# yulu-hypothesis-testing

July 10, 2024

## 1 Yulu - Hypothesis Testing Case Study

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!

## 2 Objective

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.

## 3 Column Profiling:

datetime: datetime

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

holiday: whether day is a holiday or not (extracted from http://dchr.dc.gov/page/holiday-schedule)

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

[1]:
```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.stats import norm,zscore,boxcox,probplot
from statsmodels.stats import weightstats as stests
from statsmodels.stats.proportion import proportions_ztest
from scipy.stats import ttest_ind, ttest_rel, ttest_1samp, mannwhitneyu
from scipy.stats import chisquare, chi2, chi2_contingency
from scipy.stats import f_oneway, kruskal, shapiro, levene, kstest
import statsmodels.api as sm
from statsmodels.formula.api import ols
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler
```

[2]:
```python
bike_data = pd.read_csv('Yulu_bike_sharing.csv')
```

# 4 Data analysis like checking the structure & characteristics of the dataset

[3]:
```python
bike_data.dtypes
```

[3]:
```
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
```

[4]:
```python
dt = ['season', 'holiday', 'workingday', 'weather']
for i in dt:
    bike_data[i] = bike_data[i].astype('category')
```

[5]:
```python
bike_data['datetime'] = pd.to_datetime(bike_data['datetime'])
```

[6]:
```python
bike_data.dtypes
```

[6]:
```
datetime      datetime64[ns]
season              category
holiday             category
workingday          category
weather             category
temp                 float64
atemp                float64
humidity               int64
windspeed            float64
casual                 int64
registered             int64
count                  int64
dtype: object
```

[7]:
```
bike_data.rename(columns={'count':'total_riders'},inplace=True)
```

[8]:
```
bike_data['year'] = bike_data['datetime'].dt.year
bike_data['month'] = bike_data['datetime'].dt.month
bike_data['hour'] = bike_data['datetime'].dt.hour
```

[9]:
```
bike_data['month'] = bike_data['month'].replace({1: 'January',
                                                  2: 'February',
                                                  3: 'March',
                                                  4: 'April',
                                                  5: 'May',
                                                  6: 'June',
                                                  7: 'July',
                                                  8: 'August',
                                                  9: 'September',
                                                  10: 'October',
                                                  11: 'November',
                                                  12: 'December'})

bike_data['season'] = bike_data['season'].replace({1: 'spring',
                                                    2: 'summer',
                                                    3: 'fall',
                                                    4: 'winter'})

bike_data['weather'] = bike_data['weather'].replace({1: 'Clear',
                                                      2: 'Misty_cloudy',
                                                      3: 'Rain' ,
                                                      4: 'Heavy_rain'})

bike_data['holiday'] = bike_data['holiday'].replace({1: 'Holiday', 0:␣
 ↪'Non-Holiday'})
bike_data['workingday'] = bike_data['workingday'].replace({1: 'Working day', 0:␣
 ↪'Non-Working day'})
```

```python
bike_data.groupby('weather').count().reset_index()
```

[9]:

```
        weather  datetime  season  holiday  workingday  temp  atemp  humidity  \
0         Clear      7192    7192     7192        7192  7192   7192      7192
1  Misty_cloudy      2834    2834     2834        2834  2834   2834      2834
2          Rain       859     859      859         859   859    859       859
3    Heavy_rain         1       1        1           1     1      1         1

   windspeed  casual  registered  total_riders  year  month  hour
0       7192    7192        7192          7192  7192   7192  7192
1       2834    2834        2834          2834  2834   2834  2834
2        859     859         859           859   859    859   859
3          1       1           1             1     1      1     1
```

The final operation groups the data by the weather column and provides a count of rows for each unique weather condition. This is useful to understand how many entries fall under each weather category.

The reset_index() is used to ensure the output is a DataFrame with a standard format.

[10]:

```python
bike_data.head()
```

[10]:

```
             datetime  season      holiday       workingday weather  temp  \
0 2011-01-01 00:00:00  spring  Non-Holiday  Non-Working day   Clear  9.84
1 2011-01-01 01:00:00  spring  Non-Holiday  Non-Working day   Clear  9.02
2 2011-01-01 02:00:00  spring  Non-Holiday  Non-Working day   Clear  9.02
3 2011-01-01 03:00:00  spring  Non-Holiday  Non-Working day   Clear  9.84
4 2011-01-01 04:00:00  spring  Non-Holiday  Non-Working day   Clear  9.84

    atemp  humidity  windspeed  casual  registered  total_riders  year  \
0  14.395        81        0.0       3          13            16  2011
1  13.635        80        0.0       8          32            40  2011
2  13.635        80        0.0       5          27            32  2011
3  14.395        75        0.0       3          10            13  2011
4  14.395        75        0.0       0           1             1  2011

     month  hour
0  January     0
1  January     1
2  January     2
3  January     3
4  January     4
```

## 5 Check for the Null values

[11]:
```
bike_data.isna().sum()
```

[11]:
```
datetime        0
season          0
holiday         0
workingday      0
weather         0
temp            0
atemp           0
humidity        0
windspeed       0
casual          0
registered      0
total_riders    0
year            0
month           0
hour            0
dtype: int64
```

## 6 Checking the unique values for columns

[12]:
```
for i in bike_data.columns:
  print(f'Unique Values in {i} column are :-\n {bike_data[i].unique()}\n\n')
```

```
Unique Values in datetime column are :-
 <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 in season column are :-
 ['spring', 'summer', 'fall', 'winter']
Categories (4, object): ['spring', 'summer', 'fall', 'winter']


Unique Values in holiday column are :-
 ['Non-Holiday', 'Holiday']
```

```
Categories (2, object): ['Non-Holiday', 'Holiday']


Unique Values in workingday column are :-
 ['Non-Working day', 'Working day']
Categories (2, object): ['Non-Working day', 'Working day']


Unique Values in weather column are :-
 ['Clear', 'Misty_cloudy', 'Rain', 'Heavy_rain']
Categories (4, object): ['Clear', 'Misty_cloudy', 'Rain', 'Heavy_rain']


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


Unique Values in windspeed column are :-
 [ 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]


Unique Values in casual column are :-
 [  3   8   5   0   2   1  12  26  29  47  35  40  41  15   9   6  11   4
   7  16  20  19  10  13  14  18  17  21  33  23  22  28  48  52  42  24
```

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

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


Unique Values in total_riders column are :-
[ 16  40  32  13   1   2   3   8  14  36  56  84  94 106 110  93  67  35
  37  34  28  39  17   9   6  20  53  70  75  59  74  76  65  30  22  31
   5  64 154  88  44  51  61  77  72 157  52  12   4 179 100  42  57  78
  97  63  83 212 182 112  54  48  11  33 195 115  46  79  71  62  89 190
 169 132  43  19  95 219 122  45  86 172 163  69  23   7 210 134  73  50
  87 187 123  15  25  98 102  55  10  49  82  92  41  38 188  47 178 155
  24  18  27  99 217 130 136  29 128  81  68 139 137 202  60 162 144 158
 117  90 159 101 118 129  26 104  91 113 105  21  80 125 133 197 109 161
 135 116 176 168 108 103 175 147  96 220 127 205 174 121 230  66 114 216
 243 152 199  58 166 170 165 160 140 211 120 145 256 126 223  85 206 124
 255 222 285 146 274 272 185 191 232 327 224 107 119 196 171 214 242 148
 268 201 150 111 167 228 198 204 164 233 257 151 248 235 141 249 194 259
 156 153 244 213 181 221 250 304 241 271 282 225 253 237 299 142 313 310
 207 138 280 173 332 331 149 267 301 312 278 281 184 215 367 349 292 303
 339 143 189 366 386 273 325 356 314 343 333 226 203 177 263 297 288 236
 240 131 452 383 284 291 309 321 193 337 388 300 200 180 209 354 361 306
 277 428 362 286 351 192 411 421 276 264 238 266 371 269 537 518 218 265
 459 186 517 544 365 290 410 396 296 440 533 520 258 450 246 260 344 553
 470 298 347 373 436 378 342 289 340 382 390 358 385 239 374 598 524 384
 425 611 550 434 318 442 401 234 594 527 364 387 491 398 270 279 294 295
 322 456 437 392 231 394 453 308 604 480 283 565 489 487 183 302 547 513
 454 486 467 572 525 379 502 558 564 391 293 247 317 369 420 451 404 341
 251 335 417 363 357 438 579 556 407 336 334 477 539 551 424 346 353 481
 506 432 409 466 326 254 463 380 275 311 315 360 350 252 328 476 227 601
 586 423 330 569 538 370 498 638 607 416 261 355 552 208 468 449 381 377
 397 492 427 461 422 305 375 376 414 447 408 418 457 545 496 368 245 596
 563 443 562 229 316 402 287 372 514 472 511 488 419 595 578 400 348 587
 497 433 475 406 430 324 262 323 412 530 543 413 435 555 523 441 529 532
 585 399 584 559 307 582 571 426 516 465 329 483 600 570 628 531 455 389
 505 359 431 460 590 429 599 338 566 482 568 540 495 345 591 593 446 485
 393 500 473 352 320 479 444 462 405 620 499 625 395 528 319 519 445 512
 471 508 526 509 484 448 515 549 501 612 597 464 644 712 676 734 662 782
 749 623 713 746 651 686 690 679 685 648 560 503 521 554 541 721 801 561
```

```
 573 589 729 618 494 757 800 684 744 759 822 698 490 536 655 643 626 615
 567 617 632 646 692 704 624 656 610 738 671 678 660 658 635 681 616 522
 673 781 775 576 677 748 776 557 743 666 813 504 627 706 641 575 639 769
 680 546 717 710 458 622 705 630 732 770 439 779 659 602 478 733 650 873
 846 474 634 852 868 745 812 669 642 730 672 645 694 493 668 647 702 665
 834 850 790 415 724 869 700 793 723 534 831 613 653 857 719 867 823 403
 693 603 583 542 614 580 811 795 747 581 722 689 849 872 631 649 819 674
 830 814 633 825 629 835 667 755 794 661 772 657 771 777 837 891 652 739
 865 767 741 469 605 858 843 640 737 862 810 577 818 854 682 851 848 897
 832 791 654 856 839 725 863 808 792 696 701 871 968 750 970 877 925 977
 758 884 766 894 715 783 683 842 774 797 886 892 784 687 809 917 901 887
 785 900 761 806 507 948 844 798 827 670 637 619 592 943 838 817 888 890
 788 588 606 608 691 711 663 731 708 609 688 636]


Unique Values in year column are :-
 [2011 2012]


Unique Values in month column are :-
 ['January' 'February' 'March' 'April' 'May' 'June' 'July' 'August'
 'September' 'October' 'November' 'December']


Unique Values in hour column are :-
 [ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23]
```

# 7 Checking the number of unique values for columns

[13]:
```python
for i in bike_data.columns:
  print(f'Number of Unique values in {i} column : {bike_data[i].nunique()}')
```

```
Number of Unique values in datetime column : 10886
Number of Unique values in season column : 4
Number of Unique values in holiday column : 2
Number of Unique values in workingday column : 2
Number of Unique values in weather column : 4
Number of Unique values in temp column : 49
Number of Unique values in atemp column : 60
Number of Unique values in humidity column : 89
Number of Unique values in windspeed column : 28
Number of Unique values in casual column : 309
Number of Unique values in registered column : 731
Number of Unique values in total_riders column : 822
Number of Unique values in year column : 2
```

```
Number of Unique values in month column : 12
Number of Unique values in hour column : 24
```

### 7.1 Insights

There's a total of 10,886 entries with 12 different pieces of information for each entry.

The data seems to be in good shape, with no missing values and no duplicates.

There are two main data types:

- Numbers: This includes things like temperature, humidity, windspeed, and the number of casual and registered riders.
- Categories: This includes things like the season, holiday status, working day indicator, weather conditions, and types of riders.

We made some adjustments to the data to make it easier to analyze: We converted the date and time information into a format that computers can understand better.

We changed some of the category information (season, holiday, working day, and weather) from numbers to text descriptions, since these represent different categories.

## 8 Detect Outliers

[14]:
```
categorical_var = ['datetime', 'season', 'holiday', 'workingday', 'weather']


continuous_var = ['temp', 'atemp', 'humidity', 'windspeed', 'casual',␣
↪'registered', 'total_riders']


arr = {'25th percentile or Q1': 25, '50th percentile or Q2': 50, '75th␣
↪percentile or Q3': 75,
       }
```

[15]:
```
for key, value in arr.items():
  for i in continuous_var:
    print(f'{i} : {key} -> {np.percentile(bike_data[i], value):.2f}')
  print('_'*100, sep = " ")
```

```
temp : 25th percentile or Q1 -> 13.94
atemp : 25th percentile or Q1 -> 16.66
humidity : 25th percentile or Q1 -> 47.00
windspeed : 25th percentile or Q1 -> 7.00
casual : 25th percentile or Q1 -> 4.00
registered : 25th percentile or Q1 -> 36.00
total_riders : 25th percentile or Q1 -> 42.00
________________________________________________________________________________
____________________
temp : 50th percentile or Q2 -> 20.50
```

```
atemp : 50th percentile or Q2 -> 24.24
humidity : 50th percentile or Q2 -> 62.00
windspeed : 50th percentile or Q2 -> 13.00
casual : 50th percentile or Q2 -> 17.00
registered : 50th percentile or Q2 -> 118.00
total_riders : 50th percentile or Q2 -> 145.00
--------------------------------------------------------------------------------
--------------------
temp : 75th percentile or Q3 -> 26.24
atemp : 75th percentile or Q3 -> 31.06
humidity : 75th percentile or Q3 -> 77.00
windspeed : 75th percentile or Q3 -> 17.00
casual : 75th percentile or Q3 -> 49.00
registered : 75th percentile or Q3 -> 222.00
total_riders : 75th percentile or Q3 -> 284.00
--------------------------------------------------------------------------------
--------------------
```

[16]:
```python
for i in continuous_var:
  Q1 = np.percentile(bike_data[i], arr['25th percentile or Q1'])
  Q3 = np.percentile(bike_data[i], arr['75th percentile or Q3'])
  IQR = Q3 - Q1

  # Define the outlier thresholds
  lower_threshold = Q1 - 1.5 * IQR
  upper_threshold = Q3 + 1.5 * IQR

  # Find the outliers for the iiable
  outliers = bike_data[(bike_data[i] < lower_threshold) | (bike_data[i] >␣
↪upper_threshold)]

  # Calculate the percentage of outliers
  outlier_percentage = round(len(outliers) / len(bike_data[i]) * 100, 2 )

  # Output the percentage of outliers
  print(f"IQR for {i}: {IQR:.2f}")
  print(f"Outlier above this Q3 {i} : {upper_threshold:.2f}")
  print(f"Percentage of outliers for {i}: {outlier_percentage:.2f}% ")
  print('_'*100, sep = " ")
```

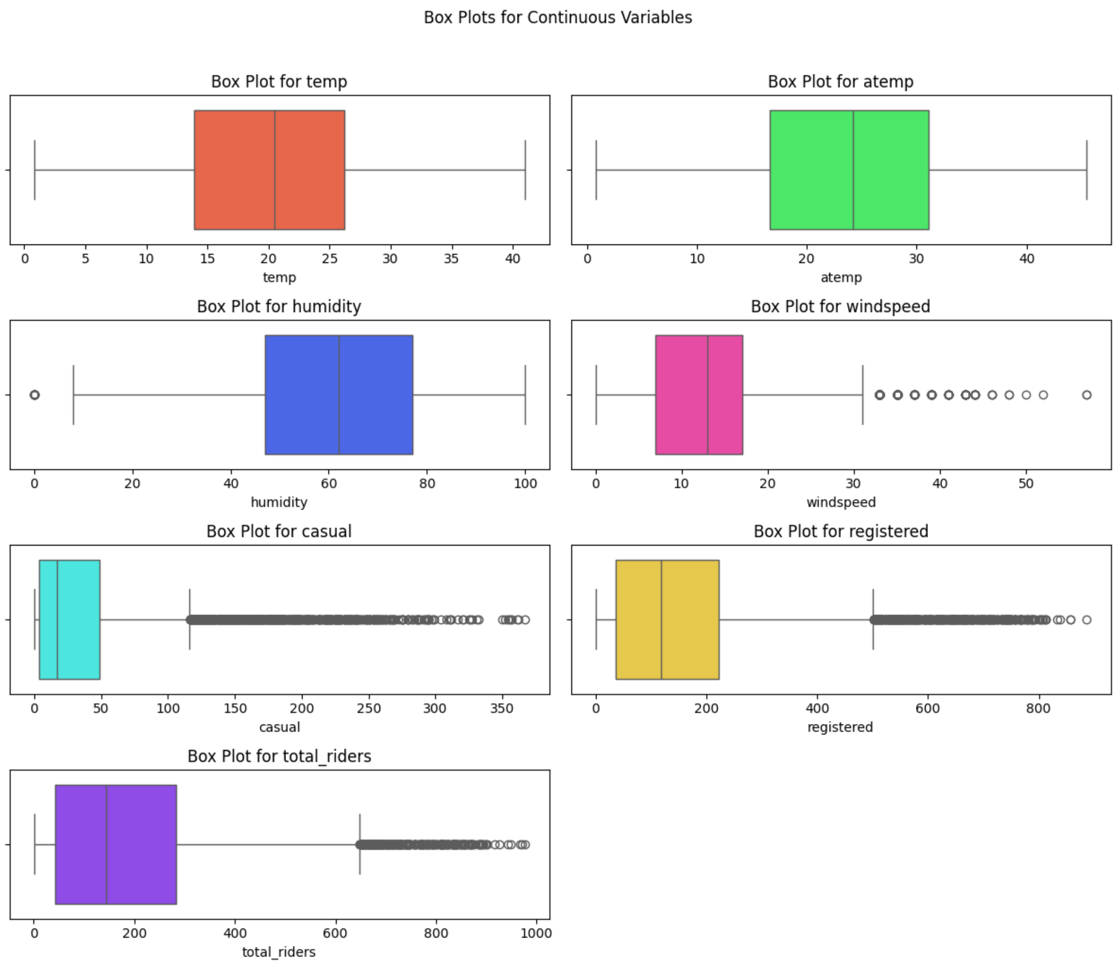
```
IQR for temp: 12.30
Outlier above this Q3 temp : 44.69
Percentage of outliers for temp: 0.00%
--------------------------------------------------------------------------------
--------------------
IQR for atemp: 14.39
Outlier above this Q3 atemp : 52.65
Percentage of outliers for atemp: 0.00%
```

```
--------------------------------------------------------------------------------
--------------------
IQR for humidity: 30.00
Outlier above this Q3 humidity : 122.00
Percentage of outliers for humidity: 0.20%
--------------------------------------------------------------------------------
--------------------
IQR for windspeed: 10.00
Outlier above this Q3 windspeed : 31.99
Percentage of outliers for windspeed: 2.09%
--------------------------------------------------------------------------------
--------------------
IQR for casual: 45.00
Outlier above this Q3 casual : 116.50
Percentage of outliers for casual: 6.88%
--------------------------------------------------------------------------------
--------------------
IQR for registered: 186.00
Outlier above this Q3 registered : 501.00
Percentage of outliers for registered: 3.89%
--------------------------------------------------------------------------------
--------------------
IQR for total_riders: 242.00
Outlier above this Q3 total_riders : 647.00
Percentage of outliers for total_riders: 2.76%
--------------------------------------------------------------------------------
--------------------
```

[17]:

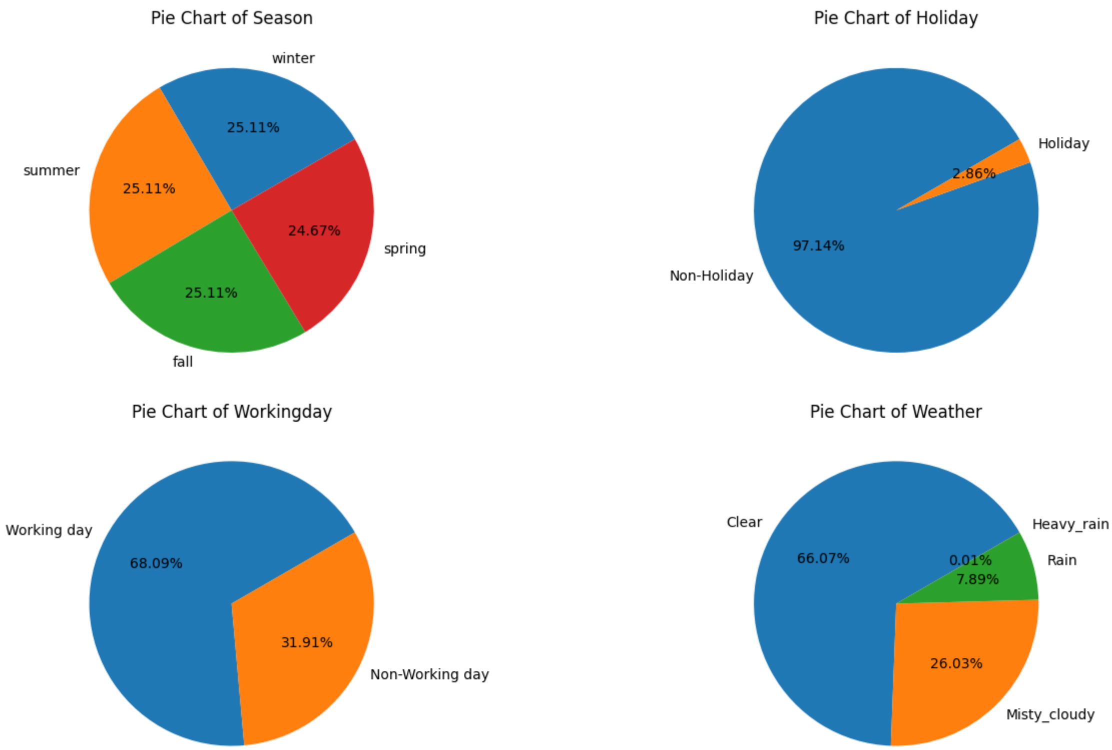
```
plt.figure(figsize=(12, 10))
fig, axes = plt.subplots(4, 2, figsize=(12, 10))
fig.suptitle("Box Plots for Continuous Variables", y=1.02)
variables = ['temp', 'atemp', 'humidity', 'windspeed', 'casual', 'registered',
 ↪'total_riders']
colors = ["#FF5733", "#33FF57", "#3357FF", "#FF33A6", "#33FFF5", "#FFD733",
 ↪"#8D33FF"]
axes = axes.flatten()
for i, var in enumerate(variables):
    sns.boxplot(ax=axes[i], x=bike_data[var], color=colors[i])
    axes[i].set_title(f'Box Plot for {var}')

for j in range(len(variables), len(axes)): #To remove Unused plots
   fig.delaxes(axes[j])

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

```
<Figure size 1200x1000 with 0 Axes>
```




# 9 Analyze the distribution

[18]:
```
categorical_columns = ['season', 'holiday', 'workingday', 'weather']

fig, axes = plt.subplots(2, 2, figsize=(15, 8))

for i, column in enumerate(categorical_columns):
    row = i // 2
    col = i % 2
    order = bike_data[column].value_counts()
    axes[row, col].pie(order, labels=order.index, autopct='%1.2f%%',␣
↪startangle=30)
    axes[row, col].set_title(f'Pie Chart of {column.capitalize()}')

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




## 9.1 Insights and Recommendations

Seasons:

There's a balanced mix of data for all four seasons (spring, summer, fall, winter) in your dataset. This means you can target promotions throughout the year.

Recommendation:

Design special offers or promotions for each season to keep riders engaged year-round.

Holidays:

We noticed there are fewer rentals on holidays compared to regular days.

Recommendation:
Run targeted campaigns or promotions specifically during holidays to boost rentals on those days.

Working Days:

The data shows more rentals happening on weekdays compared to weekends.

Recommendation:
Offer incentives or discounts for riders who use bikes for commuting during work hours. This can encourage weekday rentals.

Weather:

Most rentals occur during clear or slightly cloudy weather. There are fewer rentals on days with mist, rain, thunderstorm, or snow.

Recommendation:

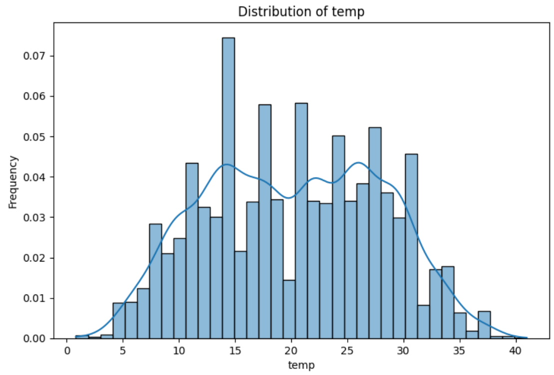
Consider offering rain gear rentals on rainy days to keep riders going even in bad weather.

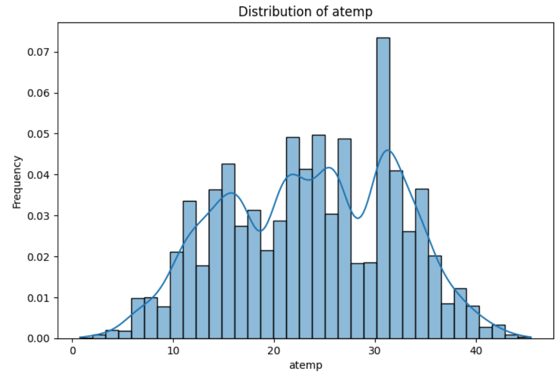
Promote bike rentals on clear days to capitalize on the favorable weather conditions.

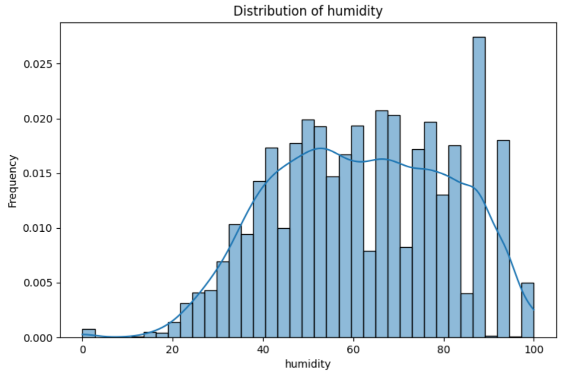
# 10 Univariate Analysis:

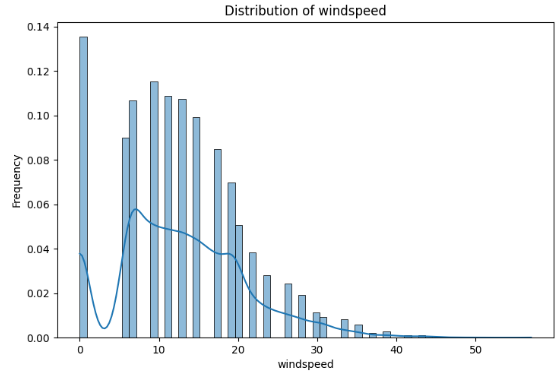
## 10.1 For each Numerical features

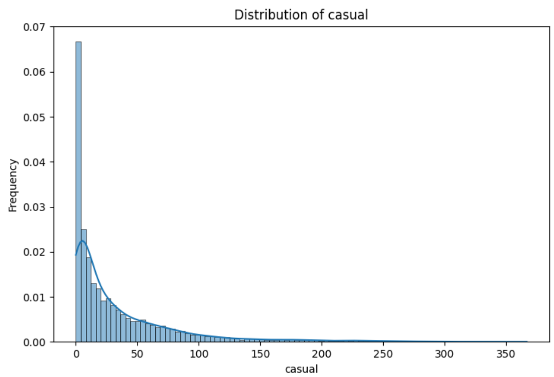
[19]:
```
fig, ax = plt.subplots(4, 2, figsize=(15, 20))

# Plot histograms for continuous variables
for i, feature in enumerate(continuous_var):
    row, col = divmod(i, 2)
    sns.histplot(bike_data[feature], kde=True, ax=ax[row, col],
 ↪line_kws={'color': 'red'}, stat='density',)
    ax[row, col].set_title('Distribution of ' + feature)
    ax[row, col].set_xlabel(feature)
    ax[row, col].set_ylabel('Frequency')

plt.tight_layout()
fig.delaxes(ax[-1, -1])

plt.show()
```

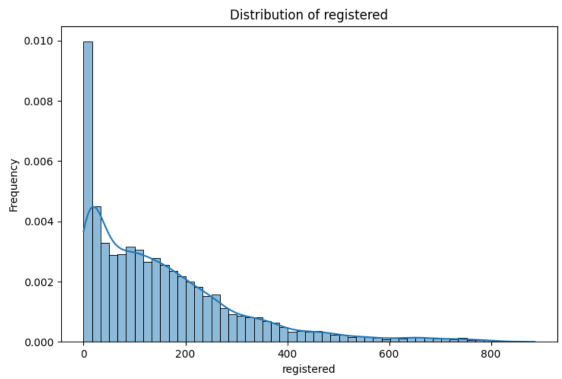

Distribution of temp
Frequency
temp

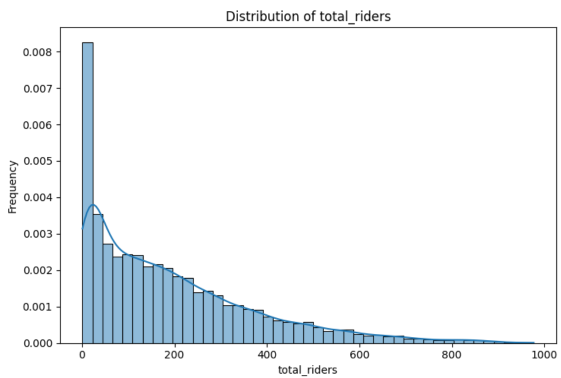

Distribution of atemp
Frequency
atemp


Distribution of humidity
Frequency
humidity


Distribution of windspeed
Frequency
windspeed


Distribution of casual
Frequency
casual


Distribution of registered
Frequency
registered


Distribution of total_riders
Frequency
total_riders

### 10.2 Insights and Recommendations

Temperature:

The temperature data is nicely balanced, with most rentals happening at a comfortable range.

Recommendation:

Promote bike rentals when the weather is pleasant! This can encourage more people to get outside and enjoy a ride.

Humidity:

We noticed that high humidity is more common than low humidity.

Recommendation:

Help riders beat the heat! Promote early morning or evening rides when it's cooler.

Wind Speed:

Most days have lower wind speeds, but there can be occasional windy days.

Recommendation:

Focus on rider safety during windy conditions. Provide information on wind-resistant routes or sheltered areas for riders to enjoy their bike rentals even on windy days.

Number of Riders:

We found that some days have many more rentals than others.

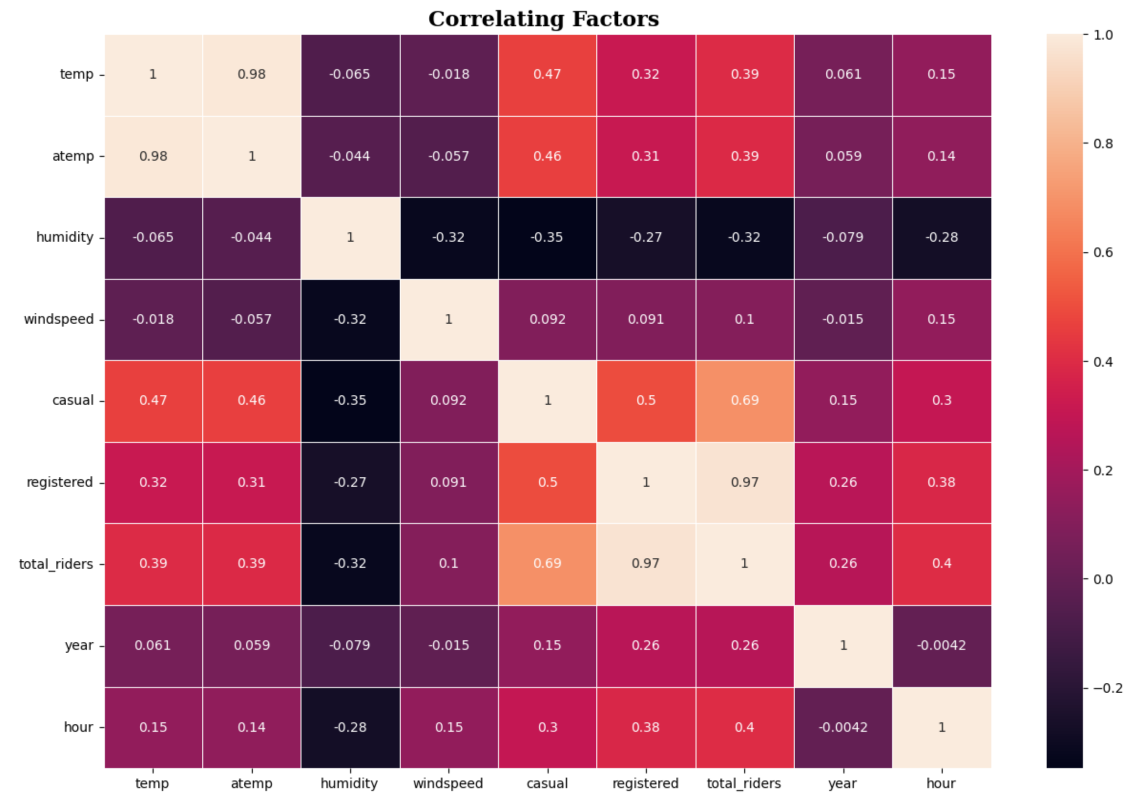
Recommendation:

Attract more riders on slower days! Develop targeted promotions or discounts to encourage rentals during off-peak times.

Tailor your marketing and services based on the typical weather conditions for each season to get the most riders throughout the year!

## 11 Relationship between the Dependent and Independent Variables

[20]:
```
corr_df =  bike_data.corr(numeric_only=True)
plt.figure(figsize=(15,10))
sns.heatmap(bike_data.corr(numeric_only=True), annot=True, linewidth=.5)
plt.yticks(rotation=0)
plt.title('Correlating Factors␣
 ↪',fontfamily='serif',fontweight='bold',fontsize=16)
plt.show()
```




## 12 Insights and Recommendations

Temperature Matters:

Warmer temperatures (including both actual temperature and how it feels) are linked to more bike rentals!

Recommendation:

When the weather forecast predicts comfortable temperatures, promote bike rentals to capitalize on these ideal riding conditions.

Humidity Has an Impact:

We noticed that drier days (lower humidity) tend to have more bike rentals.

Recommendation:

Consider offering promotions or discounts on days with higher humidity to encourage rentals during these times. This can help offset the potential decrease in ridership due to the weather.

Wind Speed is a Factor:

Strong winds can discourage some riders, with slightly fewer rentals happening on windy days.

Recommendation:

Provide wind-resistant bikes or promote routes that are sheltered from strong winds. This can help maintain ridership even on windy days.

More Users, More Rentals:

The data shows a clear link - the more users you have (both casual and registered riders), the higher the total number of rentals.

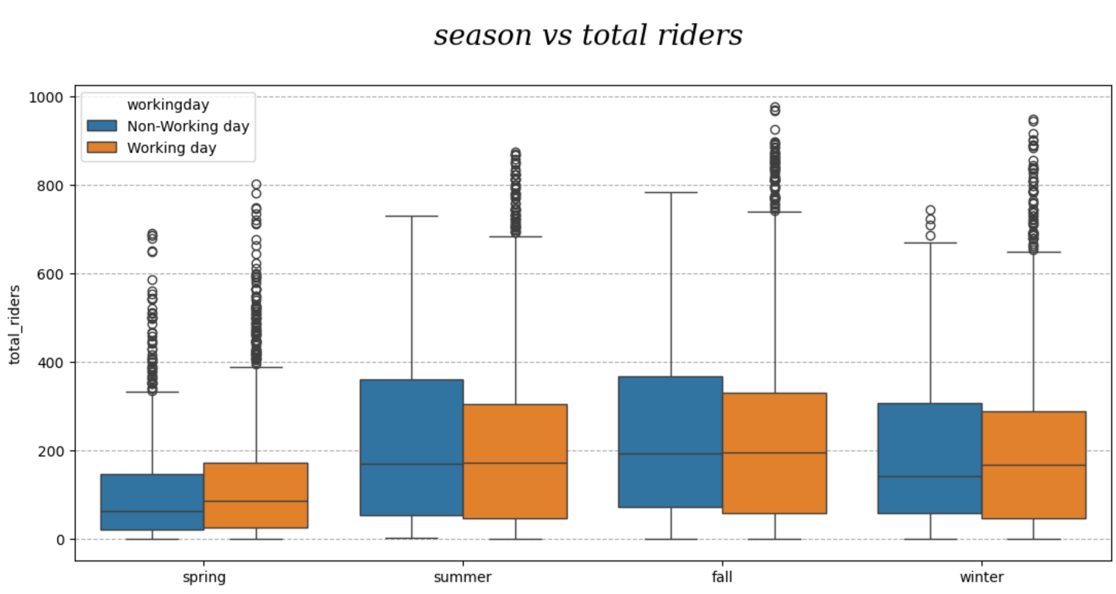
Recommendation:

Increase your marketing efforts to attract new casual riders and incentivize existing registered users. This can significantly boost your overall rentals.

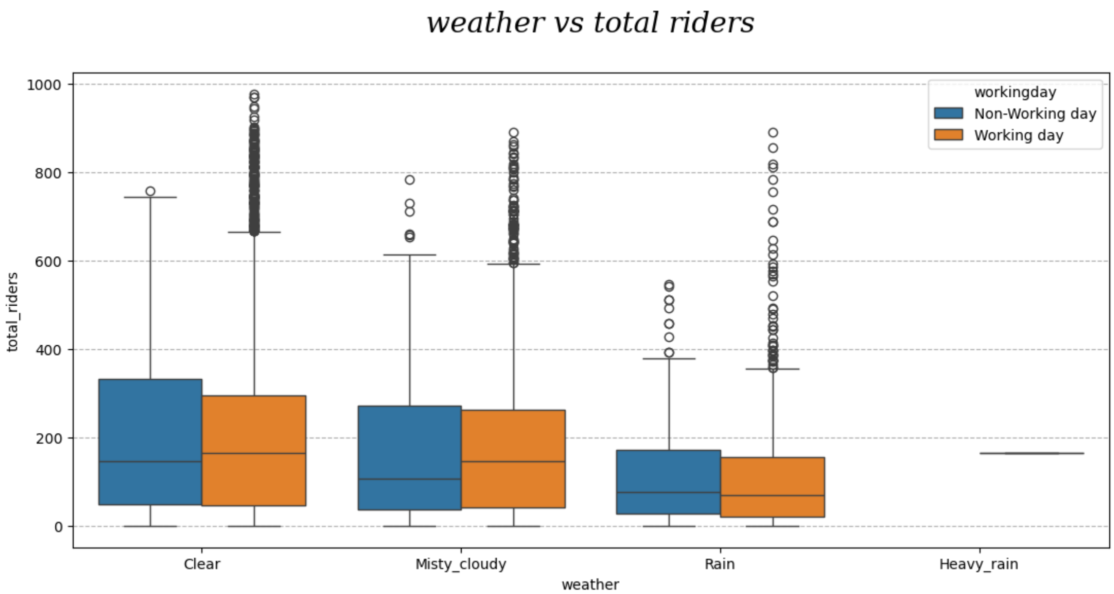
Consider loyalty programs or special offers for frequent riders. This can encourage repeat business and help grow your registered user base.

## 13 Bivariate Analysis

[21]:
```
plt.figure(figsize = (13, 6))
plt.title(f'season vs total riders \n',
          fontdict = {'size' : 20,
                      'style' : 'oblique',
                      'family' : 'serif'})
sns.boxplot(data = bike_data, x = 'season', y = 'total_riders', hue =␣
↪'workingday')
plt.grid(axis = 'y', linestyle = '--')
plt.show()
```



[22]:
```
plt.figure(figsize = (13, 6))
plt.title(f'weather vs total riders\n',
        fontdict = {'size' : 20,
                    'style' : 'oblique',
                    'family' : 'serif'})
sns.boxplot(data = bike_data, x = 'weather', y = 'total_riders', hue =␣
 ↪'workingday')
plt.grid(axis = 'y', linestyle = '--')
plt.show()
```




# 14 Hypothesis Testing:

## 14.1 Is there any effect of Working Day on the number of electric cycles rented ?

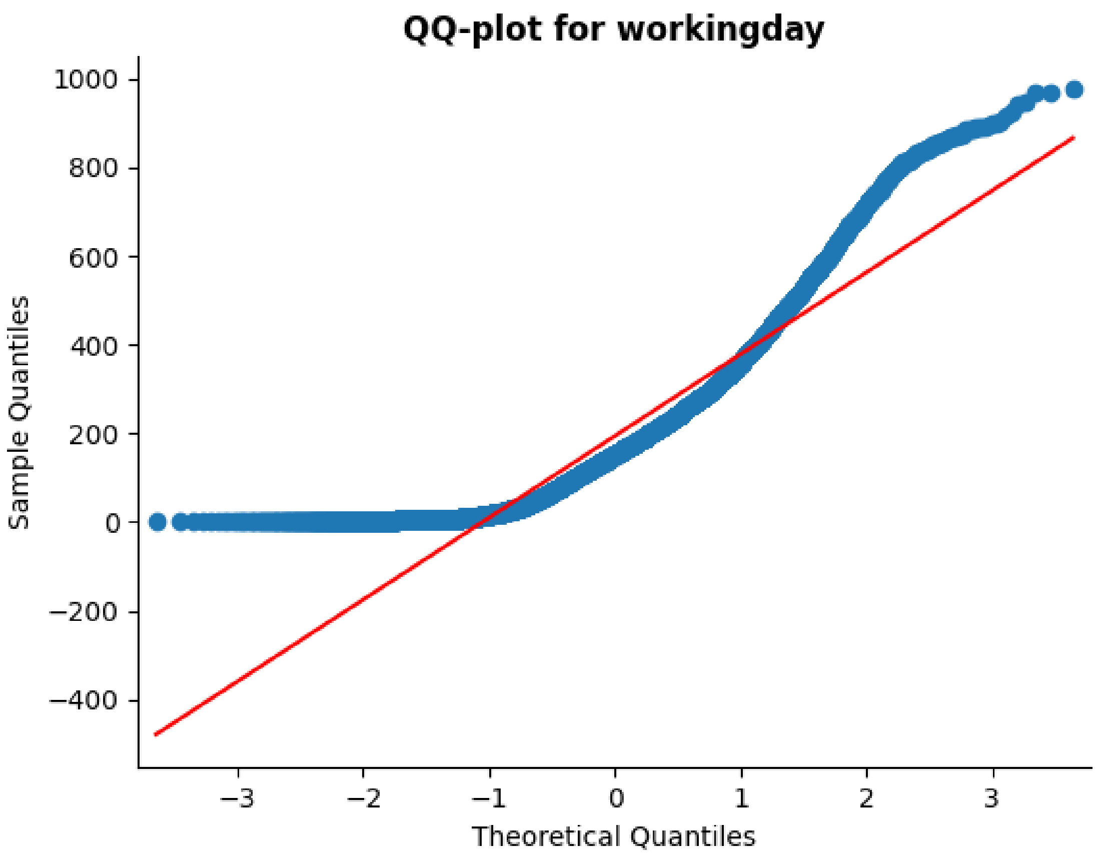
1. Set up Null and Alternate Hypothesis

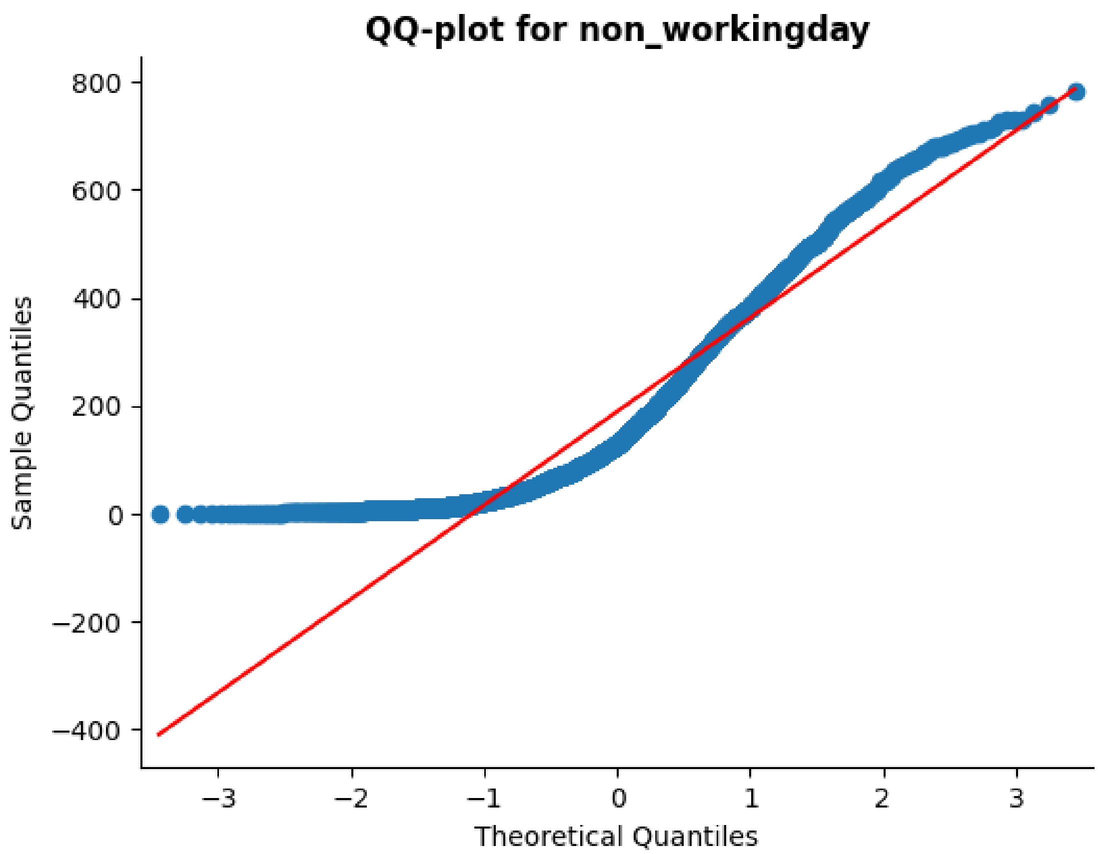
- Null Hypothesis ( H0 ) - No.of bikes rented on working days and non working days are same.
- Alternate Hypothesis ( HA ) - No.of bikes rented on working days and non working days are different.

2. Choose the distribution (Gaussian, Binomial, etc), and hence the test statistic.
3. Select the Left vs Right vs Two-Tailed test, as per the hypothesis
4. Compute the P-Value
5. Compare the P-Value to the Significance Level ( ) and Fail to reject/reject the Null Hypothesis accordingly.

[23]:
```
workingday = bike_data[bike_data['workingday']== 'Working day']['total_riders']
non_workingday = bike_data[bike_data['workingday']== 'Non-Working␣
↪day']['total_riders']
```

[24]:
```
sm.qqplot(workingday,line='s')
plt.title('QQ-plot for workingday',fontsize=12,fontweight="bold")
sns.despine()

sm.qqplot(non_workingday,line='s')
plt.title('QQ-plot for non_workingday',fontsize=12,fontweight="bold")
sns.despine()

plt.show()
```






### 14.2 Shapiro test for workingday

- Null Hypothesis – H0 – Data is Gaussian
- Alternate Hypothesis – HA – Data is not Gaussian

[25]:
```
shapiro_stat , p_val = shapiro(workingday)
print(f"shapiro_stat : {shapiro_stat} , p_value : {p_val}")

if p_val <= 0.05:
    print('Data does not follow normal distribution \n')
else:
    print('Data follows a normal distribution \n')
```

```
shapiro_stat : 0.8702582120895386 , p_value : 0.0
Data does not follow normal distribution


/usr/local/lib/python3.10/dist-packages/scipy/stats/_morestats.py:1882:
UserWarning: p-value may not be accurate for N > 5000.
  warnings.warn("p-value may not be accurate for N > 5000.")
```

[26]:
```python
shapiro_stat , p_val = shapiro(non_workingday)
print(f"shapiro_stat : {shapiro_stat} , p_value : {p_val}")

if p_val <= 0.05:
    print('Data does not follow normal distribution')
else:
    print('Data follows a normal distribution')
```

```
shapiro_stat : 0.8852126598358154 , p_value : 4.203895392974451e-45
Data does not follow normal distribution
```

### 14.3 Levene Test

- Null Hypothesis(Ho) - Data has similar variance
- Alternate Hypothesis(HA) - Data has different variance

[27]:
```python
levene_stat, p_value = levene(workingday,non_workingday)

print('Levene_stat : ', levene_stat)
print('p-value : ', p_value)

if p_value <= 0.05:
    print('The samples has different variance')
else:
    print('The samples have has similar variance')
```

```
Levene_stat :  0.004972848886504472
p-value :  0.9437823280916695
The samples have has similar variance
```

### 14.4 Ttest for Independent Variables

[28]:
```python
test_stat, p_value = ttest_ind(workingday,non_workingday)

print(f'ttest_stat : ',test_stat)
print('P-value :',p_value)

if p_value < 0.05:
    print("Reject Null Hypothesis")
    print('No.of bikes rented is not same for working and non-working days')
else:
    print("Failed to Reject Null Hypothesis")
    print('No.of bikes rented is same for working and non-working days')
```

```
ttest_stat :  1.2096277376026694
P-value : 0.22644804226361348
Failed to Reject Null Hypothesis
No.of bikes rented is same for working and non-working days
```

## 14.5 Insights and Recommendations

Our analysis shows that the average number of bikes rented is similar on both working days and non-working days.

Develop targeted marketing campaigns that appeal to a wide range of customers, no matter their work schedule. Consider offering promotions or discounts that cater to different needs and preferences.

Ensure your bike rental process is smooth and efficient on all days of the week. This includes having enough bikes available, regular maintenance, and excellent customer service. A consistent and positive experience keeps customers coming back.

Explore potential collaborations with local businesses, events, or organizations. This can help attract new customers and increase bike rentals on all days of the week. Imagine partnering with a fitness center for weekday morning rides or a festival for weekend rentals.

## 14.6 Checking if number of bikes rented is same or different in different weather

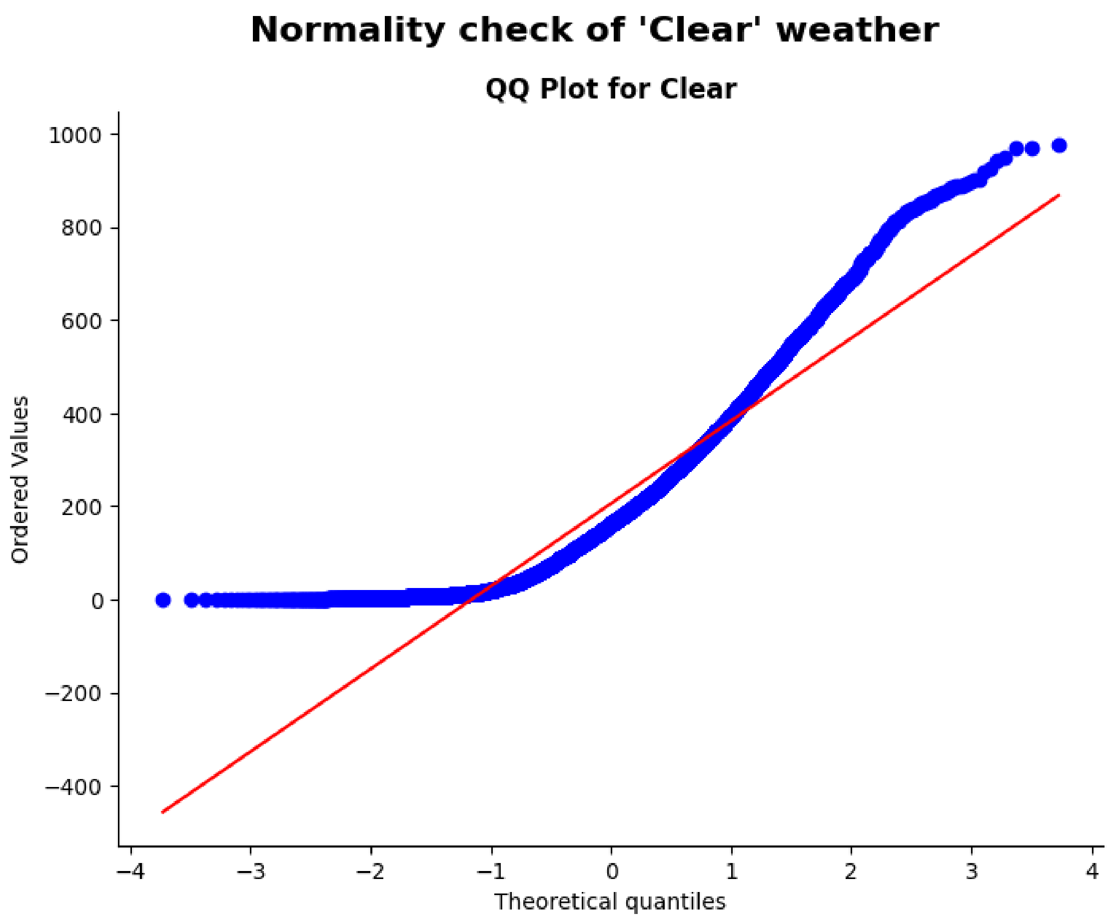
1. Set up Null and Alternate Hypothesis

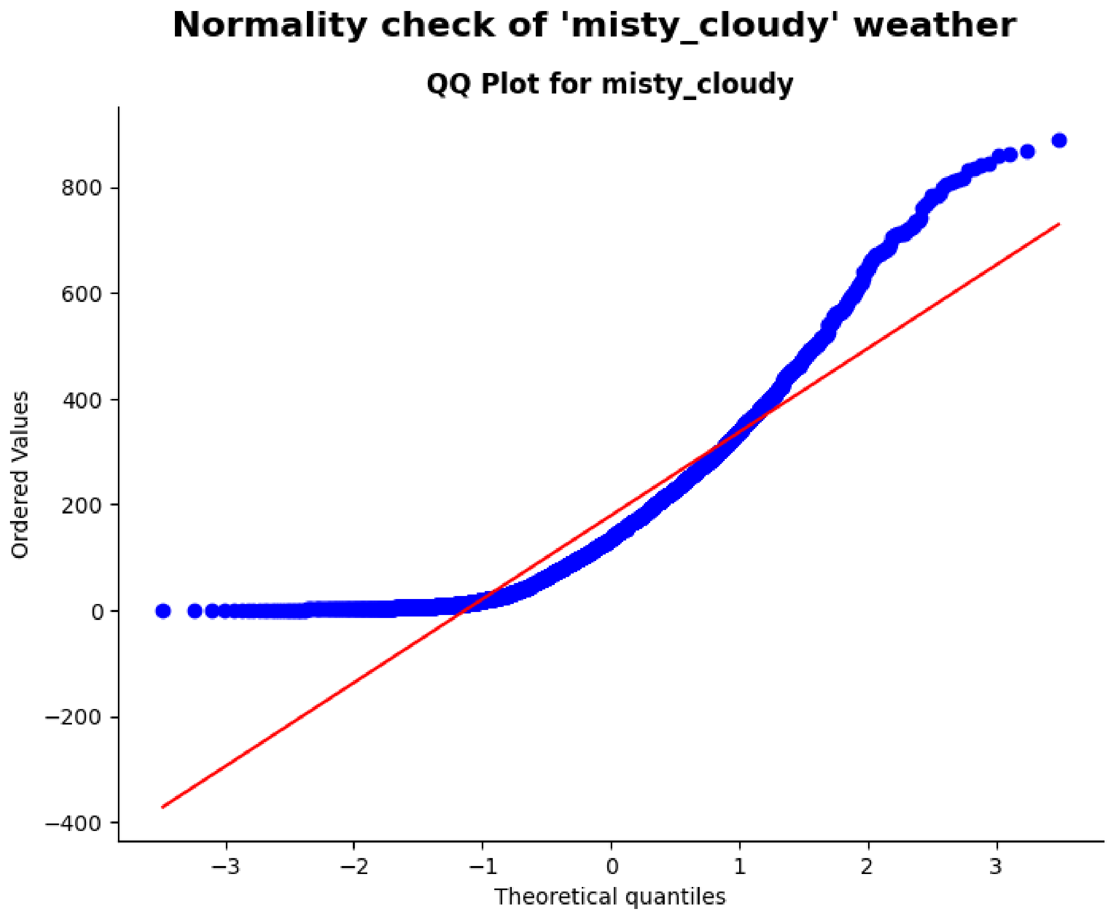
- Null Hypothesis ( H0 ) -The mean of bikes rented is same for across weather conditions.
- Alternate Hypothesis ( HA ) - The mean number of bikes rented is different across at least two weather conditions.

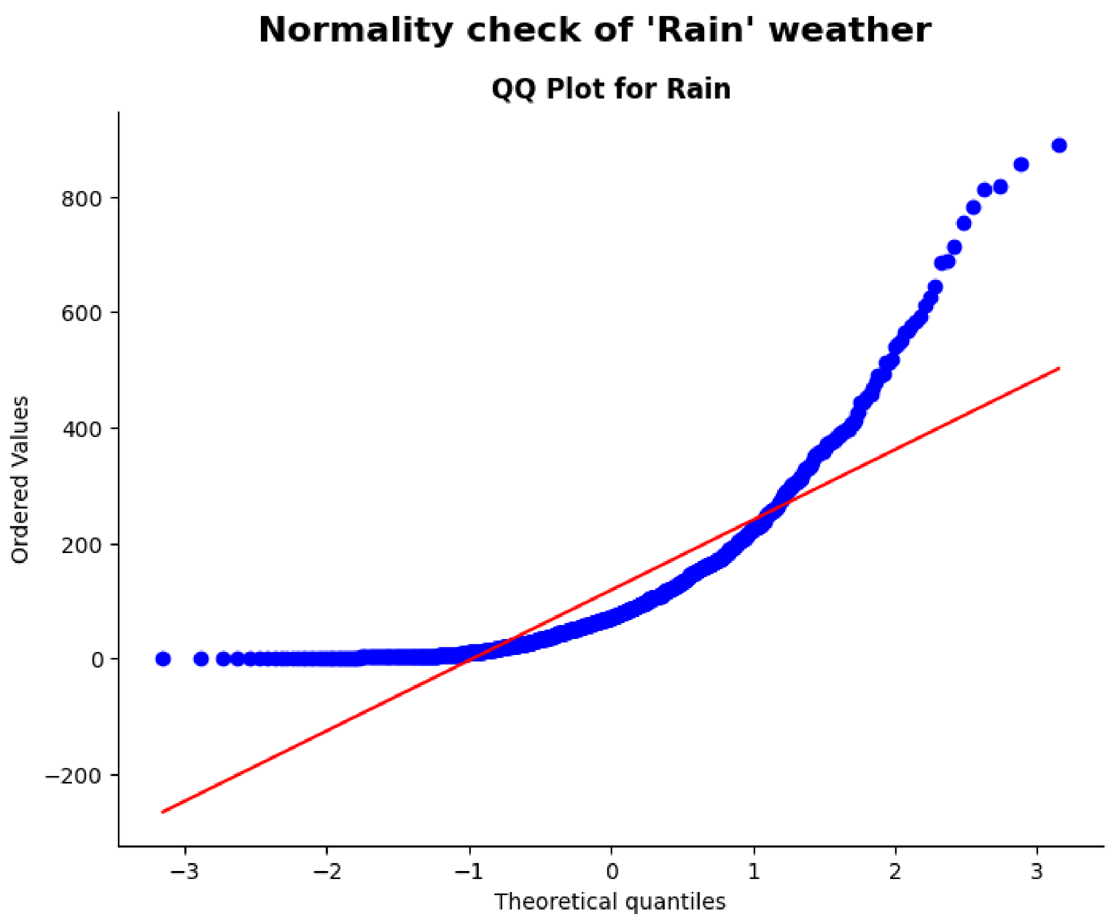
2. Choose the distribution (Gaussian, Binomial, etc), and hence the test statistic.
3. Select the Left vs Right vs Two-Tailed test, as per the hypothesis
4. Compute the P-Value
5. Compare the P-Value to the Significance Level ( ) and Fail to reject/reject the Null Hypothesis accordingly.

[29]:
```
Clear = bike_data[bike_data['weather']=='Clear']['total_riders']
misty_cloudy = bike_data[bike_data['weather']=='Misty_cloudy']['total_riders']
Rain = bike_data[bike_data['weather']=='Rain']['total_riders']
Heavy_rain = bike_data[bike_data['weather']=='Heavy rain']['total_riders']
```

[30]:
```
weather_cols = {'Clear': Clear, 'misty_cloudy': misty_cloudy, 'Rain': Rain}

for col_name, data in weather_cols.items():
    plt.figure(figsize=(8, 6))
    plt.suptitle(f'Normality check of \'{col_name}\' weather', fontsize=16,␣
 ↪fontweight="bold")

    probplot(data, dist='norm', plot=plt)
    plt.title(f'QQ Plot for {col_name}', fontsize=12, fontweight="bold")

    sns.despine()

    plt.show()
```


Normality check of 'Clear' weather
QQ Plot for Clear
Ordered Values
Theoretical quantiles
1000
800
600
400
200
0
−200
−400
−4
−3
−2
−1
0
1
2
3
4


Normality check of 'misty_cloudy' weather
QQ Plot for misty_cloudy
Ordered Values
Theoretical quantiles
800
600
400
200
0
−200
−400
−3
−2
−1
0
1
2
3




### 14.7 Shapiro-Wilk Test:

[31]:

```
weather_cols ={'Clear':Clear, 'misty_cloudy':misty_cloudy, 'Rain': Rain}

for col_name,data in weather_cols.items():
    shapiro_stat , p_val = shapiro(data)
    print(f"shapiro_stat : {shapiro_stat} , p_value : {p_val}")

    if p_val <= 0.05:
        print(f'Data is not Gaussian distribution \n')
    else:
        print(f'Data is Gaussian distribution \n')
    print('-'*125)
```

```
shapiro_stat : 0.8909230828285217 , p_value : 0.0
Data is not Gaussian distribution

--------------------------------------------------------------------------------
```

```
---------------------------------------------
shapiro_stat : 0.8767687082290649 , p_value : 9.781063280987223e-43
Data is not Gaussian distribution

--------------------------------------------------------------------------------
---------------------------------------------
shapiro_stat : 0.7674332857131958 , p_value : 3.876090133422781e-33
Data is not Gaussian distribution

--------------------------------------------------------------------------------
---------------------------------------------
```

### 14.8 Levene Test

Null Hypothesis(H0) - Data has similar variance

Alternate Hypothesis(HA) - Data has different variance

[32]:
```python
levene_stat, p_value = levene(Clear,misty_cloudy,Rain)

print('Levene_stat : ', levene_stat)
print('p-value : ', p_value)

if p_value <= 0.05:
    print('The samples has different variance')
else:
    print('The samples has similar variance')
```

```
Levene_stat :  81.67574924435011
p-value :  6.198278710731511e-36
The samples has different variance
```

The samples are not normally distributed and do not have the same variance, f_oneway test (ANOVA Test) cannot be performed here, we can perform its non parametric equivalent test i.e., Kruskal test for independent samples.

### 14.9 Kruskal Test

[33]:
```python
alpha = 0.05
test_stat, p_value = kruskal(Clear,misty_cloudy,Rain)
print('Test Statistic =', test_stat)
print('p value =', p_value)

if p_value <= alpha:
    print('Reject Null Hypothesis')
    print("The median of bikes rented is different across at weather␣
 ↪conditions")
else:
    print('Failed to reject Null Hypothesis')
```

```
    print("The median of bikes rented is same for across weather conditions.")
```

```
Test Statistic = 204.95566833068537
p value = 3.122066178659941e-45
Reject Null Hypothesis
The median of bikes rented is different across at weather conditions
```

The p-value for the kruskal test on weather is extremely low (close to 0), which means that there are statistically significant differences in the number of cycles rented based on different weather conditions.

From both the Krukal-walis test & ANOVA test , we can confirm that The mean number of E-bikes rented differs across various weather conditions.

### 14.10 Checking if number of bikes rented is similar or different in different Seasons

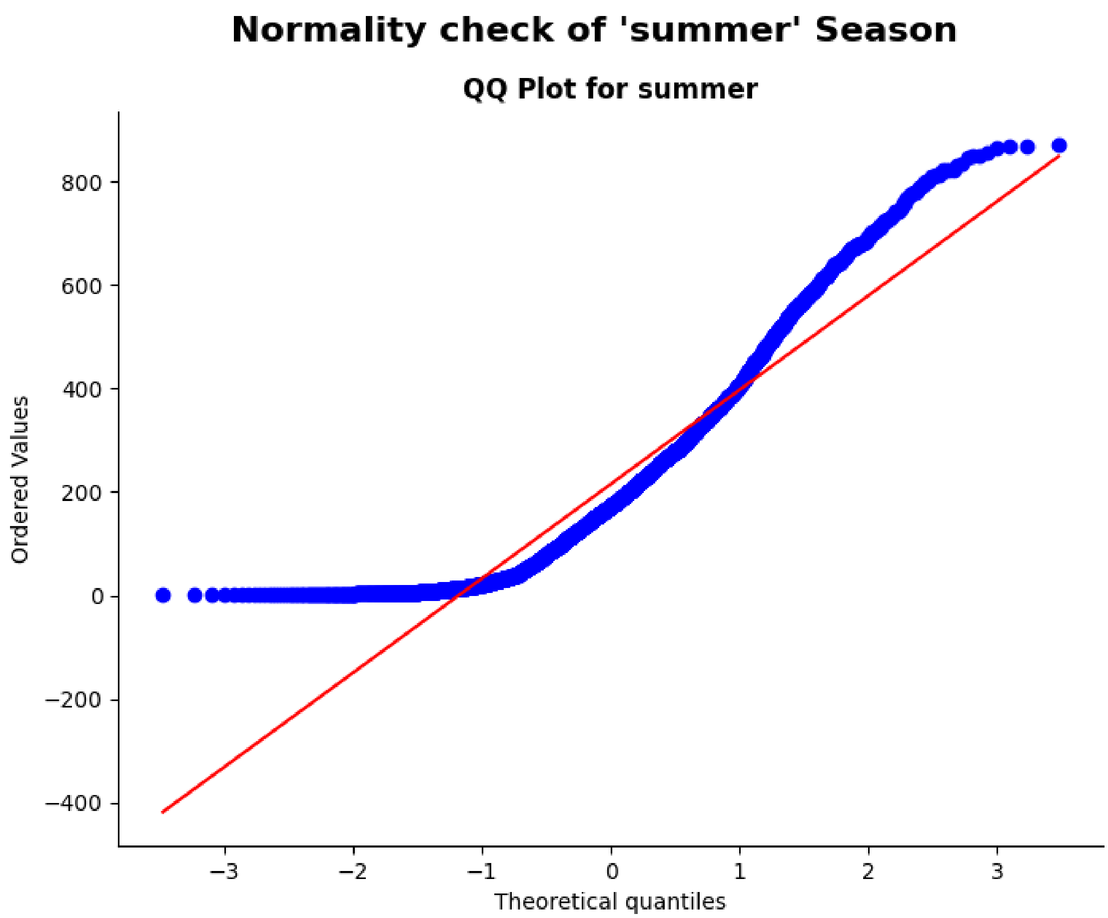
1. Set up Null and Alternate Hypothesis

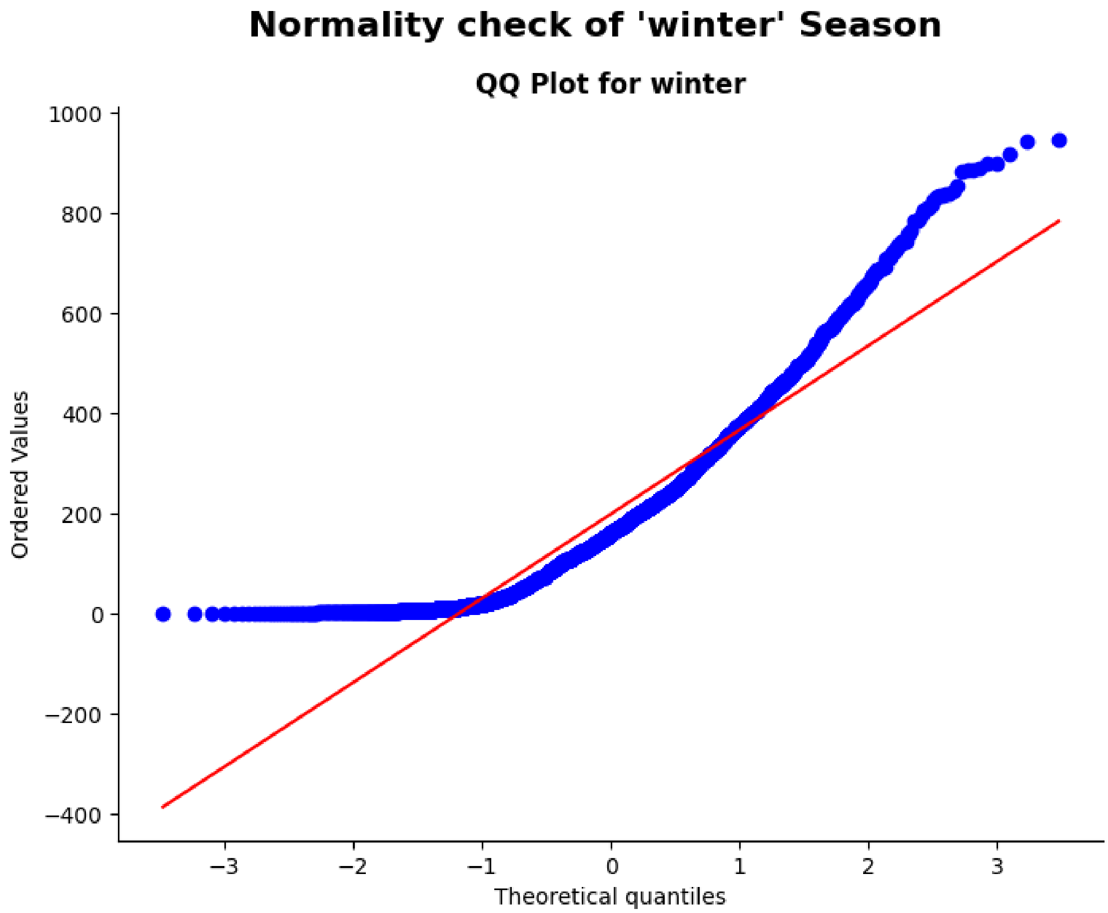
- Null Hypothesis ( H0 ) - The mean of bikes rented is same for across various Seasons.
- Alternate Hypothesis ( HA ) - The mean number of bikes rented is different for across various seasons.

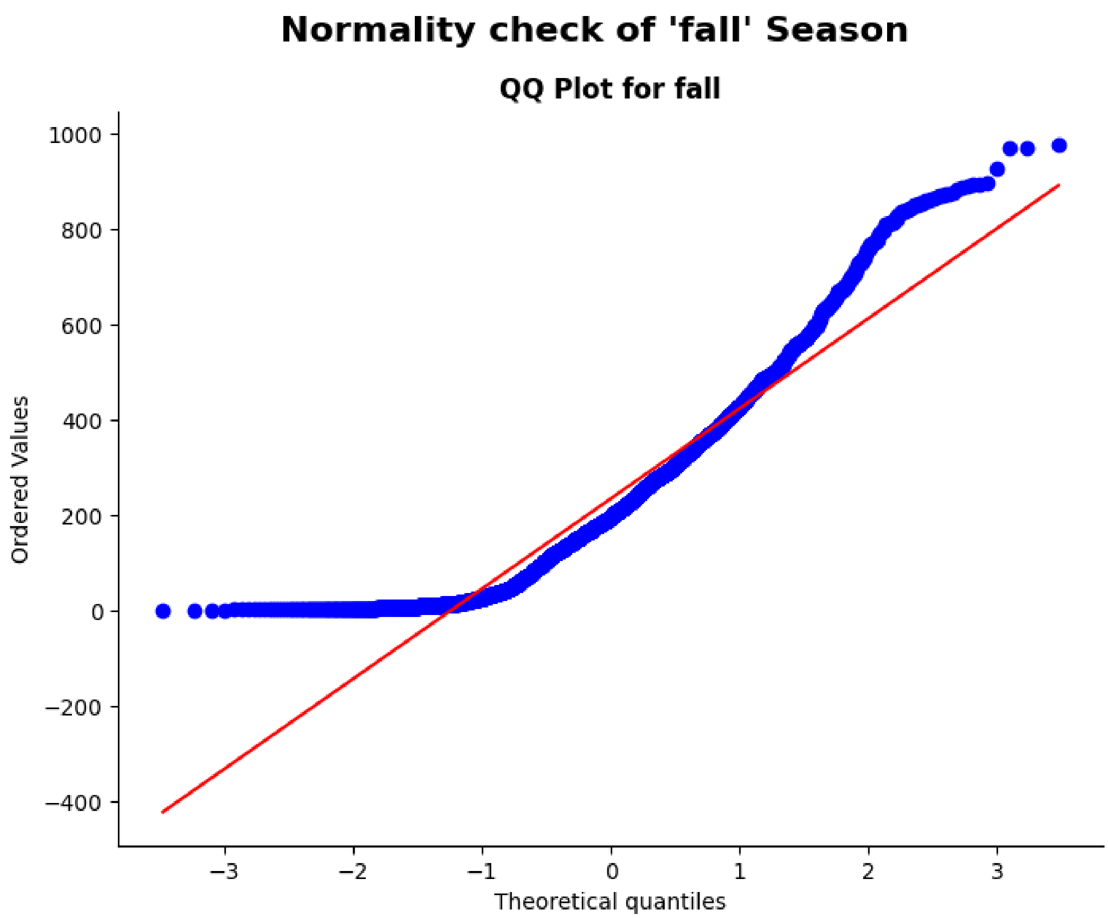
2. Choose the distribution (Gaussian, Binomial, etc), and hence the test statistic.
3. Select the Left vs Right vs Two-Tailed test, as per the hypothesis
4. Compute the P-Value
5. Compare the P-Value to the Significance Level ( ) and Fail to reject/reject the Null Hypothesis accordingly.

[34]:
```
summer = bike_data[bike_data['season']=='summer']['total_riders']
winter = bike_data[bike_data['season']=='winter']['total_riders']
fall = bike_data[bike_data['season']=='fall']['total_riders']
spring = bike_data[bike_data['season']=='spring']['total_riders']
```

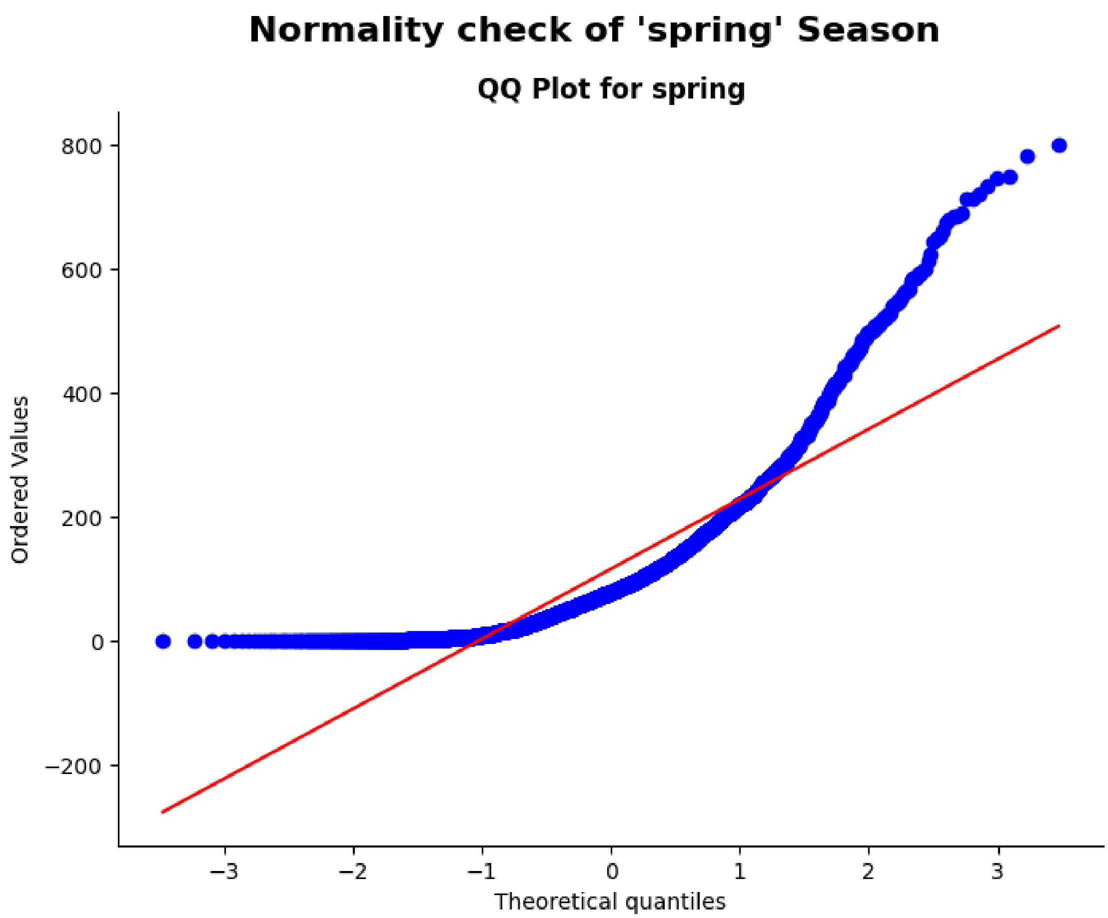
[35]:
```
season_cols = {'summer': summer, 'winter': winter, 'fall': fall, 'spring':␣
↪spring}

for col_name, data in season_cols.items():
    plt.figure(figsize=(8, 6))
    plt.suptitle(f'Normality check of \'{col_name}\' Season', fontsize=16,␣
↪fontweight="bold")

    probplot(data, dist='norm', plot=plt)
    plt.title(f'QQ Plot for {col_name}', fontsize=12, fontweight="bold")
    sns.despine()

    plt.show()
```


Normality check of 'summer' Season
QQ Plot for summer
Ordered Values
800
600
400
200
0
−200
−400
−3
−2
−1
0
1
2
3
Theoretical quantiles


Normality check of 'winter' Season
QQ Plot for winter
Ordered Values
Theoretical quantiles
1000
800
600
400
200
0
−200
−400
−3
−2
−1
0
1
2
3



Normality check of 'fall' Season
QQ Plot for fall
1000
800
600
400
200
0
−200
−400
Ordered Values
−3
−2
−1
0
1
2
3
Theoretical quantiles




### 14.11 Shapiro-Wilk Test:

[36]:
```
season_cols = {'summer':summer , 'winter':winter , 'fall':fall, 'spring':spring}

for col_name,data in season_cols.items():
    shapiro_stat , p_val = shapiro(data)
    print(f"shapiro_stat : {shapiro_stat} , p_value : {p_val}")

    if p_val < 0.05:
        print(f'Data {col_name} is not Gaussian distribution')
        print()
    else:
        print(f'Data {col_name} is Gaussian distribution')
        print()
    print('-'*125)
```

```
shapiro_stat : 0.900481641292572 , p_value : 6.039093315091269e-39
Data summer is not Gaussian distribution
```

```
--------------------------------------------------------------------------------
---------------------------------------------
shapiro_stat : 0.8954644799232483 , p_value : 1.1301682309549298e-39
Data winter is not Gaussian distribution

--------------------------------------------------------------------------------
---------------------------------------------
shapiro_stat : 0.9148160815238953 , p_value : 1.043458045587339e-36
Data fall is not Gaussian distribution

--------------------------------------------------------------------------------
---------------------------------------------
shapiro_stat : 0.8087388873100281 , p_value : 0.0
Data spring is not Gaussian distribution

--------------------------------------------------------------------------------
---------------------------------------------
```

### 14.12 Levene Test

[37]:
```python
levene_stat, p_value = levene(summer,winter,fall,spring)

print('Levene_stat : ', levene_stat)
print('p-value : ', p_value)

if p_value < 0.05:
    print('The samples has different variance')
else:
    print('The samples has similar variance')
```

```
Levene_stat :  187.7706624026276
p-value :  1.0147116860043298e-118
The samples has different variance
```

The samples are not normally distributed and do not have the same variance, f_oneway test (ANOVA Test) cannot be performed here, we can perform its non parametric equivalent test i.e., Kruskal test for independent samples.

### 14.13 Kruskal Test

[38]:
```python
alpha = 0.05
test_stat, p_value = kruskal(summer,winter,fall,spring)
print('Test Statistic =', test_stat)
print('p value =', p_value)

if p_value < alpha:
    print('Reject Null Hypothesis')
```

```
    print("The median of bikes rented is different across seasons")
else:
    print('Failed to reject Null Hypothesis')
    print("The median of bikes rented is same for across seasons.")
```

```
Test Statistic = 699.6668548181915
p value = 2.4790083726176776e-151
Reject Null Hypothesis
The median of bikes rented is different across seasons
```

From above, we can confirm that The mean number of E-bikes rented differs across various Seasons.

### 14.14 Are weather comditions significantly same or different for different Seasons?

1. Set up Null and Alternate Hypothesis

- Null Hypothesis ( H0 ) - weather is independent of season
- Alternate Hypothesis ( HA ) - -weather is dependent of seasons.

2. Choose the distribution (Gaussian, Binomial, etc), and hence the test statistic.
3. Select the Left vs Right vs Two-Tailed test, as per the hypothesis
4. Compute the P-Value
5. Compare the P-Value to the Significance Level ( ) and Fail to reject/reject the Null Hypothesis accordingly.

[39]:
```
pd.crosstab(bike_data['weather'],bike_data['season'])
```

[39]:

| season | spring | summer | fall | winter |
|---|---|---|---|---|
| weather | | | | |
| Clear | 1759 | 1801 | 1930 | 1702 |
| Misty_cloudy | 715 | 708 | 604 | 807 |
| Rain | 211 | 224 | 199 | 225 |
| Heavy_rain | 1 | 0 | 0 | 0 |

[42]:
```
chi_stat , p_value , dof , expected = chi2_contingency(pd.
 ↪crosstab(bike_data['weather'],bike_data['season']))

print("chi_stat : ",chi_stat)
print("p_value : ",p_value)
print("dof : ",dof)
print("expected : ",expected,'\n')

alpha = 0.05
if p_value< alpha:
    print("Reject Ho")
    print("Weather is dependent on season")
```

```python
else:
    print("Fail to Reject Ho")
    print("Weather is independent on season")
```

```
chi_stat :  49.15865559689363
p_value :  1.5499250736864862e-07
dof :  9
expected :  [[1.77454639e+03 1.80559765e+03 1.80559765e+03 1.80625831e+03]
 [6.99258130e+02 7.11493845e+02 7.11493845e+02 7.11754180e+02]
 [2.11948742e+02 2.15657450e+02 2.15657450e+02 2.15736359e+02]
 [2.46738931e-01 2.51056403e-01 2.51056403e-01 2.51148264e-01]]

Reject Ho
Weather is dependent on season
```

### 14.15 Insights

The Chi-square test result (chi2: 49.16) and a very low p-value (almost 0) indicate a statistically significant relationship. This means weather and season are not independent.

Spring and Summer tend to have more favorable weather conditions compared to Fall and Winter.

The weather you experience is strongly linked to the season you're in. Spring and Summer generally bring better weather for riding bikes, while Fall and Winter might have less favorable conditions.

### 14.16 Recommendations

Capitalize on spring and summer's popularity with bike rentals. Offer special discounts or packages during these peak demand months to attract more riders.

Recognize the impact of weather. Create promotions targeting clear and cloudy days with weather-specific discounts to attract more customers during these favorable conditions.

Implement time-based pricing with lower rates during off-peak hours. This encourages rentals when demand is lower, balancing demand and optimizing resources.

Provide amenities like umbrellas, rain jackets, or water bottles to combat high humidity and moderate temperatures. These small touches can significantly improve customer experience and encourage repeat business.

Collaborate with weather services to provide real-time weather updates in your app or marketing campaigns. This allows users to find ideal biking conditions and attracts those with weather preferences.

Allocate resources for seasonal bike maintenance. Conduct thorough checks before peak seasons and maintain bikes regularly year-round to minimize breakdowns and maximize customer satisfaction.

Encourage customer feedback and reviews to identify areas for improvement, understand preferences, and tailor services to better meet expectations.

Offer special discounts on environmental awareness days (Zero Emissions Day, Earth Day, World Environment Day) to attract new users.