Yulu_case_study

May 14, 2025

#**Yulu case study.** ##**Problem statement** : Yulu, India's leading micro-mobility service provider has collected data based on the service it has provided and wants to know the following.

- What independent variables significantly impact the dependent variable.
- How strongly these independent variables impact the output.

[5]:
```
#Importing the data from csv file to a pandas data frame
import pandas as pd
data = pd.read_csv('bike_sharing.csv')
```

[6]:
```
#Start of EDA. The next few blocks will be to explore the data to get an␣
↪understanding on a high level.
data.head()
```

[6]:
```
              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
```

[7]:
```
data.shape
```

[7]:
```
(10886, 12)
```

[8]:
```
data.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
```

[9]:
```
#Now I am converting categorical attributes to 'category' to check if memory␣
↪usage can be optimized.
#First I am taking the column season and will convert this to categorical.
#I am keeping datetime as object itself as this I feel is better as is.
```

[9]:
```
data['season_cat'] = data['season'].map({1: 'spring', 2: 'summer', 3: 'fall', 4:
↪ 'winter'})
data['season_cat'] = data['season_cat'].astype('category')
```

[10]:
```
#Performing the below to check if the new column matches the existing column
data['season_cat'].value_counts()
```

[10]:
```
season_cat
winter    2734
fall      2733
summer    2733
spring    2686
Name: count, dtype: int64
```

[11]:
```
data['season'].value_counts()
```

[11]:
```
season
4    2734
2    2733
3    2733
1    2686
Name: count, dtype: int64
```

[12]:
```
#As summer and fall had the same count and the order differed I am doing the␣
↪below to confirm the new column is as expected.
assert (data['season'].map({1: 'spring', 2: 'summer', 3: 'fall', 4: 'winter'})␣
↪== data['season_cat']).all(), "Mismatch found!"
```

```
print("All values correspond correctly!")
```

```
All values correspond correctly!
```

[13]:
```
#As, we can see this is as expected. Now I am deleting the original column␣
↪season
del data['season']
data.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   holiday     10886 non-null  int64
 2   workingday  10886 non-null  int64
 3   weather     10886 non-null  int64
 4   temp        10886 non-null  float64
 5   atemp       10886 non-null  float64
 6   humidity    10886 non-null  int64
 7   windspeed   10886 non-null  float64
 8   casual      10886 non-null  int64
 9   registered  10886 non-null  int64
 10  count       10886 non-null  int64
 11  season_cat  10886 non-null  category
dtypes: category(1), float64(3), int64(7), object(1)
memory usage: 946.5+ KB
```

[ ]:
```
#I can clearly see the reduction in memory usage with this. Now I am checking␣
↪this for the remaining columns quickly.
```

[14]:
```
data['holiday'].value_counts()
```

[14]:
```
holiday
0    10575
1      311
Name: count, dtype: int64
```

[15]:
```
data['holiday_cat'] = data['holiday'].map({1: 'holiday', 0: 'not_a_holiday'})
data['holiday_cat'] = data['holiday_cat'].astype('category')
```

[16]:
```
del data['holiday']
data.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   workingday   10886 non-null  int64
 2   weather      10886 non-null  int64
 3   temp         10886 non-null  float64
 4   atemp        10886 non-null  float64
 5   humidity     10886 non-null  int64
 6   windspeed    10886 non-null  float64
 7   casual       10886 non-null  int64
 8   registered   10886 non-null  int64
 9   count        10886 non-null  int64
 10  season_cat   10886 non-null  category
 11  holiday_cat  10886 non-null  category
dtypes: category(2), float64(3), int64(6), object(1)
memory usage: 872.2+ KB
```

[17]:
```python
data.head()
```

[17]:
```
              datetime  workingday  weather  temp   atemp  humidity  \
0  2011-01-01 00:00:00           0        1  9.84  14.395        81
1  2011-01-01 01:00:00           0        1  9.02  13.635        80
2  2011-01-01 02:00:00           0        1  9.02  13.635        80
3  2011-01-01 03:00:00           0        1  9.84  14.395        75
4  2011-01-01 04:00:00           0        1  9.84  14.395        75

   windspeed  casual  registered  count season_cat   holiday_cat
0        0.0       3          13     16     spring  not_a_holiday
1        0.0       8          32     40     spring  not_a_holiday
2        0.0       5          27     32     spring  not_a_holiday
3        0.0       3          10     13     spring  not_a_holiday
4        0.0       0           1      1     spring  not_a_holiday
```

[18]:
```python
data['workingday'].value_counts()
```

[18]:
```
workingday
1    7412
0    3474
Name: count, dtype: int64
```

[19]:
```python
data['workingday_cat'] = data['workingday'].map({1: 'workingday', 0:␣
↪'not_a_workingday'})
data['workingday_cat'] = data['workingday_cat'].astype('category')
del data['workingday']
data.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   weather         10886 non-null  int64
 2   temp            10886 non-null  float64
 3   atemp           10886 non-null  float64
 4   humidity        10886 non-null  int64
 5   windspeed       10886 non-null  float64
 6   casual          10886 non-null  int64
 7   registered      10886 non-null  int64
 8   count           10886 non-null  int64
 9   season_cat      10886 non-null  category
 10  holiday_cat     10886 non-null  category
 11  workingday_cat  10886 non-null  category
dtypes: category(3), float64(3), int64(5), object(1)
memory usage: 797.9+ KB
```

[20]:
```
data.shape
```

[20]:
```
(10886, 12)
```

[21]:
```
#For weather I am directly converting this to categorical column without␣
↪mapping to actual, as the actual values are long strings.
data['weather'] = data['weather'].astype('category')
```

[22]:
```
data.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   weather         10886 non-null  category
 2   temp            10886 non-null  float64
 3   atemp           10886 non-null  float64
 4   humidity        10886 non-null  int64
 5   windspeed       10886 non-null  float64
 6   casual          10886 non-null  int64
 7   registered      10886 non-null  int64
 8   count           10886 non-null  int64
 9   season_cat      10886 non-null  category
 10  holiday_cat     10886 non-null  category
 11  workingday_cat  10886 non-null  category
dtypes: category(4), float64(3), int64(4), object(1)
memory usage: 723.7+ KB
```

[23]:
```
data.shape
```

[23]:
```
(10886, 12)
```

[24]:
```
data.head()
```

[24]:
```
              datetime  weather  temp   atemp  humidity  windspeed  casual  \
0  2011-01-01 00:00:00        1  9.84  14.395        81        0.0       3
1  2011-01-01 01:00:00        1  9.02  13.635        80        0.0       8
2  2011-01-01 02:00:00        1  9.02  13.635        80        0.0       5
3  2011-01-01 03:00:00        1  9.84  14.395        75        0.0       3
4  2011-01-01 04:00:00        1  9.84  14.395        75        0.0       0

   registered  count season_cat    holiday_cat    workingday_cat
0          13     16     spring  not_a_holiday  not_a_workingday
1          32     40     spring  not_a_holiday  not_a_workingday
2          27     32     spring  not_a_holiday  not_a_workingday
3          10     13     spring  not_a_holiday  not_a_workingday
4           1      1     spring  not_a_holiday  not_a_workingday
```

[ ]:
```
#not going to make any changes to temp, atemp, humidity, windspeed, casual,␣
↪registered and count
```

[ ]:
```
#With this we can see that the data now takes less memory compared to initially␣
↪when imported.
```

[25]:
```
data.isnull().sum()
```

[25]:
```
datetime          0
weather           0
temp              0
atemp             0
humidity          0
windspeed         0
casual            0
registered        0
count             0
season_cat        0
holiday_cat       0
workingday_cat    0
dtype: int64
```

[ ]:
```
#Can see that there are no missing values in this dataset
```

[26]:
```
data.describe()
```

```
[26]:              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
```

```
[ ]: #From the above statistical summary of numerical columns we can see that there␣
     ↪are outliers in casual user count by the difference in the mean and median.
     #Although not much difference there is a difference in the mean and median of␣
     ↪registered and count as well.
```

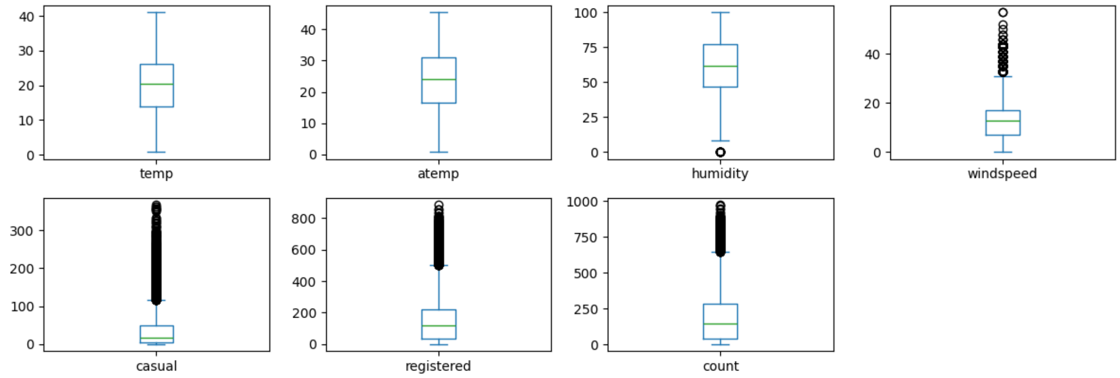
```
[27]: data.describe(include=['object', 'category'])
```

```
[27]:                   datetime  weather season_cat    holiday_cat workingday_cat
count                    10886    10886      10886          10886          10886
unique                   10886        4          4              2              2
top     2011-01-01 00:00:00        1     winter  not_a_holiday     workingday
freq                         1     7192       2734          10575           7412
```

```
[ ]: # From the above statistical summary of categorical columns we can see that␣
     ↪there are more datapoints for weather 1
     # i.e. 1: Clear, Few clouds, partly cloudy, partly cloudy, which makes sense␣
     ↪given that more rides are to be expected during a clear weather.
     # Also, as can be seen the season with top data points is winter probably␣
     ↪indicating that the rides are taken when the weather is pleasant
     # holiday_cat : Here we see that the top count is for not_a_holiday probably␣
     ↪indicating that the rides are used by employees who
     # use it for the first and last miles of office commute after a public transit
     # workingday_cat : same conclusion as above can be said of this as well
```

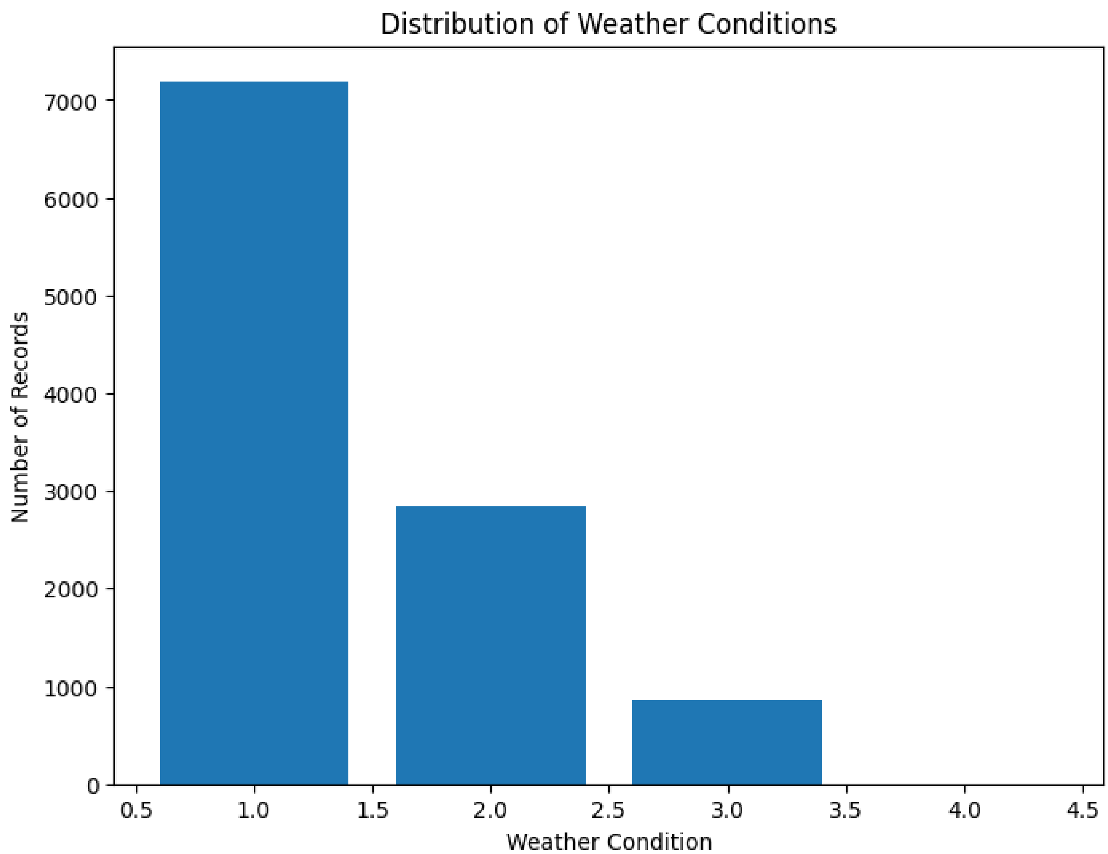
```
[ ]: #Univariate Analysis
```

[28]:
```python
#Plotting a box plot for all the numerical columns
import matplotlib.pyplot as plt

numerical_cols = data.select_dtypes(include=['number']).columns

data[numerical_cols].plot(kind='box', figsize=(12, 6), subplots=True, layout=(3, 4))
plt.tight_layout()
plt.show()
```



[ ]:
```python
#From these plots it can be seen that casual, registered and count columns have outliers as seen before in the statisrtical summary.
#But here we can also see the outliers being present in windspeed column as well.
```

[29]:
```python
#Plotting a bar chart for categorical columns
import matplotlib.pyplot as plt

weather_counts = data['weather'].value_counts()

plt.figure(figsize=(8, 6))
plt.bar(weather_counts.index, weather_counts.values)
plt.xlabel('Weather Condition')
plt.ylabel('Number of Records')
plt.title('Distribution of Weather Conditions')
plt.show()
```

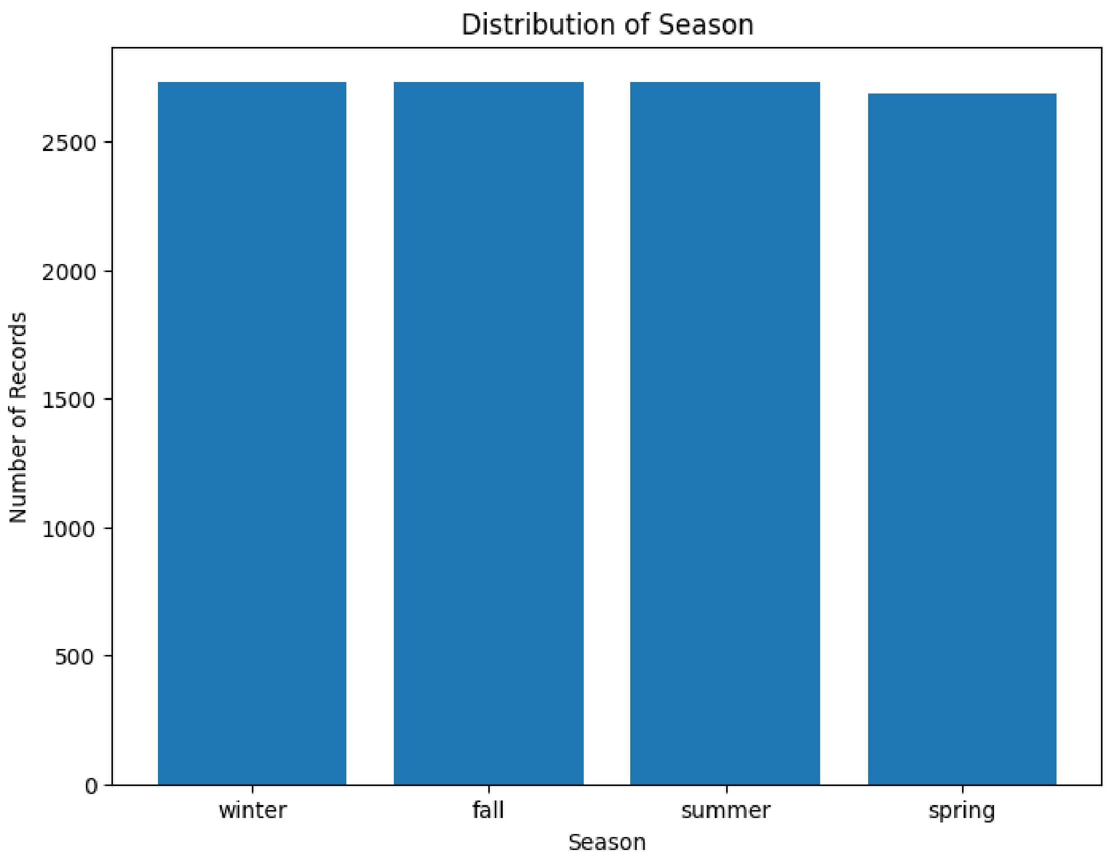


[30]:
```
data['weather'].value_counts()
```

[30]:
```
weather
1    7192
2    2834
3     859
4       1
Name: count, dtype: int64
```

[ ]:
```
# • 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

#Considering the above mapping, and the value counts along with the␣
↪visualization it is seen that the datapoints are more for weather type 1,␣
↪followed by 2
# and then we have 3 and 4, but these are very less compared to 1 and 2
```

[31]:
```python
import matplotlib.pyplot as plt

weather_counts = data['season_cat'].value_counts()

plt.figure(figsize=(8, 6))
plt.bar(weather_counts.index, weather_counts.values)
plt.xlabel('Season')
plt.ylabel('Number of Records')
plt.title('Distribution of Season')
plt.show()
```

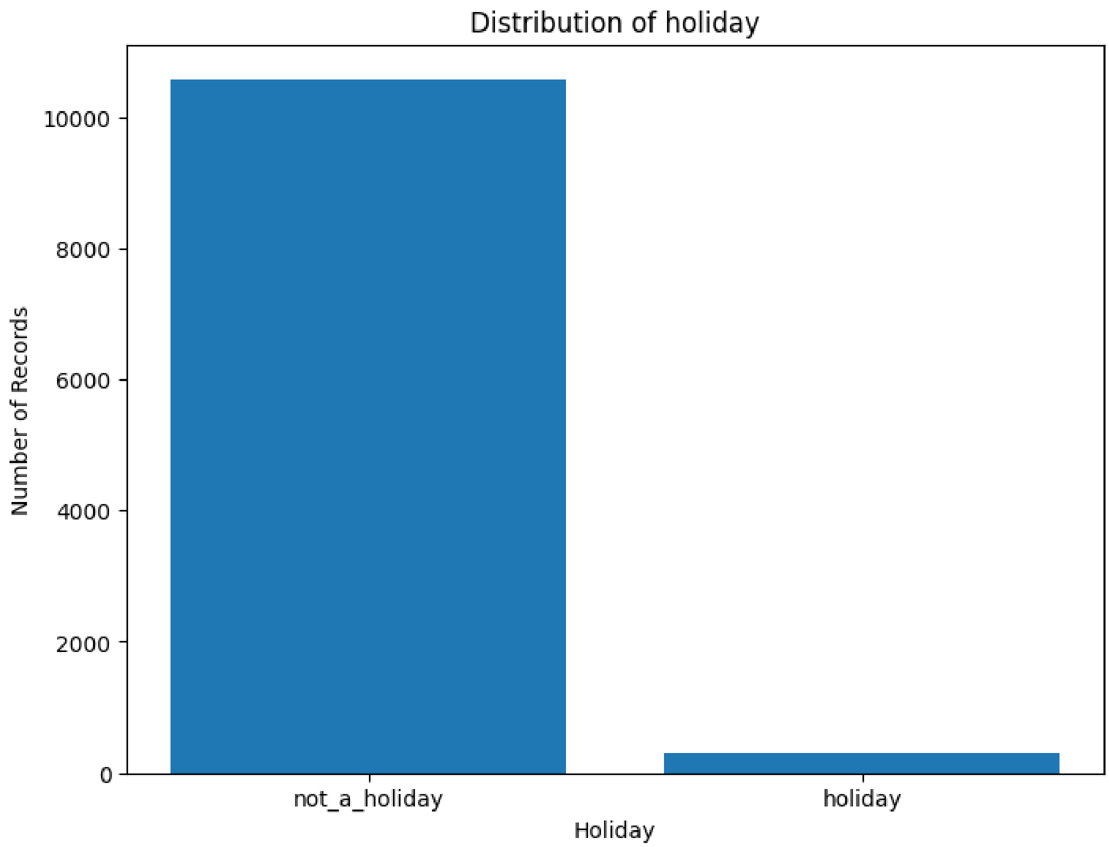


[32]:
```python
data['season_cat'].value_counts()
```

[32]:
```
season_cat
winter    2734
fall      2733
summer    2733
spring    2686
Name: count, dtype: int64
```

[ ]:
```
#from the visual as well as the above summary it is seen that all the seasons
↪have almost equal datapoints
```

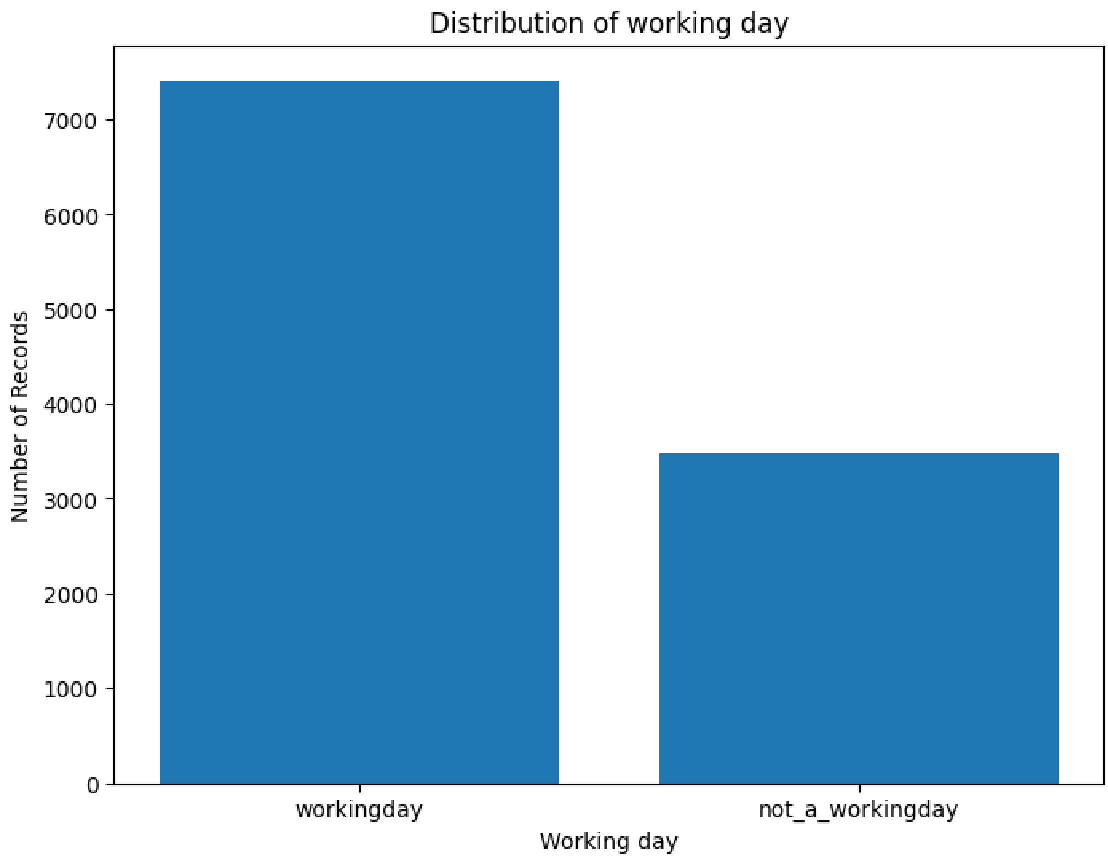
[33]:
```
import matplotlib.pyplot as plt

weather_counts = data['holiday_cat'].value_counts()

plt.figure(figsize=(8, 6))
plt.bar(weather_counts.index, weather_counts.values)
plt.xlabel('Holiday')
plt.ylabel('Number of Records')
plt.title('Distribution of holiday')
plt.show()
```



[ ]:
```
#We see there are more data points for not_a_holiday compared to holiday,
↪indicating more usage during working days
```

[34]:
```
import matplotlib.pyplot as plt

weather_counts = data['workingday_cat'].value_counts()
```

```python
plt.figure(figsize=(8, 6))
plt.bar(weather_counts.index, weather_counts.values)
plt.xlabel('Working day')
plt.ylabel('Number of Records')
plt.title('Distribution of working day')
plt.show()
```



[ ]:
```python
#We see there are more data points for workingday compared to not_a_workingday,␣
↪indicating more usage during working days, but there
#is a about 50% usage during "not_a_workingday" as well
```

[ ]:
```python
#Bi-variate-analysis
```

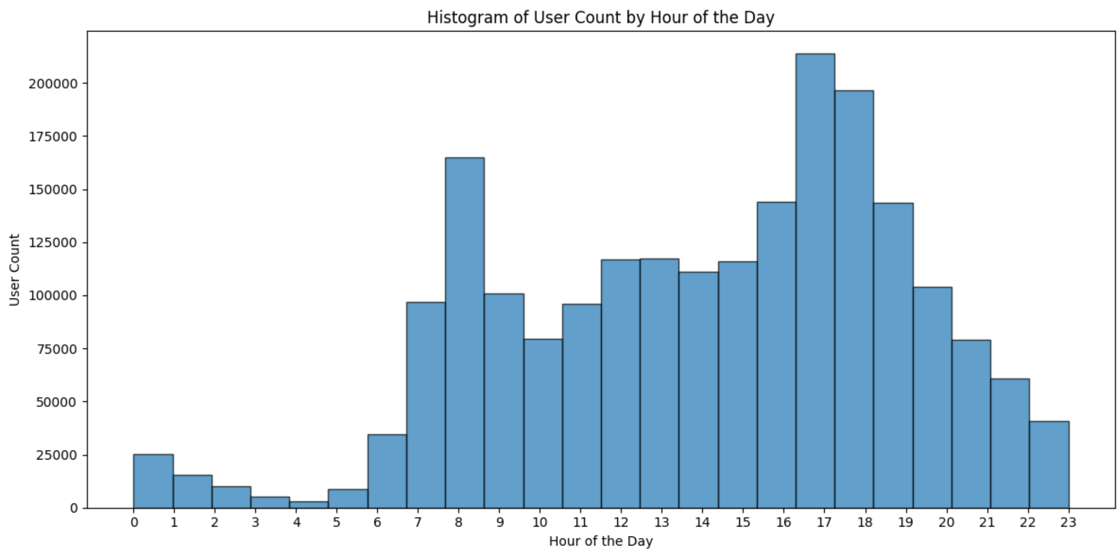
[35]:
```python
#Converting datetime to type datetime. So, that I can use this in bi-variate␣
↪plots
data['datetime'] = pd.to_datetime(data['datetime'])
```

[36]:
```python
data.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   weather         10886 non-null  category
 2   temp            10886 non-null  float64
 3   atemp           10886 non-null  float64
 4   humidity        10886 non-null  int64
 5   windspeed       10886 non-null  float64
 6   casual          10886 non-null  int64
 7   registered      10886 non-null  int64
 8   count           10886 non-null  int64
 9   season_cat      10886 non-null  category
 10  holiday_cat     10886 non-null  category
 11  workingday_cat  10886 non-null  category
dtypes: category(4), datetime64[ns](1), float64(3), int64(4)
memory usage: 723.7 KB
```

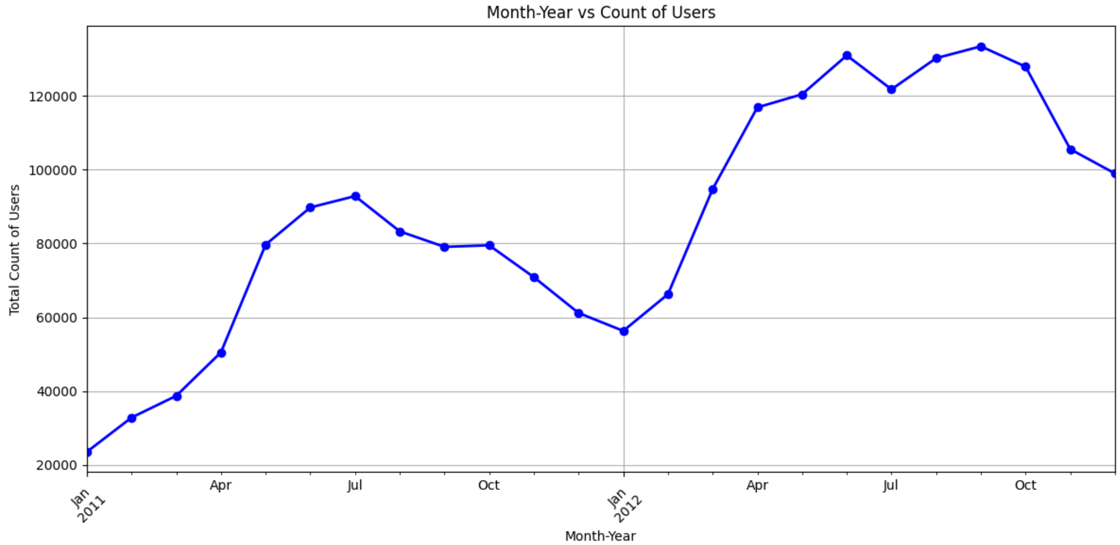
[37]:

```python
#Count vs datetime(hour extracted from this)
import matplotlib.pyplot as plt

data['hour'] = data['datetime'].dt.hour

plt.figure(figsize=(12, 6))
plt.hist(data['hour'], bins=24, weights=data['count'], edgecolor='black',␣
 ↪alpha=0.7)
plt.title('Histogram of User Count by Hour of the Day')
plt.xlabel('Hour of the Day')
plt.ylabel('User Count')
plt.xticks(range(0, 24))
plt.tight_layout()
plt.show()
```



[ ]:
```
#From this we see that the usage is more at 8 AM and then at 16,17,18,19 hours,␣
↪which is the start and end of the working hours
```

[38]:
```
import matplotlib.pyplot as plt
import pandas as pd


data['month_year'] = data['datetime'].dt.to_period('M').dt.to_timestamp()

month_year_counts = data.groupby('month_year')['count'].sum()

month_range = pd.date_range(start=month_year_counts.index.min(),
                            end=month_year_counts.index.max(),
                            freq='MS')
month_year_counts = month_year_counts.reindex(month_range, fill_value=0)

plt.figure(figsize=(12, 6))
month_year_counts.plot(kind='line', marker='o', color='b', linestyle='-',␣
↪linewidth=2, markersize=6)
plt.title('Month-Year vs Count of Users')
plt.xlabel('Month-Year')
plt.ylabel('Total Count of Users')
plt.xticks(rotation=45)
plt.tight_layout()
plt.grid(True)
plt.show()
```



[ ]:
```
#From the above plot we can see that there is no regular pattern in the count␣
↪of total users.
#It goes to a peak of 90,000+ in July 2011 and then falls to a low of 60,000 in␣
↪Jan 2012
#From here it again raises to a new peak of 120,000 in Sep 2012 and then␣
↪appears to fall to 100,000 in Dec 2012. We do not have data after this
#to check if the down trend continues.
```

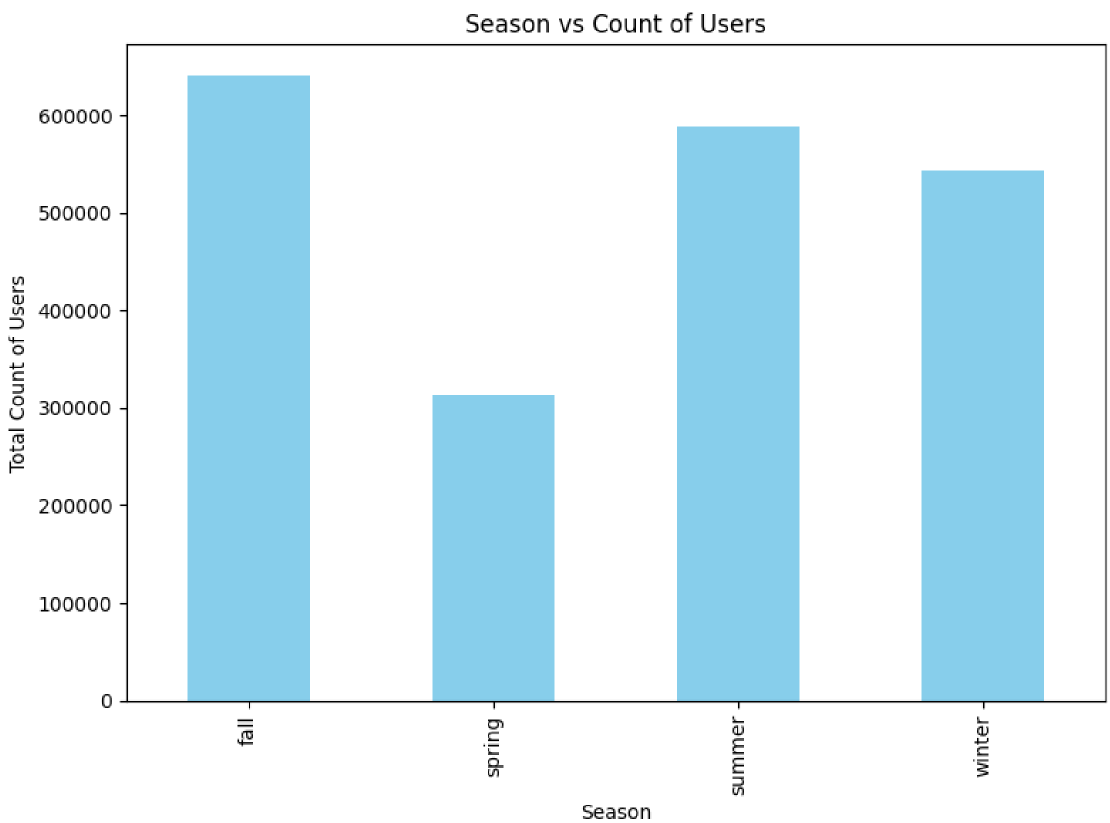
[39]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10886 entries, 0 to 10885
Data columns (total 14 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   datetime        10886 non-null  datetime64[ns]
 1   weather         10886 non-null  category
 2   temp            10886 non-null  float64
 3   atemp           10886 non-null  float64
 4   humidity        10886 non-null  int64
 5   windspeed       10886 non-null  float64
 6   casual          10886 non-null  int64
 7   registered      10886 non-null  int64
 8   count           10886 non-null  int64
 9   season_cat      10886 non-null  category
 10  holiday_cat     10886 non-null  category
 11  workingday_cat  10886 non-null  category
 12  hour            10886 non-null  int32
 13  month_year      10886 non-null  datetime64[ns]
```

```
dtypes: category(4), datetime64[ns](2), float64(3), int32(1), int64(4)
memory usage: 851.2 KB
```

[40]:
```python
#season vs count
import matplotlib.pyplot as plt


plt.figure(figsize=(8, 6))
data.groupby('season_cat')['count'].sum().plot(kind='bar', color='skyblue')

plt.title('Season vs Count of Users')
plt.xlabel('Season')
plt.ylabel('Total Count of Users')

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

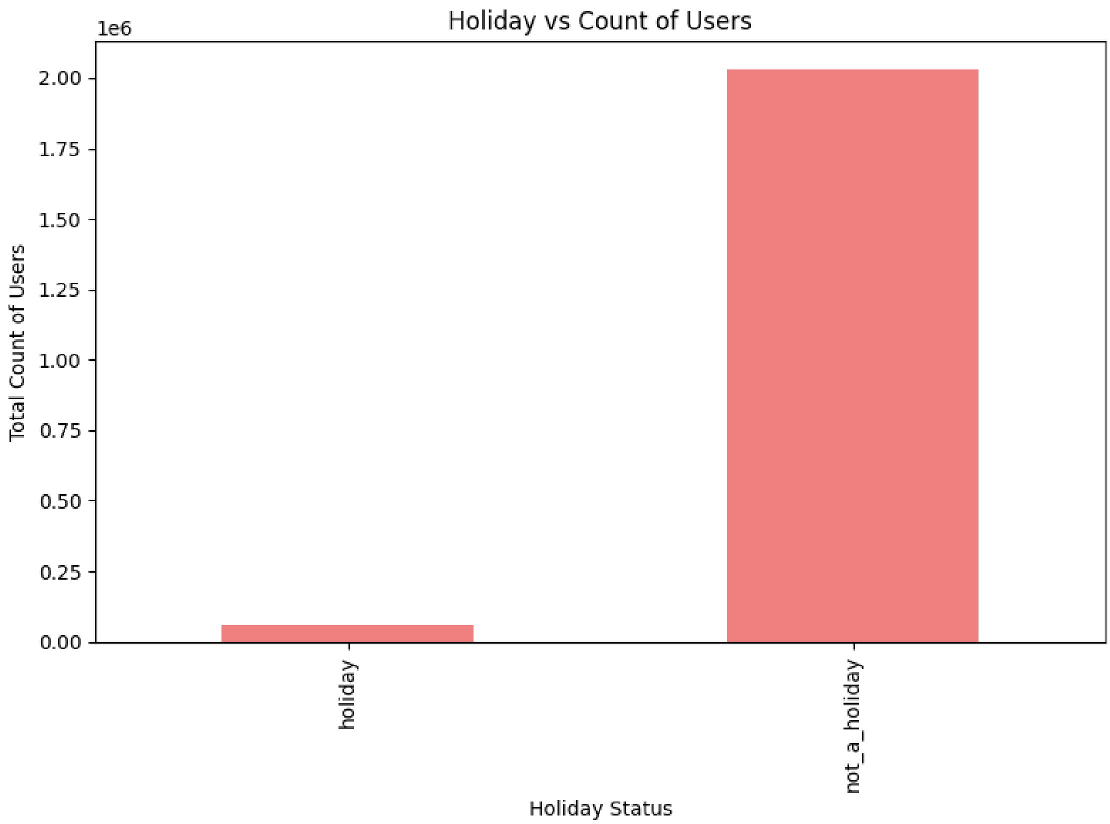
```
<ipython-input-40-7324de861110>:6: FutureWarning: The default of observed=False
is deprecated and will be changed to True in a future version of pandas. Pass
observed=False to retain current behavior or observed=True to adopt the future
default and silence this warning.
  data.groupby('season_cat')['count'].sum().plot(kind='bar', color='skyblue')
```



[ ]:
```
#From this plot we can see that count of users are more in fall followed by␣
↪summer and winter. The user count is least in spring
```

[41]:
```
#holiday vs count
import matplotlib.pyplot as plt

holiday_counts = data.groupby('holiday_cat')['count'].sum()

plt.figure(figsize=(8, 6))
holiday_counts.plot(kind='bar', color='lightcoral')

plt.title('Holiday vs Count of Users')
plt.xlabel('Holiday Status')
plt.ylabel('Total Count of Users')

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

```
<ipython-input-41-c80966273551>:4: FutureWarning: The default of observed=False
is deprecated and will be changed to True in a future version of pandas. Pass
observed=False to retain current behavior or observed=True to adopt the future
default and silence this warning.
  holiday_counts = data.groupby('holiday_cat')['count'].sum()
```

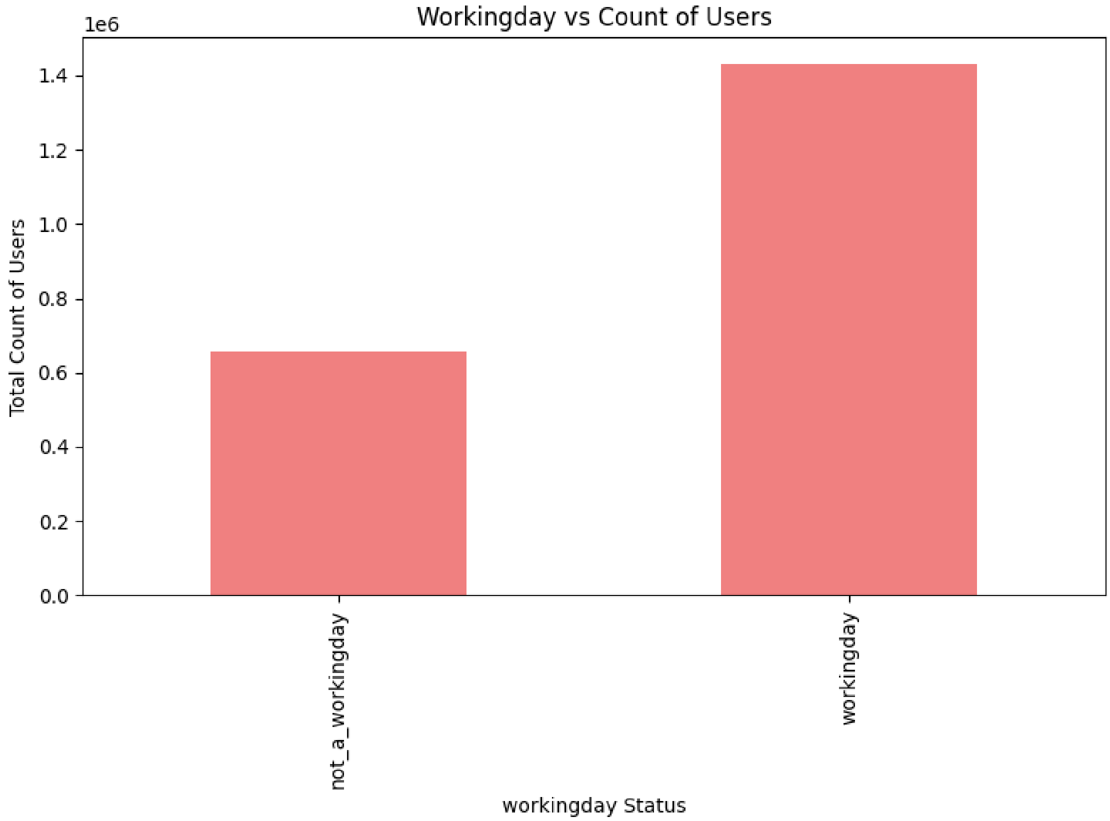


[ ]:
```
#From this it is clear that there are more users when it is not a holiday, and
↪users are very few in a holiday
```

[42]:
```
#workingday vs count
import matplotlib.pyplot as plt

holiday_counts = data.groupby('workingday_cat')['count'].sum()

plt.figure(figsize=(8, 6))
holiday_counts.plot(kind='bar', color='lightcoral')

plt.title('Workingday vs Count of Users')
plt.xlabel('workingday Status')
plt.ylabel('Total Count of Users')

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

```
<ipython-input-42-ad32c1c871e6>:4: FutureWarning: The default of observed=False
is deprecated and will be changed to True in a future version of pandas. Pass
observed=False to retain current behavior or observed=True to adopt the future
```

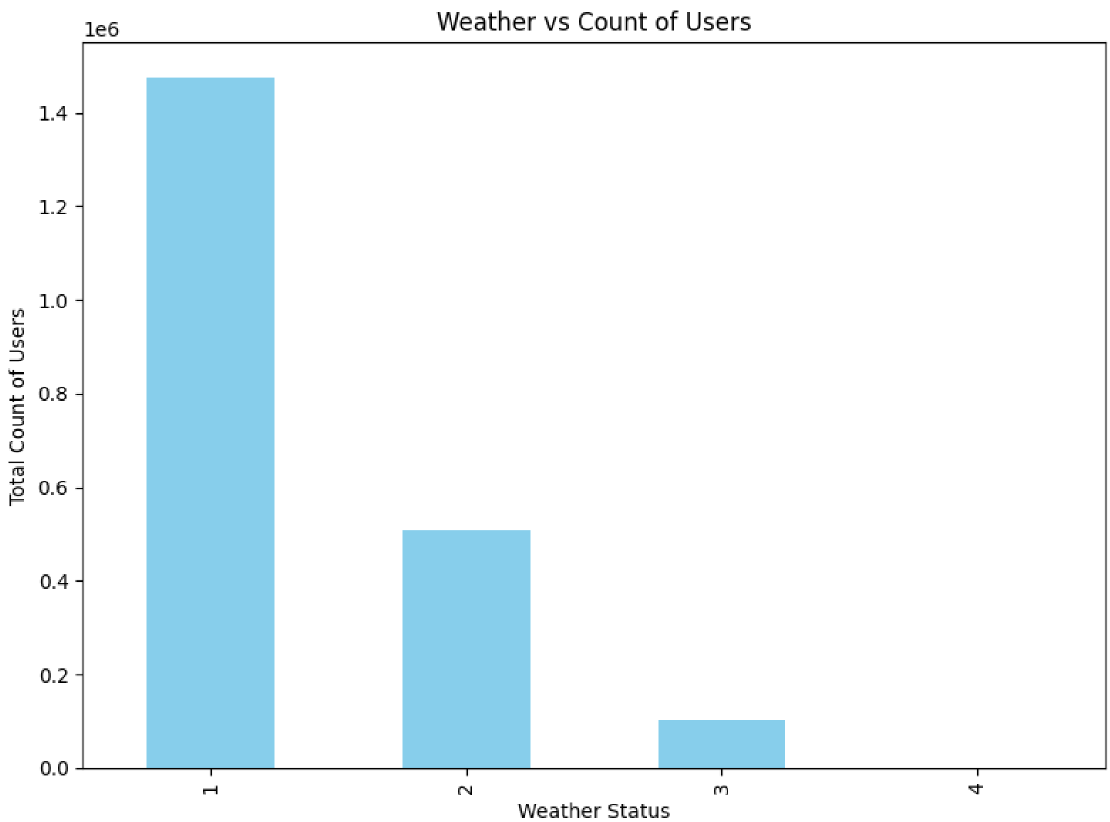
```
default and silence this warning.
  holiday_counts = data.groupby('workingday_cat')['count'].sum()
```



[ ]:
```
#From this it is clear that user count is more on a workingday when compared to
↪a non-working day. However, the count of users on
#a non-working day is approx 50% of the user count on a working day
```

[43]:
```
#weather vs count
import matplotlib.pyplot as plt

holiday_counts = data.groupby('weather')['count'].sum()

plt.figure(figsize=(8, 6))
holiday_counts.plot(kind='bar', color='skyblue')

plt.title('Weather vs Count of Users')
plt.xlabel('Weather Status')
plt.ylabel('Total Count of Users')

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

```
&lt;ipython-input-43-846b840c02ab&gt;:4: FutureWarning: The default of observed=False
is deprecated and will be changed to True in a future version of pandas. Pass
observed=False to retain current behavior or observed=True to adopt the future
default and silence this warning.
  holiday_counts = data.groupby('weather')['count'].sum()
```



[118]:
```
# • 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

# As expected the count of users follow the order 1 -> 2 -> 3 -> 4, where 1 is␣
  ↪the highest and 4 is the least with almost nigligible count
```

[128]:
```
#Checking for co-relation between temp and count
#For this I am checking if the data is normal
```

[44]:
```
import scipy.stats as stats
```

```
stat_temp, p_value_temp = stats.shapiro(data['temp'])
stat_count, p_value_count = stats.shapiro(data['count'])

print(f"Shapiro-Wilk test for temp: p-value = {p_value_temp}")
print(f"Shapiro-Wilk test for count: p-value = {p_value_count}")


if p_value_temp < 0.05:
    print("Reject the null hypothesis for temp: temp is not normally␣
 ↪distributed.")
else:
    print("Fail to reject the null hypothesis for temp: temp is normally␣
 ↪distributed.")

if p_value_count < 0.05:
    print("Reject the null hypothesis for count: count is not normally␣
 ↪distributed.")
else:
    print("Fail to reject the null hypothesis for count: count is normally␣
 ↪distributed.")
```

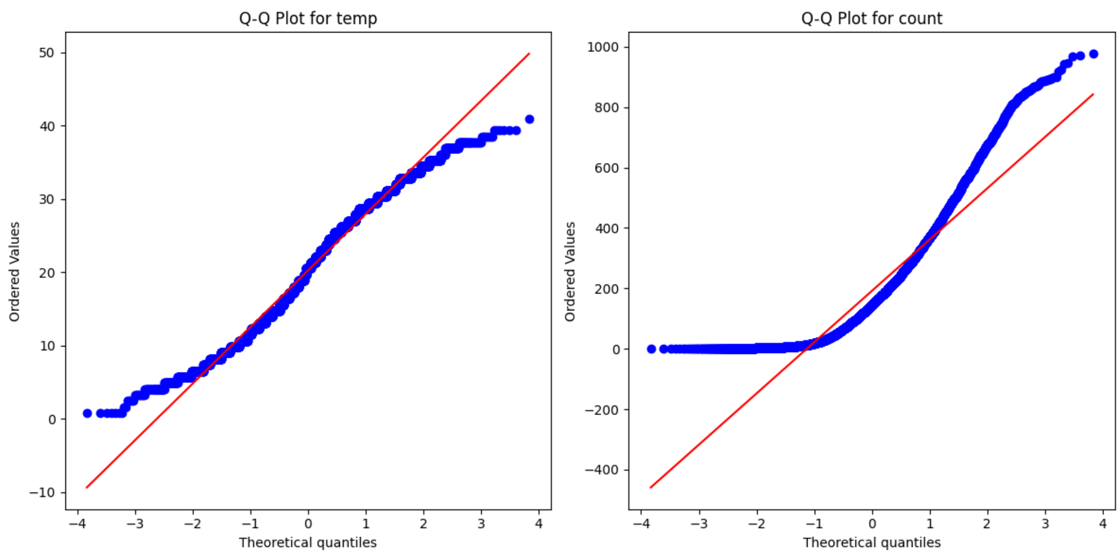
```
Shapiro-Wilk test for temp: p-value = 4.4416921644612106e-36
Shapiro-Wilk test for count: p-value = 5.369837893115507e-68
Reject the null hypothesis for temp: temp is not normally distributed.
Reject the null hypothesis for count: count is not normally distributed.
```

```
/usr/local/lib/python3.10/dist-packages/scipy/stats/_axis_nan_policy.py:531:
UserWarning: scipy.stats.shapiro: For N > 5000, computed p-value may not be
accurate. Current N is 10886.
  res = hypotest_fun_out(*samples, **kwds)
```

[ ]:
```
#both count and temp are not normal according to shapiro test. However as the␣
 ↪data is more than 5K, I am checking using a qq-plot
```

[45]:
```
import numpy as np
import matplotlib.pyplot as plt
import scipy.stats as stats

plt.figure(figsize=(12, 6))

plt.subplot(1, 2, 1)
stats.probplot(data['temp'], dist="norm", plot=plt)
plt.title('Q-Q Plot for temp')

plt.subplot(1, 2, 2)
stats.probplot(data['count'], dist="norm", plot=plt)
```

```python
plt.title('Q-Q Plot for count')

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



[ ]:
```python
#Again both these are not normal, so using Spearman's rank correlation.
```

[46]:
```python
import scipy.stats as stats


spearman_corr, p_value = stats.spearmanr(data['temp'], data['count'])


print(f"Spearman Correlation Coefficient: {spearman_corr}")
print(f"P-value: {p_value}")


if p_value < 0.05:
    print("The correlation is statistically significant.")
else:
    print("The correlation is not statistically significant.")
```

```
Spearman Correlation Coefficient: 0.40798939475098117
P-value: 0.0
The correlation is statistically significant.
```

[ ]:
```python
#We can see that temperature and count are positively correlated
```

[ ]:
```
#As count is not normal, using Spearmann test itself for the remaining␣
↪continuous variables
```

[47]:
```
import scipy.stats as stats


spearman_corr, p_value = stats.spearmanr(data['atemp'], data['count'])


print(f"Spearman Correlation Coefficient: {spearman_corr}")
print(f"P-value: {p_value}")


if p_value < 0.05:
    print("The correlation is statistically significant.")
else:
    print("The correlation is not statistically significant.")
```

```
Spearman Correlation Coefficient: 0.4065617539204584
P-value: 0.0
The correlation is statistically significant.
```

[ ]:
```
#From this we can see that the feeling temperature in Celsius is positively␣
↪correlated to count of users
```

[48]:
```
import scipy.stats as stats


spearman_corr, p_value = stats.spearmanr(data['humidity'], data['count'])


print(f"Spearman Correlation Coefficient: {spearman_corr}")
print(f"P-value: {p_value}")


if p_value < 0.05:
    print("The correlation is statistically significant.")
else:
    print("The correlation is not statistically significant.")
```

```
Spearman Correlation Coefficient: -0.35404912201756106
P-value: 0.0
The correlation is statistically significant.
```

[ ]:
```
#From this we can see that the humidity is negatively correlated to count of␣
↪users
```

[49]:
```python
import scipy.stats as stats


spearman_corr, p_value = stats.spearmanr(data['windspeed'], data['count'])


print(f"Spearman Correlation Coefficient: {spearman_corr}")
print(f"P-value: {p_value}")


if p_value < 0.05:
    print("The correlation is statistically significant.")
else:
    print("The correlation is not statistically significant.")
```

```
Spearman Correlation Coefficient: 0.1357773747113304
P-value: 5.9015220272171205e-46
The correlation is statistically significant.
```

[ ]:
```python
#From this we can see that the windspeed is positively correlated to count of␣
↪users
```

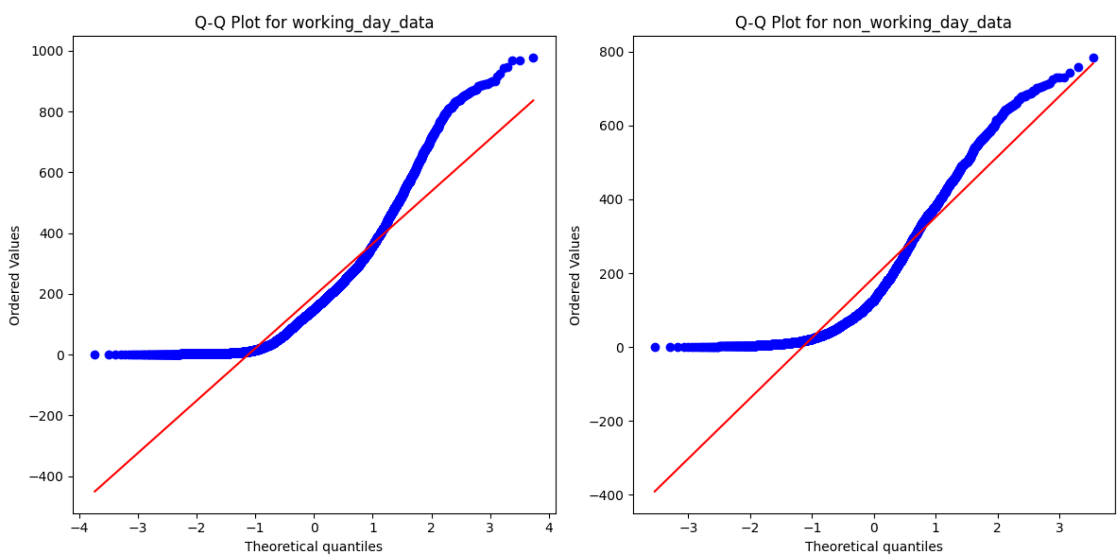
[ ]:
```python
#Hypothesis Testing:
#To check if Working Day has an effect on the number of electric cycles rented
```

[58]:
```python
data.head()
```

[58]:
```
             datetime  weather  temp   atemp  humidity  windspeed  casual  \
0 2011-01-01 00:00:00        1  9.84  14.395        81        0.0       3
1 2011-01-01 01:00:00        1  9.02  13.635        80        0.0       8
2 2011-01-01 02:00:00        1  9.02  13.635        80        0.0       5
3 2011-01-01 03:00:00        1  9.84  14.395        75        0.0       3
4 2011-01-01 04:00:00        1  9.84  14.395        75        0.0       0

   registered  count season_cat    holiday_cat    workingday_cat  hour  \
0          13     16     spring  not_a_holiday  not_a_workingday     0
1          32     40     spring  not_a_holiday  not_a_workingday     1
2          27     32     spring  not_a_holiday  not_a_workingday     2
3          10     13     spring  not_a_holiday  not_a_workingday     3
4           1      1     spring  not_a_holiday  not_a_workingday     4

   month_year
0 2011-01-01
1 2011-01-01
2 2011-01-01
3 2011-01-01
4 2011-01-01
```

[55]:
```
working_day_data = data[data['workingday_cat'] == 'workingday']['count']
non_working_day_data = data[data['workingday_cat'] ==␣
↪'not_a_workingday']['count']
```

[59]:
```
#Plotting a qq plot to check if the data is normal

import numpy as np
import matplotlib.pyplot as plt
import scipy.stats as stats

plt.figure(figsize=(12, 6))

plt.subplot(1, 2, 1)
stats.probplot(working_day_data, dist="norm", plot=plt)
plt.title('Q-Q Plot for working_day_data')

plt.subplot(1, 2, 2)
stats.probplot(non_working_day_data, dist="norm", plot=plt)
plt.title('Q-Q Plot for non_working_day_data')

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



[ ]:
```
#From these plots we can see that these datasets are not normal. So, I will␣
↪plot a 2 sample t-test and a K-W test as well to check if these
#are significantly differnt or same.
```

[60]:
```python
from scipy.stats import ttest_ind

t_stat, p_value = ttest_ind(working_day_data, non_working_day_data,␣
↪equal_var=False)

print(f"T-Statistic: {t_stat}")
print(f"P-Value: {p_value}")

if p_value < 0.05:
    print("The difference in means is statistically significant.")
else:
    print("The difference in means is not statistically significant.")
```

```
T-Statistic: 1.2362580418223226
P-Value: 0.21640312280695098
The difference in means is not statistically significant.
```

[ ]:
```python
#t-test says these 2 sets do not have a significant different means, or these␣
↪are not statistically different.
```

[ ]:
```python
#However, let me do a K-W test to check if we come to the same conclusion
```

[61]:
```python
from scipy.stats import kruskal

k_stat, p_value = kruskal(working_day_data, non_working_day_data)

print(f"Kruskal-Wallis Statistic: {k_stat}")
print(f"P-Value: {p_value}")

if p_value < 0.05:
    print("The difference between the distributions is statistically␣
↪significant.")
else:
    print("The difference between the distributions is not statistically␣
↪significant.")
```

```
Kruskal-Wallis Statistic: 0.0016182887191034687
P-Value: 0.9679113872727798
The difference between the distributions is not statistically significant.
```

[ ]:
```python
#Even a K-W test between these two comes to the same conclusion. So, these 2␣
↪sets are not different statistically.
#Therefore we can conclude that Working Day has NO effect on the number of␣
↪electric cycles rented.
```

[ ]:
```python
# ANNOVA to check if No. of cycles rented is similar or different in different
# 1. weather
```

```
# 2. season
```

[63]:
```
#Extracting the count data for each weather
import pandas as pd

weather_groups = data.groupby('weather')['count'].apply(list)

weather_1 = weather_groups[1]
weather_2 = weather_groups[2]
weather_3 = weather_groups[3]
weather_4 = weather_groups[4]
```

```
<ipython-input-63-80ce8d67a68d>:3: FutureWarning: The default of observed=False
is deprecated and will be changed to True in a future version of pandas. Pass
observed=False to retain current behavior or observed=True to adopt the future
default and silence this warning.
  weather_groups = data.groupby('weather')['count'].apply(list)
```

[69]:
```
from scipy.stats import f_oneway
anova_result = f_oneway(weather_1, weather_2, weather_3, weather_4)

print("ANOVA Test Result:")
print(f"F-statistic: {anova_result.statistic}")
print(f"P-value: {anova_result.pvalue}")
```

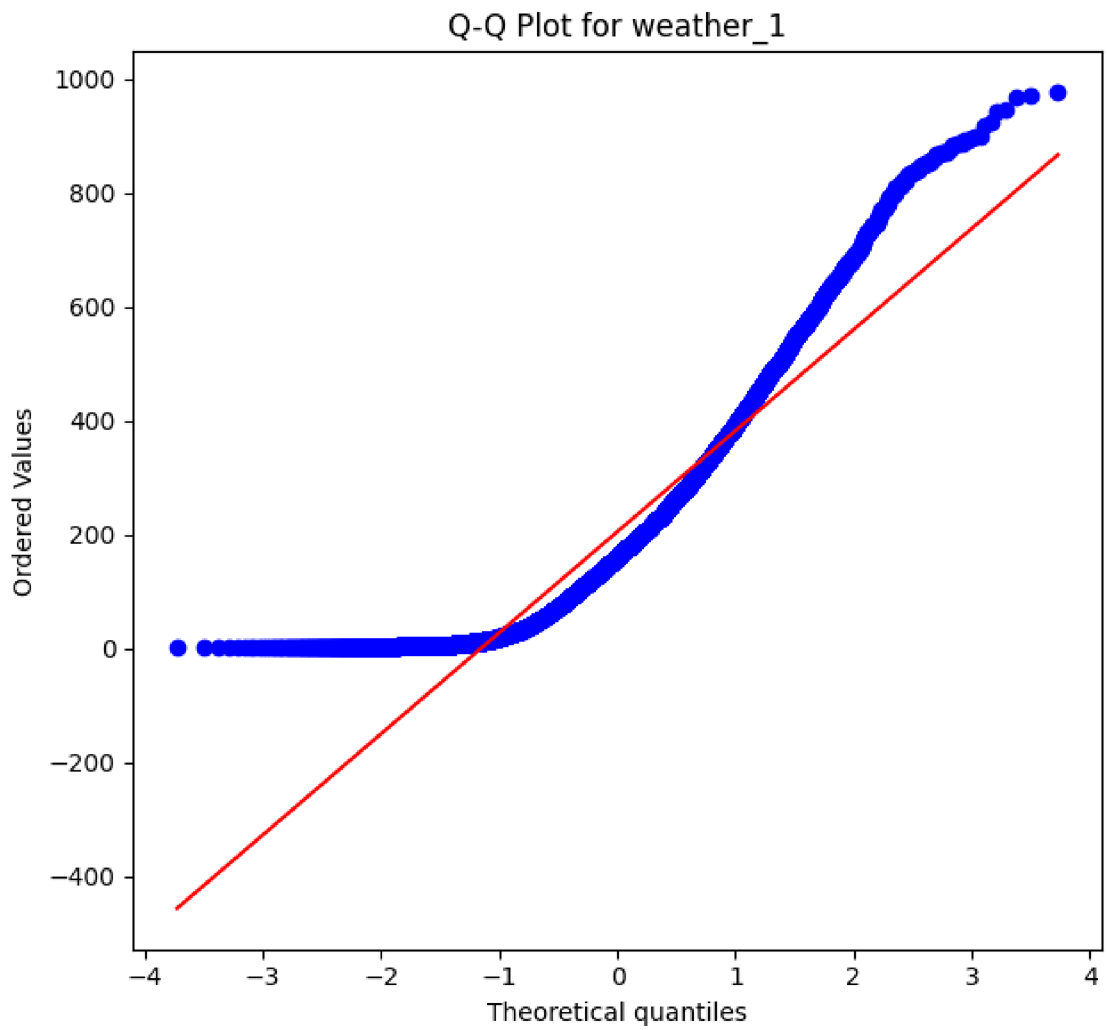
```
ANOVA Test Result:
F-statistic: 65.53024112793271
P-value: 5.482069475935669e-42
```

[ ]:
```
#As the p-value is very less this implies that at least one weather category's␣
↪mean count differs significantly from the others.
```

[ ]:
```
#Let me check if these data are normal
```

[70]:
```
#Plotting a qq plot to check if the data is normal

import numpy as np
import matplotlib.pyplot as plt
import scipy.stats as stats

plt.figure(figsize=(12, 6))

plt.subplot(1, 2, 1)
stats.probplot(weather_1, dist="norm", plot=plt)
plt.title('Q-Q Plot for weather_1')

plt.tight_layout()
```

```
plt.show()
```



[ ]:
```
#This shows that weather1 is not normal. Since one of them is not normal, I am
↪not checking for the others.
```

[ ]:
```
#Let me again run a K-W test and see if we can come to the same conclusion
```

[71]:
```
from scipy.stats import kruskal

k_stat, p_value = kruskal(weather_1, weather_2, weather_3, weather_4)

print(f"Kruskal-Wallis Statistic: {k_stat}")
print(f"P-Value: {p_value}")

if p_value < 0.05:
```

```
    print("The difference between the distributions is statistically␣
↪significant.")
else:
    print("The difference between the distributions is not statistically␣
↪significant.")
```

```
Kruskal-Wallis Statistic: 205.00216514479087
P-Value: 3.501611300708679e-44
The difference between the distributions is statistically significant.
```

[ ]:
```
#Even with K-W test we come to the same conclusion, i.e. that at least one␣
↪weather category's mean count differs significantly from the others.
```

[ ]:
```
#2. Season
```

[73]:
```
season_groups = data.groupby('season_cat')['count'].apply(list)


spring_counts = season_groups['spring']
fall_counts = season_groups['fall']
winter_counts = season_groups['winter']
summer_counts = season_groups['summer']
```

```
<ipython-input-73-8ff7ab73009a>:1: FutureWarning: The default of observed=False
is deprecated and will be changed to True in a future version of pandas. Pass
observed=False to retain current behavior or observed=True to adopt the future
default and silence this warning.
  season_groups = data.groupby('season_cat')['count'].apply(list)
```

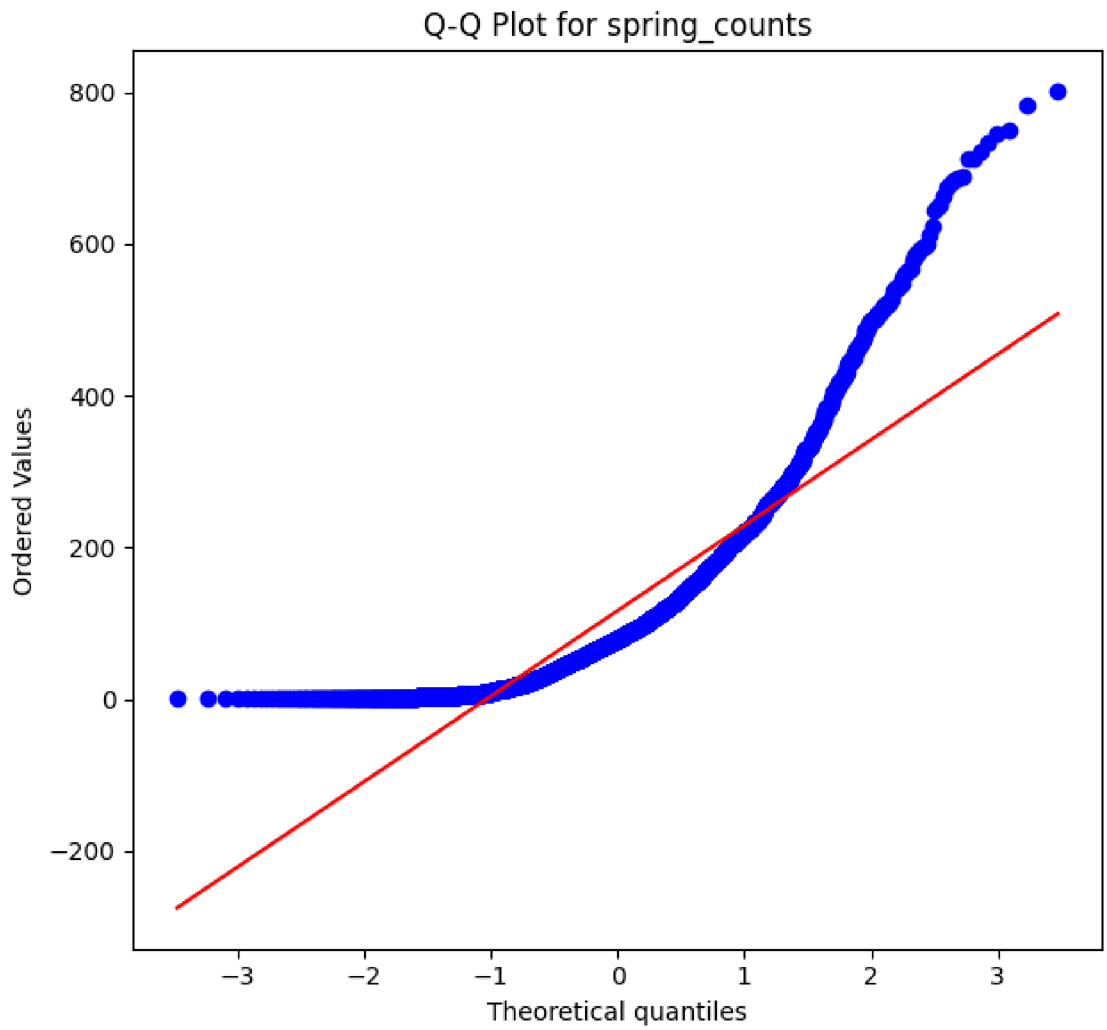
[75]:
```
from scipy.stats import f_oneway

anova_result = f_oneway(spring_counts, fall_counts, winter_counts,␣
↪summer_counts)

print("ANOVA Test Result:")
print(f"F-statistic: {anova_result.statistic}")
print(f"P-value: {anova_result.pvalue}")
```

```
ANOVA Test Result:
F-statistic: 236.94671081032106
P-value: 6.164843386499654e-149
```

[ ]:
```
#Again the very low p-value suggests that at least one season's mean count␣
↪differs significantly from the others.
```

[76]:
```
#Plotting a qq plot to check if the data is normal
```

```
import numpy as np
import matplotlib.pyplot as plt
import scipy.stats as stats

plt.figure(figsize=(12, 6))

plt.subplot(1, 2, 1)
stats.probplot(spring_counts, dist="norm", plot=plt)
plt.title('Q-Q Plot for spring_counts')

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



[77]:
```
#This shows that spring_counts is not normal. Since one of them is not normal,␣
↪I am not checking for the others.
```

[ ]:
```
#Let me again run a K-W test and see if we can come to the same conclusion
```

[78]:
```
from scipy.stats import kruskal
kw_result = kruskal(spring_counts, fall_counts, winter_counts, summer_counts)

print("Kruskal-Wallis Test Result:")
print(f"H-statistic: {kw_result.statistic}")
print(f"P-value: {kw_result.pvalue}")
```

```
Kruskal-Wallis Test Result:
H-statistic: 699.6668548181988
P-value: 2.479008372608633e-151
```

[ ]:
```
#Even a K-W test gives a low p-value indicating that at least one season's mean␣
↪count differs significantly from the others.
```

[ ]:
```
# Chi-square test to check if Weather is dependent on the season
```

[80]:
```
import pandas as pd
from scipy.stats import chi2_contingency


contingency_table = pd.crosstab(data['weather'], data['season_cat'])


chi2_stat, p_value, dof, expected = chi2_contingency(contingency_table)

print(f"Chi-Square Statistic: {chi2_stat}")
print(f"P-value: {p_value}")
print(f"Degrees of Freedom: {dof}")
print(f"Expected Frequencies Table:\n{expected}")

if p_value < 0.05:
    print("The result is significant. Weather and Season are dependent.")
else:
    print("The result is not significant. Weather and Season are independent.")
```

```
Chi-Square Statistic: 49.15865559689363
P-value: 1.5499250736864862e-07
Degrees of Freedom: 9
Expected Frequencies Table:
[[1.80559765e+03 1.77454639e+03 1.80559765e+03 1.80625831e+03]
 [7.11493845e+02 6.99258130e+02 7.11493845e+02 7.11754180e+02]
 [2.15657450e+02 2.11948742e+02 2.15657450e+02 2.15736359e+02]
 [2.51056403e-01 2.46738931e-01 2.51056403e-01 2.51148264e-01]]
The result is significant. Weather and Season are dependent.
```

[ ]:

```
#The low p-value of Chi-square test confirms that weather is dependent on the␣
↪season
```