Copy of yulu_buisiness_case_scaler

November 27, 2024

[ ]:
```
!gdown https://d2beiqkhq929f0.cloudfront.net/public_assets/assets/000/001/428/
↪original/bike_sharing.csv
```

```
Downloading…
From: https://d2beiqkhq929f0.cloudfront.net/public_assets/assets/000/001/428/ori
ginal/bike_sharing.csv
To: /content/bike_sharing.csv
   0% 0.00/648k [00:00<?, ?B/s] 100% 648k/648k [00:00<00:00, 10.5MB/s]
```

[ ]:
```
import pandas as pd
import numpy as np

df = pd.read_csv('/content/bike_sharing.csv')
df.head()
```

[ ]:

	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	0.0	3	13	16
1	2011-01-01 01:00:00	1	0	0	1	9.02	13.635	80	0.0	8	32	40
2	2011-01-01 02:00:00	1	0	0	1	9.02	13.635	80	0.0	5	27	32
3	2011-01-01 03:00:00	1	0	0	1	9.84	14.395	75	0.0	3	10	13
4	2011-01-01 04:00:00	1	0	0	1	9.84	14.395	75	0.0	0	1	1

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

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

[ ]:
```
df.describe()
```

[ ]:

	season	holiday	workingday	weather	temp
count	10886.000000	10886.000000	10886.000000	10886.000000	10886.00000
mean	2.506614	0.028569	0.680875	1.418427	20.23086

```
std        1.116174      0.166599      0.466159      0.633839      7.79159
min        1.000000      0.000000      0.000000      1.000000      0.82000
25%        2.000000      0.000000      0.000000      1.000000     13.94000
50%        3.000000      0.000000      1.000000      1.000000     20.50000
75%        4.000000      0.000000      1.000000      2.000000     26.24000
max        4.000000      1.000000      1.000000      4.000000     41.00000

              atemp      humidity     windspeed        casual    registered  \
count  10886.000000  10886.000000  10886.000000  10886.000000  10886.000000
mean      23.655084     61.886460     12.799395     36.021955    155.552177
std        8.474601     19.245033      8.164537     49.960477    151.039033
min        0.760000      0.000000      0.000000      0.000000      0.000000
25%       16.665000     47.000000      7.001500      4.000000     36.000000
50%       24.240000     62.000000     12.998000     17.000000    118.000000
75%       31.060000     77.000000     16.997900     49.000000    222.000000
max       45.455000    100.000000     56.996900    367.000000    886.000000

              count
count  10886.000000
mean     191.574132
std      181.144454
min        1.000000
25%       42.000000
50%      145.000000
75%      284.000000
max      977.000000
```

[ ]:
```
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
```

[ ]:
```
df.isna().sum().sum()
```

[ ]:
```
0
```

[ ]:
```
df.value_counts()
```

[ ]:
```
datetime             season  holiday  workingday  weather  temp   atemp
humidity  windspeed  casual  registered  count
2011-01-01 00:00:00  1       0        0           1        9.84   14.395  81
0.0000    3          13        16          1
2012-05-01 21:00:00  2       0        1           1        26.24  30.305  65
8.9981    31         251       282         1
2012-05-01 13:00:00  2       0        1           2        29.52  33.335  51
15.0013   41         208       249         1
2012-05-01 14:00:00  2       0        1           2        30.34  33.335  48
16.9979   37         167       204         1
2012-05-01 15:00:00  2       0        1           2        30.34  33.335  45
15.0013   48         186       234         1
                                                 ..
2011-09-02 04:00:00  3       0        1           1        24.60  28.030  83
6.0032    2          2         4           1
2011-09-02 05:00:00  3       0        1           2        24.60  28.030  83
8.9981    0          20        20          1
2011-09-02 06:00:00  3       0        1           1        24.60  28.030  83
8.9981    3          73        76          1
2011-09-02 07:00:00  3       0        1           1        24.60  28.030  83
7.0015    6          253       259         1
2012-12-19 23:00:00  4       0        1           1        13.12  16.665  66
8.9981    4          84        88          1
Name: count, Length: 10886, dtype: int64
```

[ ]:
```
df.duplicated().value_counts()
```

[ ]:
```
False    10886
Name: count, dtype: int64
```

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

[ ]:
```
              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
```

[ ]:
```
# seperate the datetime into date and time

df['datetime'] = pd.to_datetime(df['datetime'])
df['date'] = df.datetime.dt.date
df['time'] = df.datetime.dt.time
df.drop('datetime',axis=1, inplace=True)


df.head()
```

[ ]:
```
   season  holiday  workingday  weather  temp   atemp  humidity  windspeed  \
0       1        0           0        1  9.84  14.395        81        0.0
1       1        0           0        1  9.02  13.635        80        0.0
2       1        0           0        1  9.02  13.635        80        0.0
3       1        0           0        1  9.84  14.395        75        0.0
4       1        0           0        1  9.84  14.395        75        0.0

   casual  registered  count        date      time
0       3          13     16  2011-01-01  00:00:00
1       8          32     40  2011-01-01  01:00:00
2       5          27     32  2011-01-01  02:00:00
3       3          10     13  2011-01-01  03:00:00
4       0           1      1  2011-01-01  04:00:00
```

[ ]:
```
df['date'].max(),df['date'].min()
# date ranges from 1st jan 2011 to 19th december 2012
```

[ ]:
```
(datetime.date(2012, 12, 19), datetime.date(2011, 1, 1))
```

[ ]:
```
# eda
season_mapping = {
    1:"Spring",
    2:"Summer",
    3:"Fall",
    4:"Winter",
}


df['season_name'] = df['season'].apply(lambda x: season_mapping[x])
```

```
weather_mapping = {
    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"
}
df['weather_desc'] = df['weather'].apply(lambda x: weather_mapping[x])
```

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

[ ]:

	season	holiday	workingday	weather	temp	atemp	humidity	windspeed	\
0	1	0	0	1	9.84	14.395	81	0.0	
1	1	0	0	1	9.02	13.635	80	0.0	
2	1	0	0	1	9.02	13.635	80	0.0	
3	1	0	0	1	9.84	14.395	75	0.0	
4	1	0	0	1	9.84	14.395	75	0.0	

	casual	registered	count	date	time	season_name	\
0	3	13	16	2011-01-01	00:00:00	Spring	
1	8	32	40	2011-01-01	01:00:00	Spring	
2	5	27	32	2011-01-01	02:00:00	Spring	
3	3	10	13	2011-01-01	03:00:00	Spring	
4	0	1	1	2011-01-01	04:00:00	Spring	

	weather_desc
0	Clear, Few clouds, partly cloudy, partly cloudy
1	Clear, Few clouds, partly cloudy, partly cloudy
2	Clear, Few clouds, partly cloudy, partly cloudy
3	Clear, Few clouds, partly cloudy, partly cloudy
4	Clear, Few clouds, partly cloudy, partly cloudy

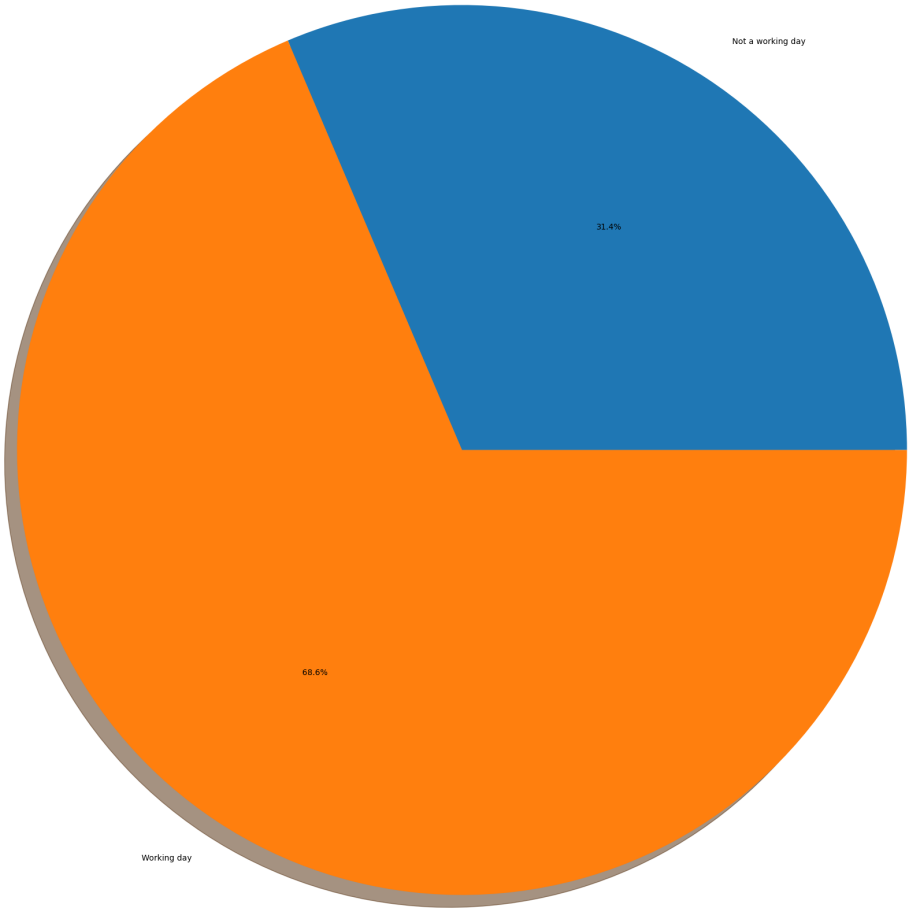
[ ]:
```
# count vs working day

import matplotlib.pyplot as plt
import seaborn as sns


working_count = df.groupby('workingday')['count'].sum()
working_count = working_count.reset_index()
```

[ ]:
```
working_count
```

[ ]:

	workingday	count
0	0	654872
1	1	1430604

[ ]:
```
plt.pie(x=working_count['count'], labels=['Not a working day','Working day'], ␣
↪autopct='%1.1f%%',shadow=True,);
```



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

[ ]:

	season	holiday	workingday	weather	temp	atemp	humidity	windspeed	\
0	1	0	0	1	9.84	14.395	81	0.0	
1	1	0	0	1	9.02	13.635	80	0.0	
2	1	0	0	1	9.02	13.635	80	0.0	
3	1	0	0	1	9.84	14.395	75	0.0	
4	1	0	0	1	9.84	14.395	75	0.0	

	casual	registered	count	date	time	season_name	\

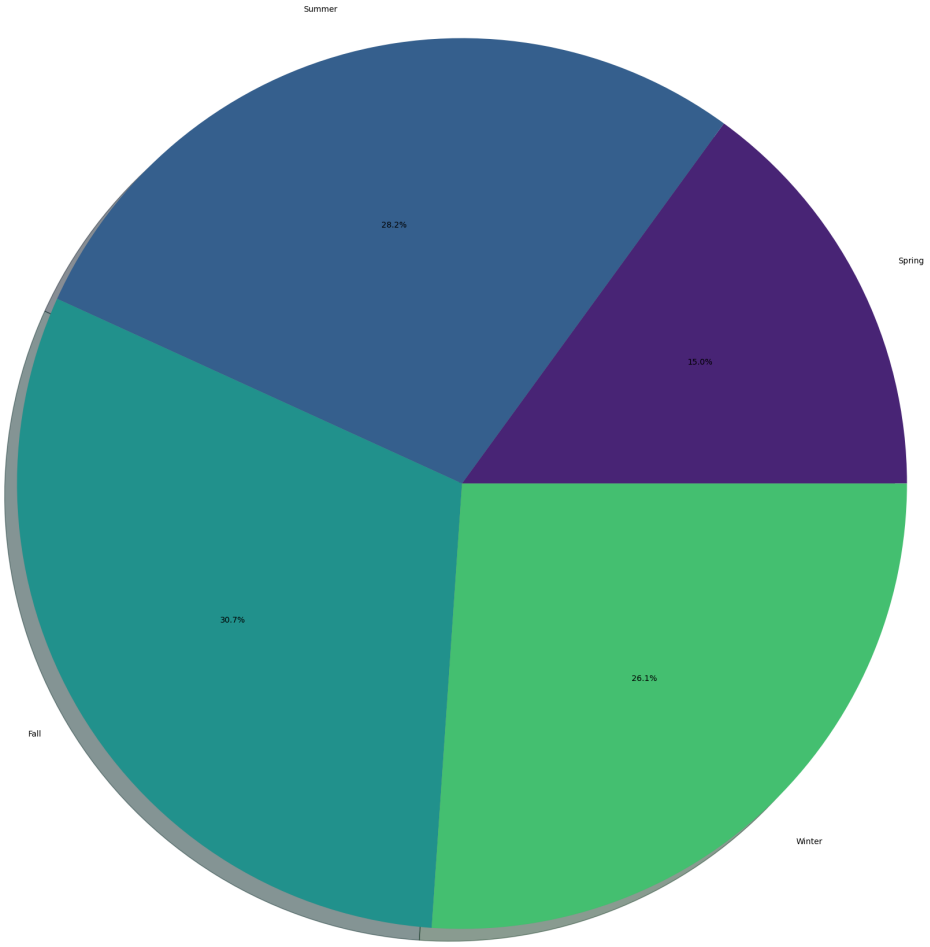
```
0       3          13     16  2011-01-01  00:00:00      Spring
1       8          32     40  2011-01-01  01:00:00      Spring
2       5          27     32  2011-01-01  02:00:00      Spring
3       3          10     13  2011-01-01  03:00:00      Spring
4       0           1      1  2011-01-01  04:00:00      Spring

                                      weather_desc
0  Clear, Few clouds, partly cloudy, partly cloudy
1  Clear, Few clouds, partly cloudy, partly cloudy
2  Clear, Few clouds, partly cloudy, partly cloudy
3  Clear, Few clouds, partly cloudy, partly cloudy
4  Clear, Few clouds, partly cloudy, partly cloudy
```

[ ]:
```
# season vs count
season_count = df.groupby(['season', 'season_name'])['count'].sum()
season_count = season_count.reset_index()
season_count
```

[ ]:
```
   season season_name   count
0       1      Spring  312498
1       2      Summer  588282
2       3        Fall  640662
3       4      Winter  544034
```

[ ]:
```
from matplotlib import cm
cmap = cm.get_cmap('viridis')
colors = cmap([0.1, 0.3, 0.5, 0.7])
plt.pie(x=season_count['count'], labels=season_count['season_name'], ␣
 ↪autopct='%1.1f%%',shadow=True,colors=colors);
```

```
<ipython-input-73-e200b01193e8>:2: MatplotlibDeprecationWarning: The get_cmap
function was deprecated in Matplotlib 3.7 and will be removed two minor releases
later. Use ``matplotlib.colormaps[name]`` or
``matplotlib.colormaps.get_cmap(obj)`` instead.
  cmap = cm.get_cmap('viridis')
```

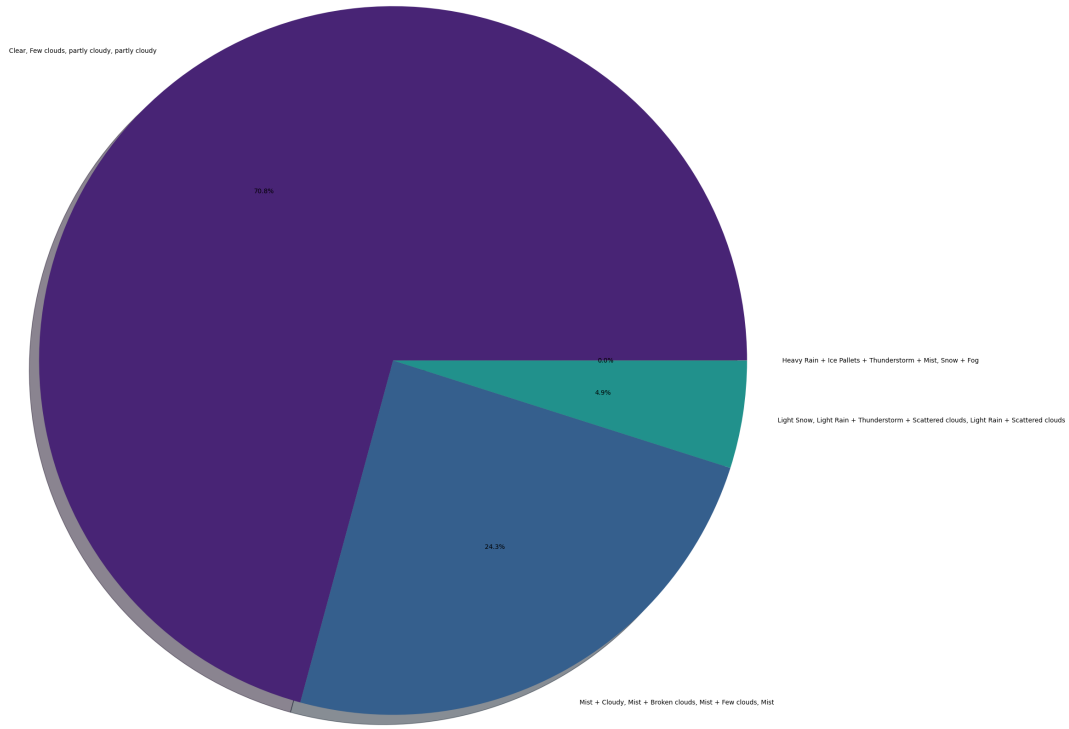


[ ]:
```
# weather vs count
weather_count = df.groupby(['weather', 'weather_desc'])['count'].sum()
weather_count = weather_count.reset_index()
weather_count
```

[ ]:

	weather	weather_desc	count
0	1	Clear, Few clouds, partly cloudy, partly cloudy	1476063
1	2	Mist + Cloudy, Mist + Broken clouds, Mist + Fe…	507160
2	3	Light Snow, Light Rain + Thunderstorm + Scatte…	102089
3	4	Heavy Rain + Ice Pallets + Thunderstorm + Mist…	164

[ ]:
```
plt.pie(x=weather_count['count'], labels=weather_count['weather_desc'], ␣
↪autopct='%1.1f%%',shadow=True,colors=colors);
```



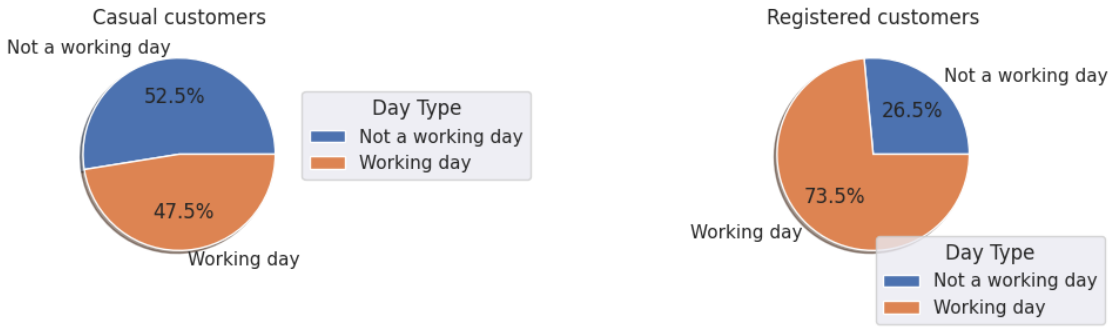
[ ]:
```
# all the above factors with registered and casual customers
working_count2 = df.groupby('workingday')[['casual', 'registered']].sum()
working_count2 = working_count2.reset_index()
working_count2
```

[ ]:

	workingday	casual	registered
0	0	206037	448835
1	1	186098	1244506

[ ]:
```
labels = ['Not a working day', 'Working day']

plt.figure(figsize=(10, 10))

plt.subplot(1, 2, 1)
plt.title("Casual customers")
pie1 = plt.pie(x=working_count2['casual'], labels=labels, autopct='%1.1f%%',␣
 ↪shadow=True)
plt.legend(pie1[0], labels, title="Day Type", loc="upper right",␣
 ↪bbox_to_anchor=(2.0, 0.8))
```

```
plt.subplot(1, 2, 2)
plt.title("Registered customers")
pie2 = plt.pie(x=working_count2['registered'], labels=labels, autopct='%1.
  ↪1f%%', shadow=True)
plt.legend(pie2[0], labels, title="Day Type", loc="upper right",␣
  ↪bbox_to_anchor=(1.5, 0.2))

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



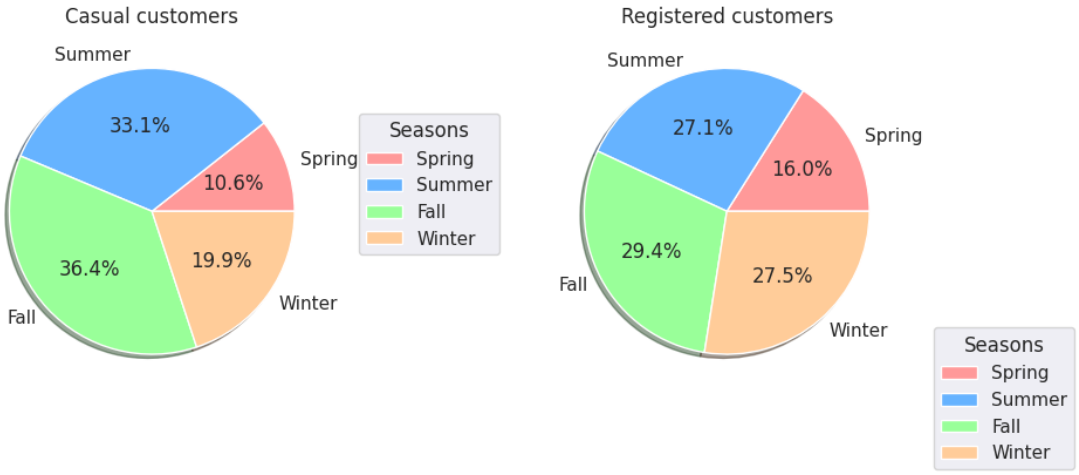
[ ]:
```
season_count2 = df.groupby(['season', 'season_name'])[['casual', 'registered']].
  ↪sum()
season_count2 = season_count2.reset_index()
season_count2
```

[ ]:

	season	season_name	casual	registered
0	1	Spring	41605	270893
1	2	Summer	129672	458610
2	3	Fall	142718	497944
3	4	Winter	78140	465894

[ ]:
```
colors = ['#ff9999', '#66b3ff', '#99ff99', '#ffcc99']

plt.figure(figsize=(10, 10))


plt.subplot(1, 2, 1)
plt.title("Casual customers")
pie1 = plt.pie(
    x=season_count2['casual'],
    labels=season_count2['season_name'],
    autopct='%1.1f%%',
    shadow=True,
```

```
    colors=colors
)
plt.legend(pie1[0], season_count2['season_name'], title="Seasons", loc="upper␣
 ↪right", bbox_to_anchor=(1.5, 0.8))


plt.subplot(1, 2, 2)
plt.title("Registered customers")
pie2 = plt.pie(
    x=season_count2['registered'],
    labels=season_count2['season_name'],
    autopct='%1.1f%%',
    shadow=True,
    colors=colors
)
plt.legend(pie2[0], season_count2['season_name'], title="Seasons", loc="upper␣
 ↪right", bbox_to_anchor=(1.5, 0.2))

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



[ ]:
```
weather_count2 = df.groupby(['weather', 'weather_desc'])[['casual',␣
 ↪'registered']].sum()
weather_count2 = weather_count2.reset_index()
weather_count2
```

[ ]:

	weather	weather_desc	casual	\
0	1	Clear, Few clouds, partly cloudy, partly cloudy	289900	
1	2	Mist + Cloudy, Mist + Broken clouds, Mist + Fe…	87246	
2	3	Light Snow, Light Rain + Thunderstorm + Scatte…	14983	

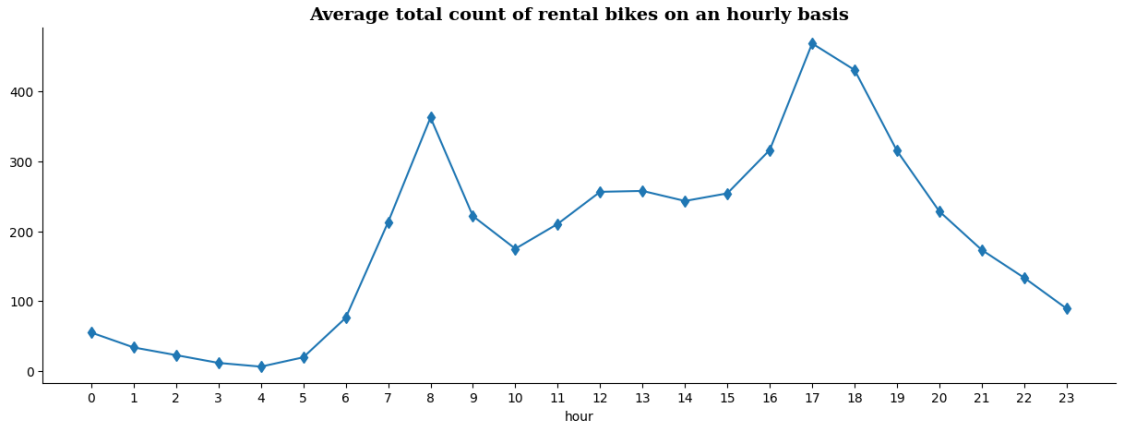
```
3        4  Heavy Rain + Ice Pallets + Thunderstorm + Mist…       6
```

```
   registered
0     1186163
1      419914
2       87106
3         158
```

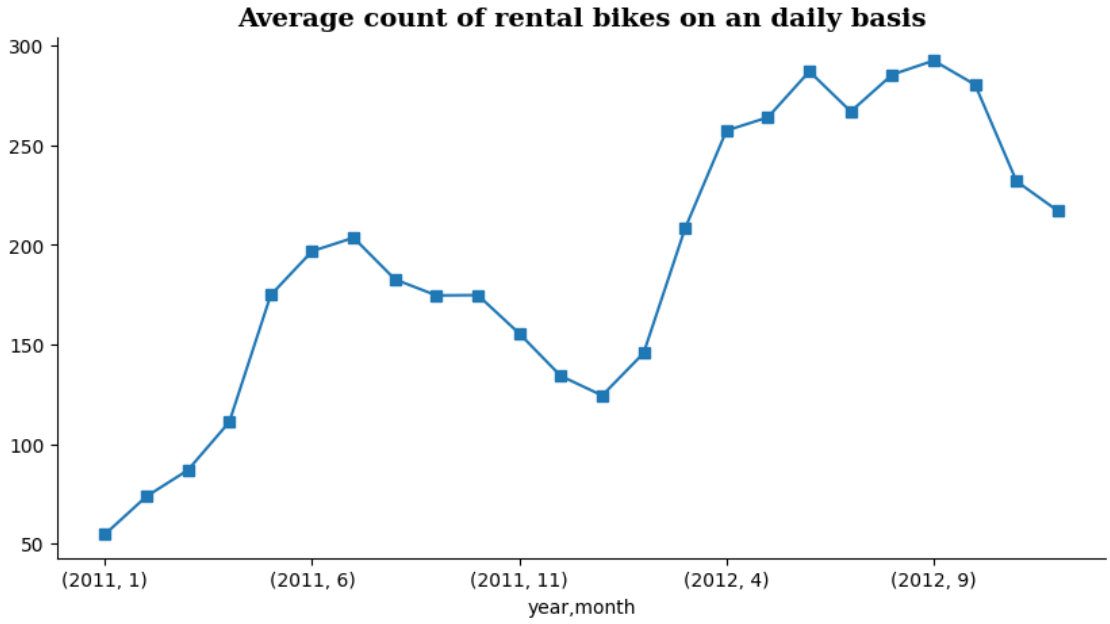
[ ]:
```
plt.figure(figsize=(20,10))
plt.subplot(2,1,1)
plt.title("Casual customers")
plt.pie(x=weather_count2['casual'], labels=weather_count2['weather_desc'], ␣
↪autopct='%1.1f%%',shadow=True,colors=colors);
plt.subplot(2,1,2)
plt.title("Registered customers")
plt.pie(x=weather_count2['registered'], labels=weather_count2['weather_desc'], ␣
↪autopct='%1.1f%%',shadow=True,colors=colors);
plt.show()
```

[ ]:
```
df['hour'] = pd.to_datetime(df['time'], format='%H:%M:%S').dt.hour

plt.figure(figsize = (15,5))
plt.title("Average total count of rental bikes on an hourly basis"
```

```
          ,fontsize=14,fontfamily='serif',fontweight='bold')
df.groupby('hour')['count'].mean().plot(kind = 'line', marker = 'd')
plt.xticks(np.arange(0, 24))
sns.despine()
plt.show()
```

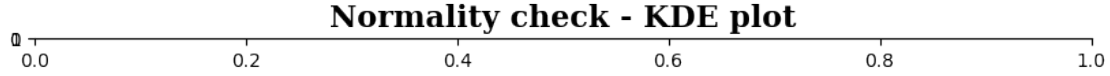


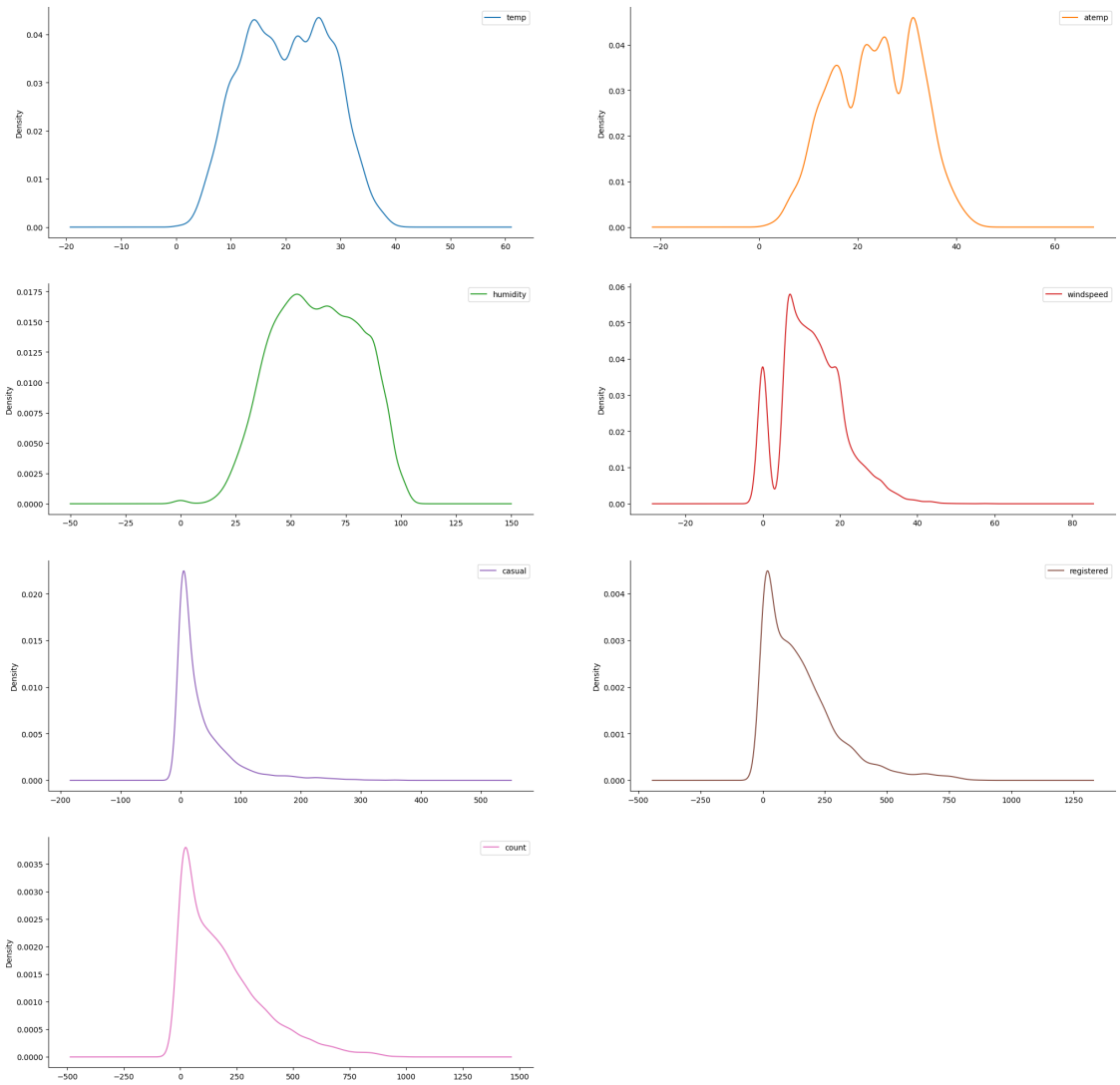
[ ]:
```
df['month'] = pd.to_datetime(df['date']).dt.month
df['year'] = pd.to_datetime(df['date']).dt.year

plt.figure(figsize = (10,5))
plt.title("Average count of rental bikes on an daily basis"
          ,fontsize=14,fontfamily='serif',fontweight='bold')
df.groupby(['year','month'])['count'].mean().plot(kind = 'line', marker = 's')
sns.despine()
plt.show()
```



[ ]:
```
# check if data is normlly distributed or not
cols = df[["temp" ,        "atemp" ,        "humidity" ,        "windspeed"␣
 ↪,        "casual" ,        "registered" ,        "count"]]
plt.figure(figsize=(10,0))
plt.title('Normality check - KDE␣
 ↪plot',fontsize=16,fontfamily='serif',fontweight='bold')
plt.rcParams['figure.figsize'] = [25, 25]
cols.plot(kind='density', subplots=True, layout=(4,2), sharex=False)
sns.despine()
plt.show()
```

```
/usr/local/lib/python3.10/dist-packages/matplotlib/_tight_bbox.py:67:
RuntimeWarning: divide by zero encountered in scalar divide
  fig.patch.set_bounds(x0 / w1, y0 / h1,
/usr/local/lib/python3.10/dist-packages/matplotlib/_tight_bbox.py:68:
RuntimeWarning: divide by zero encountered in scalar divide
  fig.bbox.width / w1, fig.bbox.height / h1)
/usr/local/lib/python3.10/dist-packages/matplotlib/patches.py:743:
RuntimeWarning: invalid value encountered in scalar add
  y1 = self.convert_yunits(self._y0 + self._height)
/usr/local/lib/python3.10/dist-packages/matplotlib/transforms.py:2039:
RuntimeWarning: invalid value encountered in scalar add
  self._mtx[1, 2] += ty
```





[ ]:
```
# Weekday vs count
# Null Hypothesis : The count on working day is the same as the count on non working days (mean1 == mean2)
# Alternate Hypothesis : The count on working day is greater than the count on non working days (mean1 > mean2)

# This is a one tailed T-Test since we are comparing categorical and a numerical column and there are exactly 2 categories in the categorial column


working = df[df['workingday'] == 1]['count']
```

```
non_working = df[df['workingday'] == 0]['count']

print(f"length working {len(working)} length non working {len(non_working)}")
working = working.sample(min(len(working), len(non_working)))
non_working = non_working.sample(min(len(working), len(non_working)))

print(f"length working {len(working)} length non working {len(non_working)}")
print(f"mean working {working.mean()} mean non working {non_working.mean()}")
```

```
length working 7412 length non working 3474
length working 3474 length non working 3474
mean working 194.3215313759355 mean non working 188.50662061024755
```

[ ]:
```
df['month_name'] = df['month'].replace({1: 'January',
                                        2: 'February',
                                        3: 'March',
                                        4: 'April',
                                        5: 'May',
                                        6: 'June',
                                        7: 'July',
                                        8: 'August',
                                        9: 'September',
                                        10: 'October',
                                        11: 'November',
                                        12: 'December'})
```

[ ]:
```
df_month_count = df.groupby(['month', 'month_name'])['count'].sum().
  ↪reset_index()
df_month_count = df_month_count.sort_values(by='count', ascending=False)
df_month_count
```

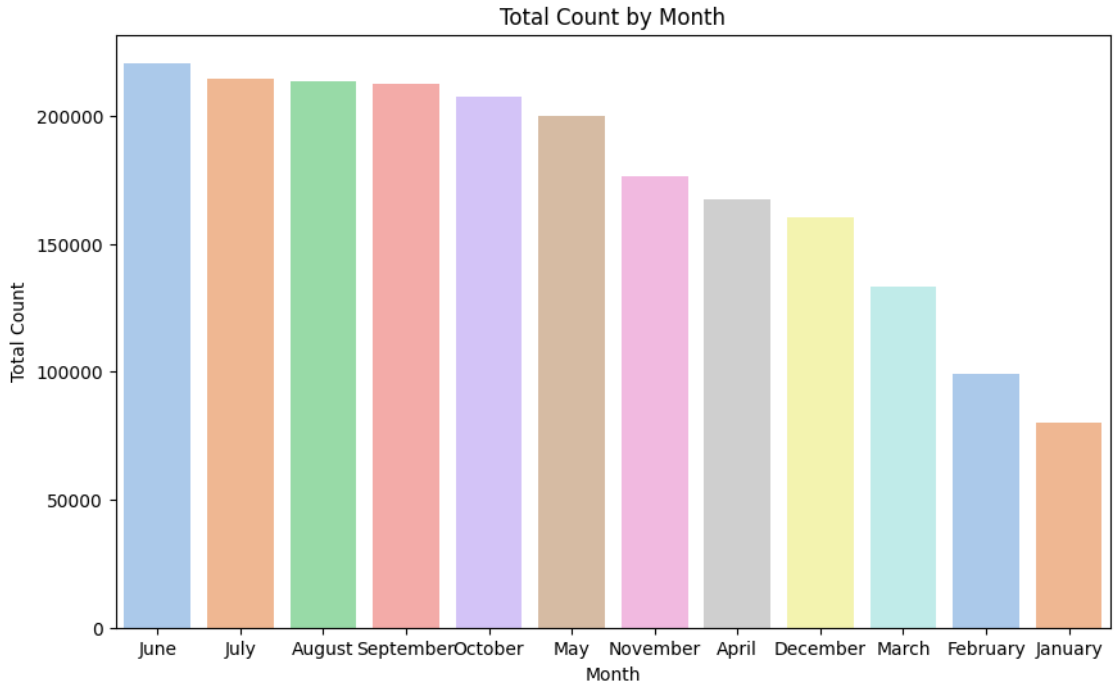
[ ]:

	month	month_name	count
5	6	June	220733
6	7	July	214617
7	8	August	213516
8	9	September	212529
9	10	October	207434
4	5	May	200147
10	11	November	176440
3	4	April	167402
11	12	December	160160
2	3	March	133501
1	2	February	99113
0	1	January	79884

[ ]:
```
plt.figure(figsize=(10, 6))
sns.barplot(x='month_name', y='count', data=df_month_count, palette='pastel')
```

```
plt.title('Total Count by Month')
plt.xlabel('Month')
plt.ylabel('Total Count')
plt.show()
```

```
<ipython-input-88-8e4c16647d2e>:2: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same
effect.

  sns.barplot(x='month_name', y='count', data=df_month_count, palette='pastel')
```



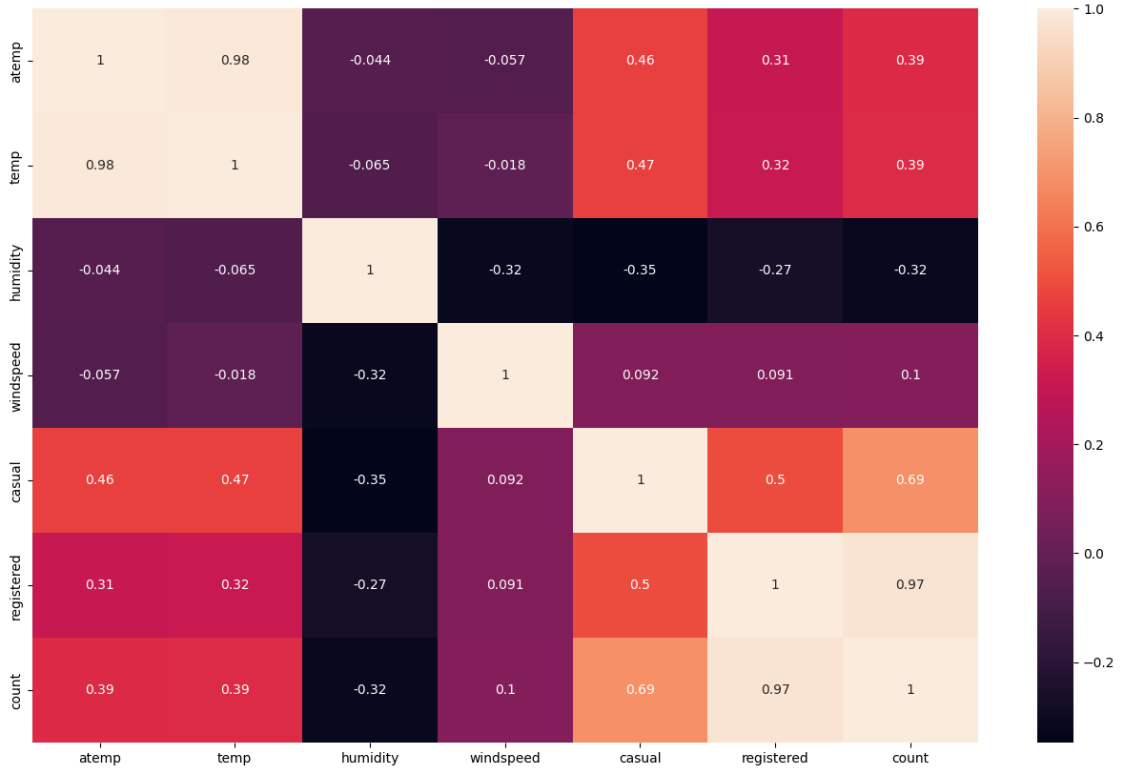
[ ]:
```
correlation_matrix = df[["atemp", "temp", "humidity", "windspeed", "casual",␣
↪"registered", "count"]].corr()
correlation_df = pd.DataFrame(correlation_matrix)
correlation_df
```

[ ]:

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

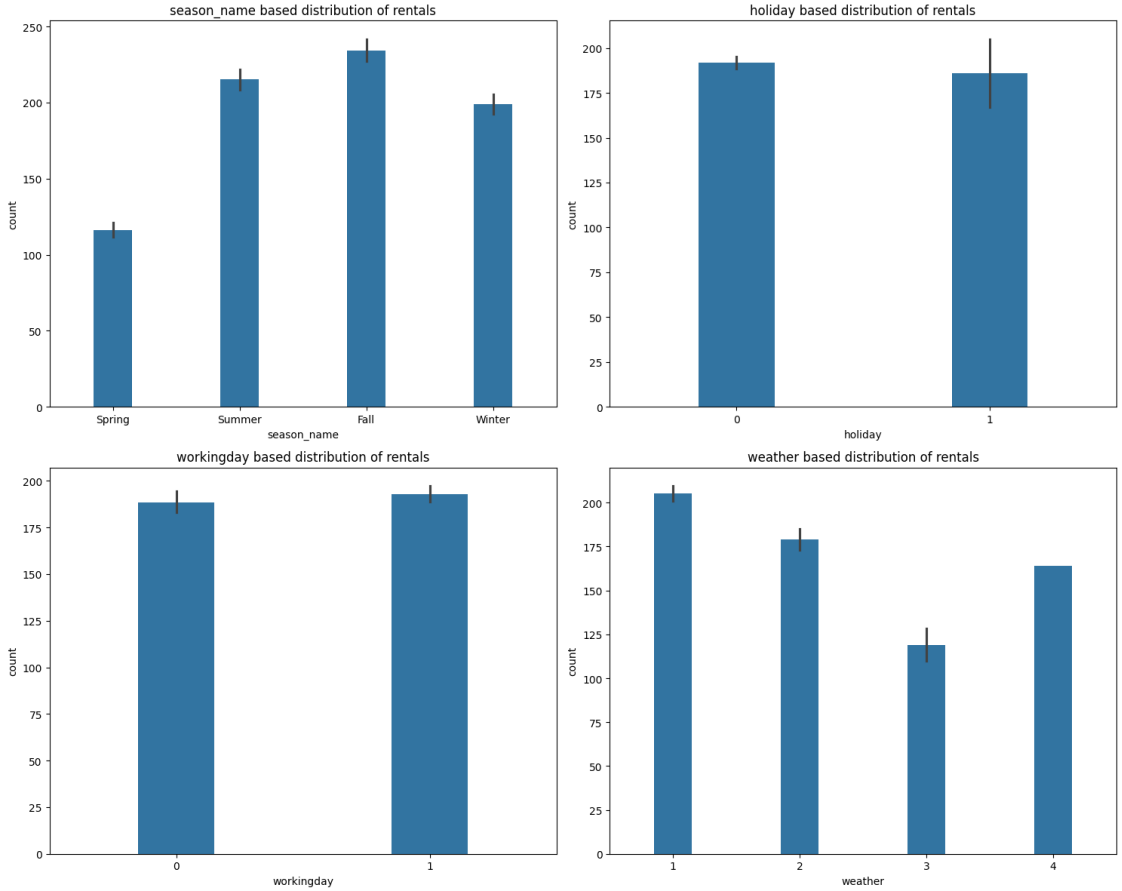
```
registered  0.314635  0.318571 -0.265458   0.091052  0.497250    1.000000
count       0.389784  0.394454 -0.317371   0.101369  0.690414    0.970948

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

[ ]:
```
plt.figure(figsize = (16, 10))
sns.heatmap(correlation_matrix, annot = True)
plt.show()
```

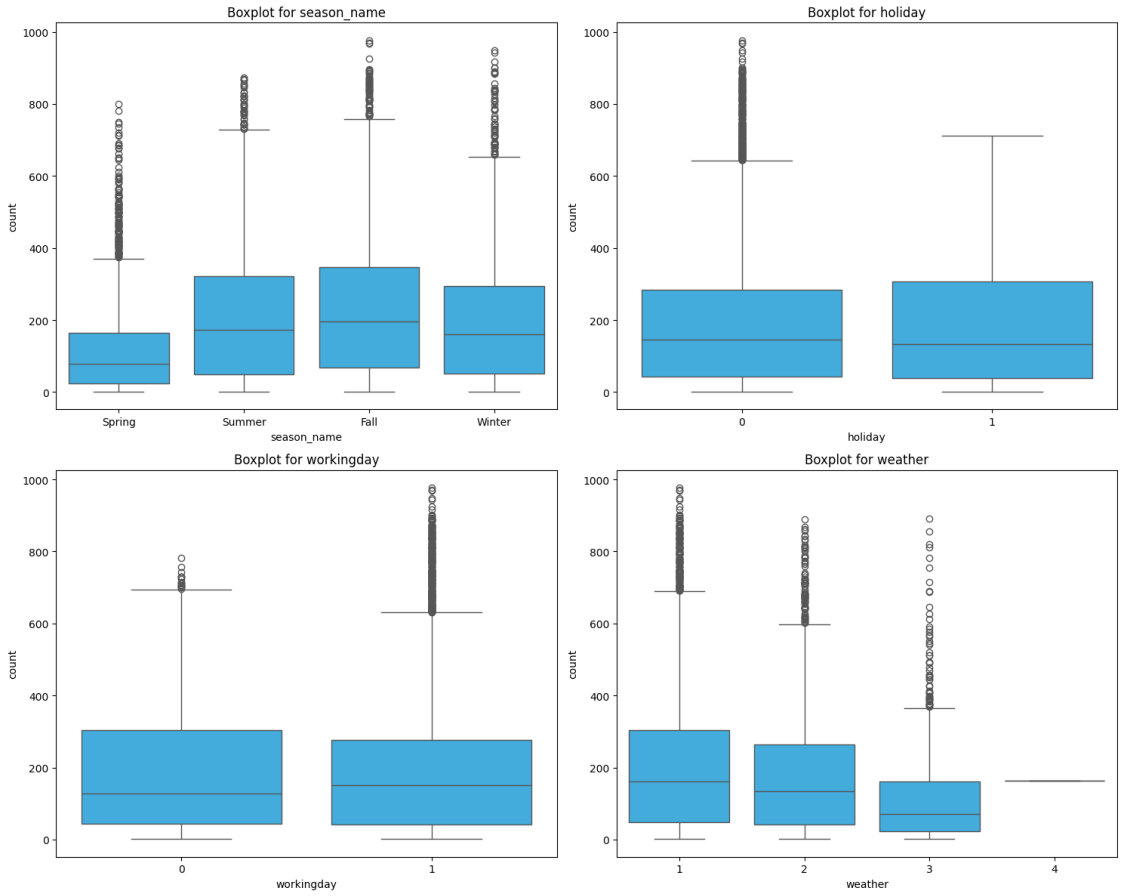


[ ]:
```
cols = ['season_name', 'holiday', 'workingday', 'weather']
plt.figure(figsize=(15, 12))

for i, column in enumerate(cols,1):
    plt.subplot(2, 2, i)
    sns.barplot(x=column, y='count', data=df, width = 0.3)
```

```
    plt.title(f'{column} based distribution of rentals')

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



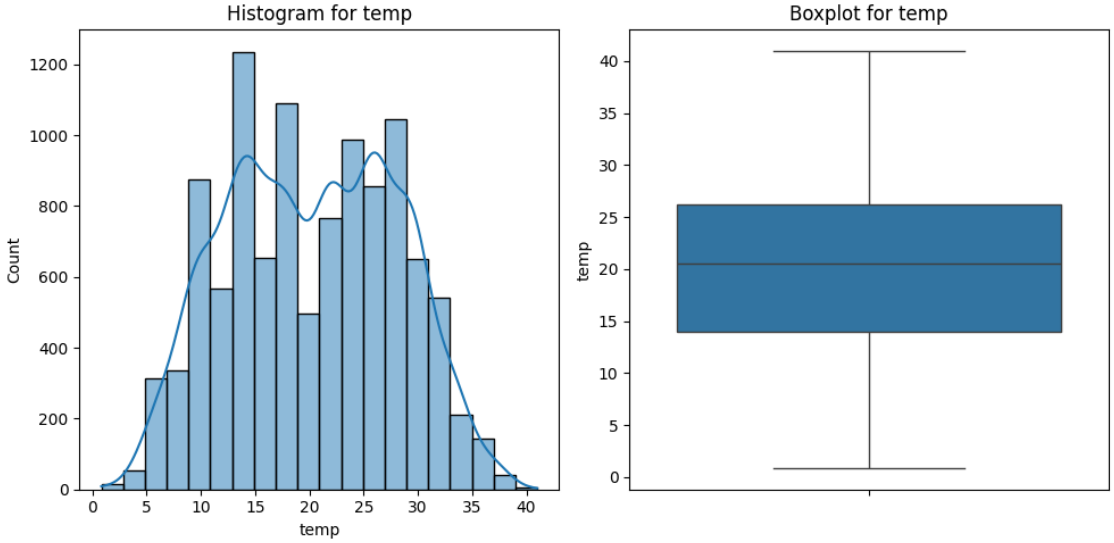
[ ]:
```
plt.figure(figsize=(15, 12))

for i, column in enumerate(cols,1):
    plt.subplot(2, 2, i)
    sns.boxplot(x=column, y='count', data=df, color="#29B6F6")
    plt.title(f'Boxplot for {column}')

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



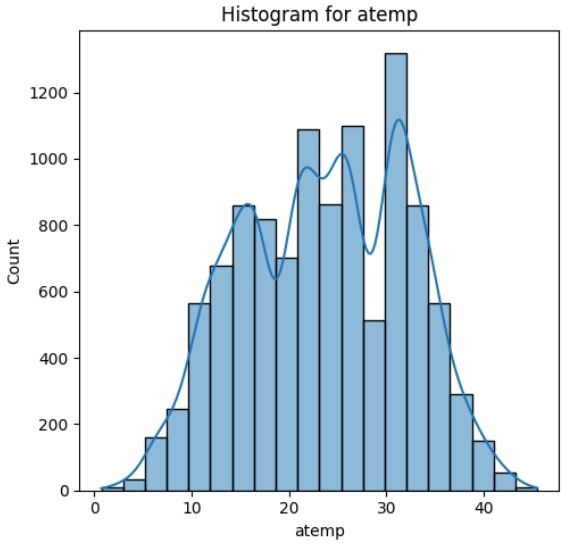
[ ]:
```python
def hist_box(column):
    f, axs = plt.subplots(1, 2, figsize=(10, 5))

    plt.subplot(1, 2, 1)
    sns.histplot(df[column], bins=20, kde=True)
    plt.title(f'Histogram for {column}')

    plt.subplot(1, 2, 2)
    sns.boxplot(df[column])
    plt.title(f'Boxplot for {column}')

    tabular_data = df[column].describe().reset_index()
    tabular_data.columns = ['Statistic', 'Value']
    display(tabular_data)

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

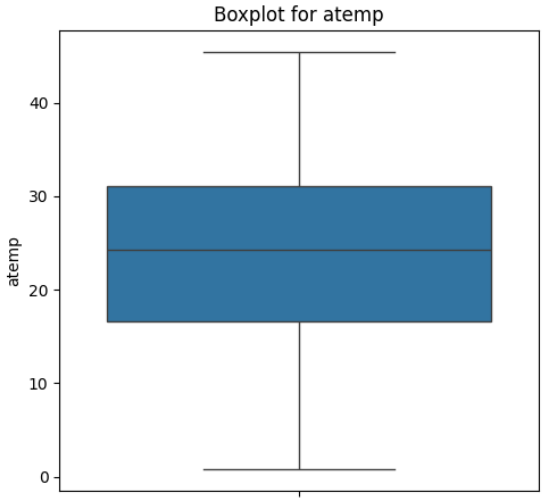
```
num_col = ['temp', 'atemp', 'humidity', 'windspeed', 'casual', 'registered',␣
  ↪'count']

for column in num_col:
    hist_box(column)
```

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

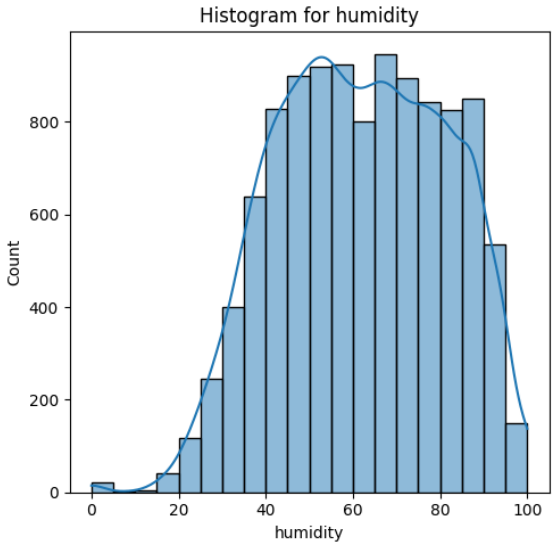


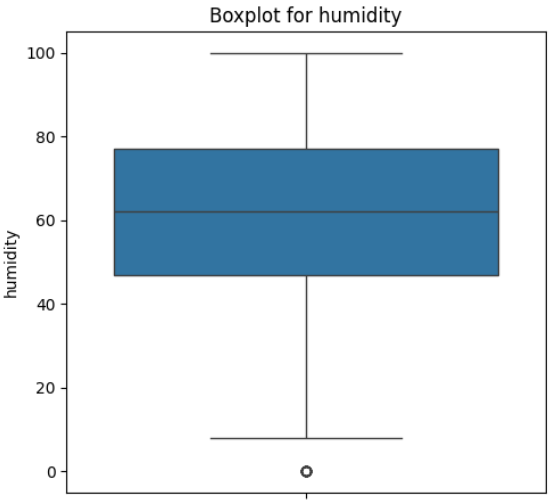
	Statistic	Value
0	count	10886.000000
1	mean	23.655084
2	std	8.474601
3	min	0.760000
4	25%	16.665000
5	50%	24.240000
6	75%	31.060000
7	max	45.455000





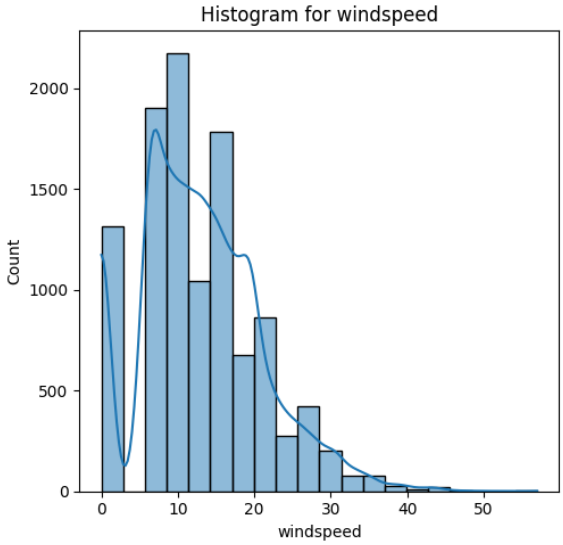
	Statistic	Value
0	count	10886.000000
1	mean	61.886460
2	std	19.245033
3	min	0.000000
4	25%	47.000000
5	50%	62.000000
6	75%	77.000000
7	max	100.000000

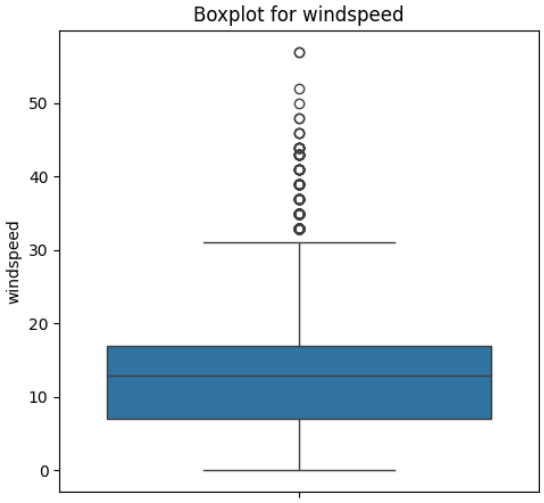




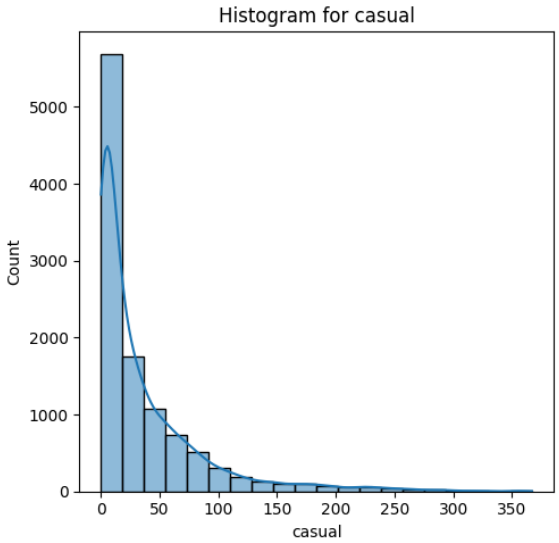
Statistic	Value

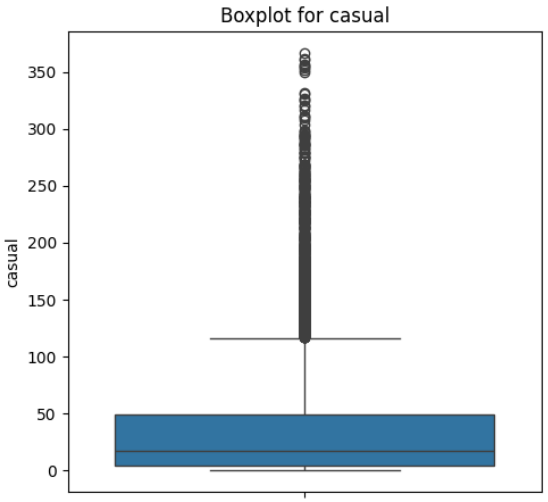
0	count	10886.000000
1	mean	12.799395
2	std	8.164537
3	min	0.000000
4	25%	7.001500
5	50%	12.998000
6	75%	16.997900
7	max	56.996900



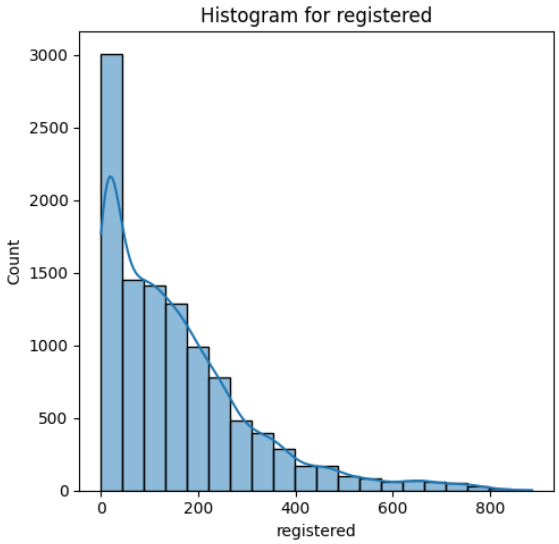


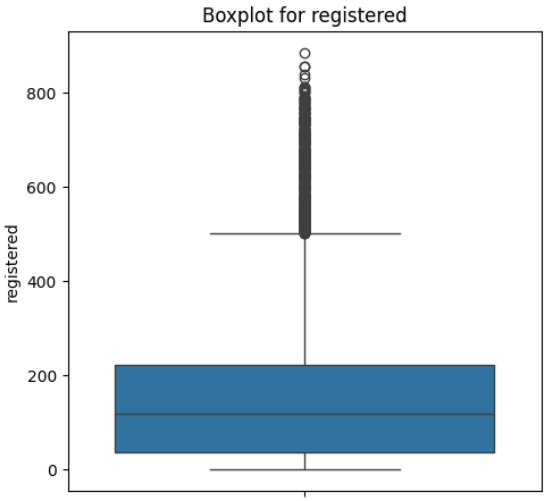
	Statistic	Value
0	count	10886.000000
1	mean	36.021955
2	std	49.960477
3	min	0.000000
4	25%	4.000000
5	50%	17.000000
6	75%	49.000000
7	max	367.000000





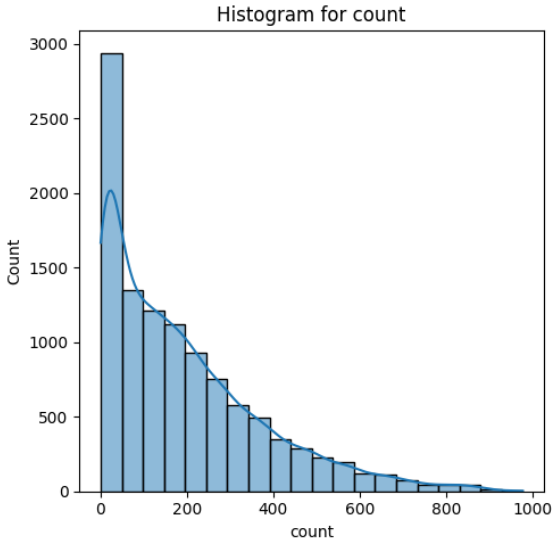
	Statistic	Value
0	count	10886.000000
1	mean	155.552177
2	std	151.039033
3	min	0.000000
4	25%	36.000000
5	50%	118.000000
6	75%	222.000000
7	max	886.000000

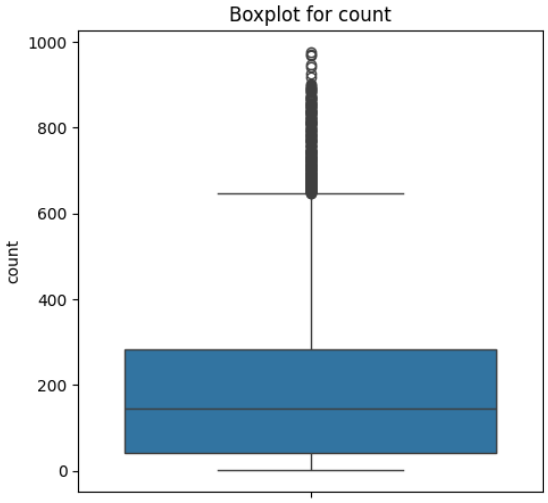




Statistic	Value

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



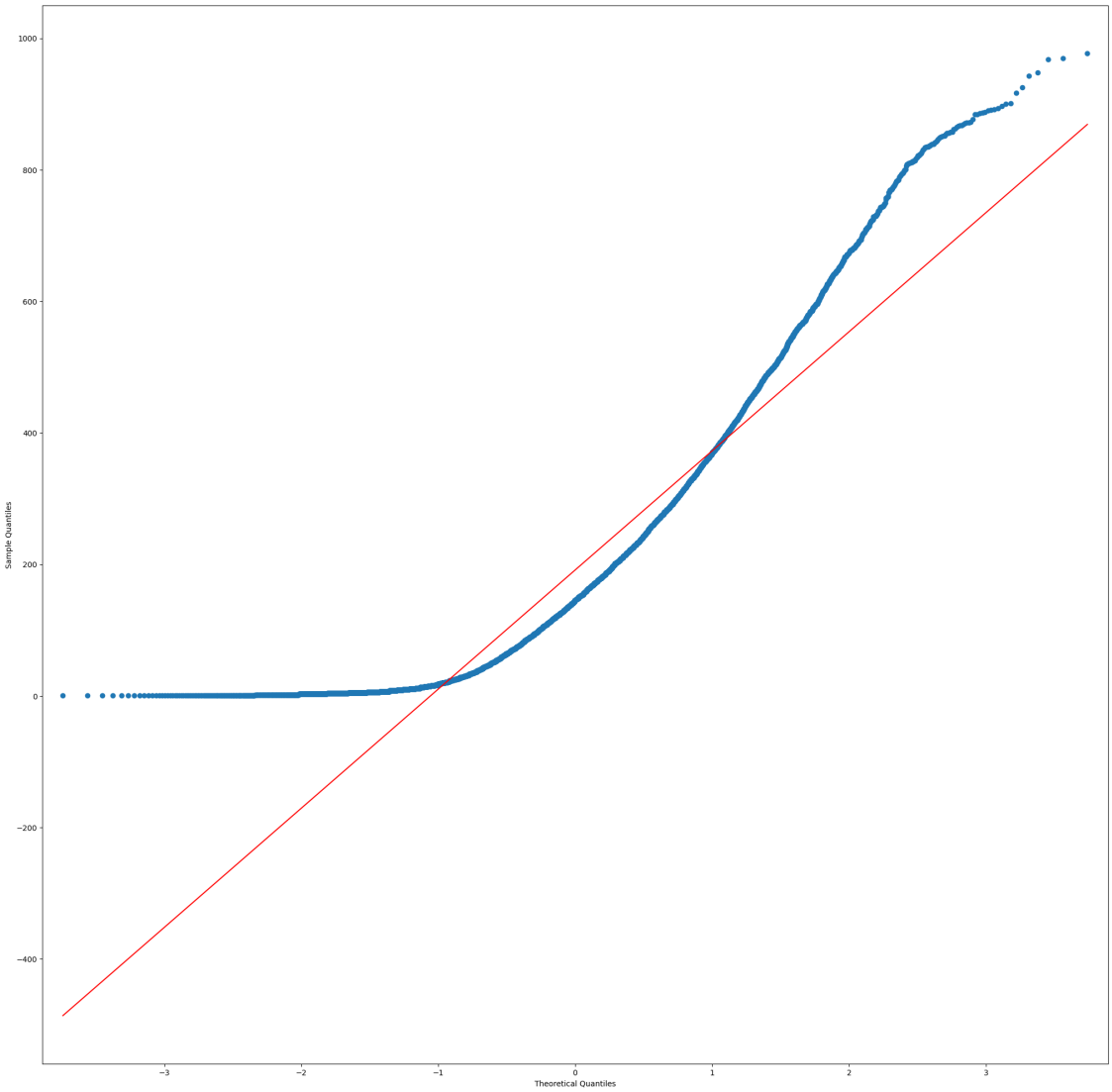


[ ]:
```
# Assumptions of Two Sample Independent T-Test :

#     The data should be normally distributed

#     variances of the two groups are equal
```

[ ]:
```
from scipy import stats
df_subset = df.sample(100)["count"]
test_stat, p_val = stats.shapiro(df_subset)
alpha = 0.05
# H0 data is normally distributed
# Ha data is not normally distributed
if p_val < alpha:
  print("Reject Null Hypothesis, data is not normally distributed")
else:
  print("Fail to reject Null Hypothesis, data is normally distributed")
```

```
Reject Null Hypothesis, data is not normally distributed
```

[ ]:
```
from statsmodels.graphics.gofplots import qqplot
qqplot(df['count'], line = 's')
plt.show()
```



[ ]:
```
# To check if the variances of two groups are equal. We will perform Levene's␣
  ↪test
# The Test hypotheses for Levene's test are:
#     Ho: The variances are equal.
#     Ha: The variances are not equal.


levene_stat, p_val = stats.levene(working, non_working)
print(p_val)
```

```
# Hence the p_values is greater than the significance level, Null hypothesis␣
 ↪can be accepted.

# Therefore, the variances are approximately equal.

# So we can go ahead with the 2 sample independent t test}
```

0.8270298494916468

[ ]:
```
# Ho: There is no significant difference between working and non-working days.

# Ha: There is a significant difference between working and non-working days.


# perform the t test
stat, p_val = stats.ttest_ind(working, non_working)
print(p_val)
# for a significance level of 5%
alpha = 0.05

if p_val < alpha:
  print("Reject Null Hypothesis, There is a significant difference between␣
 ↪working and non-working days")
else:
  print("Fail to reject Null Hypothesis, There is no significant difference␣
 ↪between working and non-working days")
```

0.1766992860641883
Fail to reject Null Hypothesis, There is no significant difference between working and non-working days

[ ]:
```
# Weather conditions
# for weather we have more than 2 categories in weather we can go ahead with␣
 ↪anova

# Assumptions for anova

# 1 The population data should be normally distributed- Data is not normal as␣
 ↪verified above.
# 2 The data points must be independent.
# 3 Approximately equal variance within groups- verified using Levene's test.

# Levenes test
```

```
# Ho: The variances are equal.

# Ha: The variances are not equal.


weather1 = df[df['weather'] == 1]['count']
weather2 = df[df['weather'] == 2]['count']
weather3 = df[df['weather'] == 3]['count']
weather4 = df[df['weather'] == 4]['count']

levene_stat, p_val = stats.levene(weather1, weather2, weather3,weather4)
alpha = 0.05
if p_val < alpha:
  print("Reject Null Hypothesis, Variances are not equal")
else:
  print("Fail to reject Null Hypothesis, Variances are equal")
```

```
Reject Null Hypothesis, Variances are not equal
```

[ ]:
```
# We will perform both anova and kruskal

# Ho: There is no significant difference between demand of bicycles for␣
 ↪different Weather conditions.

# Ha: There is a significant difference between demand of bicycles for␣
 ↪different Weather conditions.

#
anova_stat, p_val = stats.f_oneway(weather1, weather2, weather3,weather4)

# ANOVA
if p_val < alpha:
  print("Reject Null Hypothesis, There is a significant difference between␣
 ↪demand of bicycles for different Weather conditions")
else:
  print("Fail to reject Null Hypothesis, here is no significant difference␣
 ↪between demand of bicycles for different Weather conditions")
```

```
Reject Null Hypothesis, There is a significant difference between demand of
bicycles for different Weather conditions
```

[ ]:
```
# Kruskal
kruskal_stat, p_val = stats.kruskal(weather1, weather2, weather3,weather4)

if p_val < alpha:
  print("Reject Null Hypothesis, There is a significant difference between␣
 ↪demand of bicycles for different Weather conditions")
```

```
else:
  print("Fail to reject Null Hypothesis, here is no significant difference␣
 ↪between demand of bicycles for different Weather conditions")
```

Reject Null Hypothesis, There is a significant difference between demand of bicycles for different Weather conditions

[ ]:
```
# Seasons

# for seasons we have more than 2 categories in weather we can go ahead with␣
 ↪anova

# Assumptions for anova

# 1 The population data should be normally distributed- Data is not normal as␣
 ↪verified above.
# 2 The data points must be independent.
# 3 Approximately equal variance within groups- verified using Levene's test.


# levens test

# Ho: The variances are equal.

# Ha: The variances are not equal.


spring = df[df['season_name'] == 'Spring']['count']
summer = df[df['season_name'] == 'Summer']['count']
fall = df[df['season_name'] == 'Fall']['count']
winter = df[df['season_name'] == 'Winter']['count']

levene_stat, p_val = stats.levene(spring,summer,fall,winter)

alpha = 0.05
if p_val < alpha:
  print("Reject Null Hypothesis, Variances are not equal")
else:
  print("Fail to reject Null Hypothesis, Variances are equal")
```

Reject Null Hypothesis, Variances are not equal

[ ]:
```
# We will perform both anova and kruskal

# Ho: There is no significant difference between demand of bicycles for␣
 ↪different Seasons.
```

```
# Ha: There is a significant difference between demand of bicycles for␣
 ↪different Seasons.

# ANOVA
anova_stat, p_val = stats.f_oneway(spring ,summer, fall, winter)

if p_val < alpha:
  print("Reject Null Hypothesis, There is a significant difference between␣
 ↪demand of bicycles for different Seasons")
else:
  print("Fail to reject Null Hypothesis, here is no significant difference␣
 ↪between demand of bicycles for different Seasons")
```

```
Reject Null Hypothesis, There is a significant difference between demand of
bicycles for different Seasons
```

[ ]:
```
kruskal_stat, p_val = stats.kruskal(spring ,summer, fall, winter)
if p_val < alpha:
  print("Reject Null Hypothesis, There is a significant difference between␣
 ↪demand of bicycles for different Seasons")
else:
  print("Fail to reject Null Hypothesis, here is no significant difference␣
 ↪between demand of bicycles for different Seasons")
```

```
Reject Null Hypothesis, There is a significant difference between demand of
bicycles for different Seasons
```

[ ]:
```
# Weather conditions across seasons
# Since both are categorical columns we will use chi square test

# Ho: Season and Weather are independent of each other.

# Ha: Season and Weather are dependent on each other.


cont_table = pd.crosstab(df['weather'], df['season_name'])
cont_table
```

[ ]:

season_name	Fall	Spring	Summer	Winter
weather				
1	1930	1759	1801	1702
2	604	715	708	807
3	199	211	224	225
4	0	1	0	0

[ ]:
```
# perform the chi square test
stats.chi2_contingency(cont_table)
```

[ ]:
```
Chi2ContingencyResult(statistic=49.15865559689363,
pvalue=1.5499250736864862e-07, dof=9, expected_freq=array([[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]]))
```

[ ]:
```
p_val = 1.5499250736864862e-07
if p_val < alpha:
  print("Reject Null Hypothesis, Season and Weather are dependent on each␣
 ↪other")
else:
  print("Fail to reject Null Hypothesis,Season and Weather are independent of␣
 ↪each other")
```

```
Reject Null Hypothesis, Season and Weather are dependent on each other
```

[ ]:
```

```