A1

June 21, 2025

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 scenario for this assignment is that you are a new employee of a company (that rents bikes, alongside other activities) and you have been assigned the task of predicting the bike rentals. Your line manager has given you some instructions (those shown below) but is expecting you to be able to do this task without close supervision and to report back with understandable and concise text, graphics and code (and of course the company wants a copy of all the code required to perform this task). Naturally, you are wanting to show that you are a valuable member of the company and although the company allows the use of ChatGPT, you will want to show that you are making useful contributions and that you bring value to the company beyond just being able to type instructions into ChatGPT, as otherwise the company might replace you with a cheaper data entry employee. Hence, you should use ChatGPT whenever you like (or whenever instructed to - see later) but do highlight how your own knowledge and judgement makes a contribution.

The main aims of this assignment are:

- to practice using tools for loading and viewing data sets;
- to check data for common pitfalls and clean it up;
- 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. Some questions require writing code, some require graphical results, and some require short 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 be able to be 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, including ChatGPT, although do not use someone else's code or answers that directly relate to these questions. If you take a large portion of code or text from the internet or ChatGPT 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.

[1]:
```
# 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 (40%)

Download the data set `SeoulBikeData.csv` **from MyUni** using the link provided on the assignment page.

The data is stored in a CSV (comma separated values) 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 degrees Celsius
- Humidity: %
- Windspeed: m/s
- Visibility: 10m
- Dew point temperature: degrees 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)

1.3.1 1.1 Load and visualise the data

Load the data set from the csv file into a DataFrame, summarise it in text using one pandas function, and then visualise each feature with one type of plot (this can be different for each feature).

[2]:
```
raw_df = pd.read_csv("./SeoulBikeData.csv")
```

[3]:
```
raw_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8760 entries, 0 to 8759
Data columns (total 14 columns):
 #   Column                     Non-Null Count  Dtype
---  ------                     --------------  -----
 0   Date                       8760 non-null   object
 1   Rented Bike Count          8760 non-null   int64
 2   Hour                       8760 non-null   int64
 3   Temperature (C)            8760 non-null   float64
 4   Humidity (%)               8760 non-null   float64
 5   Wind speed (m/s)           8760 non-null   float64
 6   Visibility (10m)           8760 non-null   int64
 7   Dew point temperature (C)  8760 non-null   float64
 8   Solar Radiation (MJ/m2)    8760 non-null   object
 9   Rainfall(mm)               8760 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(3), object(7)
memory usage: 958.3+ KB
```

As shown above, it is clear that the solar radiation, rainfall and snowfall columns all will need to be converted to numerical data in order to be used to predict the bike rentals. This will be done in the data cleaning section. There are no missing values, as all columns have the same number of entries.

[4]:
```
raw_df.describe()
```

[4]:

	Rented Bike Count	Hour	Temperature (C)	Humidity (%)	\
count	8760.000000	8760.000000	8760.000000	8760.000000	
mean	714.876027	11.500000	12.945765	58.268014	
std	1160.468927	6.922582	12.376168	20.807845	
min	0.000000	0.000000	-17.800000	-2.200000	
25%	191.000000	5.750000	3.500000	42.000000	
50%	504.500000	11.500000	13.700000	57.000000	
75%	1066.000000	17.250000	22.500000	74.000000	
max	90997.000000	23.000000	195.000000	455.000000	

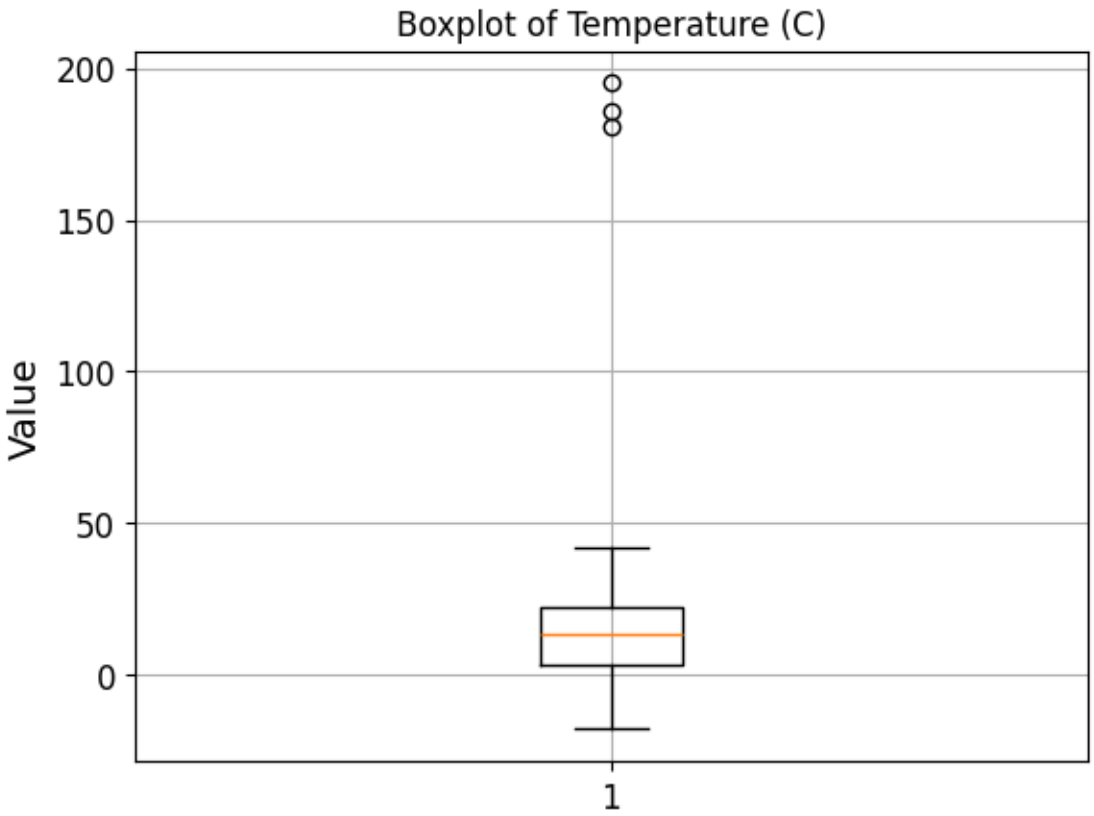
	Wind speed (m/s)	Visibility (10m)	Dew point temperature (C)
count	8760.000000	8760.000000	8760.000000
mean	1.848950	1436.825799	4.073813
std	10.665215	608.298712	13.060369
min	-0.700000	27.000000	-30.600000
25%	0.900000	940.000000	-4.700000
50%	1.500000	1698.000000	5.100000
75%	2.300000	2000.000000	14.800000
max	991.100000	2000.000000	27.200000

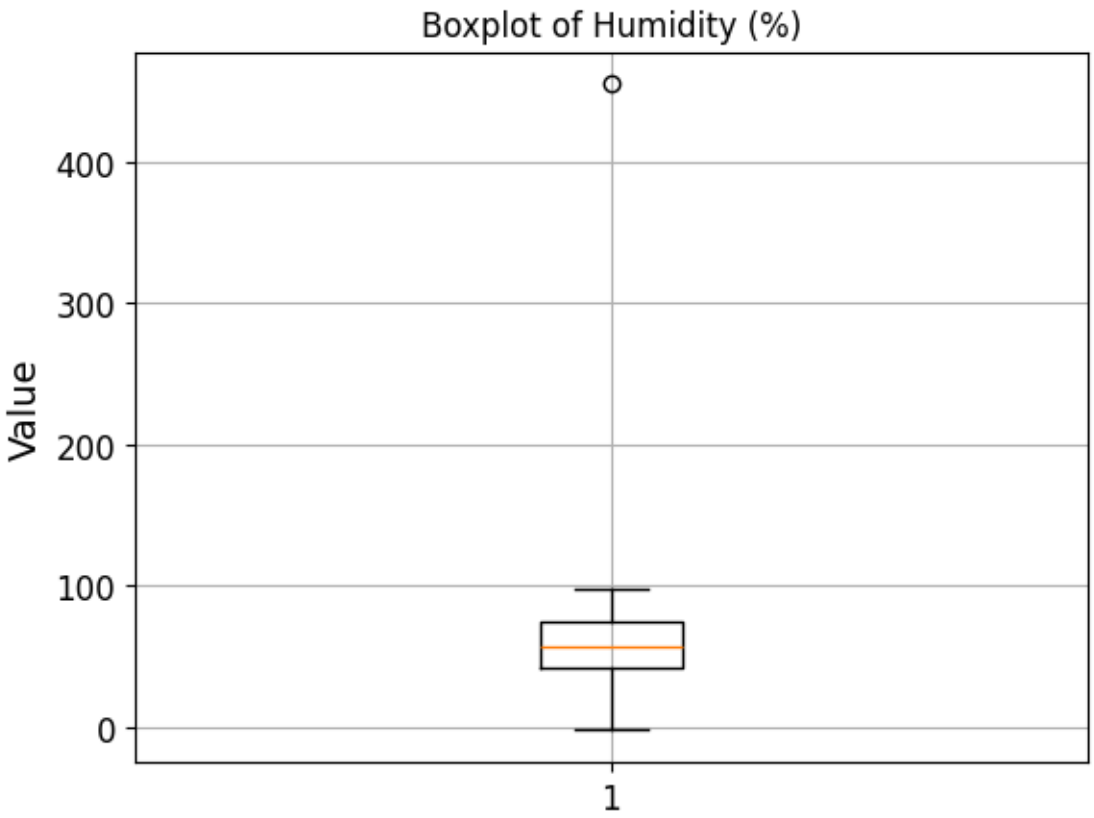
[5]:
```
for column in raw_df.columns:
    print(column)
```

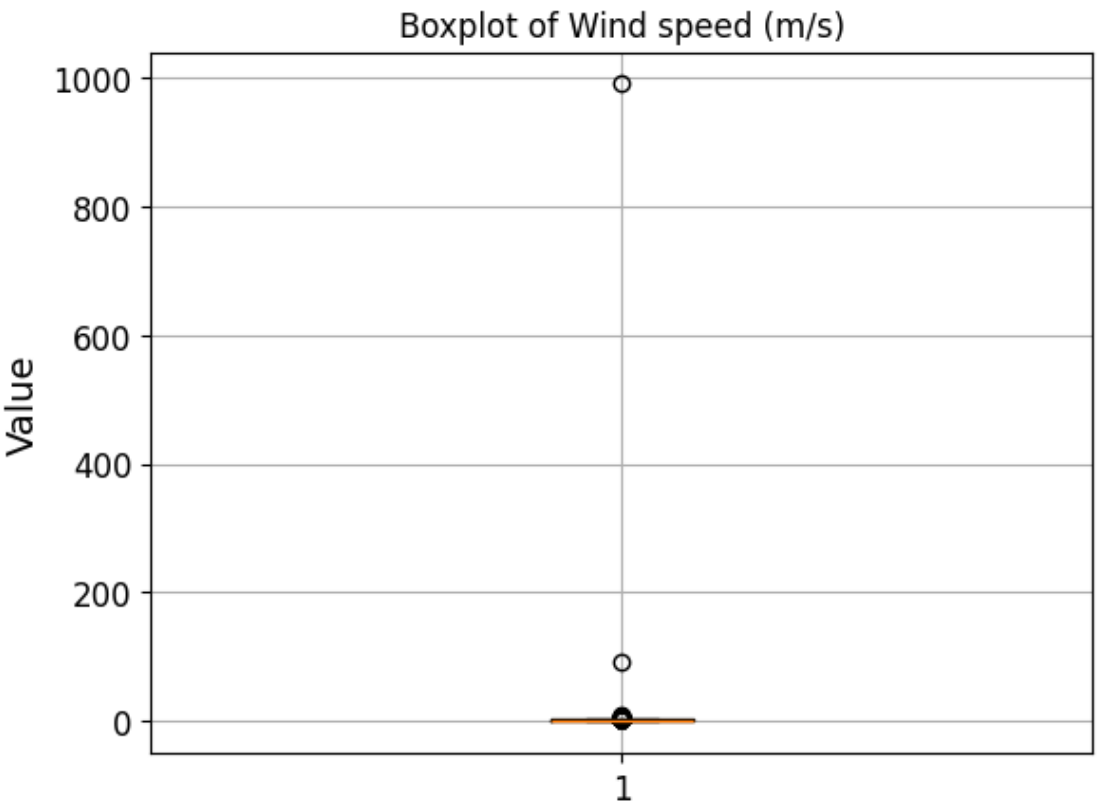
```
Date
Rented Bike Count
Hour
Temperature (C)
Humidity (%)
Wind speed (m/s)
Visibility (10m)
Dew point temperature (C)
Solar Radiation (MJ/m2)
Rainfall(mm)
Snowfall (cm)
Seasons
Holiday
Functioning Day
```

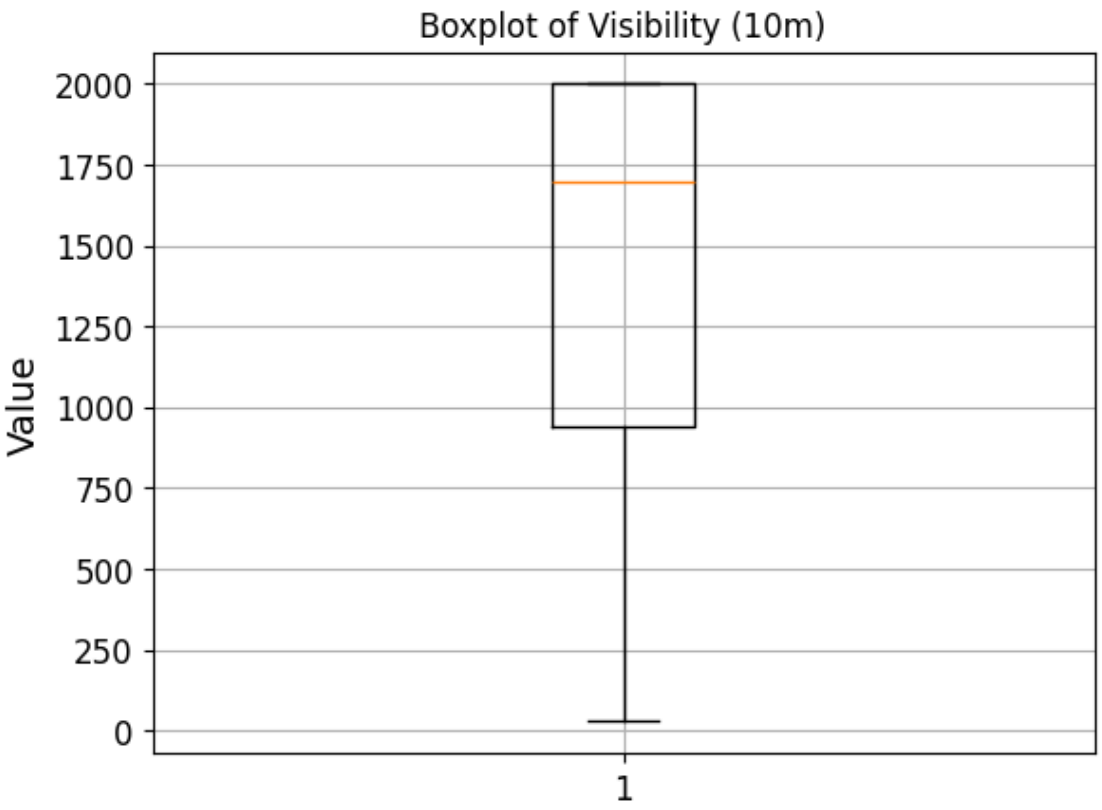
[6]:
```
# Draw box plots to display information about the columns
box_plot_columns = [
    'Temperature (C)',
    'Humidity (%)',
    'Wind speed (m/s)',
    'Visibility (10m)',
    'Dew point temperature (C)',
    'Solar Radiation (MJ/m2)',
```

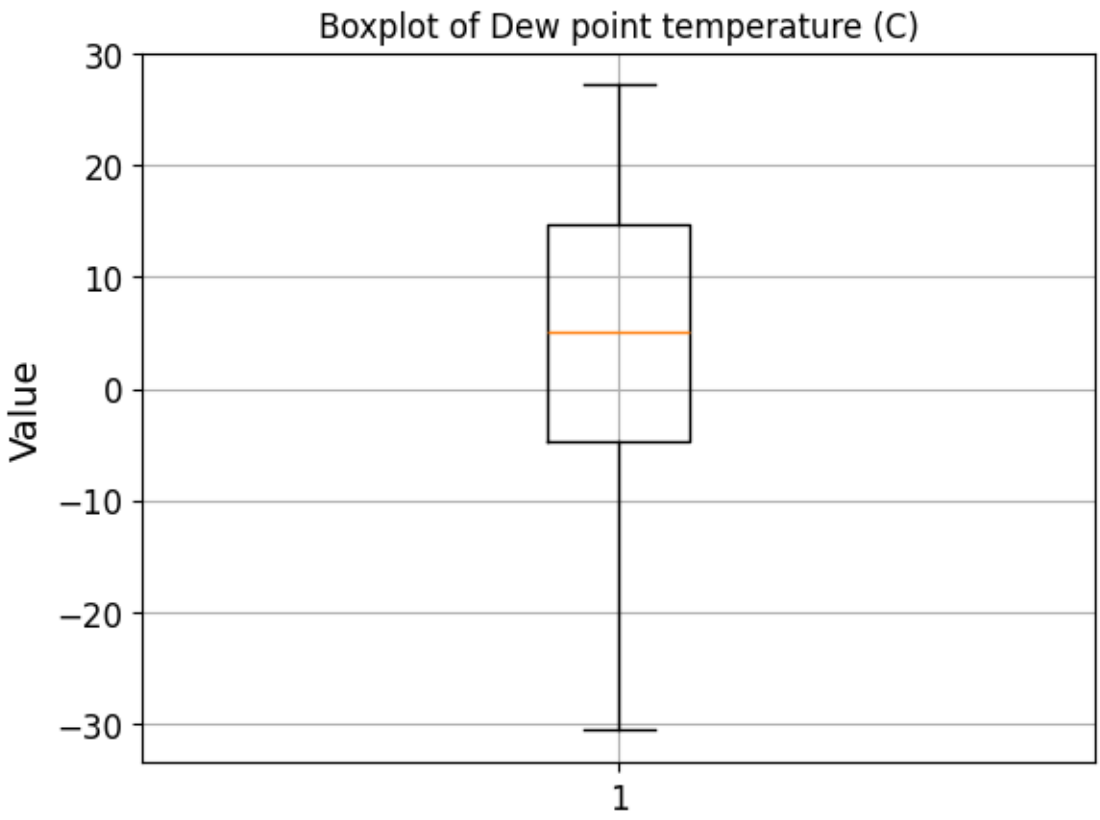
```
    'Rainfall(mm)',
    'Snowfall (cm)',
]

numerical_columns = raw_df.select_dtypes(include=np.number)

for column in box_plot_columns:
    if column not in numerical_columns:
        print(f"Column: {column} was not able to be displayed as a box plot as␣
 ↪it contains errors or non-numerical values")
        continue

    plt.figure()
    plt.boxplot(raw_df[column])
    plt.title(f'Boxplot of {column}')
    plt.ylabel('Value')
    plt.grid(True)
    plt.show()
```




Boxplot of Humidity (%)
Value
400
300
200
100
0
1


Boxplot of Wind speed (m/s)
Value
1000
800
600
400
200
0
1


Boxplot of Visibility (10m)
Value
2000
1750
1500
1250
1000
750
500
250
0
1




```
Column: Solar Radiation (MJ/m2) was not able to be displayed as a box plot as it
contains errors or non-numerical values
Column: Rainfall(mm) was not able to be displayed as a box plot as it contains
errors or non-numerical values
Column: Snowfall (cm) was not able to be displayed as a box plot as it contains
errors or non-numerical values
```

As shown above, it is clear that the solar radiation, rainfall and snowfall columns all include errors or invalid values that are preventing them from being displayed in box plots. This needs to be considered when the data is cleaned.

1.3.2 1.2 Cleaning the data

Do the following to the data: - Using the "Functioning day" feature, **remove rows from the DataFrame** where the business is closed and then **delete the Functioning Day feature from the DataFrame**. - **Convert seasons to a one hot encoded format** (1 binary feature for each of the 4 seasons). - Replace the **Date** feature with a binary **Weekday** feature (1 for a weekday and 0 for weekend) using the code sample below or your own code. - **Convert remaining non-numerical features to a numerical format** or replace with NaN (i.e. `np.nan`) where not possible. - **Identify and fix any outliers and errors in the data**.

Save the result as a new csv file called `CleanedSeoulBikeData.csv` and **upload this** to MyUni

along with this notebook when you submit your assignment.

[7]:
```
## Example code for weekday feature mapping ##

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)
```

[8]:
```
# Only include columns where the Functioning Day feature is set to Yes, and␣
 ↪don't include the Functioning Day feature
df = raw_df[raw_df['Functioning Day'] == 'Yes'].drop('Functioning Day', axis=1)

# Convert the Seasons feature to one-hot encoded
seasons_encoded = pd.get_dummies(df['Seasons'], dtype=int, prefix='Season_')
df = pd.concat([df, seasons_encoded], axis=1).drop('Seasons', axis=1)

# Convert the Date feature to a Weekday boolean feature
df['Weekday'] = df['Date'].apply(date_is_weekday)
df = df.drop('Date', axis=1)

# Convert the Holiday feature to a boolean feature
df['Holiday'] = df['Holiday'].apply(lambda holiday: 0 if holiday == 'No␣
 ↪Holiday' else 1)
```

[9]:
```
df.info()
```

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

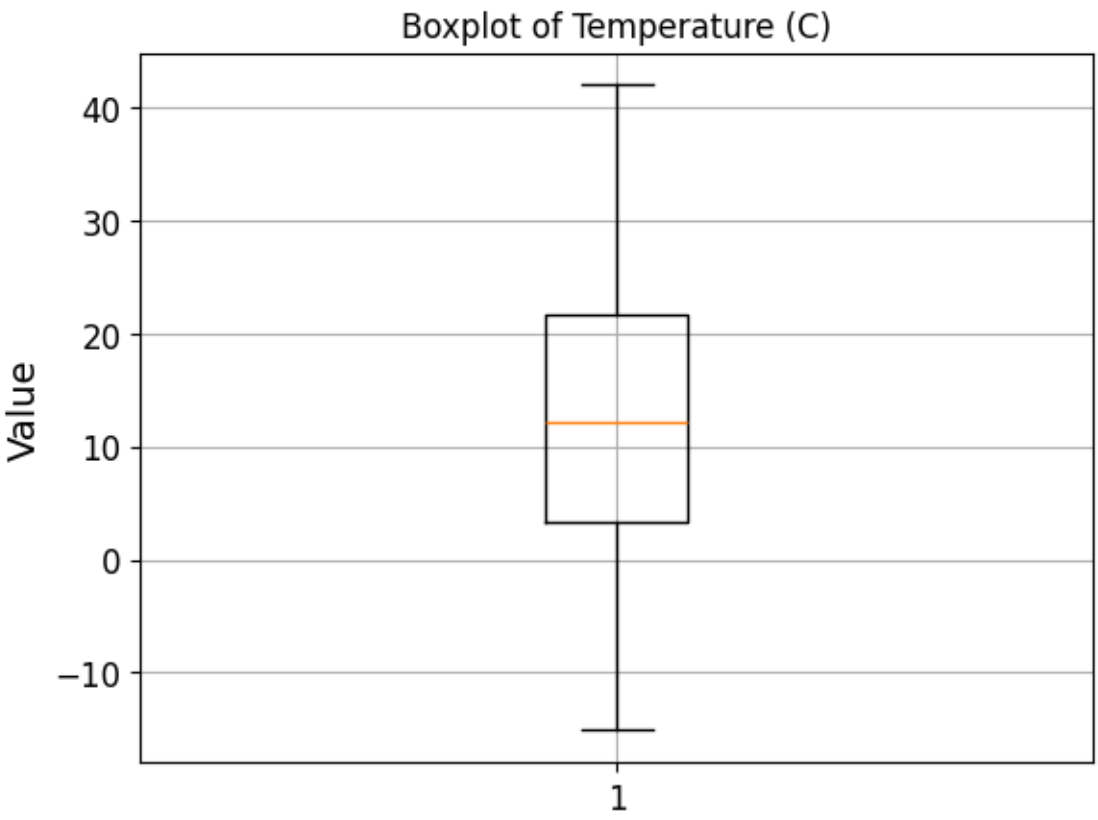
```
 13  Season__Summer              8465 non-null   int64
 14  Season__Winter              8465 non-null   int64
 15  Weekday                     8465 non-null   int64
dtypes: float64(4), int64(9), object(3)
memory usage: 1.1+ MB
```

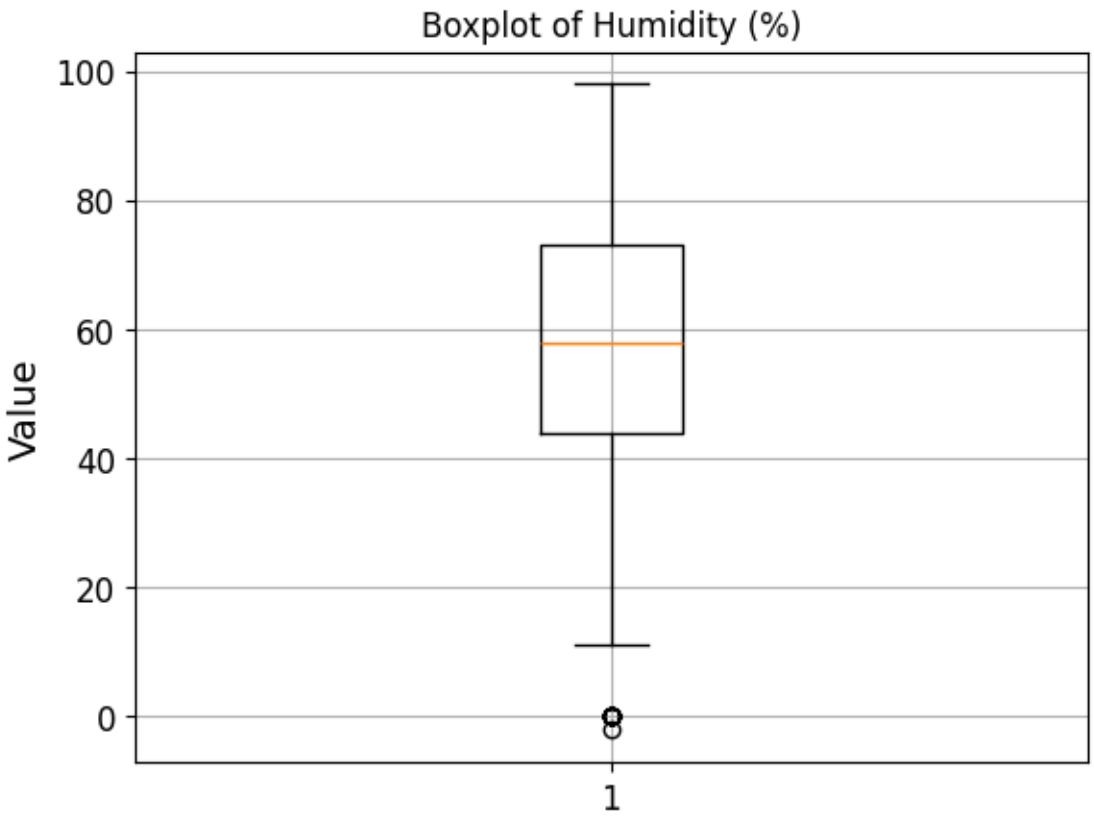
[10]:
```
# Use coerce to convert non-numeric values to NaN
df['Solar Radiation (MJ/m2)'] = pd.to_numeric(df['Solar Radiation (MJ/m2)'],␣
 ↪errors='coerce')
df['Rainfall(mm)'] = pd.to_numeric(df['Rainfall(mm)'], errors='coerce')
df['Snowfall (cm)'] = pd.to_numeric(df['Snowfall (cm)'], errors='coerce')
```

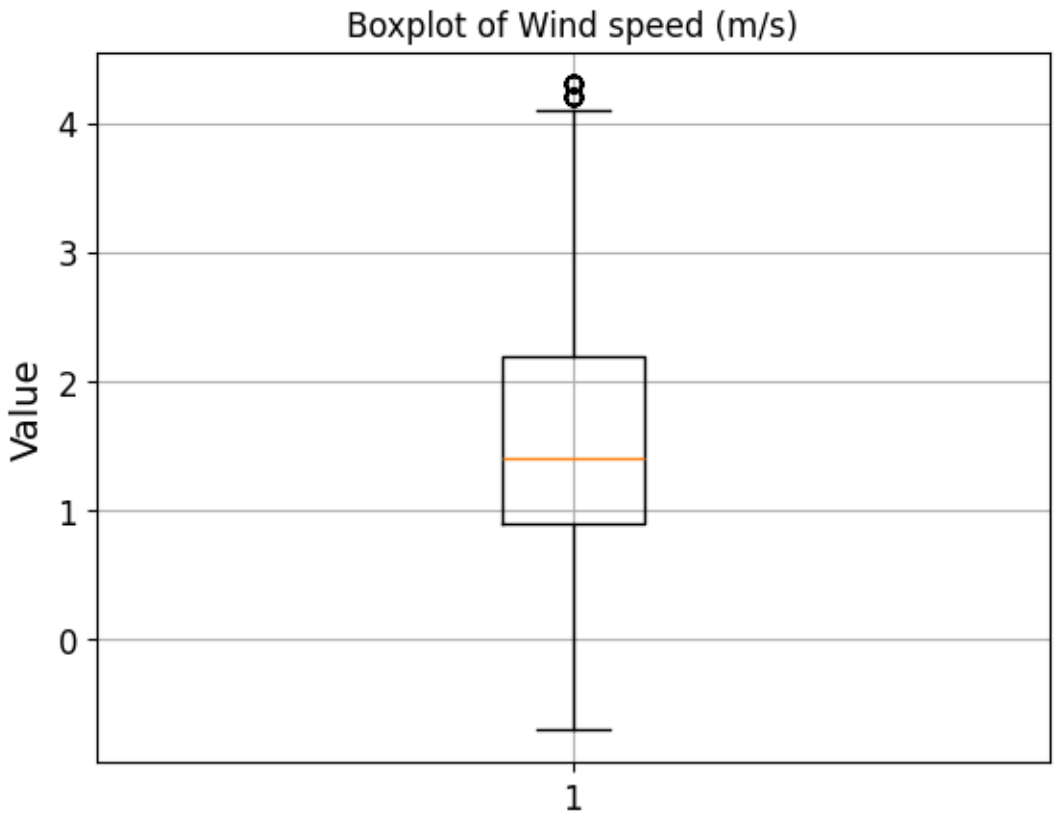
[11]:
```
numerical_columns = df.select_dtypes(include=np.number)

for column in numerical_columns:
    if column == 'Holiday':
        continue
    if str(column).startswith('Season_'):
        continue

    # Remove outliers using the interquartile range
    Q1 = df[column].quantile(0.25)
    Q3 = df[column].quantile(0.75)
    IQR = Q3 - Q1

    lower_bound = Q1 - (1.5 * IQR)
    upper_bound = Q3 + (1.5 * IQR)

    df = df[(df[column] >= lower_bound) & (df[column] <= upper_bound)]
```

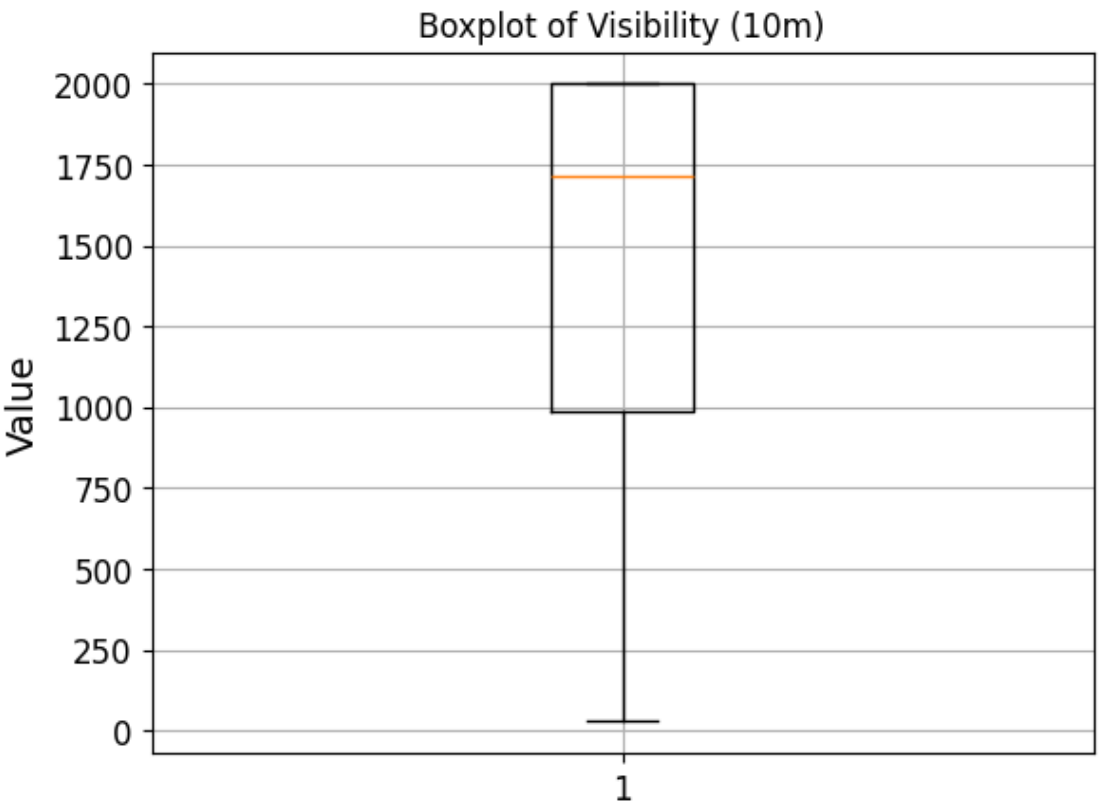
It was decided that the features should be visualised again in box plots, now that the data has been cleaned and outliers or errors have been removed.

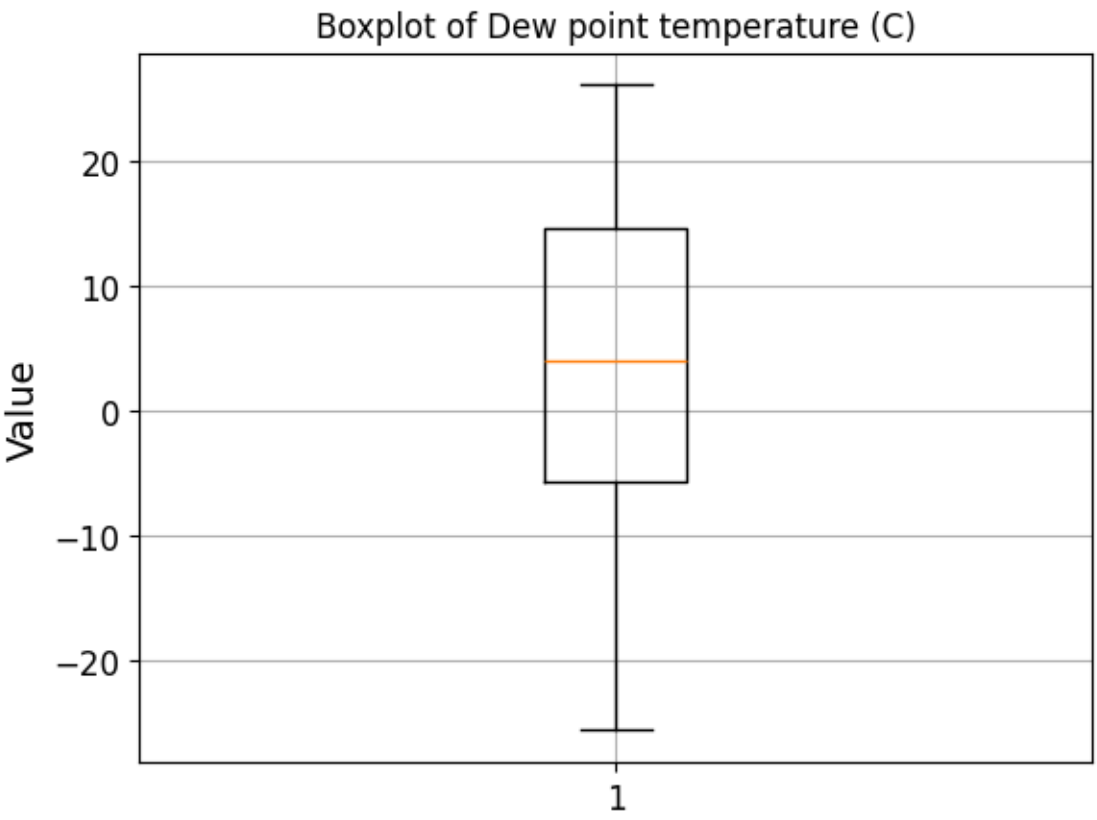
[12]:
```
for column in box_plot_columns:
    plt.figure()
    plt.boxplot(df[column])
    plt.title(f'Boxplot of {column}')
    plt.ylabel('Value')
    plt.grid(True)
    plt.show()
```

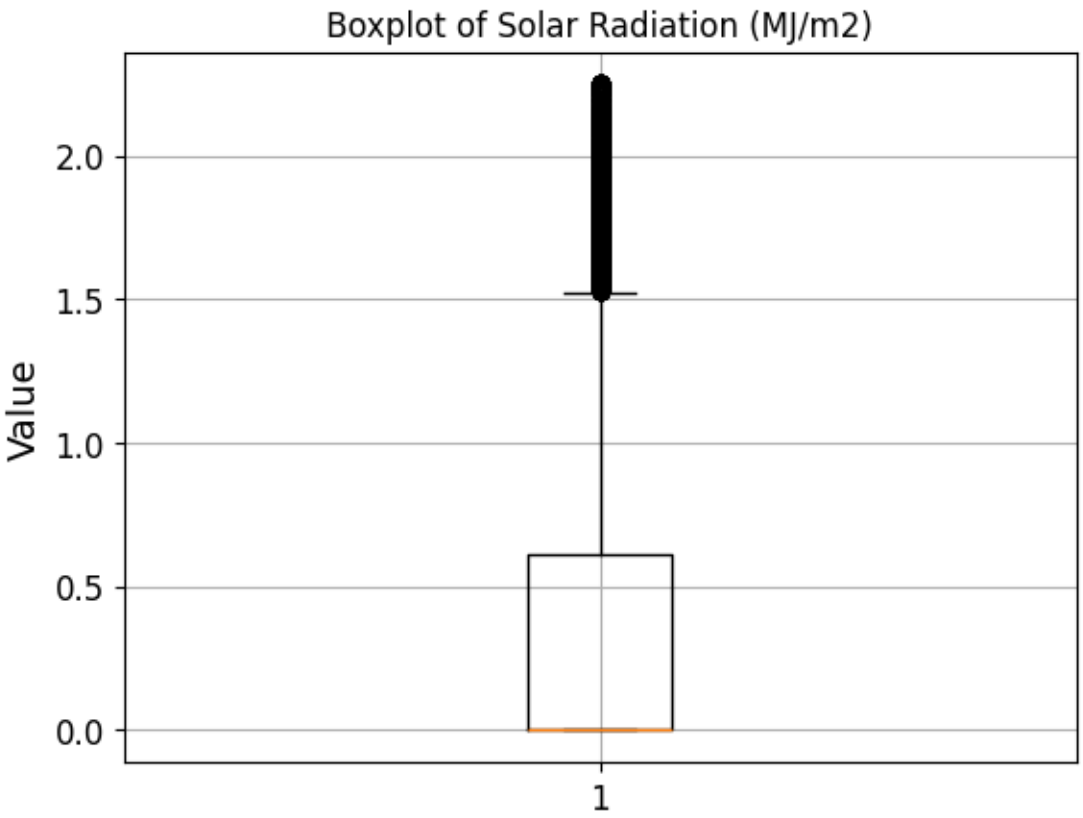

Boxplot of Temperature (C)
Value
40
30
20
10
0
−10
1

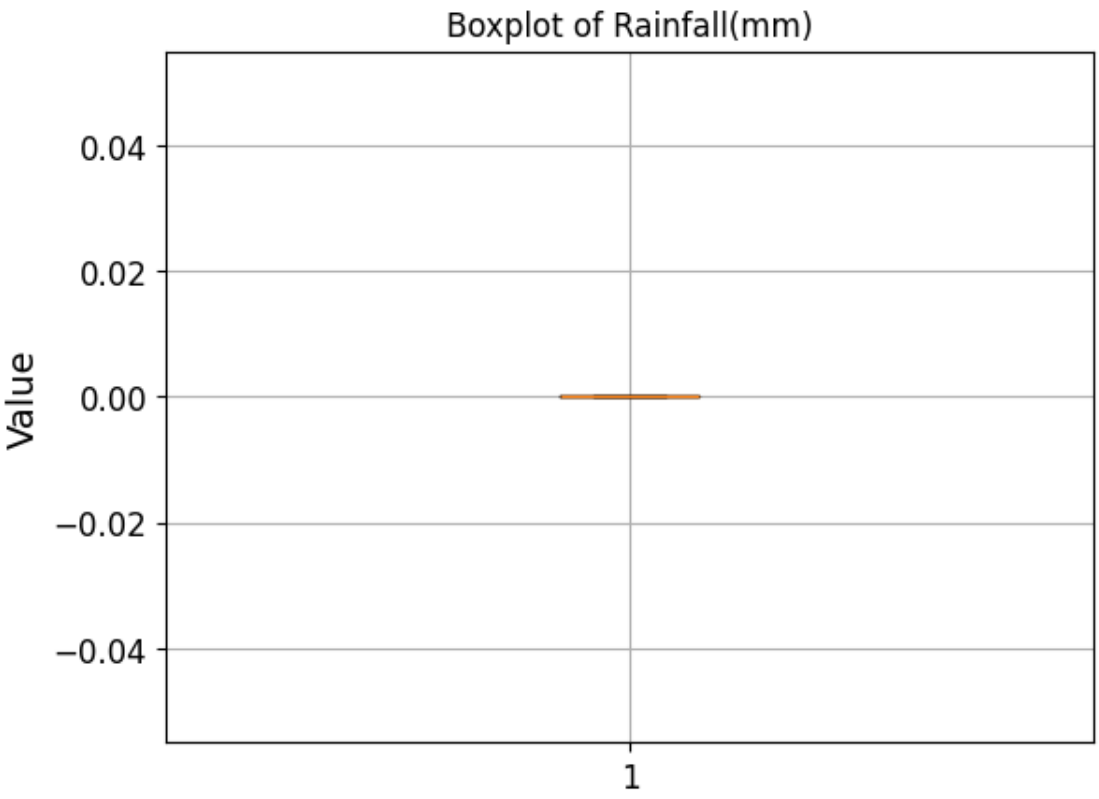

Boxplot of Humidity (%)
Value
100
80
60
40
20
0
1

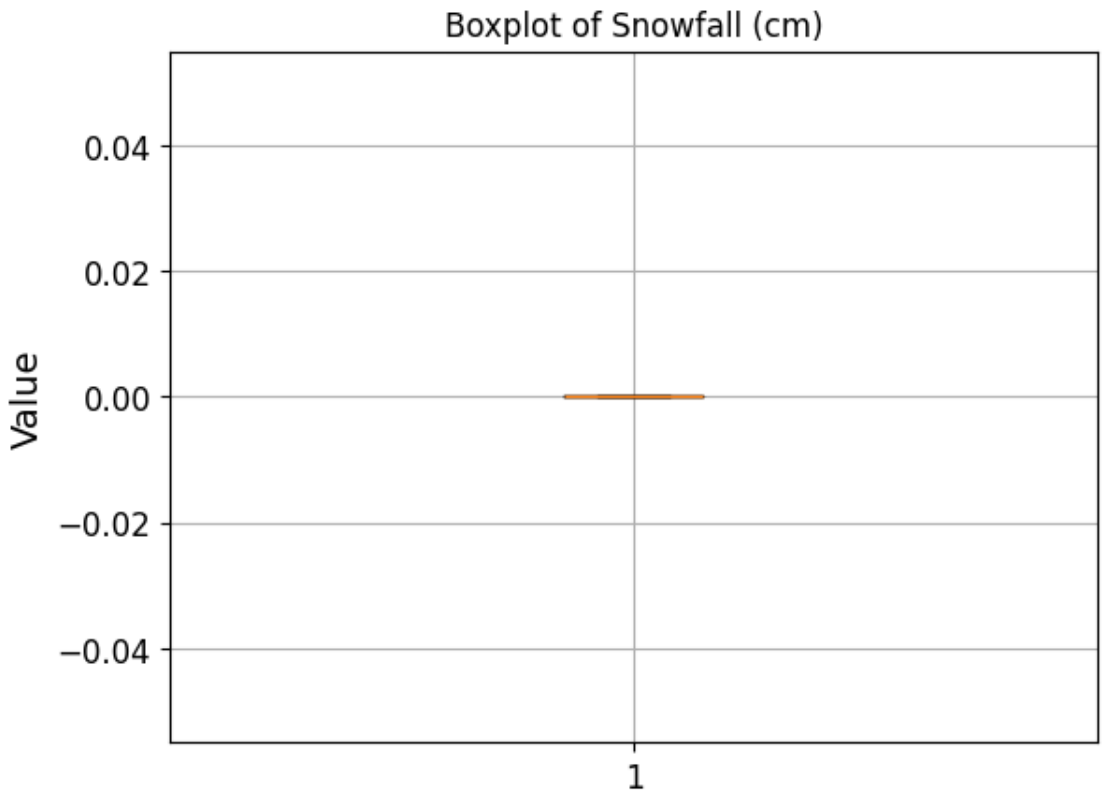

Boxplot of Wind speed (m/s)
Value
4
3
2
1
0
1


Boxplot of Visibility (10m)
Value
2000
1750
1500
1250
1000
750
500
250
0
1


Boxplot of Dew point temperature (C)
Value
20
10
0
−10
−20
1


Boxplot of Solar Radiation (MJ/m2)
Value
2.0
1.5
1.0
0.5
0.0
1


Boxplot of Rainfall(mm)
Value
0.04
0.02
0.00
−0.02
−0.04
1



[13]:
```
# Save the cleaned data to CSV
df.to_csv('./CleanedSeoulBikeData.csv')
```

1.4 Step 2: Pre-process the data and perform the first fit (20%)

1.4.1 2.1 Imputation and Pre-Processing

Make sure that you have set any problematic values in the numerical data to `np.nan` and then write code for a **sklearn *pipeline* that will perform imputation** to replace problematic entries (nan values) with an appropriate **median** value ***and* do any other pre-processing** that you think should be used.

[14]:
```
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler

preprocessing_pipeline = Pipeline([
    ('imputer', SimpleImputer(strategy='median')),
    ('standard scaler', StandardScaler(),),
])
```

1.4.2 2.2 Predicting bike rentals

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. **Using the pipeline** you wrote above, pre-process and fit a ***linear regression*** **model** to the data in an appropriate way. After this, **calculate and print the RMSE of the fit to the training data**.

To act as a simple baseline for comparison purposes, **also calculate and print the RMSE** that you would get if *all* the predictions were set to be the **mean of the training targets** (i.e. bike rentals).

[15]:
```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(df.drop('Rented Bike␣
↪Count', axis=1), df['Rented Bike Count'], test_size=0.2)
```

[16]:
```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import root_mean_squared_error

lr_pipeline = Pipeline([
    ('preprocess', preprocessing_pipeline,),
    ('classifier', LinearRegression(),)
])
lr_pipeline.fit(X_train, y_train)

train_predictions = lr_pipeline.predict(X_train)
rmse = root_mean_squared_error(y_train, train_predictions)
print(f"RMSE: {rmse}")
```

```
RMSE: 398.5393217753638
```

[17]:
```
mean_bike_rentals = np.mean(df['Rented Bike Count'])
baseline_predictions = np.full(y_train.shape, mean_bike_rentals)

baseline_rmse = root_mean_squared_error(y_train, baseline_predictions)
print(f"Baseline RMSE: {baseline_rmse}")
```

```
Baseline RMSE: 599.8395250091874
```

1.5 Step 3: Hyper-parameter optimisation (30%)

Use ChatGPT (along with any modifications that you require) to create and run code (using sklearn pipelines) that will do the following: - fit a **linear regression** and a **Support Vector Regression** method to the data using **10-fold cross validation** for each model - display the **mean and standard deviation** of the **RMSE values** for each model (at baseline) in the *appropriate datasets* - perform a **hyper-parameter optimisation** on each model using **GridSearch** - display the **mean and standard deviation** of the **RMSE values** for each model (after optimisation) in

the *appropriate datasets* - choose the **best model** and **visualise the results** with a single graphic of your choice

Display the ChatGPT prompt and the **code**, *including any fixes* that you needed to make to get the code to work, along with the **outputs** obtained by running the code.

1.5.1 ChatGPT prompt:

Create sklearn pipelines in order to fit both a linear regression and support vector regression model to a dataframe using 10-fold cross validation. Display the mean and standard deviation of the RMSE values for each model at baseline. Perform hyper-parameter optimisation on each model using GridSearch. Display the mean and standard deviation of the RMSE values for each model after the optimisation. Include a visualisation of the results of the best model.

The ChatGPT generated code was as follows:

```
import numpy as np
import pandas as pd
from sklearn.model_selection import cross_val_score, GridSearchCV, KFold
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.svm import SVR
from sklearn.metrics import mean_squared_error, make_scorer
import matplotlib.pyplot as plt

# Assuming 'df' is your DataFrame and 'target' is the name of your target column
X = df.drop(columns=['target'])
y = df['target']

# RMSE scoring function
rmse_scorer = make_scorer(mean_squared_error, squared=False)

# 10-fold CV setup
cv = KFold(n_splits=10, shuffle=True, random_state=42)

# Pipelines
linreg_pipeline = Pipeline([
    ('scaler', StandardScaler()),
    ('regressor', LinearRegression())
])

svr_pipeline = Pipeline([
    ('scaler', StandardScaler()),
    ('regressor', SVR())
])

# Cross-validation
linreg_scores = cross_val_score(linreg_pipeline, X, y, cv=cv, scoring=rmse_scorer)
```

```
svr_scores = cross_val_score(svr_pipeline, X, y, cv=cv, scoring=rmse_scorer)

print("Linear Regression Baseline RMSE: {:.3f} ± {:.3f}".format(linreg_scores.mean(), linreg_s
print("SVR Baseline RMSE: {:.3f} ± {:.3f}".format(svr_scores.mean(), svr_scores.std()))

# Grid search for Linear Regression (no params to tune really, but we include to keep consiste
linreg_grid = GridSearchCV(
    linreg_pipeline,
    param_grid={},  # no hyperparameters to tune
    cv=cv,
    scoring=rmse_scorer
)

# Grid search for SVR
svr_param_grid = {
    'regressor__C': [0.1, 1, 10],
    'regressor__epsilon': [0.01, 0.1, 1],
    'regressor__kernel': ['linear', 'rbf']
}

svr_grid = GridSearchCV(
    svr_pipeline,
    param_grid=svr_param_grid,
    cv=cv,
    scoring=rmse_scorer
)

linreg_grid.fit(X, y)
svr_grid.fit(X, y)

print("\nLinear Regression (GridSearch) RMSE: {:.3f} ± {:.3f}".format(
    -linreg_grid.best_score_, linreg_scores.std()))

print("SVR (GridSearch) RMSE: {:.3f} ± {:.3f}".format(
    -svr_grid.best_score_, svr_grid.cv_results_['std_test_score'][svr_grid.best_index_]))
)

# Predict using the best SVR model
best_model = svr_grid.best_estimator_
y_pred = best_model.predict(X)

plt.figure(figsize=(8, 6))
plt.scatter(y, y_pred, alpha=0.7)
plt.plot([y.min(), y.max()], [y.min(), y.max()], 'r--', lw=2)
plt.xlabel('Actual')
plt.ylabel('Predicted')
plt.title('SVR: Actual vs Predicted')
plt.grid(True)
```

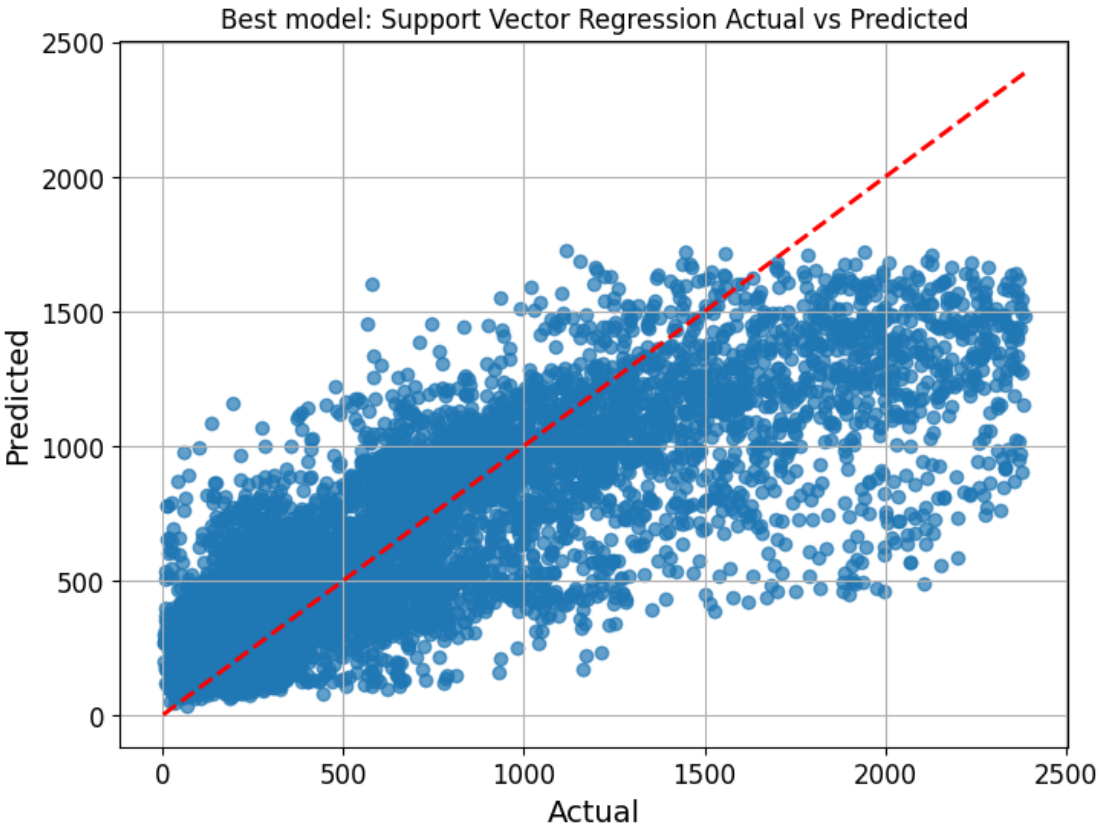
`plt.show()`

However, this code needed updating as the use of `make_scorer(mean_squared_error, squared=False)` resulted in warnings. Instead, the code should be updated to `make_scorer(root_mean_squared_error)`. The code also includes an extra bracket which would result in an error after printing the GridSearch mean and standard deviation results. While the code also does print the mean and standard deviation, it does not format it in an easy to read method. The code was updated as below to address these points.

[18]:
```
from sklearn.model_selection import cross_val_score, GridSearchCV, KFold
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.svm import SVR
import matplotlib.pyplot as plt

# Assuming 'df' is your DataFrame and 'target' is the name of your target column
X = df.drop(columns=['Rented Bike Count'])
y = df['Rented Bike Count']

# 10-fold CV setup
cv = KFold(n_splits=10, shuffle=True)

# Pipelines
linreg_pipeline = Pipeline([
    ('scaler', StandardScaler()),
    ('regressor', LinearRegression())
])

svr_pipeline = Pipeline([
    ('scaler', StandardScaler()),
    ('regressor', SVR())
])

# Cross-validation
linreg_scores = cross_val_score(linreg_pipeline, X, y, cv=cv,␣
 ↪scoring='neg_root_mean_squared_error')
svr_scores = cross_val_score(svr_pipeline, X, y, cv=cv,␣
 ↪scoring='neg_root_mean_squared_error')

print("Linear Regression Baseline RMSE Mean: {:.3f}, Standard Deviation: {:.
 ↪3f}".format(-linreg_scores.mean(), linreg_scores.std()))
print("SVR Baseline RMSE Mean: {:.3f}, Standard Deviation: {:.3f}".
 ↪format(-svr_scores.mean(), svr_scores.std()))

# Grid search for Linear Regression
lr_param_grid = {
    'regressor__fit_intercept': [True, False]
```

```
}
linreg_grid = GridSearchCV(
    linreg_pipeline,
    param_grid=lr_param_grid,
    cv=cv,
    scoring='neg_root_mean_squared_error'
)

# Grid search for SVR
svr_param_grid = {
    'regressor__C': [0.1, 1, 10],
    'regressor__epsilon': [0.01, 0.1, 1],
    'regressor__kernel': ['linear', 'rbf']
}

svr_grid = GridSearchCV(
    svr_pipeline,
    param_grid=svr_param_grid,
    cv=cv,
    scoring='neg_root_mean_squared_error'
)

linreg_grid.fit(X, y)
svr_grid.fit(X, y)

print("\nLinear Regression (Optimised) RMSE Mean: {:.3f}, Standard Deviation: {:
 ↪.3f}".format(-linreg_grid.best_score_, linreg_scores.std()))

print("SVR (Optimised) RMSE Mean: {:.3f}, Standard Deviation: {:.3f}".
 ↪format(-svr_grid.best_score_, svr_grid.
 ↪cv_results_['std_test_score'][svr_grid.best_index_]))

# Predict using the best SVR model
best_model = linreg_grid.best_estimator_ if -linreg_grid.best_score_ <␣
 ↪-svr_grid.best_score_ else svr_grid.best_estimator_
best_model_name = 'Linear Regression' if -linreg_grid.best_score_ < -svr_grid.
 ↪best_score_ else 'Support Vector Regression'
y_pred = best_model.predict(X)

plt.figure(figsize=(8, 6))
plt.scatter(y, y_pred, alpha=0.7)
plt.plot([y.min(), y.max()], [y.min(), y.max()], 'r--', lw=2)
plt.xlabel('Actual')
plt.ylabel('Predicted')
plt.title(f'Best model: {best_model_name} Actual vs Predicted')
plt.grid(True)
plt.show()
```

```
Linear Regression Baseline RMSE Mean: 398.044, Standard Deviation: 16.682
SVR Baseline RMSE Mean: 501.195, Standard Deviation: 11.850

Linear Regression (Optimised) RMSE Mean: 398.092, Standard Deviation: 16.682
SVR (Optimised) RMSE Mean: 370.258, Standard Deviation: 18.207
```



1.6 Step 4: Further improvements (10%)

Consider the code that you obtained from ChatGPT above and find one error, or one thing that could be improved, or one reasonable alternative (even if it might not necessarily lead to an improvement). **Describe this error/improvement/alternative in the box below.**

There were a number of issues that I found with the ChatGPT generated code. Mainly, the use of root mean squared error in the cross validation was not functioning as expected. Through some searching online, I found that the cross validation function always aims to maximise the scoring function given, which does not help when dealing with the root mean squared error, as a lower error indicates better performance. To fix this, I updated the code to use the negative root mean squared error instead. I also found that the generated code did not compare the linear regression model to the support vector regression model, and it would only display the support vector regression model as the best performing model. I updated the code to choose the best model out of either the linear regression model or the support vector regression model depending on which achieved

the lower error rate. I then updated the code to output which model it was and the graph of the actual compared to predicted values.