A1-template

March 29, 2022

1 Using Machine Learning Tools Assignment 1

1.1 Overview

In this assignment, you will apply some popular machine learning techniques to the problem of predicting bike rental demand. A data set has been provided containing records of bike rentals in Seoul, collected during 2017-18.

The main aims of the prac are:

- to practice using tools for loading and viewing data sets;
- to visualise data in several ways and check for common pitfalls;
- to plan a simple experiment and prepare the data accordingly;
- to run your experiment and to report and interpret your results clearly and concisely.

This assignment relates to the following ACS CBOK areas: abstraction, design, hardware and software, data and information, HCI and programming.

1.2 General instructions

This assignment is divided into several tasks. Use the spaces provided in this notebook to answer the questions posed in each task. Note that some questions require writing a small amount of code, some require graphical results, and some require comments or analysis as text. It is your responsibility to make sure your responses are clearly labelled and your code has been fully executed (with the correct results displayed) before submission!

Do not manually edit the data set file we have provided! For marking purposes, it's important that your code is written to run correctly on the original data file.

When creating graphical output, label is clearly, with appropriate titles, xlabels and ylabels, as appropriate.

Most of the tasks in this assignment only require writing a few lines of code! One goal of the assignment is explore sklearn, pandas, matplotlib and other libraries you will find useful throughout the course, so feel free to use the functions they provide. You are expected to search and carefully read the documentation for functions that you use, to ensure you are using them correctly.

Chapter 2 of the reference book is based on a similar workflow to this prac, so you may look there for some further background and ideas. You can also use any other general resources on the internet that are relevant although do not use ones which directly relate to these questions with this dataset (which would normally only be found in someone else's assignment answers). If you take a large portion of code or text from the internet then you should reference where this was taken from, but we do not expect any references for small pieces of code, such as from documentation, blogs or tutorials. Taking, and adapting, small portions of code is expected and is common practice when solving real problems.

The following code imports some of the essential libraries that you will need. You should not need to modify it, but you are expected to import other libraries as needed.

[57]:
```
# Python 3.5 is required
import sys
assert sys.version_info >= (3, 5)

import sklearn
assert sklearn.__version__ >= "0.20"

import pandas as pd
assert pd.__version__ >= "1.0"

# Common imports
import numpy as np
import os

# To plot pretty figures
%matplotlib inline
import matplotlib as mpl
import matplotlib.pyplot as plt
mpl.rc('axes', labelsize=14)
mpl.rc('xtick', labelsize=12)
mpl.rc('ytick', labelsize=12)
```

1.3 Step 1: Loading and initial processing of the dataset (20%)

Download the data set from MyUni using the link provided on the assignment page. A paper that describes one related version of this dataset is: Sathishkumar V E, Jangwoo Park, and Yongyun Cho. 'Using data mining techniques for bike sharing demand prediction in metropolitan city.' Computer Communications, Vol.153, pp.353-366, March, 2020. Feel free to look at this if you want more information about the dataset.

The data is stored in a CSV (comma separated variable) file and contains the following information

- Date: year-month-day
- Rented Bike Count: Count of bikes rented at each hour
- Hour: Hour of the day
- Temperature: Temperature in Celsius
- Humidity: %
- Windspeed: m/s
- Visibility: 10m
- Dew point temperature: Celsius
- Solar radiation: MJ/m2
- Rainfall: mm

- Snowfall: cm
- Seasons: Winter, Spring, Summer, Autumn
- Holiday: Holiday/No holiday
- Functional Day: NoFunc(Non Functional Hours), Fun(Functional hours)

Load the data set from the csv file into a DataFrame, and summarise it with at least two appropriate pandas functions.

[58]:
```
### Your code here
##load data  from csv
rentals = pd.read_csv("SeoulBikeData.csv")
```

[59]:
```
##summarise
rentals.head()
```

[59]:
```
         Date  Rented Bike Count  Hour  Temperature (C)  Humidity (%)  \
0  01/12/2017                254     0             -5.2            37
1  01/12/2017                204     1             -5.5            38
2  01/12/2017                173     2             -6.0            39
3  01/12/2017                107     3             -6.2            40
4  01/12/2017                 78     4             -6.0            36

   Wind speed (m/s)  Visibility (10m)  Dew point temperature (C)  \
0               2.2              2000                      -17.6
1               0.8              2000                      -17.6
2               1.0              2000                      -17.7
3               0.9              2000                      -17.6
4               2.3              2000                      -18.6

   Solar Radiation (MJ/m2) Rainfall(mm) Snowfall (cm) Seasons     Holiday  \
0                      0.0            0             0  Winter  No Holiday
1                      0.0            0             0  Winter  No Holiday
2                      0.0            0             0  Winter  No Holiday
3                      0.0            0             0  Winter  No Holiday
4                      0.0            0             0  Winter  No Holiday

  Functioning Day
0             Yes
1             Yes
2             Yes
3             Yes
4             Yes
```

[60]:
```
rentals.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)           8759 non-null   float64
 6   Visibility (10m)           8760 non-null   int64
 7   Dew point temperature (C)  8759 non-null   float64
 8   Solar Radiation (MJ/m2)    8760 non-null   float64
 9   Rainfall(mm)               8758 non-null   object
 10  Snowfall (cm)              8760 non-null   object
 11  Seasons                    8760 non-null   object
 12  Holiday                    8760 non-null   object
 13  Functioning Day            8760 non-null   object
dtypes: float64(4), int64(4), object(6)
memory usage: 958.2+ KB
```

[61]:
```
rentals.describe()
```

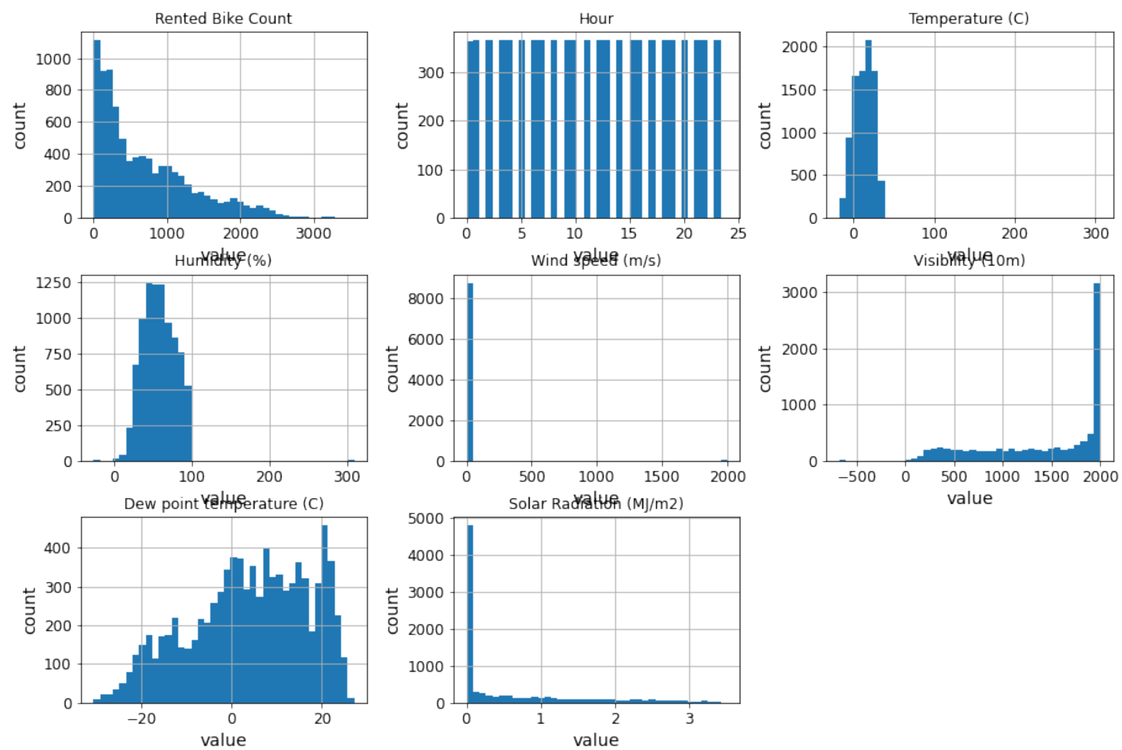
[61]:
```
       Rented Bike Count         Hour  Temperature (C)  Humidity (%)  \
count        8760.000000  8760.000000      8760.000000   8760.000000
mean          704.602055    11.502740        12.914361     58.240183
std           644.997468     6.922779        12.347109     20.584774
min             0.000000     0.000000       -17.800000    -26.000000
25%           191.000000     6.000000         3.500000     42.000000
50%           504.500000    12.000000        13.700000     57.000000
75%          1065.250000    18.000000        22.500000     74.000000
max          3556.000000    24.000000       306.000000    309.000000

       Wind speed (m/s)  Visibility (10m)  Dew point temperature (C)  \
count       8759.000000       8760.000000                8759.000000
mean           1.953237       1436.442808                   4.074369
std           21.376612        608.827735                  13.061011
min            0.000000       -678.000000                 -30.600000
25%            0.900000        939.500000                  -4.700000
50%            1.500000       1697.500000                   5.100000
75%            2.300000       2000.000000                  14.800000
max         2000.000000       2000.000000                  27.200000

       Solar Radiation (MJ/m2)
count              8760.000000
mean                  0.569111
std                   0.868746
min                   0.000000
25%                   0.000000
```

```
50%                       0.010000
75%                       0.930000
max                       3.520000
```

1.3.1 1.2 Initial visualisation

To get a feeling for the data it is a good idea to do some form of simple visualisation. **Display a set of histograms for the features** as they are right now, prior to any cleaning steps.

[62]:
```python
### Your code here
# use pd.hist to display the histograms
rental_hists = rentals.hist(bins=40,figsize=(15,10))

for rental_hist in rental_hists.flatten():
    rental_hist.set_xlabel("value")
    rental_hist.set_ylabel("count")

#this can only display the histogram for numberic values, after cleaning data,␣
↪it will show more hists
```



1.3.2 1.3 Removing unwanted information

The "Functioning day" feature records whether the bike rental was open for business on that day. For this assignment we are only interested in predicting demand on days when the business is open, so **remove rows from the DataFrame where the business is closed.** Hint: you can use the `DataFrame.loc()` function to do this. As a sanity check, ensure that the rows you are removing contain zero bike rentals! **After doing this, delete the Functioning Day feature from the DataFrame** and verify that this worked.

[63]:
```python
### Your code here

print(np.sum(rentals.isna() ))
#take the 13th column out
functioning_days = rentals.iloc[:,13]
#if functioning day == 'No', that's closed day
closed_days = functioning_days == 'No'
#set all the closed day as nan
rentals.iloc[closed_days,13] = np.nan
print("\nbefore replacing")
print("---------------------------------------------------------------------------------------
print("after replacing\n")

print(np.sum(rentals.isna() ))#we see we have already setted 295 closing_days
 ↪to nan
```

```
Date                         0
Rented Bike Count            0
Hour                         0
Temperature (C)              0
Humidity (%)                 0
Wind speed (m/s)             1
Visibility (10m)             0
Dew point temperature (C)    1
Solar Radiation (MJ/m2)      0
Rainfall(mm)                 2
Snowfall (cm)                0
Seasons                      0
Holiday                      0
Functioning Day              0
dtype: int64

before replacing
--------------------------------------------------------------------------------
-------------------------------
after replacing

Date                           0
Rented Bike Count              0
Hour                           0
Temperature (C)                0
Humidity (%)                   0
Wind speed (m/s)               1
```

```
Visibility (10m)              0
Dew point temperature (C)     1
Solar Radiation (MJ/m2)       0
Rainfall(mm)                  2
Snowfall (cm)                 0
Seasons                       0
Holiday                       0
Functioning Day             295
dtype: int64
```

[64]:
```python
print("Before Drop")
rentals.info()
#drop the closed day rows
rentals.dropna(subset=['Functioning Day'],inplace = True)
print("\nbefore dropping")
print("-------------------------------------------------------------------------------------------------------------
print("after dropping\n")
rentals.info() # we can see that rows with nan value has been dropped
```

```
Before Drop
<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)           8759 non-null   float64
 6   Visibility (10m)           8760 non-null   int64
 7   Dew point temperature (C)  8759 non-null   float64
 8   Solar Radiation (MJ/m2)    8760 non-null   float64
 9   Rainfall(mm)               8758 non-null   object
 10  Snowfall (cm)              8760 non-null   object
 11  Seasons                    8760 non-null   object
 12  Holiday                    8760 non-null   object
 13  Functioning Day            8465 non-null   object
dtypes: float64(4), int64(4), object(6)
memory usage: 958.2+ KB

before dropping
--------------------------------------------------------------------------------
-------------------------------
after dropping

<class 'pandas.core.frame.DataFrame'>
```

```
Int64Index: 8465 entries, 0 to 8759
Data columns (total 14 columns):
 #   Column                     Non-Null Count  Dtype
---  ------                     --------------  -----
 0   Date                       8465 non-null   object
 1   Rented Bike Count          8465 non-null   int64
 2   Hour                       8465 non-null   int64
 3   Temperature (C)            8465 non-null   float64
 4   Humidity (%)               8465 non-null   int64
 5   Wind speed (m/s)           8464 non-null   float64
 6   Visibility (10m)           8465 non-null   int64
 7   Dew point temperature (C)  8464 non-null   float64
 8   Solar Radiation (MJ/m2)    8465 non-null   float64
 9   Rainfall(mm)               8463 non-null   object
 10  Snowfall (cm)              8465 non-null   object
 11  Seasons                    8465 non-null   object
 12  Holiday                    8465 non-null   object
 13  Functioning Day            8465 non-null   object
dtypes: float64(4), int64(4), object(6)
memory usage: 992.0+ KB
```

[65]:

```
#drop the functioning day column
rentals.drop(columns=['Functioning Day'],inplace=True)
rentals.info() # we can see that the Functioning Day col has been dropped
rentals.head()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 8465 entries, 0 to 8759
Data columns (total 13 columns):
 #   Column                     Non-Null Count  Dtype
---  ------                     --------------  -----
 0   Date                       8465 non-null   object
 1   Rented Bike Count          8465 non-null   int64
 2   Hour                       8465 non-null   int64
 3   Temperature (C)            8465 non-null   float64
 4   Humidity (%)               8465 non-null   int64
 5   Wind speed (m/s)           8464 non-null   float64
 6   Visibility (10m)           8465 non-null   int64
 7   Dew point temperature (C)  8464 non-null   float64
 8   Solar Radiation (MJ/m2)    8465 non-null   float64
 9   Rainfall(mm)               8463 non-null   object
 10  Snowfall (cm)              8465 non-null   object
 11  Seasons                    8465 non-null   object
 12  Holiday                    8465 non-null   object
dtypes: float64(4), int64(4), object(5)
memory usage: 925.9+ KB
```

[65]:
```
         Date  Rented Bike Count  Hour  Temperature (C)  Humidity (%)  \
0  01/12/2017                254     0             -5.2            37
1  01/12/2017                204     1             -5.5            38
2  01/12/2017                173     2             -6.0            39
3  01/12/2017                107     3             -6.2            40
4  01/12/2017                 78     4             -6.0            36

   Wind speed (m/s)  Visibility (10m)  Dew point temperature (C)  \
0               2.2              2000                      -17.6
1               0.8              2000                      -17.6
2               1.0              2000                      -17.7
3               0.9              2000                      -17.6
4               2.3              2000                      -18.6

   Solar Radiation (MJ/m2) Rainfall(mm) Snowfall (cm) Seasons     Holiday
0                      0.0            0             0  Winter  No Holiday
1                      0.0            0             0  Winter  No Holiday
2                      0.0            0             0  Winter  No Holiday
3                      0.0            0             0  Winter  No Holiday
4                      0.0            0             0  Winter  No Holiday
```

1.3.3 1.4 Numerical encoding

The main task is to predict future bike rental demand from this data. Hence the target feature is "Bike Rental Count". You will use regression techniques to do this, but this requires that the other features are numerical.

The Holiday and Season features both need to be converted to a simple numerical format. **Write code to convert the Holiday feature to 0 or 1 from its current format.**

[66]:
```python
### Your code here
# No Holiday -> 0   Holiday -> 1

#take the 12th column out
holiday_col = rentals.iloc[:,12]
print(holiday_col)

#set Holiday to 1 and No Holiday to 0
holidays = holiday_col == 'Holiday'
rentals.iloc[holidays,12] = 1

holidays = holiday_col == 'No Holiday'
rentals.iloc[holidays,12] = 0

print("\nbefore replacing")
print("-----------------------------------------------------------------------------------
print("after replacing\n")
```

```python
print(holiday_col)#we see we holiday has been replaced by '0' and '1'

# make Holiday col numeric
rentals["Holiday"] = rentals["Holiday"].apply(pd.to_numeric,errors="coerce")
rentals.info() # we see the Holiday column become numeric
rentals.head()
```

```
0       No Holiday
1       No Holiday
2       No Holiday
3       No Holiday
4       No Holiday
           …
8755    No Holiday
8756    No Holiday
8757    No Holiday
8758    No Holiday
8759    No Holiday
Name: Holiday, Length: 8465, dtype: object

before replacing
--------------------------------------------------------------------------------
-------------------------------
after replacing

0       0
1       0
2       0
3       0
4       0
       ..
8755    0
8756    0
8757    0
8758    0
8759    0
Name: Holiday, Length: 8465, dtype: object
<class 'pandas.core.frame.DataFrame'>
Int64Index: 8465 entries, 0 to 8759
Data columns (total 13 columns):
 #   Column                     Non-Null Count  Dtype
---  ------                     --------------  -----
 0   Date                       8465 non-null   object
 1   Rented Bike Count          8465 non-null   int64
 2   Hour                       8465 non-null   int64
 3   Temperature (C)            8465 non-null   float64
 4   Humidity (%)               8465 non-null   int64
```

```
 5   Wind speed (m/s)          8464 non-null   float64
 6   Visibility (10m)          8465 non-null   int64
 7   Dew point temperature (C) 8464 non-null   float64
 8   Solar Radiation (MJ/m2)   8465 non-null   float64
 9   Rainfall(mm)              8463 non-null   object
 10  Snowfall (cm)             8465 non-null   object
 11  Seasons                   8465 non-null   object
 12  Holiday                   8465 non-null   int64
dtypes: float64(4), int64(5), object(4)
memory usage: 925.9+ KB
```

[66]:

```
         Date  Rented Bike Count  Hour  Temperature (C)  Humidity (%)  \
0  01/12/2017                254     0             -5.2            37
1  01/12/2017                204     1             -5.5            38
2  01/12/2017                173     2             -6.0            39
3  01/12/2017                107     3             -6.2            40
4  01/12/2017                 78     4             -6.0            36

   Wind speed (m/s)  Visibility (10m)  Dew point temperature (C)  \
0               2.2              2000                      -17.6
1               0.8              2000                      -17.6
2               1.0              2000                      -17.7
3               0.9              2000                      -17.6
4               2.3              2000                      -18.6

   Solar Radiation (MJ/m2) Rainfall(mm) Snowfall (cm) Seasons  Holiday
0                      0.0            0             0  Winter        0
1                      0.0            0             0  Winter        0
2                      0.0            0             0  Winter        0
3                      0.0            0             0  Winter        0
4                      0.0            0             0  Winter        0
```

The Season feature is a little tricker. A number could be assigned to each season, but a better solution in this case is to **add 4 new columns**, each labelled by a season, and each storing 0 or 1 according to the season in each row. In other words, the "Winter" column contains 1 whenever the season is winter, and 0 elsewhere. **Do this for each season. Afterwards, remember to delete the Season feature.**

[67]:

```python
### Your code here
# use season col to make 4 new cols

#take the 11th (seasons) column out
season_col = rentals.iloc[:,11]

for season in ['Spring','Summer','Autumn','Winter']:
    #pick out a season and make a new column
    season_row = season_col == season
    #make a new column
```

```
    rentals[season] = 0
    #set this season's row to 1
    rentals[season].values[season_row] = 1
```

[68]:
```
#see the summeries about current table
rentals.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 8465 entries, 0 to 8759
Data columns (total 17 columns):
 #   Column                     Non-Null Count  Dtype
---  ------                     --------------  -----
 0   Date                       8465 non-null   object
 1   Rented Bike Count          8465 non-null   int64
 2   Hour                       8465 non-null   int64
 3   Temperature (C)            8465 non-null   float64
 4   Humidity (%)               8465 non-null   int64
 5   Wind speed (m/s)           8464 non-null   float64
 6   Visibility (10m)           8465 non-null   int64
 7   Dew point temperature (C)  8464 non-null   float64
 8   Solar Radiation (MJ/m2)    8465 non-null   float64
 9   Rainfall(mm)               8463 non-null   object
 10  Snowfall (cm)              8465 non-null   object
 11  Seasons                    8465 non-null   object
 12  Holiday                    8465 non-null   int64
 13  Spring                     8465 non-null   int64
 14  Summer                     8465 non-null   int64
 15  Autumn                     8465 non-null   int64
 16  Winter                     8465 non-null   int64
dtypes: float64(4), int64(9), object(4)
memory usage: 1.2+ MB
```

[69]:
```
#other infos
rentals.head()
```

[69]:
```
         Date  Rented Bike Count  Hour  Temperature (C)  Humidity (%)  \
0  01/12/2017                254     0             -5.2            37
1  01/12/2017                204     1             -5.5            38
2  01/12/2017                173     2             -6.0            39
3  01/12/2017                107     3             -6.2            40
4  01/12/2017                 78     4             -6.0            36

   Wind speed (m/s)  Visibility (10m)  Dew point temperature (C)  \
0               2.2              2000                      -17.6
1               0.8              2000                      -17.6
2               1.0              2000                      -17.7
3               0.9              2000                      -17.6
```

```
4              2.3              2000                      -18.6

   Solar Radiation (MJ/m2) Rainfall(mm) Snowfall (cm) Seasons  Holiday  \
0                      0.0            0             0  Winter        0
1                      0.0            0             0  Winter        0
2                      0.0            0             0  Winter        0
3                      0.0            0             0  Winter        0
4                      0.0            0             0  Winter        0

   Spring  Summer  Autumn  Winter
0       0       0       0       1
1       0       0       0       1
2       0       0       0       1
3       0       0       0       1
4       0       0       0       1
```

[70]:

```
rentals.describe()
```

[70]:

```
       Rented Bike Count         Hour  Temperature (C)  Humidity (%)  \
count        8465.000000  8465.000000      8465.000000   8465.000000
mean          729.156999    11.509864        12.803591     58.161607
std           642.351166     6.921101        12.515429     20.713601
min             2.000000     0.000000       -17.800000    -26.000000
25%           214.000000     6.000000         3.000000     42.000000
50%           542.000000    12.000000        13.500000     57.000000
75%          1084.000000    18.000000        22.700000     74.000000
max          3556.000000    24.000000       306.000000    309.000000

       Wind speed (m/s)  Visibility (10m)  Dew point temperature (C)  \
count       8464.000000       8465.000000                8464.000000
mean           1.962169       1433.477141                   3.945558
std           21.744979        609.596083                  13.243081
min            0.000000       -678.000000                 -30.600000
25%            0.900000        935.000000                  -5.100000
50%            1.500000       1689.000000                   4.700000
75%            2.300000       2000.000000                  15.200000
max         2000.000000       2000.000000                  27.200000

       Solar Radiation (MJ/m2)      Holiday       Spring       Summer  \
count              8465.000000  8465.000000  8465.000000  8465.000000
mean                  0.567868     0.048198     0.255168     0.260839
std                   0.868245     0.214198     0.435982     0.439118
min                   0.000000     0.000000     0.000000     0.000000
25%                   0.000000     0.000000     0.000000     0.000000
50%                   0.010000     0.000000     0.000000     0.000000
75%                   0.930000     0.000000     1.000000     1.000000
max                   3.520000     1.000000     1.000000     1.000000
```

```
            Autumn       Winter
count  8465.000000  8465.000000
mean      0.228825     0.255168
std       0.420101     0.435982
min       0.000000     0.000000
25%       0.000000     0.000000
50%       0.000000     0.000000
75%       0.000000     1.000000
max       1.000000     1.000000
```

[71]:
```
'''to check if the new columns are correct, we sum the mean value of the four␣
↪seasons together to see if it is 1'''
print(rentals["Winter"].mean() + rentals["Autumn"].mean()+rentals["Summer"].
↪mean()+rentals["Spring"].mean()) # we can see it's 1
```

```
1.0
```

[72]:
```
#drop the season col
rentals.drop(columns=['Seasons'],inplace=True)
rentals.head()
rentals.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 8465 entries, 0 to 8759
Data columns (total 16 columns):
 #   Column                     Non-Null Count  Dtype
---  ------                     --------------  -----
 0   Date                       8465 non-null   object
 1   Rented Bike Count          8465 non-null   int64
 2   Hour                       8465 non-null   int64
 3   Temperature (C)            8465 non-null   float64
 4   Humidity (%)               8465 non-null   int64
 5   Wind speed (m/s)           8464 non-null   float64
 6   Visibility (10m)           8465 non-null   int64
 7   Dew point temperature (C)  8464 non-null   float64
 8   Solar Radiation (MJ/m2)    8465 non-null   float64
 9   Rainfall(mm)               8463 non-null   object
 10  Snowfall (cm)              8465 non-null   object
 11  Holiday                    8465 non-null   int64
 12  Spring                     8465 non-null   int64
 13  Summer                     8465 non-null   int64
 14  Autumn                     8465 non-null   int64
 15  Winter                     8465 non-null   int64
dtypes: float64(4), int64(9), object(3)
memory usage: 1.1+ MB
```

It is known that bike rentals depend strongly on whether it's a weekday or a weekend. **Replace**

the Date feature with a Weekday feature that stores 0 or 1 depending on whether the date represents a weekend or weekday. To do this, use the function `date_is_weekday` below, which returns 1 if it is a weekday and 0 if it is a weekend.

Apply the function to the Date column in your DataFrame (you can use `DataFrame.transform` to apply it).

[73]:
```
import datetime
def date_is_weekday(datestring):
    ### return 0 if weekend, 1 if weekday
    dsplit = datestring.split('/')
    wday = datetime.datetime(int(dsplit[2]),int(dsplit[1]),int(dsplit[0])).
 ↪weekday()
    return int(wday<=4)

### Your code to apply the function here:

#take the 0th (date) column out
date_col = rentals.iloc[:,0]
print(rentals['Date'].values[0])
is_week_day= [];
for n in range(len(date_col)):
    date = rentals['Date'].values[n]
    rentals['Date'].values[n] =date_is_weekday(date)

# for value in rentals['Date'].values:
#     print(value)#to check if the date is transformed correctly
# make date column to numeric
rentals['Date'] =  rentals["Date"].apply(pd.to_numeric,errors="coerce")
rentals.info()
```

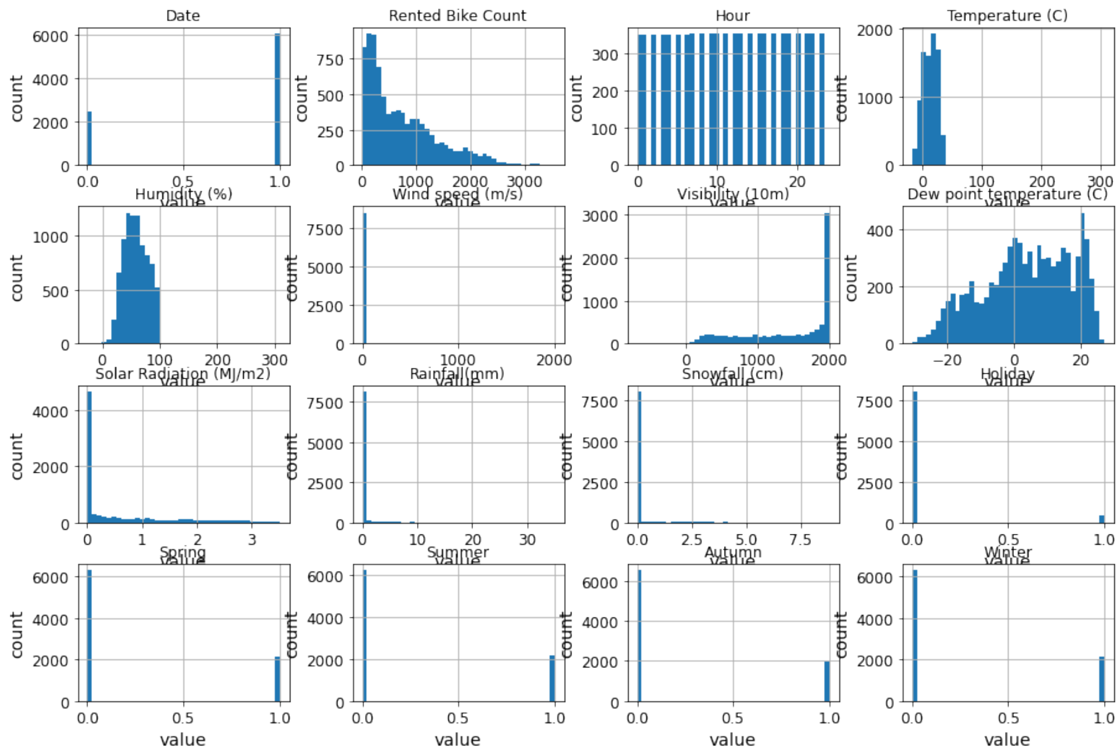
```
01/12/2017
<class 'pandas.core.frame.DataFrame'>
Int64Index: 8465 entries, 0 to 8759
Data columns (total 16 columns):
 #   Column                     Non-Null Count  Dtype
---  ------                     --------------  -----
 0   Date                       8465 non-null   int64
 1   Rented Bike Count          8465 non-null   int64
 2   Hour                       8465 non-null   int64
 3   Temperature (C)            8465 non-null   float64
 4   Humidity (%)               8465 non-null   int64
 5   Wind speed (m/s)           8464 non-null   float64
 6   Visibility (10m)           8465 non-null   int64
 7   Dew point temperature (C)  8464 non-null   float64
 8   Solar Radiation (MJ/m2)    8465 non-null   float64
 9   Rainfall(mm)               8463 non-null   object
 10  Snowfall (cm)              8465 non-null   object
```

```
 11  Holiday                    8465 non-null   int64
 12  Spring                     8465 non-null   int64
 13  Summer                     8465 non-null   int64
 14  Autumn                     8465 non-null   int64
 15  Winter                     8465 non-null   int64
dtypes: float64(4), int64(10), object(2)
memory usage: 1.1+ MB
```

Convert all the remaining data to numerical format, with any non-numerical entries set to NaN.

[74]:
```
### Your code here
# print(rentals.head())
rentals = rentals.apply(pd.to_numeric,errors="coerce")
rentals.info() #all to muneric
rental_hists = rentals.hist(bins=40,figsize=(15,10))#all good
for rental_hist in rental_hists.flatten():
    rental_hist.set_xlabel("value")
    rental_hist.set_ylabel("count")
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 8465 entries, 0 to 8759
Data columns (total 16 columns):
 #   Column                     Non-Null Count  Dtype
---  ------                     --------------  -----
 0   Date                       8465 non-null   int64
 1   Rented Bike Count          8465 non-null   int64
 2   Hour                       8465 non-null   int64
 3   Temperature (C)            8465 non-null   float64
 4   Humidity (%)               8465 non-null   int64
 5   Wind speed (m/s)           8464 non-null   float64
 6   Visibility (10m)           8465 non-null   int64
 7   Dew point temperature (C)  8464 non-null   float64
 8   Solar Radiation (MJ/m2)    8465 non-null   float64
 9   Rainfall(mm)               8440 non-null   float64
 10  Snowfall (cm)              8442 non-null   float64
 11  Holiday                    8465 non-null   int64
 12  Spring                     8465 non-null   int64
 13  Summer                     8465 non-null   int64
 14  Autumn                     8465 non-null   int64
 15  Winter                     8465 non-null   int64
dtypes: float64(6), int64(10)
memory usage: 1.1 MB
```



1.4 Step 2: Visualise the data and perform further processing (20%)

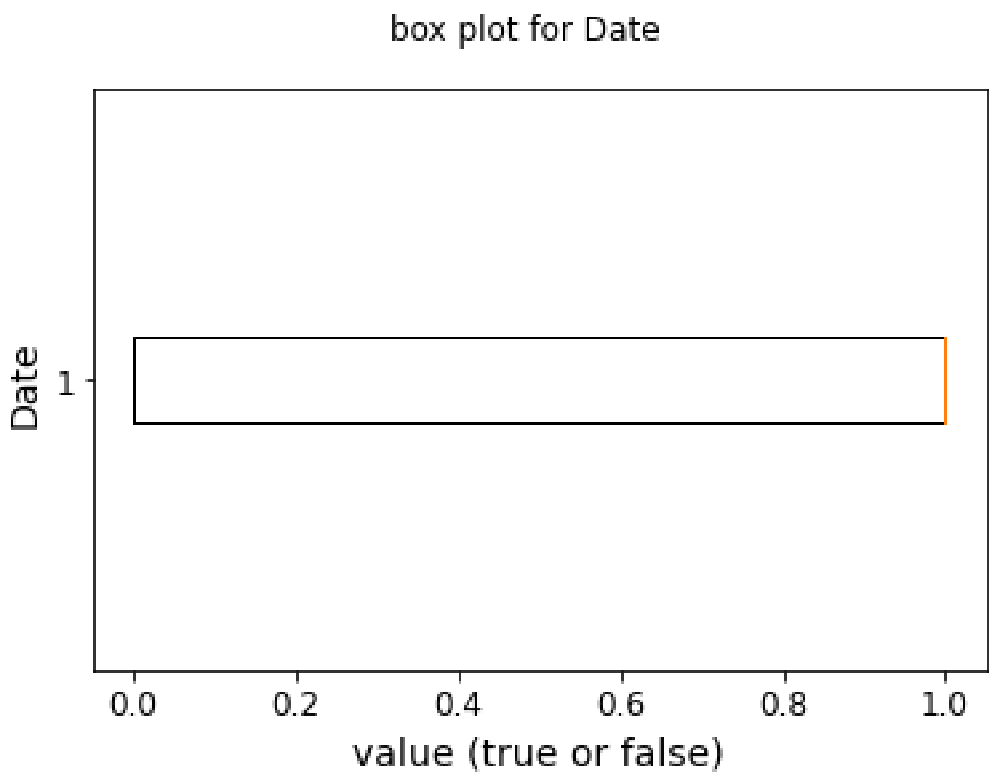
1.4.1 2.1 Visualisation

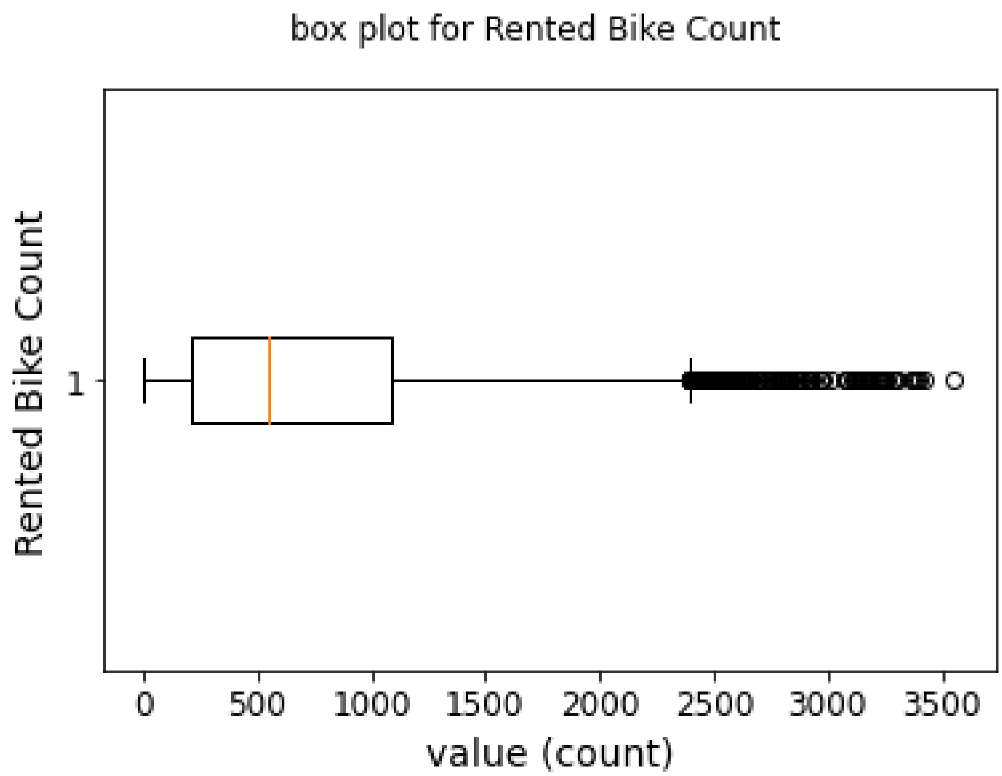
Use **at least two graphical methods to display your data** and **identify** problematic entries. **Write one sentence** that summarises what you found about problematic entries.

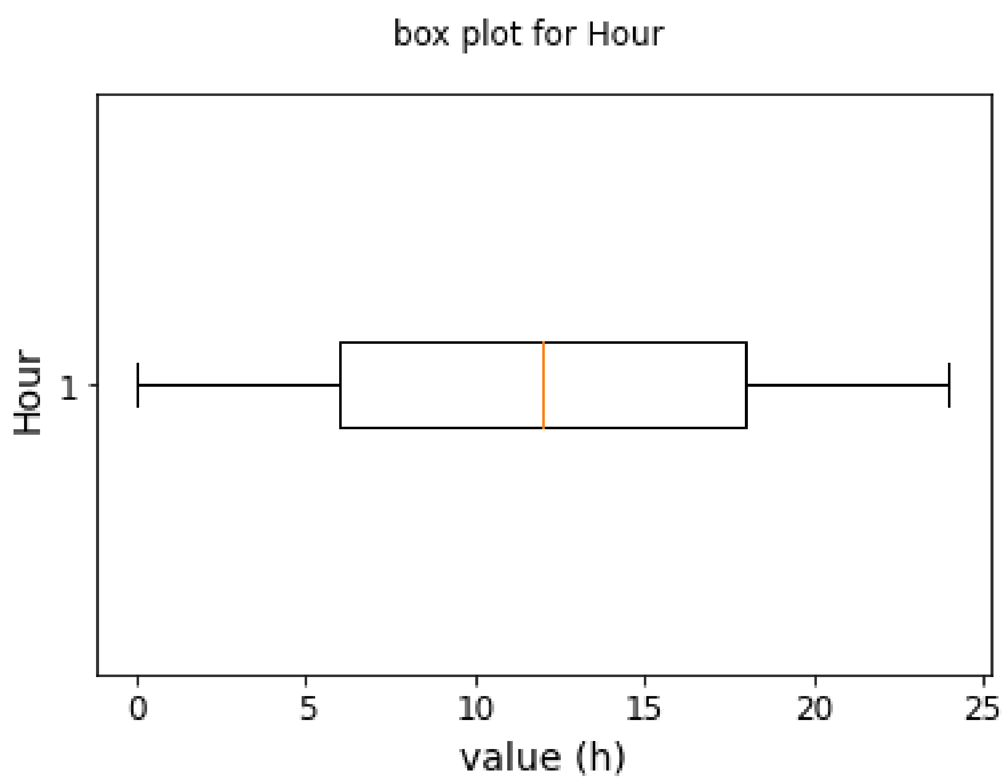
[75]:
```
#remove nan row before boxplotting
#this is just for plotting, but we want to impute it later. So we use a copy

rentals_copy = rentals.dropna()
```

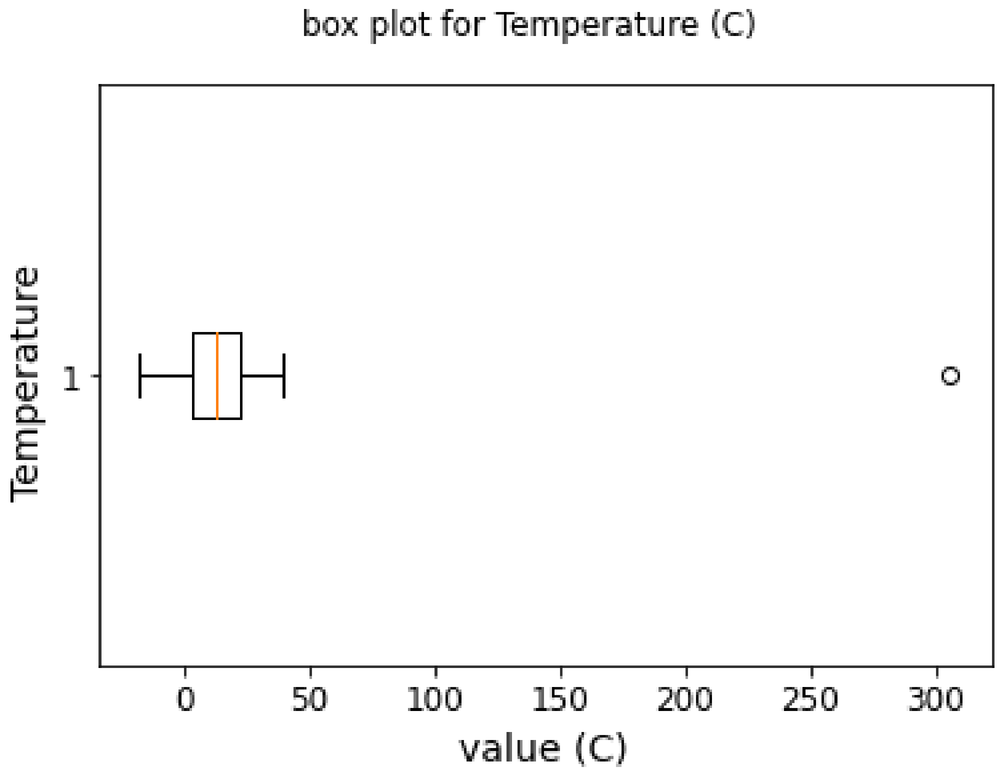
[76]:
```
### Your code here
#visualise using boxplots
for n in range (rentals_copy.shape[1]):
    plt.boxplot(rentals_copy.iloc[:,n],vert =False)
    plt.suptitle( f'{"box plot for " +rentals_copy.columns[n]}'  )
    column_splited = rentals_copy.columns[n].split("(")
#     print(column_splited)
    plt.ylabel(f'{column_splited[0]}')
    #if the column name has the unit, add unit to x axis
    #or just print 'value'
    if len(column_splited) > 1:
        plt.xlabel("value ("+ f'{column_splited[1]}')
```

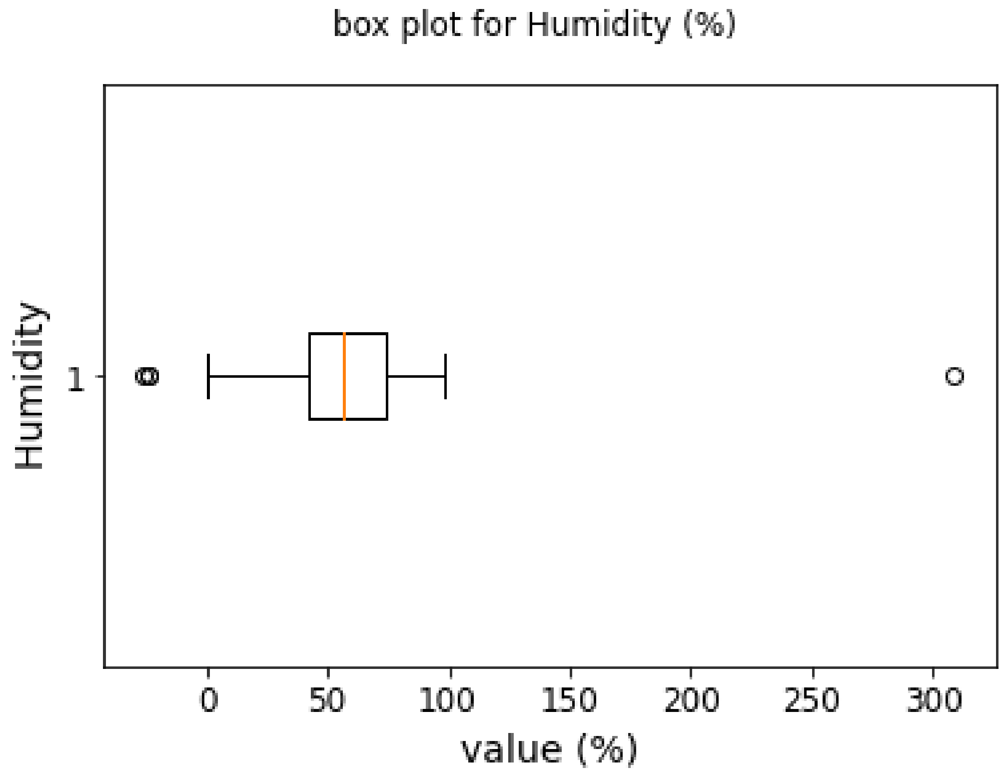
```
    else:
        # to make the lable mre clearer, manually deal with the unit
        if column_splited[0] == 'Rented Bike Count':
            plt.xlabel("value (count)")
        elif column_splited[0] == 'Hour':
            plt.xlabel("value (h)")

        else:
            plt.xlabel("value (true or false)")

plt.show()
```

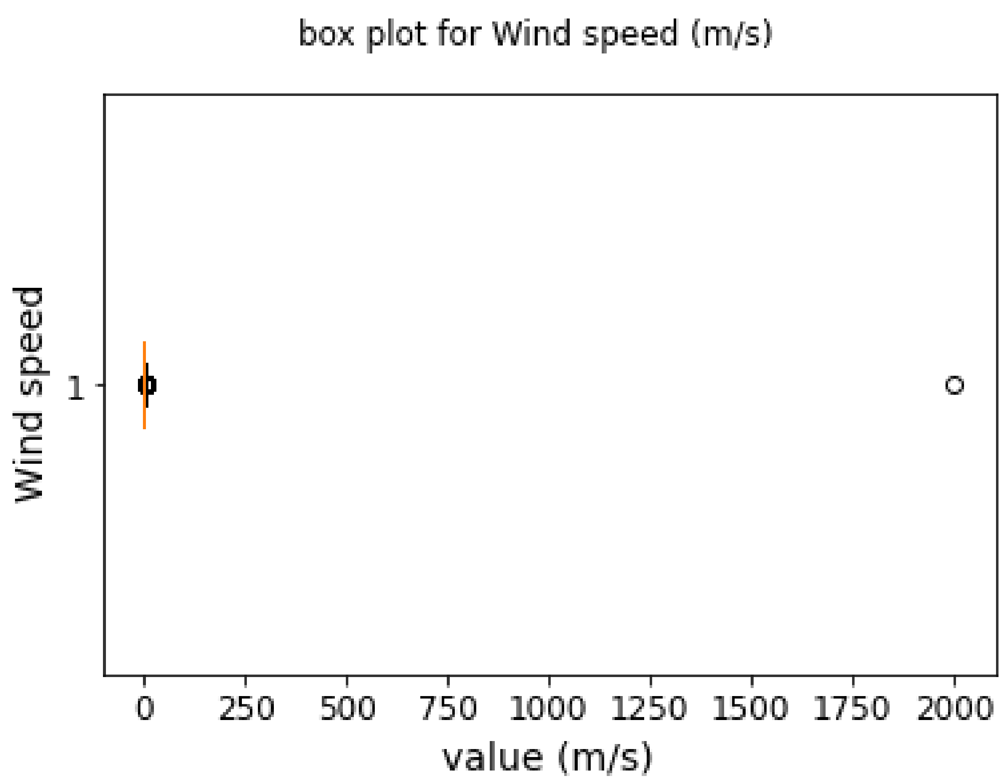


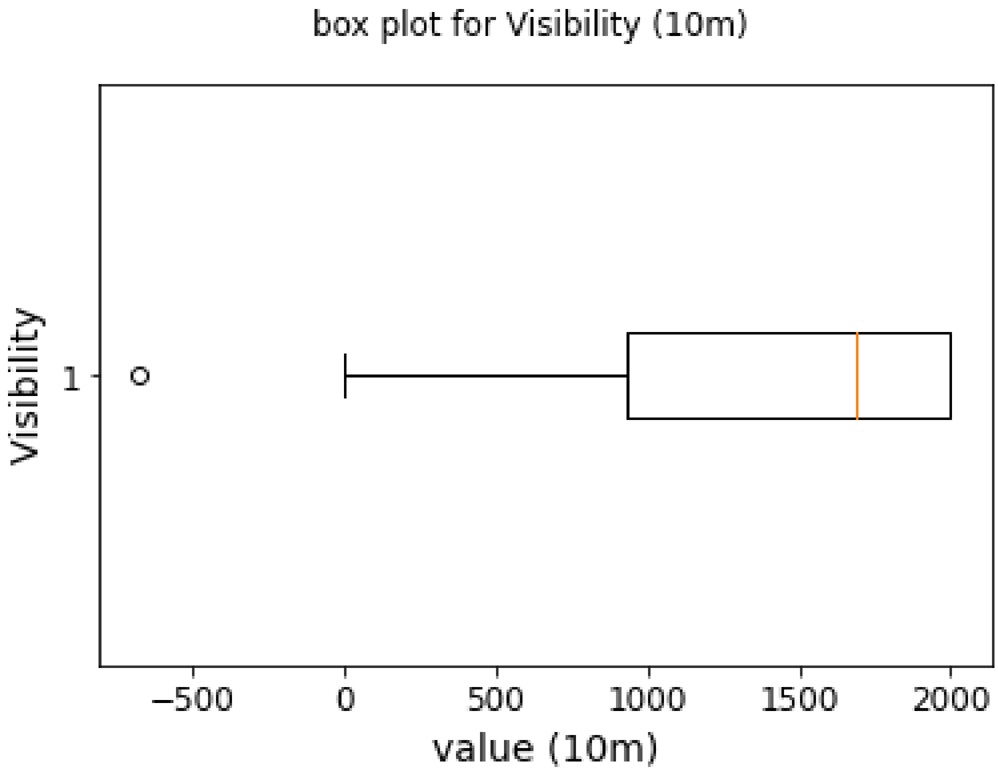

box plot for Rented Bike Count
Rented Bike Count
1
0
500
1000
1500
2000
2500
3000
3500
value (count)

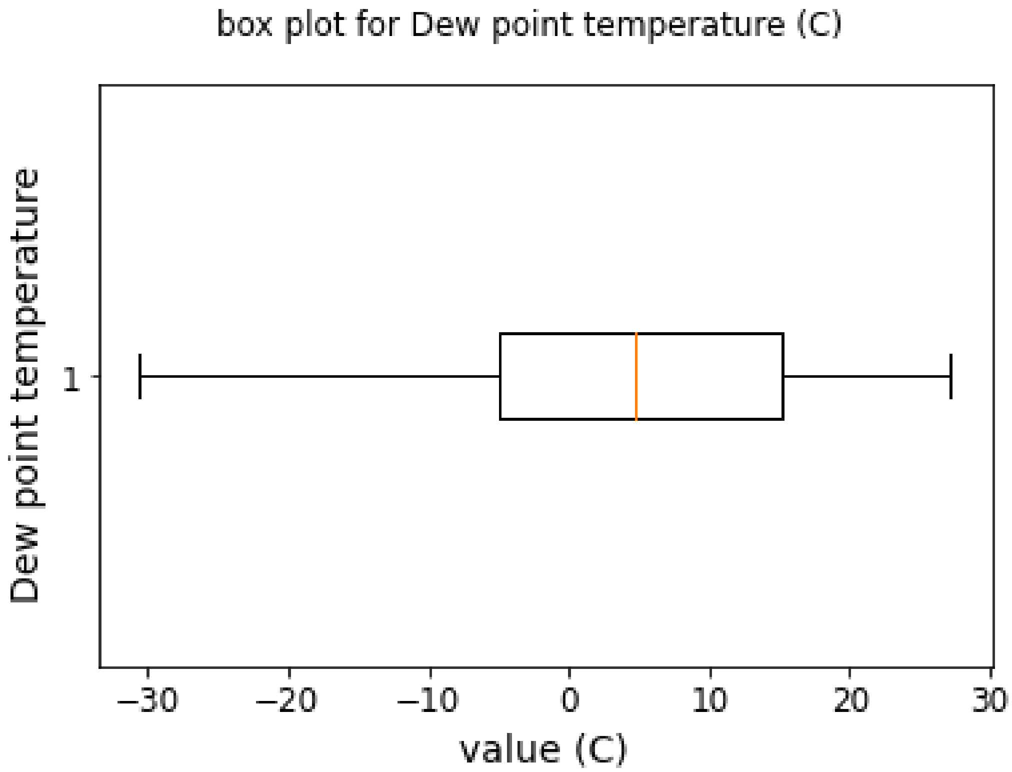

box plot for Hour
Hour
1
0
5
10
15
20
25
value (h)

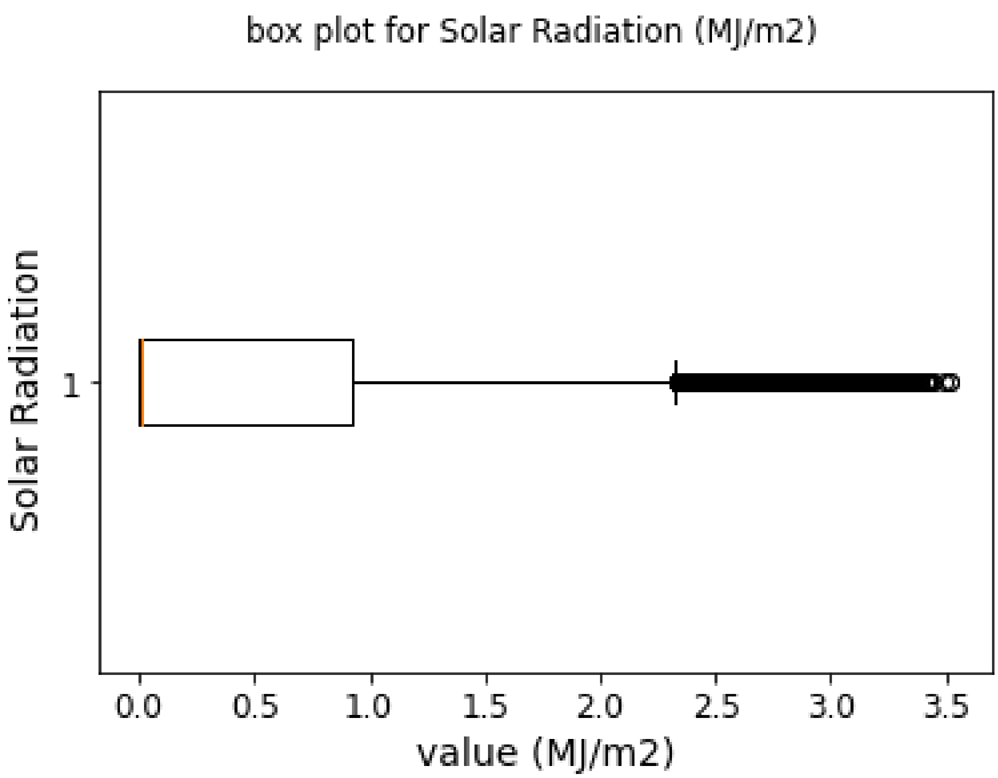

box plot for Temperature (C)
Temperature
1
0
50
100
150
200
250
300
value (C)

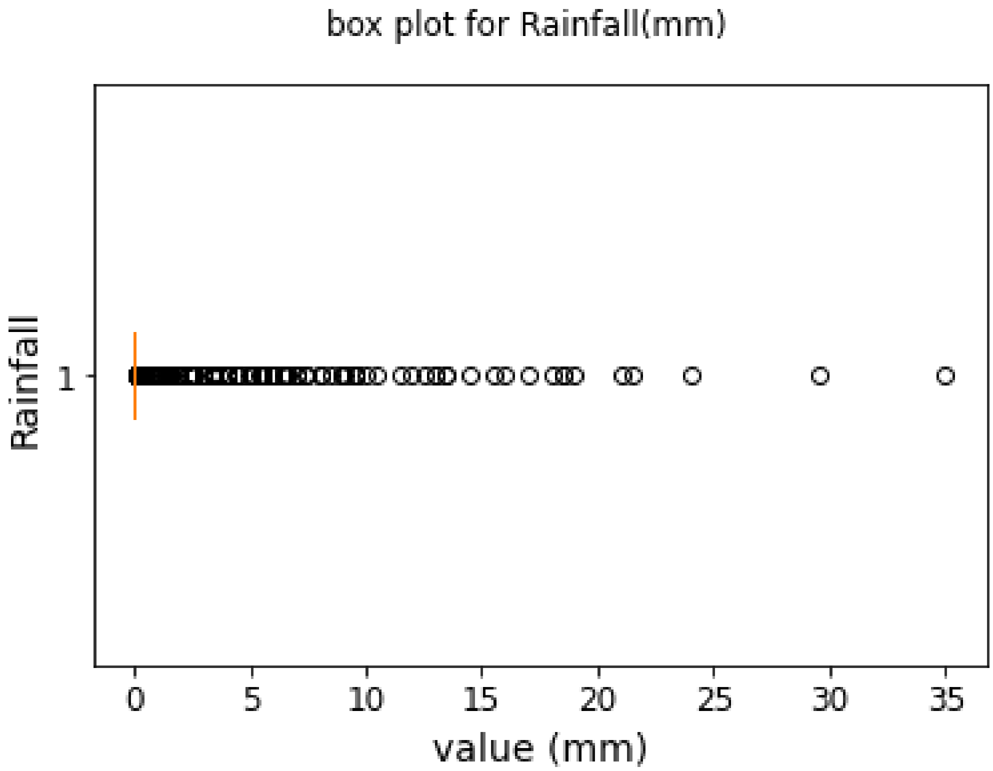

box plot for Humidity (%)
Humidity
1
0
50
100
150
200
250
300
value (%)

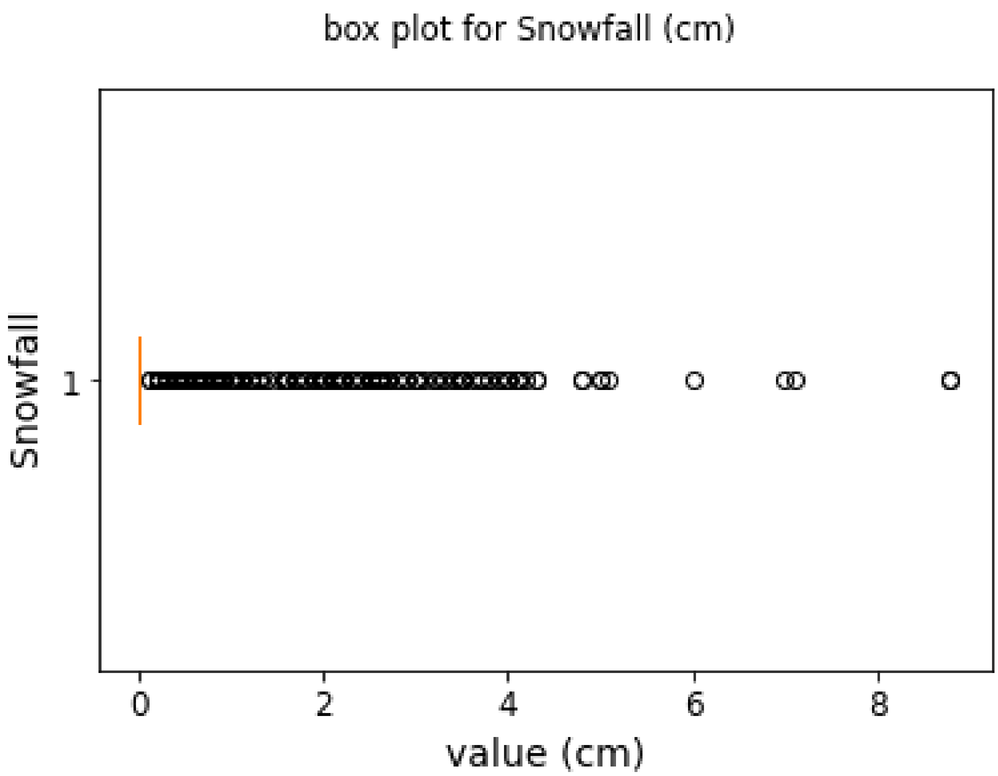

box plot for Wind speed (m/s)
Wind speed
1
0
250
500
750
1000
1250
1500
1750
2000
value (m/s)

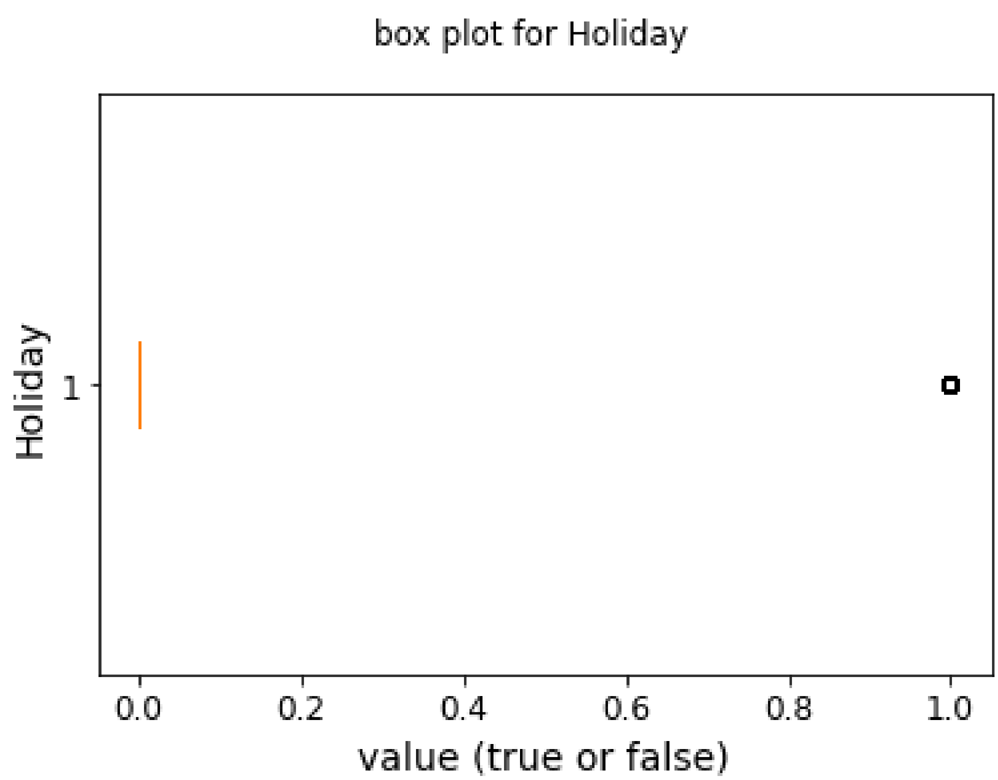

box plot for Visibility (10m)
Visibility
1
−500
0
500
1000
1500
2000
value (10m)

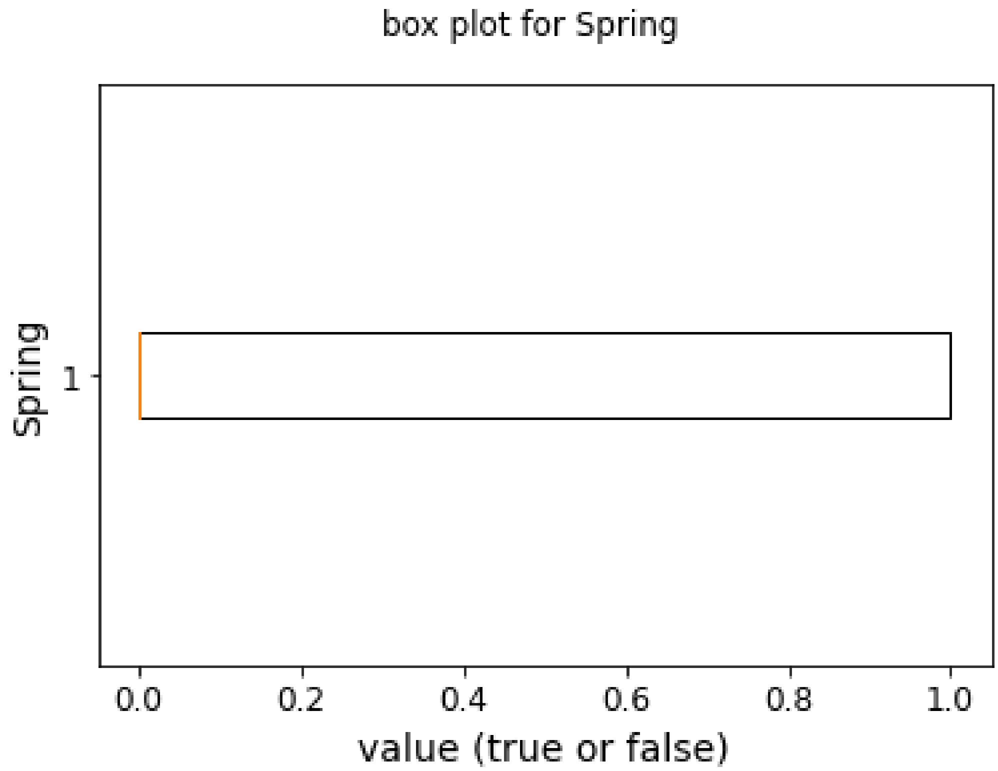

box plot for Dew point temperature (C)
Dew point temperature
1
−30
−20
−10
0
10
20
30
value (C)

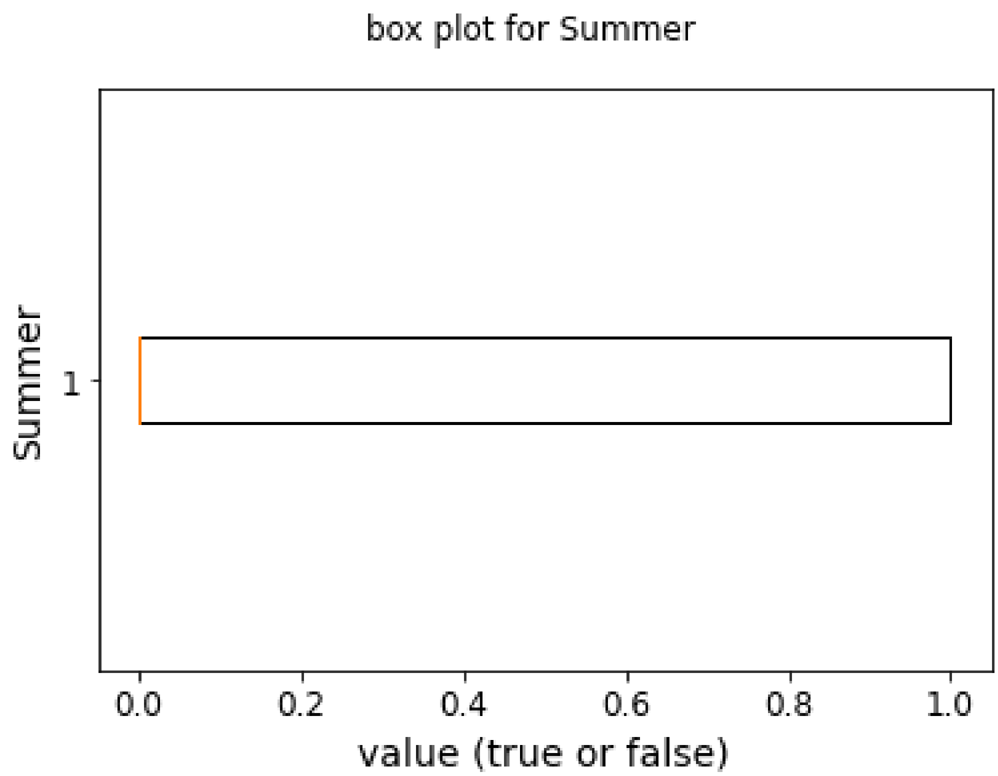

box plot for Solar Radiation (MJ/m2)
Solar Radiation
1
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
value (MJ/m2)


box plot for Rainfall(mm)
Rainfall
1
0
5
10
15
20
25
30
35
value (mm)

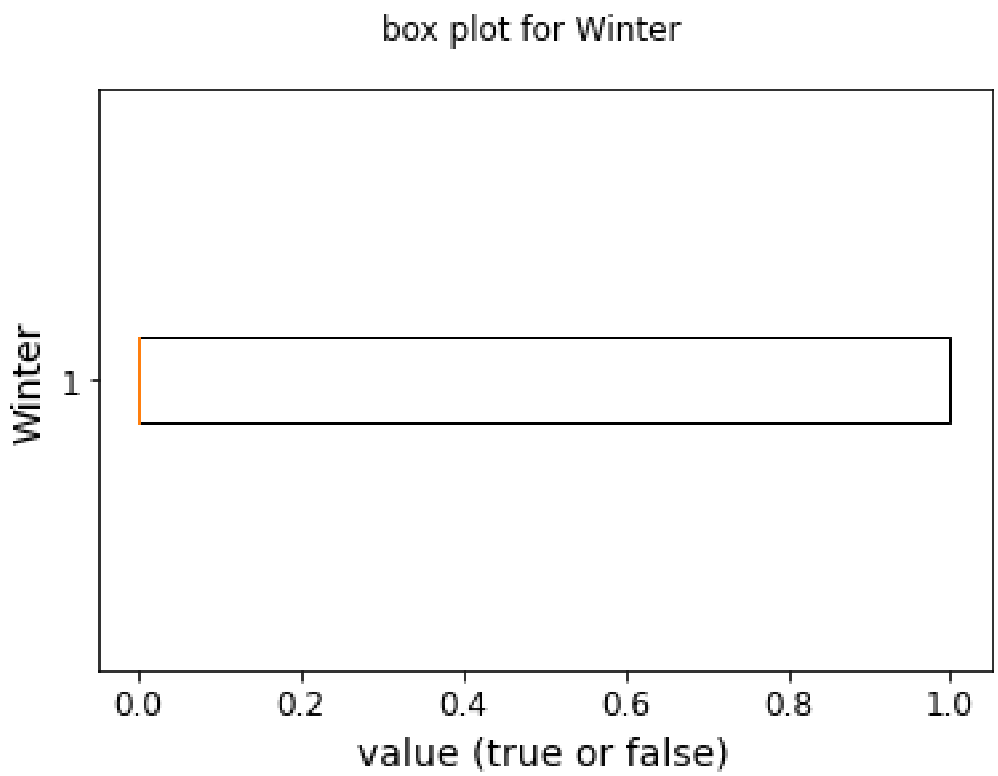

box plot for Snowfall (cm)
Snowfall
1
0
2
4
6
8
value (cm)


box plot for Holiday
Holiday
1
0.0
0.2
0.4
0.6
0.8
1.0
value (true or false)


box plot for Spring
Spring
1
0.0
0.2
0.4
0.6
0.8
1.0
value (true or false)

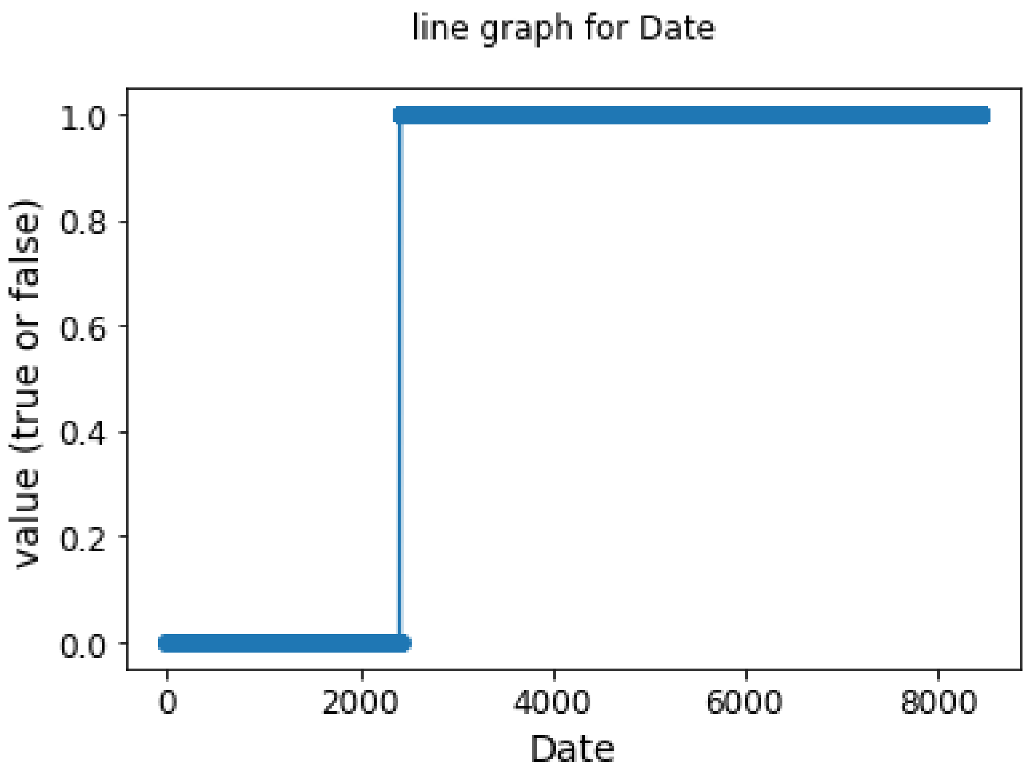

box plot for Summer
Summer
1
0.0
0.2
0.4
0.6
0.8
1.0
value (true or false)


box plot for Autumn
Autumn
1
0.0
0.2
0.4
0.6
0.8
1.0
value (true or false)

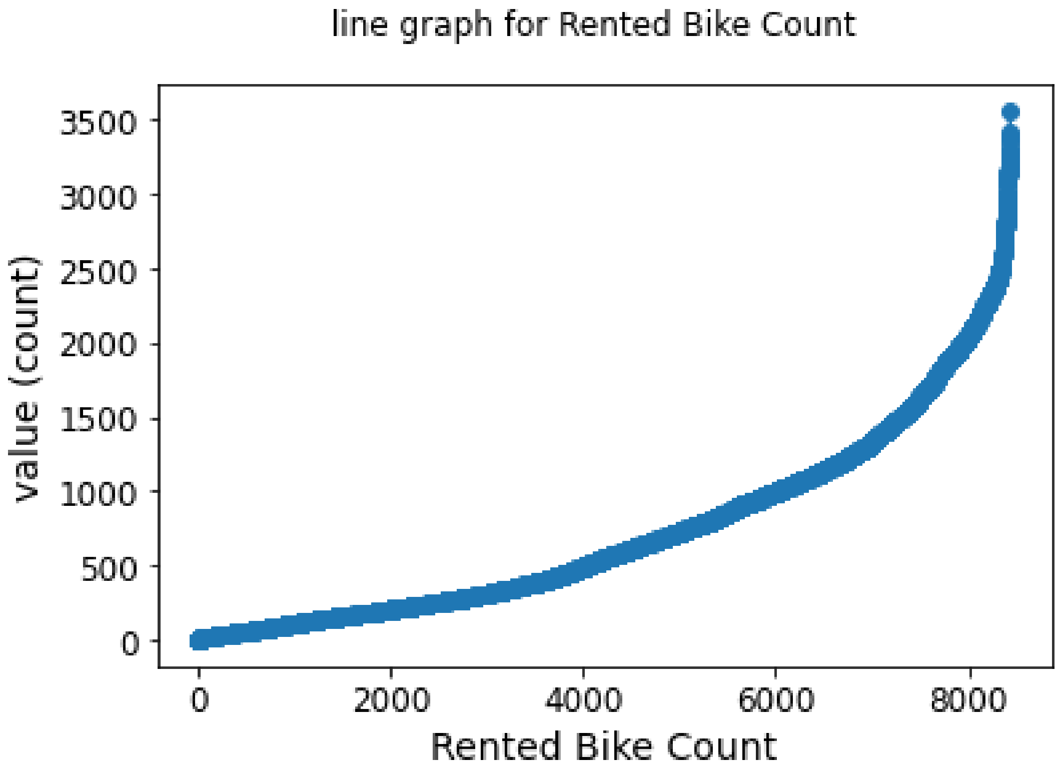


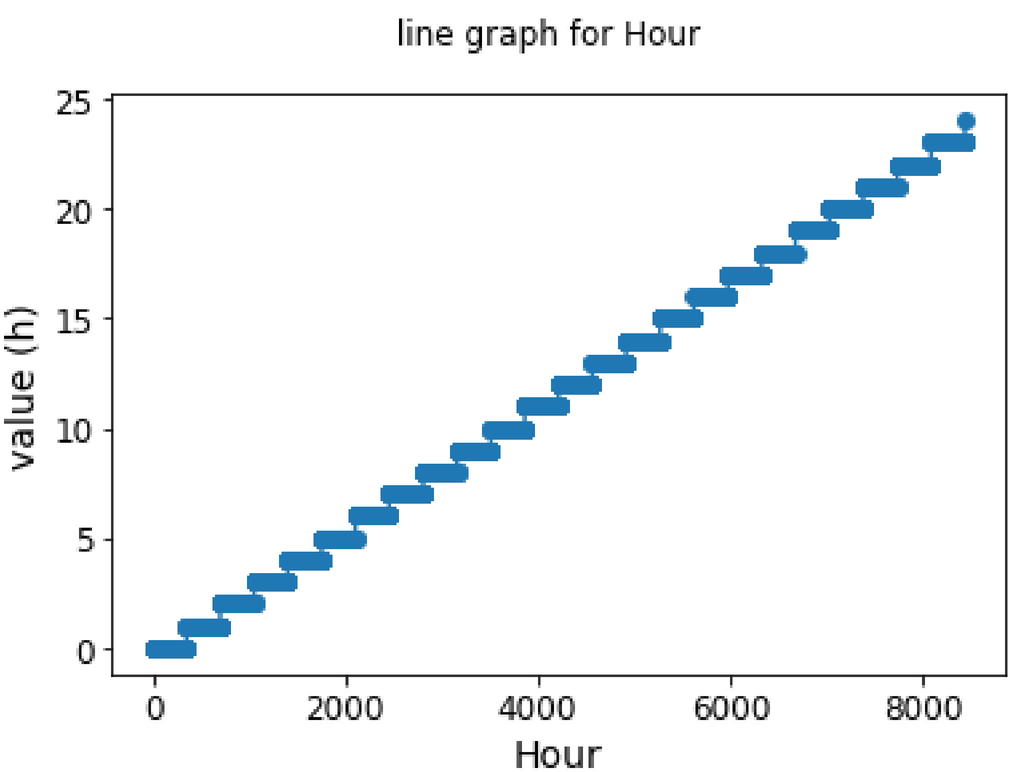
[77]:
```
#visualise using sorted  in line graph
for n in range(rentals_copy.shape[1]):
    plt.plot(np.sort(rentals_copy.iloc[:,n]),'-o')
    plt.suptitle( f'{" line graph for " +rentals_copy.columns[n]}'  )
    column_splited = rentals_copy.columns[n].split("(")
#     print(column_splited)
    plt.xlabel(f'{column_splited[0]}')
    #if the column name has the unit, add unit to y axis
    if len(column_splited) > 1:
        plt.ylabel("value ("+ f'{column_splited[1]}')
    else:
        # to make the lable mre clearer, manually deal with the unit
        if column_splited[0] == 'Rented Bike Count':
            plt.ylabel("value (count)")
        elif column_splited[0] == 'Hour':
            plt.ylabel("value (h)")

        else:
            plt.ylabel("value (true or false)")
```

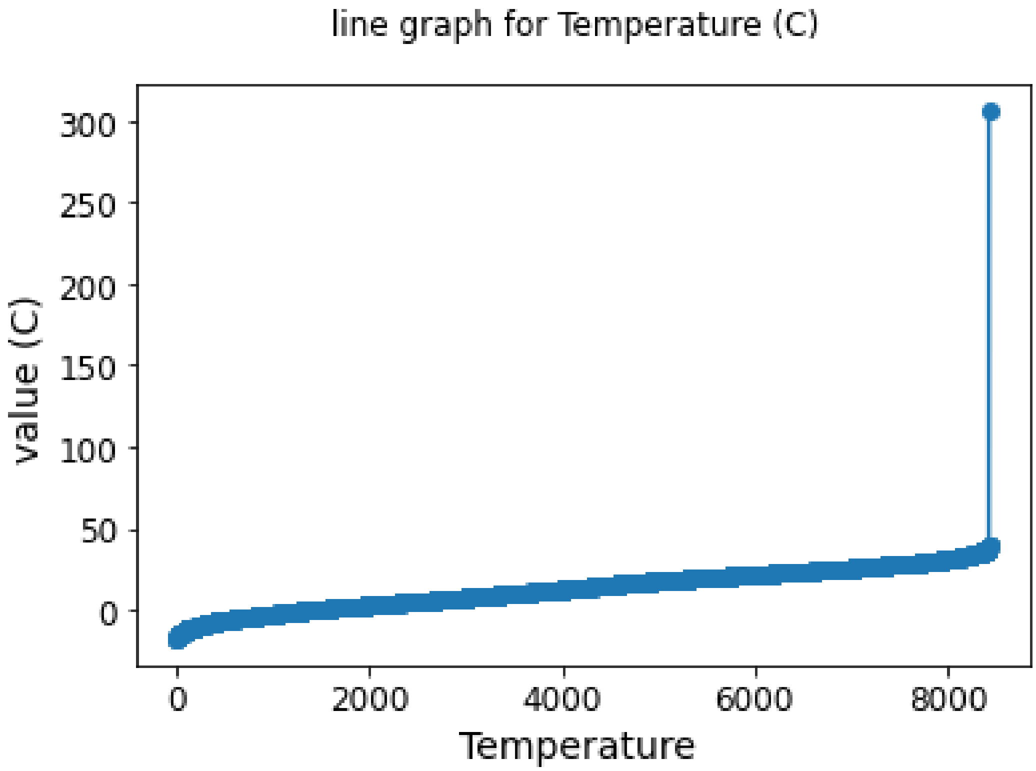
```
plt.show()
```

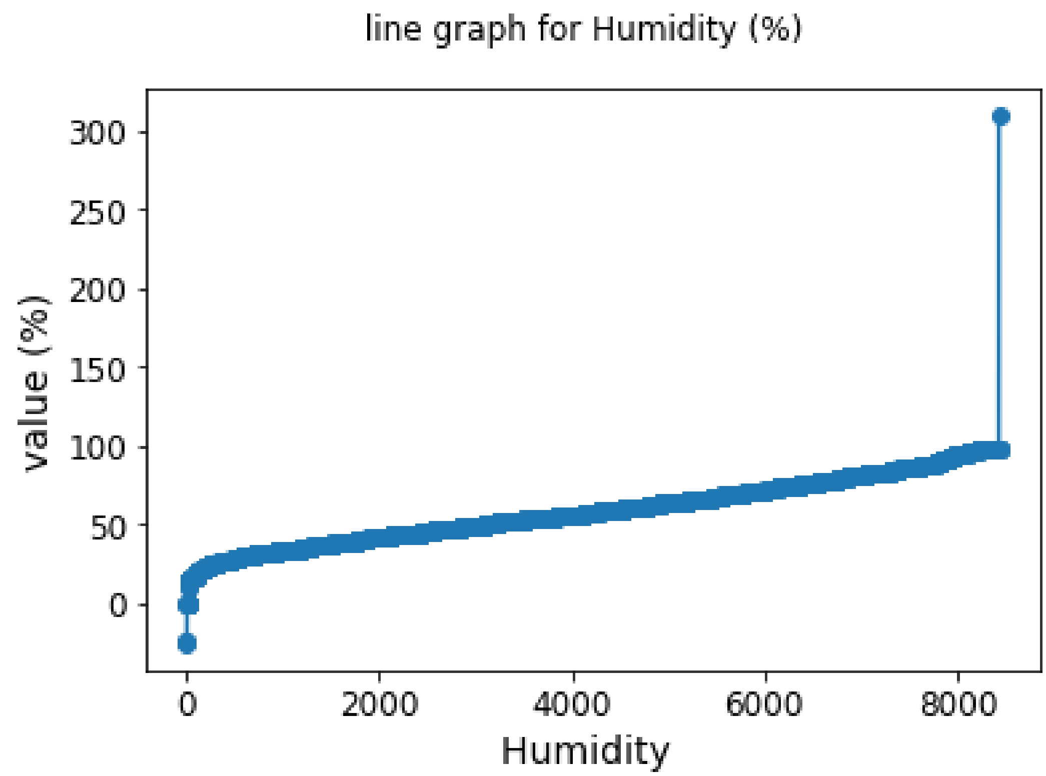


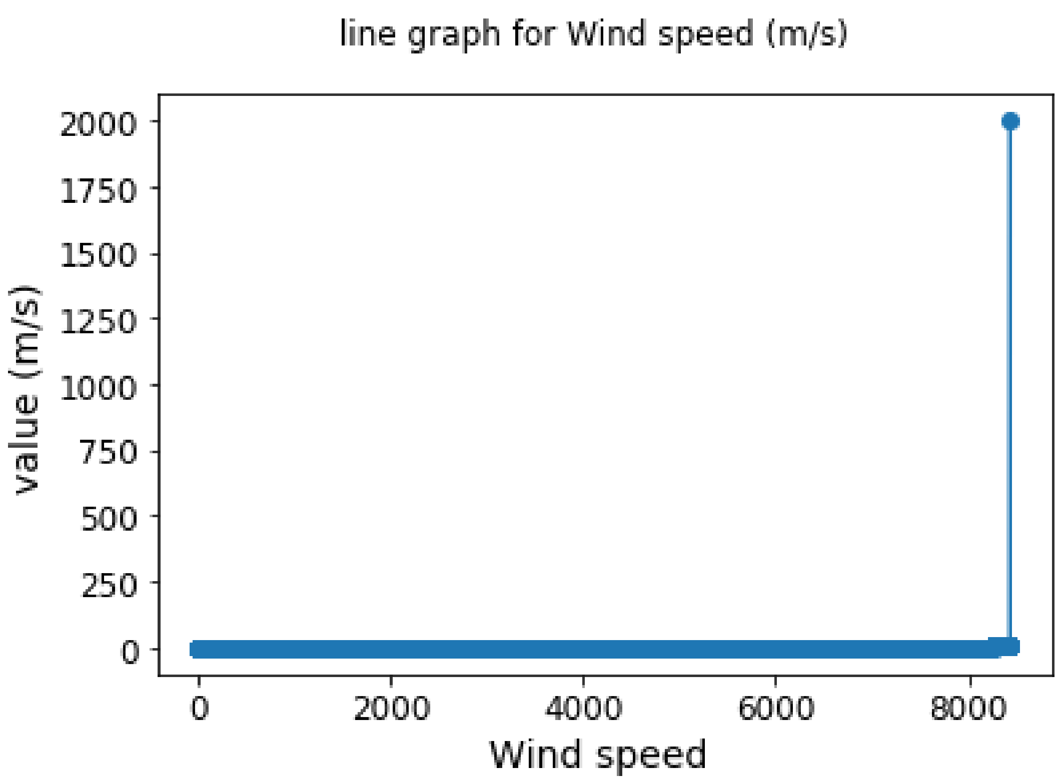
line graph for Date
1.0
0.8
0.6
0.4
0.2
0.0
value (true or false)
0
2000
4000
6000
8000
Date

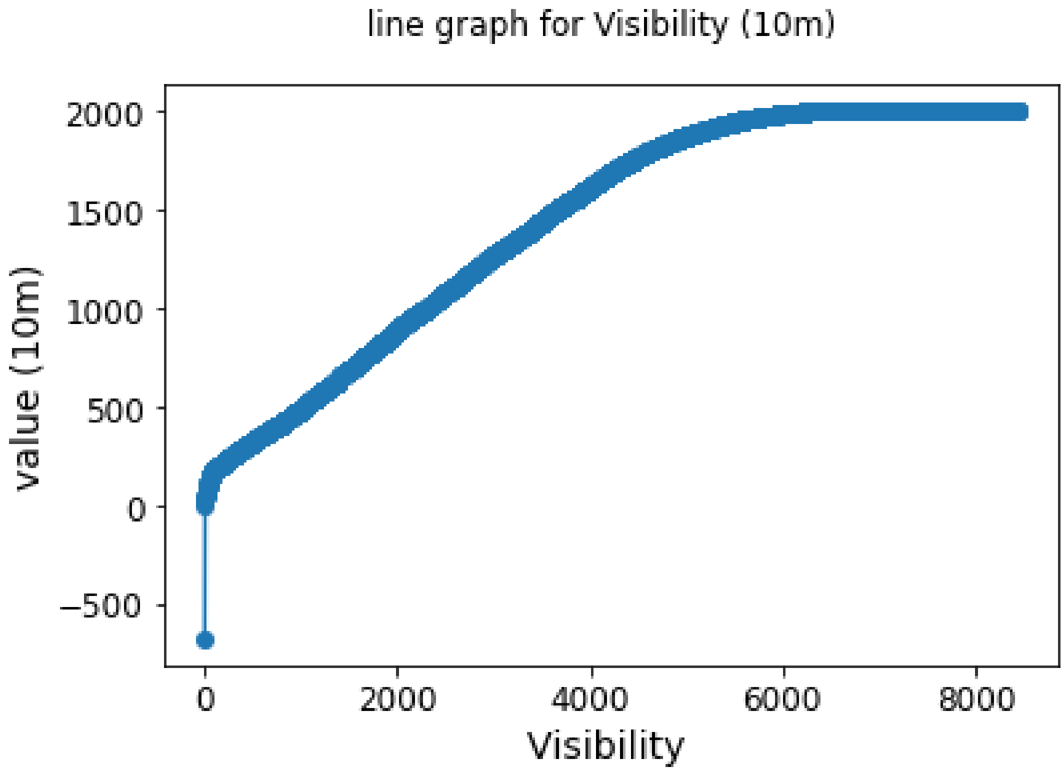

line graph for Rented Bike Count
value (count)
3500
3000
2500
2000
1500
1000
500
0
0
2000
4000
6000
8000
Rented Bike Count

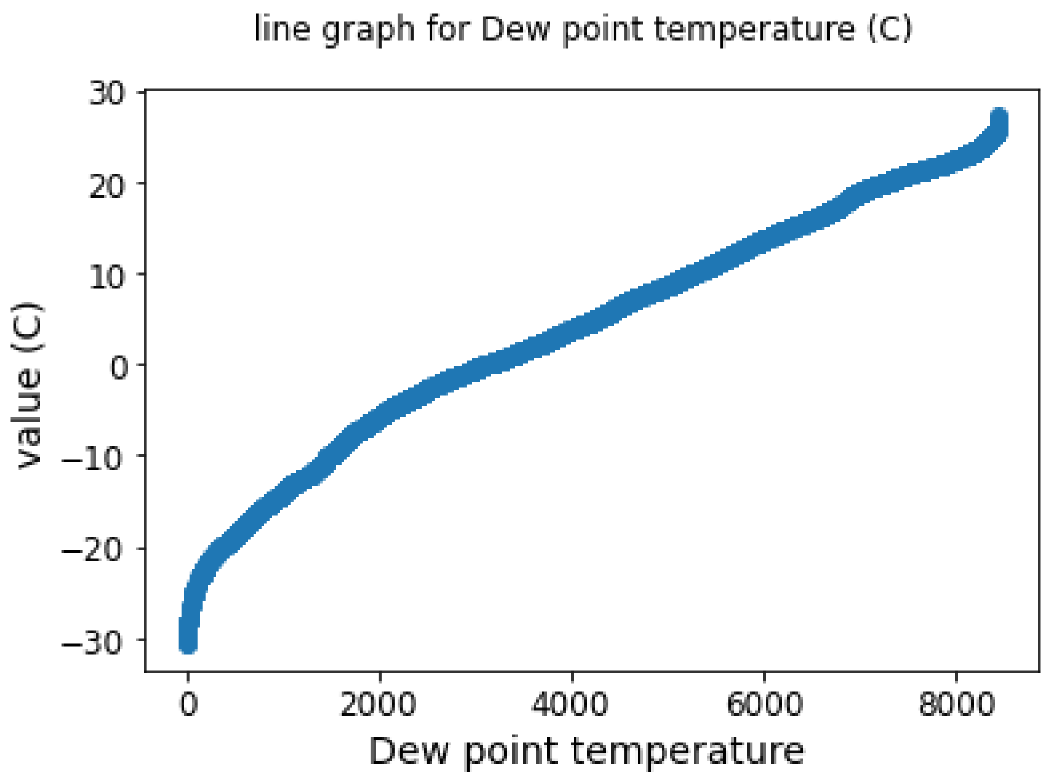

line graph for Hour
value (h)
25
20
15
10
5
0
0
2000
4000
6000
8000
Hour

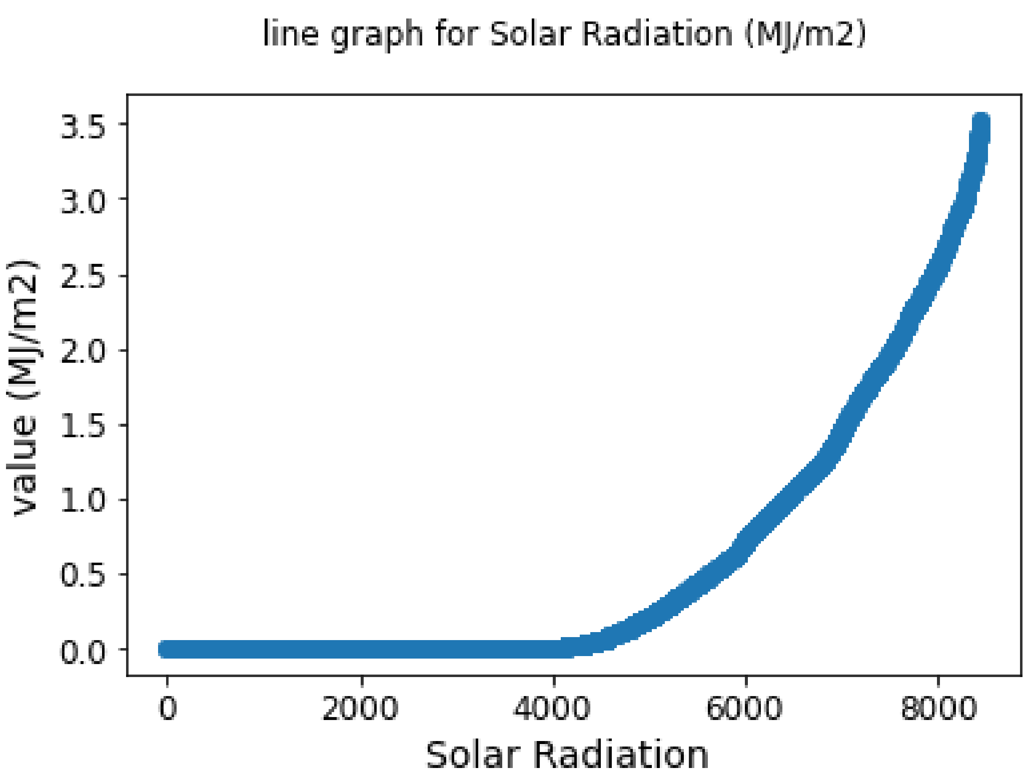

line graph for Temperature (C)
value (C)
300
250
200
150
100
50
0
0
2000
4000
6000
8000
Temperature

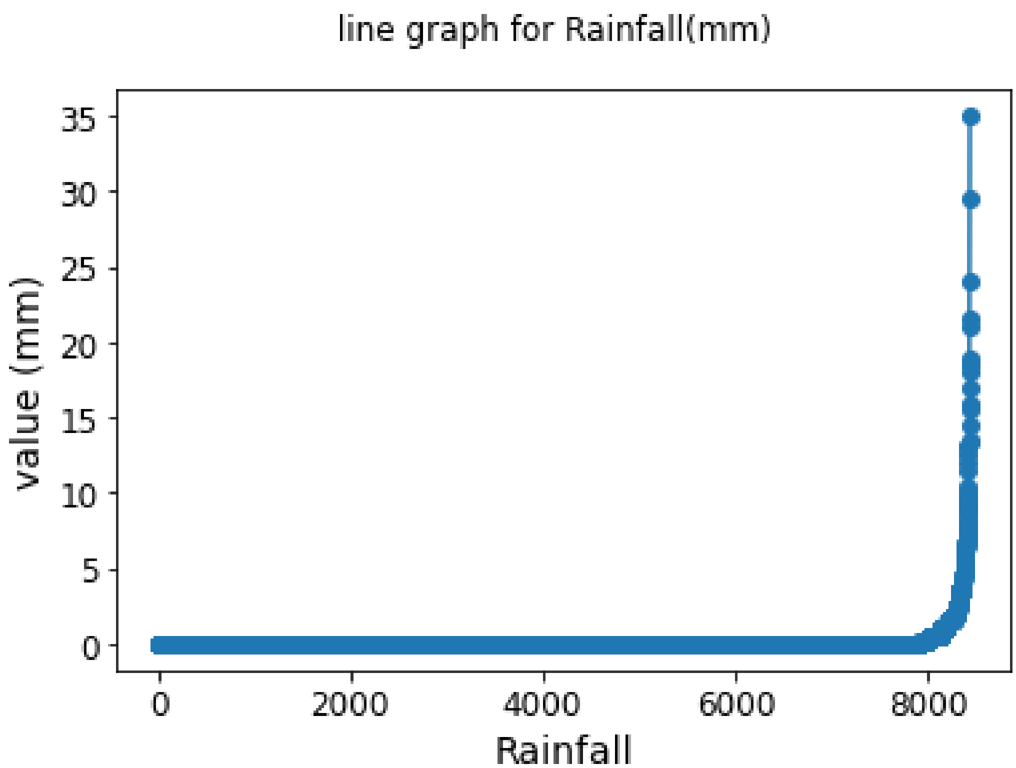

line graph for Humidity (%)
value (%)
300
250
200
150
100
50
0
0
2000
4000
6000
8000
Humidity

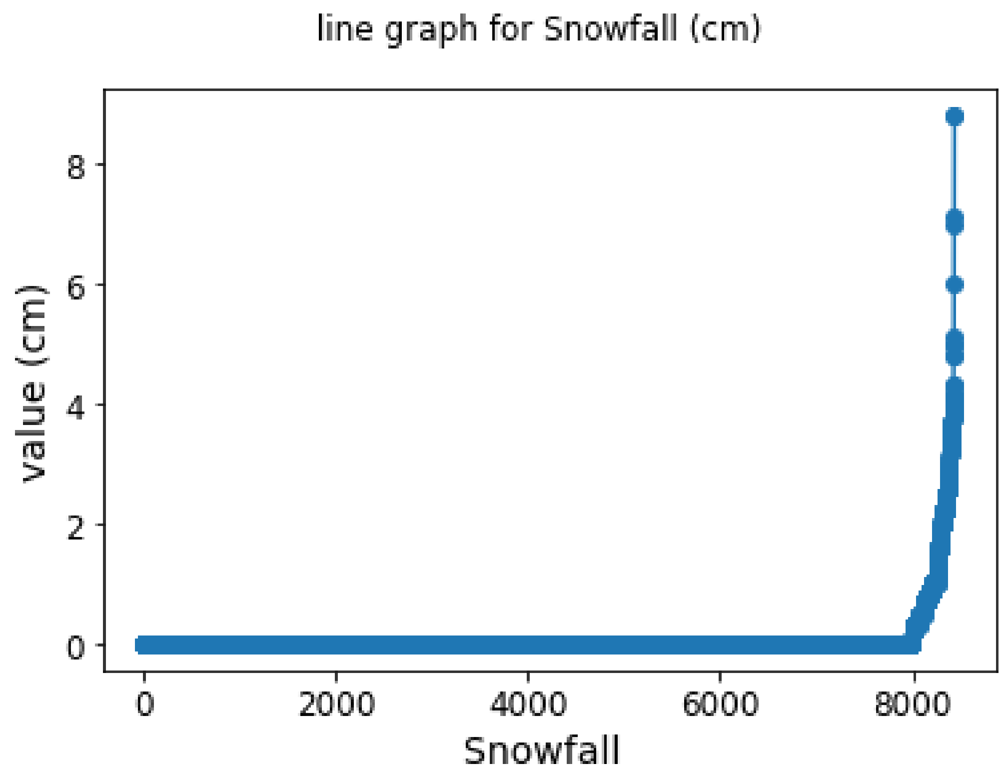

line graph for Wind speed (m/s)
value (m/s)
2000
1750
1500
1250
1000
750
500
250
0
0
2000
4000
6000
8000
Wind speed

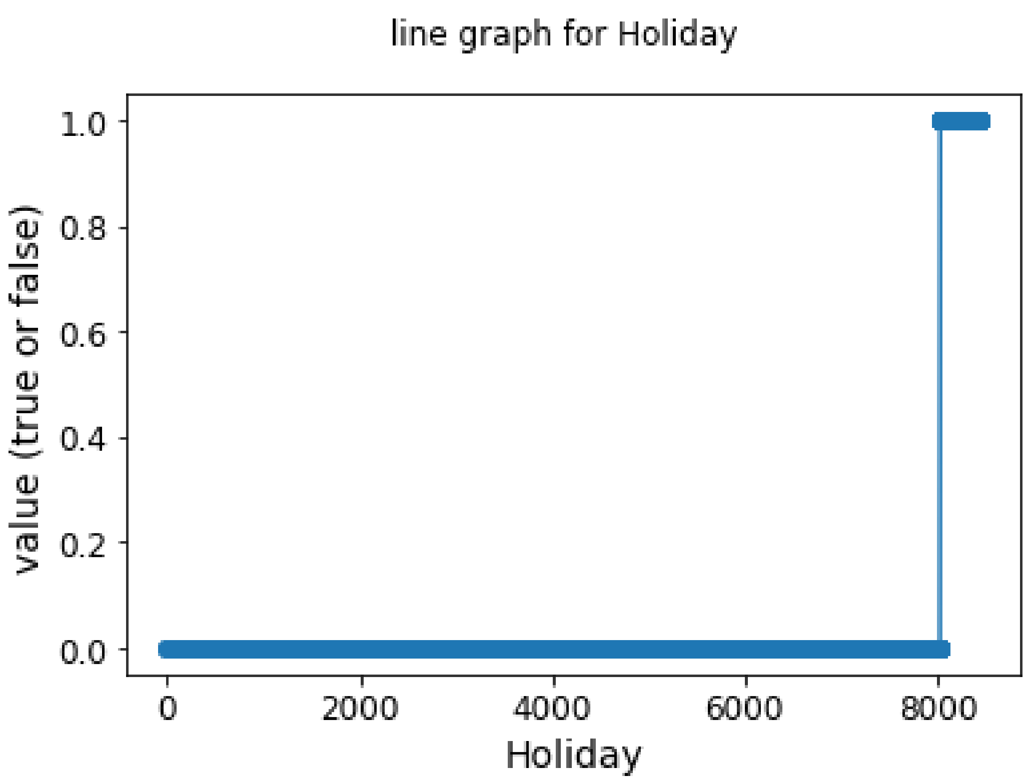

line graph for Visibility (10m)
value (10m)
2000
1500
1000
500
0
−500
0
2000
4000
6000
8000
Visibility

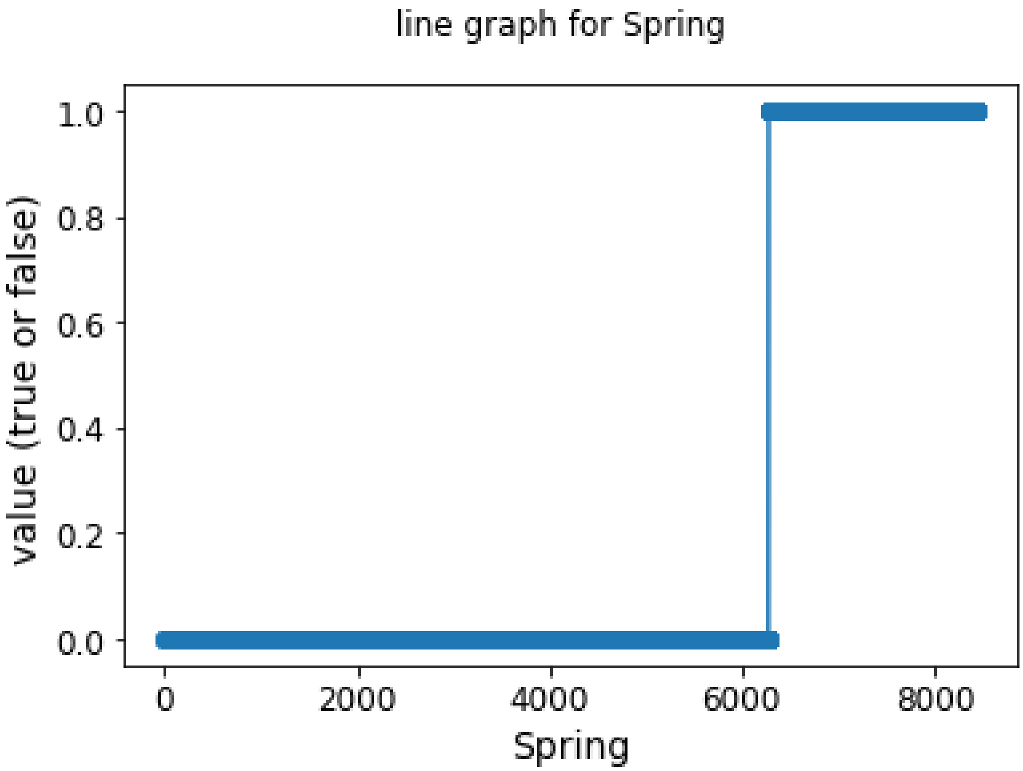

line graph for Dew point temperature (C)
value (C)
30
20
10
0
−10
−20
−30
0
2000
4000
6000
8000
Dew point temperature

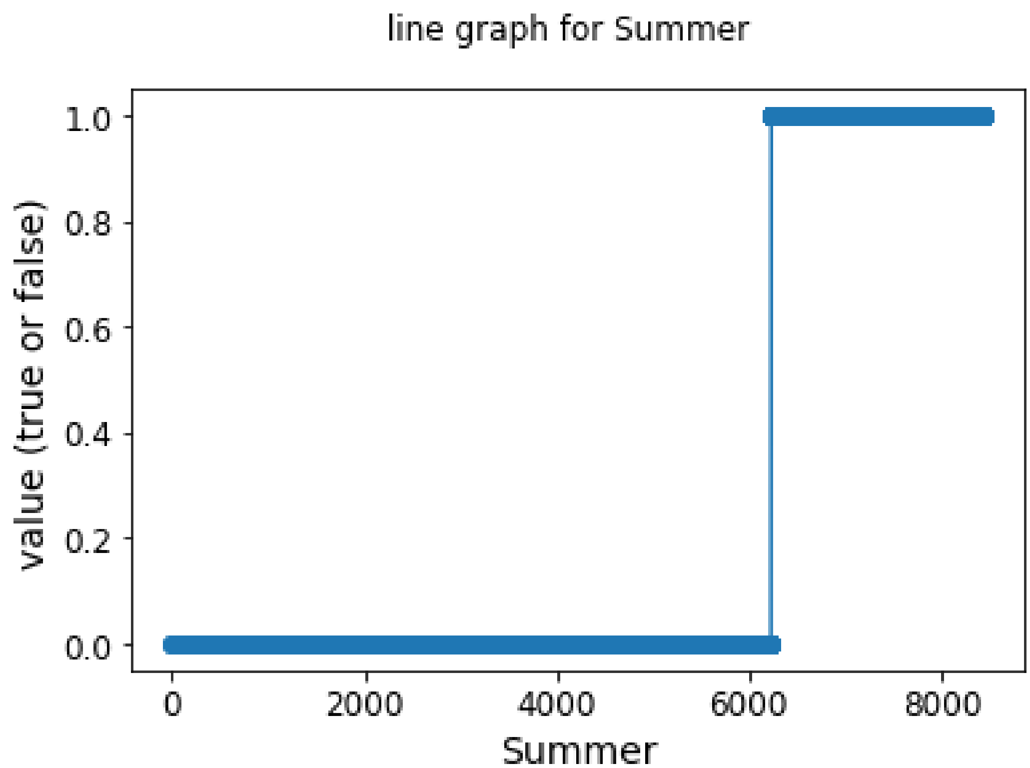

line graph for Solar Radiation (MJ/m2)
value (MJ/m2)
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
0
2000
4000
6000
8000
Solar Radiation

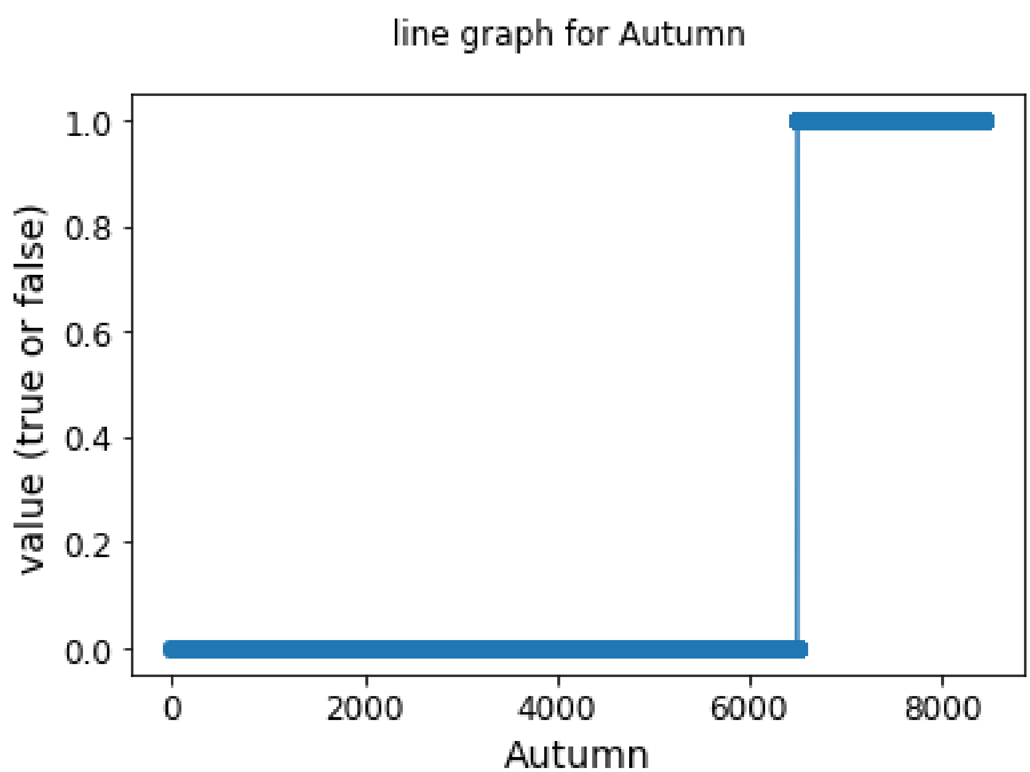

line graph for Rainfall(mm)
value (mm)
35
30
25
20
15
10
5
0
0
2000
4000
6000
8000
Rainfall

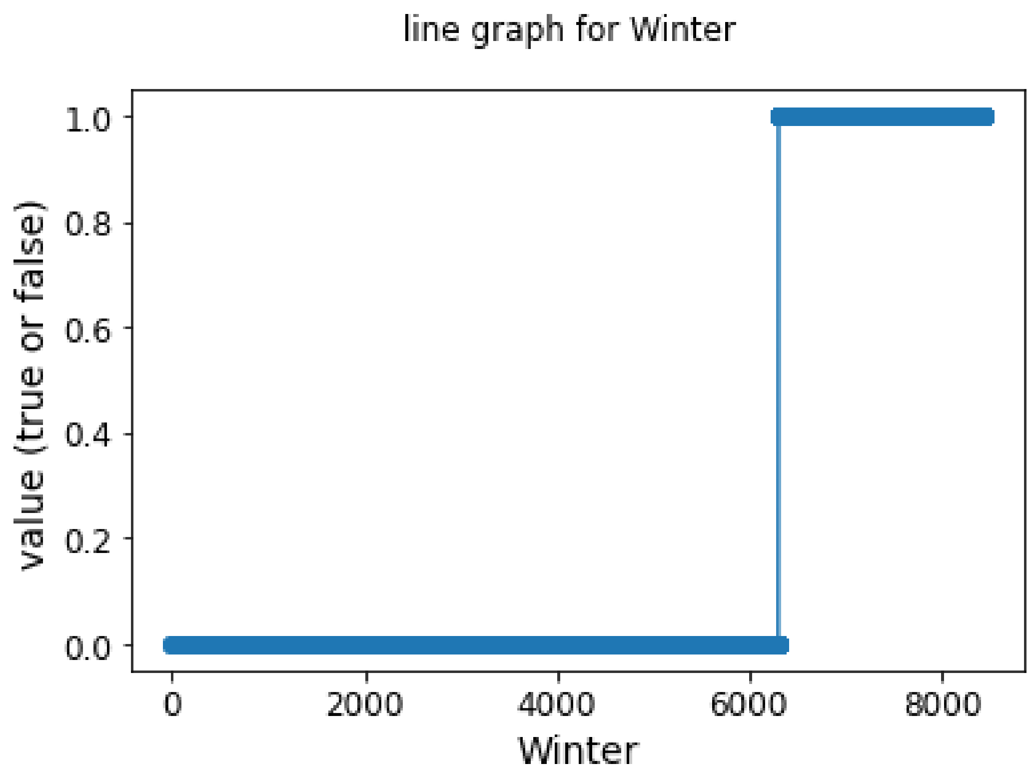

line graph for Snowfall (cm)
value (cm)
0
2
4
6
8
0
2000
4000
6000
8000
Snowfall


line graph for Holiday
value (true or false)
0.0
0.2
0.4
0.6
0.8
1.0
0
2000
4000
6000
8000
Holiday


line graph for Spring
1.0
0.8
0.6
0.4
0.2
0.0
value (true or false)
0
2000
4000
6000
8000
Spring


line graph for Summer
1.0
0.8
0.6
0.4
0.2
0.0
value (true or false)
0
2000
4000
6000
8000
Summer


line graph for Autumn
1.0
0.8
0.6
0.4
0.2
0.0
value (true or false)
0
2000
4000
6000
8000
Autumn



[78]:
```
### Your summary sentence about problematic entries
'''
'Temperature', 'Humidity', 'Wind speed',  'Visibility' contains obviously␣
↪impossible values like Humidity is less than 0,
and 'Rented Bike Count','Solar Radiation','Rainfall', 'Snowfall' contains␣
↪outliers.
'''
```

[78]:
```
"\n'Temperature', 'Humidity', 'Wind speed',  'Visibility' contains obviously
impossible values like Humidity is less than 0,\nand 'Rented Bike Count','Solar
Radiation','Rainfall', 'Snowfall' contains outliers.\n"
```

1.4.2 2.2 Imputation and Pre-Processing

Set any problematic values in the numerical data to `np.nan` and check that this has worked. Once this is done, specify a **sklearn *pipeline* that will perform imputation** to replace problematic entries (nan values) with an appropriate **median** value ***and* any other pre-processing** that you think should be used. Just specify the pipeline - do ***not*** run it now.

[79]:
```python
### Your code here

#fix  feature  1 (Temperature)
vals = rentals.iloc[:,3]
#set temperature larger than 50 to nan

bad_vals = vals >50
rentals.iloc[bad_vals,3] = np.nan
print(rentals.describe())
'''the max value of temperature becomes 39.4'''
```

```
                Date  Rented Bike Count         Hour  Temperature (C)  \
count  8465.000000        8465.000000  8465.000000      8464.000000
mean      0.711636         729.156999    11.509864        12.768951
std       0.453028         642.351166     6.921101        12.103538
min       0.000000           2.000000     0.000000       -17.800000
25%       0.000000         214.000000     6.000000         3.000000
50%       1.000000         542.000000    12.000000        13.500000
75%       1.000000        1084.000000    18.000000        22.700000
max       1.000000        3556.000000    24.000000        39.400000

       Humidity (%)  Wind speed (m/s)  Visibility (10m)  \
count   8465.000000       8464.000000       8465.000000
mean      58.161607          1.962169       1433.477141
std       20.713601         21.744979        609.596083
min      -26.000000          0.000000       -678.000000
25%       42.000000          0.900000        935.000000
50%       57.000000          1.500000       1689.000000
75%       74.000000          2.300000       2000.000000
max      309.000000       2000.000000       2000.000000

       Dew point temperature (C)  Solar Radiation (MJ/m2)  Rainfall(mm)  \
count                8464.000000              8465.000000   8440.000000
mean                    3.945558                 0.567868      0.149562
std                    13.243081                 0.868245      1.127177
min                   -30.600000                 0.000000      0.000000
25%                    -5.100000                 0.000000      0.000000
50%                     4.700000                 0.010000      0.000000
75%                    15.200000                 0.930000      0.000000
max                    27.200000                 3.520000     35.000000

       Snowfall (cm)      Holiday       Spring       Summer       Autumn  \
count    8442.000000  8465.000000  8465.000000  8465.000000  8465.000000
mean        0.077896     0.048198     0.255168     0.260839     0.228825
std         0.444649     0.214198     0.435982     0.439118     0.420101
min         0.000000     0.000000     0.000000     0.000000     0.000000
25%         0.000000     0.000000     0.000000     0.000000     0.000000
```

```
50%          0.000000     0.000000     0.000000     0.000000     0.000000
75%          0.000000     0.000000     1.000000     1.000000     0.000000
max          8.800000     1.000000     1.000000     1.000000     1.000000

            Winter
count  8465.000000
mean      0.255168
std       0.435982
min       0.000000
25%       0.000000
50%       0.000000
75%       1.000000
max       1.000000
```

[79]: 'the max value of temperature becomes 39.4'

[80]:
```python
#fix  feature  2 (Humidity)
vals = rentals.iloc[:,4]
#set humidity larger than 100 or less than 0 to nan
bad_vals = vals >100
rentals.iloc[bad_vals,4] = np.nan
bad_vals = vals < 0
rentals.iloc[bad_vals,4] = np.nan
print(rentals.describe())
'''the max value of temperature becomes 98 the least one become 0'''
```

```
              Date  Rented Bike Count         Hour  Temperature (C)  \
count  8465.000000        8465.000000  8465.000000      8464.000000
mean      0.711636         729.156999    11.509864        12.768951
std       0.453028         642.351166     6.921101        12.103538
min       0.000000           2.000000     0.000000       -17.800000
25%       0.000000         214.000000     6.000000         3.000000
50%       1.000000         542.000000    12.000000        13.500000
75%       1.000000        1084.000000    18.000000        22.700000
max       1.000000        3556.000000    24.000000        39.400000

       Humidity (%)  Wind speed (m/s)  Visibility (10m)  \
count   8461.000000       8464.000000       8465.000000
mean      58.161328          1.962169       1433.477141
std       20.478908         21.744979        609.596083
min        0.000000          0.000000       -678.000000
25%       42.000000          0.900000        935.000000
50%       57.000000          1.500000       1689.000000
75%       74.000000          2.300000       2000.000000
max       98.000000       2000.000000       2000.000000

       Dew point temperature (C)  Solar Radiation (MJ/m2)  Rainfall(mm)  \
count                8464.000000              8465.000000   8440.000000
```

```
mean                    3.945558                 0.567868      0.149562
std                    13.243081                 0.868245      1.127177
min                   -30.600000                 0.000000      0.000000
25%                    -5.100000                 0.000000      0.000000
50%                     4.700000                 0.010000      0.000000
75%                    15.200000                 0.930000      0.000000
max                    27.200000                 3.520000     35.000000

       Snowfall (cm)     Holiday       Spring       Summer       Autumn  \
count    8442.000000  8465.000000  8465.000000  8465.000000  8465.000000
mean        0.077896     0.048198     0.255168     0.260839     0.228825
std         0.444649     0.214198     0.435982     0.439118     0.420101
min         0.000000     0.000000     0.000000     0.000000     0.000000
25%         0.000000     0.000000     0.000000     0.000000     0.000000
50%         0.000000     0.000000     0.000000     0.000000     0.000000
75%         0.000000     0.000000     1.000000     1.000000     0.000000
max         8.800000     1.000000     1.000000     1.000000     1.000000

            Winter
count  8465.000000
mean      0.255168
std       0.435982
min       0.000000
25%       0.000000
50%       0.000000
75%       1.000000
max       1.000000
```

[80]: 'the max value of temperature becomes 98 the least one become 0'

[81]:
```python
#fix  feature  3 (Wind Speed)
vals = rentals.iloc[:,5]
#set wind speed larger than 30 to nan

bad_vals = vals >30
rentals.iloc[bad_vals,5] = np.nan
print(rentals.describe())
'''the max value of temperature becomes 7.4'''
```

```
              Date  Rented Bike Count         Hour  Temperature (C)  \
count  8465.000000        8465.000000  8465.000000      8464.000000
mean      0.711636         729.156999    11.509864        12.768951
std       0.453028         642.351166     6.921101        12.103538
min       0.000000           2.000000     0.000000       -17.800000
25%       0.000000         214.000000     6.000000         3.000000
50%       1.000000         542.000000    12.000000        13.500000
75%       1.000000        1084.000000    18.000000        22.700000
max       1.000000        3556.000000    24.000000        39.400000
```

```
       Humidity (%)  Wind speed (m/s)  Visibility (10m)  \
count   8461.000000       8463.000000       8465.000000
mean      58.161328          1.726078       1433.477141
std       20.478908          1.034324        609.596083
min        0.000000          0.000000       -678.000000
25%       42.000000          0.900000        935.000000
50%       57.000000          1.500000       1689.000000
75%       74.000000          2.300000       2000.000000
max       98.000000          7.400000       2000.000000

       Dew point temperature (C)  Solar Radiation (MJ/m2)  Rainfall(mm)  \
count                8464.000000              8465.000000   8440.000000
mean                    3.945558                 0.567868      0.149562
std                    13.243081                 0.868245      1.127177
min                   -30.600000                 0.000000      0.000000
25%                    -5.100000                 0.000000      0.000000
50%                     4.700000                 0.010000      0.000000
75%                    15.200000                 0.930000      0.000000
max                    27.200000                 3.520000     35.000000

       Snowfall (cm)      Holiday       Spring       Summer       Autumn  \
count    8442.000000  8465.000000  8465.000000  8465.000000  8465.000000
mean        0.077896     0.048198     0.255168     0.260839     0.228825
std         0.444649     0.214198     0.435982     0.439118     0.420101
min         0.000000     0.000000     0.000000     0.000000     0.000000
25%         0.000000     0.000000     0.000000     0.000000     0.000000
50%         0.000000     0.000000     0.000000     0.000000     0.000000
75%         0.000000     0.000000     1.000000     1.000000     0.000000
max         8.800000     1.000000     1.000000     1.000000     1.000000

            Winter
count  8465.000000
mean      0.255168
std       0.435982
min       0.000000
25%       0.000000
50%       0.000000
75%       1.000000
max       1.000000
```

[81]: 'the max value of temperature becomes 7.4'

[82]:
```
#fix  feature  4 (Visibility)
vals = rentals.iloc[:,6]
#set visibility smaller than 0 to nan
```

```python
bad_vals = vals < 0
rentals.iloc[bad_vals,6] = np.nan
print(rentals.describe())
'''the min value of temperature becomes 1'''
```

```
             Date  Rented Bike Count         Hour  Temperature (C)  \
count  8465.000000        8465.000000  8465.000000      8464.000000
mean      0.711636         729.156999    11.509864        12.768951
std       0.453028         642.351166     6.921101        12.103538
min       0.000000           2.000000     0.000000       -17.800000
25%       0.000000         214.000000     6.000000         3.000000
50%       1.000000         542.000000    12.000000        13.500000
75%       1.000000        1084.000000    18.000000        22.700000
max       1.000000        3556.000000    24.000000        39.400000

       Humidity (%)  Wind speed (m/s)  Visibility (10m)  \
count   8461.000000       8463.000000       8464.000000
mean      58.161328          1.726078       1433.726607
std       20.478908          1.034324        609.199826
min        0.000000          0.000000          1.000000
25%       42.000000          0.900000        935.000000
50%       57.000000          1.500000       1689.500000
75%       74.000000          2.300000       2000.000000
max       98.000000          7.400000       2000.000000

       Dew point temperature (C)  Solar Radiation (MJ/m2)  Rainfall(mm)  \
count                8464.000000              8465.000000   8440.000000
mean                    3.945558                 0.567868      0.149562
std                    13.243081                 0.868245      1.127177
min                   -30.600000                 0.000000      0.000000
25%                    -5.100000                 0.000000      0.000000
50%                     4.700000                 0.010000      0.000000
75%                    15.200000                 0.930000      0.000000
max                    27.200000                 3.520000     35.000000

       Snowfall (cm)      Holiday       Spring       Summer       Autumn  \
count    8442.000000  8465.000000  8465.000000  8465.000000  8465.000000
mean        0.077896     0.048198     0.255168     0.260839     0.228825
std         0.444649     0.214198     0.435982     0.439118     0.420101
min         0.000000     0.000000     0.000000     0.000000     0.000000
25%         0.000000     0.000000     0.000000     0.000000     0.000000
50%         0.000000     0.000000     0.000000     0.000000     0.000000
75%         0.000000     0.000000     1.000000     1.000000     0.000000
max         8.800000     1.000000     1.000000     1.000000     1.000000

            Winter
count  8465.000000
```

```
mean      0.255168
std       0.435982
min       0.000000
25%       0.000000
50%       0.000000
75%       1.000000
max       1.000000
```

[82]: 'the min value of temperature becomes 1'

[83]:
```python
rentals.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 8465 entries, 0 to 8759
Data columns (total 16 columns):
 #   Column                     Non-Null Count  Dtype
---  ------                     --------------  -----
 0   Date                       8465 non-null   int64
 1   Rented Bike Count          8465 non-null   int64
 2   Hour                       8465 non-null   int64
 3   Temperature (C)            8464 non-null   float64
 4   Humidity (%)               8461 non-null   float64
 5   Wind speed (m/s)           8463 non-null   float64
 6   Visibility (10m)           8464 non-null   float64
 7   Dew point temperature (C)  8464 non-null   float64
 8   Solar Radiation (MJ/m2)    8465 non-null   float64
 9   Rainfall(mm)               8440 non-null   float64
 10  Snowfall (cm)              8442 non-null   float64
 11  Holiday                    8465 non-null   int64
 12  Spring                     8465 non-null   int64
 13  Summer                     8465 non-null   int64
 14  Autumn                     8465 non-null   int64
 15  Winter                     8465 non-null   int64
dtypes: float64(8), int64(8)
memory usage: 1.1 MB
```

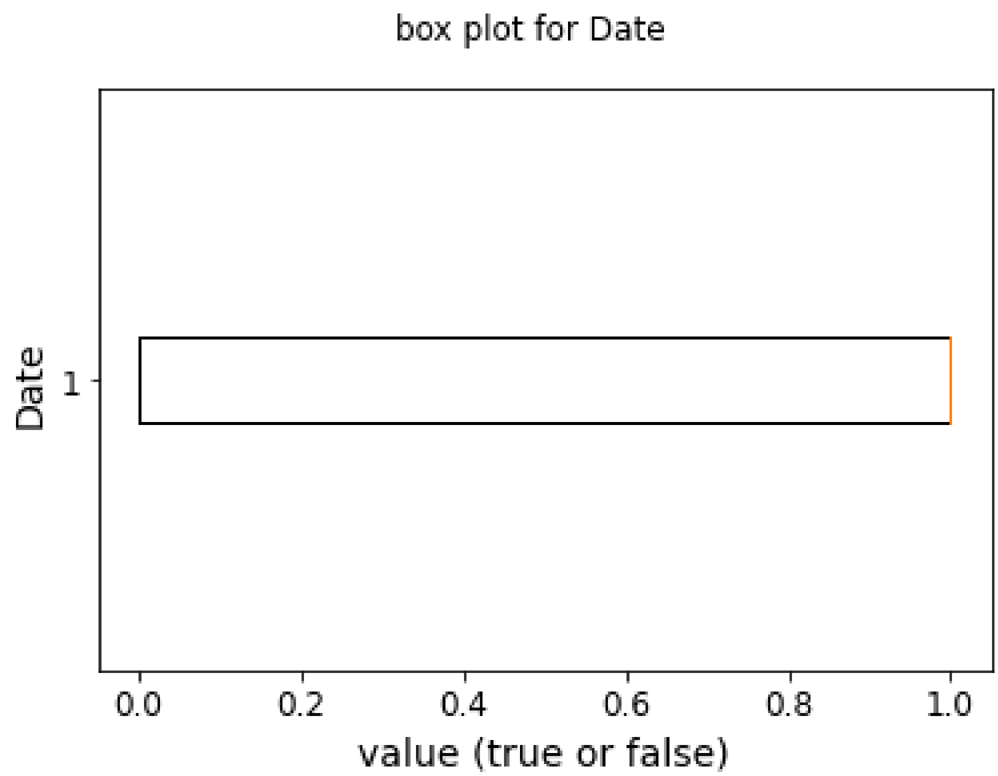
[84]:
```python
# #remove all the outliers
cols = [1,3,4,5,6,8,9,10] # one or more

rentals.iloc[:,cols] = rentals.iloc[:,cols].mask(rentals.iloc[:,cols].
 ↪sub(rentals.iloc[:,cols].mean()).div(rentals.iloc[:,cols].std()).abs().gt(2))
rentals = rentals.replace({pd.NA: np.nan})
rentals.dropna(subset=['Rented Bike Count'], inplace=True)

print(rentals.describe())
```

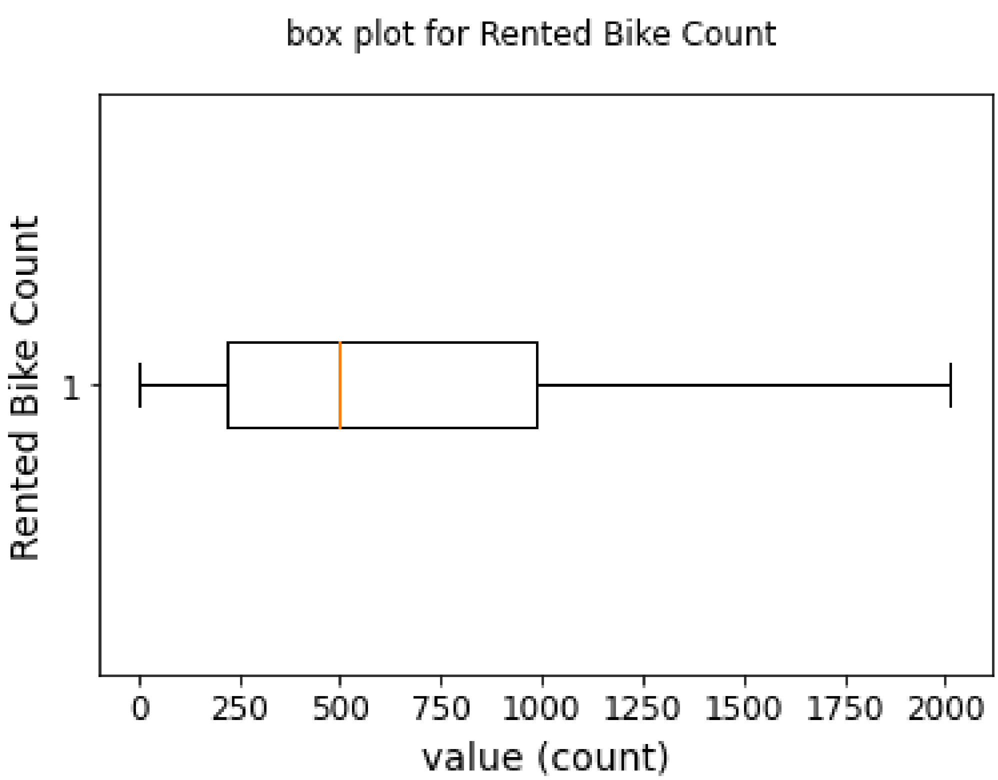
```
              Date  Rented Bike Count         Hour  Temperature (C)  \
count  7995.000000        7995.000000  7995.000000      7826.000000
```

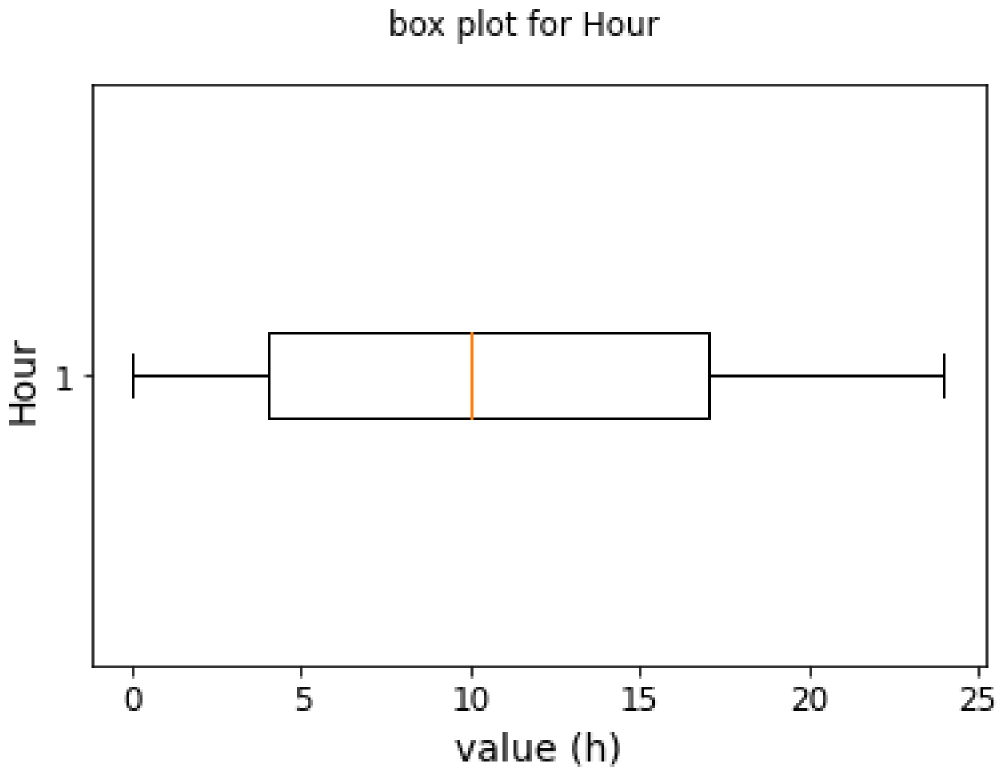
```
mean       0.707942          633.357098    11.162727         12.525428
std        0.454737          515.596976     6.913719         11.597524
min        0.000000            2.000000     0.000000        -11.400000
25%        0.000000          203.000000     5.000000          3.000000
50%        1.000000          483.000000    11.000000         12.600000
75%        1.000000          987.000000    17.000000         21.900000
max        1.000000         2013.000000    24.000000         36.900000

       Humidity (%)  Wind speed (m/s)  Visibility (10m)  \
count   7901.000000       7611.000000       7803.000000
mean      59.049361          1.573591       1450.338203
std       20.237735          0.866371        588.993597
min       18.000000          0.000000        216.000000
25%       43.000000          0.900000        962.000000
50%       58.000000          1.400000       1697.000000
75%       75.000000          2.200000       2000.000000
max       98.000000          3.700000       2000.000000

       Dew point temperature (C)  Solar Radiation (MJ/m2)  Rainfall(mm)  \
count                7994.000000              7364.000000   7824.000000
mean                    3.428246                 0.376310      0.038816
std                    13.361059                 0.620164      0.215025
min                   -30.600000                 0.000000      0.000000
25%                    -6.000000                 0.000000      0.000000
50%                     3.900000                 0.000000      0.000000
75%                    14.800000                 0.550000      0.000000
max                    27.200000                 2.300000      2.400000

       Snowfall (cm)      Holiday       Spring       Summer       Autumn  \
count    7745.000000  7995.000000  7995.000000  7995.000000  7995.000000
mean        0.015507     0.049781     0.257536     0.244653     0.227642
std         0.099461     0.217506     0.437304     0.429908     0.419337
min         0.000000     0.000000     0.000000     0.000000     0.000000
25%         0.000000     0.000000     0.000000     0.000000     0.000000
50%         0.000000     0.000000     0.000000     0.000000     0.000000
75%         0.000000     0.000000     1.000000     0.000000     0.000000
max         0.900000     1.000000     1.000000     1.000000     1.000000

            Winter
count  7995.000000
mean      0.270169
std       0.444075
min       0.000000
25%       0.000000
50%       0.000000
75%       1.000000
max       1.000000
```

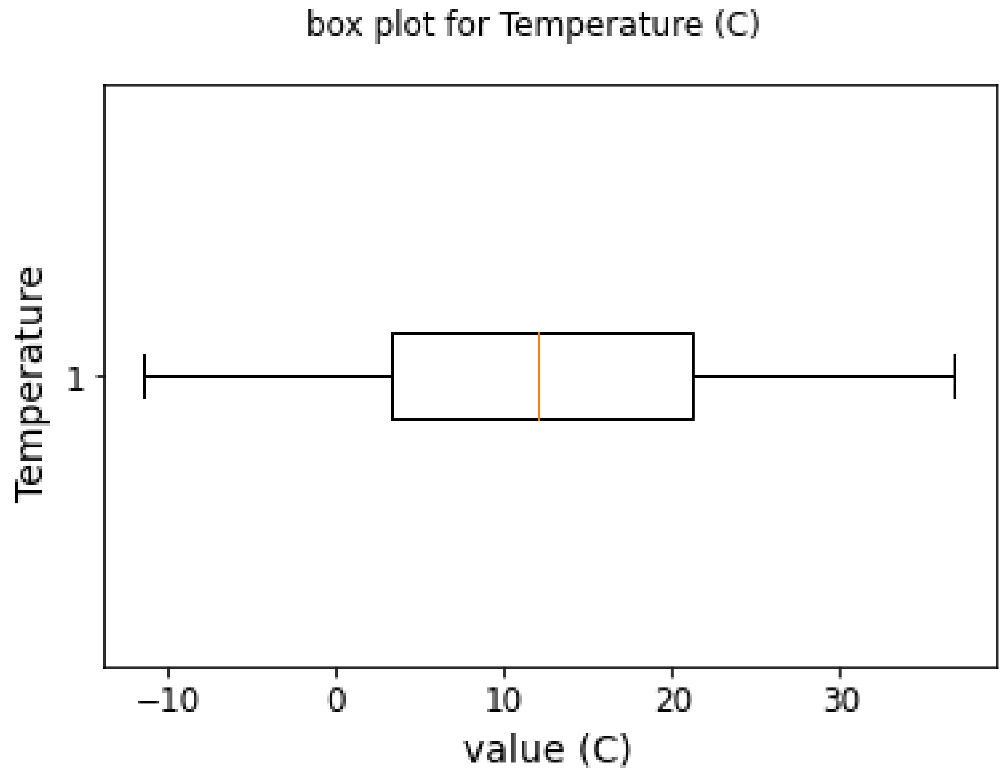
[85]:

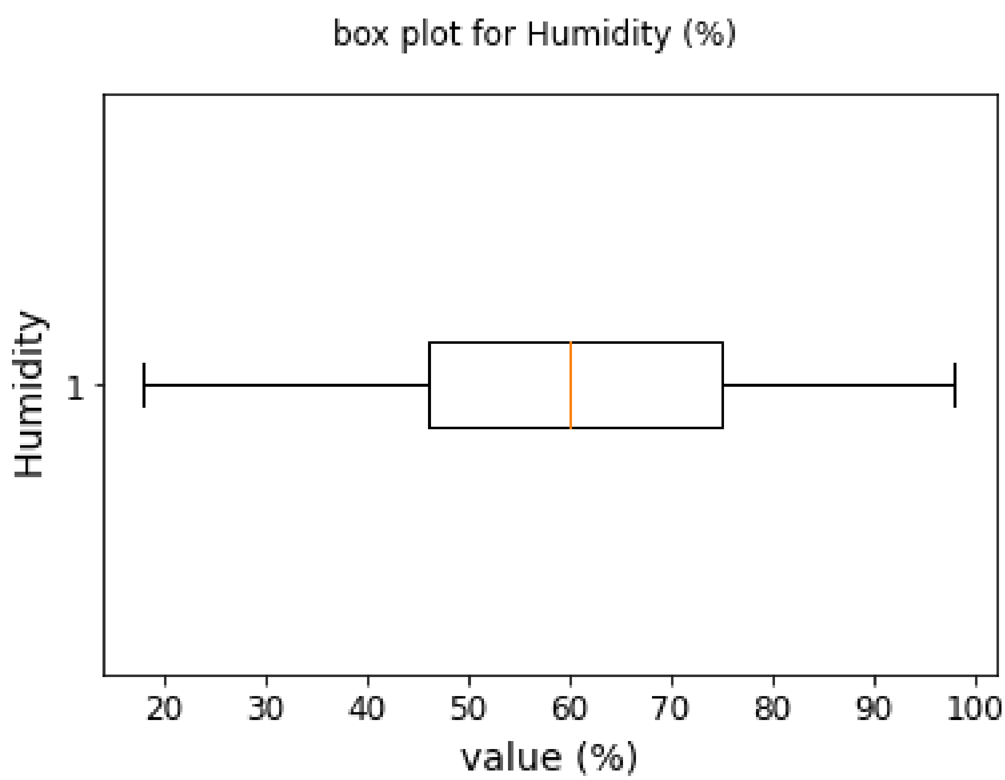
```
#remove nan row before boxplotting
#this is just for plotting, but we want to impute it later. So we use a copy

rentals_copy = rentals.dropna()


### Your code here
#visualise using boxplots
for n in range (rentals_copy.shape[1]):
    plt.boxplot(rentals_copy.iloc[:,n],vert =False)
    plt.suptitle( f'{"box plot for " +rentals_copy.columns[n]}'  )
    column_splited = rentals_copy.columns[n].split("(")
#     print(column_splited)
    plt.ylabel(f'{column_splited[0]}')
    #if the column name has the unit, add unit to x axis
    #or just print 'value'
    if len(column_splited) > 1:
        plt.xlabel("value ("+ f'{column_splited[1]}')
    else:
        # to make the lable mre clearer, manually deal with the unit
        if column_splited[0] == 'Rented Bike Count':
            plt.xlabel("value (count)")
        elif column_splited[0] == 'Hour':
            plt.xlabel("value (h)")

        else:
            plt.xlabel("value (true or false)")

    plt.show()
```

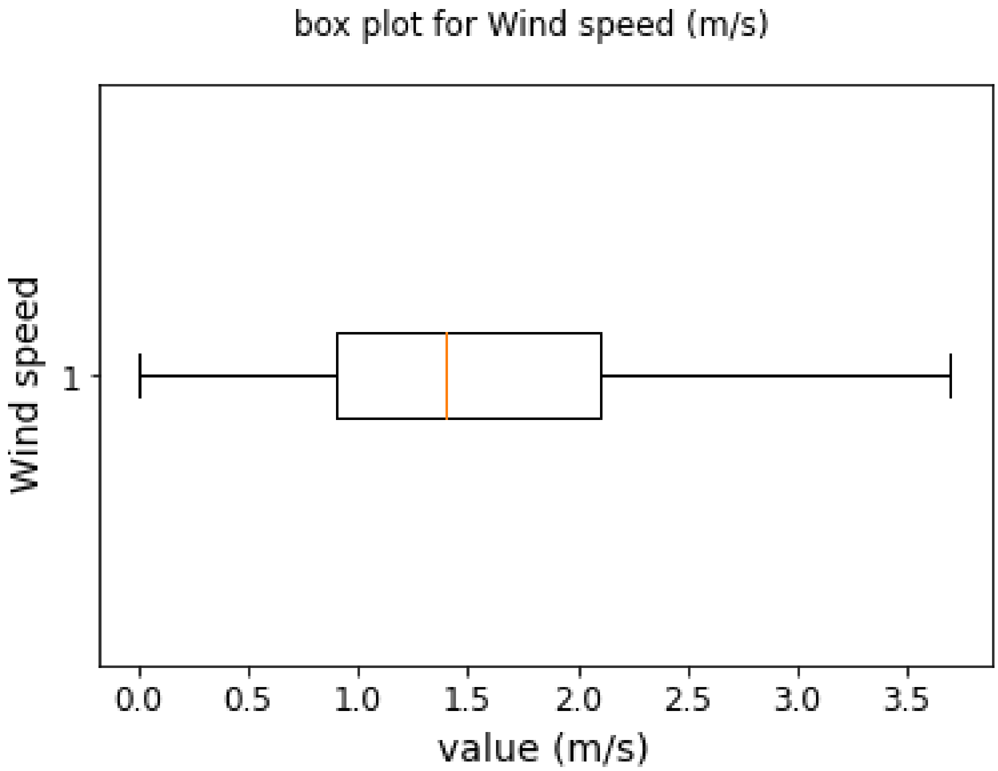

box plot for Date
Date
1
0.0
0.2
0.4
0.6
0.8
1.0
value (true or false)

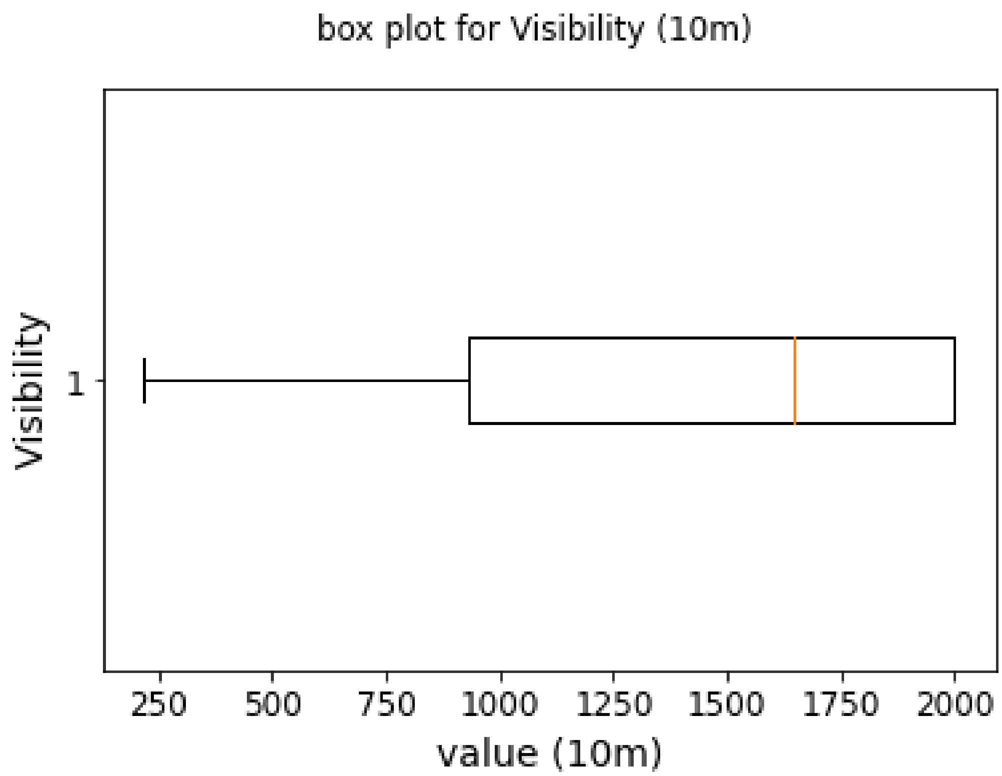

box plot for Rented Bike Count
Rented Bike Count
1
0
250
500
750
1000
1250
1500
1750
2000
value (count)

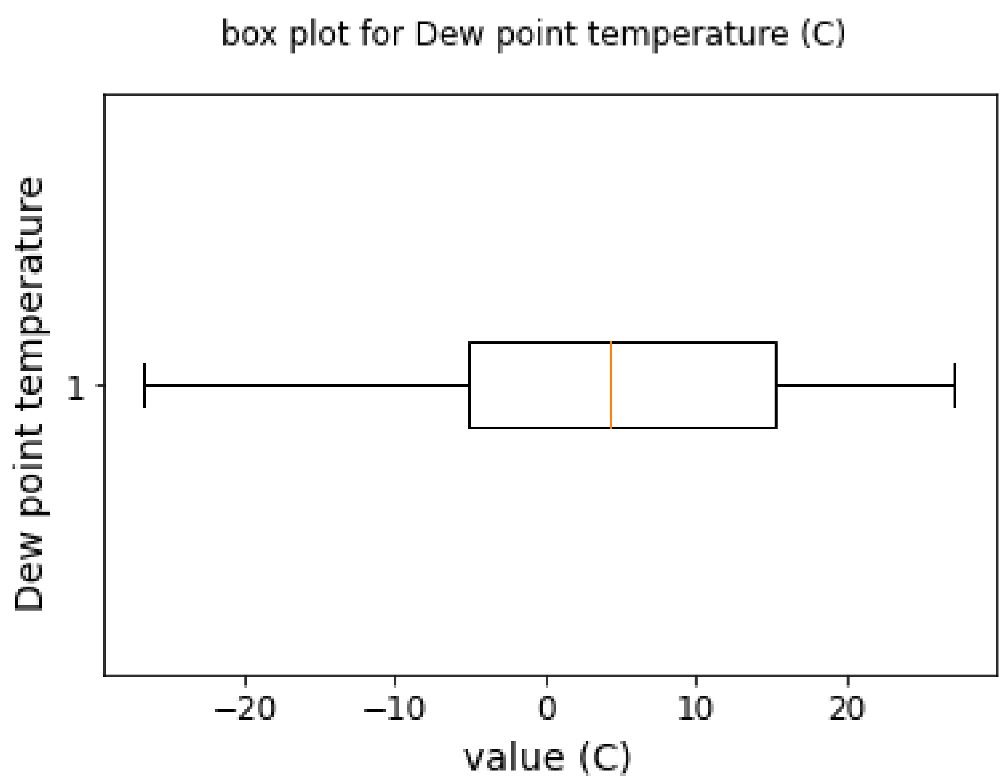

box plot for Hour
Hour
1
0
5
10
15
20
25
value (h)

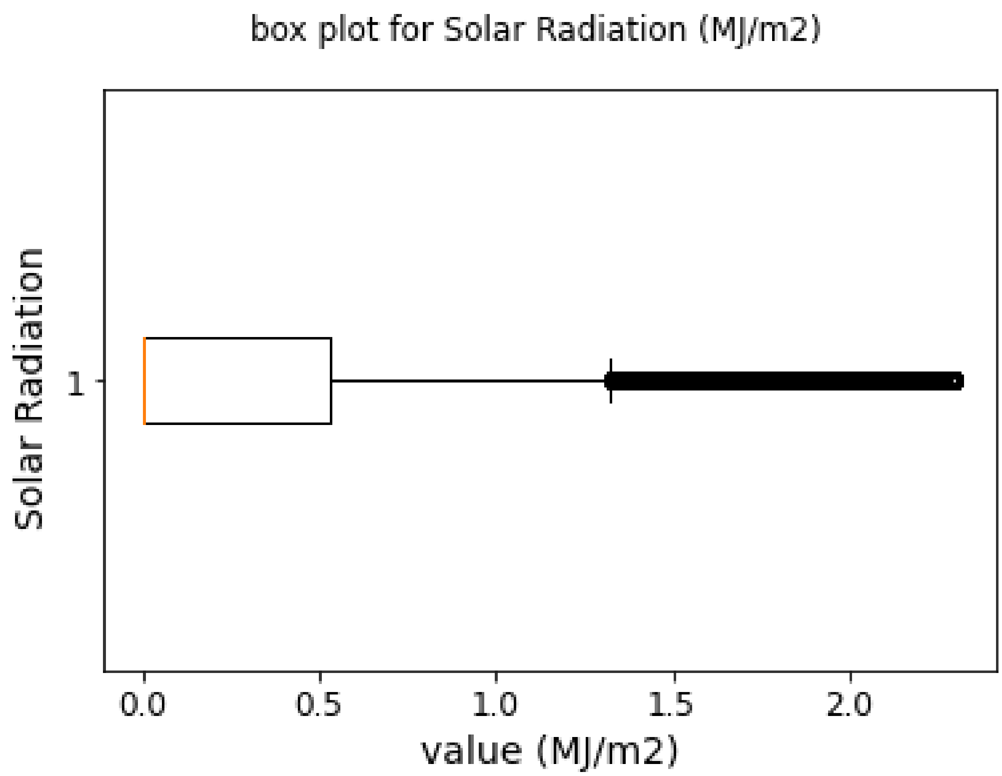

box plot for Temperature (C)
Temperature
1
−10
0
10
20
30
value (C)

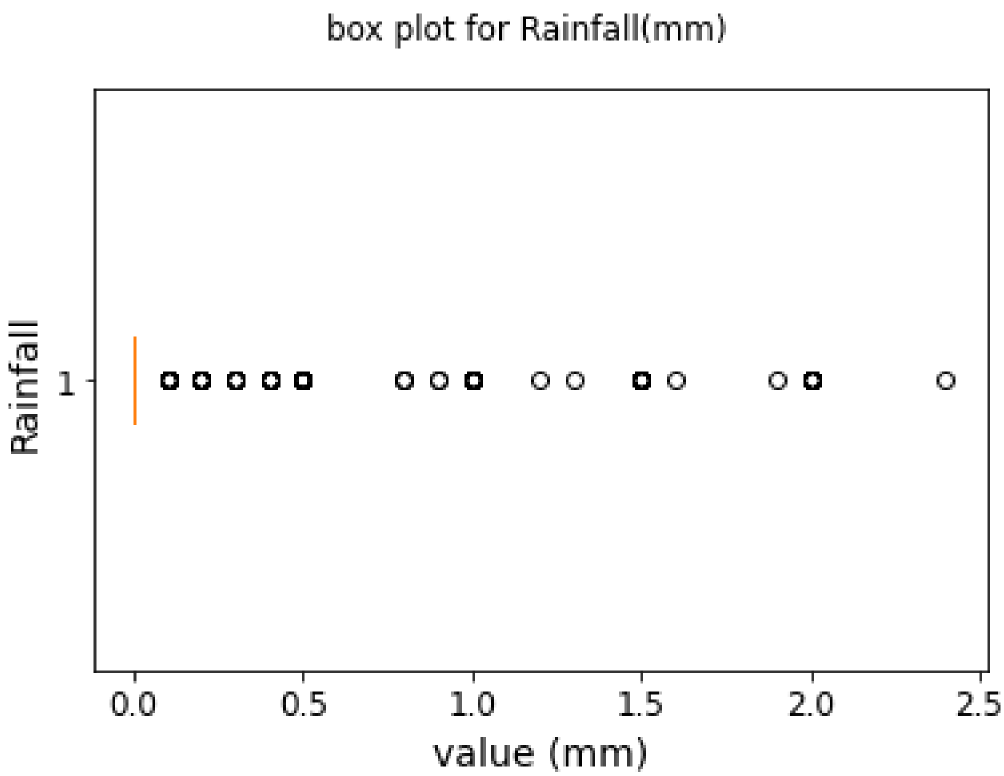

box plot for Humidity (%)
Humidity
1
20
30
40
50
60
70
80
90
100
value (%)

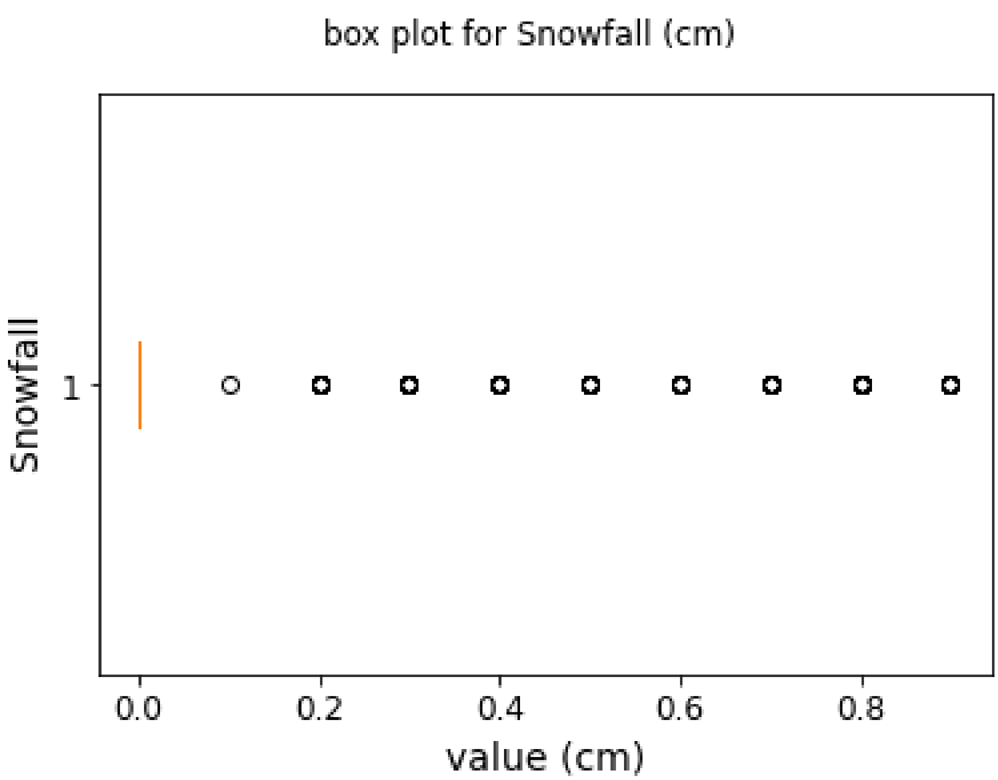

box plot for Wind speed (m/s)
Wind speed
1
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
value (m/s)

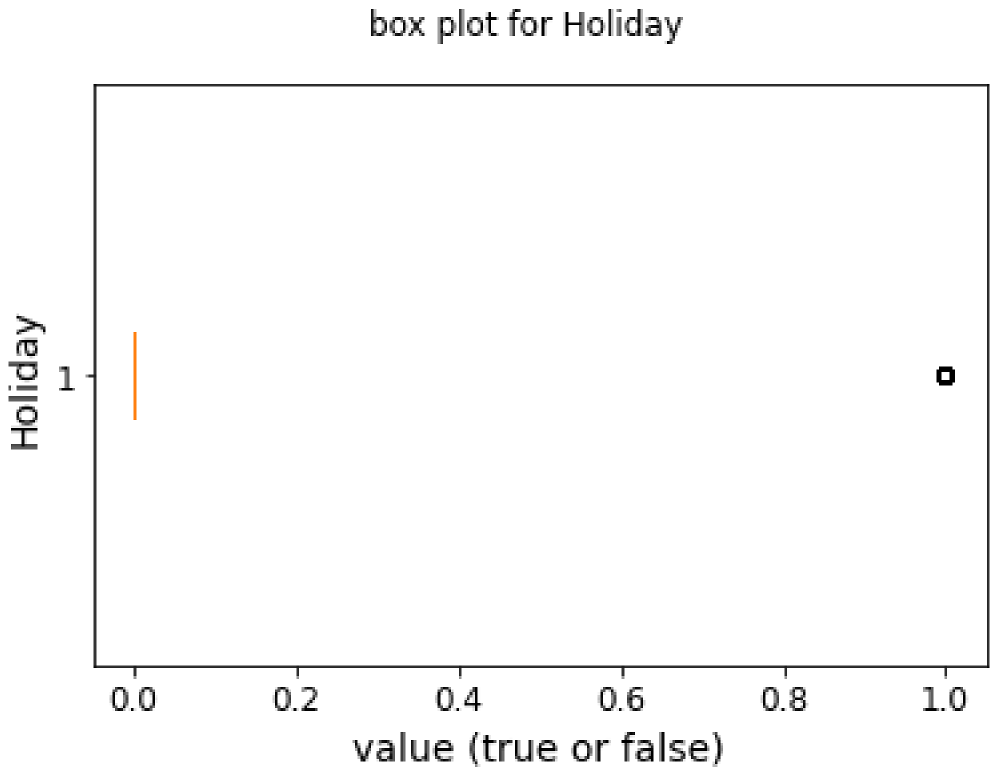

box plot for Visibility (10m)
Visibility
1
250
500
750
1000
1250
1500
1750
2000
value (10m)

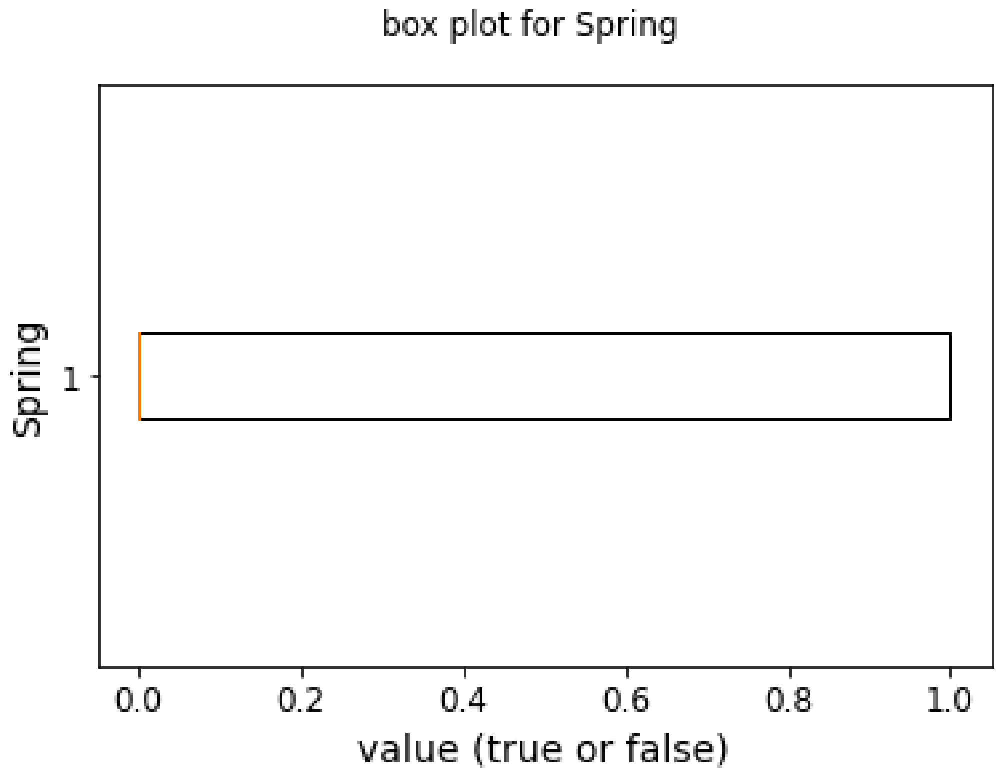

box plot for Dew point temperature (C)
Dew point temperature
1
−20
−10
0
10
20
value (C)

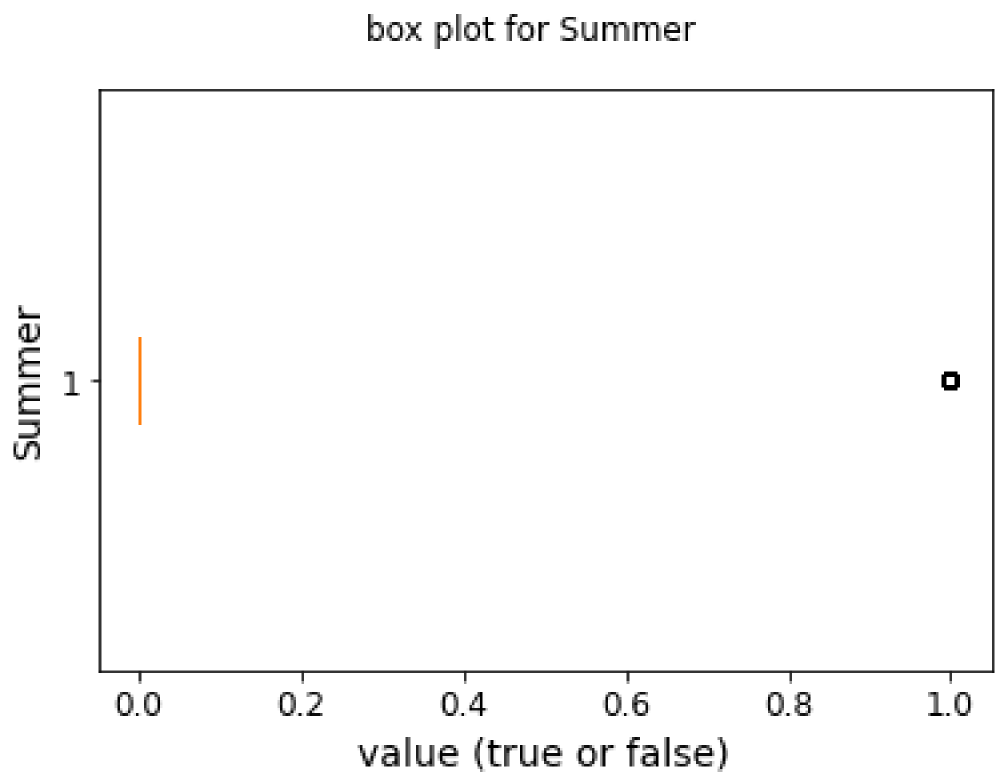

box plot for Solar Radiation (MJ/m2)
Solar Radiation
1
0.0
0.5
1.0
1.5
2.0
value (MJ/m2)

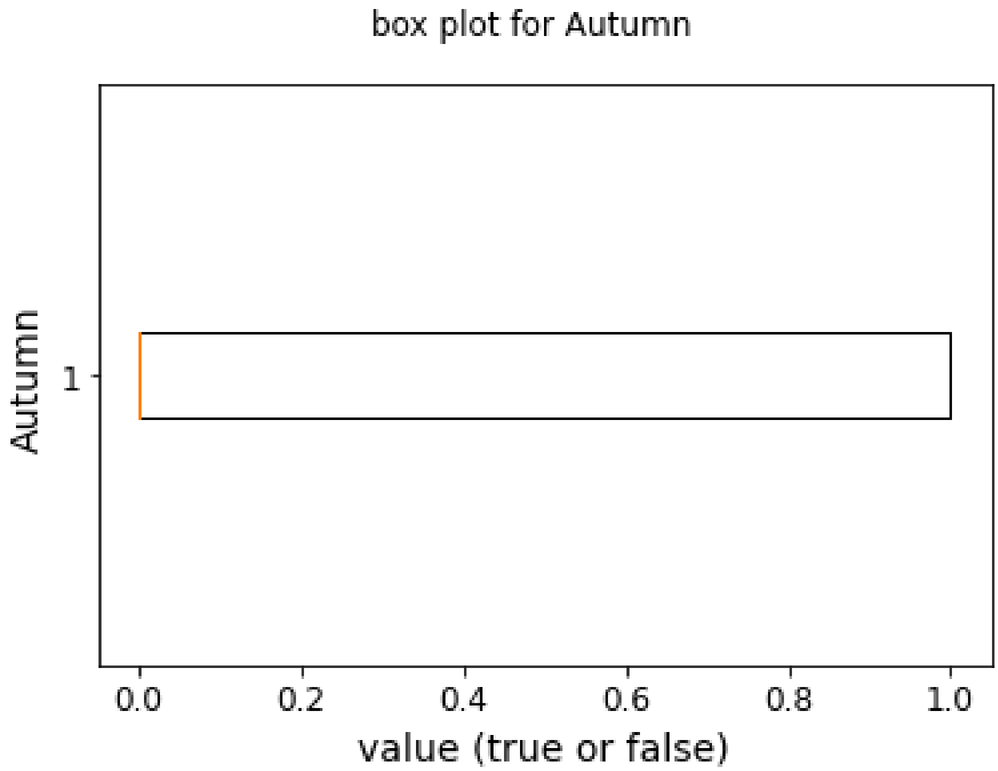

box plot for Rainfall(mm)
Rainfall
1
0.0
0.5
1.0
1.5
2.0
2.5
value (mm)

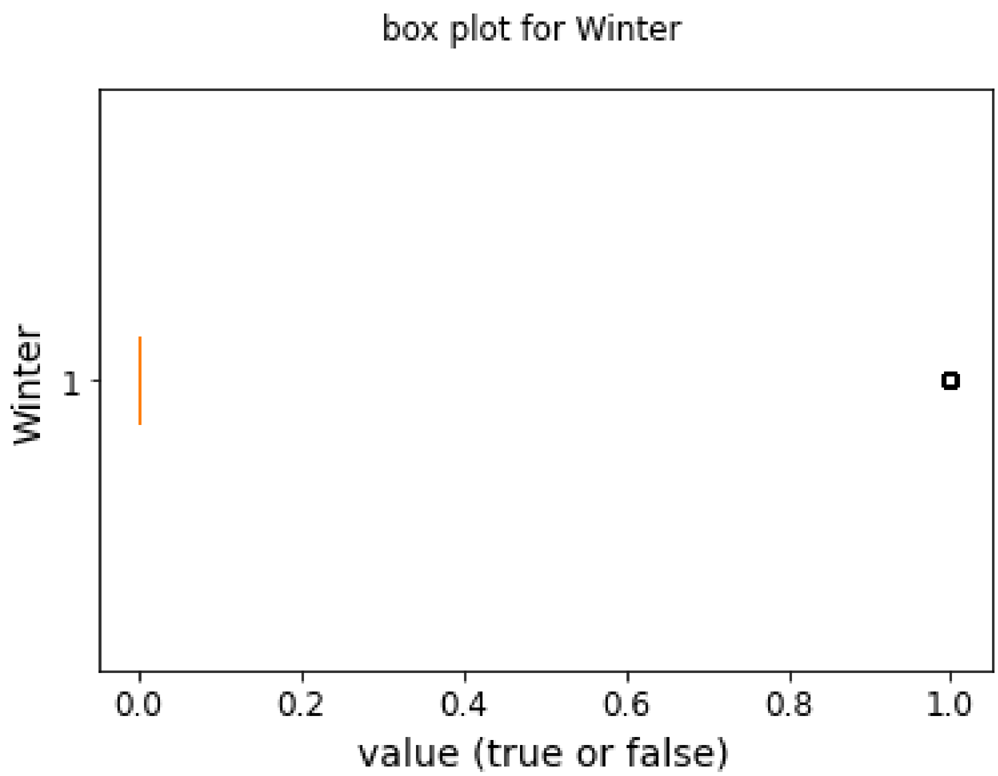

box plot for Snowfall (cm)
Snowfall
1
0.0
0.2
0.4
0.6
0.8
value (cm)


box plot for Holiday
Holiday
1
0.0
0.2
0.4
0.6
0.8
1.0
value (true or false)


box plot for Spring
Spring
1
0.0
0.2
0.4
0.6
0.8
1.0
value (true or false)


box plot for Summer
Summer
1
0.0
0.2
0.4
0.6
0.8
1.0
value (true or false)

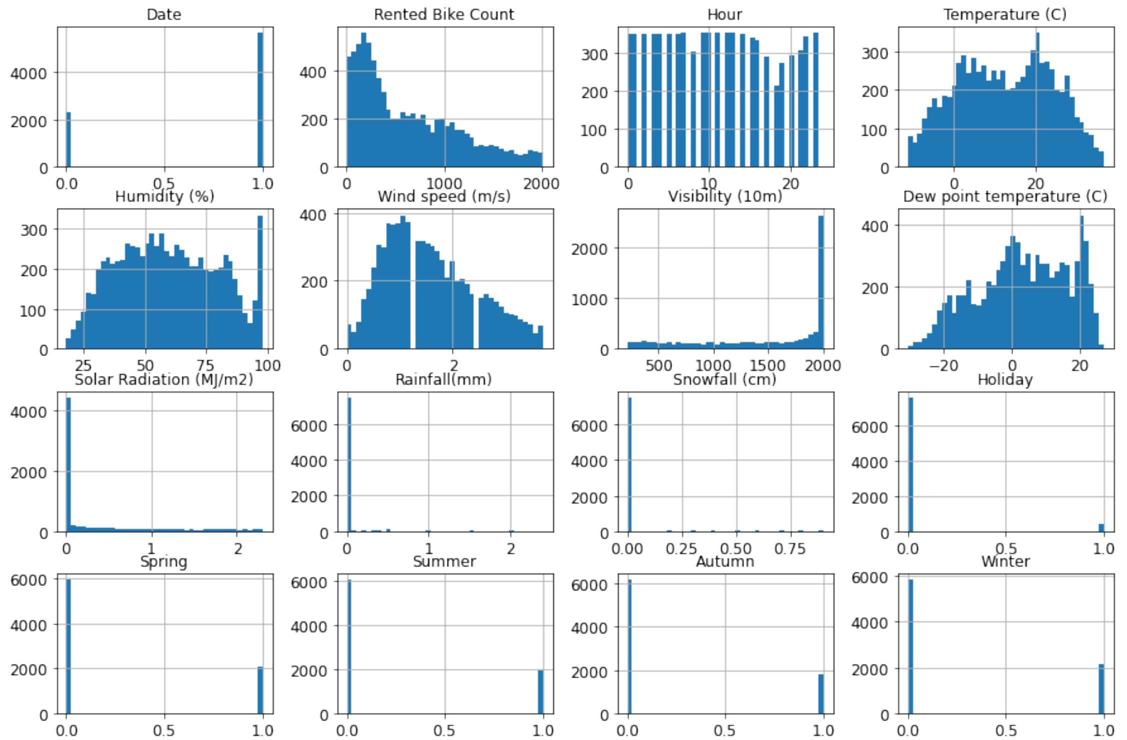

box plot for Autumn
Autumn
1
0.0
0.2
0.4
0.6
0.8
1.0
value (true or false)



[86]:
```
#plot again to see if it performs right
print(np.sum(rentals.isna() ))
rentals.hist(bins=40,figsize=(15,10))
'''it's good'''
```

```
Date                         0
Rented Bike Count            0
Hour                         0
Temperature (C)            169
Humidity (%)                94
Wind speed (m/s)           384
Visibility (10m)           192
Dew point temperature (C)    1
Solar Radiation (MJ/m2)    631
Rainfall(mm)               171
Snowfall (cm)              250
Holiday                      0
Spring                       0
Summer                       0
Autumn                       0
Winter                       0
dtype: int64
```

[86]: "it's good"



[87]:
```python
#Once this is done, specify a sklearn pipeline that will perform imputation to
 ↪replace problematic entries (nan values) with an appropriate
#median value and any other pre-processing that you think should be used. Just
 ↪specify the pipeline - do not run it now.
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.impute import SimpleImputer

#we use a simpleImputer to set nan to median and use StandardScaler to scale
 ↪the data
process_pl = Pipeline( [('imputer',SimpleImputer(missing_values=np.nan,strategy
 ↪= "median")),
                        ('std_scaler',StandardScaler() )])
```

1.4.3 2.3 Correlation

It is also useful to look at how strongly correlated the features are to the desired target (Rented Bike Count). Before anything else is done it is necessary to **fit and apply the pipeline** above to make a *temporary* version of the whole dataset that is pre-processed. **Why is it important to not use this version of the pre-processed data again?**

[88]:
```
### Your code here



#fit and apply the pipeliney
rentals_p = process_pl.fit_transform(rentals)
# print(np.sum(np.isnan(rentals_p) ))
# print(np.any(np.isnan(rentals_p)))



print(rentals_p.shape)
```

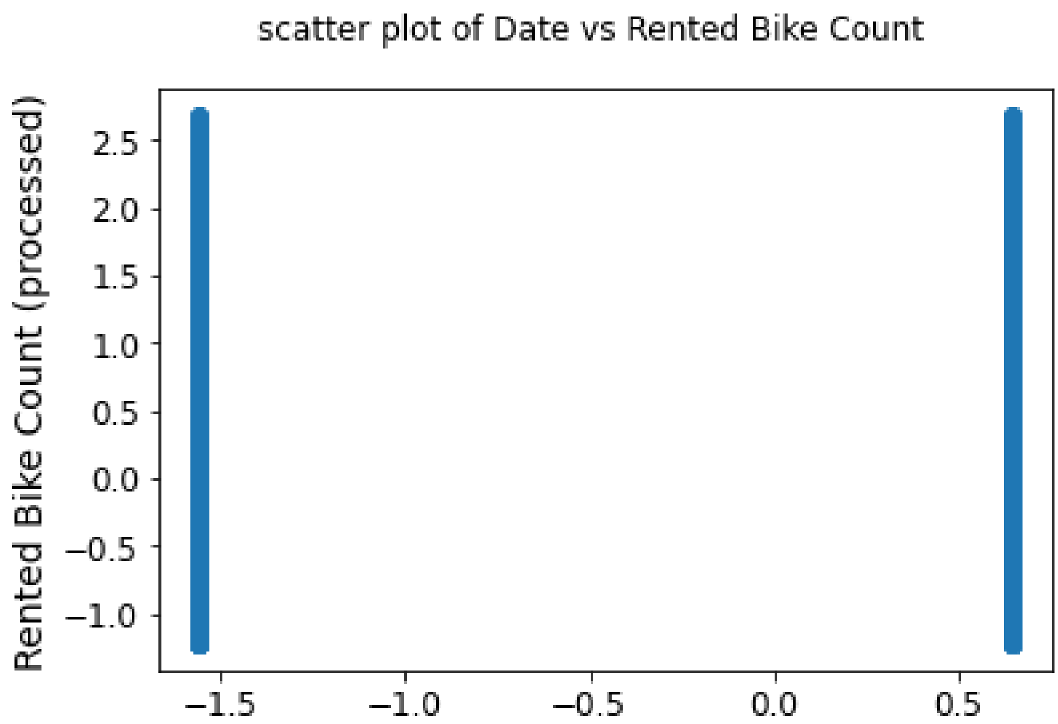
```
(7995, 16)
```

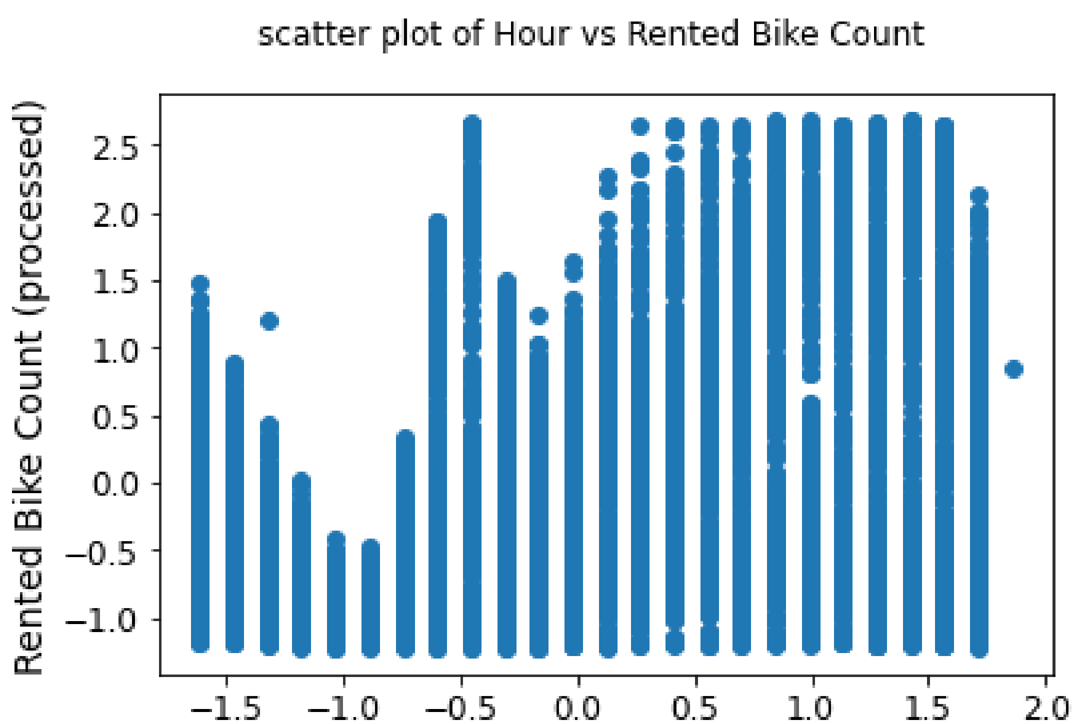
[89]:
```
### Your written answer here
'''We should protect a original version of dataset before pre-processed because␣
↪we may apply new preprosses later.'''
```

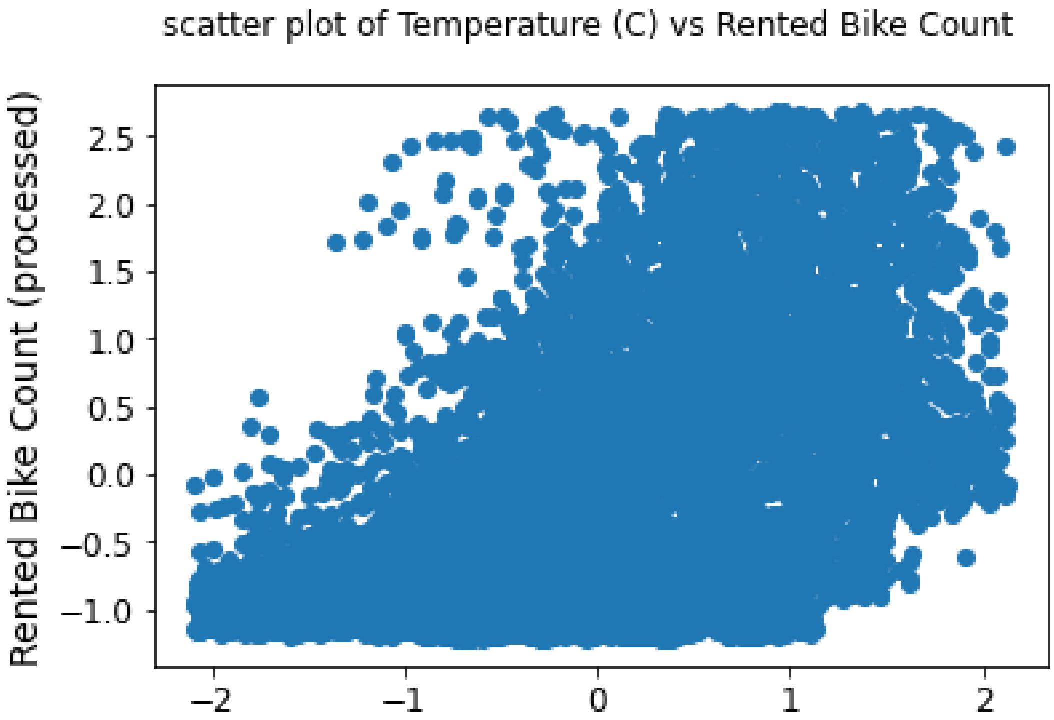
[89]:
```
'We should protect a original version of dataset before pre-processed because we
may apply new preprosses later.'
```

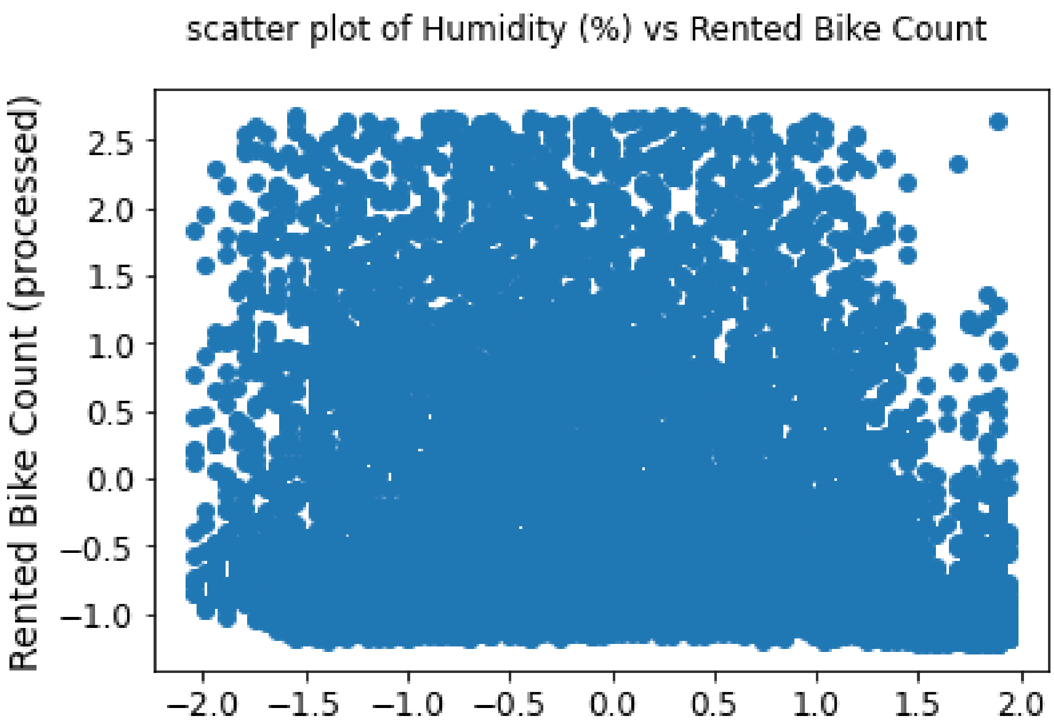
To visualise the strength of the relationships, display a **scatter plot** for each feature (separately) vs the target variable. Also **calculate the correlation** of each feature with the target (Hint: pandas function `corr()` or numpy `corrcoef()`). **Which 3 attributes are the most correlated with bike rentals?**

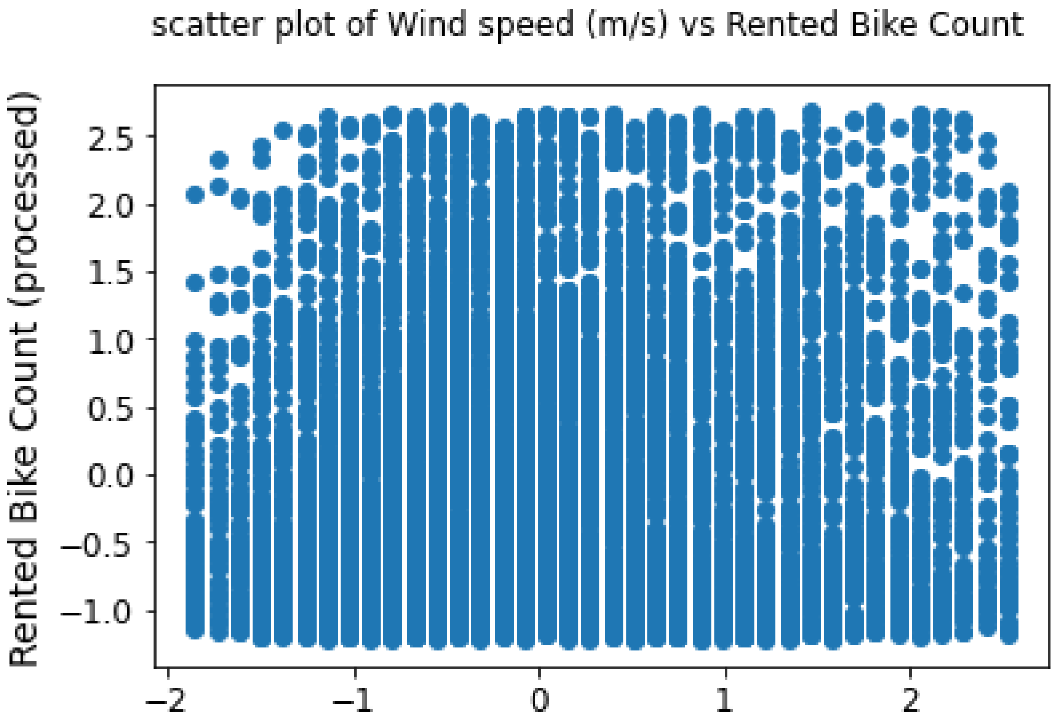
[90]:
```
### Your code here
#make a new df to store processed data
rentals_processed = pd.DataFrame(rentals_p, columns = rentals.columns)

# then split the X y
X = rentals_processed.drop(["Rented Bike Count"],axis =1)
y = rentals_processed["Rented Bike Count"].copy()
for column in X.columns:
    plt.suptitle(f'scatter plot of {column} vs Rented Bike Count')
    column_splited = rentals.columns[n].split("(")
#     print(column_splited)
    plt.ylabel("Rented Bike Count (processed)")
    #if the column name has the unit, add unit to x axis
    #or just print 'value'
    if len(column_splited) > 1:
        plt.xlabel(f'{column}')
    plt.scatter(X[column],y )
    plt.show()
# plt.plot(y_train,y_train,'r-o')
```

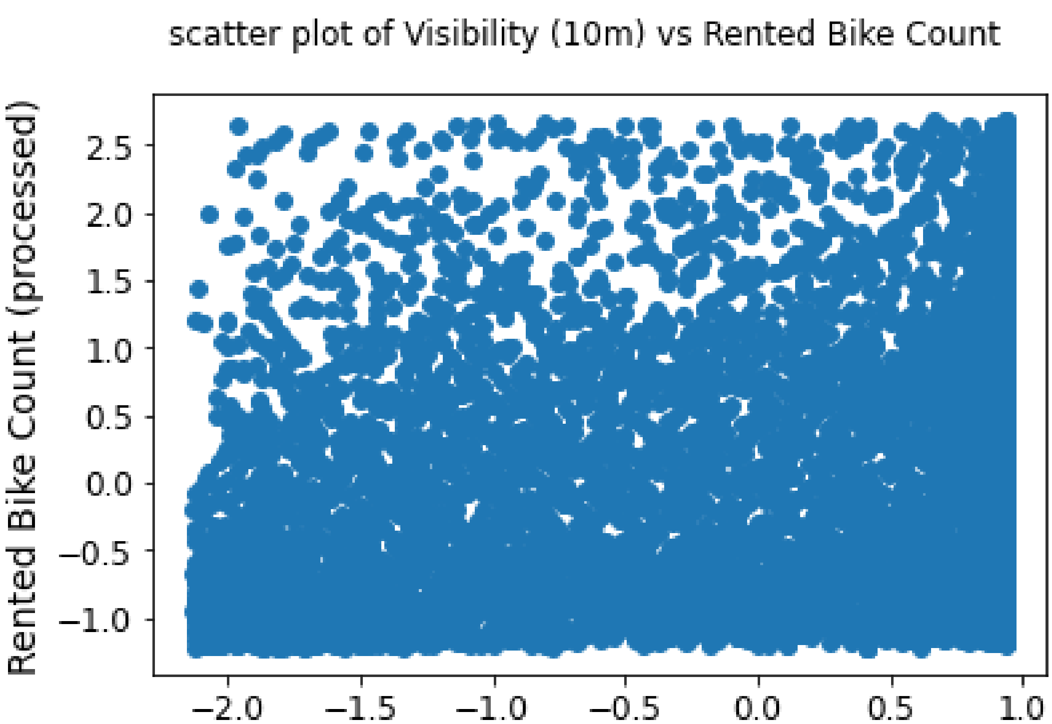

scatter plot of Date vs Rented Bike Count
Rented Bike Count (processed)
2.5
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−1.0
−0.5
0.0
0.5

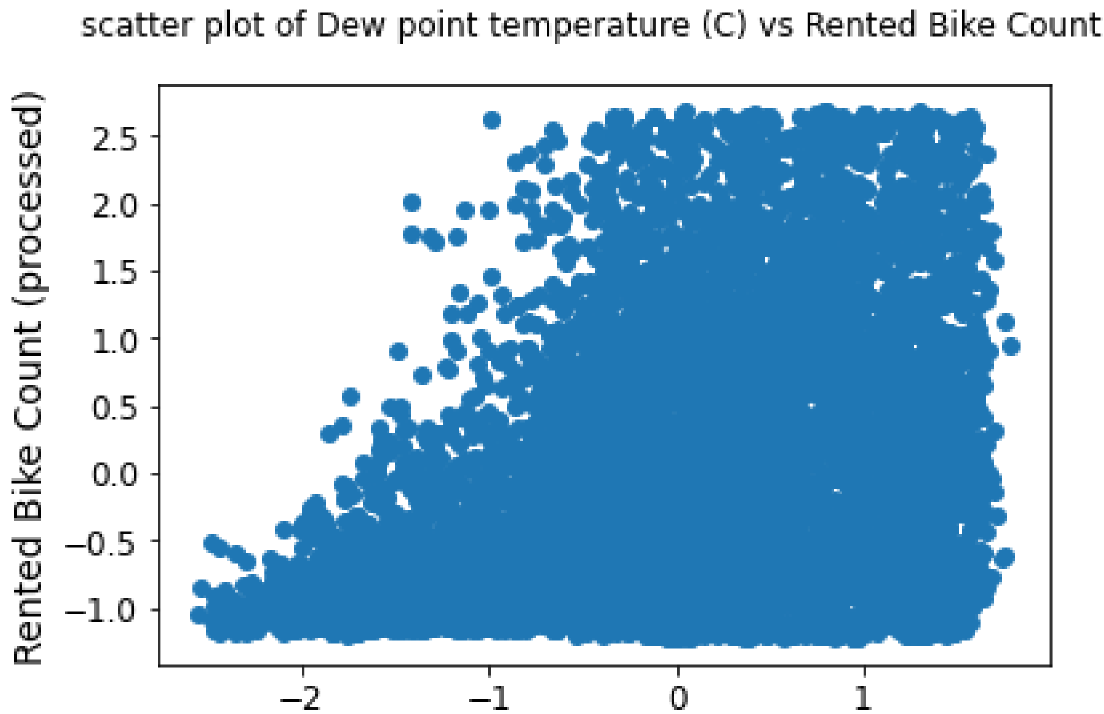

scatter plot of Hour vs Rented Bike Count
Rented Bike Count (processed)
2.5
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0

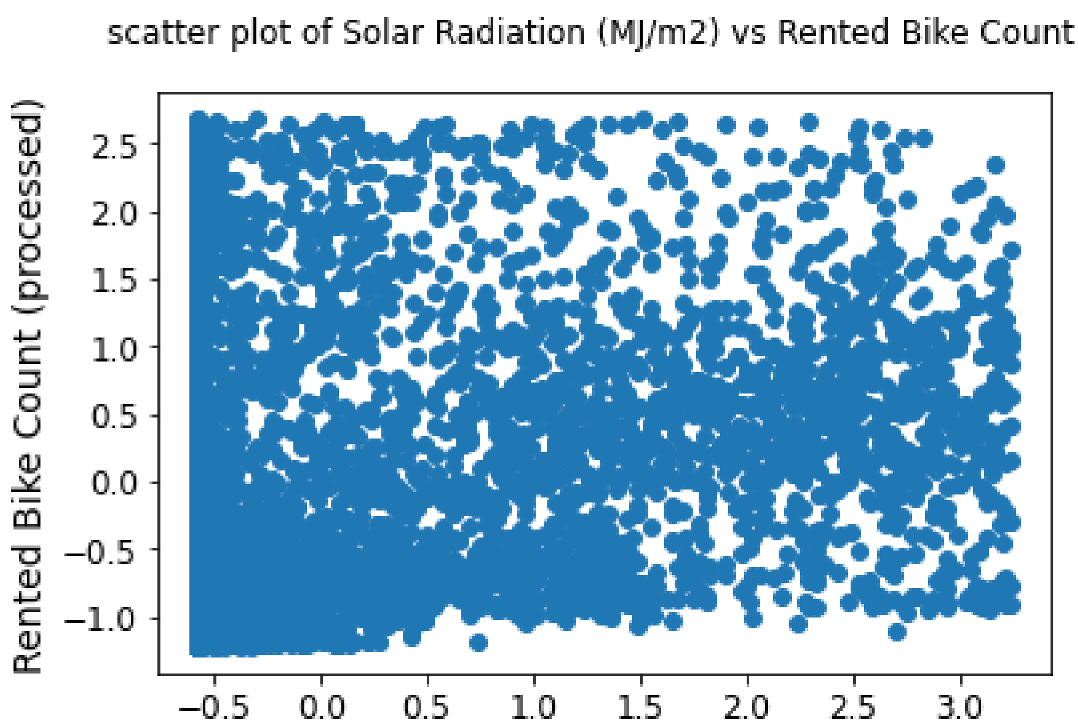

scatter plot of Temperature (C) vs Rented Bike Count
Rented Bike Count (processed)
2.5
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−2
−1
0
1
2

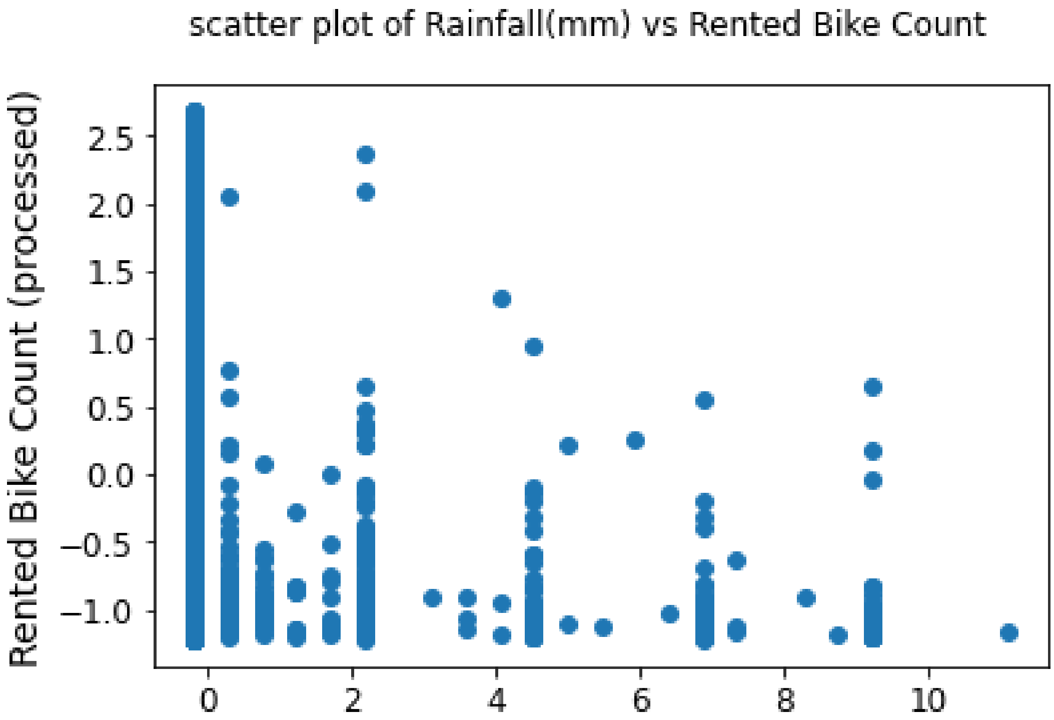

scatter plot of Humidity (%) vs Rented Bike Count
Rented Bike Count (processed)
2.5
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0

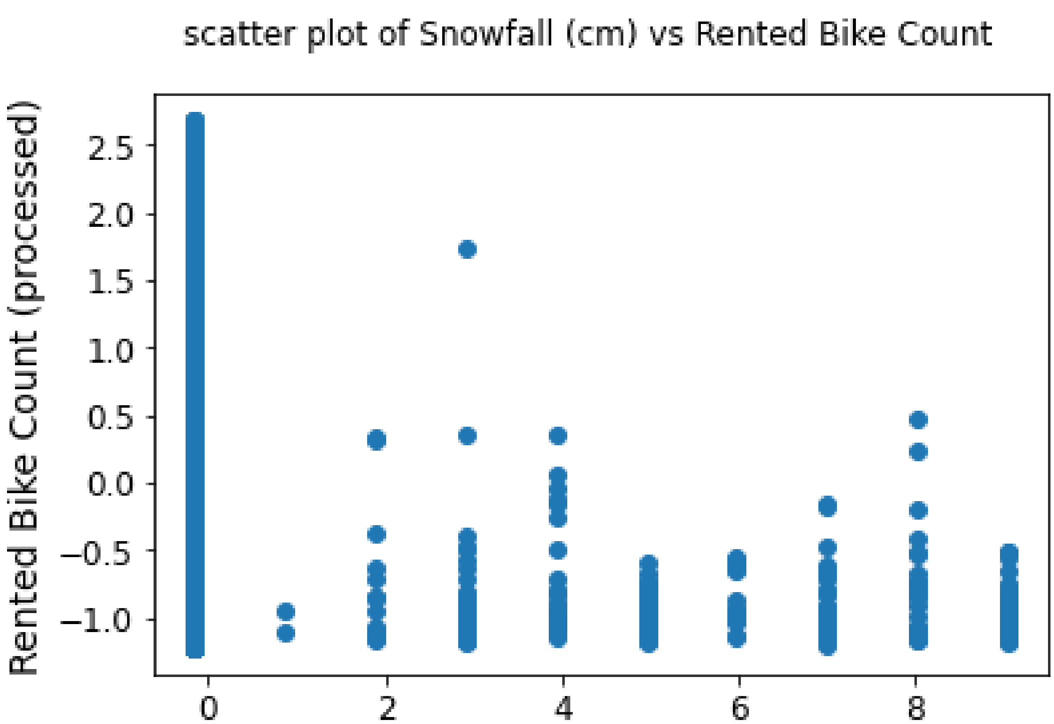

scatter plot of Wind speed (m/s) vs Rented Bike Count
Rented Bike Count (processed)
2.5
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−2
−1
0
1
2

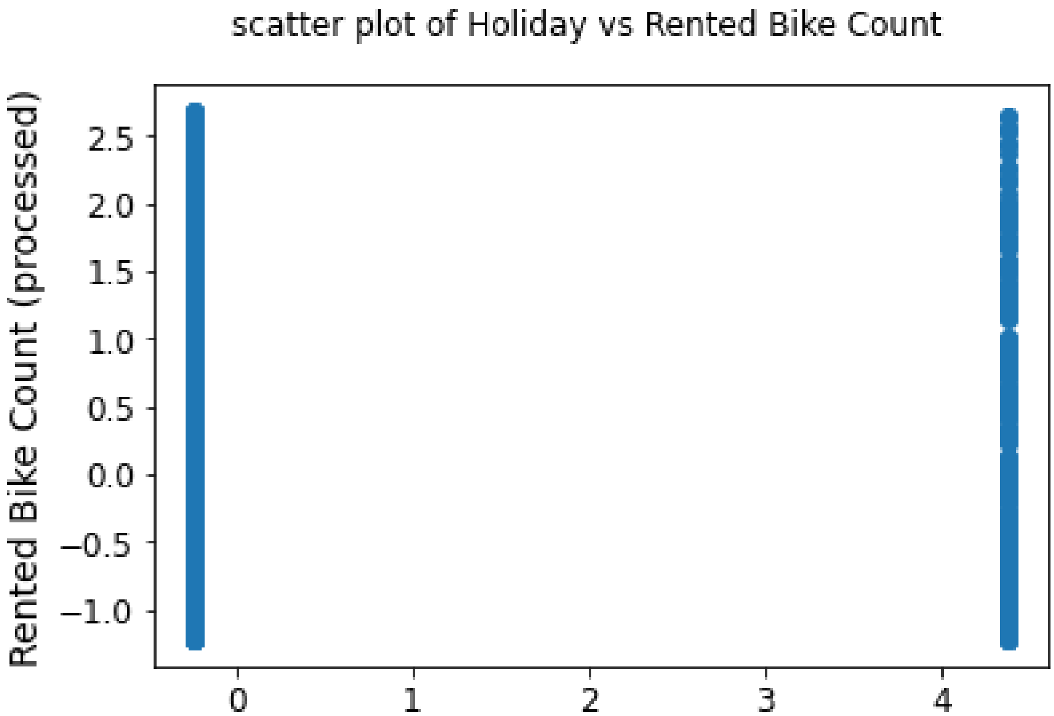

scatter plot of Visibility (10m) vs Rented Bike Count
Rented Bike Count (processed)
2.5
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0

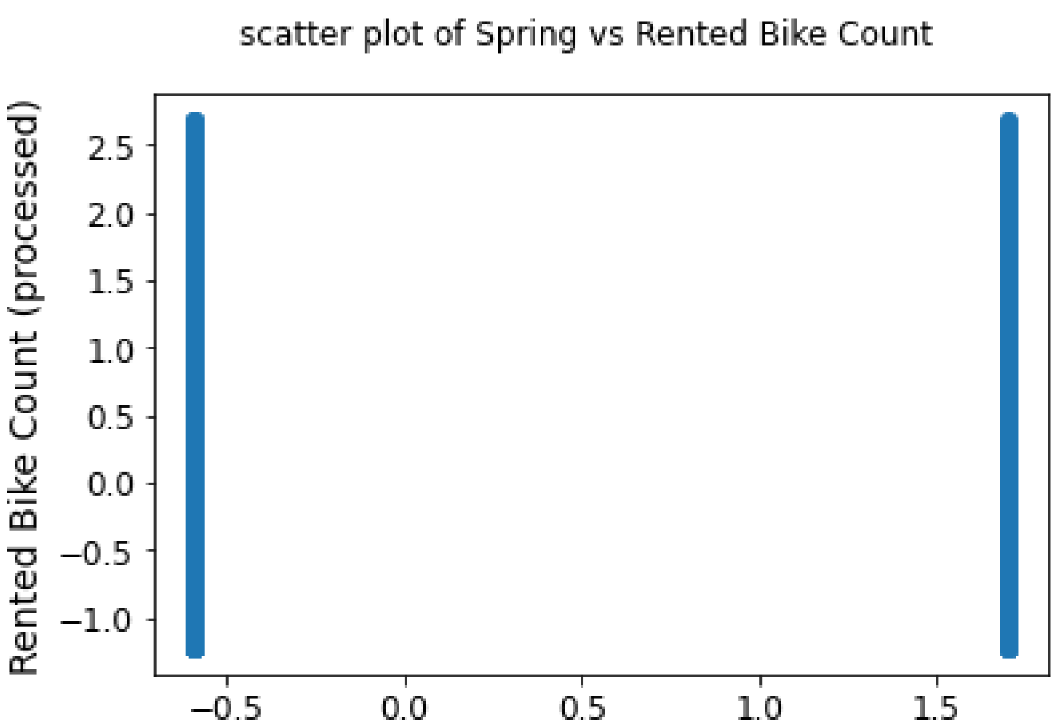

scatter plot of Dew point temperature (C) vs Rented Bike Count
Rented Bike Count (processed)
2.5
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−2
−1
0
1

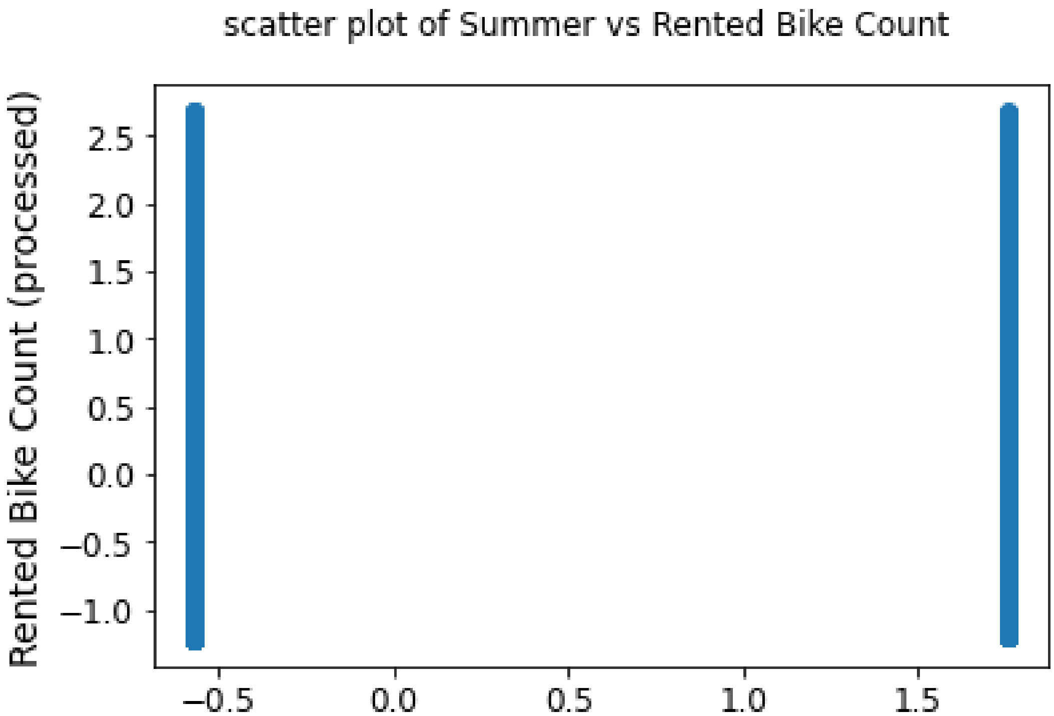

scatter plot of Solar Radiation (MJ/m2) vs Rented Bike Count
Rented Bike Count (processed)
2.5
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
2.5
3.0

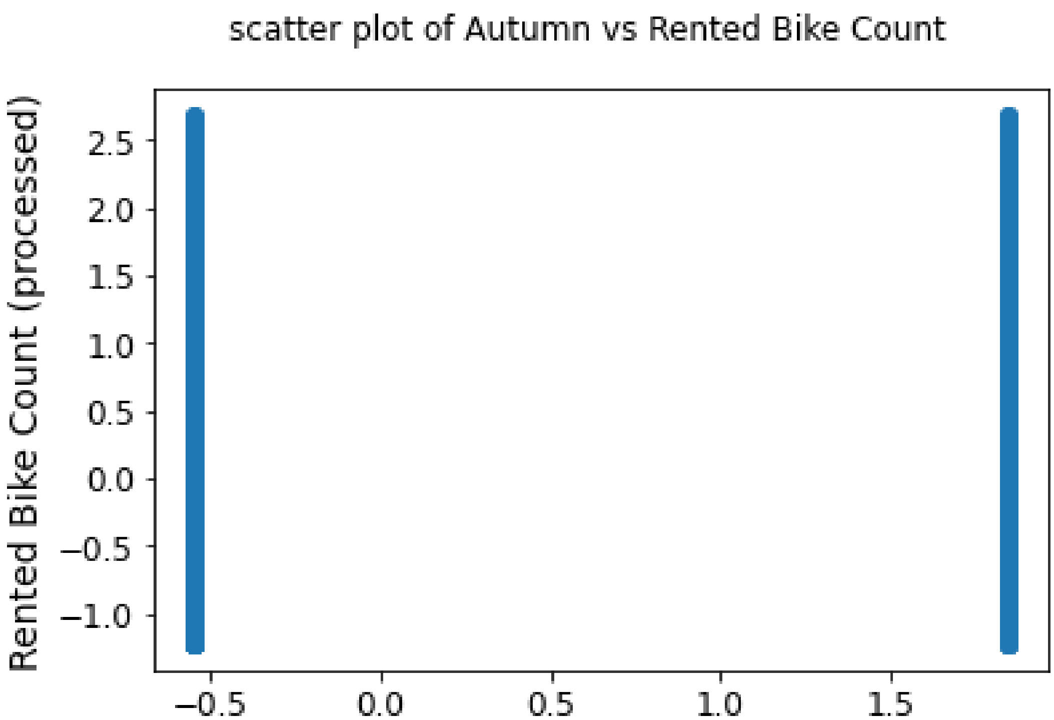

scatter plot of Rainfall(mm) vs Rented Bike Count
Rented Bike Count (processed)
2.5
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
0
2
4
6
8
10

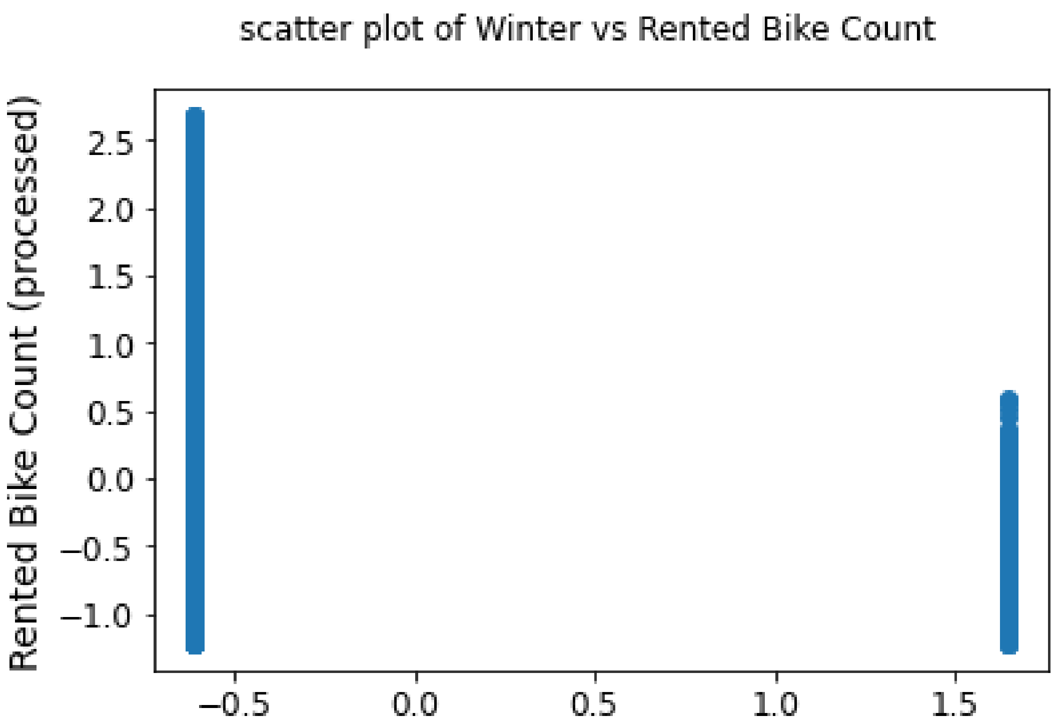

scatter plot of Snowfall (cm) vs Rented Bike Count
Rented Bike Count (processed)
2.5
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
0
2
4
6
8


scatter plot of Holiday vs Rented Bike Count
Rented Bike Count (processed)
2.5
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
0
1
2
3
4


scatter plot of Spring vs Rented Bike Count
Rented Bike Count (processed)
2.5
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−0.5
0.0
0.5
1.0
1.5


scatter plot of Summer vs Rented Bike Count
Rented Bike Count (processed)
2.5
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−0.5
0.0
0.5
1.0
1.5


scatter plot of Autumn vs Rented Bike Count
Rented Bike Count (processed)
2.5
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−0.5
0.0
0.5
1.0
1.5



[91]:
```
#calculate the correlation
for column in X.columns:
    corr = rentals_processed[column].corr(rentals_processed['Rented Bike␣
↪Count'])
    print(f'the correlation between {column} and Rented Bike Count is : {corr}')
```

```
the correlation between Date and Rented Bike Count is : 0.026094169876357046
the correlation between Hour and Rented Bike Count is : 0.3918439597288353
the correlation between Temperature (C) and Rented Bike Count is :
0.548353810704363
the correlation between Humidity (%) and Rented Bike Count is :
-0.20700345922807184
the correlation between Wind speed (m/s) and Rented Bike Count is :
0.14331299349764404
the correlation between Visibility (10m) and Rented Bike Count is :
0.15471296988224228
the correlation between Dew point temperature (C) and Rented Bike Count is :
0.3927453380351034
the correlation between Solar Radiation (MJ/m2) and Rented Bike Count is :
0.21377080574275645
the correlation between Rainfall(mm) and Rented Bike Count is :
-0.16555439654681536
the correlation between Snowfall (cm) and Rented Bike Count is :
```

```
-0.13368540714102628
the correlation between Holiday and Rented Bike Count is : -0.06449272002285067
the correlation between Spring and Rented Bike Count is : 0.04038090223453426
the correlation between Summer and Rented Bike Count is : 0.2504166980175639
the correlation between Autumn and Rented Bike Count is : 0.21081917163239267
the correlation between Winter and Rented Bike Count is : -0.481268452193194
```

[92]:
```
### Your written answers here
'''
from the correlations above, we see the Temperature,Winter, Hour are the most␣
↪correlated with bike rentals
because they have the correlation value the closest to 1(or -1)
'''
```

[92]:
```
'\nfrom the correlations above, we see the Temperature,Winter, Hour are the most
correlated with bike rentals\nbecause they have the correlation value the
closest to 1(or -1)\n'
```

1.5 Step 3: Predicting bike rentals (25%)

A regression approach will be used for this problem: that is, "bike rentals" will be treated as a real number whose value will be predicted. If necessary, it could be rounded to the nearest integer afterwards, but this will not be necessary here. The root mean squared error (rmse) metric will be used to quantify performance.

Split the data appropriately so that 20% of it will be kept as a hold-out test set. **Build a pipeline** starting with the one specified in section 2.2 above, and now include a ***linear regression*** **model**. After you've done this, **fit** this to your training data for a quick test. To get an idea of how successful this model is, **calculate the rmse of the fit to the training data**. To act as a simple baseline for comparison, **also calculate the rmse** that you would get if all the predictions were equal to the **mean of the training targets** (i.e. bike rentals).

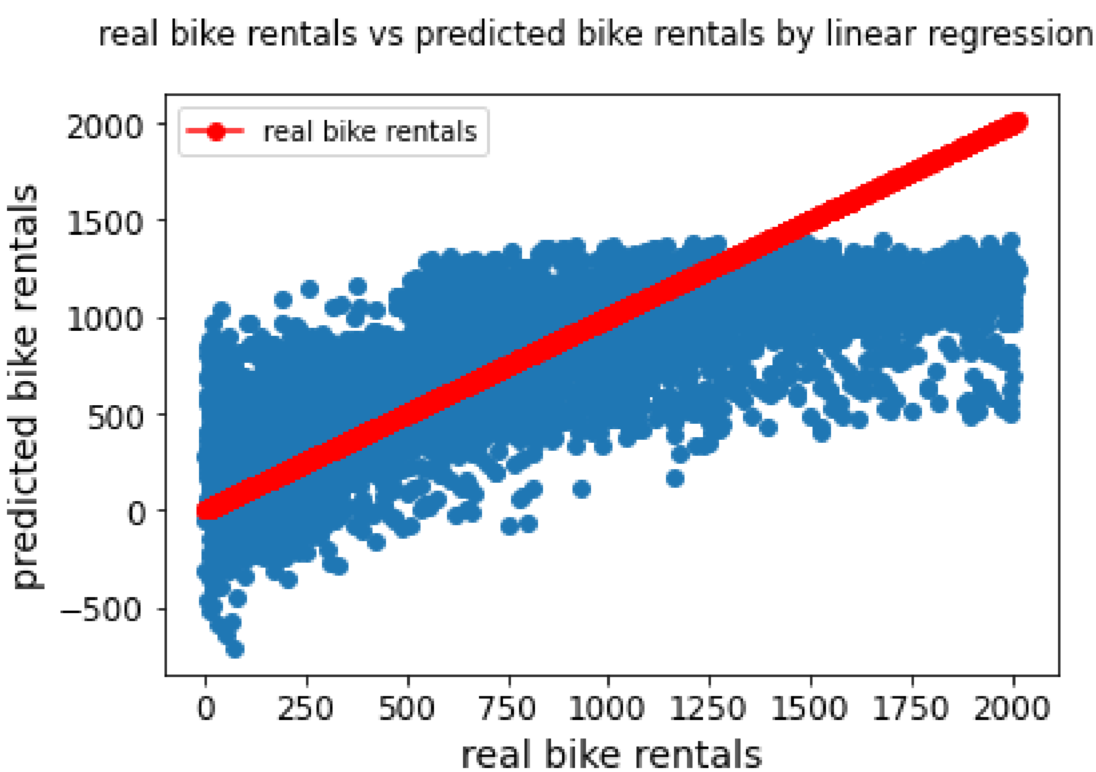
[93]:
```
### Your code here
#Split the data appropriately so that 20% of it will be kept as a hold-out test␣
↪set.
from sklearn.model_selection import train_test_split
train_set, test_set = train_test_split(rentals, test_size=0.2, random_state=42)
#split the X and y for train and test data
X_train = train_set.drop(["Rented Bike Count"],axis =1)
y_train = train_set["Rented Bike Count"].copy()
X_test  = test_set.drop(["Rented Bike Count"],axis=1)
y_test  = test_set["Rented Bike Count"].copy()
```

[94]:
```
#Build a pipeline starting with the one specified in section 2.2 above, and now␣
↪include a linear regression model.
#we use a simpleImputer to set nan to median and use StandardScaler to scale␣
↪the data
from sklearn.linear_model import LinearRegression
```

```
lr_pl = Pipeline( [('imputer',SimpleImputer(strategy = "median")),
                   ('std_scaler',StandardScaler() ),('lr',LinearRegression())])
```

Show an appropriate **visualisation of the fit for your linear regression.**

[95]:
```
### Your code here
#fit the linear regression
lr_pl.fit(X_train,y_train)
y_pred = lr_pl.predict(X_train)

#visualisation
plt.suptitle("real bike rentals vs predicted bike rentals by linear regression")
plt.xlabel("real bike rentals")
plt.ylabel("predicted bike rentals")
plt.scatter(y_train,y_pred)
plt.plot(y_train,y_train,'r-o', label = "real bike rentals")
plt.legend()
plt.show()
```



[96]:
```
#calculate the rmse of the fit to the training data.
from sklearn.metrics import mean_squared_error
```

```
rmse_lr = np.sqrt(mean_squared_error(y_pred,y_train))
print(rmse_lr) #437.2957308414973
#calculate the rmse that you would get if all the predictions were equal to the␣
 ↪mean of the training targets (i.e. bike rentals).
mock_y_pred = pd.Series([y_train.mean()]*len(y_train.values))
rmse_mock_lr = np.sqrt(mean_squared_error(mock_y_pred,y_train))
print(rmse_mock_lr) #646.170499547999

'''fitting with leanear regression model is better than just using mean value␣
 ↪because it has smaller rmse'''
```

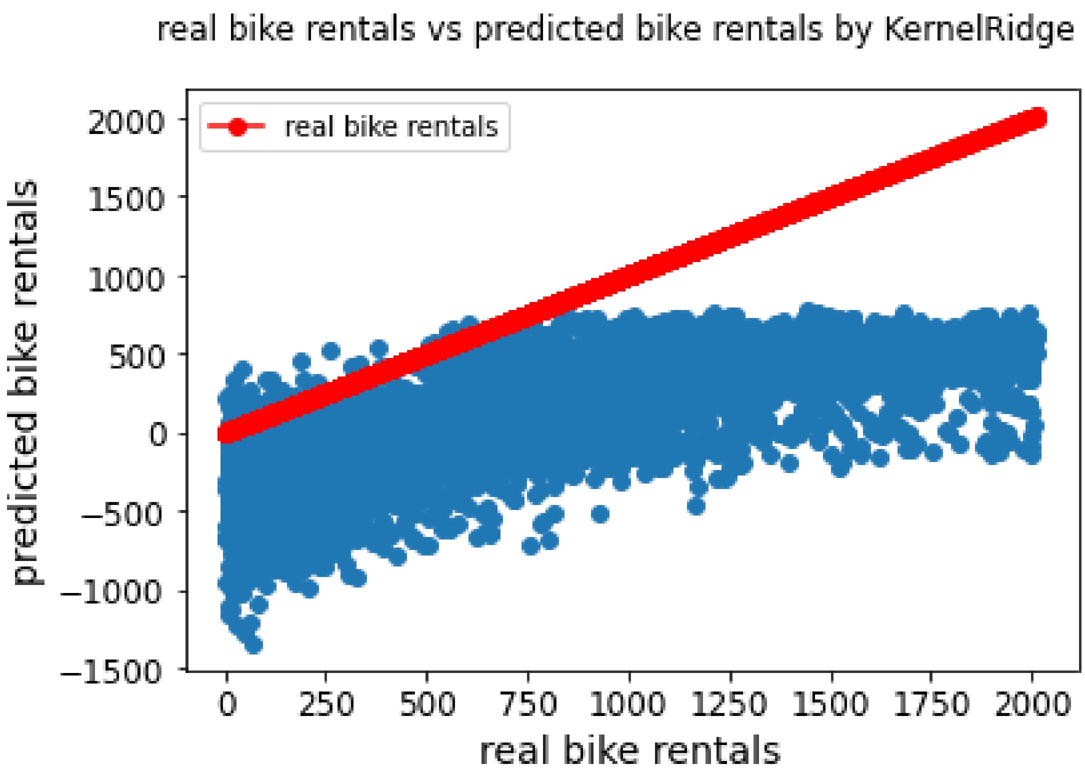
```
342.26190415270077
515.6916120115031
```

[96]:
```
'fitting with leanear regression model is better than just using mean value
because it has smaller rmse'
```

Now two other, different regression models (that you probably won't be familiar with) will be fit and later these will be compared to find the best one.

The second model to fit is ***Kernel Ridge*** regression (`from sklearn.kernel_ridge import KernelRidge`). **Build a pipeline using this and fit it to your training data**, using the default settings. Again, **plot the fit and display the rmse for the training dataset.**

[97]:
```
### Your code here
from sklearn.kernel_ridge import KernelRidge
#make the pipeline
krr_pl = Pipeline( [('imputer',SimpleImputer(strategy = "median")),
                    ('std_scaler',StandardScaler() ),('krr',KernelRidge())])

#fit the krr
krr_pl.fit(X_train,y_train)
y_pred_krr = krr_pl.predict(X_train)

#visualisation
plt.suptitle("real bike rentals vs predicted bike rentals by KernelRidge")
plt.xlabel("real bike rentals")
plt.ylabel("predicted bike rentals")
plt.scatter(y_train,y_pred_krr)
plt.plot(y_train,y_train,'r-o', label = "real bike rentals")
plt.legend()
plt.show()

#calculate the rmse of the fit to the training data.

rmse_krr = np.sqrt(mean_squared_error(y_pred_krr,y_train))
print(rmse_krr)
```

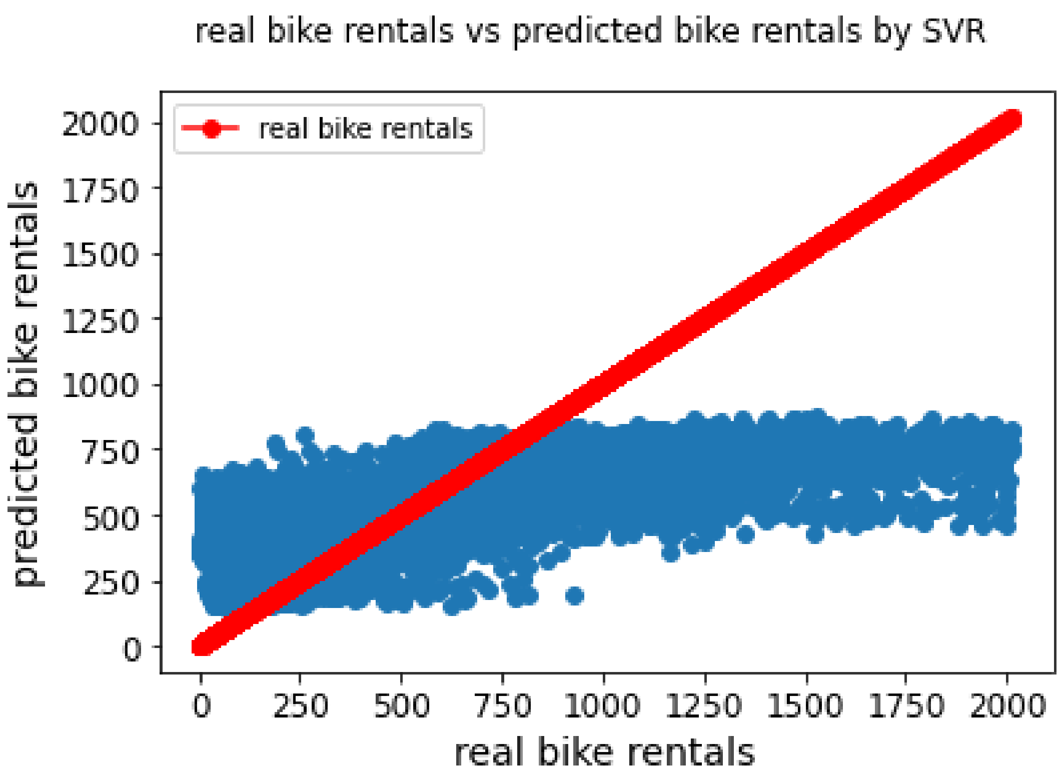



```
719.1748913398326
```

The third, and most powerful model, is ***Support Vector Regression*** (`from sklearn.svm import SVR`). **Build a pipeline using this and fit it to your training data**, using the default settings. Again, **plot the fit and display the rmse for the training dataset.**

[98]:
```
### Your code here
from sklearn.svm import SVR
#make the pipeline
svr_pl = Pipeline( [('imputer',SimpleImputer(strategy = "median")),
                    ('std_scaler',StandardScaler() ),('svr',SVR())])

#fit the krr
svr_pl.fit(X_train,y_train)
y_pred_svr = svr_pl.predict(X_train)

#visualisation
plt.suptitle("real bike rentals vs predicted bike rentals by SVR")
plt.xlabel("real bike rentals")
plt.ylabel("predicted bike rentals")
plt.scatter(y_train,y_pred_svr)
plt.plot(y_train,y_train,'r-o', label = "real bike rentals")
plt.legend()
```

```
plt.show()

#calculate the rmse of the fit to the training data.

rmse_svr = np.sqrt(mean_squared_error(y_pred_svr,y_train))
print(rmse_svr)
```




```
418.52717323978175
```

1.6 Step 4: Cross validation (20%)

Perform a *10 fold cross validation* for each model. This splits the training set (that we've used above) into 10 equal size subsets, and uses each in turn as the validation set while training a model with the other 9. You should therefore have 10 rmse values for each cross validation run.

Display the mean and standard deviation of the rmse values obtained for each model **for the validation splits** using the same settings/parameters for the models as used above. **Also display the mean and standard deviation** of the rmse values obtained for the **training data splits**.

[99]:
```
### Your code here

#10-fold cross vaidation for linear regression
```

```
from sklearn.model_selection import KFold
val_rmses=[];
train_rmses=[];
kf = KFold(n_splits =10)
kf.get_n_splits(X_train)
for train_index, val_index in kf.split(X_train):
    X_trainr,X_val =X_train.iloc[train_index],X_train.iloc[val_index]
    y_trainr,y_val =y_train.iloc[train_index],y_train.iloc[val_index]

    #pridict validate set
    lr_pl.fit(X_trainr,y_trainr)
    y_pred = lr_pl.predict(X_val)
    rmse_lr = np.sqrt(mean_squared_error(y_pred,y_val))
    val_rmses.append(rmse_lr)

    #predict train set
    y_pred = lr_pl.predict(X_trainr)
    rmse_lr = np.sqrt(mean_squared_error(y_pred,y_trainr))
    train_rmses.append(rmse_lr)

print("For the linear regression model:")
print(f"The mean of the validation split's rmse is : {np.mean(val_rmses)},␣
 ↪\nthe standard deviation  of the validation split's rmse is : {np.
 ↪std(val_rmses)} ")
print(f"The mean of the training split's rmse is : {np.mean(train_rmses)},␣
 ↪\nthe standard deviation  of the training split's rmse is : {np.
 ↪std(train_rmses)} ")
```

```
For the linear regression model:
The mean of the validation split's rmse is : 342.8231555376105,
the standard deviation  of the validation split's rmse is : 12.333853464329598
The mean of the training split's rmse is : 342.21624670175385,
the standard deviation  of the training split's rmse is : 1.377432505411076
```

[100]:

```
### Your code here

#10-fold cross vaidation for krr(KernelRidge)
val_rmses=[];
train_rmse=[];
kf = KFold(n_splits =10)
kf.get_n_splits(X_train)
for train_index, val_index in kf.split(X_train):
    X_trainr,X_val =X_train.iloc[train_index],X_train.iloc[val_index]
    y_trainr,y_val =y_train.iloc[train_index],y_train.iloc[val_index]

    #pridict validate set
    krr_pl.fit(X_trainr,y_trainr)
```

```
    y_pred = krr_pl.predict(X_val)
    rmse_krr = np.sqrt(mean_squared_error(y_pred,y_val))
    val_rmses.append(rmse_krr)

    #predict train set
    y_pred = krr_pl.predict(X_trainr)
    rmse_krr = np.sqrt(mean_squared_error(y_pred,y_trainr))
    train_rmses.append(rmse_krr)

print("For the KernelRidge model:")
print(f"The mean of the validation split's rmse is : {np.mean(val_rmses)},␣
↪\nthe standard deviation  of the validation split's rmse is : {np.
↪std(val_rmses)} ")
print(f"The mean of the training split's rmse is : {np.mean(train_rmses)},␣
↪\nthe standard deviation  of the training split's rmse is : {np.
↪std(train_rmses)} ")
```

```
For the KernelRidge model:
The mean of the validation split's rmse is : 719.3456977314243,
the standard deviation  of the validation split's rmse is : 15.933169864530207
The mean of the training split's rmse is : 530.6852494319978,
the standard deviation  of the training split's rmse is : 188.48151645327528
```

[101]:
```
### Your code here

#10-fold cross vaidation for svr (Support Vector Regression )
from sklearn.model_selection import KFold
val_rmses=[];
train_rmse=[];
kf = KFold(n_splits =10)
kf.get_n_splits(X_train)
for train_index, val_index in kf.split(X_train):
    X_trainr,X_val =X_train.iloc[train_index],X_train.iloc[val_index]
    y_trainr,y_val =y_train.iloc[train_index],y_train.iloc[val_index]

    #pridict validate set
    svr_pl.fit(X_trainr,y_trainr)
    y_pred = svr_pl.predict(X_val)
    rmse_svr = np.sqrt(mean_squared_error(y_pred,y_val))
    val_rmses.append(rmse_svr)

    #predict train set
    y_pred = svr_pl.predict(X_trainr)
    rmse_svr = np.sqrt(mean_squared_error(y_pred,y_trainr))
    train_rmses.append(rmse_svr)

print("For the Support Vector Regression  model:")
```

```
print(f"The mean of the validation split's rmse is : {np.mean(val_rmses)},␣
↪\nthe standard deviation  of the validation split's rmse is : {np.
↪std(val_rmses)} ")
print(f"The mean of the training split's rmse is : {np.mean(train_rmses)},␣
↪\nthe standard deviation  of the training split's rmse is : {np.
↪std(train_rmses)} ")
```

```
For the Support Vector Regression  model:
The mean of the validation split's rmse is : 425.60322133110776,
the standard deviation  of the validation split's rmse is : 14.371415744058412
The mean of the training split's rmse is : 495.53991441909443,
the standard deviation  of the training split's rmse is : 161.7237442321717
```

On the basis of the results you found above, would you say that any of the models were **under-fitting or over-fitting**?

Which method do you think is the best out of these three?

[102]:
```
### Your answer here
'''They are all underfitting because the training set and validation set errors␣
↪are all high and the fit of rental count prediction and real rental count␣
↪are obviously bad.'''
'''The linear regression model is the best out of these three beause it has the␣
↪lowest training error and validation error'''
```

[102]:
```
'The linear regression model is the best out of these three beause it has the
lowest training error and validation error'
```

1.7 Step 5: Grid parameter search (15%)

Both the Kernel Ridge Regression and Support Vector Regression have hyperparameters that can be adjusted to suit the problem. **Choose either the KernelRidge or SVR** (your choice entirely), and use grid search to systematically compare the generalisation performance (rmse) obtained with different hyperparameter settings (still with 10-fold CV). Use the sklearn function `GridSearchCV` to do this.

For KernelRidge, vary the hyperparameter `alpha`.

For SVR, vary the hyperparameter `C`.

Print out the hyperparameter setting for the best (i.e. chosen) method.

Finally, **train and apply your chosen method**, with appropriate hyperparameter settings, to the ***test set*** **and report the performance**.

[103]:
```
### Your code here

from sklearn.model_selection import GridSearchCV
parameters = {'svr__C': [1, 10, 100, 1000,10000]}
```

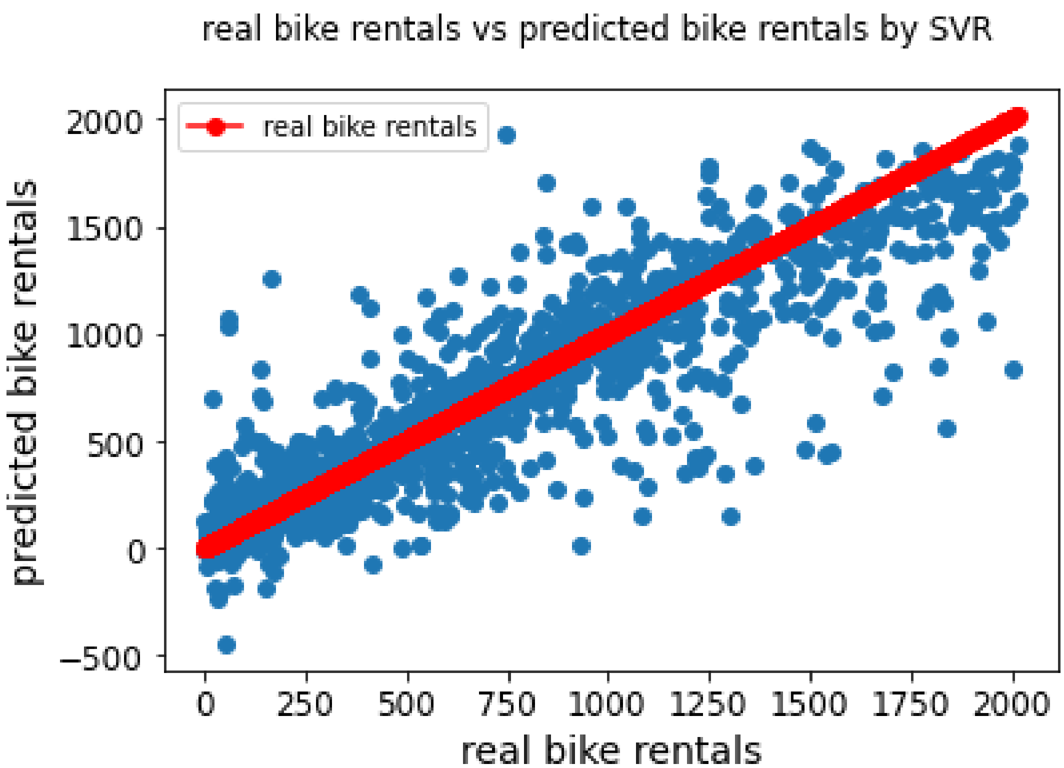
```
gridcv = GridSearchCV(svr_pl, parameters, cv=10,␣
↪scoring='neg_root_mean_squared_error')
gridcv.fit(X_train, y_train)
print(f'Best score is {-gridcv.best_score_} for best params of {gridcv.
↪best_params_}')
```

```
Best score is 223.3096230025789 for best params of {'svr__C': 10000}
```

How different was the test set performance to the validation performance, and is this suggestive of over-fitting, under-fitting or neither?

[104]:
```
### Your answers here
svr_pl2 = Pipeline( [('imputer',SimpleImputer(strategy = "median")),
                  ('std_scaler',StandardScaler() ),('svr',SVR(C = 10000))])
```

[105]:
```
#testing
#fit the svr2
svr_pl2.fit(X_train,y_train)
y_pred_svr2 = svr_pl2.predict(X_test)

#visualisation
plt.suptitle("real bike rentals vs predicted bike rentals by SVR")
plt.xlabel("real bike rentals")
plt.ylabel("predicted bike rentals")
plt.scatter(y_test,y_pred_svr2)
plt.plot(y_train,y_train,'r-o', label = "real bike rentals")
plt.legend()
plt.show()

#calculate the rmse of the fit to the training data.

rmse_svr = np.sqrt(mean_squared_error(y_pred_svr2,y_test))
print(rmse_svr)
```




```
224.26077347090134
```

[106]:
```
'''
The accurency of predicted result improves much from the previous SVR model␣
↪because it has half smaller rmse and better fitting that can be seen␣
↪obviously from the plot.
However the run time become much longer than before.
The SVR model with C value of 10000 is not under-fitting/over-fitting.
'''
```

[106]:
```
'\nThe accurency of predicted result improves much from the previous SVR model
because it has half smaller rmse and better fitting that can be seen obviously
from the plot.\nHowever the run time become much longer than before.\nThe SVR
model with C value of 10000 is not under-fitting/over-fitting.\n'
```

[ ]: