

AIVLE School 미니프로젝트

네비게이션 주행데이터를 이용한 **도착시각** 예측 문제**[미션 안내]**

- 네비게이션 주행데이터를 읽어들이어 데이터를 분석 및 전처리한 후 머신러닝과 딥러닝으로 도착시각을 예측하고 결과를 분석하세요.

[유의 사항]

- 각 문항의 답안코드는 반드시 '#여기에 답안코드를 작성하세요'로 표시된 cell에 작성해야 합니다.
- 제공된 cell을 추가/삭제하고 다른 cell에 답안코드를 작성 시 채점되지 않습니다.
- 반드시 문제에 제시된 가이드를 읽고 답안 작성하세요.
- 문제에 변수명이 제시된 경우 반드시 해당 변수명을 사용하세요.
- 문제와 데이터는 제3자에게 공유하거나 개인적인 용도로 사용하는 등 외부로 유출할 수 없으며 유출로 인한 책임은 응시자 본인에게 있습니다.

1. **scikit-learn** 패키지는 머신러닝 교육을 위한 최고의 파이썬 패키지입니다.

scikit-learn를 별칭(alias) **sk**로 임포트하는 코드를 작성하고 실행하세요.

In [1]: `# 여기에 답안코드를 작성하세요.`

```
!pip install scikit-learn
```

```
import sklearn as sk
```

Requirement already satisfied: scikit-learn in c:\users\user\anaconda3\lib\site-packages (1.3.0)

Requirement already satisfied: numpy>=1.17.3 in c:\users\user\anaconda3\lib\site-packages (from scikit-learn) (1.24.3)

Requirement already satisfied: scipy>=1.5.0 in c:\users\user\anaconda3\lib\site-packages (from scikit-learn) (1.11.1)

Requirement already satisfied: joblib>=1.1.1 in c:\users\user\anaconda3\lib\site-packages (from scikit-learn) (1.2.0)

Requirement already satisfied: threadpoolctl>=2.0.0 in c:\users\user\anaconda3\lib\site-packages (from scikit-learn) (2.2.0)

2. Pandas는 데이터 분석을 위해 널리 사용되는 파이썬 라이브러리입니다.

Pandas를 사용할 수 있도록 별칭(alias)을 pd로 해서 불러오세요.

```
In [2]: # 여기에 답안코드를 작성하세요.
import pandas as pd
```

3. 모델링을 위해 분석 및 처리할 데이터 파일을 읽어오려고 합니다.

Pandas함수로 데이터 파일을 읽어 데이터프레임 변수명 df에 할당하는 코드를 작성하세요.

- A0007IT.json 파일을 읽어 데이터 프레임 변수명 df에 할당하세요.
- Encoding = "cp949"로 지정하세요.

```
In [14]: # 여기에 답안코드를 작성하세요.
df = pd.read_json('A0007IT.json', encoding='cp949')
```

4. Address1(주소1)에 대한 분포도를 알아 보려고 합니다.

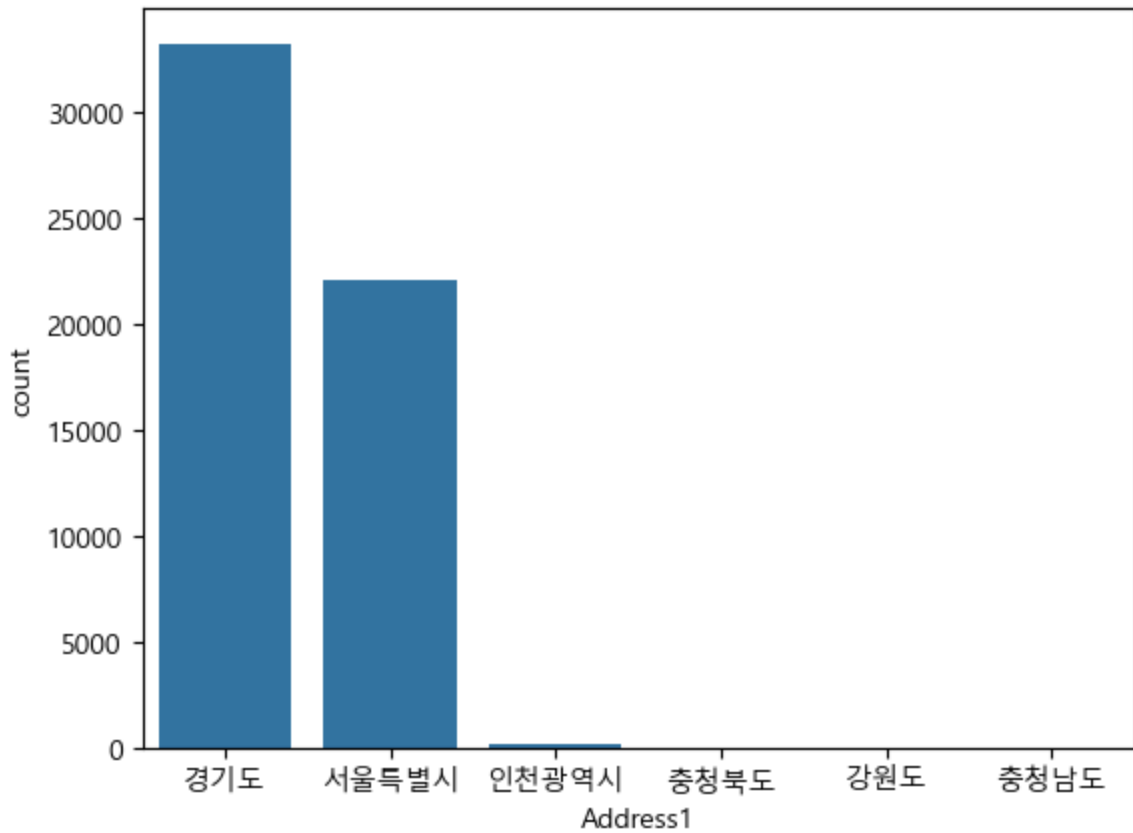
Address1(주소1)에 대해 countplot그래프로 만들고 아래 가이드에 따라 답하세요.

- Seaborn을 활용하세요.
- 첫번째, Address1(주소1)에 대해서 분포를 보여주는 countplot그래프 그리세요.
- 두번째, 지역명이 없는 '-'에 해당되는 row(행)을 삭제하세요.

```
In [17]: # 여기에 답안코드를 작성하세요.
import matplotlib.pyplot as plt
import seaborn as sns
plt.rcParams['font.family'] = 'Malgun Gothic'

df = df.loc[df['Address1'] != '-']

sns.countplot(x='Address1', data=df)
plt.show()
```

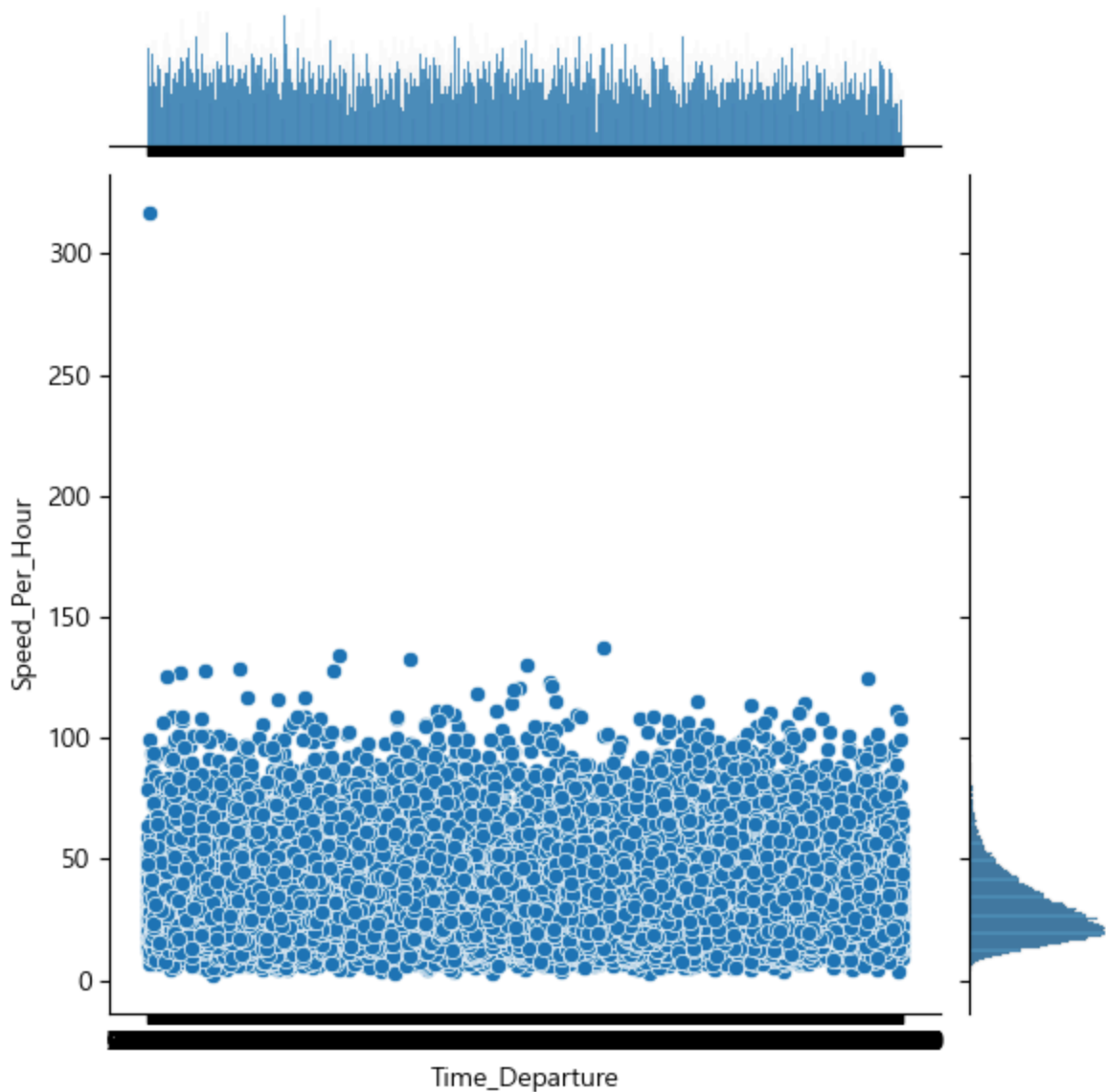


5. 실주행시간과 평균시속의 분포를 같이 확인하려고 합니다.

Time_Driving(실주행시간)과 **Speed_Per_Hour(평균시속)**을 **jointplot** 그래프로 만드세요.

- Seaborn을 활용하세요.
- X축에는 Time_Driving(실주행시간)을 표시하고 Y축에는 Speed_Per_Hour(평균시속)을 표시하세요.

```
In [19]: # 여기에 답안코드를 작성하세요.
sns.jointplot(x='Time_Departure', y='Speed_Per_Hour', data=df)
plt.show()
```



6. 위의 **jointplot** 그래프에서 시속 300이 넘는 이상치를 발견할 수 있습니다.

jointplot 그래프에서 발견한 이상치 1개를 삭제하세요.

- 대상 데이터프레임: df
- jointplot 그래프를 보고 시속 300 이상되는 이상치를 찾아 해당 행(Row)을 삭제하세요.
- 전처리 반영 후에 새로운 데이터프레임 변수명 df_temp에 저장하세요.

```
In [20]: # 여기에 답안코드를 작성하세요.
df_temp = df.loc[df['Speed_Per_Hour'] < 300]
```

7. 모델링 성능을 제대로 얻기 위해서 결측치 처리는 필수입니다.

아래 가이드를 따라 결측치 처리하세요.

- 대상 데이터프레임: df_temp
- 결측치를 확인하는 코드를 작성하세요.
- 결측치가 있는 행(row)를 삭제 하세요.
- 전처리 반영된 결과를 새로운 데이터프레임 변수명 df_na에 저장하세요.

```
In [24]: # 여기에 답안코드를 작성하세요.
print(df_temp.isna().sum())
df_na = df_temp.dropna()
```

```
Time_Departure    0
Time_Arrival      0
Distance          2
Time_Driving      3
Speed_Per_Hour    0
Address1          0
Address2          0
Signaltype        0
Weekday           0
Hour              0
Day              0
dtype: int64
```

8. 모델링 성능을 제대로 얻기 위해서 불필요한 변수는 삭제해야 합니다.

아래 가이드를 따라 불필요 데이터를 삭제 처리하세요.

- 대상 데이터프레임: df_na
- 'Time_Departure', 'Time_Arrival' 2개 컬럼을 삭제하세요.
- 전처리 반영된 결과를 새로운 데이터프레임 변수명 df_del에 저장하세요.

```
In [27]: # 여기에 답안코드를 작성하세요.
drop_cols = ['Time_Departure', 'Time_Arrival']
df_del = df_na.drop(drop_cols, axis=1)
```

9. 원-핫 인코딩(One-hot encoding)은 범주형 변수를 1과 0의 이진형 벡터로 변환하기 위하여 사용하는 방법입니다.

원-핫 인코딩으로 아래 조건에 해당하는 컬럼 데이터를 변환하세요.

- 대상 데이터프레임: df_del
- 원-핫 인코딩 대상: object 타입의 전체 컬럼
- 활용 함수: pandas의 get_dummies

- 해당 전처리가 반영된 결과를 데이터프레임 변수 df_preset에 저장해 주세요.

```
In [43]: # 여기에 답안코드를 작성하세요.
dumm_cols = df_del.select_dtypes('object').columns
#df_del.select_dtypes('object').columns.values

df_preset = pd.get_dummies(df_del, columns=dumm_cols, drop_first=True, dtype=int)
```

10. 훈련과 검증 각각에 사용할 데이터셋을 분리하려고 합니다.

Time_Driving(실주행시간) 컬럼을 **label값 y**로, 나머지 컬럼을 **feature값 X**로 할당한 후 훈련데이터셋과 검증데이터셋으로 분리하세요.

- 대상 데이터프레임: df_preset
- 훈련 데이터셋 label: y_train, 훈련 데이터셋 Feature: X_train
- 검증 데이터셋 label: y_valid, 검증 데이터셋 Feature: X_valid
- 훈련 데이터셋과 검증데이터셋 비율은 80:20
- random_state: 42
- Scikit-learn의 train_test_split 함수를 활용하세요.

```
In [47]: # 여기에 답안코드를 작성하세요.
from sklearn.model_selection import train_test_split

target = 'Time_Driving'
x = df_preset.drop(target, axis=1)
y = df_preset.loc[:, target]

x_train, x_valid, y_train, y_valid = train_test_split(x, y, test_size=.2, random_state=42)
```

11. **Time_Driving(실주행시간)**을 예측하는 머신러닝 모델을 만들려고 합니다.

의사결정나무(decision tree)는 여러 가지 규칙을 순차적으로 적용하면서 독립 변수 공간을 분할하는 모형으로

분류(classification)와 **회귀 분석(regression)**에 모두 사용될 수 있습니다.

의사결정나무(decision tree)로 학습을 진행하세요.

- 트리의 최대 깊이: 5로 설정
- 노드를 분할하기 위한 최소한의 샘플 데이터수(min_samples_split): 3로 설정
- random_state: 120로 설정

```
In [48]: # 여기에 답안코드를 작성하세요.
from sklearn.tree import DecisionTreeRegressor

model = DecisionTreeRegressor(max_depth=5, min_samples_split=3, random_state=120)
model.fit(x_train, y_train)
```

```
Out[48]: DecisionTreeRegressor
DecisionTreeRegressor(max_depth=5, min_samples_split=3, random_state=120)
```

12. 위 의사결정나무(decision tree) 모델의 성능을 평가하려고 합니다.

예측 결과의 mae(Mean Absolute Error)를 구하세요.

- 성능 평가는 검증 데이터셋을 활용하세요.
- 11번 문제에서 만든 의사결정나무(decision tree) 모델로 y값을 예측(predict)하여 y_pred에 저장하세요.
- 검증 정답(y_valid)과 예측값(y_pred)의 mae(Mean Absolute Error)를 구하고 dt_mae 변수에 저장하세요.

```
In [52]: # 여기에 답안코드를 작성하세요.
from sklearn.metrics import *

y_pred = model.predict(x_valid)
dt_mae = mean_absolute_error(y_valid, y_pred)
```

다음 문항을 풀기 전에 아래 코드를 실행하세요.

```
In [54]: import tensorflow as tf
from tensorflow.keras.models import Sequential, load_model
from tensorflow.keras.layers import Dense, Activation, Dropout, BatchNormalization
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint
from tensorflow.keras.utils import to_categorical

tf.random.set_seed(1)
```

13. Time_Driving(실주행시간)을 예측하는 딥러닝 모델을 만들려고 합니다.

아래 가이드에 따라 모델링하고 학습을 진행하세요.

- Tensorflow framework를 사용하여 딥러닝 모델을 만드세요.
- 히든레이어(hidden layer) 2개 이상으로 모델을 구성하세요.

- dropout 비율 0.2로 Dropout 레이어 1개를 추가해 주세요.
- 손실함수는 MSE(Mean Squared Error)를 사용하세요.
- 하이퍼파라미터 epochs: 30, batch_size: 16으로 설정해주세요.
- 각 에포크마다 loss와 metrics 평가하기 위한 데이터로 X_valid, y_valid 사용하세요.
- 학습정보는 history 변수에 저장해주세요

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

```
scaler = MinMaxScaler()
x_train_s = scaler.fit_transform(x_train)
x_valid_s = scaler.transform(x_valid)
```

In [91]: `# 여기에 답안코드를 작성하세요.`

```
from keras.backend import clear_session
clear_session()

n = x_train.shape[1]
model = Sequential([Dense(64, input_shape=(n,), activation='relu'),
                    Dense(64, activation='relu'),
                    Dense(128, activation='relu'),
                    Dense(128, activation='relu'),
                    Dropout(0.2),
                    Dense(256, activation='relu'),
                    Dense(256, activation='relu'),
                    Dense(1, activation='linear'),
                    #Dense(1),
                    ])

model.summary()

es = EarlyStopping(monitor='val_loss', patience=5)
mc = ModelCheckpoint('best_model.keras', monitor='val_loss', save_best_only=True)

model.compile(optimizer='adam', loss='mse', metrics=['mse'])
history = model.fit(x_train, y_train, epochs=30, batch_size=16, validation_data=(x_valid, y_val))


















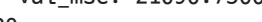
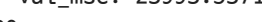

C:\Users\User\anaconda3\Lib\site-packages\keras\src\layers\core\dense.py:86: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

Model: "sequential"

Layer (type)	Output Shape	Pa
dense (Dense)	(None, 64)	
dense_1 (Dense)	(None, 64)	
dense_2 (Dense)	(None, 128)	
dense_3 (Dense)	(None, 128)	
dropout (Dropout)	(None, 128)	
dense_4 (Dense)	(None, 256)	
dense_5 (Dense)	(None, 256)	
dense_6 (Dense)	(None, 1)	

Total params: 133,697 (522.25 KB)
Trainable params: 133,697 (522.25 KB)
Non-trainable params: 0 (0.00 B)

```

Epoch 1/30
2772/2772  13s 3ms/step - loss: 174372.2031 - mse: 174372.2031 - val_loss: 44038.0781 - val_mse: 44038.0781
Epoch 2/30
2772/2772  9s 3ms/step - loss: 52937.7305 - mse: 52937.7305 - val_loss: 48970.6406 - val_mse: 48970.6406
Epoch 3/30
2772/2772  9s 3ms/step - loss: 64570.3281 - mse: 64570.3281 - val_loss: 66805.2031 - val_mse: 66805.2031
Epoch 4/30
2772/2772  9s 3ms/step - loss: 41409.9141 - mse: 41409.9141 - val_loss: 43509.1094 - val_mse: 43509.1094
Epoch 5/30
2772/2772  10s 3ms/step - loss: 64236.9844 - mse: 64236.9844 - val_loss: 41746.3867 - val_mse: 41746.3867
Epoch 6/30
2772/2772  10s 3ms/step - loss: 25587.4883 - mse: 25587.4883 - val_loss: 17971.6055 - val_mse: 17971.6055
Epoch 7/30
2772/2772  10s 4ms/step - loss: 23247.7598 - mse: 23247.7598 - val_loss: 13317.5527 - val_mse: 13317.5527
Epoch 8/30
2772/2772  9s 3ms/step - loss: 34994.4766 - mse: 34994.4766 - val_loss: 21040.3145 - val_mse: 21040.3145
Epoch 9/30
2772/2772  11s 3ms/step - loss: 28443.5645 - mse: 28443.5645 - val_loss: 15978.2998 - val_mse: 15978.2998
Epoch 10/30
2772/2772  11s 4ms/step - loss: 27833.9062 - mse: 27833.9062 - val_loss: 13222.9844 - val_mse: 13222.9844
Epoch 11/30
2772/2772  9s 3ms/step - loss: 26741.0625 - mse: 26741.0625 - val_loss: 23421.4414 - val_mse: 23421.4414
Epoch 12/30
2772/2772  9s 3ms/step - loss: 21574.9023 - mse: 21574.9023 - val_loss: 14855.6328 - val_mse: 14855.6328
Epoch 13/30
2772/2772  9s 3ms/step - loss: 21915.1562 - mse: 21915.1562 - val_loss: 6174.2588 - val_mse: 6174.2588
Epoch 14/30
2772/2772  9s 3ms/step - loss: 26548.5234 - mse: 26548.5234 - val_loss: 10257.4766 - val_mse: 10257.4766
Epoch 15/30
2772/2772  10s 3ms/step - loss: 15753.3291 - mse: 15753.3291 - val_loss: 5857.3296 - val_mse: 5857.3296
Epoch 16/30
2772/2772  9s 3ms/step - loss: 15731.9336 - mse: 15731.9336 - val_loss: 4438.4463 - val_mse: 4438.4463
Epoch 17/30
2772/2772  9s 3ms/step - loss: 35362.9023 - mse: 35362.9023 - val_loss: 21090.7500 - val_mse: 21090.7500
Epoch 18/30
2772/2772  9s 3ms/step - loss: 13727.7617 - mse: 13727.7617 - val_loss: 23993.5371 - val_mse: 23993.5371
Epoch 19/30
2772/2772  10s 3ms/step - loss: 27589.6914 - mse: 27589.6914 - val_loss: 10454.4473 - val_mse: 10454.4473
Epoch 20/30
2772/2772  9s 3ms/step - loss: 26698.7676 - mse: 26698.7676 - val_loss: 19625.2891 - val_mse: 19625.2891

```

Epoch 21/30

2772/2772 ————— 9s 3ms/step - loss: 18394.8555 - mse: 18394.8555 - val_loss: 10

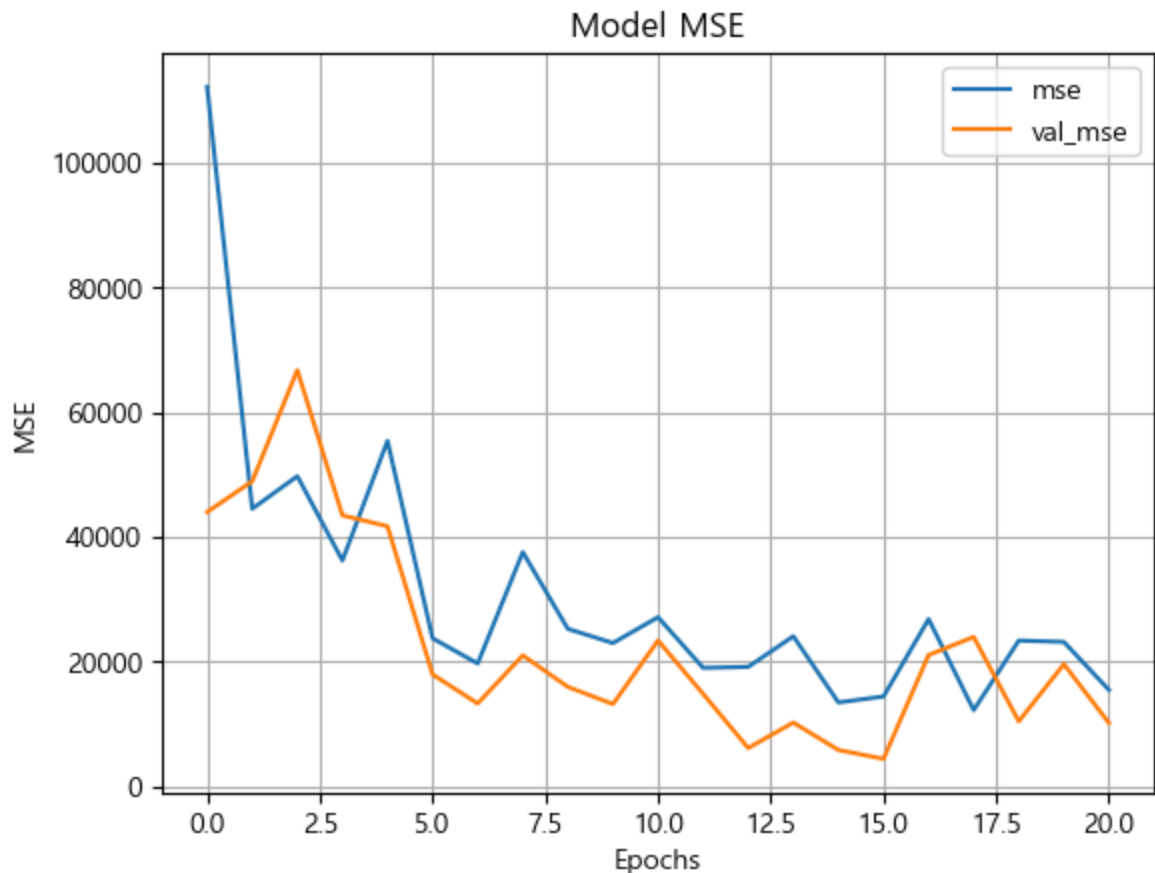
14. 위 딥러닝 모델의 성능을 평가하려고 합니다.

Matplotlib 라이브러리 활용해서 학습 **mse**와 검증 **mse**를 그래프로 표시하세요.

- 1개의 그래프에 학습 mse와 검증 mse 2가지를 모두 표시하세요.
- 위 2가지 각각의 범례를 'mse', 'val_mse'로 표시하세요.
- 그래프의 타이틀은 'Model MSE'로 표시하세요.
- X축에는 'Epochs'라고 표시하고 Y축에는 'MSE'라고 표시하세요.

```
In [92]: # 여기에 답안코드를 작성하세요.
plt.plot(history['mse'], label='mse')
plt.plot(history['val_mse'], label='val_mse')

plt.title('Model MSE')
plt.xlabel('Epochs')
plt.ylabel('MSE')
plt.legend()
plt.grid()
plt.show()
```



```
In [93]: pred = model.predict(x_valid)
```

347/347  1s 2ms/step

```
In [94]: r2_score(y_valid, pred)
```

```
Out[94]: 0.9624923347754795
```

```
In [ ]:
```