Yulu

Yulu is India's leading micro-mobility service provider, which offers unique vehicles for the daily commute. Starting off as a mission to eliminate traffic congestion in India, Yulu provides the safest commute solution through a user-friendly mobile app to enable shared, solo and sustainable commuting.

Yulu zones are located at all the appropriate locations (including metro stations, bus stands, office spaces, residential areas, corporate offices, etc) to make those first and last miles smooth, affordable, and convenient!

Yulu has recently suffered considerable dips in its revenues. They have contracted a consulting company to understand the factors on which the demand for these shared electric cycles depends. Specifically, they want to understand the factors affecting the demand for these shared electric cycles in the Indian market.

Problem Statement

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.

Dataset

datetime: datetime

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

holiday: whether day is a holiday or not

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

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

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

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

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

temp: temperature in Celsius

atemp: feeling temperature in Celsius

humidity: humidity

windspeed: wind speed

casual: count of casual users

registered: count of registered users

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

In [2]:
```
#importing libraries
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from scipy.stats import binom, norm, geom
from statsmodels.graphics.gofplots import qqplot
```

In [3]:
```
!wget https://d2beiqkhq929f0.cloudfront.net/public_assets/assets/000/001/428/original/bike_sharing.csv
```

```
--2024-05-13 03:48:29--  https://d2beiqkhq929f0.cloudfront.net/public_assets/assets/000/001/428/original/bike_sharing.csv
Resolving d2beiqkhq929f0.cloudfront.net (d2beiqkhq929f0.cloudfront.net)... 18.172.139.210, 18.172.139.61, 18.172.139.94, ...
Connecting to d2beiqkhq929f0.cloudfront.net (d2beiqkhq929f0.cloudfront.net)|18.172.139.210|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 648353 (633K) [text/plain]
Saving to: ‘bike_sharing.csv’

bike_sharing.csv    100%[===================>] 633.16K  --.-KB/s    in 0.05s

2024-05-13 03:48:29 (13.7 MB/s) - ‘bike_sharing.csv’ saved [648353/648353]
```

In [4]:

```
#Loading Dataset
yulu = pd.read_csv("bike_sharing.csv")
yulu
```

Out[4]:

	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

In [5]:

```
yulu.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10886 entries, 0 to 10885
Data columns (total 12 columns):
 #   Column      Non-Null Count  Dtype  
---  ------      --------------  -----  
 0   datetime    10886 non-null  object 
 1   season      10886 non-null  int64  
 2   holiday     10886 non-null  int64  
 3   workingday  10886 non-null  int64  
 4   weather     10886 non-null  int64  
 5   temp        10886 non-null  float64
 6   atemp       10886 non-null  float64
 7   humidity    10886 non-null  int64  
 8   windspeed   10886 non-null  float64
 9   casual      10886 non-null  int64  
 10  registered  10886 non-null  int64  
 11  count       10886 non-null  int64  
dtypes: float64(3), int64(8), object(1)
memory usage: 1020.7+ KB
```

In [6]:
```
yulu.shape
```

Out[6]:
```
(10886, 12)
```

In [7]:
```
yulu.dtypes
```

Out[7]:
```
datetime       object
season          int64
holiday         int64
workingday      int64
weather         int64
temp          float64
atemp         float64
humidity        int64
windspeed     float64
casual          int64
registered      int64
count           int64
dtype: object
```

In [8]:
```
yulu.duplicated().sum()
```

Out[8]:
```
0
```

In [9]:
```
yulu.isna().sum()
```

Out[9]:
```
datetime      0
season        0
holiday       0
workingday    0
weather       0
temp          0
atemp         0
humidity      0
windspeed     0
casual        0
registered    0
count         0
dtype: int64
```

In [10]:
```
yulu["weather"].unique()
```

Out[10]:
```
array([1, 2, 3, 4])
```

In [11]:
```
yulu["holiday"].unique()
```

Out[11]:
```
array([0, 1])
```

In [12]:
```
yulu["workingday"].unique()
```

Out[12]:
```
array([0, 1])
```

In [13]:
```
yulu["registered"].nunique()
```

Out[13]:
```
731
```

In [14]:
```
yulu["casual"].nunique()
```

Out[14]:
```
309
```

In [15]:
```
yulu.describe()
```

Out[15]:

	season	holiday	workingday	weather	temp	atemp	humidity	windspeed	casual	registered	
count	10886.000000	10886.000000	10886.000000	10886.000000	10886.00000	10886.000000	10886.000000	10886.000000	10886.000000	10886.000000	1088
mean	2.506614	0.028569	0.680875	1.418427	20.23086	23.655084	61.886460	12.799395	36.021955	155.552177	19
std	1.116174	0.166599	0.466159	0.633839	7.79159	8.474601	19.245033	8.164537	49.960477	151.039033	18
min	1.000000	0.000000	0.000000	1.000000	0.82000	0.760000	0.000000	0.000000	0.000000	0.000000	
25%	2.000000	0.000000	0.000000	1.000000	13.94000	16.665000	47.000000	7.001500	4.000000	36.000000	4
50%	3.000000	0.000000	1.000000	1.000000	20.50000	24.240000	62.000000	12.998000	17.000000	118.000000	14
75%	4.000000	0.000000	1.000000	2.000000	26.24000	31.060000	77.000000	16.997900	49.000000	222.000000	28
max	4.000000	1.000000	1.000000	4.000000	41.00000	45.455000	100.000000	56.996900	367.000000	886.000000	97

Observation

There a total 10886 rows and 12 columns in the enitre dataset.

Out of 12 columns, datetime is in object dtype which is later converted to datetime format. Humidity, temp, atemp are in float data types.

The data seems to be clean with no nulls and no duplicate entires and ready for analysis.

There are 4 unique categories in weather and season, 2 in both holidays and wokind_days colums.

There are lot of outliers in registerd, casual, count, windspeed and temp columns.

Conversion of numerical to categorical feature

In [16]:

```
#converting holiday to holiday_cat
def holiday_cat(holiday):
    if holiday == 1:
        return "holiday"
    else:
```

```
        return "not_holiday"
yulu["holiday_cat"] = yulu["holiday"].apply(holiday_cat)
yulu
```

Out[16]:

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

10886 rows × 13 columns

In [17]:

```
#conversion of season to season_cat
def categorize_season(season):
    if season == 1:
        return "spring"
    elif season == 2:
        return "summer"
    elif season == 3:
        return "fall"
    else:
        return "winter"

yulu["season_cat"] = yulu["season"].apply(categorize_season)
```

In [18]:

```
#conversion of working_day to work_day
def work_day(working_day):
```

```
    if working_day ==1:
         return "week_day"
    else:
        return "holiday/ weekend"
yulu["work_day"] = yulu["workingday"].apply(work_day)
```

In [19]:
```
#conversion of weather to weather_cat
def weather(x):
    if x == 1:
        return "cloudy"
    elif x== 2:
        return "mist"
    elif x==3 :
        return "light rain"
    else:
        return "Heavy rain"
yulu["weather_cat"] = yulu["weather"].apply(weather)
```

In [20]:
```
yulu.head()
```

Out[20]:

	datetime	season	holiday	workingday	weather	temp	atemp	humidity	windspeed	casual	registered	count	holiday_cat	season_cat	work_day
0	2011-01-01 00:00:00	1	0	0	1	9.84	14.395	81	0.0	3	13	16	not_holiday	spring	holiday/ weekend
1	2011-01-01 01:00:00	1	0	0	1	9.02	13.635	80	0.0	8	32	40	not_holiday	spring	holiday/ weekend
2	2011-01-01 02:00:00	1	0	0	1	9.02	13.635	80	0.0	5	27	32	not_holiday	spring	holiday/ weekend
3	2011-01-01 03:00:00	1	0	0	1	9.84	14.395	75	0.0	3	10	13	not_holiday	spring	holiday/ weekend
4	2011-01-01 04:00:00	1	0	0	1	9.84	14.395	75	0.0	0	1	1	not_holiday	spring	holiday/ weekend

In [21]:
```
#conversion of object dtype datetime to datetime
yulu["datetime"] = pd.to_datetime(yulu["datetime"])
```

In [22]:
```
#Extracting date from datetimestamp
yulu["date"] =  yulu["datetime"].dt.date
```

In [23]:
```
#Extracting year alone from datetimestamp
yulu["year"] =  yulu["datetime"].dt.year
yulu
```

Out[23]:

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

10886 rows × 18 columns

In [24]:
```
yulu["month_name"] = yulu["datetime"].dt.strftime("%B")
yulu.head()
```

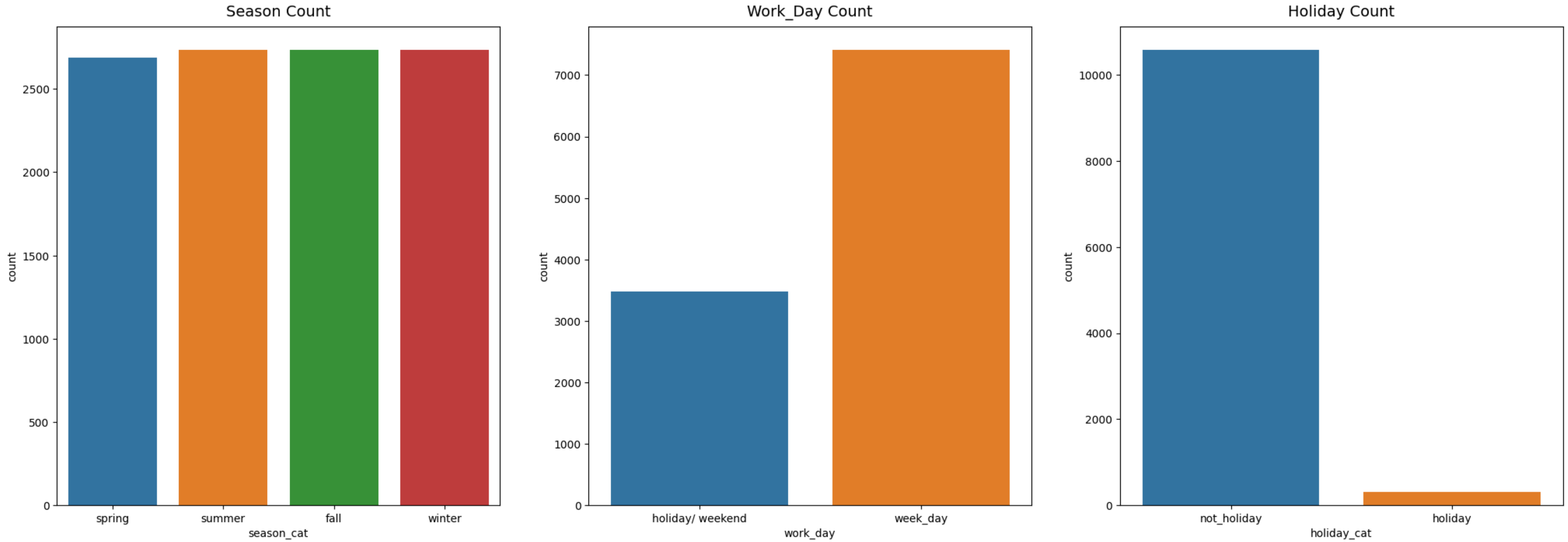
Out[24]:

	datetime	season	holiday	workingday	weather	temp	atemp	humidity	windspeed	casual	registered	count	holiday_cat	season_cat	work_day
0	2011-01-01 00:00:00	1	0	0	1	9.84	14.395	81	0.0	3	13	16	not_holiday	spring	holiday/weekend
1	2011-01-01 01:00:00	1	0	0	1	9.02	13.635	80	0.0	8	32	40	not_holiday	spring	holiday/weekend
2	2011-01-01 02:00:00	1	0	0	1	9.02	13.635	80	0.0	5	27	32	not_holiday	spring	holiday/weekend
3	2011-01-01 03:00:00	1	0	0	1	9.84	14.395	75	0.0	3	10	13	not_holiday	spring	holiday/weekend
4	2011-01-01 04:00:00	1	0	0	1	9.84	14.395	75	0.0	0	1	1	not_holiday	spring	holiday/weekend

Univariate Analysis

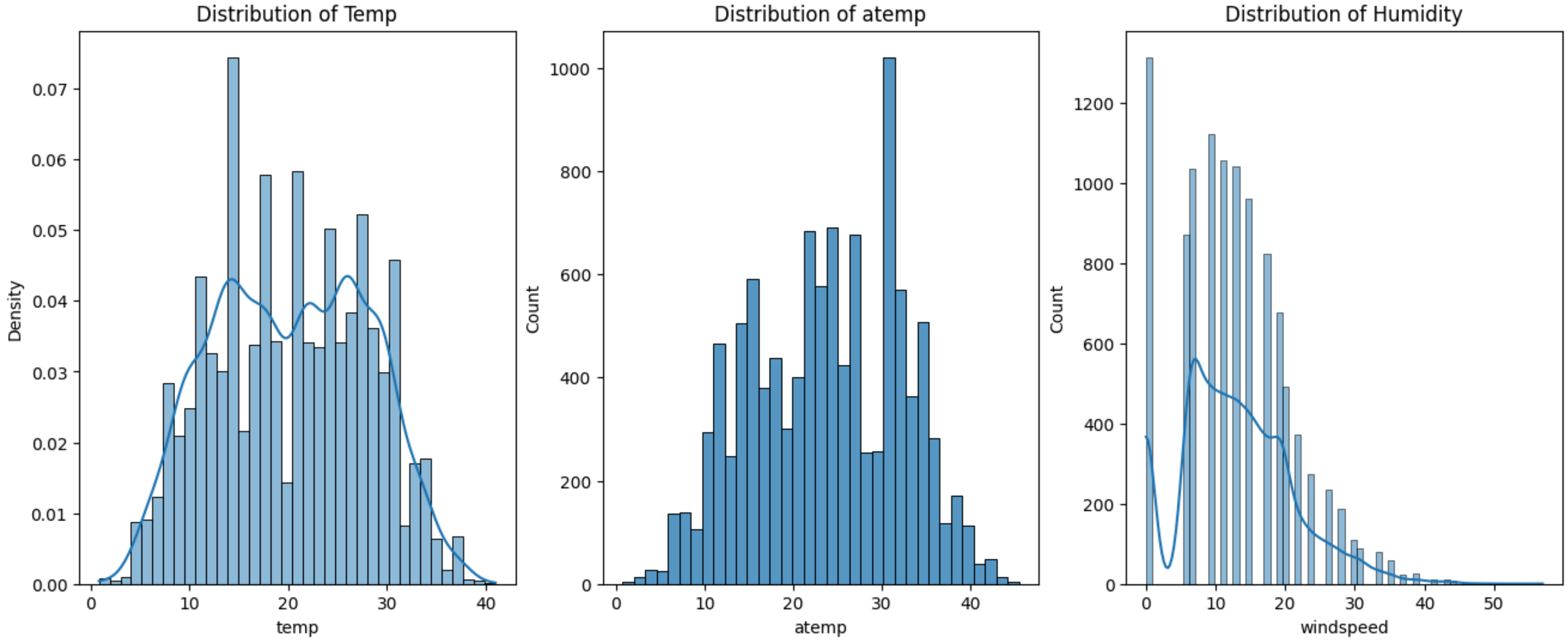
In [25]:
```
fig, axs = plt.subplots(nrows=1, ncols=3, figsize=(25, 8))
sns.countplot(data = yulu, x = "season_cat", palette = "tab10", hue = "season_cat", ax = axs[0])
sns.countplot(data = yulu, x = "work_day", palette = "tab10", hue = "work_day", ax = axs[1])
sns.countplot(data = yulu, x = "holiday_cat", palette = "tab10", hue = "holiday_cat", ax = axs[2])

axs[0].set_title("Season Count", pad=10, fontsize=14)
axs[1].set_title("Work_Day Count", pad=10, fontsize=14)
axs[2].set_title("Holiday Count", pad=10, fontsize=14)
plt.show()
```



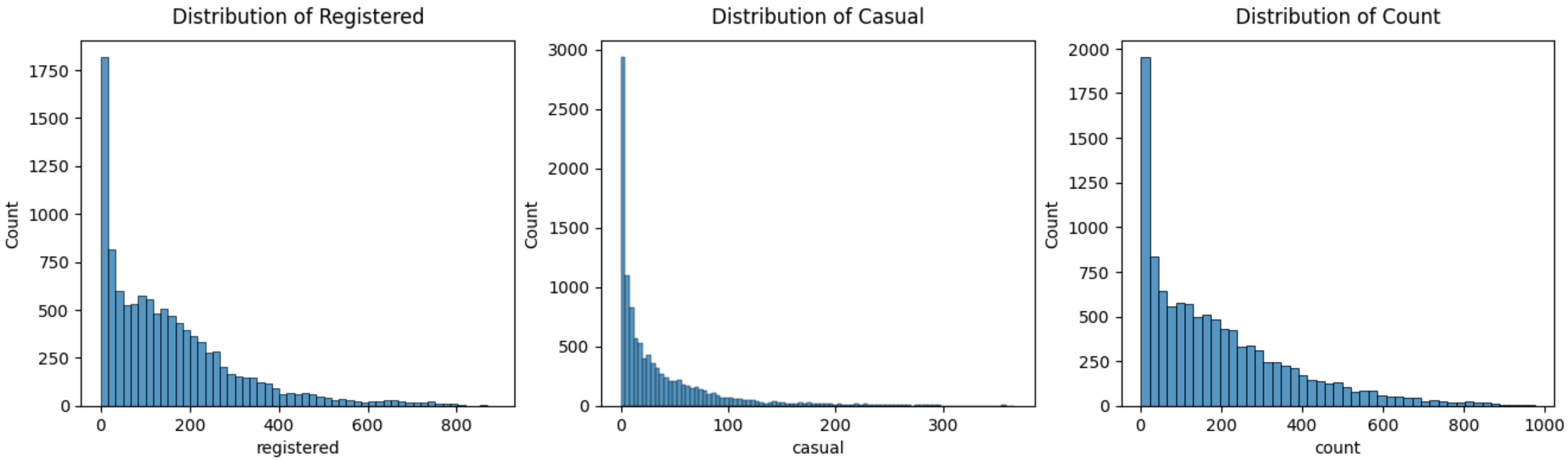
In [26]:
```
fig, axs = plt.subplots(nrows= 1, ncols =3, figsize = (16,6))
sns.histplot(yulu["temp"], kde = True, stat= "density", ax= axs[0])
sns.histplot(yulu["atemp"], ax= axs[1])
sns.histplot(yulu["windspeed"], kde = True, ax= axs[2])

axs[0].set_title("Distribution of Temp")
axs[1].set_title("Distribution of atemp")
axs[2].set_title("Distribution of Humidity")
plt.show()
```



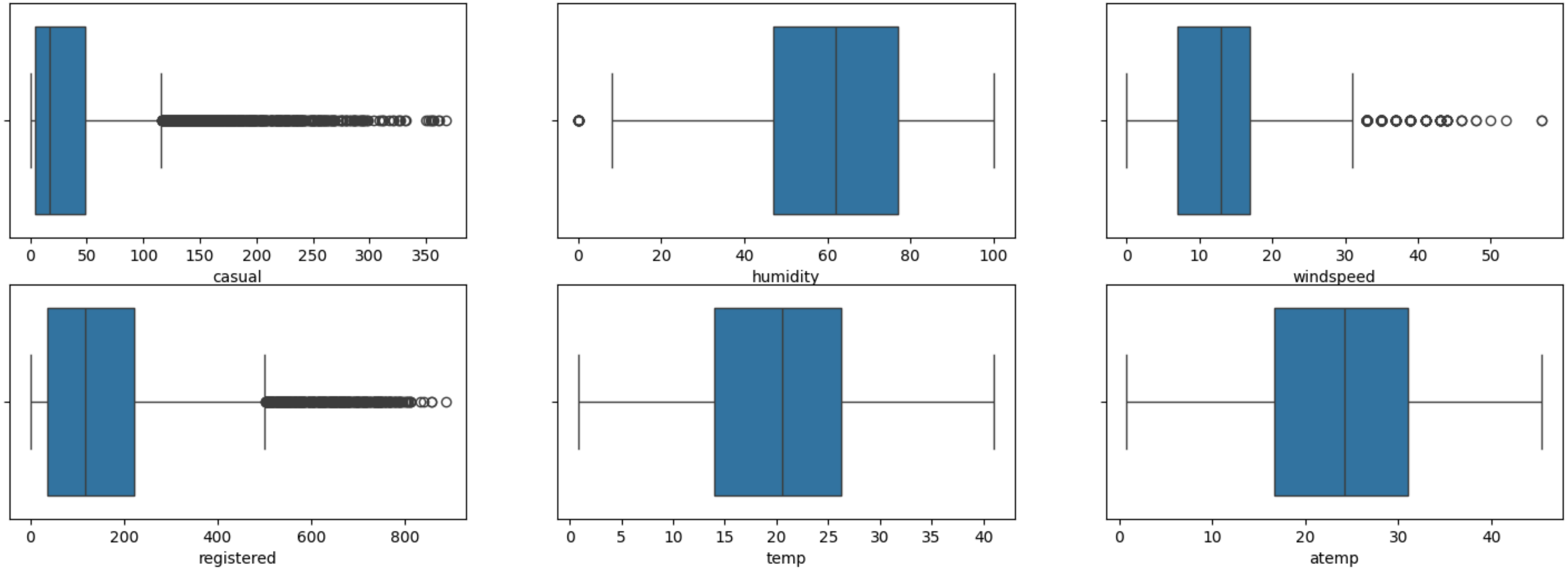
In [27]:
```
fig, axs = plt.subplots(nrows= 1, ncols= 3, figsize= (16,4))
sns.histplot(yulu["registered"], ax= axs[0])
sns.histplot(yulu["casual"], ax= axs[1])
sns.histplot(yulu["count"], ax= axs[2])

axs[0].set_title("Distribution of Registered", pad=10, fontsize=12)
axs[1].set_title("Distribution of Casual", pad=10, fontsize=12)
axs[2].set_title("Distribution of Count", pad=10, fontsize=12)
plt.show()
```



Outlier Detection

In [28]:
```
fig, axs = plt.subplots(nrows=2, ncols=3, figsize=(18, 6))
sns.boxplot(data =yulu, x= "casual", ax= axs[0][0])
sns.boxplot(data = yulu, x = "registered", ax=  axs[1][0])
sns.boxplot(data = yulu, x = "humidity", ax=  axs[0][1])
sns.boxplot(data = yulu, x = "windspeed", ax=  axs[0][2])
sns.boxplot(data = yulu, x = "temp", ax=  axs[1][1])
sns.boxplot(data = yulu, x = "atemp", ax=  axs[1][2])
plt.show()
```



- Causal, Registered and windspeed have large number of outliers.
- Outliers should be removed, since they contribute to the count.

Outlier Treatment

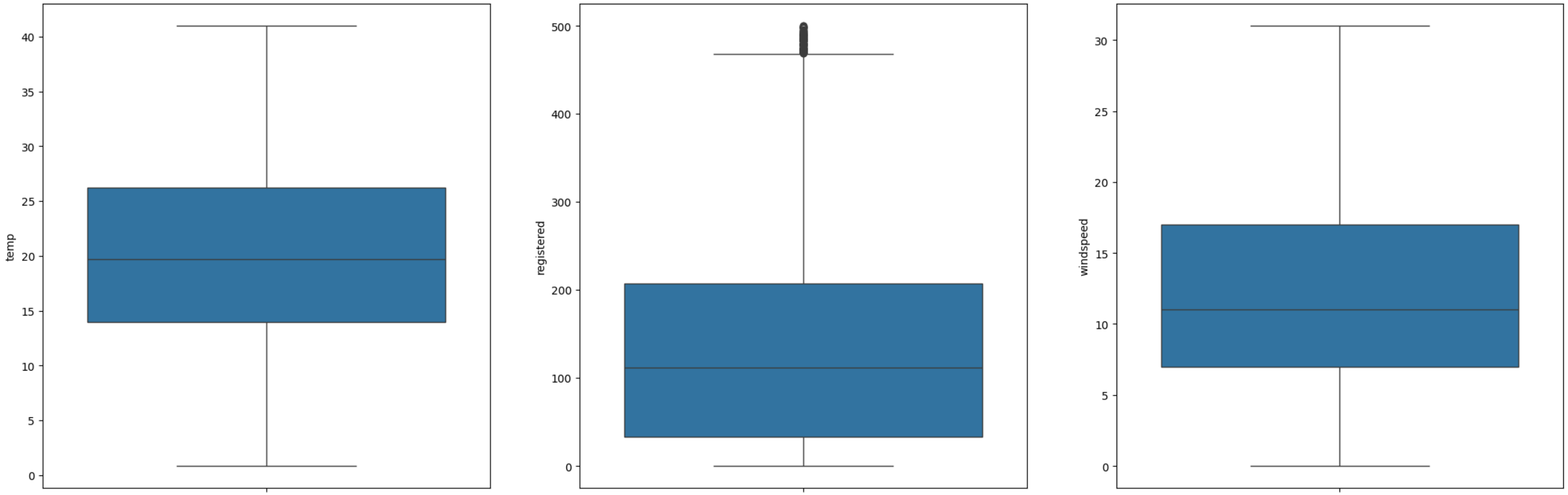
In [29]:
```
q2 = np.percentile(yulu["casual"],25)
q3 = np.percentile(yulu["casual"], 75)
iqr =  q3- q2
upper =  q3+ 1.5*iqr
lower = q2 - 1.5*iqr
casual = yulu[(yulu['casual'] <upper) & (yulu['casual']> lower)]
```

In [30]:
```
q2 = np.percentile(yulu["registered"],25)
q3 = np.percentile(yulu["registered"], 75)
iqr =  q3- q2
upper =  q3+ 1.5*iqr
```

```
lower = q2 - 1.5*iqr
reg = yulu[(yulu["registered"] < upper) & (yulu["registered"] >lower)]
```

In [31]:
```
q2 = np.percentile(yulu["windspeed"],25)
q3 = np.percentile(yulu["windspeed"], 75)
iqr =  q3- q2
upper =  q3+ 1.5*iqr
lower = q2 - 1.5*iqr
wind_speed  = yulu[(yulu["windspeed"] < upper) & (yulu["windspeed"] >lower)]
```

In [32]:
```
fig, axs =  plt.subplots(nrows = 1, ncols = 3, figsize = (25,8))
sns.boxplot(data = casual, y = "temp", ax = axs[0])
sns.boxplot(data = reg, y = "registered", ax = axs[1])
sns.boxplot(data = wind_speed, y = "windspeed", ax=  axs[2])
plt.show()
```



Bivariate Analysis

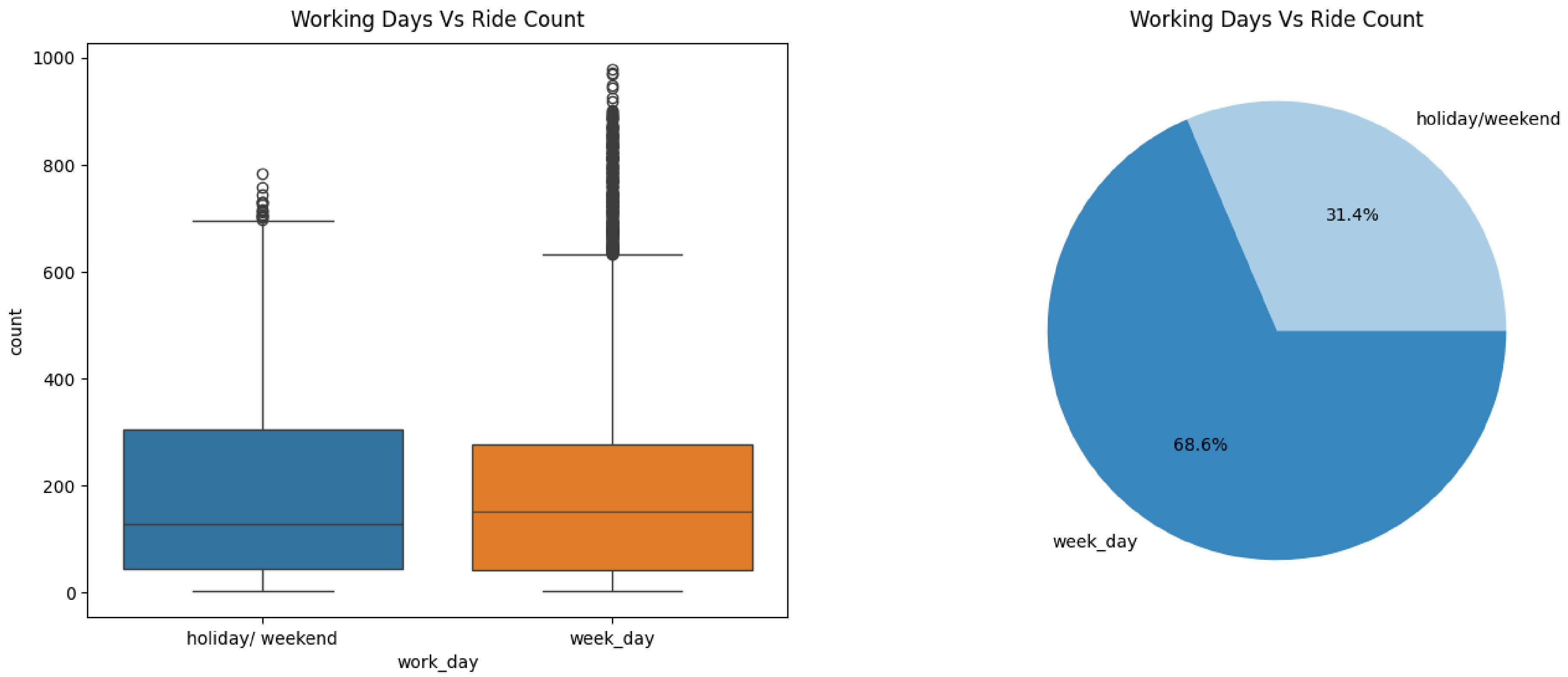
In [33]:
```
work_day_cnt = yulu.groupby("work_day")["count"].sum().reset_index()
work_day_cnt
```

Out[33]:

	work_day	count
0	holiday/ weekend	654872
1	week_day	1430604

In [34]:
```
fig, axs=  plt.subplots(nrows =  1, ncols=2, figsize = (16,6))
sns.boxplot(data =  yulu, x= 'work_day', y= "count",hue= "work_day", ax= axs[0])
colors = sns.color_palette('Blues', 2)
axs[1].pie(x = work_day_cnt["count"], labels = ["holiday/weekend", 'week_day'], autopct='%1.1f%%', colors = colors)

axs[0].set_title("Working Days Vs Ride Count", pad = 10, fontsize= 12)
axs[1].set_title("Working Days Vs Ride Count", pad = 10, fontsize= 12)
plt.show()
```



In [35]:
```
season_cnt = yulu.groupby("season_cat")["count"].sum().reset_index()
season_cnt
```

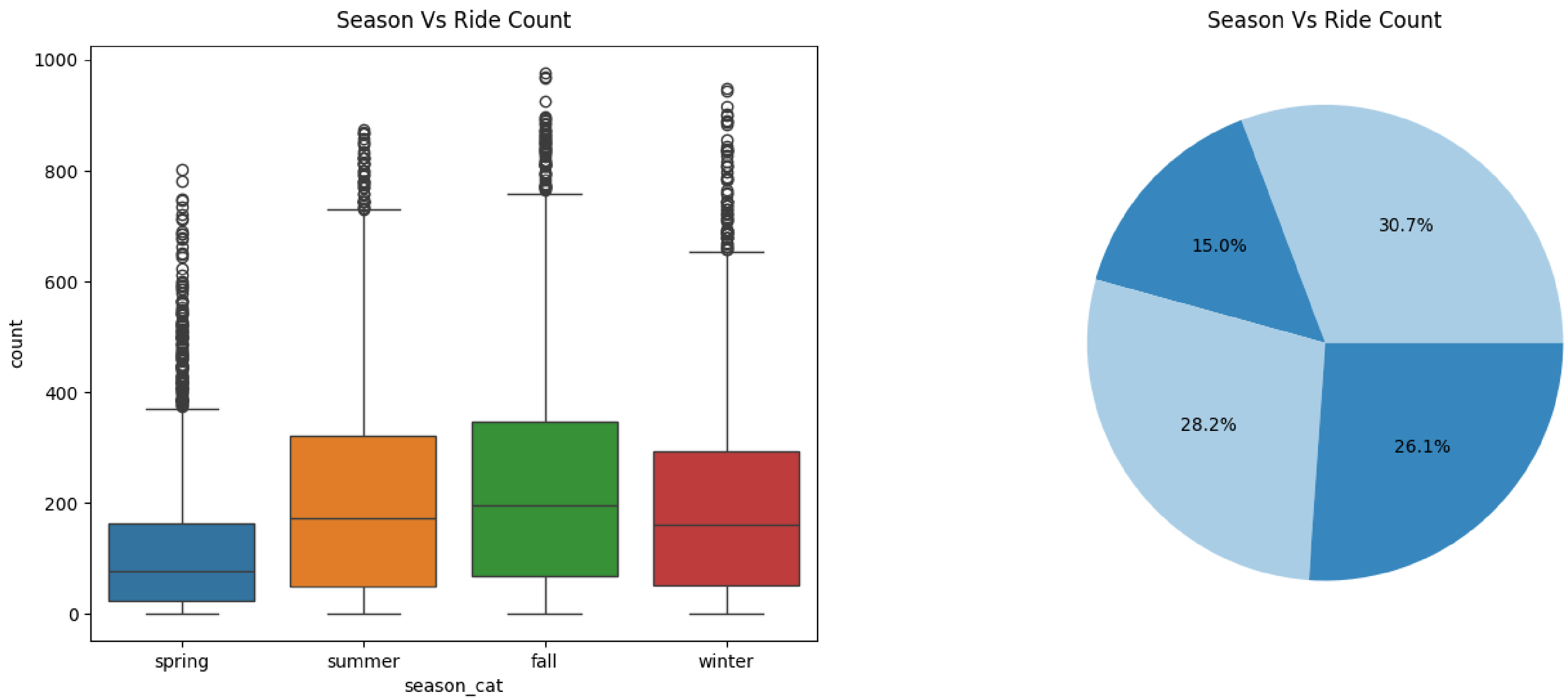
Out[35]:

	season_cat	count
0	fall	640662
1	spring	312498
2	summer	588282
3	winter	544034

In [36]:

```
fig, axs = plt.subplots(nrows= 1, ncols= 2, figsize= (16,6))
sns.boxplot(data= yulu, x= "season_cat", y= "count",hue = "season_cat", ax= axs[0])
colors = sns.color_palette('Blues', 2)
axs[1].pie(x = season_cnt["count"], autopct='%1.1f%%', colors = colors)

axs[0].set_title("Season Vs Ride Count", pad = 10, fontsize= 12)
axs[1].set_title("Season Vs Ride Count", pad = 10, fontsize= 12)
plt.show()
```



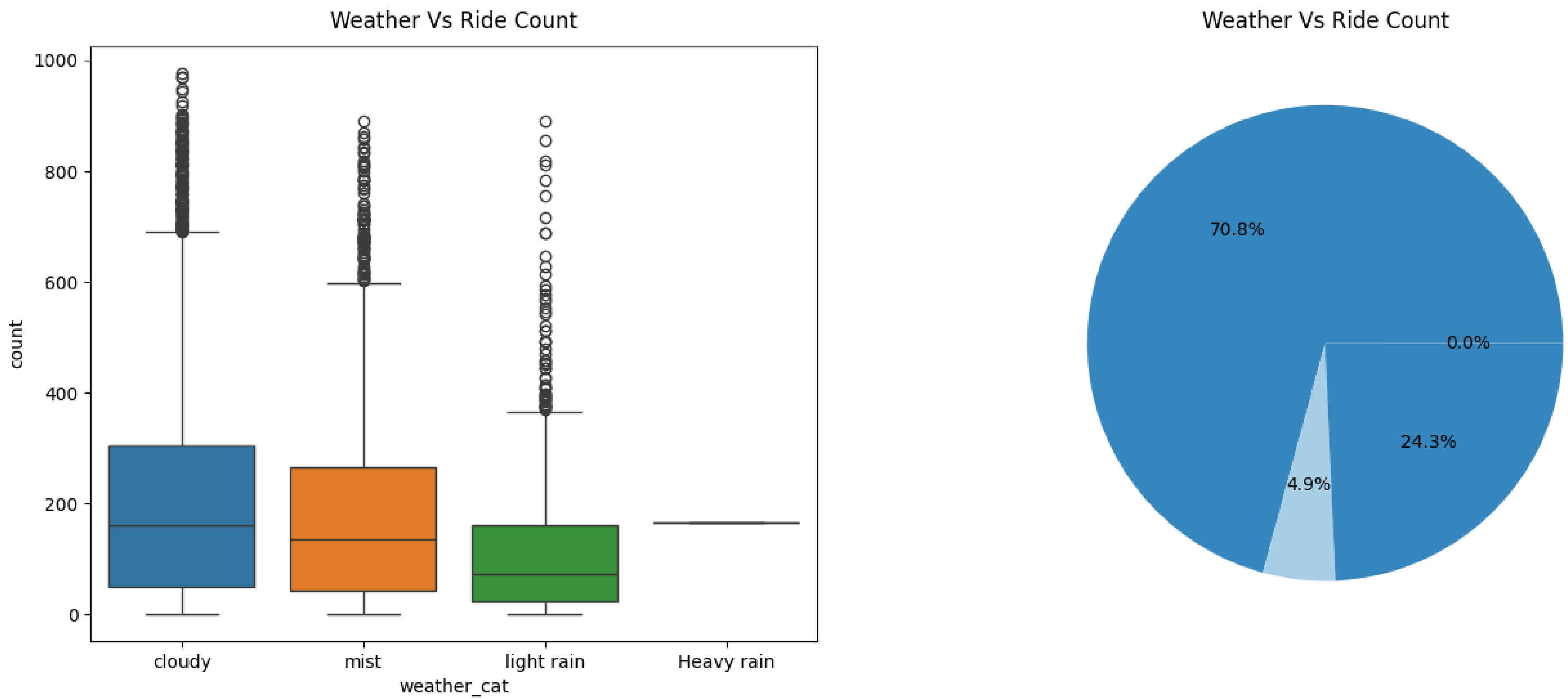
In [37]:
```
weather_cnt = yulu.groupby("weather_cat")["count"].sum().reset_index()
weather_cnt
```

Out[37]:

	weather_cat	count
0	Heavy rain	164
1	cloudy	1476063
2	light rain	102089
3	mist	507160

In [38]:
```
fig, axs = plt.subplots(nrows= 1, ncols= 2, figsize= (16,6))
sns.boxplot(data= yulu, x= "weather_cat", y= "count",hue = "weather_cat", ax= axs[0])
colors = sns.color_palette('Blues', 2)
axs[1].pie(x = weather_cnt["count"], autopct='%1.1f%%', colors = colors)
```

```
axs[0].set_title("Weather Vs Ride Count", pad = 10, fontsize= 12)
axs[1].set_title("Weather Vs Ride Count", pad = 10, fontsize= 12)
plt.show()
```



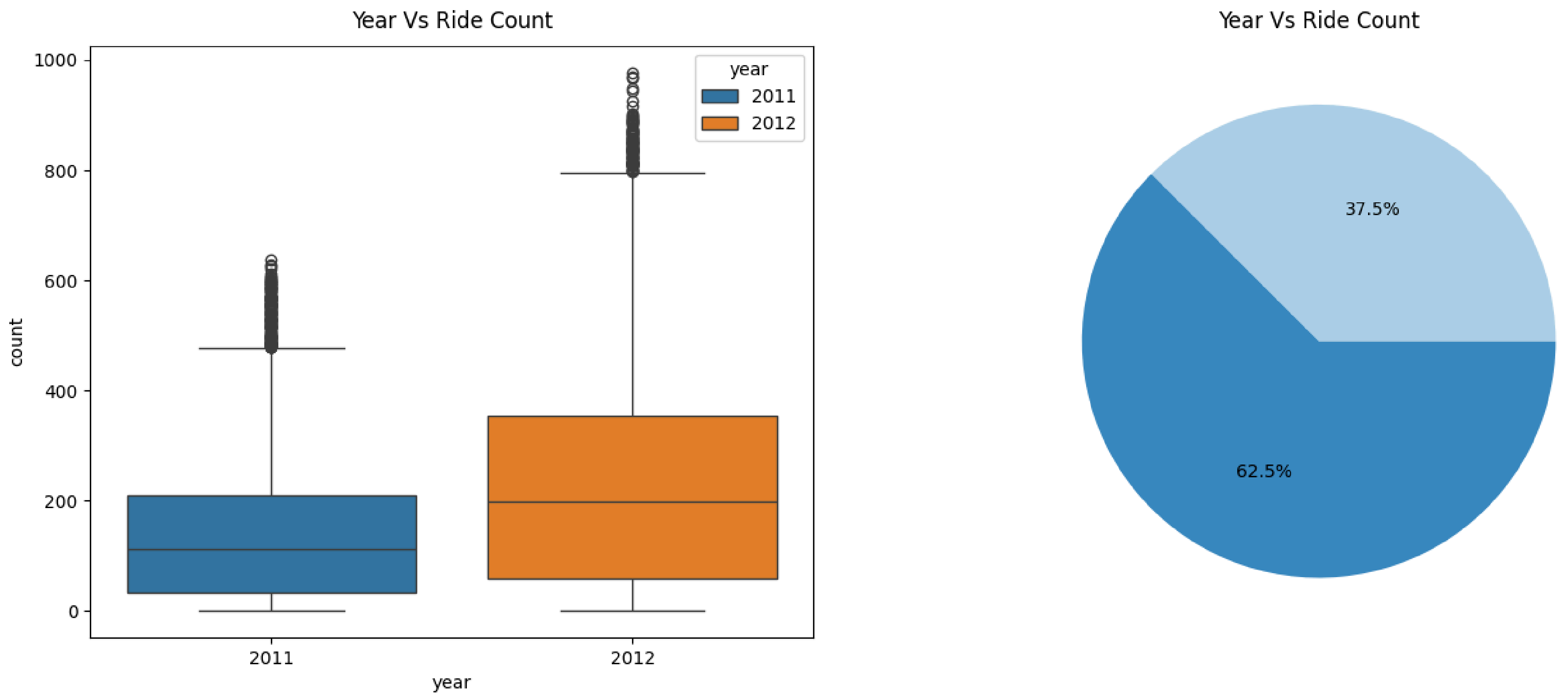
In [39]:
```
year_cnt = yulu.groupby("year")["count"].sum().reset_index()
year_cnt
```

Out[39]:

	year	count
0	2011	781979
1	2012	1303497

In [40]:
```
fig, axs = plt.subplots(nrows= 1, ncols= 2, figsize= (16,6))
sns.boxplot(data= yulu, x= "year", y= "count", hue = "year", palette= "tab10", ax= axs[0])
colors = sns.color_palette('Blues', 2)
```

```
axs[1].pie(x = year_cnt["count"], autopct='%1.1f%%', colors = colors)
axs[0].set_title("Year Vs Ride Count", pad = 10, fontsize= 12)
axs[1].set_title("Year Vs Ride Count", pad = 10, fontsize= 12)
plt.show()
```

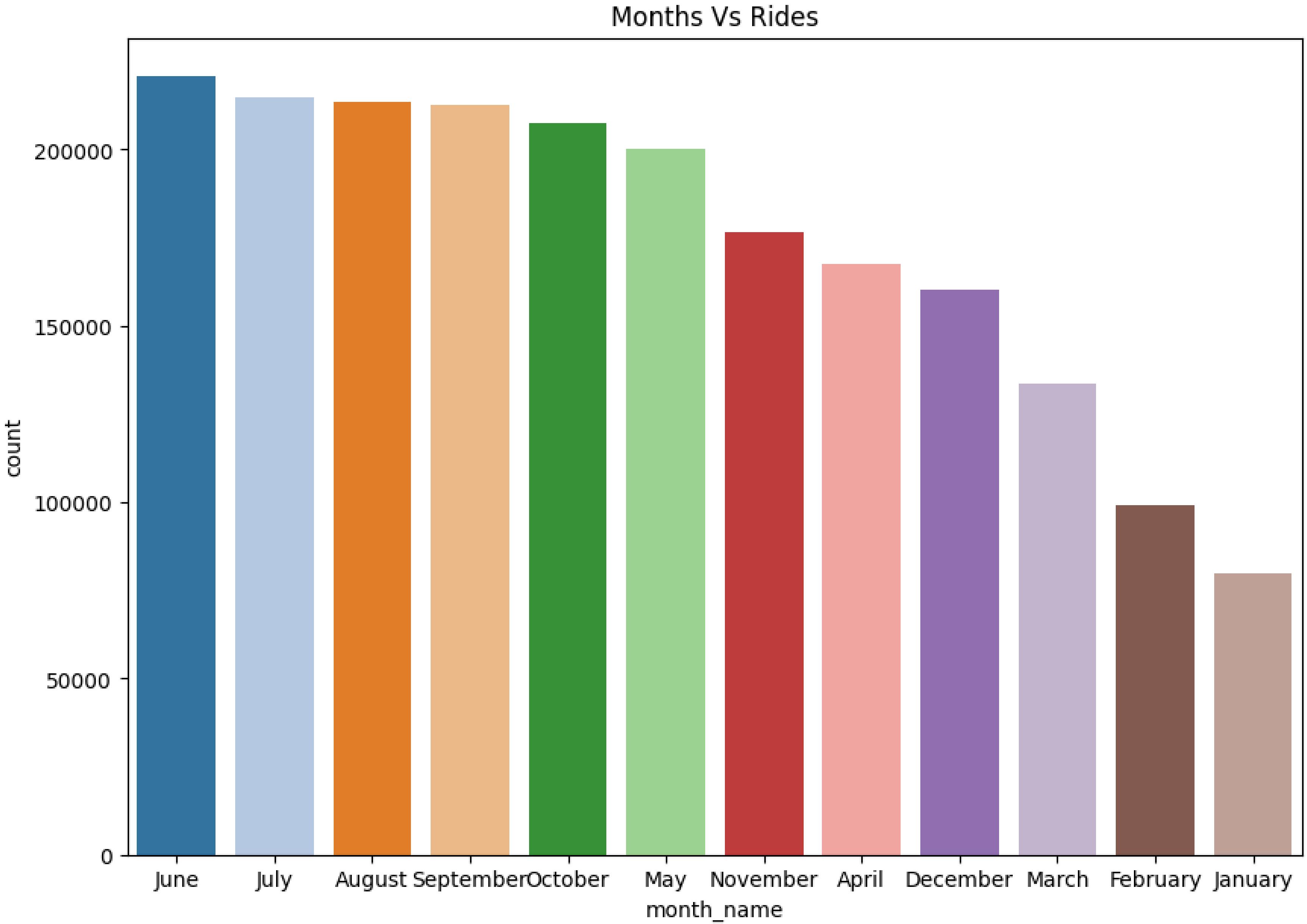


In [41]:
```
cnt = yulu.groupby("month_name")["count"].sum().reset_index()
cnt =  cnt.sort_values(by= "count", ascending= False).reset_index(drop= True)
cnt
```

Out[41]:

	month_name	count
0	June	220733
1	July	214617
2	August	213516
3	September	212529
4	October	207434
5	May	200147
6	November	176440
7	April	167402
8	December	160160
9	March	133501
10	February	99113
11	January	79884

In [72]:
```
plt.figure(figsize= (10,7))
sns.barplot(data = cnt, x= "month_name", y = "count",  hue = "month_name", palette= "tab20")
plt.title("Months Vs Rides")
plt.show()
```



Observation

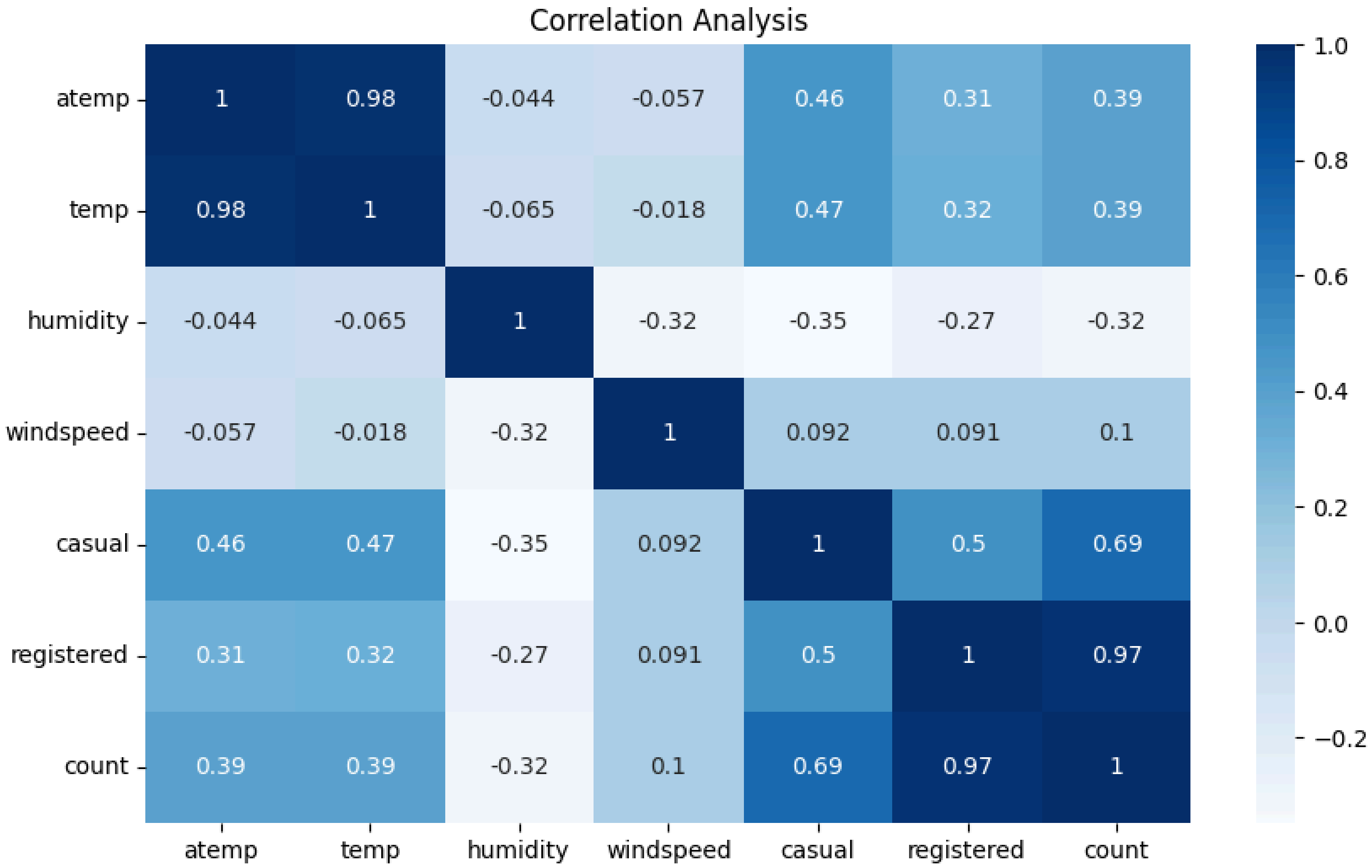
- Overall a 68.6% of bike rentals happend during a week_days and 31.4% of bike rentals happend during weekends or holidays.
- Summer and winter season has almost same percent of bike rentals with 28.2% and 26.1%. Fall Season found to have most number of rides or bike rentals with almost 31%.
- Cloudy weather conditions have highest count of bike rentals with a count of 1476063 and 71% spread, followed by mist with a count of 102089
- Heavy rain weather condititon have the lowest count 164 bike rentals

Monthly analysis on rentals

- Peak Rental Months: June stands out as the peak month for bike rentals, with the highest count of 220,733, followed closely by July and August.
- Seasonal Trend: Summer months (June, July, August) show higher bike rental counts, consistent with favorable weather conditions.
- Off-Peak Rental Months: January, February, and March have notably lower bike rental counts, indicating potential off-peak periods, possibly influenced by colder weather or fewer outdoor activities.

In [43]:
```
plt.figure(figsize = (10, 6))
sns.heatmap(data = yulu[["atemp", "temp", "humidity", "windspeed", "casual", "registered", "count"]].corr(), cmap= "Blues", annot
plt.title("Correlation Analysis")
plt.plot()
```

Out[43]:
```
[]
```



Observation

- Temp and atemp are strongly correlated to each other and negatively correlated with humidity and windspeed. Strong positive correlation with 'atemp' (0.46) and 'temp' (0.47).
- Moderate negative correlation with 'humidity' (-0.35) and positive correlation with 'windspeed' (0.09).
- Highly correlated with 'registered' (0.50) and 'count' (0.69), indicating a significant impact on overall rentals.

- Positive correlation with 'atemp' (0.31) and 'temp' (0.32). Negative correlation with 'humidity' (-0.27) and positive correlation with 'windspeed' (0.09).
- Highly correlated with 'casual' (0.50) and 'count' (0.97).
- Positive correlation with 'atemp' (0.39), 'temp' (0.39), and 'casual' (0.69).Negative correlation with 'humidity' (-0.32).
- Highly correlated with 'registered' (0.97), emphasizing the joint impact of casual and registered rentals on the overall count.

T-Test

Check if there any significant difference between the no. of bike rides on Weekdays and Weekends?

In [44]:
```
#importing ttest_ind
from scipy.stats import ttest_ind
```

Setting Hypothesis

- Let the average ride count on holidays and non_holidays be $\mu 1$ and $\mu 2$ respectively.
- **H0:$\mu 1=\mu 2$**, the average ride count on holidays is same as non-holidays
- **H0:$\mu 1<\mu 2$**, the average ride count on holidays is less than non-holidays

Selection of Test

- There is a categorical(with 2 categories) and a numerical variable. We can perform T-test

T- Test Statistics

In [45]:
```
holidays = yulu[yulu["work_day"] == "holiday/ weekend"]["count"]
not_holidays = yulu[yulu["work_day"] == "week_day"]["count"]
t_stat,p_value =  ttest_ind(holidays, not_holidays, alternative= "less")
print("t_stat:",t_stat)
print("p_value:" , p_value)
```

```
t_stat: -1.2096277376026694
p_value: 0.11322402113180674
```

Checking Significance Level

In [46]:
```
alpha = 0.05
if p_value < alpha:
     print("reject H0")
     print("non_holidays have high ride count")
else:
     print("Fail to reject H0")
     print('Difference we observe is just chance')
```

```
Fail to reject H0
Difference we observe is just chance
```

Conclusion

- The average ride count of holidays is same as non_holidays. The difference we observe is just chance

Anova

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

In [47]:
```
yulu["season_cat"].unique()
```

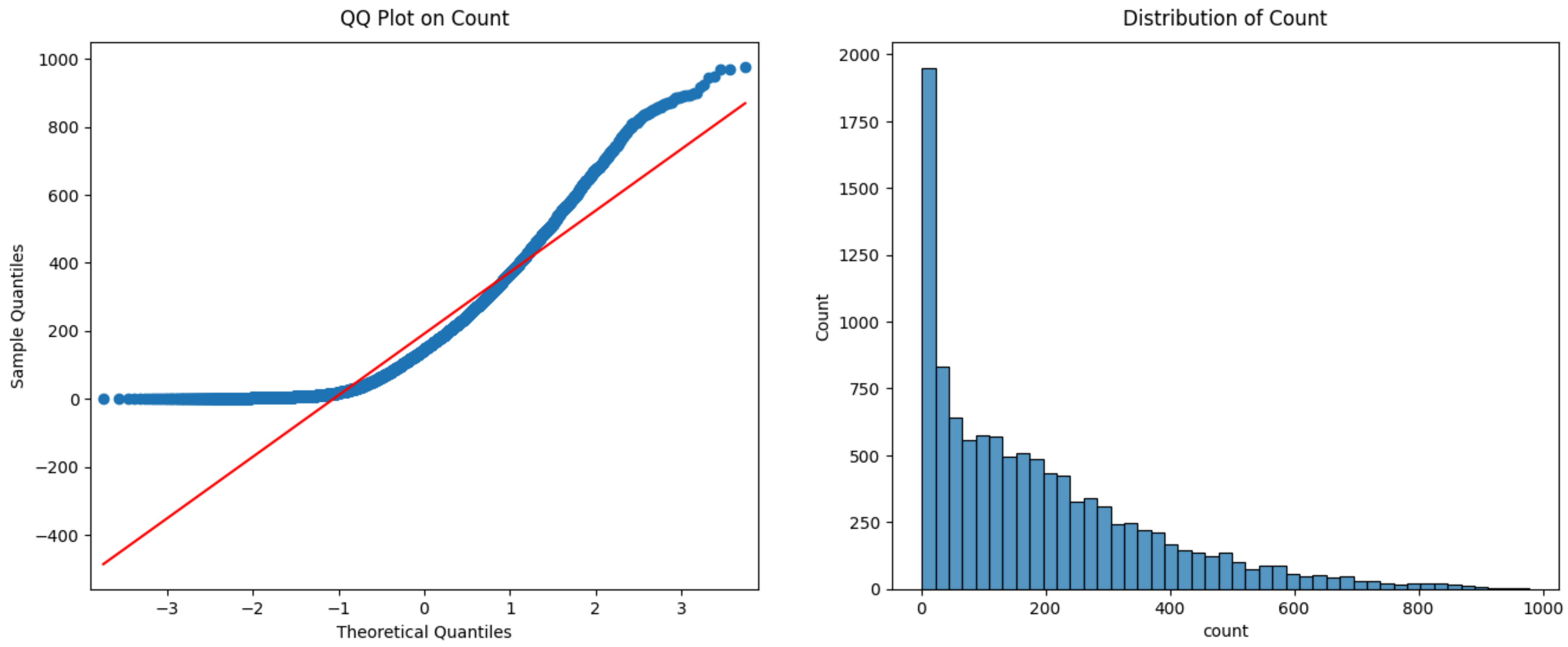
Out[47]:
```
array(['spring', 'summer', 'fall', 'winter'], dtype=object)
```

In [48]:
```
spring =  yulu[yulu["season_cat"]=="spring"]["count"]
summer =  yulu[yulu["season_cat"]=="summer"]["count"]
fall =  yulu[yulu["season_cat"]=="fall"]["count"]
winter =  yulu[yulu["season_cat"]=="winter"]["count"]
```

Assumptions of Anova

Checking distribution of the data

In [49]:
```
fig, axs = plt.subplots(nrows = 1, ncols = 2, figsize= (16,6))
qqplot(yulu["count"], line = "s", ax= axs[0])
sns.histplot(yulu["count"], ax= axs[1])
axs[0].set_title("QQ Plot on Count", pad = 10, fontsize= 12)
axs[1].set_title("Distribution of Count", pad = 10, fontsize= 12)
plt.show()
```



From the above plot, we can clear say that the data is right skewed.

In [50]:
```
yulu["count"].kurt()
```

Out[50]:
```
1.3000929518398334
```

In [51]:
```
yulu["count"].skew()
```

Out[51]:
```
1.2420662117180776
```

Distribution of data is identified using QQ-Plot, skew and Histplot.

The data is deviating from the straight line in QQ-Plot indicating that the data is not normally distributed.

From the histplot we can clearly identify the data is right skewed.

Levenes Test

Setting Hypothesis

H0: Varience of all the categories are same.

Ha: Varience of atleast one category is different.

Checking varience

Homogeneity of Variances using Levene's test

Levene's Test statistics and P_value

In [52]:

```
#importing levene test
from scipy.stats import levene
stat_val, p_val = levene(spring, summer, fall, winter)
print("stat_val:",stat_val)
print("p_value:" , p_val)
```

```
stat_val: 187.7706624026276
p_value: 1.0147116860043298e-118
```

Checking the significance levels of p_value

In [53]:

```
alpha = 0.05
if p_value < alpha:
     print("reject H0")
     print("varience are not equal")
```

Test Conclusion

Since p value is less than the significance level, we will reject the null hypothesis and varience are not equal.

Kruskal-Wallis Test

Null Hypothesis (H0):

The populations of all groups have the same median. In other words, there is no statistically significant difference in the medians of the groups being compared.

Alternative Hypothesis (H1):

At least one of the populations has a different median. There is a statistically significant difference in the medians of at least two groups.

In [54]:
```
from scipy.stats import kruskal
# If assumptions of ANOVA fail, use kruskal
stat, p_value = kruskal(spring, summer, fall, winter)

print("test statistic:",stat)
print("p_value:",p_value)
```

```
test statistic: 699.6668548181988
p_value: 2.479008372608633e-151
```

In [55]:
```
if p_value < 0.05:
    print("Reject H0")
    print("Atleast one group have different median")
else:
    print("Fail to reject H0")
    print("All groups have same median")
```

```
Reject H0
Atleast one group have different median
```

One way ANOVA

In [56]:
```
from scipy.stats import f_oneway
```

In [57]:
```
f_stat, p_value =  f_oneway(spring, summer, fall, winter)
print("p_value:", p_value)
```

```
p_value: 6.164843386499654e-149
```

In [58]:
```
if p_value < 0.05:
    print("Reject H0")
    print("Atleast one group have different mean")
else:
    print("Fail to reject H0")
    print("All groups have same mean")
```

```
Reject H0
Atleast one group have different mean
```

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

In [59]:
```
cloudy = yulu[yulu["weather"]==1]["count"]
mist =  yulu[yulu["weather"]==2]["count"]
light_rain =  yulu[yulu["weather"]==3]["count"]
heavy_rain =  yulu[yulu["weather"]==4]["count"]
```

In [60]:
```
#importing Levene test
from scipy.stats import levene
stat_val, p_val = levene(cloudy, mist, light_rain, heavy_rain)
print("stat_val:",stat_val)
print("p_value:" , p_val)
```

```
stat_val: 54.85106195954556
p_value: 3.504937946833238e-35
```

In [61]:
```
alpha = 0.05
if p_value < alpha:
     print("reject H0")
     print("varience are not same")
else:
```

```
    print("Fail to reject H0")
    print("varience are same")
```

```
reject H0
varience are not same
```

In [62]:

```
f_stat, p_val = f_oneway(cloudy, mist, light_rain, heavy_rain)
print("p_val:", p_val)
```

```
p_val: 5.482069475935669e-42
```

In [63]:

```
if p_value < 0.05:
    print("Reject H0")
    print("Atleast one group have different mean")
else:
    print("Fail to reject H0")
    print("All groups have same mean")
```

```
Reject H0
Atleast one group have different mean
```

In [64]:

```
stat, p_value = kruskal(cloudy, mist, light_rain, heavy_rain)

print("test statistic:",stat)
print("p_value:",p_value)
```

```
test statistic: 205.00216514479087
p_value: 3.501611300708679e-44
```

In [65]:

```
if p_value < 0.05:
    print("Reject H0")
    print("Atleast one group have different median")
else:
    print("Fail to reject H0")
    print("All groups have same median")
```

```
Reject H0
Atleast one group have different median
```

Chi- Square

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

Setting up Hypothesis:

Null Hypothesis

H0: season and weather are independent of each other.

Alternative Hypothesis

Ha: Season and Weather are dependent variables.

Test:

Since both are categorical variables, we perform chi2-test(Test for Independence)

In [66]:
```
#Creation of Contingency table
obs =  pd.crosstab(yulu["weather_cat"], yulu["season_cat"])
obs
```

Out[66]:

season_cat / weather_cat	fall	spring	summer	winter
Heavy rain	0	1	0	0
cloudy	1930	1759	1801	1702
light rain	199	211	224	225
mist	604	715	708	807

In [67]:
```
#Importing chi2_contingency
from scipy.stats import chi2_contingency
```

Test Statistics

In [68]:
```
chi_stat, p_val, dof, expeted_val = chi2_contingency(obs)
print("p_value:",p_value)
```

```
p_value: 3.501611300708679e-44
```

Checking Significance Level

In [69]:
```
alpha = 0.05
if p_value < alpha:
     print("reject H0")
     print("Season and weather are not independent on each other")
else:
     print("Fail to reject H0")
     print("Season and weather are independent on each other")
```

```
reject H0
Season and weather are not independent on each other
```

Test Result

Since the p value of the chi2 test is less than the significance level, we reject H0. The two variables, season and weather are dependent on each other.

Recommendations

- Optimize Bike Distribution in Peak Months: Concentrate bike deployment efforts during peak months, such as June, July, and August, to meet increased demand. Allocate resources strategically to capitalize on favorable weather conditions and high user activity.
- Seasonal Marketing Strategies: Customize marketing efforts to align with seasonal trends. Develop targeted campaigns to promote Yulu's services more aggressively during summer months, leveraging the increased demand for outdoor activities.

- Enhance User Engagement in Off-Peak Months: Implement targeted promotional campaigns or discounts during off-peak months to encourage increased bike rentals and maintain consistent revenue flow.
- Weather-Responsive Pricing: Consider implementing dynamic pricing strategies that respond to weather conditions. Adjust rental rates during extreme weather days to optimize revenue and incentivize usage during favorable weather.
- Diversify Revenue Streams: Explore additional revenue streams, such as partnerships, sponsorships, or premium membership services, to diversify income sources and enhance profitability. Seek opportunities to monetize existing infrastructure and user base effectively.
- Invest in User Experience: Allocate resources to improve the overall user experience, including app features, bike maintenance, and customer support. Enhancements in technology and infrastructure will foster user loyalty and drive repeat business.
- Optimize Bike Deployment on Weekdays: Analyze rental patterns to optimize bike deployment strategies throughout the week. Ensure optimal resource allocation on both working and non-working days to meet varying user demand.
- Adapt Promotions to Weather Conditions: Modify promotions or discounts based on weather forecasts to drive user engagement. Offer special deals during inclement weather to encourage bike usage and maintain service visibility.
- Customize Marketing for Each Season: Customize marketing messages and promotions to resonate with users in different seasons. Highlight seasonal benefits and promotions to attract and retain users throughout the year
- Integrate Seasonal and Weather Plans: Develop integrated plans that consider both seasonal trends and weather forecasts. Optimize bike availability based on anticipated demand fluctuations due to changing weather conditions and seasonal factors.