



AI 개발자 트랙 미니프로젝트 1차

# 조별 발표 템플릿

AI 05반 10조



# 1. 데이터 처리 및 분석

## ✓ 시간 처리

### 1) air\_24, air\_25 의 '측정일시'를 활용하여 'time' 변수 생성

```
[ ] from datetime import datetime, timedelta

def zerofrom24(datetime, _format):
    datetime = str(datetime)
    try:
        return pd.to_datetime(datetime, format = _format)
    except:
        datetime = datetime[:-2] + '00'
        return pd.to_datetime(datetime, format = _format)+ timedelta(days=1)

air_24['time'] = air_24['측정일시'].map(lambda x : zerofrom24(x, '%Y%m%d%H'))
air_25['time'] = air_25['측정일시'].map(lambda x : zerofrom24(x, '%Y%m%d%H'))
```

```
[ ] # 결과확인
print(air_24.shape)
air_24
```

	지역	망	측정소코드	측정소명	측정일시	S02	C0	03	N02	PM10	PM25	주소	time
0	서울 종로구	도시대기	111123	종로구	2024010101	0.0031	1.21	0.0022	0.0425	29.0	23.0	서울 종로구 종로35가길 19	2024-01-01 01:00:00
1	서울 종로구	도시대기	111123	종로구	2024010102	0.0032	1.16	0.0020	0.0393	24.0	21.0	서울 종로구 종로35가길 19	2024-01-01 02:00:00
2	서울 종로구	도시대기	111123	종로구	2024010103	0.0030	1.00	0.0022	0.0359	23.0	19.0	서울 종로구 종로35가길 19	2024-01-01 03:00:00
3	서울 종로구	도시대기	111123	종로구	2024010104	0.0020	0.88	0.0021	0.0355	22.0	18.0	서울 종로구 종로35가길 10	2024-01-01 04:00:00

### 문제 상황

air\_24['측정일시'] - 1로 하는 경우,  
time 값은 정상적으로 00시로 나오지만  
나머지 값들은 01시 값을 기준으로 되어있음

이처럼 데이터가 관측된 날짜와  
time 변수에 저장되어있는 값이 다른 현상이 나타나서  
이를 해결하고 싶었습니다.

### 해결 방법

time 변수 생성 과정  
함수 동작 원리 : pd.to\_datetime() 를 이용하여 변환을  
시도 한 후에 실패시(24시 경우) 뒤 두자리를 00으로  
바꾸고 timedelta 로 날짜에 +1을 해줍니다.

map 함수를 이용해서 컬럼에 일괄적으로 적용합니다.

# 1. 데이터 처리 및 분석

## ✓ 분석에 제외할 데이터

```
▶ # df_24, df_25에 사용할 변수들만 할당  
drop_cols = [  
    # 위치  
    '지점', '지점명', '지역', '망', '측정소코드', '측정소명', '주소',  
  
    # 시간 컬럼  
    '일시', '측정일시',  
  
    # QC 플래그  
    '기온 QC플래그', '강수량 QC플래그', '풍속 QC플래그', '풍향 QC플래그', '습도 QC플래그', '현지기압 QC플래그', '해면기압 QC플래그',  
    '일조 QC플래그', '일사 QC플래그', '지면온도 QC플래그',  
  
    # 결측치 너무 많음  
    '적설(cm)', '3시간신적설(cm)',  
  
    # 코드성 변수  
    '지면상태(지면상태코드)', '현상번호(국내식)',  
  
    # 판단 어려움  
    '운형(운형악어)'  
]
```

# 1. 데이터 처리 및 분석

## ✓ 분석에 제외할 데이터

모든 위치 관련 컬럼  
(예: 측정소명, 주소, 지역)

원본 시간 컬럼  
(time, 측정일시)

QC 플래그 컬럼  
(강수량 QC플래그 등)

결측치가 너무 많은 특성

코드성 변수  
(예: 지면상태코드, 현상번호)

단일 값 특성 (Zero Variance): 모든 데이터 포인트의 위치가 '서울'로 동일하여 모델 학습에 기여하는 정보(분산)가 전혀 없습니다. (의미 X)

비수치형 및 정보 중복: Datetime 객체는 숫자형 모델 학습을 방해하며, 이미 필요한 시간 정보는 month, day, hour 등 숫자형 특성으로 분리하여 대체했습니다.

결측치 및 노이즈 관리: 이 플래그는 결측치나 오류 데이터 발생 여부를 나타내는 보조 정보입니다. 예측 성능에 직접적인 기여가 낮고, 결측치 처리(Imputation) 후에는 의미가 사라지므로 제외합니다.

정보 부족: 결측치가 90% 이상인 경우, 보간이나 대체를 해도 원본 정보보다 인위적인 데이터의 비중이 훨씬 높아 모델 학습에 악영향을 줍니다.

값의 의미 해석 어려움: 수치 그 자체의 크기가 아닌, 이산적 상태를 나타내는 코드이므로, 의미 파악이 어려워 모델의 복잡도만 높일 수 있습니다. (판단 어려움)

# 1. 데이터 처리 및 분석

## ✓ 결측치

```
# df_24, df_25의 변수 중 결측치를 처리(결측치 처리 방법은 다양!)
```

```
# 선택해서 결측치를 처리해보세요.
```

```
# 강수량은 0으로 채우기
```

```
df_24['강수량(mm)'] = df_24['강수량(mm)'].fillna(0)
```

```
df_25['강수량(mm)'] = df_25['강수량(mm)'].fillna(0)
```

```
# 나머지 컬럼은 ffill → bfill로 채우기
```

```
df_24 = df_24.fillna().bfill()
```

```
df_25 = df_25.fillna().bfill()
```

### 결측치 처리 방안

날씨는 급변하는 경우가 적기때문에 ffill()과 bfill()을 사용해 앞 뒤에 오는 데이터로 채웠습니다.

### 따로 신경쓴 부분

강수량이 결측인 경우에는 0인 것이라 판단하여 0으로 채웠습니다.

## 2. 딥러닝 모델링

### ✓ 파라미터 튜닝

```
# 랜덤포레스트로 모델 학습 진행하고 성능을 평가하여 출력해 보세요.
# 성능평가는 MSE, R2 Score
model = RandomForestRegressor()
model.fit(train_x, train_y)
y_pred_RF = model.predict(test_x)
print('mse:', mean_squared_error(test_y, y_pred_RF))
print('R2:', r2_score(test_y, y_pred_RF))

/usr/local/lib/python3.12/dist-packages/sklearn/base.py:1389: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples,), for example using ravel().
  return fit_method(estimator, *args, **kwargs)
mse: 63.36674246686514
R2: 0.9163411897035199
```

  

```
▶ # 랜덤포레스트로 모델 학습 진행하고 성능을 평가하여 출력해 보세요.
# 성능평가는 MSE, R2 Score
params = {
    'n_estimators':[100, 200, 300],
    'max_depth' : [5, 10, 15],
    'min_samples_leaf' : [1, 2, 4],
    'min_samples_split' : [5, 10, 15]
}
model_grid = RandomizedSearchCV(RandomForestRegressor(), param_distributions=params, n_iter=20, verbose=1, cv=3)
model_grid.fit(train_x, train_y.values.ravel())
print(model_grid.best_params_)
print(model_grid.best_score_)

model = model_grid.best_estimator_
y_pred_RF = model.predict(test_x)
print('mse:', mean_squared_error(test_y, y_pred_RF))
print('R2:', r2_score(test_y, y_pred_RF))

Fitting 3 folds for each of 20 candidates, totalling 60 fits
{'n_estimators': 100, 'min_samples_split': 15, 'min_samples_leaf': 4, 'max_depth': 15}
0.8280345553307865
mse: 55.43521903817892
R2: 0.9268126419119598
```

### 3. 종합 결과

✓ 가장 좋은 성능의 모델: Gradient Boosting

✓ 팀원들의 공유 사항

김건우: (PM10을 제거하고 진행)

변수 중 시정이 가장 높은 중요도를 보였다.

시정은 공기 중 오염 물질의 양에 의해 영향을 받고, 시정이 낮을수록 부유 물질(미세먼지)이 많다는 의미로 시정은 미세먼지의 간접 지표로 1시간후 농도를 예측하는데 가장 중요한 특성으로 작용했다.

CO는 중요도 2위로 차량 통행, 난방 등 연소 활동에서 나오므로 미세먼지와 발생원인을 공유했다.

이종현: LinearRegression, RandomForest, XGBoost 가 각자 다른 feature\_importances를 관측할 수 있었습니다. 이러한 결과는 모델 선택 시 단일 모델의 feature importance에만 의존해서는 안 되며, 문제의 특성과 변수 간 관계를 함께 고려해야 함을 알게 되었습니다.

이수빈 인사이트:

Feature Importance를 비교해보았을 때, Gradiant Boosting의 경우 상위 5가지 항목에 값이 치중되어 있고, XGBoost 는 더 다양한 Feature에 값이 나누어져 있었는데, 그럼에도 Gradiant Boosting가 XGBoost 보다 성능이 높게 나왔다면 불필요한 Feature들을 덜 정제해서 인것은 아닐까 하는 생각이 들었습니다.

김근우: 사용하지 않아도 되는 열만 삭제하려고 했는데, 너무 많은 열을 제거한 것 같아 과도하게 높은 0.99017이라는  $R^2$  수치가 나온 것 같습니다. 앞으로는 feature 선택 기준을 더 신중하게 해야 할 것 같습니다.

