Open in Colab

Project Name - Bike Sharing Demand Prediction

Project Type - EDA/Regression

Contribution - Individual

Name- Pawan Kumar Singh

Project Summary -

Currently Rental bikes are introduced in many urban cities for the enhancement of mobility comfort. It is important to make the rental bike available and accessible to the public at the right time as it lessens the waiting time. Eventually, providing the city with a stable supply of rental bikes becomes a major concern. The crucial part is the prediction of bike count required at each hour for the stable supply of rental bikes.

GitHub Link -

https://github.com/Pawanme9034/Bike_Sharing_Demand_Prediction-Capstone_Project/blob/main/Bike_Sharing_Demand_Prediction_Capstone_ProjectML.ipynb

Problem Statement

Problem Statement is the prediction of bike count required at each hour for the stable supply of rental bikes.

Let's Begin !

1. Know Your Data

Import Libraries

In [1]:

```
#Let's import the modules
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns

from datetime import datetime
import datetime as dt

from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import PowerTransformer
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import MultiLabelBinarizer

from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor

from sklearn.model_selection import train_test_split
from sklearn.model_selection import RandomizedSearchCV
from sklearn.model_selection import cross_validate
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.model_selection import RandomizedSearchCV

from sklearn.decomposition import PCA

from sklearn import metrics
from sklearn.metrics import r2_score
from sklearn.metrics import mean_squared_error
from sklearn.metrics import accuracy_score
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import log_loss

from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline

import warnings
warnings.filterwarnings('ignore')
```

Dataset Loading

In [2]:
```
# Load Dataset
import requests
from io import StringIO
# uploading data through Github directly
url = "https://raw.githubusercontent.com/Pawanme9034/Bike_Sharing_Demand_Prediction-Capstone_Project/main/SeoulBikeData.c
headers = {"User-Agent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10.14; rv:66.0) Gecko/20100101 Firefox/66.0"}
req = requests.get(url, headers=headers)
data = StringIO(req.text)

bike_df=pd.read_csv(data)
```

Dataset First View

In [3]:
```
# interactive tabel (this code for google colab only)
try:
    from google.colab import data_table
    data_table.enable_dataframe_formatter()
    from vega_datasets import data

    # Dataset First Look
    data_table.DataTable(bike_df)
except Exception as e:
    print("An error occurred:", e)
```

```
An error occurred: No module named 'google.colab'
```

In [4]:
```
# Dataset First Look
bike_df.head()
```

Out[4]:

	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	H
0	01/12/2017	254	0	-5.2	37	2.2	2000	-17.6	0.0	0.0	0.0	Winter	
1	01/12/2017	204	1	-5.5	38	0.8	2000	-17.6	0.0	0.0	0.0	Winter	

	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	I
2	01/12/2017	173	2	-6.0	39	1.0	2000	-17.7	0.0	0.0	0.0	Winter	
3	01/12/2017	107	3	-6.2	40	0.9	2000	-17.6	0.0	0.0	0.0	Winter	
4	01/12/2017	78	4	-6.0	36	2.3	2000	-18.6	0.0	0.0	0.0	Winter	

Dataset Rows & Columns count

In [5]:
```
# Dataset Rows & Columns count
bike_df.shape
```

Out[5]:
```
(8760, 14)
```

Dataset Information

In [6]:
```
# Dataset Info
bike_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8760 entries, 0 to 8759
Data columns (total 14 columns):
 #   Column                     Non-Null Count  Dtype  
---  ------                     --------------  -----  
 0   Date                       8760 non-null   object 
 1   Rented Bike Count          8760 non-null   int64  
 2   Hour                       8760 non-null   int64  
 3   Temperature(°C)            8760 non-null   float64
 4   Humidity(%)                8760 non-null   int64  
 5   Wind speed (m/s)           8760 non-null   float64
 6   Visibility (10m)           8760 non-null   int64  
 7   Dew point temperature(°C)  8760 non-null   float64
 8   Solar Radiation (MJ/m2)    8760 non-null   float64
 9   Rainfall(mm)               8760 non-null   float64
```

```
 10  Snowfall (cm)              8760 non-null   float64
 11  Seasons                    8760 non-null   object
 12  Holiday                    8760 non-null   object
 13  Functioning Day            8760 non-null   object
dtypes: float64(6), int64(4), object(4)
memory usage: 958.2+ KB
```

Duplicate Values

In [7]:
```
# Dataset Duplicate Value Count
value=len(bike_df[bike_df.duplicated()])
value
```

Out[7]:
```
0
```

Missing Values/Null Values

In [8]:
```
# Missing Values/Null Values Count
bike_df.isnull().sum()
```

Out[8]:
```
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
dtype: int64
```

What did you know about your dataset?

1. This Dataset contains 8760 lines and 14 columns.
2. the data is orgenized and there are timestamp.
3. there are no missing or null values in dataset.

4. dtypes: float64(6), int64(4), object(4)
5. memory usage: 848.3+ KB

2. Understanding Your Variables

In [9]:
```
# Dataset Columns
bike_df.columns
```

Out[9]:
```
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'],
      dtype='object')
```

In [10]:
```
# Dataset Describe
bike_df.describe()
```

Out[10]:

	Rented Bike Count	Hour	Temperature(C)	Humidity(%)	Wind speed (m/s)	Visibility (10m)	Dew point temperature(C)	Solar Radiation (MJ/m2)	Rainfall(mm)	Snc
count	8760.000000	8760.000000	8760.000000	8760.000000	8760.000000	8760.000000	8760.000000	8760.000000	8760.000000	8760.0(
mean	704.602055	11.500000	12.882922	58.226256	1.724909	1436.825799	4.073813	0.569111	0.148687	0.0
std	644.997468	6.922582	11.944825	20.362413	1.036300	608.298712	13.060369	0.868746	1.128193	0.4
min	0.000000	0.000000	-17.800000	0.000000	0.000000	27.000000	-30.600000	0.000000	0.000000	0.0(
25%	191.000000	5.750000	3.500000	42.000000	0.900000	940.000000	-4.700000	0.000000	0.000000	0.0(
50%	504.500000	11.500000	13.700000	57.000000	1.500000	1698.000000	5.100000	0.010000	0.000000	0.0(
75%	1065.250000	17.250000	22.500000	74.000000	2.300000	2000.000000	14.800000	0.930000	0.000000	0.0(
max	3556.000000	23.000000	39.400000	98.000000	7.400000	2000.000000	27.200000	3.520000	35.000000	8.8(

Variables Description

Date : *The date of the day, during 365 days from 01/12/2017 to 30/11/2018, formating in DD/MM/YYYY, type : str*, we need to convert into datetime format.

Rented Bike Count : *Number of rented bikes per hour which our dependent variable and we need to predict that, type : int*

Hour: *The hour of the day, starting from 0-23 it's in a digital time format, type : int, we need to convert it into category data type.*

Temperature(°C): *Temperature in Celsius, type : Float*

Humidity(%): *Humidity in the air in %, type : int*

Wind speed (m/s) : *Speed of the wind in m/s, type : Float*

Visibility (10m): *Visibility in m, type : int*

Dew point temperature(°C): *Temperature at the beggining of the day, type : Float*

Solar Radiation (MJ/m2): *Sun contribution, type : Float*

Rainfall(mm): *Amount of raining in mm, type : Float*

Snowfall (cm): *Amount of snowing in cm, type : Float*

Seasons: *Season of the year, type : str, there are only 4 season's in data .*

Holiday: *If the day is holiday period or not, type: str*

Functioning Day: *If the day is a Functioning Day or not, type : str italicized text* **bold text**

Check Unique Values for each variable.

In [11]:
```
# Check Unique Values for each variable.
print(bike_df.apply(lambda col: col.unique()))
```

```
Date                          [01/12/2017, 02/12/2017, 03/12/2017, 04/12/201...
Rented Bike Count             [254, 204, 173, 107, 78, 100, 181, 460, 930, 4...
Hour                          [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13,...
Temperature(�C)               [-5.2, -5.5, -6.0, -6.2, -6.4, -6.6, -7.4, -7....
Humidity(%)                   [37, 38, 39, 40, 36, 35, 27, 24, 21, 23, 25, 2...
Wind speed (m/s)              [2.2, 0.8, 1.0, 0.9, 2.3, 1.5, 1.3, 1.1, 0.5, ...
Visibility (10m)              [2000, 1928, 1996, 1936, 793, 1913, 1687, 1380...
Dew point temperature(�C)     [-17.6, -17.7, -18.6, -18.7, -19.5, -19.3, -19...
Solar Radiation (MJ/m2)       [0.0, 0.01, 0.23, 0.65, 0.94, 1.11, 1.16, 1.01...
```

```
Rainfall(mm)                [0.0, 0.5, 1.0, 2.5, 0.1, 0.2, 0.3, 0.7, 1.6, ...
Snowfall (cm)               [0.0, 0.1, 0.3, 0.4, 0.2, 1.0, 0.9, 0.8, 0.7, ...
Seasons                                  [Winter, Spring, Summer, Autumn]
Holiday                                        [No Holiday, Holiday]
Functioning Day                                            [Yes, No]
dtype: object
```

In [12]:

```
#print the unique value
bike_df.nunique()
```

Out[12]:

```
Date                          365
Rented Bike Count            2166
Hour                           24
Temperature(�C)               546
Humidity(%)                    90
Wind speed (m/s)               65
Visibility (10m)             1789
Dew point temperature(�C)     556
Solar Radiation (MJ/m2)       345
Rainfall(mm)                   61
Snowfall (cm)                  51
Seasons                         4
Holiday                         2
Functioning Day                 2
dtype: int64
```

3. *Data Wrangling*

Data Wrangling Code

In [13]:

```
# Write your code to make your dataset analysis ready.
#Rename the complex columns name
bike_df=bike_df.rename(columns={'Rented Bike Count':'Rented_Bike_Count',
                                'Temperature(°C)':'Temperature',
                                'Temperature(�C)':'Temperature',
                                'Humidity(%)':'Humidity',
                                'Wind speed (m/s)':'Wind_speed',
                                'Visibility (10m)':'Visibility',
                                'Dew point temperature(°C)':'Dew_point_temperature',
                                'Dew point temperature(�C)':'Dew_point_temperature',
                                'Solar Radiation (MJ/m2)':'Solar_Radiation',
                                'Rainfall(mm)':'Rainfall',
```

```
                                'Snowfall (cm)':'Snowfall',
                                'Functioning Day':'Functioning_Day'})
```

In [14]:
```
# checking columns names
bike_df.columns
```

Out[14]:
```
Index(['Date', 'Rented_Bike_Count', 'Hour', 'Temperature', 'Humidity',
       'Wind_speed', 'Visibility', 'Dew_point_temperature', 'Solar_Radiation',
       'Rainfall', 'Snowfall', 'Seasons', 'Holiday', 'Functioning_Day'],
      dtype='object')
```

In [15]:
```
# Convert 'Date' column to datetime format
bike_df['Date'] = bike_df['Date'].apply(lambda x:dt.datetime.strptime(x,"%d/%m/%Y"))
```

In [16]:
```
# Changing the "Date" column into three "year","month","day" column
# Extract year, month, and day of the week
bike_df['year'] = bike_df['Date'].dt.year
bike_df['month'] = bike_df['Date'].dt.month
bike_df['day'] = bike_df['Date'].dt.day_name()
```

In [17]:
```
#creating a new column of "weekdays_weekend" and drop the column "Date","day","year"
bike_df['weekdays_weekend']=bike_df['day'].apply(lambda x : 1 if x=='Saturday' or x=='Sunday' else 0 )
# Drop 'Date', 'day', and 'year' columns
bike_df=bike_df.drop(columns=['Date','day','year'],axis=1)
```

In [18]:
```
# Get value counts of 'weekdays_weekend' column
bike_df['weekdays_weekend'].value_counts()
```

Out[18]:
```
0    6264
1    2496
Name: weekdays_weekend, dtype: int64
```

- So we convert the "date" column into 3 different column i.e "year","month","day"
- The "year" column in our data set is basically contain the 2 unique number contains the details of from 2017 december to 2018 november so if i consider this is a one year then we don't need the "year" column so we drop it.
- The other column "day", it contains the details about the each day of the month, for our relevence we don't need each day of each month data but we need the data about, if a day is a weekday or a weekend so we convert it into this format and drop the "day"

column

In [19]:
```
#Change the int64 column into catagory column
cols=['Hour','month','weekdays_weekend']
for col in cols:
  bike_df[col]=bike_df[col].astype('category')
```

In [20]:
```
# assign the numerical columns of the DataFrame 'bike_df' to a variable,
# Select numerical columns
numerical_columns=list(bike_df.select_dtypes(['int64','float64']).columns)
numerical_features=pd.Index(numerical_columns)
numerical_features
```

Out[20]:
```
Index(['Rented_Bike_Count', 'Temperature', 'Humidity', 'Wind_speed',
       'Visibility', 'Dew_point_temperature', 'Solar_Radiation', 'Rainfall',
       'Snowfall'],
      dtype='object')
```

What all manipulations have you done and insights you found?

1. Rename the complex columns name
2. Changing the "Date" column into three "year","month","day" column
3. Convert 'Date' column to datetime format
4. Extract year, month, and day of the week
5. creating a new column of "weekdays_weekend" and drop the column "Date","day","year"
6. Drop 'Date', 'day', and 'year' columns
7. Change the int64 column into catagory column
8. assign the numerical columns of the DataFrame 'bike_df' to a variable,

Sure! Here's a summary of the manipulation code you provided previously:

1. Extracting Year, Month, and Day:

   - You added three columns ('year', 'month', 'day') to the DataFrame 'bike_df'.
   - The 'year' column contains the year extracted from the 'Date' column using the 'dt.year' property.
   - The 'month' column contains the month extracted from the 'Date' column using the 'dt.month' property.

- The 'day' column contains the day of the week (e.g., 'Sunday', 'Monday') extracted from the 'Date' column using the 'dt.day_name()' method.

2. Converting Date column to datetime format:

- You used the 'apply()' function with a lambda function to convert the 'Date' column to a datetime format.
- The lambda function utilized 'strptime()' from the 'datetime' module to parse each date string according to the format "%d/%m/%Y".
- The resulting datetime objects were assigned back to the 'Date' column.

3. Determining Weekdays vs. Weekends:

- You added a new column called 'weekdays_weekend' to the DataFrame 'bike_df'.
- The values in the 'weekdays_weekend' column were determined using the 'apply()' function and a lambda function.
- If the 'day' value was 'Saturday' or 'Sunday', the corresponding value in the 'weekdays_weekend' column was set to 1; otherwise, it was set to 0.

4. Dropping Columns:

- You dropped the 'Date', 'day', and 'year' columns from the DataFrame 'bike_df' using the 'drop()' function.
- The 'columns' parameter was used to specify the names of the columns to be dropped, and the 'axis' parameter was set to 1 to indicate column-wise operation.

5. Renaming Complex Column Names:

- You renamed the complex column names in the DataFrame 'bike_df' to make them more analysis-ready.
- The 'rename()' function was used with a dictionary mapping the current column names to the desired new column names.
- The column names that were renamed included 'Rented Bike Count', 'Temperature(°C)', 'Humidity(%)', 'Wind speed (m/s)', 'Visibility (10m)', 'Dew point temperature(°C)', 'Solar Radiation (MJ/m2)', 'Rainfall(mm)', 'Snowfall (cm)', and 'Functioning Day'.

These manipulations demonstrate various data preprocessing steps, such as extracting date components, converting data types, creating derived features, and renaming columns.

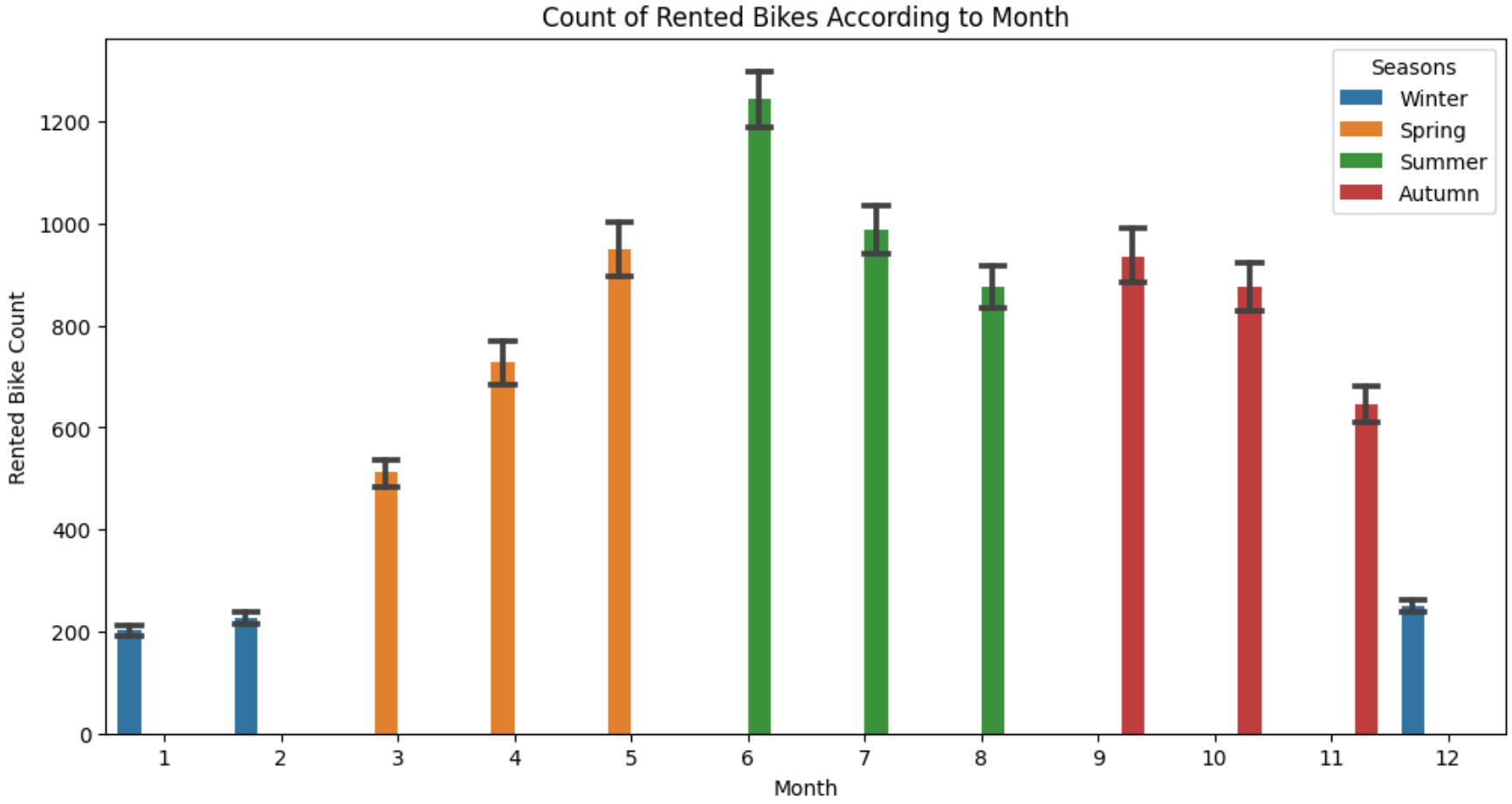
4. Data Vizualization, Storytelling & Experimenting with charts : Understand the relationships between variables

Chart - 1

Rented bikes according to the month

In [21]:

```
# Chart - 1 visualization code
fig, ax = plt.subplots(figsize=(12, 6))
sns.barplot(data=bike_df, x='month', y='Rented_Bike_Count', ax=ax, capsize=.2, hue='Seasons')
ax.set_title('Count of Rented Bikes According to Month')
ax.set_xlabel('Month')
ax.set_ylabel('Rented Bike Count')

plt.show()
```



1. Why did you pick the specific chart?

The bar plot was chosen because it allows for a clear comparison of the count of rented bikes across different months. The categorical nature of the months is well-suited for representation on the x-axis, while the numerical count of rented bikes is represented on the y-axis.

The use of different colors for each season aids in visual comparison. The bar plot enables the identification of any seasonal patterns or variations in bike rental demand and facilitates easy interpretation of the data. Overall, it is an effective and concise way to convey the count of rented bikes for each month and analyze trends over time.

2. What is/are the insight(s) found from the chart?

Seasonal Variations: The average count of rented bikes varies across seasons. Summer months (June, July, and August) have the highest average count, while winter months have relatively lower counts.

Monthly Variations: Within each season, there are variations in the average count of rented bikes across different months.

Winter Season: The winter season generally exhibits lower average counts of rented bikes, suggesting reduced demand during colder months.

3. Will the gained insights help creating a positive business impact?

Are there any insights that lead to negative growth? Justify with specific reason.

The gained insights can positively impact the business by informing decisions related to seasonal demand planning, marketing strategies, and pricing. Understanding the seasonal and monthly variations in bike rental demand allows businesses to optimize resource allocation, target promotions effectively, and adjust pricing strategies. However, the lower average counts during the winter season may pose a temporary negative impact. Nonetheless, this insight can also present opportunities for diversification or focusing on alternative revenue streams during colder months. It is crucial for businesses to analyze these insights in their specific context to make informed decisions and drive positive business outcomes.

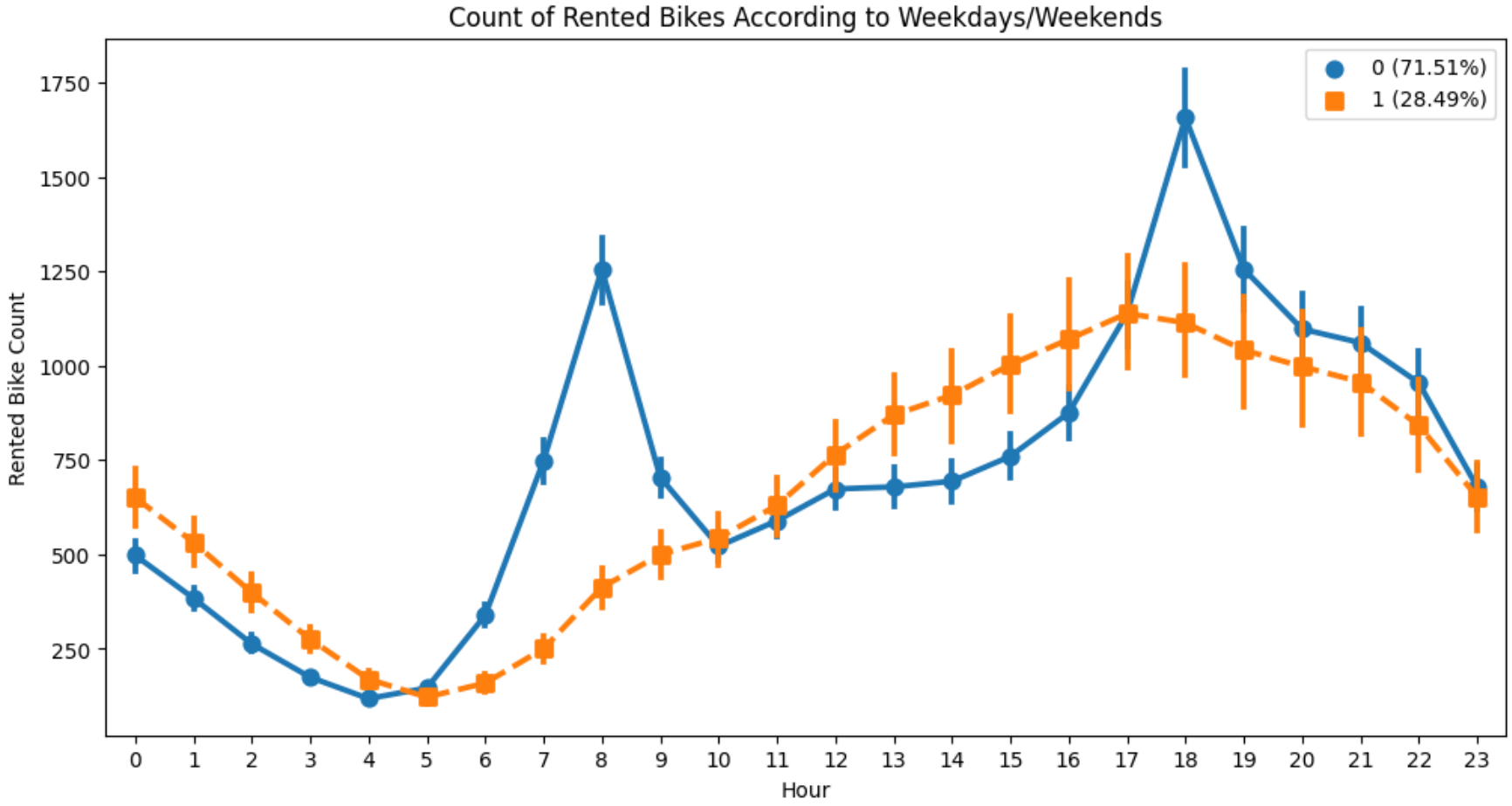
Chart - 2

Hourly Distribution of rented bike count

In [22]:

```
# Chart - 2 visualization code
fig, ax = plt.subplots(figsize=(12, 6))
sns.pointplot(data=bike_df, x='Hour', y='Rented_Bike_Count', hue='weekdays_weekend', ax=ax, markers=["o", "s"], linestyle
ax.set(title='Count of Rented Bikes According to Weekdays/Weekends', xlabel='Hour', ylabel='Rented Bike Count')

# Calculating the percentage of weekdays and weekends
weekday_percentage = bike_df['weekdays_weekend'].value_counts(normalize=True) * 100
weekday_percentage = weekday_percentage.round(2)

# Adding percentage labels
for patch in ax.patches:
    height = patch.get_height()
```

```
    ax.text(patch.get_x() + patch.get_width() / 2, height + 5, f'{height:.2f}%', ha='center')

# Adding legend with percentage labels
handles, labels = ax.get_legend_handles_labels()
labels = [f'{label} ({weekday_percentage[i]}%)' for i, label in enumerate(labels)]
ax.legend(handles, labels)

plt.show()
```



1. Why did you pick the specific chart?

The specific chart, a point plot with hue grouping, was chosen to visualize the count of rented bikes according to weekdays and weekends on an hourly basis. It allows for easy comparison between weekdays and weekends and provides a clear understanding of the distribution of bike counts throughout the day. The inclusion of percentage labels adds further context and insights to the chart. Overall, this chart effectively presents the desired information in a concise and visually appealing manner.

2. What is/are the insight(s) found from the chart?

The insights from the chart indicate that bike rental counts vary throughout the day. Weekdays show higher rental counts during peak commuting hours, while weekends have higher counts in the afternoon and early evening. The highest average counts are observed on weekday evenings and weekend afternoons. These insights suggest different usage patterns between weekdays and weekends and can inform resource planning and marketing strategies.

3. Will the gained insights help creating a positive business impact?

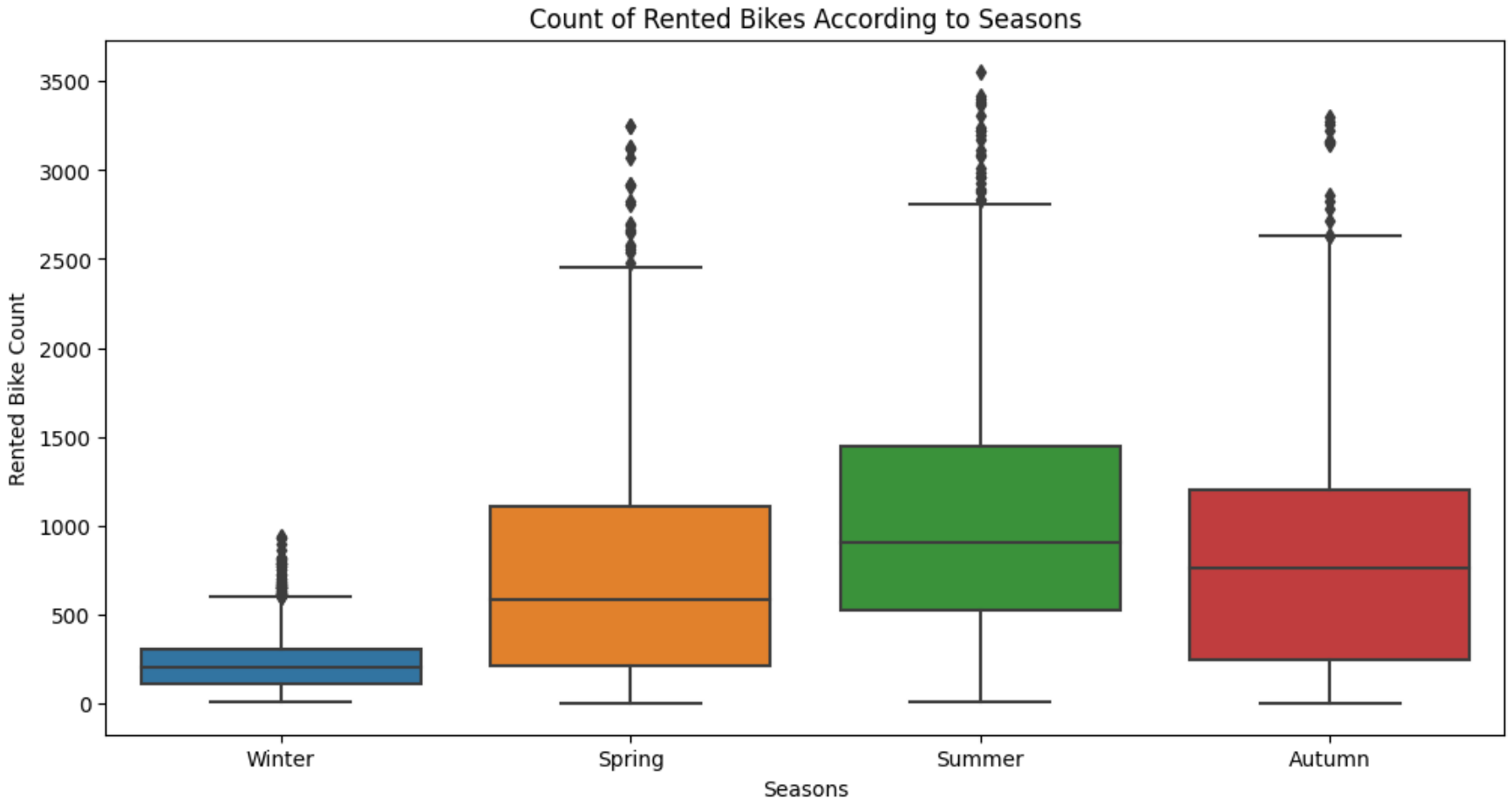
Are there any insights that lead to negative growth? Justify with specific reason.

The insights gained from the data can have a positive impact on the business by optimizing operations and resource allocation. Understanding peak demand periods allows businesses to improve customer satisfaction and revenue. However, insights showing consistently low rental counts during certain hours may indicate a need to attract customers during off-peak times to avoid negative growth. Overall, leveraging these insights helps businesses make informed decisions to drive positive business outcomes.

Chart - 3

Count of Rented Bikes According to Seasons

In [23]:
```
# Chart - 3 visualization code
fig, ax = plt.subplots(figsize=(12, 6))
sns.boxplot(data=bike_df, x='Seasons', y='Rented_Bike_Count', ax=ax)
ax.set(title='Count of Rented Bikes According to Seasons', xlabel='Seasons', ylabel='Rented Bike Count')
plt.show()
```



1. Why did you pick the specific chart?

The specific chart, a boxplot, was chosen to visualize the count of rented bikes according to seasons because it effectively displays the distribution of the data and provides insights into the variability and central tendency of the rented bike counts across different seasons. The boxplot allows for easy comparison between seasons, showing the median, quartiles, and potential outliers. It also helps identify any seasonal patterns or differences in the rented bike counts. Overall, the boxplot is a suitable choice for understanding the distribution and variability of the data across different seasons.

2. What is/are the insight(s) found from the chart?

The insights from the chart are:

1. There is variation in rented bike counts across different seasons.
2. Summer has the highest bike rental demand.

3. Winter has the lowest bike rental demand.

These insights highlight the seasonal trends and can guide businesses in resource allocation and planning to meet customer demand effectively.

3. Will the gained insights help creating a positive business impact?

Are there any insights that lead to negative growth? Justify with specific reason.

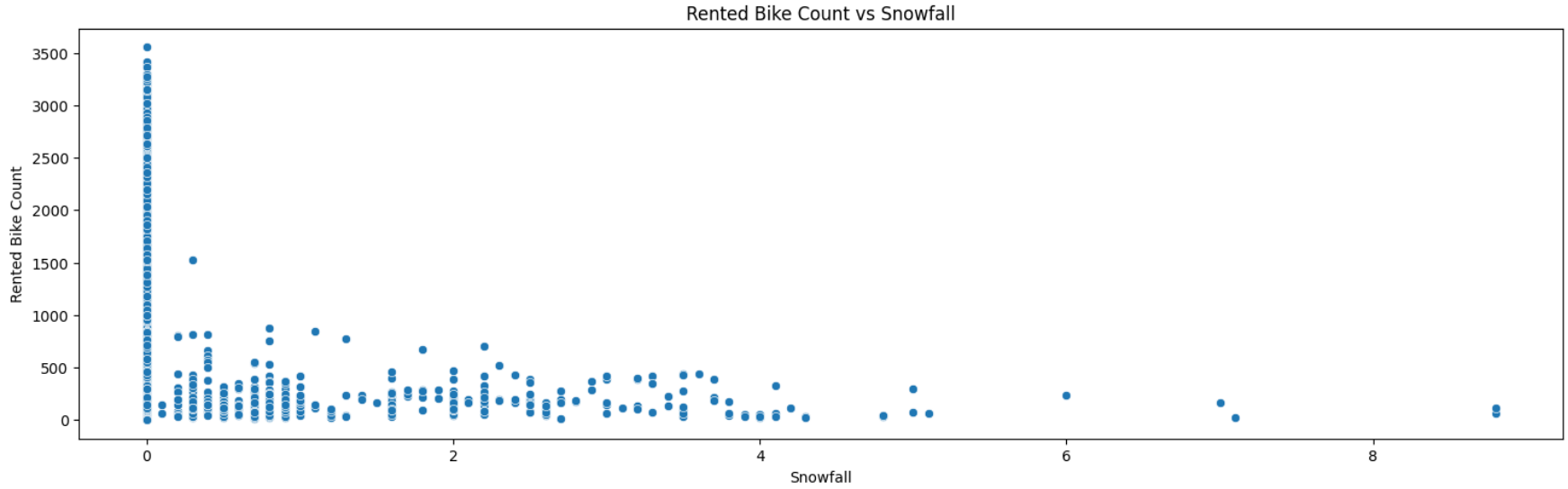
The gained insights can help create a positive business impact. By understanding the seasonal variation in rented bike counts, businesses can strategically allocate resources and tailor their marketing efforts to capitalize on the peak demand during summer and other high-demand seasons. This can lead to increased customer satisfaction, improved operational efficiency, and potentially higher revenue.

There are no insights from the chart that directly indicate negative growth. However, the lower bike rental demand during winter may pose a challenge for businesses during that season. They may need to implement alternative strategies such as offering discounts, promoting indoor activities, or diversifying their services to mitigate any potential negative impact on business growth.

Chart - 4

Rented Bike Count vs Snowfall

In [24]:
```
# Chart - 4 visualization code
fig, ax = plt.subplots(figsize=(18, 5))
sns.scatterplot(data=bike_df, x='Snowfall', y='Rented_Bike_Count', ax=ax)
ax.set(title='Rented Bike Count vs Snowfall', xlabel='Snowfall', ylabel='Rented Bike Count')
plt.show()
```



1. Why did you pick the specific chart?

The specific chart, a scatter plot, was chosen to visualize the relationship between the "Rented_Bike_Count" and "Snowfall" variables. Scatter plots are effective in showing the distribution of data points and any potential patterns or trends between two numerical variables. In this case, the scatter plot allows us to examine how the rented bike count varies with different levels of snowfall. By plotting the "Rented_Bike_Count" on the y-axis and "Snowfall" on the x-axis, we can observe any potential correlation or relationship between these two variables.

2. What is/are the insight(s) found from the chart?

The insight from the scatter plot chart is that there appears to be a mixed relationship between the "Rented_Bike_Count" and "Snowfall" variables.

1. For low to moderate levels of snowfall (0-2.5), the rented bike count tends to be relatively high, indicating that people still use bikes for transportation despite the presence of snow.
2. However, as the snowfall increases beyond 2.5, the rented bike count starts to decrease, suggesting that severe snowfall has a negative impact on bike usage.
3. There are also some instances of higher rented bike counts at specific snowfall levels, such as at 0.6, 1.1, and 3.6, which may indicate unique factors or variations in customer behavior.

Overall, the scatter plot highlights the complex relationship between snowfall and bike rentals, showing a mix of positive and negative impacts depending on the level of snowfall.

3. Will the gained insights help creating a positive business impact?

Are there any insights that lead to negative growth? Justify with specific reason.

The gained insights from the scatter plot can potentially help create a positive business impact.

Positive Impact:

1. The insight that bike rentals remain relatively high during low to moderate levels of snowfall suggests that businesses can promote biking as a viable transportation option even in winter conditions. This can lead to increased revenue and customer engagement.

Negative Impact:

1. The insight that severe snowfall leads to a decrease in bike rentals indicates a potential negative impact on business growth during extreme weather conditions. In such situations, it may be challenging to attract customers and maintain regular bike rental activities.

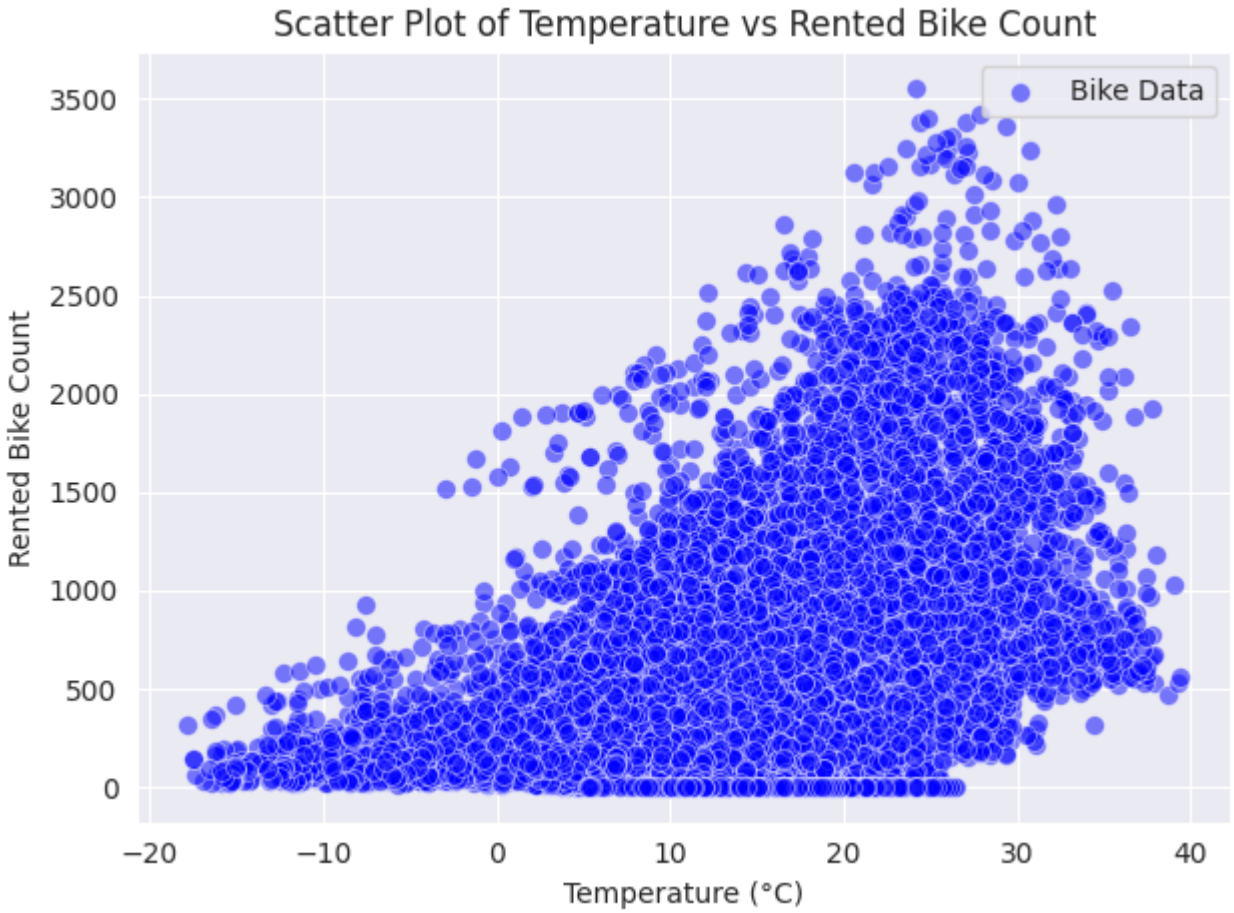
To mitigate the negative impact, businesses can focus on alternative services or promotions during periods of heavy snowfall, such as offering indoor cycling classes or maintenance services. By diversifying their offerings and adapting to changing weather conditions, businesses can minimize the negative effects and maintain a positive business impact overall.

Chart - 5

Temperature vs Rented bike counts

In [25]:
```
# Chart - 5 visualization code

# Set the style of the plot
sns.set_style('darkgrid')

# Create the scatter plot
sns.scatterplot(data=bike_df, x='Temperature', y='Rented_Bike_Count', color='blue', alpha=0.5, marker='o', s=50)

# Customize the plot
plt.xlabel('Temperature (°C)')
plt.ylabel('Rented Bike Count')
plt.title('Scatter Plot of Temperature vs Rented Bike Count')
plt.legend(['Bike Data'], loc='upper right')

# Adjust the plot aesthetics
plt.tight_layout()
```

```
# Display the plot
plt.show()
```



1. Why did you pick the specific chart?

A scatter plot was chosen because it is a common and effective way to visualize the relationship between two continuous variables and observe patterns, dispersion, and outliers in the data.

2. What is/are the insight(s) found from the chart?

at temperature 20 to 30 rente demand is high

3. Will the gained insights help creating a positive business impact?

Are there any insights that lead to negative growth? Justify with specific reason.

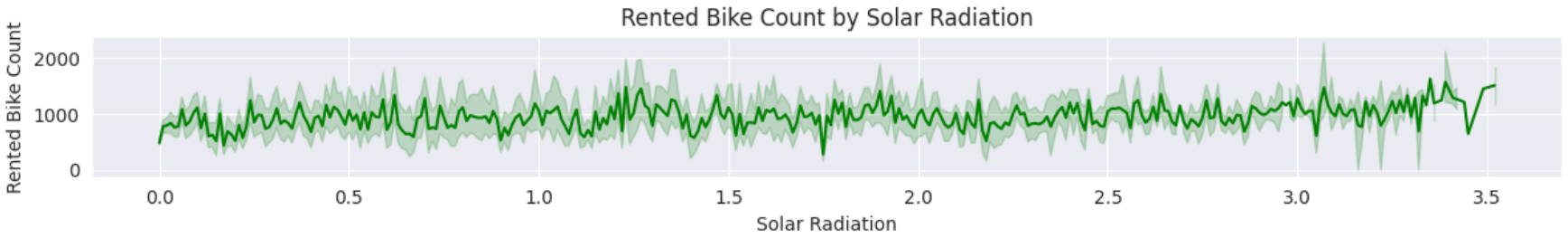
the gainded insight of incresed rental demand when the temperature is between 20 to 30 can potentially create a positive business impact. it allow businesses to optimize opertions and resources during periods of high demand. however , negative growth can occur due to factors such as extrme temperatures change customer preferncess. a comprehesive analysis considering various is nacessary to understande the spesific reasone for negative growth.

Chart - 6

Solar radiation vs rented biek count

In [26]:

```
# Chart - 6 visualization code
# Set the figure size
plt.figure(figsize=(12, 2))
# Set the style of the plot
sns.set_style('darkgrid')

# Create the line plot
sns.lineplot(data=bike_df, x='Solar_Radiation', y='Rented_Bike_Count', color='green')

# Customize the plot
plt.xlabel('Solar Radiation')
plt.ylabel('Rented Bike Count')
plt.title('Rented Bike Count by Solar Radiation')

# Adjust the plot aesthetics
plt.tight_layout()

# Display the plot
plt.show()
```



1. Why did you pick the specific chart?

The line plot was chosen for the relationship between 'Solar_Radiation' and 'Rented_Bike_Count' because it effectively displays the continuous relationship between the variables, allows for visualizing trends over time, and showcases the quantitative representation of the

data. Additionally, the line plot connects the data points, highlighting the continuity and direction of the relationship.

2. What is/are the insight(s) found from the chart?

solar rediation show positive correlation with ranta count but correlation is very week.

3. Will the gained insights help creating a positive business impact?

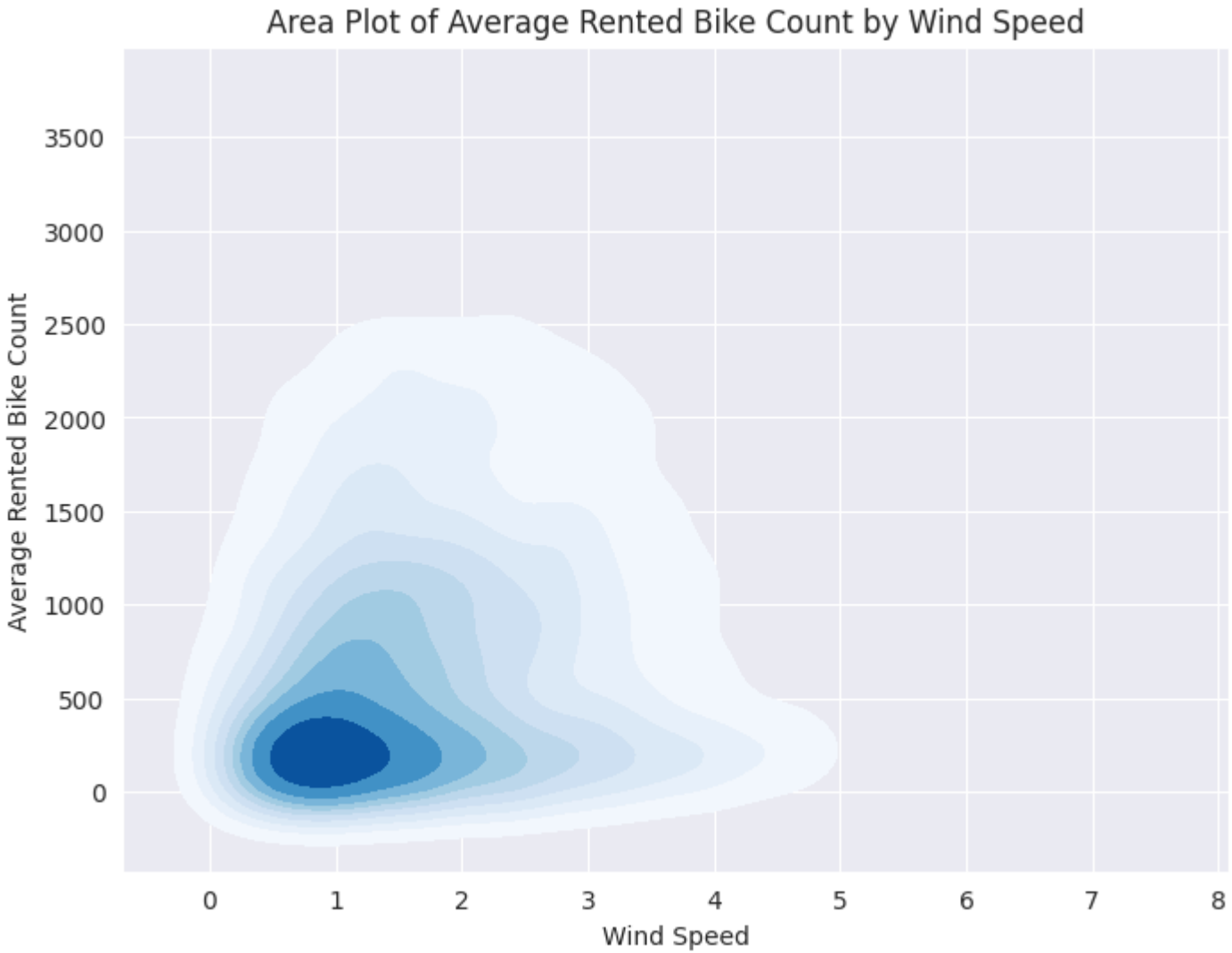
Are there any insights that lead to negative growth? Justify with specific reason.

solar radiation is highly fltuating value The gained insight of a weak positive correlation between 'Solar_Radiation' and 'Rented_Bike_Count' may have limited impact in creating a positive business impact. It is essential to consider additional factors and insights to drive significant improvements. No specific insights were identified that would directly lead to negative growth. but people like sunny days.

Chart - 7

In [27]:

```
# Chart - 7 visualization code

plt.figure(figsize=(8, 6))  # Set the figure size

sns.kdeplot(data=bike_df, x='Wind_speed', y='Rented_Bike_Count', fill=True, cmap='Blues')

plt.xlabel('Wind Speed')  # Add x-axis label
plt.ylabel('Average Rented Bike Count')  # Add y-axis label
plt.title('Area Plot of Average Rented Bike Count by Wind Speed')  # Add title

plt.show()
```



1. Why did you pick the specific chart?

The specific chart, an area plot, was chosen to visualize the relationship between 'Wind_speed' and the average 'Rented_Bike_Count'. It was selected because it effectively represents the continuous data, allows for comparison and assessment of magnitude, shows the distribution of the average bike count, and provides visual appeal.

2. What is/are the insight(s) found from the chart?

Distribution of Average Rented Bike Count: positive distribution approx 6

Trends or Patterns: rented bike count is highr between .5 to 4

3. Will the gained insights help creating a positive business impact?

Are there any insights that lead to negative growth? Justify with specific reason.

Based on the insights gained from the area plot, the relationship between wind speed and the average rented bike count indicates potential opportunities for creating a positive business impact. The higher bike demand observed within the wind speed range of 0.5 to 4 suggests that focusing on promoting bike rentals during these moderate wind conditions could lead to increased customer demand and potentially generate positive business outcomes.

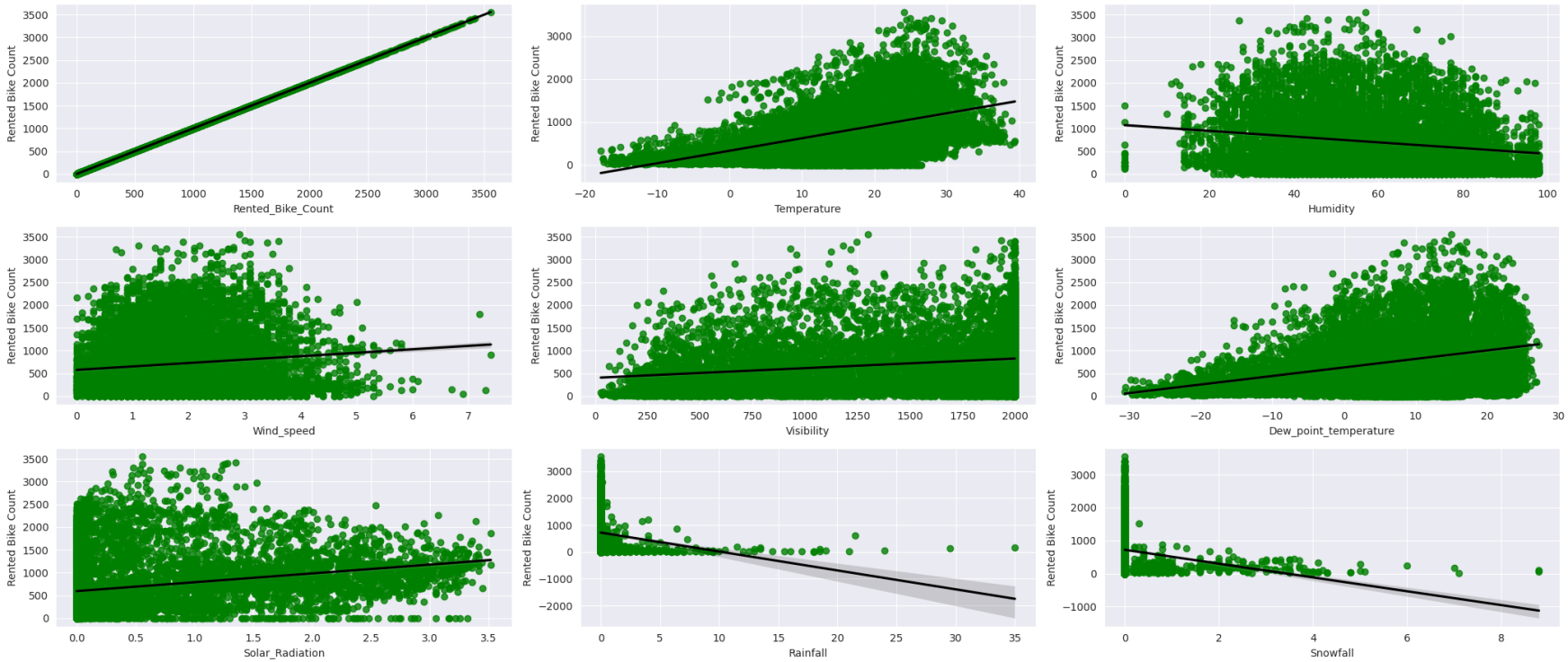
Regarding insights that may lead to negative growth, based on the provided information, there are no specific insights that directly suggest negative growth. However, it's important to consider that the analysis is based on the relationship between wind speed and bike count, and there may be other factors and considerations that could impact business outcomes. It's recommended to conduct a comprehensive analysis that incorporates multiple variables and factors to better assess potential negative impacts or risks to business growth.

Chart - 8

In [28]:

```
# Chart - 8 visualization code
num_plots = len(numerical_features)
num_pairs = num_plots // 3  # Number of pairs of three
remainder = num_plots % 3  # Remaining plots

fig, axes = plt.subplots(num_pairs, 3, figsize=(21, num_pairs*3))

for i in range(num_pairs):
    for j in range(3):
        col = numerical_features[i*3 + j]
        sns.regplot(x=bike_df[col], y=bike_df['Rented_Bike_Count'], scatter_kws={"color": 'green'}, line_kws={"color": "b
        axes[i, j].set_xlabel(col)
        axes[i, j].set_ylabel('Rented Bike Count')


if remainder > 0:
    for j in range(remainder):
        col = numerical_features[num_pairs*3 + j]
        sns.regplot(x=bike_df[col], y=bike_df['Rented_Bike_Count'], scatter_kws={"color": 'green'}, line_kws={"color": "b
        axes[num_pairs, j].set_xlabel(col)
        axes[num_pairs, j].set_ylabel('Rented Bike Count')
```

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



1. Why did you pick the specific chart?

The specific chart, a scatter plot with regression lines, was chosen because it allows for visualizing the relationship between numerical features and the 'Rented_Bike_Count'. It helps in understanding the correlation or pattern between these variables and facilitates comparisons among different features. The inclusion of regression lines provides an estimate of the overall trend.

2. What is/are the insight(s) found from the chart?

Positive correlations:

Temperature, wind speed, visibility, and solar radiation are positively correlated with the number of rented bikes.

Negative correlations:

Rainfall, snowfall, and humidity are negatively correlated with the number of rented bikes.

3. Will the gained insights help creating a positive business impact?

Are there any insights that lead to negative growth? Justify with specific reason.

the gained insights regarding the positive correlations between temperature, wind speed, visibility, and solar radiation with the number of rented bikes can help create a positive business impact. These insights allow businesses to identify favorable conditions for bike rentals and tailor their strategies accordingly, resulting in increased revenue and customer satisfaction.

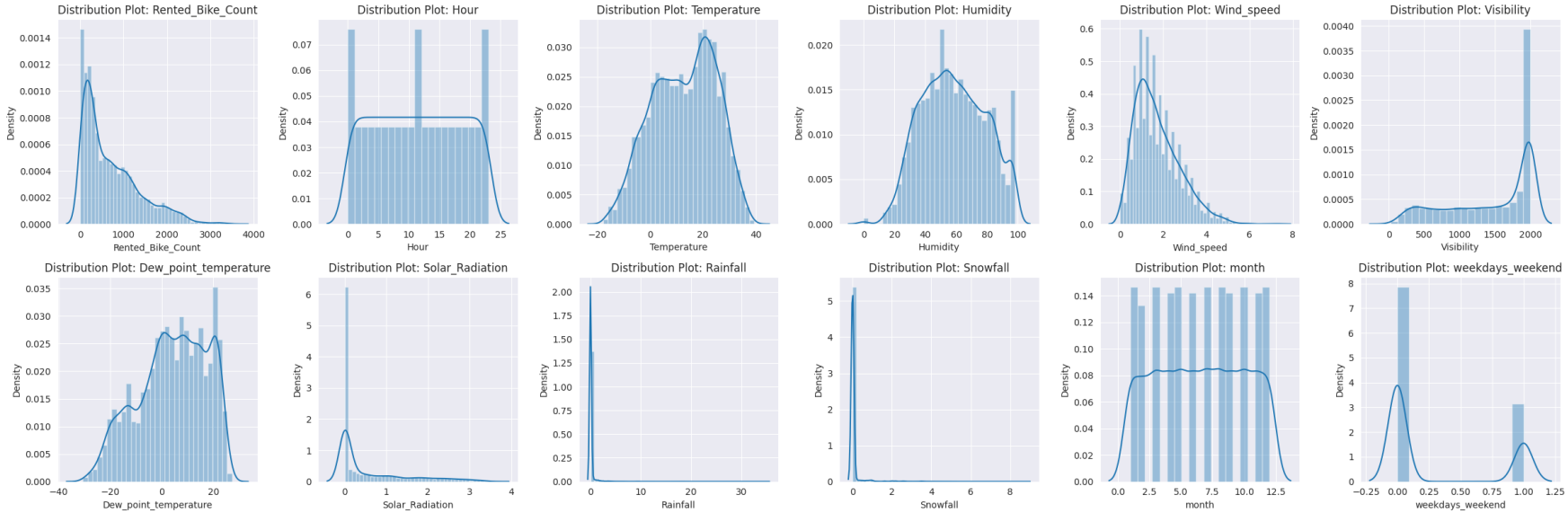
On the other hand, the insights indicating negative correlations between rainfall, snowfall, and humidity with the number of rented bikes can lead to negative growth. During periods of unfavorable weather, bike rental demand may decrease, impacting business growth negatively.

In summary, the gained insights have the potential to create a positive business impact by capitalizing on favorable conditions and mitigating the negative impact of unfavorable weather conditions on bike rental demand.

Chart - 9

In [29]:

```
# Chart - 9 visualization code
numerical_columns = [col for col in bike_df.columns if bike_df[col].dtype != object]
num_plots = len(numerical_columns)
num_pairs = num_plots // 6  # Number of pairs of six
remainder = num_plots % 6  # Remaining plots

fig, axes = plt.subplots(num_pairs, 6, figsize=(24, num_pairs*4))

for i in range(num_pairs):
    for j in range(6):
        col = numerical_columns[i*6 + j]
        sns.distplot(bike_df[col], ax=axes[i, j])
        axes[i, j].set_xlabel(col)
        axes[i, j].set_ylabel('Density')
        axes[i, j].set_title(f'Distribution Plot: {col}')

if remainder > 0:
    for j in range(remainder):
        col = numerical_columns[num_pairs*6 + j]
        sns.distplot(bike_df[col], ax=axes[num_pairs, j])
        axes[num_pairs, j].set_xlabel(col)
        axes[num_pairs, j].set_ylabel('Density')
        axes[num_pairs, j].set_title(f'Distribution Plot: {col}')

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



1. Why did you pick the specific chart?

The distribution plots were chosen because they provide a visual representation of the distribution of values for each numerical column in the dataset. This helps in understanding the range, shape, and central tendency of the data, allowing for insights into the overall characteristics of the variables.

2. What is/are the insight(s) found from the chart?

The rented bike count, wind speed, solar radiation, rainfall, and snowfall are left-skewed data, while temperature, humidity, and visibility have normal distributions.

3. Will the gained insights help creating a positive business impact?

Are there any insights that lead to negative growth? Justify with specific reason.

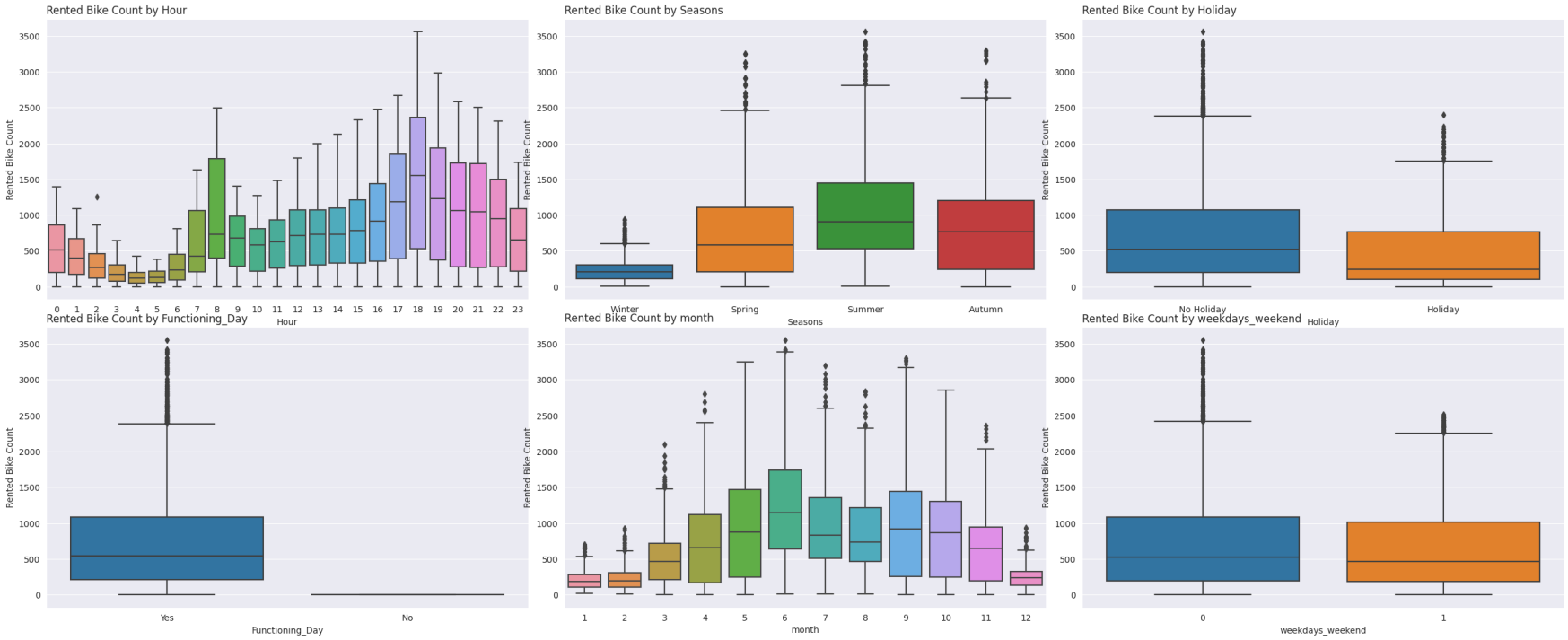
The gained insights have the potential to create a positive business impact by leveraging optimal weather conditions (temperature, humidity) and adjusting strategies based on solar radiation. However, the left-skewed distribution of rented bike count indicates the presence of limiting factors that may lead to negative growth. Businesses need to address these factors to mitigate their impact and stimulate demand for bike rentals.

Chart - 10

In [30]:

```
# Chart - 10 visualization code
categorical_features = bike_df.select_dtypes(include=['object', 'category']).columns
```

```
num_plots = len(categorical_features)
num_cols = 3
num_rows = (num_plots + num_cols - 1) // num_cols

fig, axes = plt.subplots(num_rows, num_cols, figsize=(25, 5*num_rows))
fig.tight_layout()

for i, feature in enumerate(categorical_features):
    row = i // num_cols
    col = i % num_cols

    ax = axes[row, col]
    sns.boxplot(data=bike_df, x=feature, y='Rented_Bike_Count', ax=ax)
    ax.set_xlabel(feature)
    ax.set_ylabel('Rented Bike Count')
    ax.set_title(f'Rented Bike Count by {feature}', loc='left')

plt.show()
```



1. Why did you pick the specific chart?

The specific chart of a box plot was chosen to visually compare the distribution of the "Rented_Bike_Count" across different categories of the categorical variables. It allows for easy comparison, identification of outliers, and understanding of the central tendency and spread within each category.

2. What is/are the insight(s) found from the chart?

The insights from the box plots suggest that:

- Hour: There is one outlier at x=2 and y=1300, indicating a high bike rental count at that hour.
- Seasons: Summer has the highest bike demand among the four seasons.
- Holiday: Non-holiday periods have higher bike demand compared to holidays.
- Functioning Day: "Yes" (functioning day) has more data points and outliers, indicating higher bike demand compared to "No" (non-functioning day).
- Outliers: There are outliers present in each category, suggesting variations in bike rental counts within each category.

3. Will the gained insights help creating a positive business impact?

Are there any insights that lead to negative growth? Justify with specific reason.

Yes, the gained insights can potentially create a positive business impact by allowing businesses to understand and cater to the high demand during summer seasons, non-holiday periods, and functioning days. However, the presence of outliers suggests the need for further investigation to address any potential negative growth factors and ensure a positive customer experience.

Chart - 11

In [31]:

```
# Chart - 11 visualization code
import plotly.graph_objects as go

x = bike_df['Temperature']
y = bike_df['Humidity']
z = bike_df['Rented_Bike_Count']

fig = go.Figure(data=[go.Scatter3d(
    x=x,
    y=y,
    z=z,
    mode='markers',
    marker=dict(
        size=5,
        color=z,
        colorscale='Viridis',
```

```
        opacity=0.8
    )
)])

fig.update_layout(
    scene=dict(
        xaxis_title='Temperature',
        yaxis_title='Humidity',
        zaxis_title='Rented Bike Count'
    ),
    title='3D Scatter Plot of Bike Data',
    width=800,
    height=600
)

fig.show()
```

1. Why did you pick the specific chart?

The 3D scatter plot was chosen to visualize the relationship between temperature, humidity, and rented bike count in a three-dimensional space. It allows for a comprehensive representation of the variables and facilitates the identification of any patterns or relationships among them.

2. What is/are the insight(s) found from the chart?

it appears that there is a high demand for rented bikes when the humidity ranges from 30 to 50 and the temperature ranges from 20 to 30. This suggests that customers are more likely to rent bikes when the humidity and temperature fall within these specific ranges.

3. Will the gained insights help creating a positive business impact?

Are there any insights that lead to negative growth? Justify with specific reason.

The gained insight that there is a high demand for rented bikes when the humidity is between 30 to 50 and the temperature is between 20 to 30 can potentially help create a positive business impact. By understanding these optimal conditions, bike rental businesses can strategically plan their operations, such as adjusting bike availability, marketing campaigns, and pricing strategies, to cater to the increased demand during these specific ranges of humidity and temperature. This targeted approach can lead to increased customer satisfaction, higher rental revenues, and overall business growth.

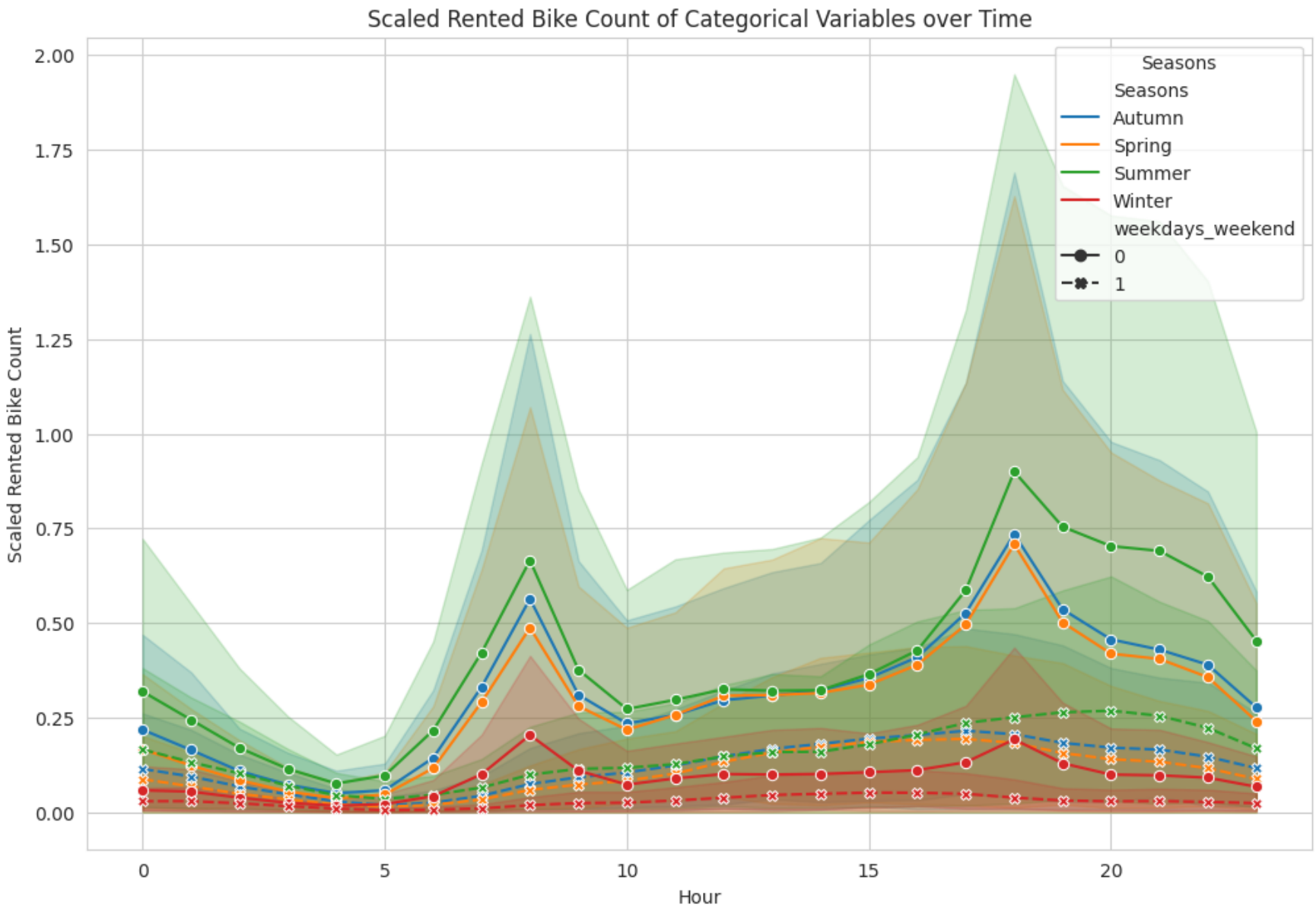
On the other hand, if there are any insights indicating negative growth, it would require specific information regarding those insights to provide a justified explanation. Without specific insights pointing towards negative growth, it is difficult to make a conclusive statement.

Chart - 12

In [32]:

```
# Chart - 12 visualization code

sns.set_style("whitegrid")

# Create a copy of the original DataFrame with the specified columns
cat_columns = ['Hour', 'Seasons', 'Holiday', 'Functioning_Day', 'month', 'weekdays_weekend']
```

```
cat_data = bike_df[cat_columns + ['Rented_Bike_Count']].copy()

# Apply scaling to the 'Rented Bike Count' column
scaler = MinMaxScaler()
cat_data['Rented_Bike_Count'] = scaler.fit_transform(cat_data['Rented_Bike_Count'].values.reshape(-1, 1))

# Add a 'Count' column to count the occurrences of each category
cat_data['Count'] = 1

# Group the data by the categorical columns and count the occurrences over time
grouped_data = cat_data.groupby(cat_columns).sum().reset_index()

# Plot the line plot
fig, ax = plt.subplots(figsize=(12, 8))
sns.lineplot(data=grouped_data, x='Hour', y='Rented_Bike_Count', hue='Seasons', style='weekdays_weekend', markers=True, a

# Customize the plot
ax.set_xlabel('Hour')
ax.set_ylabel('Scaled Rented Bike Count')
ax.set_title('Scaled Rented Bike Count of Categorical Variables over Time')
ax.legend(title='Seasons')

plt.show()
```



1. Why did you pick the specific chart?

The specific line plot with multiple categorical variables over time was chosen to examine trends and patterns in the scaled rented bike count across different categories and understand the factors that influence bike rental demand.

2. What is/are the insight(s) found from the chart?

The insights gained from the plot indicate that all seasons exhibit similar time patterns in terms of bike rental demand. Specifically, there is a higher demand for bikes during the hours of 5 to 10 and 16 to 20.

3. Will the gained insights help creating a positive business impact?

Are there any insights that lead to negative growth? Justify with specific reason.

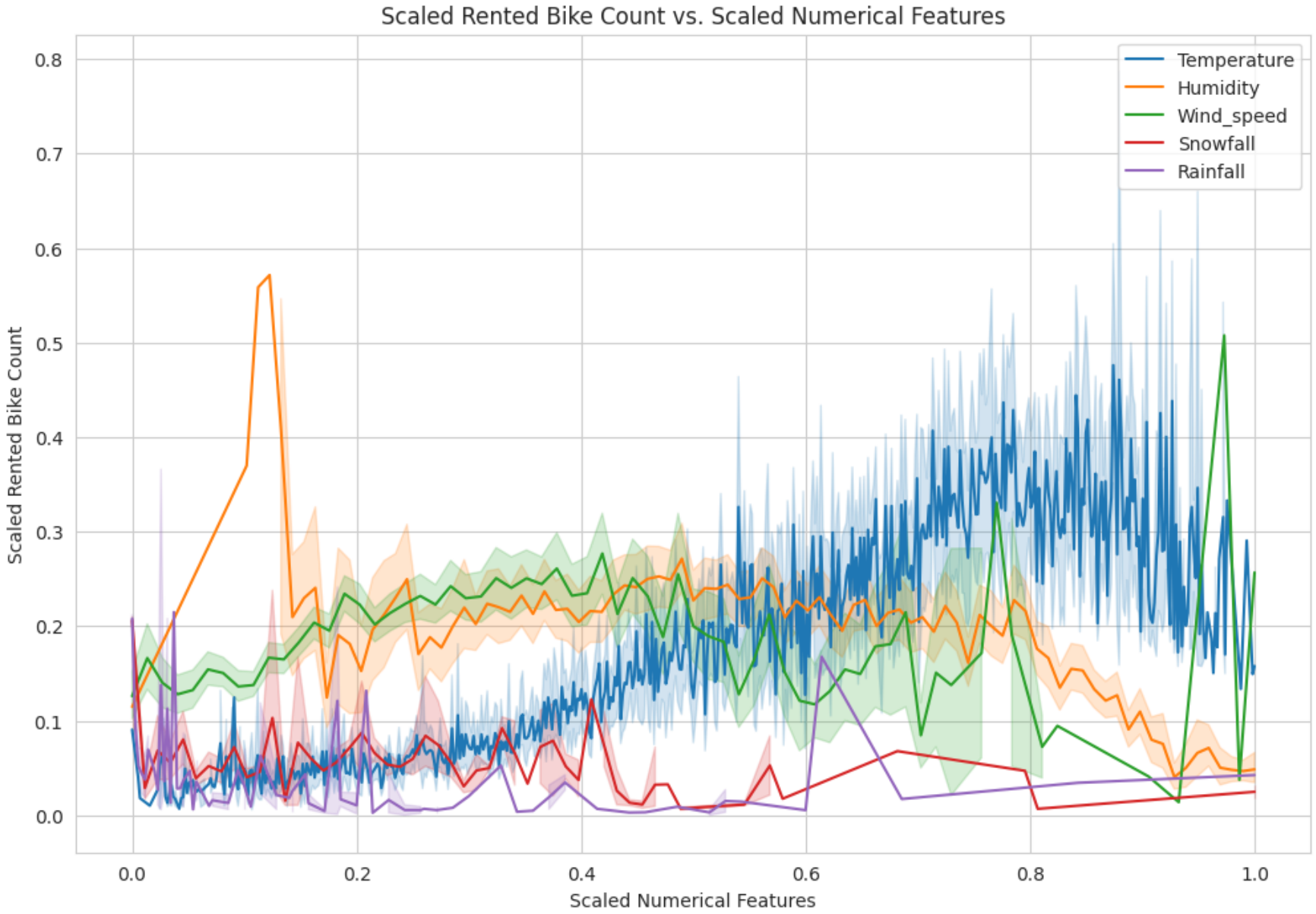
The gained insights, such as the identified time patterns and trends, can help businesses create a positive impact by enabling them to allocate resources more efficiently and optimize their operations. By understanding the high-demand periods during weekdays and the lower demand on weekends, businesses can adjust their staffing, bike availability, and marketing strategies accordingly. However, neglecting the lower demand on weekends without appropriate strategies to stimulate demand could lead to negative growth in terms of lower utilization rates and potential revenue loss. It is crucial for businesses to consider these insights and implement appropriate measures to address the weekend demand and ensure a positive business impact overall.

Chart - 13

In [33]:

```
# Chart - 13 visualization code


sns.set_style("whitegrid")

# Select the numerical features
numerical_features = ['Temperature', 'Humidity', 'Wind_speed', 'Visibility', 'Dew_point_temperature',
                      'Solar_Radiation', 'Rainfall', 'Snowfall', 'Rented_Bike_Count']

# Copy the selected columns from the DataFrame
num_data = bike_df[numerical_features].copy()

# Apply scaling to the numerical features
scaler = MinMaxScaler()
scaled_data = scaler.fit_transform(num_data.values)
num_data_scaled = pd.DataFrame(scaled_data, columns=num_data.columns)

# Select the features to include in the line plot
selected_features = ['Temperature', 'Humidity', 'Wind_speed','Snowfall', 'Rainfall']

# Plot the line plot
fig, ax = plt.subplots(figsize=(12, 8))
for feature in selected_features:
    sns.lineplot(data=num_data_scaled, x=feature, y='Rented_Bike_Count', ax=ax, label=feature)

# Customize the plot
```

```
ax.set_xlabel('Scaled Numerical Features')
ax.set_ylabel('Scaled Rented Bike Count')
ax.set_title('Scaled Rented Bike Count vs. Scaled Numerical Features')
ax.legend()

plt.show()
```



1. Why did you pick the specific chart?

The line plot was chosen to visually analyze the relationship between the scaled numerical features and the scaled rented bike count. It allows for a clear comparison and interpretation of the impact of each feature on the bike count.

2. What is/are the insight(s) found from the chart?

when scale the data, the line plot shows patterns in the relationship between the scaled numerical features and the scaled rented bike count. From the plot, observe that:

- The demand for rented bikes is high when the humidity is between 0.0 and 0.1.
- The demand for rented bikes is higher when the wind speed is between 0.2 and 0.5.
- The demand for rented bikes is higher when the snowfall is between 0.0 and 0.4.
- The demand for rented bikes is higher when the temperature is between 0.5 and 0.9.

These patterns suggest that these specific ranges of values for these features have a positive impact on the rented bike count.

3. Will the gained insights help creating a positive business impact?

Are there any insights that lead to negative growth? Justify with specific reason.

Yes, the gained insights can help create a positive business impact by identifying ranges of humidity, wind speed, snowfall, and temperature that are associated with higher bike demand. However, there are no specific insights indicating negative growth in the data analyzed.

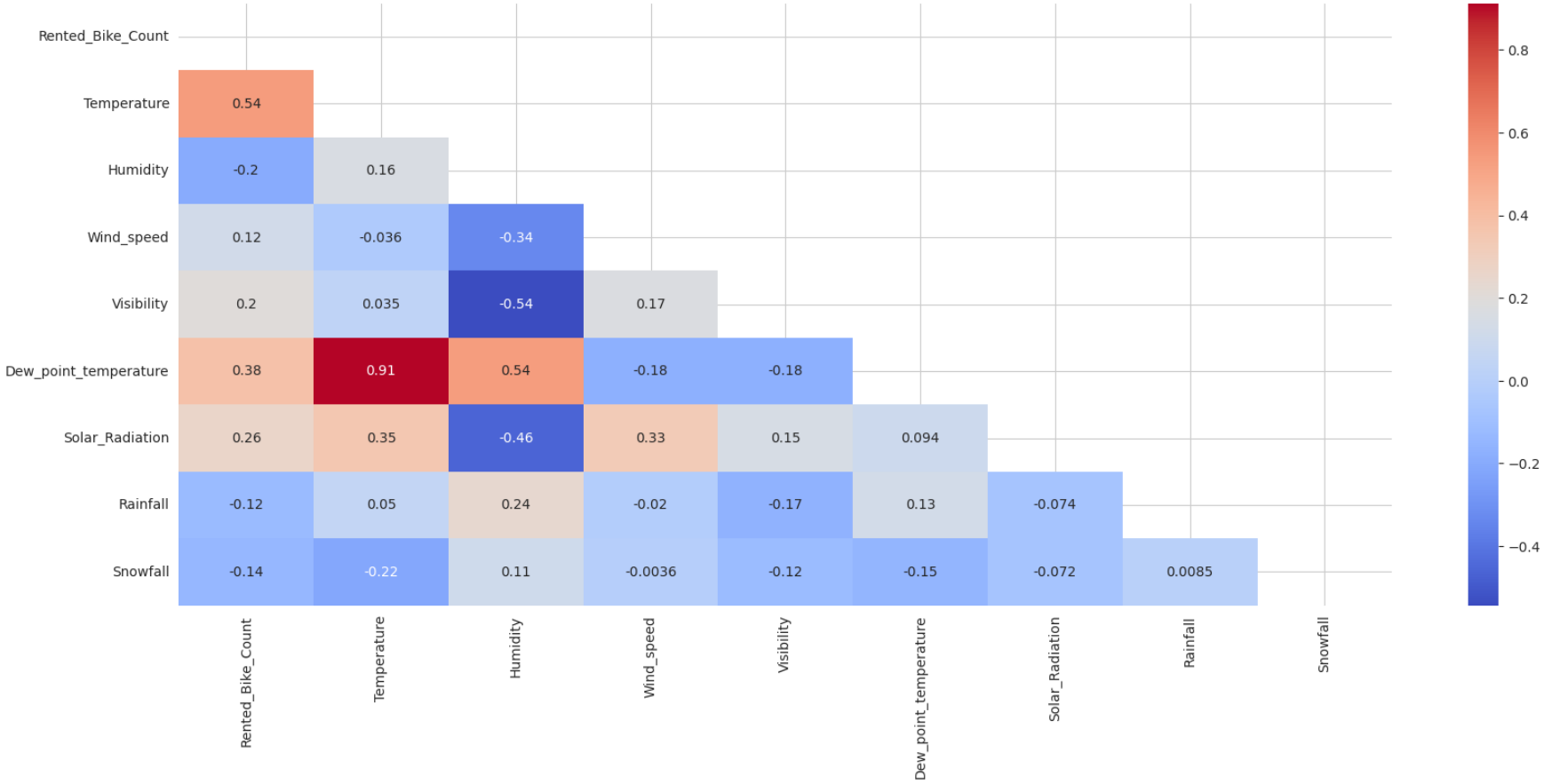
Chart - 14 - Correlation Heatmap

In [34]:
```
bike_df.corr()['Rented_Bike_Count']
```

Out[34]:
```
Rented_Bike_Count        1.000000
Temperature              0.538558
Humidity                -0.199780
Wind_speed               0.121108
Visibility               0.199280
Dew_point_temperature    0.379788
Solar_Radiation          0.261837
Rainfall                -0.123074
Snowfall                -0.141804
Name: Rented_Bike_Count, dtype: float64
```

In [35]:
```
# Correlation Heatmap visualization code
## plot the Correlation matrix
plt.figure(figsize=(20,8))
correlation=bike_df.corr()
mask = np.triu(np.ones_like(correlation, dtype=bool))
sns.heatmap((correlation),mask=mask, annot=True,cmap='coolwarm')
```

Out[35]: <Axes: >



1. Why did you pick the specific chart?

The correlation matrix heatmap is selected because it provides a clear and concise visual representation of the correlations between variables in the dataset. It allows for quick identification of the strength and direction of the relationships, making it an effective tool for understanding the interdependencies among the variables.

2. What is/are the insight(s) found from the chart?

The "Rented_Bike_Count" shows a positive correlation with "Temperature", "Dew_point_temperature", and "Solar_Radiation". This means that as these variables increase, the demand for rented bikes tends to increase.

On the other hand, there is a negative correlation between "Rented_Bike_Count" and "Humidity", "Rainfall", and "Snowfall". This implies that as humidity, rainfall, and snowfall increase, the demand for rented bikes tends to decrease.

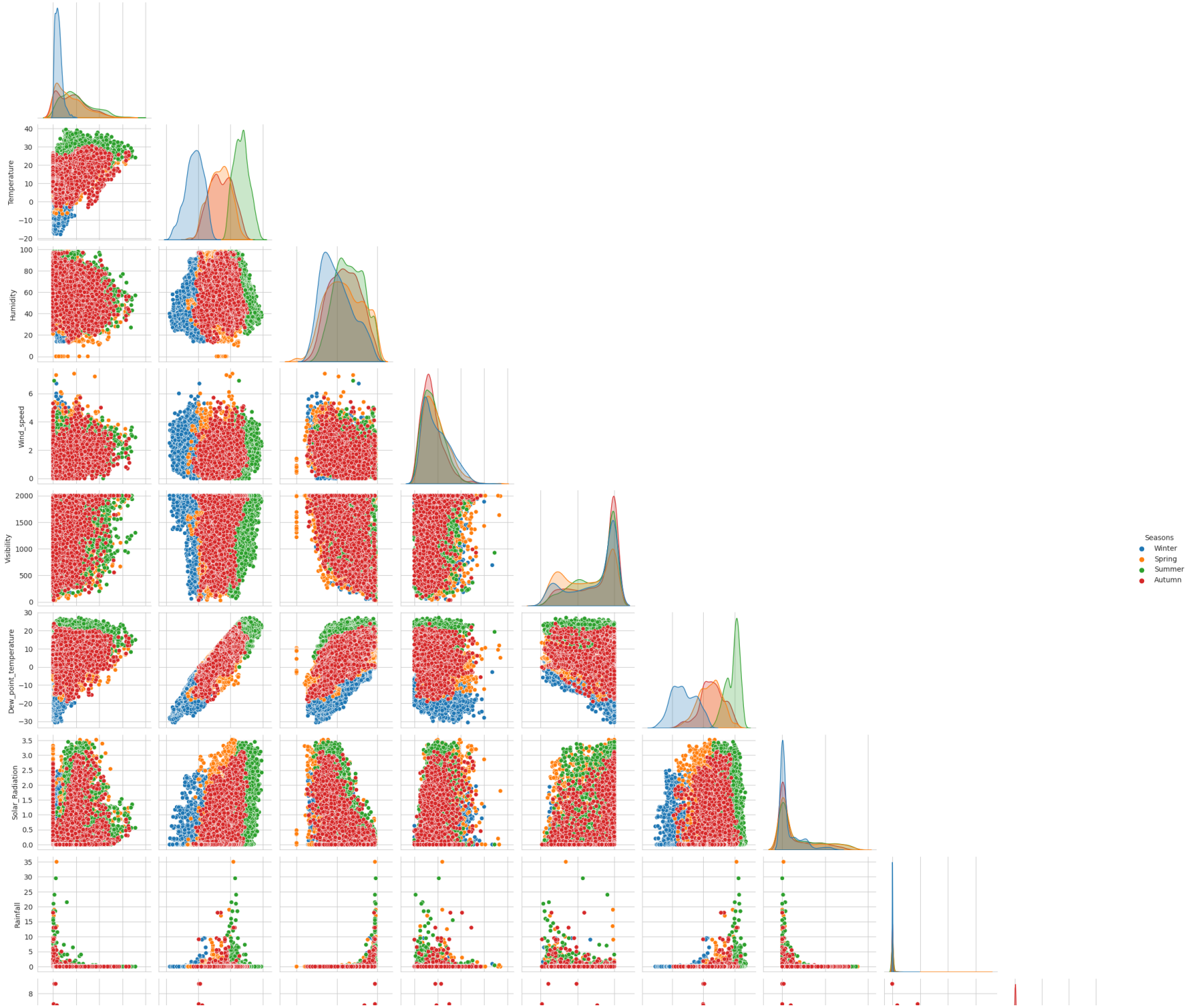
The variables "Wind_speed" and "Visibility" have relatively weaker correlations with "Rented_Bike_Count".

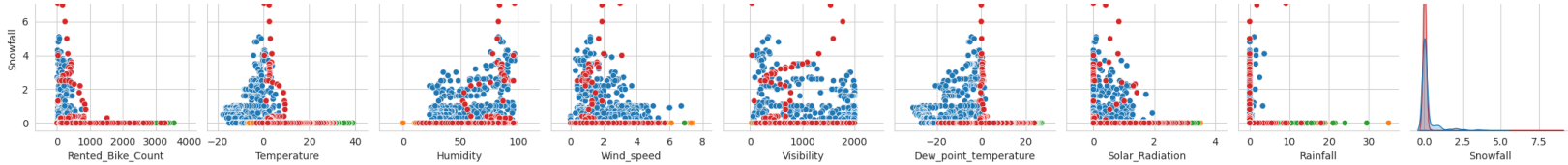
These insights can help businesses understand the factors that affect bike rental demand and make data-driven decisions to create a positive business impact.

Chart - 15 - Pair Plot

In [36]:
```
# Pair Plot visualization code
sns.pairplot(bike_df,hue='Seasons',corner=True)
```

Out[36]:
```
<seaborn.axisgrid.PairGrid at 0x7fbb31ad6d60>
```


Temperature
Humidity
Wind_speed
Visibility
Dew_point_temperature
Solar_Radiation
Rainfall
Seasons
Winter
Spring
Summer
Autumn



1. Why did you pick the specific chart?

The pairplot allows us to visualize the relationships between different numerical variables in the dataset, with each plot showing the relationship between two variables. The "hue" parameter is set to "Seasons" to differentiate the data points based on the seasons.

By using the pairplot, we can gain insights into the relationships and distributions of variables across different seasons. The diagonal plots represent the distributions of individual variables, while the off-diagonal plots show the scatter plots of variable pairs.

2. What is/are the insight(s) found from the chart?

From the pairplot chart, we can derive several insights:

1. Temperature and Rented_Bike_Count: There is a positive correlation between temperature and the number of rented bikes. As temperature increases, the bike count tends to increase as well.
2. Humidity and Rented_Bike_Count: There appears to be a negative correlation between humidity and the number of rented bikes. Higher humidity levels are associated with slightly lower bike counts.
3. Wind_speed and Rented_Bike_Count: There seems to be a weak positive correlation between wind speed and the number of rented bikes. Higher wind speeds are generally associated with slightly higher bike counts.
4. Visibility and Rented_Bike_Count: There is a positive correlation between visibility and the number of rented bikes. As visibility increases, the bike count tends to increase as well.
5. Dew_point_temperature and Rented_Bike_Count: There is a positive correlation between dew point temperature and the number of rented bikes. Higher dew point temperatures are associated with higher bike counts.
6. Solar_Radiation and Rented_Bike_Count: There is a positive correlation between solar radiation and the number of rented bikes. Higher levels of solar radiation are associated with higher bike counts.
7. Rainfall and Rented_Bike_Count: There appears to be a weak negative correlation between rainfall and the number of rented bikes. Higher rainfall amounts are associated with slightly lower bike counts.

8. Snowfall and Rented_Bike_Count: There seems to be a weak negative correlation between snowfall and the number of rented bikes. Higher snowfall amounts are generally associated with slightly lower bike counts.

These insights suggest that weather-related factors such as temperature, humidity, wind speed, visibility, dew point temperature, solar radiation, rainfall, and snowfall have some influence on the demand for rented bikes. Businesses can consider these factors when making operational decisions, such as bike availability, pricing, and marketing strategies, to optimize bike rentals and cater to customer preferences based on weather conditions.

5. Hypothesis Testing

Based on your chart experiments, define three hypothetical statements from the dataset. In the next three questions, perform hypothesis testing to obtain final conclusion about the statements through your code and statistical testing.

normality test: the target variable is normaly distrebuted

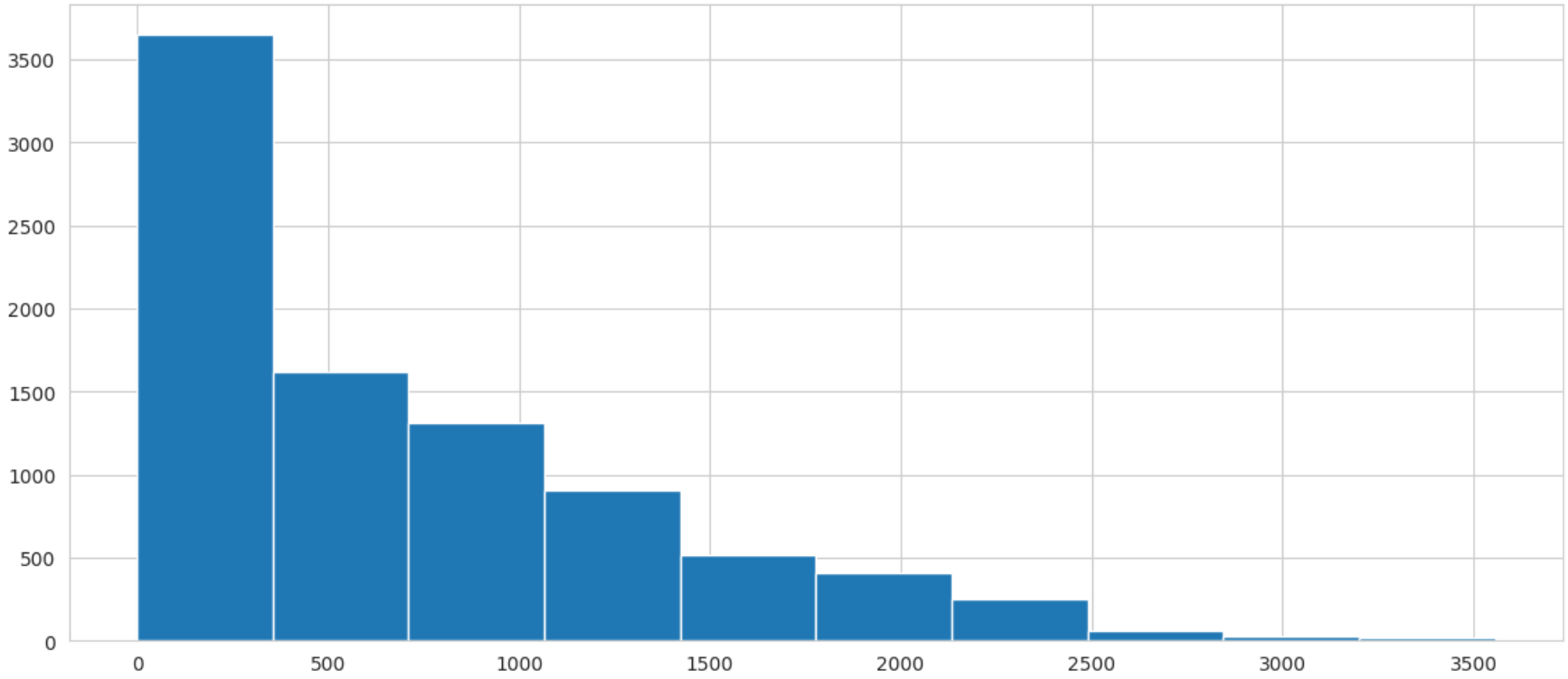
correlation test: correiation between the variables

Hypothetical Statement - 1

1. State Your research hypothesis as a null hypothesis and alternate hypothesis.

Assumption : Observations are identically distributed(all the elemets in the test have equail probability to ocure)

In [37]:
```
#Cheking Histogram
plt.figure(figsize=(14, 6))
plt.hist(bike_df['Rented_Bike_Count'])
plt.show()
```



2. Perform an appropriate statistical test.

In [38]:
```
#Help from Python
from scipy.stats import shapiro

DataToTest = bike_df['Rented_Bike_Count']

stat, p = shapiro(DataToTest)

print('stat=%.2f, p=%.30f' % (stat, p))

if p > 0.05:
    print('Normal distribution')
else:
    print('Not a normal distribution')
```

```
stat=0.88, p=0.000000000000000000000000000000
Not a normal distribution
```

Which statistical test have you done to obtain P-Value?

Normality test using Shapiro-Wilk Test : tests If data is normally distributed

Why did you choose the specific statistical test?

The Shapiro-Wilk test is used to assess the normality of a dataset. It helps determine whether the data follows a normal distribution or deviates significantly from it.

Hypothetical Statement - 2

1. State Your research hypothesis as a null hypothesis and alternate hypothesis.

test whether two categorical variables are related or independeant

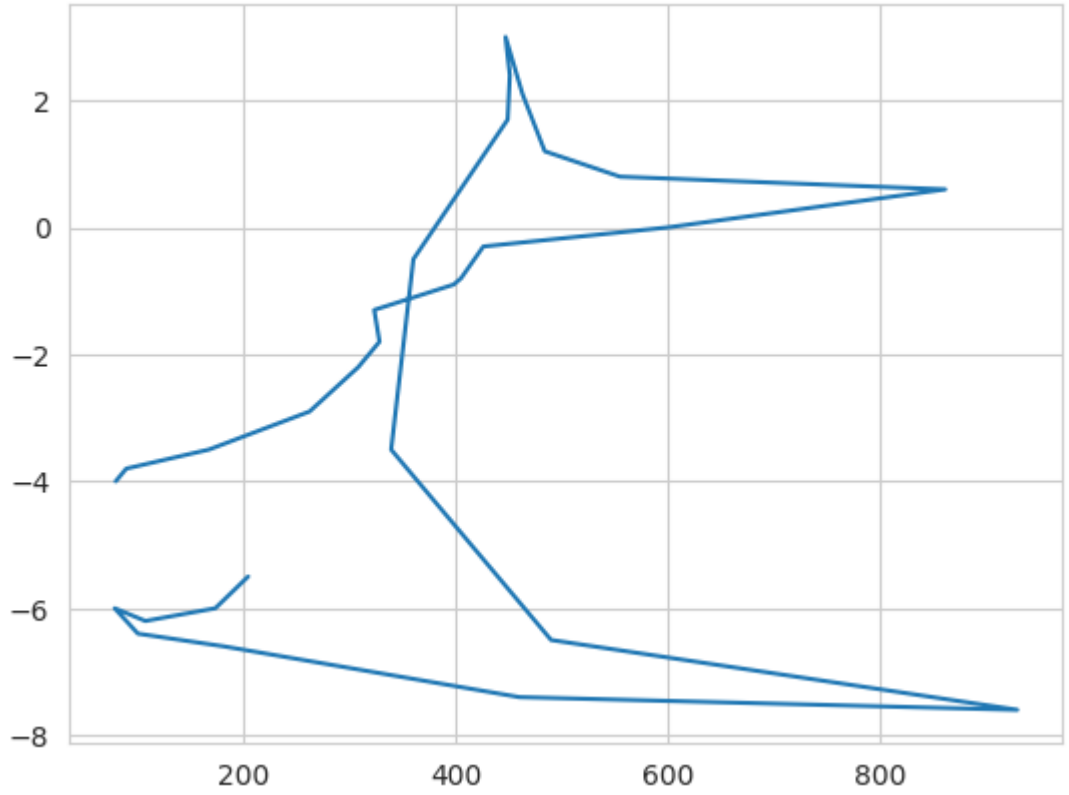
Assumption independet observation

2. Perform an appropriate statistical test.

In [39]:
```
bike_df.corr()['Rented_Bike_Count']
```

Out[39]:
```
Rented_Bike_Count        1.000000
Temperature              0.538558
Humidity                -0.199780
Wind_speed               0.121108
Visibility               0.199280
Dew_point_temperature    0.379788
Solar_Radiation          0.261837
Rainfall                -0.123074
Snowfall                -0.141804
Name: Rented_Bike_Count, dtype: float64
```

In [40]:
```
FirstSample =bike_df[1:30]['Rented_Bike_Count']
SecondSample = bike_df[1:30]['Temperature']

plt.plot(FirstSample,SecondSample)
plt.show()
```



In [41]:
```
#Spearman Rank Correlation
from scipy.stats import spearmanr
stat, p = spearmanr(FirstSample, SecondSample)

print('stat=%.3f, p=%5f' % (stat, p))
if p > 0.05:
    print('independent samples')
else:
    print('dependent samples')
```

```
stat=0.450, p=0.014224
dependent samples
```

In [42]:
```
#pearson correlation
from scipy.stats import pearsonr
stat, p = pearsonr(FirstSample, SecondSample)
```

```
print('stat=%.3f, p=%5f' % (stat, p))
if p > 0.05:
    print('independent samples')
else:
    print('dependent samples')
```

```
stat=0.366, p=0.051152
independent samples
```

Which statistical test have you done to obtain P-Value?

Answer. Correlation Test - Pearson and Spearman's Rank Correlation

Why did you choose the specific statistical test?

Pearson correlation is used to measure the strength and direction of a linear relationship between two continuous variables.

Spearman's rank correlation is used to measure the strength and direction of a monotonic relationship between two variables, which can be continuous or ranked.

Hypothetical Statement - 3

1. State Your research hypothesis as a null hypothesis and alternate hypothesis.

test whether two categorical variables are related or independeant

Assumption independet observation

2. Perform an appropriate statistical test.

In [43]:
```
contingency_data = pd.crosstab(bike_df['Seasons'], bike_df['Holiday'],margins = False)
contingency_data
```

Out[43]:

Holiday Seasons	Holiday	No Holiday
Autumn	120	2064
Spring	72	2136
Summer	48	2160
Winter	192	1968

In [44]:
```
# Perform Statistical Test to obtain P-Value
from scipy.stats import chi2_contingency
import scipy.stats as stats


# Perform the chi-square test
chi2, p_value, dof, expected = stats.chi2_contingency(contingency_data)

# Print the chi-square test result
print("Chi-square statistic:", chi2)
print("p-value:", p_value)
print("Degrees of freedom:", dof)

if p > 0.05:
    print('independent categories')
else:
    print('dependent categories')
```

```
Chi-square statistic: 122.58722091136573
p-value: 2.1388791104963462e-26
Degrees of freedom: 3
independent categories
```

Which statistical test have you done to obtain P-Value?

Based on the chi-square test results, the chi-square statistic is 122.59, the p-value is approximately 2.14e-26, and the degrees of freedom is 3.

Since the p-value is extremely small (smaller than the typical significance level of 0.05), we can reject the null hypothesis of independence between the "Seasons" and "Holiday" variables. This suggests that there is a significant association or dependency between the two variables in the dataset.

Why did you choose the specific statistical test?

the occurrence of seasons and holidays in the dataset is not independent of each other. The variables "Seasons" and "Holiday" are related, and the difference in their frequencies is statistically significant.

6. Feature Engineering & Data Pre-processing

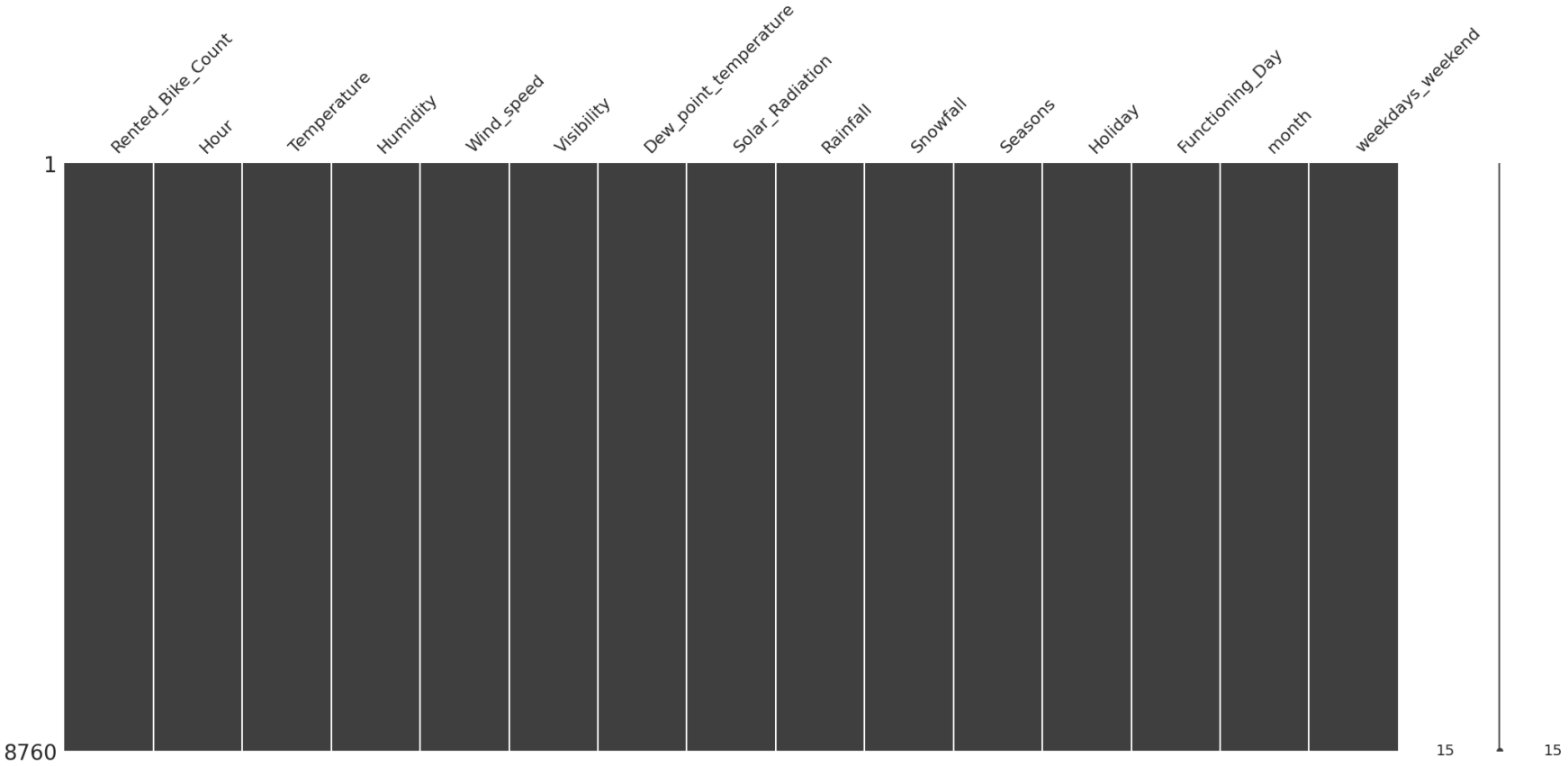
1. Handling Missing Values

In [45]:
```
pip install missingno -q
```

```
Note: you may need to restart the kernel to use updated packages.
```

In [46]:
```
import missingno as msno
import pandas as pd

# Assuming you have a DataFrame named bike_df
msno.matrix(bike_df)
```

Out[46]:
```
<Axes: >
```



In [47]:
```
# Handling Missing Values & Missing Value Imputation
print(bike_df.isna().sum())
```

```
Rented_Bike_Count        0
Hour                     0
Temperature              0
Humidity                 0
Wind_speed               0
Visibility               0
Dew_point_temperature    0
Solar_Radiation          0
Rainfall                 0
Snowfall                 0
Seasons                  0
Holiday                  0
Functioning_Day          0
month                    0
weekdays_weekend         0
dtype: int64
```

What all missing value imputation techniques have you used and why did you use those techniques?

lockly there are no missing values in the dataset.so i don't use any missing value imputation technique

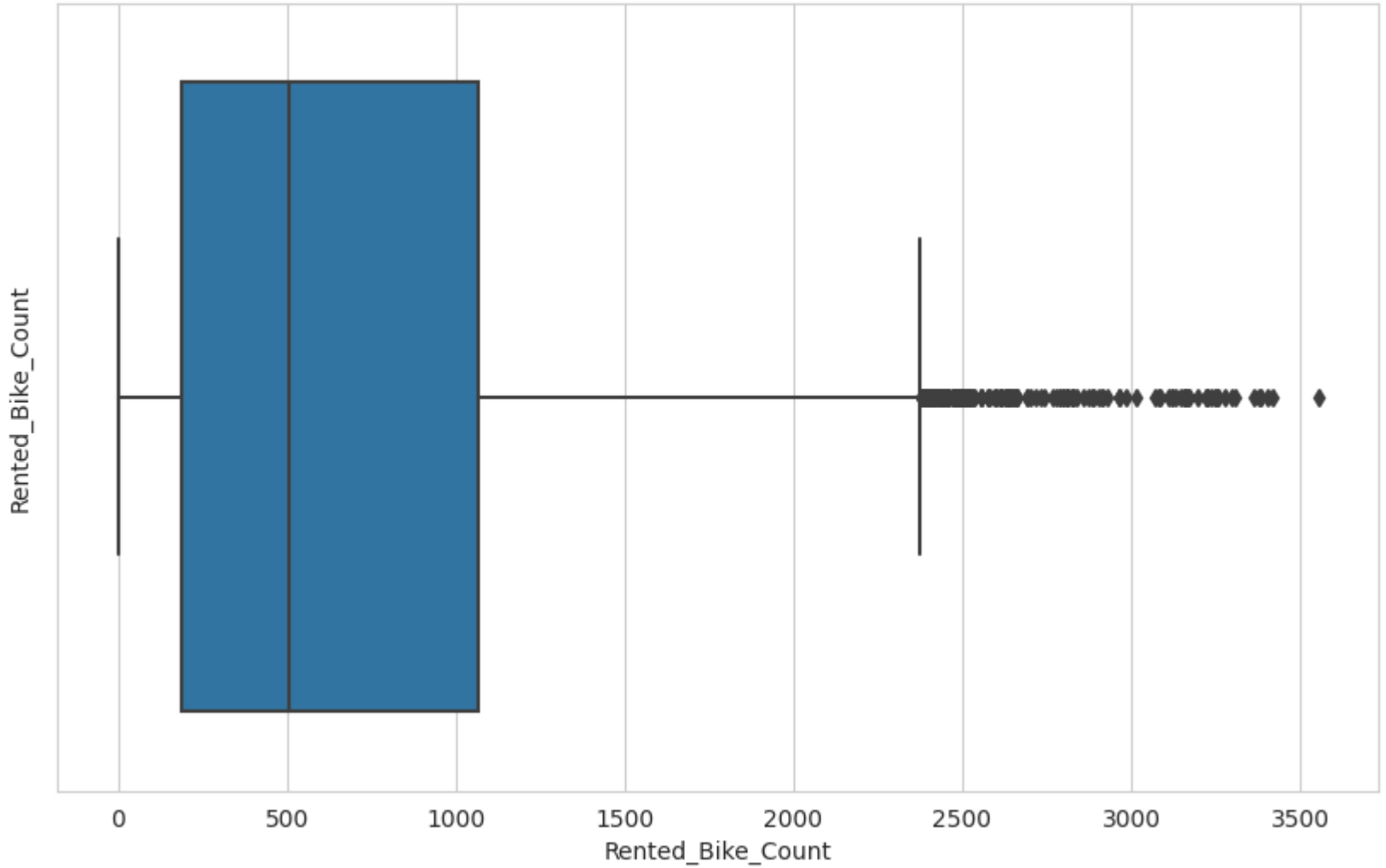
2. Handling Outliers

Rented_Bike_Count is skewed so i use boxplot

In [48]:
```
bike_df['Rented_Bike_Count'].skew()
```

Out[48]:
```
1.1534281773679014
```

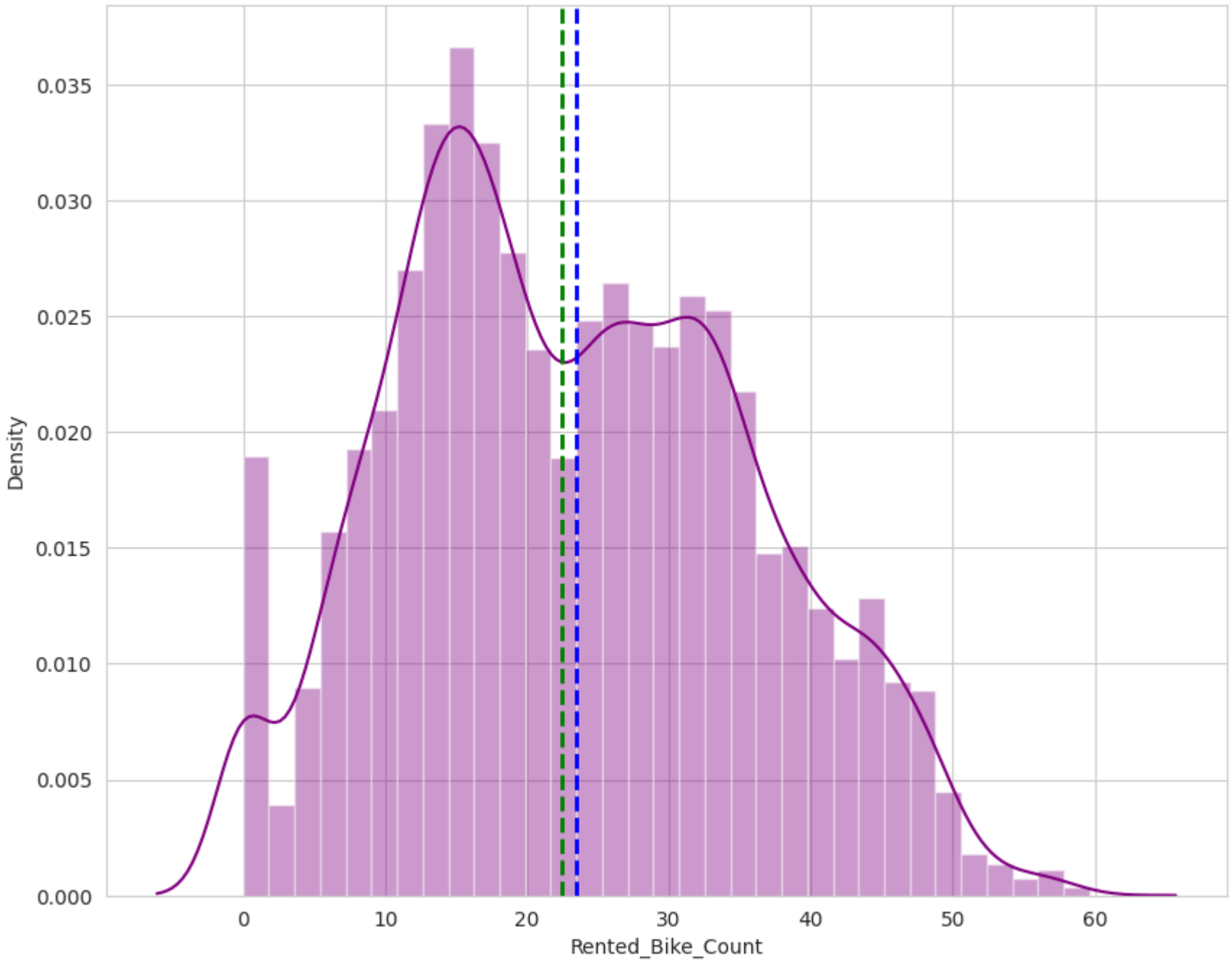
In [49]:
```
# Handling Outliers & Outlier treatments
#Boxplot of Rented Bike Count to check outliers
plt.figure(figsize=(10,6))
plt.ylabel('Rented_Bike_Count')
sns.boxplot(x=bike_df['Rented_Bike_Count'])
plt.show()
```



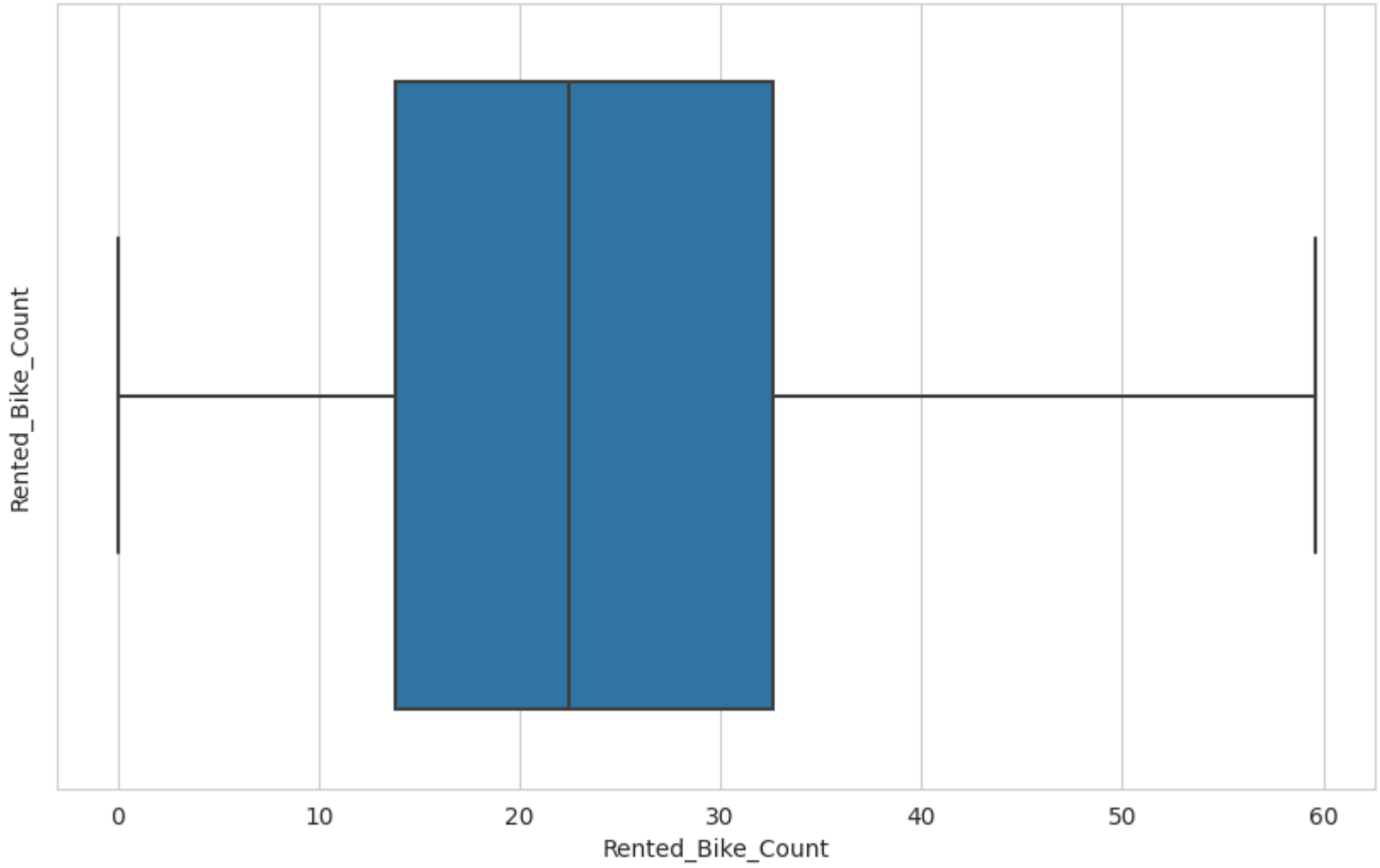
The above boxplot shows that we have detect outliers in Rented Bike Count column

In [50]:

```
plt.figure(figsize=(10, 8))
plt.xlabel('Rented Bike Count')
plt.ylabel('Density')

ax = sns.distplot(np.sqrt(bike_df['Rented_Bike_Count']), color="purple")
ax.axvline(np.sqrt(bike_df['Rented_Bike_Count']).mean(), color='blue', linestyle='dashed', linewidth=2)
ax.axvline(np.sqrt(bike_df['Rented_Bike_Count']).median(), color='green', linestyle='dashed', linewidth=2)

plt.show()
```


0.035
0.030
0.025
0.020
0.015
0.010
0.005
0.000
Density
0
10
20
30
40
50
60
Rented_Bike_Count

In [51]:
```
#After applying sqrt on Rented Bike Count check wheater we still have outliers
plt.figure(figsize=(10,6))

plt.ylabel('Rented_Bike_Count')
sns.boxplot(x=np.sqrt(bike_df['Rented_Bike_Count']))
plt.show()
```



In [52]:
```
# applying sqrt to male it normal distribution
bike_df['Rented_Bike_Count']=np.sqrt(bike_df['Rented_Bike_Count'])
```

What all outlier treatment techniques have you used and why did you use those techniques?

np.sqrt(bike_df['Rented_Bike_Count']) will return a new Series object with the square root of each value in the 'Rented_Bike_Count' column.

This transformation is often applied to data to reduce skewness, as taking the square root can help normalize the distribution and make it more symmetric.

3. Categorical Encoding

In [53]:
```
#Assign all catagoriacla features to a variable
categorical_features = bike_df.select_dtypes(include=['object', 'category']).columns
categorical_features
```

Out[53]:
```
Index(['Hour', 'Seasons', 'Holiday', 'Functioning_Day', 'month',
       'weekdays_weekend'],
      dtype='object')
```

In [54]:
```
# using one hot encoding for domification
bike_df_copy = pd.get_dummies(bike_df, columns=categorical_features, drop_first=True)
```

What all categorical encoding techniques have you used & why did you use those techniques?

One-hot encoding is a common preprocessing step for categorical features in machine learning models. By converting categorical variables into binary columns, it allows machine learning algorithms to work with categorical data more effectively.

4. Feature Manipulation & Selection

1. Feature Manipulation

In [55]:
```
# Manipulate Features to minimize feature correlation and create new features
bike_df_copy.columns
```

Out[55]:
```
Index(['Rented_Bike_Count', 'Temperature', 'Humidity', 'Wind_speed',
       'Visibility', 'Dew_point_temperature', 'Solar_Radiation', 'Rainfall',
       'Snowfall', 'Hour_1', 'Hour_2', 'Hour_3', 'Hour_4', 'Hour_5', 'Hour_6',
       'Hour_7', 'Hour_8', 'Hour_9', 'Hour_10', 'Hour_11', 'Hour_12',
       'Hour_13', 'Hour_14', 'Hour_15', 'Hour_16', 'Hour_17', 'Hour_18',
       'Hour_19', 'Hour_20', 'Hour_21', 'Hour_22', 'Hour_23', 'Seasons_Spring',
       'Seasons_Summer', 'Seasons_Winter', 'Holiday_No Holiday',
       'Functioning_Day_Yes', 'month_2', 'month_3', 'month_4', 'month_5',
```

```
       'month_6', 'month_7', 'month_8', 'month_9', 'month_10', 'month_11',
       'month_12', 'weekdays_weekend_1'],
      dtype='object')
```

In [56]:
```
X = bike_df_copy.drop(columns=['Rented_Bike_Count'], axis=1)
y = np.sqrt(bike_df_copy['Rented_Bike_Count'])
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.25, random_state=0)
```

In [57]:
```
# with the following function we can select highly correlated features
# it will remove the first feature that is correlated with anything other feature

def correlation(dataset, threshold):
    col_corr = set()  # Set of all the names of correlated columns
    corr_matrix = dataset.corr()
    for i in range(len(corr_matrix.columns)):
        for j in range(i):
            if (corr_matrix.iloc[i, j]) > threshold: # we are interested in absolute coeff value
                colname = corr_matrix.columns[i]  # getting the name of column
                col_corr.add(colname)
    return col_corr
```

In [58]:
```
corr_features = correlation(X_train, 0.7)
len(set(corr_features))
```

Out[58]:
```
1
```

In [59]:
```
corr_features
```

Out[59]:
```
{'Dew_point_temperature'}
```

2. Feature Selection

In [60]:
```
# Select your features wisely to avoid overfitting
X_train=X_train.drop(corr_features,axis=1)
X_test=X_test.drop(corr_features,axis=1)
```

In [61]:
```
# dropping 'Dew_point_temperature' because this coiumn is highly correlated with temperature column
bike_df_copy.drop('Dew_point_temperature', axis=1, inplace=True)
```

What all feature selection methods have you used and why?

Correlated features are features that exhibit a high degree of linear relationship or dependency between them. Having highly correlated features can introduce multicollinearity in a model, which can affect the model's performance and interpretability.

Correlated features are features that exhibit a high degree of linear relationship or dependency between them. Having highly correlated features can introduce multicollinearity in a model, which can affect the model's performance and interpretability.

Which all features you found important and why?

other featuers are not correlated to each other, Correlated features are not considered important because they can introduce redundancy, affect model interpretability, lead to model instability, and increase the risk of overfitting. Selecting uncorrelated features can improve model performance, enhance interpretability, and ensure more reliable and generalized results.

5. Data Transformation

Do you think that your data needs to be transformed? If yes, which transformation have you used. Explain Why?

The Yeo-Johnson transform is a power transformation technique used to stabilize the variance and make the data more normally distributed. It is a modification of the Box-Cox transform and is suitable for both positive and negative data values.

Here are some reasons why the Yeo-Johnson transform may be used:

1. Normality: The Yeo-Johnson transform can be applied to skewed data to make it more closely resemble a normal distribution. This is important because many statistical models and techniques assume that the data follows a normal distribution.

2. Homoscedasticity: The Yeo-Johnson transform helps to address heteroscedasticity, which is the unequal variance of the residuals in a regression model. By transforming the data, the variance can be stabilized, making it more consistent across different levels of the predictors.

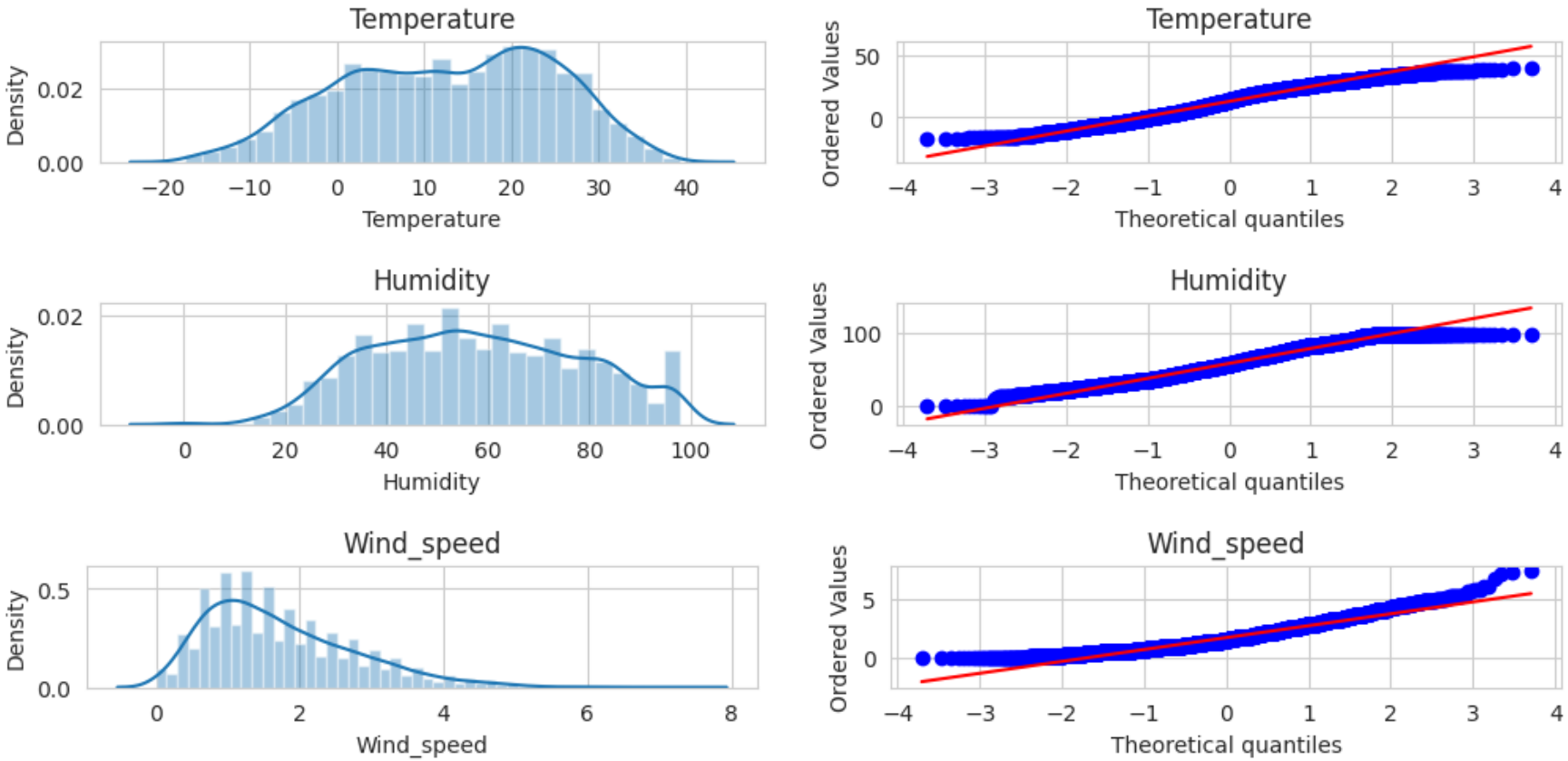
3. Outliers: The Yeo-Johnson transform is less sensitive to outliers compared to other transformations like the logarithmic or square root transform. It can handle a wider range of data values, including zeros and negative values.

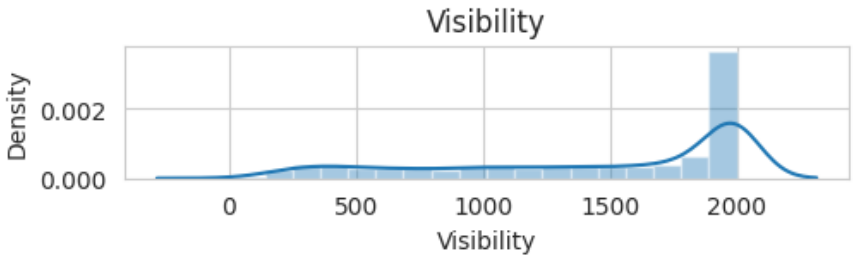
4. Flexibility: The Yeo-Johnson transform allows for different power parameters for positive and negative values, providing flexibility in handling asymmetric data distributions.

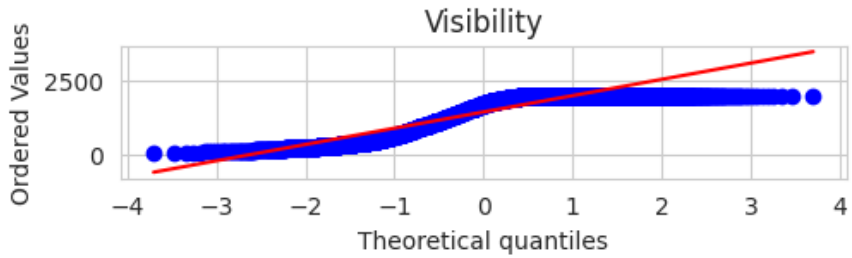
5. Interpretability: Unlike some other transformations, such as logarithmic or square root transforms, the Yeo-Johnson transform retains the ability to interpret the transformed data on its original scale.

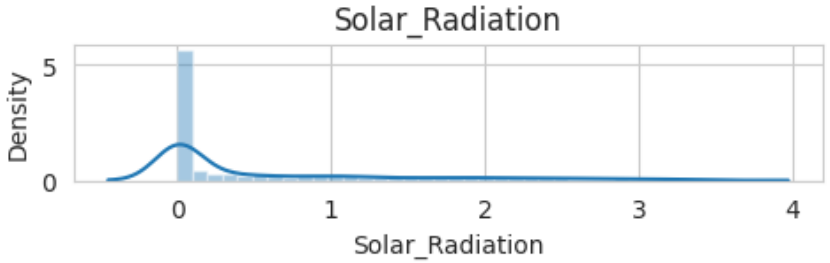
By applying the Yeo-Johnson transform to the data, we can address issues of skewness, heteroscedasticity, and non-normality, which can improve the performance of statistical models and provide more reliable insights and predictions.

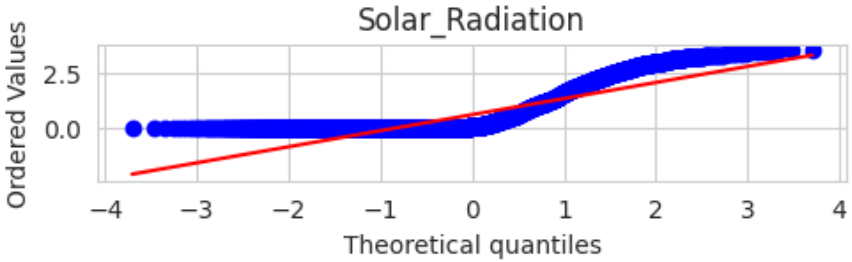
In [62]:
```
# Plotting the distplots without any transformation

for col in X_train.columns:
    plt.figure(figsize=(12,1))
    plt.subplot(121)
    sns.distplot(X_train[col])
    plt.title(col)

    plt.subplot(122)
    stats.probplot(X_train[col], dist="norm", plot=plt)
    plt.title(col)

    plt.show()
```

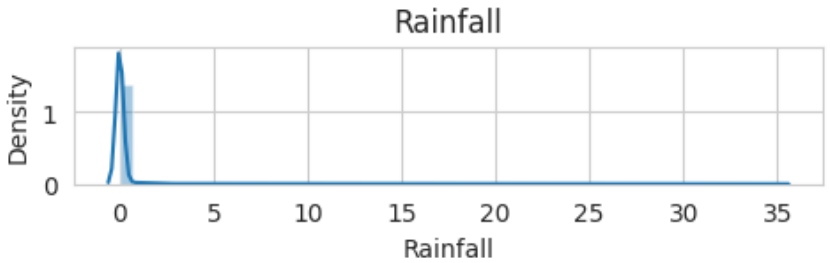


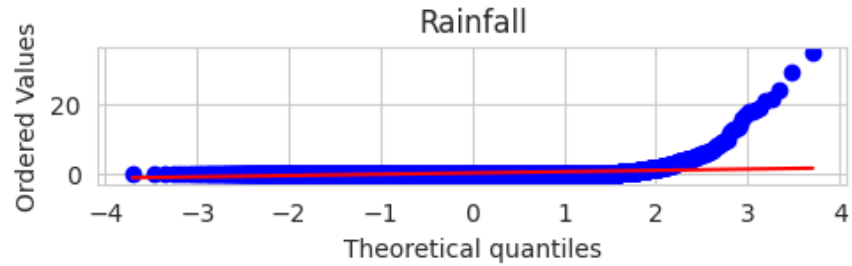

Visibility
Density
0.002
0.000
0
500
1000
1500
2000
Visibility

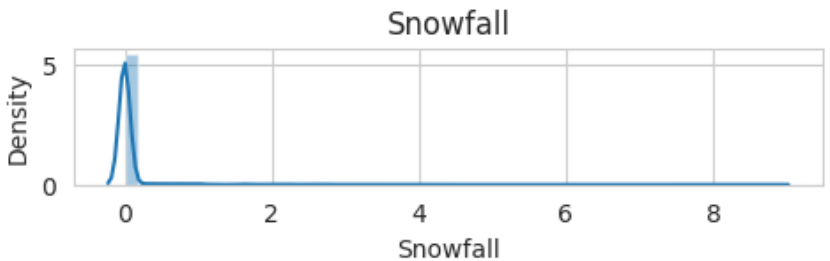

Visibility
Ordered Values
2500
0
−4
−3
−2
−1
0
1
2
3
4
Theoretical quantiles

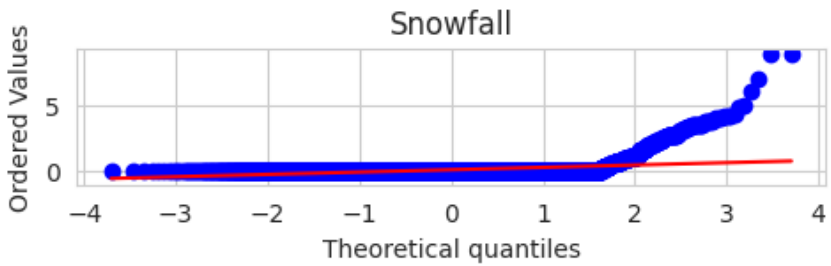

Solar_Radiation
Density
5
0
0
1
2
3
4
Solar_Radiation

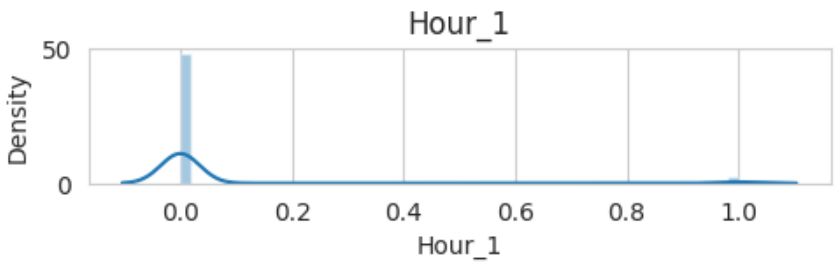

Solar_Radiation
Ordered Values
2.5
0.0
−4
−3
−2
−1
0
1
2
3
4
Theoretical quantiles

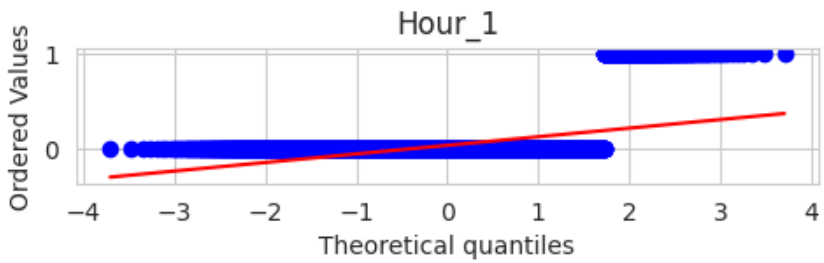

Rainfall
Density
1
0
0
5
10
15
20
25
30
35
Rainfall

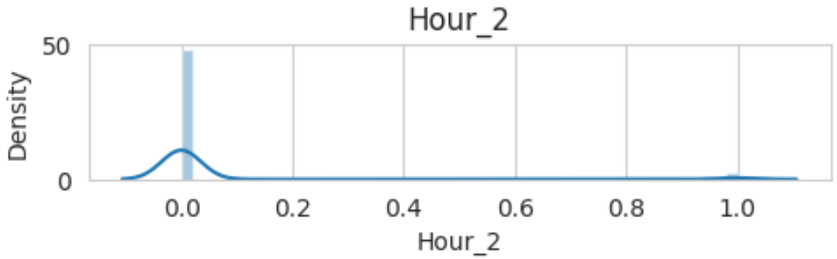

Rainfall
Ordered Values
20
0
−4
−3
−2
−1
0
1
2
3
4
Theoretical quantiles

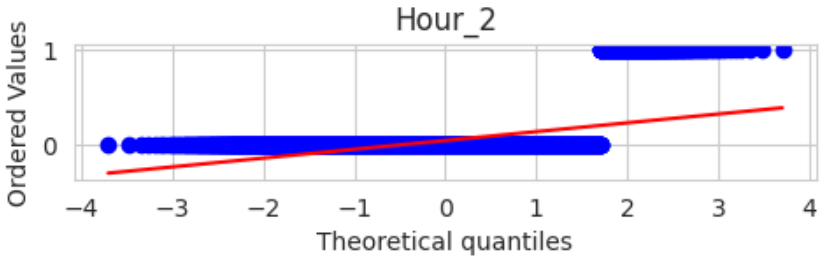

Snowfall
Density
5
0
0
2
4
6
8
Snowfall

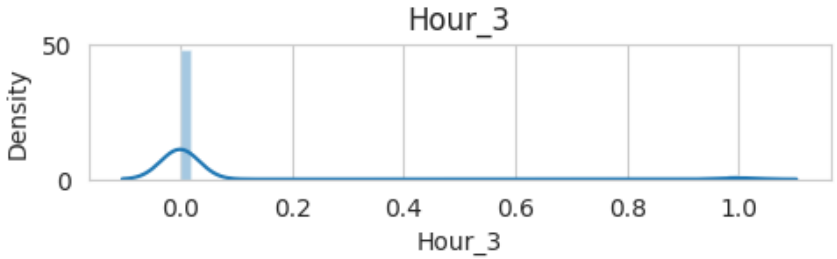

Snowfall
Ordered Values
5
0
−4
−3
−2
−1
0
1
2
3
4
Theoretical quantiles

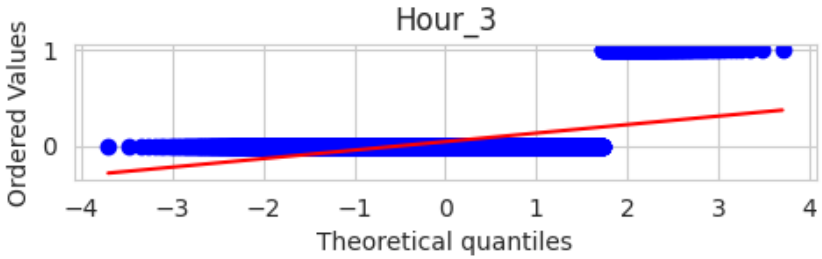

Hour_1
Density
50
0
0.0
0.2
0.4
0.6
0.8
1.0
Hour_1

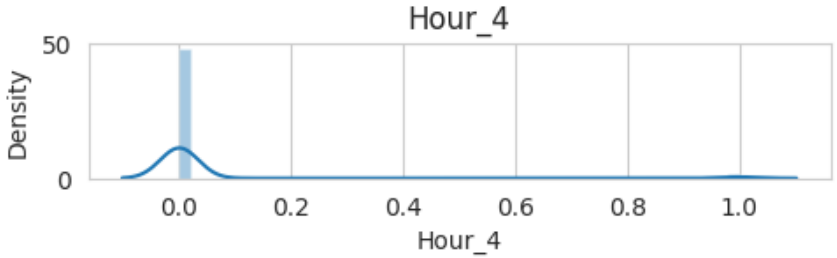

Hour_1
Ordered Values
1
0
−4
−3
−2
−1
0
1
2
3
4
Theoretical quantiles

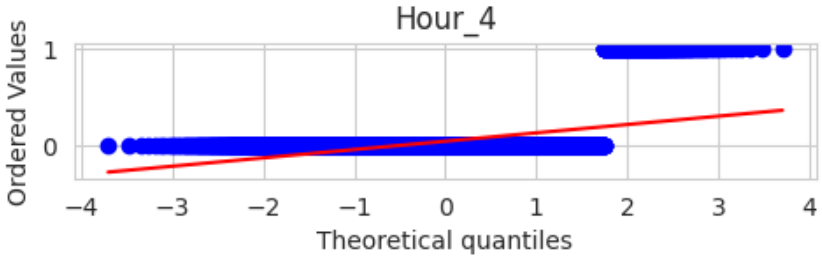

Hour_2
Density
Hour_2

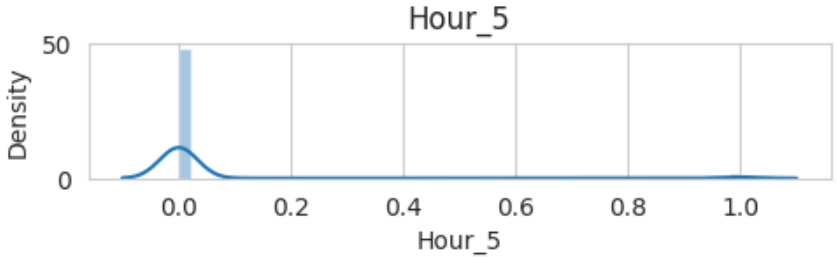

Hour_2
Ordered Values
Theoretical quantiles

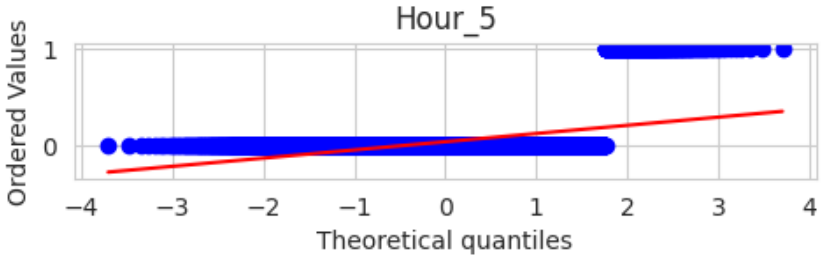

Hour_3
Density
Hour_3

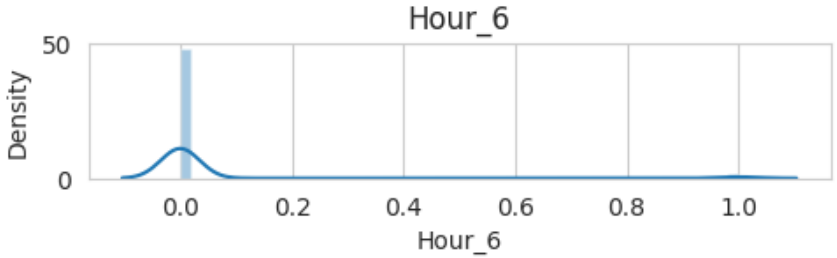

Hour_3
Ordered Values
Theoretical quantiles

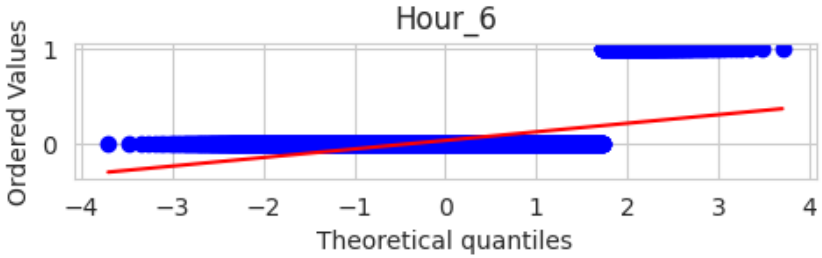

Hour_4
Density
Hour_4

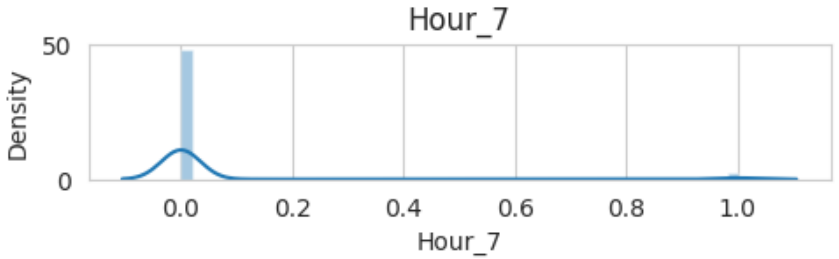

Hour_4
Ordered Values
Theoretical quantiles

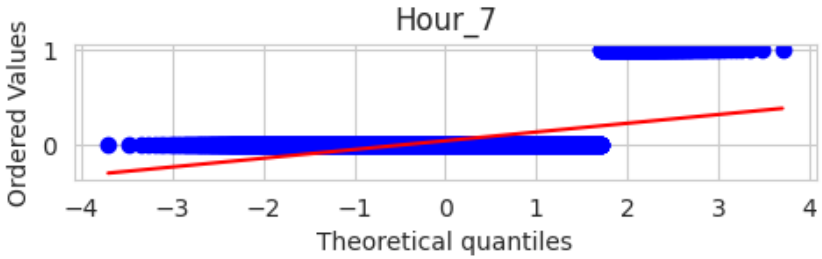

Hour_5
Density
Hour_5

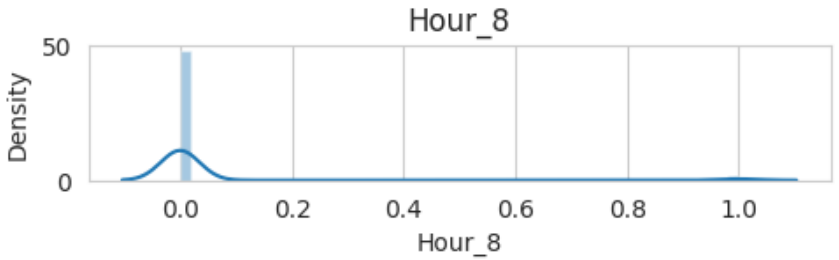

Hour_5
Ordered Values
Theoretical quantiles

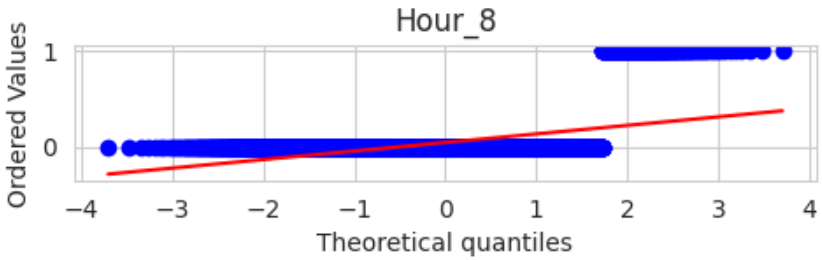

Hour_6
Density
Hour_6

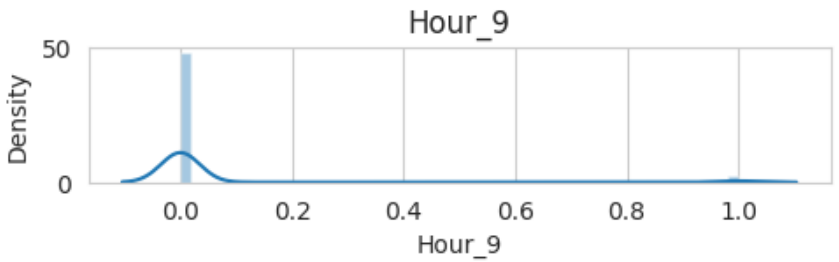

Hour_6
Ordered Values
Theoretical quantiles

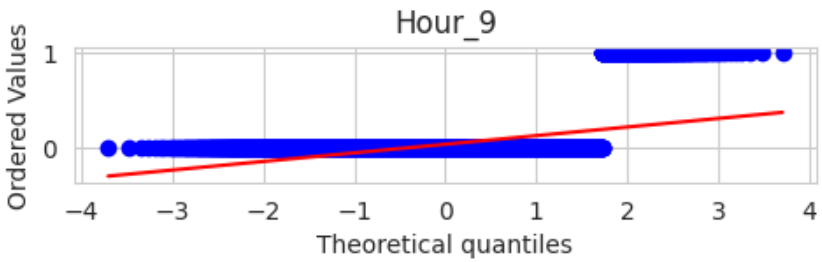

Hour_7
Density
Hour_7

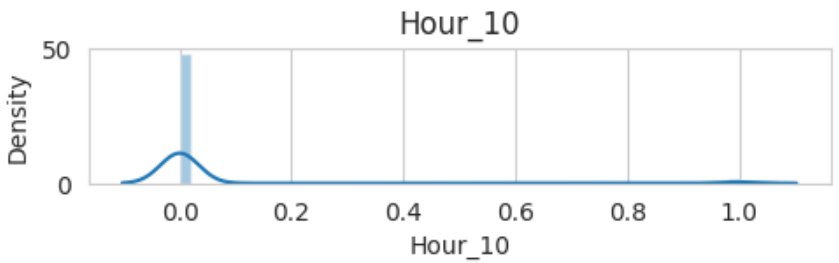

Hour_7
Ordered Values
Theoretical quantiles

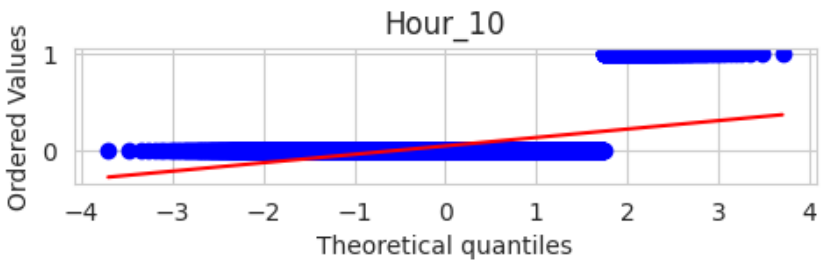

Hour_8
Density
Hour_8

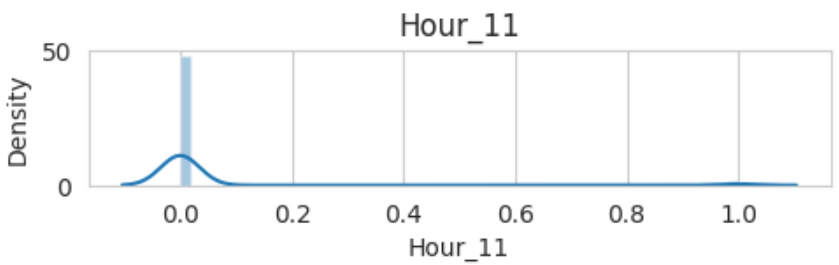

Hour_8
Ordered Values
Theoretical quantiles

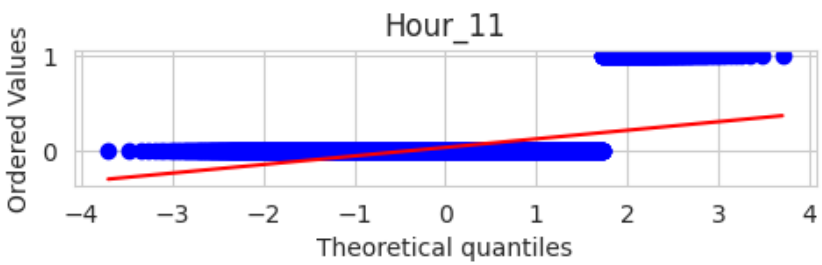

Hour_9
Density
Hour_9

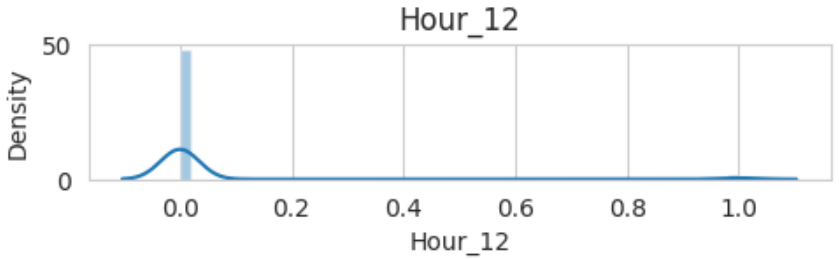

Hour_9
Ordered Values
Theoretical quantiles

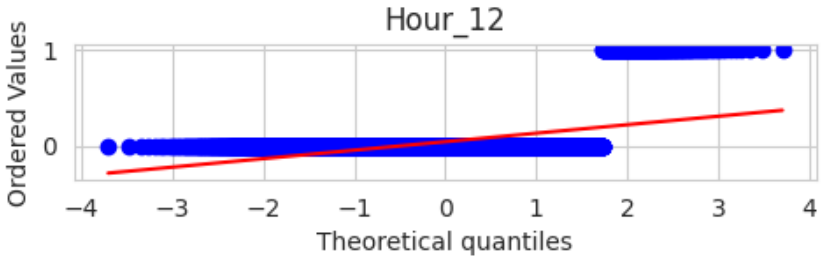

Hour_10
Density
Hour_10

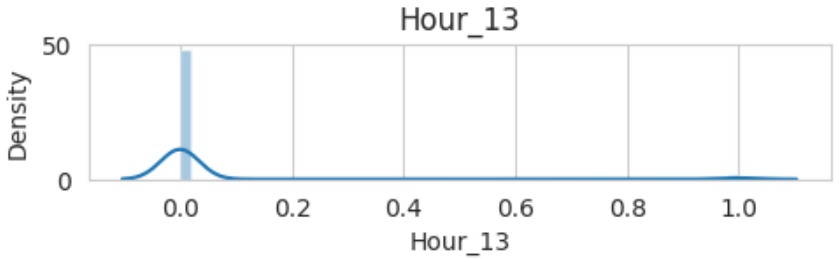

Hour_10
Ordered Values
Theoretical quantiles

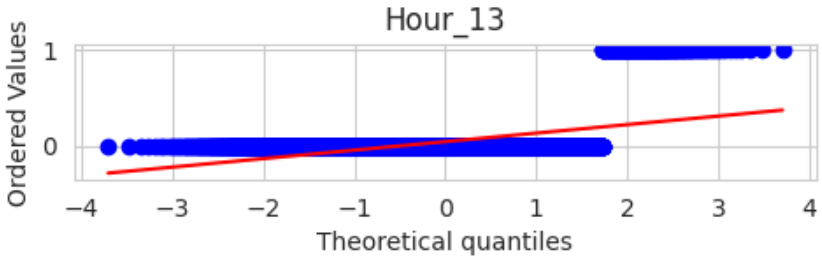

Hour_11
Density
Hour_11

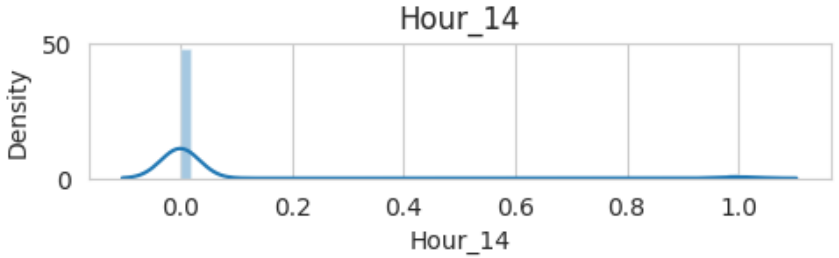

Hour_11
Ordered Values
Theoretical quantiles

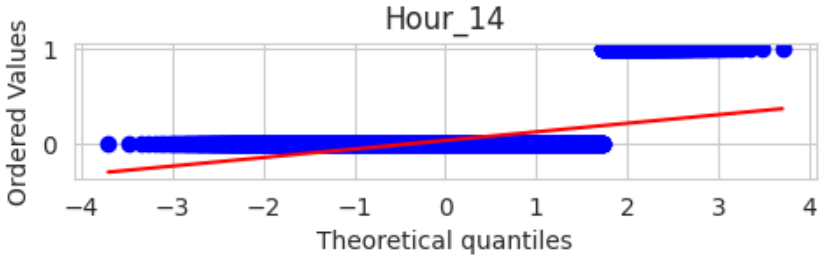

Hour_12
Density
Hour_12

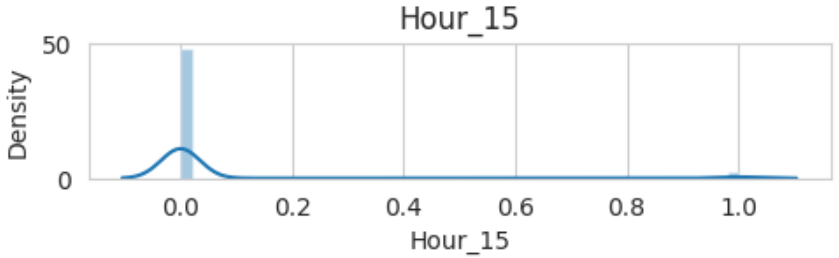

Hour_12
Ordered Values
Theoretical quantiles

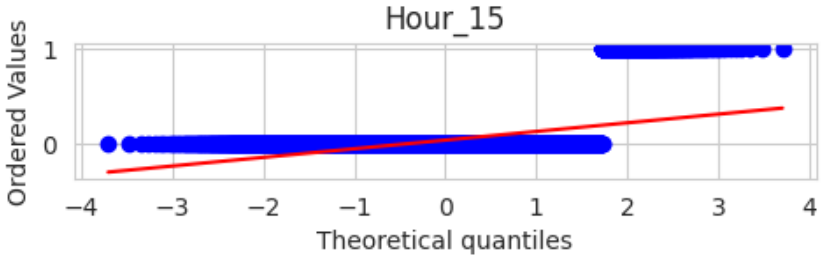

Hour_13
Density
Hour_13

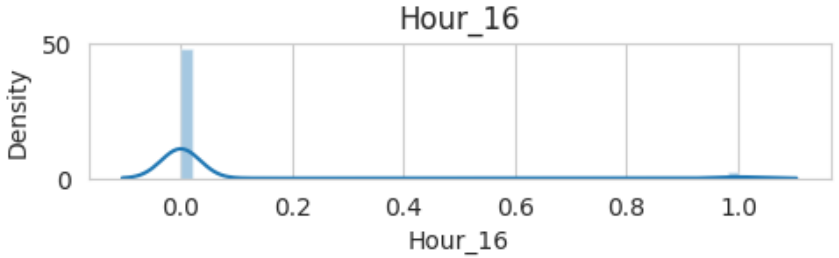

Hour_13
Ordered Values
Theoretical quantiles

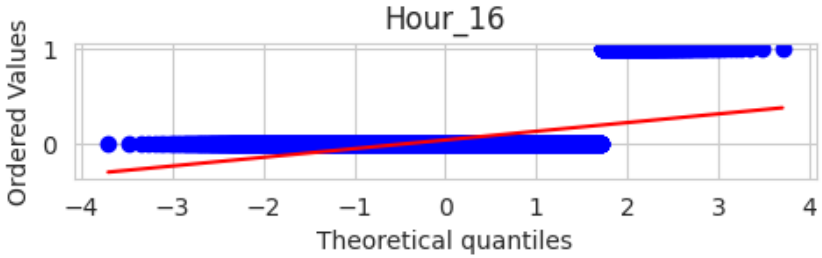

Hour_14
Density
Hour_14

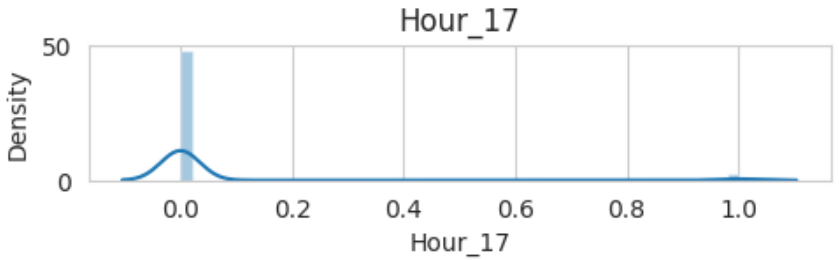

Hour_14
Ordered Values
Theoretical quantiles

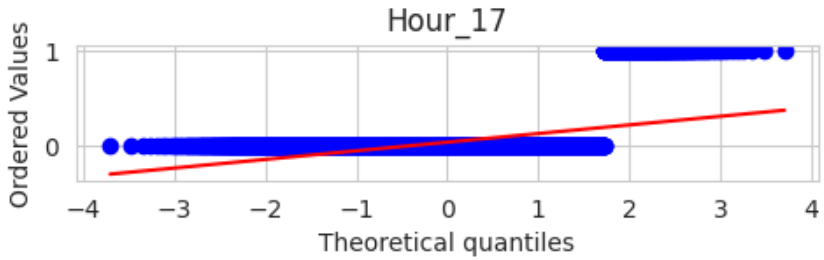

Hour_15
Density
Hour_15

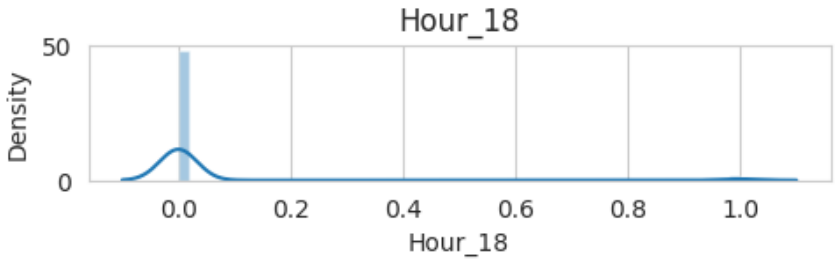

Hour_15
Ordered Values
Theoretical quantiles

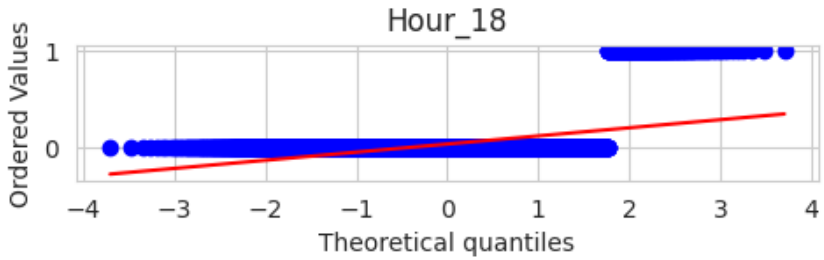

Hour_16
Density
Hour_16

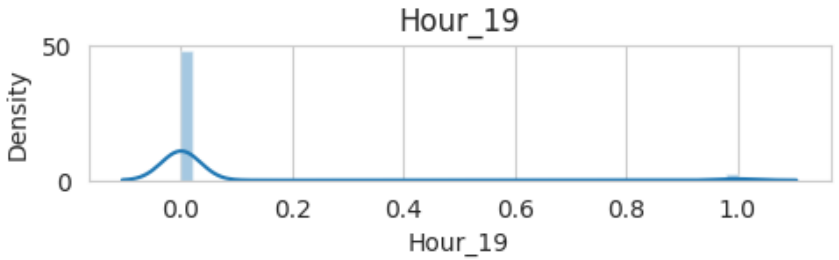

Hour_16
Ordered Values
Theoretical quantiles

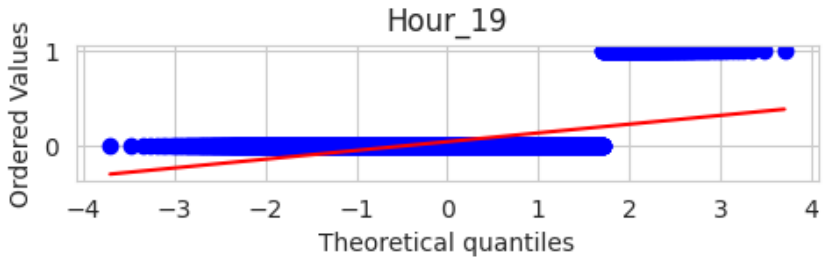

Hour_17
Density
Hour_17

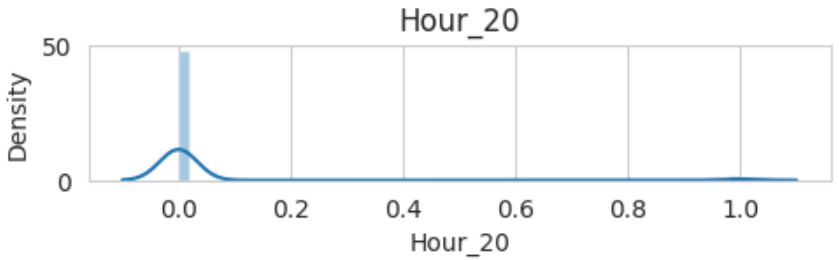

Hour_17
Ordered Values
Theoretical quantiles

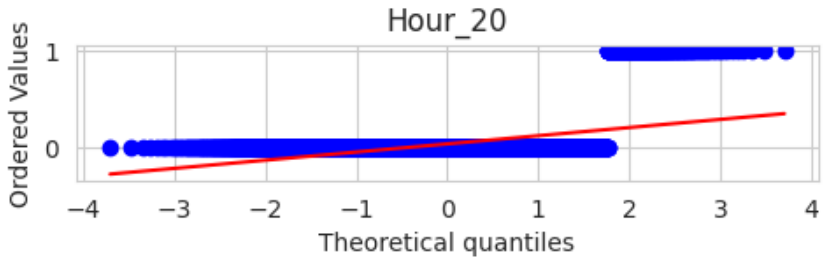

Hour_18
Density
Hour_18

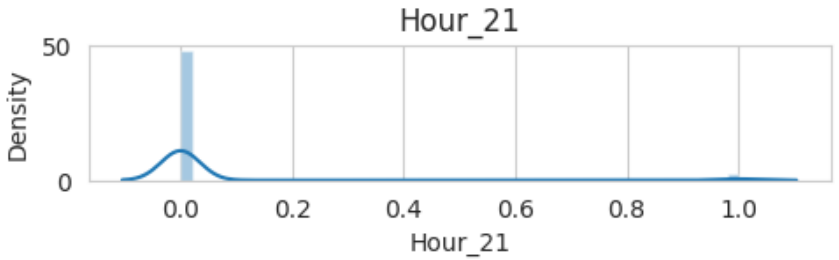

Hour_18
Ordered Values
Theoretical quantiles

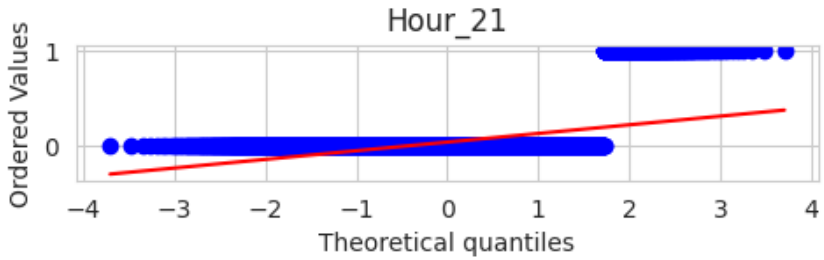

Hour_19
Density
Hour_19

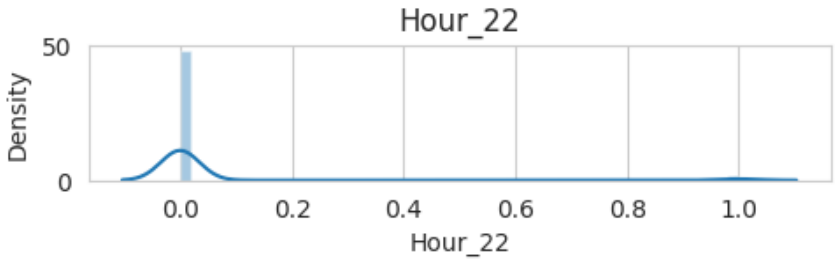

Hour_19
Ordered Values
Theoretical quantiles

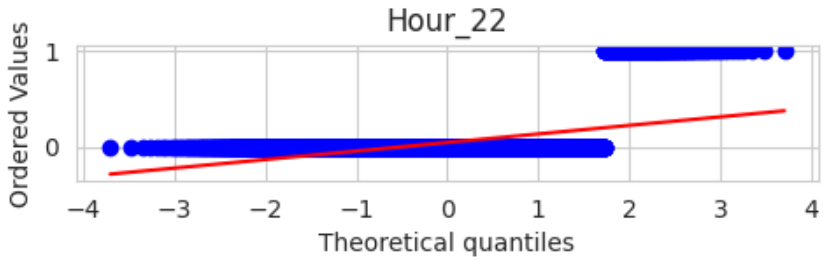

Hour_20
Density
Hour_20

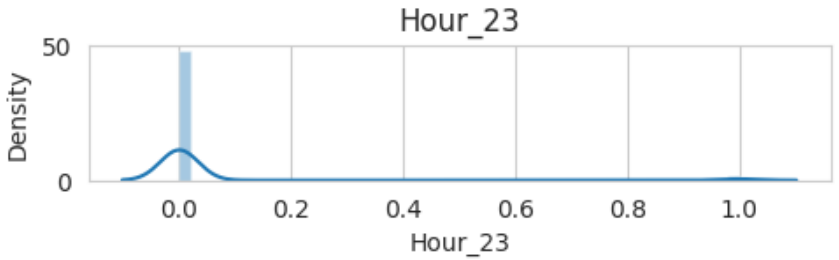

Hour_20
Ordered Values
Theoretical quantiles

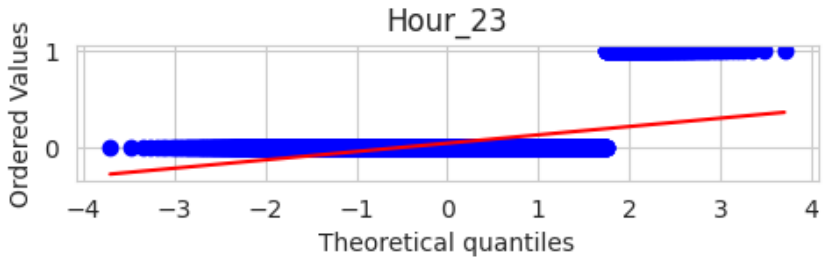

Hour_21
Density
Hour_21

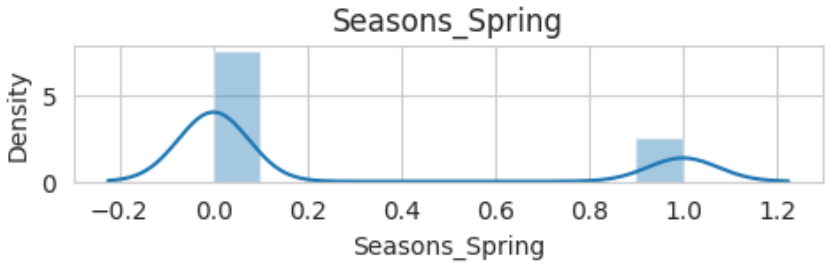

Hour_21
Ordered Values
Theoretical quantiles

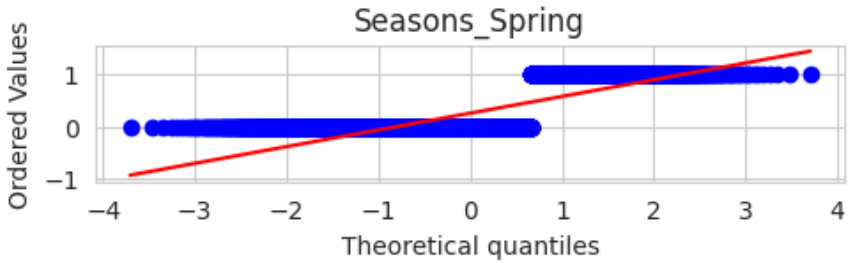

Hour_22
Density
Hour_22

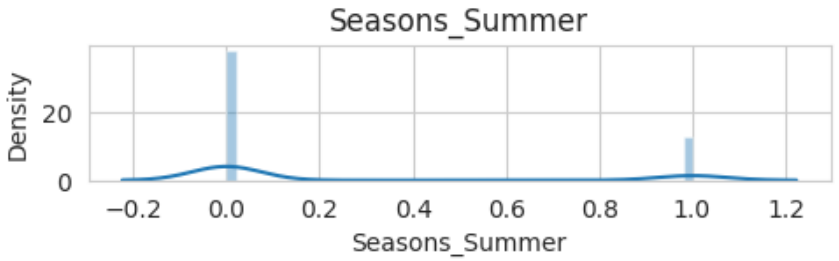

Hour_22
Ordered Values
Theoretical quantiles

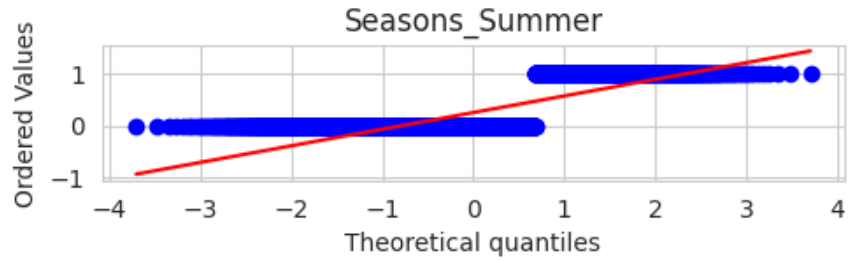

Hour_23
Density
Hour_23

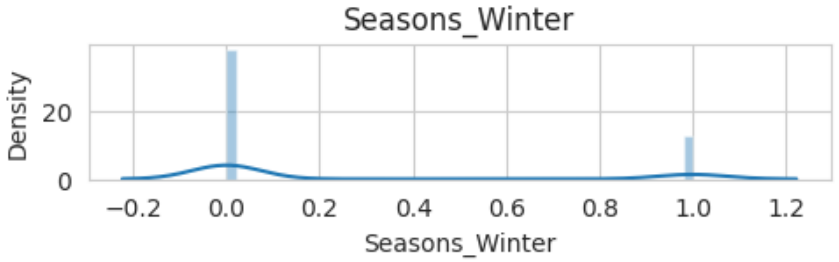

Hour_23
Ordered Values
Theoretical quantiles

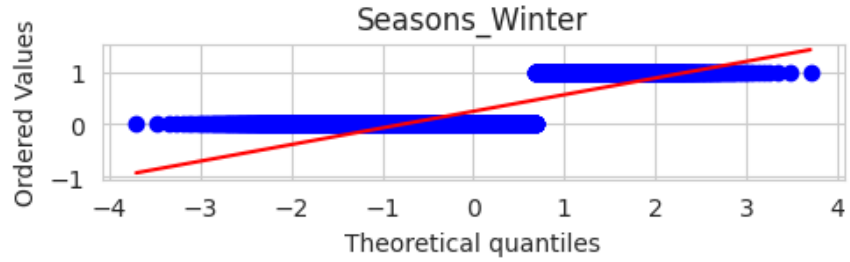

Seasons_Spring
Density
Seasons_Spring

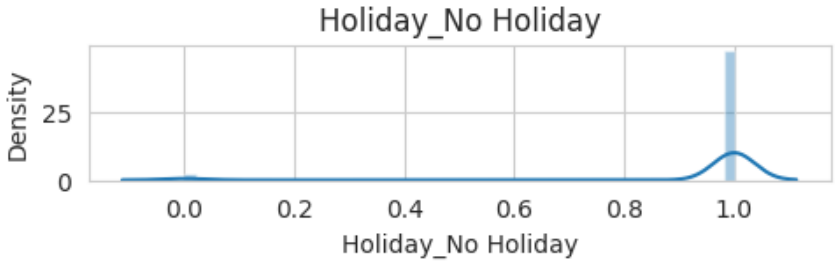

Seasons_Spring
Ordered Values
Theoretical quantiles

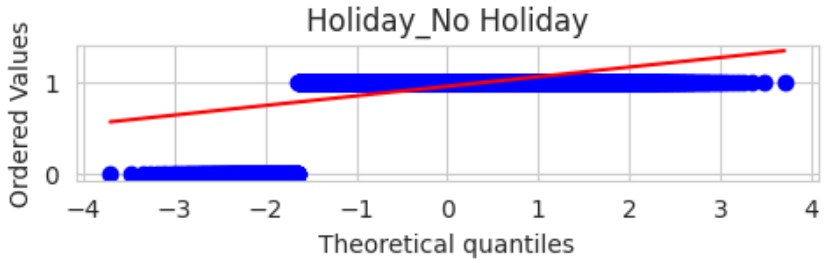

Seasons_Summer
Density
Seasons_Summer

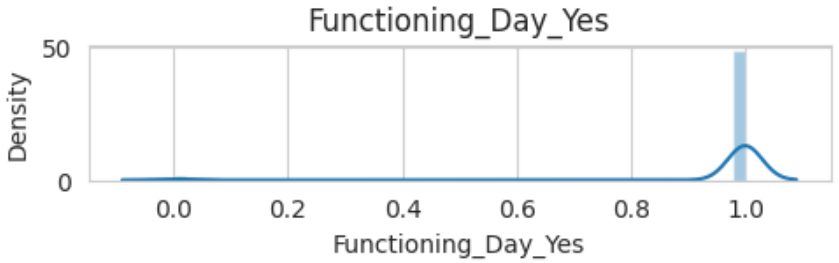

Seasons_Summer
Ordered Values
Theoretical quantiles

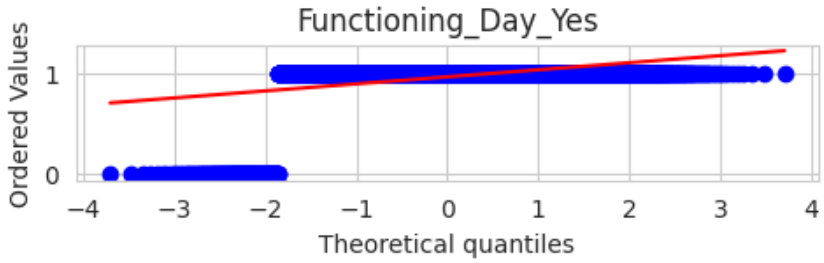

Seasons_Winter
Density
Seasons_Winter

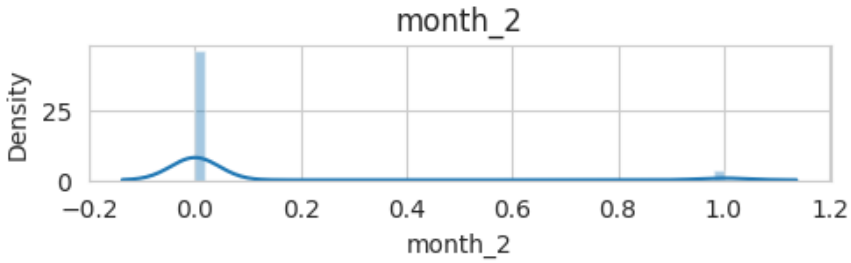

Seasons_Winter
Ordered Values
Theoretical quantiles

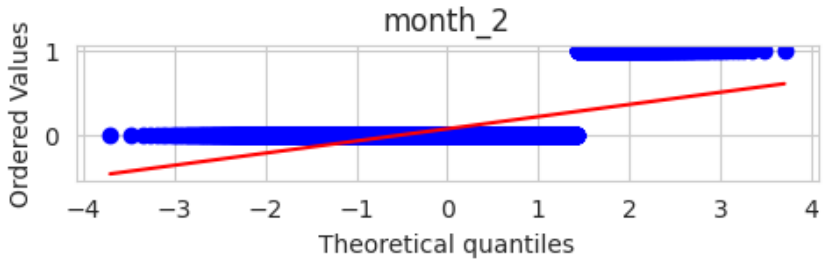

Holiday_No Holiday
Density
Holiday_No Holiday

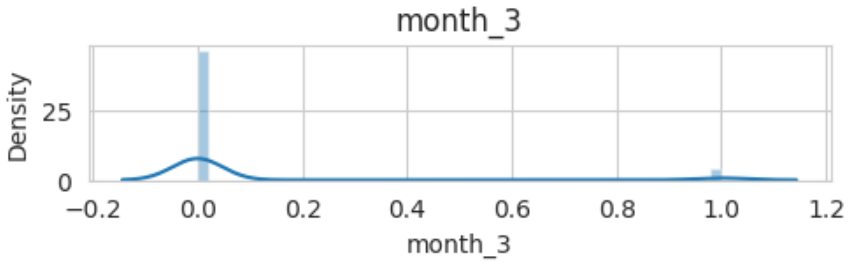

Holiday_No Holiday
Ordered Values
Theoretical quantiles

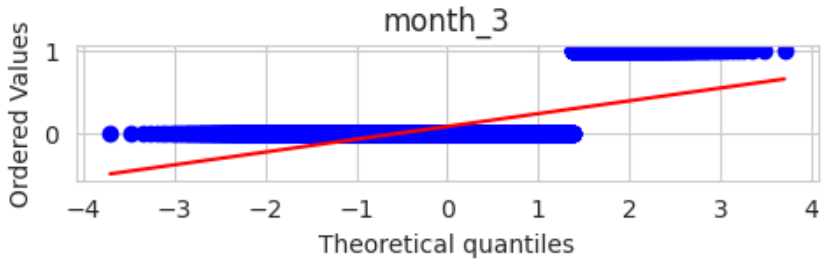

Functioning_Day_Yes
Density
Functioning_Day_Yes

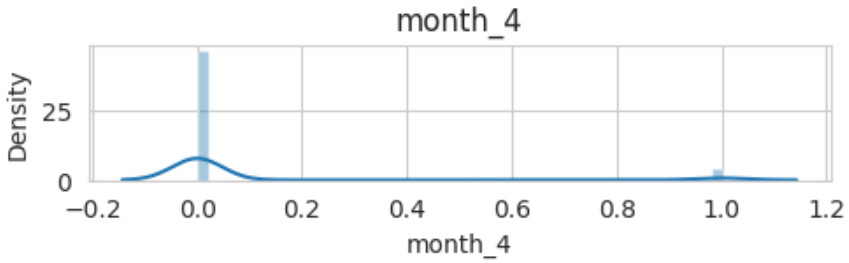

Functioning_Day_Yes
Ordered Values
Theoretical quantiles

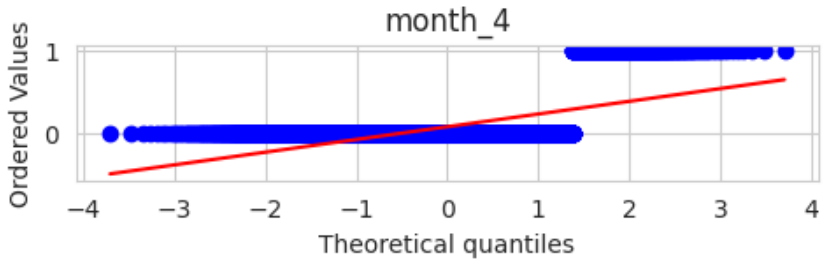

month_2
Density
month_2

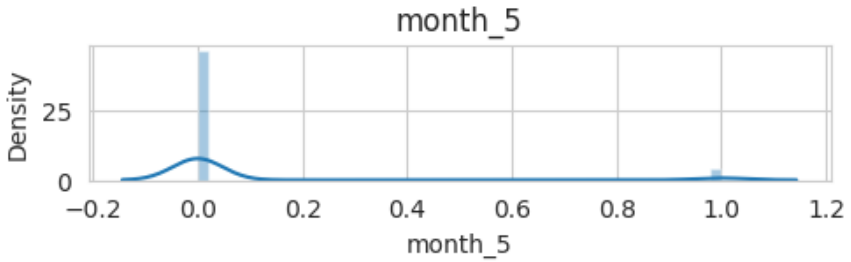

month_2
Ordered Values
Theoretical quantiles

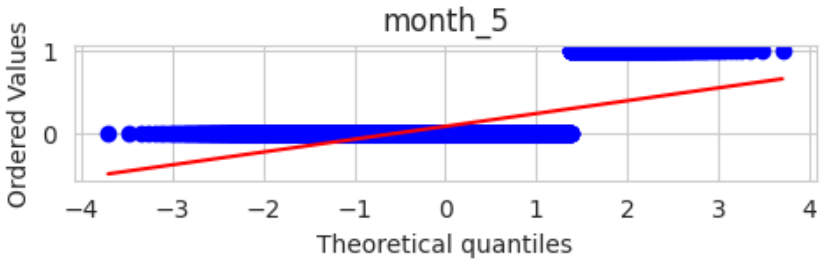

month_3
Density
month_3

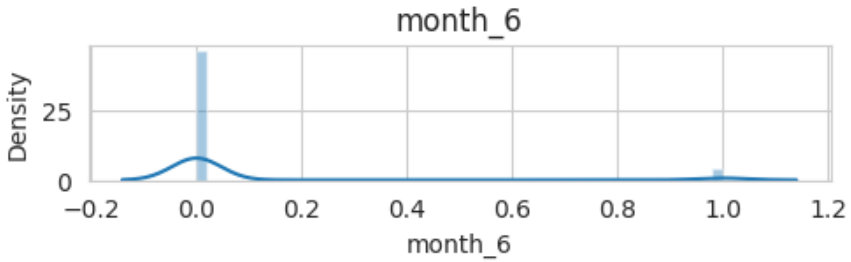

month_3
Ordered Values
Theoretical quantiles

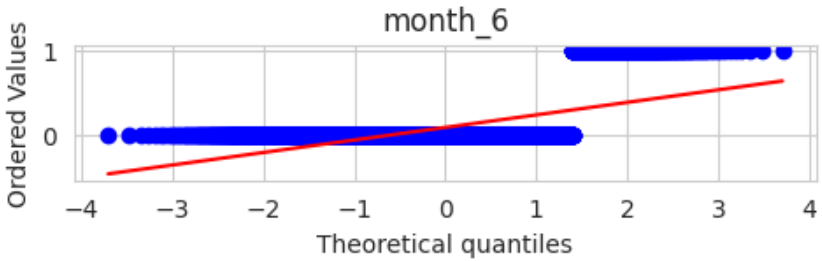

month_4
Density
month_4

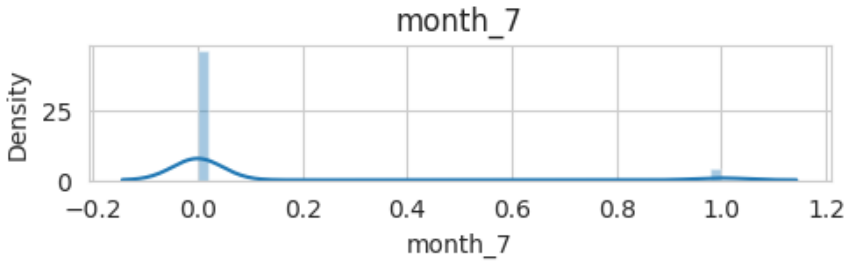

month_4
Ordered Values
Theoretical quantiles

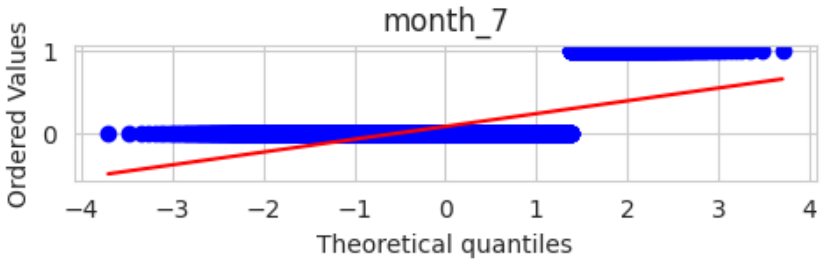

month_5
Density
month_5

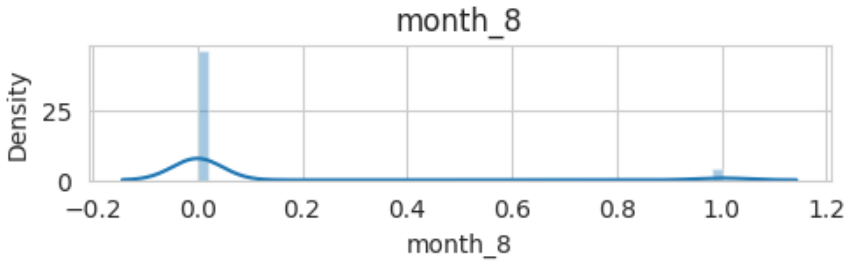

month_5
Ordered Values
Theoretical quantiles

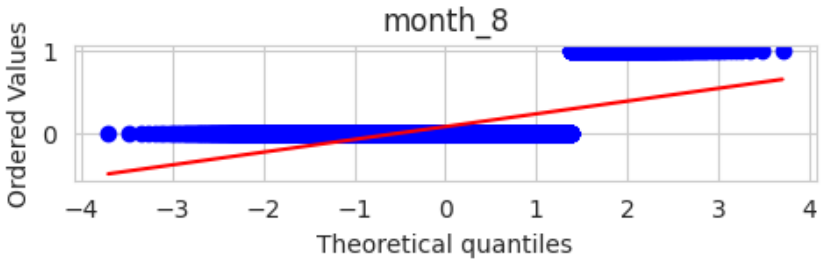

month_6
Density
month_6

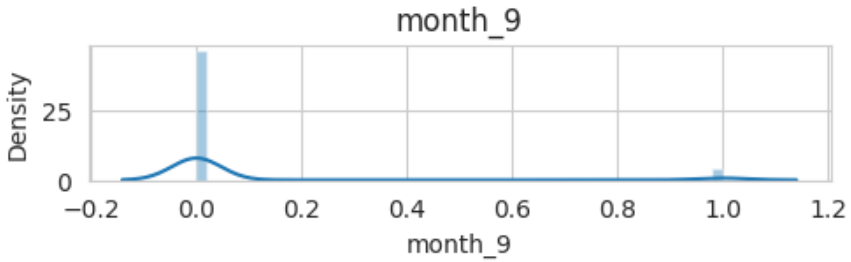

month_6
Ordered Values
Theoretical quantiles

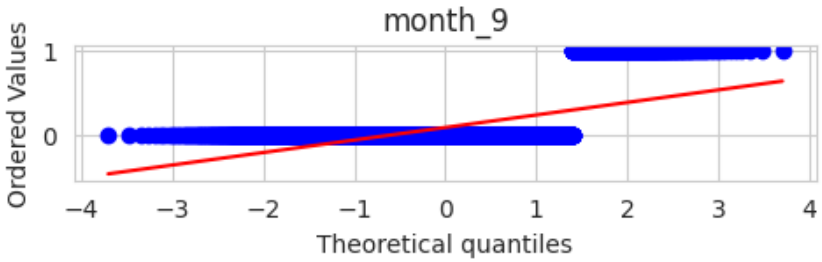

month_7
Density
month_7

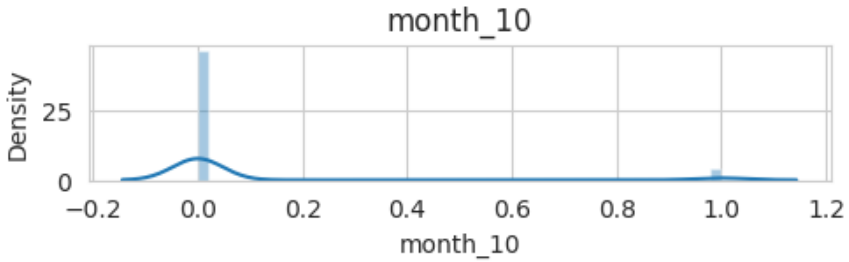

month_7
Ordered Values
Theoretical quantiles

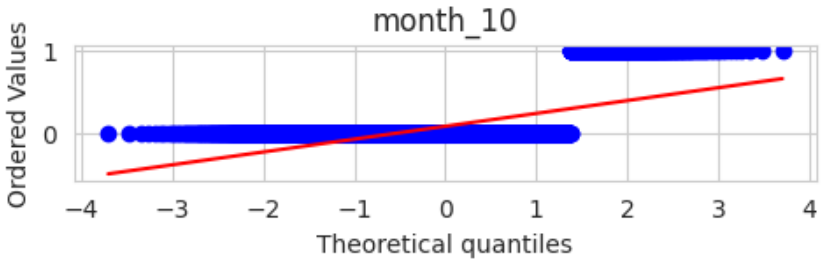

month_8
Density
month_8

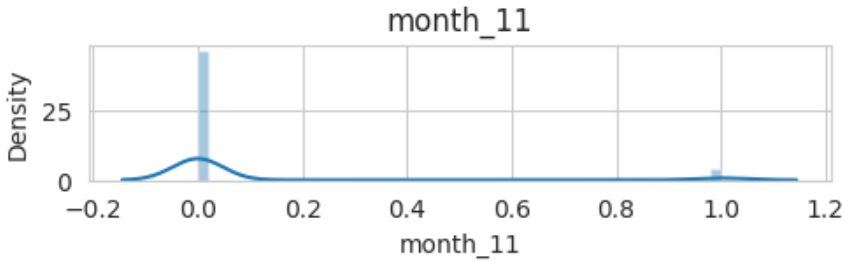

month_8
Ordered Values
Theoretical quantiles

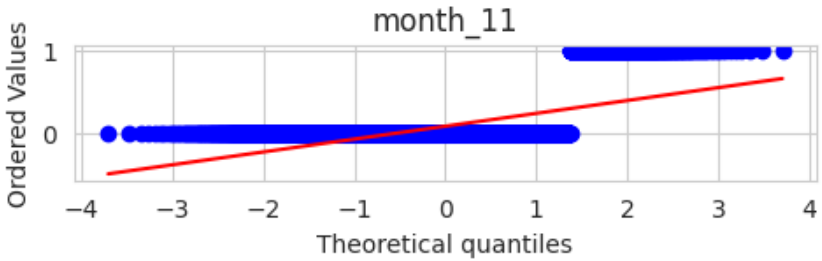

month_9
Density
month_9

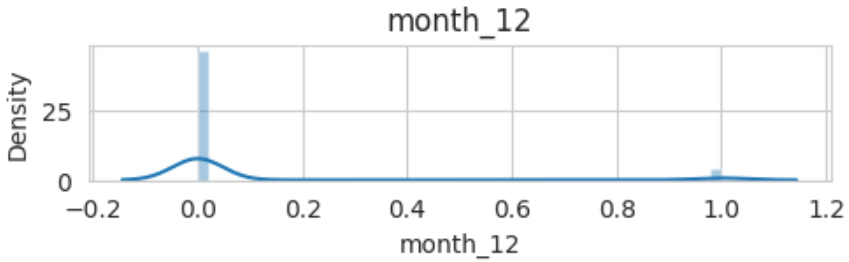

month_9
Ordered Values
Theoretical quantiles

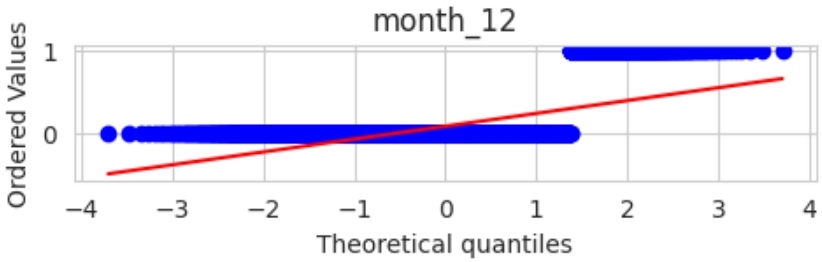


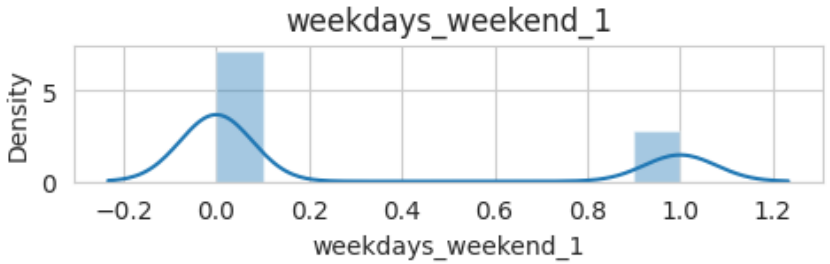


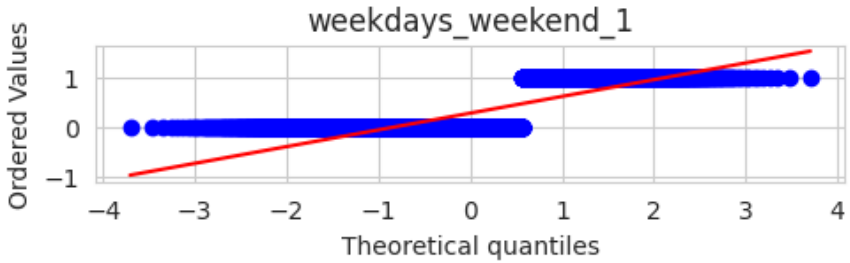












In [63]:

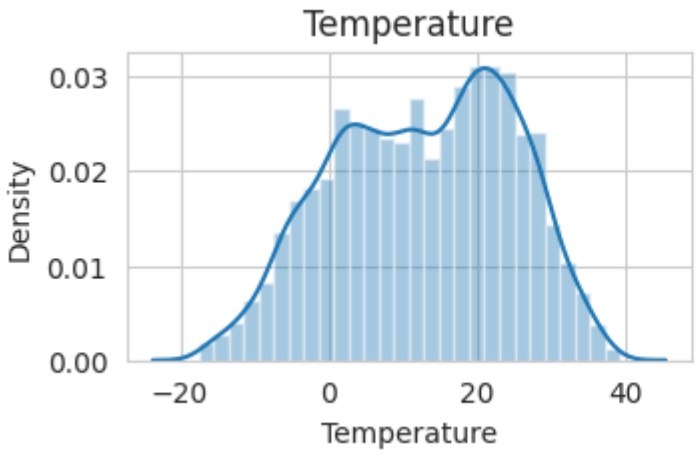
```
# Transform Your data
# Apply Yeo-Johnson transform


pt1 = PowerTransformer()
X_train_transformed2 = pt1.fit_transform(X_train)
X_test_transformed2 = pt1.transform(X_test)
```

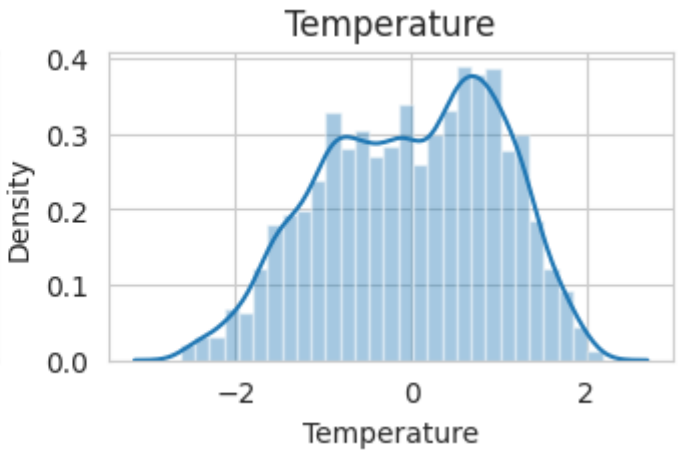
```
lr = LinearRegression()

# Perform cross-validation with 5 folds
cv_scores = cross_val_score(lr, X_train_transformed2, y_train, cv=5, scoring='r2')

# Train the model on the full training set
lr.fit(X_train_transformed2, y_train)

# Predict on the test set
y_pred3 = lr.predict(X_test_transformed2)

r2 = r2_score(y_test, y_pred3)
print("R-squared score:", r2)

df = pd.DataFrame({'cols': X_train.columns, 'Yeo_Johnson_lambdas': pt1.lambdas_})
# print(df)

print("Cross-validated R-squared scores:", cv_scores)
print("Mean R-squared score:", cv_scores.mean())
mean_r2 = np.mean(cv_scores)
print("Mean R-squared score:", mean_r2)
```

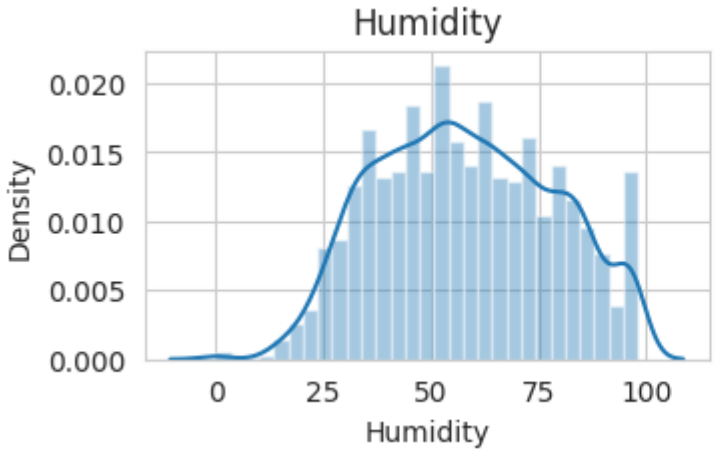
```
R-squared score: 0.8714812953128741
Cross-validated R-squared scores: [0.85014356 0.86088892 0.85701071 0.86165856 0.84973428]
Mean R-squared score: 0.8558872056469287
Mean R-squared score: 0.8558872056469287
```

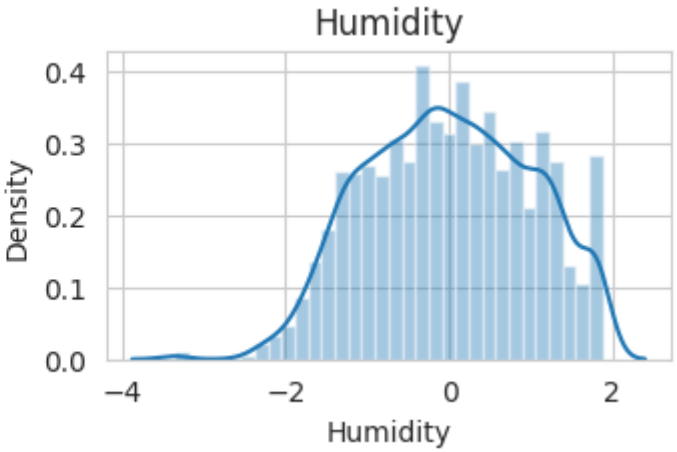
In [64]:

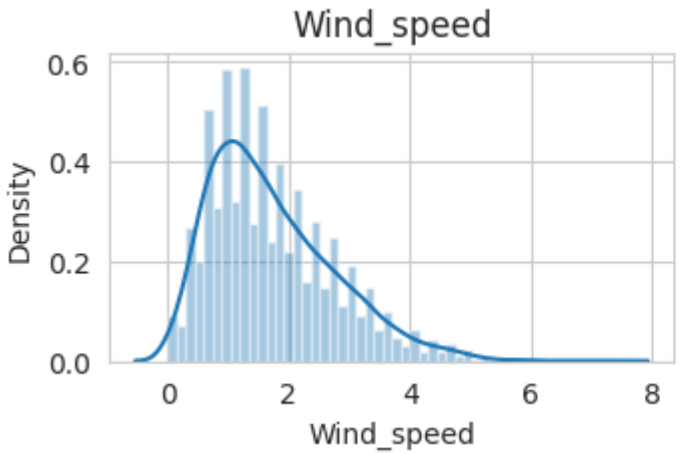
```
X_train_transformed = pd.DataFrame(X_train_transformed2,columns=X_train.columns)

for col in X_train_transformed.columns:
    plt.figure(figsize=(8,2))
    plt.subplot(121)
    sns.distplot(X_train[col])
    plt.title(col)

    plt.subplot(122)
    sns.distplot(X_train_transformed[col])
    plt.title(col)

    plt.show()
```

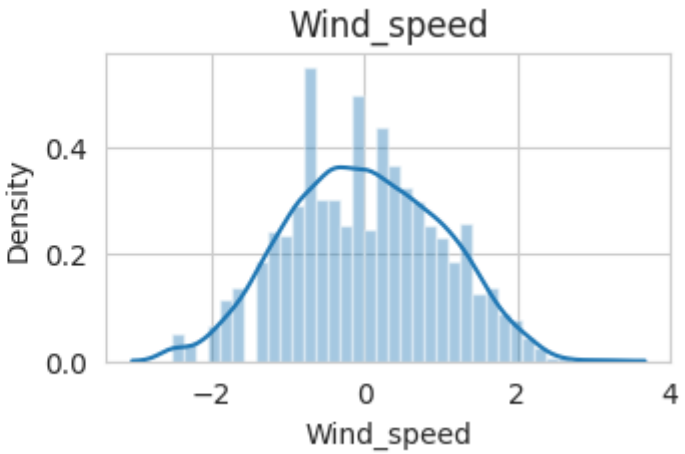

Temperature
Density
0.03
0.02
0.01
0.00
−20
0
20
40
Temperature

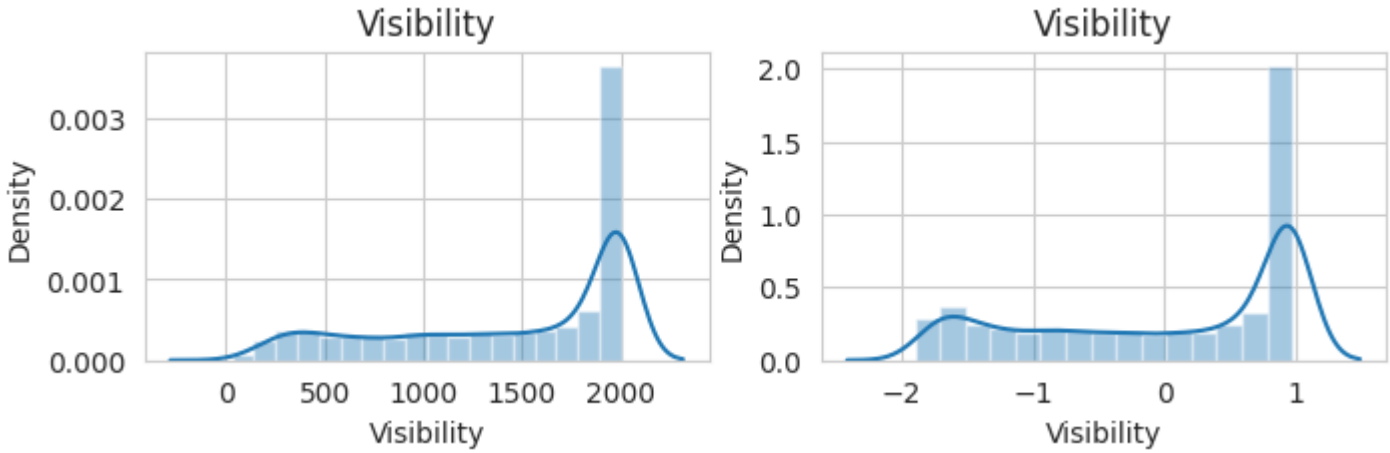

Temperature
Density
0.4
0.3
0.2
0.1
0.0
−2
0
2
Temperature


Humidity
Density
0.020
0.015
0.010
0.005
0.000
0
25
50
75
100
Humidity

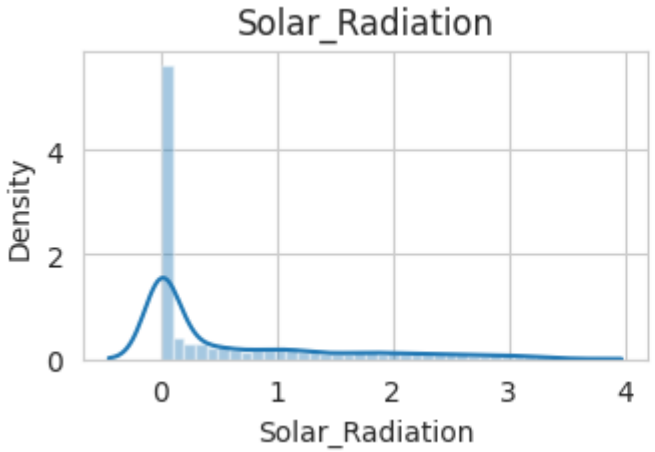

Humidity
Density
0.4
0.3
0.2
0.1
0.0
−4
−2
0
2
Humidity

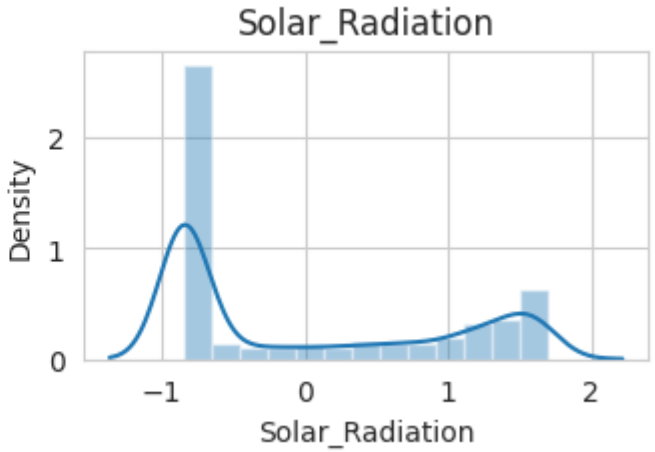

Wind_speed
Density
0.6
0.4
0.2
0.0
0
2
4
6
8
Wind_speed

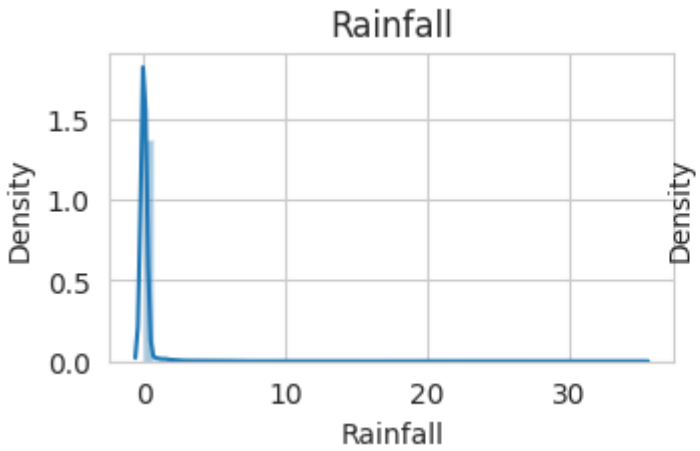

Wind_speed
Density
0.4
0.2
0.0
−2
0
2
4
Wind_speed

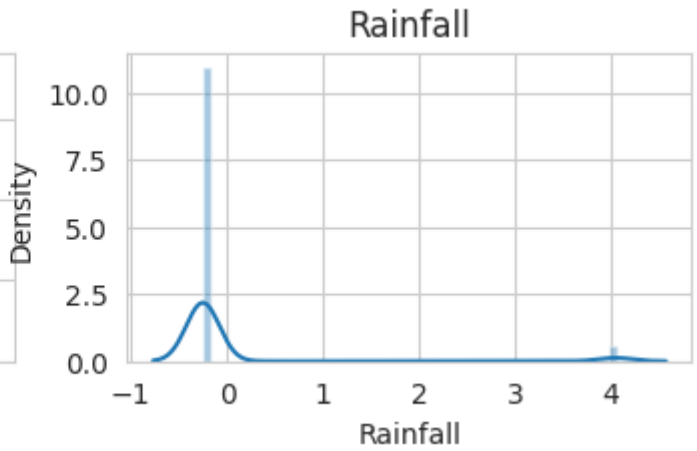

Visibility
Density
0.003
0.002
0.001
0.000
0
500
1000
1500
2000
Visibility

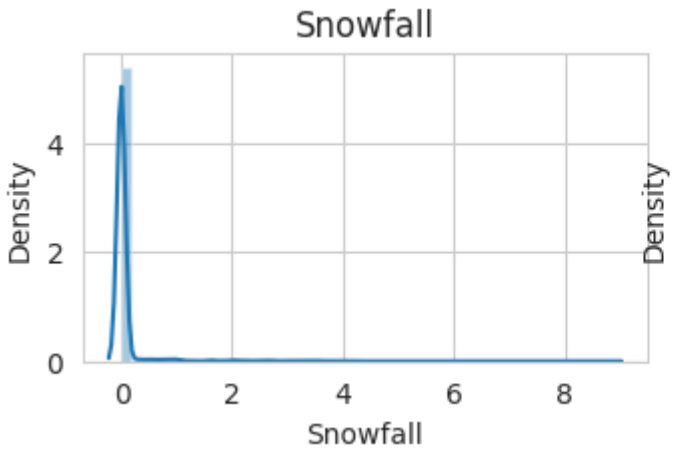
Visibility
Density
2.0
1.5
1.0
0.5
0.0
−2
−1
0
1
Visibility

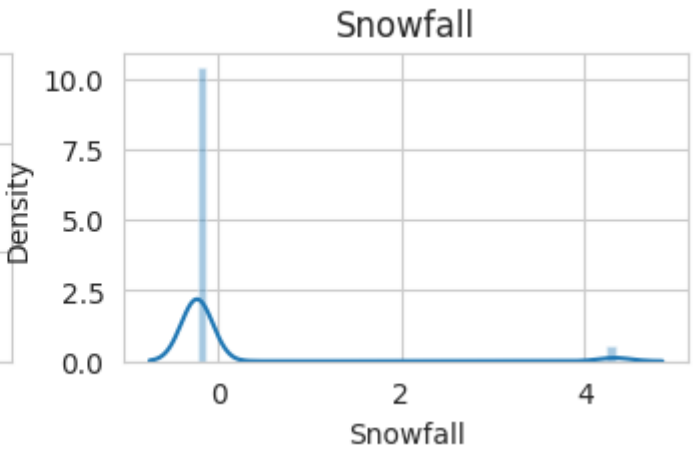

Solar_Radiation
Density
4
2
0
0
1
2
3
4
Solar_Radiation

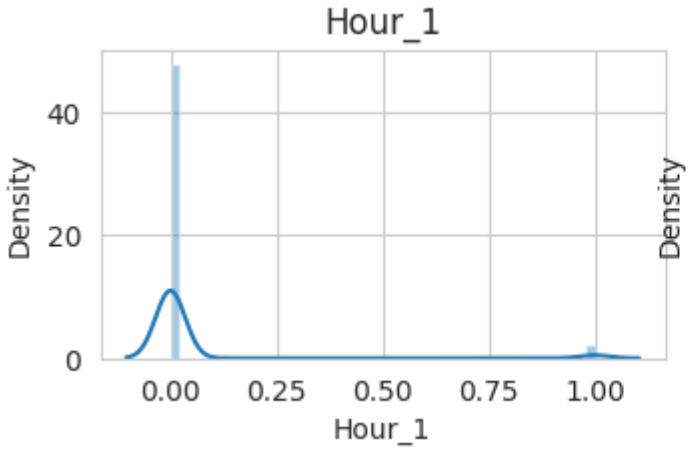

Solar_Radiation
Density
2
1
0
−1
0
1
2
Solar_Radiation

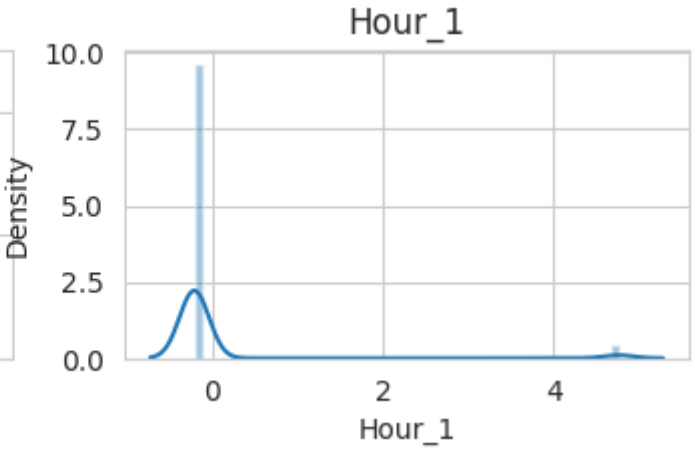

Rainfall
Density
1.5
1.0
0.5
0.0
0
10
20
30
Rainfall

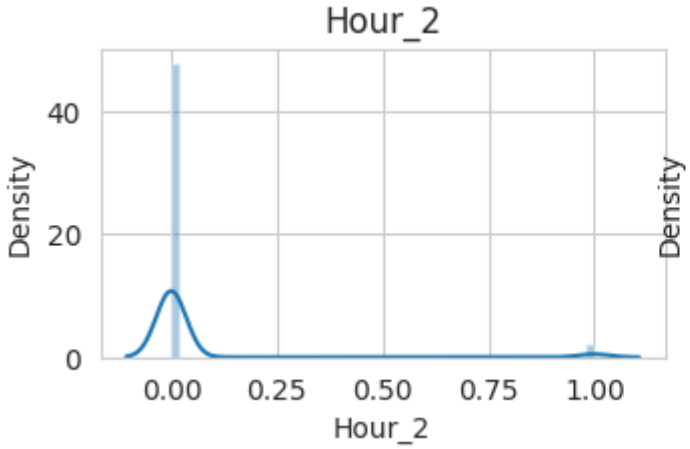

Rainfall
Density
10.0
7.5
5.0
2.5
0.0
−1
0
1
2
3
4
Rainfall

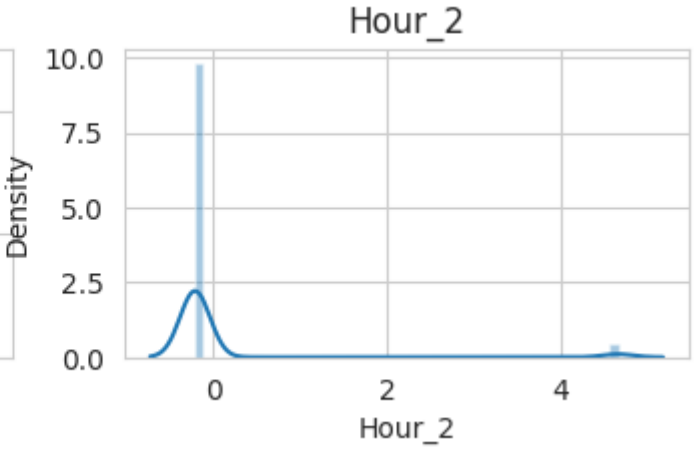

Snowfall
Density
Snowfall
0
2
4
6
8

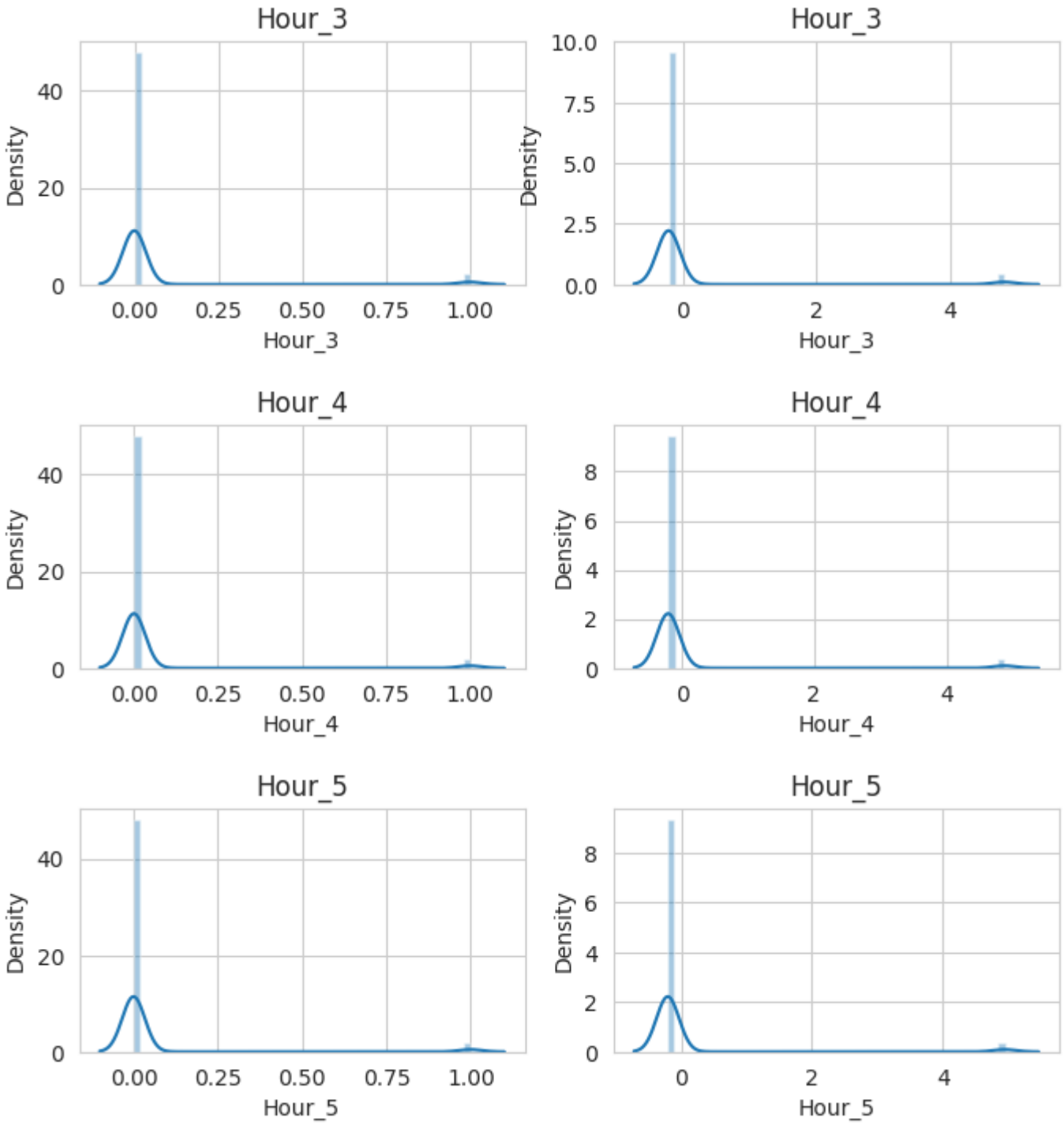

Snowfall
Density
Snowfall
0.0
2.5
5.0
7.5
10.0
0
2
4

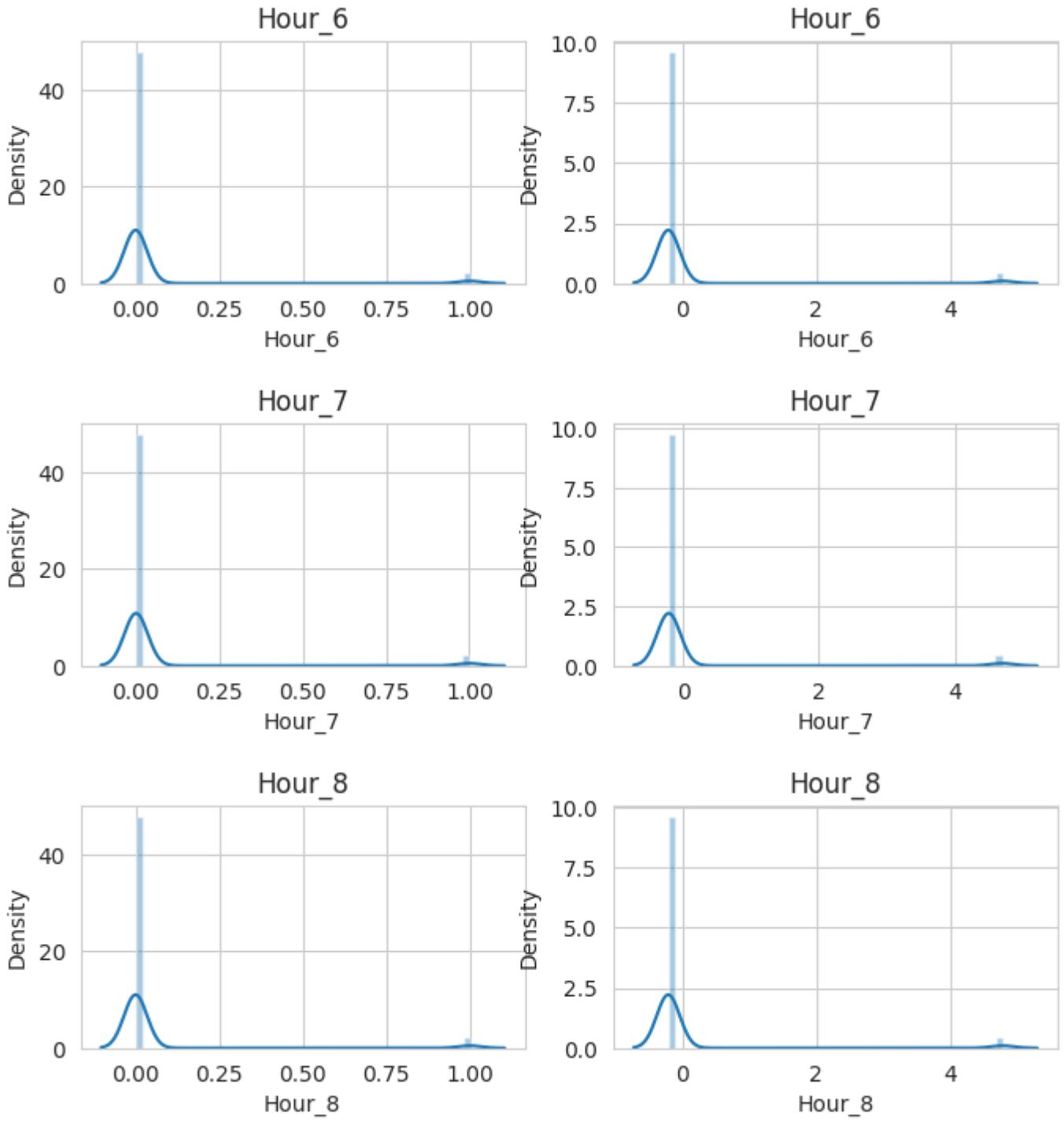

Hour_1
Density
Hour_1
0
20
40
0.00
0.25
0.50
0.75
1.00

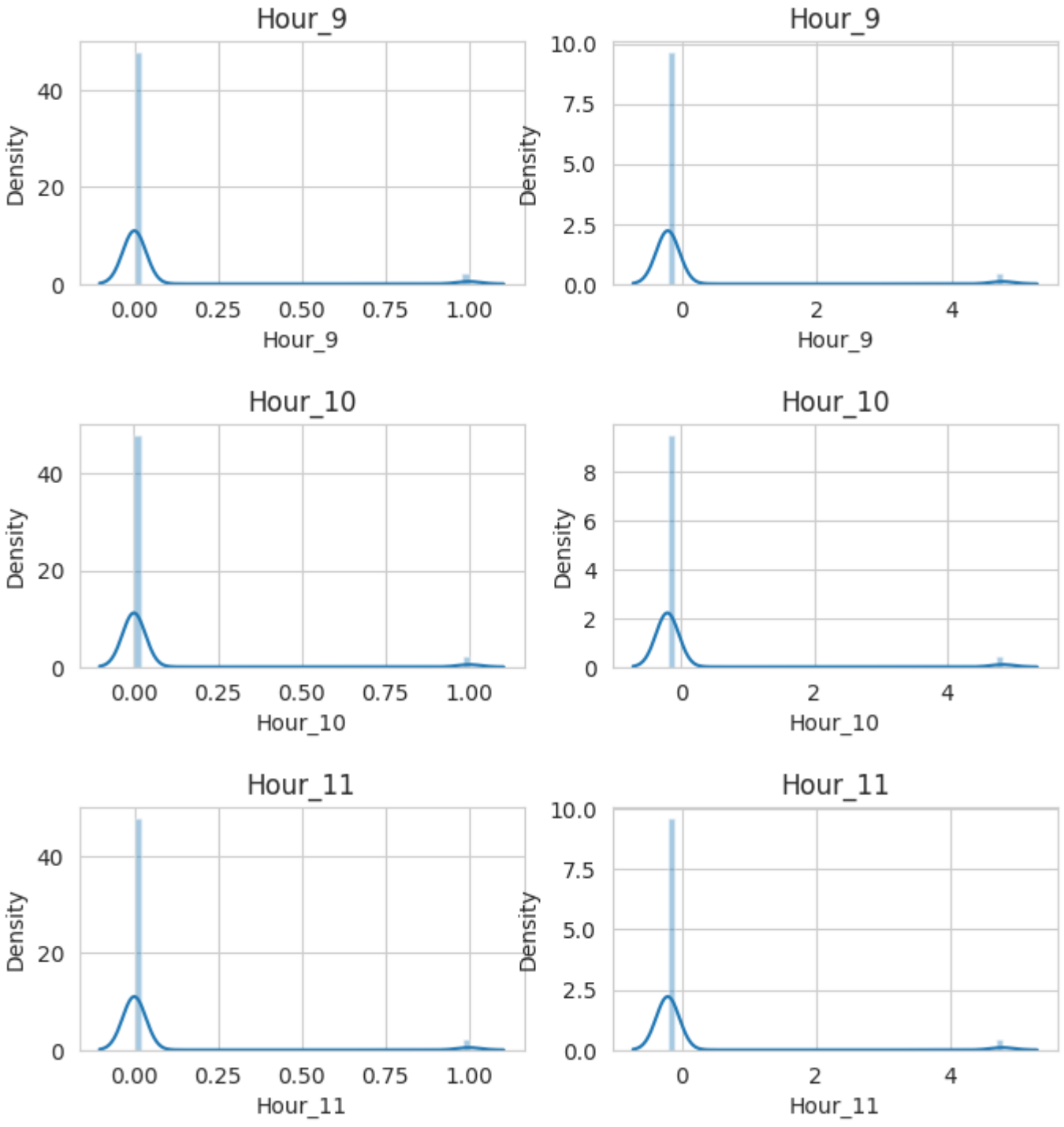

Hour_1
Density
Hour_1
0.0
2.5
5.0
7.5
10.0
0
2
4

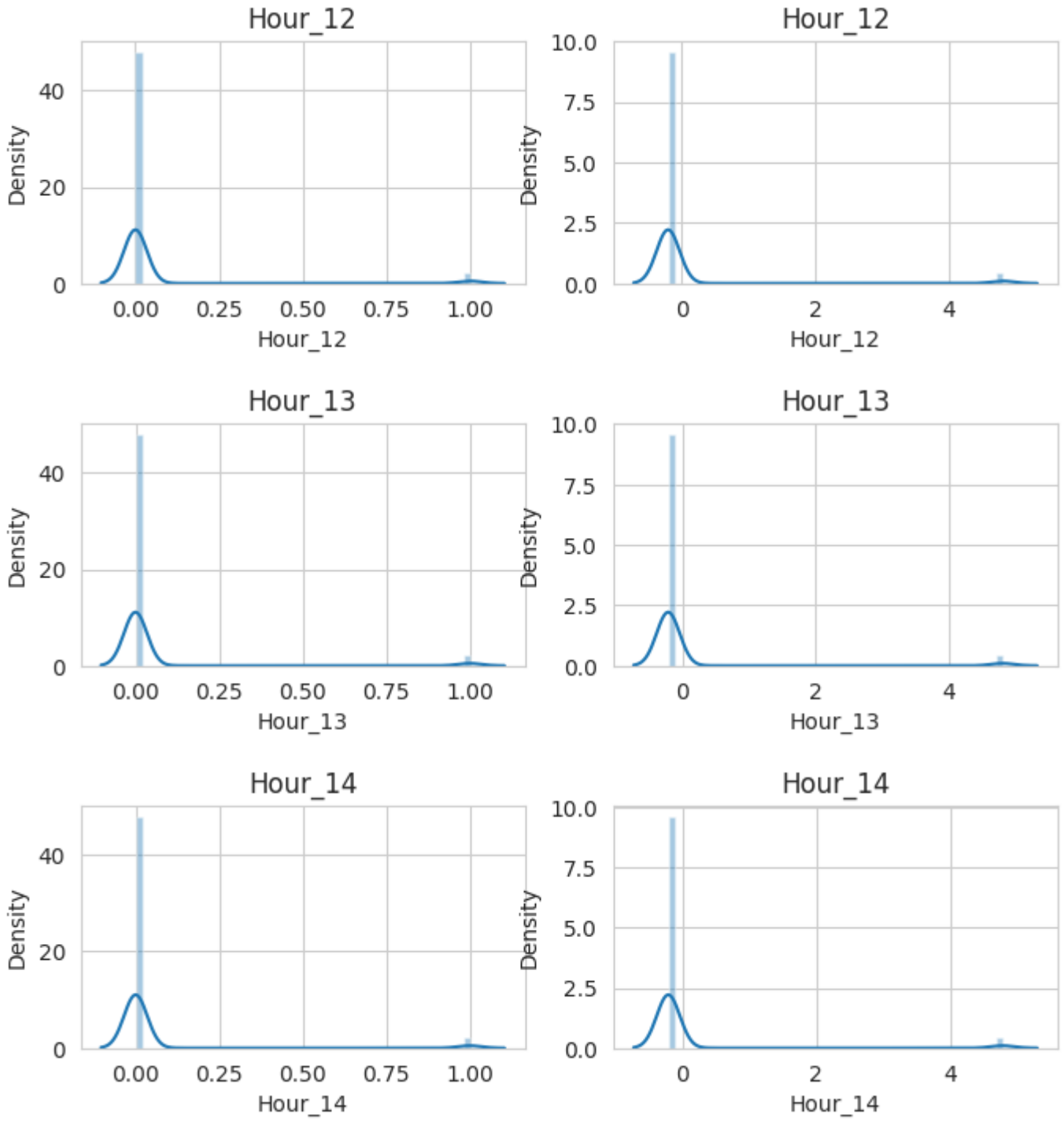

Hour_2
Density
Hour_2
0
20
40
0.00
0.25
0.50
0.75
1.00

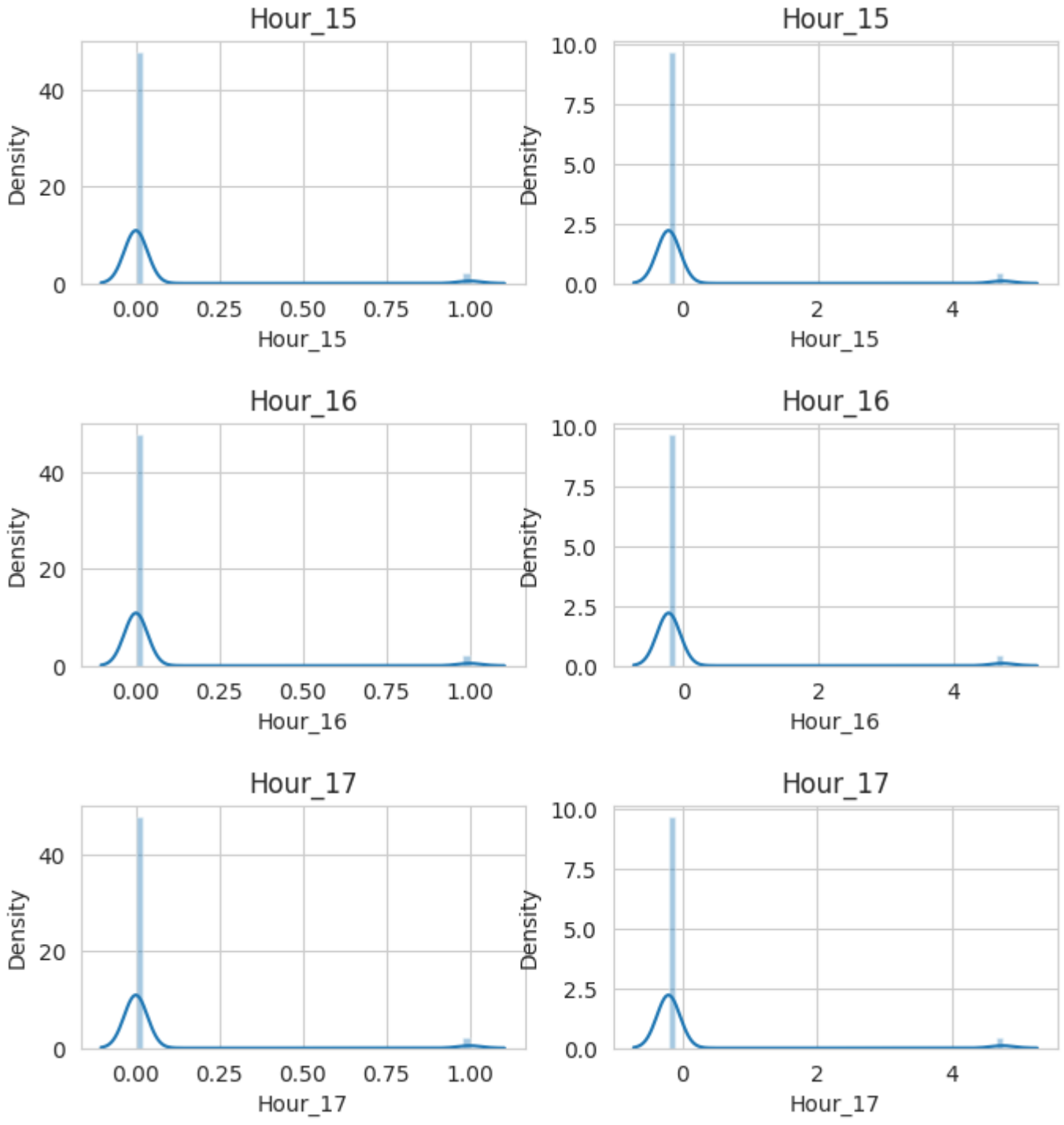

Hour_2
Density
Hour_2
0.0
2.5
5.0
7.5
10.0
0
2
4

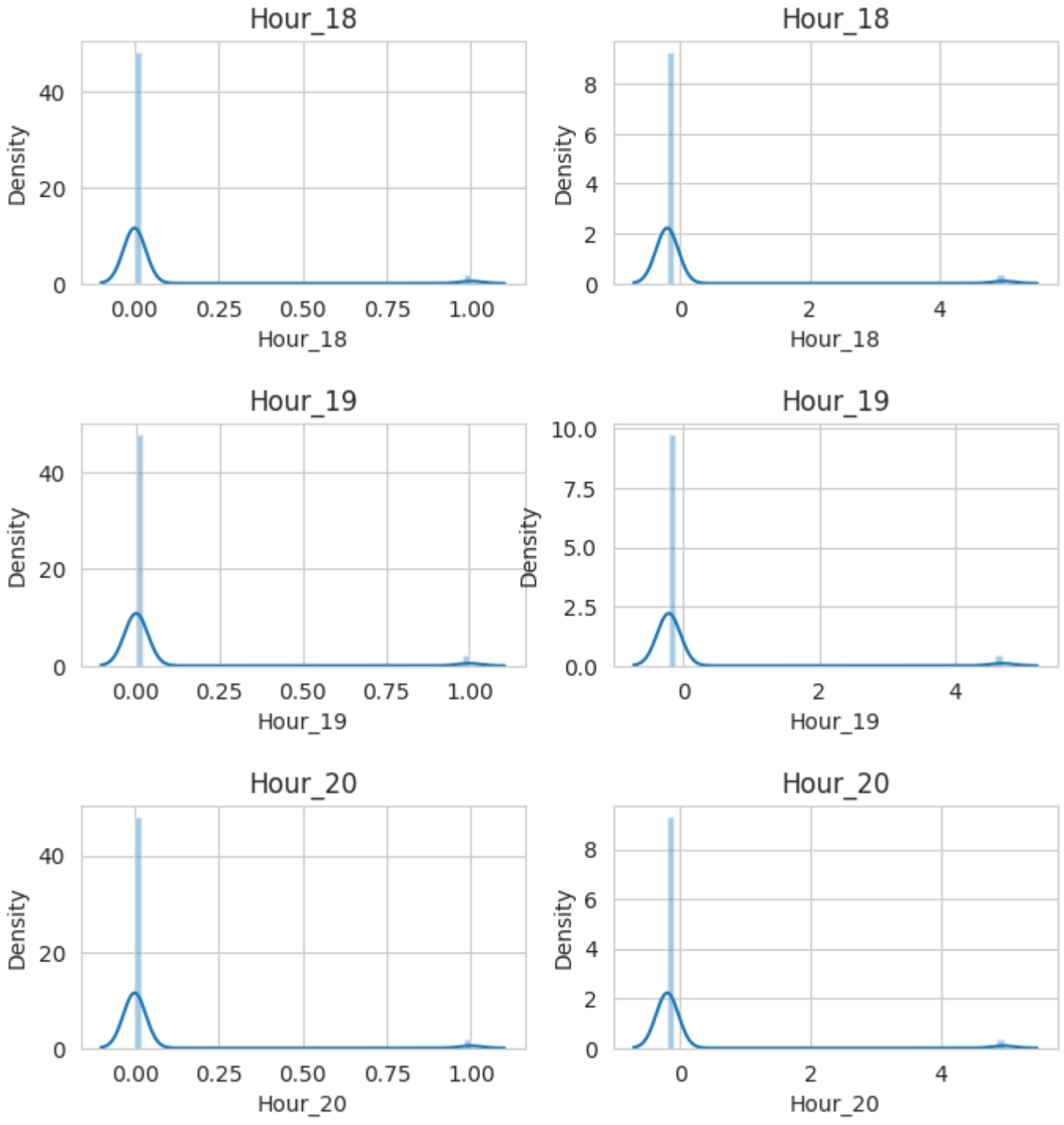

Hour_3
Density
Hour_3
Hour_3
Density
Hour_3
Hour_4
Density
Hour_4
Hour_4
Density
Hour_4
Hour_5
Density
Hour_5
Hour_5
Density
Hour_5

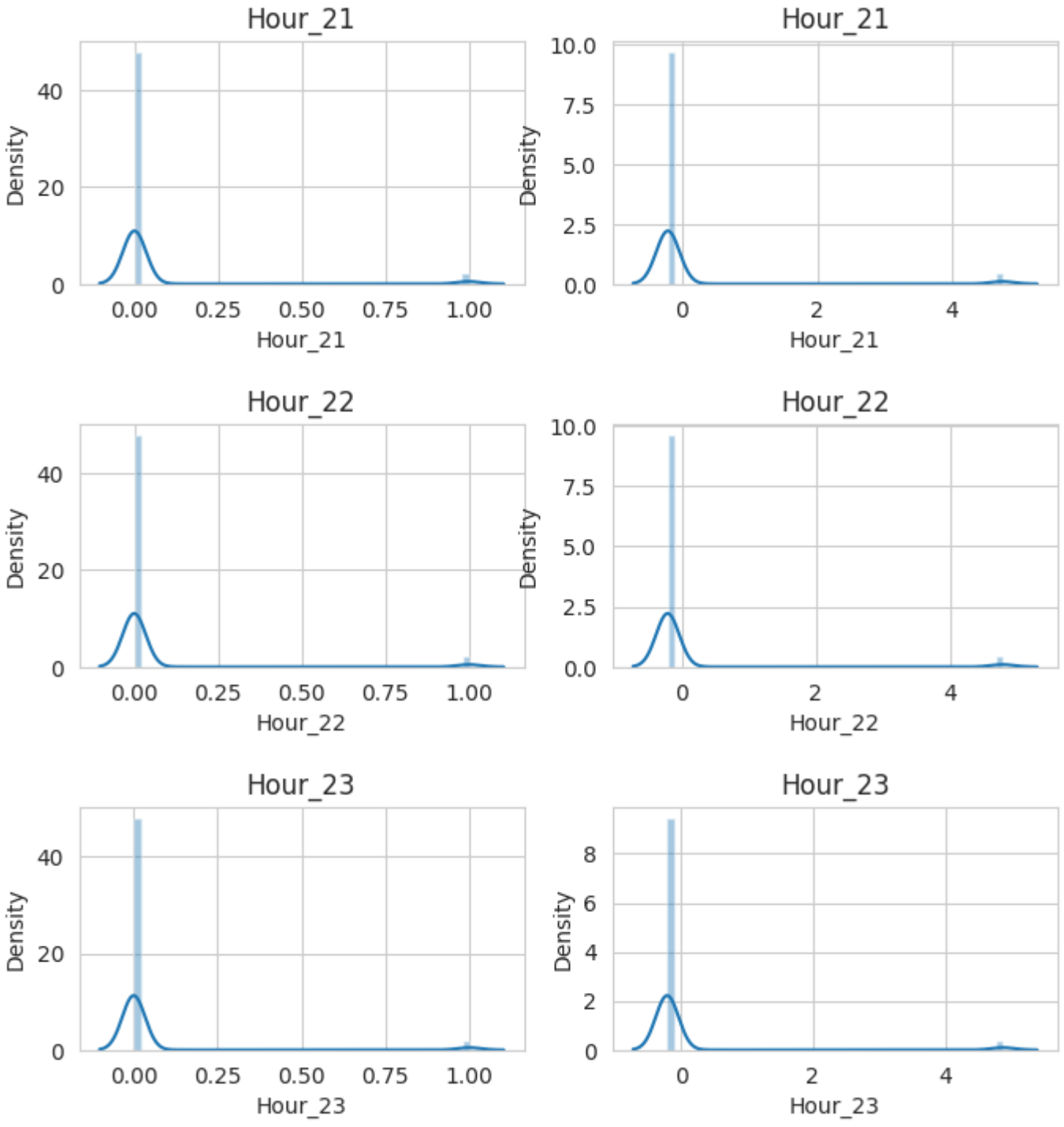

Hour_6
Density
Hour_6
Hour_6
Density
Hour_6
Hour_7
Density
Hour_7
Hour_7
Density
Hour_7
Hour_8
Density
Hour_8
Hour_8
Density
Hour_8

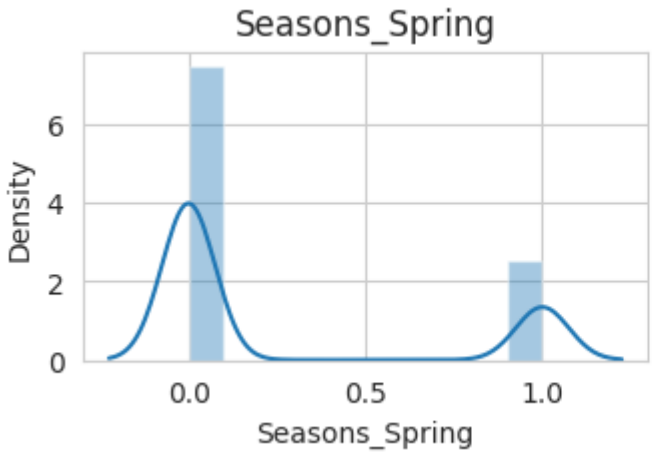

Hour_9
Density
Hour_9
Hour_9
Density
Hour_9
Hour_10
Density
Hour_10
Hour_10
Density
Hour_10
Hour_11
Density
Hour_11
Hour_11
Density
Hour_11

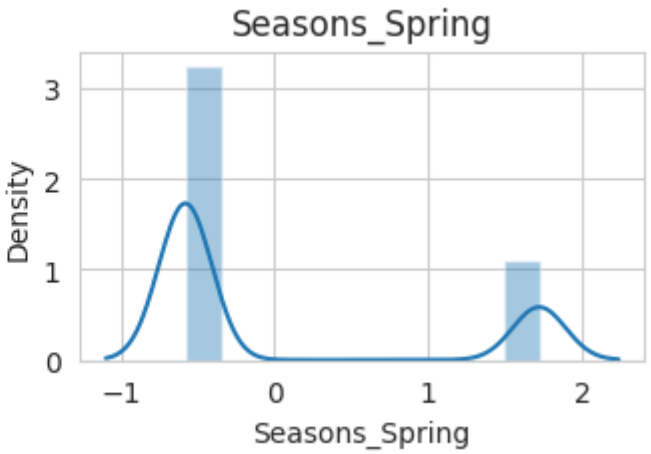

Hour_12
Density
Hour_12
Hour_12
Density
Hour_12
Hour_13
Density
Hour_13
Hour_13
Density
Hour_13
Hour_14
Density
Hour_14
Hour_14
Density
Hour_14

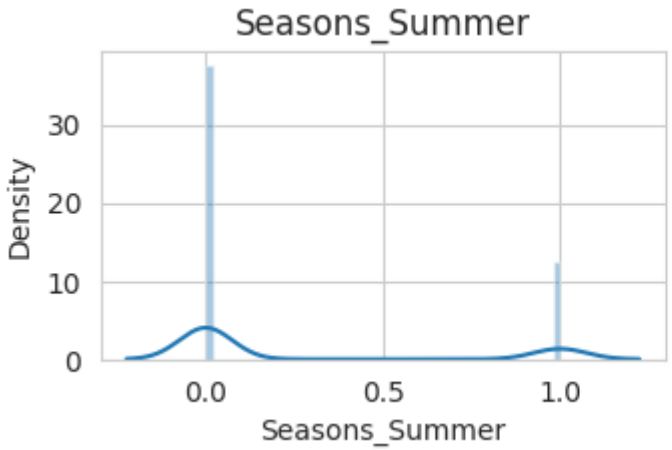

Hour_15
Density
Hour_15
Hour_15
Density
Hour_15
Hour_16
Density
Hour_16
Hour_16
Density
Hour_16
Hour_17
Density
Hour_17
Hour_17
Density
Hour_17

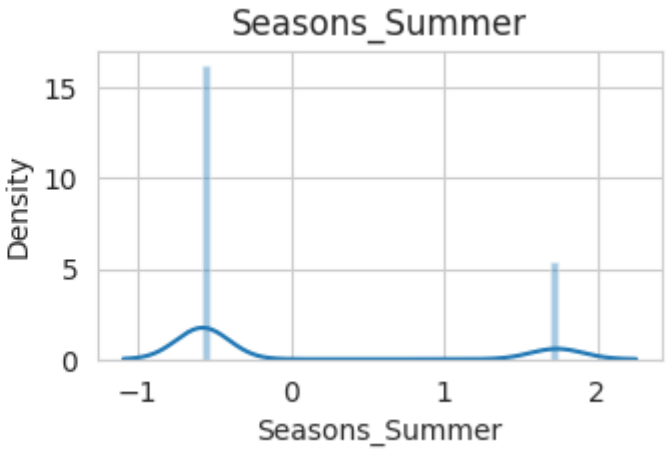

Hour_18
Density
Hour_18
Hour_18
Density
Hour_18
Hour_19
Density
Hour_19
Hour_19
Density
Hour_19
Hour_20
Density
Hour_20
Hour_20
Density
Hour_20

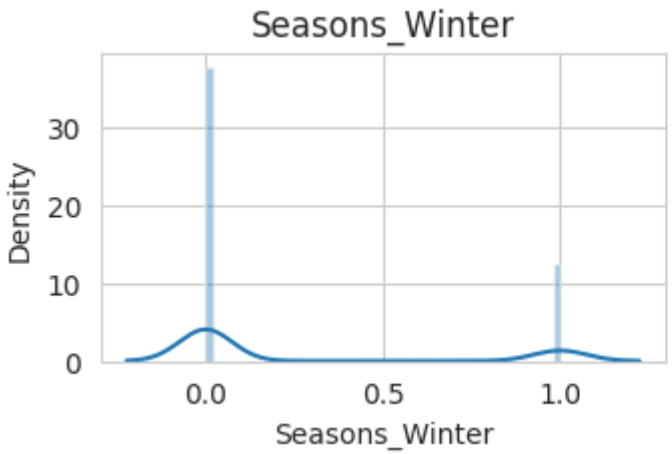

Hour_21
Density
Hour_21
Hour_21
Density
Hour_21
Hour_22
Density
Hour_22
Hour_22
Density
Hour_22
Hour_23
Density
Hour_23
Hour_23
Density
Hour_23

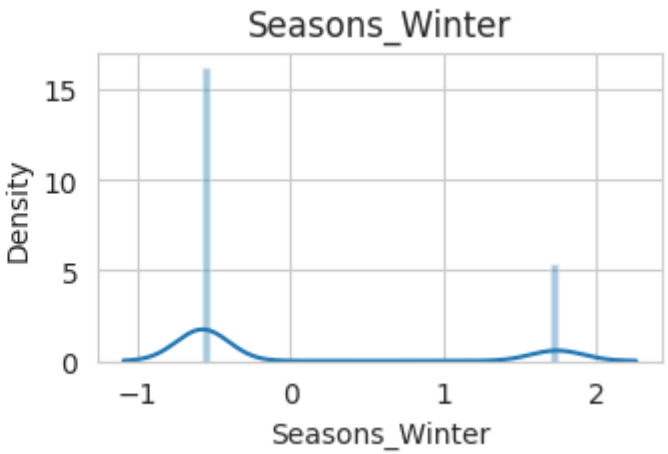

Seasons_Spring
Density
Seasons_Spring
0.0
0.5
1.0
0
2
4
6

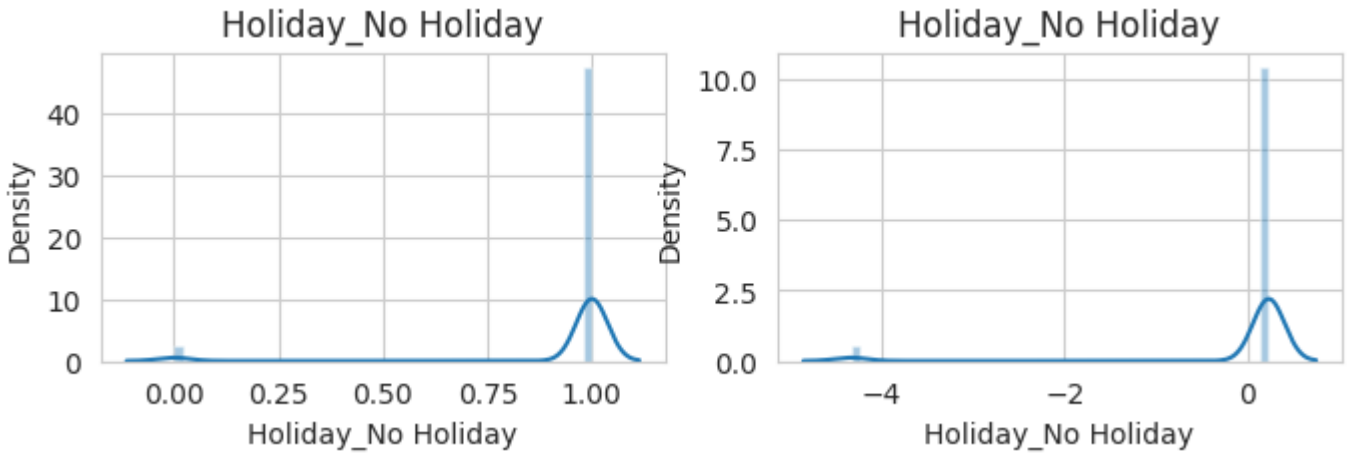

Seasons_Spring
Density
Seasons_Spring
−1
0
1
2
0
1
2
3


Seasons_Summer
Density
Seasons_Summer
0.0
0.5
1.0
0
10
20
30

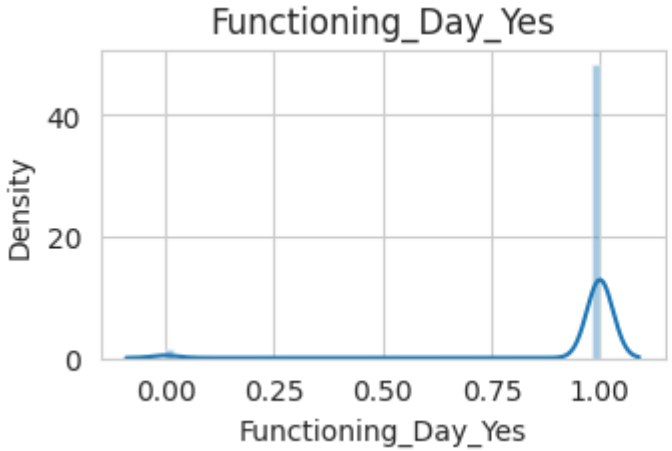

Seasons_Summer
Density
Seasons_Summer
−1
0
1
2
0
5
10
15

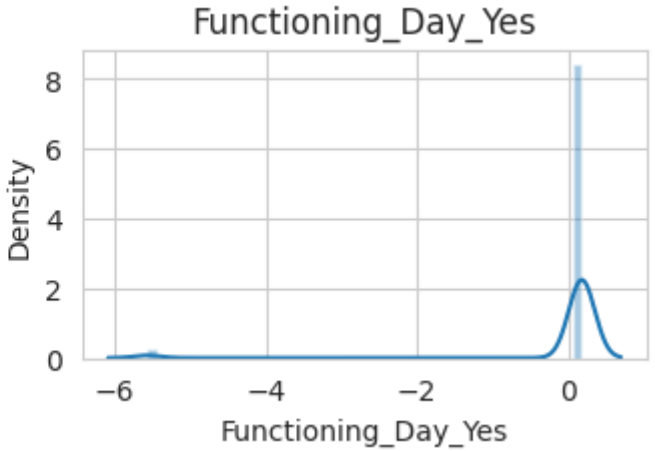

Seasons_Winter
Density
Seasons_Winter
0.0
0.5
1.0
0
10
20
30

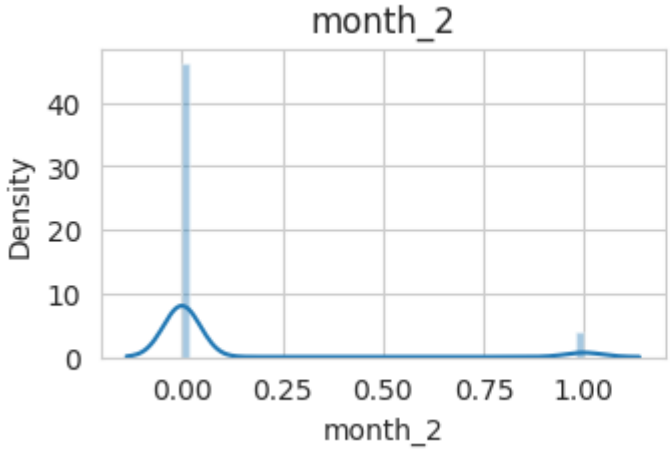

Seasons_Winter
Density
Seasons_Winter
−1
0
1
2
0
5
10
15

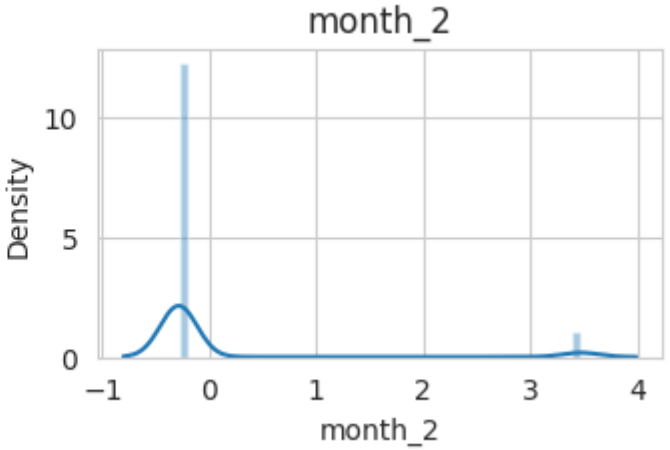

Holiday_No Holiday
Density
Holiday_No Holiday
0.00
0.25
0.50
0.75
1.00
0
10
20
30
40

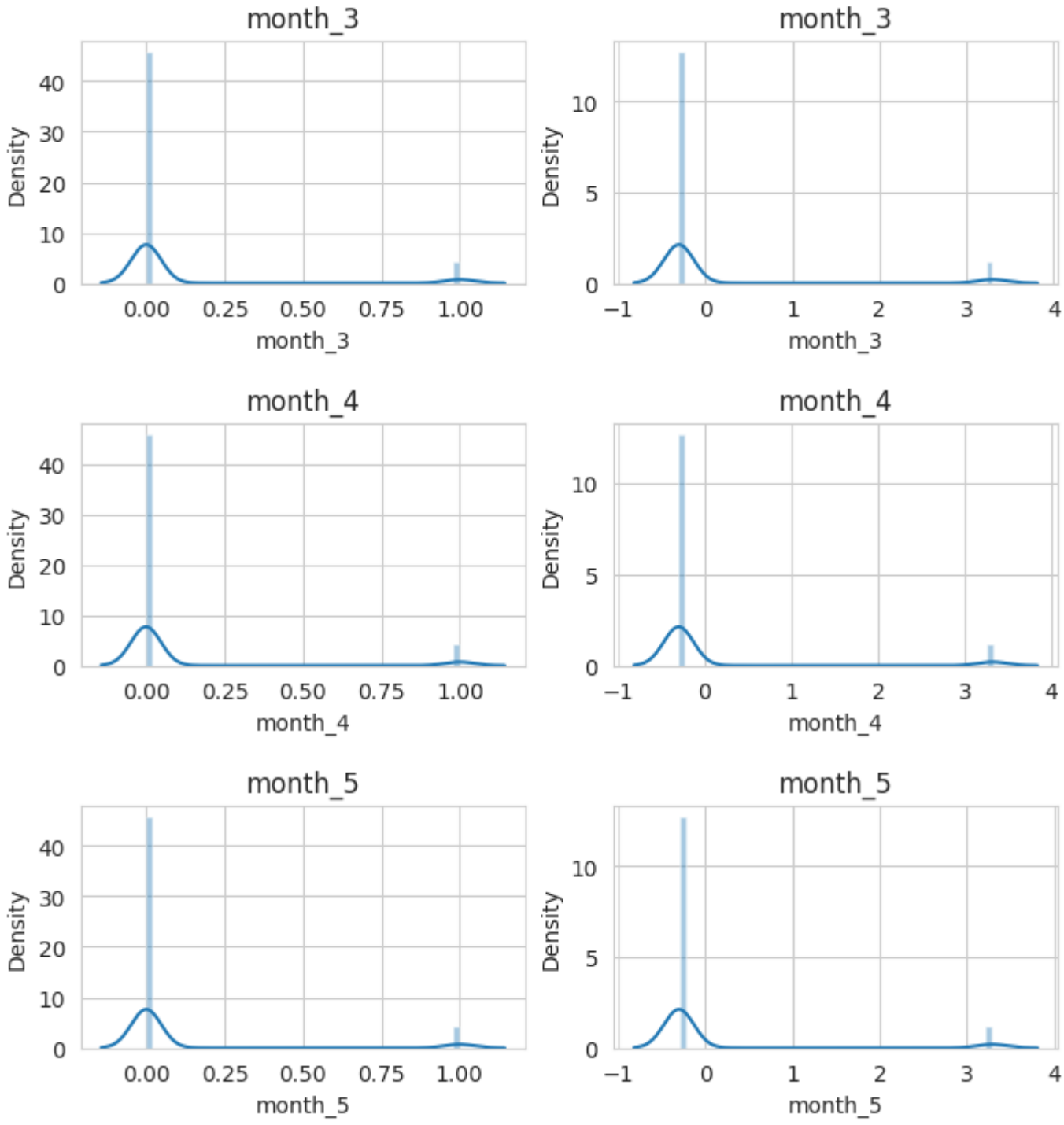
Holiday_No Holiday
Density
Holiday_No Holiday
−4
−2
0
0.0
2.5
5.0
7.5
10.0

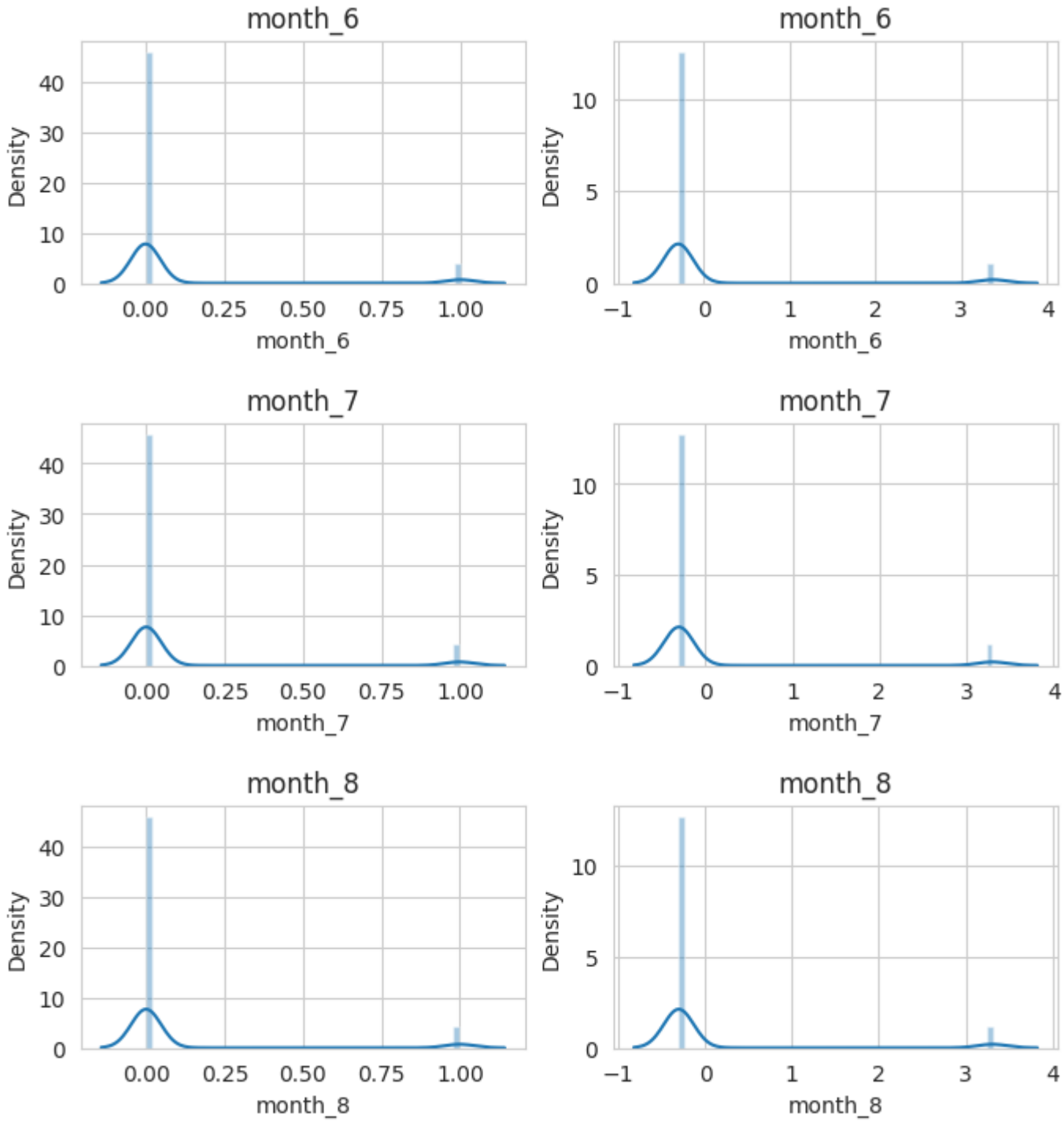

Functioning_Day_Yes
Density
Functioning_Day_Yes
0.00
0.25
0.50
0.75
1.00
0
20
40

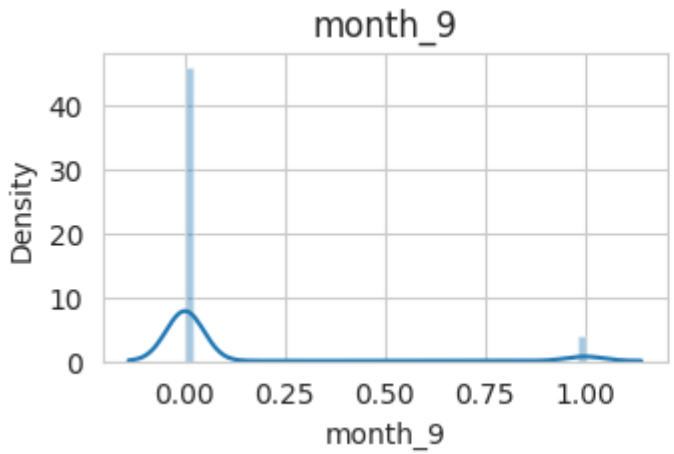

Functioning_Day_Yes
Density
Functioning_Day_Yes
−6
−4
−2
0
0
2
4
6
8

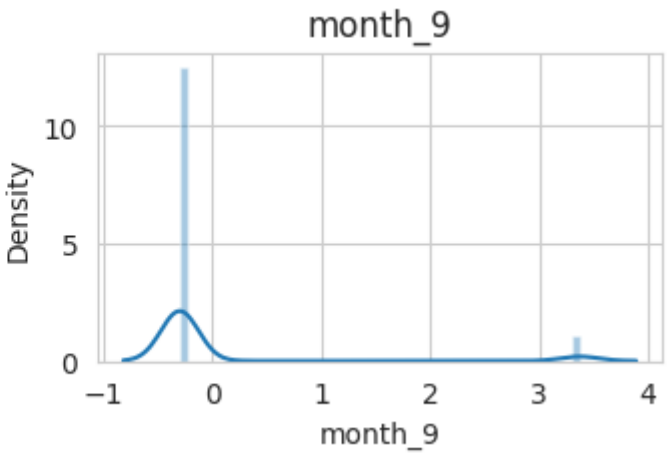

month_2
Density
month_2
0.00
0.25
0.50
0.75
1.00
0
10
20
30
40

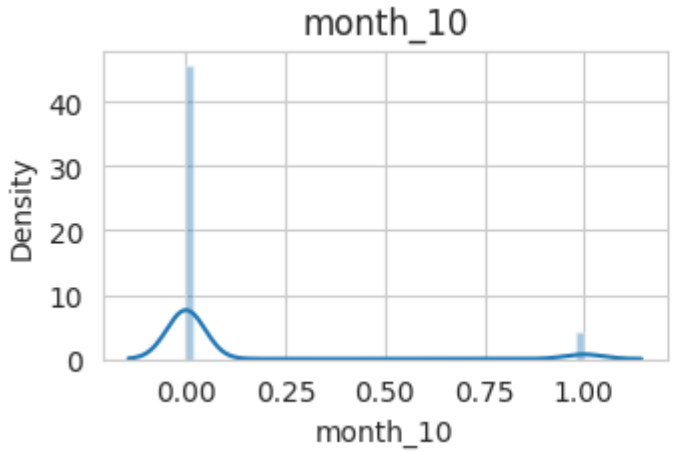

month_2
Density
month_2
−1
0
1
2
3
4
0
5
10

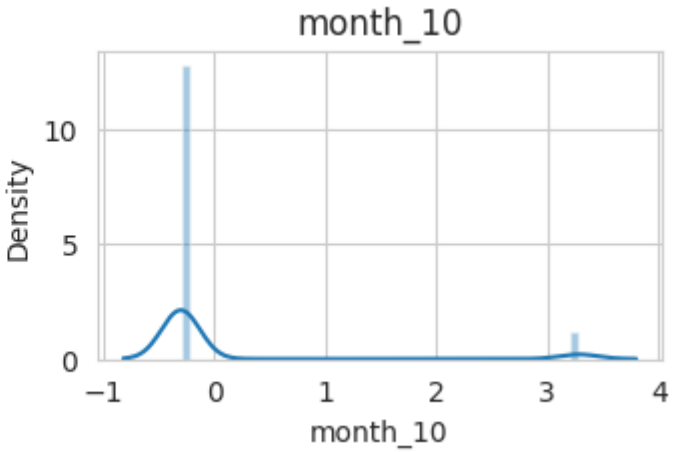

month_3
Density
month_3
month_3
Density
month_3
month_4
Density
month_4
month_4
Density
month_4
month_5
Density
month_5
month_5
Density
month_5

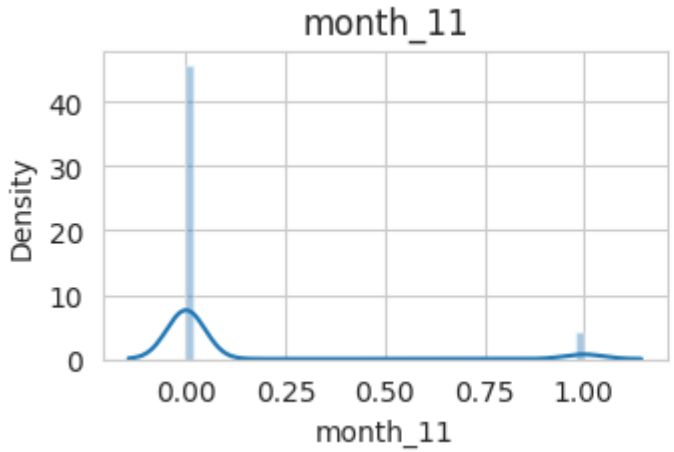

month_6
Density
month_6
month_6
Density
month_6
month_7
Density
month_7
month_7
Density
month_7
month_8
Density
month_8
month_8
Density
month_8

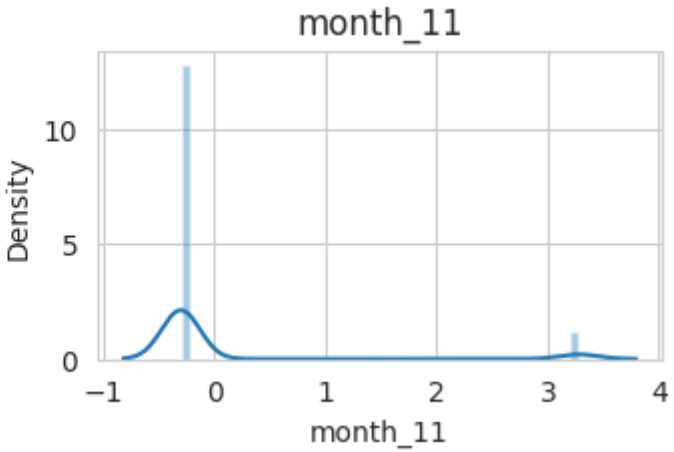

month_9
Density
0
10
20
30
40
0.00
0.25
0.50
0.75
1.00
month_9

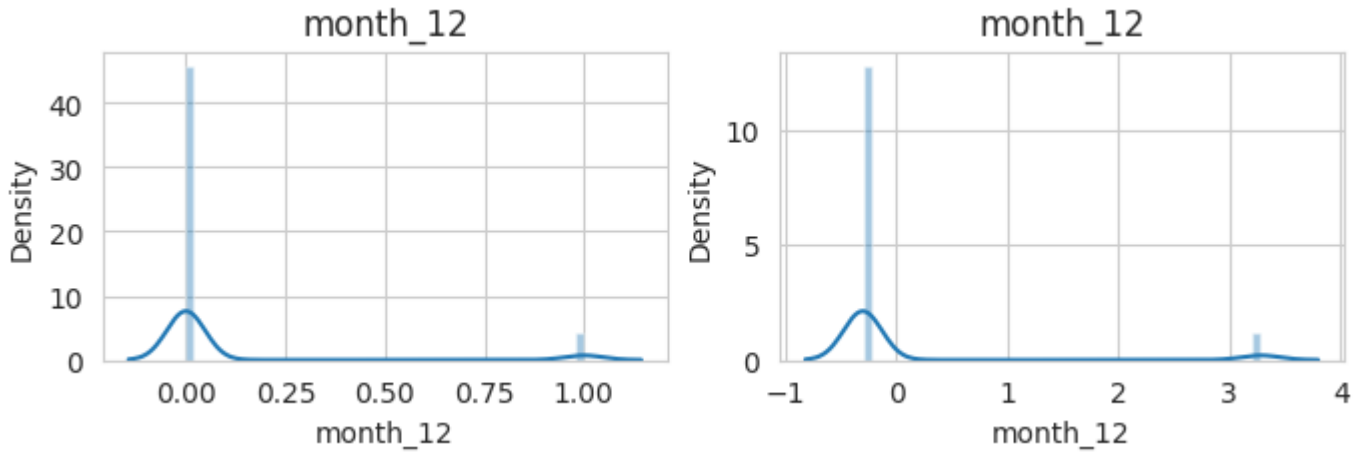

month_9
Density
0
5
10
−1
0
1
2
3
4
month_9

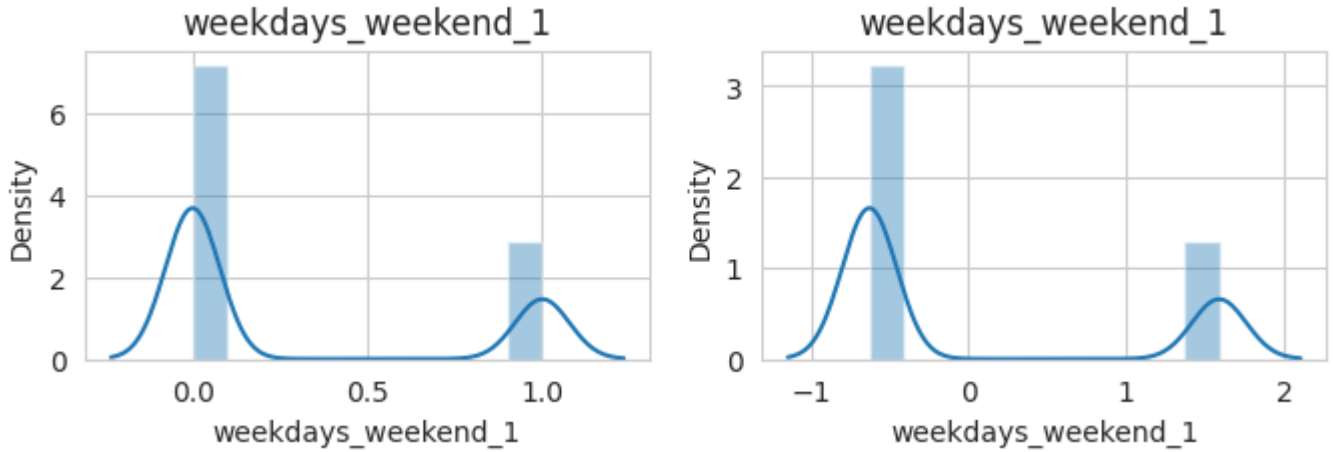

month_10
Density
0
10
20
30
40
0.00
0.25
0.50
0.75
1.00
month_10


month_10
Density
0
5
10
−1
0
1
2
3
4
month_10


month_11
Density
0
10
20
30
40
0.00
0.25
0.50
0.75
1.00
month_11


month_11
Density
0
5
10
−1
0
1
2
3
4
month_11





6. Data Scaling

In [65]:

```
# Scaling your data

scaler = StandardScaler()

# fit the scaler to the train set, it will learn the parameters
scaler.fit(X_train)

# transform train and test sets
X_train_scaled = scaler.transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

```
# Instantiate a LinearRegression model
lr = LinearRegression()

# Fit the model using the scaled features (X_train_scaled) and the continuous labels (y_train)
lr.fit(X_train_scaled, y_train)

# Predict on the test set
y_pred = lr.predict(X_test_scaled)

# Calculate the mean squared error (MSE) on the test set
mse = mean_squared_error(y_test, y_pred)
print("Mean Squared Error:", mse)

# Assuming y_test contains the true target values and y_pred contains the predicted values
r2 = r2_score(y_test, y_pred)
print("R-squared score:", r2)

# Assuming X_train_scaled and y_train contain the scaled training data and target values, respectively
# Assuming lr is your trained linear regression model

# Perform cross-validation with 5 folds
cv_scores = cross_val_score(lr, X_train_scaled, y_train, cv=5, scoring='r2')

# Print the cross-validated R-squared scores
print("Cross-validated R-squared scores:", cv_scores)
print("Mean R-squared score:", cv_scores.mean())
```

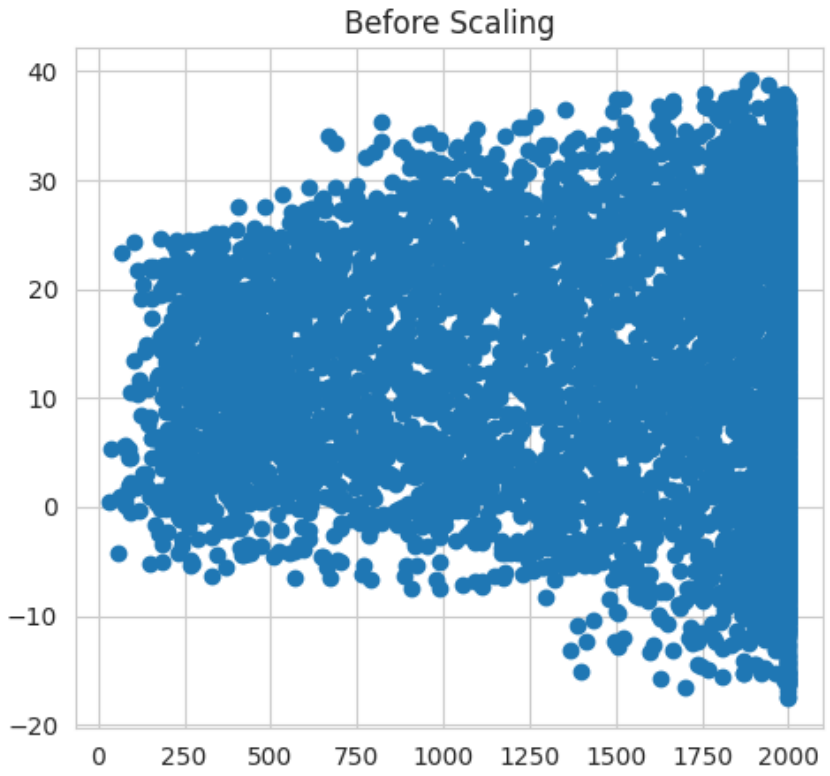
```
Mean Squared Error: 0.3583458611302685
R-squared score: 0.8506163614656173
Cross-validated R-squared scores: [0.81826859 0.82247097 0.82639171 0.81178648 0.82022059]
Mean R-squared score: 0.8198276671809047
```

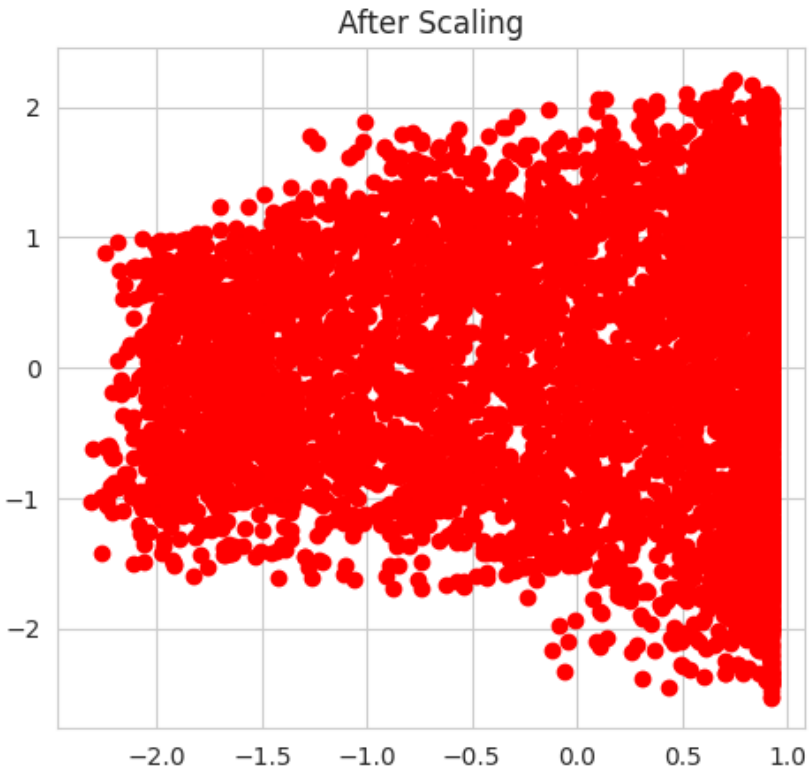
In [66]:

```
X_train_scaled = pd.DataFrame(X_train_scaled, columns=X_train.columns)
X_test_scaled = pd.DataFrame(X_test_scaled, columns=X_test.columns)
```

In [67]:

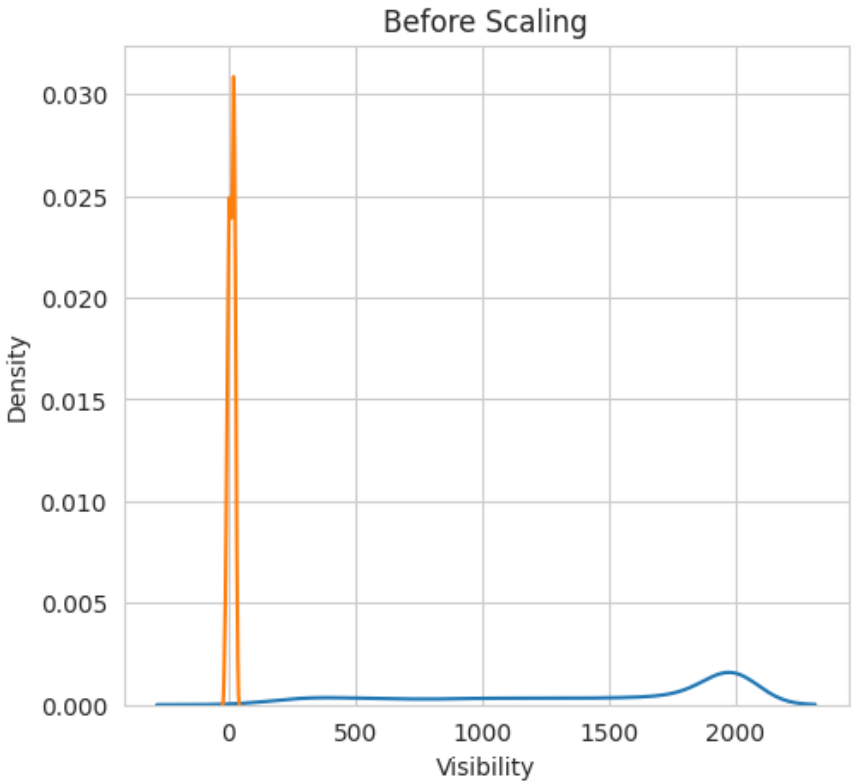
```
fig, (ax1, ax2) = plt.subplots(ncols=2, figsize=(12, 5))

ax1.scatter(X_train['Visibility'], X_train['Temperature'])
ax1.set_title("Before Scaling")
ax2.scatter(X_train_scaled['Visibility'], X_train_scaled['Temperature'],color='red')
ax2.set_title("After Scaling")
plt.show()
```

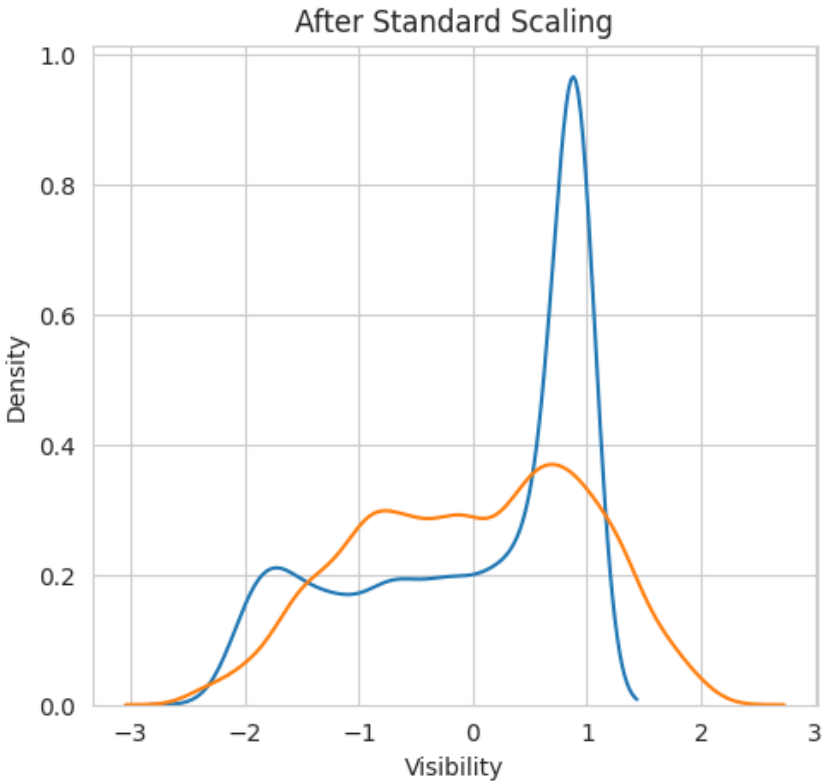




In [68]:

```
fig, (ax1, ax2) = plt.subplots(ncols=2, figsize=(12, 5))

# before scaling
ax1.set_title('Before Scaling')
sns.kdeplot(X_train['Visibility'], ax=ax1)
sns.kdeplot(X_train['Temperature'], ax=ax1)

# after scaling
ax2.set_title('After Standard Scaling')
sns.kdeplot(X_train_scaled['Visibility'], ax=ax2)
sns.kdeplot(X_train_scaled['Temperature'], ax=ax2)
plt.show()
```

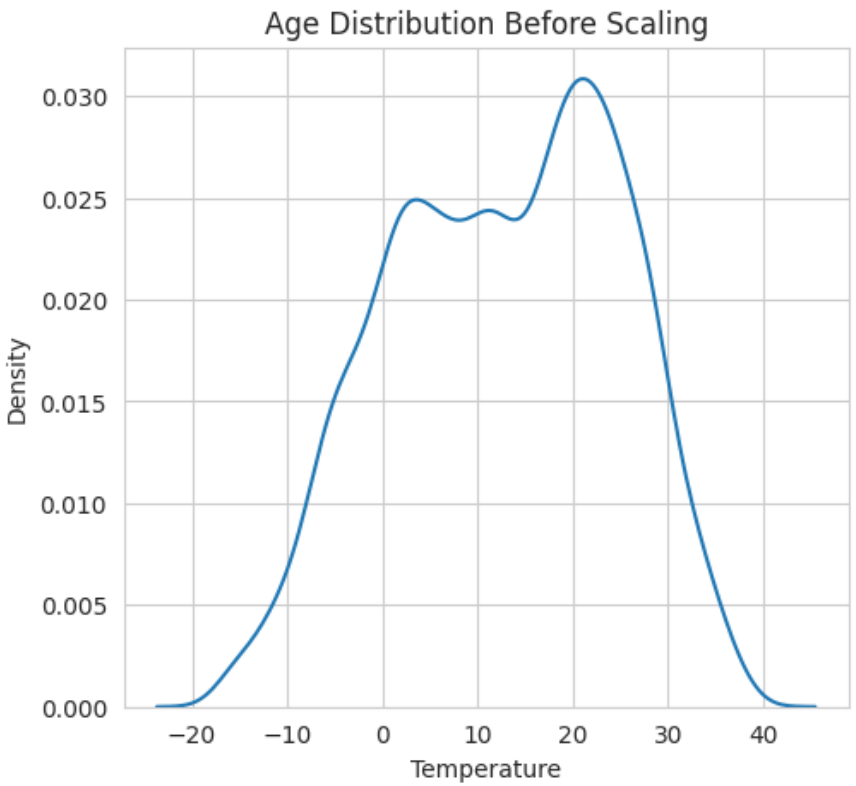


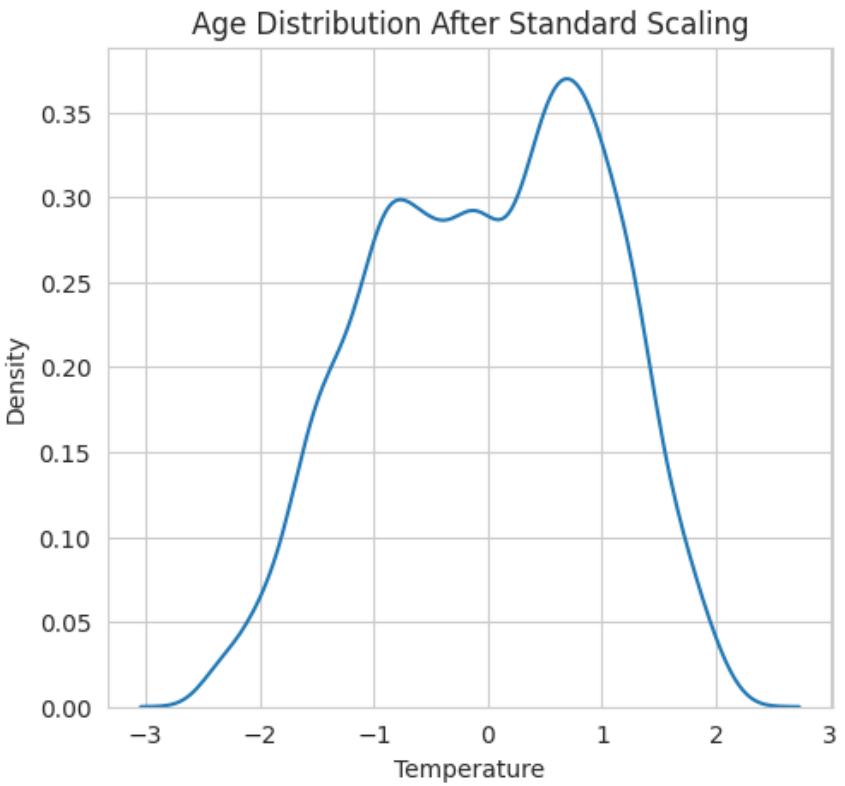


Comparison of Distributions

In [69]:

```
fig, (ax1, ax2) = plt.subplots(ncols=2, figsize=(12, 5))

# before scaling
ax1.set_title('Age Distribution Before Scaling')
sns.kdeplot(X_train['Temperature'], ax=ax1)

# after scaling
ax2.set_title('Age Distribution After Standard Scaling')
sns.kdeplot(X_train_scaled['Temperature'], ax=ax2)
plt.show()
```





In [70]:

```
MinMaxScaler_scaler = MinMaxScaler()

# fit the scaler to the train set, it will learn the parameters
MinMaxScaler_scaler.fit(X_train)

# transform train and test sets
X_train_scaled_MinMaxScaler = MinMaxScaler_scaler.transform(X_train)
X_test_scaled_MinMaxScaler = MinMaxScaler_scaler.transform(X_test)


# Create an instance of the LinearRegression model
lr = LinearRegression()

# Fit the model on the scaled training data
lr.fit(X_train_scaled_MinMaxScaler, y_train)

# Predict the target variable for the scaled test data
y_pred_MinMaxScaler = lr.predict(X_test_scaled_MinMaxScaler)
```

```
# Calculate the mean squared error (MSE)
mse = mean_squared_error(y_test, y_pred_MinMaxScaler)

print("Mean Squared Error:", mse)

# Calculate the R-squared score
r2 = r2_score(y_test, y_pred_MinMaxScaler)
print("R-squared score:", r2)

# Perform cross-validation with 5 folds
cv_scores_MinMaxScaler = cross_val_score(lr, X_train_scaled_MinMaxScaler, y_train, cv=5, scoring='r2')
mean_r2_MinMaxScaler = np.mean(cv_scores_MinMaxScaler)
print("Mean cross-validated R-squared score:", mean_r2_MinMaxScaler)
print("Mean R-squared score:", cv_scores_MinMaxScaler.mean())
```

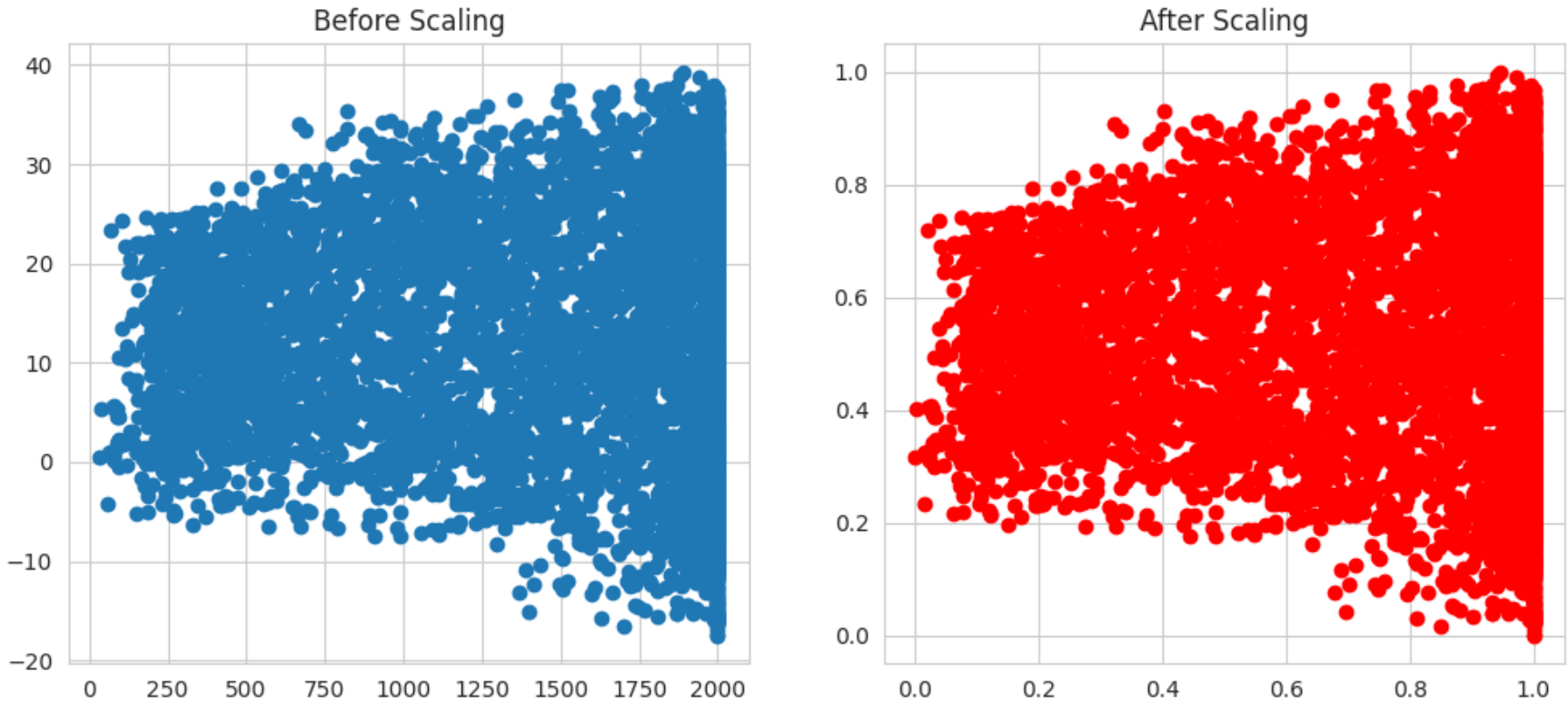
```
Mean Squared Error: 0.3583480317104422
R-squared score: 0.8506154566158639
Mean cross-validated R-squared score: 0.8198283881241284
Mean R-squared score: 0.8198283881241284
```

In [71]:

```
X_train_scaled_MinMaxScaler = pd.DataFrame(X_train_scaled_MinMaxScaler, columns=X_train.columns)
X_test_scaled_MinMaxScaler = pd.DataFrame(X_test_scaled_MinMaxScaler, columns=X_test.columns)
```

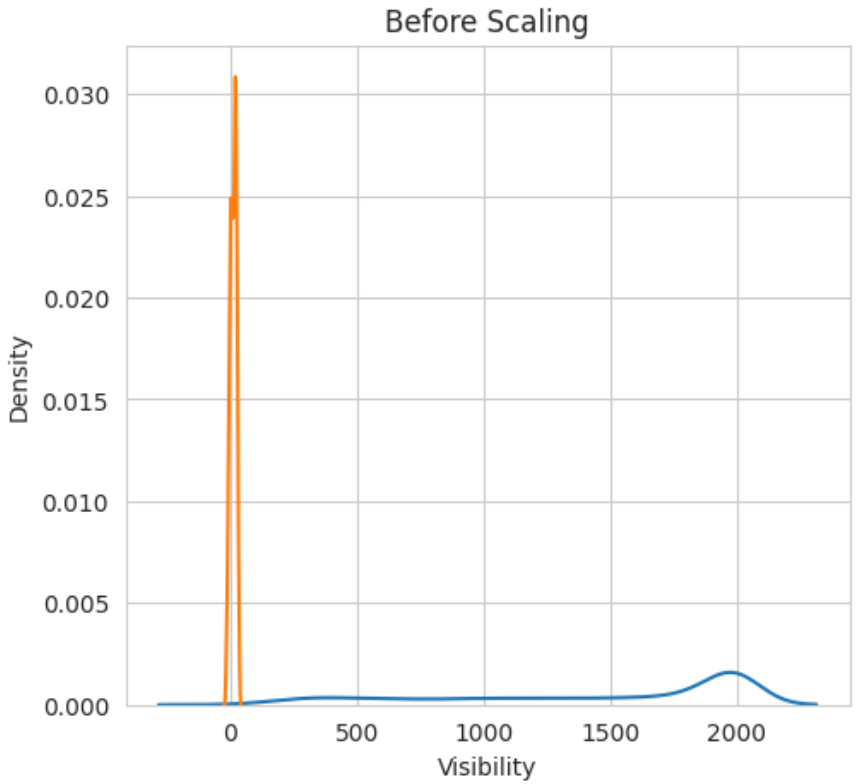
In [72]:

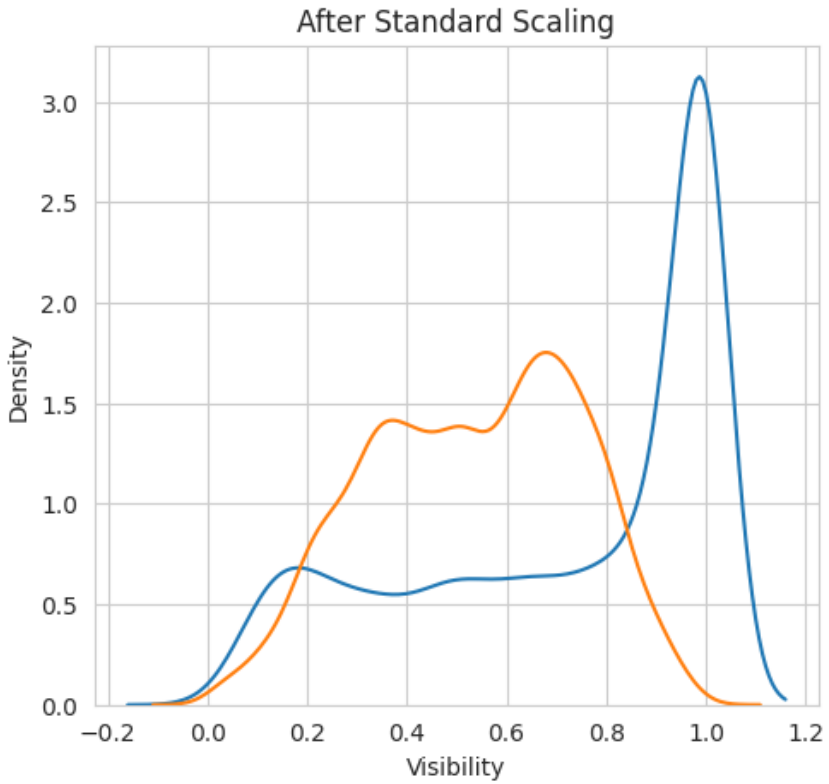
```
fig, (ax1, ax2) = plt.subplots(ncols=2, figsize=(12, 5))

ax1.scatter(X_train['Visibility'], X_train['Temperature'])
ax1.set_title("Before Scaling")
ax2.scatter(X_train_scaled_MinMaxScaler['Visibility'], X_train_scaled_MinMaxScaler['Temperature'],color='red')
ax2.set_title("After Scaling")
plt.show()
```



In [73]:

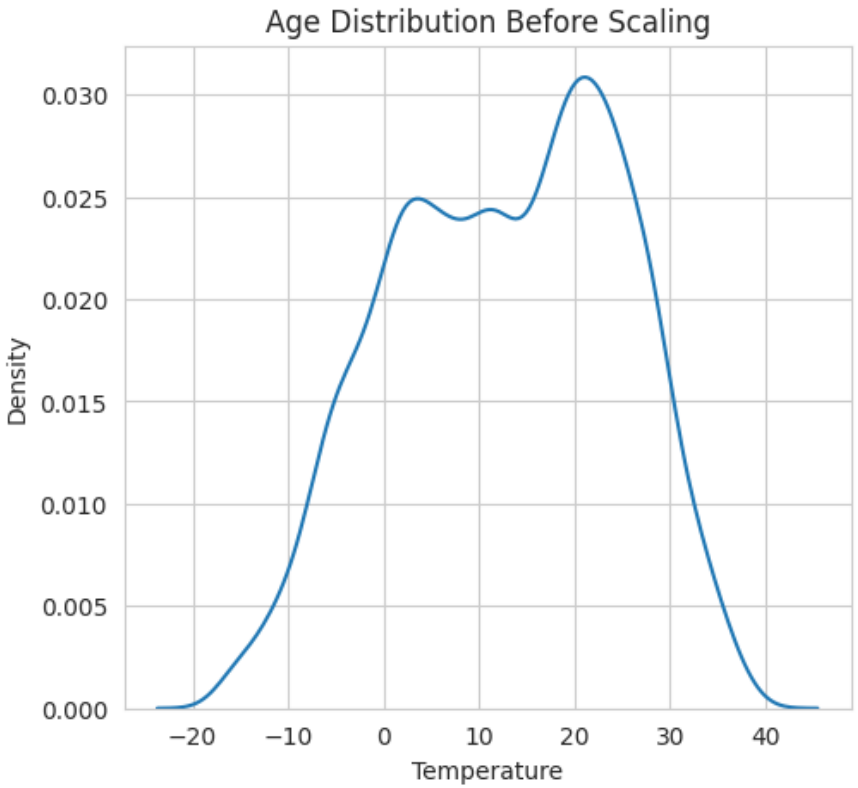
```
fig, (ax1, ax2) = plt.subplots(ncols=2, figsize=(12, 5))

# before scaling
ax1.set_title('Before Scaling')
sns.kdeplot(X_train['Visibility'], ax=ax1)
sns.kdeplot(X_train['Temperature'], ax=ax1)

# after scaling
ax2.set_title('After Standard Scaling')
sns.kdeplot(X_train_scaled_MinMaxScaler['Visibility'], ax=ax2)
sns.kdeplot(X_train_scaled_MinMaxScaler['Temperature'], ax=ax2)
plt.show()
```

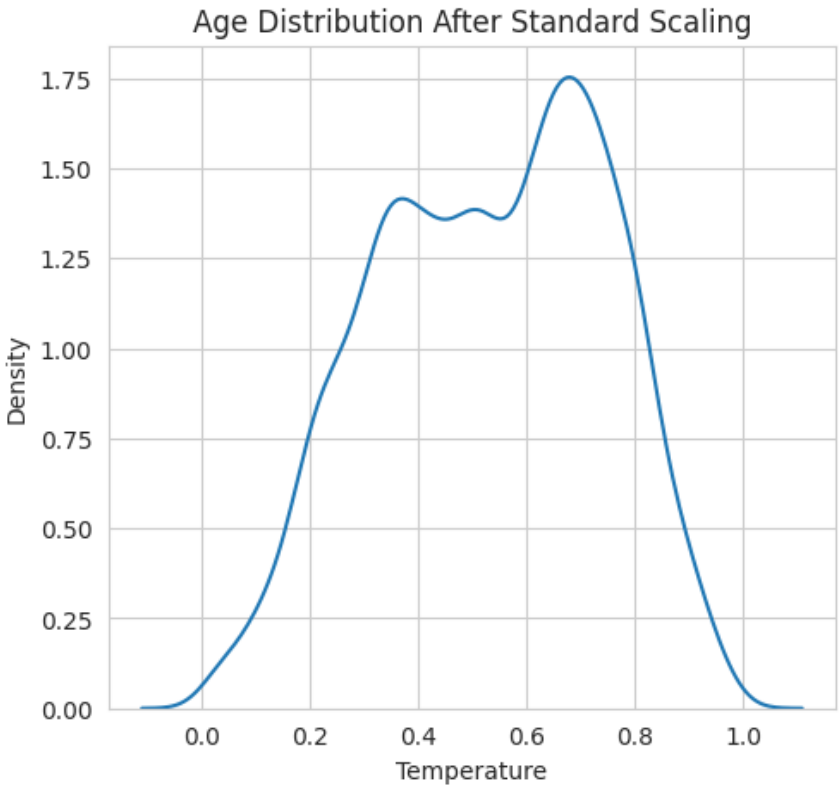




In [74]:

```
fig, (ax1, ax2) = plt.subplots(ncols=2, figsize=(12, 5))

# before scaling
ax1.set_title('Age Distribution Before Scaling')
sns.kdeplot(X_train['Temperature'], ax=ax1)

# after scaling
ax2.set_title('Age Distribution After Standard Scaling')
sns.kdeplot(X_train_scaled_MinMaxScaler['Temperature'], ax=ax2)
plt.show()
```





Why scaling is important?

Scaling is important in machine learning to:

- Normalize the features and bring them to a comparable range.
- Improve algorithm convergence and stability.
- Prevent numerical instability and biased results.
- Ensure compatibility with certain algorithms.
- Enable fair and accurate feature comparisons.

Which method have you used to scale you data and why?

I use standarization method just buecause this parform slighty better then normalization mathod.Standardization transforms data to have zero mean and unit variance, while normalization scales data to a specific range, often between 0 and 1.

7. Dimesionality Reduction

The "Curse of Dimensionality" refers to the challenges that arise when dealing with high-dimensional data. It causes sparsity of data, increased computational complexity, overfitting, and difficulties with distance-based algorithms. Mitigation strategies include feature selection, dimensionality reduction, regularization, and leveraging domain knowledge.

Do you think that dimensionality reduction is needed? Explain Why?

dimensionality reduction techniques help in simplifying complex datasets, improving computational efficiency, enhancing model performance, facilitating visualization and interpretation, and aiding in feature selection. this dataset is not that complex.

Apply PCA to the training set:

In [75]:
```
for i in range(1,47):
  pca = PCA(n_components=i)  # Specify the number of components to keep
  X_train_pca = pca.fit_transform(X_train)
#Fit a linear regression model on the transformed training set:
  lrp = LinearRegression()
  lrp.fit(X_train_pca, y_train)
# Transform the test set using the same PCA transformation:
  X_test_pca = pca.transform(X_test)
# Make predictions on the transformed test set:
  y_pred = lrp.predict(X_test_pca)
# Evaluate the performance of the model using the R-squared score:
  r2 = r2_score(y_test, y_pred)
  print("R-squared score:", r2)
```

```
R-squared score: 0.0328852031430894
R-squared score: 0.03813258496067462
R-squared score: 0.2788096781654805
R-squared score: 0.29502206786333507
R-squared score: 0.29494266432264216
R-squared score: 0.2987526572794399
R-squared score: 0.29862519742433824
R-squared score: 0.2993467838722643
R-squared score: 0.29945745529118895
R-squared score: 0.3078947758664733
R-squared score: 0.31320138046545776
R-squared score: 0.3137833845839023
R-squared score: 0.31917011622736413
R-squared score: 0.32519245413629905
R-squared score: 0.3263560235076036
R-squared score: 0.3263715665610927
R-squared score: 0.3292619678179781
R-squared score: 0.3314603790881715
```

```
R-squared score: 0.3384519017718002
R-squared score: 0.335387358077574
R-squared score: 0.3365193097522461
R-squared score: 0.3406390252195153
R-squared score: 0.3431480363205476
R-squared score: 0.3422921281192236
R-squared score: 0.3449563618822905
R-squared score: 0.3403322904030839
R-squared score: 0.35501600194201155
R-squared score: 0.3674747137040827
R-squared score: 0.348713199218473
R-squared score: 0.3469787497713791
R-squared score: 0.37400423830807283
R-squared score: 0.37344702024509746
R-squared score: 0.3741139974693597
R-squared score: 0.37797616282903523
R-squared score: 0.39631351995967057
R-squared score: 0.39665241216663083
R-squared score: 0.4063691786743845
R-squared score: 0.40586179164629
R-squared score: 0.44358290335383044
R-squared score: 0.8108874122406386
R-squared score: 0.8490843930665377
R-squared score: 0.850307690232463
R-squared score: 0.8503901949029601
R-squared score: 0.85061481823247
R-squared score: 0.850589467429501
R-squared score: 0.8505894658403804
```

In [76]:

```
pca = PCA(n_components=45)  # Specify the number of components to keep
X_train_pca = pca.fit_transform(X_train)
#Fit a linear regression model on the transformed training set:
lrp = LinearRegression()
lrp.fit(X_train_pca, y_train)
# Transform the test set using the same PCA transformation:
X_test_pca = pca.transform(X_test)
# Make predictions on the transformed test set:
y_pred = lrp.predict(X_test_pca)
# Evaluate the performance of the model using the R-squared score:
r2 = r2_score(y_test, y_pred)
print("R-squared score:", r2)
```

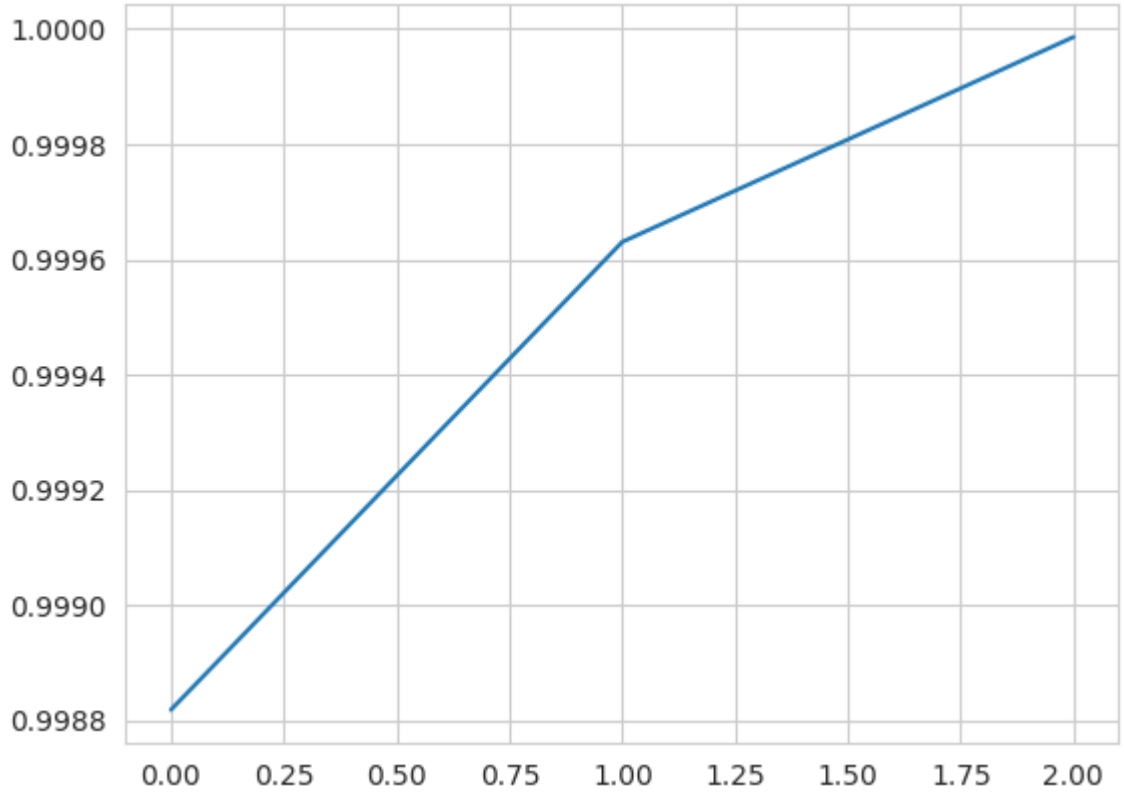
```
R-squared score: 0.850589467429501
```

In [77]:
```
# transforming in 3D
pca = PCA(n_components=3)
X_train_pca = pca.fit_transform(X_train)
X_test_pca = pca.transform(X_test)
```

In [78]:
```
import plotly.express as px
y_train_pca = y_train.astype(str)
fig = px.scatter_3d(df, x=X_train_pca[:,0], y=X_train_pca[:,1], z=X_train_pca[:,2],
              color=y_train_pca)
fig.update_layout(
    margin=dict(l=20, r=20, t=20, b=20),
    paper_bgcolor="LightSteelBlue",
)
fig.show()
```

In [79]:
```
plt.plot(np.cumsum(pca.explained_variance_ratio_))
```

Out[79]:
```
[<matplotlib.lines.Line2D at 0x7fbb2ce081f0>]
```



Which dimensionality reduction technique have you used and why? (If dimensionality reduction done on dataset.)

PCA is appled on dataset. number of columns are not very high. so PCA dont help much.

Dimensionality reduction techniques are used to reduce the number of features or variables in a dataset while preserving the essential information. These techniques are beneficial when working with high-dimensional data, as they can help in simplifying the data representation, reducing computational complexity, and removing noise or redundant information.

Principal Component Analysis (PCA): PCA is a popular linear technique that transforms the original variables into a new set of uncorrelated variables called principal components. These components are ordered by the amount of variance they explain in the data, allowing for dimensionality reduction by selecting a subset of the components that capture most of the variability.

8. Data Splitting

In [118…

```
X = bike_df_copy.drop(columns=['Rented_Bike_Count'], axis=1)
y = np.sqrt(bike_df_copy['Rented_Bike_Count'])
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.25, random_state=0)
```

What data splitting ratio have you used and why?

A data splitting ratio of 75% for training and 25% for testing was used. This ratio is commonly used in machine learning and is known as the 75/25 split.

The reason for choosing this ratio is to strike a balance between having enough data for training the model and having enough data for evaluating its performance. By allocating 75% of the data for training, the model has a substantial amount of data to learn from and generalize patterns. The remaining 25% is used for testing, allowing us to assess how well the model performs on unseen data and evaluate its generalization capabilities.

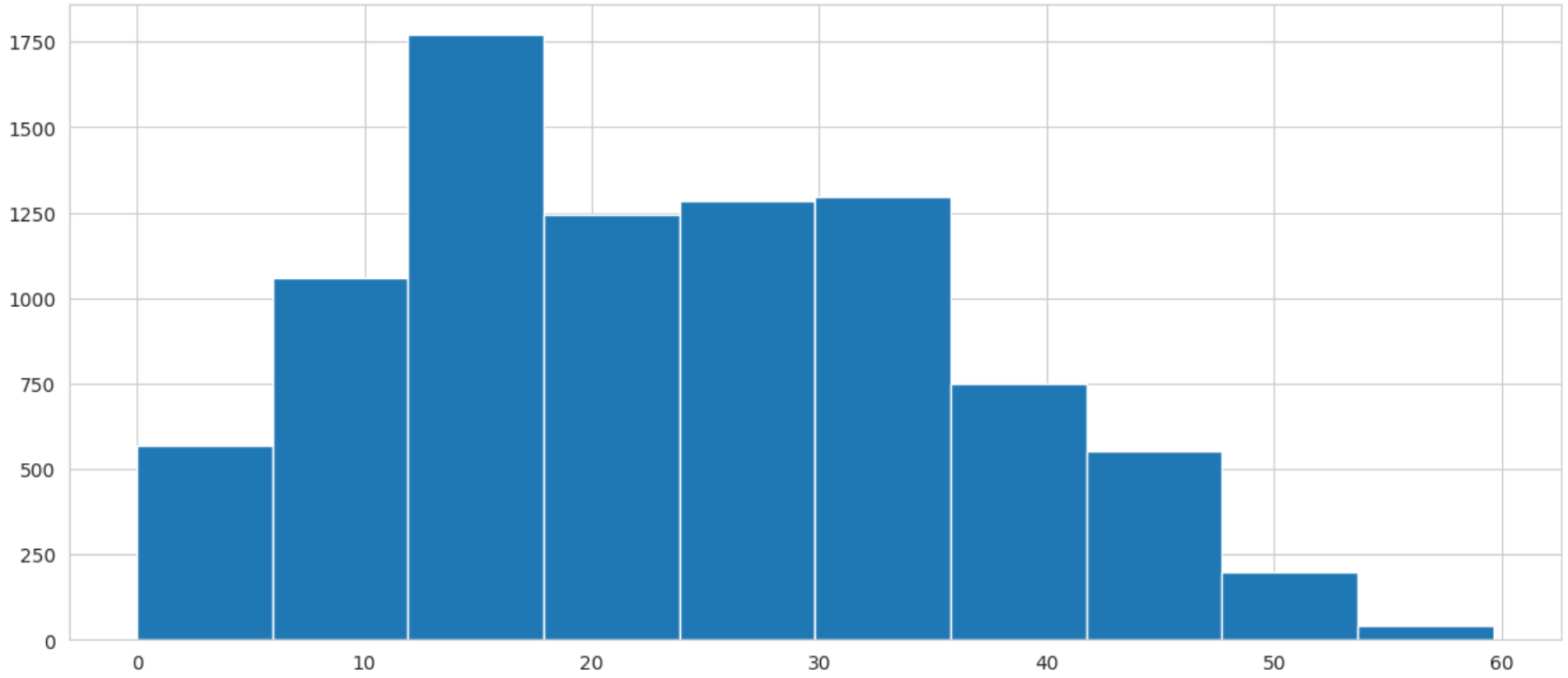
9. Handling Imbalanced Dataset

Do you think the dataset is imbalanced? Explain Why.

'Rented_Bike_Count' column is left skewed in dataset, but in privous outlier removing process i apply np.squr and is column convarted into normal distribution.. so now this dataset is balanced.

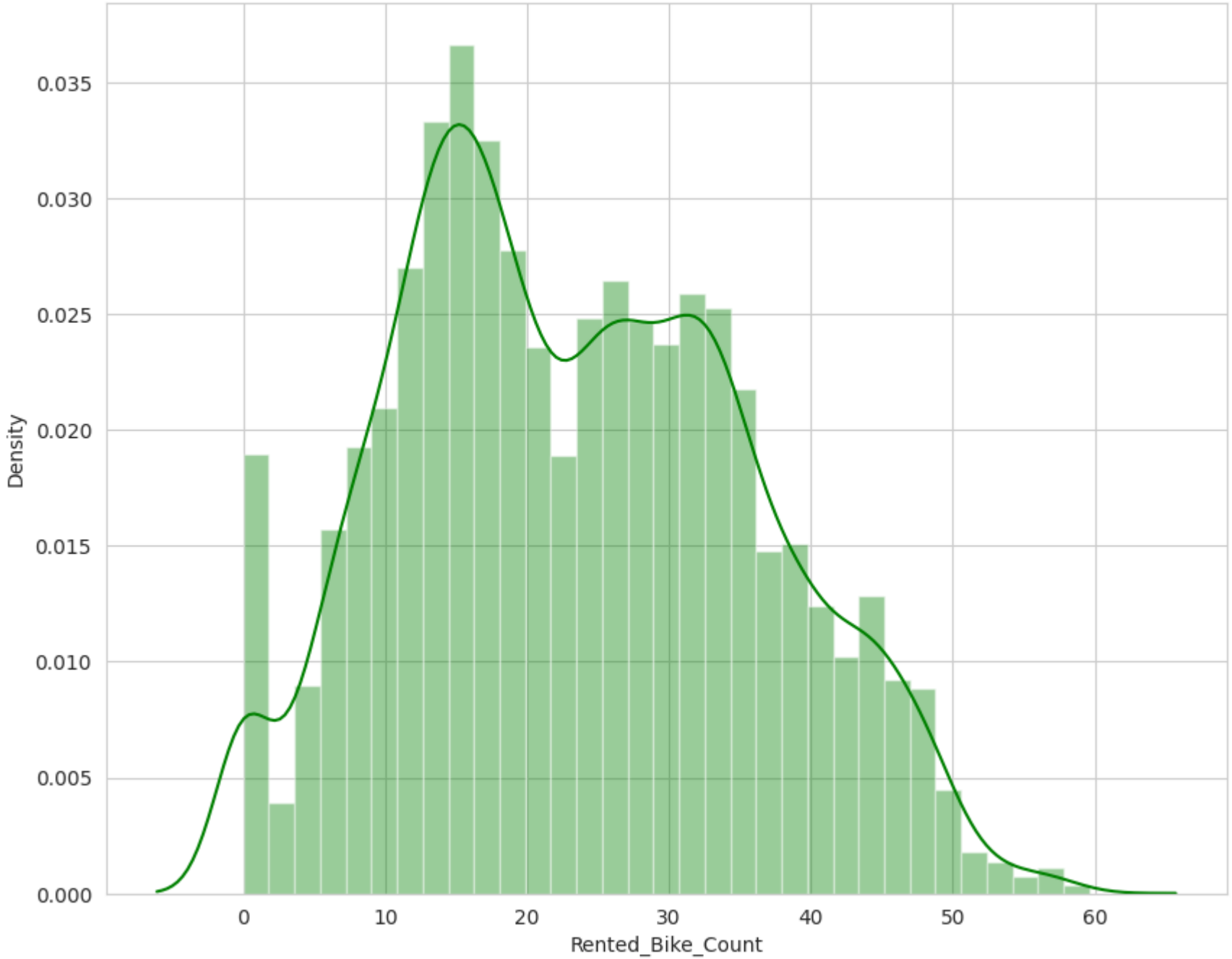
In [81]:

```
# Handling Imbalanced Dataset (If needed)
#Cheking Histogram
plt.figure(figsize=(14, 6))
plt.hist(bike_df['Rented_Bike_Count'])
plt.show()
```



In [82]:

```
plt.figure(figsize=(10, 8))
plt.xlabel('Rented Bike Count')
plt.ylabel('Density')
ax = sns.distplot(bike_df['Rented_Bike_Count'], color="green")
plt.show()
```



What technique did you use to handle the imbalance dataset and why? (If needed to be balanced)

np.sqrt(bike_df['Rented_Bike_Count']) will return a new Series object with the square root of each value in the 'Rented_Bike_Count' column.

This transformation is often applied to data to reduce skewness, as taking the square root can help normalize the distribution and make it more symmetric.

7. ML Model Implementation

ML Model - 1

Linear Regression

In [119…
```
# ML Model - 1 Implementation
# Fit the Algorithm
reg= LinearRegression()
reg.fit(X_train, y_train)
```

Out[119…
```
▾ LinearRegression

LinearRegression()
```

In [120…
```
#check the score
reg.score(X_train, y_train)
```

Out[120…
```
0.823595487877369
```

In [121…
```
# Predict on the model
#get the X_train and X-test value
y_pred_train=reg.predict(X_train)
y_pred_test=reg.predict(X_test)
print("Predicted target values:", y_pred_test)
```

```
Predicted target values: [3.83170247 4.63074851 4.20125437 ... 6.53797293 0.24193144 5.58539057]
```

1. Explain the ML Model used and it's performance using Evaluation metric Score Chart.

In [122…
```
# Visualizing evaluation Metric Score chart
# Calculate the evaluation metric scores
```

```
train_score = r2_score(y_train, y_pred_train)
print(train_score)
test_score = r2_score(y_test, y_pred_test)
print(test_score)
```

```
0.823595487877369
0.8506149122656624
```

In [123…

```
# Cross checking with cross val score
np.mean(cross_val_score(reg,X_train, y_train,scoring='r2',cv=10))
```

Out[123…

```
0.8193853968676699
```

In [124…

```
# Calculate MSE
MSE_lr = mean_squared_error(y_train, y_pred_train)
print("MSE:", MSE_lr)

# Calculate RMSE
RMSE_lr = np.sqrt(MSE_lr)
print("RMSE:", RMSE_lr)

# Calculate MAE
MAE_lr = mean_absolute_error(y_train, y_pred_train)
print("MAE:", MAE_lr)

# Calculate R2 and Adjusted R2
r2_lr = r2_score(y_train, y_pred_train)
print("R2:", r2_lr)

Adjusted_R2_lr = 1 - (1 - r2_score(y_train, y_pred_train)) * ((X_test.shape[0] - 1) / (X_test.shape[0] - X_test.shape[1]
print("Adjusted R2:", Adjusted_R2_lr)
```

```
MSE: 0.3896761204627392
RMSE: 0.6242404348187798
MAE: 0.4552547308381491
R2: 0.823595487877369
Adjusted R2: 0.819724800636583
```

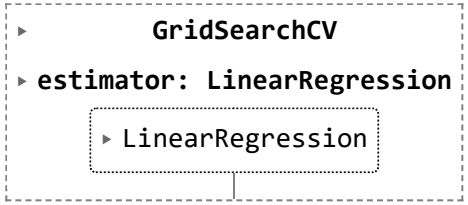
2. Cross- Validation & Hyperparameter Tuning

In [125…

```
# ML Model - 1 Implementation with hyperparameter optimization techniques (i.e., GridSearch CV, RandomSearch CV, Bayesian
# Define the parameter grid for GridSearchCV
```

```
param_grid = {
    'fit_intercept': [True, False],
}

# Fit the Algorithm
# Perform GridSearchCV with cross-validation
grid_search = GridSearchCV(reg, param_grid, cv=5)
grid_search.fit(X_train, y_train)
```

Out[125…



In [126…

```
# Predict on the model
# Print the best parameters and best score
print("Best Parameters: ", grid_search.best_params_)
print("Best Score: ", grid_search.best_score_)
```

```
Best Parameters:  {'fit_intercept': True}
Best Score:  0.8198298922320838
```

In [127…

```
# Perform cross-validation with the best estimator
linear_regression_best = grid_search.best_estimator_
cv_scores = cross_val_score(linear_regression_best, X_train, y_train, cv=10)

# Print the cross-validation scores
print("Cross-Validation Scores:", cv_scores)
print("Mean CV Score:", cv_scores.mean())

# Fit the best estimator on the training data
linear_regression_best.fit(X_train, y_train)

# Evaluate the model on the testing data
test_score = linear_regression_best.score(X_test, y_test)
print("Test Score:", test_score)
```

```
Cross-Validation Scores: [0.79605123 0.83837998 0.82675265 0.81761789 0.81521993 0.83666329
 0.81912939 0.80171001 0.79905568 0.84327392]
```

```
Mean CV Score: 0.8193853968676699
Test Score: 0.8506149122656624
```

Which hyperparameter optimization technique have you used and why?

GridSearchCV is a class in scikit-learn that facilitates performing an exhaustive search over specified parameter values for an estimator. It is commonly used for hyperparameter tuning, which involves finding the optimal values for the hyperparameters of a machine learning model.

Have you seen any improvement? Note down the improvement with updates Evaluation metric Score Chart.

yse, there are slight improvement in the model.

without girdsearchcv--

test Score: 0.7688887774277025

with girdsearchcv--

Test Score: 0.7905536900393838

ML Model - 2

DECISION TREE

1. Explain the ML Model used and it's performance using Evaluation metric Score Chart.

In [128…

```
# ML Model - 1 Implementation
decision_regressor = DecisionTreeRegressor(max_depth=None,
                                           max_features=12,
                                           max_leaf_nodes=150)

# Fit the Algorithm
decision_regressor.fit(X_train, y_train)
```

Out[128…

```
DecisionTreeRegressor
DecisionTreeRegressor(max_features=12, max_leaf_nodes=150)
```

In [129…

```
# Predict on the model
#get the X_train and X-test value
y_pred_train_d = decision_regressor.predict(X_train)
y_pred_test_d = decision_regressor.predict(X_test)
```

In [130…

```
# Visualizing evaluation Metric Score chart

# Model Score
model_score = decision_regressor.score(X_train, y_train)
print("Model Score:", model_score)

# Calculate MSE
MSE_d = mean_squared_error(y_train, y_pred_train_d)
print("MSE:", MSE_d)

# Calculate RMSE
RMSE_d = np.sqrt(MSE_d)
print("RMSE:", RMSE_d)

# Calculate MAE
MAE_d = mean_absolute_error(y_train, y_pred_train_d)
print("MAE:", MAE_d)

# Calculate R2 and adjusted R2
r2_d = r2_score(y_train, y_pred_train_d)
print("R2:", r2_d)

adjusted_r2_d = (1 - (1 - r2_score(y_train, y_pred_train_d)) * ((X_test.shape[0] - 1) / (X_test.shape[0] - X_test.shape[1
print("Adjusted R2:", adjusted_r2_d)
```

```
Model Score: 0.8601638264707616
MSE: 0.30889696043232523
RMSE: 0.5557849947887449
MAE: 0.39912780628637107
R2: 0.8601638264707616
Adjusted R2: 0.8570955257443964
```

2. Cross- Validation & Hyperparameter Tuning

In [131…

```
from sklearn.model_selection import GridSearchCV

# Define the parameter grid for GridSearchCV
```

```
param_grid = {
    'max_depth': [None, 5, 8, 9, 10, 15, 16],
    'max_features': [2, 5, 9, 10, 12],
    'max_leaf_nodes': [100, 150, 180]
}

# Create an instance of DecisionTreeRegressor
decision_regressor = DecisionTreeRegressor()

# Create the GridSearchCV object
grid_search = GridSearchCV(decision_regressor, param_grid, cv=5)

# Fit the data to perform the grid search
grid_search.fit(X_train, y_train)

# Get the best parameters and best score
best_params = grid_search.best_params_
best_score = grid_search.best_score_

# Use the best estimator for predictions
best_estimator = grid_search.best_estimator_
y_pred_test_tuned = best_estimator.predict(X_test)
```

In [132…

```
# Print the best parameters and best score
print("Best Parameters: ", best_params)
print("Best Score: ", best_score)
```

```
Best Parameters:  {'max_depth': None, 'max_features': 12, 'max_leaf_nodes': 180}
Best Score:  0.8340977083056268
```

In [133…

```
# Perform cross-validation with best parameters
cross_val_scores = cross_val_score(decision_regressor,X_train, y_train, cv=10)
print("Cross-Validation Scores: ", cross_val_scores)
print("Average Cross-Validation Score: ", cross_val_scores.mean())
```

```
Cross-Validation Scores:  [0.88442928 0.88969453 0.89698915 0.86472125 0.87541715 0.8922899
 0.8865758  0.8881226  0.87363977 0.88367461]
Average Cross-Validation Score:  0.8835554042978968
```

Which hyperparameter optimization technique have you used and why?

GridSearchCV is a class in scikit-learn that facilitates performing an exhaustive search over specified parameter values for an estimator. It is commonly used for hyperparameter tuning, which involves finding the optimal values for the hyperparameters of a machine learning

model.

Have you seen any improvement? Note down the improvement with updates Evaluation metric Score Chart.
no i dont seen any improvement.

without girdsearchcv--

test Score: 0.8512091074696333

with girdsearchcv--

Test Score: 0.7410361966835403

3. Explain each evaluation metric's indication towards business and the business impact pf the ML model used.

Each evaluation metric provides a different perspective on the performance of a machine learning model. Here's an explanation of each metric and its indication towards the business impact of the ML model used:

1. Mean Squared Error (MSE):
    - MSE measures the average squared difference between the predicted and actual values.
    - It gives more weight to larger errors due to the squaring operation.
    - A lower MSE indicates better model performance and suggests that the model's predictions are closer to the actual values.
    - In terms of business impact, a lower MSE implies that the model's predictions have less error and are more accurate. This can lead to more reliable decision-making and improved efficiency in business operations.
2. Root Mean Squared Error (RMSE):
    - RMSE is the square root of MSE and represents the average magnitude of the prediction errors.
    - It is expressed in the same units as the target variable, making it easily interpretable.
    - Like MSE, a lower RMSE indicates better model performance and suggests that the model's predictions are closer to the actual values.
    - From a business perspective, a lower RMSE implies that the model's predictions have smaller errors on average. This can increase trust in the model's output and support decision-making with more confidence.
3. Mean Absolute Error (MAE):
    - MAE measures the average absolute difference between the predicted and actual values.
    - Unlike MSE, it does not square the errors, giving equal weight to all errors.

- A lower MAE indicates better model performance and suggests that the model's predictions have less absolute error.
- In terms of business impact, a lower MAE means that the model's predictions are closer to the actual values on average. This can lead to better resource allocation, improved planning, and reduced costs.

4. R-squared (R2):

- R2 measures the proportion of the variance in the target variable that is explained by the model.
- It ranges from 0 to 1, with higher values indicating a better fit of the model to the data.
- An R2 of 1 means that the model explains all the variability in the target variable.
- From a business perspective, a higher R2 suggests that the model captures a larger portion of the target variable's variation, indicating its ability to provide useful insights and predictions.

5. Adjusted R-squared (Adjusted R2):

- Adjusted R2 is a modified version of R2 that adjusts for the number of predictors in the model.
- It penalizes the addition of irrelevant predictors that do not improve the model's performance significantly.
- The adjusted R2 value can be lower than R2 if the model's added predictors are not informative.
- In terms of business impact, the adjusted R2 helps assess the incremental value of adding more predictors to the model. It provides a more conservative estimate of the model's explanatory power and helps in avoiding overfitting.

The business impact of the ML model used depends on the specific context and objectives of the business. However, in general, a well-performing ML model with lower MSE, RMSE, and MAE indicates more accurate predictions and improved decision-making. Higher R2 and adjusted R2 values suggest that the model captures more of the target variable's variation, enabling valuable insights and predictions for the business. Ultimately, a reliable and high-performing ML model can lead to enhanced efficiency, cost reduction, improved resource allocation, and better-informed business decisions.

ML Model - 3

RandomForestRegressor

In [134…

```
# ML Model - 3 Implementation

# Create an instance of the RandomForestRegressor
rf_model = RandomForestRegressor()
```

In [135…
```
# Fit the Algorithm
rf_model.fit(X_train,y_train)
```

Out[135…
```
▾ RandomForestRegressor

RandomForestRegressor()
```

In [136…
```
# Predict on the model
y_pred_train_r = rf_model.predict(X_train)
y_pred_test_r = rf_model.predict(X_test)
```

1. Explain the ML Model used and it's performance using Evaluation metric Score Chart.

In [137…
```
# Model Score
model_score = rf_model.score(X_train, y_train)
print("Model Score:", model_score)

# Calculate MSE
MSE_rf = mean_squared_error(y_train, y_pred_train_r)
print("MSE:", MSE_rf)

# Calculate RMSE
RMSE_rf = np.sqrt(MSE_rf)
print("RMSE:", RMSE_rf)

# Calculate MAE
MAE_rf = mean_absolute_error(y_train, y_pred_train_r)
print("MAE:", MAE_rf)

# Calculate R2 and adjusted R2
r2_rf = r2_score(y_train, y_pred_train_r)
print("R2:", r2_rf)

adjusted_r2_rf = (1 - (1 - r2_score(y_train, y_pred_train_r)) * ((X_test.shape[0] - 1) / (X_test.shape[0] - X_test.shape[
print("Adjusted R2:", adjusted_r2_rf)
```

```
Model Score: 0.9916287830919259
MSE: 0.018491949491760733
RMSE: 0.13598510761021124
MAE: 0.08592215872893594
```

```
R2: 0.9916287830919259
Adjusted R2: 0.99144510092821
```

2. Cross- Validation & Hyperparameter Tuning

In [138…

```
# ML Model - 3 Implementation with hyperparameter optimization techniques (i.e., GridSearch CV, RandomSearch CV, Bayesian
# Number of trees in random forest
n_estimators = [20,60,100,120]

# Number of features to consider at every split
max_features = [0.2,0.6,1.0]

# Maximum number of levels in tree
max_depth = [2,8,None]

# Number of samples
max_samples = [0.5,0.75,1.0]

# Bootstrap samples
bootstrap = [True,False]

# Minimum number of samples required to split a node
min_samples_split = [2, 5]

# Minimum number of samples required at each leaf node
min_samples_leaf = [1, 2]
```

In [139…

```
param_grid = {'n_estimators': n_estimators,
               'max_features': max_features,
               'max_depth': max_depth,
              'max_samples':max_samples,
              'bootstrap':bootstrap,
              'min_samples_split':min_samples_split,
              'min_samples_leaf':min_samples_leaf
             }
print(param_grid)
```

```
{'n_estimators': [20, 60, 100, 120], 'max_features': [0.2, 0.6, 1.0], 'max_depth': [2, 8, None], 'max_samples': [0.5, 0.7
5, 1.0], 'bootstrap': [True, False], 'min_samples_split': [2, 5], 'min_samples_leaf': [1, 2]}
```

In [140…

```
# RandomSearch CV
rf_grid = RandomizedSearchCV(estimator = rf_model,
                        param_distributions = param_grid,
```

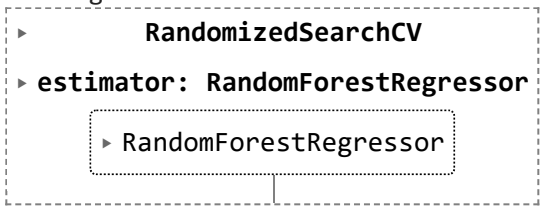
```
                    cv = 5,
                    verbose=2,
                    n_jobs = -1)
```

In [141…

```
rf_grid.fit(X_train,y_train)
```

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits
```

Out[141…



In [142…

```
# beat parameters for random forest regressor
rf_grid.best_params_
```

Out[142…

```
{'n_estimators': 120,
 'min_samples_split': 2,
 'min_samples_leaf': 2,
 'max_samples': 0.75,
 'max_features': 1.0,
 'max_depth': None,
 'bootstrap': True}
```

In [143…

```
# beat score
rf_grid.best_score_
```

Out[143…

```
0.9324976279283128
```

Which hyperparameter optimization technique have you used and why?

GridSearchCV is a class in scikit-learn that allows you to perform an exhaustive search over a specified parameter grid to find the best combination of hyperparameters for a given model. It automates the process of tuning hyperparameters by evaluating the model's performance on different combinations of parameter values using cross-validation.

Using GridSearchCV helps automate the process of hyperparameter tuning and allows you to find the optimal set of hyperparameters for your model based on the specified parameter grid and evaluation metric.

GridSearchCV is chosen for hyperparameter tuning because it performs an exhaustive search over all specified combinations of hyperparameters, incorporates cross-validation for robust evaluation, automates the process, identifies optimal hyperparameters, and improves the generalizability of the model.

GridSearchCV is good for small dataset. this take more time and more comutation in camparistion to randonmisesearchcv. if we working on big dataset then we chouse other approch to reduse time.

Have you seen any improvement? Note down the improvement with updates Evaluation metric Score Chart.
without randomsearchcv

best score : 0.9853788678117673

with randomsearchcv

beat soore : 0.8804336666598515

there are no improvenmat to apply hyparparameter tuning..

1. Which Evaluation metrics did you consider for a positive business impact and why?

When considering a Random Forest model for a positive business impact, the following evaluation metrics can be considered:

1. Mean Squared Error (MSE) or Root Mean Squared Error (RMSE):
    - These metrics measure the average squared or square root of the differences between the predicted and actual values.
    - Lower MSE or RMSE values indicate better model performance with smaller prediction errors.
    - In a business context, lower MSE or RMSE suggests more accurate predictions, which can lead to improved decision-making, cost reduction, and increased operational efficiency.
2. Mean Absolute Error (MAE):
    - MAE measures the average absolute difference between the predicted and actual values.
    - Similar to MSE and RMSE, lower MAE values indicate better model performance with smaller errors.
    - In a business setting, lower MAE suggests more accurate predictions, which can support decision-making, resource allocation, and cost reduction.
3. R-squared (R2) or Adjusted R-squared (Adjusted R2):
    - R2 measures the proportion of the variance in the target variable explained by the model.
    - Higher R2 or Adjusted R2 values indicate a better fit of the model to the data.

- In a business context, higher R2 or Adjusted R2 values suggest that the model captures a larger portion of the target variable's variation, providing valuable insights for decision-making and prediction purposes.

When evaluating a Random Forest model, it's important to consider multiple metrics to gain a comprehensive understanding of its performance. MSE, RMSE, and MAE focus on prediction errors, while R2 and Adjusted R2 assess the model's explanatory power. By assessing these metrics, businesses can determine the accuracy, reliability, and predictive power of the Random Forest model, ultimately leading to informed decision-making and positive business impact.

2. Which ML model did you choose from the above created models as your final prediction model and why?

i get best score in random forest model so choose random forest model. Random Forest is a popular and effective model for prediction due to its high accuracy, robustness to noise and outliers, ability to handle non-linear relationships, feature importance analysis, scalability, and capability to handle missing data.

3. Explain the model which you have used and the feature importance using any model explainability tool?

i am using random forest model,Random Forest Regression is a versatile and powerful algorithm that can handle a wide range of regression tasks. It is particularly useful when there are non-linear relationships and interactions among features, and when robustness to outliers is desired.

Model explainability tools provide insights into feature importance by analyzing the behavior and impact of features on the model's predictions. These tools aim to provide interpretability and transparency to complex machine learning models. Here are some popular model explainability tools that can be used to determine feature importance:

1. SHAP (SHapley Additive exPlanations): SHAP is a unified approach to explain the output of any machine learning model. It assigns each feature an importance value based on the contribution of that feature to the prediction for each instance. SHAP values capture both the individual feature importance and the interactions among features.

2. LIME (Local Interpretable Model-agnostic Explanations): LIME is a model-agnostic method that explains individual predictions. It approximates the behavior of a complex model in the local vicinity of a particular instance. LIME provides feature importances by estimating the contribution of each feature to the model's prediction for that instance.

3. ELI5 (Explain Like I'm 5): ELI5 is a Python library that offers various methods for explaining machine learning models. It supports multiple explainers, such as permutation importance, feature weights, decision tree paths, and more. ELI5 provides intuitive explanations of feature importance that can be easily understood by non-experts.

4. Feature Importance in XGBoost and LightGBM: XGBoost and LightGBM are gradient boosting frameworks that provide built-in methods to calculate feature importance. These methods are based on the gain or cover metric, which measures the contribution of each feature to the improvement of the model's performance.

5. Feature Importance in Random Forest: Random Forest models also provide feature importance measures based on the average impurity decrease caused by each feature across all the decision trees in the ensemble.

These tools enable you to explore and visualize the importance of features in a model. They help in understanding which features have the most significant impact on predictions, identifying important patterns or relationships, and identifying potential issues like feature interactions or biases. By gaining insights into feature importance, you can make more informed decisions regarding feature selection, engineering, and model optimization.

https://lime-ml.readthedocs.io/en/latest/lime.html

In [144…

```
# instaling lime
!pip install lime -q
```

In [145…

```
import lime
from lime import lime_tabular
```

In [146…

```
# Define the feature names for LIME
feature_names = X_train.columns.tolist()

# Define the LIME explainer
explainer = lime.lime_tabular.LimeTabularExplainer(X_train.values, feature_names=feature_names, mode='regression')

# Select a random test instance for explanation
instance_idx = np.random.randint(len(X_test))
instance = X_test.iloc[instance_idx]
true_label = y_test.iloc[instance_idx]

# Generate an explanation using LIME
num_features = 47  # Number of features to include in the explanation
exp = explainer.explain_instance(instance.values, rf_model.predict, num_features=num_features)

# Print the true label and the LIME explanation
print("True Label:", true_label)
print("LIME Explanation:")
exp.show_in_notebook()
```

```
True Label: 4.709678652818983
LIME Explanation:
```

8. Future Work (Optional)

1. Save the best performing ml model in a pickle file or joblib file format for deployment process.

In [154…

```
bike_df.drop('Dew_point_temperature', axis=1, inplace=True)
```

```
[CV] END bootstrap=True, max_depth=8, max_features=0.2, max_samples=0.5, min_samples_leaf=2, min_samples_split=5, n_estim
ators=60; total time=   0.2s
[CV] END bootstrap=True, max_depth=8, max_features=0.6, max_samples=0.75, min_samples_leaf=1, min_samples_split=5, n_esti
mators=120; total time=   1.1s
[CV] END bootstrap=False, max_depth=8, max_features=0.2, max_samples=0.5, min_samples_leaf=1, min_samples_split=5, n_esti
mators=120; total time=   0.0s
[CV] END bootstrap=True, max_depth=8, max_features=0.6, max_samples=1.0, min_samples_leaf=1, min_samples_split=2, n_estim
ators=20; total time=   0.2s
[CV] END bootstrap=True, max_depth=8, max_features=0.6, max_samples=0.5, min_samples_leaf=1, min_samples_split=2, n_estim
ators=20; total time=   0.2s
[CV] END bootstrap=False, max_depth=2, max_features=1.0, max_samples=0.5, min_samples_leaf=1, min_samples_split=5, n_esti
mators=120; total time=   0.0s
[CV] END bootstrap=False, max_depth=2, max_features=1.0, max_samples=0.5, min_samples_leaf=1, min_samples_split=5, n_esti
mators=120; total time=   0.0s
[CV] END bootstrap=False, max_depth=2, max_features=1.0, max_samples=0.5, min_samples_leaf=1, min_samples_split=5, n_esti
mators=120; total time=   0.0s
[CV] END bootstrap=True, max_depth=8, max_features=0.6, max_samples=0.75, min_samples_leaf=1, min_samples_split=5, n_esti
mators=120; total time=   1.1s
[CV] END bootstrap=False, max_depth=8, max_features=0.2, max_samples=0.5, min_samples_leaf=1, min_samples_split=5, n_esti
mators=120; total time=   0.0s
[CV] END bootstrap=False, max_depth=8, max_features=0.2, max_samples=0.5, min_samples_leaf=1, min_samples_split=5, n_esti
mators=120; total time=   0.0s
[CV] END bootstrap=True, max_depth=8, max_features=0.6, max_samples=1.0, min_samples_leaf=1, min_samples_split=2, n_estim
ators=20; total time=   0.2s
[CV] END bootstrap=True, max_depth=8, max_features=0.2, max_samples=0.5, min_samples_leaf=2, min_samples_split=5, n_estim
ators=60; total time=   0.2s
[CV] END bootstrap=True, max_depth=8, max_features=0.6, max_samples=0.75, min_samples_leaf=1, min_samples_split=5, n_esti
mators=120; total time=   1.1s
[CV] END bootstrap=False, max_depth=8, max_features=0.2, max_samples=0.5, min_samples_leaf=1, min_samples_split=5, n_esti
mators=120; total time=   0.0s
[CV] END bootstrap=False, max_depth=8, max_features=0.2, max_samples=0.5, min_samples_leaf=1, min_samples_split=5, n_esti
```

```
mators=120; total time=   0.0s
[CV] END bootstrap=True, max_depth=8, max_features=0.6, max_samples=1.0, min_samples_leaf=1, min_samples_split=2, n_estim
ators=20; total time=   0.2s
[CV] END bootstrap=True, max_depth=8, max_features=0.6, max_samples=0.5, min_samples_leaf=1, min_samples_split=2, n_estim
ators=20; total time=   0.2s
[CV] END bootstrap=True, max_depth=8, max_features=0.2, max_samples=0.5, min_samples_leaf=2, min_samples_split=5, n_estim
ators=60; total time=   0.2s
[CV] END bootstrap=True, max_depth=8, max_features=0.6, max_samples=0.75, min_samples_leaf=1, min_samples_split=5, n_esti
mators=120; total time=   1.1s
[CV] END bootstrap=True, max_depth=8, max_features=0.6, max_samples=0.5, min_samples_leaf=1, min_samples_split=2, n_estim
ators=20; total time=   0.2s
[CV] END bootstrap=True, max_depth=8, max_features=0.2, max_samples=0.5, min_samples_leaf=2, min_samples_split=5, n_estim
ators=60; total time=   0.2s
[CV] END bootstrap=False, max_depth=2, max_features=0.2, max_samples=1.0, min_samples_leaf=1, min_samples_split=2, n_esti
mators=120; total time=   0.0s
[CV] END bootstrap=False, max_depth=2, max_features=0.2, max_samples=1.0, min_samples_leaf=1, min_samples_split=2, n_esti
mators=120; total time=   0.0s
[CV] END bootstrap=False, max_depth=8, max_features=0.6, max_samples=1.0, min_samples_leaf=1, min_samples_split=2, n_esti
mators=100; total time=   0.0s
[CV] END bootstrap=False, max_depth=8, max_features=0.6, max_samples=1.0, min_samples_leaf=1, min_samples_split=2, n_esti
mators=100; total time=   0.0s
[CV] END bootstrap=False, max_depth=8, max_features=0.6, max_samples=1.0, min_samples_leaf=1, min_samples_split=2, n_esti
mators=100; total time=   0.0s
[CV] END bootstrap=True, max_depth=2, max_features=0.6, max_samples=1.0, min_samples_leaf=1, min_samples_split=2, n_estim
ators=120; total time=   0.5s
[CV] END bootstrap=True, max_depth=2, max_features=0.6, max_samples=1.0, min_samples_leaf=1, min_samples_split=2, n_estim
ators=120; total time=   0.5s
[CV] END bootstrap=True, max_depth=8, max_features=0.6, max_samples=1.0, min_samples_leaf=1, min_samples_split=2, n_estim
ators=20; total time=   0.2s
[CV] END bootstrap=True, max_depth=8, max_features=0.6, max_samples=0.5, min_samples_leaf=1, min_samples_split=2, n_estim
ators=20; total time=   0.2s
[CV] END bootstrap=True, max_depth=8, max_features=0.2, max_samples=0.5, min_samples_leaf=2, min_samples_split=5, n_estim
ators=60; total time=   0.2s
[CV] END bootstrap=False, max_depth=2, max_features=0.2, max_samples=1.0, min_samples_leaf=1, min_samples_split=2, n_esti
mators=120; total time=   0.0s
[CV] END bootstrap=False, max_depth=2, max_features=0.2, max_samples=1.0, min_samples_leaf=1, min_samples_split=2, n_esti
mators=120; total time=   0.0s
[CV] END bootstrap=False, max_depth=2, max_features=0.2, max_samples=1.0, min_samples_leaf=1, min_samples_split=2, n_esti
mators=120; total time=   0.0s
[CV] END bootstrap=False, max_depth=8, max_features=0.6, max_samples=1.0, min_samples_leaf=1, min_samples_split=2, n_esti
mators=100; total time=   0.0s
[CV] END bootstrap=False, max_depth=8, max_features=0.6, max_samples=1.0, min_samples_leaf=1, min_samples_split=2, n_esti
mators=100; total time=   0.0s
[CV] END bootstrap=True, max_depth=2, max_features=0.6, max_samples=1.0, min_samples_leaf=1, min_samples_split=2, n_estim
ators=120; total time=   0.5s
```

```
[CV] END bootstrap=True, max_depth=2, max_features=0.6, max_samples=1.0, min_samples_leaf=1, min_samples_split=2, n_estim
ators=120; total time=   0.5s
[CV] END bootstrap=True, max_depth=8, max_features=0.6, max_samples=1.0, min_samples_leaf=1, min_samples_split=2, n_estim
ators=20; total time=   0.2s
[CV] END bootstrap=True, max_depth=8, max_features=0.6, max_samples=0.5, min_samples_leaf=1, min_samples_split=2, n_estim
ators=20; total time=   0.2s
[CV] END bootstrap=False, max_depth=2, max_features=1.0, max_samples=0.5, min_samples_leaf=1, min_samples_split=5, n_esti
mators=120; total time=   0.0s
[CV] END bootstrap=False, max_depth=2, max_features=1.0, max_samples=0.5, min_samples_leaf=1, min_samples_split=5, n_esti
mators=120; total time=   0.0s
[CV] END bootstrap=True, max_depth=8, max_features=0.6, max_samples=0.75, min_samples_leaf=1, min_samples_split=5, n_esti
mators=120; total time=   1.1s
[CV] END bootstrap=True, max_depth=2, max_features=0.6, max_samples=1.0, min_samples_leaf=1, min_samples_split=2, n_estim
ators=120; total time=   0.4s
[CV] END bootstrap=True, max_depth=None, max_features=1.0, max_samples=0.75, min_samples_leaf=2, min_samples_split=2, n_e
stimators=120; total time=   2.6s
[CV] END bootstrap=True, max_depth=None, max_features=1.0, max_samples=0.75, min_samples_leaf=2, min_samples_split=2, n_e
stimators=120; total time=   2.6s
[CV] END bootstrap=True, max_depth=None, max_features=1.0, max_samples=0.75, min_samples_leaf=2, min_samples_split=2, n_e
stimators=120; total time=   2.6s
[CV] END bootstrap=True, max_depth=None, max_features=1.0, max_samples=0.75, min_samples_leaf=2, min_samples_split=2, n_e
stimators=120; total time=   2.7s
[CV] END bootstrap=True, max_depth=None, max_features=1.0, max_samples=0.75, min_samples_leaf=2, min_samples_split=2, n_e
stimators=120; total time=   2.7s
```

In [153…

```
categorical_features = ['Hour', 'Seasons', 'Holiday', 'Functioning_Day', 'month', 'weekdays_weekend']
X = bike_df.drop(columns=['Rented_Bike_Count'], axis=1)
y = bike_df['Rented_Bike_Count']
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.25, random_state=0)


step1 = ColumnTransformer(
    transformers=[('col_tnf', OneHotEncoder(sparse=False, drop='first'), categorical_features)],
    remainder='passthrough'
)
step2 = RandomForestRegressor()
pipe = Pipeline([
    ('step1', step1),
    ('step2', step2)
])

pipe.fit(X_train, y_train)

y_pred = pipe.predict(X_test)
```

```
# Apply the square root transformation to the target variable for evaluation
y_test_sqrt = np.sqrt(y_test)
y_pred_sqrt = np.sqrt(y_pred)

print('R2 score:', r2_score(y_test_sqrt, y_pred_sqrt))
print('MAE:', mean_absolute_error(y_test_sqrt, y_pred_sqrt))
```

```
R2 score: 0.9323362582005666
MAE: 0.2438799123543993
```

In [149…

```
# Load the File and predict unseen data.
import joblib

# Assuming 'pipe' is the trained pipeline model
joblib.dump(pipe, 'model_file.pkl')
```

Out[149…

```
['model_file.pkl']
```

2. Again Load the saved model file and try to predict unseen data for a sanity check.

In [150…

```
# Load the saved model
loaded_model = joblib.load('model_file.pkl')
```

In [151…

```
# Create a DataFrame for the unseen data
unseen_data = pd.DataFrame({
    'Hour': [23],
    'Temperature': [3.8],
    'Humidity': [83],
    'Wind_speed': [1.1],
    'Visibility': [390],
    'Solar_Radiation': [0.0],
    'Rainfall': [0.4],
    'Snowfall': [0.0],
    'Seasons': ['Autumn'],
    'Holiday': ['No Holiday'],
    'Functioning_Day': ['Yes'],
    'month': [4],
    'weekdays_weekend': [1]
})

# Make predictions on the unseen data
```

```
predictions = loaded_model.predict(unseen_data)

# Display the predictions
print(predictions)
```

```
[11.75746821]
```

Congrats! Your model is successfully created and ready for deployment on a live server for a real user interaction !!!

Conclusion

In conclusion, data visualization and exploratory data analysis play a crucial role in understanding and gaining insights from datasets. By using various charts and visualizations, we can analyze the distribution of variables, identify patterns, detect outliers, explore relationships between variables, and uncover trends or correlations.

These insights can have a positive impact on businesses by providing valuable information for decision-making, strategy formulation, and optimization. For example, understanding the factors that influence bike rental demand can help businesses optimize their inventory, pricing, and marketing strategies to meet customer needs and maximize revenue. Identifying negative growth factors can also guide businesses in mitigating risks, addressing issues, and improving their offerings.

Additionally, model evaluation metrics and feature importance analysis can aid in assessing the performance and interpretability of machine learning models. Metrics such as mean squared error (MSE), root mean squared error (RMSE), mean absolute error (MAE), and R-squared (R2) can provide insights into the accuracy and predictive power of the models. Feature importance analysis helps in identifying the most influential features that contribute to model predictions, allowing businesses to prioritize resources and focus on the most impactful variables.

Hurrah! You have successfully completed your Machine Learning Capstone Project !!!