실습

C 조

목차

실습에 사용된 데이터

Kaggle Dataset [Customer Segmentation Classification]

https://www.kaggle.com/kaushiksuresh147/customer-segmentation

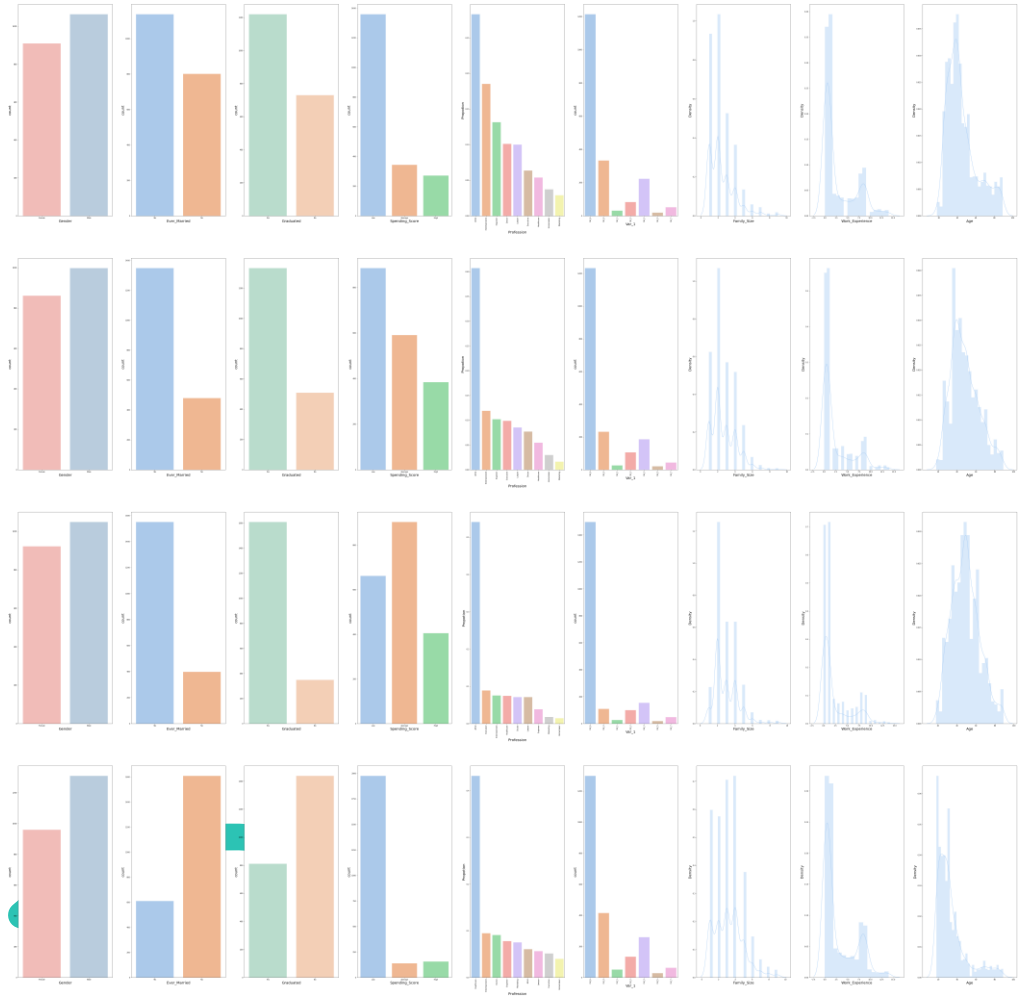
한 자동차 회사에서 소비자의 특성을 담은 데이터를 기반으로 고객을 4개의 부분(A,B,C,D)으로 분류하는 과제

변수 소개

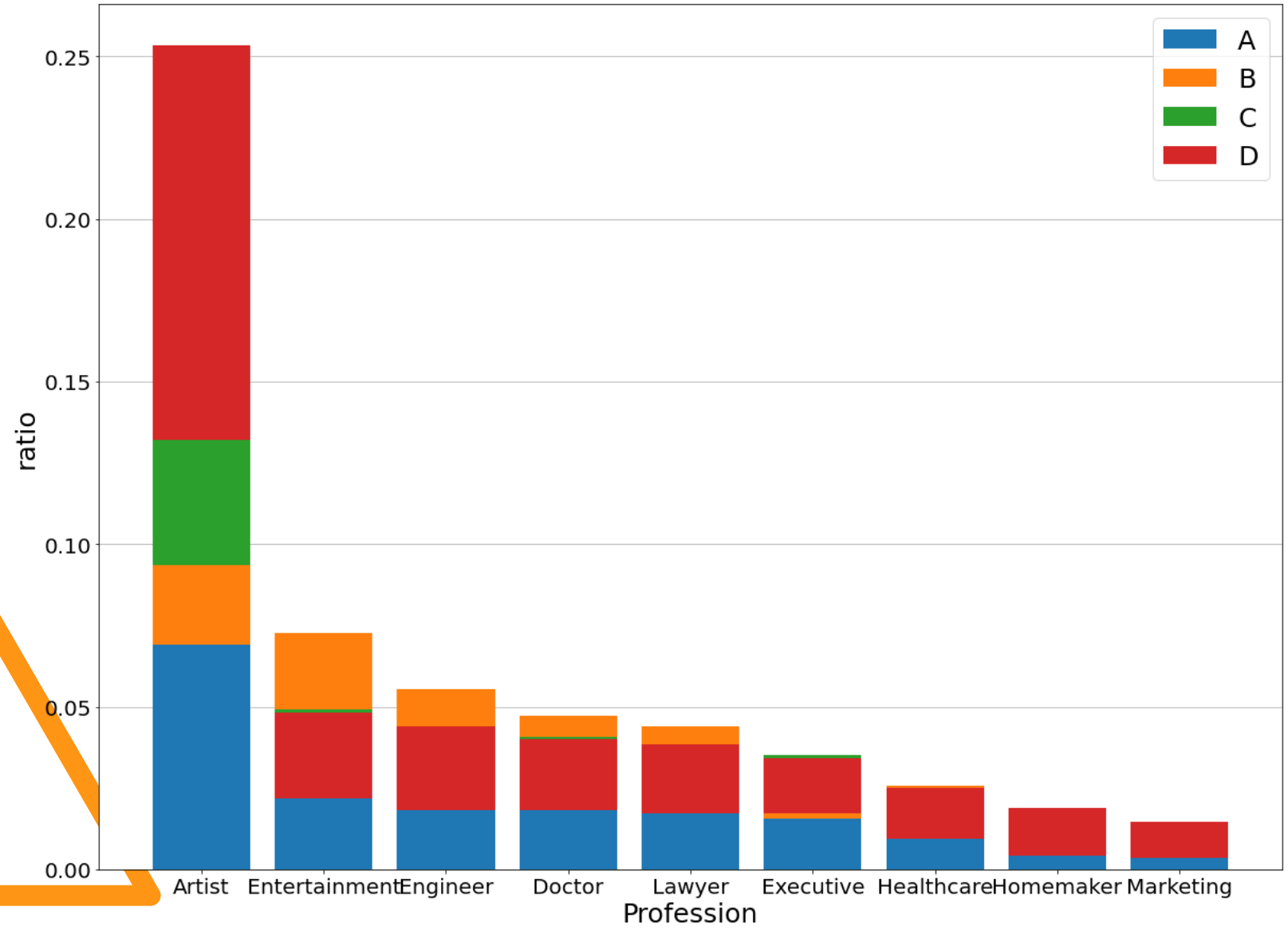
Variable	Definition
ID	개인 식별 번호
Gender	고객의 성별
Ever_Married	고객의 결혼 여부
Age	고객의 나이
Graduated	고객의 졸업 여부
Profession	고객의 직업
Work_Experience	고객의 경력(연 단위)
Spending_Score	고객의 소비 점수
Family_Size	가족 구성원 수
Var_1	익명화 된 범주
Segmentation	고객 분류 범주

데이터 분석 과정

#1 Segmentation 별로 Feature들이 어떤 분포를 보이고 있는가?



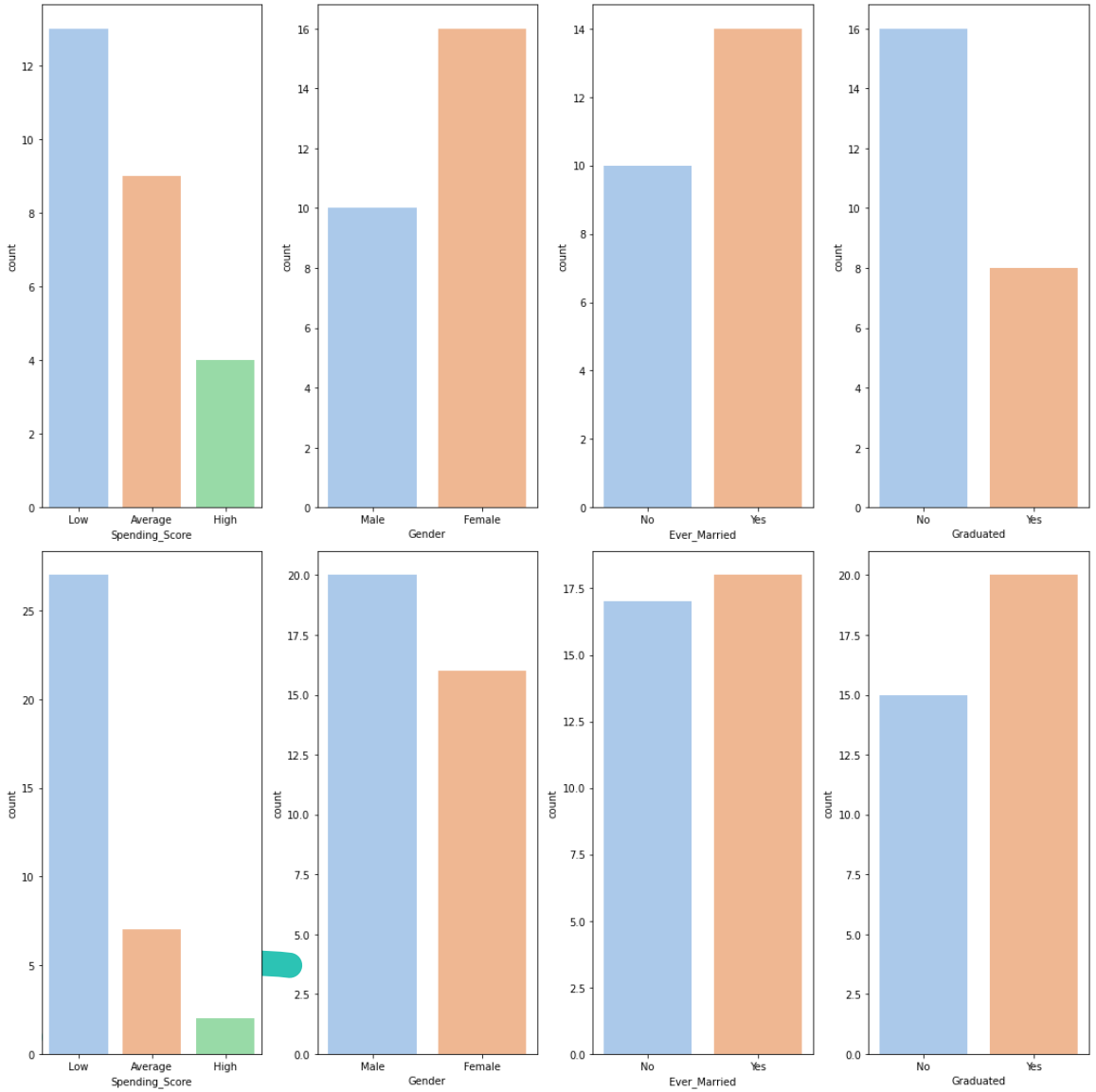
- 우선 target인 Segmentation을 기준으로 feature들의 분포를 시각화 하여 어떤 관계가 있는지 파악한다.
- A,B,C는 전체적으로 비슷한 경향을 보이나,
  D는 눈에 띄게 다른 경향을 보인다.



[각 Profession의 비율과 그 안에서 각 Segmentation이 차지하는 비중 그래프]

- 전체 Profession 중에 Artist가 약 25.5%로 가장 큰 비중을 차지한다.
- 모든 Profession에서 Segmentation D가 B, C보다 큰 비중을 차지한다.
- Segmentation C는 모든 Profession을 통틀어 가장 적다.

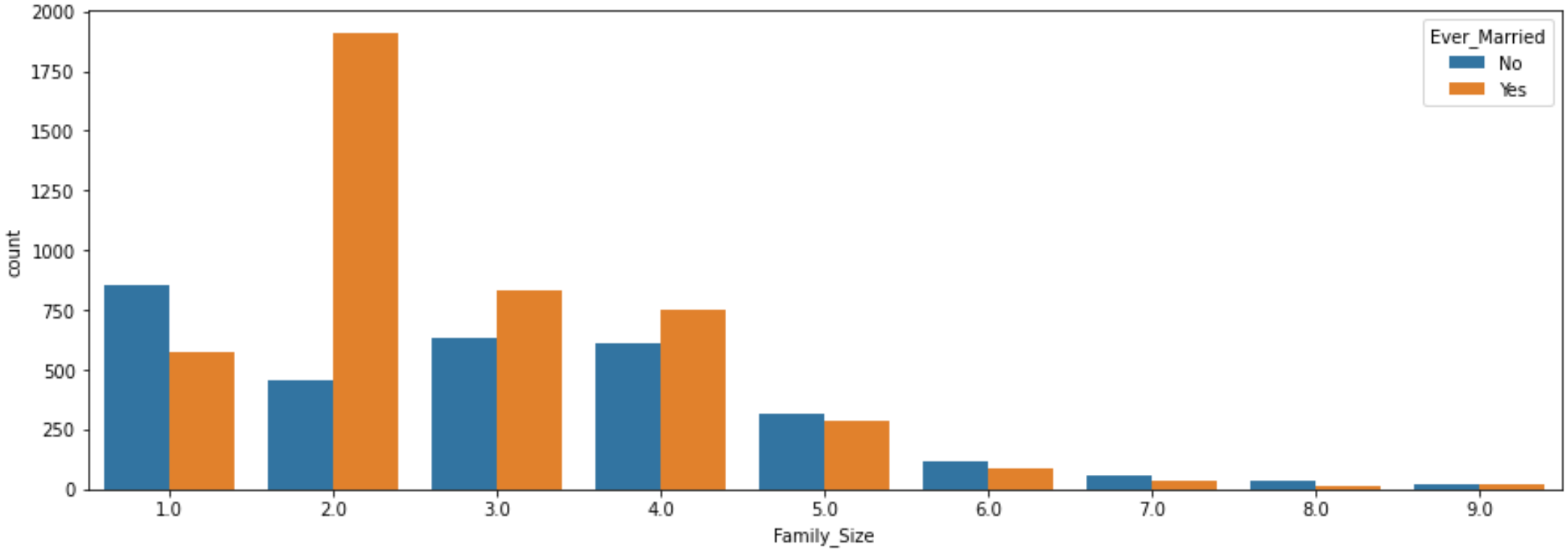
#2 Profession이 NaN일 때 Spending_Score와 Work_Experience의 관계



- 위: Profession이 NaN일 때 Work_Experience가 NaN인 경우
- 아래: Profession이 NaN일 때 Work_Experience가 0인 경우

→ 두 경우 모두 Spending_Score에서 Low가 압도적으로 큰 비중을 차지하는 것을 토대로 Profession이 NaN인 사람을 '무직'으로 간주한다.

#3 Ever_married와 family_size의 관계

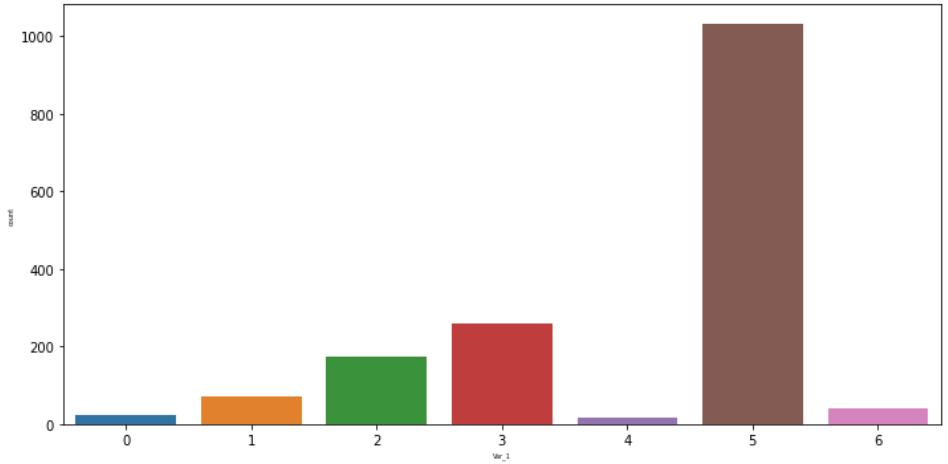


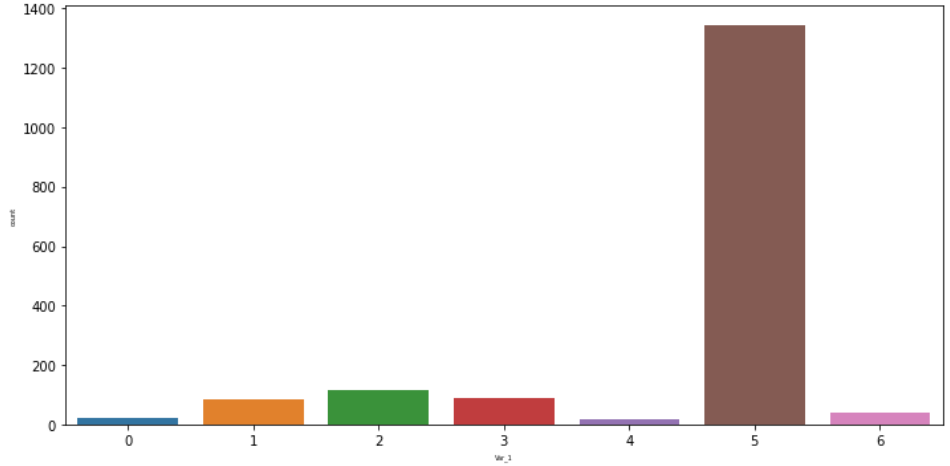
- 1인 가구에서 결혼 비경험자의 수가 유의미한 차이로 더 많다.
- 2인-4인 가구에서 결혼 경험자의 수가 유의미한 차이로 더 많다(특히 2인 가구)
- 나머지 분포는 큰 차이를 보이지 않는다.

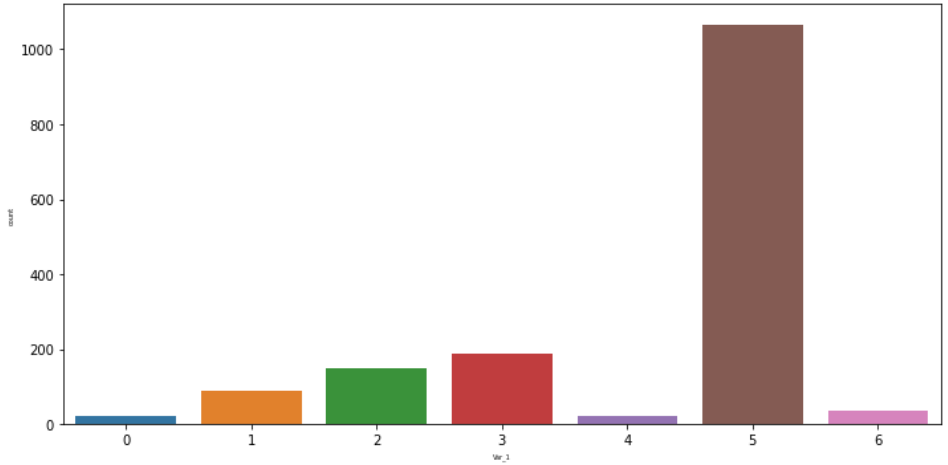
→ Family_Size가 4 이하인 경우엔 Ever_Married에 따른 차이가 존재 하나
5 이상인 경우엔 Ever_Married의 영향을 받지 않는 것으로 보인다.

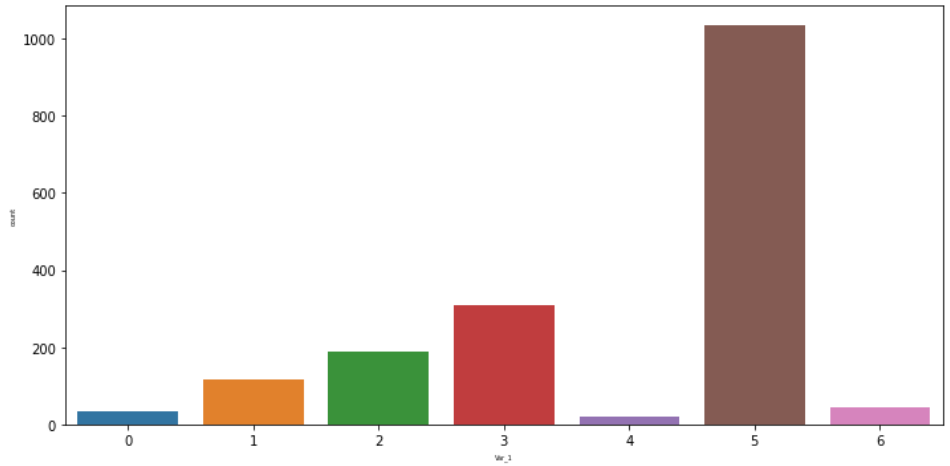
#4 Var_1의 의미

Var_1의 각 카테고리 별 빈도 그래프
→ cat_6이 압도적인 수를 차지한다. 그러나 그래프만으로는 var_1의 의미를 추정할 수는 없었다.

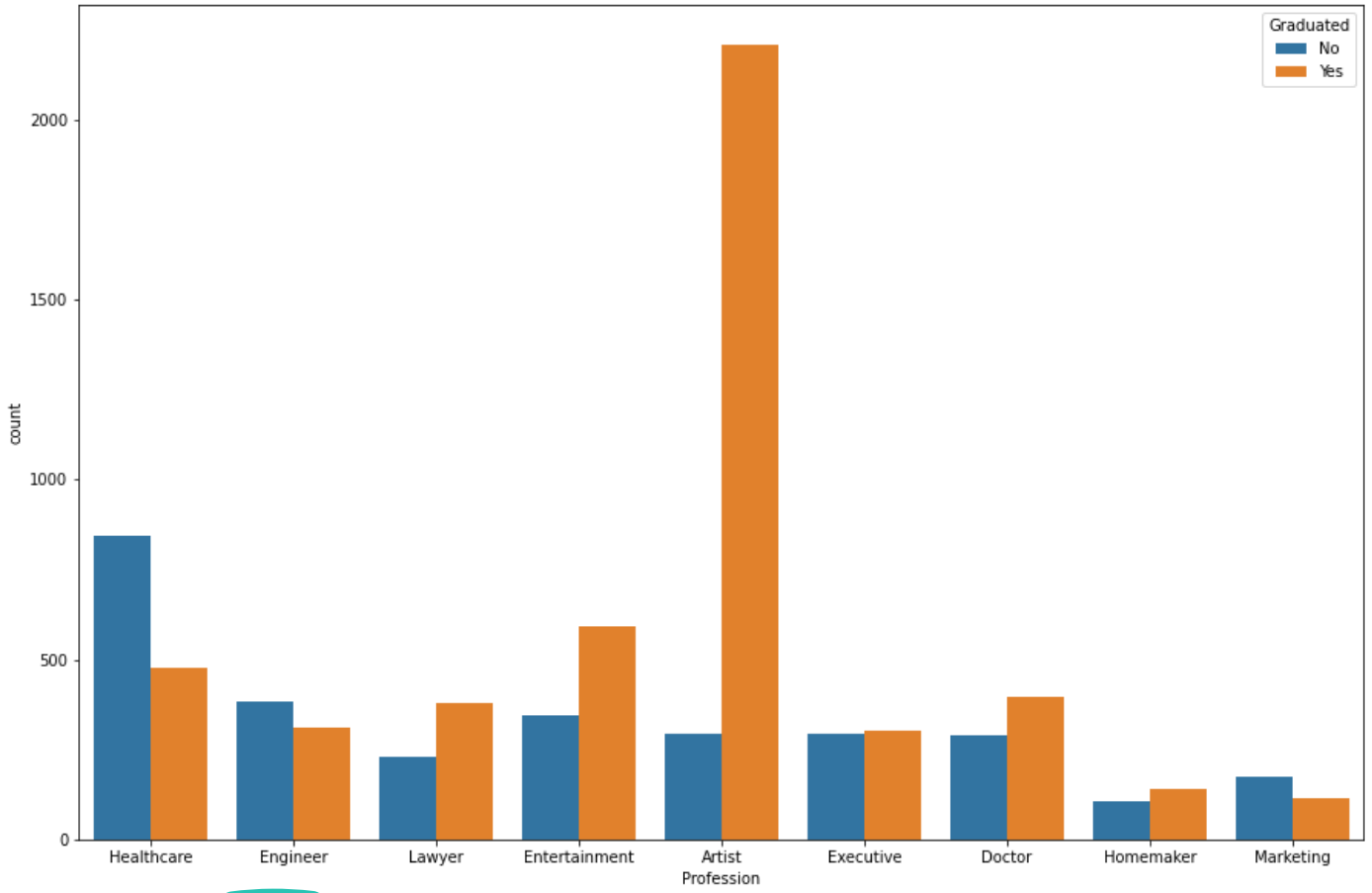




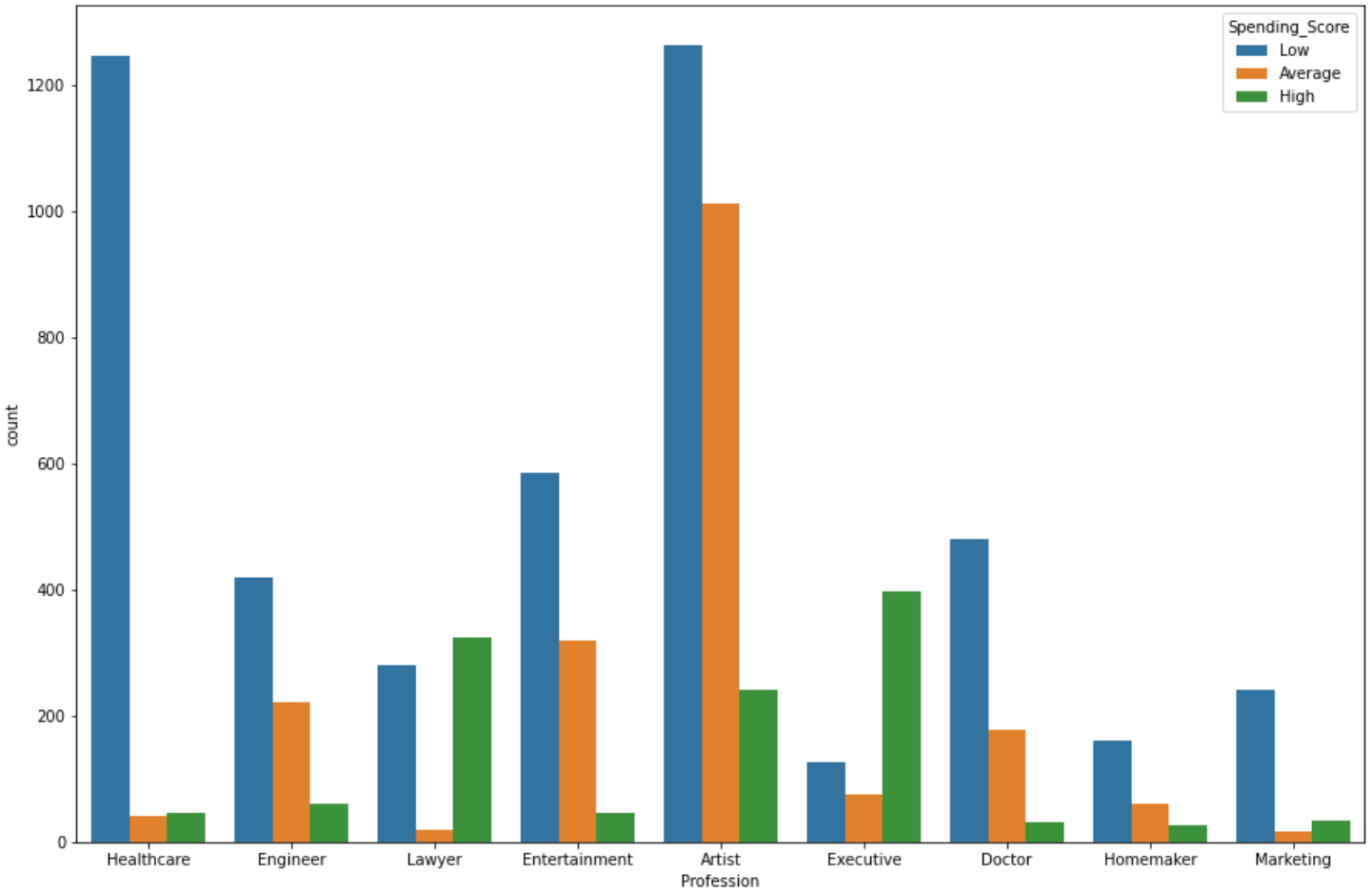




#5 Graduated, Profession 과 Spending_Score의 관계

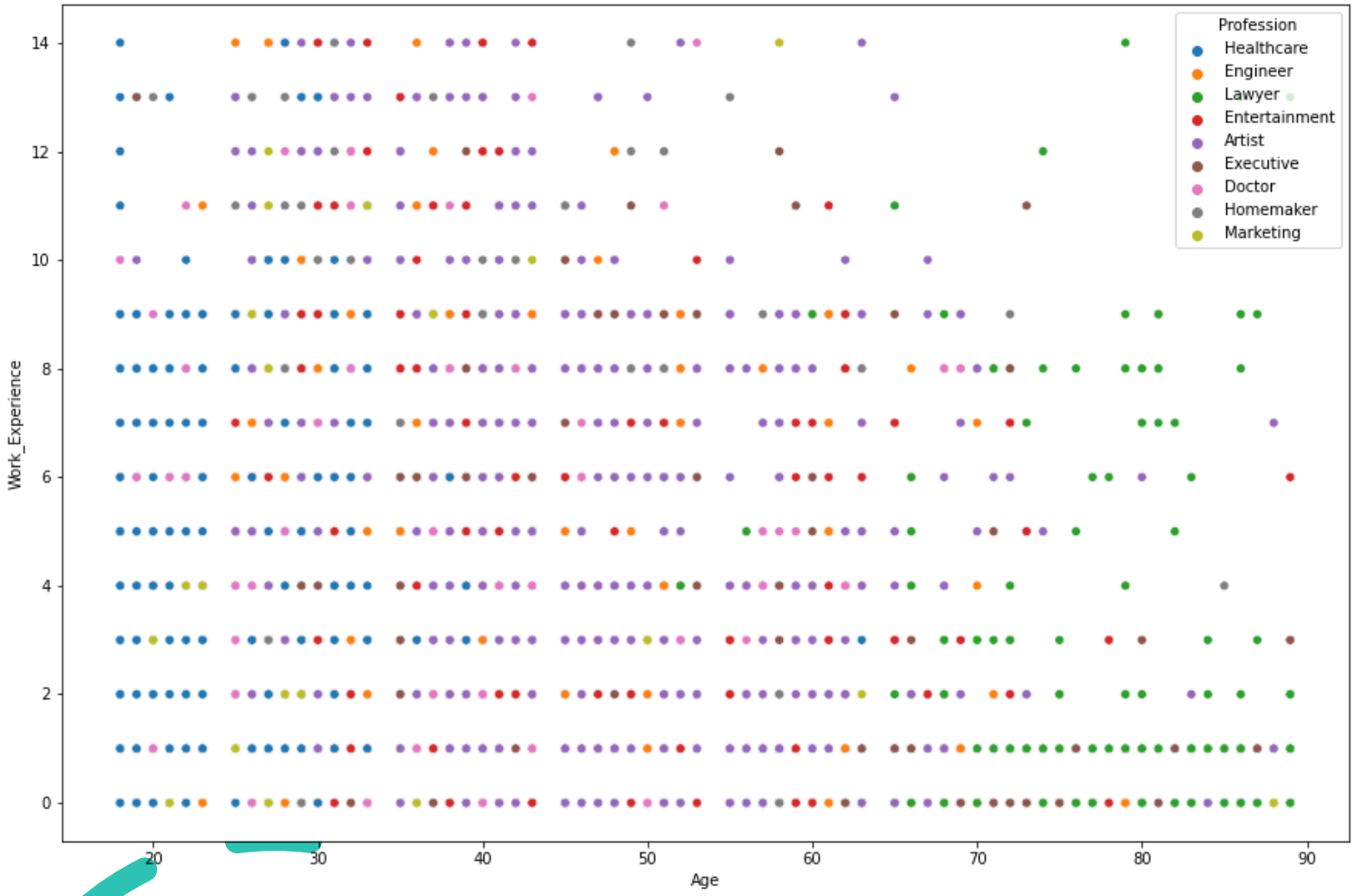


- Healthcare, Engineer, Marketing을 제외하고는 모든 직업에서 졸업자의 수가 미졸업자의 수 보다 많다.
- 특히 Artist는 졸업자와 미졸업자의 수 차이가 아주 크게 나타난다.
- Healthcare는 미졸업자가 졸업자 보다 많음 Profession들 중에서도 가장 미졸업자의 비율이 크게 나타난다.



- Lawyer와 Executive를 제외한 모든 Profession에서 Spending_Score가 Low인 경우가 가장 많다.

전처리 – 이상치



- 시각화와 데이터 분석 과정을 거치며

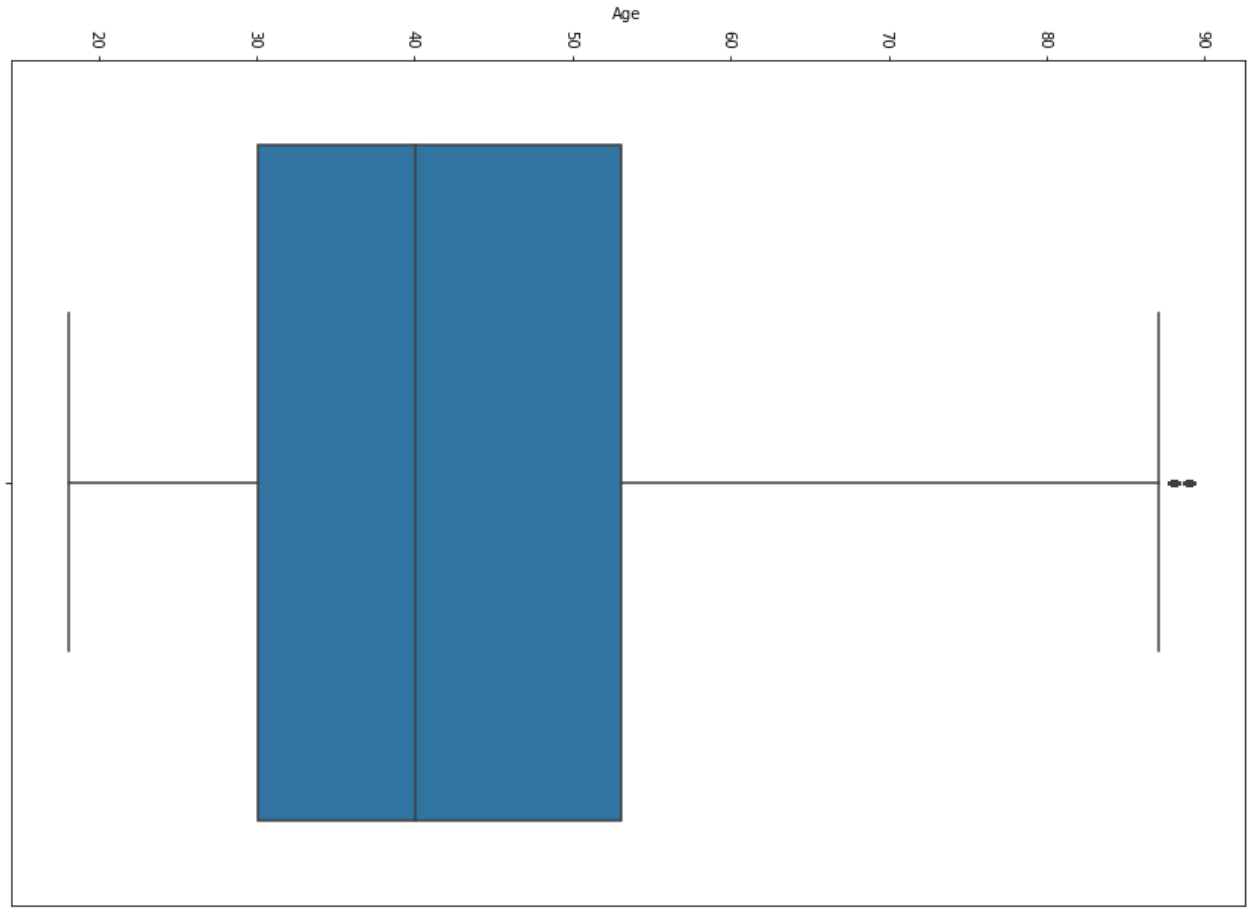
1) 10대인 응답자가 Work_Experience가 10년 이상인 경우가 존재

2) Lawyer인 응답자가 60대 이상에만 분포

3) 80세 이상인 응답자가 Work_Experience가 2년 미만인 경우가 다수 존재

등의 어색한 특징들을 발견했다.

이상치 제거



- Age에서 상식 선을 벗어난 나이를 이상치로 간주하고 제거한다.(Boxplot 참고)

→ 88세, 89세 제거

전처리 – 결측치 제거

- 결측치 처리 방법
  : 결측치를 예측하는 서브 모델을 만들고,
  이를 통해 예측한 값으로 결측치를 대체한다.

- 결측치 처리 과정
  1) 결측치가 있는 feature 중 하나를 target으로 설정
  2) target인 feature의 결측치가 모두 밑으로 가도록 데이터 정렬
  3) target의 결측치가 y_test에 들어가도록 데이터를 4등분
  4) X_train, y_train을 통해 데이터를 학습하고 이를 X_test, y_test에 적용

- 해당 과정을 통해 결측치를 대체한 feature
  : Ever_Married, Graduated, Family_Size, Var_1

- Work_Experience와 Profession은 앞서 #2에서 언급한 내용을 바탕으로
  각각 0과 'JobLess'로 결측치를 대체한다.

Encoding

- 순서형 변수 Spending_Score

→ Label Encoding

- 범주형 변수 Gender, Ever_Married, Graduated, Profession, Var_1

→ One-Hot Encoding

AutoML 개요

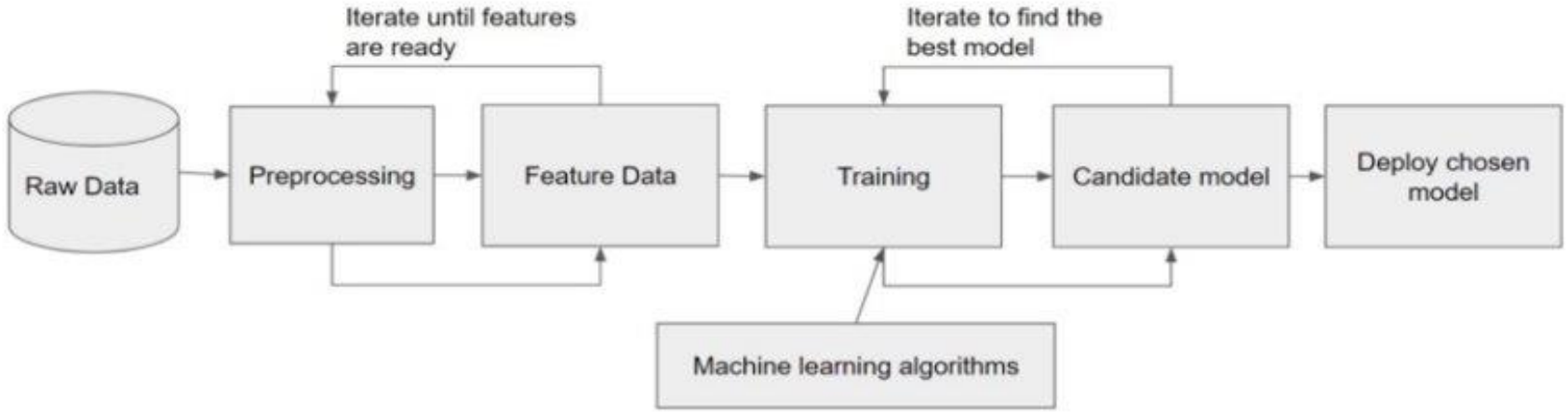
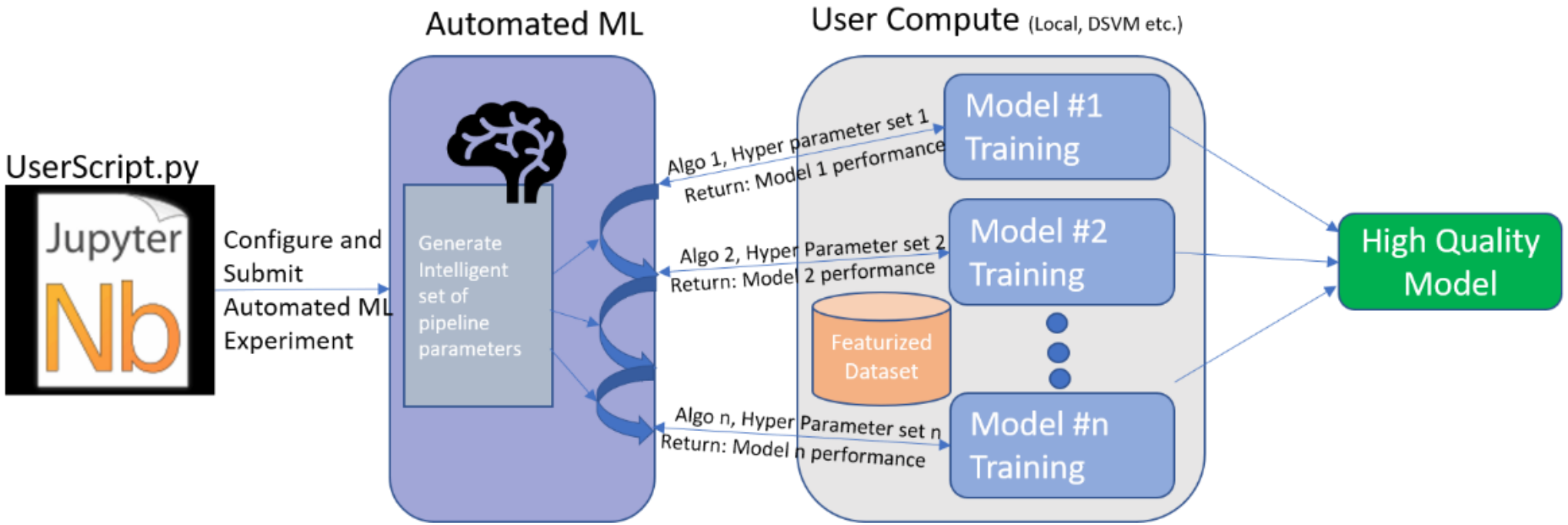


Figure 1. Machine learning process

Step1. 학습 하이퍼파라미터 최적화	Step2. 의미 있는 특성 추출	Step3. 모델링 프로세스 설계
Hyperparameter Optimization 학습률(learning rate), 배치 크기(mini-batch size) 등 학습에 큰 영향을 주는 hyperparameter들을 학습을 통해 추정하는 것을 의미합니다.	Feature Learning Extraction 혹은 Feature Engineering을 의미하는 것으로, 학습 모델에 입력을 그대로 사용하지 않고, 학습을 통하여 유의미한 feature(특징)를 추출해서 입력으로 사용하는 방법 입니다.	Architecture Search 모델링 전체 프로세스를 사람이 직접 하나하나 설계하는 대신에 학습을 통해 최적의 아키텍처를 설계하는 방법을 의미합니다.


Automated ML
User Compute (Local, DSVM etc.)
UserScript.py
Jupyter
Nb
Configure and
Submit
Automated ML
Experiment
Generate
Intelligent
set of
pipeline
parameters
Algo 1, Hyper parameter set 1
Return: Model 1 performance
Algo 2, Hyper Parameter set 2
Return: Model 2 performance
Algo n, Hyper Parameter set n
Return: Model n performance
Model #1
Training
Model #2
Training
Model #n
Training
Featurized
Dataset
High Quality
Model

AutoML 적용 : 상위 5개 모델 선정

	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC	TT (Sec)
gbc	Gradient Boosting Classifier	0.5402	0.7910	0.5314	0.5346	0.5359	0.3845	0.3852	2.459
catboost	CatBoost Classifier	0.5266	0.7811	0.5183	0.5202	0.5221	0.3664	0.3671	6.021
lightgbm	Light Gradient Boosting Machine	0.5265	0.7837	0.5173	0.5184	0.5209	0.3660	0.3668	0.443
lr	Logistic Regression	0.5152	0.7700	0.5053	0.5058	0.5042	0.3508	0.3537	3.073
xgboost	Extreme Gradient Boosting	0.5137	0.7738	0.5054	0.5078	0.5096	0.3492	0.3498	7.470
ada	Ada Boost Classifier	0.5134	0.7608	0.5041	0.5067	0.5070	0.3488	0.3502	0.291
ridge	Ridge Classifier	0.5051	0.0000	0.4933	0.4952	0.4733	0.3363	0.3462	0.024
lda	Linear Discriminant Analysis	0.5045	0.7675	0.4971	0.5075	0.5009	0.3379	0.3405	0.045
rf	Random Forest Classifier	0.4988	0.7546	0.4908	0.4954	0.4963	0.3297	0.3301	0.996
et	Extra Trees Classifier	0.4776	0.7248	0.4699	0.4764	0.4764	0.3016	0.3019	0.982
nb	Naive Bayes	0.4752	0.7337	0.4661	0.4673	0.4521	0.2981	0.3076	0.029
knn	K Neighbors Classifier	0.4607	0.7092	0.4553	0.4717	0.4640	0.2809	0.2817	0.149
dt	Decision Tree Classifier	0.4394	0.6331	0.4324	0.4408	0.4394	0.2514	0.2517	0.041
svm	SVM - Linear Kernel	0.4193	0.0000	0.4116	0.4610	0.3600	0.2238	0.2714	0.325
qda	Quadratic Discriminant Analysis	0.3045	0.5349	0.2977	0.2979	0.2826	0.0686	0.0705	0.038
dummy	Dummy Classifier	0.2811	0.5000	0.2500	0.0790	0.1233	0.0000	0.0000	0.018

AutoML 적용 : Hyperparameter Tuning

```
gbc = create_model("gbc", fold = 10)
tuned_gbc = tune_model(gbc, fold = 10, optimize = "Accuracy")
```

	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
0	0.5062	0.7762	0.4890	0.4828	0.4746	0.3338	0.3443
1	0.5125	0.7932	0.4978	0.4927	0.4848	0.3430	0.3524
2	0.5469	0.7885	0.5332	0.5354	0.5252	0.3899	0.3978
3	0.5438	0.8041	0.5295	0.5273	0.5214	0.3858	0.3935
4	0.5375	0.8174	0.5220	0.5368	0.5114	0.3755	0.3899
5	0.5469	0.7988	0.5339	0.5378	0.5259	0.3897	0.3986
6	0.5328	0.7979	0.5167	0.5176	0.5017	0.3704	0.3809
7	0.5258	0.7870	0.5117	0.5160	0.5064	0.3616	0.3684
8	0.4930	0.7651	0.4766	0.4648	0.4616	0.3170	0.3249
9	0.5493	0.8106	0.5338	0.5586	0.5192	0.3915	0.4065
Mean	0.5295	0.7939	0.5144	0.5170	0.5032	0.3658	0.3757
SD	0.0186	0.0148	0.0194	0.0274	0.0213	0.0250	0.0258

```
catboost = create_model("catboost", fold = 10)
tuned_cat = tune_model(catboost, fold = 10, optimize = "Accuracy")
```

```
lightgbm = create_model("lightgbm", fold = 10)
tuned_lgb = tune_model(lightgbm, fold = 10, optimize = "Accuracy")
```

```
lr = create_model("lr", fold = 10)
tuned_lr = tune_model(lr, fold = 10, optimize = "Accuracy")
```

```
ada = create_model("ada", fold = 10)
tuned_ada = tune_model(ada, fold = 10, optimize = "Accuracy")
```

AutoML 적용 : 최종 Hyperparameter

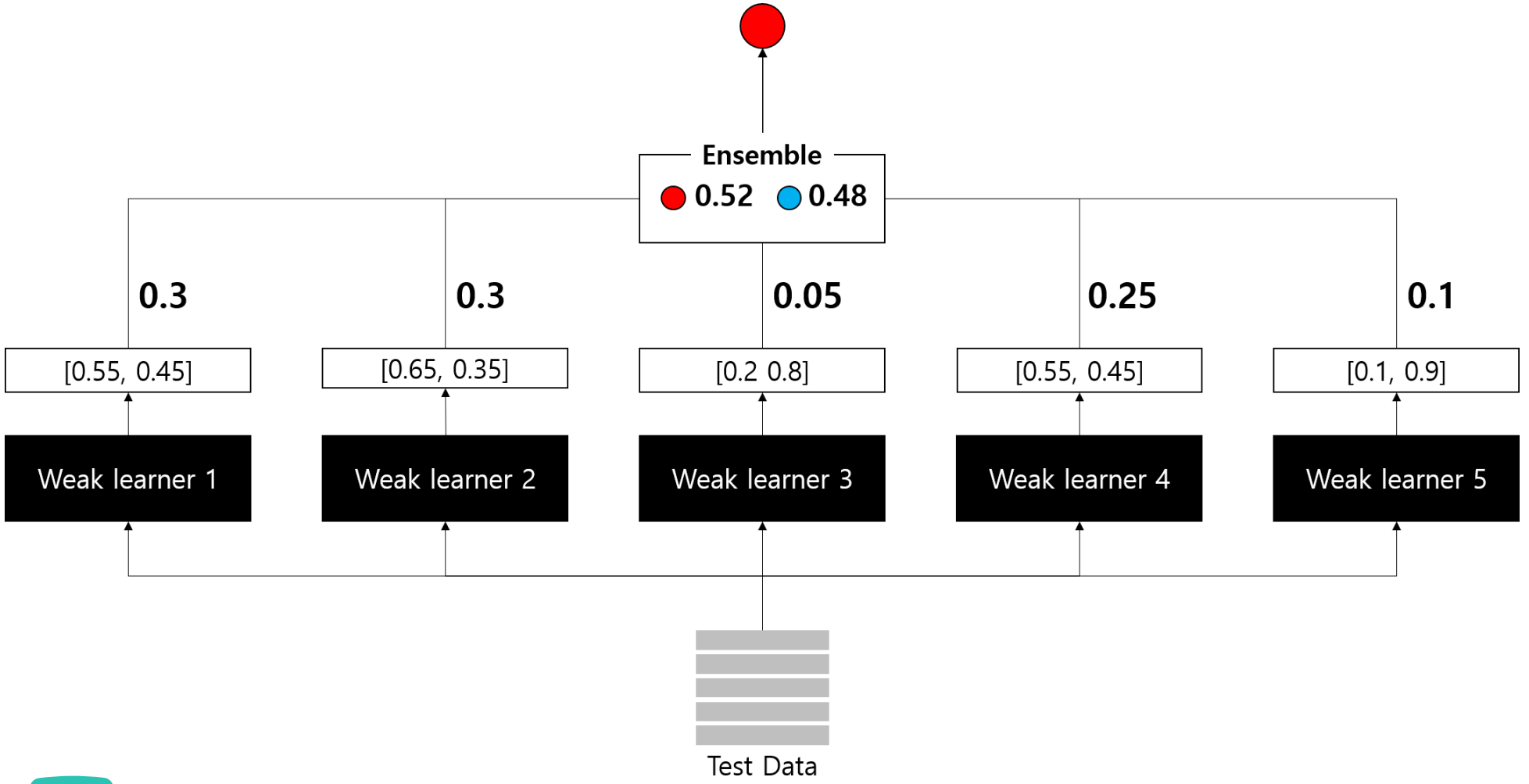
```
gbc = GradientBoostingClassifier(
  ccp_alpha = 0.0,
  criterion = 'friedman_mse',
  init = None,
  learning_rate = 0.001,
  loss = 'deviance',
  max_depth = 8,
  max_features = 'sqrt',
  max_leaf_nodes = None,
  min_impurity_decrease = 0.001,
  min_impurity_split = None,
  min_samples_leaf = 5,
  min_samples_split = 7,
  min_weight_fraction_leaf = 0.0,
  n_estimators = 250,
  n_iter_no_change = None,
  presort = 'deprecated',
  random_state = 8232,
  subsample = 0.6,
  tol = 0.0001,
  validation_fraction = 0.1,
  verbose = 0,
  warm_start = False
)
```

```
lgb = LGBMClassifier(
  boosting_type = 'gbdt',
  colsample_bytree = 1.0,
  importance_type = 'split',
  dlearning_rated = 0.5,
  dmax_depthd = -1,
  dmin_child_samplesd = 86,
  dmin_child_weightd = 0.001,
  dmin_split_gaind = 0.4,
  dn_estimatorsd = 130,
  dnum_leavesd = 90,
  dobjectived = None,
  dreg_alphad = 0.15,
  dreg_lambdad = 0.001,
  dsilentd = 'dwarnd',
  dsubsampled = 1.0,
  dsubsample_for_bind = 200000,
  dsubsample_freq = 0,
  feature_fraction = 0.9,
  bagging_freq = 7,
  bagging_fraction  = 0.8
)
```

```
cat = CatBoostClassifier(
  depth = 2,
  l2_leaf_reg = 2,
  border_count = 254,
  verbose = False,
  random_strength = 0.2,
  task_type = 'CPU',
  n_estimators = 300,
  random_state = 8232,
  eta = 0.1
)
```

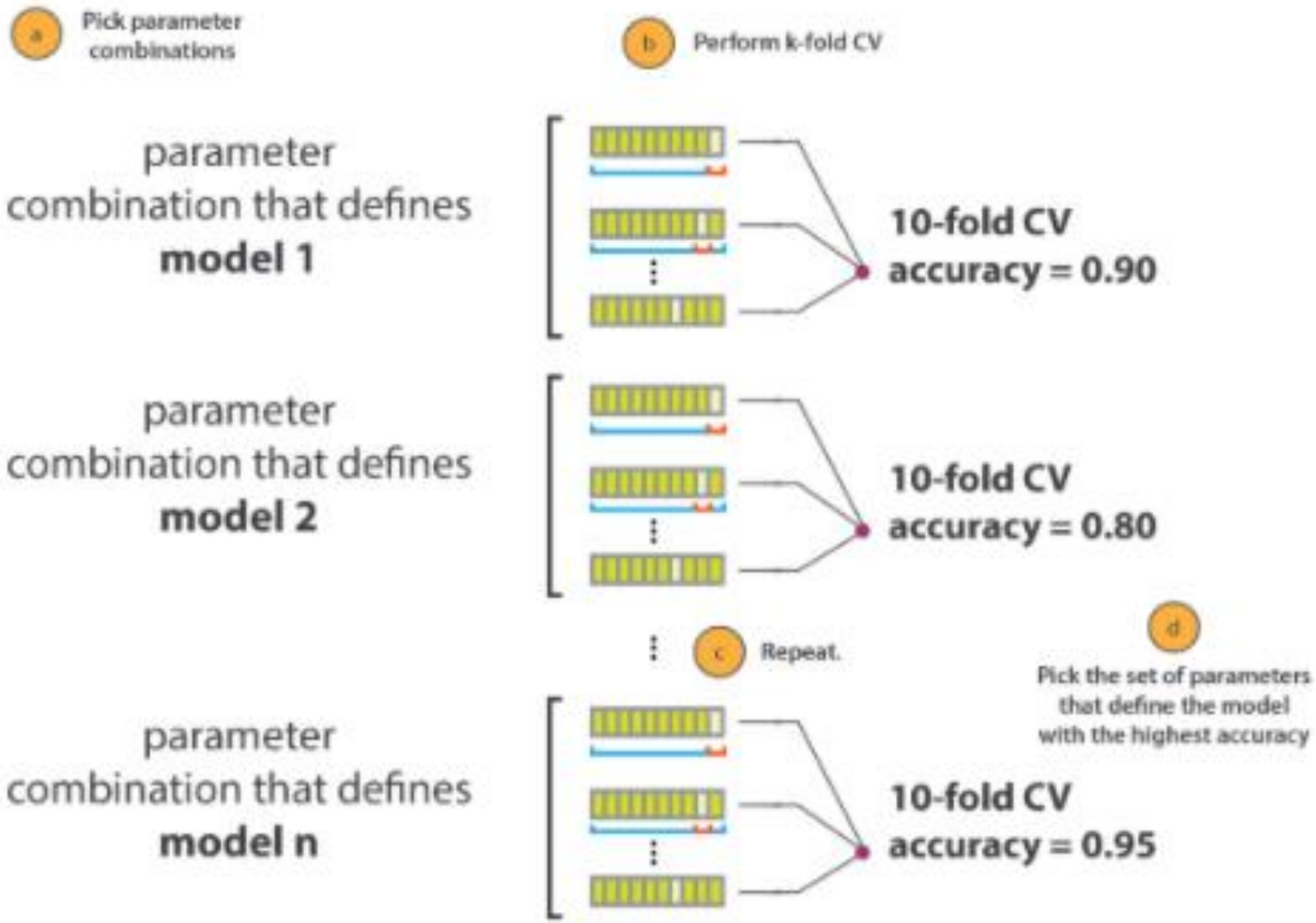
```
lr = LogisticRegression(
  C = 0.909,
  class_weight = 'balanced',
  dual = False,
  fit_intercept = True,
  intercept_scaling = 1,
  l1_ratio = None,
  max_iter = 1000,
  multi_class = 'auto',
  n_jobs = None,
  penalty = 'l2',
  random_state = 8232,
  solver = 'lbfgs',
  tol = 0.0001,
  verbose = 0,
  warm_start = False
)
```

```
ada = AdaBoostClassifier(
  algorithm = 'SAMME.R',
  base_estimator = None,
  learning_rate = 0.2,
  n_estimators = 290,
  random_state = 8232
)
```

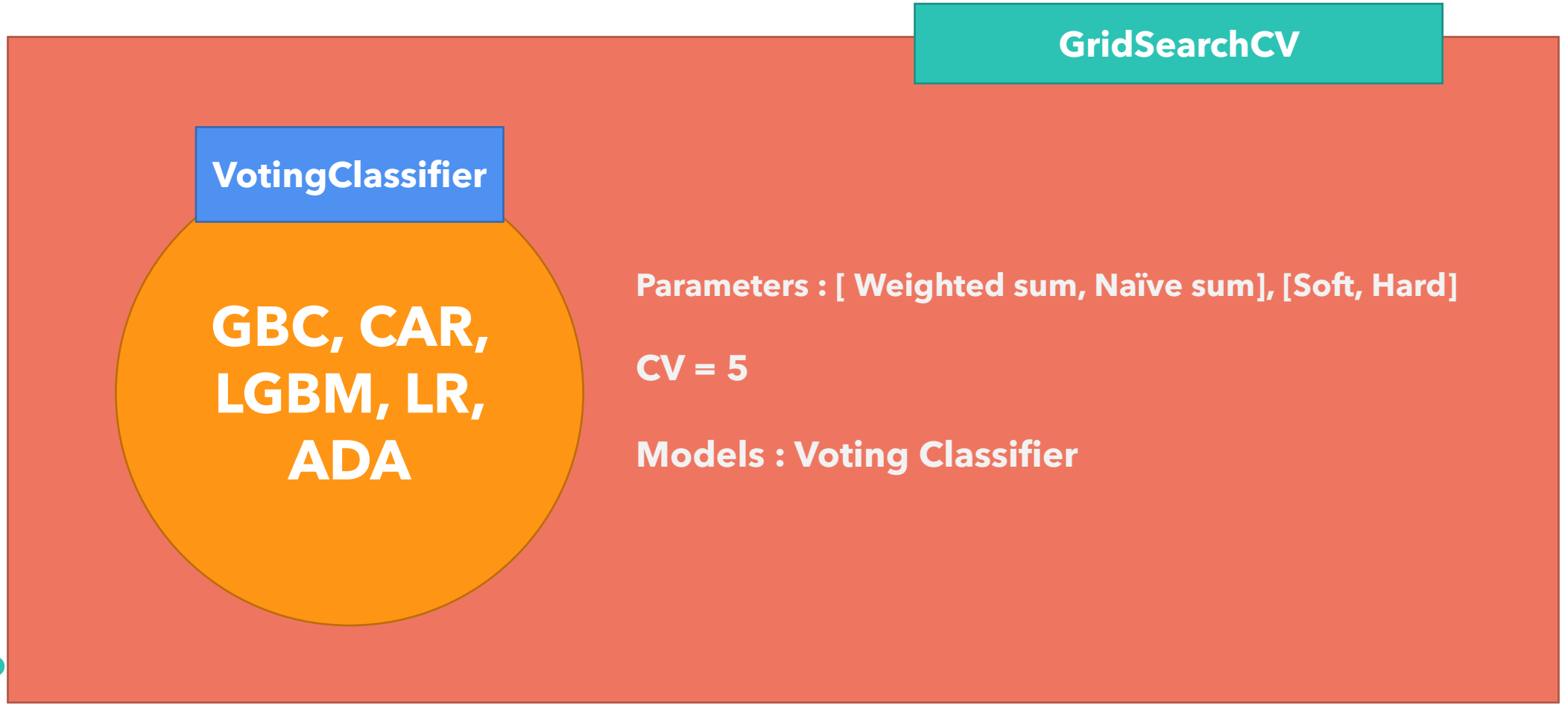

Ensemble
0.52
0.48
0.3
0.3
0.05
0.25
0.1
[0.55, 0.45]
[0.65, 0.35]
[0.2 0.8]
[0.55, 0.45]
[0.1, 0.9]
Weak learner 1
Weak learner 2
Weak learner 3
Weak learner 4
Weak learner 5
Test Data

Weighted Voting

```
from sklearn.ensemble import VotingClassifier

#각 모델의 accuracy 따른 가중치 적용
models = {
    'gbc': gbc.fit(X_train, y_train),
    'cat': cat.fit(X_train, y_train),
     'lgb': lgb.fit(X_train,y_train),
     'lr': lr.fit(X_train,y_train),
     'ada' : ada.fit(X_train,y_train),
}

# relative weights
model_scores = {
    name: accuracy_score(
       y_test,
       model.predict(X_test),
       )
    for name, model in models.items()
}
total_score = sum(model_scores.values())

# combine the parts
voting_ensemble = VotingClassifier(estimators = [("gbc", gbc), ("cat", cat),
                                                 ('lgb',lgb),('lr',lr),('ada',ada)],
                                   weights = [model_scores[name] / total_score for name in models.keys()],
                                   voting = 'soft')
```


a
Pick parameter combinations
b
Perform k-fold CV
parameter combination that defines model 1
10-fold CV accuracy = 0.90
parameter combination that defines model 2
10-fold CV accuracy = 0.80
c
Repeat.
d
Pick the set of parameters that define the model with the highest accuracy
parameter combination that defines model n
10-fold CV accuracy = 0.95

Weighted Soft Voting + GridSearchCV



Result

Soft + Naive	Soft + Weighted	Hard + Naive	Hard + Weighted
54.05	54.06	53.47	53.48

한계점과 의의

- 한계점

: 실질적인 전처리의 부재

→ feature의 해석이 결과에 반영되지 않음

- 의의

: 결측치 modeling

: weighted-sum soft voting

→ 결과 향상

<Kaggle Bronze Medal + 1.05%>

감사합니다