2018015027 정보통계학과 김한탁

1. choose publicly avaliable data

1.1 Kaggle(www.kaggle.com)

- SeoulBikeData(Seoul Bike Sharing Demand Prediction - Mendeley Data)

2. Explain the details of data

2.1 서울시 자전거 대여 데이터 셋 (파일명: SeoulBikeData.csv)

- 2017년 12월 1일부터 2018년 11월 30일까지의 서울시의 자전거 대여와 관련된 데이터 셋
- 자전거 대여 날짜와 해당 날짜의 날씨와 관련된 변수(Temperature(℃), Humidity(%) 등)가 존재함

2.2 데이터 변수 소개

변수명	변수 설명	데이터 타입
Date	자전거 대여 날짜	object
Rented Bike Count	**대여된 자전거 수**	**int64**
Hour	자전거 대여 시각(시)	int64
Temperature(℃)	온도	float64
Humidity(%)	습도	int64
Wind speed (m/s)	풍속	float64
Visibility (10m)	가시거리(물체나 빛이 분명하게 보이는 최대 거리)	int64
Dew point Temperature(℃)	수증기를 포함하는 공기를 냉각했을 때, 응결이 시작되는 온도	float64
Solar Radiation (MJ/m2)	지표면에 도달한 태양 복사에너지	float64
Rainfall(mm)	강수량	float64
Snowfall (cm)	강설량	float64
Seasons	계절	object
Holiday	휴일 유무	object
Functioning Day	자전거 대여 서비스 운영 여부	object

*** bold체로 표시된 행은 Target에 해당**

2.3 데이터 미리보기

Date	Rented Bike Count	Hour	Temperature(℃)	Humidity(%)	Wind speed (m/s)	Visibility (10m)	Dew point temperature(℃)	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	0.0	0.0	Winter	No Holiday	Yes
01/12/2017	204	1	-5.5	38	0.8	2000	-17.6	0.0	0.0	0.0	Winter	No Holiday	Yes
01/12/2017	173	2	-6.0	39	1.0	2000	-17.7	0.0	0.0	0.0	Winter	No Holiday	Yes
01/12/2017	107	3	-6.2	40	0.9	2000	-17.6	0.0	0.0	0.0	Winter	No Holiday	Yes
01/12/2017	78	4	-6.0	36	2.3	2000	-18.6	0.0	0.0	0.0	Winter	No Holiday	Yes

3. Data preprocessing

3.1 변수명 변경(Rename variable) - Data preprocessing1

1) 변수명 확인

```
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(캜)             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(캜)   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
```

2) 변수명 변경

```
df=df.rename(columns={'Temperature(캜)':'Temperature',
                      'Humidity(%)':'Humidity',
                      'Wind speed (m/s)': 'Wind speed',
                      'Visibility (10m)': 'Visibility',
                      'Dew point temperature(캜)': 'Dew point temperature',
                      'Solar Radiation (MJ/m2)': 'Solar Radiation',
                      'Rainfall(mm)':'Rainfall',
                      'Snowfall (cm)':'Snowfall'})
```

데이터의 변수명은 Data preprocessing이나 Data manipulate를 위해, 적절한 길이나 형태로 변경하는 것이 편리할 수 있다.

3.2 Data Cleaning- Data preprocessing2

1) Dealing with missing values

```
df.isnull().sum()
```

```
Date                       0
Rented Bike Count          0
Hour                       0
Temperature                0
Humidity                   0
Wind speed                 0
Visibility                 0
Dew point temperature      0
Solar Radiation            0
Rainfall                   0
Snowfall                   0
Seasons                    0
Holiday                    0
Functioning Day            0
dtype: int64
```

2) Handling duplicate values

```
df[df.duplicated(subset=['Date', 'Hour'])]
```

Date	Rented Bike Count	Hour	Temperature	Humidity	Wind speed	Visibility	Dew point temperature	Solar Radiation	Rainfall	Snowfall	Seasons	Holiday	Functioning Day

모든 변수에서 결측치는 존재하지 않았으며, Date와 Hour을 기준으로 중복값 또한 확인되지 않았다.

3.3 Make new variable & Dropping unnessery variable - Data preprocessing3

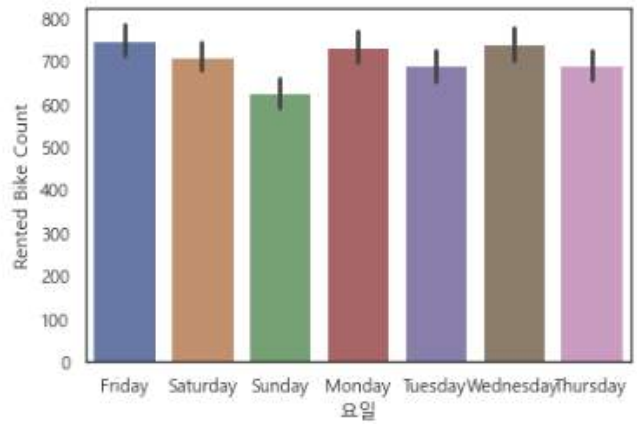
1) Date 변수를 이용해, year, month, day 변수 생성

```
df['year'] = df['Date'].str[6:10].astype('int')
df['month'] = df['Date'].str[3:5].astype('int')
df['day'] = df['Date'].str[:2].astype('int')
df['Date'] = pd.to_datetime(df[['year', 'month', 'day']])
```

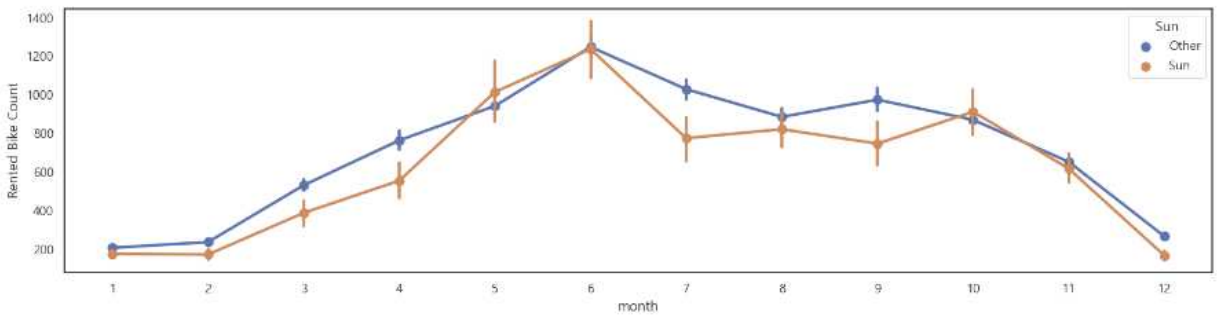
년도, 월, 일에 따른 Rented Bike Count(Target)을 확인하기 위해, year, month, day 변수 생성

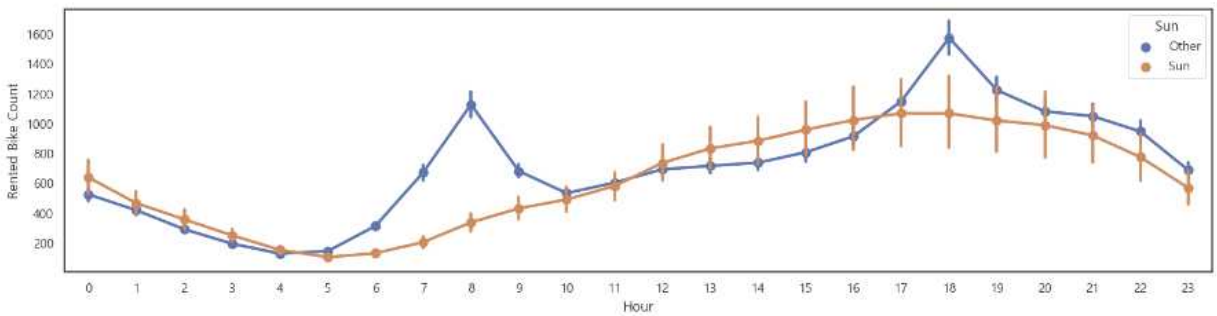
2) Sun 변수 생성 & Date 변수 제거

```
df['요일'] = df['Date'].dt.day_name()
sns.barplot(x=df['요일'], y=df['Rented Bike Count'])
plt.show()
```




```
df.loc[df['요일'] == 'Sunday', 'Sun'] = 'Sun'
df.loc[df['요일'] != 'Sunday', 'Sun'] = 'Other'
fig, axs = plt.subplots(nrows=2,ncols=1,figsize=(15,8), dpi=100)
sns.pointplot(data=df, x="month", y="Rented Bike Count",ax=axs[0],hue="Sun")
sns.pointplot(data=df, x="Hour", y="Rented Bike Count",ax=axs[1],hue="Sun")
plt.tight_layout()
```





Date 변수를 이용하여, Date에 따른 요일 변수를 생성하였고 요일에 따른 Rented Bike Count(Target)을 확인해 본 결과, Sunday의 값만 유의하게 차이가 나는 것으로 판단되었다.
이에 따라, Sunday와 나머지 요일을 구분하는 Sun 변수를 생성하였으며, month와 Hour의 변화에 따른 Sun 변수의 Rented Bike Count를 확인하였다. 이 결과 또한 유의한 것으로 생각된다.
그 후 더 이상 사용하지 않는 요일 변수를 제거하였다.

3) year 변수 제거

```
df_2017=df[df['year']==2017]
df_2018=df[df['year']==2018]
print(df_2017[['year', 'month', 'day']])
print(df_2018[['year', 'month', 'day']])
```

```
     year  month  day
0    2017     12    1
1    2017     12    1
2    2017     12    1
3    2017     12    1
4    2017     12    1
..    ...    ...  ...
739  2017     12   31
740  2017     12   31
741  2017     12   31
742  2017     12   31
743  2017     12   31

[744 rows x 3 columns]
      year  month  day
744   2018      1    1
745   2018      1    1
746   2018      1    1
747   2018      1    1
748   2018      1    1
...    ...    ...  ...
8755  2018     11   30
8756  2018     11   30
8757  2018     11   30
8758  2018     11   30
8759  2018     11   30

[8016 rows x 3 columns]
```

17년과 18년에 겹치는 month, day가 없으므로, 중요하지 않다고 판단함

3.4 Data Transformation - Data preprocessing4

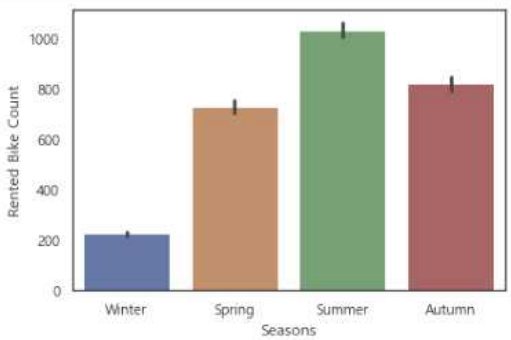
1) Category variables and their unique values

```
print('Seasons : ', df['Seasons'].unique())
print('Sun : ', df['Sun'].unique())
print('Holiday : ', df['Holiday'].unique())
print('Functioning Day : ' , df['Functioning Day'].unique())
```

```
Seasons :  ['Winter' 'Spring' 'Summer' 'Autumn']
Sun :  ['Other' 'Sun']
Holiday :  ['No Holiday' 'Holiday']
Functioning Day :  ['Yes' 'No']
```

2) Seasons - One Hot encoding

```
sns.barplot(x=df['Seasons'], y=df['Rented Bike Count'])
plt.show()
```



Seasons 변수의 범주 범위는 Winter: 12~2월, Spring: 3~5월, Summer: 6~8월, Autumn: 9~11월이다. Error bar가 서로 겹치지 않는 것을 통해, 계절에 따른 Rented Bike Count에 차이가 있는 것으로 생각된다. 따라서 One - Hot encoding을 통해, 각 범주의 독립적인 효과를 확인하는 것이 적절하다고 생각된다.

3) 나머지 Category variables(Sun, Holiday, Functioning Day) - Label encoding

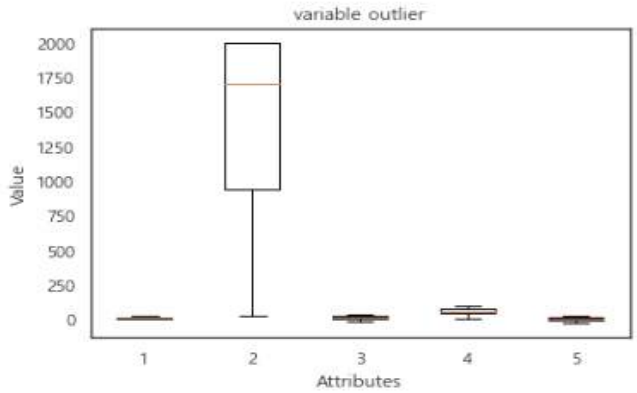
```
categories = ['Sun','Holiday','Functioning Day']
encoder = LabelEncoder()
for col in categories:
    df[col] = pd.DataFrame(encoder.fit_transform(df[col]))
df=pd.get_dummies(df, columns = ['Seasons'], drop_first=True)
```

나머지 Category variables은 범주가 2개이므로, Label encoding과 One Hot encoding 둘 다 같은 결과를 얻음. 기존 변수명을 유지하기 위해 Label encoding을 사용함

3.5 Data Normalization - Data preprocessing5

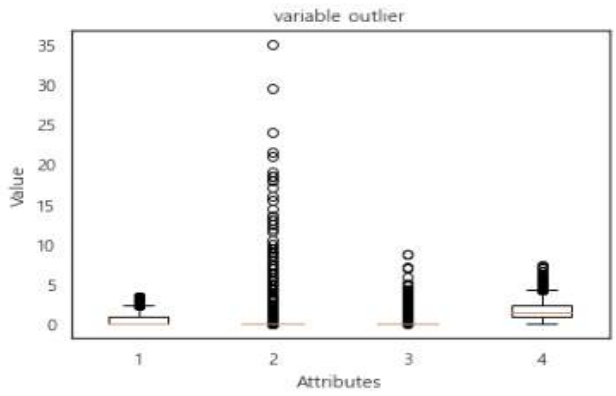
1-1) Outlier Detection - Hour, Visibility, Temperature, Humidity, Dew point temperature

```
data_to_boxplot = [df['Hour'],
                   df['Visibility'],
                   df['Temperature'],
                   df['Humidity'],
                   df['Dew point temperature']
 ]
plt.boxplot(data_to_boxplot)
plt.title('variable outlier')
plt.xlabel('Attributes')
plt.ylabel('Value')
plt.show()
```



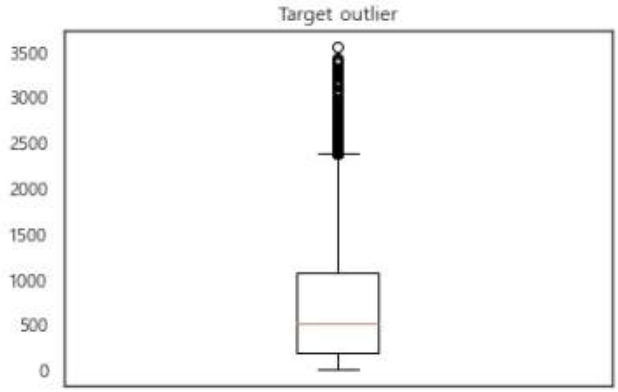
1-2) Outlier Detection - Solar Radiation, Rainfall, Snowfall, Wind speed

```
data_to_boxplot = [df['Solar Radiation'],
                   df['Rainfall'],
                   df['Snowfall'],
                   df['Wind speed']
 ]
plt.boxplot(data_to_boxplot)
plt.title('variable outlier')
plt.xlabel('Attributes')
plt.ylabel('Value')
plt.show()
```



1-3) Outlier Detection - Rented Bike Count(Target)

```
plt.boxplot(df['Rented Bike Count'])
plt.title('Target outlier')
plt.show()
```



날씨와 관련된 Solar Radiation, Rainfall, Snowfall, Wind speed 변수에서 Q3 + 1.5*IQR 보다 큰 이상치들이 확인되었으나, 사계절이 뚜렷한 우리나라 날씨를 고려하여 제거하지 않았다.

Rented Bike Count(Target)에서 또한 이상치가 발견되었지만, 퇴근 시간인 18시(Hour = 18) 부근에서 자전거 대여량이 확 늘어나는 현상을 확인할 수 있어, 마찬가지로 제거하지 않았다.

*) 실제로 Rented Bike Count(Target)의 이상치가 18시 부근에 대부분 존재함을 확인

```
q3 = np.quantile(df['Rented Bike Count'],0.75)
q1 = np.quantile(df['Rented Bike Count'],0.25)
iqr = q3 - q1
high = q3 + 1.5 * iqr
low = q1 - 1.5 * iqr
print('Q3+1.5*IQR : ', high)
print('Q1-1.5*IQR : ', low)
```

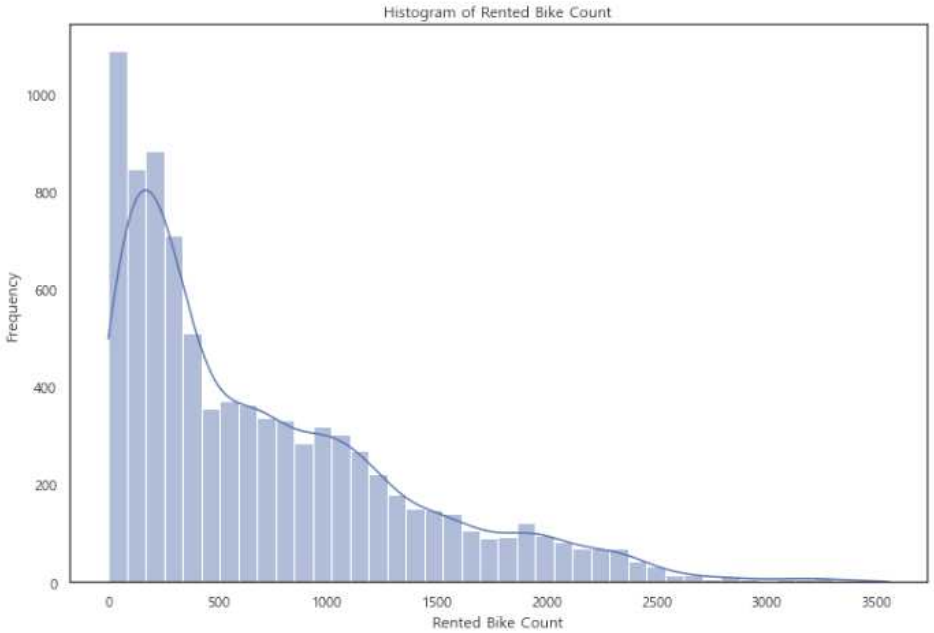
```
Q3+1.5*IQR :  2376.625
Q1-1.5*IQR :  -1120.375
```

```
df_Outlier = df[df['Rented Bike Count']>high]
df_Outlier.groupby(['month', 'day', 'Hour']).mean()[['Rented Bike Count', 'year']].head(30)
```

			Rented Bike Count	year
month	day	Hour		
4	2	18	2401.0	2018.0
	4	18	2402.0	2018.0
	9	18	2401.0	2018.0
	13	18	2404.0	2018.0
	16	18	2692.0	2018.0
	25	18	2807.0	2018.0
	26	18	2574.0	2018.0
	27	18	2577.0	2018.0
	30	18	2558.0	2018.0
5	4	18	2661.0	2018.0
	7	17	2392.0	2018.0
	9	18	3130.0	2018.0
		19	2405.0	2018.0
	11	18	2701.0	2018.0
	13	16	2379.0	2018.0
		17	2410.0	2018.0
	14	18	2906.0	2018.0
	15	18	2915.0	2018.0
	19	16	2479.0	2018.0
		18	2439.0	2018.0
	20	18	2403.0	2018.0

2-1) Rented Bike Count(Target)의 분포 확인

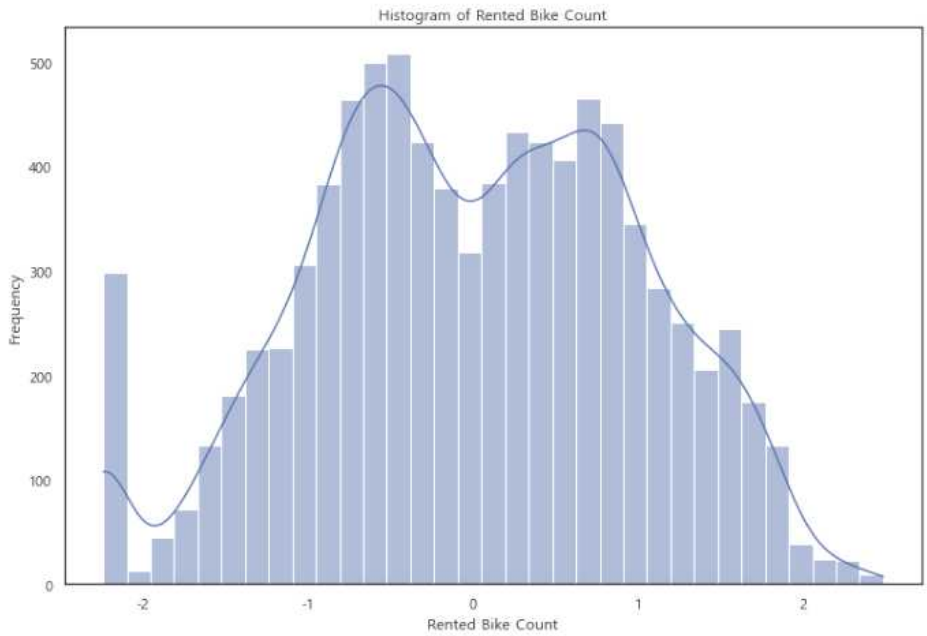
```
plt.figure(figsize=(12, 8))
sns.histplot(df['Rented Bike Count'], kde=True)
plt.title(f'Histogram of Rented Bike Count')
plt.xlabel('Rented Bike Count')
plt.ylabel('Frequency')
plt.show()
```



2-2) Rented Bike Count(Target)의 분포 -> BoxCox Transformation을 통해 정규분포로 변환

```
from sklearn.preprocessing import PowerTransformer
pt = PowerTransformer(method='yeo-johnson')
df['Rented Bike Count'] = pd.DataFrame(pt.fit_transform(df[['Rented Bike Count']]))
```

```
plt.figure(figsize=(12, 8))
sns.histplot(df['Rented Bike Count'], kde=True)
plt.title(f'Histogram of Rented Bike Count')
plt.xlabel('Rented Bike Count')
plt.ylabel('Frequency')
plt.show()
```

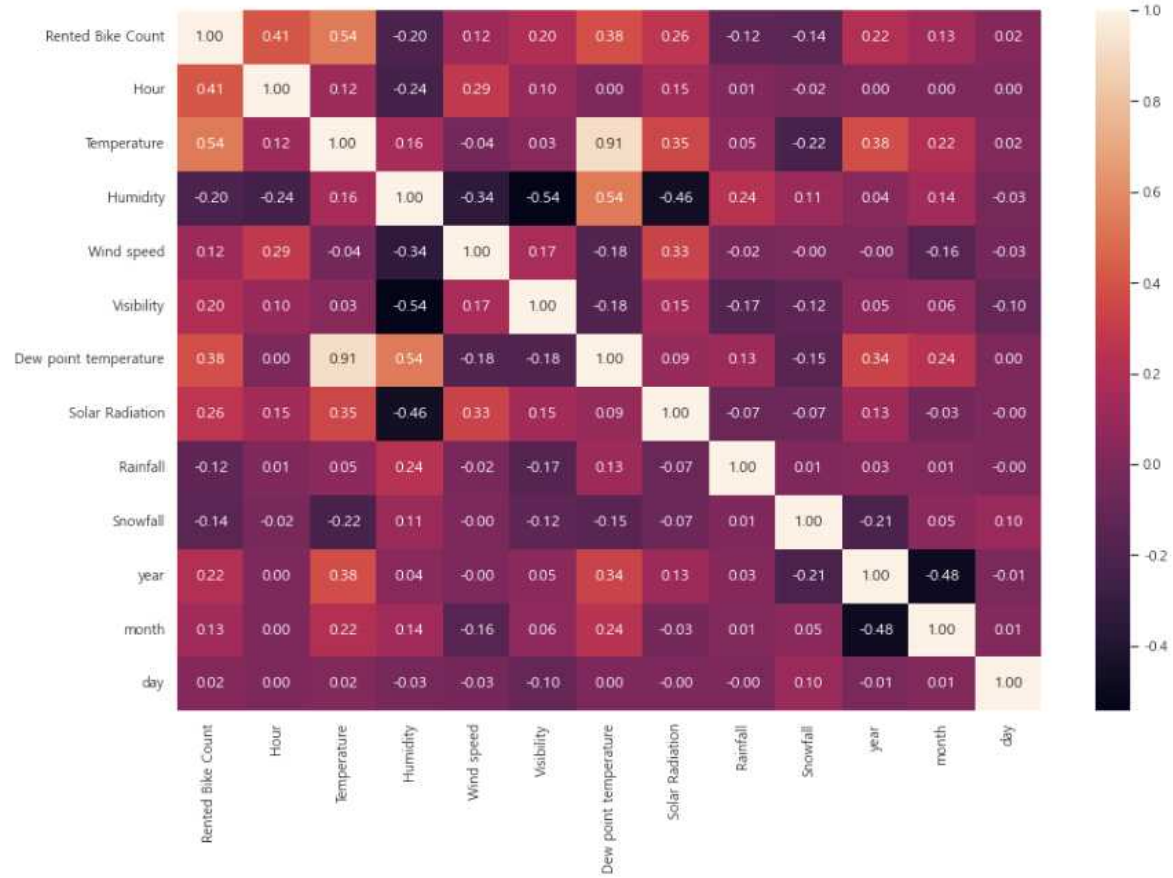


일부 model은 잔차의 정규분포를 가정하고 있기 때문에(선형 회귀), Target의 분포를 정규분포의 형태로 변환하는 것은 잔차의 정규성 및 등분산성을 만족시켜 model의 성능을 향상할 수 있다.

3.6 Feature Selection - Data preprocessing6

1) Correlation Heatmap

```
plt.figure(figsize=(15,10))
sns.heatmap(df.corr(), annot=True, fmt='.2f', annot_kws={"size": 12})
plt.show()
```



Temperature와 Dew point temperature는 다중공선성이 존재하는 것으로 생각된다. 두 변수 중 Target 변수와 상관성이 더 높은 Temperature 변수는 유지하고 Dew point temperature 변수를 제거하는 것이 적절할 것으로 생각된다.

3.7 Data preprocessing 결과

Data preprocessing	how to preprocessing	세부 사항
1. rename variable	특정 변수의 이름 변경	Temperature, Humanity 변수 등
2. Data Cleaning	Dealing with missing values	결측치는 존재하지 않음
	Handling duplicate values	중복값은 존재하지 않음
3. Make new variables & Dropping unnessery variables	make new Derived variable	year, month, day, Sun 변수
	dropping unnessery variables	Date, year 변수
4. Data Transformation	One-Hot Encoding	Seasons 변수
	Label Encoding	Sun, Holiday, Functioning Day 변수
5. Data Normalization	Outlier detection	Outlier가 존재하나, 제거하지 않았음
	Target Normalization	Rented Bike Count 변수 Box-Cox Transformation
6. Feature Selection	Correlation Heatmap	Dew point temperature 제거

4. Data Analysis

- model : Multiple Linear Regression(MLR), AdaBoost, RandomForest, Stacking
- 순차적인 Data preprocessing의 효과를 알아보고자, Data processing 과정을 Level 1~4로 나눈 뒤에 각 Level의 모델의 성능을 평가함
- 모든 Level의 모델에 사용한 데이터 셋은 8:2의 비율로, training set, test set을 분할함
- 모든 Level의 모델에 대한 평가지표는 RMSE를 사용

4.1 Level 1

- year, month, day 변수만 파생시킨 데이터 셋 이용(df1)
- Category variable encoding : Label encoding

1) df1

```
df1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8760 entries, 0 to 8759
Data columns (total 16 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   Rented Bike Count      8760 non-null   int64
 1   Hour                   8760 non-null   int64
 2   Temperature            8760 non-null   float64
 3   Humidity               8760 non-null   int64
 4   Wind speed             8760 non-null   float64
 5   Visibility             8760 non-null   int64
 6   Dew point temperature  8760 non-null   float64
 7   Solar Radiation        8760 non-null   float64
 8   Rainfall               8760 non-null   float64
 9   Snowfall               8760 non-null   float64
 10  Seasons                8760 non-null   object
 11  Holiday                8760 non-null   object
 12  Functioning Day        8760 non-null   object
 13  year                   8760 non-null   int32
 14  month                  8760 non-null   int32
 15  day                    8760 non-null   int32
dtypes: float64(6), int32(3), int64(4), object(3)
memory usage: 992.5+ KB
```

2) Data Encoding

```
categories = ['Holiday','Functioning Day', 'Seasons']
encoder = LabelEncoder()
for col in categories:
    df1[col] = pd.DataFrame(encoder.fit_transform(df1[col]))
```

3) Data Split

```
X = df1.drop(columns=['Rented Bike Count'])
y = df1['Rented Bike Count']
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size = 0.2, random_state = 0)
```

4) modeling

MLR

```
lr = LinearRegression()
lr.fit(X_train, y_train)
mlr_yhat = lr.predict(X_test)
print("R2 : ", r2_score(y_test, mlr_yhat))
print("RMSE : ", mean_squared_error(y_test, mlr_yhat, squared=False))
```

```
R2 :  0.5448908282355911
RMSE :  436.43266215552495
```

AdaBoost

```
ada = AdaBoostRegressor(n_estimators = 10, random_state=0)
ada.fit(X_train, y_train)
ada_yhat = ada.predict(X_test)
print("R2 : ", r2_score(y_test, ada_yhat))
print("RMSE : ", mean_squared_error(y_test, ada_yhat, squared=False))
```

```
R2 :  0.6516327883199484
RMSE :  381.83706887226367
```

RandomForest

```
rf = RandomForestRegressor(n_estimators = 10, random_state=0)
rf.fit(X_train, y_train)
rf_yhat = rf.predict(X_test)
print("R2 : ", r2_score(y_test, rf_yhat))
print("RMSE : ", mean_squared_error(y_test, rf_yhat, squared=False))
```

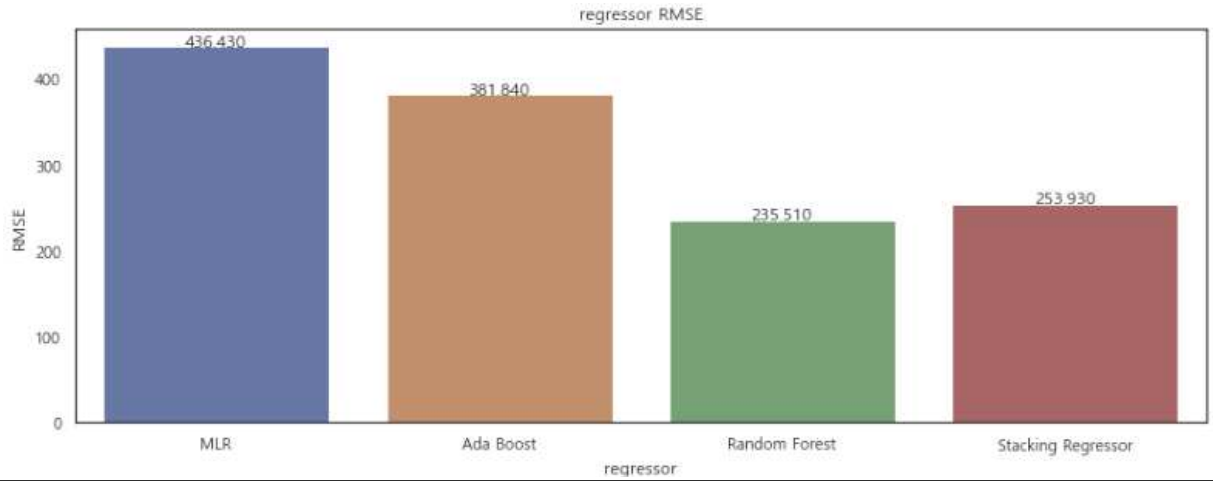
```
R2 :  0.867475231971092
RMSE :  235.5092544885905
```

Stacking

```
base_models = [
    ('lr', LinearRegression()),
    ('ada', AdaBoostRegressor(n_estimators = 10, random_state=0)),
    ('rf', RandomForestRegressor(n_estimators = 10, random_state=0))
]

stacking_reg = StackingRegressor(estimators=base_models,
                                 final_estimator=RandomForestRegressor(n_estimators = 10, random_state=0))
stacking_reg.fit(X_train, y_train)
stacking_reg_yhat = stacking_reg.predict(X_test)
print("R2 : ", r2_score(y_test, stacking_reg_yhat))
print("RMSE : ", mean_squared_error(y_test, stacking_reg_yhat, squared=False))
```

```
R2 :  0.8459284763836898
RMSE :  253.9338749970247
```

5) R2 score , RMSE by model



Random Forest 모델의 R2 score가 약 0.867로 가장 높았으며, RMSE는 가장 작은 약 235.510으로 나타났다. 반면에 MLR 모델은 R2 score는 약 0.545로 가장 낮았고, RMSE는 가장 큰 약 436.430의 값을 나타내는 것의 확인할 수 있었다.

Random Forest와 Stacking 모델은 Rented Bike Count(Target)에 대한 모델의 설명력과 예측력이 뛰어나지만, MLR과 AdaBoost 모델은 다소 떨어짐을 알 수 있다.

* Level 1 요약

1. 적용된 Data preprocessing		
- year, month, day만 파생시킨 데이터 셋 이용(df1) - Category variable encoding : Label encoding		
2. Model R2 score, RMSE		
	R2 score	RMSE
MLR	0.545	436.430
AdaBoost	0.652	381.840
RandomForest	0.867	235.510
Stacking	0.846	253.930

4.2 Level 2

기존의 데이터 셋(df)로 Data preprocessing과 함께 진행

- year, month, day 변수 생성
- Sun 변수 생성 및 Date 변수 제거
- year 변수 제거
- Dew point temperature 변수 제거(multicollinearity 제거)
- Category variable encoding
  1) Sun, Holiday, Functioning Day : Label encoding
  2) Seasons : One Hot encoding

1) df

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8760 entries, 0 to 8759
Data columns (total 18 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            8760 non-null   float64
 4   Humidity               8760 non-null   int64
 5   Wind speed             8760 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                8760 non-null   object
 13  Functioning Day        8760 non-null   object
 14  year                   8760 non-null   int32
 15  month                  8760 non-null   int32
 16  day                    8760 non-null   int32
 17  Sun                    8760 non-null   object
dtypes: datetime64[ns](1), float64(6), int32(3), int64(4), object(4)
memory usage: 1.1+ MB
```

2) Data Encoding - Sun:One Hot Encoding, 나머지:Label Encoding

```
categories = ['Sun','Holiday','Functioning Day']
encoder = LabelEncoder()
for col in categories:
    df[col] = pd.DataFrame(encoder.fit_transform(df[col]))
df=pd.get_dummies(df, columns = ['Seasons'], drop_first=True)
```

3) Data Split - Date, year, Dew point temperature 변수 제거

```
X = df.drop(columns=['Rented Bike Count', 'Date', 'year', 'Dew point temperature'])
y = df['Rented Bike Count']
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size = 0.2, random_state = 0)
```

4) modeling

MLR

```
lr = LinearRegression()
lr.fit(X_train, y_train)
mlr_yhat = lr.predict(X_test)
print("R2 : ", r2_score(y_test, mlr_yhat))
print("RMSE : ", mean_squared_error(y_test, mlr_yhat, squared=False))
```

```
R2 :  0.5497284939673281
RMSE :  434.1068949141848
```

AdaBoost

```
ada = AdaBoostRegressor(n_estimators = 10, random_state=0)
ada.fit(X_train, y_train)
ada_yhat = ada.predict(X_test)
print("R2 : ", r2_score(y_test, ada_yhat))
print("RMSE : ", mean_squared_error(y_test, ada_yhat, squared=False))
```

```
R2 :  0.6631098156234276
RMSE :  375.49454322286545
```

RandomForest

```
rf = RandomForestRegressor(n_estimators = 10, random_state=0)
rf.fit(X_train, y_train)
rf_yhat = rf.predict(X_test)
print("R2 : ", r2_score(y_test, rf_yhat))
print("RMSE : ", mean_squared_error(y_test, rf_yhat, squared=False))
```

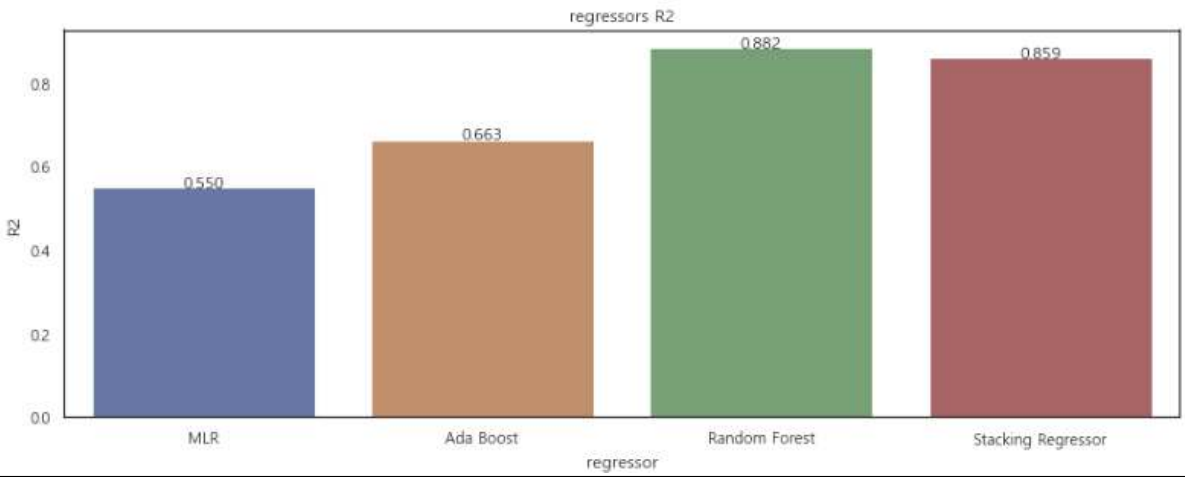
```
R2 :  0.8824322282113228
RMSE :  221.82147814810048
```

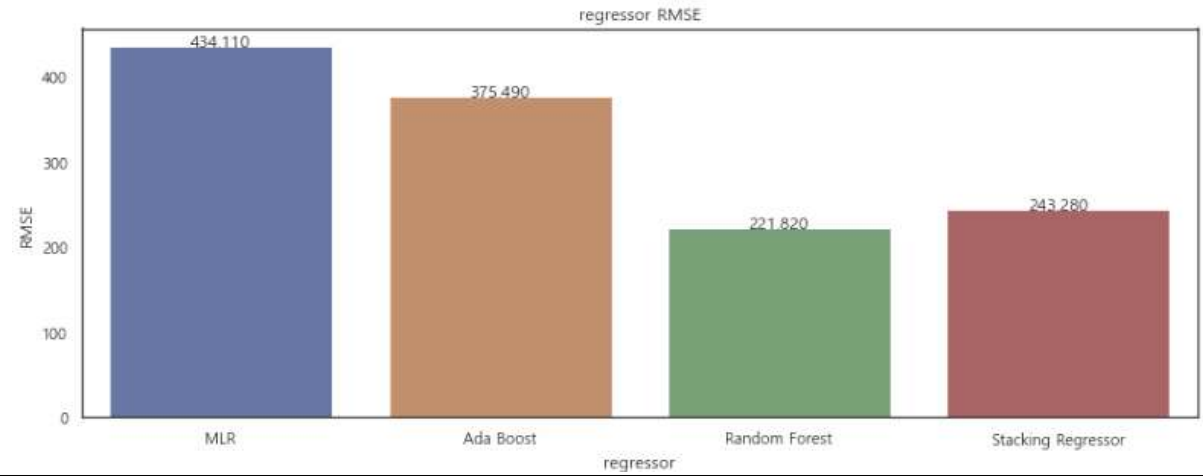
Stacking

```
base_models = [
    ('lr', LinearRegression()),
    ('ada', AdaBoostRegressor(n_estimators = 10, random_state=0)),
    ('rf', RandomForestRegressor(n_estimators = 10, random_state=0))
]

stacking_reg = StackingRegressor(estimators=base_models,
                                 final_estimator=RandomForestRegressor(n_estimators = 10, random_state=0))
stacking_reg.fit(X_train, y_train)
stacking_reg_yhat = stacking_reg.predict(X_test)
print("R2 : ", r2_score(y_test, stacking_reg_yhat))
print("RMSE : ", mean_squared_error(y_test, stacking_reg_yhat, squared=False))
```

```
R2 :  0.8585879108627145
RMSE :  243.27793484916566
```

5) R2 score , RMSE by model





Random Forest 모델의 R2 score가 약 0.882로 가장 높았으며, RMSE는 가장 작은 약 221.820으로 나타났다. 반면에 MLR 모델은 R2 score는 약 0.550으로 가장 낮았고, RMSE는 가장 큰 약 434.110의 값을 나타나는 것의 확인할 수 있었다.
Level 1과 비교하여, Level 2의 모든 모델의 R2 score가 상승했고, 또한 RMSE는 작아졌음을 알 수 있다.

* Level 2 요약

1. 적용된 Data preprocessing

- year, month, day 변수 생성
- Sun 변수 생성 및 Date 변수 제거
- year 변수 제거
- Dew point temperature 변수 제거(multicollinearity 제거)
- Category variable encoding
  1) Sun, Holiday, Functioning Day : Label encoding
  2) Seasons : One Hot encoding

2. Model R2 score, RMSE

	R2 score	RMSE
MLR	0.550	434.110
AdaBoost	0.663	375.490
RandomForest	0.882	221.820
Stacking	0.859	243.280

4.3 Level 3

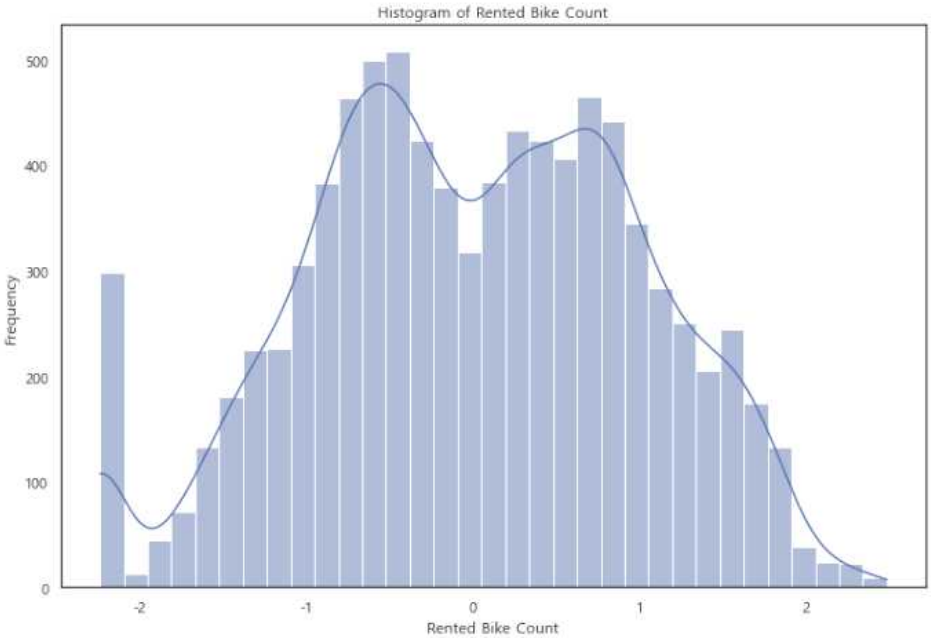
Level 2의 전처리 이후,

- Target Normalization : Box-Cox Transformation

1) Rented Bike Count(Target)의 Boxcox 변환

```
from sklearn.preprocessing import PowerTransformer
pt = PowerTransformer(method='yeo-johnson')
df['Rented Bike Count'] = pd.DataFrame(pt.fit_transform(df[['Rented Bike Count']]))
```

```
plt.figure(figsize=(12, 8))
sns.histplot(df['Rented Bike Count'], kde=True)
plt.title(f'Histogram of Rented Bike Count')
plt.xlabel('Rented Bike Count')
plt.ylabel('Frequency')
plt.show()
```



2) Data Split

```
X = df.drop(columns=['Rented Bike Count', 'Date', 'year', 'Dew point temperature'])
y = df['Rented Bike Count']
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size = 0.2, random_state = 0)
```

3) modeling

MLR

```
lr = LinearRegression()
lr.fit(X_train, y_train)
mlr_yhat = lr.predict(X_test)
print("R2 : ", r2_score(y_test, mlr_yhat))
print("RMSE : ", mean_squared_error(y_test, mlr_yhat, squared=False))
```

```
R2 :  0.6914327470898296
RMSE :  0.5620219982134196
```

AdaBoost

```
ada = AdaBoostRegressor(n_estimators = 10, random_state=0)
ada.fit(X_train, y_train)
ada_yhat = ada.predict(X_test)
print("R2 : ", r2_score(y_test, ada_yhat))
print("RMSE : ", mean_squared_error(y_test, ada_yhat, squared=False))
```

```
R2 :  0.7047930548172237
RMSE :  0.5497201837534742
```

Random Forest

```
rf = RandomForestRegressor(n_estimators = 10, random_state=0)
rf.fit(X_train, y_train)
rf_yhat = rf.predict(X_test)
print("R2 : ", r2_score(y_test, rf_yhat))
print("RMSE : ", mean_squared_error(y_test, rf_yhat, squared=False))
```

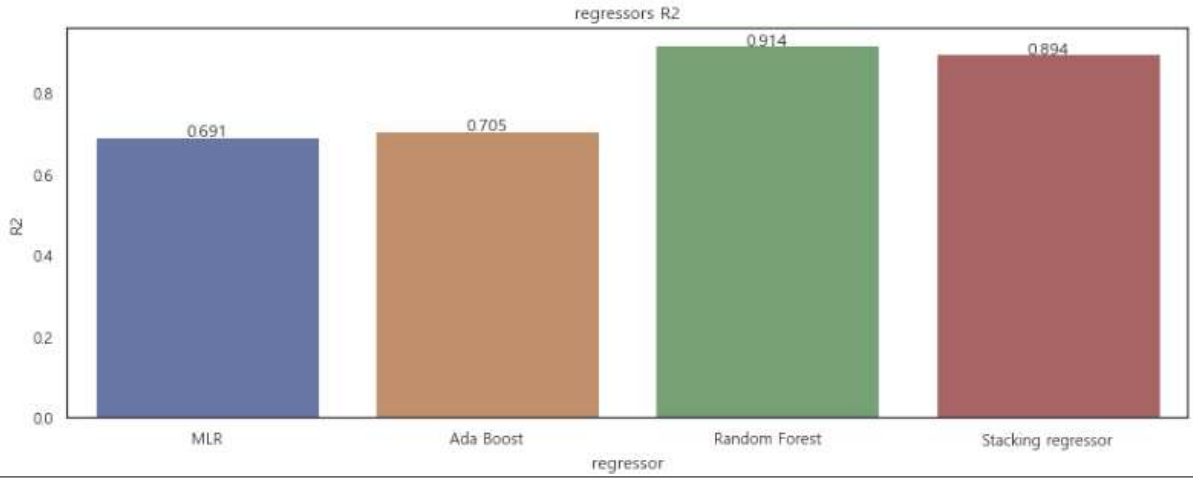
```
R2 :  0.9140583821011182
RMSE :  0.29660615586200695
```

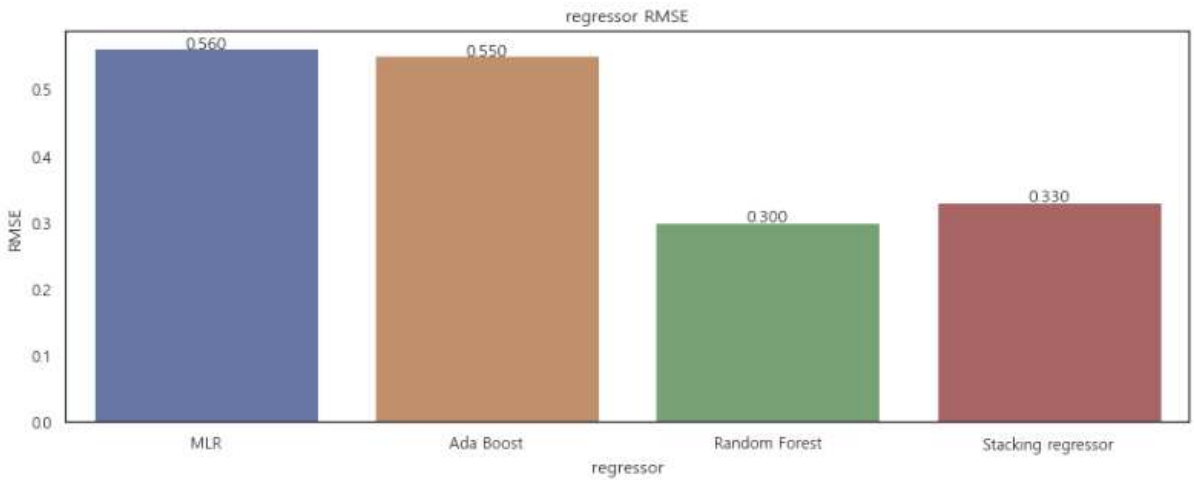
Stacking

```
base_models = [
    ('lr', LinearRegression()),
    ('ada', AdaBoostRegressor(n_estimators = 10, random_state=0)),
    ('rf', RandomForestRegressor(n_estimators = 10, random_state=0))
]

stacking_reg = StackingRegressor(estimators=base_models,
                                 final_estimator=RandomForestRegressor(n_estimators = 10, random_state=0))
stacking_reg.fit(X_train, y_train)
stacking_reg_yhat = stacking_reg.predict(X_test)
print("R2 : ", r2_score(y_test, stacking_reg_yhat))
print("RMSE : ", mean_squared_error(y_test, stacking_reg_yhat, squared=False))
```

```
R2 :  0.8936474062244495
RMSE :  0.329953272981202
```

4) R2 score , RMSE by model





Target인 Rental Bike Count의 BoxCox 변환으로 인해, 모든 모델의 R2 score가 향상되었음을 확인할 수 있다. 여전히 Random Forest의 R2 score가 약 0.914로 모델이 가장 뛰어난 설명력을 나타냈지만, R2 score가 큰 폭으로 향상된 것은 MLR 모델임을 알 수 있다.

이것은 Target의 BoxCox 변환으로 인한 선형 회귀의 가정(잔차의 정규성 혹은 등분산성 충족)을 만족한 것이 원인이 된 것으로 생각된다. Target의 변환으로 단위가 바뀌었기 때문에, RMSE가 이전 Level에 비해 확연히 줄어들었음에 주의해야한다.

* Level 3 요약

1. 적용된 Data preprocessing		
Level 2의 전처리 이후, - Target Normalization : Box-Cox Transformation		
2. Model R2 score, RMSE		
	R2 score	RMSE
MLR	0.691	0.560
AdaBoost	0.705	0.550
RandomForest	0.914	0.300
Stacking	0.894	0.330

4.4 Level 4

Level 3의 전처리 이후,

- Optuna 라이브러리를 이용해, AdaBoost, RandomForest의 Hyperparameter 최적화

라이브러리	모형	하이퍼파라미터	최적화	범위
Optuna	AdaBoost	base_estimator	Decision Tree Regressor	
		max_depth (base_estimator 해당)	18	(3, 20)
		n_estimators	91	(1, 100)
		learning_rate	약 1.6293826	(1.0, 2.0)
	Random Forest	n_estimator	236	(50, 300)
		max_depth	18	(3, 20)
		min_samples_split	2	(2, 20)
		min_samples_leaf	1	

Hyperparameter가 최적화된 모델의 결과(AdaBoost, RandomForest, Stacking)

1) Modeling

AdaBoost

```
ada = AdaBoostRegressor(
        base_estimator=DecisionTreeRegressor(max_depth=18),
        n_estimators=91,
        learning_rate=1.6293826188251432,
        random_state=0
)
ada.fit(X_train, y_train)
ada_yhat = ada.predict(X_test)
print("R2 : ", r2_score(y_test, ada_yhat))
print("RMSE : ", mean_squared_error(y_test, ada_yhat, squared=False))
```

```
R2 :  0.9250915991555909
RMSE :  0.2769132013979681
```

RandomForest

```
rf = RandomForestRegressor(
        n_estimators=236,
        max_depth=18,
        min_samples_split=2,
        min_samples_leaf=1,
        random_state=0
)
rf.fit(X_train, y_train)
rf_yhat = rf.predict(X_test)
print("R2 : ", r2_score(y_test, rf_yhat))
print("RMSE : ", mean_squared_error(y_test, rf_yhat, squared=False))
```

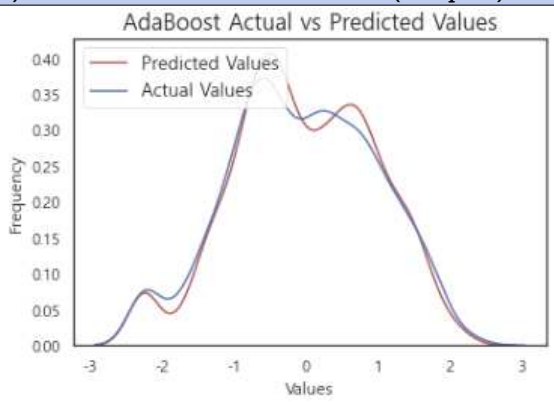
```
R2 :  0.9255055338652467
RMSE :  0.27614704718226263
```

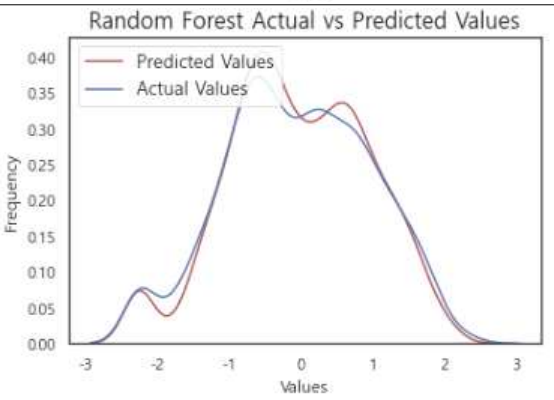
Stacking

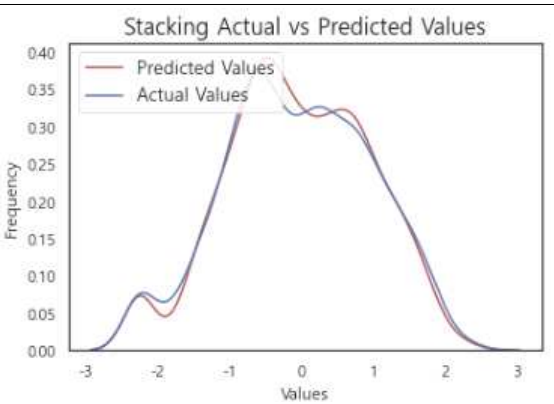
```
base_models = [
    ('lr', LinearRegression()),
    ('ada', AdaBoostRegressor(
        base_estimator=DecisionTreeRegressor(max_depth=18),
        n_estimators=91,
        learning_rate=1.6293826188251432,
        random_state=0
)),
    ('rf', RandomForestRegressor(
        n_estimators=236,
        max_depth=18,
        min_samples_split=2,
        min_samples_leaf=1,
        random_state=0
))
]

stacking_reg = StackingRegressor(estimators=base_models,
                                 final_estimator=RandomForestRegressor(n_estimators = 10, random_state=0))
stacking_reg.fit(X_train, y_train)
stacking_reg_yhat = stacking_reg.predict(X_test)
print("R2 : ", r2_score(y_test, stacking_reg_yhat))
print("RMSE : ", mean_squared_error(y_test, stacking_reg_yhat, squared=False))
```

```
R2 :  0.9112232613720862
RMSE :  0.3014588172158681
```

2) Actual vs Predicted Value (distplot)







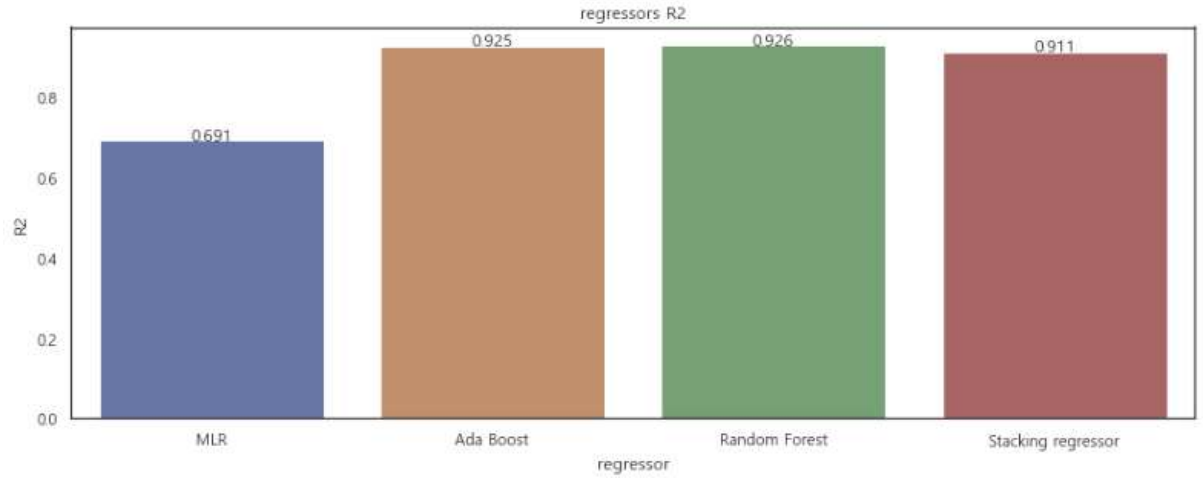
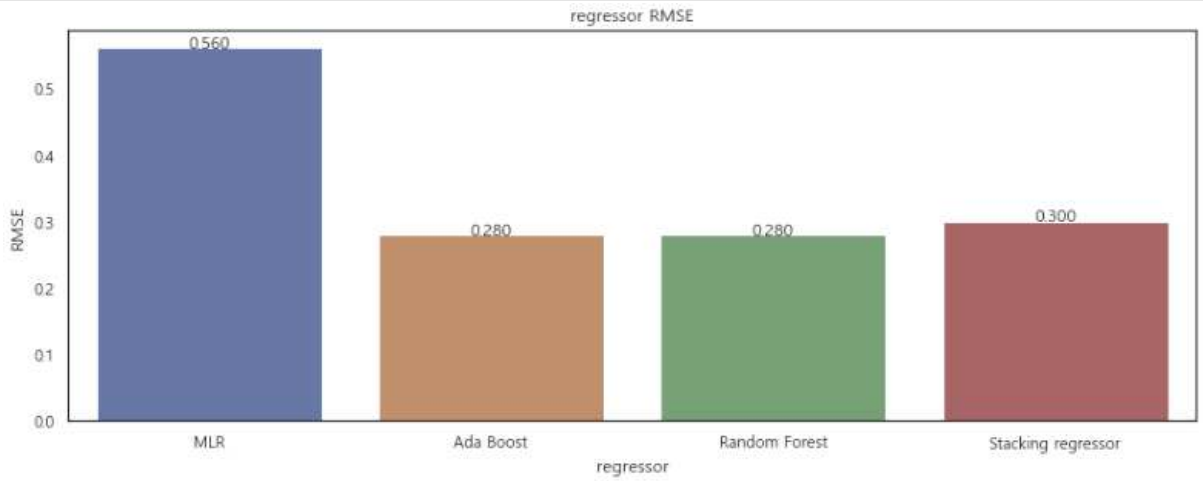
AdaBoost, RandomForest, Stacking 모델의 실제값(y_test)과 예측값(yhat)을 dist plot으로 표현한 것이다.

세 모델 모두 실제값과 매우 근접하게 값을 예측하였지만, 공통으로 -2 부근, (-1, 1) 범위에서 실제값과 유의미한 오차가 존재하는 것으로 보인다.

이를 통해, 세 모델 모두 실제값보다 (-1, 0)의 값으로 예측한 경우가 많았고, -2 부근, (0, 1)의 범위에 대해서는 예측이 실제보다 적었음을 알 수 있다.

실제값과 예측값의 R2 score가 높게 나타나더라도, 실제값에 대한 예측 성능은 좋지 않을 수 있다. (ex 실제값(y)과 예측값(x)의 그래프에서 직선이 45도 부근이 아닌 경우)

하지만 dist plot의 결과를 통해, 그러한 문제점은 발견되지 않는 것으로 생각된다.

3) R2 score , RMSE by model


Hyperparmeter를 조정한 AdaBoost, RandomForest, Stacking 모델의 R2 score가 향상되었음을 확인할 수 있다. 특히 AdaBoost의 R2 score는 기존의 약 0.705에서 약 0.925까지 확연히 성능이 좋아진 것을 통해 기존의 AdaBoost 모델의 hyperparameter가 적절하지 않았다는 것을 알 수 있다. 그리고 RMSE 또한 Level 3의 RMSE에 비해 모두 감소하였음을 확인하였다.

* Level 4 요약

1. 적용된 Data preprocessing		
Level 3의 전처리 이후, - Optuna 라이브러리를 이용해, AdaBoost, RandomForest의 Hyperparameter 최적화		
2. Hyperparameter 최적화		
모형	**하이퍼파라미터**	**최적화**
AdaBoost	base_estimator	DecisionTree Regressor(최적화 대상 x)
	max_depth (base_estimator 해당)	18
	n_estimators	91
	learning_rate	약 1.6293826
Random Forest	n_estimator	236
	max_depth	18
	min_samples_split	2
	min_samples_leaf	1(최적화 대상 x)

3. Model R2 score, RMSE		
	R2 score	RMSE
MLR	0.691	0.560
AdaBoost	0.925	0.280
RandomForest	0.925	0.280
Stacking	0.911	0.300

5. 결론

모든 Data preprocessing 과정과 hyperparameter 조정을 마친 AdaBoost, RandomForest 두 모델의 R2 score가 약 0.925로 가장 높은 성능을 가지는 것으로 나타났다.

5.1 Data preprocessing level의 구분

Data Processing Level			
Level 1	Level 2	Level 3	Level 4
- year, month, day 변수 생성 - Category variable encoding : Label encoding	- year, month, day 변수 생성 - Sun 변수 생성 및 Date 변수 제거 - year 변수 제거 - Dew point temperature 변수 제거 (multicollinearity 제거) - Category variable encoding 1) Sun, Holiday, Functioning Day : Label encoding 2) Seasons : One hot encoding	Level 2 동일	Level 2 동일
		- Target Normalization : Box-Cox Transformation	Level 3 동일
			- Hyper parameter 최적화

5.2 Data preprocessing level에 따른 모델의 성능

		R2 score			
		MLR	AdaBoost	RandomForest	Stacking
Data Processing	Level 1	0.545	0.652	0.867	0.846
	Level 2	0.550	0.663	0.882	0.859
	↓	Box-Cox Transformation			
	Level 3	0.691	0.705	0.914	0.894
	Level 4	Level 3와 동일	0.925	0.925	0.911