# AICE ASSOCIATE 자격시험

시험 시간: 04시 45분 - 06시 15분 (90분)

남은 시간: 56분 2초 ✔ 임시저장 ✔ 최종제출

# AICE Associate 자격인증 샘플문항

## 내비게이션 주행데이터를 이용한 도착시각 예측 문제

- 내비게이션 주행데이터를 읽어들여 데이터를 분석 및 전처리한 후 머신러닝과 딥러닝으로 도착시각을 예측하고 결과를 분석하세요.

---

**[유의사항]**

- 각 문항의 답안코드는 반드시 '# *여기에 답안코드를 작성하세요*' 로 표시된 cell에 작성해야 합니다.
- 제공된 시험문항 cell을 삭제하거나 답안 위치가 아닌 다른 cell에 답안코드를 작성 시 채점되지 않습니다.
- 시험 중에는 상단의 '임시저장' 버튼을 클릭하여 저장을 해주시고, 답안 제출시에는 '최종제출' 버튼을 클릭해주시기 바랍니다.
- 반드시 문제에 제시된 가이드를 읽고 답안 작성하세요.
- 문제에 변수명이 제시된 경우 반드시 해당 변수명을 사용하세요.
- 자격인증 문제에 제공된 데이터는 제 3자에게 공유하거나 개인적인 용도로 사용하는 등 외부로 유출할 수 없으며 유출로 인한 책임은 응시자 본인에게 있습니다.
- **Copy & Paste를 위한 메모장, 노션 등을 사용할 경우 부정행위로 간주될 수 있습니다.**
- **생성형 AI를 활용한 검색은 부정행위로 별도의 경고조치 없이 적발 즉시 시험 중단될 수 있습니다.**

---

**[ 데이터 컬럼 설명 (데이터 파일명: A0007IT.json) ]**

- Time_Departure : 출발시각
- Time_Arrival : 도착시각
- Distance : 이동 거리, 단위 (m)
- Time_Driving : 실주행시간(초)
- Speed_Per_Hour : 평균시속
- Address1 : 주소1
- Address2 : 주소2
- Weekday : Time_Departure(출발시각)의 요일
- Hour : Time_Departure(출발시각)의 시각
- Day : Time_Departure(출발시각)의 날짜

**[ 데이터 컬럼 설명 (데이터 파일명: signal.csv) ]**

- RID : 경로ID
- Signaltype : 경로의 신호등 갯수

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

## scikit-learn를 별칭(alias) sk로 임포트하는 코드를 작성하고 실행하세요.

---

```
[ ]: import sklearn as sk
```

## 2. Pandas는 데이터 분석을 위해 널리 사용되는 파이썬 라이브러리입니다.

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

]:
```
import pandas as pd
```

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

## Pandas함수로 2개 데이터 파일을 읽고 합쳐서 1개의 데이터프레임 변수명 df에 할당하는 코드를 작성하세요.

- A0007IT.json 파일을 읽어 데이터 프레임 변수명 df_a에 할당하세요.
- signal.csv 파일을 읽어 데이터 프레임 변수명 df_b에 할당하세요.
- df_a와 df_b 데이터프레임을 판다스의 merge 함수를 활용하여 합쳐 데이터프레임 변수명 df에 저장하세요.
  - 합칠때 사용하는 키(on) : 'RID'
  - 합치는 방법(how) : 'inner'

]:
```
df_a = pd.read_json('A0007IT.json')
df_b = pd.read_csv('signal.csv')

df = pd.merge(df_a, df_b, on='RID', how='inner')
df.head()
```

## 4. Address1(주소1)에 대한 분포도를 알아 보려고 합니다.

## Address1(주소1)에 대해 countplot그래프로 만드는 코드와 답안을 작성하세요.

- Seaborn을 활용하세요.
- 첫번째, Address1(주소1)에 대해서 분포를 보여주는 countplot 그래프 그리세요.
- 두번째, 지역명이 없는 '-' 에 해당되는 row(행)을 삭제하세요.
- 출력된 그래프를 보고 해석한 것으로 옳지 않은 선택지를 아래에서 골라 '답안04' 변수에 저장하세요.(예. 답안04 = 4)
  1. Countplot 그래프에서 Address1(주소1) 분포를 확인시 '경기도' 분포가 제일 크다.
  2. Address1(주소1) 분포를 보면 '인천광역시' 보다 '서울특별시'가 더 크다.
  3. 지역명이 없는 '-' 에 해당되는 row(행)가 2개 있다.

]:
```
import seaborn as sns
sns.countplot(data=df, x='Address1')
print(df['Address1'].value_counts())
df = df[df['Address1'] != '-']
답안04 = 3
```

```
경기도       6014
서울특별시     3956
인천광역시       27
Name: Address1, dtype: int64
```

### 5. 실주행시간과 평균시속의 분포를 같이 확인하려고 합니다.

### Time_Driving(실주행시간)과 Speed_Per_Hour(평균시속)을 jointplot그래프로 만드세요.

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

```
sns.jointplot(data=df, x='Time_Driving', y='Speed_Per_Hour')
```

```
<seaborn.axisgrid.JointGrid at 0x7fe8c418f310>
```

### 6. 위의 jointplot 그래프에서 시속 300이 넘는 이상치를 발견할수 있습니다.

### 가이드에 따라서 전처리를 수행하고 저장하세요.

- 대상 데이터프레임 : df
- jointplot 그래프를 보고 시속 300 이상되는 이상치를 찾아 해당 행(Row)을 삭제하세요.
- 불필요한 'RID' 컬럼을 삭제 하세요.
- 전처리 반영 후에 새로운 데이터프레임 변수명 df_temp 에 저장하세요.

```
df_temp = df[df['Speed_Per_Hour']<300].drop('RID', axis=1)
df_temp.head()
```

| | Time_Departure | Time_Arrival | Distance | Time_Driving | Speed_Per_Hour | Address 1 | Address 2 | Weekday | Hour | Day | Signalty pe |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 19:43:56 | 19:56:38.537000 | 3246.0 | 762.653992 | 15.322283 | 경기도 | 평택시 | 0 | 19 | 27 | 11.0 |
| 1 | 12:23:05 | 12:36:32.620000 | 3858.0 | 796.081970 | 17.446445 | 경기도 | 의정부시 | 4 | 12 | 24 | 17.0 |
| 2 | 19:10:35 | 19:29:07.306000 | 4125.0 | 1112.468018 | 13.348698 | 경기도 | 광명시 | 0 | 19 | 27 | 24.0 |
| 3 | 11:59:58 | 12:10:55.999000 | 3748.0 | 651.973022 | 20.695335 | 경기도 | 안산시 상록구 | 0 | 11 | 20 | 19.0 |
| 4 | 15:16:53 | 15:26:52.222000 | 3153.0 | 597.419006 | 18.999730 | 경기도 | 성남시 중원구 | 2 | 15 | 29 | 14.0 |

### 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_na 에 저장하세요.
- 결측치 개수를 '답안07' 변수에 저장하세요.(예. 답안07 = 5)

```
df_temp.isnull().sum()
df_na = df_temp.dropna()
답안07 = 2
```

## 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_del 에 저장하세요.

---

```
df_del = df_na.drop(['Time_Departure', 'Time_Arrival'], axis=1)
df_del.head()
```

| | Distance | Time_Driving | Speed_Per_Hour | Address1 | Address2 | Weekday | Hour | Day | Signaltype |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 3246.0 | 762.653992 | 15.322283 | 경기도 | 평택시 | 0 | 19 | 27 | 11.0 |
| 1 | 3858.0 | 796.081970 | 17.446445 | 경기도 | 의정부시 | 4 | 12 | 24 | 17.0 |
| 2 | 4125.0 | 1112.468018 | 13.348698 | 경기도 | 광명시 | 0 | 19 | 27 | 24.0 |
| 3 | 3748.0 | 651.973022 | 20.695335 | 경기도 | 안산시 상록구 | 0 | 11 | 20 | 19.0 |
| 4 | 3153.0 | 597.419006 | 18.999730 | 경기도 | 성남시 중원구 | 2 | 15 | 29 | 14.0 |

## 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.

---

```
obj_cols = df_del.select_dtypes('object').columns
print(obj_cols)

df_preset = pd.get_dummies(df_del, columns=obj_cols)
df_preset.head()
```

```
Index(['Address1', 'Address2'], dtype='object')
```

### 10. 훈련과 검증 각각에 사용할 데이터셋을 분리하려고 합니다.

### Time_Driving(실주행시간) 컬럼을 label값 y로, 나머지 컬럼을 feature값 X로 할당한 후 훈련데이터셋과 검증데이터셋으로 분리하세요.

### 추가로, 가이드 따라서 훈련데이터셋과 검증데이터셋에 스케일링을 수행하세요.

---

- 대상 데이터프레임 : df_preset
- 훈련과 검증 데이터셋 분리
  - 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.
- RobustScaler 스케일링 수행
  - sklearn.preprocessing의 RobustScaler 함수 사용
  - 훈련데이터셋의 Feature는 RobustScaler의 fit_transform 함수를 활용하여 X_train 변수로 할당
  - 검증데이터셋의 Feature는 RobustScaler의 transform 함수를 활용하여 X_test 변수로 할당

---

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

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import RobustScaler

X_train, X_valid, y_train, y_valid = train_test_split(X, y, test_size=0.2, random_state=42)
print(X_train.shape, X_valid.shape, y_train.shape, y_valid.shape)

scaler = RobustScaler()
X_train = scaler.fit_transform(X_train)
X_valid = scaler.transform(X_valid)

print(X_train.shape, X_valid.shape)
```

```
(7995, 80) (1999, 80) (7995,) (1999,)
(7995, 80) (1999, 80)
```

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

## 의사결정나무(decision tree)와 랜덤포레스트(RandomForest)는 여러 가지 규칙을 순차적으로 적용하면서

## 독립 변수 공간을 분할하는 모형으로 분류(classification)와 회귀 분석(regression)에 모두 사용될 수 있습니다.

## 아래 가이드에 따라 의사결정나무(decision tree)와 랜덤포레스트(RandomForest) 모델 만들고 학습을 진행하세요.

---

- 의사결정나무(decision tree)
  - 트리의 최대 깊이 : 5로 설정
  - 노드를 분할하기 위한 최소한의 샘플 데이터수(min_samples_split) : 3로 설정
  - random_state : 120로 설정
  - 의사결정나무(decision tree) 모델을 dt 변수에 저장해 주세요.
- 랜덤포레스트(RandomForest)
  - 트리의 최대 깊이 : 5로 설정
  - 노드를 분할하기 위한 최소한의 샘플 데이터수(min_samples_split) : 3로 설정
  - random_state : 120로 설정
  - 랜덤포레스트(RandomForest) 모델을 rf 변수에 저장해 주세요.
- 위의 2개의 모델에 대해 fit을 활용해 모델을 학습해 주세요. 학습 시 훈련데이터 셋을 활용해 주세요.

---

```
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor

dt = DecisionTreeRegressor(max_depth=5, min_samples_split=3, random_state=120)
rf = RandomForestRegressor(max_depth=5, min_samples_split=3, random_state=120)

dt.fit(X_train, y_train)
rf.fit(X_train, y_train)
```

```
RandomForestRegressor(max_depth=5, min_samples_split=3, random_state=120)
```

## 12. 위 의사결정나무(decision tree)와 랜덤포레스트(RandomForest) 모델의 성능을 평가하려고 합니다.

## 아래 가이드에 따라 예측 결과의 mae(Mean Absolute Error)를 구하고 평가하세요.

---

- 성능 평가는 검증 데이터셋을 활용하세요.
- 11번 문제에서 만든 의사결정나무(decision tree) 모델로 y값을 예측(predict)하여 y_pred_dt에 저장하세요.
- 검증 정답(y_valid)과 예측값(y_pred_dt)의 mae(Mean Absolute Error)를 구하고 dt_mae 변수에 저장하세요.
- 11번 문제에서 만든 랜덤포레스트(RandomForest) 모델로 y값을 예측(predict)하여 y_pred_rf에 저장하세요.
- 검증 정답(y_valid)과 예측값(y_pred_rf)의 mae(Mean Absolute Error)를 구하고 rf_mae 변수에 저장하세요.
- 2개의 모델에 대한 mae 성능평가 결과을 확인하여 성능좋은 모델 이름을 '답안12' 변수에 저장하세요.
  - 예) 답안12 = 'decisiontree' 혹은 답안12 = 'randomforest'

---

```
from sklearn.metrics import mean_absolute_error

y_pred_dt = dt.predict(X_valid)
dt_mae = mean_absolute_error(y_pred_dt, y_valid)

y_pred_rf = rf.predict(X_valid)
rf_mae = mean_absolute_error(y_pred_rf, y_valid)

print(dt_mae, rf_mae)

답안12 = 'RandomForest'
```

```
108.66520360619405 66.49137400827676
```

### 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) 2개이상으로 모델을 구성하세요.
- dropout 비율 0.2로 Dropout 레이어 1개를 추가해 주세요.
- 손실함수는 MSE(Mean Squared Error)를 사용하세요.
- 하이퍼파라미터 epochs : 30, batch_size : 16 으로 설정해 주세요.
- 각 에포크마다 loss와 metrics 평가하기 위한 데이터로 x_valid, y_valid 사용하세요.
- 학습정보는 history 변수에 저장해 주세요.

```
model = Sequential()
print(X_train.shape)

model.add(Dense(128, activation='relu', input_shape=(80,)))
model.add(Dropout(0.2))
model.add(Dense(68, activation='relu'))
model.add(Dense(32, activation='relu'))
model.add(Dense(1))

model.compile(optimizer='adam', loss='mse', metrics='mse')

history = model.fit(X_train, y_train,
                    validation_data = (X_valid, y_valid),
                    epochs = 30, batch_size=16
                   )
```

### 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.

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