자전거 대여 분석

연구주제

- 서울시 자전거 대여 수에 영향을 미치는 요인은 무엇인가?

- 변수 개수나 레코드 수도 적당하고 호기심을 유발하는 데이터세트였다. 연속형 타깃 변수가 존재해서 분류문제와 회귀문제를 모두 다룰 수 있는 점도 좋았다.

데이터세트 정보

Date(날짜) - 연월일
Rented Bike Count(임대 자전거 수) - 매시간 대여된 자전거 수
Hour(시간) - 그 날의 시간
Temperature(온도)- 섭씨 Humidity(습도) - %
Wind speed(풍속) - m/s
Visibility(가시성) - 10m
Dew point temperature(이슬점 온도) - 섭씨
Solar Radiation(태양 복사) - MJ/m2(제곱미터 당 메가 줄)
Rainfall(강우량) - mm
Snowfall(강설량) - cm
Seasons(계절) - 봄, 여름, 가을, 겨울
Holiday(휴일) - 공휴일/무휴
Functioning Day(영업여부) - 비영업시간, 영업시간

데이터 불러오기, ID 변수 설정

```
# 유일한 레코드 개수 세기
unique_records_count = df.drop_duplicates().shape[0]

print("유일한 레코드의 개수:", unique_records_count)
```

```
유일한 레코드의 개수: 8760
```

원래 데이터 세트에는 결측값이 없다. 현실적인 데이터분석을 하기 위해서 임의로 변수에 결측값을 만들었다. 이 데이터세트는 1시간마다 자전거 대여수를 기록하므로 중복되는 레코드가 없다. 그러므로 ID 변수가 없어도 괜찮다. Humidity에 5개, Wind speed에 4개, Holiday에 3개의 결측값이 있다.

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8760 entries, 0 to 8759
Data columns (total 14 columns):
 #   Column                 Non-Null Count  Dtype  
---  ------                 --------------  -----  
 0   Date                   8760 non-null   object 
 1   Rented Bike Count      8760 non-null   int64  
 2   Hour                   8760 non-null   int64  
 3   Temperature            8760 non-null   float64
 4   Humidity               8755 non-null   float64
 5   Wind speed             8756 non-null   float64
 6   Visibility             8760 non-null   int64  
 7   Dew point temperature  8760 non-null   float64
 8   Solar Radiation        8760 non-null   float64
 9   Rainfall               8760 non-null   float64
 10  Snowfall               8760 non-null   float64
 11  Seasons                8760 non-null   object 
 12  Holiday                8757 non-null   object 
 13  Functioning Day        8760 non-null   object 
dtypes: float64(7), int64(3), object(4)
memory usage: 958.2+ KB
```

타깃변수 설정

```
[ ]  df['Rented Bike Count'].median()
```

```
504.5
```

```
# 아래 코딩을 실시하면 경고문(warning)이 나타나는데 책의 흐름상 무시해도 무방

df.loc[df['Rented Bike Count'] >= 504.5, "Rented Bike B"] = 1  # Rented Bike Count가 120,000 이상이면 타겟변수 Rented Bike B 값은 1.
df.loc[df['Rented Bike Count'] <  504.5, "Rented Bike B"] = 0  # Rented Bike Count가 120,000 미만이면 타겟변수 Rented Bike B 값은 0.
df['Rented Bike B'].value_counts(dropna=False)        # Rented Bike B 값의 개수 분포 구하기.
```

```
Rented Bike B
0.000    4380
1.000    4380
Name: count, dtype: int64
```

자전거 대여수의 중위수를 기준으로 이진값 타깃 변수 Rented Bike B 를 만들었다.

기타 변수 데이터 처리

```
[ ] pd.options.display.float_format = '{:.2f}'.format # 소숫점 2자리로 숫자 표기 제한
    df[cols].describe()                              # 구간(interval) 변수의 요약통계 구하기
```

	Rented Bike Count	Temperature	Humidity	Wind speed	Visibility	Dew point temperature	Solar Radiation	Rainfall	Snowfall
count	8760.00	8760.00	8755.00	8756.00	8760.00	8760.00	8760.00	8760.00	8760.00
mean	704.60	12.88	58.22	1.72	1436.83	4.07	0.57	0.15	0.08
std	645.00	11.94	20.36	1.04	608.30	13.06	0.87	1.13	0.44
min	0.00	-17.80	0.00	0.00	27.00	-30.60	0.00	0.00	0.00
25%	191.00	3.50	42.00	0.90	940.00	-4.70	0.00	0.00	0.00
50%	504.50	13.70	57.00	1.50	1698.00	5.10	0.01	0.00	0.00
75%	1065.25	22.50	74.00	2.30	2000.00	14.80	0.93	0.00	0.00
max	3556.00	39.40	98.00	7.40	2000.00	27.20	3.52	35.00	8.80

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8760 entries, 0 to 8759
Data columns (total 14 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   Date                   8760 non-null   object
 1   Rented Bike Count      8760 non-null   int64
 2   Hour                   8760 non-null   int64
 3   Temperature            8760 non-null   float64
 4   Humidity               8755 non-null   float64
 5   Wind speed             8756 non-null   float64
 6   Visibility             8760 non-null   int64
 7   Dew point temperature  8760 non-null   float64
 8   Solar Radiation        8760 non-null   float64
 9   Rainfall               8760 non-null   float64
 10  Snowfall               8760 non-null   float64
 11  Seasons                8760 non-null   object
 12  Holiday                8757 non-null   object
 13  Functioning Day        8760 non-null   object
dtypes: float64(7), int64(3), object(4)
memory usage: 958.2+ KB
```

- 구간 변수: Rented Bike Count(타깃변수), Temperature, Humidity, wind speed, Visibility, Dew point temperature, Solar Radiation, SnowFall
- 범주형 변수: Date, Hour, Seasons, Holiday, Functioning Day

결측값 50% 초과변수 제거

```
print(df['Humidity'].isnull().mean())
print(df['Wind speed'].isnull().mean())
print(df['Holiday'].isnull().mean())
```

```
0.0005707762557077625
0.0004566210045662100 3
0.0003424657534246575 4
```

제거할만한 변수가 없었다.

요약 통계 및 도수분포표 검토

```
[ ]  df[cols].skew()
```

```
Rented Bike Count          1.15
Temperature               -0.20
Humidity                   0.06
Wind speed                 0.89
Visibility                -0.70
Dew point temperature     -0.37
Solar Radiation            1.50
Rainfall                  14.53
Snowfall                   8.44
dtype: float64
```

```
[ ]  df[cols].kurtosis()
```

```
Rented Bike Count          0.85
Temperature               -0.84
Humidity                  -0.80
Wind speed                 0.73
Visibility                -0.96
Dew point temperature     -0.76
Solar Radiation            1.13
Rainfall                 284.99
Snowfall                  93.80
dtype: float64
```

```
[ ]
pd.options.display.float_format = '{:.3f}'.format
df['Rainfall'].value_counts(normalize=True)
```

```
Rainfall
0.000    0.940
0.500    0.013
1.000    0.008
1.500    0.006
0.100    0.005
```

```
pd.options.display.float_format = '{:.3f}'.format
df['Snowfall'].value_counts(normalize=True)
```

```
Snowfall
0.000    0.949
0.300    0.005
1.000    0.004
0.900    0.004
0.500    0.004
```

Rainfall 변수와 Snowfall 변수는 왜도와 첨도가 너무 높게 나와서 제거하였다. 두 변수 모두 분포가 너무 극단적이었다.

이상값 제거

```
Lower = Q1-3.0*IQR
Upper = Q3+3.0*IQR
print(Lower)
```

```
Rented Bike Count        -2431.750
Temperature                -53.500
Humidity                   -54.000
Wind speed                  -3.300
Visibility               -2240.000
Dew point temperature       -63.200
Solar Radiation              -2.790
dtype: float64
```

```
print(Upper)
```

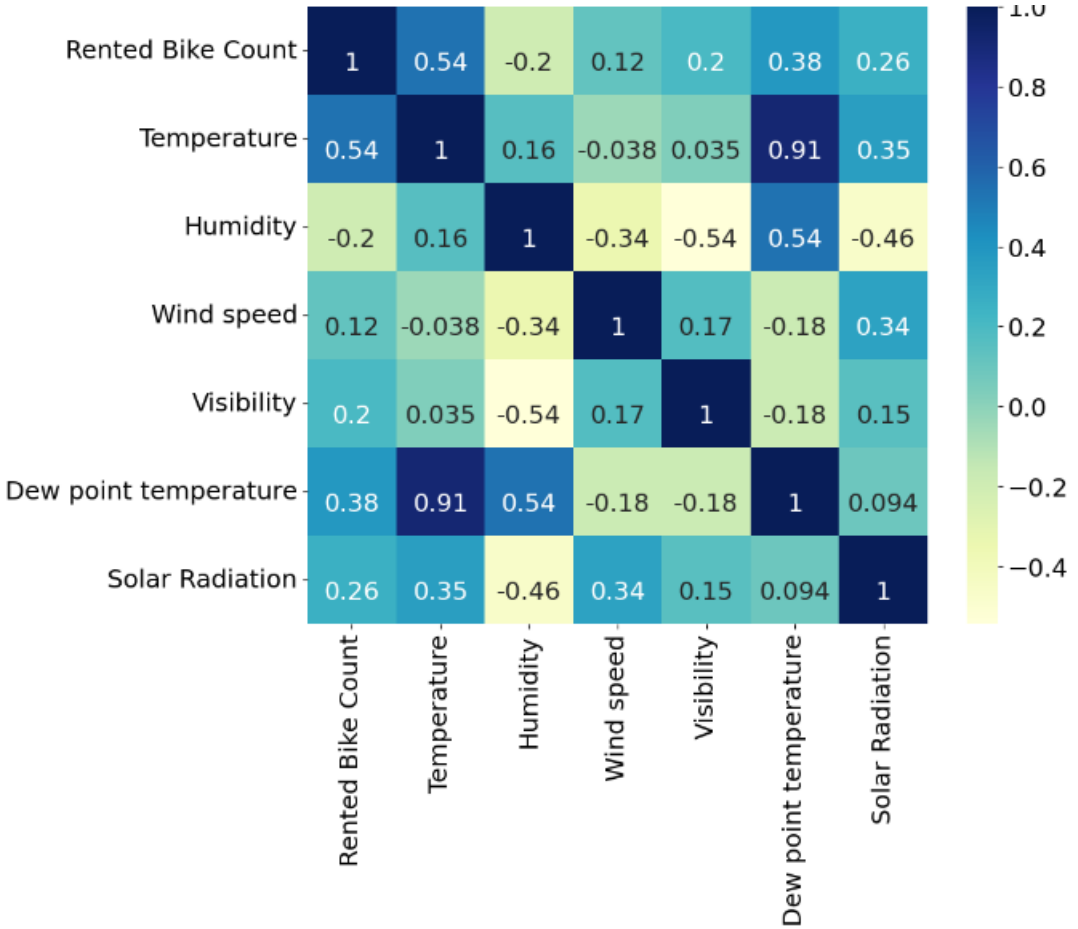
```
Rented Bike Count        3688.000
Temperature                79.500
Humidity                  170.000
Wind speed                  6.500
Visibility               5180.000
Dew point temperature       73.300
Solar Radiation              3.720
dtype: float64
```

```
c1 = df['Rented Bike Count'] <= 3688
c2 = df['Wind speed'] <= 6.5
c3 = df['Solar Radiation'] <= 3.72
df2 = df[c1&c2&c3]
df2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 8751 entries, 0 to 8759
Data columns (total 13 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   Date                   8751 non-null   object
 1   Rented Bike Count      8751 non-null   int64
 2   Hour                   8751 non-null   int64
 3   Temperature            8751 non-null   float64
 4   Humidity               8746 non-null   float64
 5   Wind speed             8751 non-null   float64
 6   Visibility             8751 non-null   int64
 7   Dew point temperature  8751 non-null   float64
 8   Solar Radiation        8751 non-null   float64
 9   Seasons                8751 non-null   object
 10  Holiday                8748 non-null   object
 11  Functioning Day        8751 non-null   object
 12  Rented Bike B          8751 non-null   float64
dtypes: float64(6), int64(3), object(4)
memory usage: 957.1+ KB
```

그래프를 보고 Rented Bike Count, Wind speed, Solar Radiation에 이상값이 존재하는 레코드를 제거하려 시도 하였고, 실제로는 Windspeed에 이상값이 존재하는 9개의 레코드가 제거되었다.(결측값 4개 포함) 나머지 2개 변수에 대해 이상값을 가지는 레코드는 없었다.

상관계수 검토



Dew point temperature과 Temperature의 상관계수가 0.7이상이여서 Dew point teperature을 제거했다.

T검정

```
[ ]  from scipy import stats

     data_1 = df2[df2['Rented Bike B'] == 1]['Temperature']
     data_0 = df2[df2['Rented Bike B'] == 0]['Temperature']

     stats.ttest_ind(data_1, data_0)   # 결과는 pvalue < 0.05
```

```
TtestResult(statistic=64.51521486646551, pvalue=0.0, df=8749.0)
```

```
[ ]  # 추가 코딩
     from scipy import stats

     data_1 = df2[df2['Rented Bike B'] == 1]['Humidity']
     data_0 = df2[df2['Rented Bike B'] == 0]['Humidity']

     stats.ttest_ind(data_1, data_0)   # 결과는 pvalue < 0.05
```

```
TtestResult(statistic=-17.15601632220937, pvalue=6.487308180792802e-65, df=8749.0)
```

```
[ ]  from scipy import stats

     data_1 = df2[df2['Rented Bike B'] == 1]['Wind speed']
     data_0 = df2[df2['Rented Bike B'] == 0]['Wind speed']

     stats.ttest_ind(data_1, data_0)   # 결과는 pvalue < 0.05
```

```
TtestResult(statistic=8.503484877615856, pvalue=2.1433039984344675e-17, df=8749.0)
```

```
[ ]  from scipy import stats

     data_1 = df2[df2['Rented Bike B'] == 1]['Visibility']
     data_0 = df2[df2['Rented Bike B'] == 0]['Visibility']

     stats.ttest_ind(data_1, data_0)   # 결과는 pvalue < 0.05
```

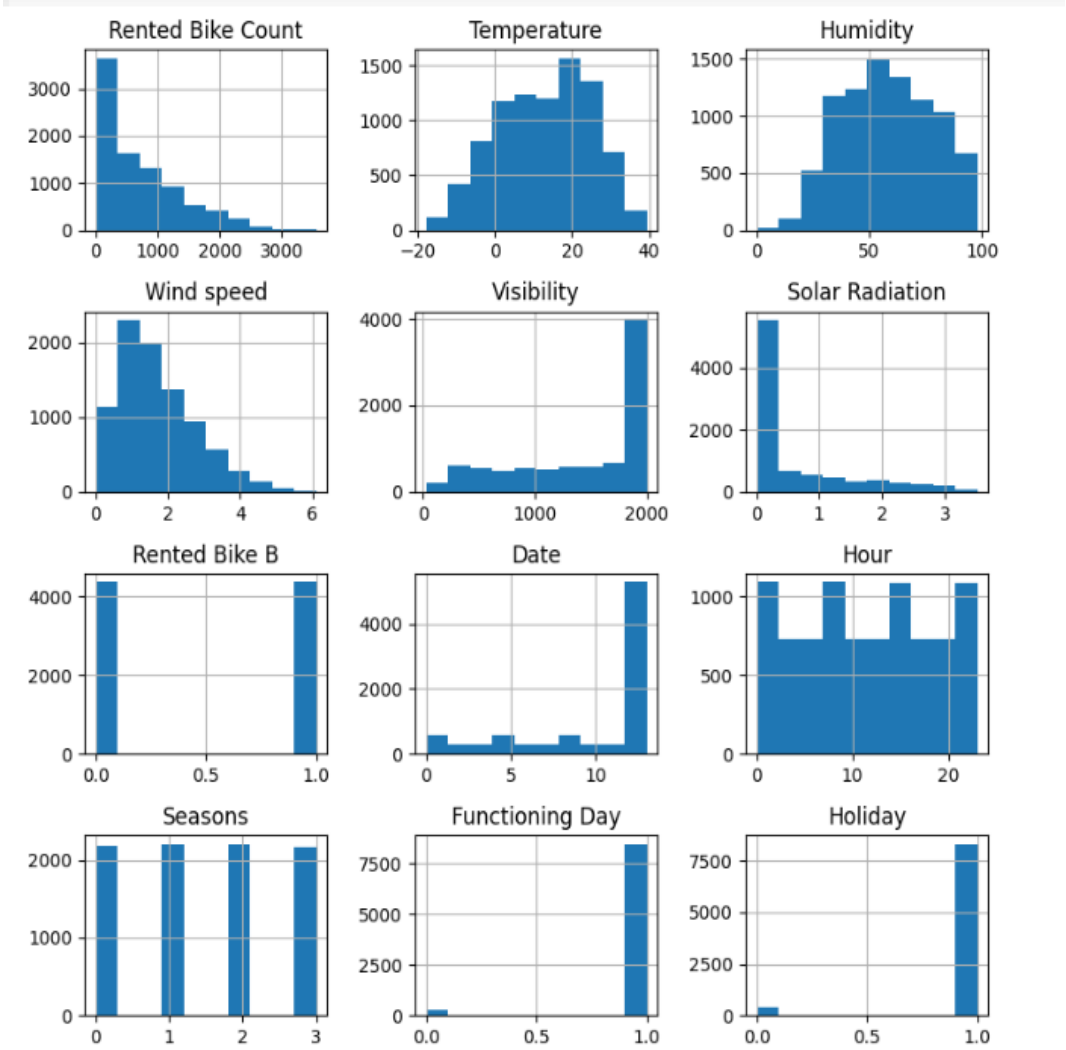
```
TtestResult(statistic=15.861804229325129, pvalue=6.97069383629778e-56, df=8749.0)
```

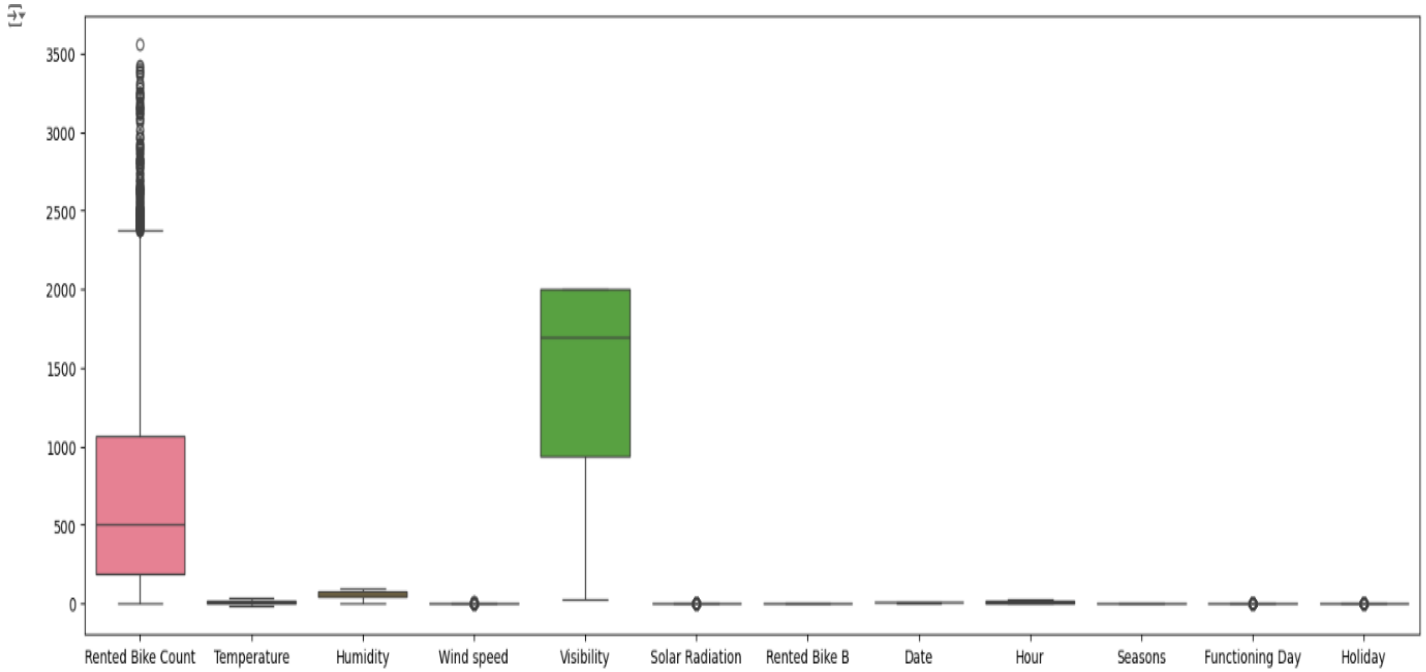
```
[ ]  from scipy import stats

     data_1 = df2[df2['Rented Bike B'] == 1]['Solar Radiation']
     data_0 = df2[df2['Rented Bike B'] == 0]['Solar Radiation']

     stats.ttest_ind(data_1, data_0)   # 결과는 pvalue < 0.05
```

```
TtestResult(statistic=38.609114238936044, pvalue=2.827727742069316e-301, df=8749.0)
```

다섯개의 변수 모두 p값이 0에 수렴하므로 두 그룹의 자전거 대여 수가 같다는 가설을 기각할 수 있다. 두 그룹의 평균 자전거 대여 수는 통계적으로 유의미하게 다르다.

시각화





범주형 변수를 인코딩해서
수치형 변수로 만들었다.
그리고 히스토그램과 박스
플롯으로 시각화했다.

데이터 추가 처리

Date	
2001-06-18	0.002743
2008-08-18	0.002743
2006-08-18	0.002743
2005-08-18	0.002743
2004-08-18	0.002743
2003-08-18	0.002743
2002-08-18	0.002743
2001-08-18	0.002743
31/07/2018	0.002743
30/07/2018	0.002743

```python
import pandas as pd
from dateutil import parser
from sklearn.preprocessing import OrdinalEncoder
# 예제 날짜 데이터
# 날짜 형식 변환 함수
def parse_date(date_str):
    try:
        return parser.parse(date_str, dayfirst=True)
    except ValueError:
        return None
df['Date'] = df['Date'].astype(str)
# 날짜 형식으로 변환
df['Date'] = df['Date'].apply(parse_date)

# 연도 추출
df['Date'] = df['Date'].dt.year

# Ordinal Encoding
encoder = OrdinalEncoder()
df['Date_encoded'] = encoder.fit_transform(df['Date'].values.reshape(-1, 1))

# 년도별로 그룹핑
df.groupby([ 'Date', 'Date_encoded']).size()
```

Date 변수의 형식이 2가지여서 날짜 형식으로 변환한다음 연도를 추출하였다.

데이터 추가 처리

Date	Date_encoded	
2001	0.0	287
2002	1.0	287
2003	2.0	288
2004	3.0	288
2005	4.0	288
2006	5.0	288
2007	6.0	288
2008	7.0	287
2009	8.0	288
2010	9.0	285
2011	10.0	288
2012	11.0	288
2017	12.0	456
2018	13.0	4845

dtype: int64

Hour	Hour_encoded	
0	0.0	365
1	1.0	365
2	2.0	365
3	3.0	365
4	4.0	364
5	5.0	365
6	6.0	365
7	7.0	364
8	8.0	365
9	9.0	365
10	10.0	365
11	11.0	365
12	12.0	365
13	13.0	365
14	14.0	364
15	15.0	364
16	16.0	364
17	17.0	365
18	18.0	364
19	19.0	365
20	20.0	365
21	21.0	362
22	22.0	365
23	23.0	365

dtype: int64

Seasons	Seasons_encoded	
Autumn	0.0	2184
Spring	1.0	2203
Summer	2.0	2207
Winter	3.0	2157

dtype: int64

Functioning Day	Functioning Day_encoded	
No	0.0	295
Yes	1.0	8456

dtype: int64

Holiday	Holiday_encoded	
Holiday	0.0	430
No Holiday	1.0	8318

dtype: int64

데이터 대체

```
[ ]  # 결측값을 평균값으로 대체
     df['Humidity'].fillna(df['Humidity'].mean(), inplace=True)
     mode_value = df['Holiday_encoded'].mode()[0]

     # 결측값을 최빈값으로 대체
     df['Holiday_encoded'].fillna(mode_value, inplace=True)
     df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8751 entries, 0 to 8750
Data columns (total 12 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   Rented Bike Count       8751 non-null   int64
 1   Temperature             8751 non-null   float64
 2   Humidity                8751 non-null   float64
 3   Wind speed              8751 non-null   float64
 4   Visibility              8751 non-null   int64
 5   Solar Radiation         8751 non-null   float64
 6   Rented Bike B           8751 non-null   float64
 7   Date_encoded            8751 non-null   float64
 8   Hour_encoded            8751 non-null   float64
 9   Seasons_encoded         8751 non-null   float64
 10  Functioning Day_encoded  8751 non-null   float64
 11  Holiday_encoded         8751 non-null   float64
dtypes: float64(10), int64(2)
memory usage: 820.5 KB
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8751 entries, 0 to 8750
Data columns (total 12 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   Rented Bike Count   8751 non-null   int64
 1   Temperature         8751 non-null   float64
 2   Humidity            8751 non-null   float64
 3   Wind speed          8751 non-null   float64
 4   Visibility          8751 non-null   int64
 5   Solar Radiation     8751 non-null   float64
 6   Rented Bike B       8751 non-null   float64
 7   Date                8751 non-null   float64
 8   Hour                8751 non-null   float64
 9   Seasons             8751 non-null   float64
 10  Functioning Day     8751 non-null   float64
 11  Holiday             8751 non-null   float64
dtypes: float64(10), int64(2)
memory usage: 820.5 KB
```

구간 변수인 Humidity는 결측값을 평균값으로 대체하였고
범주형 변수인 Holiday는 결측값을 최빈값으로 대체하였다.
뒤에 붙은 _encoded는 이름을 바꾸어 제거해주었다.

데이터 추가 처리

타깃 변수	변경없음	StandScaler 표준화	로그 변환
이진값 타깃 변수	Bike-unscaled.csv	Bike-standard.csv	Bike-log.csv
연속형 타깃 변수	Bike-unscaled-Count.csv	Bike-standard-Count.csv	Bike-log-Count.csv

```
cols1 = ['Date','Hour','Seasons']
```

```
[ ]  df1 = pd.get_dummies(df, columns=cols1)    # cols1에 담긴 변수들의 더미변수를 생성
                                                # 이 명령은 더미변수를 생성한 원본변수는 제거함에 유의
```

```
bool_columns = df1.select_dtypes(include=['bool']).columns

# 각 bool 컬럼에 대해 True를 1로, False를 0으로 변환합니다.
df1['Functioning Day'] = df['Functioning Day'].astype(int)
df1['Holiday'] = df['Holiday'].astype(int)

for column in bool_columns:
    df1[column] = df1[column].astype(int)
```

```
df1.head(3)
```

	Rented Bike Count	Temperature	Humidity	Wind speed	Visibility	Solar Radiation	Rented Bike B	Functioning Day	Holiday	Date_0.0	Date_1.0	Date_2.0	Date_3.0	Date_4.0	Date_5.0	Date_6.0	Date_7.0
0	254	-5.20	37.00	2.20	2000	0.00	0.00	1	1	1	0	0	0	0	0	0	0
1	204	-5.50	38.00	0.80	2000	0.00	0.00	1	1	1	0	0	0	0	0	0	0
2	173	-6.00	39.00	1.00	2000	0.00	0.00	1	1	1	0	0	0	0	0	0	0

데이터 추가 처리

```
# 기준 더미 변수(base dummy variable)로 정한 3개 더미 변수명을 cols2에 저장
cols2 = ['Date_0.0','Hour_0.0','Seasons_0.0']

df1.drop(cols2, axis=1, inplace=True)   # cols2에 저장된 3개 더미 변수명을 데이터프레임에서 제거
df1.shape
```

```
(8751, 48)
```

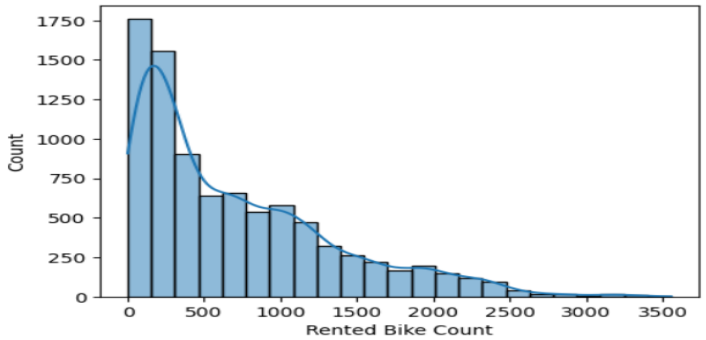
```
Date  Date_encoded
2001  0.0             287
2002  1.0             287
2003  2.0             288
2004  3.0             288
2005  4.0             288
2006  5.0             288
2007  6.0             288
2008  7.0             287
2009  8.0             288
2010  9.0             285
2011  10.0            288
2012  11.0            288
2017  12.0            456
2018  13.0           4845
dtype: int64
```

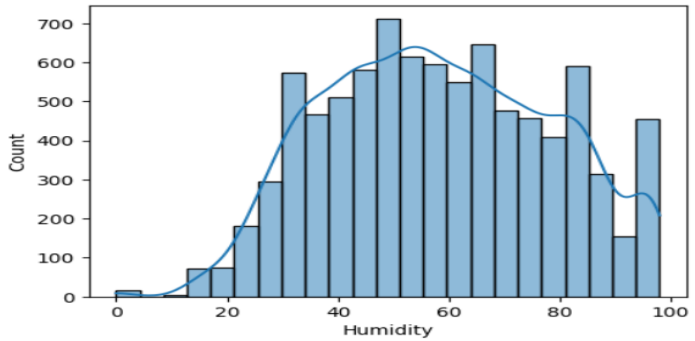
```
Hour  Hour_encoded
0     0.0             365
1     1.0             365
2     2.0             365
3     3.0             365
4     4.0             364
5     5.0             365
6     6.0             365
7     7.0             364
8     8.0             365
9     9.0             365
10    10.0            365
11    11.0            365
12    12.0            365
13    13.0            365
14    14.0            364
15    15.0            364
16    16.0            364
17    17.0            365
18    18.0            364
19    19.0            365
20    20.0            365
21    21.0            362
22    22.0            365
23    23.0            365
dtype: int64
```

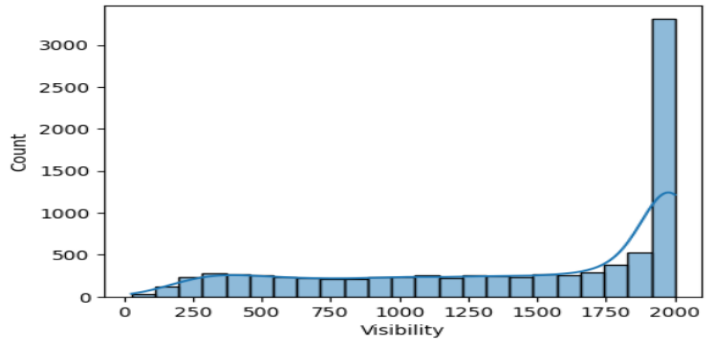
```
Seasons  Seasons_encoded
Autumn   0.0             2184
Spring   1.0             2203
Summer   2.0             2207
Winter   3.0             2157
dtype: int64
```

Date, Hour, Seasons 더미 변수의 분포가 균등해서 Date_0.0, Hour_0.0, Seasons_0.0을 기준 더미 변수로 정했다. 기준 더미 변수를 데이터 프레임에서 제거하였다.

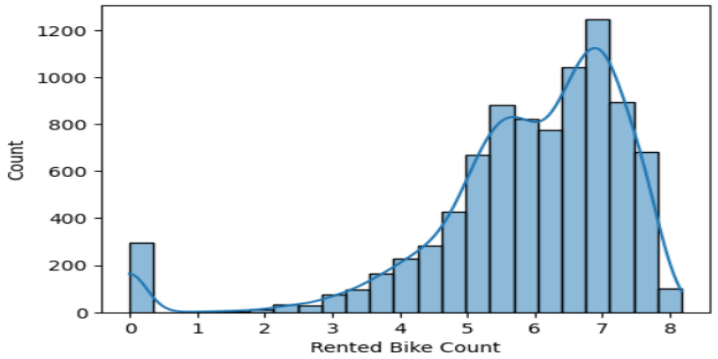
데이터 추가 처리

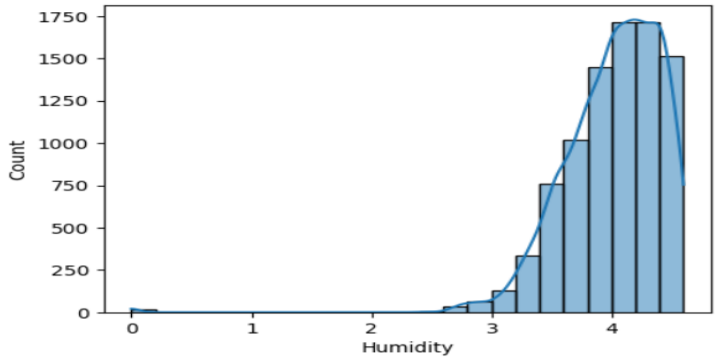


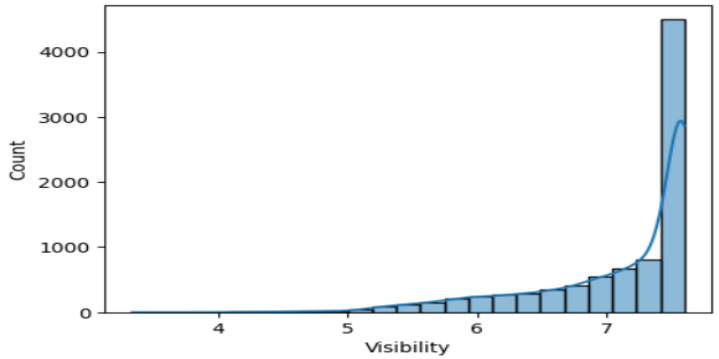




로그 변환 전

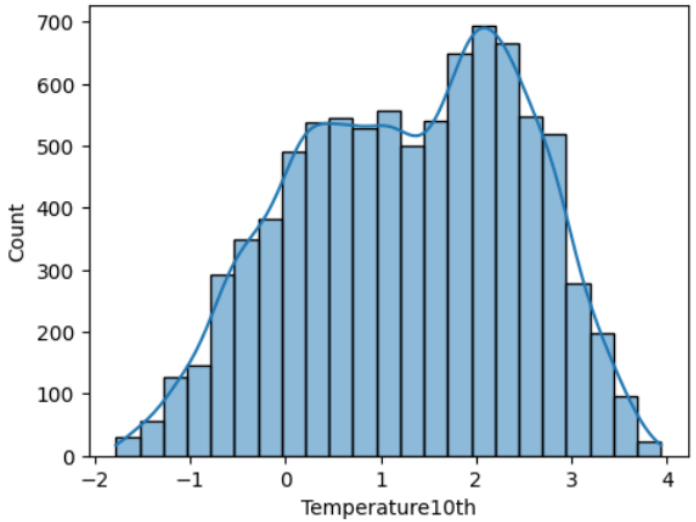


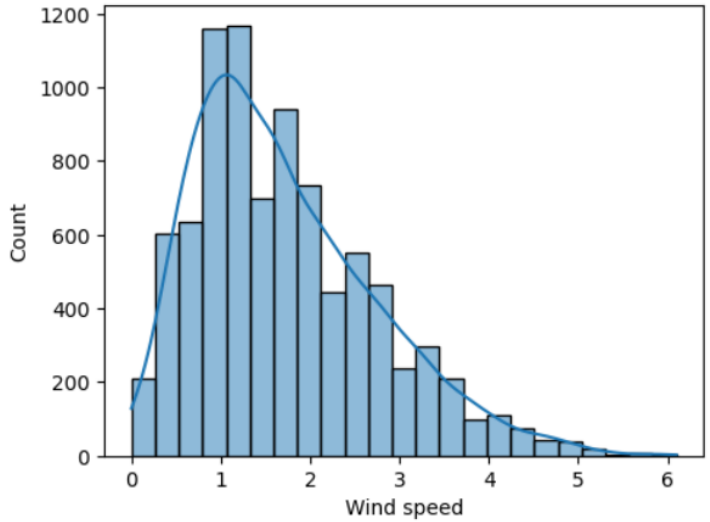


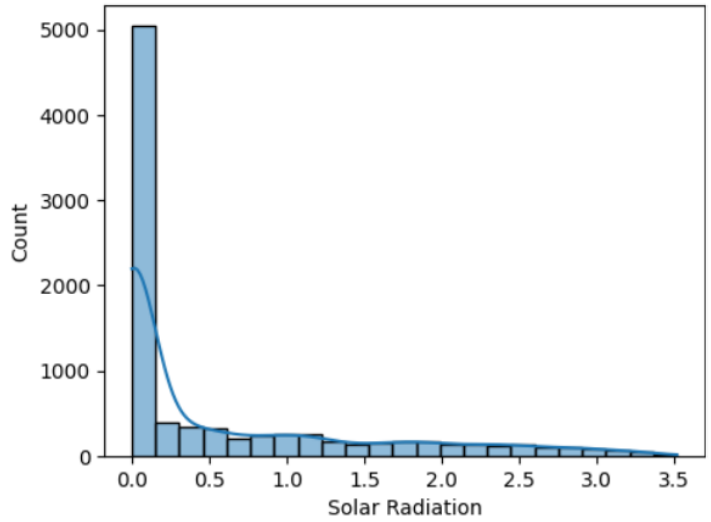


로그 변환 후

데이터 추가 처리

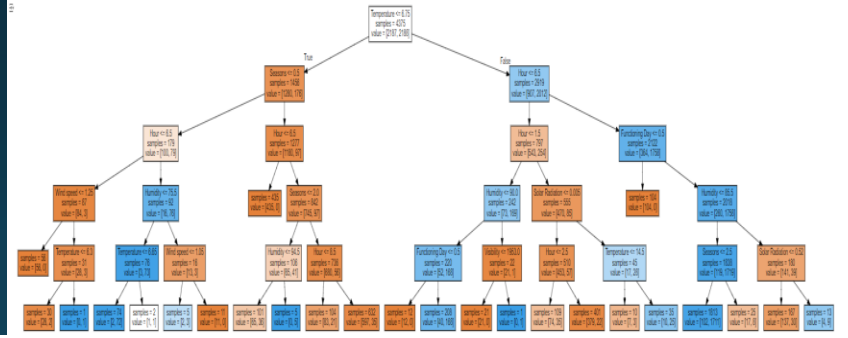






값이 큰 Rented Bike Count, Humidity, Visibility는 로그 변환을 하였다. Temperature은 음수를 포함해서 로그변환 대신에 10분의 1을 곱해주었다. wind speed, Solar Radiation은 로그 변환을 하지 않았다.

결정트리(분류)



```
print("Accuracy(GINI) on training set:{:.5f}".format(model.score(X_train, y_train)))
print("Accuracy(GINI) on test set:{:.5f}".format(accuracy_score(y_test, pred)))
```

```
Accuracy(GINI) on training set:1.00000
Accuracy(GINI) on test set:0.90014
```

```
[ ] best_clf = grid_tree.best_estimator_
    pred = best_clf.predict(X_test)
    print("Accuracy on test set:{:.5f}".format(accuracy_score(y_test, pred)))
```

```
Accuracy on test set:0.91248
```

```
[ ] # 참조 코딩
    from sklearn.metrics import roc_auc_score
    ROC_AUC = roc_auc_score(y_test,best_clf.predict_proba(X_test)[:, 1])
    print("ROC AUC on test set:{:.5f}".format(ROC_AUC))
```

```
ROC AUC on test set:0.95070
```

```
# Decision Tree 모델 (Gini 기준)
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
from sklearn.model_selection import GridSearchCV

tree = DecisionTreeClassifier(criterion="gini", random_state=0, max_depth=5)

params = {'criterion':['gini','entropy'],'max_depth': range(1,21)}

grid_tree = GridSearchCV(tree, param_grid=params, scoring='accuracy', cv=5, n_jobs=-1,
                         verbose=1)
grid_tree.fit(X_train, y_train)

print("GridSearchCV max accuracy:{:.5f}".format(grid_tree.best_score_))
print("GridSearchCV best parameter:", (grid_tree.best_params_))
```

```
Fitting 5 folds for each of 40 candidates, totalling 200 fits
GridSearchCV max accuracy:0.90697
GridSearchCV best parameter: {'criterion': 'gini', 'max_depth': 5}
```

로지스틱 회귀(분류)실행 및 결과 해석

```
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:1173:
  warnings.warn(
Logreg Training set score:0.89486
Logreg Test set score:0.90356
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:1173: FutureWa
  warnings.warn(
GridSearchCV max accuracy:0.88617
GridSearchCV best parameter: {'penalty': 'none', 'solver': 'lbfgs'}
```

```
Accuracy on test set:0.90356
```

변화 없음

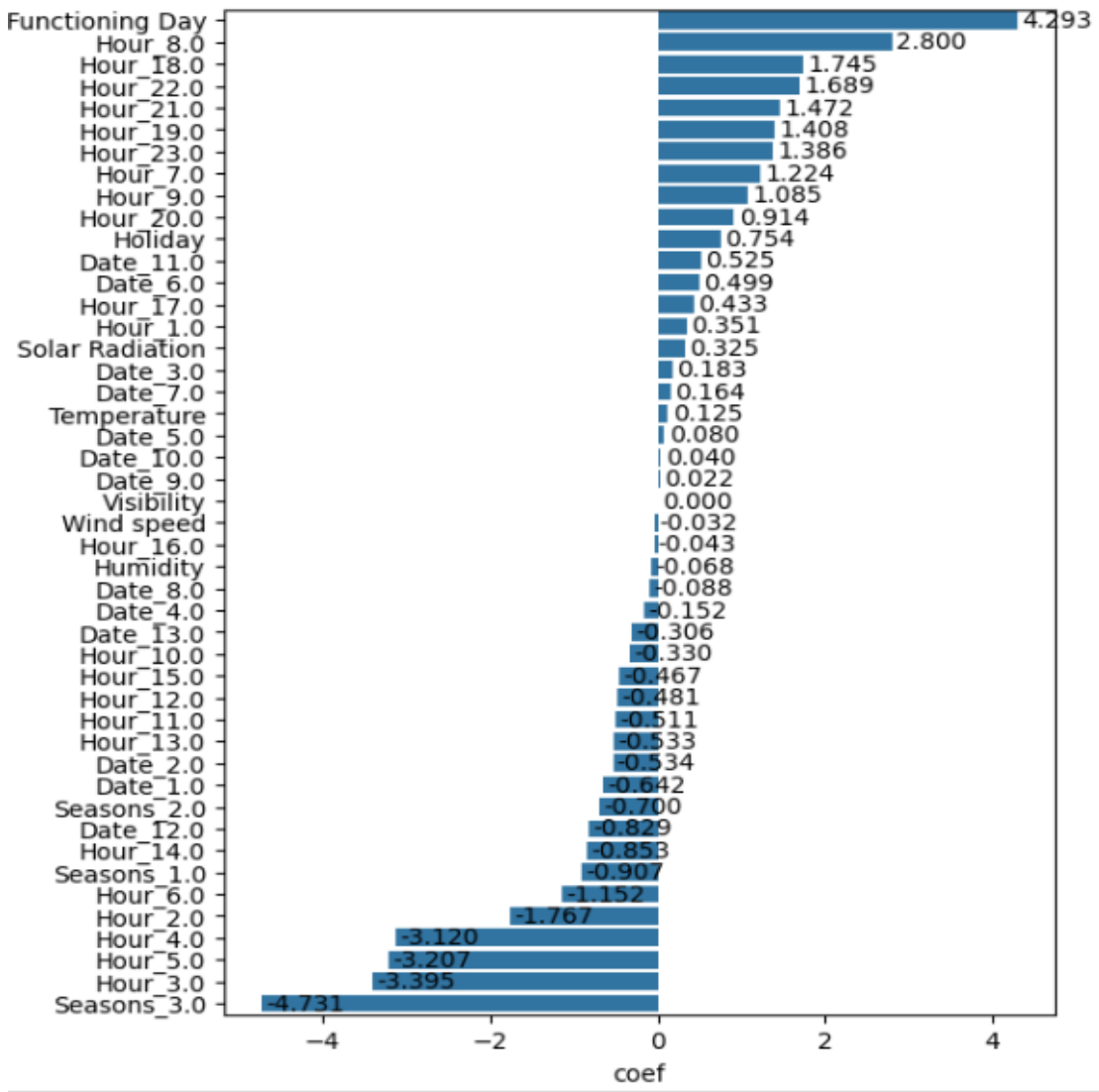
```
/usr/local/lib/python3.10/dist-packages
  warnings.warn(
Logreg Training set score:0.90560
Logreg Test set score:0.90516
```

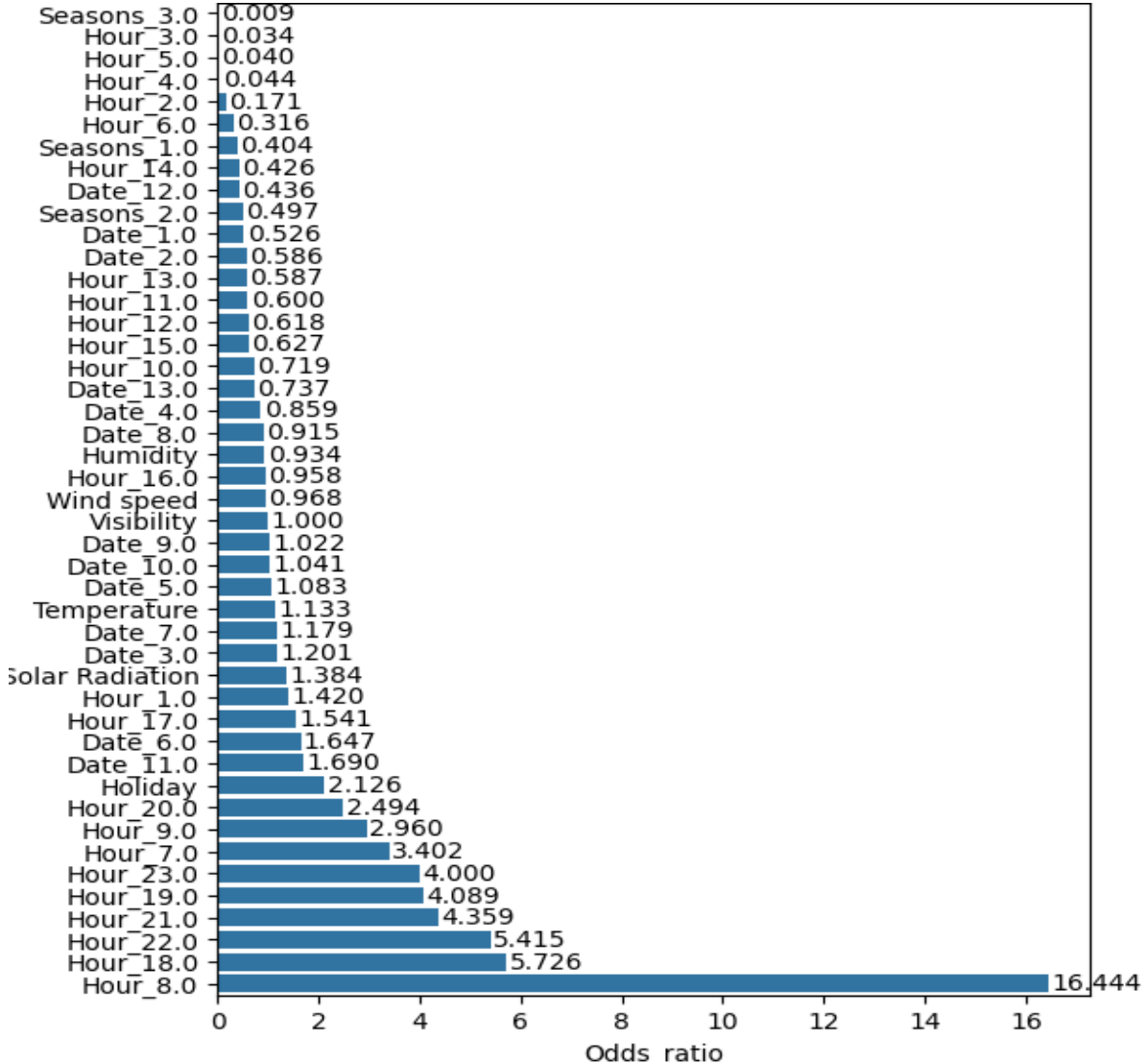
```
GridSearchCV max accuracy:0.90057
GridSearchCV best parameter: {'penalty': 'none', 'solver': 'lbfgs'}
```

```
Accuracy on test set:0.90516
```

표준화

로지스틱 회귀(분류)실행 및 결과 해석





로지스틱 회귀(분류)실행 및 결과 분석

- 로지스틱 회귀에서는 계수보다 오즈비가 훨씬 중요하다.
- 연속형 변수 해석 : 온도가 1단위 증가할 경우, 자전거 대여수가 중위값 이상일 가능성은 약 13퍼 증가한다. 바람속도가 1단위 증가할 경우, 거래금액이 중위값 이상일 가능성은 약 3퍼 감소한다.
- 범주형 변수 해석 : 평일이 공휴일에 비해 자전거 대여 횟수가 2.126배 높다.
- 결론적으로 시간, 공휴일 여부 등은 거래금액에 큰 영향을 미친다. 오즈비 그래프에서 영업여부 변수가 오즈비가 50이상 나와서 원활한 그래프 분석을 위해 제거하였다.

신경망(사이킷런, 분류)

```
print ("Neural Network Training set score:{:.5f}".format(clf_mlp.score(X_train, y_train)))
print ("Neural Network Test set score:{:.5f}".format(accuracy_score(y_test, pred)))
```

```
Neural Network Training set score:0.98743
Neural Network Test set score:0.91042
```

```
best_clf = grid_mlp.best_estimator_
pred = best_clf.predict(X_test)
print("Accuracy on test set:{:.5f}".format(accuracy_score(y_test, pred)))
```

```
Accuracy on test set:0.91933
```

```
# 신경망 기본 모델 (with adam solver)
clf_mlp = MLPClassifier(max_iter = 2000, random_state = 0)

# 그리드 서치 재실행
from sklearn.model_selection import GridSearchCV
params = {'solver':['adam', 'sgd'],
          'alpha':[0.0001, 0.01],
          'activation':['relu','tanh'],
          'hidden_layer_sizes': [(100,), (100,100)]
         }

grid_mlp = GridSearchCV(clf_mlp, param_grid=params, scoring='accuracy', cv=5, n_jobs=-1)
grid_mlp.fit(X_train, y_train)

print("GridSearchCV max accuracy:{:.5f}".format(grid_mlp.best_score_))
print("GridSearchCV best parameter:", (grid_mlp.best_params_))
```

```
GridSearchCV max accuracy:0.91520
GridSearchCV best parameter: {'activation': 'relu', 'alpha': 0.01, 'hidden_layer_sizes': (100, 100), 'solver': 'sgd'}
```

K-최근접이웃(분류)

```
print ("KNN Training set score:{:.5f}".format(clf_knn.score(X_train, y_train)))
print ("KNN Test set score:{:.5f}".format(accuracy_score(y_test, pred)))
```

```
KNN Training set score:0.93920
KNN Test set score:0.87454
```

```
[ ] best_clf = grid_knn.best_estimator_
    pred = best_clf.predict(X_test)
    print("Accuracy on test set:{:.5f}".format(accuracy_score(y_test, pred)))
```

```
Accuracy on test set:0.88551
```

```
[ ] # KNN 모델 (Default 모델 with n_neighbors=3)
    clf_knn = KNeighborsClassifier(n_neighbors=3) # random_state 파라미터가 없음에 주의!

    # 그리드 서치 실행
    from sklearn.model_selection import GridSearchCV
    params = {'n_neighbors': range(3, 31)}

    grid_knn = GridSearchCV(clf_knn, param_grid=params, scoring='accuracy', cv=3, n_jobs=-1)
    grid_knn.fit(X_train, y_train)

    print("GridSearchCV max accuracy:{:.5f}".format(grid_knn.best_score_))
    print("GridSearchCV best parameter:", (grid_knn.best_params_))
```

```
GridSearchCV max accuracy:0.88046
GridSearchCV best parameter: {'n_neighbors': 5}
```

랜덤 포레스트(분류)

```
print ("Random Forest Accuracy on training set:{:.5f}".format(model.score(X_train, y_train)))
print ("Random Forest Accuracy on test set:{:.5f}".format(accuracy_score(y_test, pred)))
```

```
Random Forest Accuracy on training set:1.00000
Random Forest Accuracy on test set:0.91865
```

```
best_clf = grid_rf.best_estimator_
pred = best_clf.predict(X_test)
print("Accuracy on test set:{:.5f}".format(accuracy_score(y_test, pred)))

from sklearn.metrics import roc_auc_score
ROC_AUC = roc_auc_score(y_test,best_clf.predict_proba(X_test)[:, 1])
print("ROC AUC on test set:{:.5f}".format(ROC_AUC))
```

```
Accuracy on test set:0.91659
ROC AUC on test set:0.97850
```

```
# 그리드 서치 실행

from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import StratifiedKFold

# StratifiedKFold의 random_state 옵션값을 특정 숫자(예: 0)로 고정
cross_validation = StratifiedKFold(n_splits=5, shuffle=True, random_state=0)
params = {'max_depth': range(10, 41), 'n_estimators': [100, 200]}

# GridSearchCV의 cv=cross_validation 옵션값은 위의 StratifiedKFold의 random_state 옵션값을 적용시켜서
# GridSearchCV를 실행할 때마다 결과가 항상 동일하게 나오도록 보장
grid_rf = GridSearchCV(rf, param_grid=params, scoring='accuracy', cv=cross_validation,
                       verbose=1, n_jobs=-1)
grid_rf.fit(X_train, y_train)

print("GridSearchCV max accuracy:{:.5f}".format(grid_rf.best_score_))
print("GridSearchCV best parameter:", (grid_rf.best_params_))
```

```
Fitting 5 folds for each of 62 candidates, totalling 310 fits
GridSearchCV max accuracy:0.92297
GridSearchCV best parameter: {'max_depth': 18, 'n_estimators': 200}
```

그레이디언트 부스팅(분류)

```
print ("grbt Accuracy on training set:{:.5f}".format(model.score(X_train, y_train)))
print ("grbt Accuracy on test set:{:.5f}".format(accuracy_score(y_test, pred)))
```

```
grbt Accuracy on training set:0.93257
grbt Accuracy on test set:0.91819
```

```
best_clf = grid_gr.best_estimator_
pred = best_clf.predict(X_test)
print("Accuracy on test set:{:.5f}".format(accuracy_score(y_test, pred)))

from sklearn.metrics import roc_auc_score
ROC_AUC = roc_auc_score(y_test,best_clf.predict_proba(X_test)[:, 1])
print("ROC AUC on test set:{:.5f}".format(ROC_AUC))
```

```
Accuracy on test set:0.91476
ROC AUC on test set:0.97437
```

```
# 그리드 서치 실행
# 아래 코딩 실행에 48초 소요
import time
start = time.time()

from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import StratifiedKFold

# Gradient Boosting 모델 (Default 모델)
gr = GradientBoostingClassifier(random_state = 0)

# StratifiedKFold의 random_state 옵션값을 특정 숫자(예: 0)로 고정
cross_validation = StratifiedKFold(n_splits=3, shuffle=True, random_state=0)
params = {'max_depth': range(5, 51, 5)}

# GridSearchCV의 cv=cross_validation 옵션값은 위의 StratifiedKFold의 random_state 옵션값을 적용시켜서
# GridSearchCV를 실행할 때마다 결과가 항상 동일하게 나오도록 보장
grid_gr = GridSearchCV(model, param_grid=params, scoring='accuracy', cv=cross_validation,₩
                       n_jobs=-1)
grid_gr.fit(X_train, y_train)

print("GridSearchCV max accuracy:{:.5f}".format(grid_gr.best_score_))
print("GridSearchCV best parameter:", (grid_gr.best_params_))

end = time.time()
print(f"Runtime of the program is {end - start}")
```

```
GridSearchCV max accuracy:0.92114
GridSearchCV best parameter: {'max_depth': 5}
Runtime of the program is 48.307960987091064
```

라쏘(로지스틱 회귀모델(분류)에 L1규제화한 분류모델)

```
Lasso Accuracy on training set:0.90514
Lasso Accuracy on test set:0.90265
Fitting 5 folds for each of 7 candidates, totalling 35 fits
GridSearchCV max accuracy:0.90080
GridSearchCV best parameter: {'C': 0.5, 'solver': 'liblinear'}
Accuracy on test set:0.90631
ROC AUC on test set:0.96721
```

표준화

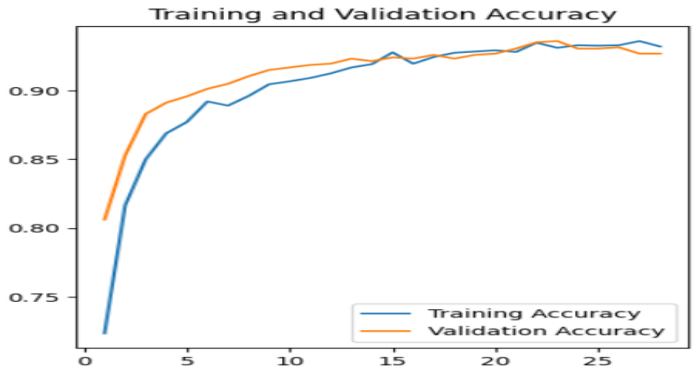
```
Lasso Accuracy on training set:0.89851
Lasso Accuracy on test set:0.89922
Fitting 5 folds for each of 7 candidates, totalling 35 fits
GridSearchCV max accuracy:0.89509
GridSearchCV best parameter: {'C': 1, 'solver': 'liblinear'}
Accuracy on test set:0.89922
ROC AUC on test set:0.96261
```

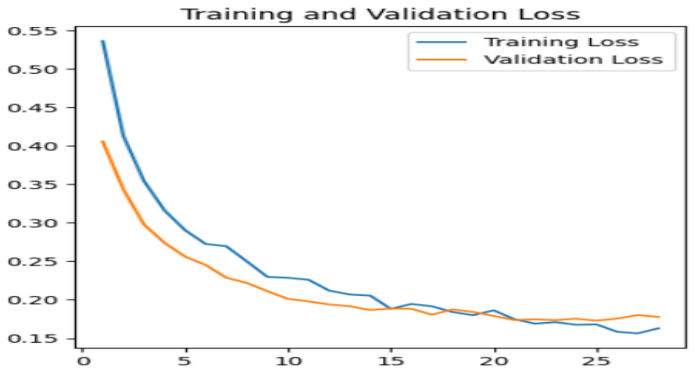
로그 변환

```
Lasso Accuracy on training set:0.90629
Lasso Accuracy on test set:0.90334
Fitting 5 folds for each of 7 candidates, totalling 35 fits
GridSearchCV max accuracy:0.90217
GridSearchCV best parameter: {'C': 1, 'solver': 'liblinear'}
Accuracy on test set:0.90334
ROC AUC on test set:0.96727
```

변화 없음

신경망(텐서플로 케라스, 분류)






```
test_loss, test_accuracy = model.evaluate(X_test, y_test)
print("test accuracy:", test_accuracy)
```

```
137/137 [==============================] - 0s 2ms/step - loss: 0.1930 - accuracy: 0.9170
test accuracy: 0.9170475602149963
```

```
[ ]  from sklearn.metrics import roc_auc_score

     y_prob = model.predict(X_test)
     ROC_AUC = roc_auc_score(y_test, y_prob)
     print("ROC AUC on test set:{:.5f}".format(ROC_AUC))
```

```
137/137 [==============================] - 0s 2ms/step
ROC AUC on test set:0.97865
```

SVM(분류)

```
print ("SVM Accuracy on training set:{:.5f}".format(model.score(X_train, y_train)))
print ("SVM Accuracy on test set:{:.5f}".format(accuracy_score(y_test, pred)))
```

```
SVM Accuracy on training set:0.91589
SVM Accuracy on test set:0.91065
```

```
best_clf = grid_svm.best_estimator_
pred = best_clf.predict(X_test)
print("Accuracy on test set:{:.5f}".format(accuracy_score(y_test, pred)))

from sklearn.metrics import roc_auc_score
ROC_AUC = roc_auc_score(y_test,best_clf.predict_proba(X_test)[:, 1])
print("ROC AUC on test set:{:.5f}".format(ROC_AUC))
```

```
Accuracy on test set:0.91682
ROC AUC on test set:0.97253
```

```
# SVM model (default 모델)
svm = SVC(kernel='rbf', C=1, gamma = 'auto', random_state=0, probability=True)
                                                    # probability=True 에 주의

# 그리드 서치 재실행. 시간이 많이 걸림에 주의
import time
start = time.time()

from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import StratifiedKFold

# StratifiedKFold의 random_state 옵션값을 특정 숫자(예: 0)로 고정
cross_validation = StratifiedKFold(n_splits=3, shuffle=True, random_state=0)
params = {'kernel':['rbf'], 'C':[0.0001, 0.01, 1, 10],
          'gamma':['auto','scale']}

grid_svm = GridSearchCV(svm, param_grid=params, scoring='accuracy',
                        cv=cross_validation, n_jobs=-1)
grid_svm.fit(X_train, y_train)

print("GridSearchCV max accuracy:{:.5f}".format(grid_svm.best_score_))
print("GridSearchCV best parameter:", (grid_svm.best_params_))

end = time.time()
print(f"Runtime of the program is {end - start}")
```

```
GridSearchCV max accuracy:0.91840
GridSearchCV best parameter: {'C': 10, 'gamma': 'scale', 'kernel': 'rbf'}
Runtime of the program is 77.03572988510132
```

선형 회귀(연속 타깃 변수)

```
print ("Linear Regression Training set r2 score:{:.5f}".format(model.score(X_train, y_train)))
print ("Linear Regression Test set r2 score:{:.5f}".format(r2_score(y_test, pred)))
```

```
Linear Regression Training set r2 score:0.65731
Linear Regression Test set r2 score:0.64409
```

표준화

```
[ ]  # 연속변수 타겟변수일 때 Linear Regression 모델 (Default 모델)
     from sklearn.linear_model import LinearRegression
     lin_r = LinearRegression(n_jobs=-1)
     model = lin_r.fit(X_train, y_train)
     pred = model.predict(X_test)

     print ("Linear Regression Training set score:{:.5f}".format(model.score(X_train, y_train)))
     print ("Linear Regression Test set score:{:.5f}".format(r2_score(y_test, pred)))
```

```
Linear Regression Training set score:0.80993
Linear Regression Test set score:0.80460
```

로그 변환

```
[ ]  # 연속변수 타겟변수일 때 Linear Regression 모델 (Default 모델)
     from sklearn.linear_model import LinearRegression
     from sklearn.metrics import r2_score

     linr = LinearRegression(n_jobs=-1)
     model = linr.fit(X_train, y_train)
     pred = model.predict(X_test)

     print ("Linear Regression Training set score:{:.5f}".format(model.score(X_train, y_train)))
     print ("Linear Regression Test set r2 score:{:.5f}".format(r2_score(y_test, pred)))
```

```
Linear Regression Training set score:0.65731
Linear Regression Test set r2 score:0.64409
```

변화 없음

릿지 회귀(연속 타깃 변수)

```
Ridge Regression Training set score:0.65730
Ridge Regression Test set score:0.64422
```

```
Fitting 5 folds for each of 49 candidates, totalling 245 fits
GridSearchCV max score:0.64674
GridSearchCV best parameter: {'alpha': 1, 'solver': 'saga'}
```

```
R2 Score on test set:0.64423
```

표준화

```
Ridge Training set score:0.80989
Ridge Test set score:0.80460
```

```
Fitting 5 folds for each of 49 candidates, totalling 245 fits
GridSearchCV max score:0.80452
GridSearchCV best parameter: {'alpha': 1, 'solver': 'auto'}
```

```
R2 Score on test set:0.80460
```

로그 변환

```
Ridge Regression Training set score:0.65730
Ridge Regression Test set score:0.64422
```

```
Fitting 5 folds for each of 49 candidates, totalling 245 fits
GridSearchCV max score:0.64674
GridSearchCV best parameter: {'alpha': 1, 'solver': 'svd'}
```

```
R2 Score on test set:0.64422
```

변화 없음

XGB(회귀, 연속 타깃 변수)

```
print('r2: {0:.5f}'.format(r2_score(y_test, pred)))
```

```
r2: 0.84832
```

```
[ ] print('GridSearchCV 최적 파라미터:', xgb_grid.best_params_)
```

```
GridSearchCV 최적 파라미터: {'colsample_bytree': 0.7, 'learning_rate': 0.05, 'max_depth': 16, 'min_child_weight': 4, 'n_estimators': 1000, 'subsample': 0.8}
```

```
[ ] model = xgb_grid.best_estimator_
    pred = model.predict(X_test)

    print('r2: {0:.5f}'.format(r2_score(y_test, pred)))
```

```
r2: 0.85388
```

LGB(회귀, 연속 타깃 변수)

```
print('r2: {0:.5f}'.format(r2_score(y_test, pred)))
```

```
[LightGBM] [Warning] Found whitespace in feature_names, replace with underlines
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of testing was 0.000793 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 941
[LightGBM] [Info] Number of data points in the train set: 4375, number of used features: 10
[LightGBM] [Info] Start training from score 700.089143
r2: 0.86481
```

```
[ ] print('GridSearchCV 최적 파라미터:', lgb_grid.best_params_)
```

```
GridSearchCV 최적 파라미터: {'colsample_bytree': 0.8, 'learning_rate': 0.1, 'max_depth': 11, 'min_child_weight': 4, 'n_estimators': 1000, 'subsample': 0.3}
```

```
[ ] from sklearn.metrics import r2_score

    model = lgb_grid.best_estimator_
    pred = model.predict(X_test)

    print('r2: {0:.5f}'.format(r2_score(y_test, pred)))
```

```
[LightGBM] [Warning] Accuracy may be bad since you didn't explicitly set num_leaves OR 2^max_depth > num_leaves. (num_leaves=31).
r2: 0.85728
```

최적 모델 선정

범주형 타깃 변수

모델명	정확도	순위
신경망(사이킷런)(표준화)	0.91933	1
신경망(tf.keras)(표준화)	0.91704	2
랜덤 포레스트(기본)	0.91659	3
SVM(표준화)	0.91608	4
그레이디언트 부스팅(기본)	0.91476	5
결정 트리(기본)	0.91248	6
라쏘(표준화)	0.90631	7
로지스틱 회귀(표준화)	0.90516	8
K-최근접이웃(표준화)	0.88551	9

연속형 타깃 변수

모델명	R2	순위
LGB(기본)	0.85728	1
XGB(기본)	0.85388	2
선형 회귀(로그 변환)	0.80993	3
릿지 회귀(로그 변환)	0.80989	4

분류모델 중에는 정확도가 가장 높은 사이킷런 신경망 모델을 골랐다.
회귀모델 중에는 R2가 가장 높은 LGB 모델을 골랐다. 여러 개의 데이터세트(기본, 표준화, 로그변환)를 사용한 경우에는 그 중 가장 높은 값을 선정했다.