🚲🚴 YULU_Bikes - Business CaseStudy 🚴🚲

Hypothesis Testing

Analysed by : KASI

`![y-pic.jpg](attachment:fa91c7cb-4810-4eb4-b85f-26a665066f4a.jpg)`

Introduction:

🚲Yulu: Zipping Through Indian Cities with Green Scooters

`Yulu` , India's pioneering micro-mobility service provider, has embarked on a mission to revolutionize daily commutes by offering unique, sustainable transportation solutions. However, recent revenue setbacks have prompted Yulu to seek the expertise of a consulting company to delve into the factors influencing the demand for their shared electric cycles, specifically in the Indian market. India's leading micromobility platform, transforming urban commutes. Founded in `2017` , Yulu isn't just about rides, it's about a sustainable future.

🌐 Website : www.yulu.bike

The Yulu Way:

- **E-scooters and e-bikes:** Ditch cars, hail a Yulu! These dockless rides, accessed through a slick app, are scattered across Bengaluru, Mumbai, Delhi, and more.
- **Convenience reigns:** Find, unlock, and park your Yulu anywhere within designated zones. No docking dramas, just hop on and off!
- **Green warriors:** Every Yulu ride cuts traffic congestion and carbon emissions, breathing new life into city air.

Impact beyond rides:

- **Over 25,000 Yulu scoots and millions of happy riders:** More than just a fun ride, Yulu is a movement.
- **Boosting the local economy:** Yulu creates jobs, supports businesses, and revitalizes cityscapes.
- **Yulu Wynn:** Introducing India's first truly keyless electric scooter, offering personal e-mobility options.

Yulu's not just a company; it's a vision:

A vision of urban streets buzzing with eco-friendly rides, a vision of cleaner air, and a vision of a future where getting around is effortless and green. So, next time you're in an Indian city, ditch the cab, skip the bus, and Yulu your way to a better future.

In a nutshell:

```
Yulu = Green Mobility. Indian cities = Grateful (and scooting!).
```

Why this case study?

- From Yulu's Perspective:
  - Strategic Expansion: Yulu's decision to enter the Indian market is a strategic move to expand its global footprint. Understanding the demand factors in this new market is essential to tailor their services and strategies accordingly.
  - Revenue Recovery: Yulu's recent revenue decline is a pressing concern. By analyzing the factors affecting demand for shared electric cycles in the Indian market, they can make informed adjustments to regain profitability.
- From Learners' Perspective:
  - Real-World Problem-Solving: It presents an opportunity to apply machine learning and data analysis techniques to address a real-world business problem.
  - Market Insights: Analyzing factors affecting demand in the Indian market equips learners with market research skills. This knowledge is transferable to various industries.
  - Consulting Skills: Learners can develop their ability to act as consultants, providing data-driven insights to organizations

---

Business Problem:

- 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.

Features of the dataset:

- Column Profiling:

Feature	Description
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.
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 - Total_riders	count of total rental bikes including both casual and registered

- weather

Category	Details
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

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
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
from scipy.stats import pearsonr,spearmanr
import statsmodels.api as sm
from statsmodels.formula.api import ols
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler
import warnings
warnings.filterwarnings('ignore')
```

In [2]:
```
yulu_data = pd.read_csv('bike_sharing.csv')
```

In [3]:
```
yd = yulu_data.copy()
yd
```

Out[3]:

	datetime	season	holiday	workingday	weather	temp	atemp	humidity	windspeed	casual	registered	count
0	2011-01-01 00:00:00	1	0	0	1	9.84	14.395	81	0.0000	3	13	16
1	2011-01-01 01:00:00	1	0	0	1	9.02	13.635	80	0.0000	8	32	40
2	2011-01-01 02:00:00	1	0	0	1	9.02	13.635	80	0.0000	5	27	32
3	2011-01-01 03:00:00	1	0	0	1	9.84	14.395	75	0.0000	3	10	13
4	2011-01-01 04:00:00	1	0	0	1	9.84	14.395	75	0.0000	0	1	1
...	...	...	...	...	...	...	...	...	...	...	...	...
10881	2012-12-19 19:00:00	4	0	1	1	15.58	19.695	50	26.0027	7	329	336
10882	2012-12-19 20:00:00	4	0	1	1	14.76	17.425	57	15.0013	10	231	241
10883	2012-12-19 21:00:00	4	0	1	1	13.94	15.910	61	15.0013	4	164	168
10884	2012-12-19 22:00:00	4	0	1	1	13.94	17.425	61	6.0032	12	117	129
10885	2012-12-19 23:00:00	4	0	1	1	13.12	16.665	66	8.9981	4	84	88

10886 rows × 12 columns

Exploration of data :

In [4]:
```
yd.rename(columns={'count':'total_riders'},inplace=True)
```

In [5]:
```
yd.shape
```

Out[5]:
```
(10886, 12)
```

In [6]:
```
yd.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  total_riders  10886 non-null  int64  
dtypes: float64(3), int64(8), object(1)
memory usage: 1020.7+ KB
```

🏷️ Insights

Observations :

- There are 10886 rows and 12 columns in the data.
- There are no null values.
- There are also no duplicate values.
- The columns **"datetime"** have object datatype.
- The columns **"season"**, **"holiday"**, **"workingday"**, **"weather"**, **"humidity"**, **"casual"**, **"registered"** and **"total_riders"** have int datatype.
- The columns **"temp"**, **"atemp"**, and **"windspeed"** have float datatype.

Data Type Conversion

- The data type of datetime should be in datetime format.
- At the same time season, holiday, workingday, weather should in object format as they are categorical in nature

In [5]:
```
yd['datetime'] = pd.to_datetime(yd['datetime'])
```

In [8]:
```
yd['datetime'].dtype
```

Out[8]:
```
dtype('<M8[ns]')
```

In [6]:
```
yd['year'] = yd['datetime'].dt.year
yd['month'] = yd['datetime'].dt.month
yd['hour'] = yd['datetime'].dt.hour
yd['month'] = yd['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'})
```

In [10]:
```
yd.skew(numeric_only = True)
```

Out[10]:
```
season         -0.007076
holiday         5.660517
workingday     -0.776163
weather         1.243484
temp            0.003691
atemp          -0.102560
humidity       -0.086335
windspeed       0.588767
casual          2.495748
registered      1.524805
total_riders    1.242066
year           -0.007717
hour           -0.009125
dtype: float64
```

🏷️ Insights:

Skewness Analysis of Variables

- Symmetrical Majority:
  - The majority of the variables, including 'season' and 'temp', exhibit skewness values close to zero, suggesting relatively symmetrical distributions.

>

- Positive Skewness Insights:
  - Variables such as 'holiday', 'weather', 'windspeed', 'casual', 'registered', and 'count' demonstrate positive skewness, pointing to a concentration of lower values and a right skewed in their distributions.
- Negative Skewness Observations:
  - In contrast, 'workingday', 'atemp', and 'humidity' exhibit negative skewness, implying a concentration of higher values and a left skewed in their distributions.

📝 Statistical Summary

In [11]:
```
yd.describe(include='all').T
```

Out[11]:

	count	unique	top	freq	mean	min	25%	50%	75%	max	std
datetime	10886	NaN	NaN	NaN	2011-12-27 05:56:22.399411968	2011-01-01 00:00:00	2011-07-02 07:15:00	2012-01-01 20:30:00	2012-07-01 12:45:00	2012-12-19 23:00:00	NaN
season	10886.0	NaN	NaN	NaN	2.506614	1.0	2.0	3.0	4.0	4.0	1.116174
holiday	10886.0	NaN	NaN	NaN	0.028569	0.0	0.0	0.0	0.0	1.0	0.166599
workingday	10886.0	NaN	NaN	NaN	0.680875	0.0	0.0	1.0	1.0	1.0	0.466159
weather	10886.0	NaN	NaN	NaN	1.418427	1.0	1.0	1.0	2.0	4.0	0.633839
temp	10886.0	NaN	NaN	NaN	20.23086	0.82	13.94	20.5	26.24	41.0	7.79159
atemp	10886.0	NaN	NaN	NaN	23.655084	0.76	16.665	24.24	31.06	45.455	8.474601
humidity	10886.0	NaN	NaN	NaN	61.88646	0.0	47.0	62.0	77.0	100.0	19.245033
windspeed	10886.0	NaN	NaN	NaN	12.799395	0.0	7.0015	12.998	16.9979	56.9969	8.164537
casual	10886.0	NaN	NaN	NaN	36.021955	0.0	4.0	17.0	49.0	367.0	49.960477
registered	10886.0	NaN	NaN	NaN	155.552177	0.0	36.0	118.0	222.0	886.0	151.039033
total_riders	10886.0	NaN	NaN	NaN	191.574132	1.0	42.0	145.0	284.0	977.0	181.144454
year	10886.0	NaN	NaN	NaN	2011.501929	2011.0	2011.0	2012.0	2012.0	2012.0	0.500019
month	10886	12	May	912	NaN	NaN	NaN	NaN	NaN	NaN	NaN
hour	10886.0	NaN	NaN	NaN	11.541613	0.0	6.0	12.0	18.0	23.0	6.915838

📜📜Duplicate Detection

In [12]:
```
yd[yd.duplicated()]
```

Out[12]:

datetime	season	holiday	workingday	weather	temp	atemp	humidity	windspeed	casual	registered	total_riders	year	month	hour

---

🏷️ Insights

- The dataset does not contain any duplicates.

🔸❓Null Detection

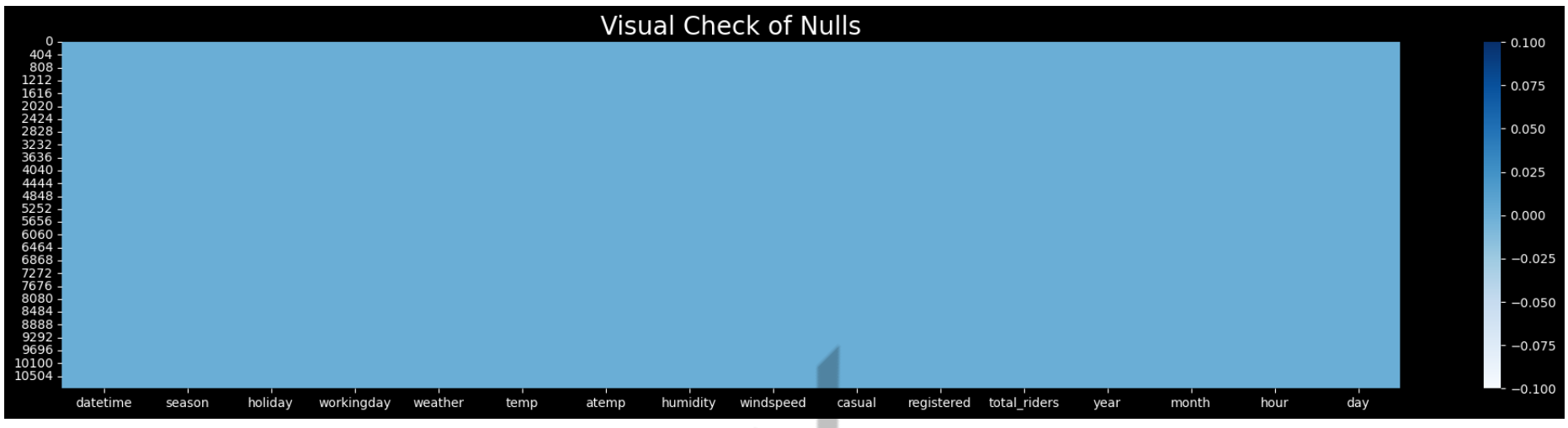
In [13]:
```
yd.isna().any()
```

Out[13]:
```
datetime        False
season          False
holiday         False
workingday      False
weather         False
temp            False
atemp           False
humidity        False
windspeed       False
casual          False
registered      False
total_riders    False
year            False
month           False
hour            False
dtype: bool
```

In [14]:
```
yd.isna().sum()
```

Out[14]:
```
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
```

In [34]:
```
plt.figure(figsize=(24,5))
plt.style.use('dark_background')
sns.heatmap(yd.isnull(),cmap='Blues')
plt.title('Visual Check of Nulls',fontsize=20)
plt.show()
```



In [16]:
```
print(list(yd.columns))
```

```
['datetime', 'season', 'holiday', 'workingday', 'weather', 'temp', 'atemp', 'humidity', 'windspeed', 'casual', 'registered', 'to
tal_riders', 'year', 'month', 'hour']
```

Changing the Datatype of Columns

In [7]:
```
for _ in yd.columns[1:5]:
    yd[_] = yd[_].astype('category')
yd.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10886 entries, 0 to 10885
Data columns (total 15 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   datetime      10886 non-null  datetime64[ns]
 1   season        10886 non-null  category
 2   holiday       10886 non-null  category
 3   workingday    10886 non-null  category
 4   weather       10886 non-null  category
 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  total_riders  10886 non-null  int64
 12  year          10886 non-null  int32
 13  month         10886 non-null  object
 14  hour          10886 non-null  int32
dtypes: category(4), datetime64[ns](1), float64(3), int32(2), int64(4), object(1)
memory usage: 893.8+ KB
```

In [18]:
```
yd.describe(include='category').T
```

Out[18]:

	count	unique	top	freq
season	10886	4	4	2734
holiday	10886	2	0	10575
workingday	10886	2	1	7412
weather	10886	4	1	7192

Insights

- The dataset does not contain any missing values.

🕵️ Exploratory Data Analysis

🔰 Non-Graphical Analysis

In [8]:
```
yd['season']= yd['season'].map(str)
season_mapping = {'1':'spring', '2':'summer', '3':'fall', '4':'winter'}
yd["season"] = yd["season"].map(lambda x: season_mapping[x])


yd['holiday']= yd['holiday'].map(str)
holiday_mapping = {'0':'no', '1':'yes'}
yd["holiday"] = yd["holiday"].map(lambda x: holiday_mapping[x])

yd['workingday']= yd['workingday'].map(str)
working_day_mapping = {'0':'no', '1':'yes'}
yd["workingday"] = yd["workingday"].map(lambda x: working_day_mapping[x])

yd['weather']= yd['weather'].map(str)
weather_mapping = {'1':'clear', '2':'partly_cloudy', '3':'rain', '4':'heavy rain'}
yd["weather"] = yd["weather"].map(lambda x: weather_mapping[x])
```

In [20]:
```
yd.sample(7)
```

Out[20]:

	datetime	season	holiday	workingday	weather	temp	atemp	humidity	windspeed	casual	registered	total_riders	year	month	hou
9995	2012-11-01 20:00:00	winter	no	yes	partly_cloudy	15.58	19.695	54	16.9979	16	280	296	2012	November	20
8567	2012-07-18 08:00:00	fall	no	yes	clear	31.98	37.120	62	7.0015	55	624	679	2012	July	8
9450	2012-09-17 03:00:00	fall	no	yes	clear	20.50	24.240	77	6.0032	0	7	7	2012	September	3
3692	2011-09-04 18:00:00	fall	no	no	clear	31.16	36.365	66	12.9980	187	169	356	2011	September	18
6141	2012-02-12 03:00:00	spring	no	no	partly_cloudy	4.10	2.275	46	46.0022	0	14	14	2012	February	3
2175	2011-05-17 13:00:00	summer	no	yes	partly_cloudy	22.96	26.515	88	22.0028	31	141	172	2011	May	13
6365	2012-03-02 11:00:00	spring	no	yes	clear	15.58	19.695	46	12.9980	40	154	194	2012	March	11

In [21]:
```
#checking the unique values for columns
for col in yd.columns:
    print()
    print('Total Unique Values in',col,'column are :-',yd[col].nunique())
    print('Unique Values in',col,'column are :-\n',yd[col].unique())
    print()
    print('-'*140)
```

```
Total Unique Values in datetime column are :- 10886
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]

------------------------------------------------------------------------------------------------------------------------
------------

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

------------------------------------------------------------------------------------------------------------------------
------------

Total Unique Values in holiday column are :- 2
Unique Values in holiday column are :-
 ['no', 'yes']
Categories (2, object): ['no', 'yes']

------------------------------------------------------------------------------------------------------------------------
------------

Total Unique Values in workingday column are :- 2
Unique Values in workingday column are :-
 ['no', 'yes']
Categories (2, object): ['no', 'yes']

------------------------------------------------------------------------------------------------------------------------
------------

Total Unique Values in weather column are :- 4
Unique Values in weather column are :-
 ['clear', 'partly_cloudy', 'rain', 'heavy rain']
Categories (4, object): ['clear', 'partly_cloudy', 'rain', 'heavy rain']

------------------------------------------------------------------------------------------------------------------------
------------

Total Unique Values in temp column are :- 49
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.  ]

------------------------------------------------------------------------------------------------------------------------
------------

Total Unique Values in atemp column are :- 60
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]

------------------------------------------------------------------------------------------------------------------------
------------

Total Unique Values in humidity column are :- 89
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]

------------------------------------------------------------------------------------------------------------------------
------------

Total Unique Values in windspeed column are :- 28
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]

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

```
Total Unique Values in casual column are :- 309
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]

------------------------------------------------------------------------------------------------------------------------
------------

Total Unique Values in registered column are :- 731
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]

------------------------------------------------------------------------------------------------------------------------
------------

Total Unique Values in total_riders column are :- 822
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]

------------------------------------------------------------------------------------------------------------------------
------------
Total Unique Values in year column are :- 2
Unique Values in year column are :-
 [2011 2012]

------------------------------------------------------------------------------------------------------------------------
------------

Total Unique Values in month column are :- 12
Unique Values in month column are :-
 ['January' 'February' 'March' 'April' 'May' 'June' 'July' 'August'
 'September' 'October' 'November' 'December']

------------------------------------------------------------------------------------------------------------------------
------------

Total Unique Values in hour column are :- 24
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]

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

In [22]:
```
yd.columns
```

Out[22]:
```
Index(['datetime', 'season', 'holiday', 'workingday', 'weather', 'temp',
       'atemp', 'humidity', 'windspeed', 'casual', 'registered',
       'total_riders', 'year', 'month', 'hour'],
      dtype='object')
```

In [15]:
```
for _ in yd.columns[5:]:
    if yd[_].dtype != 'category':
        print(f'Value_counts of the column {_} are :-{yd[_].value_counts().to_frame().reset_index()}')
        print()
        print('-'*140)
        print()
```

Value_counts of the column temp are :-

	temp	count
0	14.76	467
1	26.24	453
2	28.70	427
3	13.94	413
4	18.86	406
5	22.14	403
6	25.42	403
7	16.40	400
8	22.96	395
9	27.06	394
10	24.60	390
11	12.30	385
12	21.32	362
13	17.22	356
14	13.12	356
15	29.52	353
16	10.66	332
17	18.04	328
18	20.50	327
19	30.34	299
20	9.84	294
21	15.58	255
22	9.02	248
23	31.16	242
24	8.20	229
25	27.88	224
26	23.78	203
27	32.80	202
28	11.48	181
29	19.68	170
30	6.56	146
31	33.62	130
32	5.74	107
33	7.38	106
34	31.98	98
35	34.44	80
36	35.26	76
37	4.92	60
38	36.90	46
39	4.10	44
40	37.72	34
41	36.08	23
42	3.28	11
43	0.82	7
44	38.54	7
45	39.36	6
46	2.46	5
47	1.64	2
48	41.00	1

Value_counts of the column atemp are :-

	atemp	count
0	31.060	671
1	25.760	423
2	22.725	406
3	20.455	400
4	26.515	395
5	16.665	381
6	25.000	365
7	33.335	364
8	21.210	356
9	30.305	350
10	15.150	338
11	21.970	328
12	24.240	327
13	17.425	314
14	31.820	299
15	34.850	283
16	27.275	282
17	32.575	272
18	11.365	271
19	14.395	269
20	29.545	257
21	19.695	255
22	15.910	254
23	12.880	247
24	13.635	237
25	34.090	224
26	12.120	195
27	28.790	175
28	23.485	170
29	10.605	166
30	35.605	159
31	9.850	127
32	18.180	123
33	36.365	123
34	37.120	118
35	9.090	107

```
36  37.880     97
37  28.030     80
38   7.575     75
39  38.635     74
40   6.060     73
41  39.395     67
42   6.820     63
43   8.335     63
44  18.940     45
45  40.150     45
46  40.910     39
47   5.305     25
48  42.425     24
49  41.665     23
50   3.790     16
51   4.545     11
52   3.030      7
53  43.940      7
54   2.275      7
55  43.180      7
56  44.695      3
57   0.760      2
58   1.515      1
59  45.455      1

-----------------------------------------------------------------------------------------------------------------------
------------

Value_counts of the column humidity are :-
    humidity  count
0         88    368
1         94    324
2         83    316
3         87    289
4         70    259
..       ...    ...
84         8      1
85        10      1
86        97      1
87        96      1
88        91      1

[89 rows x 2 columns]

-----------------------------------------------------------------------------------------------------------------------
------------

Value_counts of the column windspeed are :-
    windspeed  count
0      0.0000   1313
1      8.9981   1120
2     11.0014   1057
3     12.9980   1042
4      7.0015   1034
5     15.0013    961
6      6.0032    872
7     16.9979    824
8     19.0012    676
9     19.9995    492
10    22.0028    372
11    23.9994    274
12    26.0027    235
13    27.9993    187
14    30.0026    111
15    31.0009     89
16    32.9975     80
17    35.0008     58
18    39.0007     27
19    36.9974     22
20    43.0006     12
21    40.9973     11
22    43.9989      8
23    46.0022      3
24    56.9969      2
25    47.9988      2
26    51.9987      1
27    50.0021      1

-----------------------------------------------------------------------------------------------------------------------
------------

Value_counts of the column casual are :-
     casual  count
0         0    986
1         1    667
2         2    487
3         3    438
4         4    354
..      ...    ...
304     332      1
305     361      1
306     356      1
307     331      1
```

```
308     304      1

[309 rows x 2 columns]

------------------------------------------------------------------------------------------------------------------------
------------

Value_counts of the column registered are :-
     registered  count
0             3    195
1             4    190
2             5    177
3             6    155
4             2    150
..          ...    ...
726         570      1
727         422      1
728         678      1
729         565      1
730         636      1

[731 rows x 2 columns]

------------------------------------------------------------------------------------------------------------------------
------------

Value_counts of the column total_riders are :-
     total_riders  count
0               5    169
1               4    149
2               3    144
3               6    135
4               2    132
..            ...    ...
817           801      1
818           629      1
819           825      1
820           589      1
821           636      1

[822 rows x 2 columns]

------------------------------------------------------------------------------------------------------------------------
------------

Value_counts of the column year are :-
   year  count
0  2012   5464
1  2011   5422

------------------------------------------------------------------------------------------------------------------------
------------

Value_counts of the column month are :-
        month  count
0         May    912
1        June    912
2        July    912
3      August    912
4    December    912
5     October    911
6    November    911
7       April    909
8   September    909
9    February    901
10      March    901
11    January    884

------------------------------------------------------------------------------------------------------------------------
------------

Value_counts of the column hour are :-
    hour  count
0     12    456
1     13    456
2     22    456
3     21    456
4     20    456
5     19    456
6     18    456
7     17    456
8     16    456
9     15    456
10    14    456
11    23    456
12    11    455
13    10    455
14     9    455
15     8    455
16     7    455
17     6    455
18     0    455
19     1    454
```

```
20     5    452
21     2    448
22     4    442
23     3    433

-------------------------------------------------------------------------------------------------------------------------------
------------

Value_counts of the column day are :-
         day  count
0   Saturday   1584
1     Sunday   1579
2   Thursday   1553
3     Monday   1551
4  Wednesday   1551
5    Tuesday   1539
6     Friday   1529

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

🚀What is the time period for which the data is given ?

In [24]:
```
yd.datetime.min()
```

Out[24]:
```
Timestamp('2011-01-01 00:00:00')
```

In [25]:
```
yd.datetime.max()
```

Out[25]:
```
Timestamp('2012-12-19 23:00:00')
```

In [26]:
```
yd.datetime.max() - yd.datetime.min()
```

Out[26]:
```
Timedelta('718 days 23:00:00')
```

In [12]:
```
yd['day'] = yd['datetime'].dt.day_name()
```

In [28]:
```
yd.sample(6)
```

Out[28]:

	datetime	season	holiday	workingday	weather	temp	atemp	humidity	windspeed	casual	registered	total_riders	year	month	hour
2061	2011-05-12 19:00:00	summer	no	yes	partly_cloudy	24.60	30.305	64	12.9980	59	305	364	2011	May	19
4937	2011-11-18 19:00:00	winter	no	yes	clear	11.48	13.635	41	11.0014	7	183	190	2011	November	19
6387	2012-03-03 09:00:00	spring	no	no	rain	15.58	19.695	87	6.0032	7	87	94	2012	March	9
4472	2011-10-18 09:00:00	winter	no	yes	partly_cloudy	21.32	25.000	77	6.0032	29	198	227	2011	October	9
1679	2011-04-15 21:00:00	summer	yes	no	partly_cloudy	16.40	20.455	76	19.9995	25	77	102	2011	April	21
1128	2011-03-11 16:00:00	spring	no	yes	partly_cloudy	13.94	15.150	49	19.9995	18	95	113	2011	March	16

In [13]:
```
for _ in yd.columns:
    if yd[_].dtype=='category':
        display(yd[_].value_counts().to_frame().reset_index())
        print()
        print('-'*137)
        print()
```

	season	count
0	winter	2734
1	fall	2733
2	summer	2733
3	spring	2686

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

	holiday	count
0	no	10575
1	yes	311

---

	workingday	count
0	yes	7412
1	no	3474

---

	weather	count
0	clear	7192
1	partly_cloudy	2834
2	rain	859
3	heavy rain	1

In [16]:
```
for _ in yd.columns:
    if yd[_].dtype=='category':
        print()
        display(np.round(yd[_].value_counts(normalize = True) * 100, 2).reset_index())
        print()
        print('-'*137)
        print()
```

	season	proportion
0	winter	25.11
1	fall	25.11
2	summer	25.11
3	spring	24.67

---

	holiday	proportion
0	no	97.14
1	yes	2.86

---

	workingday	proportion
0	yes	68.09
1	no	31.91

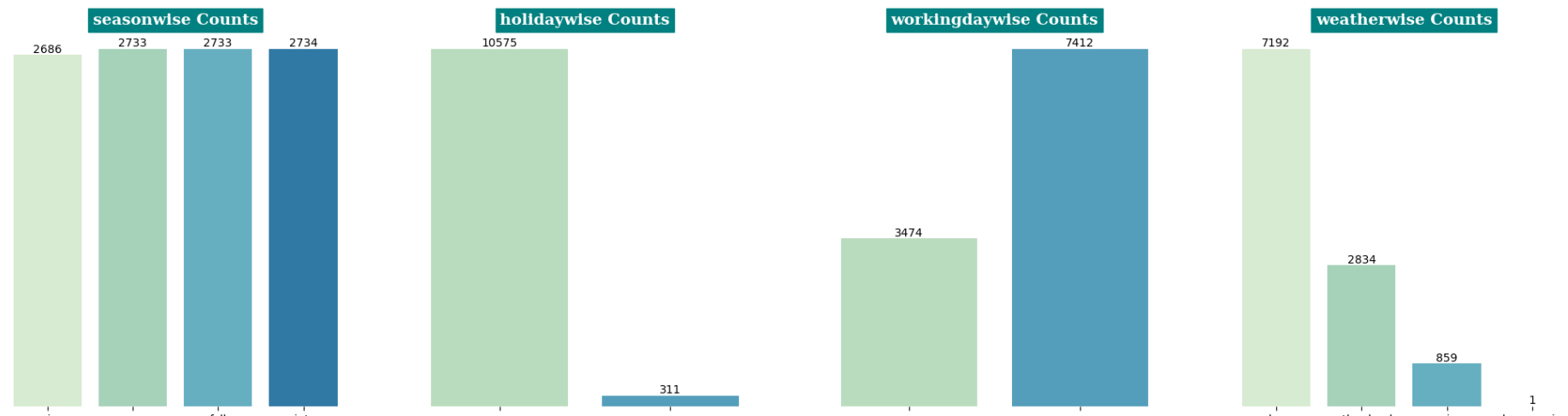
---

	weather	proportion
0	clear	66.07
1	partly_cloudy	26.03
2	rain	7.89
3	heavy rain	0.01

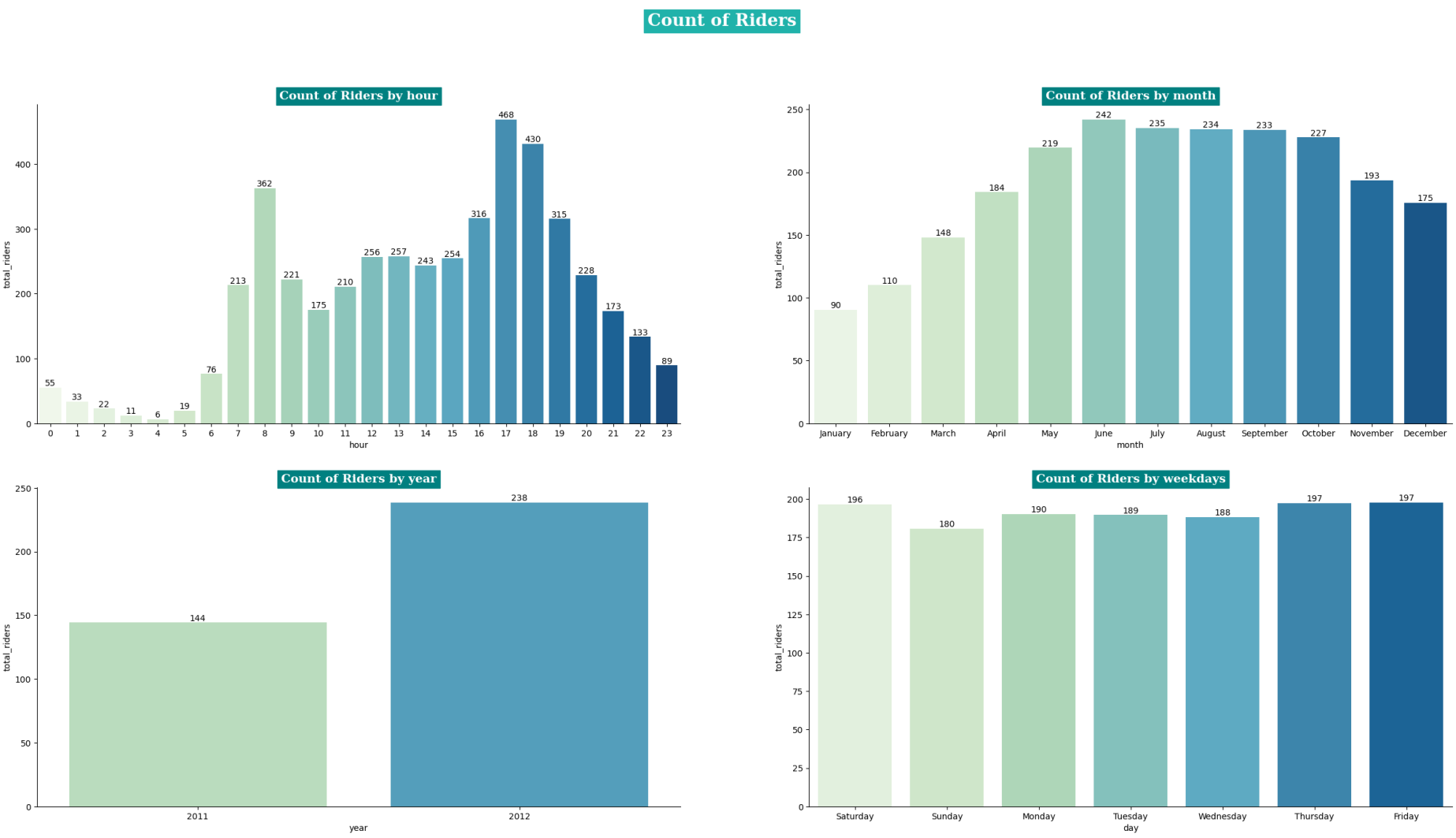
📊 Visual analysis

📈 Univariate and Bivariate graphs

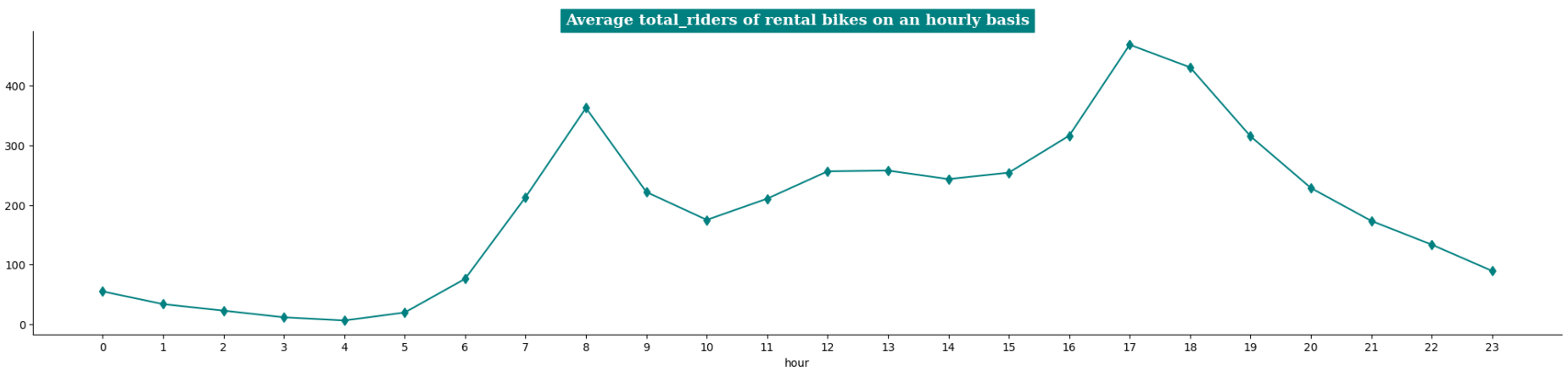
In [37]:
```
plt.figure(figsize=(25,6))
plt.style.use('default')
plt.style.use('seaborn-bright')

i=1
for _ in yd.columns:
    if yd[_].dtype=='category':
        plt.subplot(1,4,i)
        a=sns.countplot(data=yd, x=yd[_],palette='GnBu')
        plt.title(f'{_}wise Counts',fontsize=14,fontfamily='serif',fontweight='bold',backgroundcolor='teal',color='w')
        a.bar_label(a.containers[0], label_type='edge')
        sns.despine(left=True,bottom=True)
        plt.yticks([])
        plt.ylabel('')
        i+=1
```



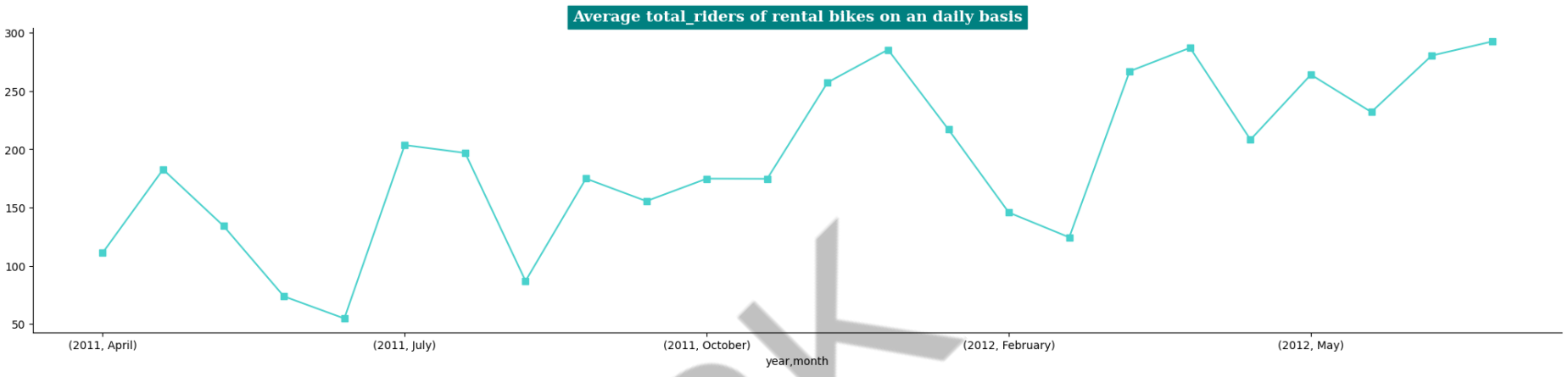
In [38]:
```
plt.figure(figsize=(30,15))
plt.suptitle('Count of Riders',fontsize=20,fontfamily='serif',fontweight='bold',backgroundcolor='lightseagreen',color='w')

plt.subplot(221)
b=sns.barplot(data=yd, x="hour", y="total_riders",palette='GnBu',ci=None)
b.bar_label(b.containers[0],fmt='%d') # %d-int
plt.title('Count of Riders by hour',fontsize=14,fontfamily='serif',fontweight='bold',backgroundcolor='teal',color='w')

plt.subplot(222)
b=sns.barplot(data=yd, x="month", y="total_riders",palette='GnBu',ci=None)
b.bar_label(b.containers[0], label_type='edge',fmt='%d')
plt.title('Count of Riders by month',fontsize=14,fontfamily='serif',fontweight='bold',backgroundcolor='teal',color='w')

plt.subplot(223)
b=sns.barplot(data=yd, x="year", y="total_riders",palette='GnBu',ci=None)
b.bar_label(b.containers[0], label_type='edge',fmt='%d')
plt.title('Count of Riders by year',fontsize=14,fontfamily='serif',fontweight='bold',backgroundcolor='teal',color='w')

plt.subplot(224)
b=sns.barplot(data=yd, x="day", y="total_riders",palette='GnBu',ci=None)
b.bar_label(b.containers[0], label_type='edge',fmt='%d')
plt.title('Count of Riders by weekdays',fontsize=14,fontfamily='serif',fontweight='bold',backgroundcolor='teal',color='w')
sns.despine()

plt.show()
```

In [112…
```
plt.figure(figsize = (25,5))
plt.title("Average total_riders of rental bikes on an hourly basis"
          ,fontsize=14,fontfamily='serif',fontweight='bold',backgroundcolor='teal',color='w')
yd.groupby('hour')['total_riders'].mean().plot(kind = 'line', marker = 'd',color=cp[0])
plt.xticks(np.arange(0, 24))
sns.despine()
plt.show()
```



In [111…
```
plt.figure(figsize = (25,5))
plt.title("Average total_riders of rental bikes on an daily basis"
          ,fontsize=14,fontfamily='serif',fontweight='bold',backgroundcolor='teal',color='w')
yd.groupby(['year','month'])['total_riders'].mean().plot(kind = 'line', marker = 's',color=cp[1])
sns.despine()
plt.show()
```



🏷️ Insights:

- These patterns indicate that there is a distinct fluctuation in total_riders throughout the day, with low counts during early morning hours, a sudden increase in the morning, a peak count in the afternoon, and a gradual decline in the evening and nighttime.
- These patterns indicate that there is a distinct fluctuation in count throughout the month,yearwise but gradually increasing the count of total_riders

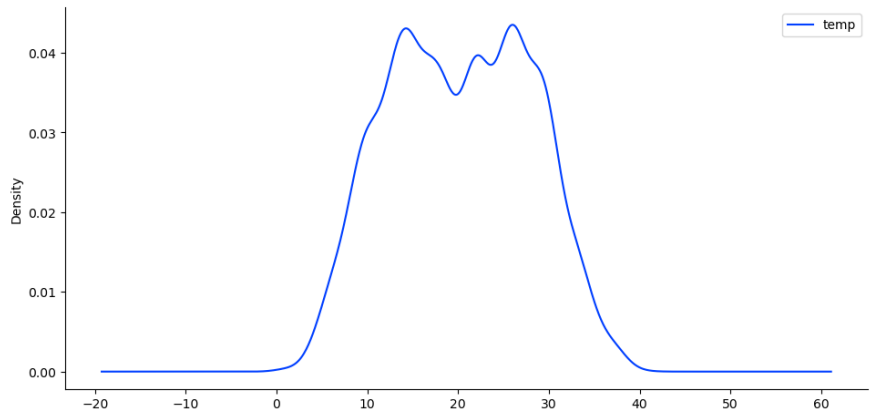
In [17]:
```
num_cols = yd.iloc[:,5:12]
num_cols
```

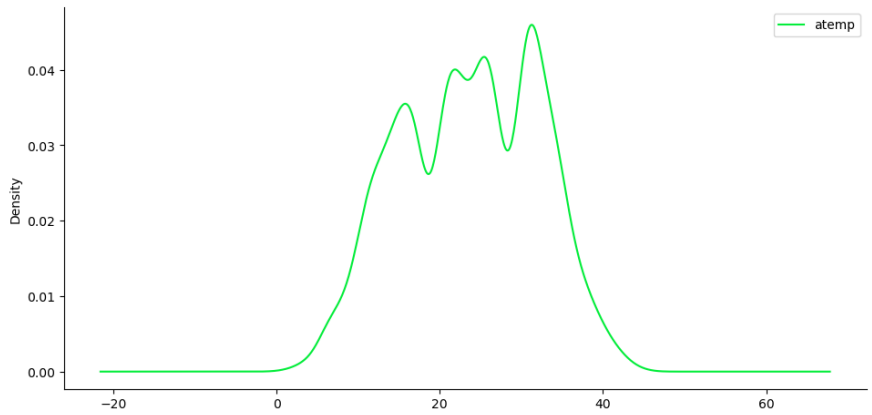
Out[17]:

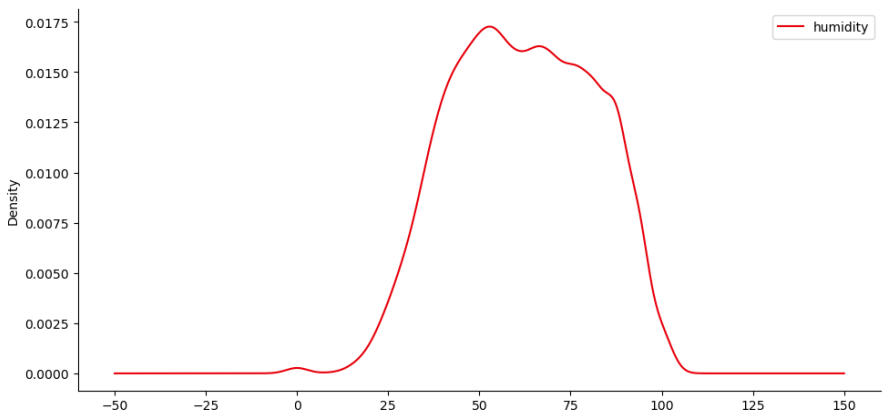
	temp	atemp	humidity	windspeed	casual	registered	total_riders
0	9.84	14.395	81	0.0000	3	13	16
1	9.02	13.635	80	0.0000	8	32	40
2	9.02	13.635	80	0.0000	5	27	32
3	9.84	14.395	75	0.0000	3	10	13
4	9.84	14.395	75	0.0000	0	1	1
...	...	...	...	...	...	...	...
10881	15.58	19.695	50	26.0027	7	329	336
10882	14.76	17.425	57	15.0013	10	231	241
10883	13.94	15.910	61	15.0013	4	164	168
10884	13.94	17.425	61	6.0032	12	117	129
10885	13.12	16.665	66	8.9981	4	84	88

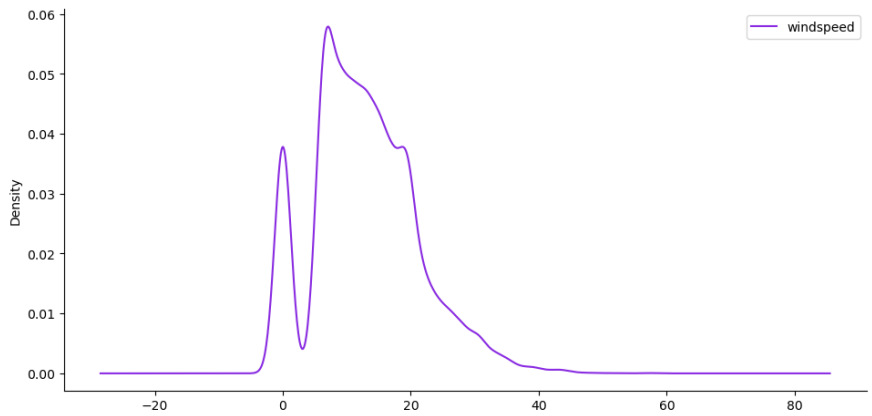
10886 rows × 7 columns

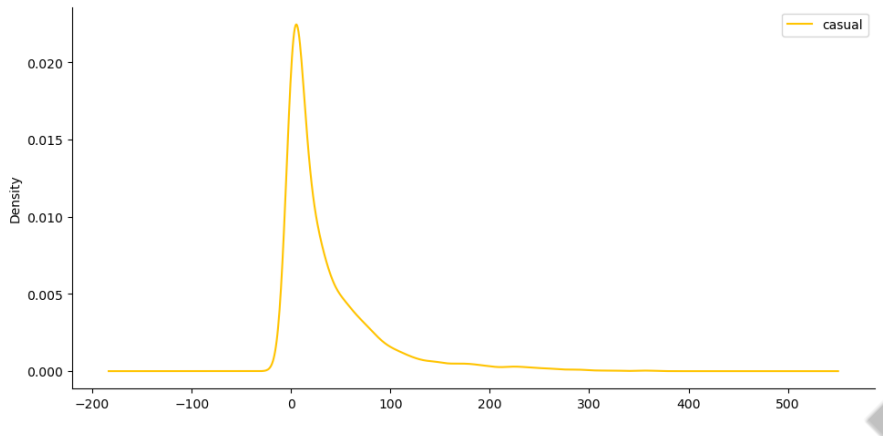
In [105…
```
plt.figure(figsize=(13,0))
plt.title('Normality check - KDE plot',fontsize=16,fontfamily='serif',fontweight='bold',backgroundcolor='lightseagreen',color='w'
plt.rcParams['figure.figsize'] = [25, 25]
num_cols.plot(kind='density', subplots=True, layout=(4,2), sharex=False)
sns.despine()
plt.show()
```

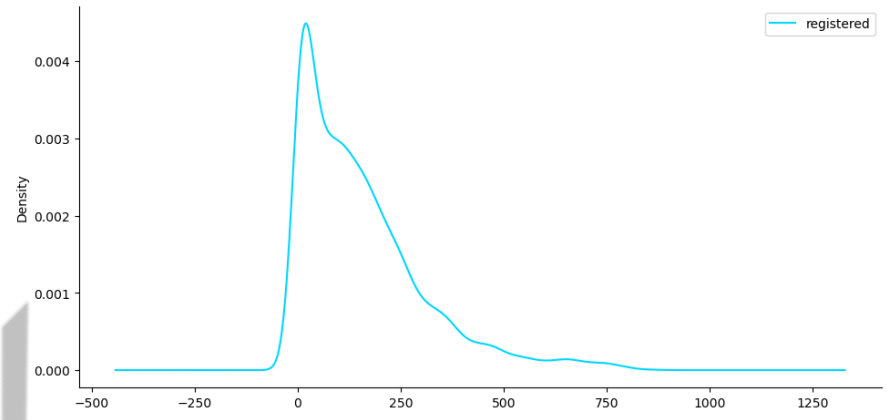


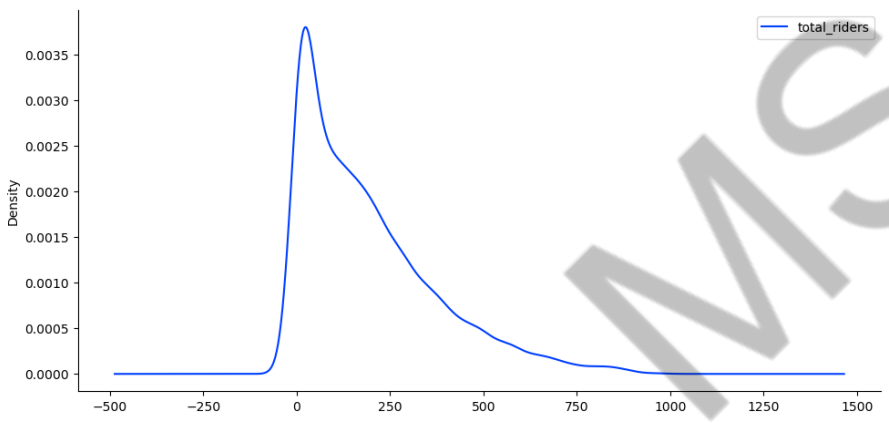












💡 Insights

> **Observations on distributions of Continous variables**
>
> - We can see that the variables - Casual, Registered and Count are skewed to the right.
> - By looking at the distribution for windspeed, we can see that it follows a binomial distribution. Because there were days when there were 0 windspeed and on the other days there was low - moderate windspeed.
> - The Temp, Atemp and Humidity follows (somewhat) a Normal distribution. Because most of their data points are centered around the mean however we will still try to test this hypothesis by using Shapiro.

SHAPIRO-Wilks Test

In [95]:
```
yd.columns
```

Out[95]:
```
Index(['datetime', 'season', 'holiday', 'workingday', 'weather', 'temp',
       'atemp', 'humidity', 'windspeed', 'casual', 'registered',
       'total_riders', 'year', 'month', 'hour'],
      dtype='object')
```

In [101…
```
for i in list(yd.columns[5:-2]):

    print()
    test_statistic, p_value = shapiro(yd[i])
    print(f'Normality Testing for {i} :')
    print("-"*30)

    print(f"The test-statistic for {i} is {test_statistic} with p_value {p_value}")

    if p_value > 0.05:
        print("Hence at 95% confidence level, we fail to reject null hypothesis")
        print(f"Hence we can say that {i} is normally distributed population")
```

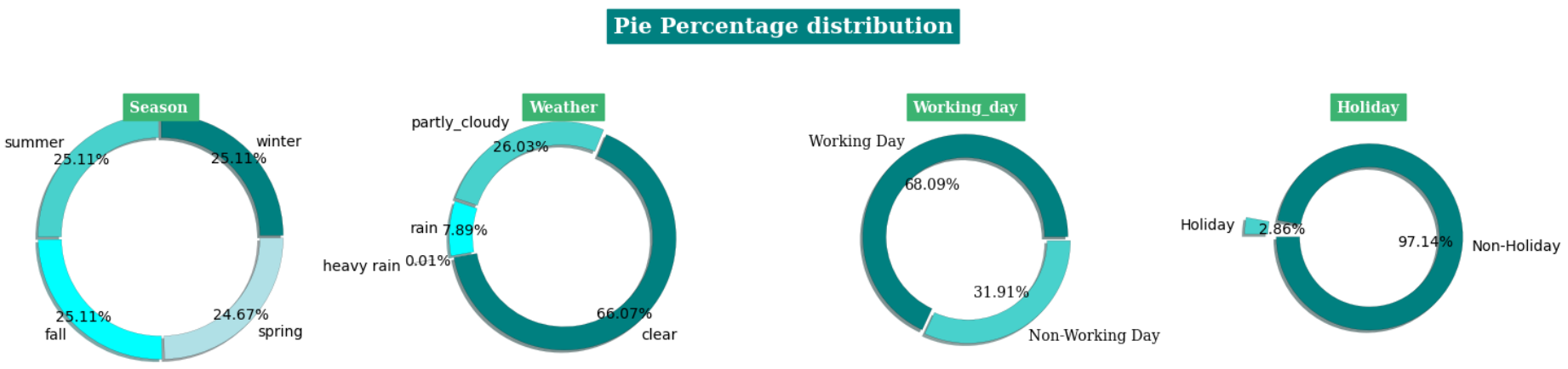
```
    else:
        print("Hence at 95% confidence level, we reject null hypothesis")
        print(f"Hence we can say that {i} is not normally distributed population")

    print()
    print("-"*100)
```

```
Normality Testing for temp :
-----------------------------
The test-statistic for temp is 0.9804227352142334 with p_value 4.577117001754969e-36
Hence at 95% confidence level, we reject null hypothesis
Hence we can say that temp is not normally distributed population

----------------------------------------------------------------------------------------------------

Normality Testing for atemp :
-----------------------------
The test-statistic for atemp is 0.9815532565116882 with p_value 3.35599504562436e-35
Hence at 95% confidence level, we reject null hypothesis
Hence we can say that atemp is not normally distributed population

----------------------------------------------------------------------------------------------------

Normality Testing for humidity :
-----------------------------
The test-statistic for humidity is 0.9822683930397034 with p_value 1.244270413072488e-34
Hence at 95% confidence level, we reject null hypothesis
Hence we can say that humidity is not normally distributed population

----------------------------------------------------------------------------------------------------

Normality Testing for windspeed :
-----------------------------
The test-statistic for windspeed is 0.958724856376648 with p_value 0.0
Hence at 95% confidence level, we reject null hypothesis
Hence we can say that windspeed is not normally distributed population

----------------------------------------------------------------------------------------------------

Normality Testing for casual :
-----------------------------
The test-statistic for casual is 0.7056366205215454 with p_value 0.0
Hence at 95% confidence level, we reject null hypothesis
Hence we can say that casual is not normally distributed population

----------------------------------------------------------------------------------------------------

Normality Testing for registered :
-----------------------------
The test-statistic for registered is 0.8562825322151184 with p_value 0.0
Hence at 95% confidence level, we reject null hypothesis
Hence we can say that registered is not normally distributed population

----------------------------------------------------------------------------------------------------

Normality Testing for total_riders :
-----------------------------
The test-statistic for total_riders is 0.8783695697784424 with p_value 0.0
Hence at 95% confidence level, we reject null hypothesis
Hence we can say that total_riders is not normally distributed population

----------------------------------------------------------------------------------------------------

Normality Testing for year :
-----------------------------
The test-statistic for year is 0.6366432905197144 with p_value 0.0
Hence at 95% confidence level, we reject null hypothesis
Hence we can say that year is not normally distributed population

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

In [14]:
```
cp = ['teal' , 'mediumturquoise' , 'aqua' , 'powderblue' , 'cyan','teal' , 'mediumturquoise' , 'aqua' , 'powderblue' , 'cyan']
```

In [43]:
```
plt.figure(figsize=(15,4))
plt.suptitle('Pie Percentage distribution',fontsize=15,fontfamily='serif',fontweight='bold',backgroundcolor=cp[0],color='w')

plt.subplot(141)
plt.pie(yd['season'].value_counts(), labels=yd['season'].value_counts().index, colors=cp,
       counterclock=True , explode=(0.02,0.02,0.02,0.02) , autopct='%.2f%%', pctdistance=0.905,
       textprops={'color':'k','fontsize':10} , shadow=True, radius=1.3,
       wedgeprops=dict(edgecolor='k',linewidth=0.1,width=0.25))
plt.title('Season ',fontsize=10,fontfamily='serif',fontweight='bold',backgroundcolor='mediumseagreen',color='w')

plt.subplot(142)
plt.pie(yd['weather'].value_counts(), labels=yd['weather'].value_counts().index,
       startangle=190 , explode=(0.02,0.04,0.02,0.45) , autopct='%.2f%%',pctdistance=0.858,
       colors=cp , textprops={'color':'k','fontsize':10} , shadow=True,radius=1.2,
        wedgeprops=dict(edgecolor='k',linewidth=0.1,antialiased=True,width=0.25))
plt.title('Weather',fontsize=10,fontfamily='serif',fontweight='bold',backgroundcolor='mediumseagreen',color='w')

plt.subplot(143)
```

```
yd_workingday = np.round(yd['workingday'].value_counts(normalize = True) * 100, 2)
plt.pie(x = yd_workingday, explode = [0, 0.05], labels = ['Working Day', 'Non-Working Day'],colors=cp,
        autopct = '%.2f%%', textprops = {'fontsize' : 10,'fontfamily' : 'serif','fontweight' : 500},
       wedgeprops=dict(edgecolor='k',linewidth=0.1,width=0.25),radius=1.1, shadow=True)
plt.title('Working_day',fontsize=10,fontfamily='serif',fontweight='bold',backgroundcolor='mediumseagreen',color='w')

plt.subplot(144)
df_holiday = np.round(yd['holiday'].value_counts(normalize = True) * 100, 2)
plt.pie(df_holiday, labels=['Non-Holiday', 'Holiday'],
       startangle=180 , explode=(0.02,0.32) , autopct='%.2f%%',
       colors=cp , textprops={'color':'k','fontsize':10} , shadow=True, radius=1,
       wedgeprops=dict(edgecolor='k',linewidth=0.1,antialiased=True,width=0.25))
plt.title('Holiday',fontsize=10,fontfamily='serif',fontweight='bold',backgroundcolor='mediumseagreen',color='w')
plt.tight_layout()
plt.show()
```



In [35]:
```
yd.columns
```

Out[35]:
```
Index(['datetime', 'season', 'holiday', 'workingday', 'weather', 'temp',
       'atemp', 'humidity', 'windspeed', 'casual', 'registered',
       'total_riders', 'year', 'month', 'hour', 'day'],
      dtype='object')
```

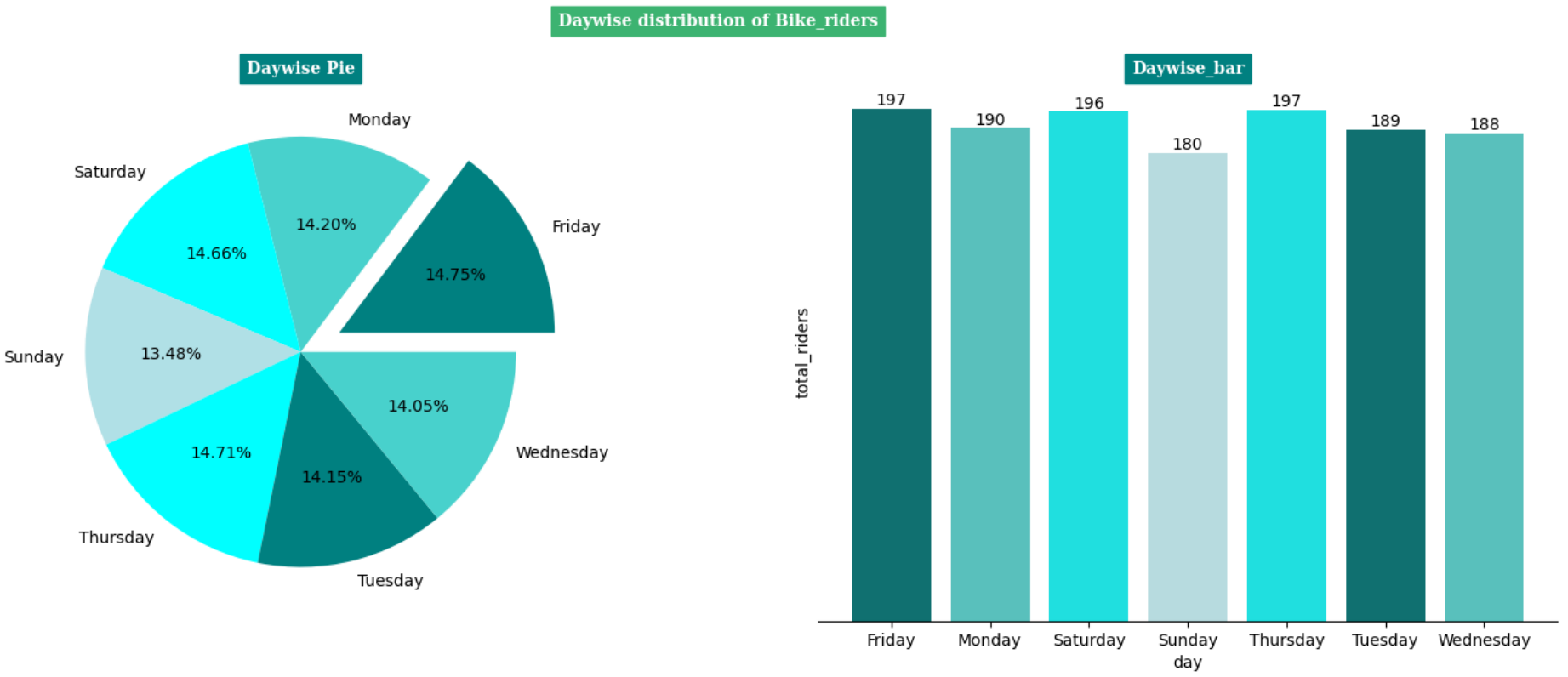
In [39]:
```
yd['day'].unique()
```

Out[39]:
```
array(['Saturday', 'Sunday', 'Monday', 'Tuesday', 'Wednesday', 'Thursday',
       'Friday'], dtype=object)
```

In [63]:
```
day_cnt_riders=yd.groupby('day')['total_riders'].mean().to_frame().reset_index()
display(day_cnt_riders)


plt.figure(figsize=(18,6))
plt.suptitle('Daywise distribution of Bike_riders',fontsize=10,fontfamily='serif',fontweight='bold',
             backgroundcolor='mediumseagreen',color='w')

plt.subplot(121)
plt.pie(data=day_cnt_riders, x=day_cnt_riders['total_riders'], colors=cp,labels=day_cnt_riders['day'],
        explode=(0.2,0,0,0,0,0,0), autopct='%0.2f%%')
plt.title('Daywise Pie',fontsize=10,fontfamily='serif',fontweight='bold',backgroundcolor='teal',color='w')

plt.subplot(122)
a=sns.barplot(data=day_cnt_riders,y=day_cnt_riders['total_riders'],x=day_cnt_riders['day'],palette=cp)
a.bar_label(a.containers[0], label_type='edge',fmt='%d')
plt.title('Daywise_bar',fontsize=10,fontfamily='serif',fontweight='bold',backgroundcolor='teal',color='w')
plt.yticks([])

sns.despine(left=True)
plt.plot()
```

	day	total_riders
0	Friday	197.844343
1	Monday	190.390716
2	Saturday	196.665404
3	Sunday	180.839772
4	Thursday	197.296201
5	Tuesday	189.723847
6	Wednesday	188.411348

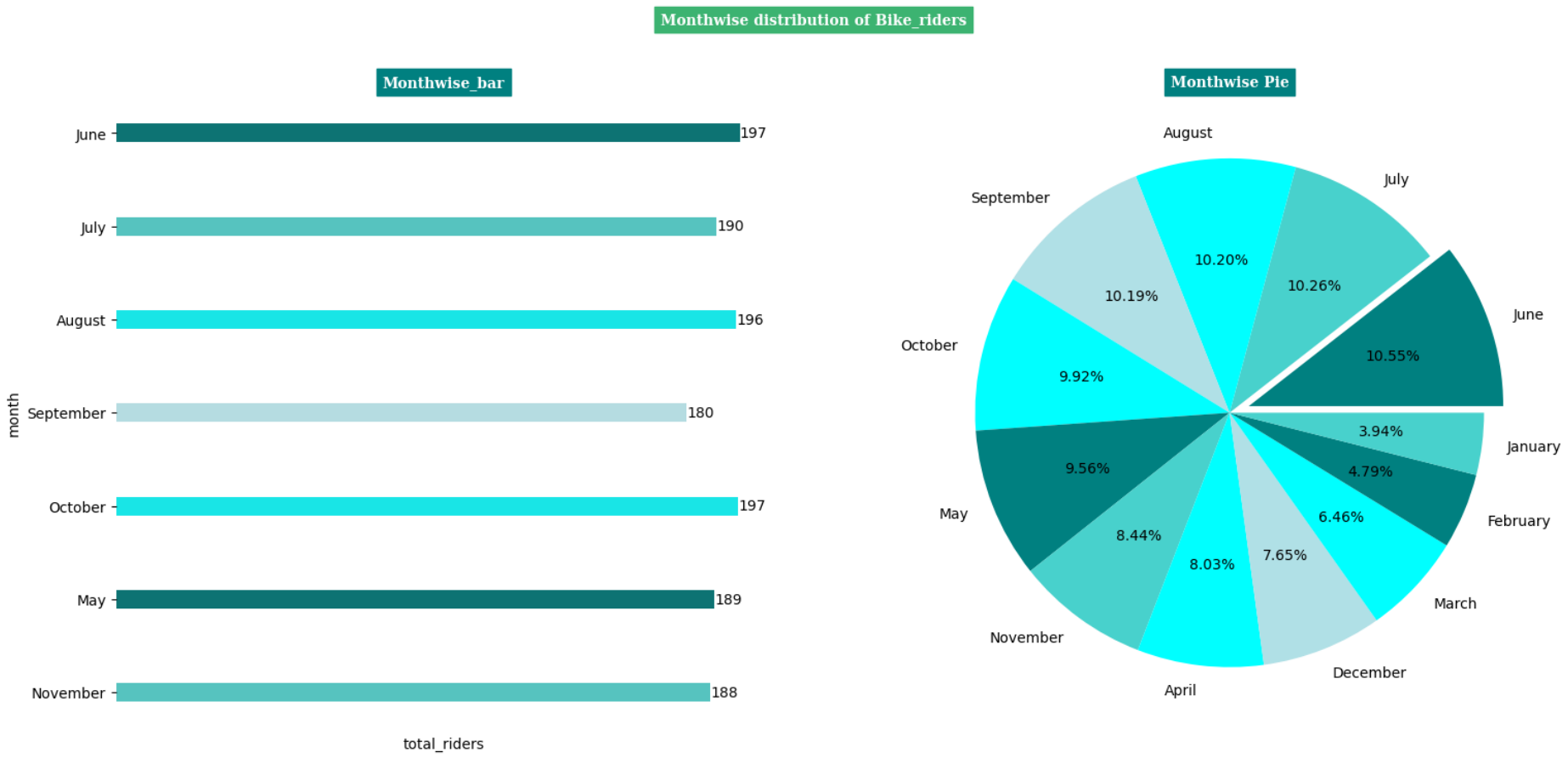
Out[63]:
```
[]
```



In [84]:
```
monthwise_riders = yd.groupby('month')['total_riders'].mean().sort_values(ascending=False).to_frame().reset_index()
display(monthwise_riders)

plt.figure(figsize=(18,8))
plt.suptitle('Monthwise distribution of Bike_riders',fontsize=10,fontfamily='serif',fontweight='bold',
             backgroundcolor='mediumseagreen',color='w')

plt.subplot(122)
plt.pie(data=monthwise_riders, x=monthwise_riders['total_riders'], colors=cp,labels=monthwise_riders['month'],
        explode=(0.08,0,0,0,0,0,0,0,0,0,0,0), autopct='%0.2f%%')
plt.title('Monthwise Pie',fontsize=10,fontfamily='serif',fontweight='bold',backgroundcolor='teal',color='w')

plt.subplot(121)
a=sns.barplot(data=monthwise_riders,x=day_cnt_riders['total_riders'],y=monthwise_riders['month'],palette=cp,width=0.2,saturation=
a.bar_label(a.containers[0], label_type='edge',fmt='%d')
plt.title('Monthwise_bar',fontsize=10,fontfamily='serif',fontweight='bold',backgroundcolor='teal',color='w')
plt.ylim()
plt.xticks([])

sns.despine(bottom=True,left=True)
plt.plot()
```

	month	total_riders
0	June	242.031798
1	July	235.325658
2	August	234.118421
3	September	233.805281
4	October	227.699232
5	May	219.459430
6	November	193.677278
7	April	184.160616
8	December	175.614035
9	March	148.169811
10	February	110.003330
11	January	90.366516

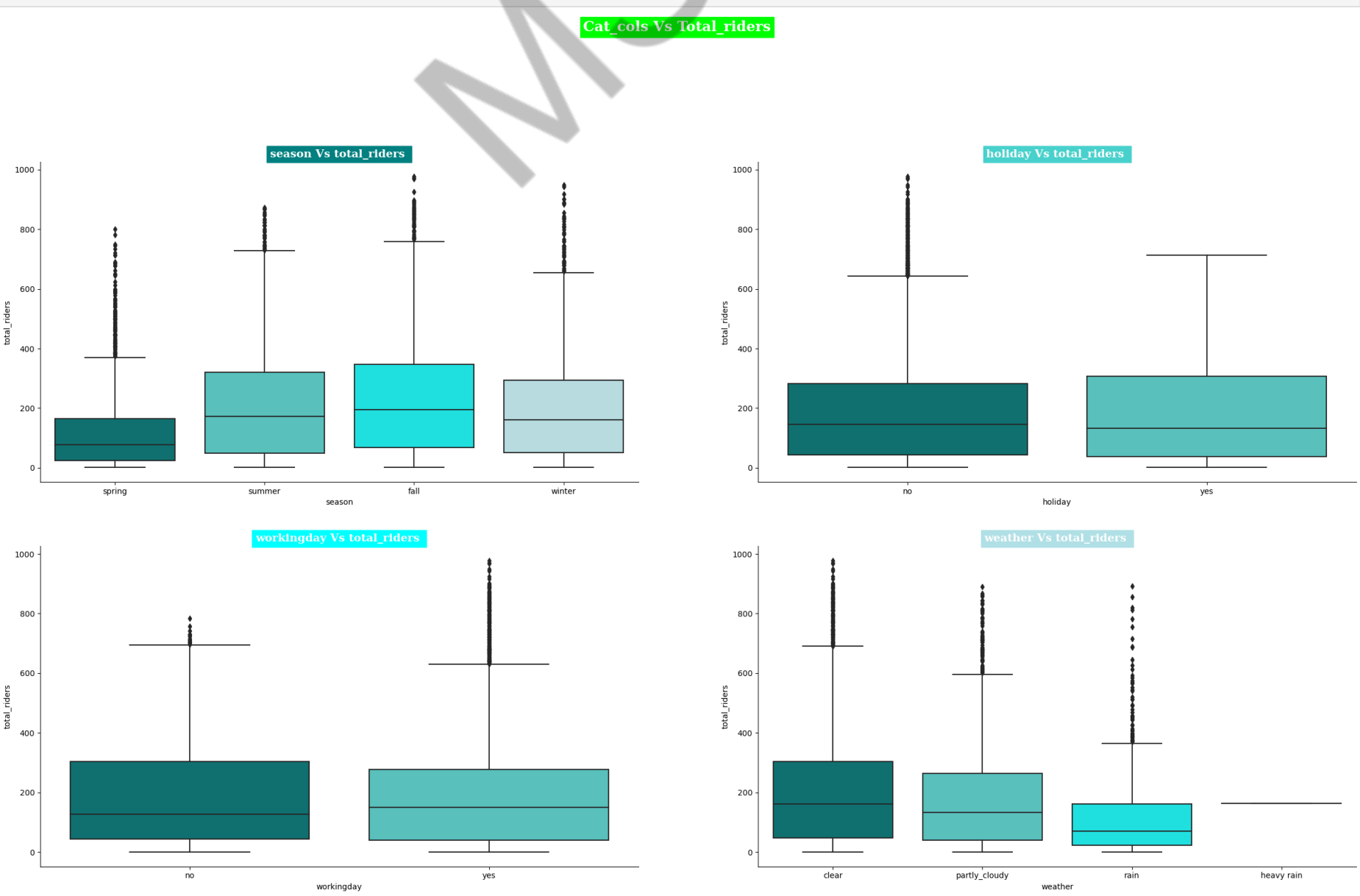
Out[84]:
```
[]
```



In [19]:
```
cat_cols = []
for _ in yd.columns:
    if yd[_].dtype=='category':
        cat_cols.append(_)
cat_cols
```

Out[19]:
```
['season', 'holiday', 'workingday', 'weather']
```

In [116…
```
plt.figure(figsize = (30,25))
plt.style.use('seaborn-v0_8-bright')
plt.suptitle(f'Cat_cols Vs Total_riders',fontsize=18,fontfamily='serif',
             fontweight='bold',backgroundcolor='lime',color='w')

for i in range(len(cat_cols)):
    plt.subplot(3,2,i+1)
    sns.boxplot(data = yd, x = cat_cols[i], y='total_riders', palette = cp)
    sns.despine()
    plt.title(f'{cat_cols[i]} Vs total_riders ',fontsize=14,fontfamily='serif',fontweight='bold',backgroundcolor=cp[i],color='w')
```



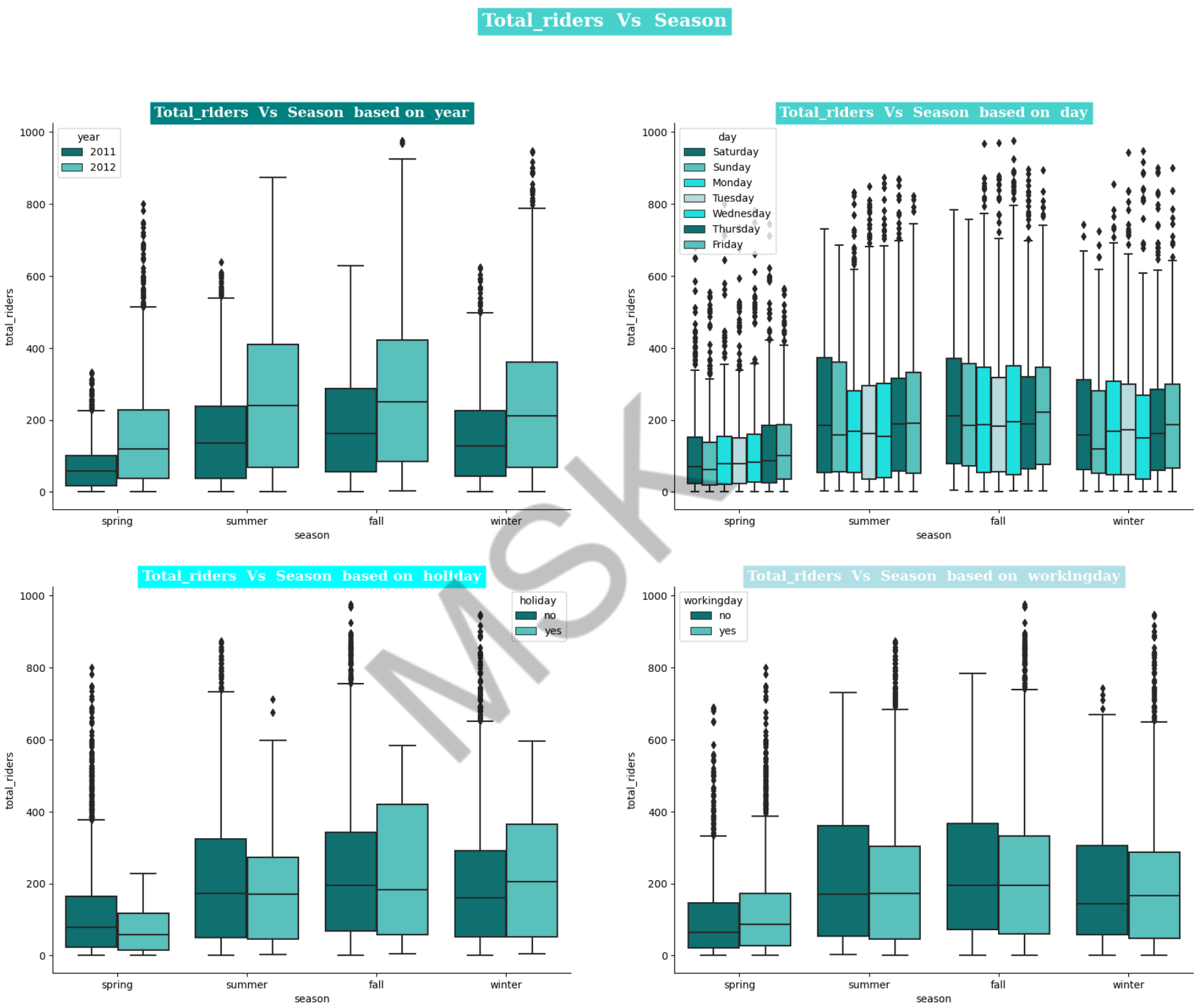
In [111…
```
yd.sample()
```

Out[111]:

	datetime	season	holiday	workingday	weather	temp	atemp	humidity	windspeed	casual	registered	total_riders	year	month	hour	
5713	2012-01-13 05:00:00	spring	no	yes	partly_cloudy	12.3	12.12	70	36.9974	0	21	21	2012	January	5	Fr

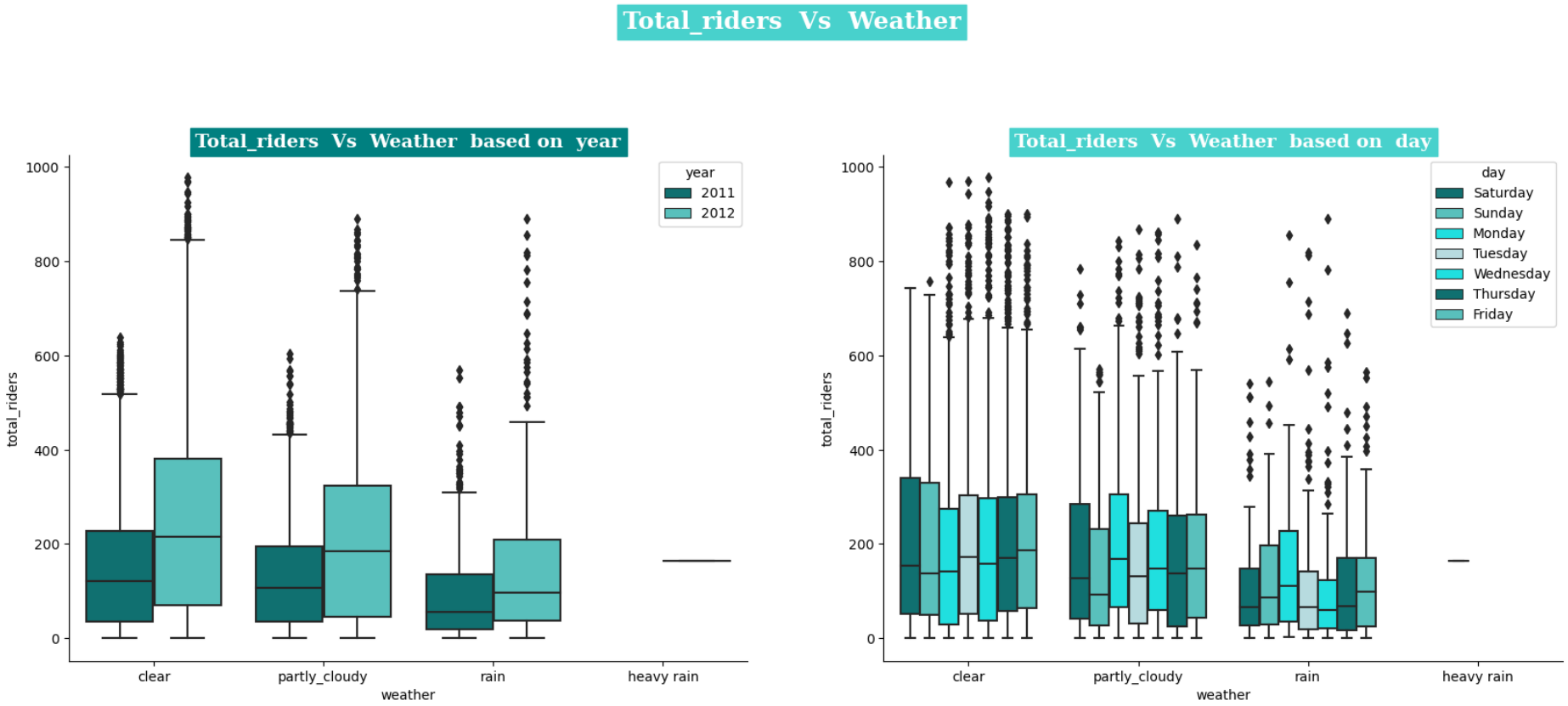
In [118…

```
cols = ['year','day','holiday','workingday']
plt.figure(figsize = (20,15))
plt.style.use('seaborn-v0_8-bright')
plt.suptitle(f'Total_riders  Vs  Season', fontsize = 18,fontfamily='serif',fontweight='bold',backgroundcolor=cp[1],color='w')
for i in range(len(cat_cols)):
    plt.subplot(2,2,i+1)
    sns.boxplot(data = yd, x = 'season', y= 'total_riders',hue = cols[i], palette=cp)
    sns.despine()
    plt.title(f'Total_riders  Vs  Season  based on  {cols[i]}', fontsize = 14,fontfamily='serif',fontweight='bold'
              ,backgroundcolor=cp[i],color='w')
```



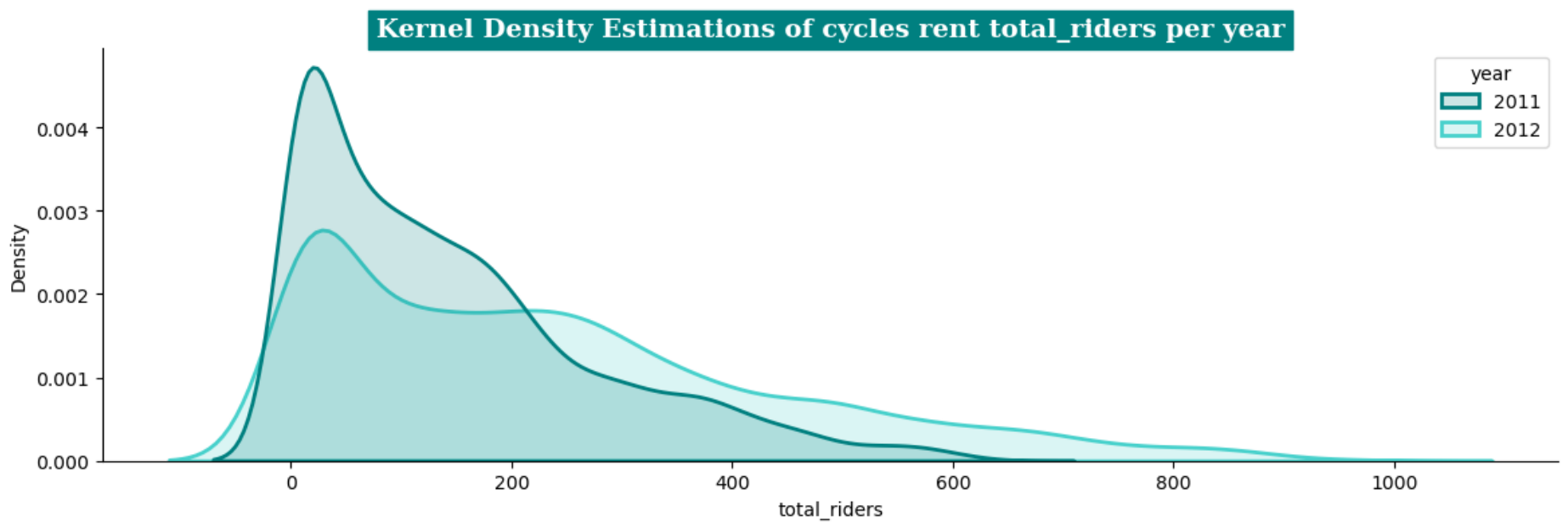
In [119…

```
cols = ['year','day','holiday','workingday']
plt.figure(figsize = (20,15))
plt.style.use('seaborn-v0_8-bright')
plt.suptitle(f'Total_riders  Vs  Weather', fontsize = 18,fontfamily='serif',fontweight='bold',backgroundcolor=cp[1],color='w')
for i in range(len(cat_cols)):
    plt.subplot(2,2,i+1)
    sns.boxplot(data = yd, x = 'weather', y= 'total_riders',hue = cols[i], palette=cp)
    sns.despine()
    plt.title(f'Total_riders  Vs  Weather  based on  {cols[i]}', fontsize = 14,fontfamily='serif',fontweight='bold'
              ,backgroundcolor=cp[i],color='w')
```



In [169…

```
plt.figure(figsize = (14,4))
sns.kdeplot(data=yd, x="total_riders", hue="year", fill=True, common_norm=False,  palette=cp, alpha=.2, linewidth=2)
plt.title("Kernel Density Estimations of cycles rent total_riders per year",fontsize = 14,fontfamily='serif',fontweight='bold'
             ,backgroundcolor=cp[i],color='w')
sns.despine()
plt.show()
```



In [21]:

```
df_num_cols = yd.select_dtypes(include=np.number)
df_num_cols.columns
```

Out[21]:

```
Index(['temp', 'atemp', 'humidity', 'windspeed', 'casual', 'registered',
       'total_riders', 'year', 'hour'],
      dtype='object')
```

■ 🔍 Outlier detection:

IQR:

In [22]:

```
numeric_cols = num_cols.columns.to_list()
numeric_cols
```

Out[22]:
```
['temp',
 'atemp',
 'humidity',
 'windspeed',
 'casual',
 'registered',
 'total_riders']
```

In [70]:
```
for i in range(len(numeric_cols)):
    data = yd[numeric_cols[i]].tolist()
    mini = np.min(data)
    Q1 = np.percentile(data, 25)
    Q2 = np.median(data)
    Q3 = np.percentile(data, 75)
    maxi = np.max(data)
    IQR = Q3 - Q1

    lo = Q1 - (1.5 * IQR)
    ho = Q3 + (1.5 * IQR)

    lower_outliers=[]
    upper_outliers=[]
    for k in data:
        if k < lo:
            lower_outliers.append(k)

        elif k > ho:
            upper_outliers.append(k)

    uo_pct = round((len(upper_outliers)*100/yd.shape[0]),2)
    lo_pct = round((len(lower_outliers)*100/yd.shape[0]),2)

    print()
    print(f"Outlier detection of {numeric_cols[i]}")
    print('.'*30)
    print("Minimum:", mini)
    print("Maximum:", maxi)
    print(f'Initial Range (with outlier) : {(maxi-mini)}')
    print("Q1:", Q1)
    print("Q2:", Q2)
    print("Q3:", Q3)
    print("IQR:",IQR)
    print(f'Final Range (without outlier) : {(ho-lo)}')
    print("Lower outliers are:", lower_outliers)
    print("Upper outliers are:", upper_outliers)
    print(f'Lower Outlier Percentage is {lo_pct}%')
    print(f'Upper Outlier Percentage is {uo_pct}%')
    print(f'Overall Outlier Percentage is {(lo_pct+uo_pct)}%')

    if len(set(lower_outliers)):
        print(f'unique Outlier points towards left of boxplot : {len(set(lower_outliers))} and they are {(set(lower_outliers))}')
    else:
        print(f'unique Outlier points towards left of boxplot : {len(set(lower_outliers))}')
    if len(set(upper_outliers)):
        print(f'unique Outlier points towards right of boxplot : {len(set(upper_outliers))} and they are {(set(upper_outliers))}'
    else:
        print(f'unique Outlier points towards right of boxplot : {len(set(upper_outliers))}')
    print()



    plt.figure(figsize=(20,7))
    plt.style.use('default')
    plt.style.use('seaborn-v0_8-bright')
    plt.suptitle(f'Customers classification Based on {numeric_cols[i]}',fontfamily='serif',fontweight='bold',fontsize=20,
             backgroundcolor='dodgerblue',color='w')

    plt.subplot(131)
    sns.violinplot(yd,x=yd[numeric_cols[i]],color=cp[i])
    plt.title(f'Violinplot of {numeric_cols[i]}',fontfamily='serif',fontweight='bold',fontsize=12,
             loc='center',backgroundcolor=cp[i],color='w')
    plt.yticks([])

    plt.subplot(132)
    bxp = sns.boxplot(yd,x=yd[numeric_cols[i]],color=cp[i],width=0.5,saturation=97,flierprops={"marker":"s"},
                      medianprops={"color": "k", "linewidth": 6})#,boxprops={"facecolor": (.3, .5, .7, .5)})
    plt.title(f'Box & Whisker plot of {numeric_cols[i]}',fontfamily='serif',fontweight='bold',fontsize=12,
             loc='center',backgroundcolor=cp[i],color='w')
    sns.despine(left=True)
    plt.yticks([])

    plt.subplot(133)
    sns.swarmplot(yd,x=yd[numeric_cols[i]],color=cp[i])
    plt.title(f'Swarmplot of {numeric_cols[i]}',fontfamily='serif',fontweight='bold',fontsize=12,
             loc='center',backgroundcolor=cp[i],color='w')
    sns.despine(left=True)
    plt.yticks([])

    tabular_data = yd[numeric_cols[i]].describe().reset_index()
    tabular_data.columns = ['Statistic', 'Value']
    display(tabular_data)

    print('-'*127)
```

```
Outlier detection of temp
.............................
Minimum: 0.82
Maximum: 41.0
Initial Range (with outlier) : 40.18
Q1: 13.94
Q2: 20.5
Q3: 26.24
IQR: 12.299999999999999
Final Range (without outlier) : 49.199999999999996
Lower outliers are: []
Upper outliers are: []
Lower Outlier Percentage is 0.0%
Upper Outlier Percentage is 0.0%
Overall Outlier Percentage is 0.0%
unique Outlier points towards left of boxplot : 0
unique Outlier points towards right of boxplot : 0
```

	Statistic	Value
0	count	10886.00000
1	mean	20.23086
2	std	7.79159
3	min	0.82000
4	25%	13.94000
5	50%	20.50000
6	75%	26.24000
7	max	41.00000

```
Outlier detection of atemp
.............................
Minimum: 0.76
Maximum: 45.455
Initial Range (with outlier) : 44.695
Q1: 16.665
Q2: 24.24
Q3: 31.06
IQR: 14.395
Final Range (without outlier) : 57.580000000000005
Lower outliers are: []
Upper outliers are: []
Lower Outlier Percentage is 0.0%
Upper Outlier Percentage is 0.0%
Overall Outlier Percentage is 0.0%
unique Outlier points towards left of boxplot : 0
unique Outlier points towards right of boxplot : 0
```

	Statistic	Value
0	count	10886.000000
1	mean	23.655084
2	std	8.474601
3	min	0.760000
4	25%	16.665000
5	50%	24.240000
6	75%	31.060000
7	max	45.455000

```
Outlier detection of humidity
.............................
Minimum: 0
Maximum: 100
Initial Range (with outlier) : 100
Q1: 47.0
Q2: 62.0
Q3: 77.0
IQR: 30.0
Final Range (without outlier) : 120.0
Lower outliers are: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
Upper outliers are: []
Lower Outlier Percentage is 0.2%
Upper Outlier Percentage is 0.0%
Overall Outlier Percentage is 0.2%
unique Outlier points towards left of boxplot : 1 and they are {0}
unique Outlier points towards right of boxplot : 0
```

	Statistic	Value
0	count	10886.000000
1	mean	61.886460
2	std	19.245033
3	min	0.000000
4	25%	47.000000
5	50%	62.000000
6	75%	77.000000
7	max	100.000000

```
Outlier detection of windspeed
.............................
Minimum: 0.0
Maximum: 56.9969
Initial Range (with outlier) : 56.9969
Q1: 7.0015
Q2: 12.998
Q3: 16.9979
IQR: 9.996400000000001
Final Range (without outlier) : 39.985600000000005
Lower outliers are: []
Upper outliers are: [32.9975, 36.9974, 35.0008, 35.0008, 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, 40.9973, 40.9973, 32.9975, 35.0008, 43.9989, 32.9975, 5
1.9987, 46.0022, 35.0008, 39.0007, 32.9975, 32.9975, 35.0008, 39.0007, 43.9989, 39.0007, 39.0007, 32.9975, 40.9973, 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, 35.00
08, 39.0007, 32.9975, 32.9975, 32.9975, 32.9975, 32.9975, 36.9974, 35.0008, 40.9973, 32.9975, 36.9974, 32.9975, 39.0007, 32.997
5, 32.9975, 32.9975, 35.0008, 32.9975, 35.0008, 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, 39.0007, 32.9975, 32.9975, 32.9975, 35.0008, 32.9975, 56.9969, 5
6.9969, 32.9975, 32.9975, 43.0006, 32.9975, 32.9975, 35.0008, 35.0008, 39.0007, 40.9973, 36.9974, 32.9975, 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, 35.00
08, 39.0007, 35.0008, 32.9975, 36.9974, 40.9973, 32.9975, 35.0008, 36.9974, 39.0007, 32.9975, 32.9975, 36.9974, 32.9975, 32.997
5, 35.0008, 35.0008, 35.0008, 32.9975, 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, 35.0008, 32.9975, 36.9974, 40.9973, 32.9975, 43.9989, 39.0007, 32.9975, 4
3.0006, 43.0006, 46.0022, 43.0006, 43.9989, 36.9974, 32.9975, 35.0008, 35.0008, 35.0008, 32.9975, 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, 35.0008, 35.00
08, 32.9975, 32.9975, 35.0008, 39.0007, 32.9975, 32.9975, 32.9975, 32.9975, 32.9975, 39.0007, 39.0007, 35.0008, 35.0008, 32.997
5, 35.0008, 35.0008, 36.9974, 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]
Lower Outlier Percentage is 0.0%
Upper Outlier Percentage is 2.09%
Overall Outlier Percentage is 2.09%
unique Outlier points towards left of boxplot : 0
unique Outlier points towards right of boxplot : 12 and they are {32.9975, 35.0008, 36.9974, 39.0007, 40.9973, 43.9989, 43.0006,
46.0022, 47.9988, 50.0021, 51.9987, 56.9969}
```


	Statistic	Value
0	count	10886.000000
1	mean	12.799395
2	std	8.164537
3	min	0.000000
4	25%	7.001500
5	50%	12.998000
6	75%	16.997900
7	max	56.996900

Outlier detection of casual
....
Minimum: 0
Maximum: 367
Initial Range (with outlier) : 367
Q1: 4.0
Q2: 17.0
Q3: 49.0
IQR: 45.0
Final Range (without outlier) : 180.0
Lower outliers are: []
Upper outliers are: [144, 149, 124, 126, 174, 168, 170, 175, 138, 139, 166, 219, 240, 174, 147, 148, 128, 121, 148, 156, 135, 11
9, 167, 181, 170, 179, 161, 143, 123, 132, 129, 196, 143, 148, 119, 138, 142, 139, 166, 126, 128, 122, 120, 120, 195, 183, 206,
158, 137, 120, 150, 188, 193, 180, 168, 142, 127, 121, 123, 154, 161, 161, 138, 126, 124, 150, 148, 142, 169, 147, 131, 120, 13
1, 176, 142, 134, 162, 135, 138, 153, 193, 210, 118, 141, 141, 149, 124, 146, 121, 159, 144, 142, 178, 177, 168, 206, 179, 193,
156, 136, 131, 215, 198, 248, 225, 194, 195, 181, 142, 166, 177, 237, 242, 235, 224, 236, 240, 222, 170, 195, 195, 145, 167, 15
8, 182, 171, 160, 132, 128, 129, 133, 139, 122, 142, 128, 146, 141, 120, 171, 187, 221, 201, 205, 183, 234, 185, 164, 127, 147,
177, 200, 168, 129, 129, 133, 147, 145, 138, 130, 176, 176, 130, 155, 161, 118, 130, 118, 122, 175, 150, 148, 123, 144, 126, 12
8, 133, 125, 153, 204, 187, 186, 170, 160, 147, 150, 174, 214, 245, 205, 218, 196, 204, 187, 178, 141, 154, 145, 144, 131, 129,
217, 170, 185, 119, 120, 144, 119, 120, 134, 180, 152, 119, 124, 123, 148, 141, 135, 162, 127, 132, 134, 120, 176, 191, 256, 25
1, 262, 221, 174, 135, 155, 189, 212, 225, 272, 198, 223, 208, 186, 139, 164, 146, 147, 130, 154, 125, 153, 195, 171, 242, 166,
179, 177, 121, 121, 159, 167, 169, 204, 182, 177, 152, 137, 148, 159, 141, 128, 120, 142, 128, 191, 165, 137, 118, 121, 120, 12
0, 149, 156, 132, 133, 128, 165, 183, 229, 186, 142, 151, 145, 167, 117, 132, 146, 176, 212, 201, 208, 199, 133, 140, 140, 226,
286, 286, 352, 357, 367, 291, 221, 155, 139, 129, 222, 198, 218, 240, 229, 233, 165, 118, 120, 132, 180, 187, 190, 283, 295, 23
6, 232, 153, 127, 132, 128, 120, 120, 145, 123, 139, 143, 145, 173, 240, 218, 184, 172, 187, 251, 320, 355, 326, 321, 354, 299,
227, 170, 195, 198, 229, 254, 260, 232, 185, 151, 125, 117, 139, 207, 274, 308, 288, 311, 253, 251, 197, 163, 186, 227, 275, 29
8, 282, 266, 286, 262, 184, 148, 221, 220, 217, 187, 241, 230, 204, 156, 151, 144, 207, 197, 226, 229, 181, 195, 131, 117, 140,
128, 132, 157, 206, 293, 257, 269, 254, 233, 164, 166, 128, 150, 189, 255, 228, 190, 225, 191, 129, 133, 134, 131, 151, 124, 17
7, 235, 276, 332, 361, 356, 331, 279, 254, 203, 118, 145, 179, 250, 279, 259, 297, 275, 248, 171, 185, 139, 127, 199, 276, 265,
267, 236, 226, 192, 153, 129, 128, 147, 117, 168, 218, 196, 236, 239, 218, 219, 238, 175, 128, 125, 148, 184, 173, 204, 186, 15
9, 173, 184, 164, 123, 117, 121, 121, 132, 125, 121, 139, 213, 254, 293, 264, 295, 238, 244, 256, 184, 123, 148, 196, 260, 267,
255, 203, 240, 232, 203, 146, 121, 143, 164, 164, 177, 190, 155, 163, 161, 159, 177, 237, 222, 175, 129, 128, 122, 124, 120, 11
8, 125, 120, 138, 123, 143, 181, 219, 269, 243, 226, 199, 156, 141, 170, 153, 171, 162, 182, 152, 122, 155, 222, 195, 183, 205,
197, 179, 169, 133, 117, 155, 161, 208, 161, 164, 143, 122, 125, 117, 119, 125, 203, 214, 228, 248, 246, 227, 220, 167, 163, 17
8, 213, 235, 213, 186, 164, 148, 124, 130, 119, 128, 127, 144, 166, 180, 266, 289, 242, 250, 287, 256, 225, 164, 138, 128, 219,
187, 237, 240, 257, 209, 205, 184, 157, 158, 226, 248, 263, 249, 179, 222, 219, 227, 140, 135, 187, 191, 209, 163, 168, 165, 18
1, 145, 173, 220, 233, 260, 175, 172, 194, 247, 238, 232, 284, 228, 213, 156, 134, 127, 168, 119, 137, 119, 228, 287, 327, 325,
312, 350, 295, 232, 169, 160, 183, 188, 240, 225, 215, 194, 177, 120, 129, 161, 117, 150, 153, 158, 130, 198, 258, 362, 310, 26
9, 279, 317, 268, 183, 121, 138, 131, 192, 202, 235, 243, 251, 294, 193, 174, 150, 156, 200, 218, 249, 213, 203, 193, 134, 122,
131, 136, 117, 134, 150, 122, 149, 149, 119, 143, 258, 268, 280, 216, 227, 183, 133, 180, 195, 262, 292, 304, 260, 151, 134, 11
7, 179, 175, 175, 144, 125, 119, 123, 142, 122, 148, 164, 167, 139]
Lower Outlier Percentage is 0.0%
Upper Outlier Percentage is 6.88%
Overall Outlier Percentage is 6.88%
unique Outlier points towards left of boxplot : 0
unique Outlier points towards right of boxplot : 192 and they are {117, 118, 119, 120, 121, 122, 123, 124, 125, 126, 127, 128, 1
29, 130, 131, 132, 133, 134, 135, 136, 137, 138, 139, 140, 141, 142, 143, 144, 145, 146, 147, 148, 149, 150, 151, 152, 153, 154,
155, 156, 157, 158, 159, 160, 161, 162, 163, 164, 165, 166, 167, 168, 169, 170, 171, 172, 173, 174, 175, 176, 177, 178, 179, 18
0, 181, 182, 183, 184, 185, 186, 187, 188, 189, 190, 191, 192, 193, 194, 195, 196, 197, 198, 199, 200, 201, 202, 203, 204, 205,
206, 207, 208, 209, 210, 212, 213, 214, 215, 216, 217, 218, 219, 220, 221, 222, 223, 224, 225, 226, 227, 228, 229, 230, 232, 23
3, 234, 235, 236, 237, 238, 239, 240, 241, 242, 243, 244, 245, 246, 247, 248, 249, 250, 251, 253, 254, 255, 256, 257, 258, 259,
260, 262, 263, 264, 265, 266, 267, 268, 269, 272, 274, 275, 276, 279, 280, 282, 283, 284, 286, 287, 288, 289, 291, 292, 293, 29
4, 295, 297, 298, 299, 304, 308, 310, 311, 312, 317, 320, 321, 325, 326, 327, 331, 332, 350, 352, 354, 355, 356, 357, 361, 362,
367}

	Statistic	Value
0	count	10886.000000
1	mean	36.021955
2	std	49.960477
3	min	0.000000
4	25%	4.000000
5	50%	17.000000
6	75%	49.000000
7	max	367.000000

Outlier detection of registered
....
Minimum: 0
Maximum: 886
Initial Range (with outlier) : 886
Q1: 36.0
Q2: 118.0
Q3: 222.0
IQR: 186.0
Final Range (without outlier) : 744.0
Lower outliers are: []
Upper outliers are: [539, 532, 540, 521, 516, 529, 510, 555, 527, 517, 517, 514, 513, 525, 516, 506, 567, 515, 515, 507, 512, 503, 530, 525, 536, 505, 534, 540, 553, 512, 523, 505, 508, 514, 522, 508, 510, 504, 508, 516, 579, 551, 533, 554, 554, 509, 573, 545, 545, 547, 557, 623, 571, 614, 638, 628, 642, 647, 602, 634, 648, 534, 563, 615, 681, 553, 601, 543, 577, 634, 661, 653, 645, 646, 610, 677, 618, 595, 565, 586, 670, 656, 626, 581, 527, 546, 604, 596, 547, 539, 621, 564, 549, 601, 589, 631, 673, 670, 551, 651, 663, 677, 516, 538, 616, 662, 640, 659, 770, 608, 617, 642, 584, 547, 564, 608, 667, 602, 605, 640, 641, 594, 629, 603, 648, 661, 579, 677, 518, 603, 665, 617, 769, 749, 605, 719, 734, 696, 688, 570, 675, 504, 643, 733, 719, 554, 540, 680, 764, 679, 531, 637, 652, 688, 506, 661, 778, 703, 537, 663, 679, 576, 613, 715, 726, 507, 598, 625, 596, 672, 782, 640, 548, 682, 750, 716, 609, 698, 572, 669, 595, 633, 725, 704, 548, 665, 658, 510, 604, 620, 542, 514, 575, 511, 557, 741, 790, 543, 615, 769, 749, 644, 740, 735, 560, 653, 704, 739, 532, 660, 697, 523, 619, 669, 697, 503, 618, 675, 712, 624, 580, 625, 678, 684, 649, 734, 786, 564, 718, 775, 681, 509, 636, 646, 573, 578, 746, 743, 531, 664, 743, 711, 506, 689, 751, 745, 567, 670, 699, 733, 533, 619, 552, 648, 716, 709, 591, 734, 757, 533, 642, 749, 768, 525, 649, 767, 723, 558, 625, 647, 561, 617, 746, 745, 502, 692, 780, 757, 598, 622, 741, 761, 504, 690, 684, 697, 510, 697, 744, 857, 562, 572, 702, 802, 767, 540, 533, 727, 811, 886, 557, 549, 725, 787, 744, 594, 715, 757, 692, 647, 740, 708, 640, 529, 758, 812, 807, 539, 744, 791, 571, 677, 639, 504, 781, 833, 756, 544, 711, 789, 812, 749, 742, 655, 698, 733, 758, 806, 857, 787, 534, 773, 737, 628, 662, 711, 706, 566, 713, 670, 540, 505, 800, 839, 767, 779, 766, 794, 803, 788, 720, 668, 652, 628, 656, 581, 629, 604, 523, 568, 597, 583, 652, 556, 668, 593, 530, 575, 514, 664, 564, 543, 625, 541, 563, 694, 539, 650, 586, 559, 712, 665, 666, 700, 700, 693, 737, 576, 580, 646, 586, 542, 539, 575, 578, 544, 708, 580, 523, 670, 540, 515, 655, 507, 623, 582, 582, 652, 533, 512, 665, 536, 546]
Lower Outlier Percentage is 0.0%
Upper Outlier Percentage is 3.89%
Overall Outlier Percentage is 3.89%
unique Outlier points towards left of boxplot : 0
unique Outlier points towards right of boxplot : 232 and they are {512, 513, 514, 515, 516, 517, 518, 521, 522, 523, 525, 527, 529, 530, 531, 532, 533, 534, 536, 537, 538, 539, 540, 541, 542, 543, 544, 545, 546, 547, 548, 549, 551, 552, 553, 554, 555, 556, 557, 558, 559, 560, 561, 562, 563, 564, 565, 566, 567, 568, 570, 571, 572, 573, 575, 576, 577, 578, 579, 580, 581, 582, 583, 584, 586, 589, 591, 593, 594, 595, 596, 597, 598, 601, 602, 603, 604, 605, 608, 609, 610, 613, 614, 615, 616, 617, 618, 619, 620, 621, 622, 623, 624, 625, 626, 628, 629, 631, 633, 634, 636, 637, 638, 639, 640, 641, 642, 643, 644, 645, 646, 647, 648, 649, 650, 651, 652, 653, 655, 656, 658, 659, 660, 661, 662, 663, 664, 665, 666, 667, 668, 669, 670, 672, 673, 675, 677, 678, 679, 680, 681, 682, 684, 688, 689, 690, 692, 693, 694, 696, 697, 698, 699, 700, 702, 703, 704, 706, 708, 709, 711, 712, 713, 715, 716, 718, 719, 720, 723, 725, 726, 727, 733, 734, 735, 737, 739, 740, 741, 742, 743, 744, 745, 746, 749, 750, 751, 756, 757, 758, 761, 764, 766, 767, 768, 769, 770, 773, 775, 778, 779, 780, 781, 782, 786, 787, 788, 789, 790, 791, 794, 800, 802, 803, 806, 807, 811, 812, 833, 839, 857, 886, 502, 503, 504, 505, 506, 507, 508, 509, 510, 511}

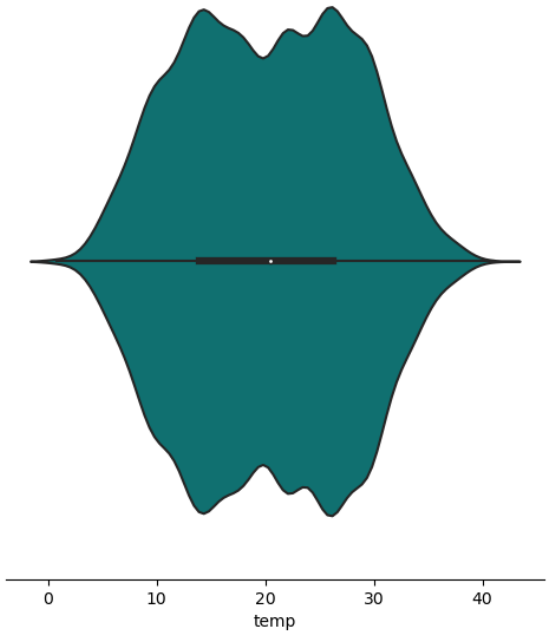
	Statistic	Value
0	count	10886.000000
1	mean	155.552177
2	std	151.039033
3	min	0.000000
4	25%	36.000000
5	50%	118.000000
6	75%	222.000000
7	max	886.000000

Outlier detection of total_riders
....
Minimum: 1
Maximum: 977
Initial Range (with outlier) : 976
Q1: 42.0
Q2: 145.0
Q3: 284.0
IQR: 242.0
Final Range (without outlier) : 968.0
Lower outliers are: []
Upper outliers are: [712, 676, 734, 662, 782, 749, 713, 746, 651, 686, 690, 679, 685, 648, 721, 801, 729, 757, 800, 684, 744, 759, 822, 698, 655, 692, 744, 704, 656, 738, 671, 678, 678, 660, 658, 681, 712, 676, 673, 781, 775, 677, 748, 776, 681, 743, 666, 729, 813, 704, 706, 769, 680, 717, 710, 705, 732, 770, 779, 659, 678, 733, 650, 873, 846, 852, 868, 745, 812, 669, 704, 730, 672, 694, 668, 679, 702, 684, 686, 678, 662, 665, 834, 822, 710, 850, 790, 668, 724, 782, 681, 869, 813, 700, 793, 723, 651, 800, 831, 681, 653, 713, 857, 744, 671, 719, 867, 823, 653, 823, 693, 723, 673, 811, 795, 747, 730, 722, 689, 849, 872, 649, 872, 819, 674, 830, 814, 702, 795, 825, 713, 835, 667, 755, 794, 661, 770, 772, 679, 657, 771, 777, 681, 837, 891, 652, 739, 865, 767, 668, 741, 671, 858, 843, 705, 868, 814, 737, 858, 862, 686, 698, 810, 811, 730, 673, 818, 812, 812, 854, 682, 851, 848, 649, 682, 897, 832, 677, 668, 791, 669, 654, 856, 839, 725, 863, 839, 662, 808, 835, 719, 772, 792, 694, 668, 757, 729, 696, 701, 671, 730, 871, 968, 750, 970, 877, 770, 925, 977, 758, 884, 852, 674, 766, 894, 808, 706, 704, 715, 654, 783, 729, 656, 694, 683, 842, 774, 672, 797, 886, 892, 784, 856, 715, 687, 809, 917, 810, 738, 901, 887, 785, 900, 761, 743, 710, 659, 713, 806, 784, 839, 948, 844, 798, 827, 692, 743, 745, 837, 670, 737, 766, 835, 943, 838, 817, 888, 884, 834, 890, 788, 680, 689, 678, 687, 648, 668, 693, 654, 651, 686, 654, 724, 653, 691, 711, 663, 731, 708, 692, 721, 743, 731, 759, 659, 724, 688, 679, 662, 678]
Lower Outlier Percentage is 0.0%
Upper Outlier Percentage is 2.76%
Overall Outlier Percentage is 2.76%
unique Outlier points towards left of boxplot : 0
unique Outlier points towards right of boxplot : 180 and they are {648, 649, 650, 651, 652, 653, 654, 655, 656, 657, 658, 659, 660, 661, 662, 663, 665, 666, 667, 668, 669, 670, 671, 672, 673, 674, 676, 677, 678, 679, 680, 681, 682, 683, 684, 685, 686, 687, 688, 689, 690, 691, 692, 693, 694, 696, 698, 700, 701, 702, 704, 705, 706, 708, 710, 711, 712, 713, 715, 717, 719, 721, 722, 723, 724, 725, 729, 730, 731, 732, 733, 734, 737, 738, 739, 741, 743, 744, 745, 746, 747, 748, 749, 750, 755, 757, 758, 759, 761, 766, 767, 769, 770, 771, 772, 774, 775, 776, 777, 779, 781, 782, 783, 784, 785, 788, 790, 791, 792, 793, 794, 795, 797, 798, 800, 801, 806, 808, 809, 810, 811, 812, 813, 814, 817, 818, 819, 822, 823, 825, 827, 830, 831, 832, 834, 835, 837, 838, 839, 842, 843, 844, 846, 848, 849, 850, 851, 852, 854, 856, 857, 858, 862, 863, 865, 867, 868, 869, 871, 872, 873, 877, 884, 886, 887, 888, 890, 891, 892, 894, 897, 900, 901, 917, 925, 943, 948, 968, 970, 977}

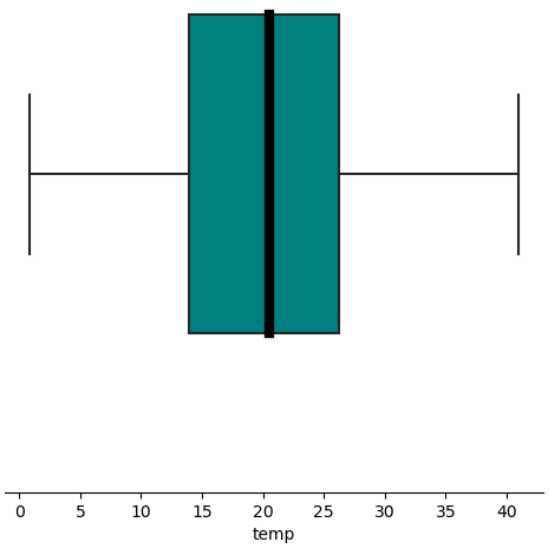
	Statistic	Value
0	count	10886.000000
1	mean	191.574132
2	std	181.144454
3	min	1.000000
4	25%	42.000000
5	50%	145.000000
6	75%	284.000000
7	max	977.000000

Customers classification Based on temp

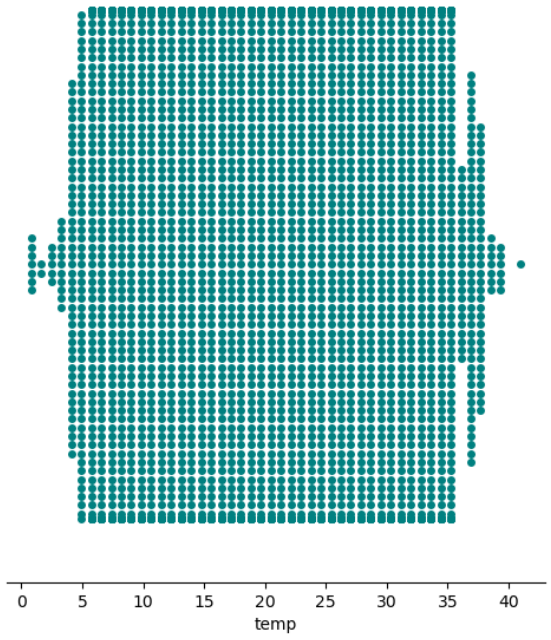
Violinplot of temp



Box & Whisker plot of temp

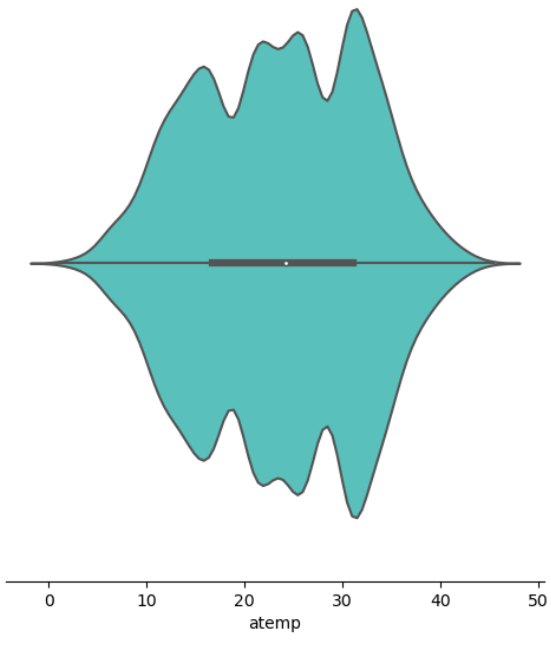


Swarmplot of temp

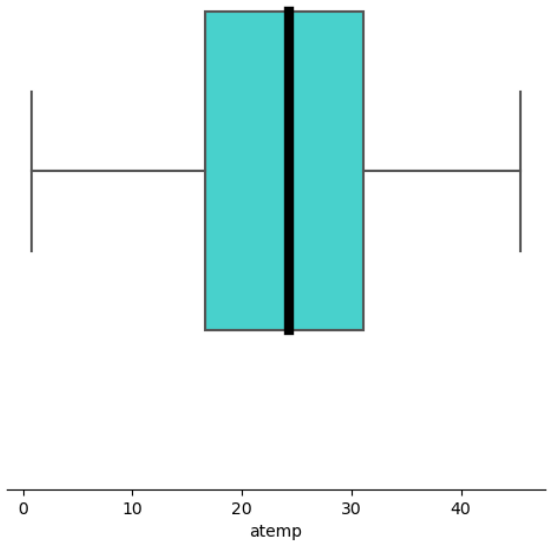


Customers classification Based on atemp

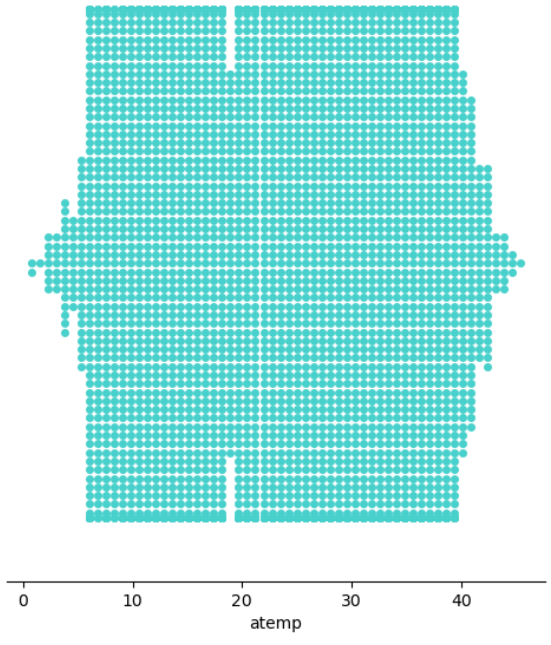
Violinplot of atemp



Box & Whisker plot of atemp



Swarmplot of atemp



Customers classification Based on humidity

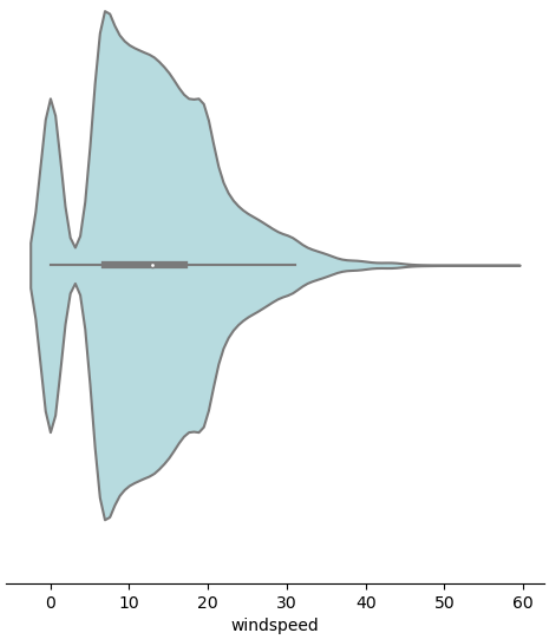
Violinplot of humidity

Box & Whisker plot of humidity

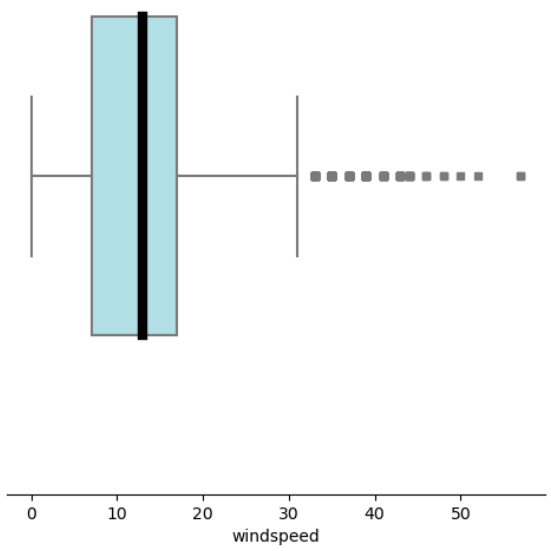
Swarmplot of humidity

Customers classification Based on windspeed

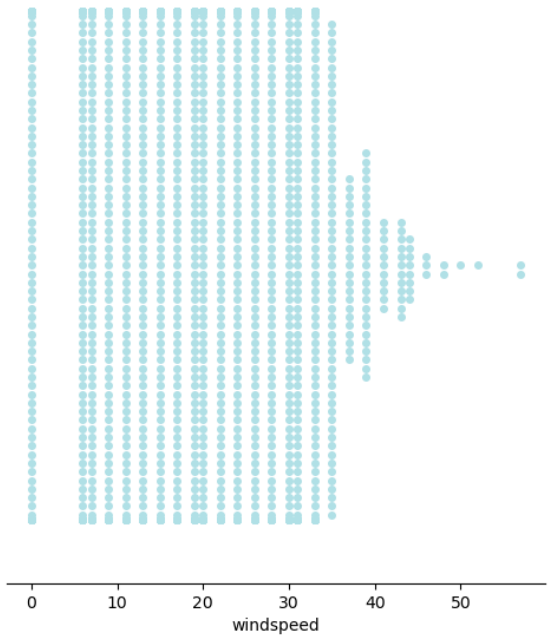
Violinplot of windspeed



Box & Whisker plot of windspeed

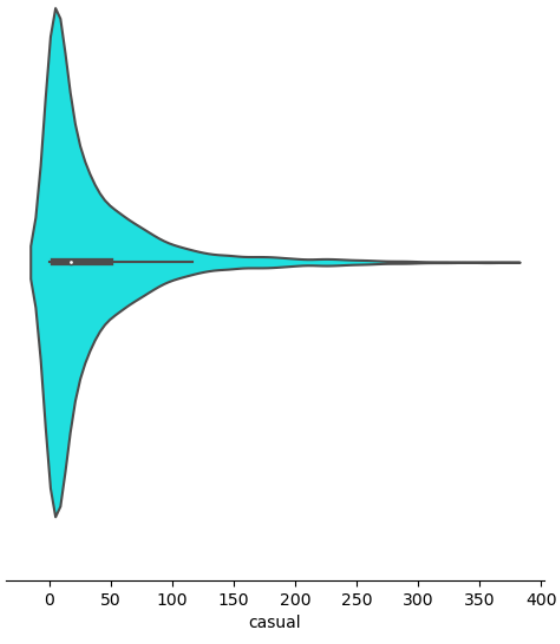


Swarmplot of windspeed

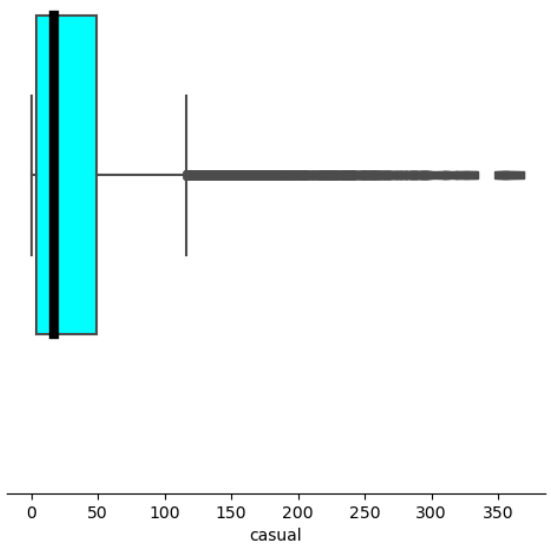


Customers classification Based on casual

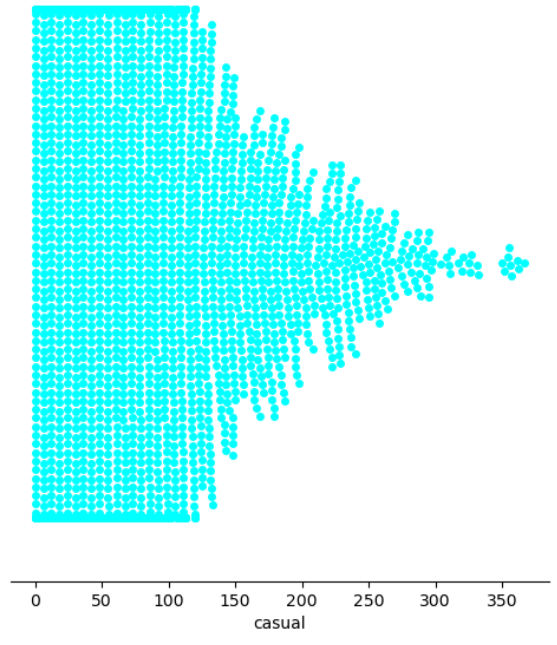
Violinplot of casual



Box & Whisker plot of casual



Swarmplot of casual



Customers classification Based on registered

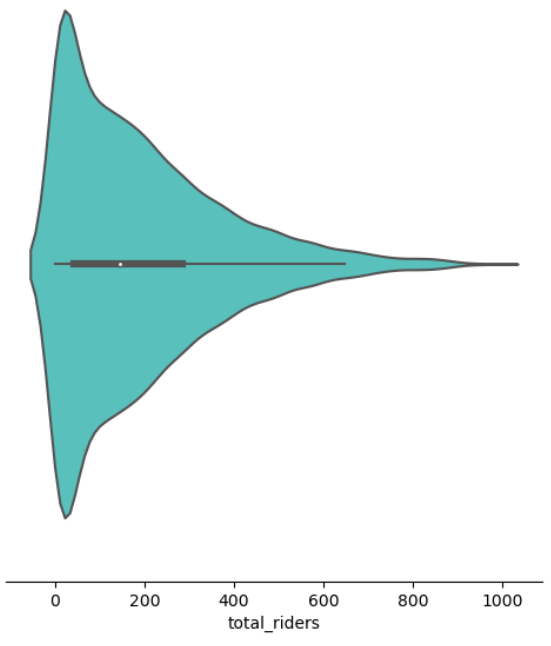
Violinplot of registered

Box & Whisker plot of registered

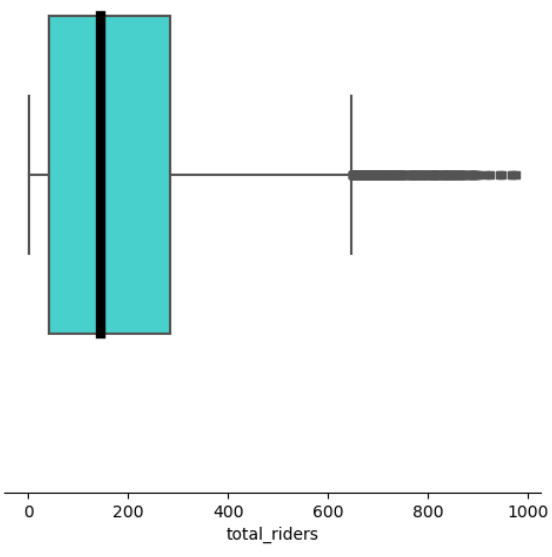
Swarmplot of registered

Customers classification Based on total_riders

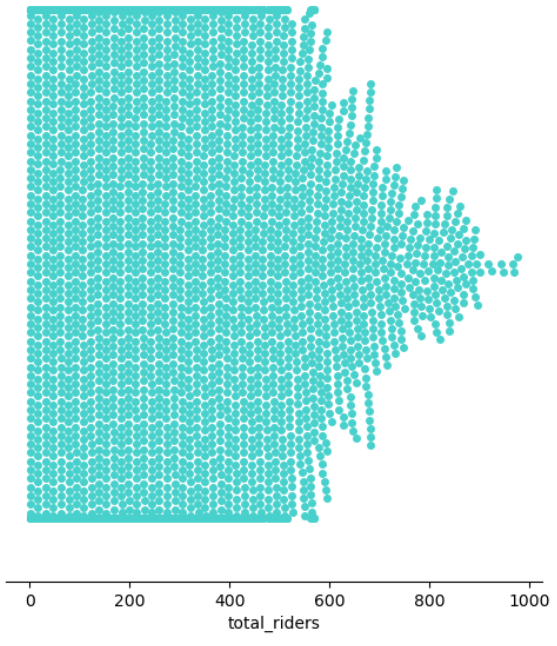
Violinplot of total_riders



Box & Whisker plot of total_riders



Swarmplot of total_riders



Method-2

In [76]:
```
df_num_cols
```

Out[76]:

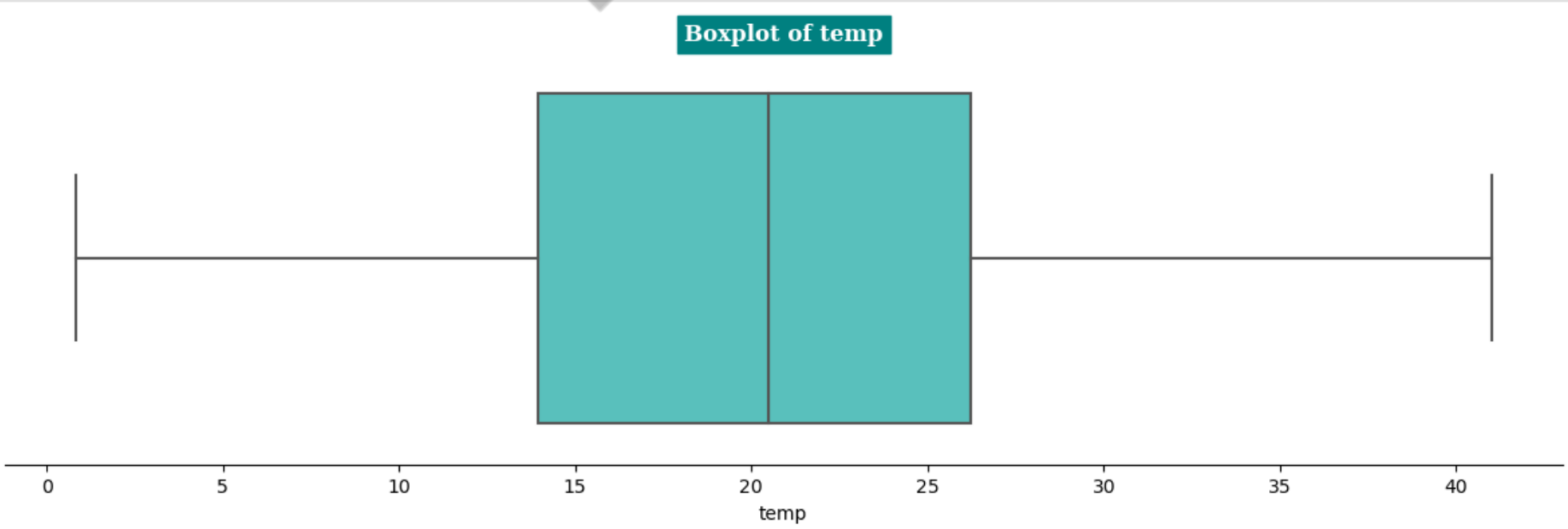
	temp	atemp	humidity	windspeed	casual	registered	total_riders	year	hour
0	9.84	14.395	81	0.0000	3	13	16	2011	0
1	9.02	13.635	80	0.0000	8	32	40	2011	1
2	9.02	13.635	80	0.0000	5	27	32	2011	2
3	9.84	14.395	75	0.0000	3	10	13	2011	3
4	9.84	14.395	75	0.0000	0	1	1	2011	4
...	...	...	...	...	...	...	...	...	...
10881	15.58	19.695	50	26.0027	7	329	336	2012	19
10882	14.76	17.425	57	15.0013	10	231	241	2012	20
10883	13.94	15.910	61	15.0013	4	164	168	2012	21
10884	13.94	17.425	61	6.0032	12	117	129	2012	22
10885	13.12	16.665	66	8.9981	4	84	88	2012	23

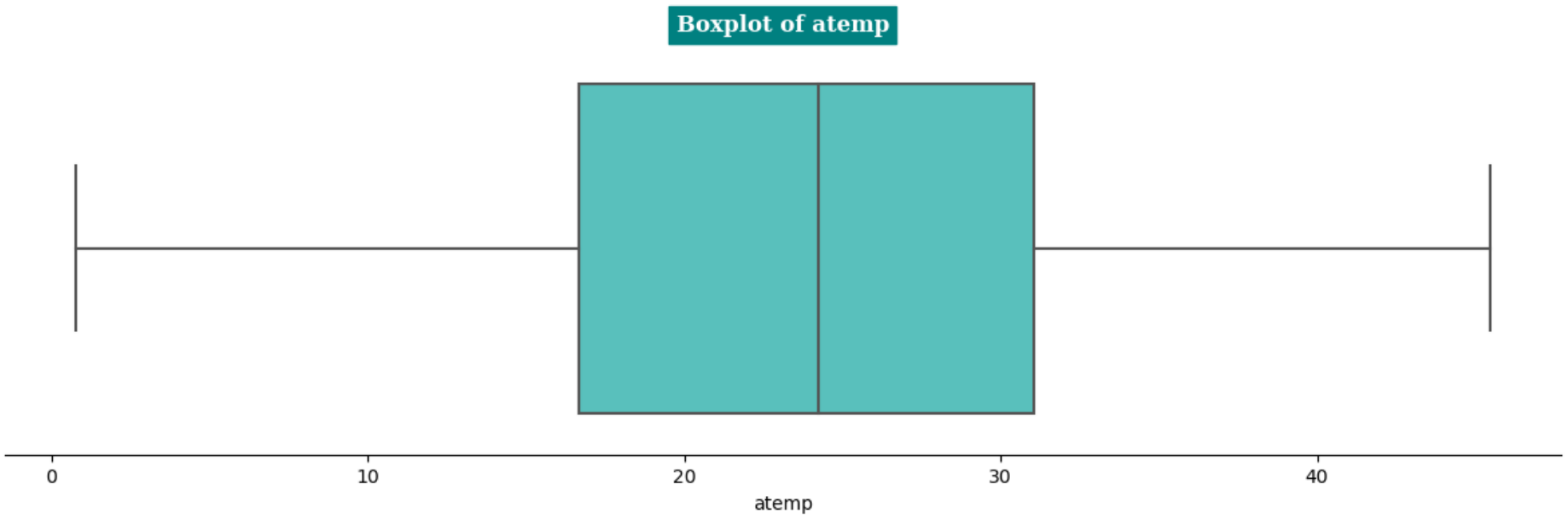
10886 rows × 9 columns

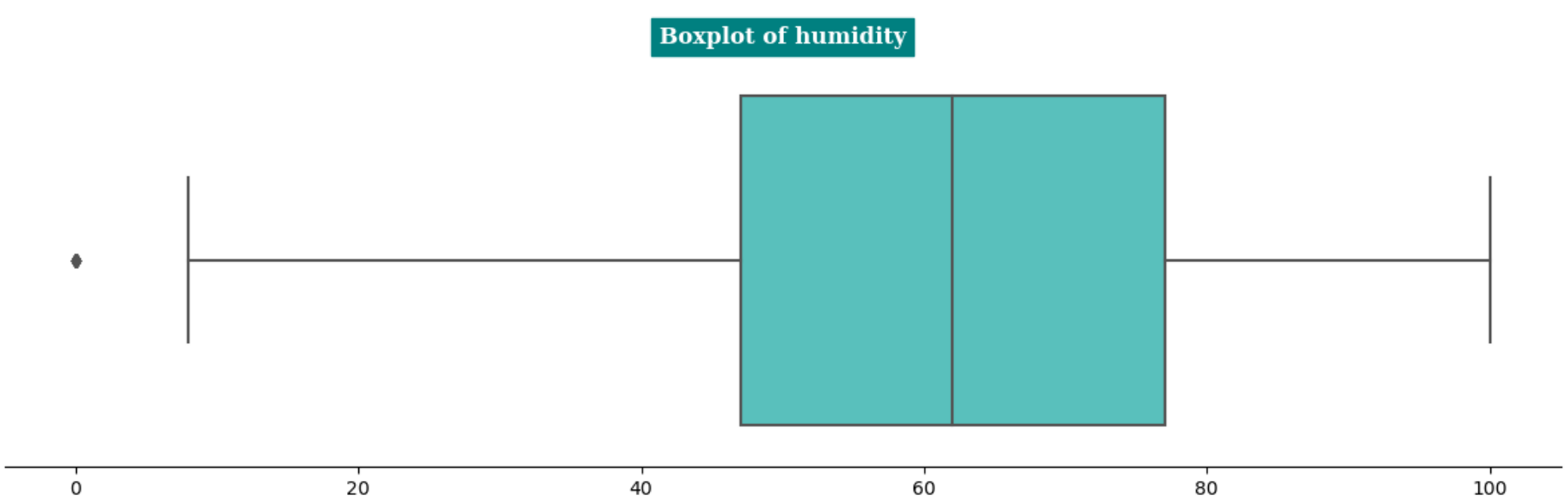
In [72]:
```
# obtain the first quartile
Q1 = df_num_cols.quantile(0.25)

# obtain the third quartile
Q3 = df_num_cols.quantile(0.75)

# obtain the IQR
IQR = Q3 - Q1

# print the IQR
print(IQR)
```

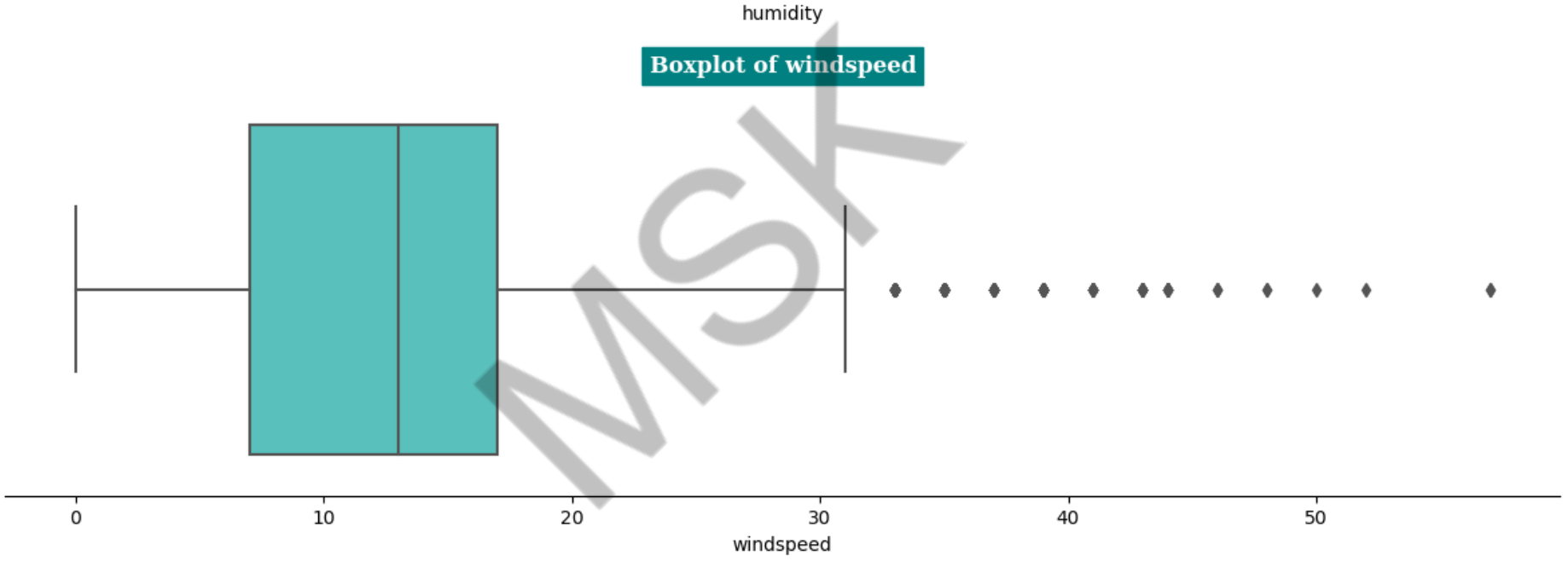
```
temp             12.3000
atemp            14.3950
humidity         30.0000
windspeed         9.9964
casual           45.0000
registered      186.0000
total_riders    242.0000
year              1.0000
hour             12.0000
dtype: float64
```

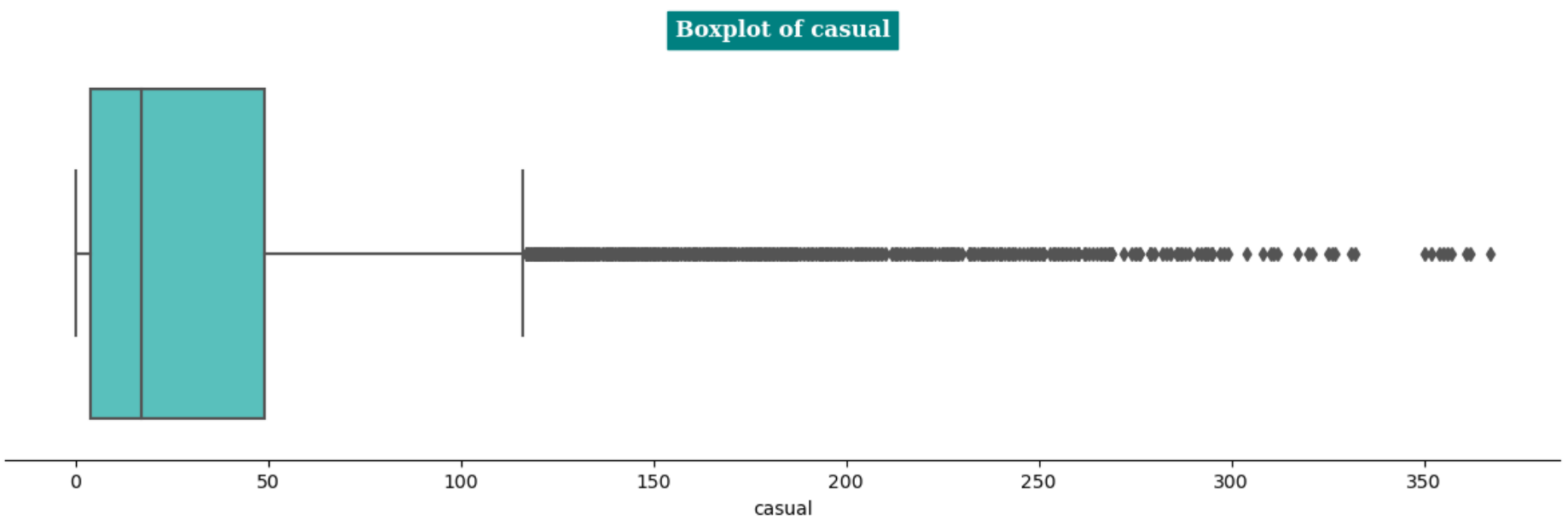
In [81]:
```
for col in enumerate(df_num_cols):
    plt.figure(figsize=(15,4))
    sns.boxplot(x=col[1], data=df_num_cols,color=cp[1])
    sns.despine(left=True)
    plt.yticks([])
    plt.title(f'Boxplot of {col[1]}',fontfamily='serif',fontweight='bold',fontsize=12,backgroundcolor=cp[0],color='w')
    plt.show()
```

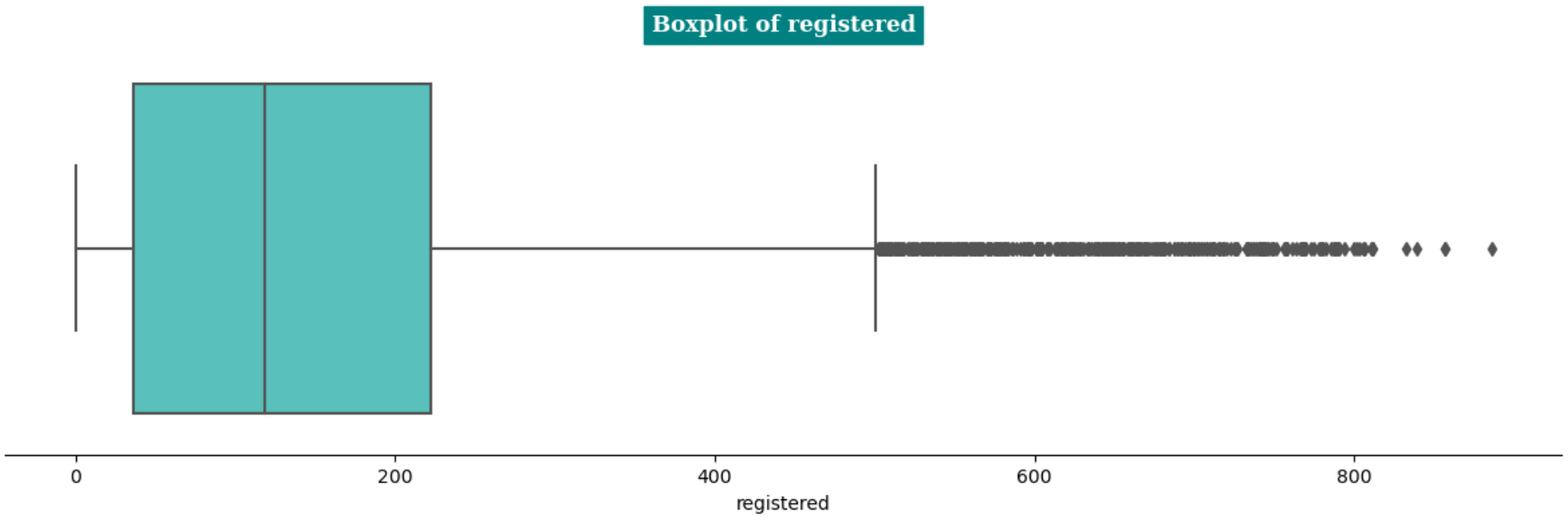


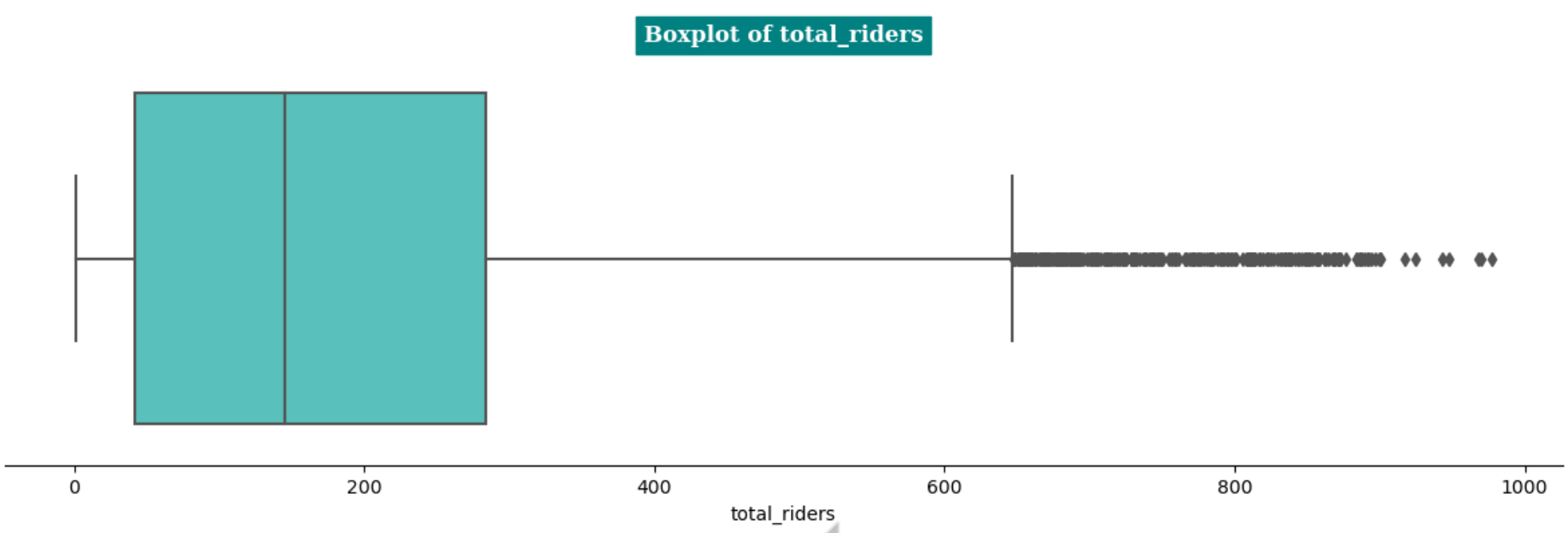

Boxplot of atemp
0
10
20
30
40
atemp

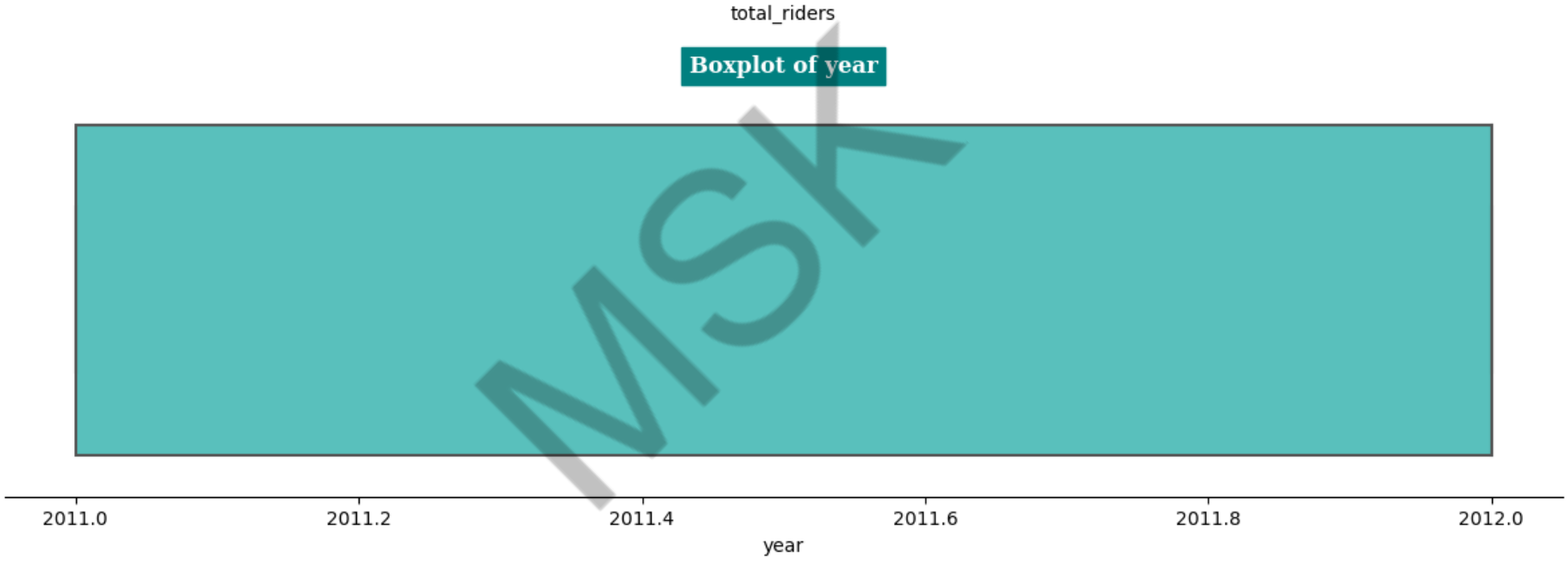

Boxplot of humidity
0
20
40
60
80
100
humidity

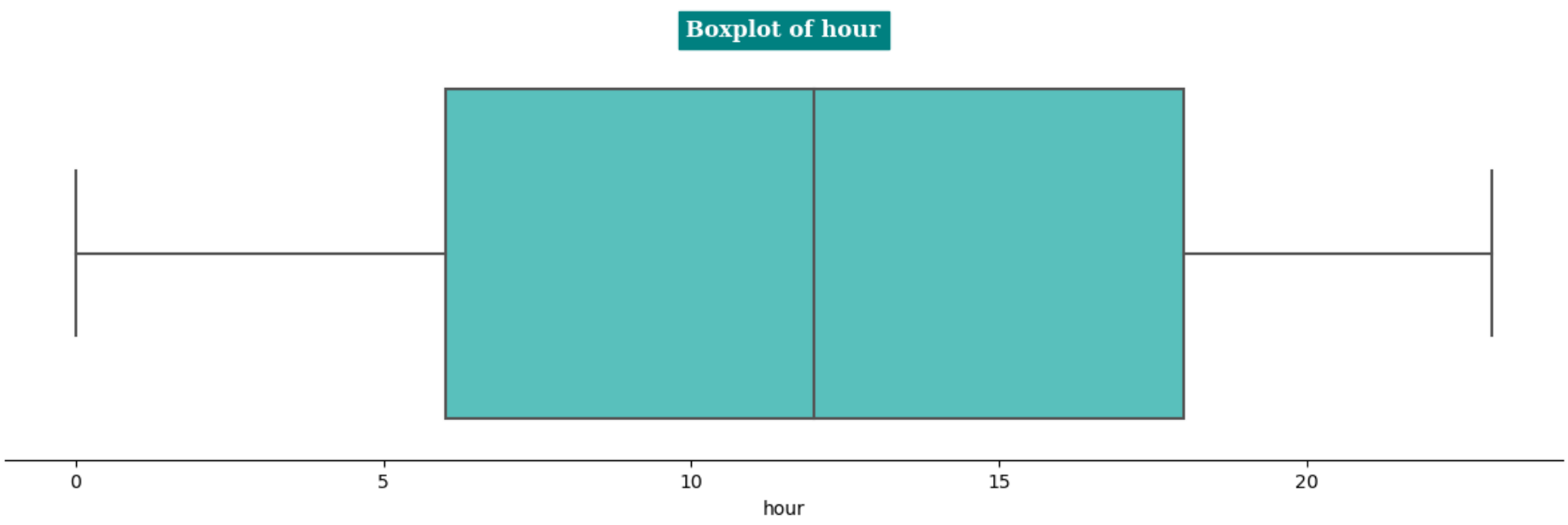

Boxplot of windspeed
0
10
20
30
40
50
windspeed


Boxplot of casual
0
50
100
150
200
250
300
350
casual









In [82]:
```
df_iqr=df_num_cols[~((df_num_cols< (Q1-1.5*IQR))|(df_num_cols > (Q3 + 1.5*IQR))).any(axis=1)]
df_iqr
```

Out[82]:

	temp	atemp	humidity	windspeed	casual	registered	total_riders	year	hour
0	9.84	14.395	81	0.0000	3	13	16	2011	0
1	9.02	13.635	80	0.0000	8	32	40	2011	1
2	9.02	13.635	80	0.0000	5	27	32	2011	2
3	9.84	14.395	75	0.0000	3	10	13	2011	3
4	9.84	14.395	75	0.0000	0	1	1	2011	4
...	...	...	...	...	...	...	...	...	...
10881	15.58	19.695	50	26.0027	7	329	336	2012	19
10882	14.76	17.425	57	15.0013	10	231	241	2012	20
10883	13.94	15.910	61	15.0013	4	164	168	2012	21
10884	13.94	17.425	61	6.0032	12	117	129	2012	22
10885	13.12	16.665	66	8.9981	4	84	88	2012	23

9518 rows × 9 columns

✳ Outlier Removal

In [140…

```
data = yd['total_riders']
display(data.to_frame())
Q1 = np.percentile(data, 25)
Q3 = np.percentile(data, 75)
IQR = Q3 - Q1

lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

clipped_data = np.clip(data, lower_bound, upper_bound)
display(filtered_data.to_frame())

filtered_data = data[(data >= lower_bound) & (data <= upper_bound)]
display(filtered_data.to_frame())

# print("Original data:", data)
# print("Clipped data:", clipped_data)
```

	total_riders
0	16
1	40
2	32
3	13
4	1
...	...
10881	336
10882	241
10883	168
10884	129
10885	88

10886 rows × 1 columns

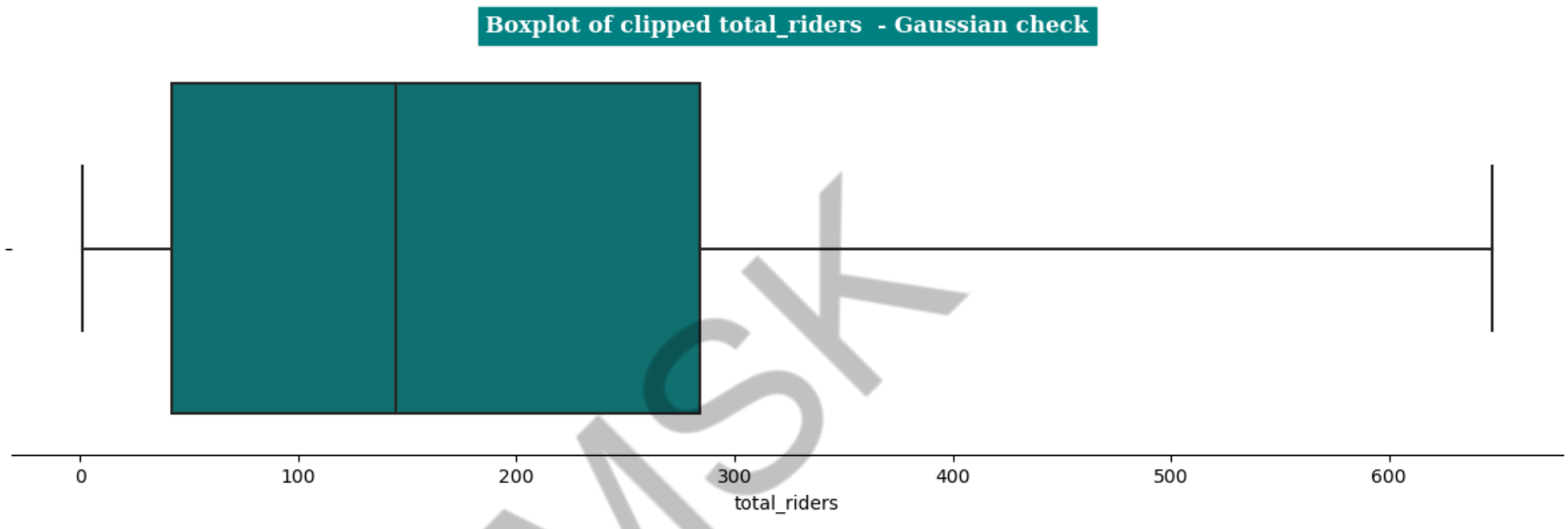
	total_riders
0	16
1	40
2	32
3	13
4	1
...	...
10881	336
10882	241
10883	168
10884	129
10885	88

10586 rows × 1 columns

	total_riders
0	16
1	40
2	32
3	13
4	1
...	...
10881	336
10882	241
10883	168
10884	129
10885	88

10586 rows × 1 columns

In [146…
```
plt.figure(figsize=(15,4))
sns.boxplot(x=clipped_data,color='teal')
sns.despine(left=True)
plt.title(f'Boxplot of clipped total_riders  - Gaussian check',fontfamily='serif',
          fontweight='bold',fontsize=12,backgroundcolor=cp[0],color='w')
plt.show()
```



Zscore Method of outlier detection - used only if data is gaussian

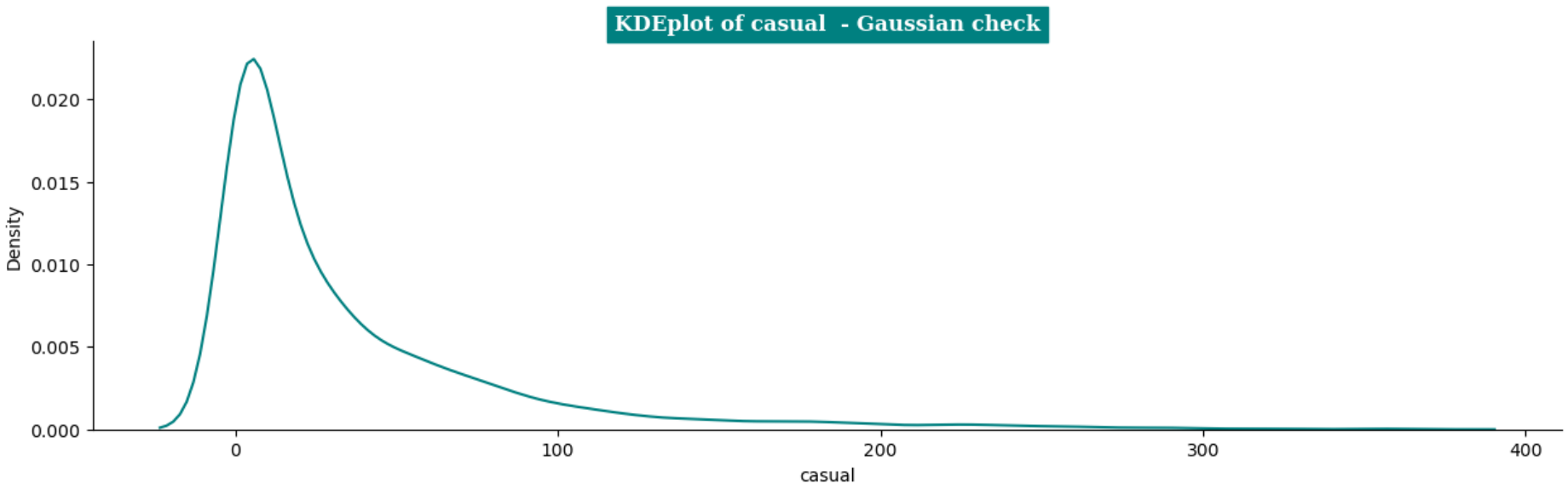
In [23]:
```
odz_cols = df_num_cols.iloc[:,4:7]
odz_cols
```

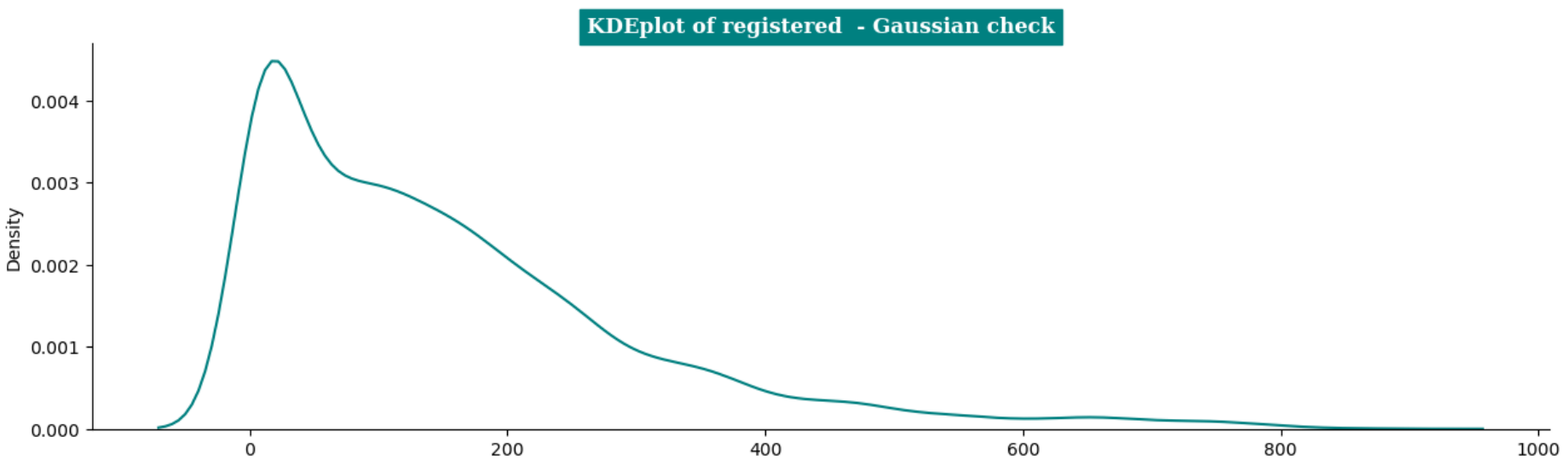
Out[23]:

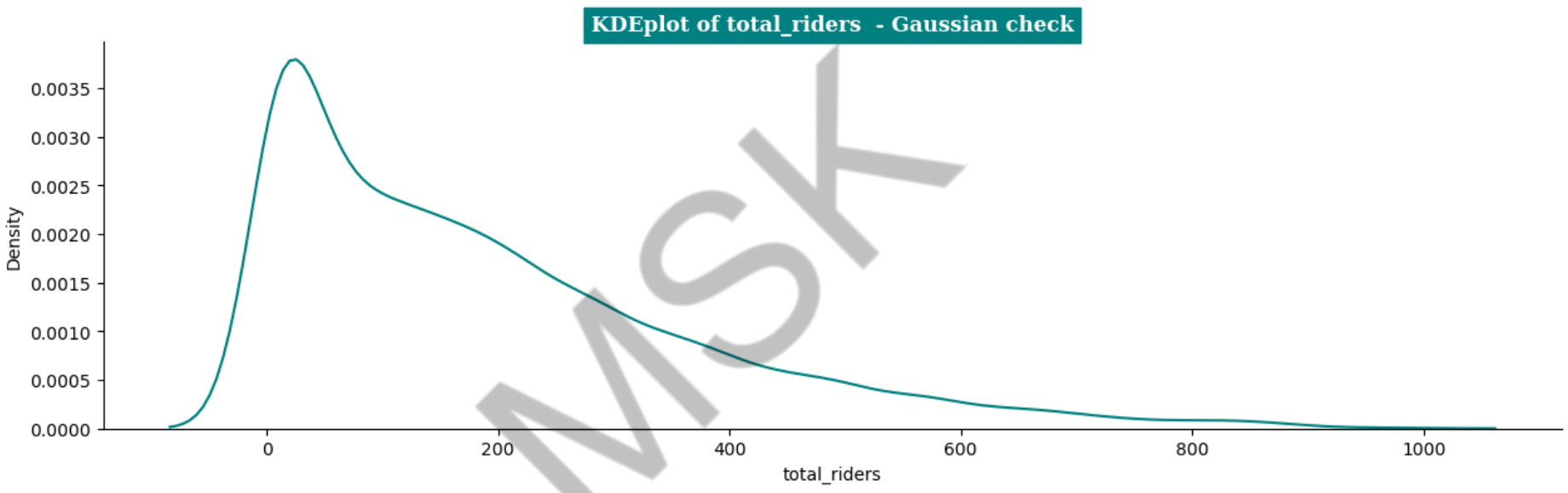
	casual	registered	total_riders
0	3	13	16
1	8	32	40
2	5	27	32
3	3	10	13
4	0	1	1
...	...	...	...
10881	7	329	336
10882	10	231	241
10883	4	164	168
10884	12	117	129
10885	4	84	88

10886 rows × 3 columns

In [121…
```
for k in enumerate(odz_cols):

    plt.figure(figsize=(15,4))
    sns.kdeplot(odz_cols,x=k[1],color='teal')
    sns.despine()
    plt.title(f'KDEplot of {k[1]}  - Gaussian check',fontfamily='serif',fontweight='bold',fontsize=12,backgroundcolor=cp[0],color
    plt.show()
```





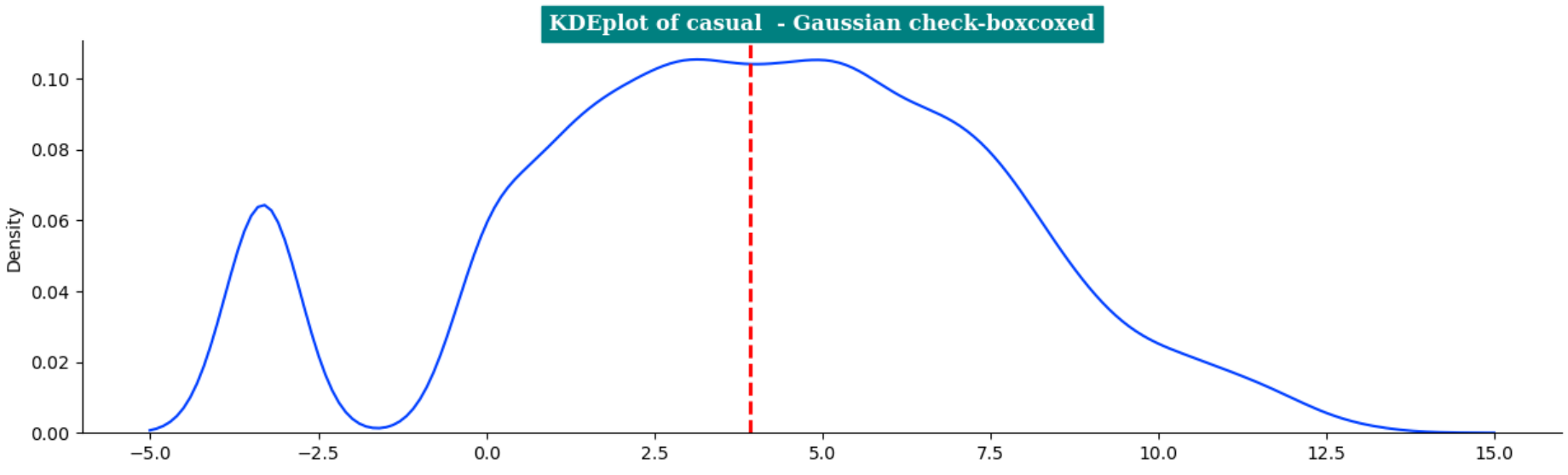


Since right skewed we will take boxcox transformation and then try to find out the outliers

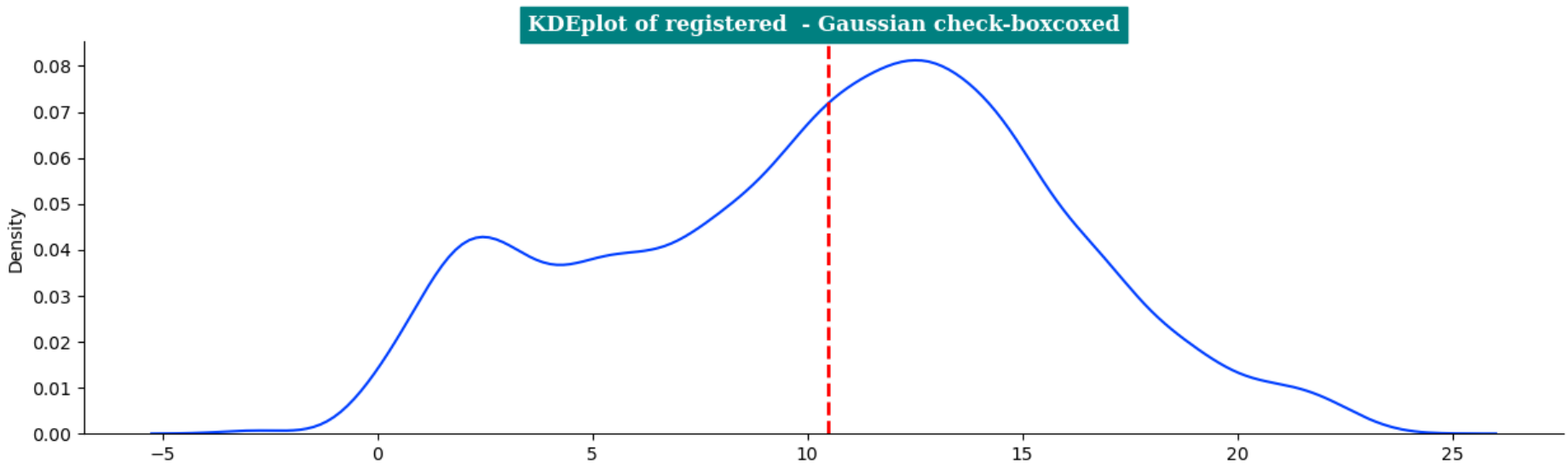
In [101…

```
for k in odz_cols:
    transformed_data , best_lambda = boxcox(odz_cols[k]+0.001)
    print(f"Best lambda of {k} = {best_lambda}")
    display(transformed_data)
    plt.figure(figsize=(15,4))
    sns.kdeplot(transformed_data)
    sns.despine()
    plt.axvline(transformed_data.mean(), color='r', linestyle='--',linewidth=2)
    plt.title(f'KDEplot of {k}  - Gaussian check-boxcoxed',fontfamily='serif',fontweight='bold',fontsize=12,backgroundcolor=cp[0]
    plt.show()
```

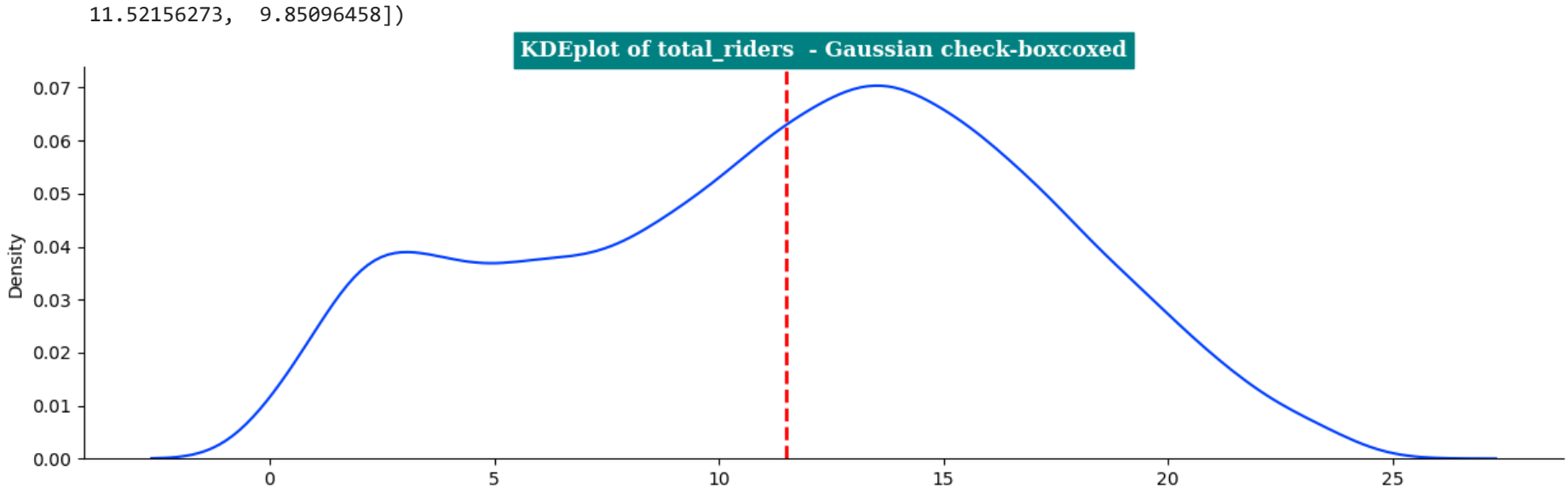
```
Best lambda of casual = 0.2459054880137843
array([1.26176369, 2.71480556, 1.97474279, ..., 1.65224204, 3.42573339,
       1.65224204])
```




```
Best lambda of registered = 0.3134521077514775
array([ 3.93836793,  6.26393537,  5.77363717, ..., 12.58863393,
       11.00379794,  9.60352783])
```




```
Best lambda of total_riders = 0.3156576641683747
array([ 4.43320701,  6.98256169,  6.29220566, ..., 12.79890375,
       11.52156273,  9.85096458])
```



Because of the presence of zero values we could use a boxcox transformation here with small value addition....

We can also use standardisation or normalisation from scikit preprocessing. Let's try standard scale or minmax scaler.

In [133…

```
for k in odz_cols:
    print(f" The minimum value of {k}:",odz_cols[k].min(),"\n",
      "The maximum value of {k}:", odz_cols[k].max())
```

```
 The minimum value of casual: 0
 The maximum value of {k}: 367
 The minimum value of registered: 0
 The maximum value of {k}: 886
 The minimum value of total_riders: 1
 The maximum value of {k}: 977
```

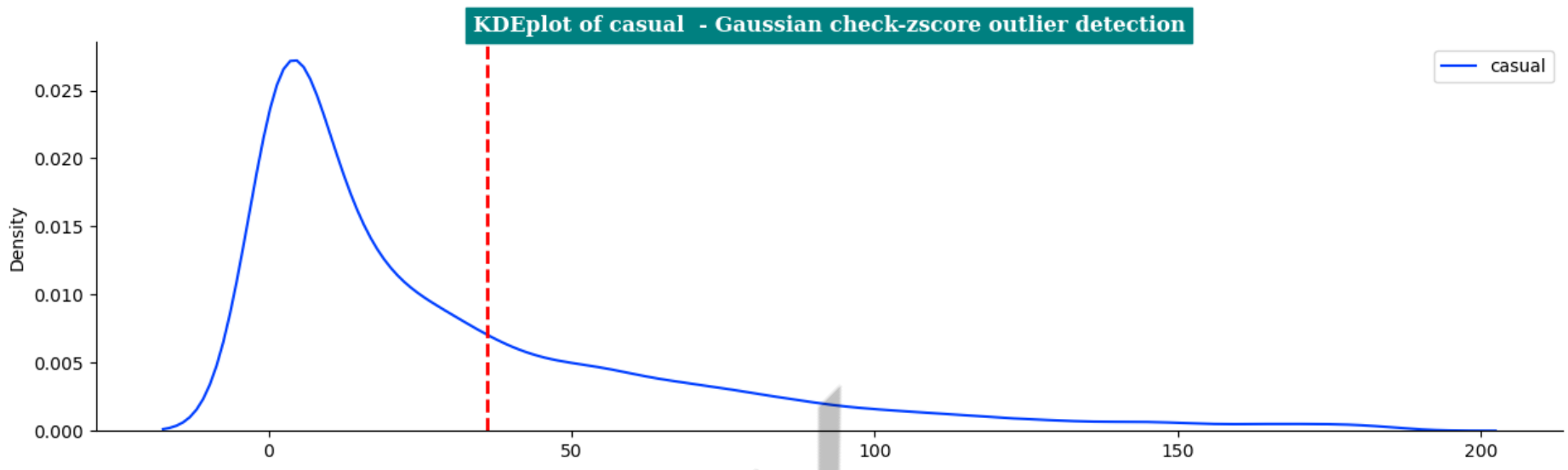
In [100…

```
for k in odz_cols:
    z_val = zscore(odz_cols[k])
    display(z_val)
    df_zscore = odz_cols[k][~((z_val < -3) | (z_val > 3))].to_frame()
    display(df_zscore)
    plt.figure(figsize=(15,4))
    sns.kdeplot(data=df_zscore,color='teal')
    sns.despine()
    plt.axvline(odz_cols[k].mean(), color='r', linestyle='--',linewidth=2)
    plt.title(f'KDEplot of {k}  - Gaussian check-zscore outlier detection',fontfamily='serif',fontweight='bold'
              ,fontsize=12,backgroundcolor=cp[0],color='w')
    plt.show()
```

```
0       -0.660992
1       -0.560908
2       -0.620958
3       -0.660992
4       -0.721042
           ...
10881   -0.580925
10882   -0.520875
10883   -0.640975
10884   -0.480841
10885   -0.640975
Name: casual, Length: 10886, dtype: float64
```

	casual
0	3
1	8
2	5
3	3
4	0
...	...
10881	7
10882	10
10883	4
10884	12
10885	4

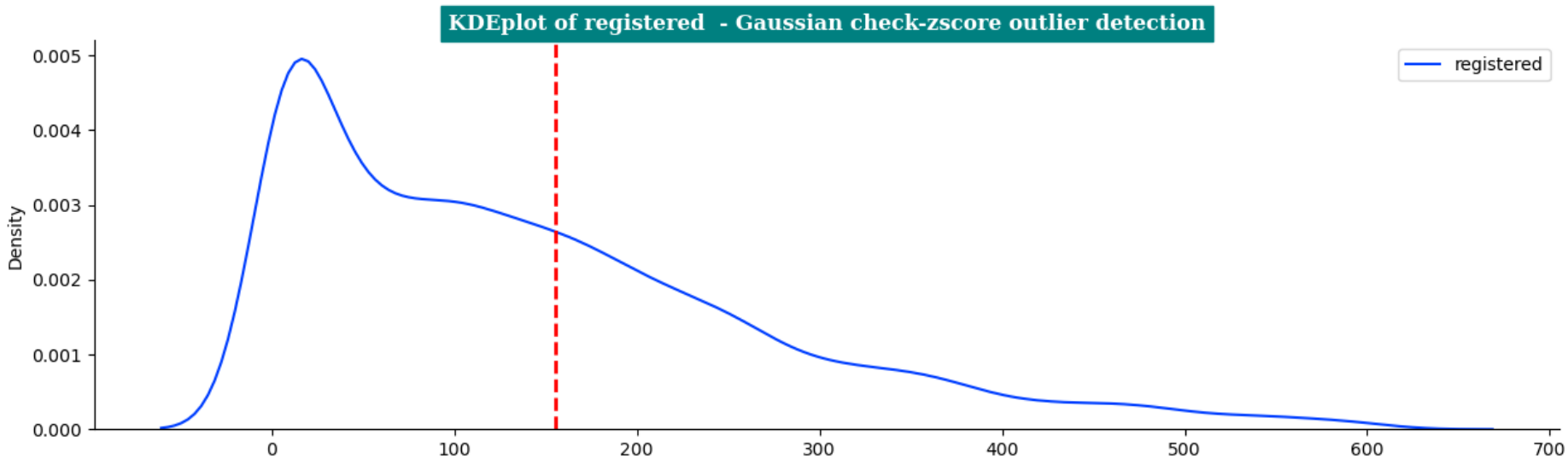
10594 rows × 1 columns




```
0       -0.943854
1       -0.818052
2       -0.851158
3       -0.963717
4       -1.023307
           ...
10881    1.148417
10882    0.499548
10883    0.055934
10884   -0.255258
10885   -0.473755
Name: registered, Length: 10886, dtype: float64
```

	registered
0	13
1	32
2	27
3	10
4	1
...	...
10881	329
10882	231
10883	164
10884	117
10885	84

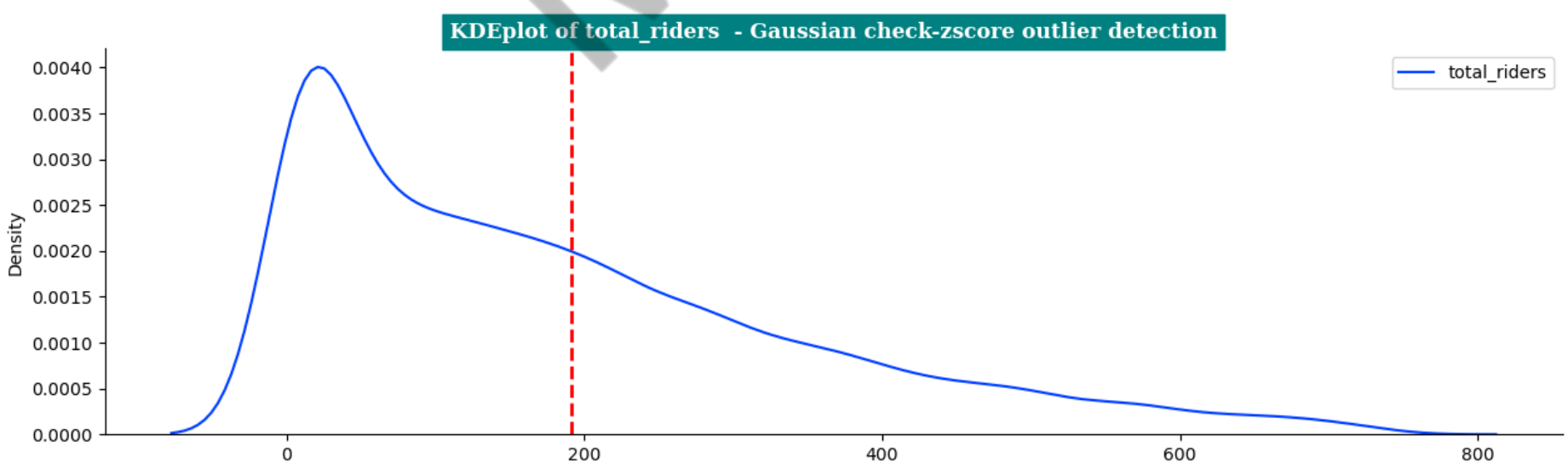
10651 rows × 1 columns




```
0        -0.969294
1        -0.836797
2        -0.880962
3        -0.985856
4        -1.052104
            ...
10881     0.797333
10882     0.272866
10883    -0.130146
10884    -0.345454
10885    -0.571803
Name: total_riders, Length: 10886, dtype: float64
```

	total_riders
0	16
1	40
2	32
3	13
4	1
...	...
10881	336
10882	241
10883	168
10884	129
10885	88

10739 rows × 1 columns



In [86]:
```
new_df = odz_cols.copy()
```

In [135…
```
for k in new_df:
    standard_scale = StandardScaler()
    new_df[k] = standard_scale.fit_transform(new_df[[k]])
    print(" The minimum value of {k}:",new_df[k].min(),"\n",
          "The maximum value of {k}:", new_df[k].max())
```

```
 The minimum value of {k}: -0.7210421496731394
The maximum value of {k}: 6.625101899627329
The minimum value of {k}: -1.0299279532173793
The maximum value of {k}: 4.836374811004244
The minimum value of {k}: -1.0521044484722468
The maximum value of {k}: 4.3361081667641255
```

In [139…
```
for k in new_df:
    minmax_scale = MinMaxScaler()
    new_df[k] = minmax_scale.fit_transform(new_df[[k]])
    print(" The minimum value of {k}:",new_df[k].min(),"\n",
          "The maximum value of {k}:", new_df[k].max())
```

```
The minimum value of {k}: 0.0
The maximum value of {k}: 1.0
The minimum value of {k}: 0.0
The maximum value of {k}: 0.9999999999999999
The minimum value of {k}: 0.0
The maximum value of {k}: 1.0
```

In [140…

```
new_df
```

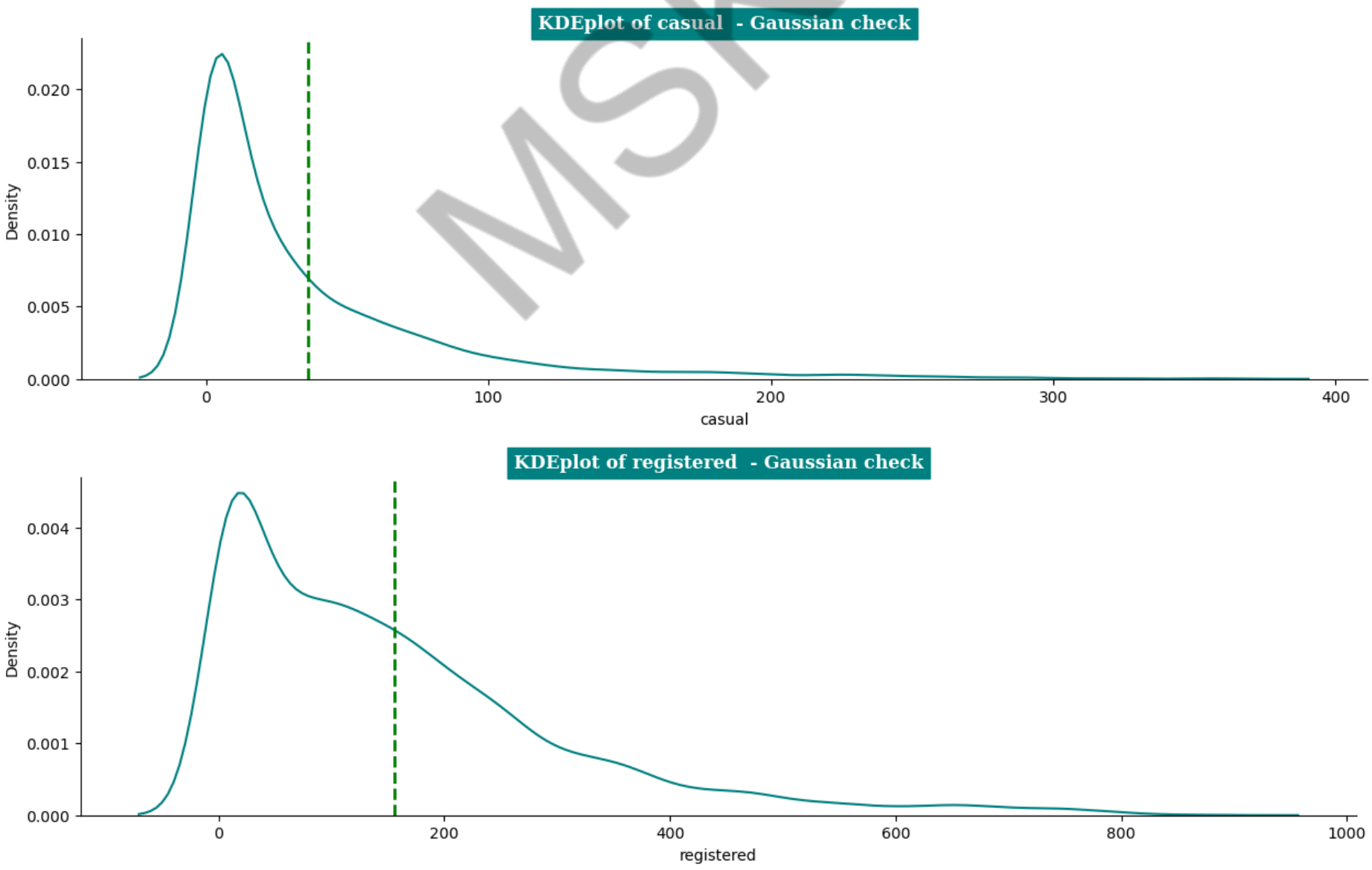
Out[140]:

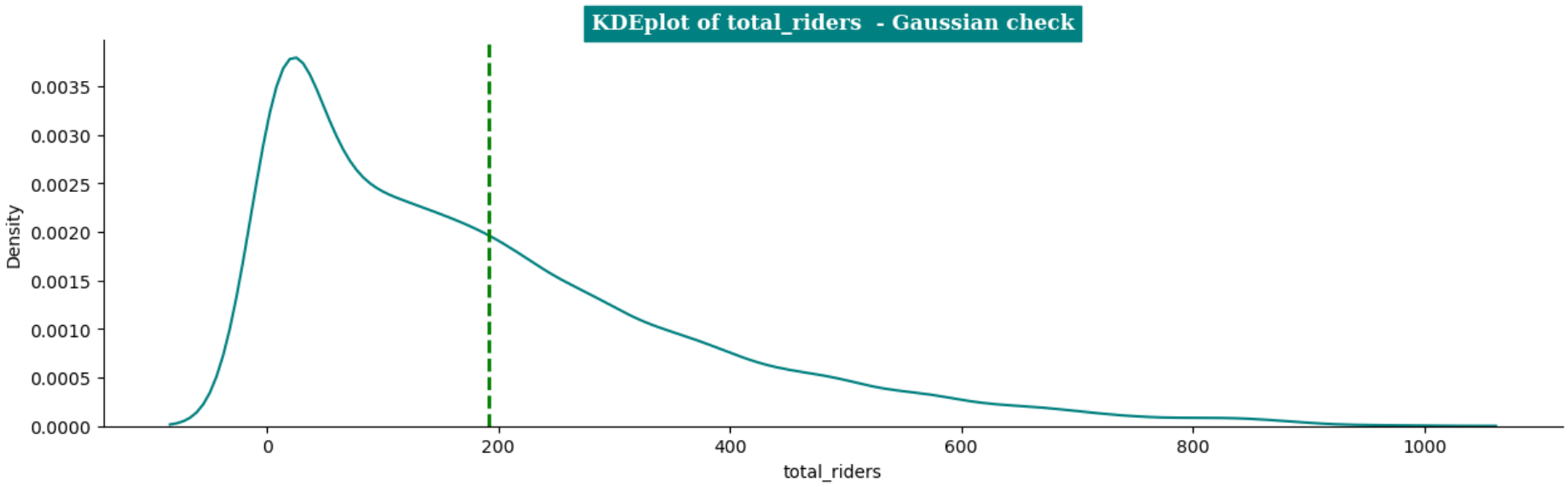
	casual	registered	total_riders
0	0.008174	0.014673	0.015369
1	0.021798	0.036117	0.039959
2	0.013624	0.030474	0.031762
3	0.008174	0.011287	0.012295
4	0.000000	0.001129	0.000000
...	...	...	...
10881	0.019074	0.371332	0.343238
10882	0.027248	0.260722	0.245902
10883	0.010899	0.185102	0.171107
10884	0.032698	0.132054	0.131148
10885	0.010899	0.094808	0.089139

10886 rows × 3 columns

In [96]:

```
for k in enumerate(new_df):

    plt.figure(figsize=(15,4))
    sns.kdeplot(new_df,x=k[1],color='teal')
    plt.axvline(new_df[k[1]].mean(), color='g', linestyle='--',linewidth=2)
    sns.despine()
    plt.title(f'KDEplot of {k[1]}  - Gaussian check',fontfamily='serif',fontweight='bold',fontsize=12,backgroundcolor=cp[0],color
    plt.show()
```

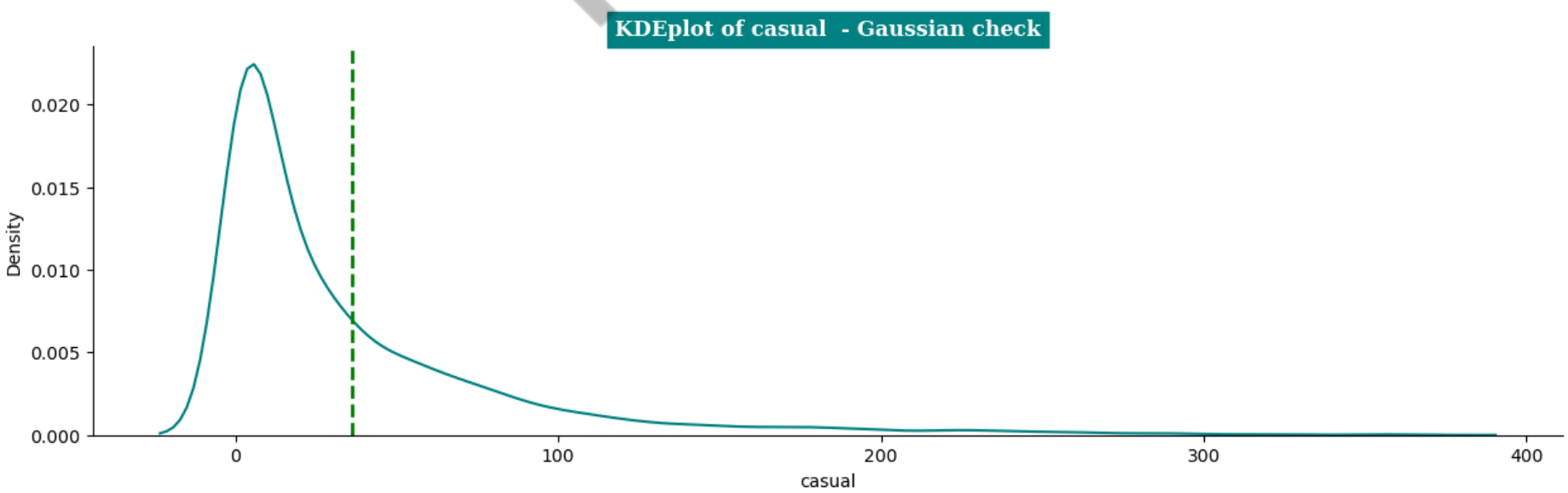




In [99]:
```
for k in new_df:
    z_val = zscore(new_df[k])
    df_zscore = new_df[k][~((z_val < -3) | (z_val > 3))].to_frame()
    display(df_zscore)

    plt.figure(figsize=(15,4))
    sns.kdeplot(new_df,x=k,color='teal')
    sns.despine()
    plt.axvline(new_df[k].mean(), color='g', linestyle='--',linewidth=2)
    plt.title(f'KDEplot of {k}  - Gaussian check',fontfamily='serif',fontweight='bold',fontsize=12,backgroundcolor=cp[0],color='w
    plt.show()
```

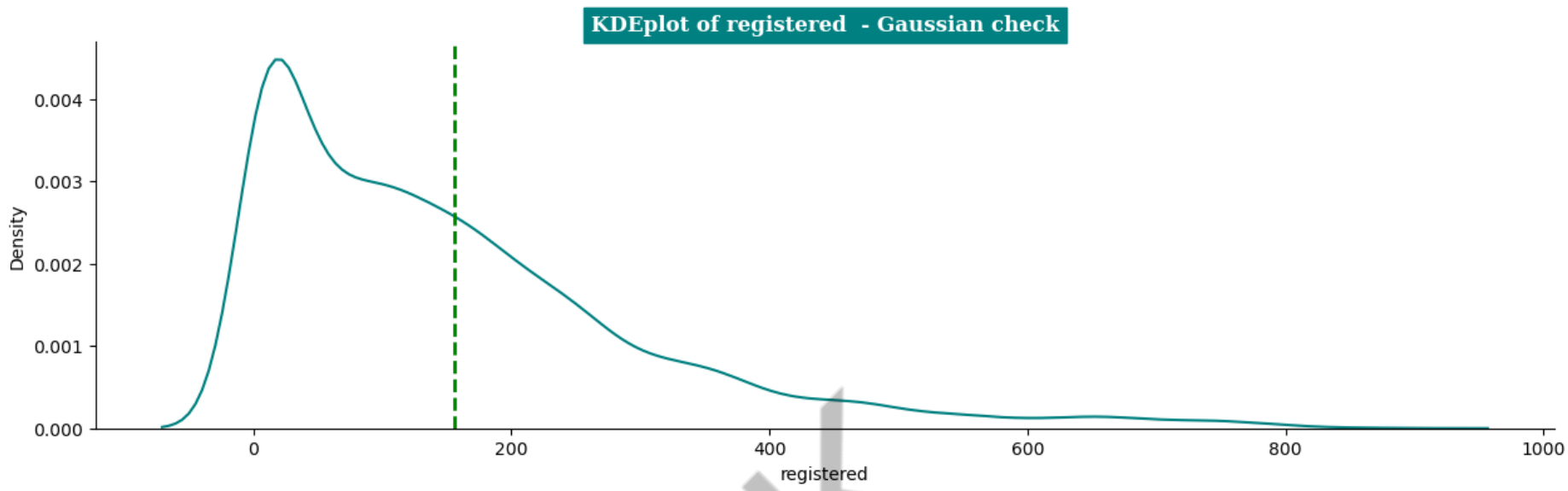
	casual
0	3
1	8
2	5
3	3
4	0
...	...
10881	7
10882	10
10883	4
10884	12
10885	4

10594 rows × 1 columns



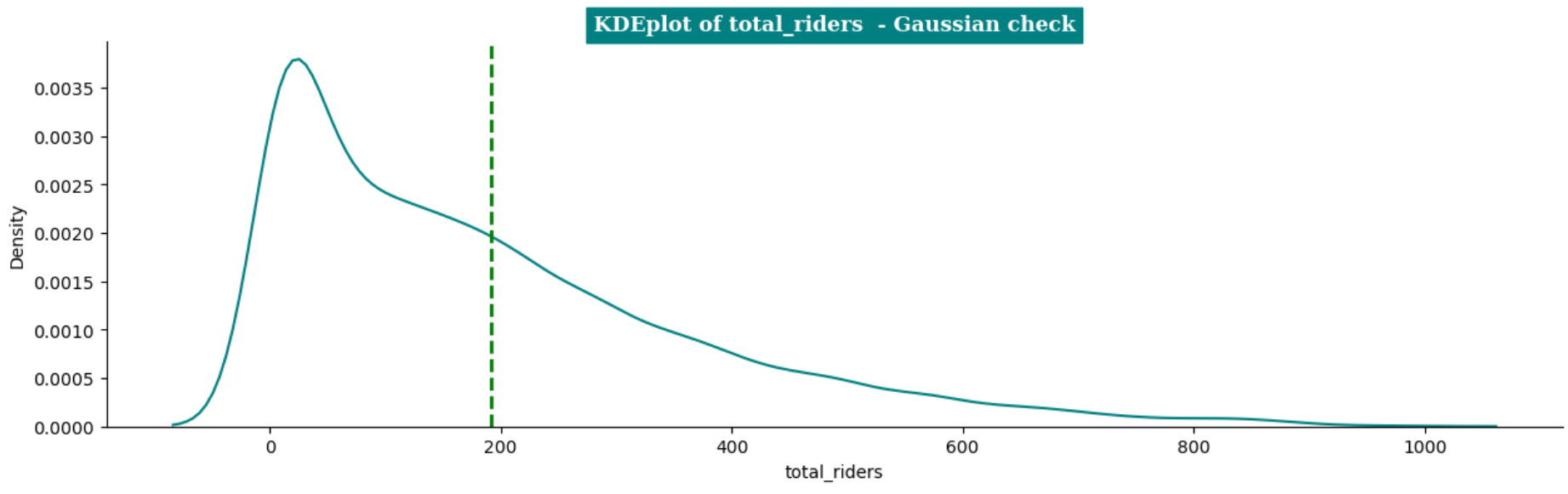
	registered
0	13
1	32
2	27
3	10
4	1
...	...
10881	329
10882	231
10883	164
10884	117
10885	84

10651 rows × 1 columns



	total_riders
0	16
1	40
2	32
3	13
4	1
...	...
10881	336
10882	241
10883	168
10884	129
10885	88

10739 rows × 1 columns



🔆Understanding:

- Understanding is that standardisation may change the plot as the mean and the variance and standard deviations are changed, But while doing MinMax Scalar we are just scaling the Axis and the plot remains the same. Also, Boxcox transformation gives you only the approximate normally distributed It doesn't mean the data is exactly normally distributed, It also has the impact of outliers.

- Another observation is that the outliers found in this Z score may or may not be the same as the iqr method. So it is always advisable to use iqr method and clip the data if necessary. Try to do an analysis with the original data and the clipped data and observe the changes.

> Even after bootstrapping the data, the plot is right skewed only ..

🤝Testing correlations between the variables:

In [146…
```
num_cols
```

Out[146]:

	temp	atemp	humidity	windspeed	casual	registered	total_riders
0	9.84	14.395	81	0.0000	3	13	16
1	9.02	13.635	80	0.0000	8	32	40
2	9.02	13.635	80	0.0000	5	27	32
3	9.84	14.395	75	0.0000	3	10	13
4	9.84	14.395	75	0.0000	0	1	1
...	...	...	...	...	...	...	...
10881	15.58	19.695	50	26.0027	7	329	336
10882	14.76	17.425	57	15.0013	10	231	241
10883	13.94	15.910	61	15.0013	4	164	168
10884	13.94	17.425	61	6.0032	12	117	129
10885	13.12	16.665	66	8.9981	4	84	88

10886 rows × 7 columns

> **Null Hypothesis** --- Two numeric columns are **independent** of each other.
>
> **Alternate Hypothesis** --- Two numeric columns are **dependent** on each other.

```
- The correration coefficient gives the strength of relationship.
- The range of spearman corrcoef is [-1,1].
```

Here's some of the datas are not normally distributed,So in order to capture the nonlinear properties, we will consider these spearman correlation.

In [148…
```
for col in num_cols:
    print(f'Testing Correlation between {col} and Total_riders ')
    #np.corrcoef(yd[col].rank(), yd['total_riders'].rank())[0,1]   -- can use this as well
    pearson_coefficient,p_val = pearsonr(yd[col],yd['total_riders'])
    spearman_coefficient,p_val = spearmanr(yd[col],yd['total_riders'])
    pc = pearson_coefficient
    sc = spearman_coefficient
    if sc>0 :
        if sc>0.5:
            print('There is Strong Positive correlation between total_riders and', col,"- pearson-corrcoef: ",np.round(pc,2))
            print('There is Strong Positive correlation between total_riders and', col,"- spearman-corrcoef: ",np.round(sc,2))
        else:
            print('There is Weak Positive correlation between total_riders and', col,"- pearson-corrcoef: ",np.round(pc,2))
            print('There is weak Positive correlation between total_riders and', col,"- spearman-corrcoef: ",np.round(sc,2))
    if sc == 0:
        print('There is No correlation between total_riders and', col,"- pearson-corrcoef: ",np.round(pc,2))
        print('There is No correlation between total_riders and', col,"- spearman-corrcoef: ",np.round(sc,2))
    if sc< 0:
        if sc<0.5:
            print('There is Strong Negative correlation between total_riders and', col,"- pearson-corrcoef: ",np.round(pc,2))
            print('There is Strong Negative correlation between total_riders and', col,"- spearman-corrcoef: ",np.round(sc,2))
        else:
            print('There is Weak Positive correlation between total_riders and', col,"- pearson-corrcoef: ",np.round(pc,2))
            print('There is weak Positive correlation between total_riders and', col,"- spearman-corrcoef: ",np.round(sc,2))
    print()
    print('-'*100)
    print()
```

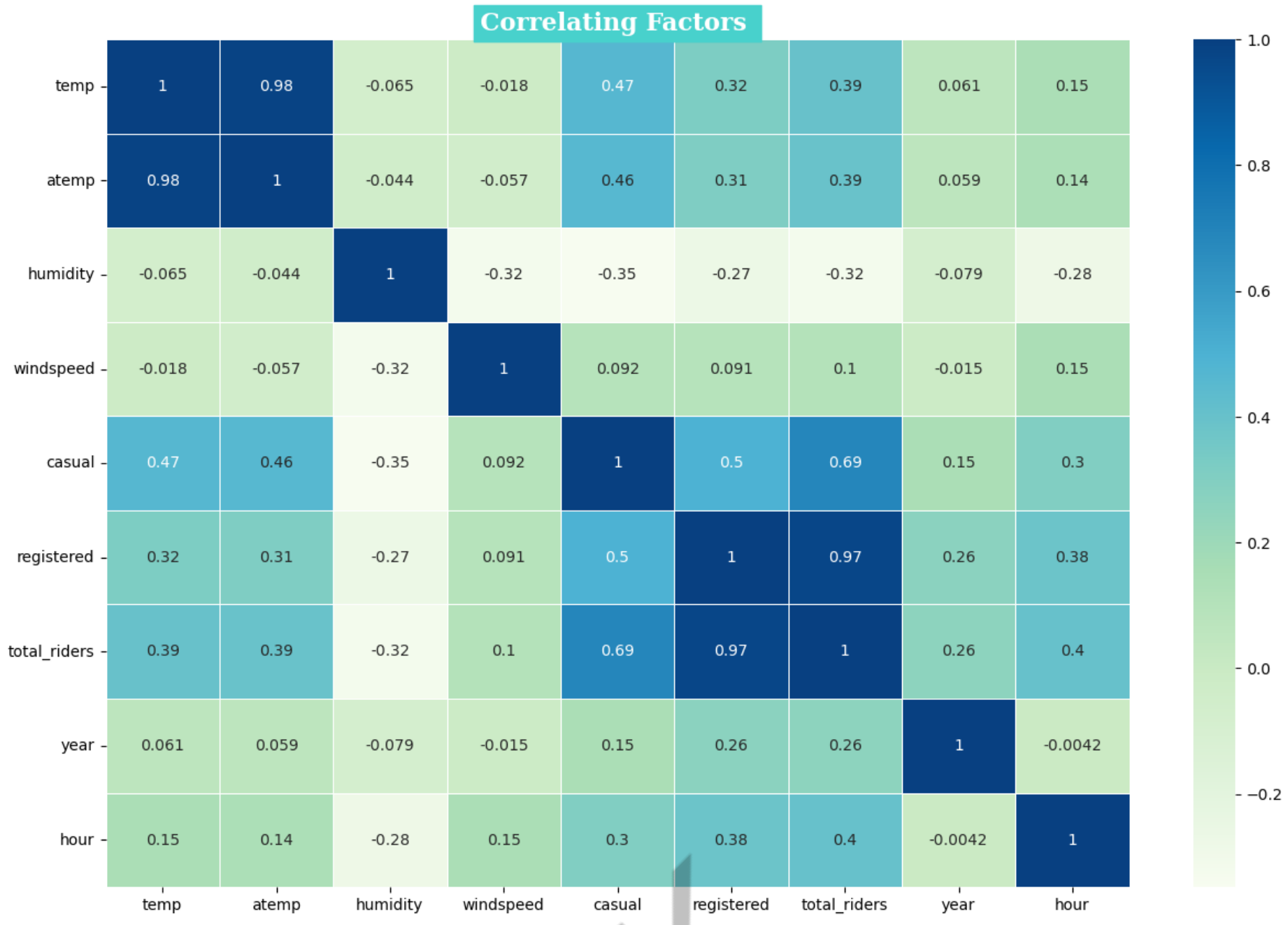
```
Testing Correlation between temp and Total_riders
There is Weak Positive correlation between total_riders and temp - pearson-corrcoef:  0.39
There is weak Positive correlation between total_riders and temp - spearman-corrcoef:  0.41

-----------------------------------------------------------------------------------------------------

Testing Correlation between atemp and Total_riders
There is Weak Positive correlation between total_riders and atemp - pearson-corrcoef:  0.39
There is weak Positive correlation between total_riders and atemp - spearman-corrcoef:  0.41

-----------------------------------------------------------------------------------------------------

Testing Correlation between humidity and Total_riders
There is Strong Negative correlation between total_riders and humidity - pearson-corrcoef:  -0.32
There is Strong Negative correlation between total_riders and humidity - spearman-corrcoef:  -0.35

-----------------------------------------------------------------------------------------------------

Testing Correlation between windspeed and Total_riders
There is Weak Positive correlation between total_riders and windspeed - pearson-corrcoef:  0.1
There is weak Positive correlation between total_riders and windspeed - spearman-corrcoef:  0.14

-----------------------------------------------------------------------------------------------------

Testing Correlation between casual and Total_riders
There is Strong Positive correlation between total_riders and casual - pearson-corrcoef:  0.69
There is Strong Positive correlation between total_riders and casual - spearman-corrcoef:  0.85

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

```
Testing Correlation between registered and Total_riders
There is Strong Positive correlation between total_riders and registered - pearson-corrcoef:  0.97
There is Strong Positive correlation between total_riders and registered - spearman-corrcoef:  0.99

-----------------------------------------------------------------------------------------------------

Testing Correlation between total_riders and Total_riders
There is Strong Positive correlation between total_riders and total_riders - pearson-corrcoef:  1.0
There is Strong Positive correlation between total_riders and total_riders - spearman-corrcoef:  1.0

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

In [183…

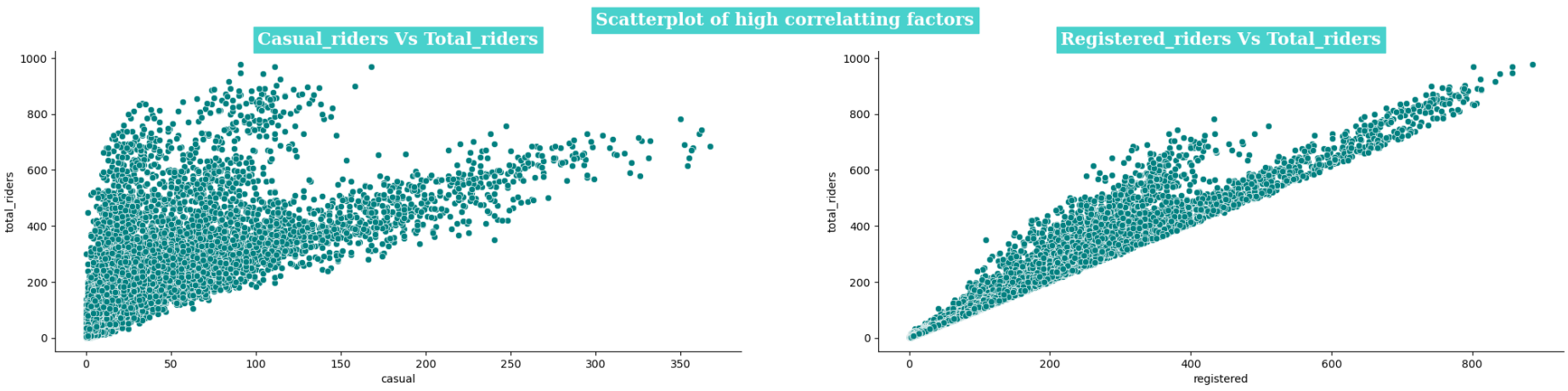
```
corr_df =  yd.corr(numeric_only=True)
display(corr_df)
plt.figure(figsize=(15,10))
sns.heatmap(yd.corr(numeric_only=True), annot=True, linewidth=.5,cmap='GnBu')
plt.yticks(rotation=0)
plt.title('Correlating Factors ',fontfamily='serif',fontweight='bold',fontsize=16,backgroundcolor=cp[1],color='w')
plt.show()
```

	temp	atemp	humidity	windspeed	casual	registered	total_riders	year	hour
temp	1.000000	0.984948	-0.064949	-0.017852	0.467097	0.318571	0.394454	0.061226	0.145430
atemp	0.984948	1.000000	-0.043536	-0.057473	0.462067	0.314635	0.389784	0.058540	0.140343
humidity	-0.064949	-0.043536	1.000000	-0.318607	-0.348187	-0.265458	-0.317371	-0.078606	-0.278011
windspeed	-0.017852	-0.057473	-0.318607	1.000000	0.092276	0.091052	0.101369	-0.015221	0.146631
casual	0.467097	0.462067	-0.348187	0.092276	1.000000	0.497250	0.690414	0.145241	0.302045
registered	0.318571	0.314635	-0.265458	0.091052	0.497250	1.000000	0.970948	0.264265	0.380540
total_riders	0.394454	0.389784	-0.317371	0.101369	0.690414	0.970948	1.000000	0.260403	0.400601
year	0.061226	0.058540	-0.078606	-0.015221	0.145241	0.264265	0.260403	1.000000	-0.004234
hour	0.145430	0.140343	-0.278011	0.146631	0.302045	0.380540	0.400601	-0.004234	1.000000



In [196…

```
plt.figure(figsize=(25,5))
plt.suptitle('Scatterplot of high correlatting factors',fontfamily='serif',fontweight='bold',
             fontsize=16,backgroundcolor=cp[1],color='w')

plt.subplot(121)
sns.scatterplot(data=yd, x='casual', y='total_riders',color=cp[0])
plt.title('Casual_riders Vs Total_riders',fontfamily='serif',fontweight='bold',fontsize=16,backgroundcolor=cp[1],color='w')

plt.subplot(122)
sns.scatterplot(data=yd, x='registered', y='total_riders',color=cp[0])
plt.title("")
plt.title('Registered_riders Vs Total_riders',fontfamily='serif',fontweight='bold',fontsize=16,backgroundcolor=cp[1],color='w')

sns.despine()
plt.show()
```



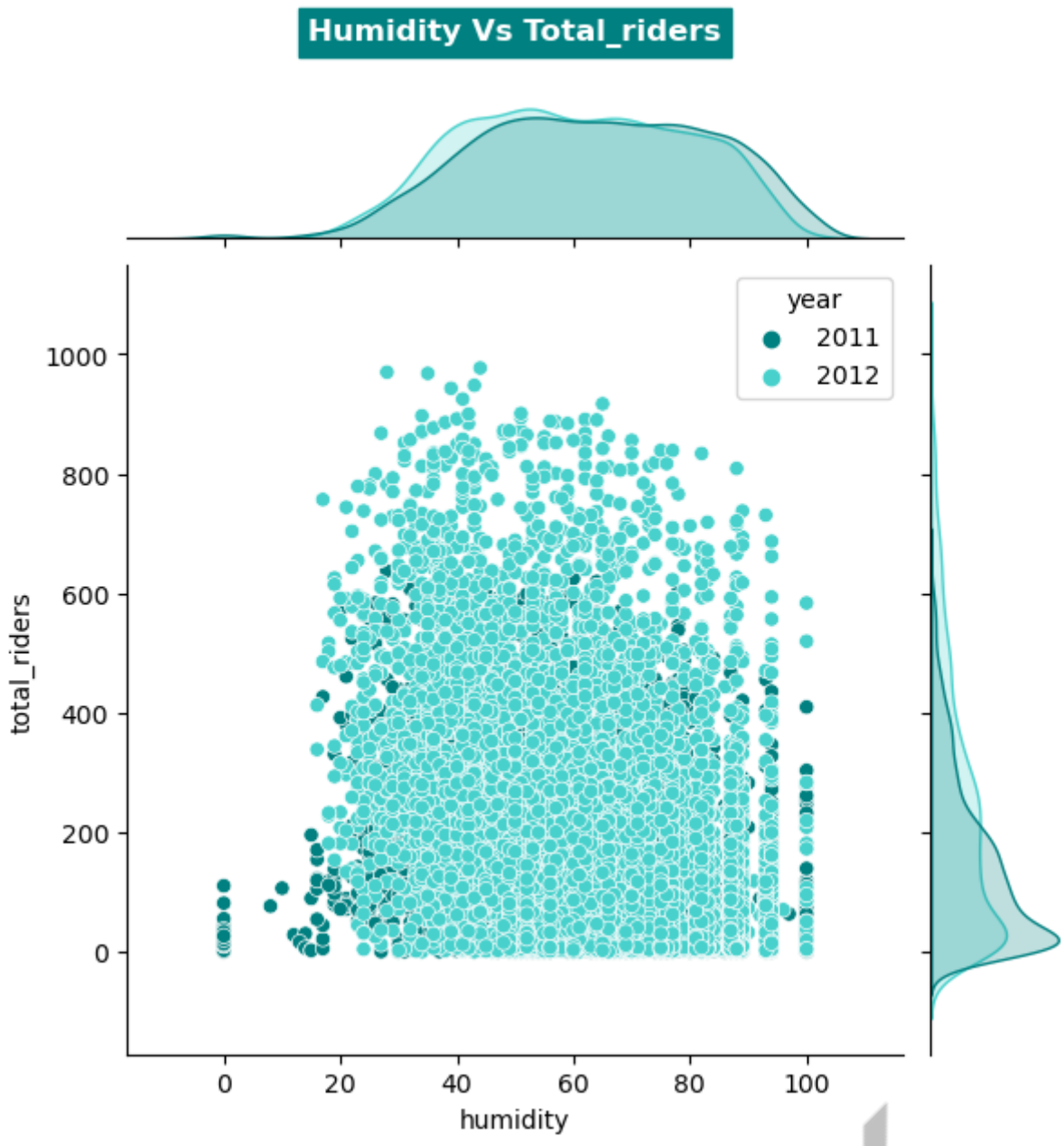
🏷️ Insights

- We can conclude that registered users contribute more towards total_riders as there is a high correlation between them.
- We can also see high correlation between temp and atemp features.
- This scatterplot confirms that there is a high correlation between registered customers and total yulu bike riders.

In [173…

```
plt.figure(figsize = (14,6))
sns.jointplot(x=yd['humidity'], y=yd['total_riders'],hue=yd['year'],kind="scatter", palette=cp)
plt.title("Humidity Vs Total_riders", loc = "center",pad= 90,fontweight="bold",backgroundcolor='teal',color='w')
plt.show()
```

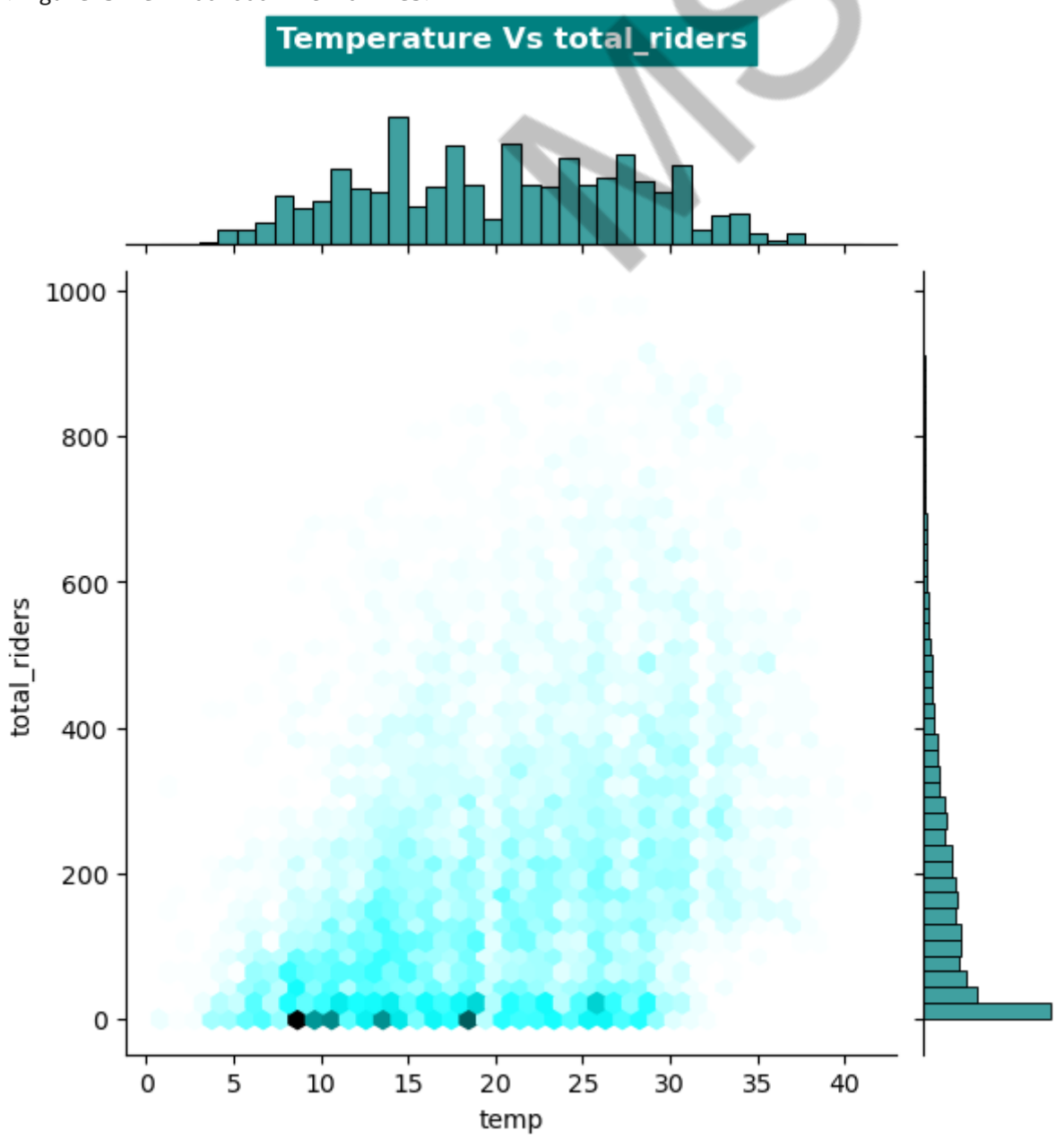
```
<Figure size 1400x600 with 0 Axes>
```



In [177…

```
plt.figure(figsize = (14,6))
sns.jointplot(x=yd['temp'], y=yd['total_riders'], kind="hex", color='teal')
plt.title("Temperature Vs total_riders", loc = "center",pad= 90,fontweight="bold",backgroundcolor='teal',color='w')
plt.show()
```

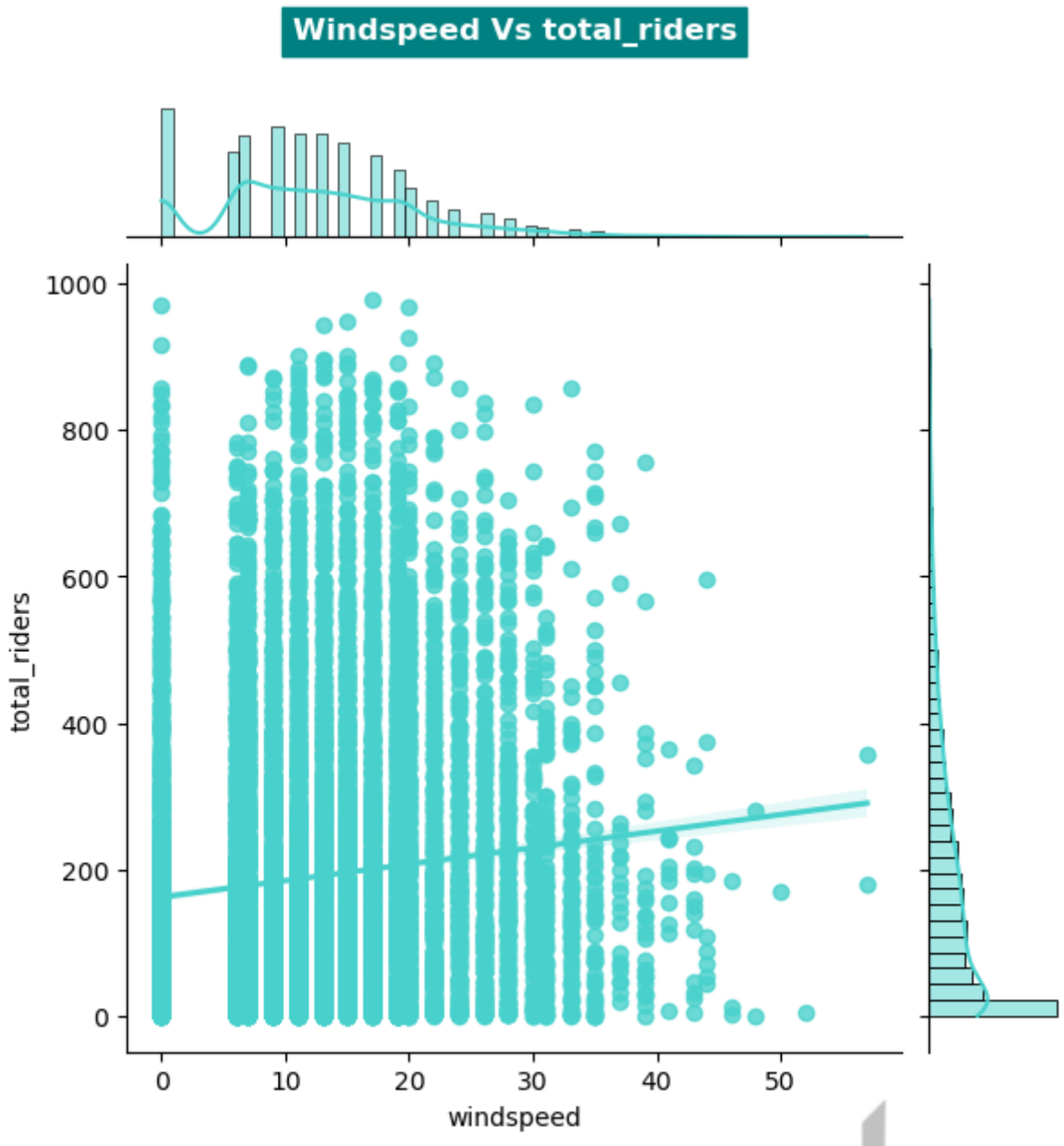
```
<Figure size 1400x600 with 0 Axes>
```



In [180…

```
plt.figure(figsize = (14,6))
sns.jointplot(x=yd['windspeed'], y=yd['total_riders'], kind="reg",color=cp[1])
plt.title("Windspeed Vs total_riders", loc = "center",pad= 90,fontweight="bold",backgroundcolor='teal',color='w')
plt.show()
```

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

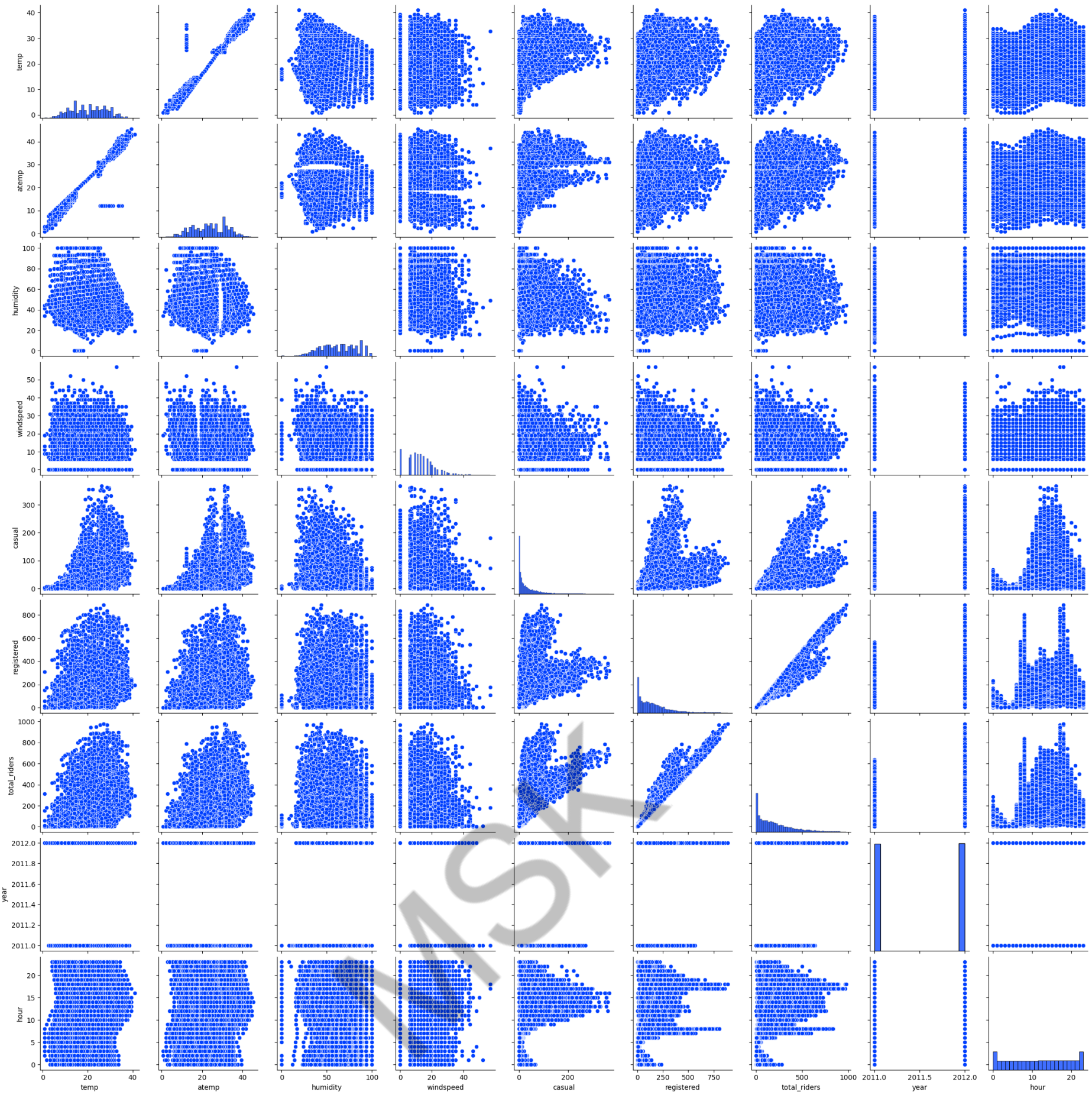


In [184…
```
cat_cols
```

Out[184]:
```
['season', 'holiday', 'workingday', 'weather']
```

In [184…
```
plt.figure(figsize=(14,0.05))
plt.axis('off')
plt.title(f' Pairplot Analysis ',fontfamily='serif',fontweight='bold',fontsize=15,backgroundcolor='teal',color='w')
sns.pairplot(data=yd,palette=cp)
plt.show()
```

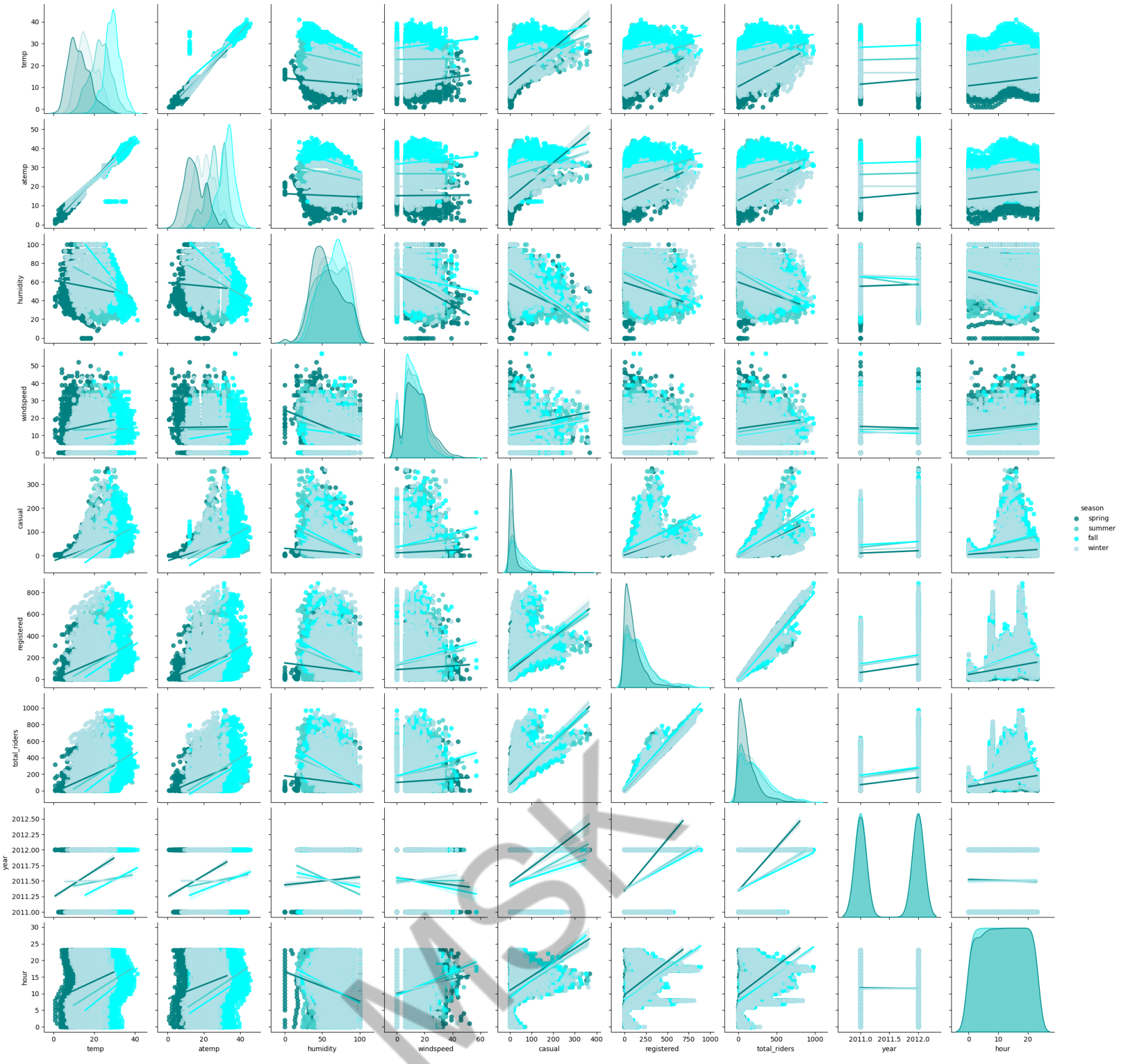
Pairplot Analysis



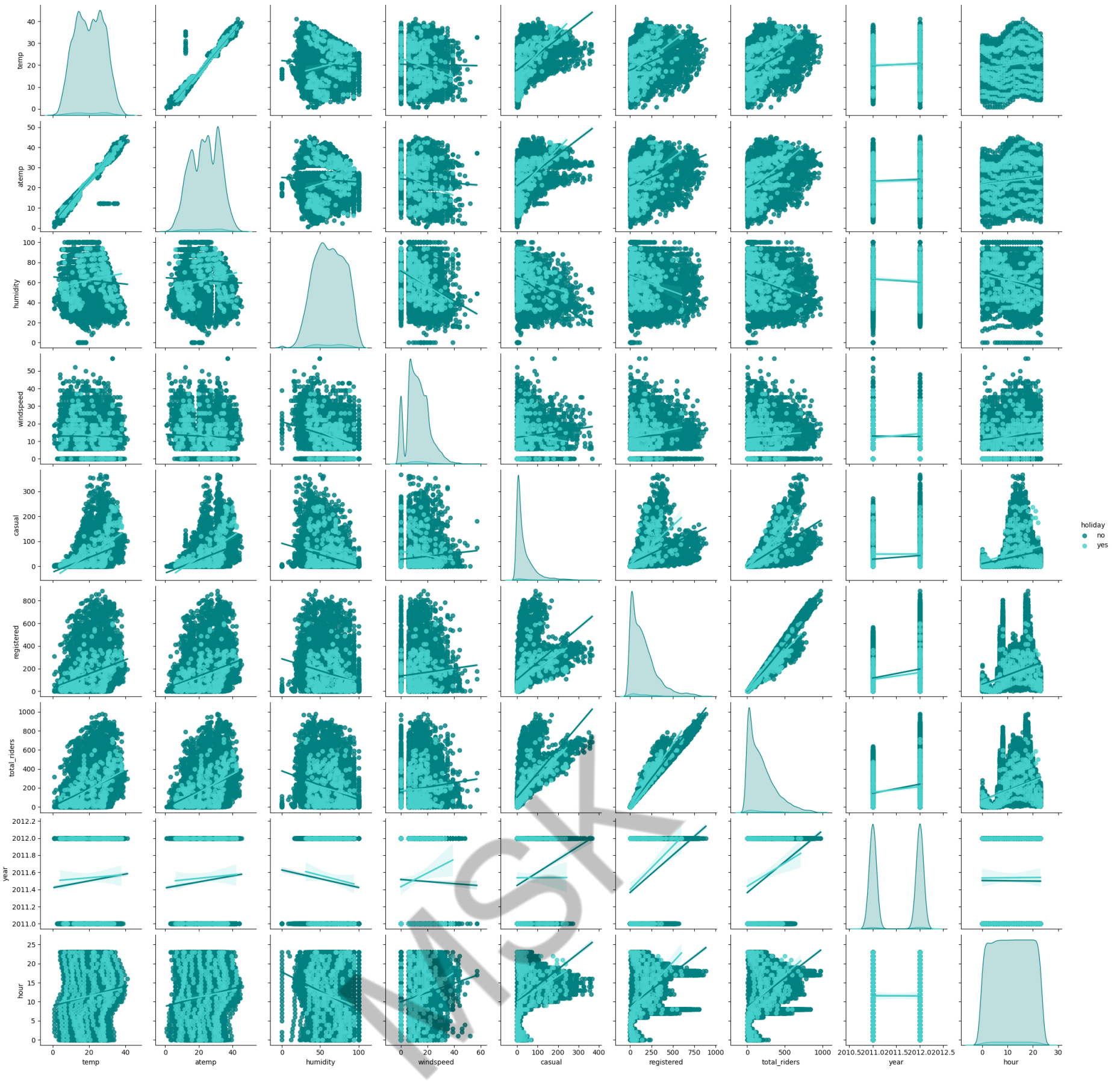
In [187…

```
for i in range(len(cat_cols)):
    plt.figure(figsize=(14,0.05))
    plt.axis('off')
    plt.title(f' Pairplot based on {cat_cols[i]} ',fontfamily='serif',fontweight='bold',fontsize=20,backgroundcolor='teal',color=
    sns.pairplot(yd,hue=cat_cols[i],kind='reg',palette=cp)
    plt.show()
    print()
```

Pairplot based on season

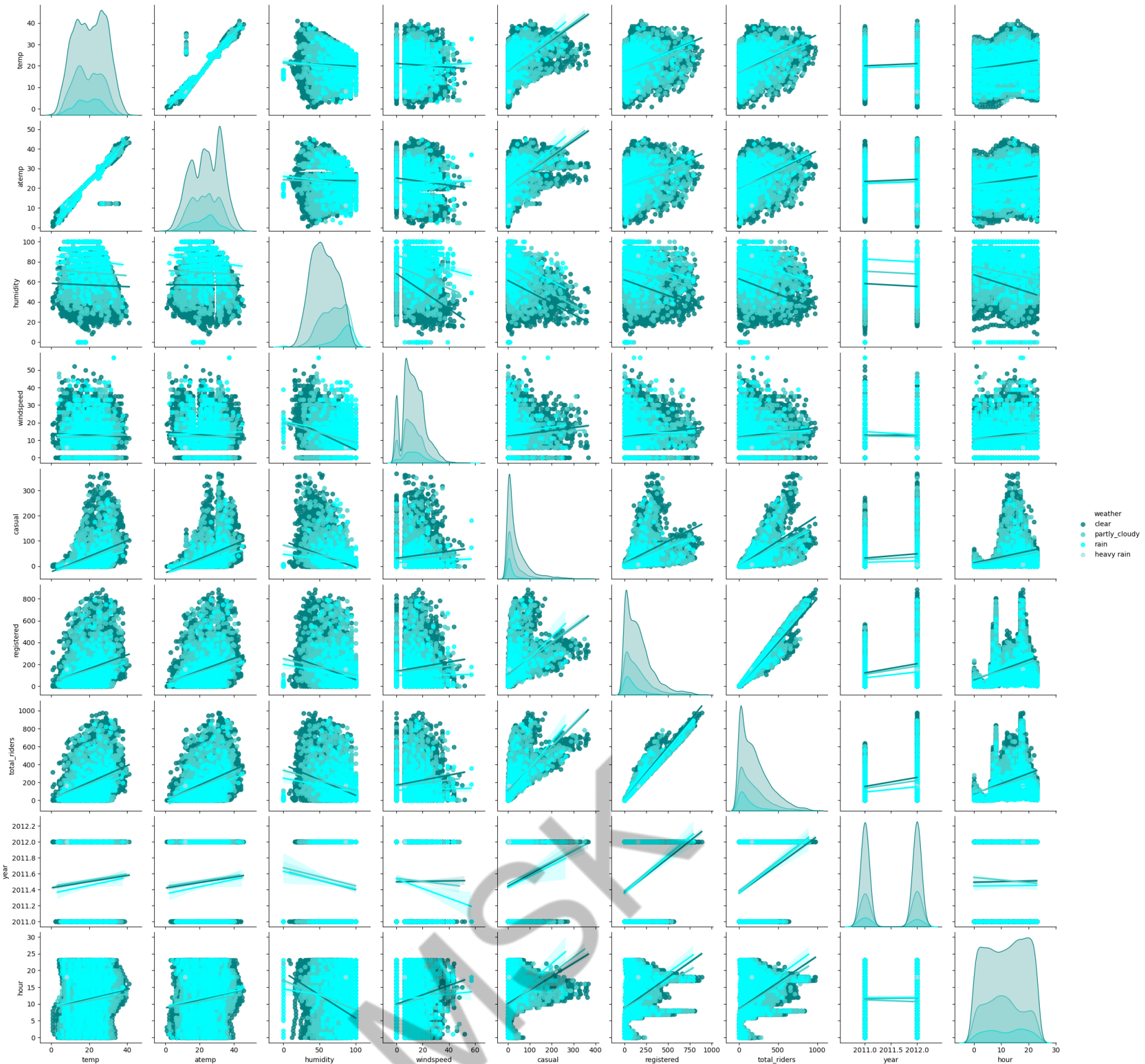


Pairplot based on holiday



Pairplot based on workingday

Pairplot based on weather



Hypothesis Testing

🧨 Check if Working Day has an effect on the number of electric bikes rented on Weekdays and Weekends ?

2- Sample T-Test to check if Working Day has an effect on the number of electric cycles rented.

Assumptions of T-Test

1. The sample size should be less than 30.
2. The population variance is unknown.
3. The population mean and standard deviation are finite.
4. The means of the two populations being compared should follow normal distributions.
5. If using Student's original definition of the t-test, the two populations being compared should have the same variance. If the sample sizes in the two groups being compared are equal, Student's original t-test is highly robust to the presence of unequal variances.

STEP-1 : Set up Null Hypothesis

> **Null Hypothesis (Ho)** - No.of bikes rented on working days and non working days are same.
> H_{o}: $μ_{workingday}$ = $μ_{non-workingday}$
>
> **Alternate Hypothesis (Ha)** - No.of bikes rented on working days and non working days are different.
> H_{a}: $μ_{workingday}$!= $μ_{non-workingday}$

STEP-2 : Checking for basic assumpitons for the hypothesis

> Normality checks

    - Distribution check using **QQ Plot**

- Distribution check using **prob Plot**
- Confirmation by **Shapiro-wilks Test**
- Homogeneity of Variances using **Levene's test**

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

- We know that the test statistic while performing a T-Test follows Tdistribution.
    - If data follows normal distribution we go with **ttest_ind**
    - Else we will go with **Mannwhitney_u test** (Non - Parametric test)

STEP-4: Decide the kind of test.

- We will be performing **Two tailed t-test for indpependent variables**

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

- we will be computing the t-stat value using the ttest function using scipy.stats.
- We set our ***alpha to be 0.05 (i.e) confidence level = 95%***

STEP-6: Compare p-value and alpha.

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

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

T-Test Independent 👇

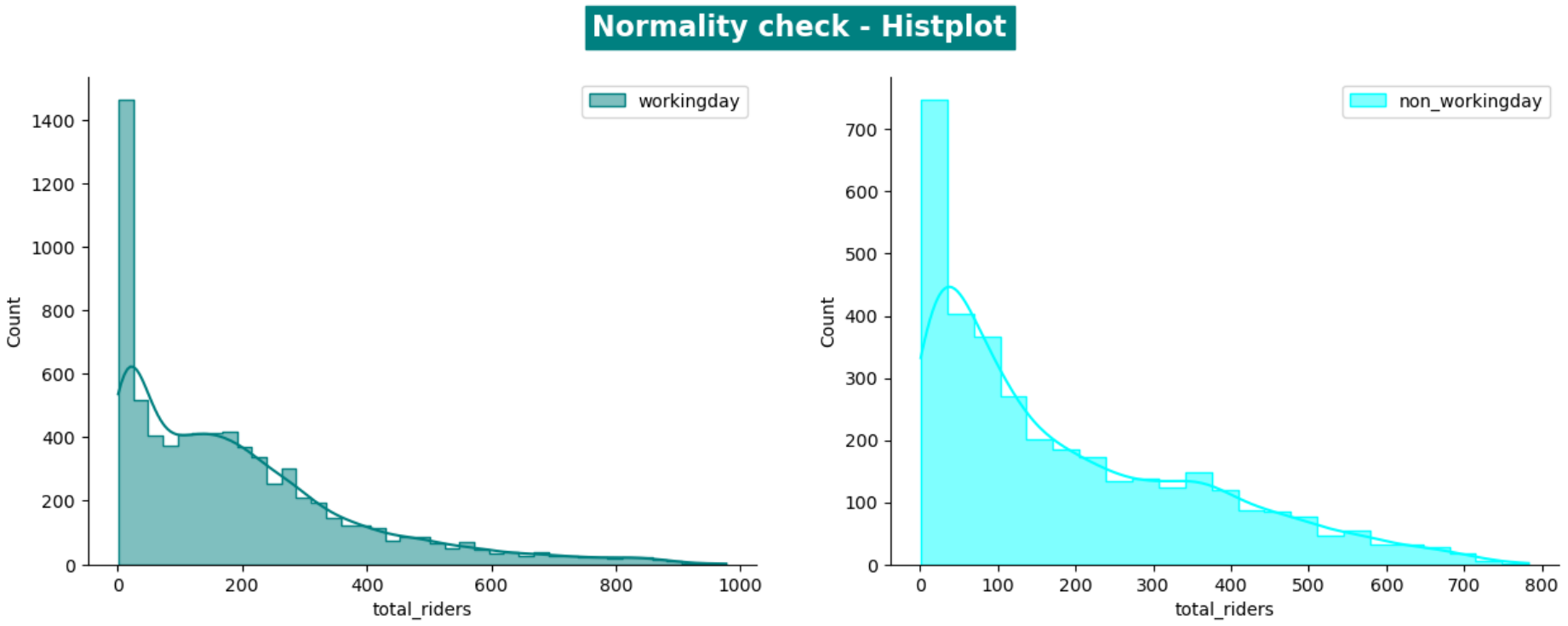
In [186…
```
yd.groupby('workingday')['total_riders'].describe()
```

Out[186]:

workingday	count	mean	std	min	25%	50%	75%	max
no	3474.0	188.506621	173.724015	1.0	44.0	128.0	304.0	783.0
yes	7412.0	193.011873	184.513659	1.0	41.0	151.0	277.0	977.0

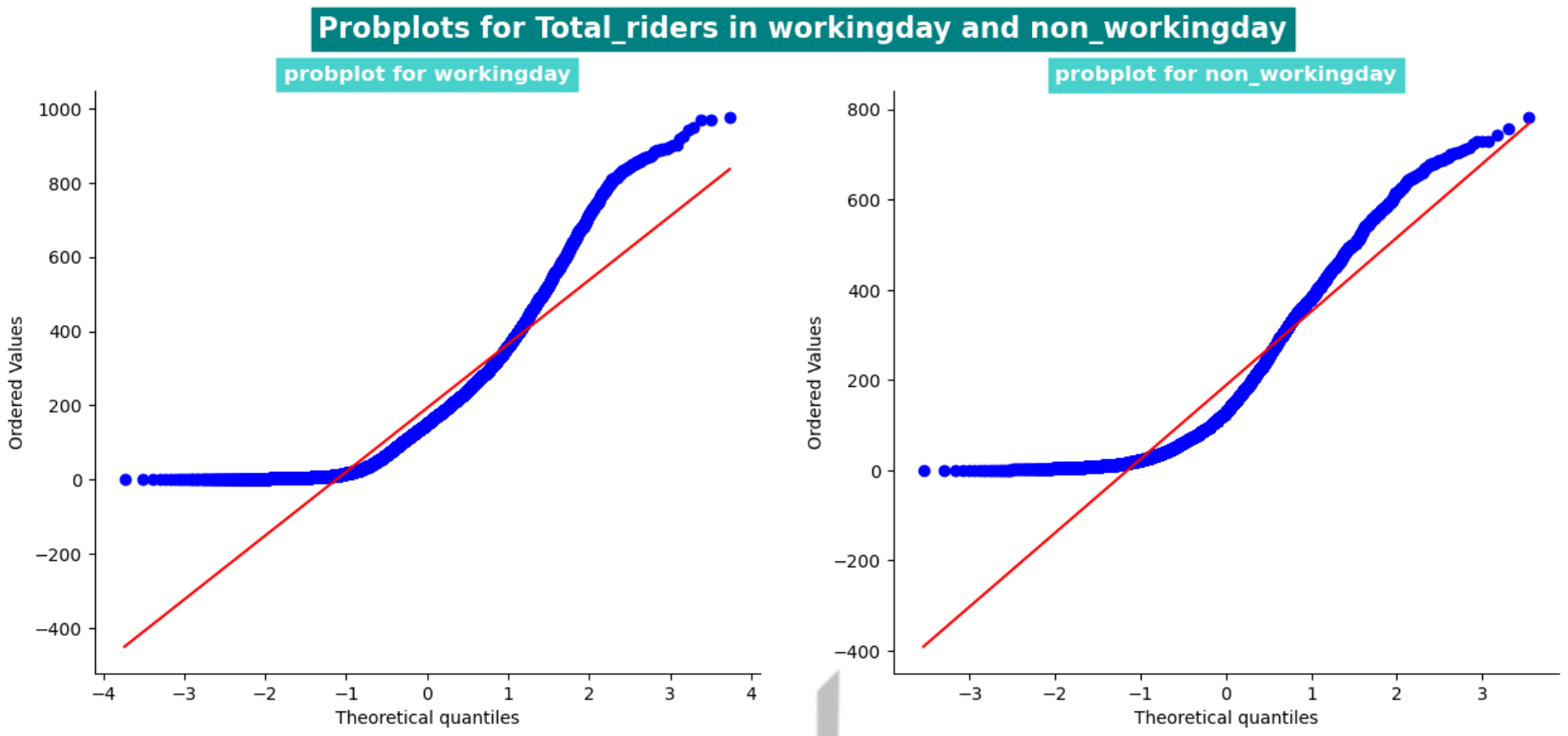
In [62]:
```
workingday_yes = yd[yd['workingday']== 'yes']['total_riders']
workingday_no = yd[yd['workingday']== 'no']['total_riders']
```

In [202…
```
plt.figure(figsize = (15, 5))
plt.suptitle("Normality check - Histplot",fontsize=16,fontweight="bold",backgroundcolor='teal',color='w')
plt.subplot(121)
sns.histplot(workingday_yes, element = 'step', color =cp[0], kde = True, label = 'workingday')
plt.legend()
plt.subplot(122)
sns.histplot(workingday_no,element = 'step', color = cp[2], kde = True, label = 'non_workingday')
plt.legend()
sns.despine()
plt.show()
```



In [206…

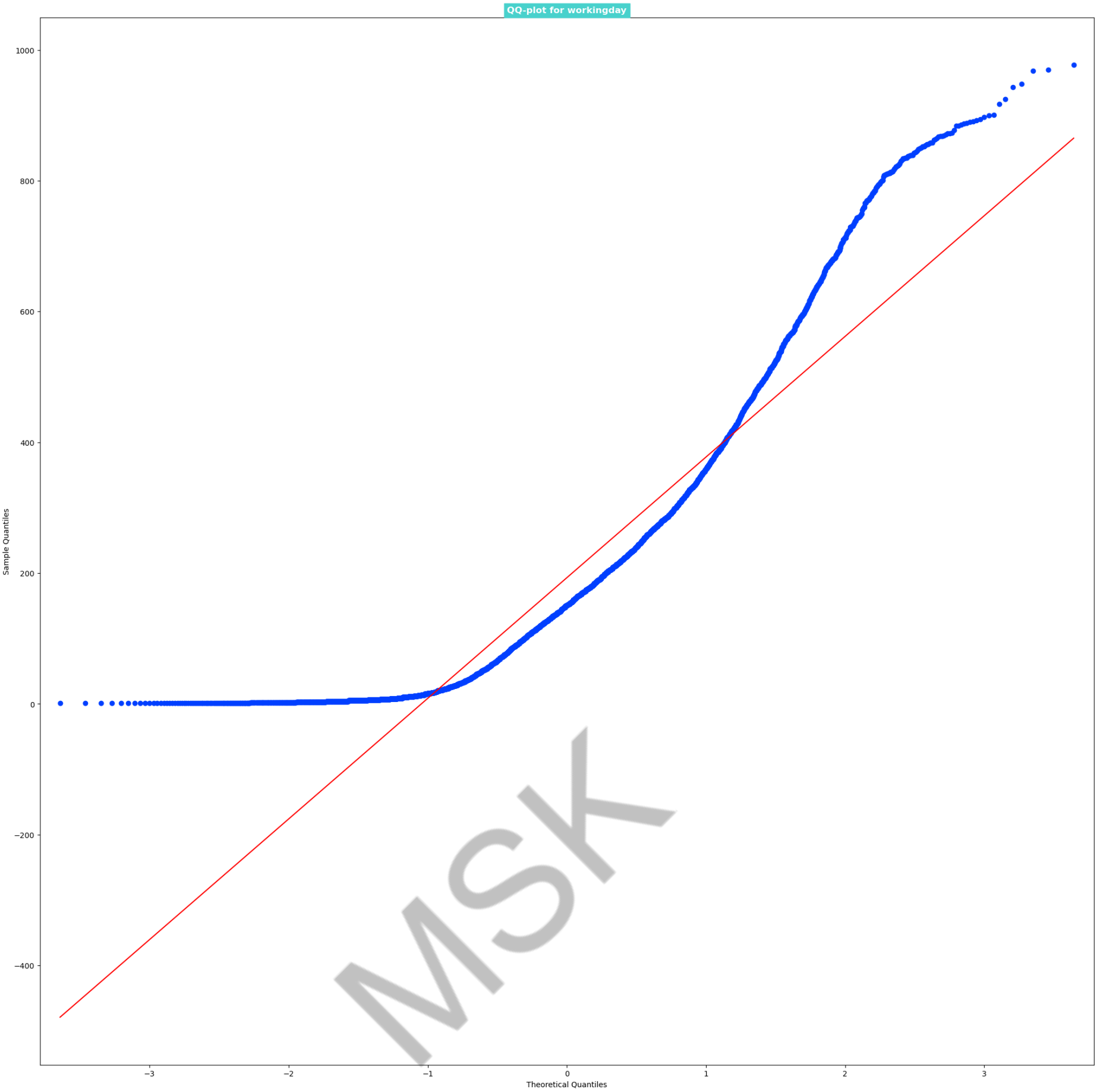
```
plt.figure(figsize = (15, 6))
plt.subplot(121)
plt.suptitle('Probplots for Total_riders in workingday and non_workingday',fontsize=16,fontweight="bold",backgroundcolor='teal',c
probplot(workingday_yes, plot = plt, dist = 'norm')
plt.title('probplot for workingday',fontsize=12,fontweight="bold",backgroundcolor=cp[1],color='w')
plt.subplot(122)
probplot(workingday_no, plot = plt, dist = 'norm')
plt.title('probplot for non_workingday',fontsize=12,fontweight="bold",backgroundcolor=cp[1],color='w')
sns.despine()
plt.show()

## can also do qqplot
```

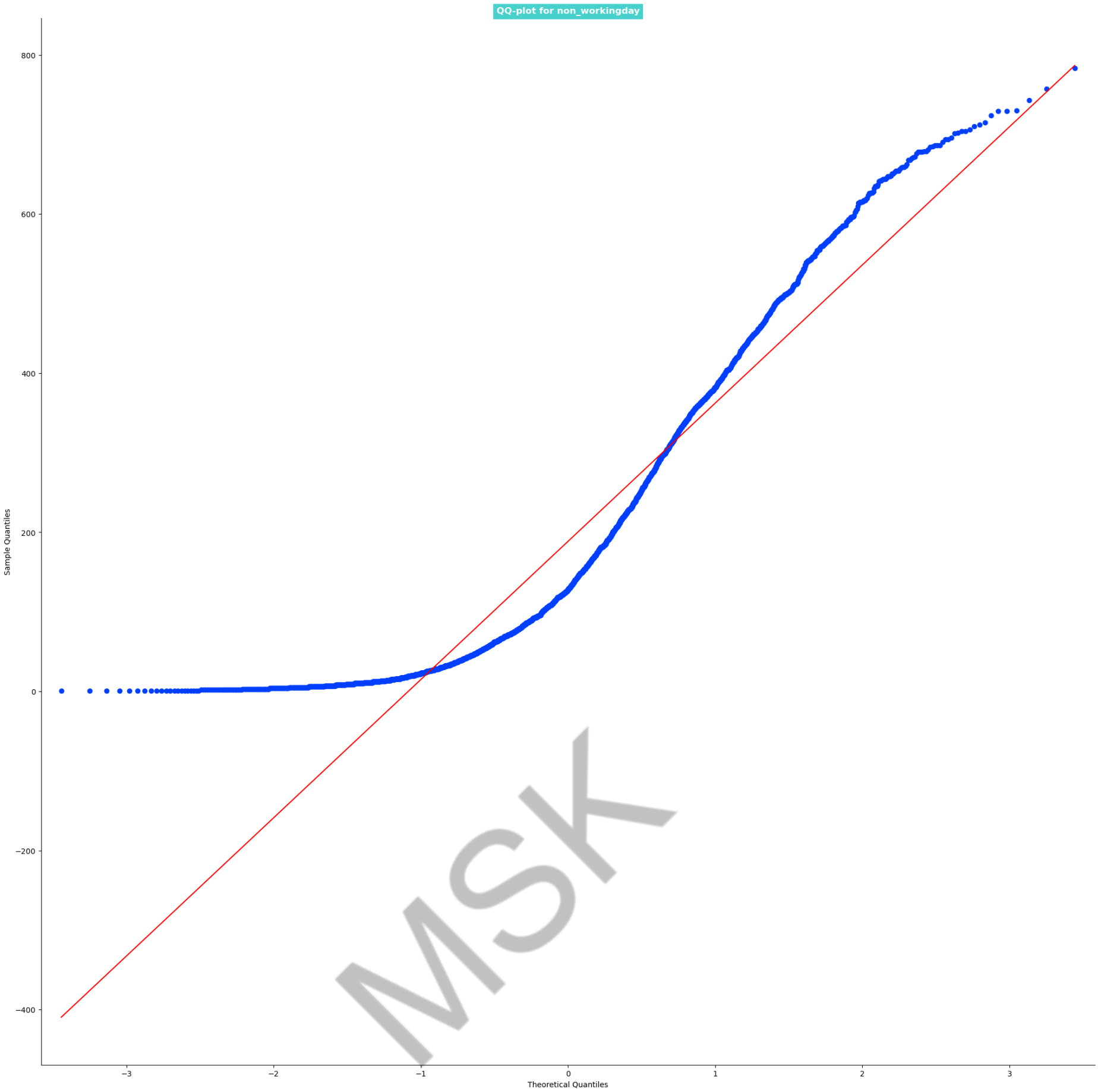


- It can be inferred from the above plot that the distributions do not follow normal distribution.

In [221…

```
sm.qqplot(workingday_yes,line='s')
plt.title('QQ-plot for workingday',fontsize=12,fontweight="bold",backgroundcolor=cp[1],color='w')

sm.qqplot(workingday_no,line='s')
plt.title('QQ-plot for non_workingday',fontsize=12,fontweight="bold",backgroundcolor=cp[1],color='w')

sns.despine()
plt.show()
```


QQ-plot for workingday
Sample Quantiles
Theoretical Quantiles
1000
800
600
400
200
0
−200
−400
−3
−2
−1
0
1
2
3
MSK



Shapiro test for workingday

> ***Null Hypothesis*** -- Ho -- Data is Gaussian
>
> ***Alternate Hypothesis*** -- Ha -- Data is not Gaussian

In [225…

```
shapiro_stat , p_val = shapiro(workingday_yes)
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 : 0.8702582120895386 , p_value : 0.0
Data does not follow normal distribution
```

In [226…

```
shapiro_stat , p_val = shapiro(workingday_no)
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
```

Levene test for variance

> ***Null Hypothesis(Ho)*** - Homogenous Variance .. Both Datas has similar variance

> ***Alternate Hypothesis(HA)*** - Non Homogenous Variance .. Both Datas has different variance

In [63]:
```
levene_stat, p_value = levene(workingday_yes,workingday_no)

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

if p_value < 0.05:
    print('The samples do not have  Homogenous Variance -- has different variance')
else:
    print('The samples have Homogenous Variance -- has similar variance')
```

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

Since the samples are not normally distributed, T-Test cannot be applied here, we can perform its non parametric equivalent test i.e., `Mann-Whitney U rank test` for two independent samples. But we will try to compare both tests....

Ttest_ind

In [233…
```
# Test statistics : Ttest for two independent samples

test_stat, p_value = ttest_ind(workingday_yes,workingday_no)

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 - Accept Ho")
    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 - Accept Ho
No.of bikes rented is same for working and non-working days
```

Non-Parametric test

MannWhitney u Rank test

In [232…
```
# Test statistics : Mann-Whitney U rank test for two independent samples

test_stat, p_value = mannwhitneyu(workingday_yes,workingday_no)

print(f'Mannwhitneyu_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 - Accept Ho")
    print('No.of bikes rented is same for working and non-working days')
```

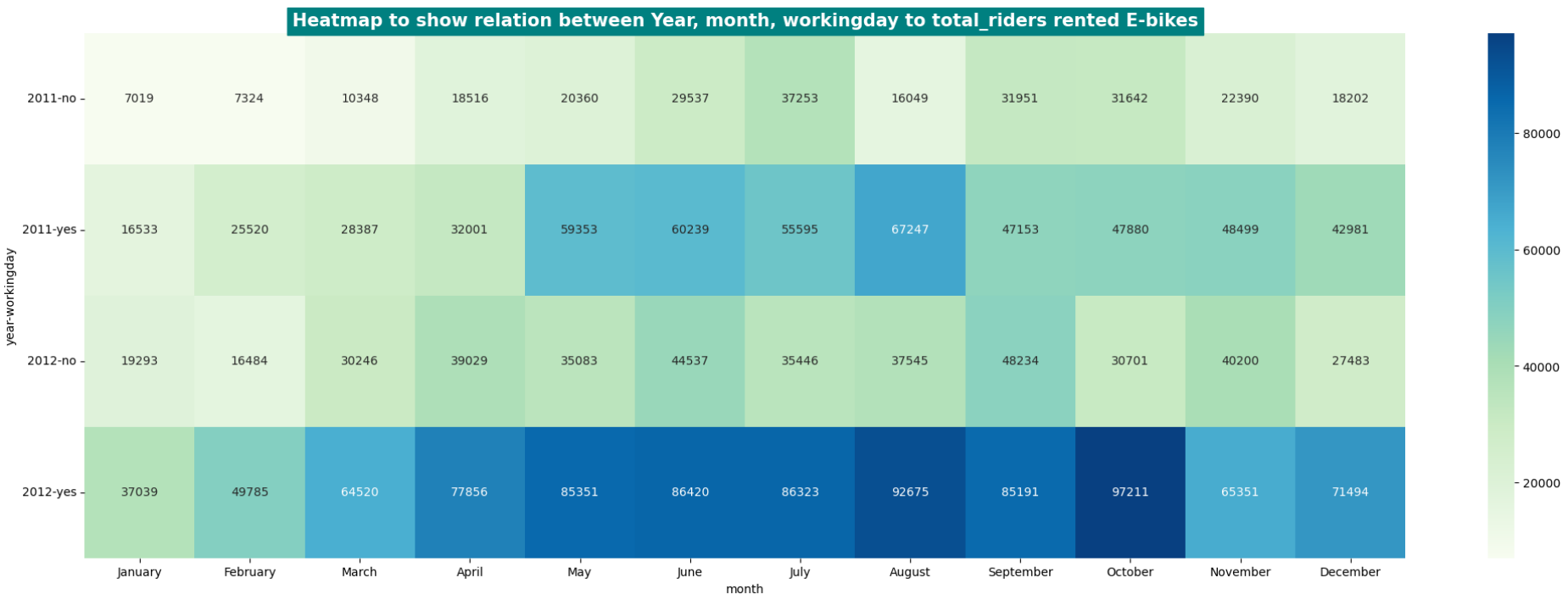
```
Mannwhitneyu_stat :  12868495.5
P-value : 0.9679139953914079
Failed to Reject Null Hypothesis - Accept Ho
No.of bikes rented is same for working and non-working days
```

🏷️ Insights:

> In Both ***ttest_ind*** & in ***MannWhitney-u-Rank*** test we find that `"Mean No.of bikes rented is same for working and non-working days"`

In [133…
```
ymwd=yd.groupby(['year','month','workingday'])['total_riders'].sum()
data= ymwd.unstack(level=[1])
data.replace(np.nan,0, inplace=True)
display(data)
plt.figure(figsize=(25,8))
sns.heatmap(data, annot=True, fmt='.6g',cmap='GnBu')
plt.title('Heatmap to show relation between Year, month, workingday to total_riders rented E-bikes',
          fontsize=15,fontweight="bold",backgroundcolor=cp[0],color='w')
plt.yticks(rotation=0)
plt.show()
```

year	workingday	month January	February	March	April	May	June	July	August	September	October	November	December
2011	**no**	7019	7324	10348	18516	20360	29537	37253	16049	31951	31642	22390	18202
	yes	16533	25520	28387	32001	59353	60239	55595	67247	47153	47880	48499	42981
2012	**no**	19293	16484	30246	39029	35083	44537	35446	37545	48234	30701	40200	27483
	yes	37039	49785	64520	77856	85351	86420	86323	92675	85191	97211	65351	71494



Check if Weather is dependent on the season

STEP-1 : Set up Null Hypothesis

> **Null Hypothesis (Ho)** - weather is independent of season
>
> **Alternate Hypothesis (Ha)** -weather is dependent of seasons.

STEP-2: Checking for basic assumptions for the hypothesis (Non-Parametric Test)

---

1. The data in the cells should be **frequencies**, or **counts** of cases.
2. The levels (or categories) of the variables are **mutually exclusive**. That is, a particular subject fits into one and only one level of each of the variables.
3. There are 2 variables, and both are measured as **categories**.
4. The **value of the cell expecteds should be 5 or more**in at least 80% of the cells, and no cell should have an expected of less than one (3).

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

The test statistic for a Chi- square test . Under Ho, the test statistic should follow **Chi-Square Distribution**.

STEP-4: Decide the kind of test.

Here we will perform the chisquare Test of independence (i.e) chi2_contingency

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

---

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

STEP-6: Compare p-value and alpha.

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

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

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

In [235…

```
ws = pd.crosstab(yd['weather'],yd['season'])
ws
```

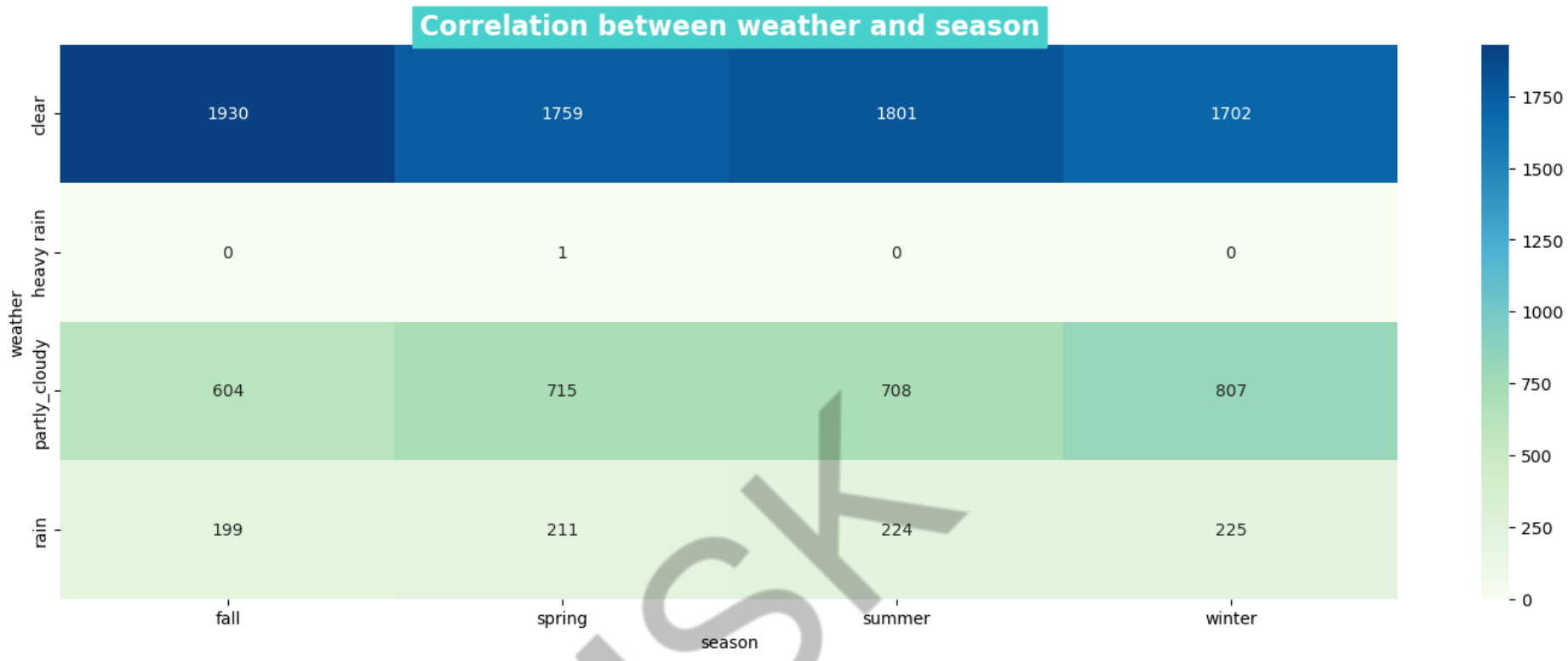
Out[235]:

season	fall	spring	summer	winter
weather				
clear	1930	1759	1801	1702
heavy rain	0	1	0	0
partly_cloudy	604	715	708	807
rain	199	211	224	225

In [240…
```
obs_value = np.array([ws.iloc[0][0:4].values ,ws.iloc[2][0:4].values ,ws.iloc[3][0:4].values])
obs_value
```

Out[240]:
```
array([[1930, 1759, 1801, 1702],
       [ 604,  715,  708,  807],
       [ 199,  211,  224,  225]], dtype=int64)
```

In [238…
```
plt.figure(figsize=(18,6))
sns.heatmap(ws, annot=True, fmt='.6g',cmap='GnBu')
plt.title('Correlation between weather and season',fontsize=16,fontweight="bold",backgroundcolor=cp[1],color='w')
plt.show()
```



In [242…
```
chi_stat , p_value , dof , expected = chi2_contingency(obs_value)

print("chi_stat : ",chi_stat)
print("p_value : ",p_value)
print("dof : ",dof)
print("expected : ",expected)

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

```
chi_stat :  46.101457310732485
p_value :  2.8260014509929403e-08
dof :  6
expected :  [[1805.76352779 1774.04869086 1805.76352779 1806.42425356]
 [ 711.55920992  699.06201194  711.55920992  711.81956821]
 [ 215.67726229  211.8892972   215.67726229  215.75617823]]
Reject Ho
Weather is dependent on season
```

🏷️ Insights:

> This chisquare test of independence confirms that **`Weather is dependent on Various seasons.`**

🚀 check if No.of bikes rented is similar or different in different weather

In [9]:
```
yd.weather.value_counts()
```

Out[9]:
```
weather
clear            7192
partly_cloudy    2834
rain              859
heavy rain          1
Name: count, dtype: int64
```

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

Specifically, it tests the null hypothesis (H0):

μ1 = μ2 = μ3 = ... = μk

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

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

STEP-1 : Set up Null Hypothesis

> **Null Hypothesis(Ho)** - The mean of bikes rented is same for across weather conditions.

(We wont be considering 'HeavyRain' as there in only 1 data point and we cannot perform a ANOVA test with a single data point for a group.)

> **Alternate Hypothesis(Ha)** - The mean number of bikes rented is different across at least two weather conditions.

STEP-2 : Checking for basic assumptions for the hypothesis

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

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

Each observations are **independent**.

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

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

F=MSR/MSE

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

STEP-4: Decide the kind of test.

We will be performing **right tailed t-test** becuz of the data right skewness

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

---

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

STEP-6: Compare p-value and alpha.

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

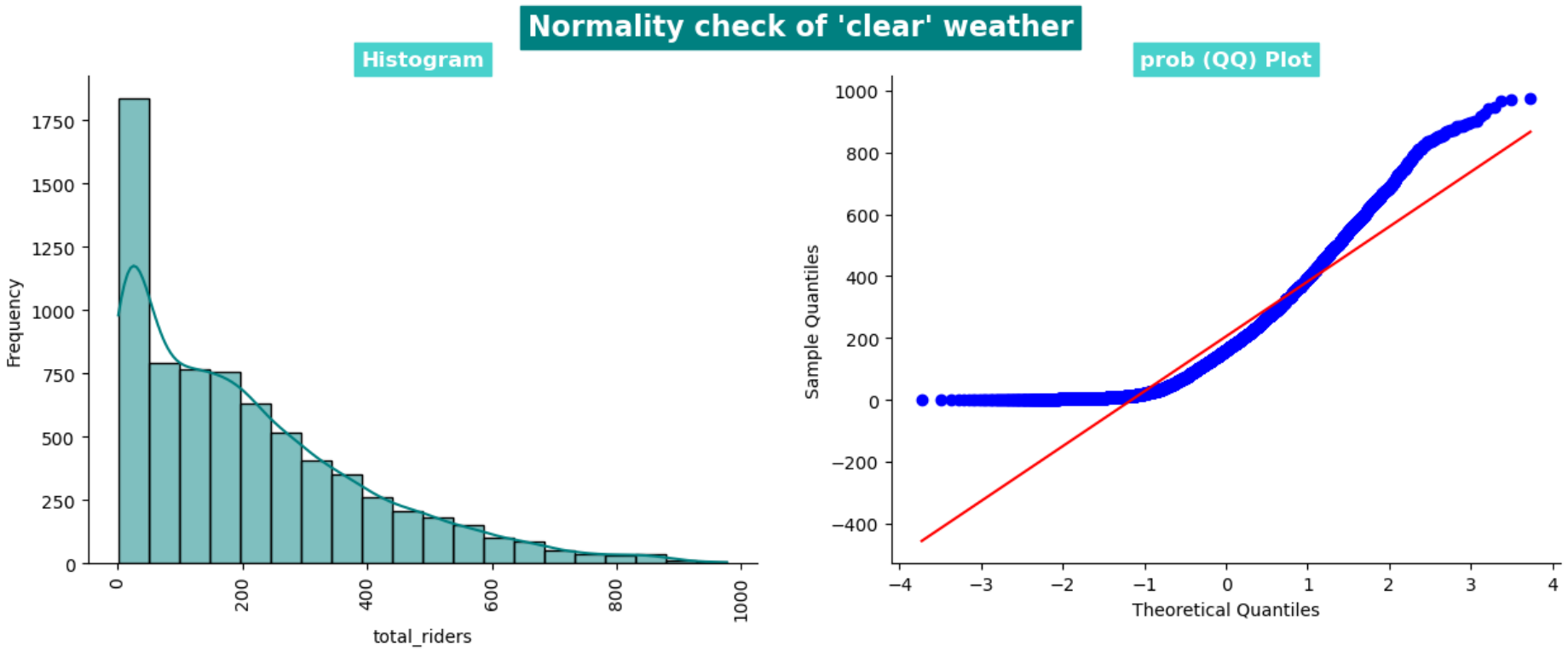
- **p-val > alpha** : Accept H0
- **p-val < alpha** : Reject H0

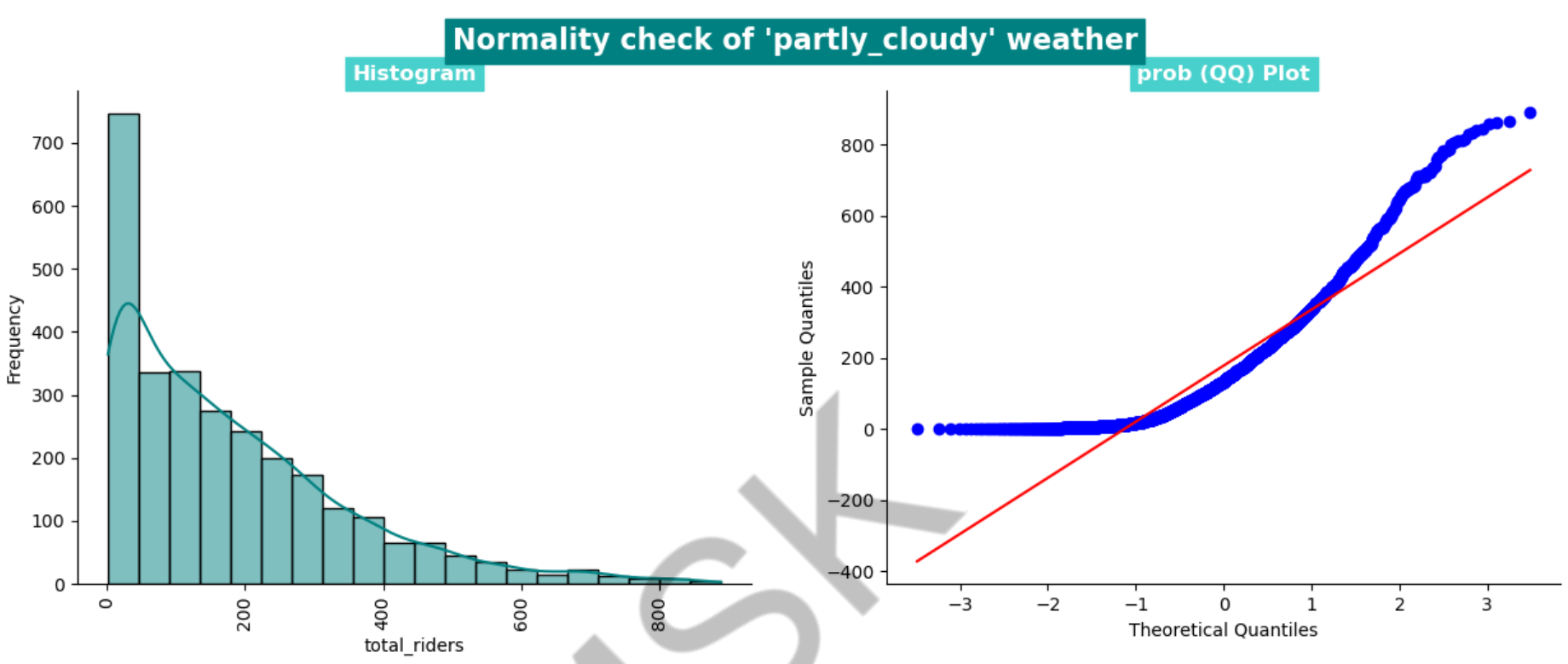
In [41]:
```
clear = yd[yd['weather']=='clear']['total_riders']
partly_cloudy = yd[yd['weather']=='partly_cloudy']['total_riders']
rain = yd[yd['weather']=='rain']['total_riders']
heavyrain = yd[yd['weather']=='heavy rain']['total_riders']
```

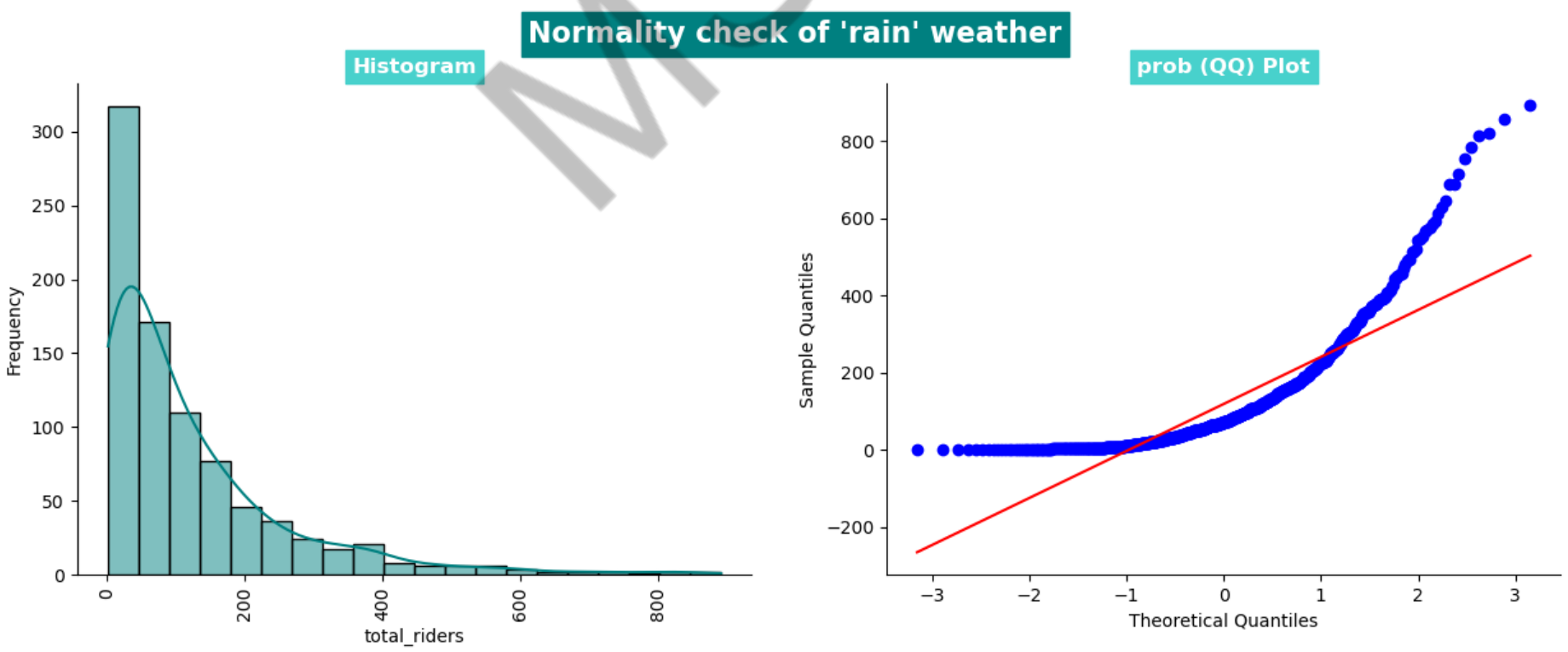
Normality Check:

Plot checks:

In [53]:
```
weather_cols = {'clear':clear, 'partly_cloudy':partly_cloudy, 'rain':rain}

for col_name,data in weather_cols.items():
    plt.figure(figsize=(15,5))
    plt.suptitle(f'Normality check of \'{col_name}\' weather',fontsize=16,fontweight="bold",backgroundcolor=cp[0],color='w')

    plt.subplot(121)
    sns.histplot(data, bins=20,kde=True,color='teal')
    plt.ylabel('Frequency')
    plt.title(f'Histogram',fontsize=12,fontweight="bold",backgroundcolor=cp[1],color='w')
    plt.xticks(rotation=90)

    plt.subplot(122)
    probplot(data, dist='norm', plot=plt)
    plt.xlabel('Theoretical Quantiles')
    plt.ylabel('Sample Quantiles')
    plt.title(f'prob (QQ) Plot',fontsize=12,fontweight="bold",backgroundcolor=cp[1],color='w')

    sns.despine()
    plt.show()
```







We already know 'Total_riders' is not a normal (gaussian) distribution. It **doesnt also satisfy the QQ - Plots.**

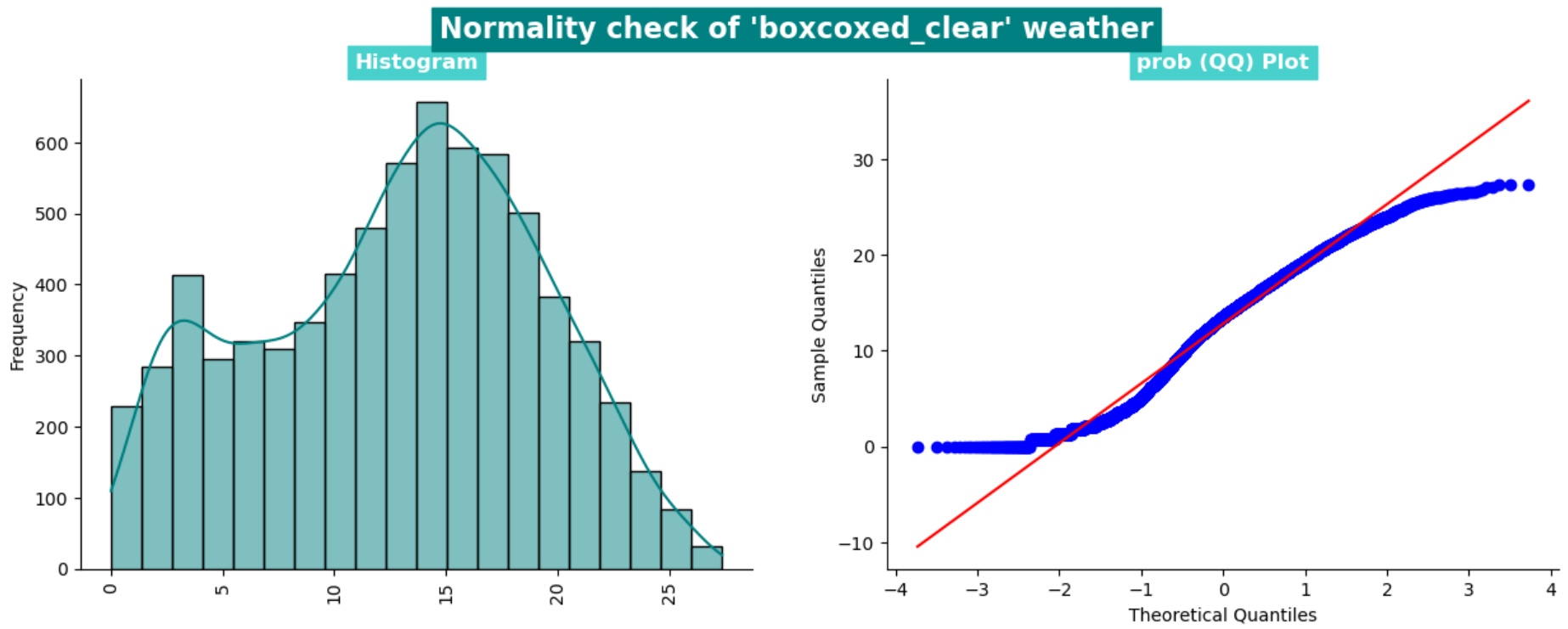
But to perform ANOVA, we need our groups to be gaussian distributed. So, we will perform **BOX-COX transform** to change the distribution of these groups to normal.

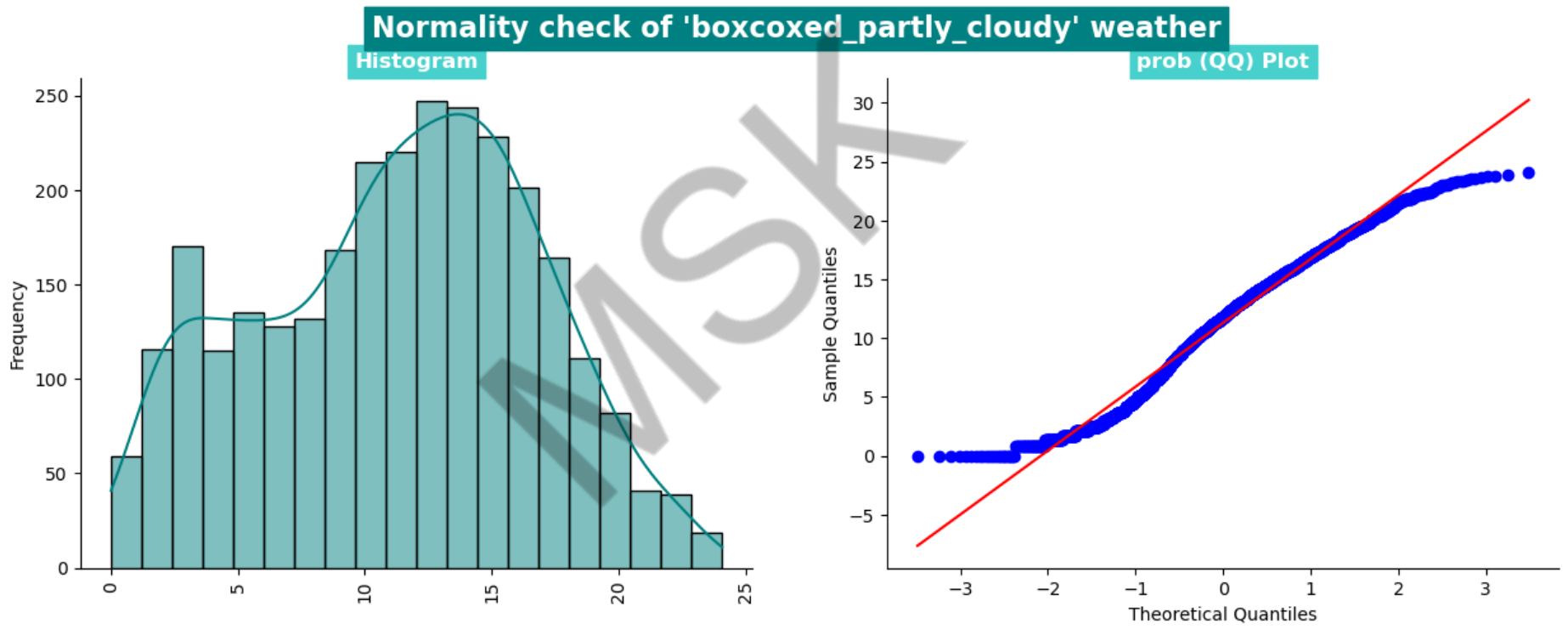
lets try ***BOX-COX Transform***

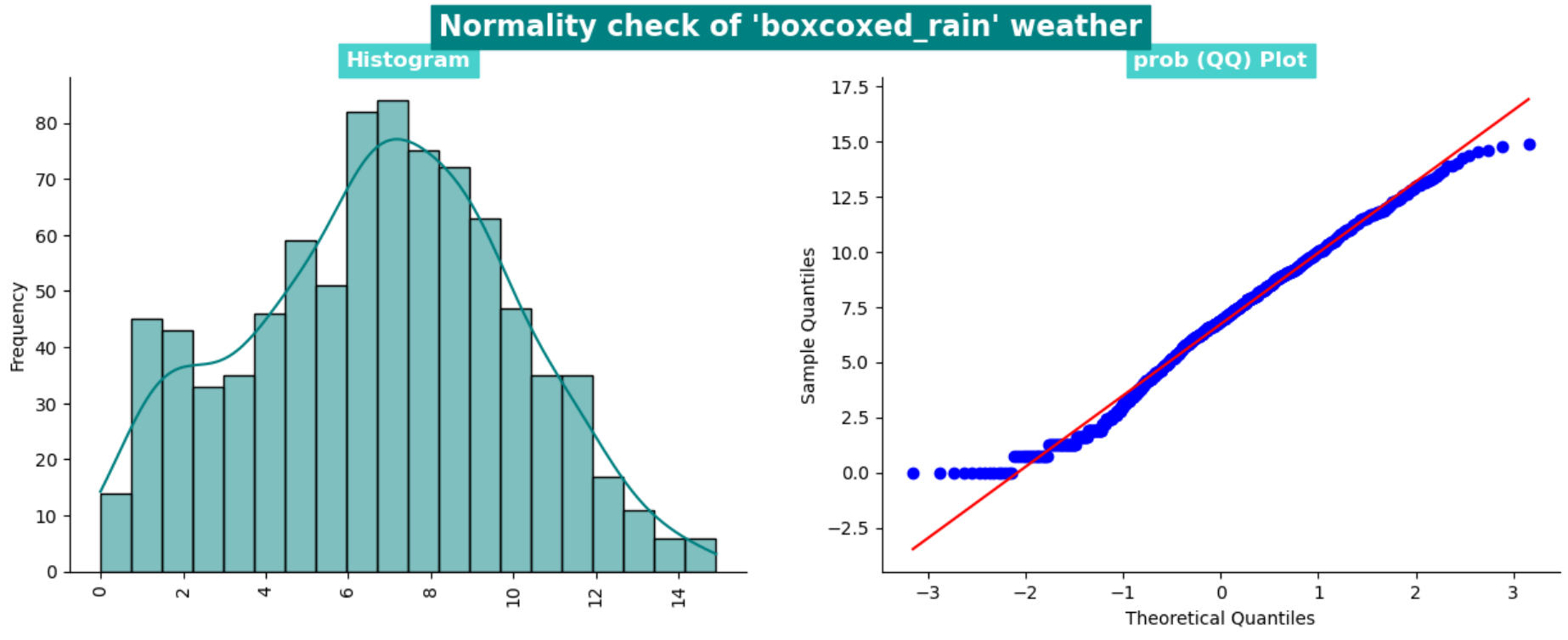
In [54]:
```
boxcoxed_clear , best_lambda = boxcox(clear)
boxcoxed_partly_cloudy , best_lambda = boxcox(partly_cloudy)
boxcoxed_rain , best_lambda = boxcox(rain)
```

In [55]:
```
weather_cols = {'boxcoxed_clear':boxcoxed_clear, 'boxcoxed_partly_cloudy':boxcoxed_partly_cloudy, 'boxcoxed_rain':boxcoxed_rain}

for col_name,data in weather_cols.items():
    plt.figure(figsize=(15,5))
    plt.suptitle(f'Normality check of \'{col_name}\' weather',fontsize=16,fontweight="bold",backgroundcolor=cp[0],color='w')

    plt.subplot(121)
```

```
    sns.histplot(data, bins=20,kde=True,color='teal')
    plt.ylabel('Frequency')
    plt.title(f'Histogram',fontsize=12,fontweight="bold",backgroundcolor=cp[1],color='w')
    plt.xticks(rotation=90)

    plt.subplot(122)
    probplot(data, dist='norm', plot=plt)
    plt.xlabel('Theoretical Quantiles')
    plt.ylabel('Sample Quantiles')
    plt.title(f'prob (QQ) Plot',fontsize=12,fontweight="bold",backgroundcolor=cp[1],color='w')

    sns.despine()
    plt.show()
```







Shapiro-Wilk Test:

In [60]:
```
weather_cols = {'clear':clear, 'partly_cloudy':partly_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 {col_name} is not Gaussian distribution')
        print()
    else:
        print(f'Data {col_name} is Gaussian distribution')
        print()
    print('-'*125)
```

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

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

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

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

In [59]:
```
weather_cols = {'boxcoxed_clear':boxcoxed_clear, 'boxcoxed_partly_cloudy':boxcoxed_partly_cloudy, 'boxcoxed_rain':boxcoxed_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 {col_name} is not Gaussian distribution')
        print()
    else:
        print(f'Data {col_name} is Gaussian distribution')
        print()
    print('-'*125)
```

```
shapiro_stat : 0.9771609306335449 , p_value : 2.061217589223373e-32
Data boxcoxed_clear is not Gaussian distribution

-----------------------------------------------------------------------------------------------------------------------------
shapiro_stat : 0.9802151918411255 , p_value : 1.9216098393369846e-19
Data boxcoxed_partly_cloudy is not Gaussian distribution

-----------------------------------------------------------------------------------------------------------------------------
shapiro_stat : 0.9877902269363403 , p_value : 1.4133181593933841e-06
Data boxcoxed_rain is not Gaussian distribution

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

Both original data and the Boxcoxed data doesn't follow the *normal distribution*, so We cannot perform Anova. We will have to go with the `Kruskel-Walis H Test` . But, we will try to do Anova as well as Kruskal test and try to compare the difference.

Levene test for variance

> ***Null Hypothesis(Ho)*** - Homogenous Variance .. Both Datas has similar variance
>
> ***Alternate Hypothesis(HA)*** - Non Homogenous Variance .. Both Datas has different variance

In [85]:
```
levene_stat, p_value = levene(clear,partly_cloudy,rain)

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

if p_value < 0.05:
    print('The samples do not have  Homogenous Variance -- has different variance')
else:
    print('The samples have Homogenous Variance -- has similar variance')
```

```
Levene_stat :  81.67574924435011
p-value :  6.198278710731511e-36
The samples do not have  Homogenous Variance -- has different variance
```

KRUSKAL_WALIS H TEST

In [66]:
```
alpha = 0.05
test_stat, p_value = kruskal(clear,partly_cloudy,rain)
print('Test Statistic =', test_stat)
print('p value =', p_value)

if p_value < alpha:
    print('Reject Null Hypothesis')
    print("The mean number of bikes rented is different across at weather conditions")
else:
    print('Failed to reject Null Hypothesis')
    print("The mean of bikes rented is same for across weather conditions.")
```

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

ANOVA

In [68]:
```
test = ols('total_riders ~ C(weather)',data=yd).fit()
sm.stats.anova_lm(test,typ=2)
```

Out[68]:

	sum_sq	df	F	PR(>F)
C(weather)	6.338070e+06	3.0	65.530241	5.482069e-42
Residual	3.508348e+08	10882.0	NaN	NaN

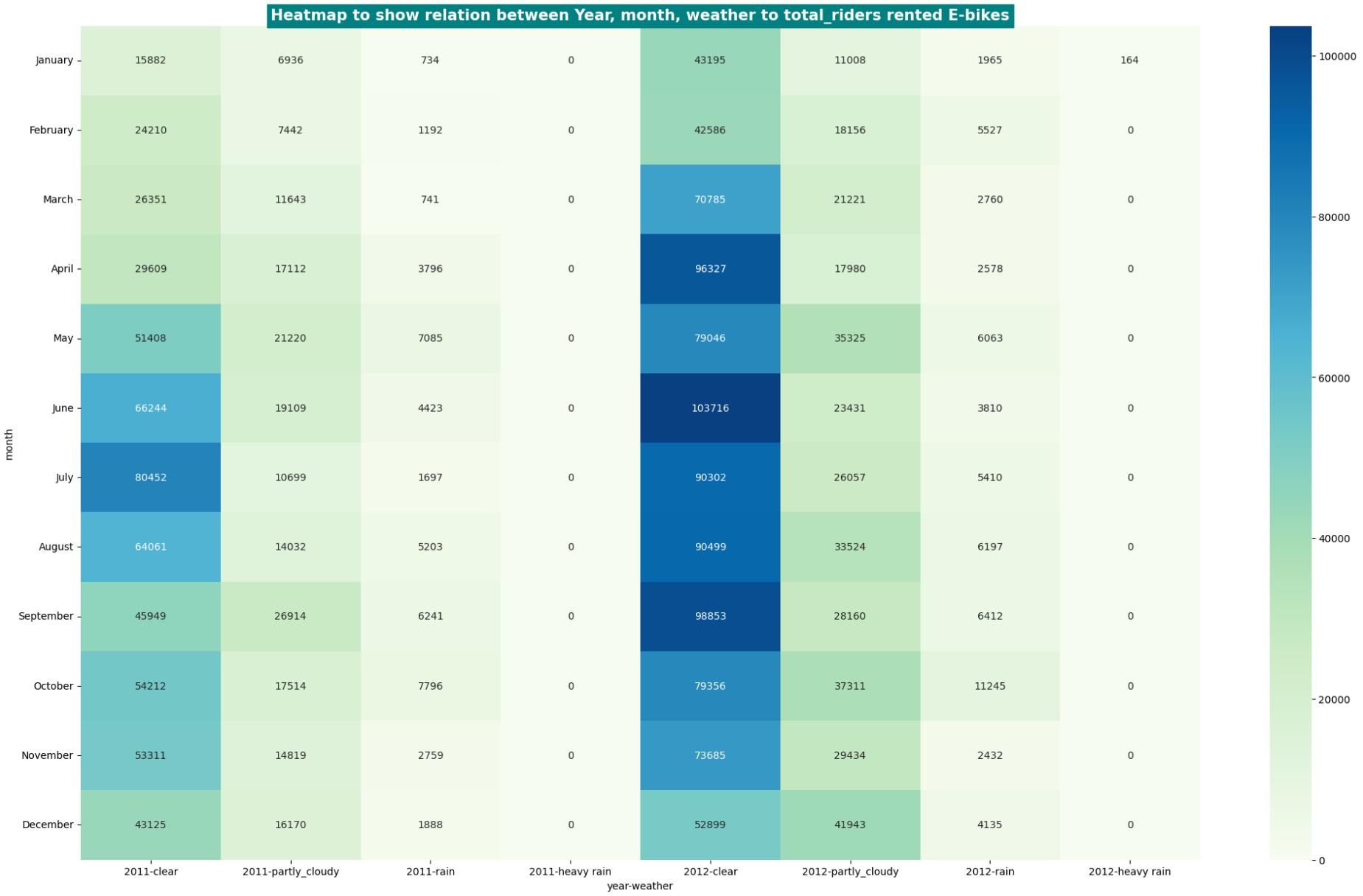
In [69]:
```
test_stat, p_value = f_oneway(clear,partly_cloudy,rain)
print('Test Statistic =', test_stat)
print('p value =', p_value)
```

```
Test Statistic = 98.28356881946706
p value = 4.976448509904196e-43
```

In [119…
```
month_order = ['January', 'February', 'March', 'April', 'May',
               'June', 'July', 'August', 'September',
               'October', 'November', 'December']
yd['month']= pd.Categorical(yd['month'],categories=month_order, ordered=True)
```

In [121…
```
ymw=yd.groupby(['year','month','weather'])['total_riders'].sum()
data= ymw.unstack(level=[0,2])
data.replace(np.nan,0, inplace=True)
display(data)
plt.figure(figsize=(25,15))
sns.heatmap(data, annot=True, fmt='.6g',cmap='GnBu')
plt.title('Heatmap to show relation between Year, month, weather to total_riders rented E-bikes',
          fontsize=15,fontweight="bold",backgroundcolor=cp[0],color='w')
plt.yticks(rotation=0)
plt.show()
```

year	2011				2012			
weather	clear	partly_cloudy	rain	heavy rain	clear	partly_cloudy	rain	heavy rain
month								
January	15882	6936	734	0	43195	11008	1965	164
February	24210	7442	1192	0	42586	18156	5527	0
March	26351	11643	741	0	70785	21221	2760	0
April	29609	17112	3796	0	96327	17980	2578	0
May	51408	21220	7085	0	79046	35325	6063	0
June	66244	19109	4423	0	103716	23431	3810	0
July	80452	10699	1697	0	90302	26057	5410	0
August	64061	14032	5203	0	90499	33524	6197	0
September	45949	26914	6241	0	98853	28160	6412	0
October	54212	17514	7796	0	79356	37311	11245	0
November	53311	14819	2759	0	73685	29434	2432	0
December	43125	16170	1888	0	52899	41943	4135	0



📌 Insights:

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

🚀 check if No.of bikes rented is similar or different in different Seasons

In [74]:
```
yd.season.unique()
```

Out[74]:
```
['spring', 'summer', 'fall', 'winter']
Categories (4, object): ['spring', 'summer', 'fall', 'winter']
```

In [76]:
```
yd.groupby('season')['total_riders'].describe()
```

Out[76]:

season	count	mean	std	min	25%	50%	75%	max
spring	2686.0	116.343261	125.273974	1.0	24.0	78.0	164.0	801.0
summer	2733.0	215.251372	192.007843	1.0	49.0	172.0	321.0	873.0
fall	2733.0	234.417124	197.151001	1.0	68.0	195.0	347.0	977.0
winter	2734.0	198.988296	177.622409	1.0	51.0	161.0	294.0	948.0

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

STEP-1 : Set up Null Hypothesis

> **Null Hypothesis(Ho)** - 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.

STEP-2 : Checking for basic assumptions for the hypothesis

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

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

Each observations are **independent**.

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

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

F=MSR/MSE

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

STEP-4: Decide the kind of test.

We will be performing **right tailed t-test** becuz of the data right skewness

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

---

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

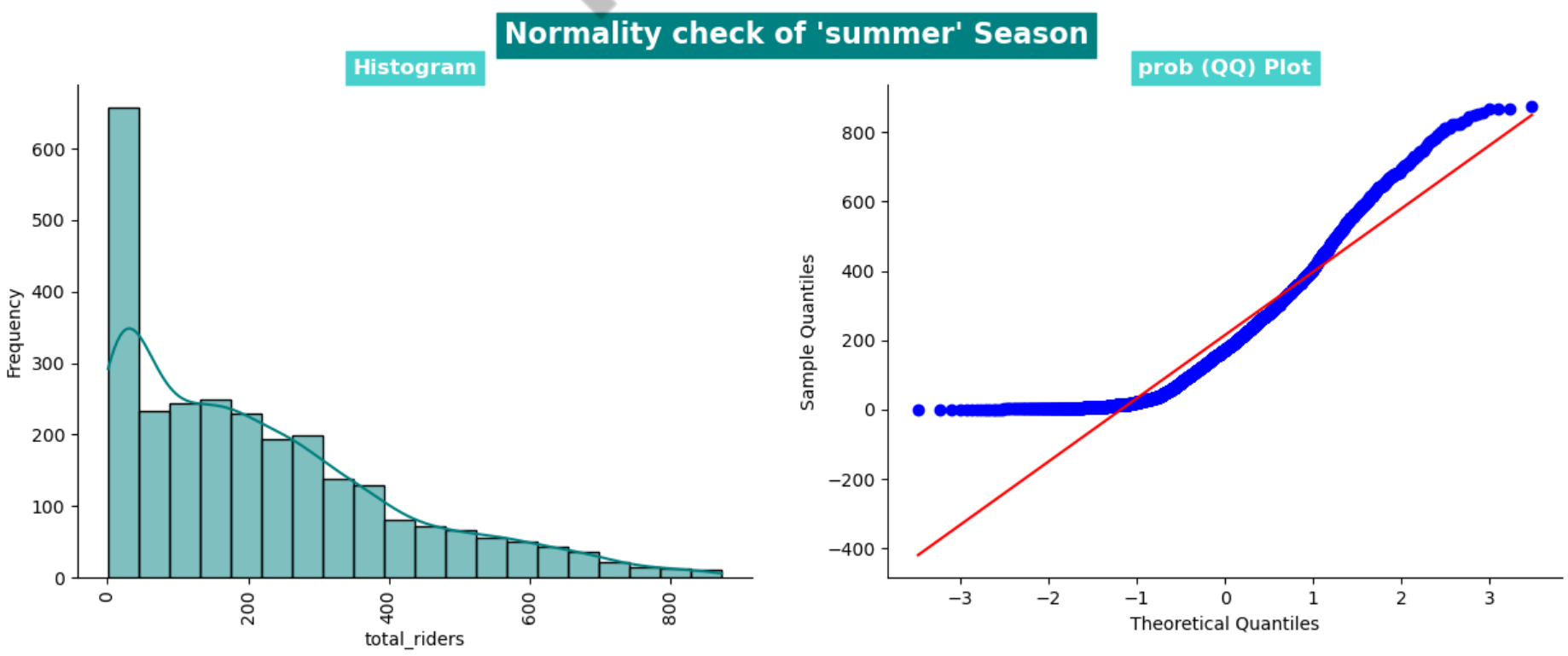
STEP-6: Compare p-value and alpha.

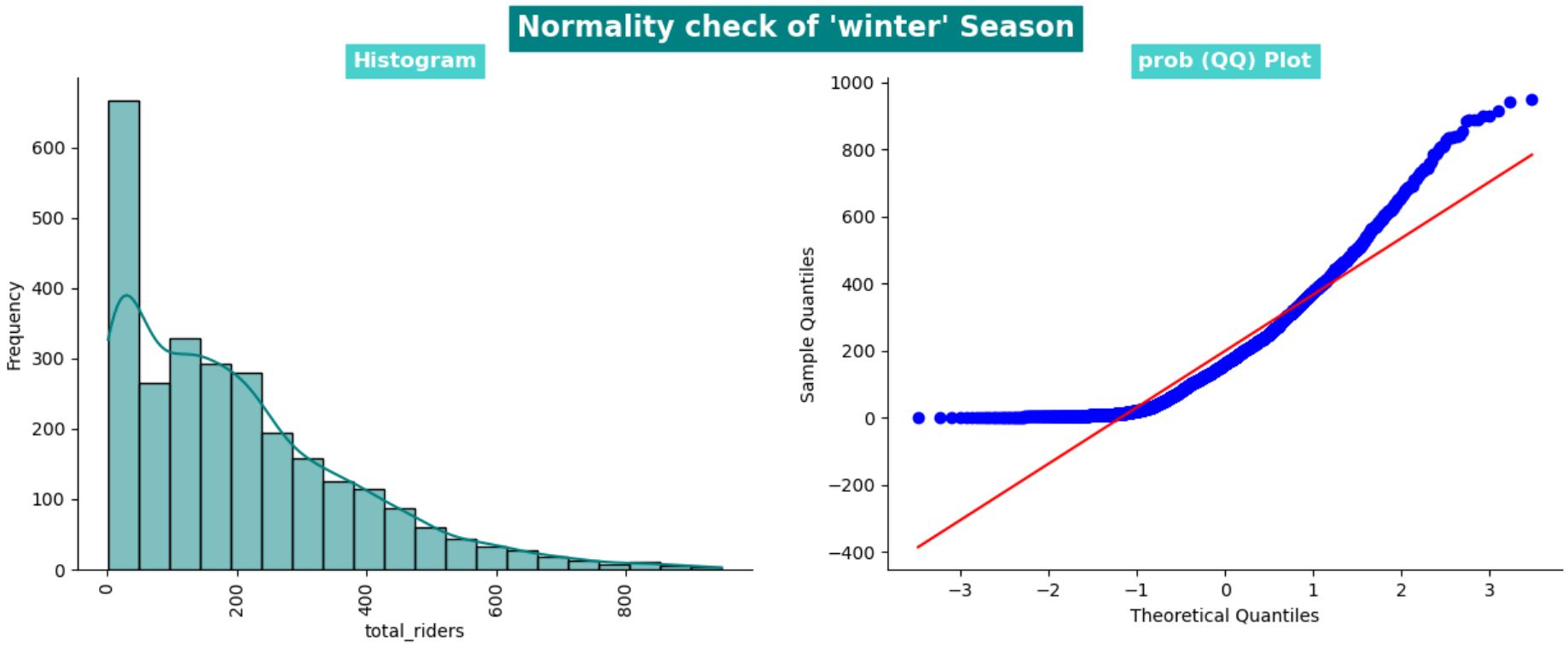
Based on p-value, we will accept or reject H0.

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

In [77]:
```
summer = yd[yd['season']=='summer']['total_riders']
winter = yd[yd['season']=='winter']['total_riders']
fall = yd[yd['season']=='fall']['total_riders']
spring = yd[yd['season']=='spring']['total_riders']
```

In [78]:
```
season_cols = {'summer':summer , 'winter':winter , 'fall':fall, 'spring':spring}

for col_name,data in season_cols.items():
    plt.figure(figsize=(15,5))
    plt.suptitle(f'Normality check of \'{col_name}\' Season',fontsize=16,fontweight="bold",backgroundcolor=cp[0],color='w')

    plt.subplot(121)
    sns.histplot(data, bins=20,kde=True,color='teal')
    plt.ylabel('Frequency')
    plt.title(f'Histogram',fontsize=12,fontweight="bold",backgroundcolor=cp[1],color='w')
    plt.xticks(rotation=90)

    plt.subplot(122)
    probplot(data, dist='norm', plot=plt)
    plt.xlabel('Theoretical Quantiles')
    plt.ylabel('Sample Quantiles')
    plt.title(f'prob (QQ) Plot',fontsize=12,fontweight="bold",backgroundcolor=cp[1],color='w')

    sns.despine()
    plt.show()
```



We already know 'Total_riders' is not a normal (gaussian) distribution. It **doesnt also satisfy the QQ - Plots.**

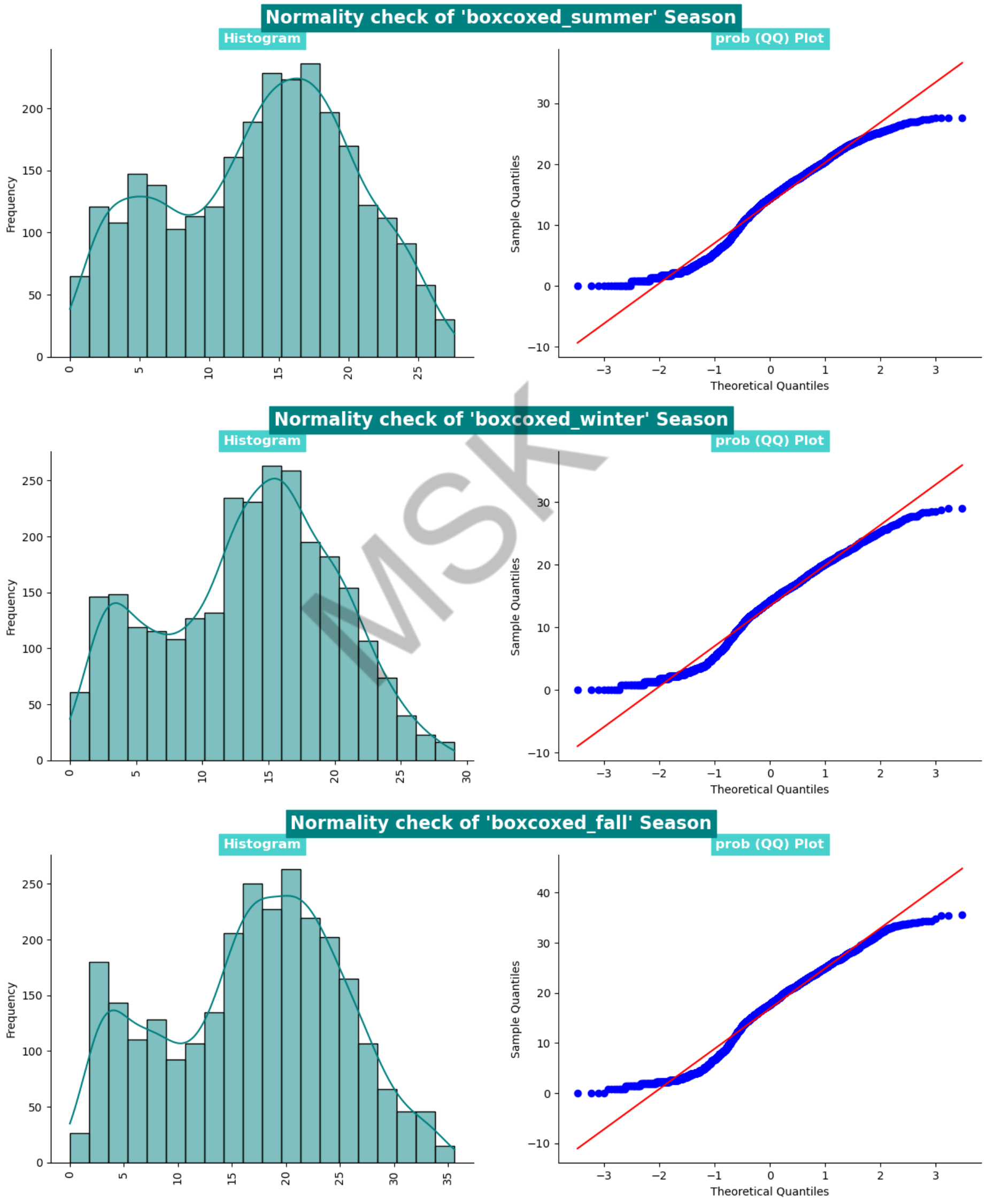
But to perform ANOVA, we need our groups to be gaussian distributed. So, we will perform **BOX-COX transform** to change the distribution of these groups to normal.

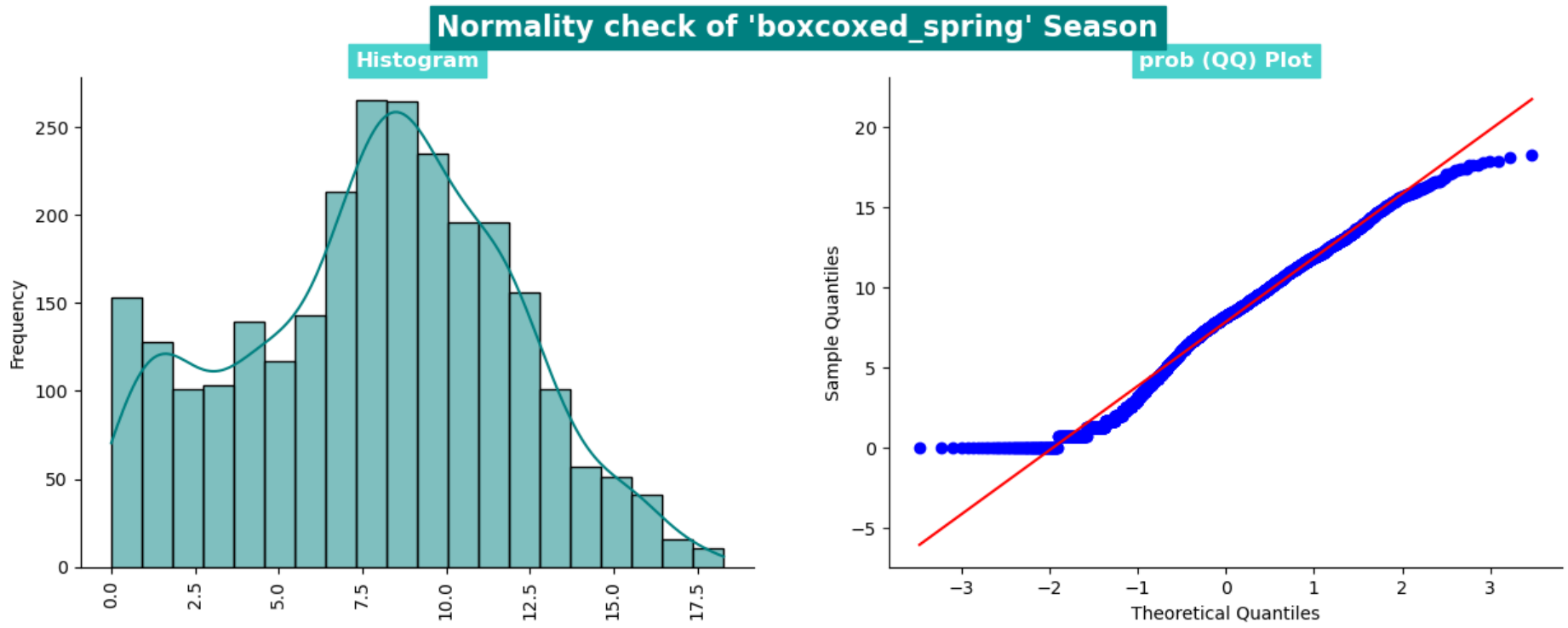
lets try ***BOX-COX Transformation***

In [80]:
```
boxcoxed_summer , best_lambda = boxcox(summer)
boxcoxed_winter , best_lambda = boxcox(winter)
boxcoxed_fall , best_lambda = boxcox(fall)
boxcoxed_spring , best_lambda = boxcox(spring)
```

In [83]:
```
season_cols = {'boxcoxed_summer':boxcoxed_summer, 'boxcoxed_winter':boxcoxed_winter,
               'boxcoxed_fall':boxcoxed_fall, 'boxcoxed_spring':boxcoxed_spring}

for col_name,data in season_cols.items():
    plt.figure(figsize=(15,5))
    plt.suptitle(f'Normality check of \'{col_name}\' Season',fontsize=16,fontweight="bold",backgroundcolor=cp[0],color='w')
```

```
    plt.subplot(121)
    sns.histplot(data, bins=20,kde=True,color='teal')
    plt.ylabel('Frequency')
    plt.title(f'Histogram',fontsize=12,fontweight="bold",backgroundcolor=cp[1],color='w')
    plt.xticks(rotation=90)

    plt.subplot(122)
    probplot(data, dist='norm', plot=plt)
    plt.xlabel('Theoretical Quantiles')
    plt.ylabel('Sample Quantiles')
    plt.title(f'prob (QQ) Plot',fontsize=12,fontweight="bold",backgroundcolor=cp[1],color='w')

    sns.despine()
    plt.show()
```





Shapiro-Wilk Test:

In [82]:
```
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

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

In [84]:
```
season_cols = {'boxcoxed_summer':boxcoxed_summer, 'boxcoxed_winter':boxcoxed_winter,
               'boxcoxed_fall':boxcoxed_fall, 'boxcoxed_spring':boxcoxed_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 : 0.9730892181396484 , p_value : 2.7910560207702335e-22
Data boxcoxed_summer is not Gaussian distribution

-----------------------------------------------------------------------------------------------------------------------------
shapiro_stat : 0.9763026833534241 , p_value : 6.342709865441161e-21
Data boxcoxed_winter is not Gaussian distribution

-----------------------------------------------------------------------------------------------------------------------------
shapiro_stat : 0.9733710289001465 , p_value : 3.6319999210910884e-22
Data boxcoxed_fall is not Gaussian distribution

-----------------------------------------------------------------------------------------------------------------------------
shapiro_stat : 0.9828581213951111 , p_value : 1.7082116755999925e-17
Data boxcoxed_spring is not Gaussian distribution

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

Both **original data and the Boxcoxed data** doesn't follow the *normal distribution*, so We cannot perform **Anova**. We will have to go with the `Kruskel-Walis H Test`. But, we will try to do Anova as well as Kruskal test and try to compare the difference.

Levene test for variance

> ***Null Hypothesis(Ho)*** - Homogenous Variance .. Both Datas has similar variance
>
> ***Alternate Hypothesis(HA)*** - Non Homogenous Variance .. Both Datas has different variance

In [86]:
```
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 do not have  Homogenous Variance -- has different variance')
else:
    print('The samples have Homogenous Variance -- has similar variance')
```

```
Levene_stat :  187.7706624026276
p-value :  1.0147116860043298e-118
The samples do not have  Homogenous Variance -- has different variance
```

KRUSKAL_WALIS H TEST

In [92]:
```
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 mean number of bikes rented is different across seasons")
else:
    print('Failed to reject Null Hypothesis')
    print("The mean of bikes rented is same for across seasons.")
```

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

ANOVA

In [88]:
```
test = ols('total_riders ~ C(season)',data=yd).fit()
sm.stats.anova_lm(test,typ=2)
```

Out[88]:

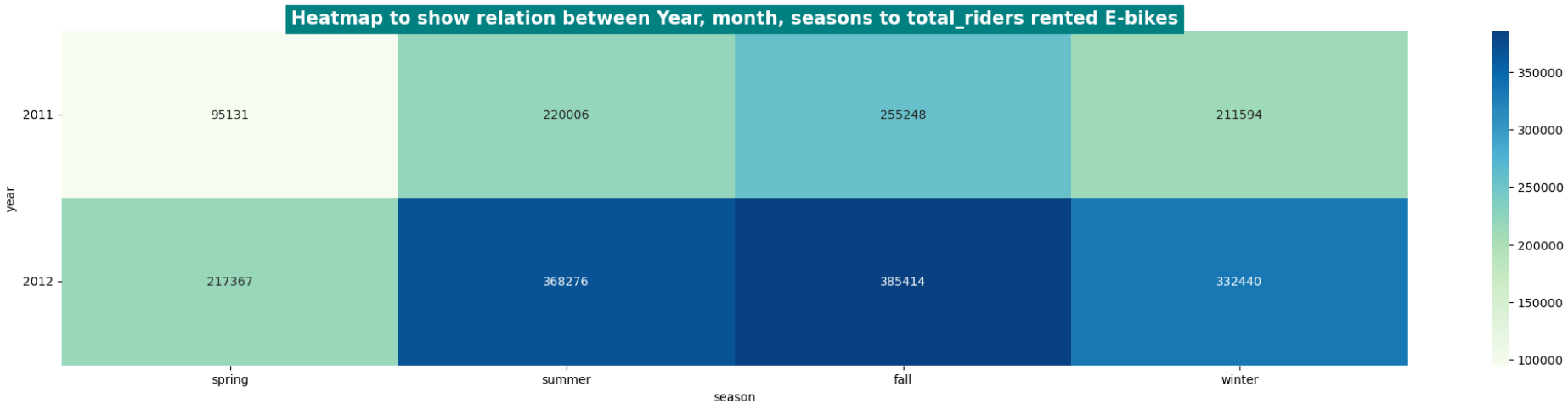
	sum_sq	df	F	PR(>F)
C(season)	2.190083e+07	3.0	236.946711	6.164843e-149
Residual	3.352721e+08	10882.0	NaN	NaN

In [91]:
```
test_stat, p_value = f_oneway(summer,winter,fall,spring)
print('Test Statistic =', test_stat)
print('p value =', p_value)

if p_value < 0.05:
    print('Reject Null Hypothesis')
    print("The mean number of bikes rented is different across seasons")
else:
    print('Failed to reject Null Hypothesis')
    print("The mean of bikes rented is same for across seasons")
```

```
Test Statistic = 236.94671081032104
p value = 6.164843386499654e-149
Reject Null Hypothesis
The mean number of bikes rented is different across seasons
```

In [128…
```
yms=yd.groupby(['year','season'])['total_riders'].sum()
data= yms.unstack(level=[1])
data.replace(np.nan,0, inplace=True)
display(data)
plt.figure(figsize=(25,5))
sns.heatmap(data, annot=True, fmt='.6g',cmap='GnBu')
plt.title('Heatmap to show relation between Year, month, seasons to total_riders rented E-bikes',
          fontsize=15,fontweight="bold",backgroundcolor=cp[0],color='w')
plt.yticks(rotation=0)
plt.show()
```

season	spring	summer	fall	winter
year				
2011	95131	220006	255248	211594
2012	217367	368276	385414	332440



🏷️ Insights:

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

🤔 💭 Buisness Insights 💭 🤔

> **Seasonal Patterns**
>
> - Maximum bike rentals occur during summer, while the minimum is observed in winter.
>
> **Conditions Impact**
>
> - Clear weather is associated with the highest bike rental counts, whereas rentals sharply decrease in rain, thunderstorm, snow, or fog.
> - Humidity, windspeed, temperature and weather are correlated with season and impacts the count of cycles rented.
>
> **Temperature Influence**
>
> - Lower temperatures correspond to lower bike rentals, and demand rises with increasing temperatures.
>
> **Time-of-Day Trends**
>
> - Bike rentals peak during the day, decline through the night, indicating a pattern fluctuation.
>
> **Holiday and Working Day Dynamics**
>
> - Less rentals on holidays and weekends, with a demand increase on non-working days. However, the overall count on working and non-holiday days are similar.
>
> **User Type Behavior**
>
> - Casual riders dominate on weekends, while registered users are more active on working days.
>
> **Yearly Growth and User Composition**
>
> - The hourly rental count shows impressive annual growth from 2011 to 2012.
> - Approximately 19% of users are casual, and 81% are registered.
>
> **Monthly and Daily Usage Patterns**
>
> - Notable seasonal patterns, with peak demand in spring and summer, and a decline in fall and winter.
> - January to March sees the lowest rental counts, and a distinctive daily trend shows peak usage during the afternoon.
>
> **Weather Impact on Usage**
>
> - Clear and partly_cloudy weather correlates with higher rental counts, while extreme weather conditions have limited data representation.
>
> **Correlations**
>
> - Temperature and feeling temperature exhibit a strong positive correlation.
> - Registered Users and Total_riders exhibit a strong positive correlation as well.
> - Limited correlation observed between weather-related factors and bike rental counts.
>
> **Statistical Significance**
>
> - ANOVA tests confirm statistically significant impacts of seasons and weather on bike rentals.
> - Working days vs. holidays have limited impact according to a 2-sample t-test.
> - ChiSquare confirms that the Weather is dependent on the Seasons.

🌟⚡ Business Recommendations ⚡🌟

Strategic Seasonal Marketing

- Leverage seasonal patterns by implementing targeted marketing during peak seasons (spring and summer).
- Introduce seasonal incentives and exclusive packages to drive higher demand.

Dynamic Time-based Pricing

- Optimize resource utilization by implementing dynamic time-based pricing.
- Adjust rental rates to encourage bike usage during off-peak hours, enhancing accessibility.

Enhanced Weather Data Collection

- Collect more data on extreme weather conditions to understand user behavior.
- Consider specialized bike models or safety measures for different weather scenarios.

Weather-sensitive Promotions

- Launch weather-specific promotional campaigns focusing on clear and partly cloudy conditions.
- Introduce weather-based discounts to attract more users during favorable weather.

User-Centric Segmentation

- Tailor marketing strategies for registered and casual users.
- Offer loyalty programs and personalized incentives for registered users, highlighting occasional use benefits for casual users.

Optimized Inventory Management

- Fine-tune inventory levels based on monthly demand patterns.
- Avoid overstocking during low-demand months and ensure sufficient bikes during peak periods.

Customer Comfort and Convenience

- Provide amenities like umbrellas or rain jackets to enhance customer comfort.
- Elevate the overall biking experience, contributing to positive customer feedback.

Collaboration with Weather Services

- Partner with weather services for real-time updates in marketing campaigns.
- Showcase ideal biking conditions through app integration, appealing to weather-specific preferences.

Seasonal Bike Maintenance

- Conduct thorough seasonal bike maintenance to prevent breakdowns.
- Ensure optimal bike performance, enhancing customer satisfaction.

Customer Feedback and Reviews

- Encourage customer feedback to identify areas for improvement.
- Customize services based on insights, exceeding customer expectations.

Strategic Social Media Marketings

- Utilize social media platforms for strategic promotions and engagement.
- Share diverse biking experiences, customer testimonials, and run targeted advertising campaigns.

These recommendations aim to maximize `Yulu`'s market reach, optimize operations, and enhance the overall customer experience. Implementing these strategies will position Yulu as a customer-centric and environmentally-conscious brand in the electric bike rental industry.

🤝 Inshort:

In summary, the pivotal factors influencing bike rentals are the season and weather, overshadowing the impact of working and non-working days. To optimize operations, emphasis should be placed on expanding bike parking zones during months excluding winter (Nov-Feb). Considering the decline in bike rentals during winter, exploring alternatives like electric car rentals for comfortable cold-weather commuting is recommended.