Yulu - Hypothesis Testing

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.

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

Loading Dataset

In [1]:
```
!gdown 11LTqn16OPombCP4CvcGYgojkxgohuwXL
```

```
Downloading...
From: https://drive.google.com/uc?id=11LTqn16OPombCP4CvcGYgojkxgohuwXL
To: /content/bike_sharing.txt

  0% 0.00/648k [00:00<?, ?B/s]
100% 648k/648k [00:00<00:00, 9.16MB/s]
```

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

data = pd.read_csv('bike_sharing.txt')
data
```

Out[2]:

	datetime	season	holiday	workingday	weather	temp	atemp	humidi
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	
...	...	...	...	...	...	...	...	
10881	2012-12-19 19:00:00	4	0	1	1	15.58	19.695	
10882	2012-12-19 20:00:00	4	0	1	1	14.76	17.425	
10883	2012-12-19 21:00:00	4	0	1	1	13.94	15.910	
10884	2012-12-19 22:00:00	4	0	1	1	13.94	17.425	
10885	2012-12-19 23:00:00	4	0	1	1	13.12	16.665	

10886 rows × 12 columns

Data Dictionary :

- datetime: datetime
- season: season (1: spring, 2: summer, 3: fall, 4: winter)
- holiday: whether day is a holiday or not.
- workingday: if day is neither weekend nor holiday is 1, otherwise is 0.
- Weather:

  1: Clear, Few clouds, partly cloudy, partly cloudy

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

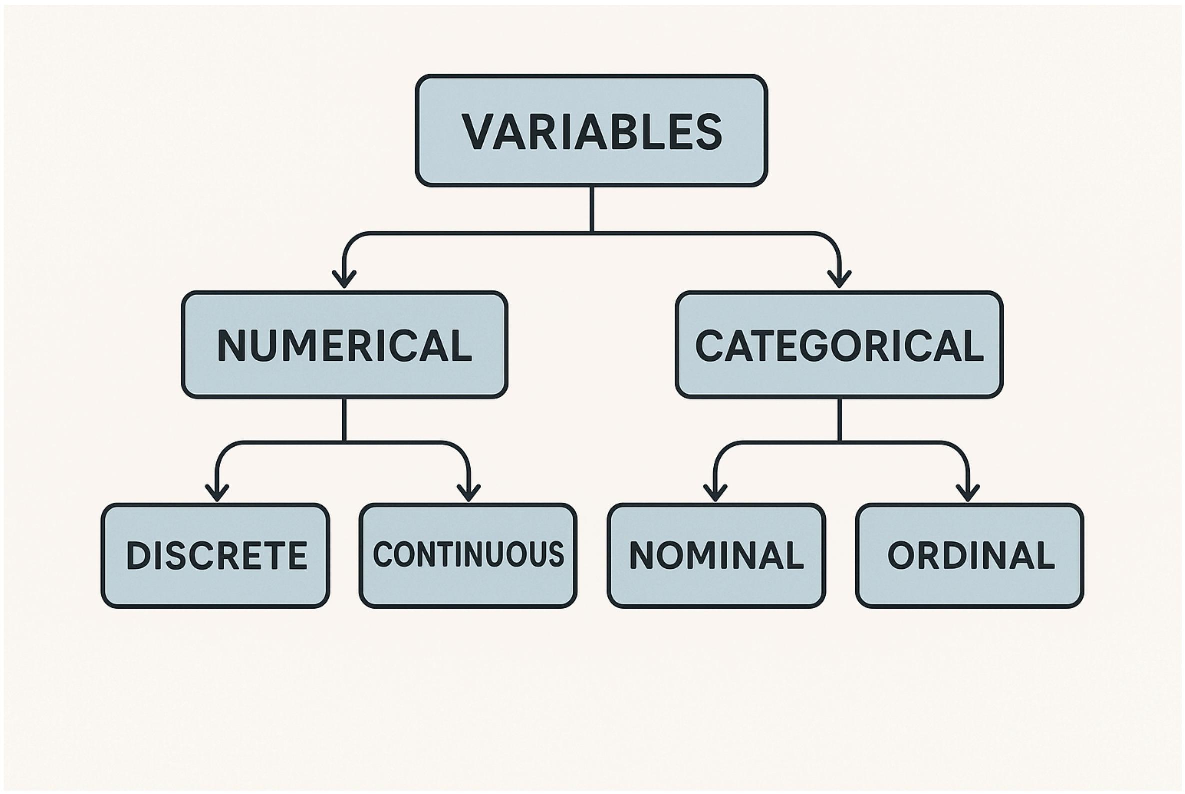
  3: Light Snow, Light Rain + Thunderstorm + Scattered clouds, Light Rain + Scattered clouds

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

- temp: temperature in Celsius
- atemp: feeling temperature in Celsius
- humidity: humidity
- windspeed: wind speed
- casual: count of casual users
- registered: count of registered users
- count: count of total rental bikes including both casual and registered

Observation on variables & datatypes

Flow chart of variables



Columns: temp , atemp , windspeed

Variable Type: Numerical → Continuous

Description:

temp - temperature in Celsius.

atemp - feeling temperature in Celsius.

windspeed - wind speed

Columns: humidity , casual , registered , count

Variable Type: Numerical → Discrete

Description:

humidity - humidity

casual - count of casual users

registered - count of registered users

count - count of total rental bikes including both casual and registered

---

Columns: season , Weather

Variable Type: Categorical → Ordinal

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

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

---

Columns: workingday , holiday

Variable Type: Categorical → Nominal

holiday - whether day is a holiday or not.

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

Column : datetime variable - DateTime

In [3]:
```
# Checking the datatypes of the columns
data.dtypes
```

Out[3]:

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

dtype: object

In [4]:
```
# Clearcut information about the datasetis given below:
data.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
```

In [5]:
```
print('Number of rows:',data.shape[0])
print('Number of columns:',data.shape[1])
```

```
Number of rows: 10886
Number of columns: 12
```

1 . Analysing the basic metrics and observation of Yulu - bike sharing data

In [6]:
```
# including all columns in a Dataframe description
data.describe(include='all')
```

Out[6]:

	datetime	season	holiday	workingday	weather	
count	10886	10886.000000	10886.000000	10886.000000	10886.000000	108
unique	10886	NaN	NaN	NaN	NaN	
top	2012-12-19 23:00:00	NaN	NaN	NaN	NaN	
freq	1	NaN	NaN	NaN	NaN	
mean	NaN	2.506614	0.028569	0.680875	1.418427	
std	NaN	1.116174	0.166599	0.466159	0.633839	
min	NaN	1.000000	0.000000	0.000000	1.000000	
25%	NaN	2.000000	0.000000	0.000000	1.000000	
50%	NaN	3.000000	0.000000	1.000000	1.000000	
75%	NaN	4.000000	0.000000	1.000000	2.000000	
max	NaN	4.000000	1.000000	1.000000	4.000000	

In [7]:
```
# including categorical columns in a Dataframe description
data.describe(include='object')
```

Out[7]:

	datetime
count	10886
unique	10886
top	2012-12-19 23:00:00
freq	1

In [8]:
```
# including numerical columns in a Dataframe description
data.describe(include=[np.number])
```

Out[8]:

	season	holiday	workingday	weather	temp	
count	10886.000000	10886.000000	10886.000000	10886.000000	10886.00000	108
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	

In [9]:
```
# Analysing the nan values for each column
data.isna().sum()/len(data)*100
```

Out[9]:

	0
datetime	0.0
season	0.0
holiday	0.0
workingday	0.0
weather	0.0
temp	0.0
atemp	0.0
humidity	0.0
windspeed	0.0
casual	0.0
registered	0.0
count	0.0

dtype: float64

- No null values are present in the Dataset

Conversion of object attributes to 'DateTime'

In [10]:
```
data['datetime'] = pd.to_datetime(data['datetime'])
data["datetime"].head()
```

Out[10]:

	datetime
0	2011-01-01 00:00:00
1	2011-01-01 01:00:00
2	2011-01-01 02:00:00
3	2011-01-01 03:00:00
4	2011-01-01 04:00:00

dtype: datetime64[ns]

2.Non-Graphical Analysis

In [11]:
```
# Number of unique values in every column of dataset

for col in data.columns:
  print(col,':',data[col].nunique())
```

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

In [12]:
```
# Unique values in every column of dataset

for col in data.columns:
  print(f"unique values of {col}")
  print("-------------------------")
  print(col,':',data[col].unique(),"\n")
```

```
unique values of datetime
-------------------------
datetime : <DatetimeArray>
['2011-01-01 00:00:00', '2011-01-01 01:00:00', '2011-01-01 02:00:00',
 '2011-01-01 03:00:00', '2011-01-01 04:00:00', '2011-01-01 05:00:00',
 '2011-01-01 06:00:00', '2011-01-01 07:00:00', '2011-01-01 08:00:00',
 '2011-01-01 09:00:00',
 ...
 '2012-12-19 14:00:00', '2012-12-19 15:00:00', '2012-12-19 16:00:00',
 '2012-12-19 17:00:00', '2012-12-19 18:00:00', '2012-12-19 19:00:00',
 '2012-12-19 20:00:00', '2012-12-19 21:00:00', '2012-12-19 22:00:00',
 '2012-12-19 23:00:00']
Length: 10886, dtype: datetime64[ns]

unique values of season
-------------------------
season : [1 2 3 4]

unique values of holiday
-------------------------
holiday : [0 1]

unique values of workingday
-------------------------
workingday : [0 1]

unique values of weather
-------------------------
weather : [1 2 3 4]

unique values of temp
-------------------------
temp : [ 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.  ]

unique values of atemp
-------------------------
atemp : [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]

unique values of humidity
-------------------------
humidity : [ 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]

unique values of windspeed
-------------------------
windspeed : [ 0.      6.0032 16.9979 19.0012 19.9995 12.998  15.0013  8.9981 1
1.0014
 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]

unique values of casual
-------------------------
casual : [  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]

unique values of registered
-------------------------
registered : [ 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]

unique values of count
-------------------------
count : [ 16  40  32  13   1   2   3   8  14  36  56  84  94 106 110  93  67  3
5
  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]
```

In [13]:
```
# Observation of value counts in each column (first four columns)

for col in data.columns:
  print(f"Value counts of {col}")
  print("------------------------")
  print(data[col].value_counts(),'\n')
```

```
Value counts of datetime
-------------------------
datetime
2012-12-19 23:00:00    1
2011-01-01 00:00:00    1
2011-01-01 01:00:00    1
2011-01-01 02:00:00    1
2011-01-01 03:00:00    1
                      ..
2011-01-01 10:00:00    1
2011-01-01 09:00:00    1
2011-01-01 08:00:00    1
2011-01-01 07:00:00    1
2011-01-01 06:00:00    1
Name: count, Length: 10886, dtype: int64

Value counts of season
-------------------------
season
4    2734
2    2733
3    2733
1    2686
Name: count, dtype: int64

Value counts of holiday
-------------------------
holiday
0    10575
1      311
Name: count, dtype: int64

Value counts of workingday
-------------------------
workingday
1    7412
0    3474
Name: count, dtype: int64

Value counts of weather
-------------------------
weather
1    7192
2    2834
3     859
4       1
Name: count, dtype: int64

Value counts of temp
-------------------------
temp
14.76    467
26.24    453
28.70    427
```

```
13.94    413
18.86    406
22.14    403
25.42    403
16.40    400
22.96    395
27.06    394
24.60    390
12.30    385
21.32    362
17.22    356
13.12    356
29.52    353
10.66    332
18.04    328
20.50    327
30.34    299
9.84     294
15.58    255
9.02     248
31.16    242
8.20     229
27.88    224
23.78    203
32.80    202
11.48    181
19.68    170
6.56     146
33.62    130
5.74     107
7.38     106
31.98     98
34.44     80
35.26     76
4.92      60
36.90     46
4.10      44
37.72     34
36.08     23
3.28      11
0.82       7
38.54      7
39.36      6
2.46       5
1.64       2
41.00      1
Name: count, dtype: int64

Value counts of atemp
-------------------------
atemp
31.060    671
25.760    423
22.725    406
```

20.455	400
26.515	395
16.665	381
25.000	365
33.335	364
21.210	356
30.305	350
15.150	338
21.970	328
24.240	327
17.425	314
31.820	299
34.850	283
27.275	282
32.575	272
11.365	271
14.395	269
29.545	257
19.695	255
15.910	254
12.880	247
13.635	237
34.090	224
12.120	195
28.790	175
23.485	170
10.605	166
35.605	159
9.850	127
36.365	123
18.180	123
37.120	118
9.090	107
37.880	97
28.030	80
7.575	75
38.635	74
6.060	73
39.395	67
6.820	63
8.335	63
40.150	45
18.940	45
40.910	39
5.305	25
42.425	24
41.665	23
3.790	16
4.545	11
3.030	7
43.940	7
43.180	7
2.275	7
44.695	3

```
0.760       2
1.515       1
45.455      1
Name: count, dtype: int64

Value counts of humidity
-------------------------
humidity
88    368
94    324
83    316
87    289
70    259
     ...
8       1
13      1
97      1
96      1
91      1
Name: count, Length: 89, dtype: int64

Value counts of windspeed
-------------------------
windspeed
0.0000     1313
8.9981     1120
11.0014    1057
12.9980    1042
7.0015     1034
15.0013     961
6.0032      872
16.9979     824
19.0012     676
19.9995     492
22.0028     372
23.9994     274
26.0027     235
27.9993     187
30.0026     111
31.0009      89
32.9975      80
35.0008      58
39.0007      27
36.9974      22
43.0006      12
40.9973      11
43.9989       8
46.0022       3
47.9988       2
56.9969       2
51.9987       1
50.0021       1
Name: count, dtype: int64
```

```
Value counts of casual
------------------------
casual
0      986
1      667
2      487
3      438
4      354
      ...
294      1
280      1
216      1
292      1
304      1
Name: count, Length: 309, dtype: int64

Value counts of registered
------------------------
registered
3      195
4      190
5      177
6      155
2      150
      ...
709      1
699      1
720      1
788      1
803      1
Name: count, Length: 731, dtype: int64

Value counts of count
------------------------
count
5      169
4      149
3      144
6      135
2      132
      ...
842      1
683      1
863      1
725      1
637      1
Name: count, Length: 822, dtype: int64
```

Observations

Here are a few **observations** for each column based on the value counts you provided:

`datetime`

- All 10,886 entries are **unique per hour** → likely hourly data over a few years.
- Values span from **2011-01-01 00:00:00** to **2012-12-19 23:00:00**.
- No duplicate timestamps → **time series is consistent** and clean.

`season`

- Each season appears roughly equally, showing **balanced seasonal distribution**:
    - Season 4 (Winter): 2734
    - Season 2 (Summer): 2733
    - Season 3 (Fall): 2733
    - Season 1 (Spring): 2686
- Slightly fewer records for Spring.

`holiday`

- Only **311 entries are holidays** (2.9%) → very **few holidays** in the dataset.
- Indicates most entries (~97%) are regular days.

`workingday`

- 7412 entries (68%) are **working days**.
- 3474 entries (32%) are **non-working days** (weekend or holiday).
- This shows that **weekday usage dominates**.

`weather`

- Mostly clear or partly cloudy days:
    - Category 1 (Clear etc.): 7192 entries (~66%)
- Very few severe weather:

        - Category 4 (Severe): only **1 entry** → could be an outlier or rare event.

◈ `temp`

- Temp values are continuous but repeated → rounded to 2 decimal places.
- Most common temperature: **14.76°C**, followed by 26.24°C and 28.70°C.
- Ranges from ~0.82°C to ~41.00°C → covers all seasons well.

◈ `atemp` (feels like temperature)

- Most common atemp: **31.06**, followed by 25.76, 22.72, and so on.
- Close correlation with `temp` likely.
- Also rounded to 2 decimal places → discrete-like behavior.

◈ `humidity`

- 89 unique values, suggesting it's **integer-based**.
- Most frequent humidity: **88%**, followed by 94% and 83%.
- Humidity ranges from **8% to 97%** → full range captured.

◈ `windspeed`

- 0.0000 appears **1313 times**, may indicate **missing or calm wind**.
- Rest show a continuous distribution with common values like 8.99, 11.00, 12.99.
- Few high values (up to ~57) → rare strong wind events.

◈ `casual` (non-registered users)

- Many zeros (986) → **many time periods with no casual users**.
- Highest frequency is at lower values (1–5).
- Long tail up to 304 → possible **occasional high traffic** days.

◈ `registered` (registered users)

- Peaks at small values like 3, 4, 5.
- Total 731 unique values → more variation than casual.
- Long tail up to 803 → **registered users dominate** usage.

◈ `count` (total users = casual + registered)

- Peaks at low values (3–6), consistent with casual + registered.
- Max count observed: 863
- 822 unique values → large variation in hourly total usage.

In [14]:
```
#Examine of duplicated values in dataset
data.duplicated().sum()
```

Out[14]:
```
np.int64(0)
```

- Dataset contains with zero duplicates

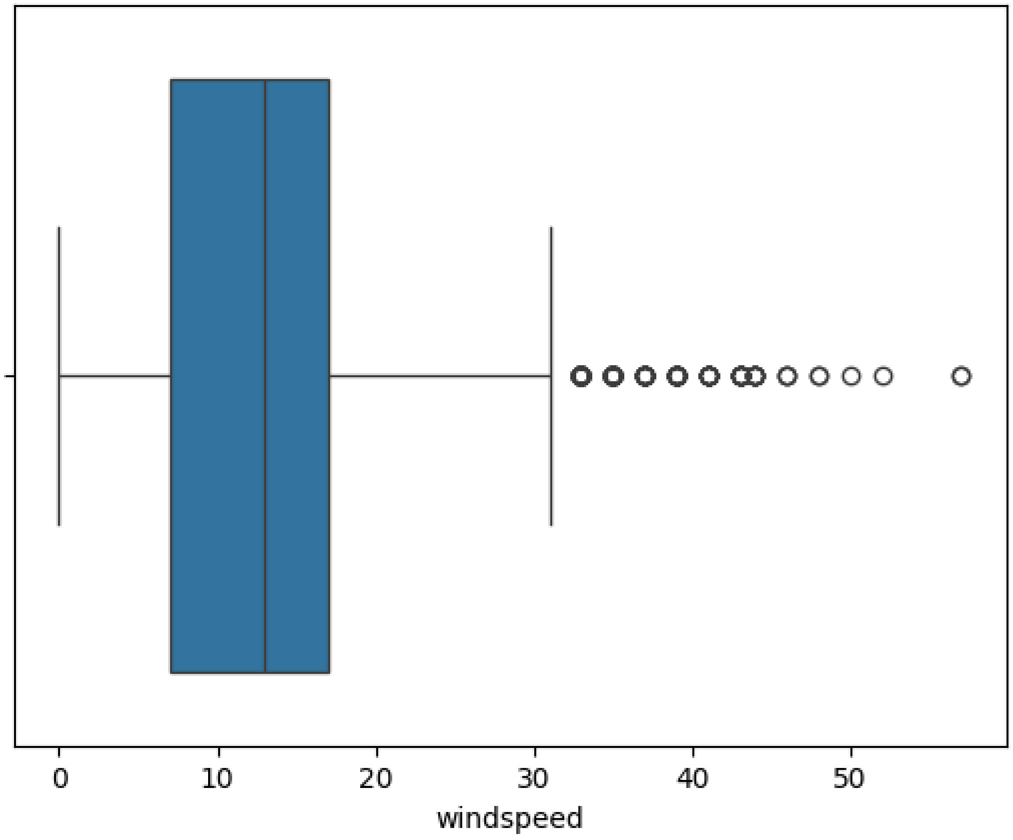
Detecting Outliers

- We detect outliers for continous variables - Purchase column

- Let's take a deepcopy of the dataset

In [15]:
```
data_windspeed = data.copy()
```

In [16]:
```
sns.boxplot(data=data_windspeed,x='windspeed')
plt.show()

Q1 = np.percentile(data_windspeed ['windspeed'],25)
Q2 = np.percentile(data_windspeed ['windspeed'],50)
Q3 = np.percentile(data_windspeed ['windspeed'],75)
IQR = Q3-Q1
print (f"The first quartile (25th percentile) for windspeed is {Q1}")
print (f"The second quartile (50th percentile) for windspeed is {Q2}")
print (f"The third quartile (75th percentile) for windspeed is {Q3}")
print(f"The Interquartile range for windspeed is {IQR}")

higher_bound = Q3+1.5*IQR
lower_bound = Q1-1.5*IQR
print(f"The higher bound for windspeed is {higher_bound}")
print(f"The lower bound for windspeed is {lower_bound}")
```



```
The first quartile (25th percentile) for windspeed is 7.0015
The second quartile (50th percentile) for windspeed is 12.998
The third quartile (75th percentile) for windspeed is 16.9979
The Interquartile range for windspeed is 9.996400000000001
The higher bound for windspeed is 31.992500000000003
The lower bound for windspeed is -7.993100000000002
```

In [17]:
```
outliers_windspeed = data_windspeed[(data_windspeed['windspeed']>higher_bound)
print(f"The Outliers present in the windspeed column are {outliers_windspeed}"
print(f"The length of Outliers in the windspeed column are {len(outliers_winds
The Outliers present in the windspeed column are [32.9975, 36.9974, 35.0008, 3
5.0008, 39.0007, 35.0008, 35.0008, 36.9974, 32.9975, 36.9974, 35.0008, 32.9975,
32.9975, 36.9974, 32.9975, 35.0008, 39.0007, 35.0008, 32.9975, 43.9989, 40.997
3, 40.9973, 32.9975, 35.0008, 43.9989, 32.9975, 51.9987, 46.0022, 35.0008, 39.0
007, 32.9975, 32.9975, 35.0008, 39.0007, 43.9989, 39.0007, 39.0007, 32.9975, 4
0.9973, 43.9989, 50.0021, 43.0006, 40.9973, 35.0008, 36.9974, 32.9975, 32.9975,
35.0008, 36.9974, 35.0008, 32.9975, 32.9975, 40.9973, 39.0007, 35.0008, 39.000
7, 32.9975, 32.9975, 32.9975, 32.9975, 32.9975, 36.9974, 35.0008, 40.9973, 32.9
975, 36.9974, 32.9975, 39.0007, 32.9975, 32.9975, 32.9975, 35.0008, 32.9975, 3
5.0008, 35.0008, 35.0008, 35.0008, 32.9975, 35.0008, 32.9975, 39.0007, 35.0008,
39.0007, 39.0007, 39.0007, 35.0008, 32.9975, 40.9973, 35.0008, 35.0008, 39.000
7, 32.9975, 32.9975, 32.9975, 35.0008, 32.9975, 56.9969, 56.9969, 32.9975, 32.9
975, 43.0006, 32.9975, 32.9975, 35.0008, 35.0008, 39.0007, 40.9973, 36.9974, 3
2.9975, 32.9975, 32.9975, 32.9975, 32.9975, 36.9974, 32.9975, 36.9974, 32.9975,
43.0006, 43.0006, 35.0008, 32.9975, 32.9975, 35.0008, 35.0008, 43.9989, 35.000
8, 39.0007, 35.0008, 32.9975, 36.9974, 40.9973, 32.9975, 35.0008, 36.9974, 39.0
007, 32.9975, 32.9975, 36.9974, 32.9975, 32.9975, 35.0008, 35.0008, 35.0008, 3
2.9975, 35.0008, 35.0008, 32.9975, 35.0008, 35.0008, 43.9989, 36.9974, 36.9974,
43.0006, 32.9975, 39.0007, 46.0022, 47.9988, 43.0006, 35.0008, 36.9974, 35.000
8, 32.9975, 36.9974, 40.9973, 32.9975, 43.9989, 39.0007, 32.9975, 43.0006, 43.0
006, 46.0022, 43.0006, 43.9989, 36.9974, 32.9975, 35.0008, 35.0008, 35.0008, 3
2.9975, 35.0008, 40.9973, 32.9975, 32.9975, 32.9975, 35.0008, 32.9975, 47.9988,
39.0007, 36.9974, 36.9974, 32.9975, 35.0008, 32.9975, 32.9975, 39.0007, 35.000
8, 35.0008, 32.9975, 32.9975, 35.0008, 39.0007, 32.9975, 32.9975, 32.9975, 32.9
975, 32.9975, 39.0007, 39.0007, 35.0008, 35.0008, 32.9975, 35.0008, 35.0008, 3
6.9974, 39.0007, 43.0006, 43.0006, 39.0007, 35.0008, 35.0008, 39.0007, 32.9975,
32.9975, 32.9975, 43.0006, 32.9975, 32.9975]
The length of Outliers in the windspeed column are 227
```

In [18]:

```
data_cleaned_windspeed = data_windspeed[~data_windspeed['windspeed'].isin(outl
data_cleaned_windspeed
```

Out[18]:

	datetime	season	holiday	workingday	weather	temp	atemp	humidi
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	
...	...	...	...	...	...	...	...	
10881	2012-12-19 19:00:00	4	0	1	1	15.58	19.695	
10882	2012-12-19 20:00:00	4	0	1	1	14.76	17.425	
10883	2012-12-19 21:00:00	4	0	1	1	13.94	15.910	
10884	2012-12-19 22:00:00	4	0	1	1	13.94	17.425	
10885	2012-12-19 23:00:00	4	0	1	1	13.12	16.665	

10659 rows × 12 columns

- Clipping data

In [19]:
```
Q5 = np.percentile(data['windspeed'],5)
Q95 = np.percentile(data['windspeed'],95)
print(f"The 5th percentile for windspeed is {Q5}")
print(f"The 95th percentile for windspeed is {Q95}")
```

```
The 5th percentile for windspeed is 0.0
The 95th percentile for windspeed is 27.9993
```

In [20]:
```
data['windspeed'] = np.clip(data['windspeed'],Q5,Q95)
data['windspeed']
```

Out[20]:

	windspeed
0	0.0000
1	0.0000
2	0.0000
3	0.0000
4	0.0000
...	...
10881	26.0027
10882	15.0013
10883	15.0013
10884	6.0032
10885	8.9981

10886 rows × 1 columns

dtype: float64

In [21]:
```
print(f"The total number of Outliers clipped {data.shape[0] - data_cleaned_wir
```

```
The total number of Outliers clipped 227
```

Intuitive Ways to Handle Outliers:

- Clipping Operation Performed We initially performed clipping using statistical thresholds (e.g., IQR or z-score) to limit the influence of extreme values. This helped reduce distortion in distributions without deleting data.

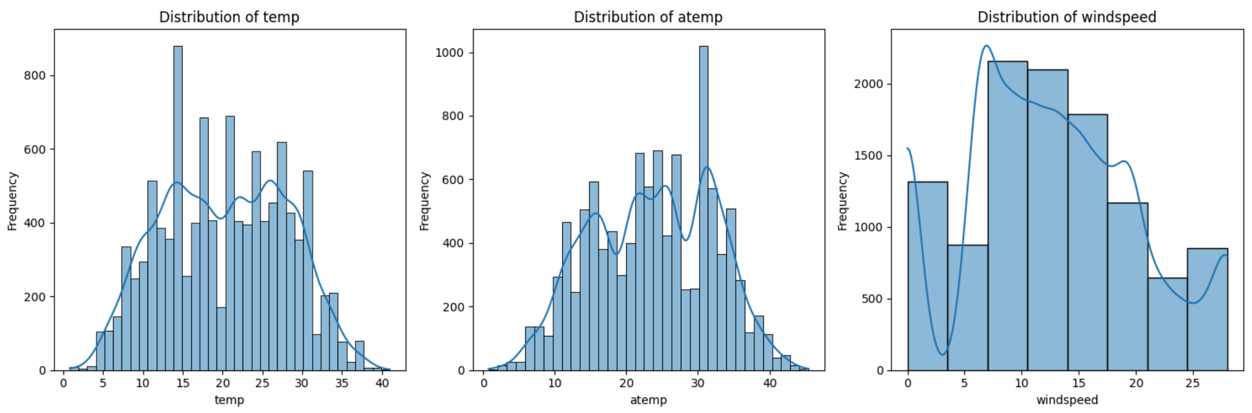
- Domain Understanding: High Windspeed Is Not an Outlier — It's Critical Upon reviewing the context, high windspeed values were identified during weather events (storms, heavy rain, etc.). These are not invalid outliers, but important signals — especially if the goal is to understand how weather impacts user behavior.

- Outlier Count Is Low (277 Entries), and They Are Not Noisy or Scrap Values With only 277 entries marked as outliers (~2.5% of the data), we concluded that these do not reflect data entry errors or noise. Removing them would lead to loss of meaningful variance, especially for edge-case scenarios that the model may need to learn.

Distribution of Numerical & Categorical variables

- **UNI-VARIATE ANALYSIS**

In [22]:
```
fig,axes = plt.subplots(1,3,figsize=(15,5))
sns.histplot(data=data,x='temp',ax=axes[0],kde=True)
axes[0].set_title('Distribution of temp')
axes[0].set_xlabel('temp')
axes[0].set_ylabel('Frequency')

sns.histplot(data=data,x='atemp',ax=axes[1],kde=True)
axes[1].set_title('Distribution of atemp')
axes[1].set_xlabel('atemp')
axes[1].set_ylabel('Frequency')

sns.histplot(data=data,x='windspeed',bins=8,ax=axes[2],kde=True)
axes[2].set_title('Distribution of windspeed')
axes[2].set_xlabel('windspeed')
axes[2].set_ylabel('Frequency')

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

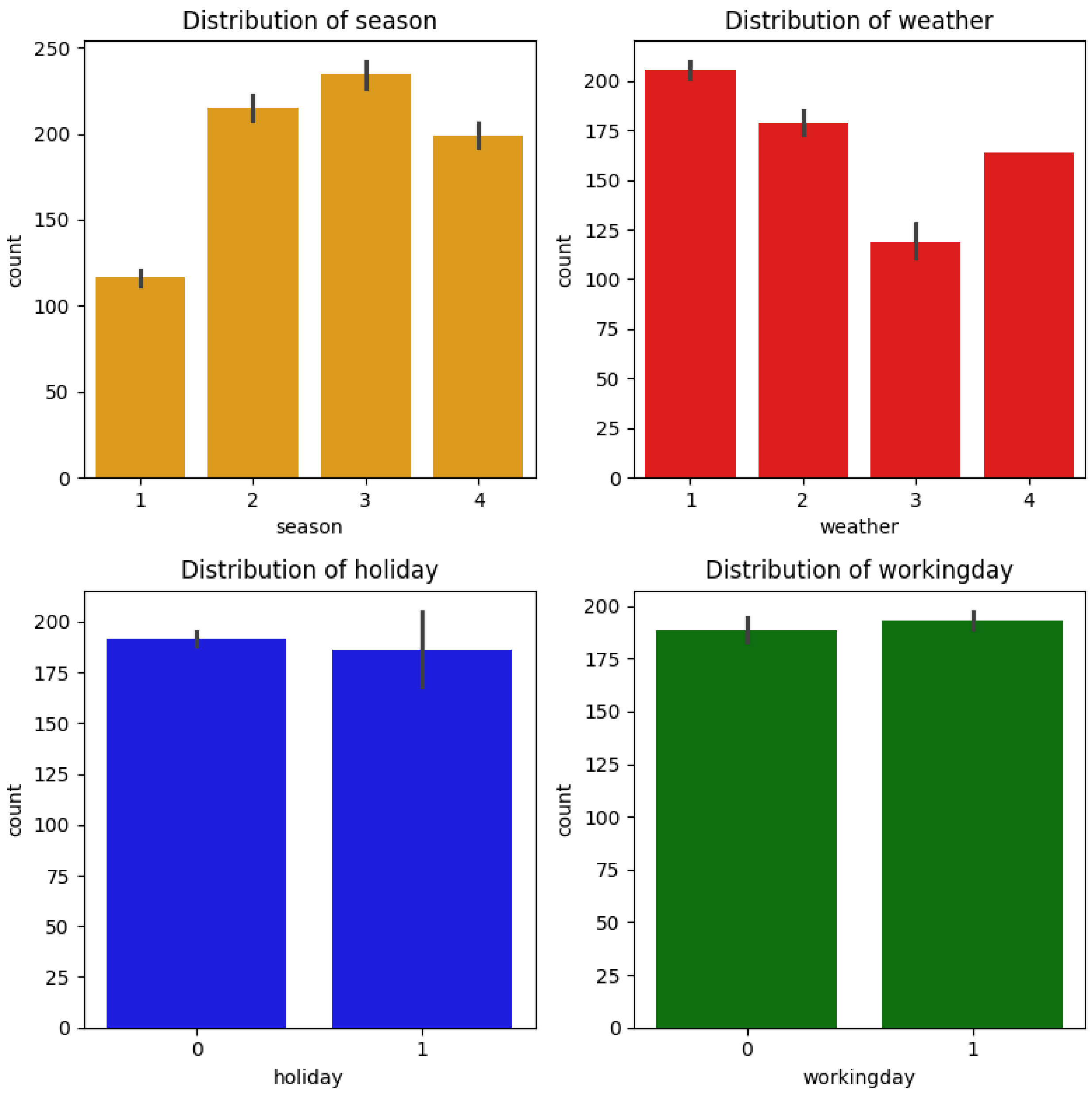


Insights from the distributions of **temp,atemp, windspeed**:

1. `temp` shows a fairly uniform and slightly right-skewed distribution, suggesting all temperature ranges are well represented.
2. `atemp` is also fairly uniform but has sharp peaks around 30°C, indicating repeated or common "feels-like" temperature values.
3. `windspeed` is right-skewed, with most observations clustered below 20 — indicating calm to moderate wind is more common.

In [23]:
```
fig,axes = plt.subplots(2,2,figsize=(8,8))
sns.barplot(data=data,x='season',y='count',ax=axes[0,0],color='orange')
```

```
axes[0,0].set_title('Distribution of season')
axes[0,0].set_xlabel('season')
axes[0,0].set_ylabel('count')

sns.barplot(data=data,x='weather',y='count',ax=axes[0,1],color='Red')
axes[0,1].set_title('Distribution of weather')
axes[0,1].set_xlabel('weather')
axes[0,1].set_ylabel('count')

sns.barplot(data=data,x='workingday',y='count',ax=axes[1,1],color='Green')
axes[1,1].set_title('Distribution of workingday')
axes[1,1].set_xlabel('workingday')
axes[1,1].set_ylabel('count')

sns.barplot(data=data,x='holiday',y='count',ax=axes[1,0],color='Blue')
axes[1,0].set_title('Distribution of holiday')
axes[1,0].set_xlabel('holiday')
axes[1,1].set_ylabel('count')

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



Insights from the bar plots of `season`, `weather`, `holiday`, and `workingday`:

1. **Season**: Most rides occur in **fall (season 3)**, while **spring (season 1)** sees the fewest — suggesting increased demand in cooler, post-summer months.
2. **Weather**: **Clear to partly cloudy days (weather 1)** have the highest ride count, whereas **severe weather (weather 3 & 4)** significantly reduces usage.
3. **Holiday**: Ride counts are nearly the same on **holidays and non-holidays**, indicating holidays don't majorly impact usage.
4. **Workingday**: **Working days and weekends/holidays** have similar ride demand — suggesting balanced weekday and weekend usage

patterns.

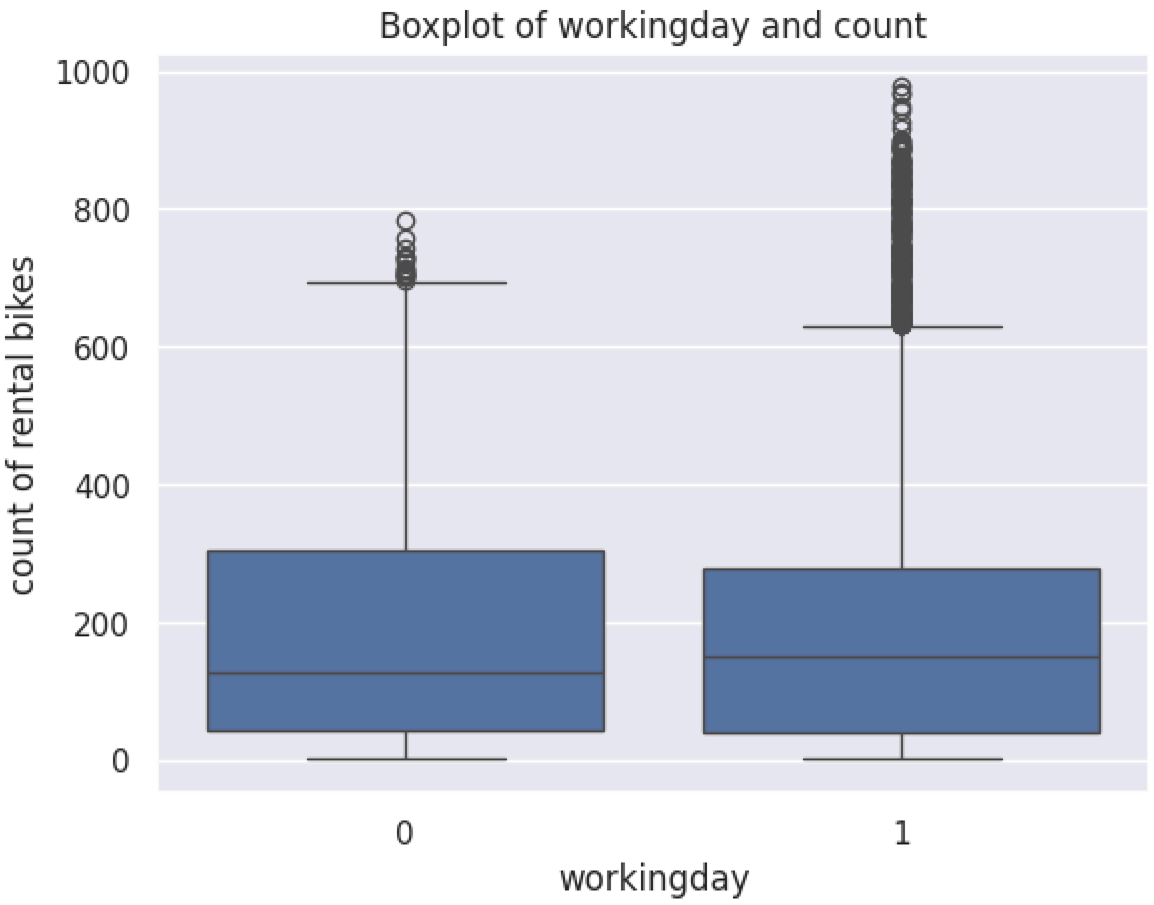
- **BI-VARIATE ANALYSIS**

```
In [64]: data.head()
```

Out[64]:

	datetime	season	holiday	workingday	weather	temp	atemp	humidity	w
0	2011-01-01 00:00:00	1	0	0	1	9.84	14.395	81	
1	2011-01-01 01:00:00	1	0	0	1	9.02	13.635	80	
2	2011-01-01 02:00:00	1	0	0	1	9.02	13.635	80	
3	2011-01-01 03:00:00	1	0	0	1	9.84	14.395	75	
4	2011-01-01 04:00:00	1	0	0	1	9.84	14.395	75	

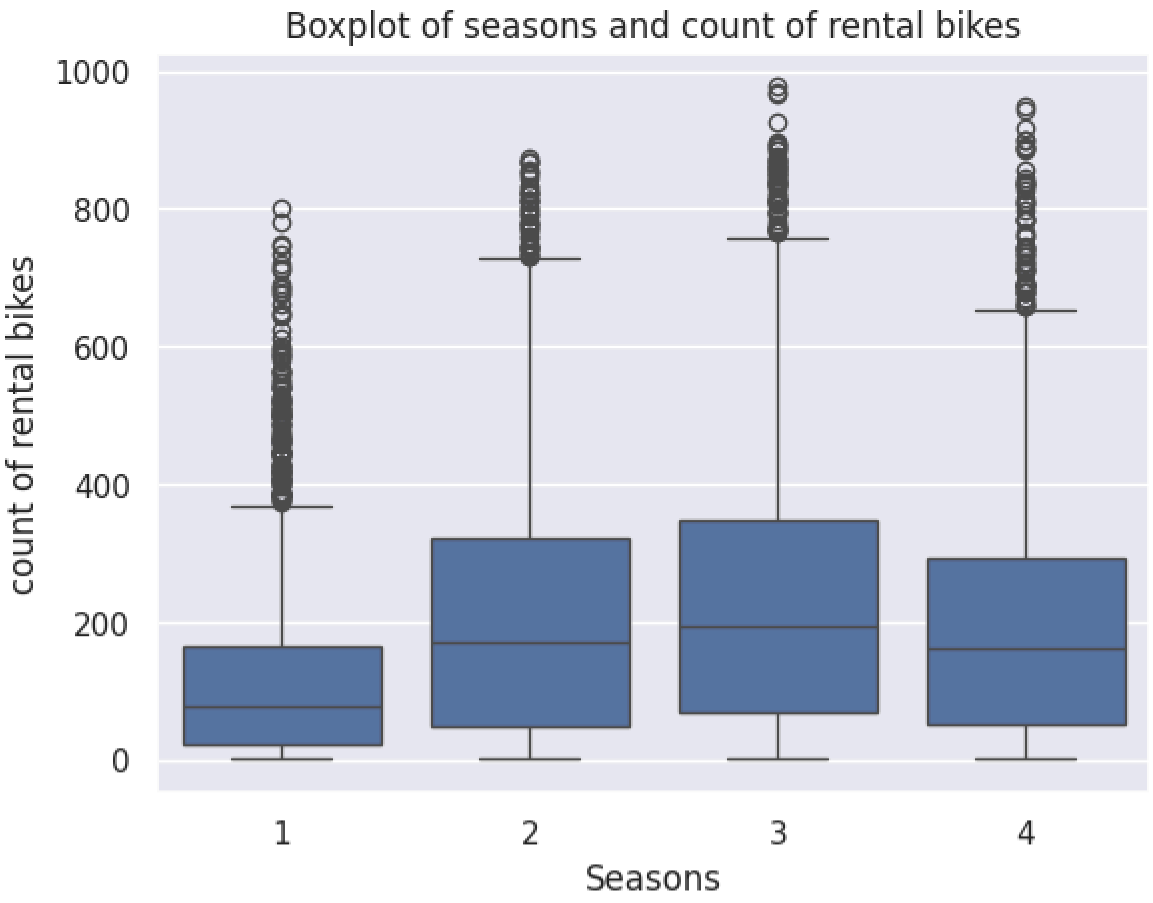
```
In [77]: sns.boxplot(data=data,x = data['workingday'],y = data['count'])
         plt.title('Boxplot of workingday and count')
         plt.xlabel('workingday')
         plt.ylabel('count of rental bikes')
         plt.show()
```



Insights:

On both working days and non-working days, the median bike rental count is similar, but working days show slightly higher upper-range usage and more extreme outliers, suggesting higher rental variability during the workweek.

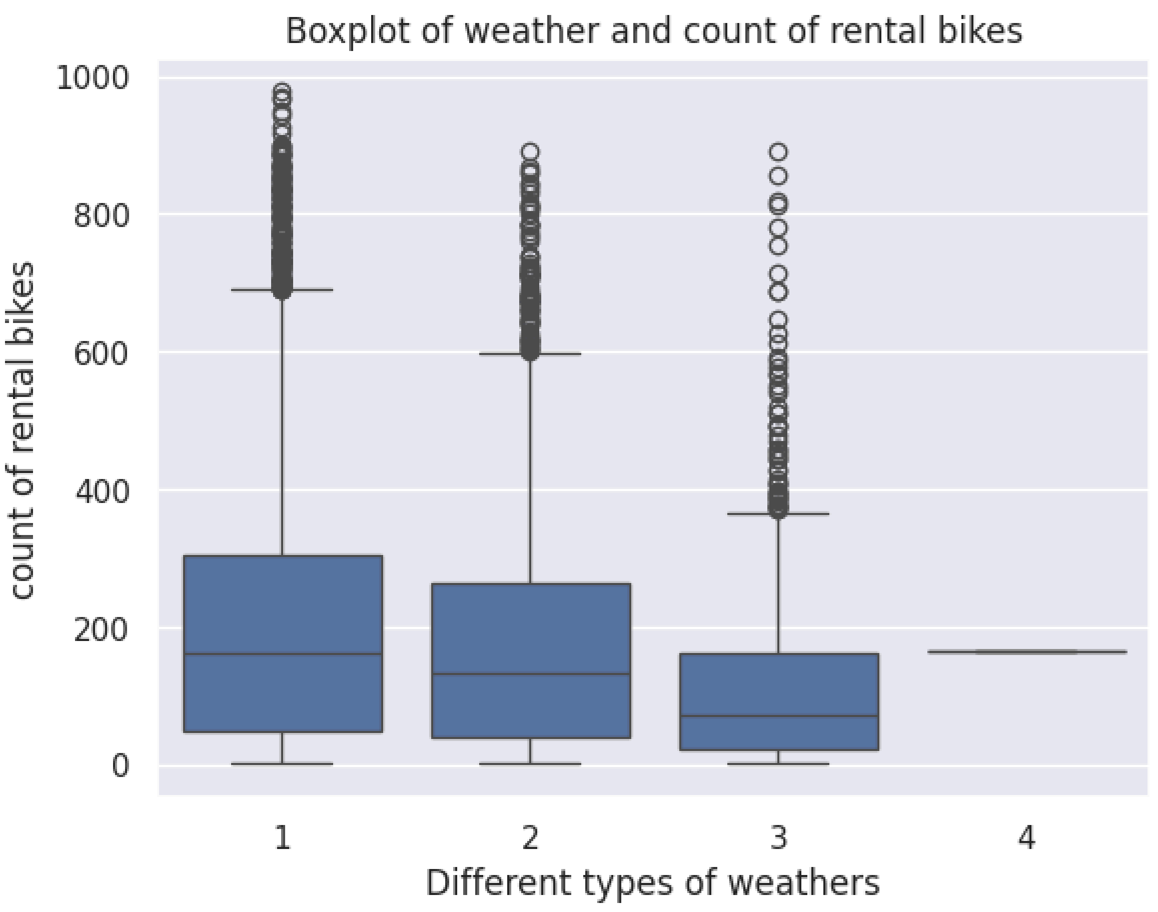
In [79]:
```
sns.boxplot(data=data,x = data['season'],y = data['count'])
plt.title('Boxplot of seasons and count of rental bikes')
plt.xlabel('Seasons')
plt.ylabel('count of rental bikes')
plt.show()
```



Insights from the boxplot of **season vs rental count**:

1. **Fall (Season 3)** shows the **highest median and widest spread** in rental counts, indicating both high demand and variability.
2. **Spring (Season 1)** has the **lowest median** and more compact distribution, suggesting lower and more consistent bike usage.
3. **Outliers are present in all seasons**, but are especially frequent in summer and fall, reflecting days with unusually high rental activity.

```
In [80]: sns.boxplot(data=data,x = data['weather'],y = data['count'])
         plt.title('Boxplot of weather and count of rental bikes')
         plt.xlabel('Different types of weathers')
         plt.ylabel('count of rental bikes')
         plt.show()
```



Insights from the boxplot of **weather vs rental bike count**:

1. **Clear to partly cloudy weather (type 1)** sees the **highest median and widest spread** in rentals, indicating strong and variable demand.
2. As weather worsens from **type 2 to type 3**, **median rentals decline noticeably**, showing reduced usage in misty or rainy conditions.
3. **Severe weather (type 4)** has almost no variability and extremely low counts, implying bike rentals are nearly absent during extreme weather events.

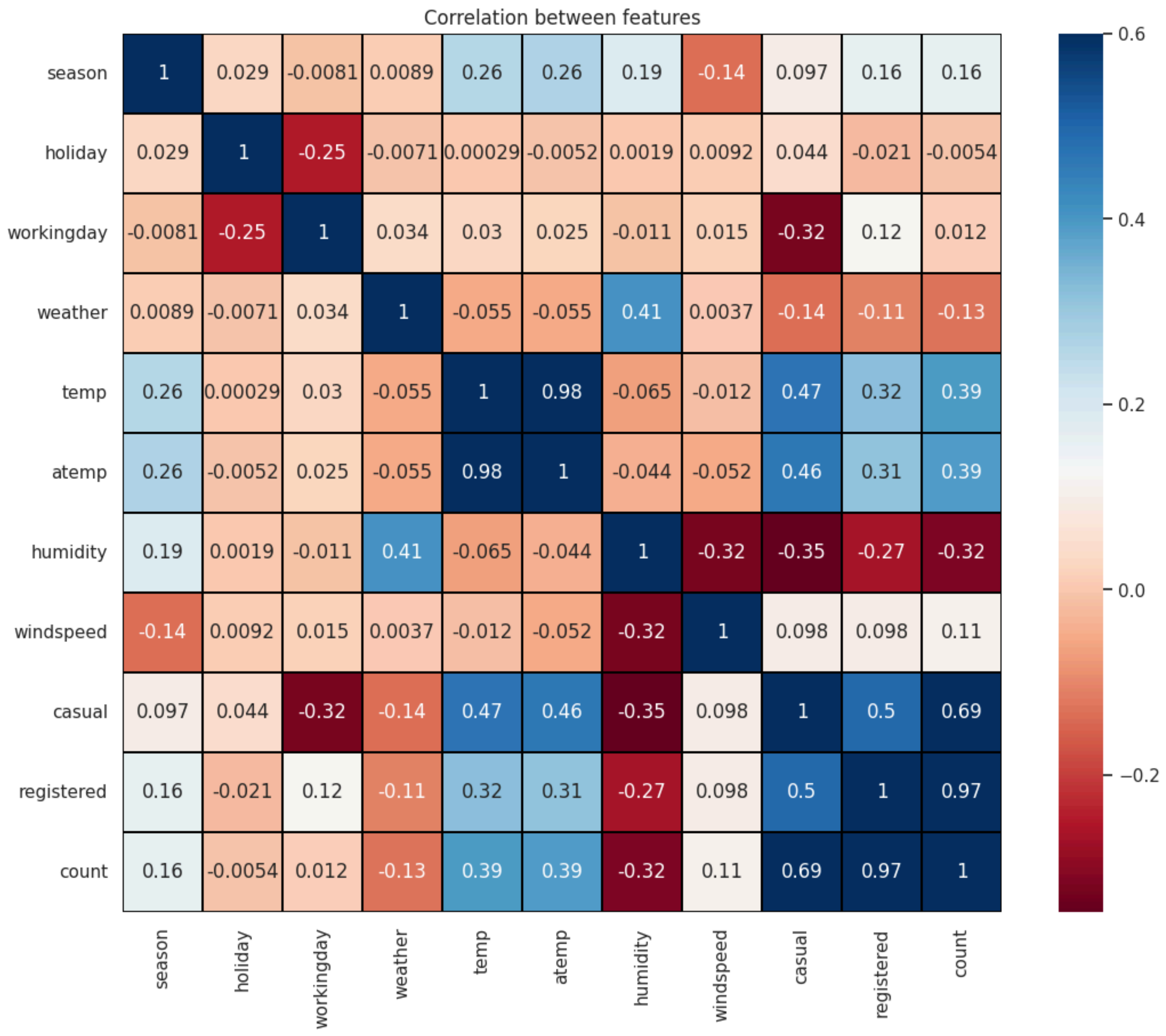
Correlation between features

In [24]:
```
data1 = data.copy()
```

In [25]:
```
data1.drop(columns=['datetime'],inplace=True)
```

In [26]:
```
sns.set(font_scale=1.0)
plt.figure(figsize=(13,10))
correlation_values = data1.corr(method = 'pearson')
sns.heatmap(correlation_values, vmax = .6, linewidths=0.01, square=True, annot
plt.title('Correlation between features')
```

```
Out[26]: Text(0.5, 1.0, 'Correlation between features')
```



	season	holiday	workingday	weather	temp	atemp	humidity	windspeed	casual	registered	count
season	1	0.029	-0.0081	0.0089	0.26	0.26	0.19	-0.14	0.097	0.16	0.16
holiday	0.029	1	-0.25	-0.0071	0.00029	-0.0052	0.0019	0.0092	0.044	-0.021	-0.0054
workingday	-0.0081	-0.25	1	0.034	0.03	0.025	-0.011	0.015	-0.32	0.12	0.012
weather	0.0089	-0.0071	0.034	1	-0.055	-0.055	0.41	0.0037	-0.14	-0.11	-0.13
temp	0.26	0.00029	0.03	-0.055	1	0.98	-0.065	-0.012	0.47	0.32	0.39
atemp	0.26	-0.0052	0.025	-0.055	0.98	1	-0.044	-0.052	0.46	0.31	0.39
humidity	0.19	0.0019	-0.011	0.41	-0.065	-0.044	1	-0.32	-0.35	-0.27	-0.32
windspeed	-0.14	0.0092	0.015	0.0037	-0.012	-0.052	-0.32	1	0.098	0.098	0.11
casual	0.097	0.044	-0.32	-0.14	0.47	0.46	-0.35	0.098	1	0.5	0.69
registered	0.16	-0.021	0.12	-0.11	0.32	0.31	-0.27	0.098	0.5	1	0.97
count	0.16	-0.0054	0.012	-0.13	0.39	0.39	-0.32	0.11	0.69	0.97	1

Insights based on your correlation heatmap:

1. **`temp` and `atemp`** are highly correlated (**0.98**) — drop one to avoid redundancy.
2. **`registered` and `count`** show a very strong correlation (**0.97**) — `count` is largely driven by registered users.
3. **`casual` and `count`** have a strong positive relationship (**0.69**) — casual users also significantly impact total rides.
4. **`humidity`** is negatively correlated with `count` (**-0.32**) — more humidity tends to lower ride counts.
5. **`weather`** and `humidity` are moderately correlated (**0.41**) — bad weather likely means higher humidity.
6. **`temp` and `count`** have a moderate positive correlation (**0.39**) —

higher temperatures encourage more rides.

7. `workingday` is weakly related to `count` — rides are fairly consistent across workdays and non-workdays.
8. `holiday` has almost no effect on ride `count` — correlation is nearly zero.
9. `windspeed` has negligible correlation with most features — likely less influential.
10. `season` has weak positive correlations with `temp`, `atemp`, and `count`.

Checking any significant difference between the no. of bike rides on Weekdays and Weekends?

Hypothesis Framework

- Null hypothesis(Ho) - No relation between weekdays and weekends.
- Alternative hypothesis(Ha) - Significance difference between weekdays and weekends.

Appropriate test

- Sample Independent T-test

SIgnifcance level (5%)

- Alpha = 0.05

In [27]:
```
weekend_rides = data[data["workingday"]==1]["count"].to_list()
weekdays_rides = data[data["workingday"]==0]["count"].to_list()
```

In [28]:
```
from scipy.stats import ttest_ind
```

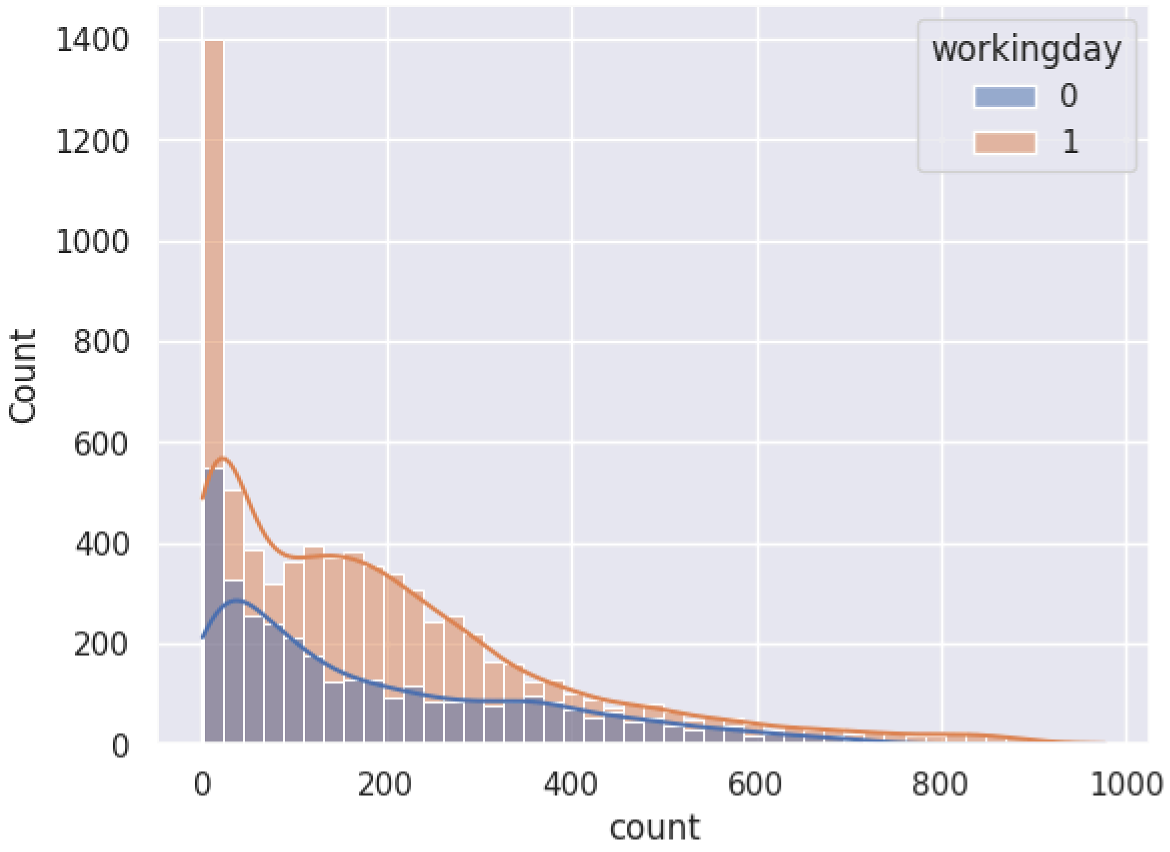
In [29]:
```
t_statistics,p_val = ttest_ind(weekend_rides,weekdays_rides)

print(f"The t-statistics is {t_statistics}")
print(f"The p-value is {p_val}")
```

```
The t-statistics is 1.2096277376026694
The p-value is 0.22644804226361348
```

In [30]:
```
Alpha = 0.05

if p_val < Alpha:
  print("Reject the null hypothesis , there is significant difference between
else:
  print("Fail to reject the null hypothesis , there is no significant differer
```

```
Fail to reject the null hypothesis , there is no significant difference between
two features
```

In [31]:
```
sns.histplot(data=data,x='count',hue='workingday',kde=True)
plt.show()
```



Interpretation :

- Users are consistently renting bikes across all days, regardless of whether it's a weekday or weekend/holiday.
- The demand pattern is stable and steady.

Recommendations:

- Maintain uniform operations throughout the week — no need to increase or reduce availability on weekends vs weekdays.
- Consider targeted promotions or discounts during low-traffic hours rather than by day-type.

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

Hypothesis Framework

- Null hypothesis (Ho) - There is no significant difference in group means.
- Alternative hypothesis (Ha) - At least one group mean differs

Appropriate test

- One-way Anova test

SIgnifcance level (5%)

- Alpha = 0.05

The assumptions of one-way ANOVA are critical to ensure that the test results are valid. There are three main assumptions:

- Independence of Observations
- Normality
- Homogeneity of Variances

- **Independence of Observations:**

In [32]:
```
data[data["weather"]==1].duplicated().sum() , data[data["weather"]==2].duplica
```

Out[32]:
```
(np.int64(0), np.int64(0), np.int64(0), np.int64(0))
```

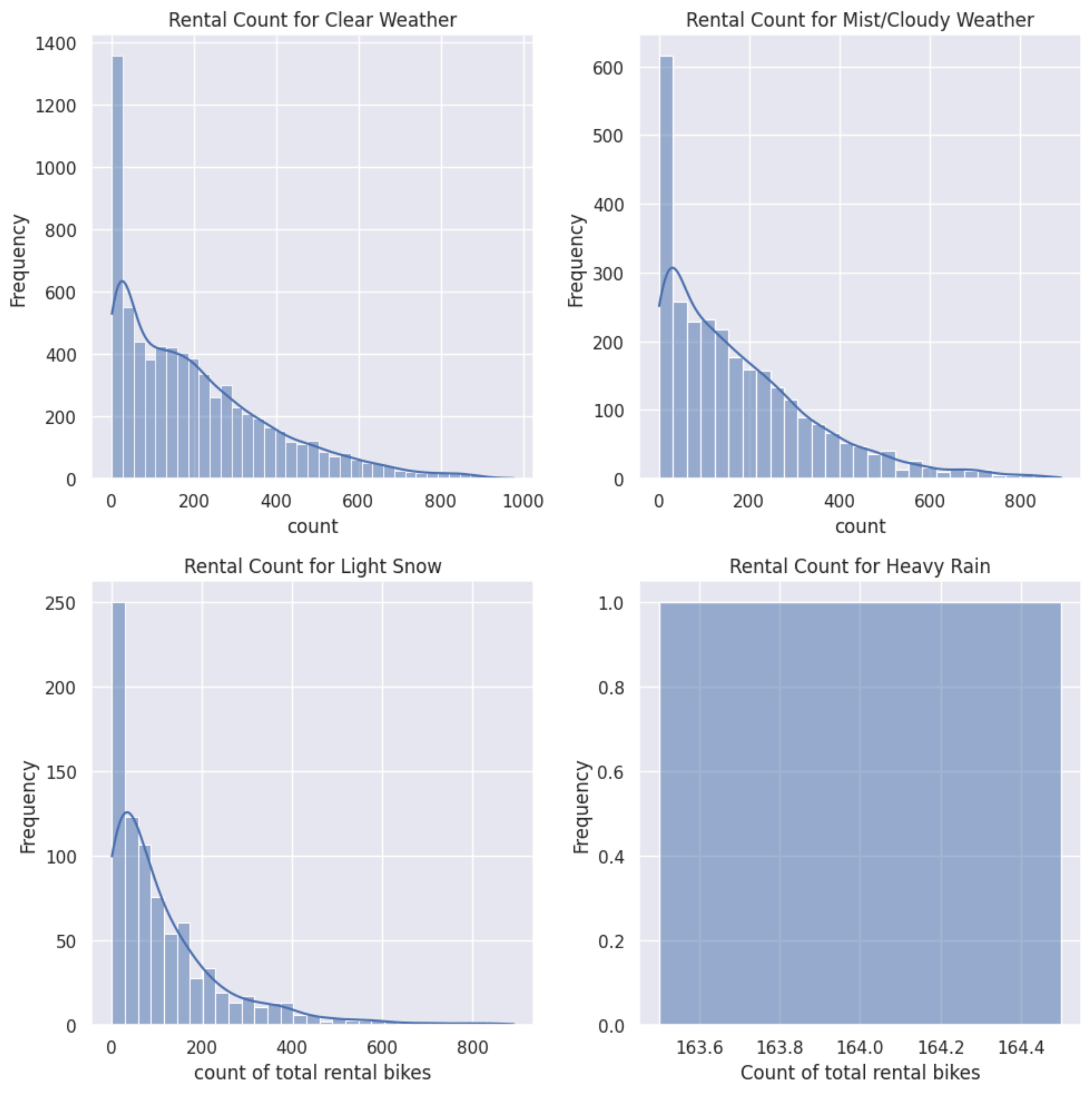
- From a duplication standpoint, each row is unique — which supports the independence assumption for ANOVA.

- **Normality**

1. Using Histplot

In [33]:
```
rental_clear = data[data["weather"]==1]['count']
rental_Mist_cloudy = data[data["weather"]==2]['count']
rental_light_snow = data[data["weather"]==3]['count']
rental_heavy_rain = data[data["weather"]==4]['count']
```

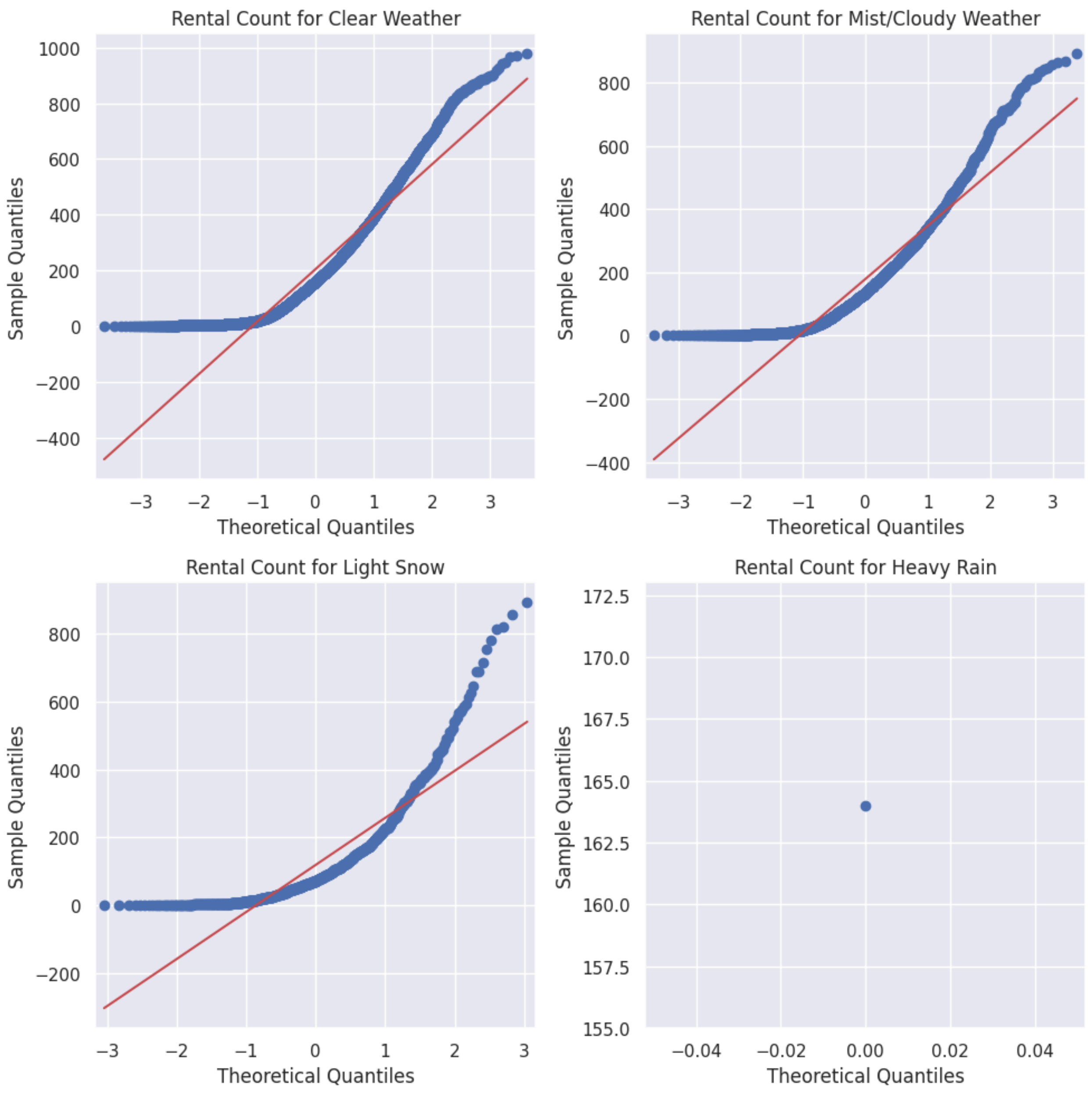
```
figs,axes=plt.subplots(2,2,figsize=(10,10))

sns.histplot(rental_clear,ax=axes[0,0],kde=True)
axes[0,0].set_title('Rental Count for Clear Weather')
axes[0,0].set_ylabel('Frequency')

sns.histplot(rental_Mist_cloudy,ax=axes[0,1],kde=True)
axes[0,1].set_title('Rental Count for Mist/Cloudy Weather')
axes[0,1].set_ylabel('Frequency')

sns.histplot(rental_light_snow,ax=axes[1,0],kde=True)
axes[1,0].set_title('Rental Count for Light Snow')
axes[1,0].set_xlabel('count of total rental bikes')
axes[1,0].set_ylabel('Frequency')

sns.histplot(rental_heavy_rain,ax=axes[1,1],kde=True)
axes[1,1].set_title('Rental Count for Heavy Rain')
axes[1,1].set_xlabel('Count of total rental bikes')
axes[1,1].set_ylabel('Frequency')

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



QQ - Plot

```
In [34]: from statsmodels.graphics.gofplots import qqplot
figs,axes=plt.subplots(2,2,figsize=(10,10))

qqplot(rental_clear,line='s',ax=axes[0,0])
axes[0,0].set_title('Rental Count for Clear Weather')

qqplot(rental_Mist_cloudy,line='s',ax=axes[0,1])
axes[0,1].set_title('Rental Count for Mist/Cloudy Weather')

qqplot(rental_light_snow,line='s',ax=axes[1,0])
axes[1,0].set_title('Rental Count for Light Snow')

qqplot(rental_heavy_rain,line='s',ax=axes[1,1])
```

```
axes[1,1].set_title('Rental Count for Heavy Rain')

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



Skewness & Kurtosis

```
In [35]: from scipy.stats import skew,kurtosis

skew(rental_clear),skew(rental_Mist_cloudy),skew(rental_light_snow),skew(renta

print(f"The skewness of rental_clear is {skew(rental_clear)}")
print(f"The skewness of rental_Mist_cloudy is {skew(rental_Mist_cloudy)}")
print(f"The skewness of rental_light_snow is {skew(rental_light_snow)}")
print(f"The skewness of rental_heavy_rain is {skew(rental_heavy_rain)}")
```

```
The skewness of rental_clear is 1.1396195185041555
The skewness of rental_Mist_cloudy is 1.293759189703101
The skewness of rental_light_snow is 2.1833160390123187
The skewness of rental_heavy_rain is nan
```

In [36]:
```python
kurtosis(rental_clear),kurtosis(rental_Mist_cloudy),kurtosis(rental_light_snow

print(f"The kurtosis of rental_clear is {kurtosis(rental_clear)}")
print(f"The kurtosis of rental_Mist_cloudy is {kurtosis(rental_Mist_cloudy)}")
print(f"The kurtosis of rental_light_snow is {kurtosis(rental_light_snow)}")
print(f"The kurtosis of rental_heavy_rain is {kurtosis(rental_heavy_rain)}")
```

```
The kurtosis of rental_clear is 0.9632151489948488
The kurtosis of rental_Mist_cloudy is 1.5835130178554868
The kurtosis of rental_light_snow is 5.961191782478394
The kurtosis of rental_heavy_rain is nan
```

Interpretation :

- Skewness & Kurtosis

Skewness > 0 → Positive Skew (Right Skewed)

Skewness < 0 → Negative Skew (Left Skewed)

Skewness ≈ 0 → Symmetrical / Normal

Kurtosis > 0 → Leptokurtic (Peaked distribution, heavy tails)

Kurtosis = 0 → Mesokurtic (Normal bell-shaped)

Kurtosis < 0 → Platykurtic (Flat-topped, light tails)

- QQ-PLOT

The datapoints aren't normally distributed here due to lack of data variation.

- Histplot

All distributions are positively skewed, especially under light snow.

2.Shapiro-Wilk’s test

In [37]:
```python
from scipy.stats import shapiro
import warnings
warnings.simplefilter("ignore")

shapiro(rental_clear),shapiro(rental_Mist_cloudy),shapiro(rental_light_snow),s

print(f"The p-value for rental_clear is {shapiro(rental_clear)[1]}")
print(f"The p-value for rental_Mist_cloudy is {shapiro(rental_Mist_cloudy)[1]}
print(f"The p-value for rental_light_snow is {shapiro(rental_light_snow)[1]}")
print(f"The p-value for rental_heavy_rain is {shapiro(rental_heavy_rain)[1]}")
```

```
The p-value for rental_clear is 1.5964921477006555e-57
The p-value for rental_Mist_cloudy is 9.777839106111785e-43
The p-value for rental_light_snow is 3.875893017396149e-33
The p-value for rental_heavy_rain is nan
```

NOTE :

Using warnings as for large datasets - shapiro test is not recommended

All the p-values are less than 0.05(significance level) - reject the Normality.

Henceforth, we can say the data with respect to weather and number of rental bikes doesn't follow the normal distribution.

- **Equality Variance**

In [38]:
```
from scipy.stats import levene

levene_stats,lev_pval = levene(rental_clear,rental_Mist_cloudy,rental_light_sn

print(f"The levene_stats is {levene_stats}")
print(f"The lev_pval is {lev_pval}")

Alpha = 0.05

if lev_pval < Alpha:
  print("Reject the null hypothesis : Variances are not equal")
else:
  print("Fail to reject the null hypothesis :  Variances are equal")
```

```
The levene_stats is 54.85106195954556
The lev_pval is 3.504937946833238e-35
Reject the null hypothesis : Variances are not equal
```

Note : It's robust to non-normal data, so it’s appropriate here since your data is not normally distributed.

Final Note - The assumptions of one-way Anova is failed .

In [39]:
```
#One-way Anova

from scipy.stats import f_oneway

f_stat, anova_p_val = f_oneway(rental_clear, rental_Mist_cloudy, rental_light_

print(f"F-statistic: {f_stat}")
print(f"p-value: {anova_p_val}")
print("-----------------------")

Alpha = 0.05

if anova_p_val < Alpha:
  print("Reject the null hypothesis , At least one group mean differs")
else:
  print("Fail to reject the null hypothesis , there is no significant differen
```

```
F-statistic: 65.53024112793271
p-value: 5.482069475935669e-42
-----------------------
Reject the null hypothesis , At least one group mean differs
```

As it never follows the anova assumptions - so we use kruskal

In [83]:
```
from scipy.stats import kruskal

kruskal_stat, kruskal_p_val = kruskal(rental_clear, rental_Mist_cloudy, rental

Alpha = 0.05

if kruskal_p_val < Alpha:
  print("Reject the null hypothesis , At least one group mean differs")
else:
  print("Fail to reject the null hypothesis , there is no significant differen
```

```
Reject the null hypothesis , At least one group mean differs
```

Conclusion

There is **strong statistical evidence** that at least one weather condition significantly affects rental counts. Therefore, **weather conditions do influence Yulu bike rentals**.

In such cases, using **non-parametric alternatives like Kruskal-Wallis test** is recommended for more reliable results.

Recommendations

1. **Weather-aware Demand Planning**:
   - Increase inventory or availability on **clear days**.
   - Prepare for **low demand during heavy rain/light snow** by reallocating resources.
   - Introduce **dynamic pricing or incentives** during bad weather to boost usage.
2. **Model Weather Impact**: Incorporate weather features into machine

learning models for **rental forecasting** or **predictive maintenance**.

3. **Customer Communication**: Notify users of safety and availability during adverse weather (mist, rain, snow) via app alerts or offers.

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

Hypothesis Framework

- Null hypothesis (Ho) - There is no significant difference in group means.
- Alternative hypothesis (Ha) - At least one group mean differs

Appropriate test

- One-way Anova test

SIgnifcance level (5%)

- Alpha = 0.05

The assumptions of one-way ANOVA are critical to ensure that the test results are valid. There are three main assumptions:

- Independence of Observations
- Normality
- Homogeneity of Variances

- **Independence of Observations**

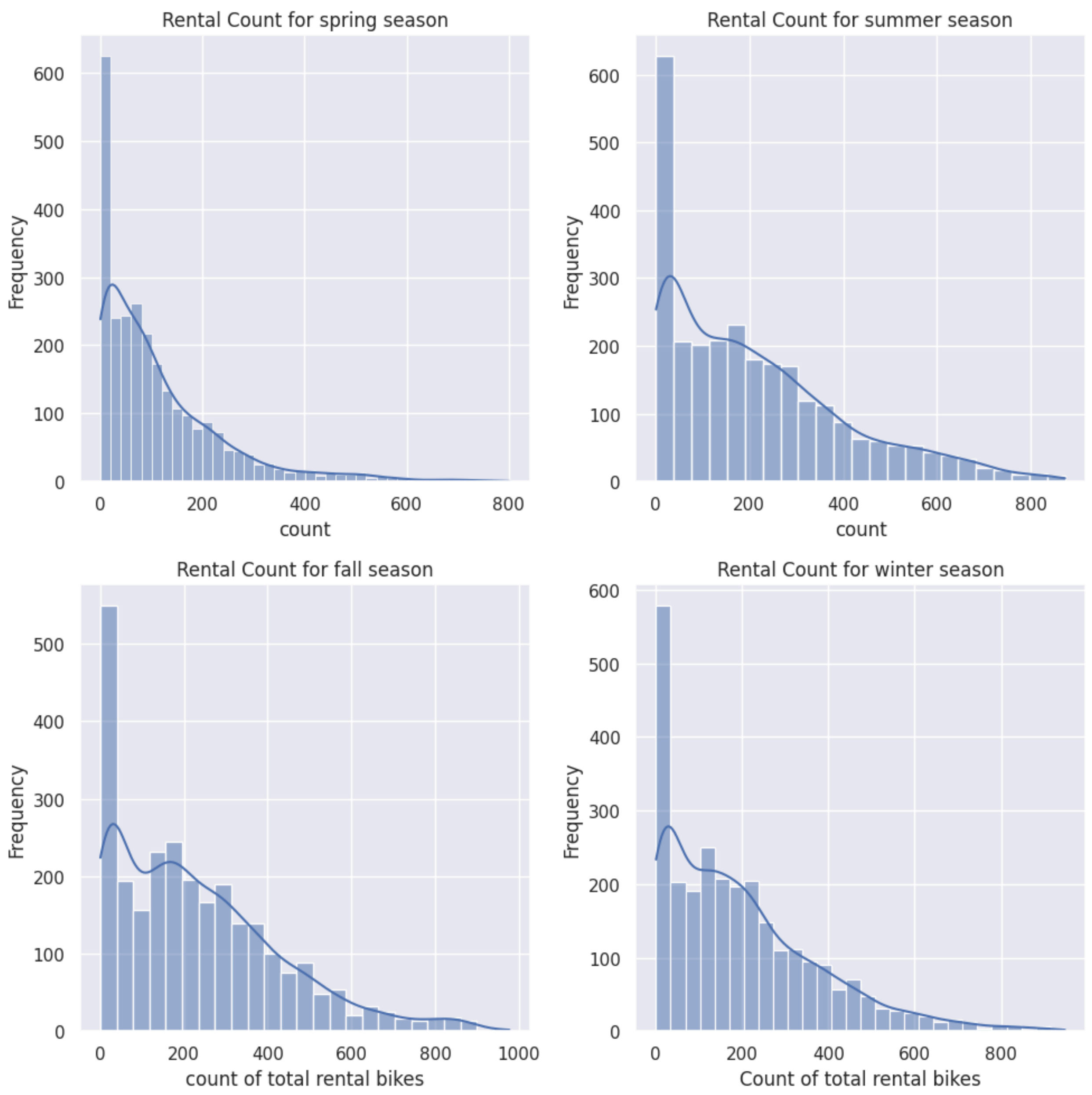
From a duplication standpoint, each row is unique — which supports the independence assumption for ANOVA.

- **Normality**

1.Using Histplot

In [40]:
```
spring_rentals = data[data["season"]==1]['count']
summer_rentals = data[data["season"]==2]['count']
fall_rentals = data[data["season"]==3]['count']
winter_rentals = data[data["season"]==4]['count']
```

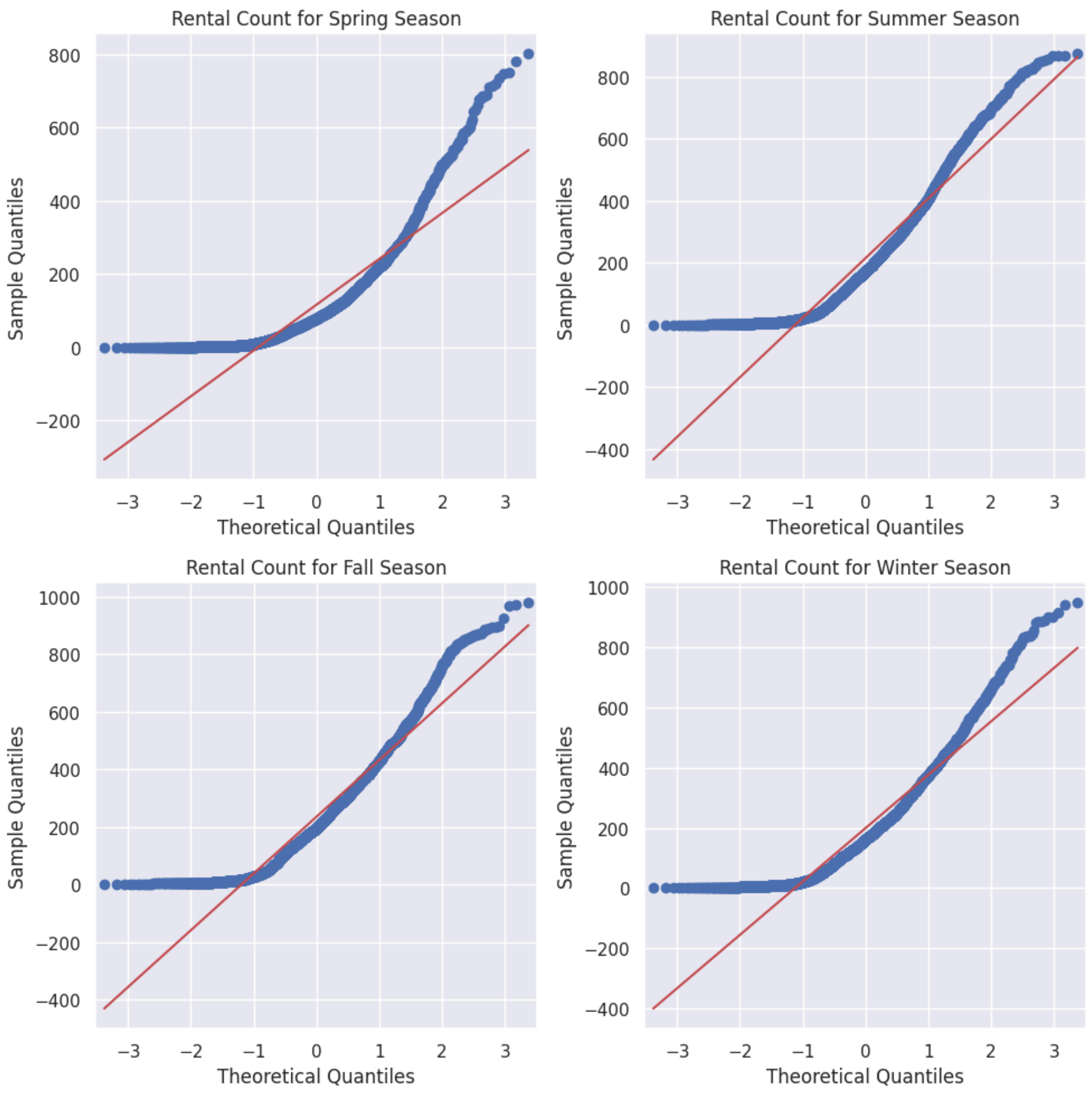
```
figs,axes=plt.subplots(2,2,figsize=(10,10))

sns.histplot(spring_rentals,ax=axes[0,0],kde=True)
axes[0,0].set_title('Rental Count for spring season')
axes[0,0].set_ylabel('Frequency')

sns.histplot(summer_rentals,ax=axes[0,1],kde=True)
axes[0,1].set_title('Rental Count for summer season')
axes[0,1].set_ylabel('Frequency')

sns.histplot(fall_rentals,ax=axes[1,0],kde=True)
axes[1,0].set_title('Rental Count for fall season')
axes[1,0].set_xlabel('count of total rental bikes')
axes[1,0].set_ylabel('Frequency')

sns.histplot(winter_rentals,ax=axes[1,1],kde=True)
axes[1,1].set_title('Rental Count for winter season')
axes[1,1].set_xlabel('Count of total rental bikes')
axes[1,1].set_ylabel('Frequency')

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



QQ-PLOT

In [41]:
```
from statsmodels.graphics.gofplots import qqplot
figs,axes=plt.subplots(2,2,figsize=(10,10))

qqplot(spring_rentals,line='s',ax=axes[0,0])
axes[0,0].set_title('Rental Count for Spring Season')

qqplot(summer_rentals,line='s',ax=axes[0,1])
axes[0,1].set_title('Rental Count for Summer Season')

qqplot(fall_rentals,line='s',ax=axes[1,0])
axes[1,0].set_title('Rental Count for Fall Season')

qqplot(winter_rentals,line='s',ax=axes[1,1])
```

```
axes[1,1].set_title('Rental Count for Winter Season')

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



Skewness & Kurtosis

In [42]:
```
from scipy.stats import skew,kurtosis

skew(spring_rentals),skew(summer_rentals),skew(fall_rentals),skew(winter_renta

print(f"The skewness of spring_rentals is {skew(spring_rentals)}")
print(f"The skewness of summer_rentals is {skew(summer_rentals)}")
print(f"The skewness of fall_rentals is {skew(fall_rentals)}")
print(f"The skewness of winter_rentals is {skew(winter_rentals)}")
```

```
The skewness of spring_rentals is 1.8870013494363214
The skewness of summer_rentals is 1.0027135037743604
The skewness of fall_rentals is 0.9909503852121176
The skewness of winter_rentals is 1.1714741534595685
```

In [43]:
```
kurtosis(spring_rentals),kurtosis(summer_rentals),kurtosis(fall_rentals),kurto

print(f"The kurtosis of spring_rentals is {kurtosis(spring_rentals)}")
print(f"The kurtosis of summer_rentals is {kurtosis(summer_rentals)}")
print(f"The kurtosis of fall_rentals is {kurtosis(fall_rentals)}")
print(f"The kurtosis of winter_rentals is {kurtosis(winter_rentals)}")
```

```
The kurtosis of spring_rentals is 4.30449666648592
The kurtosis of summer_rentals is 0.4222412657621657
The kurtosis of fall_rentals is 0.6959091337333851
The kurtosis of winter_rentals is 1.2689637849725477
```

Interpretation :

- Skewness & Kurtosis

Skewness > 0 → Positive Skew (Right Skewed)

Skewness < 0 → Negative Skew (Left Skewed)

Skewness ≈ 0 → Symmetrical / Normal

Kurtosis > 0 → Leptokurtic (Peaked distribution, heavy tails)

Kurtosis = 0 → Mesokurtic (Normal bell-shaped)

Kurtosis < 0 → Platykurtic (Flat-topped, light tails)

- QQ-PLOT

The datapoints aren't normally distributed here due to lack of data variation.

- Histplot

All distributions are positively skewed, especially under light snow.

2.Shapiro-Wilk's test

In [44]:
```
from scipy.stats import shapiro

shapiro(rental_clear),shapiro(rental_Mist_cloudy),shapiro(rental_light_snow),s

print(f"The p-value for spring_rentals is {shapiro(spring_rentals)[1]}")
print(f"The p-value for summer_rentals is {shapiro(summer_rentals)[1]}")
print(f"The p-value for fall_rentals is {shapiro(fall_rentals)[1]}")
print(f"The p-value for winter_rentals is {shapiro(winter_rentals)[1]}")
```

```
The p-value for spring_rentals is 8.749584618867662e-49
The p-value for summer_rentals is 6.039374406270491e-39
The p-value for fall_rentals is 1.043680518918597e-36
The p-value for winter_rentals is 1.1299244409282836e-39
```

NOTE :

All the p-values are less than 0.05(significance level) - reject the Normality.

Henceforth, we can say the data with respect to weather and number of rental bikes doesn't follow the normal distribution.

- **Equality Variance**

In [45]:
```
from scipy.stats import levene

levene_stats,sea_lev_pval = levene(spring_rentals,summer_rentals,fall_rentals,

print(f"The levene_stats is {levene_stats}")
print(f"The p_value is {sea_lev_pval}","\n")


Alpha = 0.05

if sea_lev_pval < Alpha:
  print("Reject the null hypothesis : Variances are not equal")
else:
  print("Fail to reject the null hypothesis :  Variances are equal")
```

```
The levene_stats is 187.7706624026276
The p_value is 1.0147116860043298e-118

Reject the null hypothesis : Variances are not equal
```

Note : It's robust to non-normal data, so it’s appropriate here since your data is not normally distributed.

Final Note - The assumptions of one-way Anova is failed .

In [46]:
```
#One way Anova - demand of bicycles on rent is the same for different Seasons

from scipy.stats import f_oneway

f_stat, season_anova_p_val = f_oneway(spring_rentals, summer_rentals, fall_rer

print(f"F-statistic: {f_stat}")
print(f"p-value: {season_anova_p_val}","\n")
```

```
Alpha = 0.05

if season_anova_p_val < Alpha:
  print("Reject the null hypothesis , At least one group mean differs")
else:
  print("Fail to reject the null hypothesis , there is no significant differer
```

```
F-statistic: 236.94671081032106
p-value: 6.164843386499654e-149

Reject the null hypothesis , At least one group mean differs
```

As it never follows the anova assumptions - so we use kruskal

In [85]:
```
from scipy.stats import kruskal

kruskal_stat, season_kruskal_p_val = kruskal(spring_rentals, summer_rentals, f

Alpha = 0.05

if season_kruskal_p_val < Alpha:
  print("Reject the null hypothesis , At least one group mean differs")
else:
  print("Fail to reject the null hypothesis , there is no significant differer
```

```
Reject the null hypothesis , At least one group mean differs
```

inferences and **recommendations** from both the ANOVA and Kruskal-Wallis tests regarding **seasonal bicycle rental patterns**:

Inference

- There is **strong statistical evidence** that bicycle rental demand **varies significantly across seasons**.
- Since the Kruskal-Wallis test does **not rely on normality assumptions**, its result is **more trustworthy** here.
- Hence, **seasonality has a significant impact** on the rental patterns.

Business Recommendations

1. **Season-Specific Demand Planning**

- **Spring/Summer** likely have higher rentals — prepare more inventory (bikes, maintenance, staff).
- **Fall/Winter** may see lower demand — optimize operational costs, offer discounts to balance usage.

2. **Dynamic Pricing Strategy**

    - Introduce **seasonal pricing**:
        - **Premium pricing** during high-demand seasons (summer, spring).
        - **Discounts or promotional offers** during low-demand periods (winter, rainy fall).

3. **Marketing Campaigns**

    - **Target marketing efforts** during low-demand seasons (e.g., winter) to boost rentals.
    - Promote **seasonal subscription plans** during transitions (spring → summer, fall → winter).

4. **Operational Optimization**

    - Reallocate resources (bikes, charging stations) **geographically** based on seasonal demand patterns.
    - Adjust **fleet size and placement** dynamically to reduce idle bikes during off-seasons.

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

Hypothesis Framework

- Null hypothesis (Ho) - No relation between weather condition and seasons.
- Alternative hypothesis (Ha) - Significance difference between weather condition and seasons.

Appropriate test

- Chi-Sq

SIgnifcance level (5%)

- Alpha = 0.05

In [47]:
```
weather_season = pd.crosstab(data["weather"],data["season"])
```

```
from scipy.stats import chi2_contingency

Chi2ContingencyResult , chi_pvalue , dof, expected_freq = chi2_contingency(wea

print(f"Chi2ContingencyResult : {Chi2ContingencyResult}")
print(f"pvalue : {chi_pvalue}","\n")

Alpha = 0.05

if chi_pvalue < Alpha:
  print("Reject the null hypothesis ,Significance difference between weather c
else:
  print("Fail to reject the null hypothesis , No relation between weather conc
```

```
Chi2ContingencyResult : 49.15865559689363
pvalue : 1.5499250736864862e-07

Reject the null hypothesis ,Significance difference between weather condition a
nd seasons.
```

Inference & Recommendations from Chi-Square Test of Independence (Weather vs Season)

Practical Interpretation

- **Rain and Snow** likely occur more often in **winter or fall**.
- **Clear or sunny weather** is likely concentrated in **summer and spring**.
- **Mist and cloudy conditions** may peak during transitional seasons like **fall and spring**.

This variation is statistically significant and **not due to random chance**.

◈ Business Recommendations for Yulu Rentals

1. **Integrate Seasonal Weather Forecasting**

- Plan operations using **season-specific weather patterns**.
- Allocate **bike inventory, servicing, and support staff** based on expected weather trends.

2. **Design Weather-Sensitive Campaigns**

- Offer **rain or winter ride incentives** (discounts, loyalty points).
- Run **summer campaigns** promoting leisure rides in good weather.

3. **Predictive Modeling**

    - Enhance rental demand models by including **weather-season interaction** as a feature.
    - Improves **forecasting accuracy** for resource allocation, pricing, and fleet balancing.

4. **User Safety & Communication**

    - Send **weather-based alerts** or safety tips, especially during seasons known for poor conditions (e.g., snow in winter).
    - Highlight **weather-based insurance or coverage** for users during high-risk periods.

5. **Dynamic Pricing**

    - Implement **seasonal pricing** that adapts to both **weather severity and frequency**.

        - For example: Increase rates on clear days in summer; decrease during rainy fall days to boost usage.

6. **Operational Buffering**

    - Increase **maintenance checks** before weather-intensive seasons (like winter).
    - Deploy **weather protection infrastructure** (e.g., bike covers, station shelters) in seasons with high rainfall or snow.

```
In [47]:
```