데이터 분석과 머신러닝/딥러닝을 활용한 다음날 장애인 콜택시 평균 대기시간 예측

김영우

- 대주제와 데이터 설명

대주제 : 다음날 장애인 콜택시 평균 대기시간을 예측하는 머신러닝/딥러닝 모델을 만드세요.

[데이터 설명]

open_data.csv : 2015-01-01 ~ 2022-12-31 까지의 서울 장애인 콜택시 운행 정보
weather.csv : 2012-01-01 ~ 2022-12-31까지의 날씨 정보

두 csv 파일을 통합한 후 마지막 3개월 데이터는 평가용으로 활용!!

- Index

1 : open_data.csv, weather.csv를 이용해 전처리 작업

2 : 통합한 Tablour 데이터를 이용해 탐색적 데이터 분석

3 : 통합한 Tablour 데이터를 이용해 머신러닝/딥러닝 모델 작업

open_data.csv, weather.csv를 이용해 전처리 작업

call_taxi.csv 파일은 2015년 ~ 2022년까지 장애인 콜택시에 대한 상세 정보를 보여준다.
weather.csv 파일은 2012년 ~ 2022년까지 날씨 정보를 보여준다.

[3]:
```
# read_csv를 활용
call_taxi = pd.read_csv(file1)
weather = pd.read_csv(file2)
```

[4]:
```
call_taxi.head(1)
```

[4]:

	기준일	차량운행	접수건	탑승건	평균대기시간	평균요금	평균승차거리
0	2015-01-01	213	1023	924	23.2	2427	10764

[5]:
```
weather.head(1)
```

[5]:

	Date	temp_max	temp_min	rain(mm)	humidity_max(%)	humidity_min(%)	sunshine(MJ/m2)
0	2012-01-01	0.4	-6.6	0.0	77.0	45.0	4.9

두 파일 결측치가 모두 없어서 결측치 작업은 거의 없다고 봐야 한다.

```
#전체 데이터의 행,열 개수 확인
call_taxi.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2922 entries, 0 to 2921
Data columns (total 7 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   기준일     2922 non-null   datetime64[ns]
 1   차량운행    2922 non-null   int64
 2   접수건     2922 non-null   int64
 3   탑승건     2922 non-null   int64
 4   평균대기시간  2922 non-null   float64
 5   평균요금    2922 non-null   int64
 6   평균승차거리  2922 non-null   int64
dtypes: datetime64[ns](1), float64(1), int64(5)
memory usage: 159.9 KB
```

[13]:
```
#전체 데이터의 행,열 개수 확인
weather.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4018 entries, 0 to 4017
Data columns (total 7 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   Date             4018 non-null   datetime64[ns]
 1   temp_max         4018 non-null   float64
 2   temp_min         4018 non-null   float64
 3   rain(mm)         4018 non-null   float64
 4   humidity_max(%)  4018 non-null   float64
 5   humidity_min(%)  4018 non-null   float64
 6   sunshine(MJ/m2)  4018 non-null   float64
dtypes: datetime64[ns](1), float64(6)
memory usage: 219.9 KB
```

- 날씨 데이터는 다음날 예보 데이터로 간주해야 한다는 Rule이 있었다.

Ex) 2020년 12월 23일 날씨 데이터는 다음날인 12월 24일 날씨 데이터로 간주해야 한다.

아래 그림은 시간에 따라 다음날의 날씨 데이터로 바꾸는 작업이다.

[19]:
```
# 날씨 데이터를 다음 날에 대한 예보 데이터로 간주한다.
# ex) 2020년 12월 23일이면 2020년 12월 24일 예보 데이터로 간주한다.
for column in list(weather.columns[1::1]):
    weather[column] = weather[column].shift(-1)
```

[20]:
```
# 2022년 12월 31일에 대한 'temp_max', 'temp_min', 'rain(mm)', 'humidity_max(%)', 'humidity_min(%)','sunshine(MJ/m2)' 결측치 처리
weather.loc[weather['Date'] == '2022-12-31', ['temp_max', 'temp_min', 'rain(mm)', 'humidity_max(%)', 'humidity_min(%)', 'sunshine(MJ/m2)']] = [-4.4, -4.4,
```

[22]:
```
weather.head(5)
```

[22]:

	Date	temp_max	temp_min	rain(mm)	humidity_max(%)	humidity_min(%)	sunshine(MJ/m2)
0	2012-01-01	-1.2	-8.3	0.0	80.0	48.0	6.16
1	2012-01-02	-0.4	-6.6	0.4	86.0	45.0	4.46
2	2012-01-03	-4.6	-9.5	0.0	66.0	38.0	8.05
3	2012-01-04	-1.4	-9.6	0.0	71.0	28.0	9.14
4	2012-01-05	1.3	-4.8	0.0	55.0	27.0	8.57

[21]:
```
weather.tail(1)
```

[21]:

	Date	temp_max	temp_min	rain(mm)	humidity_max(%)	humidity_min(%)	sunshine(MJ/m2)
4017	2022-12-31	-4.4	-4.4	0.0	66.0	66.0	0.0

Y(Target) 만들기

다음날의 평균 대기시간을 가져와서 새로운 변수(Next_AVG_Waiting_Time)를 추가한다.

[23]:
```
call_taxi['Next_AVG_Waiting_Time'] = call_taxi['Avg_Waiting_Time'].shift(-1)
```

[24]:
```
# 2022년 12월 31일에 대한 'Next_AVG_Waiting_Time'에 대한 결측치를 처리한다.
call_taxi.loc[call_taxi['Date'] == '2022-12-31', ['Next_AVG_Waiting_Time']] = 33.7
```

[25]:
```
call_taxi.head()
```

[25]:

	Date	Car_Cnt	Request_Cnt	Ride_Cnt	Avg_Waiting_Time	Avg_Fare	Avg_Distance	WeekDay	Month	Year	Next_AVG_Waiting_Time
0	2015-01-01	213	1023	924	23.2	2427	10764	목	1	2015	17.2
1	2015-01-02	420	3158	2839	17.2	2216	8611	금	1	2015	26.2
2	2015-01-03	209	1648	1514	26.2	2377	10198	토	1	2015	24.5
3	2015-01-04	196	1646	1526	24.5	2431	10955	일	1	2015	26.2
4	2015-01-05	421	4250	3730	26.2	2214	8663	월	1	2015	23.6

[26]:
```
call_taxi.tail()
```

[26]:

	Date	Car_Cnt	Request_Cnt	Ride_Cnt	Avg_Waiting_Time	Avg_Fare	Avg_Distance	WeekDay	Month	Year	Next_AVG_Waiting_Time
2917	2022-12-27	669	5635	4654	44.4	2198	8178	화	12	2022	44.8
2918	2022-12-28	607	5654	4648	44.8	2161	7882	수	12	2022	52.5
2919	2022-12-29	581	5250	4247	52.5	2229	8433	목	12	2022	38.3
2920	2022-12-30	600	5293	4200	38.3	2183	8155	금	12	2022	33.7
2921	2022-12-31	263	2167	1806	33.7	2318	9435	토	12	2022	33.7

두 데이터 통합하기

Merge를 통해 두 데이터를 통합했다.

(2) 날씨 데이터 붙이기

- merge를 활용합니다. 기준은 운행정보 입니다.

27]:
```
df = pd.merge(call_taxi, weather, how='inner', on='Date')
```

29]:
```
df.head()
```

29]:

	Date	Car_Cnt	Request_Cnt	Ride_Cnt	Avg_Waiting_Time	Avg_Fare	Avg_Distance	WeekDay	Month	Year	Next_AVG_Waiting_Time	temp_max	temp_min	rain(mm)	humidity_max(%)	humidity_min(%)	sunshine(MJ/m2)
0	2015-01-01	213	1023	924	23.2	2427	10764	목	1	2015	17.2	-2.0	-8.9	0.0	63.0	28.0	9.07
1	2015-01-02	420	3158	2839	17.2	2216	8611	금	1	2015	26.2	2.4	-9.2	0.0	73.0	37.0	8.66
2	2015-01-03	209	1648	1514	26.2	2377	10198	토	1	2015	24.5	8.2	0.2	0.0	89.0	58.0	5.32
3	2015-01-04	196	1646	1526	24.5	2431	10955	일	1	2015	26.2	7.9	-0.9	0.0	95.0	52.0	6.48
4	2015-01-05	421	4250	3730	26.2	2214	8663	월	1	2015	23.6	4.1	-7.4	3.4	98.0	29.0	10.47

- **Feature Engineering**

강수량 정도 변수 추가
기준은 인터넷에 떠도는 한국 강수량 정도를 참고했다.

[30]:
```python
# 'Rain_Value'(강수량 정도)에 대한 변수 추가

# 없는 강수량 : 0.0이면 '비가 오지않음'으로 분류
# 적은 강수량: 보통 5mm 미만의 강수량을 '적은 강수량'으로 분류합니다.
# 중간의 강수량: 5mm 이상 20mm 미만의 강수량을 '중간의 강수량'으로 분류합니다.
# 많은 강수량: 20mm 이상의 강수량
def rain_function(rain):
    if 0.1 <= rain < 5.0:
        return '적은 강수량'
    elif 5.0 <= rain < 20.0:
        return '중간 강수량'
    elif rain >= 20.0:
        return '많은 강수량'
    else:
        return '비가 오지않음'
df['Rain_Value'] = df['rain(mm)'].apply(rain_function)
```

- **Feature Engineering**

일사량 정도 변수 추가
기준은 인터넷에 떠도는 한국 일사량 정도를 참고했다.

[42]:
```
# 'Sunshine_Value'(일사량 정도)에 대한 변수 추가

# 낮은 일사량: 0 ~ 10 MJ/m^2
# 보통의 일사량: 10 ~ 20 MJ/m^2
# 높은 일사량: 20 MJ/m^2 이상
def sunshine_function(sunshine):
    if 0.0 <= sunshine < 10.0:
        return '낮은 일사량'
    elif 10.0 <= sunshine < 20.0:
        return '보통 일사량'
    elif sunshine >= 20.0:
        return '높은 일사량'
df['Sunshine_Value'] = df['sunshine(MJ/m2)'].apply(sunshine_function)
```

- **Feature Engineering**

계절 변수 추가
기준은 인터넷에 떠도는 한국 계절 정도를 참고했다.

[43]:
```
# 'Season'(계절) 변수 추가

# 봄 (Spring): 3월, 4월, 5월
# 여름 (Summer): 6월, 7월, 8월
# 가을 (Autumn): 9월, 10월, 11월
# 겨울 (Winter): 12월, 1월, 2월

def season_function(month):
    if 3 <= month <= 5:
        return '봄'
    elif 6 <= month <= 8:
        return '여름'
    elif 9 <= month <= 11:
        return '가을'
    else:
        return '겨울'
df['Season'] = df['Month'].apply(season_function)
```

- **Feature Engineering**

탑승률 변수 추가
Ride_Cnt와 Request_Cnt를 이용해 탑승률을 구하여 새로운 변수를 추가했다.

```
# 'Riding_Rate' (탑승률) 변수 추가
df['Riding_Rate'] = round(df['Ride_Cnt'] / df['Request_Cnt'], 4)
```

- **Feature Engineering**

‘쉬는 날인지 아닌지 여부’를 판단하는 ‘work_rest’ 변수 추가

주말과 공휴일을 쉬는 날로 판단

아래 그림은 일단 ‘토’, ‘일 ' 주말을 쉬는 날로 판단하고 있다.

[45]:
```
# work_rest('쉬는날인지 아닌지 여부') 변수 추가
# 쉬는 날 : 주말 + 공휴일
# 나머지 일하는 날

# 일단 주말은 쉬는 날이니 주말은 '쉬는 날'로 대치한다.
def work_rest_function(weekday):
    if weekday in ['토', '일']:
        return '쉬는날'
    else:
        return '일하는 날'
df['work_rest'] = df['WeekDay'].apply(work_rest_function)
```

- **Feature Engineering**

한국 공휴일을 가져와서 날짜가 포함된다면 쉬는날로 판단하고 있다.

8]:
```
from workalendar.asia import SouthKorea
cal = SouthKorea()
result = []
for year in [2015, 2016, 2017, 2018, 2019, 2020, 2021, 2022]:
    for yy in cal.holidays(year):
        result.append(yy[0])
```

9]:
```
# 공휴일도 쉬는 날 처리하기
df.loc[df['Date'].isin(result), 'work_rest'] = '쉬는날'
```

- **Feature Engineering**

최근 7일동안의 'Avg_Waiting_Time'의 평균을 구해서 새로운 변수로 추가했다.

3) 7일 이동평균 대기시간

- rolling().mean() 사용

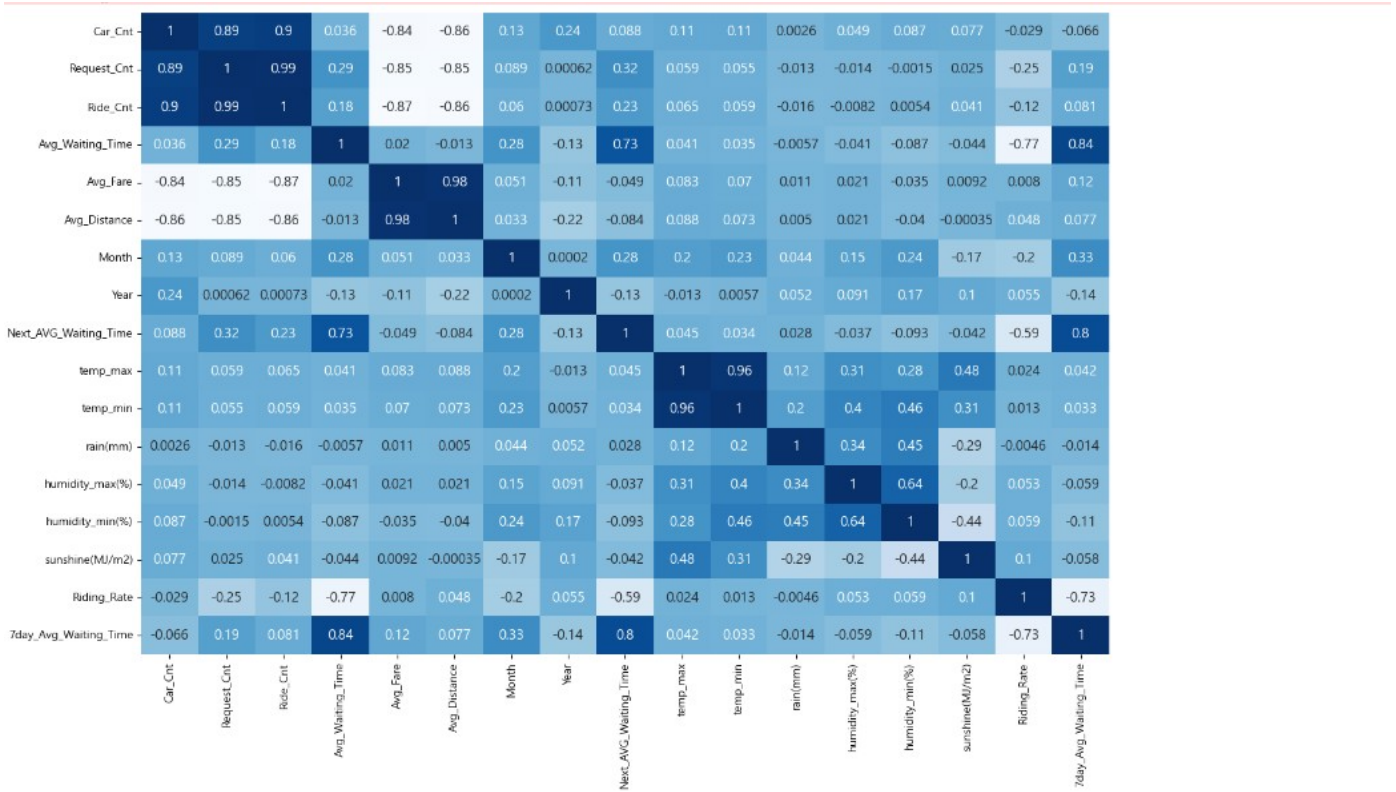
51]:
```
df['7day_Avg_Waiting_Time'] = df['Avg_Waiting_Time'].rolling(window=7, min_periods=1).mean().round(1)
```

통합한 Tablour 데이터를 이용해 탐색적 데이터 분석

- 이변량 분석에 앞서서 변수 간에 상관계수 확인

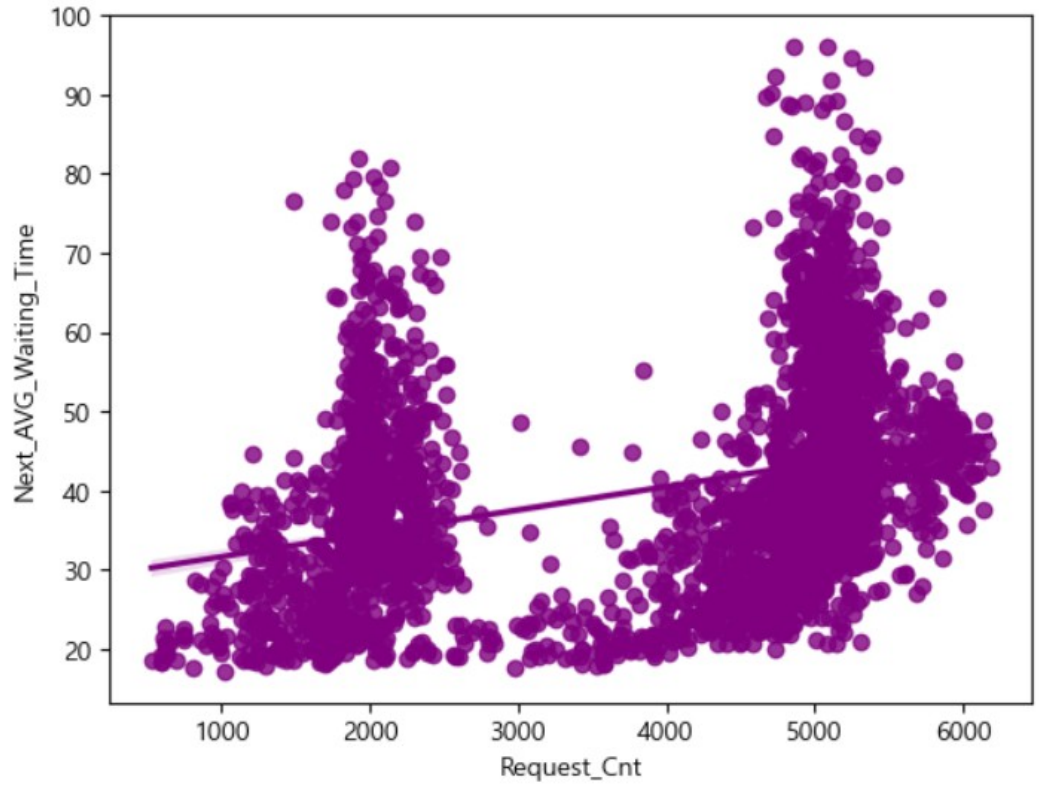
숫자형(x축) – 숫자형(y축) 간 상관계수(피어슨)를 전체적으로 확인한다.
상관계수가 높다고 판단되는 변수는 유지하고, 그렇지 않다고 판단되는 변수는 과감하게 삭제하는 방향으로 결정

범주형(x축) – 숫자형(y축)은 지원하지 않기 때문에 따로 평가지표를 봐야 한다!



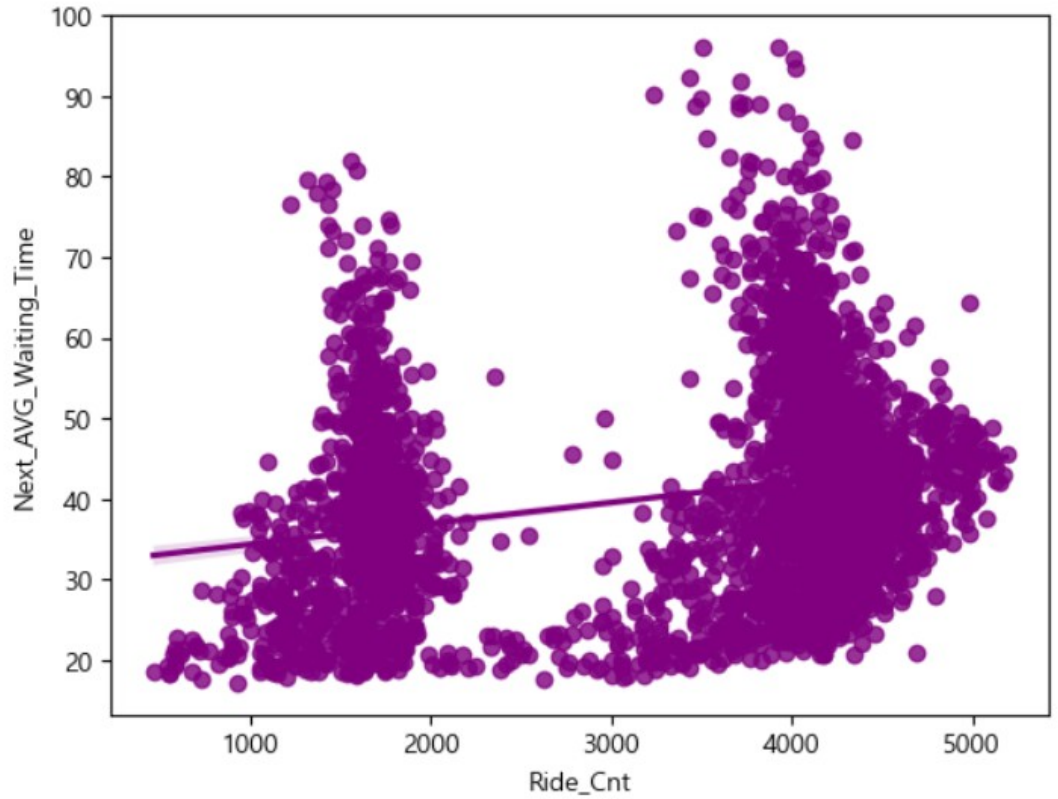
- heatmap은 x축이 숫자형, y축 숫자형일때 지표가 의미가 있다.

- 이변량 분석 (Request_Cnt, Next_AVG_Waiting_Time)



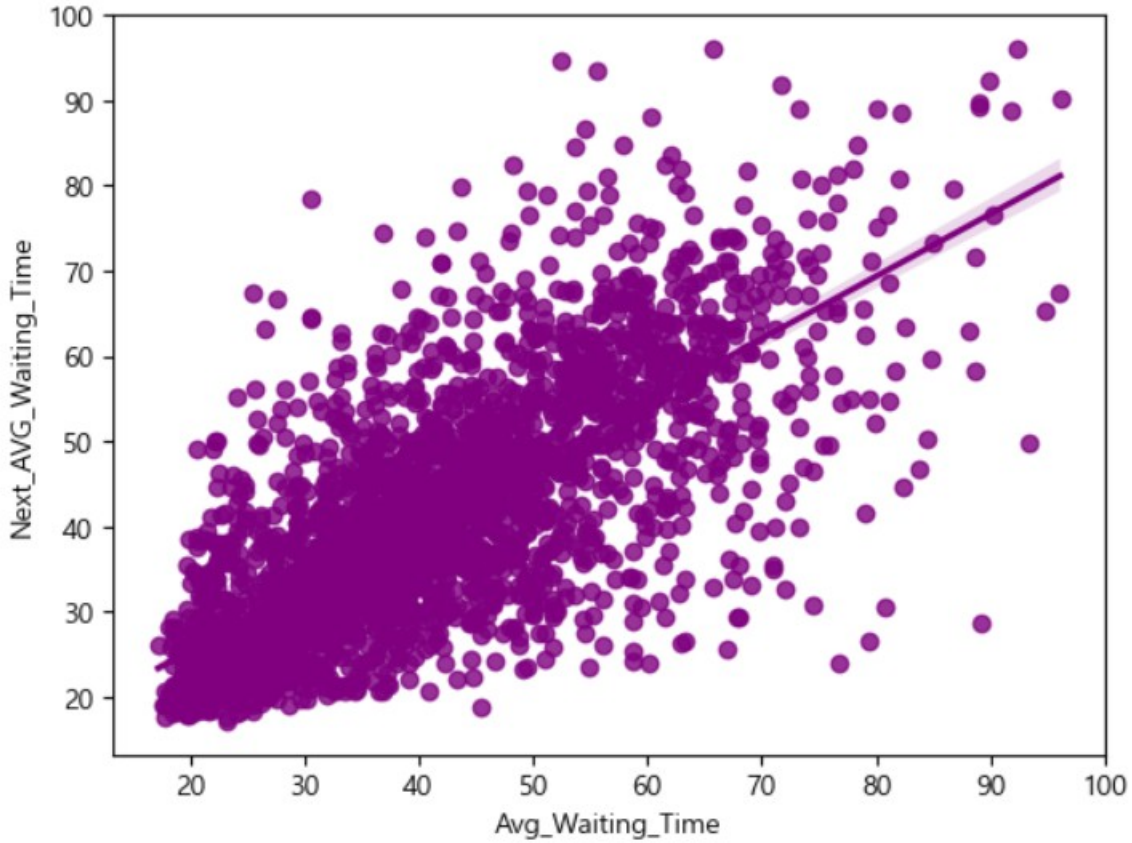

```
PearsonRResult(statistic=0.3162822498236501, pvalue=6.880088152034076e-69)
```

- 이변량 분석 (Ride_Cnt, Next_AVG_Waiting_Time)




```
PearsonRResult(statistic=0.2292532723587576, pvalue=3.7473829085525984e-36)
```

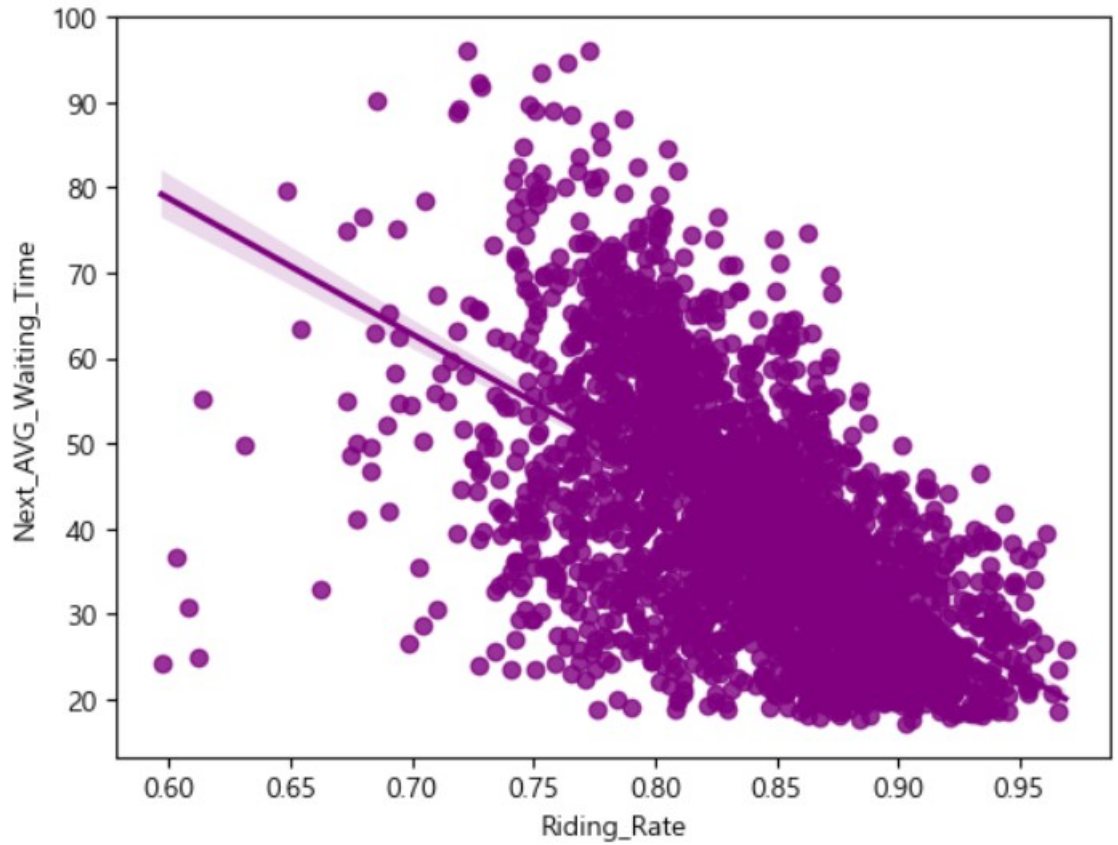
- 이변량 분석 (AVG_Waiting_Time , Next_AVG_Waiting_Time)




```
PearsonRResult(statistic=0.7323194257982135, pvalue=0.0)
```

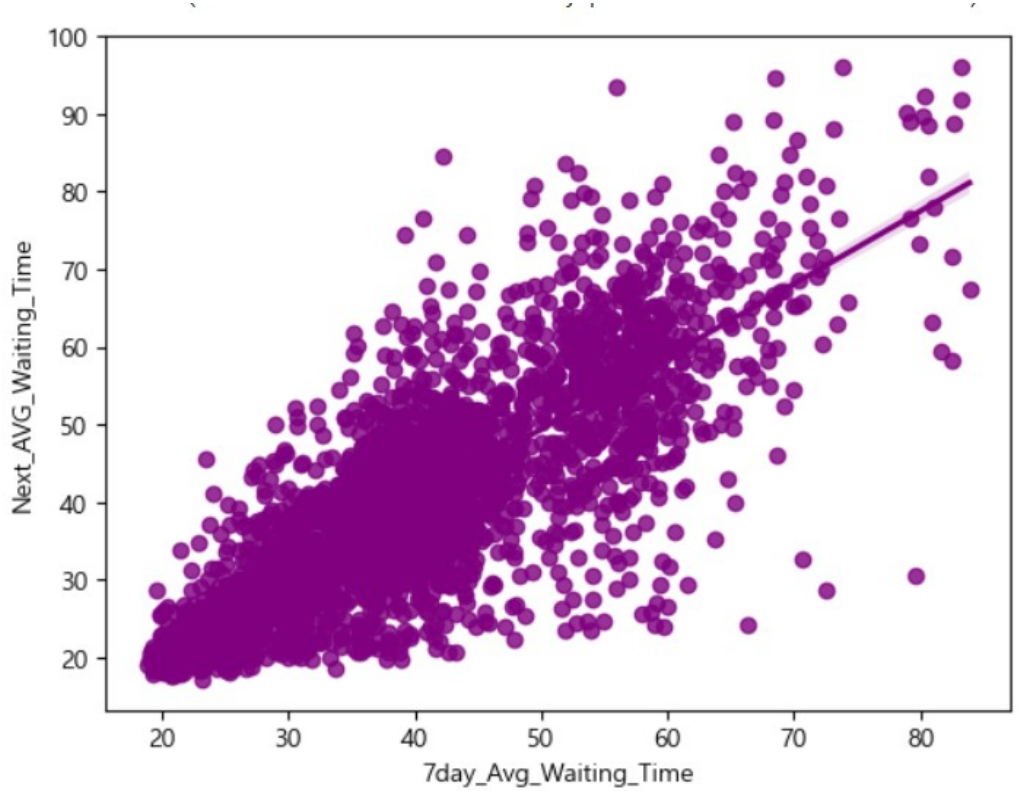
- 이변량 분석 (Riding_Rate , Next_AVG_Waiting_Time)

탑승률이 낮을수록, 택시 기사가 일단 기다리고 보니까
전체적으로 평균 대기시간이 길어지지 않을까 하는 생각이 든다.




```
PearsonRResult(statistic=-0.588740489690797, pvalue=3.512594236575448e-272)
```

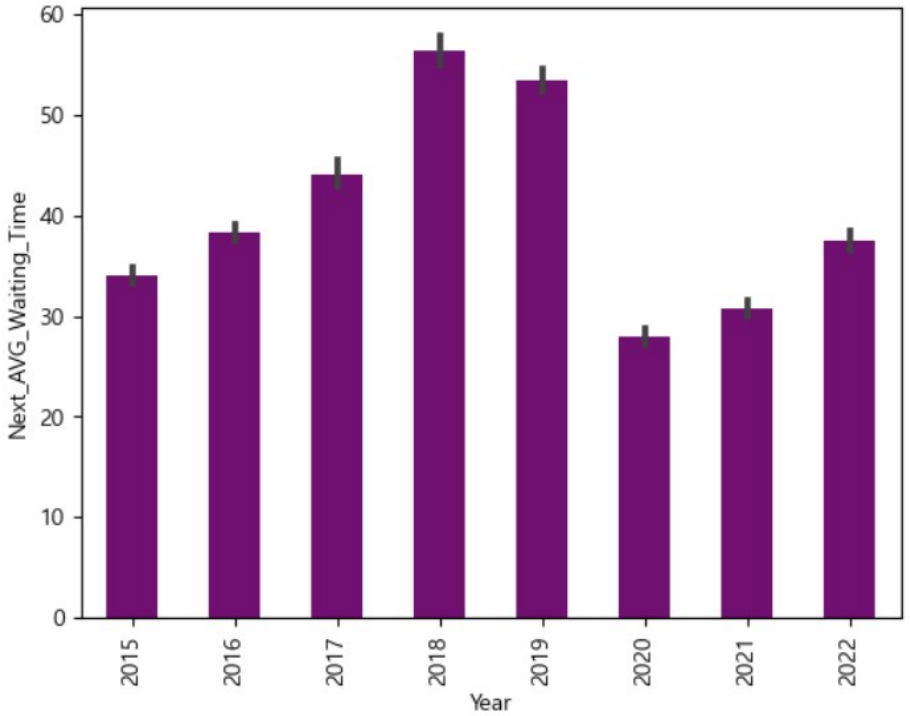
- 이변량 분석 (7day_AVG_Waiting_Time , Next_AVG_Waiting_Time)




```
PearsonRResult(statistic=0.7999795809204215, pvalue=0.0)
```

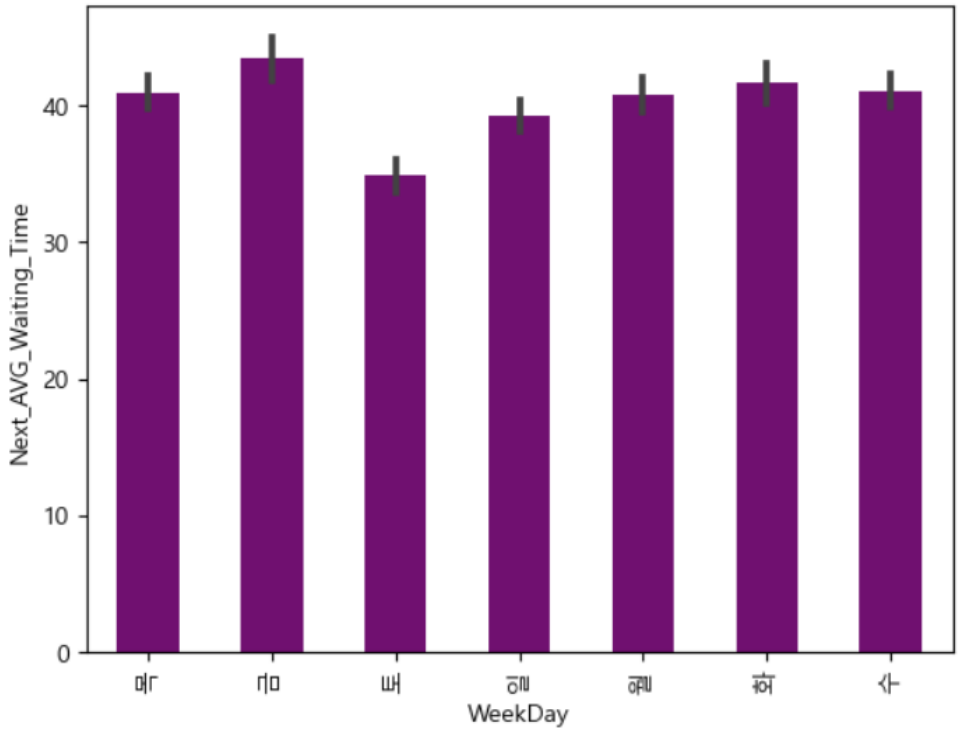
- 이변량 분석 (Year, Next_AVG_Waiting_Time)

2020년 시작된 코로나의 영향으로 자택으로 일하는 사람들이 많아져서 평균 대기시간이 줄어들지 않았을까 하는 생각이 든다.



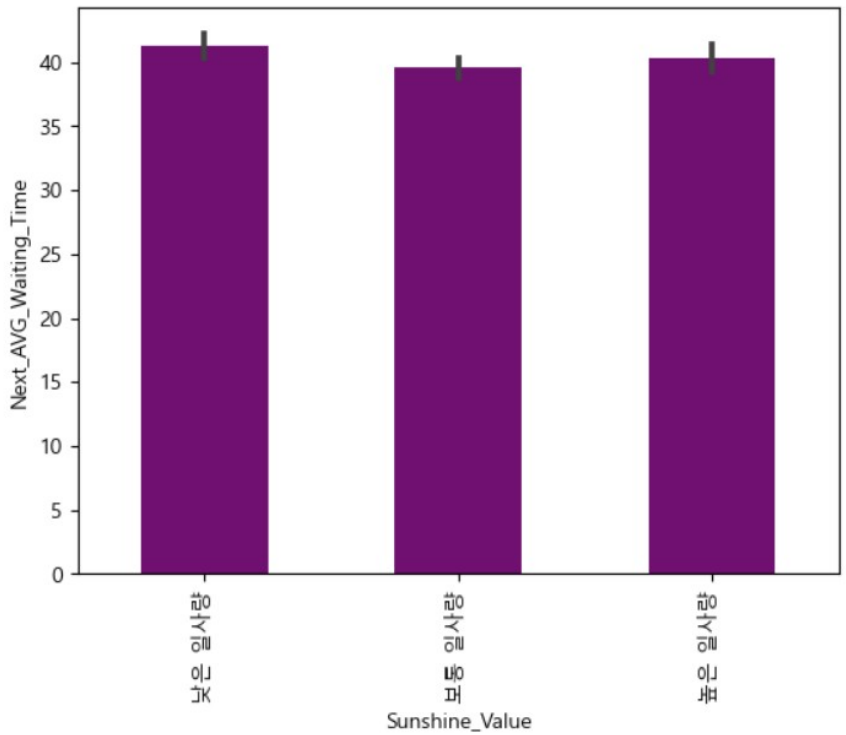

```
F_onewayResult(statistic=361.0792597034928, pvalue=0.0)
```

- 이변량 분석 (WeekDay, Next_AVG_Waiting_Time)



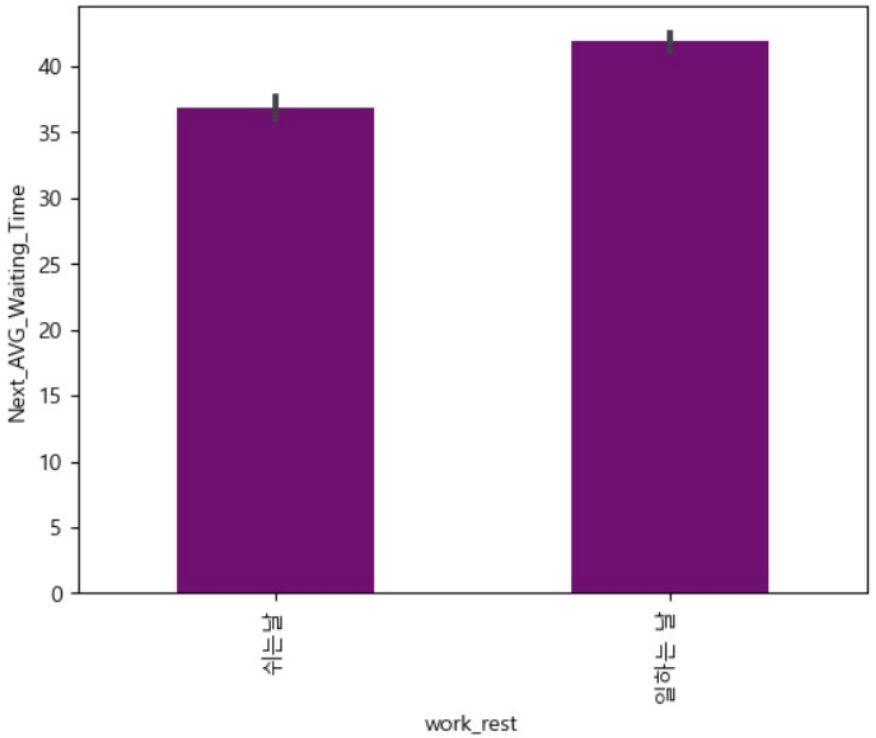
F_onewayResult(statistic=15.751468417474483, pvalue=6.961966308136947e-18)

- 이변량 분석 (Sunshine_Value, Next_AVG_Waiting_Time)



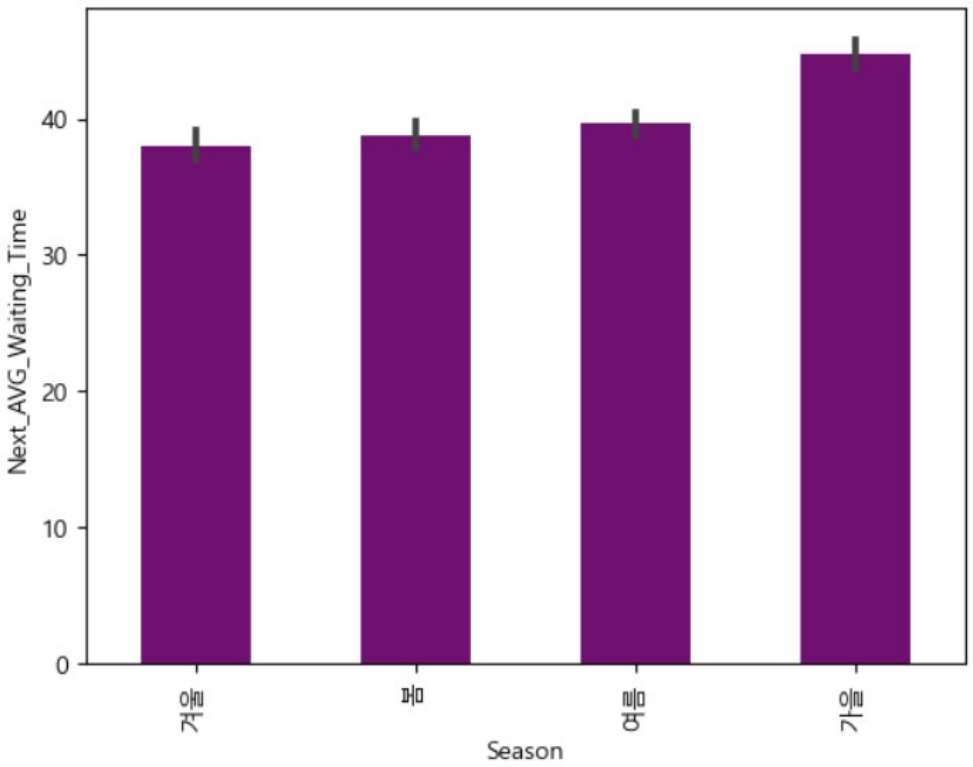
F_onewayResult(statistic=4.204261882540222, pvalue=0.015022322006234514)

- 이변량 분석 (work_rest, Next_AVG_Waiting_Time)



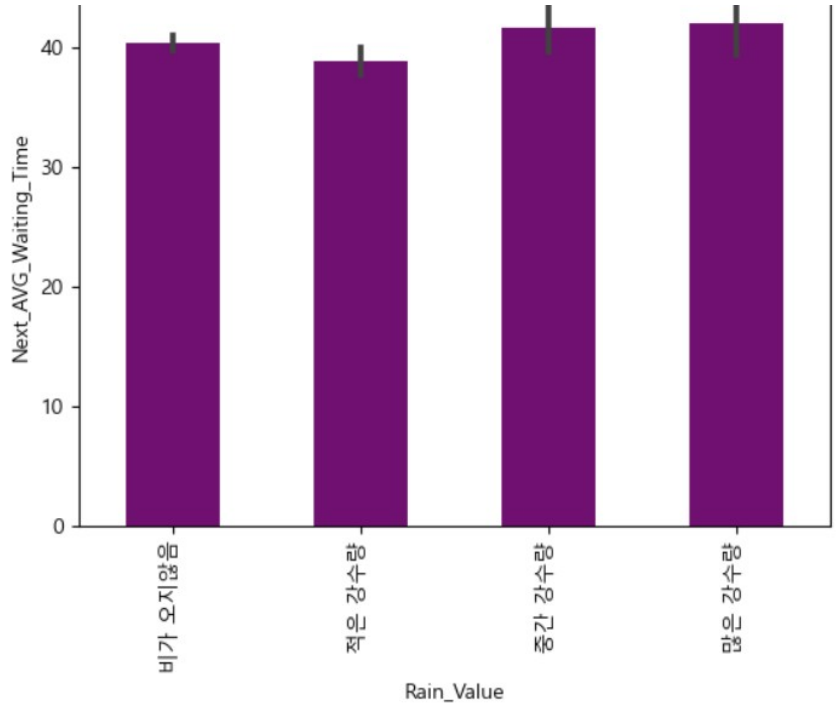

```
Ttest_indResult(statistic=-9.081110746306464, pvalue=1.9298268746630425e-19)
```

- 이변량 분석 (Season, Next_AVG_Waiting_Time)



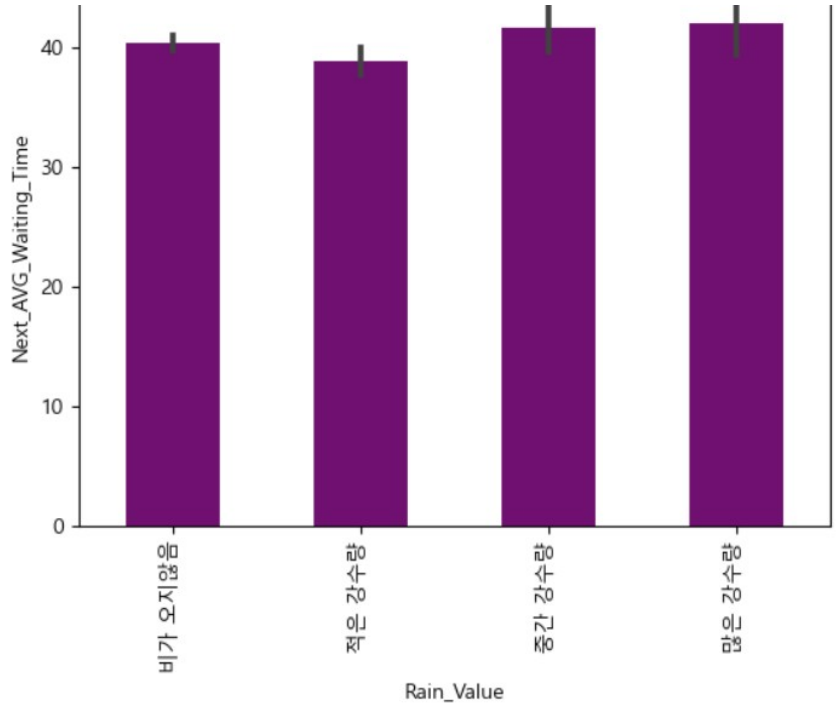

```
F_onewayResult(statistic=35.07940475582562, pvalue=2.8824417183702753e-22)
```

- 이변량 분석 (Rain_Value , Next_AVG_Waiting_Time)



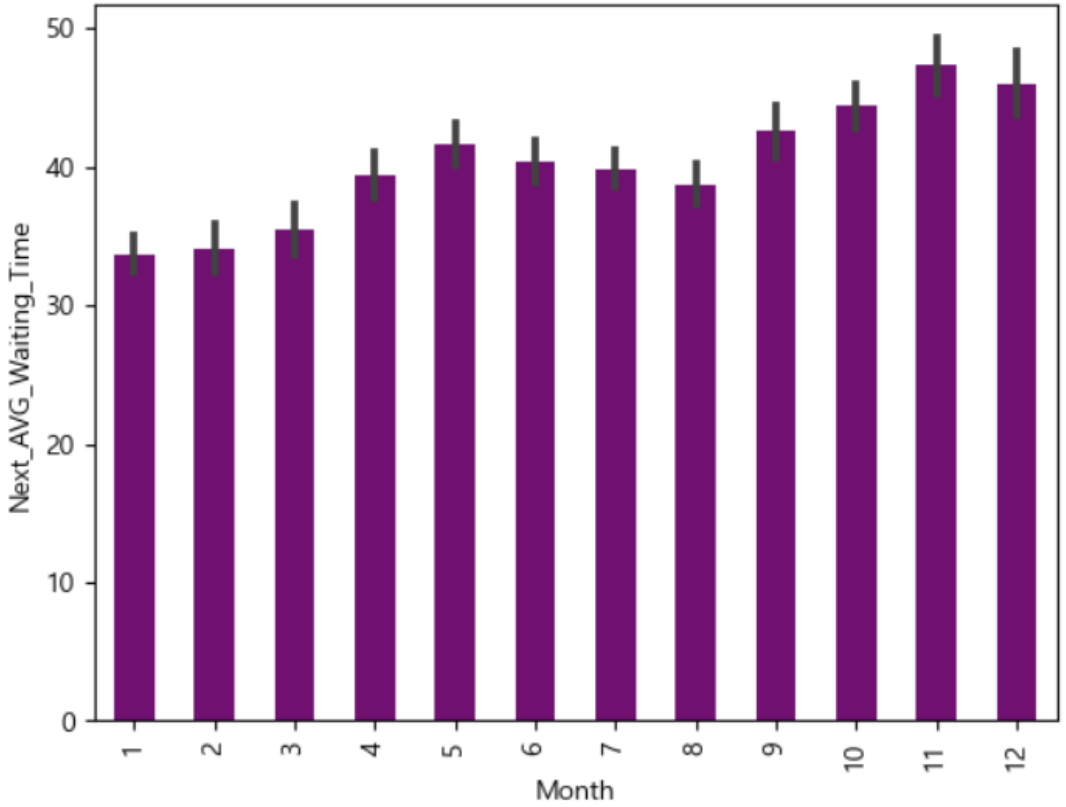
F_onewayResult(statistic=3.0802901537613407, pvalue=0.026409991584474556)

- 이변량 분석 (Rain_Value , Next_AVG_Waiting_Time)



F_onewayResult(statistic=3.0802901537613407, pvalue=0.026409991584474556)

- 이변량 분석 (Month , Next_AVG_Waiting_Time)



F_onewayResult(statistic=25.986234807972142, pvalue=4.025871362738365e-52)

- 이변량 분석 결과

정답은 없고, Y(Target)에 유의미한 영향을 줄 것이라고 생각하는 변수만 남겨놨다.

Riding_Rate	Avg_Waiting_Time	7day_Avg_Waiting_Time	Year	Season	Next_AVG_Waiting_Time

통합한 Tablour 데이터를 이용해 머신러닝/딥러닝 모델 작업

- **가변수화 및 원핫인코딩**

모델링 작업을 할 때 값은 숫자형이여야 한다.
따라서 범주형 변수는 가변수화 및 원핫인코딩을 진행한다.

(2) 가변수화

```
29]: df = pd.get_dummies(df, columns=['Year',  'Season'], drop_first=True)
```

```
30]: df.head(1)
```

30]:

vg_Waiting_Time	Next_AVG_Waiting_Time	Year_2016	Year_2017	Year_2018	Year_2019	Year_2020	Year_2021	Year_2022	Season_겨울	Season_봄	Season_여름
23.2	17.2	0	0	0	0	0	0	0	1	0	0

- **X(features)와 Y(Target)으로 나눈다음 Train과 Test로 분할하기**

test_size=91이 Tablour Data에 마지막 91개를 뜻한다.
즉 이로써 학습용과 평가용 데이터가 구분이 됐다.

2) train : validation 나누기

- 힌트 : train_test_split(, , test_size = 91, shuffle = False)

35]:
```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=91, shuffle=False)
```

- **Scaling**

모든 값을 0 ~ 1로 맞춰주면 모델을 학습 및 평가하는데 용이하다.

(4) Scaling

- KNN, SVM 알고리즘 및 DL을 적용하기 위해서는 스케일링을 해야 합니다.
- 모든 값을 0 ~ 1로 맞춰주면, 학습 및 평가하는데 용이하다.

```
# 스케일링
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
scaler.fit(x_train)
```

```
▾ MinMaxScaler
MinMaxScaler()
```

```
# 스케일링된 데이터를 데이터프레임으로 변환

# 학습 데이터와 평가 데이터에 대하여 transform
fit_x = pd.DataFrame(scaler.transform(x_train), columns=x_train.columns)
test_x = pd.DataFrame(scaler.transform(x_test), columns=x_test.columns)
```

```
fit_x.head(1)
```

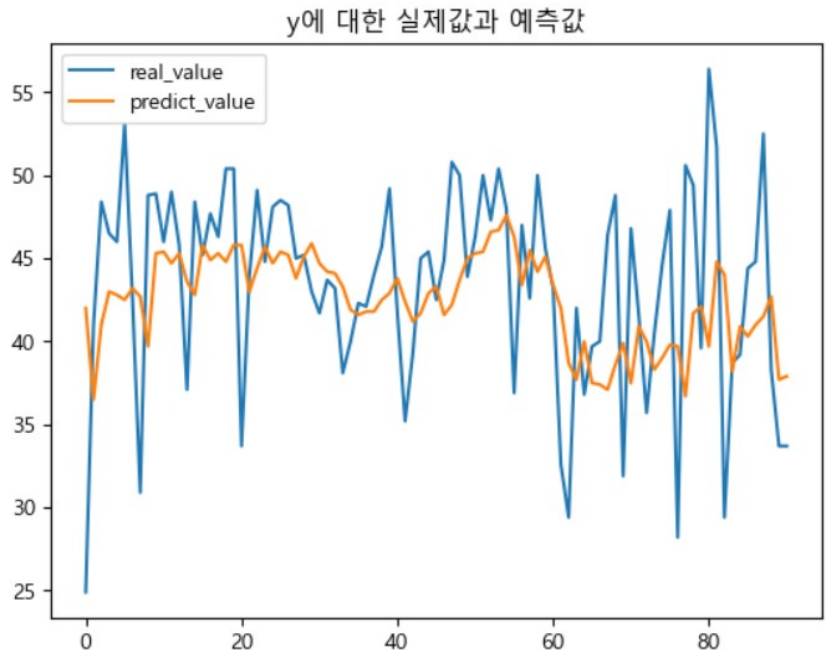
	Riding_Rate	Avg_Waiting_Time	7day_Avg_Waiting_Time	Year_2016	Year_2017	Year_2018	Year_2019	Year_2020	Year_2021	Year_2022	Season_겨울
0	0.823561	0.076046	0.067588	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0

- LinearRegression

1) 모델1 (LinearRegression)

```
model = LinearRegression()

model.fit(fit_x, y_train)

y_pred = model.predict(test_x)
print('예측값 10개 : ', np.around(y_pred[0:10:1], 1))
print('실제값 10개 : ', list(y_test[0:10:1]))
print('MAE : ', mean_absolute_error(y_test, y_pred))
print('MAPE : ', mean_absolute_percentage_error(y_test, y_pred))

show(list(y_test), list(np.around(y_pred, 1)))
```

```
예측값 10개 :  [42.  36.5 41.  43.  42.8 42.5 43.2 42.7 39.7 45.3]
실제값 10개 :  [24.9, 41.0, 48.4, 46.5, 46.0, 53.1, 42.6, 30.9, 48.8, 48.9]
MAE :  4.681228592094681
MAPE :  0.11503489272278701
```

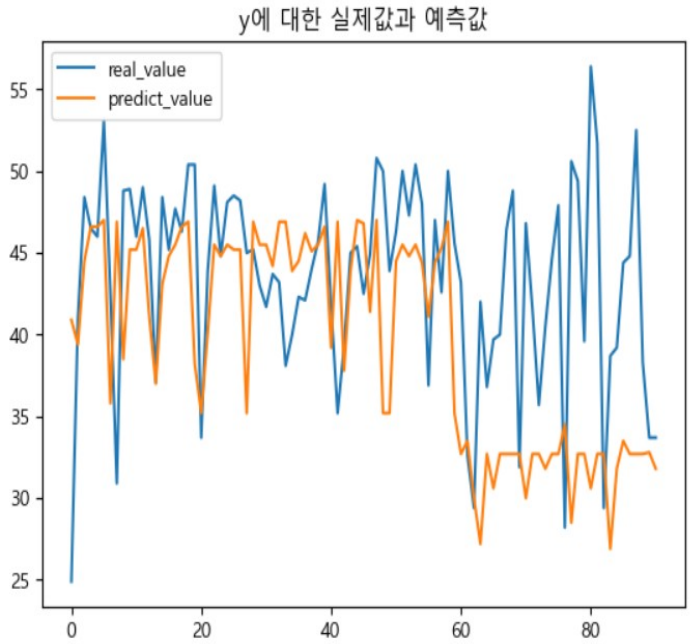


- KNN

2) 모델2 (KNN)

```
model = KNeighborsRegressor()

model.fit(fit_x, y_train)

y_pred = model.predict(test_x)
print('예측값 10개 : ', np.around(y_pred[0:10:1], 1))
print('실제값 10개 : ', list(y_test[0:10:1]))
print('MAE : ', mean_absolute_error(y_test, y_pred))
print('MAPE : ', mean_absolute_percentage_error(y_test, y_pred))

show(list(y_test), list(np.around(y_pred, 1)))
```

```
예측값 10개 :  [40.9 39.4 44.5 46.6 46.6 47.  35.8 46.9 38.5 45.2]
실제값 10개 :  [24.9, 41.0, 48.4, 46.5, 46.0, 53.1, 42.6, 30.9, 48.8, 48.9]
MAE :  6.228131868131869
MAPE :  0.14375041736556243
```

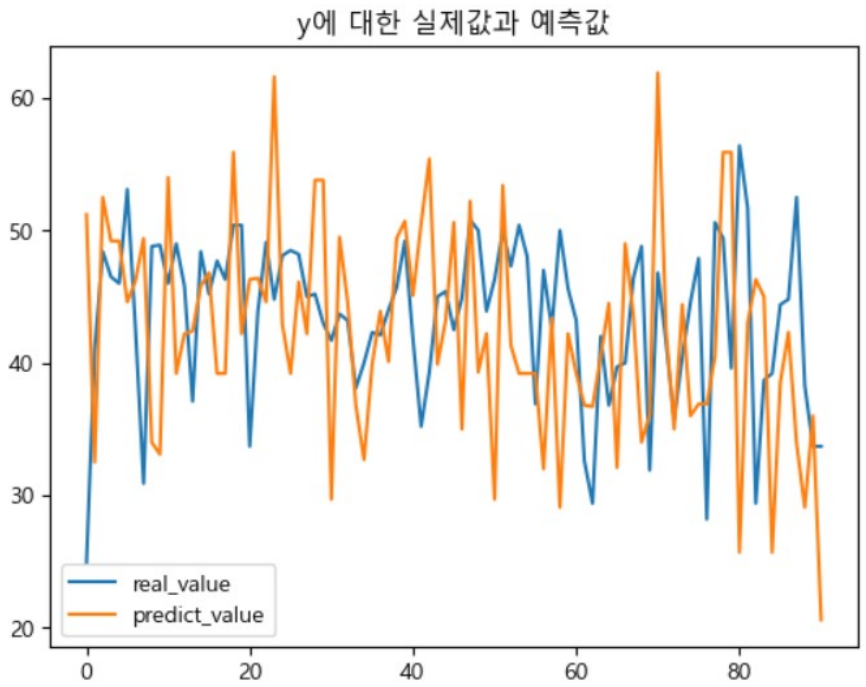


- Decision Tree

3) 모델3 (Decision Tree)

```
]: model = DecisionTreeRegressor()

model.fit(fit_x, y_train)

y_pred = model.predict(test_x)
print(type(y_pred))
print('예측값 10개 : ', np.around(y_pred[0:10:1], 1))
print('실제값 10개 : ', list(y_test[0:10:1]))
print('MAE : ', mean_absolute_error(y_test, y_pred))
print('MAPE : ', mean_absolute_percentage_error(y_test, y_pred))

show(list(y_test), list(np.around(y_pred, 1)))
```

```
<class 'numpy.ndarray'>
예측값 10개 :  [51.2 32.5 52.5 49.2 49.2 44.6 46.2 49.4 34.  33.1]
실제값 10개 :  [24.9, 41.0, 48.4, 46.5, 46.0, 53.1, 42.6, 30.9, 48.8, 48.9]
MAE :  7.882417582417582
MAPE :  0.18868577546930118
```

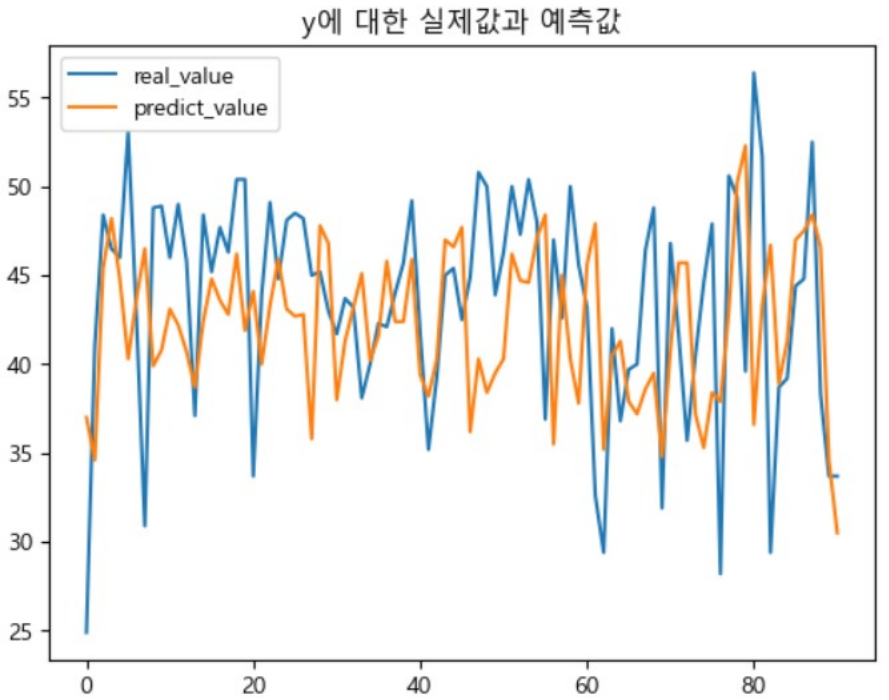


- Random Forest

4) 모델4 (Random Forest) ¶

38]:
```
model = RandomForestRegressor(n_estimators=100, max_depth=None)

model.fit(fit_x, y_train)

y_pred = model.predict(test_x)
print('예측값 10개 : ', np.around(y_pred[0:10:1], 1))
print('실제값 10개 : ', list(y_test[0:10:1]))
print('MAE : ', mean_absolute_error(y_test, y_pred))
print('MAPE : ', mean_absolute_percentage_error(y_test, y_pred))

show(list(y_test), list(np.around(y_pred, 1)))
```

```
예측값 10개 :  [37.  34.6 45.4 48.2 44.8 40.3 43.7 46.5 39.9 40.8]
실제값 10개 :  [24.9, 41.0, 48.4, 46.5, 46.0, 53.1, 42.6, 30.9, 48.8, 48.9]
MAE :  5.510769230769234
MAPE :  0.13250523077705836
```



- **Decision Tree**

맨 앞10개 예측값과 실제값
[22.8, 20.7, 20.7, 18.9, 24.4, 18.9, 24.4, 27.7, 25.9, 21.7]
[20.0, 20.0, 19.0, 24.0, 19.0, 24.0, 27.0, 26.0, 22.0, 22.0]

[417]:
```
# Decision Tree
DT = DTR(max_depth=8)

DT.fit(df_21_X, df_21_Y)

y_pred = DT.predict(df_22_X)
```

[418]:
```
# 예측값과 실제값 비교
predict = list(y_pred.flatten()[0:10:1])
real = list(df_22_Y['PM10_1'][0:10:1])
print('예측값 10개 : ', predict)
print('실제값 10개 : ', real)
```

```
예측값 10개 :  [22.77777777777778, 20.695575221238936, 20.695575221238936, 18.92156862745098, 24.41062801932367, 18.92156862745098, 24.4106280
1932367, 27.687637969094926, 25.883582089552238, 21.741622574955905]
실제값 10개 :  [20.0, 20.0, 19.0, 24.0, 19.0, 24.0, 27.0, 26.0, 22.0, 22.0]
```

[419]:
```
# test_y 데이터와 y_pred_LR 데이터로 성능을 평가하여 출력해보세요.
# 성능지표는 mse와 r2를 이용하세요.
print('mse : ', mse(df_22_Y, y_pred))
print('rmse : ', mse(df_22_Y, y_pred) ** 0.5)
print('mae : ', mae(df_22_Y, y_pred))
print('r2 : ', r2_score(df_22_Y, y_pred))
```

```
mse :  56.40727122052505
rmse :  7.510477429599602
mae :  4.2140793582068055
r2 :  0.897936538592672
```

- **Random Forest**

맨 앞10개 예측값과 실제값
[23.8, 20.5, 21.3, 18.9, 26.4, 17.6, 26.6, 28.1, 27.7, 25.0]
[20.0, 20.0, 19.0, 24.0, 19.0, 24.0, 27.0, 26.0, 22.0, 22.0]

[420]:
```
# RandomForest
```

[421]:
```
RF = RFR(n_estimators=50, max_depth=None)

RF.fit(df_21_X, df_21_Y)

y_pred = RF.predict(df_22_X)
```

```
C:\Users\user\AppData\Local\Temp\ipykernel_14848\2519420311.py:3: DataConversionWarning: A column-vector y was p
ected. Please change the shape of y to (n_samples,), for example using ravel().
  RF.fit(df_21_X, df_21_Y)
```

[422]:
```
# 예측값과 실제값 비교
predict = list(y_pred.flatten()[0:10:1])
real = list(df_22_Y['PM10_1'][0:10:1])
print('예측값 10개 : ', predict)
print('실제값 10개 : ', real)
```

```
예측값 10개 :  [23.78, 20.54, 21.28, 18.94, 26.38, 17.6, 26.56, 28.14, 27.72, 25.04]
실제값 10개 :  [20.0, 20.0, 19.0, 24.0, 19.0, 24.0, 27.0, 26.0, 22.0, 22.0]
```

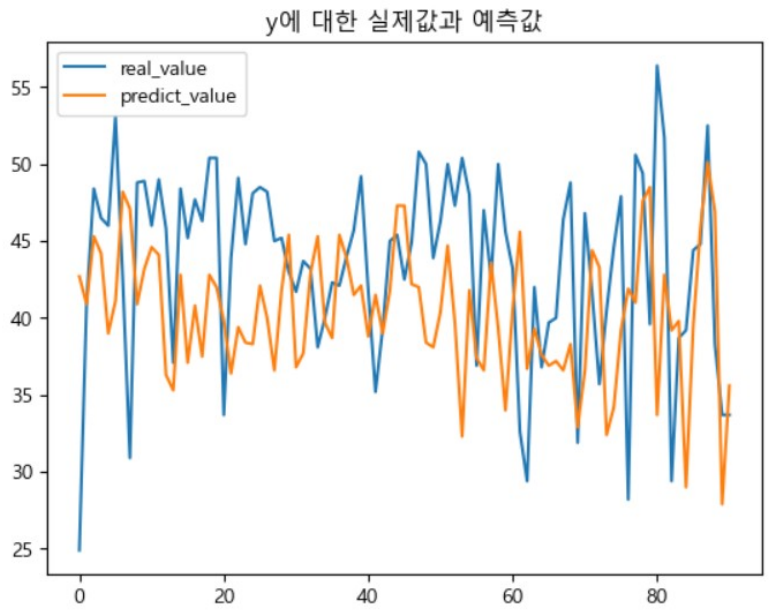
[423]:
```
# test_y 데이터와 y_pred_LR 데이터로 성능을 평가하여 출력해보세요.
# 성능지표는 mse와 r2를 이용하세요.
print('mse : ', mse(df_22_Y, y_pred))
print('rmse : ', mse(df_22_Y, y_pred) ** 0.5)
print('mae : ', mae(df_22_Y, y_pred))
print('r2 : ', r2_score(df_22_Y, y_pred))
```

```
mse :  40.447196380687465
rmse :  6.359811033410306
mae :  4.199424076047289
r2 :  0.9268147389953375
```

- XGBoost

```
model = xgb.XGBRegressor( n_estimators=100, max_depth=None )

model.fit(fit_x, y_train)

y_pred = model.predict(test_x)

print('예측값 10개 : ', np.around(y_pred[0:10:1], 1))
print('실제값 10개 : ', list(y_test[0:10:1]))
print('MAE : ', mean_absolute_error(y_test, y_pred))
print('MAPE : ', mean_absolute_percentage_error(y_test, y_pred))

show(list(y_test), list(np.around(y_pred, 1)))
```

```
예측값 10개 :  [42.7 40.9 45.3 44.2 39.  41.1 48.2 47.1 40.9 43.2]
실제값 10개 :  [24.9, 41.0, 48.4, 46.5, 46.0, 53.1, 42.6, 30.9, 48.8, 48.9]
MAE :  6.417734058086689
MAPE :  0.15176968296621277
```

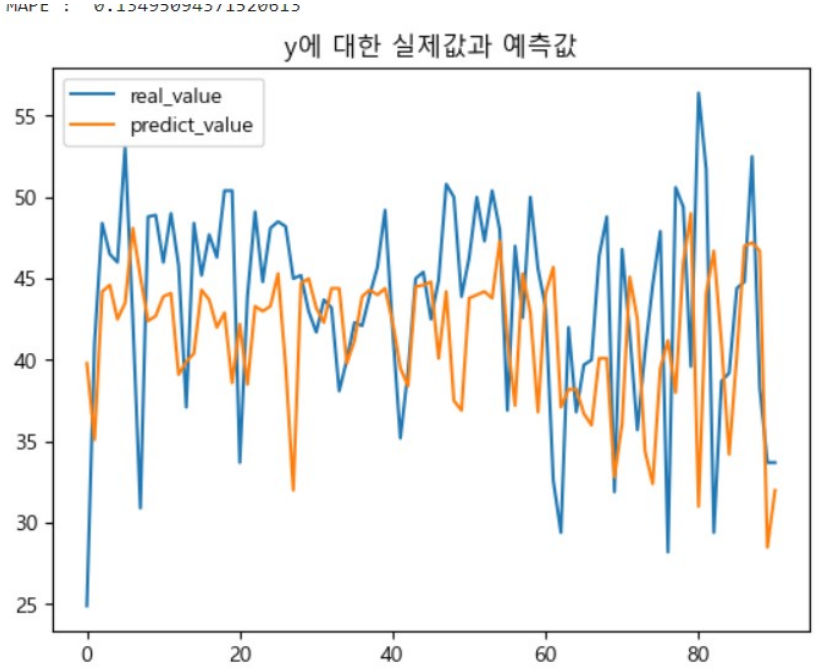


- LightGBM

6) 모델6 (LightGBM)

```
model = lgbm.LGBMRegressor( n_estimators=100, max_depth=None )

model.fit(fit_x, y_train)

y_pred = model.predict(test_x)

print('예측값 10개 : ', np.around(y_pred[0:10:1], 1))
print('실제값 10개 : ', list(y_test[0:10:1]))
print('MAE : ', mean_absolute_error(y_test, y_pred))
print('MAPE : ', mean_absolute_percentage_error(y_test, y_pred))

show(list(y_test), list(np.around(y_pred, 1)))
```

```
[LightGBM] [Warning] Auto-choosing row-wise multi-threading, the overhead of testing was 0.000845 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 785
[LightGBM] [Info] Number of data points in the train set: 2831, number of used features: 13
[LightGBM] [Info] Start training from score 40.206782
예측값 10개 :  [39.8 35.1 44.2 44.6 42.5 43.5 48.1 45.1 42.4 42.7]
실제값 10개 :  [24.9, 41.0, 48.4, 46.5, 46.0, 53.1, 42.6, 30.9, 48.8, 48.9]
MAE :  5.623559613210638
MAPE :  0.13495094371520613
```



- **LightGBM**

맨 앞10개 예측값과 실제값
[23.1, 20.6, 21.0, 19.5, 25.0, 19.6, 25.2, 28.1, 26.5, 22.5]
[20.0, 20.0, 19.0, 24.0, 19.0, 24.0, 27.0, 26.0, 22.0, 22.0]

[430]:
```
# LightGBM
import lightgbm
lgbm = lightgbm.LGBMRegressor()

lgbm.fit(df_21_X, df_21_Y)

y_pred = lgbm.predict(df_22_X)
```

```
[LightGBM] [Warning] Found whitespace in feature_names, replace with underlines
[LightGBM] [Warning] Auto-choosing row-wise multi-threading, the overhead of testing was 0.000938 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 884
[LightGBM] [Info] Number of data points in the train set: 8759, number of used features: 8
[LightGBM] [Info] Start training from score 38.786049
```

[431]:
```
# 예측값과 실제값 비교
predict = list(y_pred.flatten()[0:10:1])
real = list(df_22_Y['PM10_1'][0:10:1])
print('예측값 10개 : ', predict)
print('실제값 10개 : ', real)
```

```
예측값 10개 :  [23.13211023131903, 20.603573926791512, 21.054374213170444, 19.48381886950244, 25.0710642652207, 19.60744579024335, 25.19469118596161, 28.120141741366
695, 26.548438053429837, 22.52585730302512]
실제값 10개 :  [20.0, 20.0, 19.0, 24.0, 19.0, 24.0, 27.0, 26.0, 22.0, 22.0]
```

[432]:
```
# test_y 데이터와 y_pred_LR 데이터로 성능을 평가하여 출력해보세요.
# 성능지표는 mse와 r2를 이용하세요.
print('mse : ', mse(df_22_Y, y_pred))
print('rmse : ', mse(df_22_Y, y_pred) ** 0.5)
print('mae : ', mae(df_22_Y, y_pred))
print('r2 : ', r2_score(df_22_Y, y_pred))
```

```
mse :  34.31363770680151
rmse :  5.857784368411107
mae :  3.8534995255957245
r2 :  0.9379128158116107
```

- 딥러닝 모델1 - 은닉층 없는 것

```
# 모델의 구조를 형성한다.
X = tf.keras.Input(shape=[13])
Y = tf.keras.layers.Dense(1)(X)
model = tf.keras.Model(X, Y)
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.02),
              loss=tf.keras.losses.mean_squared_error,
              metrics=['mae', 'mape'])
```

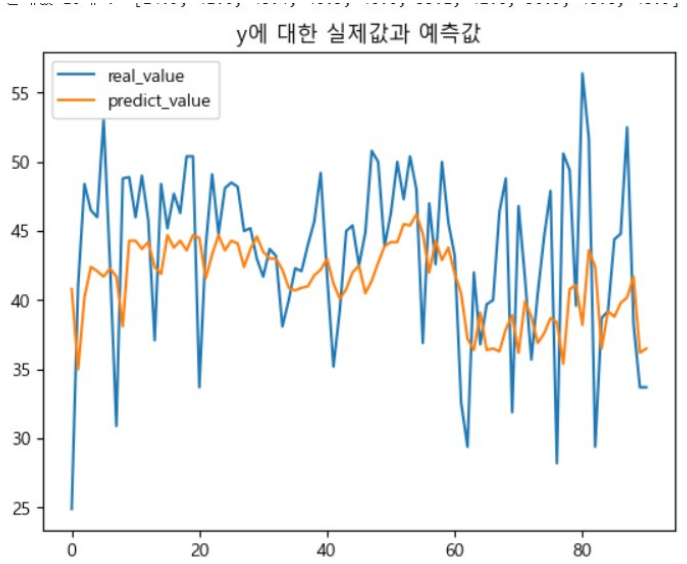
```
# 학습한다.

# 모델이 과적합에 걸리지 않도록, 과적합 되기 직전 상태로 만든다.
early_stop = tf.keras.callbacks.EarlyStopping(patience=20, restore_best_weights=True)
model.fit(fit_x, y_train, batch_size=100, epochs=500, verbose=1, validation_split=0.3, callbacks=[early_stop])
```

```
# 이용한다.
model.evaluate(test_x, y_test)
```

```
3/3 [==============================] - 0s 8ms/step - loss: 40.4350 - mae: 5.0283 - mape: 12.0060
[40.43497848510742, 5.028273582458496, 12.005993843078613]
```

```
# 회귀 모델 평가 지표
print('MAE : ', mean_absolute_error(y_test, y_pred))
print('MAPE : ', mean_absolute_percentage_error(y_test, y_pred))
```

```
MAE :  5.028273454603259
MAPE :  0.1200599384431567
```



- 딥러닝 모델2 - 은닉층 존재

2) 모델2 - 은닉층 존재

```
X = tf.keras.Input(shape=[13])
H = tf.keras.layers.Dense(1024, activation=tf.keras.activations.swish)(X)
H = tf.keras.layers.Dense(512, activation=tf.keras.activations.swish)(H)
H = tf.keras.layers.Dense(256, activation=tf.keras.activations.swish)(H)
H = tf.keras.layers.Dense(128, activation=tf.keras.activations.swish)(H)
Y = tf.keras.layers.Dense(1)(H)

model = tf.keras.Model(X, Y)
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.01),
              loss=tf.keras.losses.MeanSquaredError(),
              metrics=['mae', 'mape'],)
model.summary()
```

```
# 학습한다.

# 모델이 과적합에 걸리지 않도록, 과적합 되기 직전 상태로 만든다.
early_stop = tf.keras.callbacks.EarlyStopping(patience=20, restore_best_weights=True)
model.fit(fit_x, y_train, batch_size=50, epochs=500, verbose=1, validation_split=0.2, callbacks=[early_stop])
```

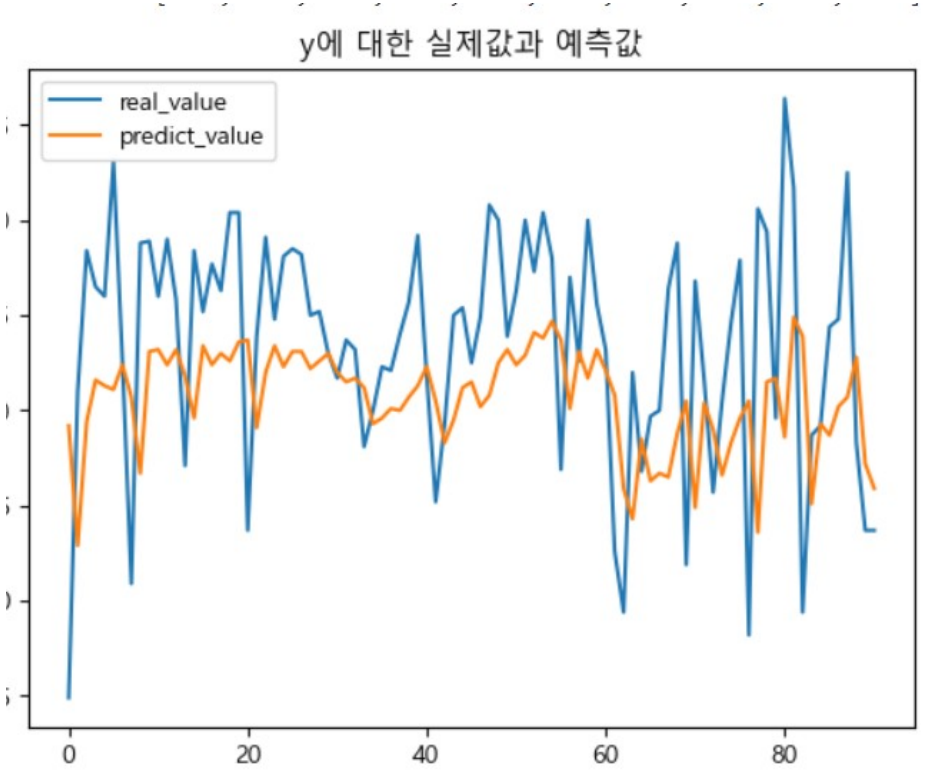
- 딥러닝 모델2 - 은닉층 존재

```
# 이용한다.
model.evaluate(test_x, y_test)
```

```
3/3 [==============================] - 0s 7ms/step - loss: 45.6789 - mae: 5.5441 - mape: 13.1085
[45.67887878417969, 5.5440874099731445, 13.108461380004883]
```

```
# 회귀 모델 평가 지표
print('MAE : ', mean_absolute_error(y_test, y_pred))
print('MAPE : ', mean_absolute_percentage_error(y_test, y_pred))
```

```
MAE :  5.544088216928335
MAPE :  0.1310846159344078
```



- 딥러닝 모델3 - 은닉층 존재, BatchNormalization, Dropout, SkipConnection 등 다양한 방법 활용

3) 모델3 - 은닉층 존재, BatchNormalization, Dropout, SkipConnection 등 다양한 방법 활용

```
# 모델 구조를 만든다.
X = tf.keras.Input(shape=[13])
H = tf.keras.layers.Dense(512)(X)
H = tf.keras.layers.BatchNormalization()(H)
H = tf.keras.layers.Activation('swish')(H)

H = tf.keras.layers.Dropout(0.6)(H)
H = tf.keras.layers.Dense(256)(H)
H = tf.keras.layers.BatchNormalization()(H)
H = tf.keras.layers.Activation('swish')(H)

for i in range(0, 10, 1):
    H1 = tf.keras.layers.Dropout(0.5)(H)
    H1 = tf.keras.layers.Dense(256)(H1)  # 이 부분을 수정
    H1 = tf.keras.layers.BatchNormalization()(H1)
    H1 = tf.keras.layers.Activation('swish')(H1)

    H1 = tf.keras.layers.Dense(256)(H1)  # 이 부분을 수정
    H1 = tf.keras.layers.BatchNormalization()(H1)
    H = tf.keras.layers.Add()([H, H1])
    H = tf.keras.layers.Activation('swish')(H)

H = tf.keras.layers.Dropout(0.6)(H)
H = tf.keras.layers.Dense(128)(H)
H = tf.keras.layers.BatchNormalization()(H)
H = tf.keras.layers.Activation('swish')(H)

H = tf.keras.layers.Dropout(0.5)(H)
H = tf.keras.layers.Dense(64)(H)
H = tf.keras.layers.BatchNormalization()(H)
H = tf.keras.layers.Activation('swish')(H)

Y = tf.keras.layers.Dense(1)(H)

model = tf.keras.Model(X, Y)
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.01),
              loss=tf.keras.losses.mean_squared_error,
              metrics=['mae', 'mape'],)
```

```
# 학습한다.

# 모델이 과적합에 걸리지 않도록, 과적합 되기 직전 상태로 만든다.
early_stop = tf.keras.callbacks.EarlyStopping(patience=20, restore_best_weights=True)
model.fit(fit_x, y_train, batch_size=128, epochs=1000, verbose=1, validation_split=0.2, callbacks=[early_stop])
```

- 딥러닝 모델3 - 은닉층 존재, BatchNormalization, Dropout, SkipConnection 등 다양한 방법 활용

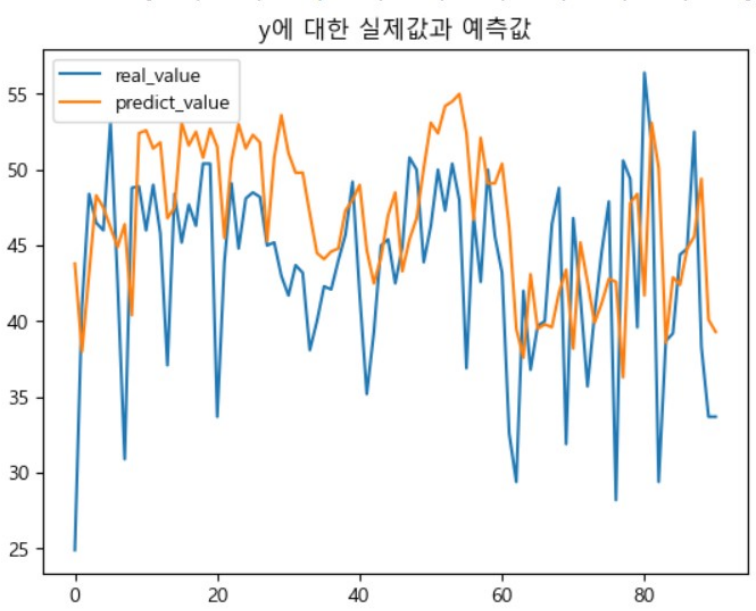
[61]:
```
# 이용한다.
model.evaluate(test_x, y_test)
```

```
3/3 [==============================] - 0s 16ms/step - loss: 53.9453 - mae: 5.7335 - mape: 14.5810
```

[61]:
```
[53.945335388183594, 5.73351526260376, 14.580961227416992]
```

[62]:
```
# 회귀 모델 평가 지표
print('MAE : ', mean_absolute_error(y_test, y_pred))
print('MAPE : ', mean_absolute_percentage_error(y_test, y_pred))
```

```
MAE :  5.733515209156078
MAPE :  0.1458095920793581
```



- 미니 프로젝트를 진행하면서 느낀점

1. 나름 이변량 분석을 해서 Y(Target)에 유의미하다고 생각한 변수를 뽑아서 머신러닝/딥러닝 모델을 만들어봤는데 생각보다 성능이 시원찮아서, 변수를 잘못 뽑았나 싶은 생각이 든다.