주차 수요 예측

단계3 : 모델링

0.미션

- 1) 모델링
  - 4개 이상의 알고리즘을 이용하여 모델 생성
  - 각 모델에 대해 성능 튜닝 수행
  - 성능 평가 및 비교
    - 결과를 데이터프레임으로 정리하여 비교.
    - 성능 비교를 통해 최적의 모델 선정
    - 성능 가이드
      - MAE : 115~178
      - MAPE : 0.69~1.51
- 2) 파이프라인모델링
  - 함수 생성
    - Input : 새로운 데이터를 입력 받아서
    - 전처리를 수행한 후
    - 선정된 모델로 예측
    - Output : 예측 결과

1.환경설정

- 세부 요구사항
  - 경로 설정 : 다음의 두가지 방법 중 하나를 선택하여 폴더를 준비하고 데이터를 로딩하시오.
    - 1) 로컬 수행(Ananconda)
      - 제공된 압축파일을 다운받아 압축을 풀고
      - anaconda의 root directory(보통 C:/Users/< ID > 에 project 폴더를 만들고, 복사해 넣습니다.
    - 2) 구글콜랩
      - 구글 드라이브 바로 밑에 project 폴더를 만들고,
      - 데이터 파일을 복사해 넣습니다.
  - 라이브러리 설치 및 로딩
    - requirements.txt 파일로 부터 라이브러리 설치
  - 기본적으로 필요한 라이브러리를 import 하도록 코드가 작성되어 있습니다.

- 필요하다고 판단되는 라이브러리를 추가하세요.

(1) 경로 설정

1) 로컬 수행(Anaconda)

- project 폴더에 필요한 파일들을 넣고, 본 파일을 열었다면, 별도 경로 지정이 필요하지 않습니다.

In [77]:
```
path = 'C:/Users/User/program/mini_pjt/mini_3/실습파일_에이블러용/데이터/'
```

2) 구글 콜랩 수행

- 구글 드라이브 연결

In [78]:
```
# from google.colab import drive
# drive.mount('/content/drive')
```

In [79]:
```
#path = '/content/drive/MyDrive/project/'
```

(2) 라이브러리 설치 및 불러오기

1) 설치

- requirements.txt 파일을 아래 위치에 두고 다음 코드를 실행하시오.
  - 로컬 : 다음 코드셀 실행
  - 구글콜랩 : requirements.txt 파일을 왼쪽 [파일]탭에 복사해 넣고 다음 코드셀 실행

In [80]:
```
#!pip install -r requirements.txt
```

2) 라이브러리 로딩

- **세부 요구사항**
  - 기본적으로 필요한 라이브러리를 import 하도록 코드가 작성되어 있습니다.
  - 필요하다고 판단되는 라이브러리를 추가하세요.

In [81]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

import joblib

# 필요한 라이브러리 로딩
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.ensemble import RandomForestRegressor
from sklearn.tree import DecisionTreeRegressor
```

```
from sklearn.linear_model import LinearRegression
from sklearn.neighbors import KNeighborsRegressor
from sklearn.metrics import *
```

3) 함수 생성

- 모델 실제 vs 예측결과 비교 그래프

In [82]:
```
def model_plot(y, pred) :
    plt.figure(figsize = (12,8))
    plt.scatter(y, pred, alpha=0.4)

    x_l = np.linspace( y.min(), y.max(), 100)
    plt.plot(x_l, x_l, color = 'black', alpha=0.4)

    plt.xlabel("Actual")
    plt.ylabel("Predicted")
    plt.grid()
    plt.show()
```

(3) 데이터 불러오기

- **세부 요구사항**
  - 탐색적 데이터분석 단계에서 저장한 파일을 불러옵니다.

In [83]:
```
train = joblib.load(path + 'train2.pkl')
```

In [84]:
```
train.head()
```

Out[84]:

	단지코드	총세대수	전용면적별세대수	건물형태	난방방식	전용면적	공급면적(공용)	임대보증금	임대료	실차량수	총면적
0	C0001	78	35	계단식	가스난방	51.89	19.2603	50758000	620370	109	1816.15
1	C0001	78	43	계단식	가스난방	59.93	22.2446	63166000	665490	109	2576.99
2	C0002	35	26	복도식	가스난방	27.75	16.5375	63062000	458640	35	721.50
3	C0002	35	9	복도식	가스난방	29.08	17.3302	63062000	481560	35	261.72
4	C0003	88	7	계단식	가스난방	59.47	21.9462	72190000	586540	88	416.29

In [85]:
```
train.columns
```

Out[85]:
```
Index(['단지코드', '총세대수', '전용면적별세대수', '건물형태', '난방방식', '전용면적', '공급면적(공용)', '임대보증금',
       '임대료', '실차량수', '총면적'],
      dtype='object')
```

2.모델링

- **세부 요구사항**
  - 모델링을 위한 전처리 : NaN 조치. 데이터 분할, 스케일링, 가변수화 등
  - 4개 이상의 알고리즘을 이용하여 모델 생성
  - 각 모델에 대해 성능 튜닝 수행
    - 선형회귀 : 릿지, 라쏘, 엘라스틱넷 중 1~2개 함께 사용하여 성능 비교
    - 랜덤포레스트 : 기본값으로 모델링 수행
    - 그외 알고리즘 : 그리드서치 튜닝으로 성능 최적화
  - 성능 비교를 통해 최적의 모델 선정
    - 검증 성능 가이드라인 : MAE : 120 내외

(1) 데이터전처리

- **세부 요구사항**
  - 모델링을 위한 전처리를 수행합니다.
    - x,y 분할(식별자 역할의 단지코드 칼럼 제거하고, x,y로 분할)
    - 단지코드 제거
    - 가변수화
    - 스케일링(필요시)
    - train : validation
      - 적절한 비율 사용하기
      - random_state 지정하기

1) x, y 분할

In [86]:
```
train.columns
```

Out[86]:
```
Index(['단지코드', '총세대수', '전용면적별세대수', '건물형태', '난방방식', '전용면적', '공급면적(공용)', '임대보증금',
       '임대료', '실차량수', '총면적'],
      dtype='object')
```

In [87]:
```
target = '실차량수'
```

In [88]:
```
train.drop('단지코드', axis=1, inplace=True)
```

In [89]:
```
x = train.drop(target, axis=1)
y = train.loc[:, target]
```

2) NaN 조치

In [90]:
```
# 1번 데이터 전처리에서 조치함
train.isna().sum()
```

Out[90]:
```
총세대수          0
전용면적별세대수     0
건물형태          0
난방방식          0
전용면적          0
공급면적(공용)     0
임대보증금         0
임대료           0
실차량수          0
총면적           0
dtype: int64
```

3) 가변수화

In [91]:
```
train
```

Out[91]:

	총세대수	전용면적별세대수	건물형태	난방방식	전용면적	공급면적(공용)	임대보증금	임대료	실차량수	총면적
0	78	35	계단식	가스난방	51.89	19.2603	50758000	620370	109	1816.15
1	78	43	계단식	가스난방	59.93	22.2446	63166000	665490	109	2576.99
2	35	26	복도식	가스난방	27.75	16.5375	63062000	458640	35	721.50
3	35	9	복도식	가스난방	29.08	17.3302	63062000	481560	35	261.72
4	88	7	계단식	가스난방	59.47	21.9462	72190000	586540	88	416.29
...	...	...	...	...	...	...	...	...	...	...
1152	956	956	복도식	가스난방	26.37	12.7500	9931000	134540	243	25209.72
1153	120	66	복도식	난방	24.83	15.1557	2129000	42350	47	1638.78
1154	120	54	복도식	난방	33.84	20.6553	2902000	57730	47	1827.36
1155	447	149	복도식	유류난방	26.37	13.3800	7134000	118880	78	3929.13
1156	447	298	복도식	유류난방	31.32	13.8500	8122000	131140	78	9333.36

1157 rows × 10 columns

In [92]:
```
drop_dumm = ['건물형태', '난방방식']
x = pd.get_dummies(x, columns=drop_dumm, drop_first=True, dtype=int)
```

In [93]:
```
x
```

Out[93]:

	총세대수	전용면적별세대수	전용면적	공급면적(공용)	임대보증금	임대료	총면적	건물형태_복도식	건물형태_혼합식	난방방식_난방	난방방식_유류난방
0	78	35	51.89	19.2603	50758000	620370	1816.15	0	0	0	0
1	78	43	59.93	22.2446	63166000	665490	2576.99	0	0	0	0
2	35	26	27.75	16.5375	63062000	458640	721.50	1	0	0	0
3	35	9	29.08	17.3302	63062000	481560	261.72	1	0	0	0
4	88	7	59.47	21.9462	72190000	586540	416.29	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...
1152	956	956	26.37	12.7500	9931000	134540	25209.72	1	0	0	0
1153	120	66	24.83	15.1557	2129000	42350	1638.78	1	0	1	0
1154	120	54	33.84	20.6553	2902000	57730	1827.36	1	0	1	0
1155	447	149	26.37	13.3800	7134000	118880	3929.13	1	0	0	1
1156	447	298	31.32	13.8500	8122000	131140	9333.36	1	0	0	1

1157 rows × 11 columns

In [94]:
```
# drop_cols = ['30이하', '40이하',      '50이하', '60이하', '70이하', '80이하', '90이하', '100
# x = x.drop(drop_cols, axis=1)
# x
```

4) train : val 분할

In [95]:
```
x_train, x_val, y_train, y_val = train_test_split(x, y, test_size=0.2, random_state=2024)
```

5) 스케일링

In [96]:
```
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()
scaler.fit(x_train)
x_train_s = scaler.transform(x_train)
x_val_s = scaler.transform(x_val)
```

(2) 알고리즘1

In [97]:
```
model_lr = LinearRegression()
model_rf = RandomForestRegressor()
model_knn = KNeighborsRegressor()
model_dt = DecisionTreeRegressor()
```

In [98]:
```
# MAE : 115~178
# MAPE : 0.69~1.51
result = {}
```

In [99]:
```
def modle_mk(modle_select, x_train, y_train, x_val, y_val):
    if modle_select == model_rf:
        name = 'Random Forest'

        param = {
        'max_depth': [1, 5, 10, 20, 50],
        'n_estimators' : [10, 30, 50, 100, 150, 200]
        }
        model = GridSearchCV(modle_select,   # 튜닝할 기본 모델
                             param,       # 테스트 대상 매개변수 범위
                             cv=5,        # K-Fold cv 개수
                             scoring='r2' # 평가지표 (회귀여서 r2)
                             )

    if modle_select == model_lr:
        name = 'LinearRegression'
        model = modle_select

    if modle_select == model_knn:
        name = 'KNeighborsRegressor'

        param = {
            'n_neighbors': [3, 5, 7, 9, 11], #이웃의 수를 결정
            'weights': ['uniform', 'distance'], # uniform'으로 설정하면 모든 이웃에 동일한 가
            'algorithm': ['auto', 'ball_tree', 'kd_tree', 'brute'], # 가장 가까운 이웃을 검색
            'p': [1, 2], # 맨하탄 거리('manhattan')(1)나 민코프스키 거리('minkowski')(2)를 사
        }

        model = GridSearchCV(modle_select,   # 튜닝할 기본 모델
                                 param,       # 테스트 대상 매개변수 범위
                                 cv=5,        # K-Fold cv 개수
                                 scoring='r2' # 평가지표 (회귀여서 r2)
                                 )

    if modle_select == model_dt:
        name = 'DecisionTreeRegressor'

        param = {
        'max_depth': [1, 5, 10, 20, 50],
        'max_features': ['auto', 'sqrt', 'log2'] #각 노드에서 분할에 사용될 특성의 최대 수를 결
                                                 # 높은 값을 설정하면 더 다양한 특성을 고려할 수
        }
        model = GridSearchCV(modle_select,   # 튜닝할 기본 모델
                             param,       # 테스트 대상 매개변수 범위
                             cv=5,        # K-Fold cv 개수
                             scoring='r2' # 평가지표 (회귀여서 r2)
                             )



    print("model_name:", name)
    model.fit(x_train, y_train)
    p1 = model.predict(x_val)

    print('MAE:', mean_absolute_error(y_val, p1)) # 낮을 수록 실제값에 가까움 # 낮을 수록 오차
    print('='*60)
    print('MSE:', mean_squared_error(y_val, p1)) # 낮을 수록 실제값에 가까움 #제곱 평균
    print('='*60)
    print('MSPE:', mean_absolute_percentage_error(y_val, p1)) # 낮을 수록 실제값에 가까움 #MSE
```

```
    print('='*60)
    print('R2:', r2_score(y_val, p1)) #결정계수 1에 가까울 수록 좋음
    print('='*60)
    print('최적의 파라미터:', model.best_params_)
    print('최고 성능:', model.best_score_)

    result[name] = model.best_score_
    return result
```

In [100…

```
# Random Forest
modle_mk(model_rf, x_train, y_train, x_val, y_val)
```

```
model_name: Random Forest
MAE: 73.0207144919855
============================================================
MSE: 13733.383367529115
============================================================
MSPE: 0.22241742161611952
============================================================
R2: 0.9091433325244744
============================================================
최적의 파라미터: {'max_depth': 20, 'n_estimators': 200}
최고 성능: 0.9048772919004978
```

Out[100]:

```
{'Random Forest': 0.9048772919004978}
```

(3) 알고리즘2

In [101…

```
# DecisionTreeRegressor
modle_mk(model_dt, x_train, y_train, x_val, y_val)
```

```
model_name: DecisionTreeRegressor
MAE: 160.9782268939336
============================================================
MSE: 63609.31366278293
============================================================
MSPE: 0.389779357793863
============================================================
R2: 0.5791765142542816
============================================================
최적의 파라미터: {'max_depth': 10, 'max_features': 'log2'}
최고 성능: 0.6909187547369653
```

```
C:\Users\User\anaconda3\Lib\site-packages\sklearn\model_selection\_validation.py:425: FitFaile
dWarning:
25 fits failed out of a total of 75.
The score on these train-test partitions for these parameters will be set to nan.
If these failures are not expected, you can try to debug them by setting error_score='raise'.

Below are more details about the failures:
--------------------------------------------------------------------------------
25 fits failed with the following error:
Traceback (most recent call last):
  File "C:\Users\User\anaconda3\Lib\site-packages\sklearn\model_selection\_validation.py", lin
e 732, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "C:\Users\User\anaconda3\Lib\site-packages\sklearn\base.py", line 1144, in wrapper
    estimator._validate_params()
  File "C:\Users\User\anaconda3\Lib\site-packages\sklearn\base.py", line 637, in _validate_par
ams
    validate_parameter_constraints(
  File "C:\Users\User\anaconda3\Lib\site-packages\sklearn\utils\_param_validation.py", line 9
5, in validate_parameter_constraints
    raise InvalidParameterError(
sklearn.utils._param_validation.InvalidParameterError: The 'max_features' parameter of Decisio
nTreeRegressor must be an int in the range [1, inf), a float in the range (0.0, 1.0], a str am
ong {'log2', 'sqrt'} or None. Got 'auto' instead.

  warnings.warn(some_fits_failed_message, FitFailedWarning)
C:\Users\User\anaconda3\Lib\site-packages\sklearn\model_selection\_search.py:976: UserWarning:
One or more of the test scores are non-finite: [       nan 0.20491731 0.12415843        nan 0.
45781378 0.58581497
        nan 0.54558013 0.69091875        nan 0.60297313 0.60897063
        nan 0.62916946 0.5904335 ]
  warnings.warn(
```

Out[101]:
```
{'Random Forest': 0.9048772919004978,
 'DecisionTreeRegressor': 0.6909187547369653}
```

(4) 알고리즘3

In [102…
```
#KNeighborsRegressor
modle_mk(model_knn, x_train_s, y_train, x_val_s, y_val)
```

```
model_name: KNeighborsRegressor
MAE: 110.18780189609153
============================================================
MSE: 30362.78847512728
============================================================
MSPE: 0.28054061362457305
============================================================
R2: 0.7991273015332205
============================================================
최적의 파라미터: {'algorithm': 'auto', 'n_neighbors': 5, 'p': 1, 'weights': 'distance'}
최고 성능: 0.8050606115132026
```

Out[102]:
```
{'Random Forest': 0.9048772919004978,
 'DecisionTreeRegressor': 0.6909187547369653,
 'KNeighborsRegressor': 0.8050606115132026}
```

(5) 알고리즘4

In [103…
```
# LinearRegression
modle_mk(model_lr, x_train, y_train, x_val, y_val)
```

```
model_name: LinearRegression
MAE: 160.21151504982086
============================================================
MSE: 47541.161287451345
============================================================
MSPE: 0.47274713452455047
============================================================
R2: 0.6854794359919306
============================================================
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Cell In[103], line 2
      1 # LinearRegression
----> 2 modle_mk(model_lr, x_train, y_train, x_val, y_val)

Cell In[99], line 63, in modle_mk(modle_select, x_train, y_train, x_val, y_val)
     61 print('R2:', r2_score(y_val, p1)) #결정계수 1에 가까울 수록 좋음
     62 print('='*60)
---> 63 print('최적의 파라미터:', model.best_params_)
     64 print('최고 성능:', model.best_score_)
     66 result[name] = model.best_score_

AttributeError: 'LinearRegression' object has no attribute 'best_params_'
```

(6) 성능결과 비교

- 세부 요구사항
  - 각 모델에 대해서 test 데이터로 성능 측정후, 데이터프레임으로 저장하고 비교한다.

In [27]:
```
result
```

Out[27]:
```
{'Random Forest': 0.9053422066779074,
 'DecisionTreeRegressor': 0.68564518663057,
 'KNeighborsRegressor': 0.828274861109211}
```

3.파이프라인 구축

- **세부요구사항**
  - new data : data02_test.csv 를 읽어서 저장
  - 파이프라인 함수를 생성
    - data pipeline 함수
    - ML pipeline 함수

(1) New Data 불러오기

- **세부요구사항**
  - test.xlsx 를 읽어서 new_data 이름으로 저장

- 해당 데이터는 최초 데이터와 동일한 구조입니다. 이 데이터를 이용하여 전처리와 예측을 수행해야 합니다.

In [104…
```
new_data = joblib.load(path + 'test3.pkl')
```

In [105…
```
new_data
```

Out[105]:

	단지코드	총세대수	전용면적별세대수	지역	건물형태	난방방식	전용면적	공급면적(공용)	임대보증금	임대료	실차량수	총면적
0	C0005	20	6	서울	복도식	가스난방	17.53	11.7251	50449000	263710	21	105.18
1	C0005	20	10	서울	복도식	가스난방	24.71	16.5275	52743000	321040	21	247.10
2	C0005	20	4	서울	복도식	가스난방	26.72	17.8720	53890000	332510	21	106.88
3	C0017	822	228	대구경북	계단식	난방	51.87	20.9266	29298000	411200	797	11826.36
4	C0017	822	56	대구경북	계단식	난방	59.85	24.1461	38550000	462600	797	3351.60
...	...	...	...	...	...	...	...	...	...	...	...	...
99	C0353	768	90	대전충남	복도식	난방	40.32	16.5100	8848000	122290	123	3628.80
100	C0360	588	98	서울	복도식	난방	51.37	21.5569	183228000	0	559	5034.26
101	C0360	588	186	서울	복도식	난방	51.39	21.5652	183228000	0	559	9558.54
102	C0360	588	102	서울	복도식	난방	59.76	25.0776	215057000	0	559	6095.52
103	C0360	588	202	서울	복도식	난방	59.80	25.0944	215057000	0	559	12079.60

104 rows × 12 columns

(2) 데이터 파이프라인 구축

- **세부요구사항**
  - data pipeline 함수를 생성합니다.
    - 입력 : new_data

- 처리 :
  - 1.데이터전처리, 2.탐색적 데이터분석 단계에서 수행했던 전처리 코드들을 순차적으로 처리합니다.
  - 모델링을 위한 전처리 : Target 제거, NaN조치, 가변수화, (스케일링) 등을 수행합니다.
- 출력 : 전처리 완료된 데이터 프레임

In [106…
```
drop_cols = ['단지코드', '지역']
new_data.drop(drop_cols, axis=1, inplace=True)
new_y_val = new_data.loc[:, '실차량수']
new_data.drop('실차량수', axis=1, inplace=True)
```

In [107…
```
new_data
```

Out[107]:

	총세대수	전용면적별세대수	건물형태	난방방식	전용면적	공급면적(공용)	임대보증금	임대료	총면적
0	20	6	복도식	가스난방	17.53	11.7251	50449000	263710	105.18
1	20	10	복도식	가스난방	24.71	16.5275	52743000	321040	247.10
2	20	4	복도식	가스난방	26.72	17.8720	53890000	332510	106.88
3	822	228	계단식	난방	51.87	20.9266	29298000	411200	11826.36
4	822	56	계단식	난방	59.85	24.1461	38550000	462600	3351.60
…	…	…	…	…	…	…	…	…	…
99	768	90	복도식	난방	40.32	16.5100	8848000	122290	3628.80
100	588	98	복도식	난방	51.37	21.5569	183228000	0	5034.26
101	588	186	복도식	난방	51.39	21.5652	183228000	0	9558.54
102	588	102	복도식	난방	59.76	25.0776	215057000	0	6095.52
103	588	202	복도식	난방	59.80	25.0944	215057000	0	12079.60

104 rows × 9 columns

In [108…
```
dumm_cols = ['건물형태',        '난방방식']
new_data = pd.get_dummies(new_data, columns=dumm_cols, drop_first=True, dtype=int)
new_data
```

Out[108]:

	총세대수	전용면적별세대수	전용면적	공급면적(공용)	임대보증금	임대료	총면적	건물형태_복도식	건물형태_혼합식	난방방식_난방	난방방식_유류난방
0	20	6	17.53	11.7251	50449000	263710	105.18	1	0	0	0
1	20	10	24.71	16.5275	52743000	321040	247.10	1	0	0	0
2	20	4	26.72	17.8720	53890000	332510	106.88	1	0	0	0
3	822	228	51.87	20.9266	29298000	411200	11826.36	0	0	1	0
4	822	56	59.85	24.1461	38550000	462600	3351.60	0	0	1	0
...	...	...	...	...	...	...	...	...	...	...	...
99	768	90	40.32	16.5100	8848000	122290	3628.80	1	0	1	0
100	588	98	51.37	21.5569	183228000	0	5034.26	1	0	1	0
101	588	186	51.39	21.5652	183228000	0	9558.54	1	0	1	0
102	588	102	59.76	25.0776	215057000	0	6095.52	1	0	1	0
103	588	202	59.80	25.0944	215057000	0	12079.60	1	0	1	0

104 rows × 11 columns

In [109…
```
new_data
```

Out[109]:

	총세대수	전용면적별세대수	전용면적	공급면적(공용)	임대보증금	임대료	총면적	건물형태_복도식	건물형태_혼합식	난방방식_난방	난방방식_유류난방
0	20	6	17.53	11.7251	50449000	263710	105.18	1	0	0	0
1	20	10	24.71	16.5275	52743000	321040	247.10	1	0	0	0
2	20	4	26.72	17.8720	53890000	332510	106.88	1	0	0	0
3	822	228	51.87	20.9266	29298000	411200	11826.36	0	0	1	0
4	822	56	59.85	24.1461	38550000	462600	3351.60	0	0	1	0
...	...	...	...	...	...	...	...	...	...	...	...
99	768	90	40.32	16.5100	8848000	122290	3628.80	1	0	1	0
100	588	98	51.37	21.5569	183228000	0	5034.26	1	0	1	0
101	588	186	51.39	21.5652	183228000	0	9558.54	1	0	1	0
102	588	102	59.76	25.0776	215057000	0	6095.52	1	0	1	0
103	588	202	59.80	25.0944	215057000	0	12079.60	1	0	1	0

104 rows × 11 columns

In [110…
```
x_train
```

Out[110]:

	총세대수	전용면적별세대수	전용면적	공급면적(공용)	임대보증금	임대료	총면적	건물형태_복도식	건물형태_혼합식	난방방식_난방	난방방식_유류난방
582	1308	177	46.760	19.0907	24665000	135960	8276.520	1	0	0	0
601	384	42	39.850	13.3232	19819000	110110	1673.700	0	1	0	0
1026	60	31	59.501	13.0100	10405000	86700	1844.531	0	0	0	0
63	532	70	59.670	23.6057	41054000	509410	4176.900	0	0	1	0
548	714	224	51.490	21.1473	17972000	234890	11533.760	0	1	0	0
...	...	...	...	...	...	...	...	...	...	...	...
183	307	147	84.560	24.0516	0	0	12430.320	0	0	0	0
446	291	91	51.550	28.6965	50775000	422950	4691.050	1	0	0	0
539	302	108	59.570	18.1137	26082000	234340	6433.560	0	0	0	0
640	496	206	46.220	21.0120	18319000	102660	9521.320	1	0	0	0
608	449	24	39.750	15.1769	19819000	109230	954.000	0	1	0	0

925 rows × 11 columns

(3) test

- **세부요구사항**
  - new_data로 부터, 전처리 및 예측 결과를 출력해 봅시다.

In [111…
```
modle_mk(model_rf, x_train, y_train, new_data, new_y_val)
```

```
model_name: Random Forest
MAE: 147.34930316558442
============================================================
MSE: 69132.61173372745
============================================================
MSPE: 0.3543322721597507
============================================================
R2: 0.32621182571932417
============================================================
최적의 파라미터: {'max_depth': 20, 'n_estimators': 100}
최고 성능: 0.9038001156088213
```

Out[111]:
```
{'Random Forest': 0.9038001156088213,
 'DecisionTreeRegressor': 0.6909187547369653,
 'KNeighborsRegressor': 0.8050606115132026}
```

In [112…
```
model = RandomForestRegressor(max_depth=50, n_estimators=100)
model.fit(x_train, y_train)
y_pred = model.predict(new_data)
print('R2:', r2_score(new_y_val, y_pred))
```

```
R2: 0.3544866965107959
```

In [113…
```
modle_mk(model_lr, x_train, y_train, new_data, new_y_val)
```

```
model_name: LinearRegression
MAE: 173.27308363897225
============================================================
MSE: 78001.10304614305
============================================================
MSPE: 0.5279485491706327
============================================================
R2: 0.23977672048951093
============================================================
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Cell In[113], line 1
----> 1 modle_mk(model_lr, x_train, y_train, new_data, new_y_val)

Cell In[99], line 63, in modle_mk(modle_select, x_train, y_train, x_val, y_val)
     61 print('R2:', r2_score(y_val, p1)) #결정계수 1에 가까울 수록 좋음
     62 print('='*60)
---> 63 print('최적의 파라미터:', model.best_params_)
     64 print('최고 성능:', model.best_score_)
     66 result[name] = model.best_score_

AttributeError: 'LinearRegression' object has no attribute 'best_params_'
```

In [ ]:
```
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()
scaler.fit(x_train)
x_train_s = scaler.transform(x_train)
new_val_s = scaler.transform(new_data)
```

In [ ]:
```
modle_mk(model_knn, x_train_s, y_train, new_val_s, new_y_val)
```

In [ ]:
```

```