# Seoul Bike Sharing Demand

Group Name: Liv.52

| | |
|---|---|
| Abhey Raheja | 22UCS003 |
| Akhil Murarka | 22UCS009 |
| Sameer Khan | 22UCS176 |
| Sanyam Lamba | 22UCS184 |

Submitted to Dr. Aloke Dutta

**Link for Dataset:** [Seoul Bike Sharing Demand Dataset]()
**Citation:** "Seoul Bike Sharing Demand," UCI Machine Learning Repository, 2020. [Online].

**Description of Dataset:** 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. The Dataset is subjected to the Business area.
The Dataset has multivariate characteristics as it contains column features which includes Integer, Date, Binary, Continuous, Categorical.

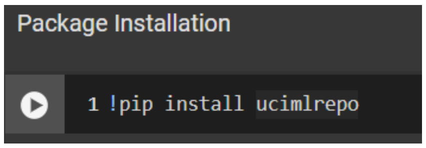
**Feature Description:**

1. **Date:** The date when the data was recorded.
2. **Rented Bike Count:** The number of bikes rented on that particular day and hour.
3. **Hour:** The hour of the day when the data was recorded.
4. **Temperature:** The ambient temperature at the time the data was recorded (usually in Celsius or Fahrenheit).
5. **Humidity:** The relative humidity level at the time the data was recorded (usually in percentage).
6. **Wind Speed:** The wind speed at the time the data was recorded (usually in km/h or m/s).
7. **Visibility:** The distance at which objects can be clearly seen (usually in kilometers or meters).
8. **Dew point temperature:** The temperature at which air becomes saturated with moisture and dew forms.
9. **Solar Radiation:** The amount of solar energy received at a specific location, often measured in watts per square meter ($W/m^2$).
10. **Rainfall:** The amount of rainfall at the time the data was recorded (usually in millimeters).
11. **Snowfall:** The amount of snowfall at the time the data was recorded (usually in centimeters or inches).
12. **Seasons:** The season during which the data was recorded.
13. **Holiday:** Whether the day is a public holiday or not.
14. **Functioning Day:** Indicates if the bike rental service is operational on that day or not.

**Preliminary Analysis Summary:**

- **Data Structure:** The dataset includes numerical features (e.g., Temperature, Humidity) and categorical ones (e.g., Seasons, Holiday). The target variable is Rented Bike Count.
- **Missing Values:** Checked for missing data across features. Imputation or removal methods will be applied if necessary.
- **Outliers:** Visualized via box plots. Winsorization was used to cap extreme values in numerical features.
- **Correlations:** Analyzed relationships between weather features and the target variable. Strong correlations suggest key predictors.
- **Class Distribution:** Checked for any class imbalance, especially in the target variable. Techniques like oversampling may be used if necessary.
- **Feature Encoding:** Categorical features will be encoded (e.g., Seasons, Holiday). Scaling may be applied to numerical features.
- **Conclusion:** Data quality is validated, and preprocessing steps (handling missing values, outliers, encoding) will follow to prepare for model building.

**What used:**

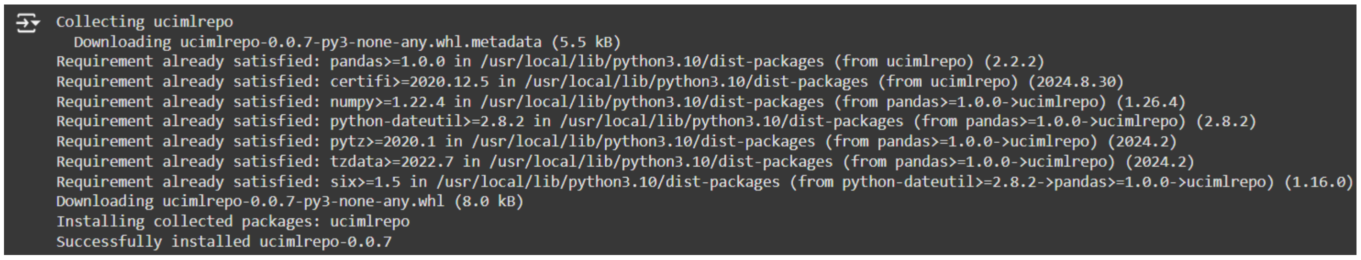


Package Installation

```
!pip install ucimlrepo
```

**Reason:** It is used to install the ucimlrepo packages which contain different types of datasets and provides easy installation of the datasets.

**Output:**



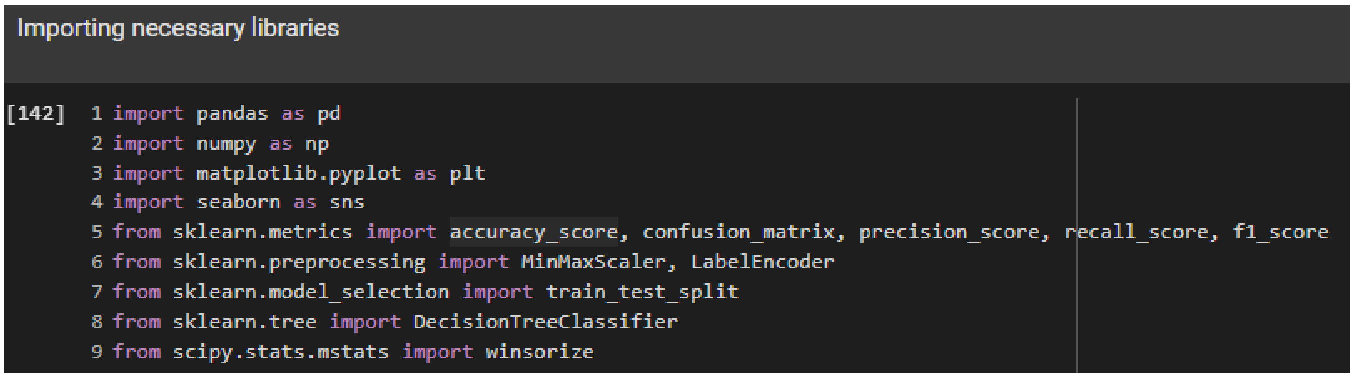
```
Collecting ucimlrepo
  Downloading ucimlrepo-0.0.7-py3-none-any.whl.metadata (5.5 kB)
Requirement already satisfied: pandas>=1.0.0 in /usr/local/lib/python3.10/dist-packages (from ucimlrepo) (2.2.2)
Requirement already satisfied: certifi>=2020.12.5 in /usr/local/lib/python3.10/dist-packages (from ucimlrepo) (2024.8.30)
Requirement already satisfied: numpy>=1.22.4 in /usr/local/lib/python3.10/dist-packages (from pandas>=1.0.0->ucimlrepo) (1.26.4)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.10/dist-packages (from pandas>=1.0.0->ucimlrepo) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas>=1.0.0->ucimlrepo) (2024.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.10/dist-packages (from pandas>=1.0.0->ucimlrepo) (2024.2)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.8.2->pandas>=1.0.0->ucimlrepo) (1.16.0)
Downloading ucimlrepo-0.0.7-py3-none-any.whl (8.0 kB)
Installing collected packages: ucimlrepo
Successfully installed ucimlrepo-0.0.7
```

**Inferences:**

- **Easy Dataset Access:** Directly fetch UCI datasets without manual downloading.
- Dataset Metadata Retrieves dataset details such as size, features, and descriptions.
- **Preprocessing Simplified:** Datasets are naturally ML-ready, reducing preprocessing efforts.

- **Diverse Options:** Datasets across domains are accessible - healthcare, text, finance, etc.

**What used:**



Importing necessary libraries

```
[142] import pandas as pd
      import numpy as np
      import matplotlib.pyplot as plt
      import seaborn as sns
      from sklearn.metrics import accuracy_score, confusion_matrix, precision_score, recall_score, f1_score
      from sklearn.preprocessing import MinMaxScaler, LabelEncoder
      from sklearn.model_selection import train_test_split
      from sklearn.tree import DecisionTreeClassifier
      from scipy.stats.mstats import winsorize
```

**Reason:**

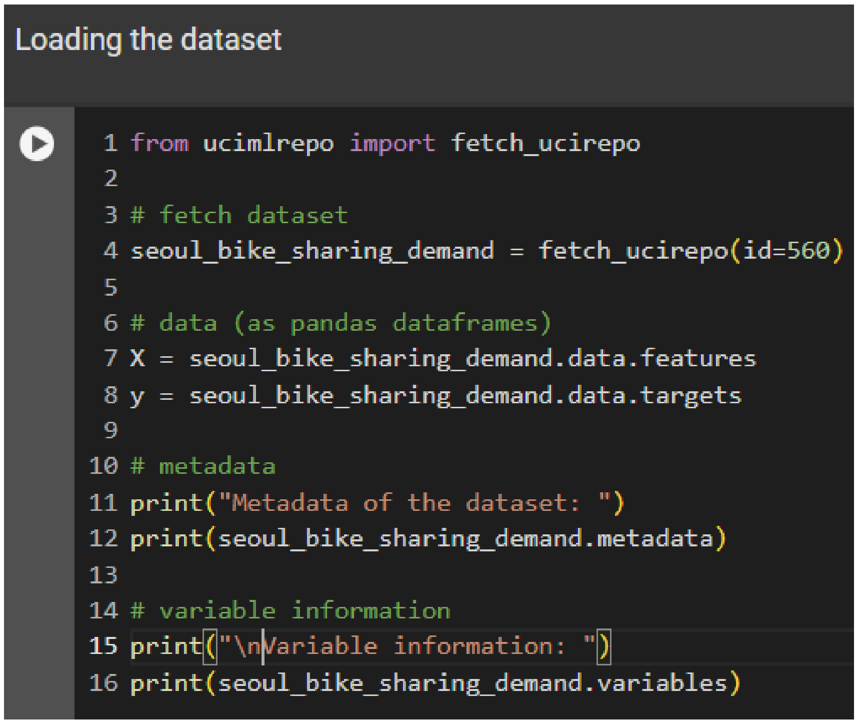
- **Data Handling**: pandas and numpy for manipulation, analysis, and computations.
- **Visualization**: matplotlib and seaborn for exploring data trends and relationships.
- **Model Evaluation**: Metrics like accuracy, precision, recall, and F1-score for performance analysis.
- **Preprocessing**: MinMaxScaler for scaling, LabelEncoder for categorical encoding, winsorize for outlier handling.
- **Modeling**: DecisionTreeClassifier for building a classification model.
- **Data Splitting**: train_test_split for separating training and testing datasets.

**Output:** All the specified libraries are installed and ready to be used.

**Inferences:**

- **Exploratory Data Analysis (EDA):** Visualizations reveal trends, relationships, and outliers.
- **Preprocessing:** Scaling and encoding improve model compatibility.
- **Outlier Handling:** Winsorization ensures stability by capping extremes.
- **Model Performance:** Metrics highlight strengths and weaknesses of the classifier.
- **Decision Tree:** Chosen for interpretability and feature importance insights.

**What used:**



Loading the dataset

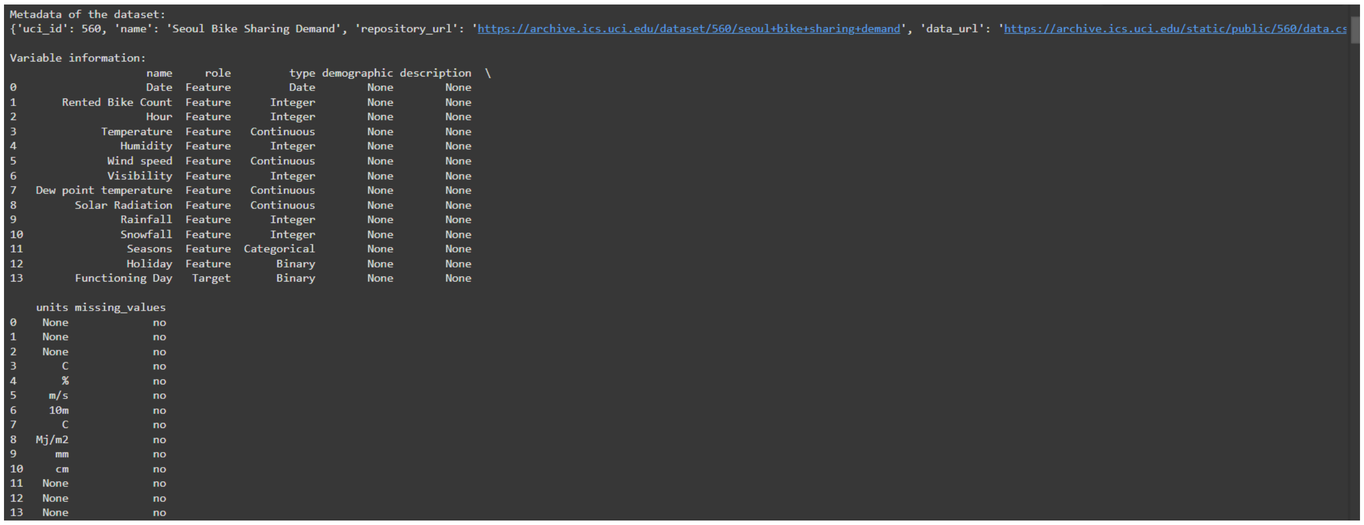
```python
from ucimlrepo import fetch_ucirepo

# fetch dataset
seoul_bike_sharing_demand = fetch_ucirepo(id=560)

# data (as pandas dataframes)
X = seoul_bike_sharing_demand.data.features
y = seoul_bike_sharing_demand.data.targets

# metadata
print("Metadata of the dataset: ")
print(seoul_bike_sharing_demand.metadata)

# variable information
print("\nVariable information: ")
print(seoul_bike_sharing_demand.variables)
```

Imported the dataset from *ucimlrepo* package by using *fetch_ucirepo(id=560)*

**Reason:**
*fetch_ucirepo* used to fetch the dataset with id 560 and storing it in *seoul_bike_sharing_demand*

Splitting the dataset into features by using *seoul_bike_sharing_demand.data.features* and target by *seoul_bike_sharing_demand.data.targets* to combine it into a pandas DataFrame afterwards.

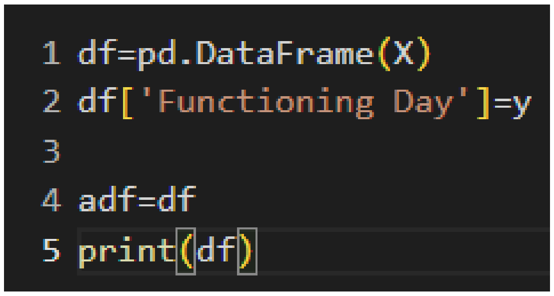
**Output:**


```
Metadata of the dataset:
{'uci_id': 560, 'name': 'Seoul Bike Sharing Demand', 'repository_url': 'https://archive.ics.uci.edu/dataset/560/seoul+bike+sharing+demand', 'data_url': 'https://archive.ics.uci.edu/static/public/560/data.cs

Variable information:
                       name     role         type demographic description  \
0                      Date  Feature         Date        None        None
1         Rented Bike Count  Feature      Integer        None        None
2                      Hour  Feature      Integer        None        None
3               Temperature  Feature   Continuous        None        None
4                  Humidity  Feature      Integer        None        None
5                Wind speed  Feature   Continuous        None        None
6                Visibility  Feature      Integer        None        None
7     Dew point temperature  Feature   Continuous        None        None
8           Solar Radiation  Feature   Continuous        None        None
9                  Rainfall  Feature      Integer        None        None
10                 Snowfall  Feature      Integer        None        None
11                  Seasons  Feature  Categorical        None        None
12                  Holiday  Feature       Binary        None        None
13          Functioning Day   Target       Binary        None        None

    units missing_values
0    None             no
1    None             no
2    None             no
3       C             no
4       %             no
5     m/s             no
6     10m             no
7       C             no
8   Mj/m2             no
9      mm             no
10     cm             no
11   None             no
12   None             no
13   None             no
```

**Inferences:** *seoul_bike_sharing_demand.metadata* gives the metadata of the dataset which includes dataset uci_id, name, repository_url, data_url, abstract, columns information,etc.

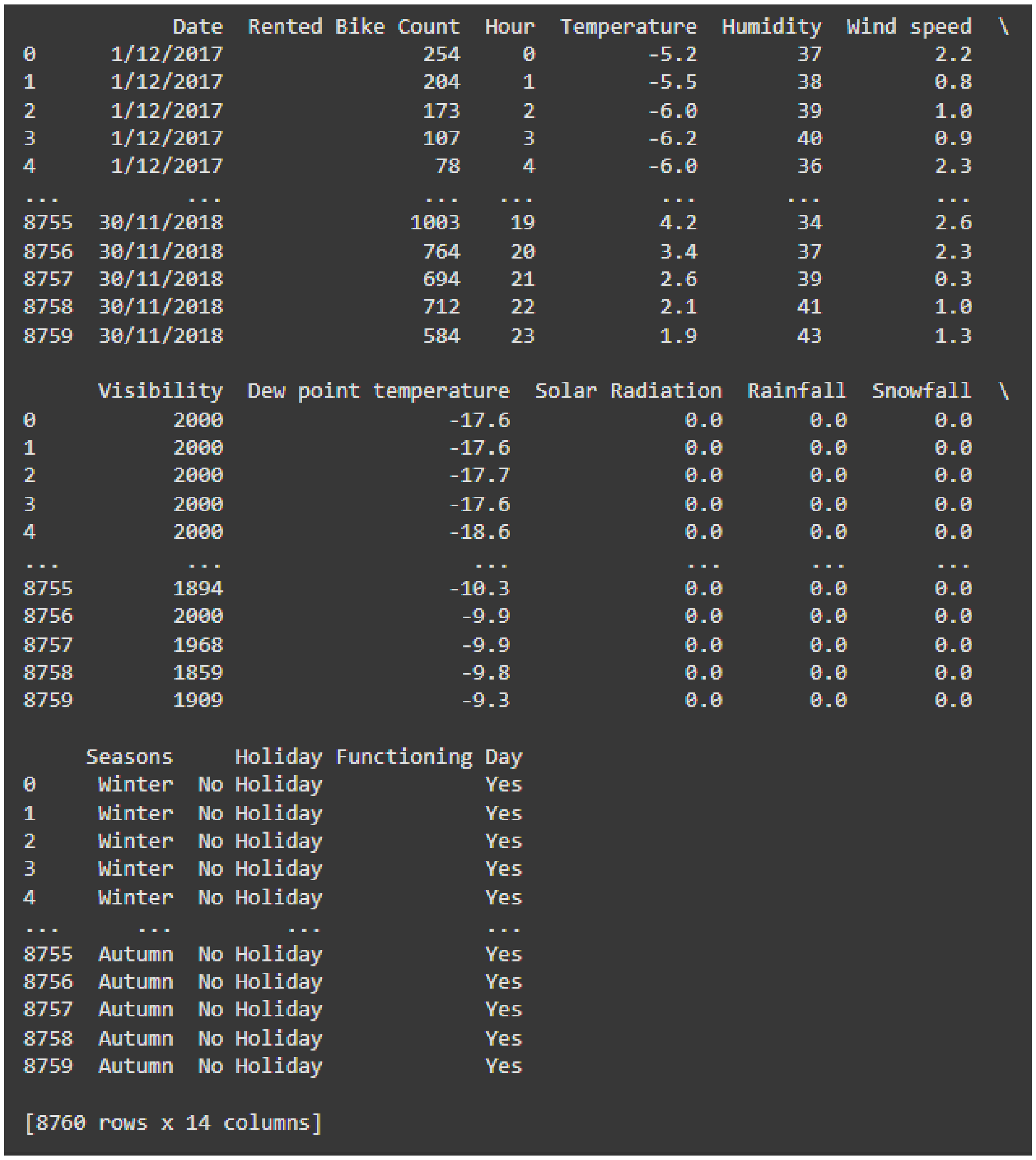
*seoul_bike_sharing_demand.variables* gives the information about column name, data type, missing values, role, units, etc.

**What used:**


```python
df=pd.DataFrame(X)
df['Functioning Day']=y

adf=df
print(df)
```

**Reason:** Combining features and target column data into a single pandas DataFrame to perform operations on them. Creating a copy for future operations.

**Output:**

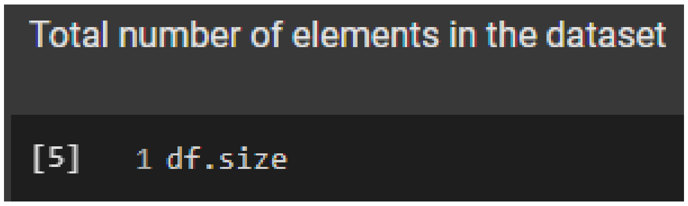


```
            Date  Rented Bike Count  Hour  Temperature  Humidity  Wind speed  \
0      1/12/2017                254     0         -5.2        37         2.2
1      1/12/2017                204     1         -5.5        38         0.8
2      1/12/2017                173     2         -6.0        39         1.0
3      1/12/2017                107     3         -6.2        40         0.9
4      1/12/2017                 78     4         -6.0        36         2.3
...          ...                ...   ...          ...       ...         ...
8755  30/11/2018               1003    19          4.2        34         2.6
8756  30/11/2018                764    20          3.4        37         2.3
8757  30/11/2018                694    21          2.6        39         0.3
8758  30/11/2018                712    22          2.1        41         1.0
8759  30/11/2018                584    23          1.9        43         1.3

      Visibility  Dew point temperature  Solar Radiation  Rainfall  Snowfall  \
0           2000                  -17.6              0.0       0.0       0.0
1           2000                  -17.6              0.0       0.0       0.0
2           2000                  -17.7              0.0       0.0       0.0
3           2000                  -17.6              0.0       0.0       0.0
4           2000                  -18.6              0.0       0.0       0.0
...          ...                    ...              ...       ...       ...
8755        1894                  -10.3              0.0       0.0       0.0
8756        2000                   -9.9              0.0       0.0       0.0
8757        1968                   -9.9              0.0       0.0       0.0
8758        1859                   -9.8              0.0       0.0       0.0
8759        1909                   -9.3              0.0       0.0       0.0

     Seasons     Holiday Functioning Day
0     Winter  No Holiday             Yes
1     Winter  No Holiday             Yes
2     Winter  No Holiday             Yes
3     Winter  No Holiday             Yes
4     Winter  No Holiday             Yes
...      ...         ...             ...
8755  Autumn  No Holiday             Yes
8756  Autumn  No Holiday             Yes
8757  Autumn  No Holiday             Yes
8758  Autumn  No Holiday             Yes
8759  Autumn  No Holiday             Yes

[8760 rows x 14 columns]
```

**Inferences:**

- **DataFrame Structure:** It combines Features (*X*) and target(*y*) into a single dataframe, *df*.
- **Target Column:** *Functioning Day* reflects the operational day of bike-sharing services.
- **Preview of Data:** Shows column names, values.

**What used:**

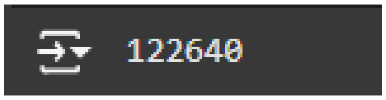


Total number of elements in the dataset

```
[5]   df.size
```

**Reason:** *df.size* gives the total number of elements present in the DataFrame.

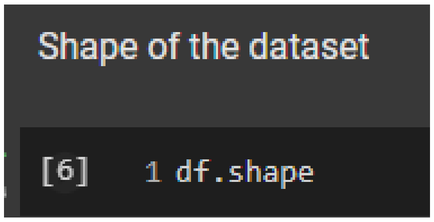
**Output:**



```
122640
```

**Inferences:**
- Total number of elements in the DataFrame is 122640.
- **Preparation for Analysis**:Knowing the size helps estimate memory requirements and computational resources needed for processing.

**What used:**

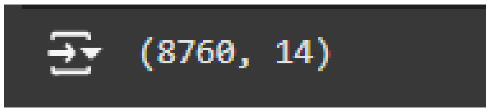


Shape of the dataset

```
[6]   df.shape
```

**Reason:** *df.shape* gives the shape of the dataset in the form of (number of rows, number of columns).

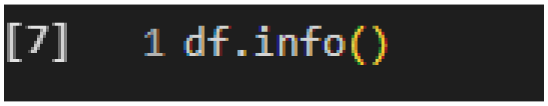
**Output:**



```
(8760, 14)
```

**Inferences:**
- **Dataset Size:** Number of rows (samples) is 8760 and columns (features + target) is 14.
- **Feature Count:** Shows the complexity of the dataset..
- **Balance:** Ratio of rows to columns suggests data sufficiency for analysis.
- **Preparation:** Facilitates planning of data division and resource allocation.

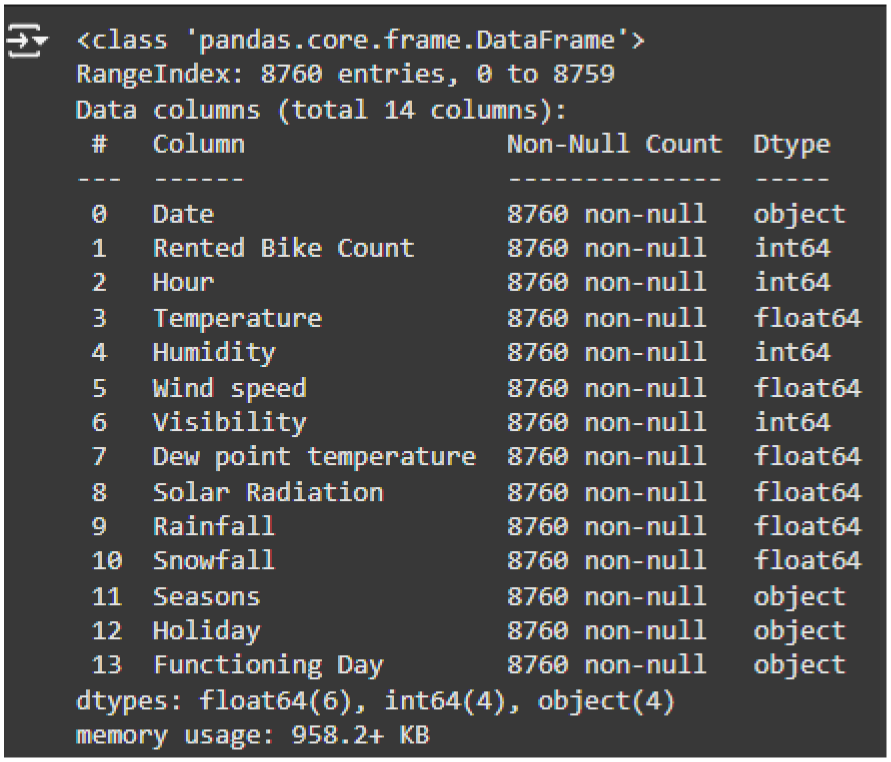
**What used:**



```
[7]   df.info()
```

**Reason:** *df.info()* provides the column information about the dataset, It gives the range Index. Column Index, Column Name, count of non-null values and their respective data types. Also gives the count of each data type like number of columns with float64 data type, and so on for other data types present in the dataset. Memory used for storing the dataset.

**Output:**

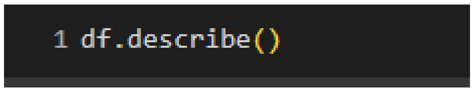


```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8760 entries, 0 to 8759
Data columns (total 14 columns):
 #   Column                 Non-Null Count  Dtype  
---  ------                 --------------  -----  
 0   Date                   8760 non-null   object 
 1   Rented Bike Count      8760 non-null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

**Inferences:**

- **Data Types:** Reveals column data types (e.g., int, float, object).
- **Missing Values:** Indicates columns with missing or null data.
- **Column Names:** Lists the features and target variables.
- **Number of Rows and Columns:** Provides the number of rows and columns in the dataset.
- **Memory Usage:** Provides an estimate of the dataset's memory consumption.
- **Data Integrity:** Helps confirm dataset structure and readiness for analysis.

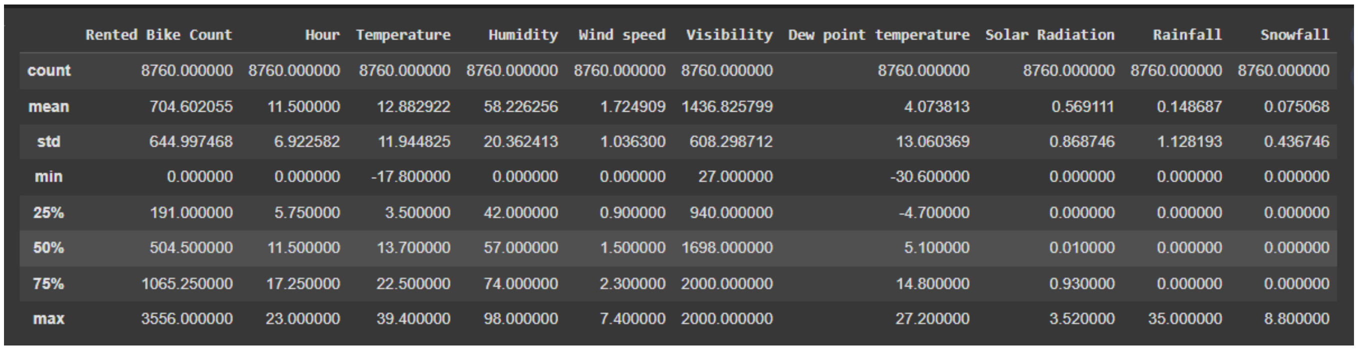
**What used:**



```
df.describe()
```

**Reason:** *df.describe()* gives the information about the numerical features in the dataset. Outputs various statistical measures such as count(number of patterns), minimum, maximum, standard deviation, mean, quartiles.

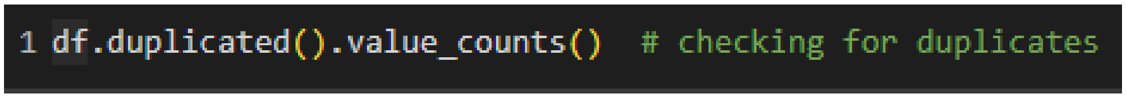
**Output:**



```
       Rented Bike Count         Hour  Temperature     Humidity   Wind speed   Visibility  Dew point temperature  Solar Radiation     Rainfall     Snowfall
count        8760.000000  8760.000000  8760.000000  8760.000000  8760.000000  8760.000000            8760.000000      8760.000000  8760.000000  8760.000000
mean          704.602055    11.500000    12.882922    58.226256     1.724909  1436.825799               4.073813         0.569111     0.148687     0.075068
std           644.997468     6.922582    11.944825    20.362413     1.036300   608.298712              13.060369         0.868746     1.128193     0.436746
min             0.000000     0.000000   -17.800000     0.000000     0.000000    27.000000             -30.600000         0.000000     0.000000     0.000000
25%           191.000000     5.750000     3.500000    42.000000     0.900000   940.000000              -4.700000         0.000000     0.000000     0.000000
50%           504.500000    11.500000    13.700000    57.000000     1.500000  1698.000000               5.100000         0.010000     0.000000     0.000000
75%          1065.250000    17.250000    22.500000    74.000000     2.300000  2000.000000              14.800000         0.930000     0.000000     0.000000
max          3556.000000    23.000000    39.400000    98.000000     7.400000  2000.000000              27.200000         3.520000    35.000000     8.800000
```

**Inferences:**
- Summarizes numeric columns with essential statistics.
- Highlights data spread, central tendency, and outliers.
- Guides feature scaling and data preprocessing.

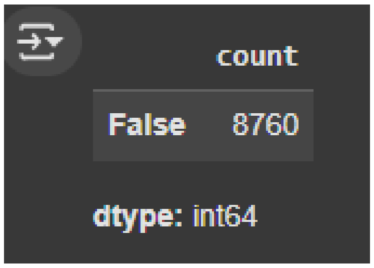
**What used:**



```
df.duplicated().value_counts()  # checking for duplicates
```

**Reason:** *df.duplicated()* gives the bool value. If the row is the same as any row above that, it gives true value and if unique values for row, it gives false. *df.duplicated().value_counts()* provides the total number of true and false values.

**Output:**

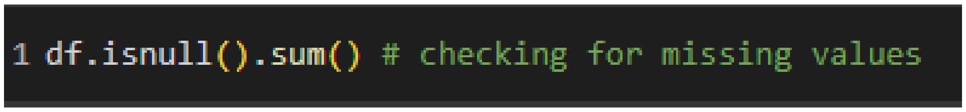


```
       count
False   8760

dtype: int64
```

**Inferences:**

- Shows number of duplicate (true values) and unique rows (false values) in the dataset.
- Helps identify data redundancy that should be addressed in preprocessing.
- As there are no true values, it indicates there are no duplicate values. If duplicate values were there, then one can use *df.drop_duplicates()* which removes the duplicate rows from the dataframe.

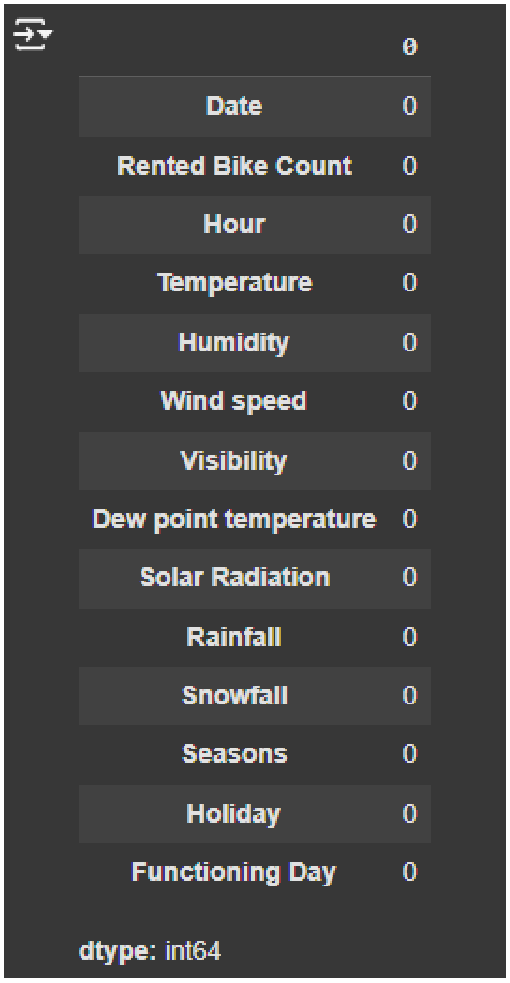
**What used:**



```
df.isnull().sum() # checking for missing values
```

**Reason:** *df.isnull()* gives the bool values for each column. True for null values and False for non-null values. *df.isnull().sum()* does the summation of the true values in each column.

**Output:**



```
                          0
Date                      0
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

dtype: int64
```

**Inferences:**

- First column represents the attributes of the dataframe and the second column represents the number of null values present in it. As this dataset does not contain any null values, all of them are zero.
- If the dataset contains any null values, for features with numerical data type, it can be replaced with null values with mean, median, mode, max, min , standard deviation, etc of the respective column. For features with categorical data it can be replaced with the mode of the respective column.

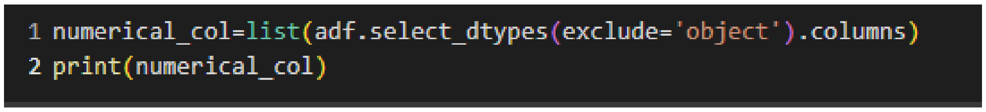
Command for replacing null values: *df.fillna(value)*

- If any column contains zero values instead of null values, the same logic can be used here.

Command for replacing zero values: *df.replace(0,value,inplace=True)*

- **Data Cleaning**: Identifies columns that need handling for missing data (e.g., imputation or removal).

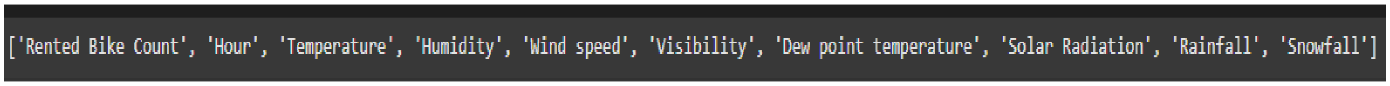
**What used:**



```
numerical_col=list(adf.select_dtypes(exclude='object').columns)
print(numerical_col)
```

**Reason:** *adf.select_dtypes(exclude='object')* outputs the column and their values with data type which are not object data type. *adf.select_dtypes(exclude='object').columns* give the column labels and *list(adf.select_dtypes(exclude='object').columns)* converts the column labels into a list so that we can iterate over it.

**Output:**

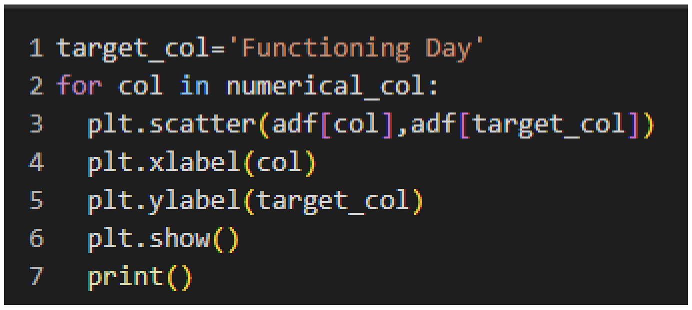


```
['Rented Bike Count', 'Hour', 'Temperature', 'Humidity', 'Wind speed', 'Visibility', 'Dew point temperature', 'Solar Radiation', 'Rainfall', 'Snowfall']
```

**Inferences:**

- **Numerical Columns:** Lists all columns with numerical data types (e.g., int, float) from the DataFrame *adf*.

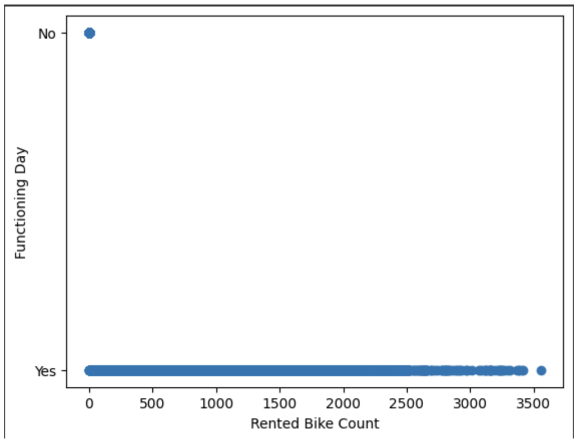
**What used:**

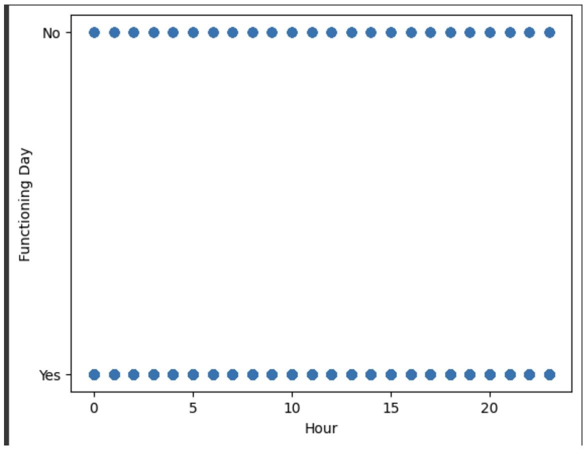


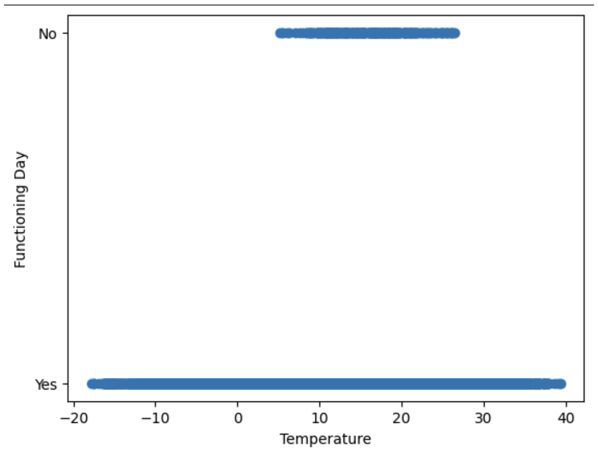
```
target_col='Functioning Day'
for col in numerical_col:
  plt.scatter(adf[col],adf[target_col])
  plt.xlabel(col)
  plt.ylabel(target_col)
  plt.show()
  print()
```

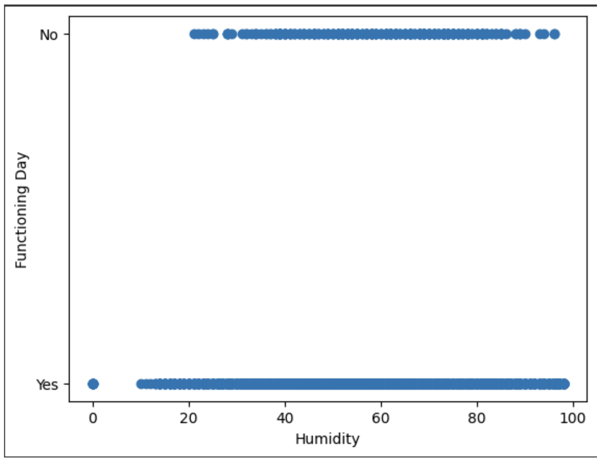
**Reason:** To plot a scatter plot for column labels in the list and the target column by using a for loop.

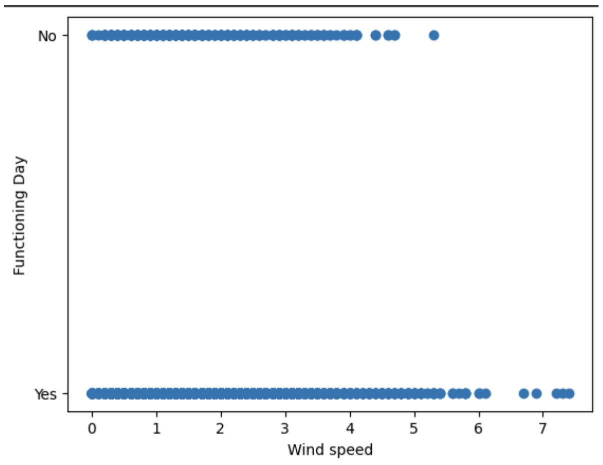
**Output:**

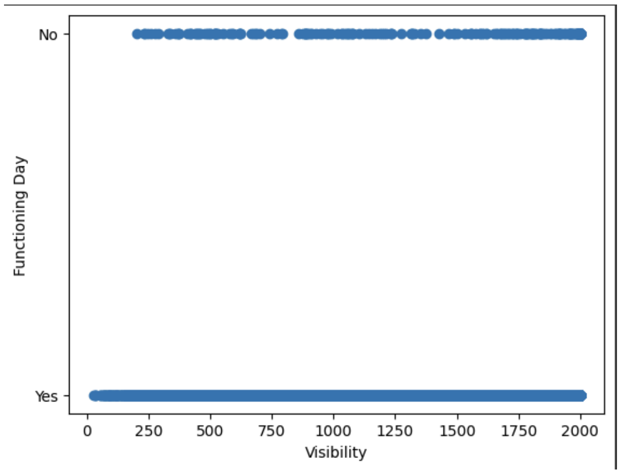

No
Yes
Functioning Day
0
500
1000
1500
2000
2500
3000
3500
Rented Bike Count

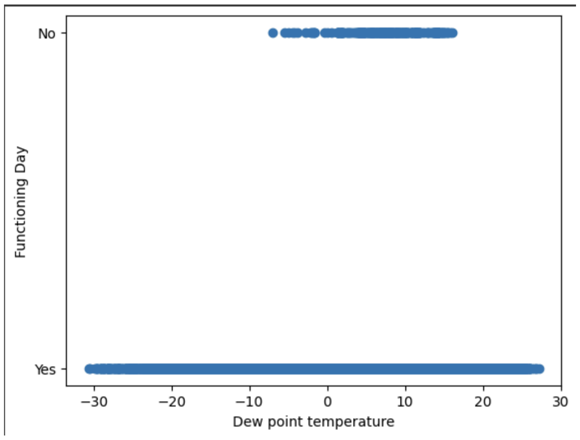

No
Yes
Functioning Day
0
5
10
15
20
Hour

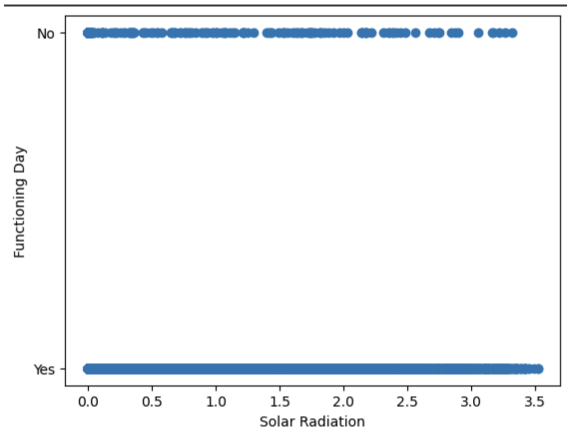

No
Yes
Functioning Day
−20
−10
0
10
20
30
40
Temperature

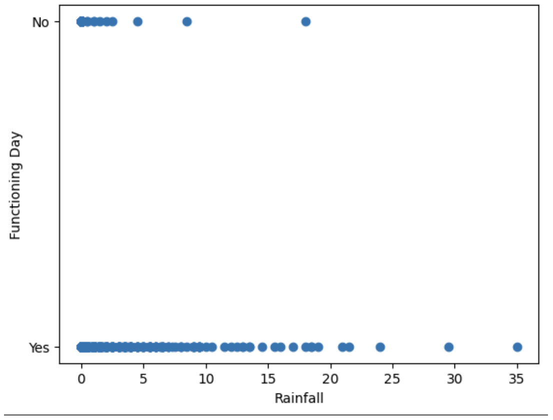

No
Yes
Functioning Day
0
20
40
60
80
100
Humidity

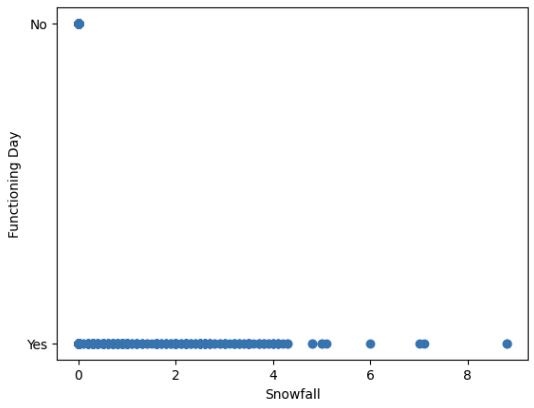

No
Yes
Functioning Day
0
1
2
3
4
5
6
7
Wind speed


No
Yes
Functioning Day
0
250
500
750
1000
1250
1500
1750
2000
Visibility





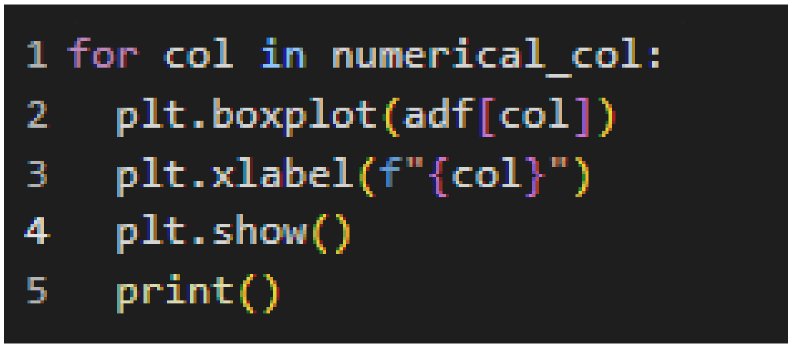




**Inferences:**

- **Scatter Plots**: Visualizes the relationship between each numerical feature and the target variable (*Functioning Day*).
- **Data Patterns**: Helps identify potential correlations, trends, or outliers between features and the target.
- **Feature-Target Analysis**: Useful for assessing which numerical features may influence the target variable.

**What used:**

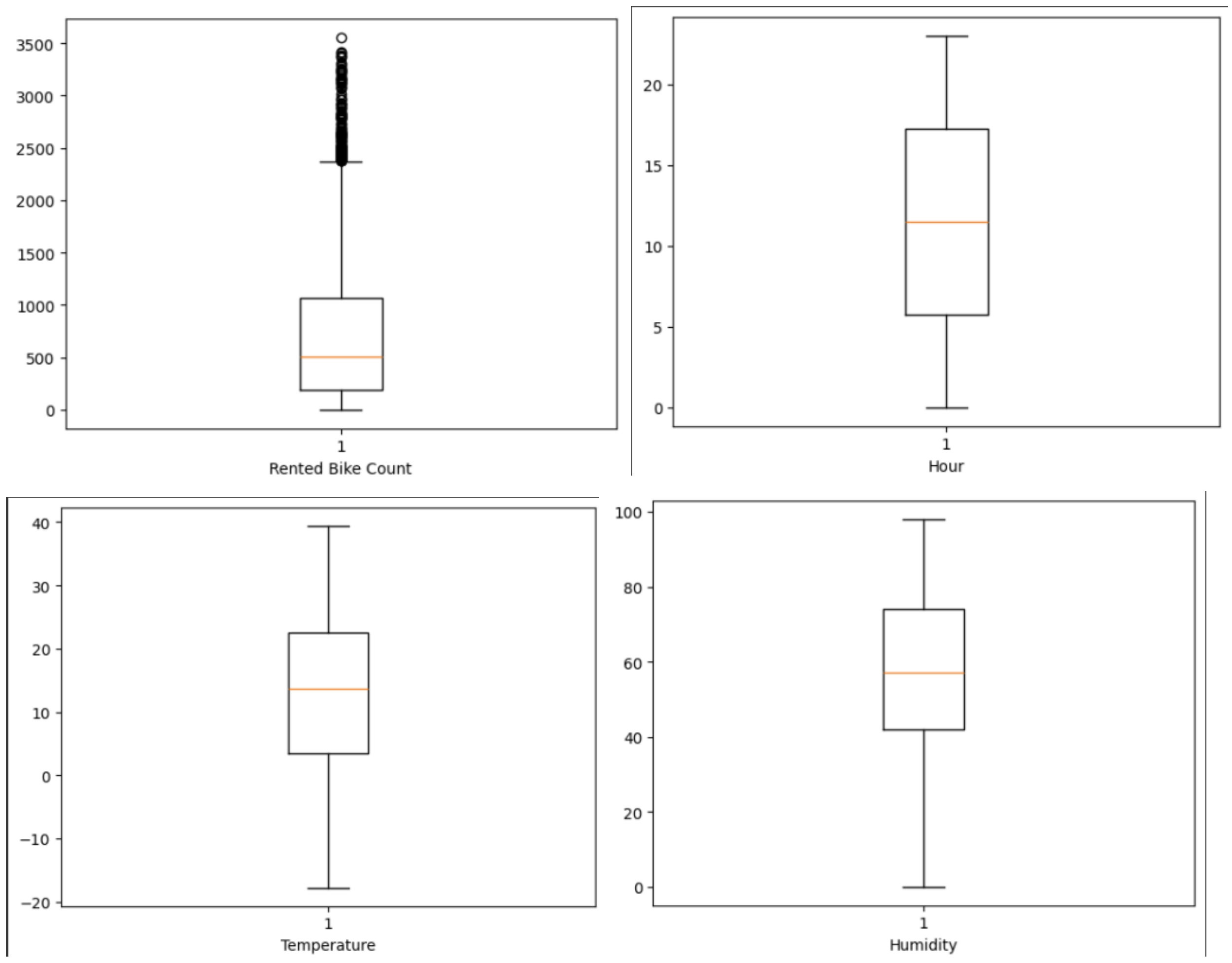


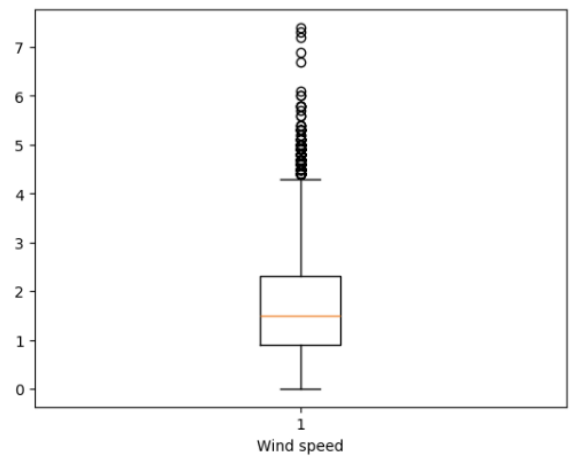
```
for col in numerical_col:
  plt.boxplot(adf[col])
  plt.xlabel(f"{col}")
  plt.show()
  print()
```

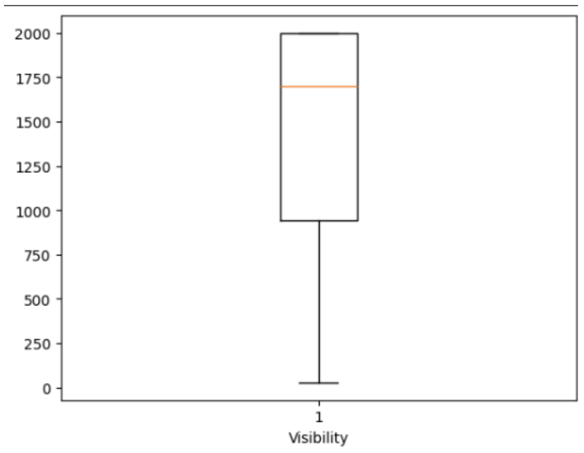
**Reason:**

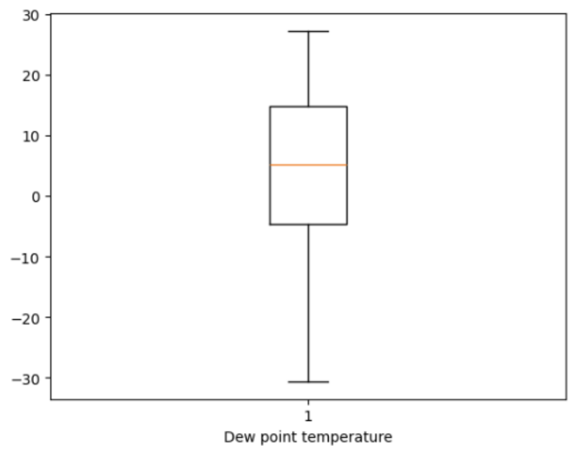
- **Purpose:** To visualize the distribution and detect outliers in numerical columns using box plots.
- **numerical_col:** Assumes it contains the names of numeric columns in the dataset adf.
- **Box Plot:** Highlights the median, quartiles, and potential outliers for each column.

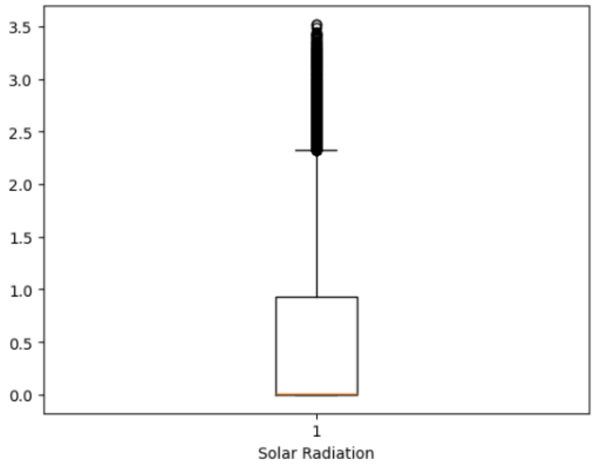
**Output:**

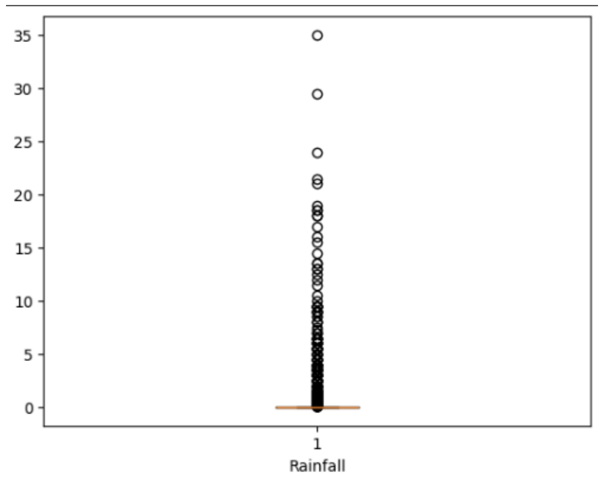


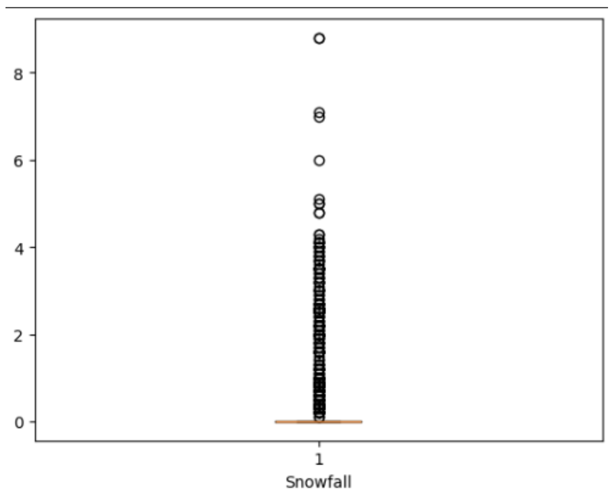










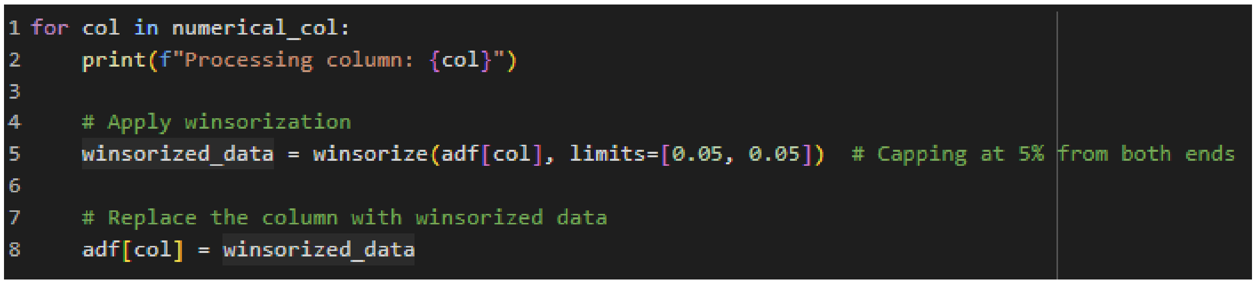


**Inferences:**

- **Outliers:** Points outside the whiskers indicate potential outliers.
- **Skewness:** Uneven spread above or below the median suggests skewed data.

- **Spread:** Width of the interquartile range (IQR) shows data variability for each column.

**What used:**

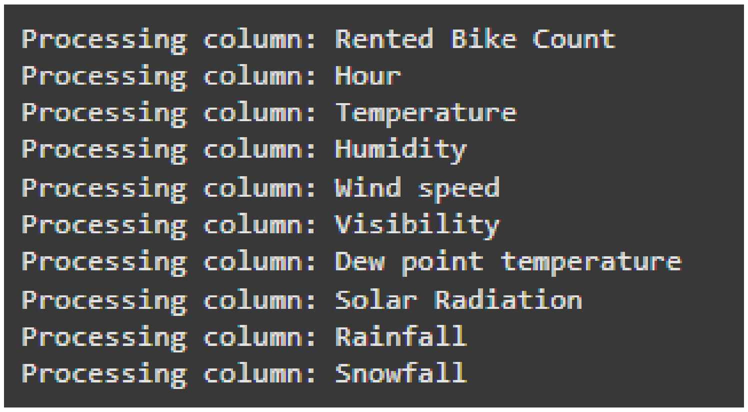


```python
for col in numerical_col:
    print(f"Processing column: {col}")

    # Apply winsorization
    winsorized_data = winsorize(adf[col], limits=[0.05, 0.05])  # Capping at 5% from both ends

    # Replace the column with winsorized data
    adf[col] = winsorized_data
```

**Reason:**

- **Purpose:** To reduce the impact of outliers in numerical columns by applying winsorization.
- **Winsorization:** Caps extreme values at the 5th and 95th percentiles (*limits=[0.05, 0.05]*), ensuring outliers do not distort the dataset.
- **Column Replacement:** Updates the dataset (*adf*) with the adjusted values to maintain consistency.

**Output:**

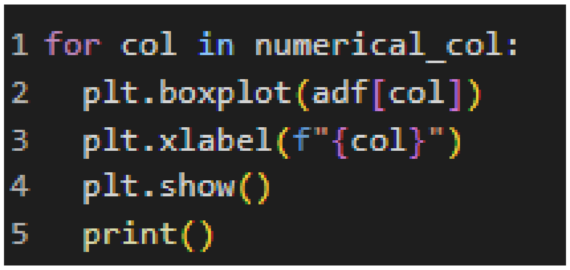


```
Processing column: Rented Bike Count
Processing column: Hour
Processing column: Temperature
Processing column: Humidity
Processing column: Wind speed
Processing column: Visibility
Processing column: Dew point temperature
Processing column: Solar Radiation
Processing column: Rainfall
Processing column: Snowfall
```

**Inferences:**

- **Outlier Mitigation:** Extreme values are capped, leading to more robust statistical analysis and model performance.
- **Uniform Data:** Ensures columns have consistent ranges, reducing potential biases in computations or machine learning models.
- **Preprocessing Step:** Prepares data for better compatibility with models sensitive to outliers, such as regression or tree-based models.

**What used:**

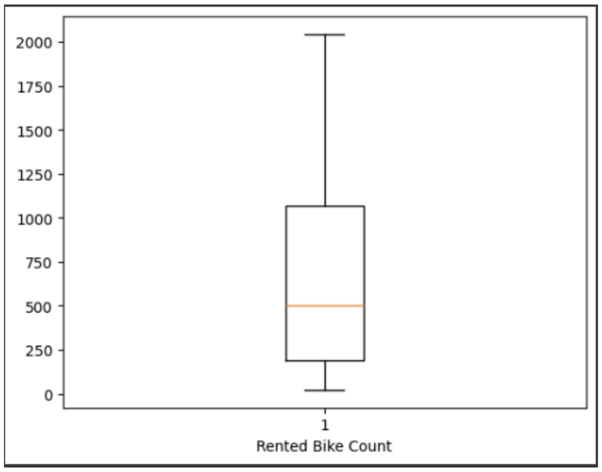


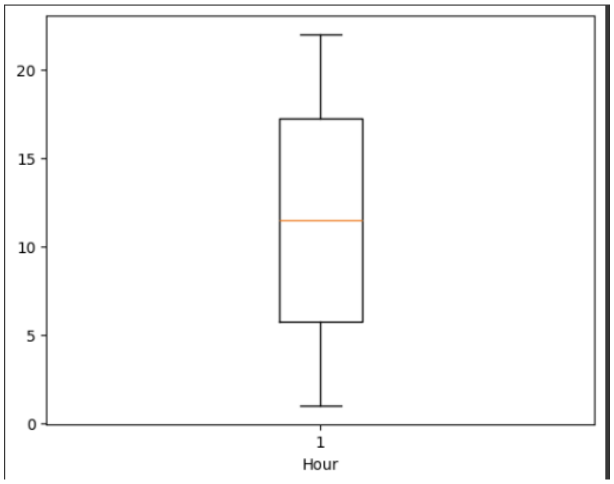
```python
for col in numerical_col:
   plt.boxplot(adf[col])
   plt.xlabel(f"{col}")
   plt.show()
   print()
```

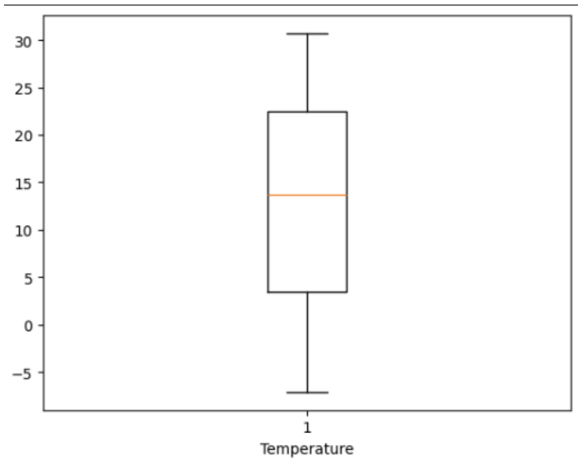
**Reason:**

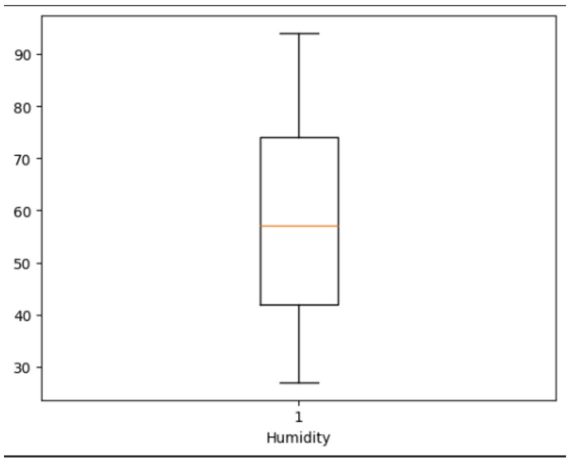
- **Purpose:** To visualize the updated distribution of numerical columns after winsorization.
- **Box Plot:** Helps confirm if outliers were effectively capped by displaying the new data range and quartiles.

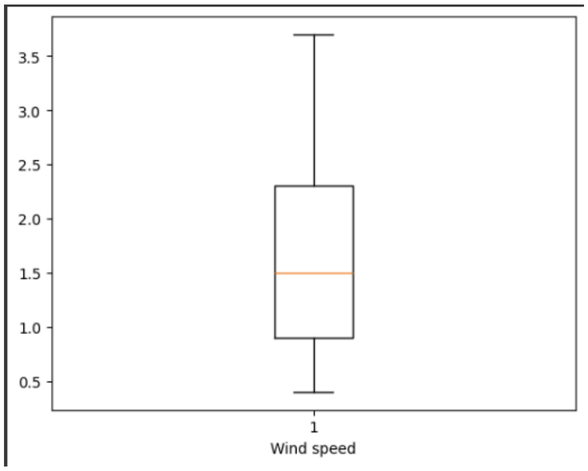
**Output:**

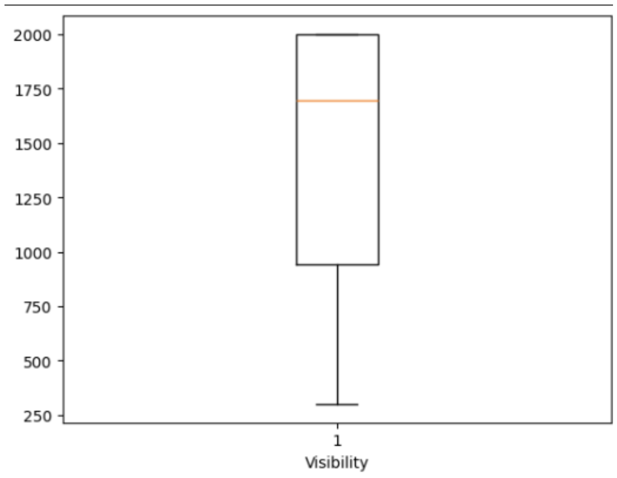


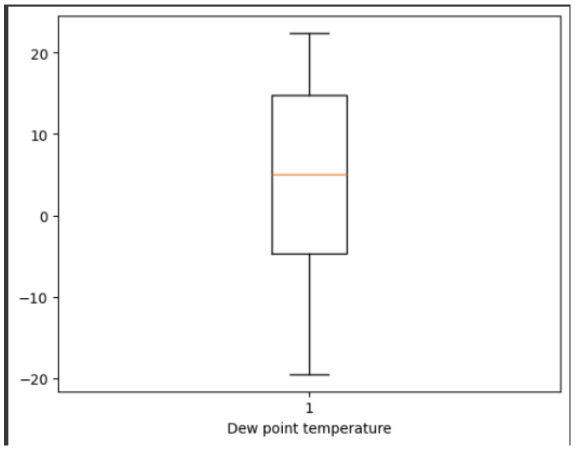


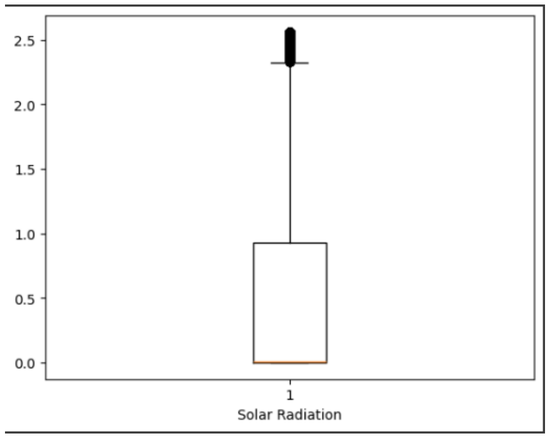


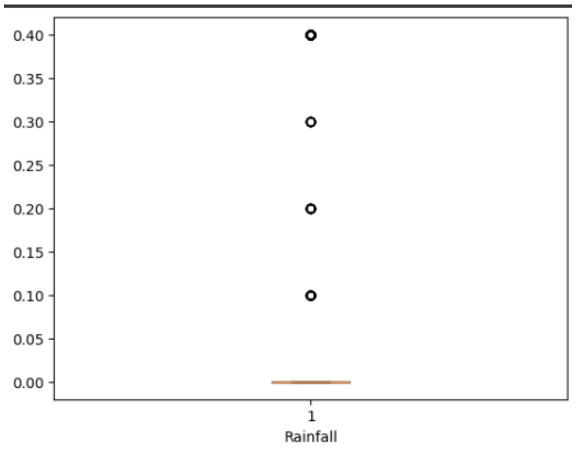


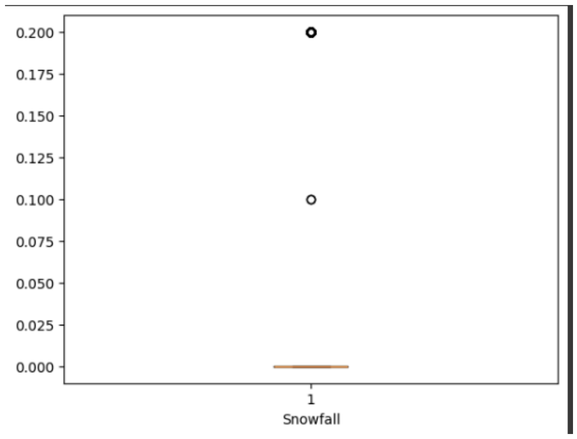










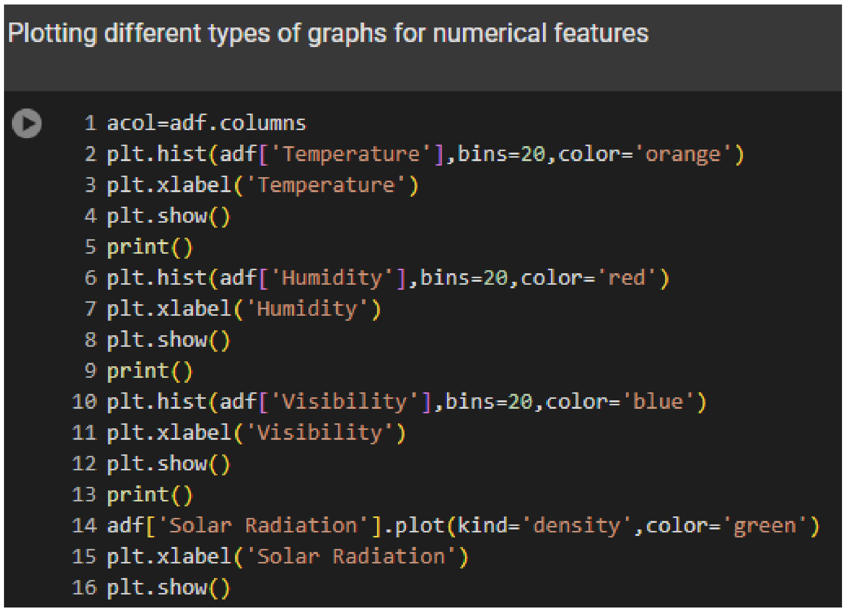


**Inferences:**

- **Outlier Handling Validation:** Post-winsorization, fewer or no points outside the whiskers indicate successful capping.
- **Spread Adjustment:** Changes in interquartile range (IQR) or median might reflect the impact of winsorization.

- **Data Quality Improvement:** Ensures smoother distributions, suitable for further analysis or modeling.

**What used:**

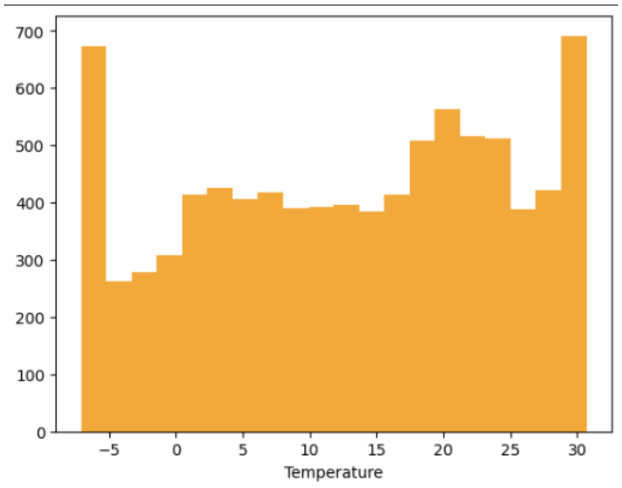


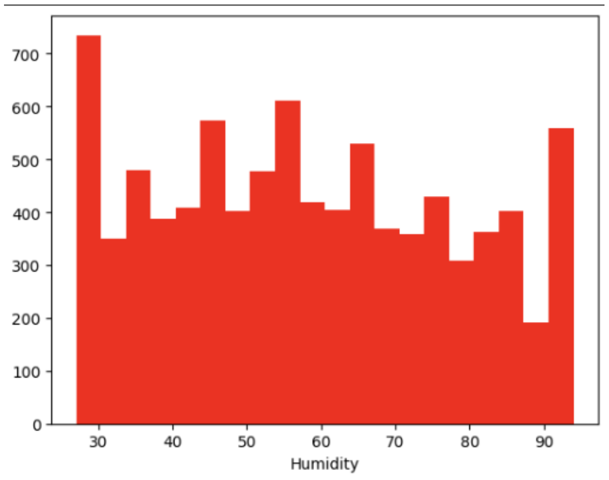
Plotting different types of graphs for numerical features

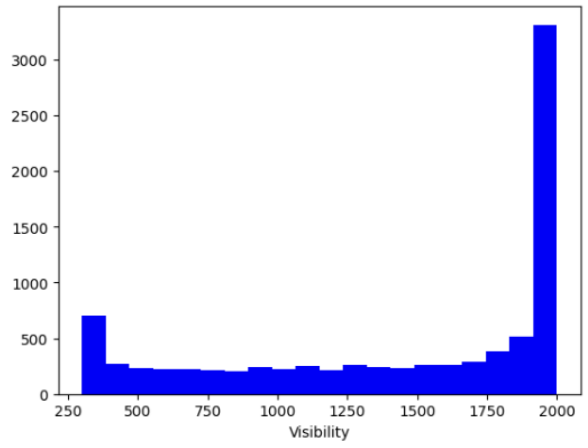
```
acol=adf.columns
plt.hist(adf['Temperature'],bins=20,color='orange')
plt.xlabel('Temperature')
plt.show()
print()
plt.hist(adf['Humidity'],bins=20,color='red')
plt.xlabel('Humidity')
plt.show()
print()
plt.hist(adf['Visibility'],bins=20,color='blue')
plt.xlabel('Visibility')
plt.show()
print()
adf['Solar Radiation'].plot(kind='density',color='green')
plt.xlabel('Solar Radiation')
plt.show()
```

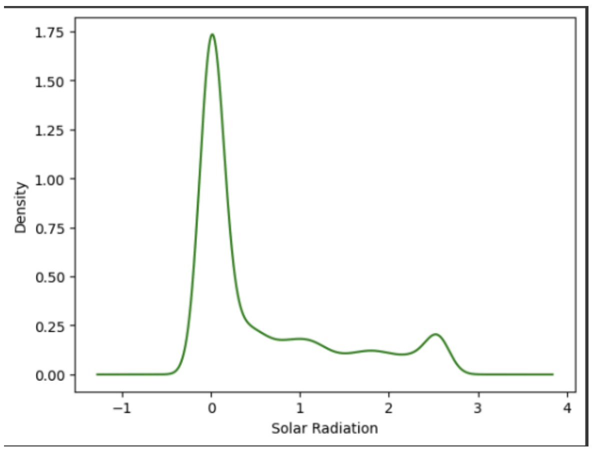
**Reason:** Visualize the distribution of numerical features (*'Temperature', 'Humidity', 'Visibility', 'Solar Radiation'*) to understand their spread and patterns.

**Output:**





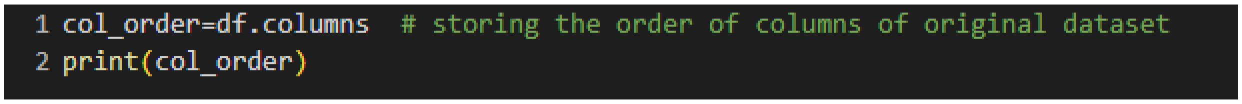




**Inferences:**

- **Temperature, Humidity, Visibility:** Histograms show how these features are distributed (e.g., normal, skewed).
- **Solar Radiation:** The density plot illustrates the distribution of *'Solar Radiation'*, highlighting any peaks or skewness.
- **Feature Understanding:** Helps identify potential outliers and informs decisions for preprocessing or modeling.

**What used:**

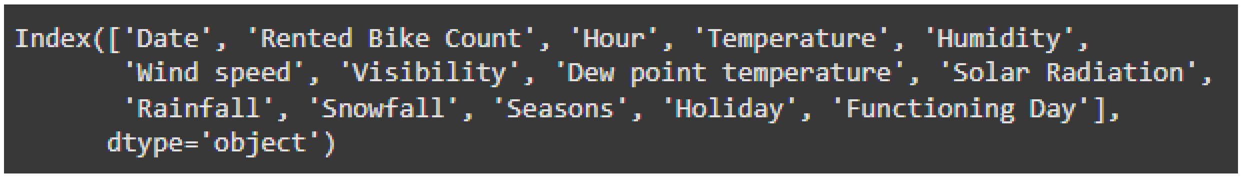


```
col_order=df.columns  # storing the order of columns of original dataset
print(col_order)
```

**Reason:**

- **Store Column Order:** Captures the current order of columns in the DataFrame (*df*) and stores it in *col_order*.

**Output:**

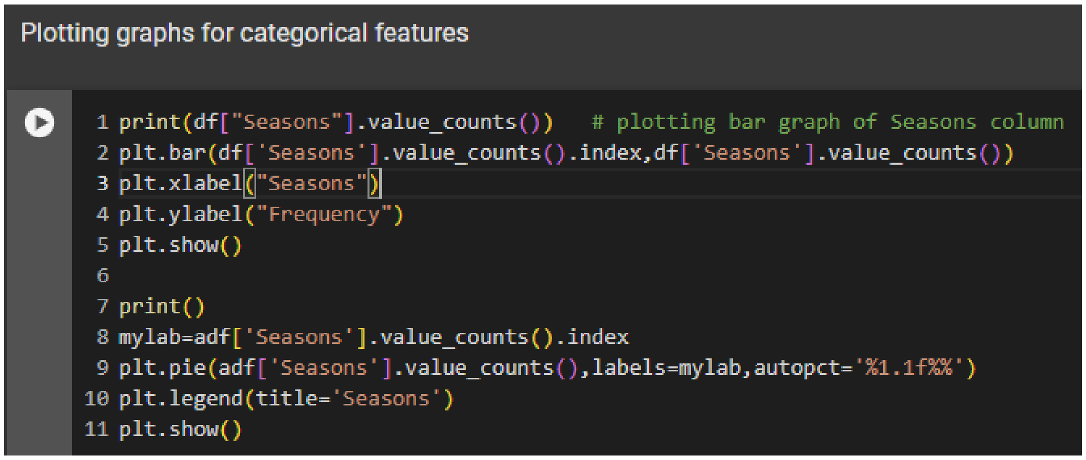


```
Index(['Date', 'Rented Bike Count', 'Hour', 'Temperature', 'Humidity',
       'Wind speed', 'Visibility', 'Dew point temperature', 'Solar Radiation',
       'Rainfall', 'Snowfall', 'Seasons', 'Holiday', 'Functioning Day'],
      dtype='object')
```

**Inferences:**

- **Column Order:** The output lists the column names in their original order.
- **Data Organization:** Helps preserve the order of columns for later use (e.g., after transformations or sorting).

**What used:**



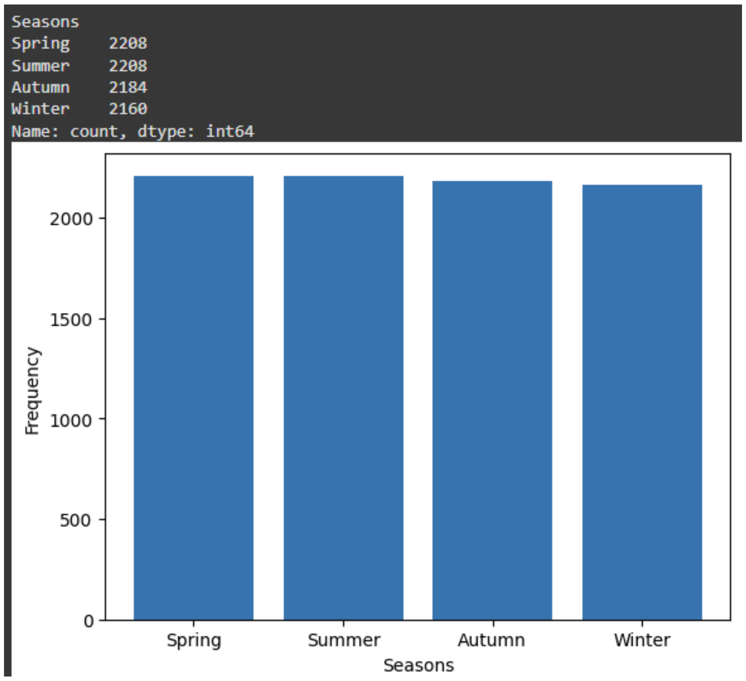
Plotting graphs for categorical features

```
print(df["Seasons"].value_counts())   # plotting bar graph of Seasons column
plt.bar(df['Seasons'].value_counts().index,df['Seasons'].value_counts())
plt.xlabel("Seasons")
plt.ylabel("Frequency")
plt.show()

print()
mylab=adf['Seasons'].value_counts().index
plt.pie(adf['Seasons'].value_counts(),labels=mylab,autopct='%1.1f%%')
plt.legend(title='Seasons')
plt.show()
```

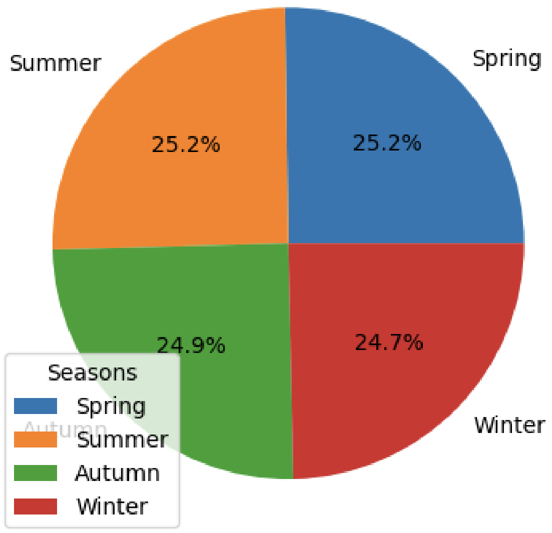
**Reason:**

- **Count and Visualize:** The code counts the occurrences of each unique value in the *'Seasons'* column and visualizes it using a bar graph and a pie chart.

**Output:**



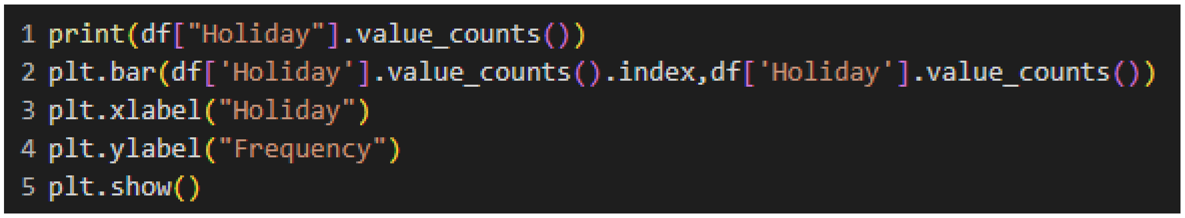
```
Seasons
Spring    2208
Summer    2208
Autumn    2184
Winter    2160
Name: count, dtype: int64
```



**Inferences:**

- **Bar Graph:** Displays the frequency distribution of seasons, allowing easy comparison of how many times each season occurs.
- **Pie Chart:** Shows the proportion of each season category in the dataset, giving a clear view of their relative percentages.
- **Data Distribution:** Helps understand the distribution and balance of the *'Seasons'* variable.

**What used:**



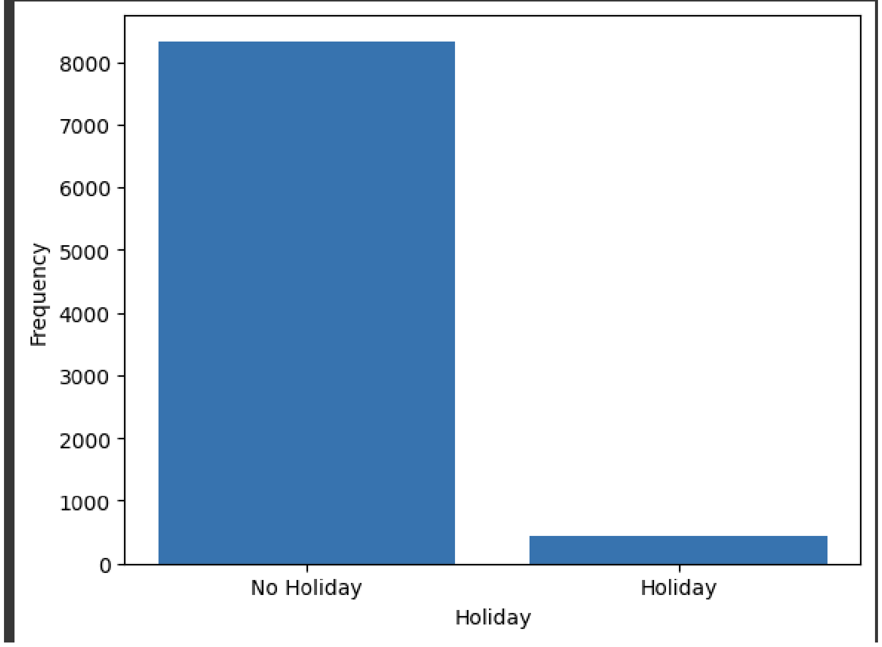
```
print(df["Holiday"].value_counts())
plt.bar(df['Holiday'].value_counts().index,df['Holiday'].value_counts())
plt.xlabel("Holiday")
plt.ylabel("Frequency")
plt.show()
```

**Reason:**

- **Count and Visualize:** The code counts the occurrences of each unique value in the *'Holiday'* column and visualizes it using a bar graph.

**Output:**

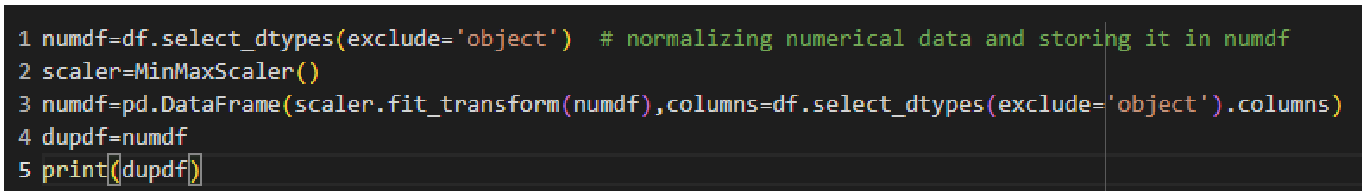
```
Holiday
No Holiday    8328
Holiday        432
Name: count, dtype: int64
```



**Inferences:**

- **Bar Graph:** Displays the frequency of different holiday values, helping to identify which holidays are most common in the dataset.
- **Data Distribution:** Provides insights into how the *'Holiday'* variable is distributed, highlighting any imbalances in the categories.

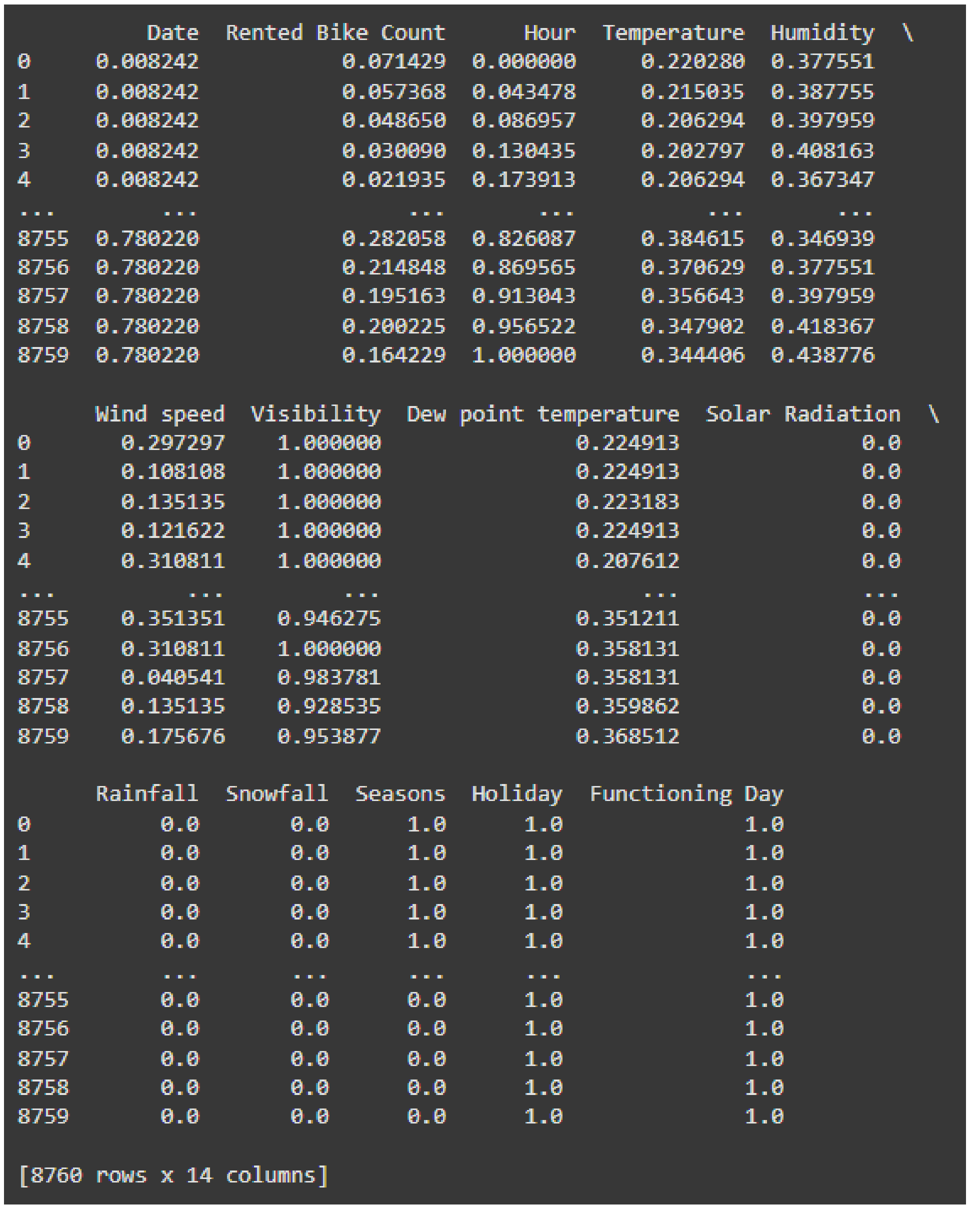
**What used:**


```
numdf=df.select_dtypes(exclude='object')  # normalizing numerical data and storing it in numdf
scaler=MinMaxScaler()
numdf=pd.DataFrame(scaler.fit_transform(numdf),columns=df.select_dtypes(exclude='object').columns)
dupdf=numdf
print(dupdf)
```

**Reason:**

- **Normalize Numerical Data:** The code normalizes all numerical columns in the DataFrame (*df*) using MinMaxScaler to scale the values between 0 and 1, and stores the result in *dupdf*.

**Output:**

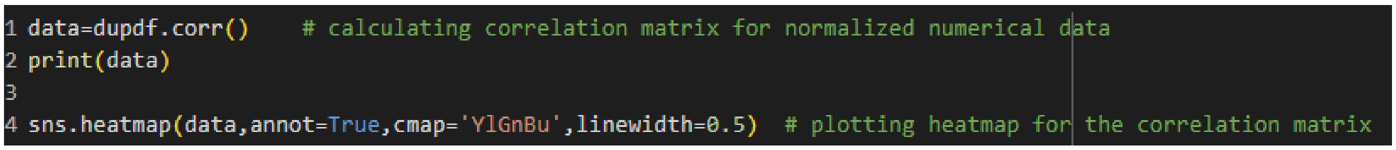


```
          Date  Rented Bike Count      Hour  Temperature  Humidity  \
0     0.008242           0.071429  0.000000     0.220280  0.377551   
1     0.008242           0.057368  0.043478     0.215035  0.387755   
2     0.008242           0.048650  0.086957     0.206294  0.397959   
3     0.008242           0.030090  0.130435     0.202797  0.408163   
4     0.008242           0.021935  0.173913     0.206294  0.367347   
...        ...                ...       ...          ...       ...   
8755  0.780220           0.282058  0.826087     0.384615  0.346939   
8756  0.780220           0.214848  0.869565     0.370629  0.377551   
8757  0.780220           0.195163  0.913043     0.356643  0.397959   
8758  0.780220           0.200225  0.956522     0.347902  0.418367   
8759  0.780220           0.164229  1.000000     0.344406  0.438776   

      Wind speed  Visibility  Dew point temperature  Solar Radiation  \
0       0.297297    1.000000               0.224913              0.0   
1       0.108108    1.000000               0.224913              0.0   
2       0.135135    1.000000               0.223183              0.0   
3       0.121622    1.000000               0.224913              0.0   
4       0.310811    1.000000               0.207612              0.0   
...          ...         ...                    ...              ...   
8755    0.351351    0.946275               0.351211              0.0   
8756    0.310811    1.000000               0.358131              0.0   
8757    0.040541    0.983781               0.358131              0.0   
8758    0.135135    0.928535               0.359862              0.0   
8759    0.175676    0.953877               0.368512              0.0   

      Rainfall  Snowfall  Seasons  Holiday  Functioning Day  
0          0.0       0.0      1.0      1.0              1.0  
1          0.0       0.0      1.0      1.0              1.0  
2          0.0       0.0      1.0      1.0              1.0  
3          0.0       0.0      1.0      1.0              1.0  
4          0.0       0.0      1.0      1.0              1.0  
...        ...       ...      ...      ...              ...  
8755       0.0       0.0      0.0      1.0              1.0  
8756       0.0       0.0      0.0      1.0              1.0  
8757       0.0       0.0      0.0      1.0              1.0  
8758       0.0       0.0      0.0      1.0              1.0  
8759       0.0       0.0      0.0      1.0              1.0  

[8760 rows x 14 columns]
```

**Inferences:**

- **Normalized Data:** All numerical columns in *dupdf* are scaled to a range between 0 and 1, ensuring consistency in the data for machine learning models.
- **Feature Scaling:** Helps improve model performance by avoiding issues caused by differing feature scales.

**What used:**

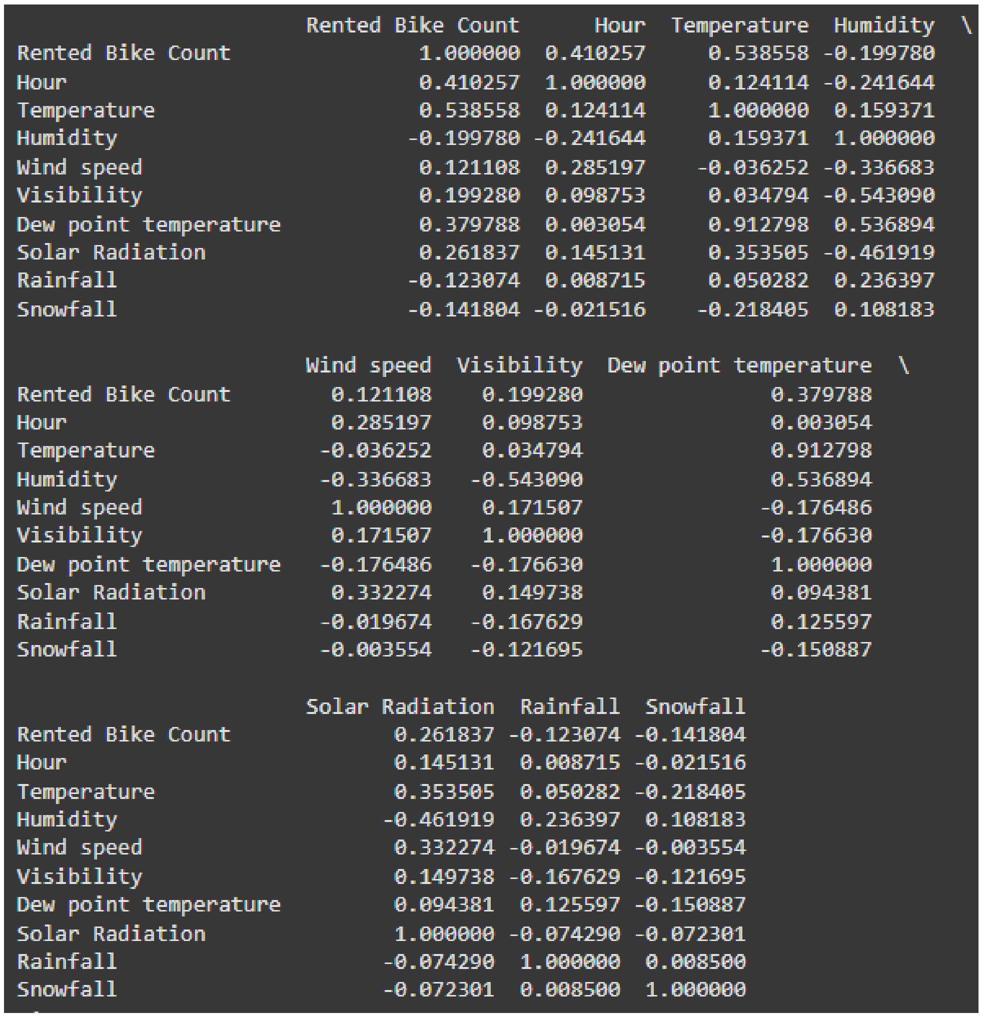


```
data=dupdf.corr()    # calculating correlation matrix for normalized numerical data
print(data)

sns.heatmap(data,annot=True,cmap='YlGnBu',linewidth=0.5)  # plotting heatmap for the correlation matrix
```

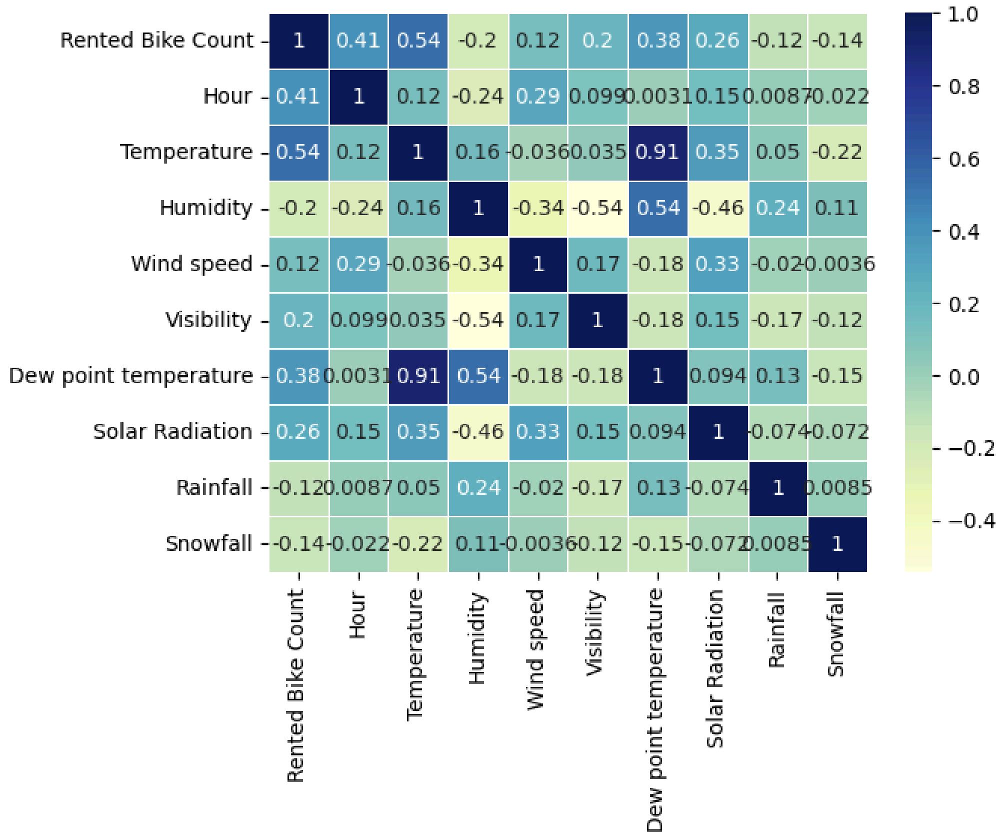
**Reason:**

- **Correlation Matrix:** The code calculates the correlation between normalized numerical features in *dupdf* to understand how each feature relates to others.
- **Heatmap:** It then visualizes the correlation matrix using a heatmap for better insight into the relationships between features.

**Output:**



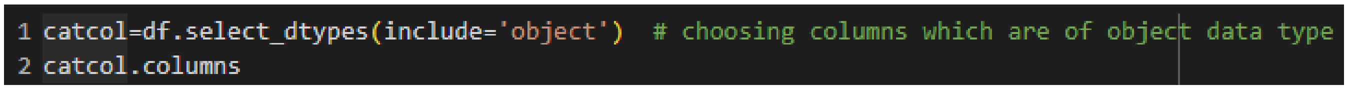
```
                      Rented Bike Count      Hour  Temperature  Humidity  \
Rented Bike Count              1.000000  0.410257     0.538558 -0.199780
Hour                           0.410257  1.000000     0.124114 -0.241644
Temperature                    0.538558  0.124114     1.000000  0.159371
Humidity                      -0.199780 -0.241644     0.159371  1.000000
Wind speed                     0.121108  0.285197    -0.036252 -0.336683
Visibility                     0.199280  0.098753     0.034794 -0.543090
Dew point temperature          0.379788  0.003054     0.912798  0.536894
Solar Radiation                0.261837  0.145131     0.353505 -0.461919
Rainfall                      -0.123074  0.008715     0.050282  0.236397
Snowfall                      -0.141804 -0.021516    -0.218405  0.108183

                      Wind speed  Visibility  Dew point temperature  \
Rented Bike Count       0.121108    0.199280               0.379788
Hour                    0.285197    0.098753               0.003054
Temperature            -0.036252    0.034794               0.912798
Humidity               -0.336683   -0.543090               0.536894
Wind speed              1.000000    0.171507              -0.176486
Visibility              0.171507    1.000000              -0.176630
Dew point temperature  -0.176486   -0.176630               1.000000
Solar Radiation         0.332274    0.149738               0.094381
Rainfall               -0.019674   -0.167629               0.125597
Snowfall               -0.003554   -0.121695              -0.150887

                      Solar Radiation  Rainfall  Snowfall
Rented Bike Count            0.261837 -0.123074 -0.141804
Hour                         0.145131  0.008715 -0.021516
Temperature                  0.353505  0.050282 -0.218405
Humidity                    -0.461919  0.236397  0.108183
Wind speed                   0.332274 -0.019674 -0.003554
Visibility                   0.149738 -0.167629 -0.121695
Dew point temperature        0.094381  0.125597 -0.150887
Solar Radiation              1.000000 -0.074290 -0.072301
Rainfall                    -0.074290  1.000000  0.008500
Snowfall                    -0.072301  0.008500  1.000000
```



**Inferences:**

- **Correlation Matrix:** Shows the strength and direction (positive or negative) of relationships between features, with values ranging from -1 (strong negative) to 1 (strong positive).
- **Heatmap:** Provides a visual representation, with color gradients indicating stronger or weaker correlations. Annotated values show exact correlation coefficients.
- **Feature Relationships:** Helps identify highly correlated features, which may be important for model selection or feature engineering (e.g., potential multicollinearity issues).

**What used:**

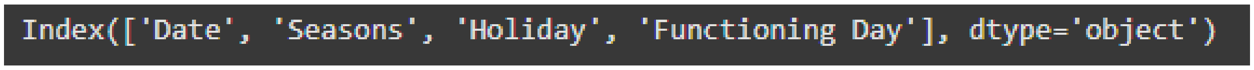


```
catcol=df.select_dtypes(include='object')  # choosing columns which are of object data type
catcol.columns
```

**Reason:**

- **Select Categorical Columns:** The code selects columns with the object data type (usually categorical variables) from the DataFrame (*df*) and stores them in *catcol*.

**Output:**

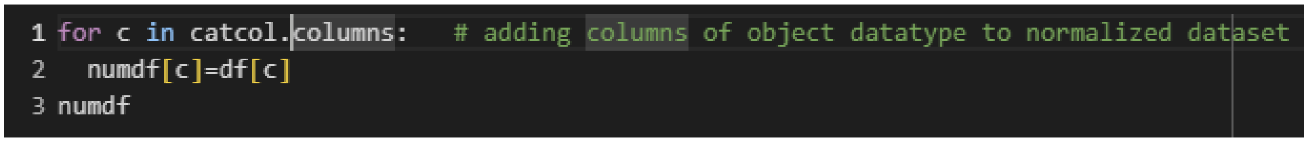


```
Index(['Date', 'Seasons', 'Holiday', 'Functioning Day'], dtype='object')
```

**Inferences:**

- **Categorical Features:** Displays the names of columns that contain categorical data (e.g., text, labels).
- **Data Preprocessing:** Identifies columns that may require encoding (e.g., using one-hot encoding or label encoding) for machine learning models.

**What used:**

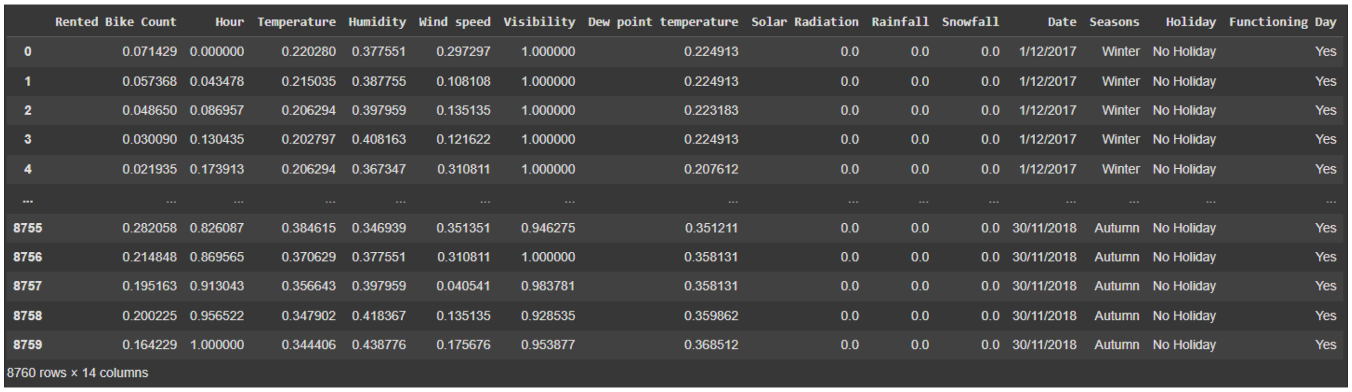


```
for c in catcol.columns:   # adding columns of object datatype to normalized dataset
  numdf[c]=df[c]
numdf
```

**Reason:**

- **Combine Categorical and Normalized Data:** Adds categorical columns from the original DataFrame (*df*) to the normalized numerical dataset (*numdf*) to create a unified dataset.

**Output:**



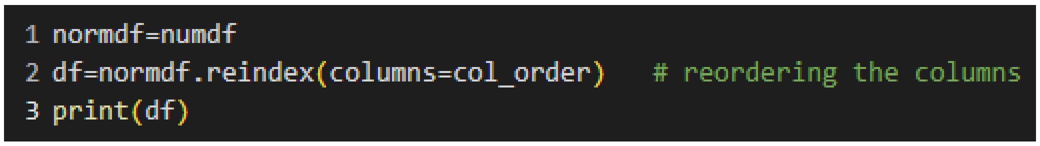
| | Rented Bike Count | Hour | Temperature | Humidity | Wind speed | Visibility | Dew point temperature | Solar Radiation | Rainfall | Snowfall | Date | Seasons | Holiday | Functioning Day |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0.071429 | 0.000000 | 0.220280 | 0.377551 | 0.297297 | 1.000000 | 0.224913 | 0.0 | 0.0 | 0.0 | 1/12/2017 | Winter | No Holiday | Yes |
| 1 | 0.057368 | 0.043478 | 0.215035 | 0.387755 | 0.108108 | 1.000000 | 0.224913 | 0.0 | 0.0 | 0.0 | 1/12/2017 | Winter | No Holiday | Yes |
| 2 | 0.048650 | 0.086957 | 0.206294 | 0.397959 | 0.135135 | 1.000000 | 0.223183 | 0.0 | 0.0 | 0.0 | 1/12/2017 | Winter | No Holiday | Yes |
| 3 | 0.030090 | 0.130435 | 0.202797 | 0.408163 | 0.121622 | 1.000000 | 0.224913 | 0.0 | 0.0 | 0.0 | 1/12/2017 | Winter | No Holiday | Yes |
| 4 | 0.021935 | 0.173913 | 0.206294 | 0.367347 | 0.310811 | 1.000000 | 0.207612 | 0.0 | 0.0 | 0.0 | 1/12/2017 | Winter | No Holiday | Yes |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 8755 | 0.282058 | 0.826087 | 0.384615 | 0.346939 | 0.351351 | 0.946275 | 0.351211 | 0.0 | 0.0 | 0.0 | 30/11/2018 | Autumn | No Holiday | Yes |
| 8756 | 0.214848 | 0.869565 | 0.370629 | 0.377551 | 0.310811 | 1.000000 | 0.358131 | 0.0 | 0.0 | 0.0 | 30/11/2018 | Autumn | No Holiday | Yes |
| 8757 | 0.195163 | 0.913043 | 0.356643 | 0.397959 | 0.040541 | 0.983781 | 0.358131 | 0.0 | 0.0 | 0.0 | 30/11/2018 | Autumn | No Holiday | Yes |
| 8758 | 0.200225 | 0.956522 | 0.347902 | 0.418367 | 0.135135 | 0.928535 | 0.359862 | 0.0 | 0.0 | 0.0 | 30/11/2018 | Autumn | No Holiday | Yes |
| 8759 | 0.164229 | 1.000000 | 0.344406 | 0.438776 | 0.175676 | 0.953877 | 0.368512 | 0.0 | 0.0 | 0.0 | 30/11/2018 | Autumn | No Holiday | Yes |

8760 rows × 14 columns

**Inferences:**

- **Unified Dataset:** Combines both numerical and categorical features into a single DataFrame (*numdf*) for comprehensive analysis.
- **Readiness for Modeling:** Ensures that all features (numerical and categorical) are available for preprocessing or machine learning workflows.

**What used:**

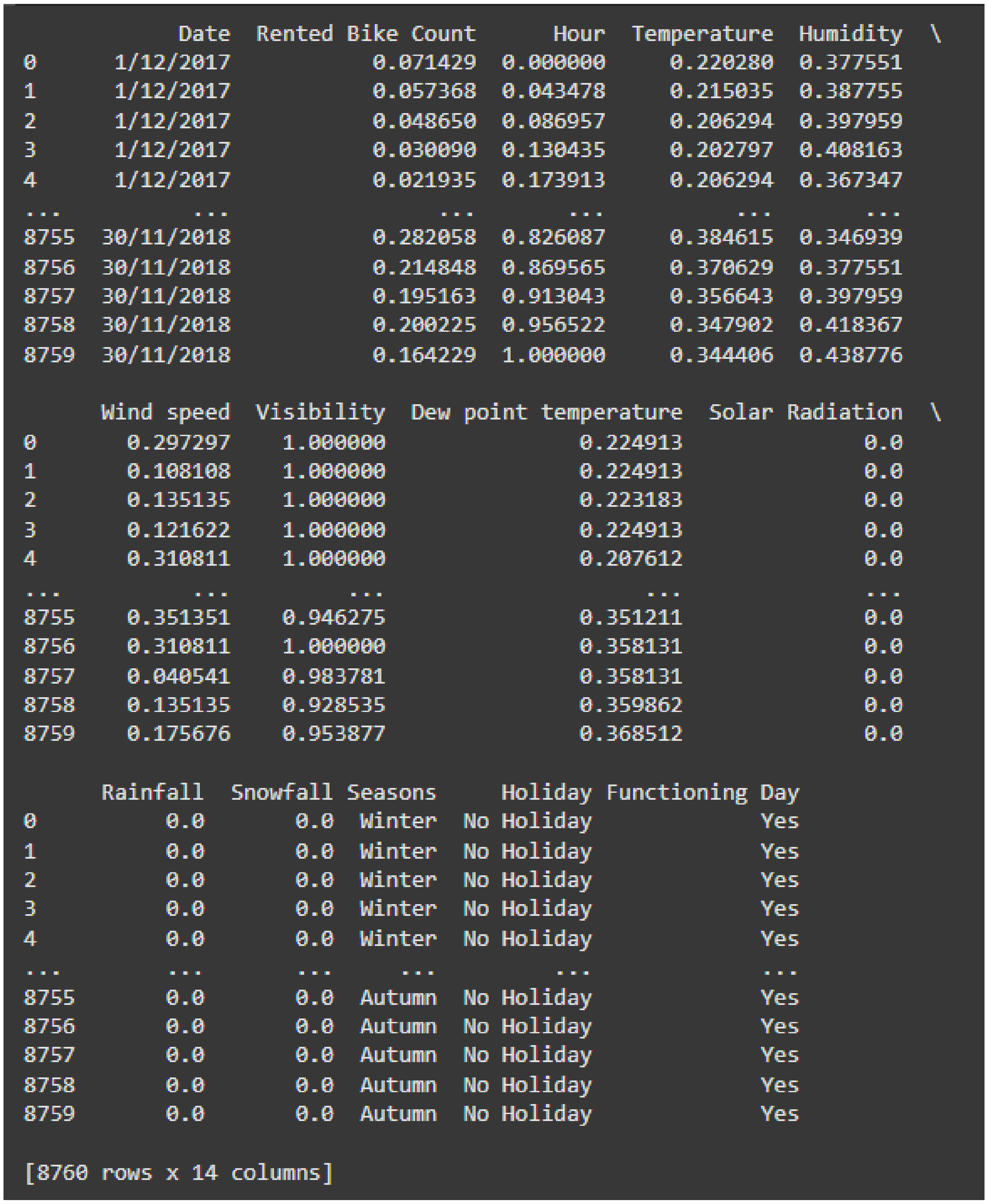


```python
normdf=numdf
df=normdf.reindex(columns=col_order)    # reordering the columns
print(df)
```

**Reason:**

- **Reorder Columns:** Restores the column order of the dataset (*df*) to its original arrangement as stored in *col_order*.

**Output:**

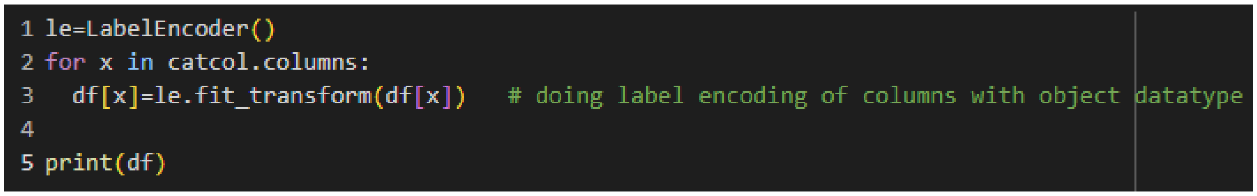


```
            Date  Rented Bike Count      Hour  Temperature  Humidity  \
0      1/12/2017           0.071429  0.000000     0.220280  0.377551   
1      1/12/2017           0.057368  0.043478     0.215035  0.387755   
2      1/12/2017           0.048650  0.086957     0.206294  0.397959   
3      1/12/2017           0.030090  0.130435     0.202797  0.408163   
4      1/12/2017           0.021935  0.173913     0.206294  0.367347   
...          ...                ...       ...          ...       ...   
8755  30/11/2018           0.282058  0.826087     0.384615  0.346939   
8756  30/11/2018           0.214848  0.869565     0.370629  0.377551   
8757  30/11/2018           0.195163  0.913043     0.356643  0.397959   
8758  30/11/2018           0.200225  0.956522     0.347902  0.418367   
8759  30/11/2018           0.164229  1.000000     0.344406  0.438776   

      Wind speed  Visibility  Dew point temperature  Solar Radiation  \
0       0.297297    1.000000               0.224913              0.0   
1       0.108108    1.000000               0.224913              0.0   
2       0.135135    1.000000               0.223183              0.0   
3       0.121622    1.000000               0.224913              0.0   
4       0.310811    1.000000               0.207612              0.0   
...          ...         ...                    ...              ...   
8755    0.351351    0.946275               0.351211              0.0   
8756    0.310811    1.000000               0.358131              0.0   
8757    0.040541    0.983781               0.358131              0.0   
8758    0.135135    0.928535               0.359862              0.0   
8759    0.175676    0.953877               0.368512              0.0   

      Rainfall  Snowfall Seasons     Holiday Functioning Day  
0          0.0       0.0  Winter  No Holiday             Yes  
1          0.0       0.0  Winter  No Holiday             Yes  
2          0.0       0.0  Winter  No Holiday             Yes  
3          0.0       0.0  Winter  No Holiday             Yes  
4          0.0       0.0  Winter  No Holiday             Yes  
...        ...       ...     ...         ...             ...  
8755       0.0       0.0  Autumn  No Holiday             Yes  
8756       0.0       0.0  Autumn  No Holiday             Yes  
8757       0.0       0.0  Autumn  No Holiday             Yes  
8758       0.0       0.0  Autumn  No Holiday             Yes  
8759       0.0       0.0  Autumn  No Holiday             Yes  

[8760 rows x 14 columns]
```

**Inferences:**

- **Consistency:** Ensures that the updated dataset (*df*) maintains the original column order, making it easier to compare with earlier stages or documentation.
- **Data Organization:** Keeps the dataset well-structured for analysis or modeling.

**What used:**

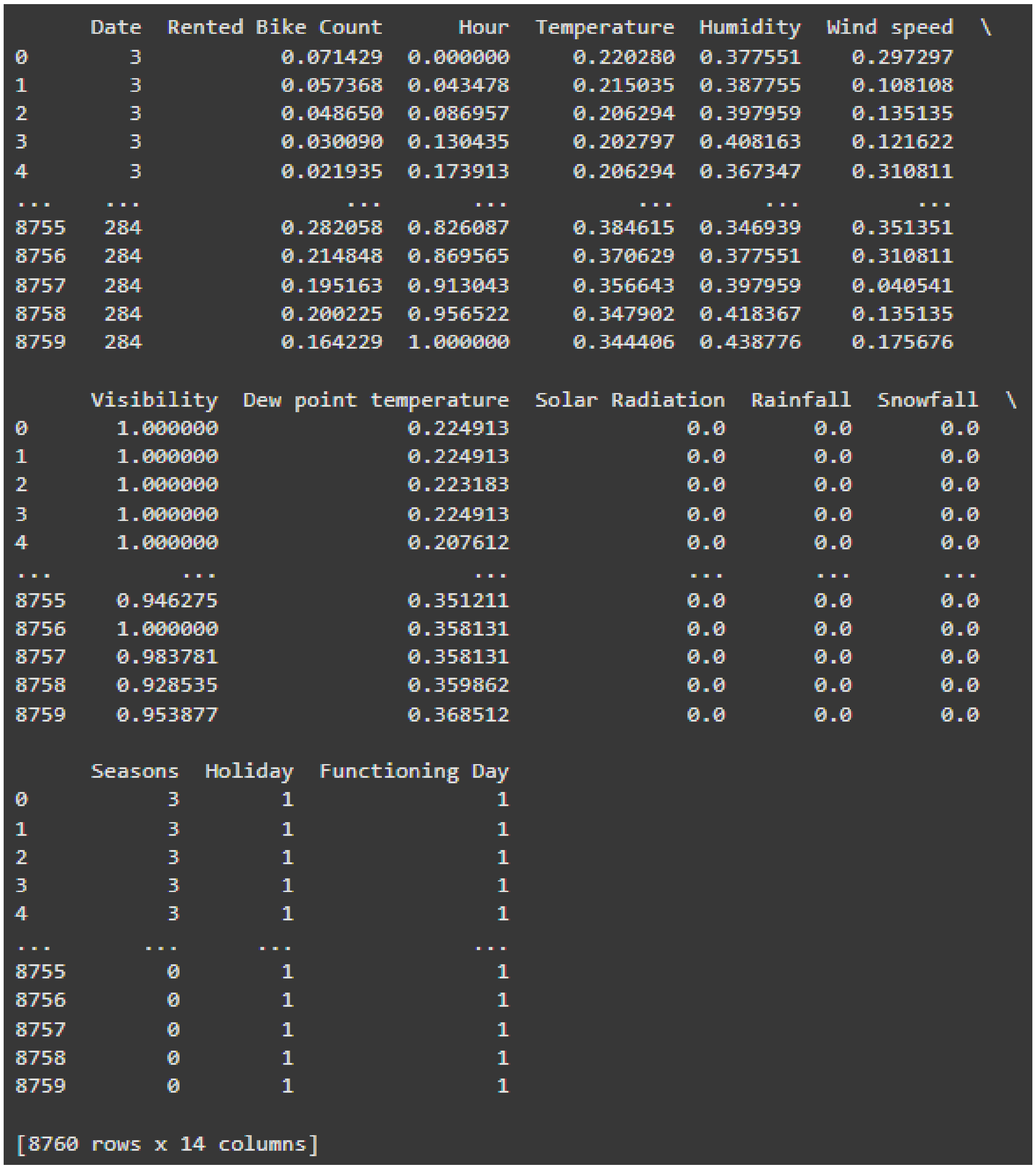


```
le=LabelEncoder()
for x in catcol.columns:
  df[x]=le.fit_transform(df[x])   # doing label encoding of columns with object datatype

print(df)
```

**Reason:**

- **Label Encoding:** Converts categorical columns into numerical format using LabelEncoder, making them compatible with machine learning algorithms.

**Output:**

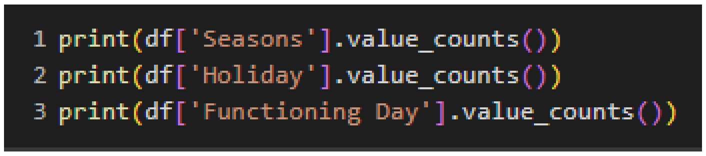


```
      Date  Rented Bike Count      Hour  Temperature  Humidity  Wind speed  \
0        3           0.071429  0.000000     0.220280  0.377551    0.297297   
1        3           0.057368  0.043478     0.215035  0.387755    0.108108   
2        3           0.048650  0.086957     0.206294  0.397959    0.135135   
3        3           0.030090  0.130435     0.202797  0.408163    0.121622   
4        3           0.021935  0.173913     0.206294  0.367347    0.310811   
...    ...                ...       ...          ...       ...         ...   
8755   284           0.282058  0.826087     0.384615  0.346939    0.351351   
8756   284           0.214848  0.869565     0.370629  0.377551    0.310811   
8757   284           0.195163  0.913043     0.356643  0.397959    0.040541   
8758   284           0.200225  0.956522     0.347902  0.418367    0.135135   
8759   284           0.164229  1.000000     0.344406  0.438776    0.175676   

      Visibility  Dew point temperature  Solar Radiation  Rainfall  Snowfall  \
0       1.000000               0.224913              0.0       0.0       0.0   
1       1.000000               0.224913              0.0       0.0       0.0   
2       1.000000               0.223183              0.0       0.0       0.0   
3       1.000000               0.224913              0.0       0.0       0.0   
4       1.000000               0.207612              0.0       0.0       0.0   
...          ...                    ...              ...       ...       ...   
8755    0.946275               0.351211              0.0       0.0       0.0   
8756    1.000000               0.358131              0.0       0.0       0.0   
8757    0.983781               0.358131              0.0       0.0       0.0   
8758    0.928535               0.359862              0.0       0.0       0.0   
8759    0.953877               0.368512              0.0       0.0       0.0   

      Seasons  Holiday  Functioning Day  
0           3        1                1  
1           3        1                1  
2           3        1                1  
3           3        1                1  
4           3        1                1  
...       ...      ...              ...  
8755        0        1                1  
8756        0        1                1  
8757        0        1                1  
8758        0        1                1  
8759        0        1                1  

[8760 rows x 14 columns]
```

**Inferences:**

- **Encoded Categorical Data:** Each unique category in the columns is replaced with a numerical value (e.g., 0, 1, 2).
- **Model Compatibility:** Transforms categorical data into a format suitable for algorithms that require numerical inputs.

**What used:**

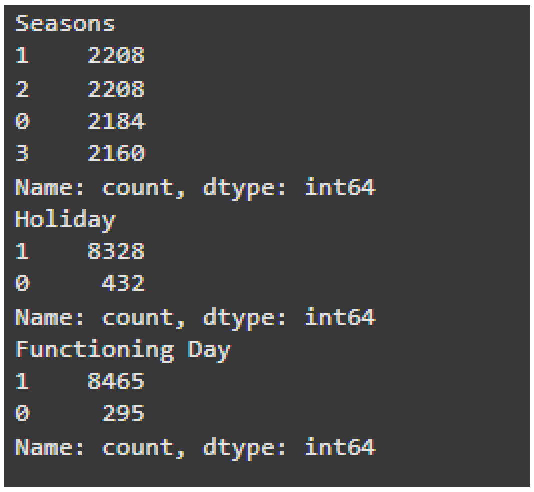


```python
print(df['Seasons'].value_counts())
print(df['Holiday'].value_counts())
print(df['Functioning Day'].value_counts())
```

**Reason:**

- **Analyze Encoded Data:** Displays the frequency counts for the encoded categorical columns (*Seasons, Holiday, and Functioning Day*) to understand their distributions.

**Output:**

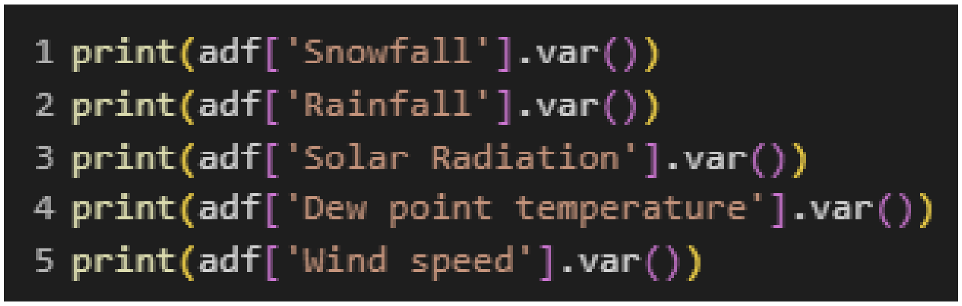


```
Seasons
1    2208
2    2208
0    2184
3    2160
Name: count, dtype: int64
Holiday
1    8328
0     432
Name: count, dtype: int64
Functioning Day
1    8465
0     295
Name: count, dtype: int64
```

**Inferences:**

- **Seasons:** Shows the frequency of each encoded season, helping identify which seasons are more prevalent in the dataset.
- **Holiday:** Displays the distribution of holiday and non-holiday entries.
- **Functioning Day:** Reveals the count of functioning versus non-functioning days.

**What used:**

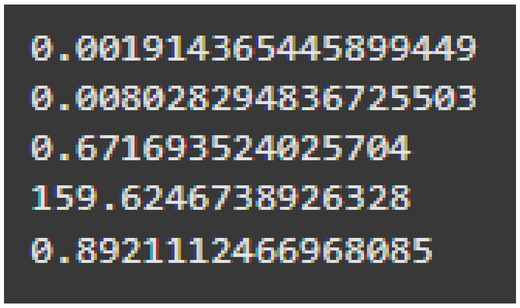


```python
print(adf['Snowfall'].var())
print(adf['Rainfall'].var())
print(adf['Solar Radiation'].var())
print(adf['Dew point temperature'].var())
print(adf['Wind speed'].var())
```

**Reason:**

- **Purpose:** To calculate and display the variance for specific numerical columns in the dataset (*adf*).
- **Variance:** Measures the spread or variability of data within each column. Higher variance indicates greater data dispersion.

**Output:**

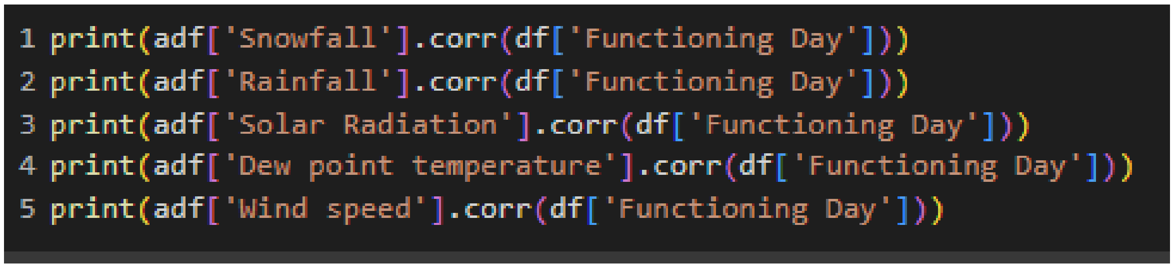


```
0.0019143654458994449
0.008028294836725503
0.671693524025704
159.6246738926328
0.892111246696808 5
```

**Inferences:**

- **Column Insights:**
  - A *high variance* (e.g., Dew point temperature) indicates wide fluctuations in data values.
  - A *low variance* (e.g., Rainfall) suggests data values are closely clustered around the mean.
- **Data Behavior:** Variance helps identify columns with significant variability, which may impact model training or feature scaling.
- **Comparison:** Provides a basis to assess which environmental factors (e.g., snowfall, rainfall) show more consistency over time.

**What used:**



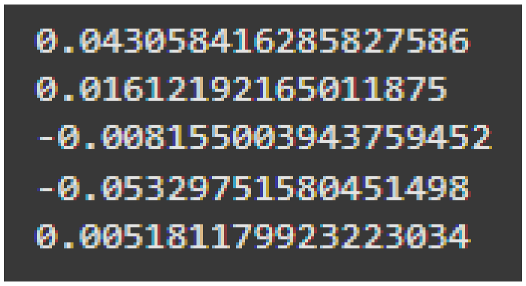
```python
print(adf['Snowfall'].corr(df['Functioning Day']))
print(adf['Rainfall'].corr(df['Functioning Day']))
print(adf['Solar Radiation'].corr(df['Functioning Day']))
print(adf['Dew point temperature'].corr(df['Functioning Day']))
print(adf['Wind speed'].corr(df['Functioning Day']))
```

**Reason:**

- **Purpose:** To compute the correlation between weather-related features (e.g., Snowfall, Rainfall) and the Functioning Day variable.

- **Correlation:** Indicates the strength and direction of the relationship between each feature and the target variable (*Functioning Day*), typically on a scale from -1 to 1.

**Output:**

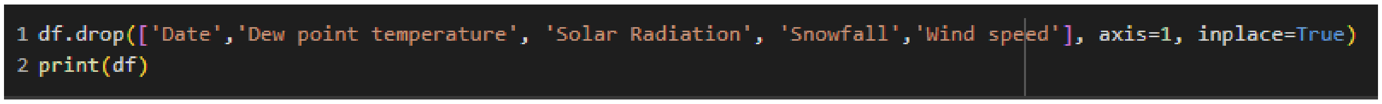


```
0.043058416285827586
0.01612192165011875
-0.008155003943759452
-0.05329751580451498
0.005181179923223034
```

**Inferences:**

- **Positive Correlation:** Values closer to 1 suggest that as the feature increases, the likelihood of a functioning day also increases.
- **Negative Correlation:** Values closer to -1 imply an inverse relationship, where higher feature values are associated with non-functioning days.
- **Weak/No Correlation:** Values near 0 indicate little to no linear relationship between the feature and the target.

**What used:**

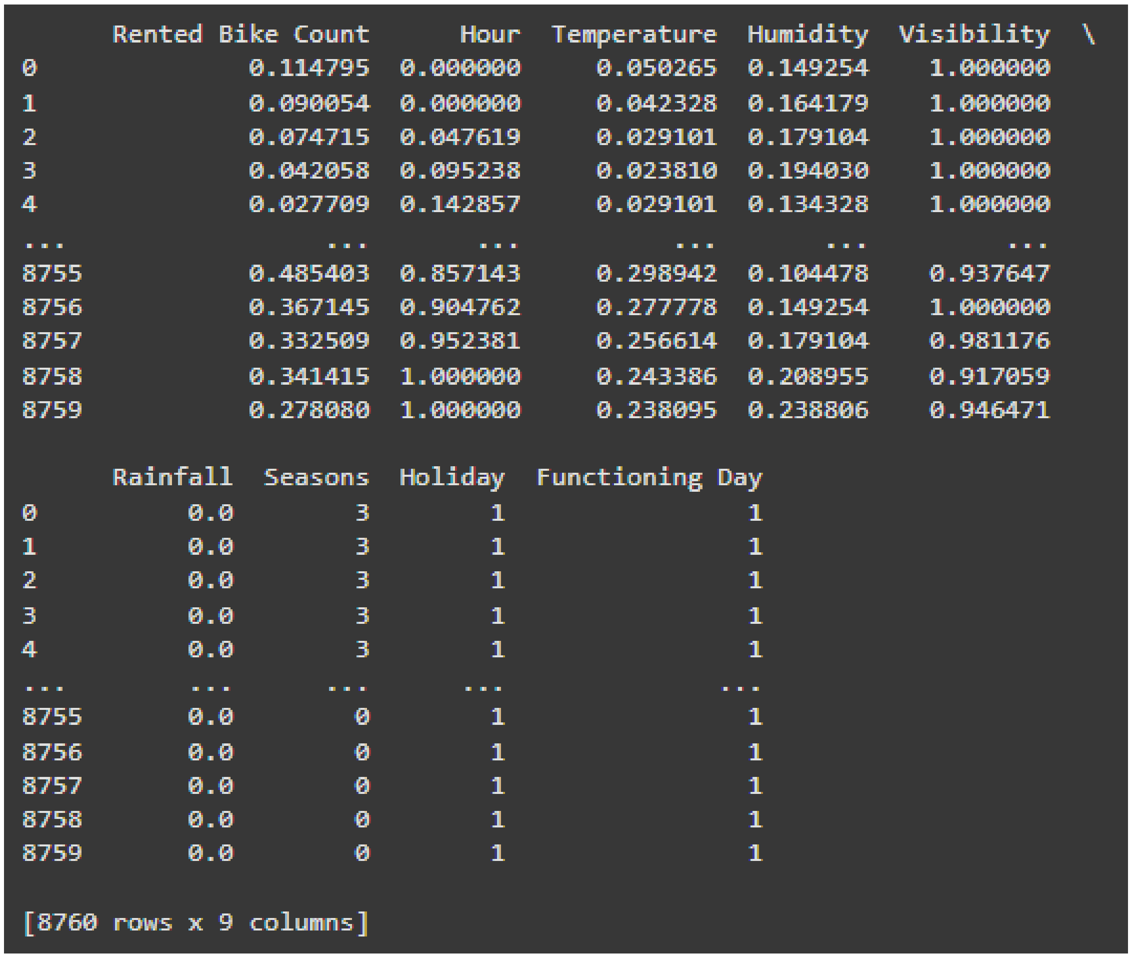


```
df.drop(['Date','Dew point temperature', 'Solar Radiation', 'Snowfall','Wind speed'], axis=1, inplace=True)
print(df)
```

**Reason:**

- **Purpose:** To remove non-essential or redundant features (*Date, Dew point temperature, Solar Radiation, Snowfall, Wind speed*) from the dataset df.
- **Feature Selection:** Simplifies the dataset by keeping only the columns relevant for the analysis or model, reducing computational complexity and avoiding noise.

**Output:**

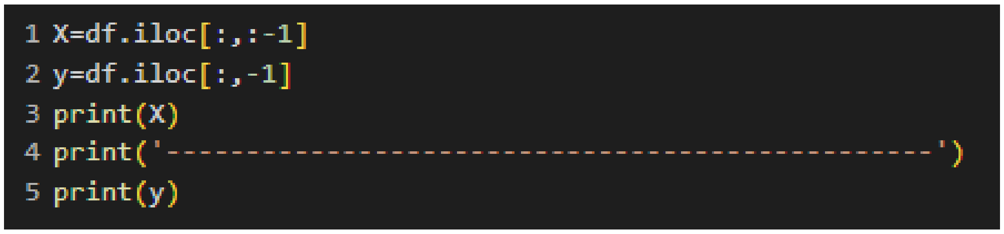


```
      Rented Bike Count      Hour  Temperature  Humidity  Visibility  \
0              0.114795  0.000000     0.050265  0.149254    1.000000
1              0.090054  0.000000     0.042328  0.164179    1.000000
2              0.074715  0.047619     0.029101  0.179104    1.000000
3              0.042058  0.095238     0.023810  0.194030    1.000000
4              0.027709  0.142857     0.029101  0.134328    1.000000
...                 ...       ...          ...       ...         ...
8755           0.485403  0.857143     0.298942  0.104478    0.937647
8756           0.367145  0.904762     0.277778  0.149254    1.000000
8757           0.332509  0.952381     0.256614  0.179104    0.981176
8758           0.341415  1.000000     0.243386  0.208955    0.917059
8759           0.278080  1.000000     0.238095  0.238806    0.946471

      Rainfall  Seasons  Holiday  Functioning Day
0          0.0        3        1                1
1          0.0        3        1                1
2          0.0        3        1                1
3          0.0        3        1                1
4          0.0        3        1                1
...        ...      ...      ...              ...
8755       0.0        0        1                1
8756       0.0        0        1                1
8757       0.0        0        1                1
8758       0.0        0        1                1
8759       0.0        0        1                1

[8760 rows x 9 columns]
```

**Inferences:**

- **Streamlined Dataset:** The remaining columns likely contain more relevant or impactful features for the task.
- **Impact of Dropping:**
  - Dropped columns may have low correlation with the target (*Functioning Day*) or might not provide significant predictive value.
  - This step can prevent overfitting by excluding irrelevant variables.

**What used:**

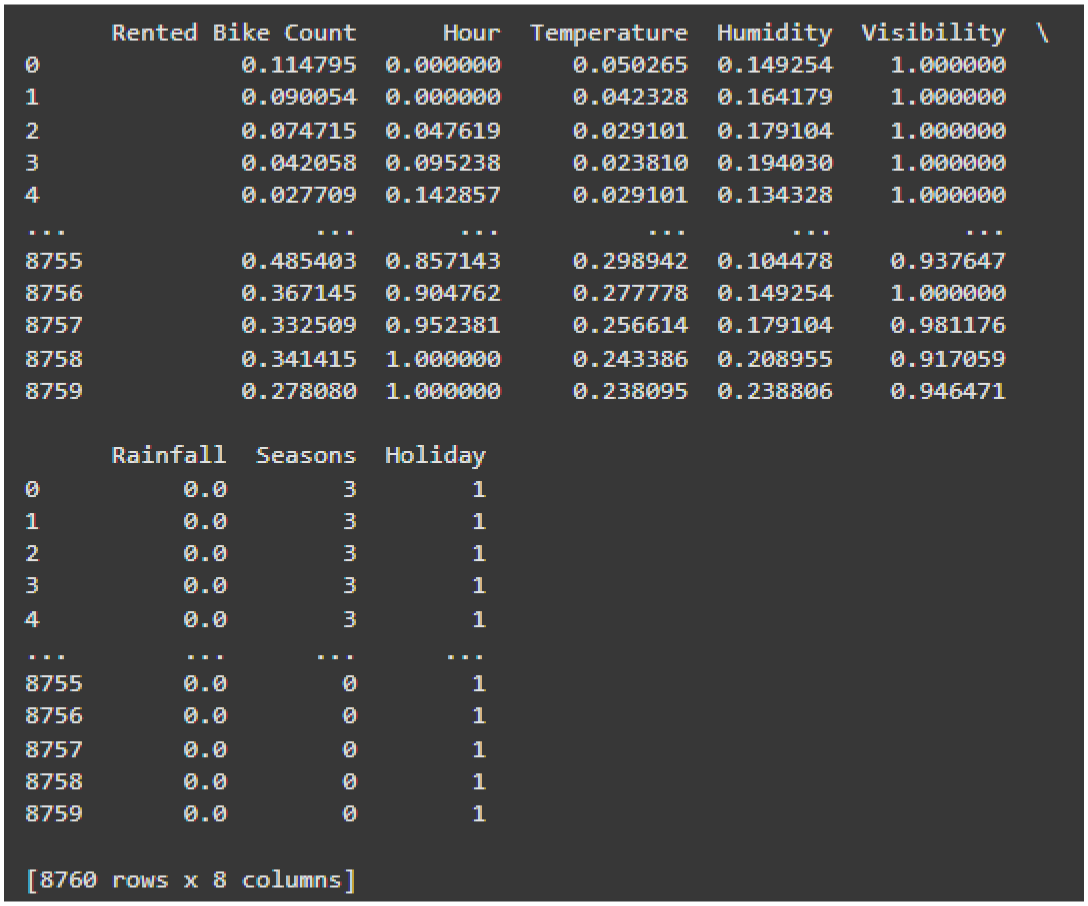


```
X=df.iloc[:,:-1]
y=df.iloc[:,-1]
print(X)
print('-----------------------------------------------')
print(y)
```

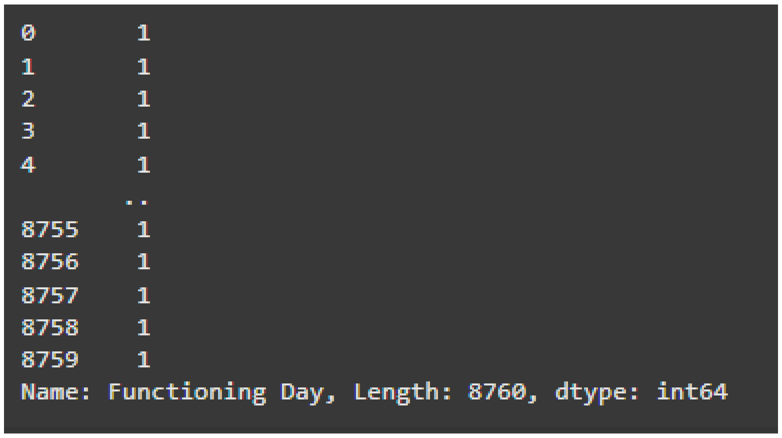
**Reason:**

- **Purpose**: To split the dataset df into features (*X*) and the target variable (*y*).
    - **X**: All columns except the last one, representing independent variables (features).
    - **y**: The last column, representing the dependent variable or target for prediction.

**Output:**



```
      Rented Bike Count      Hour  Temperature  Humidity  Visibility  \
0              0.114795  0.000000     0.050265  0.149254    1.000000   
1              0.090054  0.000000     0.042328  0.164179    1.000000   
2              0.074715  0.047619     0.029101  0.179104    1.000000   
3              0.042058  0.095238     0.023810  0.194030    1.000000   
4              0.027709  0.142857     0.029101  0.134328    1.000000   
...                 ...       ...          ...       ...         ...   
8755           0.485403  0.857143     0.298942  0.104478    0.937647   
8756           0.367145  0.904762     0.277778  0.149254    1.000000   
8757           0.332509  0.952381     0.256614  0.179104    0.981176   
8758           0.341415  1.000000     0.243386  0.208955    0.917059   
8759           0.278080  1.000000     0.238095  0.238806    0.946471   

      Rainfall  Seasons  Holiday  
0          0.0        3        1  
1          0.0        3        1  
2          0.0        3        1  
3          0.0        3        1  
4          0.0        3        1  
...        ...      ...      ...  
8755       0.0        0        1  
8756       0.0        0        1  
8757       0.0        0        1  
8758       0.0        0        1  
8759       0.0        0        1  

[8760 rows x 8 columns]
```

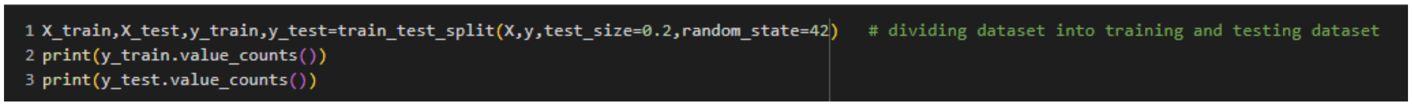


```
0       1
1       1
2       1
3       1
4       1
       ..
8755    1
8756    1
8757    1
8758    1
8759    1
Name: Functioning Day, Length: 8760, dtype: int64
```

**Inferences:**

- **Feature Set (X)**: Contains the predictors needed for modeling. Printing it allows you to verify the selected features.
- **Target (y)**: Includes the values the model aims to predict. Its structure and values can influence the choice of the model.
- **Dataset Preparedness**: Splitting ensures the data is ready for machine learning workflows, like training and evaluation.

**What used:**

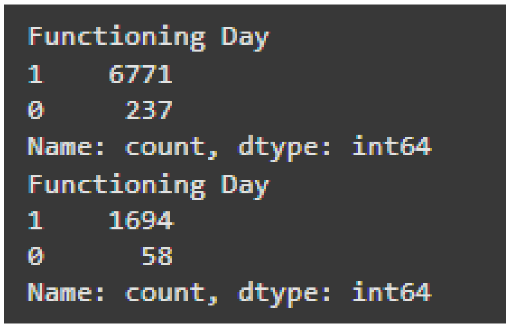


```
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2,random_state=42)   # dividing dataset into training and testing dataset
print(y_train.value_counts())
print(y_test.value_counts())
```

**Reason:**

- **Purpose**:
  - **train_test_split**: Divides the dataset into training and testing subsets, ensuring a model is trained on one portion and evaluated on another.
  - **test_size=0.2:** Allocates 20% of the data for testing and 80% for training.
  - **random_state=42:** Ensures reproducibility by setting a seed for the random split.
- **value_counts()**: Displays the distribution of the target variable (*y*) in both the training and testing sets to check for class imbalance.

**Output:**

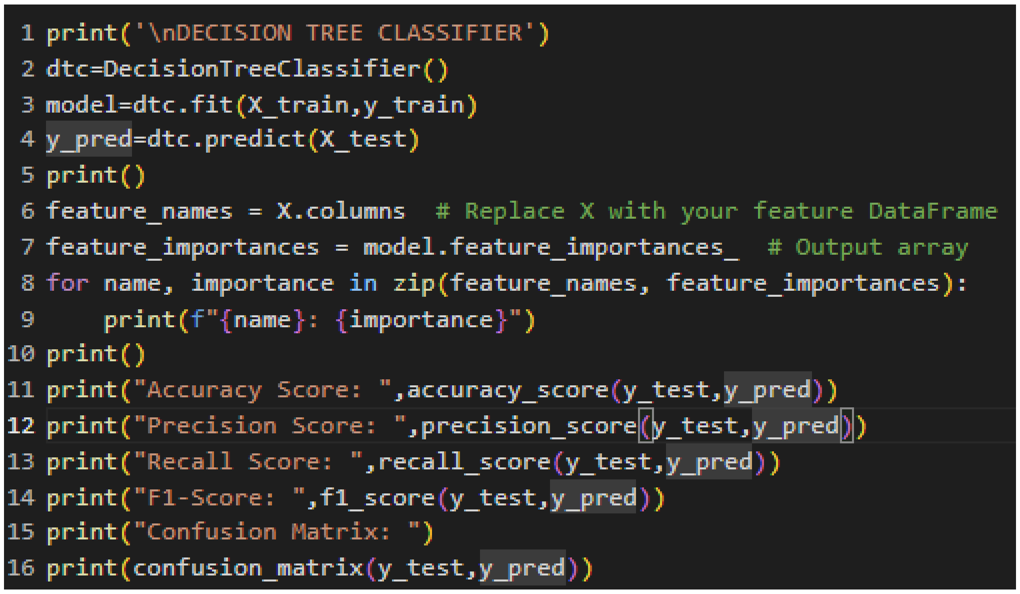


```
Functioning Day
1    6771
0     237
Name: count, dtype: int64
Functioning Day
1    1694
0      58
Name: count, dtype: int64
```

**Inferences:**

- **Class Distribution**:
  - Helps verify whether the split maintains the original class proportions.
  - Any significant imbalance may require techniques like oversampling, undersampling, or class weighting.
- **Training-Test Consistency**: Ensures both sets are representative of the overall dataset, critical for reliable model performance.

**What used:**

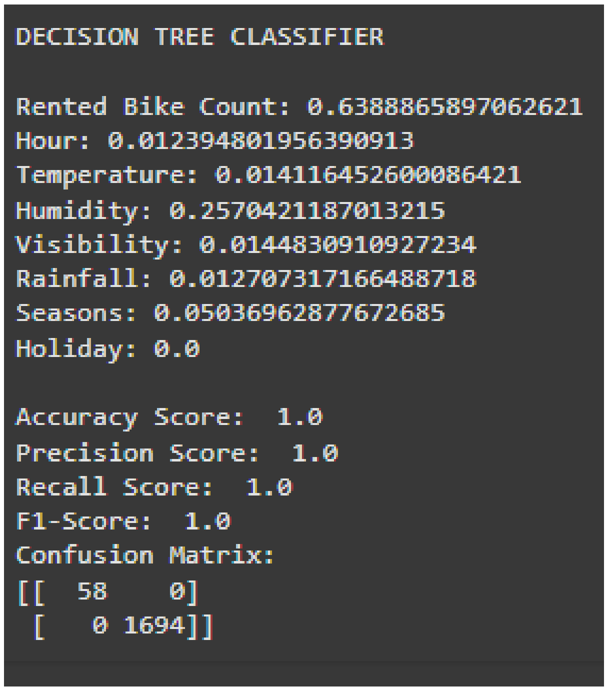


```
print('\nDECISION TREE CLASSIFIER')
dtc=DecisionTreeClassifier()
model=dtc.fit(X_train,y_train)
y_pred=dtc.predict(X_test)
print()
feature_names = X.columns  # Replace X with your feature DataFrame
feature_importances = model.feature_importances_  # Output array
for name, importance in zip(feature_names, feature_importances):
    print(f"{name}: {importance}")
print()
print("Accuracy Score: ",accuracy_score(y_test,y_pred))
print("Precision Score: ",precision_score(y_test,y_pred))
print("Recall Score: ",recall_score(y_test,y_pred))
print("F1-Score: ",f1_score(y_test,y_pred))
print("Confusion Matrix: ")
print(confusion_matrix(y_test,y_pred))
```

**Reason:**

- **Model Training:**
    - **DecisionTreeClassifier():** Implements a decision tree algorithm for classification tasks.
    - **fit(X_train, y_train)**: Trains the model using the training dataset.
- **Feature Importance**:Extracts and displays the importance of each feature in the model's decision-making process. Helps identify the most impactful features.
- **Model Evaluation**:
    - **accuracy_score**: Overall proportion of correct predictions.
    - **precision_score**: Proportion of true positive predictions among all positive predictions.
    - **recall_score**: Proportion of true positives correctly identified.
    - **f1_score**: Harmonic mean of precision and recall, balancing both metrics.
    - **Confusion Matrix**: Provides detailed insight into prediction accuracy, with counts of true positives, true negatives, false positives, and false negatives.

**Output:**



```
DECISION TREE CLASSIFIER

Rented Bike Count: 0.6388865897062621
Hour: 0.012394801956390913
Temperature: 0.014116452600086421
Humidity: 0.2570421187013215
Visibility: 0.0144830910927234
Rainfall: 0.012707317166488718
Seasons: 0.05036962877672685
Holiday: 0.0

Accuracy Score:  1.0
Precision Score:  1.0
Recall Score:  1.0
F1-Score:  1.0
Confusion Matrix:
[[  58    0]
 [   0 1694]]
```

**Inferences:**

- **Feature Importance**: Displays which features contribute most to the model, enabling data-driven decisions about feature selection.
- **Performance Metrics**:
  - High accuracy, precision, recall, or F1-score indicates effective classification.
  - Imbalance in precision and recall might suggest improvements are needed for specific class predictions.
- **Error Analysis**:The confusion matrix identifies areas where the model misclassifies, such as false positives or negatives.

**Why Preprocessing?**

- **Improves Data Quality:** Fixes errors, missing values, and inconsistencies.
- **Ensures Model Accuracy:** Prepares data in a format suitable for algorithms.
- **Enhances Model Performance:** By scaling, encoding, and engineering features.
- **Addresses Class Imbalance:** Balances data to avoid bias.

**How Preprocessing?**

- **Missing Data:** Impute (mean, median) or remove missing values.
- **Outliers:** Cap extreme values (winsorization) or remove them.
- **Encoding:** Convert categorical data using label or one-hot encoding.
- **Scaling:** Normalize or standardize features for consistency.

- **Feature Selection:** Remove irrelevant features or combine them for better performance.

**What Preprocessing?**
- **Cleaning:** Fixing errors and missing values.
- **Transforming:** Encoding and scaling features.

## Basic Questions About the Data

1. **What is the distribution of the target variable?**
    - **Why:** Understanding the distribution helps identify if the data is skewed or imbalanced, which can impact model performance.
    - **How:** We'll visualize the distribution using histograms or box plots and check for skewness.
2. **Are there missing values in the dataset?**
    - **Why:** Missing data can lead to biased models or reduced accuracy. Identifying and handling it is crucial for reliable model building.
    - **How:** We'll check for missing values across all columns and use imputation or removal strategies.
3. **Are there outliers in numerical features (e.g., Temperature, Rainfall)?**
    - **Why:** Outliers can distort model training, leading to overfitting or poor generalization.
    - **How:** We will use box plots and statistical methods like winsorization to identify and mitigate outliers.
4. **What are the correlations between features and the target variable?**
    - **Why:** Understanding which features are most strongly related to the target helps in feature selection and model building.
    - **How:** We'll compute correlation coefficients (Pearson's or Spearman's) to assess the relationships.
5. **Is there a relationship between weather features (e.g., Temperature, Wind Speed) and bike rentals?**
    - **Why:** Weather factors likely influence bike rentals, so identifying strong predictors will improve the model's predictive power.
    - **How:** We'll explore scatter plots, correlations, and feature importance techniques.
6. **Are the categorical features (Seasons, Holiday, Functioning Day) properly encoded?**
    - **Why:** Many machine learning algorithms require categorical data to be in a numerical format.

   - **How:** We'll use label encoding or one-hot encoding and check for any inconsistencies.
7. **Is the dataset imbalanced with respect to the target variable?**
   - **Why:** Imbalanced datasets can lead to biased predictions, especially for minority classes.
   - **How:** We'll analyze the class distribution and consider techniques like oversampling or undersampling if necessary.
8. **What impact do weather conditions (e.g., Solar Radiation, Dew point temperature) have on bike rentals?**
   - **Why:** Weather conditions are likely to affect the number of bike rentals, and understanding these relationships can guide feature selection.
   - **How:** We'll analyze correlations, plot relationships, and check feature importance after model training.

## How We Address These Questions Through Analysis

1. **Data Distribution:** Visualize with histograms and box plots to check for skewness, imbalances, and outliers. This helps understand the spread and identify data issues like imbalances.
2. **Missing Values:** Use *.isnull()* or *.info()* to detect missing data. Depending on the amount and pattern, impute (mean/median) or drop the missing values.
3. **Outliers:** Visualize with box plots and use winsorization to cap extreme values at specific percentiles (e.g., 5%). This reduces the impact of extreme values on model performance.
4. **Correlation:** Use correlation matrices (heatmaps) to identify relationships between features and the target. Strong correlations guide feature selection for models.
5. **Feature Relationships:** Explore scatter plots and correlations for numerical features and use feature importance techniques in models (e.g., Decision Trees) to determine relevant predictors.
6. **Encoding Categorical Data:** Apply Label Encoding or One-Hot Encoding for categorical variables to ensure the data is in a format suitable for machine learning algorithms.
7. **Class Imbalance:** Analyze class distributions with *.value_counts()* to balance classes if needed.
8. **Feature Impact:** After training models (e.g., Decision Trees), evaluate feature importance scores to see which weather-related features most affect bike rentals.

Through these analyses, we can understand data patterns, clean and transform the data effectively, and prepare it for building accurate machine learning models.

**Reasons for choosing Decision Tree Classifier:**

- **Interpretability:** The tree structure is easy to understand and visualize, making it ideal for explaining model decisions.
- **Non-linearity:** Can model complex, non-linear relationships without needing feature transformations.
- **Handles Mixed Data:** Works well with both numerical and categorical data without the need for extensive preprocessing.
- **No Scaling Required:** Doesn’t require feature scaling, unlike models like SVM or k-NN.
- **Handling Missing Data:** Can handle missing values naturally, unlike many other algorithms.
- **Flexibility:** Easily tunable for better performance, with options like pruning to prevent overfitting.
- **Robust to Outliers:** Less sensitive to outliers compared to models like k-NN or linear classifiers.