



# 인구 데이터 기반 소득 예측

가이드 프로젝트 C조

# Contents

**1. EDA**

**2. ML**

**3. 의의 및 한계**

# **1. EDA**

# 1-1. fnlwgt 변수

- 인구 조사 데이터에서 전체 인구로 일반화할 때 사용되는  
가중치 변수
- 모델링에서 가중치를 사용할 계획 X

→ drop 결정

## 1-2. Relationship, Martial.status 파생 변수 생성

relationship	count
Husband	10017
Not-in-family	54
Own-child	2666
Unmarried	1242
Wife	777
Other-relative	777

Relationship  
변수로 예측가능

카테고리의 모호성

marital.status	count
Married-civ-spouse	11421
Never-married	8242
Divorced	3429
Separated	793
Widowed	788
Married-spouse-absent	330
Married-AF-spouse	17

# 1-2. Relationship, Martial.status 파생 변수 생성

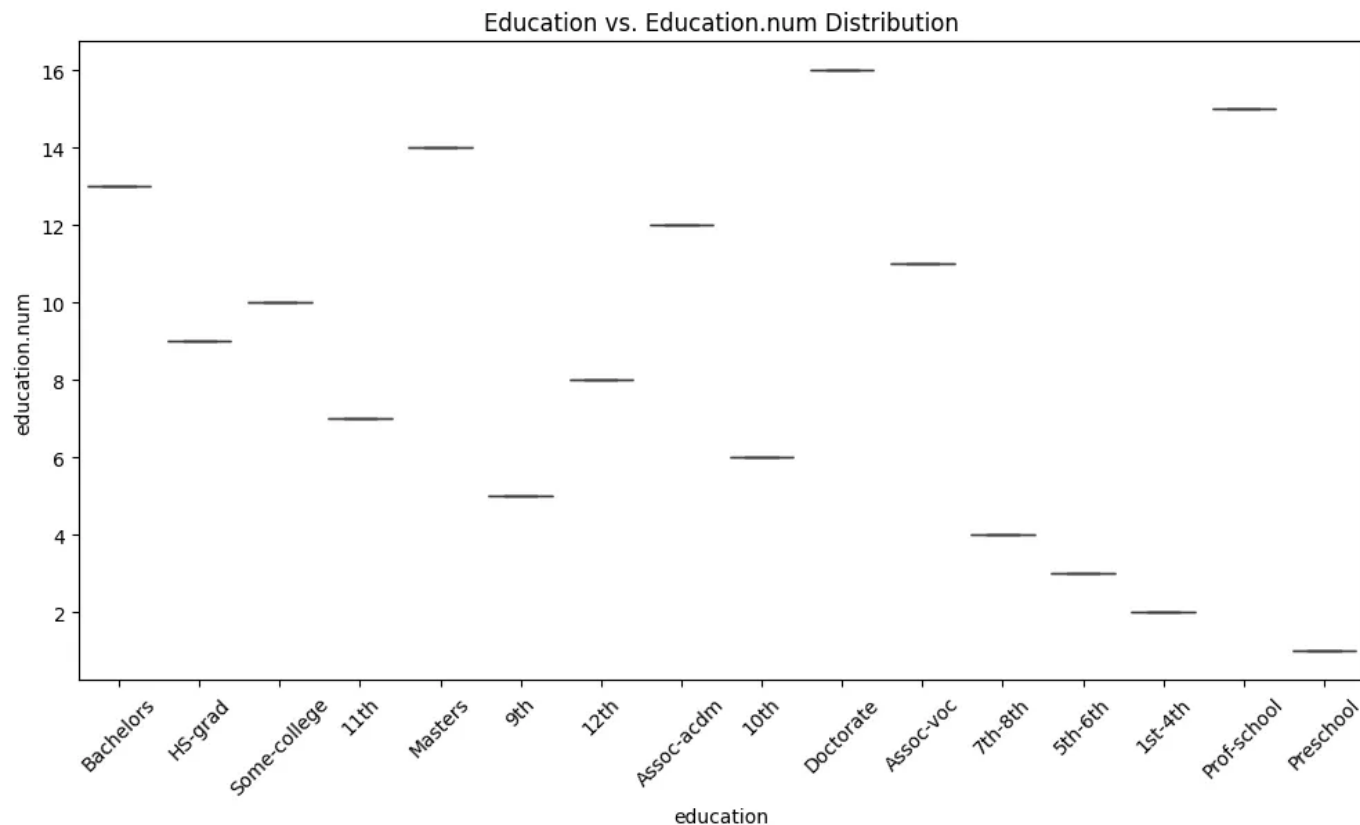
**20** What is this person's marital status?

- ☐ Now married
- ☐ Widowed
- ☐ Divorced
- ☐ Separated
- ☐ Never married → SKIP to **1**

출처: Census 인구통계조사

파생 변수의 범주	기존 데이터셋의 범주
Now Married	Married-civ-spouse, Married-Af-spouse
Widowed	Widowed
Divorced	Divorced
Seperated	Seperated, Married-spouse-absent
Never married	Never-married
Spouse(No Kid)	Husband, Wife
Spouse(Yes Kid)	Own-Child
Unmarried	Unmarried, Not-in-family
Other relative	Other relative

# 1-3. Education.num, Education 변수



**Education.num은 Education  
을 숫자로 범주화한 변수**

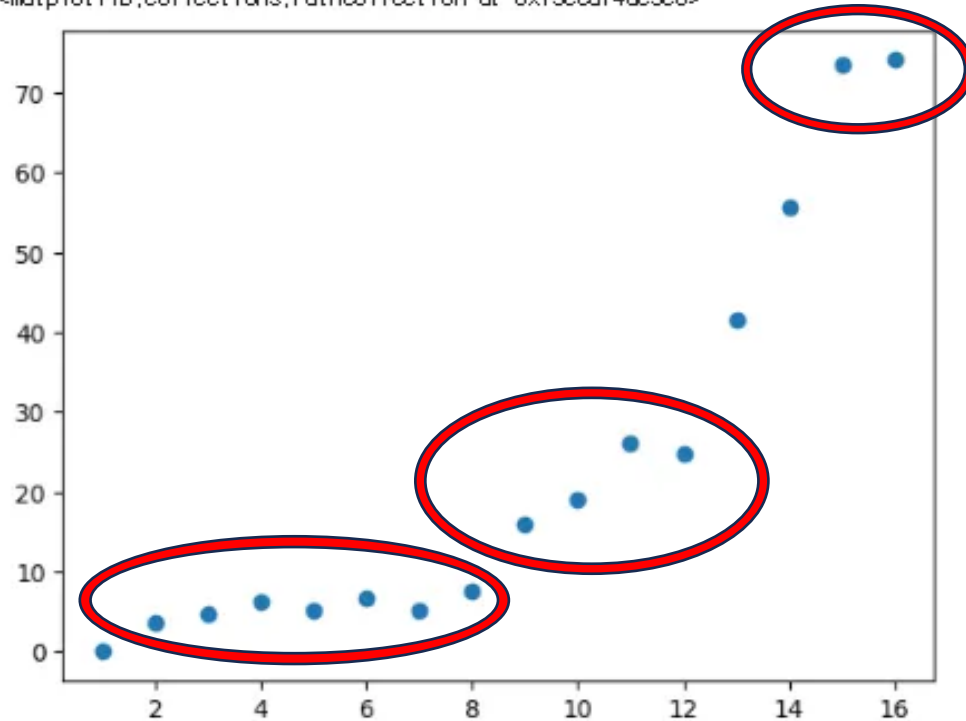
- 교육 수준이 높을수록 개인의 교육 수준이 높음을 의미
- **Eudcation 변수 drop 결정**

NEW!!

## 1-3. Education.num 변수 단순화

```
plt.scatter(income_proportion, index, income_proportion['yes_income_ratio'])
```

<matplotlib.collections.PathCollection at 0x79eeaf4de5c0>



Education.num별 income0이 1인 비율

- 비슷한 값을 가지는 범주 확인
- 재범주화 진행  
(1~8, 9~12, 13, 14, 15~16)



NEW!!

## 1-3. Education.num 변수 단순화

education.num income

education.num 1.000000 0.334147

income 0.334147 1.000000

Education.num별 income이 1인 비율

- 범주화 후 상관계수 증가

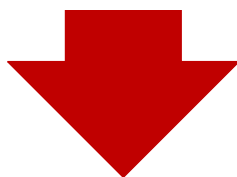
```
[26] whole_data[['education_num_prep', 'income']].corr()
```



education\_num\_prep income

education\_num\_prep 1.000000 0.359988

income 0.359988 1.000000

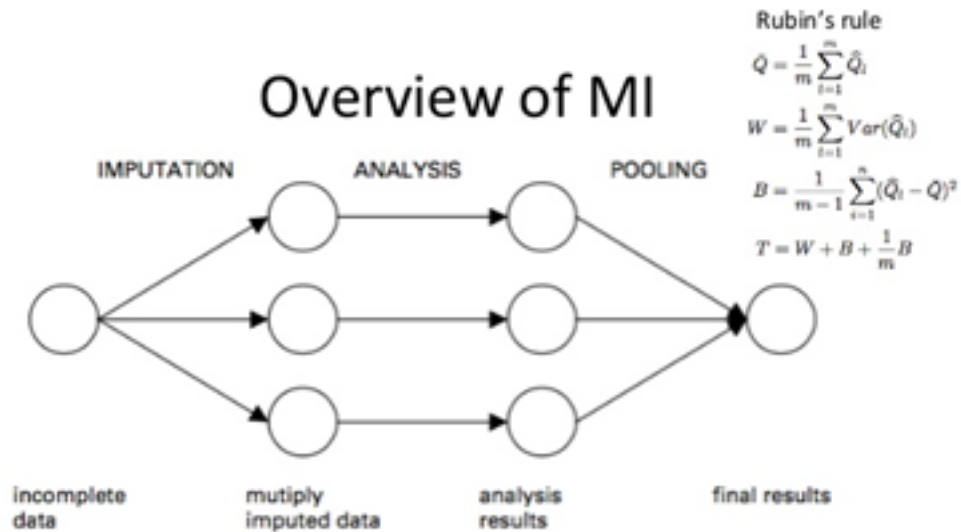


# 2-1 결측치 처리



# 2-2 workclass - occupation 결측치 처리

## 대중대치법활용



- 목적: 데이터셋의 결측치를 여러 번 대체하여 다양한 대체 데이터 생성
- 방법: 각 대체 데이터로 분석 후 결과를 결합해 결측치로 인한 불확실성 반영 (연쇄방정식 활용)

## 2-2 다중대치법

- MAR 전제:

데이터 자료의 결측이 그 문항 자체 때문이 아님.

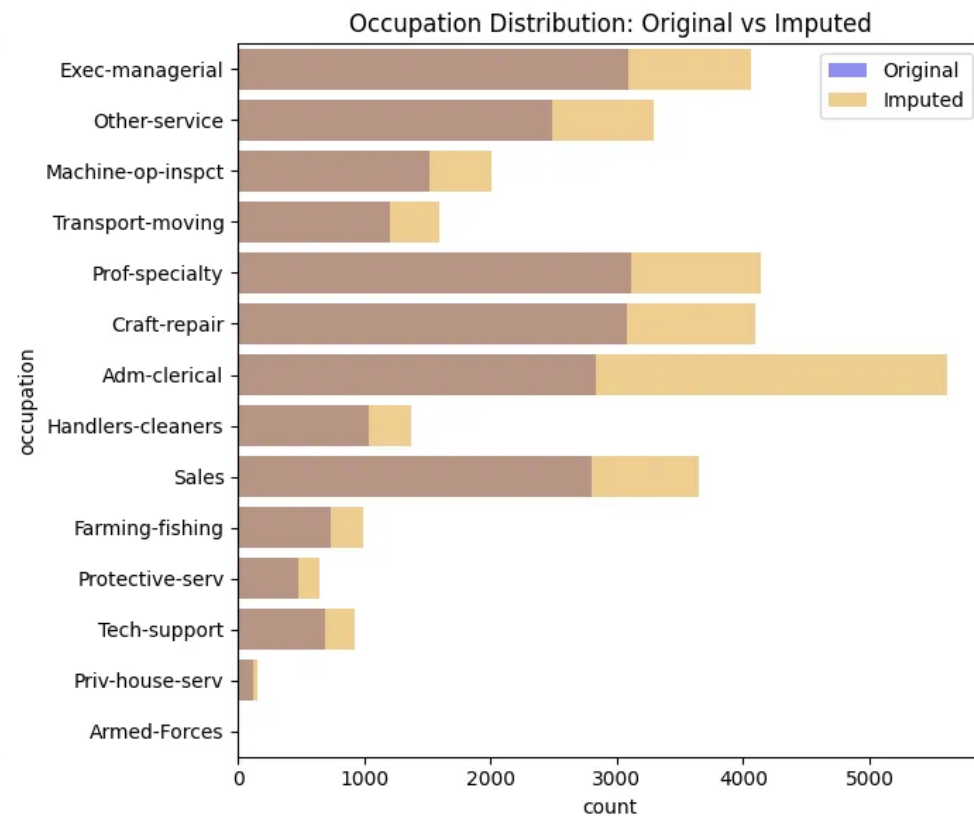
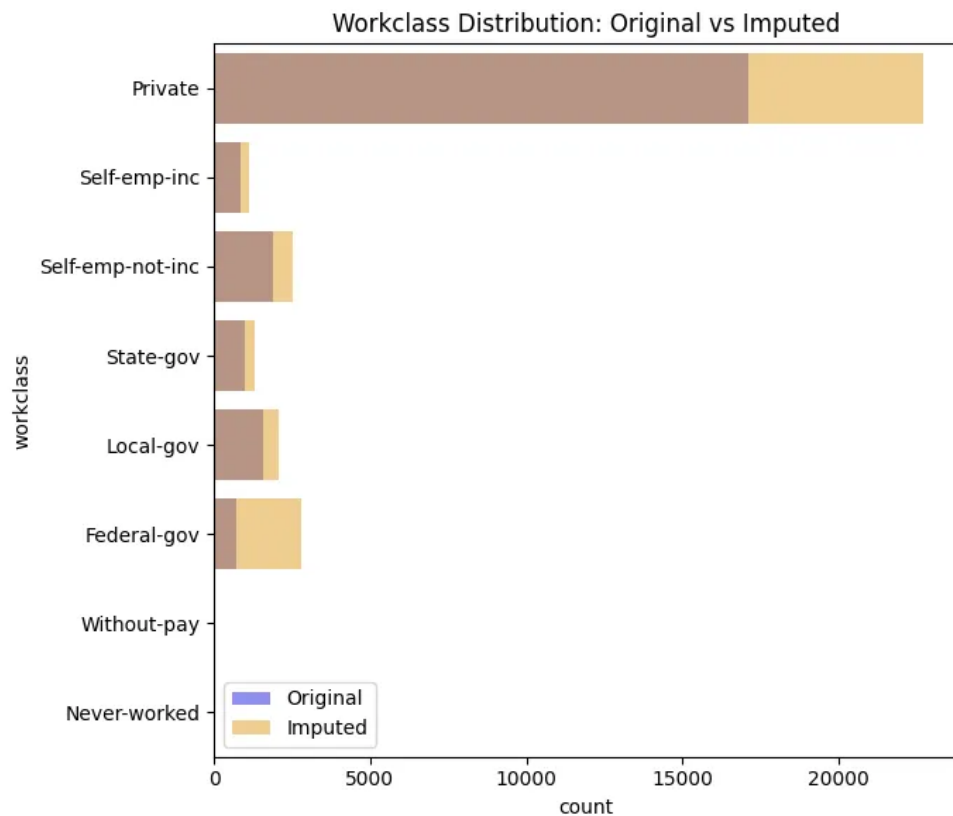
따라서 다른 변수들을 통해 그 결측 여부와 생략된 응답을 추정할 수 있음

- 대치모델 형식:

workclass, occupation은 범주형 변수이므로 범주형 형식 사용

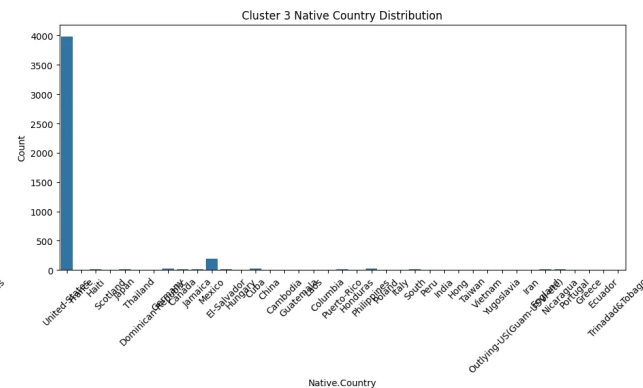
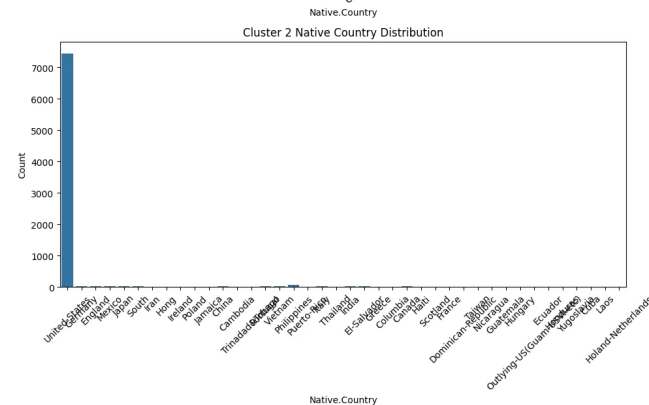
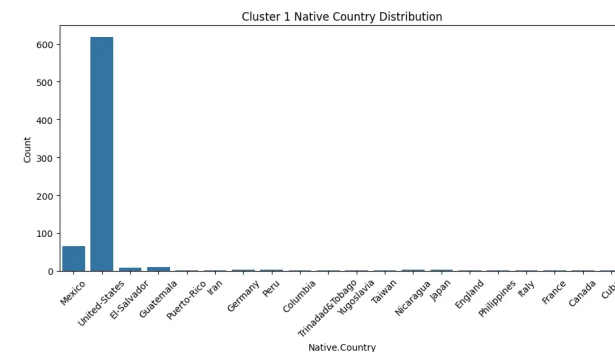
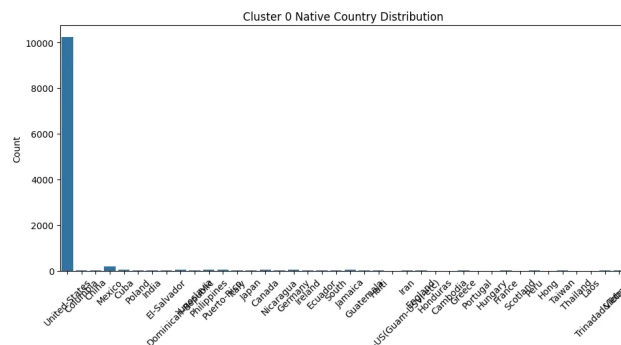
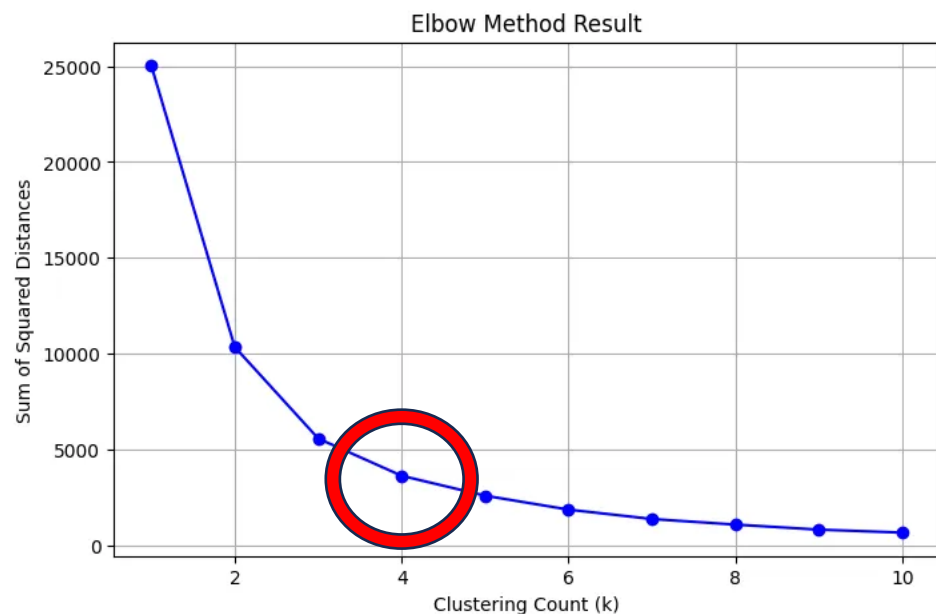
- 대체 과정에서의 iteration 횟수: 10

## 2-2 다중대치법 결과



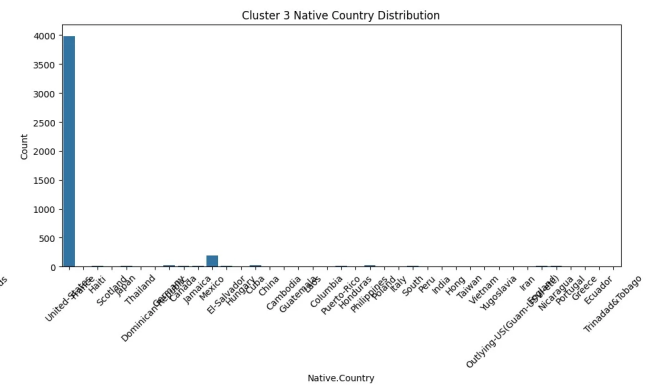
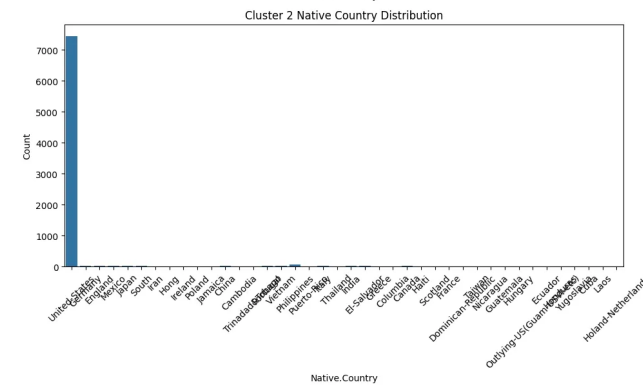
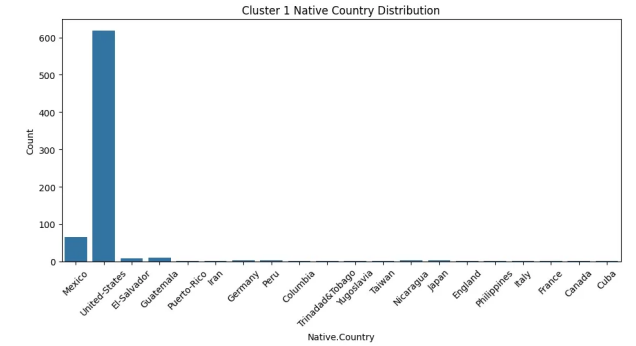
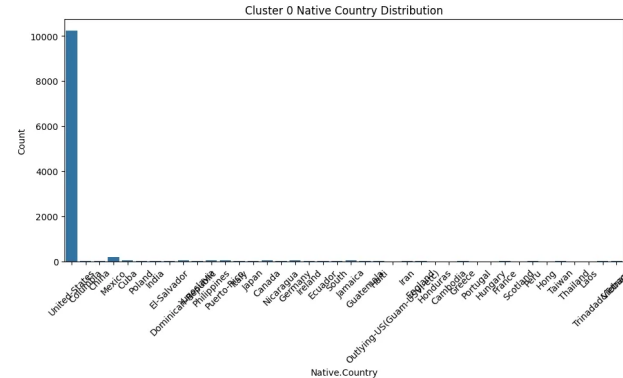
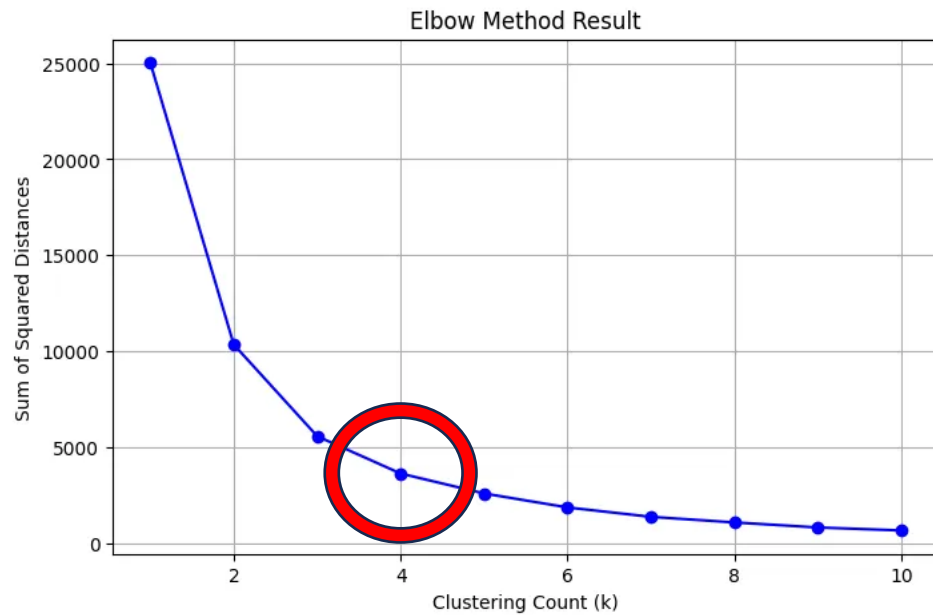
그래프. 결측치 처리 결과

# 2-4 native.country 결측치 처리



Fnlwgt 변수 활용하여 클러스터링 진행 → 대실패

# 2-4 native.country 결측치 처리



Fnlwgt 변수 활용하여 클러스터링 진행 → 대실패

NEW!!

## 2-4 native.country 결측치 처리

```
[39] # Native.country - race 상관관계 높음
import pandas as pd
import numpy as np
import scipy.stats as stats

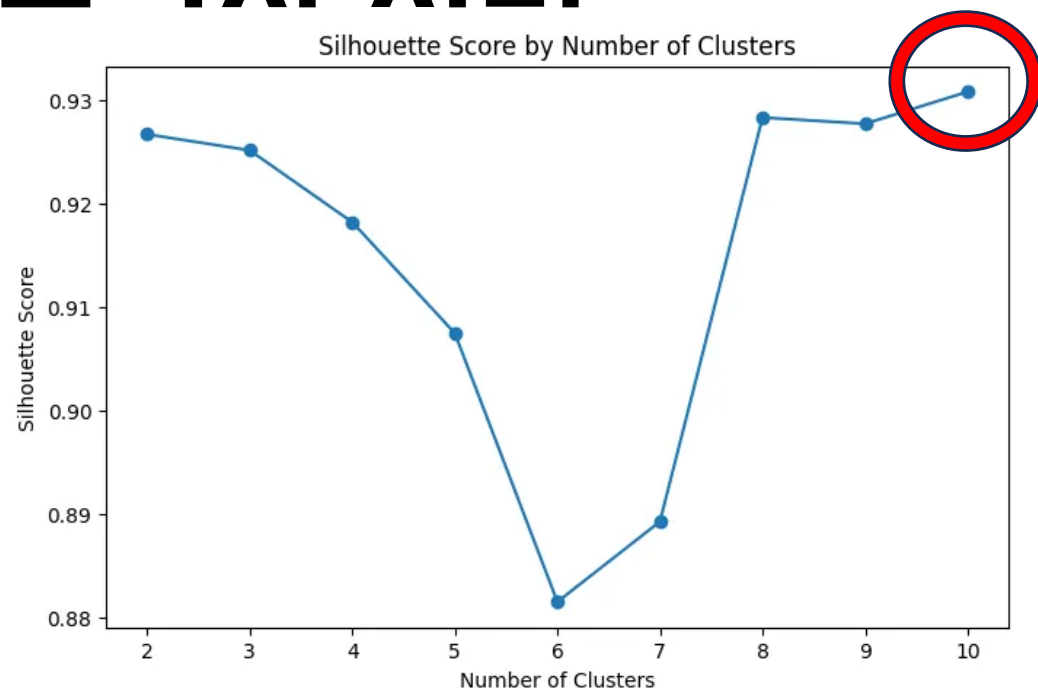
# Cramer's V 계산 함수
def cramers_v(x, y):
    confusion_matrix = pd.crosstab(x, y)
    chi2 = stats.chi2_contingency(confusion_matrix)[0]
    n = confusion_matrix.sum().sum()
    return np.sqrt(chi2 / (n * (min(confusion_matrix.shape) - 1)))

# native.country와 race 간의 상관관계 계산
cramers_v_value = cramers_v(total['native.country'], total['race'])
print(f"Cramer's V between native.country and race: {cramers_v_value:.2f}")
```

→ Cramer's V between native.country and race: 0.42

- Native.country와 race 상관관계 높음
- Country, race 클러스터링 활용하여 대푯값으로 채우기

→ 모든 결측치가 US로 채워져 대실패: 결측치 Unknown 처리

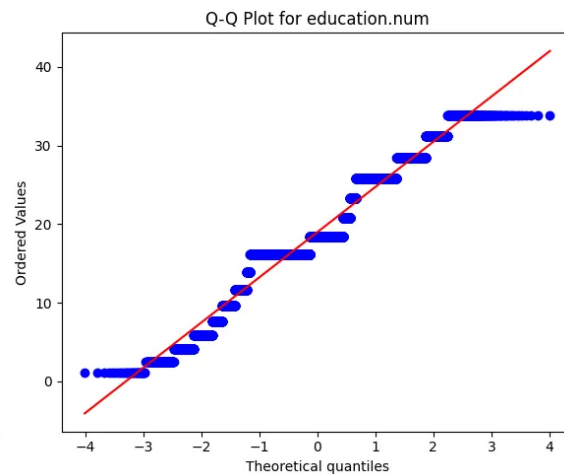
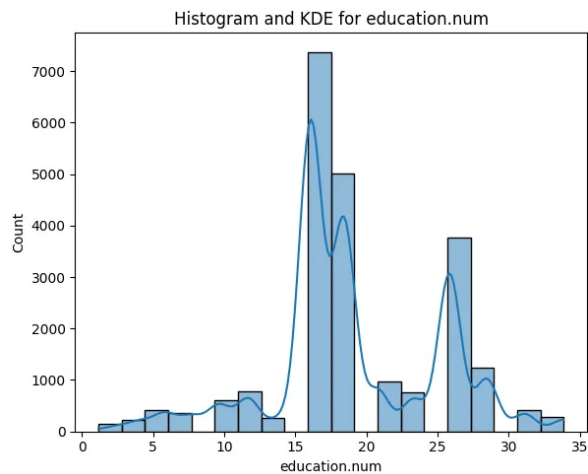
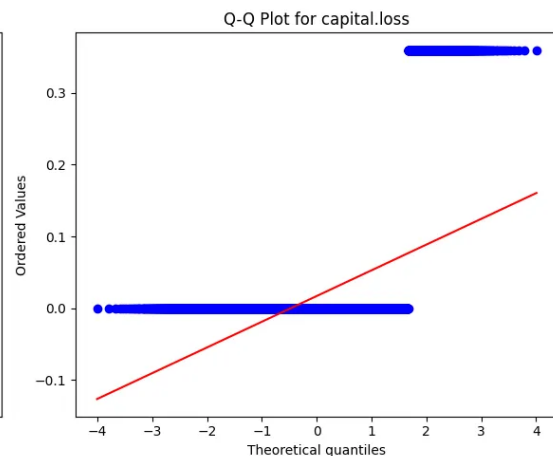
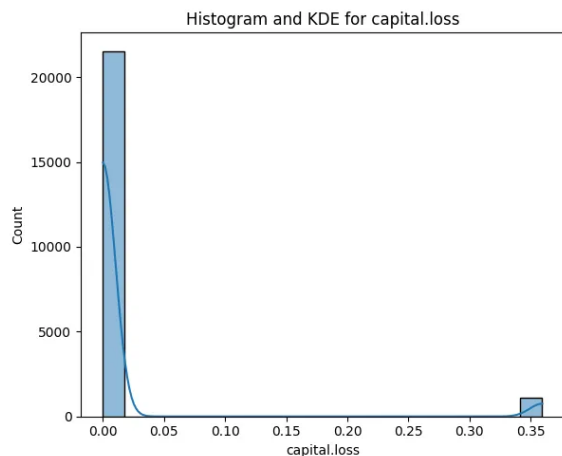




## 3-1. 범주형 변수 이상치 제거

- 범주형 변수별로 unique 값 → 각각의 비율을 계산
- 전체의 1% 미만인 경우: others로 묶어 처리

# 3-1. 수치형 변수 이상치 제거



1. 수치형 변수 정규분포 따르지 않아  
box-cox 변환 실행
  2. Kolmogorov-Smirnov 검정,  
Anderson-Darling 검정:  
정규분포 따르지 않음
- isolation forest 사용 결정

# 3-1. 수치형 변수 이상치 제거

총 데이터 샘플 수: 22621

이상치로 탐지된 데이터 수: 2417

- 이상치 발견 후 제거

- 성능이 좋지 못함!

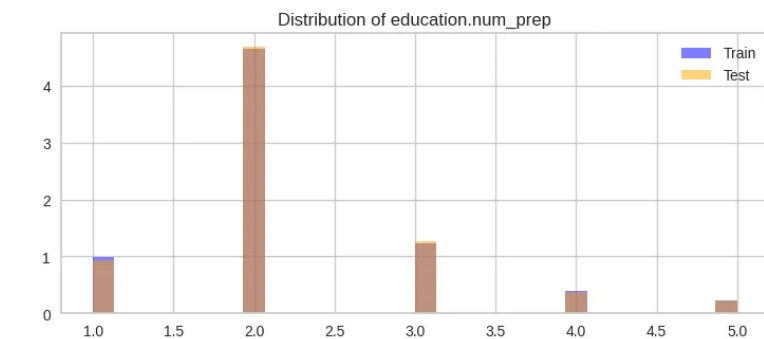
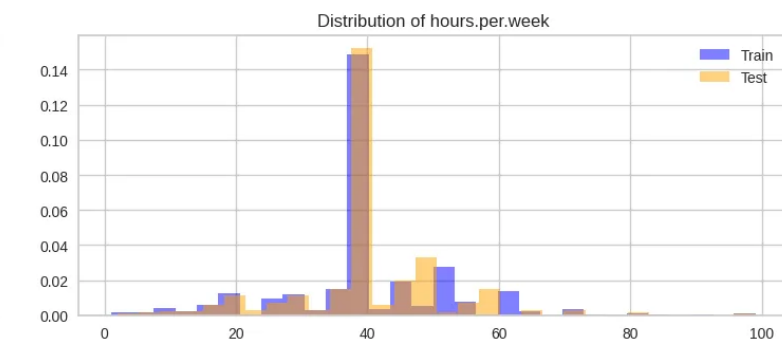
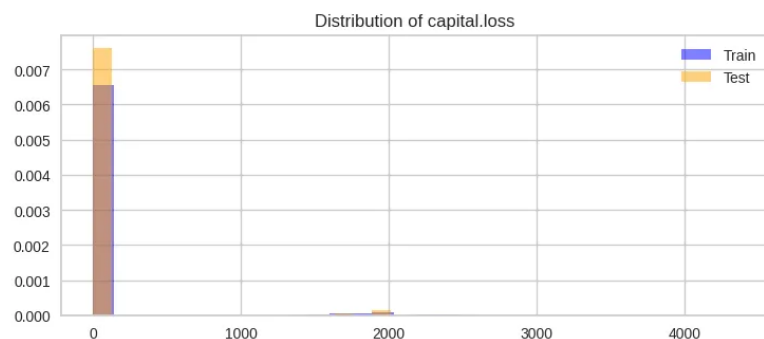
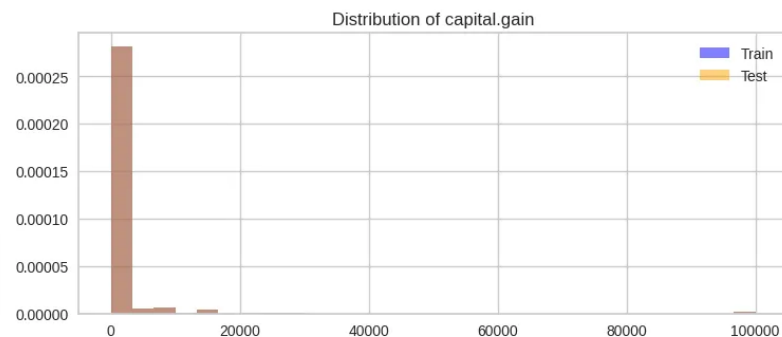
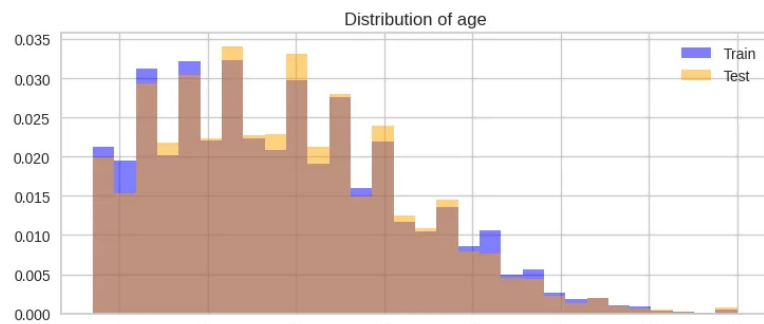
→ TEST 데이터의 문제가 있는 것 아닌가?

## 3-2. TEST 데이터 확인

<pre># IQR 벗어난 train 데이터 비율 계산 및 출력 outlier_percentages = calculate_outlier_percentage(train_df, numerical_cols) for col, pct in outlier_percentages.items():     print(f"{col}: {pct:.2f}% 이상의 데이터가 IQR을 벗어났습니다.")</pre>	
<pre>age: 0.34% 이상의 데이터가 IQR을 벗어났습니다. capital.gain: 8.28% 이상의 데이터가 IQR을 벗어났습니다. capital.loss: 4.69% 이상의 데이터가 IQR을 벗어났습니다. hours.per.week: 28.11% 이상의 데이터가 IQR을 벗어났습니다. education.num_prep: 37.95% 이상의 데이터가 IQR을 벗어났습니다.</pre>	
<pre>[ ] # IQR 벗어난 test 데이터 비율 계산 및 출력 outlier_percentages = calculate_outlier_percentage(test_df, numerical_cols) for col, pct in outlier_percentages.items():     print(f"{col}: {pct:.2f}% 이상의 데이터가 IQR을 벗어났습니다.")</pre>	
<pre>age: 0.74% 이상의 데이터가 IQR을 벗어났습니다. capital.gain: 8.49% 이상의 데이터가 IQR을 벗어났습니다. capital.loss: 4.57% 이상의 데이터가 IQR을 벗어났습니다. hours.per.week: 26.19% 이상의 데이터가 IQR을 벗어났습니다. education.num_prep: 37.48% 이상의 데이터가 IQR을 벗어났습니다.</pre>	

**Train, test 모두 iqr 방법으로 이상치 탐지하여 비교**  
**각 변수별 이상치 비율이 유사**

## 3-2. TEST 데이터 확인




범주형 변수 시각화:  
train-test 분포가  
유사

Train, Test 이상치 데이터 전부 중앙값으로 대체

## **3-3. 라벨인코딩**

- **라벨인코딩 방식 활용하여 인코딩**

## 3-4. 다중공선성




	Feature	VIF
0	age	1.091112
1	capital.gain	1.010883
2	capital.loss	1.001590
3	hours.per.week	1.091638
4	workclass	1.065723
5	marital.status	1.280312
6	occupation	1.053512
7	relationship	1.395621
8	race	1.114644
9	sex	1.243705
10	native.country	1.112406
11	education.num_prep	1.028245

다중공선성 나타나지 않음

**2. ML**



# 1. Pycaret으로 최적의 모델 선정



	Model	Accuracy
lightgbm	Light Gradient Boosting Machine	0.8547
xgboost	Extreme Gradient Boosting	0.8506
gbc	Gradient Boosting Classifier	0.8485
ada	Ada Boost Classifier	0.8413
rf	Random Forest Classifier	0.8332
et	Extra Trees Classifier	0.8252
knn	K Neighbors Classifier	0.8194
dt	Decision Tree Classifier	0.8030
ridge	Ridge Classifier	0.7902
lr	Logistic Regression	0.7878
lda	Linear Discriminant Analysis	0.7877
nb	Naive Bayes	0.7871
svm	SVM – Linear Kernel	0.7835
qda	Quadratic Discriminant Analysis	0.7797
dummy	Dummy Classifier	0.7625

- lightbgm, xgboost, gradient boosting 모델로 사용

## 2. 하이퍼파라미터 튜닝

- 랜덱서치 활용 lightgbm, xgboost, gradient boosting 최적의 하이퍼파라미터 탐색

```
# LightGBM 하이퍼파라미터 탐색 공간 정의
lgbm_param_dist = {
    'num_leaves': np.arange(20, 150, 10),
    'max_depth': np.arange(3, 16),
    'learning_rate': [0.01, 0.05, 0.1, 0.2],
    'n_estimators': np.arange(50, 300, 50),
    'min_child_samples': np.arange(10, 100, 10),
    'subsample': [0.6, 0.7, 0.8, 0.9, 1.0],
    'colsample_bytree': [0.6, 0.7, 0.8, 0.9, 1.0]
}
```

```
# LightGBM 모델 객체 생성
lgbm = lgb.LGBMClassifier(random_state=42)
```

```
# RandomizedSearchCV 설정 및 실행
lgbm_search = RandomizedSearchCV(
    estimator=lgbm,
    param_distributions=lgbm_param_dist,
    n_iter=100,
    scoring='accuracy',
    cv=3,
    random_state=42,
    n_jobs=-1,
    verbose=1
)
```

```
# LightGBM 모델 튜닝
lgbm_search.fit(X_train, y_train)
```

```
# 최적 모델 추출
best_lgbm = lgbm_search.best_estimator_
print("Best LightGBM Parameters:", lgbm_search.best_params_)
```

# 3. 스택킹 앙상블

```
# 7. 스택킹 모델 구성 및 학습
print("\nConstructing and training the Stacking Classifier...")

# 메타 모델 정의 (로지스틱 회귀)
meta_model = LogisticRegression(random_state=42, max_iter=1000)

# 스택킹 모델 정의
stacking_clf = StackingClassifier(
    estimators=[
        ('lightgbm', best_lgbm),
        ('xgboost', best_xgboost),
        ('gbc', best_gbc)
    ],
    final_estimator=meta_model,
    cv=3,
    n_jobs=-1
)

# 스택킹 모델 학습
stacking_clf.fit(X_train, y_train)
print("Stacking Classifier training completed.")
```

- 스택킹 앙상블 기법 활용하여 세 개의 모델 모두 사용

## 〈스택킹 앙상블〉

1. 여러 개의 모델이 각각 예측을 수행
2. 예측된 결과들을 새로운 메타 모델이 학습하여 최종 예측 수행

## 4. 결과

```
# 정확도 출력
```

```
stacking_accuracy = accuracy_score(y_test, stacking_pred)  
print(f"Stacking Model Accuracy: {stacking_accuracy:.4f}")
```

Stacking Model Accuracy 0.8332

### **3. 의의 및 한계**

**감사합니다**