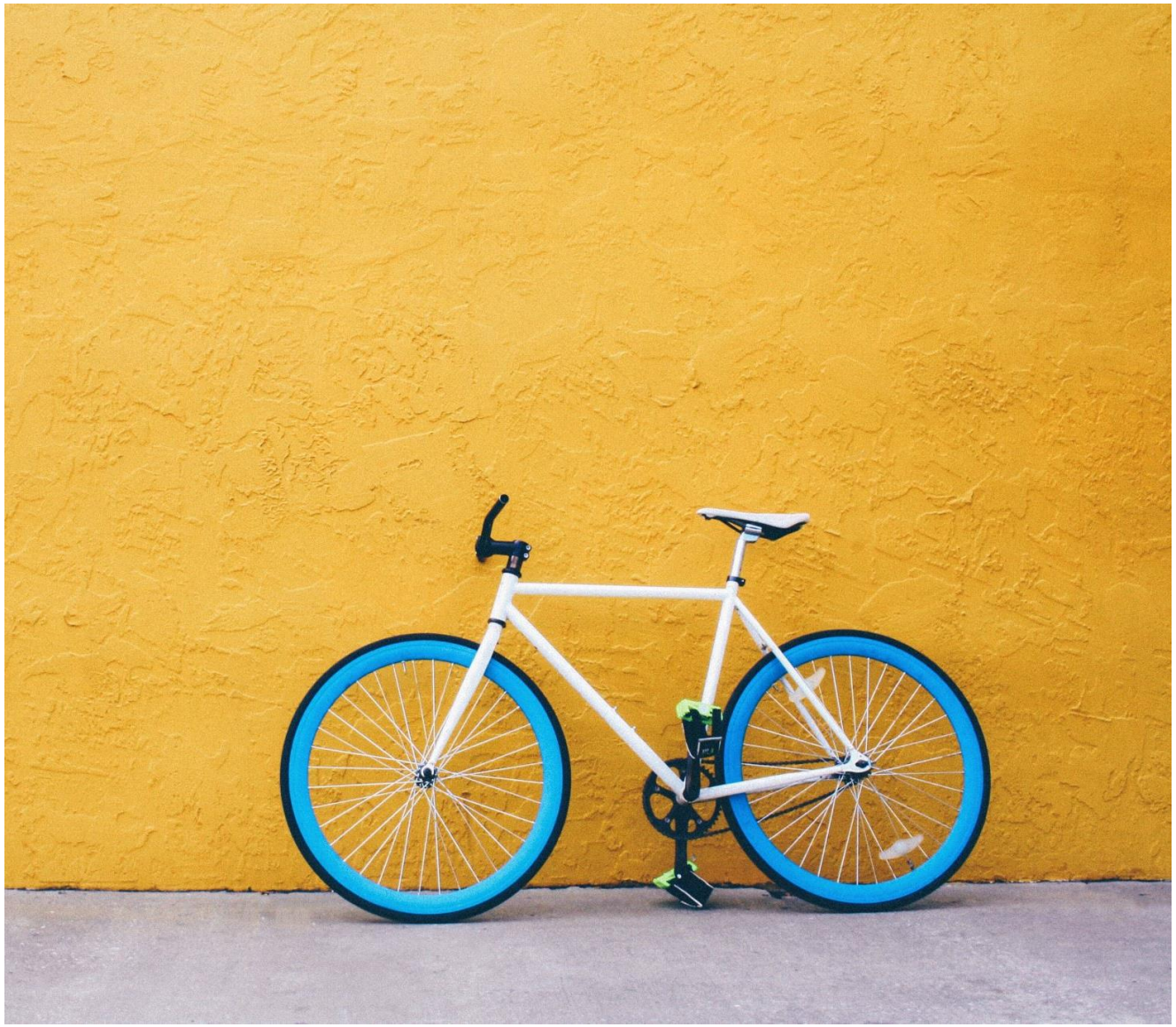
자전거 대여 데이터 분석



새싹 도봉캠퍼스 김현서

목차

1. What?
2. 사용한 라이브러리
3. 데이터 확인
4. 데이터 전처리
5. 선형 회귀 분석
6. 분류 분석

What?

시간, 날씨 데이터를 활용한 대여 횟수 예측

→ 대여 횟수가 많은 조건을 찾아 자전거가 부족하지 않도록 추가로 배치하는 등의 조치

미등록 대여 횟수와 등록 대여 횟수를 비교하여 분류

→ 미등록 대여 횟수가 등록 대여 횟수보다 더 많은 조건을 찾아 등록을 유도할 수 있는 방안 탐색

사용한 라이브러리

pandas	데이터 조작 및 분석을 위한 소프트웨어 라이브러리	
matplotlib	pyplot	MATLAB과 유사한 플로팅 방식을 제공하는 라이브러리
seaborn	matplotlib에 기반한 파이썬 시각화 라이브러리	
scikit-learn	model_selection	교차 검증 및 하이퍼파라미터 튜닝과 같은 모델 선택을 위한 도구
	linear_model	다양한 선형 모델
	metrics	평가를 위한 도구
	preprocessing	Scaling, Normalization 등 전처리를 수행하는 도구
	tree	분류 및 회귀를 위한 결정 트리 기반 모델
	ensemble	분류, 회귀 및 이상 탐지를 위한 앙상블 기반 방법

Source: https://scikit-learn.org/1.5/api/index.html

데이터 확인

datetime	일시
season	계절
holiday	휴일여부
workingday	주말주중
weather	날씨
temp	온도
atemp	체감온도
humidity	상대습도
windspeed	풍속
casual	미등록대여횟수
registered	등록대여횟수
count	대여횟수

편의를 위해 컬럼명을 전부 한글로 수정

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10886 entries, 0 to 10885
Data columns (total 12 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   일시       10886 non-null  object
 1   계절       10886 non-null  int64
 2   휴일여부     10886 non-null  int64
 3   주말주중     10886 non-null  int64
 4   날씨       10886 non-null  int64
 5   온도       10886 non-null  float64
 6   체감온도     10886 non-null  float64
 7   상대습도     10886 non-null  int64
 8   풍속       10886 non-null  float64
 9   미등록대여횟수  10886 non-null  int64
 10  등록대여횟수   10886 non-null  int64
 11  대여횟수     10886 non-null  int64
dtypes: float64(3), int64(8), object(1)
memory usage: 1020.7+ KB
```

데이터의 기본 정보 확인

10886개의 row
12개의 column
결측치 없음
일시: object type

describe() 로 데이터의 기초 통계 자료 확인

2011년부터 2012년까지의 데이터를 보유
한 시간 단위로 기록
날씨의 평균은 1.42 정도인 반면 상대습도의 평균은 61.89 정도로 차이가 크다

	일시	계절	휴일여부	주말주중	날씨	온도	체감온도	상대습도	풍속	미등록대여횟수	등록대여횟수	대여횟수
count	10886	10886.000000	10886.000000	10886.000000	10886.000000	10886.00000	10886.000000	10886.000000	10886.000000	10886.000000	10886.000000	10886.000000
mean	2011-12-27 05:56:22.399411968	2.506614	0.028569	0.680875	1.418427	20.23086	23.655084	61.886460	12.799395	36.021955	155.552177	191.574132
min	2011-01-01 00:00:00	1.000000	0.000000	0.000000	1.000000	0.82000	0.760000	0.000000	0.000000	0.000000	0.000000	1.000000
25%	2011-07-02 07:15:00	2.000000	0.000000	0.000000	1.000000	13.94000	16.665000	47.000000	7.001500	4.000000	36.000000	42.000000
50%	2012-01-01 20:30:00	3.000000	0.000000	1.000000	1.000000	20.50000	24.240000	62.000000	12.998000	17.000000	118.000000	145.000000
75%	2012-07-01 12:45:00	4.000000	0.000000	1.000000	2.000000	26.24000	31.060000	77.000000	16.997900	49.000000	222.000000	284.000000
max	2012-12-19 23:00:00	4.000000	1.000000	1.000000	4.000000	41.00000	45.455000	100.000000	56.996900	367.000000	886.000000	977.000000
std	NaN	1.116174	0.166599	0.466159	0.633839	7.79159	8.474601	19.245033	8.164537	49.960477	151.039033	181.144454

데이터 전처리

1. 일시를 연, 월, 일, 시간으로 나누기

일시 데이터를 datetime type으로 변경 후 → 각 단위로 나누어 컬럼 생성

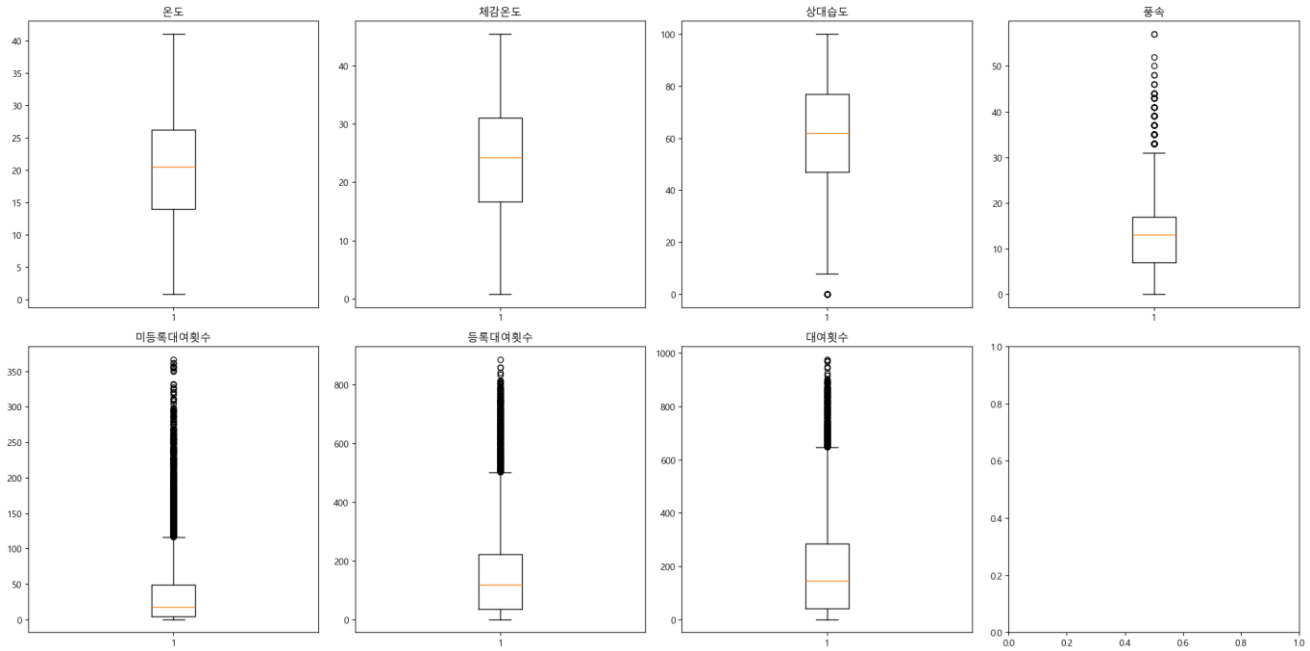
```
df_time['연도'] = df_time['일시'].dt.year
df_time['월'] = df_time['일시'].dt.month
df_time['일'] = df_time['일시'].dt.day
df_time['시간'] = df_time['일시'].dt.hour
```

	연도	월	일	시간
0	2011	1	1	0
1	2011	1	1	1
2	2011	1	1	2

2. 이상치 확인

박스 플롯을 사용하여 컬럼의 분포를 확인

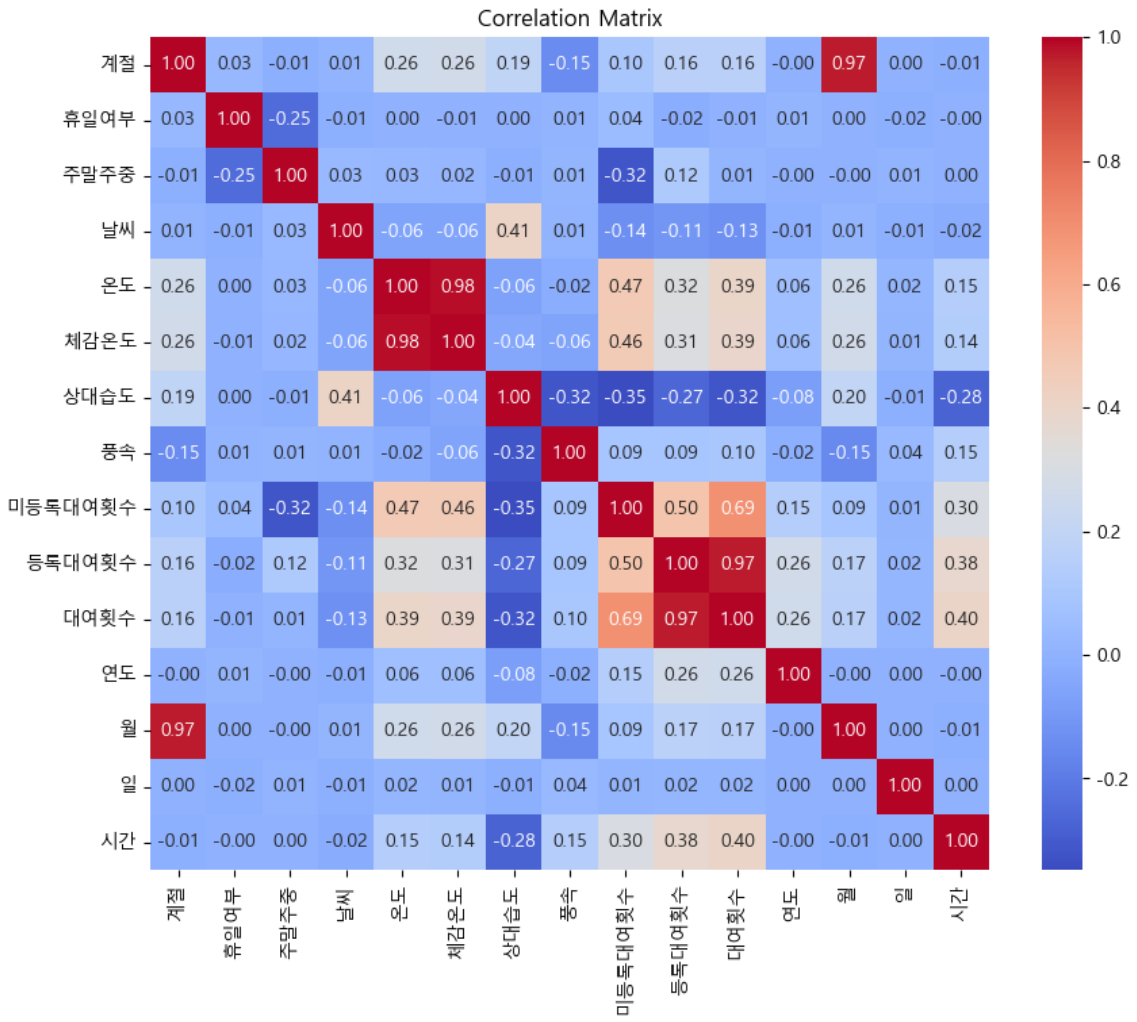
- ✓ 대여 횟수를 나타내는 세 가지 컬럼들에서 데이터가 편향되어 있음을 확인
- ✓ 풍속 컬럼에 이상치로 보이는 값들이 존재하지만, 심각하지 않으므로 제거한 경우와 제거하지 않은 경우를 모두 수행 후 비교



선형 회귀 분석

: 시간, 날씨 데이터를 활용한 대여 횟수 예측

1. 상관관계 확인



상관계수를 정렬하고 값이 0.9 이상인 경우만 출력

체감온도	온도	0.985
계절	월	0.972
대여횟수	등록대여횟수	0.971

2. 다중공선성 방지

선형 회귀에서는 다중공선성이 높다면 오차가 발생할 수 있기 때문에 두 독립변수의
상관계수가 0.9 이상인 경우 하나의 변수를 제거하고 분석을 수행

```
df_cor = df_time.drop(['계절', '체감온도'], axis=1)
```

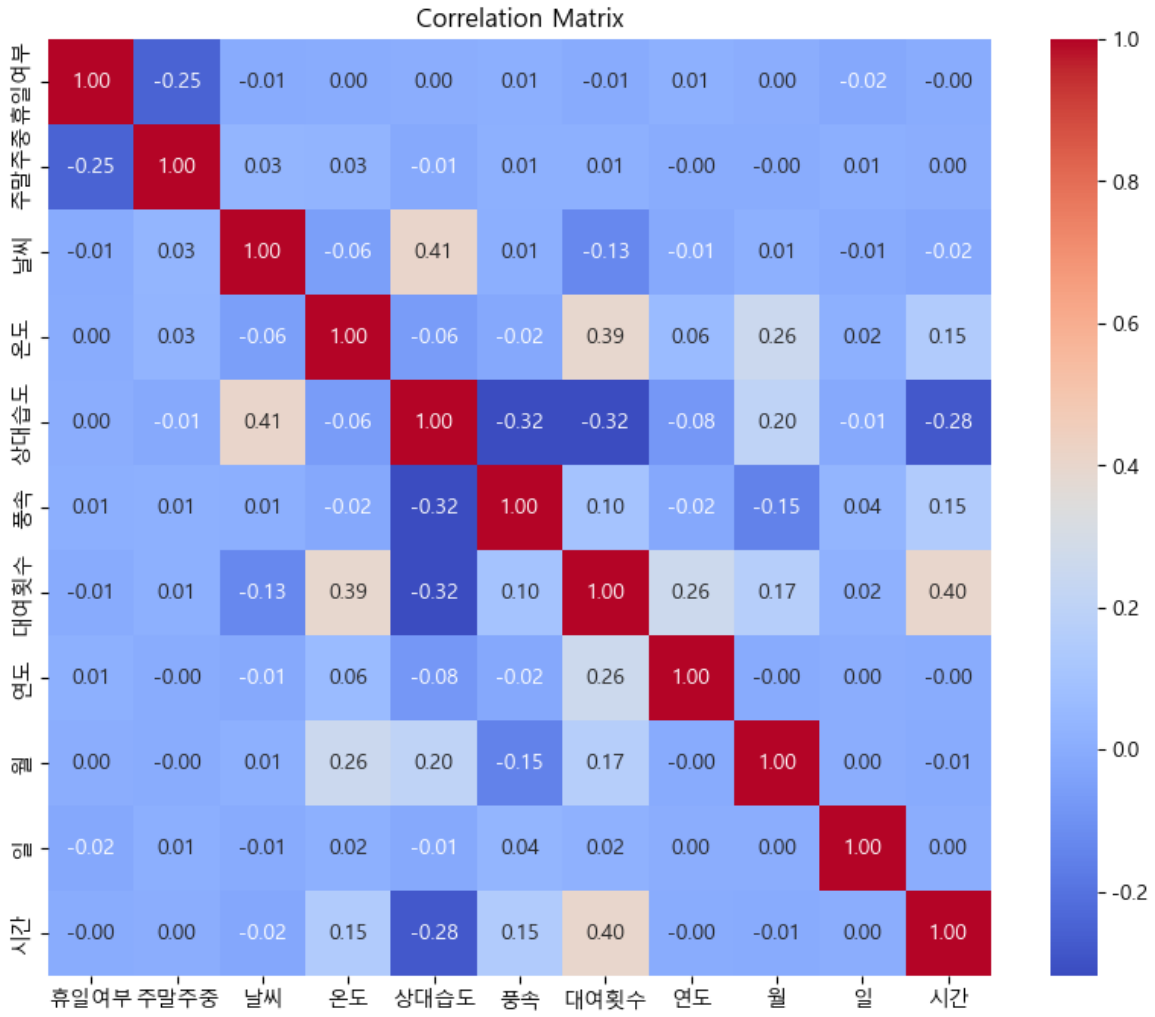
✓ 실제 기온을 나타내는 값을 유지하는 것이 좋을 것이라고 판단,
체감온도는 날씨와 상대습도 등의 변수에서 추가적으로 고려

✓ 월이 계절보다 더 좁은 범위를 나타내기 때문에, 조건을 찾기 더 좋을 것이라고 판단

3. 미등록 / 등록 대여 횟수 변수 제거

대여 횟수를 모른다면 미등록 대여 횟수와 등록 대여 횟수도 알지 못하기 때문에,
예측에 필요 없는 변수라고 판단하여 제거

4. 예측에 사용할 최종 데이터로 다시 상관관계 확인



온도, 상대습도, 시간이 종속 변수인 대여 횟수와 높은 상관계수를 보이고 있다

대여 횟수	온도	0.39
	상대습도	-0.32
	시간	0.4

5. 모델링

Linear Regression

✓ 데이터 분할, 모델 학습

test_size = 0.2
random_state = 42

```
X11 = df_2.drop('대여횟수', axis=1)
y11 = df_2['대여횟수']

X11_train, X11_test, y11_train, y11_test = train_test_split(X11, y11, test_size=0.2, random_state=42)

lr11 = LinearRegression()
lr11.fit(X11_train, y11_train)
```

✓ 모델 예측, coefficient 확인

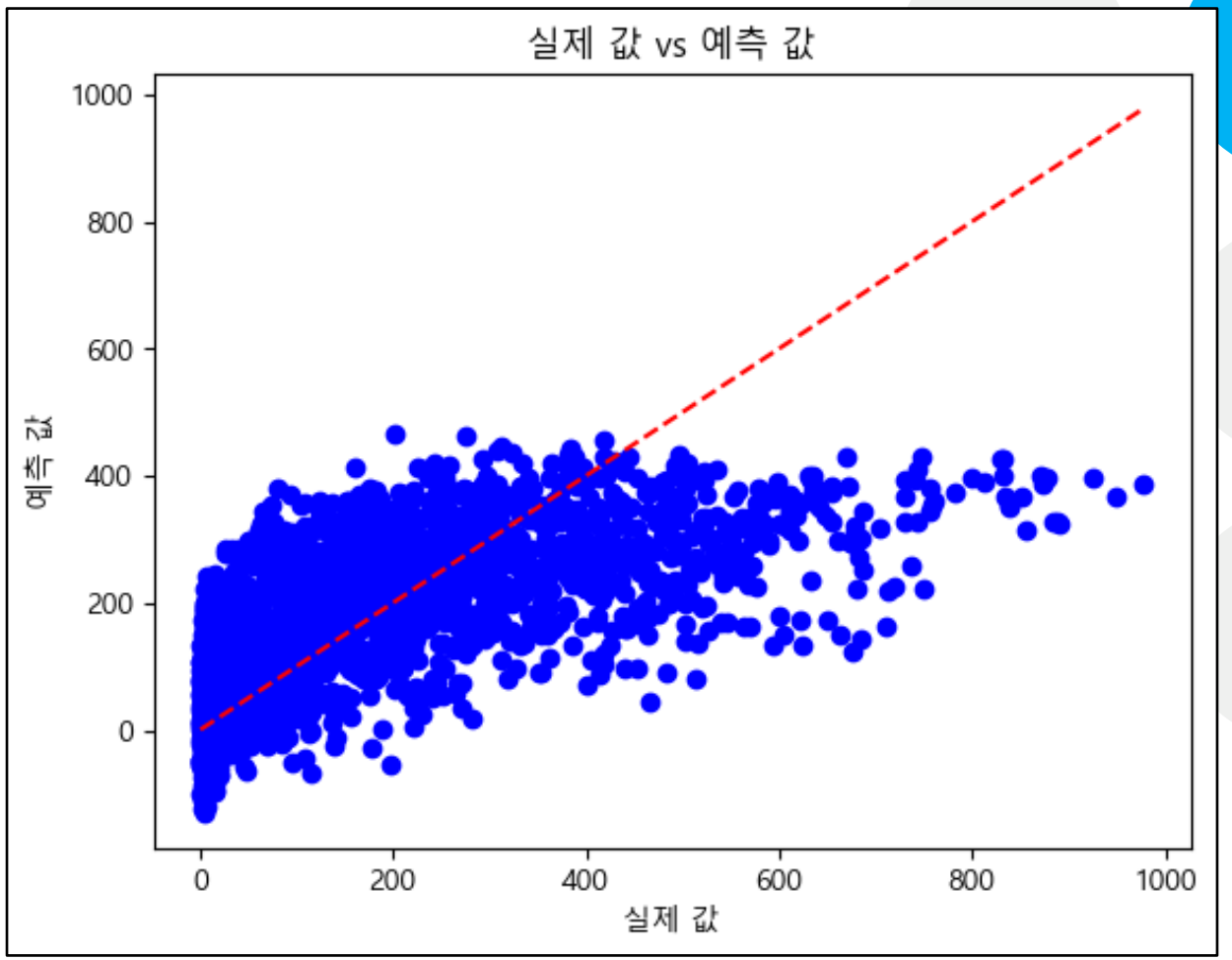
```
y11_pred = lr11.predict(X11_test)
print(f'기울기: {lr11.coef_}')
```

```
기울기: [-12.06654403   0.23311336  -5.59426865   6.508382    -2.00738343
   0.35879906  82.55369586   7.64606064   0.21617355   7.89506792]
```

✓ 예측 결과 확인 및 평가 지표

R^2	RMSE
0.395132	141.2970

```
r211 = r2_score(y11_test, y11_pred)
rmse11 = root_mean_squared_error(y11_test, y11_pred)

plt.scatter(y11_test, y11_pred, c='b')
plt.plot([min(y11_test), max(y11_test)], [min(y11_test), max(y11_test)], 'r--')
plt.xlabel('실제 값')
plt.ylabel('예측 값')
plt.title('실제 값 vs 예측 값')
plt.show()
```



✓ 결정계수가 매우 낮다

✓ 평균 제곱근 오차가 매우 크다

✓ 그래프에서 예측 값과 실제 값을 비교한 파란색 점들과 완벽한 예측인 빨간색 점선 사이의 거리가 멀어 보인다

5. 모델링

Lasso

✓ 데이터 분할, 모델 학습

test_size = 0.2
random_state = 42

alpha: [0.1, 0.5, 1]

✓ 모델 예측, coefficient 확인

```
기울기: [-0.          0.         -2.32487245  6.52730718 -2.0681406   0.29246085
 78.34946117  7.58710313  0.19546627  7.8404207 ]
```

몇 가지 변수들의 기울기 값이 0이 된 것을 볼 수 있다

✓ 예측 결과 확인 및 평가 지표

alpha = 0.1		alpha = 0.5		alpha = 1	
R^2	RMSE	R^2	RMSE	R^2	RMSE
0.395228	141.2858	0.395317	141.2754	0.395149	14.2958

전반적으로 Linear Regression 보다는 결정계수 값이 크고, RMSE 값은 줄었지만 차이가 미미하고 성능이 별로이다

5. 모델링

Random Forest Regressor

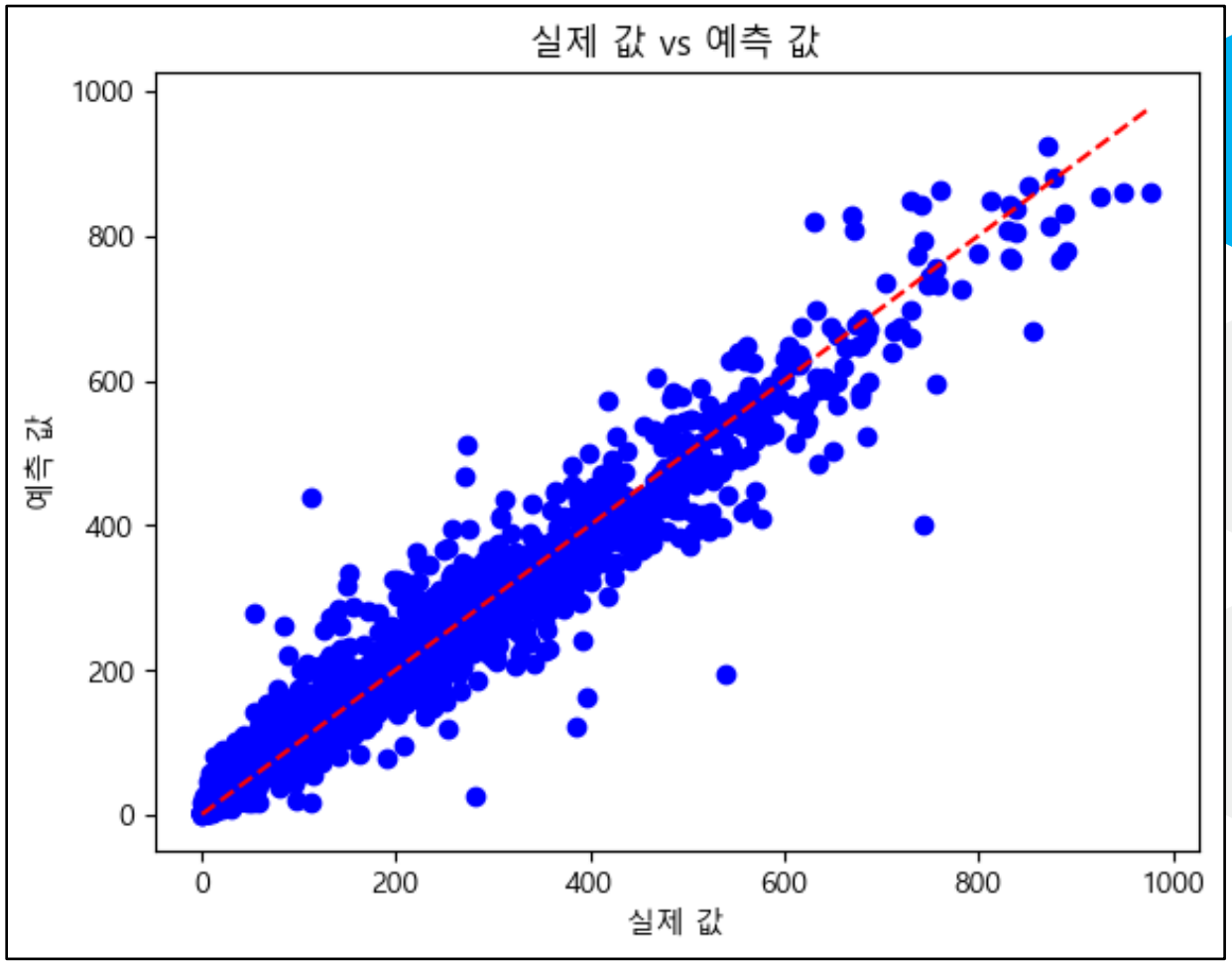
✓ 데이터 분할, 모델 학습

test_size = 0.2
random_state = 42

n_estimators = 100

✓ 예측 결과 확인 및 평가 지표

R^2	RMSE
0.9437	42.8511



결정계수의 값이 매우 커지고 RMSE 값은 작아져, 실험한 모델들 중 가장 성능이 좋은 것을 확인했다

6. 최적화

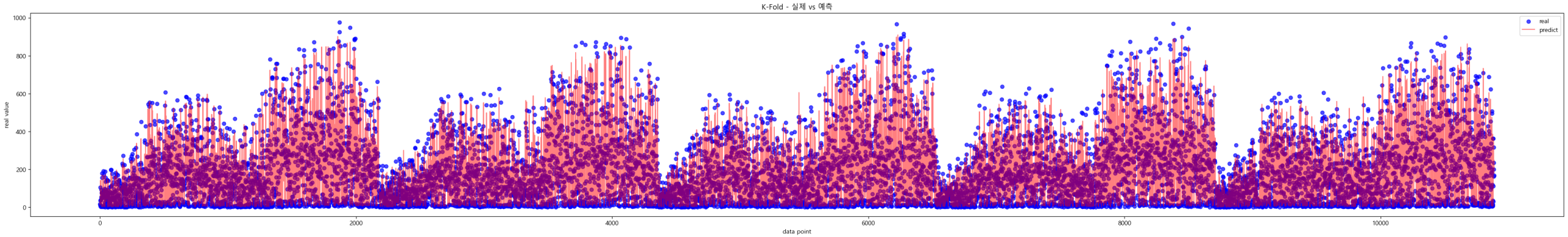
과적합

✓ K-Fold

K-Fold 사용 X	
R^2	RMSE
0.9437	42.8511
K-Fold 사용	
R^2	RMSE
0.9427	43.3603

결정계수 값과 RMSE 값이 기존 값과
크게 차이 나지 않으므로 과적합 되지 않음을 확인했다

```
kf = KFold(n_splits=5, shuffle=True, random_state=42)
rfrk = RandomForestRegressor(random_state=42, n_estimators=100)

y_test_all = []
y_pred_all = []

for train_index, test_index in kf.split(X11):
    Xrfr_train, Xrfr_test = X11.iloc[train_index], X11.iloc[test_index]
    yrfr_train, yrfr_test = y11.iloc[train_index], y11.iloc[test_index]

    rfrk.fit(Xrfr_train, yrfr_train)
    yrfr_pred = rfrk.predict(Xrfr_test)

    y_test_all.extend(yrfr_test)
    y_pred_all.extend(yrfr_pred)

resultrfr = pd.DataFrame({
    'real': y_test_all,
    'predict': y_pred_all
})

plt.figure(figsize=(45, 6))
plt.scatter(range(len(resultrfr)), resultrfr['real'], color='blue', label='real', alpha=.7)
plt.plot(resultrfr['predict'].values, color='red', label='predict', alpha=.5)
plt.title('K-Fold - 실제 vs 예측')
plt.xlabel('data point')
plt.ylabel('real value')
plt.legend()
plt.show()
```



6. 최적화

과적합

✓ 가지치기 & 깊이 제한

max_depth = 22
min_samples_leaf = 5

```
rfrcut = RandomForestRegressor(random_state=42, n_estimators=100, max_depth=22, min_samples_leaf=5)
rfrcut.fit(Xcut_train, ycut_train)
```

가지치기 & 깊이 제한 X	
R^2	RMSE
0.9437	42.8511
가지치기 & 깊이 제한 O	
R^2	RMSE
0.9293	48.3101

파라미터를 설정하지 않은 기본 값이 더 성능이 좋게 나왔기 때문에 파라미터는 기본 값으로 지정

---

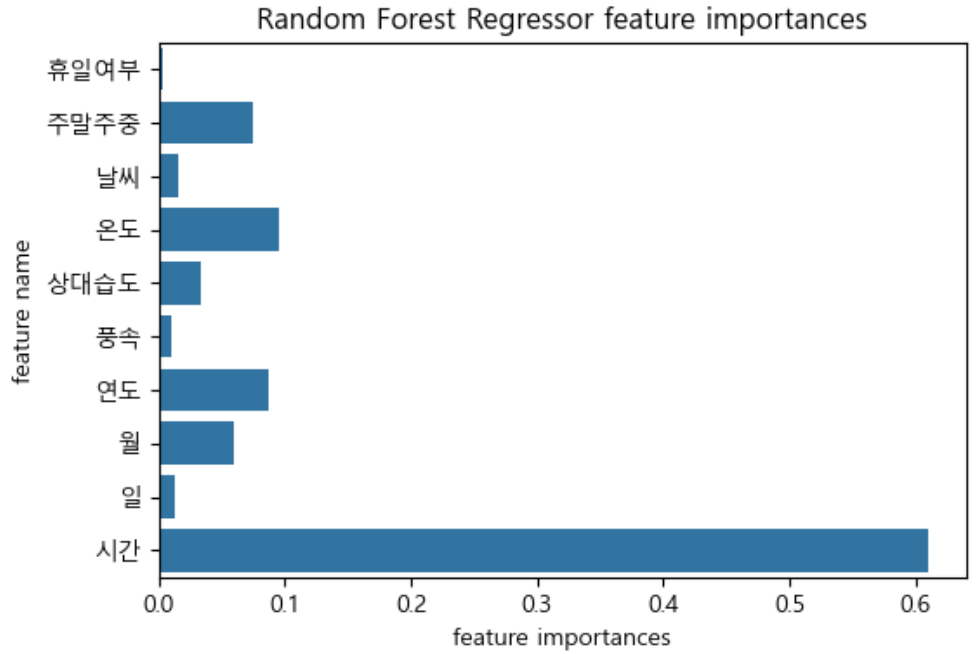
n_estimators=100, bootstrap=True, max_depth=None, max_feature='sqrt', min_samples_split=2, min_samples_leaf=1

6. 최적화

성능 향상

✓ 변수 선택

특성 중요도 값을 기준으로 변수를 선정하여 모델 학습




```
rfr_fi = rfr.feature_importances_
```

```
rfr_fiv.sort_values('특성 중요도', ascending=False)
```

특성	특성 중요도
시간	0.6096
온도	0.0948
연도	0.0872
주말주중	0.0750
월	0.0591
상대습도	0.0332
날씨	0.0155
일	0.0133
풍속	0.0100
휴일여부	0.0024

6. 최적화

성능 향상

✓ 변수 선택

0.05 > 0.03 > 0.01 순서로 cut

특성 중요도 > 0.05	시간, 온도, 연도, 주말주중, 월 / 대여횟수
특성 중요도 > 0.03	시간, 온도, 연도, 주말주중, 월, 상대습도 / 대여횟수
특성 중요도 0.01	시간, 온도, 연도, 주말주중, 월, 상대습도, 일, 풍속 / 대여횟수

특성 중요도 > 0.05		특성 중요도 > 0.03		특성 중요도 > 0.01	
R^2	RMSE	R^2	RMSE	R^2	RMSE
0.8937	59.2351	0.9311	47.7001	0.9347	46.4340

변수를 모두 사용했을 때가 변수를 일부만 선택했을 때보다 성능이 좋으므로, 전체 변수를 사용

6. 최적화

성능 향상

✓ 이상치 제거

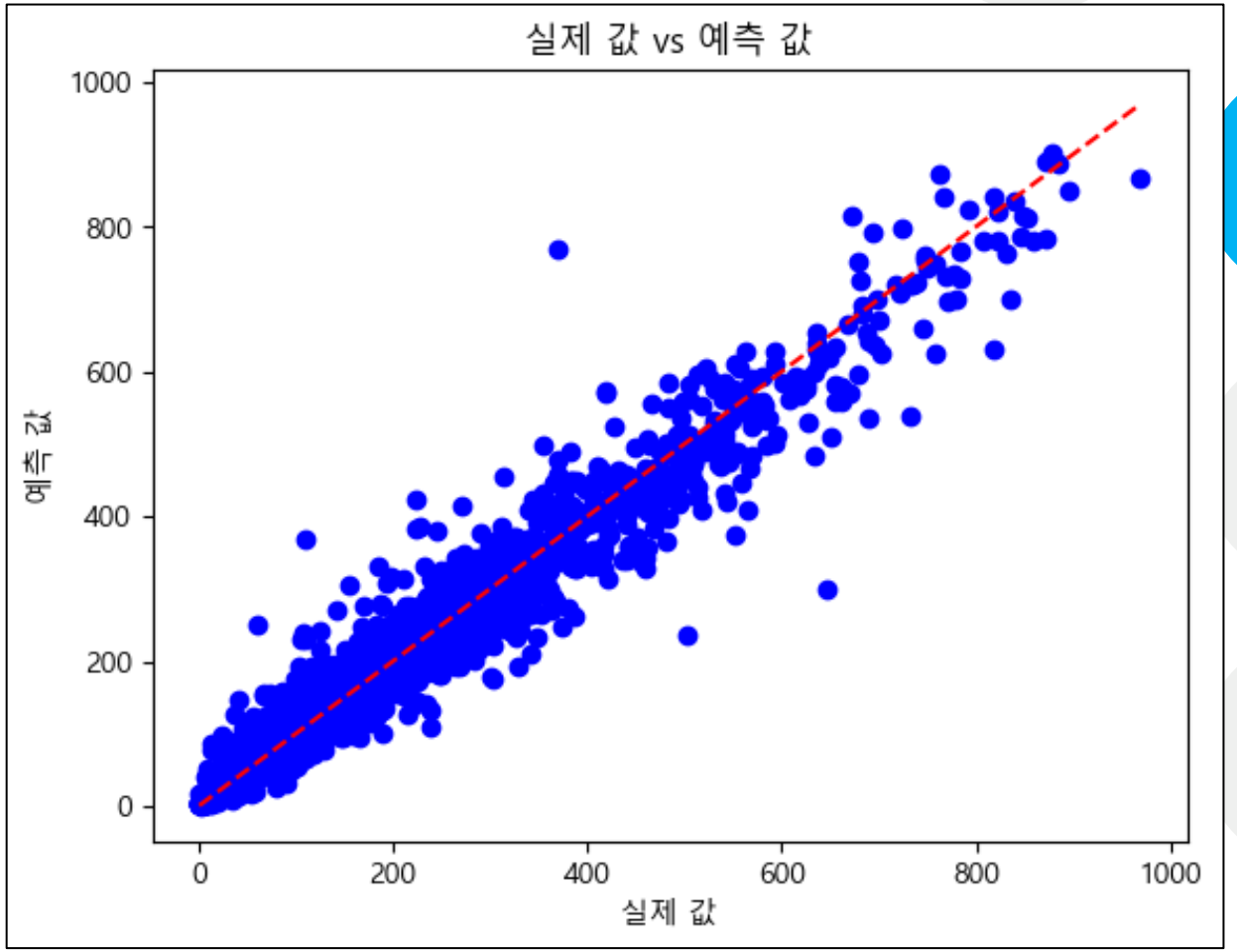
```
df_out = df_time.copy()
df_out = df_out[df_out['풍속'] <= 30]
df_out
```

박스 플롯에서 윗 수염을 벗어나는 부분을 제거

✓ 결정계수가 이상치를 제거하지 않았을 때보다 높다

✓ RMSE가 이상치를 제거하지 않았을 때보다 작다

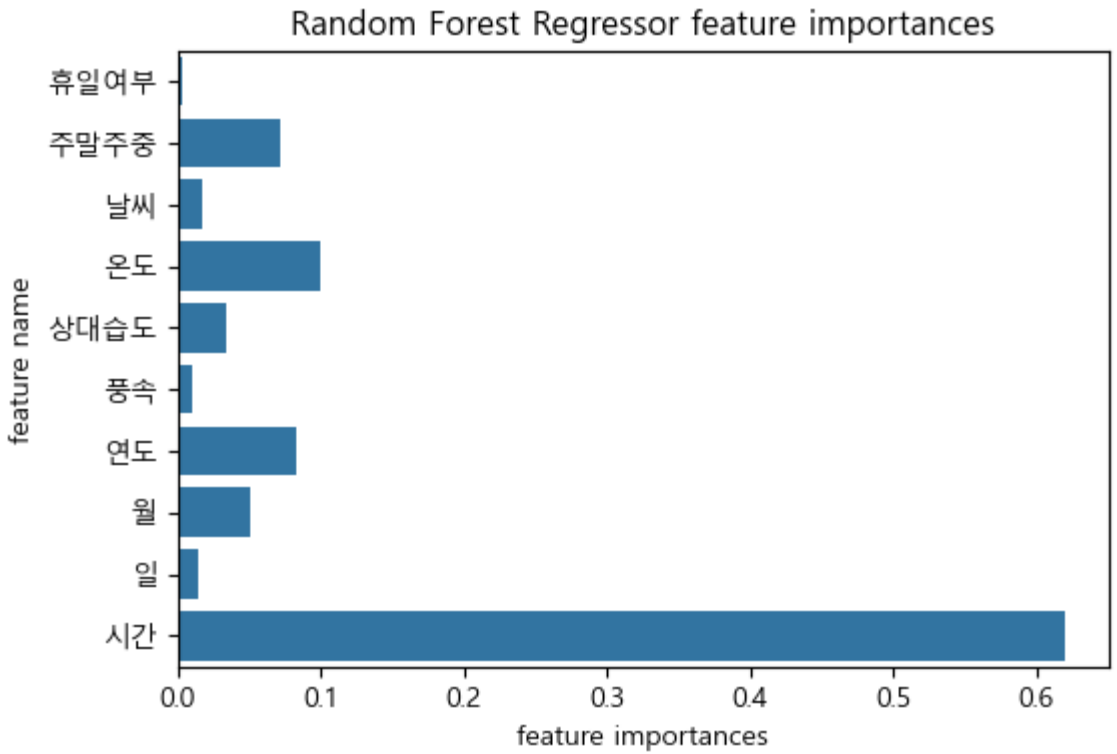
✓ train과 test의 결정계수 차이가 크지 않다



train R^2	test R^2	RMSE
0.9916	0.9501	40.4980

→ 이상치를 제거하고, 변수를 모두 사용하고, 하이퍼파라미터 튜닝을 하지 않은 모델이 가장 성능이 좋다

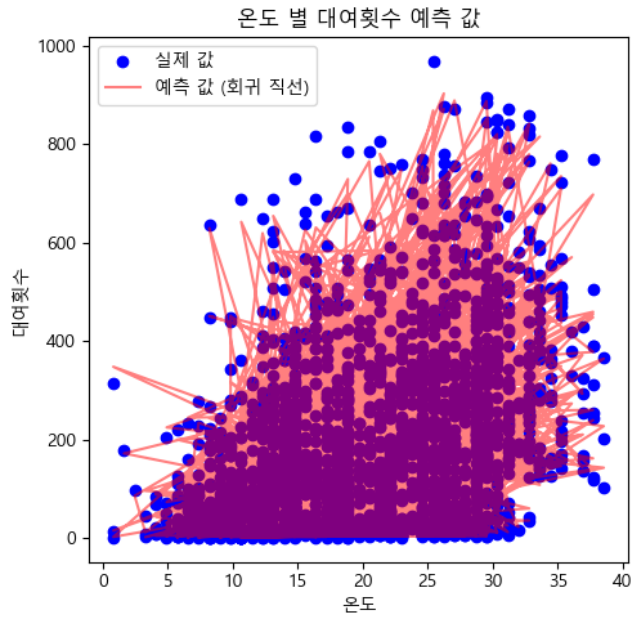
7. 인사이트 도출

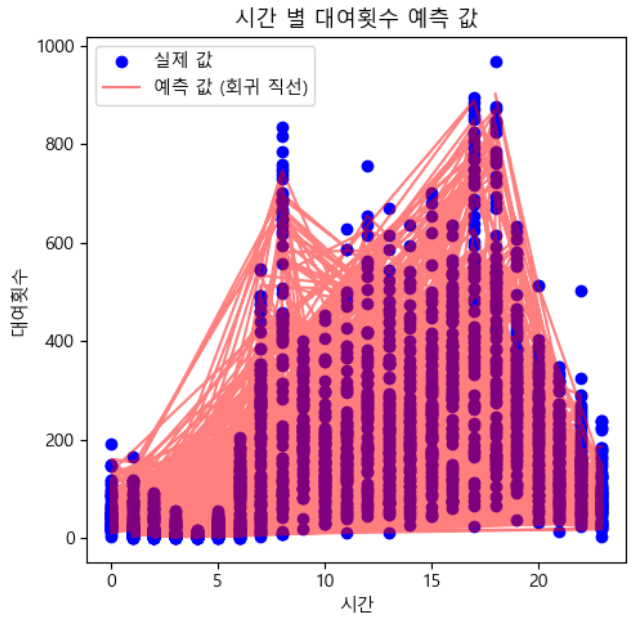


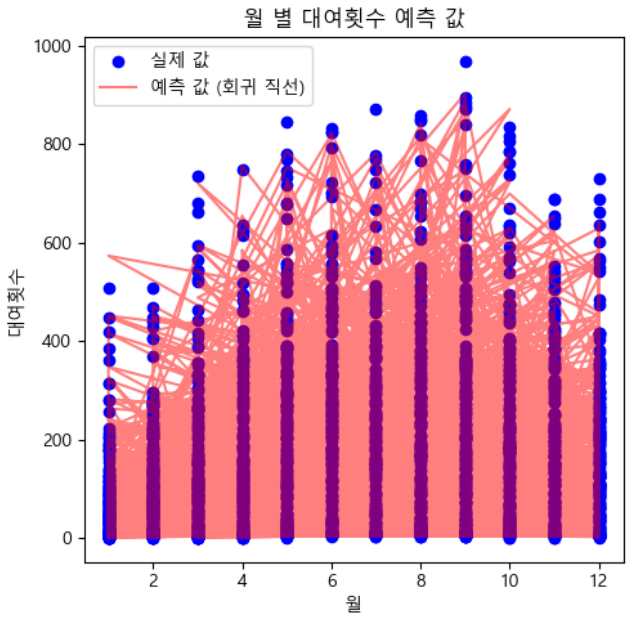
최종 모델의 특성 중요도를 그래프로 출력하니
시간, 온도, 주말주중, 연도, 월이
중요한 특성으로 출력되었다

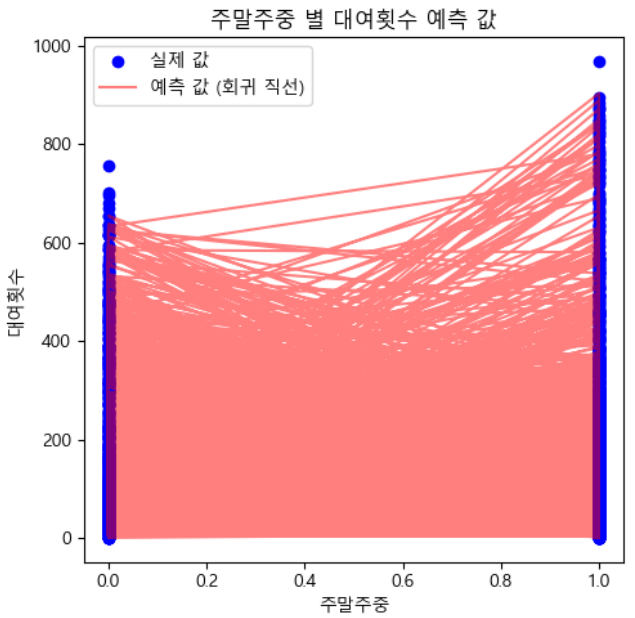
연도는 2011년과 2012년으로 고정되어
있으므로 제외하고, 나머지 네 가지 특성 별
대여 횟수의 그래프를 출력해봤다

7. 인사이트 도출









✓ 시간이 7-8시, 17-18시 즉, 출퇴근 시간에 대여 횟수가 증가한다
✓ 11-2월 즉, 겨울에 대여 횟수가 감소한다
✓ 주말보다 주중에 대여 횟수가 증가한다

→ 봄 ~ 가을, 주중, 출퇴근 시간에 자전거를 더 많이 비치한다면 자전거 대여 이용자의 만족도가 증가할 것이다.

그러나 시간을 제외한 나머지 변수들은 상관관계도 특성 중요도도 높지 않으므로
해당 모델의 결과를 무조건적으로 신뢰하여 판단한다면 엉뚱한 결과가 도출될 수 있음을 주의

분류 분석

: 미등록 대여 횟수와 등록 대여 횟수를 비교

1. 종속 변수 생성

```
df_time['등록여부'] = (df_time['미등록대여횟수'] < df_time['등록대여횟수'])
```

시간대 별로 등록 대여 횟수가 미등록 대여 횟수보다 많은 경우 True,
적은 경우를 False로 표시하는 등록 여부 변수 생성

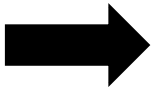
2. 대여 횟수 변수 제거

```
df_reg = df_time.drop(['미등록대여횟수', '등록대여횟수', '대여횟수'], axis=1)
```

대여 횟수, 미등록 대여 횟수, 등록 대여 횟수 변수는 분류 분석을 하려는
입장에서 알 수 없는 내용이므로 제거

3. 표준화

```
scale_features = ['온도', '체감온도', '상대습도', '풍속']

std = StandardScaler()

df_std = Xc.copy()
df_std[scale_features] = std.fit_transform(df_reg[scale_features])
```



온도	체감온도	상대습도	풍속
-1.333661	-1.092737	0.993213	-1.567754
-1.438907	-1.182421	0.941249	-1.567754
-1.438907	-1.182421	0.941249	-1.567754
-1.333661	-1.092737	0.681430	-1.567754
-1.333661	-1.092737	0.681430	-1.567754

수치적 특성을 가지는 네 변수가
서로 다른 범위를 가지고
있으므로 표준화

4. 모델링

```
models = {
    'Random Forest': RandomForestClassifier(random_state=42),
    'Ada Boost': AdaBoostClassifier(random_state=42, algorithm='SAMME'),
    'Voting': VotingClassifier(
        estimators=[
            ('logistic', LogisticRegression(random_state=42, max_iter=5000)),
            ('dt', DecisionTreeClassifier(random_state=42))
        ],
        voting='soft'
    )
}
```

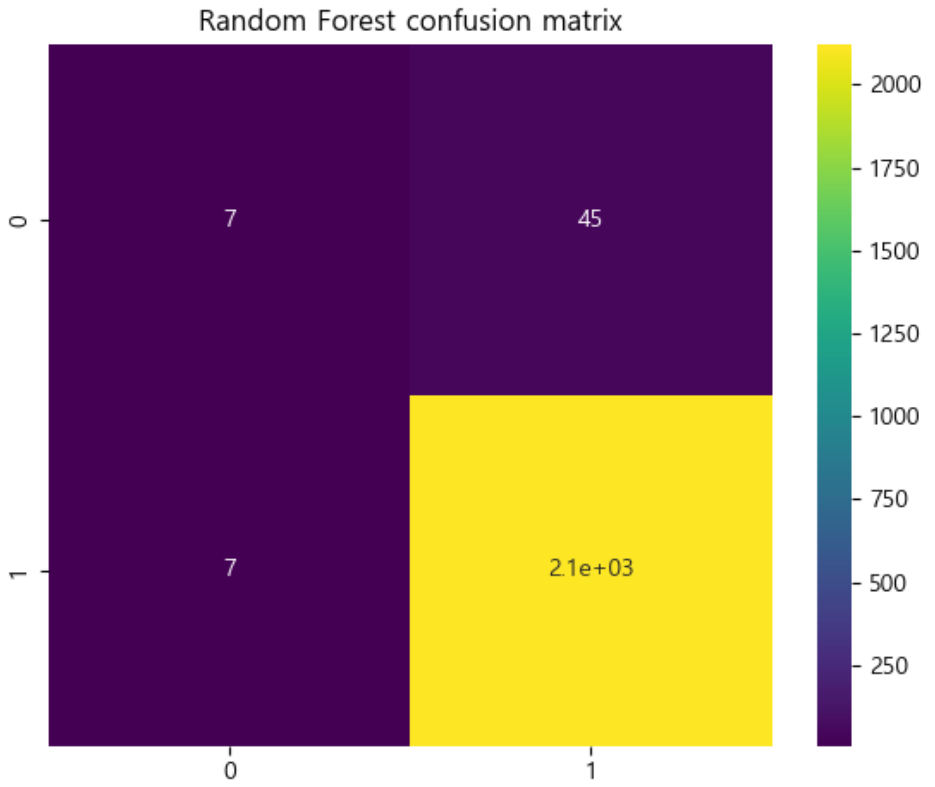
Voting, Bagging, Boosting 등
다양한 모델을 사용 후 비교

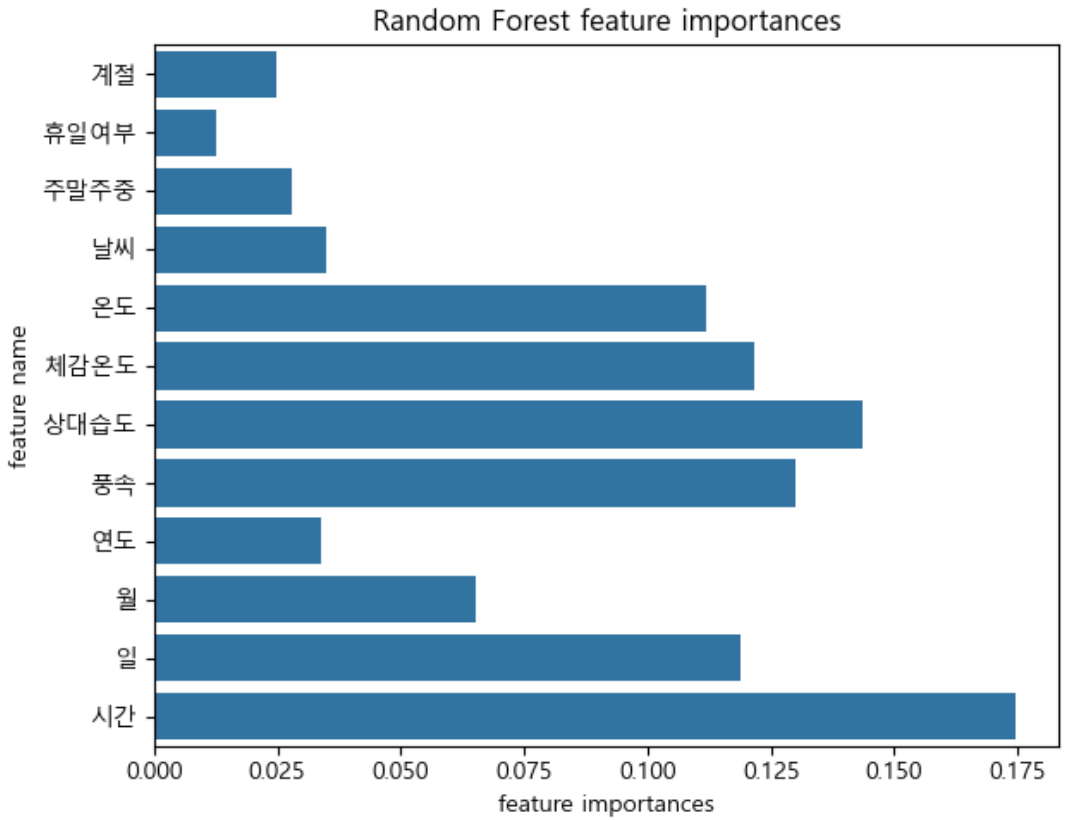
```
for name, model in models.items():
    model.fit(Xc_train, yc_train)
    yc_pred = model.predict(Xc_test)
    print(f'---------------------------{name}---------------------------')
    print(f'{name} 정확도: {accuracy_score(yc_test, yc_pred):.3f}, 정밀도: {precision_score(yc_test, yc_pred):.3f}, 재현율: {recall_score(yc_test, yc_pred):.3f}, f1: {f1_score(yc_test, yc_pred):.3f}')

    plt.figure(figsize=(12, 5))

    plt.subplot(1, 2, 1)
    sns.heatmap(confusion_matrix(yc_test, yc_pred), cmap='viridis', annot=True)
    plt.title(f'{name} confusion matrix')

    if name != 'Voting':
        cfeatures = Xc.columns
        feature_importances = model.feature_importances_

        plt.subplot(1, 2, 2)
        sns.barplot(x=feature_importances, y=cfeatures)
        plt.title(f'{name} feature importances')
        plt.xlabel('feature importances')
        plt.ylabel('feature name')

    plt.tight_layout()
    plt.show()
```

반복문을 통해 dictionary에 저장한 모델들을
순차적으로 학습, 예측 및 평가 수행

Random Forest Classifier

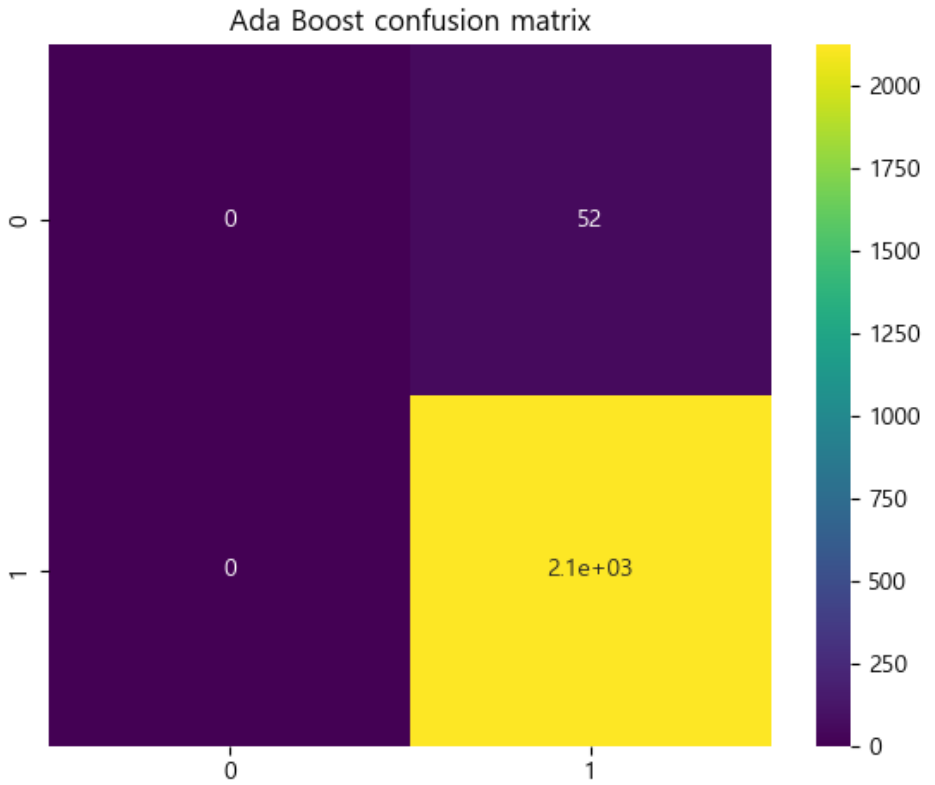
정확도	정밀도	재현율	F1
0.976	0.979	0.997	0.988

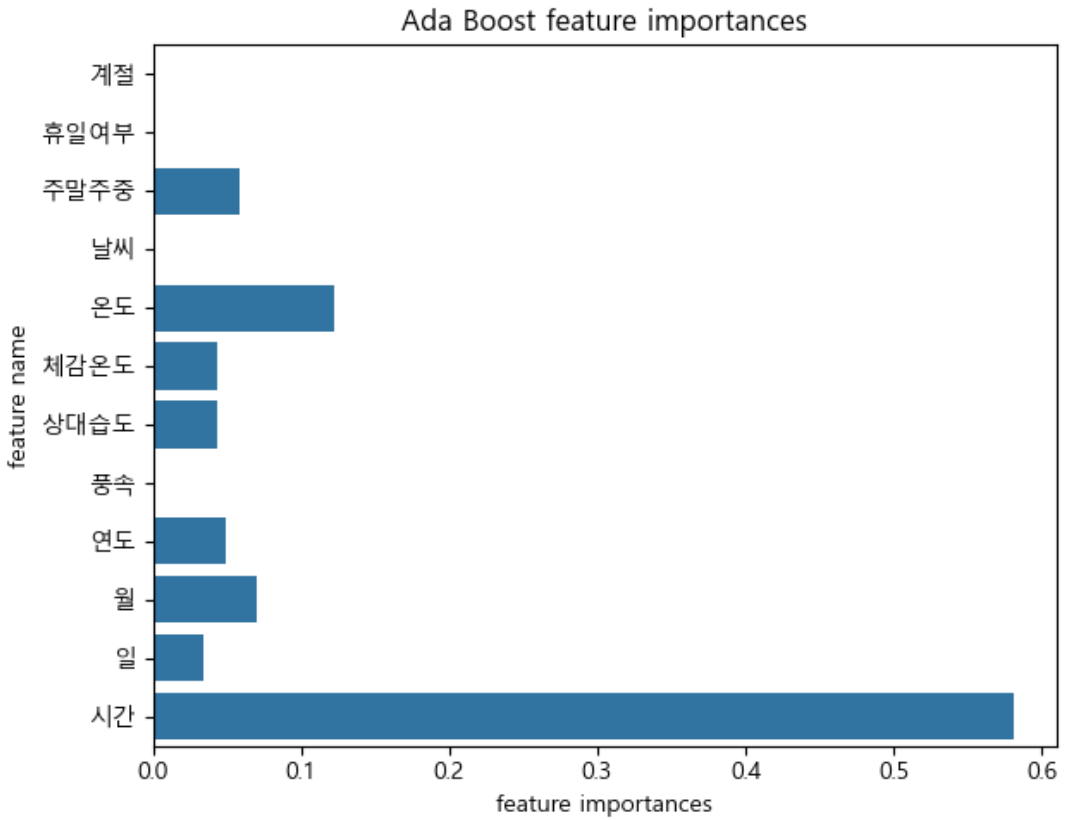




Ada Boost Classifier

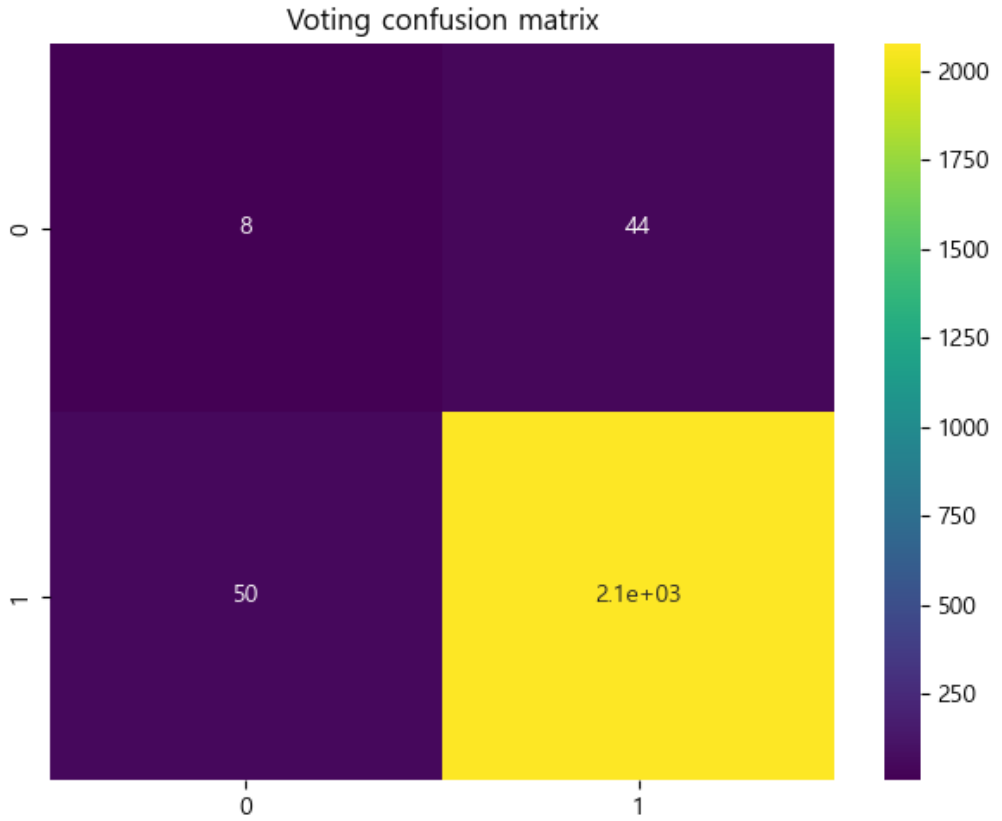
정확도	정밀도	재현율	F1
0.976	0.976	1.000	0.988





Voting Classifier

정확도	정밀도	재현율	F1
0.957	0.979	0.976	0.978



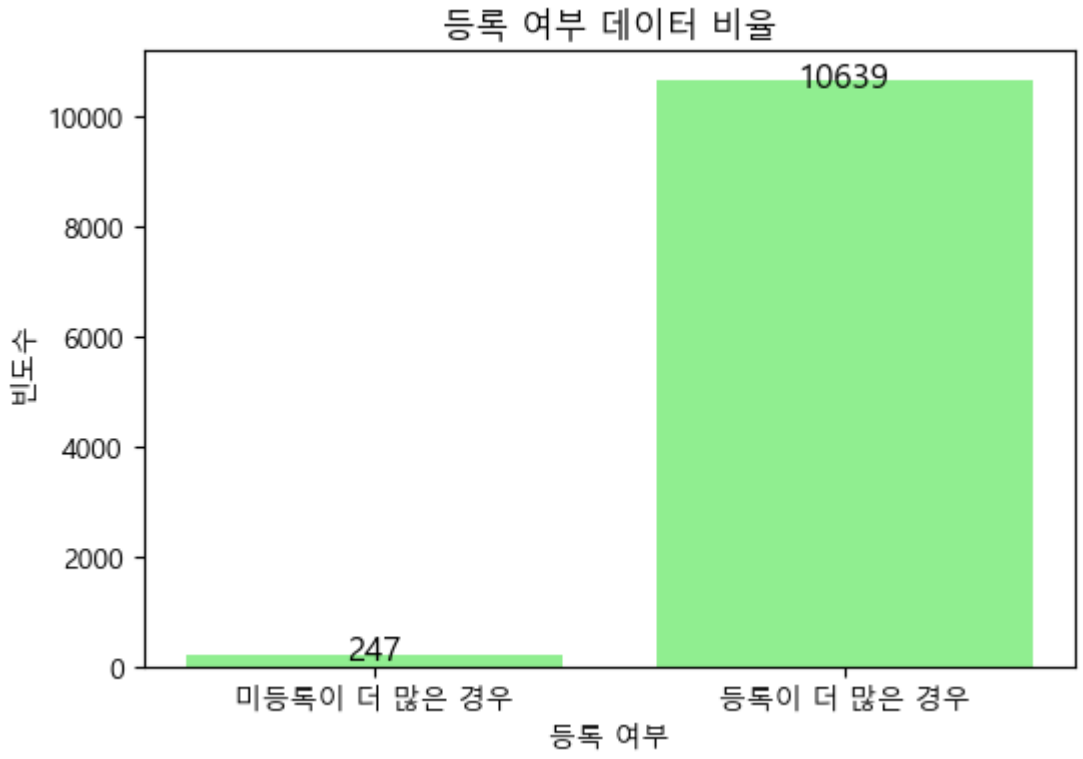
5. 평가

세 가지 모델 중 Voting Classifier의 성능이 가장 떨어지고,
Random Forest Classifier은 정밀도가, Ada Boost Classifier은 재현율이 높다

그러나 Ada Boost의 Confusion Matrix를 봤을 때,
0, False로 예측된 값이 전혀 없으므로 모든 값을 True로 예측한 것으로 보인다

데이터 불균형이 의심되므로 종속 변수의 값 비율을 출력했다

```
count = df_reg['등록여부'].value_counts()

fig, ax = plt.subplots(figsize=(6, 4))
bars = ax.bar(count.index, count.values, color='lightgreen')
ax.set_xticks([0, 1])
ax.set_xticklabels(['미등록이 더 많은 경우', '등록이 더 많은 경우'])
ax.bar_label(bars, padding=-5, fontsize=12)
ax.set_title('등록 여부 데이터 비율')
ax.set_xlabel('등록 여부')
ax.set_ylabel('빈도수')

plt.show()
```



미등록이 더 많은 경우 : 등록이 더 많은 경우는
대략 1:43으로 엄청난 불균형이 나타났다

6. 최적화

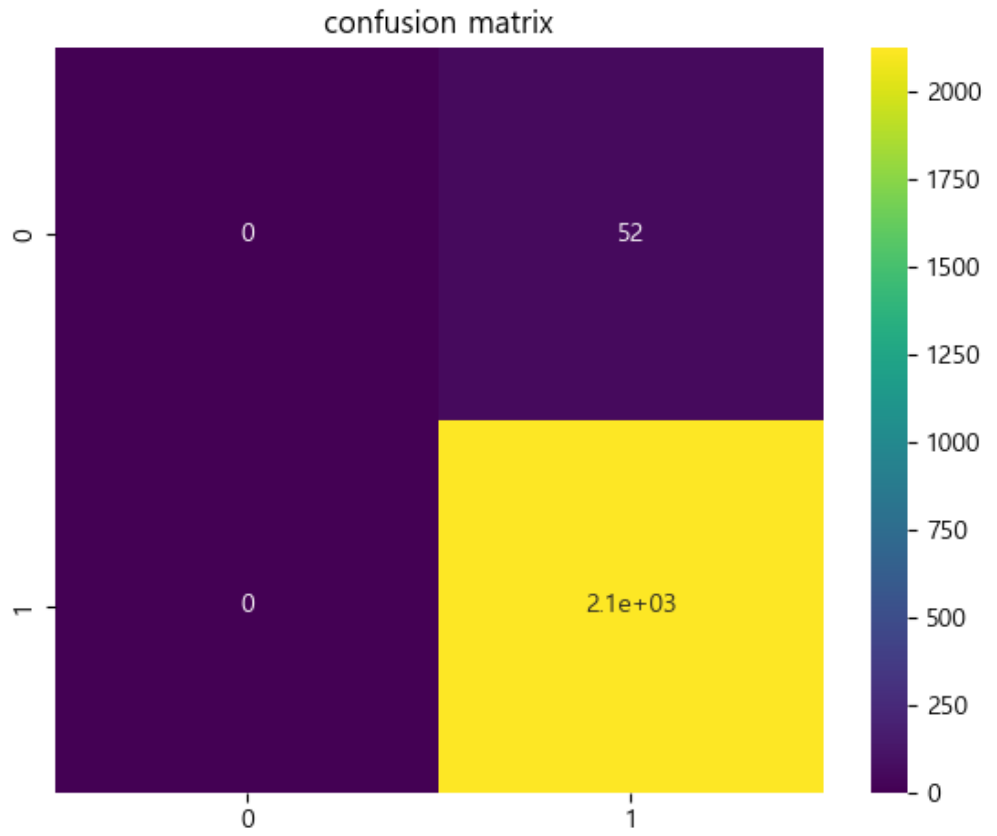
Random Forest Classifier은 데이터 불균형에도 False 값을 예측했으므로,
Random Forest Classifier로 최적화를 시도

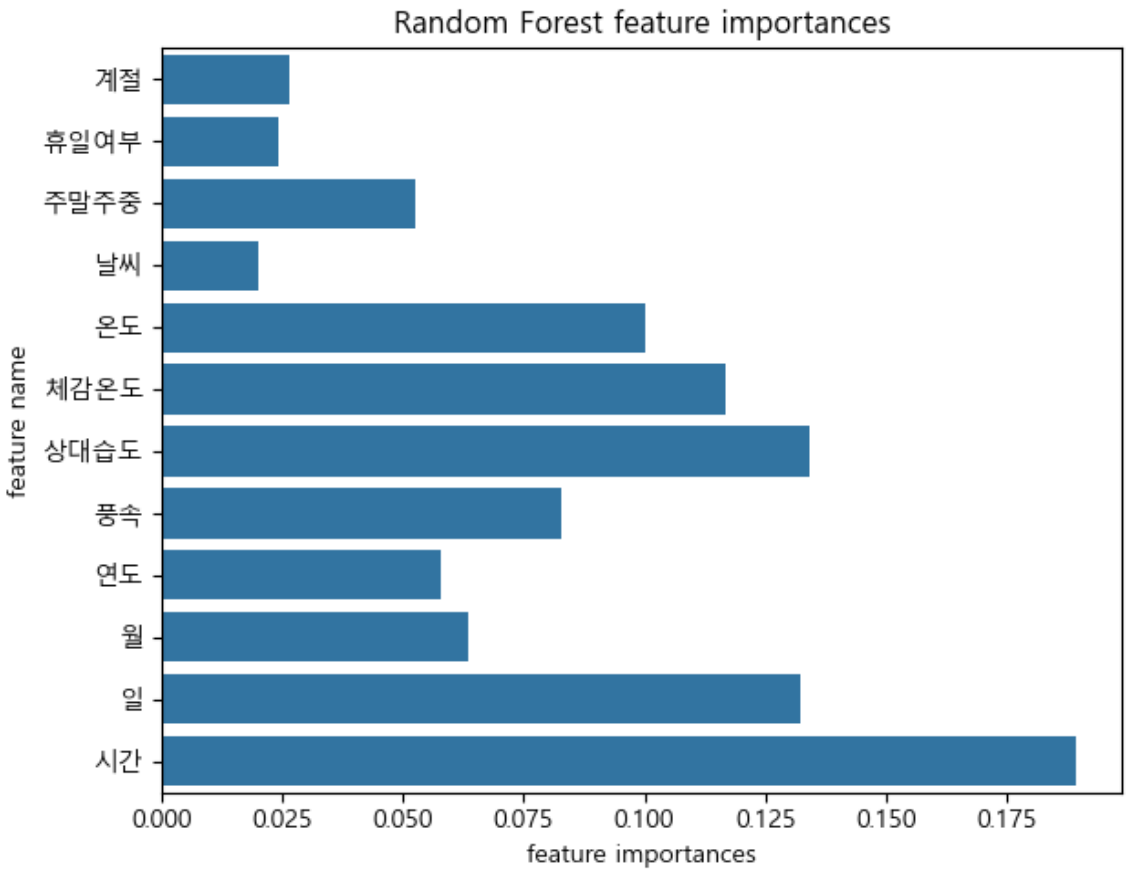
```
rf = RandomForestClassifier(random_state=42)

grid_prams = {
    'n_estimators': [100, 150],
    'max_depth': [3, 5, 7, 10],
    'min_samples_split': [3, 5],
    'min_samples_leaf': [5, 10]
}

grid_search = GridSearchCV(estimator=rf, param_grid=grid_prams, cv=5, scoring='accuracy')
grid_search.fit(Xc_train, yc_train)

print(f'Best Parameters: {grid_search.best_params_}')
print(f'Best Score: {grid_search.best_score_:.3f}')

best_model = grid_search.best_estimator_
yr_pred = best_model.predict(Xc_test)
```

Grid Search를 통해 최적의 하이퍼파라미터를 탐색했다

max_depth: 10, min_samples_leaf: 5, min_sample_split: 3, n_estimators: 10			
정확도	정밀도	재현율	F1
0.976	0.976	1.000	0.988

→ 정밀도는 떨어졌으나 재현율이 상승했다





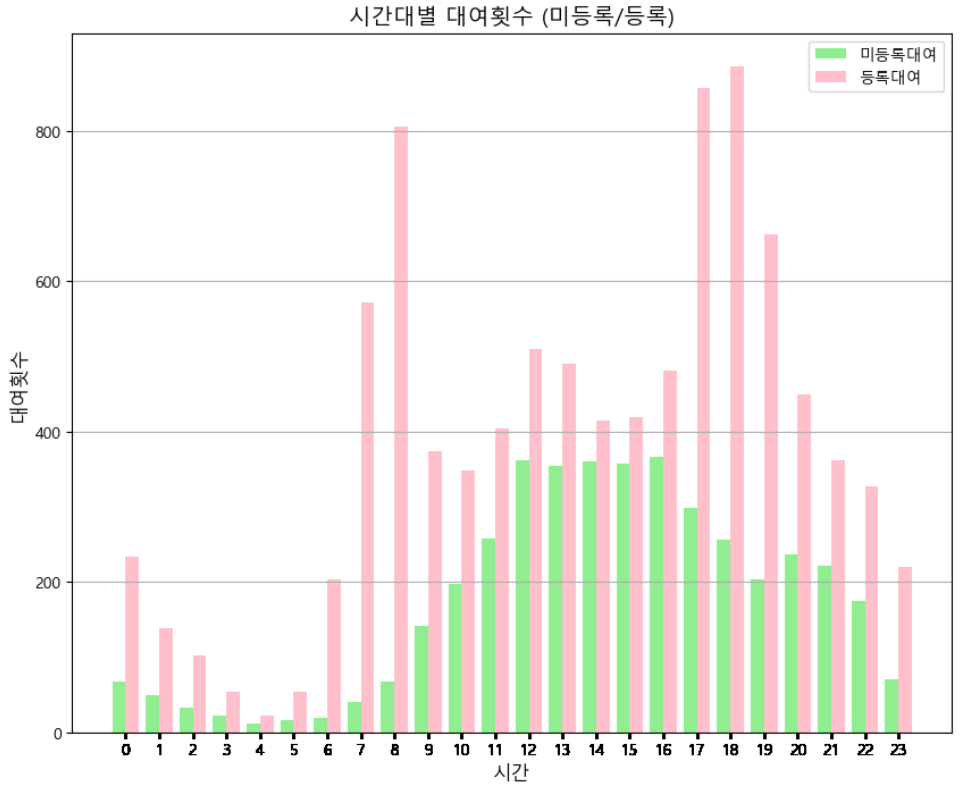
그러나 confusion matrix를 출력했을 때, Ada Boost Classifier와 마찬가지로 모든 값을 False로 예측했다
특성 중요도가 0.05 이상인 특성들만으로 다시 학습, 예측, 평가를 수행했지만 결과는 같았다

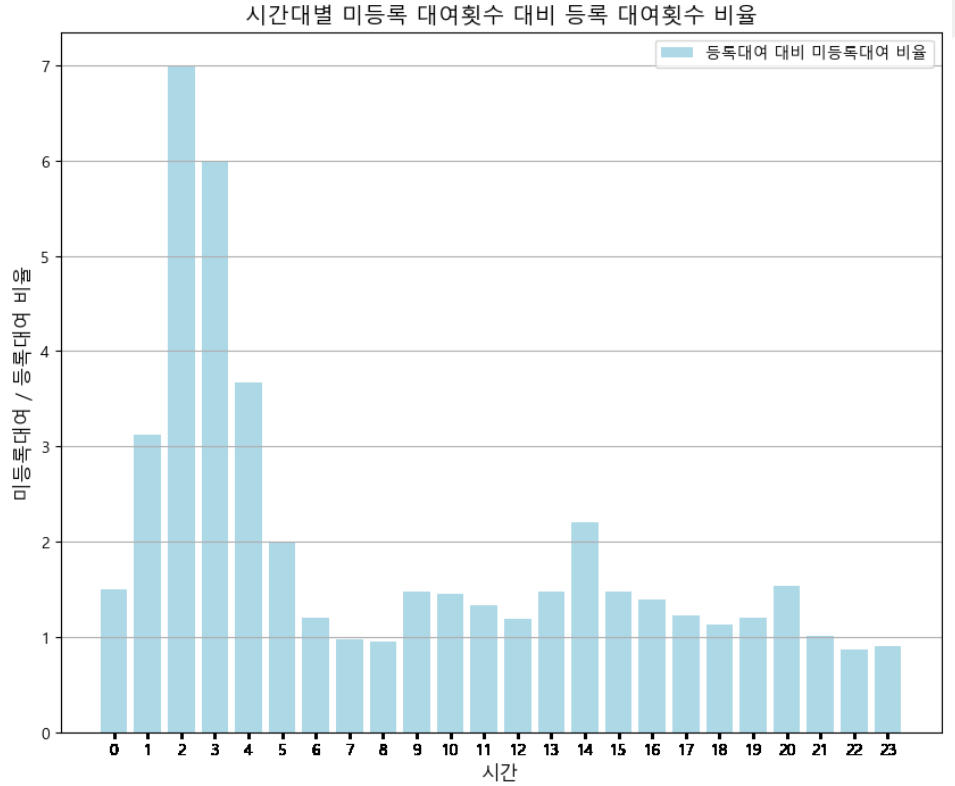
```
refeatures = fi[fi['특성 중요도'] >= 0.05]['특성'].tolist()
refeatures
```

```
Best Parameters: {'max_depth': 7, 'min_samples_leaf': 5, 'min_samples_split': 3, 'n_estimators': 150}
Best Score: 0.978
정확도: 0.976, 정밀도: 0.976, 재현율: 1.000, f1: 0.988
```

7. 인사이트 도출

특성 중요도가 시간이 가장 높으므로 시간대 별 대여 횟수와 미등록/등록 비율을 그래프로 출력했다





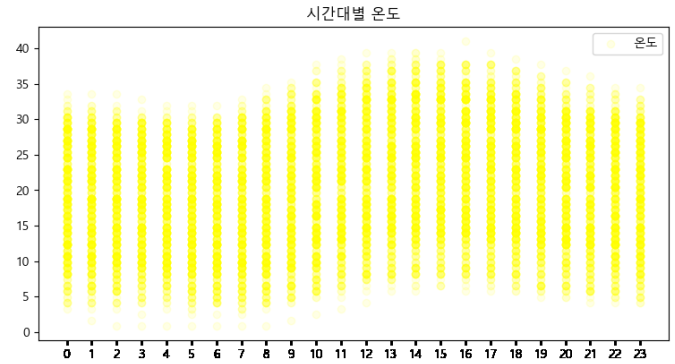

```
plt.bar(df_time['시간'] - 0.2, df_time['미등록대여횟수'], width=0.4, label='미등록대여', color='lightgreen')
plt.bar(df_time['시간'] + 0.2, df_time['등록대여횟수'], width=0.4, label='등록대여', color='pink')
```

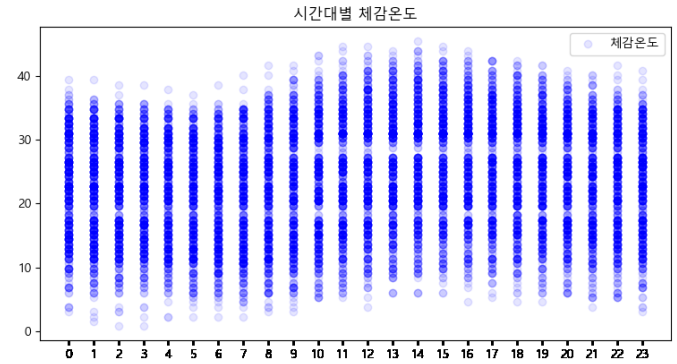
```
df_time['비율'] = df_time['미등록대여횟수'] / df_time['등록대여횟수']
```

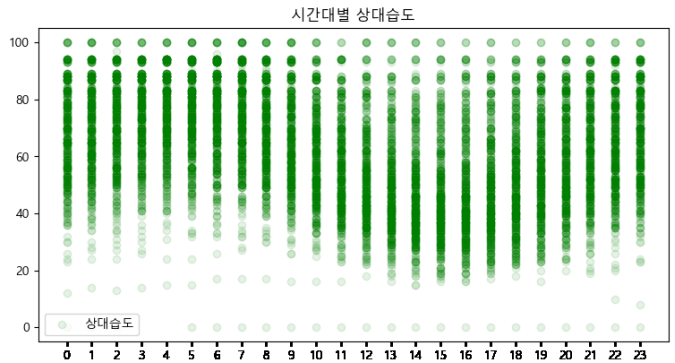
→ 미등록자의 대여 횟수가 많은 구간은 12-16시이고, 미등록자의 비율이 큰 시간대는 1-4시이다

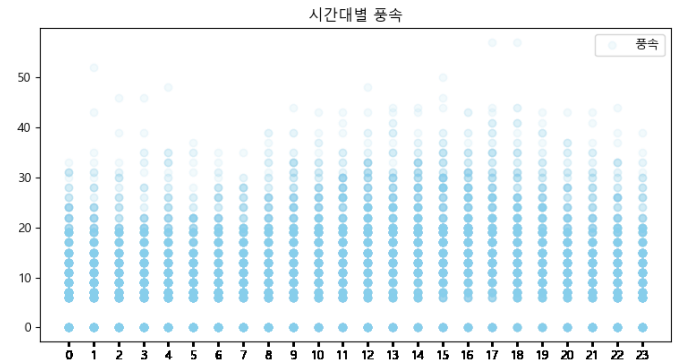
7. 인사이트 도출

해당 시간대의 환경을 찾기 위해 시간대 별 날씨 특성을 그래프로 출력했다









→ 미등록자의 이용이 많은 시간대는 온도가 높고, 습도가 낮고, 풍속은 보통 정도
→ 미등록자의 비율이 높은 시간대에는 온도가 낮고 습도가 높고 풍속은 낮다

7. 결론

- ✓ 미등록자 이용이 많은 시간대에는 좋은 날씨에 야외 활동을 하러 나온 사람들이 많을 것이라고 추측된다 → 간편한 회원 가입 시스템 제공, 특정 시간대 프로모션 등을 생각해볼 수 있다

- ✓ 미등록자 비율이 많은 시간대에는 새벽에 빠르게 집에 들어가려는 사람들이 많을 것이라고 추측된다 → 회원 가입 시 더 빠르고 편리하게 이용할 수 있음을 강조하여 마케팅에 활용 등을 생각해볼 수 있다

- ✓ 등록 대여 횟수가 미등록 대여 횟수보다 많을 경우의 조건을 찾기 위한 예측은 종속 변수의 데이터 불균형으로 인해 모델 학습 및 예측이 어려움을 확인

- ✓ 전체적인 평가 기준 값들은 높았으나 모델이 거의 모든 값을 True로 예측해 분별력이 없다고 판단

- ✓ 해당 종속 변수를 꼭 예측해야 한다면 Random Forest Classifier를 사용하고 parameter는 기본 값으로 수행하는 것을 추천하지만, 예측의 의미는 크게 없다고 생각된다

전체 코드

GltHub Link