Overview

Overview Alerts 18 Reproduction

Dataset statistics

Number of variables	19
Number of observations	8760
Missing cells	0
Missing cells (%)	0.0%
Duplicate rows	0
Duplicate rows (%)	0.0%
Total size in memory	1.3 MiB
Average record size in memory	152.0 B

Variable types

Numeric	12
Categorical	7

Variables

Select Columns

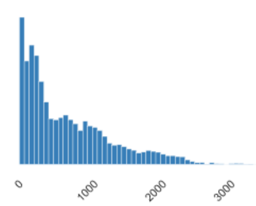
Rented Bike Count

Real number (ℝ)

HIGH CORRELATION ZEROS

Distinct	2166
Distinct (%)	24.7%
Missing	0
Missing (%)	0.0%
Infinite	0
Infinite (%)	0.0%
Mean	704.60205

Minimum	0
Maximum	3556
Zeros	295
Zeros (%)	3.4%
Negative	0
Negative (%)	0.0%
Memory size	68.6 KiB



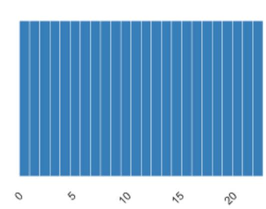
More details

Hour

Real number (ℝ)

Distinct	24
Distinct (%)	0.3%
Missing	0
Missing (%)	0.0%
Infinite	0
Infinite (%)	0.0%
Mean	11.5

Minimum	0
Maximum	23
Zeros	365
Zeros (%)	4.2%
Negative	0
Negative (%)	0.0%
Memory size	68.6 KiB



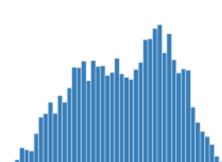
More details

Temperature(°C)

Real number (ℝ)

Distinct	546
Distinct (%)	6.2%
Missing	0
Missing (%)	0.0%
Infinite	0

Minimum	-17.8
Maximum	39.4
Zeros	21
Zeros (%)	0.2%
Negative	1433

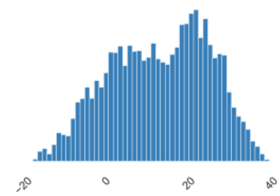


Temperature(°C)

Real number (ℝ)

Distinct	546
Distinct (%)	6.2%
Missing	0
Missing (%)	0.0%
Infinite	0
Infinite (%)	0.0%
Mean	12.882922

Minimum	-17.8
Maximum	39.4
Zeros	21
Zeros (%)	0.2%
Negative	1433
Negative (%)	16.4%
Memory size	68.6 KiB



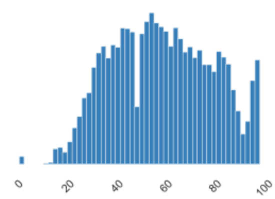
More details

Humidity(%)

Real number (ℝ)

Distinct	90
Distinct (%)	1.0%
Missing	0
Missing (%)	0.0%
Infinite	0
Infinite (%)	0.0%
Mean	58.226256

Minimum	0
Maximum	98
Zeros	17
Zeros (%)	0.2%
Negative	0
Negative (%)	0.0%
Memory size	68.6 KiB



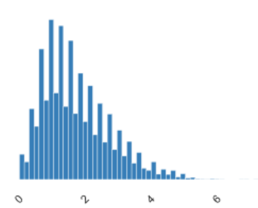
More details

Wind speed (m/s)

Real number (ℝ)

Distinct	65
Distinct (%)	0.7%
Missing	0
Missing (%)	0.0%
Infinite	0
Infinite (%)	0.0%
Mean	1.7249087

Minimum	0
Maximum	7.4
Zeros	74
Zeros (%)	0.8%
Negative	0
Negative (%)	0.0%
Memory size	68.6 KiB



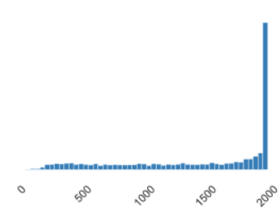
More details

Visibility (10m)

Real number (ℝ)

Distinct	1789
Distinct (%)	20.4%
Missing	0
Missing (%)	0.0%
Infinite	0
Infinite (%)	0.0%
Mean	1436.8258

Minimum	27
Maximum	2000
Zeros	0
Zeros (%)	0.0%
Negative	0
Negative (%)	0.0%
Memory size	68.6 KiB



More details

Dew point temperature(°C)

Real number (ℝ)

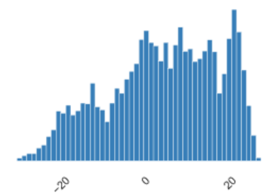
Distinct	556

Minimum	-30.6

Dew point temperature(°C)

Real number (ℝ)

Distinct	556	Minimum	-30.6
Distinct (%)	6.3%	Maximum	27.2
Missing	0	Zeros	60
Missing (%)	0.0%	Zeros (%)	0.7%
Infinite	0	Negative	3138
Infinite (%)	0.0%	Negative (%)	35.8%
Mean	4.0738128	Memory size	68.6 KiB

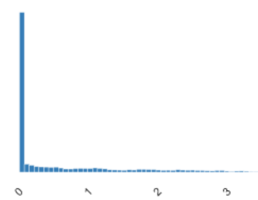


More details

Solar Radiation (MJ/m2)

Real number (ℝ)

Distinct	345	Minimum	0
Distinct (%)	3.9%	Maximum	3.52
Missing	0	Zeros	4300
Missing (%)	0.0%	Zeros (%)	49.1%
Infinite	0	Negative	0
Infinite (%)	0.0%	Negative (%)	0.0%
Mean	0.56911073	Memory size	68.6 KiB

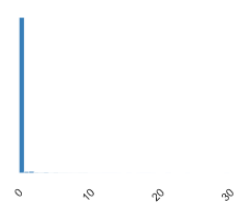


More details

Rainfall(mm)

Real number (ℝ)

Distinct	61	Minimum	0
Distinct (%)	0.7%	Maximum	35
Missing	0	Zeros	8232
Missing (%)	0.0%	Zeros (%)	94.0%
Infinite	0	Negative	0
Infinite (%)	0.0%	Negative (%)	0.0%
Mean	0.14868721	Memory size	68.6 KiB

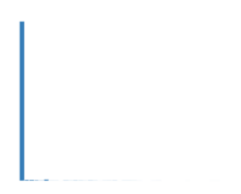


More details

Snowfall (cm)

Real number (ℝ)

Distinct	51	Minimum	0
Distinct (%)	0.6%	Maximum	8.8
Missing	0	Zeros	8317
Missing (%)	0.0%	Zeros (%)	94.9%
Infinite	0	Negative	0
Infinite (%)	0.0%	Negative (%)	0.0%
Mean	0.075068493	Memory size	68.6 KiB



More details

Holiday

Categorical

Distinct	2

Holiday

Categorical

Distinct	2
Distinct (%)	< 0.1%
Missing	0
Missing (%)	0.0%
Memory size	68.6 KiB

More details

Functioning Day

Categorical

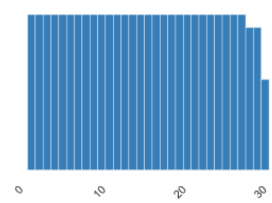
Distinct	2
Distinct (%)	< 0.1%
Missing	0
Missing (%)	0.0%
Memory size	68.6 KiB

More details

Day

Real number (ℝ)

Distinct	31	Minimum	1
Distinct (%)	0.4%	Maximum	31
Missing	0	Zeros	0
Missing (%)	0.0%	Zeros (%)	0.0%
Infinite	0	Negative	0
Infinite (%)	0.0%	Negative (%)	0.0%
Mean	15.720548	Memory size	68.6 KiB

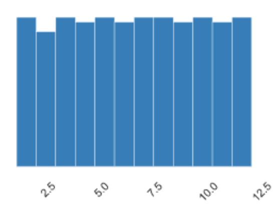


More details

Month

Real number (ℝ)

Distinct	12	Minimum	1
Distinct (%)	0.1%	Maximum	12
Missing	0	Zeros	0
Missing (%)	0.0%	Zeros (%)	0.0%
Infinite	0	Negative	0
Infinite (%)	0.0%	Negative (%)	0.0%
Mean	6.5260274	Memory size	68.6 KiB



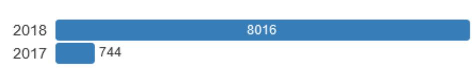
More details

Year

Categorical

HIGH CORRELATION IMBALANCE

Distinct	2
Distinct (%)	< 0.1%
Missing	0
Missing (%)	0.0%



Year

Categorical

HIGH CORRELATION IMBALANCE

Distinct	2
Distinct (%)	< 0.1%
Missing	0
Missing (%)	0.0%
Memory size	68.6 KiB

More details

Autumn

Categorical

Distinct	2
Distinct (%)	< 0.1%
Missing	0
Missing (%)	0.0%
Memory size	68.6 KiB

More details

Spring

Categorical

Distinct	2
Distinct (%)	< 0.1%
Missing	0
Missing (%)	0.0%
Memory size	68.6 KiB

More details

Summer

Categorical

Distinct	2
Distinct (%)	< 0.1%
Missing	0
Missing (%)	0.0%
Memory size	68.6 KiB

More details

Winter

Categorical

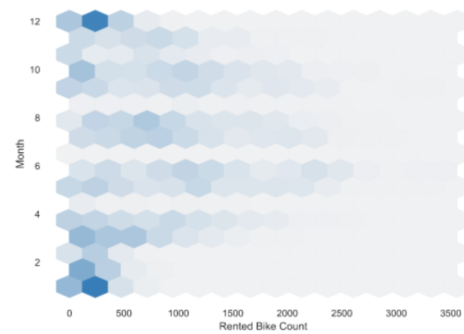
Distinct	2
Distinct (%)	< 0.1%
Missing	0
Missing (%)	0.0%
Memory size	68.6 KiB

More details

More details

Interactions

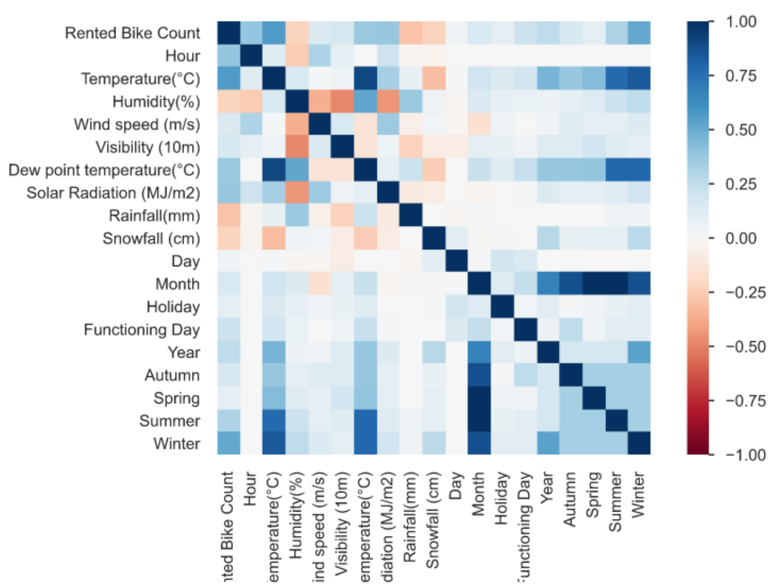
Rented Bike Count	Month
Hour	Rented Bike Count
Temperature(°C)	Hour
Humidity(%)	Temperature(°C)
Wind speed (m/s)	Humidity(%)
Visibility (10m)	Wind speed (m/s)
Dew point temperature(°C	Visibility (10m)
Solar Radiation (MJ/m2)	Dew point temperature(°C
Rainfall(mm)	Solar Radiation (MJ/m2)
Snowfall (cm)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



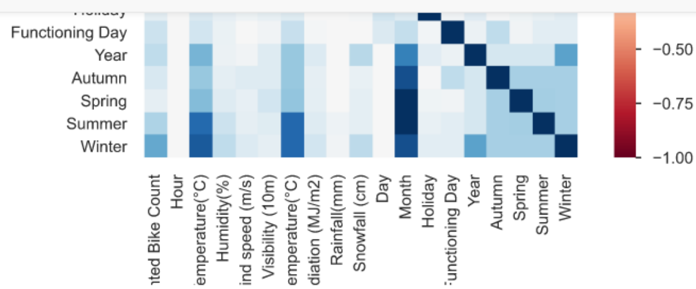
Correlations

Auto

Heatmap Table

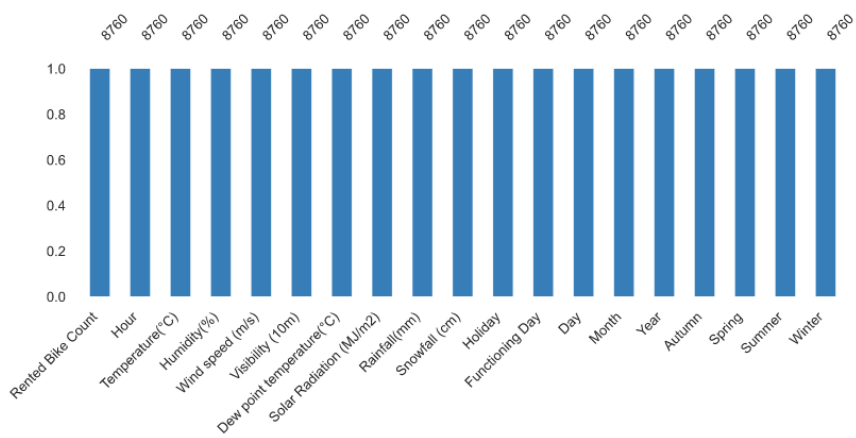


Missing values



Missing values

Count Matrix



A simple visualization of nullity by column.

Sample

First rows Last rows

	Rented Bike Count	Hour	Temperature(°C)	Humidity(%)	Wind speed (m/s)	Visibility (10m)	Dew point temperature(°
0	254	0	-5.2	37	2.2	2000	-17.6
1	204	1	-5.5	38	0.8	2000	-17.6
2	173	2	-6.0	39	1.0	2000	-17.7
3	107	3	-6.2	40	0.9	2000	-17.6
4	78	4	-6.0	36	2.3	2000	-18.6
5	100	5	-6.4	37	1.5	2000	-18.7
6	181	6	-6.6	35	1.3	2000	-19.5
7	460	7	-7.4	38	0.9	2000	-19.3
8	930	8	-7.6	37	1.1	2000	-19.8
9	490	9	-6.5	27	0.5	1928	-22.4

data_preparation

March 13, 2023

Data Preparation

For the SeoulBikeData.csv dataset, I will convert the categorical attributes to quantitative attributes such that they are in a valid format for use in building the regression models.

[46]:
```python
# Import the required modules
import pandas as pd
```

[47]:
```python
# Read in the dataset into a pandas dataframe

seoul_bike_data = pd.read_csv("SeoulBikeData.csv", encoding="ansi")
seoul_bike_data.head()
```

[47]:
```
         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            0.0  Winter  No Holiday
1                      0.0           0.0            0.0  Winter  No Holiday
2                      0.0           0.0            0.0  Winter  No Holiday
3                      0.0           0.0            0.0  Winter  No Holiday
4                      0.0           0.0            0.0  Winter  No Holiday

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

```
4            Yes
```

For this dataset, the categorical variables which I will transform are Date, Seasons, Holiday, and Functioning Day.

[48]:
```
seoul_bike_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8760 entries, 0 to 8759
Data columns (total 14 columns):
 #   Column                     Non-Null Count  Dtype  
---  ------                     --------------  -----  
 0   Date                       8760 non-null   object 
 1   Rented Bike Count          8760 non-null   int64  
 2   Hour                       8760 non-null   int64  
 3   Temperature(°C)            8760 non-null   float64
 4   Humidity(%)                8760 non-null   int64  
 5   Wind speed (m/s)           8760 non-null   float64
 6   Visibility (10m)           8760 non-null   int64  
 7   Dew point temperature(°C)  8760 non-null   float64
 8   Solar Radiation (MJ/m2)    8760 non-null   float64
 9   Rainfall(mm)               8760 non-null   float64
 10  Snowfall (cm)              8760 non-null   float64
 11  Seasons                    8760 non-null   object 
 12  Holiday                    8760 non-null   object 
 13  Functioning Day            8760 non-null   object 
dtypes: float64(6), int64(4), object(4)
memory usage: 958.2+ KB
```

1. Date Attribute First, for the Date attribute, I will transform it to get the Day, Month, and Year as these 3 new attributes will be quantitative.

[49]:
```
# Use pandas to convert the Date attribute to a datetime data type
seoul_bike_data["Date"] = pd.to_datetime(seoul_bike_data["Date"], format="%d/%m/
↪%Y")
seoul_bike_data.head()
```

[49]:
```
        Date  Rented Bike Count  Hour  Temperature(°C)  Humidity(%)  \
0 2017-12-01                254     0             -5.2           37   
1 2017-12-01                204     1             -5.5           38   
2 2017-12-01                173     2             -6.0           39   
3 2017-12-01                107     3             -6.2           40   
4 2017-12-01                 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            0.0  Winter  No Holiday
1                      0.0           0.0            0.0  Winter  No Holiday
2                      0.0           0.0            0.0  Winter  No Holiday
3                      0.0           0.0            0.0  Winter  No Holiday
4                      0.0           0.0            0.0  Winter  No Holiday

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

[50]:
```
seoul_bike_data.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   datetime64[ns]
 1   Rented Bike Count          8760 non-null   int64
 2   Hour                       8760 non-null   int64
 3   Temperature(°C)            8760 non-null   float64
 4   Humidity(%)                8760 non-null   int64
 5   Wind speed (m/s)           8760 non-null   float64
 6   Visibility (10m)           8760 non-null   int64
 7   Dew point temperature(°C)  8760 non-null   float64
 8   Solar Radiation (MJ/m2)    8760 non-null   float64
 9   Rainfall(mm)               8760 non-null   float64
 10  Snowfall (cm)              8760 non-null   float64
 11  Seasons                    8760 non-null   object
 12  Holiday                    8760 non-null   object
 13  Functioning Day            8760 non-null   object
dtypes: datetime64[ns](1), float64(6), int64(4), object(3)
memory usage: 958.2+ KB
```

[51]:
```
# Create the Day, Month, and Year quantitative attributes from the Date␣
 ↪attribute
seoul_bike_data["Day"] = seoul_bike_data["Date"].dt.day
seoul_bike_data["Month"] = seoul_bike_data["Date"].dt.month
seoul_bike_data["Year"] = seoul_bike_data["Date"].dt.year
```

```
seoul_bike_data.head()
```

[51]:
```
         Date  Rented Bike Count  Hour  Temperature(°C)  Humidity(%)  \
0 2017-12-01                 254     0             -5.2           37
1 2017-12-01                 204     1             -5.5           38
2 2017-12-01                 173     2             -6.0           39
3 2017-12-01                 107     3             -6.2           40
4 2017-12-01                  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            0.0  Winter  No Holiday
1                      0.0           0.0            0.0  Winter  No Holiday
2                      0.0           0.0            0.0  Winter  No Holiday
3                      0.0           0.0            0.0  Winter  No Holiday
4                      0.0           0.0            0.0  Winter  No Holiday

  Functioning Day  Day  Month  Year
0             Yes    1     12  2017
1             Yes    1     12  2017
2             Yes    1     12  2017
3             Yes    1     12  2017
4             Yes    1     12  2017
```

[52]:
```
# Drop the Date attribute as it is no longer required
seoul_bike_data.drop(columns=["Date"], inplace=True)
```

2. Seasons Attribute Second, for the Seasons attribute, I will use one hot encoding to create 4 new binary attributes to represent each Seasons value. This is because the Seasons attribute is not ordinal hence one hot encoding is a suitable method to ensure the 4 new attributes are valid for regression model building.

[53]:
```
# To display all unique values of the Seasons attribute
seoul_bike_data["Seasons"].unique()
```

[53]:
```
array(['Winter', 'Spring', 'Summer', 'Autumn'], dtype=object)
```

[54]:
```
# Conduct the one hot encoding using pandas' get_dummies method
seoul_bike_data = pd.get_dummies(seoul_bike_data, prefix="", prefix_sep="",␣
 ↪columns=["Seasons"])
seoul_bike_data.head()
```

[54]:
```
   Rented Bike Count  Hour  Temperature(°C)  Humidity(%)  Wind speed (m/s)  \
0                254     0             -5.2           37               2.2
1                204     1             -5.5           38               0.8
2                173     2             -6.0           39               1.0
3                107     3             -6.2           40               0.9
4                 78     4             -6.0           36               2.3

   Visibility (10m)  Dew point temperature(°C)  Solar Radiation (MJ/m2)  \
0              2000                      -17.6                      0.0
1              2000                      -17.6                      0.0
2              2000                      -17.7                      0.0
3              2000                      -17.6                      0.0
4              2000                      -18.6                      0.0

   Rainfall(mm)  Snowfall (cm)     Holiday Functioning Day  Day  Month  Year  \
0           0.0            0.0  No Holiday             Yes    1     12  2017
1           0.0            0.0  No Holiday             Yes    1     12  2017
2           0.0            0.0  No Holiday             Yes    1     12  2017
3           0.0            0.0  No Holiday             Yes    1     12  2017
4           0.0            0.0  No Holiday             Yes    1     12  2017

   Autumn  Spring  Summer  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
```

3. Holiday and Functioning Day Attributes Third, I will focus on transforming the Holiday and Functioning Day attributes to binary quantitative attributes. Specifically, for the Holiday attribute, if the record in our dataset is "No Holiday" then it will be mapped as 0 and if it is "Holiday" then it will be mapped as 1.

[55]:
```
# To display the binary values of the Holiday attribute
seoul_bike_data["Holiday"].unique()
```

[55]:
```
array(['No Holiday', 'Holiday'], dtype=object)
```

[56]:
```
# Conduct the mapping using the map method and a dictionary for the binary key␣
↪value pair
seoul_bike_data["Holiday"] = seoul_bike_data["Holiday"].map({"No Holiday":0,␣
↪"Holiday":1})
seoul_bike_data["Holiday"].unique()
```

[56]:
```
array([0, 1], dtype=int64)
```

For the Functioning Day attribute, I will map the value "No" to 0 and "Yes" to 1.

[57]:
```python
# To display the binary values of the Functioning Day attribute
seoul_bike_data["Functioning Day"].unique()
```

[57]: array(['Yes', 'No'], dtype=object)

[58]:
```python
# Conduct the mapping using the map method and a dictionary for the binary key␣
↪value pair
seoul_bike_data["Functioning Day"] = seoul_bike_data["Functioning Day"].
↪map({"No":0, "Yes":1})
seoul_bike_data["Functioning Day"].unique()
```

[58]: array([1, 0], dtype=int64)

4. Prepared Dataset: seoul_bike_data_prepared.csv Finally, I have completed data preparation and will now convert the dataframe containing the Seoul Bike data into a .csv flat file for the next step which is predictive modeling.

[59]:
```python
# To display the prepared dataset
seoul_bike_data.head()
```

[59]:
```
   Rented Bike Count  Hour  Temperature(°C)  Humidity(%)  Wind speed (m/s)  \
0                254     0             -5.2           37               2.2
1                204     1             -5.5           38               0.8
2                173     2             -6.0           39               1.0
3                107     3             -6.2           40               0.9
4                 78     4             -6.0           36               2.3

   Visibility (10m)  Dew point temperature(°C)  Solar Radiation (MJ/m2)  \
0              2000                      -17.6                      0.0
1              2000                      -17.6                      0.0
2              2000                      -17.7                      0.0
3              2000                      -17.6                      0.0
4              2000                      -18.6                      0.0

   Rainfall(mm)  Snowfall (cm)  Holiday  Functioning Day  Day  Month  Year  \
0           0.0            0.0        0                1    1     12  2017
1           0.0            0.0        0                1    1     12  2017
2           0.0            0.0        0                1    1     12  2017
3           0.0            0.0        0                1    1     12  2017
4           0.0            0.0        0                1    1     12  2017

   Autumn  Spring  Summer  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
```

[60]:

```
# Output the prepared dataset into a .csv in the same directory as this Jupyter␣
↪Notebook
seoul_bike_data.to_csv("seoul_bike_data_prepared.csv", index=False)
```

baseline_models

March 13, 2023

Predictive Modeling - Baseline Models

This Jupyter Notebook contains the 3 regression models using all attributes of the prepared dataset (seoul_bike_data_prepared.csv). Namely, linear regression, regression tree, and k-nearest neighbours. Since all attributes of the dataset will be used in the regression analyses, I have therefore named these as baseline models. 10-fold cross validation will be used.

The intent is to compare these baseline models with selected features models in terms of model performance. Selected features models in this case means regression models that I will build with only attributes that are deemed statistically significant.

Prepared Dataset

[2]:
```
# Import required modules
import pandas as pd
import numpy as np
import matplotlib.pylab as plt
import seaborn as sns
from sklearn.model_selection import cross_validate
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.neighbors import KNeighborsRegressor
```

[3]:
```
# Read in the dataset
seoul_bike = pd.read_csv("seoul_bike_data_prepared.csv", encoding="utf-8")
seoul_bike.head()
```

[3]:
```
   Rented Bike Count  Hour  Temperature(°C)  Humidity(%)  Wind speed (m/s)  \
0                254     0             -5.2           37               2.2
1                204     1             -5.5           38               0.8
2                173     2             -6.0           39               1.0
3                107     3             -6.2           40               0.9
4                 78     4             -6.0           36               2.3

   Visibility (10m)  Dew point temperature(°C)  Solar Radiation (MJ/m2)  \
0              2000                      -17.6                      0.0
1              2000                      -17.6                      0.0
2              2000                      -17.7                      0.0
3              2000                      -17.6                      0.0
4              2000                      -18.6                      0.0
```

```
   Rainfall(mm)  Snowfall (cm)  Holiday  Functioning Day  Day  Month  Year  \
0           0.0            0.0        0                1    1     12  2017
1           0.0            0.0        0                1    1     12  2017
2           0.0            0.0        0                1    1     12  2017
3           0.0            0.0        0                1    1     12  2017
4           0.0            0.0        0                1    1     12  2017

   Autumn  Spring  Summer  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
```

1. Linear Regression Model

[4]:
```
# Assign the target variable, independent variables, and desired performance␣
↪metrics
target_name = "Rented Bike Count"
target_variable = seoul_bike[target_name]
independent_variables = seoul_bike.drop(columns=target_name)
performance_metrics = ["r2", "neg_root_mean_squared_error",␣
↪"neg_mean_absolute_error"]
```

[5]:
```
# Build the linear regression model and use cross validation
lr_model = LinearRegression()
lr_scores = cross_validate(
    estimator=lr_model,
    X=independent_variables,
    y=target_variable,
    cv=10,
    scoring=performance_metrics
)
lr_scores
```

[5]:
```
{'fit_time': array([0.01155519, 0.0045011 , 0.00401855, 0.0040009 , 0.00402284,
        0.00400305, 0.00452137, 0.00350046, 0.00451279, 0.00401044]),
 'score_time': array([0.00201273, 0.00149941, 0.0010004 , 0.0010004 ,
0.00149989,
        0.00150108, 0.0009985 , 0.0010035 , 0.00149918, 0.00102329]),
 'test_r2': array([-12.37966   ,  -4.18163805,  -0.37609285,   0.45292654,
          0.06343567,  -0.01655934,  -0.01606348,   0.44079453,
          0.57989526,   0.41778741]),
 'test_neg_root_mean_squared_error': array([-572.20720111, -270.87176628,
-360.12129946, -385.88570039,
        -706.69277924, -791.0456306 , -583.69217493, -514.45991858,
        -457.16038902, -374.40065613]),
```

```
 'test_neg_mean_absolute_error': array([-497.93935515, -230.05636989,
-289.2433827 , -304.93846573,
        -543.31749225, -598.69609858, -504.0723804 , -425.02535882,
        -346.68697398, -272.11677486])}
```

[6]:
```python
# Array of performance metrics scores
# Note: abs() is applied to scores returned by sklearn that are the negative␣
↪value of the metric
lr_r2_scores = lr_scores["test_r2"]
lr_root_mean_squared_error_scores =␣
↪abs(lr_scores["test_neg_root_mean_squared_error"])
lr_mean_absolute_error_scores = abs(lr_scores["test_neg_mean_absolute_error"])
```

[7]:
```python
# Dataframe capturing the overall performance metrics of the linear regression␣
↪model
lr_metrics = pd.DataFrame(
    {
        "Model": ["Linear Regression"],
        "R Squared": lr_r2_scores.mean(),
        "Root Mean Squared Error": lr_root_mean_squared_error_scores.mean(),
        "Mean Absolute Error": lr_mean_absolute_error_scores.mean(),
    }
)
lr_metrics
```

[7]:

	Model	R Squared	Root Mean Squared Error	Mean Absolute Error
0	Linear Regression	-1.501517	501.653752	401.209265

2. Regression Tree Model

[8]:
```python
# Build the regression tree model and use cross validation
# random_state is set to 3 for reproducibility
rt_model = DecisionTreeRegressor(random_state=3)
rt_scores = cross_validate(
    estimator=rt_model,
    X=independent_variables,
    y=target_variable,
    cv=10,
    scoring=performance_metrics
)
rt_scores
```

[8]:
```
{'fit_time': array([0.04061127, 0.0391283 , 0.04418612, 0.04155159, 0.04010677,
        0.04256129, 0.04359698, 0.03919291, 0.03850412, 0.04002833]),
 'score_time': array([0.00201488, 0.00149965, 0.00149894, 0.00150299,
0.00150299,
        0.00150061, 0.00149894, 0.00150061, 0.00149941, 0.00149941]),
 'test_r2': array([ 0.11716893, -0.35848985, -0.01613418,  0.11199505,
```

```
0.63390278,
        0.64301151,  0.46790528,  0.58146458,  0.71809834,  0.38548109]),
 'test_neg_root_mean_squared_error': array([-146.98383877, -138.69423269,
-309.45711228, -491.63586313,
        -441.83494511, -468.77266399, -422.39417766, -445.07371382,
        -374.48861783, -384.64795771]),
 'test_neg_mean_absolute_error': array([-108.02739726,  -96.21347032,
-186.07191781, -322.14383562,
        -311.36757991, -325.85388128, -279.55821918, -300.53082192,
        -229.55365297, -263.68378995])}
```

[9]:
```python
# Array of performance metrics scores
# Note: abs() is applied to scores returned by sklearn that are the negative␣
 ↪value of the metric
rt_r2_scores = rt_scores["test_r2"]
rt_root_mean_squared_error_scores =␣
 ↪abs(rt_scores["test_neg_root_mean_squared_error"])
rt_mean_absolute_error_scores = abs(rt_scores["test_neg_mean_absolute_error"])
```

[10]:
```python
# Dataframe capturing the overall performance metrics of the regression tree␣
 ↪model
rt_metrics = pd.DataFrame(
    {
        "Model": ["Regression Tree"],
        "R Squared": rt_r2_scores.mean(),
        "Root Mean Squared Error": rt_root_mean_squared_error_scores.mean(),
        "Mean Absolute Error": rt_mean_absolute_error_scores.mean(),
    }
)
rt_metrics
```

[10]:

	Model	R Squared	Root Mean Squared Error	Mean Absolute Error
0	Regression Tree	0.32844	362.398312	242.300457

3. K-Nearest Neighbours Model

[11]:
```python
# Build the k-nearest neighbours model and use cross validation
# I will use a range of k values (1 to 20) to determine which k contributes to␣
 ↪the best performing model

# Create empty lists to append each metric
knn_k = []
knn_r2 = []
knn_root_mean_squared_error = []
knn_mean_absolute_error = []

for k in range(1, 21):
    knn_model = KNeighborsRegressor(n_neighbors = k)
```

```
    knn_scores = cross_validate(
        estimator=knn_model,
        X=independent_variables,
        y=target_variable,
        cv=10,
        scoring=performance_metrics
    )

    # Array of performance metrics scores
    # Note: abs() is applied to scores returned by sklearn that are the␣
 ↪negative value of the metric
    knn_r2_scores = knn_scores["test_r2"]
    knn_root_mean_squared_error_scores =␣
 ↪abs(knn_scores["test_neg_root_mean_squared_error"])
    knn_mean_absolute_error_scores =␣
 ↪abs(knn_scores["test_neg_mean_absolute_error"])

    # Average the scores from each fold of the cross validation
    # Append the metrics to lists
    knn_k.append(k)
    knn_r2.append(knn_r2_scores.mean())
    knn_root_mean_squared_error.append(knn_root_mean_squared_error_scores.
 ↪mean())
    knn_mean_absolute_error.append(knn_mean_absolute_error_scores.mean())
```

[12]:
```
# Dataframe capturing the overall performance metrics of the k-nearest␣
 ↪neighbours model
knn_metrics = pd.DataFrame(
    {
        "Model": "K-Nearest Neighbours",
        "k": knn_k,
        "R Squared": knn_r2,
        "Root Mean Squared Error": knn_root_mean_squared_error,
        "Mean Absolute Error": knn_mean_absolute_error,
    }
)
knn_metrics
```

[12]:
```
                  Model  k  R Squared  Root Mean Squared Error  \
0  K-Nearest Neighbours  1  -1.085467               596.061047
1  K-Nearest Neighbours  2  -0.586315               528.561680
2  K-Nearest Neighbours  3  -0.400623               502.123941
3  K-Nearest Neighbours  4  -0.284610               483.807584
4  K-Nearest Neighbours  5  -0.251234               475.589239
5  K-Nearest Neighbours  6  -0.210121               469.568640
6  K-Nearest Neighbours  7  -0.179231               466.370528
7  K-Nearest Neighbours  8  -0.169162               464.046049
```

```
8   K-Nearest Neighbours   9  -0.158443             462.093335
9   K-Nearest Neighbours  10  -0.156480             461.219524
10  K-Nearest Neighbours  11  -0.153458             460.387483
11  K-Nearest Neighbours  12  -0.149372             458.850701
12  K-Nearest Neighbours  13  -0.144513             457.627993
13  K-Nearest Neighbours  14  -0.149054             456.636305
14  K-Nearest Neighbours  15  -0.152750             456.879584
15  K-Nearest Neighbours  16  -0.154608             456.526530
16  K-Nearest Neighbours  17  -0.154704             455.835054
17  K-Nearest Neighbours  18  -0.160048             455.993327
18  K-Nearest Neighbours  19  -0.163734             456.069071
19  K-Nearest Neighbours  20  -0.166518             455.877304

    Mean Absolute Error
0            424.247146
1            386.013185
2            369.978729
3            358.960674
4            353.850411
5            350.092028
6            348.113258
7            346.585873
8            346.519673
9            346.696370
10           346.343192
11           346.010578
12           345.872111
13           345.616120
14           346.179863
15           346.261558
16           346.350020
17           346.643398
18           346.785809
19           347.192620
```

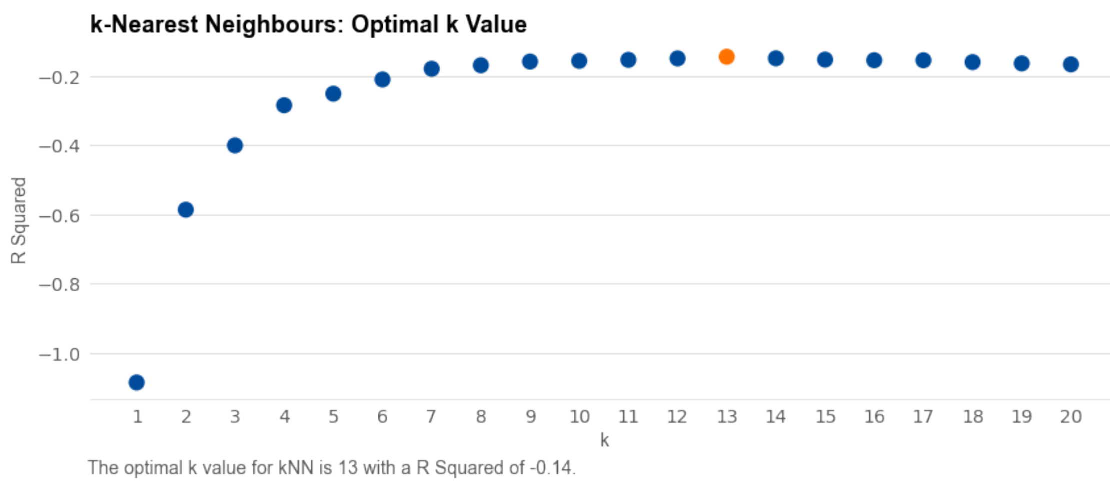
[13]:

```python
# Plot the k values by R squared to visualize the performance of each k-nearest␣
 ↪neighbours model
value_r2 = knn_metrics["R Squared"] == knn_metrics["R Squared"].max()
knn_metrics["colour_r2"] = np.where(value_r2 == True, "#FF7200", "#004C9B")
knn_plt = sns.regplot(
    data=knn_metrics,
    x="k",
    y="R Squared",
    fit_reg=False,
    scatter_kws={
        "alpha": 1,
        "facecolors": knn_metrics["colour_r2"],
```

```
        "linewidths": 0,
        "s": 150,
        "zorder": 10,
    }
)

# Add title
knn_plt.set_title(
    "k-Nearest Neighbours: Optimal k Value",
    font="Arial",
    fontsize="18",
    fontweight="bold",
    loc="left"
)

# X-axis
plt.xlabel(
    "k", color="#595959", font="Arial", fontsize="14",␣
↪horizontalalignment="center"
)

# Y-axis
plt.ylabel(
    "R Squared",
    color="#595959",
    font="Arial",
    fontsize="14",
    horizontalalignment="center"
)

# Ticks
plt.xticks(range(int(knn_metrics["k"].min()), int(knn_metrics["k"].max()) + 1,␣
↪1))
plt.tick_params(colors="#595959", bottom=False, left=False, labelsize="14")

# Add horizontal gridlines
plt.grid(axis="y", color="#D9D9D9")

# Set plot size
knn_plt.figure.set_size_inches(14, 5)

# Spines
sns.despine(left=True)
for _, s in knn_plt.spines.items():
    s.set_color("#D9D9D9")

# Get row with max R squared
```

```
max_y_row = knn_metrics.loc[knn_metrics["R Squared"].idxmax()]

# Get the max R squared value and the corresponding x value
max_y_value = np.round(max_y_row["R Squared"], 2)
corresponding_x_value = np.round(max_y_row["k"], 2)

# Add a caption
plt.text(
    0,
    -1.35,
    f"The optimal k value for kNN is {corresponding_x_value} with a R Squared␣
  ↪of {max_y_value}.",
    color="#595959",
    font="Arial",
    fontsize="14"
)

knn_plt
```

[13]: <AxesSubplot: title={'left': 'k-Nearest Neighbours: Optimal k Value'},
xlabel='k', ylabel='R Squared'>



[14]:
```
# For the k-nearest neighbours models, I will choose the one with the highest R␣
  ↪squared value
knn_metrics_sorted = knn_metrics.sort_values("R Squared", ascending=False)
knn_metrics_sorted.drop(columns=["k", "colour_r2"], inplace=True)

# Subset the dataframe to keep the record with the highest R Squared
highest_knn_metrics = knn_metrics_sorted.head(1)
highest_knn_metrics
```

[14]:

```
                      Model  R Squared  Root Mean Squared Error  \
12  K-Nearest Neighbours  -0.144513               457.627993

    Mean Absolute Error
12           345.872111
```

Performance Metrics The performance metrics of all regression models are displayed below as a pandas dataframe and a Tableau visualization.

[15]:

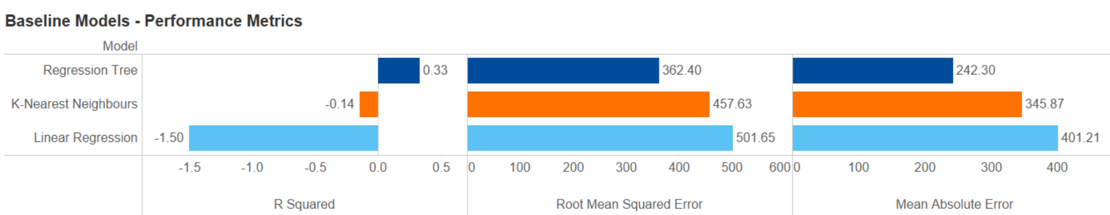
```python
# Create a baseline_metrics dataframe to capture the performance metrics of all
 ↪regression models
baseline_metrics = pd.concat([lr_metrics, rt_metrics, highest_knn_metrics])
baseline_metrics.reset_index(drop=True, inplace=True)
baseline_metrics
```

[15]:

```
                     Model  R Squared  Root Mean Squared Error  \
0     Linear Regression  -1.501517               501.653752
1       Regression Tree   0.328440               362.398312
2  K-Nearest Neighbours  -0.144513               457.627993

   Mean Absolute Error
0           401.209265
1           242.300457
2           345.872111
```

[16]:

```python
# Export the baseline_metrics dataframe to a .csv flat file
baseline_metrics.to_csv("baseline_metrics.csv", index=False)
```



Based on the performance metrics of each model, the regression tree model is the best performing, followed by k-nearest neighbours in second place, and linear regression being last place. Notably, the R squared values for both k-nearest neighbours and linear regression are negative. As per the sklearn module, the best score for the R squared is 1.0 and a negative score means that the regression model is arbitrarily worse. In terms of the error metrics, root mean squared error and mean absolute error, the lower the error values the better performing the models are.

Therefore, it can be concluded that using all features of the prepared dataset is suboptimal for regression tree, and provides very poor performance for the k-nearest neighbours and linear regression models.

This highlights the need for variable selection using the prepared dataset (seoul_bike_data_prepared.csv) in order to optimize the performance of all regression models.

selected_features_models

March 13, 2023

Predictive Modeling - Selected Features Models

This Jupyter Notebook contains the 3 regression models using selected features of the prepared dataset (seoul_bike_data_prepared.csv). Namely, linear regression, regression tree, and k-nearest neighbours. Since select features of the dataset will be used in the regression analyses, I have therefore named these as selected features models. 10-fold cross validation will be used.

The intent is to compare these selected features models with the baseline models in terms of model performance. Selected features models in this case means regression models that I will build with only attributes that are deemed statistically significant.

Prepared Dataset

[1]:
```
# Import required modules
import pandas as pd
import numpy as np
import matplotlib.pylab as plt
import seaborn as sns
from sklearn.model_selection import cross_validate
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.neighbors import KNeighborsRegressor
from mlxtend.feature_selection import SequentialFeatureSelector as SFS
from mlxtend.plotting import plot_sequential_feature_selection as plot_sfs
import matplotlib.pyplot as pyplot
```

[2]:
```
# Read in the dataset
seoul_bike = pd.read_csv("seoul_bike_data_prepared.csv", encoding="utf-8")
seoul_bike.head()
```

[2]:
```
   Rented Bike Count  Hour  Temperature(°C)  Humidity(%)  Wind speed (m/s)  \
0                254     0             -5.2           37               2.2
1                204     1             -5.5           38               0.8
2                173     2             -6.0           39               1.0
3                107     3             -6.2           40               0.9
4                 78     4             -6.0           36               2.3

   Visibility (10m)  Dew point temperature(°C)  Solar Radiation (MJ/m2)  \
0              2000                      -17.6                      0.0
1              2000                      -17.6                      0.0
```

```
2               2000                          -17.7                        0.0
3               2000                          -17.6                        0.0
4               2000                          -18.6                        0.0

   Rainfall(mm)  Snowfall (cm)  Holiday  Functioning Day  Day  Month  Year  \
0           0.0            0.0        0                1    1     12  2017
1           0.0            0.0        0                1    1     12  2017
2           0.0            0.0        0                1    1     12  2017
3           0.0            0.0        0                1    1     12  2017
4           0.0            0.0        0                1    1     12  2017

   Autumn  Spring  Summer  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
```

1. Linear Regression Model

[3]:
```
# Assign the target variable, independent variables, and desired performance␣
↪metrics
target_name = "Rented Bike Count"
target_variable = seoul_bike[target_name]
independent_variables = seoul_bike.drop(columns=target_name)
performance_metrics = ["r2", "neg_root_mean_squared_error",␣
↪"neg_mean_absolute_error"]
```

Feature Selection Sequential Forward Selection (SFS) will be conducted by using the mlxtend module's SequentialFeatureSelector. SFS is a sequential feature selection algorithm which automatically selects a subset of features that are the most important for the regression model. In particular, SFS adds one feature at a time based on the R squared performance metric. This reduces the model's error by discarding insignificant features and as a result, improves the performance metrics.

[4]:
```
# mlxtend modules feature selection tool: Sequential Feature Selector
lr_sfs = SFS(estimator=LinearRegression(),
             k_features=(1,18),
             forward=True,
             floating=False,
             scoring="r2",
             cv=10)

# Perform the feature selection
lr_sfs.fit(independent_variables, target_variable)
```

[4]:
```
SequentialFeatureSelector(cv=10, estimator=LinearRegression(),
                          k_features=(1, 18), scoring='r2')
```

[5]:
```
# To visualize the Sequential Feature Selector tool results
lr_sfs_results = pd.DataFrame.from_dict(lr_sfs.get_metric_dict()).T
lr_sfs_results
```

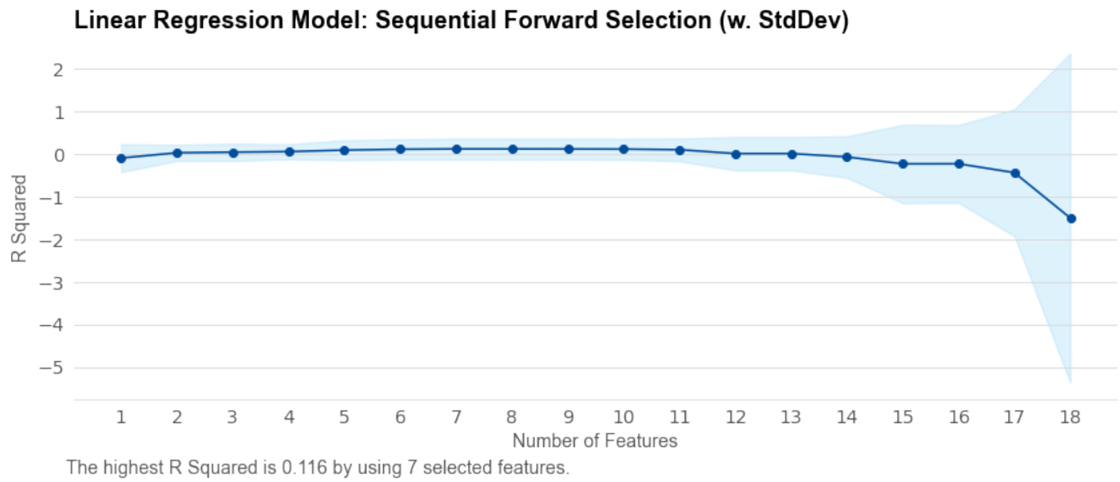
[5]:
```
                                           feature_idx  \
1                                                 (1,)
2                                               (1, 4)
3                                            (1, 4, 7)
4                                        (1, 4, 7, 17)
5                                    (1, 4, 7, 10, 17)
6                                (1, 4, 7, 10, 14, 17)
7                             (1, 4, 7, 9, 10, 14, 17)
8                          (1, 4, 6, 7, 9, 10, 14, 17)
9                       (1, 4, 6, 7, 8, 9, 10, 14, 17)
10                  (1, 4, 6, 7, 8, 9, 10, 11, 14, 17)
11              (1, 4, 6, 7, 8, 9, 10, 11, 13, 14, 17)
12          (1, 4, 6, 7, 8, 9, 10, 11, 13, 14, 15, 17)
13      (1, 4, 6, 7, 8, 9, 10, 11, 13, 14, 15, 16, 17)
14  (1, 3, 4, 6, 7, 8, 9, 10, 11, 13, 14, 15, 16, 17)
15  (1, 2, 3, 4, 6, 7, 8, 9, 10, 11, 13, 14, 15, 1…
16  (1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 13, 14, 15…
17  (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 13, 14,…
18  (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13,…

                                             cv_scores avg_score  \
1   [-0.3368353988674757, -0.918271532762825, 0.19… -0.097393
2   [-0.017461238538012402, -0.23849300655795957, …  0.025709
3   [0.010939557944065204, -0.2462070899370501, 0…  0.040204
4   [-0.014180339709577883, -0.08067897017882486, …  0.053138
5   [-0.03862803818220639, -0.06337005489155612, 0…  0.087882
6   [-0.0011678909621402056, -0.14506010569897887,…  0.108275
7   [0.038193597168479365, -0.1261734721658878, 0…  0.116176
8   [0.0004548344983327235, -0.13342980581946895, …  0.115867
9   [-0.005743059846415877, -0.13461998812823261, …  0.114459
10  [-0.025528276671164196, -0.12605749383568243, …  0.112318
11  [-0.025528276671174632, -0.283343590416704l, 0…  0.097686
12  [-0.04250607825731101, -0.8969077226324087, 0…  0.007405
13  [-0.0425060782573079, -0.896907722632655, 0.24…  0.007405
14  [-0.6991402286332673, -1.1238234474386553, 0.1… -0.071273
15  [-1.612694820869097, -2.359029144458398, 0.174… -0.233478
16  [-1.616223784580654, -2.3300170968614142, 0.17… -0.232455
17  [-2.401920146558146, -4.164705102239599, -0.05…  -0.43978
18  [-12.379659996789123, -4.181638046309049, -0.3… -1.501517

                                 feature_names  ci_bound   std_dev  \
1                            (Temperature(°C),)  0.242919   0.32707
2             (Temperature(°C), Visibility (10m))  0.141344  0.190308
```

```
3   (Temperature(°C), Visibility (10m), Rainfall(mm))  0.152056  0.204731
4   (Temperature(°C), Visibility (10m), Rainfall(m…  0.132518  0.178424
5   (Temperature(°C), Visibility (10m), Rainfall(m…  0.174499  0.234947
6   (Temperature(°C), Visibility (10m), Rainfall(m…  0.178259   0.24001
7   (Temperature(°C), Visibility (10m), Rainfall(m…   0.18264   0.24591
8   (Temperature(°C), Visibility (10m), Solar Radi…   0.18398  0.247713
9   (Temperature(°C), Visibility (10m), Solar Radi…   0.18395  0.247673
10  (Temperature(°C), Visibility (10m), Solar Radi…  0.183854  0.247544
11  (Temperature(°C), Visibility (10m), Solar Radi…  0.198289   0.26698
12  (Temperature(°C), Visibility (10m), Solar Radi…  0.289845  0.390251
13  (Temperature(°C), Visibility (10m), Solar Radi…  0.289845  0.390251
14  (Temperature(°C), Wind speed (m/s), Visibility…  0.361759  0.487078
15  (Temperature(°C), Humidity(%), Wind speed (m/s…  0.682669  0.919157
16  (Temperature(°C), Humidity(%), Wind speed (m/s…   0.67776  0.912547
17  (Hour, Temperature(°C), Humidity(%), Wind spee…  1.106082  1.489245
18  (Hour, Temperature(°C), Humidity(%), Wind spee…  2.869342  3.863325

     std_err
1   0.109023
2   0.063436
3   0.068244
4   0.059475
5   0.078316
6   0.080003
7    0.08197
8   0.082571
9   0.082558
10  0.082515
11  0.088993
12  0.130084
13  0.130084
14  0.162359
15  0.306386
16  0.304182
17  0.496415
18  1.287775
```

[29]:

```python
# Global pyplot parameter
pyplot.rcParams["axes.edgecolor"] = "#D9D9D9"

# Plotting the Results
plot_lr_sfs_results = plot_sfs(lr_sfs.get_metric_dict(), kind="std_dev",␣
 ↪color="#004C9B", bcolor="#5bc2f4", figsize=[14, 5])

# Add title
pyplot.title(
    "Linear Regression Model: Sequential Forward Selection (w. StdDev)",
```

```
    font="Arial",
    fontsize="18",
    fontweight="bold",
    loc="left"
)

# X-axis
pyplot.xlabel(
    "Number of Features", color="#595959", font="Arial", fontsize="14",␣
 ↪horizontalalignment="center"
)

# Y-axis
pyplot.ylabel(
    "R Squared",
    color="#595959",
    font="Arial",
    fontsize="14",
    horizontalalignment="center"
)

# Ticks
pyplot.tick_params(colors="#595959", bottom=False, left=False, labelsize="14")

# Add horizontal gridlines
pyplot.grid(axis="y", color="#D9D9D9")

# Spines
sns.despine(left=True)

# Add a caption
pyplot.text(
    0,
    -7.5,
    f"The highest R Squared is {np.round(lr_sfs.k_score_,3)} by using␣
 ↪{len(lr_sfs.k_feature_idx_)} selected features.",
    color="#595959",
    font="Arial",
    fontsize="14"
)

pyplot.show()
```



[7]:
```python
# Subset the dataframe to get the record with the best R Squared results
lr_sfs_results_sorted = lr_sfs_results.sort_values("avg_score", ascending=False)
highest_lr_r2 = lr_sfs_results_sorted.head(1)
highest_lr_r2
```

[7]:
```
                    feature_idx  \
7  (1, 4, 7, 9, 10, 14, 17)

                                                   cv_scores avg_score  \
7  [0.038193597168479365, -0.1261734721658878, 0…  0.116176

                                             feature_names ci_bound   std_dev  \
7  (Temperature(°C), Visibility (10m), Rainfall(m…  0.18264  0.24591

   std_err
7  0.08197
```

[8]:
```python
# The records of the selected features
lr_selected_features =␣
↪independent_variables[list(highest_lr_r2["feature_names"].values[0])]
lr_selected_features.head()
```

[8]:
```
   Temperature(°C)  Visibility (10m)  Rainfall(mm)  Holiday  Functioning Day  \
0             -5.2              2000           0.0        0                1
1             -5.5              2000           0.0        0                1
2             -6.0              2000           0.0        0                1
3             -6.2              2000           0.0        0                1
4             -6.0              2000           0.0        0                1

   Autumn  Winter
```

```
0        0        1
1        0        1
2        0        1
3        0        1
4        0        1
```

[9]:
```python
# Build the linear regression model and use cross validation
lr_model = LinearRegression()
lr_scores = cross_validate(
    estimator=lr_model,
    X=lr_selected_features,
    y=target_variable,
    cv=10,
    scoring=performance_metrics
)
lr_scores
```

[9]:
```
{'fit_time': array([0.00453258, 0.00300002, 0.00252867, 0.00199986, 0.00253892,
        0.00251341, 0.00200129, 0.00302982, 0.00200129, 0.001508  ]),
 'score_time': array([0.00150323, 0.00099874, 0.00150776, 0.00099993,
0.00404382,
        0.00099945, 0.0010078 , 0.001508  , 0.00149894, 0.00152254]),
 'test_r2': array([ 0.0381936 , -0.12617347,  0.29447898,  0.35963657,
0.19260796,
         0.04516704, -0.45533749,  0.12238027,  0.40105209,  0.28975816]),
 'test_neg_root_mean_squared_error': array([-153.41739785, -126.27949886,
-257.85758852, -417.492927  ,
        -656.1514295 , -766.6530513 , -698.56256408, -644.49422079,
        -545.86384631, -413.52229945]),
 'test_neg_mean_absolute_error': array([-107.3295154 ,  -97.28586879,
-201.16345441, -321.02988833,
        -511.21040635, -577.54951072, -619.2063464 , -554.64286717,
        -435.2508013 , -308.99506717])}
```

[10]:
```python
# Array of performance metrics scores
# Note: abs() is applied to scores returned by sklearn that are the negative␣
 ↪value of the metric
lr_r2_scores = lr_scores["test_r2"]
lr_root_mean_squared_error_scores =␣
 ↪abs(lr_scores["test_neg_root_mean_squared_error"])
lr_mean_absolute_error_scores = abs(lr_scores["test_neg_mean_absolute_error"])
```

[69]:
```python
# Dataframe capturing the overall performance metrics of the linear regression␣
 ↪model
lr_metrics = pd.DataFrame(
    {
        "Model": ["Linear Regression"],
```

```
        "R Squared": lr_r2_scores.mean(),
        "Root Mean Squared Error": lr_root_mean_squared_error_scores.mean(),
        "Mean Absolute Error": lr_mean_absolute_error_scores.mean(),
        "Number of Features": len(lr_sfs.k_feature_idx_),
        "Feature Indices": [lr_sfs.k_feature_idx_],
        "Feature Names": [lr_sfs.k_feature_names_]
    }
)
lr_metrics
```

[69]:
```
               Model  R Squared  Root Mean Squared Error  Mean Absolute Error  \
0  Linear Regression   0.116176               468.029482           373.366373

   Number of Features           Feature Indices  \
0                   7  (1, 4, 7, 9, 10, 14, 17)

                                     Feature Names
0  (Temperature(°C), Visibility (10m), Rainfall(m…
```

2. Regression Tree Model

Feature Selection Sequential Forward Selection (SFS) will be conducted by using the mlxtend module's SequentialFeatureSelector. SFS is a sequential feature selection algorithm which automatically selects a subset of features that are the most important for the regression model. In particular, SFS adds one feature at a time based on the R squared performance metric. This reduces the model's error by discarding insignificant features and as a result, improves the performance metrics.

[12]:
```
# mlxtend modules feature selection tool: Sequential Feature Selector
rt_sfs = SFS(estimator=DecisionTreeRegressor(random_state=3),
             k_features=(1,18),
             forward=True,
             floating=False,
             scoring="r2",
             cv=10)

# Perform the feature selection
rt_sfs.fit(independent_variables, target_variable)
```

[12]:
```
SequentialFeatureSelector(cv=10,
                          estimator=DecisionTreeRegressor(random_state=3),
                          k_features=(1, 18), scoring='r2')
```

[13]:
```
# To visualize the Sequential Feature Selector tool results
rt_sfs_results = pd.DataFrame.from_dict(rt_sfs.get_metric_dict()).T
rt_sfs_results
```

[13]:

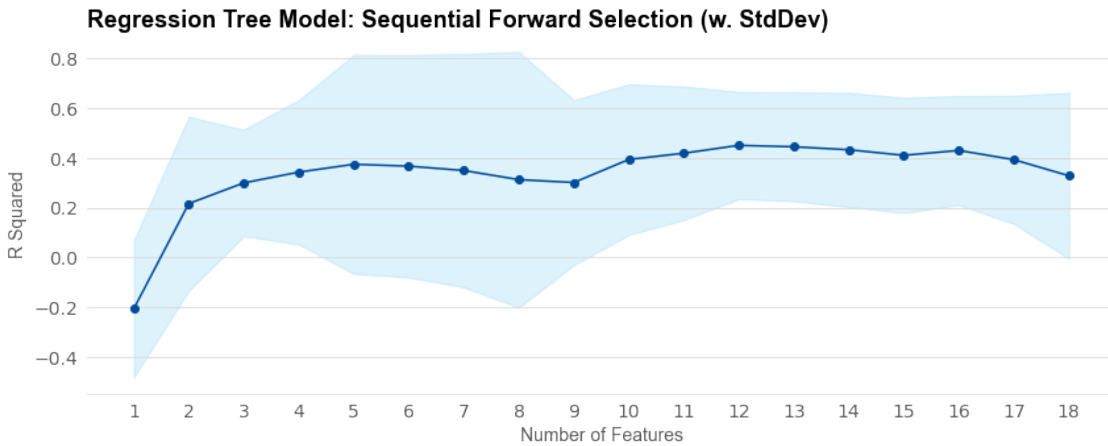
```
                                              feature_idx  \
1                                                  (17,)
2                                                (0, 17)
3                                            (0, 16, 17)
4                                         (0, 7, 16, 17)
5                                     (0, 7, 10, 16, 17)
6                                 (0, 7, 10, 13, 16, 17)
7                              (0, 7, 8, 10, 13, 16, 17)
8                           (0, 7, 8, 9, 10, 13, 16, 17)
9                        (0, 1, 7, 8, 9, 10, 13, 16, 17)
10                    (0, 1, 2, 7, 8, 9, 10, 13, 16, 17)
11                (0, 1, 2, 7, 8, 9, 10, 13, 14, 16, 17)
12             (0, 1, 2, 6, 7, 8, 9, 10, 13, 14, 16, 17)
13         (0, 1, 2, 6, 7, 8, 9, 10, 13, 14, 15, 16, 17)
14      (0, 1, 2, 5, 6, 7, 8, 9, 10, 13, 14, 15, 16, 17)
15   (0, 1, 2, 3, 5, 6, 7, 8, 9, 10, 13, 14, 15, 16…
16   (0, 1, 2, 3, 5, 6, 7, 8, 9, 10, 11, 13, 14, 15…
17   (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 13, 14,…
18   (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13,…

                                               cv_scores avg_score  \
1   [-0.06178284381364185, -0.25318375175919794, -… -0.206011
2   [0.3986806973882365, 0.19908658027429715, -0.7…  0.216475
3   [0.3986806973882365, 0.19908658027429715, -0.2…  0.299796
4   [0.4149292463114904, 0.1737705108060431, -0.45…  0.343045
5   [0.4149292463114904, 0.1737705108060431, -0.87…  0.374553
6   [0.4149292463114904, 0.11311814771476014, -0.8…  0.367048
7   [0.4075799273024534, -0.013257495179018086, -0…  0.349629
8   [0.45535361183009726, -0.3821754258535781, -0…  0.312782
9   [0.11386365245038643, -0.21352497256649006, -0…  0.301336
10  [0.1757459296888947, -0.1166019136110068, -0.1…   0.39384
11  [0.19387731539100617, -0.1520598473318615, 0.1…   0.41904
12  [0.19067333453364177, 0.01956974753667373, 0.3…  0.450626
13  [0.17654687825509563, 0.022763132631624994, 0…  0.444913
14  [0.17390822620808588, -0.060981821435387884, 0…  0.432915
15  [0.13612132012107392, -0.0557058955368015, 0.3…  0.410525
16  [0.2706576817097469, -0.09312906993473158, 0.4…  0.430233
17  [0.2903717455105228, -0.1615740565661823, 0.11…  0.393023
18  [0.11716892715250127, -0.358489850263368, -0.0…   0.32844

                                           feature_names  ci_bound   std_dev  \
1                                              (Winter,)  0.204342  0.275129
2                                         (Hour, Winter)  0.259982  0.350043
3                                 (Hour, Summer, Winter)  0.158896  0.213941
4                   (Hour, Rainfall(mm), Summer, Winter)  0.215707  0.290431
5   (Hour, Rainfall(mm), Functioning Day, Summer, …  0.326822  0.440038
6   (Hour, Rainfall(mm), Functioning Day, Year, Su…   0.33196  0.446957
```

```
7   (Hour, Rainfall(mm), Snowfall (cm), Functionin…  0.348485  0.469205
8   (Hour, Rainfall(mm), Snowfall (cm), Holiday, F…  0.381782  0.514037
9   (Hour, Temperature(°C), Rainfall(mm), Snowfall…  0.246263  0.331573
10  (Hour, Temperature(°C), Humidity(%), Rainfall(…  0.224895  0.302801
11  (Hour, Temperature(°C), Humidity(%), Rainfall(…  0.199242  0.268262
12  (Hour, Temperature(°C), Humidity(%), Solar Rad…  0.160101  0.215562
13  (Hour, Temperature(°C), Humidity(%), Solar Rad…  0.162683  0.219038
14  (Hour, Temperature(°C), Humidity(%), Dew point…  0.170548  0.229628
15  (Hour, Temperature(°C), Humidity(%), Wind spee…  0.171972  0.231545
16  (Hour, Temperature(°C), Humidity(%), Wind spee…   0.16262  0.218954
17  (Hour, Temperature(°C), Humidity(%), Wind spee…  0.191165  0.257387
18  (Hour, Temperature(°C), Humidity(%), Wind spee…  0.247771  0.333603

     std_err
1    0.09171
2   0.116681
3   0.071314
4    0.09681
5   0.146679
6   0.148986
7   0.156402
8   0.171346
9   0.110524
10  0.100934
11  0.089421
12  0.071854
13  0.073013
14  0.076543
15  0.077182
16  0.072985
17  0.085796
18  0.111201
```

[40]:

```
# Global pyplot parameter
pyplot.rcParams["axes.edgecolor"] = "#D9D9D9"

# Plotting the Results
plot_rt_sfs_results = plot_sfs(rt_sfs.get_metric_dict(), kind="std_dev",␣
 ↪color="#004C9B", bcolor="#5bc2f4", figsize=[14, 5])

# Add title
pyplot.title(
    "Regression Tree Model: Sequential Forward Selection (w. StdDev)",
    font="Arial",
    fontsize="18",
    fontweight="bold",
    loc="left"
```

```
)

# X-axis
pyplot.xlabel(
    "Number of Features", color="#595959", font="Arial", fontsize="14",␣
 ↪horizontalalignment="center"
)

# Y-axis
pyplot.ylabel(
    "R Squared",
    color="#595959",
    font="Arial",
    fontsize="14",
    horizontalalignment="center"
)

# Ticks
pyplot.tick_params(colors="#595959", bottom=False, left=False, labelsize="14")

# Add horizontal gridlines
pyplot.grid(axis="y", color="#D9D9D9")

# Spines
sns.despine(left=True)

# Add a caption
pyplot.text(
    0,
    -0.87,
    f"The highest R Squared is {np.round(rt_sfs.k_score_,3)} by using␣
 ↪{len(rt_sfs.k_feature_idx_)} selected features.",
    color="#595959",
    font="Arial",
    fontsize="14"
)

pyplot.show()
```



The highest R Squared is 0.451 by using 12 selected features.

[14]:
```python
# Subset the dataframe to get the record with the best R Squared results
rt_sfs_results_sorted = rt_sfs_results.sort_values("avg_score", ascending=False)
highest_rt_r2 = rt_sfs_results_sorted.head(1)
highest_rt_r2
```

[14]:
```
                                           feature_idx  \
12  (0, 1, 2, 6, 7, 8, 9, 10, 13, 14, 16, 17)

                                                cv_scores avg_score  \
12  [0.19067333453364177, 0.01956974753667373, 0.3…  0.450626

                                           feature_names  ci_bound   std_dev  \
12  (Hour, Temperature(°C), Humidity(%), Solar Rad…  0.160101  0.215562

     std_err
12  0.071854
```

[15]:
```python
# The records of the selected features
rt_selected_features = independent_variables[list(highest_rt_r2["feature_names"].values[0])]
rt_selected_features.head()
```

[15]:
```
   Hour  Temperature(°C)  Humidity(%)  Solar Radiation (MJ/m2)  Rainfall(mm)  \
0     0             -5.2           37                      0.0           0.0
1     1             -5.5           38                      0.0           0.0
2     2             -6.0           39                      0.0           0.0
3     3             -6.2           40                      0.0           0.0
4     4             -6.0           36                      0.0           0.0

   Snowfall (cm)  Holiday  Functioning Day  Year  Autumn  Summer  Winter
```

0	0.0	0	1	2017	0	0	1
1	0.0	0	1	2017	0	0	1
2	0.0	0	1	2017	0	0	1
3	0.0	0	1	2017	0	0	1
4	0.0	0	1	2017	0	0	1

[16]:
```python
# Build the regression tree model and use cross validation
# random_state is set to 3 for reproducibility
rt_model = DecisionTreeRegressor(random_state=3)
rt_scores = cross_validate(
    estimator=rt_model,
    X=rt_selected_features,
    y=target_variable,
    cv=10,
    scoring=performance_metrics
)
rt_scores
```

[16]:
```
{'fit_time': array([0.02261305, 0.02108574, 0.02150464, 0.02118039, 0.02501607,
        0.01952267, 0.02003741, 0.02053094, 0.02002835, 0.02059603]),
 'score_time': array([0.00149989, 0.00150275, 0.00100374, 0.00150132,
0.00149989,
        0.00252652, 0.00150156, 0.00100136, 0.0015018 , 0.00100732]),
 'test_r2': array([0.19067333, 0.01956975, 0.36915237, 0.3618115 , 0.64874401,
        0.69024584, 0.53063535, 0.57789491, 0.72323866, 0.39429686]),
 'test_neg_root_mean_squared_error': array([-140.73194994, -117.82530021,
-243.83002257, -416.78333755,
        -432.78650869, -436.66033057, -396.71501422, -446.96769315,
        -371.0586084 , -381.87895085]),
 'test_neg_mean_absolute_error': array([ -97.57534247,  -78.53424658,
-157.88584475, -281.46004566,
        -295.06164384, -299.46118721, -239.42808219, -294.96347032,
        -230.15525114, -253.41210046])}
```

[17]:
```python
# Array of performance metrics scores
# Note: abs() is applied to scores returned by sklearn that are the negative␣
 ↪value of the metric
rt_r2_scores = rt_scores["test_r2"]
rt_root_mean_squared_error_scores =␣
 ↪abs(rt_scores["test_neg_root_mean_squared_error"])
rt_mean_absolute_error_scores = abs(rt_scores["test_neg_mean_absolute_error"])
```

[70]:
```python
# Dataframe capturing the overall performance metrics of the regression tree␣
 ↪model
rt_metrics = pd.DataFrame(
    {
        "Model": ["Regression Tree"],
```

```
        "R Squared": rt_r2_scores.mean(),
        "Root Mean Squared Error": rt_root_mean_squared_error_scores.mean(),
        "Mean Absolute Error": rt_mean_absolute_error_scores.mean(),
        "Number of Features": len(rt_sfs.k_feature_idx_),
        "Feature Indices": [rt_sfs.k_feature_idx_],
        "Feature Names": [rt_sfs.k_feature_names_]
    }
)
rt_metrics
```

[70]:

```
             Model  R Squared  Root Mean Squared Error  Mean Absolute Error  \
0  Regression Tree   0.450626               338.523772           222.793721

   Number of Features                           Feature Indices  \
0                  12  (0, 1, 2, 6, 7, 8, 9, 10, 13, 14, 16, 17)

                                       Feature Names
0  (Hour, Temperature(°C), Humidity(%), Solar Rad…
```

3. K-Nearest Neighbours Model

Feature Selection Sequential Forward Selection (SFS) will be conducted by using the mlxtend module's SequentialFeatureSelector. SFS is a sequential feature selection algorithm which automatically selects a subset of features that are the most important for the regression model. In particular, SFS adds one feature at a time based on the R squared performance metric. This reduces the model's error by discarding insignificant features and as a result, improves the performance metrics.

[42]:

```
# Build the k-nearest neighbours model and use cross validation
# I will use a range of k values (1 to 20) to determine which k contributes to␣
↪the best performing model

# Create empty lists to append each metric
knn_k = []
knn_r2 = []
knn_root_mean_squared_error = []
knn_mean_absolute_error = []
knn_number_of_features = []
knn_feature_indices = []
knn_feature_names = []

for k in range(1, 21):

    knn = KNeighborsRegressor(n_neighbors = k)

    # mlxtend modules feature selection tool: Sequential Feature Selector
    knn_sfs = SFS(estimator=knn,
```

```
                k_features=(1,18),
                forward=True,
                floating=False,
                scoring="r2",
                cv=10)

    # Perform the feature selection
    knn_sfs.fit(independent_variables, target_variable)

    # Sequential Feature Selector tool results
    knn_sfs_results = pd.DataFrame.from_dict(knn_sfs.get_metric_dict()).T

    # Subset the dataframe to get the record with the best R Squared results for the current k value
    knn_sfs_results_sorted = knn_sfs_results.sort_values("avg_score", ascending=False)
    highest_knn_r2 = knn_sfs_results_sorted.head(1)

    # Capture the records of the selected features
    knn_selected_features = independent_variables[list(highest_knn_r2["feature_names"].values[0])]

    knn_model = knn
    knn_scores = cross_validate(
        estimator=knn_model,
        X=knn_selected_features,
        y=target_variable,
        cv=10,
        scoring=performance_metrics
    )

    # Array of performance metrics scores
    # Note: abs() is applied to scores returned by sklearn that are the negative value of the metric
    knn_r2_scores = knn_scores["test_r2"]
    knn_root_mean_squared_error_scores = abs(knn_scores["test_neg_root_mean_squared_error"])
    knn_mean_absolute_error_scores = abs(knn_scores["test_neg_mean_absolute_error"])
    knn_k_number_of_features = len(knn_sfs.k_feature_idx_)
    knn_k_feature_indices = knn_sfs.k_feature_idx_
    knn_k_feature_names = knn_sfs.k_feature_names_

    # Average the scores from each fold of the cross validation
    # Append the metrics to lists
    knn_k.append(k)
```

```
    knn_r2.append(knn_r2_scores.mean())
    knn_root_mean_squared_error.append(knn_root_mean_squared_error_scores.
 ↪mean())
    knn_mean_absolute_error.append(knn_mean_absolute_error_scores.mean())
    knn_number_of_features.append(knn_k_number_of_features)
    knn_feature_indices.append(knn_k_feature_indices)
    knn_feature_names.append(knn_k_feature_names)
```

[43]:
```
# Dataframe capturing the overall performance metrics of the k-nearest␣
 ↪neighbours model
knn_metrics = pd.DataFrame(
    {
        "Model": "K-Nearest Neighbours",
        "k": knn_k,
        "R Squared": knn_r2,
        "Root Mean Squared Error": knn_root_mean_squared_error,
        "Mean Absolute Error": knn_mean_absolute_error,
        "Number of Features": knn_number_of_features,
        "Feature Indices": knn_feature_indices,
        "Feature Names": knn_feature_names
    }
)
knn_metrics
```

[43]:

	Model	k	R Squared	Root Mean Squared Error	\
0	K-Nearest Neighbours	1	0.219436	421.667416	
1	K-Nearest Neighbours	2	0.327433	378.059800	
2	K-Nearest Neighbours	3	0.381221	364.634444	
3	K-Nearest Neighbours	4	0.401190	358.527755	
4	K-Nearest Neighbours	5	0.413967	354.322939	
5	K-Nearest Neighbours	6	0.426136	351.644301	
6	K-Nearest Neighbours	7	0.430954	349.242086	
7	K-Nearest Neighbours	8	0.430172	349.301746	
8	K-Nearest Neighbours	9	0.428678	349.272094	
9	K-Nearest Neighbours	10	0.429712	348.841335	
10	K-Nearest Neighbours	11	0.430958	348.498114	
11	K-Nearest Neighbours	12	0.431430	348.876643	
12	K-Nearest Neighbours	13	0.428501	349.494830	
13	K-Nearest Neighbours	14	0.427186	349.486821	
14	K-Nearest Neighbours	15	0.425427	350.304684	
15	K-Nearest Neighbours	16	0.424333	350.871335	
16	K-Nearest Neighbours	17	0.421535	351.153687	
17	K-Nearest Neighbours	18	0.420542	351.736602	
18	K-Nearest Neighbours	19	0.420047	352.077889	
19	K-Nearest Neighbours	20	0.419644	352.322494	

Mean Absolute Error	Number of Features	\

```
0          275.031507                 11
1          258.132192                 10
2          250.500495                  8
3          247.725828                  7
4          245.218904                  9
5          243.903063                 10
6          243.780528                  9
7          243.914740                  8
8          244.077220                  8
9          244.519863                  8
10         244.424491                  8
11         244.921299                  8
12         245.473147                  8
13         245.437565                  8
14         246.307549                  8
15         246.826641                  8
16         247.152941                  7
17         247.377194                  9
18         247.762179                  9
19         248.339846                  9

                              Feature Indices  \
0   (0, 1, 6, 7, 8, 9, 10, 13, 14, 16, 17)
1      (0, 1, 5, 6, 7, 8, 10, 13, 14, 17)
2           (0, 1, 5, 7, 10, 14, 16, 17)
3              (0, 1, 5, 7, 10, 14, 17)
4         (0, 1, 5, 7, 8, 10, 14, 16, 17)
5      (0, 1, 5, 6, 7, 8, 10, 14, 16, 17)
6         (0, 1, 5, 7, 8, 10, 14, 16, 17)
7           (0, 1, 5, 7, 10, 14, 16, 17)
8           (0, 1, 5, 7, 10, 14, 16, 17)
9           (0, 1, 5, 7, 10, 14, 16, 17)
10          (0, 1, 5, 7, 10, 14, 16, 17)
11          (0, 1, 5, 7, 10, 14, 16, 17)
12          (0, 1, 5, 7, 10, 14, 16, 17)
13          (0, 1, 5, 7, 10, 14, 16, 17)
14          (0, 1, 5, 7, 10, 14, 16, 17)
15          (0, 1, 5, 7, 10, 14, 16, 17)
16             (0, 1, 5, 7, 10, 14, 17)
17        (0, 1, 5, 6, 7, 10, 14, 16, 17)
18        (0, 1, 5, 6, 7, 10, 14, 16, 17)
19        (0, 1, 5, 6, 7, 10, 14, 16, 17)

                                         Feature Names
0   (Hour, Temperature(°C), Solar Radiation (MJ/m2…
1   (Hour, Temperature(°C), Dew point temperature(…
2   (Hour, Temperature(°C), Dew point temperature(…
```

```
3   (Hour, Temperature(°C), Dew point temperature(…
4   (Hour, Temperature(°C), Dew point temperature(…
5   (Hour, Temperature(°C), Dew point temperature(…
6   (Hour, Temperature(°C), Dew point temperature(…
7   (Hour, Temperature(°C), Dew point temperature(…
8   (Hour, Temperature(°C), Dew point temperature(…
9   (Hour, Temperature(°C), Dew point temperature(…
10  (Hour, Temperature(°C), Dew point temperature(…
11  (Hour, Temperature(°C), Dew point temperature(…
12  (Hour, Temperature(°C), Dew point temperature(…
13  (Hour, Temperature(°C), Dew point temperature(…
14  (Hour, Temperature(°C), Dew point temperature(…
15  (Hour, Temperature(°C), Dew point temperature(…
16  (Hour, Temperature(°C), Dew point temperature(…
17  (Hour, Temperature(°C), Dew point temperature(…
18  (Hour, Temperature(°C), Dew point temperature(…
19  (Hour, Temperature(°C), Dew point temperature(…
```

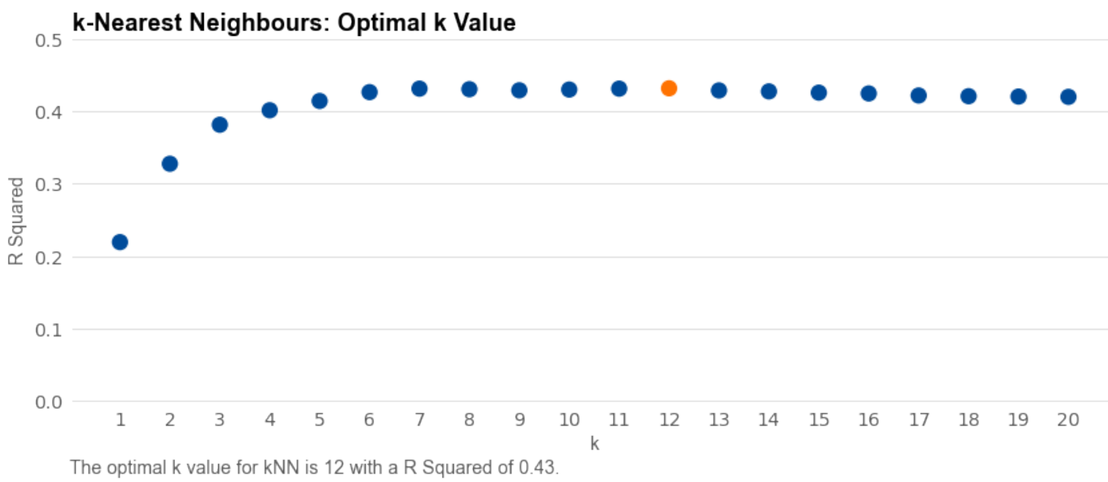
[60]:

```python
# Plot the k values by R squared to visualize the performance of each k-nearest␣
↪neighbours model
value_r2 = knn_metrics["R Squared"] == knn_metrics["R Squared"].max()
knn_metrics["colour_r2"] = np.where(value_r2 == True, "#FF7200", "#004C9B")
knn_plt = sns.regplot(
    data=knn_metrics,
    x="k",
    y="R Squared",
    fit_reg=False,
    scatter_kws={
        "alpha": 1,
        "facecolors": knn_metrics["colour_r2"],
        "linewidths": 0,
        "s": 150,
        "zorder": 10,
    }
)

# Add title
knn_plt.set_title(
    "k-Nearest Neighbours: Optimal k Value",
    font="Arial",
    fontsize="18",
    fontweight="bold",
    loc="left"
)

# X-axis
plt.xlabel(
```

```python
    "k", color="#595959", font="Arial", fontsize="14",␣
 ↪horizontalalignment="center"
)

# Y-axis
plt.ylabel(
    "R Squared",
    color="#595959",
    font="Arial",
    fontsize="14",
    horizontalalignment="center"
)

# Ticks
plt.xticks(range(int(knn_metrics["k"].min()), int(knn_metrics["k"].max()) + 1,␣
 ↪1))
plt.tick_params(colors="#595959", bottom=False, left=False, labelsize="14")

# Add horizontal gridlines
plt.grid(axis="y", color="#D9D9D9")

# Set plot size
knn_plt.figure.set_size_inches(14, 5)

# Spines
sns.despine(left=True)
for _, s in knn_plt.spines.items():
    s.set_color("#D9D9D9")

# Get the current Axes object
ax = plt.gca()

# Set the ylim to begin at 0
ax.set_ylim(top=0.5, bottom=0)

# Get row with max R squared
max_y_row = knn_metrics.loc[knn_metrics["R Squared"].idxmax()]

# Get the max R squared value and the corresponding x value
max_y_value = np.round(max_y_row["R Squared"], 2)
corresponding_x_value = np.round(max_y_row["k"], 2)

# Add a caption
plt.text(
    0,
    -0.1,
```

```
    f"The optimal k value for kNN is {corresponding_x_value} with a R Squared␣
↪of {max_y_value}.",
    color="#595959",
    font="Arial",
    fontsize="14"
)

knn_plt
```

[60]: <AxesSubplot: title={'left': 'k-Nearest Neighbours: Optimal k Value'}, xlabel='k', ylabel='R Squared'>



[61]:
```
# For the k-nearest neighbours models, I will choose the one with the highest R␣
↪squared value
knn_metrics_sorted = knn_metrics.sort_values("R Squared", ascending=False)
knn_metrics_sorted.drop(columns=["k", "colour_r2"], inplace=True)

# Subset the dataframe to keep the record with the highest R Squared
highest_knn_metrics = knn_metrics_sorted.head(1)
highest_knn_metrics
```

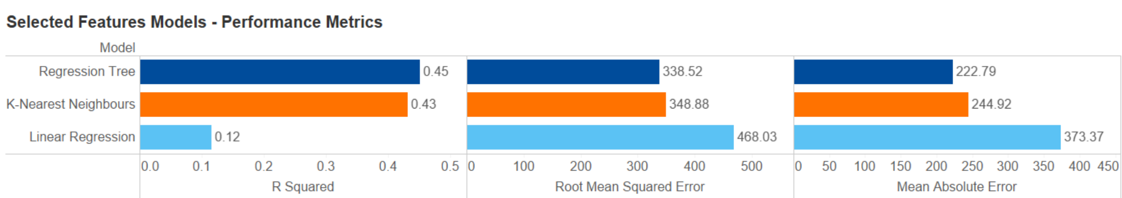
[61]:
```
                     Model  R Squared  Root Mean Squared Error  \
11  K-Nearest Neighbours    0.43143               348.876643

    Mean Absolute Error  Number of Features              Feature Indices  \
11           244.921299                   8  (0, 1, 5, 7, 10, 14, 16, 17)

                                        Feature Names
11  (Hour, Temperature(°C), Dew point temperature(…
```

Performance Metrics The performance metrics of all regression models are displayed below as a pandas dataframe and Tableau visualizations.

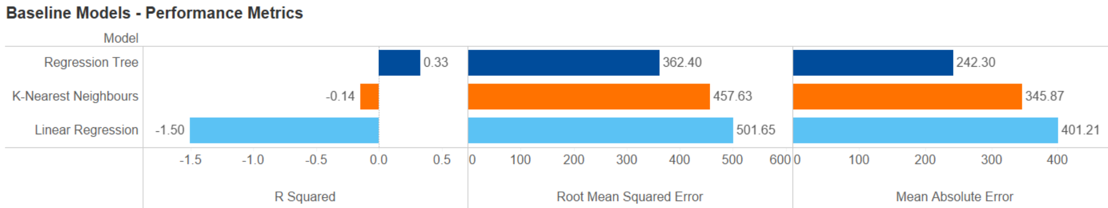
[71]:
```
# Create a selected_features_metrics dataframe to capture the performance␣
↪metrics of all regression models built with selected features
selected_features_metrics = pd.concat([lr_metrics, rt_metrics,␣
↪highest_knn_metrics])
selected_features_metrics.reset_index(drop=True, inplace=True)
selected_features_metrics
```

[71]:
```
                   Model  R Squared  Root Mean Squared Error  \
0      Linear Regression   0.116176               468.029482
1        Regression Tree   0.450626               338.523772
2  K-Nearest Neighbours   0.431430               348.876643

   Mean Absolute Error  Number of Features  \
0           373.366373                   7
1           222.793721                  12
2           244.921299                   8

                          Feature Indices  \
0                 (1, 4, 7, 9, 10, 14, 17)
1  (0, 1, 2, 6, 7, 8, 9, 10, 13, 14, 16, 17)
2             (0, 1, 5, 7, 10, 14, 16, 17)

                                         Feature Names
0  (Temperature(°C), Visibility (10m), Rainfall(m…
1  (Hour, Temperature(°C), Humidity(%), Solar Rad…
2  (Hour, Temperature(°C), Dew point temperature(…
```

[74]:
```
# Export the selected_features_metrics dataframe to a .csv flat file
selected_features_metrics.to_csv("selected_features_metrics.csv", index=False,␣
↪encoding="utf-8-sig")
```



Based on the performance metrics of each model, the regression tree model is the best performing, followed by k-nearest neighbours in second place, and linear regression being last. Notably, the R Squared for regression tree and k-nearest neighbours are quite close in value whereas the linear regression model currently has suboptimal performance. As per the sklearn module, the best score for the R squared is 1.0. This means that the performance of the regression tree and k-nearest neighbours is approaching the halfway mark. In terms of the error metrics, root mean squared error and mean absolute error, the lower the error values the better performing the models are.



By comparing the 2 graphs, "Selected Features Models - Performance Metrics" and "Baseline Models - Performance Metrics", we can see that using selected features in our models has improved the R squared of regression tree by approximately 0.12. In particular, for both k-nearest neighbours and linear regression, using selected features in our models has significantly improved the R squared from negative arbitrary values (very poor performance) to tangible R squared values of 0.43 and 0.12 respectively.

Therefore, by comparing the performance of the selected features models (as seen in the graph "Selected Features Models - Performance Metrics") and the baseline models (as seen in the graph "Baseline Models - Performance Metrics"), it can be concluded that using a feature selection method, namely Sequential Forward Selection from the mlxtend module has optimized the performance of all regression models.