```
1 import numpy as np
2 import pandas as pd
3 import matplotlib.pyplot as plt
4 import seaborn as sns
5 from scipy.stats import norm, ttest_ind
```

```
1 !gdown https://drive.google.com/file/d/1o94fXnmvrx6jRgI6S-SeZ3tfnKjCDY0i/view?usp=sharing --fuzzy
```

```
Downloading...
From: https://drive.google.com/uc?id=1o94fXnmvrx6jRgI6S-SeZ3tfnKjCDY0i
To: /content/bike_sharing.csv
100% 648k/648k [00:00<00:00, 84.7MB/s]
```

```
1 df = pd.read_csv('bike_sharing.csv')
```

```
1 df.head()
```

	datetime	season	holiday	workingday	weather	temp	atemp	humidity	windspeed
0	2011-01-01 00:00:00	1	0	0	1	9.84	14.395	81	0.0
1	2011-01-01 01:00:00	1	0	0	1	9.02	13.635	80	0.0
	2011-01-								

Next steps: Generate code with df | View recommended plots

```
1 df.shape
```

```
(10886, 12)
```

```
1 df.describe()
```

	season	holiday	workingday	weather	temp	atem
count	10886.000000	10886.000000	10886.000000	10886.000000	10886.00000	10886.00000
mean	2.506614	0.028569	0.680875	1.418427	20.23086	23.65508
std	1.116174	0.166599	0.466159	0.633839	7.79159	8.47460
min	1.000000	0.000000	0.000000	1.000000	0.82000	0.76000
25%	2.000000	0.000000	0.000000	1.000000	13.94000	16.66500
50%	3.000000	0.000000	1.000000	1.000000	20.50000	24.24000
75%	4.000000	0.000000	1.000000	2.000000	26.24000	31.06000
max	4.000000	1.000000	1.000000	4.000000	41.00000	45.45500

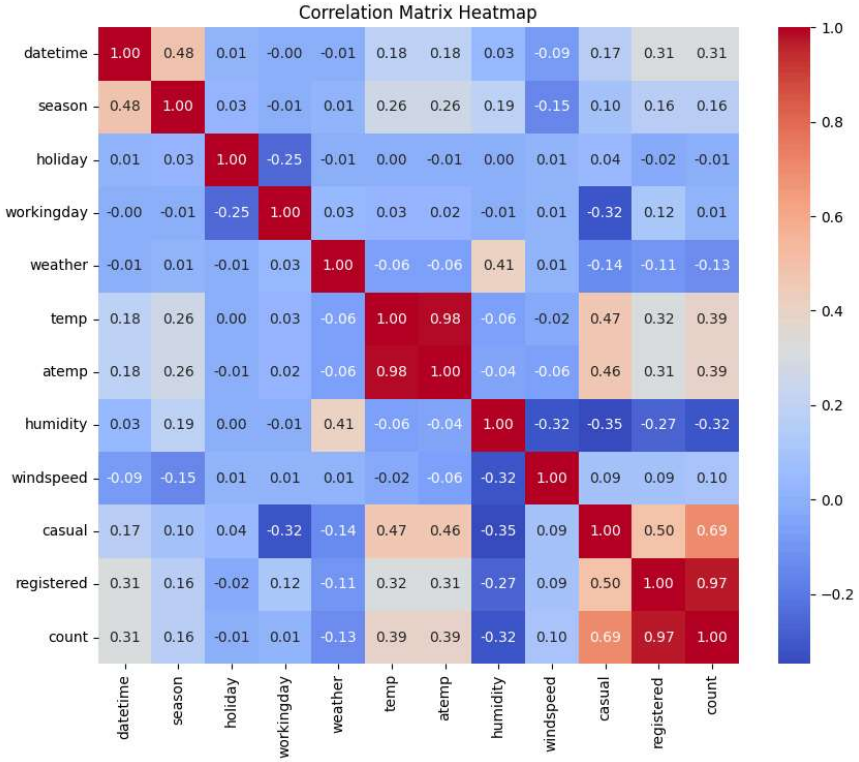
```
1  df['datetime'] = pd.to_datetime(df['datetime'])
2  max(df['datetime']) - min(df['datetime'])
```

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

```
1 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  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: datetime64[ns](1), float64(3), int64(8)
memory usage: 1020.7 KB
```

```
df_heatmap = df.corr()
plt.figure(figsize=(10, 8))
sns.heatmap(df_heatmap, annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlation Matrix Heatmap')
plt.show()
```



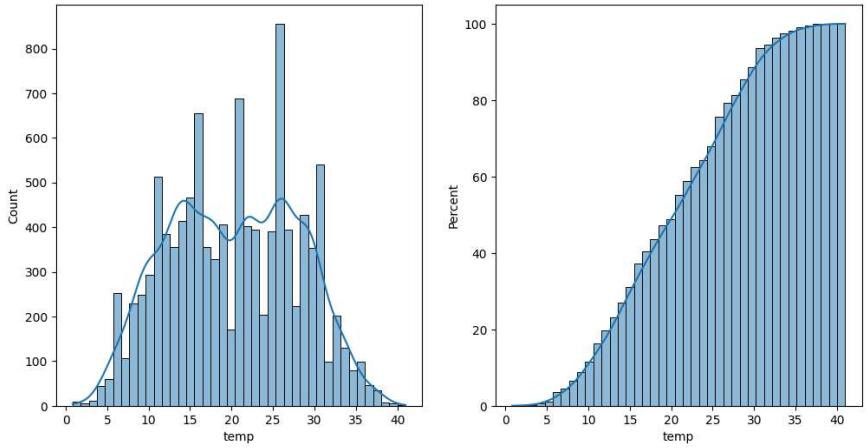
- The correlation of the parameters is evident that as correlation factor is >0.7. The attributes with corelation factor higher are 'registered' <-> 'count' and 'temp' <-> 'atemp'.
- The attribute atemp can be dropped but registered and count shows the different values even if are correlated.

```
df.drop('atemp', axis=1, inplace=True)
```

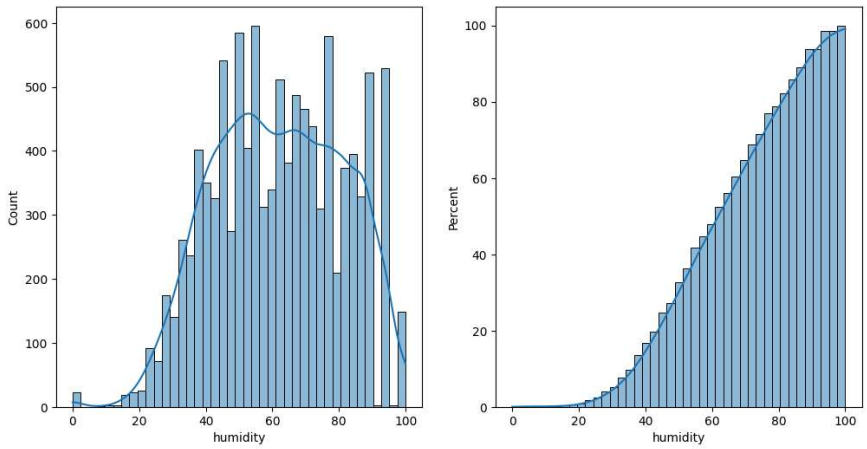
```
df.describe()
```

	datetime	season	holiday	workingday	weather	
count	10886	10886.000000	10886.000000	10886.000000	10886.000000	10886
mean	2011-12-27 05:56:22.399411968	2.506614	0.028569	0.680875	1.418427	20
min	2011-01-01 00:00:00	1.000000	0.000000	0.000000	1.000000	0
25%	2011-07-02 07:15:00	2.000000	0.000000	0.000000	1.000000	13
50%	2012-01-01 20:30:00	3.000000	0.000000	1.000000	1.000000	20
75%	2012-07-01 12:45:00	4.000000	0.000000	1.000000	2.000000	26
max	2012-12-19 23:00:00	4.000000	1.000000	1.000000	4.000000	41
std	NaN	1.116174	0.166599	0.466159	0.633839	7

```
plt.figure(num = 2, figsize = (12,6))

plt.subplot(121)
sns.histplot(df, x = 'temp',kde = True, bins = 41)

plt.subplot(122)
sns.histplot(df, x = 'temp',kde = True, bins = 41, cumulative = True, stat = 'percent')

plt.show()
```

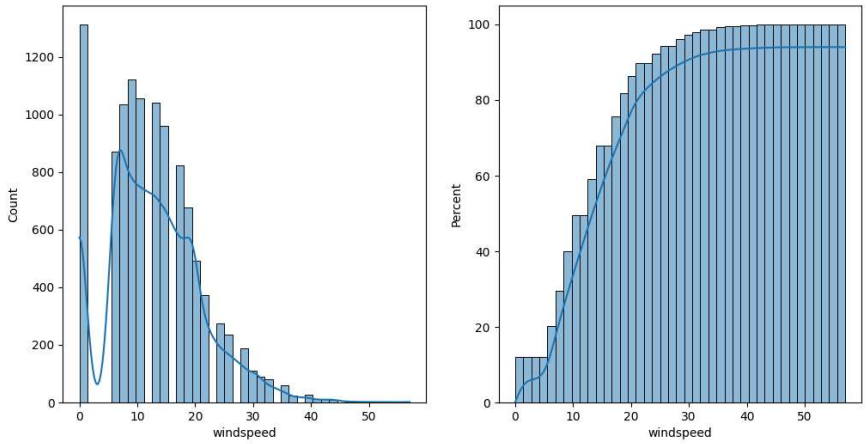



```
plt.figure(num = 2, figsize = (12,6))

plt.subplot(121)
sns.histplot(df, x = 'humidity',kde = True, bins = 41)

plt.subplot(122)
sns.histplot(df, x = 'humidity',kde = True, bins = 41, cumulative = True, stat = 'percent')

plt.show()
```

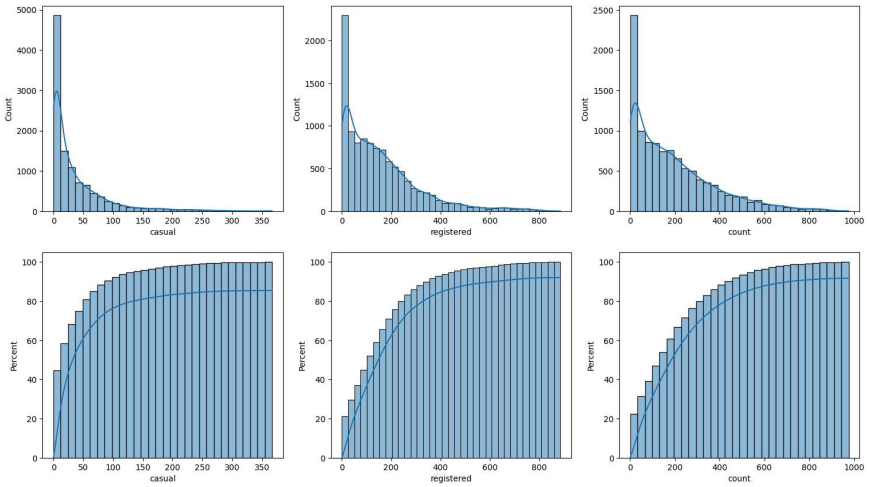



```
plt.figure(num = 2, figsize = (12,6))

plt.subplot(121)
sns.histplot(df, x = 'windspeed',kde = True, bins = 41)
```

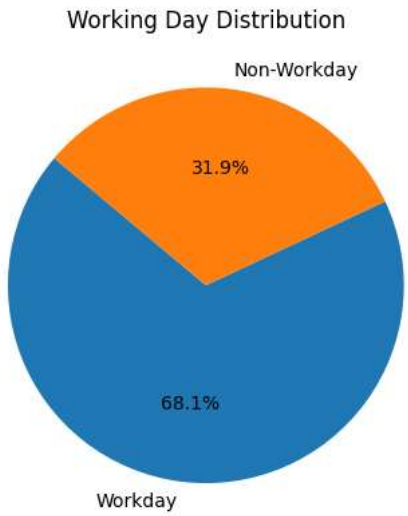
```

plt.subplot(122)
sns.histplot(df, x = 'windspeed',kde = True, bins = 41, cumulative = True, stat = 'percent')

plt.show()
```



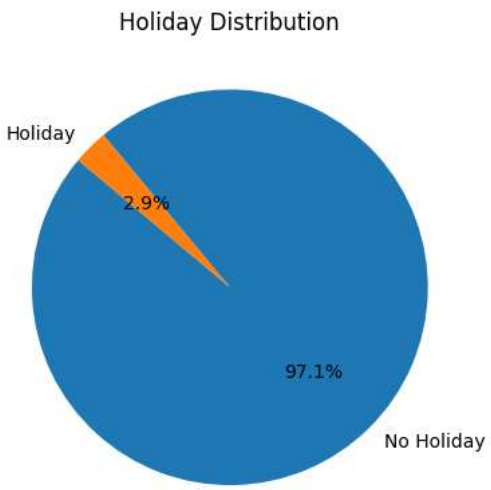

```
plt.figure(num = 6 , figsize = (18, 10))

plt.subplot(231)
sns.histplot(df, x = 'casual', kde = True ,bins = 30)

plt.subplot(232)
sns.histplot(df, x = 'registered', kde = True ,bins = 35)

plt.subplot(233)
sns.histplot(df, x = 'count', kde = True,bins = 30)

plt.subplot(234)
sns.histplot(df, x = 'casual', kde = True ,cumulative = True, bins  = 30 , stat = 'percent')

plt.subplot(235)
sns.histplot(df, x = 'registered', kde = True ,bins = 35, cumulative = True, stat = 'percent')

plt.subplot(236)
sns.histplot(df, x = 'count', kde = True,bins = 30, cumulative = True, stat = 'percent')

plt.show()
```




```
1 workingday_counts = df['workingday'].value_counts()
2 plt.pie(workingday_counts, labels=['Workday', 'Non-Workday'], autopct='%1.1f%%', startangle=140)
3 plt.title('Working Day Distribution')
4 plt.show()
```




```
1 holiday_counts = df['holiday'].value_counts()
2 plt.pie(holiday_counts, labels=['No Holiday', 'Holiday'], autopct='%1.1f%%', startangle=140)
3 plt.title('Holiday Distribution')
4 plt.show()
```




```
1 len(df[(df['workingday'] == 2) & (df['holiday'] == 1)])
```

```
    0
```

```
1 (len(df[(df['workingday'] == 0) & (df['holiday'] == 1)]),
2 len(df[(df['workingday'] == 0) & (df['holiday'] == 1)]) / len(df['workingday']) * 100)
```

```
    (311, 2.856880396839978)
```

There are 0 days with weekday and holiday overlapping. There are 311 days when the holiday is on the weekend.

```
1 season_mapping = {1: 'spring', 2: 'summer', 3: 'fall', 4: 'winter'}
2 df['season'] = df['season'].map(lambda x: season_mapping.get(x, x))
```

```
1 weather_mapping = {1 : 'favorable' , 2 : 'mild', 3 : 'challenging', 4: 'unfavorable'}
2 df['weather'] = df['weather'].map(lambda x: weather_mapping.get(x,x))
```

```
 1 def wind_harshness(windspeed_kmph):
 2     if windspeed_kmph < 5:
 3         return 'Calm'
 4     elif windspeed_kmph < 20:
 5         return 'Light Breeze'
 6     elif windspeed_kmph < 40:
 7         return 'Moderate Breeze'
 8     elif windspeed_kmph < 60:
 9         return 'Strong Breeze'
10     else:
11         return 'High Winds'
12 df['windspeed'] = df['windspeed'].apply(wind_harshness)
```

```
 1 def temp_harshness(temperature_degC):
 2     if temperature_degC <= 10:
 3         return 'Cool'
 4     elif temperature_degC <= 20:
 5         return 'Mild'
 6     elif temperature_degC <= 30:
 7         return 'Warm'
 8     else:
 9         return 'Hot'
10 df['temp'] =df['temp'].apply(temp_harshness)
```

```
 1 def humidity_harshness(humidity_percent):
 2     if humidity_percent <= 30:
 3         return 'Dry'
 4     elif humidity_percent <= 60:
 5         return 'Comfortable'
 6     elif humidity_percent <= 80:
 7         return 'Humid'
 8     else:
 9         return 'Very Humid'
10 df['humidity'] = df['humidity'].apply(humidity_harshness)
```

```
df['date'] = df['datetime'].dt.date
df['time'] = df['datetime'].dt.hour
```

```
df['day_of_month'] = df['datetime'].dt.day
df['month'] = df['datetime'].dt.month
```

```
df['weekday'] = df['datetime'].dt.day_of_week
```

```
df['holiday'] = df['holiday'].astype('category')
df['season'] = df['season'].astype('category')
df['workingday'] = df['workingday'].astype('category')
df['weather'] = df['weather'].astype('category')
df['windspeed'] =df['windspeed'].astype('category')
df['temp'] =df['temp'].astype('category')
df['humidity'] = df['humidity'].astype('category')
df['day_of_month'] = df['day_of_month'].astype('category')
df['month'] = df['month'].astype('category')
df['weekday'] =df['weekday'].astype('category')
```

```
df.describe()
```

	datetime	casual	registered	count	time
count	10886	10886.000000	10886.000000	10886.000000	10886.000000
mean	2011-12-27 05:56:22.399411968	36.021955	155.552177	191.574132	11.541613
min	2011-01-01 00:00:00	0.000000	0.000000	1.000000	0.000000
25%	2011-07-02 07:15:00	4.000000	36.000000	42.000000	6.000000
50%	2012-01-01 20:30:00	17.000000	118.000000	145.000000	12.000000
	2012-07-01				

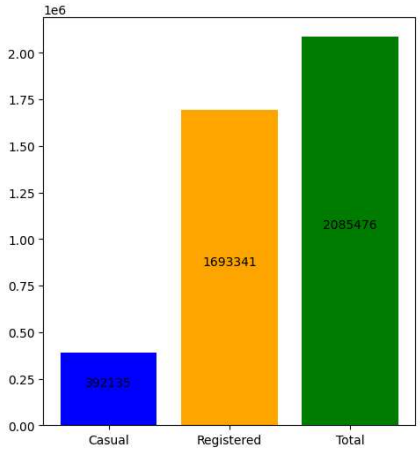
```
df.drop_duplicates(keep = 'first', inplace = True)
```

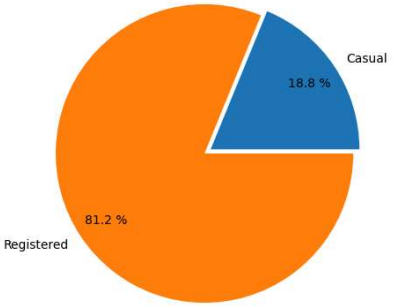
```
df = df[['date', 'time', 'holiday', 'workingday', 'season', 'weather', 'temp',
       'humidity', 'windspeed', 'casual', 'registered', 'count',
       'datetime', 'day_of_month', 'month', 'weekday']]
```

```
df['month'] = df['month'].apply(lambda x: pd.Timestamp(year= 2023 ,month=x, day=1).strftime('%B'))
```

```
weekday_mapping = {0 : 'Monday',1 : 'Tuesday', 2 : 'Wednesday', 3 : 'Thursday', 4 : 'Friday', 5 : 'Saturday' , 6 : 'Sunday'}
df['weekday'] = df['weekday'].apply(lambda x: weekday_mapping.get(x,x))
```

```
plt.figure(num = 2, figsize = (12,6))
plt.subplot(121)
ax = plt.bar(x = ['Casual', 'Registered','Total'], height= [df['casual'].sum(), df['registered'].sum(),df['count'].sum()], color = [

for p in ax.patches:
    plt.annotate(format(p.get_height(), '.0f'),
                 (p.get_x() + p.get_width() / 2., p.get_height()/2),
                 ha = 'center', va = 'center',
                 xytext = (0, 5),
                 textcoords = 'offset points')

plt.subplot(122)

plt.pie(x = [df['casual'].sum(), df['registered'].sum()], labels = ['Casual', 'Registered'],
        autopct = "%.1f %%", pctdistance = 0.8,
        explode = [0, 0.05])

plt.show()
```



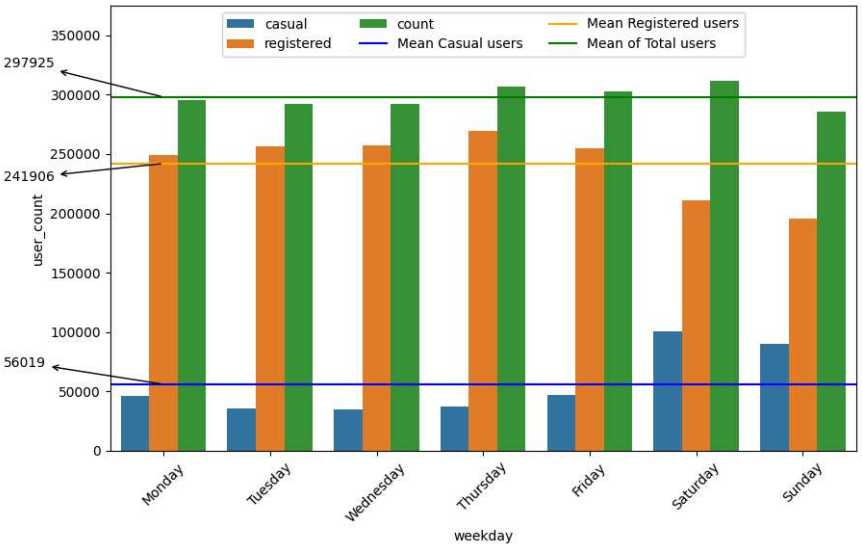



```
1 df_weekday = pd.DataFrame(df.groupby('weekday').agg({'casual' : 'sum' , 'registered' : 'sum' , 'count' : 'sum'})).reset_index()
2 df_weekday
```

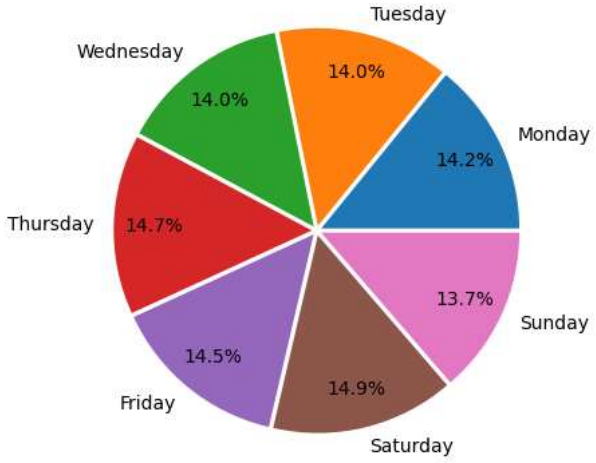
	weekday	casual	registered	count
0	Monday	46288	249008	295296
1	Tuesday	35365	256620	291985
2	Wednesday	34931	257295	292226
3	Thursday	37283	269118	306401
4	Friday	47402	255102	302504
5	Saturday	100782	210736	311518
6	Sunday	90084	195462	285546

Next steps: Generate code with df_weekday | View recommended plots

```
1 df_weekday_melt = pd.melt(df_weekday, id_vars=['weekday'], var_name='user_type', value_name='user_count')
2 plt.figure(figsize = (10,6))
3
4 sns.barplot(df_weekday_melt, x = 'weekday', y = 'user_count' , hue = 'user_type')
5
6
7 mean_casual = df_weekday_melt[df_weekday_melt['user_type'] == 'casual']['user_count'].mean()
8 plt.axhline(y=mean_casual, label='Mean Casual users', color='blue')
9 plt.annotate(f'{mean_casual:.0f}', xy=(0, mean_casual), xytext=(-1.5, mean_casual+15000),
10              arrowprops=dict(facecolor='blue', arrowstyle='<-'))
11
12 mean_registered = df_weekday_melt[df_weekday_melt['user_type'] == 'registered']['user_count'].mean()
13 plt.axhline(y=mean_registered, label='Mean Registered users', color='orange')
14 plt.annotate(f'{mean_registered:.0f}', xy=(0, mean_registered), xytext=(-1.5, mean_registered - 15000),
15              arrowprops=dict(facecolor='orange', arrowstyle='<-'))
16
17 mean_total = df_weekday_melt[df_weekday_melt['user_type'] == 'count']['user_count'].mean()
18 plt.axhline(y=mean_total, label='Mean of Total users', color='green')
19 plt.annotate(f'{mean_total:.0f}', xy=(0, mean_total), xytext=(-1.5, mean_total + 25000),
20              arrowprops=dict(facecolor='green', arrowstyle='<-'))
21
22 plt.ylim(0, 375000)
23 plt.xticks(rotation = 45)
24 plt.legend(loc='upper center', bbox_to_anchor=(0.5, 1), ncol=3)
25 plt.show()
```




```
1 plt.pie(df_weekday['count'], labels = df_weekday['weekday'],
2         autopct = '%1.1f%%', pctdistance = 0.8,
3         explode = [0.025]*len(df_weekday['weekday']))
4 plt.show()
```

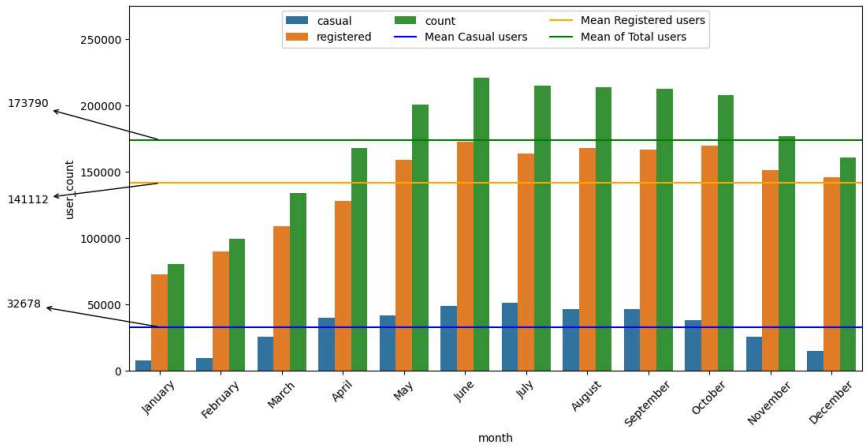



```
1 df_month = pd.DataFrame(df.groupby('month').agg({'casual' : 'sum' , 'registered' : 'sum' , 'count' : 'sum'})).reset_index()
2 df_month
```

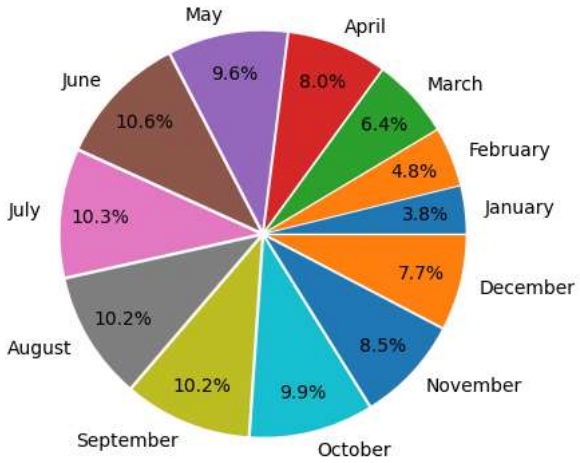
	month	casual	registered	count
0	January	7252	72632	79884
1	February	9297	89816	99113
2	March	25056	108445	133501
3	April	39813	127589	167402
4	May	41285	158862	200147
5	June	48574	172159	220733
6	July	50947	163670	214617
7	August	45870	167646	213516
8	September	45901	166628	212529
9	October	38087	169347	207434
10	November	25353	151087	176440
11	December	14700	145460	160160

Next steps: Generate code with df_month View recommended plots

```
df_month_melt = pd.melt(df_month, id_vars=['month'], var_name='user_type', value_name='user_count')
plt.figure(figsize = (12,6))

sns.barplot(df_month_melt, x = 'month', y = 'user_count' , hue = 'user_type')

mean_casual = df_month_melt[df_month_melt['user_type'] == 'casual']['user_count'].mean()
plt.axhline(y=mean_casual, label='Mean Casual users', color='blue')
plt.annotate(f'{mean_casual:.0f}', xy=(0, mean_casual), xytext=(-2.5, mean_casual+15000),
             arrowprops=dict(facecolor='blue', arrowstyle='<-'))

mean_registered = df_month_melt[df_month_melt['user_type'] == 'registered']['user_count'].mean()
plt.axhline(y=mean_registered, label='Mean Registered users', color='orange')
plt.annotate(f'{mean_registered:.0f}', xy=(0, mean_registered), xytext=(-2.5, mean_registered - 15000),
             arrowprops=dict(facecolor='orange', arrowstyle='<-'))

mean_total = df_month_melt[df_month_melt['user_type'] == 'count']['user_count'].mean()
plt.axhline(y=mean_total, label='Mean of Total users', color='green')
plt.annotate(f'{mean_total:.0f}', xy=(0, mean_total), xytext=(-2.5, mean_total + 25000),
             arrowprops=dict(facecolor='green', arrowstyle='<-'))

plt.ylim(0, 275000)
plt.xticks(rotation = 45)
plt.legend(loc='upper center', bbox_to_anchor=(0.5, 1), ncol=3)
plt.show()

```




```
1 plt.pie(df_month['count'], labels = df_month['month'],
2         autopct = '%1.1f%%', pctdistance = 0.8,
3         explode = [0.025]*len(df_month['month']))
4 plt.show()
```




```
1 month_order = ['January', 'February', 'March', 'April', 'May', 'June',
2                'July', 'August', 'September', 'October', 'November', 'December']
3
4 df_month_season = df.groupby(['season', 'month']).agg({'count': 'sum'}).reset_index()
5 df_month_season = df_month_season[df_month_season['count'] != 0]
6 df_month_season['month'] = pd.Categorical(df_month_season['month'], categories=month_order, ordered=True)
7 df_month_season = df_month_season.sort_values('month')
8 df_month_season.reset_index(drop=True, inplace=True)
9 df_month_season
```

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

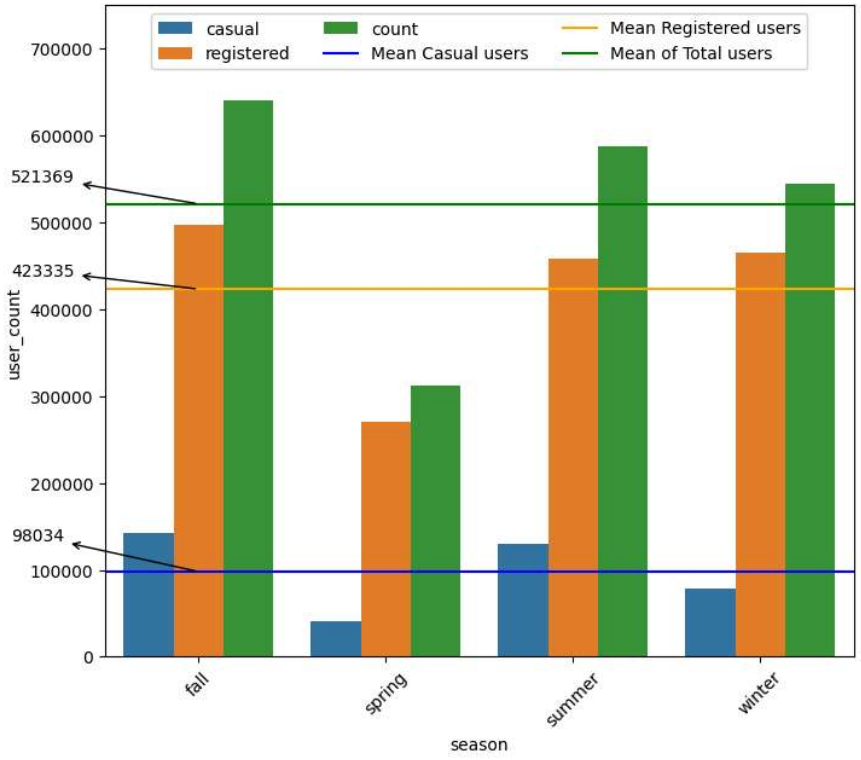
Next steps: Generate code with df_month_season View recommended plots

```
1 # The users distribution depending on the season
2 df_season = pd.DataFrame(df.groupby(['season']).agg({'casual' : 'sum', 'registered' : 'sum', 'count' : 'sum'}).reset_index())
3 df_season
```

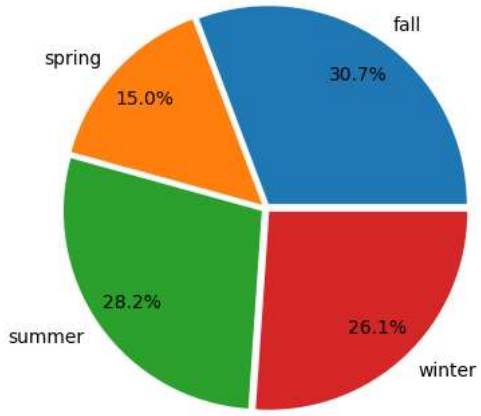
	season	casual	registered	count
0	fall	142718	497944	640662
1	spring	41605	270893	312498
2	summer	129672	458610	588282
3	winter	78140	465894	544034

Next steps: Generate code with df_season View recommended plots

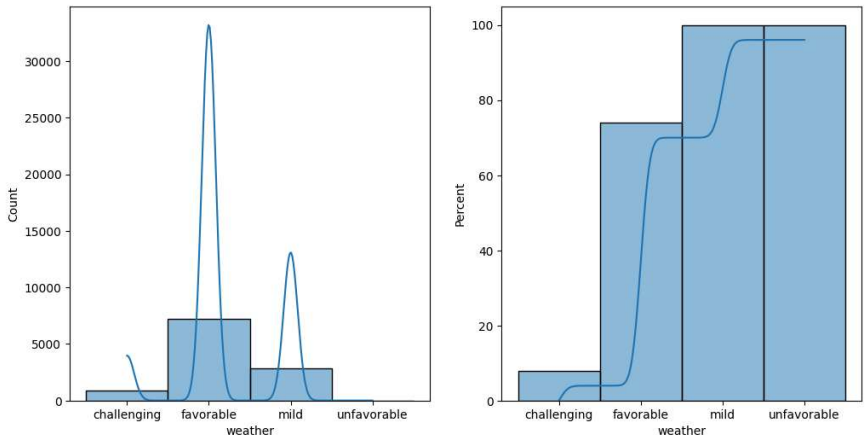
```
1 df_season_melt = pd.melt(df_season, id_vars=['season'], var_name='user_type', value_name='user_count')
2 plt.figure(figsize = (8,7))
3
4 sns.barplot(df_season_melt, x = 'season', y = 'user_count' , hue = 'user_type')
5
6 mean_casual = df_season_melt[df_season_melt['user_type'] == 'casual']['user_count'].mean()
7 plt.axhline(y=mean_casual, label='Mean Casual users', color='blue')
8 plt.annotate(f'{mean_casual:.0f}', xy=(0, mean_casual), xytext=(-1, mean_casual+ 35000),
9              arrowprops=dict(facecolor='blue', arrowstyle='<-'))
10
11 mean_registered = df_season_melt[df_season_melt['user_type'] == 'registered']['user_count'].mean()
12 plt.axhline(y=mean_registered, label='Mean Registered users', color='orange')
13 plt.annotate(f'{mean_registered:.0f}', xy=(0, mean_registered), xytext=(-1, mean_registered + 15000),
14              arrowprops=dict(facecolor='orange', arrowstyle='<-'))
15
16 mean_total = df_season_melt[df_season_melt['user_type'] == 'count']['user_count'].mean()
17 plt.axhline(y=mean_total, label='Mean of Total users', color='green')
18 plt.annotate(f'{mean_total:.0f}', xy=(0, mean_total), xytext=(-1, mean_total + 25000),
19              arrowprops=dict(facecolor='green', arrowstyle='<-'))
20
21 plt.ylim(0, 750000)
22 plt.xticks(rotation = 45)
23 plt.legend(loc='upper center', bbox_to_anchor=(0.5, 1), ncol=3)
24 plt.show()
```



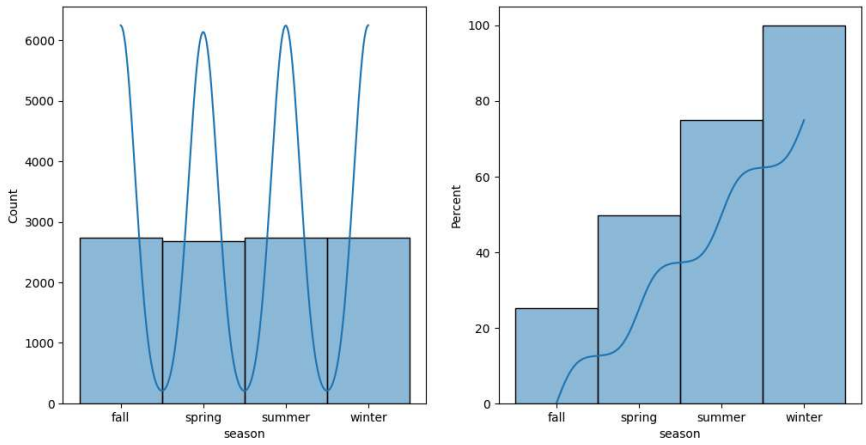

```
1 plt.pie(df_season['count'], labels = df_season['season'],
2         autopct = '%1.1f%%', pctdistance = 0.8,
3         explode = [0.025]*len(df_season['season']))
4 plt.show()
```




```
1 plt.figure(num = 2, figsize = (12,6))
2
3 plt.subplot(121)
4 sns.histplot(df, x = 'weather',kde = True, bins = 4)
5
6 plt.subplot(122)
7 sns.histplot(df, x = 'weather',kde = True, bins = 4, cumulative = True, stat = 'percent')
8
9 plt.show()
```

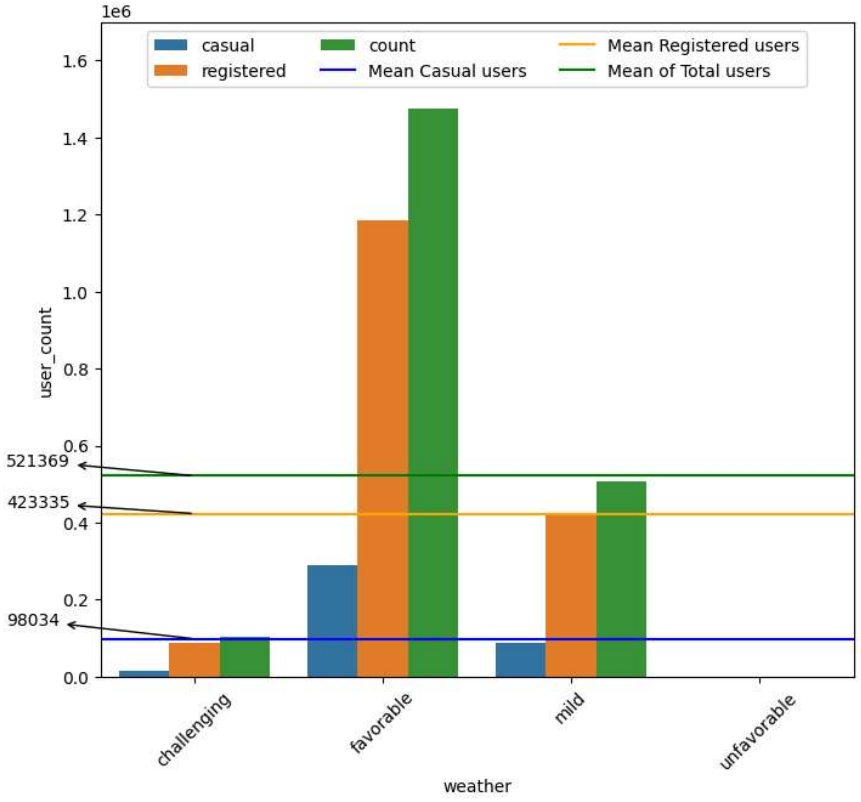



```
1 plt.figure(num = 2, figsize = (12,6))
2
3 plt.subplot(121)
4 sns.histplot(df, x = 'season',kde = True, bins = 4)
5
6 plt.subplot(122)
7 sns.histplot(df, x = 'season',kde = True, bins = 4, cumulative = True, stat = 'percent')
8
9 plt.show()
```

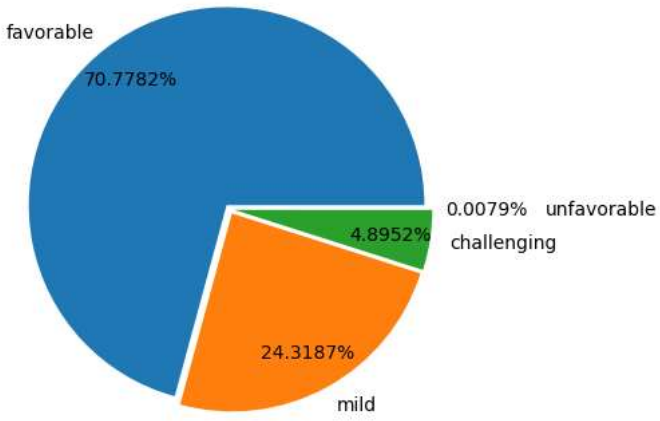



```
1   # The users distribution depending on the weather
2   df_weather = pd.DataFrame(df.groupby(['weather']).agg({'casual' : 'sum', 'registered' : 'sum', 'count' : 'sum'}).reset_index())
3   df_weather.sort_values('count', ascending = False, inplace = True)
```

```
df_weather_melt = pd.melt(df_weather, id_vars=['weather'], var_name='user_type', value_name='user_count')

plt.figure(figsize = (8,7))

sns.barplot(df_weather_melt, x = 'weather', y = 'user_count' , hue = 'user_type')

mean_casual = df_weather_melt[df_weather_melt['user_type'] == 'casual']['user_count'].mean()
plt.axhline(y=mean_casual, label='Mean Casual users', color='blue')
plt.annotate(f'{mean_casual:.0f}', xy=(0, mean_casual), xytext=(-1, mean_casual+ 35000),
             arrowprops=dict(facecolor='blue', arrowstyle='<-'))

mean_registered = df_weather_melt[df_weather_melt['user_type'] == 'registered']['user_count'].mean()
plt.axhline(y=mean_registered, label='Mean Registered users', color='orange')
plt.annotate(f'{mean_registered:.0f}', xy=(0, mean_registered), xytext=(-1, mean_registered + 15000),
             arrowprops=dict(facecolor='orange', arrowstyle='<-'))

mean_total = df_weather_melt[df_weather_melt['user_type'] == 'count']['user_count'].mean()
plt.axhline(y=mean_total, label='Mean of Total users', color='green')
plt.annotate(f'{mean_total:.0f}', xy=(0, mean_total), xytext=(-1, mean_total + 25000),
             arrowprops=dict(facecolor='green', arrowstyle='<-'))

plt.ylim(0, 1.7*10**6)
plt.xticks(rotation = 45)
plt.legend(loc='upper center', bbox_to_anchor=(0.5, 1), ncol=3)
plt.show()
```




```
plt.pie(df_weather['count'], labels = df_weather['weather'],
        autopct = '%1.4 f%%', pctdistance = 0.7,
        explode = [0.025, 0.025, 0.025, 0.5])
plt.show()
```

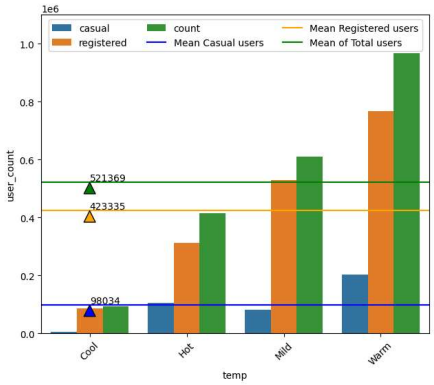


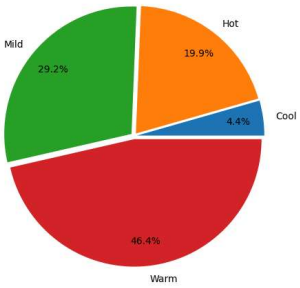

```
df_temp = pd.DataFrame(df.groupby('temp').agg({'casual' : 'sum', 'registered' : 'sum', 'count' : 'sum'}).reset_index())
df_temp
```

	temp	casual	registered	count
0	Cool	4973	87168	92141
1	Hot	104288	311549	415837
2	Mild	80678	528555	609233
3	Warm	202196	766069	968265

Next steps: Generate code with df_temp | View recommended plots

```
df_temp_melt = pd.melt(df_temp, id_vars=['temp'], var_name='user_type', value_name='user_count')
plt.figure(figsize = (16,6), num = 2)
plt.subplot(121)
sns.barplot(df_temp_melt, x = 'temp', y = 'user_count' , hue = 'user_type')

mean_casual = df_temp_melt[df_temp_melt['user_type'] == 'casual']['user_count'].mean()
plt.axhline(y=mean_casual, label='Mean Casual users', color='blue')
plt.annotate(f'{mean_casual:.0f}', xy=(0, mean_casual), xytext=(0, mean_casual+ 5000),
             arrowprops=dict(facecolor='blue'))

mean_registered = df_temp_melt[df_temp_melt['user_type'] == 'registered']['user_count'].mean()
plt.axhline(y=mean_registered, label='Mean Registered users', color='orange')
plt.annotate(f'{mean_registered:.0f}', xy=(0, mean_registered), xytext=(0, mean_registered + 5000),
             arrowprops=dict(facecolor='orange'))

mean_total = df_temp_melt[df_temp_melt['user_type'] == 'count']['user_count'].mean()
plt.axhline(y=mean_total, label='Mean of Total users', color='green')
plt.annotate(f'{mean_total:.0f}', xy=(0, mean_total), xytext=(0, mean_total + 5000),
             arrowprops=dict(facecolor='green'))

plt.ylim(0, 1100000)
plt.xticks(rotation = 45)
plt.legend(loc='upper center', bbox_to_anchor=(0.5, 1), ncol=3)

plt.subplot(122, aspect = 'equal')
plt.pie(df_temp['count'], labels = df_temp['temp'],
        autopct = '%1.1f%%', pctdistance = 0.8,
        explode = [0.025]*len(df_temp['temp']))
plt.show()
```



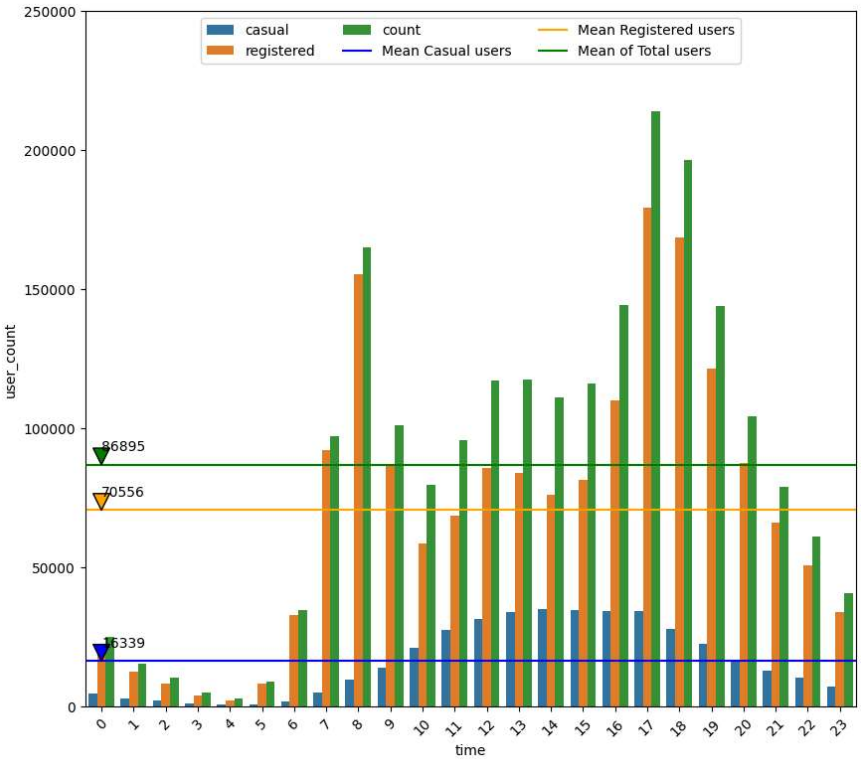


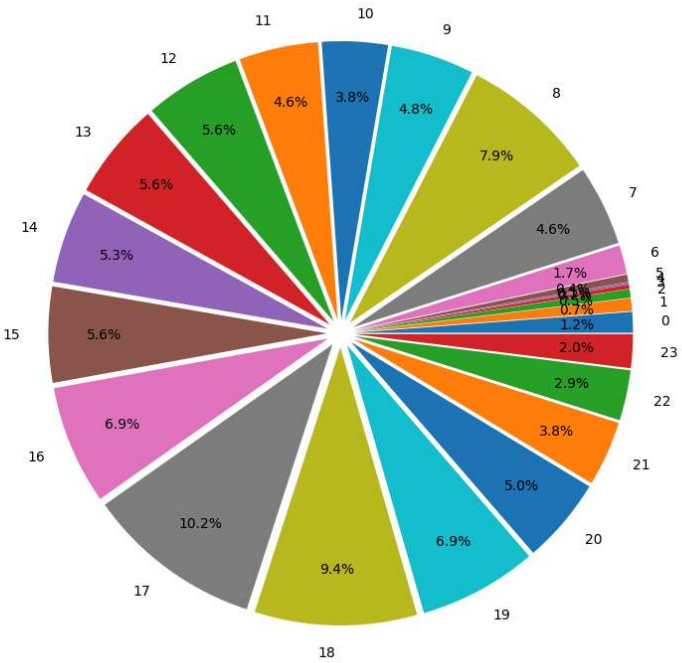

```
1 df_time = pd.DataFrame(df.groupby('time').agg({'casual' : 'sum', 'registered' : 'sum', 'count' : 'sum'}).reset_index())
2 df_time
```

	time	casual	registered	count
0	0	4692	20396	25088
1	1	2957	12415	15372
2	2	2159	8100	10259
3	3	1161	3930	5091
4	4	558	2274	2832
5	5	658	8277	8935
6	6	1888	32810	34698
7	7	4966	92002	96968
8	8	9802	155258	165060
9	9	14085	86825	100910
10	10	20984	58683	79667
11	11	27324	68533	95857
12	12	31387	85581	116968
13	13	33771	83780	117551
14	14	34925	76085	111010
15	15	34669	81291	115960
16	16	34238	110028	144266
17	17	34401	179356	213757
18	18	27997	168475	196472
19	19	22378	121389	143767
20	20	16750	87454	104204
21	21	13027	66030	79057
22	22	10307	50604	60911
23	23	7051	33765	40816

Next steps: Generate code with df_time | View recommended plots

```
df_time_melt = pd.melt(df_time, id_vars=['time'], var_name='user_type', value_name='user_count')

plt.figure(figsize = (10,20), num = 2)
plt.subplot(211)
sns.barplot(df_time_melt, x = 'time', y = 'user_count' , hue = 'user_type')

mean_casual = df_time_melt[df_time_melt['user_type'] == 'casual']['user_count'].mean()
plt.axhline(y=mean_casual, label='Mean Casual users', color='blue')
plt.annotate(f'{mean_casual:.0f}', xy=(0, mean_casual), xytext=(0, mean_casual+ 5000),
             arrowprops=dict(facecolor='blue'))

mean_registered = df_time_melt[df_time_melt['user_type'] == 'registered']['user_count'].mean()
plt.axhline(y=mean_registered, label='Mean Registered users', color='orange')
plt.annotate(f'{mean_registered:.0f}', xy=(0, mean_registered), xytext=(0, mean_registered + 5000),
             arrowprops=dict(facecolor='orange'))

mean_total = df_time_melt[df_time_melt['user_type'] == 'count']['user_count'].mean()
plt.axhline(y=mean_total, label='Mean of Total users', color='green')
plt.annotate(f'{mean_total:.0f}', xy=(0, mean_total), xytext=(0, mean_total + 5000),
             arrowprops=dict(facecolor='green'))

plt.ylim(0, 250000)
plt.xticks(rotation = 45)
plt.legend(loc='upper center', bbox_to_anchor=(0.5, 1), ncol=3)

plt.subplot(212, aspect = 'equal')
plt.pie(df_time['count'], labels = df_time['time'],
        autopct = '%1.1f%%', pctdistance = 0.8,
        explode = [0.05]*len(df_time))
plt.show()
```


casual
registered
count
Mean Casual users
Mean Registered users
Mean of Total users
user_count
time
86895
70556
16339


3.8%
4.8%
7.9%
4.6%
1.7%
1.2%
2.0%
2.9%
3.8%
5.0%
6.9%
9.4%
10.2%
6.9%
5.6%
5.3%
5.6%
5.6%
4.6%

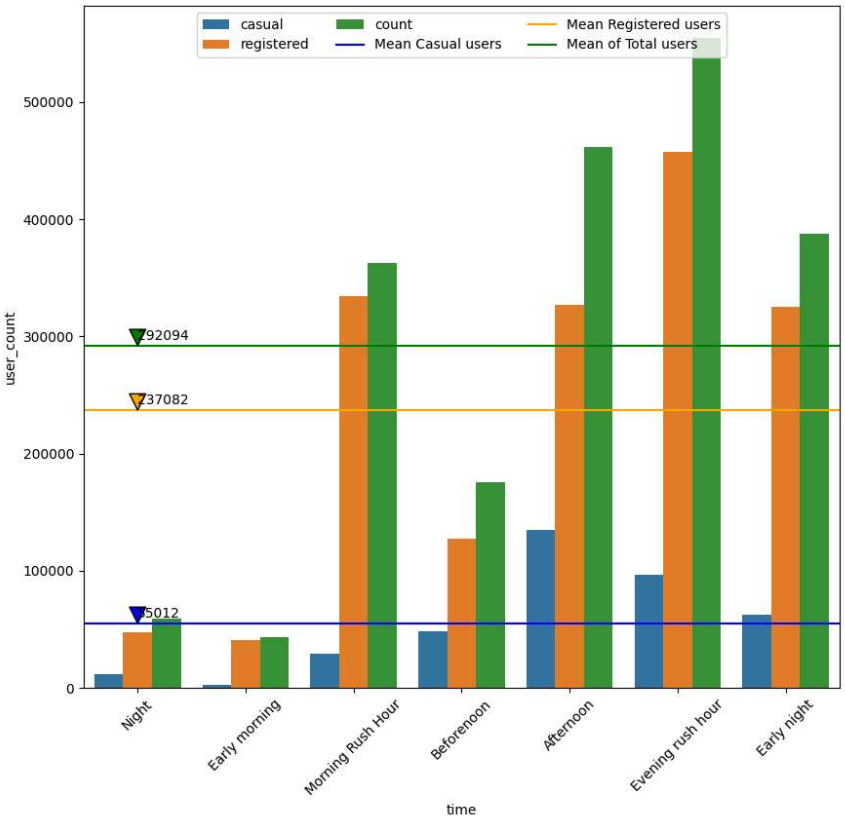
```
#Converting the 24 hour Shcedule 3-6 hour groups depending on the time of day and rush hours.
bins = [-1, 5, 7, 10, 12, 16, 19, 23]  # Define bins for the groups
labels = ['Night', 'Early morning', 'Morning Rush Hour', 'Beforenoon','Afternoon', 'Evening rush hour', 'Early night']

df['time'] = pd.cut(df['time'], bins=bins, labels=labels, right=False)
```

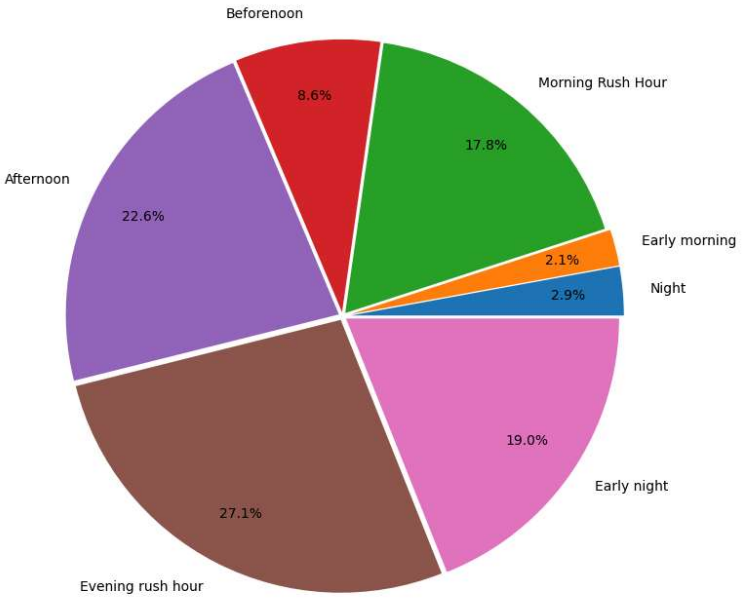
```
df_time = pd.DataFrame(df.groupby('time').agg({'casual' : 'sum', 'registered' : 'sum', 'count' : 'sum'}).reset_index())
df_time
```

	time	casual	registered	count
0	Night	11527	47115	58642
1	Early morning	2546	41087	43633
2	Morning Rush Hour	28853	334085	362938
3	Beforenoon	48308	127216	175524
4	Afternoon	134752	326737	461489
5	Evening rush hour	96636	457859	554495
6	Early night	62462	325477	387939

Next steps: Generate code with df_time | View recommended plots

```
df_time_melt = pd.melt(df_time, id_vars=['time'], var_name='user_type', value_name='user_count')

plt.figure(figsize = (10,20), num = 2)
plt.subplot(211)
sns.barplot(df_time_melt, x = 'time', y = 'user_count' , hue = 'user_type')

mean_casual = df_time_melt[df_time_melt['user_type'] == 'casual']['user_count'].mean()
plt.axhline(y=mean_casual, label='Mean Casual users', color='blue')
plt.annotate(f'{mean_casual:.0f}', xy=(0, mean_casual), xytext=(0, mean_casual+ 5000),
             arrowprops=dict(facecolor='blue'))

mean_registered = df_time_melt[df_time_melt['user_type'] == 'registered']['user_count'].mean()
plt.axhline(y=mean_registered, label='Mean Registered users', color='orange')
plt.annotate(f'{mean_registered:.0f}', xy=(0, mean_registered), xytext=(0, mean_registered + 5000),
             arrowprops=dict(facecolor='orange'))

mean_total = df_time_melt[df_time_melt['user_type'] == 'count']['user_count'].mean()
plt.axhline(y=mean_total, label='Mean of Total users', color='green')
plt.annotate(f'{mean_total:.0f}', xy=(0, mean_total), xytext=(0, mean_total + 5000),
             arrowprops=dict(facecolor='green'))

# plt.ylim(0, 250000)
plt.xticks(rotation = 45)
plt.legend(loc='upper center', bbox_to_anchor=(0.5, 1), ncol=3)

plt.subplot(212, aspect = 'equal')
plt.pie(df_time['count'], labels = df_time['time'],
        autopct = '%1.1f%%', pctdistance = 0.8,
        explode = [0.03 , 0.03, 0.015,0.015,0.015,0.015,0.015])
plt.show()
```


casual
registered
count
Mean Casual users
Mean Registered users
Mean of Total users
user_count
time
192094
137082
35012
Night
Early morning
Morning Rush Hour
Beforenoon
Afternoon
Evening rush hour
Early night


Beforenoon
8.6%
Morning Rush Hour
17.8%
Afternoon
22.6%
Early morning
2.1%
Night
2.9%
19.0%
Early night
27.1%
Evening rush hour

```
df.columns
```

```
Index(['date', 'time', 'holiday', 'workingday', 'season', 'weather', 'temp',
       'humidity', 'windspeed', 'casual', 'registered', 'count', 'datetime',
       'day_of_month', 'month', 'weekday'],
      dtype='object')
```

Q3. Check if there any significant difference between the no. of bike rides on Weekdays and Weekends?

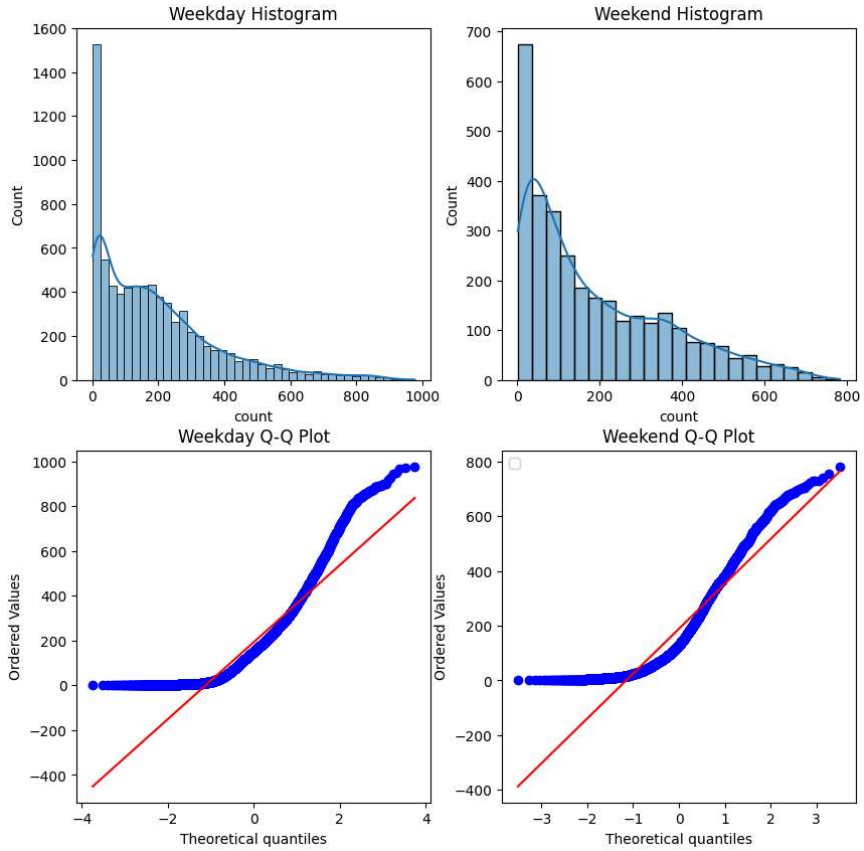
```
from scipy.stats import shapiro, f_oneway, levene
from scipy import stats
```

```
weekday = df[~df['weekday'].isin(['Saturday', 'Sunday'])]['count']
weekend = df[df['weekday'].isin(['Saturday', 'Sunday'])]['count']
(len(weekday),len(weekend))
```

```
(7723, 3163)
```

```
#Checking Normality of data with Histogram and QQplot

plt.figure(num = 4, figsize = (10,10))

plt.subplot(221)
sns.histplot(x = weekday, kde = True)
plt.title("Weekday Histogram")

plt.subplot(222)
sns.histplot(x = weekend, kde = True)
plt.title("Weekend Histogram")

plt.subplot(223)
stats.probplot(x = weekday, dist = 'norm', plot = plt)
plt.title("Weekday Q-Q Plot")

plt.subplot(224)
stats.probplot(x = weekend, dist = 'norm', plot = plt)
plt.title("Weekend Q-Q Plot")

plt.legend()
plt.show()
```

```
WARNING:matplotlib.legend:No artists with labels found to put in legend.  Note that a
```



From above the distribution is not Normal. Further we go for the Shapiro-Wilkins test for effective analysis.

```
H0_shapiro = 'The sample data is Normally Distributed.'
Ha_shapiro = 'The sample data is not Normally Distributed.'

alpha = 0.05

# Analysis for the Weekday sample.
test_stat, p_value = shapiro(weekday.sample(5000))

if p_value > alpha:
    print(f'For sample Weekday, {H0_shapiro}')
else:
    print(f'For sample Weekday, {Ha_shapiro}')

# Analysis for the Weekend sample.
test_stat, p_value = shapiro(weekend)

if p_value > alpha:
    print(f'For sample Weekend, {H0_shapiro}')
else:
    print(f'For sample Weekend, {Ha_shapiro}')
```

```
For sample Weekday, The sample data is not Normally Distributed.
For sample Weekend, The sample data is not Normally Distributed.
```

As the data is not normally distributed, we use boxcox method for converting the data to Normal Distribution and again testing the normality with Shapiro test.

```
#For Boxcox,
H0_shapiro = 'The sample data is Normally Distributed.'
Ha_shapiro = 'The sample data is not Normally Distributed.'

alpha = 0.05

# Analysis for the Weekday sample.
weekday_boxcox = stats.boxcox(weekday.sample(5000))[0]
test_stat, p_value = stats.shapiro(weekday_boxcox)

if p_value > alpha:
    print(f'For sample Weekday, {H0_shapiro}')
else:
    print(f'For sample Weekday, {Ha_shapiro}')

# Analysis for the Weekend sample.
weekend_boxcox = list(stats.boxcox(weekend))[0]
test_stat, p_value = stats.shapiro(weekend_boxcox)

if p_value > alpha:
    print(f'For sample Weekend, {H0_shapiro}')
else:
    print(f'For sample Weekend, {Ha_shapiro}')
```

```
For sample Weekday, The sample data is not Normally Distributed.
For sample Weekend, The sample data is not Normally Distributed.
```

The results after the boxcox normalization method the data is still not Normally Distributed. The T-Test Independent method cannot be applicable as the data is not Normal. After searching for the alternate test for comparing two independent samples. Use of ttest_ind with Welch's variation, i.e. Non equal variances and Mann-Whitney U test are found. Applying the ttest_ind as suggested in the Solution Approach.

```
H0 = 'The number of rides on Weekdays and Weekends are not affected by the nature of the day.' # Null Hypothesis
Ha = 'The number of rides are affected by the nature of the day i.e. Weekdays and Weekends.' #Alternate Hypothesis

tstat1,p_value1 = ttest_ind(weekday, weekend, equal_var = False)

tstat1,p_value1 = round(tstat1,3), round(p_value1,3)

print(f'The T-test statistic is', tstat1 , 'p_value is', p_value1)

if p_value1 > 0.05:
  print(f'The Null Hypothesis cannot be rejected.', H0)
else:
  print(f'The Null Hypothesis is Rejected.' , Ha)
```

```
The T-test statistic is 1.059 p_value is 0.29
The Null Hypothesis cannot be rejected. The number of rides on Weekdays and Weekends are not affected by the nature of the day.
```

```
from scipy.stats import mannwhitneyu
```

```
tstat1,p_value1 = mannwhitneyu(weekday, weekend)

tstat1,p_value1 = round(tstat1,3), round(p_value1,3)

print(f'The T-test statistic is', tstat1 , 'p_value is', p_value1)

if p_value1 > 0.05:
  print(f'The Null Hypothesis cannot be rejected.', H0)
else:
  print(f'The Null Hypothesis is Rejected.' , Ha)
```

```
The T-test statistic is 12198463.5 p_value is 0.917
The Null Hypothesis cannot be rejected. The number of rides on Weekdays and Weekends are not affected by the nature of the day.
```

Q4. Check if the demand of bicycles on rent is the same for different Weather conditions?

```
favorable = df[df['weather'] == 'favorable']['count']
mild= df[df['weather'] == 'mild']['count']
challenging = df[df['weather'] == 'challenging']['count']
unfavorable = df[df['weather'] == 'unfavorable']['count']
```

```
len(favorable),len(mild),len(challenging),len(unfavorable)
```

```
(7192, 2834, 859, 1)
```

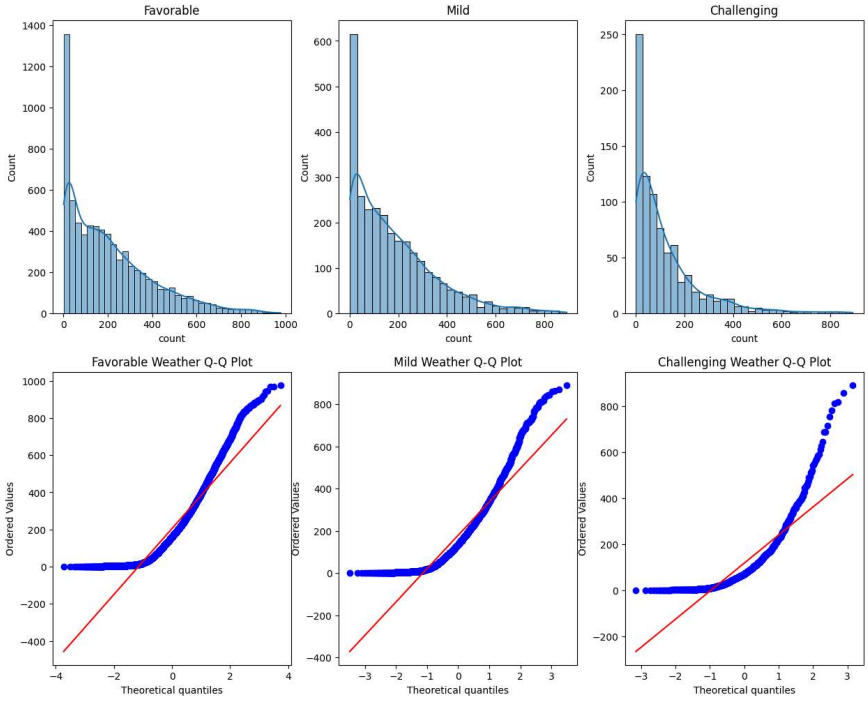
Given the disparate lengths of weather conditions, with only one data point in the case of unfavorable weather, this outlier condition can be disregarded as it might distort the statistical analysis. Consequently, unfavorable weather conditions are excluded from further analysis.

```
H0_weather = 'The condition of the weather is not affecting the number of rented bicycles.'
Ha_weather = 'The condition of the weather is affecting the number of rented bicycles.'

print(f'The Null Hypothesis is {H0_weather} & \nThe alternate hypothesis is {Ha_weather}')
```

```
The Null Hypothesis is The condition of the weather is not affecting the number of rented bicycles. &
The alternate hypothesis is The condition of the weather is affecting the number of rented bicycles.
```

To perform the statistical analysis the basic assumption is data is Normally Distributed. The Normality is to be checked with the Histograms and the QQplot.

```
plt.figure(num = 6, figsize = (15, 12))


plt.subplot(231)
sns.histplot(favorable, kde = True)
plt.title('Favorable')

plt.subplot(234)
stats.probplot(x = favorable, dist = 'norm', plot = plt)
plt.title("Favorable Weather Q-Q Plot")

plt.subplot(232)
sns.histplot(mild, kde = True)
plt.title('Mild')

plt.subplot(235)
stats.probplot(x = mild, dist = 'norm', plot = plt)
plt.title("Mild Weather Q-Q Plot")

plt.subplot(233)
sns.histplot(challenging, kde = True)
plt.title('Challenging')

plt.subplot(236)
stats.probplot(x = challenging, dist = 'norm', plot = plt)
plt.title("Challenging Weather Q-Q Plot")

plt.show()
```




```
 1 s1 = stats.skew(favorable)
 2 s2 = stats.skew(mild)
 3 s3 = stats.skew(challenging)
 4
 5 k1 = stats.kurtosis(favorable)
 6 k2 = stats.kurtosis(mild)
 7 k3 = stats.kurtosis(challenging)
 8
 9 print(s1,s2,s3)
10 print(k1,k2,k3)
```

```
1.1396195185041555 1.293759189703101 2.1833160390123187
0.9632151489948488 1.5835130178554868 5.961191782478394
```

Above Skewness and Kurtosis value indecates that all the samples are Rightly-skewed and the tails get heavier in weightage as the weather condition moves from Favorable to Challenging.

The data of all weather conditions are not normally distributed. Further Shapiro test to be performed to get the Normality of the data. The length of data for favorable weather to be sampled down to 5000 to conform to the limits of the Shapiro tests.

```
H0_shapiro = 'The sample data is Normally Distributed.'
Ha_shapiro = 'The sample data is not Normally Distributed.'

alpha = 0.05

# Analysis for the Favorable weather sample.
test_stat, p_value = shapiro(favorable.sample(5000))

if p_value > alpha:
    print(f'For sample Favorable weather, {H0_shapiro}')
else:
    print(f'For sample Favorable weather, {Ha_shapiro}')

# Analysis for the Mild weather sample.
test_stat, p_value = shapiro(mild)

if p_value > alpha:
    print(f'For sample Mild weather, {H0_shapiro}')
else:
    print(f'For sample Mild weather, {Ha_shapiro}')

# Analysis for the Challenging weather sample.
test_stat, p_value = shapiro(challenging)

if p_value > alpha:
    print(f'For sample Challenging weather, {H0_shapiro}')
else:
    print(f'For sample Challenging weather, {Ha_shapiro}')
```

```
For sample Favorable weather, The sample data is not Normally Distributed.
For sample Mild weather, The sample data is not Normally Distributed.
For sample Challenging weather, The sample data is not Normally Distributed.
```

As Shapiro test revealed the data samples are not Normally distributed. The Boxcox transformation is neede if the data can be normalised.

```
H0_shapiro = 'The sample data is Normally Distributed.'
Ha_shapiro = 'The sample data is not Normally Distributed.'

alpha = 0.05

# Analysis for the Favorable weather sample.
test_stat, p_value = shapiro(stats.boxcox(favorable.sample(5000))[0])

if p_value > alpha:
    print(f'For sample Favorable weather, {H0_shapiro}')
else:
    print(f'For sample Favorable weather, {Ha_shapiro}')

# Analysis for the Mild weather sample.
test_stat, p_value = shapiro(stats.boxcox(mild)[0])

if p_value > alpha:
    print(f'For sample Mild weather, {H0_shapiro}')
else:
    print(f'For sample Mild weather, {Ha_shapiro}')

# Analysis for the Challenging weather sample.
test_stat, p_value = shapiro(stats.boxcox(challenging)[0])

if p_value > alpha:
    print(f'For sample Challenging weather, {H0_shapiro}')
else:
    print(f'For sample Challenging weather, {Ha_shapiro}')
```

```
For sample Favorable weather, The sample data is not Normally Distributed.
For sample Mild weather, The sample data is not Normally Distributed.
For sample Challenging weather, The sample data is not Normally Distributed.
```

Even after using the Boxcox transformation the data is not Normally Distributed. This requires further investigation using the Levene's Test to perform the One-way ANOVA analysis.(As weather samples are greater than 2, Normal Statistical tests cannot be used.)

```
H0_Levene = 'The Data samples have Homogenous Variances.'
Ha_Levene = 'The Data samples do not have Homogenous Variances.'

alpha = 0.05

# The samples are resampled without bootstrapping to 500 length.

test_stat, p_value = levene(favorable.sample(500),
                            mild.sample(500),
                            challenging.sample(500))

# p_value = round(p_value,2)
print(f'p_value = {p_value}')

if p_value < alpha:
  print(f'The Null hypothesis is rejected and {Ha_Levene}')
else:
  print(f'The Null Hypothesis is failed to reject and {H0_Levene}')
```

```
p_value = 3.305236962075413e-11
The Null hypothesis is rejected and The Data samples do not have Homogenous Variances.
```

The two of the assumtions of the Oneway ANOVA test are violated which were,Normality of the Samples and the Homogenity of the Variances. The Need for the alternate method of statistical analysis is required.

As per the requirement of 3 sample statistical analysis and Non-Normal and Non-Homogenous test. The Kruskal-Wallis Test can be perfomed to test the Hypothesis.

```
kw_teststat , p_value_kw = stats.kruskal(favorable, mild, challenging)
print(f'The Test statistics is {kw_teststat}.\nThe P_value is {p_value_kw} ')

if p_value_kw < alpha:
  print(f'The Null Hypotesis is rejected. {Ha_weather}')
else:
  print(f'The Null Hypothesis cannot be rejected. {H0_weather}')
```

```
The Test statistics is 204.95566833068537.
The P_value is 3.122066178659941e-45
The Null Hypotesis is rejected. The condition of the weather is affecting the number of rented bicycles.
```

Q5. Check if the demand of bicycles on rent is the same for different Seasons?

```
df['season'].value_counts()
```

```
season
winter    2734
fall      2733
summer    2733
spring    2686
Name: count, dtype: int64
```

```
summer = df[df['season'] == 'summer']['count']
winter= df[df['season'] == 'winter']['count']
fall = df[df['season'] == 'fall']['count']
spring = df[df['season'] == 'spring']['count']
```

```
(len(summer),len(winter),len(fall),len(spring))
```

```
(2733, 2734, 2733, 2686)
```

As the samples are similar in size, all the samples are selected for the statistical analysis.

```
H0_season = 'The condition of the season is not affecting the number of rented bicycles.'
Ha_season = 'The condition of the season is affecting the number of rented bicycles.'

print(f'The Null Hypothesis is {H0_season} & \nThe alternate hypothesis is {Ha_season}')
```

```
The Null Hypothesis is The condition of the season is not affecting the number of rented bicycles. &
The alternate hypothesis is The condition of the season is affecting the number of rented bicycles.
```

```
# Checking the Normal Distribution of the samples using Histograms and QQplot.

plt.figure(num = 8, figsize = (15,8))

plt.subplot(241)

sns.histplot(summer,kde = True,bins = 40)
plt.title('Summer')
plt.subplot(242)
sns.histplot(winter,kde = True,bins = 40)
plt.title('Winter')
plt.subplot(243)
sns.histplot(fall,kde = True,bins = 40)
plt.title('Fall')
plt.subplot(244)
sns.histplot(spring,kde = True,bins = 40)
plt.title('Spring')
plt.subplot(245)
stats.probplot(x = summer, dist = 'norm', plot = plt)
plt.title("Summer")
plt.subplot(246)
stats.probplot(x = winter, dist = 'norm', plot = plt)
plt.title("Winter")
plt.subplot(247)
stats.probplot(x = fall, dist = 'norm', plot = plt)
plt.title("Fall")
plt.subplot(248)
stats.probplot(x = spring, dist = 'norm', plot = plt)
plt.title("Spring")
plt.show()
```