제주도 도로 교통량 예측 AI 경진대회

제주도 도로 교통량 예측 AI 알고리즘 개발

1조

20191534 박성일

20182490 김진용

20212603 이가빈

목차

1 모듈 설치, 임포트

2 데이터 가져오기 ⊶ https://data.kma.go.kr/data/rmt/rmtList.do?code=400&pgmNo=570

3 특성 추가하기

4 특성 추가 1, 2, 3

5 특성을 추가해주는 파이프라인

6 train_set, test_set 나누기

7 표준화, 범주형 변수 구분하기

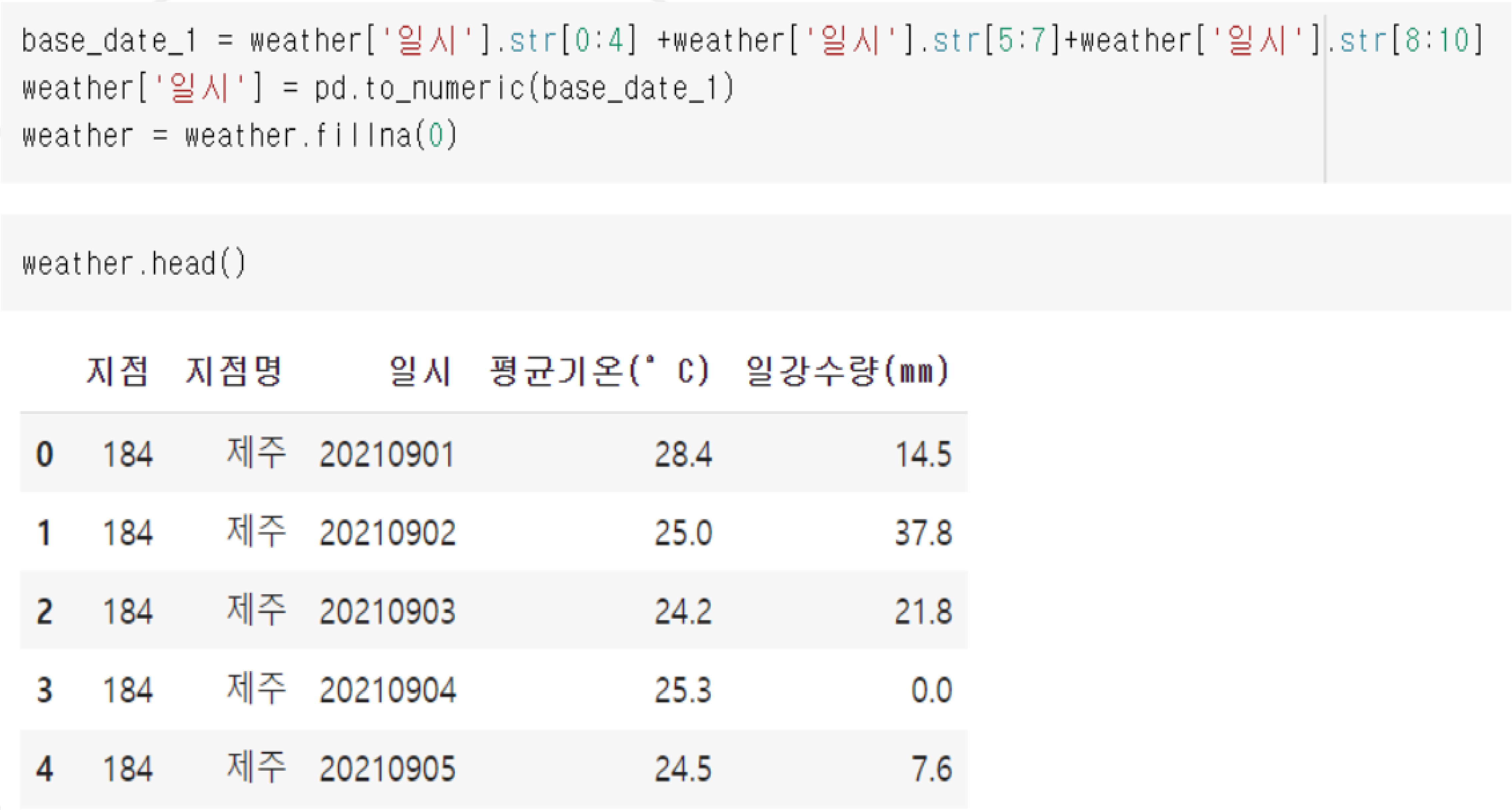
8 모델 성능 비교시 시간 단축을 위해 샘플 추출

9 모델 선정하기

10 모델 성능 시각화

11 예측, 제출

특성추가1



```
base_date_1 = weather['일시'].str[0:4] +weather['일시'].str[5:7]+weather['일시'].str[8:10]
weather['일시'] = pd.to_numeric(base_date_1)
weather = weather.fillna(0)
```

```
weather.head()
```

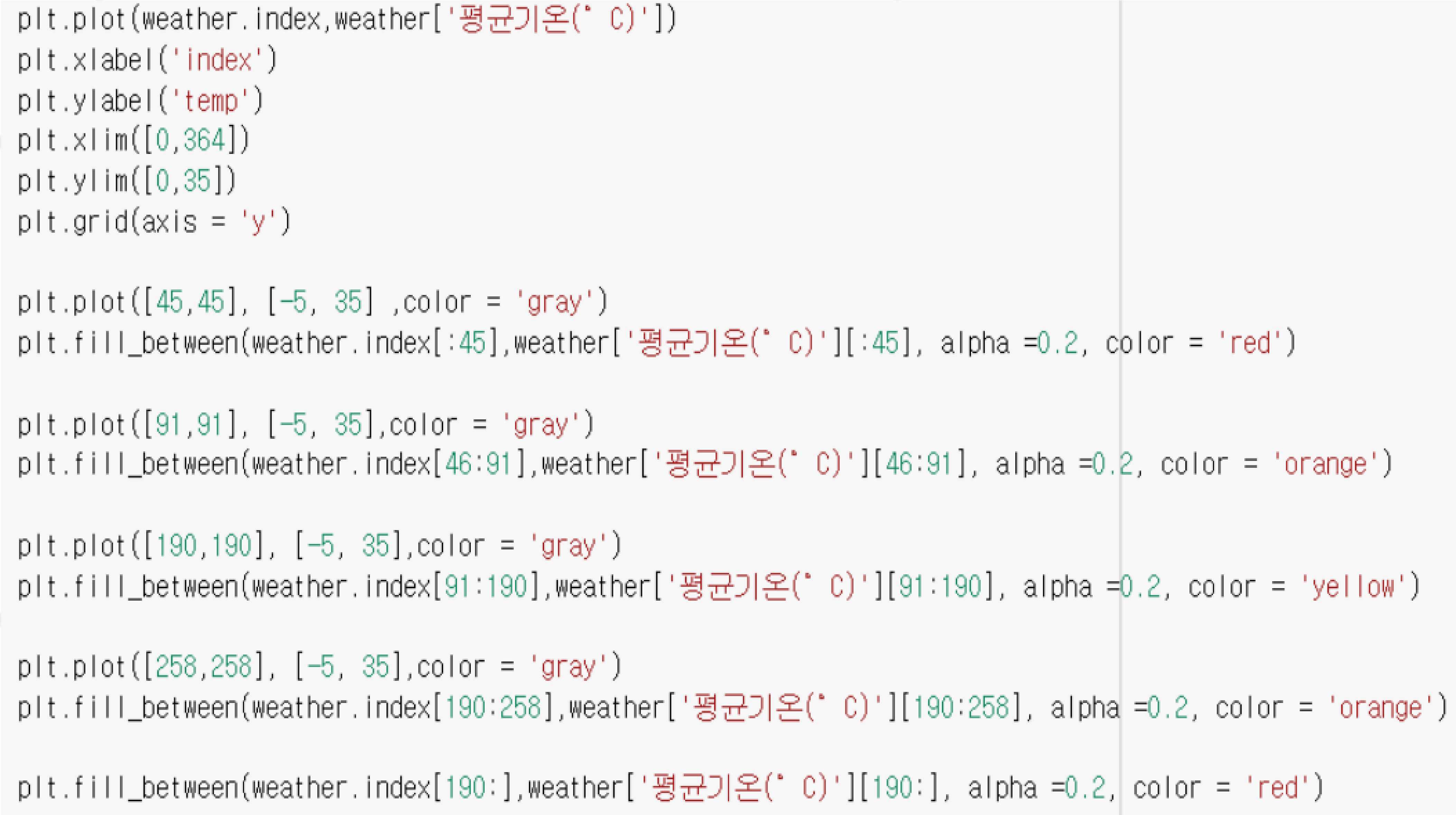
	지점	지점명	일시	평균기온(° C)	일강수량(mm)
0	184	제주	20210901	28.4	14.5
1	184	제주	20210902	25.0	37.8
2	184	제주	20210903	24.2	21.8
3	184	제주	20210904	25.3	0.0
4	184	제주	20210905	24.5	7.6

기온 데이터로 계절 구분하기

출저: 기상자료 개방 포털

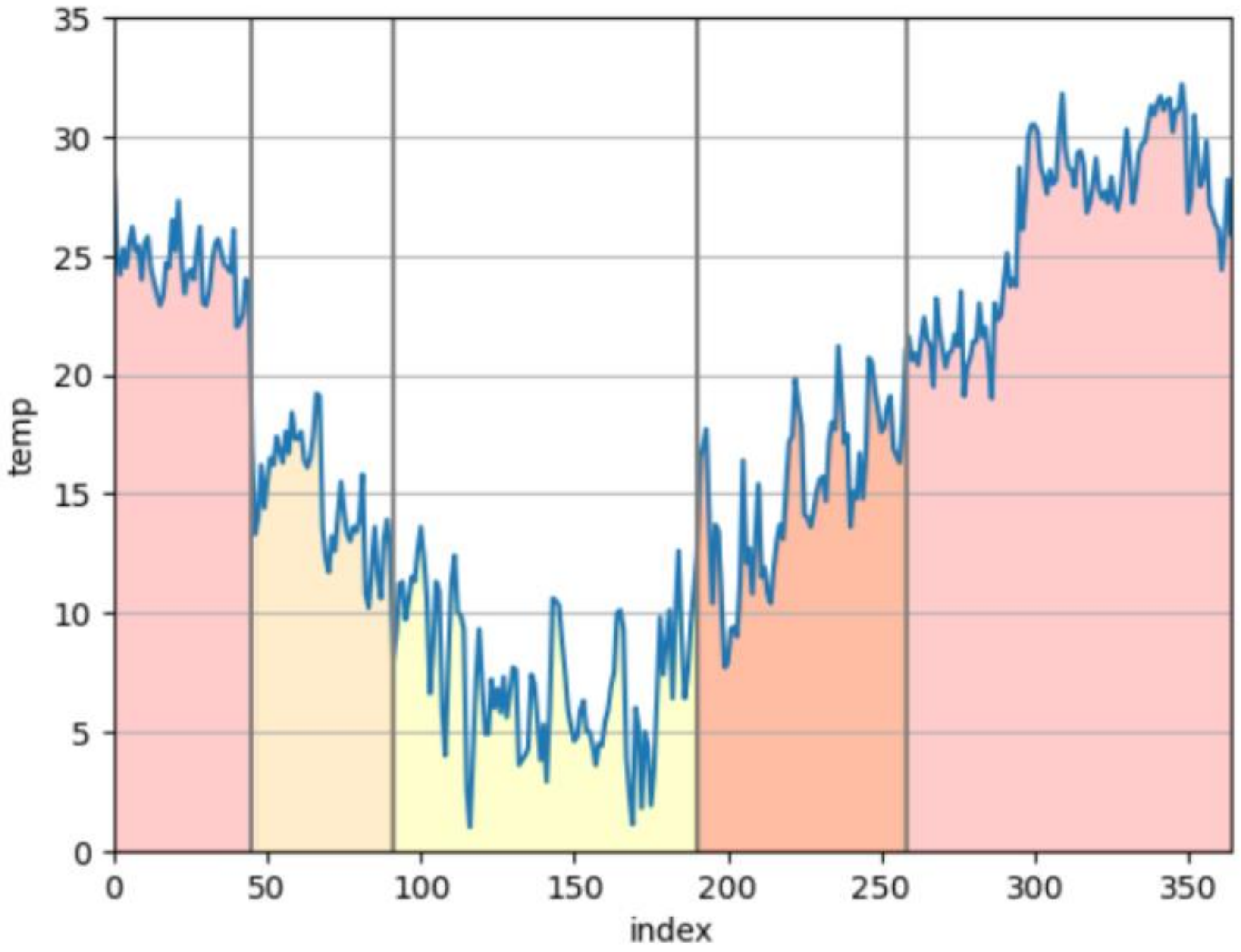
- https://data.kma.go.kr/cmmn/main.do

특성추가1

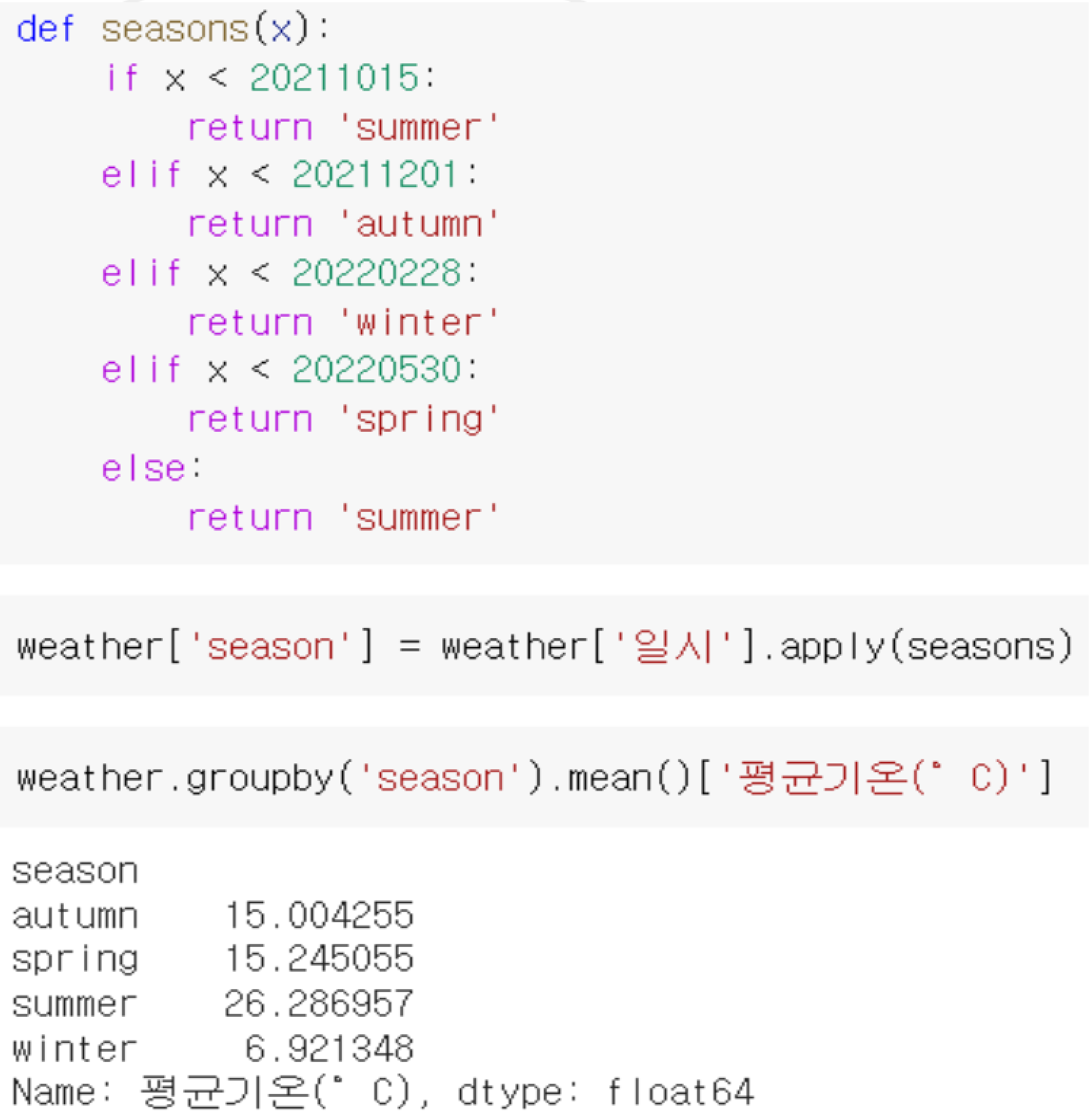


```
plt.plot(weather.index,weather['평균기온(° C)'])
plt.xlabel('index')
plt.ylabel('temp')
plt.xlim([0,364])
plt.ylim([0,35])
plt.grid(axis = 'y')

plt.plot([45,45], [-5, 35] ,color = 'gray')
plt.fill_between(weather.index[:45],weather['평균기온(° C)'][:45], alpha =0.2, color = 'red')

plt.plot([91,91], [-5, 35],color = 'gray')
plt.fill_between(weather.index[46:91],weather['평균기온(° C)'][46:91], alpha =0.2, color = 'orange')

plt.plot([190,190], [-5, 35],color = 'gray')
plt.fill_between(weather.index[91:190],weather['평균기온(° C)'][91:190], alpha =0.2, color = 'yellow')

plt.plot([258,258], [-5, 35],color = 'gray')
plt.fill_between(weather.index[190:258],weather['평균기온(° C)'][190:258], alpha =0.2, color = 'orange')

plt.fill_between(weather.index[190:],weather['평균기온(° C)'][190:], alpha =0.2, color = 'red')
```

```
<matplotlib.collections.PolyCollection at 0x112bdd4fe20>
```



특성추가1



```python
def seasons(x):
    if x < 20211015:
        return 'summer'
    elif x < 20211201:
        return 'autumn'
    elif x < 20220228:
        return 'winter'
    elif x < 20220530:
        return 'spring'
    else:
        return 'summer'
```

```python
weather['season'] = weather['일시'].apply(seasons)
```

```python
weather.groupby('season').mean()['평균기온(°C)']
```

```
season
autumn    15.004255
spring    15.245055
summer    26.286957
winter     6.921348
Name: 평균기온(°C), dtype: float64
```

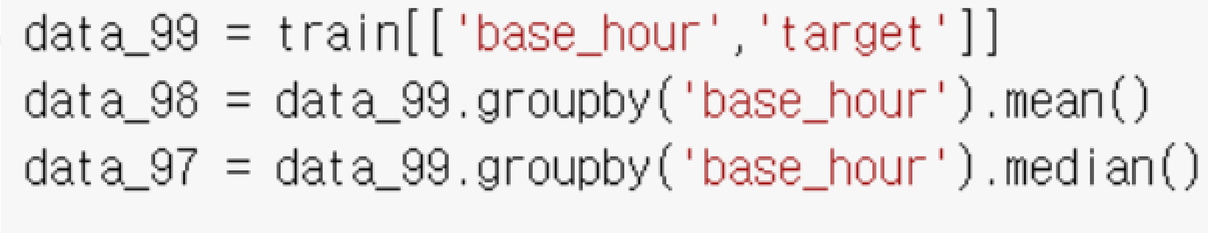
2021-09-01 ~ 2021-10-14 여름

2021-10-15 ~ 2021-11-30 가을

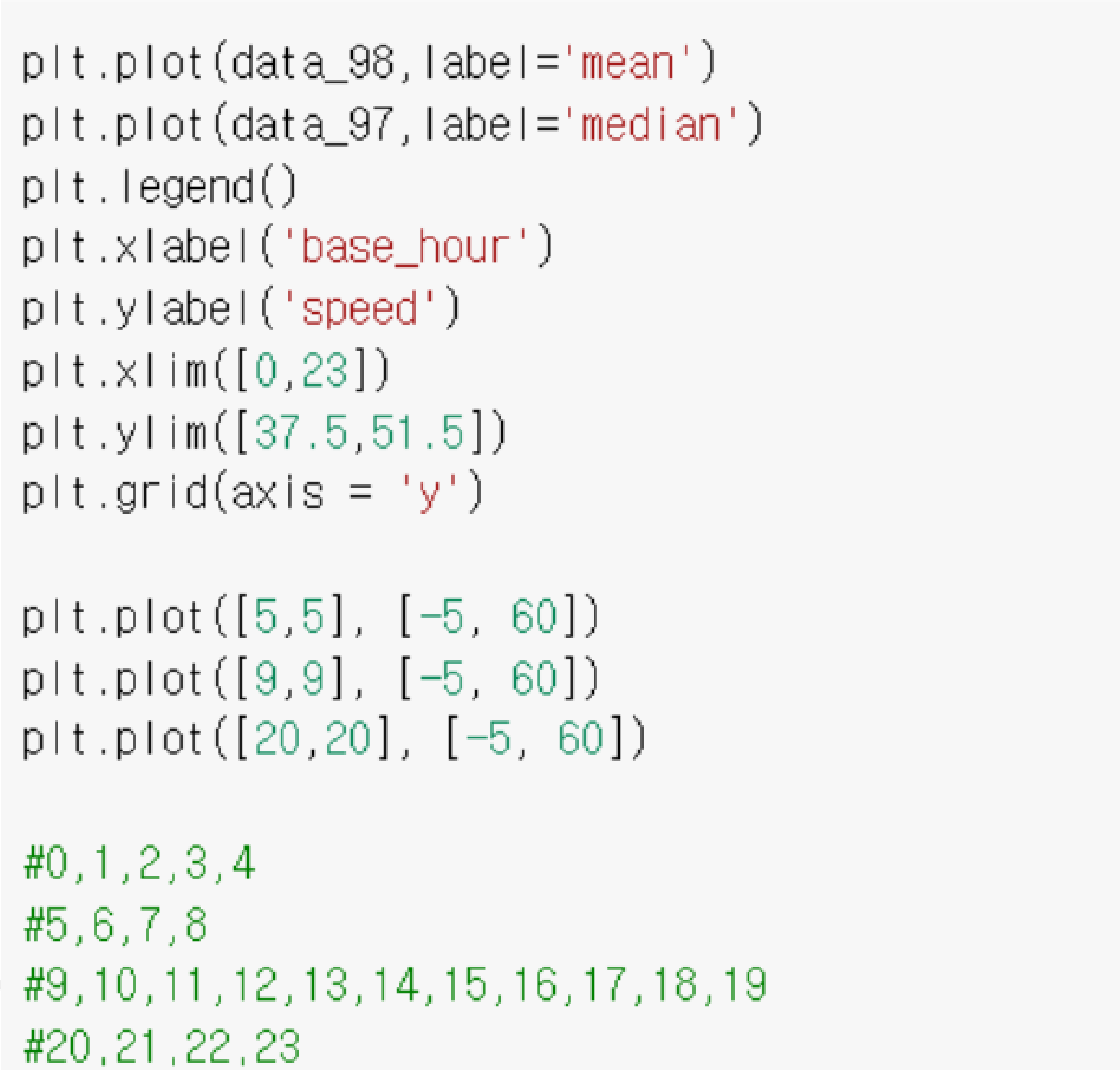
2021-12-01 ~ 2022-02-27 겨울

2022-02-28 ~ 2022-05-29 봄

2022-05-30 ~ 2022-08-31 여름



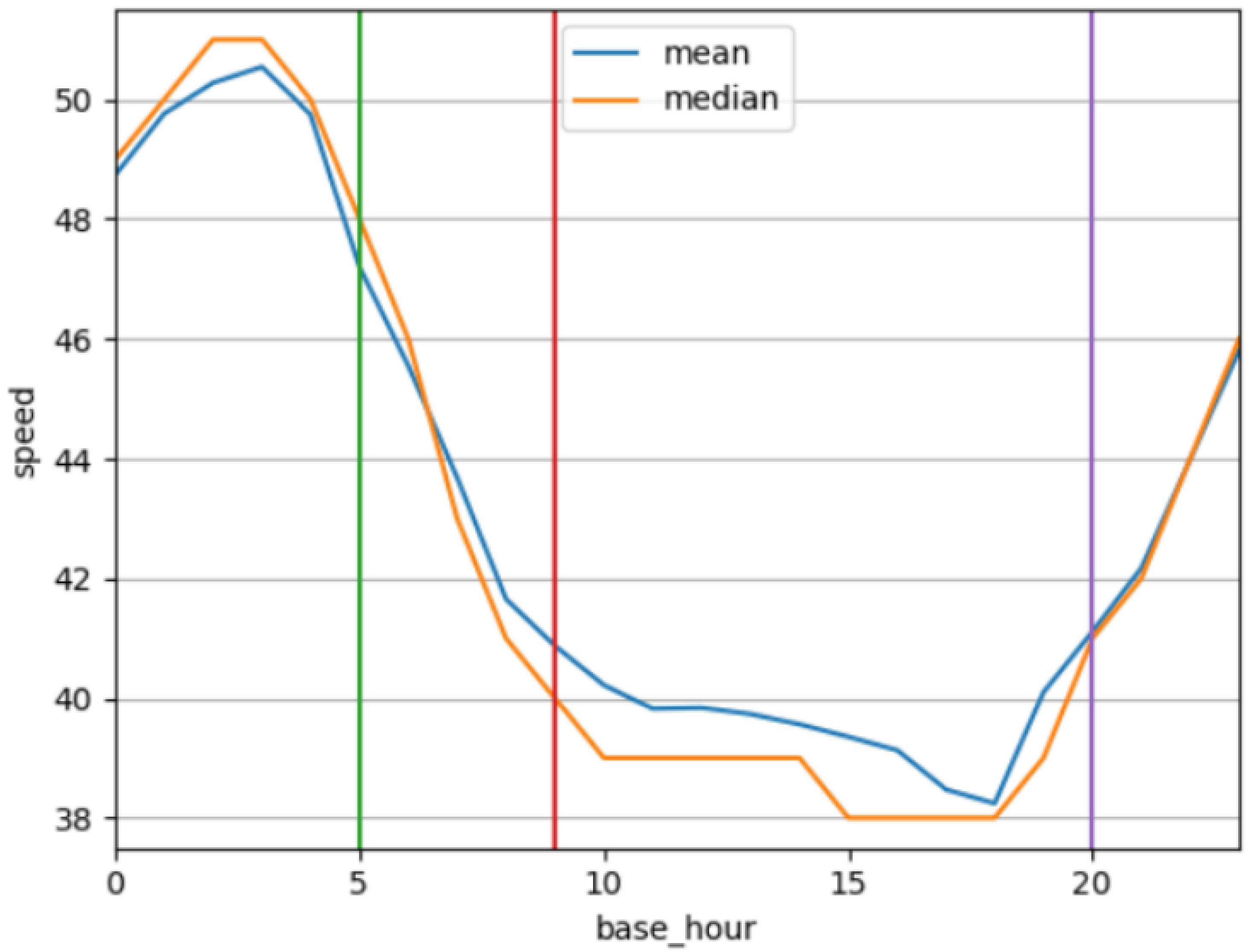
```
data_99 = train[['base_hour','target']]
data_98 = data_99.groupby('base_hour').mean()
data_97 = data_99.groupby('base_hour').median()
```



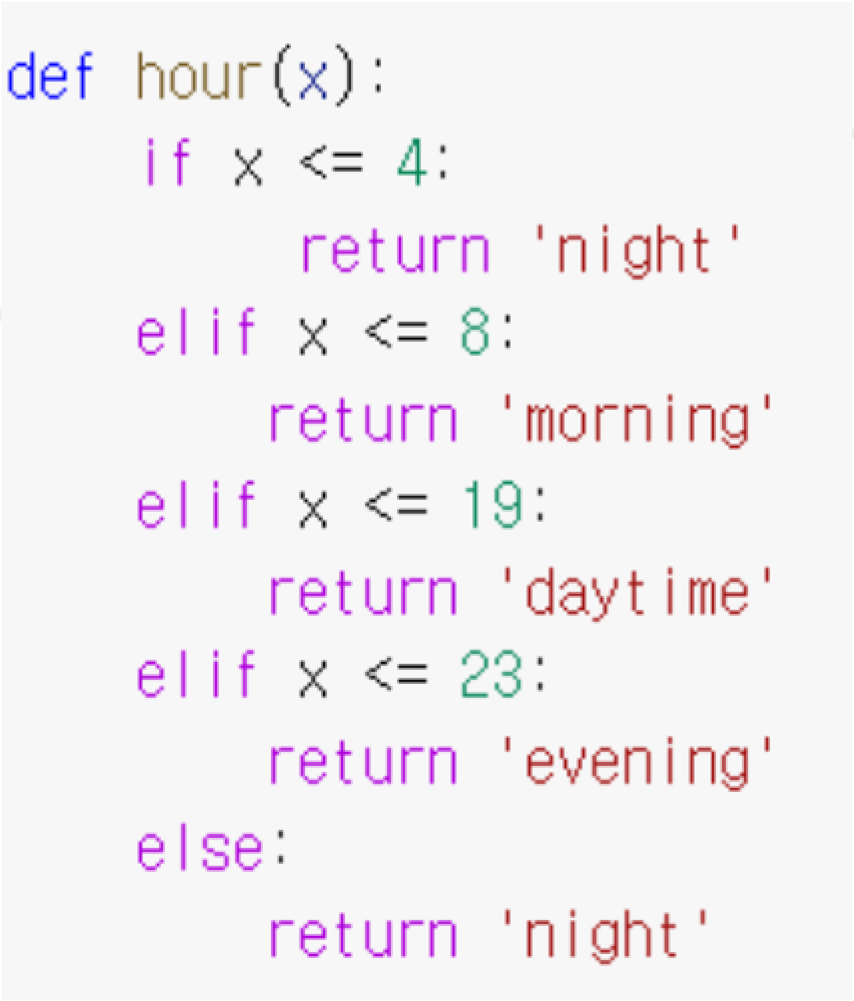
```
plt.plot(data_98,label='mean')
plt.plot(data_97,label='median')
plt.legend()
plt.xlabel('base_hour')
plt.ylabel('speed')
plt.xlim([0,23])
plt.ylim([37.5,51.5])
plt.grid(axis = 'y')

plt.plot([5,5], [-5, 60])
plt.plot([9,9], [-5, 60])
plt.plot([20,20], [-5, 60])

#0,1,2,3,4
#5,6,7,8
#9,10,11,12,13,14,15,16,17,18,19
#20,21,22,23
```

시간대별 속도를 개수가 적은 카테고리로 구분

```
[<matplotlib.lines.Line2D at 0x112bddb9190>]
```



특성추가2



```python
def hour(x):
    if x <= 4:
          return 'night'
    elif x <= 8:
        return 'morning'
    elif x <= 19:
        return 'daytime'
    elif x <= 23:
        return 'evening'
    else:
        return 'night'
```

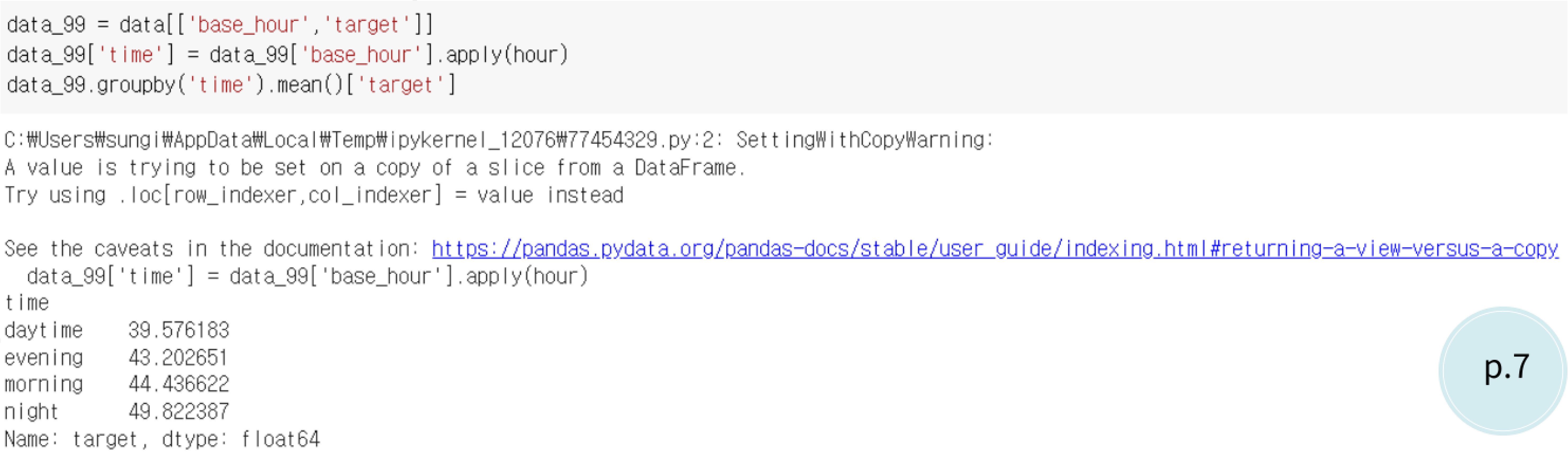
0~4 야간

5~8 아침

9~19 낮

20~23 저녁

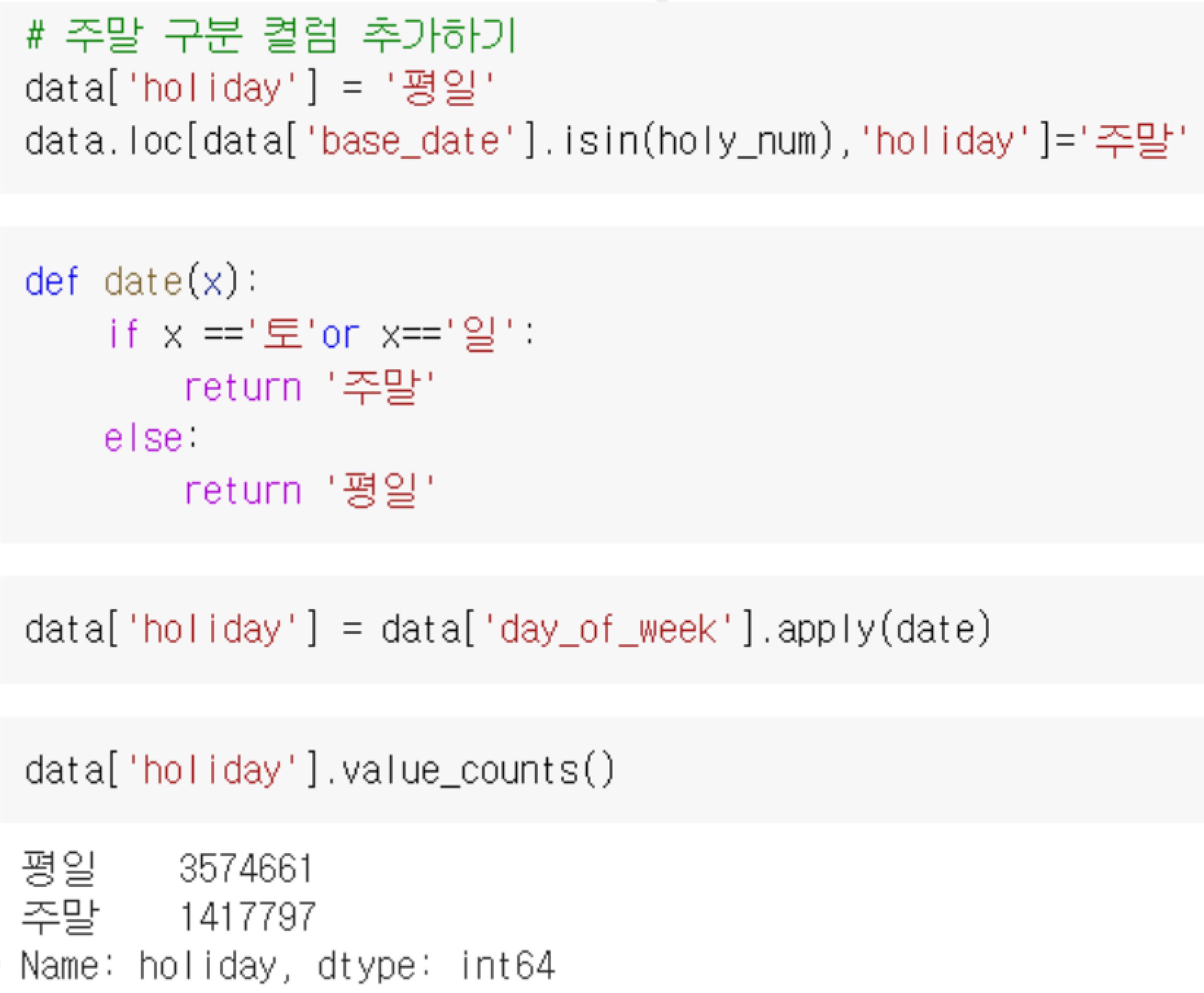
시간 카테고리별 속도 확인 밤의 평균속도가 높다



```python
data_99 = data[['base_hour','target']]
data_99['time'] = data_99['base_hour'].apply(hour)
data_99.groupby('time').mean()['target']
```

```
C:₩Users₩sungi₩AppData₩Local₩Temp₩ipykernel_12076₩77454329.py:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  data_99['time'] = data_99['base_hour'].apply(hour)
time
daytime    39.576183
evening    43.202651
morning    44.436622
night      49.822387
Name: target, dtype: float64
```

특성추가3



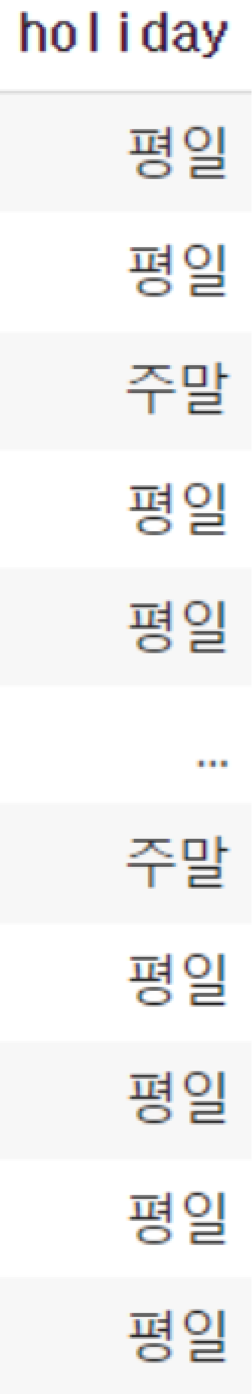
```
# 주말 구분 컬럼 추가하기
data['holiday'] = '평일'
data.loc[data['base_date'].isin(holy_num),'holiday']='주말'
```

```
def date(x):
    if x =='토'or x=='일':
        return '주말'
    else:
        return '평일'
```

```
data['holiday'] = data['day_of_week'].apply(date)
```

```
data['holiday'].value_counts()
```

```
평일    3574661
주말    1417797
Name: holiday, dtype: int64
```

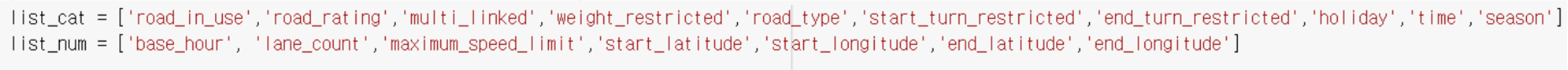


holiday
평일
평일
주말
평일
평일
...
주말
평일
평일
평일
평일

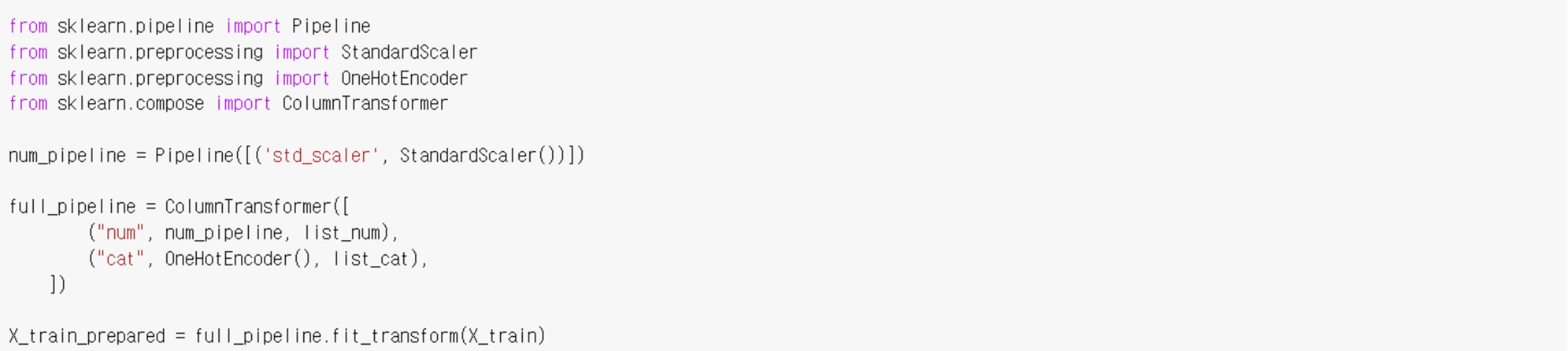
holiday

공휴일, 주말 = 주말
평일 = 평일

파이프라인

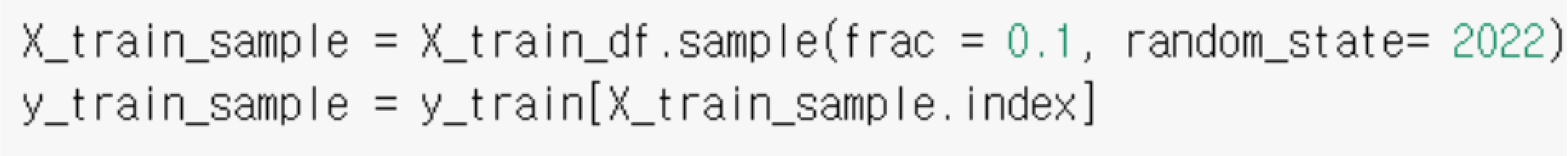


```python
list_cat = ['road_in_use','road_rating','multi_linked','weight_restricted','road_type','start_turn_restricted','end_turn_restricted','holiday','time','season']
list_num = ['base_hour', 'lane_count','maximum_speed_limit','start_latitude','start_longitude','end_latitude','end_longitude']
```



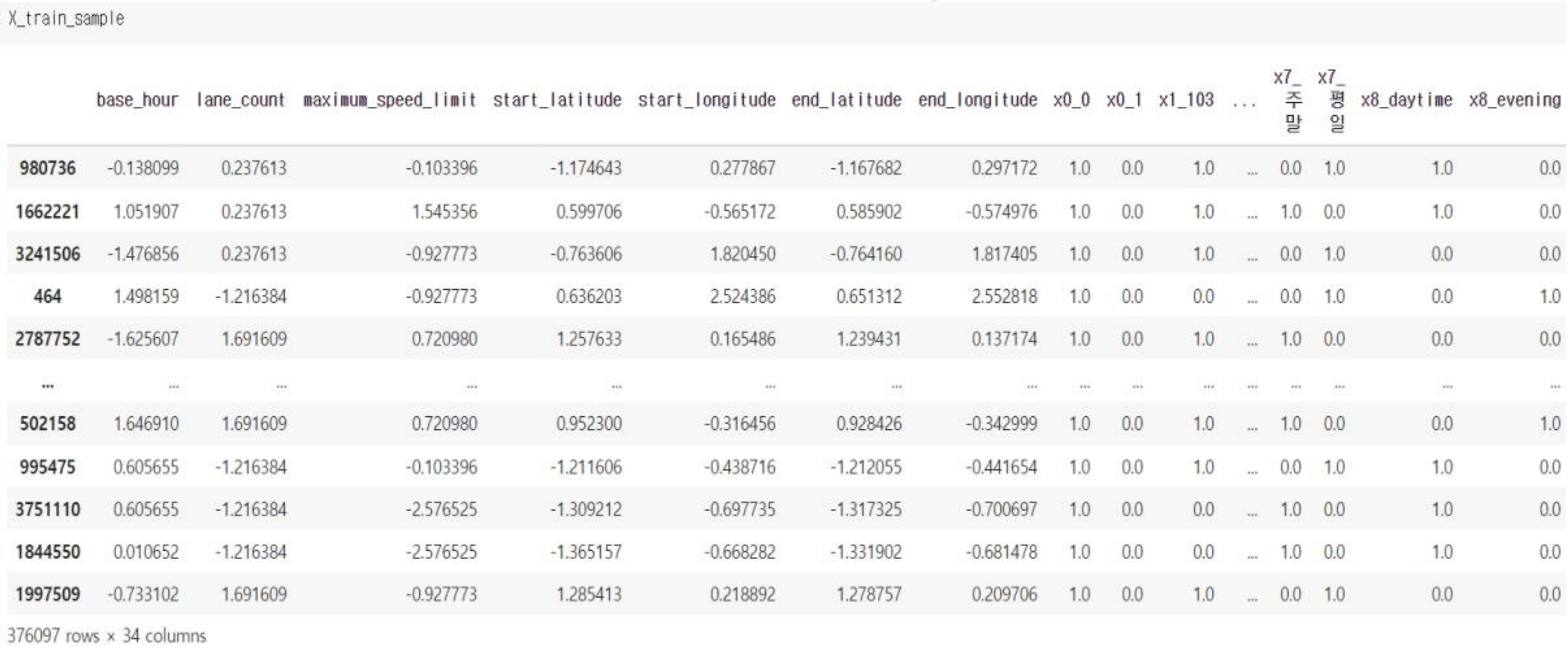
```python
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer

num_pipeline = Pipeline([('std_scaler', StandardScaler())])

full_pipeline = ColumnTransformer([
        ("num", num_pipeline, list_num),
        ("cat", OneHotEncoder(), list_cat),
    ])

X_train_prepared = full_pipeline.fit_transform(X_train)
```

모델 성능 비교시 시간 단축을 위해 샘플 추출



```
X_train_sample = X_train_df.sample(frac = 0.1, random_state= 2022)
y_train_sample = y_train[X_train_sample.index]
```

X_train_sample

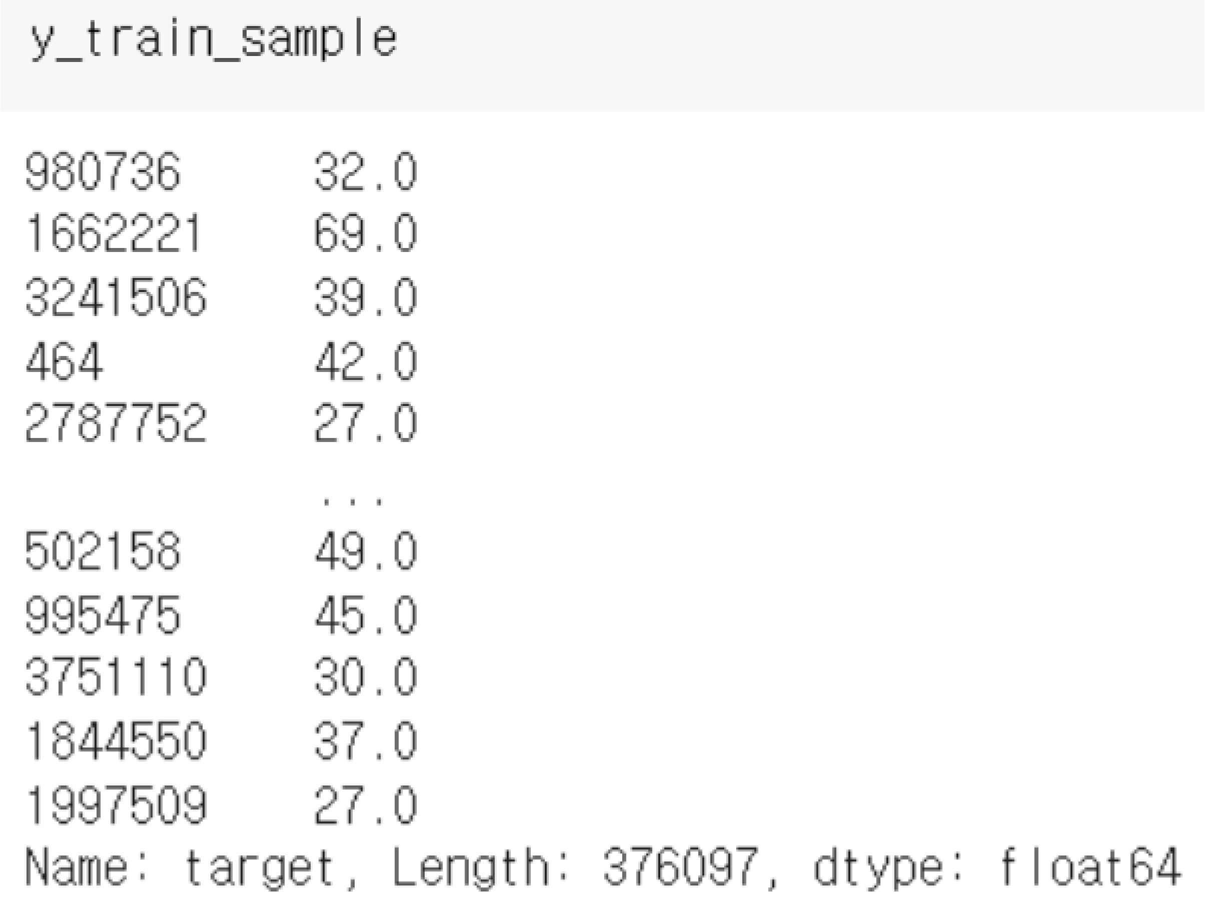


```
X_train_sample
```

	base_hour	lane_count	maximum_speed_limit	start_latitude	start_longitude	end_latitude	end_longitude	x0_0	x0_1	x1_103	...	x7_주말	x7_평일	x8_daytime	x8_evening
980736	-0.138099	0.237613	-0.103396	-1.174643	0.277867	-1.167682	0.297172	1.0	0.0	1.0	...	0.0	1.0	1.0	0.0
1662221	1.051907	0.237613	1.545356	0.599706	-0.565172	0.585902	-0.574976	1.0	0.0	1.0	...	1.0	0.0	1.0	0.0
3241506	-1.476856	0.237613	-0.927773	-0.763606	1.820450	-0.764160	1.817405	1.0	0.0	1.0	...	0.0	1.0	0.0	0.0
464	1.498159	-1.216384	-0.927773	0.636203	2.524386	0.651312	2.552818	1.0	0.0	0.0	...	0.0	1.0	0.0	1.0
2787752	-1.625607	1.691609	0.720980	1.257633	0.165486	1.239431	0.137174	1.0	0.0	1.0	...	1.0	0.0	0.0	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
502158	1.646910	1.691609	0.720980	0.952300	-0.316456	0.928426	-0.342999	1.0	0.0	1.0	...	1.0	0.0	0.0	1.0
995475	0.605655	-1.216384	-0.103396	-1.211606	-0.438716	-1.212055	-0.441654	1.0	0.0	1.0	...	0.0	1.0	1.0	0.0
3751110	0.605655	-1.216384	-2.576525	-1.309212	-0.697735	-1.317325	-0.700697	1.0	0.0	0.0	...	1.0	0.0	1.0	0.0
1844550	0.010652	-1.216384	-2.576525	-1.365157	-0.668282	-1.331902	-0.681478	1.0	0.0	0.0	...	1.0	0.0	1.0	0.0
1997509	-0.733102	1.691609	-0.927773	1.285413	0.218892	1.278757	0.209706	1.0	0.0	1.0	...	0.0	1.0	0.0	0.0

376097 rows × 34 columns

y_train_sample

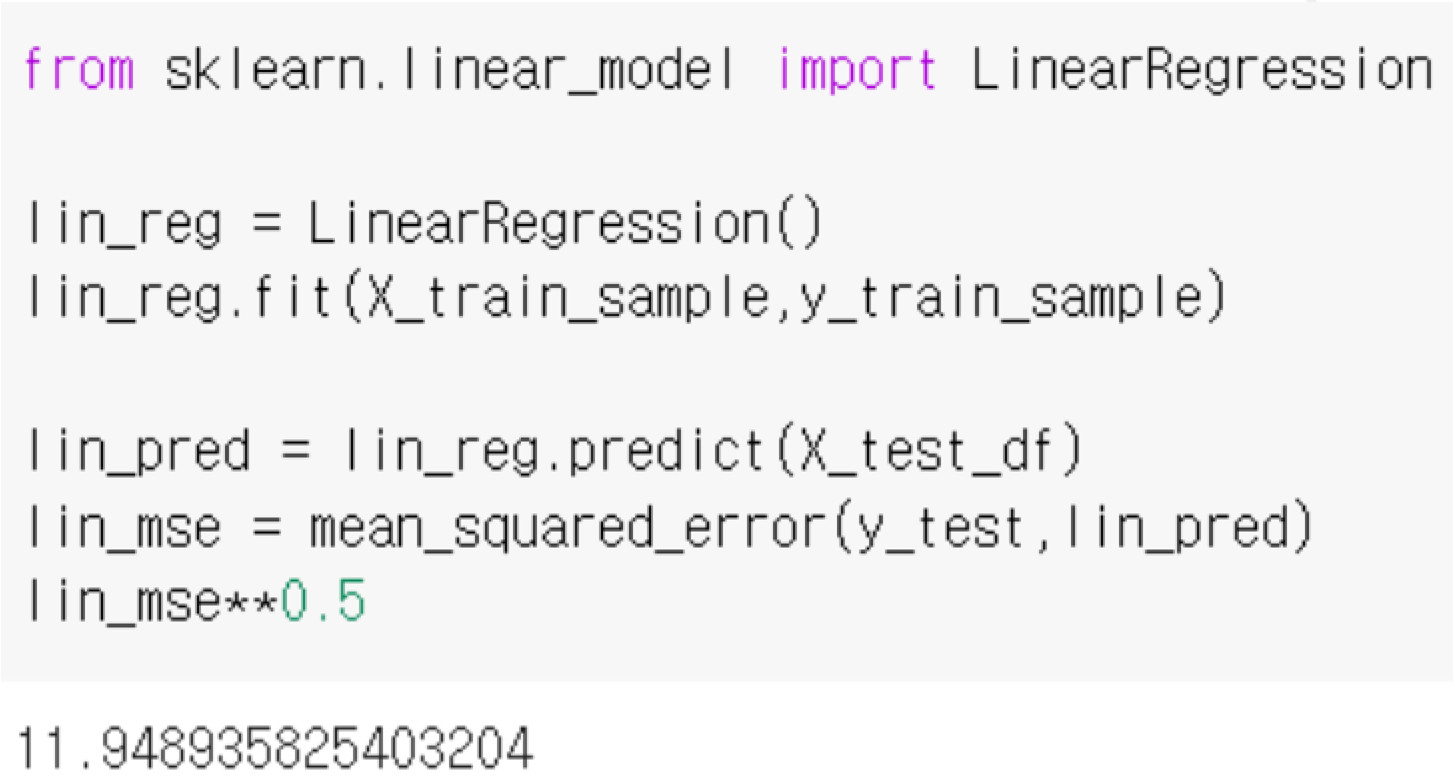


```
y_train_sample
```

```
980736     32.0
1662221    69.0
3241506    39.0
464        42.0
2787752    27.0
           ...
502158     49.0
995475     45.0
3751110    30.0
1844550    37.0
1997509    27.0
Name: target, Length: 376097, dtype: float64
```

X_test에 대한 예측값과 y_test의 RMSE 값을 비교

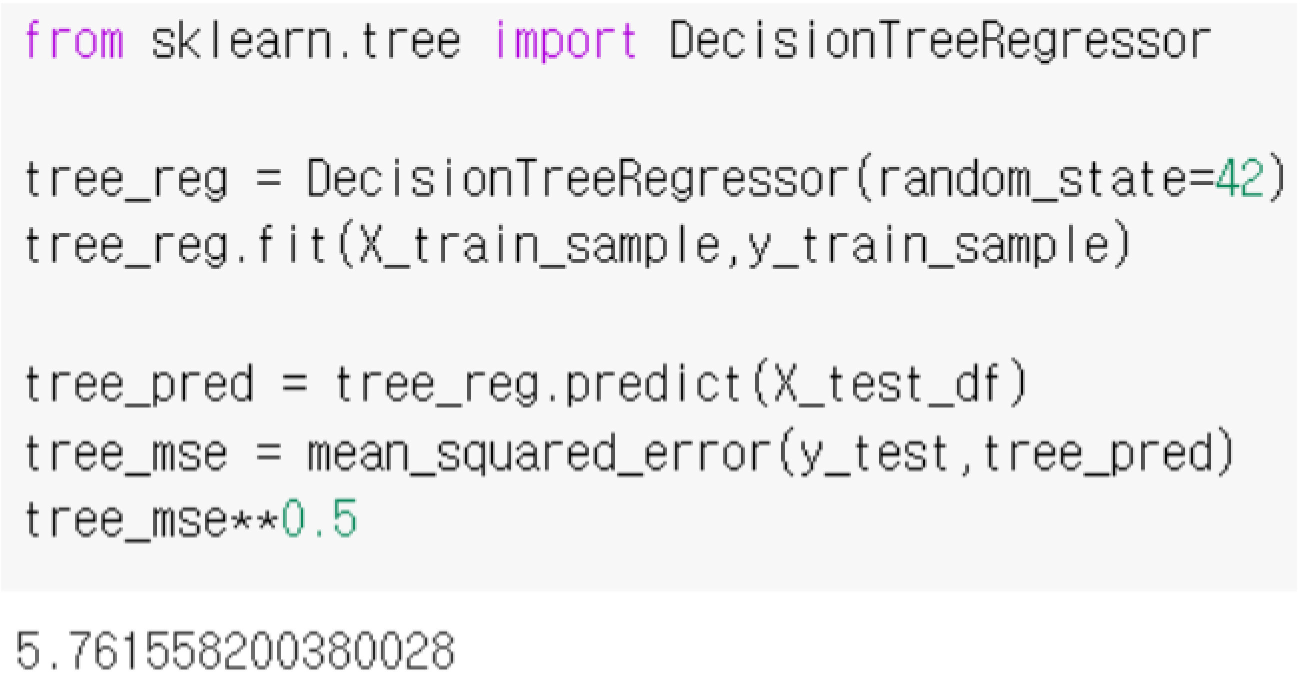
LinearRegression



```
from sklearn.linear_model import LinearRegression

lin_reg = LinearRegression()
lin_reg.fit(X_train_sample,y_train_sample)

lin_pred = lin_reg.predict(X_test_df)
lin_mse = mean_squared_error(y_test,lin_pred)
lin_mse**0.5
```

```
11.948935825403204
```

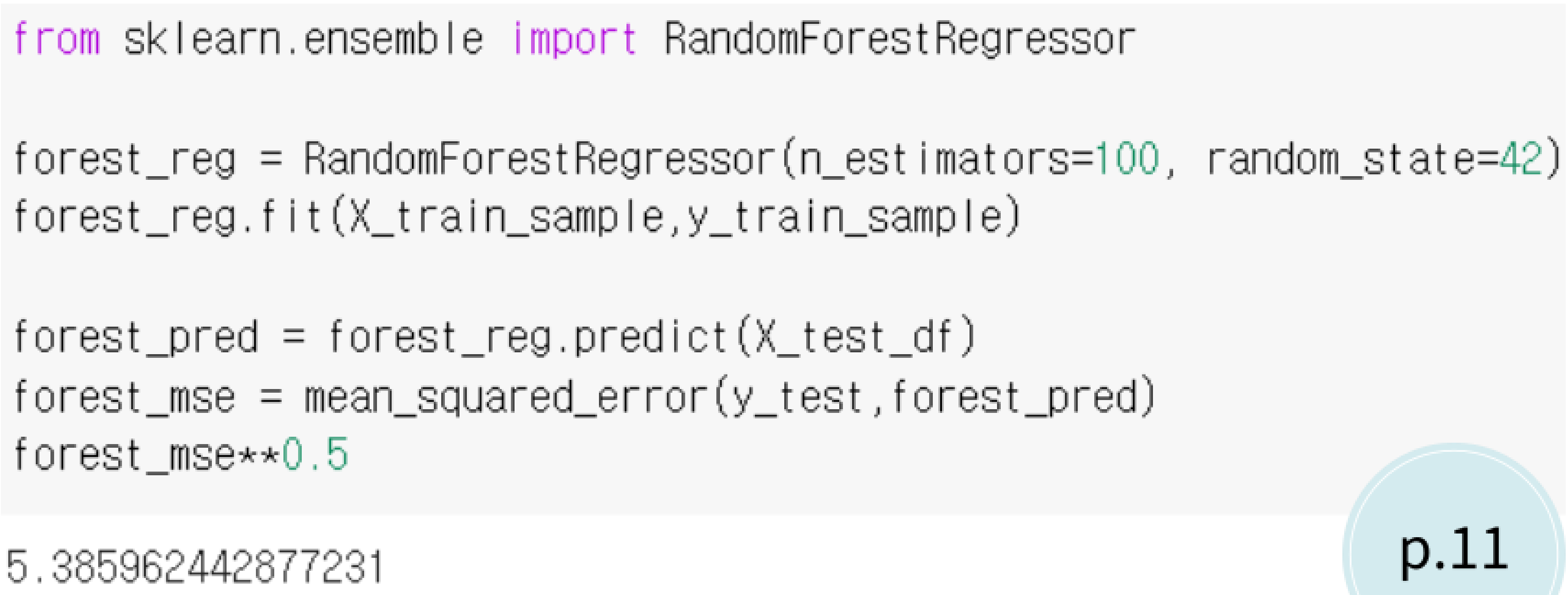
DecisionTreeRegressor



```
from sklearn.tree import DecisionTreeRegressor

tree_reg = DecisionTreeRegressor(random_state=42)
tree_reg.fit(X_train_sample,y_train_sample)

tree_pred = tree_reg.predict(X_test_df)
tree_mse = mean_squared_error(y_test,tree_pred)
tree_mse**0.5
```

```
5.761558200380028
```

RandomForest (Default)

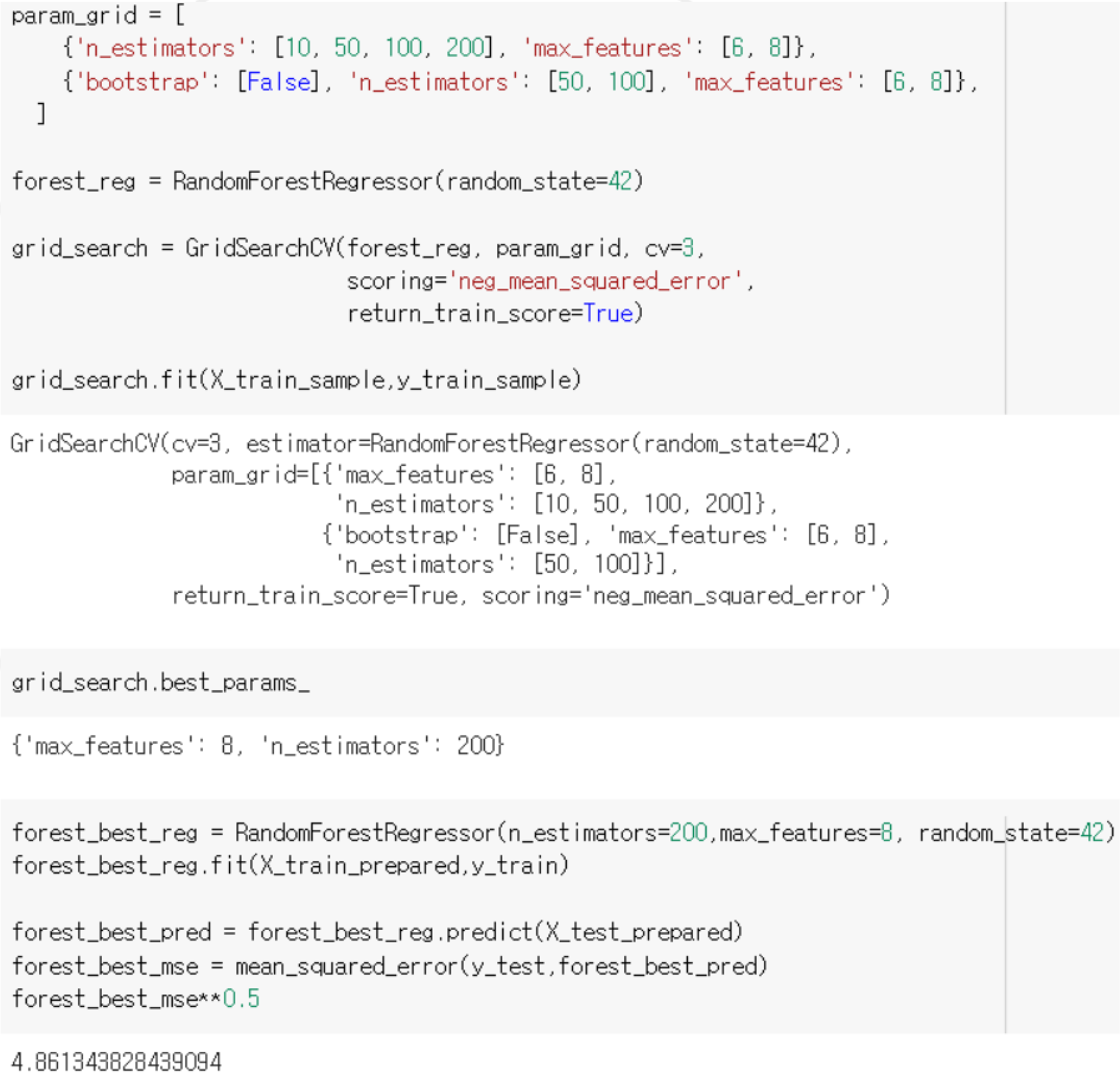


```
from sklearn.ensemble import RandomForestRegressor

forest_reg = RandomForestRegressor(n_estimators=100, random_state=42)
forest_reg.fit(X_train_sample,y_train_sample)

forest_pred = forest_reg.predict(X_test_df)
forest_mse = mean_squared_error(y_test,forest_pred)
forest_mse**0.5
```

```
5.385962442877231
```

RandomForest GridSearch

n_estimators : [10, 50, 100, 200], max_features : [6, 8]
bootstrap: [False], n_estimators: [50, 100], max_features: [6, 8]



```
param_grid = [
    {'n_estimators': [10, 50, 100, 200], 'max_features': [6, 8]},
    {'bootstrap': [False], 'n_estimators': [50, 100], 'max_features': [6, 8]},
  ]

forest_reg = RandomForestRegressor(random_state=42)

grid_search = GridSearchCV(forest_reg, param_grid, cv=3,
                           scoring='neg_mean_squared_error',
                           return_train_score=True)

grid_search.fit(X_train_sample,y_train_sample)
```

```
GridSearchCV(cv=3, estimator=RandomForestRegressor(random_state=42),
             param_grid=[{'max_features': [6, 8],
                          'n_estimators': [10, 50, 100, 200]},
                         {'bootstrap': [False], 'max_features': [6, 8],
                          'n_estimators': [50, 100]}],
             return_train_score=True, scoring='neg_mean_squared_error')
```

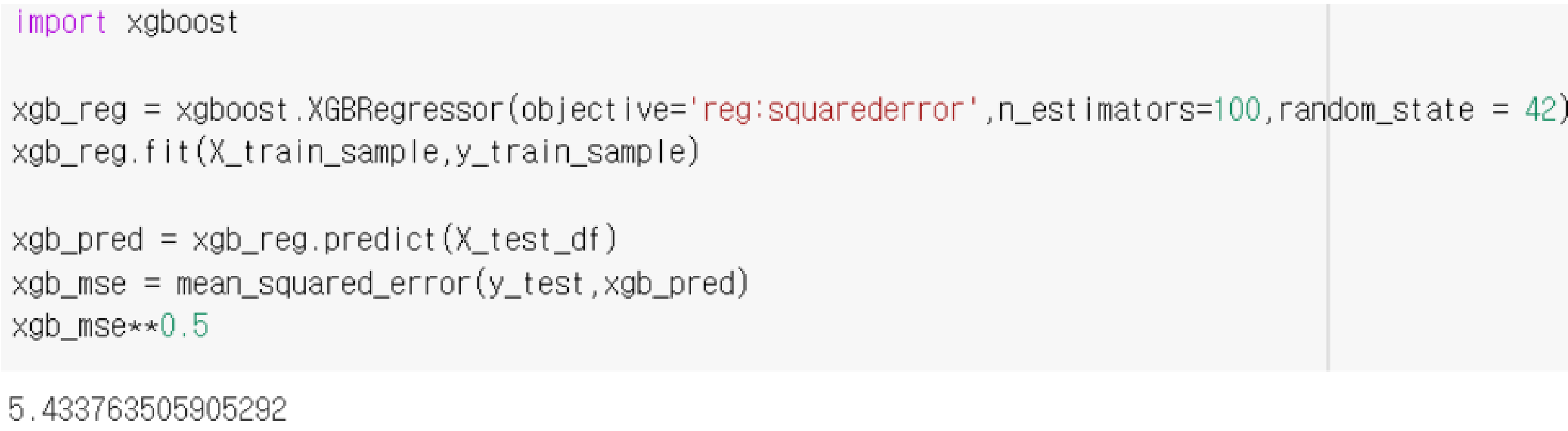
```
grid_search.best_params_
```

```
{'max_features': 8, 'n_estimators': 200}
```

```
forest_best_reg = RandomForestRegressor(n_estimators=200,max_features=8, random_state=42)
forest_best_reg.fit(X_train_prepared,y_train)

forest_best_pred = forest_best_reg.predict(X_test_prepared)
forest_best_mse = mean_squared_error(y_test,forest_best_pred)
forest_best_mse**0.5
```

```
4.8613438828439094
```

XGBRegressor (Default)

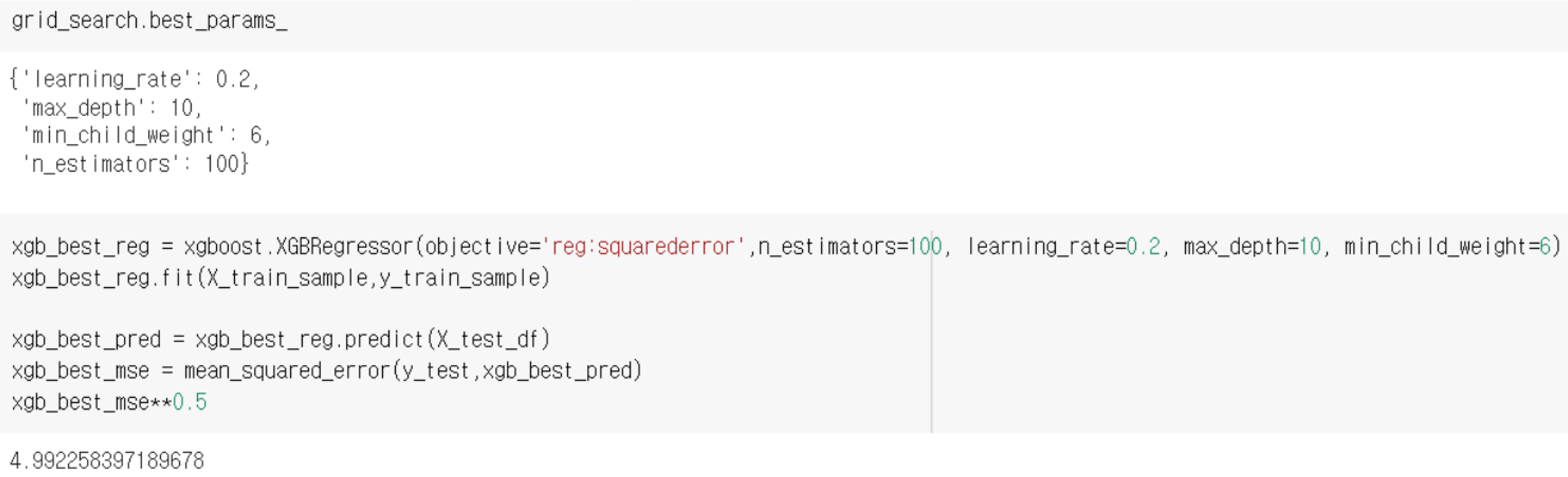


```
import xgboost

xgb_reg = xgboost.XGBRegressor(objective='reg:squarederror',n_estimators=100,random_state = 42)
xgb_reg.fit(X_train_sample,y_train_sample)

xgb_pred = xgb_reg.predict(X_test_df)
xgb_mse = mean_squared_error(y_test,xgb_pred)
xgb_mse**0.5
```

```
5.433763505905292
```

XGBRegressor GridSearch

max_depth: [10,30,50], min_child_weight : [3,6],
n_estimators : [100], learning_rate : [0.1, 0.16, 0.2]



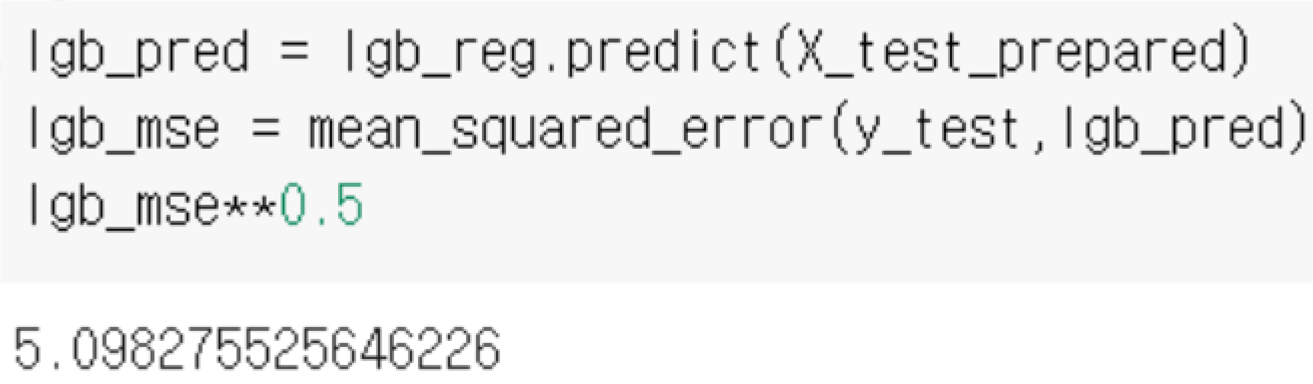
```
grid_search.best_params_
```

```
{'learning_rate': 0.2,
 'max_depth': 10,
 'min_child_weight': 6,
 'n_estimators': 100}
```

```
xgb_best_reg = xgboost.XGBRegressor(objective='reg:squarederror',n_estimators=100, learning_rate=0.2, max_depth=10, min_child_weight=6)
xgb_best_reg.fit(X_train_sample,y_train_sample)

xgb_best_pred = xgb_best_reg.predict(X_test_df)
xgb_best_mse = mean_squared_error(y_test,xgb_best_pred)
xgb_best_mse**0.5
```

```
4.992258397189678
```

lightgbm (Default)

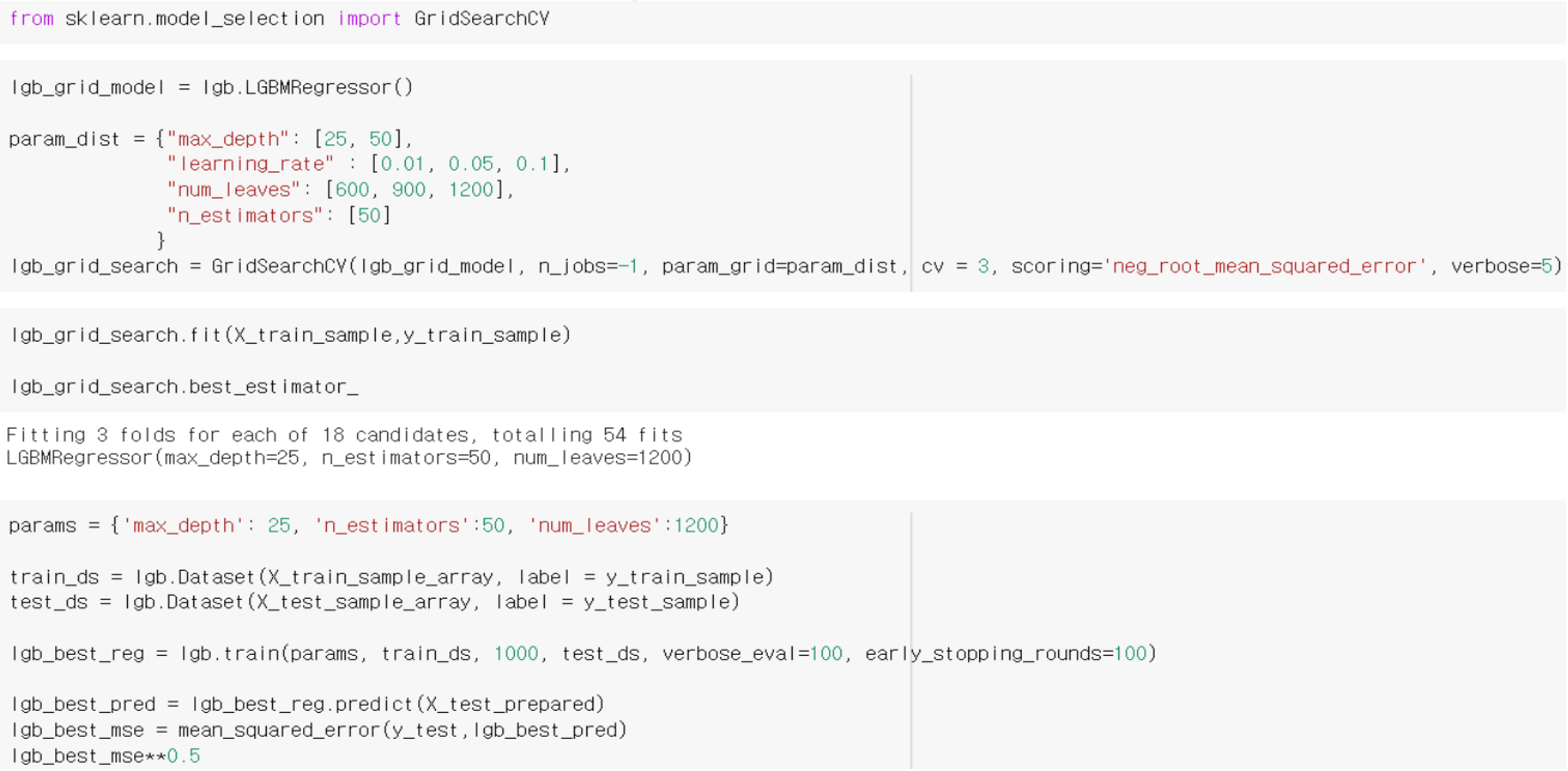


```
lgb_pred = lgb_reg.predict(X_test_prepared)
lgb_mse = mean_squared_error(y_test,lgb_pred)
lgb_mse**0.5
```

```
5.098275525646226
```

lightgbm GridSearch

max_depth: [25, 50, 75], learning_rate: [0.01, 0.05, 0.1], num_leaves: [300, 900, 1200], n_estimators: [200]



```
from sklearn.model_selection import GridSearchCV
```

```
lgb_grid_model = lgb.LGBMRegressor()

param_dist = {"max_depth": [25, 50],
              "learning_rate" : [0.01, 0.05, 0.1],
              "num_leaves": [600, 900, 1200],
              "n_estimators": [50]
             }
lgb_grid_search = GridSearchCV(lgb_grid_model, n_jobs=-1, param_grid=param_dist, cv = 3, scoring='neg_root_mean_squared_error', verbose=5)
```

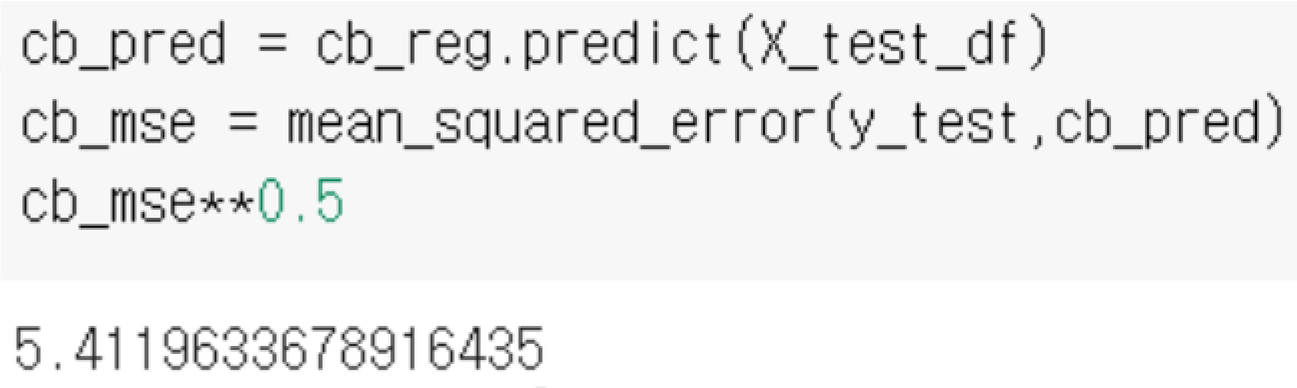
```
lgb_grid_search.fit(X_train_sample,y_train_sample)

lgb_grid_search.best_estimator_
```

```
Fitting 3 folds for each of 18 candidates, totalling 54 fits
LGBMRegressor(max_depth=25, n_estimators=50, num_leaves=1200)
```

```
params = {'max_depth': 25, 'n_estimators':50, 'num_leaves':1200}

train_ds = lgb.Dataset(X_train_sample_array, label = y_train_sample)
test_ds = lgb.Dataset(X_test_sample_array, label = y_test_sample)

lgb_best_reg = lgb.train(params, train_ds, 1000, test_ds, verbose_eval=100, early_stopping_rounds=100)

lgb_best_pred = lgb_best_reg.predict(X_test_prepared)
lgb_best_mse = mean_squared_error(y_test,lgb_best_pred)
lgb_best_mse**0.5
```

모델선정하기

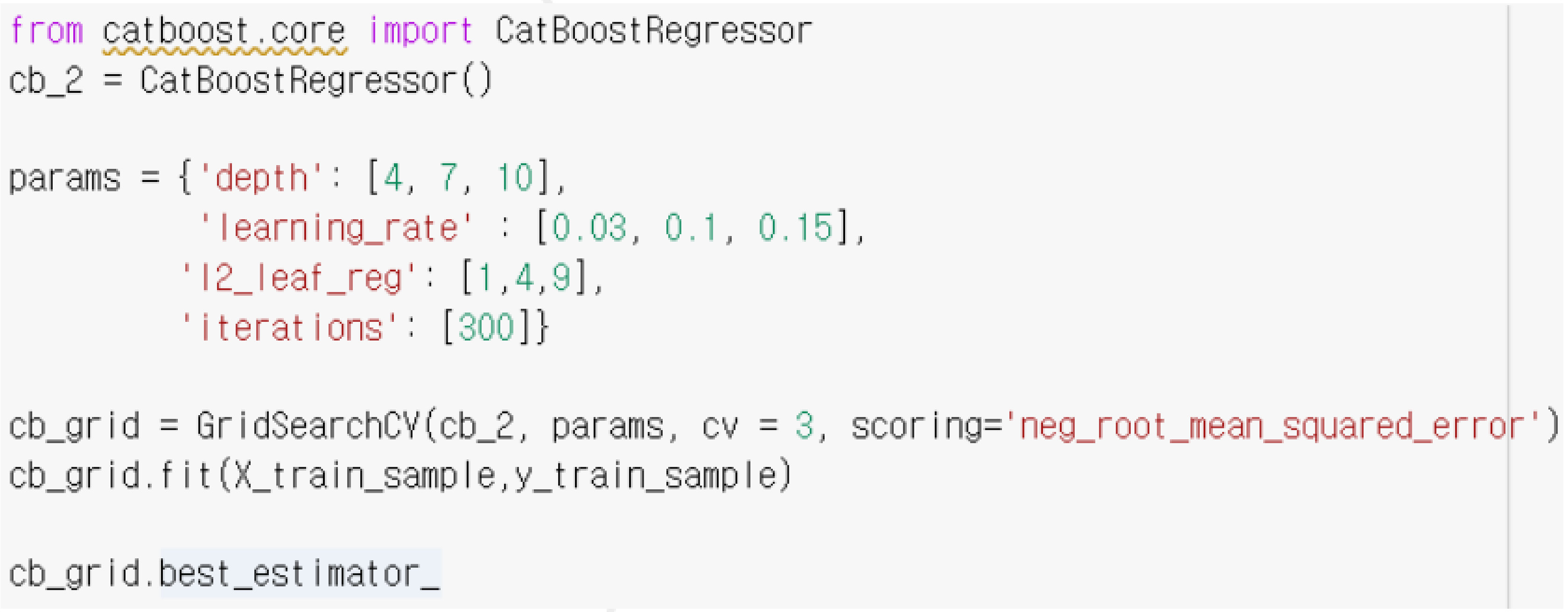
catboost (Default)



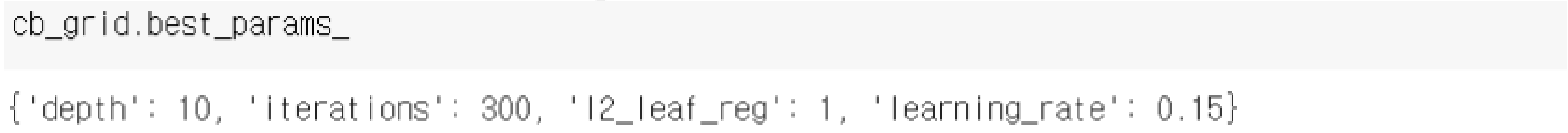
```
cb_pred = cb_reg.predict(X_test_df)
cb_mse = mean_squared_error(y_test,cb_pred)
cb_mse**0.5
```

```
5.4119633678916435
```

catboost GridSearch

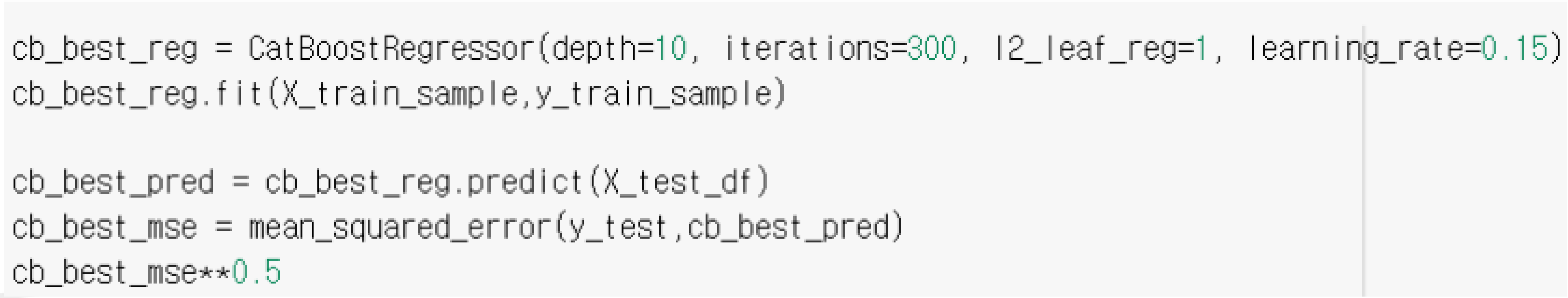


```
from catboost.core import CatBoostRegressor
cb_2 = CatBoostRegressor()

params = {'depth': [4, 7, 10],
          'learning_rate' : [0.03, 0.1, 0.15],
         'l2_leaf_reg': [1,4,9],
         'iterations': [300]}

cb_grid = GridSearchCV(cb_2, params, cv = 3, scoring='neg_root_mean_squared_error')
cb_grid.fit(X_train_sample,y_train_sample)

cb_grid.best_estimator_
```



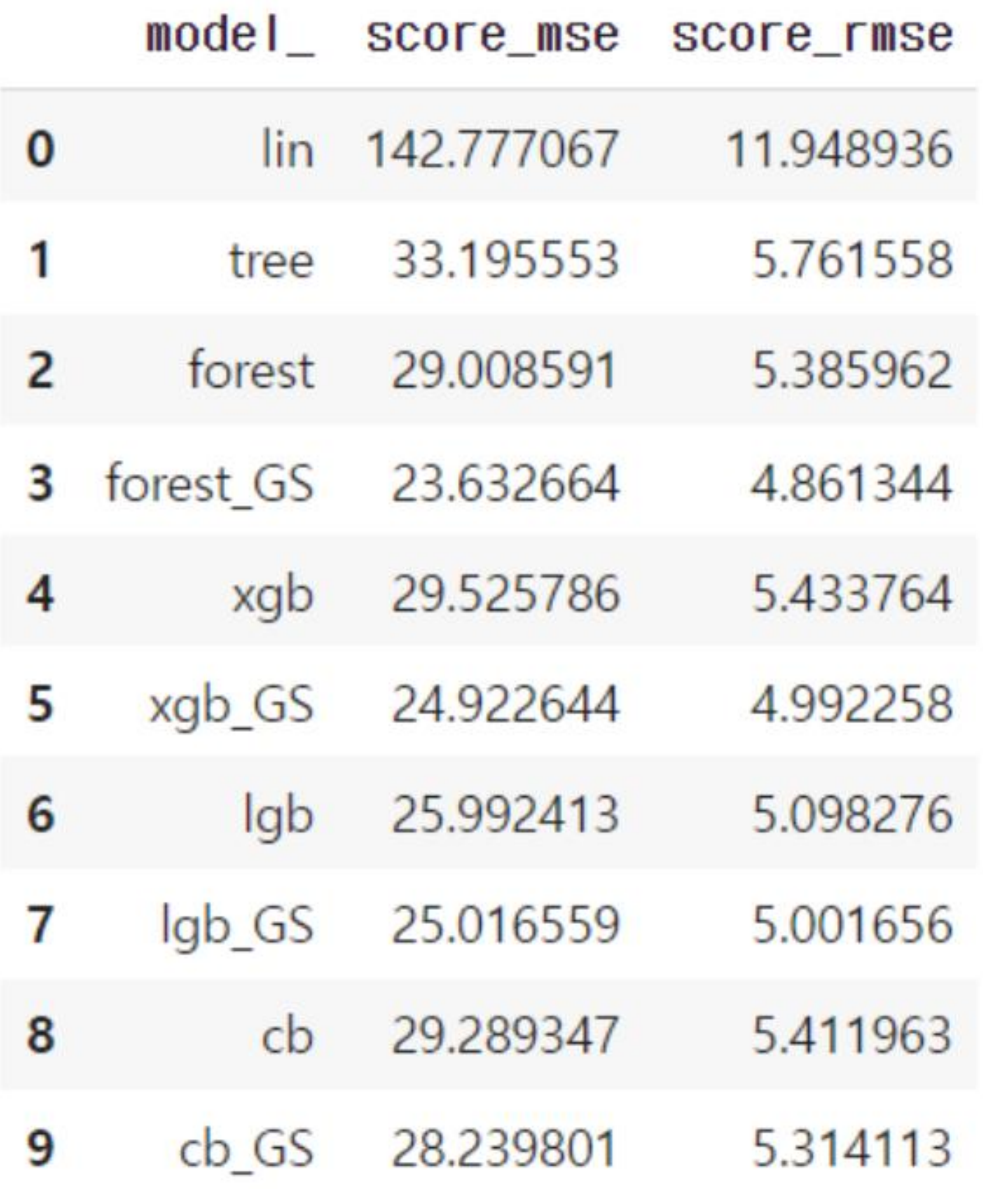
```
cb_grid.best_params_
```

```
{'depth': 10, 'iterations': 300, 'l2_leaf_reg': 1, 'learning_rate': 0.15}
```



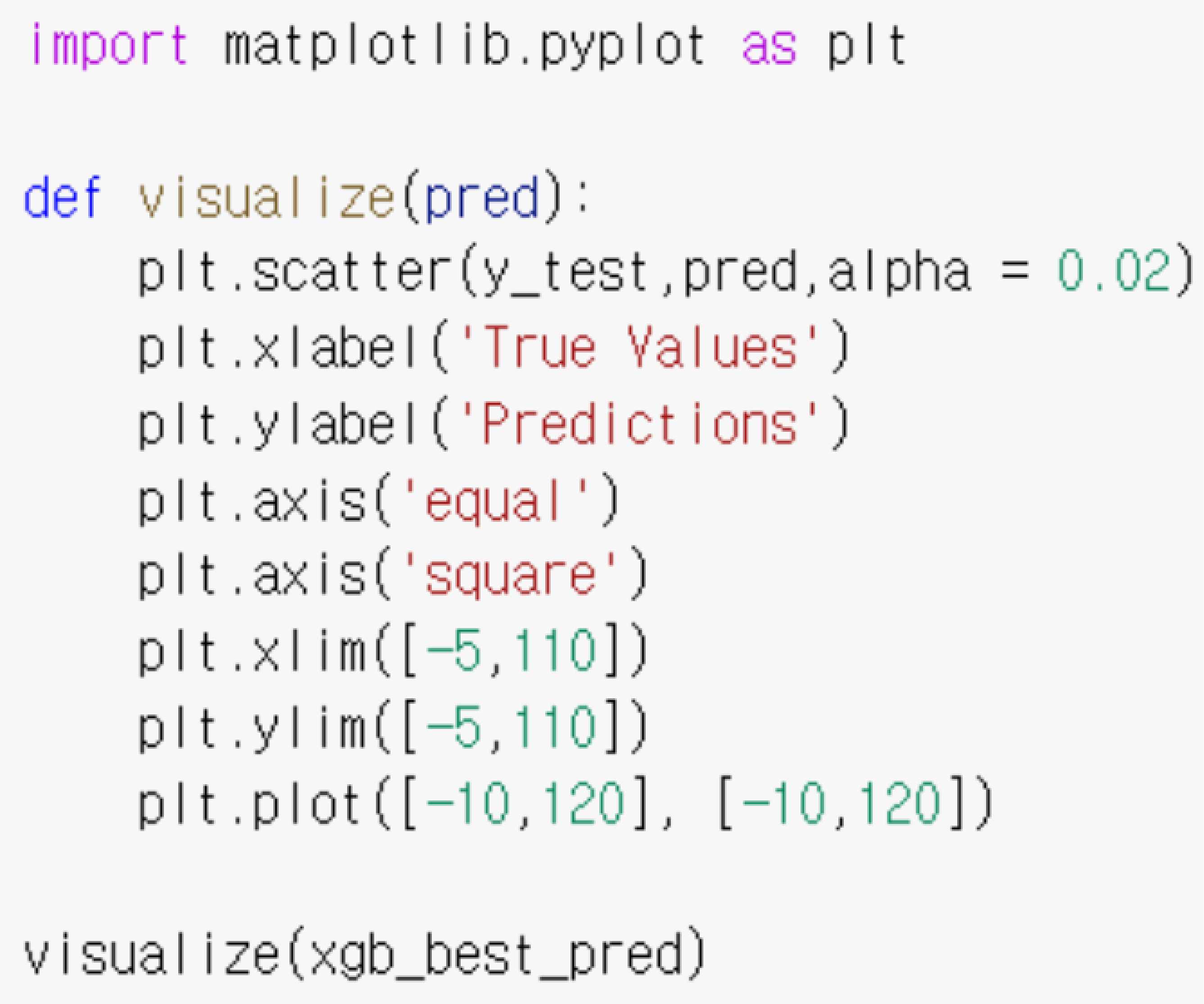
```
cb_best_reg = CatBoostRegressor(depth=10, iterations=300, l2_leaf_reg=1, learning_rate=0.15)
cb_best_reg.fit(X_train_sample,y_train_sample)

cb_best_pred = cb_best_reg.predict(X_test_df)
cb_best_mse = mean_squared_error(y_test,cb_best_pred)
cb_best_mse**0.5
```

5.314113340734843

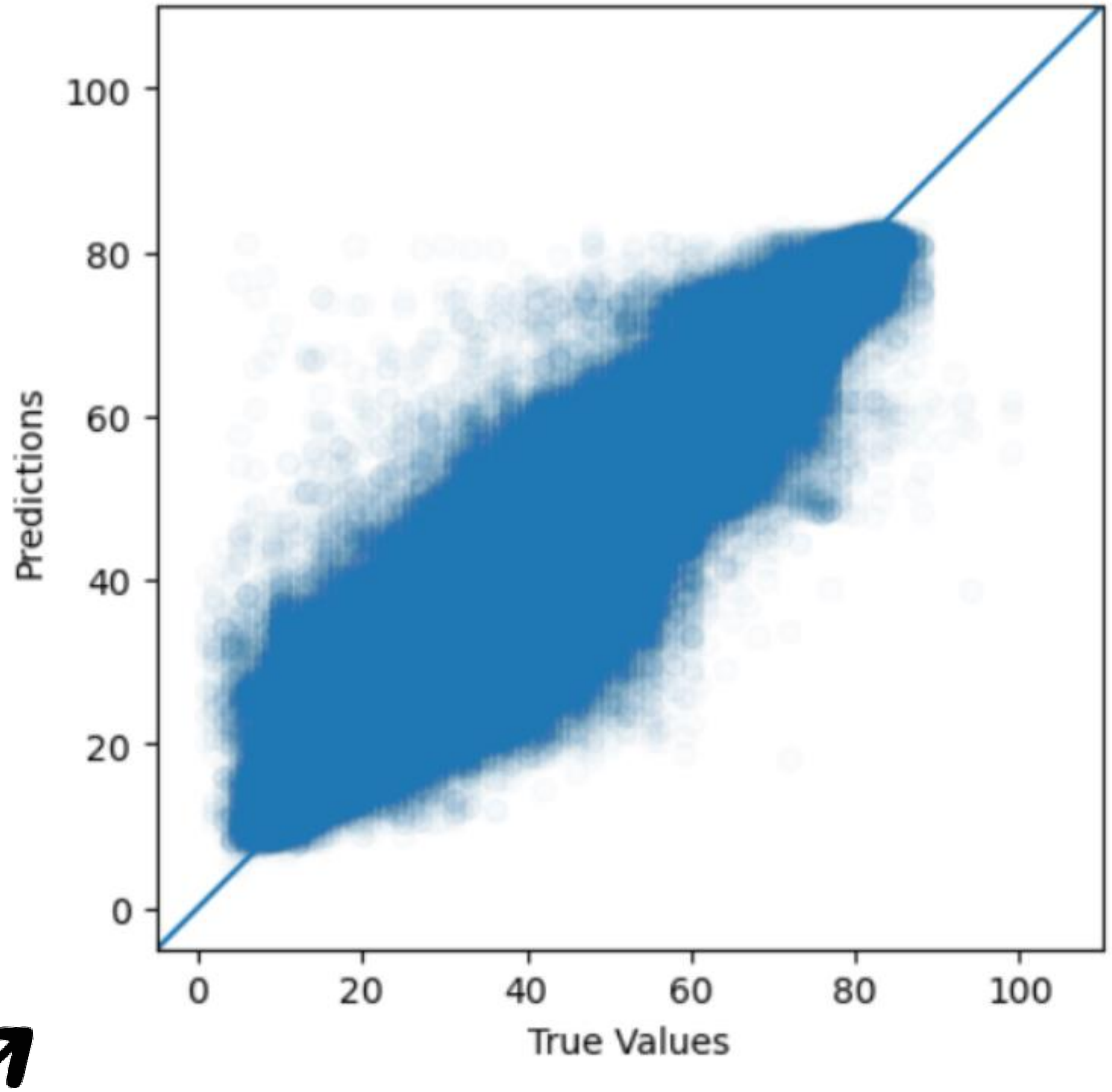


	model_	score_mse	score_rmse
0	lin	142.777067	11.948936
1	tree	33.195553	5.761558
2	forest	29.008591	5.385962
3	forest_GS	23.632664	4.861344
4	xgb	29.525786	5.433764
5	xgb_GS	24.922644	4.992258
6	lgb	25.992413	5.098276
7	lgb_GS	25.016559	5.001656
8	cb	29.289347	5.411963
9	cb_GS	28.239801	5.314113

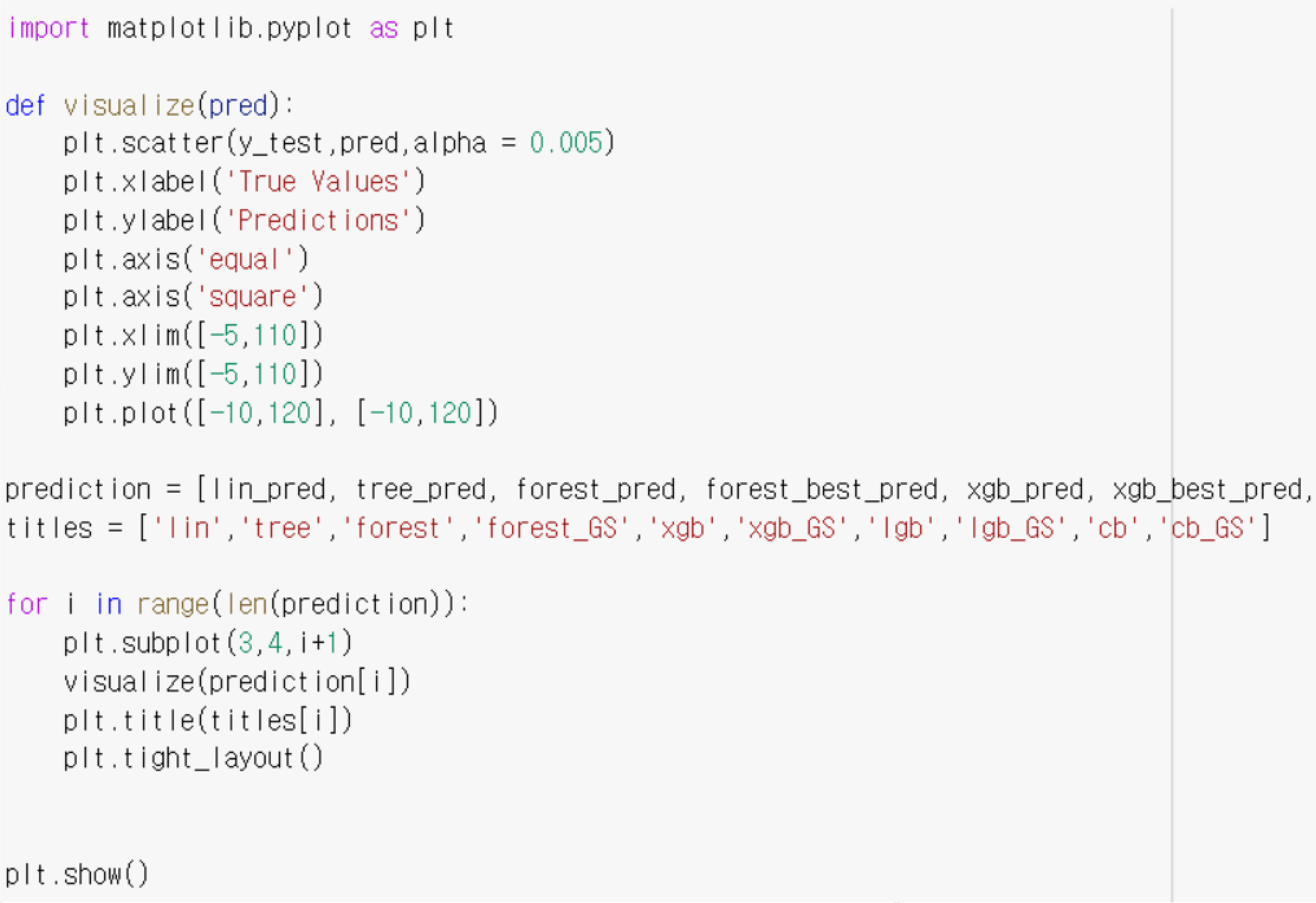
모델성능시각화



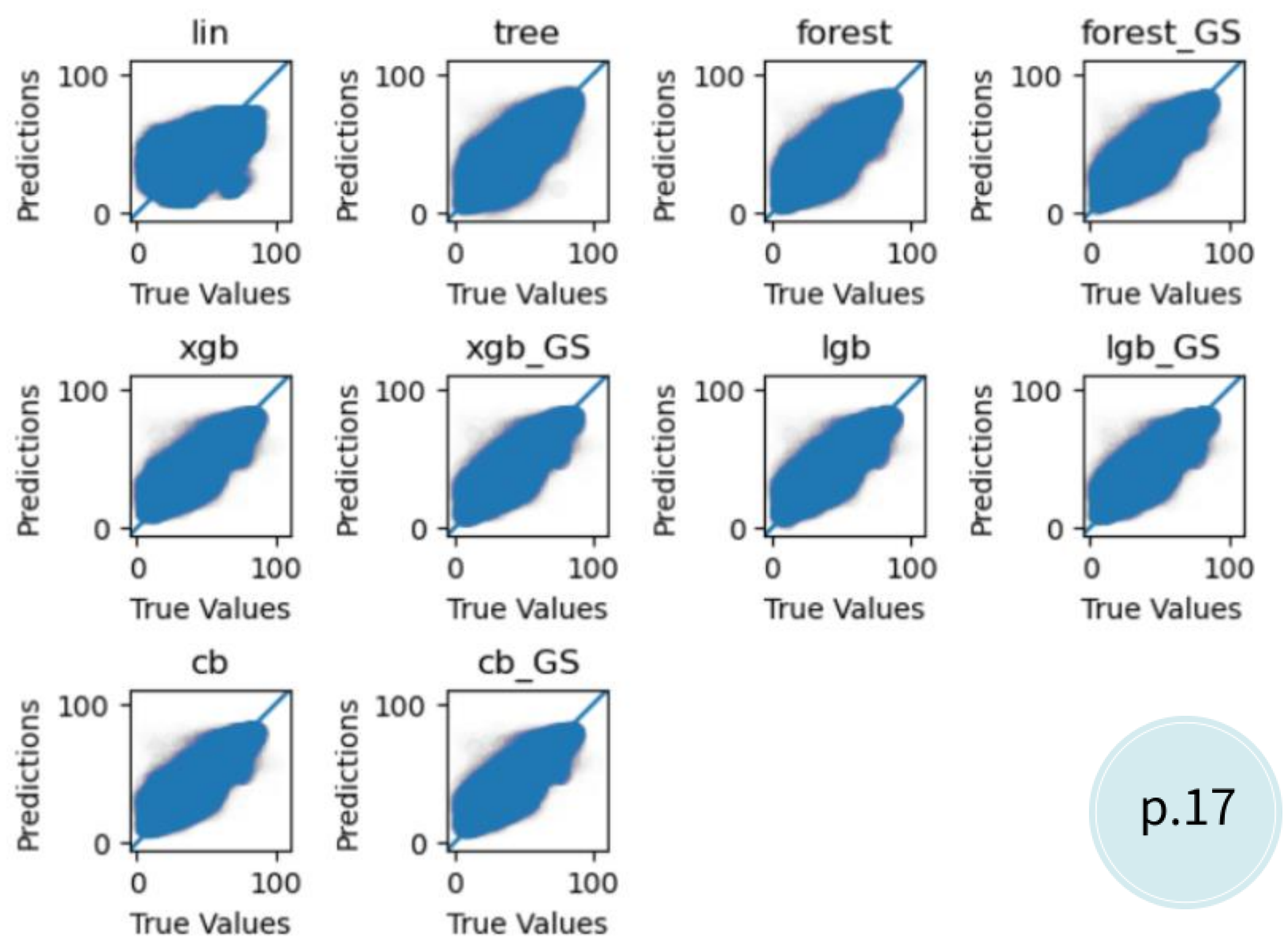
```
import matplotlib.pyplot as plt

def visualize(pred):
    plt.scatter(y_test,pred,alpha = 0.02)
    plt.xlabel('True Values')
    plt.ylabel('Predictions')
    plt.axis('equal')
    plt.axis('square')
    plt.xlim([-5,110])
    plt.ylim([-5,110])
    plt.plot([-10,120], [-10,120])

visualize(xgb_best_pred)
```



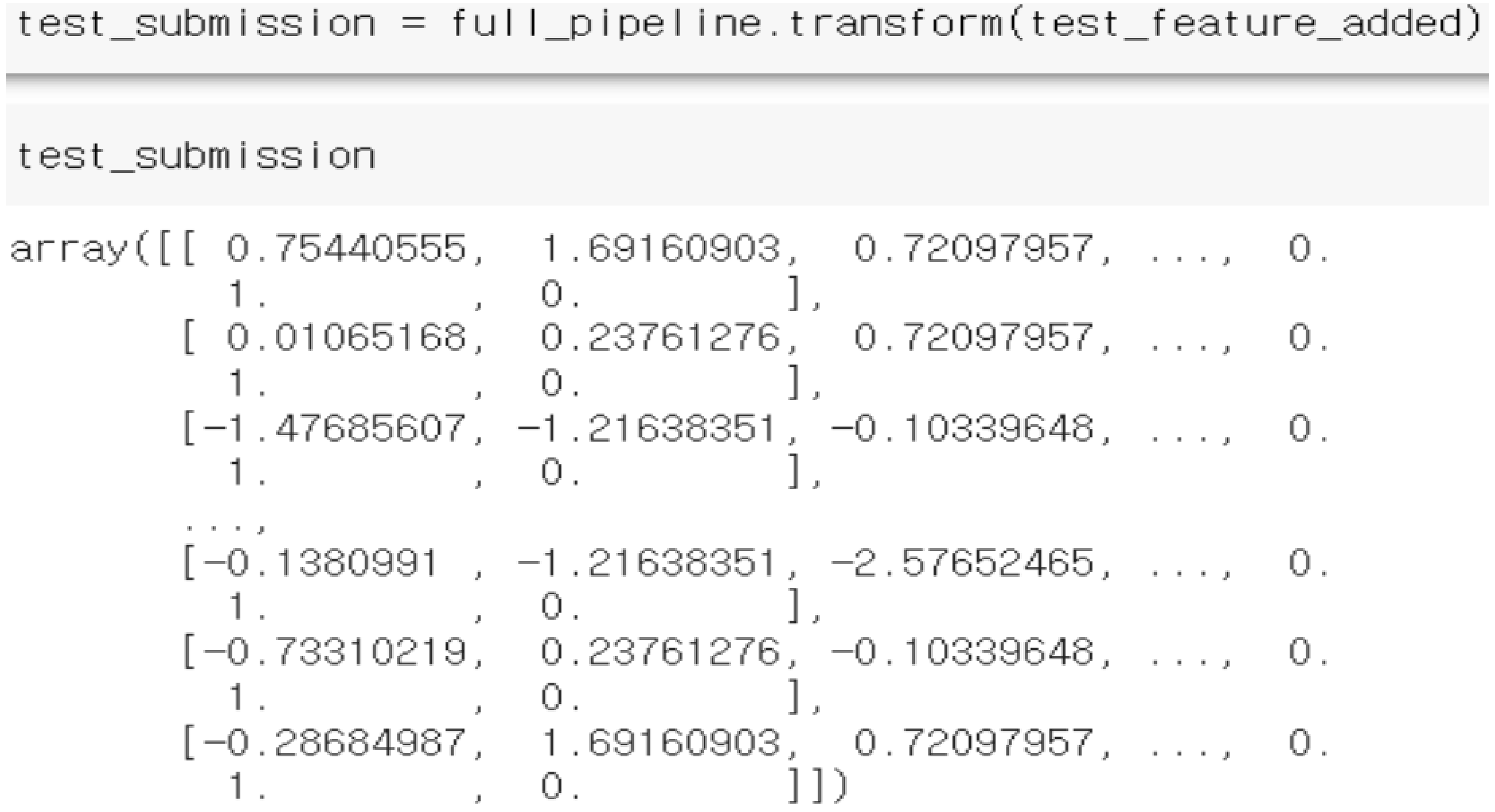
모델성능시각화



```
import matplotlib.pyplot as plt

def visualize(pred):
    plt.scatter(y_test,pred,alpha = 0.005)
    plt.xlabel('True Values')
    plt.ylabel('Predictions')
    plt.axis('equal')
    plt.axis('square')
    plt.xlim([-5,110])
    plt.ylim([-5,110])
    plt.plot([-10,120], [-10,120])

prediction = [lin_pred, tree_pred, forest_pred, forest_best_pred, xgb_pred, xgb_best_pred, lgb_pred, lgb_best_pred, cb_pred, cb_best_pred]
titles = ['lin','tree','forest','forest_GS','xgb','xgb_GS','lgb','lgb_GS','cb','cb_GS']

for i in range(len(prediction)):
    plt.subplot(3,4,i+1)
    visualize(prediction[i])
    plt.title(titles[i])
    plt.tight_layout()


plt.show()
```



파이프라인 통과시키기



```
test_submission = full_pipeline.transform(test_feature_added)
```

```
test_submission
```

```
array([[ 0.75440555,  1.69160903,  0.72097957, ...,  0.
         1.        ,  0.        ],
       [ 0.01065168,  0.23761276,  0.72097957, ...,  0.
         1.        ,  0.        ],
       [-1.47685607, -1.21638351, -0.10339648, ...,  0.
         1.        ,  0.        ],
       ...,
       [-0.1380991 , -1.21638351, -2.57652465, ...,  0.
         1.        ,  0.        ],
       [-0.73310219,  0.23761276, -0.10339648, ...,  0.
         1.        ,  0.        ],
       [-0.28684987,  1.69160903,  0.72097957, ...,  0.
         1.        ,  0.        ]])
```

선정한 모델을 전체 데이터로 학습

```
xgb_reg_sub = xgboost.XGBRegressor(objective='reg:squarederror',
n_estimators=100, learning_rate=0.2, max_depth=10,
min_child_weight=6)
xgb_reg_sub.fit(X_train_prepared,y_train)
```

선정한 모델로 예측

```
target_submission = xgb_reg_sub.predict(test_submission)
```

데이터프레임으로 바꾼 후, id 값과 붙히기

```
target_submission_df = pd.DataFrame(target_submission, columns=['target'])
submission_file = pd.concat([test['id'], target_submission_df], axis = 1)
```

CSV로 저장

```
submission_file.to_csv('submission_file.csv', index = False)
```

저장 함수

```
def submi_fn(model,filename):
 x1 = model.predict(test_submission)
 x2 = pd.DataFrame(x1,columns=['target'])
 x3 = pd.concat([test['id'], x2], axis = 1)
 x3.to_csv(f'./{filename}.csv', index = False)

submi_fn(lin_reg,'lin_reg')
```

THANK YOU

1조

20191534 박성일 코드 만들기

20182490 김진용 발표

20212603 이가빈 PPT