AIVLE School 미니프로젝트

네비게이션 주행데이터를 이용한 도착시각 예측 문제

[미션 안내]

- 네비게이션 주행데이터를 읽어들여 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) sk로 임포트하는 코드를 작성하고 실행하세요.

```
# 여기에 답안코드를 작성하세요
import sklearn as sk
```

2. Pandas는 데이터 분석을 위해 널리 사용되는 파이썬 라이브러리입니다.

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

```
# 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.

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
# 여기에 답안코드를 작성하세요.
df = pd.read_json('/content/drive/MyDrive/미프5/A0007IT.json')
```

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.

```
# 여기에 답안코드를 작성하세요.
import seaborn as sns

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

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

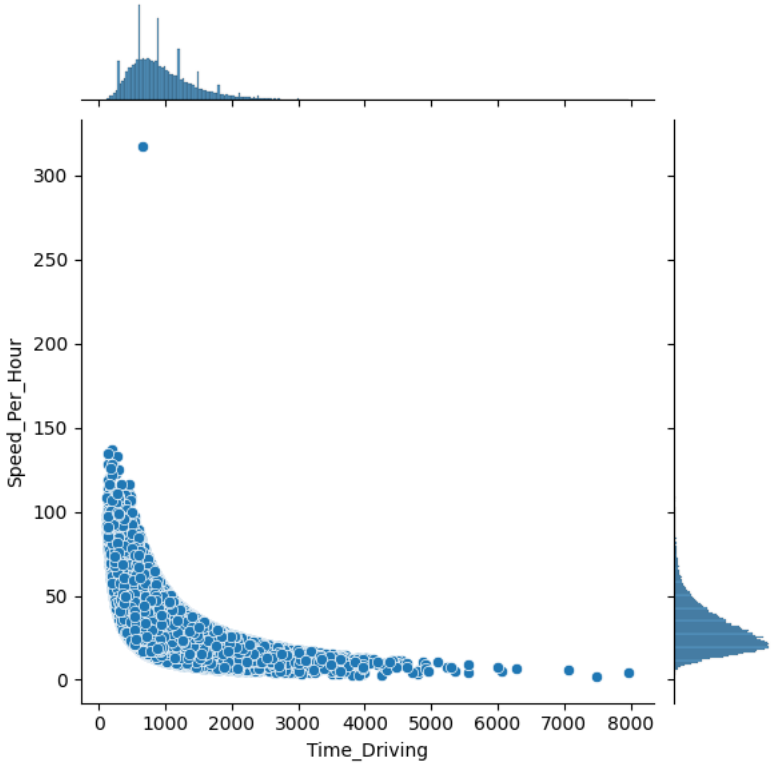
5. 실주행시간과 평균시속의 분포를 같이 확인하려고 합니다.

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.

```
1 # 여기에 답안코드를 작성하세요.
2 sns.jointplot(x='Time_Driving', y='Speed_Per_Hour', data=df)
```

```
<seaborn.axisgrid.JointGrid at 0x7b0b4c234c40>
```




```
/usr/local/lib/python3.10/dist-packages/IPython/core/pylabtools.py:151: UserWarning: Glyph 45224 (₩N{HANGUL SYLLABLE
```

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.

```
# 여기에 답안코드를 작성하세요.
df_temp= df.copy()
df_temp.Speed_Per_Hour = df[df.Speed_Per_Hour < 300].Speed_Per_Hour
```

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(raw)를 삭제 하세요.
- 전처리 반영된 결과를 새로운 데이터프레임 변수명 df_na에 저장하세요.

```
# 여기에 답안코드를 작성하세요.
print(df_temp.isnull().sum())
df_na = df_temp.dropna()
```

```
Time_Departure     0
Time_Arrival       0
Distance           2
Time_Driving       3
Speed_Per_Hour     6
Address1          90
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.

```
# 여기에 답안코드를 작성하세요.
df_del = df_na.drop(columns=['Time_Departure', 'Time_Arrival'])
```

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.

```
# 여기에 답안코드를 작성하세요.
cols = df_del.select_dtypes(include='object').columns
df_preset = pd.get_dummies(df_del, columns=cols)
```

- 황성업 : 반복문을 사용하지 않고 select_dtypes로 해결한다면 코드가 훨씬 간결해지겠네요!

10. 훈련과 검증 각각에 사용할 데이터셋을 분리하려고 합니다.

Time_Driving(실주행시간) 컬럼을 label값 y로, 나머지 컬럼을 feature값 X로 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.

```
# 여기에 답안코드를 작성하세요.
from sklearn.model_selection import train_test_split

X = df_preset.drop('Time_Driving', axis=1)
y = df_preset['Time_Driving']

X_train, X_valid, y_train, y_valid = train_test_split(X, y, test_size=0.2, 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

```
# 여기에 답안코드를 작성하세요.
from sklearn.tree import DecisionTreeRegressor

dt = DecisionTreeRegressor(max_depth=5, min_samples_split=3, random_state=120)
dt.fit(X_train, y_train)
```

```
▾ 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.

```
# 여기에 답안코드를 작성하세요.
from sklearn.metrics import mean_absolute_error

y_pred = dt.predict(X_valid)
dt_mae = mean_absolute_error(y_valid, y_pred)
```

다음 문항을 풀기 전에 아래 코드를 실행하세요.

```
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
```

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

아래 가이드에 따라 모델링하고 학습을 진행하세요.

- Tensoflow framework를 사용하여 딥러닝 모델을 만드세요.
- 히든레이어(hidden layer) 2개이상으로 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

```
# 여기에 답안코드를 작성하세요.
model = Sequential()

model.add(Dense(128, activation='relu', input_shape=(X_valid.shape[1],)))
model.add(Dense(64, activation='relu'))
model.add(Dense(32, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(1, activation='linear'))

model.compile(optimizer='adam', loss='mean_squared_error', metrics=['mae'])
history = model.fit(X_train, y_train, epochs=30, batch_size=16, validation_data=(X_valid, y_valid), verbose=1)
```

```
Epoch 1/30
2772/2772 [==============================] - 8s 3ms/step - loss: 148514.0312 - mae: 258.9168 - val_loss: 60850.1328 - val_mae: 161.8190
Epoch 2/30
2772/2772 [==============================] - 11s 4ms/step - loss: 61647.5078 - mae: 169.4059 - val_loss: 27159.3027 - val_mae: 111.4418
Epoch 3/30
2772/2772 [==============================] - 7s 3ms/step - loss: 44458.0430 - mae: 143.3756 - val_loss: 23391.2148 - val_mae: 98.2396
Epoch 4/30
2772/2772 [==============================] - 8s 3ms/step - loss: 37055.0312 - mae: 129.6944 - val_loss: 22719.5352 - val_mae: 98.2359
Epoch 5/30
2772/2772 [==============================] - 9s 3ms/step - loss: 32961.3047 - mae: 120.8363 - val_loss: 11260.8818 - val_mae: 71.5278
Epoch 6/30
2772/2772 [==============================] - 7s 3ms/step - loss: 29500.3867 - mae: 114.5732 - val_loss: 6112.5864 - val_mae: 49.1317
Epoch 7/30
2772/2772 [==============================] - 9s 3ms/step - loss: 28513.6133 - mae: 112.0693 - val_loss: 10594.4561 - val_mae: 61.8819
Epoch 8/30
2772/2772 [==============================] - 8s 3ms/step - loss: 24793.7715 - mae: 106.1438 - val_loss: 6102.0020 - val_mae: 44.6898
Epoch 9/30
2772/2772 [==============================] - 9s 3ms/step - loss: 25405.2695 - mae: 105.7974 - val_loss: 3831.0283 - val_mae: 36.9975
Epoch 10/30
2772/2772 [==============================] - 7s 3ms/step - loss: 25856.6387 - mae: 106.1837 - val_loss: 51458.8164 - val_mae: 178.1608
Epoch 11/30
2772/2772 [==============================] - 9s 3ms/step - loss: 24261.0254 - mae: 103.0578 - val_loss: 2831.6211 - val_mae: 33.6777
Epoch 12/30
2772/2772 [==============================] - 9s 3ms/step - loss: 24652.7129 - mae: 103.9368 - val_loss: 7016.6919 - val_mae: 53.5744
Epoch 13/30
2772/2772 [==============================] - 7s 3ms/step - loss: 24139.2598 - mae: 102.5301 - val_loss: 5760.2222 - val_mae: 46.9970
Epoch 14/30
2772/2772 [==============================] - 9s 3ms/step - loss: 23531.3203 - mae: 101.2748 - val_loss: 2085.2156 - val_mae: 31.6545
Epoch 15/30
2772/2772 [==============================] - 7s 2ms/step - loss: 23645.7715 - mae: 101.7941 - val_loss: 19667.0898 - val_mae: 102.4977
Epoch 16/30
2772/2772 [==============================] - 9s 3ms/step - loss: 22281.9219 - mae: 99.7355 - val_loss: 45751.5820 - val_mae: 128.1630
Epoch 17/30
2772/2772 [==============================] - 7s 3ms/step - loss: 21731.1484 - mae: 98.7403 - val_loss: 3473.8267 - val_mae: 31.3906
Epoch 18/30
2772/2772 [==============================] - 9s 3ms/step - loss: 23210.8672 - mae: 100.8446 - val_loss: 6720.1787 - val_mae: 43.5697
Epoch 19/30
2772/2772 [==============================] - 8s 3ms/step - loss: 23748.1211 - mae: 100.1209 - val_loss: 3994.6926 - val_mae: 33.7454
Epoch 20/30
2772/2772 [==============================] - 8s 3ms/step - loss: 20117.8105 - mae: 96.1620 - val_loss: 5111.6094 - val_mae: 40.5208
Epoch 21/30
2772/2772 [==============================] - 8s 3ms/step - loss: 20864.3379 - mae: 96.9965 - val_loss: 41500.9375 - val_mae: 132.7883
Epoch 22/30
2772/2772 [==============================] - 8s 3ms/step - loss: 20919.3574 - mae: 96.5023 - val_loss: 2014.9163 - val_mae: 24.0584
Epoch 23/30
2772/2772 [==============================] - 8s 3ms/step - loss: 20715.7227 - mae: 96.8813 - val_loss: 6728.9497 - val_mae: 46.5182
Epoch 24/30
2772/2772 [==============================] - 7s 3ms/step - loss: 20359.1621 - mae: 96.2040 - val_loss: 4040.1350 - val_mae: 45.4051
Epoch 25/30
2772/2772 [==============================] - 9s 3ms/step - loss: 29845.0664 - mae: 109.7644 - val_loss: 15907.7422 - val_mae: 67.7674
Epoch 26/30
2772/2772 [==============================] - 7s 3ms/step - loss: 29466.8047 - mae: 115.6782 - val_loss: 14386.2510 - val_mae: 71.8840
Epoch 27/30
2772/2772 [==============================] - 8s 3ms/step - loss: 24423.7441 - mae: 105.2120 - val_loss: 3275.5195 - val_mae: 28.2036
Epoch 28/30
```

```
2772/2772 [==============================] - 7s 3ms/step - loss: 23552.8027 - mae: 104.8235 - val_loss: 5212.0645 - val_mae: 35.2087
Epoch 29/30
2772/2772 [==============================] - 9s 3ms/step - loss: 22491.9785 - mae: 102.8338 - val_loss: 15786.0791 - val_mae: 106.8777
```

- 황성업 : 완벽하지만 loss 값이 너무 높네요^^

14. 위 딥러닝 모델의 성능을 평가하려고 합니다.

Matplotlib 라이브러리 활용해서 학습 mse와 검증 mse를 그래프로 표시하세요.

- 1개의 그래프에 학습 mse과 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.

```
# 여기에 답안코드를 작성하세요.
history.history.keys()

plt.figure()
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.legend(['mse', 'val_mse'])
plt.title('Model MSE')
plt.xlabel('Epochs')
plt.ylabel('MSE')
plt.show()
```