in {colname} category')
    k += 1

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

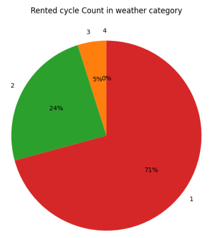

1e6
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
Rented Cycles Count
0
1
holiday categories

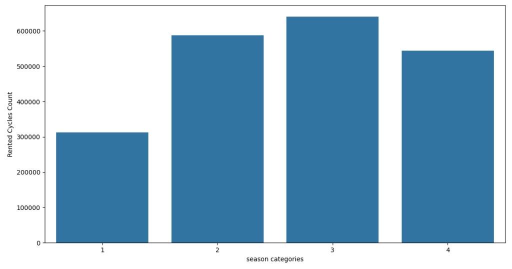

Rented cycle Count in holiday category
1
3%
97%
0

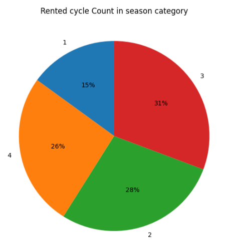

1e6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
Rented Cycles Count
0
1
workingday categories

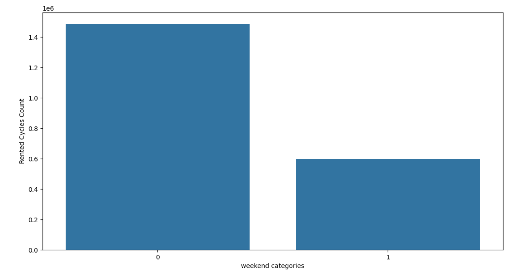

Rented cycle Count in workingday category
0
31%
69%
1

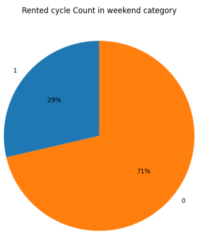

1e6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
Rented Cycles Count
1
2
3
4
weather categories

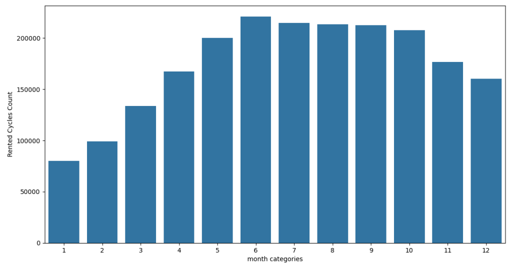

Rented cycle Count in weather category
3
4
5%
0%
2
24%
71%
1

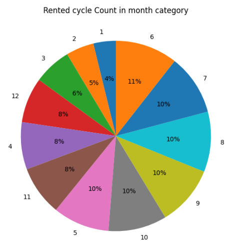

600000
500000
400000
300000
200000
100000
0
Rented Cycles Count
1
2
3
4
season categories

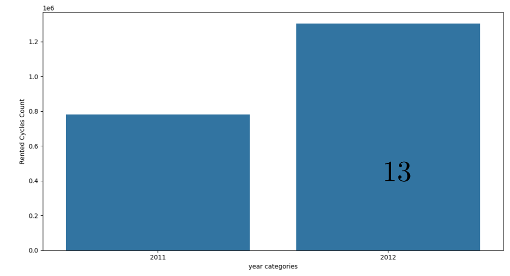

Rented cycle Count in season category
1
15%
3
31%
4
26%
28%
2

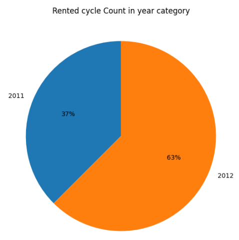

1e6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
Rented Cycles Count
0
1
weekend categories


Rented cycle Count in weekend category
1
29%
71%
0


200000
150000
100000
50000
0
Rented Cycles Count
1
2
3
4
5
6
7
8
9
10
11
12
month categories

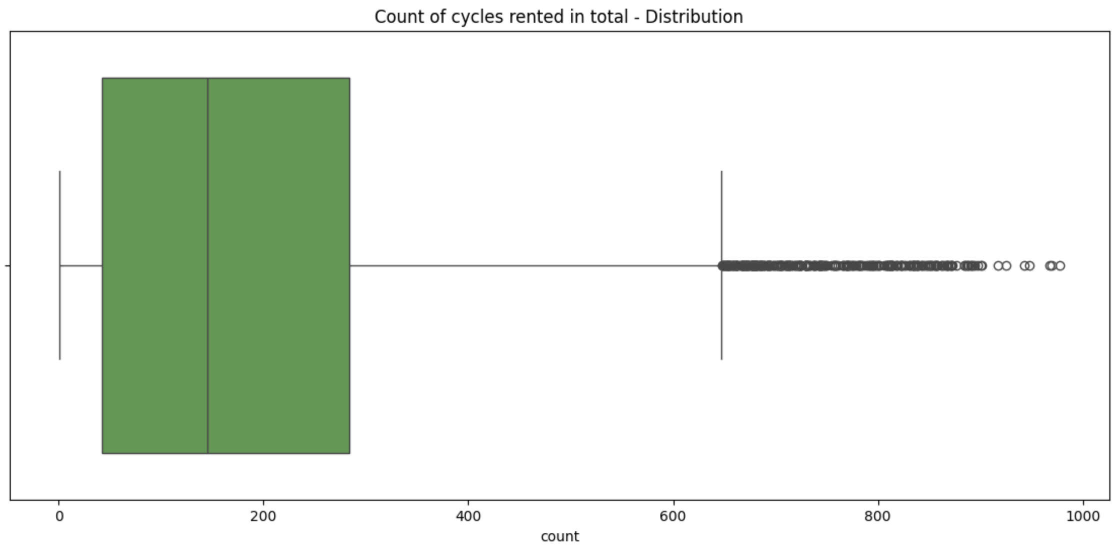

Rented cycle Count in month category
1
2
3
6
7
12
4%
5%
6%
11%
10%
8%
10%
8
4
8%
10%
11
8%
10%
10%
9
5
10


1e6
1.2
1.0
0.8
0.6
0.4
0.2
0.0
Rented Cycles Count
2011
2012
year categories


Rented cycle Count in year category
2011
37%
63%
2012

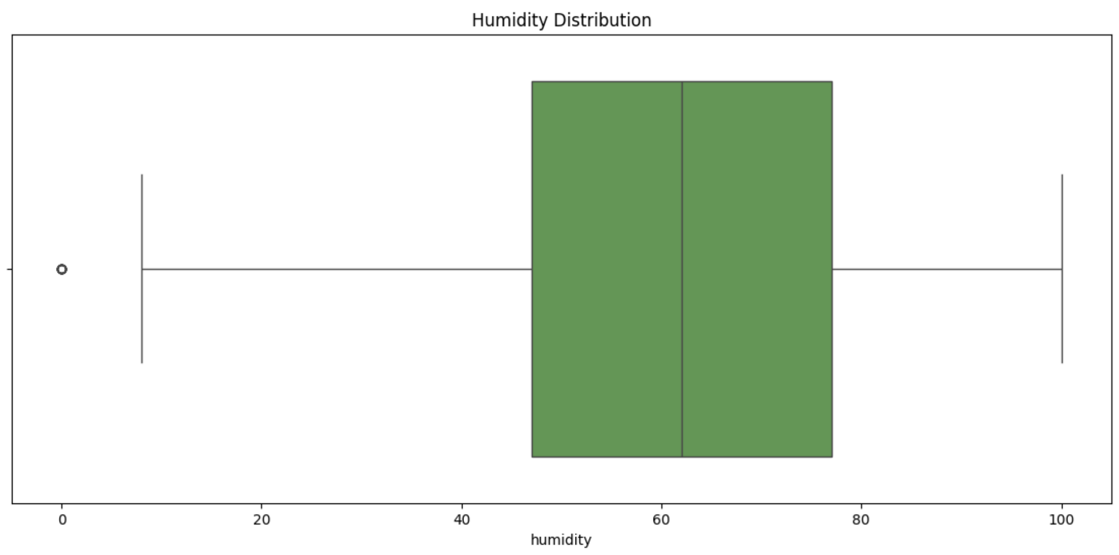
```
[ ]: plt.figure(figsize = (14,6))
     sns.boxplot(data=yulu, x='count', palette='gist_earth')
     plt.title("Count of cycles rented in total - Distribution")
     plt.show()
```

```
<ipython-input-48-d0a99a8a7b09>:2: FutureWarning:

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

  sns.boxplot(data=yulu, x='count', palette='gist_earth')
```



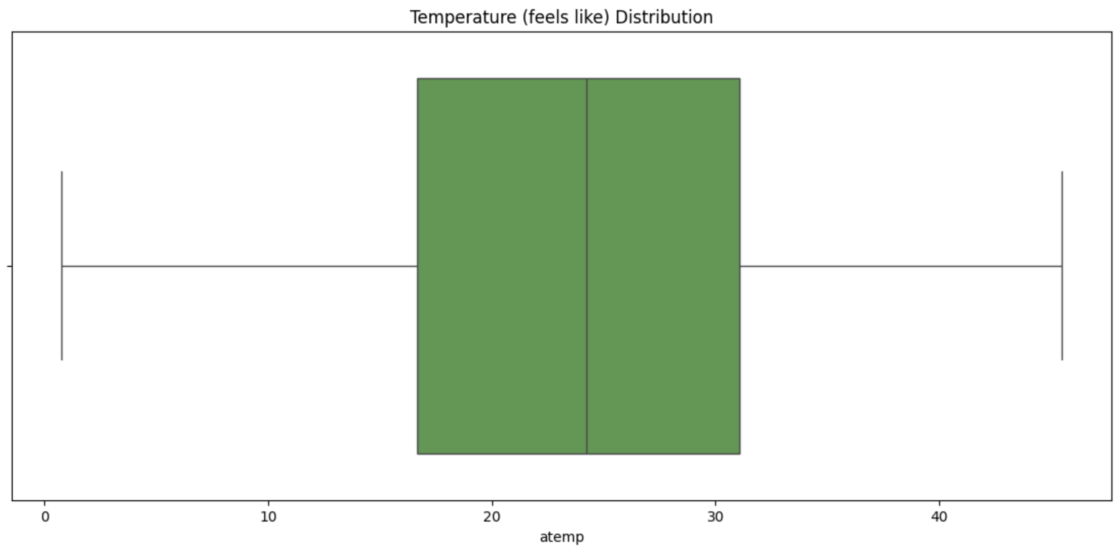
```
[ ]: plt.figure(figsize = (14,6))
     sns.boxplot(data=yulu, x='humidity', palette='gist_earth')
     plt.title("Humidity Distribution")
     plt.show()
```

```
<ipython-input-49-6746b8e79018>:2: FutureWarning:

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

  sns.boxplot(data=yulu, x='humidity', palette='gist_earth')
```



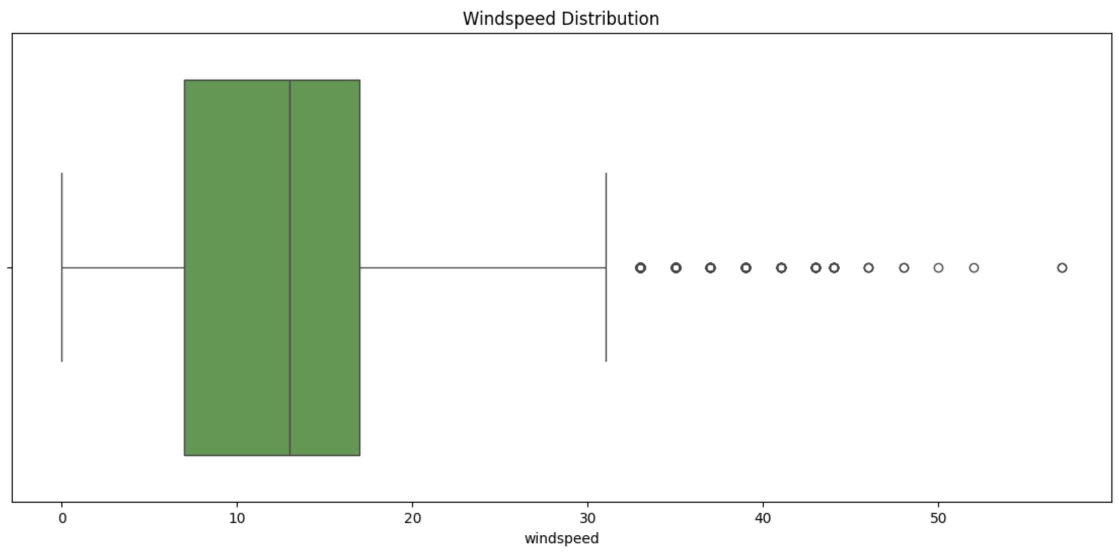
```
[ ]: plt.figure(figsize = (14,6))
     sns.boxplot(data=yulu, x='atemp', palette='gist_earth')
     plt.title("Temperature (feels like) Distribution")
     plt.show()
```

```
<ipython-input-50-002a10a06e72>:2: FutureWarning:

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

  sns.boxplot(data=yulu, x='atemp', palette='gist_earth')
```



```
[ ]: plt.figure(figsize = (14,6))
     sns.boxplot(data=yulu, x='windspeed', palette='gist_earth')
     plt.title("Windspeed Distribution")
     plt.show()
```

```
<ipython-input-51-8d78f15c2b1e>:2: FutureWarning:

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

  sns.boxplot(data=yulu, x='windspeed', palette='gist_earth')
```



#Inferences

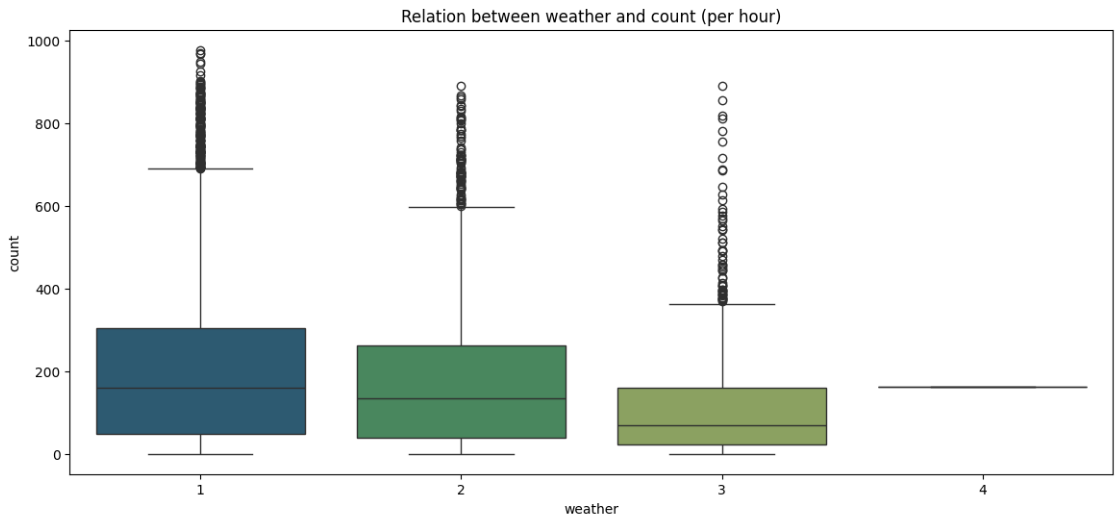
1. **Season:** Season 3 sees the highest amount of cycles rented, followed by 2, 4 and 1.
2. **weather:** Users mostly rent cycle during weather 1, followed by 2 and 3. Weather 4 was recorded only once in the period of 2 years.
3. **Atemp:** The average temperature while users rent cycles fall between 17 degrees to 31 degrees.
4. **Humidity:** Humidity generally is between 47 to 77, with some outliers at 0.
5. **Windspeed:** The 25 percentile to 75 percentile windspeed is between 7km/he and 17km/hr.
6. **Casual:** average count of casual users on a hourly basis is 36 users. maximum casual users recorded in an hour is 367 and minimum is 0.
7. **Registered:** average count of registered users on an hourly basis is 155 users. maximum registered users in an hour is 886 and minimum is zero.
8. **Count:**

- 2012 saw a significance rise in cycle rent count (63%) as compared to 2011 (37%)
- Only 21% cycle rents happened over the weekends , whereas weekdays contributed to 79% of cycle rent count.
- Month of June (all at 11% of gross cycle rents) saw the maximam gross of Cycle rents, followed by May, July, August, September and October (all at 10% of gross cycle rents)
- Gross Cycle rents Maximum at season 3 followed by Season 2.
- Cycle rents were maximum at Weather Category 1 (71%), followed by weather category 2 (24%)

#3. **Bivariate Analysis**

```
[ ]: plt.figure(figsize = (14,6))
     sns.boxplot(data=yulu, x='weather', y= 'count', palette='gist_earth')
     plt.title("Relation between weather and count (per hour)")
     plt.show()
```

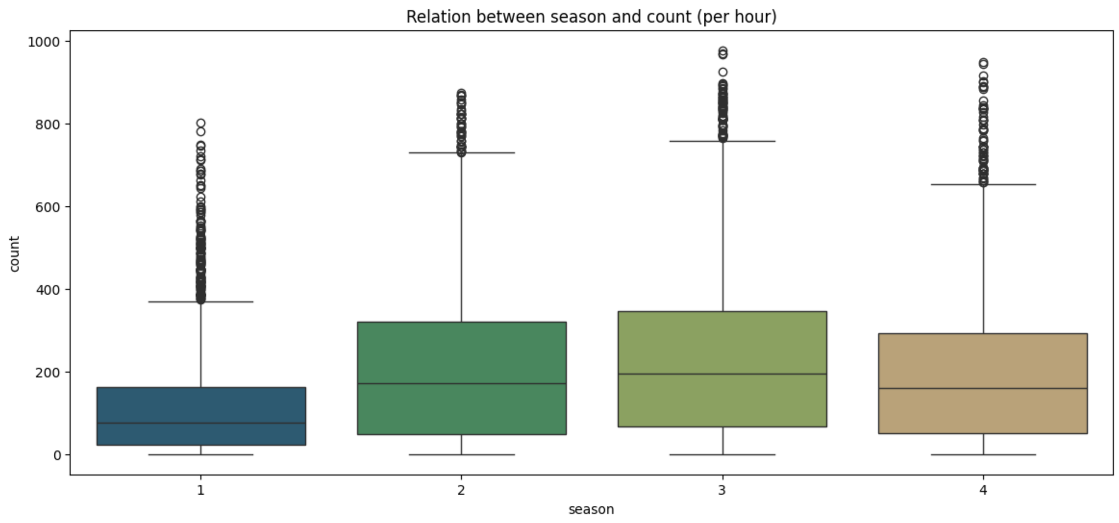
```
<ipython-input-52-65148092a2b5>:2: FutureWarning:

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

  sns.boxplot(data=yulu, x='weather', y= 'count', palette='gist_earth')
```



```
[ ]: plt.figure(figsize = (14,6))
     sns.boxplot(data=yulu, x='season', y= 'count', palette='gist_earth')
     plt.title("Relation between season and count (per hour)")
     plt.show()
```

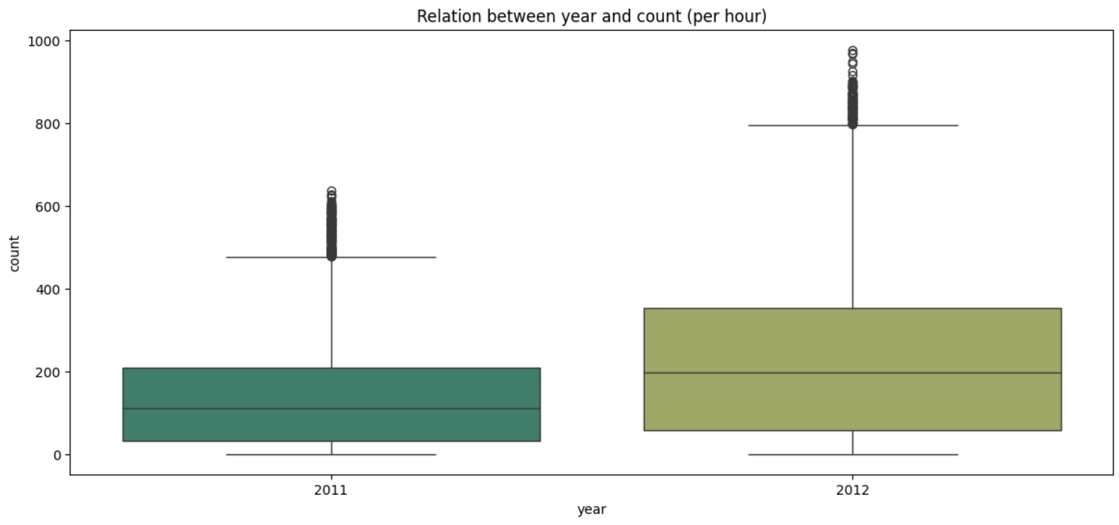
```
<ipython-input-53-07c7c04eff83>:2: FutureWarning:
```

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

  sns.boxplot(data=yulu, x='season', y= 'count', palette='gist_earth')
```



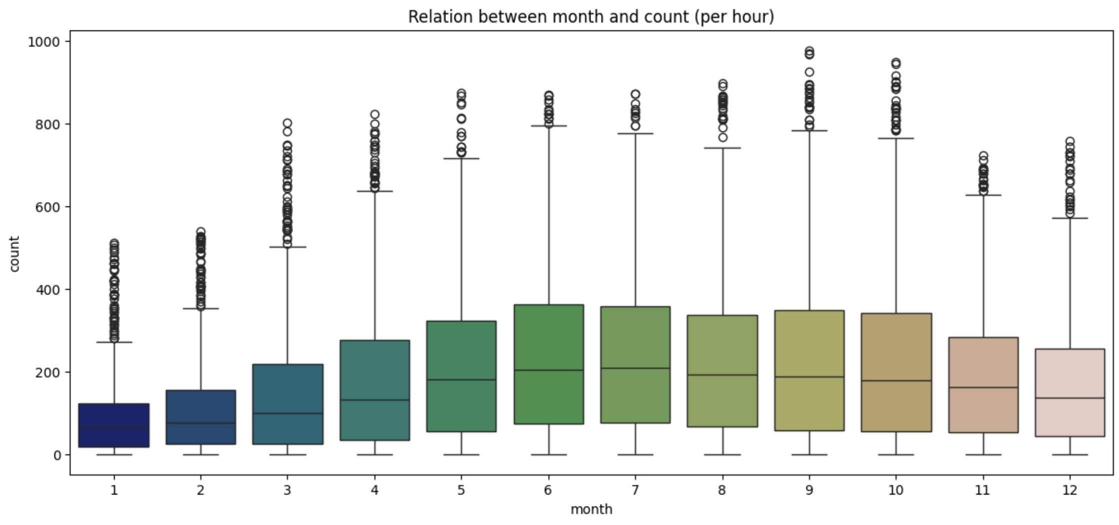
[ ]:
```
plt.figure(figsize = (14,6))
sns.boxplot(data=yulu, x='year', y= 'count', palette='gist_earth')
plt.title("Relation between year and count (per hour)")
plt.show()
```

```
<ipython-input-54-e411940ac66b>:2: FutureWarning:

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

  sns.boxplot(data=yulu, x='year', y= 'count', palette='gist_earth')
```



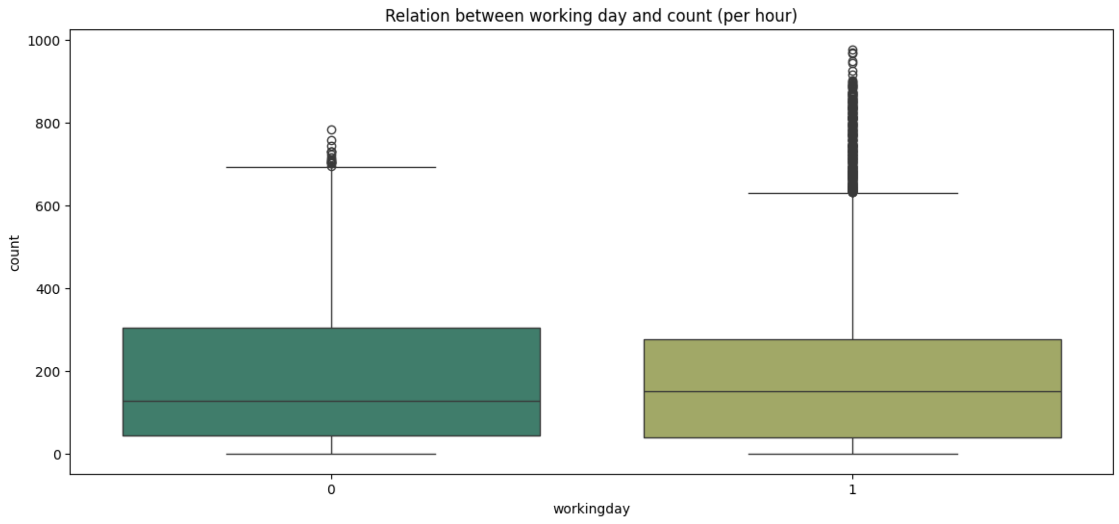
```
[ ]: plt.figure(figsize = (14,6))
     sns.boxplot(data=yulu, x='month', y= 'count', palette='gist_earth')
     plt.title("Relation between month and count (per hour)")
     plt.show()
```

```
<ipython-input-55-409a5d743bb0>:2: FutureWarning:

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

  sns.boxplot(data=yulu, x='month', y= 'count', palette='gist_earth')
```



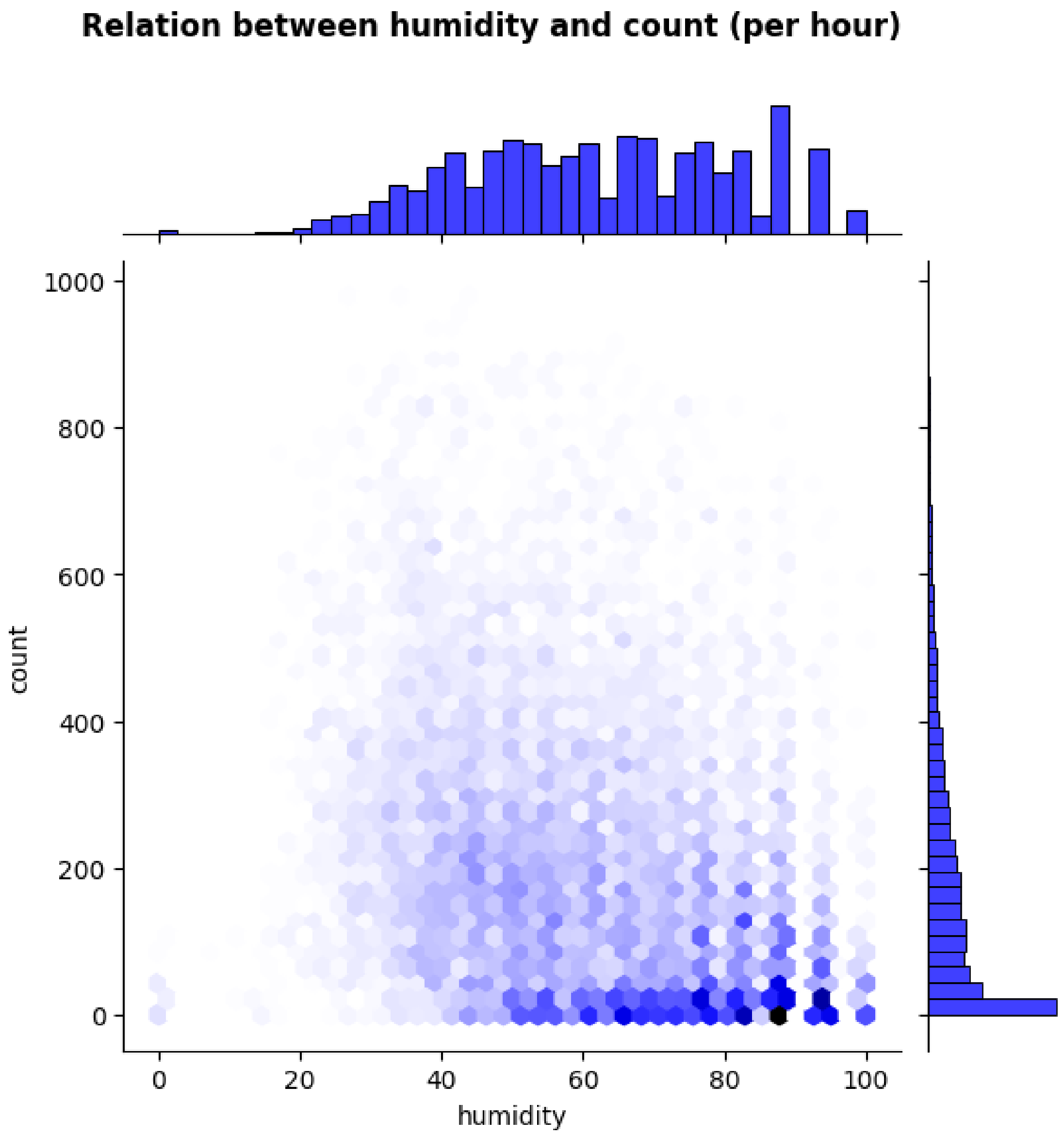
```
[ ]: plt.figure(figsize = (14,6))
     sns.boxplot(data=yulu, x='workingday', y= 'count', palette='gist_earth')
     plt.title("Relation between working day and count (per hour)")
     plt.show()
```

```
<ipython-input-56-d4ed37682c60>:2: FutureWarning:

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

  sns.boxplot(data=yulu, x='workingday', y= 'count', palette='gist_earth')
```



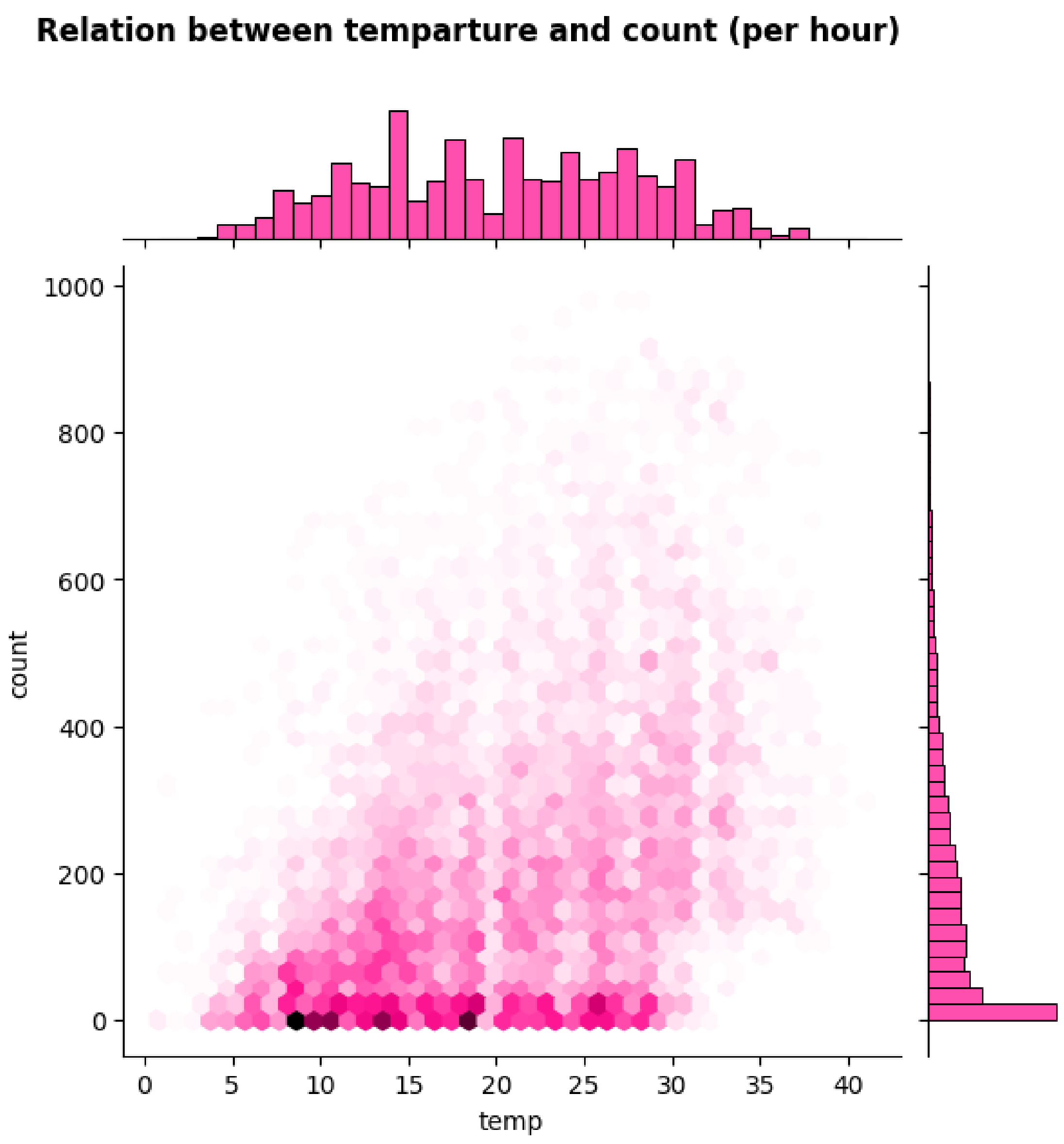
```
[ ]: plt.figure(figsize = (14,6))
     # sns.set_theme(style="ticks")
     sns.jointplot(x=yulu['humidity'], y=yulu['count'], kind="hex", color="b")
     plt.title("Relation between humidity and count (per hour)", loc = "right",pad=␣
      ↪90,fontweight="bold")
     plt.show()
```

```
<Figure size 1400x600 with 0 Axes>
```



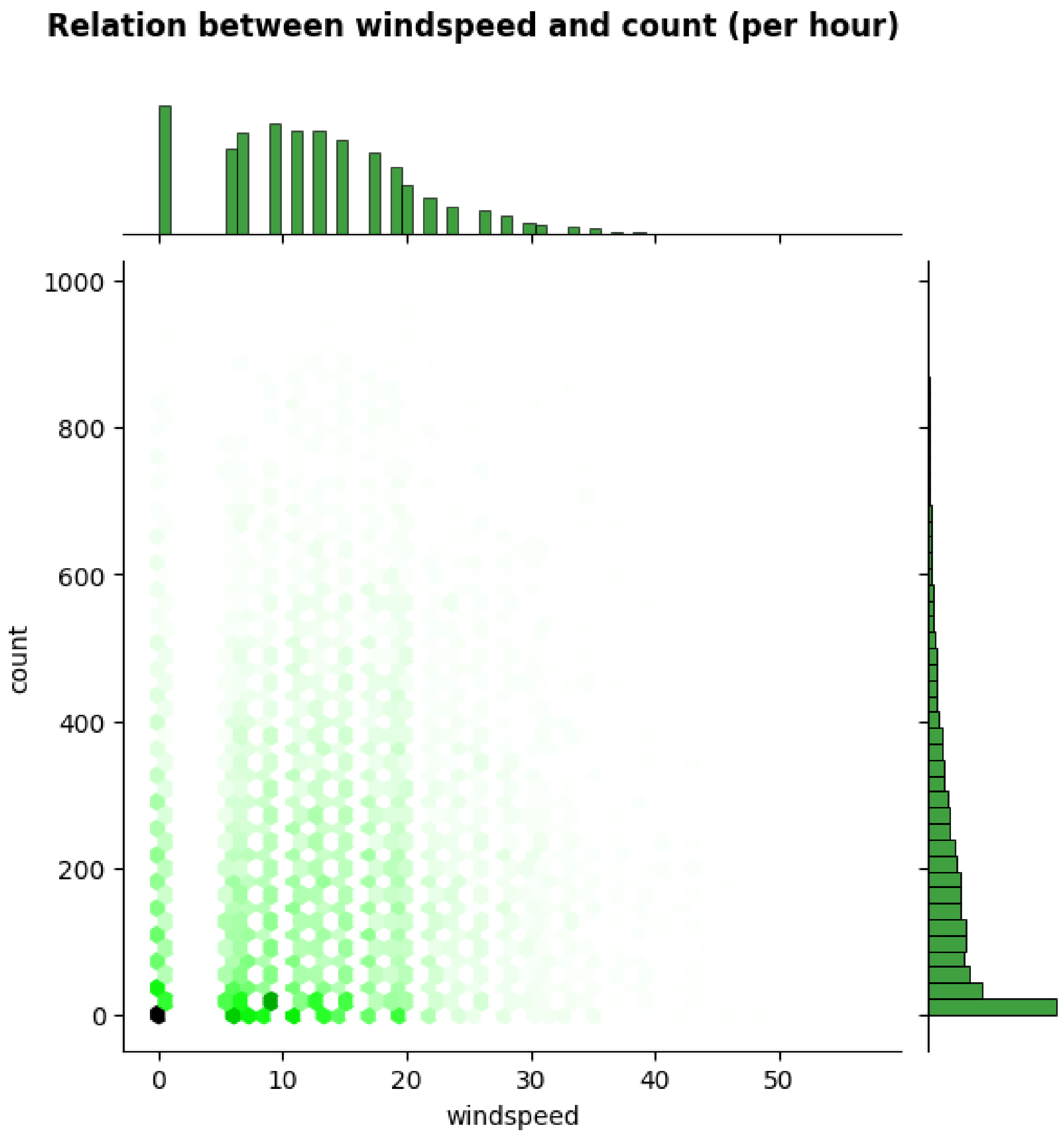
```
[ ]: plt.figure(figsize = (14,6))
     # sns.set_theme(style="ticks")
     sns.jointplot(x=yulu['temp'], y=yulu['count'], kind="hex", color="deeppink")
     plt.title("Relation between temparture and count (per hour)", loc =␣
      ↪"right",pad= 90,fontweight="bold")
     plt.show()
```

```
<Figure size 1400x600 with 0 Axes>
```



```
[ ]: plt.figure(figsize = (14,6))
     # sns.set_theme(style="ticks")
     sns.jointplot(x=yulu['windspeed'], y=yulu['count'], kind="hex", color="g")
     plt.title("Relation between windspeed and count (per hour)", loc = "right",pad=␣
      ↪90,fontweight="bold")
     plt.show()
```

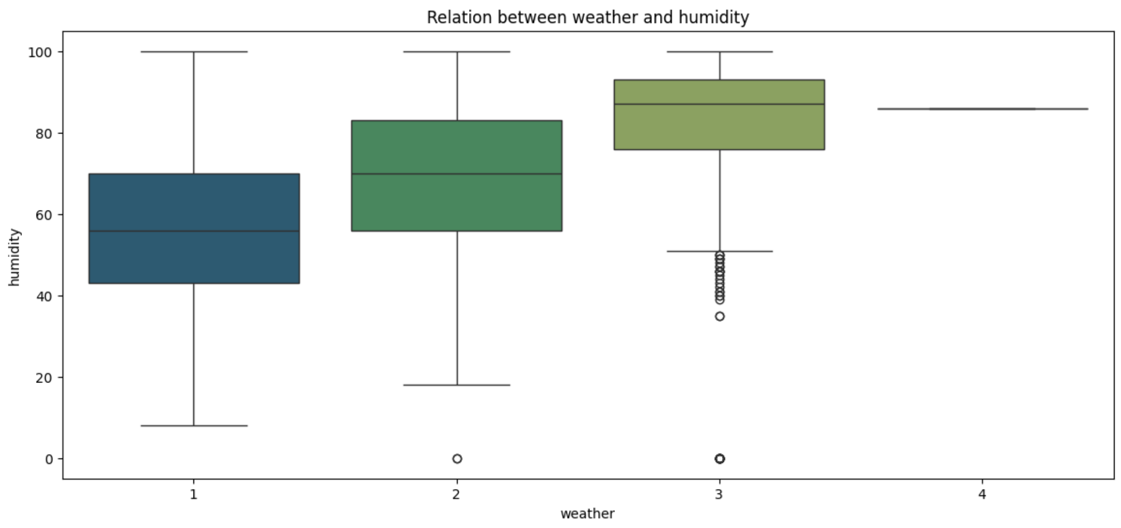
```
<Figure size 1400x600 with 0 Axes>
```



```
[ ]: plt.figure(figsize = (14,6))
     sns.boxplot(data=yulu, x='weather', y= 'humidity', palette='gist_earth')
     plt.title("Relation between weather and humidity")
     plt.show()
```

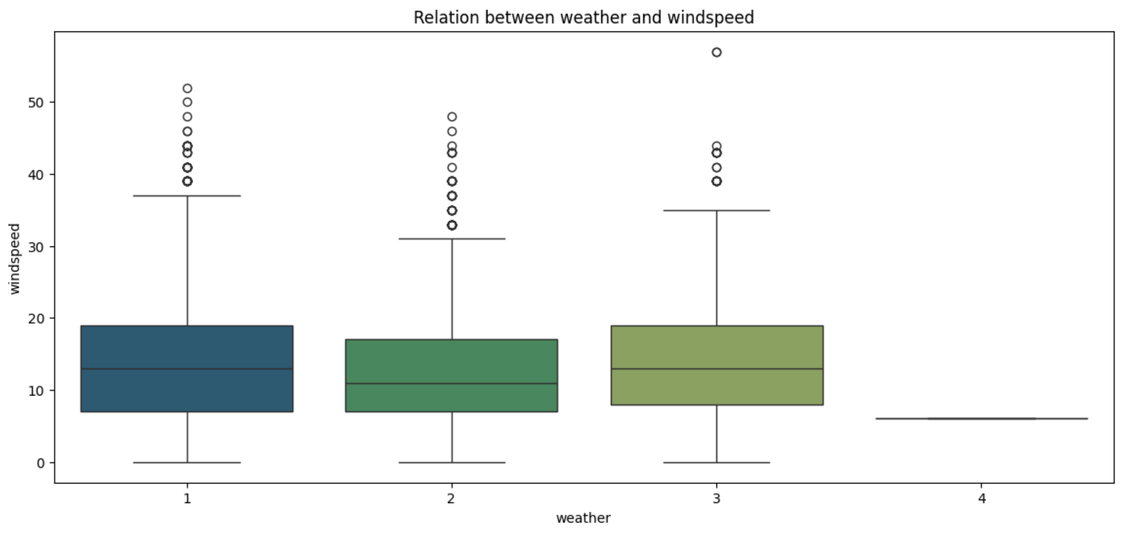
```
<ipython-input-60-509fd7fbc9f5>:2: FutureWarning:

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

```
  sns.boxplot(data=yulu, x='weather', y= 'humidity', palette='gist_earth')
```



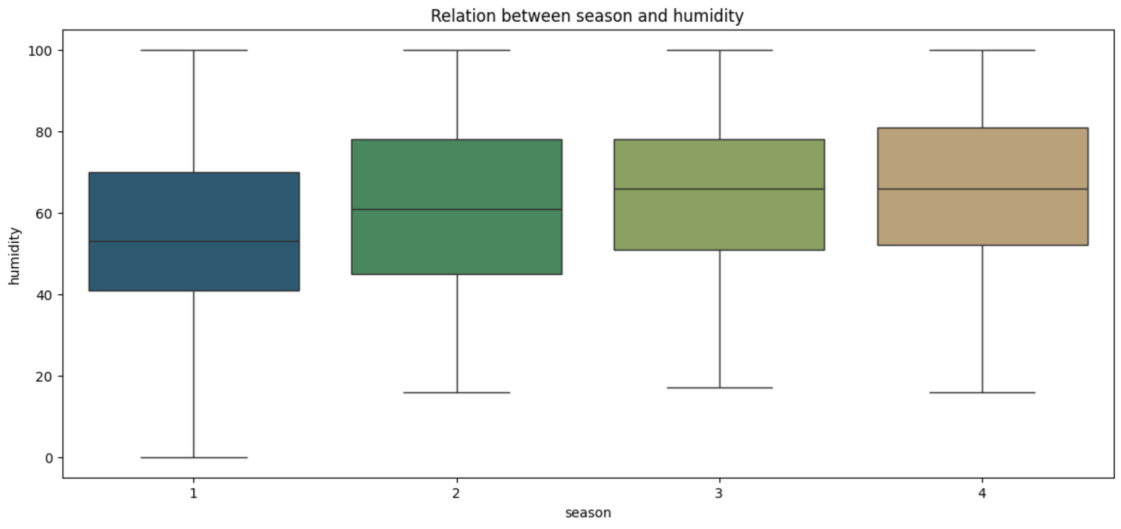
```
[ ]: plt.figure(figsize = (14,6))
     sns.boxplot(data=yulu, x='weather', y= 'windspeed', palette='gist_earth')
     plt.title("Relation between weather and windspeed")
     plt.show()
```

```
<ipython-input-61-2968f32485d0>:2: FutureWarning:

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

  sns.boxplot(data=yulu, x='weather', y= 'windspeed', palette='gist_earth')
```



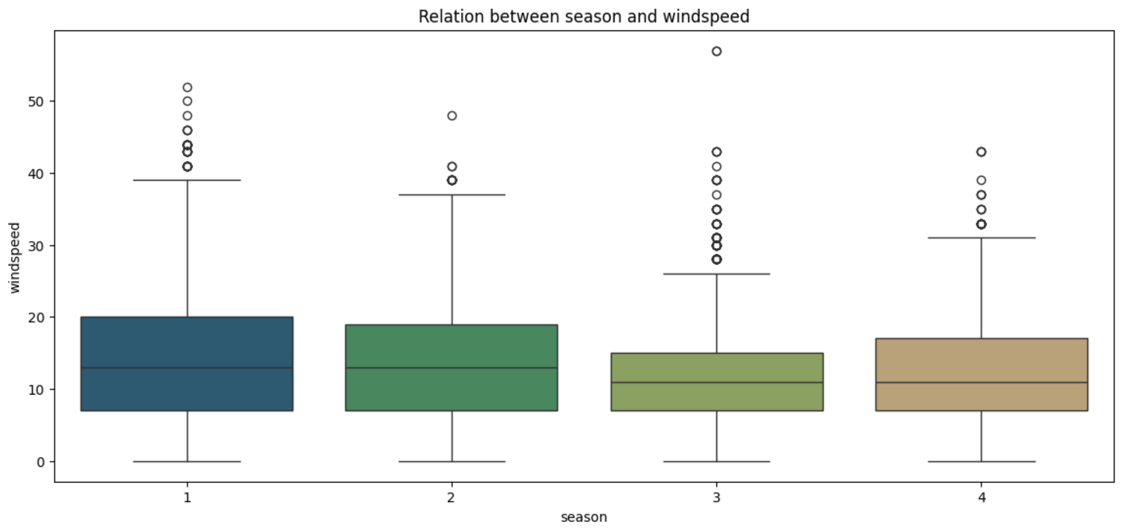
[ ]:
```
plt.figure(figsize = (14,6))
sns.boxplot(data=yulu, x='season', y= 'humidity', palette='gist_earth')
plt.title("Relation between season and humidity")
plt.show()
```

```
<ipython-input-62-d0f5721a313c>:2: FutureWarning:

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

  sns.boxplot(data=yulu, x='season', y= 'humidity', palette='gist_earth')
```



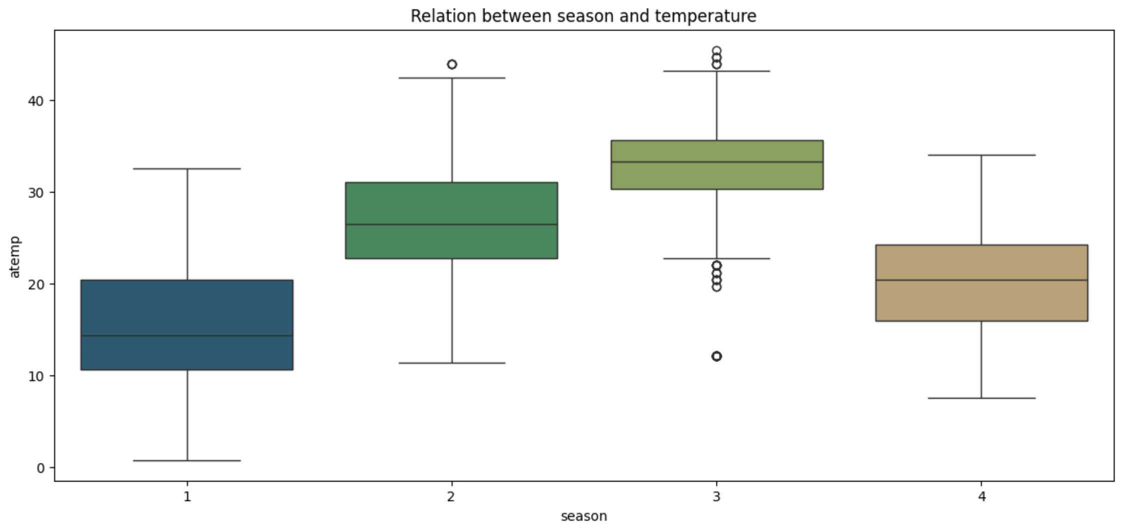
[ ]:
```
plt.figure(figsize = (14,6))
sns.boxplot(data=yulu, x='season', y= 'windspeed', palette='gist_earth')
plt.title("Relation between season and windspeed")
plt.show()
```

```
<ipython-input-63-641d2a2b79a8>:2: FutureWarning:

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

  sns.boxplot(data=yulu, x='season', y= 'windspeed', palette='gist_earth')
```

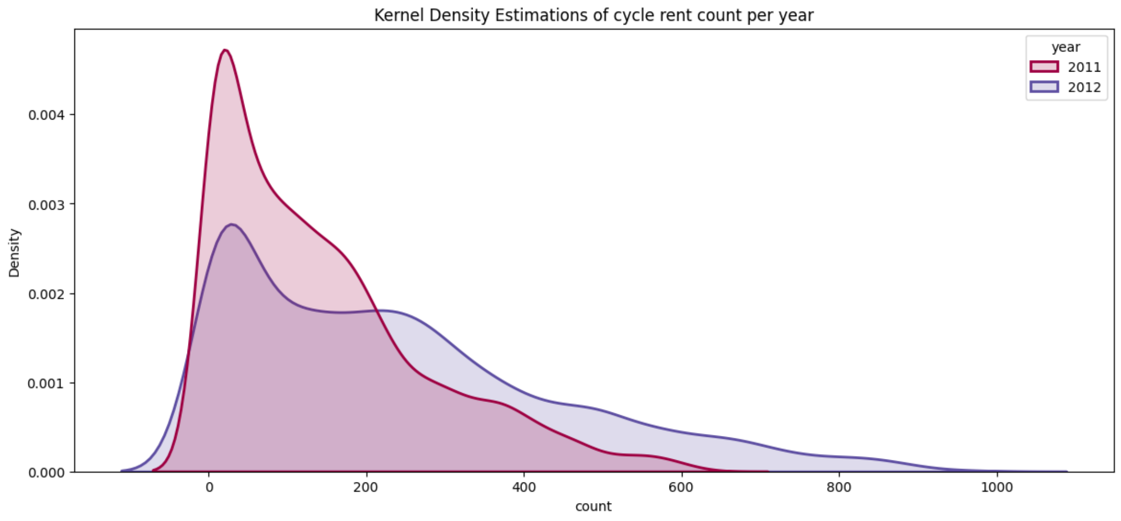


```
[ ]: plt.figure(figsize = (14,6))
     sns.boxplot(data=yulu, x='season', y= 'atemp', palette='gist_earth')
     plt.title("Relation between season and temperature")
     plt.show()
```

```
<ipython-input-64-45c0a260fd02>:2: FutureWarning:

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

  sns.boxplot(data=yulu, x='season', y= 'atemp', palette='gist_earth')
```

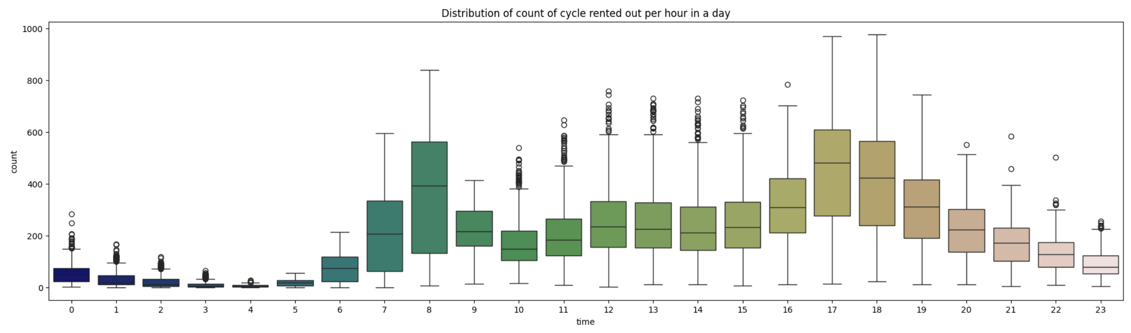


```
[ ]: plt.figure(figsize = (14,6))
     sns.kdeplot(data=yulu, x="count", hue="year", fill=True, common_norm=False, ␣
      ↪palette='Spectral', alpha=.2, linewidth=2)
     plt.title("Kernel Density Estimations of cycle rent count per year")
     plt.show()
```



```
[ ]: yulu['time'] = yulu['time'].astype(str).apply(lambda x: x[:2]).astype(int)
     plt.figure(figsize = (23,6))
     sns.boxplot(data=yulu, x='time', y= 'count', palette='gist_earth')
     plt.title("Distribution of count of cycle rented out per hour in a day")
     plt.show()
```

```
<ipython-input-66-f74ca101eb26>:3: FutureWarning:

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

  sns.boxplot(data=yulu, x='time', y= 'count', palette='gist_earth')
```



PER DAY ANALYSIS

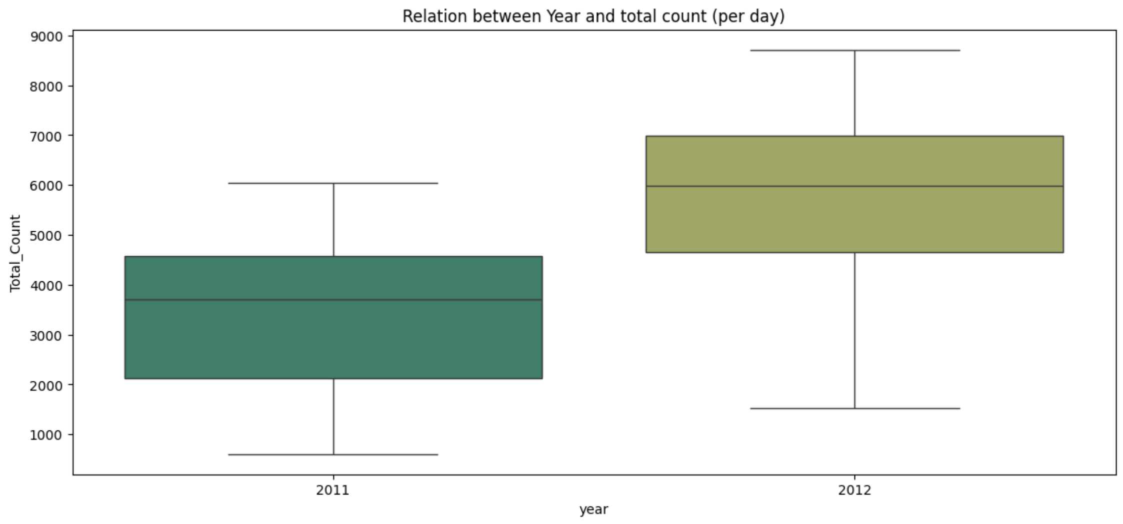
```
[ ]: modf_yulu = yulu.copy()
     modf_yulu.drop(labels = ['datetime',  'weather', 'time', 'windspeed'], axis =␣
       ↪1, inplace= True)
     modf_yulu['Total_Count'] = modf_yulu['count'].groupby(modf_yulu['onlydate']).
       ↪transform('sum')
     modf_yulu['Total_Casual'] = modf_yulu['casual'].groupby(modf_yulu['onlydate']).
       ↪transform('sum')
     modf_yulu['Total_Registered'] = modf_yulu['registered'].
       ↪groupby(modf_yulu['onlydate']).transform('sum')
     modf_yulu['temp']= modf_yulu['temp'].groupby(modf_yulu['onlydate']).
       ↪transform('mean')
     modf_yulu['atemp']= modf_yulu['atemp'].groupby(modf_yulu['onlydate']).
       ↪transform('mean')
     modf_yulu['humidity']= modf_yulu['humidity'].groupby(modf_yulu['onlydate']).
       ↪transform('mean')
     modf_yulu.drop(labels = ['registered','casual', 'count'], axis = 1, inplace=␣
       ↪True)
     modf_yulu.drop_duplicates(inplace=True)
     modf_yulu.head(5)
```

```
[ ]:     season  holiday  workingday       temp      atemp   humidity  date  month  \
     0        1        0           0  14.110833  18.181250  80.583333     1      1
     24       1        0           0  14.902609  17.686957  69.608696     2      1
     47       1        0           1   8.050909   9.470227  43.727273     3      1
     69       1        0           1   8.200000  10.606087  59.043478     4      1
     92       1        0           1   9.305217  11.463478  43.695652     5      1

         year        day    onlydate  weekend  Total_Count  Total_Casual  \
     0   2011   Saturday  2011-01-01        1          985           331
     24  2011     Sunday  2011-01-02        1          801           131
     47  2011     Monday  2011-01-03        0         1349           120
     69  2011    Tuesday  2011-01-04        0         1562           108
     92  2011  Wednesday  2011-01-05        0         1600            82
```

```
    Total_Registered
0                654
24               670
47              1229
69              1454
92              1518
```

[ ]:
```
plt.figure(figsize = (14,6))
sns.boxplot(data=modf_yulu, x='year', y= 'Total_Count', palette='gist_earth')
plt.title("Relation between Year and total count (per day)")
plt.show()
```

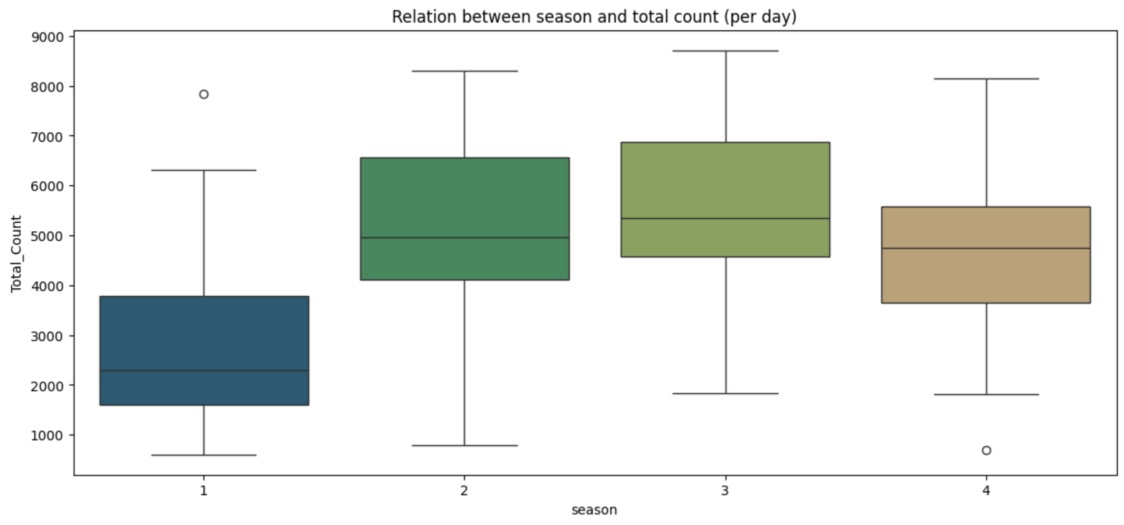
```
<ipython-input-68-0026e1d77299>:2: FutureWarning:

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

  sns.boxplot(data=modf_yulu, x='year', y= 'Total_Count', palette='gist_earth')
```



[ ]:
```
plt.figure(figsize = (14,6))
sns.boxplot(data=modf_yulu, x='season', y= 'Total_Count', palette='gist_earth')
plt.title("Relation between season and total count (per day)")
plt.show()
```

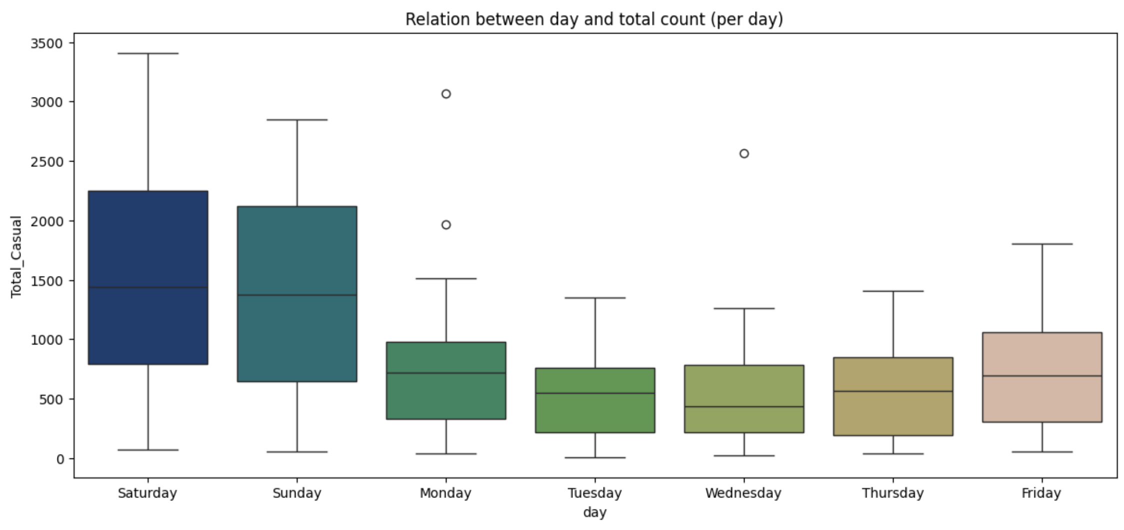
```
<ipython-input-69-e24311e2a774>:2: FutureWarning:

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

```
effect.

  sns.boxplot(data=modf_yulu, x='season', y= 'Total_Count',
palette='gist_earth')
```



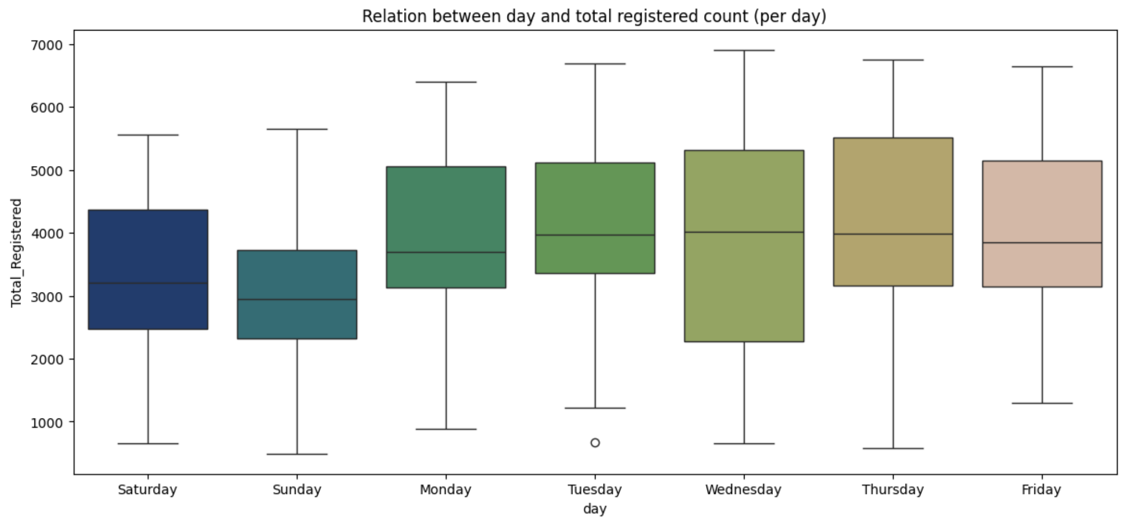
[ ]:
```
plt.figure(figsize = (14,6))
sns.boxplot(data=modf_yulu, x='day', y= 'Total_Casual', palette='gist_earth')
plt.title("Relation between day and total count (per day)")
plt.show()
```

```
<ipython-input-70-6a308a356ae7>:2: FutureWarning:

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

  sns.boxplot(data=modf_yulu, x='day', y= 'Total_Casual', palette='gist_earth')
```



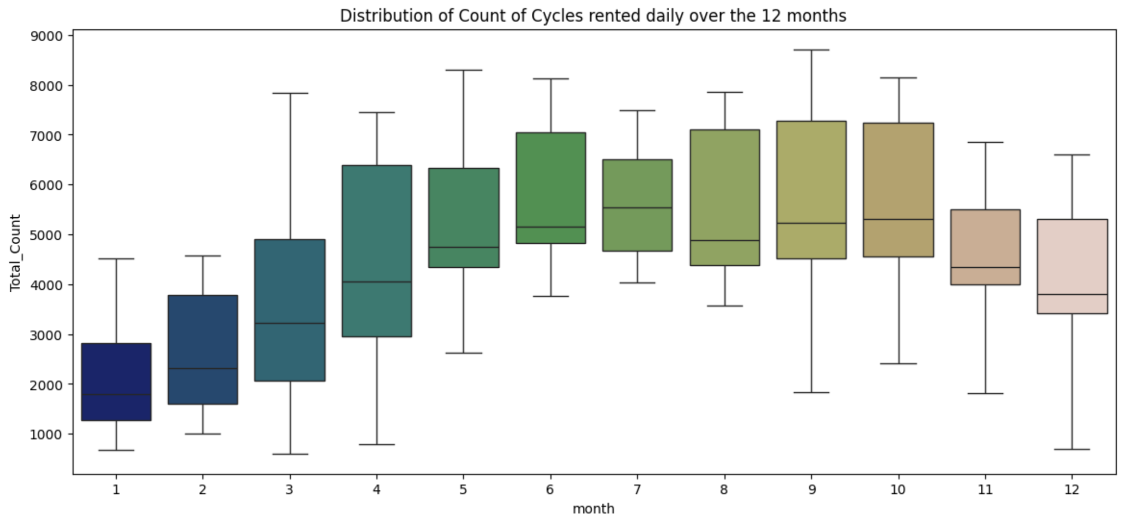
```
[ ]: plt.figure(figsize = (14,6))
     sns.boxplot(data=modf_yulu, x='day', y= 'Total_Registered',␣
      ↪palette='gist_earth')
     plt.title("Relation between day and total registered count (per day)")
     plt.show()
```

```
<ipython-input-71-1cc79a4cd26a>:2: FutureWarning:

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

  sns.boxplot(data=modf_yulu, x='day', y= 'Total_Registered',
palette='gist_earth')
```



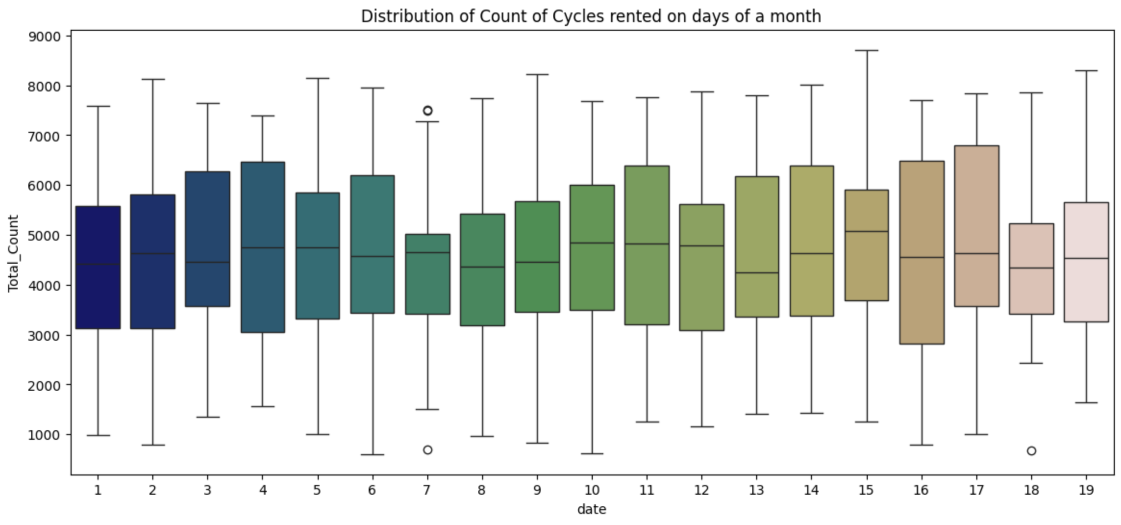
```
[ ]: plt.figure(figsize = (14,6))
     sns.boxplot(data=modf_yulu, x='month', y= 'Total_Count', palette='gist_earth')
     plt.title("Distribution of Count of Cycles rented daily over the 12 months")
     plt.show()
```

```
<ipython-input-72-5b77162cf37f>:2: FutureWarning:

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

  sns.boxplot(data=modf_yulu, x='month', y= 'Total_Count', palette='gist_earth')
```



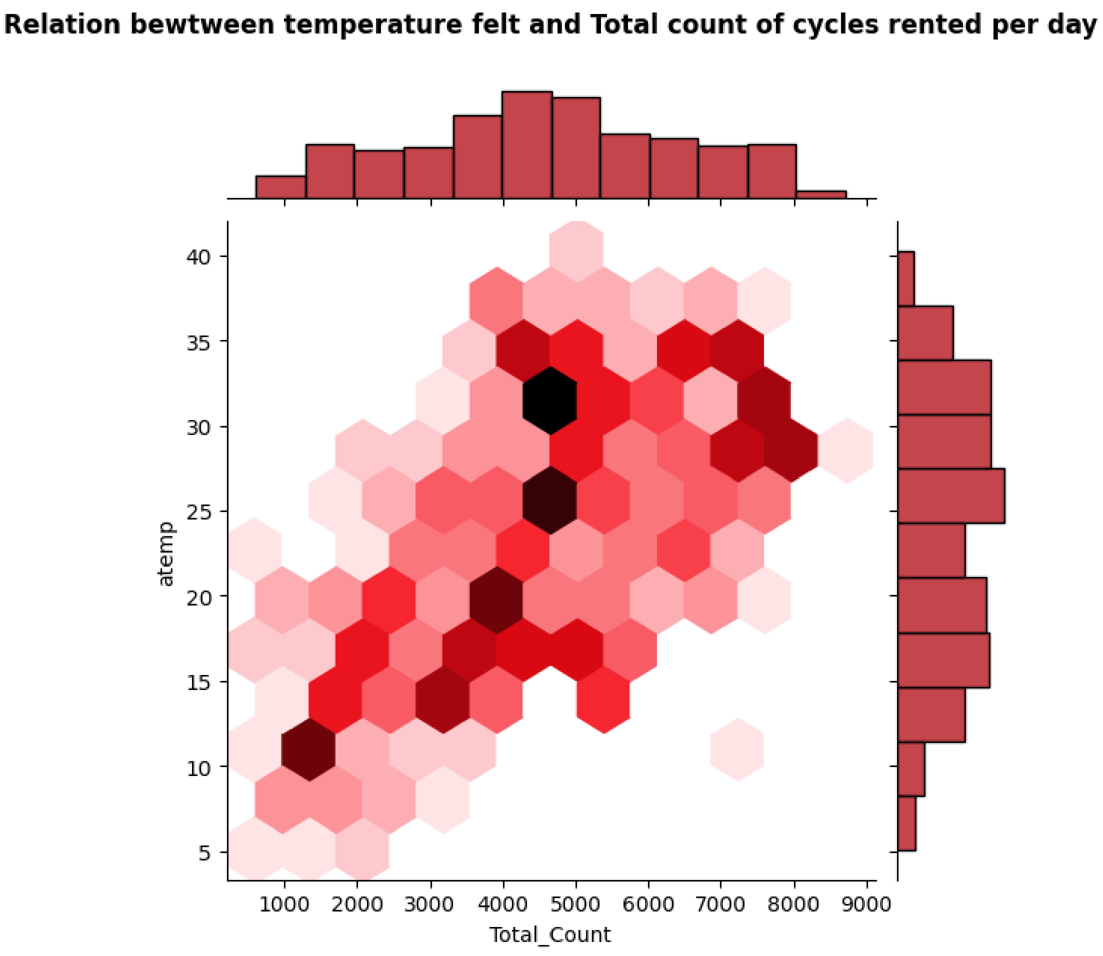
```
[ ]: plt.figure(figsize = (14,6))
     sns.boxplot(data=modf_yulu, x='date', y= 'Total_Count', palette='gist_earth')
     plt.title("Distribution of Count of Cycles rented on days of a month")
     plt.show()
```

```
<ipython-input-73-9a5dc4f89606>:2: FutureWarning:

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

  sns.boxplot(data=modf_yulu, x='date', y= 'Total_Count', palette='gist_earth')
```

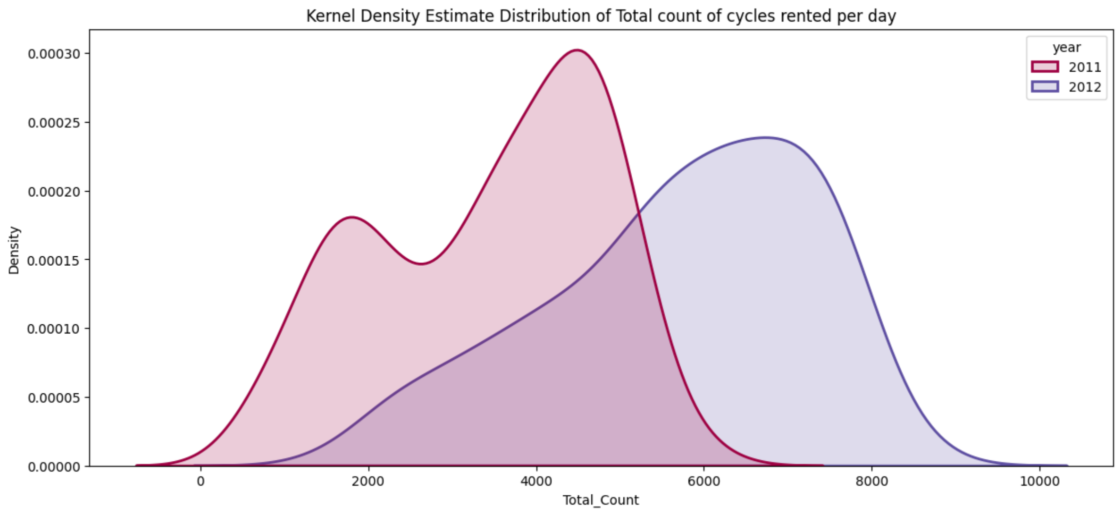


```
[ ]: plt.figure(figsize = (14,6))
     # sns.set_theme(style="ticks")
     sns.jointplot(x=modf_yulu['Total_Count'], y=modf_yulu['atemp'], kind="hex",␣
      ↪color="#b20710")
     plt.title("Relation bewtween temperature felt and Total count of cycles rented␣
      ↪per day", loc = "center",pad= 90,fontweight="bold")
     plt.show()
```

```
<Figure size 1400x600 with 0 Axes>
```



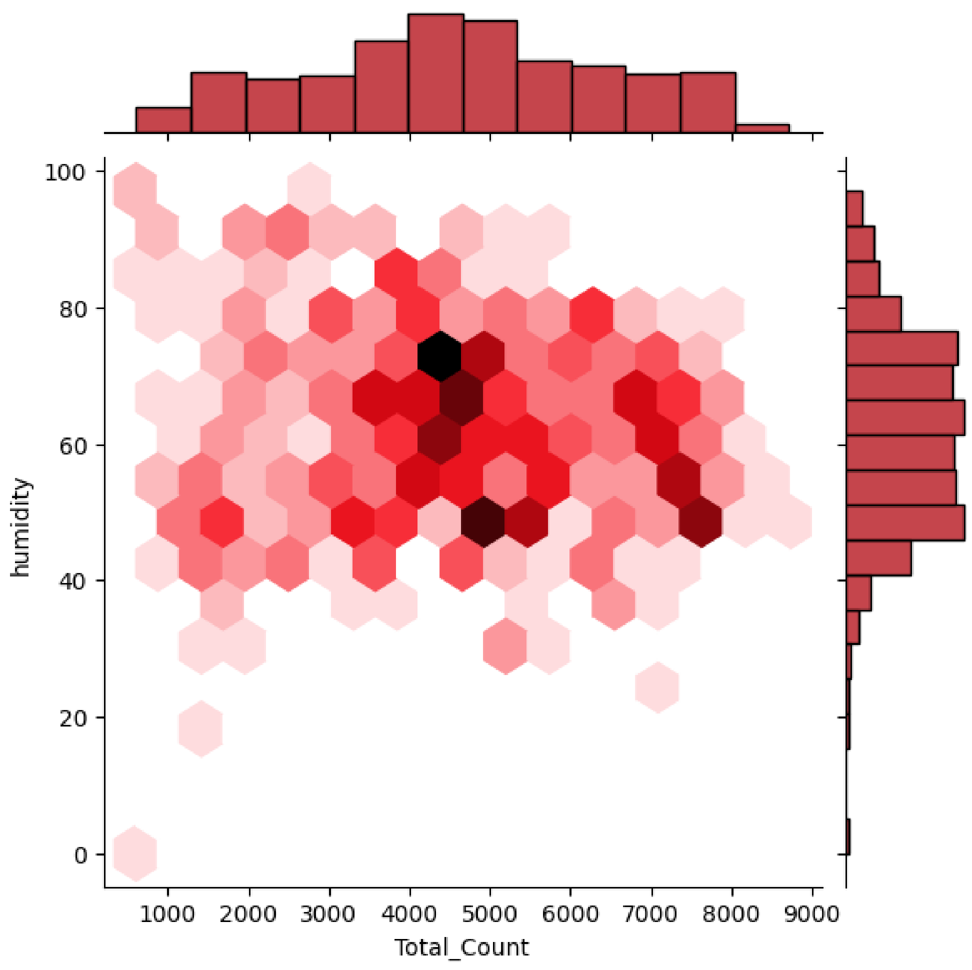
```
[ ]: plt.figure(figsize = (14,6))
     sns.kdeplot(data=modf_yulu, x="Total_Count", hue="year", fill=True,␣
      ↪common_norm=False, palette='Spectral', alpha=.2, linewidth=2)
     plt.title("Kernel Density Estimate Distribution of Total count of cycles rented␣
      ↪per day")
     plt.show()
```



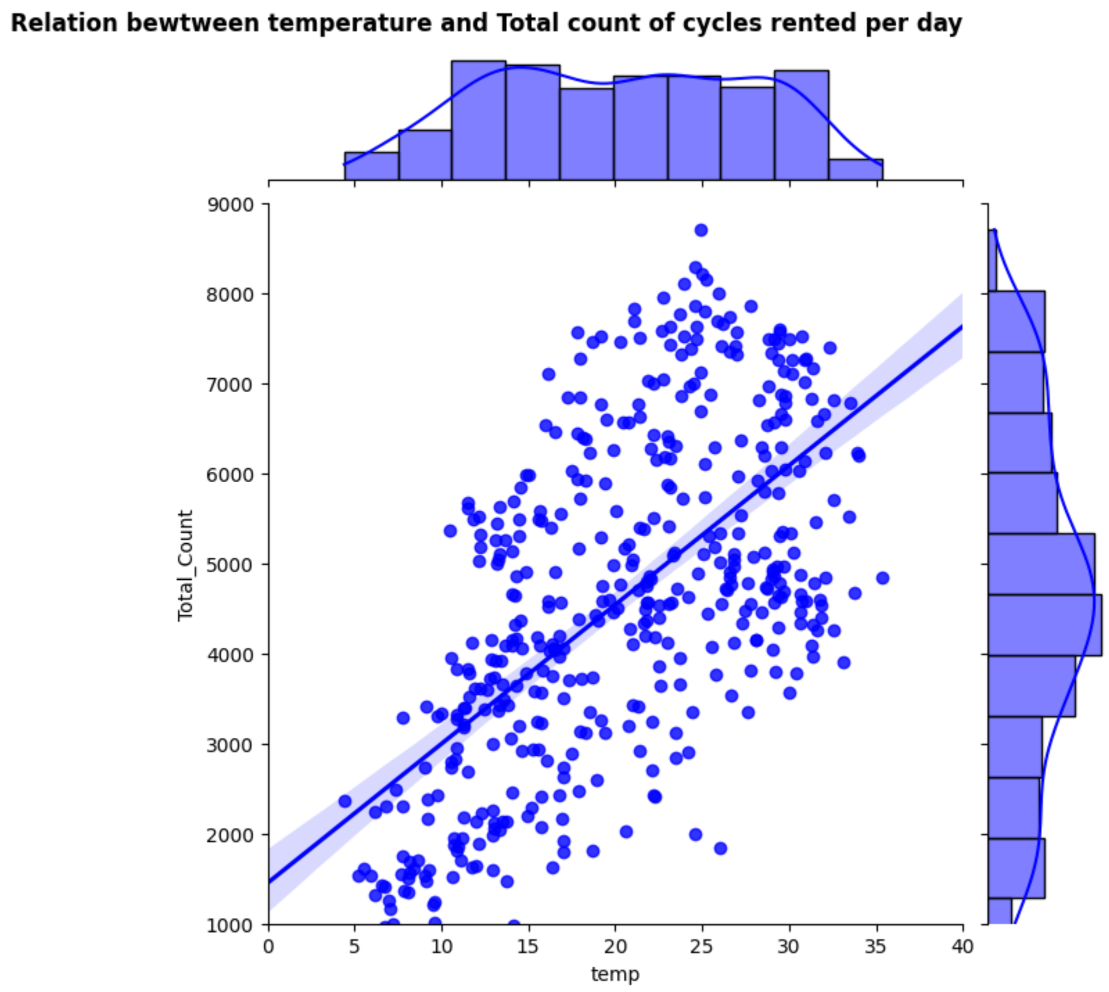
```
[ ]: plt.figure(figsize = (14,6))
     # sns.set_theme(style="ticks")
     sns.jointplot(x=modf_yulu['Total_Count'], y=modf_yulu['humidity'], kind="hex",␣
      ↪color="#b20710")
     plt.title("Relation bewtween Humidity and Total count of cycles rented per␣
      ↪day", loc = "center",pad= 90,fontweight="bold")
     plt.show()
```

```
<Figure size 1400x600 with 0 Axes>
```

Relation bewtween Humidity and Total count of cycles rented per day



```
[ ]: g = sns.jointplot(x="temp", y="Total_Count", data=modf_yulu,
                       kind="reg", truncate=False,
                       xlim=(0,40), ylim=(1000, 9000),
                       color="b", height=7)
     plt.title('Relation bewtween temperature and Total count of cycles rented per␣
      ↪day', loc = "right",pad= 90,fontweight="bold")
     plt.show()
```



#Inferences

1. **Season:** Season 3 sees the highest amount of cycles rented, followed by 2, 4 and 1.
2. **weather:** Users mostly rent cycle during weather 1, followed by 2 and 3. Weather 4 was recorded only once in the period of 2 years.
3. **temp:** Temperature is highest in Season 3, which is the season when most cycles are rented.
4. **Humidity:** Humidity is highest in season 3, which is the season when most cycles are rented.
5. **Windspeed:** Windspeed is lowest in season 3, which is the season when most cycles are rented.
6. **Count:**

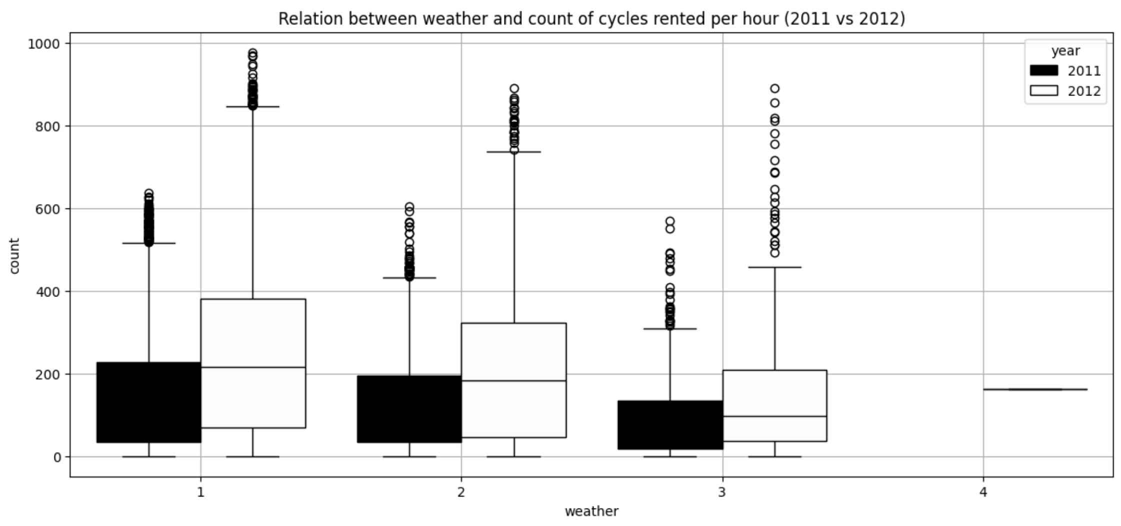
- Count of Rented cycles are lower at lower and higher temperatures and maximizes around the average temperature range **15 degrees and 27 degrees**.
- Count of rented Cycle are highest around **humidity levels of 40 to 80**. beyond those values, count of rented cycles drop.

- The daily count of rented cycles have **increased from year 2011 to 2012**.
- Cycles are rented less at temperatures lower than 10 degrees and rises with rise of temperature and again starts falling as temperature reaches 30 degrees.
- Cycles rented count are significantly lower for the first 3 months of both years, starts increasing from **months 4 to 10**, and falls again at months 11 and 12.
- **Tuesdays and Fridays** are the days when the most cycles are rented. on Sundays, the least number of cycles are rented.
- Gross Cycle rents maximizes at season 3 followed by Season 2.
- In a day, cycles are mostly rented around **5pm and 6pm**, wheras cycles are least rented around 3am and 4am.

7. **Casual:** Count of casual cycles rented falls during the weekdays and maximizes on weekends.
8. **Registered:** Count of registered cycles rented maximizes during weekdays and falls in number on weekends.

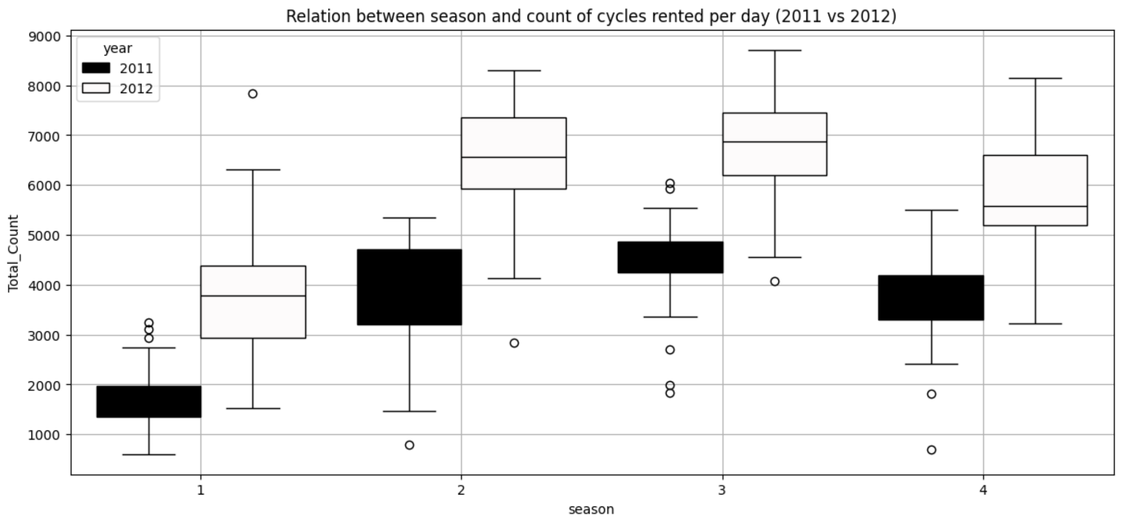
#4. **Multivariate Analysis**

```
[ ]: plt.figure(figsize = (14,6))
     sns.boxplot(data=yulu, x='weather', y= 'count', hue='year',palette='gist_earth')
     plt.title("Relation between weather and count of cycles rented per hour (2011␣
      ↪vs 2012)")
     plt.grid()
     plt.show()
```

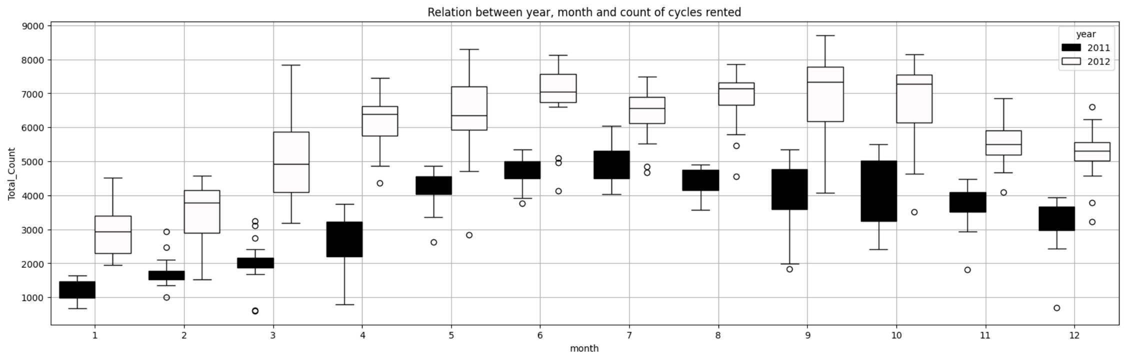


```
[ ]: plt.figure(figsize = (14,6))
     sns.boxplot(data=modf_yulu, x='season', y= 'Total_Count', hue='year',␣
      ↪palette='gist_earth')
```

```
plt.title("Relation between season and count of cycles rented per day (2011 vs␣
↪2012)")
plt.grid()
plt.show()
```

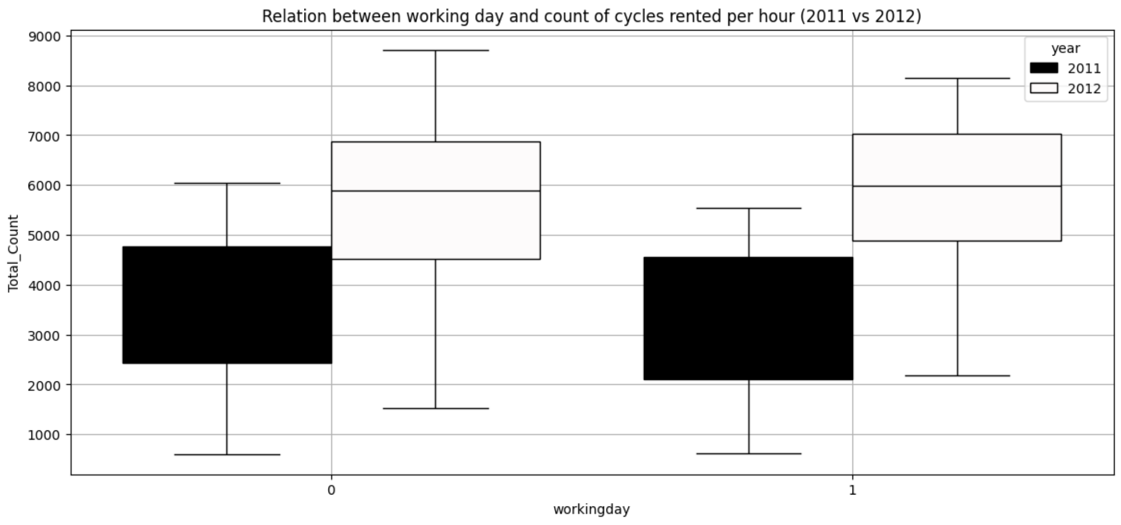


[ ]:
```
plt.figure(figsize = (21,6))
sns.boxplot(data=modf_yulu, x='month', y= 'Total_Count', hue='year',␣
↪palette='gist_earth')
plt.title("Relation between year, month and count of cycles rented")
plt.grid()
plt.show()
```



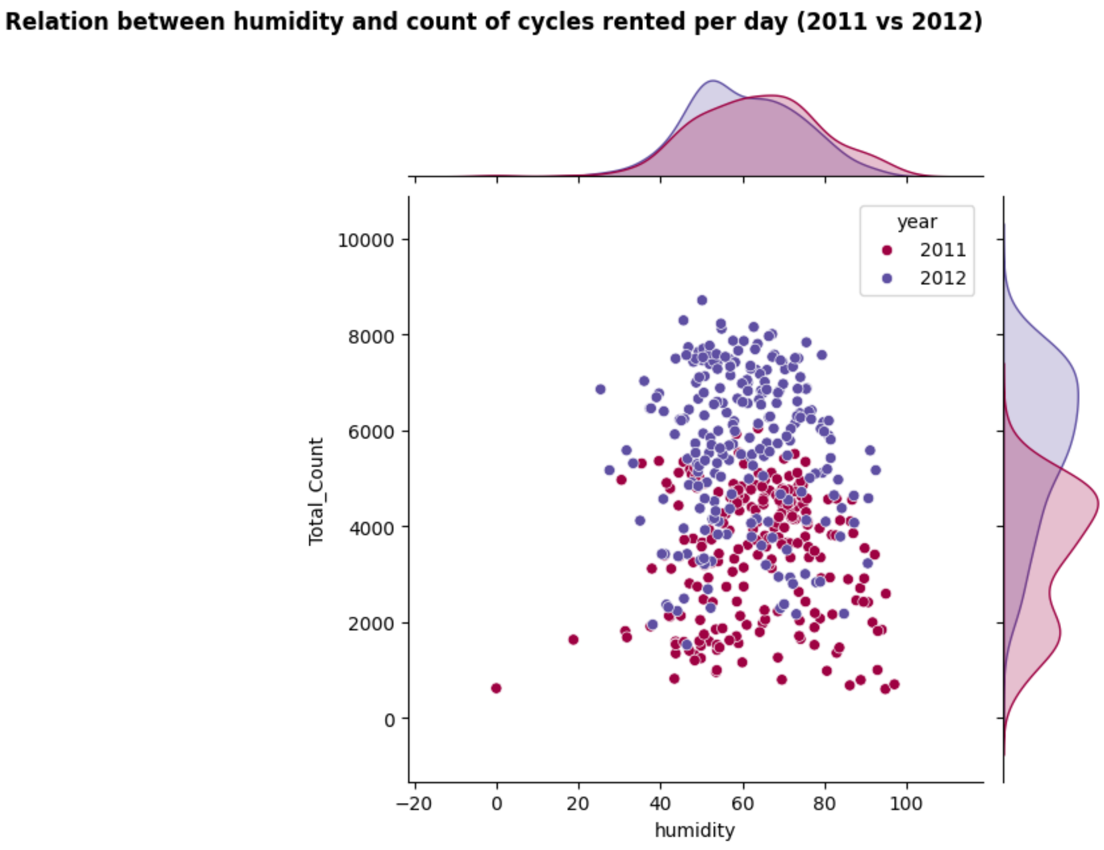
[ ]:
```
plt.figure(figsize = (14,6))
sns.boxplot(data=modf_yulu, x='workingday', y= 'Total_Count',␣
↪hue='year',palette='gist_earth')
```

```
plt.title("Relation between working day and count of cycles rented per hour␣
↪(2011 vs 2012)")
plt.grid()
plt.show()
```

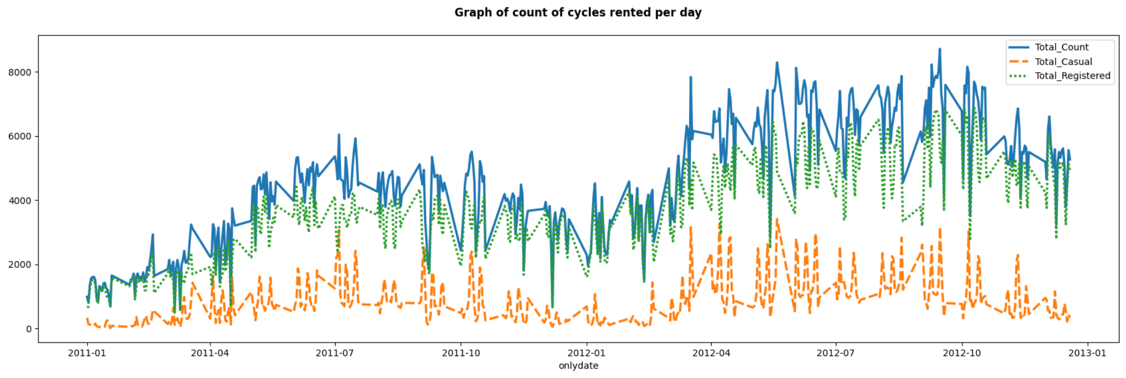


```
[ ]: plt.figure(figsize = (14,6))
     # sns.set_theme(style="white")
     sns.jointplot(data=modf_yulu, x="humidity", y="Total_Count", hue="year",␣
     ↪palette='Spectral' )
     plt.title('Relation between humidity and count of cycles rented per day (2011␣
     ↪vs 2012)', loc = "right",pad= 90,fontweight="bold")
     plt.show()
```

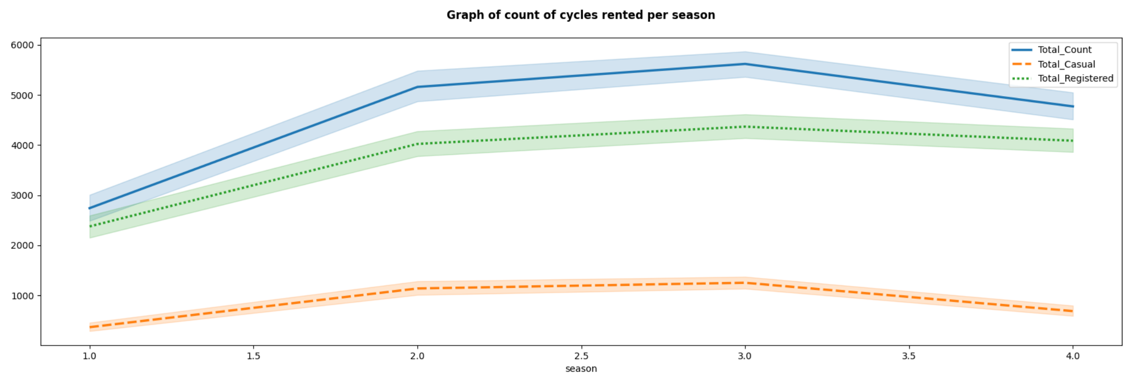
```
<Figure size 1400x600 with 0 Axes>
```



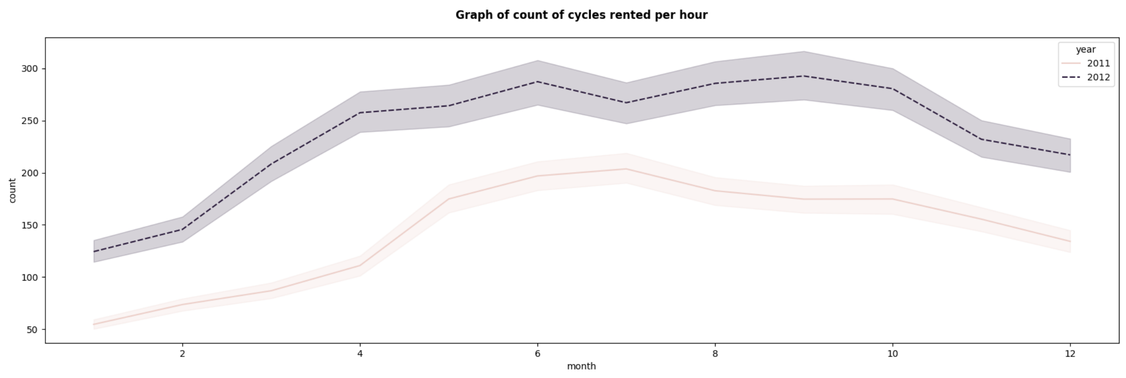
```
[ ]: data = pd.DataFrame({'onlydate':modf_yulu['onlydate'], 'Total_Count':␣
      ↪modf_yulu['Total_Count']        , 'Total_Casual': modf_yulu['Total_Casual'],␣
      ↪'Total_Registered': modf_yulu['Total_Registered'] })
     data = data.set_index('onlydate')
     plt.figure(figsize = (21,6))
     sns.lineplot(data=data, palette="tab10", linewidth=2.5)
     plt.title('Graph of count of cycles rented per day', loc = "center",pad=␣
      ↪20,fontweight="bold")
     plt.show()
```



```
[ ]: data = pd.DataFrame({'season':modf_yulu['season'], 'Total_Count':␣
      ↪modf_yulu['Total_Count']         , 'Total_Casual': modf_yulu['Total_Casual'],␣
      ↪'Total_Registered': modf_yulu['Total_Registered'] })
     data = data.set_index('season')
     plt.figure(figsize = (21,6))
     sns.lineplot(data=data, palette="tab10", linewidth=2.5)
     plt.title('Graph of count of cycles rented per season', loc = "center",pad=␣
      ↪20,fontweight="bold")
     plt.show()
```

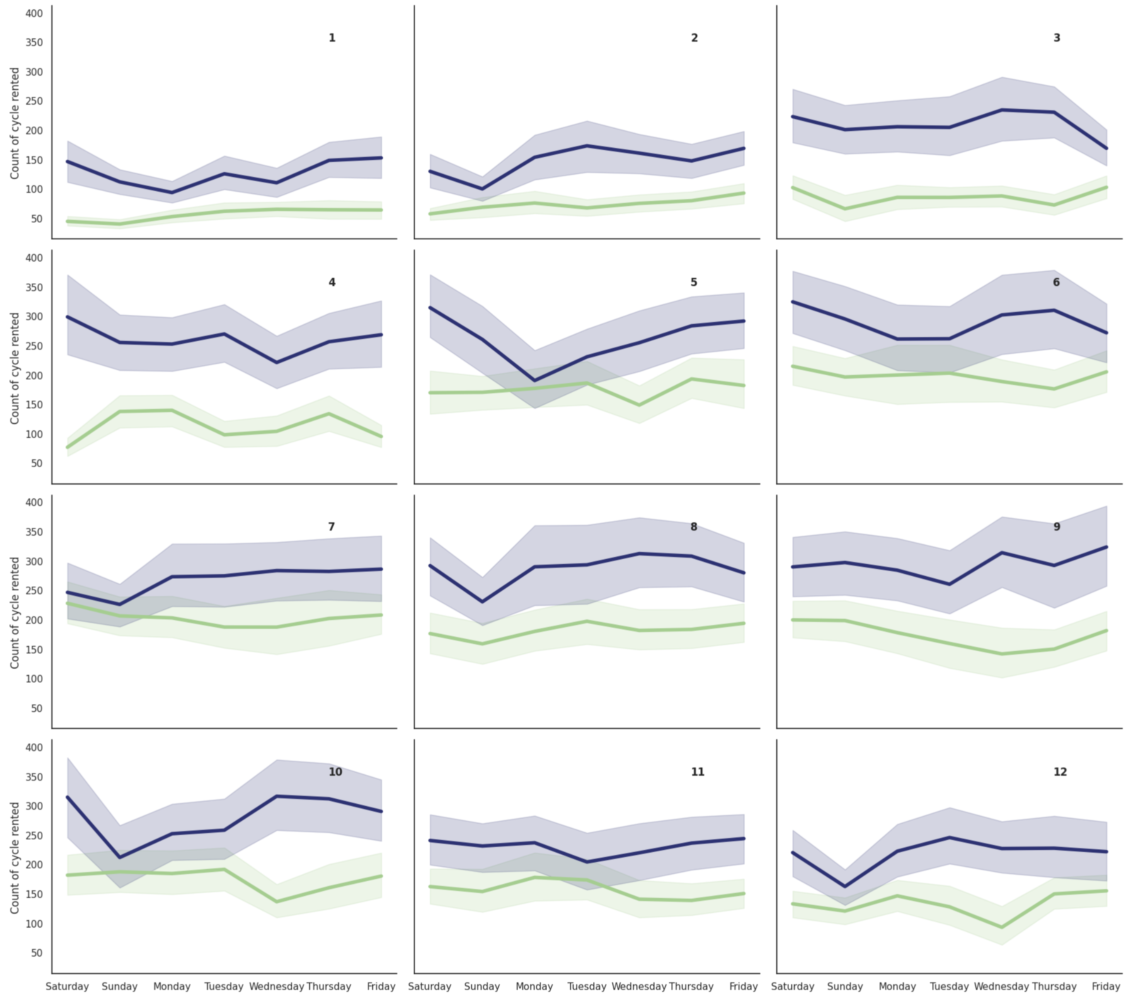


```
[ ]: plt.figure(figsize = (21,6))
     sns.lineplot(data=yulu, x="month", y="count", hue="year", style="year")
     plt.title('Graph of count of cycles rented per hour', loc = "center",pad=␣
      ↪20,fontweight="bold")
     plt.show()
```

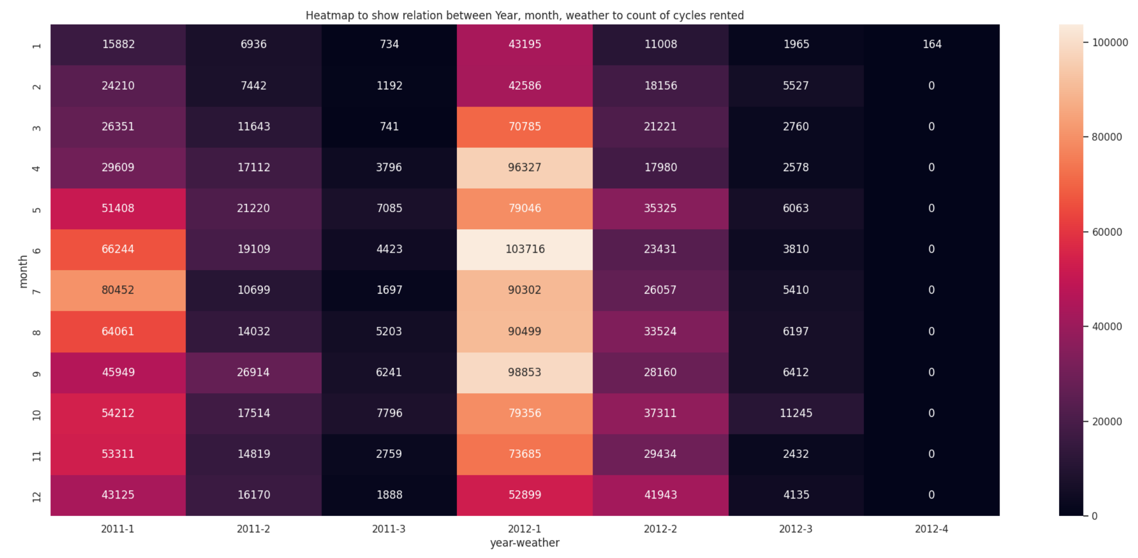


```
[ ]: plt.figure(figsize = (21,21))
     sns.set_theme(style="white")
     g = sns.relplot(
         data=yulu,
         x="day", y="count", col="month", hue="year",
         kind="line", palette="crest", linewidth=4, zorder=5,
         col_wrap=3, height=4, aspect=1.5, legend=False,
     )
     for month, ax in g.axes_dict.items():
         ax.text(.8, .85, month, transform=ax.transAxes, fontweight="bold")
     g.set_titles("")
     g.set_axis_labels("", "Count of cycle rented")
     g.tight_layout()
     # plt.title('Graph of count of cycles rented each month in 2011 and 2012')
     plt.show()
```

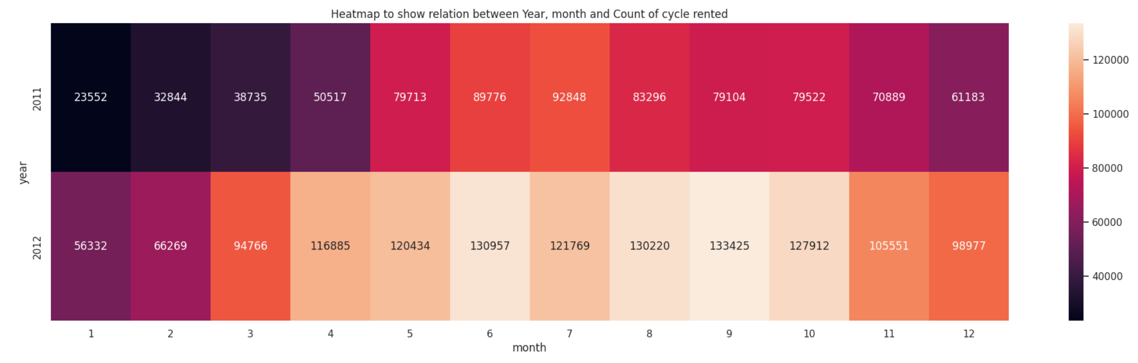
```
<Figure size 2100x2100 with 0 Axes>
```



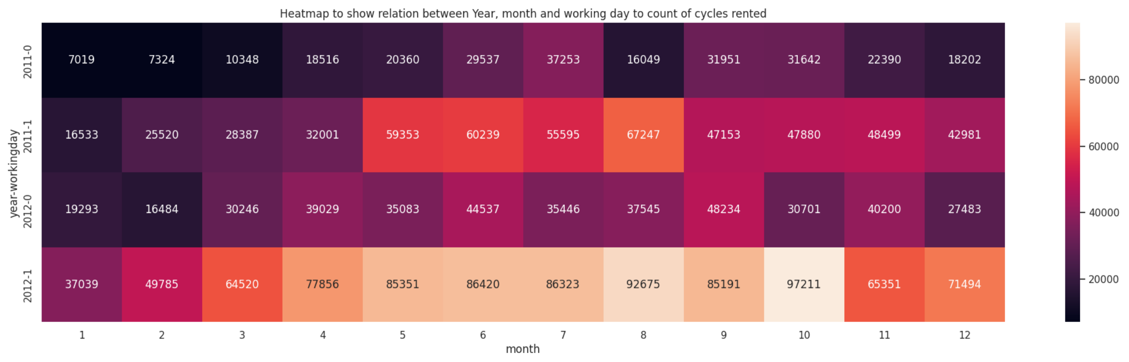
```
[ ]: ymw_count=yulu.groupby(['year','month','weather'])['count'].sum()
     data= ymw_count.unstack(level=[0,2])
     data.replace(np.nan,0, inplace=True)
     plt.figure(figsize=(24,10))
     sns.heatmap(data, annot=True, fmt='.6g')
     plt.title('Heatmap to show relation between Year, month, weather to count of␣
      ↪cycles rented')
     plt.show()
```



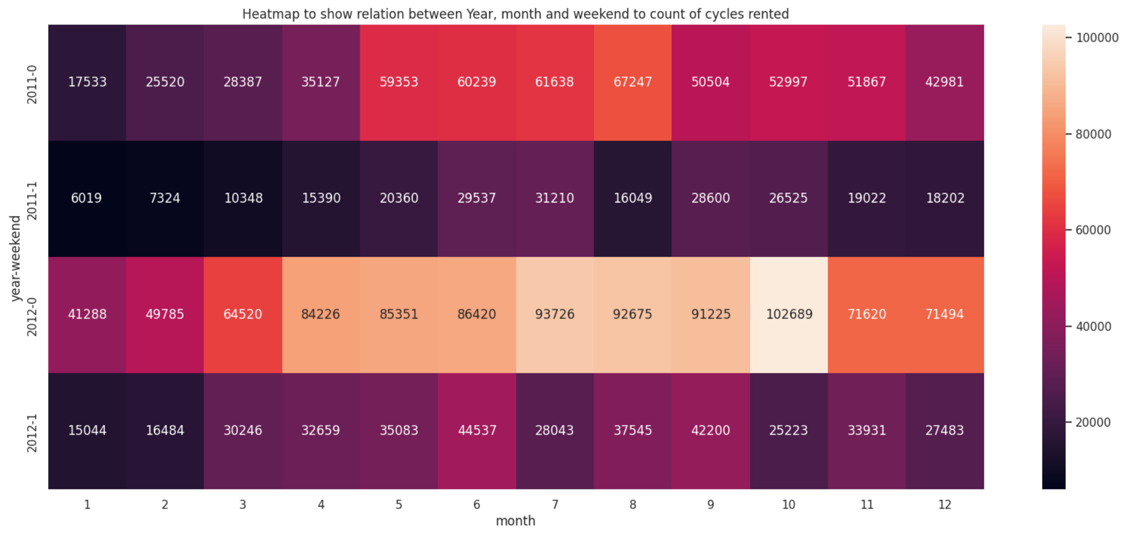
```
[ ]: ym_count=yulu.groupby(['year','month'])['count'].sum()
     data= ym_count.unstack(level=[1])
     plt.figure(figsize=(24,6))
     sns.heatmap(data, annot=True, fmt='.6g')
     plt.title('Heatmap to show relation between Year, month and Count of cycle␣
      ↪rented')
     plt.show()
```



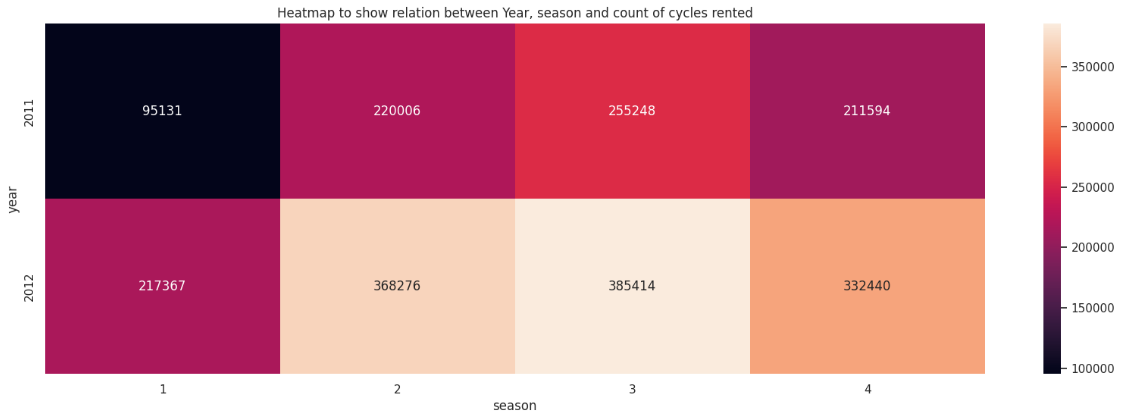
```
[ ]: ymw_count=yulu.groupby(['year','month','workingday'])['count'].sum()
     data=ymw_count.unstack(level=1)
     plt.figure(figsize=(24,6))
     sns.heatmap(data, annot=True, fmt='.6g')
     plt.title('Heatmap to show relation between Year, month and working day to␣
       ↪count of cycles rented')
     plt.show()
```



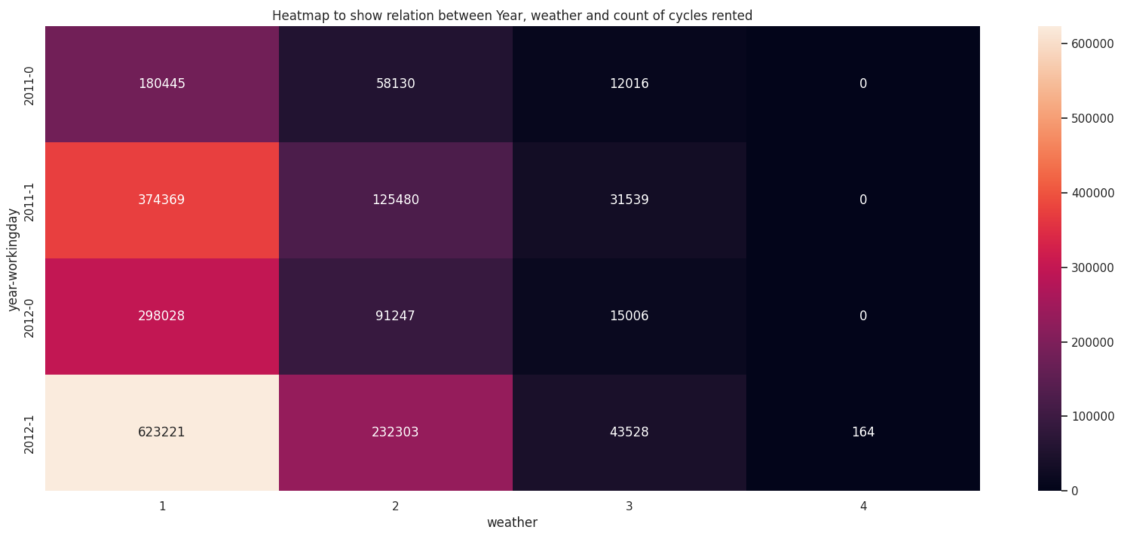
```
[ ]: ymwk_count=yulu.groupby(['year','month','weekend'])['count'].sum()
     data=ymwk_count.unstack(level=1)
     data.replace(np.nan,0, inplace=True)
     plt.figure(figsize=(20,8))
     sns.heatmap(data=data, annot=True, fmt ='.6g')
     plt.title('Heatmap to show relation between Year, month and weekend to count of␣
       ↪cycles rented')
     plt.show()
```



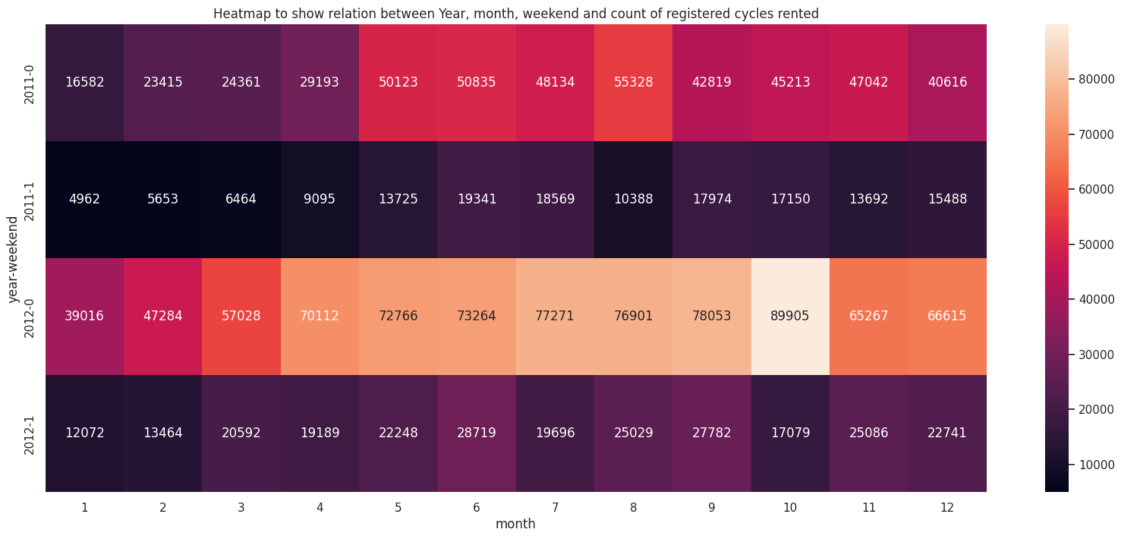
```
[ ]: ys_count=yulu.groupby(['year','season'])['count'].sum()
     data= ys_count.unstack(level=[1])
     data.replace(np.nan,0, inplace=True)
     plt.figure(figsize=(20,6))
     sns.heatmap(data, annot=True, fmt='.6g')
     plt.title('Heatmap to show relation between Year, season and count of cycles␣
      ↪rented')
     plt.show()
```



```
[ ]: yww_count=yulu.groupby(['year','weather','workingday'])['count'].sum()
     data=yww_count.unstack(level=1)
     data.replace(np.nan,0, inplace=True)
     plt.figure(figsize=(20,8))
     sns.heatmap(data=data, annot=True, fmt='.6g')
     plt.title('Heatmap to show relation between Year, weather and count of cycles␣
      ↪rented')
     plt.show()
```

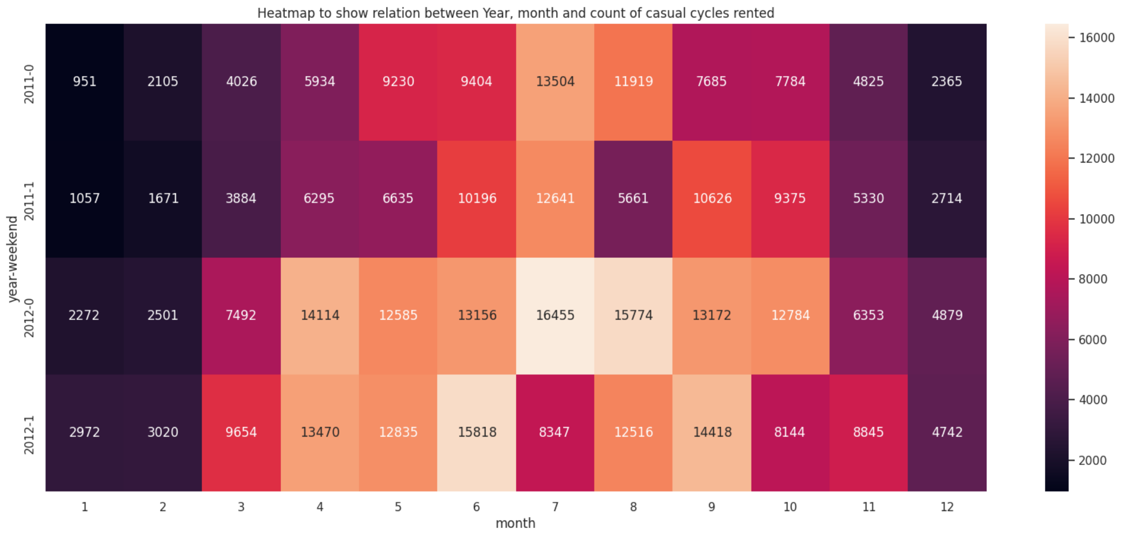


```
[ ]: ymw=yulu.groupby(['year','month','weekend'])['registered'].sum()
     data= ymw.unstack(level=1)
     data.replace(np.nan,0, inplace=True)
     plt.figure(figsize=(20,8))
     sns.heatmap(data=data, annot=True, fmt='.6g')
     plt.title('Heatmap to show relation between Year, month, weekend and count of␣
       ↪registered cycles rented')
     plt.show()
```



```
[ ]: ymw=yulu.groupby(['year','month','weekend'])['casual'].sum()
     data= ymw.unstack(level=1)
```

```
data.replace(np.nan,0, inplace=True)
plt.figure(figsize=(20,8))
sns.heatmap(data=data, annot=True, fmt='.6g')
plt.title('Heatmap to show relation between Year, month and count of casual␣
  ↪cycles rented')
plt.show()
```



#Inferences

1. **Season:** Out of the 2 years, season 1 of year 2011, showed the least gross cycle rented.
2. **weather:** Users mostly rent cycle during weather 1, followed by 2. Users renting bikes in Weather 3 and 4 are comparitively very less in number, even though this number improved from 2011 to 2012.
3. **Atemp:** With lower temperature count of registered cycles are less. With increasing temperature the number of cycle rent count increases uptill 35 degree celsius.
4. **Count:**

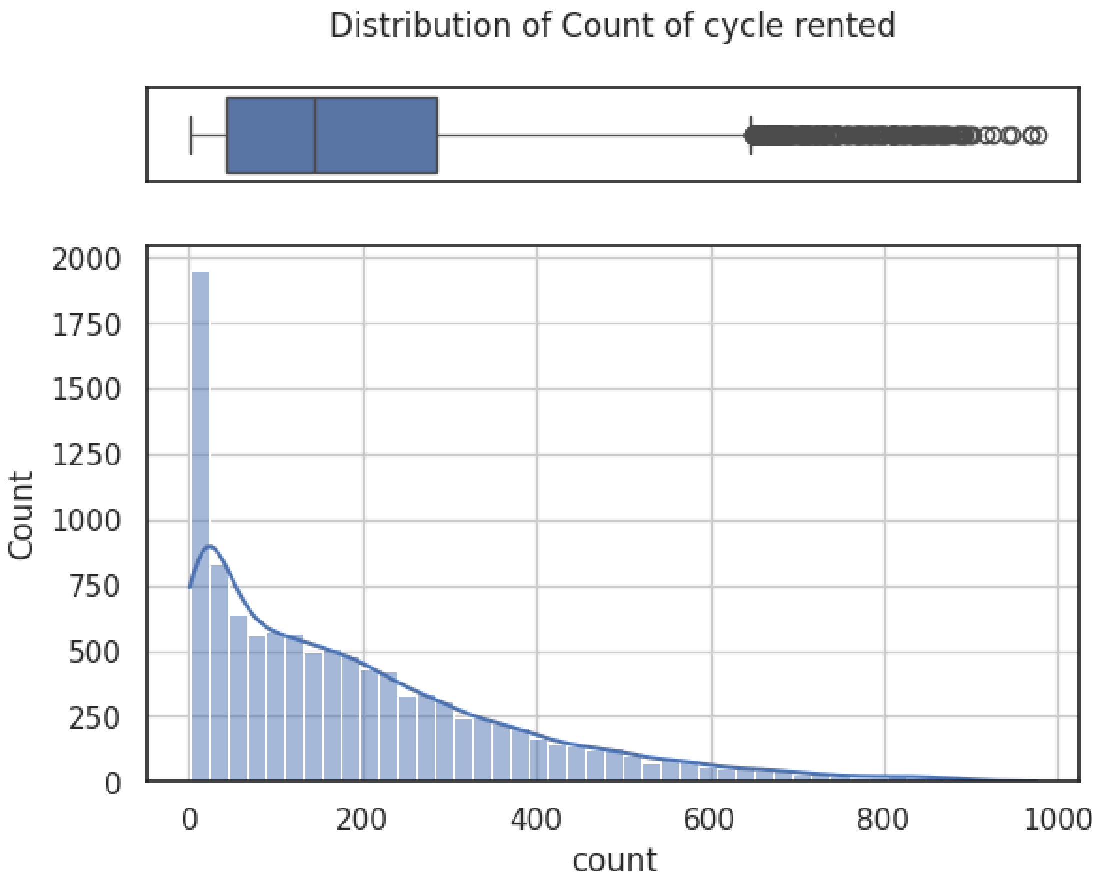
- Out of the 24 months, the count of total rented bikes is highest in **month 9 of year 2012** and lowest in month 1 of 2011.
- **Season 3** shows maximum count of rented bikes per hour.
- **Weather 1** shows the maximum count of rented bikes.

5. **Count of registered Bikes:** Count of registered bikes over weekend are less as compared to count of casual bikes.
6. **Count of Casual Bikes:** Count of casual bikes over weekend are much higher in number than on working days.

#OUTLIER TREATMENT

```
[ ]: f, (ax_box, ax_hist) = plt.subplots(2, sharex=True,␣
       ↪gridspec_kw={"height_ratios": (.15, .85)})
```

```
sns.boxplot(data=yulu,x='count', ax=ax_box)
sns.histplot(data=yulu, x="count", ax=ax_hist,kde=True)
ax_box.set(xlabel='')
plt.title('Distribution of Count of cycle rented', pad=80)
plt.grid()
plt.show()
```



#**Inferences**

1. We can see that there are outliers in our data in the count (count of cycles rented) variable, which means there are hours where very high number of bikes were rented which vary significantly from the other observations.
2. Outliers affect our data analysis while finding distribution types or even confidence intervals. So, we will performed modifications to the data to remove outliers.
3. After performing, we can see the number of data points identified as outliers are very low.

#**CLT and Confidence Intervals**

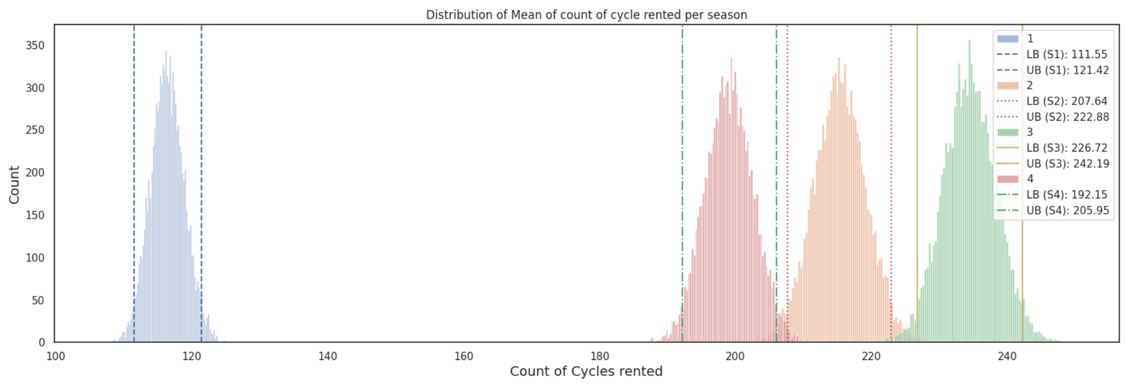
A confidence interval is a range of values above and below a point estimate that captures the true population parameter at some predetermined confidence level. For example, if we want to have a 95% chance of capturing the true population parameter with a point estimate and a corresponding confidence interval, you'd set your confidence level to 95%. Higher confidence levels result in a wider confidence intervals.

The means of the samples in a set of samples (the sample means) will be approximately normally distributed. This normal distribution will have a mean close to the mean of the population.

The variance of the sample means will be close to the variance of the population divided by the sample size.

A) Confidence interval mean of the number of Cycles rented per SEASON.

```
[ ]: plt.figure(figsize = (20,6))
     u_season= yulu['season'].unique()
     style = ['--', ':', '-', '-.']
     color = ['b', 'r','y','g']
     for k in range(len(u_season)):
       r = 10000
       data = yulu[(yulu["season"] == u_season[k])] ["count"]
       size = 2500
       bs_means = np.empty(r)
       for i in range(r):
           bs_sample = np.random.choice(data, size=size)
           bs_means[i] = np.mean(bs_sample)
       plt.hist(bs_means, bins=100, alpha=0.5 ,label= u_season[k])
       plt.axvline(x = round(np.percentile(bs_means,2.5), 2), color = color[k],␣
      ↪linestyle= style[k], label = f"LB (S{u_season[k]}): {round(np.
      ↪percentile(bs_means,2.5), 2)}")
       plt.axvline(x = round(np.percentile(bs_means,97.5), 2), color = color[k],␣
      ↪linestyle= style[k], label =f"UB (S{u_season[k]}): {round(np.
      ↪percentile(bs_means,97.5), 2)}")
       plt.xlabel("Count of Cycles rented", size=14)
       plt.ylabel("Count", size=14)
       plt.title(f"Distribution of Mean of count of cycle rented per season")
       plt.legend(loc='upper right')
       plt.grid()
```

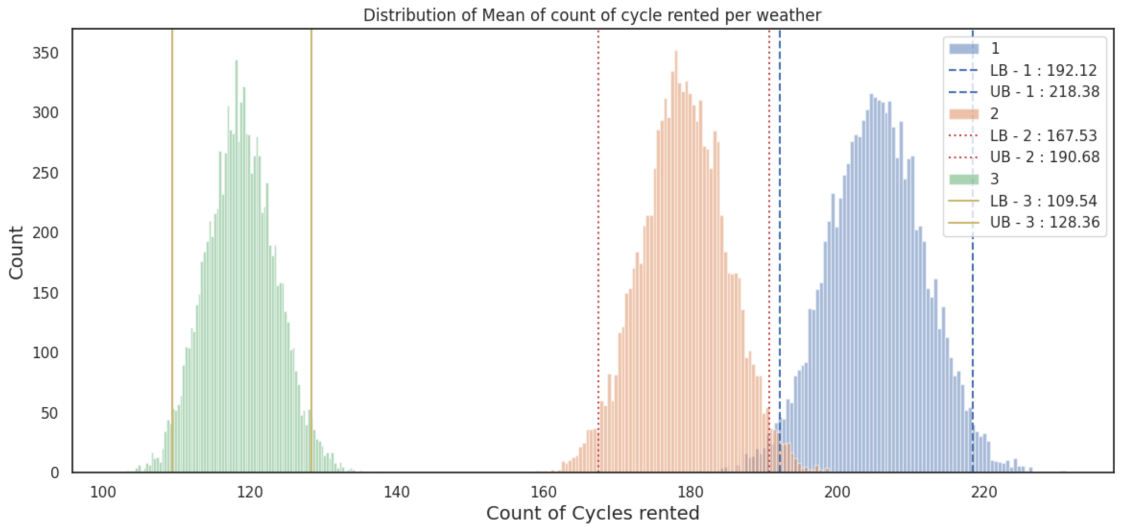


Inference:

As, indicated by the plot above, the mean count of bikes rented on an hourly basis in **season 3 is greatest**, followed by season 2, 4 and **least in season 1**. **The 95% CI for bike count in season 1 is [111-121], in season 2 is [207-222], in season 3 is [226-242] and in season 4 is [192-205].**

B) Confidence interval mean of the number of Cycles rented per WEATHER.

```
[ ]: yulu_m = yulu[yulu['weather']!=4]
     plt.figure(figsize = (14,6))
     u_citycat= yulu_m['weather'].unique()
     style = ['--', ':', '-']
     color = ['b', 'r','y']
     for k in range(len(u_citycat)):
       r = 10000
       data = yulu_m[(yulu_m["weather"] == u_citycat[k])] ["count"]
       size = 800
       bs_means = np.empty(r)
       for i in range(r):
           bs_sample = np.random.choice(data, size=size)
           bs_means[i] = np.mean(bs_sample)
       plt.hist(bs_means, bins=100, alpha=0.5 ,label= u_citycat[k])
       plt.axvline(x = round(np.percentile(bs_means,2.5), 2), color = color[k],␣
      ↪linestyle= style[k], label = f"LB - {u_citycat[k]} : {round(np.
      ↪percentile(bs_means,2.5), 2)}")
       plt.axvline(x = round(np.percentile(bs_means,97.5), 2), color = color[k],␣
      ↪linestyle= style[k], label = f"UB - {u_citycat[k]} : {round(np.
      ↪percentile(bs_means,97.5), 2)}")
       plt.xlabel("Count of Cycles rented", size=14)
       plt.ylabel("Count", size=14)
       plt.title(f"Distribution of Mean of count of cycle rented per weather")
       plt.legend(loc='upper right')
```

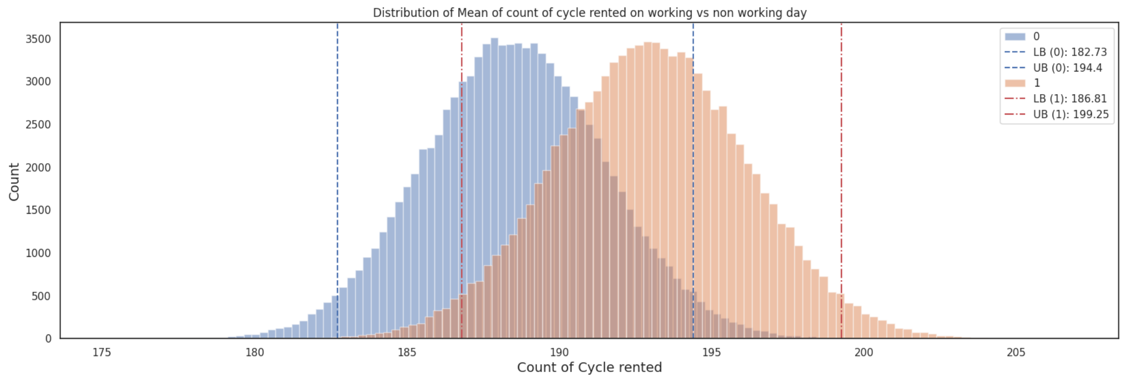


Inference:

As, indicated by the plot above, the mean count of bikes rented on an hourly basis in **weather 1 is greatest**, followed by weather 2 and **least in weather 3**. **The 95% CI for bike count in weather 1 is [192-218], in weather 2 is [167-190] and in weather 3 is [109-128].**

C) Confidence interval mean of the number of Cycles rented Working Day vs Non-Working Day.

```
[ ]: plt.figure(figsize = (20,6))
     u_workingday= yulu['workingday'].unique()
     style = ['--','-.']
     color = ['b', 'r']
     for k in range(len(u_workingday)):
       r = 100000
       data = yulu[(yulu["workingday"] == u_workingday[k])]["count"]
       size = 3400
       bs_means = np.empty(r)
       for i in range(r):
           bs_sample = np.random.choice(data, size=size)
           bs_means[i] = np.mean(bs_sample)
       plt.hist(bs_means, bins=100, alpha=0.5 ,label= u_workingday[k])
       plt.axvline(x = round(np.percentile(bs_means,2.5), 2), color = color[k],␣
      ↪linestyle= style[k], label = f"LB ({u_workingday[k]}): {round(np.
      ↪percentile(bs_means,2.5), 2)}")
       plt.axvline(x = round(np.percentile(bs_means,97.5), 2), color = color[k],␣
      ↪linestyle= style[k], label =f"UB ({u_workingday[k]}): {round(np.
      ↪percentile(bs_means,97.5), 2)}")
       plt.xlabel("Count of Cycle rented", size=14)
       plt.ylabel("Count", size=14)
       plt.title(f"Distribution of Mean of count of cycle rented on working vs non␣
      ↪working day")
       plt.legend(loc='upper right')
       plt.grid()
```

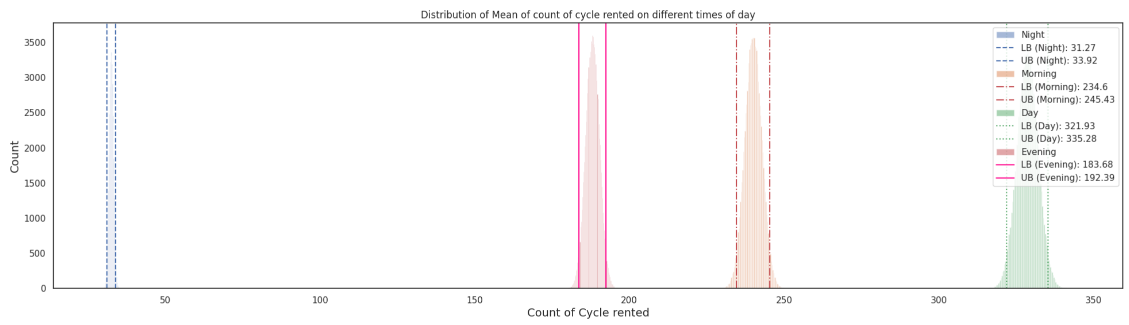


Inference:

As, indicated by the plot above, the mean count of bikes rented on an hourly basis is higher for working days and tends to be lower for non-working days. but the confidence intervals are somewhat overlapping. *The 95% CI for bike count in working day is [186-199] and in non working day is [182-194].**

D) Confidence interval mean of the number of Cycles rented in different times of day.

```
[ ]: import datetime as dt
     yulu_copy=yulu.copy()
     # yulu_copy['time'] = yulu_copy['time'].astype(str).apply(lambda x: x[:2]).
      ↪astype(int)
     bins = [-1,6,12,18,24]
     labels =["Night", "Morning", "Day", "Evening"]
     yulu_copy['time_catg'] = pd.cut(yulu_copy['time'], bins,labels=labels)

     plt.figure(figsize = (24,6))
     u_tc= yulu_copy['time_catg'].unique()
     style = ['--','-.', ':', '-']
     color = ['b', 'r','g', 'deeppink']
     for k in range(len(u_tc)):
       r = 100000
       data = yulu_copy[(yulu_copy["time_catg"] == u_tc[k])]["count"]
       size = 3400
       bs_means = np.empty(r)
       for i in range(r):
           bs_sample = np.random.choice(data, size=size)
           bs_means[i] = np.mean(bs_sample)
       plt.hist(bs_means, bins=100, alpha=0.5 ,label= u_tc[k])
       plt.axvline(x = round(np.percentile(bs_means,2.5), 2), color = color[k],␣
      ↪linestyle= style[k], label = f"LB ({u_tc[k]}): {round(np.
      ↪percentile(bs_means,2.5), 2)}")
       plt.axvline(x = round(np.percentile(bs_means,97.5), 2), color = color[k],␣
      ↪linestyle= style[k], label =f"UB ({u_tc[k]}): {round(np.
      ↪percentile(bs_means,97.5), 2)}")
       plt.xlabel("Count of Cycle rented", size=14)
       plt.ylabel("Count", size=14)
       plt.title(f"Distribution of Mean of count of cycle rented on different times␣
      ↪of day")
       plt.legend(loc='upper right')
       plt.grid()
```



Inference:

As, indicated by the plot above, the mean count of bikes rented on an hourly basis is **higher during the day (12pm - 6pm)**, followed by morning (6am - 12pm), then evening (6pm - 12pm) and then **least at night (12pm - 6am)**. **The 95% CI for bike count in morning is [234 - 245] and during day is [321 - 335] and during evening is [183-192] and during night is [31 - 34].**

#Hypothesis Testing

Setting up a function to return result on the basis of the significance value(0.05).

```
[ ]: def htResult(p_value):
         significance_level = 0.05
         if p_value <= significance_level:
             print('Reject NULL HYPOTHESIS')
         else:
             print('Fail to Reject NULL HYPOTHESIS')
```

#Question -1 : Working Day has effect on number of electric cycles rented or not? Using 2- Sample T-Test

Mathematically, the t-test takes a sample from each of the two sets and establishes the problem statement. It assumes a null hypothesis that the two means are equal.

Using the formulas, values are calculated and compared against the standard values. The assumed null hypothesis is accepted or rejected accordingly. If the null hypothesis qualifies to be rejected, it indicates that data readings are strong and are probably not due to chance. The correlated t-test, or paired t-test, is a dependent type of test and is performed when the samples consist of matched pairs of similar units, or when there are cases of repeated measures. For example, there may be instances where the same patients are repeatedly tested before and after receiving a particular treatment. Each patient is being used as a control sample against themselves.

This method also applies to cases where the samples are related or have matching characteristics, like a comparative analysis involving children, parents, or siblings.

STEP-1 : Set up Null Hypothesis

- **Null Hypothesis (H0)** - number of electric cycles rented on working days and non working days are same.

- **Alternate Hypothesis (HA)** - number of electric cycles rented on working days and non working days are different.

STEP-2 : Checking for basic assumpitons for the hypothesis

- Distribution check using **QQ Plot**

- Homogeneity of Variances using **Lavene's test**

STEP-3: Define Test statistics; Distribution of T under H0.

- We know that the test statistic while performing a T-Test follows Tdistribution.

STEP-4: Decide the kind of test.

- We will be performing **Two tailed t-test**

STEP-5: Compute the p-value and fix value of alpha.

- We will be com puting the t-test value using the ttest function using scipy.stats.
- We set our **alpha to be 0.05**

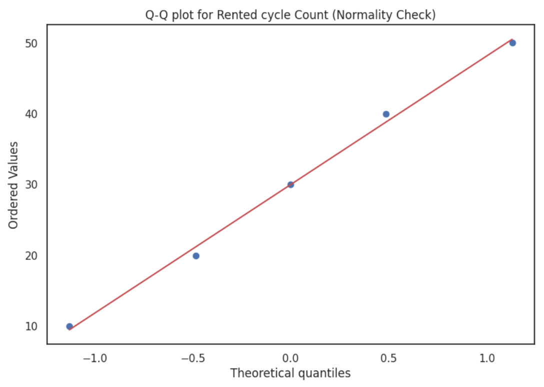
STEP-6: Compare p-value and alpha.

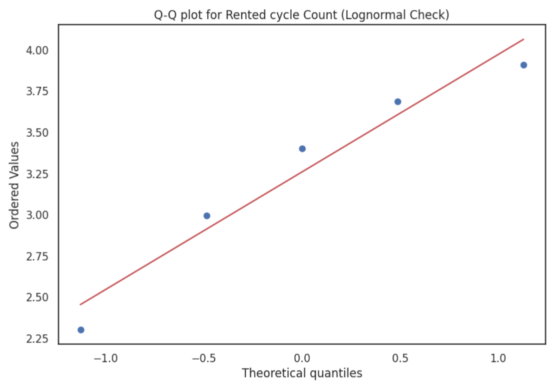
- Based on p-value, we will accept or reject H0.

1. **p-val** $>$ **alpha** : Accept H0
2. **p-val** $<$ **alpha** : Reject H0

A) QQ-Plot for Distribution check.

```
[ ]: outlier_treated_yulu = pd.DataFrame({'count': [10, 20, 30, 40, 50]})  # Replace␣
     ↪with your actual data

     plt.figure(figsize=(20,6))
     plt.subplot(1,2,1)
     stats.probplot(outlier_treated_yulu['count'], plot= plt, dist="norm")
     plt.title('Q-Q plot for Rented cycle Count (Normality Check)')

     plt.subplot(1,2,2)
     stats.probplot(outlier_treated_yulu['count'].apply(np.log), plot= plt,␣
     ↪dist="norm")
     plt.title('Q-Q plot for Rented cycle Count (Lognormal Check)')

     plt.show()
```





The graph doesnt look like a normal gaussian distribution. it doesnt also satisfy the QQ - Plots.

We know, the **count** random variable, is actually count of the number of cycles rented in an hourly basis.

we also know, that the **Poisson distribution** is a discrete probability distribution that expresses the probability of a given number of **events occurring in a fixed interval of time** or space if these events occur with a known constant mean rate and independently of the time since the last event.

So, we will go and try doing a Poisson distribution check.

Also, we need to keep in mind, there are a lot of factors, that would affect the Count variable, like weather, time of day, temperature, working day or not, season, etc.

so, intutively we can assume that, if we look at data for a particular kind of time, like if we consider only working days, the count variable might be poisson distributed. let's go ahead and check it.

[134]:
```
import datetime as dt
d1 = dt.datetime(2012, 4, 1, 11, 00, 00)
d2 = dt.datetime(2012, 6, 30, 15, 00, 00)
```

B) Homogeneity of Variances using Levene's test

- **Null Hypothesis(H0)** - Homogenous Variance
- **Alternate Hypothesis(HA)** - Non Homogenous variance

[5]:
```
from scipy.stats import levene
workingday = yulu[yulu['workingday']==1]['count']
nonworkingday = yulu[yulu['workingday']==0]['count']
stat,p = levene(workingday,nonworkingday)
```

[6]:
```
print('P-value :',p)
```

```
P-value : 0.9437823280916695
```

Fail to Reject NULL HYPOTHESIS

- We fail to reject null hypothesis, which means variance is same/similar.

C) Performing two tailed t-test:

[10]:
```
workingday = yulu[yulu['workingday']==1]['count']
nonworkingday = yulu[yulu['workingday']==0]['count']
```

[12]:
```
import scipy.stats as stats
st,p = stats.ttest_ind(workingday,nonworkingday)
print('P-value :',(p))
```

```
P-value : 0.22644804226361348
```

Fail to Reject NULL HYPOTHESIS

We fail to reject null hypothesis, which means the number of electric cycles rented on working days and non working days are similar.

1 Question 2: Number of cycles rented:- are the numbers similar or different in different seasons? - ANOVA test

The one-way ANOVA compares the means between the groups you are interested in and determines whether any of those means are statistically significantly different from each other.

Specifically, it tests the null hypothesis (H0):

µ1 = µ2 = µ3 = ….. = µk

where, µ = group mean and k = number of groups.

If, however, the one-way ANOVA returns a statistically significant result, we accept the alternative hypothesis (HA), which is that there are at least two group means that are statistically significantly different from each other.

STEP-1 : Set up Null Hypothesis

- Null Hypothesis (H0) - Mean of cycle rented per hour is same for season 1,2,3 and 4 are same.
- Alternate Hypothesis (HA) -Mean of cycle rented per hour is same for season 1,2,3 and 4 are different.

STEP-2 : Checking for basic assumpitons for the hypothesis

1. Normality check using QQ Plot. If the distribution is not normal, use BOX-COX transform to transform it to normal distribution.
2. Homogeneity of Variances using Lavene's test
3. Each observations are independent.

STEP-3: Define Test statistics; Distribution of T under H0.

The test statistic for a One-Way ANOVA is denoted as F. For an independent variable with k groups, the F statistic evaluates whether the group means are significantly different.

F=MSR/MSE

Under H0, the test statistic should follow **F-Distribution.**

STEP-4: Decide the kind of test.

We will be performing **right tailed t-test**

STEP-5: Compute the p-value and fix value of alpha.

we will be computing the anova-test p-value using the **f_oneway** function using scipy.stats. We set our alpha to be **0.05**

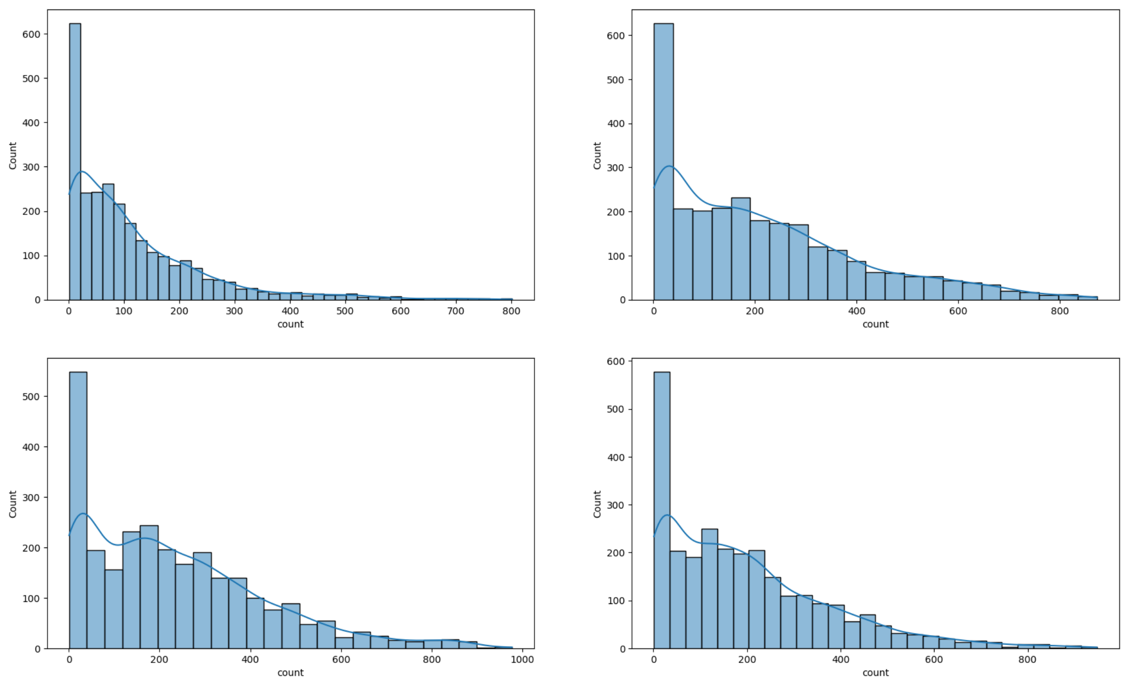
STEP-6: Compare p-value and alpha.

Based on p-value, we will accept or reject H0. p-val > alpha : Accept H0 p-val < alpha : Reject H0

A) Normality Check using QQ-Plots

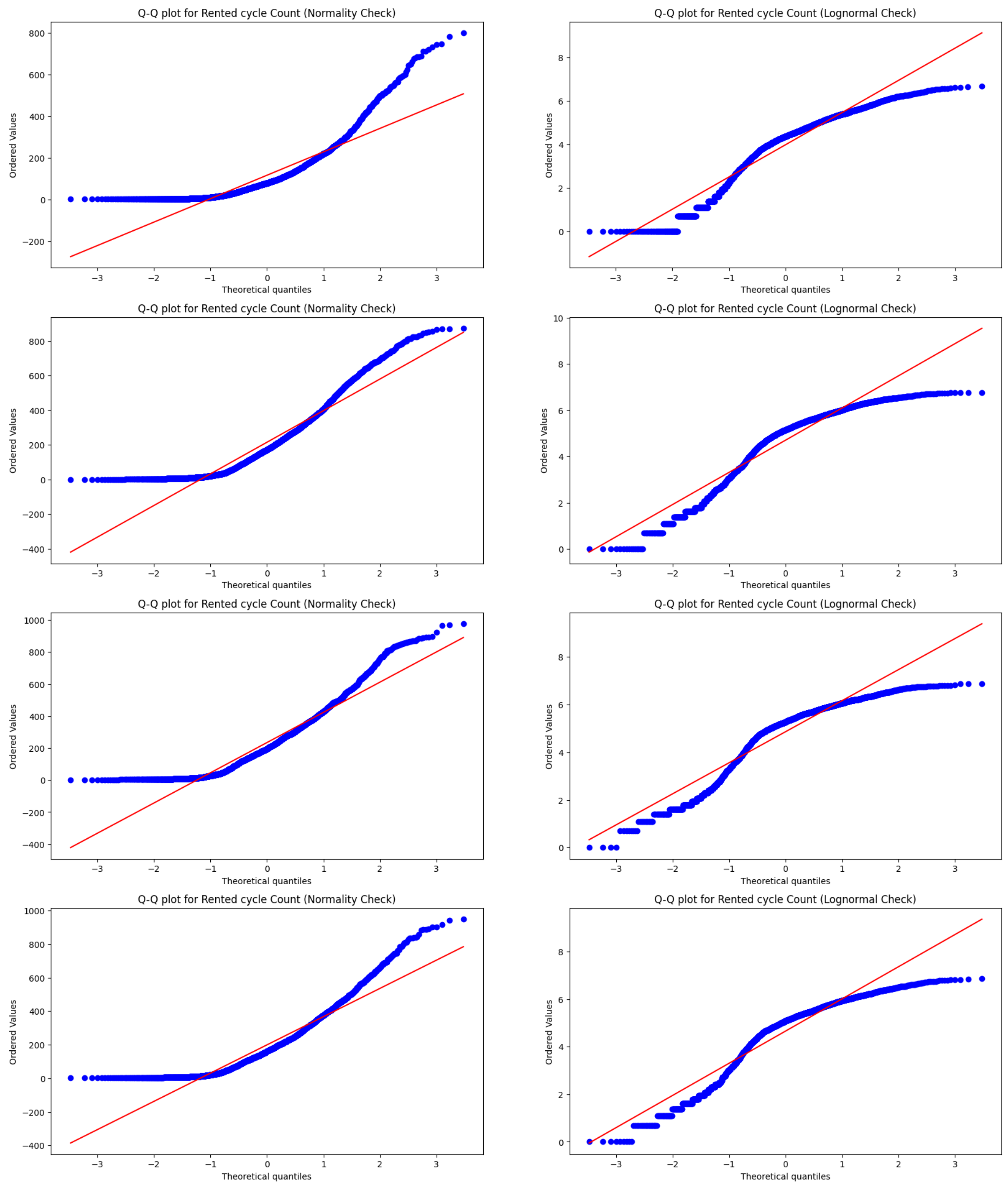
[21]:
```
season1 = yulu[yulu['season']==1]['count']
season2 = yulu[yulu['season']==2]['count']
season3 = yulu[yulu['season']==3]['count']
season4 = yulu[yulu['season']==4]['count']
```

[22]:
```
plt.figure(figsize=(20,12))
plt.subplot(2,2,1)
sns.histplot(data=season1, kde=True)
plt.subplot(2,2,2)
sns.histplot(data=season2, kde=True)
plt.subplot(2,2,3)
sns.histplot(data=season3, kde=True)
plt.subplot(2,2,4)
sns.histplot(data=season4, kde=True)
plt.show()
```



[23]:
```
import scipy.stats as stats
plt.figure(figsize=(20,24))
plt.subplot(4,2,1)
stats.probplot(season1, plot= plt, dist="norm")
plt.title('Q-Q plot for Rented cycle Count (Normality Check)')
plt.subplot(4,2,2)
stats.probplot(season1.apply(np.log), plot= plt, dist="norm")
plt.title('Q-Q plot for Rented cycle Count (Lognormal Check)')
plt.subplot(4,2,3)
stats.probplot(season2, plot= plt, dist="norm")
plt.title('Q-Q plot for Rented cycle Count (Normality Check)')
plt.subplot(4,2,4)
stats.probplot(season2.apply(np.log), plot= plt, dist="norm")
plt.title('Q-Q plot for Rented cycle Count (Lognormal Check)')
plt.subplot(4,2,5)
stats.probplot(season3, plot= plt, dist="norm")
plt.title('Q-Q plot for Rented cycle Count (Normality Check)')
plt.subplot(4,2,6)
stats.probplot(season3.apply(np.log), plot= plt, dist="norm")
plt.title('Q-Q plot for Rented cycle Count (Lognormal Check)')
plt.subplot(4,2,7)
stats.probplot(season4, plot= plt, dist="norm")
plt.title('Q-Q plot for Rented cycle Count (Normality Check)')
plt.subplot(4,2,8)
stats.probplot(season4.apply(np.log), plot= plt, dist="norm")
```

```python
plt.title('Q-Q plot for Rented cycle Count (Lognormal Check)')
plt.show()
```



The plots doesnt look like a normal gaussian distribution. it doesnt also satisfy the QQ - Plots.

We know, the **count** random variable, is actually count of the number of cycles rented in an hourly basis. We have seen pereviously that Count matches to **poisson distribution** to some extent.

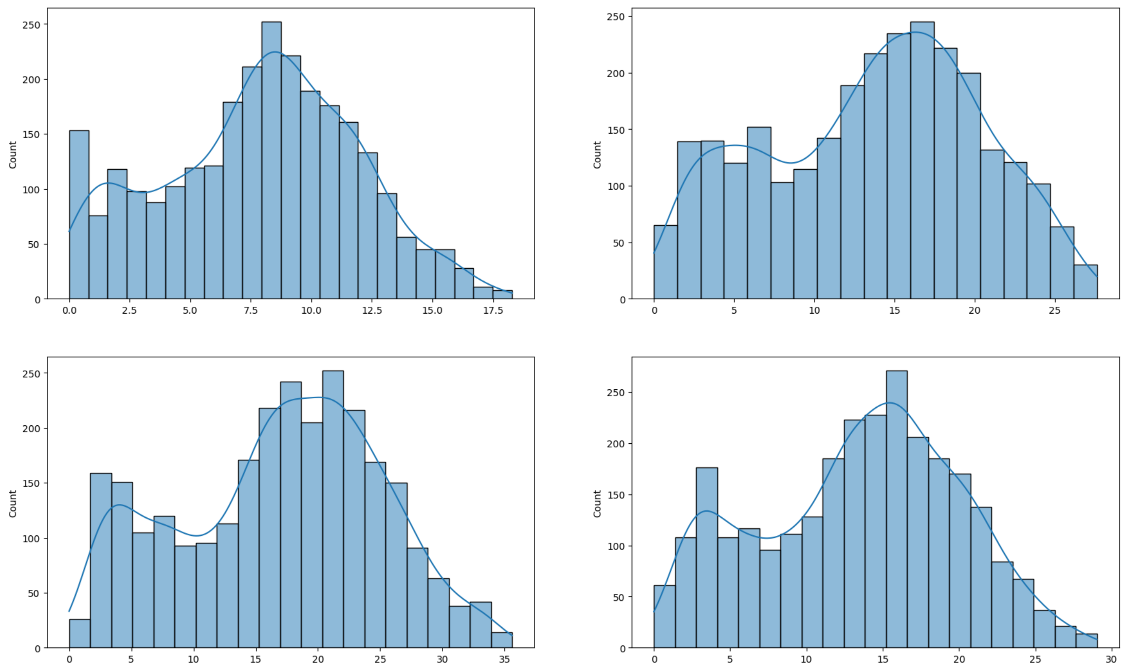
But to perform ANOVA, we need our groups to be gaussian distributed. So, we will perform

BOX-COX transform to change the distribution of these groups to normal.

B) BOX-COX Transform

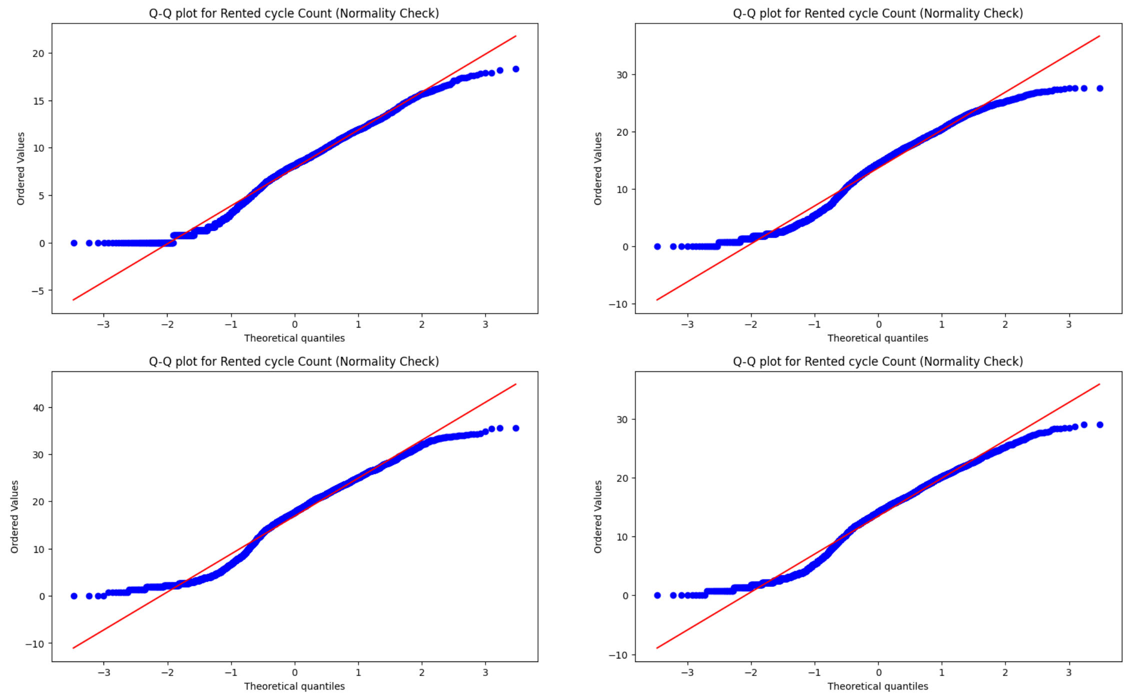
[24]:
```
season1= stats.boxcox(season1)[0]
season2= stats.boxcox(season2)[0]
season3= stats.boxcox(season3)[0]
season4= stats.boxcox(season4)[0]
```

[25]:
```
plt.figure(figsize=(20,12))
plt.subplot(2,2,1)
sns.histplot(data=season1, kde=True)
plt.subplot(2,2,2)
sns.histplot(data=season2, kde=True)
plt.subplot(2,2,3)
sns.histplot(data=season3, kde=True)
plt.subplot(2,2,4)
sns.histplot(data=season4, kde=True)
plt.show()
```



[26]:
```
import scipy.stats as stats
plt.figure(figsize=(20,12))
plt.subplot(2,2,1)
stats.probplot(season1, plot= plt, dist="norm")
plt.title('Q-Q plot for Rented cycle Count (Normality Check)')
plt.subplot(2,2,2)
```

```
stats.probplot(season2, plot= plt, dist="norm")
plt.title('Q-Q plot for Rented cycle Count (Normality Check)')
plt.subplot(2,2,3)
stats.probplot(season3, plot= plt, dist="norm")
plt.title('Q-Q plot for Rented cycle Count (Normality Check)')
plt.subplot(2,2,4)
stats.probplot(season4, plot= plt, dist="norm")
plt.title('Q-Q plot for Rented cycle Count (Normality Check)')
plt.show()
```



The plots tends to look like a normal gaussian distribution. it also paritally satisfies the QQ - Plots for normality check.

C) Homogeneity of Variances using Levene's test

1. **Null Hypothesis(H0)** - Homogenous Variance
2. **Alternate Hypothesis(HA)** - Non Homogenous variance

[27]:
```
stat,p = levene(season1,season2,season3,season4)
print('P-value :',p)
```

```
P-value : 2.132703669346175e-233
```

Reject NULL HYPOTHESIS

We reject null hypothesis, which means the variance is not similar across all the groups. But, we will still go ahead and perform our one-way ANOVA

#Performing One-Way Anova

[29]:
```
stat,p = stats.f_oneway(season1, season2, season3, season4)
print('P-value :',p)
```

```
P-value : 0.0
```

Reject NULL HYPOTHESIS

#Answer: We reject null hypothesis, which means the number of electric cycles rented on different seasons are different.

#Question-3: Number of cycles rented:- are the numbers similar or different for different weather? - ANOVA TEST

STEP-1 : Set up Null Hypothesis

- **Null Hypothesis (H0)** - Mean of cycle rented per hour is same for weather 1, 2 and 3. (* We wont be considering weather 4 as there in only 1 data point for weather 4 and we cannot perform a ANOVA test with a single data point for a group.)
- **Alternate Hypothesis (HA)** -Mean of cycle rented per hour is same for season 1,2,3 and 4 are different.

STEP-2 : Checking for basic assumpitons for the hypothesis

Normality check using **QQ Plot.** If the distribution is not normal, use **BOX-COX transform** to transform it to normal distribution.

Homogeneity of Variances using **Lavene's test**

Each observations are **independent.**

STEP-3: Define Test statistics; Distribution of T under H0.

The test statistic for a One-Way ANOVA is denoted as F. For an independent variable with k groups, the F statistic evaluates whether the group means are significantly different.

F=MSR/MSE

Under H0, the test statistic should follow **F-Distribution.**

STEP-4: Decide the kind of test.

We will be performing **right tailed t-test**

STEP-5: Compute the p-value and fix value of alpha.

we will be computing the anova-test p-value using the f_oneway function using scipy.stats. We set our **alpha to be 0.05**

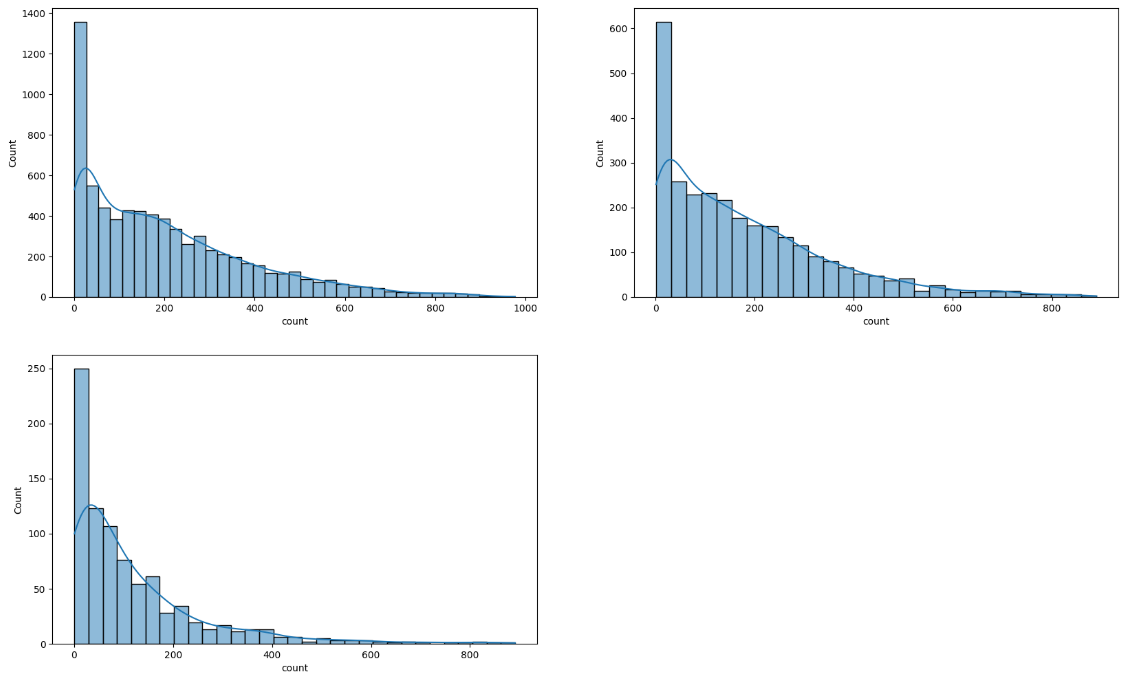
STEP-6: Compare p-value and alpha.

Based on p-value, we will accept or reject H0.

- **p-val > alpha** : Accept H0
- **p-val < alpha** : Reject H0

[30]:
```
weather1 = yulu[yulu['weather']==1]['count']
weather2 = yulu[yulu['weather']==2]['count']
weather3 = yulu[yulu['weather']==3]['count']
```

[31]:
```
plt.figure(figsize=(20,12))
plt.subplot(2,2,1)
sns.histplot(data=weather1, kde=True)
plt.subplot(2,2,2)
sns.histplot(data=weather2, kde=True)
plt.subplot(2,2,3)
sns.histplot(data=weather3, kde=True)
plt.show()
```



We already know Count is not a normal gaussian distribution. It **doesnt also satisfy the QQ - Plots.**

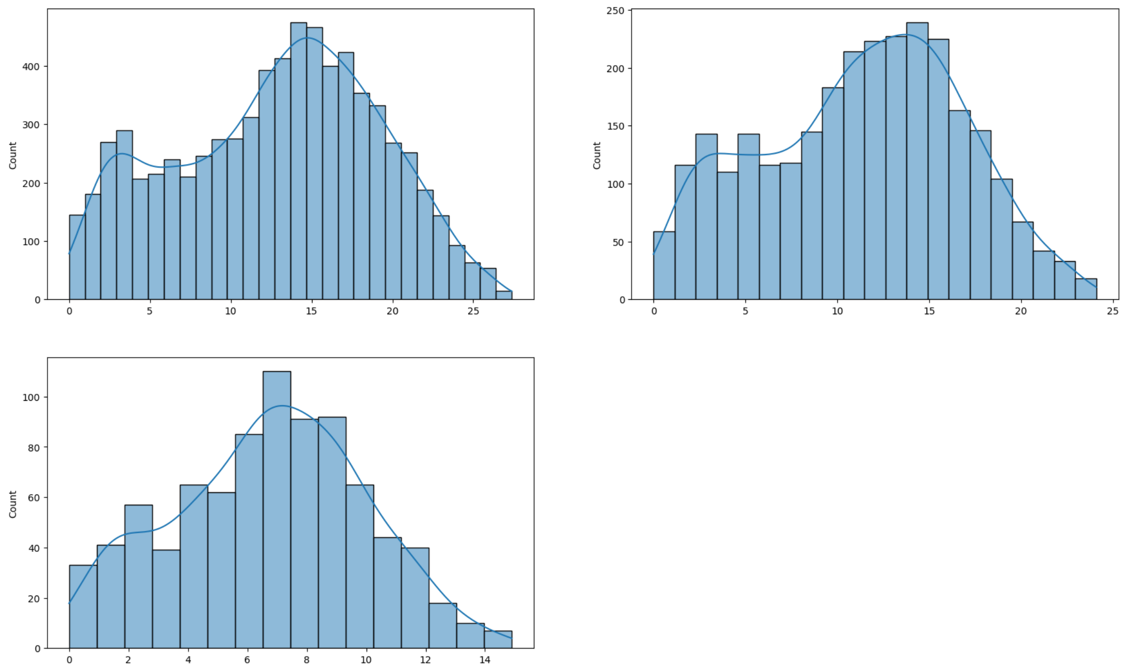
We know, the count random variable, is actually count of the number of cycles rented in an hourly basis. We have seen pereviously that Count matches to poisson distribution to some extent.

But to perform ANOVA, we need our groups to be gaussian distributed. So, we will perform **BOX-COX transform** to change the distribution of these groups to normal.

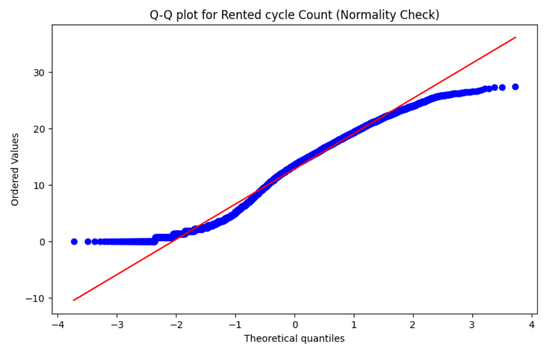
B) BOX-COX Transform

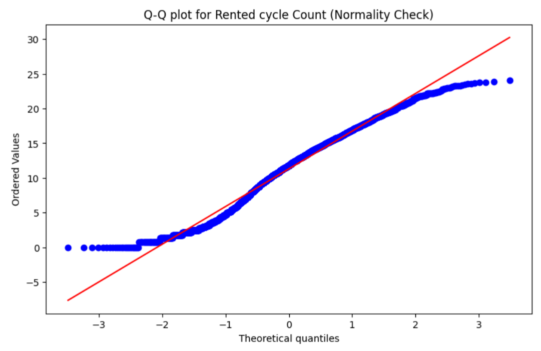
[32]:
```
weather1= stats.boxcox(weather1)[0]
weather2= stats.boxcox(weather2)[0]
weather3= stats.boxcox(weather3)[0]
```

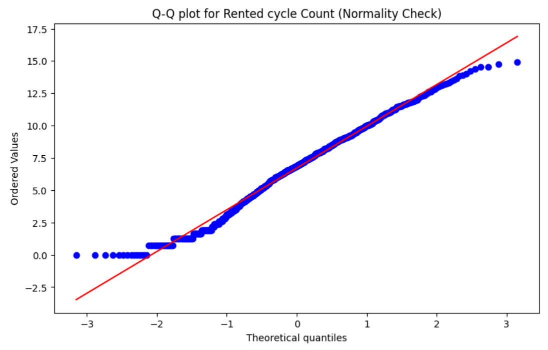
[33]:
```
plt.figure(figsize=(20,12))
plt.subplot(2,2,1)
sns.histplot(data=weather1, kde=True)
plt.subplot(2,2,2)
sns.histplot(data=weather2, kde=True)
plt.subplot(2,2,3)
sns.histplot(data=weather3, kde=True)
plt.show()
```



[34]:
```
import scipy.stats as stats
plt.figure(figsize=(20,12))
plt.subplot(2,2,1)
stats.probplot(weather1, plot= plt, dist="norm")
plt.title('Q-Q plot for Rented cycle Count (Normality Check)')
plt.subplot(2,2,2)
stats.probplot(weather2, plot= plt, dist="norm")
plt.title('Q-Q plot for Rented cycle Count (Normality Check)')
plt.subplot(2,2,3)
stats.probplot(weather3, plot= plt, dist="norm")
plt.title('Q-Q plot for Rented cycle Count (Normality Check)')
plt.show()
```







The plots tends to look like a normal **gaussian distribution**. it also satisfies the QQ - Plots for normality check to some extent, even though there is still some difference.

C) Homogeneity of Variances using Levene's test

Null Hypothesis(H0) - Homogenous Variance

Alternate Hypothesis(HA) - Non Homogenous variance

[35]:
```
stat,p = levene(weather1,weather2,weather3)
print('P-value :',p)
```

```
P-value : 1.0502360215620662e-101
```

Reject NULL HYPOTHESIS We **reject null hypothesis**, which means the **variance is not similar** across all the groups. But, we will still go ahead and perform our one-way ANOVA

D) Performing One-Way Anova

[36]:
```
stat,p = stats.f_oneway(weather1,weather2,weather3)
print('P-value :',p)
```

```
P-value : 3.4867243611236345e-181
```

Reject NULL HYPOTHESIS

#Answer: We reject null hypothesis, which means the number of electric cycles rented on different Weathers are different.

#Question-4:- Weather is dependent on season or not? (check between 2 predictor variable) ~ Chi-square test

The Chi-square statistic is a non-parametric (distribution free) tool designed to analyze group differences when the dependent variable is measured at a nominal level. Like all non-parametric statistics, the Chi-square is robust with respect to the distribution of the data. Specifically, it does not require equality of variances among the study groups or homoscedasticity in the data.

STEP-1 : Set up Null Hypothesis

Null Hypothesis (H0) - weather is independent of season

Alternate Hypothesis (HA) -weather is dependent of seasons.

STEP-2: Checking for basic assumpitons for the hypothesis (Non-Parametric Test) 1. The data in the cells should be **frequencies**, or **counts** of cases. 2. The levels (or categories) of the variables are **mutually exclusive. That is, a particular subject fits into one and only one level of each of the variables. 3. There are 2 variables, and both are measured as** categories. **4. The value of the cell expecteds should be 5 or morein at least 80% of the cells, and no cell should have an expected of less than one (3).** STEP-3: Define Test statistics; Distribution of T under H0.**

The test statistic for a Chi- square test is denoted as T. Under H0, the test statistic should follow Chi-Square Distribution.

STEP-4: Decide the kind of test.

We will be performing **right tailed t-test.**

STEP-5: Compute the p-value and fix value of alpha.

we will be computing the chi square-test p-value using the chi 2 function using scipy.stats. We set our **alpha to be 0.05**

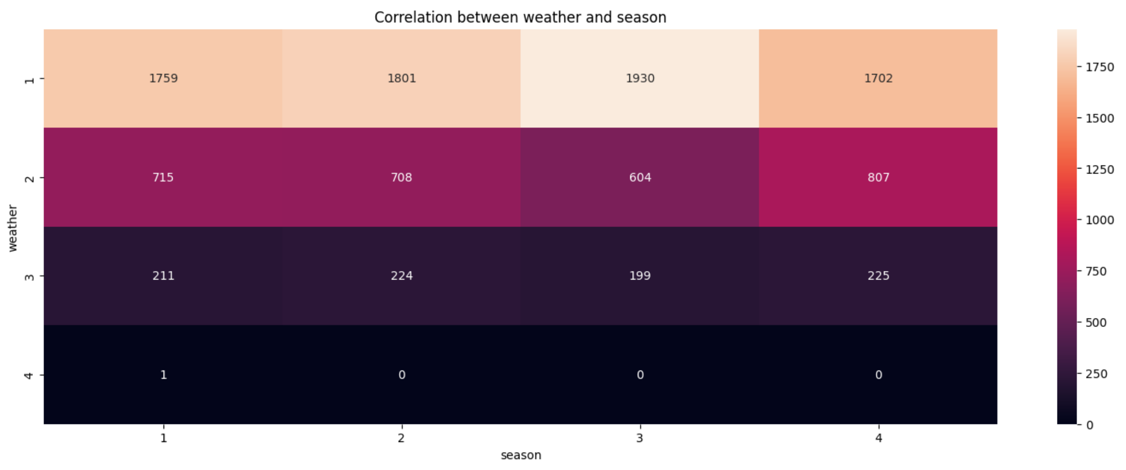
STEP-6: Compare p-value and alpha.

Based on p-value, we will accept or reject H0.

1. **p-val > alpha** : Accept H0
2. **p-val < alpha** : Reject H0

[37]:
```
cont = pd.crosstab(yulu['weather'],yulu['season'])
value = np.array([cont.iloc[0][0:4].values ,cont.iloc[1][0:4].values ,cont.
 ↪iloc[2][0:4].values])
```

[38]:
```
plt.figure(figsize=(18,6))
sns.heatmap(cont, annot=True, fmt='.6g')
plt.title('Correlation between weather and season')
plt.show()
```



We will be dropping the last row as frequency/count is less than 3.

Performing chi-square test

[39]:
```python
c, p, dof, expected = stats.chi2_contingency(value)
print('P-value :',p)
```

```
P-value : 2.8260014509929403e-08
```

Reject NULL HYPOTHESIS #**Answer: We reject null hypothesis, which means that the season has impact on the weather.**

#**Observations and Conclusions:**

1. **Season** - Count of rented bikes are maximum in season 3 and minimum in season 1
2. **Weather** - Count of rented bikes are maximum in weather 1 and falls sharply in weather 3 and 4.
3. **Temperature** - Count of rented bikes are very low at lower temperatures and rises with increase in temperature.
4. **Time of day** - Count of cycle rent falls steeply through the night and attains peak during the day.
5. Humidity, windspeed, temperature and weather are correlated with season and impacts the count of cycles rented.
6. Cycles rented on working and non-working days are similar in number.
7. Casual Cycles rented on Weekends are higher than Registered cycles rented on weekends.
8. Registered cycles rented on working days are higher than casual cycles rented on working days.

#**Recommendations:**

1. The EDA suggests that count of cycle rented is very low during season1. **Yulu can rollout some challenges during this season, like New year goals on fitness and roll out exciting prizes for people participating in this season.**

2. The count of cycle is very low during the night hours. **Yulu can introduce some safety**

features in its app and also some promotional discounts for people renting yulu bikes from 12am - 6am, which can motivate people to ride Yulu bikes during this time frame.

3. The count of bikes rented on holidays are significantly lower than on non-holidays. **Yulu can set up bike centers near entertainment hotspots, like malls and amusemennt parks to hike up its rental count on weekends and holidays.**

4. The EDA proves that Temperature, humidity, windspeed and weather - all these environmental factors affect the count of bikes rented on Yulu. Even though Yulu can not change the weather conditions, **Yulu can offer reduced pricing for people renting in these weather conditions (weather 2 & 3).**

5. Yulu can introduce exclusive benifits for registered users to get more users to register, which can pull up the amount of cycles rented per hour.

6. Yulu can introduce some early biker scheme (4am -7am), **that would target the people who are motivated in physical fitness and also roll out some exciting prizes for people covering the longest distance rides.**