Seoul National University

SEOUL SHARED BIKE CLASSIFICATION

Machine Learning

2024-81602 Hsu Fang Yu

Project-II

Step 0 Introduce to the dataset

Seoul Bike Sharing Demand Dataset

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
01/12/2017	254	0	-5.2	37	2.2	2000	-17.6	0	0	0	Winter	No Holiday	Yes
01/12/2017	204	1	-5.5	38	0.8	2000	-17.6	0	0	0	Winter	No Holiday	Yes
01/12/2017	173	2	-6	39	1	2000	-17.7	0	0	0	Winter	No Holiday	Yes
01/12/2017	107	3	-6.2	40	0.9	2000	-17.6	0	0	0	Winter	No Holiday	Yes
01/12/2017	78	4	-6	36	2.3	2000	-18.6	0	0	0	Winter	No Holiday	Yes
01/12/2017	100	5	-6.4	37	1.5	2000	-18.7	0	0	0	Winter	No Holiday	Yes
01/12/2017	181	6	-6.6	35	1.3	2000	-19.5	0	0	0	Winter	No Holiday	Yes
01/12/2017	460	7	-7.4	38	0.9	2000	-19.3	0	0	0	Winter	No Holiday	Yes
01/12/2017	930	8	-7.6	37	1.1	2000	-19.8	0.01	0	0	Winter	No Holiday	Yes
01/12/2017	490	9	-6.5	27	0.5	1928	-22.4	0.23	0	0	Winter	No Holiday	Yes
01/12/2017	339	10	-3.5	24	1.2	1996	-21.2	0.65	0	0	Winter	No Holiday	Yes
01/12/2017	360	11	-0.5	21	1.3	1936	-20.2	0.94	0	0	Winter	No Holiday	Yes
01/12/2017	449	12	1.7	23	1.4	2000	-17.2	1.11	0	0	Winter	No Holiday	Yes
01/12/2017	451	13	2.4	25	1.6	2000	-15.6	1.16	0	0	Winter	No Holiday	Yes
01/12/2017	447	14	3	26	2	2000	-14.6	1.01	0	0	Winter	No Holiday	Yes
01/12/2017	463	15	2.1	36	3.2	2000	-11.4	0.54	0	0	Winter	No Holiday	Yes
01/12/2017	484	16	1.2	54	4.2	793	-7	0.24	0	0	Winter	No Holiday	Yes
01/12/2017	555	17	0.8	58	1.6	2000	-6.5	0.08	0	0	Winter	No Holiday	Yes
01/12/2017	862	18	0.6	66	1.4	2000	-5	0	0	0	Winter	No Holiday	Yes
01/12/2017	600	19	0	77	1.7	2000	-3.5	0	0	0	Winter	No Holiday	Yes
01/12/2017	426	20	-0.3	79	1.5	1913	-3.5	0	0	0	Winter	No Holiday	Yes
01/12/2017	405	21	-0.8	81	0.8	1687	-3.6	0	0	0	Winter	No Holiday	Yes
01/12/2017	398	22	-0.9	83	1.5	1380	-3.4	0	0	0	Winter	No Holiday	Yes
01/12/2017	323	23	-1.3	84	1	1265	-3.6	0	0	0	Winter	No Holiday	Yes
02/12/2017	328	0	-1.8	87	1.1	994	-3.6	0	0	0	Winter	No Holiday	Yes
02/12/2017	308	1	-2.2	86	0.6	990	-4.2	0	0	0	Winter	No Holiday	Yes
02/12/2017	262	2	-2.9	86	1.5	1256	-4.9	0	0	0	Winter	No Holiday	Yes

Source: UCI Machine Learning Repository
(https://archive.ics.uci.edu/dataset/560/seoul+bike+sharing+demand)

Variables: 14

- Date : year-month-day
- Rented Bike count - Count of bikes rented at each hour (**Targeted Variable**)
- Hour - Hour of he day
- Temperature-Temperature in Celsius
- Humidity - %
- Windspeed - m/s
- Visibility - 10m
- Dew point temperature - Celsius
- Solar radiation - MJ/m2
- Rainfall - mm
- Snowfall - cm
- Seasons - Winter, Spring, Summer, Autumn
- Holiday - Holiday/No holiday
- Functional Day - NoFunc(Non Functional Hours), Fun(Functional hours)

Instance: 8760

Description:
Currently, rental bikes have been introduced in many urban cities to enhance mobility comfort. Ensuring that rental bikes are available and accessible to the public at the right time is crucial, as it reduces waiting times. Consequently, maintaining a stable supply of rental bikes throughout the city has become a

significant concern. A critical aspect of this is predicting the number of bikes needed each hour to ensure a consistent supply.

The resaon why I choose this dataset as my term project is because being an exchange student to Korea, I want to do some projects related to Korean social issues, and then I found something in common with my home country, Taiwan. In Taiwan, we have a shard bike network called Ubike, which is so ubiquitous that people take advantage of it almost every day. But sometimes, I cannot find even one Ubike's bicycle on all surrounding stops. It indeicates that perhaps the government does not accurately estimate the demand of shared bikes and assign them to each stop accordingly. Being inspired by the similar problem in Taiwan, I would like to perform classification using Seoul Bike Sharing Dataset and try to address this issue by logistic regression we learned during classes.

*Note:

1. Although the targeted variable, Rented Bike count, is numerial, we still can perform classification by manipulating the data to transform it into a categorical variable. I will show how to do so in the step 2.
2. In order to maintain the integrity of the dataset, we do not reduce any data point and keep the original 8760 ones.

Step 1 Load the Dataset

Code:

```
import pandas as pd

df = pd.read_csv("/Users/hsufangyu/Desktop/SNU/ML Term Project/Project 2/SeoulBikeData.csv", encoding='latin1')
df.head()
```

Result:

	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
0	01/12/2017	254	0	-5.2	37	2.2	2000	-17.6	0.0	0.0	0.0	Winter	No Holiday	Yes
1	01/12/2017	204	1	-5.5	38	0.8	2000	-17.6	0.0	0.0	0.0	Winter	No Holiday	Yes
2	01/12/2017	173	2	-6.0	39	1.0	2000	-17.7	0.0	0.0	0.0	Winter	No Holiday	Yes
3	01/12/2017	107	3	-6.2	40	0.9	2000	-17.6	0.0	0.0	0.0	Winter	No Holiday	Yes
4	01/12/2017	78	4	-6.0	36	2.3	2000	-18.6	0.0	0.0	0.0	Winter	No Holiday	Yes

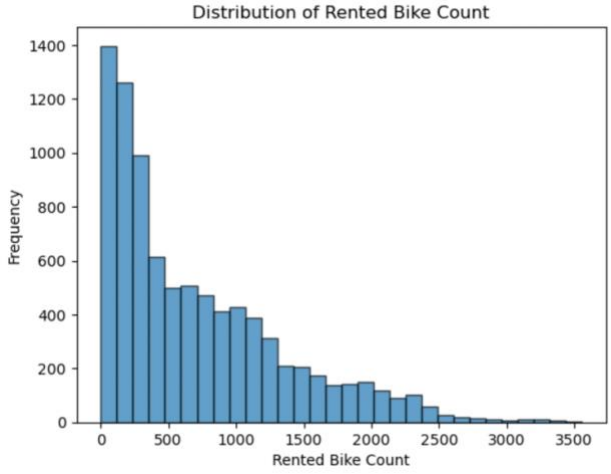
Explanation:
Let's load the dataset and display it. The dataset provides various features related to weather conditions, time, and bike rental counts, which will be used for building and evaluating logistic regression models in subsequent steps.

Step 2 Preprocess the Data for Logistic Regression

Code:

```python
import matplotlib.pyplot as plt

plt.hist(df['Rented Bike Count'], bins=30, edgecolor='k', alpha=0.7)
plt.xlabel('Rented Bike Count')
plt.ylabel('Frequency')
plt.title('Distribution of Rented Bike Count')
plt.show()

percentile_75 = df['Rented Bike Count'].quantile(0.75)
df['Demand'] = df['Rented Bike Count'].apply(lambda x: 1 if x > percentile_75 else 0)

df = df.drop('Date', axis=1)

categorical_columns = df.select_dtypes(include=['object']).columns
df = pd.get_dummies(df, columns=categorical_columns, drop_first=True)

df.head()
```

Result:



	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)	Demand	Seasons_Spring	Seasons_Summer	Seasons_Winter	Holiday_No Holiday	Functioning Day_Yes
0	254	0	-5.2	37	2.2	2000	-17.6	0.0	0.0	0.0	0	0	0	1	1	1
1	204	1	-5.5	38	0.8	2000	-17.6	0.0	0.0	0.0	0	0	0	1	1	1
2	173	2	-6.0	39	1.0	2000	-17.7	0.0	0.0	0.0	0	0	0	1	1	1
3	107	3	-6.2	40	0.9	2000	-17.6	0.0	0.0	0.0	0	0	0	1	1	1
4	78	4	-6.0	36	2.3	2000	-18.6	0.0	0.0	0.0	0	0	0	1	1	1

Explanation:

Here are some issues in the dataset, so we address them first.

1. Transforming “Rented Bike Count” into a catergorical variable: Our targeted value in this dataset is “Rented Bike Count”. However, it is numerial, which cannot perform classification. Therefore, we plot the distribution of its data and decide to select the 75th percentile as the boundary to distinguish “Hign Demand” and “Low Demand” values. In other words, if the value of data below the 75th percentile, we recognize it as “Low Demand”, and vice versa. By this method, we create a new column called “Demand” as our new targeted value.
2. Irrelevant “Date” Column: “Date” column seems to be irrelevant to predict the demand of shared bike, so we remove it from the dataset.
3. Handling Categorical Data: There are some categorical variables, like “Seansons” and “Holiday”. We converts them into dummy variables and drops the first category to avoid multicollinearity.

Now, we have finished the data preprocessing step. We can utilize our new dataset to perform the binary classification using logistic regression.

*The reason why we choose the 75th percentile as the boundary is because the value of “Rented Bike Count” concentrates on “the left”. Given the skewed distribution, this boundary ensures that a significant portion of data is considered "low demand," while the top 25% of the highest values are marked as "high demand." This creates a more balanced classification, especially in a dataset where high counts are less frequent.

Step 3 Split the Data into Training, Cross-Validation, and Test Sets

Code:

```
from sklearn.model_selection import train_test_split
import numpy as np

def calculate_cov(dataframe):
    return dataframe.std() / dataframe.mean()

features = df.drop(['Rented Bike Count', 'Demand'], axis=1)
target = df['Demand']

X_train, X_temp, y_train, y_temp = train_test_split(features, target, test_size=0.4, random_state=42)
X_cv, X_test, y_cv, y_test = train_test_split(X_temp, y_temp, test_size=0.5, random_state=42)

train_stats = X_train.describe().T
cv_stats = X_cv.describe().T
test_stats = X_test.describe().T
df_stats = df.describe().T

train_cov = calculate_cov(X_train)
cv_cov = calculate_cov(X_cv)
test_cov = calculate_cov(X_test)
df_cov = calculate_cov(df.drop(['Rented Bike Count', 'Demand'], axis=1))

train_corr = X_train.corr()
cv_corr = X_cv.corr()
test_corr = X_test.corr()
df_corr = df.drop(['Rented Bike Count', 'Demand'], axis=1).corr()

print("Training set statistics:\n", train_stats)
print("\nCross-validation set statistics:\n", cv_stats)
print("\nTest set statistics:\n", test_stats)
print("\nEntire set statistics:\n", df_stats)
print("Training set COV:\n", train_cov)
print("\nCross-validation set COV:\n", cv_cov)
print("\nTest set COV:\n", test_cov)
print("\nEntire set COV:\n", df_cov)

print("\nTraining set Correlation Coefficients:\n", train_corr)
print("\nCross-validation set Correlation Coefficients:\n", cv_corr)
print("\nTest set Correlation Coefficients:\n", test_corr)
print("\nEntire set Correlation Coefficients:\n", df_corr)
```

Result:

train_stats

	count	mean	std	min	25%	50%	75%	max
Hour	5256.0	11.522260	6.865257	0.0	6.000	12.00	17.0000	23.00
Temperature(°C)	5256.0	12.939193	11.902740	-17.5	3.675	13.75	22.5250	39.00
Humidity(%)	5256.0	58.051750	20.409313	0.0	42.000	57.00	74.0000	98.00
Wind speed (m/s)	5256.0	1.735864	1.037008	0.0	1.000	1.50	2.3000	7.40
Visibility (10m)	5256.0	1438.497146	611.550354	33.0	940.750	1707.00	2000.0000	2000.00
Dew point temperature(°C)	5256.0	4.074182	12.933099	-30.6	-4.400	5.00	14.6000	26.80
Solar Radiation (MJ/m2)	5256.0	0.587327	0.881983	0.0	0.000	0.01	0.9725	3.52
Rainfall(mm)	5256.0	0.145186	1.034790	0.0	0.000	0.00	0.0000	24.00
Snowfall (cm)	5256.0	0.072374	0.442347	0.0	0.000	0.00	0.0000	8.80
Seasons_Spring	5256.0	0.256849	0.436937	0.0	0.000	0.00	1.0000	1.00
Seasons_Summer	5256.0	0.245624	0.430498	0.0	0.000	0.00	0.0000	1.00
Seasons_Winter	5256.0	0.241058	0.427766	0.0	0.000	0.00	0.0000	1.00
Holiday_No Holiday	5256.0	0.949772	0.218437	0.0	1.000	1.00	1.0000	1.00
Functioning Day_Yes	5256.0	0.964612	0.184776	0.0	1.000	1.00	1.0000	1.00

cv_stats

	count	mean	std	min	25%	50%	75%	max
Hour	1752.0	11.269977	6.985720	0.0	5.000	11.00	17.00	23.00
Temperature(°C)	1752.0	12.812043	12.072478	-16.5	3.100	13.50	22.50	39.40
Humidity(%)	1752.0	58.106735	20.091667	0.0	42.000	57.00	73.00	98.00
Wind speed (m/s)	1752.0	1.736701	1.062386	0.0	0.900	1.50	2.40	5.40
Visibility (10m)	1752.0	1434.138128	603.332145	70.0	946.000	1670.50	2000.00	2000.00
Dew point temperature(°C)	1752.0	3.982078	13.288767	-29.7	-5.525	5.20	15.00	27.20
Solar Radiation (MJ/m2)	1752.0	0.553071	0.857475	0.0	0.000	0.01	0.88	3.42
Rainfall(mm)	1752.0	0.138870	1.180893	0.0	0.000	0.00	0.00	35.00
Snowfall (cm)	1752.0	0.079566	0.437458	0.0	0.000	0.00	0.00	5.00
Seasons_Spring	1752.0	0.244292	0.429789	0.0	0.000	0.00	0.00	1.00
Seasons_Summer	1752.0	0.260274	0.438909	0.0	0.000	0.00	1.00	1.00
Seasons_Winter	1752.0	0.252283	0.434447	0.0	0.000	0.00	1.00	1.00
Holiday_No Holiday	1752.0	0.950913	0.216111	0.0	1.000	1.00	1.00	1.00
Functioning Day_Yes	1752.0	0.966324	0.180445	0.0	1.000	1.00	1.00	1.00

test_stats

	count	mean	std	min	25%	50%	75%	max
Hour	1752.0	11.663242	7.027778	0.0	5.00	11.50	18.000	23.00
Temperature(°C)	1752.0	12.784989	11.948382	-17.8	3.30	13.90	22.425	39.30
Humidity(%)	1752.0	58.869292	20.488928	0.0	43.00	59.00	75.000	98.00
Wind speed (m/s)	1752.0	1.680251	1.006727	0.0	0.90	1.50	2.300	5.80
Visibility (10m)	1752.0	1434.499429	603.773567	27.0	928.75	1685.50	2000.000	2000.00
Dew point temperature(°C)	1752.0	4.164441	13.215964	-30.5	-4.90	5.30	15.200	26.10
Solar Radiation (MJ/m2)	1752.0	0.530502	0.838298	0.0	0.00	0.01	0.820	3.52
Rainfall(mm)	1752.0	0.169007	1.326031	0.0	0.00	0.00	0.000	29.50
Snowfall (cm)	1752.0	0.078653	0.418942	0.0	0.00	0.00	0.000	5.10
Seasons_Spring	1752.0	0.245434	0.430467	0.0	0.00	0.00	0.000	1.00
Seasons_Summer	1752.0	0.263128	0.440457	0.0	0.00	0.00	1.000	1.00
Seasons_Winter	1752.0	0.257420	0.437338	0.0	0.00	0.00	1.000	1.00
Holiday_No Holiday	1752.0	0.953196	0.211278	0.0	1.00	1.00	1.000	1.00
Functioning Day_Yes	1752.0	0.971461	0.166554	0.0	1.00	1.00	1.000	1.00

df_stats

	count	mean	std	min	25%	50%	75%	max
Rented Bike Count	8760.0	704.602055	644.997468	0.0	191.00	504.50	1065.25	3556.00
Hour	8760.0	11.500000	6.922582	0.0	5.75	11.50	17.25	23.00
Temperature(°C)	8760.0	12.882922	11.944825	-17.8	3.50	13.70	22.50	39.40
Humidity(%)	8760.0	58.226256	20.362413	0.0	42.00	57.00	74.00	98.00
Wind speed (m/s)	8760.0	1.724909	1.036300	0.0	0.90	1.50	2.30	7.40
Visibility (10m)	8760.0	1436.825799	608.298712	27.0	940.00	1698.00	2000.00	2000.00
Dew point temperature(°C)	8760.0	4.073813	13.060369	-30.6	-4.70	5.10	14.80	27.20
Solar Radiation (MJ/m2)	8760.0	0.569111	0.868746	0.0	0.00	0.01	0.93	3.52
Rainfall(mm)	8760.0	0.148687	1.128193	0.0	0.00	0.00	0.00	35.00
Snowfall (cm)	8760.0	0.075068	0.436746	0.0	0.00	0.00	0.00	8.80
Demand	8760.0	0.250000	0.433037	0.0	0.00	0.00	0.25	1.00
Seasons_Spring	8760.0	0.252055	0.434217	0.0	0.00	0.00	1.00	1.00
Seasons_Summer	8760.0	0.252055	0.434217	0.0	0.00	0.00	1.00	1.00
Seasons_Winter	8760.0	0.246575	0.431042	0.0	0.00	0.00	0.00	1.00
Holiday_No Holiday	8760.0	0.950685	0.216537	0.0	1.00	1.00	1.00	1.00
Functioning Day_Yes	8760.0	0.966324	0.180404	0.0	1.00	1.00	1.00	1.00

Coefficient of Variation

```
Training set COV:
 Hour                          0.595826
Temperature(°C)                0.919898
Humidity(%)                    0.351571
Wind speed (m/s)               0.597402
Visibility (10m)               0.425131
Dew point temperature(°C)      3.174404
Solar Radiation (MJ/m2)        1.501690
Rainfall(mm)                   7.127315
Snowfall (cm)                  6.111919
Seasons_Spring                 1.701142
Seasons_Summer                 1.752669
Seasons_Winter                 1.774537
Holiday_No Holiday             0.229988
Functioning Day_Yes            0.191555
dtype: float64
```

```
Cross-validation set COV:
 Hour                          0.619852
Temperature(°C)                0.942276
Humidity(%)                    0.345772
Wind speed (m/s)               0.611726
Visibility (10m)               0.420693
Dew point temperature(°C)      3.337144
Solar Radiation (MJ/m2)        1.550390
Rainfall(mm)                   8.503593
Snowfall (cm)                  5.498041
Seasons_Spring                 1.759325
Seasons_Summer                 1.686336
Seasons_Winter                 1.722061
Holiday_No Holiday             0.227267
Functioning Day_Yes            0.186733
dtype: float64
```

```
Test set COV:
 Hour                          0.602558
Temperature(°C)                0.934563
Humidity(%)                    0.348041
Wind speed (m/s)               0.599153
Visibility (10m)               0.420895
Dew point temperature(°C)      3.173527
Solar Radiation (MJ/m2)        1.580198
Rainfall(mm)                   7.846022
Snowfall (cm)                  5.326456
Seasons_Spring                 1.753903
Seasons_Summer                 1.673927
Seasons_Winter                 1.698926
Holiday_No Holiday             0.221653
Functioning Day_Yes            0.171447
dtype: float64
```

```
Entire set COV:
 Hour                          0.601964
Temperature(°C)                0.927183
Humidity(%)                    0.349712
Wind speed (m/s)               0.600785
Visibility (10m)               0.423363
Dew point temperature(°C)      3.205933
Solar Radiation (MJ/m2)        1.526498
Rainfall(mm)                   7.587693
Snowfall (cm)                  5.817969
Seasons_Spring                 1.722710
Seasons_Summer                 1.722710
Seasons_Winter                 1.748115
Holiday_No Holiday             0.227770
Functioning Day_Yes            0.186691
dtype: float64
```

Correlation Coefficients (we just partially show the table. The full ones can be found in the Python file.)

	Hour	Temperature(°C)	Humidity(%)	Wind speed (m/s)	Visibility (10m)	Dew point temperature(°C)	Solar Radiation (MJ/m2)	Rainfall(mm)	Snowfall (cm)
Hour	1.000000	0.138584	-0.240358	0.283530	0.111588	0.015823	0.148246	0.001850	-0.022011
Temperature(°C)	0.138584	1.000000	0.139966	-0.029973	0.044212	0.910493	0.355516	0.050052	-0.208991
Humidity(%)	-0.240358	0.139966	1.000000	-0.329675	-0.549582	0.524499	-0.468284	0.250928	0.114631
Wind speed (m/s)	0.283530	-0.029973	-0.329675	1.000000	0.167497	-0.169811	0.324999	-0.040200	0.009449
Visibility (10m)	0.111588	0.044212	-0.549582	0.167497	1.000000	-0.172727	0.149538	-0.190118	-0.123950
Dew point temperature(°C)	0.015823	0.910493	0.524499	-0.169811	-0.172727	1.000000	0.092940	0.131920	-0.141675
Solar Radiation (MJ/m2)	0.148246	0.355516	-0.468284	0.324999	0.149538	0.092940	1.000000	-0.079907	-0.069752
Rainfall(mm)	0.001850	0.050052	0.250928	-0.040200	-0.190118	0.131920	-0.079907	1.000000	0.027347
Snowfall (cm)	-0.022011	-0.208991	0.114631	0.009449	-0.123950	-0.141675	-0.069752	0.027347	1.000000

	Hour	Temperature(°C)	Humidity(%)	Wind speed (m/s)	Visibility (10m)	Dew point temperature(°C)	Solar Radiation (MJ/m2)	Rainfall(mm)	Snowfall (cm)
Hour	1.000000	0.105789	-0.268271	0.296207	0.069999	-0.021350	0.167573	0.016256	-0.029814
Temperature(°C)	0.105789	1.000000	0.189681	-0.026856	0.012228	0.917130	0.368781	0.048629	-0.232867
Humidity(%)	-0.268271	0.189681	1.000000	-0.340465	-0.529025	0.555439	-0.440746	0.214581	0.085550
Wind speed (m/s)	0.296207	-0.026856	-0.340465	1.000000	0.169618	-0.165680	0.346265	-0.011799	-0.025223
Visibility (10m)	0.069999	0.012228	-0.529025	0.169618	1.000000	-0.186932	0.136276	-0.161941	-0.099010
Dew point temperature(°C)	-0.021350	0.917130	0.555439	-0.165680	-0.186932	1.000000	0.120095	0.114130	-0.169472
Solar Radiation (MJ/m2)	0.167573	0.368781	-0.440746	0.346265	0.136276	0.120095	1.000000	-0.064647	-0.071608
Rainfall(mm)	0.016256	0.048629	0.214581	-0.011799	-0.161941	0.114130	-0.064647	1.000000	-0.012833
Snowfall (cm)	-0.029814	-0.232867	0.085550	-0.025223	-0.099010	-0.169472	-0.071608	-0.012833	1.000000

	Hour	Temperature(°C)	Humidity(%)	Wind speed (m/s)	Visibility (10m)	Dew point temperature(°C)	Solar Radiation (MJ/m2)	Rainfall(mm)	Snowfall (cm)
Hour	1.000000	0.100344	-0.220740	0.280919	0.089274	-0.009570	0.113683	0.017338	-0.011230
Temperature(°C)	0.100344	1.000000	0.187641	-0.066122	0.028980	0.915412	0.331818	0.053523	-0.232984
Humidity(%)	-0.220740	0.187641	1.000000	-0.353691	-0.537471	0.555346	-0.462932	0.226516	0.110879
Wind speed (m/s)	0.280919	-0.066122	-0.353691	1.000000	0.186203	-0.208158	0.339739	0.022728	-0.021664
Visibility (10m)	0.089274	0.028980	-0.537471	0.186203	1.000000	-0.178055	0.164068	-0.123623	-0.138100
Dew point temperature(°C)	-0.009570	0.915412	0.555346	-0.208158	-0.178055	1.000000	0.072992	0.123546	-0.160306
Solar Radiation (MJ/m2)	0.113683	0.331818	-0.462932	0.339739	0.164068	0.072992	1.000000	-0.071141	-0.080684
Rainfall(mm)	0.017338	0.053523	0.226516	0.022728	-0.123623	0.123546	-0.071141	1.000000	-0.018082
Snowfall (cm)	-0.011230	-0.232984	0.110879	-0.021664	-0.138100	-0.160306	-0.080684	-0.018082	1.000000

	Hour	Temperature(°C)	Humidity(%)	Wind speed (m/s)	Visibility (10m)	Dew point temperature(°C)	Solar Radiation (MJ/m2)	Rainfall(mm)	Snowfall (cm)
Hour	1.000000e+00	0.124114	-0.241644	0.285197	0.098753	0.003054	0.145131	0.008715	-0.021516
Temperature(°C)	1.241145e-01	1.000000	0.159371	-0.036252	0.034794	0.912798	0.353505	0.050282	-0.218405
Humidity(%)	-2.416438e-01	0.159371	1.000000	-0.336683	-0.543090	0.536894	-0.461919	0.236397	0.108183
Wind speed (m/s)	2.851967e-01	-0.036252	-0.336683	1.000000	0.171507	-0.176486	0.332274	-0.019674	-0.003554
Visibility (10m)	9.875348e-02	0.034794	-0.543090	0.171507	1.000000	-0.176630	0.149738	-0.167629	-0.121695
Dew point temperature(°C)	3.054372e-03	0.912798	0.536894	-0.176486	-0.176630	1.000000	0.094381	0.125597	-0.150887
Solar Radiation (MJ/m2)	1.451309e-01	0.353505	-0.461919	0.332274	0.149738	0.094381	1.000000	-0.074290	-0.072301
Rainfall(mm)	8.714642e-03	0.050282	0.236397	-0.019674	-0.167629	0.125597	-0.074290	1.000000	0.008500
Snowfall (cm)	-2.151645e-02	-0.218405	0.108183	-0.003554	-0.121695	-0.150887	-0.072301	0.008500	1.000000

Explanation:

- Training Set: Contains 60% of the data.
- Cross-Validation Set: Contains 20% of the data, used to tune model hyperparameters.
- Test Set: Contains 20% of the data, used to evaluate the final model performance.

Comparison of Statistics:

The consistency in statistics across the training, cross-validation, and test sets suggests that the data was split correctly, maintaining the overall distribution and characteristics of the dataset in each subset. This ensures reliable model training, validation, and testing, leading to robust model performance evaluation.

Step 4 Implement and Evaluate Logistic Regression

Code:

```
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_cv_scaled = scaler.transform(X_cv)

log_reg = LogisticRegression(max_iter=1000)
log_reg.fit(X_train_scaled, y_train)
y_cv_pred_scaled = log_reg.predict(X_cv_scaled)

accuracy_scaled = accuracy_score(y_cv, y_cv_pred_scaled)
precision_scaled = precision_score(y_cv, y_cv_pred_scaled, average='weighted')
recall_scaled = recall_score(y_cv, y_cv_pred_scaled, average='weighted')
f1_scaled = f1_score(y_cv, y_cv_pred_scaled, average='weighted')

print(f"Accuracy with Standardized Features: {accuracy_scaled}")
print(f"Precision with Standardized Features: {precision_scaled}")
print(f"Recall with Standardized Features: {recall_scaled}")
print(f"F1-Score with Standardized Features: {f1_scaled}")
```

Result:

```
Accuracy with Standardized Features: 0.8527397260273972
Precision with Standardized Features: 0.8498858615181464
Recall with Standardized Features: 0.8527397260273972
F1-Score with Standardized Features: 0.8510463749576349
```

Explanation:

- Accuracy: 0.8527 (85.27%) is a strong indication that the logistic regression model performs well on the cross-validation set, correctly predicting the majority of instances.
- Precision: 0.8499 (84.99%) precision indicates that when the model predicts high demand, it is correct 84.99% of the time. This shows the model's effectiveness in minimizing false positives.
- Recall: 0.8527 (85.27%) recall suggests that the model correctly identifies 85.27% of all actual high-demand instances. This shows the model's ability to minimize false negatives, meaning it effectively captures most of the high-demand cases.
- F1-Score: 0.8510 (85.10%) F1-score indicates a balanced performance of the model in terms of precision and recall, showing that it effectively balances the trade-offs between these two metrics.

The logistic regression model demonstrates strong performance across all four metrics: accuracy, precision, recall, and F1-score. This indicates that the model is effective at both correctly identifying high-demand instances and minimizing incorrect predictions, providing a robust and reliable binary classification for the given dataset.

*Why we scale the data on this step? Since the model fails to converge due to the maximum number of iterations being reached. We solve this problem by increasing the number of iterations and scaling the data to help in achieving better and faster convergence.

Step 5 Polynomial Features Regularization

Code:

```
from sklearn.preprocessing import PolynomialFeatures

poly = PolynomialFeatures(degree=2, include_bias=False)
X_train_poly = poly.fit_transform(X_train_scaled)
X_cv_poly = poly.transform(X_cv_scaled)

log_reg_poly = LogisticRegression(max_iter=1000)
log_reg_poly.fit(X_train_poly, y_train)
y_cv_pred_poly = log_reg_poly.predict(X_cv_poly)

accuracy_poly = accuracy_score(y_cv, y_cv_pred_poly)
precision_poly = precision_score(y_cv, y_cv_pred_poly, average='weighted')
recall_poly = recall_score(y_cv, y_cv_pred_poly, average='weighted')
f1_poly = f1_score(y_cv, y_cv_pred_poly, average='weighted')

print(f"Accuracy with Polynomial Features: {accuracy_poly}")
print(f"Precision with Polynomial Features: {precision_poly}")
print(f"Recall with Polynomial Features: {recall_poly}")
print(f"F1-Score with Polynomial Features: {f1_poly}")
```

Result:

```
Accuracy with Polynomial Features: 0.8984018264840182
Precision with Polynomial Features: 0.8959340876278529
Recall with Polynomial Features: 0.8984018264840182
F1-Score with Polynomial Features: 0.8959362716077307
```

Explanation:

- Accuracy: The incorporation of polynomial features has significantly improved the model's accuracy from 85.27% to 89.84%. This suggests that the model benefits from the additional complexity introduced by the polynomial terms, capturing more intricate relationships within the data.
- Precision: The precision has improved from 84.99% to 89.59%. This indicates that the model with polynomial features is better at minimizing false positives, making more reliable positive predictions.
- Recall: The recall has improved from 85.27% to 89.84%. This shows that the model with polynomial features is more effective at capturing the actual positive instances, reducing the number of false negatives.
- F1-Score: The F1-score has improved from 85.10% to 89.59%. This confirms that the model with polynomial features has achieved a better balance between precision and recall, resulting in a more reliable classifier.

The incorporation of polynomial features into the logistic regression model has significantly improved its performance across all four metrics: accuracy, precision, recall, and F1-score. This enhancement indicates that the polynomial terms help the model capture more complex patterns and interactions within the data, leading to more accurate and reliable predictions.

Step 6 L1 and L2 Regularization

Code:

```
log_reg_l1 = LogisticRegression(penalty='l1', solver='liblinear', max_iter=1000)
log_reg_l1.fit(X_train_scaled, y_train)
y_cv_pred_l1 = log_reg_l1.predict(X_cv_scaled)

log_reg_l2 = LogisticRegression(penalty='l2', max_iter=1000)
log_reg_l2.fit(X_train_scaled, y_train)
y_cv_pred_l2 = log_reg_l2.predict(X_cv_scaled)

metrics = {
    'L1': (y_cv_pred_l1, 'L1 Regularization'),
    'L2': (y_cv_pred_l2, 'L2 Regularization')
}

for key, (pred, name) in metrics.items():
    accuracy = accuracy_score(y_cv, pred)
    precision = precision_score(y_cv, pred, average='weighted')
    recall = recall_score(y_cv, pred, average='weighted')
    f1 = f1_score(y_cv, pred, average='weighted')
    print(f"\n{name}:")
    print(f"Accuracy: {accuracy}")
    print(f"Precision: {precision}")
    print(f"Recall: {recall}")
    print(f"F1-Score: {f1}")
```

Result:

```
L1 Regularization:
Accuracy: 0.8538812785388128
Precision: 0.8512323593250583
Recall: 0.8538812785388128
F1-Score: 0.8523270272238344

L2 Regularization:
Accuracy: 0.8527397260273972
Precision: 0.8498858615181464
Recall: 0.8527397260273972
F1-Score: 0.8510463749576349
```

Explanation:

Model	Accuracy	Precision	Recall	F1-Score
Original Logistic Regression	0.8527	0.8499	0.8527	0.8510
Polynomial Features (Degree 2)	0.8984	0.8959	0.8984	0.8959
L1 Regularization	0.8539	0.8512	0.8539	0.8523
L2 Regularization	0.8527	0.8499	0.8527	0.8510

- Polynomial Features Regularization: This model performs the best across all metrics (accuracy, precision, recall, F1-score), demonstrating the significant benefit of capturing more complex relationships in the data through polynomial terms.
- L1 Regularization: Shows slight improvements over the original logistic regression model in all metrics, likely due to its ability to produce sparse models and reduce overfitting.
- L2 Regularization: Performs similarly to the original logistic regression model, indicating that it does not significantly change the model performance for this dataset.

In conclusion, while L1 and L2 regularization provide slight improvements or maintain the performance of the original logistic regression model, incorporating polynomial features offers the most substantial enhancement in model performance.

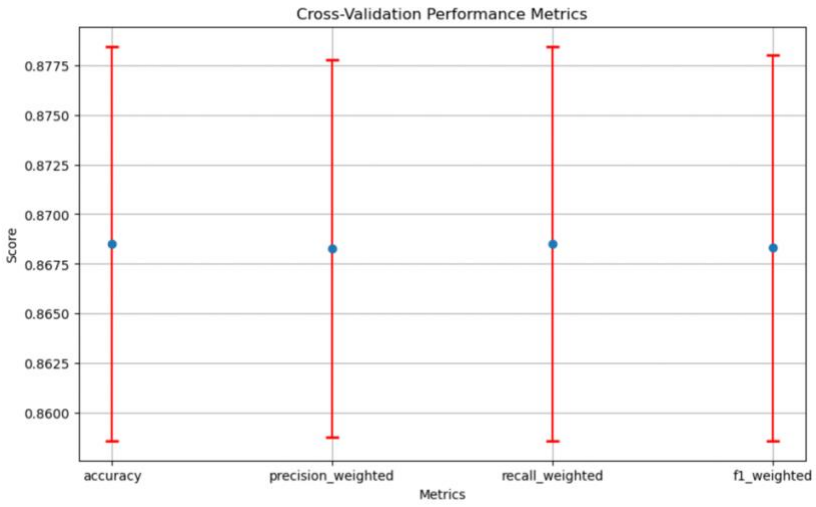
Step 7 Employ Cross-Validation Techniques to Evaluate the Generalization Performance

Code:

```
from sklearn.model_selection import cross_val_score

def evaluate_model(model, X, y):
    scoring = ['accuracy', 'precision_weighted', 'recall_weighted', 'f1_weighted']
    results = {}

    for metric in scoring:
        scores = cross_val_score(model, X, y, cv=5, scoring=metric)
        results[metric] = {
            'mean': scores.mean(),
            'std': scores.std()
        }

    return results

cv_results = evaluate_model(log_reg, X_train_scaled, y_train)

for metric, scores in cv_results.items():
    print(f"{metric.capitalize()} - Mean: {scores['mean']:.4f}, Std: {scores['std']:.4f}")

metrics = ['accuracy', 'precision_weighted', 'recall_weighted', 'f1_weighted']
means = [cv_results[metric]['mean'] for metric in metrics]
stds = [cv_results[metric]['std'] for metric in metrics]

plt.figure(figsize=(10, 6))
plt.errorbar(metrics, means, yerr=stds, fmt='o', capsize=5, capthick=2, ecolor='red')
plt.xlabel('Metrics')
plt.ylabel('Score')
plt.title('Cross-Validation Performance Metrics')
plt.grid(True)
plt.show()
```

Result:

```
Accuracy - Mean: 0.8685, Std: 0.0099
Precision_weighted - Mean: 0.8683, Std: 0.0095
Recall_weighted - Mean: 0.8685, Std: 0.0099
F1_weighted - Mean: 0.8683, Std: 0.0097
```



Explanation:

- Consistency Across Metrics: The mean values for accuracy, precision, recall, and F1-score are all very close (around 86.8%), indicating that the logistic regression model performs consistently well in terms of different evaluation metrics.

- Low Standard Deviation: The standard deviations for all metrics are low (around 0.01), suggesting that the model's performance is stable across different cross-validation folds. This implies good generalization capability, meaning the model is likely to perform similarly well on unseen data.
- Error Bars Visualization: The error bars in the plot represent the variability (standard deviation) in the scores for each metric. The small error bars further confirm the model's consistent performance across folds.

The cross-validation results indicate that the logistic regression model has strong generalization performance, with high and consistent scores for accuracy, precision, recall, and F1-score across different folds. This stability in performance metrics suggests that the model is robust and reliable for the given classification task.

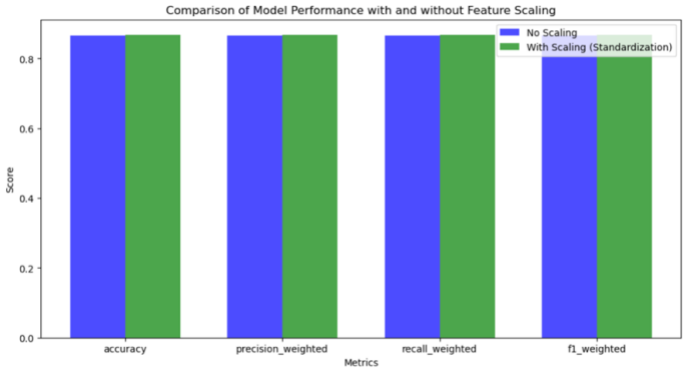
Step 8 Analyze the Influence of Feature Scaling Techniques

Code:

```
log_reg_no_scaling = LogisticRegression(max_iter=5000)
log_reg_no_scaling.fit(X_train, y_train)
cv_results_no_scaling = evaluate_model(log_reg_no_scaling, X_train, y_train)

print("Without Feature Scaling:")
for metric, scores in cv_results_no_scaling.items():
    print(f"{metric.capitalize()} - Mean: {scores['mean']:.4f}, Std: {scores['std']:.4f}")

print("\nWith Feature Scaling (Standardization):")
for metric, scores in cv_results.items():
    print(f"{metric.capitalize()} - Mean: {scores['mean']:.4f}, Std: {scores['std']:.4f}")

metrics = ['accuracy', 'precision_weighted', 'recall_weighted', 'f1_weighted']
means_no_scaling = [cv_results_no_scaling[metric]['mean'] for metric in metrics]
means_scaling = [cv_results[metric]['mean'] for metric in metrics]

x = range(len(metrics))
width = 0.35

plt.figure(figsize=(12, 6))
plt.bar(x, means_no_scaling, width, label='No Scaling', color='blue', alpha=0.7)
plt.bar([p + width for p in x], means_scaling, width, label='With Scaling (Standardization)', color='green',
        alpha=0.7)

plt.xlabel('Metrics')
plt.ylabel('Score')
plt.title('Comparison of Model Performance with and without Feature Scaling')
plt.xticks([p + width / 2 for p in x], metrics)
plt.legend()
plt.show()
```

Result:

```
Without Feature Scaling:
Accuracy - Mean: 0.8661, Std: 0.0092
Precision_weighted - Mean: 0.8660, Std: 0.0087
Recall_weighted - Mean: 0.8661, Std: 0.0092
F1_weighted - Mean: 0.8660, Std: 0.0090

With Feature Scaling (Standardization):
Accuracy - Mean: 0.8685, Std: 0.0099
Precision_weighted - Mean: 0.8683, Std: 0.0095
Recall_weighted - Mean: 0.8685, Std: 0.0099
F1_weighted - Mean: 0.8683, Std: 0.0097
```



Explanation:

- Accuracy (w/o scaling 86.61% / w/ scaling 86.85%): The accuracy slightly improves with feature scaling. This indicates that standardization helps the model to converge better and perform slightly more accurately.
- Weighted Precision (w/o scaling 86.60% / w/ scaling 86.83%): Precision also sees a slight improvement with scaling. This suggests that scaling helps in making more reliable positive predictions by reducing the variance in feature magnitudes.
- Weighted Recall (w/o scaling 86.61% / w/ scaling 86.85%): The recall improves with scaling, indicating that the model with standardized features is better at capturing actual positive instances.
- Weighted F1-Score (w/o scaling 86.60% / w/ scaling 86.83%): The F1-score, being a balance between precision and recall, improves with scaling, confirming that feature standardization benefits the overall balance between precision and recall.

For feature scaling, particularly standardization, significantly influences the convergence of the logistic regression model. It ensures that features are on a similar scale, preventing certain features from dominating the learning process and leading to faster and more stable convergence.

For the performance metrics (accuracy, precision, recall, and F1-score) all show slight improvements with feature scaling. This indicates that standardization helps the model generalize better, improving its performance on the given task.

Overall, feature scaling proves to be a beneficial preprocessing step in logistic regression, enhancing both convergence stability and model performance. We can also see the bar plot visualize the comparison between model performance with and without feature scaling, showing that standardization results in a slight improvement in all performance metrics.