yulu_solution

August 23, 2022

1 Business Case: Yulu - Hypothesis Testing

1.1 About 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.

1.2 Business problem

The company wants to know:

1. Which variables are significant in predicting the demand for shared electric cycles in the Indian market?
2. How well those variables describe the electric cycle demands

1.2.1 Column Profiling:

- datetime: datetime
- season: season (1: spring, 2: summer, 3: fall, 4: winter)
- holiday: whether day is a holiday or not (extracted from http://dchr.dc.gov/page/holiday-schedule)
- workingday: if day is neither weekend nor holiday is 1, otherwise is 0.
- weather: 1: Clear, Few clouds, partly cloudy, partly cloudy 2: Mist + Cloudy, Mist + Broken clouds, Mist + Few clouds, Mist 3: Light Snow, Light Rain + Thunderstorm + Scattered clouds, Light Rain + Scattered clouds 4: Heavy Rain + Ice Pallets + Thunderstorm + Mist, Snow + Fog
- temp: temperature in Celsius
- atemp: feeling temperature in Celsius
- humidity: humidity
- windspeed: wind speed

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

1.3 Concepts Used:

- Bi-Variate Analysis
- 2-sample t-test: testing for difference across populations
- ANNOVA
- Chi-square

2 Solution

[300]:
```
#common imports
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings

warnings.filterwarnings('ignore')
```

2.1 Read data

[195]:
```
data = pd.read_csv("data/bike_sharing.csv")

#we keep original data intact and create a copy 'df' for updates as necessary
df = data.copy()
```

[196]:
```
df.head()
```

[196]:
```
              datetime  season  holiday  workingday  weather  temp   atemp  \
0  2011-01-01 00:00:00       1        0           0        1  9.84  14.395
1  2011-01-01 01:00:00       1        0           0        1  9.02  13.635
2  2011-01-01 02:00:00       1        0           0        1  9.02  13.635
3  2011-01-01 03:00:00       1        0           0        1  9.84  14.395
4  2011-01-01 04:00:00       1        0           0        1  9.84  14.395

   humidity  windspeed  casual  registered  count
0        81        0.0       3          13     16
1        80        0.0       8          32     40
2        80        0.0       5          27     32
3        75        0.0       3          10     13
4        75        0.0       0           1      1
```

[197]:
```
df.shape
```

[197]: (10886, 12)

[198]:
```
df.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
```

[199]:
```
for col in df.columns:
    print(f'{col}: {df[col].nunique()}')
```

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

[200]:
```
for col in ['season', 'holiday', 'workingday', 'weather']:
    print(df[col].value_counts())
    print('\n')
```

```
4    2734
2    2733
3    2733
1    2686
```

```
Name: season, dtype: int64


0    10575
1      311
Name: holiday, dtype: int64


1    7412
0    3474
Name: workingday, dtype: int64


1    7192
2    2834
3     859
4       1
Name: weather, dtype: int64
```

Observations

1. The dataset has 10886 rows and 12 columns.
2. The dataset **does not have any missing (null) values.**
3. 'casual', 'registered', and 'count' are continuous variables. 'count' is the dependent variable for this analysis.
4. 'temp', 'atemp', 'humidity', and 'windspeed' are also continuous variables.
5. 'season' is a nominal categorical variable with levels 1, 2, 3, and 4. All levels are fairly evenly distributed.
6. 'weather' is a nominal categorical variable with levels 1, 2, 3, and 4. Levels are unevenly distributed.
7. 'holiday' is a dichotomous categorical variable with imbalanced data (very few rows with holiday = 1)
8. 'workingday' is also a dichotomous categorical variable.

Converting data types of columns

[201]:
```
#convert season, weather, hoiday, and workingday to cateogrical columns
for col in ['season', 'holiday', 'workingday', 'weather']:
    df[col] = df[col].astype('category')

#convert datetime to datetime format (if needed later)
df['datetime'] = pd.to_datetime(df['datetime'])
```

```
#extract hour component from datetime and add as a new column
#df['hour'] = df['datetime'].apply(lambda dt: dt.hour)

df.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  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  count       10886 non-null  int64
dtypes: category(4), datetime64[ns](1), float64(3), int64(4)
memory usage: 723.7 KB
```

2.2 Understanding nature of datetime column

The purpose of this case study is to apply several statistical tests some of which assume normality of dependent variable(count in this case). Before checking normality of count variable, it will be helpful to analyze what each 'row' in the dataset really represents. Based on the given columns, it appears that each row captures several parameters for a specific 'block of time' starting at value captured by'datetime' column. Ideally, each row should represent fixed length time block. We check this assumption in the code below (we sort datetime column and then take differences of each successive columns).

[202]:
```
#get sorted datetime column
sorted_df = df.sort_values(by='datetime')
sorted_dt = sorted_df['datetime']

#find difference between successive rows
print(pd.Series([(sorted_dt[i] - sorted_dt[i-1]) for i in range(1,sorted_dt.
 ↪size)]).value_counts())

print(f'\nDifference between largest and smallest datetimes =␣
 ↪{sorted_dt[len(sorted_dt)-1] - sorted_dt[0]}')
```

```
0 days 01:00:00     10820
0 days 02:00:00        36
```

```
12 days 01:00:00       13
11 days 01:00:00        8
0 days 03:00:00         5
0 days 13:00:00         1
9 days 01:00:00         1
10 days 01:00:00        1
dtype: int64

Difference between largest and smallest datetimes = 718 days 23:00:00
```

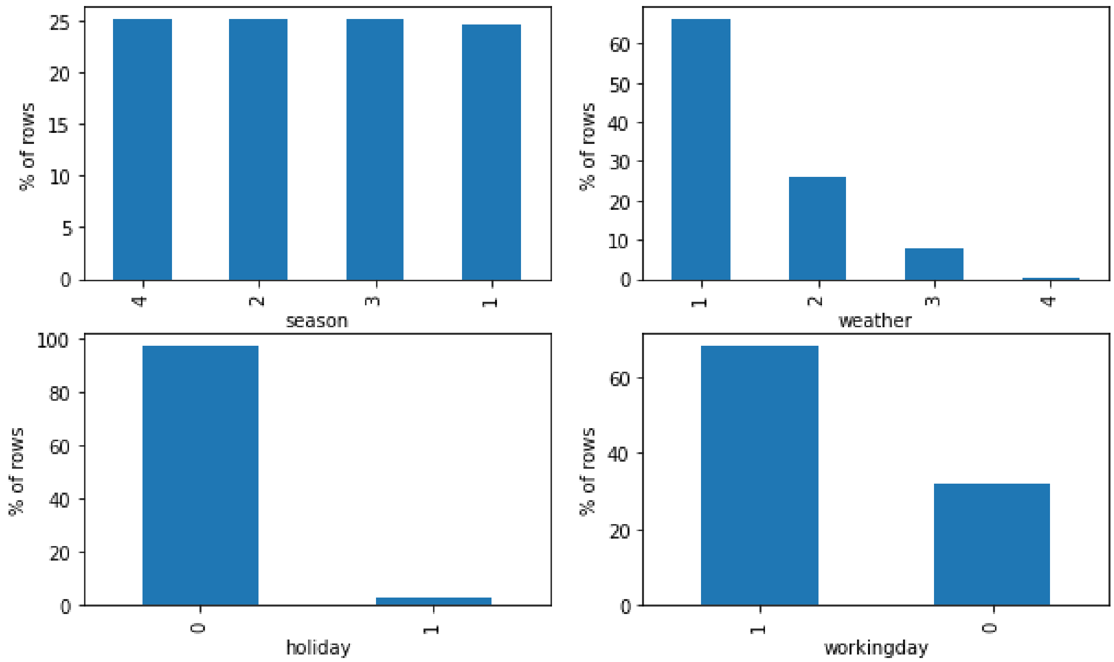
Observations

As we can see, for majority of the cases, difference between two successive rows is 1 hour. So it's safe to assume that each row in the given data represents various parameters (including rental count) for 1 hour block starting at the value stored in 'datetime' column. This means that we don't need to combine rows to form uniform length time blocks.

2.3 Univariate analysis

2.3.1 categorical variables

[203]:
```python
fig, ax = plt.subplots(2, 2, figsize=(10, 6))

#sns.countplot(x=df['season'], ax = ax[0][0])
#sns.countplot(x=df['weather'], ax = ax[0][1])
#sns.countplot(x=df['holiday'], ax = ax[1][0])
#sns.countplot(x=df['workingday'], ax = ax[1][1])

df['season'].value_counts(normalize=True).mul(100).plot(kind='bar',␣
 ↪xlabel='season', ylabel='% of rows' ,ax = ax[0][0])
df['weather'].value_counts(normalize=True).mul(100).plot(kind='bar',␣
 ↪xlabel='weather', ylabel='% of rows', ax = ax[0][1])
df['holiday'].value_counts(normalize=True).mul(100).plot(kind='bar',␣
 ↪xlabel='holiday', ylabel='% of rows', ax = ax[1][0])
df['workingday'].value_counts(normalize=True).mul(100).plot(kind='bar',␣
 ↪xlabel='workingday', ylabel='% of rows', ax = ax[1][1])

plt.show()
```



Observations

1. The dataset contains fairly even number of data for each seasons.
2. The dataset contains ~65% rows for weather-1 (*Clear, Few clouds, partly cloudy*), ~25% rows for weather-2 (*Mist + Cloudy, Mist + Broken clouds, Mist + Few clouds, Mist*), ~9% rows for weather-3 (*Light Snow, Light Rain + Thunderstorm + Scattered clouds, Light Rain + Scattered clouds*). There is just one row for weather-4 (*Heavy Rain + Ice Pallets + Thunderstorm + Mist, Snow + Fog*).
3. The dataset contains ~97% rows for non-holidays. This shows imbalanced data.
4. The dataset contains ~68% rows for workingdays, remaining for non-working days.

NOTE - This analysis only focuses on 'number of rows'. We will analyze actual 'rental count' against these variables in the bi-variate analysis section.

2.3.2 continuous variables

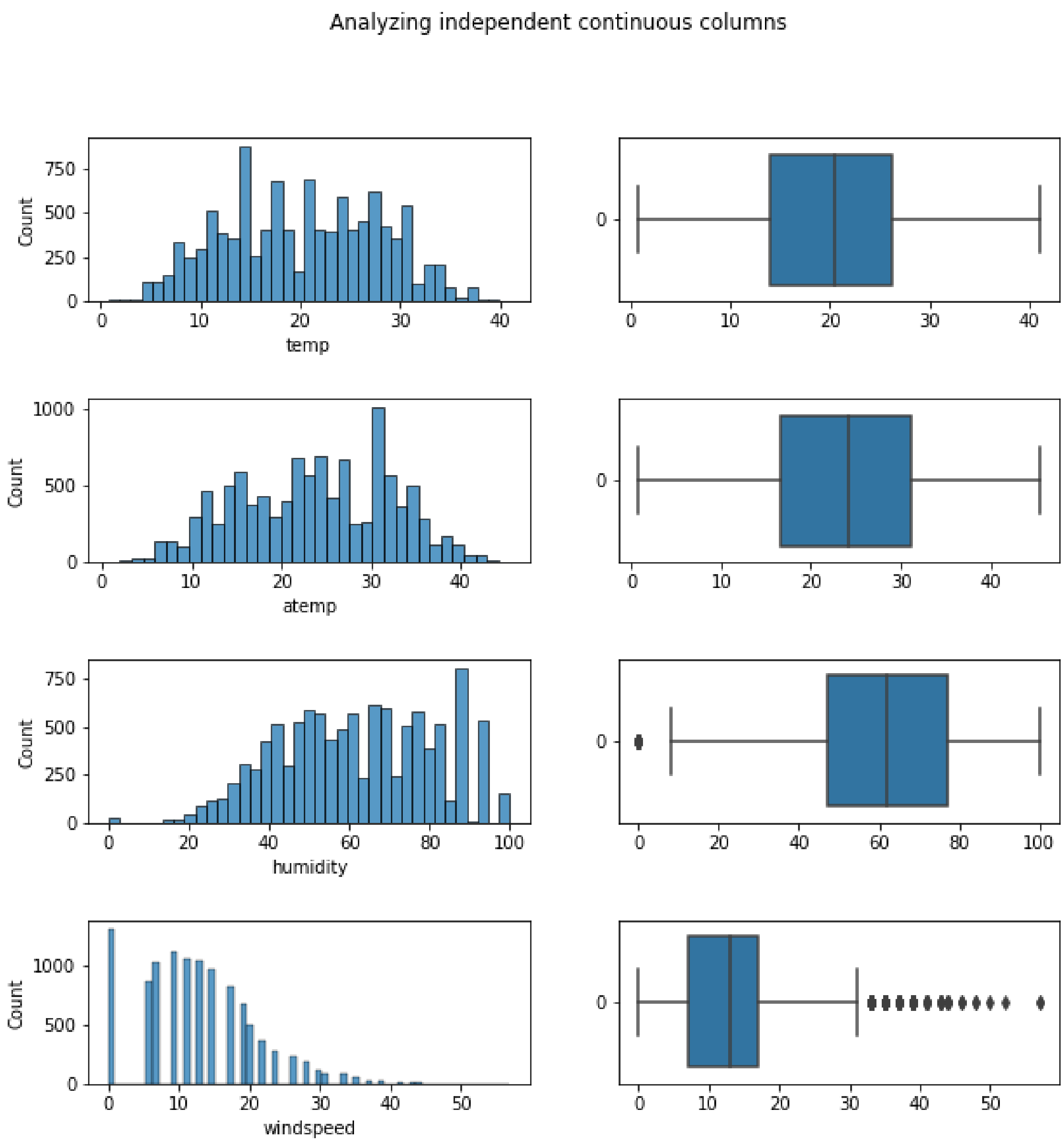
[204]:
```
#check statistical parameters for various countnuous variables.
cols_cont = [col for col in df.columns if (col not in ['hour']) and (df[col].
↪dtype.kind in 'biufc')]
df[cols_cont].describe()
```

[204]:
```
              temp         atemp      humidity     windspeed        casual  \
count  10886.00000  10886.000000  10886.000000  10886.000000  10886.000000
mean      20.23086     23.655084     61.886460     12.799395     36.021955
std        7.79159      8.474601     19.245033      8.164537     49.960477
```

```
min        0.82000    0.760000    0.000000    0.000000    0.000000
25%       13.94000   16.665000   47.000000    7.001500    4.000000
50%       20.50000   24.240000   62.000000   12.998000   17.000000
75%       26.24000   31.060000   77.000000   16.997900   49.000000
max       41.00000   45.455000  100.000000   56.996900  367.000000

         registered         count
count  10886.000000  10886.000000
mean     155.552177    191.574132
std      151.039033    181.144454
min        0.000000      1.000000
25%       36.000000     42.000000
50%      118.000000    145.000000
75%      222.000000    284.000000
max      886.000000    977.000000
```

[205]:

```
#Analyze indepedent continuous columns
cols = ['temp', 'atemp', 'humidity', 'windspeed']

fig, ax = plt.subplots(len(cols), 2, figsize=(10, 10))
fig.suptitle('Analyzing independent continuous columns')
for i in range(len(cols)):
    col = cols[i]
    sns.histplot(data=sorted_df[col], ax=ax[i][0])
    sns.boxplot(data=sorted_df[col], orient="horizontal", ax=ax[i][1])
plt.subplots_adjust(hspace=0.6)
plt.show()
```



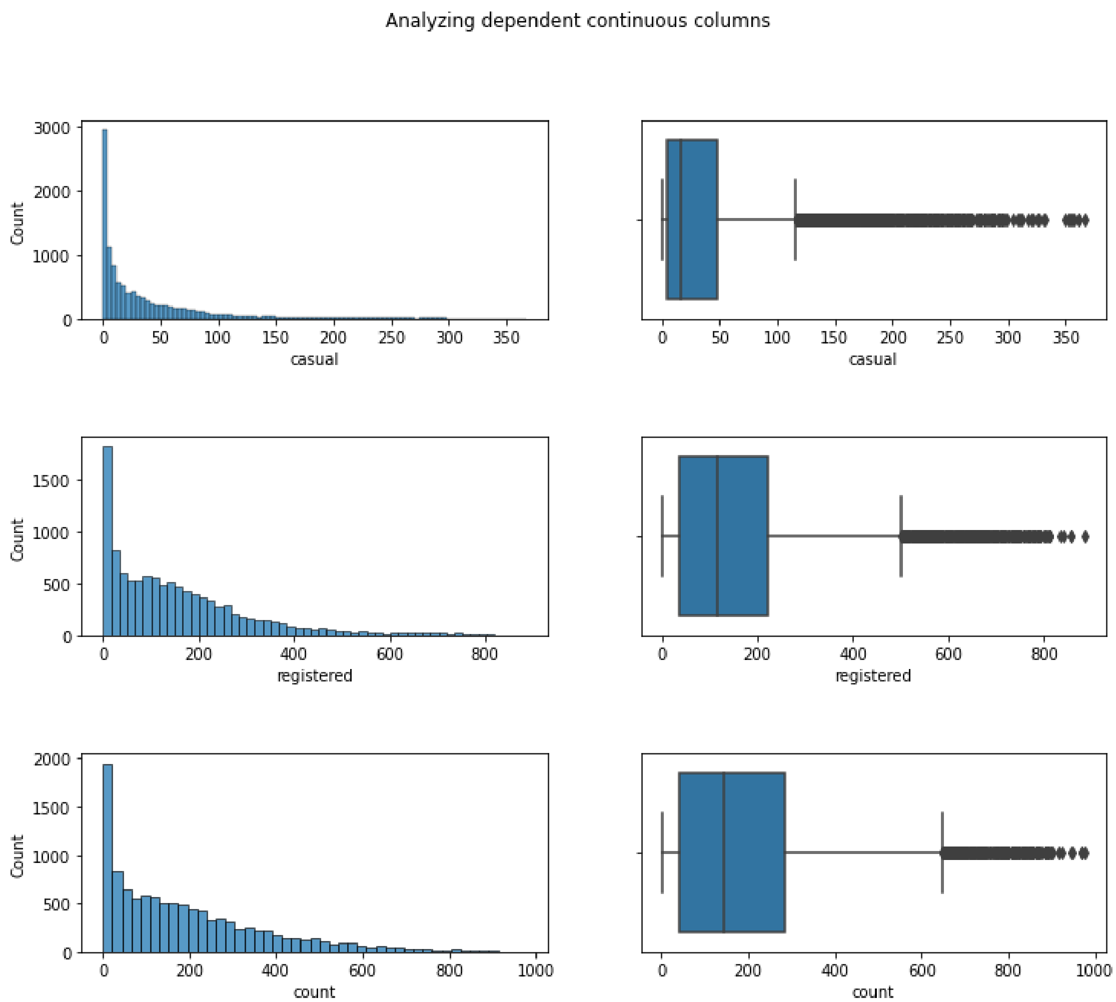
Observations

We observe that 'temp' and to a lesser extent 'atemp' look reasonably symmetric and bell shaped. 'humidity' looks left skewed (with some outliers near zero). 'windspeed', on the other hand, looks right skewed (with several outliers beyond 30).

However, for this case-study, we may not need these indepedent variables, hence, we skip further analysis.

[206]:
```
#Analyze dependent columns

cols = ['casual', 'registered', 'count']
```

```python
fig, ax = plt.subplots(len(cols), 2, figsize=(12, 10))
fig.suptitle('Analyzing dependent continuous columns')
for i in range(len(cols)):
    col = cols[i]
    sns.histplot(data=sorted_df[col], ax=ax[i][0])
    sns.boxplot(x=sorted_df[col], orient="horizontal", ax=ax[i][1])
plt.subplots_adjust(hspace=0.6)
plt.show()

print(f'skew of "casual": {sorted_df["casual"].skew()}')
print(f'skew of "registered": {sorted_df["registered"].skew()}')
print(f'skew of "count": {sorted_df["count"].skew()}')
```




```
skew of "casual": 2.4957483979812567
skew of "registered": 1.5248045868182296
```

```
skew of "count": 1.2420662117180776
```

Observations

1. All three variables - 'casual', 'registered', and 'count' are positively skewed.
2. All three variables have high number of outliers towards their right tail.
3. IQR range for 'casual' is (4, 49) with median value 17 and mean 36. IQR range for 'registered' is (36, 222) with median value 118 and mean 155. IQR range for 'count' is (42, 284) with median value 145 and mean 191.

   Since 'count' is sum of 'casual' and 'registered', we note that compared to 'casual' users, 'registered' users constitute larger portion of total 'count' users.
4. In later sections, we will run several statistical tests with 'count' as the dependent variable. These tests make assumption of normality. To address this, in the section on transformation, we will attempt to transform the 'count' variable so that it can better match normal curve.

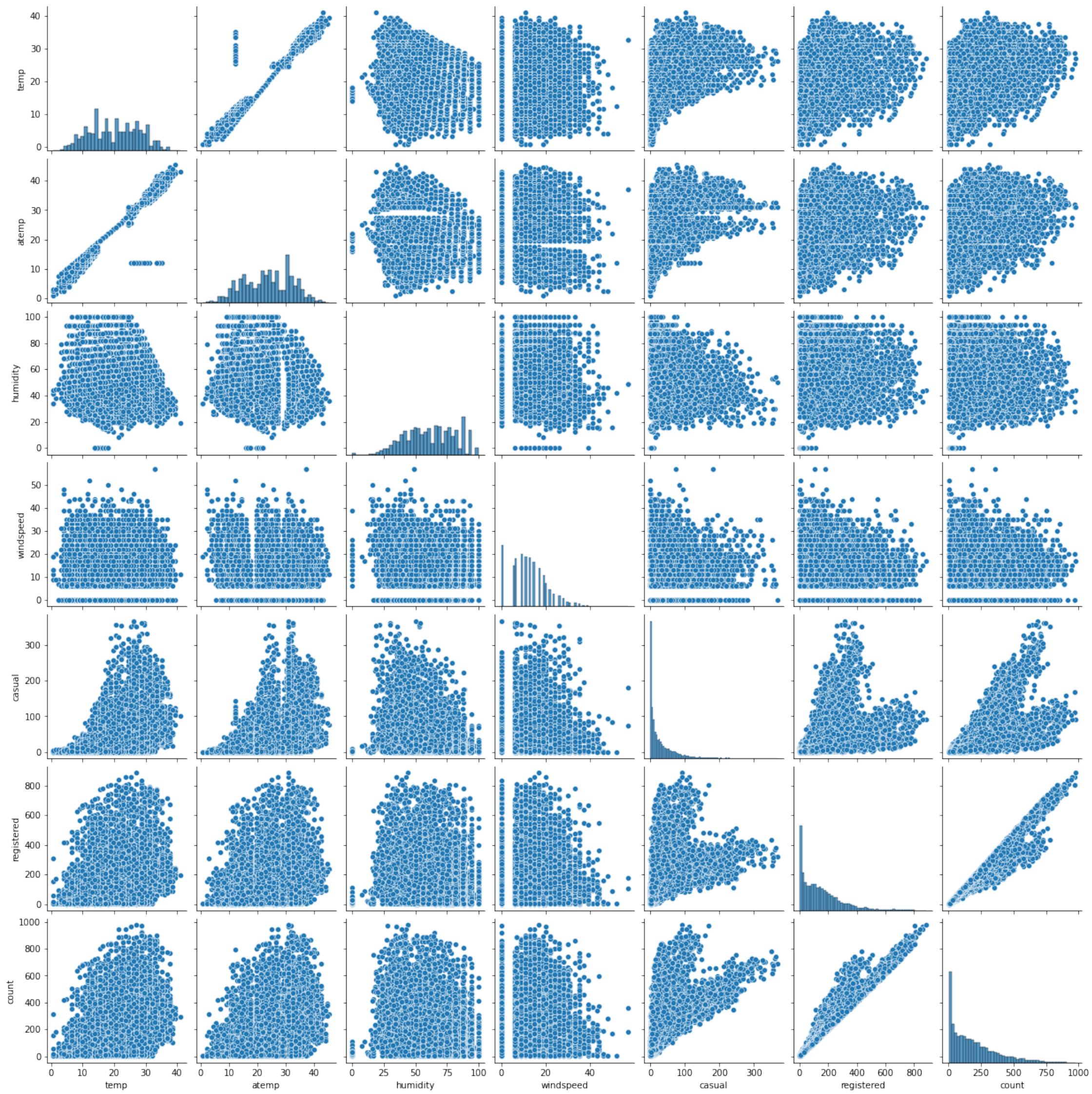
2.4 Missing value and outlier treatment

There are no missing values in the given dataset. However, distributions of 'count', 'casual', and 'registered' variables are positively skewed with large number of outliers (on the right tail especially). Since these outliers seem to represent genuine data, we cannot remove them. One possible solution is to apply transformations such as log, square root, or cube root which may help reduce the number of outliers. However, in this casestudy, the main focus is on conducting statistical tests which usually make an assumption of normally distributed data. So we will not treat outliers further in this case study.

2.5 Bivariate Analysis

2.5.1 pair plots and correlation (for continuous variables)

[207]:
```
#pair plot
sns.pairplot(df)
plt.show()
```

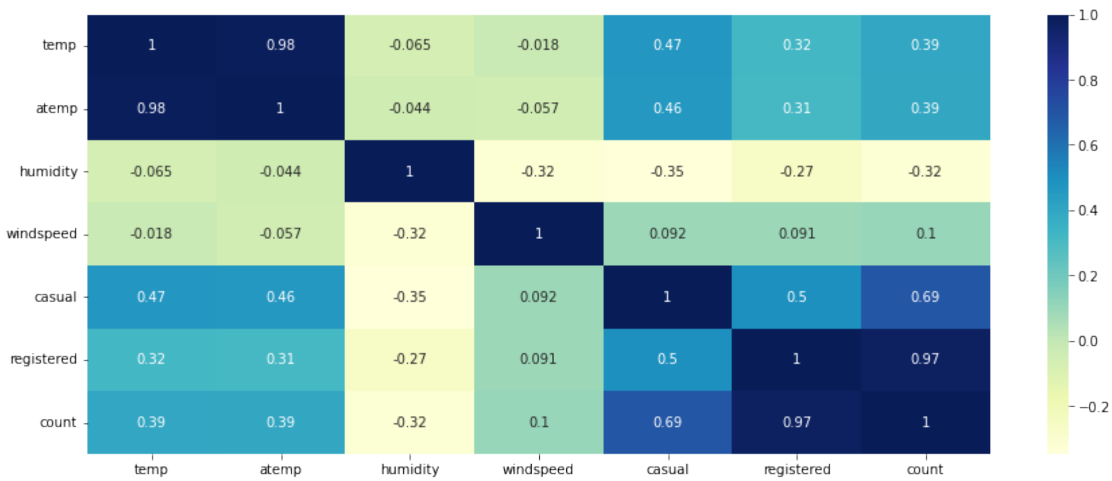



```
[216]: corr_df = df.corr(method='pearson')
       corr_df
```

```
[216]:                temp     atemp  humidity  windspeed    casual  registered  \
       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
       registered  0.318571  0.314635 -0.265458   0.091052  0.497250    1.000000
       count       0.394454  0.389784 -0.317371   0.101369  0.690414    0.970948

                      count
```

```
temp          0.394454
atemp         0.389784
humidity     -0.317371
windspeed     0.101369
casual        0.690414
registered    0.970948
count         1.000000
```

[217]:
```python
plt.figure(figsize=(15,6))
sns.heatmap(corr_df, cmap="YlGnBu", annot=True)
plt.show()
```



Observations

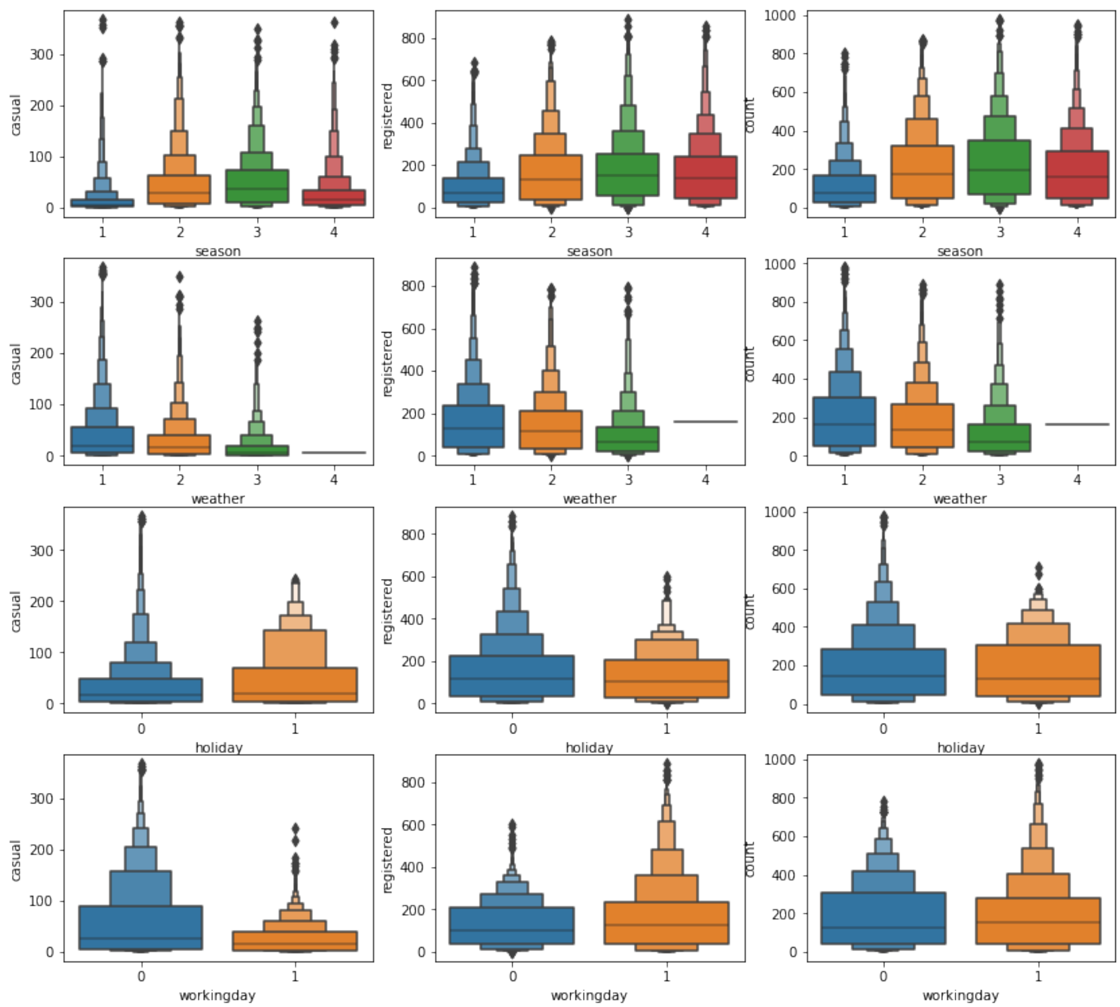
1. There is a very high positive correlation between 'registered' and 'count' users. There is also a high positive correlation (but lesser degree) between 'casual' and 'count' columns. This is expected as 'count' is sum of both 'casual' and 'registered' users with 'registered' user contributing the major portion while 'casual' users being smaller portion of the total count.
2. There is a moderate positive correlation between 'casual' and 'registered' users (both of which are indepedent variables). This shows that there are common factors influencing both 'casual' and 'registered' users to some extent.
3. There is a very high positive correlation between 'temp' and 'atemp' columns. This is understandable given their definitions.
4. We see low(weak) positive correlation between 'temp' and 'count', 'temp' and 'registered', and 'temp' and 'casual'. Relatively comparing, there's higher correlation between 'temp' and 'casual' users than 'temp' and 'registered' users. This potentially shows that 'casual' users are somewhat more responsive to increasing 'temp' in purchasing rental bikes compared to the 'registered' users. This can be also visually seen in the pair plot.
5. Similarly, there is low(weak) negative correlation between 'humidity' and 'count', 'humidity'

and 'registered', and 'humidity' and 'casual' users. It appears that compared to 'registered' users, 'casual' users respond to increasing 'humidty' somewhat more (negatively).

6. 'windspeed' has very negligible correlation with 'count'

2.5.2 Distribution plots for rental counts for various categorical variables

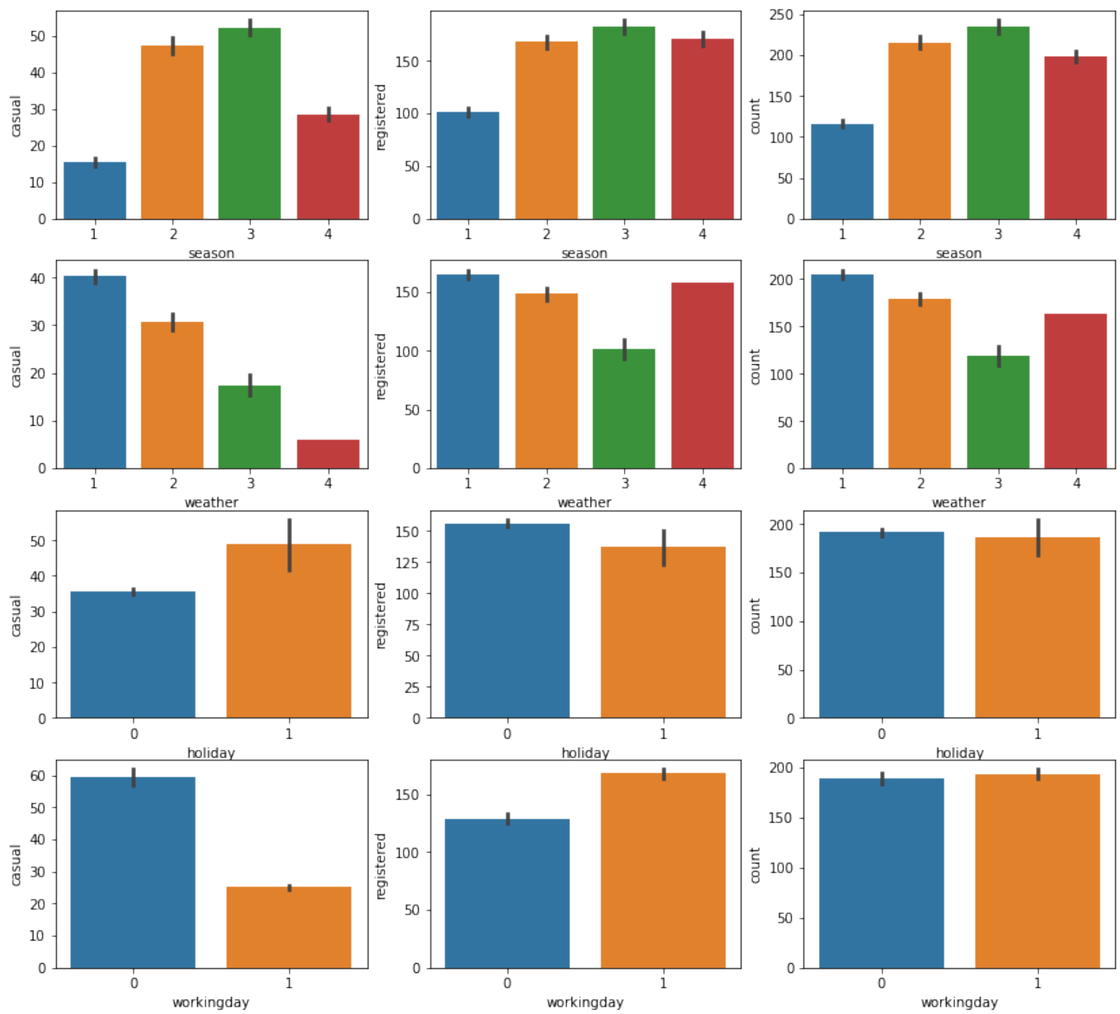
[239]:
```
fig, ax = plt.subplots(4, 3, figsize=(14, 13))

sns.boxenplot(x='season', y='casual', data=df, ax=ax[0][0])
sns.boxenplot(x='season', y='registered', data=df, ax=ax[0][1])
sns.boxenplot(x='season', y='count', data=df, ax=ax[0][2])

sns.boxenplot(x='weather', y='casual', data=df, ax=ax[1][0])
sns.boxenplot(x='weather', y='registered', data=df, ax=ax[1][1])
sns.boxenplot(x='weather', y='count', data=df, ax=ax[1][2])

sns.boxenplot(x='holiday', y='casual', data=df, ax=ax[2][0])
sns.boxenplot(x='holiday', y='registered', data=df, ax=ax[2][1])
sns.boxenplot(x='holiday', y='count', data=df, ax=ax[2][2])

sns.boxenplot(x='workingday', y='casual', data=df, ax=ax[3][0])
sns.boxenplot(x='workingday', y='registered', data=df, ax=ax[3][1])
sns.boxenplot(x='workingday', y='count', data=df, ax=ax[3][2])

plt.show()
```



2.5.3 mean for rental counts for each categorical variables

[270]:
```
fig, ax = plt.subplots(4, 3, figsize=(14, 13))

sns.barplot(x='season', y='casual',ci=95, data=df, ax=ax[0][0])
sns.barplot(x='season', y='registered', ci=95, data=df, ax=ax[0][1])
sns.barplot(x='season', y='count', ci=95, data=df, ax=ax[0][2])

sns.barplot(x='weather', y='casual', ci=95, data=df, ax=ax[1][0])
sns.barplot(x='weather', y='registered', ci=95, data=df, ax=ax[1][1])
sns.barplot(x='weather', y='count', ci=95, data=df, ax=ax[1][2])

sns.barplot(x='holiday', y='casual', ci=95, data=df, ax=ax[2][0])
sns.barplot(x='holiday', y='registered', ci=95, data=df, ax=ax[2][1])
sns.barplot(x='holiday', y='count', ci=95, data=df, ax=ax[2][2])
```

```
sns.barplot(x='workingday', y='casual', ci=95, data=df, ax=ax[3][0])
sns.barplot(x='workingday', y='registered', ci=95, data=df, ax=ax[3][1])
sns.barplot(x='workingday', y='count', ci=95, data=df, ax=ax[3][2])

plt.show()
```



Observations

1. total rental demands are highest in season 3 and 2, somewhat less in season 4, and lowest in season 1. Specifically in season 4, demand from 'casual' user goes down considerably.
2. total demand is highest in weather 1,followed by weather 2. Weather 4 has somewhat lesser demand while weather 3 has lowest demands. Specifically, demands for casual users go down considerably in weather 4.
3. There is more demand from 'casual' users on 'holiday' than on 'non-holidays'. On the other hand, demands from'registered' users on 'holiday' is slightly less than on 'nonholidays'.The total average demand, however, is comparable on 'holidays' and 'non-holidays'.

4. The average demand from 'casual' users on 'nonworking' day is more than double than that on 'workingday'. On the other hand, average demand from 'registered' users is more on 'workingday' than on 'nonworkingday'. The total average demand, however, is comparable on 'workday' and 'non-working days'.

Recommendations - Covered in sections for specific statistical tests.

2.6 Statistical tests

In this section, we attempt to answer the following questions using appropriate statistical tests.

1. Check if Working Day has an effect on the number of electric cycles rented (2- Sample T-Test)
2. Check if No. of cycles rented is similar or different in different weather (one way ANOVA)
3. Check if No. of cycles rented is similar or different in different season (one way ANOVA)
4. Check if Weather is dependent on the season (Chi-square test)

2.7 Helper functions

function to test Normality

[310]:
```python
import scipy.stats as stats
from statsmodels.graphics.gofplots import qqplot

#helper function to perform normality test
#for each dataset, it plots its histogram, boxplot, and QQ plot
#it also prints Shapiro-Wilk metrics
#in addition, additional transformation functions (such as log, sqrt etc) can␣
↪be supplied in t_arr
def testnorm(data, title, t_arr = []):
    arr = [('', lambda x: x)] if (t_arr == None or len(t_arr) == 0) else t_arr
    cnt = len(arr)

    fig = plt.figure(figsize=(15, cnt*3.5))
    subfig = fig.subfigures(nrows=cnt, ncols=1)
    res = [] #to hold shapiro-wilk results

    for i in range(cnt):
        item = arr[i]
        text = title + ' ' + item[0]
        fn = item[1]
        tr_data = pd.Series([fn(ele) for ele in data])

        figref = subfig[i] if (cnt > 1) else subfig

        figref.suptitle(text)
        ax = figref.subplots(nrows=1, ncols=3)
        sns.histplot(tr_data, kde=True, ax=ax[0])
        sns.boxplot(x=tr_data, ax=ax[1])
```

```
        qqplot(tr_data, line='s', ax = ax[2])

        res.append(stats.shapiro(tr_data))

    plt.show()

    print('\nShapiro-Wilk Test metrics')
    for i in range(cnt):
        print(f'{title} {arr[i][0]} : {res[i]}')
```

2.8 Test 1: Does Working Day have an effect on the number of cycles rented?

'workingday' is a categorical variable with two levels- **1(working)** and **0(non working)**. We can use **Two-sample independent t-test** to check if 'workingday' is a significant factor in predicting cycle demand. Before we apply two-sample t-test, we need to check for the following assumptions.

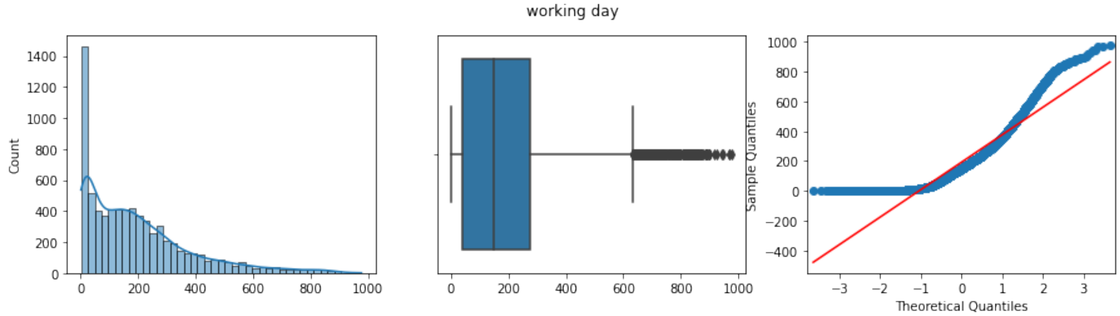
2.8.1 Two-sample t-test assumptions

1. Data values must be independent. Measurements for one observation do not affect measurements for any other observation. (**our data satisfy this condition**)
2. Data in each group must be obtained via a random sample from the population. (**our data satisfy this condition**)
3. Data values are continuous. (**our data satisfy this condition**)
4. Data in each group are normally distributed. (**Needs further investigation**)
5. The variances for the two independent groups are equal. (**Needs further investigation**)

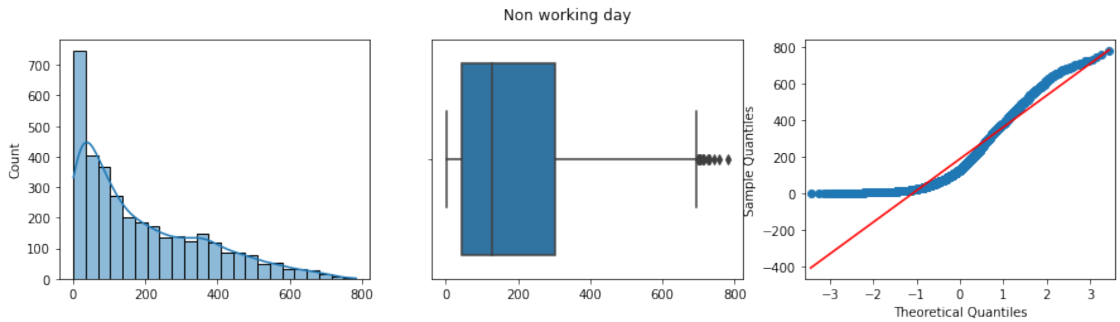
2.8.2 Check for normality

We need to check if both samples (workingday=0 and workingday=1) are normally distributed. We rely on distribution plot, qq-plot, and metric from Shapiro-welk test to determine normality.

[340]:
```
df_wd = df[df['workingday'] == 1]
df_nwd = df[df['workingday'] == 0]
sample_wd = df_wd['count']
sample_nwd = df_nwd['count']

testnorm(sample_wd, 'working day')
testnorm(sample_nwd, 'Non working day')
```



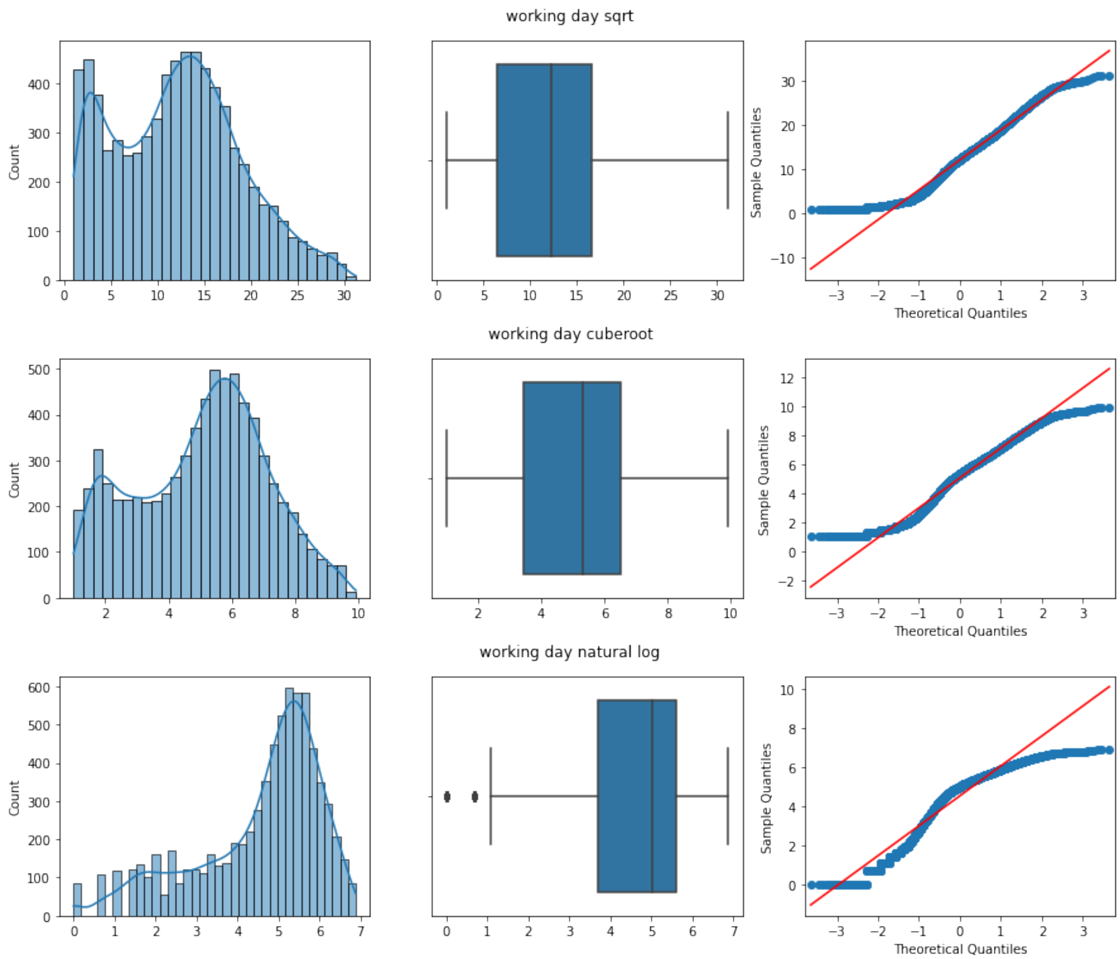

```
Shapiro-Wilk Test metrics
working day  : ShapiroResult(statistic=0.8702576160430908, pvalue=0.0)
```



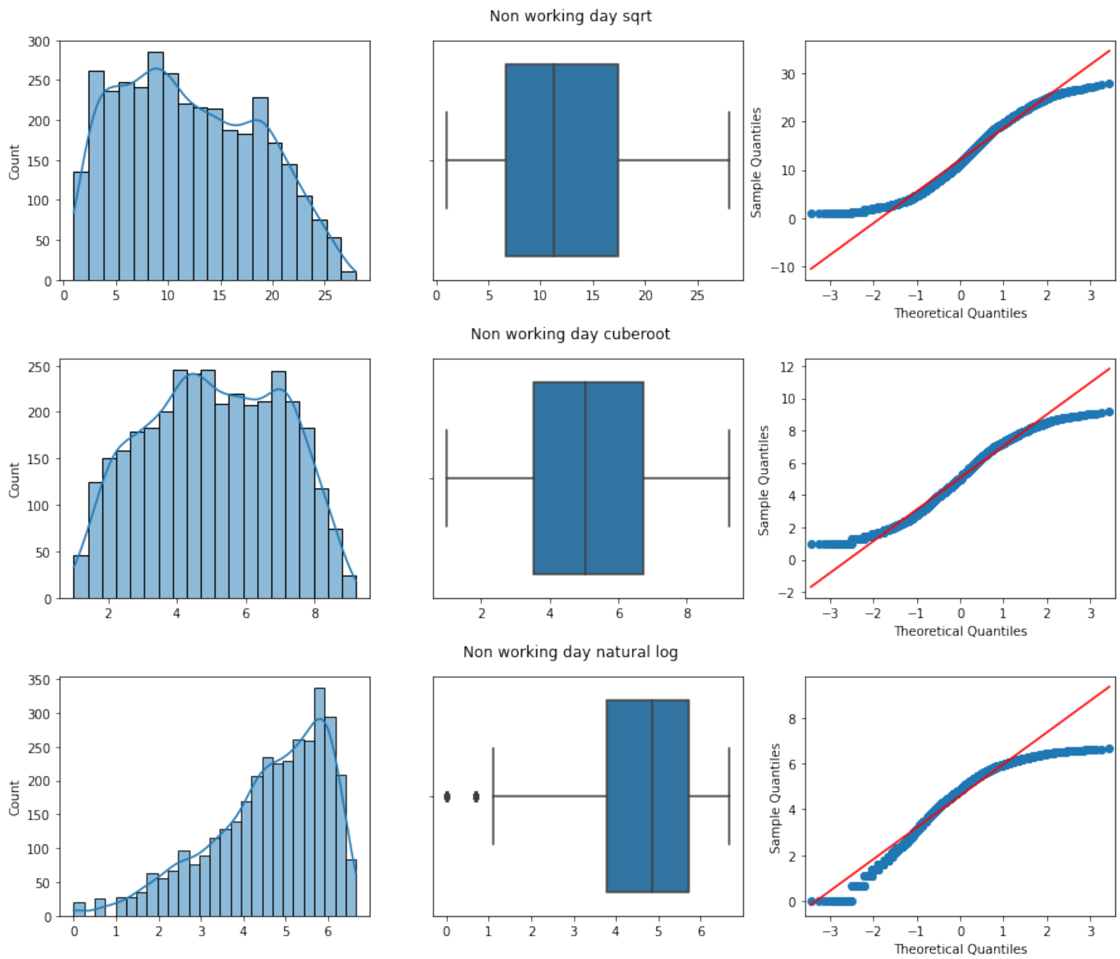

```
Shapiro-Wilk Test metrics
Non working day  : ShapiroResult(statistic=0.8852126598358154,
pvalue=4.203895392974451e-45)
```

From the QQ-plot of both the samples, it's evident that none of the samples follow normal distribution. The distribution is right skewed with a lot of outliers. P-value from Shapiro test for both the samples are very small, which further strenghens the hypothesis that data values are not normally distributed. Next, we try to apply 'square root', 'cube root', and 'log' transformations on the data to see if that can make data closer to normal distribution.

[341]:
```
testnorm(sample_wd, 'working day', [('sqrt', lambda x: x**0.5), ('cuberoot',␣
↪lambda x: x**(1/3)), ('natural log', lambda x:np.log(x))])
testnorm(sample_nwd, 'Non working day', [('sqrt', lambda x: x**0.5),␣
↪('cuberoot', lambda x: x**(1/3)), ('natural log', lambda x:np.log(x))])
```




```
Shapiro-Wilk Test metrics
working day sqrt : ShapiroResult(statistic=0.9736786484718323,
pvalue=8.141092182514257e-35)
working day cuberoot : ShapiroResult(statistic=0.9765961170196533,
pvalue=3.6330350032131795e-33)
working day natural log : ShapiroResult(statistic=0.9049559235572815,
pvalue=0.0)
```




```
Shapiro-Wilk Test metrics
Non working day sqrt : ShapiroResult(statistic=0.9658713936805725,
pvalue=6.166544088743618e-28)
Non working day cuberoot : ShapiroResult(statistic=0.9759071469306946,
pvalue=8.362062857140234e-24)
Non working day natural log : ShapiroResult(statistic=0.9339465498924255,
pvalue=9.749252684808278e-37)
```

Observation

Clearly, **'count' for 'workingday' and 'nonworkingday' samples are not normally distributed. So the normality assumption doesn't hold**. Also, none of the square root, cube root, and log transformations are effective in transforming the data into normal distribution (confirmed by qq-plot and shapiro-welk test metrics). So we do not see a need to apply any transformation.

2.9 check variance homogeneity

[321]:
```
var_wd = np.var(sample_wd, ddof=1)
var_nwd = np.var(sample_nwd, ddof=1)

print(f'variance of workingday sample: {var_wd}, variance of nonworkingday␣
↪sample: {var_nwd}')

#levene's test
print("\nLevene's test to check if population variances are equal")
print('H0: population variances are equal')
print('H1: population variances are not equal')
print(f"Levene's test metric: {stats.levene(sample_wd, sample_nwd)}")
```

```
variance of workingday sample: 34045.29037312209, variance of nonworkingday
sample: 30180.03350064094

Levene's test to check if population variances are equal
H0: population variances are equal
H1: population variances are not equal
Levene's test metric: LeveneResult(statistic=0.004972848886504472,
pvalue=0.9437823280916695)
```

Observation

P-value from Levene's test is very high, and hence we fail to reject the null hypothesis. So **the assumption of variance homogeniety holds.**

2.10 Two-sample indepedent t-test

All the assumptions except for the normality assumption hold. For this casestudy, we go ahead with the test. Since there is no significant difference between the population variances (as confirmed above), we can use **pooled variance**. We use **two tailed t-test** and set **significance level (alpha) at 0.05** (0.25 on both tails)

H0: The two population means are equal. In other words, rental counts for workingday and nonworkingday populations are equal.

H1: The two population means are not equal.

[338]:
```
#helper function for two-sample indepedent t-test with equal population␣
↪variances
def twosample_ind_ttest_equal_var(sample1, sample2, alpha=0.05, diaginfo=True):
    n1 = sample1.size
    n2 = sample2.size

    mean1 = np.mean(sample1)
    mean2 = np.mean(sample2)

    var1 = np.var(sample1, ddof=1)
```

```python
    var2 = np.var(sample2, ddof=1)
    var_pooled = ((n1-1)*var1 + (n2-1)*var2)/(n1+n2-2)
    std_pooled = var_pooled**0.5

    std_err_diff = std_pooled * (1/n1 + 1/n2)**0.5

    t_stat = (mean1 - mean2)/std_err_diff
    v = n1+n2-2 #dof
    cv = stats.t.ppf(1-alpha/2, v) #critical value corresponding to the given␣
 ↪significance level in t-dist with given dof.

    if(diaginfo):
        print(f'sample1: mean={mean1}, var={var1}')
        print(f'sample2: mean={mean2}, var={var2}')

        print(f'\npooled var = {var_pooled}, pooled standard deviation =␣
 ↪{std_pooled}')
        print(f'standard error of difference of sample means = {std_err_diff} ')

        print(f'\nt-stat={t_stat}, dof={v}')
        print(f'Critical values for two-tailed test at significance level␣
 ↪{alpha} and dof {v} are : +{cv} and -{cv}')
        print(f'is abs(t_stat) < cv? : {np.abs(t_stat) < cv}')

    return (t_stat, cv, v)

twosample_ind_ttest_equal_var(sample_wd, sample_nwd, 0.05)

print(f'\nResults from scipy.ttest_ind function: {stats.ttest_ind(sample_wd,␣
 ↪sample_nwd, equal_var=True)}')
```

```
sample1: mean=193.01187263896384, var=34045.29037312209
sample2: mean=188.50662061024755, var=30180.03350064094

pooled var = 32811.916878255586, pooled standard deviation = 181.14059975128598
standard error of difference of sample means = 3.724494642992504

t-stat=1.2096277376026694, dof=10884
Critical values for two-tailed test at significance level 0.05 and dof 10884 are
: +1.9601819678713073 and -1.9601819678713073
is abs(t_stat) < cv? : True

Results from scipy.ttest_ind function:
Ttest_indResult(statistic=1.2096277376026694, pvalue=0.22644804226361348)
```

Result

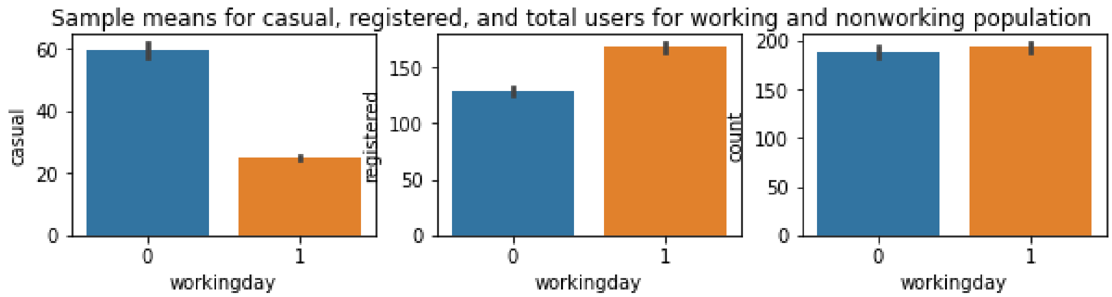
Since the calculated t-stat value (~1.21) is less than right tail critical value (~1.96) at 0.05 significance level and dof=10884 (in other words, t-stat doesn't fall in critical region), we fail to reject the null hypothesis. Thus means of the two populations (workingday and nonworkingday) are to be considered equal. **Thus, with 95% confidence, we can state that workingday variable is not significant in predicting 'total' cycle rental demands.**

2.10.1 Additional Two-sample t-tests with 'casual' and 'registered' users as the depedent variable

[356]:
```
print('\nTwo-sample t test for casual users')

sample_wd = df_wd['casual']
ample_nwd = df_nwd['casual']

l_stat, l_pval = stats.levene(sample_wd, sample_nwd)
print(f"Levene's metric for variance homogeniety: statistic={l_stat},␣
↪pval={l_pval}")
print(f'Results from scipy.ttest_ind function: {stats.ttest_ind(sample_wd,␣
↪sample_nwd, equal_var=(l_pval >= 0.05))}')

print('\nTwo-sample t test for registered users')

sample_wd = df_wd['registered']
ample_nwd = df_nwd['registered']

l_stat, l_pval = stats.levene(sample_wd, sample_nwd)
print(f"Levene's metric for variance homogeniety: statistic={l_stat},␣
↪pval={l_pval}")
print(f'Results from scipy.ttest_ind function: {stats.ttest_ind(sample_wd,␣
↪sample_nwd, equal_var=(l_pval >= 0.05))} \n')

fig, ax = plt.subplots(1, 3, figsize=(10, 2))
fig.suptitle('Sample means for casual, registered, and total users for working␣
↪and nonworking population')
sns.barplot(x='workingday', y='casual', ci=95, data=df, ax=ax[0])
sns.barplot(x='workingday', y='registered', ci=95, data=df, ax=ax[1])
sns.barplot(x='workingday', y='count', ci=95, data=df, ax=ax[2])
plt.show()
```

```
Two-sample t test for casual users
Levene's metric for variance homogeniety: statistic=6394.0670238657385, pval=0.0
Results from scipy.ttest_ind function:
Ttest_indResult(statistic=-55.088757235572864, pvalue=0.0)

Two-sample t test for registered users
Levene's metric for variance homogeniety: statistic=50.130801041681806,
pval=1.526989550784087e-12
```

```
Results from scipy.ttest_ind function:
Ttest_indResult(statistic=-5.851397746409238, pvalue=5.111277838927982e-09)
```



Observations

1. For two-sample indepedent t-test for 'casual' users, pval is very low and hence we reject the H0. In other words, **the difference in 'casual' users mean value between 'workingday' and 'nonworkingday' samples are significant (at 95% confidence level)**. This can also be visualized in the bi-viriate plot shown above.
2. Similarly, for two-sample indepedent t-test for 'registered' users, pval is very low and hence we reject the H0. In other words, **the difference in 'registered' users mean value between 'workingday' and 'nonworkingday' samples are significant (at 95% confidence level)**. This can also be visualized in the bi-viriate plot shown above.
3. However, the effect of workingday on 'casual' and 'registered' users is opposite. While there are more 'casual' users on 'non-working' days compared to 'working' days, there are more 'registered' users on 'working' days compared to 'non working' days. Since 'count' is addition of 'casual' and 'registered' users, this opposite effects cancel out and we do not see significant impact of 'workingday' variable on overall 'count'.

Recommendations

At present, overall rental demands remain the same on both working and non-working days. However, relative demands from 'registered' users is higher on 'working' days than on 'non-working' days. Similarly, relative demands from 'casual' users is higher on 'nonworking' days than on 'working' days. There might be a potential opportunity to attract more 'casual' users during the 'workday', but outside of working hours (say early morning, late evening). Some of the possible options include offering dicounted rental prices and extended times for 'casual' users on workdays.

2.11 Test 2: Check if No. of cycles rented is similar or different in different weather?

'weather' is a categorical variable with 4 levels. We need to compare rental count means for each weather level. We can use one-way ANOVA test for this purpose (one factor with four levels).

H0: All population means are equal. That is, cycle rental count is similar across different weather types.

H1: Not all population means are equal. In other words, cycle rental count is different for atleast one weather type.

2.11.1 One-way ANOVA assumptions

For Anova results to be valid, the following assumptions need to hold.

1. Responses for a given group are independent - **We assume this as true**.
2. Variances of populations are equal. **(needs to be checked)**
3. Response variable residuals are normally distributed (or approximately normally distributed) - **We will check this post ANOVA test.**

2.12 check variance homogeneity

[599]:
```
factor = 'weather'
resp = 'count'
levels = df.groupby(factor)[resp].groups.keys()

#generate groups for each level of weather variable.
groups = []
for level in levels:
    group = df[df[factor] == level][resp]
    print(f'variance for group {level}: {np.var(group, ddof=1)}')
    groups.append(group)

#levene's test
print("\nLevene's test to check if population variances are equal")
print('H0: All four population variances are equal')
print('H1: Not all four population variances are equal')
print(f"Levene's test metric: {stats.levene(*groups)}")
```

```
variance for group 1: 35328.79846268022
variance for group 2: 28347.248993301797
variance for group 3: 19204.77589271419
variance for group 4: nan

Levene's test to check if population variances are equal
H0: All four population variances are equal
H1: Not all four population variances are equal
Levene's test metric: LeveneResult(statistic=54.85106195954556,
pvalue=3.504937946833238e-35)
```

Note

Ideally, for ANOVA results to be considered valid, all population variances (that is variance for each group) should be the same. We see high variations among different groups. Levene's test confirms this by returning a very small p-value. We thus reject the null hypothesis of variance homogeneity. For this case-study, we will continue with ANOVA test.

2.13 One way ANOVA test (count ~ weather)

[572]:
```
#Define helper function to compute one way ANOVA statistics
#takes as input a list of groups (corresponding to each level of the factor)
#returns ANOVA table

def anova_oneway(groups, alpha=0.05):
    #define Sum of Squares variables
    SSt = 0
    SSw = 0
    SSb = 0

    #define other variables
    total_sum =0
    total_n = 0
    grand_mean = 0
    resid = pd.Series([])

    #calculate sum of square (within groups)
    for group in groups:
        grp_mean = np.mean(group)
        grp_size = group.size

        ss = np.sum((group - grp_mean)**2)
        SSw += ss
        total_n += grp_size
        total_sum += np.sum(group)

        resid = pd.concat([resid, (group - grp_mean)])

    #compute grand mean
    grand_mean = total_sum / total_n

    #calculate sum of square (between groups)
    for group in groups:
        grp_mean = np.mean(group)
        grp_size = group.size

        SSb += (grp_size * (grp_mean - grand_mean)**2)

    #compute SSt
    SSt = SSb + SSw

    #compute degree of freedoms
    df_t = total_n - 1
    df_b = len(groups) - 1
    df_w = df_t - df_b
```

```
    #compute F statistic
    MSb = (SSb / df_b)
    MSw = (SSw / df_w)
    f_stat = MSb / MSw

    #find p-value and critical value(corresponding to alpha).
    f_cr = stats.f.ppf(1 - alpha, dfn=df_b, dfd=df_w)
    pval = 1 - stats.f.cdf(f_stat, dfn=df_b, dfd=df_w)


    res = [(SSb, df_b, MSb, f_stat, f_cr, pval),
     (SSw, df_w, MSw, None, None, None),
     (SSt, df_t, None, None, None, None)]

    return (
        pd.DataFrame(data = res, index=['Between', 'Within', 'Total'], columns␣
↪= ['SS', 'df', 'MS', 'F', 'F-Cr', 'P-Val']),
        resid
    )
```

[566]:
```
#compute ANOVA using custom function
table, resid = anova_oneway(groups, 0.05)
table
```

	SS	df	MS	F	F-Cr	P-Val
Between	6.338070e+06	3	2.112690e+06	65.530241	2.605725	1.110223e-16
Within	3.508348e+08	10882	3.223992e+04	NaN	NaN	NaN
Total	3.571729e+08	10885	NaN	NaN	NaN	NaN

[568]:
```
#compute ANOVA using statsmodel
import statsmodels.api as sm
from statsmodels.formula.api import ols

model = ols('count ~ C(weather)', data=df).fit()
aov_table = sm.stats.anova_lm(model, typ=2)
print(aov_table)
```

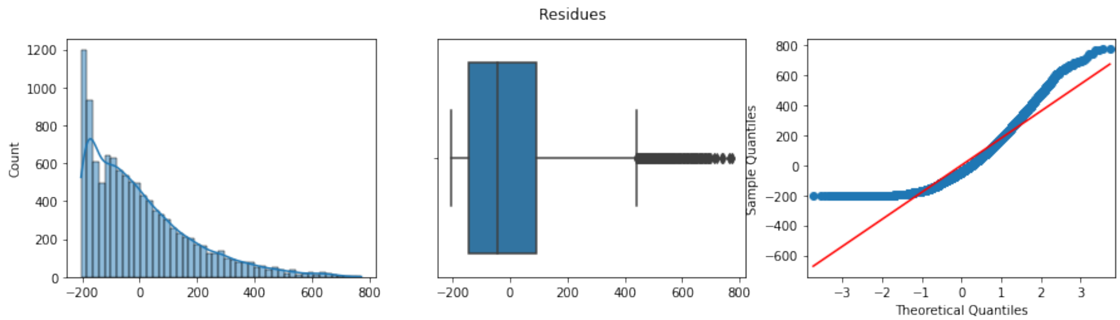
```
                  sum_sq       df          F        PR(>F)
C(weather)  6.338070e+06      3.0  65.530241  5.482069e-42
Residual    3.508348e+08  10882.0        NaN           NaN
```

[569]:
```
#compute ANOVA using sci-py
stats.f_oneway(*groups)
```

[569]: F_onewayResult(statistic=65.53024112793271, pvalue=5.482069475935669e-42)

2.13.1 Residue analysis

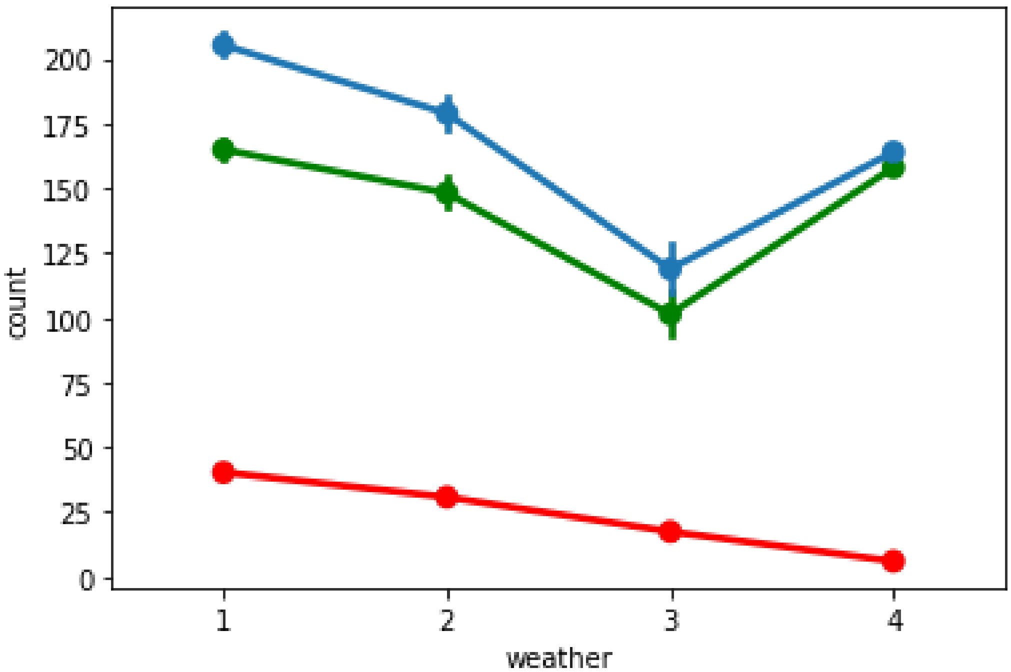
[576]:
```
testnorm(resid, 'Residues')
```




```
Shapiro-Wilk Test metrics
Residues   : ShapiroResult(statistic=0.8916589021682739, pvalue=0.0)
```

2.13.2 mean effect plot for count ~ weather

[598]:
```
sns.pointplot(y="casual", x='weather', ci=95, data=df, color='red', dodge=True)
sns.pointplot(y="registered", x='weather', ci=95, data=df, color='green',␣
↪dodge=True)
sns.pointplot(y="count", x='weather', ci=95, data=df, dodge=True)

plt.show()
```



Observation

1. Our data values do not meet variance homogeniety assumption. Similarly, residues from the ANOVA analysis are not normally distributed (as confirmed above). This findings of ANOVA test may be potentially invalid. Regardless, we can draw insights from the analysis.

2. Based on the output of the one way ANOVA test, we see that p-value is very low, and therefore we reject the null hypothesis of mean equality. This means that there is atleast one weather level combination for which mean cycle demand values are significantly different. We can also visually see this in the bi-variate graph between count and weather. **In other words, weather is a signigicant factor in predicting rental counts (at significance level 0.05)**

3. Based on plots of casual (red) and registered(green) users, we can observe that rental demands from 'registered' users vary similar to the rental demands for total users (blue). However, 'casual' users rental demand has a different pattern. Specifically, count of casual users is very low in 'weather 4' while 'registered' users is fairly high.

Recommendations

1. Cycle rental demands are highest in weather 1, second highest in weather 2, third highest in weather 4, and lowest in weather 3. The business should consider adopting a strategy to increase cycle demands in weather-3. Also, weather-1 and weather-2 see highest rental demands, so business should continue to spend resources to continue to meet the demands.

2. Cycle rental demands for casual users goes down considerably in weather-4 even when the demands from registered users goes up (compared to weather 3). The business should consider finding a strategy to attract more casual users in weather-4.

3. A general trend shows that there are more number of registered users than casual users. This is not necessarily a bad sign. There might be a potential to attract more casual users by by offering discounts and running promotional campaigns advocating health/environment benefits of cycling.

2.14 Test 3: Check if No. of cycles rented is similar or different in different seasons?

'season' is a categorical variable with 4 levels. We need to compare rental count means for each season level. We can use one-way ANOVA test for this purpose (one factor with four levels).

H0: All population means are equal. That is, cycle rental count is similar across different seasons.

H1: Not all population means are equal. In other words, cycle rental count is different for atleast one season type.

2.14.1 One-way ANOVA assumptions

For Anova results to be valid, the following assumptions need to hold.

1. Responses for a given group are independent - **We assume this as true**.
2. Variances of populations are equal. **(needs to be checked)**
3. Response variable residuals are normally distributed (or approximately normally distributed) - **We will check this post ANOVA test.**

2.15 check variance homogeneity

[579]:
```
factor = 'season'
resp = 'count'
levels = df.groupby(factor)[resp].groups.keys()

#generate groups for each level of weather variable.
groups = []
for level in levels:
    group = df[df[factor] == level][resp]
    print(f'variance for group {level}: {np.var(group, ddof=1)}')
    groups.append(group)

#levene's test
print("\nLevene's test to check if population variances are equal")
print('H0: All four population variances are equal')
print('H1: Not all four population variances are equal')
print(f"Levene's test metric: {stats.levene(*groups)}")
```

```
variance for group 1: 15693.568533717144
variance for group 2: 36867.01182553242
variance for group 3: 38868.517012662865
variance for group 4: 31549.720316669307
```

```
Levene's test to check if population variances are equal
H0: All four population variances are equal
H1: Not all four population variances are equal
Levene's test metric: LeveneResult(statistic=187.7706624026276,
pvalue=1.0147116860043298e-118)
```

Observation

Ideally, for ANOVA results to be considered valid, all population variances (that is variance for each group) should be the same. We see high variations among different groups. Levene's test confirms this by returning a very small p-value. We thus reject the null hypothesis of variance homogeneity. For this case-study, we will however continue with ANOVA test.

2.16 One way ANOVA test (count ~ season)

[581]:
```
#compute ANOVA using custom function
table, resid = anova_oneway(groups, 0.05)
table
```

[581]:

	SS	df	MS	F	F-Cr	P-Val
Between	2.190083e+07	3	7.300277e+06	236.946711	2.605725	1.110223e-16
Within	3.352721e+08	10882	3.080979e+04	NaN	NaN	NaN
Total	3.571729e+08	10885	NaN	NaN	NaN	NaN

[584]:
```
#compute ANOVA using statsmodel
import statsmodels.api as sm
from statsmodels.formula.api import ols

model = ols('count ~ C(season)', data=df).fit()
aov_table = sm.stats.anova_lm(model, typ=2)
print(aov_table)
```

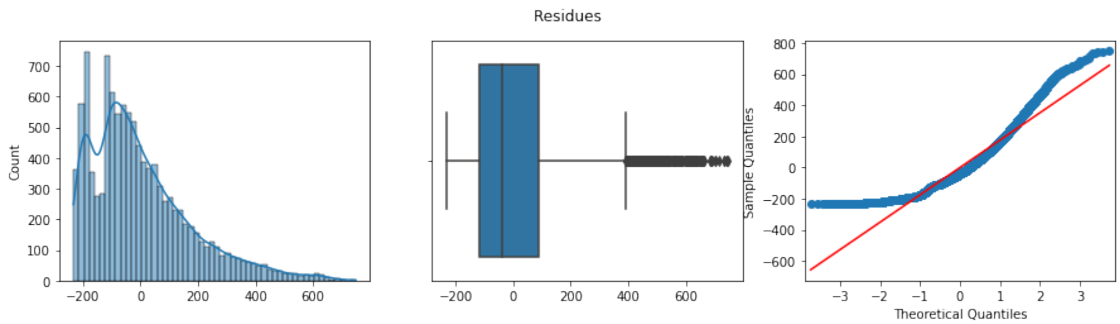
```
                 sum_sq       df           F         PR(>F)
C(season)  2.190083e+07      3.0  236.946711  6.164843e-149
Residual   3.352721e+08  10882.0         NaN            NaN
```

[583]:
```
#compute ANOVA using sci-py
stats.f_oneway(*groups)
```

[583]: F_onewayResult(statistic=236.94671081032106, pvalue=6.164843386499654e-149)

2.16.1 Residue analysis

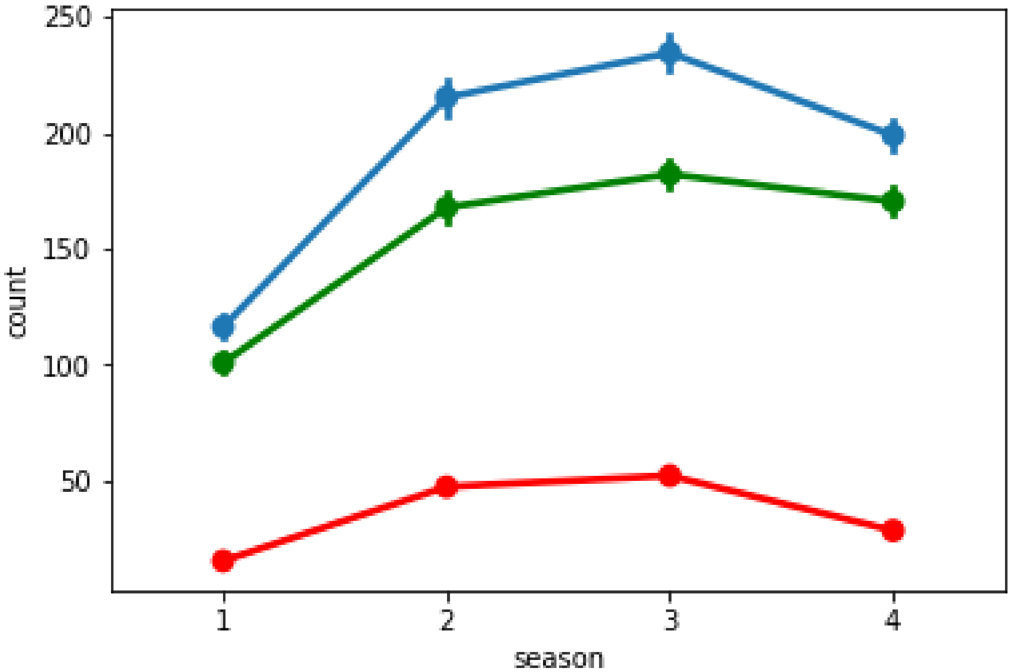
[586]:
```
testnorm(resid, 'Residues')
```




```
Shapiro-Wilk Test metrics
Residues  : ShapiroResult(statistic=0.9127053022384644, pvalue=0.0)
```

2.16.2 mean effect plot for count ~ season

[597]:
```
sns.pointplot(y="casual", x='season', ci=95, data=df, color='red', dodge=True)
sns.pointplot(y="registered", x='season', ci=95, data=df, color='green',␣
 ↪dodge=True)
sns.pointplot(y="count", x='season', ci=95, data=df, dodge=True)

plt.show()
```



Observation

1. Our data values do not meet variance homogeniety assumption. Similarly, residues from the ANOVA analysis are also not normally distributed (as confirmed above). Thus the findings of ANOVA test may be potentially invalid. Regardless, we can draw insights from the analysis.

2. Based on the output of the one way ANOVA test, we see that p-value is very low, and therefore we reject the null hypothesis of mean equality. This means that there is atleast one season level combination for which mean cycle demand values are significantly different. We can also visually see this in the bi-variate graph between count and season. **In other words, season is a signigicant factor in predicting rental counts (at significance level 0.05)**

3. Based on the plot above, we see that demands from both 'casual'and 'registered' user tend to follow similar pattern across various seasons.

Recommendation

1. Season 3 and Season 2 have the highest rental demands from both registered and casual users. season 4 has somewhat lesser demand. However,the demands goes down considerably in season 1. The business should consider adopting a strategty to increase cycle demands in season-1. Also, season-3 and season-2 see highest rental demands, so business should spend necessary resources to continue to meet the demands.

2. A general trend shows that there are more number of registered users than casual users. This is not necessarily a bad sign. There might be a potential to attract more casual users by by offering discounts and running promotional campaigns advocating health/environment benefits of cycling.

2.17 Test 4 - Is weather dependent on the season?

'weather' and 'season' both are categorical variables with four levels. We can use 'chi-squared' test for indepedence.

H0: weather and season are indepedent variables.

H1: weather and season are not independent variables.

[367]:
```
#calculate observed freq of rental counts
obs_freq = pd.crosstab(index=df['weather'], columns=df['season'],␣
↪values=df['count'], aggfunc=np.sum, margins=True)
obs_freq
```

[367]:
```
season        1       2       3       4      All
weather
1        223009  426350  470116  356588  1476063
2         76406  134177  139386  157191   507160
3         12919   27755   31160   30255   102089
4           164       0       0       0      164
All      312498  588282  640662  544034  2085476
```

Note: Since cell values in 3 cells above are zero (less than 5), we should consider using Fisher's exact test. However, it's out of scope for this case study, so we continue with

chi-square method

[386]:
```
#calculate expected freq of rental counts (assuming indepedence between weather␣
↪and season)
exp_freq = obs_freq.copy().astype('float')
for i in range(1,5):
    for j in range(1,5):
        exp_freq.loc[i][j] = (exp_freq.loc[i]['All'] * exp_freq.loc['All'][j]) /
↪ exp_freq.loc['All']['All']

exp_freq
```

[386]:

season weather	1	2	3	4	All
1	221180.553204	416375.587044	453449.223921	385057.635831	1476063.0
2	75995.353425	143062.350811	155800.469495	132301.826269	507160.0
3	15297.518802	28797.800166	31361.925488	26631.755545	102089.0
4	24.574568	46.261980	50.381097	42.782356	164.0
All	312498.000000	588282.000000	640662.000000	544034.000000	2085476.0

[401]:
```
# calculate chi-square statistic
chi_stat = 0
for i in range(1,5):
    for j in range(1,5):
        chi_stat += ((obs_freq.loc[i][j] - exp_freq.loc[i][j])**2 / exp_freq.
↪loc[i][j])

chi_cv = stats.chi2.ppf(0.95,df=9)

print(f'chi-square statistic = {chi_stat}, chi-square critical value (at 95%␣
↪confidence level) = {chi_cv}')
print(f'dof = 9')

print('\ncalling scipy chi2_contigency to cross verify')
print(stats.chi2_contingency(pd.crosstab(index=df['weather'],␣
↪columns=df['season'], values=df['count'], aggfunc=np.sum)))
```

```
chi-square statistic = 11769.559450959447, chi-square critical value (at 95%
confidence level) = 16.918977604620448
dof = 9

calling scipy chi2_contigency to cross verify
(11769.559450959445, 0.0, 9, array([[2.21180553e+05, 4.16375587e+05,
4.53449224e+05, 3.85057636e+05],
       [7.59953534e+04, 1.43062351e+05, 1.55800469e+05, 1.32301826e+05],
       [1.52975188e+04, 2.87978002e+04, 3.13619255e+04, 2.66317555e+04],
       [2.45745681e+01, 4.62619795e+01, 5.03810967e+01, 4.27823557e+01]]))
```

Observation

For the Chi-square test of independence between 'weather' and 'season', the calculated chi-square statistic (11769) is greater than critical value ~16.91 at 95% confidence level. We also observe that pvalue is ~0. Thus we reject the null hypothesis. Thus, **we can state with 95% confidence that 'weather' and 'season' are not indepedent.**

Recommendations - None

[ ]:

```

```