Close

```
# <h1 align='center'> <font color='royalbl
Business CaseStudy🚴‍♂️🚲</font> </font></h1
<h1 align='center'><font color='royalblue'
font> </font></h1>
<h2 align='right'>Analysed by : <font colo

<kbd>![y-pic.jpg](attachment:fa91c7cb-4810
```

🚲🚴‍♂️ YULU_Bikes - Business CaseStudy

🚴‍♀️🚲

Hypothesis Testing

Analysed by : **Aakash**

y-pic.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

```
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')
```

```
yulu_data = pd.read_csv('bike_sharing.csv')
```

```
yd = yulu_data.copy()
yd
```

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

Exploration of data :

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

```
yd.shape
```

```
(10886, 12)
```

```
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

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

```
yd['datetime'].dtype
```

```
dtype('<M8[ns]')
```

```
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'})
```

```
yd.skew(numeric_only = True)
```

```
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

```
yd.describe(include='all').T
```

	count	unique	top	freq	mean	min	25%	
datetime	10886	NaN	NaN	NaN	2011-12-27 05:56:22.399411968	2011-01-01 00:00:00	2011-07-02 07:15:00	2 0 20:3
season	10886.0	NaN	NaN	NaN	2.506614	1.0	2.0	
holiday	10886.0	NaN	NaN	NaN	0.028569	0.0	0.0	
workingday	10886.0	NaN	NaN	NaN	0.680875	0.0	0.0	
weather	10886.0	NaN	NaN	NaN	1.418427	1.0	1.0	
temp	10886.0	NaN	NaN	NaN	20.23086	0.82	13.94	
atemp	10886.0	NaN	NaN	NaN	23.655084	0.76	16.665	2
humidity	10886.0	NaN	NaN	NaN	61.88646	0.0	47.0	
windspeed	10886.0	NaN	NaN	NaN	12.799395	0.0	7.0015	12
casual	10886.0	NaN	NaN	NaN	36.021955	0.0	4.0	
registered	10886.0	NaN	NaN	NaN	155.552177	0.0	36.0	1
total_riders	10886.0	NaN	NaN	NaN	191.574132	1.0	42.0	1
year	10886.0	NaN	NaN	NaN	2011.501929	2011.0	2011.0	20
month	10886	12	May	912	NaN	NaN	NaN	

📜📜Duplicate Detection

```
yd[yd.duplicated()]
```

datetime	season	holiday	workingday	weather	temp	atemp	humidity	windspee

🏷️ Insights

- The dataset does not contain any duplicates.

🔸❓Null Detection

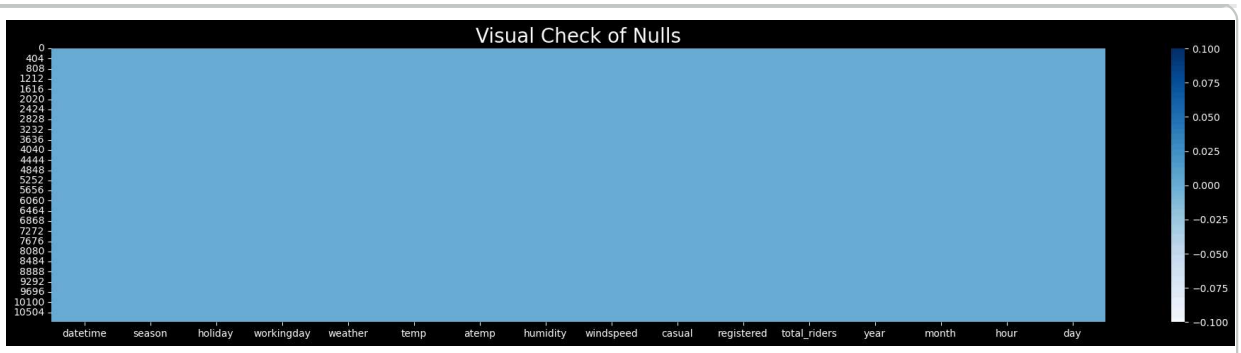
```
yd.isna().any()
```

```
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
```

```
yd.isna().sum()
```

```
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
```

```
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()
```




```
print(list(yd.columns))
```

```
['datetime', 'season', 'holiday', 'workingday', 'weather', 'temp', 'atemp', 'hum
```

Changing the Datatype of Columns

```
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
```

```
yd.describe(include='category').T
```

	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

```
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 rai
yd["weather"] = yd["weather"].map(lambda x: weather_mapping[x])
```

```
yd.sample(7)
```

	datetime	season	holiday	workingday	weather	temp	atemp	humidity
9995	2012-11-01 20:00:00	winter	no	yes	partly_cloudy	15.58	19.695	54
8567	2012-07-18 08:00:00	fall	no	yes	clear	31.98	37.120	62
9450	2012-09-17 03:00:00	fall	no	yes	clear	20.50	24.240	77
	2011-09-							

```
#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
```

```
yd.columns
```

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

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

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

```
yd.datetime.min()
```

```
Timestamp('2011-01-01 00:00:00')
```

```
yd.datetime.max()
```

```
Timestamp('2012-12-19 23:00:00')
```

```
yd.datetime.max() - yd.datetime.min()
```

```
Timedelta('718 days 23:00:00')
```

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

```
yd.sample(6)
```

	datetime	season	holiday	workingday	weather	temp	atemp	humidity
2061	2011-05-12 19:00:00	summer	no	yes	partly_cloudy	24.60	30.305	64
4937	2011-11-18 19:00:00	winter	no	yes	clear	11.48	13.635	41
6387	2012-03-03 09:00:00	spring	no	no	rain	15.58	19.695	87

```
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

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

---

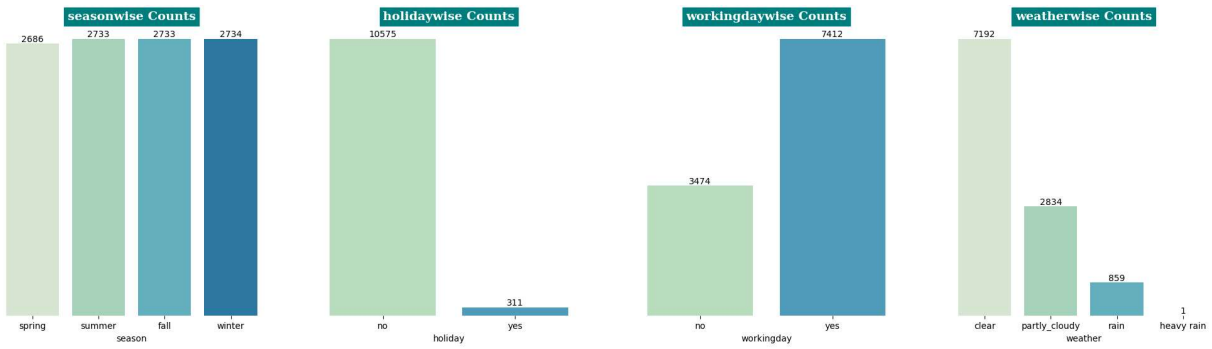
	weather	proportion
0	clear	66.07
1	partly_cloudy	26.03
2	rain	7.89
3	heavy rain	0.01

📊 Visual analysis

📈 Univariate and Bivariate graphs

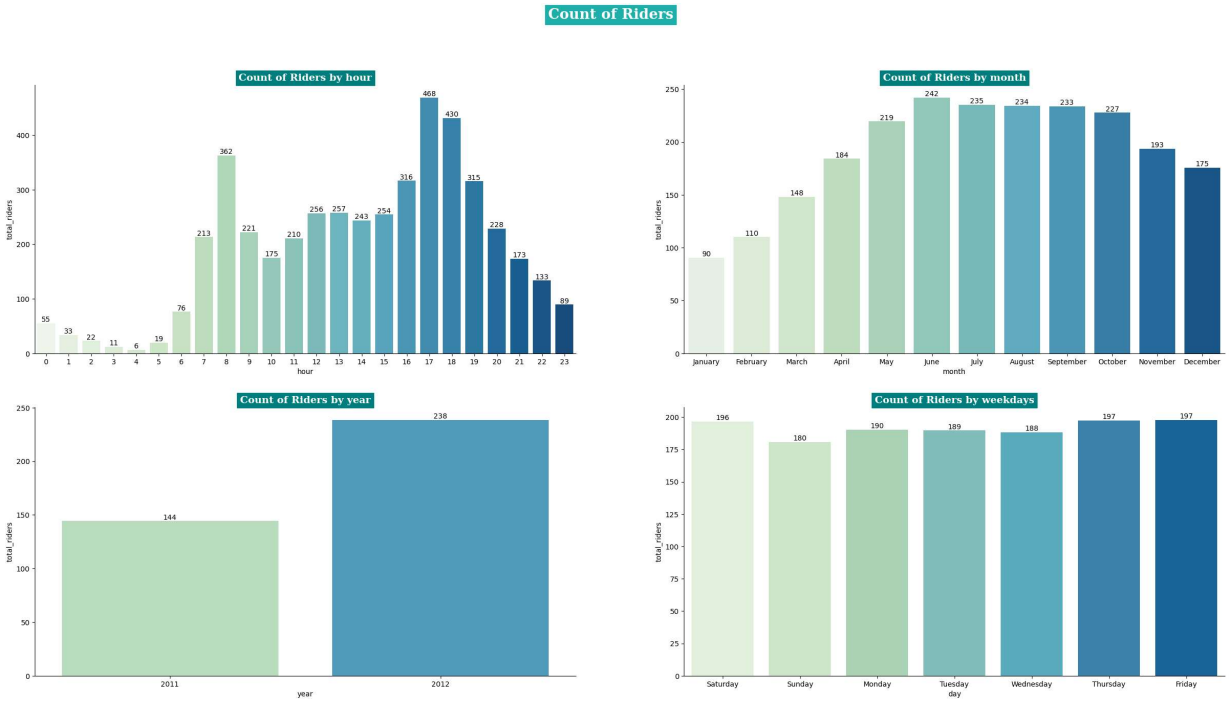
```
plt.figure(figsize=(25,6))
plt.style.use('default')
```

```
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='
        a.bar_label(a.containers[0], label_type='edge')
        sns.despine(left=True,bottom=True)
        plt.yticks([])
        plt.ylabel('')
        i+=1
```

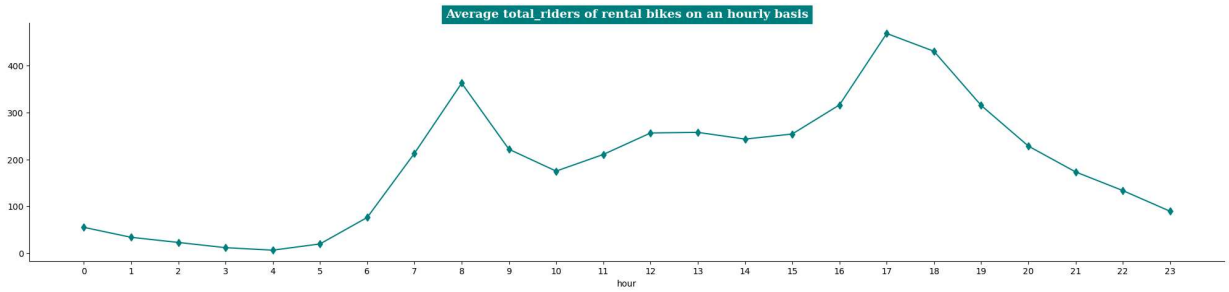



```
plt.figure(figsize=(30,15))
plt.suptitle('Count of Riders',fontsize=20,fontfamily='serif',fontweight='bold'

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='

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=

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')
```

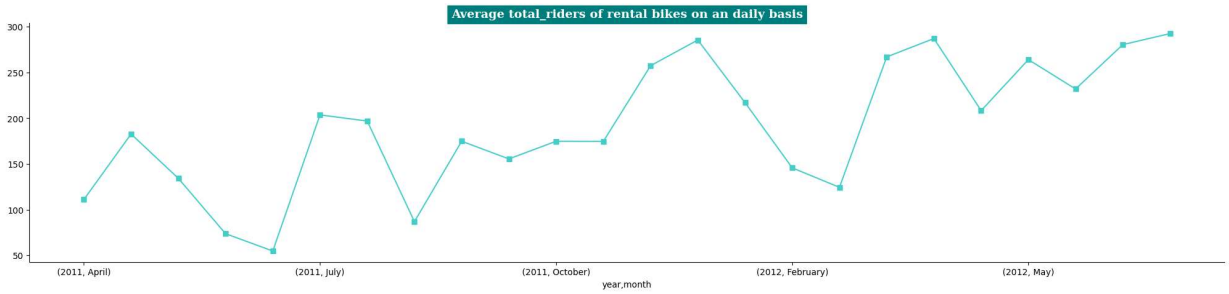
```
plt.title('Count of Riders by year',fontsize=14,fontfamily='serif',fontweight='

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',fontweig
sns.despine()

plt.show()
```



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




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



🏷️Insights:

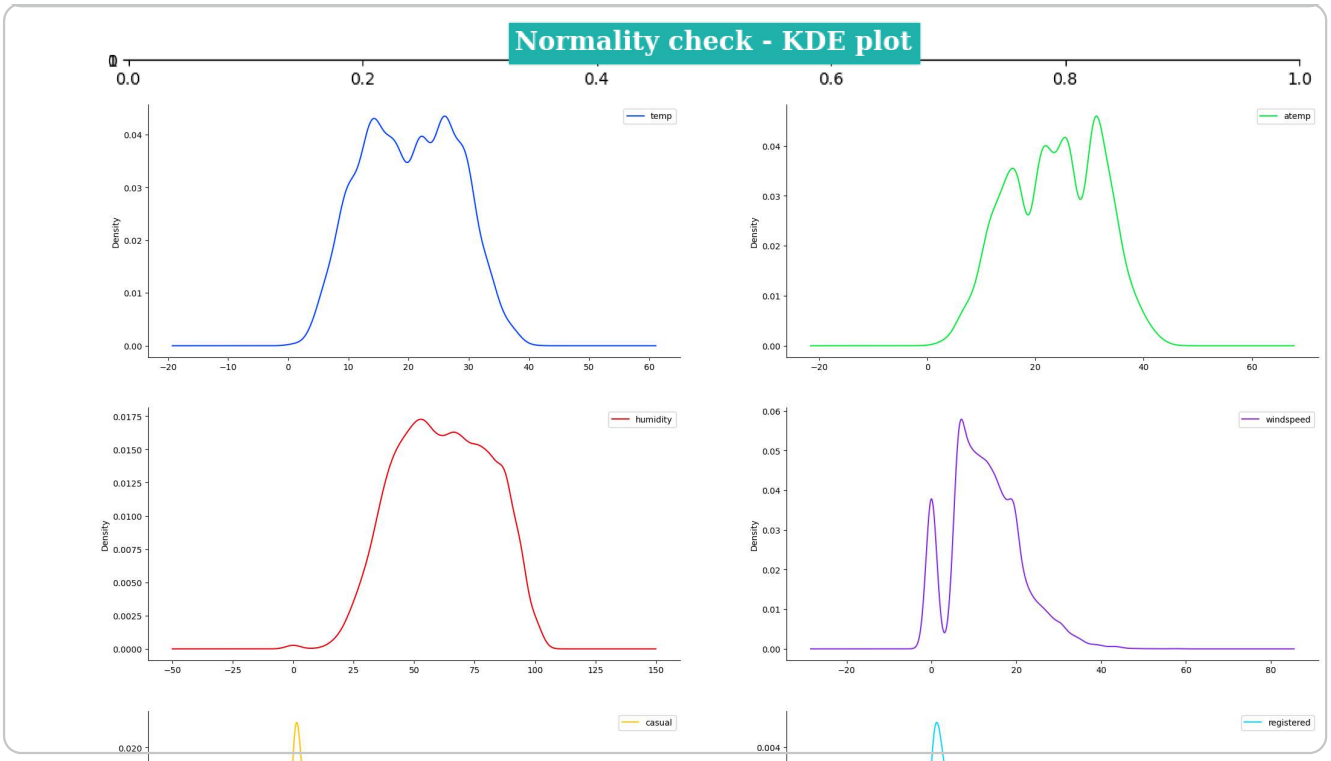
- These patterns indicate that there is a distinct fluctuation in total_riders throughout the day, with low counts during early morning hours, a sudden increase in the morning, a peak count in the afternoon, and a gradual decline in the evening and nighttime.
- These patterns indicate that there is a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

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

	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

```
plt.figure(figsize=(13,0))
plt.title('Normality check - KDE plot',fontsize=16,fontfamily='serif',fontweigh
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

```
yd.columns
```

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

```
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_valu

    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 populatic

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

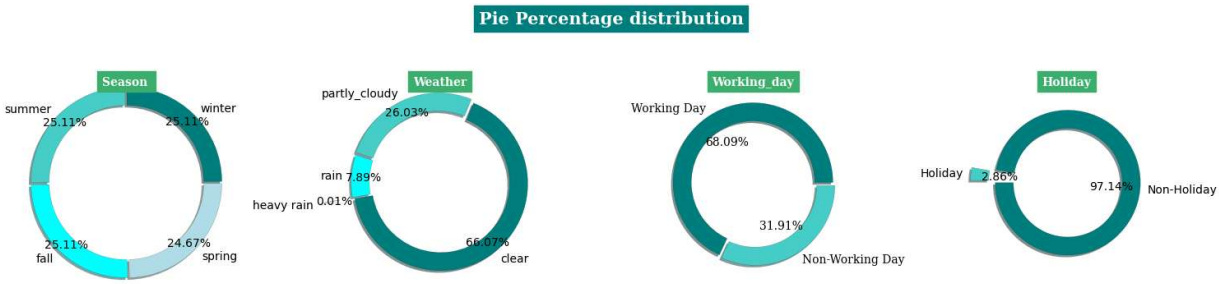
```
Normality Testing for temp :
------------------------------
The test-statistic for temp is 0.9804227352142334 with p_value 4.5771170017549
Hence at 95% confidence level, we reject null hypothesis
Hence we can say that temp is not normally distributed population

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

Normality Testing for atemp :
------------------------------
The test-statistic for atemp is 0.9815532565116882 with p_value 3.355995045624
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 1.244270413
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

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

```
cp = ['teal' , 'mediumturquoise' , 'aqua' , 'powderblue' , 'cyan','teal' , 'mec
```

```
plt.figure(figsize=(15,4))
plt.suptitle('Pie Percentage distribution',fontsize=15,fontfamily='serif',fontw

plt.subplot(141)
plt.pie(yd['season'].value_counts(), labels=yd['season'].value_counts().index,
       counterclock=True , explode=(0.02,0.02,0.02,0.02) , autopct='%.2f%%', pc
       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',backgrounc

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%%',pctdis
       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('Weather',fontsize=10,fontfamily='serif',fontweight='bold',backgrounc

plt.subplot(143)
yd_workingday = np.round(yd['workingday'].value_counts(normalize = True) * 100,
plt.pie(x = yd_workingday, explode = [0, 0.05], labels = ['Working Day', 'Non-W
        autopct = '%.2f%%', textprops = {'fontsize' : 10,'fontfamily' : 'serif'
       wedgeprops=dict(edgecolor='k',linewidth=0.1,width=0.25),radius=1.1, shac
plt.title('Working_day',fontsize=10,fontfamily='serif',fontweight='bold',backgr

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=
        wedgeprops=dict(edgecolor='k',linewidth=0.1,antialiased=True,width=0.25)
plt.title('Holiday',fontsize=10,fontfamily='serif',fontweight='bold',backgrounc
plt.tight_layout()
plt.show()
```




```
yd.columns
```

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

```
yd['day'].unique()
```

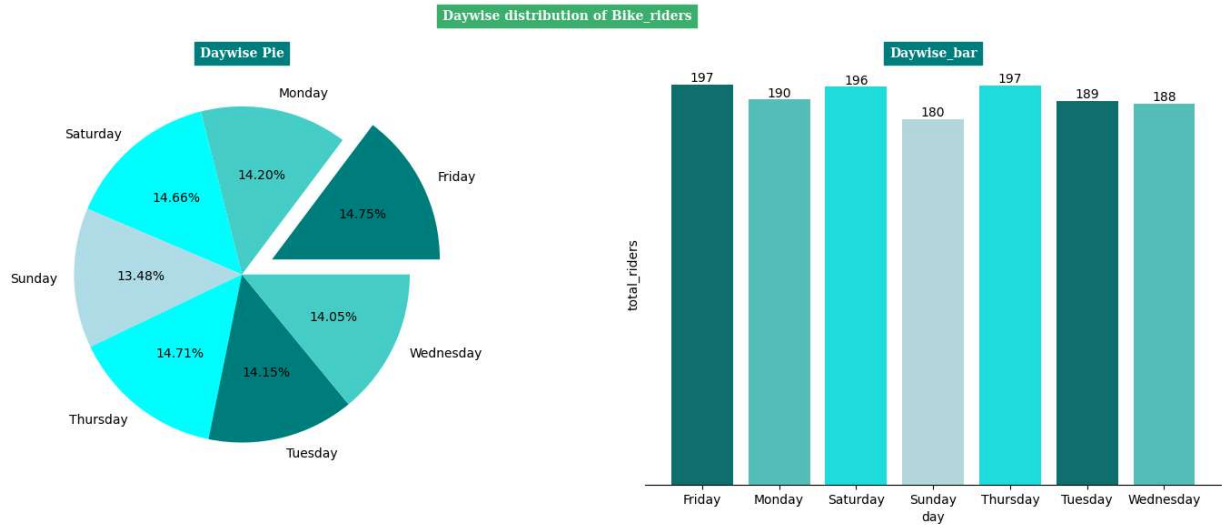
```
array(['Saturday', 'Sunday', 'Monday', 'Tuesday', 'Wednesday', 'Thursday',
       'Friday'], dtype=object)
```

```
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='seri
             backgroundcolor='mediumseagreen',color='w')

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

plt.subplot(122)
a=sns.barplot(data=day_cnt_riders,y=day_cnt_riders['total_riders'],x=day_cnt_ri
```

```
a.bar_label(a.containers[0], label_type='edge',fmt='%d')
plt.title('Daywise_bar',fontsize=10,fontfamily='serif',fontweight='bold',backgr
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

[ ]




```
monthwise_riders = yd.groupby('month')['total_riders'].mean().sort_values(ascer
display(monthwise_riders)

plt.figure(figsize=(18,8))
plt.suptitle('Monthwise distribution of Bike_riders',fontsize=10,fontfamily='se
```

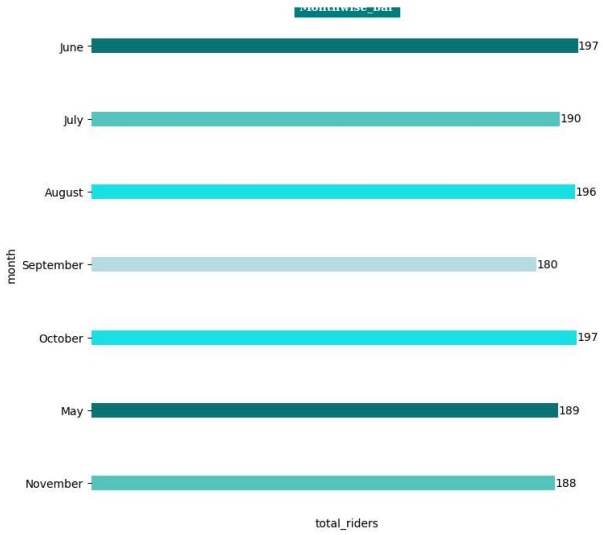
```
                backgroundcolor='mediumseagreen',color='w')

    plt.subplot(122)
    plt.pie(data=monthwise_riders, x=monthwise_riders['total_riders'], colors=cp,la
            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',back

    plt.subplot(121)
    a=sns.barplot(data=monthwise_riders,x=day_cnt_riders['total_riders'],y=monthwis
    a.bar_label(a.containers[0], label_type='edge',fmt='%d')
    plt.title('Monthwise_bar',fontsize=10,fontfamily='serif',fontweight='bold',back
    plt.ylim()
    plt.xticks([])

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

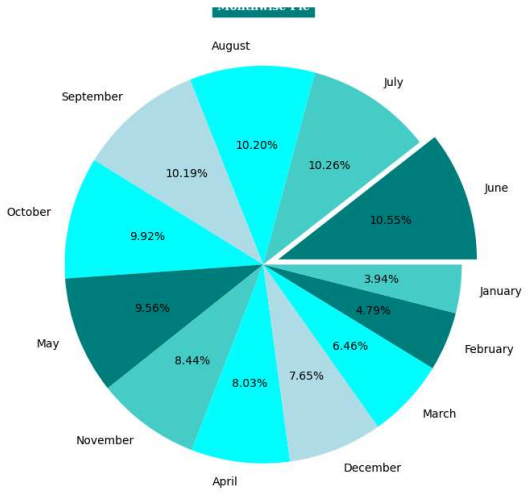
	month	total_riders
0	June	242.031798
1	July	235.325658
5	May	219.459430
6	November	193.677278

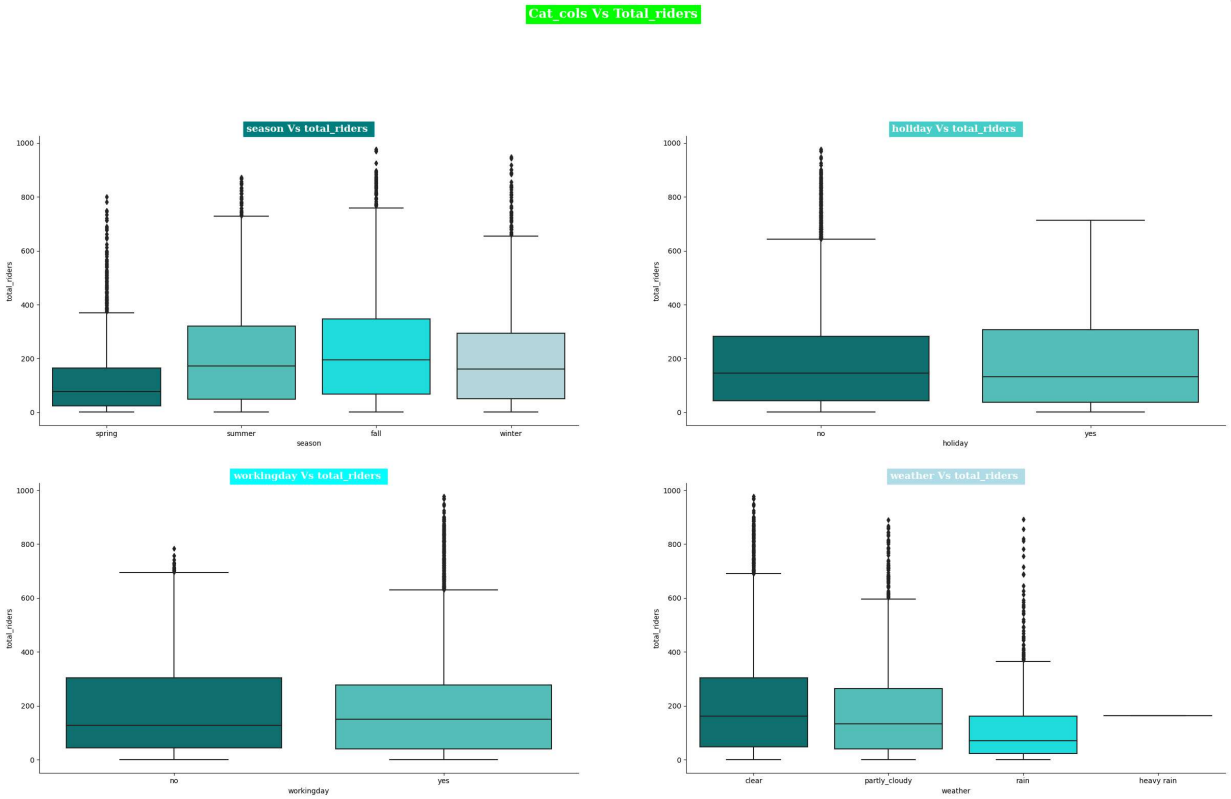
```
cat_cols = []
for _ in yd.columns:
    if yd[_].dtype=='category':
        cat_cols.append(_)
cat_cols
```

```
['season', 'holiday', 'workingday', 'weather']
```

```
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',
```



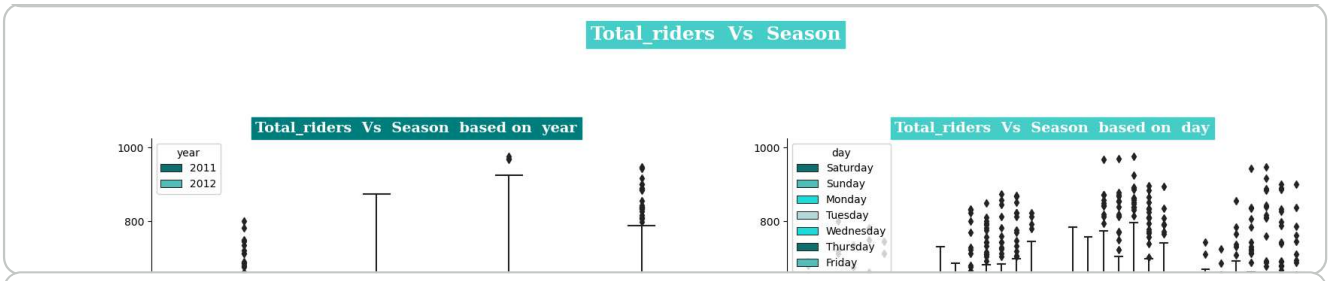



Cat_cols Vs Total_riders
season Vs total_riders
total_riders
season
spring
summer
fall
winter
holiday Vs total_riders
holiday
no
yes
workingday Vs total_riders
workingday
no
yes
weather Vs total_riders
weather
clear
partly_cloudy
rain
heavy rain

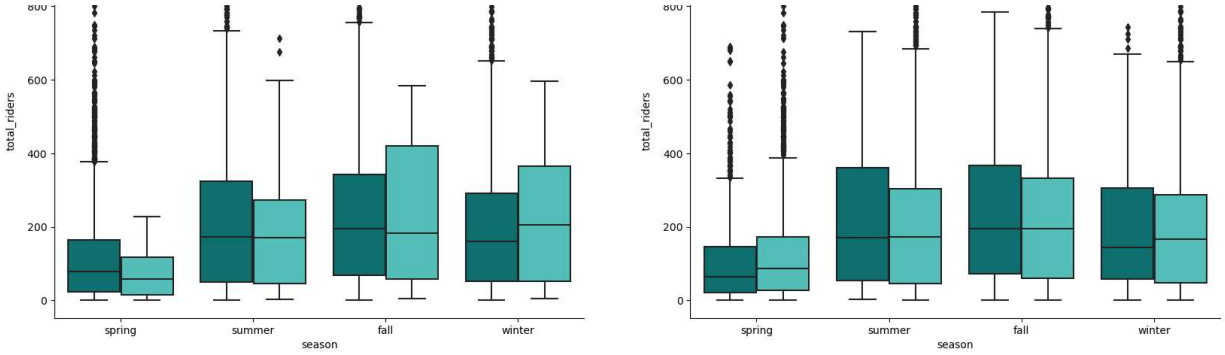
```
yd.sample()
```

	datetime	season	holiday	workingday	weather	temp	atemp	humidity
	2012-01-							

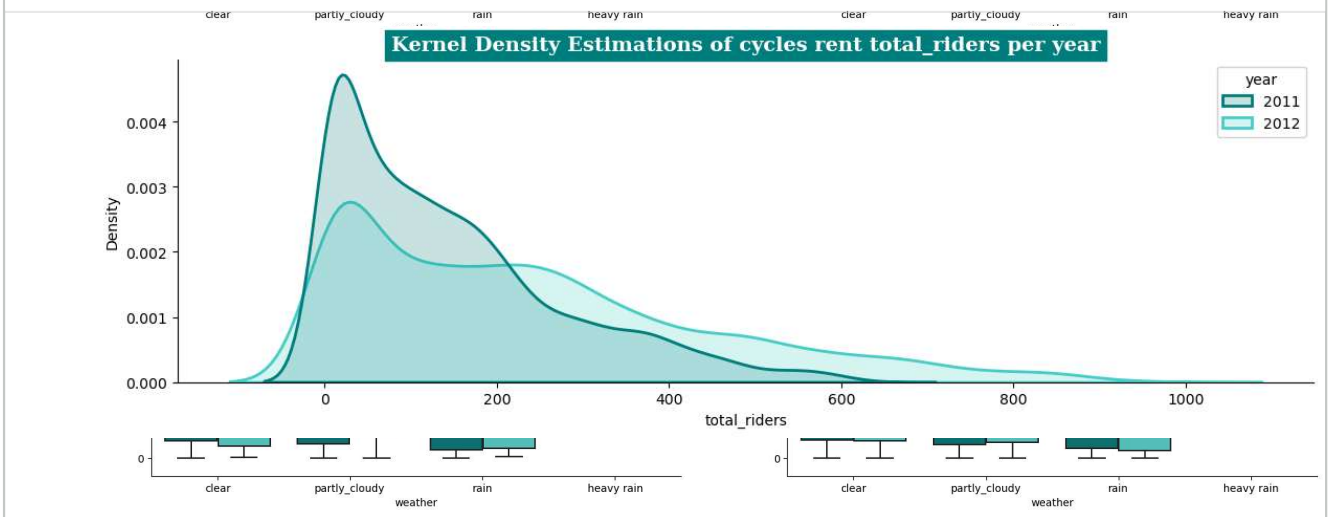
```
cols = ['year','day','holiday','workingday']
plt.figure(figsize = (20,15))
plt.style.use('seaborn-v0_8-bright')
plt.suptitle(f'Total_riders  Vs  Season', fontsize = 18,fontfamily='serif',font
for i in range(len(cat_cols)):
    plt.subplot(2,2,i+1)
    sns.boxplot(data = yd, x = 'season', y= 'total_riders',hue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




```
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',for
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], pale
    sns.despine()
    plt.title(f'Total_riders  Vs  Weather  based on  {cols[i]}', fontsize = 14,
              ,backgroundcolor=cp[i],color='w')
```




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




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

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

🔳🔍Outlier detection:

IQR:

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

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

```
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_
    else:
        print(f'unique Outlier points towards left of boxplot : {len(set(lower_
    if len(set(upper_outliers)):
        print(f'unique Outlier points towards right of boxplot : {len(set(upper
    else:
        print(f'unique Outlier points towards right of boxplot : {len(set(upper
    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]}',fontfam
              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=
              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
                      medianprops={"color": "k", "linewidth": 6})#,boxprops={"f
    plt.title(f'Box & Whisker plot of {numeric_cols[i]}',fontfamily='serif',for
              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='
              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,
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, 35.00
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, 35.00
```

	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,
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
```

	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

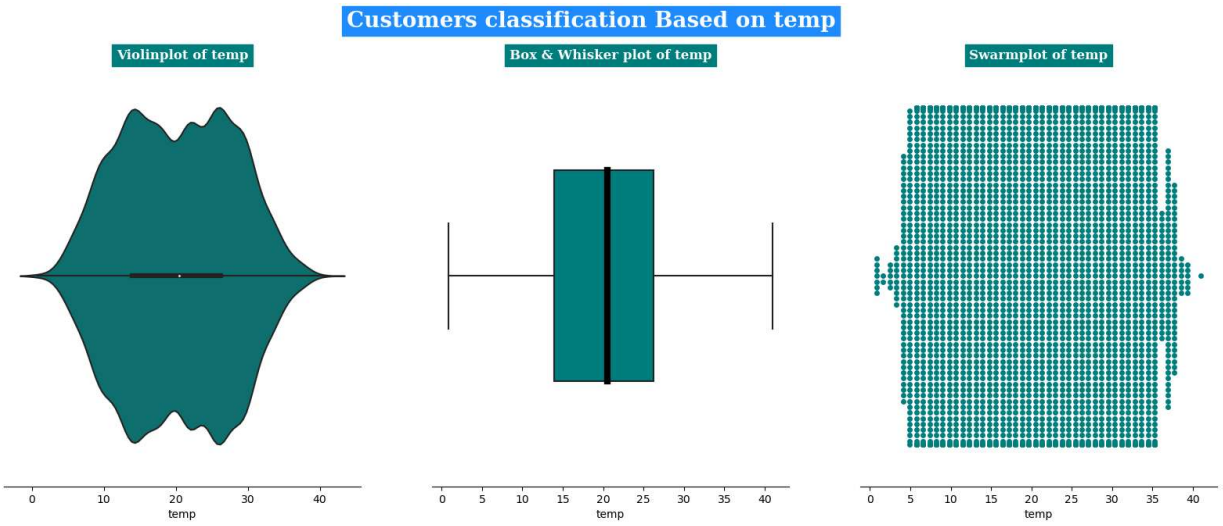
```
--------------------------------------------------------------------------------

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,
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
```

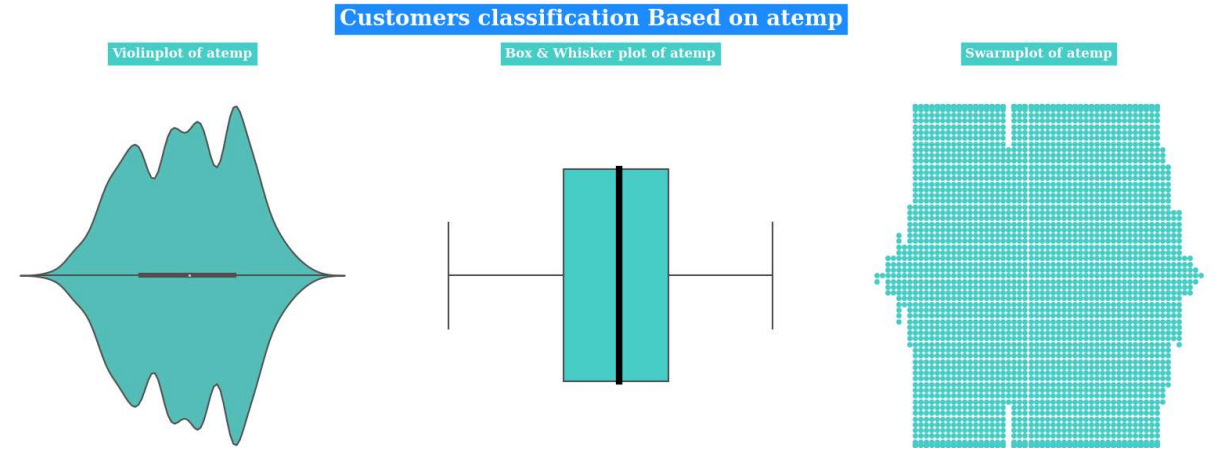
	Statistic	Value
0	count	10886.000000
1	mean	155.552177
2	std	151.039033
3	min	0.000000
4	25%	36.000000
5	50%	118.000000
6	75%	222.000000
7	max	886.000000

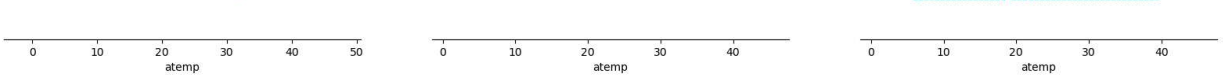
```
--------------------------------------------------------------------------------

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,
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
```

	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

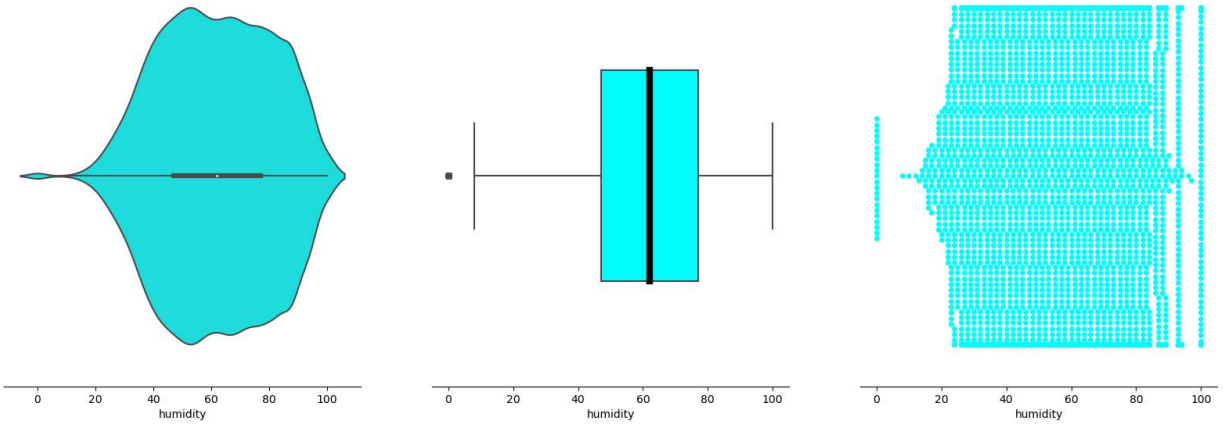






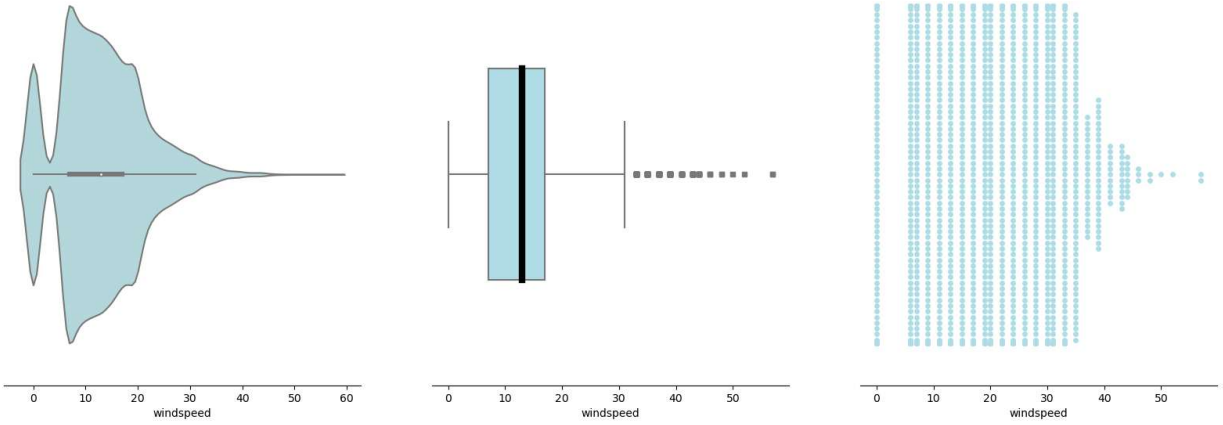
Customers classification Based on humidity

Violinplot of humidity | Box & Whisker plot of humidity | Swarmplot of humidity



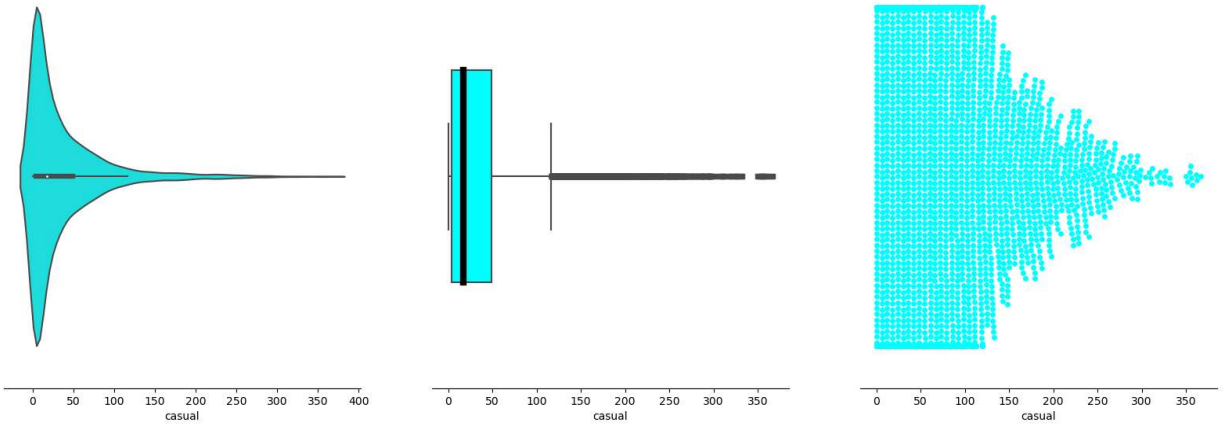
Customers classification Based on windspeed

Violinplot of windspeed | Box & Whisker plot of windspeed | Swarmplot of windspeed



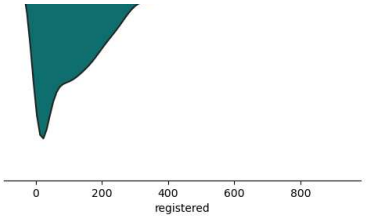
Customers classification Based on casual

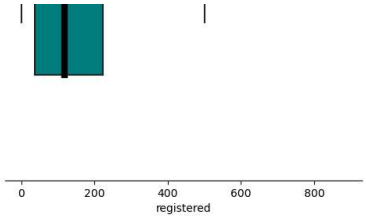
Violinplot of casual | Box & Whisker plot of casual | Swarmplot of casual

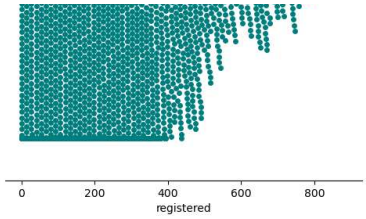


Customers classification Based on registered

Violinplot of registered | Box & Whisker plot of registered | Swarmplot of registered

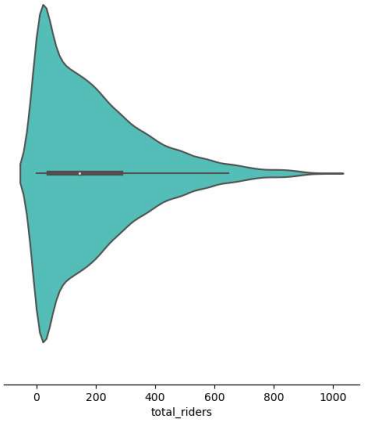




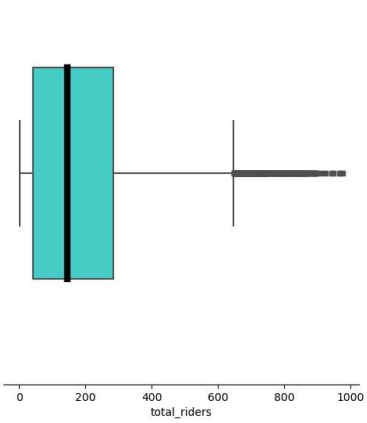


Customers classification Based on total_riders

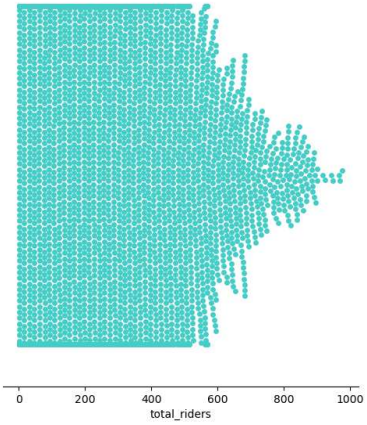
Violinplot of total_riders



Box & Whisker plot of total_riders



Swarmplot of total_riders



Method-2

```
df_num_cols
```

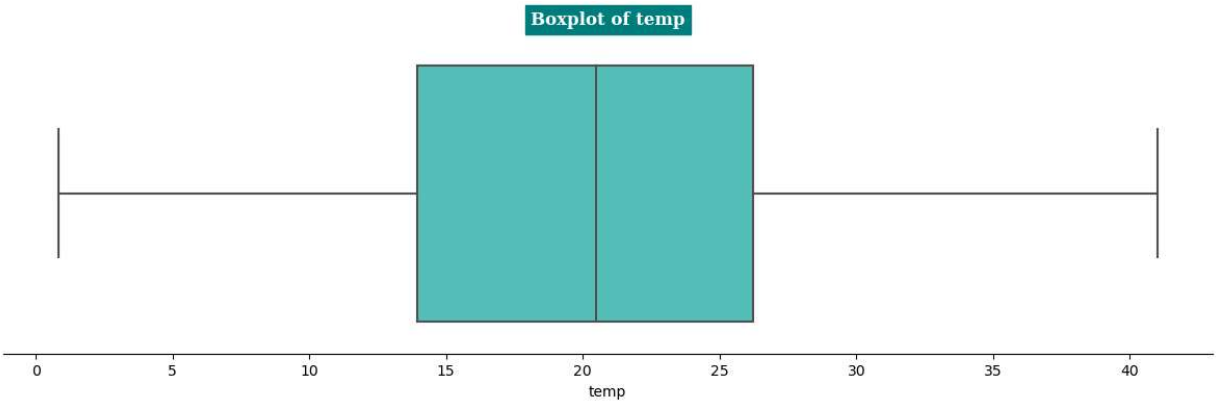
	temp	atemp	humidity	windspeed	casual	registered	total_riders	year
0	9.84	14.395	81	0.0000	3	13	16	2011
1	9.02	13.635	80	0.0000	8	32	40	2011
2	9.02	13.635	80	0.0000	5	27	32	2011
3	9.84	14.395	75	0.0000	3	10	13	2011
4	9.84	14.395	75	0.0000	0	1	1	2011
...	...	...	...	...	...	...	...	...
10881	15.58	19.695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

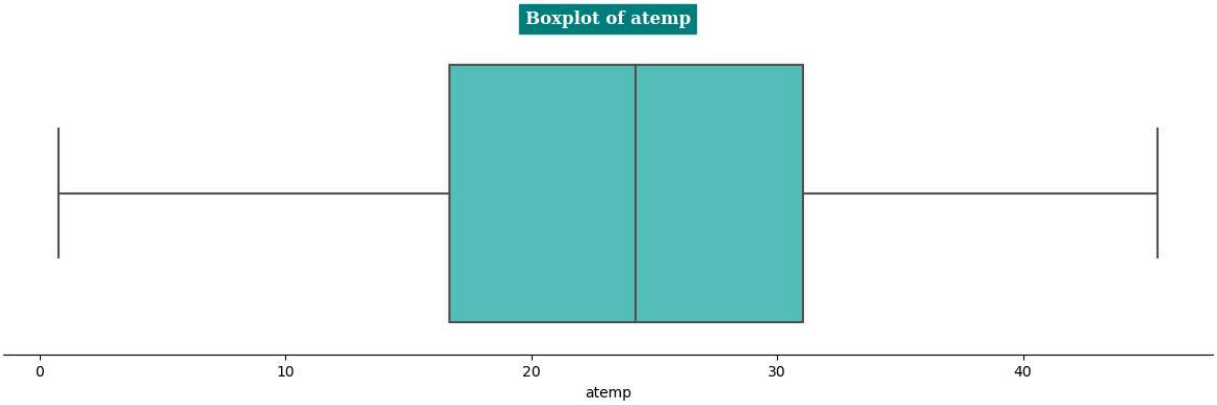
10886 rows × 9 columns

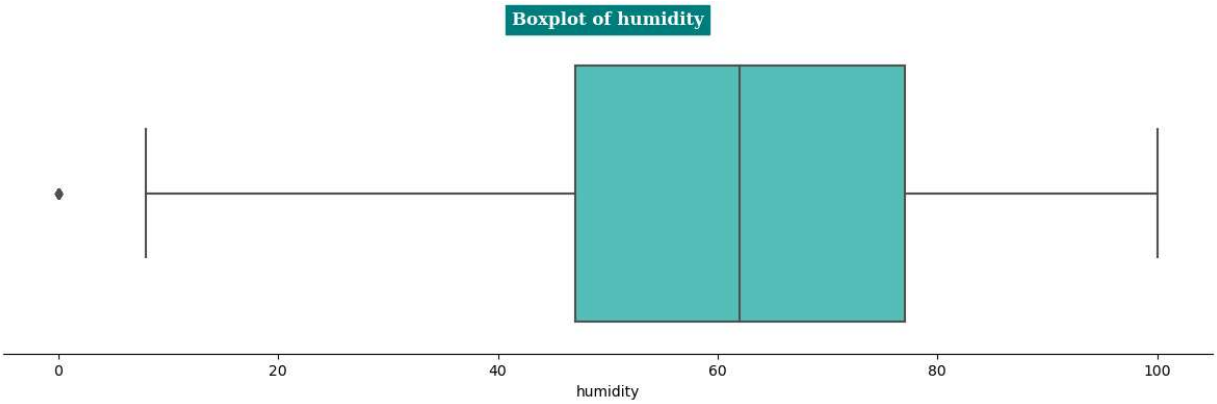
```
# obtain the first quartile
Q1 = df_num_cols.quantile(0.25)

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

# obtain the IQR
IQR = Q3 - Q1

# print the IQR
print(IQR)
```

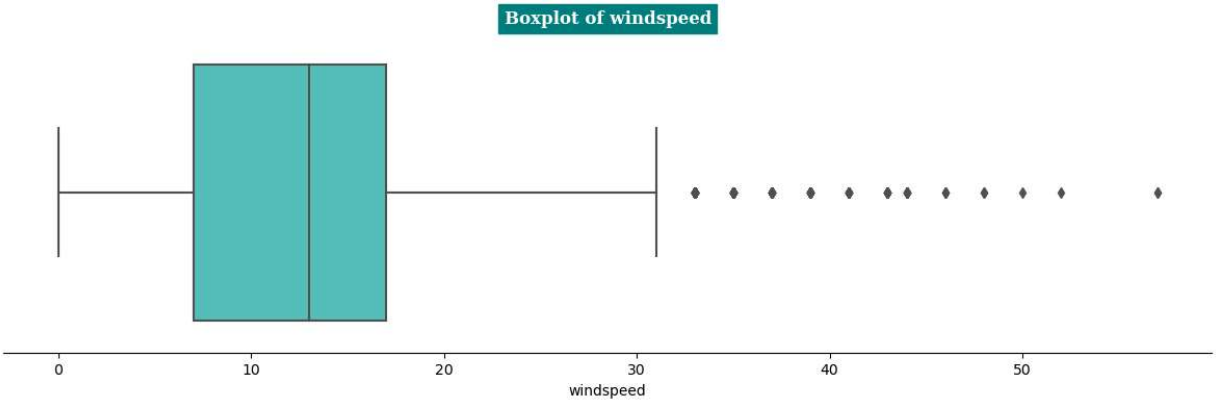
```
temp             12.3000
atemp            14.3950
humidity         30.0000
windspeed         9.9964
casual           45.0000
registered      186.0000
total_riders    242.0000
year              1.0000
hour             12.0000
dtype: float64
```

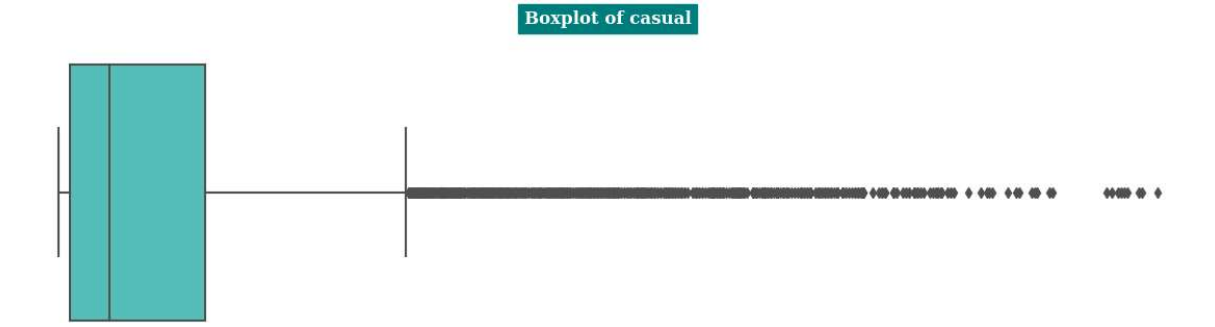
```
    for col in enumerate(df_num_cols):
        plt.figure(figsize=(15,4))
        sns.boxplot(x=col[1], data=df_num_cols,color=cp[1])
        sns.despine(left=True)
        plt.yticks([])
        plt.title(f'Boxplot of {col[1]}',fontfamily='serif',fontweight='bold',fonts
        plt.show()
```

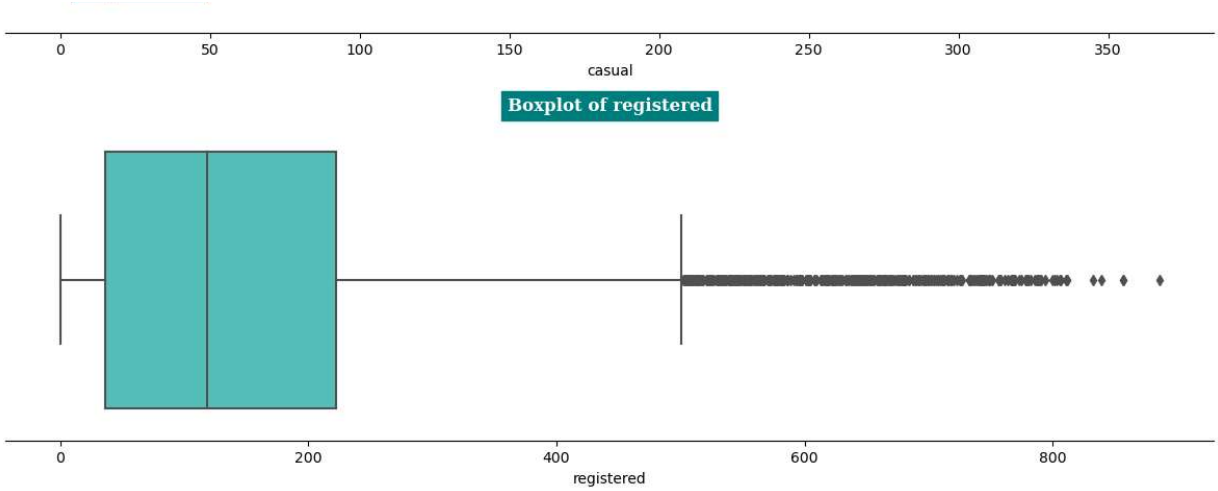

Boxplot of temp
0
5
10
15
20
25
30
35
40
temp


Boxplot of atemp
0
10
20
30
40
atemp

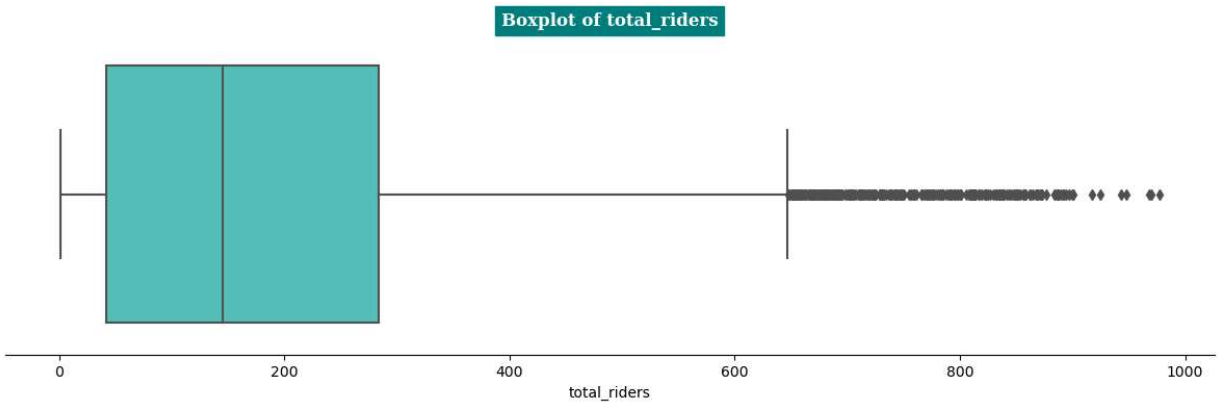

Boxplot of humidity
0
20
40
60
80
100
humidity

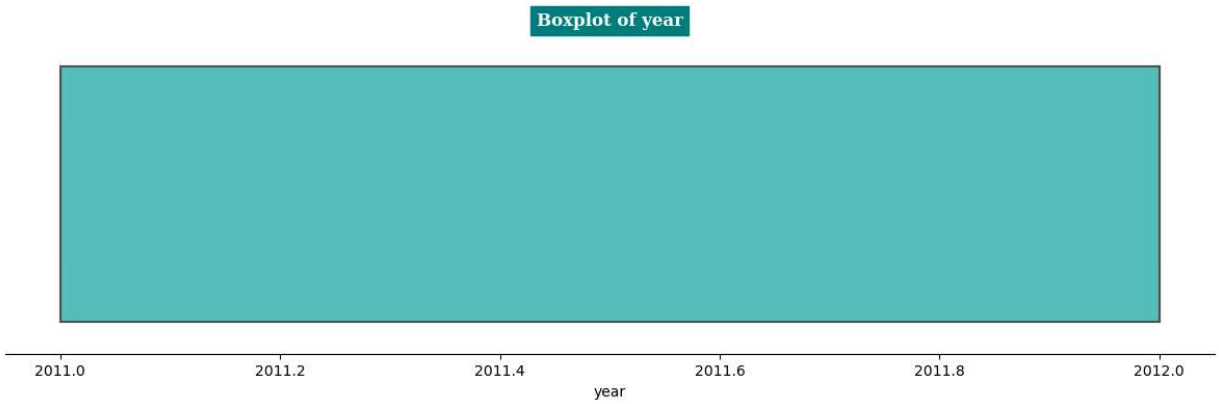

Boxplot of windspeed
0
10
20
30
40
50
windspeed

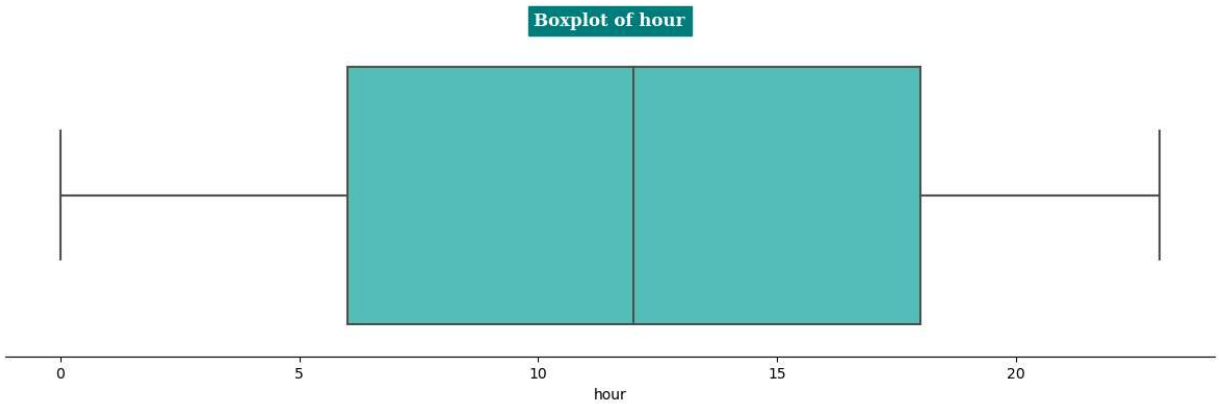

Boxplot of casual


0
50
100
150
200
250
300
350
casual

Boxplot of registered
0
200
400
600
800
registered


Boxplot of total_riders
0
200
400
600
800
1000
total_riders


Boxplot of year
2011.0
2011.2
2011.4
2011.6
2011.8
2012.0
year


Boxplot of hour
0
5
10
15
20
hour

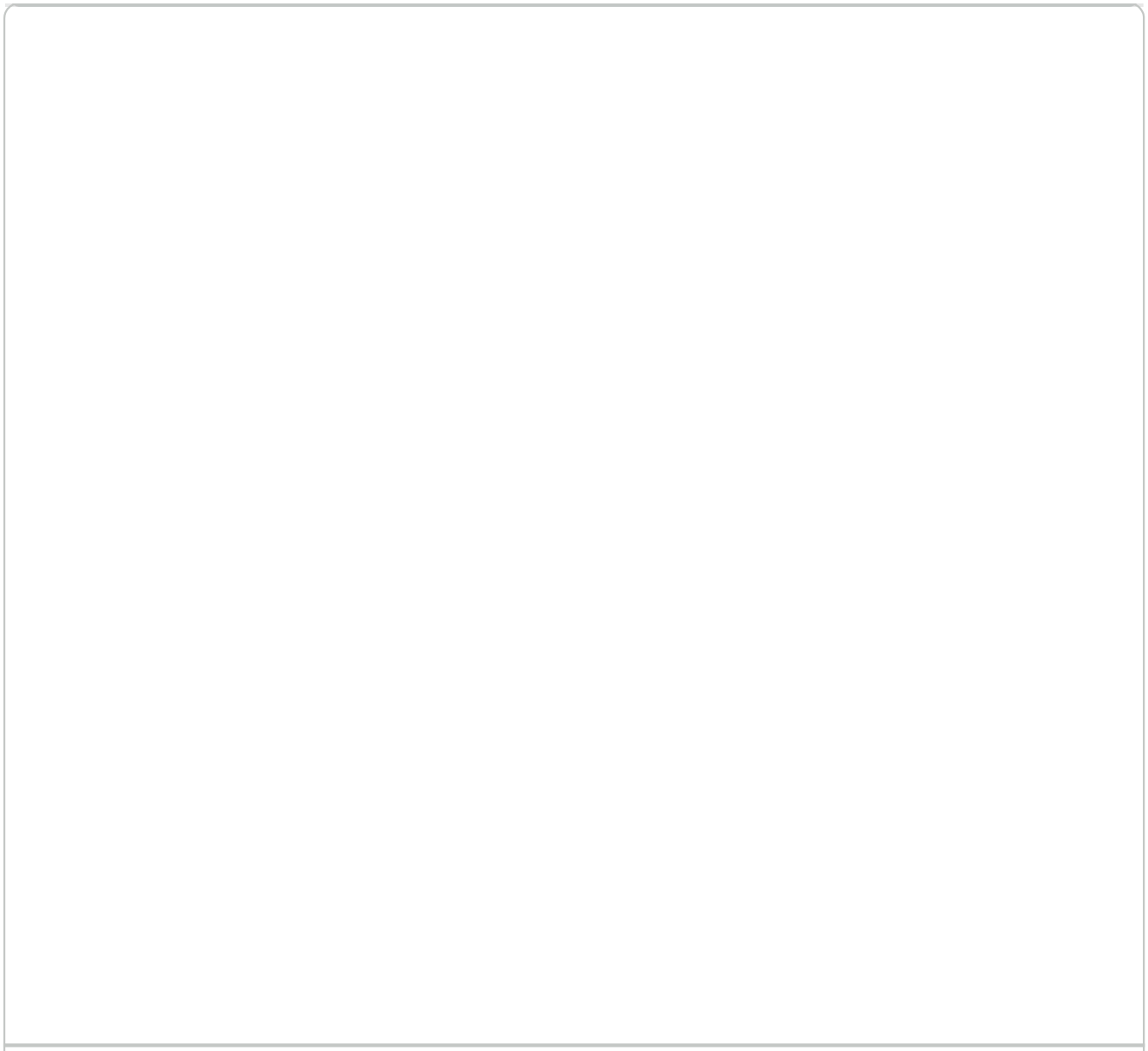
```
df_iqr=df_num_cols[~((df_num_cols< (Q1-1.5*IQR))|(df_num_cols > (Q3 + 1.5*IQR))
df_iqr
```

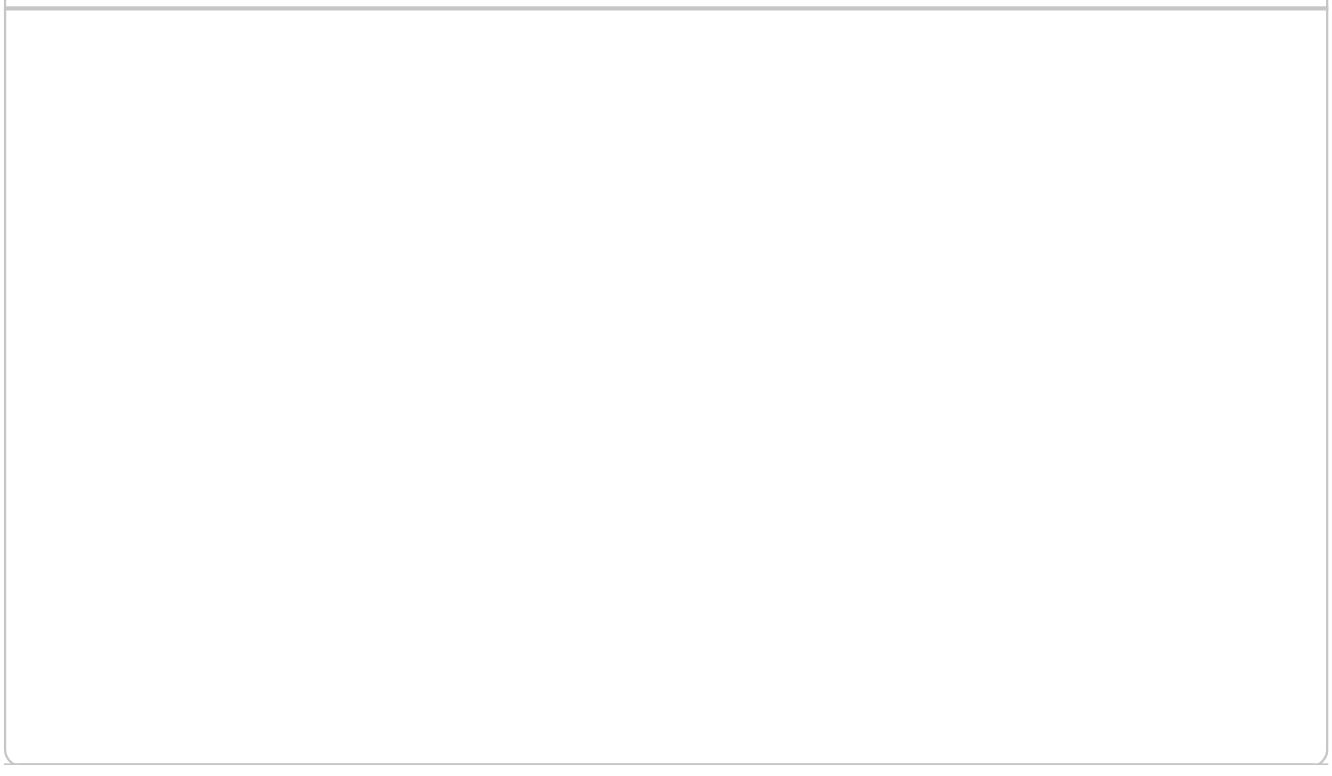
	temp	atemp	humidity	windspeed	casual	registered	total_riders	year
0	9.84	14.395	81	0.0000	3	13	16	2011
1	9.02	13.635	80	0.0000	8	32	40	2011
2	9.02	13.635	80	0.0000	5	27	32	2011
3	9.84	14.395	75	0.0000	3	10	13	2011
4	9.84	14.395	75	0.0000	0	1	1	2011
...	...	...	...	...	...	...	...	...
10881	15.58	19.695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

9518 rows × 9 columns

✨ Outlier Removal

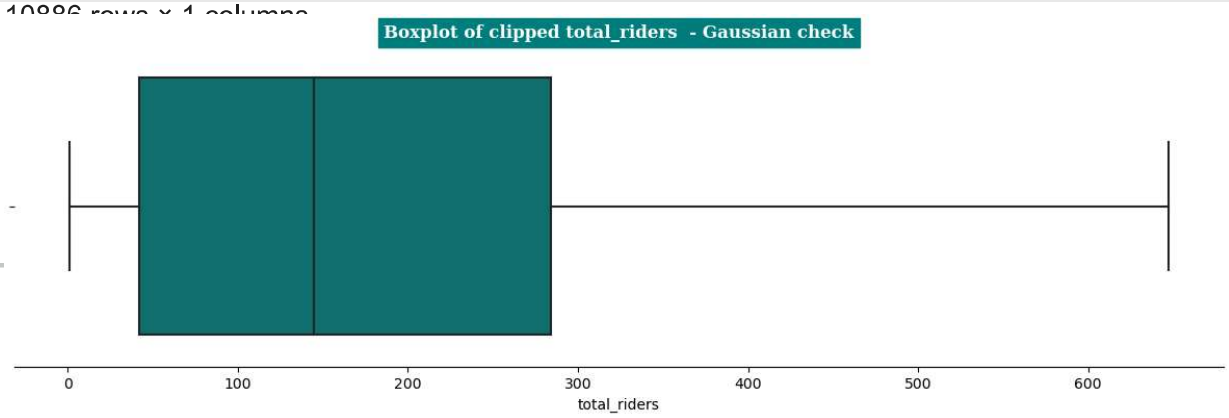
```
data = yd['total_riders']
display(data.to_frame())
Q1 = np.percentile(data, 25)
Q3 = np.percentile(data, 75)
IQR = Q3 - Q1

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

```
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='seri
          fontweight='bold',fontsize=12,backgroundcolor=cp[0],color='w')
plt.show()
```



...	...
10881	336
10882	241
10883	168
10884	129
10885	88

10586 rows × 1 columns

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

	total_riders
0	16

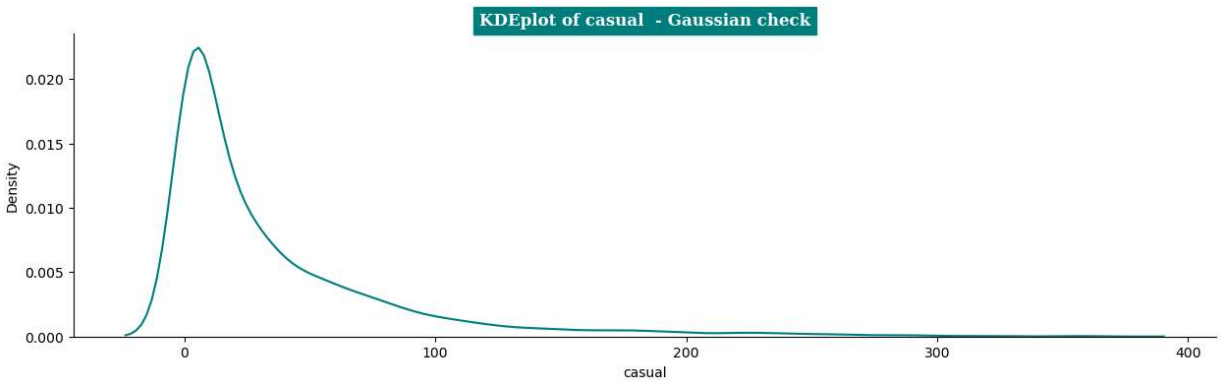
```
odz_cols = df_num_cols.iloc[:,4:7]
odz_cols
```

3	13

	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

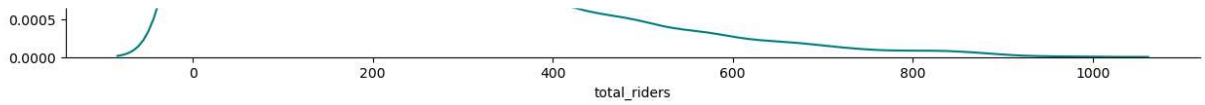
10886 rows × 3 columns

```
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',fontwei
    plt.show()
```

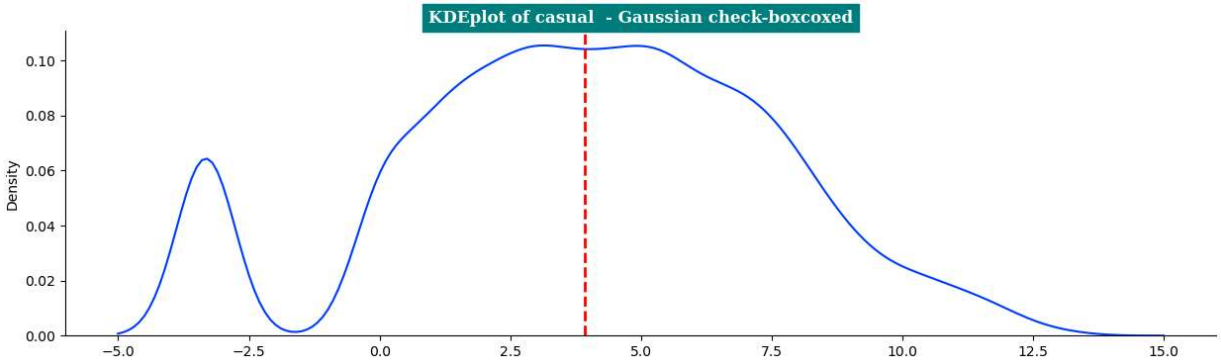


Since right skewed we will take boxcox transformation and then try to find out the outliers

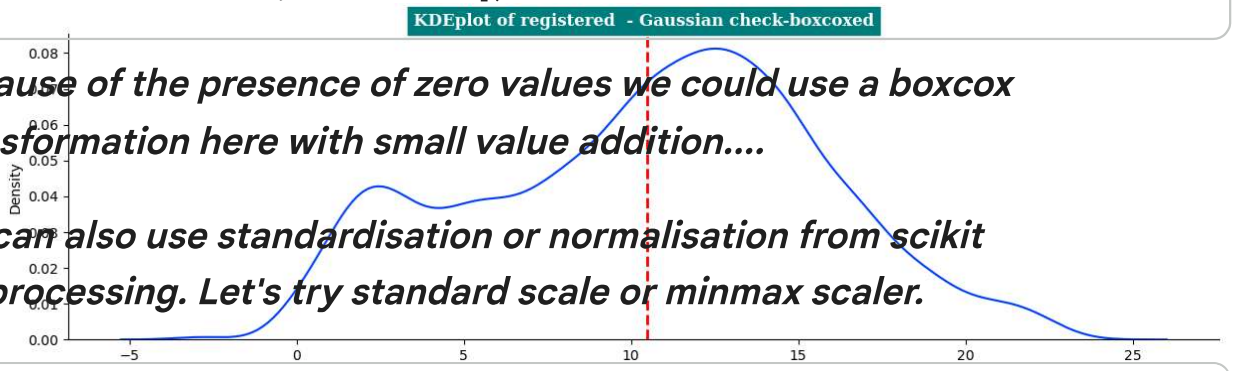
```
for k in odz_cols:
    transformed_data , best_lambda = boxcox(odz_cols[k]+0.001)
    print(f"Best lambda of {k} = {best_lambda}")
    display(transformed_data)
    plt.figure(figsize=(15,4))
    sns.kdeplot(transformed_data)
    sns.despine()
    plt.axvline(transformed_data.mean(), color='r', linestyle='--',linewidth=2)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



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.

```
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())
```

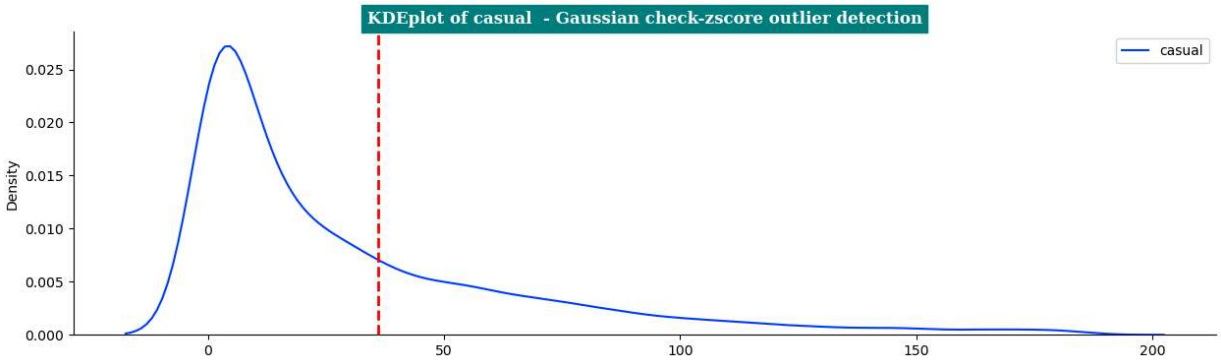
```
 The minimum value of casual: 0
 The maximum value of {k}: 367
 The minimum value of registered: 0
 The maximum value of {k}: 886
 The minimum value of total_riders: 1
 The maximum value of {k}: 977
```

```
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',font
              ,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

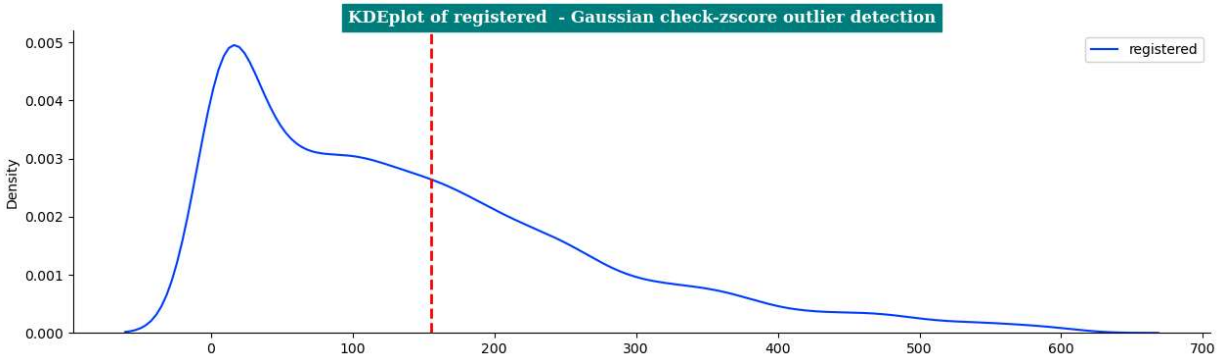



```
0       -0.943854
1       -0.818052
2       -0.851158
3       -0.963717
4       -1.023307
           ...
10881    1.148417
10882    0.499548
10883    0.055934
10884   -0.255258
```

```
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

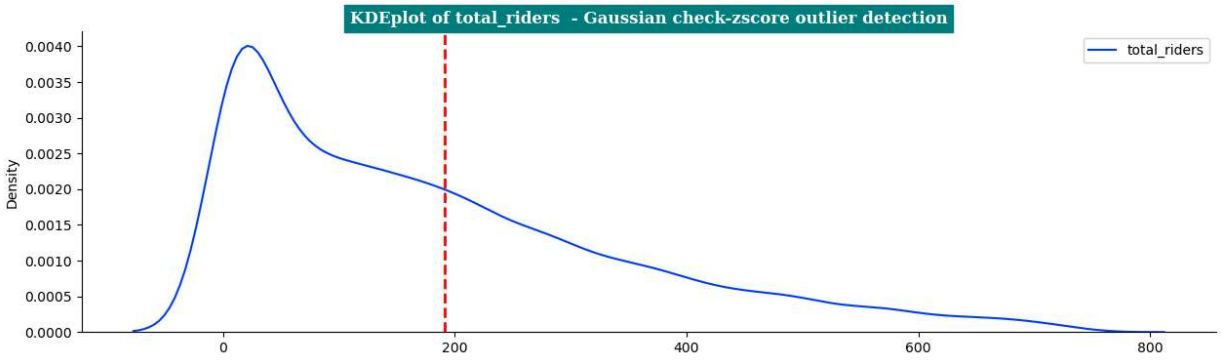



```
0       -0.969294
1       -0.836797
2       -0.880962
3       -0.985856
4       -1.052104
           ...
10881    0.797333
10882    0.272866
10883   -0.130146
10884   -0.345454
10885   -0.571803
Name: total_riders, Length: 10886, dtype: float64
```

	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



```
new_df = odz_cols.copy()
```

```
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
```

```
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())
```

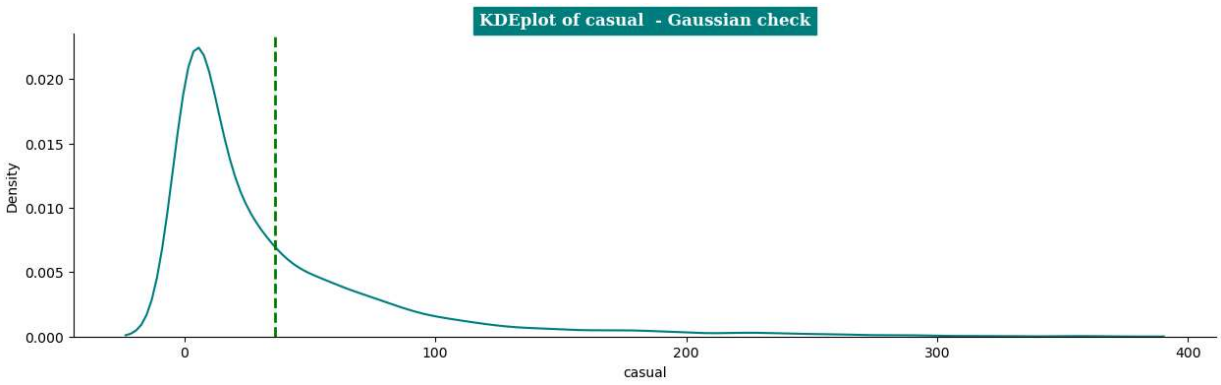
```
 The minimum value of {k}: 0.0
 The maximum value of {k}: 1.0
 The minimum value of {k}: 0.0
 The maximum value of {k}: 0.9999999999999999
 The minimum value of {k}: 0.0
 The maximum value of {k}: 1.0
```

```
new_df
```

	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

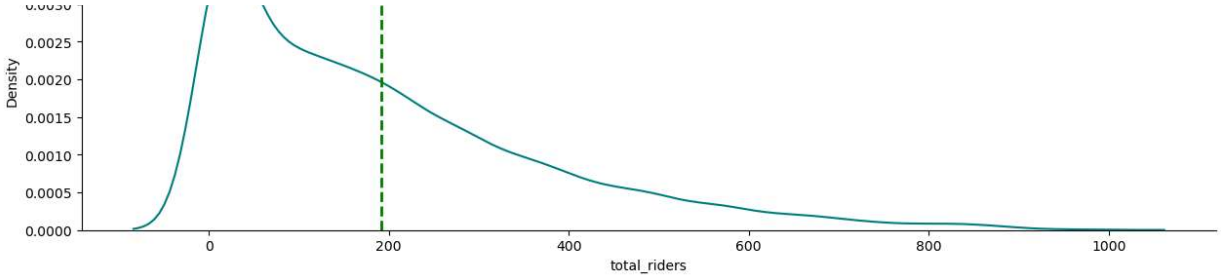
10886 rows × 3 columns

```
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',fontwei
    plt.show()
```



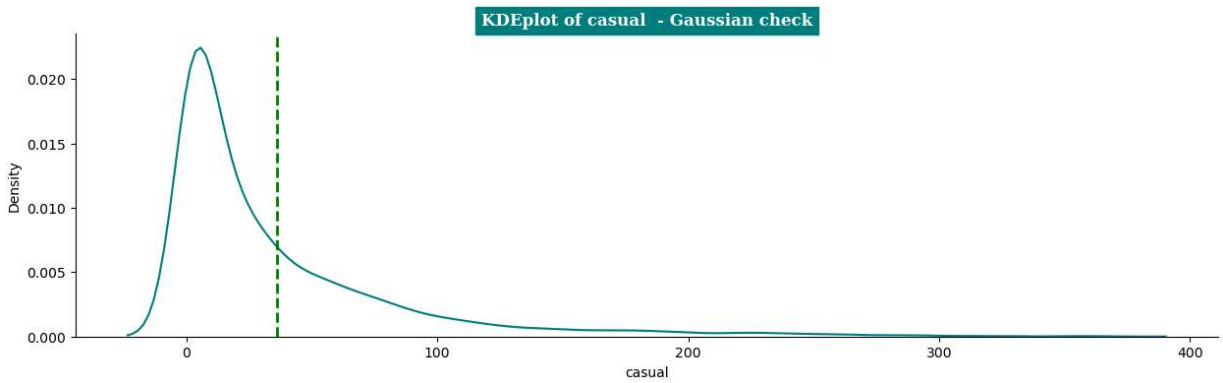
KDEplot of registered - Gaussian check

```
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
    plt.show()
```



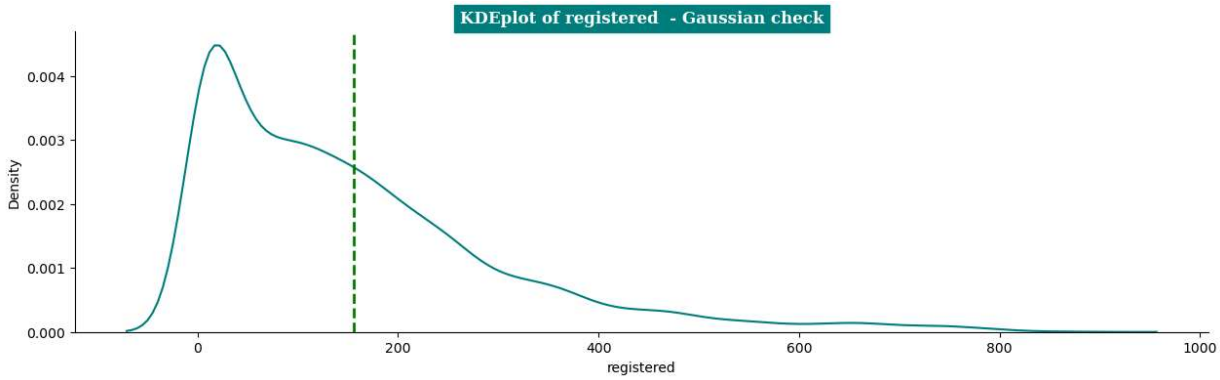
	casual
0	3
1	8
2	5
3	3
4	0
...	...
10881	7
10882	10
10883	4
10884	12
10885	4

10594 rows × 1 columns



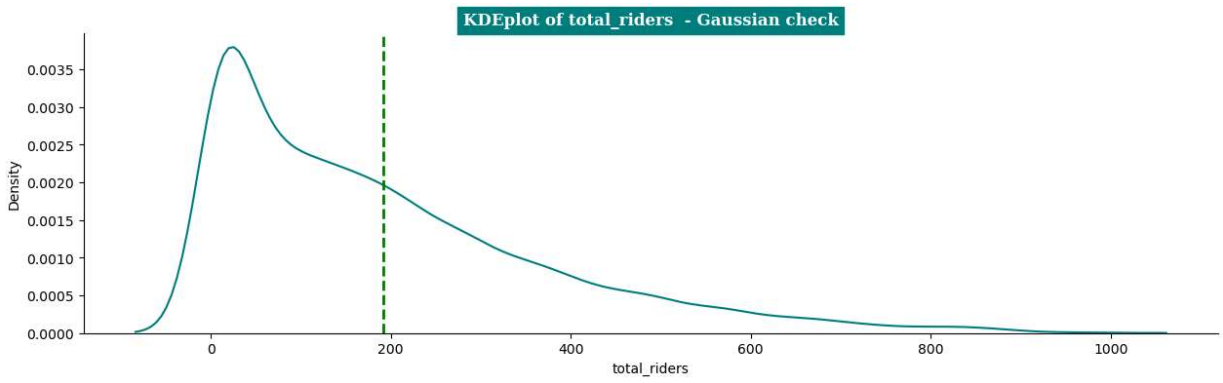
	registered
0	13
1	32
2	27
3	10
4	1
...	...
10881	329
10882	231
10883	164
10884	117
10885	84

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:

`num_cols`

	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.

```
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 t
    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 ar
            print('There is Strong Positive correlation between total_riders ar
        else:
            print('There is Weak Positive correlation between total_riders and'
            print('There is weak Positive correlation between total_riders and'
    if sc == 0:
        print('There is No correlation between total_riders and', col,"- pearsc
        print('There is No correlation between total_riders and', col,"- spearm
    if sc< 0:
        if sc<0.5:
            print('There is Strong Negative correlation between total_riders ar
            print('There is Strong Negative correlation between total_riders ar
        else:
            print('There is Weak Positive correlation between total_riders and'
            print('There is weak Positive correlation between total_riders and'
    print()
    print('-'*100)
    print()
```

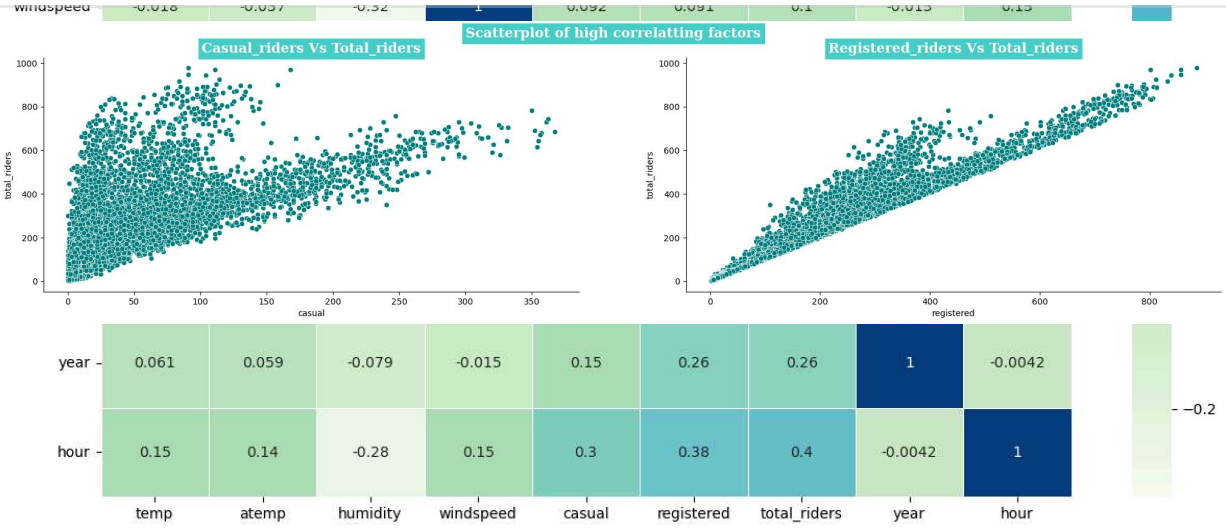
```
Testing Correlation between temp and Total_riders
There is Weak Positive correlation between total_riders and temp - pearson-corrc
There is weak Positive correlation between total_riders and temp - spearman-corr

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

Testing Correlation between atemp and Total_riders
```

```
There is Weak Positive correlation between total_riders and atemp - pearson-corr
There is weak Positive correlation between total_riders and atemp - spearman-cor

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

Testing Correlation between humidity and Total_riders
There is Strong Negative correlation between total_riders and humidity - pearson
There is Strong Negative correlation between total_riders and humidity - spearma

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

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

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

Testing Correlation between casual and Total_riders
There is Strong Positive correlation between total_riders and casual - pearson-c
There is Strong Positive correlation between total_riders and casual - spearman-

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

Testing Correlation between registered and Total_riders
There is Strong Positive correlation between total_riders and registered - pears
There is Strong Positive correlation between total_riders and registered - spear

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

Testing Correlation between total_riders and Total_riders
There is Strong Positive correlation between total_riders and total_riders - pea
There is Strong Positive correlation between total_riders and total_riders - spe

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

```
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=
plt.show()
```

	temp	atemp	humidity	windspeed	casual	registered	tota
temp	1.000000	0.984948	-0.064949	-0.017852	0.467097	0.318571	
atemp	0.984948	1.000000	-0.043536	-0.057473	0.462067	0.314635	
humidity	-0.064949	-0.043536	1.000000	-0.318607	-0.348187	-0.265458	-
windspeed	-0.017852	-0.057473	-0.318607	1.000000	0.092276	0.091052	
casual	0.467097	0.462067	-0.348187	0.092276	1.000000	0.497250	

```
plt.figure(figsize=(25,5))
plt.suptitle('Scatterplot of high correlatting factors',fontfamily='serif',font
             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',

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='bc

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



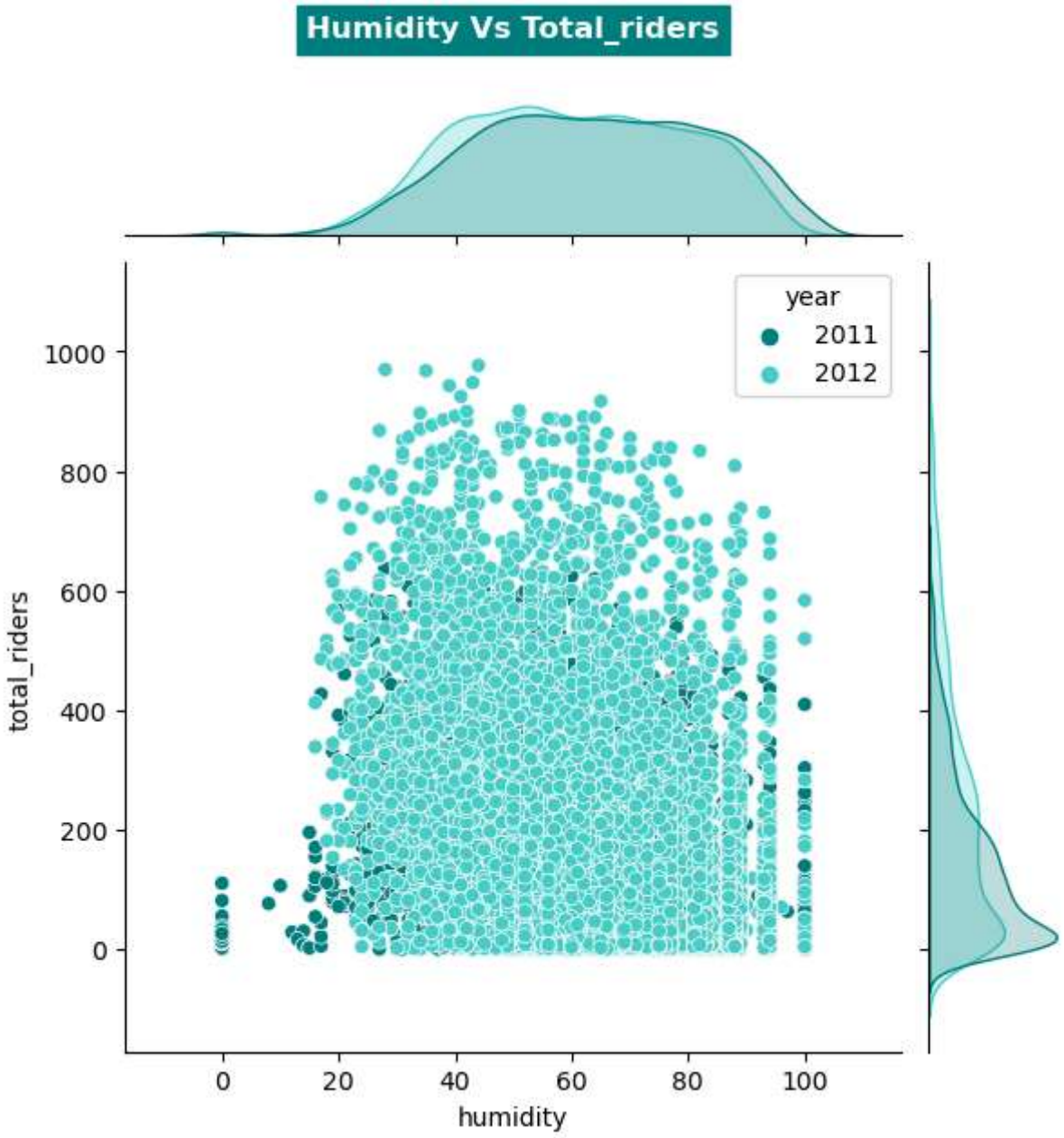
🏷️Insights

- 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.

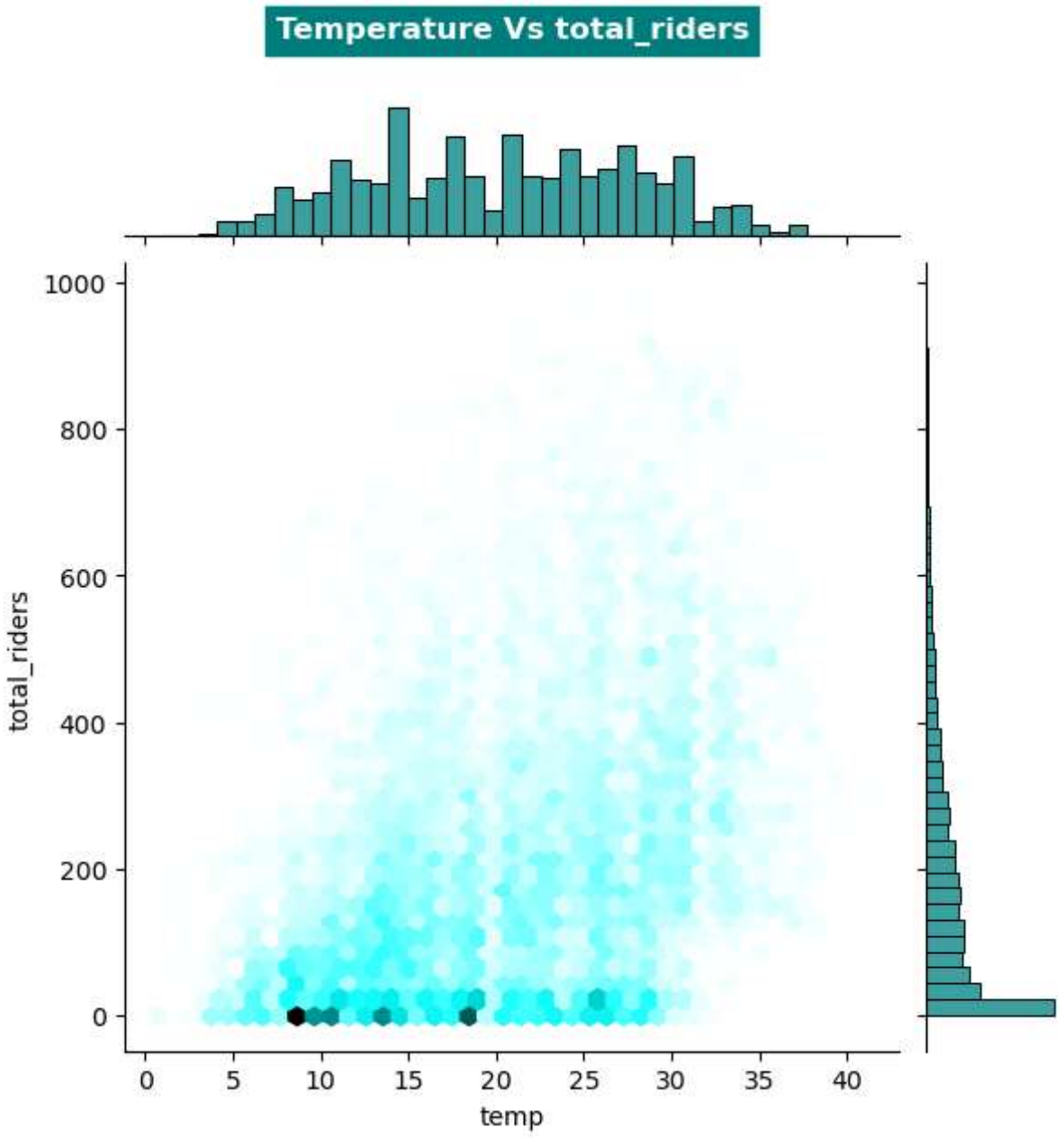
```
plt.figure(figsize = (14,6))
sns.jointplot(x=yd['humidity'], y=yd['total_riders'],hue=yd['year'],kind="scatt
plt.title("Humidity Vs Total_riders", loc = "center",pad= 90,fontweight="bold",
plt.show()
```

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



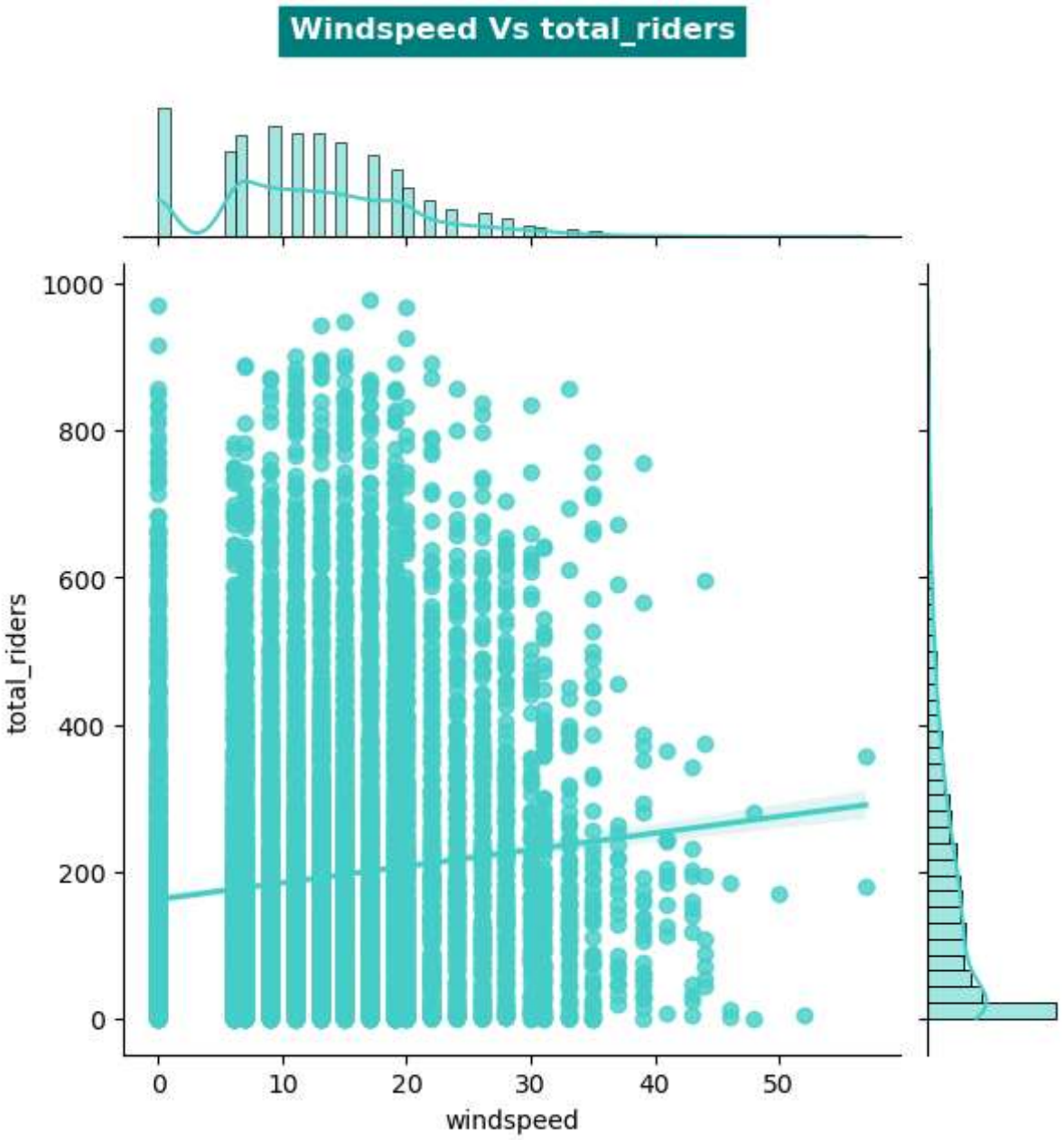

```
plt.figure(figsize = (14,6))
sns.jointplot(x=yd['temp'], y=yd['total_riders'], kind="hex", color='teal')
plt.title("Temperature Vs total_riders", loc = "center",pad= 90,fontweight="bol
plt.show()
```

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




```
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"
plt.show()
```

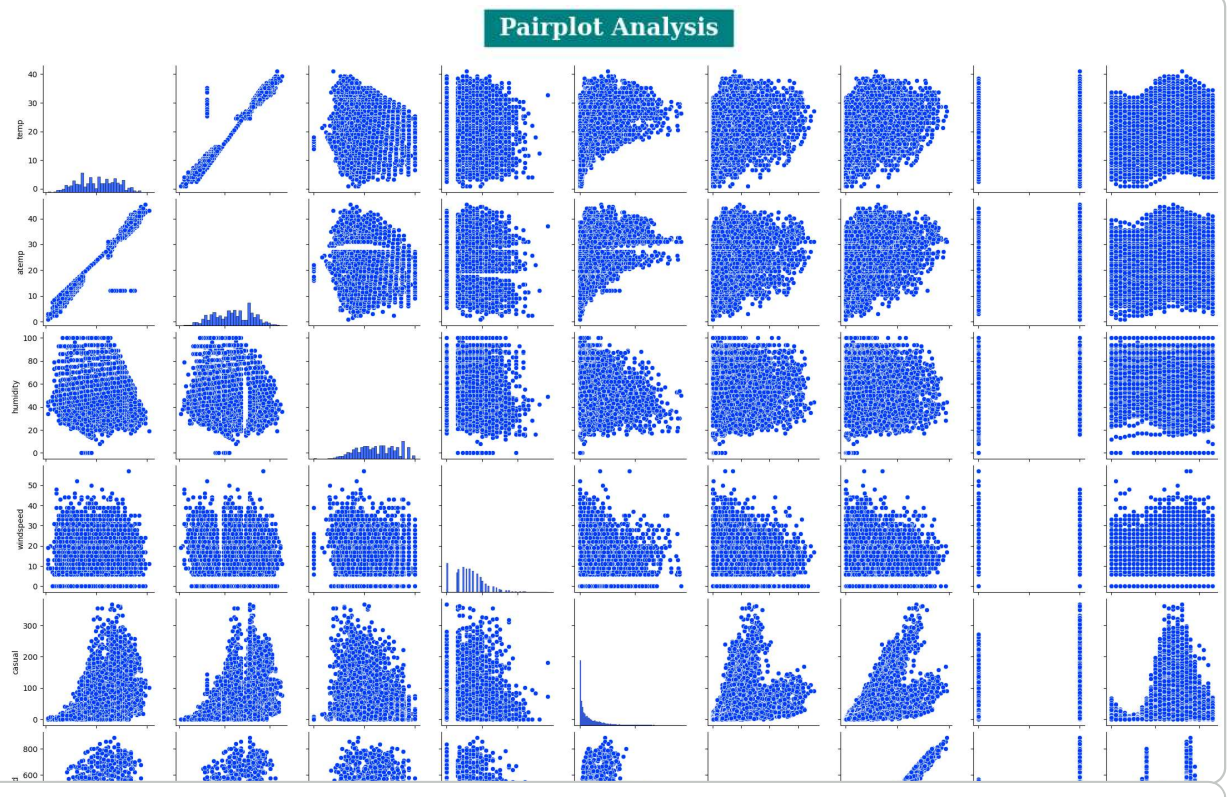
```
<Figure size 1400x600 with 0 Axes>
```



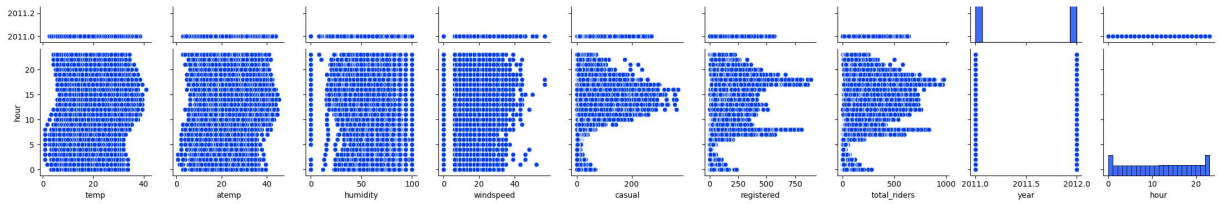

```
cat_cols
```

```
['season', 'holiday', 'workingday', 'weather']
```

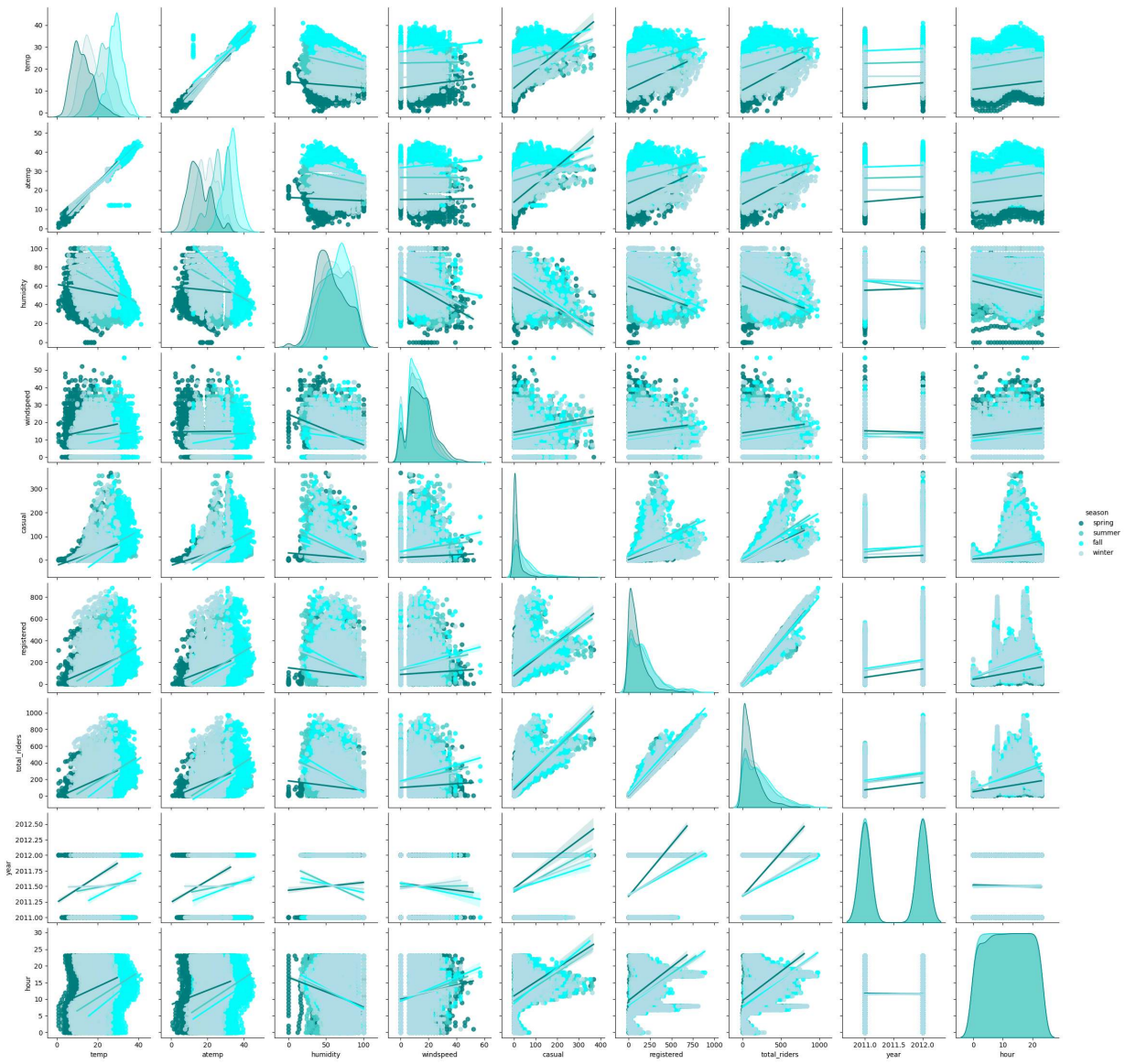
```
plt.figure(figsize=(14,0.05))
plt.axis('off')
plt.title(f' Pairplot Analysis ',fontfamily='serif',fontweight='bold',fontsize=
sns.pairplot(data=yd,palette=cp)
plt.show()
```



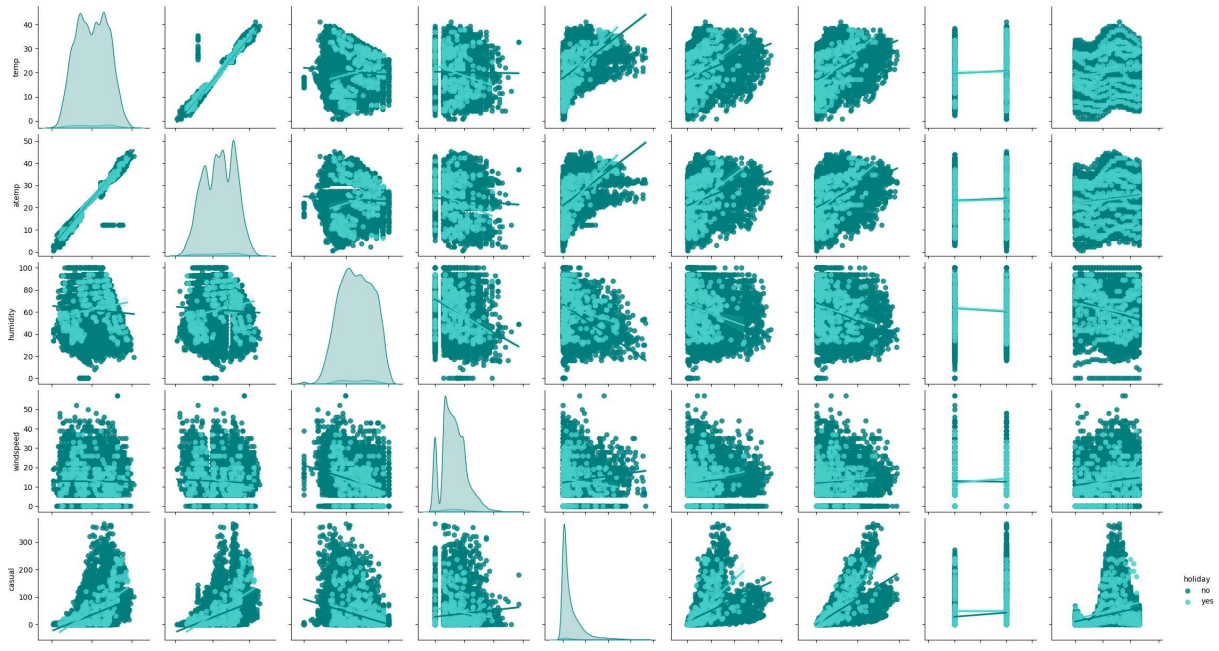

```
for i in range(len(cat_cols)):
    plt.figure(figsize=(14,0.05))
    plt.axis('off')
    plt.title(f' Pairplot based on {cat_cols[i]} ',fontfamily='serif',fontweigh
    sns.pairplot(yd,hue=cat_cols[i],kind='reg',palette=cp)
    plt.show()
    print()
```

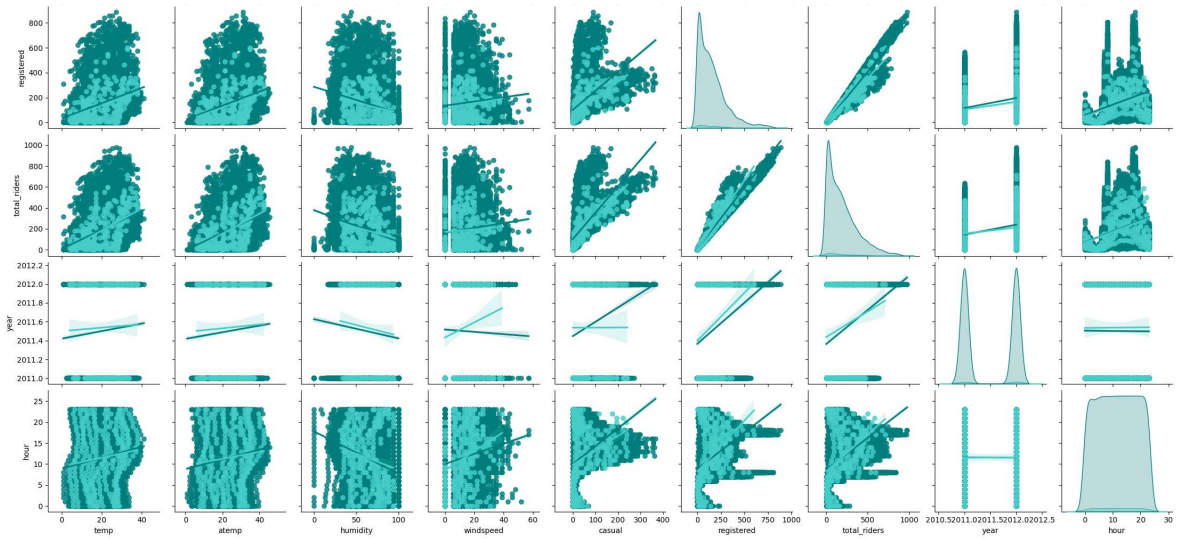


Pairplot based on season



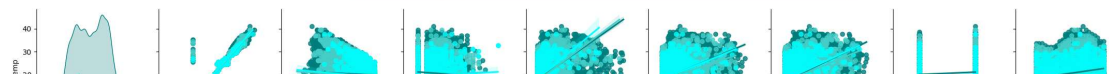
Pairplot based on holiday

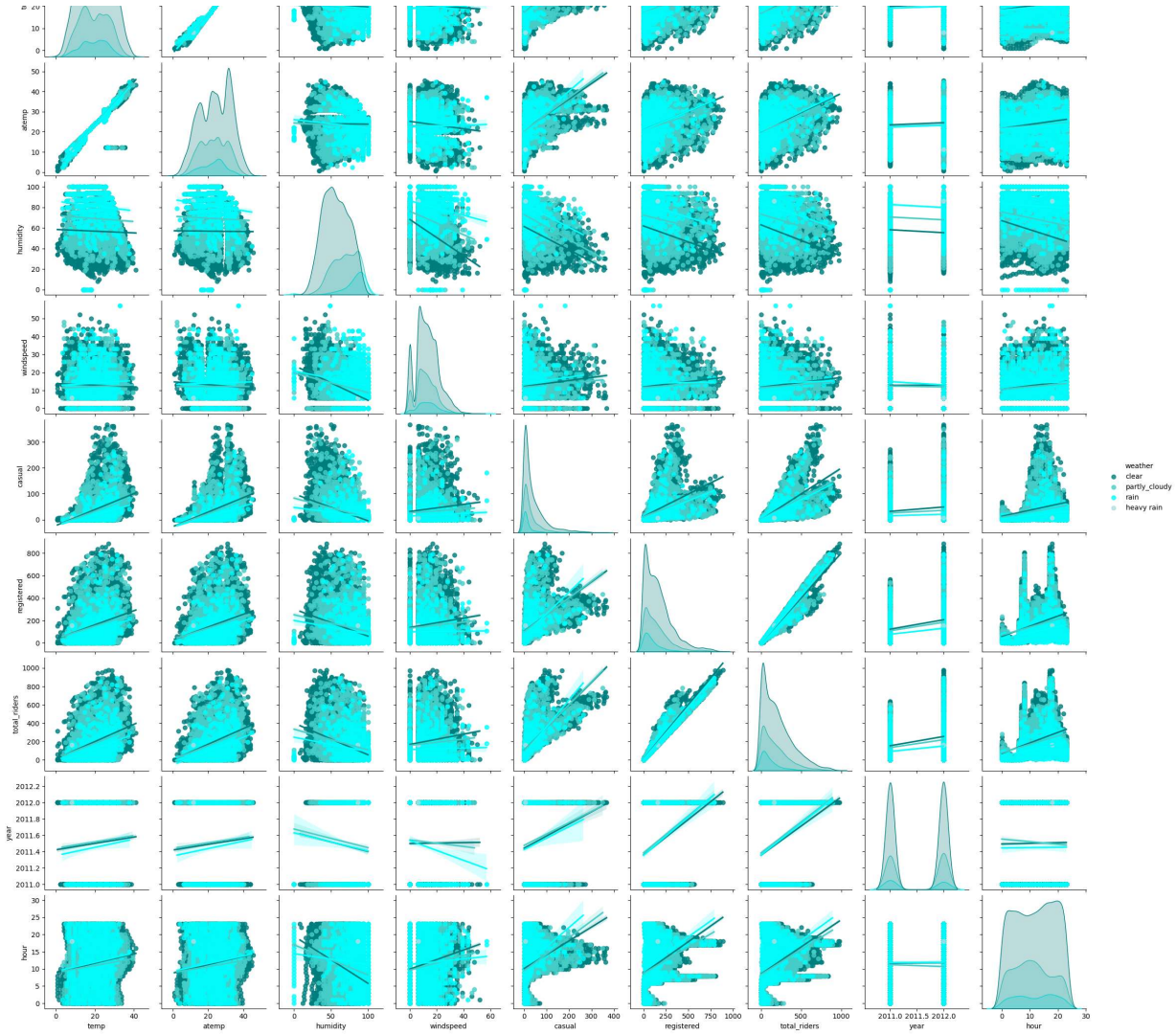




Pairplot based on workingday

Pairplot based on weather




atemp
humidity
windspeed
casual
registered
total_riders
year
hour
temp
weather
clear
partly_cloudy
rain
heavy rain