In Seoul there are 9.3 million inhabitants and they are all hard workers. They commute to work or spend time with friends and family and all either walk, take the Seoul Metropolitan Subway, drive their cars, or ride a bike. Here we find Seoul bike rental, an alternative to walking, driving, and the metro. The rental bike sector of the transportation sector has become a reliable method of transportation over the past few years and ridership has fluctuated by season.

We are looking for the best times to get more bike riders by having advertisements placed during prime hours during the day.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

bike = pd.read_csv('/content/Seoul bicycle rental dataset - CLEANED FOR DATE, CLEANED FOR HOUR, CLEANED FOR HOLIDAY, CLEANED FOR FUNCTIONING

def convert_date(date_str):
    try:
        return pd.to_datetime(date_str, format='%m/%d/%Y').strftime('%m/%d/%Y')
    except ValueError:
        try:
            return pd.to_datetime(date_str, format='%d/%m/%Y').strftime('%m/%d/%Y')
        except ValueError:
            return date_str

bike['Date'] = bike['Date'].apply(convert_date)

bike
print(bike['Date'].head())
```

```
0    10/04/2018
1    10/04/2018
2    10/04/2018
3    08/21/2018
4    08/01/2018
Name: Date, dtype: object
```

```
#bike.head(10)
bike.columns
```

```
Index(['Date', 'Rented Bike Count', 'Hour', 'Temperature(蚓)', 'Humidity(%)',
       'Wind speed (m/s)', 'Visibility (10m)', 'Dew point temperature(蚓)',
       'Solar Radiation (MJ/m2)', 'Rainfall(mm)', 'Snowfall (cm)', 'Seasons',
       'Holiday', 'Functioning Day', 'Day Of the Week', 'Day', 'Month', 'Year',
       'Date New', 'Holday New', 'New Functioning'],
      dtype='object')
```

rename columns

```
bike = bike.rename(columns={'Rented Bike Count' : 'rented_bike_count',
                            'Temperature(蚓)':'temperature(C)',
                            'Solar Radiation (MJ/m2)':'solar_radiation(MJ/m2)',
                            'Dew point temperature(蚓)': 'dew_point_temperature(C)',
                            'Snowfall (cm)' : 'snowfall(cm)',
                            'Functioning Day' : 'functioning_day',
                            'Wind speed (m/s)':'wind_speed(m/s)',
                            'Visibility (10m)':'visibility(10m)',
                            'Holday New':'holiday_new',
                            'New Functioning':'new_functioning'})

bike.columns
```

```
Index(['Date', 'rented_bike_count', 'Hour', 'temperature(C)', 'Humidity(%)',
       'wind_speed(m/s)', 'visibility(10m)', 'dew_point_temperature(C)',
       'solar_radiation(MJ/m2)', 'Rainfall(mm)', 'snowfall(cm)', 'Seasons',
       'Holiday', 'functioning_day', 'Day Of the Week', 'Day', 'Month', 'Year',
       'Date New', 'holiday_new', 'new_functioning'],
      dtype='object')
```

look for missing data

```
bike.isna().sum()
```

	0
Date	0
rented_bike_count	0
Hour	0
temperature(C)	0
Humidity(%)	0
wind_speed(m/s)	0
visibility(10m)	0
dew_point_temperature(C)	0
solar_radiation(MJ/m2)	0
Rainfall(mm)	0
snowfall(cm)	0
Seasons	0
Holiday	0
functioning_day	0
Day Of the Week	0
Day	0
Month	0
Year	0
Date New	0
holiday_new	0
new_functioning	0

dtype: int64

```
bike.duplicated().sum()
```

```
np.int64(0)
```

Fix and format dates

```
# Sample data (replace with your actual data)
data = {'date_col': ['01/02/2024', 'text', '15/03/2023', '31-12-1999']}
df = pd.DataFrame(data)

# Define format strings
old_format = '%d/%m/%y'
new_format = '%m/%d/%y'

def convert_date_format(date_str):
  # Try converting the date string to datetime object
  try:
    date_obj = pd.to_datetime(date_str, format=old_format)
    # If successful, convert to new format string and return
    return date_obj.strftime(new_format)
  except ValueError:
    # If conversion fails (not a valid date or format), return the original string
    return date_str

# Apply the function to the column using vectorized apply
bike['Date'] = bike['Date'].apply(convert_date_format)

# Print the modified DataFrame
bike.head(10)

#call chart where solar radiation is 1
#call chart where hour is 1 2 3 4 5 6 7 8 9 10 11 12
#call chart where visibility is 10m
```

Show hidden output

```
bike.head()
```

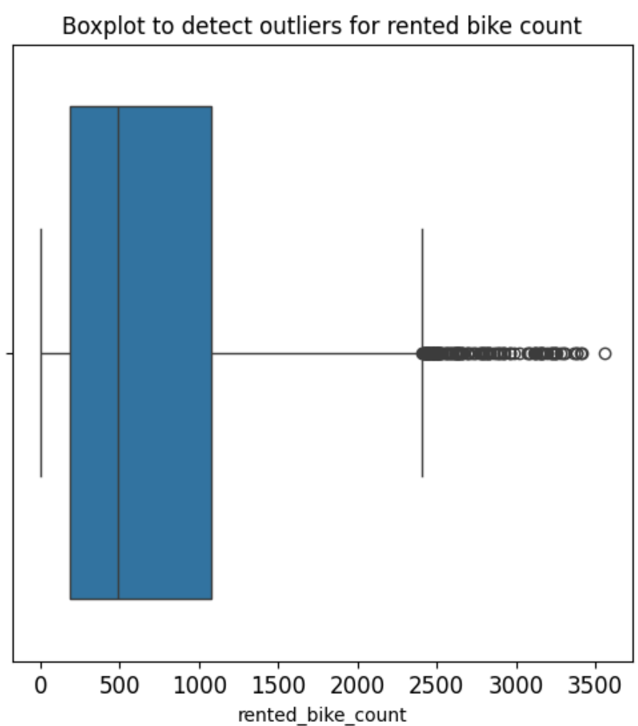
t	Hour	temperature(C)	Humidity(%)	wind_speed(m/s)	visibility(10m)	dew_point_temperature(C)	solar_ra
4	0	-5.2	37	2.2	2000	-17.6	
4	1	-5.5	38	0.8	2000	-17.6	
3	2	-6.0	39	1.0	2000	-17.7	
7	3	-6.2	40	0.9	2000	-17.6	
3	4	-6.0	36	2.3	2000	-18.6	

Next steps: Generate code with bike View recommended plots New interactive sheet

Check for Outliers

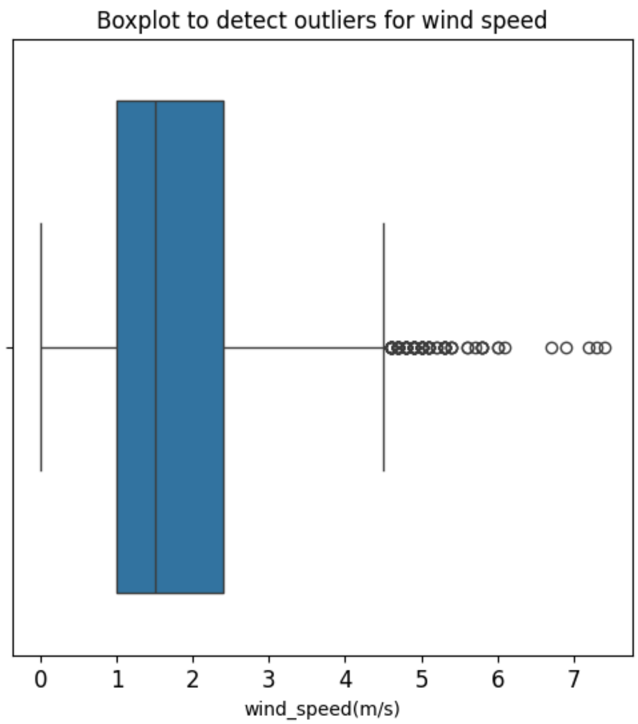
We find outliers in the rented bike count where there are odd instances of a bike rental.

```
#Create a boxplot to visualize distribution of 'rented_bike_count' and detect any outliers
plt.figure(figsize=(6,6))
plt.title('Boxplot to detect outliers for rented bike count')
plt.xticks(fontsize=12)
plt.yticks(fontsize=12)
sns.boxplot(x=bike['rented_bike_count'])
plt.show()
```

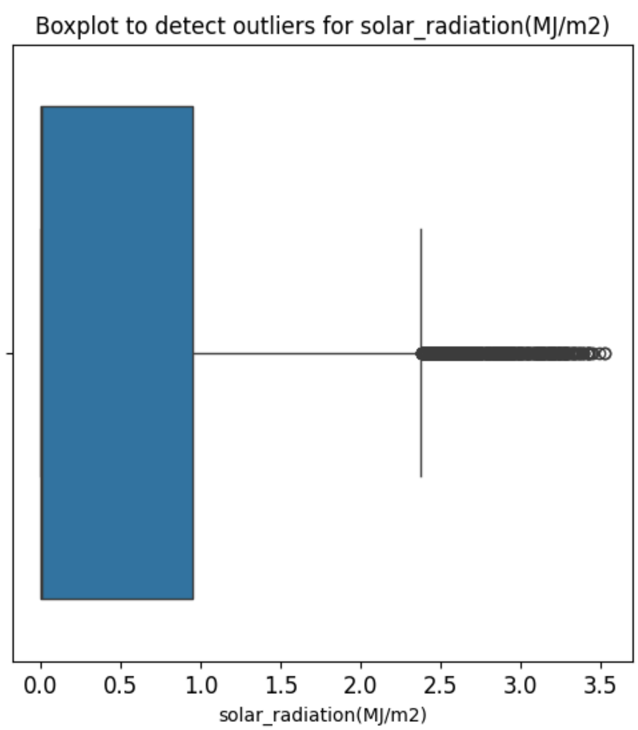



```
#Create a boxplot to visualize distribution of 'wind speed' and detect any outliers
plt.figure(figsize=(6,6))
```

```
plt.title('Boxplot to detect outliers for wind speed')
plt.xticks(fontsize=12)
plt.yticks(fontsize=12)
sns.boxplot(x=bike['wind_speed(m/s)'])
plt.show()
```




```
#Create a boxplot to visualize distribution of 'solar_radiation(MJ/m2)' and detect any outliers
plt.figure(figsize=(6,6))
plt.title('Boxplot to detect outliers for solar_radiation(MJ/m2)')
plt.xticks(fontsize=12)
plt.yticks(fontsize=12)
sns.boxplot(x=bike['solar_radiation(MJ/m2)'])
plt.show()
```



Below is a chart indicating Bike Rentals by season and temperature. We see that there are more rentals in the warmer seasons and some warmer days in fall where it tends to be cooler have similar ridership interest as if it were summer.

```
#Determine the number of rows containing outliers
percentile25 = bike['rented_bike_count'].quantile(0.25)
percentile75 = bike['rented_bike_count'].quantile(0.75)
iqr = percentile75 - percentile25
lower_bound = percentile25 - (1.5 * iqr)

#Define the upper limit and lower limit for non-outlier values in rented_bike_count
upper_limit = percentile75 + (1.5 * iqr)
lower_limit = percentile25 - (1.5 * iqr)
print("Lower limit:", lower_limit)
print("Upper limit:", upper_limit)

#Identify subset of data containing outliers in rented_bike_count
outliers = bike['Date'][(bike['rented_bike_count'] < lower_limit) | (bike['rented_bike_count'] > upper_limit)]

#count how many rows in the data contain outliers in rented_bike_count
print("Number of rows containing outliers in rented_bike_count:", len(outliers))
```

```
Lower limit: -1142.875
Upper limit: 2406.125
Number of rows containing outliers in rented_bike_count: 138
```

```
#Identify subset of data containing outliers in rented_bike_count
outliers = bike['Date'][(bike['wind_speed(m/s)'] < lower_limit) | (bike['rented_bike_count'] > upper_limit)]

#count how many rows in the data contain outliers in rented_bike_count
print("Number of rows containing outliers in wind_speed(m/s):", len(outliers))
```

```
Lower limit: -1.0999999999999996
Upper limit: 4.5
Number of rows containing outliers in wind_speed(m/s): 8023
```

```
#Identify subset of data containing outliers in rented_bike_count
outliers = bike['Date'][(bike['Hour'] < lower_limit) | (bike['Hour'] > upper_limit)]

#count how many rows in the data contain outliers in rented_bike_count
print("Number of rows containing outliers in Hour:", len(outliers))
```

```
Lower limit: -11.5
Upper limit: 34.5
Number of rows containing outliers in Hour: 0
```

```
bike.columns
```

```
Index(['Date', 'rented_bike_count', 'Hour', 'temperature(C)', 'Humidity(%)',
       'wind_speed(m/s)', 'visibility(10m)', 'dew_point_temperature(C)',
       'solar_radiation(MJ/m2)', 'Rainfall(mm)', 'snowfall(cm)', 'Seasons',
       'Holiday', 'functioning_day', 'Day Of the Week', 'Day', 'Month', 'Year',
       'Date New', 'holiday_new', 'new_functioning'],
      dtype='object')
```

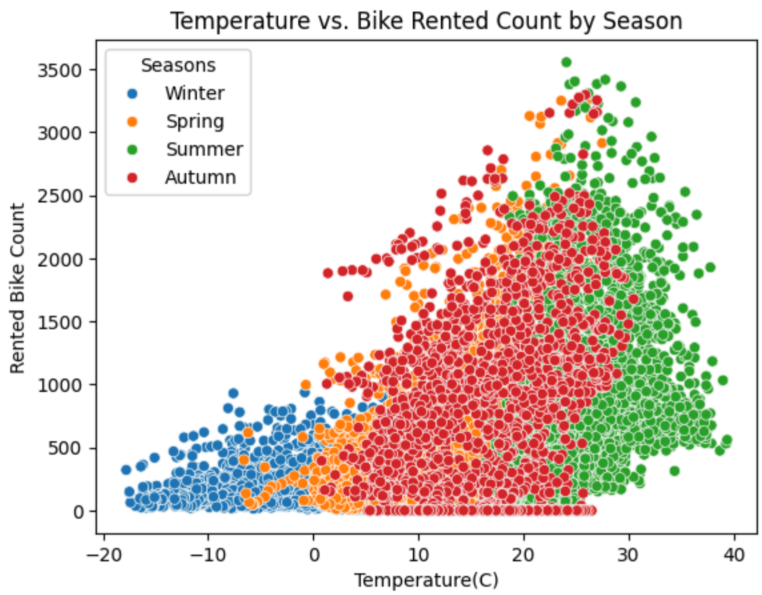
```
#Find out how many riders are by season
bike.groupby('Seasons')['rented_bike_count'].sum()
```

Seasons	rented_bike_count
Autumn	1486131
Spring	1611909
Summer	2283234
Winter	487169

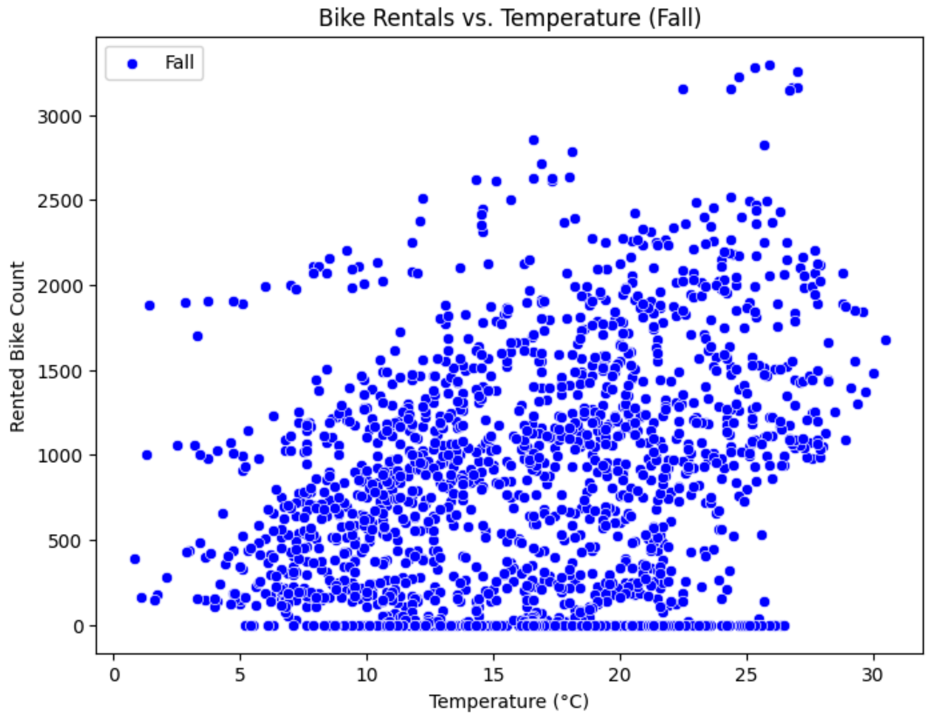
dtype: int64

We can check further for linearity and significance of the reasons to why riders chose a warmer temperature. Using P values, t-level, OLS, regression models, correlation, coefficient correlation, check for outliers, sns pairplot

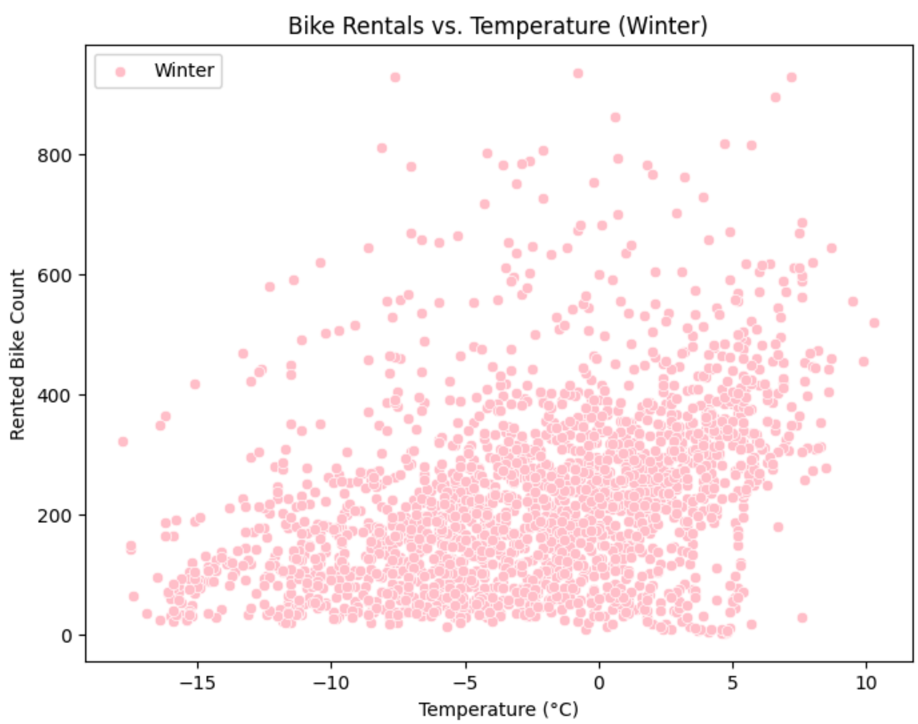
```
# Create a scatter plot with temperature on the x-axis,
# bike rented count on the y-axis, and different colors for each season.
sns.scatterplot(x='temperature(C)', y='rented_bike_count', hue='Seasons', data=bike)
plt.title('Temperature vs. Bike Rented Count by Season')
plt.xlabel('Temperature(C)')
plt.ylabel('Rented Bike Count')
plt.show()
```




```
import seaborn as sns
import matplotlib.pyplot as plt

# Filter data for Spring and Fall
fall_data = bike[bike['Seasons'] == 'Autumn']

#Filter Data for Spring
spring_data = bike[bike['Seasons'] == 'Spring']

# Filter data for Winter
winter_data = bike[bike['Seasons'] == 'Winter']

# Filter data for Summer
summer_data = bike[bike['Seasons'] == 'Summer']

# Create the first scatter plot for Fall
plt.figure(figsize=(8, 6))  # Adjust figure size if needed
sns.scatterplot(x='temperature(C)', y='rented_bike_count', data=fall_data, color='blue', label='Fall')
plt.title('Bike Rentals vs. Temperature (Fall)')
plt.xlabel('Temperature (°C)')
plt.ylabel('Rented Bike Count')
plt.legend(loc='upper left')
plt.show()
```

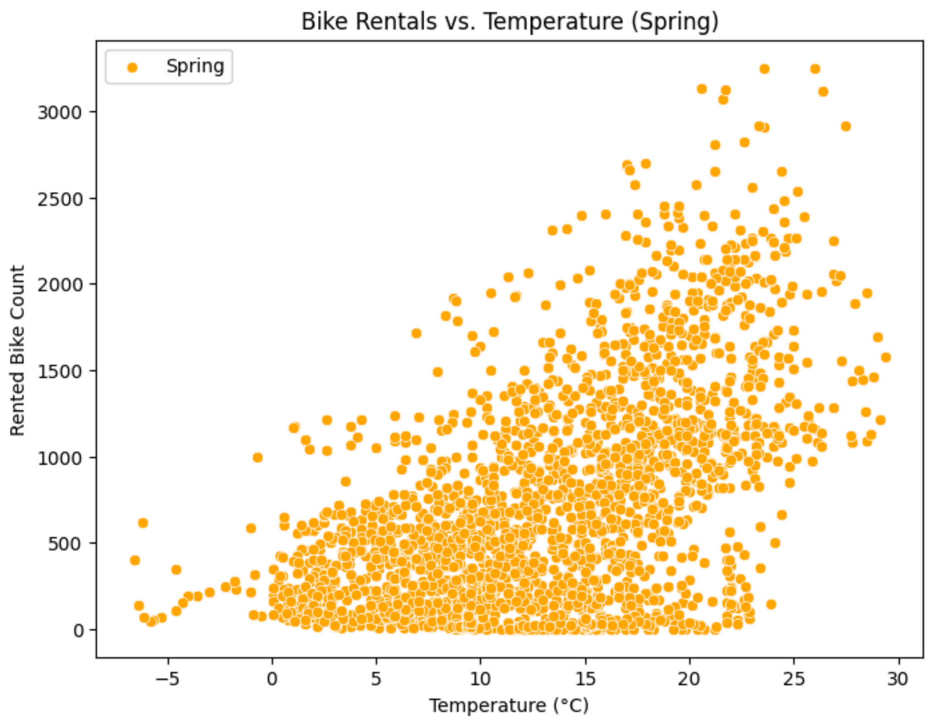



```
# Create the first scatter plot for Fall
plt.figure(figsize=(8, 6))  # Adjust figure size if needed
sns.scatterplot(x='temperature(C)', y='rented_bike_count', data=winter_data, color='pink', label='Winter')
plt.title('Bike Rentals vs. Temperature (Winter)')
plt.xlabel('Temperature (°C)')
plt.ylabel('Rented Bike Count')
#plot the legend on the right side
plt.legend(loc='upper left')
plt.show()
```

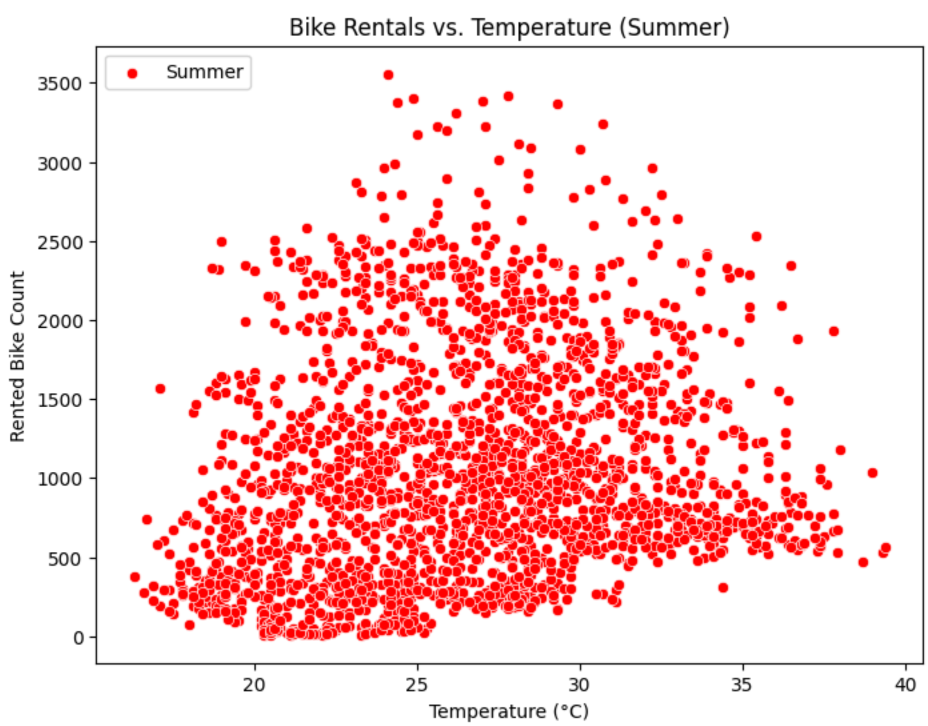



```
# Create the first scatter plot for Fall
plt.figure(figsize=(8, 6))  # Adjust figure size if needed
sns.scatterplot(x='temperature(C)', y='rented_bike_count', data=spring_data, color='orange', label='Spring')
plt.title('Bike Rentals vs. Temperature (Spring)')
plt.xlabel('Temperature (°C)')
plt.ylabel('Rented Bike Count')
```

```
plt.legend(loc = 'upper left')
plt.show()
```



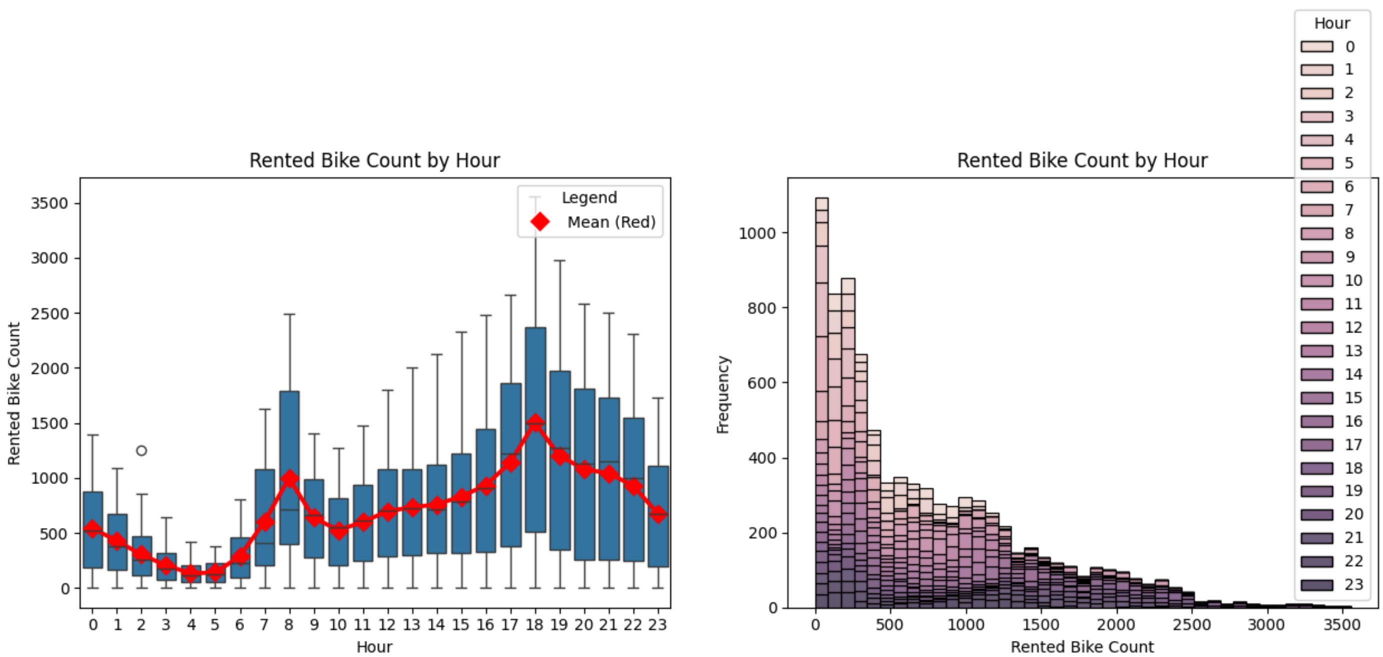

```
# Create the second scatter plot for Summer
plt.figure(figsize=(8, 6))  # Adjust figure size if needed
sns.scatterplot(x='temperature(C)', y='rented_bike_count', data=summer_data, color='red', label='Summer')
plt.title('Bike Rentals vs. Temperature (Summer)')
plt.xlabel('Temperature (°C)')
plt.ylabel('Rented Bike Count')
plt.legend(loc='upper left')
plt.show()
```



I have compiled data that represents the seasons of the year and their ridership. According to the data, temperature does play a huge role in bike rental along with the addition that school may be out during the summer months.

```
#create a plot as needed
### YOUR CODE HERE ###

#Set Figure and axes
fig, ax= plt.subplots(1,2, figsize=(15,5))

#Create boxplot showing 'Hours' distributions for 'rented_bike_count', comparing rented bikes by the hour
sns.boxplot(data=bike, x='Hour', y='rented_bike_count', ax=ax[0])
sns.pointplot(data=bike, x='Hour', y='rented_bike_count', ax=ax[0], color='red', markers='D', ci=None)
custom_legend = [plt.Line2D([0], [0], marker='D', color='w', markerfacecolor='red', markersize=10, label='Mean (Red)')]
ax[0].legend(handles=custom_legend, title='Legend', loc='upper right')


ax[0].set_title('Rented Bike Count by Hour')
ax[0].set_xlabel('Hour')
ax[0].set_ylabel('Rented Bike Count')

#Create histogram showing distribution of 'rented_bike_count', comparing hours bike rentals are high those hours which arent as many rentals
sns.histplot(data=bike, x='rented_bike_count', hue='Hour',
multiple='stack', ax=ax[1])
ax[1].set_title('Rented Bike Count by Hour')
ax[1].set_xlabel('Rented Bike Count')
ax[1].set_ylabel('Frequency')
```

```
<ipython-input-68-3b7fdb4a6cb5>:9: FutureWarning:

The `ci` parameter is deprecated. Use `errorbar=None` for the same effect.

  sns.pointplot(data=bike, x='Hour', y='rented_bike_count', ax=ax[0], color='red', markers='D', ci=None)
Text(0, 0.5, 'Frequency')
```




```
#Find the interquartile of the 17nth hour in hour with rented bike count
q1 = bike['rented_bike_count'].quantile(0.25)
q3 = bike['rented_bike_count'].quantile(0.75)
iqr = q3 - q1

# Filter the data to include only the 17th hour
hour_18_data = bike[bike['Hour'] == 18]

# Calculate the first quartile (Q1) for the 17th hour
q1_hour_18 = hour_18_data['rented_bike_count'].quantile(0.25)
```

```
# Calculate the third quartile (Q3) for the 17th hour
q3_hour_18 = hour_18_data['rented_bike_count'].quantile(0.75)

# Calculate the IQR for the 17th hour
iqr_hour_18 = q3_hour_18 - q1_hour_18

# Print the IQR for the 17th hour
print("Interquartile Range for the 17th hour:", iqr_hour_18)
```

```
Interquartile Range for the 17th hour: 1861.0
```

It is more frequent that bike rentals late at night and towards the evening hours are going home from an event at higher numbers, than bike rentals that are early in the day where there tends to be work. However, bike rentals are high for the 8 o'clock hour meaning riders enter their work with bikes rentals.

It might be natural that bike rentals are typically for people who go to work. This appears to be the case here, with the highest observations of rentals being at 8 and after the 18nth hour of the day.

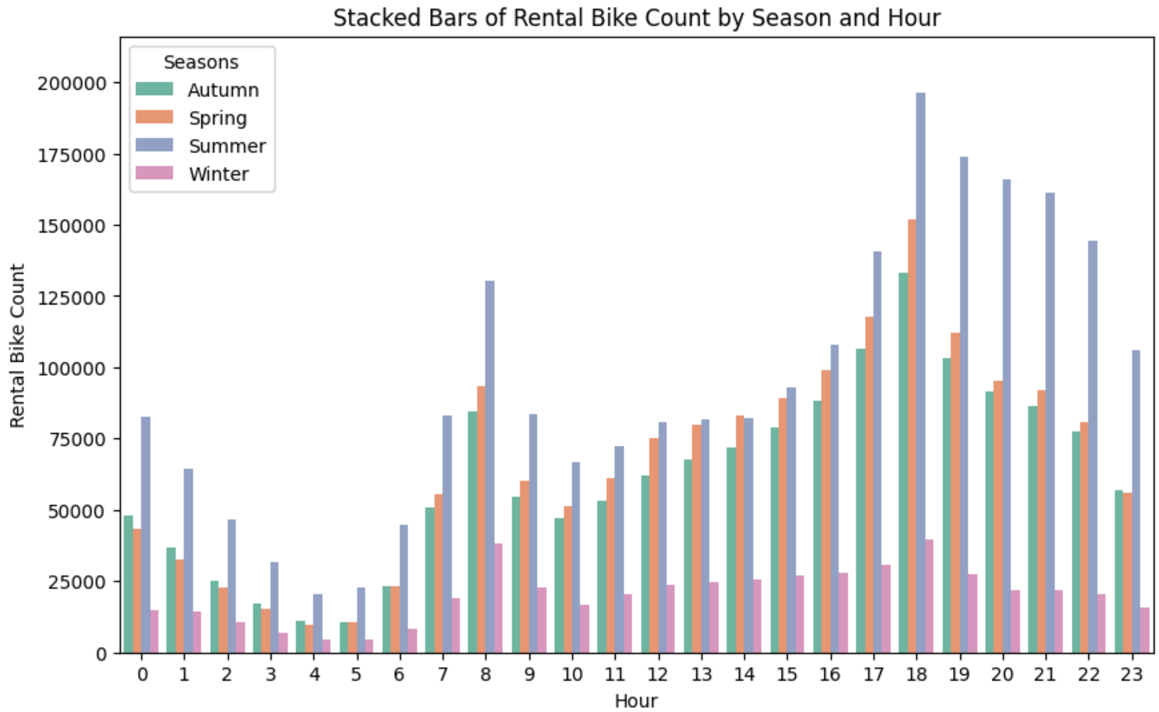
There are two groups of people renting bikes: (A) Those who go to work are found using the bikes at peak rush hour. It's safe to assume that these bike rentals are planned a day before where they would get bikes for rent to head to work. (B) Those who are riding for liesure or may have different work schedules ride consistently with daylight and may be returning clients or just having fun.

Everyone who is riding a rental bike is going somewhere, and the interquartile ranges of bike rentals at the 18nth hour of the day is 1861 rentals/year - much more than any other hour.

The optimal marketing hours would seem to be during the evening and hours before the 9-5 work day.

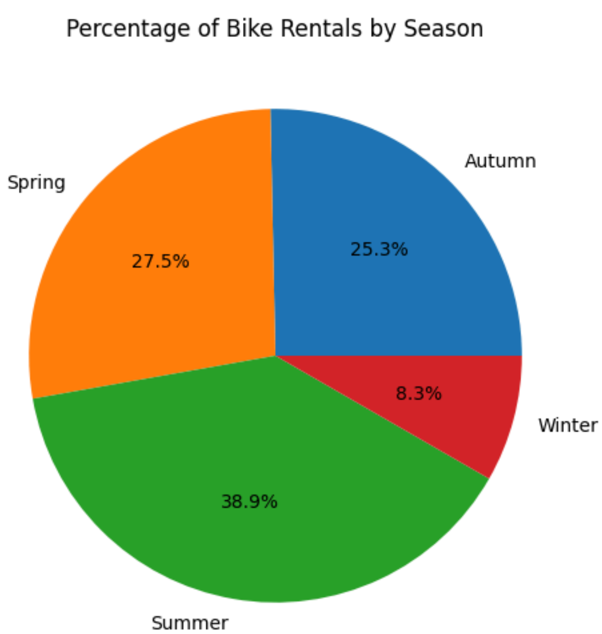
For the year 2018, on 06/19/2018, there was the highest rentals of 3556. And of course there are many days where there arent any rentals

```
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd  # Make sure you import pandas

# Assuming 'bike' is your DataFrame
# The following line performs the aggregation correctly
aggregated_counts = bike.groupby(['Hour', 'Seasons'])['rented_bike_count'].sum().reset_index()

plt.figure(figsize=(10, 6))
ax = sns.barplot(data=aggregated_counts, x='Hour', y='rented_bike_count', hue='Seasons', palette='Set2')

# Optional: Set y-axis limits based on the aggregated data
max_count = aggregated_counts['rented_bike_count'].max()
ax.set_ylim(0, max_count * 1.1)

plt.title('Stacked Bars of Rental Bike Count by Season and Hour')
plt.xlabel('Hour')
plt.ylabel('Rental Bike Count')
plt.legend(title='Seasons')
plt.show()
```



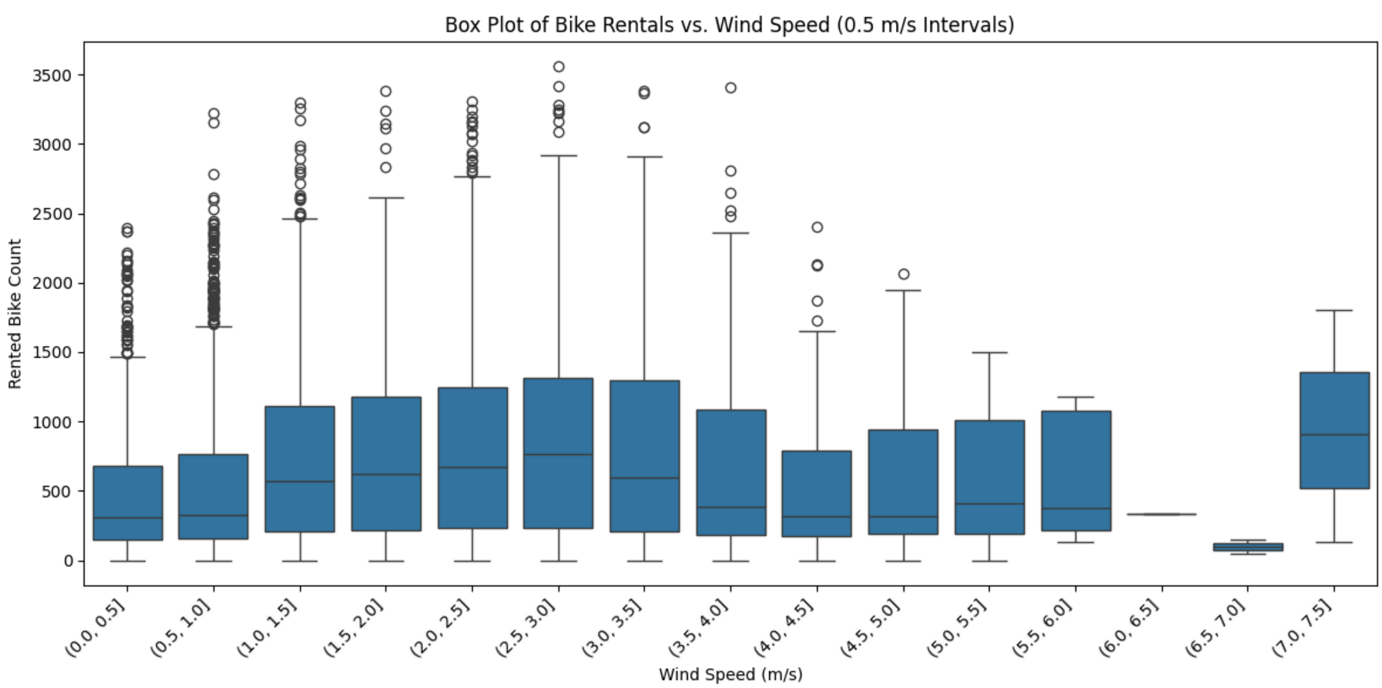
The stacked bar chart above indicated that throughout the four seasons, we see the 18th hour to be the highest time to the bike rental industry.

```
#Create a pie chart of the sum of rental_bike_Count by season, which slice of the pie is the biggest
#plt.figure(figsize=(10, 6))
#plt.pie(bike.groupby('Seasons')['rented_bike_count'].sum(), labels=bike['Seasons'].unique(), autopct='%1.1f%%')
#plt.title('Percentage of Bike Rentals by Season')
#plt.show()

seasonal_data = bike.groupby('Seasons')['rented_bike_count'].sum()
labels = seasonal_data.index
plt.figure(figsize=(10, 6))
plt.pie(seasonal_data.values, labels=labels, autopct='%1.1f%%')
plt.title('Percentage of Bike Rentals by Season')
plt.show()
```



This pie chart above further reinforced that we see the best season for bike rentals. It is indicated that Summer and Spring are the peak times for bike rentals

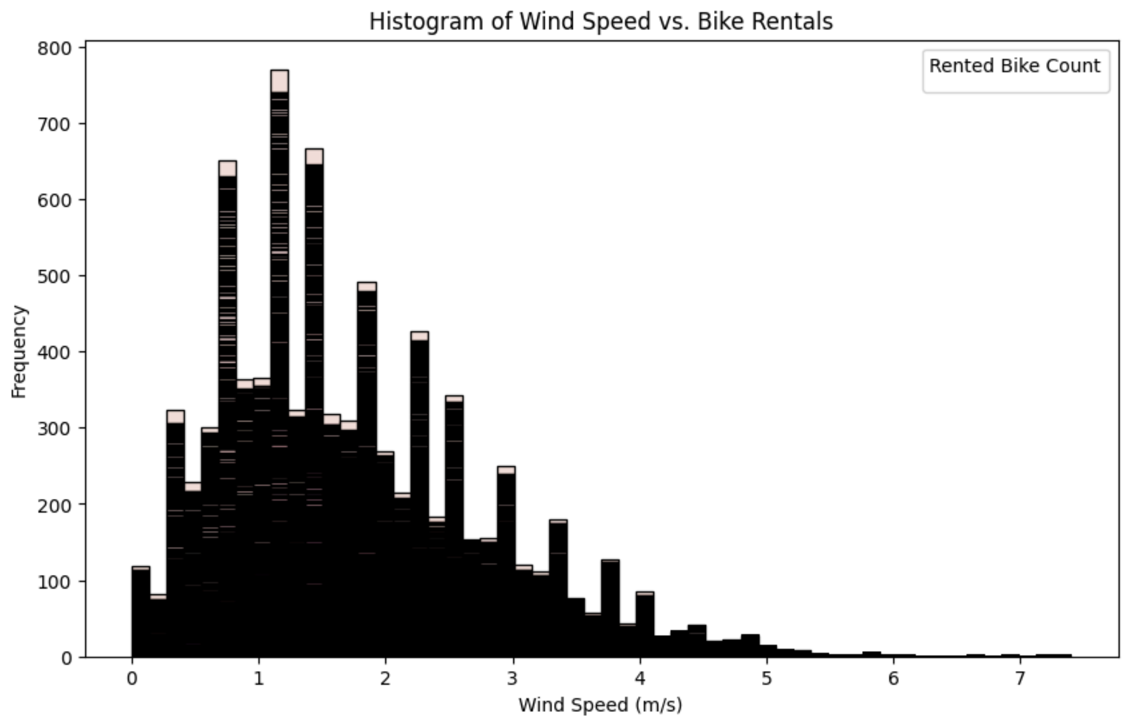
```
import seaborn as sns
import matplotlib.pyplot as plt

# Assuming 'bike' is your DataFrame

#plt.figure(figsize=(10, 6))  # Adjust figure size if needed
#sns.boxplot(x='wind_speed(m/s)', y='rented_bike_count', data=bike)
#plt.title('Box Plot of Bike Rentals vs. Wind Speed')
#plt.xlabel('Wind Speed (m/s)')
#plt.ylabel('Rented Bike Count')
#plt.show()

import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# Assuming 'bike' is your DataFrame

# Create bins for wind speed with intervals of 0.5 m/s
bike['wind_speed(m/s)'] = pd.cut(bike['wind_speed(m/s)'], bins=np.arange(0, bike['wind_speed(m/s)'].max() + 0.5, 0.5))

# Create the box plot with wind speed bins
plt.figure(figsize=(12, 6))  # Adjust figure size as needed
sns.boxplot(x='wind_speed(m/s)', y='rented_bike_count', data=bike)
plt.title('Box Plot of Bike Rentals vs. Wind Speed (0.5 m/s Intervals)')
plt.xlabel('Wind Speed (m/s)')
plt.ylabel('Rented Bike Count')
plt.xticks(rotation=45, ha='right')  # Rotate x-axis labels for better readability
plt.tight_layout()  # Adjust layout to prevent labels from overlapping
plt.show()
```



According to the boxplot above, windspeed is not a factor, even with winds of 7.5m/s, there are still significant ridership. However, residents may estimate some days to be too windy by the data from the 6-7m/s windspeed and may have indulged in a challenging bike ride on a 7m/s windy day.

```
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np

# ... (Your previous code) ...

# Create a histogram of windspeed(m/s) compared to rental_bike_count
plt.figure(figsize=(10, 6))

# Create the histplot with 'hue'
ax = sns.histplot(data=bike, x='wind_speed(m/s)', hue='rented_bike_count', multiple='stack')

# Get the handles and labels for the legend
handles, labels = ax.get_legend_handles_labels()

# Limit the legend to 10 entries
num_legend_entries = 10  # Or any number you desire
ax.legend(handles[:num_legend_entries], labels[:num_legend_entries], title='Rented Bike Count')

plt.title('Histogram of Wind Speed vs. Bike Rentals')
plt.xlabel('Wind Speed (m/s)')
plt.ylabel('Frequency')
plt.show()
```

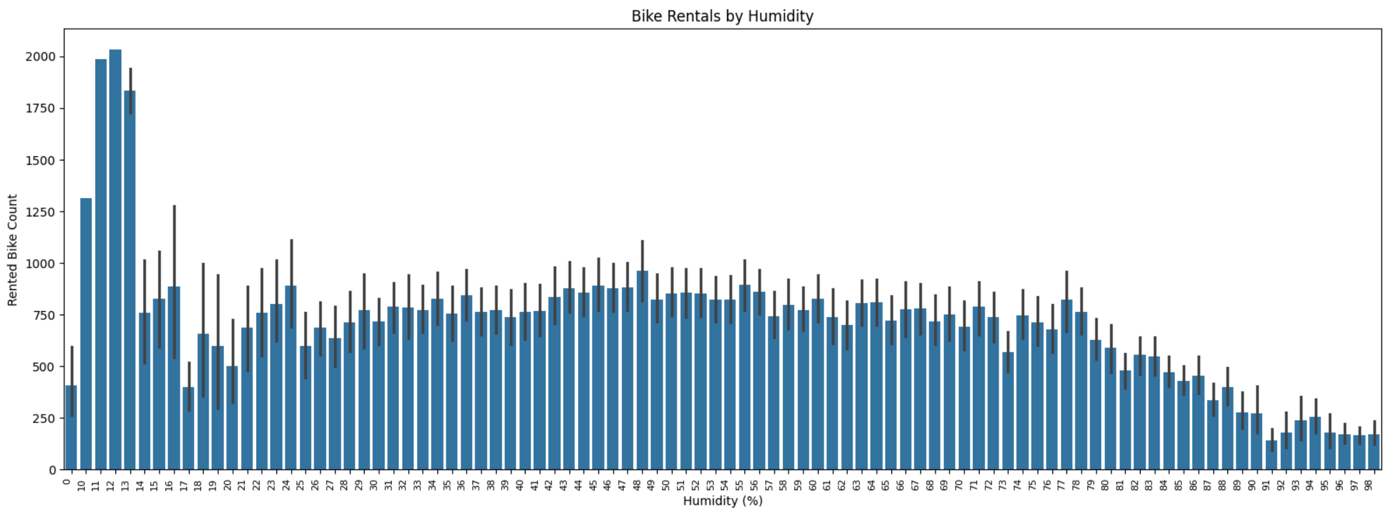
```
/usr/local/lib/python3.11/dist-packages/seaborn/distributions.py:267: PerformanceWarning: DataFrame is highly fragmented.  This is usual
  baselines[cols] = curves[cols].shift(1, axis=1).fillna(0)
/usr/local/lib/python3.11/dist-packages/IPython/core/pylabtools.py:151: UserWarning: Creating legend with loc="best" can be slow with la
  fig.canvas.print_figure(bytes_io, **kw)
```



This chart is showing that there was a case of at least 700-800 **instances** of bike rentals where the windspeed was 1-2 m/s. Bike rental clients enjoy the visibility more than low visibility

```
import seaborn as sns
import matplotlib.pyplot as plt
```

```
# Assuming 'bike' is your DataFrame with 'Humidity(%)' and 'rented_bike_count' columns

plt.figure(figsize=(16, 6))  # Increase the figure width to make the x-axis fit better
sns.barplot(x='Humidity(%)', y='rented_bike_count', data=bike)
plt.title('Bike Rentals by Humidity')
plt.xlabel('Humidity (%)')
plt.ylabel('Rented Bike Count')
plt.xticks(rotation=90, ha='right', fontsize=8) # Rotate x-axis labels for better readability and reduce font size
plt.tight_layout() # Adjust layout to prevent labels from overlapping
plt.show()
```

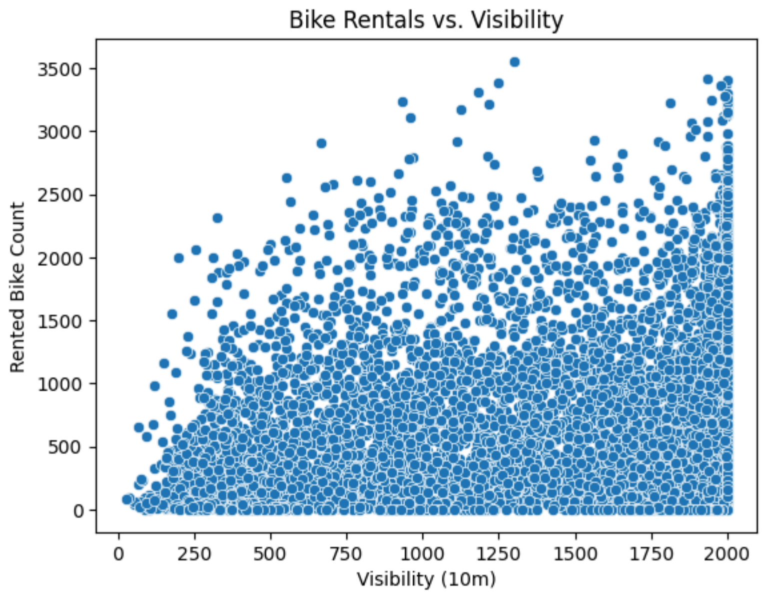


When we see low humidity, we see high bike rentals while humidity levels being higher result in lower bike rentals. Seoul bike riders tend to enjoy the drier days to maybe get a sweat.

```
#plot bike_rental_count and visibility(10m) to see what the resutls are in the best case for the data
plt.figure
sns.scatterplot(x='visibility(10m)', y='rented_bike_count', data=bike)
plt.title('Bike Rentals vs. Visibility')
plt.xlabel('Visibility (10m)')
plt.ylabel('Rented Bike Count')
plt.show
```

```
matplotlib.pyplot.show
def show(*args, **kwargs) -> None
```

```
Display all open figures.

Parameters
----------
block : bool, optional
    Whether to wait for all figures to be closed before returning.
```

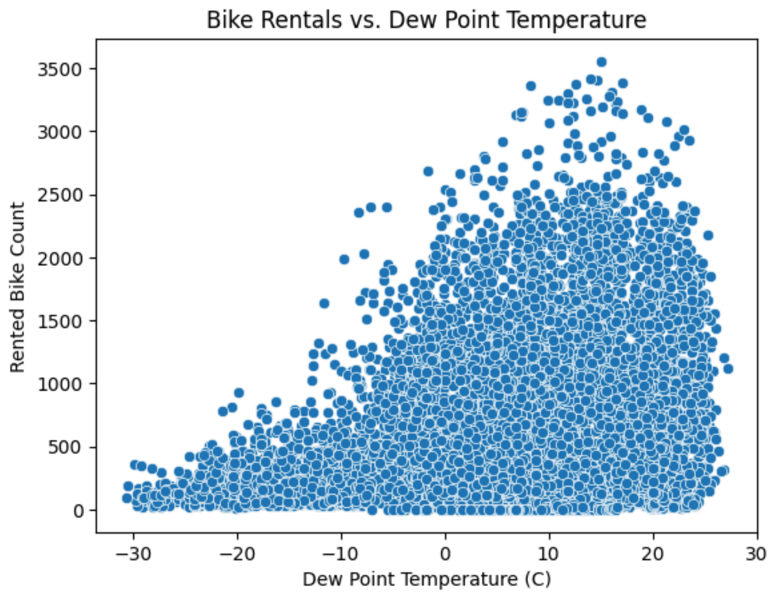


The best case scenario is visibility of 20000m and riders love weather conditions where they can see.

```
#plot dew_point_temperature(C) and rented_bike_count for what we need to see from the data
plt.figure
sns.scatterplot(x='dew_point_temperature(C)', y='rented_bike_count', data=bike)
plt.title('Bike Rentals vs. Dew Point Temperature')
plt.xlabel('Dew Point Temperature (C)')
plt.ylabel('Rented Bike Count')
plt.show
```

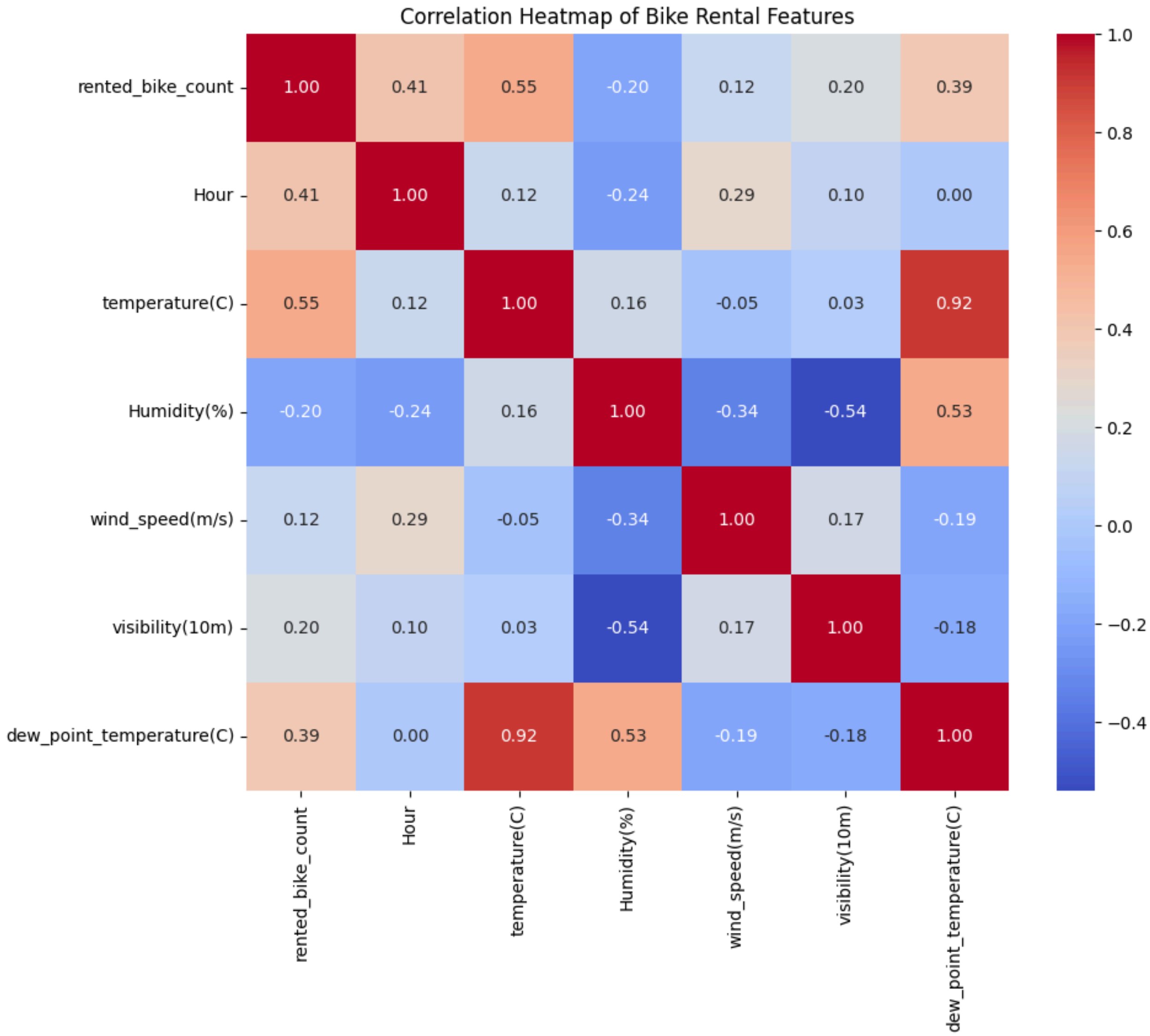
```
matplotlib.pyplot.show
def show(*args, **kwargs) -> None
```

```
Display all open figures.

Parameters
----------
block : bool, optional
    Whether to wait for all figures to be closed before returning.
```



Our team has researched what higher dew point temperatures means and it means the temperature at which the air needs to be cooled for saturation and condenses into dew, fog, or clouds. Higher dew points also mean warmer temperature. Riders enjoy the warmer temperatures more than the colder weather.

```
import seaborn as sns
import matplotlib.pyplot as plt

# Assuming 'bike' is your DataFrame
# Select the columns you want to include in the correlation matrix
columns_of_interest = ['rented_bike_count', 'Hour', 'temperature(C)', 'Humidity(%)',
                       'wind_speed(m/s)', 'visibility(10m)', 'dew_point_temperature(C)']

# Calculate the correlation matrix
correlation_matrix = bike[columns_of_interest].corr()

# Plot the heatmap
plt.figure(figsize=(10, 8))  # Adjust figure size if needed
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlation Heatmap of Bike Rental Features')
plt.show()
```



The heatmap indicates that hour, temperature, and dew point temperature have positive correlation with rented bike count and whether a rider decides to rent a bike is positively correlated with the temperature, hour, and dew point temperature.

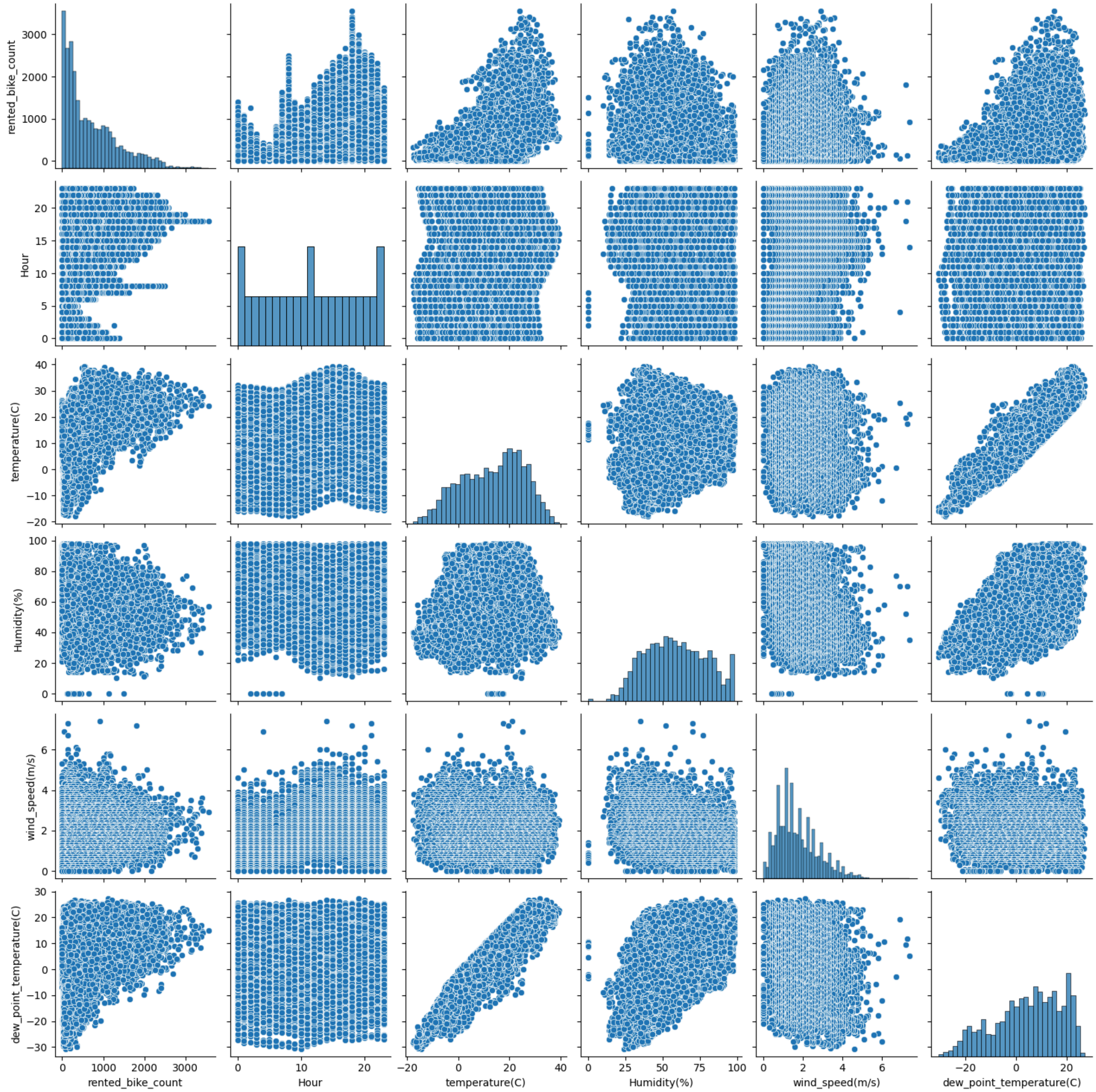
Insight

It appears that riders are riding bikes when the temperature is relatively warmer. On some colder days riders may have priorities or somewhere to be. Colder weather could be harsh and unpleasant for being outdoors. There is a sizeable group of riders who are in school, which is why we think the summer months are the most highest with rider base.

```
tocorr_data = bike[['rented_bike_count', 'Hour', 'temperature(C)', 'Humidity(%)',
                    'wind_speed(m/s)','dew_point_temperature(C)']]
```

```
sns.pairplot(data=tocorr_data)
```

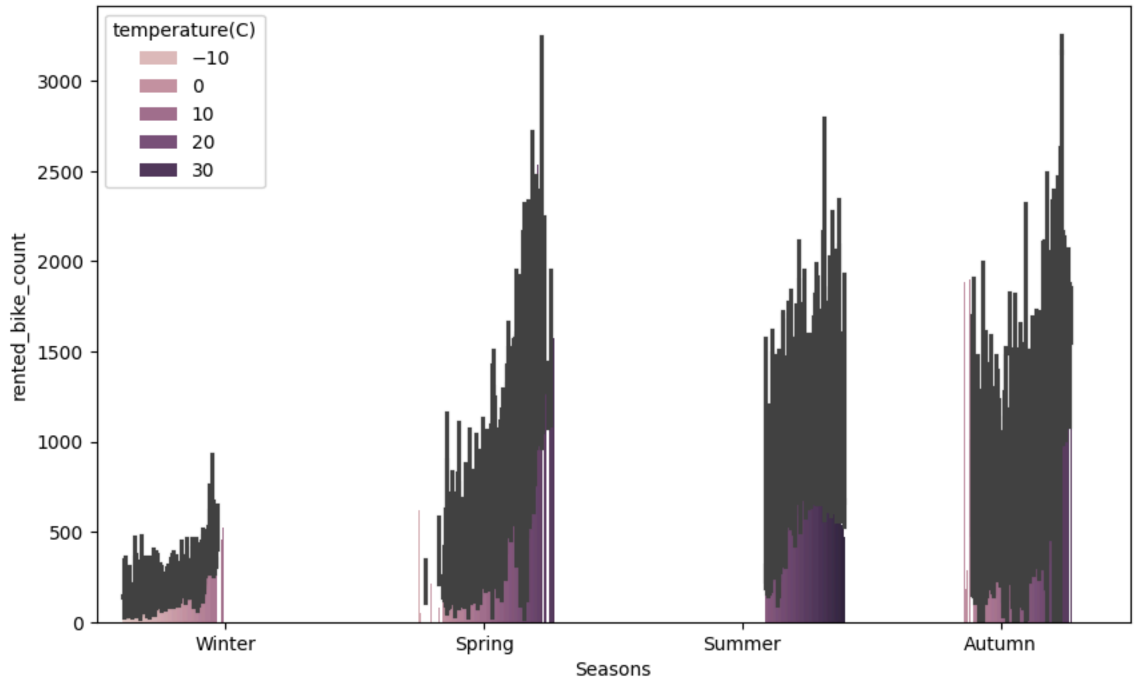
```
<seaborn.axisgrid.PairGrid at 0x7bf1e9ef0490>
```



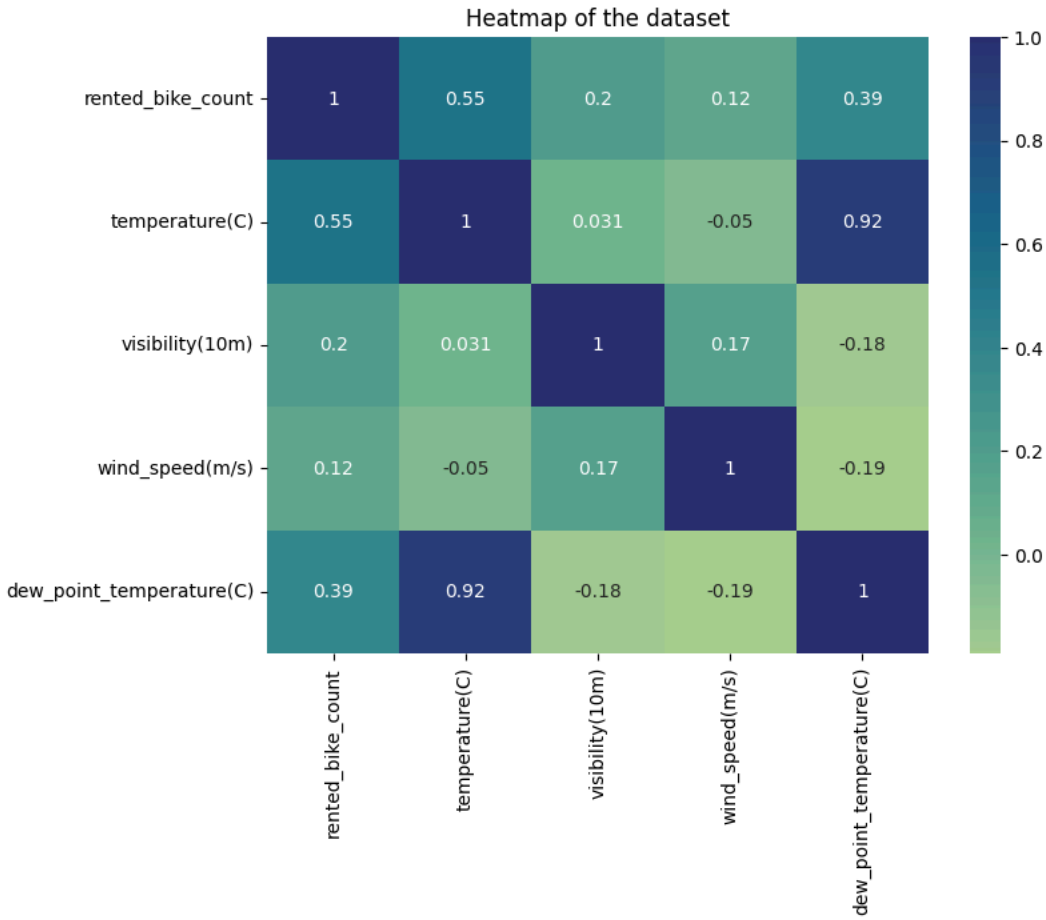
Here we can see various relationships between our data, most are irrelevant however, some metrics like dew point temperature and temperature are linear, which means that there is a high correlation between dew point temperature and temperature (nothing to do with rental bikes).

```
#create a stacked bar plot to visualize bike_rental_count comparing it to season and temperature
plt.figure(figsize=(10, 6))
sns.barplot(x='Seasons', y='rented_bike_count', hue='temperature(C)', data=bike)
```

```
<Axes: xlabel='Seasons', ylabel='rented_bike_count'>
```

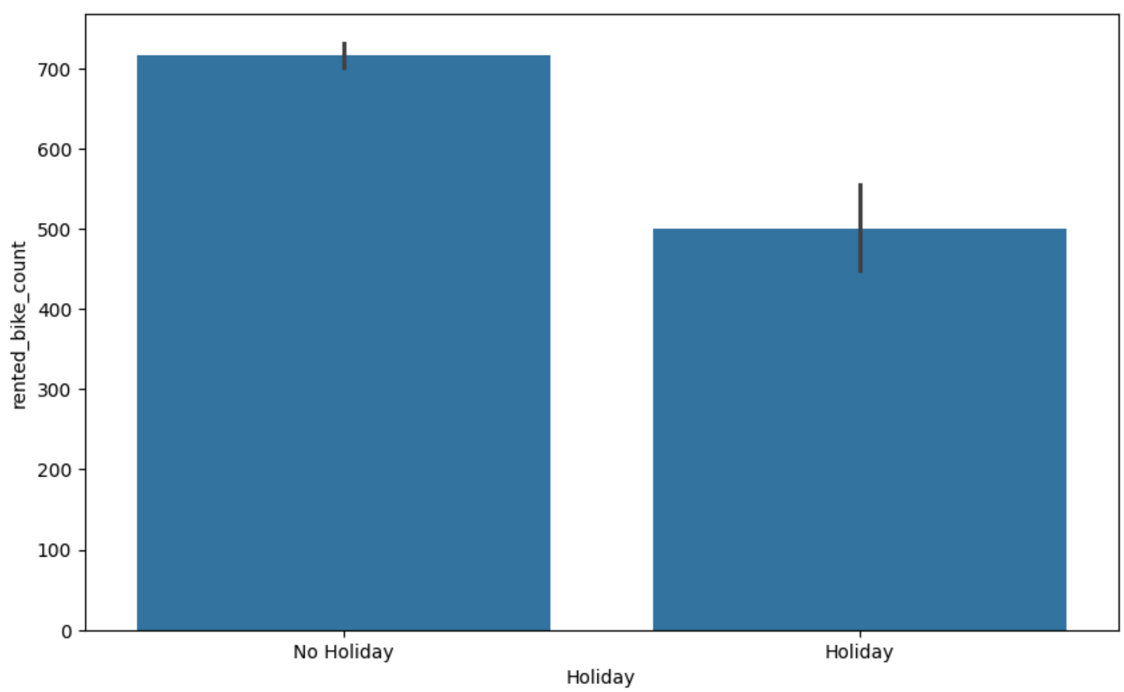



```
#create a heatmap to visualize how correlated variables are
plt.figure(figsize=(8,6))
sns.heatmap(df_bike[['rented_bike_count', 'temperature(C)','visibility(10m)','wind_speed(m/s)','dew_point_temperature(C)',]]
            .corr(), annot=True, cmap="crest")
plt.title('Heatmap of the dataset')
plt.show()
```




```
#Plot the sum of rented_bike_count on holiday or no holiday
plt.figure(figsize=(10, 6))
sns.barplot(x='Holiday', y='rented_bike_count', data=bike)
```

```
<Axes: xlabel='Holiday', ylabel='rented_bike_count'>
```

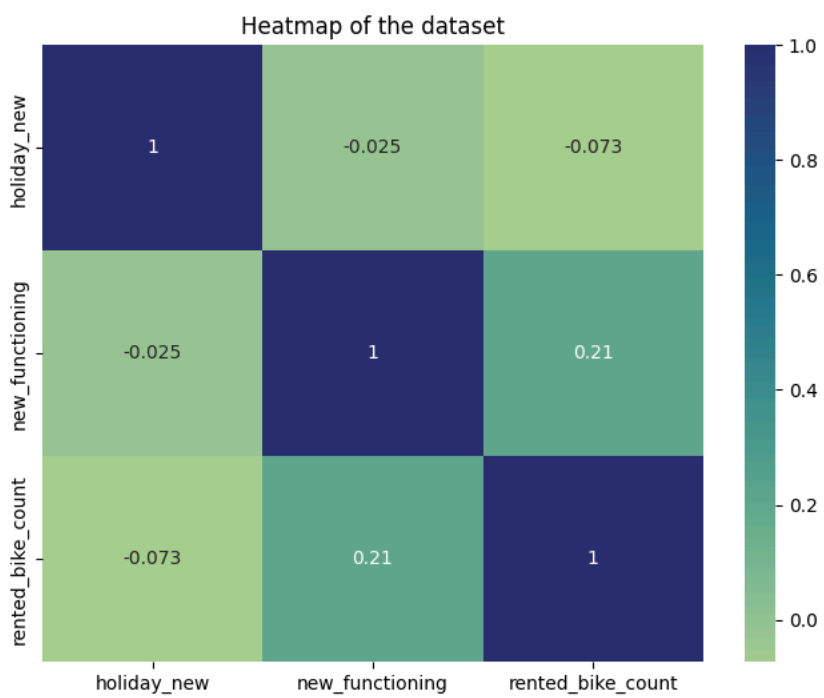



```
#in the dataset change the holiday values from 0 and 1 where no holiday is 0 and holiday is 1 for the entire column
bike.head()
```

m/s)	visibility(10m)	dew_point_temperature(C)	solar_ra
7.4	1992	5.1	
7.3	1634	11.9	
7.2	2000	9.5	
6.9	925	19.4	
6.7	692	-2.8	

We can see that there are some riders on holiday who rent bikes, but fewer than those without holiday leave.

```
plt.figure(figsize=(8,6))
sns.heatmap(bike[['holiday_new','new_functioning','rented_bike_count']].corr(), annot=True, cmap="crest")
plt.title('Heatmap of the dataset')
plt.show()
```




```
bike.columns
```

```
Index(['rented_bike_count', 'Hour', 'temperature(C)', 'Humidity(%)',
       'wind_speed(m/s)', 'visibility(10m)', 'dew_point_temperature(C)',
       'solar_radiation(MJ/m2)', 'Rainfall(mm)', 'snowfall(cm)',
       ...
       'Date New_9/28/2018', 'Date New_9/29/2018', 'Date New_9/3/2018',
       'Date New_9/30/2018', 'Date New_9/4/2018', 'Date New_9/5/2018',
       'Date New_9/6/2018', 'Date New_9/7/2018', 'Date New_9/8/2018',
       'Date New_9/9/2018'],
      dtype='object', length=724)
```

We can see there is a positive correlation for if the bike is available and works that affects riders decision to take the rental bike.

```
!pip install scikit-learn
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
```

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report

# Assuming 'bike' is your DataFrame

# Preprocessing: Ensure 'new_functioning' is binary and one-hot encode categorical features
# Check unique values in 'new_functioning' before mapping
print(bike['new_functioning'].unique())
# Adjust mapping to include all possible values, and use correct values
bike['new_functioning'] = bike['new_functioning'].map({'Yes': 1, 'No': 0})
# Check unique values again after mapping
print(bike['new_functioning'].unique())


# Filter for Variability (this step is likely unnecessary now)
# df_model = bike[bike['new_functioning'].isin([0, 1])]  # This line might be redundant
df_model = bike  # Use the full DataFrame

# Print the shape of df_model to check if it's empty
print(df_model.shape)

# Separate features (X) and target (y)
X = df_model.drop('new_functioning', axis=1)
y = df_model['Hour']

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, stratify=y, random_state=42)

# Decision Tree Model
decision_tree = DecisionTreeClassifier(random_state=42)
decision_tree.fit(X_train, y_train)
y_pred_dt = decision_tree.predict(X_test)

# Evaluate Decision Tree
accuracy_dt = accuracy_score(y_test, y_pred_dt)
print("Decision Tree Accuracy:", accuracy_dt)
print(classification_report(y_test, y_pred_dt))


# Random Forest Model
random_forest = RandomForestClassifier(random_state=42)
random_forest.fit(X_train, y_train)
y_pred_rf = random_forest.predict(X_test)

# Evaluate Random Forest
accuracy_rf = accuracy_score(y_test, y_pred_rf)
print("\nRandom Forest Accuracy:", accuracy_rf)
print(classification_report(y_test, y_pred_rf))
```

```
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.11/dist-packages (1.6.1)
Requirement already satisfied: numpy>=1.19.5 in /usr/local/lib/python3.11/dist-packages (from scikit-learn) (2.0.2)
Requirement already satisfied: scipy>=1.6.0 in /usr/local/lib/python3.11/dist-packages (from scikit-learn) (1.14.1)
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.11/dist-packages (from scikit-learn) (1.4.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/python3.11/dist-packages (from scikit-learn) (3.6.0)
[nan]
[nan]
(8328, 5743)
Decision Tree Accuracy: 1.0
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        86
           1       1.00      1.00      1.00        87
           2       1.00      1.00      1.00        87
           3       1.00      1.00      1.00        87
           4       1.00      1.00      1.00        87
           5       1.00      1.00      1.00        87
           6       1.00      1.00      1.00        87
           7       1.00      1.00      1.00        87
           8       1.00      1.00      1.00        86
           9       1.00      1.00      1.00        86
          10       1.00      1.00      1.00        87
          11       1.00      1.00      1.00        86
          12       1.00      1.00      1.00        87
          13       1.00      1.00      1.00        87
          14       1.00      1.00      1.00        87
          15       1.00      1.00      1.00        87
          16       1.00      1.00      1.00        86
          17       1.00      1.00      1.00        87
          18       1.00      1.00      1.00        86
```

```
          19       1.00      1.00      1.00        87
          20       1.00      1.00      1.00        87
          21       1.00      1.00      1.00        87
          22       1.00      1.00      1.00        87
          23       1.00      1.00      1.00        87

    accuracy                           1.00      2082
   macro avg       1.00      1.00      1.00      2082
weighted avg       1.00      1.00      1.00      2082


Random Forest Accuracy: 0.5629202689721422
              precision    recall  f1-score   support

           0       0.82      0.90      0.86        86
           1       0.59      0.67      0.62        87
           2       0.61      0.64      0.63        87
           3       0.59      0.56      0.58        87
           4       0.55      0.51      0.53        87
           5       0.42      0.54      0.47        87
           6       0.49      0.51      0.50        87
           7       0.55      0.40      0.46        87
           8       0.58      0.67      0.62        86
           9       0.56      0.69      0.62        86
```