# 소득이 50000$ 를 초과하는지 아닌지 분류!

(uci adult.data 활용)

# 피처 소개

In [1]:
```python
# 인구 조사 자료를 바탕으로 소득이 $ 50,000 / 년을 초과하는지 예측합니다
import pandas as pd
import numpy as np
import seaborn as sns
names=['age','workclass', 'fnlwgt', 'education', 'education-num', 'marital-status', 'occupation', 'relationship', 'race', 'se
df_adult = pd.read_csv('adult.data.csv',names=names)
df_adult.head(5)
```

Out[1]:

| | age | workclass | fnlwgt | education | education-num | marital-status | occupation | relationship | race | sex | capital-gain | capital-loss | hours-per-week | native-country | target |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 39 | State-gov | 77516 | Bachelors | 13 | Never-married | Adm-clerical | Not-in-family | White | Male | 2174 | 0 | 40 | United-States | <=50K |
| 1 | 50 | Self-emp-not-inc | 83311 | Bachelors | 13 | Married-civ-spouse | Exec-managerial | Husband | White | Male | 0 | 0 | 13 | United-States | <=50K |
| 2 | 38 | Private | 215646 | HS-grad | 9 | Divorced | Handlers-cleaners | Not-in-family | White | Male | 0 | 0 | 40 | United-States | <=50K |
| 3 | 53 | Private | 234721 | 11th | 7 | Married-civ-spouse | Handlers-cleaners | Husband | Black | Male | 0 | 0 | 40 | United-States | <=50K |
| 4 | 28 | Private | 338409 | Bachelors | 13 | Married-civ-spouse | Prof-specialty | Wife | Black | Female | 0 | 0 | 40 | Cuba | <=50K |

- age: 나이
- workclass: 노동 계급
- fnlwgt: ??
- education: 교육
- education-num: 교육 수
- marital-status: 결혼 상태
- occupation: 직업
- relationship: 가족내에서 관계
- race: 인종
- sex: 성별
- capital-gain: 자본이득
- capital-loss: 자본손실
- hours-per-week: 주당 근로시간
- native-country: 출신 국가

In [2]:
```python
df_adult.shape  # 데이터 모양 파악
```

Out[2]: (32561, 15)

# 이상치 제거

In [3]:
```
df_adult.info() # 데이터 정보 확인
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 32561 entries, 0 to 32560
Data columns (total 15 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   age             32561 non-null  int64
 1   workclass       32561 non-null  object
 2   fnlwgt          32561 non-null  int64
 3   education       32561 non-null  object
 4   education-num   32561 non-null  int64
 5   marital-status  32561 non-null  object
 6   occupation      32561 non-null  object
 7   relationship    32561 non-null  object
 8   race            32561 non-null  object
 9   sex             32561 non-null  object
 10  capital-gain    32561 non-null  int64
 11  capital-loss    32561 non-null  int64
 12  hours-per-week  32561 non-null  int64
 13  native-country  32561 non-null  object
 14  target          32561 non-null  object
dtypes: int64(6), object(9)
memory usage: 3.7+ MB
```

In [5]:
```
# replace()로 데이터 값 중에 ? 값 => nan 으로 바꿔주기
df_adult.replace(' ?',np.nan, inplace=True)
df_adult.isna().sum()
```

Out[5]:
```
age                  0
workclass         1836
fnlwgt               0
education            0
education-num        0
marital-status       0
occupation        1843
relationship         0
race                 0
sex                  0
capital-gain         0
capital-loss         0
hours-per-week       0
native-country     583
target               0
dtype: int64
```

In [6]:
```
# dropna() 로 nan 값 없애주기
df_adult=df_adult.dropna(axis=0)
df_adult.isna().sum()
```

Out[6]:
```
age               0
workclass         0
fnlwgt            0
education         0
education-num     0
marital-status    0
occupation        0
relationship      0
race              0
sex               0
capital-gain      0
capital-loss      0
hours-per-week    0
native-country    0
target            0
dtype: int64
```

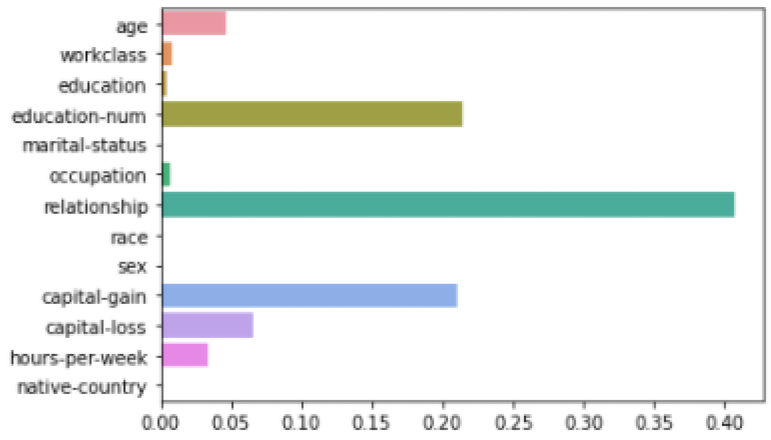
# 데이터 전처리

In [9]:
```python
# fnlwgt 컬럼은 필요없다고 판단하여 제거
# marital-status, relationship 별로 상관없는 거 같은데 영향력이 커서   제거하고 해보기로함.

df_adult.drop(['fnlwgt','marital-status','relationship'],axis=1,inplace=True)
df_adult.shape
```

Out[9]: (30162, 12)

```
native-country - 0.001
```

<AxesSubplot:>




In [10]:
```python
# 레이블 인코더 사용해서 문자열=> 숫자
from sklearn.preprocessing import LabelEncoder

col=names=['workclass','education','occupation',  'race', 'sex', 'native-country','target']
le=LabelEncoder()
for k in col:
    df_adult[k]=le.fit_transform(df_adult[k])


df_adult.head()
```

Out[10]:

| | age | workclass | education | education-num | occupation | race | sex | capital-gain | capital-loss | hours-per-week | native-country | target |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 39 | 5 | 9 | 13 | 0 | 4 | 1 | 2174 | 0 | 40 | 38 | 0 |
| 1 | 50 | 4 | 9 | 13 | 3 | 4 | 1 | 0 | 0 | 13 | 38 | 0 |
| 2 | 38 | 2 | 11 | 9 | 5 | 4 | 1 | 0 | 0 | 40 | 38 | 0 |
| 3 | 53 | 2 | 1 | 7 | 5 | 2 | 1 | 0 | 0 | 40 | 38 | 0 |
| 4 | 28 | 2 | 9 | 13 | 9 | 2 | 0 | 0 | 0 | 40 | 4 | 0 |

In [11]:
```python
# 데이터 비율 맞춰주기 위해  StandardScaler 적용
from sklearn.preprocessing import StandardScaler
Xdf=df_adult.loc[:,'age':'native-country']
ydf=df_adult.loc[:,'target']

scaler = StandardScaler()
scaler.fit(Xdf)
adult_scaled = scaler.transform(Xdf)
```

# 평가지표

In [16]:
```
from sklearn.metrics import confusion_matrix, accuracy_score
from sklearn.metrics import precision_score, recall_score
from sklearn.metrics import f1_score, roc_auc_score

# 배웠던 평가지표를 구현하는 함수
def get_clf_eval(y_test , pred):
    confusion = confusion_matrix( y_test, pred)
    accuracy = accuracy_score(y_test , pred)
    precision = precision_score(y_test , pred)
    recall = recall_score(y_test , pred)
    f1 = f1_score(y_test,pred)
    roc_auc = roc_auc_score(y_test, pred)
    print('오차 행렬')
    print(confusion)
    print('정확도: {0:.4f}, 정밀도: {1:.4f}, 재현율: {2:.4f},\
    F1: {3:.4f}, AUC:{4:.4f}'.format(accuracy, precision, recall, f1, roc_auc))
```

# 결정 트리 분류

In [20]:

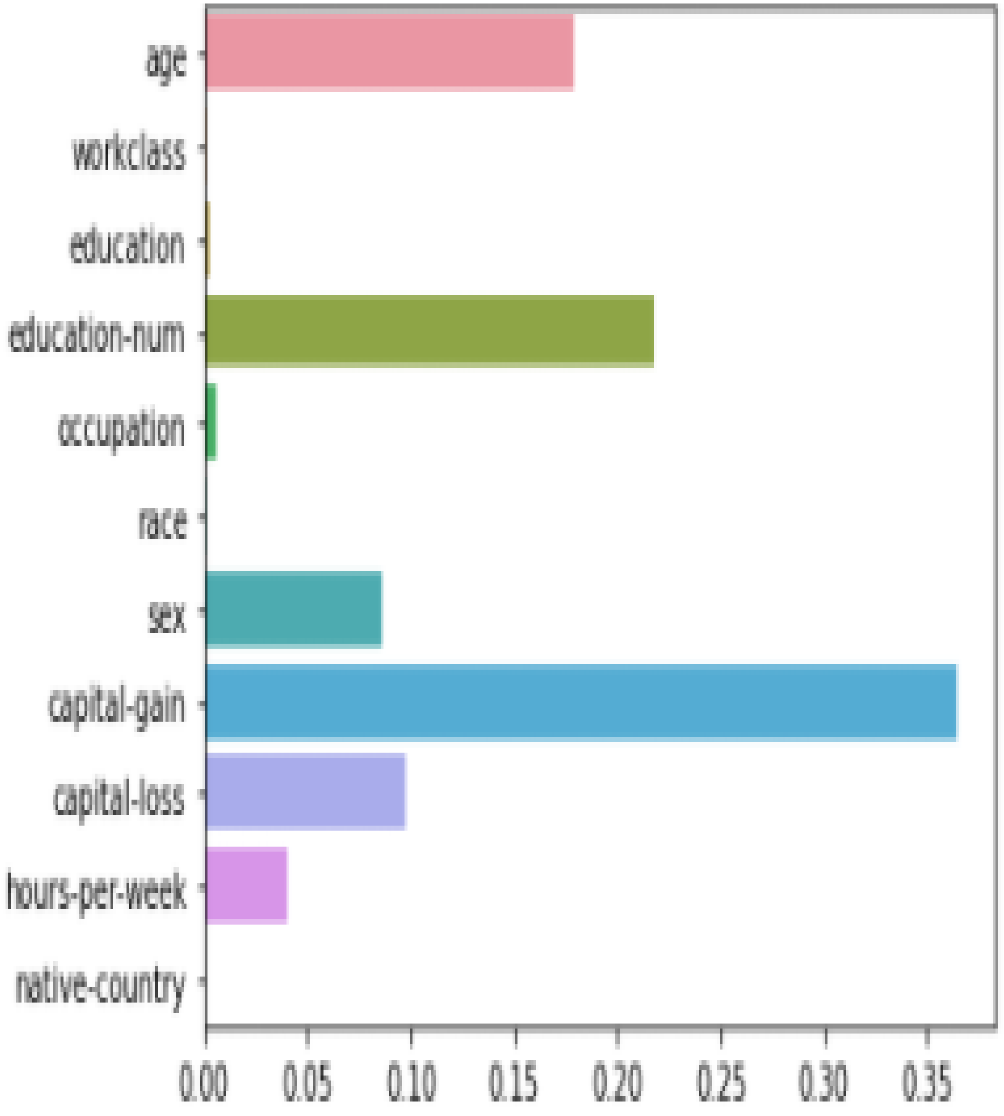
```
#결정트리분류기
from sklearn.tree import DecisionTreeClassifier

clf=DecisionTreeClassifier(max_depth=8)
clf.fit(X_train,y_train)

print("훈련 세트 정확도: {:.3f}".format(clf.score(X_train, y_train)))
print("테스트 세트 정확도: {:.3f}".format(clf.score(X_test, y_test)))
print()
pred=clf.predict(X_test)
get_clf_eval(y_test , pred)
```

```
훈련 세트 정확도: 0.845
테스트 세트 정확도: 0.831

오차 행렬
[[5412  227]
 [1048  854]]
정확도: 0.8309, 정밀도: 0.7900, 재현율: 0.4490,    F1: 0.5726, AUC:0.7044
```

In [21]:

```
import seaborn as sns
import numpy as np
%matplotlib inline
# feature importance 추출
print("Feature importances:₩n{0}".format(np.round(clf.feature_importances_,
3)))
# feature별 importance 매핑
for name, value in zip(names , clf.feature_importances_):
    print('{0} : {1:.3f}'.format(name, value))
# feature importance를 column 별로 시각화 하기
sns.barplot(x=clf.feature_importances_ , y=names)
```

```
Feature importances:
[0.178 0.002 0.001 0.219 0.007 0.002 0.087 0.365 0.098 0.041 0.001]
age : 0.178
workclass : 0.002
education : 0.001
education-num : 0.219
occupation : 0.007
race : 0.002
sex : 0.087
capital-gain : 0.365
capital-loss : 0.098
hours-per-week : 0.041
native-country : 0.001
```



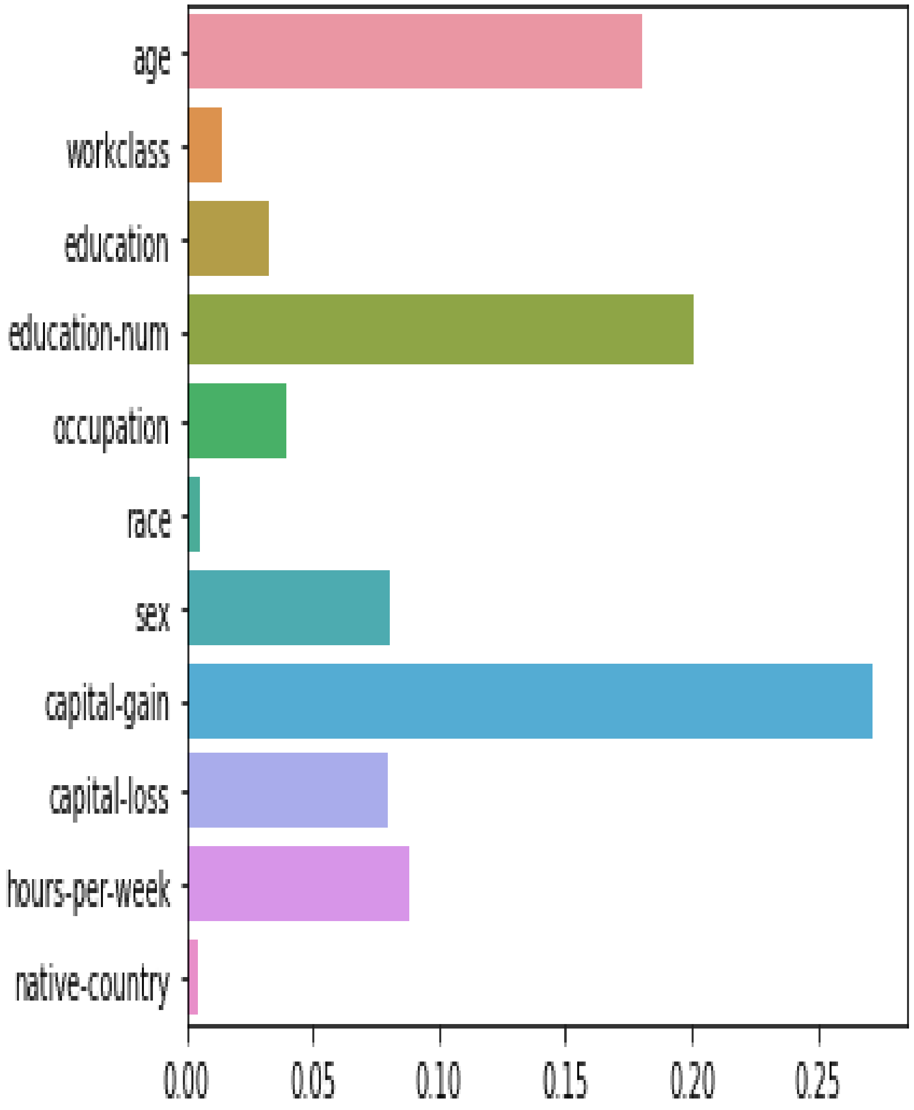
# 랜덤 포레스트

```
In [41]: #랜덤포레스트
from sklearn.ensemble import RandomForestClassifier

clf=RandomForestClassifier(max_depth=10, min_samples_leaf=4, min_samples_split=10, n_estimators=300)
clf.fit(X_train,y_train)

print("훈련 세트 정확도: {:.3f}".format(clf.score(X_train, y_train)))
print("테스트 세트 정확도: {:.3f}".format(clf.score(X_test, y_test)))
print()
pred=clf.predict(X_test)
get_clf_eval(y_test , pred)
```

```
훈련 세트 정확도: 0.849
테스트 세트 정확도: 0.836

오차 행렬
[[5394  245]
 [ 995  907]]
정확도: 0.8356, 정밀도: 0.7873, 재현율: 0.4769,    F1: 0.5940, AUC:0.7167
```

```
In [42]: import seaborn as sns
import numpy as np
%matplotlib inline
# feature importance 추출
print("Feature importances:\n{0}".format(np.round(clf.feature_importances_,
3)))
# feature별 importance 매핑
for name, value in zip(names , clf.feature_importances_):
    print('{0} : {1:.3f}'.format(name, value))
# feature importance를 column 별로 시각화 하기
sns.barplot(x=clf.feature_importances_ , y=names)
```

```
Feature importances:
[0.181 0.014 0.033 0.201 0.04  0.006 0.081 0.272 0.08  0.089 0.004]
age : 0.181
workclass : 0.014
education : 0.033
education-num : 0.201
occupation : 0.040
race : 0.006
sex : 0.081
capital-gain : 0.272
capital-loss : 0.080
hours-per-week : 0.089
native-country : 0.004
```



# 딥러닝

```python
# 딥러닝코드

# 딥러닝을 구동하는 데 필요한 케라스 함수를 불러옵니다.
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

# 필요한 라이브러리를 불러옵니다.
import numpy as np
import tensorflow as tf

# 실행할 때마다 같은 결과를 출력하기 위해 설정하는 부분입니다.
np.random.seed(0)
tf.random.set_seed(0)
```

```python
from keras.callbacks import EarlyStopping

# 딥러닝 구조를 결정합니다(모델을 설정하고 실행하는 부분입니다).
model = Sequential()
model.add(Dense(30, input_dim=11, activation='relu'))
model.add(Dense(12,  activation='relu'))
model.add(Dense(8,  activation='relu'))
model.add(Dense(1, activation='sigmoid'))

# 딥러닝을 실행합니다.
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

early_stopping_callback=EarlyStopping(monitor='val_loss',patience=10)

model.fit(X_train, y_train, epochs=1000, batch_size=100)
```

```
Epoch 992/1000
227/227 [==============================] - 0s 977us/step - loss: 0.3256 - accuracy: 0.8497
Epoch 993/1000
227/227 [==============================] - 0s 749us/step - loss: 0.3265 - accuracy: 0.8503
Epoch 994/1000
227/227 [==============================] - 0s 810us/step - loss: 0.3262 - accuracy: 0.8497
Epoch 995/1000
227/227 [==============================] - 0s 762us/step - loss: 0.3260 - accuracy: 0.8503
Epoch 996/1000
227/227 [==============================] - 0s 872us/step - loss: 0.3261 - accuracy: 0.8501
Epoch 997/1000
227/227 [==============================] - 0s 806us/step - loss: 0.3260 - accuracy: 0.8497
Epoch 998/1000
227/227 [==============================] - 0s 830us/step - loss: 0.3259 - accuracy: 0.8504
Epoch 999/1000
227/227 [==============================] - 0s 828us/step - loss: 0.3258 - accuracy: 0.8510
Epoch 1000/1000
227/227 [==============================] - 0s 819us/step - loss: 0.3260 - accuracy: 0.8504
```

| | 모델 | 정확도 | 정밀도 | 재현율 | F1 score | AUC |
|---|---|---|---|---|---|---|
| 0 | 딥러닝 | 0.8504 | NaN | NaN | NaN | NaN |
| 1 | LGBM | 0.8488 | 0.7737 | 0.5662 | 0.6539 | 0.7552 |
| 2 | GBM | 0.8472 | 0.7774 | 0.5526 | 0.6460 | 0.7496 |
| 3 | XGB | 0.8455 | 0.7744 | 0.5468 | 0.6410 | 0.7465 |
| 4 | ada부스트 | 0.8373 | 0.7582 | 0.5210 | 0.6176 | 0.7325 |
| 5 | 랜덤포레스트 | 0.8356 | 0.7873 | 0.4769 | 0.5940 | 0.7167 |
| 6 | 결정트리 | 0.8309 | 0.7900 | 0.4490 | 0.5726 | 0.7044 |
| 7 | svm | 0.8211 | 0.7530 | 0.4327 | 0.5496 | 0.6924 |
| 8 | 로지스틱회귀 | 0.8118 | 0.7102 | 0.4290 | 0.5349 | 0.6850 |
| 9 | knn | 0.8088 | 0.6686 | 0.4795 | 0.5585 | 0.6997 |